ELECTRICITY PRICES PREDICTION

TEAM MEMBER

ANANYA A

Phase 5 Submission Document

Project : ELECTRICITY PRICES PREDICTION

Topic: *Documentation and Submission*

Introduction:

In a world that thrives on energy as the lifeblood of modern society, the dynamics of electricity pricing have a profound impact on both consumers and producers. The ability to accurately predict electricity prices holds immense significance for various stakeholders, ranging from individual households seeking to manage their energy costs to utility companies striving to optimize resource allocation and policy makers working towards a sustainable energy future.

Electricity price prediction is not merely a matter of financial prudence; it's a critical element in the broader landscape of energy management and sustainability. It empowers us to make informed decisions, reduce energy waste, and align our consumption patterns with fluctuating supply and demand dynamics.

This discussion or project aims to delve into the intricate world of electricity price prediction. We will explore the multifaceted factors that influence pricing, from supply and demand patterns to environmental conditions and regulatory policies. Through the lens of data-driven approaches, machine learning, and statistical modeling, we will uncover the methodologies and tools used to forecast electricity prices with increasing precision.

Throughout our journey, we will address the real-world implications of electricity price prediction. From enabling cost-efficient strategies for businesses to encouraging renewable energy adoption and grid optimization, the ability to foresee price trends stands as a linchpin in the pursuit of an efficient, sustainable, and equitable energy ecosystem.

As we embark on this exploration of electricity price prediction, we invite you to discover the intricate interplay between data, technology, and the future of energy management. Join us as we uncover the valuable insights hidden within the numbers and explore the potential to make more informed, economically sound, and environmentally responsible decisions in an electrified world.

Given Dataset

1	DateTime	Holiday	HolidayFla	DayOfWe	WeekOfYe	Day	Month	Year	PeriodOfD	ForecastW	SystemLo	SMPEA	ORKTemp	ORKWind	CO2Intens	ActualWir	SystemLo	SMPEP2
2	########	None	0	1	44	1	11	2011	0	315.31	3388.77	49.26	6	9.3	600.71	356	3159.6	54.32
3	########	None	0	1	44	1	11	2011	1	321.8	3196.66	49.26	6	11.1	605.42	317	2973.01	54.23
4	########	None	0	1	44	1	11	2011	2	328.57	3060.71	49.1	5	11.1	589.97	311	2834	54.23
5	########	None	0	1	44	1	11	2011	3	335.6	2945.56	48.04	6	9.3	585.94	313	2725.99	53.47
6	########	None	0	1	44	1	11	2011	4	342.9	2849.34	33.75	6	11.1	571.52	346	2655.64	39.87
7	########	None	0	1	44	1	11	2011	5	342.97	2810.01	33.75	5	11.1	562.61	342	2585.99	39.87
8	########	None	0	1	44	1	11	2011	6	343.18	2780.52	33.75	5	7.4	545.81	336	2561.7	39.87
9	########	None	0	1	44	1	11	2011	7	343.46	2762.67	33.75	5	9.3	539.38	338	2544.33	39.87
10	########	None	0	1	44	1	11	2011	8	343.88	2766.63	33.75	4	11.1	538.7	347	2549.02	39.87
11	########	None	0	1	44	1	11	2011	9	344.39	2786.8	33.75	4	7.4	540.39	338	2547.15	39.87
12	########	None	0	1	44	1	11	2011	10	345.02	2817.59	33.75	4	7.4	532.3	372	2584.58	39.87
13	########	None	0	1	44	1	11	2011	11	342.23	2895.62	47.42	5	5.6	547.57	361	2641.37	39.87
14	########	None	0	1	44	1	11	2011	12	339.22	3039.67	44.31	5	3.7	556.14	383	2842.19	51.45
15	########	None	0	1	44	1	11	2011	13	335.39	3325.1	45.14	5	3.7	590.34	358	3082.97	51.45
16	########	None	0	1	44	1	11	2011	14	330.95	3661.02	46.25	4	9.3	596.22	402	3372.55	52.82
17	########	None	0	1	44	1	11	2011	15	325.93	4030	52.84	5	3.7	581.52	368	3572.64	53.65
18	########	None	0	1	44	1	11	2011	16	320.91	4306.54	59.44	5	5.6	577.27	361	3852.42	54.21
19	########	None	0	1	44	1	11	2011	17	365.15	4438.05	62.15	6	5.6	568.76	340	4116.03	58.33
20	########	None	0	1	44	1	11	2011	18	410.55	4585.84	61.81	8	7.4	560.79	358	4345.42	58.33
21	########	None	0	1	44	1	11	2011	19	458.56	4723.93	61.88	9	7.4	542.8	339	4427.29	58.33
22	########	None	0	1	44	1	11	2011	20	513.17	4793.6	61.46	?	?	535.37	324	4460.41	58.33
23	########	None	0	1	44	1	11	2011	21	573.36	4829.44	61.28	11	13	532.52	335	4493.22	58.27

Some list of Tools and Softwares commonly used in this process:

1. Programming Language: *Python is the most popular language for machine learning due to its extensive libraries and frameworks. You can use libraries like NumPy, pandas, scikit-learn, and more.*

2. Integrated Development Environment (IDE): *IDEs like PyCharm, VSCode, and RStudio provide a user-friendly environment for writing and debugging code*

3. Python: *Python is the go-to programming language for data analysis and machine learning. Many libraries and frameworks are available to work with data and build predictive models, including NumPy, Pandas, Scikit-Learn, and TensorFlow.*

4.Jupyter Notebook: *Jupyter Notebook is an interactive environment widely used for data exploration, analysis, and model development. It allows for easy experimentation and documentation.*

5. R: *R is another programming language used for statistical analysis and data visualization. It has a strong community of users in the data science field.*

6. SQL Databases: *Databases like MySQL, PostgreSQL, or SQLite are often used to store historical datas,weather information,demand statistics,generation capacity and other market-related variables in structured tables.*

7.Web Scraping Tools: *Tools like Beautiful Soup and Scrapy are used to collect data from electricity prices prediction's website or other sources. Web scraping helps to gather relevant data from various sources,aiding in the creation of comparehensive datasets for analysis.*

8.Data Visualization Tools: *Tools like Matplotlib, Seaborn, and Plotly are used to create visualizations to better understand the data and the relationships between different variables.*

9.Machine Learning Libraries: *Scikit-Learn, XGBoost, LightGBM, and Keras (for deep learning) are commonly used for building predictive models to estimate electricity prices prediction.*

10. Feature Engineering Tools: *Feature engineering is crucial in creating relevant predictors. Python libraries like Feature-engine and Featuretools can be helpful.*

11. Text Analysis Libraries: *Natural language processing (NLP) libraries like NLTK and spaCy are used for sentiment analysis and text-based features extraction from reviews and plot summaries.*

12. Big Data Tools: *In cases where large datasets are involved, tools like Apache Spark can be used for distributed data processing and machine learning.*

13. Cloud Computing Platforms: *Services like Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure provide cloud-based resources for handling large-scale data and machine learning tasks.*

14. Machine Learning Platforms: *AutoML platforms like Google AutoML and H2O.ai provide automated machine learning solutions that can be used for model building and optimization.*

15. Version Control: *Tools like Git and GitHub/GitLab are essential for tracking changes in code and collaborating with other team members on the project.*

16. Containerization: *Docker and container orchestration tools like Kubernetes can be used for packaging and deploying machine learning models.*

17. Data Analysis and Visualization Software: *Tools like Tableau, Power BI, or even Excel can be used for additional data analysis and visualization, especially in the context of reporting and sharing results.*

18. Statistical Analysis Software: *Software like IBM SPSS or SAS can be used for advanced statistical analysis, especially in more traditional statistical modeling approaches.*

1.DESIGN THINKING AND PRESENT IN FORM OF DOCUMENT

1. Empathize: Understanding Stakeholder Needs

- *Identify stakeholders (utility companies, consumers, traders, regulators) and understand their unique requirements regarding electricity price prediction.*

- *Conduct interviews, surveys, and workshops to empathize with their pain points, expectations, and desired outcomes.*

2. Define: Problem Statement and Objectives

- *Consolidate the insights gained from stakeholder interactions to define a clear problem statement.*
- *Outline specific objectives for the predictive model, considering accuracy, real-time forecasting, adaptability, and user-friendly outputs.*

3. Ideate: Exploring Solutions and Methodologies

- *Brainstorm various approaches to electricity price prediction, considering machine learning algorithms, statistical models, and data sources.*
- *Explore different techniques for feature engineering, model selection, and evaluation methods for accurate predictions.*

4. Prototype: Model Development and Testing

- *Develop prototypes of predictive models using historical electricity pricing data.*
- *Implement machine learning algorithms (e.g., regression, time series analysis, neural networks) and validate these models with test datasets.*

5. Test: Evaluation and Feedback Gathering

- *Evaluate the prototypes using various metrics such as Mean Absolute Error, Root Mean Squared Error, or accuracy metrics relevant to electricity price forecasting.*
- *Gather feedback from stakeholders and adjust the models based on their insights and validation results.*

6. Implement: Deployment and Integration

- *Implement the finalized predictive model into the operational workflow, ensuring seamless integration with existing systems.*
- *Develop an interface or API for easy access and utilization by stakeholders.*

7. Iterate: Continuous Improvement and Adaptation

- *Establish a feedback loop to continuously refine and improve the predictive model based on new data, changing market conditions, and stakeholder feedback.*
- *Consider the incorporation of advanced techniques and emerging data sources for enhanced accuracy.*

Design Into Innovation

Data Collection and Preprocessing:

- *Gather historical data on electricity prices.This data should include timestamp,location and the corresponding electricity prices.*
- *To prepare the data for modeling, we performed the following preprocessing steps;*

- *Handled missing data.*
- *Removed outliers.*
- *Converted date and time into suitable formats.*
- *Normalized prices for consistent scaling.*

Exploratory Data Analysis (EDA):

- *Visualize and analyze the dataset to gain insights into the relationships between variables.*
- *Identify correlations and patterns that can inform feature selection and engineering.*
- *Present various data visualizations to gain insights into the dataset.*
- *Explore correlations between features and the target variable (electricity prices).*
- *Discuss any significant findings from the EDA phase that inform feature selection.*

Feature Engineering:

- *Create new features or transform existing ones to capture valuable information.*
- *Explain the process of creating new features or transforming existing ones.*
- *Showcase domain-specific feature engineering, such as proximity scores or composite indicators.*
- *Emphasize the impact of engineered features on model performance.*

- *Create relevant features that could impact electricity prices,such as weather data(temperature,humidity),economic indicators,holidays or even events like major sports games.*

Model Training:

The selected model was trained on a portion of the dataset, and hyperparameters were tuned for optimal performance.

Model Evaluation and Selection:

- *The model's performance was evaluated using the test dataset. Key metrics included:*
    - *Mean Squared Error (MSE)*
    - *R-squared (R2)*
- *We opted for a machine learning approach to predict electricity prices. The selected model was known for its ability to handle time series data effectively.*

PYTHON PROGRAM:

```
#Let's load the relevant libraries

import numpy as np
import pandas as pd
from sklearn.model_selection import
train_test_split,GridSearchCV,RandomizedSearchCV
from sklearn.metrics import mean_squared_error,r2_score
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn import model_selection
```

```
from sklearn.linear_model import
Ridge,Lasso,RidgeCV,LassoCV,ElasticNet,ElasticNetCV,LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn import neighbors
from sklearn.svm import SVR
import warnings
warnings.filterwarnings("ignore")
```

Data Loading And Preprocessing

1. Data Collection:

- **Acquiring Data:** *Collect historical electricity price data from various sources, such as energy market platforms, government agencies (like EIA), or commercial data providers.*
- **Additional Data Sources:** *Gather supplementary data like weather patterns, demand trends, generation capacities, fuel prices, economic indicators, or regulatory changes that might impact electricity prices.*

2. Data Cleaning:

- **Handling Missing Values:** *Address missing data by imputation or removal, ensuring data completeness.*
- **Removing Outliers:** *Identify and handle outliers that might negatively impact the accuracy of predictive models.*
- **Data Formatting:** *Standardize data formats and units to ensure consistency across different sources.*

3. Feature Engineering:

- **Temporal Features:** *Extract time-related features such as day of the week, month, season, or holidays, which might influence pricing.*
- **Lag Features:** *Create lag features to capture historical prices or trends, aiding in time series analysis.*
- **Aggregated Features:** *Generate aggregated statistics (mean, median, standard deviation) for different time windows.*

4. Data Transformation:

- **Normalization or Scaling:** *Normalize numerical features to a similar scale to prevent dominance of certain variables.*
- **Categorical Data Encoding:***Convert categorical variables into a numerical format suitable for modeling (e.g., one-hot encoding).*

5. Splitting Data:

- **Training and Test Sets:** *Divide the data into training and testing sets for model validation and performance assessment.*
- **Time-Based Split:** *Consider time-based splitting to maintain the temporal order in data for time series analysis.*

6. Data Validation:

- **Cross-Validation:** *Implement cross-validation techniques to validate model performance across different subsets of the dataset.*
- **Validation Metrics:** *Use appropriate metrics (e.g., Mean Absolute Error, Root Mean Squared Error) to assess model accuracy.*

7. Data Preprocessing for Machine Learning Models:

- **Model-Specific Preprocessing:** *Prepare data according to the requirements of chosen machine learning algorithms (e.g., reshaping for LSTM models in neural networks).*
- **Handling Imbalanced Data:** *Address class imbalance if present, ensuring the model's ability to predict both high and low price variations.*

Program:

Data Loading:

In[1]:

```
df=pd.read_csv("C:\Users\MY PC\Downloads\electricity.csv")
df.head()
```

Out[1]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOfDa
0	01/11/2011 00:00	None	0	1	44	1	11	2011	0
1	01/11/2011 00:30	None	0	1	44	1	11	2011	1
2	01/11/2011 01:00	None	0	1	44	1	11	2011	2
3	01/11/2011 01:30	None	0	1	44	1	11	2011	3
4	01/11/2011 02:00	None	0	1	44	1	11	2011	4

In[2]:

```
df.tail()
```

Out[2]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Perioc
38009	31/12/2013 21:30	New Year's Eve	1	1	1	31	12	2013	43
38010	31/12/2013 22:00	New Year's Eve	1	1	1	31	12	2013	44
38011	31/12/2013 22:30	New Year's Eve	1	1	1	31	12	2013	45
38012	31/12/2013 23:00	New Year's Eve	1	1	1	31	12	2013	46
38013	31/12/2013 23:30	New Year's Eve	1	1	1	31	12	2013	47

EDA:

In[3]:

```
df.shape
```

Out[3]:

(38014, 18)

In[4]:

```
#columns
df.columns
```

Out[4]:

```
Index(['DateTime', 'Holiday', 'HolidayFlag', 'DayOfWeek', 'WeekOfYear', 'Day',
       'Month', 'Year', 'PeriodOfDay', 'ForecastWindProduction',
       'SystemLoadEA', 'SMPEA', 'ORKTemperature', 'ORKWindspeed',
```

```
 'CO2Intensity', 'ActualWindProduction', 'SystemLoadEP2', 'SMPEP2'],
dtype='object')
```

In[5]:

```
# datatypes
df.dtypes
```

Out[5]:

```
DateTime                  object
Holiday                   object
HolidayFlag                int64
DayOfWeek                  int64
WeekOfYear                 int64
Day                        int64
Month                      int64
Year                       int64
PeriodOfDay                int64
ForecastWindProduction    object
SystemLoadEA              object
SMPEA                     object
ORKTemperature            object
ORKWindspeed              object
CO2Intensity              object
ActualWindProduction      object
SystemLoadEP2             object
SMPEP2                    object
dtype: object
```

In[6]:

```
#structural information
df.info()
```

Out[6]:

```
<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                 38014 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
```

In[7]:

```
# dataset summary
df.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
HolidayFlag	38014.0	0.040406	0.196912	0.0	0.0	0.0	0.00	1.0
DayOfWeek	38014.0	2.997317	1.999959	0.0	1.0	3.0	5.00	6.0
WeekOfYear	38014.0	28.124586	15.587575	1.0	15.0	29.0	43.00	52.0
Day	38014.0	15.739412	8.804247	1.0	8.0	16.0	23.00	31.0
Month	38014.0	6.904246	3.573696	1.0	4.0	7.0	10.00	12.0
Year	38014.0	2012.383859	0.624956	2011.0	2012.0	2012.0	2013.00	2013.0
PeriodOfDay	38014.0	23.501105	13.853108	0.0	12.0	24.0	35.75	47.0

In[8]:

```
df.describe([0.1,0.25,0.5,0.65,0.75,0.9,0.95]).T
```

Out[8]:

	count	mean	std	min	10%	25%	50%	65%
HolidayFlag	38014.0	0.040406	0.196912	0.0	0.0	0.0	0.0	0.0
DayOfWeek	38014.0	2.997317	1.999959	0.0	0.0	1.0	3.0	4.0
WeekOfYear	38014.0	28.124586	15.587575	1.0	6.0	15.0	29.0	37.0
Day	38014.0	15.739412	8.804247	1.0	4.0	8.0	16.0	20.0
Month	38014.0	6.904246	3.573696	1.0	2.0	4.0	7.0	9.0
Year	38014.0	2012.383859	0.624956	2011.0	2012.0	2012.0	2012.0	2013.0
PeriodOfDay	38014.0	23.501105	13.853108	0.0	4.0	12.0	24.0	31.0

In[9]:

```
# unique value counts
df.nunique()
```

Out[9]:

```
DateTime                  38014
Holiday                      15
```

```
HolidayFlag                    2
DayOfWeek                      7
WeekOfYear                    52
Day                           31
Month                         12
Year                           3
PeriodOfDay                   48
ForecastWindProduction     29312
SystemLoadEA               36166
SMPEA                       8661
ORKTemperature                32
ORKWindspeed                  53
CO2Intensity               25115
ActualWindProduction        2940
SystemLoadEP2              36171
SMPEP2                      9277
dtype: int64
```

In[10]:

```
col=["Holiday","HolidayFlag","DayOfWeek","WeekOfYear","Day","Mo
nth",
  "Year","PeriodOfDay","ORKTemperature"]

for i in col:
  print(df[i].value_counts())
  print("*"*30)
```

Out[10]:

```
None                     36478
Christmas Eve              144
Christmas                  144
St Stephen's Day           144
New Year's Eve             144
New Year's Day              96
St Patrick's Day            96
Good Friday                 96
Holy Saturday               96
Easter                      96
Easter Monday               96
May Day                     96
June Bank Holiday           96
August Bank Holiday         96
```

```
October Bank Holiday       96
Name: Holiday, dtype: int64
******************************
0    36478
1     1536
Name: HolidayFlag, dtype: int64
******************************
1    5472
2    5424
3    5424
4    5424
5    5424
0    5424
6    5422
Name: DayOfWeek, dtype: int64
******************************
45    1008
46    1008
47    1008
48    1008
49    1008
50    1008
51    1008
52    1008
44     960
1      768
25     672
26     672
33     672
27     672
28     672
29     672
30     672
31     672
32     672
37     672
34     672
35     672
36     672
23     672
38     672
39     672
40     672
41     672
42     672
24     672
18     672
22     672
10     672
2      672
3      672
4      672
5      672
6      672
7      672
8      672
```

```
9      672
11     672
21     672
13     672
14     672
15     672
16     672
17     672
19     672
20     672
43     672
12     670
Name: WeekOfYear, dtype: int64
******************************
1     1248
15    1248
28    1248
27    1248
26    1248
24    1248
23    1248
22    1248
21    1248
20    1248
19    1248
18    1248
17    1248
2     1248
16    1248
14    1248
13    1248
12    1248
11    1248
10    1248
9     1248
8     1248
7     1248
6     1248
5     1248
4     1248
3     1248
25    1246
29    1200
30    1152
31     720
Name: Day, dtype: int64
******************************
12    4464
11    4320
1     2976
5     2976
7     2976
8     2976
10    2976
3     2974
4     2880
```

```
6     2880
9     2880
2     2736
Name: Month, dtype: int64
******************************
2012    17566
2013    17520
2011     2928
Name: Year, dtype: int64
******************************
0     792
1     792
26    792
27    792
28    792
29    792
30    792
31    792
32    792
33    792
34    792
35    792
36    792
37    792
38    792
39    792
40    792
41    792
42    792
43    792
44    792
45    792
46    792
25    792
24    792
23    792
22    792
4     792
5     792
6     792
7     792
8     792
9     792
10    792
11    792
12    792
13    792
14    792
15    792
16    792
17    792
18    792
19    792
20    792
21    792
47    792
```

```
3     791
2     791
Name: PeriodOfDay, dtype: int64
******************************
9.00     3525
10.00    3230
8.00     3225
11.00    3017
7.00     2894
12.00    2712
6.00     2617
5.00     2027
13.00    2009
15.00    1883
14.00    1855
4.00     1580
16.00    1451
3.00     1399
17.00    1001
2.00      953
18.00     592
1.00      531
19.00     330
?         295
0.00      213
20.00     195
21.00     135
-1.00     103
22.00      75
23.00      64
24.00      43
-2.00      30
-3.00      14
25.00      13
-4.00       2
-0.00       1
Name: ORKTemperature, dtype: int64
******************************
```

In[11]:

we have accessed the class counts for each category

In [12]:

#Let's convert string values to floats;

In [13]:

#pd.to_numeric?

In [14]:

```
linkcode
# convert
df["ForecastWindProduction"]=pd.to_numeric(df["ForecastWindProduction"], errors= 'coerce')
df["SystemLoadEA"] = pd.to_numeric(df["SystemLoadEA"], errors= 'coerce')
df["SMPEA"] = pd.to_numeric(df["SMPEA"], errors= 'coerce')
df["ORKTemperature"] = pd.to_numeric(df["ORKTemperature"], errors= 'coerce')
df["ORKWindspeed"] = pd.to_numeric(df["ORKWindspeed"], errors= 'coerce')
df["CO2Intensity"] = pd.to_numeric(df["CO2Intensity"], errors= 'coerce')
df["ActualWindProduction"] = pd.to_numeric(df["ActualWindProduction"], errors= 'coerce')
df["SystemLoadEP2"] = pd.to_numeric(df["SystemLoadEP2"], errors= 'coerce')
df["SMPEP2"] = pd.to_numeric(df["SMPEP2"], errors= 'coerce')
```

In[15]:

```
df.info()
```

Out[15]:

```
<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                 38014 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
dtypes: float64(9), int64(7), object(2)
memory usage: 5.2+ MB
```

In[16]:

```
df.describe([0.05,0.1,0.25,0.35,0.5,0.65,0.75,0.9,0.95,0.98]).T
```

Out[16]:

	count	mean	std	min	5%	10%	25%	35%	50%	65%	75%	90%	95%	98%	max
HolidayFlag	38014.0	0.040406	0.196912	0.00	0.0000	0.000	0.0000	0.0000	0.000	0.000	0.0000	0.000	0.0000	1.0000	1.00
DayOfWeek	38014.0	2.997317	1.999959	0.00	0.0000	0.000	1.0000	2.0000	3.000	4.000	5.0000	6.000	6.0000	6.0000	6.00
WeekOfYear	38014.0	28.124586	15.587575	1.00	3.0000	6.000	15.0000	20.0000	29.000	37.000	43.0000	49.000	51.0000	52.0000	52.00
Day	38014.0	15.739412	8.804247	1.00	2.0000	4.000	8.0000	11.0000	16.000	20.000	23.0000	28.000	29.0000	30.0000	31.00
Month	38014.0	6.904246	3.573696	1.00	1.0000	2.000	4.0000	5.0000	7.000	9.000	10.0000	12.000	12.0000	12.0000	12.00
Year	38014.0	2012.383859	0.624956	2011.00	2011.0000	2012.000	2012.0000	2012.0000	2012.000	2013.000	2013.0000	2013.000	2013.0000	2013.0000	2013.00

	count	mean	std	min	5%	10%	25%	35%	50%	65%	75%	90%	95%	98%	max
PeriodOfDay	38014.0	23.501105	13.853108	0.00	2.0000	4.000	12.0000	16.0000	24.000	31.000	35.7500	43.000	45.0000	47.0000	47.00
ForecastWindProduction	38009.0	544.261451	414.364629	0.68	52.3220	80.376	189.6700	279.3000	441.980	649.992	839.4600	1206.110	1352.4400	1441.4580	1680.00
SystemLoadEA	38012.0	4020.085019	860.476866	2183.94	2626.2495	2812.033	3281.2075	3646.8335	4103.600	4467.160	4638.5325	5093.912	5367.0760	5713.1028	6492.91
SMPEA	38012.0	62.720388	32.252334	0.00	33.8200	38.360	45.5300	49.1900	55.230	63.610	70.3200	90.627	110.2290	154.8378	587.58
ORKTemperature	37719.0	9.626369	4.439934	-4.00	3.0000	4.000	6.0000	8.0000	9.000	11.000	13.0000	16.000	17.0000	19.0000	25.00
ORKWindspeed	37715.0	19.211770	9.571311	0.00	5.6000	7.400	13.0000	14.8000	18.500	22.200	24.1000	31.500	37.0000	42.6000	75.90
CO2Intensity	38007.0	479.373040	85.354706	0.00	336.0930	367.216	421.1050	446.8800	480.310	512.620	537.5200	587.870	619.1910	656.0452	842.88
ActualWindProduction	38009.0	520.762819	378.282975	1.00	43.0000	77.000	199.0000	287.8000	445.000	637.000	793.0000	1098.000	1243.0000	1349.0000	1769.00
SystemLoadEP2	38012.0	3785.973841	843.269455	1809.96	2449.5410	2599.070	3058.2775	3403.7040	3865.745	4239.709	4427.5900	4830.294	5087.5430	5397.7758	6309.75

	count	mean	std	min	5%	10%	25%	35%	50%	65%	75%	90%	95%	98%	max
SMPEP2	38012.0	64.136823	35.415036	-47.74	33.1900	37.922									

In[17]:

```
df.sort_values("ForecastWindProduction",ascending=False).head(10)
```

Out[17]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Perio
24326	21/03/2013 20:00	None	0	3	12	21	3	2013	40
24327	21/03/2013 20:30	None	0	3	12	21	3	2013	41
24325	21/03/2013 19:30	None	0	3	12	21	3	2013	39
24328	21/03/2013 21:00	None	0	3	12	21	3	2013	42
24324	21/03/2013 19:00	None	0	3	12	21	3	2013	38
37342	18/12/2013 00:00	None	0	2	51	18	12	2013	0
25476	14/04/2013 19:00	None	0	6	15	14	4	2013	38
24329	21/03/2013 21:30	None	0	3	12	21	3	2013	43
24323	21/03/2013 18:30	None	0	3	12	21	3	2013	37
24330	21/03/2013 22:00	None	0	3	12	21	3	2013	44

In[18]:

```
df.sort_values("ForecastWindProduction").head(10)
```

Out[18]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Perioc
29729	12/07/2013 09:30	None	0	4	28	12	7	2013	19
29728	12/07/2013 09:00	None	0	4	28	12	7	2013	18
29727	12/07/2013 08:30	None	0	4	28	12	7	2013	17
29730	12/07/2013 10:00	None	0	4	28	12	7	2013	20
29726	12/07/2013 08:00	None	0	4	28	12	7	2013	16
29731	12/07/2013 10:30	None	0	4	28	12	7	2013	21
29725	12/07/2013 07:30	None	0	4	28	12	7	2013	15
29732	12/07/2013 11:00	None	0	4	28	12	7	2013	22
27150	19/05/2013 16:00	None	0	6	20	19	5	2013	32
27149	19/05/2013 15:30	None	0	6	20	19	5	2013	31

In[19]:

```
df.sort_values("ORKWindspeed",ascending=False).head(10)
# highest
```

Out[19]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Per
37770	26/12/2013 22:00	St Stephen's Day	1	3	52	26	12	2013	44
37767	26/12/2013 20:30	St Stephen's Day	1	3	52	26	12	2013	41
25619	17/04/2013 18:30	None	0	2	16	17	4	2013	37
37768	26/12/2013 21:00	St Stephen's Day	1	3	52	26	12	2013	42
25617	17/04/2013 17:30	None	0	2	16	17	4	2013	35
37775	27/12/2013 00:30	None	0	4	52	27	12	2013	1
25621	17/04/2013 19:30	None	0	2	16	17	4	2013	39
37779	27/12/2013 02:30	None	0	4	52	27	12	2013	5
37778	27/12/2013 02:00	None	0	4	52	27	12	2013	4
37771	26/12/2013 22:30	St Stephen's Day	1	3	52	26	12	2013	45

In[20]:

```
df.sort_values("ORKWindspeed").head(10) # lowest
```

Out[20]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Perioc
35131	01/11/2013 22:30	None	0	4	44	1	11	2013	45
18299	16/11/2012 06:30	None	0	4	46	16	11	2012	13
9320	13/05/2012 05:00	None	0	6	19	13	5	2012	10
145	04/11/2011 00:30	None	0	4	44	4	11	2011	1
35477	09/11/2013 03:30	None	0	5	45	9	11	2013	7
18258	15/11/2012 10:00	None	0	3	46	15	11	2012	20
21265	17/01/2013 01:30	None	0	3	3	17	1	2013	3
17559	31/10/2012 20:30	None	0	2	44	31	10	2012	41
19981	21/12/2012 07:30	None	0	4	51	21	12	2012	15
18294	16/11/2012 04:00	None	0	4	46	16	11	2012	8

In[21]:

```
df.sort_values("SMPEA",ascending=False).head(10)  # highest
```

Out[21]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Perioc
1474	01/12/2011 17:00	None	0	3	48	1	12	2011	34
6951	24/03/2012 19:30	None	0	5	12	24	3	2012	39
1667	05/12/2011 17:30	None	0	0	49	5	12	2011	35
37857	28/12/2013 17:30	None	0	5	52	28	12	2013	35
37907	29/12/2013 18:30	None	0	6	52	29	12	2013	37
37569	22/12/2013 17:30	None	0	6	51	22	12	2013	35
37137	13/12/2013 17:30	None	0	4	50	13	12	2013	35
17698	03/11/2012 18:00	None	0	5	44	3	11	2012	36
23892	12/03/2013 19:00	None	0	1	11	12	3	2013	38
16963	19/10/2012 10:30	None	0	4	42	19	10	2012	21

In[22]:

```
df.sort_values("SMPEA").head(10) # lowest
```

Out[22]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Period(
2218	17/12/2011 05:00	None	0	5	50	17	12	2011	10
2025	13/12/2011 04:30	None	0	1	50	13	12	2011	9
1398	30/11/2011 03:00	None	0	2	48	30	11	2011	6
442	10/11/2011 05:00	None	0	3	45	10	11	2011	10
441	10/11/2011 04:30	None	0	3	45	10	11	2011	9
2217	17/12/2011 04:30	None	0	5	50	17	12	2011	9
3130	05/01/2012 05:00	None	0	3	1	5	1	2012	10
2023	13/12/2011 03:30	None	0	1	50	13	12	2011	7
3845	20/01/2012 02:30	None	0	4	3	20	1	2012	5
3850	20/01/2012 05:00	None	0	4	3	20	1	2012	10

In[23]:

```
df.groupby("Holiday")[["Month","Year"]].describe().T
```

Out[23]:

	Holiday	August Bank Holiday	Christmas	Christmas Eve	Easter	Easter Monday	Gooc
Month	count	96.000000	144.000000	144.000000	96.000000	96.000000	96.0
	mean	8.000000	12.000000	12.000000	3.500000	4.000000	3.50
	std	0.000000	0.000000	0.000000	0.502625	0.000000	0.50
	min	8.000000	12.000000	12.000000	3.000000	4.000000	3.00
	25%	8.000000	12.000000	12.000000	3.000000	4.000000	3.00
	50%	8.000000	12.000000	12.000000	3.500000	4.000000	3.50
	75%	8.000000	12.000000	12.000000	4.000000	4.000000	4.00
	max	8.000000	12.000000	12.000000	4.000000	4.000000	4.00
Year	count	96.000000	144.000000	144.000000	96.000000	96.000000	96.0
	mean	2012.500000	2012.000000	2012.000000	2012.500000	2012.500000	201
	std	0.502625	0.819346	0.819346	0.502625	0.502625	0.50
	min	2012.000000	2011.000000	2011.000000	2012.000000	2012.000000	201
	25%	2012.000000	2011.000000	2011.000000	2012.000000	2012.000000	201
	50%	2012.500000	2012.000000	2012.000000	2012.500000	2012.500000	201
	75%	2013.000000	2013.000000	2013.000000	2013.000000	2013.000000	201
	max	2013.000000	2013.000000	2013.000000	2013.000000	2013.000000	201

In[24]:

```
df[df.SMPEP2==-47.74]
```

Out[24]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Period(
3131	05/01/2012 05:30	None	0	3	1	5	1	2012	11

In[25]:

```
df[df.SMPEP2<0]
```

Out[25]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Perioc
3131	05/01/2012 05:30	None	0	3	1	5	1	2012	11
19872	19/12/2012 01:00	None	0	2	51	19	12	2012	2
19877	19/12/2012 03:30	None	0	2	51	19	12	2012	7

In[26]:

```
df[df.SMPEP2==1000]
```

Out[26]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Perioc
23193	26/02/2013 05:30	None	0	1	9	26	2	2013	11

In[27]:

```
df[df.ORKTemperature<0]
```

Out[27]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Perioc
4463	01/02/2012 23:30	None	0	2	5	1	2	2012	47
4464	02/02/2012 00:00	None	0	3	5	2	2	2012	0
4467	02/02/2012 01:30	None	0	3	5	2	2	2012	3
21502	22/01/2013 00:00	None	0	1	4	22	1	2013	0
21503	22/01/2013 00:30	None	0	1	4	22	1	2013	1
...	...	...	...	...	...	...	...	...	...
36154	23/11/2013 06:00	None	0	5	47	23	11	2013	12
36155	23/11/2013 06:30	None	0	5	47	23	11	2013	13
36156	23/11/2013 07:00	None	0	5	47	23	11	2013	14
36157	23/11/2013 07:30	None	0	5	47	23	11	2013	15
36159	23/11/2013 08:30	None	0	5	47	23	11	2013	17

In[28]:

```
df[df.ORKTemperature==25]
```

Out[28]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	Period
29498	07/07/2013 14:00	None	0	6	27	7	7	2013	28
29594	09/07/2013 14:00	None	0	1	28	9	7	2013	28
29637	10/07/2013 11:30	None	0	2	28	10	7	2013	23
29638	10/07/2013 12:00	None	0	2	28	10	7	2013	24
29784	13/07/2013 13:00	None	0	5	28	13	7	2013	26
29785	13/07/2013 13:30	None	0	5	28	13	7	2013	27
29786	13/07/2013 14:00	None	0	5	28	13	7	2013	28
29787	13/07/2013 14:30	None	0	5	28	13	7	2013	29
29788	13/07/2013 15:00	None	0	5	28	13	7	2013	30
29789	13/07/2013 15:30	None	0	5	28	13	7	2013	31
29790	13/07/2013 16:00	None	0	5	28	13	7	2013	32

Data Preprocessing:

In[1]:

```
# missing value query
df.isna().sum()
```

Out[1]:

```
DateTime        0
Holiday         0
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
```

In[2]:

```
cat_list=[]
num_list=[]

for i in df.columns:
    unique_val=len(df[i].unique())

    if unique_val<40:
        cat_list.append(i)
    else:
        num_list.append(i)
```

In [3]:

```
cat_list.append("WeekOfYear")
```

In [4]:

```
cat_list
```

Out[4]:

```
['Holiday',
 'HolidayFlag',
```

```
 'DayOfWeek',
 'Day',
 'Month',
 'Year',
 'ORKTemperature',
 'WeekOfYear']
```

In [5]:

```
num_list
```

Out[5]:

```
['DateTime',
 'WeekOfYear',
 'PeriodOfDay',
 'ForecastWindProduction',
 'SystemLoadEA',
 'SMPEA',
 'ORKWindspeed',
 'CO2Intensity',
 'ActualWindProduction',
 'SystemLoadEP2',
 'SMPEP2']
```

In[6]:

```
num_list.remove("DateTime")
num_list
```

Out[6]:

```
['WeekOfYear',
 'PeriodOfDay',
 'ForecastWindProduction',
 'SystemLoadEA',
 'SMPEA',
 'ORKWindspeed',
 'CO2Intensity',
```

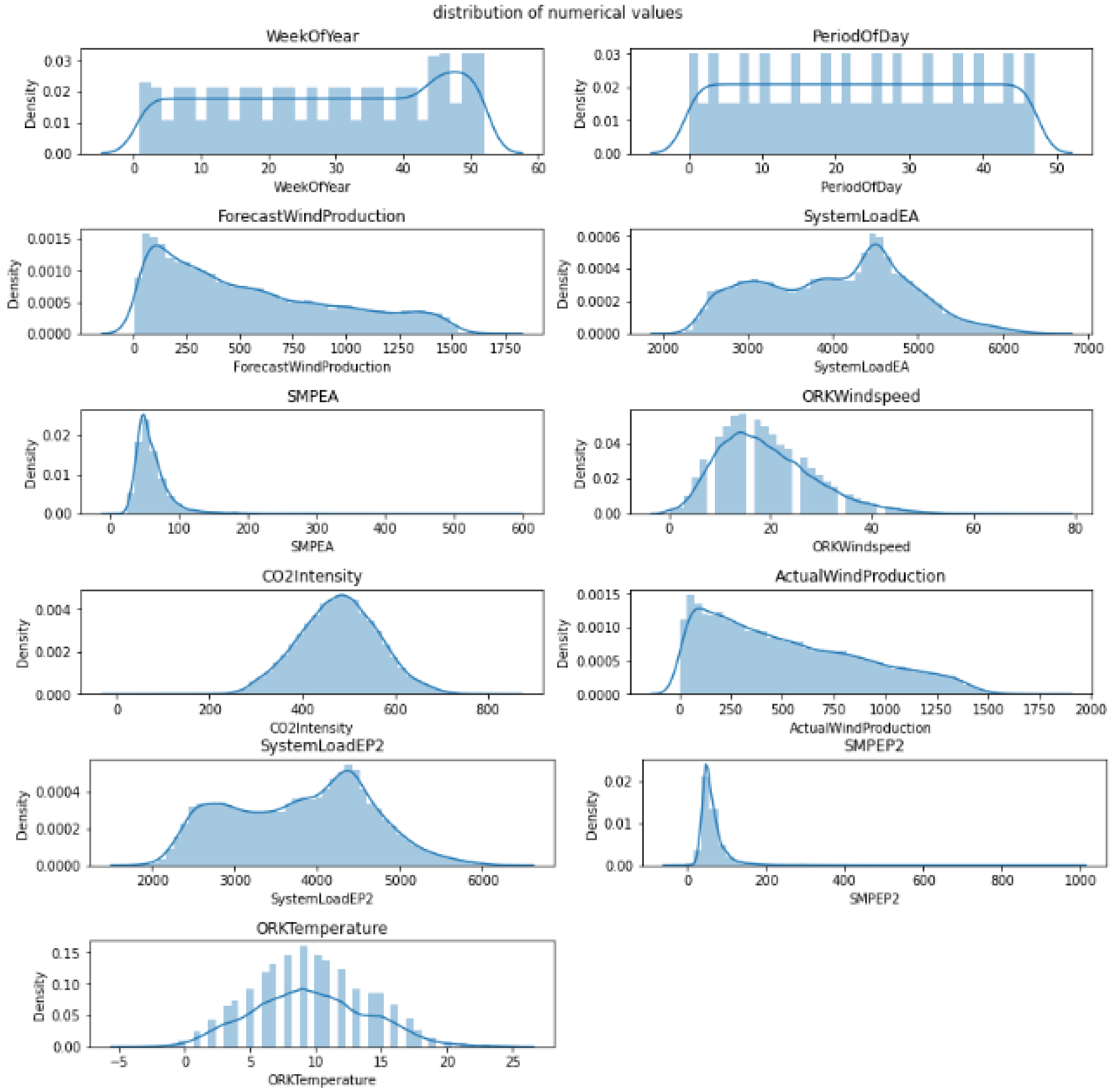
```
 'ActualWindProduction',
 'SystemLoadEP2',
 'SMPEP2']
```

In [7]:

```
num_list.append("ORKTemperature")
```

In [8]:

```
k=1
plt.figure(figsize=(12,12))
plt.suptitle("distribution of numerical values")


for i in df.loc[:,num_list]:
    plt.subplot(6,2,k)
    sns.distplot(df[i])
    plt.title(i)
    k+=1
    plt.tight_layout()
```



Visualization of missing values:

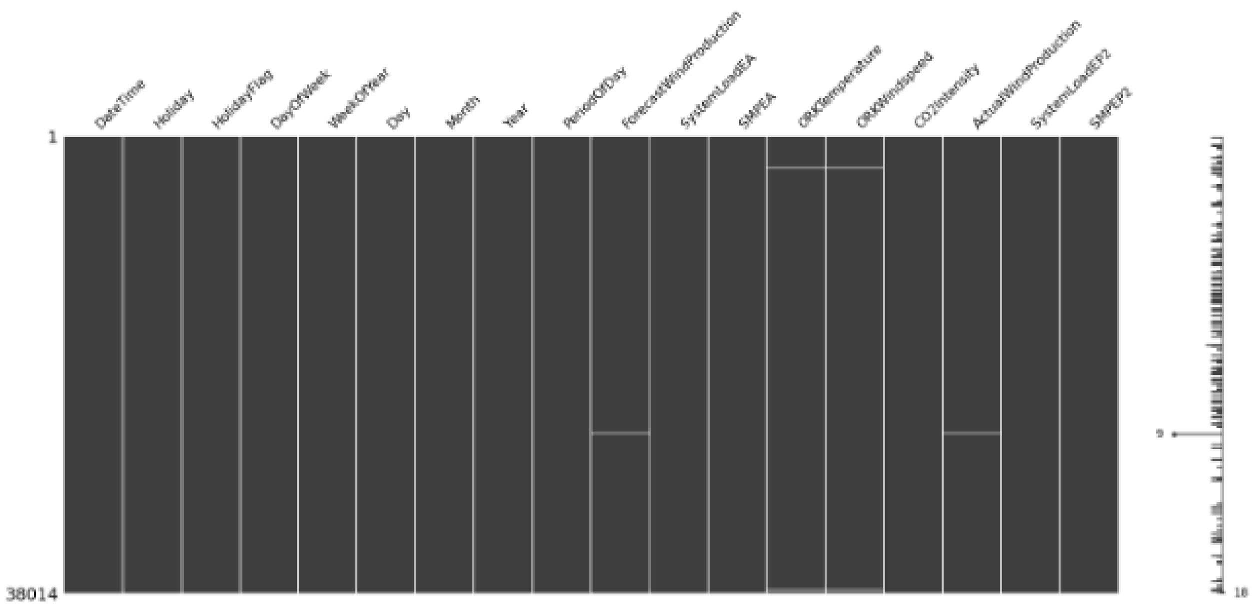
In [1]:

```
import missingno as msno
```

In [2]:

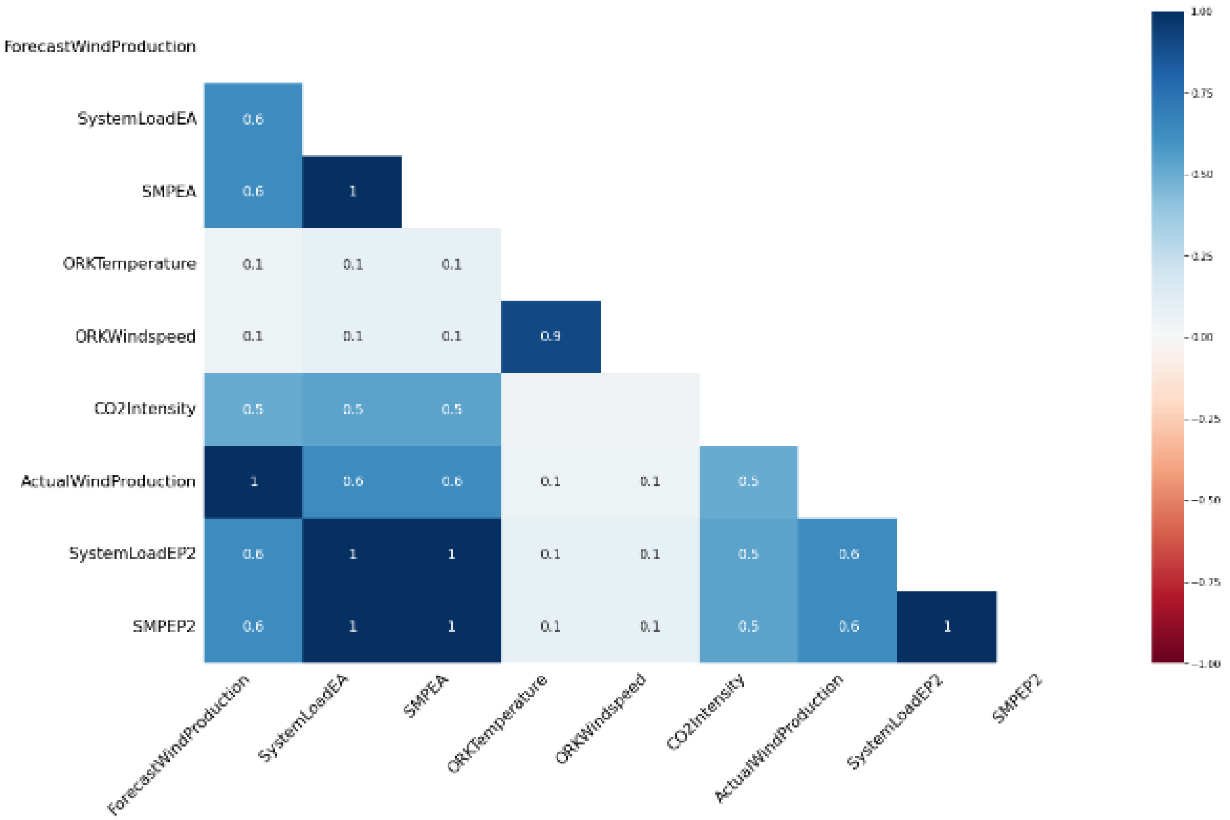
linkcode

```
msno.matrix(df);
```



In[3]:

msno.heatmap(df);



In[4]:

```
# missing values based on distribution states# eksik değer gide
rme
```

In[5]:

```
df["ForecastWindProduction"].fillna(df.ForecastWindProduction.m
ean(),inplace=True)
df["SystemLoadEA"].fillna(df.SystemLoadEA.mean(),inplace=True)
df["SMPEA"].fillna(df.SMPEA.mean(),inplace=True)
df["CO2Intensity"].fillna(df.CO2Intensity.median(),inplace=True
)
df["ActualWindProduction"].fillna(value=250,inplace=True)
df["SystemLoadEP2"].fillna(df.SystemLoadEP2.median(),inplace=Tr
ue)
df["SMPEP2"].fillna(df.SMPEP2.median(),inplace=True)
In [23]:
df["ORKTemperature"].fillna(value=10,inplace=True)
df["ORKWindspeed"].fillna(value=20,inplace=True)
```

In[6]:

```
df.isna().sum()
```

Out[6]:

```
DateTime                  0
Holiday                   0
HolidayFlag               0
DayOfWeek                 0
WeekOfYear                0
Day                       0
Month                     0
Year                      0
PeriodOfDay               0
ForecastWindProduction    0
SystemLoadEA              0
SMPEA                     0
ORKTemperature            0
ORKWindspeed              0
CO2Intensity              0
ActualWindProduction      0
SystemLoadEP2             0
```

```
SMPEP2                    0
dtype: int64
In [25]:
linkcode
```

```
# we have removed the missing values
```

Outlier Problem:

In[1]:

```
linkcode
df.describe([0.05,0.1,0.25,0.35,0.5,0.65,0.75,0.9,0.95,0.98]).T
```

Out[1]:

	count	mean	std	min	5%	10%
HolidayFlag	38014.0	0.040406	0.196912	0.00	0.0000	0.000
DayOfWeek	38014.0	2.997317	1.999959	0.00	0.0000	0.000
WeekOfYear	38014.0	28.124586	15.587575	1.00	3.0000	6.000
Day	38014.0	15.739412	8.804247	1.00	2.0000	4.000
Month	38014.0	6.904246	3.573696	1.00	1.0000	2.000
Year	38014.0	2012.383859	0.624956	2011.00	2011.0000	2012.000
PeriodOfDay	38014.0	23.501105	13.853108	0.00	2.0000	4.000
ForecastWindProduction	38014.0	544.261451	414.337377	0.68	52.3420	80.395
SystemLoadEA	38014.0	4020.085019	860.454229	2183.94	2626.2585	2812.039
SMPEA	38014.0	62.720388	32.251486	0.00	33.8200	38.360
ORKTemperature	38014.0	9.629268	4.422794	-4.00	3.0000	4.000
ORKWindspeed	38014.0	19.217970	9.533848	0.00	5.6000	7.400
CO2Intensity	38014.0	479.373213	85.346848	0.00	336.0965	367.229
ActualWindProduction	38014.0	520.727206	378.270841	1.00	43.0000	77.000
SystemLoadEP2	38014.0	3785.978038	843.247470	1809.96	2449.5430	2599.070
SMPEP2	38014.0	64.136371	35.414160	-47.74	33.1900	37.926

In[2]:

```
num_list
```

Out[2]:

```
['WeekOfYear',
 'PeriodOfDay',
 'ForecastWindProduction',
 'SystemLoadEA',
 'SMPEA',
 'ORKWindspeed',
 'CO2Intensity',
 'ActualWindProduction',
 'SystemLoadEP2',
 'SMPEP2',
 'ORKTemperature']
```

In[3]:

```
out_list=["ForecastWindProduction","SystemLoadEA","SMPEA",
          "ORKWindspeed","SMPEP2"]
```

In[4]:

```
for i in df.loc[:,out_list]:
    Q1 = df[i].quantile(0.02)
    Q3 = df[i].quantile(0.98)
    IQR = Q3-Q1
    up = Q3 + 1.5*IQR
    low = Q1 - 1.5*IQR

    if df[(df[i] > up) | (df[i] < low)].any(axis=None):
        print(i,"yes")
    else:
        print(i, "no")
```

Out[4]:

```
ForecastWindProduction no
SystemLoadEA no
SMPEA yes
ORKWindspeed no
SMPEP2 yes
```

In[5]:

```
#accessing outliers

def outliers_df(df):
    q1,q3=np.percentile(df,[0.02,0.98])
    ıqr=q3-q1
    low,high=q1-1.5*(ıqr),q3+1.5*(ıqr)
    outliers_train=[i for i in df if i<low or i>high]
    return outliers_train
```

In[6]:

```
len(outliers_df(df.SMPEA))
```

Out[6]:

```
9923
```

In[7]:

```
len(outliers_df(df.SMPEP2))
```

Out[7]:

```
12036
```

In[8]:

```
df_remove_out=df.copy()
```

In[9]:

```
# remove outliers;

for i in df_remove_out.loc[:,out_list]:

   Q1 = df_remove_out[i].quantile(0.02)
```

```
    Q3 = df_remove_out[i].quantile(0.98)
    IQR = Q3 - Q1
    up_lim=Q3+1.5 *IQR
    low_lim=Q1-1.5 *IQR

    df_remove_out.loc[df_remove_out[i]>up_lim,i]=up_lim
    df_remove_out.loc[df_remove_out[i]<low_lim,i]=low_lim
```

In[10]:

```
for i in df_remove_out.loc[:,out_list]:
    Q1 = df_remove_out[i].quantile(0.02)
    Q3 = df_remove_out[i].quantile(0.98)
    IQR = Q3-Q1
    up = Q3 + 1.5*IQR
    low = Q1 - 1.5*IQR

    if df[(df_remove_out[i] > up) | (df_remove_out[i] < low)].any(axis=None):
        print(i,"yes")
    else:
        print(i, "no")
```

Out[10]:

```
ForecastWindProduction no
SystemLoadEA no
SMPEA no
ORKWindspeed no
SMPEP2 no
```

In[11]:

```
df.describe([0.05,0.1,0.25,0.35,0.5,0.65,0.75,0.9,0.95,0.98]).T
```

Out[11]:

	count	mean	std	min	5%	10%
HolidayFlag	38014.0	0.040406	0.196912	0.00	0.0000	0.000
DayOfWeek	38014.0	2.997317	1.999959	0.00	0.0000	0.000
WeekOfYear	38014.0	28.124586	15.587575	1.00	3.0000	6.000
Day	38014.0	15.739412	8.804247	1.00	2.0000	4.000
Month	38014.0	6.904246	3.573696	1.00	1.0000	2.000
Year	38014.0	2012.383859	0.624956	2011.00	2011.0000	2012.000
PeriodOfDay	38014.0	23.501105	13.853108	0.00	2.0000	4.000
ForecastWindProduction	38014.0	544.261451	414.337377	0.68	52.3420	80.395
SystemLoadEA	38014.0	4020.085019	860.454229	2183.94	2626.2585	2812.039
SMPEA	38014.0	62.720388	32.251486	0.00	33.8200	38.360
ORKTemperature	38014.0	9.629268	4.422794	-4.00	3.0000	4.000
ORKWindspeed	38014.0	19.217970	9.533848	0.00	5.6000	7.400
CO2Intensity	38014.0	479.373213	85.346848	0.00	336.0965	367.229
ActualWindProduction	38014.0	520.727206	378.270841	1.00	43.0000	77.000
SystemLoadEP2	38014.0	3785.978038	843.247470	1809.96	2449.5430	2599.070
SMPEP2	38014.0	64.136371	35.414160	-47.74	33.1900	37.926

In[12]:

```
df_remove_out.describe([0.05,0.1,0.25,0.35,0.5,0.65,0.75,0.9,0.95,0.98]).T
```

Out[12]:

	count	mean	std	min	5%	10%
HolidayFlag	38014.0	0.040406	0.196912	0.00	0.0000	0.000
DayOfWeek	38014.0	2.997317	1.999959	0.00	0.0000	0.000
WeekOfYear	38014.0	28.124586	15.587575	1.00	3.0000	6.000
Day	38014.0	15.739412	8.804247	1.00	2.0000	4.000
Month	38014.0	6.904246	3.573696	1.00	1.0000	2.000
Year	38014.0	2012.383859	0.624956	2011.00	2011.0000	2012.000
PeriodOfDay	38014.0	23.501105	13.853108	0.00	2.0000	4.000
ForecastWindProduction	38014.0	544.261451	414.337377	0.68	52.3420	80.395
SystemLoadEA	38014.0	4020.085019	860.454229	2183.94	2626.2585	2812.039
SMPEA	38014.0	62.621453	31.179620	0.00	33.8200	38.360
ORKTemperature	38014.0	9.629268	4.422794	-4.00	3.0000	4.000
ORKWindspeed	38014.0	19.217970	9.533848	0.00	5.6000	7.400
CO2Intensity	38014.0	479.373213	85.346848	0.00	336.0965	367.229
ActualWindProduction	38014.0	520.727206	378.270841	1.00	43.0000	77.000
SystemLoadEP2	38014.0	3785.978038	843.247470	1809.96	2449.5430	2599.070
SMPEP2	38014.0	63.958532	33.351453	-47.74	33.1900	37.926

In[13]:

```
df[df.SMPEP2<0]=0
```

In[14]:

```
linkcode
df_remove_out[df_remove_out.SMPEP2<0]=0
```

<u>Time Series Analysis:</u>

In[1]:

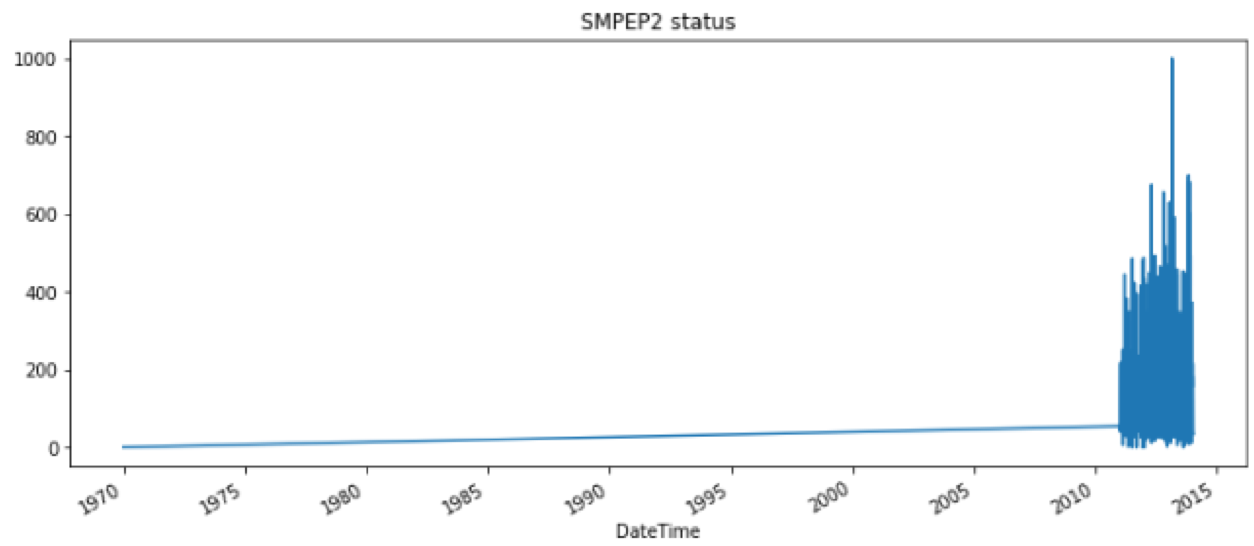
```
from datetime import datetime
df["DateTime"] = pd.to_datetime(df.DateTime)
```

In [2]:

```
linkcode
df['year'] = df['DateTime'].dt.year
df['month'] = df['DateTime'].dt.month
df["day"]=df["DateTime"].dt.day
```

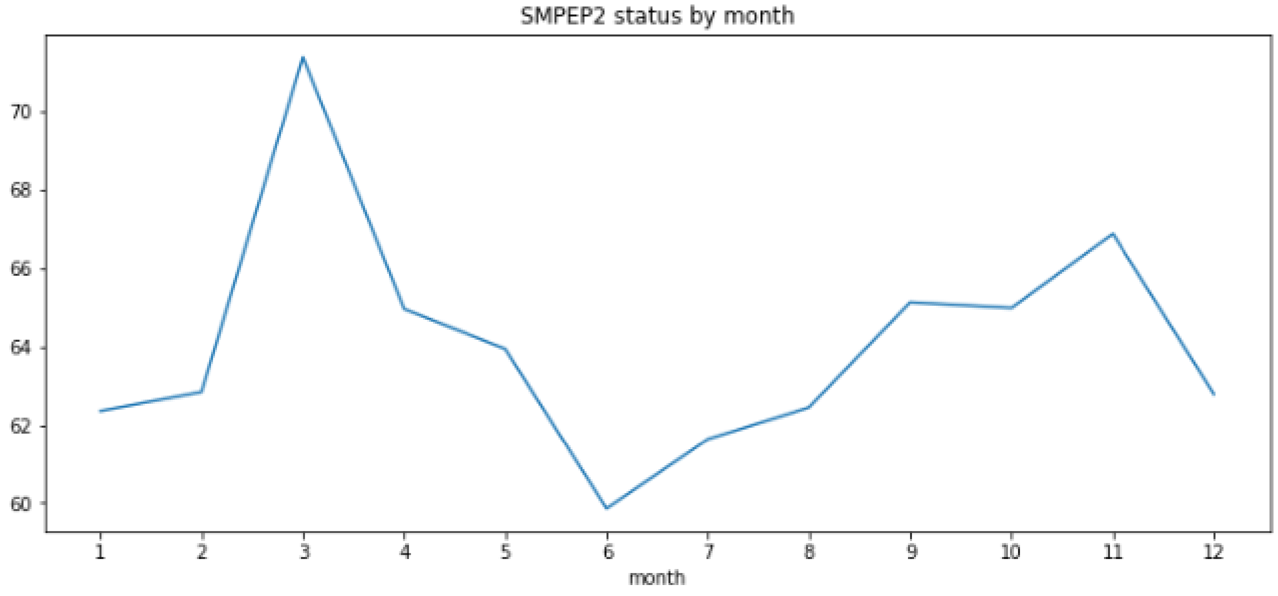
In[3]:

```
custgroup=df.groupby('DateTime').mean()
plt.figure(figsize=(12,5))
custgroup['SMPEP2'].plot(x=df.DateTime)
plt.title("SMPEP2 status")
plt.show()
```



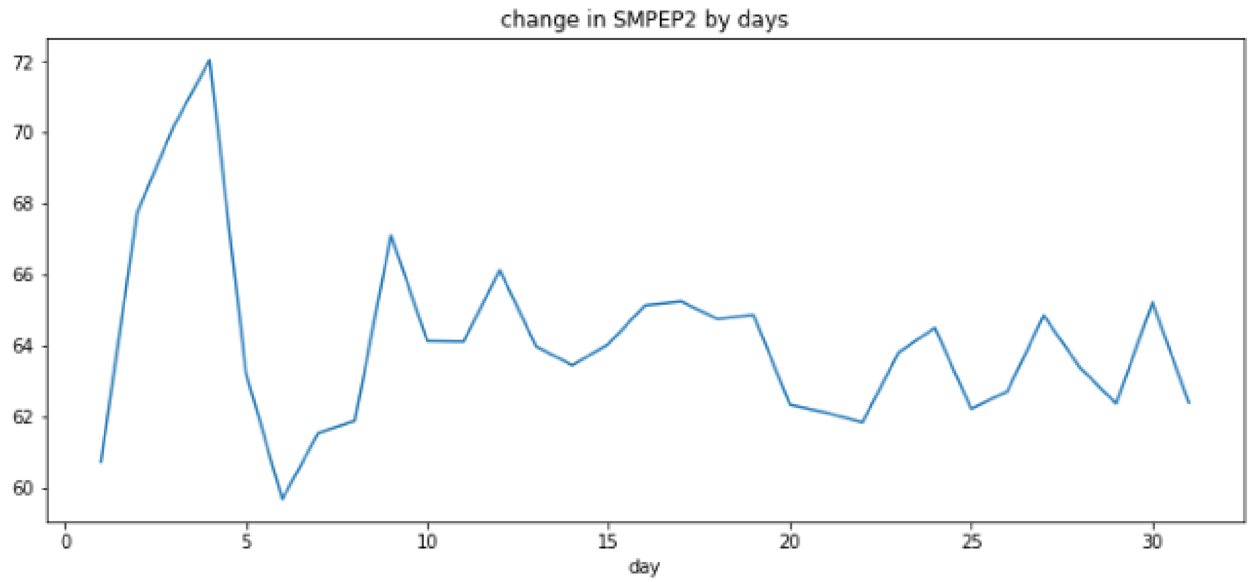
In[4]:

```
custgroup=df.groupby('month').mean()
fig,ax=plt.subplots(figsize=(12,5))
ax.xaxis.set(ticks=range(0,13))
custgroup['SMPEP2'].plot(x=df.DateTime)
plt.title("SMPEP2 status by month")
plt.show()
```



In[5]:

```
custgroup=df.groupby('day').mean()
plt.figure(figsize=(12,5))
custgroup['SMPEP2'].plot(x=df.DateTime)
plt.title("change in SMPEP2 by days")
plt.show()
```



In[6]:

```
df_remove_out["DateTime"] =
pd.to_datetime(df_remove_out.DateTime)
```

In[7]:

```
df_remove_out['year'] = df_remove_out['DateTime'].dt.year
df_remove_out['month'] = df_remove_out['DateTime'].dt.month
df_remove_out["day"]=df_remove_out["DateTime"].dt.day
```

In[8]:

```
linkcode
df_remove_out
```

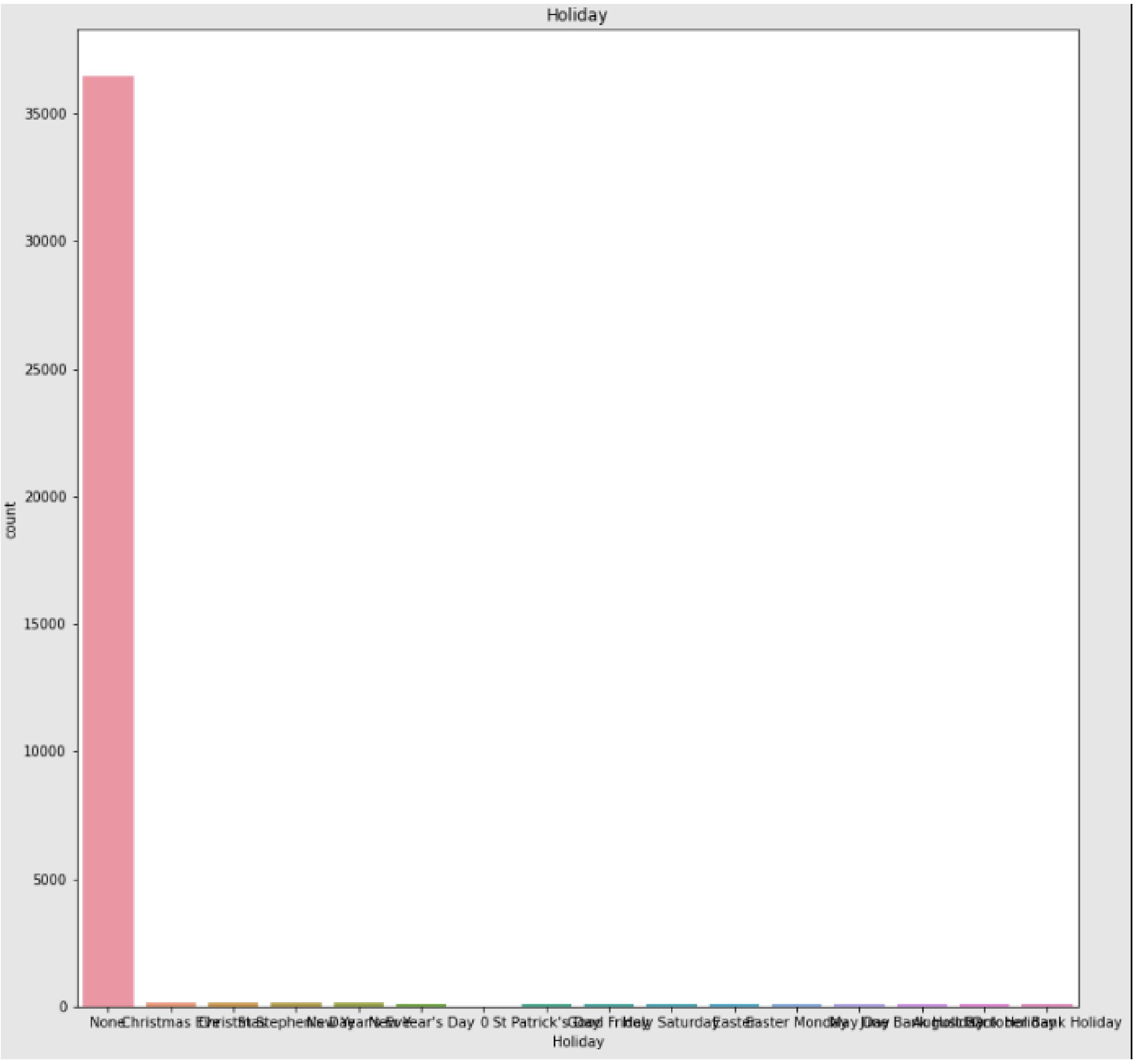
	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOf
0	2011-01-11 00:00:00	None	0	1	44	1	11	2011	0
1	2011-01-11 00:30:00	None	0	1	44	1	11	2011	1
2	2011-01-11 01:00:00	None	0	1	44	1	11	2011	2
3	2011-01-11 01:30:00	None	0	1	44	1	11	2011	3
4	2011-01-11 02:00:00	None	0	1	44	1	11	2011	4
...	...	...	...	...	...	...	...	...	...
38009	2013-12-31 21:30:00	New Year's Eve	1	1	1	31	12	2013	43

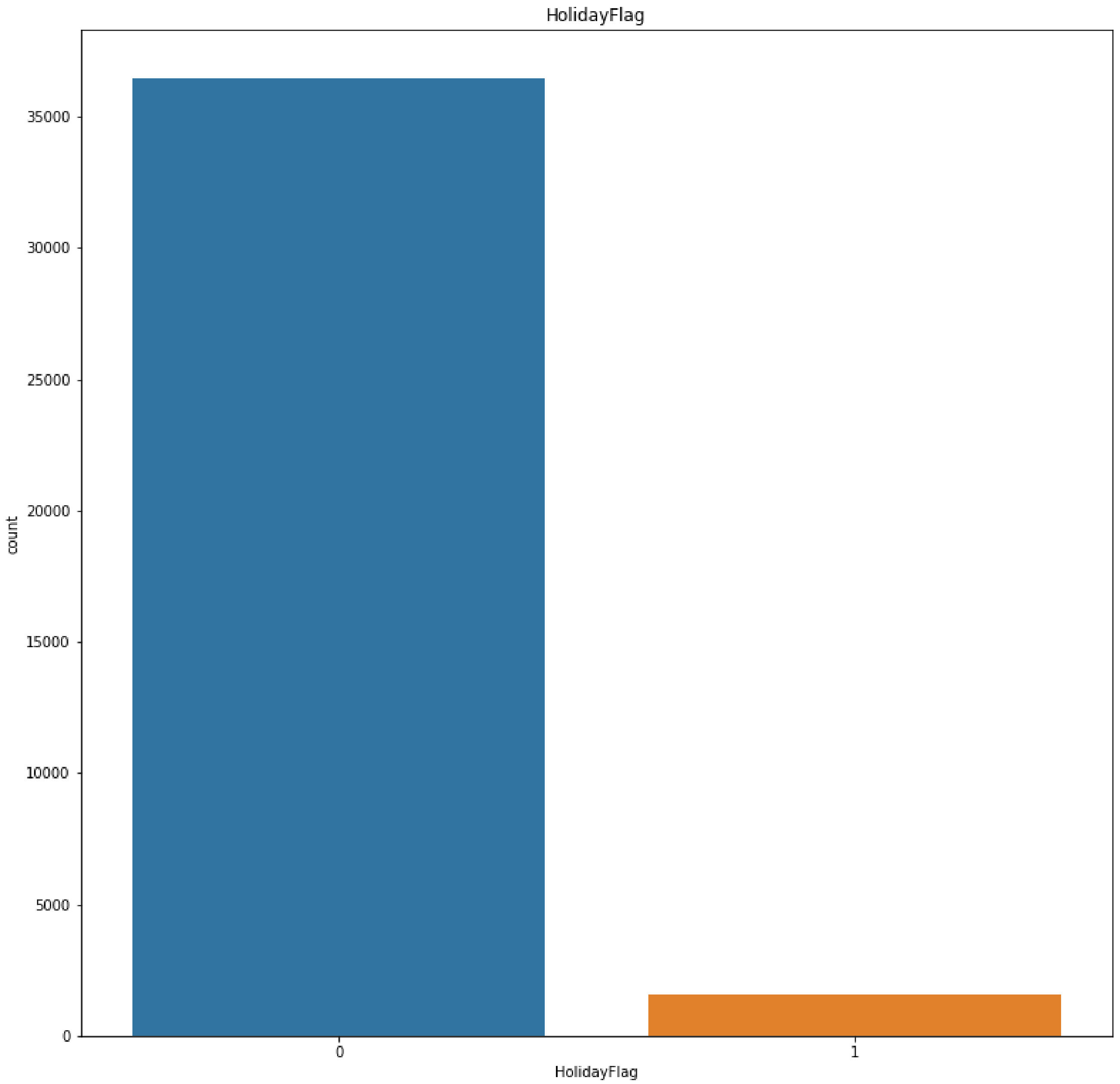
38010	2013-12-31 22:00:00	New Year's Eve	1	1	1	31	12	2013	44
38011	2013-12-31 22:30:00	New Year's Eve	1	1	1	31	12	2013	45
38012	2013-12-31 23:00:00	New Year's Eve	1	1	1	31	12	2013	46
38013	2013-12-31 23:30:00	New Year's Eve	1	1	1	31	12	2013	47

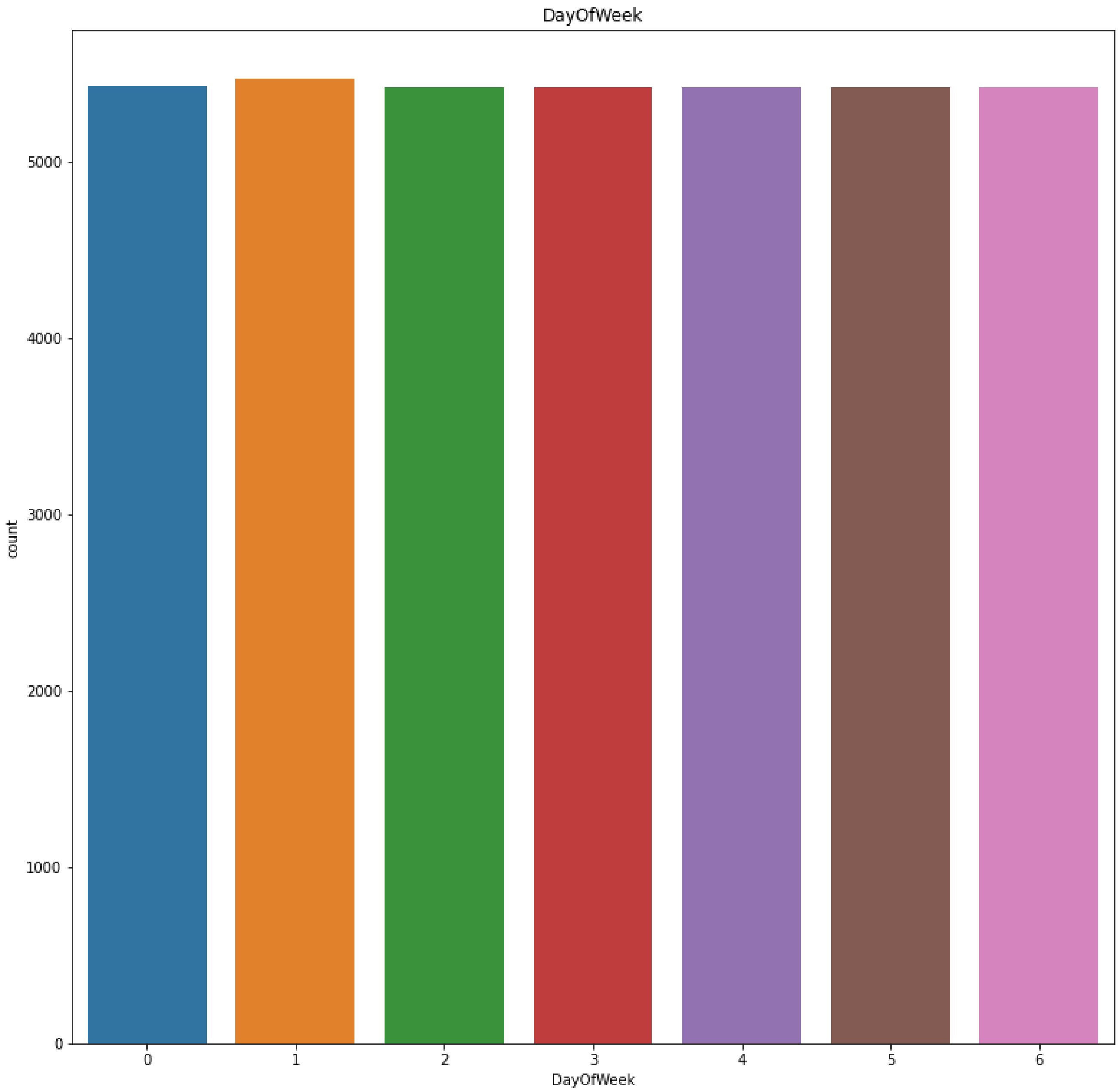
Data Visualize:

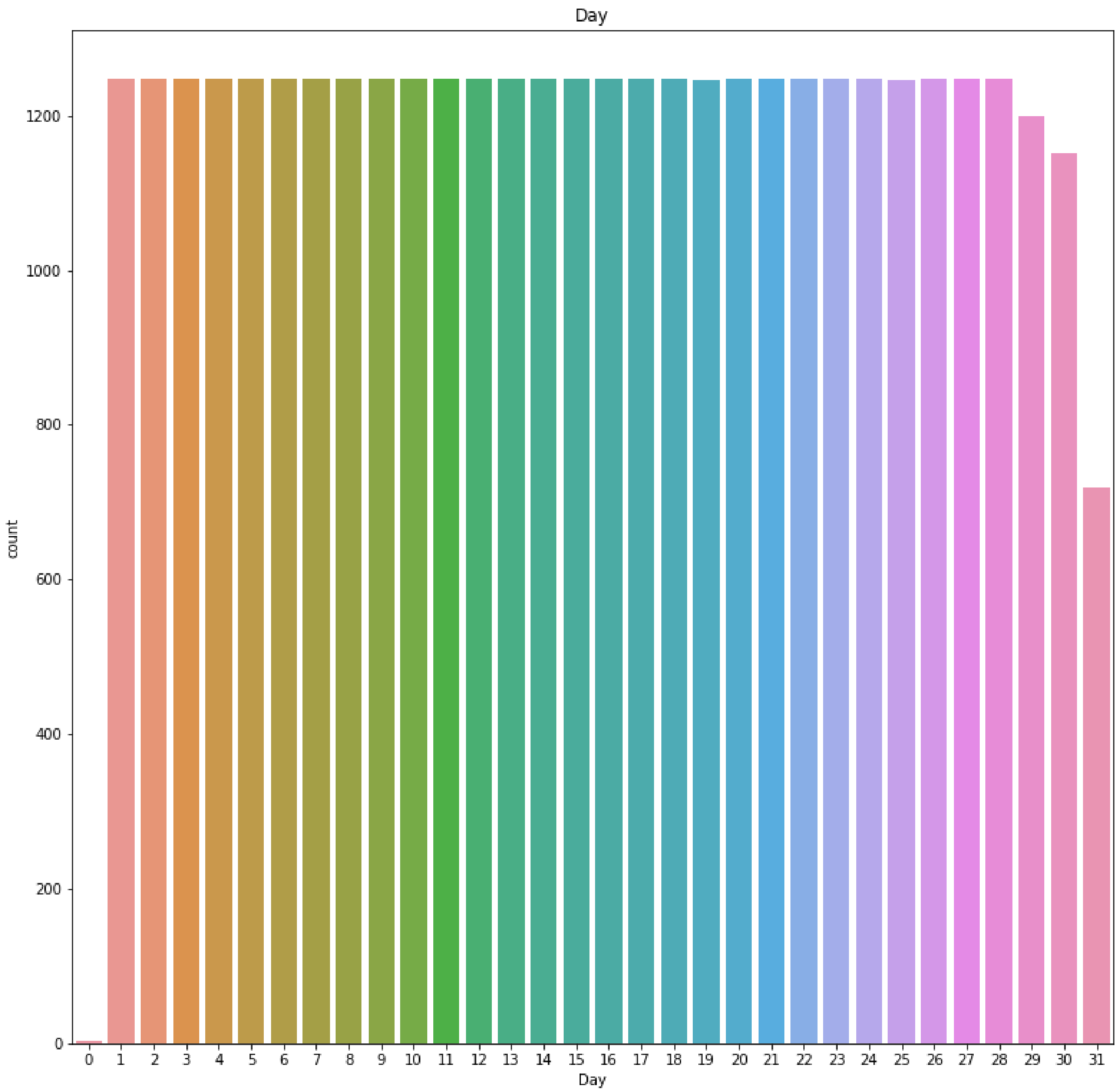
In[1]:

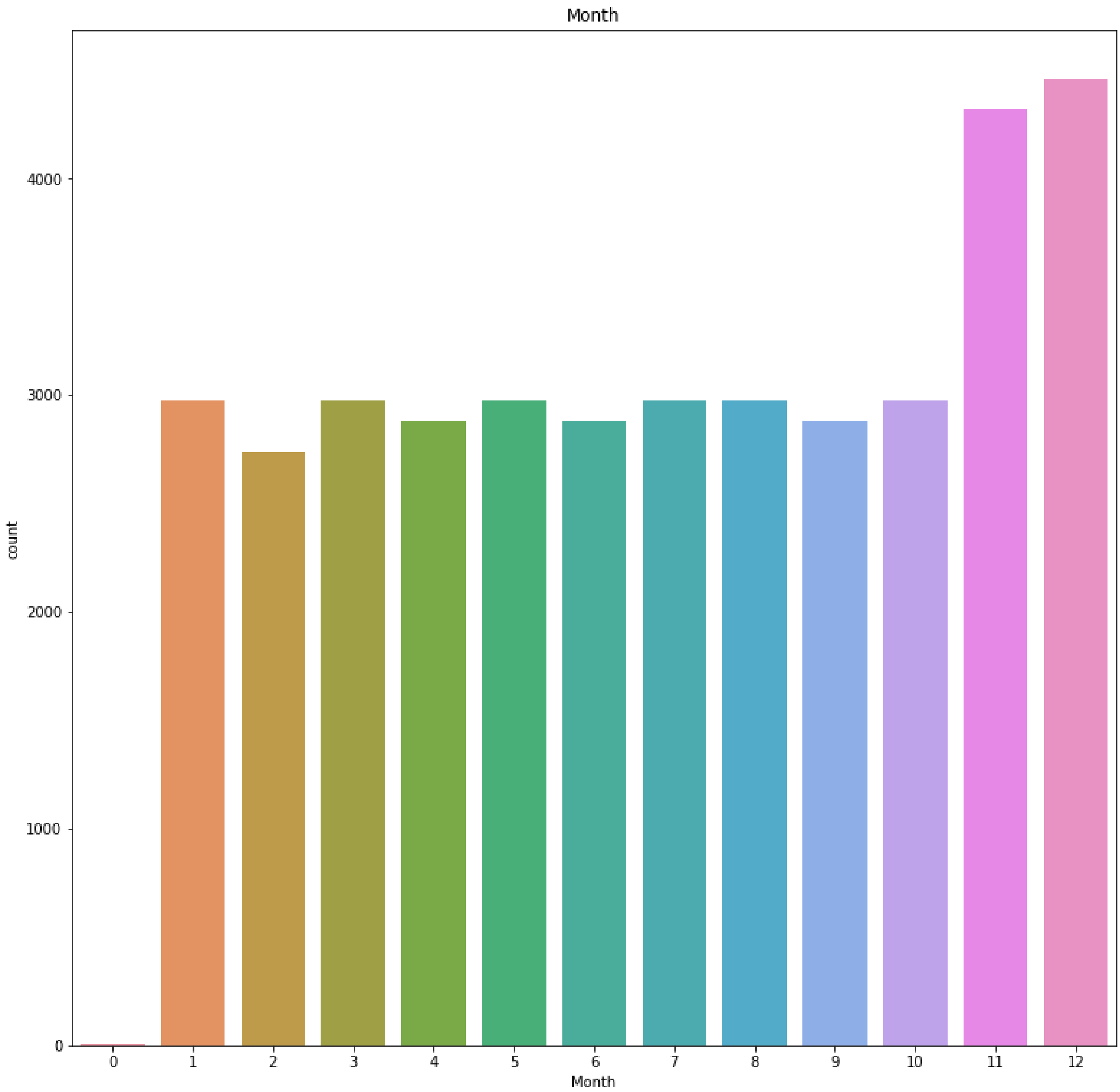
```
for i in cat_list:

    plt.figure(figsize=(13,13))
    sns.countplot(x=i,data=df.loc[:,cat_list])
    plt.title(i)
```

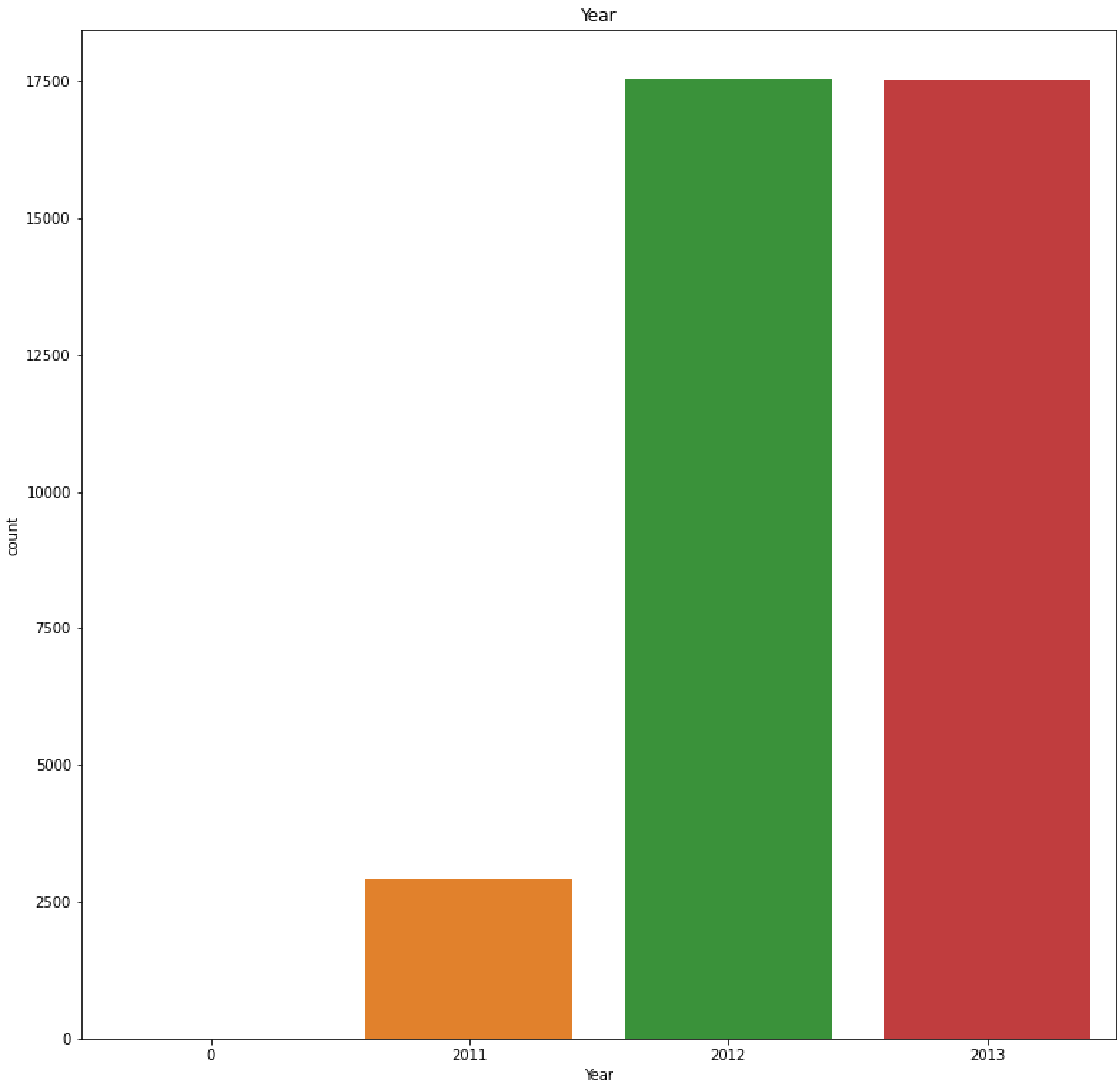

Holiday
count
Holiday
35000
30000
25000
20000
15000
10000
5000
0
None
Christmas Eve
Christmas
St Stephen's Day
New Year's Eve
New Year's Day
0
St Patrick's Day
Good Friday
Holy Saturday
Easter
Easter Monday
May Day
June Bank Holiday
August Bank Holiday
October Bank Holiday

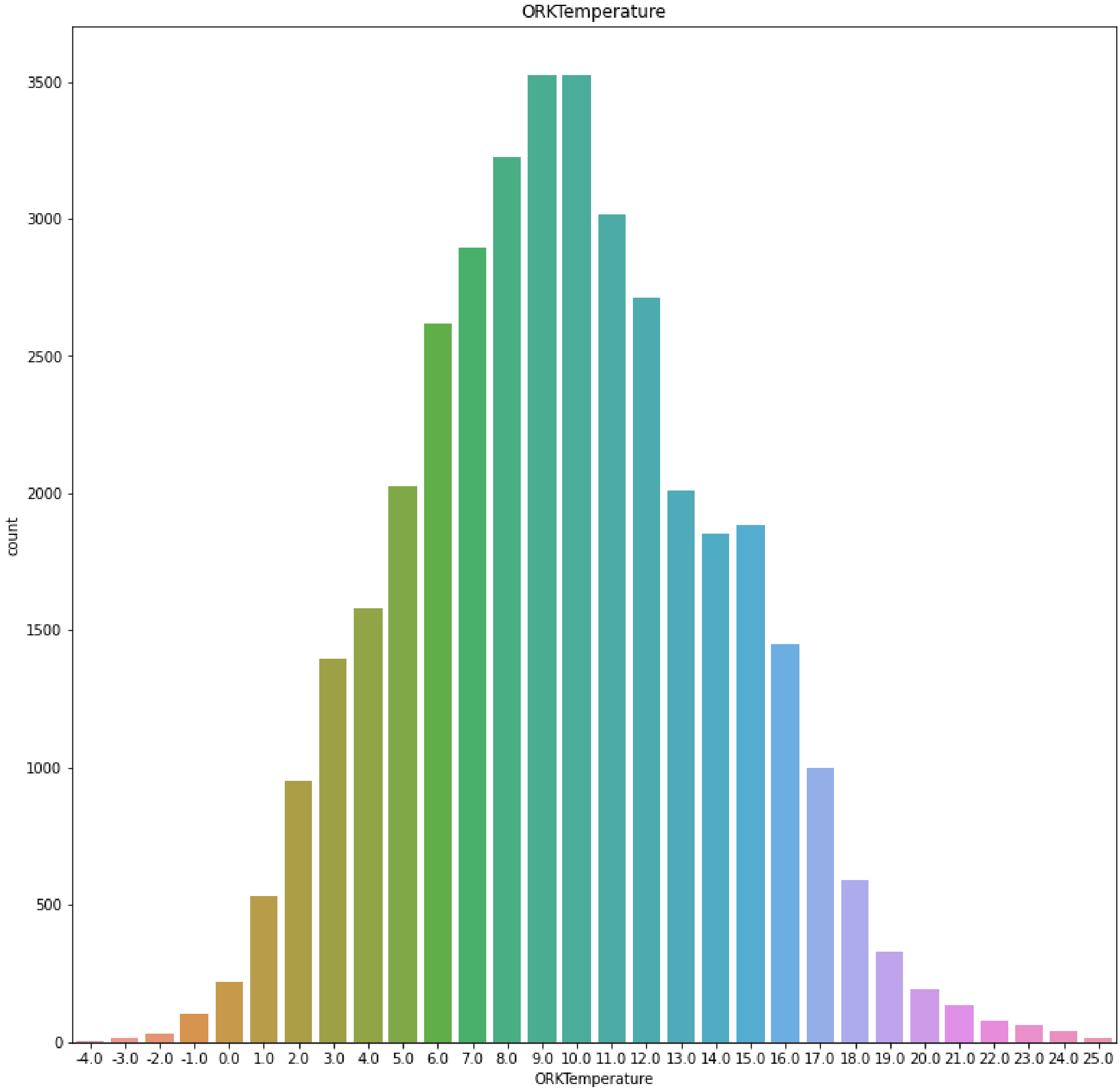

HolidayFlag
count
35000
30000
25000
20000
15000
10000
5000
0
0
1
HolidayFlag

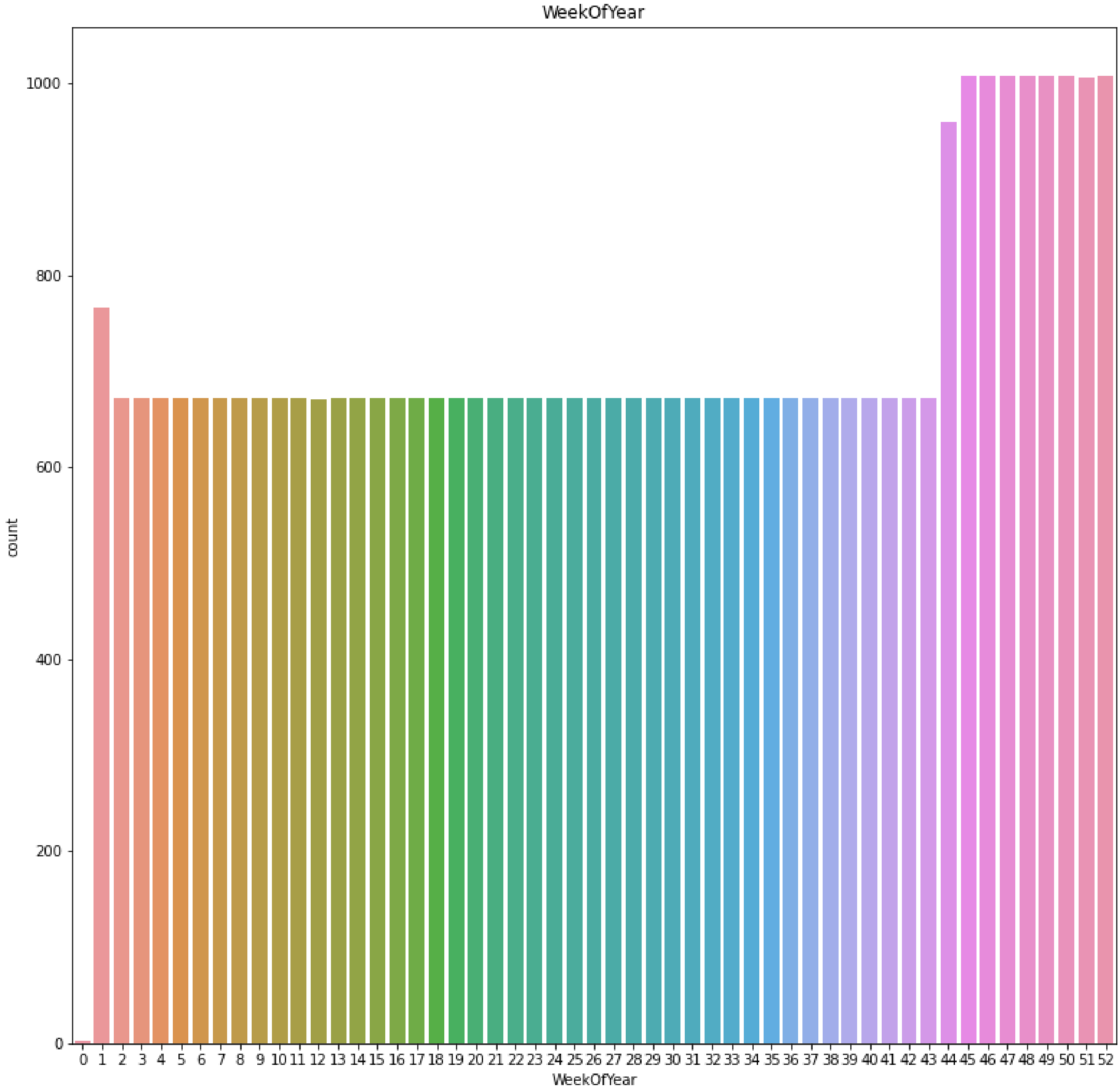

DayOfWeek
count
5000
4000
3000
2000
1000
0
0
1
2
3
4
5
6
DayOfWeek


Day
count
1200
1000
800
600
400
200
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Day


Month
count
4000
3000
2000
1000
0
0
1
2
3
4
5
6
7
8
9
10
11
12
Month

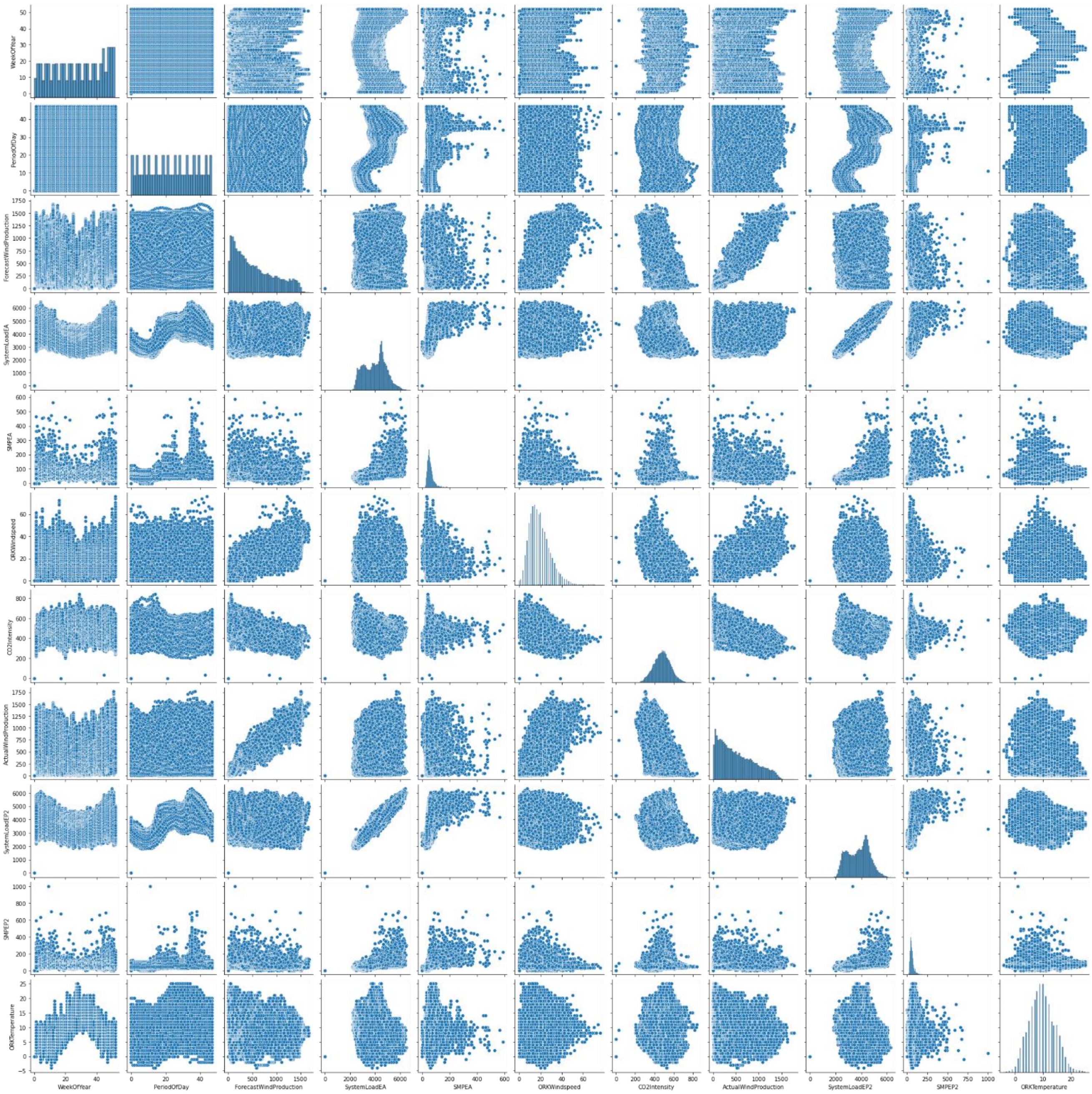

Year
17500
15000
12500
10000
7500
5000
2500
0
count
0
2011
2012
2013
Year


ORKTemperature
count
3500
3000
2500
2000
1500
1000
500
0
-4.0 -3.0 -2.0 -1.0 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0 17.0 18.0 19.0 20.0 21.0 22.0 23.0 24.0 25.0
ORKTemperature



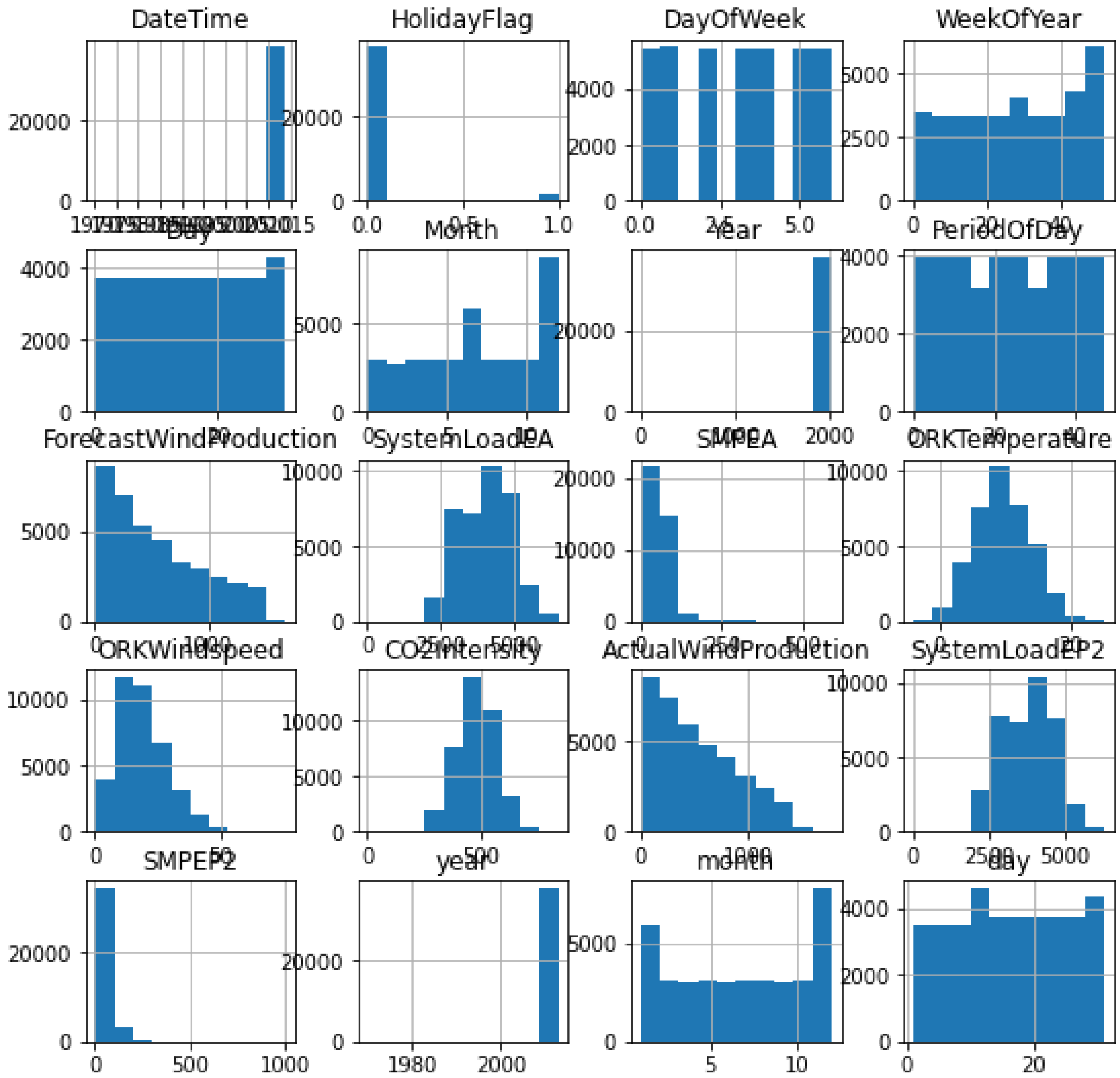
In[2]:

```
sns.pairplot(df.loc[:,num_list]);
```



In[3]:

```
df.hist(figsize=(9,9));
```



In[4]:

cat_list

Out[4]:

['Holiday',
 'HolidayFlag',
 'DayOfWeek',
 'Day',

```
 'Month',
 'Year',
 'ORKTemperature',
 'WeekOfYear']
```

In[5]:

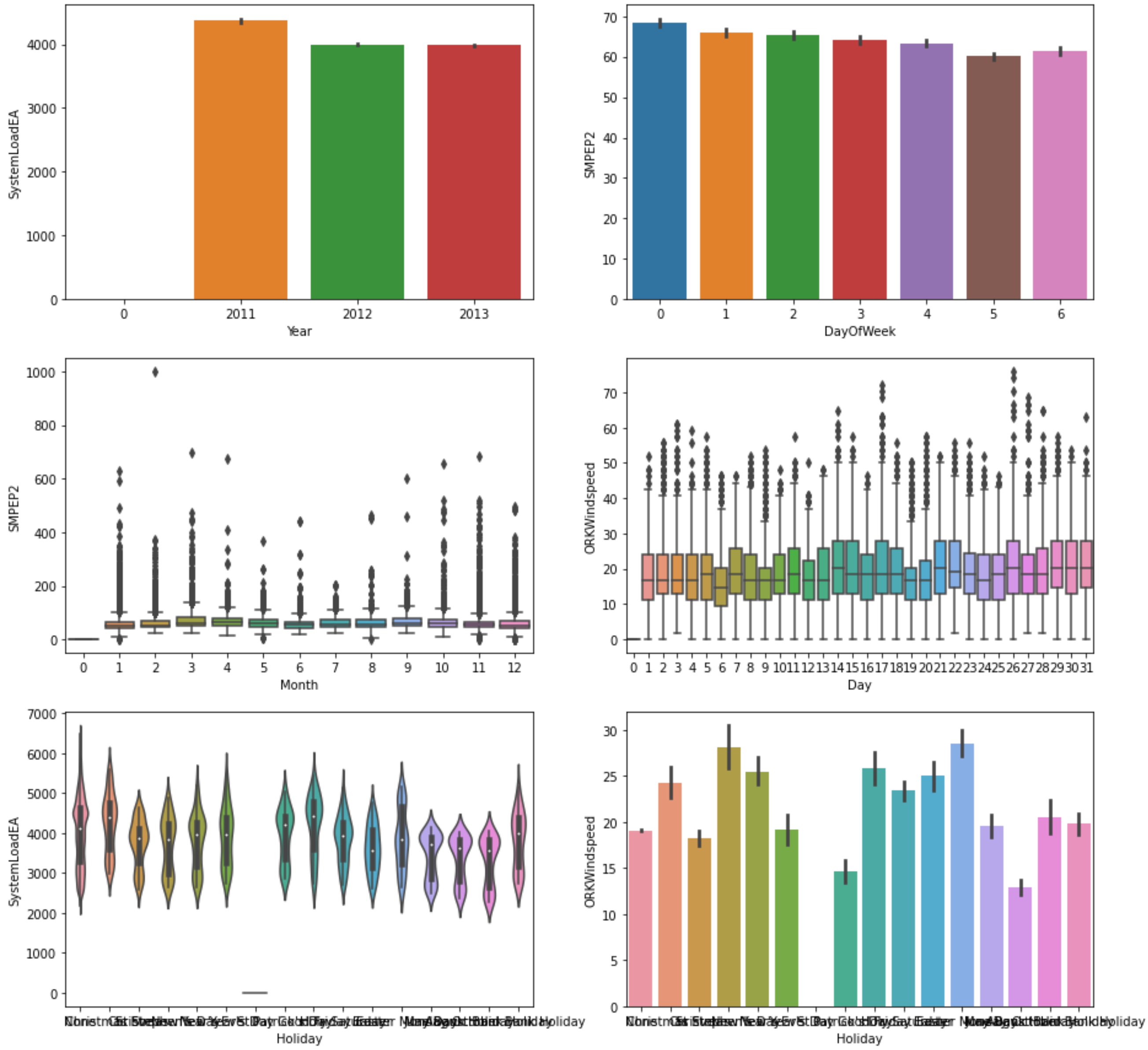
```
num_list
```

Out[5]:

```
['WeekOfYear',
 'PeriodOfDay',
 'ForecastWindProduction',
 'SystemLoadEA',
 'SMPEA',
 'ORKWindspeed',
 'CO2Intensity',
 'ActualWindProduction',
 'SystemLoadEP2',
 'SMPEP2',
 'ORKTemperature']
```

In[6]:

```
linkcode
plt.figure(figsize=(15,15))
plt.subplot(3,2,1)
sns.barplot(x ='Year',y ='SystemLoadEA',data = df)
plt.subplot(3,2,2)
sns.barplot(x="DayOfWeek",y="SMPEP2",data=df)
plt.subplot(3,2,3)
sns.boxplot(x="Month",y="SMPEP2",data=df)
plt.subplot(3,2,4)
sns.boxplot(x="Day",y="ORKWindspeed",data=df)
plt.subplot(3,2,5)
sns.violinplot(x="Holiday",y="SystemLoadEA",data=df)
plt.subplot(3,2,6)
sns.barplot(x="Holiday",y="ORKWindspeed",data=df)
plt.show()
```



In[7]:

```
df.drop("DateTime",axis=1,inplace=True)
```

In[8]:

linkcode

```
df.head(2)
```

Out[8]:

	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOfDay	ForecastWi
0	None	0	1	44	1	11	2011	0	315.31
1	None	0	1	44	1	11	2011	1	321.80

Encoding:

In[1]:

```
dms=pd.get_dummies(df["Holiday"])
dms
```

Out[1]:

	0	August Bank Holiday	Christmas	Christmas Eve	Easter	Easter Monday	Good Friday	Holy Saturday	June Bank Holiday	May Day
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
38009	0	0	0	0	0	0	0	0	0	0
38010	0	0	0	0	0	0	0	0	0	0
38011	0	0	0	0	0	0	0	0	0	0
38012	0	0	0	0	0	0	0	0	0	0
38013	0	0	0	0	0	0	0	0	0	0

In[2]:

```
df.drop("Holiday",axis=1,inplace=True)
df=pd.concat([df,dms],axis=1)
```

```
df.head()
```

Out[2]:

	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOfDay	ForecastWindProduct
0	0	1	44	1	11	2011	0	315.31
1	0	1	44	1	11	2011	1	321.80
2	0	1	44	1	11	2011	2	328.57
3	0	1	44	1	11	2011	3	335.60
4	0	1	44	1	11	2011	4	342.90

In[3]:

```
dms2=pd.get_dummies(df_remove_out["Holiday"])
df_remove_out.drop("Holiday",axis=1,inplace=True)
df_remove_out=pd.concat([df_remove_out,dms2],axis=1)

df_remove_out.head()
```

Out[3]:

	DateTime	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOfDay	Forecast'
0	2011-01-11 00:00:00	0	1	44	1	11	2011	0	315.31
1	2011-01-11 00:30:00	0	1	44	1	11	2011	1	321.80
2	2011-01-11 01:00:00	0	1	44	1	11	2011	2	328.57
3	2011-01-11 01:30:00	0	1	44	1	11	2011	3	335.60
4	2011-01-11 02:00:00	0	1	44	1	11	2011	4	342.90

In[4]:

```
df_remove_out.drop("DateTime",axis=1,inplace=True)
```

Feature Engineering

Feature engineering is a critical step in building accurate electricity price prediction models. Effective feature engineering can help capture relevant patterns and relationships in the data. Here are some feature engineering techniques and considerations for electricity price prediction:

1. **Time-Based Features:**
   - *Time of Day: Create features to represent the time of day, such as hour of the day or minute of the hour. Electricity prices often exhibit daily and hourly patterns.*
   - *Day of the Week: Include features for the day of the week to capture weekly seasonality.*
   - *Month and Season: Incorporate features for the month and season to capture monthly and seasonal patterns.*

- *Holidays: Add binary features to indicate holidays or special events that may affect electricity prices.*

2. **Lagged Features:**

   - *Lagged Prices: Include lagged electricity prices as features. Lagged values can capture autocorrelation and previous price trends.*
   - *-Lagged Demand: Consider lagged electricity demand as a feature, as demand patterns can influence prices.*

3. **Rolling Statistics:**

   - *Rolling Mean and Rolling Standard Deviation: Calculate rolling statistics over a certain window (e.g., 7 days) to capture short-term trends and volatility.*

4. **Weather Data:**

   - *Incorporate weather data, such as temperature, humidity, or wind speed, as these factors can impact electricity consumption and prices.*

5. **Demand Data:**

   - *Include features related to electricity demand, such as historical demand levels and peak demand periods.*

6. **Market Data:**

   - *Consider variables related to the energy market, such as fuel prices, electricity generation capacity, or the state of the grid.*

7. **Feature Scaling:**

   - *Normalize or scale features as needed to ensure that they have the same magnitude. This is important for models like linear regression or neural networks.*

8. **Categorical Variables:**

- *If you have categorical variables (e.g., region or market type), use one-hot encoding or other categorical encoding techniques to convert them into numerical features.*

9. **Special Events:**

- *Include features that indicate special events or anomalies, such as power outages or significant market changes.*

10. **Price Differencing:**

- *Calculate differences between consecutive price values to create features that capture price changes.*

11. **Calendar Events:**

- *Incorporate calendar-related features, such as the number of days until the next holiday or the number of days remaining in the billing cycle.*

12. **Feature Selection:**

- *Use feature selection techniques to identify the most relevant features for your model. Eliminate redundant or unimportant features to reduce model complexity.*

13. **Domain-Specific Features:**

- *Consult with domain experts in the energy industry to identify domainspecific features that might influence electricity prices.*

14. **Time Series Decomposition:**

- *Decompose the time series data into trend, seasonality, and residual components using methods like seasonal decomposition of time series (STL) and use these components as features.*

15. **External Data Sources:**

- *Consider incorporating external data sources, such as economic indicators, news sentiment, or energy market reports, to enhance the model's predictive power.*

Model Training

Training a model for electricity price prediction involves several key steps. Here's a high-level overview of the process:

1. **Data Collection:** *Gather historical data on electricity prices. This data may include information such as time of day, season, weather conditions, demand, and more. High-quality and comprehensive data are crucial for accurate predictions.*

2. **Data Preprocessing:** *Clean and preprocess the data. This includes handling missing values, outliers, and encoding categorical variables. Time series data may also require specific preprocessing steps like resampling, differencing, or decomposing.*

3. **Feature Engineering**: *Create relevant features that can help the model capture patterns and trends in the data. Feature engineering can include lag features, moving averages, and seasonality indicators.*

4. **Splitting the Data:** *Divide your dataset into training, validation, and test sets. The training set is used to train the model, the validation set helps with hyperparameter tuning, and the test set is reserved for final model evaluation.*

5. **Model Selection:** *Choose an appropriate machine learning or statistical model for electricity price prediction. Common choices include regression models (e.g., linear regression, random forest, or gradient boosting), time series models (e.g., ARIMA, SARIMA, or Prophet), or deep learning models (e.g., recurrent neural networks or LSTM).*

6. **Model Training:** *Train the selected model using the training dataset. Ensure that the model optimizes a relevant loss function, such as mean squared error (MSE) for regression tasks. Adjust hyperparameters as needed to improve model performance.*

7. **Hyperparameter Tuning:** *Use techniques like grid search or random search to fine-tune hyperparameters. This process helps you find the best configuration for your model.*

8. **Model Validation:** *Evaluate the model's performance on the validation dataset using appropriate evaluation metrics. Adjust the model and repeat training if necessary.*

9. **Model Testing:** *Once you're satisfied with the model's performance on the validation set, test it on the reserved test set to assess how well it generalizes to new, unseen data.*

10. **Model Deployment:** *If the model meets your performance requirements, deploy it to make real-time predictions on new electricity price data. Ensure that the deployment environment is scalable and reliable.*

11. **Monitoring and Maintenance:** *Continuously monitor the model's performance in a production environment and update it as needed. Electricity prices can be influenced by various factors that may change over time, so model maintenance is crucial.*

12. **Interpretability and Visualization:** *Provide clear explanations of the model's predictions, and use visualization techniques to communicate insights to stakeholders.*

Model Evaluation

1. **Mean Absolute Error (MAE):** *Calculate the absolute differences between predicted and actual prices, and then take the mean. It measures the average magnitude of errors.*

2. **Mean Squared Error (MSE):** *Square the differences between predicted and actual prices, and then take the mean. MSE gives more weight to larger errors.*

3. **Root Mean Squared Error (RMSE):** *Take the square root of the MSE. It's in the same unit as the target variable and provides a clearer interpretation.*

4. **R-squared (R^2):** *This measures the proportion of variance in the target variable that's predictable from the features. A higher R-squared indicates a better fit.*

5. **Mean Absolute Percentage Error (MAPE):** *Calculate the percentage difference between predicted and actual prices, and then take the mean. It's useful when you want to understand the error as a percentage of the actual values.*

6. **Time Series-Specific Metrics:** *If your electricity price data is time-series data, you may want to use metrics like Mean Absolute Scaled Error (MASE), Seasonal decomposition of time series (STL), or Autocorrelation to assess model performance.*

7. **Cross-Validation:** *Split your dataset into training and testing subsets, using techniques like k-fold cross-validation, time series cross-validation, or walkforward validation. This helps you assess how well your model generalizes to new data.*

8. **Visual Inspection:** *Plot the predicted prices against the actual prices to visually assess how well the model captures trends and patterns.*

9. **Residual Analysis:** *Examine the residuals (the differences between actual and predicted prices) for any patterns or autocorrelation. This can help identify model deficiencies.*

10. **Domain Expertise:** *Consulting with domain experts in the energy industry can provide valuable insights into whether your model's predictions make practical sense.*

PROGRAM

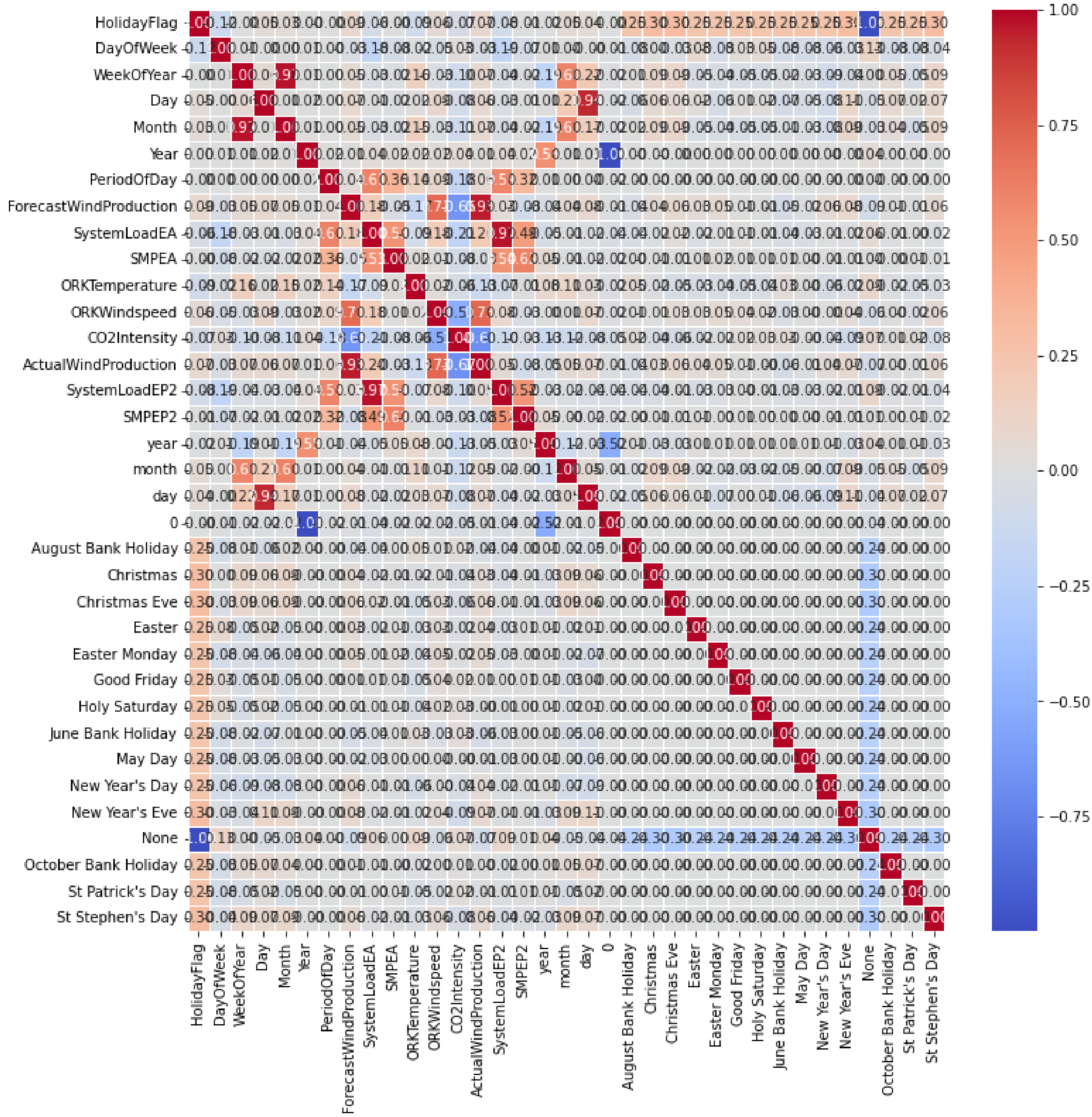
Correlation Analysis:

In[1]:

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

```
sns.heatmap(df.corr(),annot=True,linewidths=0.7,fmt=".2f",cmap="coolwarm")
plt.show()
```

Out[1]:



In[2]:

```
cor=df.corr()["SMPEP2"].sort_values(ascending=False)
pd.DataFrame({"column":cor.index,"Correlation with a":cor.values})
```

Out[2]:

	column	Correlation with a
0	SMPEP2	1.000000
1	SMPEA	0.617234
2	SystemLoadEP2	0.516938
3	SystemLoadEA	0.490945
4	PeriodOfDay	0.323052
5	year	0.047701
6	Year	0.017688
7	Easter	0.014242
8	St Patrick's Day	0.012972
9	Good Friday	0.011269
10	None	0.006365
11	May Day	0.004863
12	June Bank Holiday	0.004235
13	Holy Saturday	0.003777
14	October Bank Holiday	0.003084
15	Easter Monday	-0.001341
16	month	-0.001578

17	August Bank Holiday	-0.003159
18	HolidayFlag	-0.005645
19	ORKTemperature	-0.008571
20	New Year's Day	-0.009114
21	Christmas Eve	-0.009609
22	Christmas	-0.011435
23	New Year's Eve	-0.011679
24	Day	-0.013459
25	Month	-0.015255
26	0	-0.016091
27	WeekOfYear	-0.016170
28	St Stephen's Day	-0.018729
29	day	-0.019355
30	CO2Intensity	-0.033225
31	ORKWindspeed	-0.034614
32	DayOfWeek	-0.069597
33	ForecastWindProduction	-0.079611
34	ActualWindProduction	-0.082813

Modeling:

In[3]:

```
X=df.drop("SMPEP2",axis=1)  y=df["SMPEP2"]
```

In[4]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
3,random_state=0)
```

In[5]:

```
!pip install catboost
```

Out[5]:

```
Requirement already satisfied: catboost in /opt/conda/lib/python3.7/site-packages (1
```

```
.1)
Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages (from
catboost) (1.7.3)
Requirement already satisfied: graphviz in /opt/conda/lib/python3.7/site-packages (f
rom catboost) (0.8.4)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.7/site-packages
(from catboost) (3.5.3)
Requirement already satisfied: numpy>=1.16.0 in /opt/conda/lib/python3.7/site-packag
es (from catboost) (1.21.6)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from c
atboost) (1.15.0)
Requirement already satisfied: pandas>=0.24.0 in /opt/conda/lib/python3.7/site-packa
ges (from catboost) (1.3.5)
Requirement already satisfied: plotly in /opt/conda/lib/python3.7/site-packages (fro
m catboost) (5.10.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/conda/lib/python3.7/si
te-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.7/site-package
s (from pandas>=0.24.0->catboost) (2022.1)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packag
es (from matplotlib->catboost) (9.1.1)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.7/site-pack
ages (from matplotlib->catboost) (21.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-package
s (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.7/site-pa
ckages (from matplotlib->catboost) (4.33.3)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-pac
kages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-pa
ckages (from matplotlib->catboost) (1.4.3)
Requirement already satisfied: tenacity>=6.2.0 in /opt/conda/lib/python3.7/site-pack
ages (from plotly->catboost) (8.0.1)
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.7/site-pa
ckages (from kiwisolver>=1.0.1->matplotlib->catboost) (4.4.0)
```

In[6]:

```
!pip install lightgbm
```

Out[6]:

```
Requirement already satisfied: lightgbm in /opt/conda/lib/python3.7/site-packages (3
.3.2)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from
lightgbm) (1.21.6)
Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages (from
lightgbm) (1.7.3)
Requirement already satisfied: scikit-learn!=0.22.0 in /opt/conda/lib/python3.7/site
-packages (from lightgbm) (1.0.2)
```

```
Requirement already satisfied: wheel in /opt/conda/lib/python3.7/site-packages (from lightgbm) (0.37.1)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.7/site-packages (from scikit-learn!=0.22.0->lightgbm) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from scikit-learn!=0.22.0->lightgbm) (3.1.0)
```

In[7]:

```
!pip install xgboost
```

Out[7]:

```
Requirement already satisfied: xgboost in /opt/conda/lib/python3.7/site-packages (1.6.2)
Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages (from xgboost) (1.7.3)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from xgboost) (1.21.6)
```

In[8]:

```
from xgboost import XGBRegressor from
catboost import CatBoostRegressor from
lightgbm import LGBMRegressor
```

In[9]:

```
ridge=Ridge().fit(X_train,y_train)
lasso=Lasso().fit(X_train,y_train)
enet=ElasticNet().fit(X_train,y_train)
knn=KNeighborsRegressor().fit(X_train,y_train)
ada=AdaBoostRegressor().fit(X_train,y_train)
```

In[10]:

```
svm=SVR().fit(X_train,y_train)
mlpc=MLPRegressor().fit(X_train,y_train)
```

```
dtc=DecisionTreeRegressor().fit(X_train,y_train)
rf=RandomForestRegressor().fit(X_train,y_train)
xgb=XGBRegressor().fit(X_train,y_train)
gbm=GradientBoostingRegressor().fit(X_train,y_train)
lgb=LGBMRegressor().fit(X_train,y_train)
catbost=CatBoostRegressor().fit(X_train,y_train)
```

Out[10]:

```
14:	learn: 27.6408000	total: 138ms	remaining: 9.07s
15:	learn: 27.4359313	total: 143ms	remaining: 8.8s
16:	learn: 27.2173157	total: 148ms	remaining: 8.55s
17:	learn: 27.0422970	total: 153ms	remaining: 8.36s
18:	learn: 26.8771233	total: 158ms	remaining: 8.16s
19:	learn: 26.6893652	total: 164ms	remaining: 8.03s
20:	learn: 26.5538529	total: 169ms	remaining: 7.86s
21:	learn: 26.4492316	total: 174ms	remaining: 7.71s
22:	learn: 26.3094102	total: 178ms	remaining: 7.58s
23:	learn: 26.1955562	total: 184ms	remaining: 7.47s
24:	learn: 26.0928013	total: 189ms	remaining: 7.39s
25:	learn: 25.9962431	total: 195ms	remaining: 7.3s
26:	learn: 25.9124311	total: 200ms	remaining: 7.19s
27:	learn: 25.8302408	total: 204ms	remaining: 7.09s
28:	learn: 25.7387686	total: 209ms	remaining: 7s
29:	learn: 25.6728034	total: 214ms	remaining: 6.92s
30:	learn: 25.5926581	total: 219ms	remaining: 6.84s
```

```
31:	learn: 25.4940606	total: 224ms	remaining: 6.77s
32:	learn: 25.4182581	total: 229ms	remaining: 6.7s
33:	learn: 25.3652752	total: 233ms	remaining: 6.63s
34:	learn: 25.3223579	total: 238ms	remaining: 6.56s
35:	learn: 25.2489125	total: 243ms	remaining: 6.5s
36:	learn: 25.2076996	total: 247ms	remaining: 6.43s
37:	learn: 25.1667858	total: 252ms	remaining: 6.38s
38:	learn: 25.1180288	total: 257ms	remaining: 6.34s
39:	learn: 25.0752794	total: 262ms	remaining: 6.3s
40:	learn: 25.0339522	total: 267ms	remaining: 6.25s
41:	learn: 24.9977064	total: 272ms	remaining: 6.2s
42:	learn: 24.9697517	total: 276ms	remaining: 6.15s
43:	learn: 24.9190191	total: 281ms	remaining: 6.11s
44:	learn: 24.8957832	total: 285ms	remaining: 6.06s
45:	learn: 24.8597053	total: 290ms	remaining: 6.01s
46:	learn: 24.8392094	total: 294ms	remaining: 5.97s
47:	learn: 24.8102239	total: 299ms	remaining: 5.92s
48:	learn: 24.7789045	total: 303ms	remaining: 5.89s
49:	learn: 24.7500880	total: 308ms	remaining: 5.85s
50:	learn: 24.7178926	total: 313ms	remaining: 5.82s
51:	learn: 24.6968006	total: 317ms	remaining: 5.78s
52:	learn: 24.6800087	total: 321ms	remaining: 5.74s
53:	learn: 24.6507233	total: 326ms	remaining: 5.71s
54:	learn: 24.6186034	total: 331ms	remaining: 5.68s
55:	learn: 24.5907057	total: 335ms	remaining: 5.65s
56:	learn: 24.5572812	total: 340ms	remaining: 5.63s
57:	learn: 24.5264215	total: 345ms	remaining: 5.6s
58:	learn: 24.4987088	total: 350ms	remaining: 5.58s
59:	learn: 24.4794572	total: 355ms	remaining: 5.56s
60:	learn: 24.4427210	total: 359ms	remaining: 5.53s
61:	learn: 24.4156283	total: 364ms	remaining: 5.51s
62:	learn: 24.3917659	total: 369ms	remaining: 5.49s
63:	learn: 24.3642679	total: 374ms	remaining: 5.47s
64:	learn: 24.3450265	total: 379ms	remaining: 5.45s
```

```
65:	learn: 24.3204788	total: 383ms	remaining: 5.42s
66:	learn: 24.2830890	total: 389ms	remaining: 5.41s
67:	learn: 24.2650713	total: 393ms	remaining: 5.39s
68:	learn: 24.2399899	total: 397ms	remaining: 5.36s
69:	learn: 24.2164965	total: 402ms	remaining: 5.34s
70:	learn: 24.1964369	total: 407ms	remaining: 5.32s
71:	learn: 24.1807660	total: 412ms	remaining: 5.31s
72:	learn: 24.1689245	total: 417ms	remaining: 5.29s
73:	learn: 24.1436655	total: 421ms	remaining: 5.27s
74:	learn: 24.1253300	total: 426ms	remaining: 5.25s
75:	learn: 24.0935058	total: 430ms	remaining: 5.23s
76:	learn: 24.0763530	total: 435ms	remaining: 5.21s
77:	learn: 24.0532679	total: 439ms	remaining: 5.19s
78:	learn: 24.0366265	total: 443ms	remaining: 5.17s
79:	learn: 24.0167814	total: 448ms	remaining: 5.15s
80:	learn: 23.9978824	total: 453ms	remaining: 5.14s
81:	learn: 23.9867891	total: 457ms	remaining: 5.12s
82:	learn: 23.9763622	total: 462ms	remaining: 5.1s
83:	learn: 23.9460132	total: 467ms	remaining: 5.09s
84:	learn: 23.9186528	total: 471ms	remaining: 5.07s
85:	learn: 23.9048177	total: 476ms	remaining: 5.06s
86:	learn: 23.8918401	total: 481ms	remaining: 5.05s
87:	learn: 23.8653562	total: 486ms	remaining: 5.04s
88:	learn: 23.8508496	total: 491ms	remaining: 5.02s
89:	learn: 23.8371390	total: 495ms	remaining: 5.01s
90:	learn: 23.8302205	total: 500ms	remaining: 4.99s
91:	learn: 23.8033151	total: 504ms	remaining: 4.98s
92:	learn: 23.7824900	total: 509ms	remaining: 4.96s
93:	learn: 23.7735478	total: 514ms	remaining: 4.96s
94:	learn: 23.7586709	total: 519ms	remaining: 4.94s
95:	learn: 23.7389694	total: 523ms	remaining: 4.93s
96:	learn: 23.7269390	total: 528ms	remaining: 4.91s
97:	learn: 23.7126293	total: 532ms	remaining: 4.9s
```

```
98:     learn: 23.6890770       total: 537ms    remaining: 4.88s
99:     learn: 23.6685269       total: 541ms    remaining: 4.87s
100:    learn: 23.6485375       total: 546ms    remaining: 4.86s
101:    learn: 23.6257557       total: 551ms    remaining: 4.85s
102:    learn: 23.6065188       total: 556ms    remaining: 4.84s
103:    learn: 23.5901493       total: 561ms    remaining: 4.83s
104:    learn: 23.5784682       total: 565ms    remaining: 4.82s
105:    learn: 23.5614537       total: 570ms    remaining: 4.81s
106:    learn: 23.5429277       total: 574ms    remaining: 4.79s
107:    learn: 23.5283062       total: 579ms    remaining: 4.78s
108:    learn: 23.5054101       total: 584ms    remaining: 4.77s
109:    learn: 23.4962552       total: 590ms    remaining: 4.78s
110:    learn: 23.4881442       total: 595ms    remaining: 4.77s
111:    learn: 23.4706061       total: 600ms    remaining: 4.76s
112:    learn: 23.4534164       total: 605ms    remaining: 4.75s
113:    learn: 23.4365195       total: 609ms    remaining: 4.73s
114:    learn: 23.4195741       total: 614ms    remaining: 4.72s
```

In[11]:

```
models=[ridge,lasso,dtc,rf,xgb,gbm,lgb,catbost,enet,knn,ada,mlp
c,svm]
```

In[12]:

```
def ML(y,models):
accuary=models.score(X_train,y_train)
return accuary
```

In[13]:

```
for i in
models:
  print(i,"Algorithm succed rate :",ML("SMPEP2",i))
```

Out[13]:

```
Ridge() Algorithm succed rate : 0.43121105926644243
```

```
Lasso() Algorithm succed rate : 0.42883198265818245
DecisionTreeRegressor() Algorithm succed rate : 1.0
RandomForestRegressor() Algorithm succed rate : 0.9424727172628374
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
early_stopping_rounds=None, enable_categorical=False,
eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
importance_type=None, interaction_constraints='',
learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0,
num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
reg_lambda=1, ...) Algorithm succed rate : 0.8732530340524252
GradientBoostingRegressor() Algorithm succed rate : 0.5739399134995518
LGBMRegressor() Algorithm succed rate : 0.6953551703738294
<catboost.core.CatBoostRegressor object at 0x7f29e8bcc0d0> Algorithm succed rate : 0
.7878389350009978
ElasticNet() Algorithm succed rate : 0.4290970871174
KNeighborsRegressor() Algorithm succed rate : 0.5964451293083145
AdaBoostRegressor() Algorithm succed rate : 0.26365965295085403
MLPRegressor() Algorithm succed rate : 0.15355550237922466
SVR() Algorithm succed rate : 0.23458514968207922
```

In[14]:

```
cor=df.corr()["SMPEP2"].sort_values(ascending=False)
pd.DataFrame({"column":cor.index,"Correlation with
a":cor.values})
```

Out[14]:

	column	Correlation with a
0	SMPEP2	1.000000
1	SMPEA	0.617234
2	SystemLoadEP2	0.516938
3	SystemLoadEA	0.490945
4	PeriodOfDay	0.323052
5	year	0.047701
6	Year	0.017688
7	Easter	0.014242
8	St Patrick's Day	0.012972
9	Good Friday	0.011269
10	None	0.006365
11	May Day	0.004863
12	June Bank Holiday	0.004235
13	Holy Saturday	0.003777
14	October Bank Holiday	0.003084
15	Easter Monday	-0.001341
16	month	-0.001578
17	August Bank Holiday	-0.003159
18	HolidayFlag	-0.005645
19	ORKTemperature	-0.008571
20	New Year's Day	-0.009114
21	Christmas Eve	-0.009609
22	Christmas	-0.011435
23	New Year's Eve	-0.011679
24	Day	-0.013459
25	Month	-0.015255
26	0	-0.016091
27	WeekOfYear	-0.016170
28	St Stephen's Day	-0.018729
29	day	-0.019355
30	CO2Intensity	-0.033225
31	ORKWindspeed	-0.034614

In[15]:

```
X2=df[["SMPEA","SystemLoadEP2","SystemLoadEA","PeriodOfDay",
"year","ActualWindProduction"]] y2=df["SMPEP2"]
```

In[16]:

```
X_train2,X_test2,y_train2,y_test2=train_test_split(X2,y2,test_s
ize=0.3,random_state=0)
```

In[17]:

```
rf2=RandomForestRegressor().fit(X_train2,y_train2)
```

In[18]:

```
rf2.score(X_train2,y_train2)
```

Out[18]:

```
0.9345853165856398
```

In[19]:

```
X3=df_remove_out.drop("SMPEP2",axis=1)
y3=df_remove_out["SMPEP2"]
```

In[20]:

```
X_train3,X_test3,y_train3,y_test3=train_test_split(X3,y3,test_s
ize=0.3,random_state=0)
```

In[21]:

```
rf3=RandomForestRegressor().fit(X_train3,y_train3)
```

In[22]:

```
rf3.score(X_train,y_train)
```

Out[22]:

```
0.8965242074080007
```

In[23]:

```
dtc3=DecisionTreeRegressor().fit(X_train3,y_train3)
```

In[24]:

```
rf3.score(X_train3,y_train3)
```

Out[24]:

```
0.9525199962605772
```

Random Forest:

In[1]:

```
!pip install hyperopt
from hyperopt import tpe,STATUS_OK,Trials,fmin,hp from
hyperopt.pyll.base import scope
```

Out[1]:

```
Requirement already satisfied: hyperopt in /opt/conda/lib/python3.7/site-packages (0
.2.7)
Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages (from
hyperopt) (1.7.3)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages (from
hyperopt) (4.64.0)
Requirement already satisfied: py4j in /opt/conda/lib/python3.7/site-packages (from
hyperopt) (0.10.9.7)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from
hyperopt) (1.21.6)
```

```
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from h
yperopt) (1.15.0)
Requirement already satisfied: cloudpickle in /opt/conda/lib/python3.7/site-packages
 (from hyperopt) (2.1.0)
Requirement already satisfied: networkx>=2.2 in /opt/conda/lib/python3.7/site-packag
es (from hyperopt) (2.5)
Requirement already satisfied: future in /opt/conda/lib/python3.7/site-packages (fro
m hyperopt) (0.18.2)
Requirement already satisfied: decorator>=4.3.0 in /opt/conda/lib/python3.7/site-pac
kages (from networkx>=2.2->hyperopt) (5.1.1)
```

In[2]:

```
space={
  "max_depth":hp.randint("max_depth",2,15),
  "min_samples_split":hp.randint("min_samples_split",2,20),
  "min_samples_leaf":hp.randint("min_samples_leaf",1,20),
  "n_estimators":hp.randint("n_estimators",50,1000)


}
```

In[3]: `def hyperparameter_tuning(params):`

```
clf=RandomForestRegressor(**params).fit(X_train,y_train)
acc=rf.score(X_train,y_train)   return acc
```

In[4]:

```
trials=Trials()


best=fmin(fn=hyperparameter_tuning,
     space=space,
     algo=tpe.suggest,max_evals=100,trials=trials
     )
print("best:{}".format(best))
```

In[5]: `best`

Out[5]:

```
{'max_depth': 12,
 'min_samples_leaf': 2,
 'min_samples_split': 8,
 'n_estimators': 303}
```

Conclusion

In conclusion, the prediction of electricity prices is a complex yet critical endeavor, influencing various stakeholders within the energy market. By leveraging advanced methodologies and data-driven approaches, accurate price forecasting can yield numerous benefits, including cost reduction, resource optimization, market stability, and improved decision-making.

Machine learning, supported by robust data collection, preprocessing, and model development, stands at the forefront of enhancing predictive accuracy. Incorporating design thinking principles promotes user-centric solutions and adaptive models that evolve with dynamic market conditions.

The continual pursuit of innovation in this field involves integrating advanced technologies, exploring unconventional data sources, and fostering collaborative partnerships to ensure a more comprehensive and adaptable predictive framework.

Efficient data loading, thorough preprocessing, and model validation contribute significantly to the creation of reliable predictive models. These models, when continuously refined and iterated upon, cater to the evolving nature of the electricity market, empowering stakeholders to make informed decisions and optimize resource allocation.

As the energy landscape continues to evolve, refining predictive models for electricity prices remains an ongoing pursuit, encouraging ethical, transparent, and responsible innovation to meet the ever-changing needs of a dynamic energy market.