ELECTRICITY PRICES PREDICTION

TEAM MEMBER

ANANDA M P

# 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.*

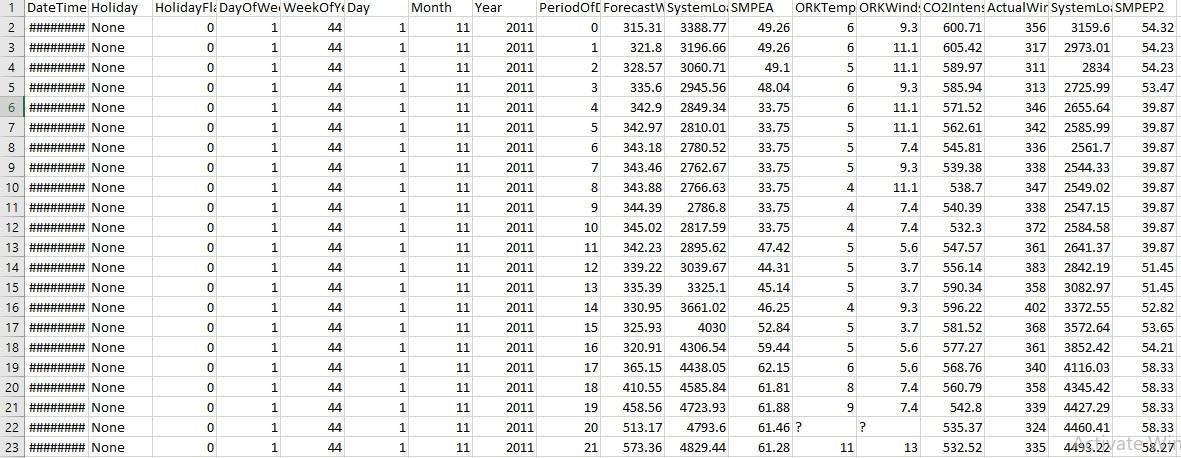
*This discussion or project aims to delve into the intricate world of electricity price prediction. We will explore the multifaceted factors that influence pricing, from supply and demand patterns to environmental conditions and regulatory policies. Through the lens of data-driven approaches, machine 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



**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*

1. **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.*

1. **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.*

1. **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.*

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

1. **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.*

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

1. **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.*

1. **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.*

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

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

1. **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.*

1. **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.*

1. **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, realtime forecasting, adaptability, and user-friendly outputs.*

1. **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.*

1. **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.*

1. **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.*

1. **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.*

1. **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:*

o *Mean Squared Error (MSE)* o *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.*

1. **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.*

1. **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.*

1. **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).*

1. **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.*

1. **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.*

1. **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.*

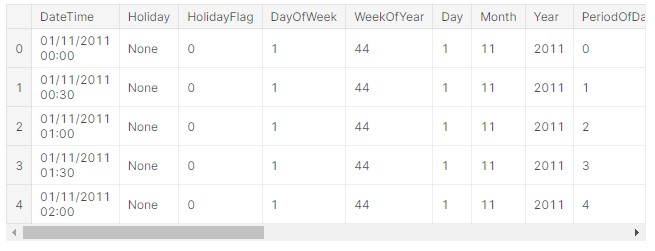
**Program:**

**Data Loading:**

**In[1]:**

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

**Out[1]:**



**In[2]:**

df.tail()

**Out[2]:**



**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*

*--- ------ -------------- -----*

1. *DateTime 38014 non-null object*
2. *Holiday 38014 non-null object*
3. *HolidayFlag 38014 non-null int64*
4. *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

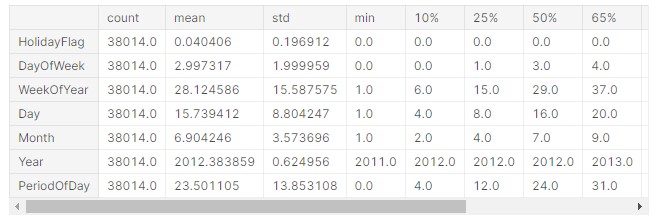
**Out[7]:**



**In[8]:**

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

**Out[8]:**



**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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. 36478
2. 1536

Name: HolidayFlag, dtype: int64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. 5472
2. 5424
3. 5424
4. 5424
5. 5424

0 5424

6 5422

Name: DayOfWeek, dtype: int64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. 1008
2. 1008
3. 1008
4. 1008
5. 1008
6. 1008
7. 1008
8. 1008

44 960

1 768

1. 672
2. 672

33 672

1. 672
2. 672
3. 672
4. 672
5. 672
6. 672

37 672

1. 672
2. 672
3. 672

23 672

1. 672
2. 672
3. 672
4. 672
5. 672

24 672

18 672

22 672

10 672

1. 672
2. 672
3. 672
4. 672
5. 672
6. 672
7. 672
8. 672

11 672

21 672

1. 672
2. 672
3. 672
4. 672
5. 672
6. 672
7. 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

1. 1200
2. 1152
3. 720

Name: Day, dtype: int64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12 4464

11 4320

1 2976

5 2976

1. 2976
2. 2976

10 2976

1. 2974
2. 2880

6 2880

9 2880

2 2736

Name: Month, dtype: int64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. 17566
2. 17520

2011 2928

Name: Year, dtype: int64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. 792
2. 792
3. 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
22. 792
23. 792

25 792

24 792

23 792

22 792

1. 792
2. 792
3. 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

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

--- ------ -------------- -----

1. DateTime 38014 non-null object
2. Holiday 38014 non-null object
3. HolidayFlag 38014 non-null int64
4. DayOfWeek 38014 non-null int64
5. WeekOfYear 38014 non-null int64
6. Day 38014 non-null int64
7. Month 38014 non-null int64
8. Year 38014 non-null int64
9. PeriodOfDay 38014 non-null int64
10. ForecastWindProduction 38009 non-null float64
11. SystemLoadEA 38012 non-null float64
12. SMPEA 38012 non-null float64
13. ORKTemperature 37719 non-null float64
14. ORKWindspeed 37715 non-null float64
15. CO2Intensity 38007 non-null float64
16. ActualWindProduction 38009 non-null float64
17. 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]:**

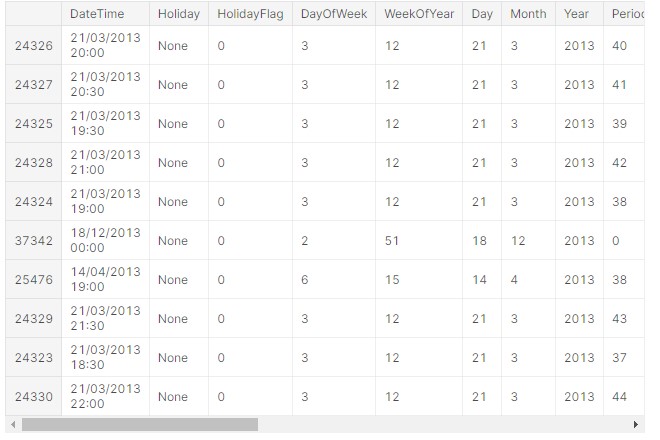
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | cou  nt | mean | std | mi  n | 5% | 10  % | 25% | 35% | 50  % | 65  % | 75% | 90  % | 95% | 98% | ma  x |
| HolidayFla g | 380 14.  0 | 0.040  406 | 0.19  6912 | 0.0  0 | 0.00  00 | 0.00  0 | 0.00  00 | 0.00  00 | 0.00  0 | 0.00  0 | 0.00  00 | 0.00  0 | 0.00  00 | 1.00  00 | 1.0  0 |
| DayOfWee  k | 380 14.  0 | 2.997  317 | 1.99  9959 | 0.0  0 | 0.00  00 | 0.00  0 | 1.00  00 | 2.00  00 | 3.00  0 | 4.00  0 | 5.00  00 | 6.00  0 | 6.00  00 | 6.00  00 | 6.0  0 |
| WeekOfYe  ar | 380 14.  0 | 28.12  4586 | 15.5  8757  5 | 1.0  0 | 3.00  00 | 6.00  0 | 15.0  000 | 20.0  000 | 29.0  00 | 37.0  00 | 43.0  000 | 49.0  00 | 51.0  000 | 52.0  000 | 52.  00 |
| Day | 380 14.  0 | 15.73  9412 | 8.80  4247 | 1.0  0 | 2.00  00 | 4.00  0 | 8.00  00 | 11.0  000 | 16.0  00 | 20.0  00 | 23.0  000 | 28.0  00 | 29.0  000 | 30.0  000 | 31.  00 |
| Month | 380 14.  0 | 6.904  246 | 3.57  3696 | 1.0  0 | 1.00  00 | 2.00  0 | 4.00  00 | 5.00  00 | 7.00  0 | 9.00  0 | 10.0  000 | 12.0  00 | 12.0  000 | 12.0  000 | 12.  00 |
| Year | 380 14.  0 | 2012.  3838  59 | 0.62  4956 | 201  1.0  0 | 201  1.00  00 | 201  2.00  0 | 201  2.00  00 | 201  2.00  00 | 201  2.00  0 | 201  3.00  0 | 201  3.00  00 | 201  3.00  0 | 201  3.00  00 | 201  3.00  00 | 201  3.0  0 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | cou  nt | mean | std | mi  n | 5% | 10  % | 25% | 35% | 50  % | 65  % | 75% | 90  % | 95% | 98% | ma  x |
| PeriodOfD ay | 380 14.  0 | 23.50  1105 | 13.8  5310  8 | 0.0  0 | 2.00  00 | 4.00  0 | 12.0  000 | 16.0  000 | 24.0  00 | 31.0  00 | 35.7  500 | 43.0  00 | 45.0  000 | 47.0  000 | 47.  00 |
| ForecastWi ndProducti on | 380 09.  0 | 544.2  6145  1 | 414.  3646  29 | 0.6  8 | 52.3  220 | 80.3  76 | 189.  670  0 | 279.  300  0 | 441.  980 | 649.  992 | 839.  460  0 | 120  6.11  0 | 135  2.44  00 | 144  1.45  80 | 168  0.0  0 |
| SystemLoa dEA | 380 12.  0 | 4020.  0850  19 | 860.  4768  66 | 218  3.9  4 | 262  6.24  95 | 281  2.03  3 | 328  1.20  75 | 364  6.83  35 | 410  3.60  0 | 446  7.16  0 | 463  8.53  25 | 509  3.91  2 | 536  7.07  60 | 571  3.10  28 | 649  2.9  1 |
| SMPEA | 380 12.  0 | 62.72  0388 | 32.2  5233  4 | 0.0  0 | 33.8  200 | 38.3  60 | 45.5  300 | 49.1  900 | 55.2  30 | 63.6  10 | 70.3  200 | 90.6  27 | 110.  229  0 | 154.  837  8 | 587  .58 |
| ORKTemp  erature | 377 19.  0 | 9.626  369 | 4.43  9934 | 4.0  0 | 3.00  00 | 4.00  0 | 6.00  00 | 8.00  00 | 9.00  0 | 11.0  00 | 13.0  000 | 16.0  00 | 17.0  000 | 19.0  000 | 25.  00 |
| ORKWind  speed | 377 15.  0 | 19.21  1770 | 9.57  1311 | 0.0  0 | 5.60  00 | 7.40  0 | 13.0  000 | 14.8  000 | 18.5  00 | 22.2  00 | 24.1  000 | 31.5  00 | 37.0  000 | 42.6  000 | 75.  90 |
| CO2Intensi  ty | 380 07.  0 | 479.3  7304  0 | 85.3  5470  6 | 0.0  0 | 336.  093  0 | 367.  216 | 421.  105  0 | 446.  880  0 | 480.  310 | 512.  620 | 537.  520  0 | 587.  870 | 619.  191  0 | 656.  045  2 | 842  .88 |
| ActualWin dProductio n | 380 09.  0 | 520.7 6281  9 | 378.  2829  75 | 1.0  0 | 43.0  000 | 77.0  00 | 199. 000  0 | 287. 800  0 | 445.  000 | 637.  000 | 793. 000  0 | 109  8.00  0 | 124  3.00  00 | 134  9.00  00 | 176 9.0  0 |
| SystemLoa dEP2 | 380 12.  0 | 3785.  9738 41 | 843.  2694 55 | 180  9.9  6 | 244  9.54  10 | 259  9.07  0 | 305  8.27  75 | 340  3.70  40 | 386  5.74  5 | 423  9.70  9 | 442  7.59  00 | 483  0.29  4 | 508  7.54  30 | 539  7.77  58 | 630  9.7  5 |
|  | cou  nt | mean | std | mi  n | 5% | 10  % | 25% | 35% | 50  % | 65  % | 75% | 90  % | 95% | 98% | ma  x |
| SMPEP2 | 380 12.  0 | 64.1  3682  3 | 35.4  1503  6 | -  47. 74 | 33.1  900 | 37.  922 |  |  |  |  |  |  |  |  |  |

**In[17]:**

df.sort\_values("ForecastWindProduction",ascending=False).head(1 0)

**Out[17]:**



**In[18]:**

df.sort\_values("ForecastWindProduction").head(10)

**Out[18]:**



**In[19]:**

|  |
| --- |
|  |

df.sort\_values("ORKWindspeed",ascending=False).head(10)

# highest

**Out[19]:**



**In[20]:**

df.sort\_values("ORKWindspeed").head(10) # lowest

**Out[20]:**



**In[21]:**

df.sort\_values("SMPEA",ascending=False).head(10) # highest

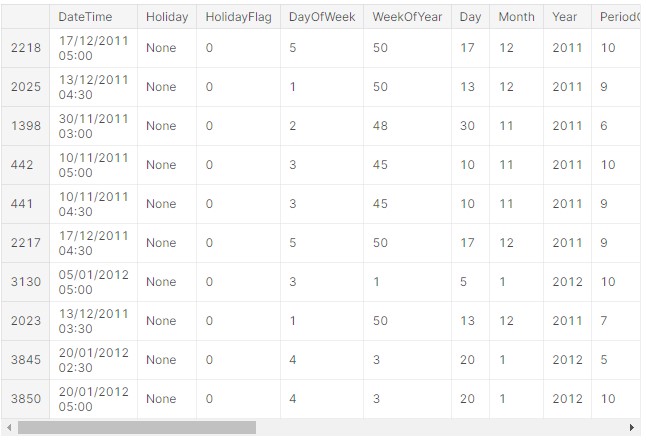
**Out[21]:**



**In[22]:**

df.sort\_values("SMPEA").head(10) # lowest

**Out[22]:**



**In[23]:**

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

**Out[23]:**



**In[24]:**

df[df.SMPEP2==-47.74]

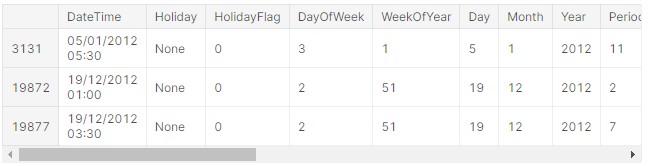
**Out[24]:**



**In[25]:**

df[df.SMPEP2<0]

**Out[25]:**



**In[26]:**

df[df.SMPEP2==1000]

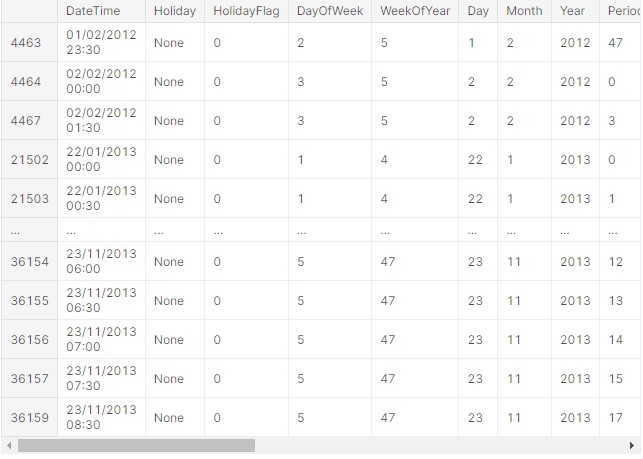
**Out[26]:**



**In[27]:**

df[df.ORKTemperature<0]

**Out[27]:**



**In[28]:**

df[df.ORKTemperature==25]

**Out[28]:**



**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")

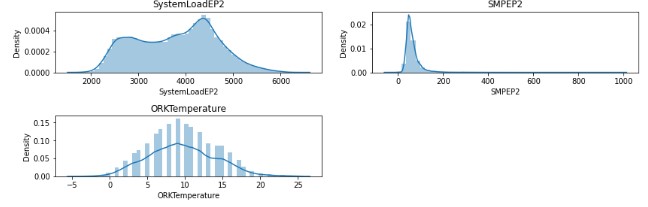
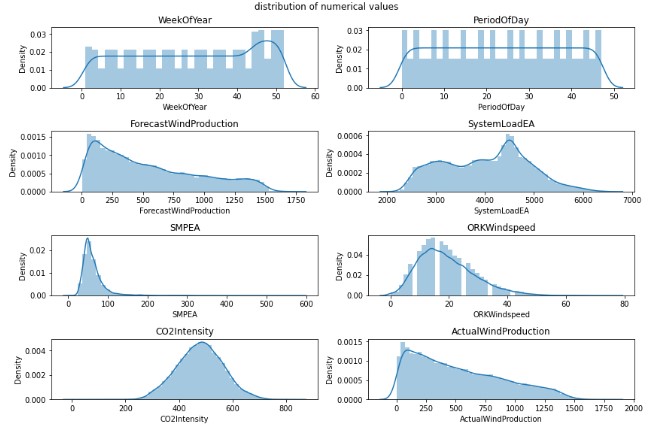
**In [8]:**

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:**

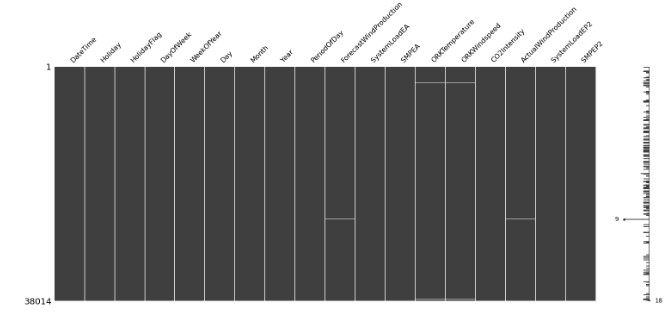
**In [1]:**

import missingno as msno

**In [2]:**

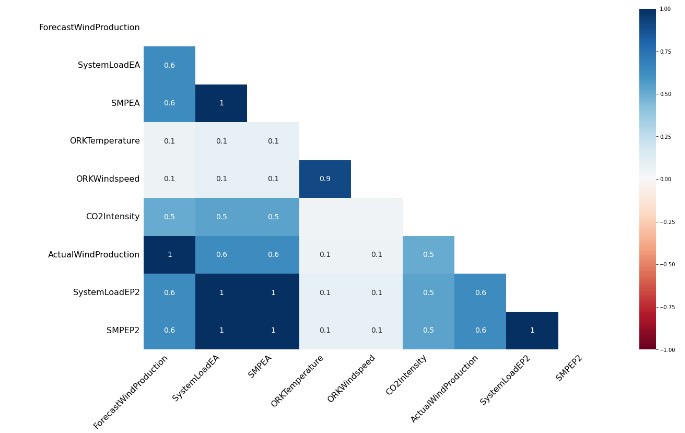
linkcode

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

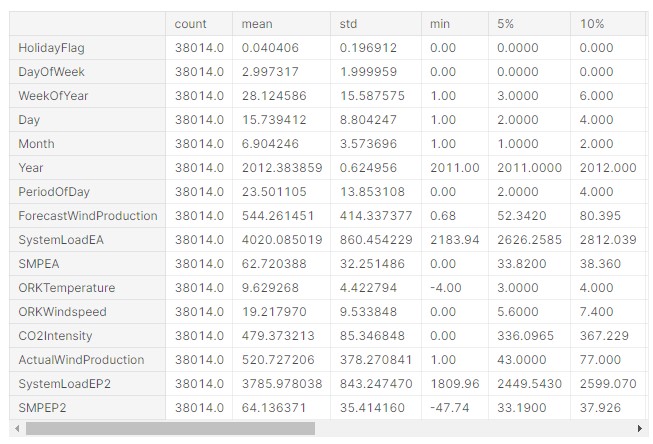
**Outlier Problem:**

**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]:**



**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

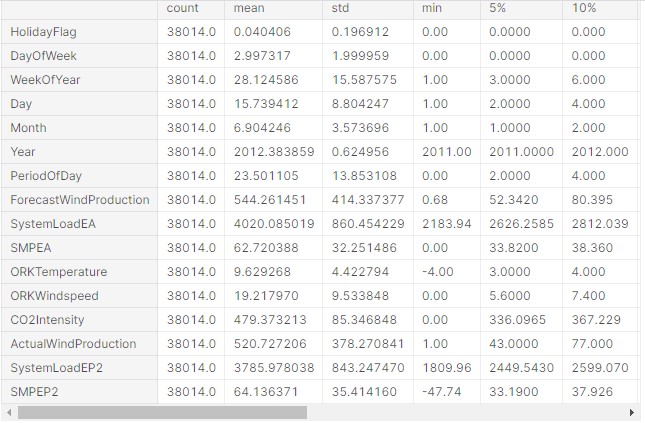
ORKWindspeed no

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]:**



**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]:**



**In[13]:**

df[df.SMPEP2<0]=0

**In[14]:**

linkcode df\_remove\_out[df\_remove\_out.SMPEP2<0]=0

**Time Series Analysis:**

**In[1]:**

from datetime import datetime

df["DateTime"] = pd.to\_datetime(df.DateTime)

**In [2]:**

linkcode

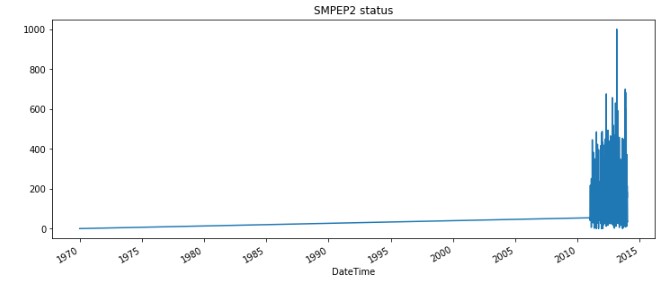
df['year'] = df['DateTime'].dt.year df['month'] = df['DateTime'].dt.month

df["day"]=df["DateTime"].dt.day

**In[3]:**

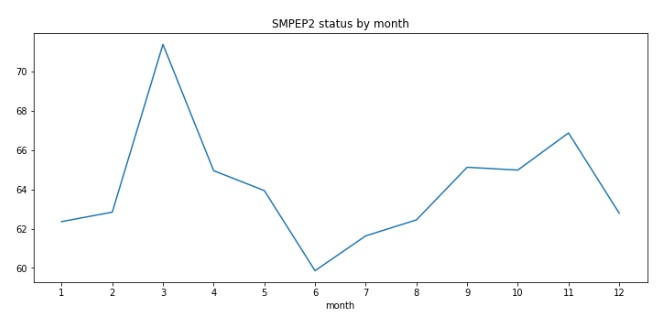
custgroup=df.groupby('DateTime').mean() plt.figure(figsize=(12,5)) custgroup['SMPEP2'].plot(x=df.DateTime)

plt.title("SMPEP2 status") plt.show()



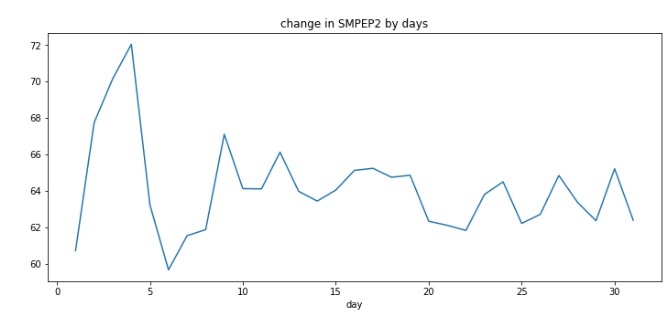
**In[4]:**

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





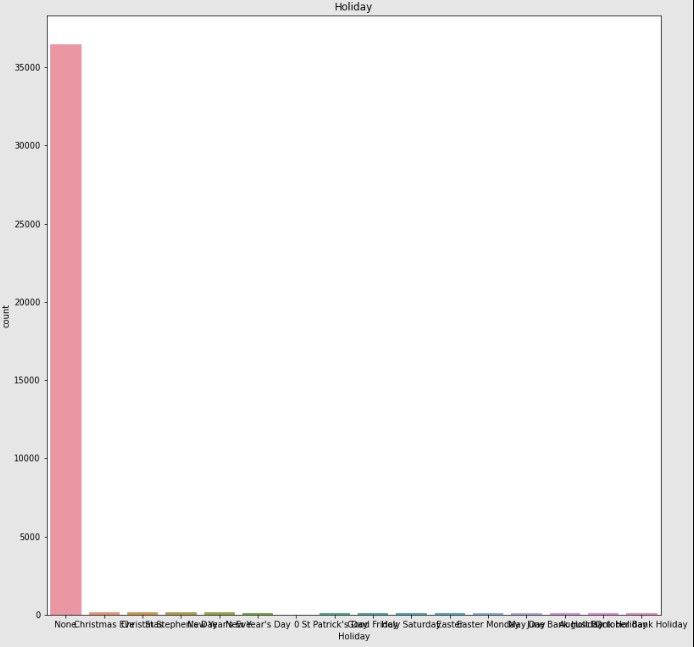
**Data Visualize:**

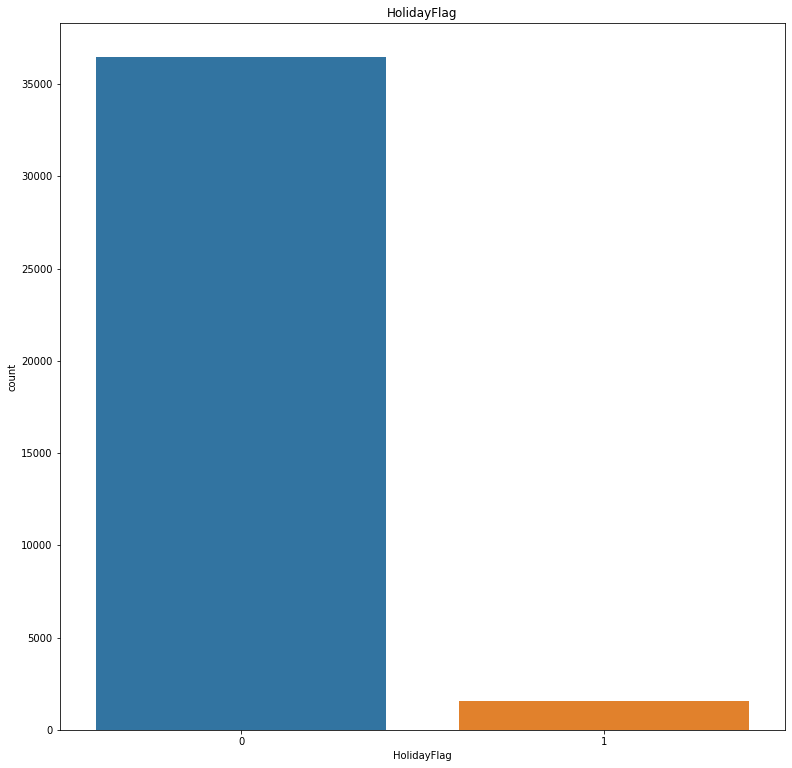
**In[1]:**

for i **in** cat\_list:

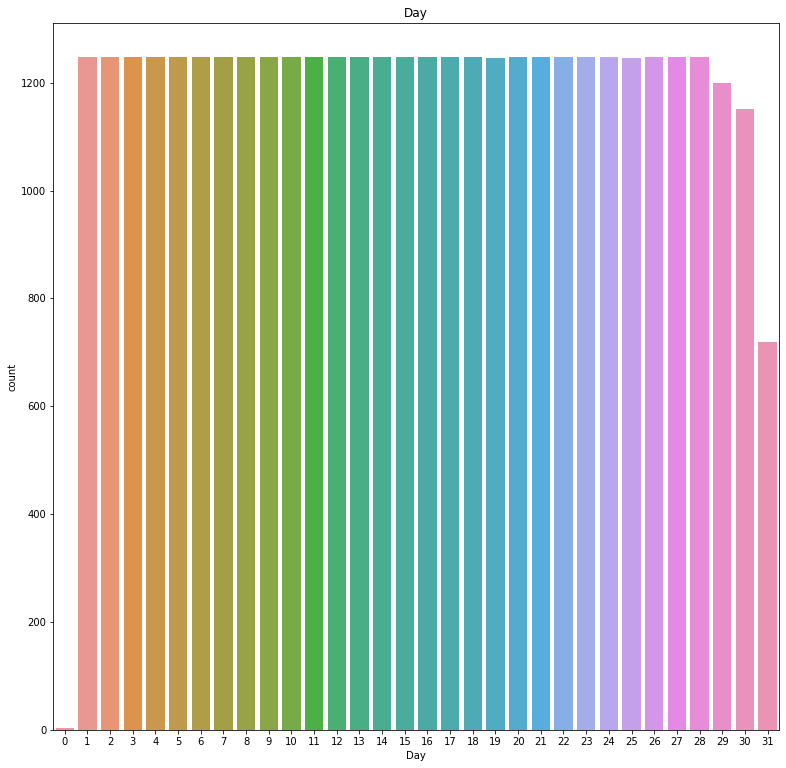
plt.figure(figsize=(13,13))

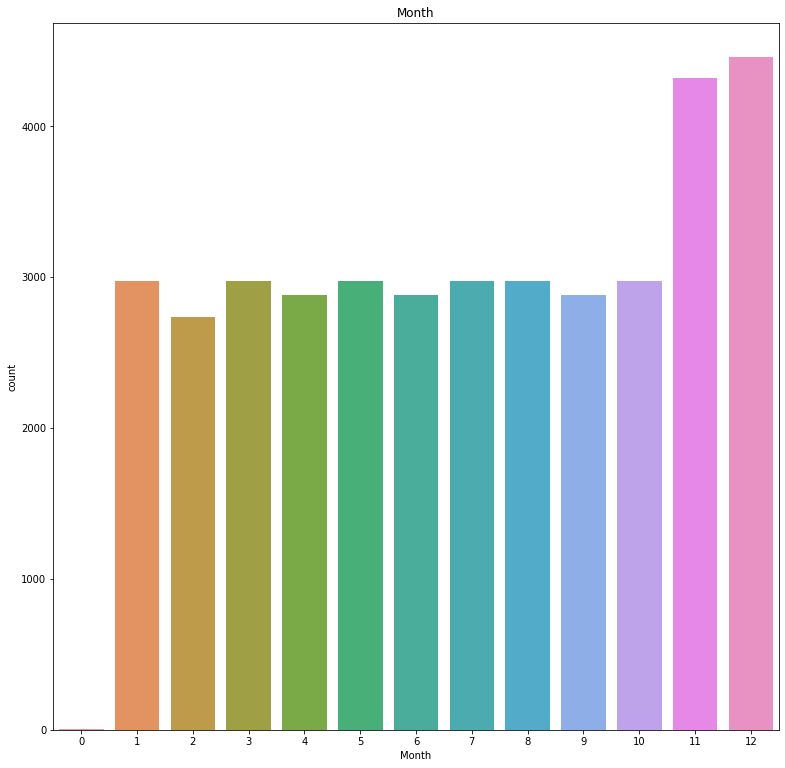
sns.countplot(x=i,data=df.loc[:,cat\_list]) plt.title(i)

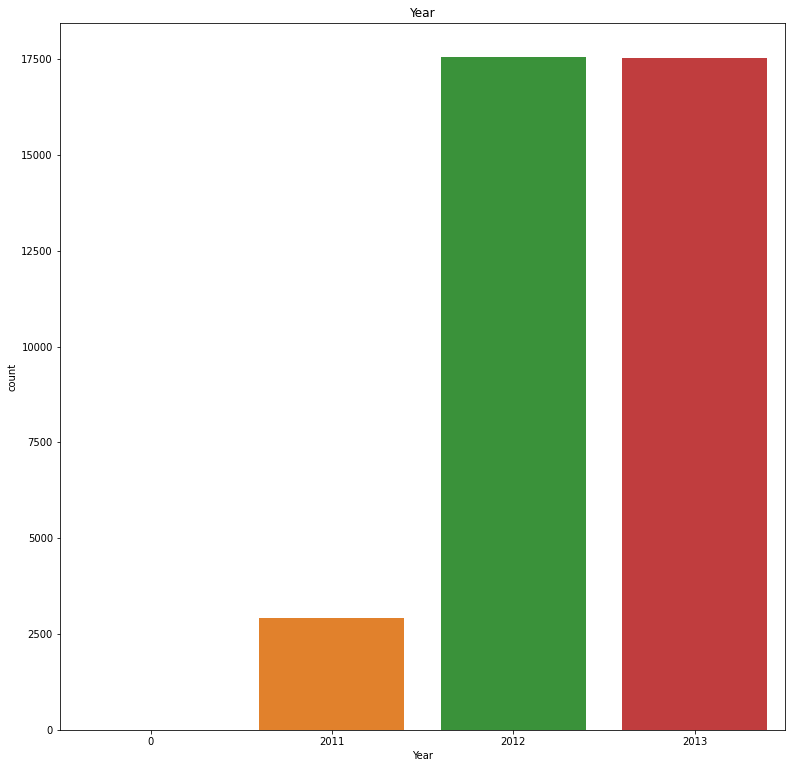


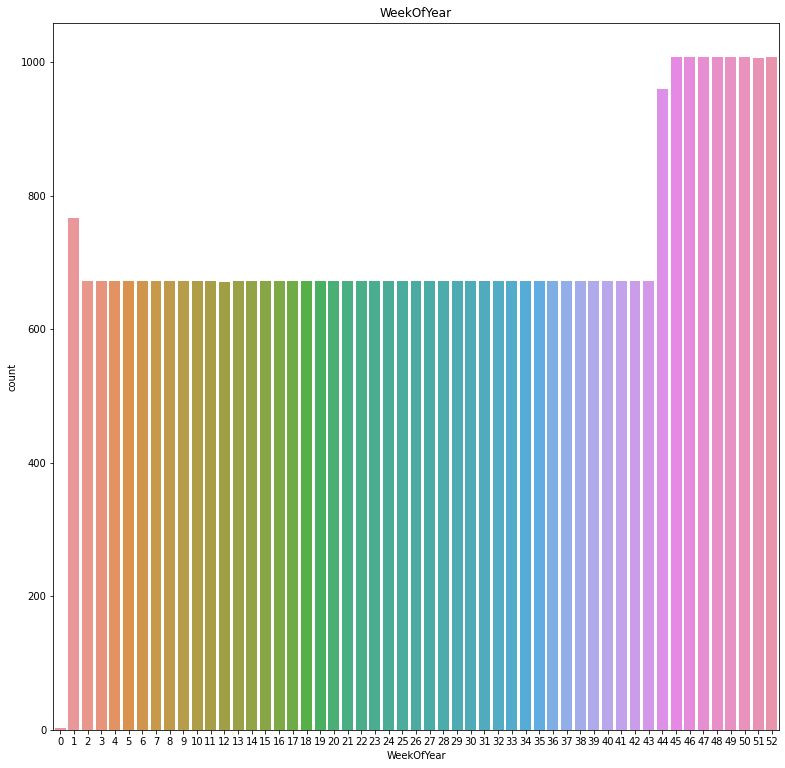
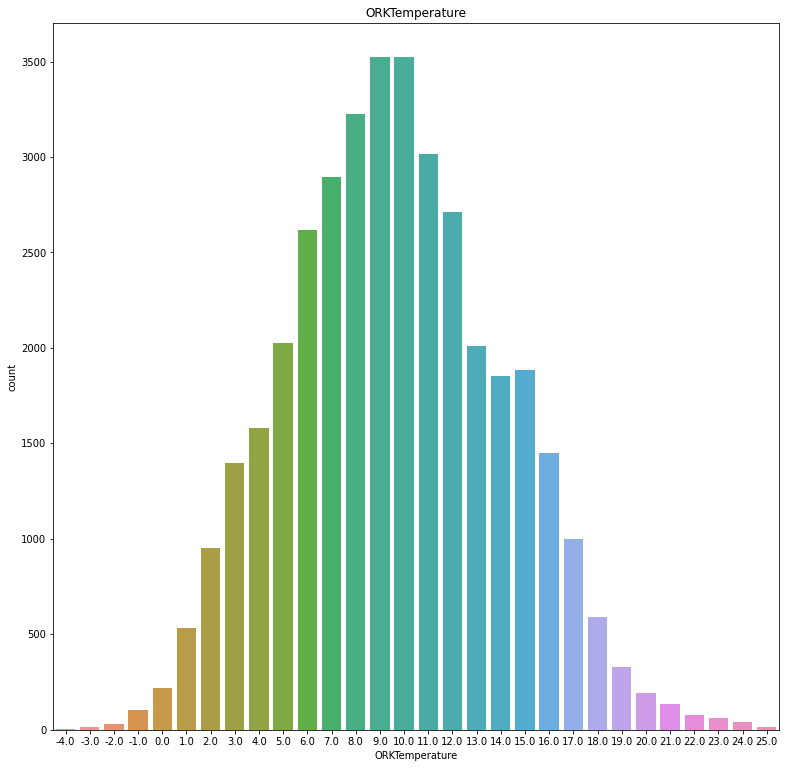






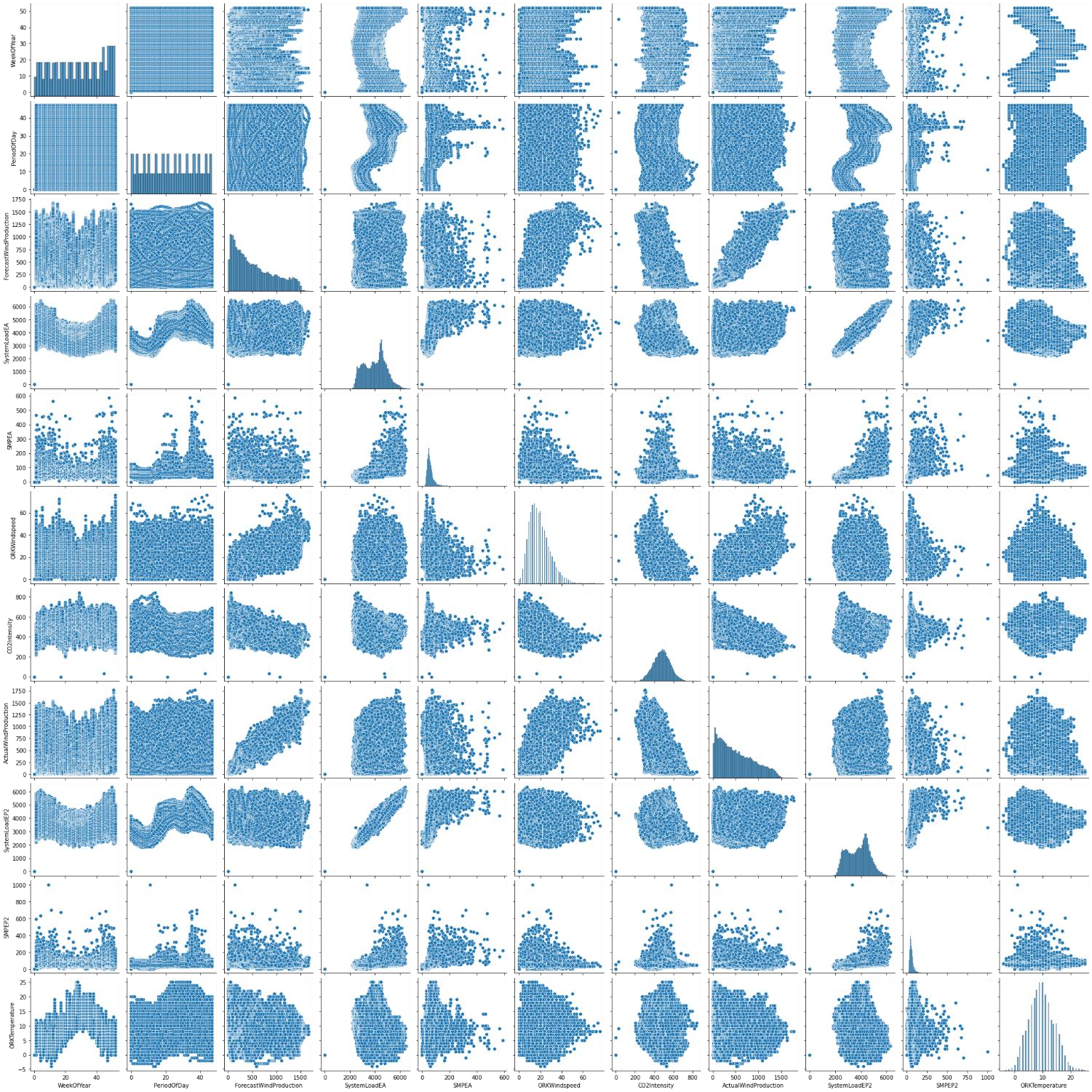






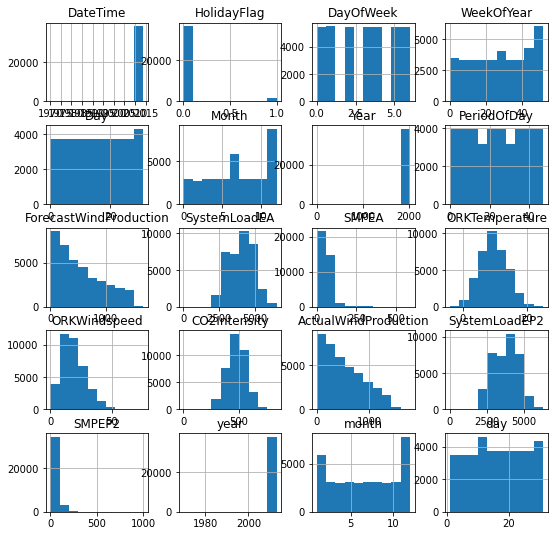
**In[2]:**

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',

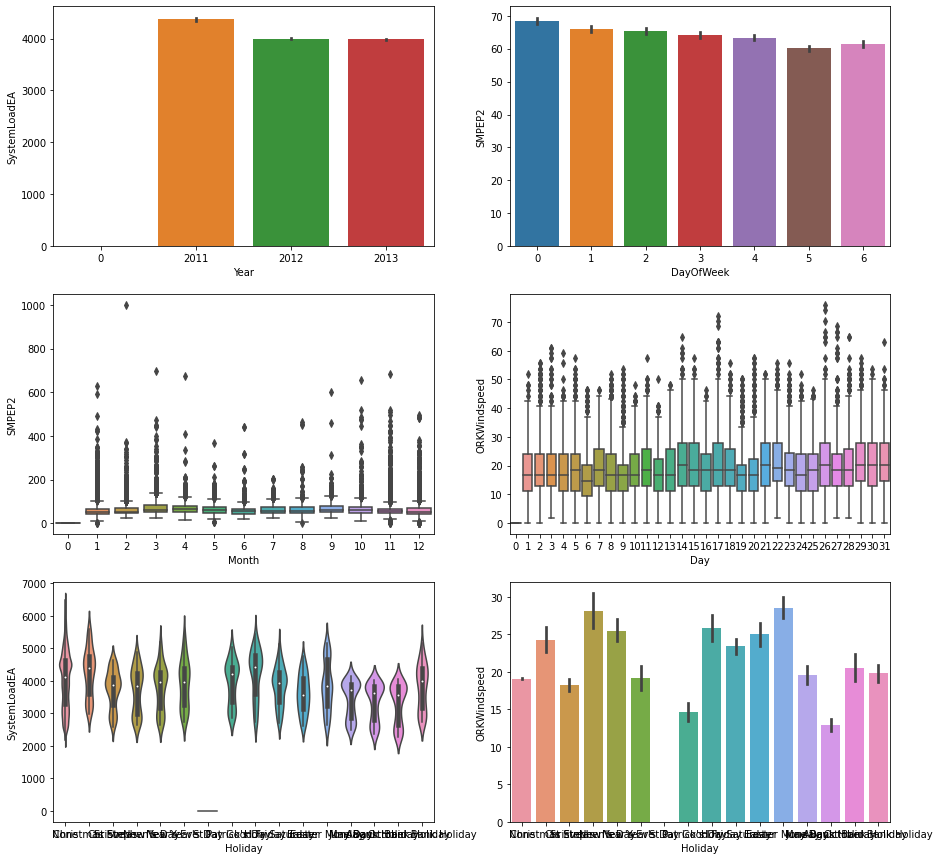
'SystemLoadEP2',

'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]:**

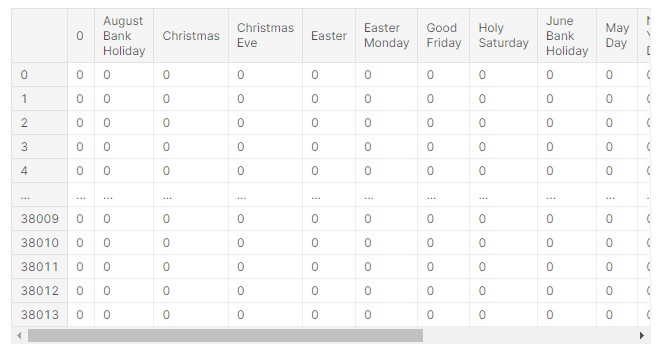


**Encoding:**

**In[1]:**

dms=pd.get\_dummies(df["Holiday"]) dms

**Out[1]:**

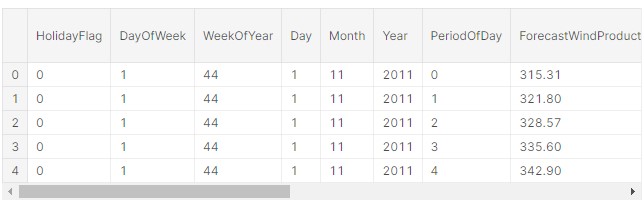


**In[2]:**

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

df.head()

**Out[2]:**



**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]:**



**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.*

1. **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.*

1. **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.*

1. **Weather Data:**

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

1. **Demand Data:**

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

1. **Market Data:**

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

1. **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.*

1. **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.*

1. **Special Events:**

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

1. **Price Differencing:**

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

1. **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.*

1. **Feature Selection:**

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

1. **Domain-Specific Features:**

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

1. **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.*

1. **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.*

1. **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.*

1. **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.*

1. **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.*

1. **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).*

1. **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.*

1. **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.*

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

1. **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.*

1. **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.*

1. **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.*

1. **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.*

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

1. **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.*

1. **R-squared (R²):** *This measures the proportion of variance in the target variable that's predictable from the features. A higher R-squared indicates a better fit.*

1. **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.*

1. **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.*

1. **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.*

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

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

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

## PROGRAM

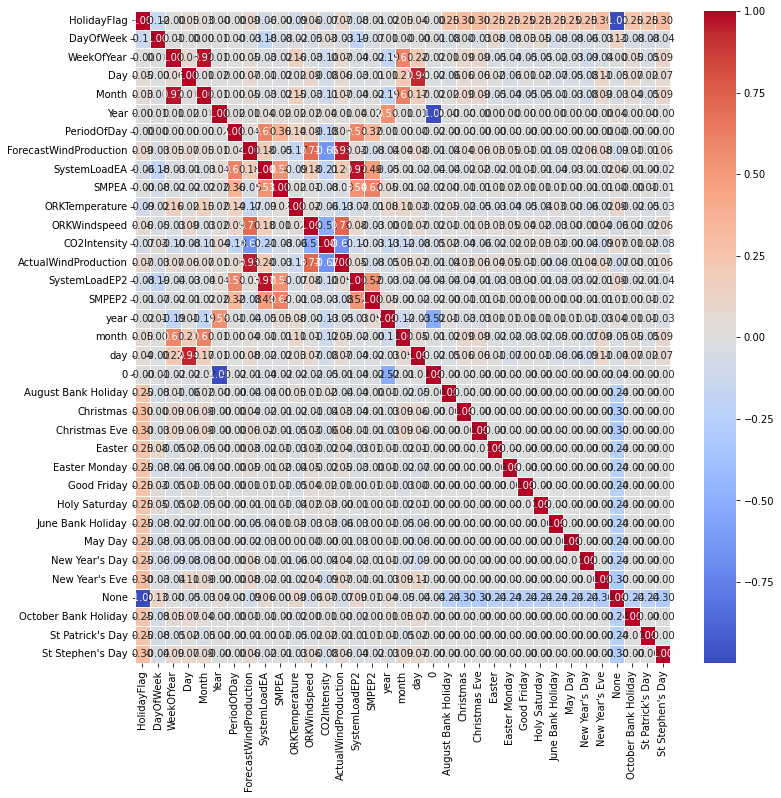
**Correlation Analysis:**

**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]:**





**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-package s (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

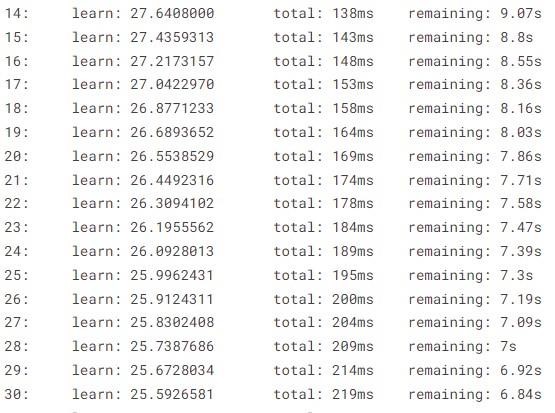
**In[9]:**

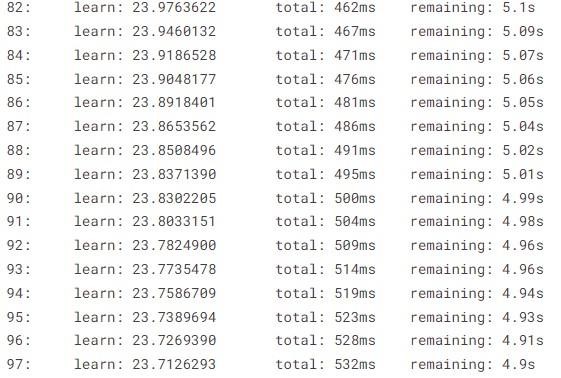
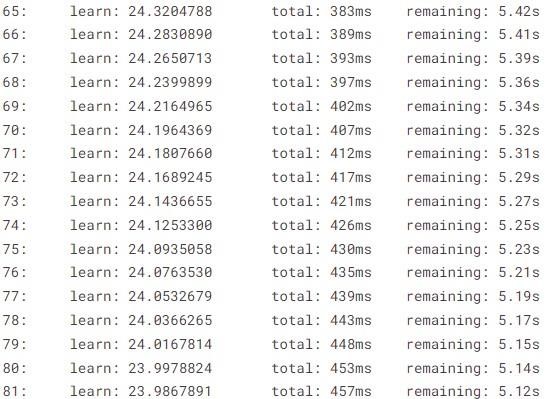
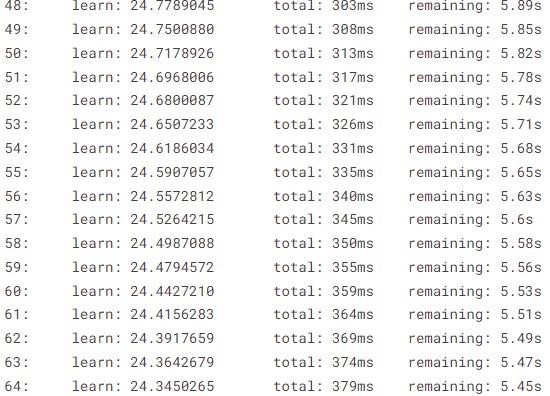
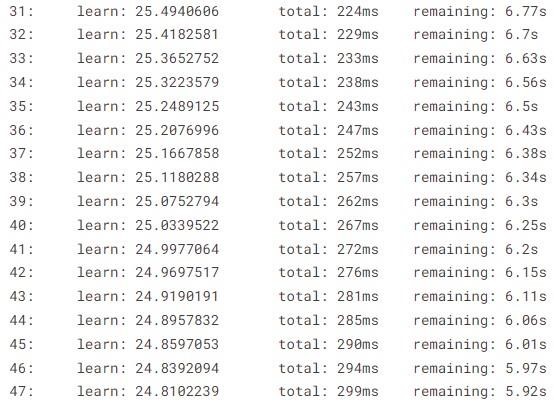
ridge=Ridge().fit(X\_train,y\_train) lasso=Lasso().fit(X\_train,y\_train) enet=ElasticNet().fit(X\_train,y\_train) knn=KNeighborsRegressor().fit(X\_train,y\_train) ada=AdaBoostRegressor().fit(X\_train,y\_train)

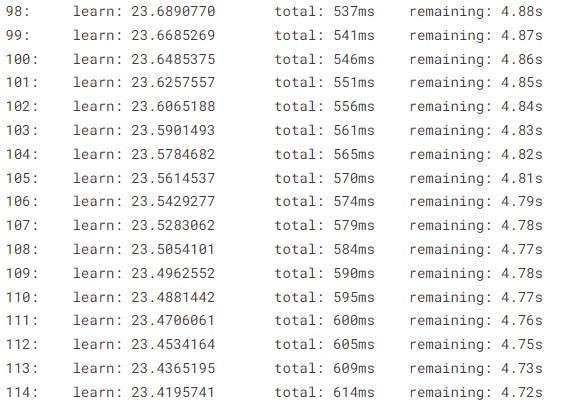
**In[10]:**

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]:**







**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

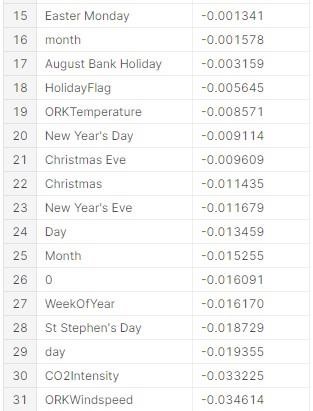
AdaBoostRegressor() Algorithm succed rate : 0.26365965295085403

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]:**



**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.*