MEASURE ENERGY CONSUMPTION

PHASE:4 PROJECT SUBMISSION

Analyzing the energy consumption data & Creating visualizations.

MEASURE ENERGY CONSUMPTION

VISUALIZATIONS

- Visualization (graphics), the physical or imagining creation of images, diagrams, or animations to communicate a message.
- Data and information visualization, the practice of creating visual representations of complex data and information.
- Music visualization, animated imagery based on a piece of music.
- Mental image, the experience of images without the relevant external stimuli.

FEATURE ENGINEERING

The process of using domain knowledge to select and transform the most relevant variables from raw data when creating a predictive model using machine learning or statistical modeling.

MODEL TRAINING

Model training is the process of feeding engineered data to a parametrized machine learning algorithm in order to output a model with optimal learned trainable parameters that minimize an objective function.

MODEL EVALUATION

Model evaluation in machine learning is the process of determining a model's performance via a metrics-driven analysis. It can be performed in two ways:

Offline: The model is evaluated after training during experimentation or continuous retraining.

Online: The model is evaluated in production as part of model monitoring.

```
importnumpyasnp

importpandasaspd

importmatplotlib.pyplotasplt

importtensorflowastf

importsklearn.preprocessing

fromsklearn.metricsimportr2_score

fromkeras.layersimportDense,Dropout,SimpleRNN,LS
TM

fromkeras.modelsimportSequential



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

mau=["#F8766D","#D39200","#93AA00","#00BA38","#00C1
9F","#00B9E3","#619CFF","#DB72FB"]
bieudo=AEP.plot(style='.',figsize=(15,5),color=mau[0],title='
AEP')
```

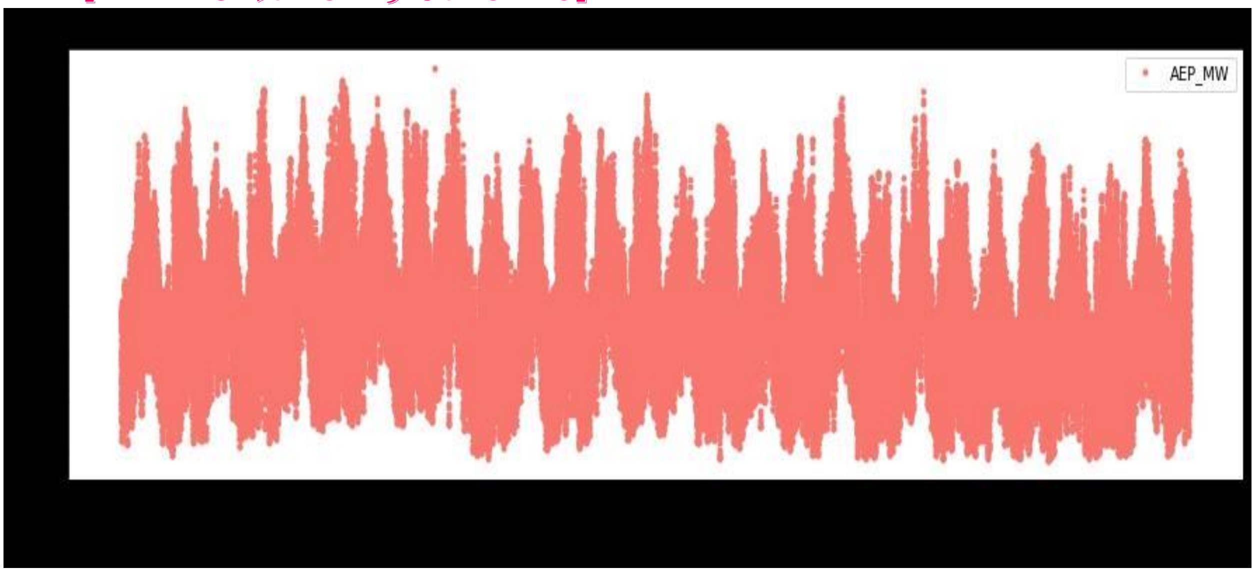
#Data transformation

```
defcreate_features(df,label=None):
df=df.copy()
df['date']=df.index
df['hour']=df['date'].dt.hour
df['dayofweek']=df['date'].dt.dayofweek
df['quarter']=df['date'].dt.quarter
df['month']=df['date'].dt.month
df['year']=df['date'].dt.year
df['dayofyear']=df['date'].dt.dayofyear
df['dayofmonth']=df['date'].dt.day
df['weekofyear']=df['date'].dt.weekofyear

X=df[['hour','dayofweek','quarter','month','year',
'dayofyear','dayofmonth','weekofyear']]
iflabel:
y=df[label]
returnX,y
returnX

X,y=create_features(AEP,label='AEP_MW')
features_and_target=pd.concat([X,y],axis=1)
print(features_and_target)
plt.show()

plt.figure(figsize=(15,6))
data_csv=AEP.dropna()
dataset=data_csv.values
dataset=dataset.astype('float32')
max_value=np.max(dataset)
min_value=np.min(dataset)
scalar=max_value-min_value
```

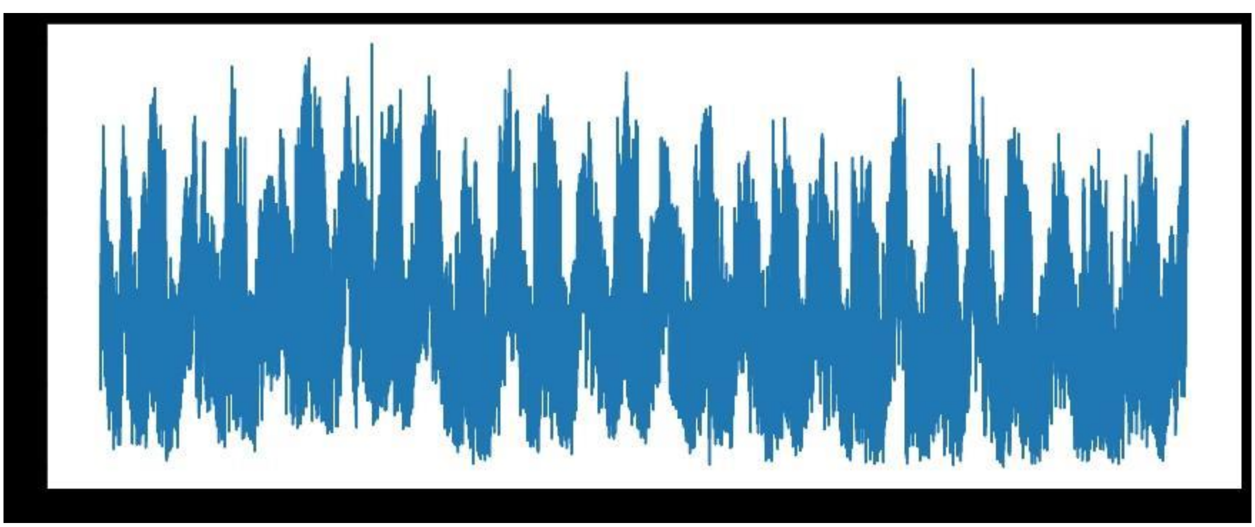
```
dataset=list(map(lambdax:(x-min_value)/scalar,dataset))
plt.plot(dataset)
print(max_value,min_value)
```

```
hour  dayofweek  quarter  month  year  dayofyear  \
Datetime

2004-12-31 01:00:00    1        4        4     12  2004       366
2004-12-31 02:00:00    2        4        4     12  2004       366
2004-12-31 03:00:00    3        4        4     12  2004       366
2004-12-31 04:00:00    4        4        4     12  2004       366
2004-12-31 05:00:00    5        4        4     12  2004       366
...                  ...       ...      ...   ...   ...       ...
2018-01-01 20:00:00   20        0        1      1  2018         1
2018-01-01 21:00:00   21        0        1      1  2018         1
2018-01-01 22:00:00   22        0        1      1  2018         1
2018-01-01 23:00:00   23        0        1      1  2018         1
2018-01-02 00:00:00    0        1        1      1  2018         2

dayofmonth  weekofyear  AEP_MW
Datetime

2004-12-31 01:00:00         31          53  13478.0
2004-12-31 02:00:00         31          53  12865.0
2004-12-31 03:00:00         31          53  12577.0
2004-12-31 04:00:00         31          53  12517.0
2004-12-31 05:00:00         31          53  12670.0
...                        ...         ...      ...
2018-01-01 20:00:00          1           1  21089.0
```

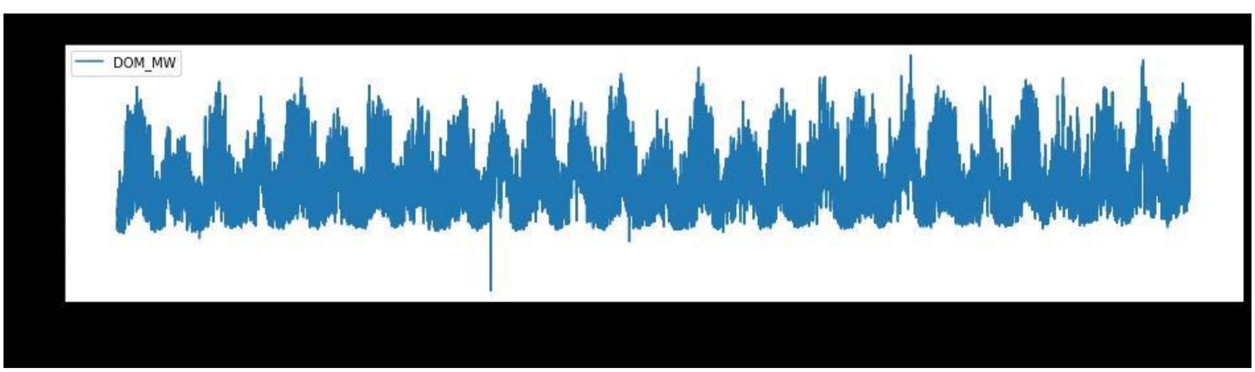
2018-01-01 21:00:00 1 1 20999.0
2018-01-01 22:00:00 1 1 20820.0
2018-01-01 23:00:00 1 1 20415.0
2018-01-02 00:00:00 2 1 19993.0
[121273 rows x 9 columns]



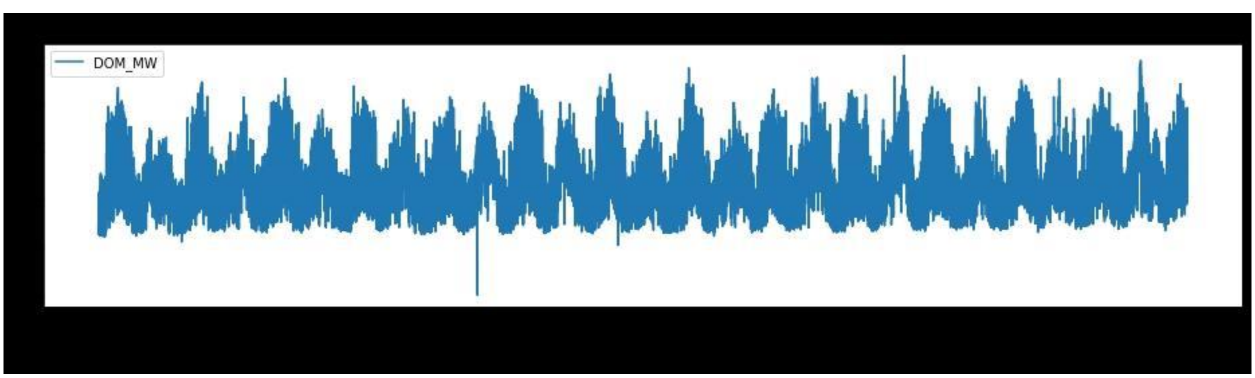
25695.0 9581.0



```
fpath='../input/hourly-energy-consumption/DOM_hour
ly.csv'

#Let's use datetime(2012-10-01 12:00:00,...) as index i
nstead of numbers(0,1,...)
#This will be helpful for further data analysis as we ar
e dealing with time series data
df=pd.read_csv(fpath,index_col='Datetime',parse_dat
es=['Datetime'])
df.head()

#checking missing data
df.isna().sum()

#Data visualization

df.plot(figsize=(16,4),legend=True)

plt.title('DOM hourly power consumption data - BEFO
RE NORMALIZATION')

plt.show()
```



```
defnormalize_data(df):
scaler=sklearn.preprocessing.MinMaxScaler()
df['DOM_MW']=scaler.fit_transform(df['DOM_MW'].val
ues.reshape(-1,1))
returndf

df_norm=normalize_data(df)
df_norm.shape

#Visualize data after normalization

df_norm.plot(figsize=(16,4),legend=True)

plt.title('DOM hourly power consumption data - AFTE
R NORMALIZATION')

plt.show()
```



```
defload_data(stock,seq_len):
X_train=[]
y_train=[]
foriinrange(seq_len,len(stock)):
X_train.append(stock.iloc[i-seq_len:i,0])
y_train.append(stock.iloc[i,0])

# 1 last 6189 days are going to be used in test
X_test=X_train[110000:]
y_test=y_train[110000:]

# 2 first 110000 days are going to be used in training
X_train=X_train[:110000]
y_train=y_train[:110000]

# 3 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)

# 4 reshape data to input into RNN 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]
```

```
#create train, test data

seq_len=20#choose sequence length
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,)

```
#RNN model

rnn_model=Sequential()
rnn_model.add(SimpleRNN(40,activation="tanh",retu
rn_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()
rnn_model.compile(optimizer="adam",loss="MSE")
rnn_model.fit(X_train,y_train,epochs=10,batch_size=1000)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 20, 40)	1680
dropout (Dropout)	(None, 20, 40)	0
simple_rnn_1 (SimpleRNN)	(None, 20, 40)	3240
dropout_1 (Dropout)	(None, 20, 40)	0
simple_rnn_2 (SimpleRNN)	(None, 40)	3240
dropout_2 (Dropout)	(None, 40)	0

```
dense (Dense)            (None, 1)            41
=================================================================
Total params: 8,201
Trainable params: 8,201
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
2022-08-19 16:26:37.061384: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
110/110 [==============================] - 10s 73ms/step - loss: 0.0820
Epoch 2/10
110/110 [==============================] - 9s 81ms/step - loss: 0.0178
Epoch 3/10
110/110 [==============================] - 8s 74ms/step - loss: 0.0096
Epoch 4/10
110/110 [==============================] - 8s 73ms/step - loss: 0.0065
Epoch 5/10
110/110 [==============================] - 8s 74ms/step - loss: 0.0050
Epoch 6/10
110/110 [==============================] - 8s 73ms/step - loss: 0.0040
Epoch 7/10
110/110 [==============================] - 9s 81ms/step - loss: 0.0035
Epoch 8/10
110/110 [==============================] - 8s 73ms/step - loss: 0.0030
Epoch 9/10
110/110 [==============================] - 8s 74ms/step - loss: 0.0027
Epoch 10/10
110/110 [==============================] - 8s 75ms/step - loss: 0.0024
```

```python
rnn_predictions=rnn_model.predict(X_test)
rnn_score=r2_score(y_test,rnn_predictions)
print("R2 Score of RNN model = ",rnn_score)
```

R2 SCORE OF RNN MODEL = 0.9466957722475382

```python
defplot_predictions(test,predicted,title):
```

```
plt.figure(figsize=(16,4))
plt.plot(test,color='blue',label='Actual power consum
ption data')
plt.plot(predicted,alpha=0.7,color='orange',label='Pr
edicted power consumption data')
plt.title(title)
plt.xlabel('Time')
plt.ylabel('Normalized power consumption scale')
plt.legend()
plt.show()
plot_predictions(y_test,rnn_predictions,"Predictions
made by simple RNN model")
```

```
lstm_model=Sequential()
```

```
lstm_model.add(LSTM(40,activation="tanh",return_sequenc
es=True,input_shape=(X_train.shape[1],1)))
lstm_model.add(Dropout(0.15))
lstm_model.add(LSTM(40,activation="tanh",return_sequenc
es=True))
lstm_model.add(Dropout(0.15))
lstm_model.add(LSTM(40,activation="tanh",return_sequenc
es=False))
lstm_model.add(Dropout(0.15))
lstm_model.add(Dense(1))
lstm_model.summary()
lstm_model.compile(optimizer="adam",loss="MSE")
lstm_model.fit(X_train,y_train,epochs=10,batch_size=1000)
```

Model: "sequential_1"

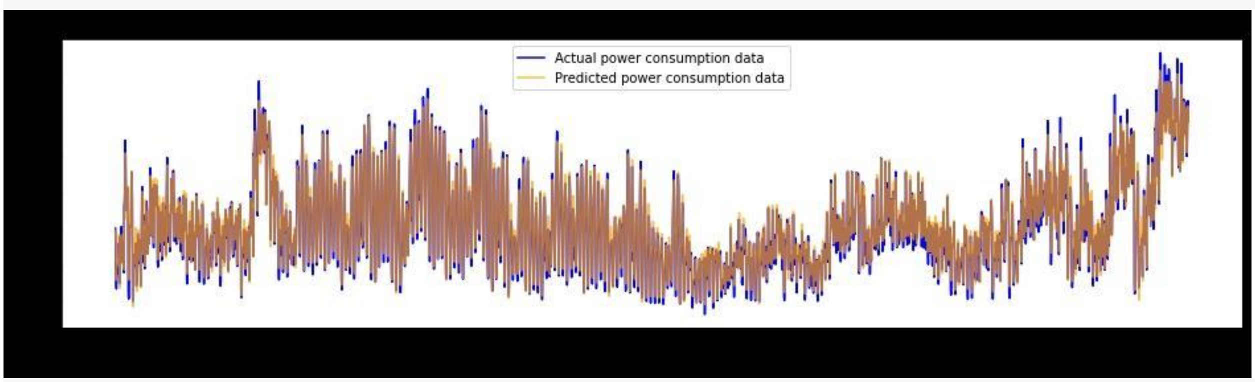
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 20, 40)            6720
_________________________________________________________________
dropout_3 (Dropout)          (None, 20, 40)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 20, 40)            12960
_________________________________________________________________
dropout_4 (Dropout)          (None, 20, 40)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 40)                12960
_________________________________________________________________
dropout_5 (Dropout)          (None, 40)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 41
=================================================================
Total params: 32,681
Trainable params: 32,681
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
110/110 [==============================] - 26s 193ms/step - loss: 0.0211
Epoch 2/10
110/110 [==============================] - 20s 185ms/step - loss: 0.0119
Epoch 3/10
110/110 [==============================] - 21s 191ms/step - loss: 0.0080
Epoch 4/10
110/110 [==============================] - 21s 186ms/step - loss: 0.0047
Epoch 5/10
110/110 [==============================] - 20s 185ms/step - los
s: 0.0037
Epoch 6/10
110/110 [==============================] - 21s 194ms/step - los
s: 0.0030
Epoch 7/10
```

110/110 [==============================] - 20s 185ms/step - loss: 0.0026
Epoch 8/10
110/110 [==============================] - 21s 192ms/step - loss: 0.0022
Epoch 9/10
110/110 [==============================] - 20s 185ms/step - loss: 0.0020
Epoch 10/10
110/110 [==============================] - 21s 191ms/step –

OUT[10]:

<KERAS.CALLBACKS.HISTORY AT 0X7F01D4E38810>

r2 score for the values predicted by the above trained LSTM model

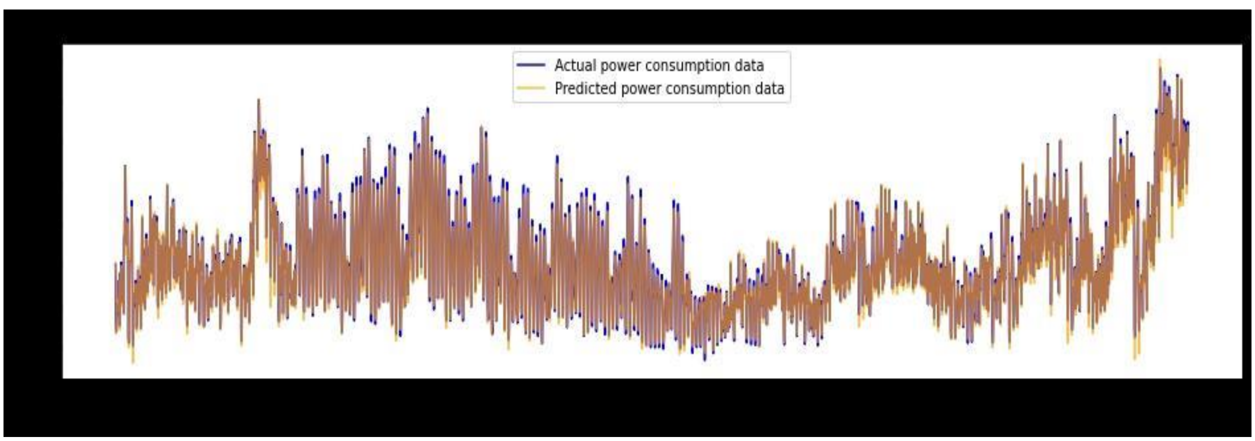



```
lstm_predictions=lstm_model.predict(X_test)
lstm_score=r2_score(y_test,lstm_predictions)
```

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

R^2 SCORE OF LSTM MODEL = **0.94996673239313**

```
#actual values vs predicted values by plotting a graph

plot_predictions(y_test,lstm_predictions,"Predictions made by LSTM model")
```



RNN, LSTM model by plotting data in a single graph

```
plt.figure(figsize=(15,8))
plt.plot(y_test,c="orange",linewidth=3,label="Original values")
plt.plot(lstm_predictions,c="red",linewidth=3,label="LSTM predictions")
plt.plot(rnn_predictions,alpha=0.5,c="blue",linewidth=3,label="RNN predictions")
plt.legend()
plt.title("Predictions vs actual data",fontsize=20)
plt.show()
```