MEASURE ENERGY CONSUMPTION

For

Feature engineering

Feature engineering for measuring energy consumption typically involves creating relevant input variables or features that can help a machine learning model predict or analyze energy usage. Here are some common features to consider:

1. **Historical Usage Data**:
   Include past energy consumption data to identify trends and patterns.

2. **Time-Related Features**:
   Incorporate time and date information, such as hour of the day, day of the week, and season, as energy consumption often varies with time.

3. **Weather Data:**

   Weather conditions can impact energy consumption. Include features like temperature, humidity, wind speed, and precipitation.

4. **Building Characteristics**:

   Information about the building, such as square footage, number of occupants, and insulation quality, can be important features.

5. **Appliance Information**:

   If you're monitoring a specific location, include data about appliances or equipment in use, their power ratings, and usage patterns.

6. **Occupancy Information**:

   Sensors or data on occupancy can help predict energy consumption in commercial or residential buildings.

7. **Economic Factors**:

   Economic indicators, electricity prices, and tariffs can affect energy usage patterns.

8. **Day Type**:
   Create a feature to distinguish between weekdays, weekends, and holidays, as usage patterns often differ.

9. **Energy Source**:
   If multiple energy sources are used (e.g., solar, grid, generator), incorporate data on their contributions.

10. **Outages and Maintenance**:
    Record information on power outages and maintenance periods.

11. **Social Factors**:
    In some cases, demographic data and lifestyle information may be relevant.

12. **Environmental Factors**:
    Include data related to environmental initiatives or events that could influence energy consumption.

13. **Energy Efficiency Measures**:
    Indicators of energy-saving measures, such as the use of energy-efficient appliances, insulation upgrades, or smart thermostats.

Remember to preprocess and scale your features appropriately, and consider using techniques like dimensionality reduction or feature selection to improve model performance. The choice of features will depend on the specific context and goals of your energy consumption prediction or analysis task.

Training model

To train a model for measuring energy consumption, you can use various machine learning algorithms depending on the complexity of your problem and the characteristics of your data. Here are some common algorithms for this task:

1. **Linear Regression**:

- Linear regression is a simple and interpretable model that can be used when there is a linear relationship between the input features and energy consumption.

2. **Decision Trees**:

- Decision trees are useful for capturing non-linear relationships in the data and can handle both numerical and categorical features. Ensemble methods like Random Forests and Gradient Boosting with decision trees can improve accuracy.

3. **Neural Networks**

- Deep learning techniques, particularly feedforward neural networks, can capture complex patterns in the data. They are suitable for large datasets and high-dimensional feature spaces.

4. **Time Series Models**:

- For time-dependent data, models like ARIMA (AutoRegressive Integrated Moving Average) or LSTM (Long Short-Term Memory) networks can be effective in predicting energy consumption over time.

5. **Support Vector Machines (SVM):**

- SVMs can be used for regression tasks and are effective when there is a clear separation between different levels of energy consumption.

6. **K-Nearest Neighbors (KNN):**

- KNN is a non-parametric algorithm that can be used for regression. It makes predictions based on the similarity of data points in the feature space.

7. **XGBoost and LightGBM**:

- These gradient boosting libraries are known for their high performance in regression tasks. They can handle complex relationships in the data and work well with tabular data.

8. **Time-Series Forecasting Models**

- If your data has a strong time-dependent component, consider using time-series forecasting models like Holt-Winters, Prophet, or Exponential Smoothing.

The choice of the model depends on the nature of your dataset, the volume of data, and the specific requirements of your energy consumption prediction task. It's often a good practice to experiment with multiple models and compare their performance using appropriate evaluation metrics (e.g., MAE, MSE, RMSE, R-squared) on a validation dataset.

Additionally, consider hyperparameter tuning, cross-validation, and feature engineering to optimize the performance of your chosen model.

Evaluation

To evaluate a model for measuring energy consumption, you can use various evaluation metrics to assess its performance. The choice of metrics depends on the nature of your regression problem. Here are some commonly used evaluation metrics:

1**. Mean Absolute Error (MAE):**

- MAE measures the average absolute difference between the predicted values and the actual values. It provides a straightforward understanding of the model's prediction error.

- Formula: MAE = (1/n) ∑ |predicted - actual|

2. **Mean Squared Error (MSE)**:

- MSE measures the average squared difference between the predicted and actual values. It emphasizes larger errors more than MAE.

- Formula: MSE = (1/n) ∑ (predicted - actual)^2

3. Root Mean Squared Error (RMSE)

- RMSE is the square root of the MSE and provides an easily interpretable metric in the same units as the target variable.

- Formula: RMSE = √MSE

4. **R-squared (R2) Score**:

- R2 measures the proportion of the variance in the dependent variable (energy consumption) that is predictable from the independent variables (features). It ranges from 0 to 1, with 1 indicating a perfect fit.

- Formula: R2 = 1 - (MSE(model) / MSE(mean))

5. **Mean Absolute Percentage Error (MAPE)**

- MAPE expresses prediction errors as a percentage relative to the actual values. It's useful when you want to understand the model's performance in a more interpretable way.

- Formula: MAPE = (1/n) ∑ (|actual - predicted| / actual) * 100%

6. Coefficient of Determination (COD):

- COD is an alternative to R-squared that provides information about the proportion of the variance in the dependent variable that is accounted for by the independent variables.

- Formula: COD = 1 - (MSE(model) / Var(actual))

7**. Adjusted R-squared**

- Adjusted R-squared is a modified version of R-squared that adjusts for the number of predictors in the model, providing a balance between model complexity and goodness of fit.

When evaluating your model, it's essential to consider the specific context of your energy consumption prediction problem and select the metric(s) that align with your objectives. Additionally, you can use techniques such as cross-validation to ensure that your model's performance is consistent and not overfitting the data.

DATA SET LINK:

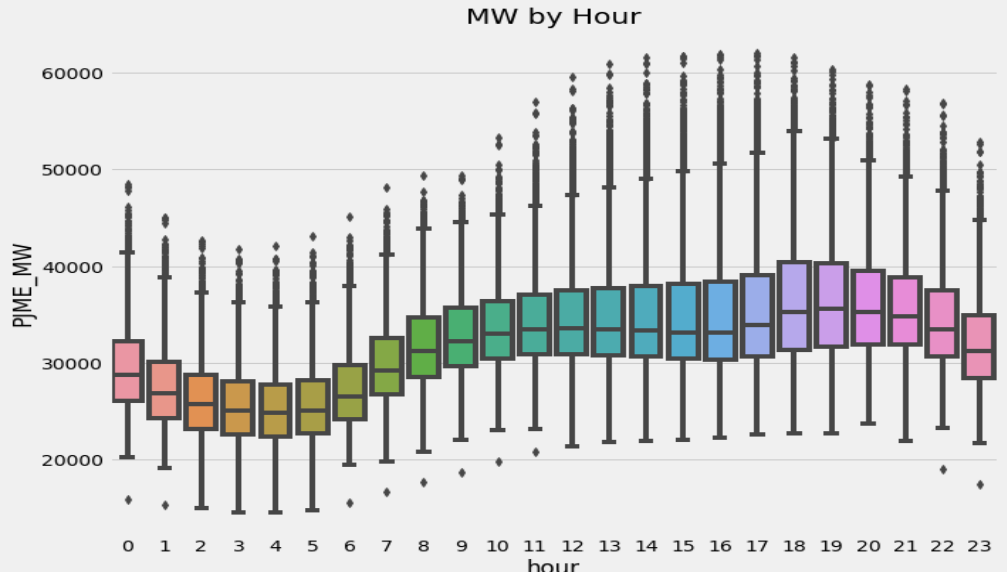
https://www.kaggle.com/datasets/ro bikscube/hourly-energy-consumption

Feature engineering

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error
color_pal = sns.color_palette()
plt.style.use('fivethirtyeight')
reference: https://engineering.99x.io/time-series-forecasting-in-machine-learning-3972f7a7a467

df = pd.read_csv('../input/hourly-energy-consumption/PJME_hourly.csv')
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)

df.plot(style='.',
        figsize=(15, 5),
        color=color_pal[0],
        title='PJME Energy Use in MW')
plt.show()

def create_features(df):
    """
    Create time series features based on time series index.
    """
    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 our Feature / Target Relationship
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='hour', y='PJME_MW')
ax.set_title('MW by Hour')
plt.show()
```



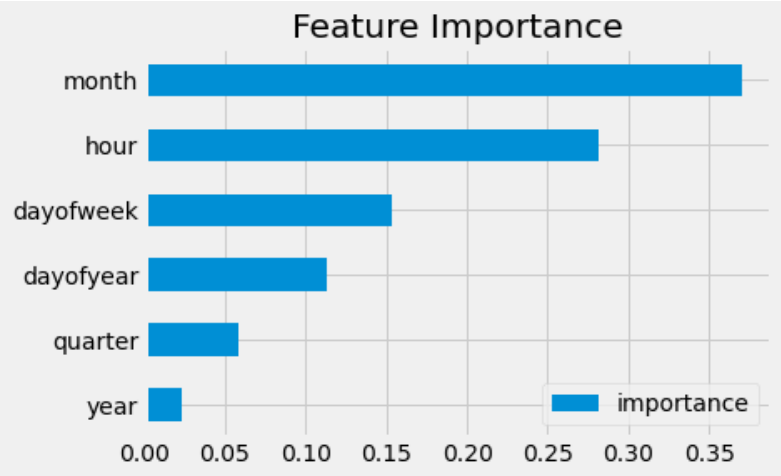

```
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='month', y='PJME_MW', palette='Blues')
ax.set_title('MW by Month')
plt.show()
```

MODEL :

```
Create our Model
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]
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)
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=50, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
```

```
        importance_type=None, interaction_constraints='',
        learning_rate=0.01, max_bin=256, max_cat_to_onehot=4,
        max_delta_step=0, max_depth=3, max_leaves=0, min_child_weight=1,
        missing=nan, monotone_constraints='()', n_estimators=1000,
        n_jobs=0, num_parallel_tree=1, objective='reg:linear',
        predictor='auto', random_state=0, reg_alpha=0, ...)
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()
```

https://www.kaggle.com/code/robikscube/time-series-forecasting-with-machine-learning-yt



EVALUATION

MODELLING AND EVALUATION
#TRANSFORM THE GLOBAL_ACTIVE_POWER COLUMN OF THE DATA DATAFRAME INTO A NUMPY ARRAY OF FLOAT VALUES

DATASET = DATA.GLOBAL_ACTIVE_POWER.VALUES.ASTYPE('FLOAT32')
#RESHAPE THE NUMPY ARRAY INTO A 2D ARRAY WITH 1 COLUMN

DATASET = NP.RESHAPE(DATASET, (-1, 1))
#CREATE AN INSTANCE OF THE MINMAXSCALER CLASS TO SCALE THE VALUES BETWEEN 0 AND 1

SCALER = MINMAXSCALER(FEATURE_RANGE=(0, 1))
#FIT THE MINMAXSCALER TO THE TRANSFORMED DATA AND TRANSFORM THE VALUES

DATASET = SCALER.FIT_TRANSFORM(DATASET)
#SPLIT THE TRANSFORMED DATA INTO A TRAINING SET (80%) AND A TEST SET (20%)

TRAIN_SIZE = INT(LEN(DATASET) * 0.80)
TEST_SIZE = LEN(DATASET) - TRAIN_SIZE
TRAIN, TEST = DATASET[0:TRAIN_SIZE,:], DATASET[**TRAIN_SIZE:LEN(DATASET),:]**

```
def create_dataset(dataset, look_back=1):

    X, Y = [], []

    For I in range(len(dataset)-look_back-1):

        A = dataset[i☹i+look_back), 0]
```

```
        X.append(a)

        Y.append(dataset[I + look_back, 0])

    Return np.array(X), np.array(Y)

X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

Evaluation

```
# make predictions
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
# invert predictions
train_predict = scaler.inverse_transform(train_predict)
Y_train = scaler.inverse_transform([Y_train])
test_predict = scaler.inverse_transform(test_predict)
Y_test = scaler.inverse_transform([Y_test])

print('Train Mean Absolute Error:', mean_absolute_error(Y_train[0], train_predict[:,0]))
print('Train Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_train[0], train_predict[:,0])))
print('Test Mean Absolute Error:', mean_absolute_error(Y_test[0], test_predict[:,0]))
print('Test Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_test[0], test_predict[:,0])))
```

Output:

Train Mean Absolute Error: 0.10347279920086673

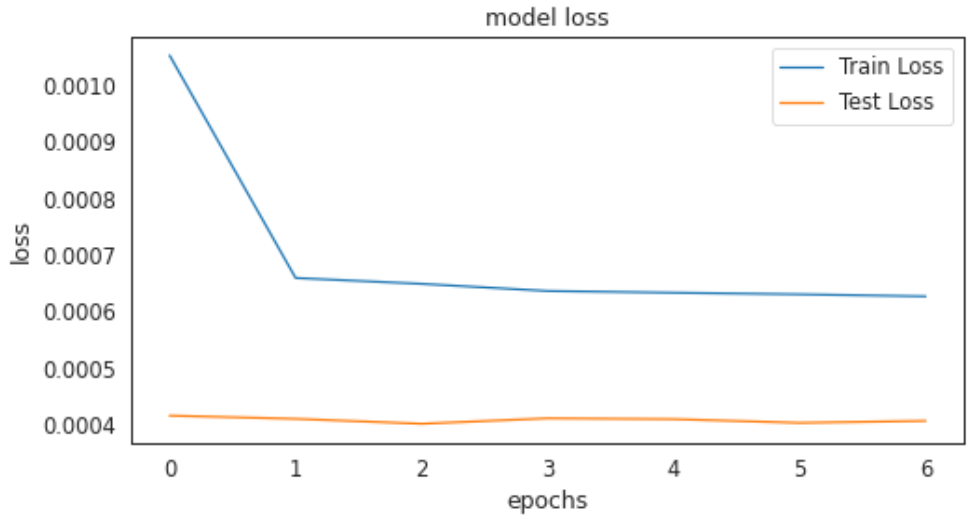
Train Root Mean Squared Error: 0.2696353979332258

Test Mean Absolute Error: 0.09029906112109547

Test Root Mean Squared Error: 0.22271064457180156

```
plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(loc='upper right')
plt.show();

aa=[x for x in range(200)]
# Creating a figure object with desired figure size
plt.figure(figsize=(20,6))

# Plotting the actual values in blue with a dot marker
```

plt.plot(aa, Y_test[0][:200], marker='.', label="actual", color='purple')



Plotting the predicted values in green with a solid line

plt.plot(aa, test_predict[:,0][:200], '-', label="prediction", color='red')

Removing the top spines

sns.despine(top=True)

Adjusting the subplot location

plt.subplots_adjust(left=0.07)

Labeling the y-axis

plt.ylabel('Global_active_power', size=14)

Labeling the x-axis

plt.xlabel('Time step', size=14)

Adding a legend with font size of 15

plt.legend(fontsize=16)# Display the plotplt.show()