Aalto University
CS-C3240 - Machine Learning D

Predicting price of electricity with machine learning methods

October 10, 2023

Contents

1 Introduction

In the same way there are markets for exchanging services and goods needed in our daily life, there also exists a broad market for buying and selling electricity. Again similarly than for more traditional commodities, the price of electricity is closely followed by many, since it can have large effects on our daily choices and lifestyle. Large variations in price can have negative impacts on operations of both small households as well as large corporations. The consumption of electricity may not be a matter of choice in our technology driven society. These markets differ from many others, since the supply must always meet the demand. This, with other factors, can lead to considerable fluctuations in price.

The purpose of this paper is to use machine learning methods to predict the price of electricity with certain real life factors or variables. Even though this topic is not new in general and research has probably already been done on it, this project stands out by focusing only on price in Finland. Possible application domains for this project and its results coud be e.g. defining optimal time to consume electricity or optimizing the profits for electricity suppliers.

Section 2 discusses the formulation of this problem, introducing the dataset used. Section 3 then further discusses the details of the dataset, data preprocessing and the Machine Learning models used in this project. In section 4 the results of the models are discussed and compared, and finally in section 5 the results are summarized and the quality of the model is assessed.

2 Problem formulation

The focus of this project is the price of electricity in Finland in 2022. The objective is to predict the price of electricity at specific hour of the day with different real life factors.

The first step is to find out which factors affect the price of electricity. According to several sources, the price is influenced by number of factors, such as supply and demand, fuel prices, production of renewable energy, weather conditions and several others [3][4][5].

Data availability is a key criterion for the feature values, as well as relevance and suitability for this ML model. Based on this, the seven feature values selected are *Precipitation*, *Air temperature*, *Wind speed*, *Electricity production*, *Wind power production*, *Electricity consumption* and *Hour* of the day. The data is continuous, except for the feature *Hour*, which is integer valued between 0 - 23.

Precipitation is measured in millimeters (mm), *Air temperature* in degrees Celsius (degC) and *Wind speed* in meters per second (m/s). The values have been recorded at the beginning of each hour. The weather data is from the Finnish Meteorological Institute [1] and the observation station used is Helsinki-Vantaa airport.

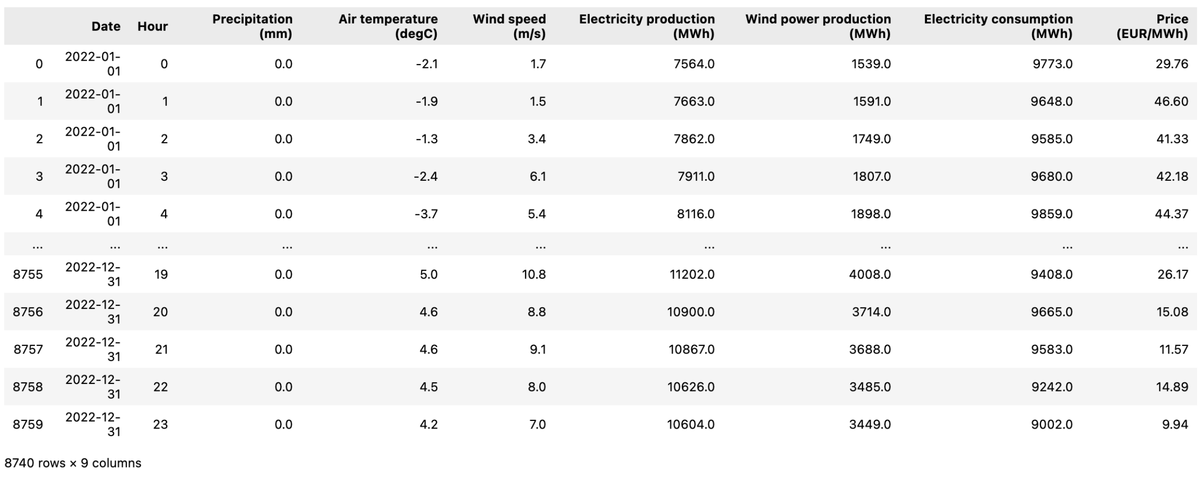
Electricity production, *Wind power production* and *Electricity consumption* are all measured in megawatt hours (MWh), and they represent the production/ consumption during each hour in Finland. The electricity data is from Finland's transmission system operator Fingrid [2].

The label value is the price of electricity, measured in EUR/MWh, for each hour. The price data is from ENTSO-E Transparency Platform [6], which is maintained by European transmission system operators. As the data has clear label values, it is natural to use supervised learning methods.

3 Methods

The original data has 8760 data points, where one point consists of seven measurements, one for each feature. One data point corresponds to one hour of the year 2022. Since some data is missing on 20 rows, these rows of the data frame have to be deleted in order to be able to use the LinearRegression -function. The data is also parsed into suitable form. As the range of values varies between features, the data is normalized. After preprocessing there are 8740 data points.

After doing research on the factors that affect the price of electricity, and searching corresponding data, the following feature values were selected: *Precipitation*, *Air temperature*, *Wind speed*, *Electricity production*, *Wind power production*, *Electricity consumption* and *Hour* of the day. These features were selected, since the data was free and available for Finland, as well as measured at frequent enough intervals.

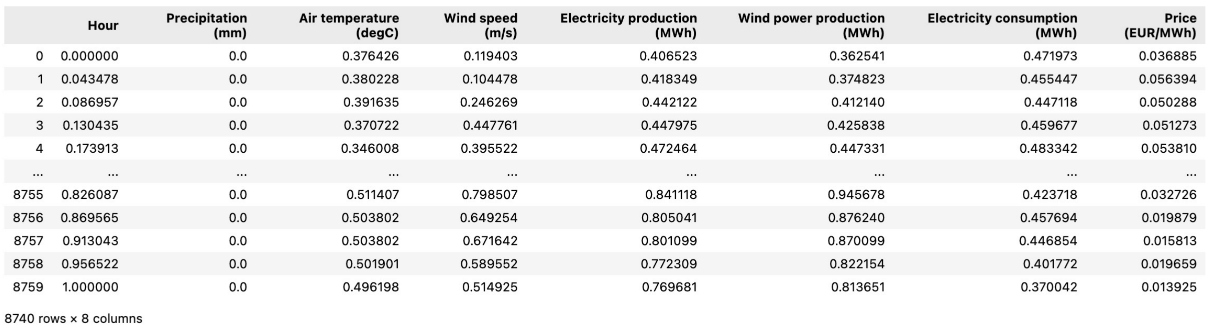


	Date	Hour	Precipitation (mm)	Air temperature (degC)	Wind speed (m/s)	Electricity production (MWh)	Wind power production (MWh)	Electricity consumption (MWh)	Price (EUR/MWh)
0	2022-01-01	0	0.0	-2.1	1.7	7564.0	1539.0	9773.0	29.76
1	2022-01-01	1	0.0	-1.9	1.5	7663.0	1591.0	9648.0	46.60
2	2022-01-01	2	0.0	-1.3	3.4	7862.0	1749.0	9585.0	41.33
3	2022-01-01	3	0.0	-2.4	6.1	7911.0	1807.0	9680.0	42.18
4	2022-01-01	4	0.0	-3.7	5.4	8116.0	1898.0	9859.0	44.37
...	...	...	...	...	...	...	...	...	...
8755	2022-12-31	19	0.0	5.0	10.8	11202.0	4008.0	9408.0	26.17
8756	2022-12-31	20	0.0	4.6	8.8	10900.0	3714.0	9665.0	15.08
8757	2022-12-31	21	0.0	4.6	9.1	10867.0	3688.0	9583.0	11.57
8758	2022-12-31	22	0.0	4.5	8.0	10626.0	3485.0	9242.0	14.89
8759	2022-12-31	23	0.0	4.2	7.0	10604.0	3449.0	9002.0	9.94

8740 rows × 9 columns

Figure 1: Dataset used for the model, before normalization.

From Figure 1, we can see the combined dataset used for this project. The *Date* column is not used for the model but rather exists to clarify and illustrate the measurement time of each datapoint. However, we notice that the feature values are of different units and orders of magnitude. To achieve better and more accurate model, the data should be normalized, meaning the features are scaled to a more similar range.



	Hour	Precipitation (mm)	Air temperature (degC)	Wind speed (m/s)	Electricity production (MWh)	Wind power production (MWh)	Electricity consumption (MWh)	Price (EUR/MWh)
0	0.000000	0.0	0.376426	0.119403	0.406523	0.362541	0.471973	0.036885
1	0.043478	0.0	0.380228	0.104478	0.418349	0.374823	0.455447	0.056394
2	0.086957	0.0	0.391635	0.246269	0.442122	0.412140	0.447118	0.050288
3	0.130435	0.0	0.370722	0.447761	0.447975	0.425838	0.459677	0.051273
4	0.173913	0.0	0.346008	0.395522	0.472464	0.447331	0.483342	0.053810
...	...	...	...	...	...	...	...	...
8755	0.826087	0.0	0.511407	0.798507	0.841118	0.945678	0.423718	0.032726
8756	0.869565	0.0	0.503802	0.649254	0.805041	0.876240	0.457694	0.019879
8757	0.913043	0.0	0.503802	0.671642	0.801099	0.870099	0.446854	0.015813
8758	0.956522	0.0	0.501901	0.589552	0.772309	0.822154	0.401772	0.019659
8759	1.000000	0.0	0.496198	0.514925	0.769681	0.813651	0.370042	0.013925

8740 rows × 8 columns

Figure 2: Dataset used for the model, after normalization.

In Figure 2 we can see the data after normalizing the columns by scaling. Scaling is conducted by the following formula:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}, \tag{1}$$

where X is the original column and X_{max} and X_{min} are the maximum and minimum values of the column.

The data is then split into training, test and validation sets using the *train_ test_ split* -function, which splits the data randomly. As the dataset is quite large, a bit smaller proportion can be allocated to the validation and test sets, still having enough data for meaningful evaluation. The sizes of the sets are 80 %, 10 % and 10 %, respectively, so that there is enough data to train the model, but the test and validation sets remain representative.

3.1 Linear Regression

Generally, the usage of linear regression is well suited when predicting continuous data. It can also be a great first step for understanding the relationship between the variables, before jumping to more advanced methods and models [8]. This goes well with the problem formulation of this project, since the values of *Price* are continuous.

The correlations between *Price* and the explaining features can be visualized by plotting a following heatmap:

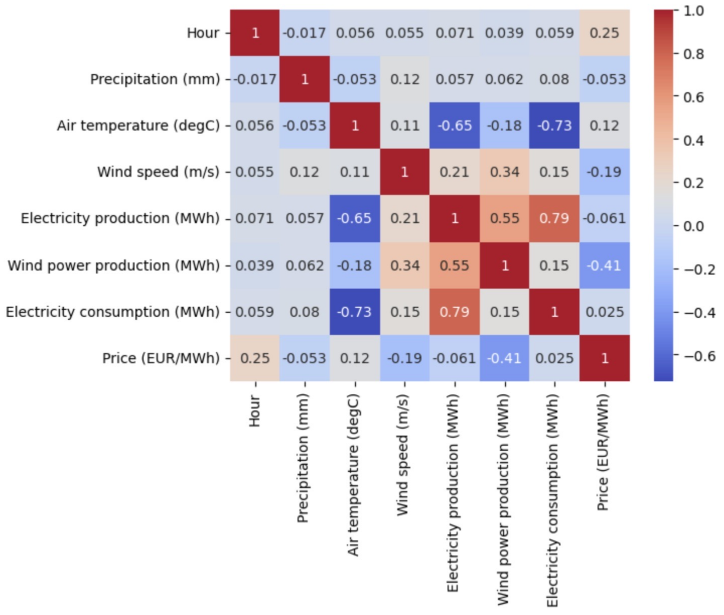


Figure 3: Heatmap of the correlations between variables.

From the heatmap in Figure 4, we can see that there exists at least some correlations between *Price* and the features, meaning they are somewhat linearly related. It correlates the most and second most with the variables *Wind power production* and *Hour* respectively. This further reinforces the use of linear regression as a first method. Some of the features e.g. *Electricity consumption*, *Electricity production* and probably also *Air temperature* seem to have much smaller correlation with *Price* than intuitively expected. This might suggest that still some more data is needed for better prediction.

As is customary with linear regression, the loss function used is mean squared error (MSE):

$$MSE = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{2}$$

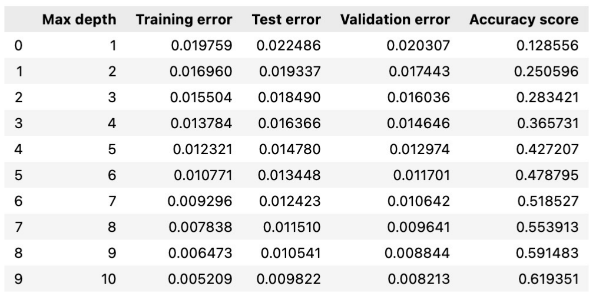
where N is the size of the sample, y_i is the true label value and $\hat{y}_i$ is the predicted label value. MSE is the average of the squared differences between the predicted and the actual values. As larger errors are penalized more heavily, MSE is a good choice for linear regression, since the goal is to minimize the overall prediction error.

3.2 Random Forest Regression

Random Forest Regression is a powerful Machine Learning model for predicting continuous outcomes from high-dimensional inputs [9]. It uses several decision trees and outputs their combined result [10]. As it can be used with data showing both linear and non-linear relationships, it is well suited for this project.

The loss function associated with Random Forest Regression is mean squared error (MSE), which can be seen in Equation 2, and was also used with Linear Regression. MSE measures the average of the squared differences of the actual and predicted values, which aligns well with the objective of minimizing the overall prediction error.

When creating a Random Forest model, it is important to consider the choice for maximum depth of the tree. Choosing the depth to be too large can cause overfitting and choosing it to be too low could lead to a less accurate and less predictive model. The model is fitted for values of maximum depth from 1 to 10, to see which leads to best results:



	Max depth	Training error	Test error	Validation error	Accuracy score
0	1	0.019759	0.022486	0.020307	0.128556
1	2	0.016960	0.019337	0.017443	0.250596
2	3	0.015504	0.018490	0.016036	0.283421
3	4	0.013784	0.016366	0.014646	0.365731
4	5	0.012321	0.014780	0.012974	0.427207
5	6	0.010771	0.013448	0.011701	0.478795
6	7	0.009296	0.012423	0.010642	0.518527
7	8	0.007838	0.011510	0.009641	0.553913
8	9	0.006473	0.010541	0.008844	0.591483
9	10	0.005209	0.009822	0.008213	0.619351

Figure 4: Training, test and validation errors and accuracy score for values of maximum depth.

The errors and accuracy score for different values of maximum depth can be seen in Figure 4. For the final model, the depth value 7 is chosen because here the training and validation errors are both small and quite close to each other. Even though the errors get smaller for larger values of depth, the difference between validation error and training error grows steadily. This might imply that for larger values than 7, the model starts to overfit.

4 Results

Model	training error	test error	validation error	accuracy score
Linear Regression	0.015338	0.017107	0.015793	0.337023
Random Forest	0.009296	0.012423	0.010642	0.518527

Table 1: Results of both models.

The training and validation errors for both the Linear Regression model and the Random Forest Regression model can be seen in Table 1. Both the training and validation errors of the Linear Regression model are larger than those of the Random Forest Regression model. Also the accuracy score is clearly larger for the Random Forest model. The final chosen method is the Random Forest Regression model, as the training and validation errors are smaller and the accuracy is better.

To evaluate the performance of the final model, the test set should be considered. The data in the test set is independent from the training and validation sets. As the model is developed using the training set, and validation set is used to select the appropriate model, the assessment of the model accuracy is done using the test set. The test error of the Random Forest Regression model, and thus the final model, is 0,012423. The test error is computed using the loss function MSE (2), and the maximum depth of the tree was chosen to be 7. The test error seems quite small, which hints that the model performs quite well on the test set. An aspect worth noticing is that the training error is a bit smaller that the test error, but as the difference is quite small, no conclusions about overfitting can be drawn.

5 Conclusions

This project tried to predict electricity prices with supervised machine learning methods. The methods chosen were Linear Regression and Random Forest Regression, from which the latter was chosen as the final model due to its better fit.

The accuracy score of the final model is 0,518527, which implies that the model explains about half of the variation in the label values. Although the errors are quite small, the accuracy of the model could be higher.

The model could be improved by finding more relevant data, and considering the dependencies between the features. Although the features were chosen based on several sources, some more relevant features could be found. There is some room for improvement in several areas of the project. For example the weather data was recorded at Helsinki-Vantaa Airport, although based on our domain knowledge most wind power plants are located on other parts of Finland. As Finland is part of a joint European energy market, the price of electricity depends on the conditions of other countries too. Thus it is natural that considering only data from Finland results in unsatisfactory results.

In conclusion, more research is needed to better determine the features affecting price of electricity. To be able to accurately predict the price of electricity in Finland, the whole energy market should be considered, which would call for a more complex model.

References

[1] https://www.ilmatieteenlaitos.fi/havaintojen-lataus

[2] https://data.fingrid.fi/open-data-forms/search/fi/

[3] https://www.nordpoolgroup.com/en/the-power-market/

[4] EU Energy Markets and Energy Prices - European Union europa.eu https://ec.europa.eu › presscorner › api › files › attachment

[5] https://www.fingridlehti.fi/sahkon-hinta-heilahtelee-nain-sahkon-hinta-maaraytyy/

[6] https://transparency.entsoe.eu/

[7] https://www.geeksforgeeks.org/training-vs-testing-vs-validation-sets/

[8] https://peopleanalytics-regression-book.org/linear-reg-ols.html

[9] https://medium.com/@bhatshrinath41/a-comprehensive-guide-to-random-forest-regression-43da559342bf

[10] https://www.ibm.com/topics/random-forest

Appendix

October 10, 2023

NOTE that the numerical values might differ a little bit from the values discussed in the previous sections. The results of the models can differ between each run of the code.

Relevant libraries are imported.

[165]:
```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from sklearn import preprocessing
```

The model uses electricity production, wind power production and electricity consumption data from Fingrid, weather recordings from the Finnish Meteorological Institute, and electricity price data from ENTSO-E Transparency Platform. The data is downloaded in csv-files and then stored in three pandas dataFrames. The data is then preprocessed, removing unnecessary information and parsing the data to suitable format.

[166]:
```
energy_stats = pd.read_csv("data/fingrid_stat.csv")
weather = pd.read_csv("data/weather.csv")
prices = pd.read_csv("data/Day-ahead-Prices-2022.csv", sep='"""', engine="python")

energy_stats = energy_stats.drop(columns = ["Alkuaika UTC", "Lopetusaika UTC",␣
 ↪"Alkuaika UTC+03:00", "Lopetusaika UTC+03:00"])

weather = weather.rename(columns={"Vuosi":"year", "Kk":"month", "Pv":"day"})
weather["Date"] = pd.to_datetime(weather[["year","month","day"]],␣
 ↪infer_datetime_format=True)
weather = weather.drop(columns = ["year", "month", "day", "Aikavyöhyke"])
weather = weather[["Date", "Klo", "Sademäärä (mm)", "Ilman lämpötila (degC)",␣
 ↪"Tuulen nopeus (m/s)"]]

prices = prices["Day-ahead Price [EUR/MWh]"]
```

The three dataFrames are combined, the data types are transformed to more suitable ones, and rows with missing data are removed.

[167]:
```
data = pd.concat([weather, energy_stats, prices], axis = 1)
data.columns = ["Date", "Hour", "Precipitation (mm)", "Air temperature (degC)",␣
 ↪"Wind speed (m/s)", "Electricity production (MWh)", "Wind power production␣
 ↪(MWh)", "Electricity consumption (MWh)", "Price (EUR/MWh)"]
data.Hour = data.Hour.str.split(":", expand = True)[0]
data["Precipitation (mm)"] = data["Precipitation (mm)"].replace('-', np.nan)
data["Precipitation (mm)"] = data["Precipitation (mm)"].map(float)
data["Wind speed (m/s)"] = data["Wind speed (m/s)"].replace('-', np.nan)
data["Wind speed (m/s)"] = data["Wind speed (m/s)"].map(float)
data["Hour"] = data["Hour"].astype('int')

data = data.dropna(axis=0, how='any')
data
```

[167]:
```
            Date  Hour  Precipitation (mm)  Air temperature (degC)  \
0     2022-01-01     0                 0.0                    -2.1
1     2022-01-01     1                 0.0                    -1.9
2     2022-01-01     2                 0.0                    -1.3
3     2022-01-01     3                 0.0                    -2.4
4     2022-01-01     4                 0.0                    -3.7
...          ...   ...                 ...                     ...
8755  2022-12-31    19                 0.0                     5.0
8756  2022-12-31    20                 0.0                     4.6
8757  2022-12-31    21                 0.0                     4.6
8758  2022-12-31    22                 0.0                     4.5
8759  2022-12-31    23                 0.0                     4.2

      Wind speed (m/s)  Electricity production (MWh)  \
0                  1.7                        7564.0
1                  1.5                        7663.0
2                  3.4                        7862.0
3                  6.1                        7911.0
4                  5.4                        8116.0
...                ...                           ...
8755              10.8                       11202.0
8756               8.8                       10900.0
8757               9.1                       10867.0
8758               8.0                       10626.0
8759               7.0                       10604.0

      Wind power production (MWh)  Electricity consumption (MWh)  \
0                          1539.0                         9773.0
1                          1591.0                         9648.0
2                          1749.0                         9585.0
```

```
3                          1807.0                        9680.0
4                          1898.0                        9859.0
...                           ...                           ...
8755                       4008.0                        9408.0
8756                       3714.0                        9665.0
8757                       3688.0                        9583.0
8758                       3485.0                        9242.0
8759                       3449.0                        9002.0

      Price (EUR/MWh)
0               29.76
1               46.60
2               41.33
3               42.18
4               44.37
...               ...
8755            26.17
8756            15.08
8757            11.57
8758            14.89
8759             9.94

[8740 rows x 9 columns]
```

Columns are normalized by scaling.

```
[168]: data_norm = data.drop(columns=["Date"])
       data_norm = (data_norm - data_norm.min()) / (data_norm.max() - data_norm.min())
       data_norm
```

```
[168]:          Hour  Precipitation (mm)  Air temperature (degC)  Wind speed (m/s)  \
       0     0.000000                 0.0                0.376426          0.119403
       1     0.043478                 0.0                0.380228          0.104478
       2     0.086957                 0.0                0.391635          0.246269
       3     0.130435                 0.0                0.370722          0.447761
       4     0.173913                 0.0                0.346008          0.395522
       ...        ...                 ...                     ...               ...
       8755  0.826087                 0.0                0.511407          0.798507
       8756  0.869565                 0.0                0.503802          0.649254
       8757  0.913043                 0.0                0.503802          0.671642
       8758  0.956522                 0.0                0.501901          0.589552
       8759  1.000000                 0.0                0.496198          0.514925

             Electricity production (MWh)  Wind power production (MWh)  \
       0                         0.406523                     0.362541
       1                         0.418349                     0.374823
       2                         0.442122                     0.412140
       3                         0.447975                     0.425838
```

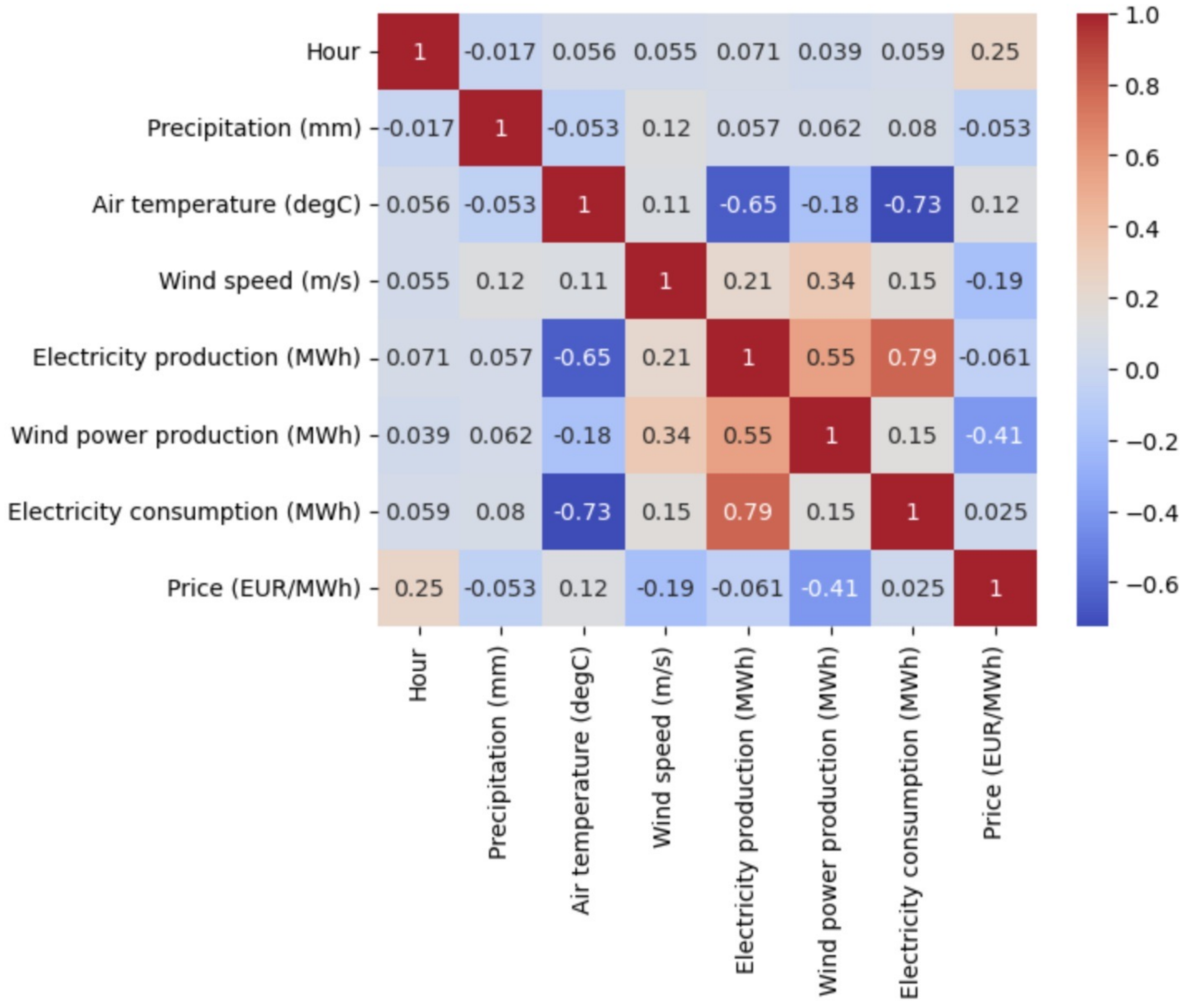
```
4                          0.472464                     0.447331
...                             ...                          ...
8755                       0.841118                     0.945678
8756                       0.805041                     0.876240
8757                       0.801099                     0.870099
8758                       0.772309                     0.822154
8759                       0.769681                     0.813651

      Electricity consumption (MWh)  Price (EUR/MWh)
0                          0.471973         0.036885
1                          0.455447         0.056394
2                          0.447118         0.050288
3                          0.459677         0.051273
4                          0.483342         0.053810
...                             ...              ...
8755                       0.423718         0.032726
8756                       0.457694         0.019879
8757                       0.446854         0.015813
8758                       0.401772         0.019659
8759                       0.370042         0.013925

[8740 rows x 8 columns]
```

The heatmap visualizes correlation between the numerical values of the data.

```
[169]: dataplot = sb.heatmap(data_norm.corr(numeric_only=True), cmap="coolwarm",␣
        ↪annot=True)
       plt.show()
```



	Hour	Precipitation (mm)	Air temperature (degC)	Wind speed (m/s)	Electricity production (MWh)	Wind power production (MWh)	Electricity consumption (MWh)	Price (EUR/MWh)
Hour	1	-0.017	0.056	0.055	0.071	0.039	0.059	0.25
Precipitation (mm)	-0.017	1	-0.053	0.12	0.057	0.062	0.08	-0.053
Air temperature (degC)	0.056	-0.053	1	0.11	-0.65	-0.18	-0.73	0.12
Wind speed (m/s)	0.055	0.12	0.11	1	0.21	0.34	0.15	-0.19
Electricity production (MWh)	0.071	0.057	-0.65	0.21	1	0.55	0.79	-0.061
Wind power production (MWh)	0.039	0.062	-0.18	0.34	0.55	1	0.15	-0.41
Electricity consumption (MWh)	0.059	0.08	-0.73	0.15	0.79	0.15	1	0.025
Price (EUR/MWh)	0.25	-0.053	0.12	-0.19	-0.061	-0.41	0.025	1

The features and labels of the model are stored in corresponding variables. The data is then split into training, test and validation sets, so that the sizes of the groups are 0.8, 0.1, and 0.1 respectively.

```
[170]: features = data_norm.drop(columns=["Price (EUR/MWh)"]).to_numpy()
       labels = data_norm["Price (EUR/MWh)"].to_numpy()

       X_train, X_combined, y_train, y_comined = train_test_split(features, labels,␣
        ↪train_size=0.8)
       X_val, X_test, y_val, y_test = train_test_split(X_combined, y_comined,␣
        ↪train_size=0.5)
```

Linear regression model is the fitted into the training set, the label values are predicted and training error is computed. Similarly, label values are predicted for both test and validation sets, and errors computed for both sets respectively. Finally the R-squared value is computed for the model.

[171]:
```
model = LinearRegression()
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)
train_error = mean_squared_error(y_train, y_train_pred)

y_test_pred = model.predict(X_test)
test_error = mean_squared_error(y_test, y_test_pred)

y_val_pred = model.predict(X_val)
val_error = mean_squared_error(y_val, y_val_pred)

print('Training error:', train_error)
print('Test error:', test_error)
print('Validation error:', val_error)
lin_score = model.score(X_test, y_test)
print("\n")
print("Accuracy score:", lin_score)
```

```
Training error: 0.015721635159469997
Test error: 0.01449616265680009
Validation error: 0.015307627317048935


Accuracy score: 0.3368686296805933
```

Random Forest model is the fitted into the training set, with different values (1 to 10) of maximum depth of the tree. Training errors, validation errors and test errors are also computed for these depth values.

[172]:
```
train_errors = []
test_errors = []
val_errors = []
scores = []

depths = range(1, 11)

for i in depths:
    rf_regr = RandomForestRegressor(max_depth=i, random_state=0)
    rf_regr.fit(X_train, y_train)
    rf_score = rf_regr.score(X_test, y_test)

    rf_train_pred = rf_regr.predict(X_train)
    rf_train_error = mean_squared_error(y_train, rf_train_pred)

    rf_test_pred = rf_regr.predict(X_test)
    rf_test_error = mean_squared_error(y_test, rf_test_pred)
    rf_val_pred = rf_regr.predict(X_val)
    rf_val_error = mean_squared_error(y_val, rf_val_pred)

    train_errors.append(rf_train_error)
    test_errors.append(rf_test_error)
    val_errors.append(rf_val_error)
    scores.append(rf_score)
```

The results for different values of max depth are gathered to a DataFrame for clarity.

```
[173]: rf_errors = pd.DataFrame({"Max depth": range(1,11), "Training error" :␣
        ↪train_errors, "Test error": test_errors, "Validation error": val_errors,␣
        ↪"Accuracy score": scores})
       rf_errors
```

```
[173]:    Max depth  Training error  Test error  Validation error  Accuracy score
       0          1        0.020299    0.018539          0.019714        0.151937
       1          2        0.017498    0.015907          0.016770        0.272308
       2          3        0.015916    0.014685          0.015396        0.328238
       3          4        0.014159    0.013439          0.014036        0.385237
       4          5        0.012446    0.012369          0.012581        0.434155
       5          6        0.010751    0.011548          0.011360        0.471739
       6          7        0.009154    0.010852          0.010343        0.503593
       7          8        0.007596    0.010206          0.009610        0.533104
       8          9        0.006149    0.009648          0.008895        0.558638
       9         10        0.004849    0.009123          0.008318        0.582676
```