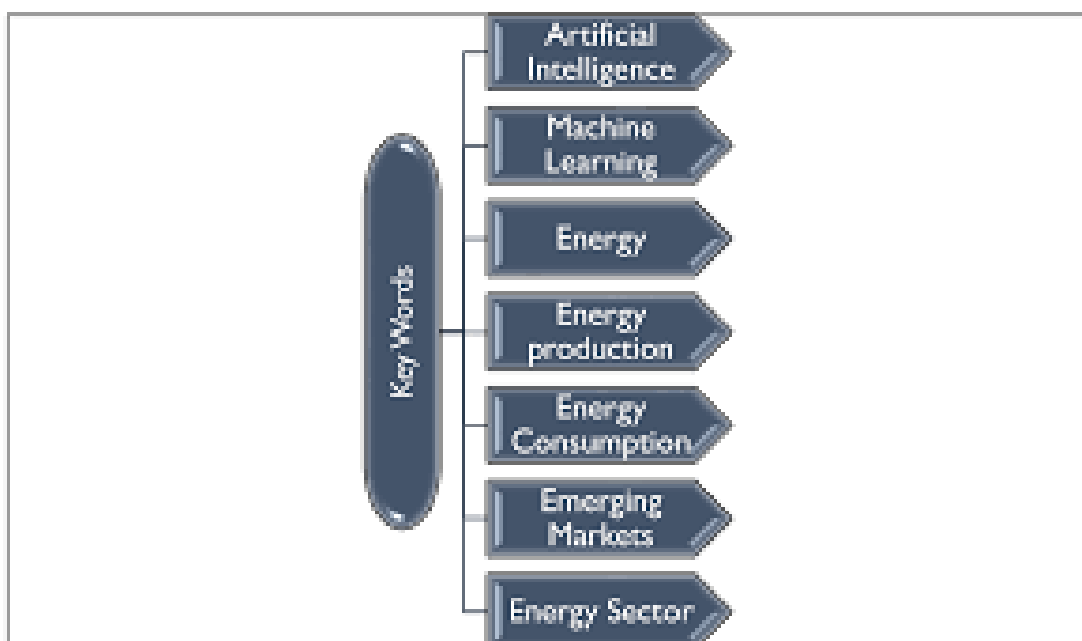


MEASURE ENERGY CONSUMPTION USING AI

PHASE 5: PROJECT DOCUMENTATION & SUBMISSION



INTRODUCTION



Measuring energy consumption in detail requires a systematic approach and the use of appropriate tools and devices. Here are the key steps and methods for measuring energy consumption:

- Identify the Energy Sources:** Determine the energy sources you want to measure. This could include electricity, natural gas, heating oil, or other forms of energy.
- Select Measurement Units:** Choose the appropriate units for measuring energy consumption. Common units include kilowatt-hours (kWh) for electricity, therms for natural gas, and gallons for heating oil.
- Install Energy Meters:** Install energy meters or monitoring devices for each energy source you want to measure. For electricity, this may involve using smart meters provided by utility companies. For other sources, you may need separate energy meters.
- Record Readings:** Regularly record readings from the energy meters. The frequency of readings may vary depending on your needs, but daily, weekly, or monthly readings are common.
- Analyze Consumption Patterns:** Use software or tools to analyze the consumption data. Look for patterns and trends in energy usage. Some utility companies provide online platforms for tracking and analyzing energy consumption.
- Monitor Real-time Data:** For a more detailed understanding, consider real-time monitoring systems that provide continuous data on energy usage. These systems can offer insights into peak usage times and help identify energy-saving opportunities.
- Weather Normalisation:** Consider normalising energy consumption data for weather variations. Heating and cooling energy consumption can be significantly affected by temperature fluctuations. Normalisation helps to compare energy use across different time periods accurately.
- Energy Audits:** Conduct energy audits, either self-assessment or with the help of professionals, to identify areas of energy waste and opportunities for improvement. This can involve inspecting insulation, HVAC systems, lighting, and appliances.
- Appliance-level Monitoring:** For a detailed breakdown of energy usage, you can install specialised energy monitoring devices on individual appliances and equipment. These devices provide real-time data on how much energy each appliance consumes.
- Energy Management Systems:** In commercial and industrial settings, energy management systems (EMS) are used to monitor and control energy usage. These systems integrate various sensors and controls to optimise energy consumption.
- Data Logging:** Maintain a detailed record of all your energy consumption data. This historical data is valuable for tracking long-term trends and making informed decisions about energy efficiency improvements.
- Set Energy Goals:** Based on your analysis, set energy reduction goals and implement energy-efficient measures to achieve them. Regularly monitor progress toward these goals.
- Educate and Involve Stakeholders:** Educate building occupants or employees about energy conservation practices and encourage their involvement in reducing energy consumption.
- Review and Adjust:** Continuously review and adjust your energy management strategies based on the data and insights gathered. This iterative process can lead to ongoing energy savings.

Accurate and detailed energy consumption measurement is essential for

optimising energy use, reducing costs, and minimising environmental impact. Depending on the scale and complexity of your energy needs, you may need to consult with energy professionals or use advanced energy management systems to achieve the highest level of detail and efficiency.

1. PROBLEM STATEMENT & DESIGN THINKING

This project will bring to the light to one possible solution of an IEMS using Machine Learning techniques, this case is related to optimise and rationalise energy consumed with one specific department (Local refrigeration [department responsible of generating cold for cooling systems and production need]) in a company blueprint during the blackout periods when the principal power resources are down and alternative resources are very limited.

2. PROBLEM DEFINITION

The project aims to come up with a solution to well manage the energy usage inside an industrial environment where there is a high requirement for energy availability to keep the production lines running especially for the critical industries that are essential for human life like medicines and food manufacturing. We already know that companies while trying to fill the market demand they can consume a lot of energy that sometimes don't need, and because companies don't have the mechanisms to control their needs in term of energy especially on blackout periods, therefore, this ends up with a down time for them when the power source are fluctuating or unavailable for whatever reason. The IEMS systems can play an important role in this area of application to well streamlining the management of the energy flow during these critical situations.

3. DESIGN THINKING

a) Data source:

In measuring energy consumption prediction, the choice of data sources is critical for driving the transition to cleaner and more sustainable energy sources. These trends collectively enhance energy efficiency, reduce costs, and promote environmental sustainability in the energy sector.

b) Data preprocessing:

Data preprocessing is a crucial step in measuring energy consumption prediction as it ensures that the data used for modelling is clean, relevant, and suitable for the predictive task.

c) Feature Engineering:

Create additional features, such as lag features (past values) or rolling statistics. These features can provide more information for forecasting.

4.MODEL SELECTION

a) Time Series Forecasting:

Choose a time series forecasting model, such as ARIMA, Exponential Smoothing, or LSTM. Split the data into training and testing sets. Train the selected model on the training data and evaluate it on the test data.

b) AI Models:

TensorFlow: TensorFlow applications can be run conveniently on your local machine, cloud, android and iOS devices. As it is built on a deployable scale, it runs on CPU and GPU.

PyTorch: PyTorch is similar to TensorFlow in terms of the nature of the projects chosen. However, when the priority is for faster development, PyTorch is the better choice. TensorFlow is gone in case the project involves larger and more complex projects.

Scikit Learn: Scikit-learn is a widely praised Artificial Intelligence tool that simplifies the complexities of machine learning tasks. It boasts an intuitive and user-friendly interface that caters to learners across different proficiency levels. Scikit-learn equips users with the means to construct and deploy machine learning models effortlessly.

Natural Language Processing: It focuses on developing natural interactions between humans and computers. Specialised software helps machines process human language, create understandable words, and interact with humans through language.

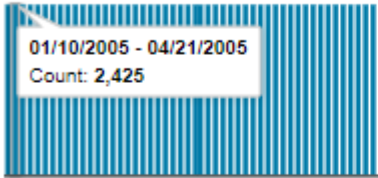
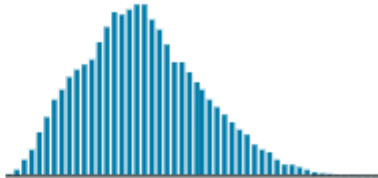
Figstack: Figstack provides a comprehensive set of artificial intelligence tools designed to support developers in comprehending and documenting code more effectively. Its diverse array of features is geared towards simplifying the coding process, featuring a natural language interpreter capable of understanding code in nearly any programming language.

I. DATASET

The dataset used for measuring energy consumption prediction typically consists of historical data related to hourly energy consumption and various relevant factors that can influence the economization of its consumption.

1. DATASET LINK:

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

Datetime	# AEP_MW
Date	Megawatt Energy Consumption
 <p>01/10/2005 - 04/21/2005 Count: 2,425</p> <p>10Oct04 3Aug18</p>	 <p>9.58k 25.7k</p>
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
2004-12-31 06:00:00	13038.0
2004-12-31 07:00:00	13692.0
2004-12-31 08:00:00	14297.0
2004-12-31 09:00:00	14719.0

2004-12-31 10:00:00	14941.0
2004-12-31 11:00:00	15184.0
2004-12-31 12:00:00	15009.0
2004-12-31 13:00:00	14808.0
2004-12-31 14:00:00	14522.0
2004-12-31 15:00:00	14349.0
2004-12-31 16:00:00	14107.0
2004-12-31 17:00:00	14410.0
2004-12-31 18:00:00	15174.0
2004-12-31 19:00:00	15261.0
2004-12-31 20:00:00	14774.0
2004-12-31 21:00:00	14363.0

II. DESIGN INTO INNOVATION

A model for measuring energy consumption should encompass several essential features to be effective. Here's how such a model should be designed:

Data Collection: The model should gather data from various sources, including smart meters, IoT devices, sensors, and historical records. Data accuracy is crucial for reliable measurement.

Real-Time Monitoring: It should provide real-time monitoring of energy consumption, enabling immediate response to changes and anomalies.

Machine Learning Algorithms: Incorporate machine learning algorithms to analyse and process data, allowing for predictive analytics and pattern recognition.

Customization: The model should be adaptable to different environments, industries, and energy systems, with the ability to fine-tune parameters to match specific needs.

Anomaly Detection: Implement anomaly detection mechanisms to identify irregular patterns or sudden spikes in energy usage, which could indicate inefficiencies or issues.

Predictive Analytics: Utilise historical data and machine learning to predict future energy consumption, aiding in demand forecasting and resource planning.

Visualization: Provide user-friendly dashboards and data visualization tools to present energy consumption data in a clear and comprehensible manner.

Recommendations: Offer actionable insights and recommendations for optimizing energy usage and reducing costs, including suggestions for adjusting equipment settings, improving insulation, or scheduling operations during off-peak hours.

Integration: Ensure seamless integration with existing energy management systems, building management systems, and other related platforms to streamline operations.

Energy Efficiency Metrics: Calculate key performance indicators like energy intensity, efficiency ratios, and carbon emissions, allowing organizations to track their progress toward sustainability goals.

Cloud-Based or On-Premises: The model can be hosted on the cloud or on-premises, depending on the organisation's preferences and requirements.

Security: Prioritise data security and privacy to safeguard sensitive energy consumption information from unauthorised access or breaches.

Reporting: Generate comprehensive reports and alerts to keep stakeholders informed about energy consumption trends and deviations from set targets.

Scalability: Ensure the model can handle growing datasets and the addition of new sensors or devices as an organization's needs evolve.

Smart Meter Data Analysis: AI can analyze data from smart meters to provide real-time insights into energy consumption patterns. It can detect anomalies, forecast usage, and help consumers make informed decisions.

Predictive Maintenance: AI can predict when equipment like HVAC systems or industrial machinery might fail or require maintenance. This prevents energy waste due to inefficient operations.

Demand Response Optimization: AI can optimize demand response programs by predicting peak demand periods and adjusting energy usage accordingly, reducing costs and strain on the grid.

Energy Efficiency Recommendations: AI can offer personalized energy-saving recommendations based on historical data and user behavior. This can be applied to households, commercial buildings, and industrial settings.

Energy Grid Management: AI can help grid operators manage the distribution of energy more efficiently by analyzing data from various sources and adjusting supply in real-time.

Energy Consumption Anomalies Detection: AI can identify unusual consumption patterns that might indicate theft or equipment malfunctions, improving security and efficiency.

Renewable Energy Integration: AI can optimize the integration of renewable energy sources, ensuring they are used efficiently and reliably.

Energy Pricing Forecast: AI can predict energy pricing, allowing consumers to adjust their usage to take advantage of lower costs.

Carbon Emission Reduction: AI can calculate and minimize the carbon footprint of energy consumption by suggesting cleaner energy sources and consumption patterns.

Home Energy Management: AI-powered systems can control and optimize the energy usage of appliances and devices in homes to reduce waste and lower bills.

These innovations can lead to a more sustainable and cost-effective energy future, benefiting both consumers and the environment.

a) IMPORT DATASET

To begin this exploratory analysis, first use `matplotlib` to import libraries and define functions for plotting the data. Depending on the data, not all plots will be made.

```
from mpl_toolkits.mplot3d import Axes3D
```

```
from sklearn.preprocessing import StandardScaler
```

```
import matplotlib.pyplot as plt # plotting
```

```
import numpy as np # linear algebra
```

```
import os # accessing directory structure
```

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

There are 13 csv files in the current version of the dataset:

```
print(os.listdir('../input'))
```

```
['FE_hourly.csv',          'NI_hourly.csv',          'DOM_hourly.csv',  
'est_hourly.paruquet',    'PJMW_hourly.csv',        'COMED_hourly.csv',  
'EKPC_hourly.csv',        'PJME_hourly.csv',        'DEOK_hourly.csv',  
'DAYTON_hourly.csv',      'DUQ_hourly.csv',         'PJM_Load_hourly.csv',  
'pjm_hourly_est.csv', 'AEP_hourly.csv']
```

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
import warnings
```

```
warnings.filterwarnings("ignore", category=UserWarning)
```

```
RED = "\033[91m"
```

```
GREEN = "\033[92m"
```

```
YELLOW = "\033[93m"
```

```
BLUE = "\033[94m"
```

```
RESET = "\033[0m"
```

```
df =
```

```
pd.read_csv("/kaggle/input/hourly-energy-consumption/AEP_hourly.csv")
```

```
"Datetime"] = pd.to_datetime(df["Datetime"])
```

```
# DATA CLEANING
```

```

print(BLUE + "\nDATA CLEANING" + RESET)

# --- Check for missing values
missing_values = df.isnull().sum()

print(GREEN + "Missing Values : " + RESET)

print(missing_values)

# --- Handle missing values
df.dropna(inplace=True)

# --- Check for duplicate values
duplicate_values = df.duplicated().sum()

print(GREEN + "Duplicate Values : " + RESET)

print(duplicate_values)

# --- Drop duplicate values
df.drop_duplicates(inplace=True)


# DATA ANALYSIS

print(BLUE + "\nDATA ANALYSIS" + RESET)

# --- Summary Statistics
summary_stats = df.describe()

print(GREEN + "Summary Statistics : " + RESET)

print(summary_stats)

# DATA VISUALIZATION

print(BLUE + "\nDATA VISUALIZATION" + RESET)

# --- Line plot

print(GREEN + "LinePlot : " + RESET)

plt.figure(figsize=(10, 6))

sns.lineplot(data=df, x="Datetime", y="AEP_MW")

plt.xlabel("Datetime")

plt.ylabel("Energy Consumption (MW)")

plt.title("Energy Consumption Over Year")

plt.grid()

```

```
plt.show()

# --- Histogram

print(GREEN + "Histogram : " + RESET)

plt.figure(figsize=(10, 6))

plt.hist(

    df["AEP_MW"],

    bins=100,

    histtype="barstacked"

    edgecolor="white",

)

plt.xlabel("AEP_MW")

plt.ylabel("Frequency")

plt.title("Histogram of MEGAWATT USAGE")

plt.show()
```

DATA CLEANING

Missing Values :

Datetime 0

AEP_MW 0

dtype: int64

Duplicate Values :

0

DATA ANALYSIS

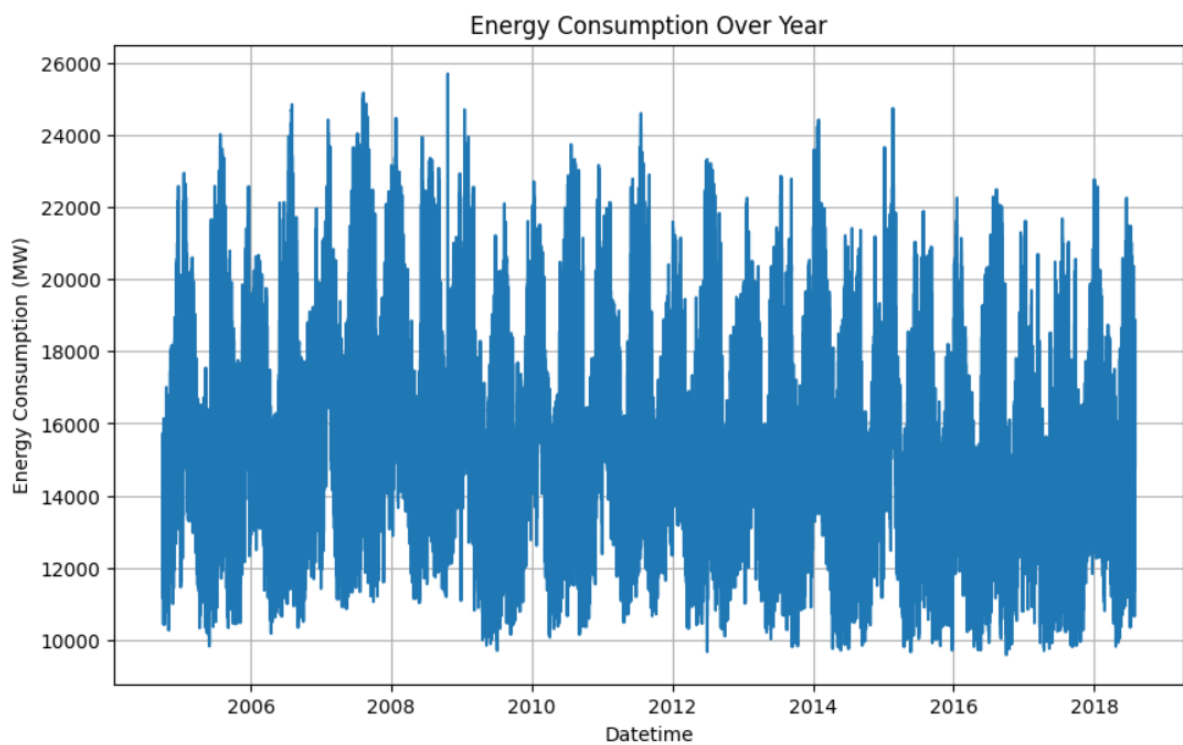
Summary Statistics :

	Datetime	AEP_MW
count	121273	121273.000000
mean	2011-09-02 03:17:01.553025024	15499.513717
min	2004-10-01 01:00:00	9581.000000
25%	2008-03-17 15:00:00	13630.000000

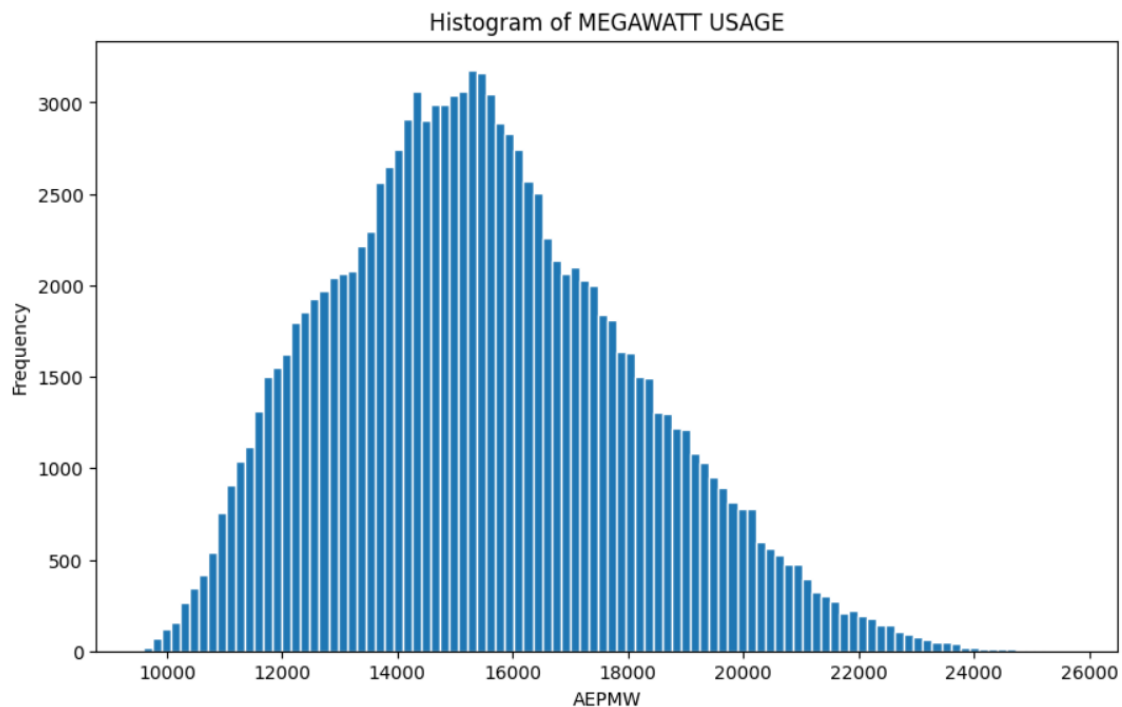
50%	2011-09-02 04:00:00	15310.000000
75%	2015-02-16 17:00:00	17200.000000
max	2018-08-03 00:00:00	25695.000000
std	NaN	2591.399065

DATA VISUALIZATION

LinePlot :

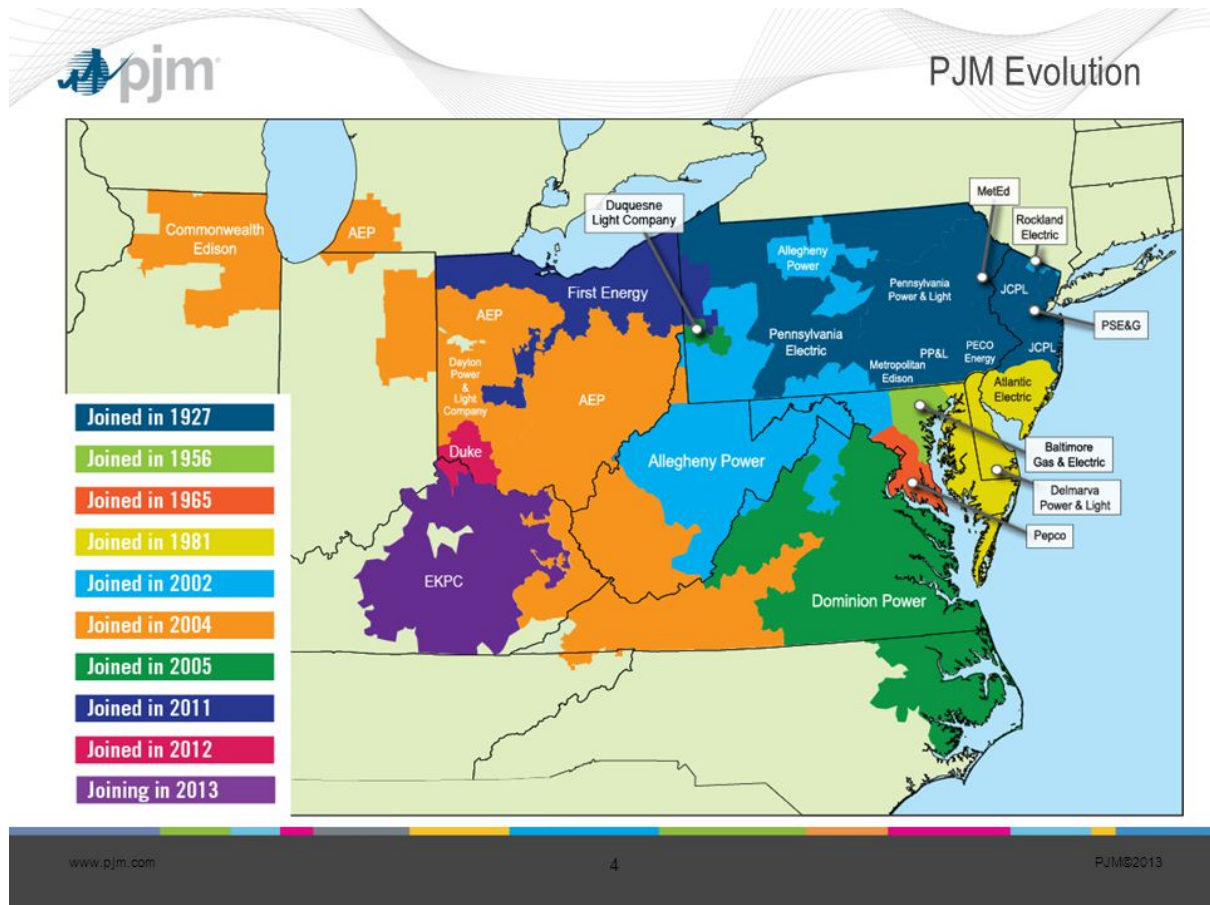


Histogram :



III. DEVELOPMENT PART I

```
import matplotlib.pyplot as plt
# plotting
import numpy as np
# linear algebra
import os
# accessing directory structure
import pandas as pd
# data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
plt.style.use('ggplot')
# Make it pretty
# Data is saved in parquet format so schema is preserved.
df = pd.read_parquet('../input/est_hourly.parquet')
# Show PJM Regions from IPython.display
Image(url= "http://slideplayer.com/4238181/14/images/4/PJM+Evolution.jpg")
```



```
df.head()
```

```
df.describe().T
```

```
_ = df['PJME'].plot.hist(figsize=(15, 5), bins=200, title='Distribution of PJME Load')
```

```
_ = df['DOM'].plot.hist(figsize=(15, 5), bins=200, title='Distribution of DOMINION Load')
```

```
_ = df.plot.hist(figsize=(15, 5), bins=200, title='Distribution of Load by Region')
```

Plot Time Series

```
plot = df.plot(style='.', figsize=(15, 8), title='Entire PJM Load 1998-2001')
```

Plotting Regions

```
_ = df[['PJM_Load', 'PJME', 'PJM_W']] \ .plot(style='.', figsize=(15, 5), title='PJM Load 1998-2002 - Split East and West 2002-2018')
```

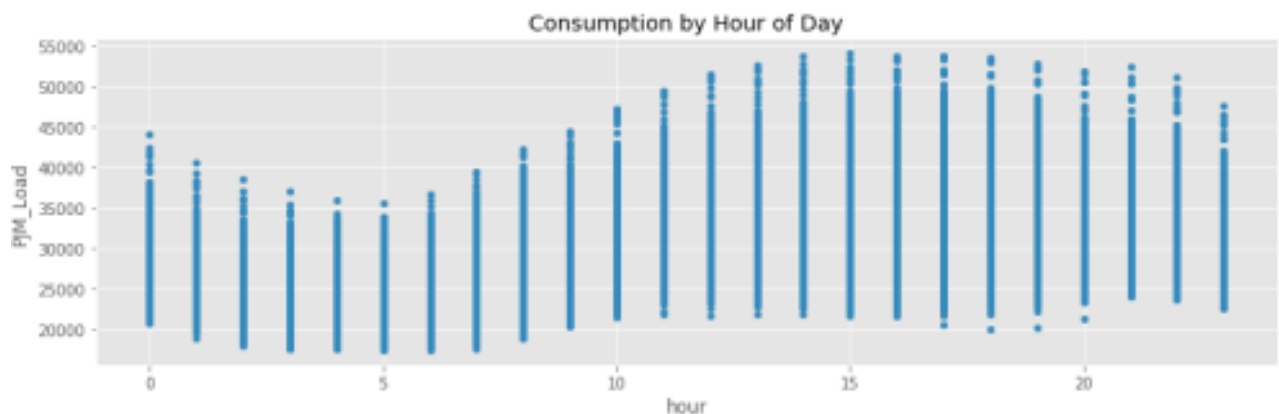
```

_ = df['PJME'].loc[(df['PJME'].index >= '2017-11-01') & (df['PJME'].index <
'2017-12-01')] \ .plot(figsize=(15, 5), title = 'November 2017')

_ = df['PJME'].loc[(df['PJME'].index >= '2017-06-01') & (df['PJME'].index <
'2017-07-01')] \ .plot(figsize=(15, 5), title = 'June 2017')

Create Time Series Features df['dow'] = df.index.dayofweek df['doy'] =
df.index.dayofyear df['year'] = df.index.year df['month'] = df.index.month df['quarter']
= df.index.quarter df['hour'] = df.index.hour df['weekday'] = df.index.weekday_name
df['woy'] = df.index.weekofyear df['dom'] = df.index.day # Day of Month df['date'] =
df.index.date _ = df[['PJM_Load', 'hour']].plot(x='hour', y='PJM_Load', kind='scatter',
figsize=(14,4), title='Consumption by Hour of Day')

```



IV. DEVELOPMENT PART II

Overview Of The Process

The goal of this phase is to process the energy consumption data for time series forecasting, you'll typically need to perform several steps, including data loading, cleaning, and transformation.

Exploratory Data Analysis (EDA):

Perform basic data exploration to understand the data's characteristics. Visualise the time series data to identify trends and patterns.

```
# This Python 3 environment comes with many helpful analytics libraries installed
```

```
# It is defined by the kaggle/python Docker image:  
https://github.com/kaggle/docker-python
```

```
# For example, here's several helpful packages to load
```

```
import numpy as np # linear algebra
```

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
```

```
# For example, running this (by clicking run or pressing Shift+Enter)  
will list all files under the input directory
```

```
import os
```

```
for dirname, _, filenames in os.walk('/kaggle/input'):
```

```
    for filename in filenames:
```

```
        print(os.path.join(dirname, filename))
```

```
# You can write up to 20GB to the current directory (/kaggle/working/)  
that gets preserved as output when you create a version using "Save & Run  
All"
```

```
# You can also write temporary files to /kaggle/temp/, but they won't be  
saved outside of the current session
```

```
/kaggle/input/hourly-energy-consumption/est_hourly.paruquet
/kaggle/input/hourly-energy-consumption/DOM_hourly.csv
/kaggle/input/hourly-energy-consumption/EKPC_hourly.csv
/kaggle/input/hourly-energy-consumption/DUQ_hourly.csv
/kaggle/input/hourly-energy-consumption/DAYTON_hourly.csv
/kaggle/input/hourly-energy-consumption/PJME_hourly.csv
/kaggle/input/hourly-energy-consumption/PJM_Load_hourly.csv
/kaggle/input/hourly-energy-consumption/NI_hourly.csv
/kaggle/input/hourly-energy-consumption/FE_hourly.csv
/kaggle/input/hourly-energy-consumption/COMED_hourly.csv
/kaggle/input/hourly-energy-consumption/AEP_hourly.csv
/kaggle/input/hourly-energy-consumption/pjm_hourly_est.csv
/kaggle/input/hourly-energy-consumption/DEOK_hourly.csv
/kaggle/input/hourly-energy-consumption/PJMW_hourly.csv
```

Data Preprocessing:

```
import pandas as pd
```

```
# Load the dataset from Kaggle, setting the "Datetime" column as the
index and parsing it as dates
```

```
data =
```

```
pd.read_csv('/kaggle/input/hourly-energy-consumption/PJME_hourly.csv',
index_col=[0], parse_dates=[0])
```

```
# Display the first few rows of the dataset
```

```
print("First Few Rows:")
```

```
print(data.head())
```

```
# Display the last few rows of the dataset
```

```
print("\nLast Few Rows:")
```

```
print(data.tail())
```

First Few Rows:

PJME_MW

Datetime

```
2002-12-31 01:00:00    26498.0
2002-12-31 02:00:00    25147.0
2002-12-31 03:00:00    24574.0
2002-12-31 04:00:00    24393.0
2002-12-31 05:00:00    24860.0
```

Last Few Rows:

```
                PJME_MW
Datetime
2018-01-01 20:00:00    44284.0
2018-01-01 21:00:00    43751.0
2018-01-01 22:00:00    42402.0
2018-01-01 23:00:00    40164.0
2018-01-02 00:00:00    38608.0
```

```
# Check the column names in your DataFrame
```

```
column_names = data.columns
```

```
print("Column Names:", column_names)
```

```
Column Names: Index(['PJME_MW'], dtype='object')
```

```
# Check for missing values in the "PJME_MW" column
```

```
missing_values = data['PJME_MW'].isnull().sum()
```

```
# Print the count of missing values
```

```
print("Number of Missing Values in 'PJME_MW' column:", missing_values)
```

```
Number of Missing Values in 'PJME_MW' column: 0
```

```
# Resample to daily data by computing the daily mean
```

```
daily_data = data.resample('D').mean()
```

```
# Print the first few rows of the resampled data
```

```
print(daily_data.head())
```

```
PJME_MW
Datetime
2002-01-01    31080.739130
2002-01-02    34261.541667
2002-01-03    34511.875000
2002-01-04    33715.458333
```

2002-01-05 30405.125000

***Create Lag Features: ***

Lag features capture autocorrelation by including previous values of the target variable "AEP_MW" as features.

```
# Create lag features for 1 hour, 2 hours, and 3 hours
```

```
data['AEP_MW_Lag1'] = data['PJME_MW'].shift(1)
```

```
data['AEP_MW_Lag2'] = data['PJME_MW'].shift(2)
```

```
data['AEP_MW_Lag3'] = data['PJME_MW'].shift(3)
```

```
# Drop rows with missing values in the lag features
```

```
data.dropna(inplace=True)
```

```
# Print the DataFrame with lag features
```

```
print(data.head())
```

PJME_MW	AEP_MW_Lag1	AEP_MW_Lag2	AEP_MW_Lag3
Datetime			
2002-12-31 04:00:00	24393.0	24574.0	25147.0
2002-12-31 05:00:00	24860.0	24393.0	24574.0
2002-12-31 06:00:00	26222.0	24860.0	24393.0
2002-12-31 07:00:00	28702.0	26222.0	24860.0
2002-12-31 08:00:00	30698.0	28702.0	26222.0

Extract Time-Based Features:

Extracting time-based features allows you to capture patterns related to the time of day, day of the week, or seasonality.

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
# Load the time series data (assuming you already have the data loaded)
```

```
# data = pd.read_csv('your_dataset.csv', index_col=[0], parse_dates=[0])
```

```
# Extract day of the week and hour of the day
```

```
data['DayOfWeek'] = data.index.dayofweek # Monday=0, Sunday=6
```

```
data['HourOfDay'] = data.index.hour
```

```

# Extract month and year
data['Month'] = data.index.month
data['Year'] = data.index.year

# Extract quarter
data['Quarter'] = data.index.quarter

# Extract week of the year
data['WeekOfYear'] = data.index.isocalendar().week

# Create binary holiday indicator (replace 'your_holiday_date' with an
actual date)
data['IsHoliday'] = (data.index.date ==
'your_holiday_date').astype(int)

# Calculate 7-day and 30-day moving averages
data['7DayMovingAvg'] = data['PJME_MW'].rolling(window=7).mean()
data['30DayMovingAvg'] = data['PJME_MW'].rolling(window=30).mean()

# Calculate cyclical features for day of the week
data['DayOfWeekSin'] = np.sin(2 * np.pi * data['DayOfWeek'] / 6)
data['DayOfWeekCos'] = np.cos(2 * np.pi * data['DayOfWeek'] / 6)

# Visualize time series data
plt.figure(figsize=(12, 6))

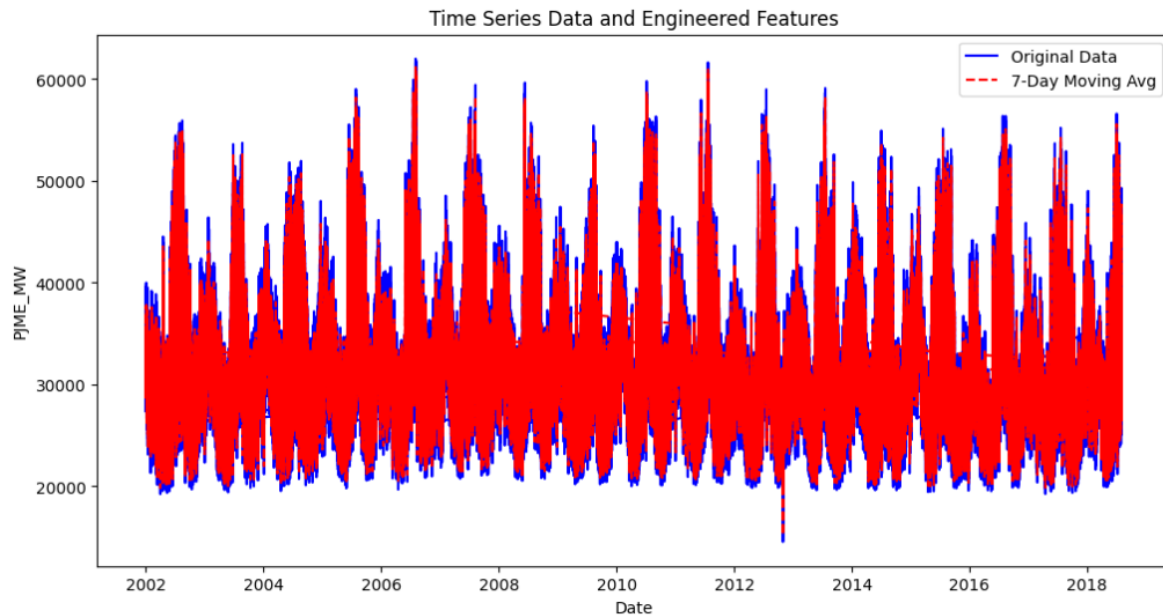
# Plot the original time series data
plt.plot(data.index, data['PJME_MW'], label='Original Data',
color='blue')

# Plot some of the engineered features for illustration (e.g., 7-day
moving average)
plt.plot(data.index, data['7DayMovingAvg'], label='7-Day Moving Avg',
color='red', linestyle='--')

# Customize the plot
plt.xlabel('Date')
plt.ylabel('PJME_MW')
plt.title('Time Series Data and Engineered Features')
plt.legend()

# Show the plot
plt.show()

```



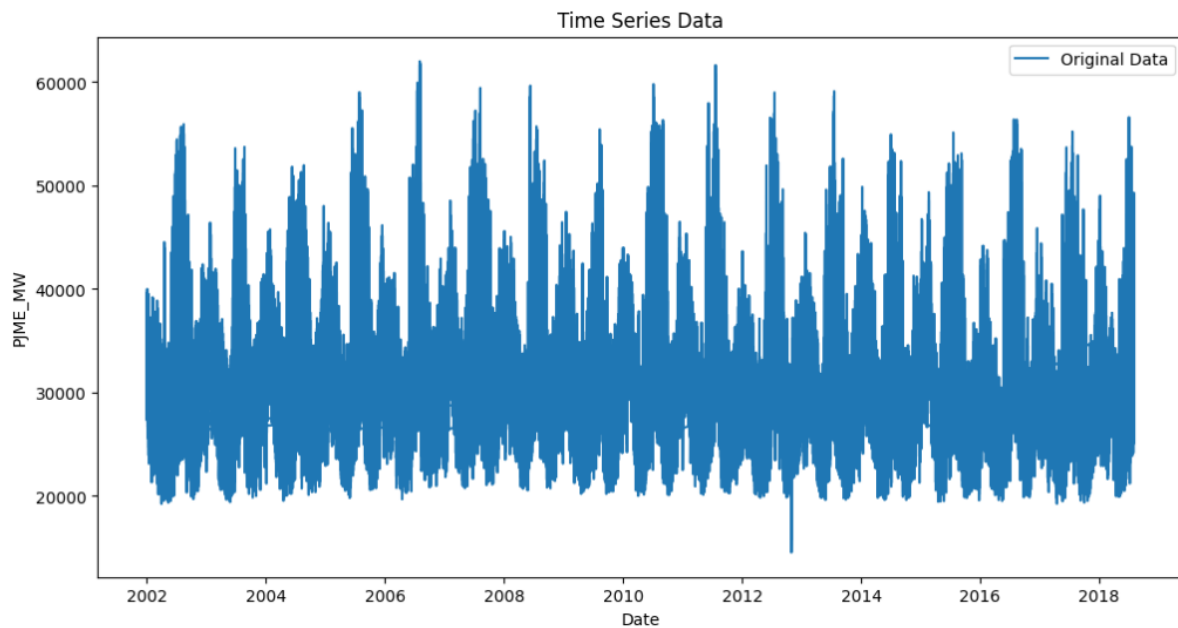
Decompose the Time Series:

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['PJME_MW'], label='Original Data')
plt.xlabel('Date')
plt.ylabel('PJME_MW') # Replace with the correct column name if needed
plt.title('Time Series Data')
plt.legend()
plt.show()
```

```
train = data.loc[data.index < '01-01-2015']
test = data.loc[data.index >= '01-01-2015']
```

```
fig, ax = plt.subplots(figsize=(15, 5))
train.plot(ax=ax, label='Training Set', title='Data Train/Test Split')
test.plot(ax=ax, label='Test Set')
ax.axvline('01-01-2015', color='black', ls='--')
ax.legend(['Training Set', 'Test Set'])
plt.show()
```



`/opt/conda/lib/python3.10/site-packages/IPython/core/pylabtools.py:152:`
 UserWarning: Creating legend with loc="best" can be slow with large amounts of data.

```
fig.canvas.print_figure(bytes_io, **kw)
```



```
import xgboost as xgb
from sklearn.metrics import mean_squared_error
```

```
# Convert the date index to numerical representation (e.g., days since the start)
```

```
train['date_numeric'] = (train.index - train.index.min()).days
test['date_numeric'] = (test.index - train.index.min()).days
```

```
# Create an XGBoost regressor instance (you can tune hyperparameters)
model = xgb.XGBRegressor()
```

```

# Train the model on the training data
X_train = train['date_numeric'].values.reshape(-1, 1)
y_train = train['PJME_MW']
model.fit(X_train, y_train)

# Make predictions on the test set
X_test = test['date_numeric'].values.reshape(-1, 1)
test['prediction'] = model.predict(X_test)

# Calculate RMSE as an example of evaluation metric
rmse = mean_squared_error(test['PJME_MW'], test['prediction'],
squared=False)
print("Root Mean Squared Error (RMSE):", rmse)

# Visualization (plot actual vs. predicted)
fig, ax = plt.subplots(figsize=(15, 5))
test[['PJME_MW']].plot(ax=ax, label='Actual Load')
test['prediction'].plot(ax=ax, label='Predicted Load', linestyle='--')
ax.legend()
plt.title('Actual vs. Predicted Load')
plt.show()

```

/tmp/ipykernel_20/1741646912.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:

https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
train['date_numeric'] = (train.index - train.index.min()).days
```

/tmp/ipykernel_20/1741646912.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:

https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test['date_numeric'] = (test.index - train.index.min()).days
```

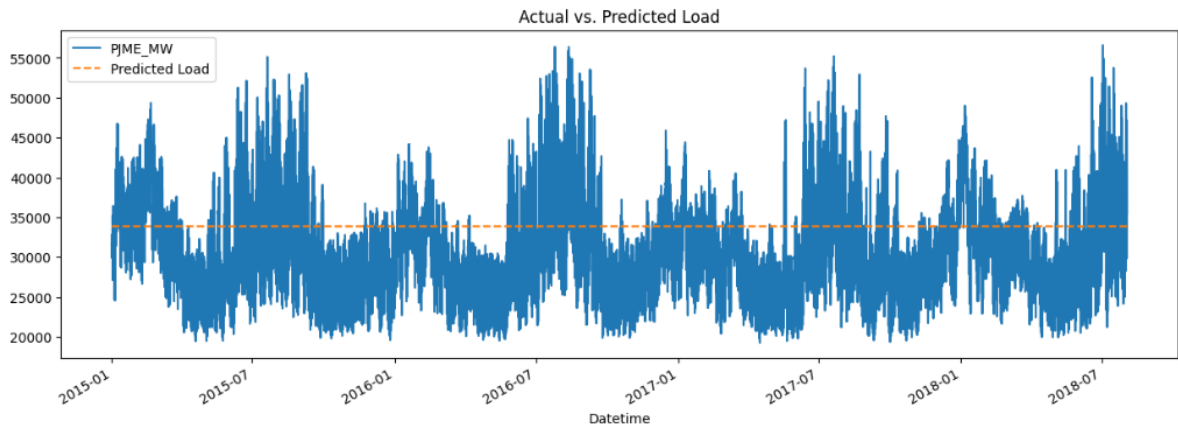
/tmp/ipykernel_20/1741646912.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:

https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy


```
test['prediction'] = model.predict(X_test)
```

Root Mean Squared Error (RMSE): 6933.449249719269



V. TRAINING & EVALUATION

Setup.

While the manual installation can walk you through around the various commands basics for each used packages in the project, chances are you might already know those commands and you don't want to bother yourself about taping every single command, so that's why we have provided the possibility to use make scripting, to make life easy for you. You find in the following the commands you will need to do this.

In case you have it installed in your system, for Linux based system it comes already installed in your system you don't need to install anything just sleep this part to {...}. For windows based systems there are multiple ways to get GNU make installed, like for example Cygwin, Nmake, Cmake..., we recommend to go for chocolatey, we think it's the most straightforward way to install make for windows systems with less effort.

chocolatey.

First thing first, we will install chocolatey, make sure you are using the Powershell command as an admin,

Then run this command first :

Get-ExecutionPolicy

If it returns Restricted, then run :

Set-ExecutionPolicy AllSigned or Set-ExecutionPolicy Bypass -Scope Process

Now, to install chocolatey run the following command by coping it at once and past it in command line, then hit enter:

```
Set-ExecutionPolicy Bypass -Scope Process -Force;  
[System.Net.ServicePointManager]::SecurityProtocol =  
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object  
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

Please check chocolatey website for more guidance !

Make.

Now that you have chocolatey installed, we can install make by running the command :

```
choco install make --version=3.81
```

Once make installation is done , and assuming that you have downloaded the project files in your local machine it's very easy to workout everything.

To create the conda environemnt run the command :

- \$ make create_env

To setup the project run the command :

- \$ make setup

Test units.

Testing code is important to guarantee consistency and availability of this project. Test units are devide in two main aspects: testing code syntax and style conforming to PEP8 standard, testing internal modules which are mainly functions used for implemeting the code.

environment testing.

To test Python environment run the command :

- \$ make env_test

Testing syntax & style.

To lint code scripts we are using flake8, just run the following command :

- \$ make lint

Under the hood, make will go over the Makefile located in our directory which itself will chain to all coding resources in /src/. and /notebooks directories and will check the syntax and style of your code using flake8 to meet PEP8 standards.

Model Dataset.

The collected data used in this particular project use case is based on the author work experience and estimations.

Setup.py

```
import setuptools

from setuptools import find_namespace_packages, setup


def get_install_requirements():

    with open("requirements.txt", "r", encoding="utf-8") as f:

        reqs = [x.strip() for x in f.read().splitlines()]

        reqs = [x for x in reqs if not x.startswith("#")]

    return reqs


setup(name='AIEMS',

      version='0.1.4',

      description='Ai-based Energy Managment System',
```

```
author='Ahmed Mabrouk',

author_email='mabrook.amed@gmail.com',

license='CC0 1.0 Universal',

        packages=setuptools.find_namespace_packages(include=['src',
'src.models','src.features', 'src.data'] ),

install_requires= get_install_requirements()

)
```

Test_env.py

```
import sys
```

```
REQUIRED_PYTHON = "python3"
```

```
def main():
```

```
    system_major = sys.version_info.major
```

```
    if REQUIRED_PYTHON == "python":
```

```
        required_major = 2
```

```
        elif REQUIRED_PYTHON == "python3":
```

```
            required_major = 3
```

else:

```
raise ValueError("Unrecognized python interpreter: {}".format(
    REQUIRED_PYTHON))
```

if system_major != required_major:

```
raise TypeError(
```

```
"This project requires Python {}. Found: Python {}".format(
    required_major, sys.version))
```

else:

```
print(">>> Development environment passes all tests!")
```

```
if __name__ == '__main__':
```

```
    main()
```

CONCLUSION

Artificial intelligence (AI) is revolutionising the measurement of energy consumption with several key trends. Firstly, AI-powered smart meters and IoT sensors are becoming ubiquitous, providing real-time data for granular analysis. Secondly, machine learning algorithms are being used for load forecasting, helping utilities

predict and manage energy demand more accurately. Thirdly, AI-driven building and industrial energy management systems are optimizing energy usage in real-time, reducing waste. Moreover, AI is facilitating predictive maintenance of energy-related equipment, ensuring operational efficiency. Lastly, AI is supporting the development of data-driven energy policies and grid expansion plans, driving the transition to cleaner and more sustainable energy sources. These trends collectively enhance energy efficiency, reduce costs, and promote environmental sustainability in the energy sector.