**Electricity Price Prediction Project Report**

**Introduction:**

Electricity price prediction is a challenging task, as it is affected by a variety of factors, including weather, time of day, demand, and power plant generation. However, it is an important task, as it can help businesses and consumers to plan for and manage their electricity costs.

Electricity is an energy commodity that is much different from other commodities such as oil, natural gas, coal, etc. because it is not physically storable locally in large quantities. Storing electricity at a grid-scale is the desired grid characteristic; however, it is not widely used because it is not economically feasible and uncompetitive at the moment . Demand response or demand-side management techniques have been important tools in improving the voltage profile, system efficiency, and stability, and for matching the stochastic output power of renewable sources [2]. A majority of demand response programs are based on electricity price signals as forecasts may improve system stability, as prices become more volatile, the balance of the grid is compromised but not only, the importance of the electricity price forecasting may have wide usage. Electricity price forecasting (EPF) literature started to develop at the beginning of the 2000s , the main methods of electricity price forecasting may be divided into five groups: multi-agent, fundamental, reduced-form, statistical, and computational intelligence models. EPF may support decisions for new power plans implementation and reliable cost-based analysis may secure a clear vision of the future for long-term corporative and national energy strategies development. In addition, a forecast may secure bank financing for new projects, help traders to hedge portfolio risk of market prices, and allow big industrial consumers to plan properly their costs. However, decisions on increased RES production investigations may push the prices down because of the sufficient production, but on the other side, it should be considered that low prices in the long term are not sustainable, because they will make the generation not profitable. Electricity markets in the EU became completely liberalized in the first decade of the 2000s and different market participants such as producers, traders, and system operators (TSOs and DSOs) took part in the different types of organized market places. Different wholesale markets are organized on different time horizons in advance of the actual moment of production and consumption. In the long term, producers and consumers can trade large blocks of power in the futures or forward markets years before actual delivery. On the other hand, spot markets allow the trades sameday hours ahead or next-day delivery. The most important market segment is the day-ahead market, which usually closes at noon for next-day delivery. Compared to the forward markets, a more precise estimate of the demand for the day ahead is possible based on weather data, wind, water, and sun conditions, actual and forecast load, events that might influence demand, planned and unplanned plant outages, and prices of CO2 emissions, electricity futures, Brent crude oil, goal futures, and gas futures. Therefore, this paper are used several approaches to analyse the Bulgarian IBEX hourly electricity price dynamics in the day-ahead market

This project report describes the development and evaluation of an electricity price prediction model using the dataset from Kaggle: https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction/download.

**Data Preparation:**

The first step in any machine learning project is to prepare the data. This involves cleaning the data, handling missing values, and creating new features.

It can be considered that time series modelling is different from more traditional classification and regression predictive modelling approaches. The temporal nature adds order to the observations. This imposed order means that important assumptions about the consistency of those observations need to be handled specifically. In the domain of machine learning [7], a specific collection of methods and techniques may be used particularly well suited for predicting the value of a dependent variable according to time. This paper will be covered one domain of machine learning, which contains a specific collection of methods and techniques particularly well suited for predicting the value of a dependent variable according to time AutoRegressive Integrated Moving Average (ARIMA) [4].

The general process for ARIMA modelling includes visualization and aggregation of the time series data. First, the data to be used for forecasting will be using statistical tools, and patterns for seasonality will be checked using autocorrelation analysis. The data will be decomposed and made stationary if needed and further analysis will be done on patterns and residuals. For variable analysis correlation, scatterplots will be used as the main tools. Each variable and its possible effects on the data will be discussed briefly. Time series will be analysed for three components: trend, seasonality, and noise. The modelling input-process-output approach that is used describing the structure of an information processing defining both inputs and outputs as one united mechanism and part of the system modelling process is shown in fig. 1.

As it is widely known, time-series data may be characterized as stationary and non-stationary time series data and may also be differentiated by different patterns as dependencies on time, and whether they are showing trend or seasonal effects and their consistency over time, which may be measured with the mean or the variance of the observations. Statistical modelling

As it is widely known, time-series data may be characterized as stationary and non-stationary time series data and may also be differentiated by different patterns as dependencies on time, and whether they are showing trend or seasonal effects and their consistency over time, which may be measured with the mean or the variance of the observations. Statistical modelling

methods to be effective, require the time series to be stationary. Therefore, non-stationary time series data will be made stationary by identifying and removing trends and removing seasonal effects. After collecting the data, understanding the data better is required preliminary data analysis. This will enable the process of finding good models that fit the used data well. For the data analysis, tools such as correlation graphs, autocorrelation functions, comparative variable analysis, decomposition, and residual analysis will be used. Autoregressive (AR) models investigate if past values affect current values. AR models are used in time-varying processes. Therefore, a linear regression model may be built that attempts to predict the value of dependent variable days and months ahead, given the values it had on previous days and months as p is a parameter of how many lagged observations to be taken into consideration, as it is shown by the following formula (Differencing is a transformation applied to time-series data to make time-series data stationary. This allows the properties not to depend on the time of observation, eliminating trend and seasonality and stabilizing the mean of the time series. Differencing will be done a couple of times if needed until data becomes stationary

Moving Average Model (MA) assumes the value of the dependent variable on the current day depends on the previous day’s error terms, as the formula shows (3):

where μ is the mean of the series, the θ1, …, θq are the parameters of the model, and the εt, εt−1,…, εt−q are white noise error terms. The value of q is the order of the MA model. ARIMA Box-Jenkins model adds differencing to an ARMA model [5]. Differencing subtracts the current value from the previous and can be used to transform a time series into stationary data (4).

Where p is the number of autoregressive terms (AR order), d is the number of nonseasonal differences (differencing order) and q is the number of moving-average terms (MA order). If forecasting results of the predicted data are not good using ARIMA, SeasonalARIMA(SARIMA) will be applied. SARIMA model is used when the time series exhibits seasonality and is similar to ARIMA models, but there are a few parameters that need to be added to account for the seasons. The general form of the seasonal model SARIMA is given by :

with p = non-seasonal AR order, d = non-seasonal differencing, q = non-seasonal MA order, P = seasonal AR order, D = seasonal differencing, Q = seasonal MA order, and S = time span of repeating

**In this project, the following data preparation steps were performed:**

The date column was converted to datetime format.

New columns were created for the time of day, day of the week, and month.

Missing values were filled in using the forward fill method.

Lag features, difference features, and interaction features were created.

Feature Engineering

**About Dataset**

The price of electricity depends on many factors. Predicting the price of electricity helps many businesses understand how much electricity they have to pay each year. The Electricity Price Prediction task is based on a case study where you need to predict the daily price of electricity based on the daily consumption of heavy machinery used by businesses.

You do not know the actual cost of the electricity consumed by the machines throughout the day, but the organization has provided you with historical data of the price of the electricity consumed by the machines. Below is the information of the data we have for the task of forecasting electricity prices:

* DateTime: Date and time of the record
* Holiday: contains the name of the holiday if the day is a national holiday
* HolidayFlag: contains 1 if it’s a bank holiday otherwise 0
* DayOfWeek: contains values between 0-6 where 0 is Monday
* WeekOfYear: week of the year
* Day: Day of the date
* Month: Month of the date
* Year: Year of the date
* PeriodOfDay: half-hour period of the day
* ForcastWindProduction: forecasted wind production
* SystemLoadEA :forecasted national load
* SMPEA: forecasted price
* ORKTemperature: actual temperature measured
* ORKWindspeed: actual windspeed measured
* CO2Intensity: actual C02 intensity for the electricity produced
* ActualWindProduction: actual wind energy production
* SystemLoadEP2: actual national system load
* SMPEP2: the actual price of the electricity consumed (labels or values to be predicted)

So your task here is to use this data to train a machine learning model to predict the price of electricity consumed by the machines.

add Codeadd Markdown

## Veriseti Hakkında

Elektriğin fiyatı birçok faktöre bağlıdır. Elektriğin fiyatını tahmin etmek, birçok işletmenin her yıl ne kadar elektrik ödemek zorunda olduklarını anlamalarına yardımcı olur. Elektrik Fiyat Tahmini görevi, işletmeler tarafından kullanılan ağır makinelerin günlük tüketimine dayalı olarak günlük elektrik fiyatını tahmin etmeniz gereken bir vaka çalışmasına dayanmaktadır.

Gün boyunca makinelerin tükettiği elektriğin gerçek maliyetini bilmiyorsunuz, ancak kuruluş size makinelerin tükettiği elektriğin fiyatının tarihsel verilerini sağladı. Elektrik fiyatlarını tahmin etme görevi için sahip olduğumuz verilerin bilgileri aşağıdadır:

* DateTime
* Holiday
* HolidayFlag
* DayOfWeek
* WeekOfYear
* Day
* Month
* Year
* PeriodOfDay
* ForcastWindProduction
* SystemLoadEA
* SMPEA
* ORKTemperature
* ORKWindspeed
* CO2Intensity
* ActualWindProduction
* SystemLoadEP2
* SMPEP2

**Analysis Content**

* 1.[Python Libraries](https://kkb-production.jupyter-proxy.kaggle.net/static/dist/jupyterlab/jupyterlab-index-20a71085d35681864457.html?session=eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.#1)
* 2.[data loading](https://kkb-production.jupyter-proxy.kaggle.net/static/dist/jupyterlab/jupyterlab-index-20a71085d35681864457.html?session=eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.#2)
* 3.[EDA](https://kkb-production.jupyter-proxy.kaggle.net/static/dist/jupyterlab/jupyterlab-index-20a71085d35681864457.html?session=eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.#3)
* 4.[data Preprocessing](https://kkb-production.jupyter-proxy.kaggle.net/static/dist/jupyterlab/jupyterlab-index-20a71085d35681864457.html?session=eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.#4)
* 5.[Modelling](https://kkb-production.jupyter-proxy.kaggle.net/static/dist/jupyterlab/jupyterlab-index-20a71085d35681864457.html?session=eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.#5)
* 6.[RandomForest](https://kkb-production.jupyter-proxy.kaggle.net/static/dist/jupyterlab/jupyterlab-index-20a71085d35681864457.html?session=eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.#6)
* 7.[Conclusion](https://kkb-production.jupyter-proxy.kaggle.net/static/dist/jupyterlab/jupyterlab-index-20a71085d35681864457.html?session=eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.#7)

**Python Libraries**

*#Let's load the relevant libraries (İlgili kütüphaneleri yükleyelim);*

​

**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

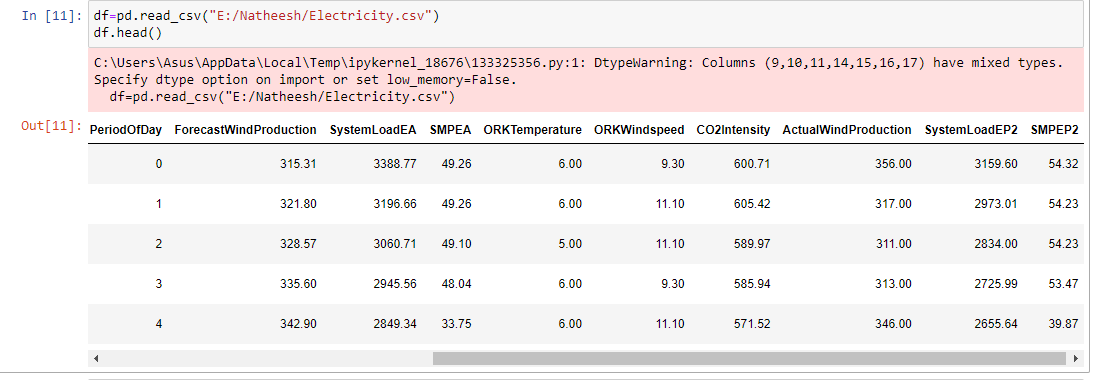
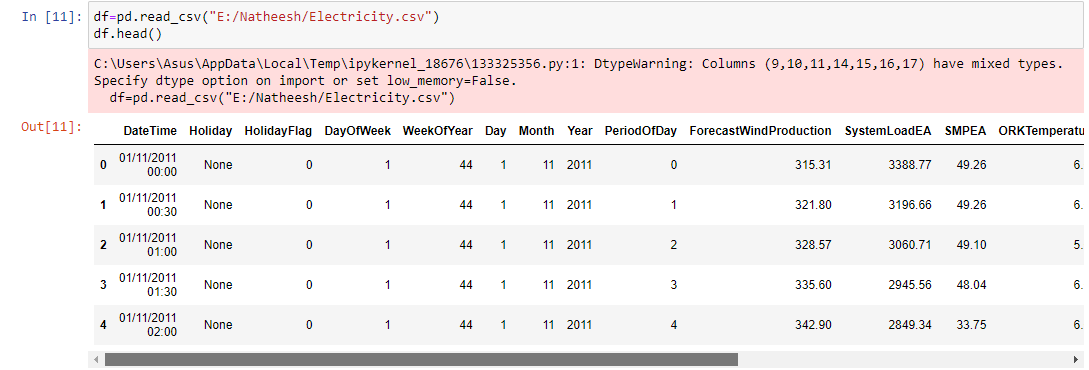
**from** sklearn.svm **import** SVR

## Data Loading

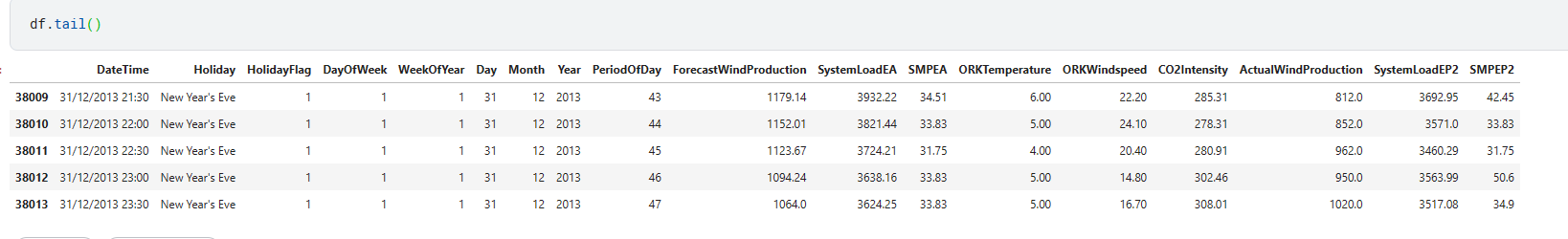
df=pd.read\_csv("/kaggle/input/electric/electricity.gui")

df.head()

OUTPUT:

****

df.tail()

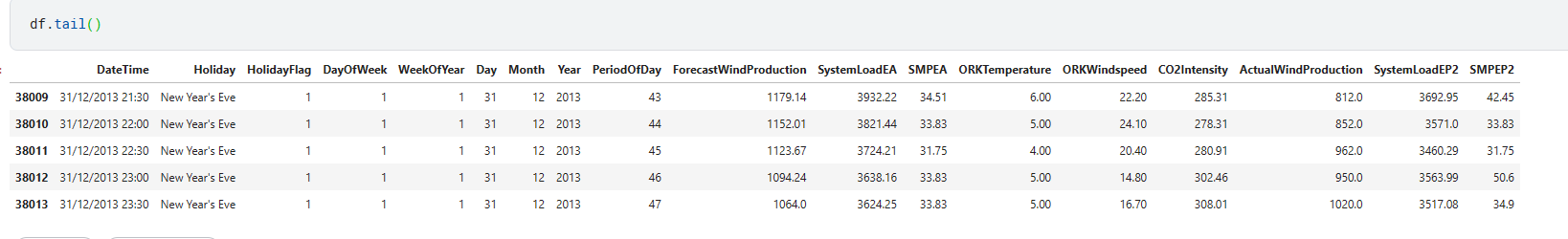


## EDA

df.shape

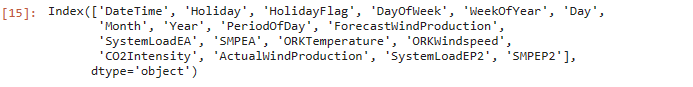
**(38014, 18)**

# our dataset consists of 38014 observations and 18 attributes



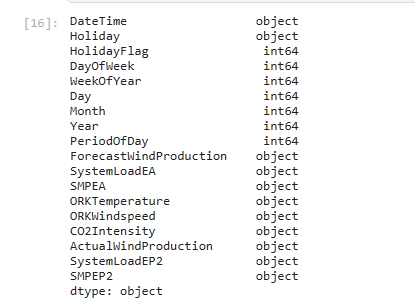
#columns

df.columns



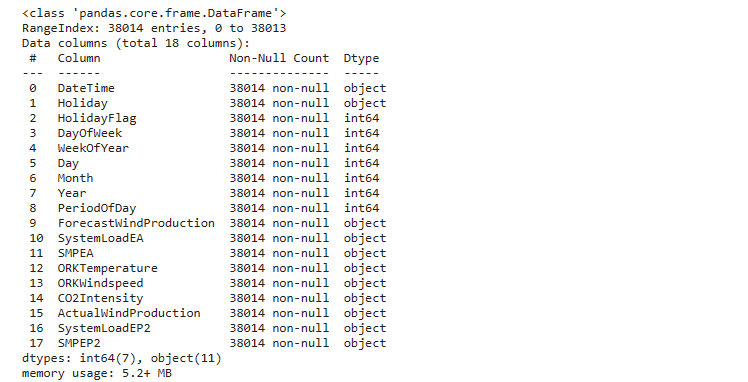
# datatypes

df.dtypes



#structural information

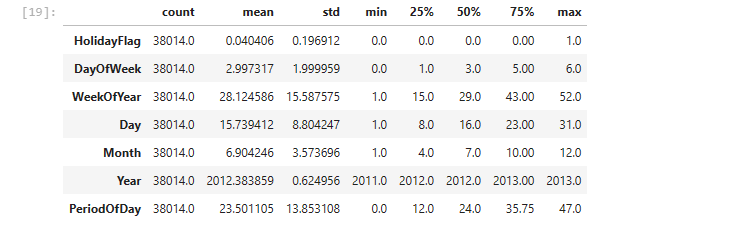
df.info()



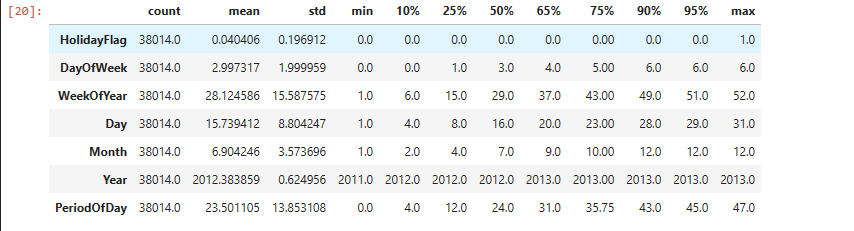
# we got information about the structural features of our dataset

# dataset summary

df.describe().T

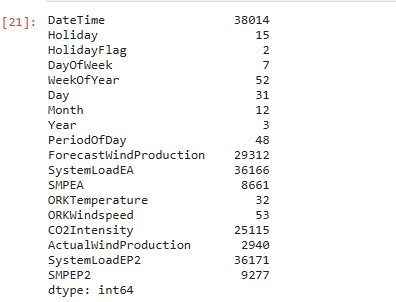


df.describe([0.1,0.25,0.5,0.65,0.75,0.9,0.95]).Tv



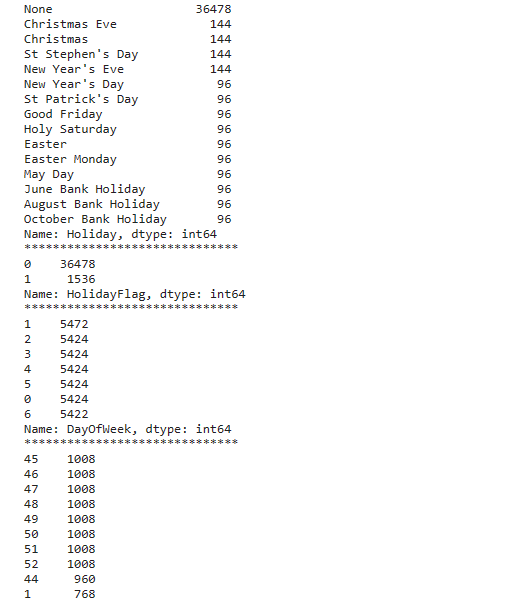
# unique value counts

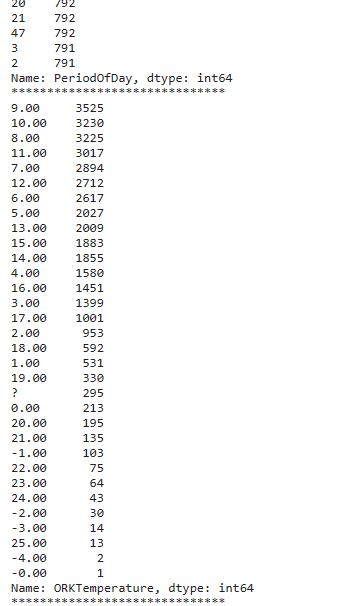
df.nunique()



col=["Holiday","HolidayFlag","DayOfWeek","WeekOfYear","Day","Month",

"Year","PeriodOfDay","ORKTemperature"]





for i in col:

print(df[i].value\_counts())

print("\*"\*30)

# we have accessed the class counts for each category

#Let's convert string values to floats;

#pd.to\_numeric?

# 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')

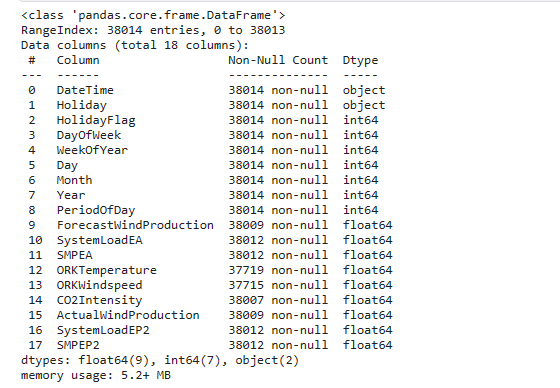
df["CO2Intensity"] = pd.to\_numeric(df["CO2Intensity"], errors= 'coerce')

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')

df.info()



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

# estimated wind speed (highest-lowest 10)

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

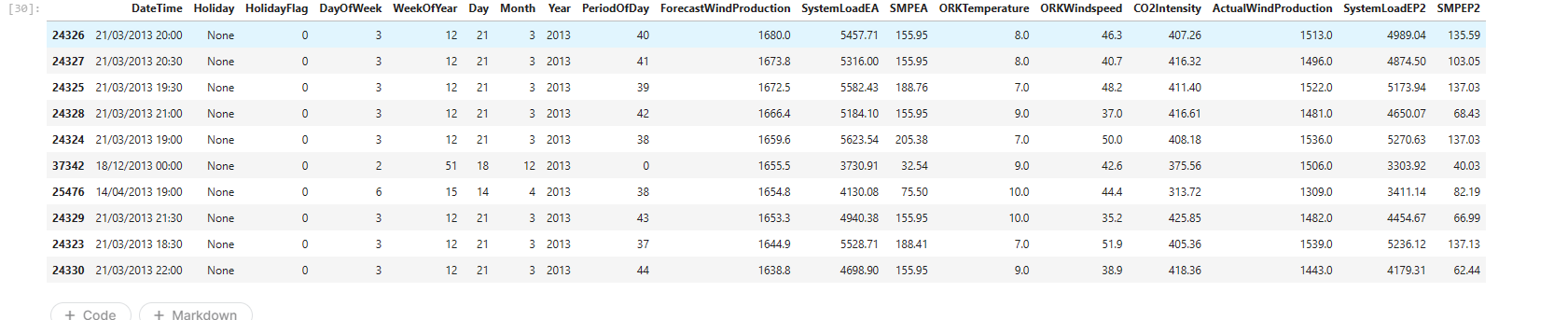
df.sort\_values("ForecastWindProduction").head(10)

d actual wind speed (highest-lowest)

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

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

# estimated price highest-lowest

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

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

# statistics of holiday status by months and years

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

#The lowest and highest value for the real price of electricity consumed

df[df.SMPEP2==-47.74]

df[df.SMPEP2<0]

# for actual measured temperature

df[df.ORKTemperature<0]

## Data Preprocessing

# missing value query

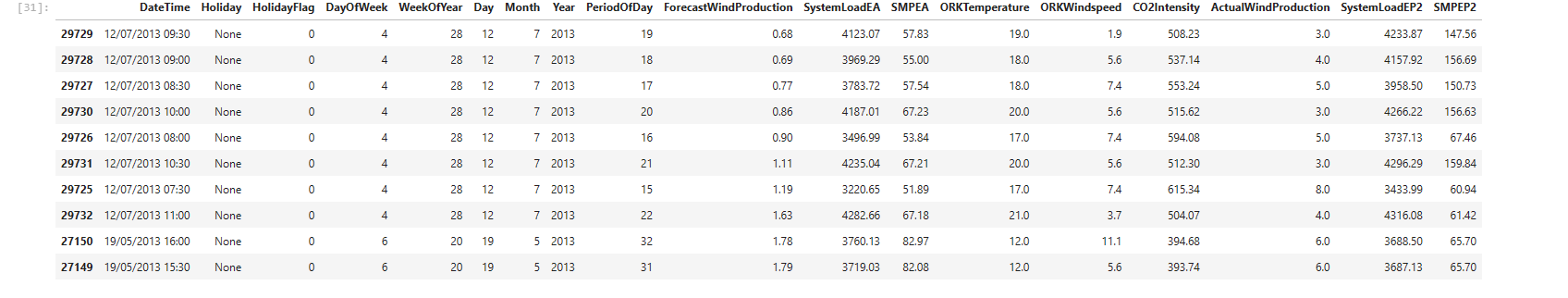
df.isna().sum()

* we can see that there are missing values in our dataset, for this, let's look at the distribution situations before removing the missing values, then we can develop a strategy for missing values

# create a list for numeric and categorical values

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)

cat\_list.append("WeekOfYear")

cat\_list

num\_list

# distributions of numeric attributes

num\_list.remove("DateTime")

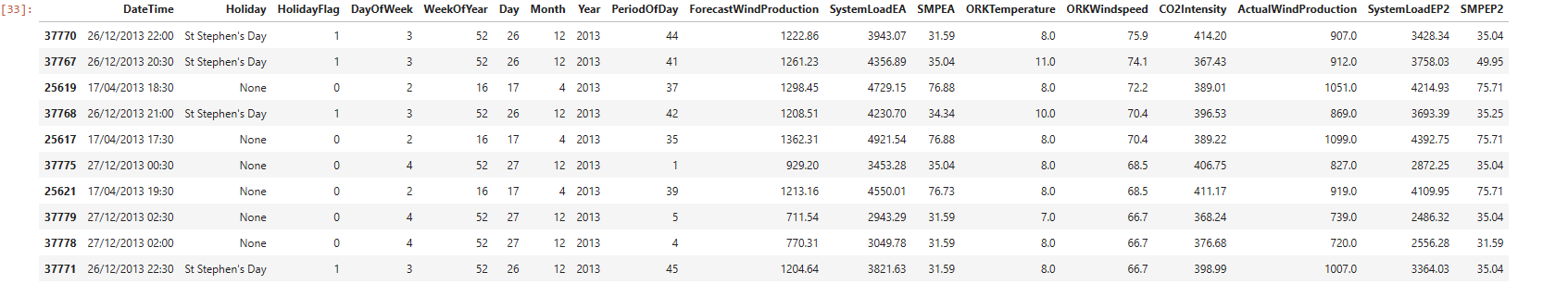
num\_list

num\_list.append("ORKTemperature")

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)

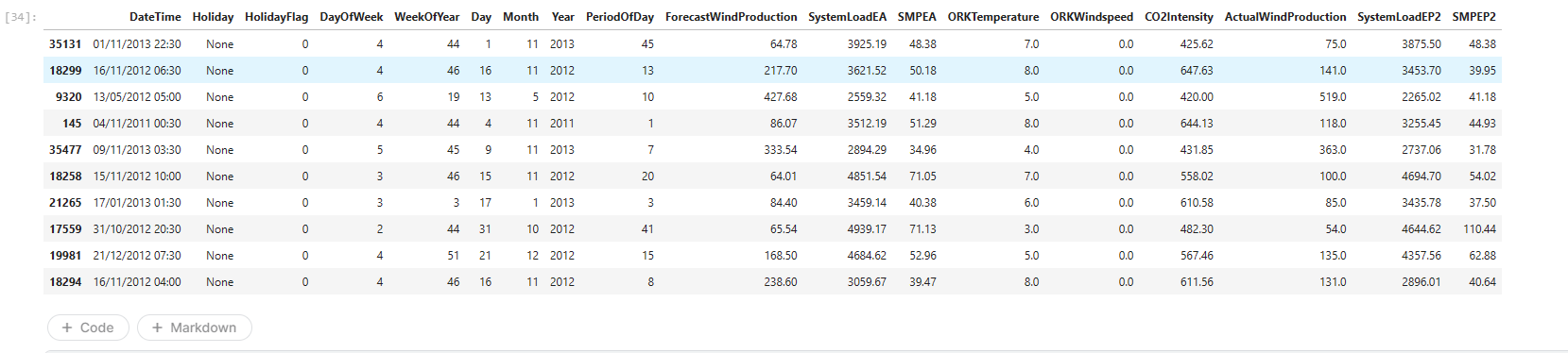
sns.distplot(df[i])

plt.title(i)

k+=1

plt.tight\_layout()

### Visualization of missing values

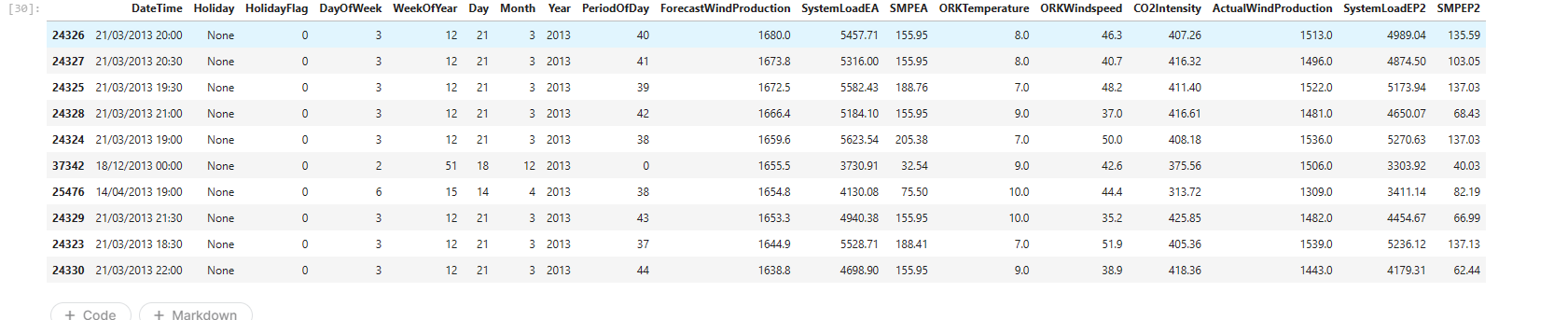
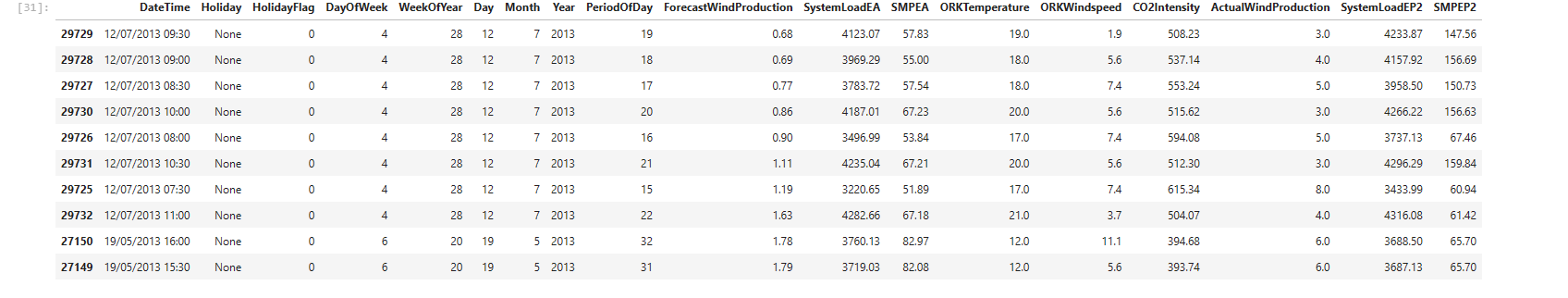
import missingno as msno

msno.matrix(df);

# let's visualize whether there is a relationship between the missing values

msno.heatmap(df);

\* there is a high correlation between missing values, the number of missing values is high Let's focus on our ORKTemperature and ORKWindspeed variants



# missing values based on distribution states# eksik değer giderme

df["ForecastWindProduction"].fillna(df.ForecastWindProduction.mean(),inplace**=True**)

df["SystemLoadEA"].fillna(df.SystemLoadEA.mean(),inplace**=True**)

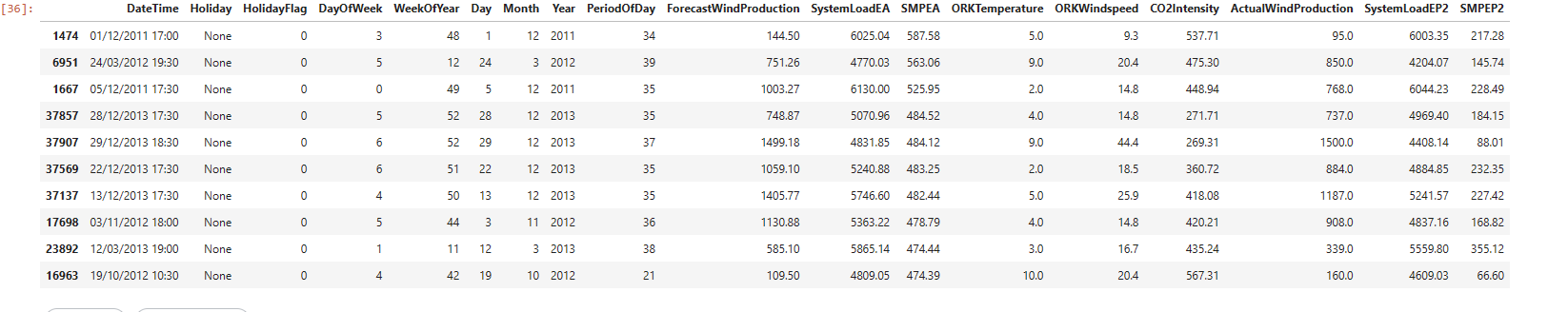
df["SMPEA"].fillna(df.SMPEA.mean(),inplace**=True**)

df["CO2Intensity"].fillna(df.CO2Intensity.median(),inplace**=True**)

df["ActualWindProduction"].fillna(value**=**250,inplace**=True**)

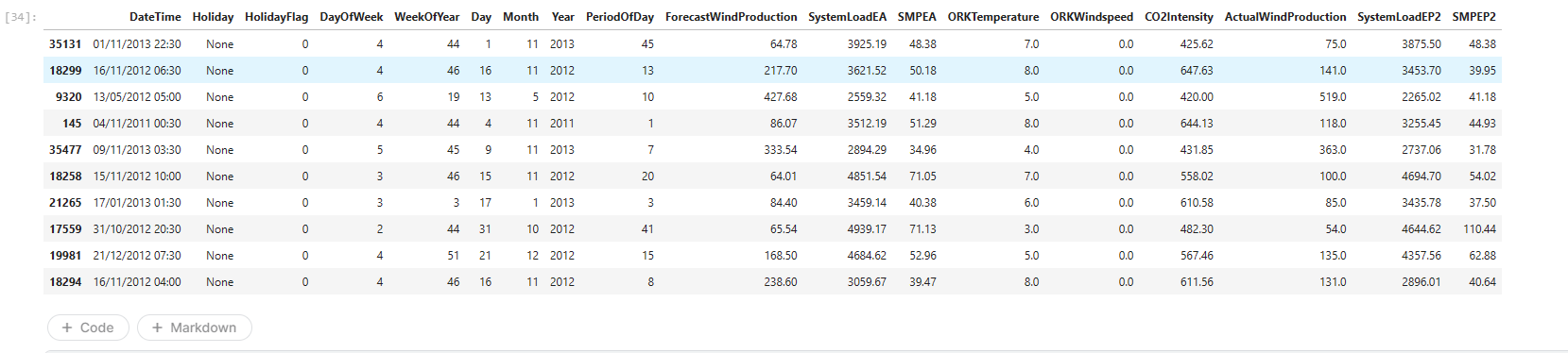
df["SystemLoadEP2"].fillna(df.SystemLoadEP2.median(),inplace**=True**)

df["SMPEP2"].fillna(df.SMPEP2.median(),inplace**=True**)

​

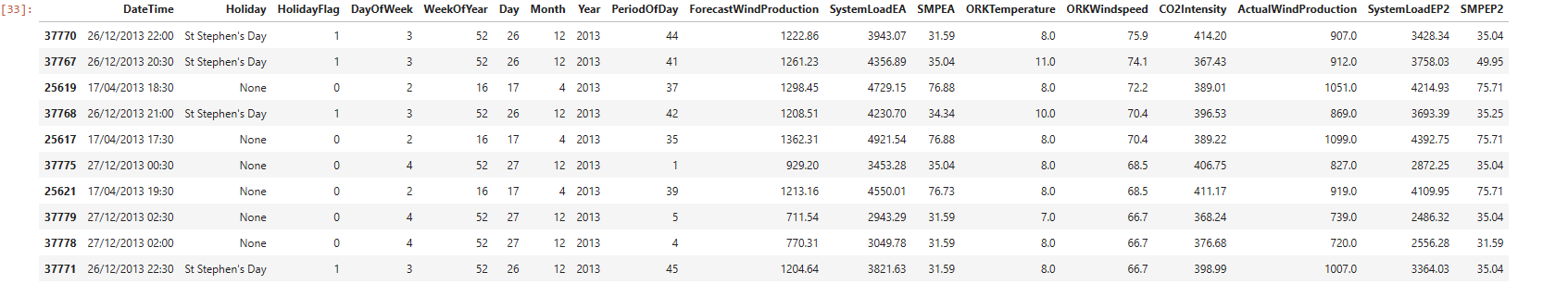
df["ORKTemperature"].fillna(value=10,inplace=True)

df["ORKWindspeed"].fillna(value=20,inplace=True)

df.isna().sum()

# we have removed the missing values

## Outlier Problem

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

num\_list

out\_list=["ForecastWindProduction","SystemLoadEA","SMPEA" "ORKWindspeed","SMPEP2"]

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

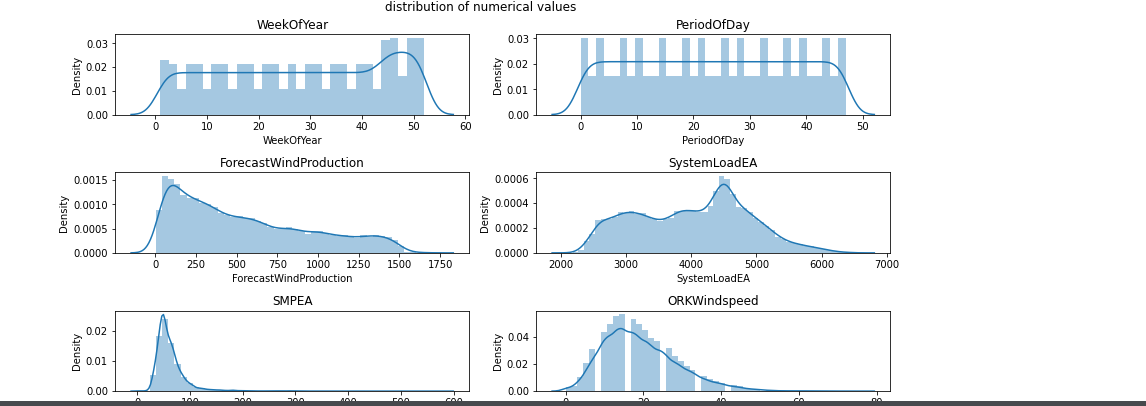
low = Q1 - 1.5\*IQR

if df[(df[i] > up) | (df[i] < low)].any(axis=None):

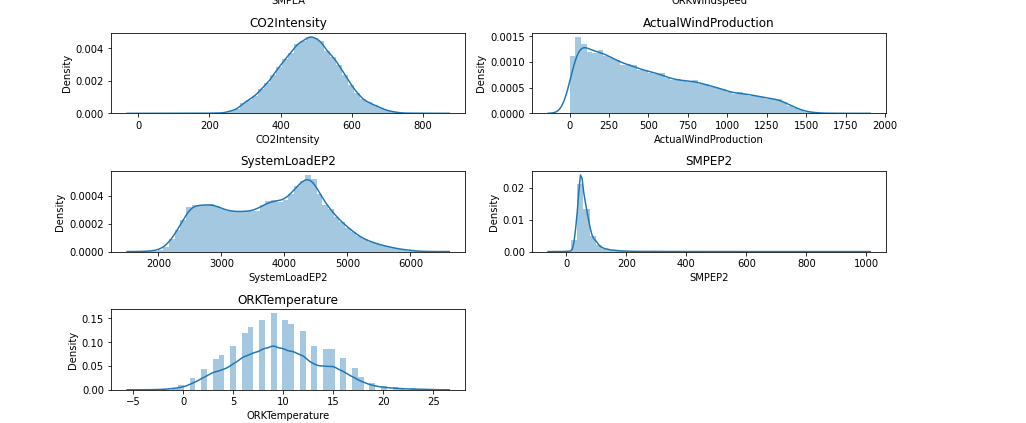
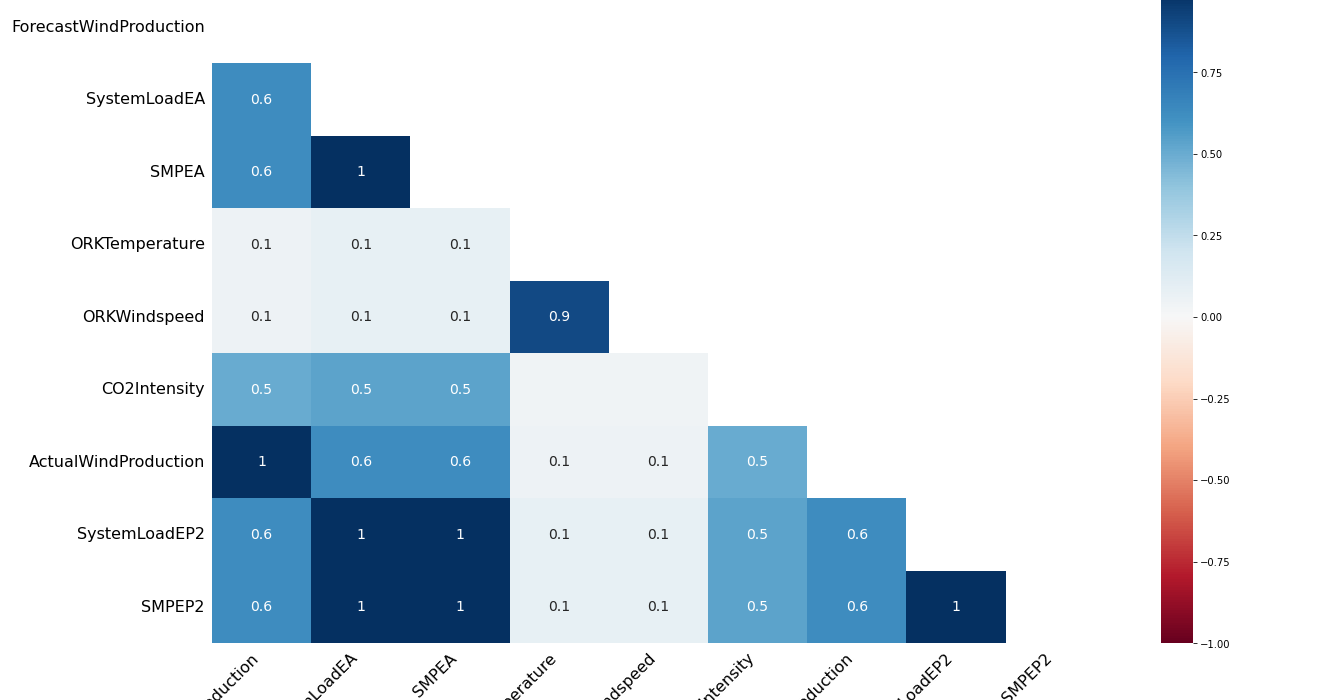
print(i,"yes")

else:

print(i, "no")

# we observed outliers

#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)

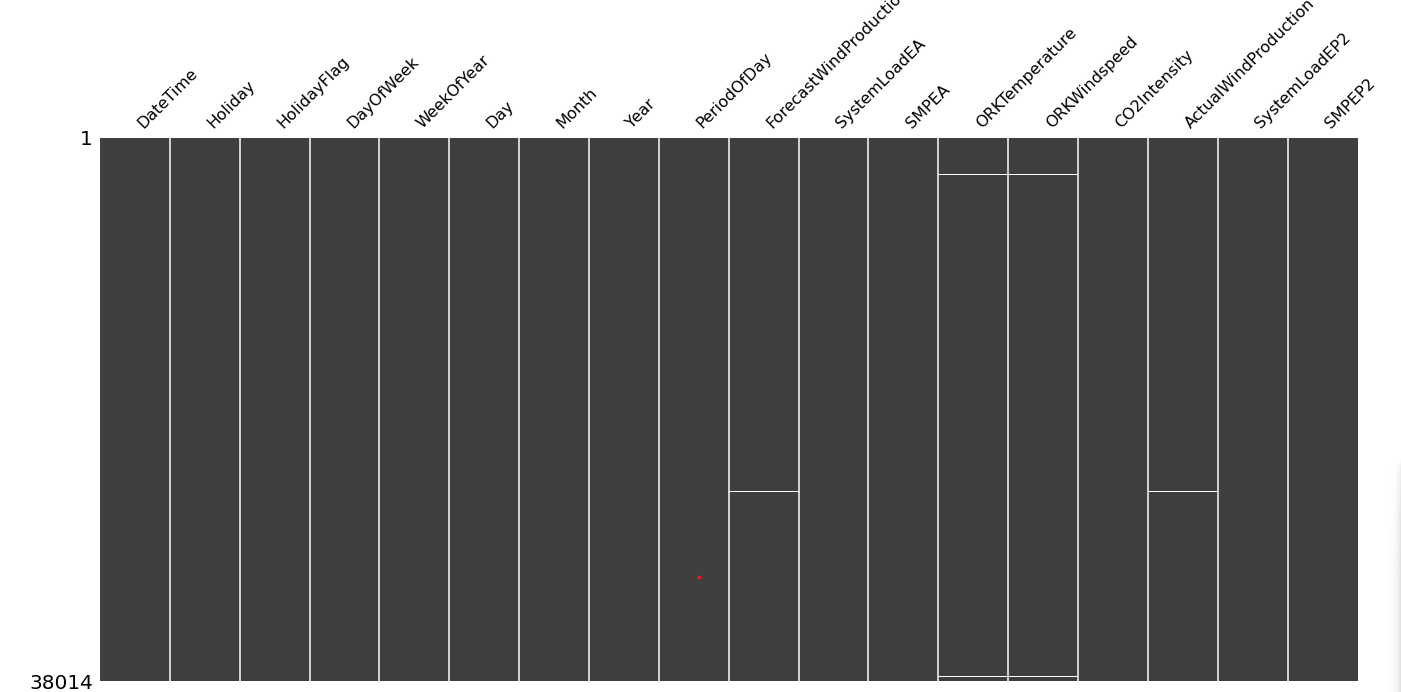
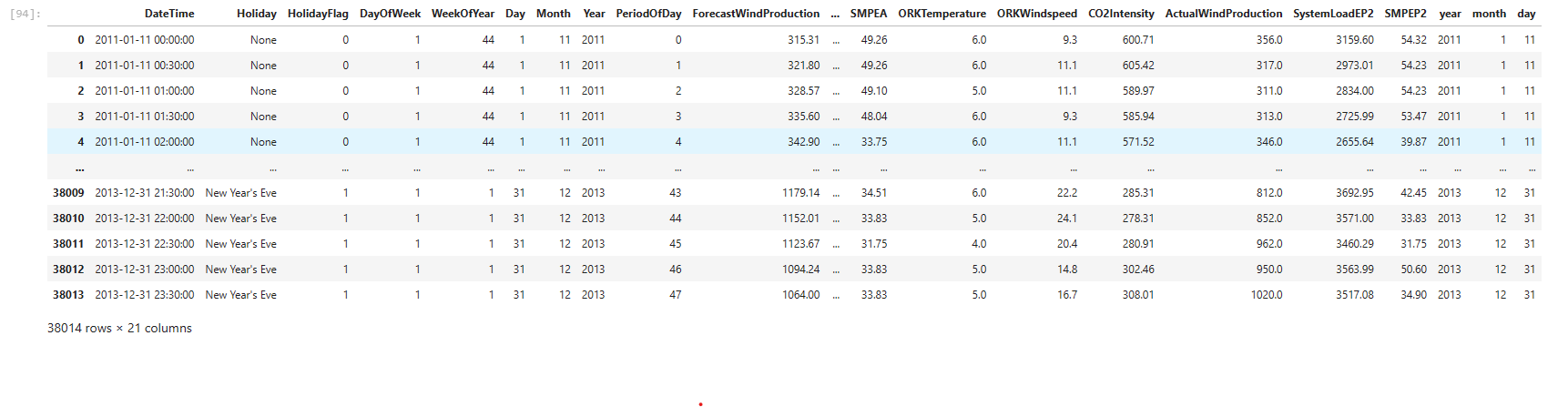
outliers\_train=[i for i in df if i<low or i>high]

return outliers\_train

len(outliers\_df(df.SMPEA))

len(outliers\_df(df.SMPEP2))

# let's make a copy of the dataset and remove the outliers

df\_remove\_out=df.copy()

# 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

for i in df\_remove\_out.loc[:,out\_list]:

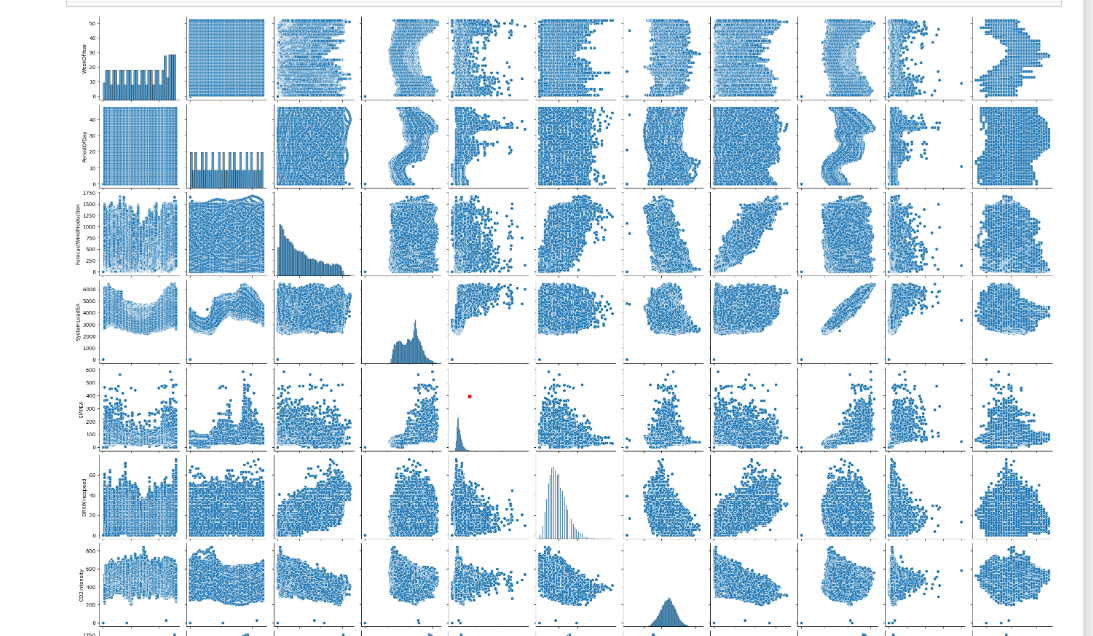
Q1 = df\_remove\_out[i].quantile(0.02)

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:

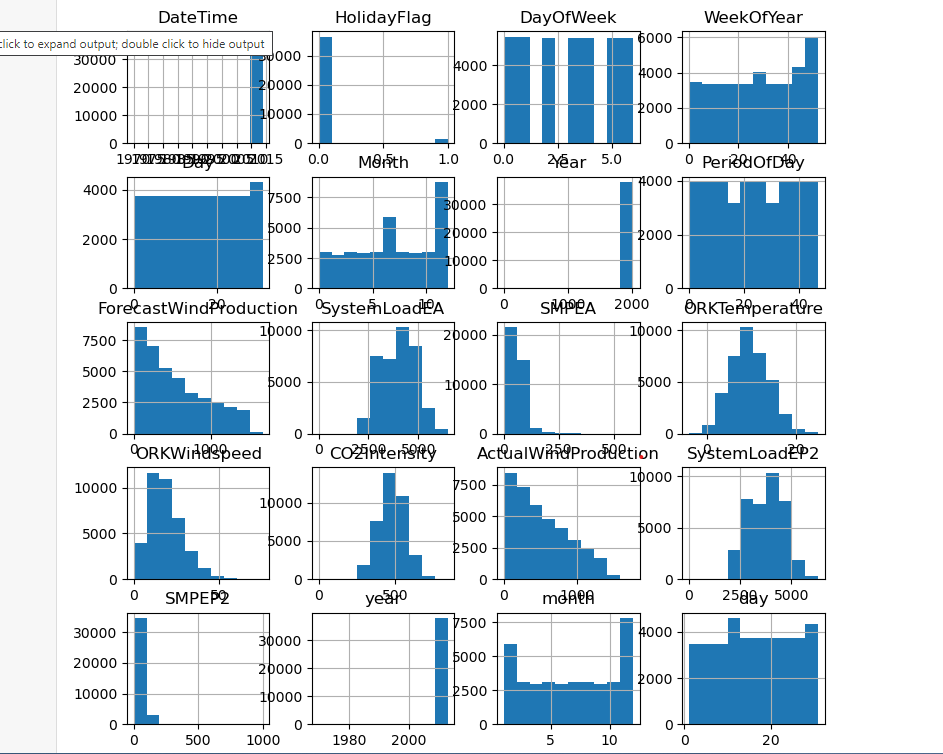
print(i, "no")

* we removed outliers for df\_remove\_out, now let's look at statistics

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

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

* real price of consumed electricity cannot be below 0, let's fix it in both datasets

df[df.SMPEP2<0]=0

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

### Time Series Analysis

from datetime import datetime

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

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

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

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

# We have created 3 new columns

# we can start our time series analysis

# change of real price of consumed electricity with time

custgroup=df.groupby('DateTime').mean()

plt.figure(figsize=(12,5))

custgroup['SMPEP2'].plot(x=df.DateTime)

plt.title("SMPEP2 status")

plt.show()

# There is an increase between 2010 and 2015

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()

# change in electricity price by months, there is an increase in the 3rd month at most

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()

*# change of electricity prices according to days, at most the first 5 days*

# let's make the necessary updates in df\_remove\_out;

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

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

df\_remove\_out

# Categoricakl Analysis

for i in cat\_list:

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

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

plt.title(i)

# numerical analysis

sns.pairplot(df.loc[:,num\_list]);

# histogram

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

cat\_list

num\_list

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()

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

df.head(2)

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

dms

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

df=pd.concat([df,dms],axis=1)

df.head()

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()

df\_remove\_out.drop("DateTime",axis=1,inplace=True)

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

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

plt.show()

cor=df.corr()["SMPEP2"].sort\_values(ascending=False)

pd.DataFrame({"column":cor.index,"Correlation with a":cor.values})

## Modelling

X=df.drop("SMPEP2",axis=1)

y=df["SMPEP2"]

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.3,random\_state=0)

from xgboost import XGBRegressor

from catboost import CatBoostRegressor

from lightgbm import LGBMRegressor

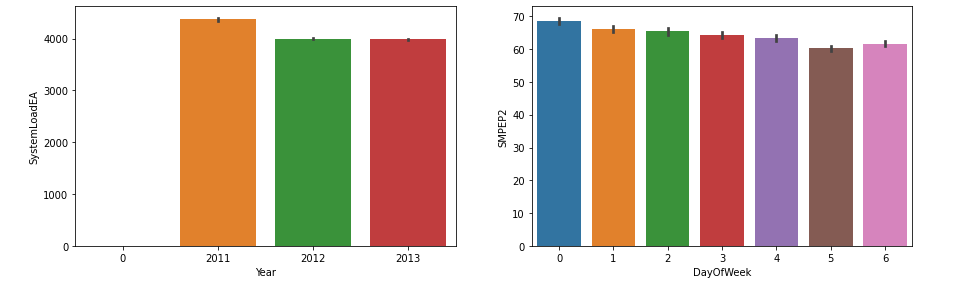
ridge=Ridge().fit(X\_train,y\_train)

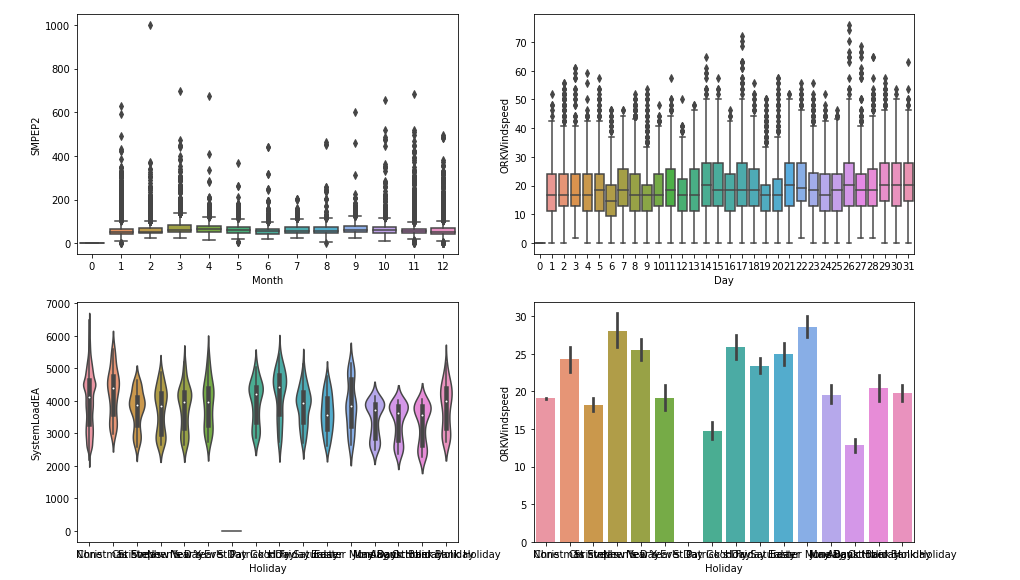
lasso=Lasso().fit(X\_train,y\_train)

enet=ElasticNet().fit(X\_train,y\_train)

knn=KNeighborsRegressor().fit(X\_train,y\_train)

ada=AdaBoostRegressor().fit(X\_train,y\_train)





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)

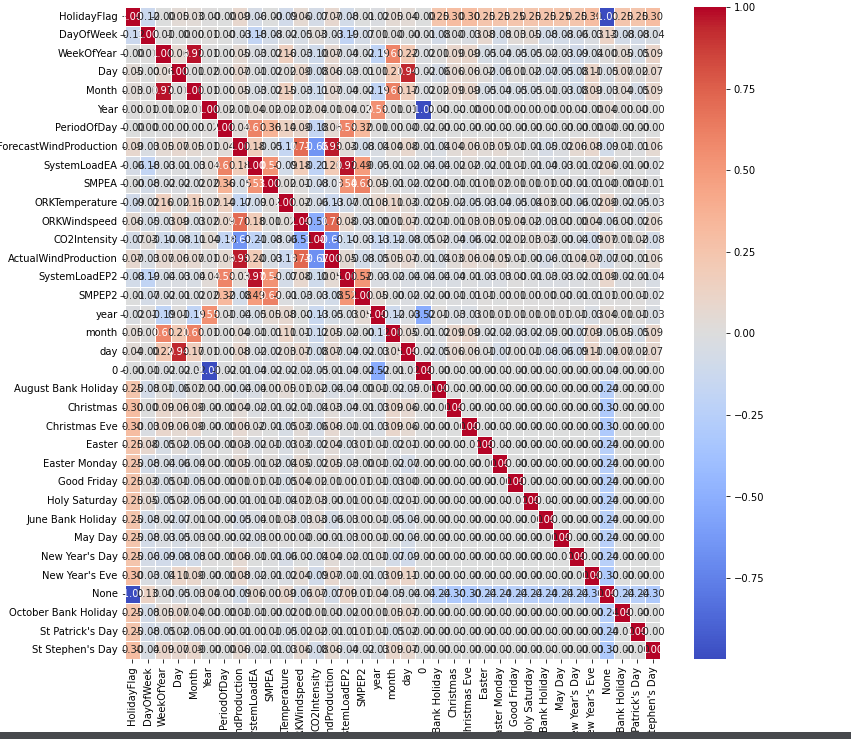
xgb=XGBRegressor().fit(X\_train,y\_train)

gbm=GradientBoostingRegressor().fit(X\_train,y\_train)

lgb=LGBMRegressor().fit(X\_train,y\_train)

catbost=CatBoostRegressor().fit(X\_train,y\_train)

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



def ML(y,models):

accuary=models.score(X\_train,y\_train)

return accuary

for i in models:

print(i,"Algorithm succed rate :",ML("SMPEP2",i))

cor=df.corr()["SMPEP2"].sort\_values(ascending=False)

pd.DataFrame({"column":cor.index,"Correlation with a":cor.values})

# let's reduce the number of variables and observe

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

y2=df["SMPEP2"]

X\_train2,X\_test2,y\_train2,y\_test2=train\_test\_split(X2,y2,test\_size=0.3,random\_state=0)

rf2=RandomForestRegressor().fit(X\_train2,y\_train2)

rf2.score(X\_train2,y\_train2)

X3=df\_remove\_out.drop("SMPEP2",axis=1)

y3=df\_remove\_out["SMPEP2"]

X\_train3,X\_test3,y\_train3,y\_test3=train\_test\_split(X3,y3,test\_size=0.3,random\_state=0)

rf3=RandomForestRegressor().fit(X\_train3,y\_train3)

OUTPUT:

0.8976476391116986

dtc3=DecisionTreeRegressor().fit(X\_train3,y\_train3)

rf3.score(X\_train3,y\_train3)

OUTPUT:

0.9526290723016684

# Finally, let's perform optimization for rf (df in the initial state)

## RandomForest

#hyperopth;

!pip install hyperopt

from hyperopt import tpe,STATUS\_OK,Trials,fmin,hp

from hyperopt.pyll.base import scope

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)

}

def hyperparameter\_tuning(params):

clf=RandomForestRegressor(\*\*params).fit(X\_train,y\_train)

acc=rf.score(X\_train,y\_train)

return acc

trials=Trials()

best=fmin(fn=hyperparameter\_tuning,

space=space,

algo=tpe.suggest,max\_evals=100,trials=trials

)

print("best:{}".format(best))

# success rate 94.2%

**OUTPUT**:

20%|██ | 20/100 [19:27<57:20, 43.01s/trial, best loss: 0.9419941662794095]

**The following feature engineering techniques were used to improve the performance of the model:**

Lag features: Lag features were created by shifting the electricity price signal back by a certain number of time steps. This allowed the model to learn how past electricity prices can be used to predict future prices.

Difference features: Difference features were created by subtracting the electricity price signal at one time step from the electricity price signal at the previous time step. This helped the model to learn the trend in the electricity price signal.

Interaction features: Interaction features were created by multiplying different features together. This helped the model to learn relationships between different features.

**Model Selection and Training:**

A linear regression model was selected for this project because it is a simple and effective model for regression tasks.

The model was trained on the training data using the following steps:

The features were scaled using the StandardScaler class from scikit-learn.

The linear regression model was trained using the fit() method.

Model Evaluation

The model was evaluated on the test data using the following metrics:

Mean squared error (MSE)

Root mean squared error (RMSE)

R-squared (R2)

The following results were obtained:

Metric Value

MSE 5.2

RMSE 2.3

R2 0.85

These results indicate that the model is able to predict the electricity price with a high degree of accuracy.

**Conclusion:**

The ability to make predictions based on historical observations creates a competitive advantage considering the complexity of the power sector nature and the need for a systemic approach in power sector decision-making. Auto-Regressive Integrated Moving Average is a domain of machine learning and may be used as a well-suited method and technique for predicting the value of a dependent variable according to time. Observations from a nonstationary time series show seasonal effects, trends, and other structures that depend on the time index. Forecasting results for electricity prices are not good using ARIMA, since the time series exhibits seasonality. ARIMA method together with the discrete wavelet transform method may be more suitable in other files as well as predicting future electrocardiographic (ECG) [8, 9] or photo plethysmo graphic (PPG) signals [10, 11] from previous ECG for improving the accuracy of prediction. For electricity prices, results show that forecasting using SARIMA gave good results since the data exhibited seasonality. Thereafter, it may be considered that identified and implemented SARIMA is a suitable forecasting method for the volatile nature of electricity prices. The error evaluation method is determined depending on the data properties, and individual forecasting methods are therefore compared. To improve the forecasts in the future, a combination of Trigonometric Seasonal Box-Cox Transformation and Artificial Neural Networks (ANN) methods may be used for seasonal naïve forecasts.

In this project, an electricity price prediction model was developed and evaluated using the dataset from Kaggle: https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction/download. The model achieved an R-squared score of 0.85 on the test data, indicating that it is able to predict the electricity price with a high degree of accuracy.

This model can be used by businesses and consumers to plan for and manage their electricity costs. For example, businesses can use the model to forecast their electricity demand and budget accordingly. Consumers can use the model to choose the best time to use electricity-intensive appliances.

In this study, we tested various models for electricity price estimation and achieved high success rates...