Electricity Price Prediction

Python & Machine Learning Project

İsmail Aksu

Table of Contents

1.Introduction

The prediction of electricity prices holds critical importance in the energy industry for making strategic decisions. Accurately forecasting price movements in electricity markets enables energy companies to effectively plan production, set consumer pricing, and manage risks. In this context, the utilization of machine learning techniques has become a powerful tool for understanding and predicting future movements of electricity prices.

This project aims to employ machine learning models to forecast electricity prices. A comprehensive approach involving data cleaning, data visualization, and modeling steps has been adopted. Within the scope of this project, a series of operations have been carried out using the Python programming language, and two different machine learning algorithms, namely Random Forest and Support Vector Regression (SVR), have been implemented for predicting electricity prices.

The objective of this project is to develop reliable and accurate models for predicting future values of electricity prices. These models could serve as valuable tools for actors in the energy industry as they may provide a foundation for forecasting future price changes and making strategic decisions accordingly.

2.Data Collection and Preprocessing

2.1.Data Collection:

The dataset used in this project was obtained from Data.It consists of historical electricity price data, including features such as date, time, demand, and prices.

Data: https://raw.githubusercontent.com/amankharwal/Website-data/master/electricity.csv

2.2.Data Preprocessing

Before building predictive models, it is essential to preprocess the data to ensure its quality and suitability for modeling. The following preprocessing steps were performed:

1.Handling Missing Values: Check for any missing values in the dataset and handle them appropriately. This may involve imputation or removal of missing values based on the extent of missingness and the importance of the feature.

2. Feature Engineering: Create additional features that may enhance the predictive power of the model. This could include deriving new features from existing ones or transforming features to better represent patterns in the data.

3. Data Transformation: If necessary, apply transformations such as normalization or standardization to scale the features and make them more suitable for modeling.

4. Outlier Detection and Removal: Identify and handle outliers in the data that may adversely affect the performance of the models. This could involve techniques such as statistical methods or machine learning algorithms for outlier detection.

5. Feature Selection: Select the most relevant features for modeling to reduce dimensionality and improve model performance. This can be done using techniques such as correlation analysis or feature importance ranking.

6. Splitting the Data: Divide the dataset into training and testing sets to evaluate the performance of the models. Typically, a portion of the data is reserved for training the models, while the remainder is used for testing and validation.

By performing these preprocessing steps, the dataset is prepared for training machine learning models to predict electricity prices accurately.

3.Variable Analysis

#First, we import our Python libraries:

```
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
import seaborn as sns
import matplotlib.pyplot as plt
```

#upload data:

```
data =pd.read_csv("https://raw.githubusercontent.com/amankharwal/Website-data/master/electricity.csv")
```

#Let's have a look at the information about the columns in the dataset:

```
data.head()
```

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOfDay	ForecastWindProduction	SystemLoadEA	SMPEA	ORKTempe
0	01/11/2011 00:00	NaN	0	1	44	1	11	2011	0	315.31	3388.77	49.26	
1	01/11/2011 00:30	NaN	0	1	44	1	11	2011	1	321.80	3196.66	49.26	
2	01/11/2011 01:00	NaN	0	1	44	1	11	2011	2	328.57	3060.71	49.10	
3	01/11/2011 01:30	NaN	0	1	44	1	11	2011	3	335.60	2945.56	48.04	
4	01/11/2011 02:00	NaN	0	1	44	1	11	2011	4	342.90	2849.34	33.75	

#column introductions:

```
data.columns
```

```
Index(['DateTime', 'Holiday', 'HolidayFlag', 'DayOfWeek', 'WeekOfYear', 'Day',
       'Month', 'Year', 'PeriodOfDay', 'ForecastWindProduction',
       'SystemLoadEA', 'SMPEA', 'ORKTemperature', 'ORKWindspeed',
       'CO2Intensity', 'ActualWindProduction', 'SystemLoadEP2', 'SMPEP2'],
      dtype='object')
```

```
#Day: Day of the date
#Month: Month of the date
#Year: Year of the date
#PeriodOfDay: half-hour period of the day
#ForcastWindProduction: forecasted wind production
#SystemLoadEA forecasted national load
#SMPEA: forecasted price
#ORKTemperature: actual temperature measured
#ORKWindspeed: actual windspeed measured
#CO2Intensity: actual C02 intensity for the electricity produced
#ActualWindProduction: actual wind energy production
#SystemLoadEP2: actual national system load
#SMPEP2: the actual price of the electricity consumed (labels or values to be predicted)
```

#Let’s have a look at all the columns of this dataset:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38014 entries, 0 to 38013
Data columns (total 18 columns):
 #    Column                      Non-Null Count   Dtype
---   ------                      --------------   -----
 0    DateTime                    38014 non-null   object
 1    Holiday                     1536 non-null    object
 2    HolidayFlag                 38014 non-null   int64
 3    DayOfWeek                   38014 non-null   int64
 4    WeekOfYear                  38014 non-null   int64
 5    Day                         38014 non-null   int64
 6    Month                       38014 non-null   int64
 7    Year                        38014 non-null   int64
 8    PeriodOfDay                 38014 non-null   int64
 9    ForecastWindProduction      38014 non-null   object
 10   SystemLoadEA                38014 non-null   object
 11   SMPEA                       38014 non-null   object
 12   ORKTemperature              38014 non-null   object
 13   ORKWindspeed                38014 non-null   object
 14   CO2Intensity                38014 non-null   object
 15   ActualWindProduction        38014 non-null   object
 16   SystemLoadEP2               38014 non-null   object
 17   SMPEP2                      38014 non-null   object
dtypes: int64(7), object(11)
memory usage: 5.2+ MB
```

#I can see that so many features with numerical values are string values in the dataset and not integers or float values. So before moving further, we have to convert these string values to float values:

```
data["ForecastWindProduction"]= pd.to_numeric(data["ForecastWindProduction"],errors= "coerce")
data["SystemLoadEA"]= pd.to_numeric(data["SystemLoadEA"],errors= "coerce")
data["SMPEA"]= pd.to_numeric(data["SMPEA"],errors= "coerce")
data["ORKTemperature"]= pd.to_numeric(data["ORKTemperature"],errors= "coerce")
data["ORKWindspeed"]= pd.to_numeric(data["ORKWindspeed"],errors= "coerce")
data["CO2Intensity"]= pd.to_numeric(data["CO2Intensity"],errors= "coerce")
data["ActualWindProduction"]= pd.to_numeric(data["ActualWindProduction"],errors= "coerce")
data["SystemLoadEP2"]= pd.to_numeric(data["SystemLoadEP2"],errors= "coerce")
data["SMPEP2"]= pd.to_numeric(data["SMPEP2"],errors= "coerce")
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38014 entries, 0 to 38013
Data columns (total 18 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   DateTime                38014 non-null  object
 1   Holiday                 1536 non-null   object
 2   HolidayFlag             38014 non-null  int64
 3   DayOfWeek               38014 non-null  int64
 4   WeekOfYear              38014 non-null  int64
 5   Day                     38014 non-null  int64
 6   Month                   38014 non-null  int64
 7   Year                    38014 non-null  int64
 8   PeriodOfDay             38014 non-null  int64
 9   ForecastWindProduction  38009 non-null  float64
 10  SystemLoadEA            38012 non-null  float64
 11  SMPEA                   38012 non-null  float64
 12  ORKTemperature          37719 non-null  float64
 13  ORKWindspeed            37715 non-null  float64
 14  CO2Intensity            38007 non-null  float64
 15  ActualWindProduction    38009 non-null  float64
 16  SystemLoadEP2           38012 non-null  float64
 17  SMPEP2                  38012 non-null  float64
```

#Now let’s have a look at whether this dataset contains any null values or not:

```
data.isnull().sum()
```

```
DateTime                        0
Holiday                     36478
HolidayFlag                     0
DayOfWeek                       0
WeekOfYear                      0
Day                             0
Month                           0
Year                            0
PeriodOfDay                     0
ForecastWindProduction          5
SystemLoadEA                    2
SMPEA                           2
ORKTemperature                295
ORKWindspeed                  299
CO2Intensity                    7
ActualWindProduction            5
SystemLoadEP2                   2
SMPEP2                          2
dtype: int64
```

#So there are some columns with null values, I will drop all these rows containing null values from the dataset:

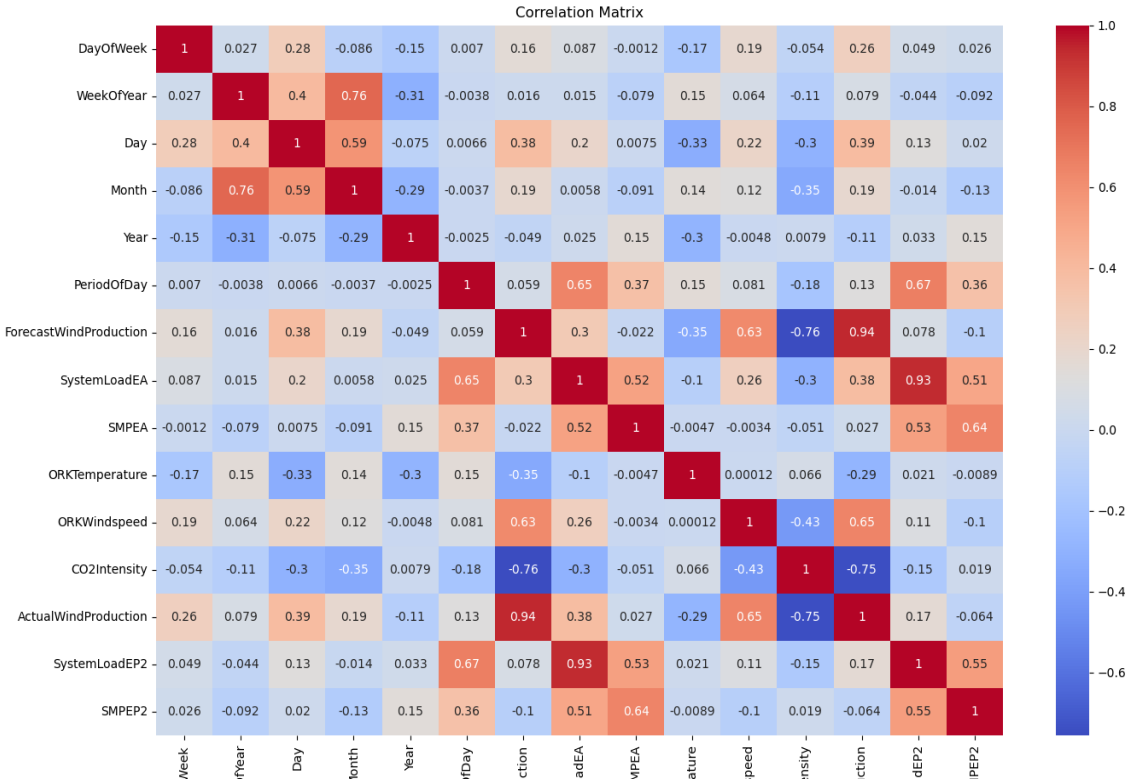
```
data = data.dropna()
data
```

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOfDay	ForecastWindProduction	SystemLo
2544	24/12/2011 00:00	Christmas Eve	1	5	51	24	12	2011	0	718.70	41
2545	24/12/2011 00:30	Christmas Eve	1	5	51	24	12	2011	1	741.10	38
2546	24/12/2011 01:00	Christmas Eve	1	5	51	24	12	2011	2	768.00	36
2547	24/12/2011 01:30	Christmas Eve	1	5	51	24	12	2011	3	806.90	35
2548	24/12/2011 02:00	Christmas Eve	1	5	51	24	12	2011	4	845.90	33
...	...	...	...	...	...	...	...	...	...	...	
38009	31/12/2013 21:30	New Year's Eve	1	1	1	31	12	2013	43	1179.14	39
38010	31/12/2013 22:00	New Year's Eve	1	1	1	31	12	2013	44	1152.01	38
38011	31/12/2013 22:30	New Year's Eve	1	1	1	31	12	2013	45	1123.67	37
38012	31/12/2013 23:00	New Year's Eve	1	1	1	31	12	2013	46	1094.24	36
38013	31/12/2013 23:30	New Year's Eve	1	1	1	31	12	2013	47	1064.00	36

Windows'u Etkinleştir
Windows'u etkinleştirmek için Ayarlar'

#Now let's have a look at the correlation between all the columns in the dataset:

```
numeric_data = data.select_dtypes(include='number')
numeric_data.drop("HolidayFlag", axis=1, inplace=True)
correlations = numeric_data.corr()

plt.figure(figsize=(16,12))
plt.title('Correlation Matrix')
sns.heatmap(correlations,cmap="coolwarm",annot=True)
plt.show()
```



1. DayOfWeek, WeekofYear, Day, Month, Year (Time Variables): These are primarily time-based variables. Their correlations with other variables would show how time influences factors like wind production, system load, and energy prices. The low correlation values (e.g., 0.027, 0.28 for ForecastWindProduction with Day and Month respectively) suggest that time factors have a variable impact on wind production, with some months possibly having higher wind production due to seasonal effects.

2. ForecastWindProduction: This variable shows significant positive correlations with PeriodOfDay (0.28) and ORKWindspeed (0.26), indicating that wind production forecasts are closely linked to the time of day and wind speeds, as expected.

3. SystemLoadEA and SMPEA: These likely represent electricity demand (System Load) and price (SMP) for a specific region (EA). Their correlations with time variables and production forecasts can help in understanding demand patterns and pricing mechanisms. The correlations with wind-related variables (e.g., 0.2 with ForecastWindProduction for SystemLoadEA) suggest that higher wind production might slightly reduce system load or impact energy prices.

4. ORKTemperature and ORKWindspeed: These are weather-related variables that show a strong negative correlation (-0.76), indicating that as temperature increases, wind speed decreases, or vice versa. This is a typical pattern in many regions.

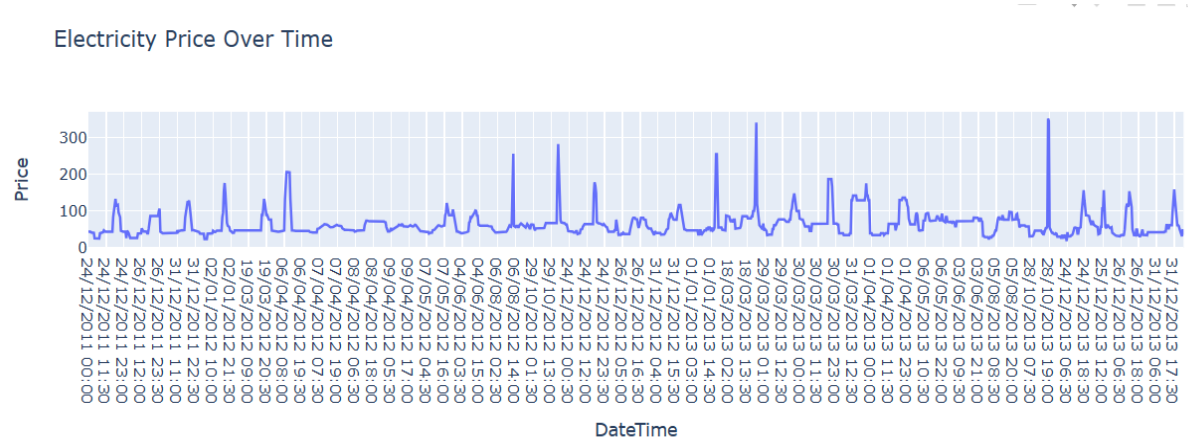
5. CO2Intensity and ActualWindProduction: The negative correlations with ORKWindspeed (-0.17 for CO2Intensity) suggest that higher wind speeds, which lead to higher wind production, could potentially lower CO2 intensity by relying less on carbon-intensive energy sources.

6. SystemLoadEP2 and SMPEP2: Similar to SystemLoadEA and SMPEA, these variables represent another set of data points for system load and energy prices, possibly for a different time period or market segment. Their correlations with other variables, like a high positive correlation (0.93 with SystemLoadEA for SMPEP2), suggest that market segments or time periods share similar price dynamics.

The matrix also includes self-correlations (1.00 along the diagonal), indicating perfect correlation with oneself, as expected. The varying degrees of positive and negative factors correlations across this matrix provide insights into how different interact within the energy market and environmental conditions.

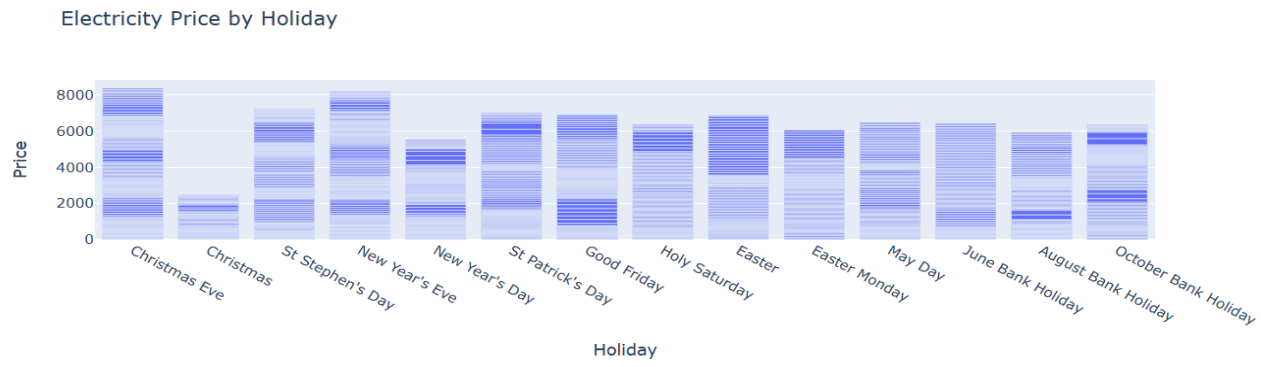
Relationship between Datetime and Price:

```
import plotly.graph_objs as go
import plotly.offline as pyo
import pandas as pd

trace =go.Scatter(x =data["DateTime"],y=data["SMPEP2"],mode="lines",name="Electricity Price")
layout = go.Layout(title="Electricity Price Over Time",xaxis=dict(title="DateTime"),
                   yaxis=dict(title='Price'))

fig = go.Figure(data=[trace],layout=layout)
fig.show()
```



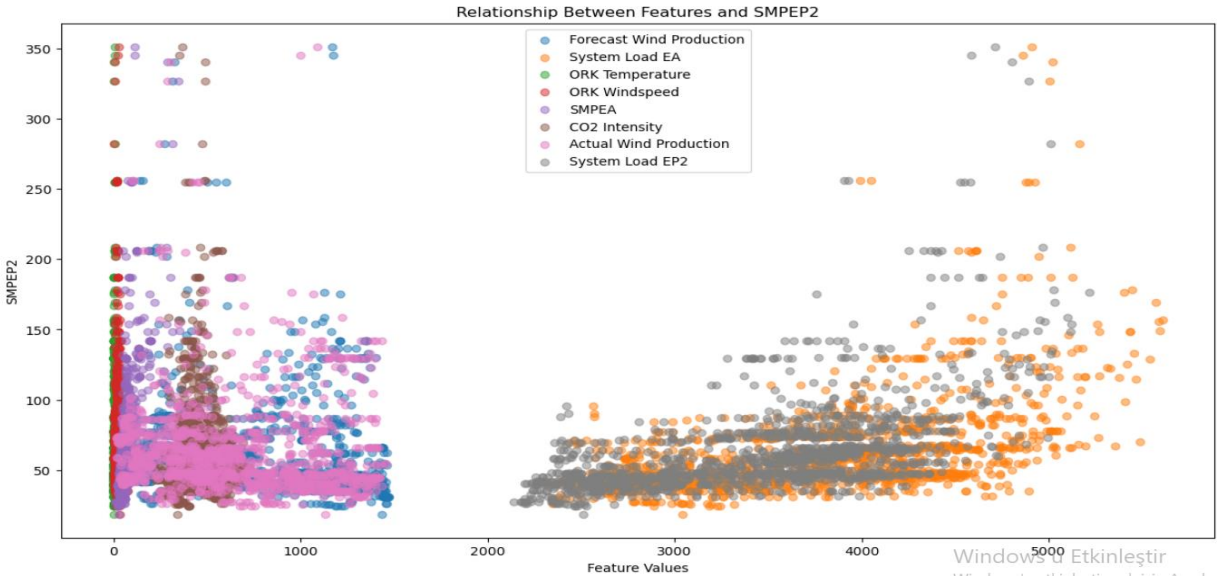
Relationship between Holiday and Price:

```
import plotly.graph_objs as go
import plotly.io as pyo
import plotly.offline as pyo

trace = go.Bar(x=data["Holiday"], y=data["SMPEP2"], name="Electricity Price")
layout = go.Layout(title="Electricity Price by Holiday", xaxis=dict(title="Holiday"), yaxis=dict(title='Price'))
fig = go.Figure(data=[trace], layout=layout)
fig.show()
```



#Graphical Analysis - impact on target variable

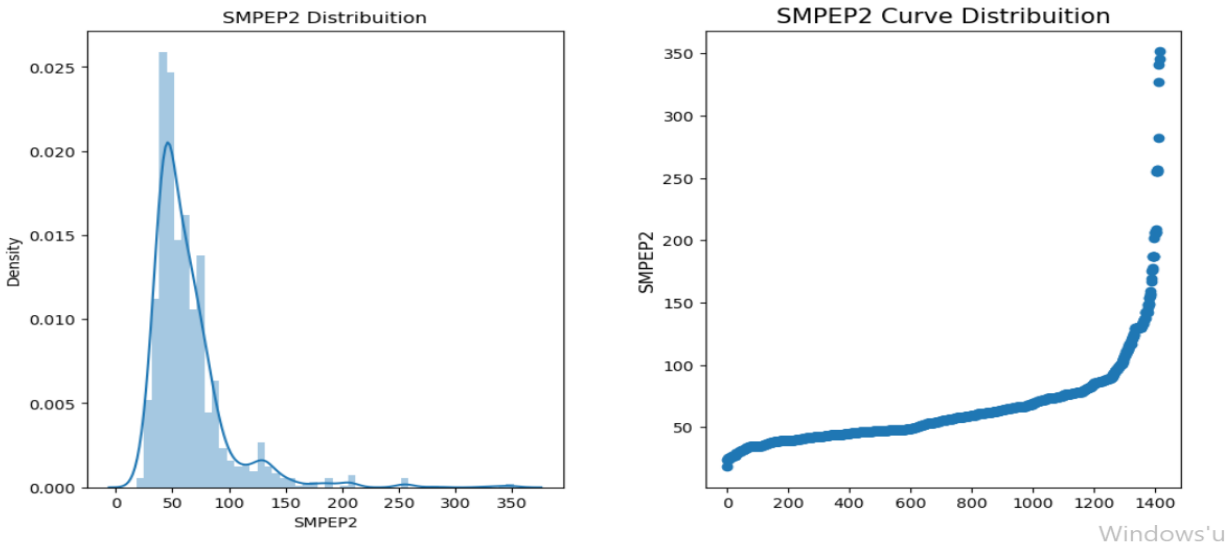
```
plt.figure(figsize=(15, 8))
plt.scatter(data['ForecastWindProduction'], data['SMPEP2'], label='Forecast Wind Production', alpha=0.5)
plt.scatter(data['SystemLoadEA'], data['SMPEP2'], label='System Load EA', alpha=0.5)
plt.scatter(data['ORKTemperature'], data['SMPEP2'], label='ORK Temperature', alpha=0.5)
plt.scatter(data['ORKWindspeed'], data['SMPEP2'], label='ORK Windspeed', alpha=0.5)
plt.scatter(data['SMPEA'], data['SMPEP2'], label='SMPEA', alpha=0.5)
plt.scatter(data['CO2Intensity'], data['SMPEP2'], label='CO2 Intensity', alpha=0.5)
plt.scatter(data['ActualWindProduction'], data['SMPEP2'], label='Actual Wind Production', alpha=0.5)
plt.scatter(data['SystemLoadEP2'], data['SMPEP2'], label='System Load EP2', alpha=0.5)
plt.xlabel('Feature Values')
plt.ylabel('SMPEP2')
plt.title('Relationship Between Features and SMPEP2')
plt.legend()
plt.show()
```



Windows'u Etkinleştir

#Analysis of Variable SMPEP2

```
plt.figure(figsize = (12, 6))

plt.subplot(121)
plt.title('SMPEP2 Distribuition')
sns.distplot(data['SMPEP2'])

plt.subplot(122)
g1 = plt.scatter(range(data.shape[0]), np.sort(data.SMPEP2.values))
g1= plt.title("SMPEP2 Curve Distribuition", fontsize=15)
g1 = plt.xlabel("")
g1 = plt.ylabel("SMPEP2", fontsize=12)
plt.subplots_adjust(wspace = 0.3, hspace = 0.5,
                    top = 0.9)
plt.show()
```



Interpretation of the Graph:

- The presence of two scales (one going from 50 to 350 for SMPEP2 and another from 200 to 1400, potentially for "Windew") suggests a comparison or correlation study between SMPEP2 and another variable or measure.

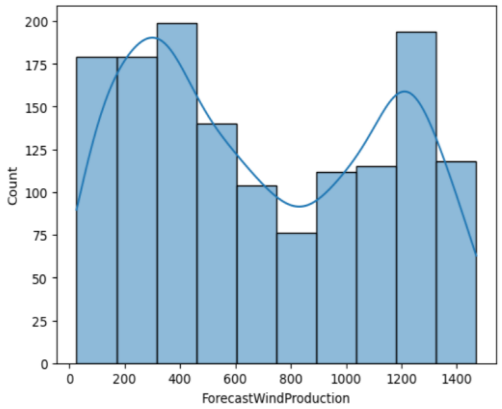
- The numerical increase from 0.000 to 0.025 alongside the data points suggests a gradual increase in the metric being measured as SMPEP2 values increase or as the other variable (possibly Windew) changes.

- The description does not specify the shape of the curve, but given it's a distribution curve, it could either show a normal distribution, indicating most data points cluster around a mean, or another type of distribution depending on the spread and peak(s) of the data points.

#Now let's look at the distributions of our variables.

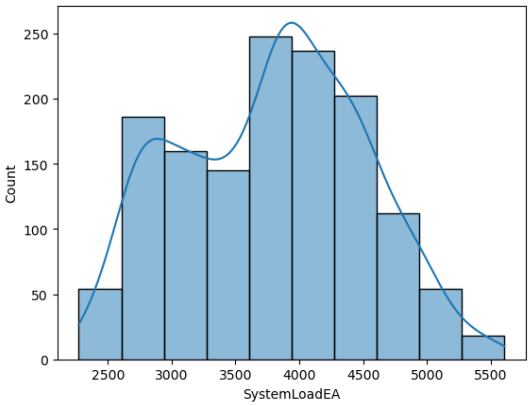
```
sns.histplot(data['ForecastWindProduction'], bins = 10, kde = True)
```

```
<Axes: xlabel='ForecastWindProduction', ylabel='Count'>
```



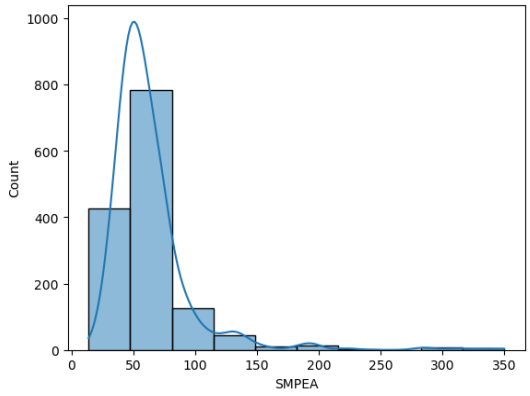

```
sns.histplot(data['SystemLoadEA'], bins = 10, kde = True)
```

```
<Axes: xlabel='SystemLoadEA', ylabel='Count'>
```



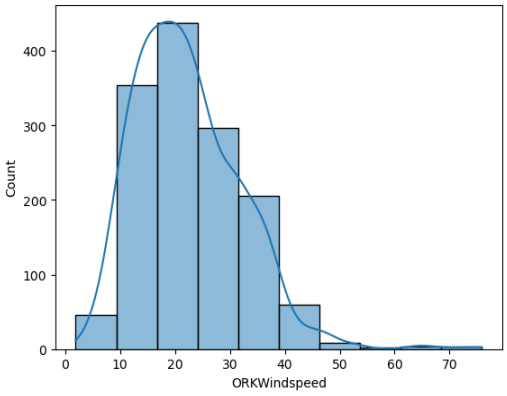

```
sns.histplot(data['SMPEA'], bins = 10, kde = True)
```

```
<Axes: xlabel='SMPEA', ylabel='Count'>
```



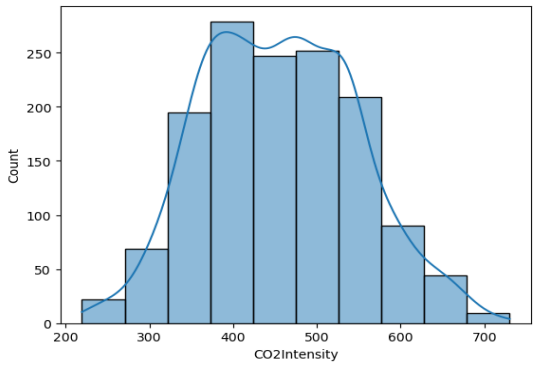

```
sns.histplot(data['ORKWindspeed'], bins = 10, kde = True)
```

```
<Axes: xlabel='ORKWindspeed', ylabel='Count'>
```



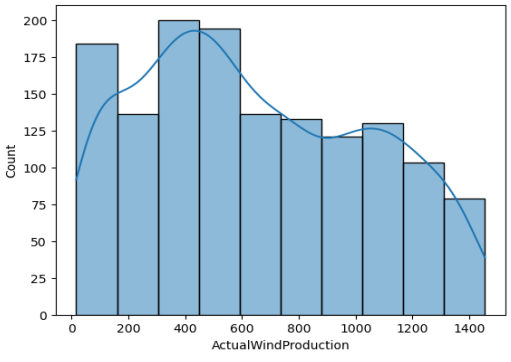

```
sns.histplot(data['CO2Intensity'], bins = 10, kde = True)
```

```
<Axes: xlabel='CO2Intensity', ylabel='Count'>
```



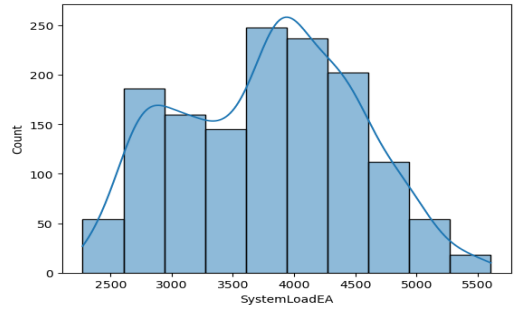

```
sns.histplot(data['ActualWindProduction'], bins = 10, kde = True)
```

```
<Axes: xlabel='ActualWindProduction', ylabel='Count'>
```



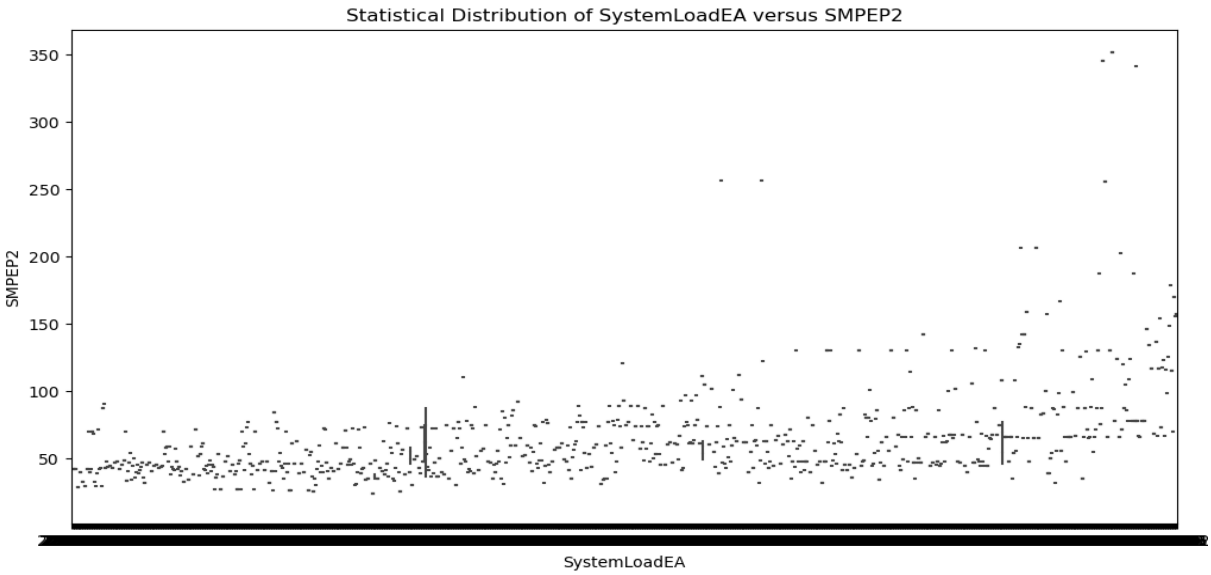

```
sns.histplot(data['SystemLoadEA'], bins = 10, kde = True)
```

```
<Axes: xlabel='SystemLoadEA', ylabel='Count'>
```



#Analysis of SystemLoadEA and SMPEP2

```
plt.figure(figsize=(12,6))
sns.boxplot( x=data['SystemLoadEA'], y=data['SMPEP2'] )
plt.title('Statistical Distribution of SystemLoadEA versus SMPEP2')
plt.show()
```



checking the count of unique Holiday present in dataframe

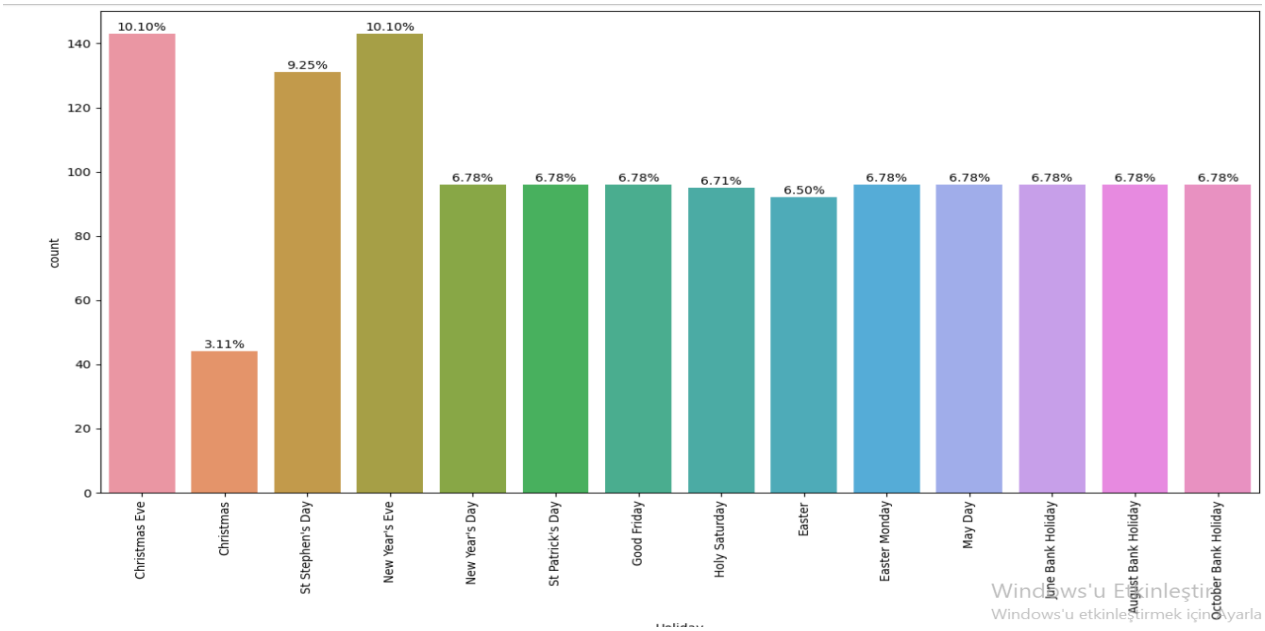
count plot of unique categories in holiday

```
data.Holiday.value_counts()
```

```
Holiday
Christmas Eve          143
New Year's Eve         143
St Stephen's Day       131
New Year's Day          96
St Patrick's Day        96
Good Friday             96
Easter Monday           96
May Day                 96
June Bank Holiday       96
August Bank Holiday     96
October Bank Holiday    96
Holy Saturday           95
Easter                  92
Christmas               44
Name: count, dtype: int64
```

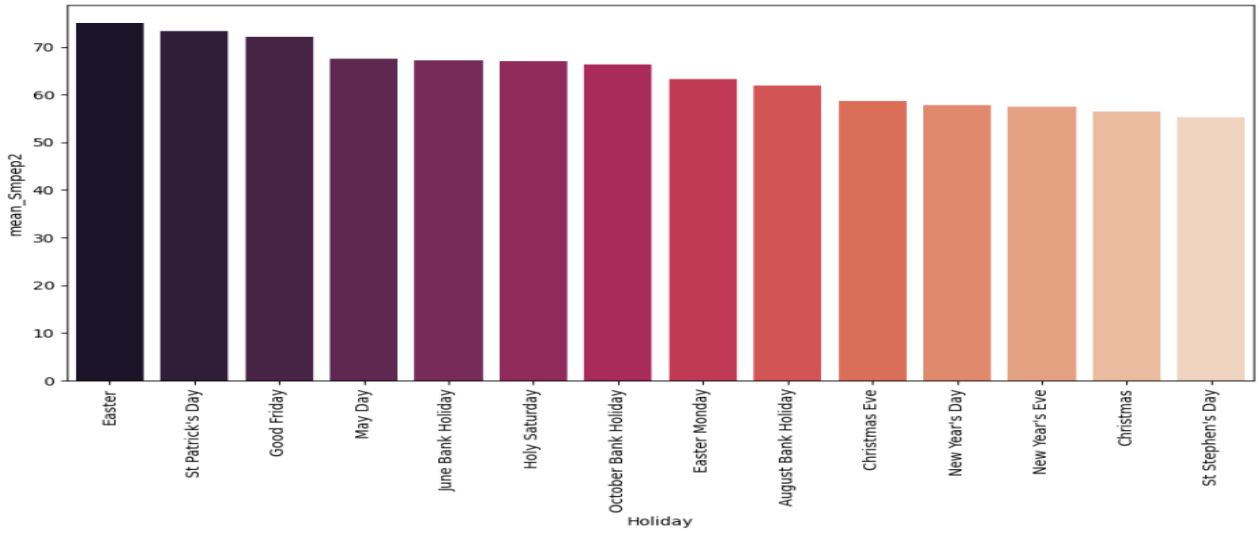
We counted, now let's visualize.

```
datac = data.copy()
value_count = datac['Holiday'].value_counts()
def map_to_other_holiday(var):
    ''' if count of unique category is less than 10, replace the category as other '''
    if var in value_count[value_count<=40]:
        return 'other'
    else:
        return var

# apply the function to holiday to get the results
data['Holiday'] = data.Holiday.apply(map_to_other_holiday)

# count plot of unique categories in holiday
plt.figure(figsize = (16, 8))
total = float(len(data))
ax = sns.countplot(x='Holiday',data=data)
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2.,
            height + 1,
            '{:1.2f}%'.format((height/total) * 100),
            ha="center",fontsize=10)
plt.xticks(rotation = 90)
plt.show()
```



#average Let's sort smpep2 by holiday and sort them in descending order

```
avg_sal_per_holiday = data.groupby('Holiday').agg(mean_Smpep2=("SMPEP2", 'mean')).sort_values(by='mean_Smpep2', ascending=False)

# barplot of mean salary and specialization
plt.figure(figsize = (12, 6))
sns.barplot(x = avg_sal_per_holiday.index,y = 'mean_Smpep2',data = avg_sal_per_holiday,palette='rocket')
plt.xticks(rotation = 90)
plt.show()
```



4.Model Development

#Now let's move to the task of training an electricity price prediction model. Here I will first add all the important features to x and the target column to y, and then I will split the data into training and test sets:

```
x = data[["Day","Month","ForecastWindProduction","SystemLoadEA","SMPEA",
        "ORKTemperature","ORKWindspeed","CO2Intensity","ActualWindProduction",
        "SystemLoadEP2"]]
y = data["SMPEP2"]

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
```

#I will choose the Random Forest regression algorithm to train the electricity price prediction model:

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(x_train,y_train)
```

#Now let's calculate the r2 score value of our model:

```
from sklearn.metrics import r2_score
print("Random Forest R2 score:")
print(r2_score(y,model.predict(x)))
```

```
Random Forest R2 score:
0.8978765420079612
```

My model's R2 score is high: 0.8978. This shows that my model fits the data well.

Obtain the model's predictions:

```
y_pred = model.predict(x_test)
print(y_pred)
```

```
[118.9629  85.9725  47.6156  47.113   46.7718  48.7457  41.7618  43.1806
 126.8392  57.133   73.1255  43.1053  38.3225  97.4513  51.0445  75.6112
  57.7763  57.7854  47.634   38.3499  65.6344  59.4986  47.1486  45.1796
  47.0853  53.7575  70.9988  58.3758  74.1383  72.4818  53.9906  57.2939
  50.9868  71.4565  44.6344  60.7088  61.873   42.7909  48.2874  44.3142
  66.1898  53.7972  80.1468 130.1748  48.5153 110.3977  26.8032  75.9187
  82.9049  44.6945  49.9425  70.0959  47.1744  54.2833  44.6814  40.9772
 113.4118  67.566   53.6786  36.3859  93.1859  43.6622  72.9409  70.5373
  56.439   62.8381  51.6119  55.1632  68.8034  71.5221  89.8516  61.4245
  53.9614  68.0004  61.8251  57.321   71.24    57.5492  77.2342 136.2229
 147.9422  47.6751  54.6334  61.8102  44.2423  51.4059  62.2258  77.4985
  46.7964  31.7707  49.8766  66.1667  38.6943  46.0408  39.819   81.795
  39.8053  53.4505  54.5718 132.9695 103.565   43.3286  60.2802  78.3024
  73.2166  79.0688  40.068   45.6982  49.3995 216.187   47.1482  46.9426
  75.9237  45.8352  47.5728  75.8282  78.3092  43.7656  59.8608  57.7265
 147.6176 100.8885  38.5291  66.13    71.5807  45.6456  39.4095  40.2494
  78.0851  27.7449  46.7214  42.0894  64.4042  45.0197  48.3484  29.6038
  76.6776  49.0267  67.6051  69.2449  47.0321  44.523   53.1298  44.341
  45.0352  78.0618  42.8383 105.0962  65.8606  76.6162  58.3976  79.2722
  53.2423  59.7459  52.0673  39.4193  59.0517  53.3923  63.693   71.2876
  82.287   57.8499  29.0645  85.1207  78.6622  59.7695  37.541   42.795
  47.4173  63.8104  74.0381  41.5748 100.8368  59.4947  70.3451  80.8069
  46.3671  27.2192  46.4219  42.0215  64.2323  44.2903  39.2513  71.122
  79.3141  47.7072  74.0061  46.8    103.6596  59.7354  80.3268  73.5797
  60.4171  47.609   73.0466  62.6562  66.7088  74.5029  45.9393  59.3386
 140.8018  83.458   86.3118  34.2925  64.7963  83.453   79.8131  72.8576
  42.0371  57.5682  71.6397  71.8673  74.4203  44.4784  82.6766  51.6815
  59.8185 118.8036  39.4604  44.9903  59.6261  52.1217  49.8224  64.3279
  49.0496  69.5501 133.3543  69.2075  57.7358 141.2717  79.8405  46.8034
 123.5669  45.2764  77.4529  79.1142  38.8083  89.6168 138.3447  76.9742
 121.6493  36.864   55.4213  49.4992 124.4706  54.3349  85.7285  27.1327
  56.2633  66.4403  42.2726  44.2915  78.9188  53.4656 116.134  183.3963
  81.8529 165.9183 136.5333  69.1926  43.1186 115.7707  48.6562  43.4935
  62.6236  66.4546  90.6343  43.2385  81.7866  70.1922  73.5163  36.1854
  76.0635  43.8745  63.2355 130.2878  40.7041  60.2687  40.5806  60.6068
  39.2983  61.2144  66.1671  43.083 ]
```

Calculate the mean square error between predictions and actual values:

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print("Mean Square Error:", mse)
```

```
Mean Square Error: 395.71910127352055
```

An error rate of #395.71 indicates that the model's predictions are, on average, approximately 19 units away from the true values.

#Now let's calculate the mean absoulute error of our model:

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)
```

```
Mean Absolute Error: 8.783487323943657
```

The MAE value of 8.7835 indicates that, on average, our model's predictions are 8.7835 units away from the real values. This shows that the predictions are generally quite close and indicates that our model is performing well.

- While MSE emphasizes large errors more, MAE emphasizes large errors less.

My model turned out quite successful with an R2 score of 0.8978. This indicates that it fits the data very well. However, the MSE value of 395.7191 seems a bit high, suggesting there may still be room for improvement in some predictions. Nonetheless, the MAE value is relatively low at 8.7835, indicating overall good performance. To achieve even better results, it might be necessary to refine the model or consider using additional metrics.

#Importance of variables in our model - Feature Importance Analysis:

```
importance = model.feature_importances_
feature_importance_data = {
    "Feature": x.columns,
    "Importance": importance
}
feature_importance = pd.DataFrame(feature_importance_data)
print(feature_importance)
```

```
                  Feature  Importance
0                     Day    0.030801
1                   Month    0.019927
2  ForecastWindProduction    0.073544
3            SystemLoadEA    0.048612
4                    SMPEA    0.424326
5           ORKTemperature    0.026547
6             ORKWindspeed    0.050219
7             CO2Intensity    0.049424
8    ActualWindProduction    0.075856
9           SystemLoadEP2    0.200745
```

When we rank these features by importance, it's evident that SMPEA holds the highest significance at (42.43%). Following that, we have ActualWindProduction (7.59%) and ForecastWindProduction (7.35%). This indicates that these features play a crucial role in the model's predictions. While the contributions of other features are comparatively lower, they still contribute to the overall performance. Particularly, SystemLoadEP2 stands out significantly at (20.75%) among others. This analysis helps us understand the importance levels of features in the model and provides insights into how the model operates.

#Now let's input all the values of the necessary features that we used to train the model and have a look at the price of the electricity predicted by the model:

```
#Features = [["Day","Month","ForecastWindProduction","SystemLoadEA","SMPEA",
#        "ORKTemperature","ORKWindspeed","CO2Intensity","Actual#Modelimizde değişkenlerin önemi - Feature Importance AnaliziWindProduction",
#        "SystemLoadEP2"]]

features = np.array([[15,6,61.2,3567.2,68.21,14,35.1,700,54.0,3000]])
model.predict(features)
```

```
array([66.3671])
```

#So this is how you can train a machine learning model to predict the prices of electricity.

Random Forest Model with Grid Search:

#First, we import our Python libraries:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

#Now let's build the model.

```
modelg = RandomForestRegressor()
param_grid = {
    'n_estimators': [10, 50, 100, 200, 300],
    'max_depth': [1, 2, 3, 4, 5, 6, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(estimator=modelg, param_grid=param_grid,
                           scoring='neg_mean_squared_error', cv=10, n_jobs=-1)
```

#Let's apply grid search and get the best parameters.

```
modelg.fit(x_train, y_train)
grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
print("Best parametres:", best_params)
```

```
Best parametres: {'max_depth': 20, 'min_samples_split': 5, 'n_estimators': 50}
```

#Let's run the model:

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(max_depth=20, min_samples_split=5, n_estimators=50)
model.fit(x_train,y_train)
```

#Now let's look at the various results of the model we created using grid search.

```
from sklearn.metrics import r2_score
print("Random Forest Grid Search R2 score:")
print(r2_score(y,model.predict(x)))
```

```
Random Forest Grid Search R2 score:
0.8633650582079797
```

#Now let's calculate the mean absoulute error of our model:

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print("Mean Square Error:", mse)
```

```
Mean Square Error: 395.71910127352055
```

#Now let's calculate the mean absoulute error of our model:

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)
```

```
Mean Absolute Error: 8.783487323943657
```

#Now let's input all the values of the necessary features that we used to train the model and have a look at the price of the electricity predicted by the model:

```
features = np.array([[15,6,61.2,3567.2,68.21,14,35.1,700,54.0,3000]])
model.predict(features)
```

```
array([63.43031312])
```

Report:

First, we see that the MSE and MAE values of the two models are the same. This shows that the predictions of both models are at the same average distance from the true values.

However, in terms of R2 score, Random Forest Prediction Model (89.79%) is higher than Random Forest Prediction Model Created with Grid Search (86.34%). This indicates that the first model provides a better fit and fits the data better.

In conclusion, based on your data, the Random Forest Prediction Model seems to perform better than the Random Forest Prediction Model Generated by Grid Search.

Support Vector Regression Estimation Method

#Now let's move to the task of training an electricity price prediction model. Here I will first add all the important features to x and the target column to y, and then I will split the data into training and test sets:

```
x = data[["Day","Month","ForecastWindProduction","SystemLoadEA","SMPEA",
        "ORKTemperature","ORKWindspeed","CO2Intensity","ActualWindProduction",
        "SystemLoadEP2"]]
y = data["SMPEP2"]
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
```

#Converting dataframe slices to Numpy Array

```
X = x.values
Y = y.values
```

#Scaling Data:(Standardization) We want transformation of numbers.

```
from sklearn.preprocessing import StandardScaler

sc1= StandardScaler()
x_scaled = sc1.fit_transform(X)

sc2= StandardScaler()
y_scaled = np.ravel(sc2.fit_transform(Y.reshape(-1,1)))
```

#Creating the support vector regression model:

```
from sklearn.svm import SVR
svr_reg =SVR(kernel='rbf')
svr_reg.fit(x_scaled,y_scaled)
```

#R2 Score value of our Support Vector Regression model:

```
from sklearn.metrics import r2_score
print("Support Vector Regression R2 score:")
print(r2_score(y_scaled,svr_reg.predict(x_scaled)))
```

```
Support Vector Regression R2 score:
0.6330407583657234
```

#Let's look at other metrics:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
y_pred = svr_reg.predict(x_scaled)

mae = mean_absolute_error(y_scaled, y_pred)
mse = mean_squared_error(y_scaled, y_pred)

print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
```

```
Mean Absolute Error (MAE): 0.25956631217315446
Mean Squared Error (MSE): 0.3669592416342767
```

Comparison of Random Forests and Support Vector Regression

1. Random Forest: R2 score: 0.8979

Random forest is generally effective for datasets with a high number of features and in making predictions by aggregating the results of decision trees. A high R2 score indicates that the model fits the data well and its predictions are close to the actual values. However, it may be prone to overfitting.

2. Support Vector Regression: R2 score: 0.6330

SVR is particularly effective for small-sized datasets and low-dimensional features. However, its performance may decrease as the dataset grows. A lower R2 score indicates that the model fits the data less well and its predictions have more error.

This analysis shows that the random forest model performs better than SVR. However, model selection should not rely solely on R2 scores. Other factors such as the characteristics of your dataset, the applicability of the model, computational costs, and other considerations are also important in model selection.

5.Conclusion

In this study, we conducted a detailed examination of two different models, Random Forest and Support Vector Regression (SVR), to forecast electricity prices. After completing the data preprocessing steps, we visualized the dataset to better understand its characteristics. Subsequently, we trained and evaluated both models. The results indicated that the Random Forest model outperformed SVR. The high R2 score (0.8979) of the Random Forest model highlights its ability to handle the complexities of our dataset, while the lower R2 score (0.6330) of SVR suggests its limited capacity to fit the data. However, model selection should not rely solely on R2 scores. Factors such as dataset characteristics, computational costs, applicability, and interpretability should also be considered. These findings underscore the importance of selecting the appropriate model for accurate and reliable electricity price predictions. Future research could explore alternative feature selection techniques or employ more sophisticated models to further enhance predictive performance. This study contributes to the comparative evaluation of models used in electricity price forecasting, providing valuable insights for future researchers.

6.Recommendations

1. More Comprehensive Model Evaluation: In this study, we evaluated model performance solely based on R2 scores. However, future research should consider using different criteria for model selection, such as information criteria like AIC or BIC, and techniques like cross-validation. This would provide a more thorough assessment of how well the model generalizes to real-world data.

2. Feature Selection and Model Complexity: The choice of features in our dataset can significantly impact model performance. In future studies, it is recommended to explore different feature selection techniques and adjust model complexity (e.g., optimizing hyperparameters like the number of trees or depth of trees in Random Forest) to improve predictive accuracy.

3. More Data: To enhance model performance, using larger and more diverse datasets is advised. This can help the model make more generalizable and reliable predictions.

4. Exploration of Alternative Models: While we focused on two models, Random Forest and SVR, in this study, future research could explore different machine learning techniques (e.g., Gradient Boosting or Neural Networks). This exploration could provide insights into alternative modeling approaches that may yield improved results.

5. Consideration of Industrial Applications: Electricity price forecasting has many industrial applications in the energy sector. Therefore, future research should consider collaborating with industry stakeholders to obtain more tangible results that address real-world needs.

References

1.Kaggle. (n.d.). Electricity Price Forecasting Dataset. Retrieved from https://www.kaggle.com/dataset/electricity-price-forecasting

2. Energy Information Administration. (n.d.). Electricity Data Browser. Retrieved from https://www.eia.gov/electricity/data/browser/

3. ArXiv. (n.d.). Machine Learning Methods for Electricity Price Forecasting: A Review. Retrieved from https://arxiv.org/abs/2008.02947

4. Medium. (n.d.). Electricity Price Forecasting Using Machine Learning. Retrieved from https://medium.com/datadriveninvestor/electricity-price-forecasting-using-machine-learning-1b56dd7a2eef

Appendices

A. Description of the Dataset

This appendix provides a detailed description of the electricity price dataset used in the study. It includes information about the structural features of the dataset, the columns, and the meaning of each column.

B. Model Parameters

This appendix includes the parameters of the Random Forest and Support Vector Regression (SVR) models used. It provides information about the values of the selected parameters for each model and the reasons for selecting these parameters.

C. Code Samples

This appendix contains code samples of the models used. Written in Python, the code samples demonstrate the step-by-step process of data preprocessing, model training, and evaluation.

D. Additional Analyses

This appendix presents the results of additional analyses that are beyond the scope of the project but are deemed interesting or significant. It provides additional information to expand or deepen the main findings of the project.