Measure energy consumption using machine learning model

Phase 5 Documentation

Team members: Swastina S, Vasundhra J, Surya L, Vimala Devi S, Sathya P, Sowmiya S.

Problem thinking

The problem at hands is to create an automated system that measures the energy consumption analyzes the data and provide visualisations for inform the decision making. This solution AIIMS to enhance efficiency ,accuracy , and the each of understanding in managing energy consumption across various sectors.

Design thinking:

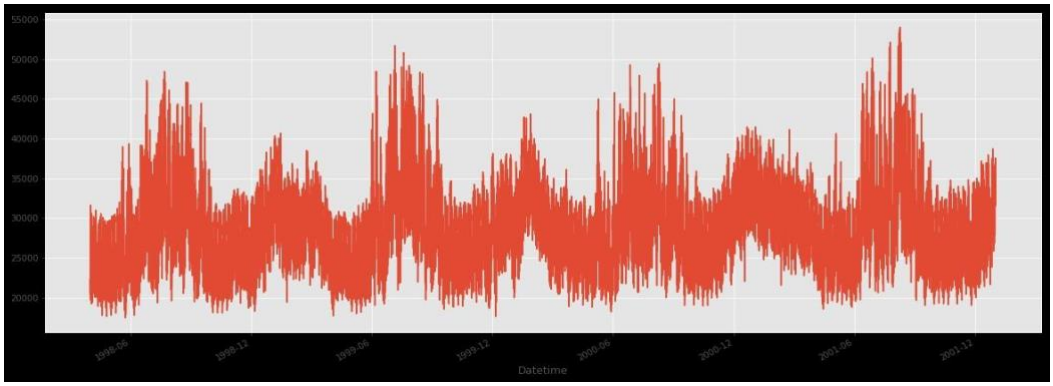
- Data source
- Data pre processing
- Feature extraction
- Model development
- Visualisation
- Automation

It proposes a data driven modelling method for building energy consumption protection and a place it to a to actual commercial buildings. Time series analysis is adopted as a main methodology to produce the data driven model based on monthly actual energy consumption data. This model can be used to predict a building future energy consumption after being modified and verified.

About dataset:

PJM Hourly Energy Consumption Data

PJM Interconnection LLC (PJM) is a regional transmission organization (RTO) in the United States. It is part of the Eastern Interconnection grid operating an electric transmission system serving all or parts of Delaware, Illinois, Indiana, Kentucky, Maryland, Michigan, New Jersey,



North Carolina, Ohio, Pennsylvania, Tennessee, Virginia, West Virginia, and the District of Columbia.

The hourly power consumption data comes from PJM's website and are in megawatts (MW).

The regions have changed over the years so data may only appear for certain dates per region.

Given dataset

```
 # This Python 3 environment comes with many helpful analytics libraries instal
led
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/do
cker-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 get
s 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
```

```
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')
```

In [4]:

```
df = pd.read_csv('../input/hourly-energy-consumption/PJME_hourly.csv')
```

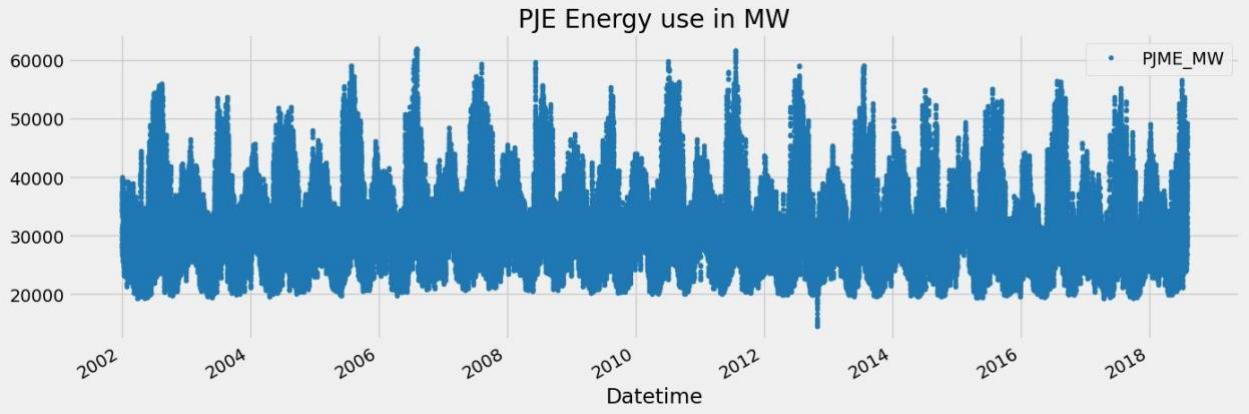
```
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)
```

<iframe src="https://www.kaggle.com/embed/agusabdulrahman/time-series-forecasting-with-machine-learning?cellIds=5&kernelSessionId=149170869" height="300" style="margin: 0 auto; width: 100%; max-width: 950px;" frameborder="0" scrolling="auto" title="Time Series Forecasting with Machine Learning"></iframe

Output

Datetime	PJME_MW
2018-01-01 20:00:00	44282.0
2018-01-01 21:00:00	43751.0
2018-01-01 22:00:00	42402.0
2018-01-01 23:00:00	38608.0

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

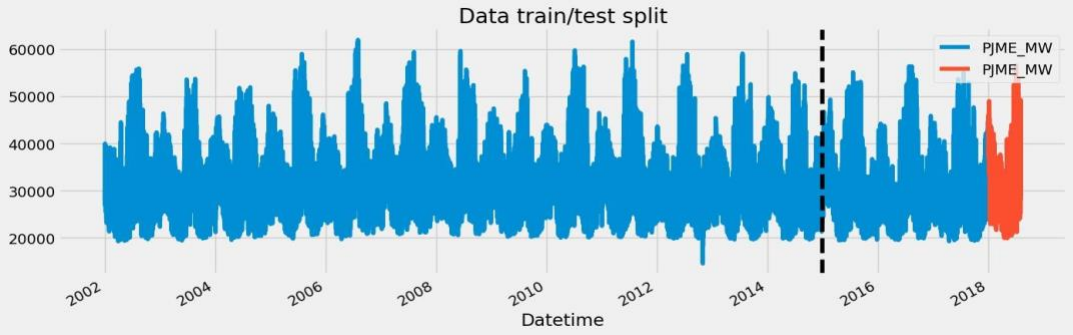


Train / Test Split

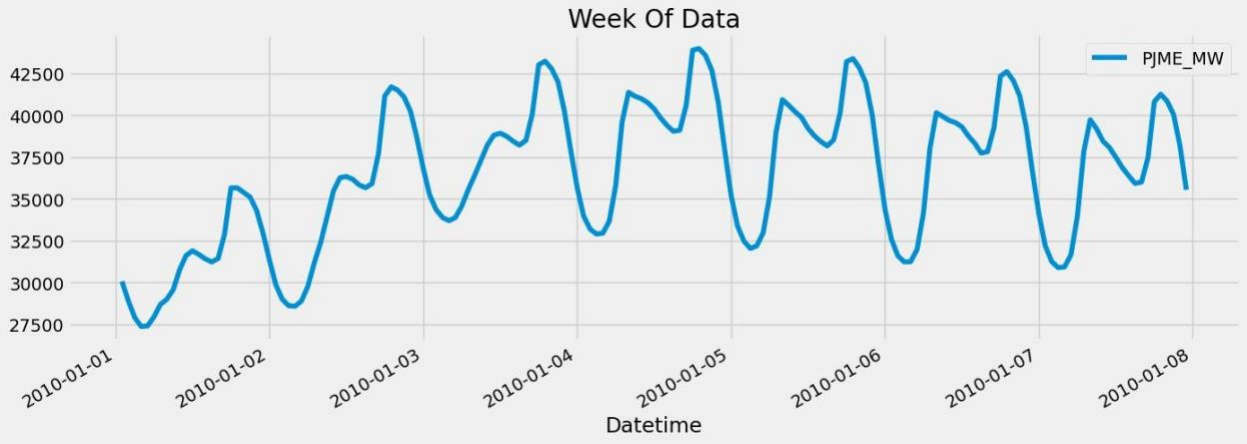
In [7]:

```
train = df.loc[df.index < '01-01-2018']
test = df.loc[df.index >= '01-01-2018']

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='--')
```

```
plt.show()
```




```
df.loc[(df.index >'01-01-2010') & (df.index < '01-08-2010')] \
     .plot(figsize=(15, 5), title='Week Of Data')
plt.show()
```



Feature create

In [9]:

```
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)
```

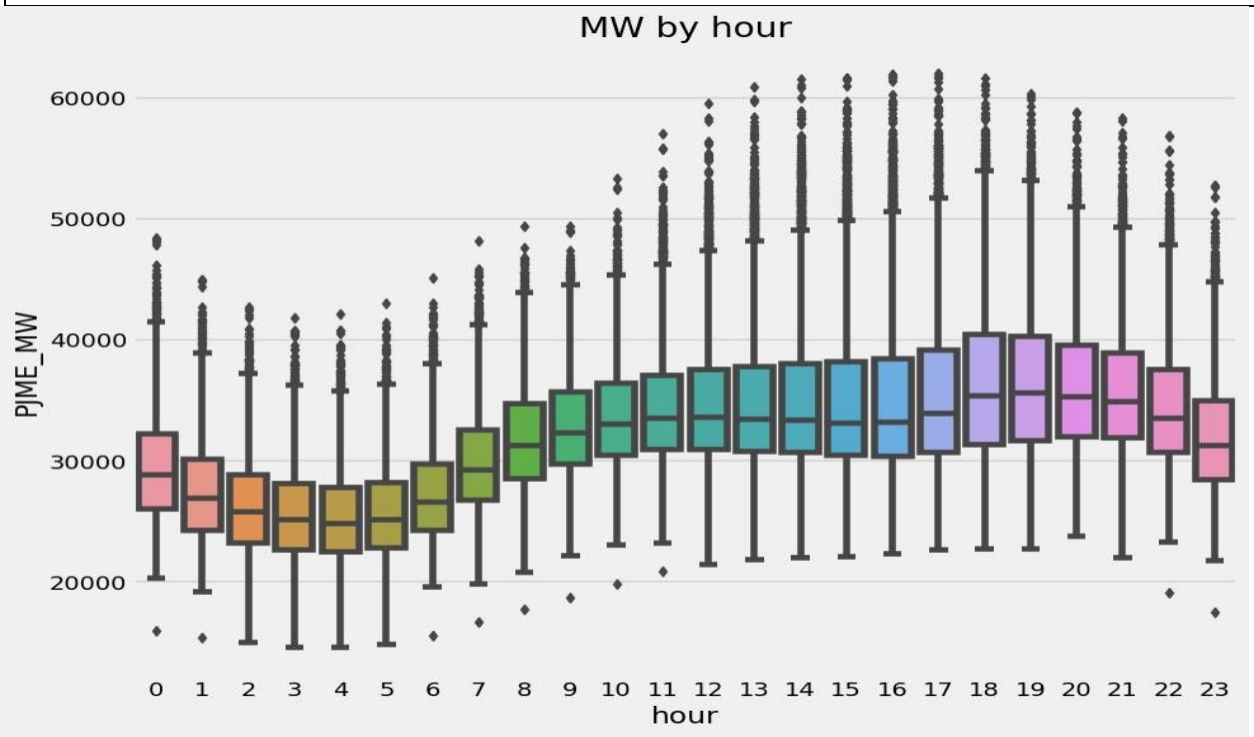
In [ ]:

```
```

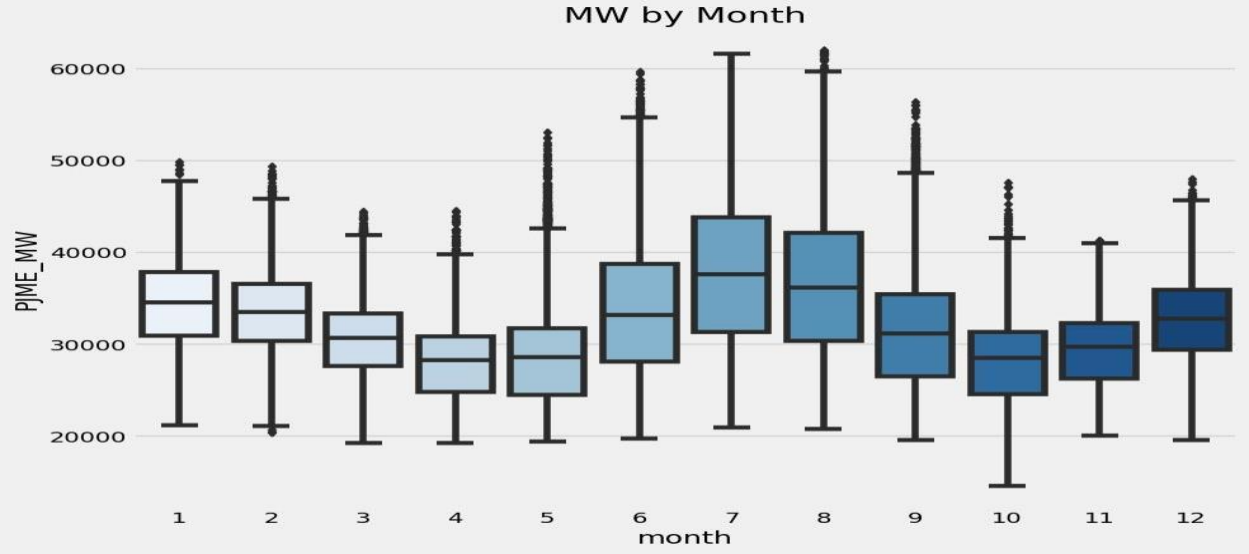
Visualize our Feature / Target Relationship

In [10]:

```
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()
```



Create model

In [12]:

linkcode

```
train = create_features(train)
test = create_features(test)

FEATURE = ['dayofyear', 'hour', 'dayofweek', 'quarter', 'month', 'year']
TARGET = 'PJME_MW'

X_train = train[FEATURE]
y_train = train[TARGET]

X_test = test[FEATURE]
y_test = test[TARGET]
```

In [13]:

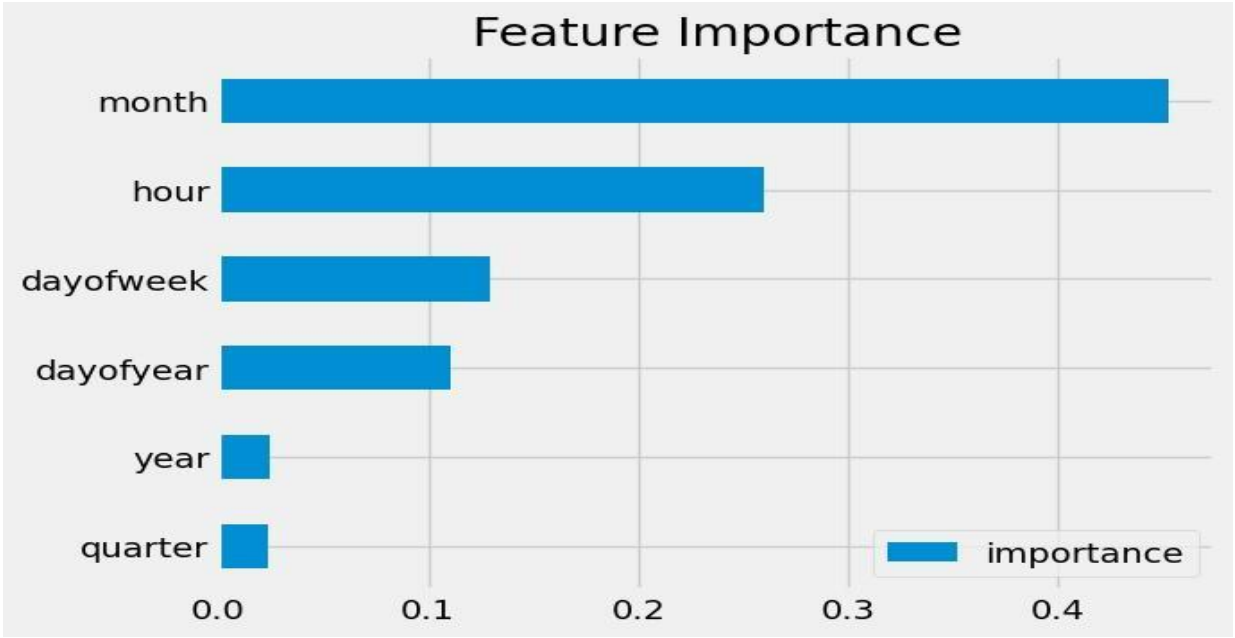
```
reg = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
                       n_estimators = 1000,
                       early_stoping_rounds=50,
                       objective='reg:linear',
                       max_depth=3,
                       learning_rate=0.01)
reg.fit(X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
         verbose=100)
```

XGBRegressor

```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
```

```
             colsample_bytree=None, early_stoping_rounds=50,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=None, gpu_id=Non
e,
             grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.01, max_bin=No
ne,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=Non
e,
             n_estimators=1000, n_jobs=None, num_parallel_tree=None,
             objective='reg:linear', ...)
```

```
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()
```

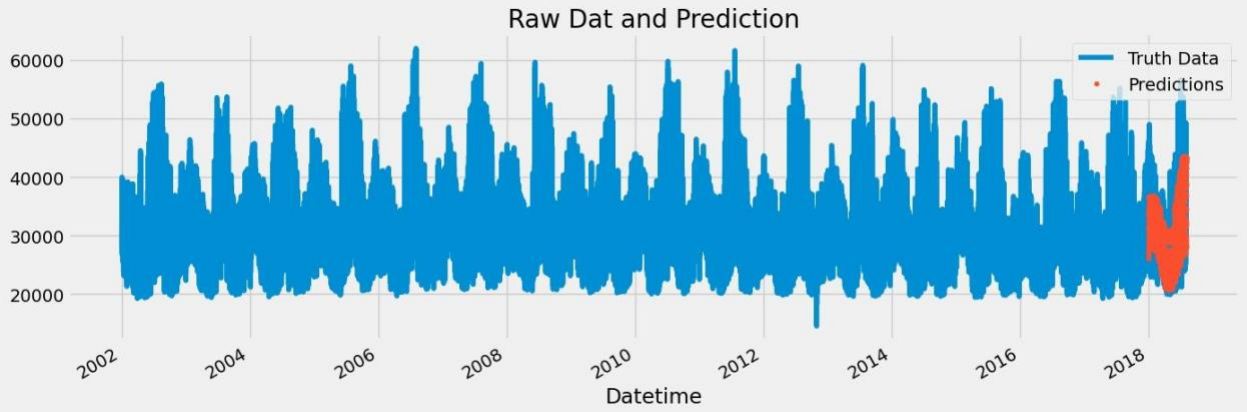


Forcast on test

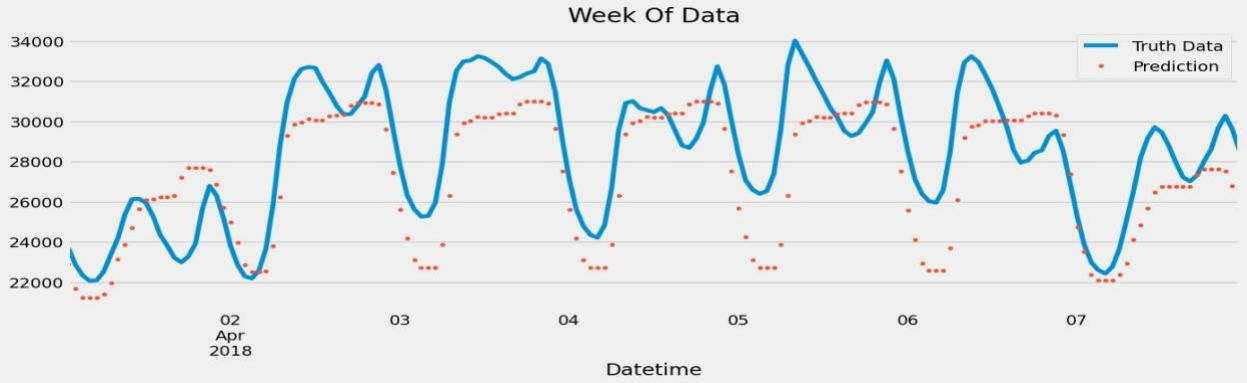
In [15]:

```
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()
```




```
ax = df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]['PJME_MW']
 \
    .plot(figsize=(15, 5), title='Week Of Data')
df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]['prediction'] \
    .plot(style='.')
plt.legend(['Truth Data','Prediction'])
plt.show()
```



Score (RSME)

In [17]:

```
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: 3874.43In [ ]:
```

Calculate Error

- Look at the worst and best predicted days

In [18]:

```
test['error'] = np.abs(test[TARGET] - test['prediction'])
test['date'] = test.index.date
test.groupby(['date'])['error'].mean().sort_valu
```

Output

```
date
2018-01-06    11831.736328
2018-01-07    11022.930827
2018-01-05    10573.553874
2018-07-02    10008.177816
2018-07-03     9773.810710
2018-01-01     9767.635742
2018-01-02     9130.083659
2018-07-01     8649.950846
2018-01-04     7491.049072
2018-06-18     7465.802734
Name: error, dtype: float64
```

Innovative methods:

```
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')
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()
```

```
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='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()

df.loc[(df.index > '01-01-2010') & (df.index < '01-08-2010')] \
.plot(figsize=(15, 5), title='Week Of Data')
plt.show()

Feature Creation

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['weekofmonth'] = 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()

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
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=50,
```

```
             enable_categorical=False, eval_metric=None, feature_types=No
ne,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=N
one,
             interaction_constraints=None, learning_rate=0.01, max_bin=No
ne,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=Non
e,
             n_estimators=1000, n_jobs=None, num_parallel_tree=None,
             objective='reg:linear', predictor=None, ...)
Feature 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()

Forecast on Test

test['prediction'] = reg.predict(x_test)
df = df.merge(test[['prediction']], how= 'left', left_index=True, right_i
ndex=True)
ax = df[['PJME_MW']].plot(figsize=(15, 5))
df['prediction'].plot(ax=ax, style='.')
plt.legend(['Truth Data', 'Predication'])
ax.set_title('Raw Dat and Prediction')
plt.show()

ax = df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]['PJME_
MW'] \
    .plot(figsize=(15, 5), title='Week Of Data')
df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]['prediction
'] \
    .plot(style='.')
```

```
plt.legend(['Truth Data','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
Calculate Error Look at the worst and best predicted days


test['error'] = np.abs(test[TARGET] - test['prediction'])
test['date'] = test.index.date
  test.groupby(['date'])['error'].mean().sort_values(ascending=False).hea
  d(10)
```

Output

date	
2016-08-13	12839.597087
2016-08-14	12780.209961
2016-09-10	11356.302979
2015-02-20	10965.982259
2016-09-09	10864.954834
2018-01-06	10506.845622
2016-08-12	10124.051595
2015-02-21	9881.803711
2015-02-16	9781.552246
2018-01-07	9739.144206

Overview of the process

Prepare the data This includes cleaning data , removing outliers the handling missing values.

Perform feature selection This can be done using a variety of methods such as correlation analysis information gain and recursive feature elimination.

Train the model There are many machine learning algorithms that can be used for measure energy consumption . Some popular choices include linear regression random forest and gradient boosting machines.

Evaluate with the model This can be done by calculating the main square error or the root means square error of the models prediction on held out test set.

Deploy the model Once the model has been evaluated and found to be performing well it can be deployed

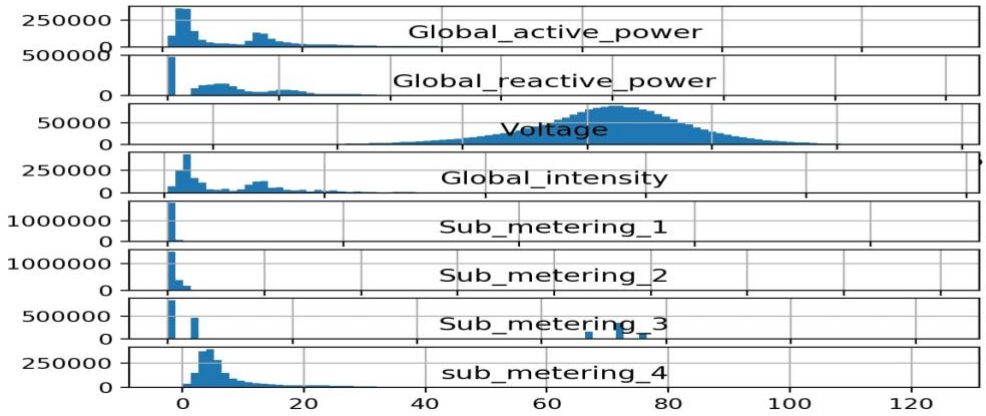
to protection so that it can be used to predicts the energy consumption for an hour.

Approaches to the model development part 1

```
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')
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()

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='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()

df.loc[(df.index > '01-01-2010') & (df.index < '01-08-2010')] \
.plot(figsize=(15, 5), title='Week Of Data')
plt.show()

Feature Creation
```

```
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['weekofmonth'] = 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()

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
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
          colsample_bylevel=None, colsample_bynode=None,
          colsample_bytree=None, early_stopping_rounds=50,
          enable_categorical=False, eval_metric=None, feature_types=None,
          gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
          interaction_constraints=None, learning_rate=0.01, max_bin=None,
          max_cat_threshold=None, max_cat_to_onehot=None,
          max_delta_step=None, max_depth=3, max_leaves=None,
          min_child_weight=None, missing=nan, monotone_constraints=None,
          n_estimators=1000, n_jobs=None, num_parallel_tree=None,
          objective='reg:linear', predictor=None, ...)
Feature 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()

Forecast on Test

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', 'Predication'])
ax.set_title('Raw Dat and Prediction')
plt.show()
```




```
ax = df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]['PJME_MW'] \
   .plot(figsize=(15, 5), title='Week Of Data')
df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]['prediction'] \
   .plot(style='.')
plt.legend(['Truth Data','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

Mean square error = 6758395. 805638

R Squared = 0.002701

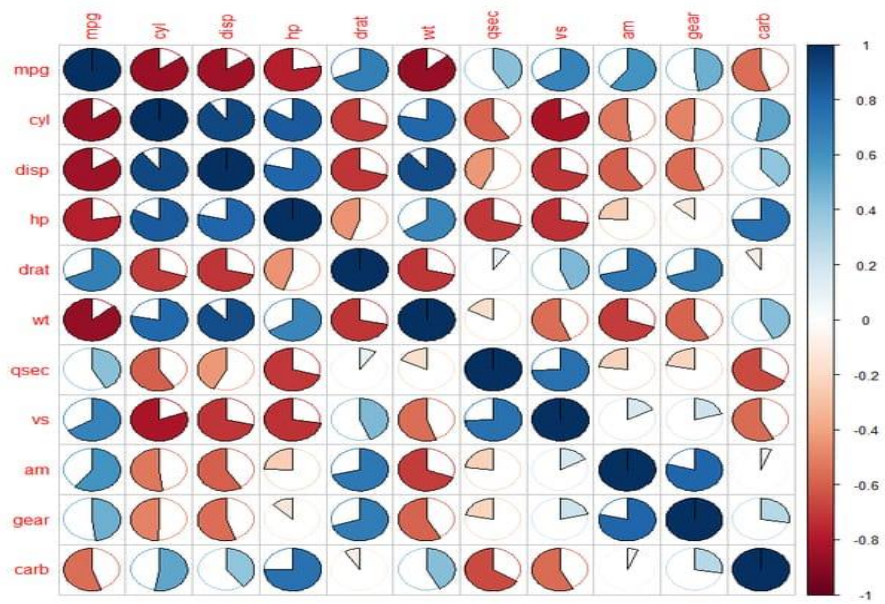
Split the data

```
# Importing the turicreate Library
import turicreate as tc
# Now Loading the data
data=tc.Sframe(“data.csv”)
 # Turicreate has a library named as random
# split that will the data randomly among the train,test
# Dev will be part of test set and we will split that data later.
Train_data_set,test_data=data.random_split(.8,
```

Feature scaling

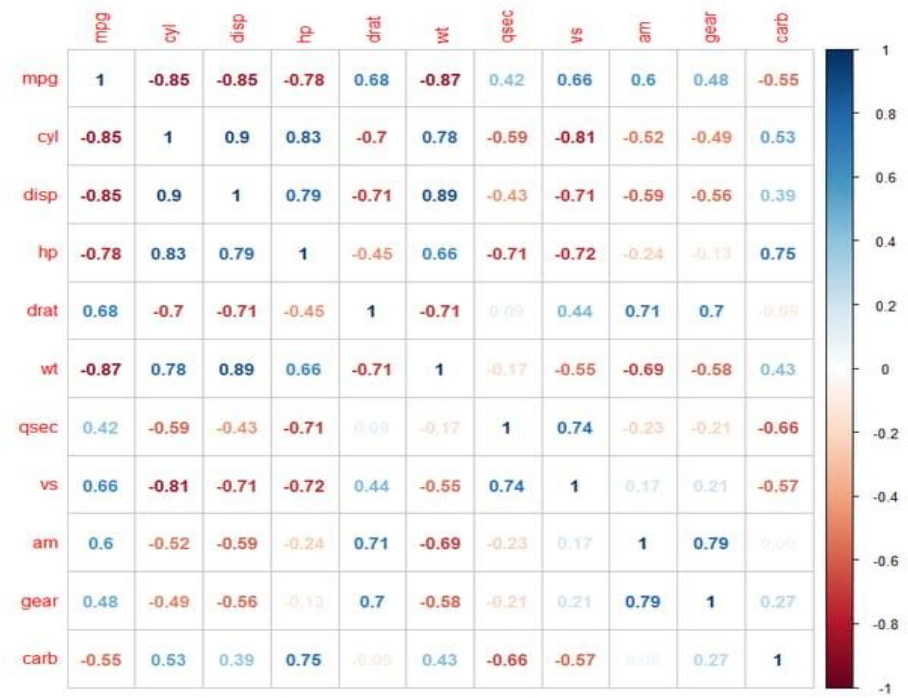
```
From sklearn.preprocessing import MinMaxScaler
```

```
Scaler = MinMaxScaler()
Scaled_data = scaler.fit_transform(df)
Scaled_df = pd.DataFrame(scaled_data,
Columns=df.columns
Scaled_df.head()
```




```
Loading the dataset
Input:
: monthly line plots
From pandas import read_csv
From matplotlib import pyplot
# load the new file
Dataset = read_csv('household_power_consumption.csv', header=0, infer_datetime_format=True, parse_dates=['datetime'], index_col=['datetime'])
# plot active power for each year
Months = [x for x in range(1, 13)]
Pyplot.figure()
For I in range(len(months)):
        # prepare subplot
        Ax = pyplot.subplot(len(months), 1, i+1)
        # determine the month to plot
        Month = '2007-' + str(months[i])
        # get all observations for the month
```

```
                Result = dataset[month]
                # plot the active power for the month
                Pyplot.plot(result['Global_active_power'])
                # add a title to the subplot
                Pyplot.title(month, y=0, loc='left')
Pyplot.show()
```

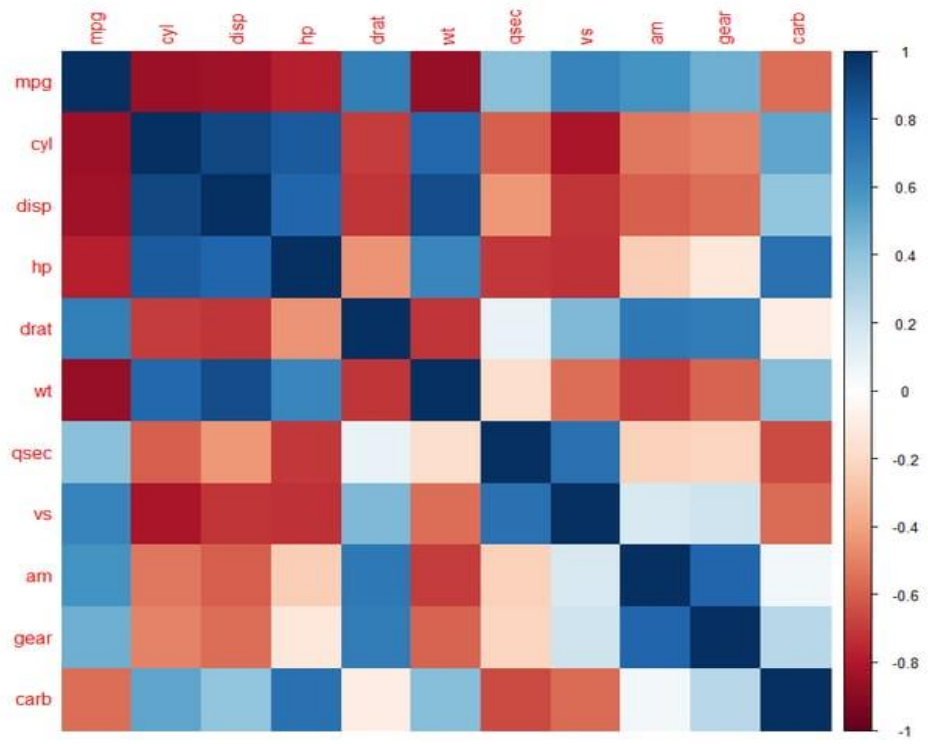



```
2016-08-13    12839.597087
2016-08-14    12780.209961
2016-09-10    11356.302979
2015-02-20    10965.982259
2016-09-09    10864.954834
2018-01-06    10506.845622
2016-08-12    10124.051595
2015-02-21     9881.803711
2015-02-16     9781.552246
2018-01-07     9739.144206
```

Visual correlation and pre processing data set

```
Import pandas
import matplotlib.pyplot as plt
```

```
data = 'iris_df.csv'
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pandas.read_csv(data, names=names)
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()Input:
Energy.count()
Output:
day          829
energy_sum    829
LCLid         829
dtype: int64
```



Visualization and correlation

```
# Chronogram in R
# including the required packages
library(corrplot)
head(mtcars)
Head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
```

Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4

Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1

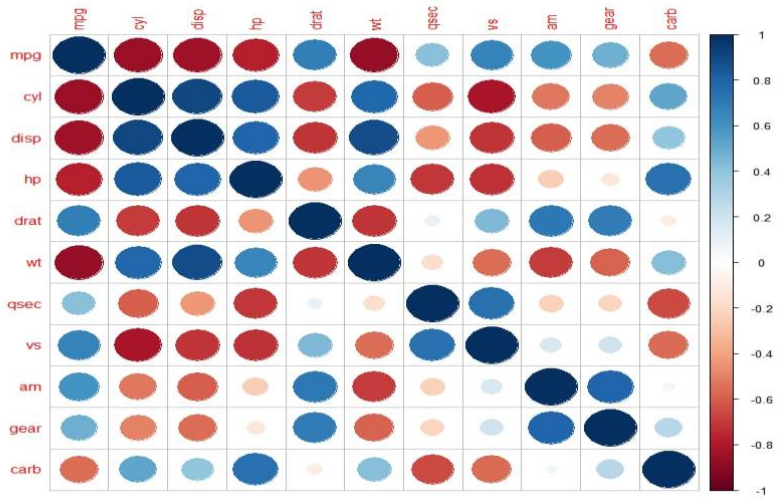
Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1

Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2

Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1

[11/5, 2:31 PM] +91 93845 03699: # Correlogram in R

```
# required packages
library(corrplot)
head(mtcars)
#correlation matrix
M<-cor(mtcars)
head(round(M,2))
#visualizing correlogram
```




```
#as circle
corrplot(M, method="circle")
#as pie
corrplot(M, method="pie")
# as colour
corrplot(M, method="color")
# as number
correlate(M, method="number")
```

[11/5, 2:31 PM] +91 93845 03699: Import libraries:

```
import pandas as pd
from sklearn.pre-processing import Imputer
Read the dataset:
dataset = pd.read_csv('Data.csv')
```

Pre-process the data:

X = dataset.iloc[:, :-1].valuesimputer = imputer.fit(X[:, 1:3])

PROCEDURE

Feature selections

1) Identify the target variable: This is the variable that you want to predict such as energy consumption.

2) Explore the data : This will help you to understand the relationships between the di erent features and the target variable you can use data visualisation under correlation analysis to identify features that are highly correlated with the target variable.

3) Remove redundant features: If two features are highly correlated with each other then you can remove one of the fishes as they are likely to contain redundant information.

4) Remove in irrelevant h features: If feature is not call later with the target variable then you can remove it , as it is unlikely to be useful for prediction.

Model training:

Model training is a process of teaching a machine learning model to measure energy consumption. It involve feeding the model historical data.

Once the model is strained it can be used to measure energy consumption for a new data for example you cou ld use the model to measure energy consumption.Prepare the data This involves cleaning the data removing a ny errors are inconsistencies and transforming the data into a format that is compatible with the machine learning algorithm that you will be using.

Split the data into training and test sets The training set will be used to change the model on the test Set will be used to evaluate the performance and the model of an unseen data.

Choose the machine learning algorithm Data cleaning , data analysis , Support vector modelling, Data visualisation .

Turn the hyper parameters of the algorithm The hyperparameters of an a machine learning algorithm parameters that control the learning process. It is important to tune the hyper parameters of the algorithm to optimise its performance.

Train the model on training set: This involves feeding the training data to the model allowing its to Learn the relationships between features and energy consumption.

Evolved the model on test set: This involves feeding the test test data to the model measuring how will it predicts the measure energy consumption. If the model performs well well on the test sets then you can be confident that it will generalize well to new data.

Model evaluation:

Model evaluation is the process of assessing the performance of a machine learning model on unseen data.

This is important to ensure that the model will generalize well to new data .

There are a number of metrics that can be used to evaluate the performance of a house price prediction model .some of the most common metrics include :

Mean squared error (MSE) : This metric measures the average squared di erence between the measured and actual energy consumption.

Root mean squared error (RMSE): This metrics is the square root of the MSE.

R squared: This metric measures how well the model explains the variation in the actual energy consumption. In addition to these metrics, it is also important to consider the following factors when evaluating a energy consumption model.

Bias: Bias is the tendency of a model to consistently over or underestimate energy consumption.

Variance: Variance is the measure of how much the protections of a model very around the true. Energy con. s umed.

Interpretability : Interpretability is the ability to understand how the model makes its protections. This is important for energy consumption models as it allows users to understand the factors.

Feature engineering:

Feature engineering is is a crucial aspect of building a energy consumption using machine learning. Was creating a new features transforming existing ones and selecting the most relevant variables to improve the models predictive power here or some feature engineering ideas for energy consumption measurement.

Various feature to perform model training:

Use a variety of feature engineering techniques

Feature engineering is the process of transforming raq data into features that more informative and a protective for a machine learning models.

Use cross validation

Cross validation is a technique for evolving the performance of machine learning model on unseen data. It is important to use cross validation to evaluate the performance of your model during a training process. This will help you to avoid over fitting and to Ensure that you are model will be generalize will to new data.

Use ensemble methods

Ensemble methods are a machine learning methods that combine the protections of a multiple models to produce or a more accurate protection ensemble methods can often are so better performance then entirely machine learning models.

Hold out test sets

A hold out test set is a set of data that is not used to train or evaluate the model during the training process.

This data is used to evolve the performance of the model on unseen data after training process is Complete

Compare the model to a baseline

A baseline is a simple model that is used to compare the performance of your model to. For example

You could use the mean value as a baseline.

Analyze the models predictions

Once you have a little evaluated the performance of the model you can analyse the models prediction to identify any patterns or biosis. This will help you to understand and weaknesses of the model and two improve it.

Conclusion :

In the quest to build an accurate and Reliable energy consumption measurement model we have embarked on a journey that encompasses critical phases from feature selection to model training and evaluation. Each of the stages place and dispensable role in crafting a model that can provide meaningful insights and estimates for one of the most significant financial decision make real estate transactions.