Machine Learning Engineer Nanodegree

Capstone Project

Monish Ananthu
October 28th, 2018

Energy Price Prediction

I. Definition

a. Project Overview

The Bundesnetzagentur's electricity market information platform "SMARD" is an abbreviation of the German term “Strommarktdaten”, which translates to electricity market data. Data that is published on SMARD’s website gives an up-to-date and in-depth overview of what is happening on the German electricity market.

The SMARD Website offers real time data for analysis. This data is available for download in different formats.
Link to SMARD Website (https://www.smard.de/en)

The following electricity market data categories can be accessed/downloaded:

- Electricity generation
  - Actual generation
  - Forecasted generation
  - Installed capacity

- Electricity consumption
  - Realised consumption
  - Forecasted consumption

- The market
  - Wholesale market price
  - Commercial exchanges
  - Physical flows

- System stability
  - Balancing energy
  - Total costs
  - Primary balancing capacity
  - Secondary balancing capacity
  - Tertiary balancing reserve
  - Exported balancing energy
  - Imported balancing energy

The above data is available from 2015 onwards. The statistical data available is visuallized and limited to a specific subcategory (for example: Electricity generation --> Actual generation). The visualization does not convey how the data is correlated to one another and also the correlation of data between different catagories like "Actual generation" and "Wholesale market price" would be a very interesting to determine.

What makes SMARD Data so interesting?

- Data is already consolidated from different transmission system operators in a standard format
- High frequency of data (in 15 minute / hourly intervals) provides a good basis for data analysis
- Data available from 2015 ist constantly updated

b. Problem Statement

The problem to be solved is the prediction of the wholesale market price of energy [Euro/MWh] using the data available above. The problem at hand is a supervised learning problem in the field of Machine Learning. From the **Datasets and Inputs** section below, we have the following input data:

a. Actual generation
b. Realized Consumption
c. Balancing energy

It is important to find correlations among the above input features and use this information to predict the wholesale market price of energy.

We will discuss the data available in detail in the **Analysis and Preprocessing** section below. Our strategy for arriving at a solution to the above problem comprises of the the following steps:

1. **Data Preparation -** Data first needs to be collected and pre-screened for relevance and completeness. The the data can be visualized to check for relationships between different variables and outliers. The data may also need to be normalized and scaled. This data is split into training and testing sets. We will use the majority of the data for training and evaluate model performance later on using the testing set.
2. **Model Selection & Experimentation -** There are many models available for regression. We will try different models in this step and choose the model with the best prediction.
3. **Model Training -** In the data prepraration step we already split the data set into Training and testing set. We will use the training data set to train the model. If necessary we also can use a subset of data for cross validation, to make sure the model doesn't overfit.
4. **Model Evaluation -** In this step we feed the trained model data it hasn't seen before and evaluate how good the prediction is.
5. **Model Parameter Tuning -** Once we know how a chosen model is performing, we can take a look at the model parameters for possibilities to increase prediction rate. The Grid Search technique is a useful tool to determine the best parameters.
6. **Prediction -** We have chosen a model and tuned it to our problem. This is the final step which helps us fullfill our problem statement.

All data is available in CSV format

The data sets can be downloaded at
https://www.smard.de/en/downloadcenter/download_market_data (Link)
Select category, sub-category, country = Germany, Dates: 01/01/2015 - 31/12/2015,
Filetype: CSV and download file.

We will consider a Data sets for the years 2015 and 2016.

c. Metrics

The problem we've addressed is

- Supervised Learning problem: Input features like Generation, Consumption data are known and we also have a target variable Energy Price we are trying to predict
- Regression based problem: Prediction of a specific value i.e. Energy price

For the above Regression problem, we have the choice of the following metrics:

- Mean Absolute Error (MAE)
- Mean Square Error / Root Mean Square Error (RMSE)
- R squared Error
- Adjusted R squared Error

Mean Absolute Error (MAE)

Mean Absolute Error is the average error for a set of predictions. The absolute value of the differences in the predictions and the actual value is considered [8]. The mathematical of MAE can be represented as

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

Root Mean Squared Error

Root Mean squared Error is the square root of the average of squared differences between prediction and actual observation [8] The mathematical representation is as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

R Squared Error

According to Wikipedia, "R2 is also the square of the correlation (correlation written as a “p” or “rho”) between the actual and predicted outcomes".

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Sum of squares of residuals

$$SS_{res} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

Total sum of squares

$$SS_{res} = \sum_i (y_i - \bar{y})^2$$

Adjusted R Squared

According to [], *Adjusted R squared is a modified version of R Squared that has been adjusted for the number of predictors in the model. The adjusted R-squared increases only if the new term improves the model more than would be expected by chance. It decreases when a predictor improves the model by less than expected by chance. The adjusted R-squared can be negative, but it's usually not. It is always lower than the R-squared.*

For the regression problem, my choice of metric is the Root Mean Square Error (Error)

Both Mean Absolute Error and Root mean squared error are expressed in the same unit of the variable to be predicted. R squared on the other hand is scaled between 0 and 1. If R squared value approaches closer to 1, we know that the prediction is very close to the actual outcome. The prediction does not relate to actual outcome if the value is closer to 0.

In our case choosing R squared gives us an indirect interpretation of the reliablity of the prediction. It does not tell us how much we have swayed in our prediction. However, when we consider RMSE we have the error rate in the same unit as the target variable which provides us with a direct comparision to judge how good the model is doing.

II. Analysis

a. Data Exploration

The data sets can be downloaded at
https://www.smard.de/en/downloadcenter/download_market_data (Link)
Select category, sub-category, country = Germany, Dates: 01/01/2015 - 31/12/2015,
Filetype: CSV and download file.

We will consider a Data sets for the years 2015 and 2016.

The subcategories below refer to feature sets. If a sub-category is not relevant, all features in the feature set can be discarded. Partial relevance means that a part of the the features need to be considered.

Category	Sub-category	Relevant?	Data frequency	Details
Electricity generation	Actual generation	yes	15 mins	Amount of energy generated by different sources at a specific time period
	Forecasted generation	no	15 mins	Forecasted features are not relevant
	Installed Capacity	no	NA	Not enough data
Electricity consumption	Realized consumption	yes	15 mins	Energy consumption at specific time period
	Forecasted consumption	no	15 mins	Forecasted features are not relevant
The Market	Wholesale market price	yes (partially)	15 mins	Energy price per MWh. Only data for Germany is relevant
	Commercial exchanges	no	60 mins	Energy Imports and Exports are out of scope for price prediction
	Physical flows	no	60 mins	Energy Imports and Exports are out of scope for price prediction
System stability	Balancing energy	yes	15 mins	Overall energy balancing volumes and balancing price
	Total costs	no	monthly	Not enough data
	Primary balancing capacity	no	15 mins	Energy balancing efforts and resulting costs are not in scope
	Secondary bancing capacity	no	15 mins	Energy balancing efforts and resulting costs are not in scope

Category	Sub-category	Relevant?	Data frequency	Details
	Tertiary balancing reserve	no	15 mins	Energy balancing efforts and resulting costs are not in scope
	Exported balancing energy	no	NA	No data available for 2015
	Imported balancing energy	no	NA	No data available for 2015

After the initial screening we have the following features:

Category	Sub-category	Feature	Data frequency	Comments
		Date		Date starting 01/01/2015 - 31/12/2015
		Time of day		Timestamps in 15 min intervals for the date range specified above
Electricity generation	Actual generation	Hydropower[MWh]	15 mins	Generated energy in MWh
		Wind offshore[MWh]	15 mins	Generated energy in MWh
		Wind onshore[MWh]	15 mins	Generated energy in MWh
		Photovoltaics[MWh]	15 mins	Generated energy in MWh
		Other renewable[MWh]	15 mins	Generated energy in MWh
		Nuclear[MWh]	15 mins	Generated energy in MWh
		Fossil brown coal[MWh]	15 mins	Generated energy in MWh
		Fossil hard coal[MWh]	15 mins	Generated energy in MWh
		Fossil gas[MWh]	15 mins	Generated energy in MWh
		Hydro pumped storage[MWh]	15 mins	Generated energy in MWh
		Other conventional[MWh]	15 mins	Generated energy in MWh

Category	Sub-category	Feature	Data frequency	Comments
Electricity consumption	Actual consumption	Total[MWh]	15 mins	Feature name needs to be modified to Total consumption for simplicity
The Market	Wholesale market price	Germany/ Austria/ Luxembourg [Euro/MWh]	60 mins	Market prices of other countries are not relevant and need not be considered. The feature name will be renamed to Price Germany for simplicity
System stability	Balancing energy	Balancing energy volume[MWh]	15 mins	Balancing energy in MWh
		Balancing energy price[Euro/MWh]	15 mins	Price for balancing energy Euro/MWh

Each of the features in the dataset contains a value for a particular time period/interval. The feature we like to predict "Wholesale market price" is available every 60 minutes. This implies that the input features which are currently available every 15 minutes need to be reduced to once every 60 minutes to correspond with the predicted feature.

Why is this data set relevant for the problem?

From the information presented in [2], we see clearly

The electricity market brings supply and demand together.

The main element to control the market is the Price.

We already have supply data i.e. energy supply data from different sources and demand data which is the consumption data. We also have the energy price for any give time period. If supply and demand are key factors which influence the price, we already have the relevant data to analyse using Supervised Machine Learning.

b. Exploratory Visualization

We have 4 types of data and we have to explore them separately.

a. Actual generation - contains values of individual energy sources
b. Realized Consumption - total energy consumption (Germany)
c. Balancing energy - Energy required for balancing and price Euro/Mwh
d. Wholesale energy price - target variable

In [20]:

```
master_data[actual_generation].describe()
```

Out[20]:

	biomass_mwh	hydropower_mwh	wind_offshore_mwh	wind_onshore_mwh	p
count	17393.000000	17461.000000	17483.000000	17479.000000	1
mean	1059.656126	453.039030	285.107447	1907.624707	9
std	89.666552	128.472593	231.897884	1606.903028	1
min	660.500000	202.250000	0.000000	28.000000	0
25%	986.250000	354.250000	85.750000	726.250000	0
50%	1080.250000	423.750000	210.250000	1412.250000	1
75%	1147.000000	530.750000	481.750000	2616.250000	1
max	1206.000000	785.250000	915.500000	7738.250000	6

1. Detecting yearly patterns

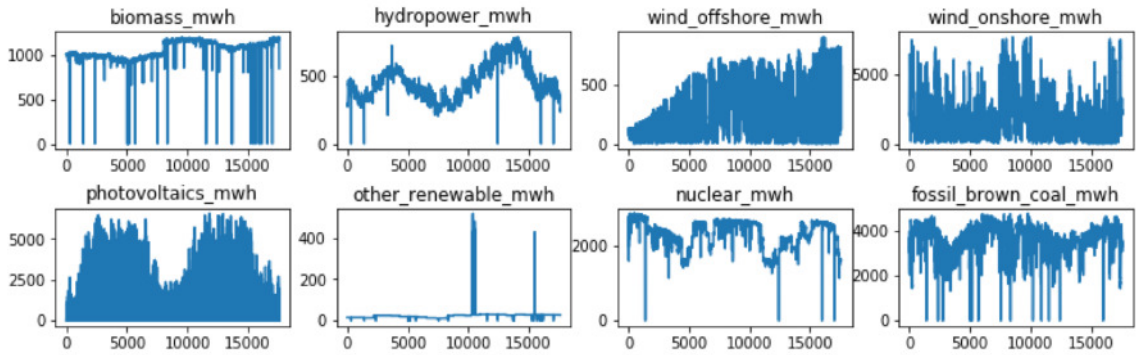
As the data is spread over 2 years, we visualize the data to discover yearly patterns.

In [22]:

```
# Plot actual generation
fig = plt.figure(figsize=(13,8))
fig.subplots_adjust(hspace=.5)

plt.subplot(4,4,1)
master_data['biomass_mwh'].plot()
plt.title('biomass_mwh')

plt.subplot(4,4,2)
master_data['hydropower_mwh'].plot()
plt.title('hydropower_mwh')

plt.subplot(4,4,3)
master_data['wind_offshore_mwh'].plot()
plt.title('wind_offshore_mwh')

plt.subplot(4,4,4)
master_data['wind_onshore_mwh'].plot()
plt.title('wind_onshore_mwh')

plt.subplot(4,4,5)
master_data['photovoltaics_mwh'].plot()
plt.title('photovoltaics_mwh')

plt.subplot(4,4,6)
master_data['other_renewable_mwh'].plot()
plt.title('other_renewable_mwh')

plt.subplot(4,4,7)
master_data['nuclear_mwh'].plot()
plt.title('nuclear_mwh')

plt.subplot(4,4,8)
master_data['fossil_brown_coal_mwh'].plot()
plt.title('fossil_brown_coal_mwh')
```

Out[22]:

```
Text(0.5,1,'fossil_brown_coal_mwh')
```

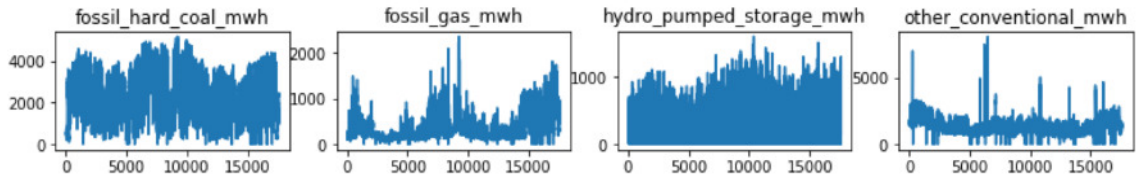


In [23]:

```
fig = plt.figure(figsize=(13,8))
fig.subplots_adjust(hspace=.5)

plt.subplot(4,4,9)
master_data['fossil_hard_coal_mwh'].plot()
plt.title('fossil_hard_coal_mwh')

plt.subplot(4,4,10)
master_data['fossil_gas_mwh'].plot()
plt.title('fossil_gas_mwh')

plt.subplot(4,4,11)
master_data['hydro_pumped_storage_mwh'].plot()
plt.title('hydro_pumped_storage_mwh')

plt.subplot(4,4,12)
master_data['other_conventional_mwh'].plot()
plt.title('other_conventional_mwh')
```

Out[23]:

Text(0.5,1,'other_conventional_mwh')



We currently have 17596 hourly dataset rows over a persiod of 2 years 2015-2016 and 2016-2017 i.e. 8798 per year.

Looking at the above data we can observe that non-renewable sources of energy have a pattern which can be predicted if we were to divide the graphs right in the middle. This is true for nuclear, fossil brown coal, fossil hard coal, fossil gas, hydro pumped storage and other conventional energy sources.

The same is not consistent for renewable sources. Wind energy is weather dependent which does not have set patterns. If we observe biomass and hydropower, they reflect similar patterns but in varying sizes. This could be a result of increasing yearly investments in these energy sources. Photovoltaics appears to be consistent with a predictable pattern.

In [24]:

```
master_data[actual_consumption].describe()
```

Out[24]:

	total_consumption_mwh
count	17596.000000
mean	13663.896255
std	2477.577389
min	7856.750000
25%	11586.750000
50%	13582.500000
75%	15906.437500
max	18975.250000

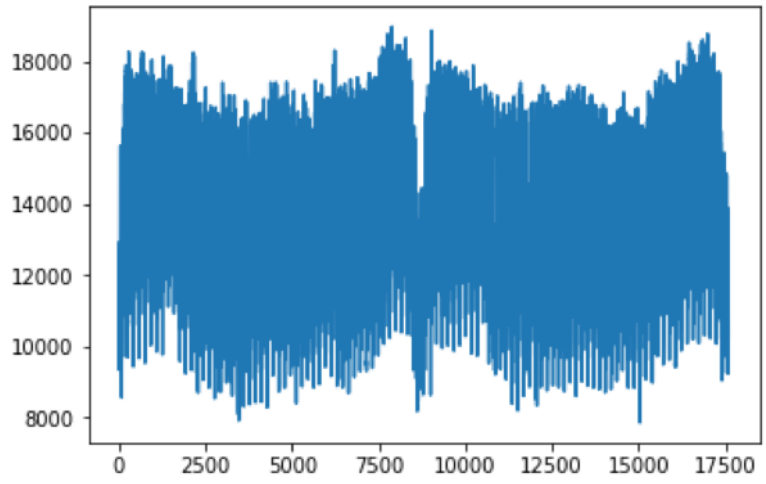
The mean and median of 'total_consumption_mwh' varies to a large extent when compared to features grouped under 'actual_generation'. Scaling is therefore necessary.

In [25]:

```
master_data['total_consumption_mwh'].plot()
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xa8efb70>
```



The consumption data for the second year seems to follow a similar pattern as the first year. This implies that the energy consumption for a certain time of year is consistent to the previous year.

In [26]:

```
master_data[balancing_energy].describe()
```

Out[26]:

	balancing_energy_volume_mwh	balancing_energy_price_euro/mwh
count	17596.000000	17596.000000
mean	165.372471	31.917856
std	464.695247	150.874693
min	-3210.000000	-5997.420000
25%	-102.250000	1.890000
50%	160.000000	39.940000
75%	437.000000	60.882500
max	3271.000000	5824.630000

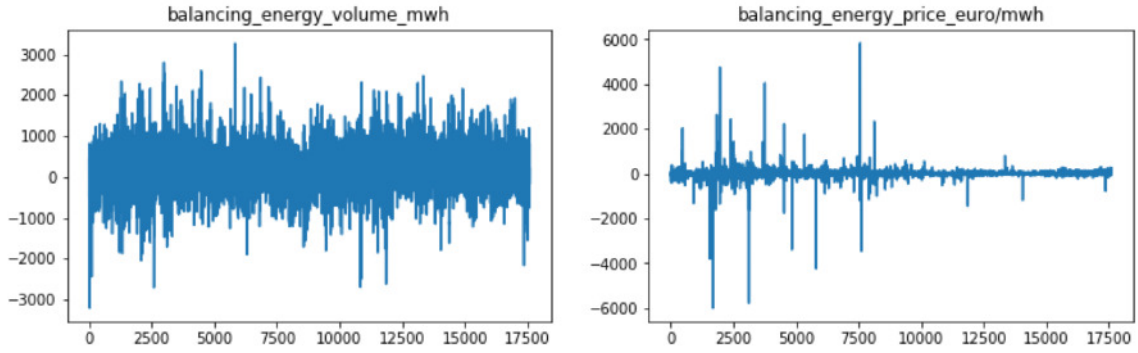
Similar to 'total_consumption_mwh', Data Scaling is also required.

In [27]:

```
fig = plt.figure(figsize=(13,8))

plt.subplot(2,2,1)
master_data['balancing_energy_volume_mwh'].plot()
plt.title('balancing_energy_volume_mwh')

plt.subplot(2,2,2)
master_data['balancing_energy_price_euro/mwh'].plot()
plt.title('balancing_energy_price_euro/mwh')
```

Out[27]:

```
Text(0.5,1,'balancing_energy_price_euro/mwh')
```



The balancing energy volume contains a consistent yearly pattern. This does not apply to the balancing energy price which appears more random.

In [28]:

```
master_data[target].describe()
```

Out[28]:

	price_germany_euro/mwh
count	17476.000000
mean	30.386641
std	12.551637
min	-130.090000
25%	23.460000
50%	29.660000
75%	37.060000
max	104.960000

We still have 120 missing values for **price_germany**. The reason for missing values could be an error in the logging system. In other cases if we look at the data above, we notice negative values. In certain rare periods particularly at the end of the year, there is a such a surplus of wind energy produced that the prices go below zero i.e the consumer gets paid for consuming energy. In the energy market, there is always supply and demand and hence, a price for each time period. Earlier for energy generation we replaced all missing values with zeroes. The case here is different and a price of zero has a false implication and would have a negative influence on the prediction.

The best strategy here would be to either use the Median or the mean.

Median = 29.66
Mean = 30.386641

Since there isn't a large difference between Median and Mean, let us consider the Mean value to fill the missing values

In [29]:

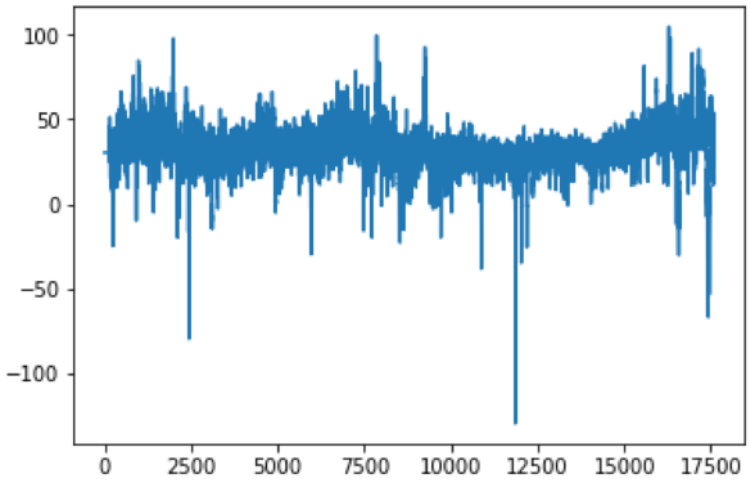
```
master_data['price_germany_euro/mwh'].fillna(master_data['price_germany_euro/mwh']
                                             .mean(),inplace=True)
```

In [30]:

```
master_data['price_germany_euro/mwh'].plot()
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xa94cf28>
```



From the above figure we can observe a pattern with a lot of activity at start and end of the year. As the year proceeds, it appears more constant. The pattern cannot be reproduced the next year 100% but the tendency and value ranges remain the same. Moreover, pricing is a complex issue which not only depends on demand and supply but other factors which we haven't considered in our case. One factor could be the increase of investment every year in renewable energies increasing output but also at the same time making renewable energy expensive in the initial phases.

Since the machine learning models cannot handle timeseries data, we will delete time related data like 'date' and 'time_of_day', however perserving the order of recorded data as the order of data is essential for training a supervised regression model for time series.

Our final list of features can be seen below.

In [32]:

```
list(master_data)
```

Out[32]:

```
['biomass_mwh',
 'hydropower_mwh',
 'wind_offshore_mwh',
 'wind_onshore_mwh',
 'photovoltaics_mwh',
 'other_renewable_mwh',
 'nuclear_mwh',
 'fossil_brown_coal_mwh',
 'fossil_hard_coal_mwh',
 'fossil_gas_mwh',
 'hydro_pumped_storage_mwh',
 'other_conventional_mwh',
 'total_consumption_mwh',
 'balancing_energy_volume_mwh',
 'balancing_energy_price_euro/mwh',
 'price_germany_euro/mwh']
```

2. Linear correlation among input features and target variable

In the scatter plots below, we attempt to find linear correlations between the input features and the target variable.

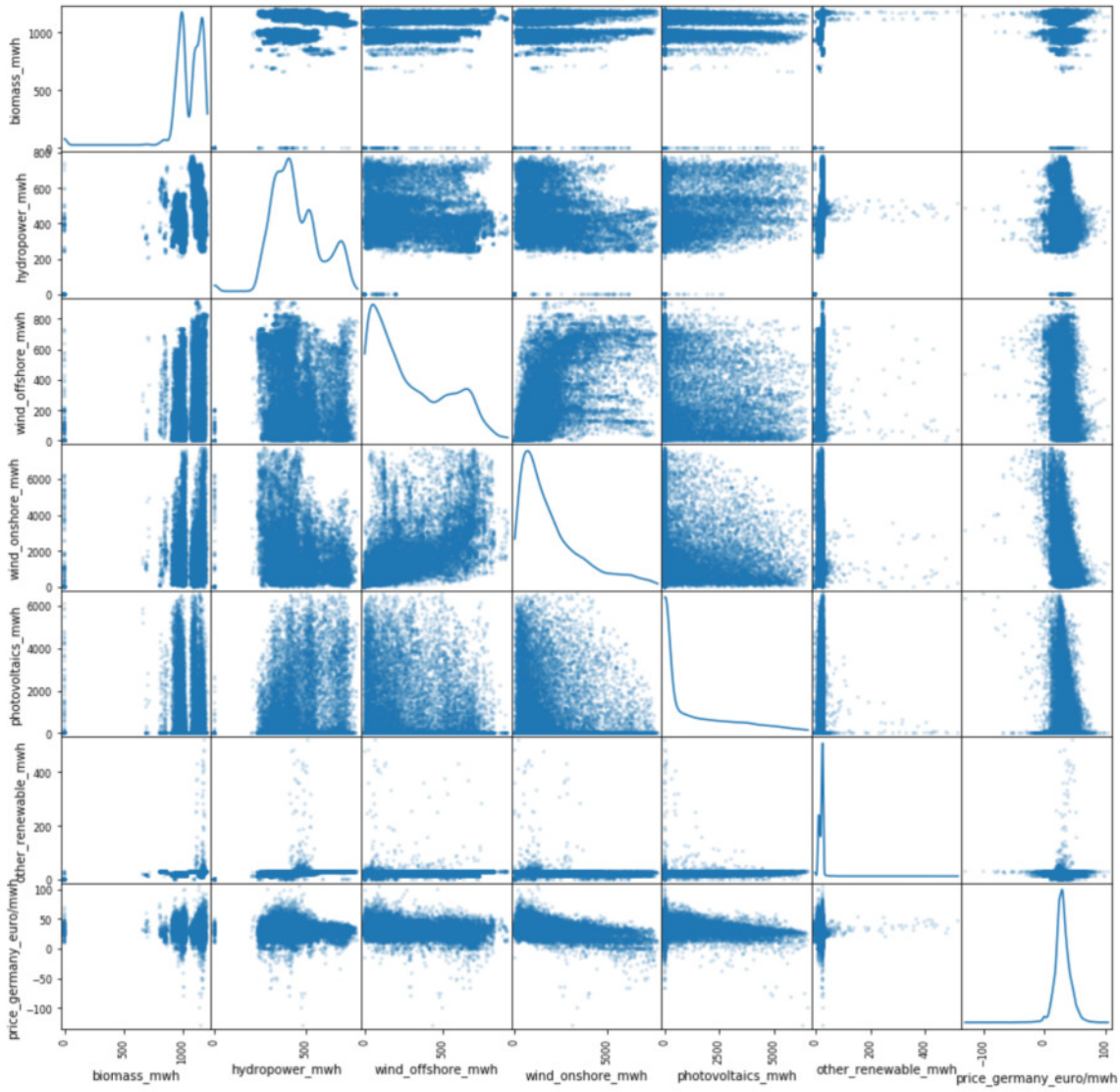
In [33]:

```
from pandas.plotting import scatter_matrix
scatter_matrix(master_data[['biomass_mwh','hydropower_mwh','wind_offshore_mwh',
                            'wind_onshore_mwh','photovoltaics_mwh','other_renewable_mw
h'
                            ,'price_germany_euro/mwh']], alpha=0.2, figsize=(15,15), di
agonal='kde')
```

Out[33]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000000000ACE81D
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000AD1894
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000AD3EF2
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000AD6B5F
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000AD92C8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000AD92CC
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000ADEC9E
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000AE1F0B
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000AE4674
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000AE6FDD
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B1804A
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B1A9B3
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B1DB20
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B20489
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000B22AF2
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B25D5F
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B285C8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B2B535
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B2DE9E
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B3100B
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B33774
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000B3D1DD
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B4044A
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B42AB3
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000C56F20
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000C70389
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000C72EF2
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CBFF5F
8>],
```

```
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000CC26C8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CCB935
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CCE09E
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CD120B
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D22974
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D252DD
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D2854A
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000D2ADB3
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D2DF20
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D30889
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D37FF2
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D3AF5F
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D41AC8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D44B35
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000D4729E
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D4A40B
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D4CE74
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D4F3DD
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D5264A
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D54FB3
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000D58220
8>]],
      dtype=object)
```


biomass_mwh
hydropower_mwh
wind_offshore_mwh
wind_onshore_mwh
photovoltaics_mwh
other_renewable_mwh
price_germany_euro/mwh

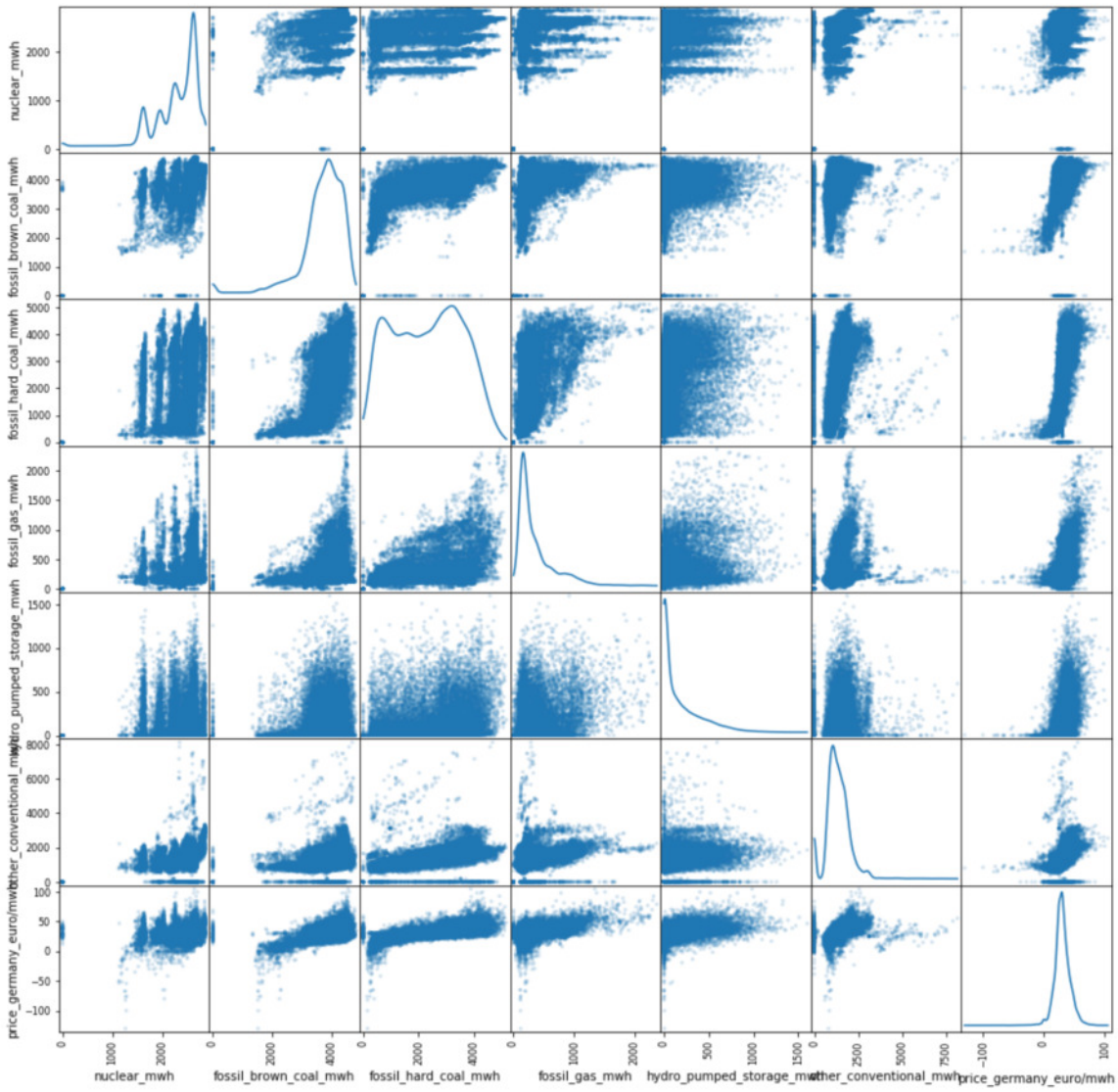
In [34]:

```
scatter_matrix(master_data[['nuclear_mwh','fossil_brown_coal_mwh','fossil_hard_coal_mw
h',
                      'fossil_gas_mwh','hydro_pumped_storage_mwh', 'other_conventional_m
wh',
                             'price_germany_euro/mwh']], alpha=0.2, figsize=(15.5,15.5),
 diagonal='kde')
```

Out[34]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000000000CAD04E
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CABB5C
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CAF4A9
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CB180F
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CB3F78
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CB3F7B
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CB984E
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000DA6EB3
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000DAA120
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000DAC989
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000DAF3F2
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000DB235F
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000DB4CC8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000DFAE35
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000DFD59E
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E0070B
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E02F74
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E058DD
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E08A4A
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E0B0B3
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E52620
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000E54A89
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E574F2
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E5A75F
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E5CFC8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E60035
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E62C9E
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E65C0B
8>],
```

```
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000E68374
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E6AADD
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E8FC4A
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E925B3
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E95520
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000ECAE89
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000ECD5F2
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000ED085F
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000ED32C8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000ED6335
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000ED8A9E
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000EDBB0B
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000EDE374
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000EE0BDD
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000EE404A
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000FE67B3
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000FE9820
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000FEC189
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000FEE9F2
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000FF1A5F
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000FF44C8
8>]],
      dtype=object)
```

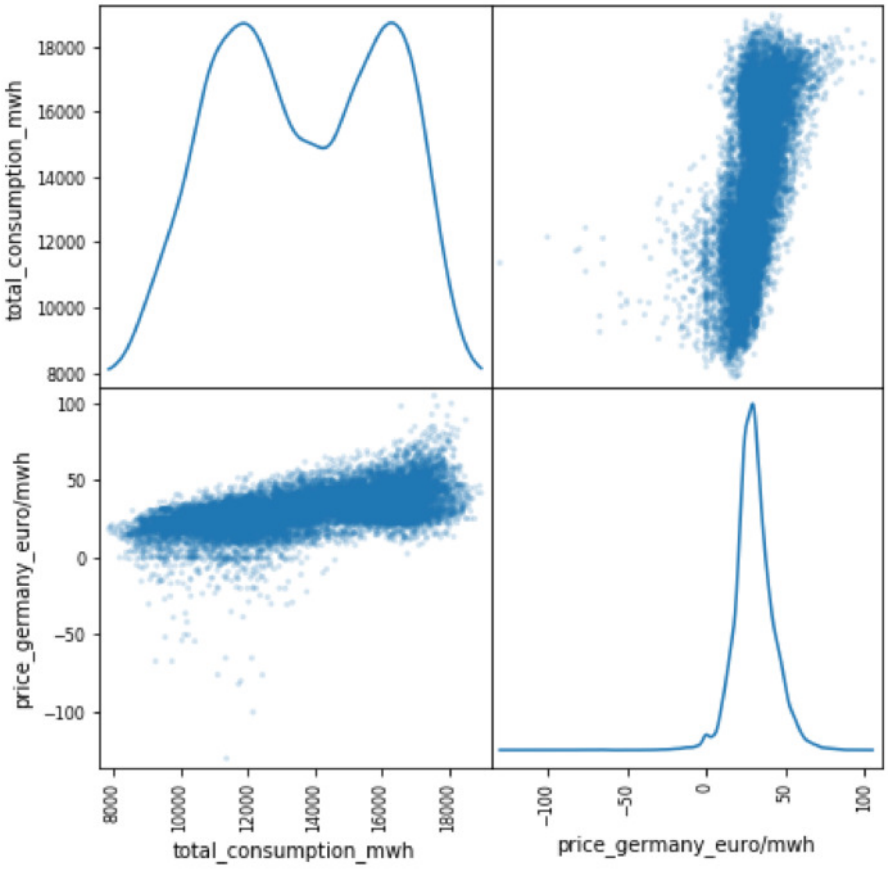

nuclear_mwh
fossil_brown_coal_mwh
fossil_hard_coal_mwh
fossil_gas_mwh
hydro_pumped_storage_mwh
other_conventional_mwh
price_germany_euro/mwh

In [35]:

```
scatter_matrix(master_data[['total_consumption_mwh','price_germany_euro/mwh']],
               alpha=0.2, figsize=(7,7), diagonal='kde')
```

Out[35]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000000106F586
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000000107BF86
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000000010B2F7B
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000010B50AC
8>]],
      dtype=object)
```

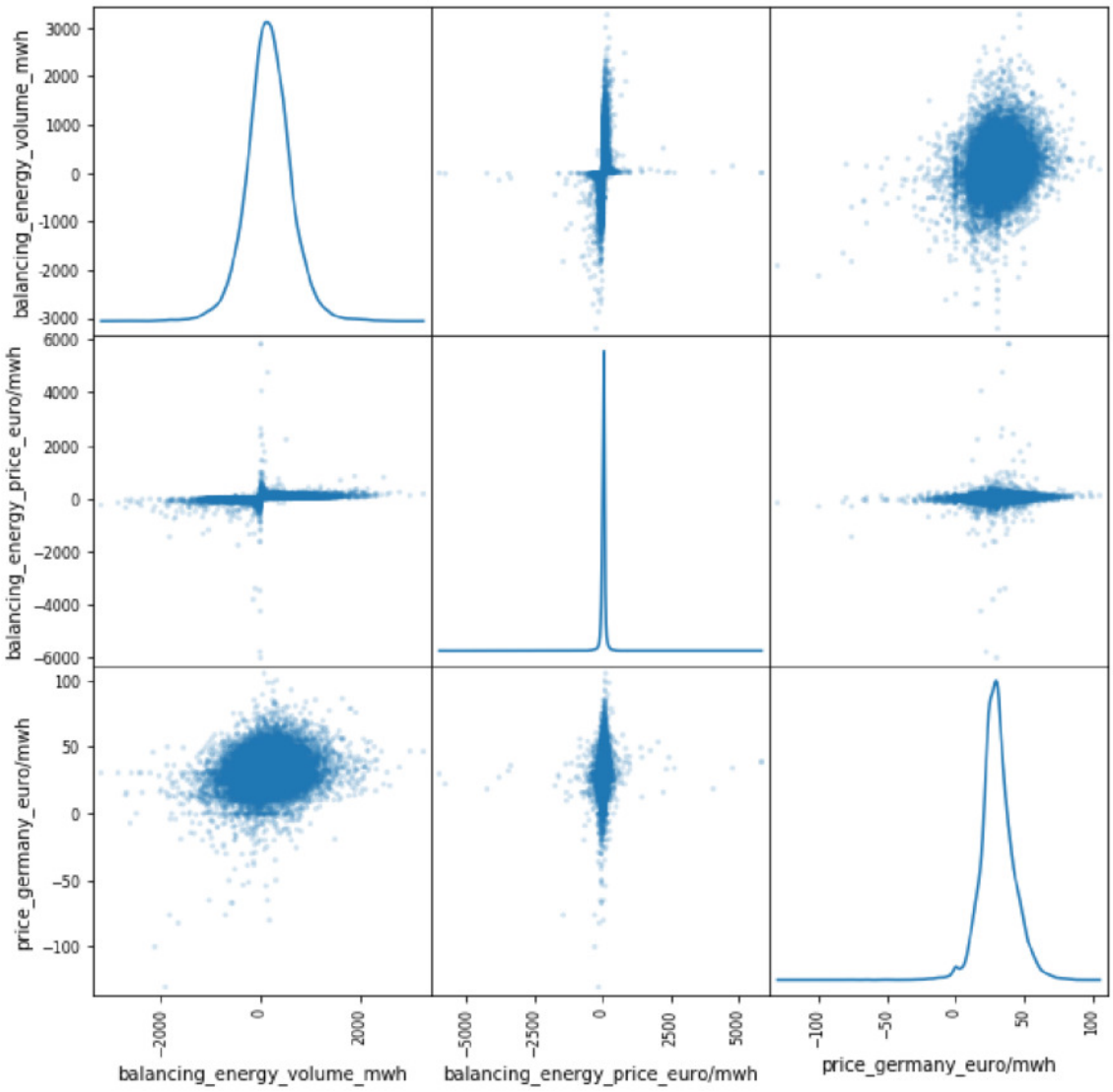


In [36]:

```
scatter_matrix(master_data[['balancing_energy_volume_mwh','balancing_energy_price_euro/
mwh',
                            'price_germany_euro/mwh']], alpha=0.2, figsize=(10,10), dia
gonal='kde')
```

Out[36]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000000010BCBB7
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000010CB67B
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000010CDEE1
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000000010D104E
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000010D38B7
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000010D38BA
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000000010D918D
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000010DBBF6
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000010DEA63
0>]],
      dtype=object)
```



As we can observe we cannot identify a direct linear correlation between the input features and the target variable.

c. Algorithms and Techniques

For the Regression problem we have considered Linear Regression as our benchmark for analysis. The benchmark model will serve as a basis for comparing results with other models. During the preprocessing phase, certain non relevant features were dropped, outliers removed and data was scaled. With Linear Regression as our benchmark, we have chosen the most simple model for a supervised learning regression problem. Supervised Learning offers óther regression models, each of them conceptionally different and complex when compared to Linear Regression. So our approach here is to try different Regression models on the problem at hand to find the best model which fits our data. Once this model is identified, we will optimize this model later.

Regularized versions of Linear Regression

Ridge and Lasso Regressions improve on simple Linear Regression to make sure the data doesn't overfit.

1) Ridge

Ridge Regression performs L2 Regularization i.e. the model penalizes error by adding the sum of the squares of the coefficients to the error. Ridge Regression is used to reduce multicollinearity in multiple regression data. Multicollinearity is the existence of near-linear relationships among independent variables.

The cost function for a Ridge Regression [12] can be denoted by:

$$min(||Y - X(\theta)||_2^2 + \lambda||\theta||_2^2)$$

lambda is used to control the penalty term. The bigger the alpha value, the more we penalize the model and the magnitude of the coefficients are reduced.

Ridge Regression can be used:

- to reduce multicollinearity
- to reduce model complexity by shrinking the coefficients
- on data that are equally distributed
- when no feature selection is necessary
- for operations which require computational eficiency

2) Lasso

Lasso Regression performs L1 Regularization i.e. the model penalizes error by adding the absolute values of the coefficients to the error. Mathematically, Lasso is similar Ridge regression. Instead of adding the squares of theta, we will add the absolute value of theta.

The cost function for a Lasso Regression [12] can be denoted by:

$$min(||Y - X(\theta)||_2^2 + \lambda||\theta||_1)$$

Lasso Regression can be used:

- only for sparse data which require computational efficiency
- feature selection. All non relevant columns are converted directly to noise

Ensemble models

Ensemble models refer to different models which are put together to achieve a better results compared to traditional models. Therer are 2 different strategies used for ensemble methods:

- Bagging: Predictions are made by averaging the results of different estimators which execute in parallel. In Bagging, a certain number of estimators are built using small samples taken from the data (using sampling with replacement). Finally the results are pooled and the estimates are obtained by averaging the results. The algorithms to be used for training can be chosen. Here ist is recommended to use weak learners or simply an algorithm that performs poorly i.e. just above the baseline for the current problem. Weak learners can be trained more faster than other complex algorithms. Though weak in prediction, when combined, they usually achieve comparable or better performance than more sophisticated single algorithms[13].

- Boosting: Predictions are made by using a weighted average of sequential aggregated estimators. A sequence of weak learners(for example, single level decision trees) are fit on reweighted versions of the data. Weights are assigned based on the predictability of the case. Cases that are more difficult are weighted more. The idea is that the trees first learn easy examples and then concentrate on the more difficult ones. In the end, the sequence of weak learners is weighted to maximize overall performance[13].

3) Random Forest Regression

Random Forests are based on bagging, but operates only using binary split decision trees, which are left to grow to their extremes. Moreover, it samples the cases to be used in each of its models using bootstrapping. As the tree is grown, at each split of a branch, the set of variables to be considered for the split is drawn randomly,too. Ensemble trees, that due to different samples and considered variables at splits are very different than each other. Being different they are also uncorrelated. This is beneficial because when the results are ensembled, much variance is ruled out as in a mean extreme values on both sides in a distribution tend to balance each other. In other words, bagging algorithms such as Random Forest guarantee a certain level of diversity in the predictions, allowing for developing rules that a single learner (such as decision tree) might never come across[13].

4) Gradient Boosting Regression

Gradient Boosting is an improved version of boosting based on a gradient descent function. Gradient Boosting is nore sensible to noise and is computationally expensive due to the nonparallel operations.
Gradient Boosted Decision Tree

5) XGBoost Regression

XGBoost stands for eXtreme Gradient Boosting and is a new generation Gradient Boosting Algorithm. XGBoost has the following advantages[13]:

- sparse-aware algorithm: it can leverage sparse matrices, saving both memory and computation time
- approximate tree learning which bears similar results but in much less time than the classical complete explorations of possible branch cuts
- parallel computing on a single machine (using multi threading in the phase of search for best split) and similarly distributed computations on multiple ones
- out-of-core computations on a single machine leveraging a data storage solution called column block. This arranges data on disk by columns, thus saving time by pulling data from the disk as the optimization algorithm (which works on column vectors) expects it.
- XGBoost can also deal with missing data in an effective way. Ensembles based on standard decision trees require missing data first to be imputed using an off-scale value, such as a negative number, in order to develop an appropriate branching of the tree to deal with missing values.

XGBoost instead fits all the non-missing values. After having created the branching for the variable, it decides which branch is better for the missing values to take in order to minimize the prediction error. Such an approach leads to both trees that are more compact and an effective im putation starategy, leading to more predictive power.

Tree Based Gradient Boosting

6) LightGBM Regression

XGBoost has many advantages over classical ensemble methods but at the same time takes a long time to train. This is what motivated an open source team at Microsoft to create a high performance Gradient Boosting algorithm called LightGBM. Though LightGBM is based on decision trees like XGBoost, it follows a different strategy. Whereas XGBoost uses decision trees to split on a variable and exploring different cuts at that variable, LightGBM concentrates on a split and goes on splitting from there in order to achieve a better fitting (leaf wise tree growth strategy). This allows LightGBM to fit frist and fast a good fit of the data, and to generate alternative solutions compared to XGBoost[13].

Some Highlights of LightGBM are shown below:

- More complex trees due to a leaf wise strategy which increases prediction but at the same time with a higher risk of overfitting
- Faster on larger datasets
- Can leverage parallelization and GPU and can therfore be scaled on even larger problems
- LighGBM is not heavy on memory as it doesnt store and handles continous variables as they are, but it turns them into discrete bins of values.

Benchmark Model

Since energy price prediction is a classic regression problem, we will start with Linear Regression as our benchmark model.

We observe the following results for the Benchmark model:
a. RMSE Score = 4.953005879681328
b. Train Prediction Accuracy = 75.82%
c. Test Prediction Accuracy = 42.43%
d. Execution time: 0.77 seconds

III. Methodology

a. Data Preprocessing

For data preparation / Preprocessing we follow the following steps:

Data Structuring

1. Features which have data entries every 15 minutes (Actual generation, actual consumption, balancing energy) need to be reduced to a data entry every 60 minutes (hourly)
2. Features from 1 will be merged with other features (wholesale market price) which already are on an hourly basis
3. Drop non relevant features
4. Simplify feature names for better readability
5. Drop date and time_of_day time series data as it cannot be handled by machine learning algorithms

Dealing with NaNs

Energy generation data with NaNs imply that there was no energy produced for a specified time period. These entries can be filled with zeros

Target variable price_germany with NaNs should be treated differently compared to energy generation data. Here we will replace NaNs with mean / median as we will see later

Outlier Detection and Elimination

Using Tukey's method of Outlier detection [1], we look for datasets which are 1.5 times below the 1st quartile (25th percentile) and 1.5 times above the 3rd quartile (75th percentile) and delete them from the master dataset.
Tukey's method detects 5326 outliers, which will be removed.

Feature Scaling

From our data analysis we observed that each of the features are on a different scale. There are significant differences in the mean values of each of the individual features. We cannot use this data to train the machine learning model without scaling them. In our case, we will use the MinMaxScaler to scale all the features as some features also include negative values. The values will be scaled between -1 and 1 [5].

Avoiding Look ahead bias and cross validation for the energy Time Series data set

The available data set is a time series data set on an hourly basis. Regular cross validation methods like kfold, holdout are not appropriate due to the look ahead bias they generate. Considering the current time 't', we are predicting data for a future time period 't + n', where x is the difference in time to reach the target time for our prediction. Here it is important to not use future data for out training set. We need a special function 'TimeSeriesSplit' to generate training and testing data sets without look forward bias. 'TimeSeriesSplit' ensures that the future data is always used as the test set for prediction to simulate learning under real conditions.

b. Implementation

I have implemented the model as specified below:

1. Save a list of estimators as input using variable *estimators*
2. _run*pipeline* function to accept estimator list *estimators* and execute each of the estimators and get a list of performance metrics
3. Create dataframe using the performance metrics obtained for easy visualization

All the Regression models specified under 'Techniques and Algorithms' were used:

1. Ridge Regression
2. Lasso Regression
3. Random Forest Regressor
4. Gradient Boosting Regression
5. eXtreme Gradient Boosting Regression
6. LightGBM Regression

Performance Metric: Root Mean Squared Error (RMSE)

Challenges faced during Preprocessing and Implementation

- From the above representations we have features which are available to us in different time intervals. Some features are recorded every 15 minutes and others every 60 minutes. For analysis, the basis for time frequency needs to be the same across all the features. A 15 minute frequency is not possible as we already have data missing for features which have been recorded on an hourly basis. Hence, we need to reduce the features with a 15 minute frequency to an hourly basis i.e. 60 minutes
- During the process of reducing features, feature names become unreadable when the context is missing. These features need to be renamed to something more understandable
- Some Regression techniques like Light GBM Regressor encounter processing problems when square brackets are present in feature names
- The concept of backtesting using TimeSeriesSplit was new to me and for the current problem very important, which caused a bit of a challenge to implement.
- Determining the optimal number of splits for TimeSeriesSplit is time consuming and cumbersome process which is currently only possible via trial and error

c. Refinement

The hyper parameters of LGBM Regressor can now be tuned using GridSearch CV for better fitting leading to higher accuracy of prediction.

Accuracy: 67.91%
Error rate (RMSE): 4.36

For tuning the model, I considered the following parameters:

i. num_leaves: maximum number of leaves in a tree (default=31)
ii. min_data_inleaf: minimum number of examples for a leaf to be created
iii. max*depth* maximum depth that a tree can reach

The parameter values are as follows:

- num_leaves: 50
- max_depth: 30
- min_data_inleaf: 60
- min_ data: 1
- min_data_in_bin: 1

During execution of GridSearchCV with TimeSeriesSplit we encountered a problem with LightGBM (LightGBMError : Cannot construct Dataset since there are not useful features) due to the nature of the different input features. The actual_generation data which contains generation data of different energy sources is to a certain extent correlated. Since in our case all sources of energy generation are relevant, none of the features can be excluded. As a workaround for the error, the parameters _min*data* and _min_data_in*bin* have been set to 1.

Prediction rate before tuning: 67.91%
Prediction rate after tuning: 68.22%

Therefore, we have a gain of **0.31%**

In [46]:

```
# Construct dataframe from dictionary list
model_summary = pd.DataFrame.from_dict(perf_list)
model_summary.rename(index={0:'Ridge',1:'Lasso', 2:'Random Forest',
                            3:'SVR', 4:'Gradient Boosting', 5:'LGBM',
                            6:'XGBoost'}, inplace=True)
model_summary
```

Out[46]:

	name	rmse	test_score	train_score	train_time
Ridge	Ridge	4.976745	0.511779	0.758339	0.268
Lasso	Lasso	5.529965	-0.008370	0.292662	0.260
Random Forest	RandomForestRegressor	5.401900	0.596863	0.972212	39.046
SVR	SVR	5.387532	0.486393	0.701419	259.357
Gradient Boosting	GradientBoostingRegressor	5.260943	0.649664	0.883455	40.815
LGBM	LGBMRegressor	5.150525	0.679183	0.953913	20.458
XGBoost	XGBRegressor	5.070604	0.655381	0.882269	33.593

IV. Results

a. Model Evaluation and Validation

Our final model is the LightLBM Regressor with a test pridiction accuracy of **68.22%** after tuning. We initially used the default parameters to get an accuracy of 67.91%.

After hyper paramter tuning we have the following features:

- num_leaves: 50
- max_depth: 30
- min_data_inleaf: 60
- min_ data: 1
- min_data_in_bin: 1

Robustness of the model

To check the robustness of the model,it is best advised to use a cross validation mechanism like k-fold cross validation. Using k-fold cross validation it is possible to check if the model can generalize for unseen data.

The general procedure for k-fold cross validation is as follows:

1. Split data into k groups
2. Select any group as test data. All other groups will be considered as train data
3. Fit the model using train data selected and evaluate on test set
4. Summarize model results
5. Repeat step 1 with a new group and continue until all groups have been exhausted.

For our dataset, since we are dealing with time series data, it is not possible to directly use k-fold cross validation because of the look ahead bias we would create. Hence, I have used a modified version of k-fold for time series data calles TimeSeriesSplit in sklearn. TimeSeriesSplit has been implemented for the dataset starting with the benchmark model and further on for the complete pipeline to have comparable results.

From [14], TimeSeriesSplit is a variation of k-fold which returns first k folds as train set and the (k+1)th fold as test set. Note that unlike standard cross-validation methods, successive training sets are supersets of those that come before them. Also, it adds all surplus data to the first training partition, which is always used to train the model.

Therefore, the final model is robust, generalizes to unseen data and can be trusted.

b. Justification

Perf. Metrics	Final Model	Benchmark Model
Train score	95.39%	75.82%
Test score	68.22%	42.43%
RMSE	5.15	4.95

We clearly see, that the final model has improved on the results of the benchmark model. Therefore, the results of the final model are satisfactory/good.

V. Conclusion

a. Free-Form Visualization

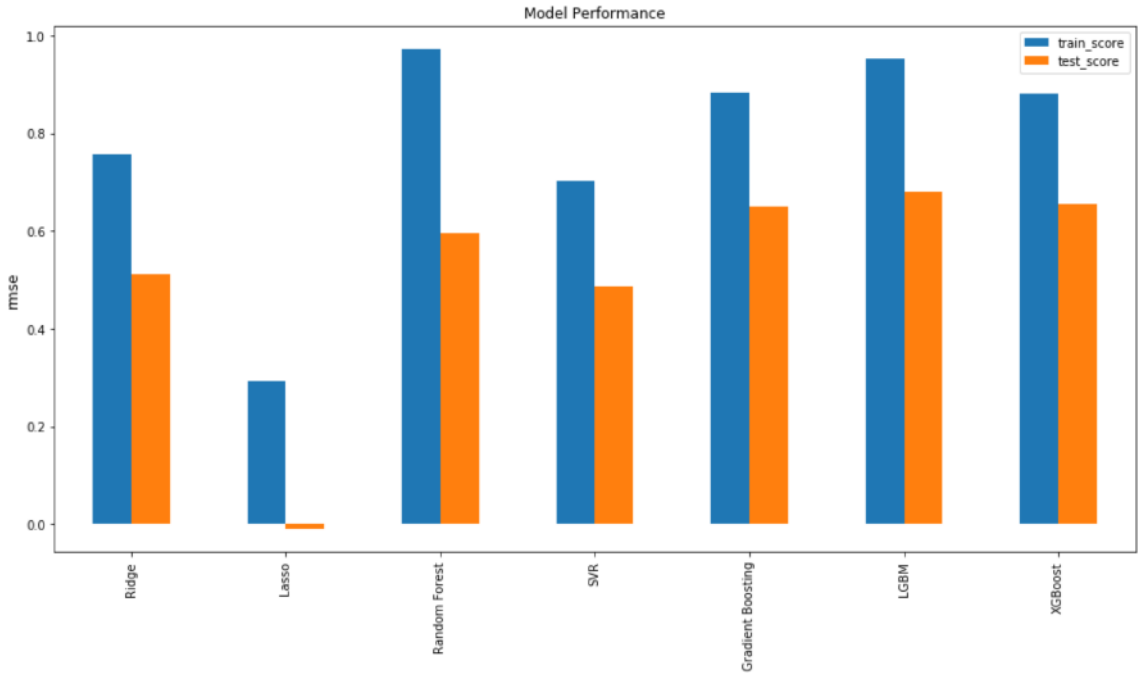
Below we have compared the train and test scores of all our models we used in our pipeline.

In [47]:

```
# Plot train and test scores
axis = model_summary[["train_score", "test_score"]].plot(kind="bar",
                    title="Model Performance", figsize=(16, 8))
axis.set_ylabel("rmse", fontsize="large")
```

Out[47]:

```
Text(0,0.5,'rmse')
```



The worst model for test accuracy is Lasso Regression. This is surprising as Lasso Regression performs L1 regularization on a Linear Regression. The Linear Regression model which is the benchmark model actually outperforms Lasso Regression. The presence of collinearity in the data could explain why Ridge Regression has performed better than the benchmark model. Random Forest performs excellent when it comes training data but average for test data. Gradient Boosting algorithms GBM, LightGBM and XGBoost show solid performances and are similar. However LightGBM performs better resulting in the best model.

b. Reflection

I can summarize my thought processs for the project in the following steps:

1. Looking for a machine learning problem to solve
2. Formulating a proposal for the problem based on the dataset found
3. Understanding the energy market business to understand the data
4. Analyzing data and preparing a preprocessing strategy
5. Visualizing the data
6. Deciding on which algorithms to use to solve the problem and error metrics
7. Creating a benchmark model
8. Running the regression model pipeline and comparing results
9. Hyper parameter tuning for the final model
10. Check for robustness and summarize results

Interesting aspects:
I found the search for the machine learning problem very interesting since I was important to me that the problem needs to be a real world problem to test all my knowledge acquired in the Nanodegreee. Moreover, it is also my motivation to use machine learning to solve problems in my professional field which is Electromobility. Hence, the energy market comes pretty close. It was a very satisfying experience creating the benchmark model and running the regression pipeline to know which model actually solves the problem at hand. This was very interesting due to the practical hands on comparision I achieved which beats any theory lesson I've had until now on machine learning.

Difficult aspects:
I've listed the difficult aspects of the project below.

1. Handling different data with varying time frewquencies - Some Features had time series data every 15 minutes and others every hour. The first difficulty was to find a strategy how to deal with this dataset as it was important to have a data set with a single time frequency.
2. My expectation for the data to be uniform for better predictability was reduced as soon as I visualized my data. The data showed more outliers than I expected. But then this was real world data and such data is much more complex as there are many factors which are not apparently visible and one has to handle this data to the best of one's knowledge and abilities.
3. Another difficult aspect was the dealing with a new concept for time series data and avoiding look ahead bias. TimeSeriesSplit cv was the solution which was not easy implementing. I did some research on regression for time series data and surprisingly very few projects used TimeSeriesSplit for time series cross validation.

c. Improvement

Based on the results obtained from real world energy data, an accuracy of 68.22% is a good result when we take into account the factors that we haven't accounted for in our problem.

Improvement 1
We have only considered the energy produced and consumed inside Germany. Germany also exports and imports energy from its neighbours. These factors have not been accounted for in this dataset. If the exports or import needs have changed between 2015 and 2016, they have not been considered.

Improvement 2
Some of the energy generation features are dependent on weather, for example wind or water. We have to calculate certain amount of loss with these features.

Improvement 3
The information available does not account for the local energy distribution within Germany and the network structures. If a certain amount of energy is available at a certain point of time, this does not guarantee that the energy can be supplied throughout Germany when demand arises.

Improvement 4
The final model LightGBM has many parameters which can be explored for model optimization. An exhaustive parameter tuning could improve prediction results further

Improvement 5
I haven't yet explored the possibility of using deep learning for my problem. There is a good chance that deep learning could outperform our final model.

For any given market, supply and demand define the price. The energy data that was analyzed shows the complexity of an energy market and multiple dependencies that determine prices. Therefore a greater

References

[1] http://colingorrie.github.io/outlier-detection.html (http://colingorrie.github.io/outlier-detection.html) (Tukey's Method for outlier detection)
[2] https://medium.com/apteo/avoid-time-loops-with-cross-validation-aa595318543e (https://medium.com/apteo/avoid-time-loops-with-cross-validation-aa595318543e) (Look ahead bias)
[3] https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/ (https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/) (XGBoost)
[4] https://smard.de (https://smard.de)
[5] http://benalexkeen.com/feature-scaling-with-scikit-learn/ (http://benalexkeen.com/feature-scaling-with-scikit-learn/) (MinMaxScaler)
[6] A Comparative Study of Different Machine LearningMethods for Electricity Prices Forecasting of an Electricity Market (https://www.researchgate.net/publication/282250672_A_Comparative_Study_of_Different_Machine_Learning
[7] Forecasting day ahead electricity prices in Europe (https://ac.els-cdn.com/S0306261917316999/1-s2.0-S0306261917316999-main.pdf?_tid=163a3320-4362-45ca-b979-300481ec5d1c&acdnat=1540837518_1c64917ccea34a96720342f96cfdaf08)
[8] https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d (https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d)
[9] https://en.wikipedia.org/wiki/Coefficient_of_determination (https://en.wikipedia.org/wiki/Coefficient_of_determination)
[10] http://blog.minitab.com/blog/adventures-in-statistics-2/multiple-regession-analysis-use-adjusted-r-squared-and-predicted-r-squared-to-include-the-correct-number-of-variables (http://blog.minitab.com/blog/adventures-in-statistics-2/multiple-regession-analysis-use-adjusted-r-squared-and-predicted-r-squared-to-include-the-correct-number-of-variables) (R squared and adjusted r squared)
[11] https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf (https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf) (Ridge Regression)
[12] https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/ (https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/) (Ridge Regression and Lasso Regression)
[13] Boschetti,A and Massaron,L. (2018). Python Data Science Essentials Third edition, Birmingham: Packt Publishing [14] https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation (https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation) (TimeSeriesSplit)