Department of Mathematics and Computer Science

University of Southern Denmark

Master Thesis in Data Science

Exploring climate data's relevance to predict energy consumption

Author

Mathias Østergaard Hansen

Exam nr: 453228

mathh17@student.sdu.dk

Supervisors

Vaidotas Characiejus(SDU)

Jeppe Miki Busk Olsen(Energinet)

May 31, 2022

Contents

1 Introduction **3**

2 Data Collection and Preprocessing **5**

2.1 Observational data from DMI 5

2.2 Electricity Consumption Data from Energinet 8

2.3 Meteorological Forecast Data 8

3 Data Exploration **12**

3.1 Data Visualization 12

3.2 Feature importance 15

4 Literature Review **17**

5 Methodology **20**

5.1 Extreme Gradient Boosting 20

5.1.1 Gradient Boosting 20

5.1.2 XGBoosting 22

5.2 Long Short Term Memory 23

5.2.1 Neural Networks 24

5.2.2 Gradient Descent 24

5.2.3 Recurrent Neural Networks 25

5.2.4 Long Short Term Memory 27

6 Implementation **29**

6.1 Error metrics 29

6.1.1 Parameter Tuning of XGBoost 30

6.1.2 Encoder/Decoder LSTM model 32

7 Results 36

8 Discussion and Further Work 40

9 Conclusion 40

1 Introduction

In today's society where countries are shifting from fossil fuels towards renewable energy, it becomes increasingly important for these countries to be able to sustain a reliable electrical grid. If more and more sectors become dependent on renewable energies, which in Denmark consists of mostly wind and solar electricity, the consequences of the electrical grid going down becomes worse. A big part of sustaining a reliable electrical grid is being able to allocate the need for the electricity and supply in advance. The transmission system operators(TSOs) handles this responsibility, and a big part of this is forecasting the electricity consumption in advance. The more precise these forecasts are, the better these TSOs can plan the production and behavior of the needed electricity.

The topic of this thesis is to investigate the use of climate data in the predictions and forecasts of electricity consumption. The reason this could be an interesting topic is because in Denmark approximately 80% of the produced electricity is used for heating homes, Energistyrelsen, 2019. The goal of this project is to explore if the use of weather data would have a beneficial effect when predicting electricity consumption. in order to utilize the information from the relevant weather data, machine learning algorithms will be implemented and trained on this data. The process of collecting data and how it is preprocessed will be explained, together with tables clearly illustrating the data when it is collected. When the data is collected, it will be explored, looking for possible correlations and dependencies. The importance of the chosen features will also be explored through SHAP values. From these values and the possible correlations the final features that will be used in the project will be chosen. Before investigating the use of this data with machine learning models, a literature review will be carried out in order to find out what is considered as being best practice and industry standards for predicting the electricity consumption. Two machine learning models will then be explained in order to gain insight to how these models function. Because having a complete understanding of the models before implementation is of high importance in order to utilize the full potential of these models. The two models will be implemented using observed weather data to train the models and evaluate their initial performances. After which, they will make forecasts for the electricity consumption using forecast weather data. The precision of these predictions will then be compared to a naive baseline which will work as a frame of reference for the performance of these models.

This project will be carried out in collaboration with the danish TSO, Energinet, in order to gain initial insights in their data, experiences and needs for such forecasts.
In Denmark the Electrical grid can be divided in two zones, DK1 and DK2, by the bridge Storebælt, which connects Fyn and Sjælland. DK1 being west of Storebælt and DK2 being east of Storebælt. Energinet is interested in seeing the performance of machine learning models for forecasts on electricity consumption for these two zones. Specifically are Energinet interested in hourly forecasts, which will serve as the scope of the predictions done to asses the benefit of weather data in these predictions.

2 Data Collection and Preprocessing

With a description of what exactly we want to investigate, the relevant data will have to be collected and preprocessed. In this chapter 3 types of data will be collected, historic observations for weather data, historic electricity consumption and historic forecast data produced by DMI.

2.1 Observational data from DMI

DMI has their historic meteorological observations publicly available though their API-service. The dataset that is interesting for this project is the MetObs dataset which contains 48 features, where the three features relevant to this project will be hourUTC, temp_mean_past1h and radia_glob_past_1h:

Table 1: Table showing top and bottom rows of the relevant features from the MetObs dataset

hourUTC	temp_mean_past1h (C°)	radia_glob_past_1h (W/m^2)
2010-01-01T00:00:00	-4.62	0
2010-01-01T01:00:00	-4.92	0
2010-01-01T02:00:00	-5.31	0
...	...	...
2019-12-31T21:00:00	3.21	0
2019-12-31T22:00:00	2.89	0
2019-12-31T23:00:00	2.93	0

The features temp_mean_past1h and radia_glob_past_1h each corresponds to the mean over the observations of the past hour, for the given feature. HourUTC is the time of which the observed values were reported. The data set of the historical observations is downloaded from 00:00 01-012010 to 23:00 31-12-2019. The reason for this is because Energinet was concerned of the effect that COVID-19 has had on the behavioural patterns for electricity consumption. This lead to the decision of excluding the years 2020 and 2021.

When the data set is collected through the API, the data is downloaded station by station, then in each station contains a dataframe of all the recorded observations. For the purpose of predicting the consumption for the areas DK1 and DK2 the weather stations will be divided into three groups, DK1,DK2 and neither. The stations that get placed in the neither category will be excluded. The criteria for excluding a station is if the station is placed either at sea or on the island of Bornholm. The reason for this is that Bornholm is only responsible for a minimal amount of the overall consumption of DK2 which is the area it would placed in, and the weather on Bornholm might differ significantly from the rest of DK2.

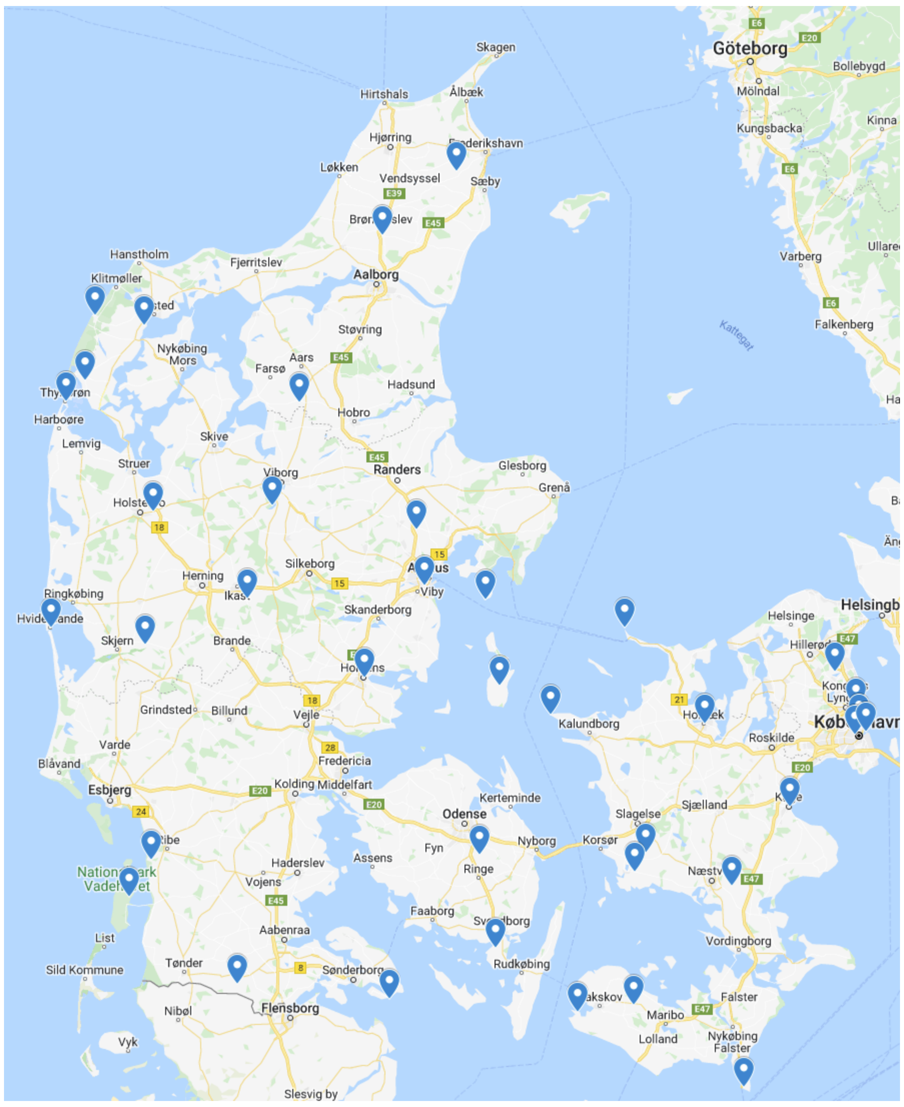


Figure 1: Map of weather stations in the groups DK1 and DK2

This was done by taking the most eastern, western, southern and northern coordinates from both DK1 and DK2. The returned stations are then checked manually to see if it is in within the right category, and if it is not the data from the station is removed. This resulted in complete data from 25 stations for DK1, and 15 DK2, Figure 1. All the data from each station in each area are then added together to get a mean for each feature for the given price area.

This will be the data that will be investigated further and possibly used to train the models. The reason for choosing the features time, temperature and radiation comes from advice and experience from Energinet. From their experience it would be the two interesting features from the observations. Time and temperature is used given the amount of electricity used for heating. Radiation is added as a feature because Energinet wants to exclude the amount that residents produced from solar panels. Therefore an initial presumption is that there is a correlation between high radiation and a decrease in consumption. The solar panels produce electricity that is added directly to the danish electrical grid. This means Energinet will have to produce less electricity from the power plants because the solar produced electricity is not consumed by the single consumer but it is fed into the general grid.

The downside to the MetObs dataset is that the data set is not quality checked and requires the user to handle erroneous data. DMI describes the occurrence of outliers usually comes from malfunctioning sensors, wear and tear, and extremely rarely vandalism.

In the case of this project the two interesting parameters from the metObs data set are the temperature and radiation. These two features are means over an hour of data. Which means the possible outliers will be watered down before being reported. In this project, each recorded observation will be added together with a set of observations from other weather stations and then have another mean calculated from those values. Thus impact of an outlier from one reading will be very small. Therefore, the only action that will be taken against outliers for this dataset is by looking over the data sets for possible malfunctioning stations and removing them. Which ended up being two stations that would continuously report wrong data for radiation.

In 2022 DMI releases a new data set through their API service called "Climate Data". The climate data set contains the same weather data as the MetObs one, but this data set is quality checked and rinsed for outliers. Unfortunately, this was not available at the time this project was carried out, but it would be an interesting to explore the use of this data set instead of the MetObs data set.

2.2 Electricity Consumption Data from Energinet

Just like DMI, Energinet's data is available through an API. From their web portal: energidataservice.dk. The relevant data set for this project is the one called *"Production and Consumption - Settlement"*. This is a data set of the over all electricity consumption divided into the two price zones DK1 and DK2, and fortunately this data set is quality checked and 99.9% correct. This data set contains many features, of which the price area, time, gross consumption and solar power self consumption will be the ones relevant in this case. The feature, solar power self consumption, will not be a part of the final data set, but the data from this feature will be deducted from the gross consumption feature. The reason for this is because Energinet wants to know how much electricity they need the power plants to produce, and that is exclusive of this production from the solar panels.

Table 2: Table showing top and bottom rows of net consumption data from Energinet, from which the solar produced electricity has been deducted

hourUTC	PriceArea	Consumption (MWh)
2010-01-01T00:00:00	DK1	2025.6
2010-01-01T01:00:00	DK2	1554.8
2010-01-01T02:00:00	DK1	1951.1
...	...	...
2019-12-31T21:00:00	DK1	1840.5
2019-12-31T22:00:00	DK2	1349.3
2019-12-31T23:00:00	DK2	1773.2

2.3 Meteorological Forecast Data

When the models have been trained on the observed data from the MetObs dataset, the models should be able to use the forecast data Energinet receives directly from DMI. For the purpose of this project, the time horizon that the models will try to make forecasts for is the year of 2019. Thus the model will be tested on a complete seasonal year, and it should yield enough predictions to give insights to the performance of the models. This test data set is different from the historical observations.

Table 3: Table showing the forecast data produced by DMI

Date	Predicted Ahead	Radiation Mean (W/m^2)	Temperature Mean (Kelvin)
2019-01-01T00:00:00	3	0	280.1
2019-01-01T00:00:00	6	0	280.1
2019-01-01T00:00:00	9	0	280.1
...	...	...	...
2019-12-31T23:00:00	46	0	279.5
2019-12-31T23:00:00	47	0	279.3
2019-12-31T23:00:00	48	0	279.5

Energinet receives forecast data from DMI in many different time resolutions, for this project the data set will be the hourly forecast data which comes in every 3rd hour 54 hours ahead, Tabel 3. The way this is structured could be expressed as such: $\hat{T}_{24(k-1)+j,h}$. Where k represents the day of the year in a range from 1 to 365, j represents the hour of that day in a range from 1 to 24 and h represents the number of hours ahead the values were predicted. As an example, the temperature 20 hours ahead for the date 5/1 at 8 a.m. would be represented as $\hat{T}_{24(5-1)+8,20}$ .This allows for predictions being done for single time steps 48 hours ahead, for each available time step. An interesting topic to explore was how precise the forecast data were in relation to the actual observed values. Also, how much does the time ahead the values is predicted affect the precision of the forecast?

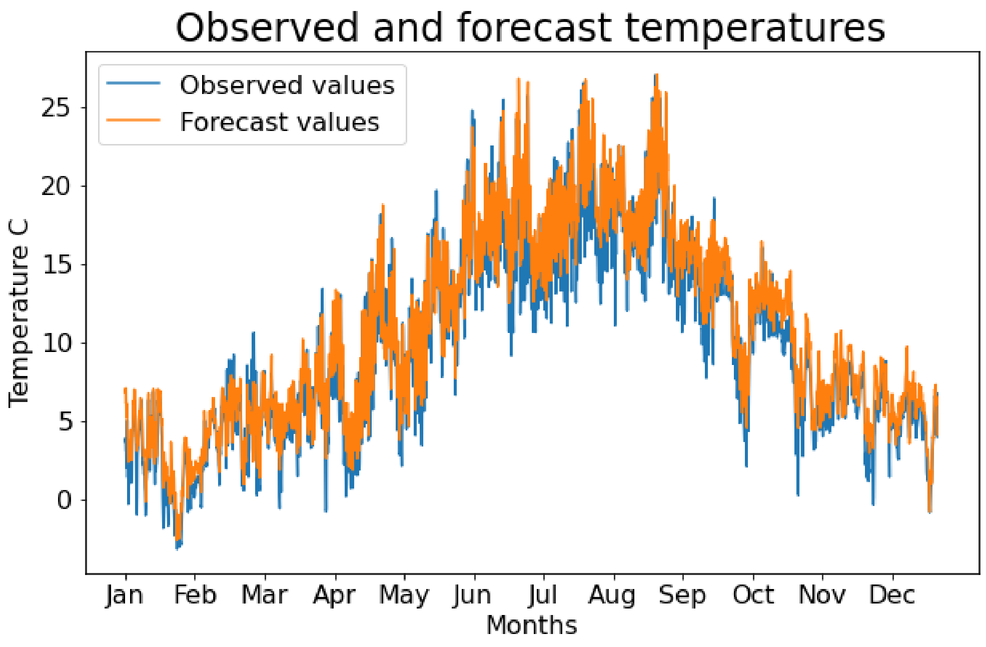


Figure 2: Graph of observed and forecast temperature for the year 2019

Looking at Figure 2, the forecasts for temperature fits the behaviour of the observed values. The forecast struggles most with predicting the temperature during the hotter months such as July and August.

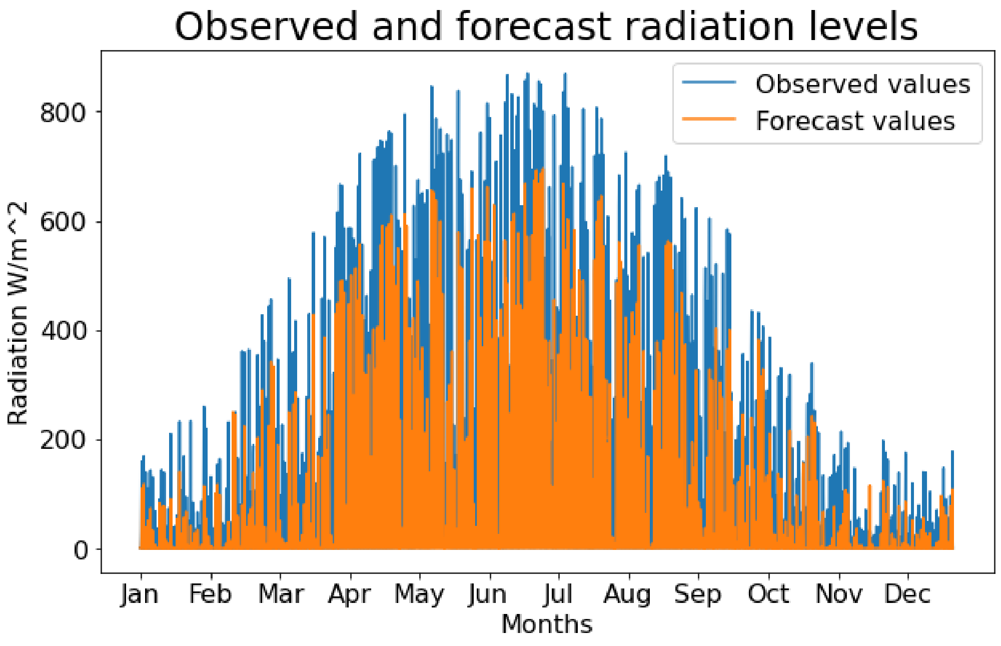


Figure 3: Graph of observed and forecast radiation for the year 2019

From Figure 3 we can gather that the forecast radiation values deviate from the observed values. looking at the graphs it mostly underestimates the radiation. In order to get an understanding of how precise these forecasts are the R^2 value was calculated for both 1 and 48 hour ahead forecasts.

Table 4: R^2 values for the year 2019 for 1 and 48 hours ahead forecasts.

R^2 values for forecast data		
	1 hour ahead	48 hours ahead
temperature	94.2	93.8
Radiation	70.5	67.4

Looking at the R^2 values from Table 4 for temperature, DMI is able to produce precise forecasts for the temperature over the course of a year. Both for 1 and 48 hours ahead, the reduction in the R^2 value going from 1 hour to 48 hours is minimal and does not appear to be of any concern. However, the R^2 values for radiation suggests the same as the plots. There are errors in the forecast of radiation, when compared to temperature. This does make sense, since when it comes to radiation, the value is much more locally variable than temperature. The reason for this is that cloud coverage has quite the effect on radiation and it is difficult to predict the precise placement of the clouds. In terms of data quality this might turn out to be quite the factor for eventual error in the predictions, since you want the training data to be as similar to the actual data as possible. It should be investigated if this feature should remain in the final dataset, since it might end up affecting the precision of the model.

3 Data Exploration

Now that the relevant data has been collected it will be explored through visualization in order to get a overview of the correlations and dependencies between the data. From experience from Energinet and general knowledge there are relations which will be expected to show and that is:

- There is a weekly pattern of power consumption where there is a decline during weekends and holidays.
- There is a correlation between the time of day and consumption.
- There is a negative correlation between sun radiation and consumption.
- There is a correlation between temperature and consumption.

3.1 Data Visualization

Firstly the time of the observations will have to be organized in a manner that allows to fully investigate it's impact on consumption. For the purpose of this project where there are behavioural patterns for electricity during the week and holidays, a binary feature will be added to the data set, if it is a holiday or not. This feature should capture the decline in consumption that evidently happens on weekends and holidays Ziel, 2018.

Looking at the behaviour of consumption and weekends or holidays, it appears that there is a correlation between weekends and consumption, Figure 4. The consumption appears to decrease during the weekends, which confirms the results put forth by Ziel. This pattern possibly comes from a large part of the industrial sector closes down over weekends and public holidays. Ziel also compares the effect of public holidays like Easter over a seven year period, and it appears that the impact of holidays is rather stable. This representation of days of the week seems to capture the behaviour of the electricity consumption in a sufficient manner.

Since most of the danish population is sleeping during the night it is expected

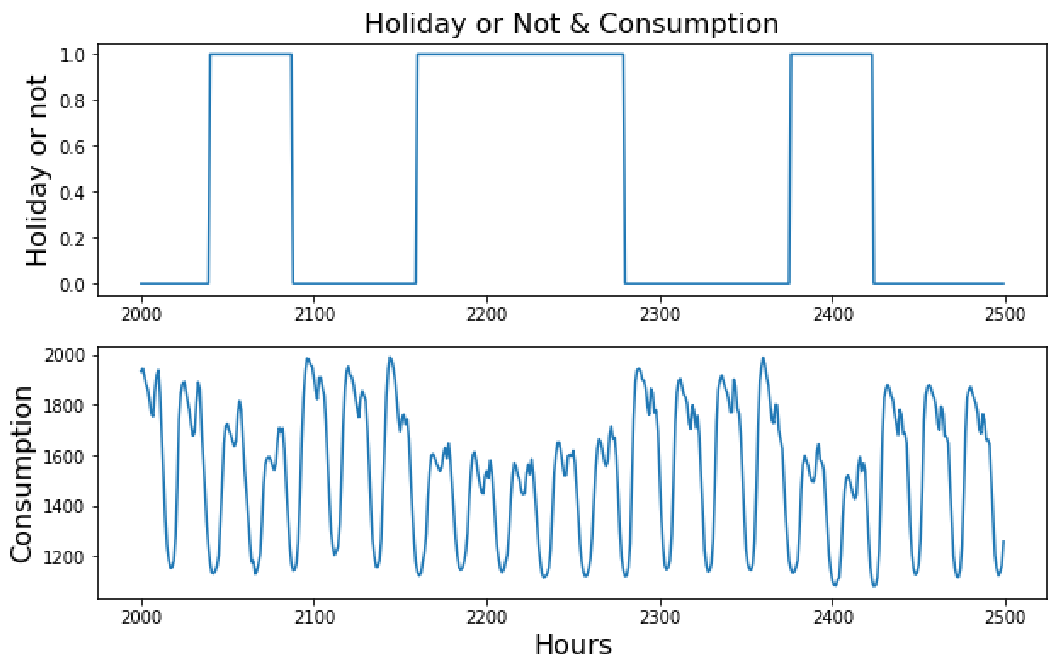


Figure 4: Graphs over consumption and holiday feature over a period of time, specifically around Easter

that the electricity consumption is reduced during those hours, but exactly how is the behavioural pattern of the consumption in relation to the hour of the day?

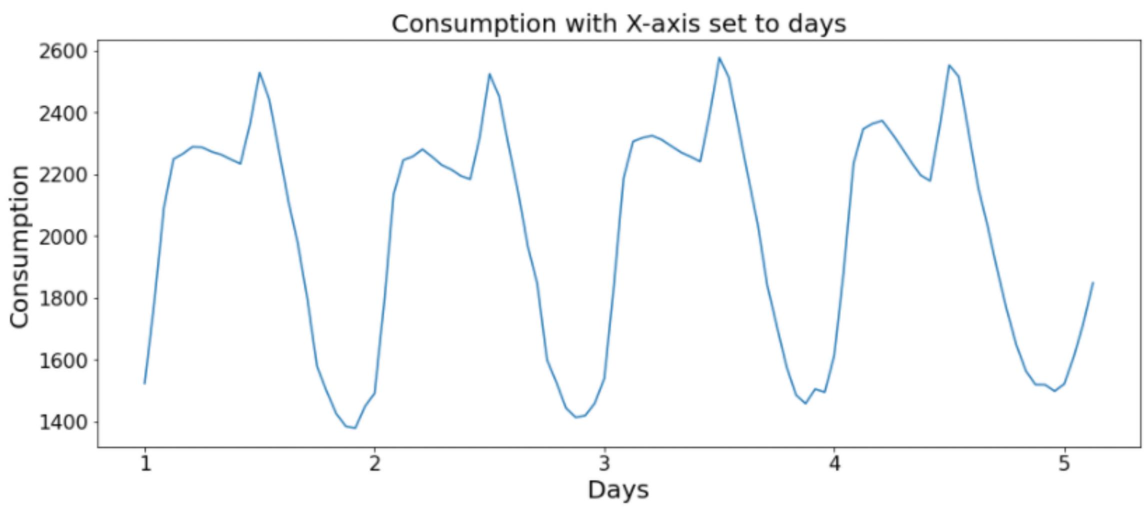


Figure 5: Graph over consumption over a period of 5 days

As expected there is a clear correlation between the time of day and electricity consumption. Something that seems apparent from the graph, Figure 5, is that there are daily peaks after 4pm, where people return from work. This makes sense since people generally will be preparing dinner, and using their appliances.

During the last 10 years solar panels have become increasingly more popular in Denmark, in September 2019 106,461 soloar panels to be exact, Bolius. As stated earlier, this is not a form of consumption that Energinet needs to predict, therefore the consumption is deducted from the gross consumption. By now, with so many solar panels installed the effect from these solar panels is expected be noticeable on the net consumption.

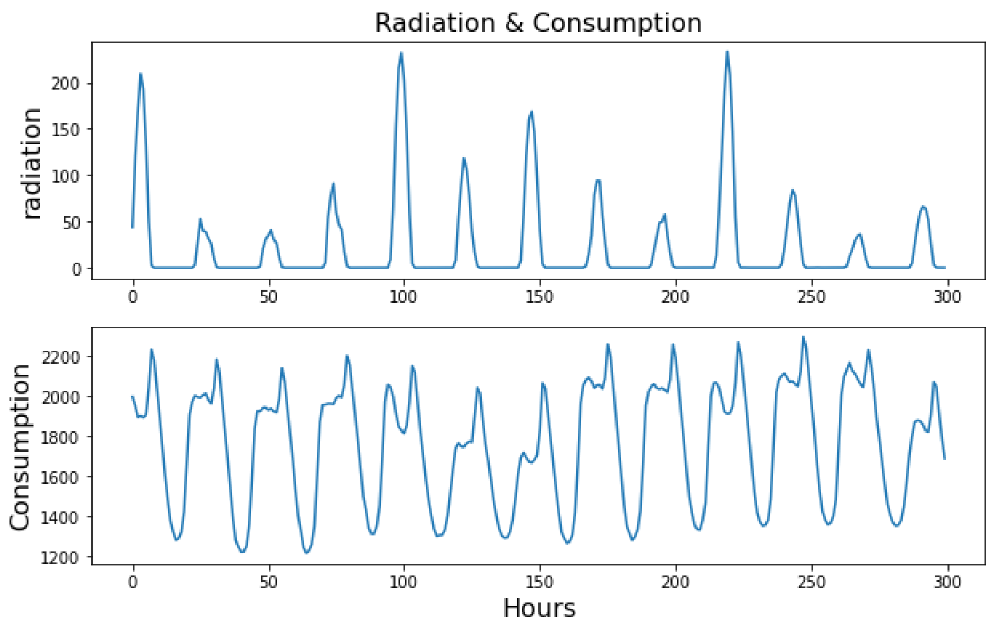


Figure 6: Graphs over consumption and radiation over a period of time

Inspecting the graph, Figure 6, it is possible to see a noticeable dip in consumption on the days with high radiation. Around 100 hours, and 220 there are small dips in the consumption, which seems to be caused by the increased radiation. There are also other possible reasons this could happen. It might be because people are more willing to spend time outside when the sun is shining, and that they just simply are not using their appliances. However, the radiation appears to be an indicator of this reduction in consumption.

Since such a big amount of the residential electricity consumption is used for heating up houses, it is expected to see some form of correlation between temperature and electricity. In the danish heating industry, the companies convert the temperature to degree-days to best capture the energy required to heat up a house, *Energihåndbogen* 2019. Degree-days is a unit for measuring the difference between the mean outside temperature and the mean inside temperature, in this case being, 17 degrees Celsius. This gives a degree-day of 5 if the outside temperature is 12°and -2 if the outside temperature is 19°. It can also be expressed as $degreedays = -C + 17$, where C is the temperature in Celsius. The reason the heating industry uses this unit rather than just using temperature, is because this best captures the impact the cold has on the energy required to keep a house at the mean temperature.

By advice from Energinet, this will also be the unit used for this project, since Energinet also uses degree-days when measuring temperature in relation to energy consumption.

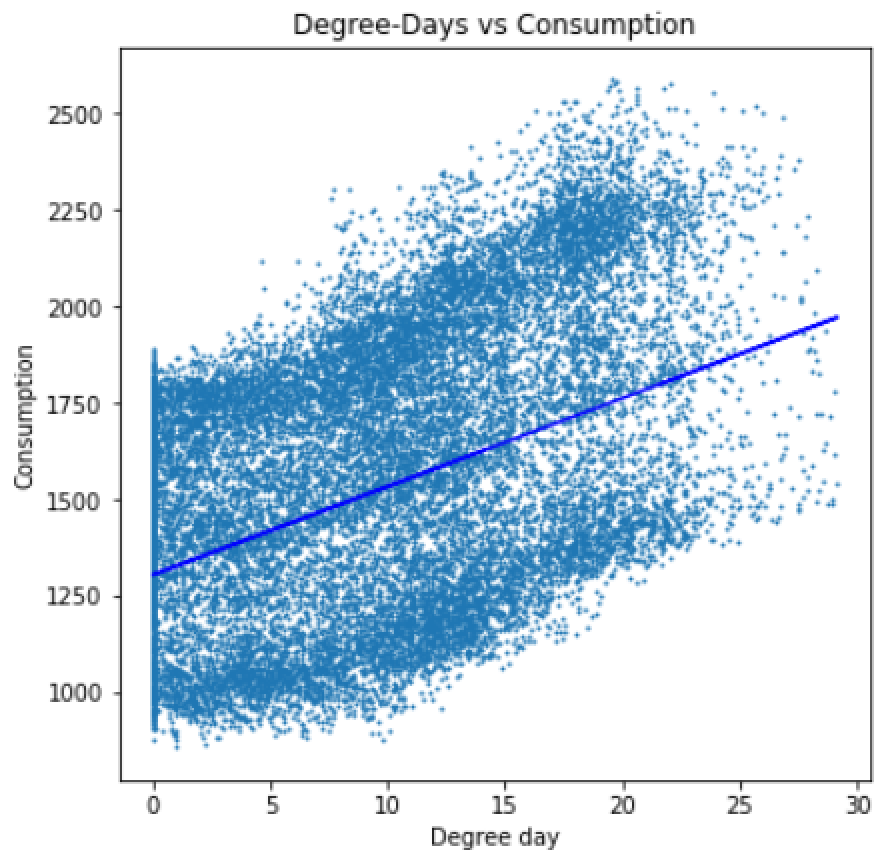


Figure 7: Plot of the consumption in relation to degree-days with a tendency line

There appears to be a correlation between degree-days and the electricity consumption, Figure 7. As the degree-days gets closer to zero it does seem like the correlation begins to stagnate, which fits the narrative stated earlier, that after a certain point the temperature has no effect on the consumption.

3.2 Feature importance

After having visually inspected all the relevant features available, a final step was taken to see the importance of these features. As it is visible from the graphs and plots above, the correlations are not equal. Therefore, to get a better understanding of the information a machine learning model is able to extract from these features, the SHAP values from the features will be calculated. By calculating a predictions SHAP values, SHapley Additive exPlanations, you can break down the impact the values of each feature has on a set of predictions. This is useful for checking for redundant features in your data set. Shapley values are based on a solution concept from game theory used to determine the contribution of each player, and the name shapley comes from the mathematician, Lloyd Shapley who introduced the theory in 1951. In 2017 this theory was used in machine learning, Lundberg and Lee, 2017, with a python package that calculates these values based on ones predicter, SHAP. In order to calculate the SHAP values a very basic predicter will be implemented, in this case a XGBoost predicter. XGBoost is a boosting algorithm which boosts many small decision tree, for calculation the SHAP values on a prediction like this, the "tree interpreter" function will be used from the SHAP package. Using the simple XGBoost model the SHAP values calculated for the features can be presented in a summary plot:

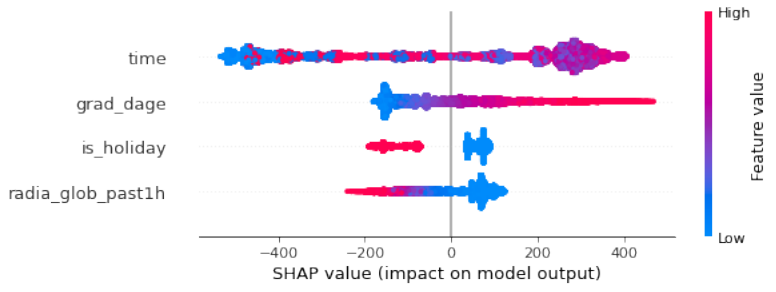


Figure 8: Summary plot of the SHAP values, for each feature

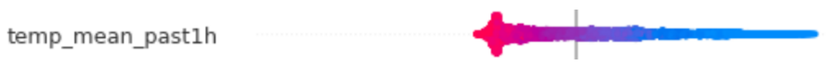


Figure 9: Plot of the SHAP values for temperature

Degree-days have a nice gradient coloring throughout the whole plot. It seems to be very clear to the model that a high value degree-day means an increase in consumption. the impact of appears to be more consistent for degree-days than the time feature. The degree-day feature stops abruptly for low values, this is at the 17 degrees, where the degree-days are capped and goes to zero. In order to check whether or not the model is able to gather some information from higher temperatures i made the same plot, but with temperature in Celsius, Figure 9, and it appears that the temperature has the same impact as degree-days. For the is_holiday feature all values that indicates a weekend or a holiday seem to produce a negative deviation for the consumption, which is in line with the correlation found in the plot.

The radiation plot shows that the impact of the radiation actually causes a negative deviation, and this confirms that the model is actually able to pick up on the small dips that appeared to be present on sunny days. Since the temperature stops yielding information for the model when the temperature is above 17 degrees, hopefully this feature is able to give some information on sunny days during the summer, where the model does not get information from degree-days.

However, the time feature seems to be more difficult to decipher. Since the SHAP values represents data in a continuous manner, representing the time feature is not very good, since it is cyclical and not continuous. This means whenever the time passes midnight, the feature value goes from the highest possible value to the lowest, but the consumption is not cyclical. This explains the mix of the peak values throughout the plot. Looking at the right side of the plot it would appear that the model is able to pick up the daily peaks in the data, that happens around 16 when people return home from work.

After inspecting both the data correlations and their impact on data models, there does not seem to be any redundant features. The final data set that will be used for training looks like this

Table 5: Features in the data set that will be used for training the model

Training Dataset				
hour	Degree-days	Radiation	holiday	Consumption
0	22.2	0.4	1	2025.6
1	22.2	0.4	1	1951.1
2	22.2	0.4	1	1864.5
...	...	...	...	...

4 Literature Review

Predicting energy consumption is not a new subject of study in the field of machine learning. There are wast amounts of papers, reports and projects on the subject. With the increase in popularity for the subject of machine learning and especially neural networks in the last 20 years, many new methods of predicting this consumption have been developed. In the report Ghalehkhondabi, Ardjmand, and Weckman, 2016 it is made clear how big of an increase forecasting energy consumption has seen, and that is just from 2005-2015. In the figure from the report, the increase is depicted in a diagram showing the growth from 1994-2014, which appears to be exponential. However, it is not investigated if this matches the general increase in published papers or if this only applies for machine learning papers. The report states that there is a big interest in the subject. In the report they state that the reason for the importance of being able to predict the electricity consumption has many factors. Distribution of electricity is a big factor, since you do not want to overproduce, or under produce. In Denmark overproducing is handled by exporting excess power to neighbouring countries. This happens quite frequently because Denmark is able to produce large amounts of electricity by wind. Therefore correctly distributing and allocation of electricity is of big importance. To get some perspective on state of the art models and what is considered best practice on this topic, relevant literature will have to be researched and reviewed. Review papers will be the main way of gaining a broad perspective on relevant models and implementations, given the wast amounts of research on the topic. In the report, Román-Portabales, López-Nores, and Pazos-Arias, 2021, a wide variety of papers on the subject of Artificial Nerual Networks(ANN's) used to forecast electricity demand are reviewed. In total they review 56 papers, and interestingly enough 46 of the papers are on Short Term Load Forecasting(STLF). It appears from the paper that the most popular load forecasting method is short term, in terms of ANN's. From the applicability of ANN's and the way they use data to make predictions this makes sense. Because ANN's is very reliant on the quality and precision of the data available to the model, so making long term load forecasting(LTLF) or even medium term would seem difficult. STLF seems to be predicted from weather forecasts, which becomes more and more incorrect the further in the future the forecasts are done. In terms of best performing models the paper states that usually the best performing models are models that are able to capture recurrent patters, such as recurrent neural networks(RNN). Which is in line with they claim that using the previous energy consumption for forecasting in the future is a better parameter than the weather. Especially the Long Short Term Memory,LSTM, is able to achieve some very good results. An interesting note is that the paper states that with the LSTM the predictions tend to become more accurate the more consumers the data is aggregated over, which might be of interest for TSO's such as Energinet, if they want to implement forecasting in smaller scales than the zones DK1 and DK2. The review paper, Runge and Zmeureanu, 2021, comes to similar conclusions as the previous paper when applied to larger scale predictions. However in one of the papers that is reviewed, the extreme gradient boosting algorithm, XGBoost, shows the best results. But in that specific paper it seems to be a regular Deep Neural Network (DNN) rather than a form of RNN, that the XGBoost outperforms. In the review paper, Tyralis and Papacharalampous, 2021, they state the use of boosting algorithms as being "underexploited". The modern boosting algorithms, such as XGBoost, LightGBM and CatBoost, have shown state-of-the-art prediction performances. The paper goes into great length to explain the ease of implementation of these modern boosting algorithms. Especially the XGBoost algorithm is generally perceived as having a very good out-of-the-box performance. This is in line with one of the conclusions from Ghalehkhondabi, Ardjmand, and Weckman, 2016 in which they state:*"As a conclusion, developing simple methods with acceptable accuracy is an attractive area of future studies."* Since the computational resources needed for these very deep neural networks are significantly higher than the newer machine learning algorithms, such as gradient boosting algorithms. From the review papers, the state-of-the-art methods of load forecasting seems to be using a form of RNN, more specifically LSTM's seems to be very popular. Boosting algorithms also seem to be able to achieve acceptable performances. These two types of models will be the ones that will be implemented in this project. The LSTM is chosen because it seems to be the most popular for this specific task, and XGBoost is chosen because from the review papers they show great promise.

5 Methodology

Now that a general understanding of what is perceived to be some of the best practices for modeling load forecasting, it is necessary to understand the chosen methods on a deeper level. In this subsection the models: Gradient Boosting, specifically XGBoost and the RNN model LSTM will be explained in order to gain an in-depth understanding of the models.

5.1 Extreme Gradient Boosting

5.1.1 Gradient Boosting

The general idea behind machine learning models is to construct one model that is able to make predictions. However, boosting algorithms aims to train a sequence of weak learners that as a whole is able to make predictions, also called ensemble learning. Each model is trained to compensate for the weaknesses of the model before. In the relevance of gradient boosting, the weak learners consists of small decision trees, which are sequentially improved upon based on the residual errors of the previous decision tree. The first tree in the sequence is usually just a leaf which predicts the average of the target values, for regression problems. After this initial prediction, the boosting algorithm will gradiently reduce the error by adding more trees.

Decision trees consists of a certain amount of binary statements, also called splits, these splits is used to either make a prediction or there will be a new split until they reach a prediction, Figure 10. Gradient boosting algorithms utilizes these trees as the weak learners they train sequentially.

The sequential connection of these trees means for each tree, that tree is fitted to the residual errors of the previous tree, which it then tries to minimize. In the case of regressional implementations of gradient boosting algorithms, the residual error is calculated by a chosen loss func-

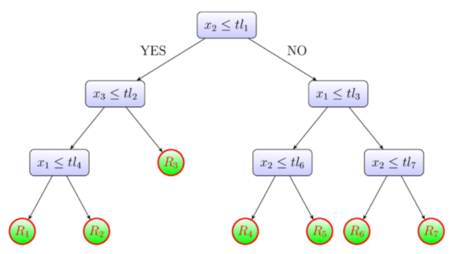


Figure 10: Example of decision tree

tion. Each trees contribution is scaled by the learning rate. The learning rate ensures that each tree contributes the same amount. Friedman who was the one to first implement gradient boosting states that lower values for the learning rate typically yields better results Friedman, 1999. There is however, a trade-off in terms of implementations. Having a low learning rate require more trees in the model and that increases the computational requirements for the model. Each tree then contributes to the overall prediction with a weight that is added to the rest of the trees weights. This means the model aggregates the weights of all the small trees, this value is then the deviance from the average value that the prediction is.

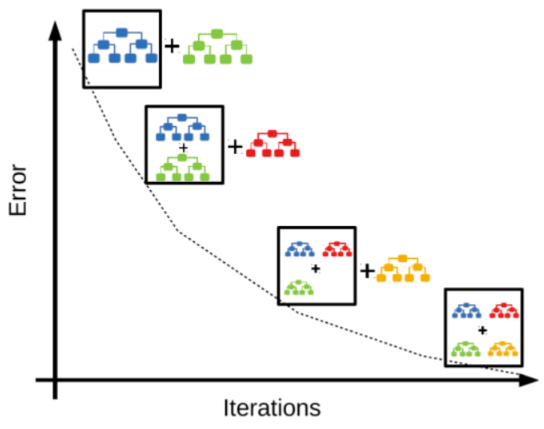


Figure 11: Illustration of how the gradient boosting algorithm sequentially adds trees to reduce error

When implementing gradient boosting algorithms the depth of the trees will have to specified as it is one of the hyperparameters for this form of boosting algorithm. The trees calculated by gradient boosting algorithms have a fixed depth, and allows for the individual trees to be quite deep. However increasing the depth of the trees, also increases the computational recourses required by the model quite significantly.

5.1.2 XGBoosting

The objective of the XGBoost algorithm is to minimize the value of $\mathcal{L}$ in and additive manner given by the regularized formula:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$
$$\text{Where } \Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2 \tag{1}$$

Where l is the loss function that calculates the residual errors in that particular iteration step. y_i represents the target, and $\hat{y}_i^{(t-1)}$ is the prediction from previous time step. f_t represents a tree structure q given the input vector x_i, and leaf weights w. γ and λ are the regularization parameters of the XGBoost model is a key difference in regular gradient boosting algorithms and the XGBoost algorithm. γ is the threshold for the pruning done on leafs, and λ aims to reduce the predictions sensitivity to individual observations, thus helping the model from overfitting. The initial prediction of the XGBoost algorithm consists of just the average value of the target value. From this, the residuals g, is calculated and is used to calculate the score of the structure in the tree using equation 2, which is thoroughly explained in the original paper by Chen and Guestrin, 2016.

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \tag{2}$$

Where t is the iteration, $\tilde{\mathcal{L}}^{(t)}(q)$ is the quality of the tree structure at q, T is the number of leaves in the tree, γ and λ are regularization terms. In XGBoosting the splits for the decision trees are based on a value $\mathcal{L}_{split}$. This is a representation of the information gained from performing the split on the specific values in the particular tree. The information gain of a split is then calculated using the formula:

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma \tag{3}$$

From the original paper for XGBoost,Chen and Guestrin, 2016, the computational algorithm for split finding is expressed as:

```
**Algorithm 1:** Exact Greedy Algorithm for Split Finding
Input: $I$, instance set of current node
Input: $d$, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i,\ H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ to $m$ do
    $G_L \leftarrow 0,\ H_L \leftarrow 0$
    for $j$ in sorted($I$, by $\mathbf{x}_{jk}$) do
        $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
        $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
        $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end
end
Output: Split with max score
```

Algorithm 1, the exact greedy algorithm, iterates over all possible splits and thus comes to the best possible split. But from the XGBoost documentation it is advised to only use this algorithm for smaller dataset since it becomes quite computationally heavy. In the case of this project, the data set is not large enough that this becomes a problem, so the greedy algorithm will be the one used in this project. However, XGBoost has an algorithm, the approximate algorithm, was developed to increase XGBoost scaling capabilities for bigger data sets. Even though this helps reducing the computational requirements for finding the best split. XGBoost goes to an even greater extend by implementing column blocks for parallel learning. This lets the algorithm split up the dataset so multiple threads can store and work on calculating statistical properties. In the paper describing XGBoost, Chen and Guestrin, 2016, they state that at larger data sets the cache-aware algorithm is able to run twice as fast as the naive.

5.2 Long Short Term Memory

LSTM's are a form of Neural Network, NN, in the subcategory of NN's called Recurrent Neural Networks, RNN. It is therefor crucial to understand both NN's and RNN's to fully utilize the power of LSTM's.

5.2.1 Neural Networks

Neural networks are a set of algorithms that together tries to generalize over a set of training data enough to make predictions for a given feature. A NN in its simplest form consist of 3 types of layers, a input layer which contains the input data for the algorithms to generalize over. An output layer which will contain a representation of the target feature, typically in a value between 0 and 1. In the middle of these two layers are what is called the hidden layers. The reason the middle layers are called hidden layers is because the value these layers contain are usually not relevant for the user to know.

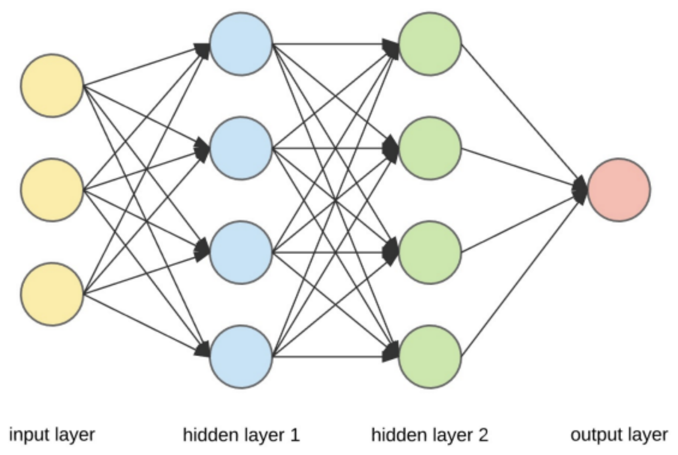
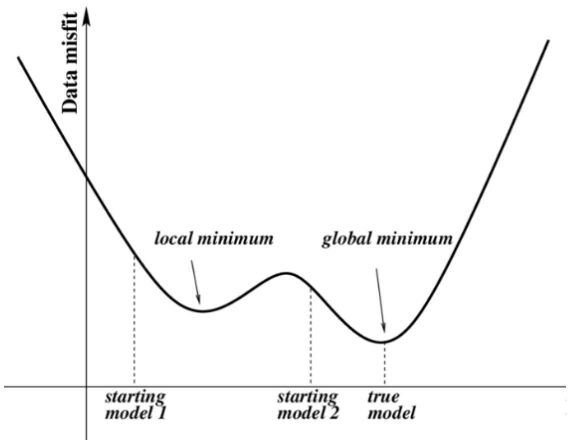


Figure 12: Diagram of a simple neural network

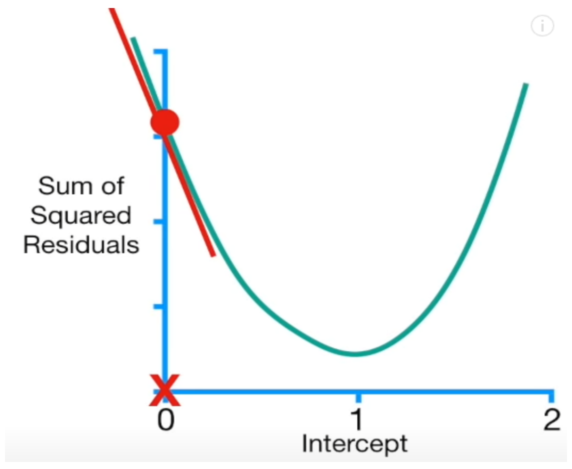
In a implementation of a NN, there can be many hidden layers, but usually there will be one input and one output layer. The job for these hidden layers are to capture particular patterns for the data. Each node in the layers will try and capture some part of information from the data sent from the previous layer. The change in the information between the layers happens because each node is connected to the nodes in the previous layer, and these connections contains weights and biases. When we talk about training a neural network or that the network is learning, what is meant is that these weights and biases are tweaked and tuned to better capture the input data for the right output.

5.2.2 Gradient Descent

When a model's weights and biases are updated it is done through backpropagation. What is meant by updating the weights and biases for a model is that they are updated in an effort to reduce the Sum of the Squared Residuals(SSR). Reaching the lowest point of SSR's is called reaching the global minimum, this is where the model is as precise as it can possible be. However, finding the global minimum cannot be guaranteed, the reason for this is the way the SSR are minimized during training. In order to decrease the SSR in a timely manner, the SSR is minimized by gradiently calculating the descent of the slope of the SSR. This is called gradient descent and is the way the SSR is minimized so that when the model updates the weights and biases the new values should reduce the SSR.



(a) Depiction of local and global minimum, Ma, 2022



(b) Depiction of gradient descent slope and the steps along the X-axis

Figure 13: Caption for this figure with two images

This might seem simple for the model, the issue is that there might be multiple local minimum and the model might not be able to move past local minimum. With the initialization of the model being random this is the reason there is no guarantee for finding the global minimum. But with gradient descent the model will usually find a local minimum.

5.2.3 Recurrent Neural Networks

A problem that regular feed-forward neural networks suffers from are the lack of memory between predictions. FFN's are not able to transfer knowledge from one prediction to the next. In terms of predicting on sequential data where the previous prediction are related to the next this is a downfall. Recurrent Neural Networks(RNN) solves that issue by containing a "hidden state" which contains knowledge from previous time steps, this hidden state then feeds that information into the next prediction and is updated with new knowledge again.

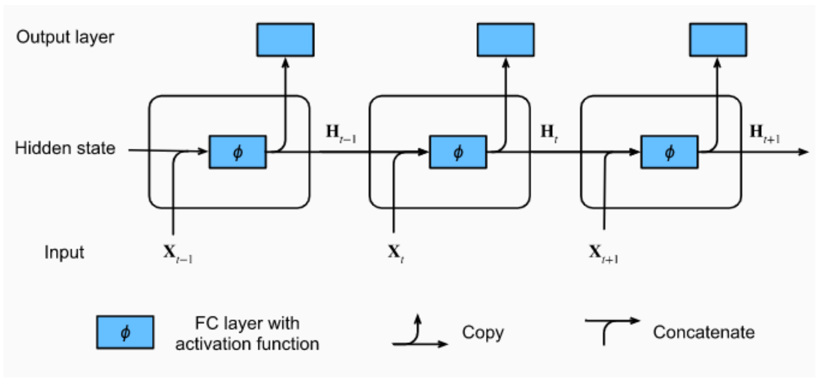


Figure 14: Diagram of a RNN over three time steps.

Figure 14 shows the functionality of a RNN over a three time step cycle. Time step t the hidden state can be computed by: concatenating the input X_t and the hidden state H_{t-1}, from the previous time step, forwarding this information this result into a fully connected layer with the activation function ϕ . The output of such a layer is the hidden state H_t. At this moment the model parameters are the concatenation of the weights W_{xh} and W_{hh} and a bias of b_h. Including the parameters $W_{xh} \in \mathbb{R}^{d\times h}$, $W_{hh} \in \mathbb{R}^{h\times h}$ and biases $b_h \in \mathbb{R}^{1\times h}$, this can be expressed through a function:

$$H_t = \phi(X_t W_{xh} + H_{t-1} W_{hh} + b_h) \tag{4}$$

Thus the ouput of the model can be computed as:

$$O_t = H_t W_{hq} + b_q \tag{5}$$

where W_{hq} and b_q represents the weights and biases of the output layer. However an issue with the RNN is that it is not able to retain enough information to correctly capture the sequential relations for longer periods of time. The gradients of RNN's typically decays over time and this is called the vanishing gradient problem that regular RNN's suffer from. Therefore an extended RNN is needed, to fully capture the sequential information of the data.

5.2.4 Long Short Term Memory

Long Term Short Memory(LSTM) models aim to solve the issue of the vanishing gradient of RNN's by introducing a memory cell. The purpose of this memory cell is to extend the memory of the model even further than the RNN. The information that goes through the memory cell is controlled by different gates. Conventionally they are called the output gate, O_t, input gate I_t and forget gate F_t. the out- and input gates simply controls what comes in and out of the cell. The forget gate gives the cell the ability to reset its content in cases where it is best to ignore inputs.

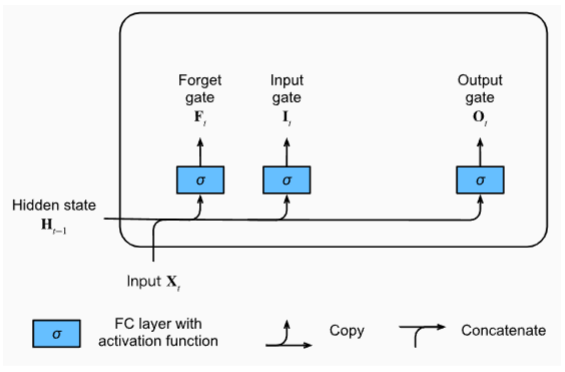


Figure 15: Illustration of the computation done by the gates inside an LSTM

The data fed into the LSTM gates are the input at the current time step, X_t, and the hidden state from the previous time step H_{t-1}. This is processed through three layers with a sigmoid activation function, which calculates the values for each gate, 15. To express this mathematically assume: h hidden units, a batch size of n and the number of units is d. this gives the input $X_t \in \mathbb{R}^{n\times d}$, hiddenst state $H_{t-1} \in \mathbb{R}^{n\times h}$. The gates can be expressed as such: $I_t \in \mathbb{R}^{n\times h}$, $F_t \in \mathbb{R}^{n\times h}$, $O_t \in \mathbb{R}^{n\times h}$. Given this notation, the calculations looks like this:

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i) \tag{6}$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f) \tag{7}$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o) \tag{8}$$

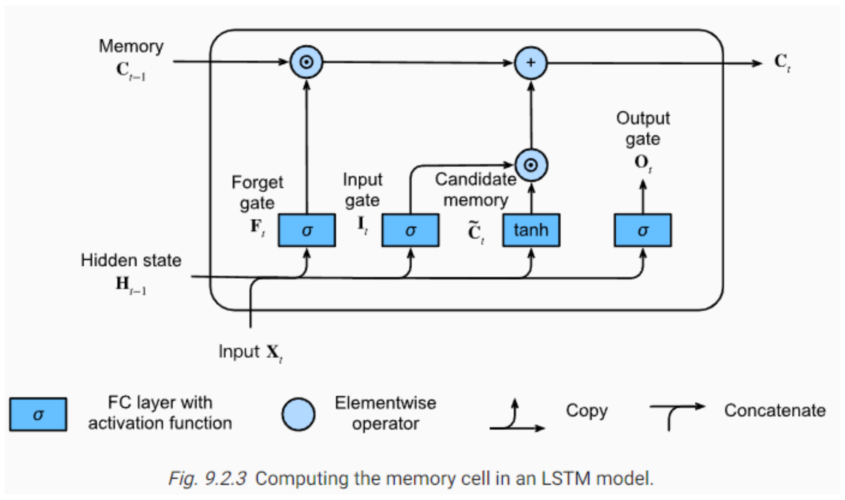


Figure 16: Illustration of how the memory cell of the LSTM model is computed

The input and forget gates are controlled by two instances of a memory cell. The candidate memory cell $\tilde{C}_t$ controls the affect of new data from I_t. The forget gate controls how much information is kept from the previous memory cell, $C_{t-1} \in \mathbb{R}^{n \times h}$. Thus updating the memory cell leads to the following expression:

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t \tag{9}$$

If F_t always approximates 1 and I_t approximates 0, C_{t-1} will be passed through to the current time step. This is how the LSTM model aims to overcome the vanishing gradient issue and better capture longer sequential dependencies.

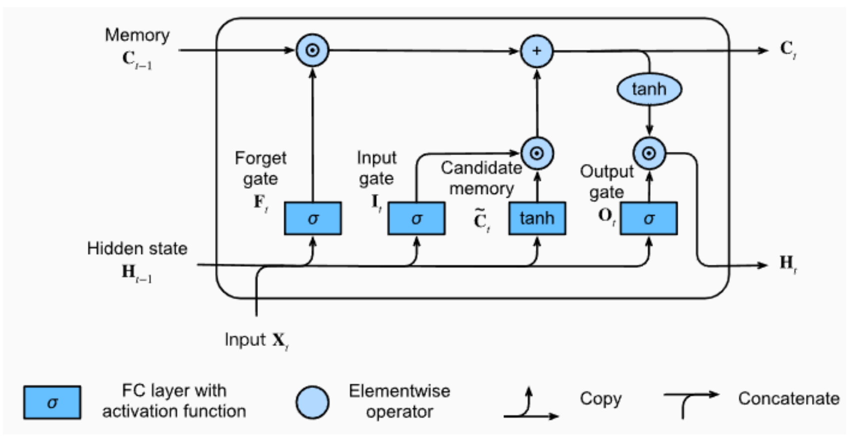


Figure 17: Illustration of a complete LSTM model

Finally the hidden state of the current time step, $H_t \in \mathbb{R}^{n \times h}$ needs to be computed. The current H_t is dependent on O_t and in the case of LSTM it is a gated tanh version of the memory cell, and is expressed as:

$$H_t = O_t \odot tanh(C_t) \tag{10}$$

6 Implementation

With an understanding of the how the models we want to implement works, it is time to implement and parameter tune these models. In order to understand the performance of the models during testing, the observed data from DMI will be split into two datasets. One for training and one for validation, this is generally best practice and an industry standard. In the case of this project the training data will be the data from 00:00 01-01-2010 to 23:00 31-12-2017 and the validation set will consist of the data from 2018 and 2019. This is the same year as the test set, which is the forecast data from the year 2019, but it should not be a problem when the data is not the same. This will actually give us a way of seeing how much the forecasts affect the performance of the models since the energy consumption is the same.

Other than these two measures for error and goodness of fit, the models will be compared to a very naive model or base line. Which will consist of taking the value for consumption from the same time, the day before. Expressed as the prediction made for time step $\hat{T}_{24(5-1)+8,20}$ will be compared to the actual value from $T_{24(4-1)+8,20}$. The naive model is to get a bare minimum threshold for the performance of the model, which should give a frame of reference for the implemented models performance.

6.1 Error metrics

After having acquired an understanding of how the both XGBoost and LSTM's work, it is time to implement and train them on the data from Energinet and DMI. During training and parameter tuning of the models it is important to be able to have some form of error metric from which you can interpret the performance of the models. It is also important to choose an error metric that allows for comparison between models. For this project, two error metrics was chosen, Mean Squared Error(MSE), and the same as was used in the data subsection, R^2.

MSE was chosen since it gives an actual value for how much the models predictions deviate from the actual values. It is calculated with the following formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{11}$$

R^2 was chosen since it gives a value between 0 and 1, which is allows for easy comparison between different models. In simple terms, R^2 is interpreted as a percentage for how much of the variance in the dependent variable can be explained by the independent variables in the predictor. R^2 can be expressed as such:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y}_i)^2} \tag{12}$$

6.1.1 Parameter Tuning of XGBoost

Achieving a well performing model in machine learning can sometimes be a tough task, dependent on many different aspects such as quality and amount of data, choosing the right model for the problem at hand. Afterwards, implementing the chosen model correctly can turn out to be quite the task as well. The goal is of parameter tuning is to make you model learn as much as possible on the amount of data you have available, but still avoiding that the model overfits on the training data and then performs very poorly on new data. Even though the XGBoost is perceived as being a model that is well performing "out of the box", it still contains parameters that needs to be tuned to arrive at the best performance. Choosing the parameters to tune was done by reading the documentation for the XGBoost, and then pick the ones that seemed relevant for this type of problem.

The first parameter was the max depth of the tree and the number of boosting iterations. These two are dependent on each other, since if you add more depth to the individual trees the model needs more iterations to generalize over the data. However, deeper trees have a higher chance of overfitting. next it was the step size shrinkage, or the learning rate, this is a parameter that aims to make the model more conservative and reduce overfitting. Then γ was chosen since it is the threshold of minimum loss required to make further partitions of the tree. Increasing γ makes the model more conservative. Lastly, the parameter minimum child weight was chosen, this sets the threshold for the minimum sum of weight needed in a child, this does also help regulate for overfitting. With 5 different parameters, that each will have different values that are optimal, this would require an extensive parameter search if it was doing manually. However, since XGBoost is very fast to train and test, this was done in a nested for loop. A set of possible values were chosen for the model to go through for each parameter. The model then trained for all possible combinations and stored the results in a dataframe with the respective values for all parameters. This might sound like an extensive way of brute forcing the parameter search like this, but with the efficiency of the XGBoost it only took an hour to run it through. The benefit of running the parameters search like this, other than ensuring we get the best model, is that this allows for inspection of the impact of the parameters on the result.

Table 6: Table of the parameters that were set in the search for the best possible setup of the XGBoost model

Parameter settings used in parameter search				
Tree depth	Learning rate	γ	Min child depth	Iterations
6	0.01	4	3	100
8	0.03	6	5	250
10	0.1	8	8	500
				1000

Looking at optimal parameter settings for both zones it would appear that increasing the tree depth further should increase the performance of the model. This was tested afterwards in separate tests and did not result in any increase in performance of the model. Thus the final parameter setup for the XGBoost model for DK1 and DK2 are presented in Tabel 7.

Table 7: Optimal settings for the XGBoost model for the two zones DK1 and DK2

Zone	Max Tree Depth	Learning Rate	γ	Min Child Depth	Iterations
DK1	10	0.01	4	8	500
DK2	10	0.03	6	8	100

6.1.2 Encoder/Decoder LSTM model

As seen from the literature review, there are near endless directions one can go when setting up a neural network. With the most important feature of the training data being time, it was important to choose a setup that is able to account for that. Therefore, by suggestion from Energinet, the setup that was chosen for the model was an encoding and decoding LSTM setup. This allows for the LSTM model to become bi-directional by taking in past values, and store the information from previous time steps in the memory cell and using it along with future time steps.

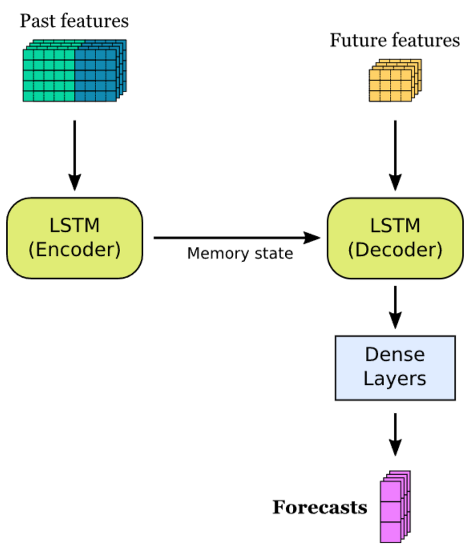
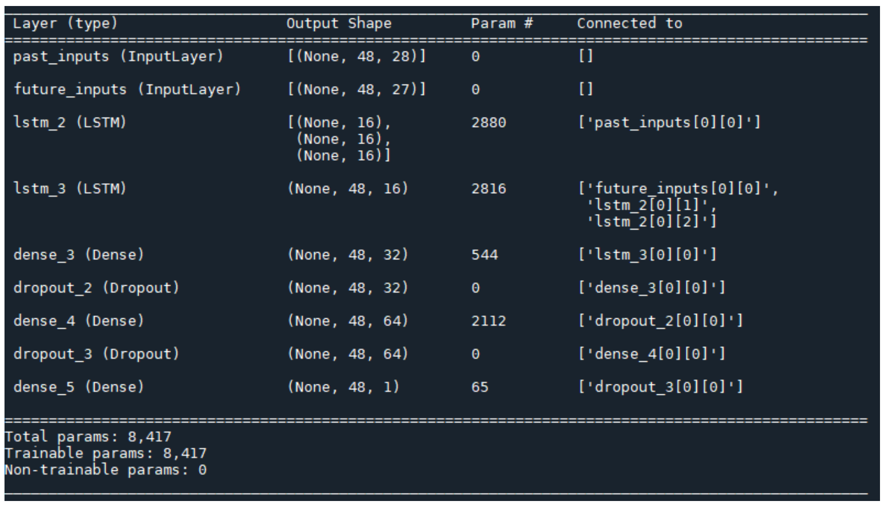


Figure 18: Illustration of the encoder decoder implementation in the LSTM model

The implementation used in this setup is 48 hours backwards, and 48 hours forwards. This is a hyper parameter that will have to be set for the final LSTM model. Since it is difficult to set a preferable previous time step that is optimal for the model prior to training it. With a general structure of the

LSTM, it is time to prepare the observed data for the purpose of being used in a NN. Two steps are needed; the data needs to be normalized between 0 and 1, and categorical values needs to be one-hot-encoded. It is widely considered best practice to normalize you data before using it with a NN, reasons being that it speeds up the training time and the way the activation functions inside the hidden layers calculate weights. The normalization of the dataset is done with package from SKLearn, called MinMaxScaler. This package just performs a regular minmax scaling on the dataset, but it lets us store the scaling parameters in a variable inside Python, which makes it easy to re-scale the predictions back to regular values again afterwards. The one-hot-encoding is performed on the feature for time of day. It is a categorical feature without ordinal relationship between the 24 values that represent the hours in a day. In practice this means instead of having one column for time, the dataset now contains 24 columns representing on hour each, containing binary values.



```
Layer (type)                 Output Shape         Param #     Connected to
==================================================================================================
past_inputs (InputLayer)     [(None, 48, 28)]     0           []

future_inputs (InputLayer)   [(None, 48, 27)]     0           []

lstm_2 (LSTM)                [(None, 16),         2880        ['past_inputs[0][0]']
                              (None, 16),
                              (None, 16)]

lstm_3 (LSTM)                (None, 48, 16)       2816        ['future_inputs[0][0]',
                                                               'lstm_2[0][1]',
                                                               'lstm_2[0][2]']

dense_3 (Dense)              (None, 48, 32)       544         ['lstm_3[0][0]']

dropout_2 (Dropout)          (None, 48, 32)       0           ['dense_3[0][0]']

dense_4 (Dense)              (None, 48, 64)       2112        ['dropout_2[0][0]']

dropout_3 (Dropout)          (None, 48, 64)       0           ['dense_4[0][0]']

dense_5 (Dense)              (None, 48, 1)        65          ['dropout_3[0][0]']

==================================================================================================
Total params: 8,417
Trainable params: 8,417
Non-trainable params: 0
```

Figure 19: Table of the layers and parameters for the LSTM model

The two first layers of the model should be the encoding and decoding LSTM layers. These layers functions as the models input layer and has the windowed shape: $y_{encoder} \in \mathbb{R}^{48\times28}$ and $y_{decoder} \in \mathbb{R}^{48\times27}$, to fit the previous time steps and the future time steps without the consumption feature. In this implementation the output from the decoder layer is then fed into two consecutive dense layers containing the activation function called rectified Linear Unit, (ReLU). The output layer of the model is a 1-dimensional dense layer, also containing a ReLU activation function. In between all these layers are dropout layers with a 0.2 setting in order to combat overfitting. What dropout layers does is turning off a given percentage of the neurons during testing. This prevents the neurons from becoming dominant in specific cases and forces the weights of the other neurons to account for the information not captured by the turned off neurons.

Table 8: Parameter settings for LSTM Model

Zone	Past input	Dropout	Activation function	Optimizer	Momentum	Learning rate	Loss function	Epochs
DK1	48	0.2	Relu	SGD	0.1	0.01	MSE	700
DK2	48	0.2	Relu	SGD	0.1	0.01	MSE	1000

Next it was time to choose the optimizer for the model, here two different optimizers were tested. Firstly the ADAM was tested, but this would cause the model to overfit instantly after one or two Epochs. Afterwards a Stochastic Gradient Descent(SGD) optimizer was implemented, which, at first also overfitted, but lowering the momentum of the SGD enabled to model to train for longer, with 0.1 ending up being the optimal setting for the model. The learning ended being a 0.01, smaller ones were investigated, but the training time would be increase significantly and would not converge.

Initially the model was also implemented with early stopping with a patience of 5. However, when the model would stop training at around 150-200 epochs it was not able to predict the maximum and minimum values for the validation set. From experience at Energinet, it was suggested that letting the model train as long as it does not start overfitting. Looking at the graphs in Figure 20 and 21, it is clear that letting the model for DK2 train for longer yielded a significantly increase to the performance of the model.

For the DK1 implementation it started overfitting at around 700 epochs, and as it can be seen from Figure 22, the model was not able to achieve the same performance for DK1 as for DK2.

Validation Results Now that the parameters have been tuned, the error metrics can give insights to the performance of the models on the validation

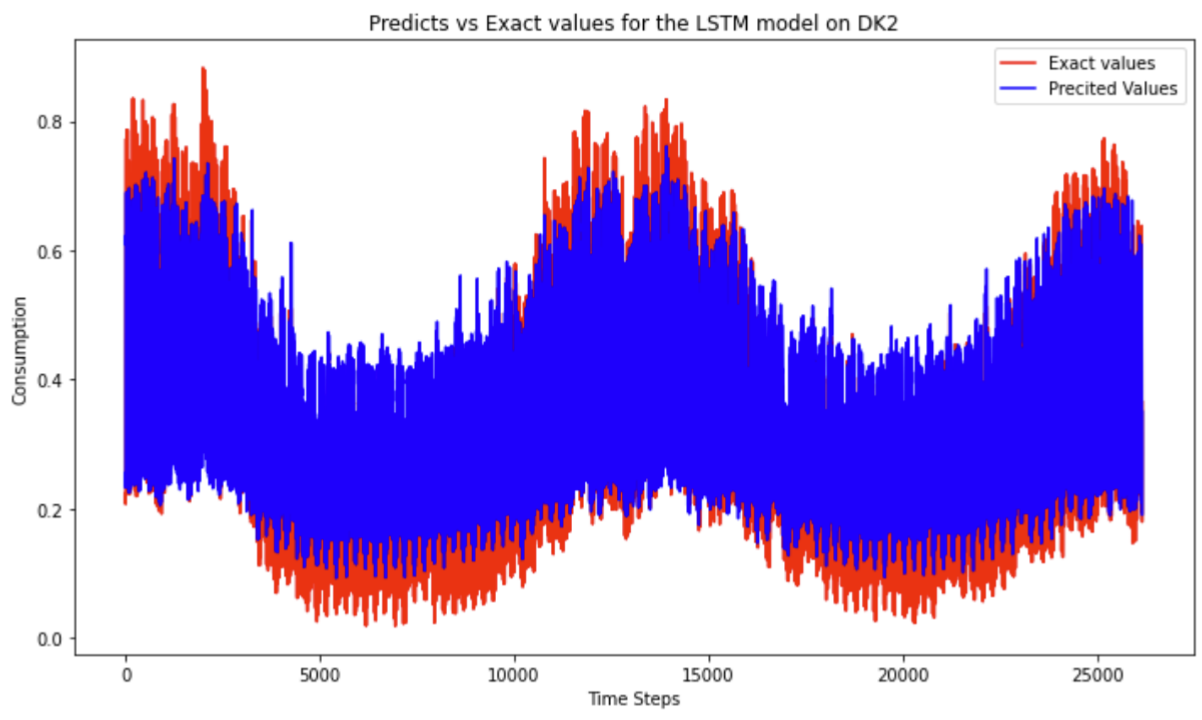


Figure 20: Plot of the predictions done by the LSTM and the real values, for the validation set with early stopping enabled which did approximately 150 epochs

set.

Table 9: Comparison of results for the XGBoost and the naive model

Results of predictions on the validation set			
Model	zone	MSE	R^2
XGBoost	DK1	18645.2	90.5
LSTM	DK1	22104	88.7
Naive model	DK1	67384.9	64.7
XGBoost	DK2	7187.5	92.2
LSTM	DK2	7095.9	92.3
Naive model	DK2	13714.9	84

Inspecting the metrics from table 9, the performance of the XGBoost and LSTM models appears to be very close, for DK2 the difference is only 0.1 for the R^2 score. Looking at the R^2 for the naive model, for DK2, it seems to perform rather well. But from the MSE it is apparent that the trained models still almost halves the error value. Another interesting find from the table is the difference in the performance of the naive models. It would appear as

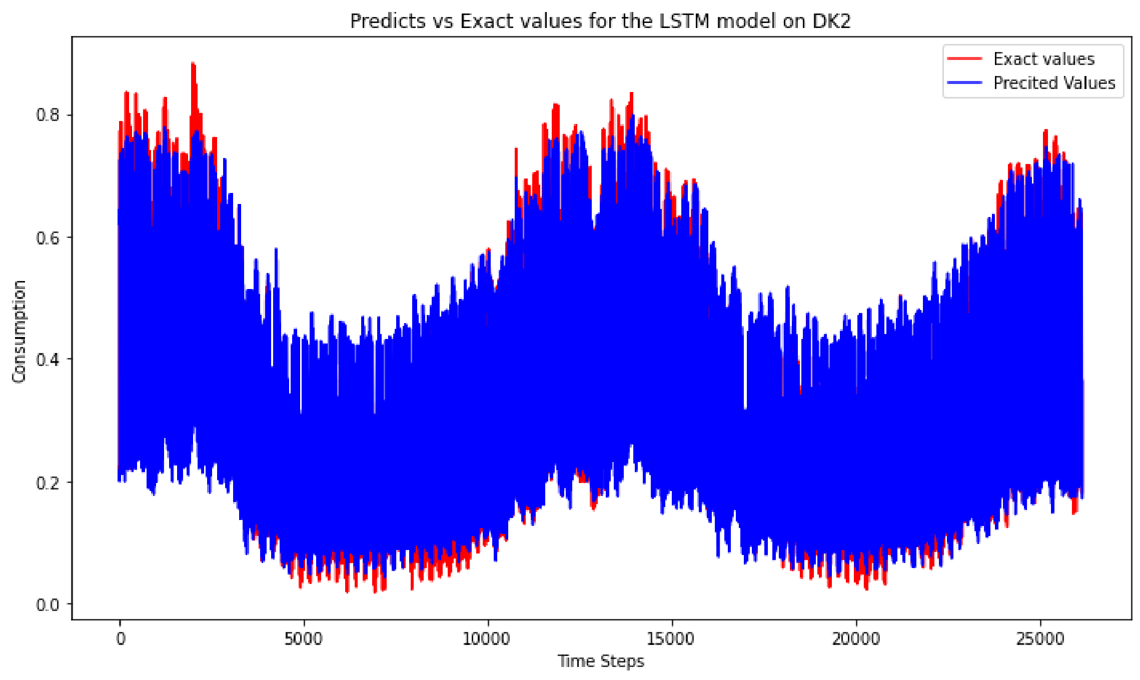


Figure 21: Plot of the predictions done by the LSTM and the real values, for the validation set doing 1000 epochs without early stopping

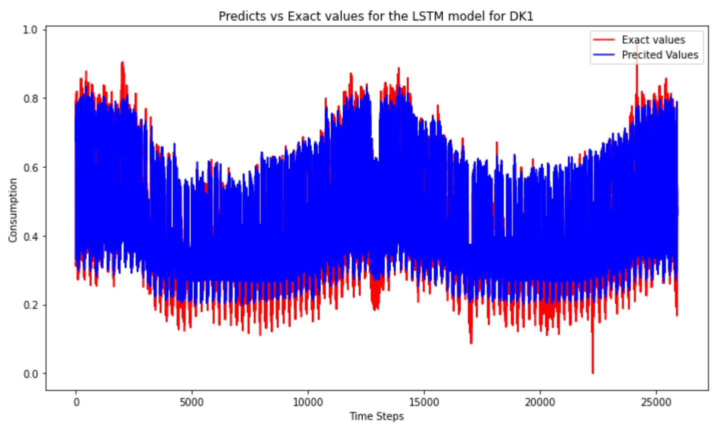


Figure 22: Plot of the predictions done by the LSTM and the real values, for the validation set for the DK1 model.

if the electricity consumption for DK1 has a much higher variance than the one for DK2. The variance for DK2 is 1554.8 and 2025.6 for DK1. So the difference in performance of the naive models does make sense, the positive take away from this is that both the LSTM and XGBoost is able to achieve similar performances despite the difference in the consumption pattern.

7 Results

The predictions in this subsection of the project will be done solely on the forecast data for the year 2019. This is the data that the models will have to predict on in a real world scenario if it is implemented at Energinet. The error metrics for the models performance on this data will be same metrics used in previous predictions to ensure the new performance metrics will be relatable to the earlier ones.

Table 10: Performance metrics for the models on the test set

Results of predictions on the test set			
Model	zone	MSE	R^2
XGBoost	DK1	33216.7	82.7
LSTM	DK1	33100.4	82.7
Naive	DK1	67384.9	64.7
XGBoost	DK2	9279.9	89.2
LSTM	DK2	17038.9	80.2
Naive	DK2	13714.9	84

From table 10 it is clear that most of the models perform significantly worse on the test data than the validation set. taking in to account the precision of the radiation forecast data, a drop in performance would have been expected, however a 12% drop for the LSTM model for DK2 makes is so it does not even outperform the naive model. Another interesting find is that the XGboost model for DK2 only dropped 3%, which is very interesting also. Because, the XGBoost model for DK2 is the one that trained the lowest amount of time. It was then suspected that the drop in the performance could be because of the models overfitting, but this turned out to not be the case. The XGBoost model for DK1 was trained and tested for less iterations than the 500 that was earlier described as optimal, which just led to even worse performance. Giving the XGBoost model for DK1 even more iterations also lead to a drop in performance. Higher regularization of the model was also tested but nothing helped, the values described earlier still came out as the optimal tuning for the DK1 model. The same thing was done for the LSTM model, by letting it train for lesser epochs. The LSTM models did not benefit from lesser training either.

In order to gain a better perspective on where the models are struggling, the models will be plotted and compared. For the test results, the hourly predictions have been calculated into a daily mean. This will give a better overview of the seasonal patterns and daily patterns of the models.

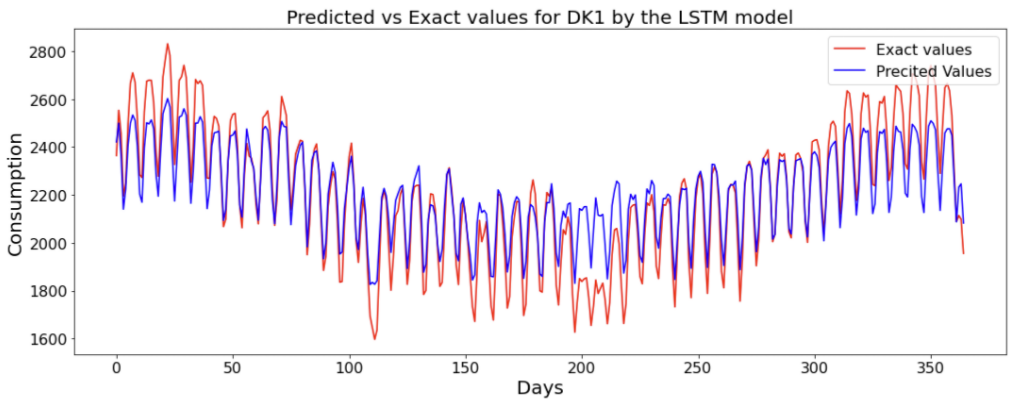


Figure 23: Plot of the predictions done by the LSTM and the real values, for the test set for the DK1 model.

Just as with the validation set, the LSTM for DK1 still suffers from the lack of ability to correctly predict the highs and lows of the electricity consumption.

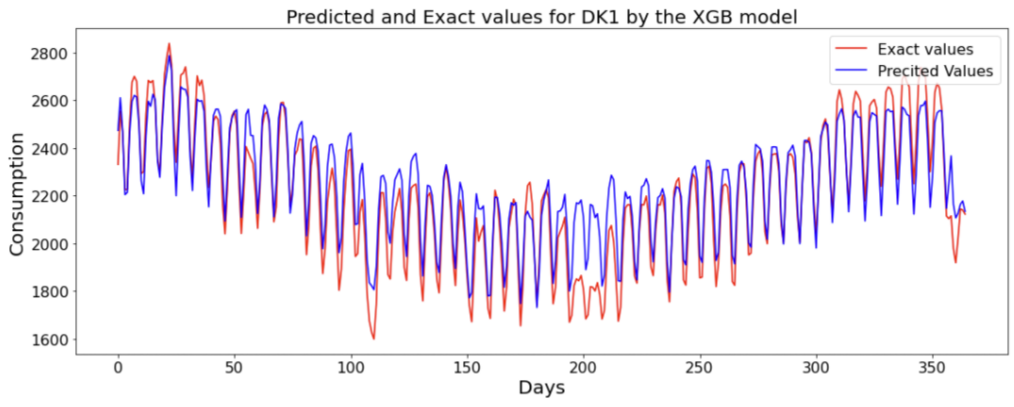


Figure 24: Plot of the predictions done by the XGBoost and the real values, for the test set for the DK1 model.

The XGBoost model for DK1 appears to better fit the high and lower values for the consumption. However, on the the individual days the LSTM performs better than the XGBoost model. This would possible be the reason they end up achieving similar results.

The LSTM model for DK2 is not able to fit the consumption at all over the months with low consumption. Even more surprising, it seems like the model also performs poorly on the individual days as well. The LSTM for DK2 also so a significant drop in performance when going from the validation set to the test set.

The XGBoost model for DK2, the best performing model, fit the data rather

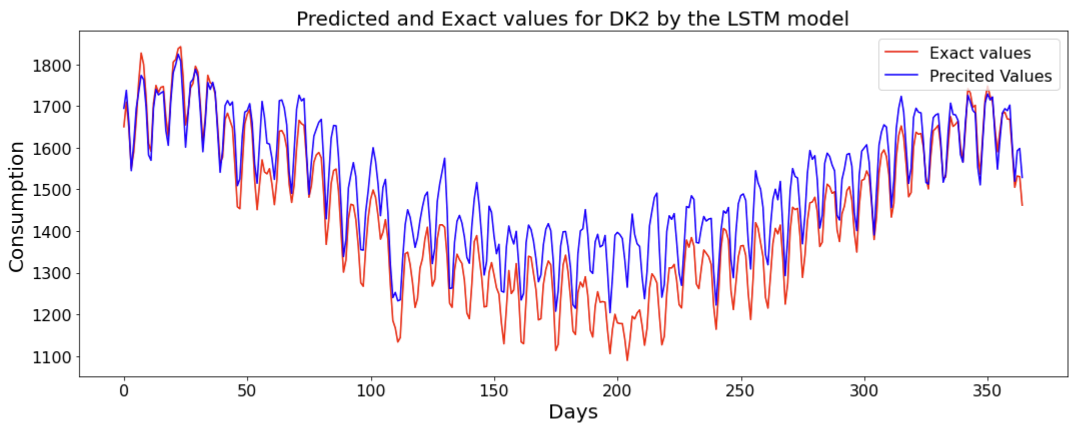


Figure 25: Plot of the predictions done by the LSTM and the real values, for the test set for the DK2 model.

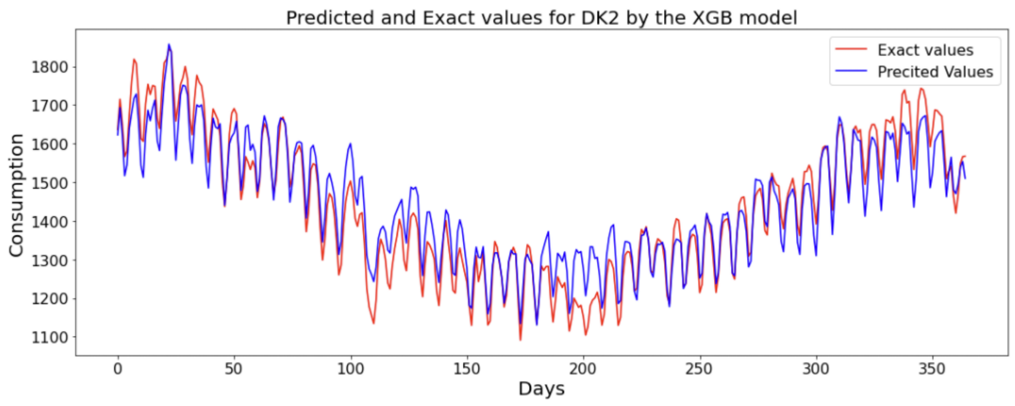


Figure 26: Plot of the predictions done by the XGBoost and the real values, for the test set for the DK2 model.

well. Something that all the models seem to struggle with is the consumption around 200 days into the year. There is a sudden drop in the consumption which none of the models are able to predict correctly. This sudden drop would highly likely be caused buy the summer vacation starting in Denmark. Danes usually have 3-4 weeks of summer vacation, so it would make sense that the consumption drops during the months of July and August. Many companies reduces production during these months, which is not a feature that either of the models takes into account. Generally the best performing model of these 4 are the one that trained the least amount of time. The XGBoost model for DK2 only trained for only 100 iterations and ended being the best performing model. The reason for this could be the other models overfit too much to the observed data and then ends up performing worse on the forecast data.

Because of the relative poor performance of the radiation forecast, it was briefly investigated if it would increase the precision of the forecasts if it was dropped. Removing the radiation parameter showed to further decrease the performance of the models.

It is also a possibility that the poor performance of the LSTM models is caused by an error in the code of the implementation when going from the observed data to the forecasts. However due to the limiting factor of time in this project, this will have be further investigated.

8 Discussion and Further Work

It should also be investigated if other features could be relevant for these models. The difficulty in this lies in the fact that the features needs to be available in advance. Energinet has access to many different kinds of data that relates to the consumption of electricity. Examples could be information about electricity by industrial codes, also residential and industrial consumption could be used, since the residential heating makes up such a large amount of the consumption. These features are not available in advance, but the LSTM might be able to use these features in the encoder layer. The models could also be trained using months as a feature since this might be able to pick up on the reduction in electricity consumption that happens during the Summer.

Looking at how precise the temperature feature is from the forecast data, and how the produced radiation forecasts tends to underestimate the radiation, it would be interesting to see the performance of the models if trained on the forecast data instead of observed values. If the plot of the forecast data is indicative of previous year's predictions as well, then it would be interesting to see the models performance if trained on this data instead.

9 Conclusion

The goal of this project was to explore if using weather data would have a benefit in producing accurate forecasts for electricity consumption. During data collection it was found that the forecasts Energinet receives from DMI were accurate enough to be used for as a feature in these prediction. There was an initial concern for the low R^2 score with radiation, but testing the models without this feature showed that it is indeed a benefit to the performance. The data was thereafter thoroughly visualized and explored, from which it was found that the biggest correlation is the time of day. this was further proven calculating the SHAP values for the features. The literature review showed that the industry standard for doing short term load forecasts for electricity was using LSTM's. Other articles suggested the use of the XGBoost algorithm for similar tasks, stating it being able to produce state-of-the-art predictions.

From the performance metrics of the models implemented it was found that 3 out of 4 models are able to produce predictions that outperform the naive model. With the best performing model being the XGBoost model, with the LSTM suffering from either bad implementation or lack of further tuning. These results were achieved using observed weather data during the training process, as well as forecast weather data as input during the actual predictions. Therefor, it can be concluded that weather data can benefit predictions for electricity consumption, given the consumption pattern that is present in Denmark at the moment.

References

Chen, Tianqi and Carlos Guestrin (2016). *XGBoost: A Scalable Tree Boosting System.* DOI: https://doi.org/10.48550/arXiv.1603.02754.

Energihåndbogen (2019). URL: https://evu.dk/wp-content/uploads/2019/06/Graddage.pdf.

Energistyrelsen (2019). *Energistatistik*, pp. 34–39. URL: https://ens.dk/sites/ens.dk/files/Statistik/energistatistik2019_dk-webtilg.pdf.

Friedman, Jerome H. (1999). "GREEDY FUNCTION APPROXIMATION: A GRADIENT BOOSTING MACHINE". In: pp. 1189–1232.

Ghalehkhondabi, Iman, Ehsan Ardjmand, and Gary R Weckman (2016). *An overview of energy demand forecasting methods published in 2005–2015.* DOI: https://doi.org/10.1007/s12667-016-0203-y.

Lundberg, Scott M. and Su-In Lee (2017). *A Unified Approach to Interpreting Model Predictions.* URL: https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

Ma, Yong (May 2022). "Full waveform inversion with image-guided gradient". In.

Román-Portabales, Antón, Martín López-Nores, and José Juan Pazos-Arias (2021). *Systematic Review of Electricity Demand Forecast Using ANN-Based Machine Learning Algorithms.* DOI: https://doi.org/10.3390/s21134544.

Runge, Jason and Radu Zmeureanu (2021). *A Review of Deep Learning Techniques for Forecasting Energy use in Buildings.* DOI: https://doi.org/10.3390/en14030608.

Tyralis, Hristos and Georgia Papacharalampous (2021). *Boosting algorithms in energy research: a systematic review.* DOI: https://doi.org/10.1007/s00521-021-05995-8.

Ziel, Florian (2018). *Modeling public holidays in load forecasting: a German case study.* DOI: https://doi.org/10.1007/s40565-018-0385-5.