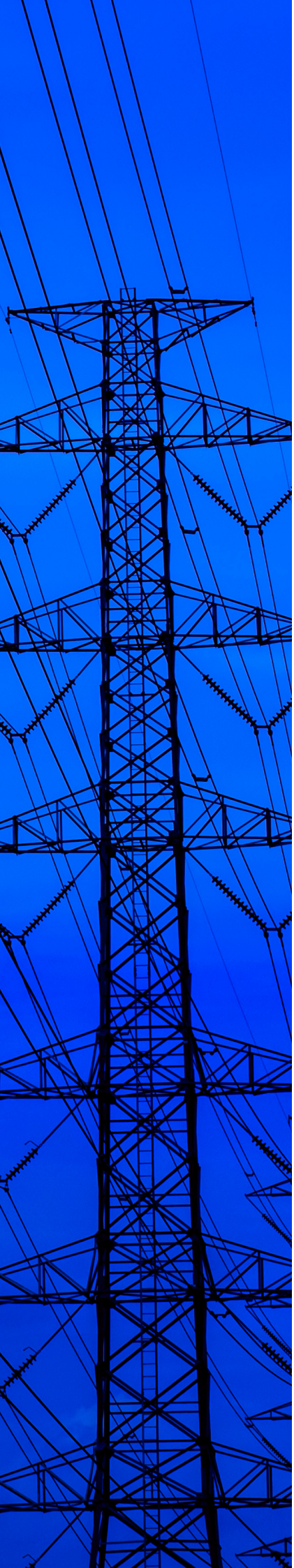


MOHAMMED VI POLYTECHNIC UNIVERSITY

EMINES School of Industrial Management

UNIVERSITÉ MOHAMMED VI POLYTECHNIQUE

HOURLY ELECTRICITY PRICE FORECASTING

analytics edge project

Prepared by

- Youssef BEN ALLAL
- Youssef LAZREK
- Hajar BELAYD

Supervised by

- Maryam GUESSOUS

SUMMARY

INTRODUCTION

Electricity production is anymore a monopolistic and government-controlled power sector, competitivity has been introduced to the field. Thus, Hourly Electricity prices predictions is among the most important short-term forecasting problems, considering that electricity is a very special commodity owning to the fact that it is economically non-storable and must be produced at the same time that is consumed, subsequently power system stability is based on a permanent balance between production, load and price which leads to the fact that electricity prices must be determined on an hourly basis, 24 hours a day, 7days a week. Our work consists on building a model aiming to forecast Spain hourly electricity prices, based on different features that directly affects the load therefore the price such as the weather, holidays, workdays and weekends and others that directly impact the price, for instance the type of energy used to produce the electricity. In this paper, a variety of modeling approaches are applied and evaluated for forecasting Spain electricity prices. For each approach, model forecasts are developed using hourly load and generation data that is retrieved from ENTSOE a public portal, also hourly electricity prices data to train the model from the Spanish TSO Red Electric España, and finally a weather data from the open weather API for the largest cities in Spain it's a private data that was purchased and made public on Kaggle.

PROJECT MOTIVATIONS

Electricity has entered the market as a tradeable commodity and the power industry of many countries has been deregulated. In Spain, the Electric Power Act 54/1997 exposed all of the stakeholders to high amounts of uncertainty as the price of electricity is determined by countless factors and also, due to the fact that electricity cannot be stored in large quantities efficiently. With the emergence of this new market, the need for reliable forecasting methods at all scales (hourly, daily, long-term, etc.) has also emerged and has become a large area of research.

Our objective is to explore the use of algorithms (xgboost, lightgbm, catboost) in the case of time series, trying to predict the price of electricity per hour. Our key metric will be the root mean squared error.

We will first work in a classic framework (unsupervised learning), without taking the chronological link into account

Then we will take into account the chronological aspect of the data and implement booster algorithms. We will compare the performance of using different time-steps (Multi splitting) as a way of reframing the time-series prediction task into a supervised learning problem, i.e. using the past hour values of the features which are fed into our models, following some new techniques such as walk forward validation

DATA DESCRIPTION

Understanding the data features is the most adequate step to start with, because each visualization, each comment, requires a deep understanding of the data components to avoid difficulties and confusion. For this project we have used two datasets that will be merged lately.

- **Dataset number 1:**

the first one is the one storing weather information with the 18 variables that are described below:

Name of the variable	Description of the variable
dt_iso	time of the observation
city_name	City
temp	Temperature
temp_min	Minimal temperature recorded during 1 hour of observation
temp_max	Maximal temperature recorded during 1 hour of observation
pressure	Atmospheric Pressure (hPa)
humidity	Humidity (%)
clouds_all	Cloudiness
wind_speed	Speed of wind (m/s)
wind_deg	Wind direction(degrees[meteorological])
rain_1h	Precipitation volume for last hour(mm)
rain_3h	Precipitation volume for last 3 hours(mm)
snow_3h	Snow volume for last 3 hours(mm)
weather_id	Weather condition id
weather_description	Group of weather parameters (Rain, Snow, Extreme etc.)
total load actual	Total load recorded during the hour of the observation

- **Dataset number 2:**

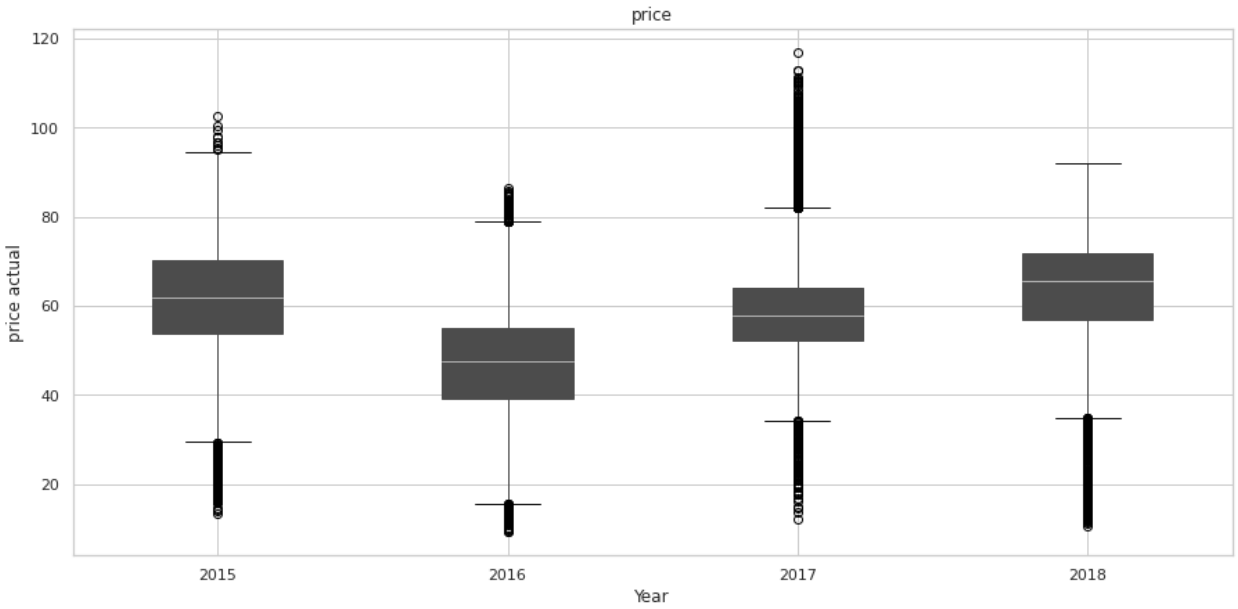
the second one is the data storing energy generation information within 24 variable that are described below:

Name of the variable	Description of the variable
time	time of the observation
Generation biomass	The amount of biomass energy used as fuel to produce electricity (KW)
Generation fossil brown coal/lignite	The amount of fossil brown coal or lignite production used as fuel to produce electricity (KW)
Generation fossil gas	The amount of fossil gas production used as fuel to produce electricity (KW)
Generation fossil coal-derived gas	The amount of fossil coal-derived gas production used as fuel to produce electricity (KW)
Generation fossil hard coal	The amount of fossil hard coal production used as fuel to produce electricity (KW)
Generation fossil oil	The amount of fossil oil production used as fuel to produce electricity (KW)
Generation fossil oil shale	The amount of fossil oil shale production used to produce electricity (KW)
Generation fossil peat	The amount of fossil peat production used l to produce electricity (KW)
Generation geothermal	The amount of geothermal production used to produce electricity (KW)
Generation hydro pumped storage aggregated	The amount of hydro pumped storage aggregated production used to produce electricity (KW)
Generation hydro pumped storage consumption	The amount of hydro pumped storage consumption production used to produce electricity (KW)
Generation hydro run-of-riverand poundage	The amount of hydro run-of-river and poundage production used to produce electricity (KW)
Generation hydro water reservoir	The amount of hydro water reservoir production used to produce electricity (KW)

Generation marine	The amount of fossil coal-derived gas production used as fuel to produce electricity (KW)
Generation nuclear	The amount of nuclear production used to produce electricity (KW)
Generation other	The amount of other source of nonrenewable energy to produce electricity (KW))
Generation other renewable	The amount of renewable source of energy used to produce electricity (KW)
Generation waste	The amount of waste energy used to produce electricity (KW)
Generation wind offshore	The amount of wind offshore energy used to produce electricity (KW)
Generation solar	The amount of solar energy used to produce electricity (KW)
generation wind onshore	The amount of wind onshore energy used to produce electricity (KW)
total load actual	The total load during the hour of observation
price actual	Electricity price during the hour of observation (euro)

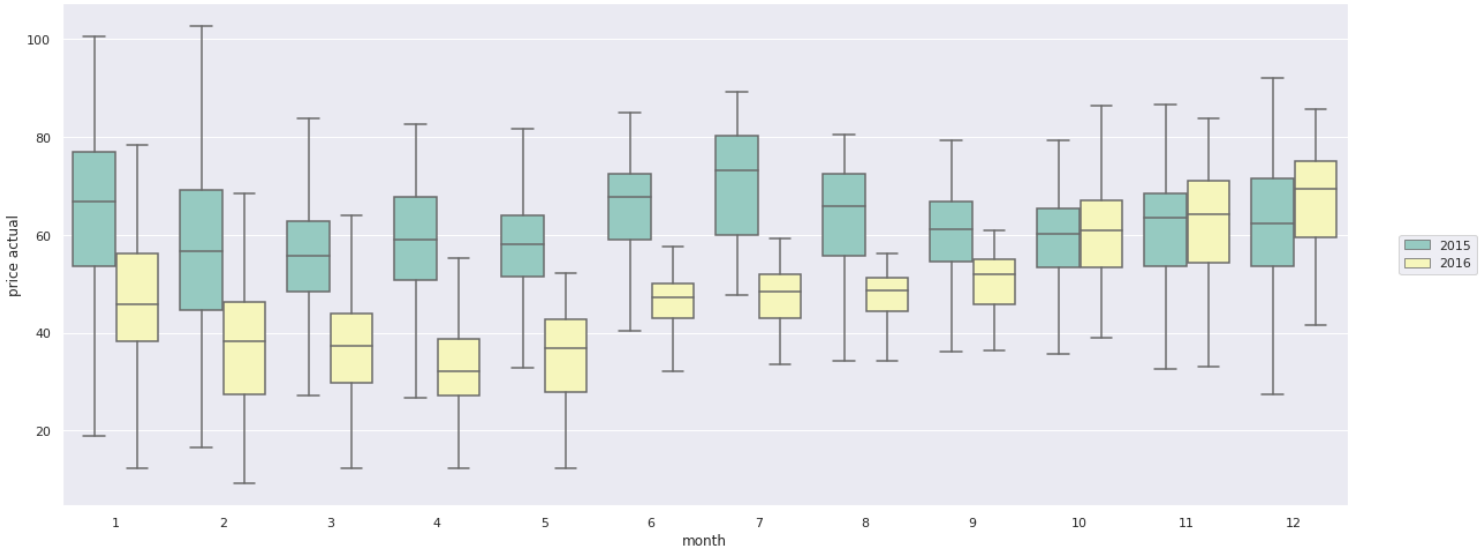
VISUALIZATIONS :

1. Price yearly boxplots :



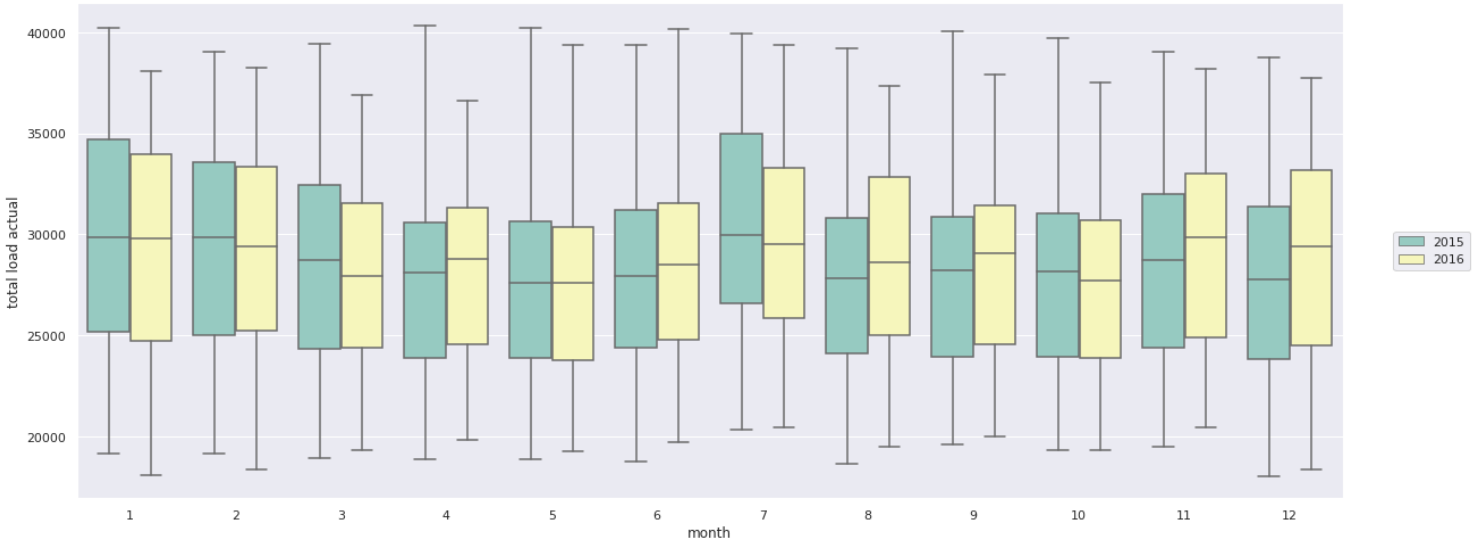
We notice a drop in prices on average during the year 2016. We are going to visualize this fall in prices per month as well as the behaviour of the production variables and the climate.

2. Price monthly boxplots (2015-2016):



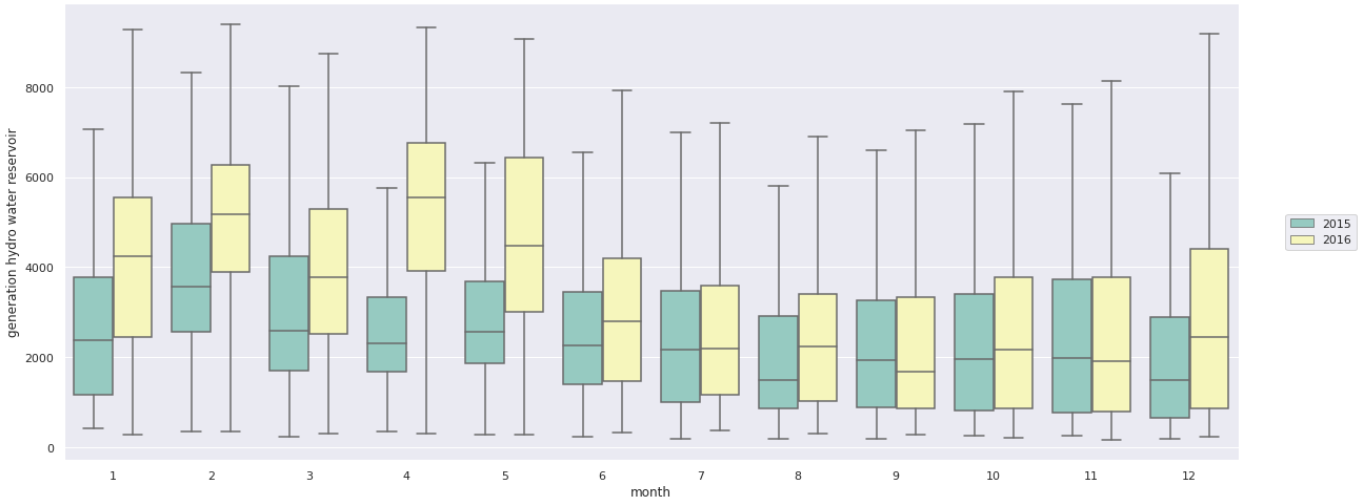
In the first half of the year, price gaps are considerable, sometimes the price goes down of up to 50% for some months in particular. What about the total load gaps and the evolution of renewable production sources led by hydroelectric and wind generation reservoir between 2015 and 2016 ?

3. Total load monthly boxplots 2015-2016 :

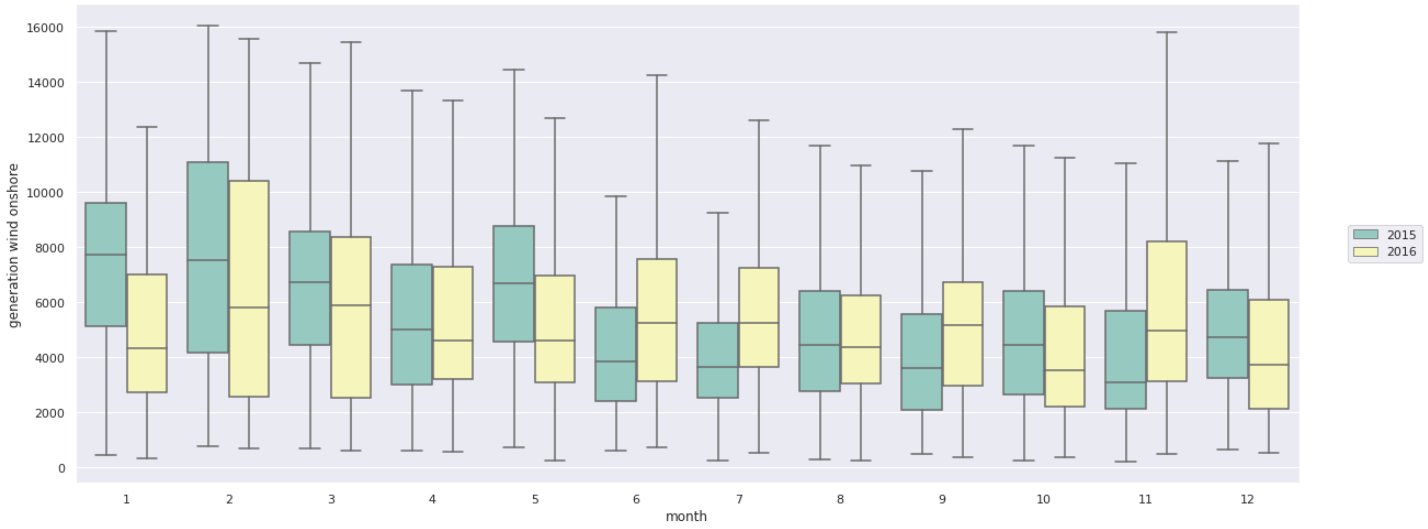


the electricity demand in Spain in 2016 was 0.5% higher than that of 2015. This shows that the price is not completely correlated to the load, hence other variables might explain the electricity price variation between 2015 and 2016. Concerning the demand growth, once the effects of temperature and working patterns will be corrected, growth might become null.

3. Hydrowater generation monthly variation boxplots 2015-2016 :

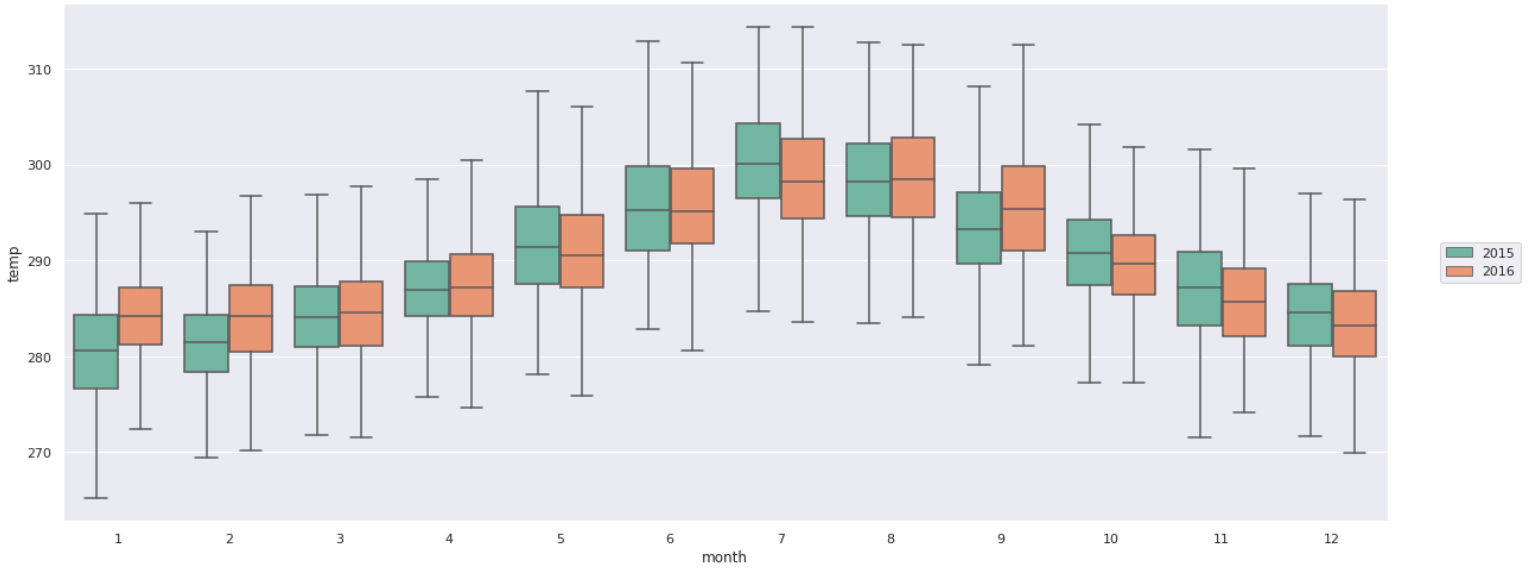


4. Wind monthly variation boxplots 2015-2016 :



The decrease in the average price has been determined by the low prices during the first half of the year, mainly due to the increase in electricity production with less expensive technologies such as hydroelectric, wind, as well as the decrease during that period in the prices of fossil fuels used in the electricity generation.

5. Evolution of temperatures between 2015-2016 :



TWe can clearly see the impact of the temperature on the price with a rise in temperature during the first months of 2016, particularly in January when demand fell compared to 2015.

PREPROCESSING

1.Preprocessing for energy data :

Handling NaN and Zéro values :

	Zero Values	Missing Values	Total Zero + Missing Values	% Total Zero + Missing Values
generation fossil coal-derived gas	35046	18	35064	100.00000
forecast wind offshore eday ahead	0	35064	35064	100.00000
generation wind offshore	35046	18	35064	100.00000
generation marine	35045	19	35064	100.00000
generation fossil oil shale	35046	18	35064	100.00000
generation fossil peat	35046	18	35064	100.00000
generation geothermal	35046	18	35064	100.00000
generation hydro pumped storage aggregated	0	35064	35064	100.00000
generation hydro pumped storage consumption	12607	19	12626	36.00000
generation fossil brown coal/lignite	10517	18	10535	30.00000
total load actual	0	36	36	0.10000
generation biomass	4	19	23	0.10000
generation fossil oil	3	19	22	0.10000
generation other	4	18	22	0.10000
generation waste	3	19	22	0.10000
generation hydro run-of-river and poundage	3	19	22	0.10000
generation hydro water reservoir	3	18	21	0.10000
generation other renewable	3	18	21	0.10000
generation solar	3	18	21	0.10000
generation fossil hard coal	3	18	21	0.10000
generation wind onshore	3	18	21	0.10000
generation nuclear	3	17	20	0.10000
generation fossil gas	1	18	19	0.10000

After changing the variable formats, we group the number of missing values (Number of zeros and NAN) for each energy column in a data frame. Some columns have 100% of values (zeros + NANs), they will be dropped. In the other remaining columns, generation hydro pumped storage consumption may look suspicious, but we should consider that this type of energy is generally used for balancing the actual load, it is consumed when there are peaks in energy demand. Generation fossil brown coal\lignite will be merged with fossil hard coal later in the feature engineering part.

In the second part, we will need to replace NAN values for the remaining variables in the energy data.

For the other production columns, the missing values represent 10% of the data, which is a low rate, it should also be noted that the missing values for these different production variables coincide. The demand variable has 36 Missing values, some of these values coincide with the missing values of the production types, however other NAN values are found at the level of other observations.

Interpolation is a simple and adequate method for this case to handle NAN values for our case.

2.Preprocessing for weather data :

A. Dropping duplicates :

The climate data contains 178,396 observations grouping together the climate data of the 5 cities mentioned above. This still does not allow a merge with data energy. Since the weather observations for each city needs to be equal to the number of observations in the energy data. We have to drop the duplicate data following the index rows.

We use the following code to drop duplicates :

```
weather = weather.reset_index().drop_duplicates(subset=['time', 'city_name'],keep='first').set_index('time')
```

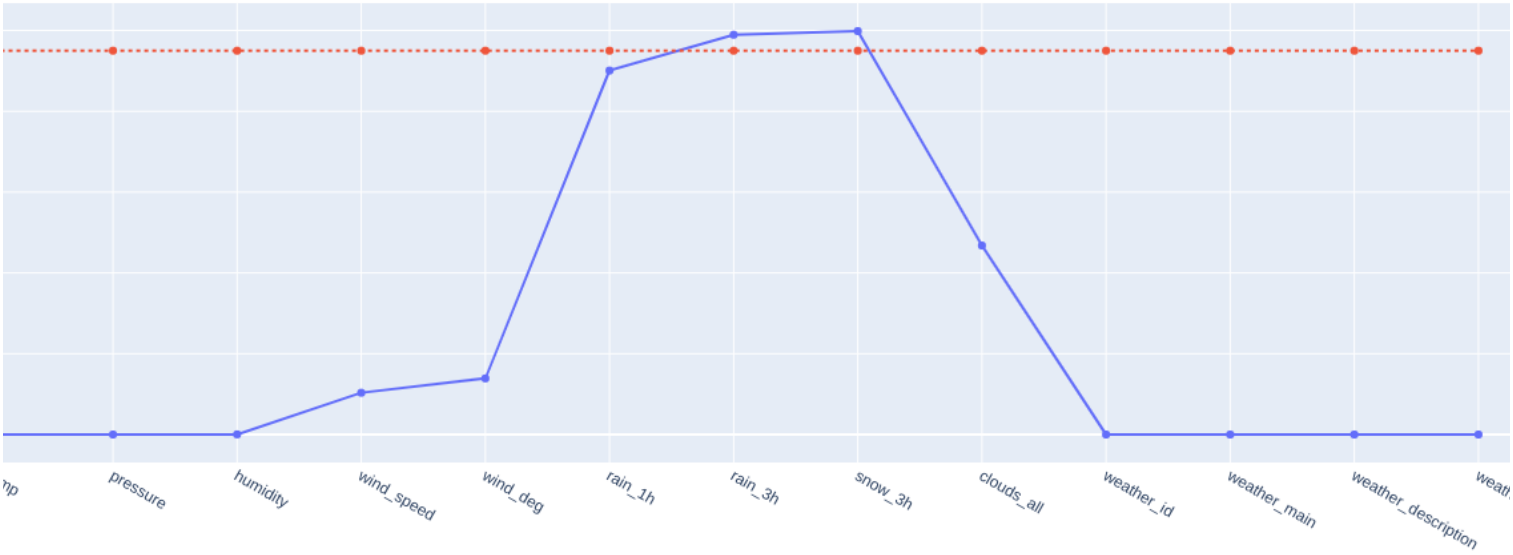
B. Partial output :

As a result we get a weather data with the following shape : (175320,16)

C. Cleaning :

We will apply some cleaning to the weather data beforemerging it with energy data.

→ Since there are no NaN values in this data, we visualize the number of zero values for each variable :



- We will drop the train_3h and snow_3h columns following the number of zero values they contain.

- We will also drop the temp_min / hour and temp_max / hour, because there is not a considerable gap between them, and the column temp is sufficient to provide precise information.

- We also drop the some of weather descriptive columns such as weather_main, weather_id and weather_icon. We keep weather_description which provides richer informations.

D. Anomaly removal (Obvious outliers) :

Generally speaking, it is difficult to judge what are the values that we can consider as outliers, if we take the dataset of production by source, demand and price, some strong variations in the price are explained by changes in the source of production, variations in demand and climate or market structure in time t. However, it is clear that there is a certain level of pressure and certain wind speeds which have never been reached in Spain and which we have to fill using interpolation (since it is a time series data).

```
weather.loc[weather.pressure > 1050, 'pressure'] = np.nan
weather.loc[weather.pressure < 950, 'pressure'] = np.nan
weather.loc[weather.wind_speed > 25, 'wind_speed'] = np.nan
weather.interpolate(method='linear', limit_direction='forward', inplace=True, axis=0)
```

Later in the process, we will try to handle a larger amount of outliers following z-score criteria and see the impact on model performance

3.Merging Datasets :

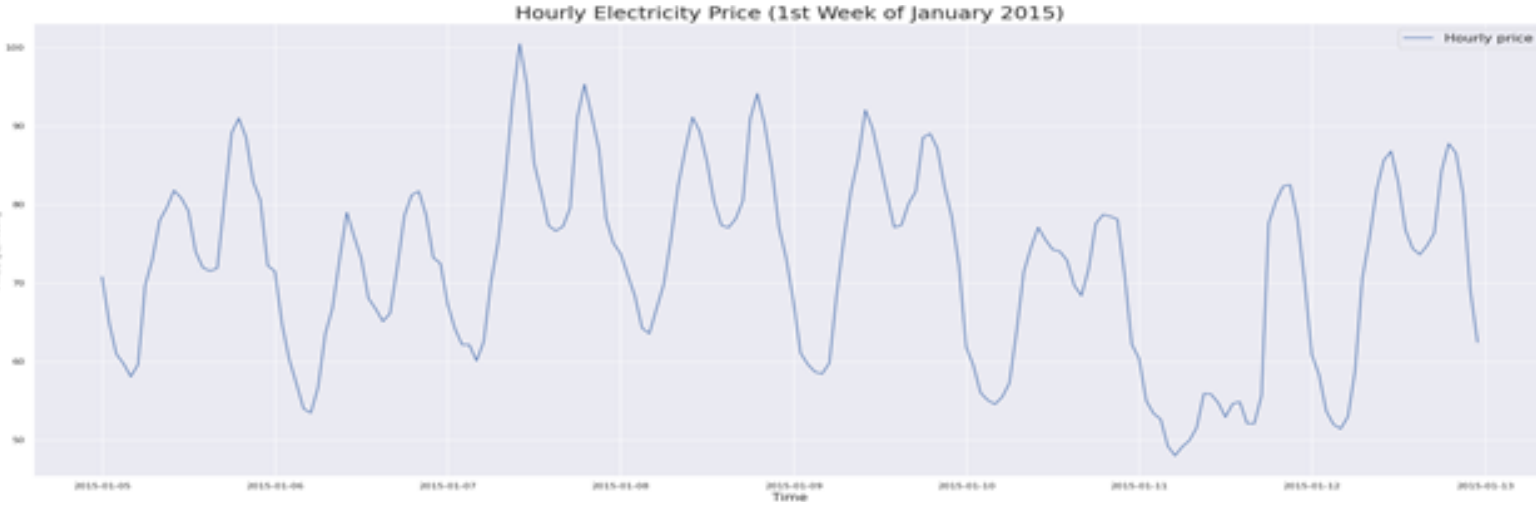
We have merged the two datas. Weathergy is the final data we will be working with. This data contains variables representing the climate in a given region (for example a feature as temp_Barcelona representing the temperature in Barcelona), the sources of production, the total demand as well as the electricity price (euro/MWh). (35064,56)

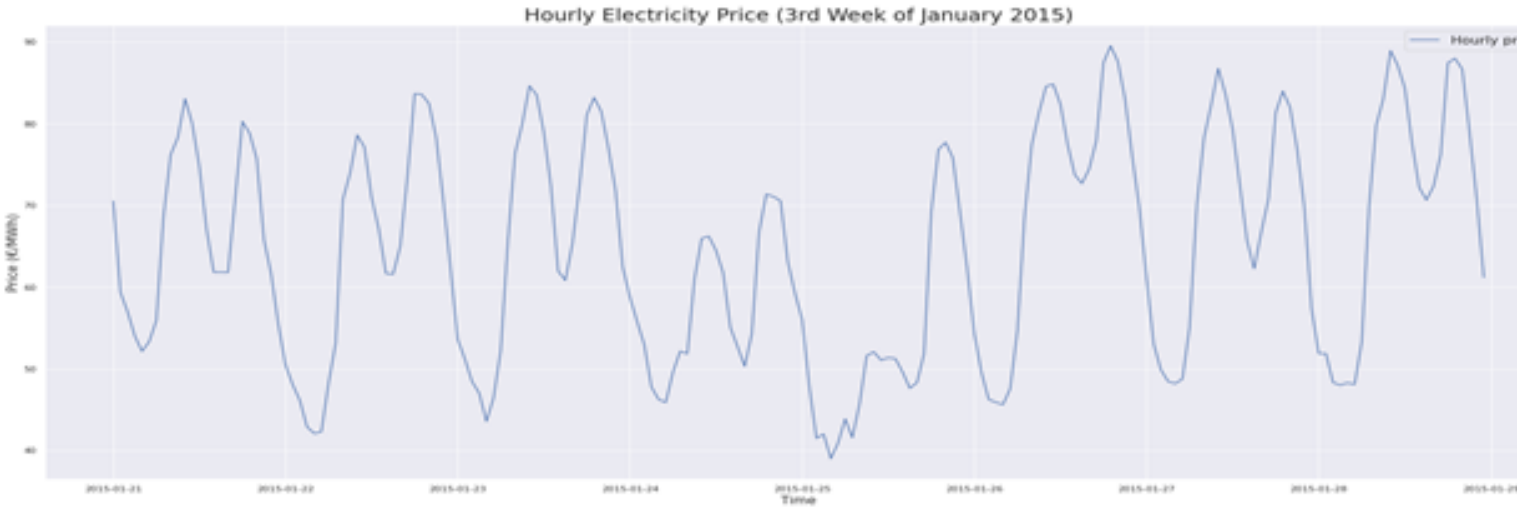
4.Seasonality and trends visualization :

A. Electricity price (monthly frequency-1year lagged) :

The objective of this graph is to visualize the seasonality of the price. This seasonality is important following a monthly scale. We will benefit from adding a month column to handle this seasonality.

B. Hourly price (1st and 2nd week of January) :





We clearly see that the hourly electricity price follows some trends each week. On average, the price drops on the weekends. We visualize some price variations during the same day when the price goes down for a few hours before going up. As a conclusion, It will be interesting to add some features describing the following events : weekends and working hours.

FEATURE ENGINEERING :

1.correlations between variables :

→ **We do the feature engineering based on the correlations between the different variables. We sort the variables which are correlated up to 0.75.**

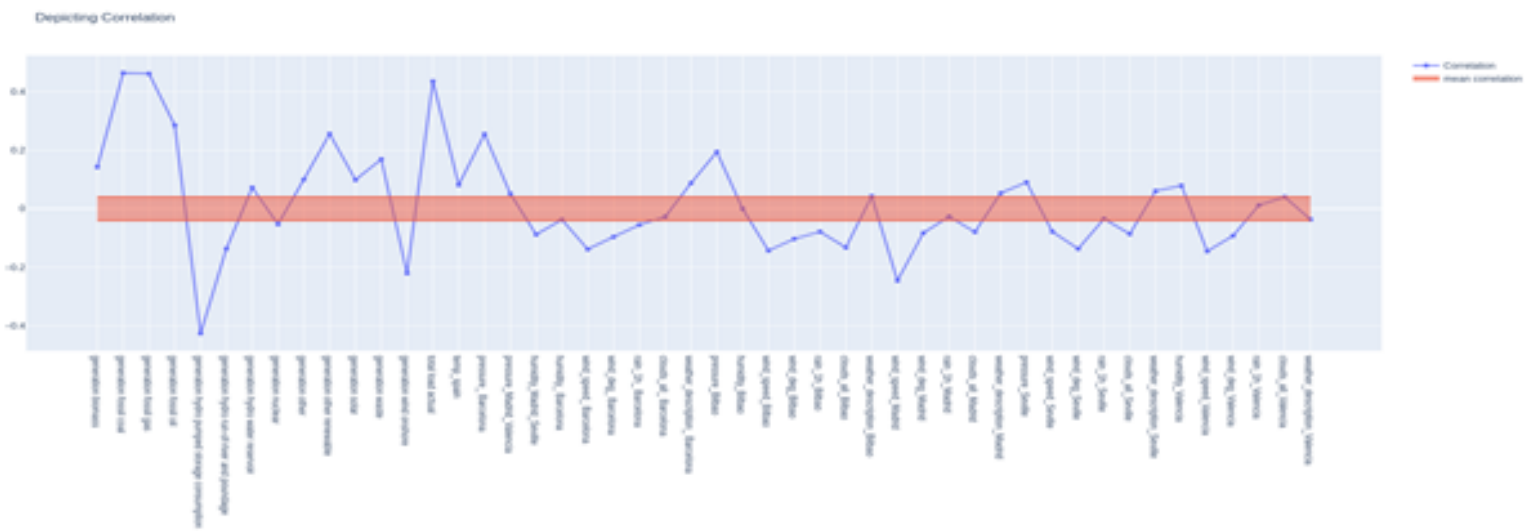
✓ result

The picture below summarize the result:

```
generation fossil brown coal/lignite  generation fossil hard coal          0.768843
generation fossil hard coal           generation fossil brown coal/lignite  0.768843
temp_ Barcelona                       temp_Bilbao                           0.866727
                                      temp_Madrid                           0.903996
                                      temp_Seville                          0.841910
                                      temp_Valencia                         0.917506
```

Some types of production are highly correlated with each other. The other variables that strongly correlates are temperatures. We will apply a feature engineering by merging the correlated variables. We will merge the fossil hard coal and the fossil coal brown/lignite in one variable, it is logical since they represent the same categories of production. The temperatures will be merged to extract one temperature column representing the average temperature in Spain.

→ After that we select relevant features based on their correlation with the price.



- There are 7 features with correlation coefficient LOWER than mean of all correlation coefficients are: ['humidity_ Barcelona', 'clouds_all_ Barcelona', 'humidity_Bilbao', 'rain_1h_Madrid', 'rain_1h_Seville', 'rain_1h_Valencia', 'weather_description_Valencia']

- The remaining dataset contains 40 features and 35064 rows

2.Generating new features :

Generally in Spain, the working hours do not follow a continuous schedule. Between 14:00 and 16:00 there is a drop in activity that can be confirmed by the electricity price graph already shown. The new variable '**business'** will take 3 values :

0: during working hours

1: during break times

2 : outside the mentioned hours

We create a variable named **'weekend'** that takes three values (0,1,2) depending on the days of the week.

- The output (pending the creation of new variables) is a datatset with 35064 observations and 43 columns.

STATIONARITY TEST (PRICE):

Stationarity is an important concept in the field of time series analysis with tremendous influence on how the data is perceived and predicted. When forecasting or predicting the future, most time series models assume that each point is independent of one another. The best indication of this is when the dataset of past instances is stationary. For data to be stationary, the statistical properties of a system do not change over time. This does not mean that the values for each data point have to be the same, but the overall behavior of the data should remain constant.

The Augmented Dickey-Fuller (ADF) test, a type of unit root test, determines how strongly a time series is defined by a trend. Its hypotheses are the following:

- **Null Hypothesis H0:** There is a unit root in the time series, i.e. the series is autocorrelated with (r=1), a time dependent structure and thus, is not stationary.
- **Alternate Hypothesis H1:** The time series has no unit root and is either stationary or can be made stationary using differencing.

Here you can find the code implementation in python :

```
from statsmodels.tsa.stattools import adfuller
X = weathergy['price actual'].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
  print('\t%s: %.3f' % (key, value))
```

✓ result

```
ADF Statistic: -9.147016
p-value: 0.000000
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
```

The ADF statistic (-9.147016) is less than the critical value at 1% (-3.431) and thus, we can say that we reject the null hypothesis H0 with a significance level 1%, meaning that there is not a root-unit in the time series and thus, that it is stationary and don't some updates Also the p-value is less than 0.05.

MODEL IMPLEMENTATION

1.Unsupervised learning methods :

In this section we started by dividing our data set into a train set containing the observations of the years 2015, 2016 and 2017 and a test set containing the remaining observations of 2018

We then chose to fit five models [LinearRegression(), RandomForestRegressor(), LGBMRegressor(), CatBoostRegressor() and XGBRegressor()] to our train set. The results are as follow :

	Models	Train_RMSE	Test_RMSE
0	LinearRegression()	9.438	12.798
1	RandomForestRegressor()	1.455	14.136
2	LGBMRegressor()	4.164	13.368
3	CatBoostRegressor()	3.031	13.527
4	XGBRegressor()	6.338	13.062

We clearly see that linear regression model works better than the other ones but still the values of RMSEs obtained are not that impressive. We also see the great difference between train RMSEs values and test RMSEs values and that's because we are dealing with a time series problem which means that any estimate of performance on the training data would be optimistic and any interpretation would be biased.

As we mentioned before this is a time series problem, and because of that we sought new methods to study it, such as Multiple Train-Test Splits and Walk Forward Validation.

2.Multiple train test splits :

First we used **Multiple Train-Test Splits** which is a method that consists in repeating the process of splitting the time series into train and test sets multiple times such that each test set has the same size and the training set's size keeps growing after every split. This will require multiple models to be trained and evaluated

In our case we chose to split the data 10 times each time the size of the training data grows and the size of the test set remains the same :

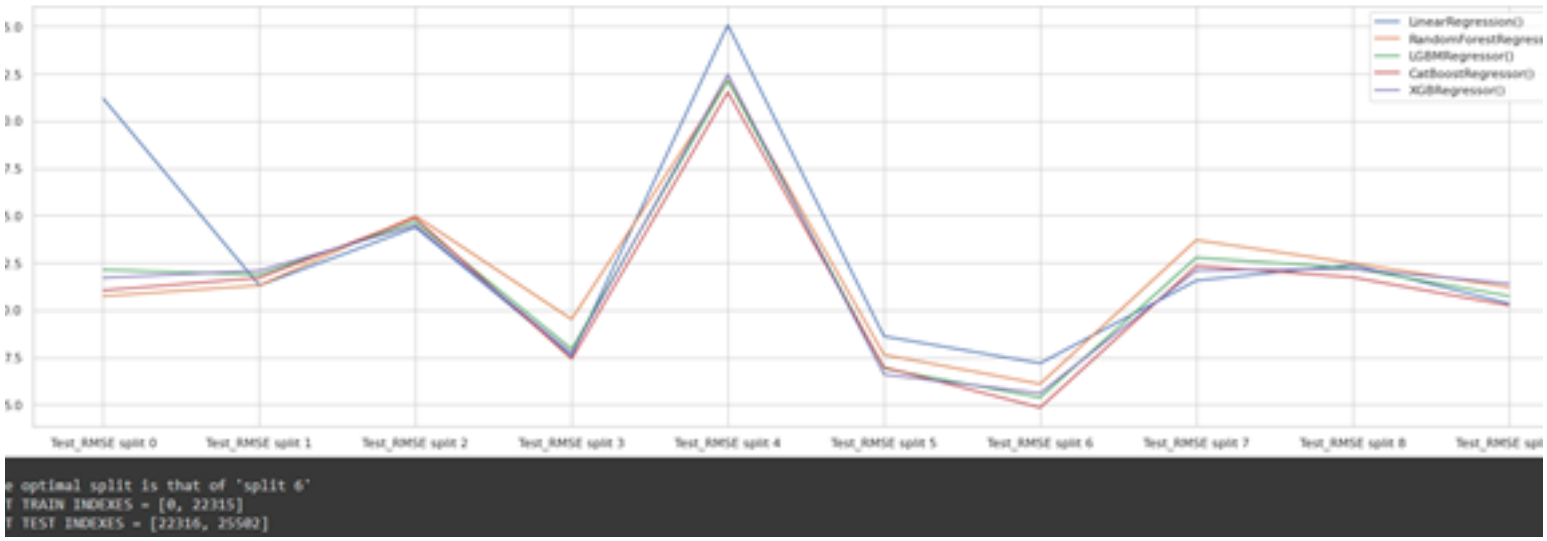
```
# Multiple Train-Test Splits :
from sklearn.model_selection import TimeSeriesSplit

splits = TimeSeriesSplit(n_splits = 10)

p = -1
data = pd.DataFrame()

for (train_index, test_index) in splits.split(weathergy_time) :
  R = []
  p += 1
  train = weathergy_time.iloc[train_index]
  test = weathergy_time.iloc[test_index]

  a_train = train.drop('price actual', axis = 1)
  b_train = train['price actual']

  a_test = test.drop('price actual', axis = 1)
  b_test = test['price actual']

  for (name, model) in all_models :
    model.fit(X = a_train, y = b_train)
    model_predicted_train = model.predict(a_train)
    model_predicted_test = model.predict(a_test)

    model_RMSE_train = np.sqrt(MSE(y_true = b_train , y_pred =  model_predicted_train))
    model_RMSE_test = np.sqrt(MSE(y_true = b_test , y_pred = model_predicted_test))

    R.append((round(model_RMSE_train, 3), round(model_RMSE_test, 3)))

  dat = pd.DataFrame(data = R, columns = [('Train_RMSE split %d' % p), ('Test_RMSE split %d' % p)])
  data = pd.concat([data, dat], axis = 1)

MODELS_with_splits = pd.concat([MODELS, data], axis = 1)
```

✓ result

The results obtained from this method are as follow :

Models	Train_RMSE	Test_RMSE	Train_RMSE split 0	Test_RMSE split 0	Train_RMSE split 1	Test_RMSE split 1	Train_RMSE split 2	Test_RMSE split 2	Train_RMSE split 3	Test_RMSE split 3	Train_RMSE split 4	Test_RMSE split 4	Train_RMSE split 5	Test_RMSE split 5	Train_RMSE split 6	Test_RMSE split 6	Train_RMSE split 7	Test_RMSE split 7
LinearRegression()	9.438	12.798	7.867	21.186	7.632	11.308	7.654	14.378	8.246	7.569	8.022	25.074	9.909	8.616	9.657	7.187	9.363	11.563
omForestRegressor()	1.461	14.131	1.772	10.741	1.619	11.305	1.626	14.576	1.496	9.514	1.404	22.151	1.512	7.638	1.448	6.097	1.443	13.702
LGBMRegressor()	4.164	13.368	2.255	12.134	2.826	11.874	3.332	14.664	3.614	7.939	3.629	22.147	3.967	6.908	4.025	5.369	4.057	12.779
CatBoostRegressor()	3.031	13.527	1.767	11.041	2.074	11.704	2.476	14.864	2.718	7.415	2.643	21.529	2.885	6.972	2.941	4.844	2.973	12.338
XGBRegressor()	6.338	13.062	5.070	11.709	5.128	12.085	5.532	14.478	5.784	7.726	5.770	22.463	6.421	6.595	6.390	5.607	6.285	12.099

We then drew a plot depicting the evolution of Test_RMSE in each split for all models :



We concluded that the 6th split lead to reletavely the smallest values of RMSE for all models. Therefore we used the sizes of both the train and test set of the 6th split in our next application :

We isolated the last 3187 hours of 2018 (that's approximately the last 4 months and a half of 2018) and tried to predict the electricity price in that period using the 22315 hours (which is approximately 2 years and a half) prior to the last 4 months of 2018

✓ result

	Models	Train_RMSE - train : 2 1/2 Y test 2 M	Test_RMSE - train : 2 1/2 Y test 2 M
0	LinearRegression()	8.619	5.991
1	RandomForestRegressor()	1.311	6.628
2	LGBMRegressor()	3.821	6.457
3	CatBoostRegressor()	2.826	7.680
4	XGBRegressor()	6.265	5.541

We can see right away that the models performed better wich lead us to the assertion that trainning our models on smaller data sets leads to more robust estimates.

3.Walk forward validation :

```
# Walk Forward Validation :
n_train = 5000
n_records = len(weathergy_time)
step = 1500

p = -1
data = pd.DataFrame()

for i in range(n_train, n_records, step) :
  R = []
  p += 1
  train, test = weathergy_time[0 : i], weathergy_time[ i : i + step]

  a_train = train.drop('price actual', axis = 1)
  b_train = train['price actual']

  a_test = test.drop('price actual', axis = 1)
  b_test = test['price actual']

  for (name, model) in all_models :
    model.fit(X = a_train, y = b_train)
    model_predicted_train = model.predict(a_train)
    model_predicted_test = model.predict(a_test)

    model_RMSE_train = np.sqrt(MSE(y_true = b_train , y_pred =  model_predicted_train))
    model_RMSE_test = np.sqrt(MSE(y_true = b_test , y_pred = model_predicted_test))

    R.append((round(model_RMSE_train, 3), round(model_RMSE_test, 3)))

  dat = pd.DataFrame(data = R, columns = [('Train_RMSE expansion %d' % p), ('Test_RMSE expansion %d' % p)])
  data = pd.concat([data, dat], axis = 1)

MODELS_expansions = pd.concat([MODELS, data], axis = 1)
```

In the second approach we implemented the method known as Walk Forward Validation which consists in training our models on a minimum number of observations (in our case 5 000 hours), only to test them on a relatively small size of data that comes immediatly after the train set forming a time step also known as sliding or expanding window and then we add the later to the original training set and test the resulting data set on the following time step and so on and so forth. We repeat this process through out our entire data set.

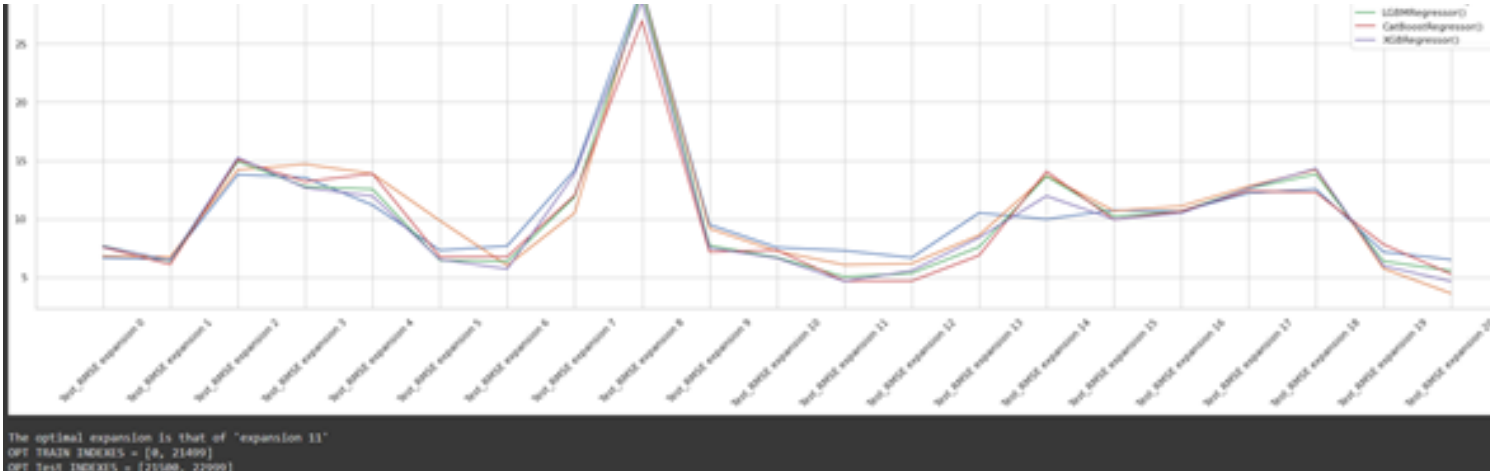
This would give the models the best opportunity to make good forecasts at each time step. We can evaluate our machine learning models under this assumption.

result

The results are as follow :

Models	Train_RMSE	Test_RMSE	Train_RMSE expansion 0	Test_RMSE expansion 0	Train_RMSE expansion 1	Test_RMSE expansion 1	Train_RMSE expansion 2	Test_RMSE expansion 2	Train_RMSE expansion 3	Test_RMSE expansion 3	Train_RMSE expansion 4	Test_RMSE expansion 4	Train_RMSE expansion 5	Test_RMSE expansion 5	Train_RMSE expansion 6	Test_RMSE expansion 6	Train_RMSE expansion 7	Test_RMSE expansion 7
LinearRegression()	9.438	12.798	7.937	6.668	7.607	6.509	7.335	13.795	7.639	13.542	8.014	11.186	8.244	7.309	8.106	7.684	8.042	14.174
RandomForestRegressor()	1.461	14.131	1.675	6.815	1.590	6.749	1.575	14.206	1.620	14.684	1.554	13.862	1.504	9.848	1.439	6.009	1.347	10.503
LGBMRegressor()	4.164	13.368	2.601	7.685	2.805	6.397	3.002	14.916	3.339	12.766	3.462	12.543	3.585	6.452	3.608	6.351	3.585	11.880
CatBoostRegressor()	3.031	13.527	1.991	7.576	2.097	6.090	2.257	15.080	2.465	13.187	2.638	13.873	2.695	6.764	2.688	6.760	2.640	12.021
XGBRegressor()	6.338	13.062	5.145	7.634	5.124	6.460	5.142	15.254	5.482	12.664	5.706	11.958	5.792	6.522	5.818	5.708	5.746	13.867

We then drew a plot depicting the evolution of Test_RMSE in each split for all models



We concluded that the 11th time step lead to relatively the smallest values of RMSE for all models. Therefore we used the sizes of both the train and test set of the the 11th time step in our next application :

We isolated the last 1500 hours of 2018 (that's approximately the last 2 months of 2018) and tried to predict the electricity price in that period using the 21500 hours (which is approximately 2 years and a half) prior to the last 2 months of 2018

result

We obtained the following results :

	Models	Train_RMSE - train : 2 1/2 Y test 2 M	Test_RMSE - train : 2 1/2 Y test 2 M
0	LinearRegression()	8.619	5.991
1	RandomForestRegressor()	1.311	6.628
2	LGBMRegressor()	3.821	6.457
3	CatBoostRegressor()	2.826	7.680
4	XGBRegressor()	6.265	5.541

As we can see this method lead to the best results so far so we decided that from this point forward we will keep our train and test data the way they were established in this method :

```
a = len(weathergy_time) - len(train_exp[11]) - len(test_exp[11])
b = len(weathergy_time) - len(test_exp[11])

to_train = list(weathergy_time.index)[a : b]
to_test = list(weathergy_time.index)[b:]

X_train = weathergy_time.drop(['price actual'], axis = 1).iloc[to_train]
Y_train = weathergy_time['price actual'][to_train]

X_test = weathergy_time.drop(['price actual'], axis = 1).iloc[to_test]
Y_test = weathergy_time['price actual'][to_test]
```

A. Handling outliers :

In this section we used two methods to perform the task at hand, first we used Isolation Forest and then z-score.

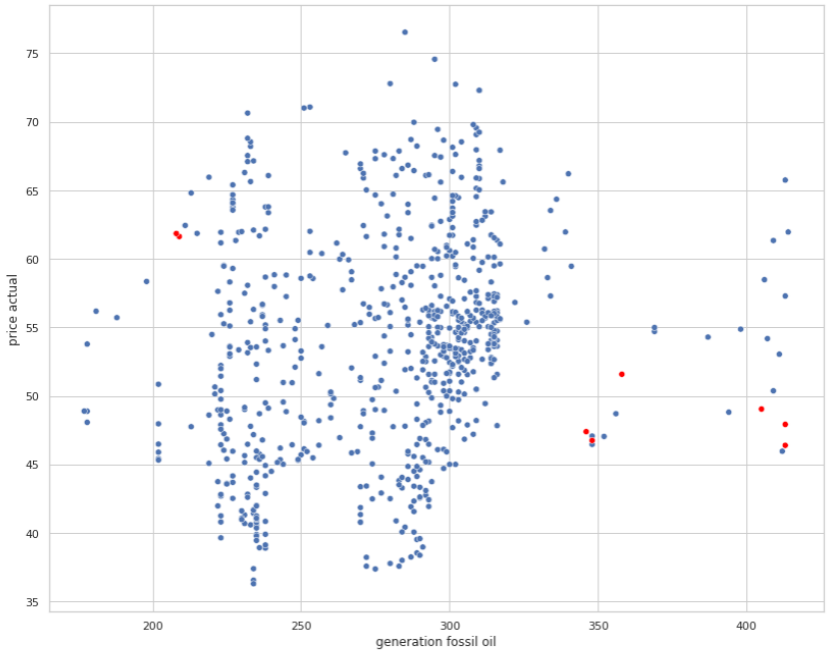
- Isolation Forest :

This algorithm uses decision trees to detect anomalies in the data set that is to say an observation or event that deviates so much from other events. Isolation Forest is a method that thrives on subsets of the original data because it makes the task of finding the anomalies easier for the algorithm.

We divided our train data set into chunks of 720 hours (or 1 month) before implementing this method :

```python
# Let's split the TRAIN SET into mutiple subsets :
DATA = weathergy.drop(['time', 'price actual'], axis = 1).iloc[to_train]

INDEXES = list(DATA.index)

step = 720  # Monthly divide
increment = len(INDEXES) // step

time_jumps = []
for i in range(increment) :
  if i == 0 :
    split = INDEXES[i + i*step : (1+i)*step]
  else : split = INDEXES[i + i*step - 1: i + (1+i)*step]
  time_jumps.append(split)

time_jumps[-1] += list(range(time_jumps[-1][-1] , list(DATA.index)[-1]))

splits = []
for span in time_jumps :
  splits.append(DATA.loc[span])

# Anomaly detection using Isolation Forest on the sub sets :

iso = IsolationForest(n_estimators = 500, contamination = 0.01, random_state = 42, n_jobs = -1)

k = 0
k_index_split_outliers = []
for chunk in splits :
  k += 1
  not_inliers = iso.fit_predict(chunk)
  k_index_split_outliers.append((k, chunk.index, chunk, not_inliers))
```

We then visualised the outliers in a random subset :



After detecting outliers (238 anomaly), we isolated them and removed them before applying our models on the newly refined data and this we got :

	Models	Train_RMSE iforest	Test_RMSE iforest
0	LinearRegression()	8.553	6.029
1	RandomForestRegressor()	1.300	6.541
2	LGBMRegressor()	3.787	6.632
3	CatBoostRegressor()	2.791	7.528
4	XGBRegressor()	6.251	5.576

The RMSEs didn't change significantly so we opted for another method to see if there was going to be any difference.

- z-score method :

A z-score describes the position of a raw score in terms of its distance from the mean, when measured in standard deviation units. The z-score is positive if the value lies above the mean, and negative if it lies below the mean.

While implementing this method we built a loop to see which threshold works best for each model.

✓ result

we got the following results :

	Models	Threshold	Train_RMSE z-score	Test_RMSE z-score
0	LinearRegression()	5	8.597	5.975
1	RandomForestRegressor()	3	1.290	6.309
2	LGBMRegressor()	7	3.795	6.415
3	CatBoostRegressor()	6	2.813	7.443
4	XGBRegressor()	5	6.189	5.518

We can clearly see that this method of detecting outliers works best in our case. therefore we will be using it later in our study after tunning our models

4.Hyperparameters tuning using Bayesian Optimization

Bayesian Optimization is a probabilistic model based approach for finding the minimum of any function that returns a real-value metric. it allows us to simultaneously tune more parameters with fewer experiments and find better values, unlike grid search which analyses all possible combinations of parameters.

We implemented this method by building a function, for each model, that receives as an input the train data set and the target. And returns the parameters tuned

For example : here's the function used to tune the parameters of Catboostregressor() :

```python
def CAT_Bayes_Tuning(X, Y, init_round, opt_round, n_folds, random_seed, n_estimators):
    # Prepare the features dataset :
    train_data = CAT.Pool(data = X, label = Y)

    # Parameters selected for tuning (2 Parameters are selected) :
    def eval(depth, bagging_temperature):
        params = {
            "iterations": 100,
            "learning_rate": 0.05,
            "eval_metric": "R2",
            "verbose": False,
            "loss_function" : 'RMSE',
        }
        params[ "depth"] = int(round(depth))
        params["bagging_temperature"] = bagging_temperature

        cv_result = CAT.cv(train_data, params, nfold = n_folds, seed = random_seed, verbose_eval = 200, stratified = False)

        return np.max(cv_result['test-R2-mean'])

    # Tuning the 2 parameters selected using BayesianOptimization :
    Opt = BayesianOptimization(f = eval, pbounds = {'depth': (1, 12),           # Set ranges upon which the tuning will take place
                                                    'bagging_temperature': (1, 30)
                                                    }, random_state = 42)
    Opt.maximize(init_points = init_round, n_iter = opt_round, acq = 'ei')

        # Return best parameters :
    BEST = pd.DataFrame(Opt.res)
    best_params = list(BEST[BEST.target == max(BEST.target)].params)[0]
    return best_params
```

- Implementation :

```python
CAT_best_params = CAT_Bayes_Tuning(X_train, Y_train, init_round = 5, opt_round = 5, n_folds = 10, random_seed = 42, n_estimators = 7000)
```

- Output :

```python
CAT_best_params
```

```
{'bagging_temperature': 30.0, 'depth': 12.0}
```

After using this process for three models : LGBMRegressor, CatBoostRegressor and Xgboost we got the following newly tunned models :

```
Lgbm_TUNNED = LGBMRegressor(
                colsample_bytree = 0.8826251881706276,
                feature_fraction = 0.9993473383012929,
                lambda_l1 = 3.3906808650076914,
                lambda_l2 = 7.141816021177675,
                learning_rate = 0.19228001074573395,
                max_depth = 13,
                min_child_samples = 36.17363226157706,
                min_child_weight = 37.854444970521875,
                min_data_in_leaf = 26,
                min_split_gain = 0.30010198951756967,
                num_leaves = 120,
                reg_alpha = 6.894364111158153,
                reg_lambda = 5.970701521997151,
                subsample = 0.1385307620912042,
                objective = 'regression',
                bagging_fraction = 0.8,
                bagging_freq = 1,
                metric = 'rmse',
                verbosity = -1,
                random_state = 42
)

Cat_TUNNED = CatBoostRegressor(
                bagging_temperature = 30.0,
                depth = 12,
)

Xgb_TUNNED = XGBRegressor(
                colsample_bytree = 0.5621780831931538,
                gamma = 9.51207163345817,
                learning_rate = 0.7322619478695936,
                max_delta_step = 2.993292420985183,
                max_depth = 9,
                min_child_weight = 1.5599452033620265,
                reg_alpha = 1.5227525095137953,
                reg_lambda = 8.795585311974417,
                scale_pos_weight = 6.41003510568888,
                subsample = 0.8540362888980227,
                booster = 'gbtree',
                objective = 'reg:squarederror'
)
```

result

Next, we fit the tunned models on the train data set and got the following results :

	Models	Train_RMSE	Test_RMSE
0	LGBMRegressor(TUNNED)	1.695	6.348
1	CatBoostRegressor(TUNNED)	0.455	5.812
2	XGBRegressor(TUNNED)	1.231	7.074

result

After that we removed the outliers from the train data again using z-score method only to get better results :

	Models	Threshold	Train_RMSE z-score	Test_RMSE z-score
0	LGBMRegressor(TUNNED)	7	1.667	5.937
1	CatBoostRegressor(TUNNED)	7	0.418	5.439
2	XGBRegressor(TUNNED)	7	1.243	6.487

5.Blending

In this section, we utilized a method known as blending in which we combined the forecasts of the most prominent models in our study to see if we will get a better RMSE value.

First, we built a function capable of blending a list of models and returns a data frame containing train and test RMSEs of all possible combinations as well as the coefficients used to perform this application :

Here are the coefficients used to combine the predictions :

```
27    A = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
```

We chose to combine the predictions of both CatBoostRegressor(TUNNED) and Xgboost() after removing outliers :

```
# Models upon which we apply the function Blend :
blend_xgb_Cat_TUNNED = [('XGBRegressor()', xgb), ('CatBoostRegressor(TUNNED)', Cat_TUNNED)]

BLEND_xgb_Cat_TUNNED = Blend(blend_xgb_Cat_TUNNED)
```

✓ result

After using the Blend function we got these results :

As we can see from this data frame the 5th row contains the optimal combination, which is the best result obtained in our entire study

	RMSE_Train	RMSE_Test	Combination
0	0.852947	5.552278	[0.1, 0.9]
1	1.423757	5.471544	[0.2, 0.8]
2	2.020710	5.413150	[0.3, 0.7]
3	2.626038	5.377822	[0.4, 0.6]
4	3.235042	5.366017	[0.5, 0.5]
5	3.845978	5.377890	[0.6, 0.4]
6	4.458050	5.413284	[0.7, 0.3]
7	5.070847	5.471743	[0.8, 0.2]
8	5.684136	5.552539	[0.9, 0.1]

Summary

Finally her's a recap of all the obtained results through out our study

	Models	Train_RMSE	Test_RMSE	Train_RMSE - train : 2 1/2 Y test 4 1/2 M	Test_RMSE - train : 2 1/2 Y test 4 1/2 M	Train_RMSE - train : 2 1/2 Y test 2 M	Test_RMSE - train : 2 1/2 Y test 2 M	Train_RMSE iforest	Test_RMSE iforest	Train_RMSE z-score	Test_RMSE z-score
0	LinearRegression()	9.438	12.798	8.964	10.047	8.619	5.991	8.553	6.029	8.597	5.975
1	RandomForestRegressor()	1.455	14.136	1.323	11.274	1.308	6.647	1.3	6.541	1.29	6.309
2	LGBMRegressor()	4.164	13.368	3.866	10.682	3.821	6.457	3.787	6.632	3.795	6.415
3	CatBoostRegressor()	3.031	13.527	2.854	9.899	2.826	7.68	2.791	7.528	2.813	7.443
4	XGBRegressor()	6.338	13.062	6.222	11.102	6.265	5.541	6.251	5.576	6.189	5.518
5	pm.auto_arima()	6.298	5.655								
6	LGBMRegressor(TUNNED)					1.695	6.348			1.667	5.937
7	CatBoostRegressor(TUNNED)					0.455	5.812			0.418	5.439
8	XGBRegressor(TUNNED)					1.231	7.074			1.243	6.487
9	Blending XGBRegressor() AND CatBoostRegressor(...									3.235	5.366

3. Hourly electricity price prediction using time lag (history size) :

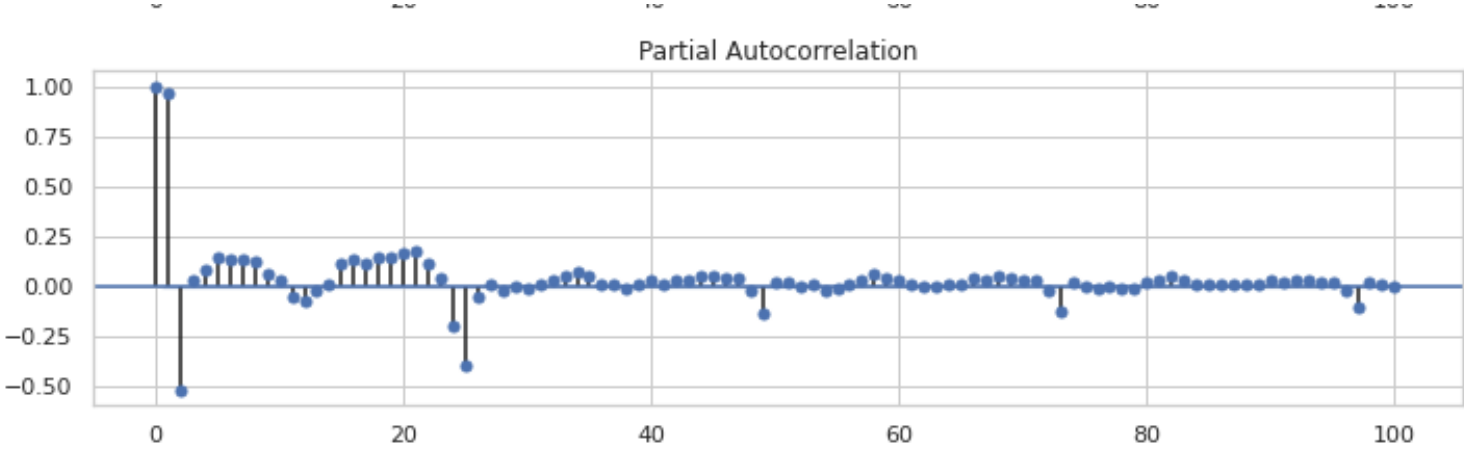
As we know the particularity in time series is that we have to handle the dependance between the price and the other features, also we have to take into consideration the time dependance of the price and other features, to do so we need to reframe the time series data to adapt it to the supervised learning algorithms. As a projection, in this section we will take into consideration the historical values (previous hours) of each feature including the price (Multivariate time series).

- **Function implementation :**

```
def twenty_four_hour_split(start_index, end_index, history_size, dataset_target, target) :

  data = []
  labels = []

  start_index = start_index + history_size
  if end_index is None :
    end_index = len(dataset) - target_size

  for i in range(start_index, end_index) :
    indices = range(i - history_size, i)
    data.append(dataset_target.iloc[indices])
    labels.append(target[i])

  data_array = np.array(data)
  data_shaped = data_array.reshape(-1, data_array.shape[1] * data_array.shape[2])

  t_feats = []

  for i in list(range(24, 0, -1)) :
    for feat in list(dataset_target.columns) :
      t_feats.append((feat + ' t-%d' % i))

  DATA = pd.DataFrame(data = data_shaped, columns = t_feats)

  return (DATA, labels)
```

The first step is that we split the data into twenty four hours data and affect them to a list. We reshape the list in order to obtain 24 hours values in one row, The final misses 24 hours record because as we mentionned the goal is to predict the hourly electricity price based on the previous 24 hours data

3. Hourly electricity price prediction using time lag (history size) :



- **Why do we choose the previous 24 hours (time lag) :**

The partial autocorrelation plot of the eletricity price time series shows that the direct relationship between an observation at a given hour (t) is strongest with the observations at t-1, t-2, t-24. We are going to use the 25 previous values of each time series which will constitute a feature for our models.

if we choose to use specific past values of a given feature, based on the cross-correlation between the electricity price and each one of the features in the dataset, it will be better but we will keep the 24 previous hours to reduce the complexity.

- **Results :**

We use the supervised learning algorithms as in the walk forward validation we get an smaller RMSE especially for booster algorithms.

The following table summarizes the results found :

	Models	Train_RMSE 24h	Test_RMSE 24h
0	LinearRegression()	2.290	2.293
1	RandomForestRegressor()	0.950	2.488
2	LGBMRegressor()	1.902	2.338
3	CatBoostRegressor()	1.669	2.264
4	XGBRegressor()	2.378	2.415

CONCLUSION

In this analytics edge project, we manipulated a time Serie dataset of which the manipulation is slightly different from what we used to do in the analytics edge course, owning to the fact that time is involved in the analysis, which means that the split for the train-set and the test-set must not be completely a random. Also the performance evaluation of the models must take in consideration the period of time on which the model has been trained, a thing that justities the different methods that we have chosen for the predictions, like Multiple train test splits and walk forward validation. The models we've build have conducted to some fairly precise predictions with a best RMSE of 5,36 considering the use of 21500 hours to predict 1500 hour.

Other than that, our analysis is far from being complete, because the data we succeeded to collect is not enough, there was some important features that must be taken into consideration for the electricity price forecasting, for instance, the amount of exported source of energy used for the electricity production, the number of outages that have occurred during the period of the electricity production, but unfortunately, we could not find any hourly data about these variables. Due to these previous predicament and the short period of time in which we had to prepare this project we decided that we will keep conducting further research on this subject for a better RMSE.