MEASURE ENERGY CONSUMPTION

Team Leader

422521104016 : KAMALESH P

Phase-2 Documentation Submission

Introduction:

- Measuring energy consumption in Python is an essential practice for assessing and optimizing the efficiency of devices and systems. Python, as a versatile and powerful programming language, provides a flexible and capable platform for collecting, analyzing, and visualizing energy consumption data. In this context, we'll explore the fundamental concepts and tools involved in measuring and managing energy consumption using Python, empowering you to make data-driven decisions and contribute to sustainability efforts.
- Traditional methods for measure energy consumption, such as time series forcasting and deep learning(RNN-LSTM) are used for this project.

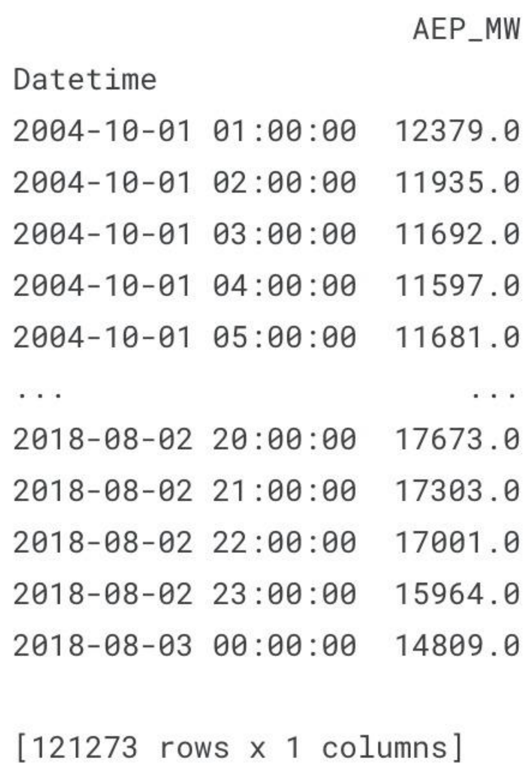
Content for Project Phase 2 :

- Consider exploring advanced techniques like time series forcasting and deep learning models (e.g., RNN-LSTM) for improved measure energy consumption accuracy.

Data Source :

- A good data source for measure energy consumption using machine learning and deep learning should be Accurate.
- The dataset used in this project is obtained from Kaggle.

Dataset Link: **https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption**



```
                      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]
```

Modules:

- Data Source Identification
- Data Preprocessing Module
- Feature Extraction Module
- Model development

- Visualization
- Automation

Data Source Identification:

- Identify a suitable dataset that contains energy consumption measurements. Some potential sources include government agencies, utility companies, research institutions, or open data portals. You can search for datasets on websites like data.gov, Kaggle, or the U.S. Energy Information Administration (EIA) website.

Data Preprocessing:

- Clean the dataset by addressing missing values, outliers, and data inconsistencies.
- Convert data types as needed (e.g., date/time conversion).

Feature Extraction Module:

Identify relevant features and metrics that can provide insights into energy consumption. These might include:

- Time-based features (hourly, daily, monthly, seasonal patterns).
- Weather data (temperature, humidity) if available, as it can impact energy usage.
- - Demographic data (population, building types) if relevant.

Model Development:

- Utilize statistical analysis techniques to uncover trends, patterns, and anomalies in the data. This can involve time series analysis, regression, clustering, or anomaly detection methods.
- - Develop predictive models if you want to forecast future energy consumption based on historical data.

Visualization:

Create visualizations to present energy consumption trends and insights. This can include:

- Time series plots to visualize changes over time.

- Histograms or bar charts to show distribution of energy consumption.

Automation:

- Build a script or pipeline to automate the entire process, from data collection to visualization.

Use scripting languages like Python to create a workflow that:

- Downloads and updates the dataset at regular intervals.
- Performs data preprocessing and feature extraction.
- Runs statistical analyses and generates insights.

Time Series Forcasting:

Forecasting energy consumption using time series analysis is a common and valuable application of AI and machine learning. It can help utility companies, businesses, and individuals make informed decisions about energy production, distribution, and efficiency. Here's a high-level overview of the steps involved in a time series forecasting project for energy consumption:

- **Data Collection.**
- **Data Preprocessing.**
- **Exploratory Data Analysis (EDA).**
- **Feature Engineering.**
- **Model Selection.**
- **Model Training.**

Deep learning:

RNN:

- Recurrent Neural Networks (RNNs) are a type of deep learning model that has proven to be highly effective for time series forecasting, including the prediction of energy consumption.
- RNNs are used in AI to work with data sequences. They excel in tasks where the order and context of data points matter, such as time series forecasting, natural language processing, and speech recognition.

LSTM:

- LSTM is a type of recurrent neural network (RNN) designed for processing sequential data, making it suitable for tasks where order and temporal dependencies are important.

- LSTMs have memory cells that can store and retrieve information over long sequences, allowing them to capture long-range dependencies in data, making them suitable for tasks like text summarization and machine translation.

Ensemble Method:

- An ensemble method in machine learning is a technique that combines the predictions of multiple individual models to produce a more accurate and robust prediction. Voting ,Stacking ,Boosting ,Bagging these are popular ensemble methods.

Model Evaluation and Selection:

- Split the dataset into training and testing sets.
- Evaluate models using appropriate metrics (e.g., Mean Absolute Error, Mean Squared Error, R-squared) to assess their performance.

- Use cross-validation techniques to tune hyperparameters and ensure model stability.
- Compare the results with traditional linear regression models to highlight improvements.

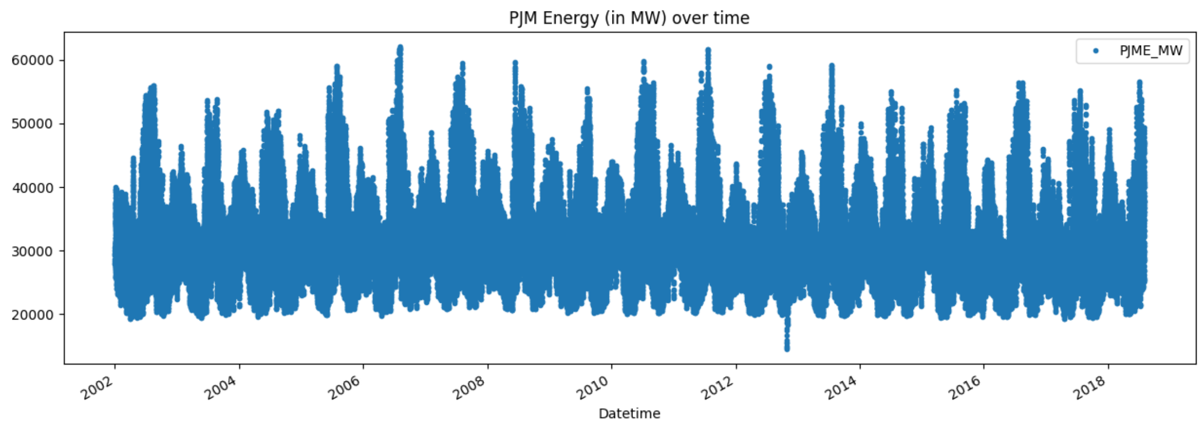
Select the best-performing model for further analysis.

PROGRAM:

```
# import the libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


# customize the style

pd.options.display.float_format = '{:.5f}'.format
pd.options.display.max_rows = 12


# load the data
filepath = '../input/hourly-energy-consumption/PJME_hourly.csv'
df = pd.read_csv(filepath)
```

Explore the data:

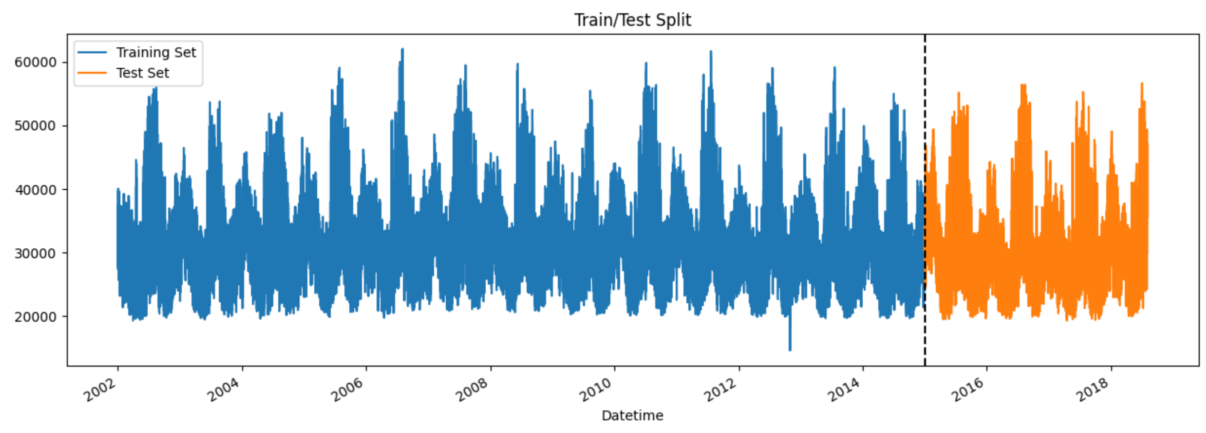
```
# turn data to datetime
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)

# create the plot
df.plot(style='.',
        figsize=(15, 5),
        title='PJM Energy (in MW) over time')
plt.show()
```



Split the data:

```
# train / test split
train = df.loc[df.index < '01-01-2015']
```

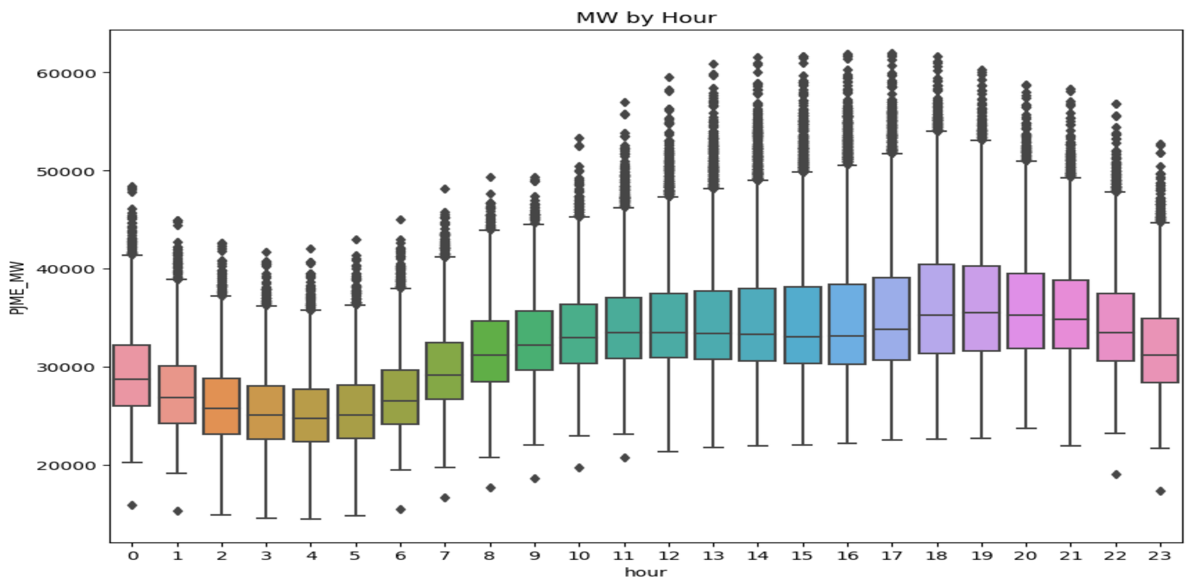
```
test = df.loc[df.index >= '01-01-2015']

fig, ax = plt.subplots(figsize=(15, 5))

train.plot(ax=ax, label='Training Set', title='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()
```



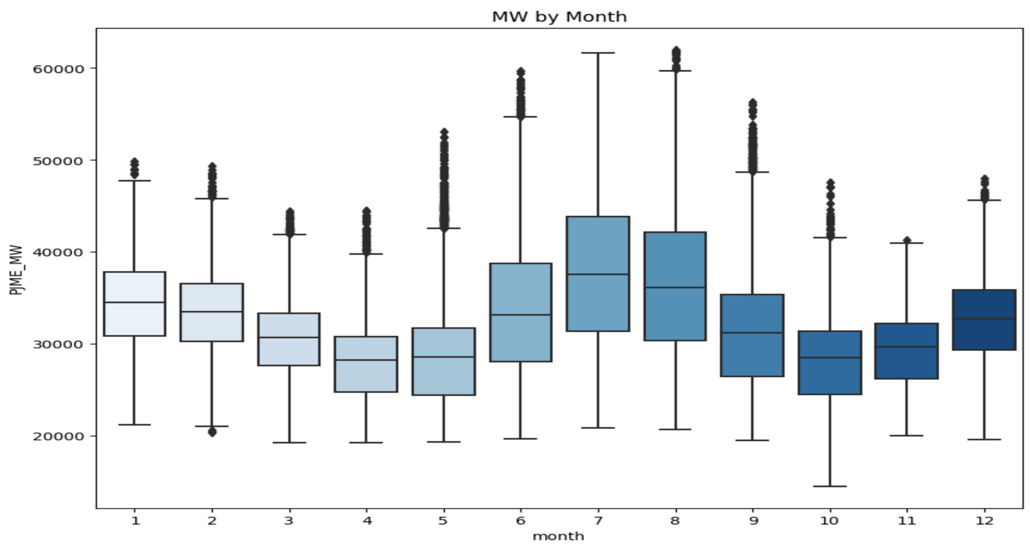
Feature Engineering:

```
# feature creation
def create_features(df):
    df = df.copy()
    df['hour'] = df.index.hour
    df['dayofweek'] = df.index.dayofweek
    df['quarter'] = df.index.quarter
    df['month'] = df.index.month
    df['year'] = df.index.year
    df['dayofyear'] = df.index.dayofyear
    df['dayofmonth'] = df.index.day
```

```
  df['weekofyear'] = df.index.isocalendar().week
    return df

df = create_features(df)
# visualize the hourly Megawatt
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='hour', y='PJME_MW')
ax.set_title('MW by Hour')
plt.show()
```



```
# viaualize the monthly Megawatt
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='month', y='PJME_MW', palette='Blues')
ax.set_title('MW by Month')
plt.show()
```



Modelling:

Prepare the data:

```
# preprocessing

train = create_features(train)

test = create_features(test)

features = ['dayofyear', 'hour', 'dayofweek', 'quarter', 'month', 'year']

target = 'PJME_MW'

X_train = train[features]

y_train = train[target]

X_test = test[features]

y_test = test[target]
```

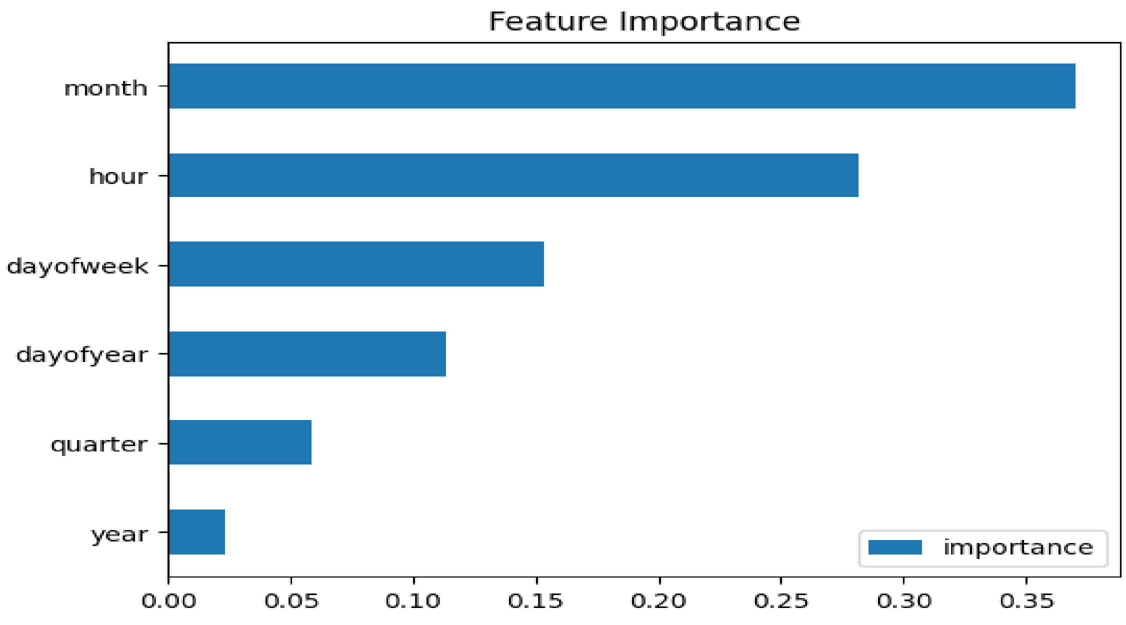
Build the model:

```
import xgboost as xgb

from sklearn.metrics import mean_squared_error

# build the regression model

reg = xgb.XGBRegressor(base_score=0.5, booster='gbtree',

                       n_estimators=1000,

                       early_stopping_rounds=50,

                       objective='reg:linear',

                       max_depth=3,

                       learning_rate=0.01)

reg.fit(X_train, y_train,

        eval_set=[(X_train, y_train), (X_test, y_test)],

        verbose=100)
```

```
[14:48:48] WARNING: ../src/objective/regression_obj.cu:213: reg:linear i
s now deprecated in favor of reg:squarederror.
[0]	validation_0-rmse:32605.13860	validation_1-rmse:31657.15907
[100]	validation_0-rmse:12581.21569	validation_1-rmse:11743.75114
[200]	validation_0-rmse:5835.12466	validation_1-rmse:5365.67709
[300]	validation_0-rmse:3915.75557	validation_1-rmse:4020.67023
[400]	validation_0-rmse:3443.16468	validation_1-rmse:3853.40423
[500]	validation_0-rmse:3285.33804	validation_1-rmse:3805.30176
[600]	validation_0-rmse:3201.92936	validation_1-rmse:3772.44933
[700]	validation_0-rmse:3148.14225	validation_1-rmse:3750.91108
[800]	validation_0-rmse:3109.24248	validation_1-rmse:3733.89713
[900]	validation_0-rmse:3079.40079	validation_1-rmse:3725.61224
[999]	validation_0-rmse:3052.73503	validation_1-rmse:3722.92257
```

Features importance:

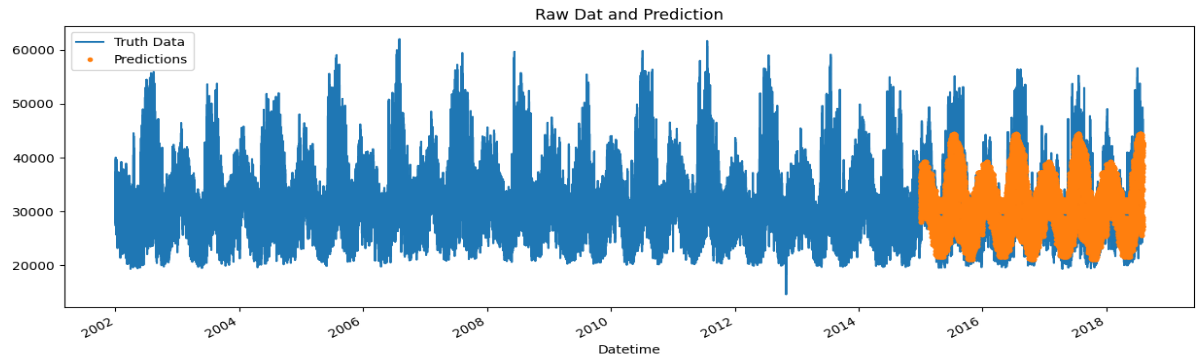
```
fi = pd.DataFrame(data=reg.feature_importances_,
        index=reg.feature_names_in_,
        columns=['importance'])
fi.sort_values('importance').plot(kind='barh', title='Feature Importance')
plt.show()
```



Forecasting on test data:

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

```
df = df.merge(test[['prediction']], how='left', left_index=True,
right_index=True)
ax = df[['PJME_MW']].plot(figsize=(15, 5))
df['prediction'].plot(ax=ax, style='.')
plt.legend(['Truth Data', 'Predictions'])
ax.set_title('Raw Dat and Prediction')
plt.show()
```



```
# Score (RMSE)
score = np.sqrt(mean_squared_error(test['PJME_MW'], test['prediction']))
print(f'RMSE Score on Test set: {score:0.2f}')
```

```
RMSE Score on Test set: 3721.75
```

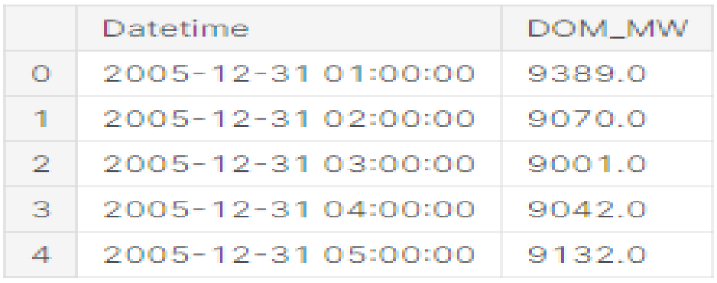
RNN-LSTM:

Libraries and Data Information:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
import tensorflow as tf
from keras.layers import Dense,Dropout,SimpleRNN,LSTM
from keras.models import Sequential
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
    import warnings
warnings.filterwarnings("ignore")
```

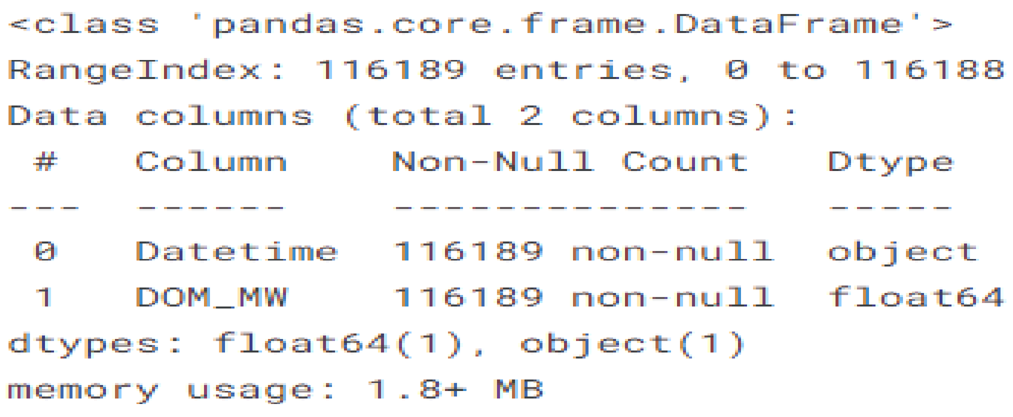
Read and Check Data:

```
df = pd.read_csv("/kaggle/input/hourly-energy-consumption/DOM_hourly.csv")
df.head()
```



	Datetime	DOM_MW
0	2005-12-31 01:00:00	9389.0
1	2005-12-31 02:00:00	9070.0
2	2005-12-31 03:00:00	9001.0
3	2005-12-31 04:00:00	9042.0
4	2005-12-31 05:00:00	9132.0

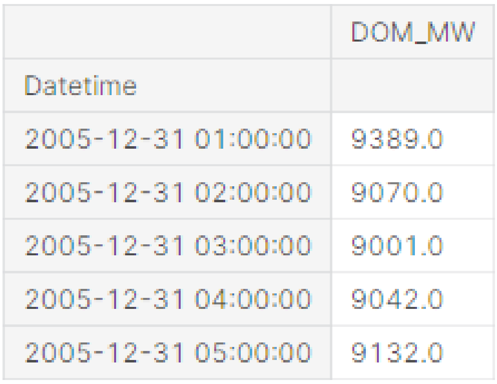
```
df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 116189 entries, 0 to 116188
Data columns (total 2 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   Datetime  116189 non-null  object 
 1   DOM_MW    116189 non-null  float64
dtypes: float64(1), object(1)
memory usage: 1.8+ MB
```

```
# We must convert the Datetime column to Datetime format
df['Datetime'] = pd.to_datetime(df['Datetime'])
```

```
# We index the Datetime column after transformation

df.set_index('Datetime', inplace=True)

df.head()
```

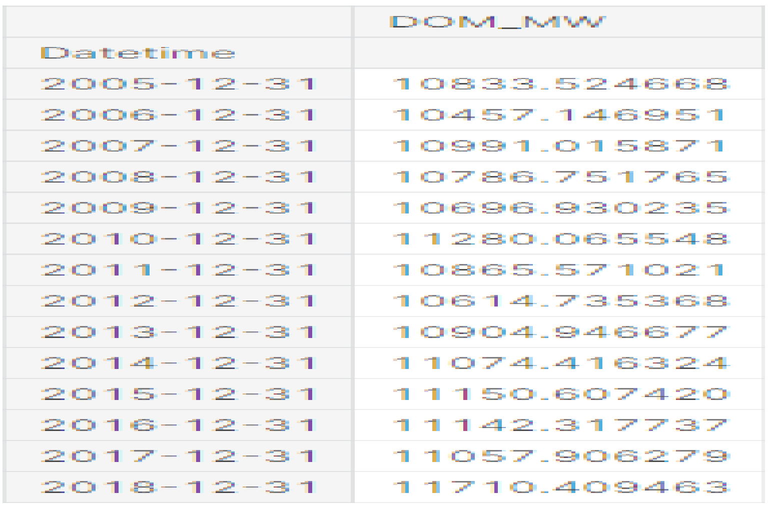


	DOM_MW
Datetime	
2005-12-31 01:00:00	9389.0
2005-12-31 02:00:00	9070.0
2005-12-31 03:00:00	9001.0
2005-12-31 04:00:00	9042.0
2005-12-31 05:00:00	9132.0

```
# Let's look at the years in the data set
years = df.index.year.unique()
years
```

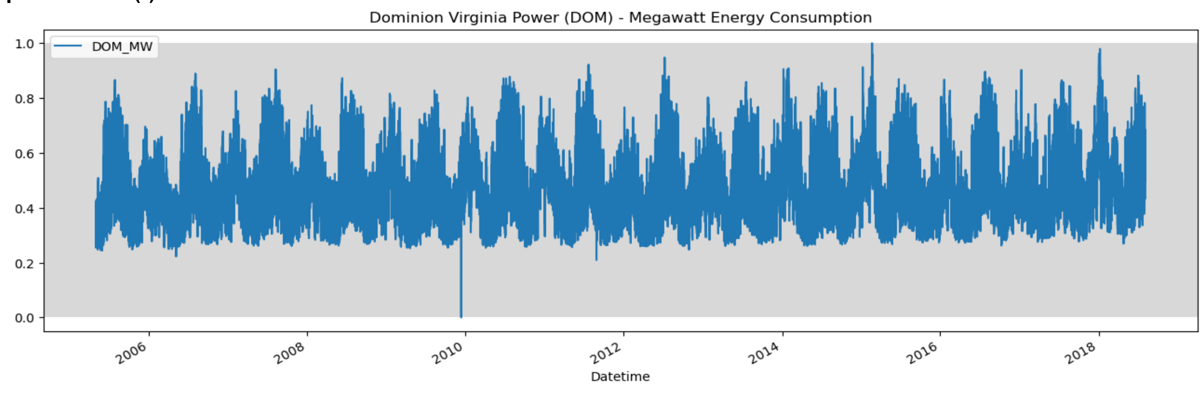
```
Int64Index([2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014,
2015,
            2016, 2017, 2018],
           dtype='int64', name='Datetime')
```

```
# Let's see the average energy consumed per year

df_yearly_avg = df['DOM_MW'].resample('Y').mean()

df_yearly_avg.to_frame()

df.plot(figsize=(16,5),legend=True)
```



	DOM_MW
Datetime	
2005-12-31	10833.524668
2006-12-31	10457.146951
2007-12-31	10991.015871
2008-12-31	10786.751765
2009-12-31	10696.930235
2010-12-31	11280.065548
2011-12-31	10865.571021
2012-12-31	10614.735368
2013-12-31	10904.946677
2014-12-31	11074.416324
2015-12-31	11150.607420
2016-12-31	11142.317737
2017-12-31	11057.906279
2018-12-31	11710.409463

```
plt.axhspan(0, 1, facecolor='gray', alpha=0.3)

plt.title('Dominion Virginia Power (DOM) - Megawatt Energy Consumption')

plt.show()
```

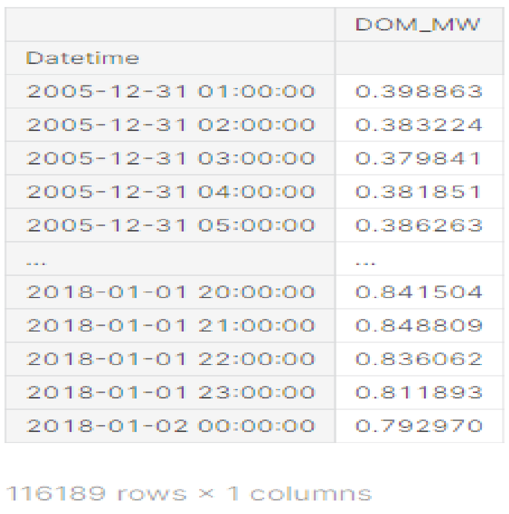


Normalization Process:

```
def normalize_data(df):
    scaler = MinMaxScaler()
    normalized_data = scaler.fit_transform(df['DOM_MW'].values.reshape(-1,1))
    df['DOM_MW'] = normalized_data
    return df, scaler
df_norm, scaler = normalize_data(df)
df_norm.shape
```

```
(116189, 1)
```

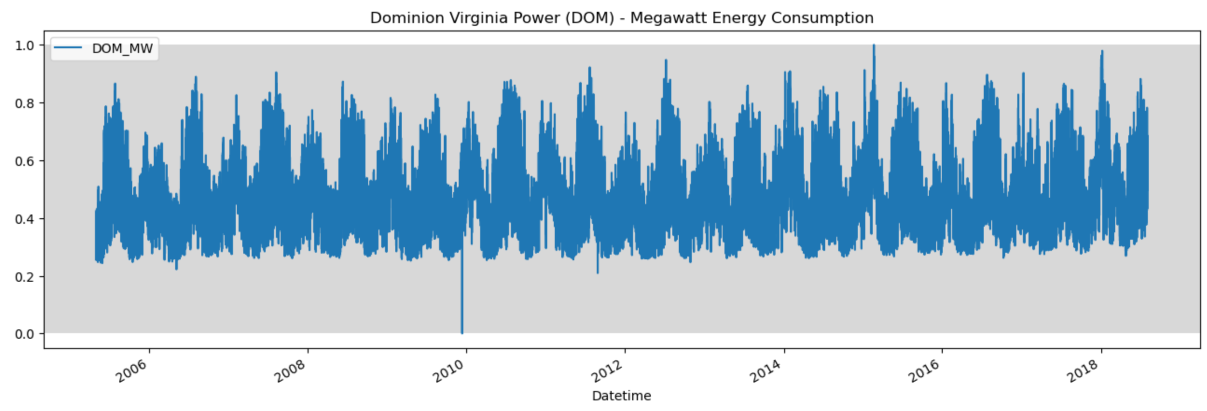
```
df_norm
```



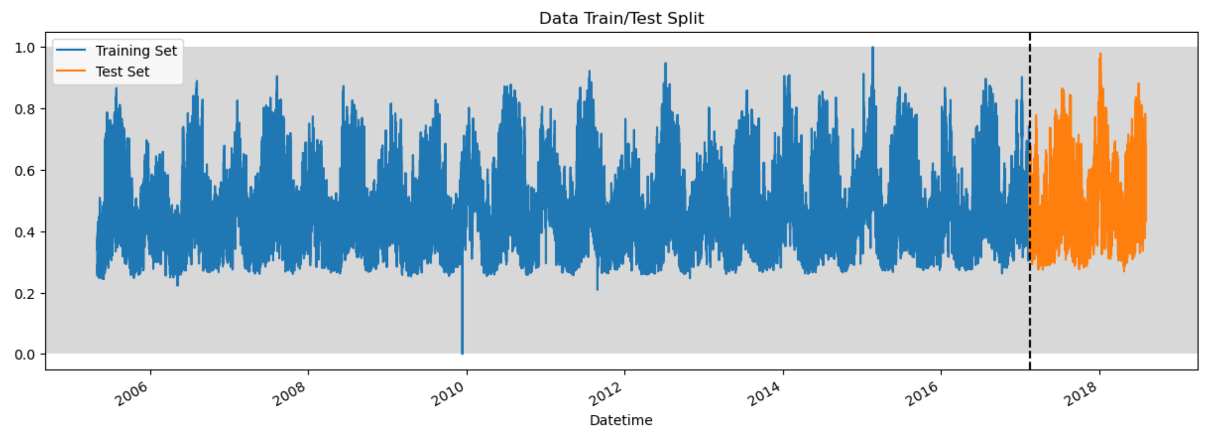
	DOM_MW
Datetime	
2005-12-31 01:00:00	0.398863
2005-12-31 02:00:00	0.383224
2005-12-31 03:00:00	0.379841
2005-12-31 04:00:00	0.381851
2005-12-31 05:00:00	0.386263
...	...
2018-01-01 20:00:00	0.841504
2018-01-01 21:00:00	0.848809
2018-01-01 22:00:00	0.836062
2018-01-01 23:00:00	0.811893
2018-01-02 00:00:00	0.792970

116189 rows × 1 columns

```
# Now after normalization we can observe that the data range on y-axis is 0.0 -
1.0

df.plot(figsize=(16,5),legend=True)

plt.axhspan(0, 1, facecolor='gray', alpha=0.3)

plt.title('Dominion Virginia Power (DOM) - Megawatt Energy Consumption')

plt.show()
```



```
# 2017-02-13 after this date we will choose the test set

split_date = '2017-02-13'

DOM_train = df_norm.loc[df_norm.index <= split_date].copy()

DOM_test = df_norm.loc[df_norm.index > split_date].copy()

fig, ax = plt.subplots(figsize=(15, 5))

DOM_train.plot(ax=ax, label='Training Set', title='Data Train/Test Split')

DOM_test.plot(ax=ax, label='Test Set')

ax.axvline('2017-02-13', color='black', ls='--')

ax.legend(['Training Set', 'Test Set'])

plt.axhspan(0, 1, facecolor='gray', alpha=0.3)

plt.show()
```



Prepare Data for Training the RNN & LSTM:

```
def load_data(data, seq_len):
    X_train = []
    y_train = []
    for i in range(seq_len, len(data)):
        X_train.append(data.iloc[i-seq_len : i, 0])
        y_train.append(data.iloc[i, 0])

  # last 6189 days are going to be used in test
    X_test = X_train[110000:]
    y_test = y_train[110000:]
    # first 110000 days are going to be used in training
    X_train = X_train[:110000]
    y_train = y_train[:110000]
    # convert to numpy array
    X_train = np.array(X_train)
    y_train = np.array(y_train)
    X_test = np.array(X_test)
    y_test = np.array(y_test)

    # reshape data to input into RNN&LSTM models
    X_train = np.reshape(X_train, (110000, seq_len, 1))

    X_test = np.reshape(X_test, (X_test.shape[0], seq_len, 1))
    return [X_train, y_train, X_test, y_test]
seq_len = 20
```

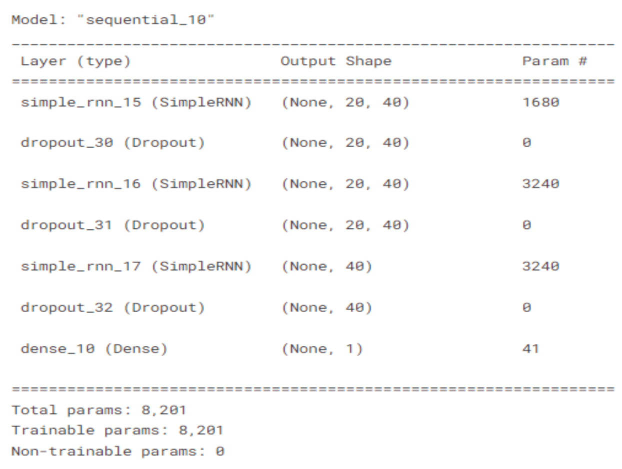
# Let's create train, test data

```
X_train, y_train, X_test, y_test = load_data(df, seq_len)
print('X_train.shape = ',X_train.shape)
print('y_train.shape = ', y_train.shape)
print('X_test.shape = ', X_test.shape)
print('y_test.shape = ',y_test.shape)
```

```
X_train.shape =  (110000, 20, 1)
y_train.shape =  (110000,)
X_test.shape =  (6169, 20, 1)
y_test.shape =  (6169,)
```

Build a RNN model:

```
rnn_model = Sequential()
rnn_model.add(SimpleRNN(40,activation="tanh",return_sequences=True,
input_shape=(X_train.shape[1],1)))
rnn_model.add(Dropout(0.15))
rnn_model.add(SimpleRNN(40,activation="tanh",return_sequences=True))
rnn_model.add(Dropout(0.15))
rnn_model.add(SimpleRNN(40,activation="tanh",return_sequences=False))
rnn_model.add(Dropout(0.15))
rnn_model.add(Dense(1))
rnn_model.summary()
```



```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_15 (SimpleRNN)   (None, 20, 40)            1680

 dropout_30 (Dropout)        (None, 20, 40)            0

 simple_rnn_16 (SimpleRNN)   (None, 20, 40)            3240

 dropout_31 (Dropout)        (None, 20, 40)            0

 simple_rnn_17 (SimpleRNN)   (None, 40)                3240

 dropout_32 (Dropout)        (None, 40)                0

 dense_10 (Dense)            (None, 1)                 41

=================================================================
Total params: 8,201
Trainable params: 8,201
Non-trainable params: 0
```

rnn_model.compile(optimizer="adam",loss="MSE")

rnn_model.fit(X_train, y_train, epochs=10, batch_size=1000

```
Epoch 1/10
110/110 [==============================] - 14s 98ms/step - loss: 0.0969
Epoch 2/10
110/110 [==============================] - 12s 113ms/step - loss: 0.018
0
Epoch 3/10
110/110 [==============================] - 12s 111ms/step - loss: 0.010
0
Epoch 4/10
110/110 [==============================] - 10s 94ms/step - loss: 0.0070
Epoch 5/10
110/110 [==============================] - 10s 92ms/step - loss: 0.0054
Epoch 6/10
110/110 [==============================] - 10s 94ms/step - loss: 0.0044
Epoch 7/10
110/110 [==============================] - 10s 92ms/step - loss: 0.0038
Epoch 8/10
110/110 [==============================] - 10s 93ms/step - loss: 0.0032
Epoch 9/10
110/110 [==============================] - 10s 90ms/step - loss: 0.0029
Epoch 10/10
110/110 [==============================] - 10s 93ms/step - loss: 0.0026
```

rnn_predictions = rnn_model.predict(X_test)

rnn_score = r2_score(y_test,rnn_predictions)

print("R2 Score of RNN model = ",rnn_score)

```
193/193 [==============================] - 2s 8ms/step
R2 Score of RNN model =  0.9504153338386626
```

Build an LSTM model:

```
lstm_model = Sequential()

lstm_model.add(LSTM(40,activation="tanh",return_sequences=True,
input_shape=(X_train.shape[1],1)))

lstm_model.add(Dropout(0.15))

lstm_model.add(LSTM(40,activation="tanh",return_sequences=True))

lstm_model.add(Dropout(0.15))

lstm_model.add(LSTM(40,activation="tanh",return_sequences=False))

lstm_model.add(Dropout(0.15))

lstm_model.add(Dense(1))

lstm_model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_18 (LSTM)              (None, 20, 40)            6720

 dropout_36 (Dropout)        (None, 20, 40)            0

 lstm_19 (LSTM)              (None, 20, 40)            12960

 dropout_37 (Dropout)        (None, 20, 40)            0

 lstm_20 (LSTM)              (None, 40)                12960

 dropout_38 (Dropout)        (None, 40)                0

 dense_12 (Dense)            (None, 1)                 41

=================================================================
Total params: 32,681
Trainable params: 32,681
Non-trainable params: 0
_________________________________________________________________
```

```
lstm_model.compile(optimizer="adam",loss="MSE")

lstm_model.fit(X_train, y_train, epochs=10, batch_size=1000)
```

```
Epoch 1/10
110/110 [==============================] - 33s 240ms/step - loss: 0.021
8
Epoch 2/10
110/110 [==============================] - 26s 240ms/step - loss: 0.012
1
```

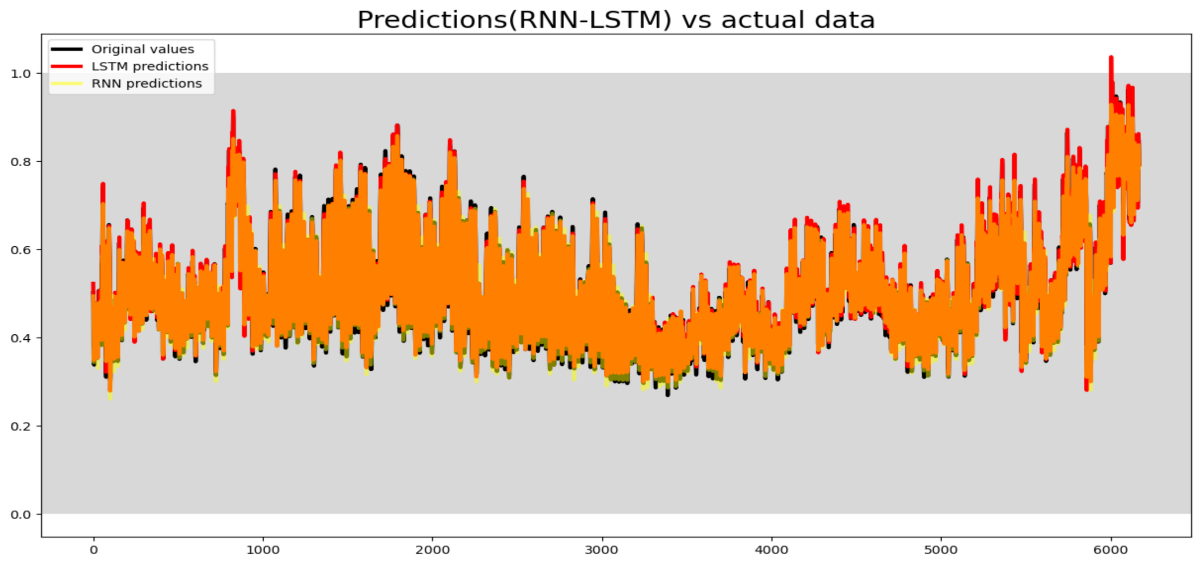
```
Epoch 3/10
110/110 [==============================] - 26s 241ms/step - loss: 0.010
5
Epoch 4/10
110/110 [==============================] - 26s 234ms/step - loss: 0.006
2
Epoch 5/10
110/110 [==============================] - 27s 243ms/step - loss: 0.004
7
Epoch 6/10
110/110 [==============================] - 28s 256ms/step - loss: 0.003
9
Epoch 7/10
110/110 [==============================] - 28s 254ms/step - loss: 0.003
2
Epoch 8/10
110/110 [==============================] - 27s 249ms/step - loss: 0.002
6
Epoch 9/10
110/110 [==============================] - 28s 253ms/step - loss: 0.002
2
Epoch 10/10
110/110 [==============================] - 28s 257ms/step - loss: 0.002
0
```

```
lstm_predictions = lstm_model.predict(X_test)

lstm_score = r2_score(y_test, lstm_predictions)

print("R^2 Score of LSTM model = ",lstm_score)
```

```
193/193 [==============================] - 4s 14ms/step
R^2 Score of LSTM model =  0.9488749347340549
```

Compare Predictions:

```
plt.figure(figsize=(15,8))
plt.plot(y_test, c="black", linewidth=3, label="Original values")
plt.plot(lstm_predictions, c="red", linewidth=3, label="LSTM predictions")
plt.plot(rnn_predictions, alpha=0.5, c="yellow", linewidth=3, label="RNN
predictions")
plt.axhspan(0, 1, facecolor='gray', alpha=0.3)
plt.legend()
plt.title("Predictions(RNN-LSTM) vs actual data", fontsize=20)
plt.show()
```



Ensemble Method:

An ensemble method in machine learning is a technique that combines the predictions of multiple individual models to produce a more accurate and robust prediction. Voting,Stacking,Boosting,Bagging these are popular ensemble methods.

Load the Dataset:

import pandas as pd

# Replace 'your_dataset.csv' with your dataset file path

data = pd.read_csv('PJMW_hourly.csv')

Data Preprocessing:

data['Year'] = data['Datetime'].dt.year

data['Month'] = data['Datetime'].dt.month

data['Day'] = data['Datetime'].dt.day

data['Hour'] = data['Datetime'].dt.hour

Split the Data:

X = data[['Year', 'Month', 'Day', 'Hour']]

y = data['DOM_MW']

from sklearn.model_selection import train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Choose Ensemble Methods:

```
from sklearn.ensemble import BaggingRegressor, AdaBoostRegressor

from sklearn.tree import DecisionTreeRegressor

# Bagging

bagging_regressor =
BaggingRegressor(base_estimator=DecisionTreeRegressor(), n_estimators=100,
random_state=42)

# AdaBoost

adaboost_regressor =
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(),
n_estimators=100, random_state=42)
```

Train the Models:

```
bagging_regressor.fit(X_train, y_train)

adaboost_regressor.fit(X_train, y_train)
```

Make Predictions:

```
bagging_predictions = bagging_regressor.predict(X_test)

adaboost_predictions = adaboost_regressor.predict(X_test)
```

Evaluate the Models:

```
from sklearn.metrics import mean_squared_error, r2_score

bagging_mse = mean_squared_error(y_test, bagging_predictions)

adaboost_mse = mean_squared_error(y_test, adaboost_predictions)

bagging_r2 = r2_score(y_test, bagging_predictions)

adaboost_r2 = r2_score(y_test, adaboost_predictions)

print(f"Bagging MSE: {bagging_mse}, R-squared: {bagging_r2}")

print(f"AdaBoost MSE: {adaboost_mse}, R-squared: {adaboost_r2}")
```

```
Random Forest MSE: 607039.2920758186, R-squared: 0.3668044417989488
AdaBoost MSE: 757151.3235057434, R-squared: 0.21022434430817016
Bagging MSE: 622329.5988700816, R-squared: 0.3508553023082396
```