VIVEKANANDA SCHOOL OF ENGINEERING AND TECHNOLOGY

Minor Project Progress Report on

Topic: “Prediction of behaviour of Prosumers using Machine Learning ”

Krishna Goel, 02717711621, AIML-A

Anant Chauhan, 01517711621, AIML-A

Akanksha,08017711621, AIML-B

Supervisor:

Dr. Monika Bansal

&

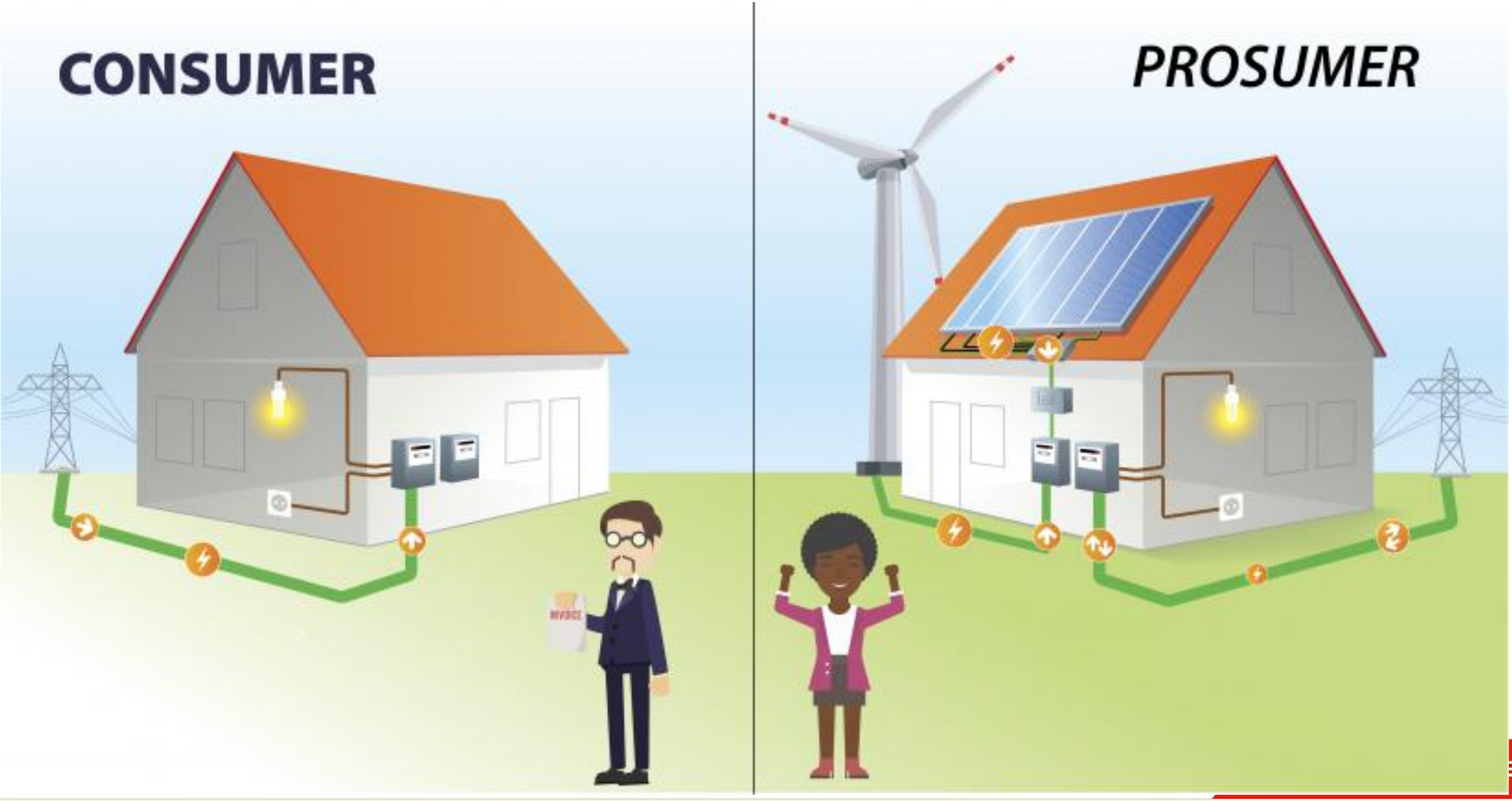
Dr. Bhawna Rawat

SCHOOL OF ENGINEERING AND TECHNOLOGY

Table of Content

1. Introduction & Motivation

- The increasing adoption of renewable energy sources has led to the emergence of "prosumers" who both produce and consume energy, such as through solar panels.
- This decentralized energy model poses challenges in predicting energy patterns due to factors like weather variability and fluctuating energy prices.
- Accurate forecasting is crucial to minimize grid imbalance costs and ensure efficient energy management. Motivated by the need for better grid stability, this project aims to develop a machine learning-based model to predict the energy behavior of Estonian prosumers, optimizing energy utilization and supporting sustainable energy practices.


CONSUMER
PROSUMER
INVOICE

2. Problem Statement

- The number of prosumers is rapidly increasing, associated with higher energy imbalance - increased operational costs, potential grid instability, and inefficient use of energy resources. The goal of the competition is to create an energy prediction model of prosumers to reduce energy imbalance costs If solved, it would reduce the imbalance costs, improve the reliability of the grid, and make the integration of prosumers into the energy system more efficient and sustainable. Moreover, it could potentially incentivize more consumers to become prosumers and thus promote renewable energy production and use.

3. Background Study

- Previous studies and energy forecasting competitions highlight the importance of using time-series data and factors like weather for accurate predictions. Machine learning algorithms, especially XGBoost, have shown effectiveness in handling large datasets with non-linear relationships. These insights form the foundation for developing a robust forecasting model to predict prosumer energy behavior, incorporating weather data, energy prices, and time-dependent features.

4. Technology Stack

- **Programming Language:**
  - Python 3.x: Python will be the primary language used for data analysis, model development, and evaluation.
- **Libraries and Dependencies:**
  - XGBoost: The primary library for model implementation
  - Pandas and NumPy: For data manipulation and handling
  - Scikit-learn: For data preprocessing and model evaluation
  - Matplotlib and Seaborn: For data visualization and feature analysis
  - Jupyter Notebook/Google Colab/Kaggle Notebooks: For interactive development and testing.
  - Kaggle API: For data access and leaderboard submission.
- **Development Environment:**
  - Anaconda (optional): A Python distribution with pre-installed libraries
  - Kaggle Notebooks or Google Colab: for cloud-based computation, eliminating the need for high-end local hardware

4. Project Progress

VIVEKANANDA SCHOOL OF ENGINEERING AND TECHNOLOGY

First, we imported all the necessary libraries required

```
!pip install xgboost -U
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
```

```
[30] !pip install colorama
```

```
Requirement already satisfied: colorama in /usr/local/lib/python3.10/dist-packages (0.4.6)
```

```
[31] #General
import pandas as pd
import numpy as np
import json
# Visualization
import seaborn as sns
import matplotlib.pyplot as plt
from colorama import Fore, Style, init;

# Modeling
import xgboost as xgb
import lightgbm as lgb
import torch

# Geolocation
from geopy.geocoders import Nominatim

# Options
pd.set_option('display.max_columns', 100)
```

Step 2

COLLEGE OF ENGINEERING

Access or import the data into the project

```
[35] # Read CSVs and parse relevant date columns
     train = pd.read_csv("/content/train.csv")
     client = pd.read_csv("/content/client.csv")
     historical_weather = pd.read_csv("/content/historical_weather.csv")
     forecast_weather = pd.read_csv("/content/forecast_weather.csv")
     electricity = pd.read_csv("/content/electricity_prices.csv")
     gas = pd.read_csv("/content/gas_prices.csv")
```

```
[ ] # Location from https://www.kaggle.com/datasets/michaelo/fabiendaniels-mapping-locations-and-county-codes/data
    location = (pd.read_csv("/content/county_lon_lats.csv")
                .drop(columns = ["Unnamed: 0"])
               )
```

```
[37] display_df(train, 'train')
     display_df(client, 'client')
     display_df(historical_weather, 'historical weather')
     display_df(forecast_weather, 'forecast weather')
     display_df(electricity, 'electricity prices')
     display_df(gas, 'gas prices')
     display_df(location, 'location data')
```

Step 3

Data preprocessing

```
class FeatureProcessorClass():
    def __init__(self):
        self.weather_join = ['datetime', 'county', 'data_block_id']
        self.gas_join = ['data_block_id']
        self.electricity_join = ['datetime', 'data_block_id']
        self.client_join = ['county', 'is_business', 'product_type', 'data_block_id']

        self.lat_lon_columns = ['latitude', 'longitude']

        self.agg_stats = ['mean']

        self.category_columns = ['county', 'is_business', 'product_type', 'is_consumption', 'data_block_id']

    def create_new_column_names(self, df, suffix, columns_no_change):
        '''Change column names by given suffix, keep columns_no_change, and return back the data'''
        df.columns = [col + suffix
                      if col not in columns_no_change
                      else col
                      for col in df.columns
                      ]
        return df

    def flatten_multi_index_columns(self, df):
        df.columns = ['_'.join([col for col in multi_col if len(col)>0])
                      for multi_col in df.columns]
        return df

    def create_data_features(self, data):
        '''📊Create features for main data (test or train) set📊'''
        data['datetime'] = pd.to_datetime(data['datetime'])


        data['date'] = data['datetime'].dt.normalize()
        data['year'] = data['datetime'].dt.year
```

Data preprocessing

Data Preprocessing


```python
class FeatureProcessorClass():
    def __init__(self):
        self.weather_join = ['datetime', 'county', 'data_block_id']
        self.gas_join = ['data_block_id']
        self.electricity_join = ['datetime', 'data_block_id']
        self.client_join = ['county', 'is_business', 'product_type', 'data_block_id']

        self.lat_lon_columns = ['latitude', 'longitude']

        self.agg_stats = ['mean']

        self.category_columns = ['county', 'is_business', 'product_type', 'is_consumption', 'data_block_id']

    def create_new_column_names(self, df, suffix, columns_no_change):
        '''Change column names by given suffix, keep columns_no_change, and return back the data'''
        df.columns = [col + suffix
                      if col not in columns_no_change
                      else col
                      for col in df.columns
                      ]
        return df

    def flatten_multi_index_columns(self, df):
        df.columns = ['_'.join([col for col in multi_col if len(col)>0])
                      for multi_col in df.columns]
        return df

    def create_data_features(self, data):
        '''📊Create features for main data (test or train) set📊'''
        data['datetime'] = pd.to_datetime(data['datetime'])


        data['date'] = data['datetime'].dt.normalize()
        data['year'] = data['datetime'].dt.year
```

Data Preprocessing

COLLEGE OF ENGINEERING

```
    data['day_of_year'] = data['datetime'].dt.day_of_year
    data['day_of_month']  = data['datetime'].dt.day
    data['day_of_week'] = data['datetime'].dt.day_of_week
    return data

def create_client_features(self, client):
    '''💼 Create client features 💼'''

    client = self.create_new_column_names(client,
                                           suffix='_client',
                                           columns_no_change = self.client_join
                                          )
    return client

def create_historical_weather_features(self, historical_weather):
    '''⏳🌥️ Create historical weather features 🌥️⏳'''


    historical_weather['datetime'] = pd.to_datetime(historical_weather['datetime'])


    historical_weather[self.lat_lon_columns] = historical_weather[self.lat_lon_columns].astype(float).round(1)
    historical_weather = historical_weather.merge(location, how = 'left', on = self.lat_lon_columns)


    historical_weather = self.create_new_column_names(historical_weather,
                                                      suffix='_h',
                                                      columns_no_change = self.lat_lon_columns + self.weather_join
                                                     )


    agg_columns = [col for col in historical_weather.columns if col not in self.lat_lon_columns + self.weather_join]
    agg_dict = {agg_col: self.agg_stats for agg_col in agg_columns}
    historical_weather = historical_weather.groupby(self.weather_join).agg(agg_dict).reset_index()
```

Data Preprocessing

COLLEGE OF ENGINEERING

VIPS Technical Campus

```
    agg_columns = [col for col in historical_weather.columns if col not in self.lat_lon_columns + self.weather_join]
    agg_dict = {agg_col: self.agg_stats for agg_col in agg_columns}
    historical_weather = historical_weather.groupby(self.weather_join).agg(agg_dict).reset_index()


    historical_weather = self.flatten_multi_index_columns(historical_weather)


    historical_weather['hour_h'] = historical_weather['datetime'].dt.hour
    historical_weather['datetime'] = (historical_weather
                                      .apply(lambda x:
                                             x['datetime'] + pd.DateOffset(1)
                                             if x['hour_h']< 11
                                             else x['datetime'] + pd.DateOffset(2),
                                             axis=1)
                                     )

    return historical_weather

def create_forecast_weather_features(self, forecast_weather):
    ''' 🔮 ⛅ Create forecast weather features ⛅ 🔮 '''


    forecast_weather = (forecast_weather
                        .rename(columns = {'forecast_datetime': 'datetime'})
                        .drop(columns = 'origin_datetime')
                       )


    forecast_weather['datetime'] = (pd.to_datetime(forecast_weather['datetime'])
                                    .dt
                                    .tz_localize(None)
                                   )
```

Data Preprocessing

```
    forecast_weather[self.lat_lon_columns] = forecast_weather[self.lat_lon_columns].astype(float).round(1)
    forecast_weather = forecast_weather.merge(location, how = 'left', on = self.lat_lon_columns)


    forecast_weather = self.create_new_column_names(forecast_weather,
                                                     suffix='_f',
                                                     columns_no_change = self.lat_lon_columns + self.weather_join
                                                     )


    agg_columns = [col for col in forecast_weather.columns if col not in self.lat_lon_columns + self.weather_join]
    agg_dict = {agg_col: self.agg_stats for agg_col in agg_columns}
    forecast_weather = forecast_weather.groupby(self.weather_join).agg(agg_dict).reset_index()


    forecast_weather = self.flatten_multi_index_columns(forecast_weather)
    return forecast_weather

def create_electricity_features(self, electricity):
    ''' ⚡ Create electricity prices features ⚡ '''

    electricity['forecast_date'] = pd.to_datetime(electricity['forecast_date'])


    electricity['datetime'] = electricity['forecast_date'] + pd.DateOffset(1)


    electricity = self.create_new_column_names(electricity,
                                                suffix='_electricity',
                                                columns_no_change = self.electricity_join
                                               )
    return electricity

def create_gas_features(self, gas):
    ''' ⛽ Create gas prices features ⛽ '''
```

Data Preprocessing

VIPS Technical Campus
योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION
COLLEGE OF ENGINEERING

```python
    gas['mean_price_per_mwh'] = (gas['lowest_price_per_mwh'] + gas['highest_price_per_mwh'])/2


    gas = self.create_new_column_names(gas,
                                       suffix='_gas',
                                       columns_no_change = self.gas_join
                                      )

    return gas

def __call__(self, data, client, historical_weather, forecast_weather, electricity, gas):
    '''Processing of features from all datasets, merge together and return features for dataframe df '''
    # Create features for relevant dataset
    data = self.create_data_features(data)
    client = self.create_client_features(client)
    historical_weather = self.create_historical_weather_features(historical_weather)
    forecast_weather = self.create_forecast_weather_features(forecast_weather)
    electricity = self.create_electricity_features(electricity)
    gas = self.create_gas_features(gas)

    # 🔗 Merge all datasets into one df 🔗
    df = data.merge(client, how='left', on = self.client_join)
    df = df.merge(historical_weather, how='left', on = self.weather_join)
    df = df.merge(forecast_weather, how='left', on = self.weather_join)
    df = df.merge(electricity, how='left', on = self.electricity_join)
    df = df.merge(gas, how='left', on = self.gas_join)

    # Change columns to categorical for XGBoost
    df[self.category_columns] = df[self.category_columns].astype('category')
    return df
```

Data Preprocessing

COLLEGE OF ENGINEERING

```
[41] df
```

	county	is_business	product_type	target	is_consumption	datetime	data_block_id	row_id	prediction_unit_id	date	year	quarter	month	week	hour	day_of_year	day_of_month	day_of_week	eic_count_client	installed_capacity_client	date
0	0	0	1	0.713	0	2021-09-01 00:00:00	0	0	0	2021-09-01	2021	3	9	35	0	244	1	2	NaN	NaN	
1	0	0	1	96.590	1	2021-09-01 00:00:00	0	1	0	2021-09-01	2021	3	9	35	0	244	1	2	NaN	NaN	
2	0	0	2	0.000	0	2021-09-01 00:00:00	0	2	1	2021-09-01	2021	3	9	35	0	244	1	2	NaN	NaN	
3	0	0	2	17.314	1	2021-09-01 00:00:00	0	3	1	2021-09-01	2021	3	9	35	0	244	1	2	NaN	NaN	
4	0	0	3	2.904	0	2021-09-01 00:00:00	0	4	2	2021-09-01	2021	3	9	35	0	244	1	2	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2018347	15	1	0	197.233	1	2023-05-31 23:00:00	637	2018347	64	2023-05-31	2023	2	5	22	23	151	31	2	15.0	620.0	20
2018348	15	1	1	0.000	0	2023-05-31 23:00:00	637	2018348	59	2023-05-31	2023	2	5	22	23	151	31	2	20.0	624.5	20
2018349	15	1	1	28.404	1	2023-05-31 23:00:00	637	2018349	59	2023-05-31	2023	2	5	22	23	151	31	2	20.0	624.5	20
2018350	15	1	3	0.000	0	2023-05-31 23:00:00	637	2018350	60	2023-05-31	2023	2	5	22	23	151	31	2	55.0	2188.2	20
2018351	15	1	3	196.240	1	2023-05-31 23:00:00	637	2018351	60	2023-05-31	2023	2	5	22	23	151	31	2	55.0	2188.2	20

2018352 rows × 71 columns

Step 4

K-Fold Validation Technique to form training set and testing set and using early stopping criteria on best iteration

```
[43] #### Create single fold split ######
     # Remove empty target row
     target = 'target'
     df = df[df[target].notnull()].reset_index(drop=True)

     train_block_id = list(range(0, 600))

     tr = df[df['data_block_id'].isin(train_block_id)] # first 600 data_block_ids used for training
     val = df[~df['data_block_id'].isin(train_block_id)] # rest data_block_ids used for validation
```

```
[44] # Remove columns for features
     no_features = ['date',
                    'latitude',
                    'longitude',
                    'data_block_id',
                    'row_id',
                    'hours_ahead',
                    'hour_h',
                   ]

     remove_columns = [col for col in df.columns for no_feature in no_features if no_feature in col]
     remove_columns.append(target)
     features = [col for col in df.columns if col not in remove_columns]
     PrintColor(f'There are {len(features)} features: {features}')
```

```
There are 59 features: ['county', 'is_business', 'product_type', 'is_consumption', 'prediction_unit_id', 'year', 'quarter', 'month', 'week', 'hour', 'day_of_year', 'day_of_month', 'day_of_week', 'eic_count_
```

K-Fold Validation Technique to form training set and testing set and using early stopping criteria on best iteration

VIPS Technical Campus
योग: कर्मसु कौशलम्
IN PURSUIT OF PERFECTION
COLLEGE OF ENGINEERING

```
[45] clf = xgb.XGBRegressor(
                    device = device,
                    enable_categorical=True,
                    objective = 'reg:absoluteerror',
                    n_estimators = 2 if DEBUG else 1500,
                    early_stopping_rounds=100
                   )
```

```
[46] clf.fit(X = tr[features],
         y = tr[target],
         eval_set = [(tr[features], tr[target]), (val[features], val[target])],
         verbose=True #False #True
        )
```

K-Fold Validation Technique to form training set and testing set and using early stopping criteria on best iteration


```
[0]	validation_0-mae:241.12323	validation_1-mae:312.26397
[1]	validation_0-mae:215.48373	validation_1-mae:280.35122
[2]	validation_0-mae:190.58266	validation_1-mae:249.47455
[3]	validation_0-mae:170.05462	validation_1-mae:222.15992
[4]	validation_0-mae:154.46917	validation_1-mae:205.46406
[5]	validation_0-mae:141.12830	validation_1-mae:190.38314
[6]	validation_0-mae:127.05446	validation_1-mae:175.56972
[7]	validation_0-mae:107.91998	validation_1-mae:155.25746
[8]	validation_0-mae:92.86249	validation_1-mae:136.60018
[9]	validation_0-mae:82.42118	validation_1-mae:124.58340
[10]	validation_0-mae:75.22385	validation_1-mae:116.87053
[11]	validation_0-mae:69.87380	validation_1-mae:111.35297
[12]	validation_0-mae:65.86708	validation_1-mae:108.42840
[13]	validation_0-mae:63.41799	validation_1-mae:106.62030
[14]	validation_0-mae:61.83447	validation_1-mae:105.41863
[15]	validation_0-mae:60.67036	validation_1-mae:104.81440
[16]	validation_0-mae:60.03713	validation_1-mae:104.31592
[17]	validation_0-mae:59.73466	validation_1-mae:104.17829
[18]	validation_0-mae:59.06531	validation_1-mae:103.89732
[19]	validation_0-mae:58.77230	validation_1-mae:103.61167
[20]	validation_0-mae:58.52856	validation_1-mae:102.94702
[21]	validation_0-mae:58.39721	validation_1-mae:102.92430
[22]	validation_0-mae:57.65531	validation_1-mae:102.44742
[23]	validation_0-mae:57.59143	validation_1-mae:102.44925
[24]	validation_0-mae:57.29335	validation_1-mae:102.41553
[25]	validation_0-mae:57.20160	validation_1-mae:102.32758
[26]	validation_0-mae:57.12754	validation_1-mae:102.29553
[27]	validation_0-mae:57.08240	validation_1-mae:102.23743
[28]	validation_0-mae:57.06863	validation_1-mae:102.23522
[29]	validation_0-mae:57.02440	validation_1-mae:102.23712
[30]	validation_0-mae:57.01546	validation_1-mae:102.25039
[31]	validation_0-mae:56.69954	validation_1-mae:102.09448
[32]	validation_0-mae:56.69584	validation_1-mae:102.08793
[33]	validation_0-mae:56.58093	validation_1-mae:102.08320
[34]	validation_0-mae:56.46632	validation_1-mae:102.01176
[35]	validation_0-mae:56.39323	validation_1-mae:102.00280
[36]	validation_0-mae:56.07047	validation_1-mae:101.76820
[37]	validation_0-mae:56.00755	validation_1-mae:101.75514
[38]	validation_0-mae:55.90150	validation_1-mae:101.75510
[39]	validation_0-mae:55.85911	validation_1-mae:101.75749
[40]	validation_0-mae:55.83376	validation_1-mae:101.75476
[41]	validation_0-mae:55.81430	validation_1-mae:101.75730
[42]	validation_0-mae:55.78380	validation_1-mae:101.71542
[43]	validation_0-mae:55.77400	validation_1-mae:101.70947
[44]	validation_0-mae:55.18879	validation_1-mae:101.14827
[45]	validation_0-mae:55.13238	validation_1-mae:101.06159
[46]	validation_0-mae:55.00017	validation_1-mae:100.70203
[47]	validation_0-mae:54.98069	validation_1-mae:100.70195
[48]	validation_0-mae:54.97029	validation_1-mae:100.68946
[49]	validation_0-mae:54.91898	validation_1-mae:100.62329
[50]	validation_0-mae:54.65175	validation_1-mae:100.41582
[51]	validation_0-mae:54.64617	validation_1-mae:100.41261
[52]	validation_0-mae:54.63752	validation_1-mae:100.41365
[53]	validation_0-mae:54.59981	validation_1-mae:100.39751
```

```
[259]	validation_0-mae:42.23728	validation_1-mae:94.12433
[260]	validation_0-mae:42.23487	validation_1-mae:94.12414
[261]	validation_0-mae:42.20502	validation_1-mae:94.11834
[262]	validation_0-mae:42.20119	validation_1-mae:94.11456
[263]	validation_0-mae:42.18261	validation_1-mae:94.11194
[264]	validation_0-mae:42.17070	validation_1-mae:94.09846
[265]	validation_0-mae:42.16569	validation_1-mae:94.10016
[266]	validation_0-mae:42.16375	validation_1-mae:94.10085
[267]	validation_0-mae:42.15673	validation_1-mae:94.10520
[268]	validation_0-mae:42.11565	validation_1-mae:94.27788
[269]	validation_0-mae:42.07482	validation_1-mae:94.32983
[270]	validation_0-mae:42.05008	validation_1-mae:94.32710
[271]	validation_0-mae:42.03620	validation_1-mae:94.32772
[272]	validation_0-mae:42.01983	validation_1-mae:94.33281
[273]	validation_0-mae:41.96606	validation_1-mae:94.32981
[274]	validation_0-mae:41.90699	validation_1-mae:94.52406
[275]	validation_0-mae:41.80366	validation_1-mae:94.47910
[276]	validation_0-mae:41.65386	validation_1-mae:94.41210
[277]	validation_0-mae:41.60196	validation_1-mae:94.43916
[278]	validation_0-mae:41.60141	validation_1-mae:94.43905
[279]	validation_0-mae:41.59750	validation_1-mae:94.43943
[280]	validation_0-mae:41.56954	validation_1-mae:94.94530
[281]	validation_0-mae:41.55908	validation_1-mae:94.94355
[282]	validation_0-mae:41.54911	validation_1-mae:94.94388
[283]	validation_0-mae:41.49693	validation_1-mae:94.94824
[284]	validation_0-mae:41.48755	validation_1-mae:94.92834
[285]	validation_0-mae:41.48714	validation_1-mae:94.92910
[286]	validation_0-mae:41.48553	validation_1-mae:94.93897
[287]	validation_0-mae:41.46526	validation_1-mae:94.91240
[288]	validation_0-mae:41.42701	validation_1-mae:94.89352
[289]	validation_0-mae:41.40710	validation_1-mae:94.89433
[290]	validation_0-mae:41.31379	validation_1-mae:94.81008
[291]	validation_0-mae:41.31128	validation_1-mae:94.81189
[292]	validation_0-mae:41.30984	validation_1-mae:94.81071
[293]	validation_0-mae:41.30933	validation_1-mae:94.81063
[294]	validation_0-mae:41.27753	validation_1-mae:94.66597
[295]	validation_0-mae:41.27584	validation_1-mae:94.66588
[296]	validation_0-mae:41.27261	validation_1-mae:94.66658
[297]	validation_0-mae:41.26813	validation_1-mae:94.66682
[298]	validation_0-mae:41.26775	validation_1-mae:94.66689
[299]	validation_0-mae:41.21081	validation_1-mae:94.32328
[300]	validation_0-mae:41.20718	validation_1-mae:94.32300
[301]	validation_0-mae:41.20170	validation_1-mae:94.32621
```

```
[47] PrintColor(f'Early stopping on best iteration #{clf.best_iteration} with MAE error on validation set of {clf.best_score:.2f}')
```

```
Early stopping on best iteration #202 with MAE error on validation set of 92.59
```

Step 5

VIPS Technical Campus COLLEGE OF ENGINEERING
योग: कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

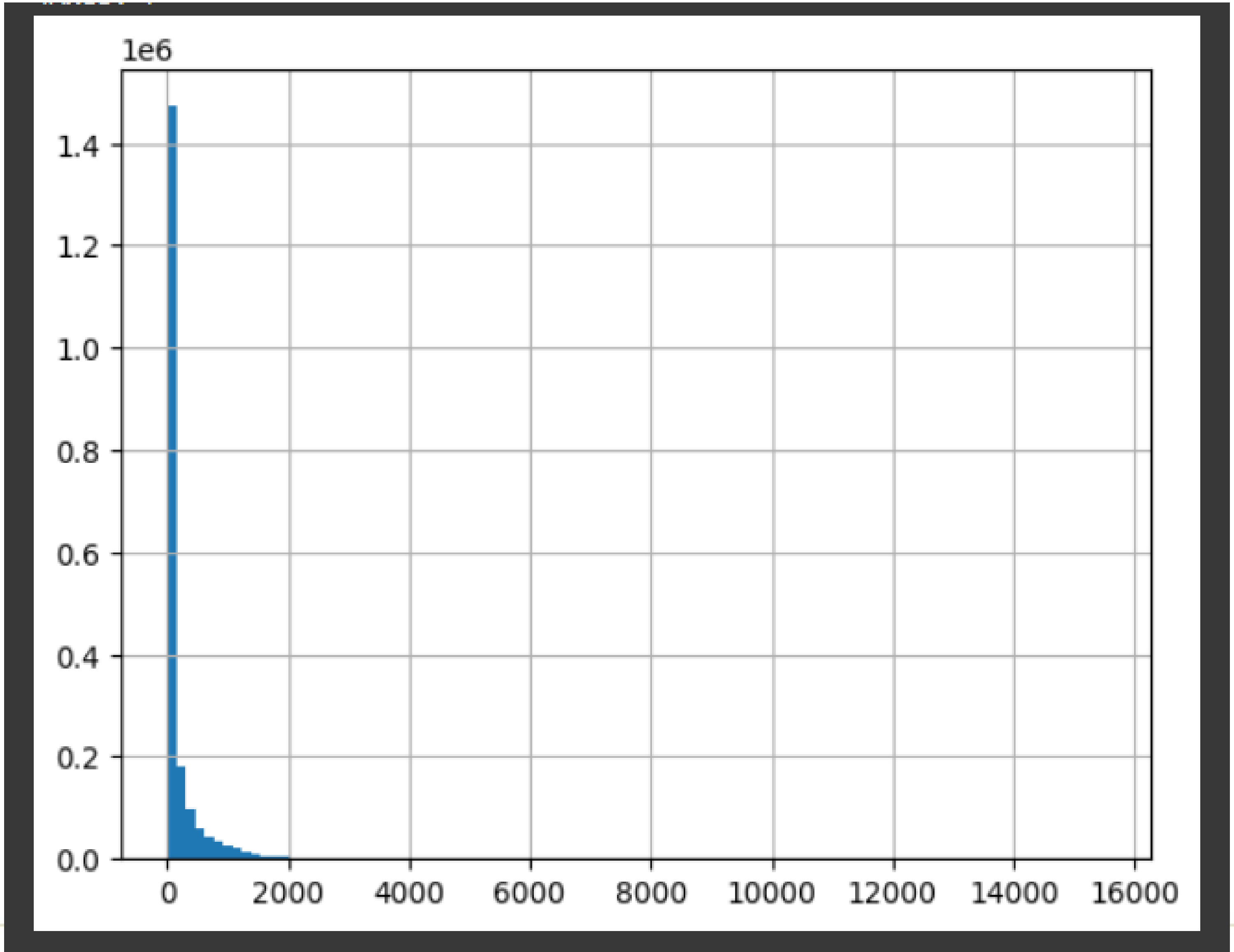
Target Distribution

```
[54] # Show target distribution
     display(train['target'].describe())
     train['target'].hist(bins=100)
```

	target
count	2017824.00000
mean	274.85556
std	909.50238
min	0.00000
25%	0.37800
50%	31.13300
75%	180.20625
max	15480.27400

dtype: float64

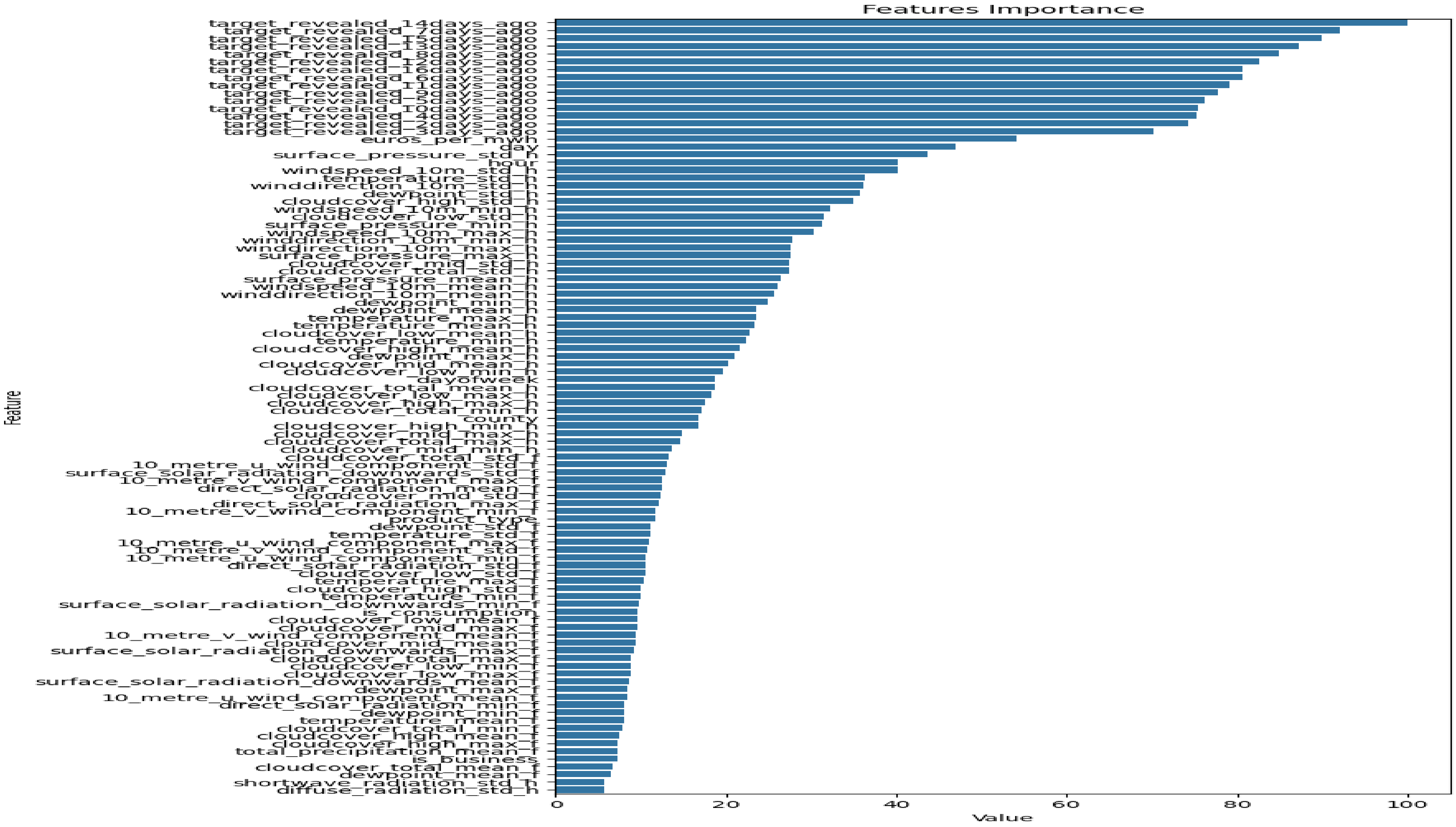
Target Distribution



Step 6

Identifies the most relevant and important features from the data set to obtain accuracy

```
[ ] # Features Importance
    if len(Feature_Imp) > 90 : plt.figure(figsize=(7, 15))
    elif len(Feature_Imp) > 60 : plt.figure(figsize=(7, 12))
    elif len(Feature_Imp) > 30 : plt.figure(figsize=(7, 10))
    else :
        plt.figure(figsize=(5, 5))
    sns.barplot(x="Value", y="Feature", data=Feature_Imp.head(100))
    plt.title('Features Importance')
    plt.show()
```

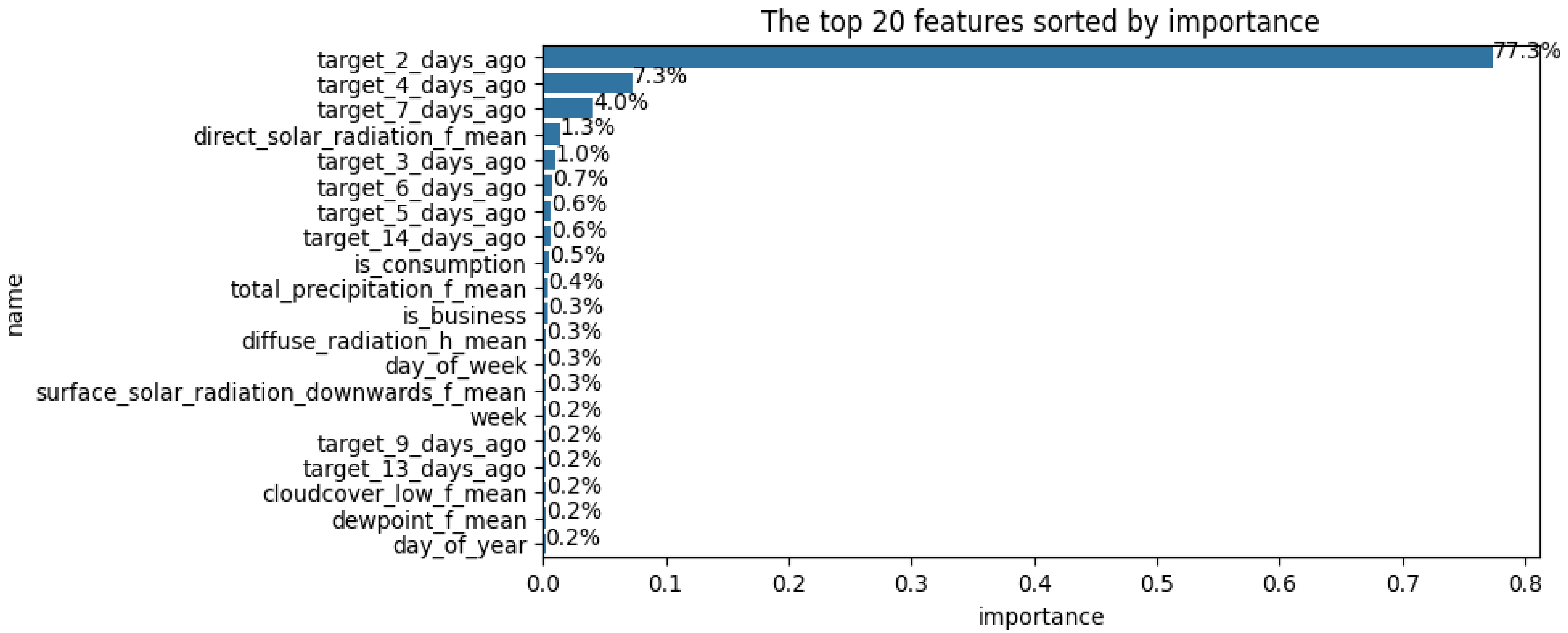

Features Importance
target_revealed_14days_ago
target_revealed_7days_ago
target_revealed_15days_ago
target_revealed_13days_ago
target_revealed_8days_ago
target_revealed_12days_ago
target_revealed_16days_ago
target_revealed_6days_ago
target_revealed_11days_ago
target_revealed_9days_ago
target_revealed_5days_ago
target_revealed_10days_ago
target_revealed_4days_ago
target_revealed_2days_ago
target_revealed_3days_ago
euros_per_mwh
day
surface_pressure_std_h
hour
windspeed_10m_std_h
temperature_std_h
winddirection_10m_std_h
dewpoint_std_h
cloudcover_high_std_h
windspeed_10m_min_h
cloudcover_low_std_h
surface_pressure_min_h
windspeed_10m_max_h
winddirection_10m_min_h
winddirection_10m_max_h
surface_pressure_max_h
cloudcover_mid_std_h
cloudcover_total_std_h
surface_pressure_mean_h
windspeed_10m_mean_h
winddirection_10m_mean_h
dewpoint_min_h
dewpoint_mean_h
temperature_max_h
temperature_mean_h
cloudcover_low_mean_h
temperature_min_h
cloudcover_high_mean_h
dewpoint_max_h
cloudcover_mid_mean_h
cloudcover_low_min_h
dayofweek
cloudcover_total_mean_h
cloudcover_low_max_h
cloudcover_high_max_h
cloudcover_total_min_h
county
cloudcover_high_min_h
cloudcover_mid_max_h
cloudcover_total_max_h
cloudcover_mid_min_h
cloudcover_total_std_f
10_metre_u_wind_component_std_f
surface_solar_radiation_downwards_std_f
10_metre_v_wind_component_max_f
direct_solar_radiation_mean_f
cloudcover_mid_std_f
direct_solar_radiation_max_f
10_metre_v_wind_component_min_f
product_type
dewpoint_std_f
temperature_std_f
10_metre_u_wind_component_max_f
10_metre_v_wind_component_std_f
10_metre_u_wind_component_min_f
direct_solar_radiation_std_f
cloudcover_low_std_f
temperature_max_f
cloudcover_high_std_f
temperature_min_f
surface_solar_radiation_downwards_min_f
is_consumption
cloudcover_low_mean_f
cloudcover_mid_max_f
10_metre_v_wind_component_mean_f
cloudcover_mid_mean_f
surface_solar_radiation_downwards_max_f
cloudcover_total_max_f
cloudcover_low_min_f
cloudcover_low_max_f
surface_solar_radiation_downwards_mean_f
dewpoint_max_f
10_metre_u_wind_component_mean_f
direct_solar_radiation_min_f
dewpoint_min_f
temperature_mean_f
cloudcover_total_min_f
cloudcover_high_mean_f
cloudcover_high_max_f
total_precipitation_mean_f
is_business
cloudcover_total_mean_f
dewpoint_mean_f
shortwave_radiation_std_h
diffuse_radiation_std_h
Feature
0
20
40
60
80
100
Value

Step 7

Extracted the most relevant top 20 features based on their importance

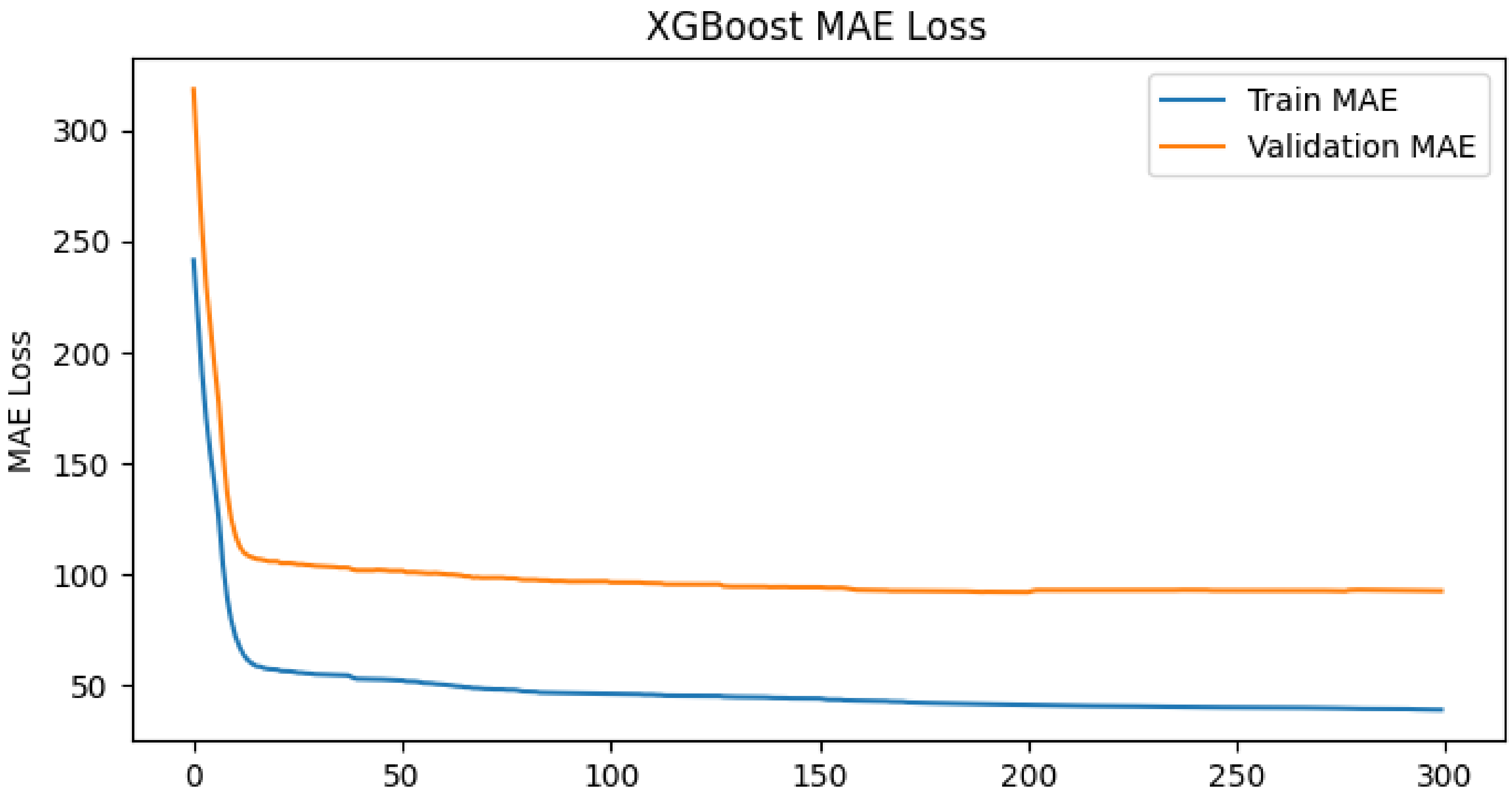
```python
TOP = 20
importance_data = pd.DataFrame({'name': clf.feature_names_in_, 'importance': clf.feature_importances_})
importance_data = importance_data.sort_values(by='importance', ascending=False)

fig, ax = plt.subplots(figsize=(8,4))
sns.barplot(data=importance_data[:TOP],
            x = 'importance',
            y = 'name'
        )
patches = ax.patches
count = 0
for patch in patches:
    height = patch.get_height()
    width = patch.get_width()
    perc = 100*importance_data['importance'].iloc[count]#100*width/len(importance_data)
    ax.text(width, patch.get_y() + height/2, f'{perc:.1f}%')
    count+=1

plt.title(f'The top {TOP} features sorted by importance')
plt.show()
```

Extracted the most relevant top 20 features based on their importance



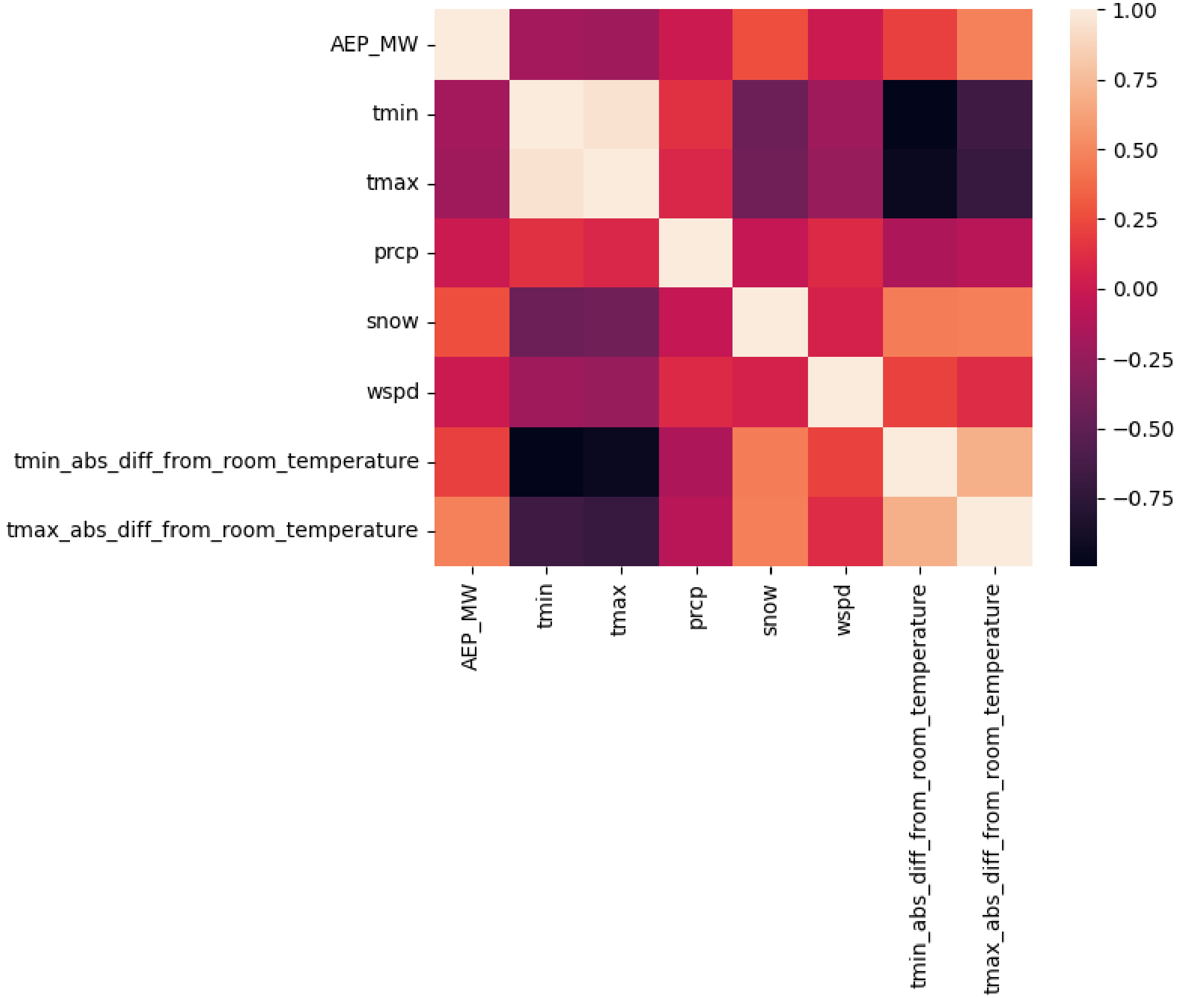
Step 8

RMSE CURVE



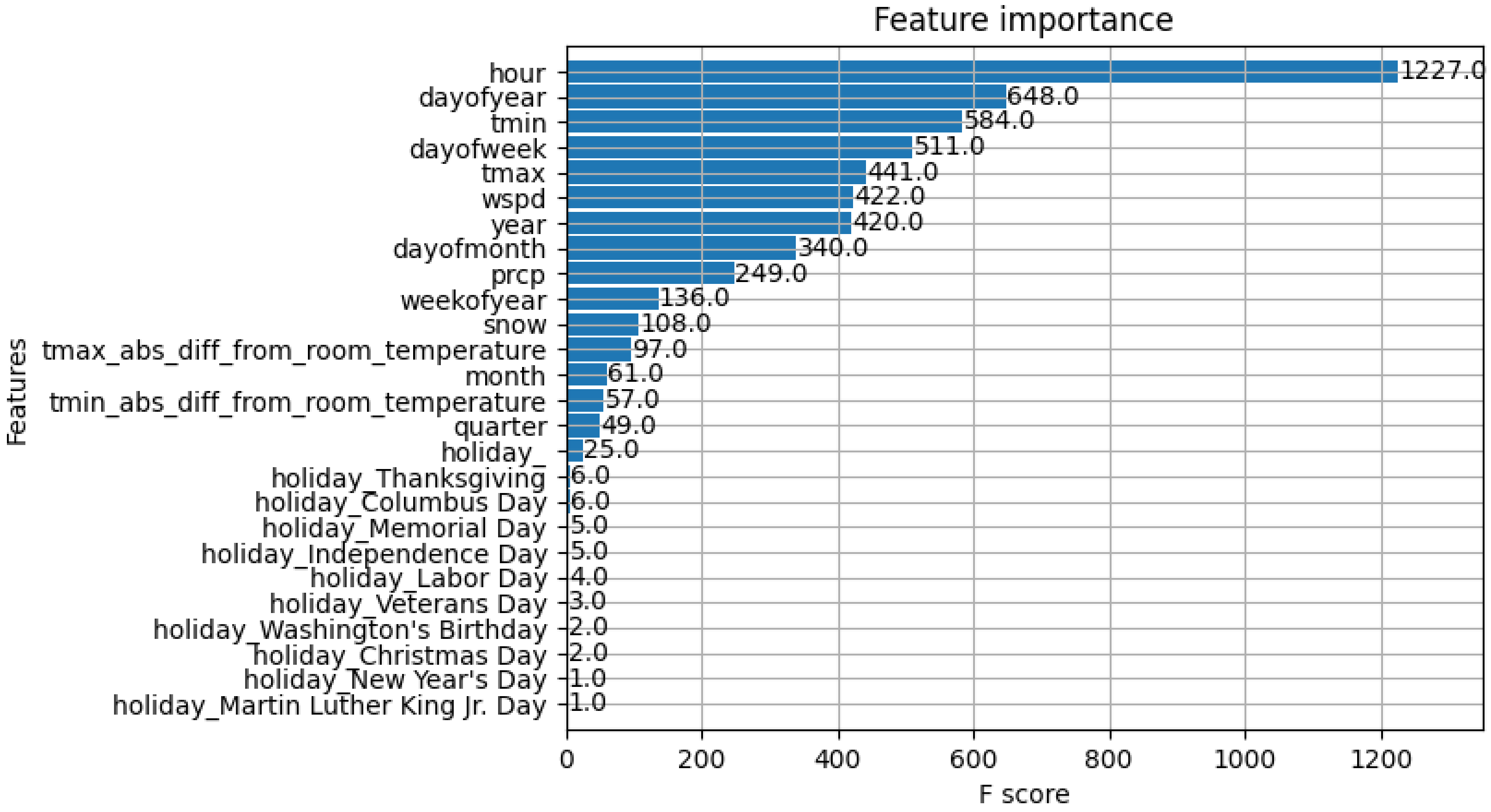
Step 9

CORRELATION MATRIX



Step 10

Most Important features in the new data



Step 11

VIPS Technical Campus COLLEGE OF ENGINEERING

Mean Absolute Error Percentage

```
train = data.loc[data['datetime'] <= split_date].copy()
test = data.loc[data['datetime'] > split_date].copy()
train_cols = ['hour', 'dayofweek', 'quarter', 'month', 'year',
       'dayofyear', 'dayofmonth']
X_train = train[train_cols]
# Replace 'target_column_name' with the actual name of your target column
target_column_name = 'AEP_MW'  # Example: Assuming 'AEP_MW' is your target column
y_train = train[target_column_name]
X_test = test[train_cols]
y_test = test[target_column_name]

reg = xgb.XGBRegressor(
    n_estimators=1000,
    early_stopping_rounds=50
)
reg.fit(X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
       verbose=False)
predictions = reg.predict(X_test)
# Assuming mean_absolute_percentage_error is defined elsewhere in your code
mean_absolute_percentage_error(y_true=y_test, y_pred=predictions)
```

```
9.373152324932883
```

6. Project Timeline

VIPS
योग: कर्मसु कौशलम्
IN PURSUIT OF PERFECTION
VIVEKANANDA SCHOOL OF ENGINEERING AND TECHNOLOGY

1. **Week 1-2: Project Planning and Data Collection**
   - Define project scope and objectives
   - Gather datasets (weather data, energy prices, photovoltaic capacities)
2. **Week 3-4: Data Preprocessing and Exploration**
   - Clean and preprocess data
   - Perform exploratory data analysis and feature engineering
3. **Week 5-6: Model Development**
   - Train initial machine learning models (XGBoost, Random Forest)
   - Fine-tune model parameters and select the best model
4. **Week 7: Model Evaluation**
   - Test model performance using metrics like MAE and RMSE
   - Optimize the model based on evaluation results
5. **Week 8: Deployment and Report Preparation**
   - Deploy the model for practical use
   - Prepare the final project report and presentation slides

7. References

1. *Hong, T., Pinson, P., & Fan, S. (2016).* "Global Energy Forecasting Competition 2012 and beyond." International Journal of Forecasting, 32(3), 596-608. DOI: [10.1016/j.ijforecast.2015.11.014](https://doi.org/10.1016/j.ijforecast.2015.11.014)
2. *Fan, S., & Hyndman, R. J. (2012).* "Short-term load forecasting based on a semi parametric additive model." IEEE Transactions on Power Systems, 27(1), 134-141. DOI:[10.1109/TPWRS.2011.2162082](https://doi.org/10.1109/TPWRS.2011.21620.
3. *Chen, T., & Guestrin, C. (2016).* "XGBoost: A scalable tree boosting system." In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge and Data Mining, [10.1145/2939672.2939785](https://doi.org/10.1145/2939672.2939785) 794.DOI: pp. 785
4. *Li, L., Yu, J., & Yang, Z. (2015).* "Renewable energy forecasting based on time series method and machine learning." Renewable and Sustainable Energy Reviews, 52, 273-284. DOI: [10.1016/j.rser.2015.07.078](https://doi.org/10.1016/j.rser.2015.07.078)