# MEASURING ENERGY CONSUMPTION

**NAME:** **S.SURYA-822621104039**

**EMAIL:** **suryajambai144@gmail.com**

Phase-1 Document submission

Energy consumption

Project: AI Measure Energy Consumption

**Abstract:**

Measuring energy consumption is crucial for optimizing energy usage and sustainability in various domains, from industrial processes to residential applications. This abstract outlines a modular approach to measure energy consumption effectively.

**Introduction:**

Energy consumption measurement is vital for assessing the environmental impact and operational efficiency of energyconsuming systems. This abstract presents a modular framework for measuring energy consumption that can be applied across diverse contexts.

1. **Sensor Modules:**

The foundation of our approach lies in sensor modules capable of collecting data on energy usage. These modules encompass various sensor types, such as current sensors, voltage sensors, and smart meters, to capture accurate and real-time energy consumption data.

2. **Data Acquisition:**

The collected data from sensor modules are processed and transmitted to a central data acquisition module. This module is responsible for data aggregation, synchronization, and initial data preprocessing.

3. **Data Analysis:**

To derive actionable insights, data analysis modules utilize advanced algorithms and techniques. These modules identify consumption patterns, anomalies, and trends, enabling users to make informed decisions about energy management.

4. **User Interface:**

A user-friendly interface module provides stakeholders with access to energy consumption data in a comprehensible format. Visualization tools, dashboards, and customizable reports allow users to monitor and analyze energy consumption effortlessly.

6. **Integration:**

Our modular framework supports integration with existing systems and IoT platforms. This ensures compatibility with a wide range of applications, including smart homes, industrial processes, and commercial buildings.

**7. Control and Automation:**

In addition to monitoring, our framework enables control and automation modules. Users can implement energy-saving strategies based on real-time data, such as load shedding, scheduling, and adaptive control.

**8. Security and Privacy:**

We prioritize the security and privacy of energy consumption data through encryption, access control, and compliance with data protection regulations.

## Python 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
```

linkcode

```
df=pd.read_csv("../input/hourly-energy-
consumption/AEP_hourly.csv",index_col='Datetime',p arse_dates=True) 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 |

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

```
                      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]
```

In [4]:

df.shape

(121273, 1) In
[5]:

df.info()

<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 df.describe()

output:

| | 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 |

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

In [8]:

linkcode

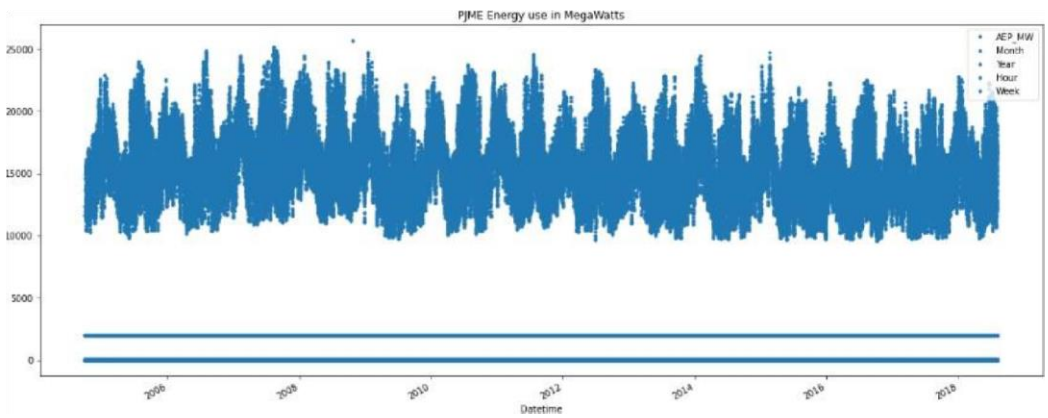
```
# Extract all Data Like Year MOnth Day Time etc
df["Month"] = df.index.month df["Year"] =
df.index.year df["Date"] = df.index.date
df["Hour"] = df.index.hour df["Week"] =
df.index.week df["Day"] = df.index.day_name()
df.head()
```

output:

| | AEP_MW | 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 |

```
df.plot(title="PJME Energy use in MegaWatts",
      figsize=(20, 8), style=".",
```

```
      color=sns.color_palette()[0])

plt.show()
```



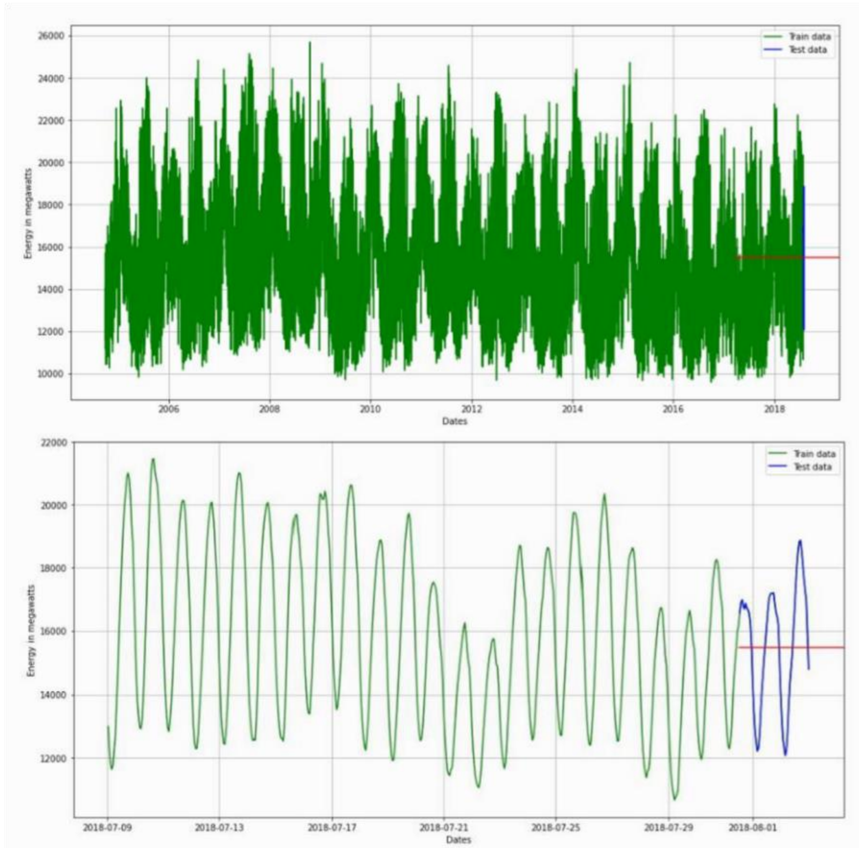

```
from sklearn.metrics import mean_squared_error, mean_absolute_error from
math import sqrt
from sklearn.preprocessing import MinMaxScaler

# 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 In [13]:
linkcode
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)
```

output:

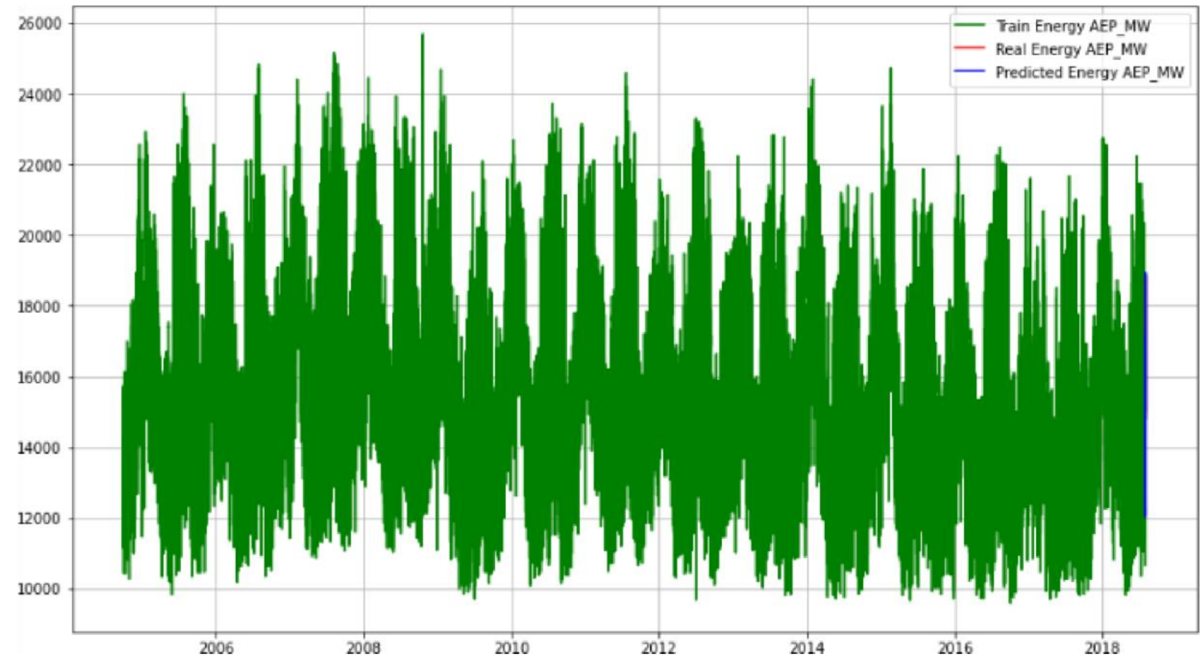
| | t | t+1 | t+5 | t+10 | t+30 |
|---|---|---|---|---|---|
| t | 1.000000 | 0.731161 | 0.345667 | 0.501972 | 0.976223 |
| t+1 | 0.731161 | 1.000000 | 0.630009 | 0.847210 | 0.630007 |
| t+5 | 0.345667 | 0.630009 | 1.000000 | 0.644479 | 0.317277 |
| t+10 | 0.501972 | 0.847210 | 0.644479 | 1.000000 | 0.408315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

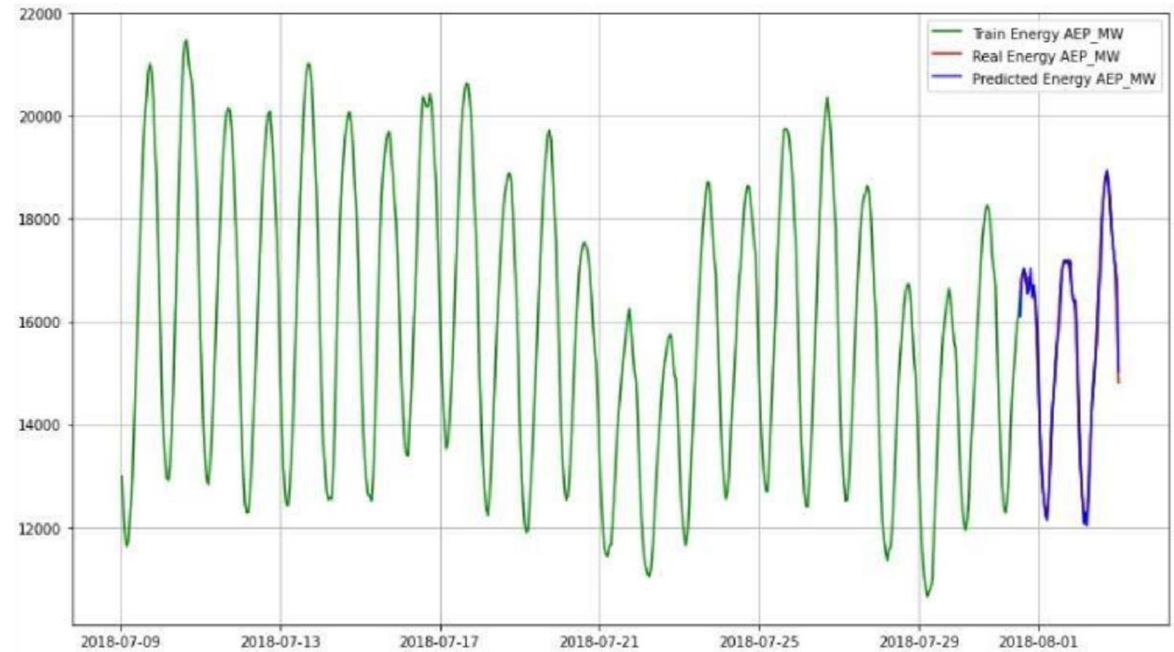
```
mean_value = df['AEP_MW'].mean() # calculation of mean price

plt.figure(figsize=(16,8)) plt.grid(True) plt.xlabel('Dates')
plt.ylabel('Energy in megawatts') plt.plot(df['AEP_MW'], 'green', label='Train data')
plt.plot(test_data['AEP_MW'], 'blue', label='Test data') plt.axhline(y=mean_value, xmin=0.864,
xmax=1, color='red') plt.legend()

plt.figure(figsize=(16,8)) 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.axhline(y=mean_value, xmin=0.864, xmax=1, color='red')
plt.legend()

print('MSE: '+str(mean_squared_error(test_data['AEP_MW'], np.full(len(test_data), mean_valu
e))))
print('MAE: '+str(mean_absolute_error(test_data['AEP_MW'], np.full(len(test_data), mean_val
ue))))
print('RMSE: '+str(sqrt(mean_squared_error(test_data['AEP_MW'], np.full(len(test_data), mean
_value)))))
```




```
import statsmodels.api as sm

#Train Arima Model 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]
#          invert          transformed          prediction
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()

print('MSE:      '+str(mean_squared_error(y,      predictions)))
print('MAE:      '+str(mean_absolute_error(y,      predictions)))
print('RMSE: '+str(sqrt(mean_squared_error(y, predictions))))
```




MSE: 57710.45153428949

MAE: 177.320844006739

RMSE: 240.2299971574938

**Scalability:**

The modular design allows for easy scalability, accommodating both small-scale and large-scale energy monitoring applications.

**Conclusion**:

Measuring energy consumption using a modular approach enhances energy management, sustainability, and costefficiency. By leveraging sensor technology, data analysis, user-friendly interfaces, and integration capabilities, our framework empowers users to make informed decisions for a greener and more efficient future.

This abstract provides an overview of our modular framework for measuring energy consumption, offering a versatile and adaptable solution for various industries and applications.