Predicting Energy Behavior of Prosumers

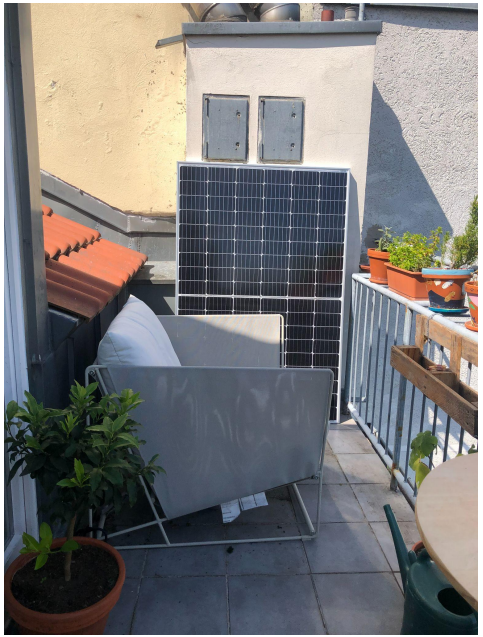
Final Project ML 2024

Theo, Xaver, Inigo, Alicja

12.06.2024

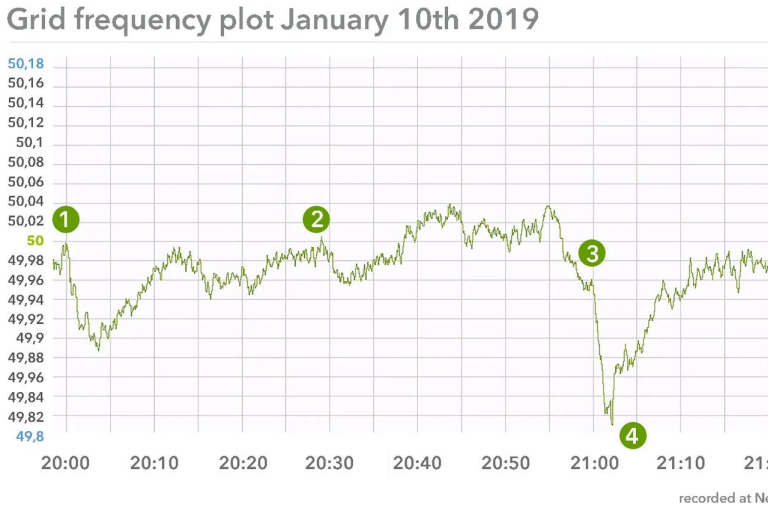
Prosumers

- Prosumers are individuals that produce and consume their own energy using renewable sources like solar panels
- Homeowners and businesses in Estonia with solar panels that use their energy and feed the surplus back to the grid
- Decentralized energy production poses challenges for grid management



Why do we care?

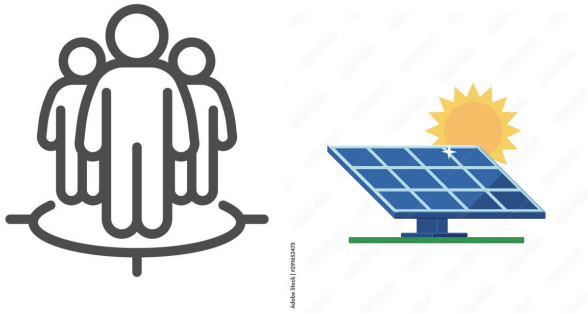
- The energy production and consumption must be in balance
- On the 10 of January 2019 we had a near blackout event in europe
- The subtitle in Kaggle: Predict Prosumer Energy Patterns to Minimize Imbalance Costs
- Behind the Kaggle competition is a Estonian renewable energy provider



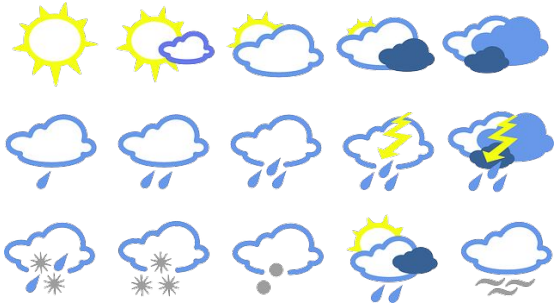
Our Data

- The data provided by Kaggle was across 7 data sets corresponding to:

Electricity and Gas prices



Client information (solar capacity, type etc..)

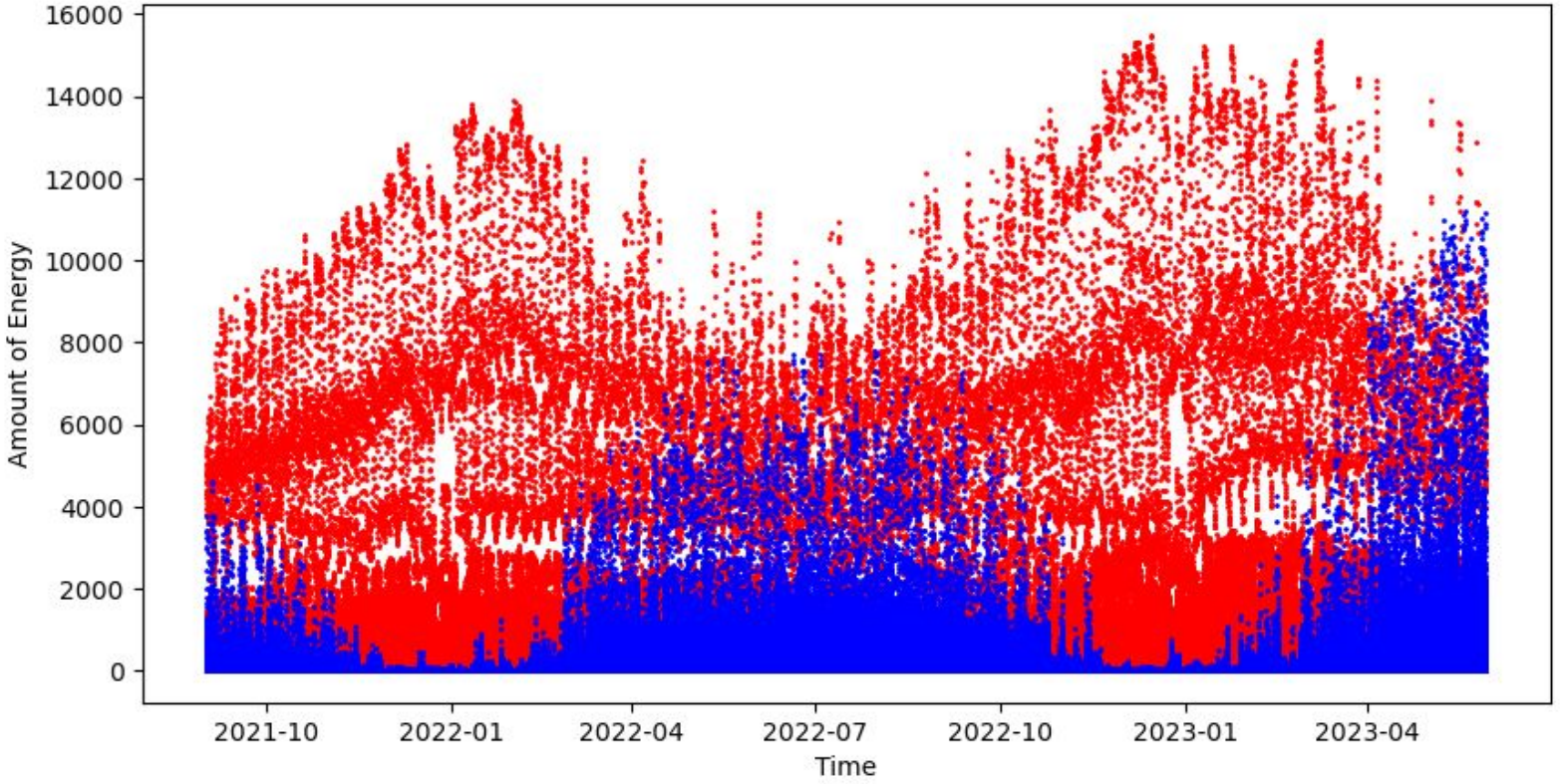


Historical and predicted weather data

Main things to note

- Hourly data across ~ 1.5 years
- When combined, dataset became quite large (1GB)
- 2 million rows, 54 columns
- Kaggle competition uses MAE for evaluation
- Turned datetime object into individual features: hours, days, weeks, months, years

Consumption and production



Quick LightGBM model - MAE scores

Data type	Single Model	Separate models
Entire data set	25.61	21.22*
Production	17.14	11.97
Consumption	34.10	30.47

* From (production + consumption)/2

Feature selection - what we think is important

Production

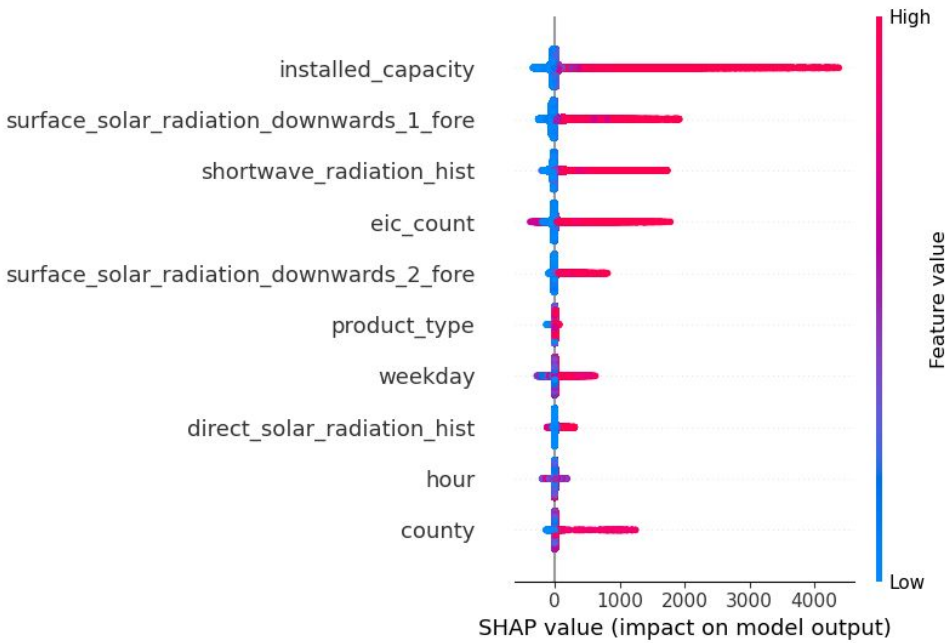
1. Amount of sun (direct_solar_radiation_hist)
2. Solar Capacity (installed_capacity)
3. Cloud cover (cloudcover_total_hist)

Consumption

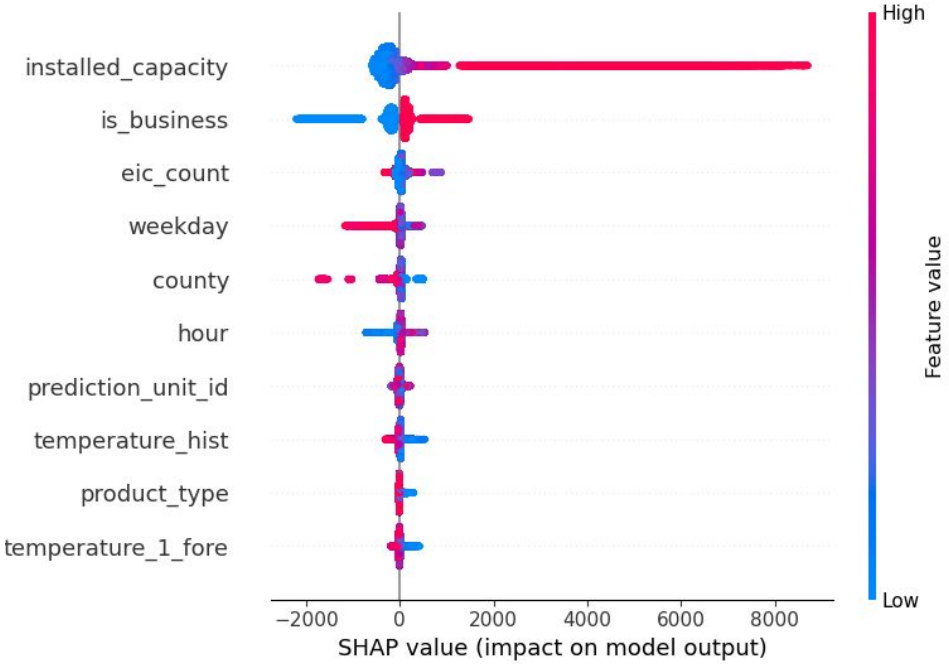
1. Time (hours)
2. Temperature (temperature_hist)
3. Electricity price (euros_per_mwh)

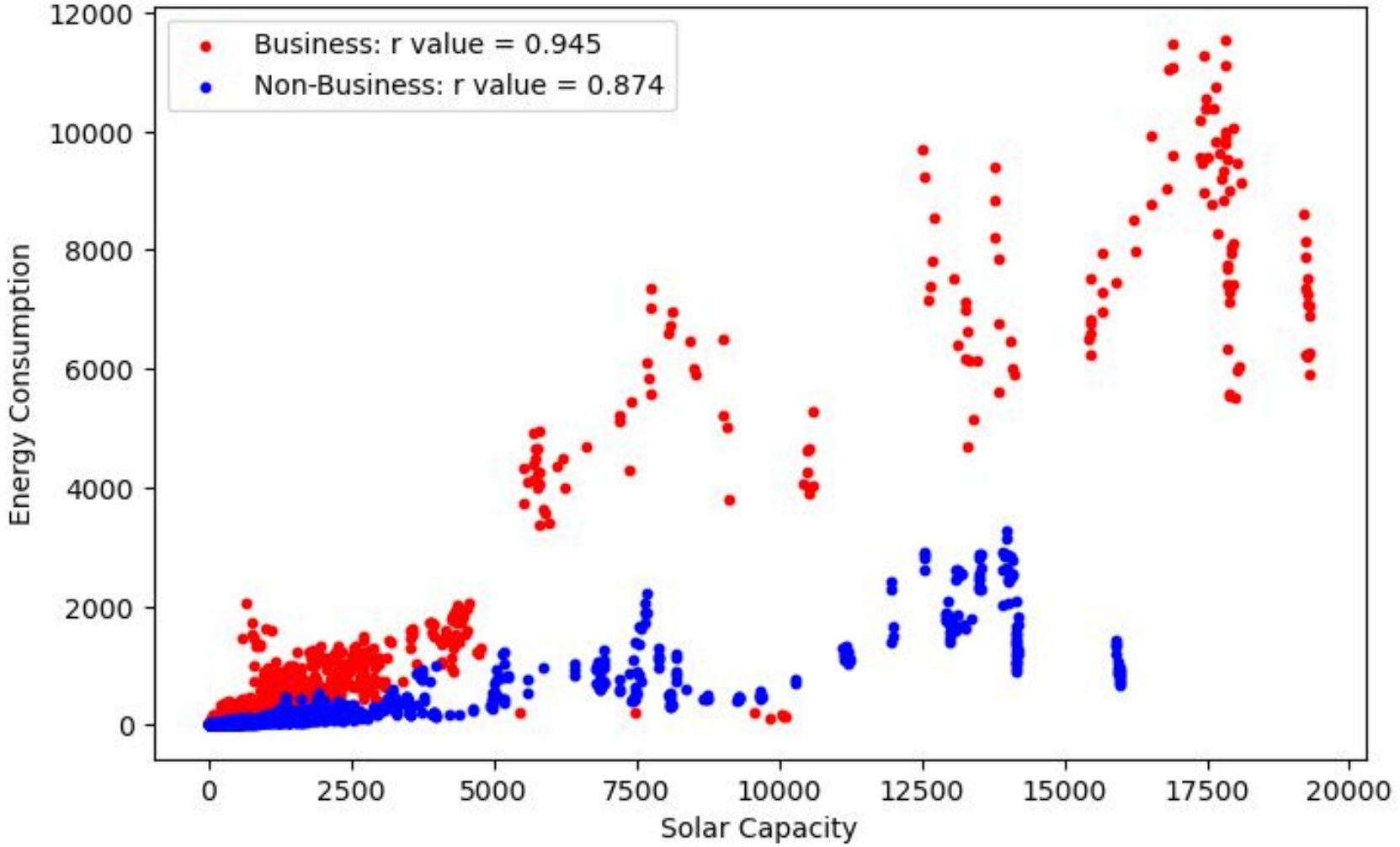
LightGBM + Shap

Production



Consumption




Business: r value = 0.945
Non-Business: r value = 0.874
Energy Consumption
Solar Capacity
0
2000
4000
6000
8000
10000
12000
0
2500
5000
7500
10000
12500
15000
17500
20000

MAE of different features comparison

	All features	Top 5 features	% Change
Production	11.97	24.72	106.52
Consumption	30.47	102.81	237.41

Why?

- Dataset lacks directly relevant features for consumption, which involves an unpredictable human element
- Production is more straightforwardly influenced by measurable factors such as solar capacity and sunlight.

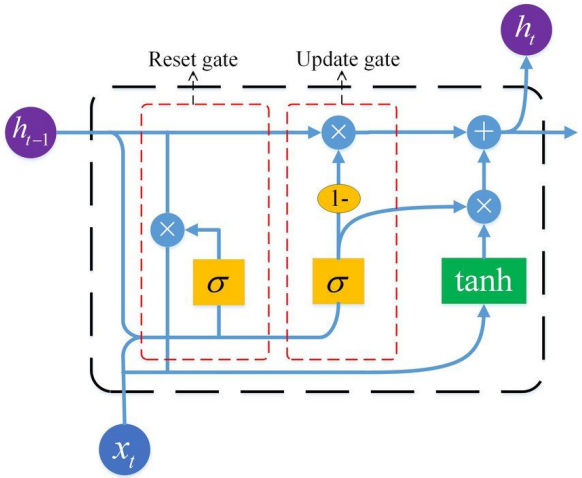
THE GRU (Gated Recurrent Unit)

GRUs are a form of RNN that are specifically designed for sequence prediction problems that uses gates to regulate information flow at each time step.

Effective in processing sequence data such as text or time series.

The primary function of a GRU is to mitigate the vanishing or exploding gradient problem with vanilla RNNs.

The update gate determines how much of the past information is passed on, and the reset gate decides how much of the past information is forgotten.



First attempt at a GRU

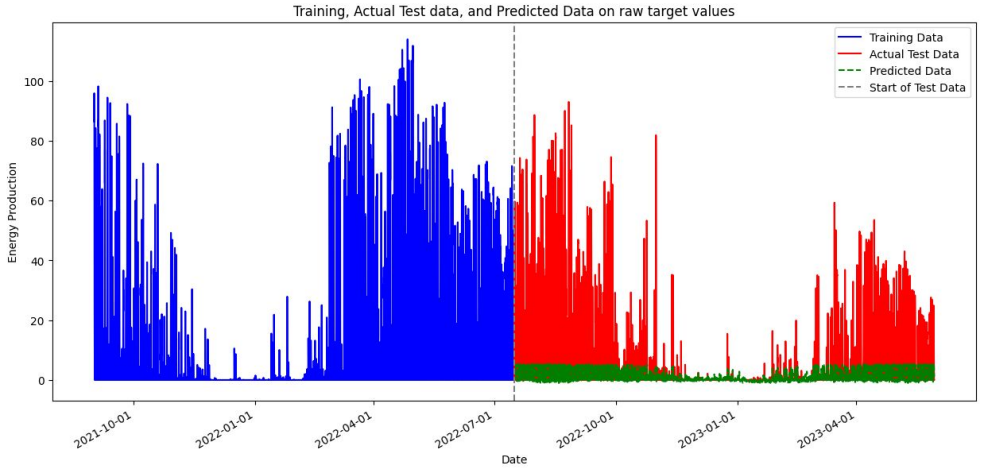
- Our GRU had sequenced data that took 24 hours and predicted the energy usage for the following hour
- At first we tried to build a single GRU for all entities that could group together relevant factors that identify each prosumer and predict their future energy usage.
- This took a lot of time and computational power (2 million entries).
- Could be possible with more time and resources (and knowledge).

GRU for one prosumer

After we tried a GRU for one prosumer, focusing on production.

This is a useful model for anyone that wants to predict their own future energy behaviour.

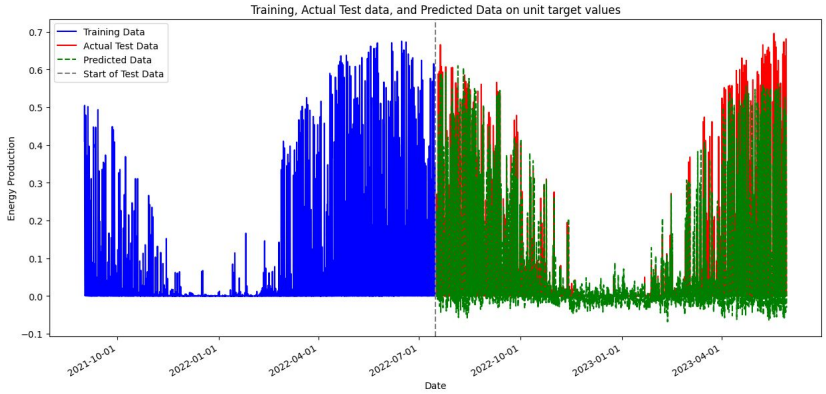
The data is over 1.5 years and was split into two halves. The training was done on the first half and GRU used to predict the second half.



Clearly, we can see that the GRU was not incredible at predicting the future energy production. Although it did catch on slightly to the seasonal trend.

Some revisions are needed.

On target



This is a visibly huge improvement! Now the GRU has success in predicting future energy production after only being trained one half the data for one prosumer.

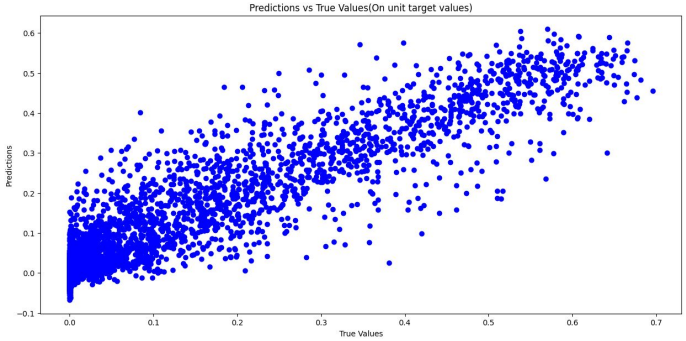
Mean Absolute Error for target/capacity predictions: 0.02919

A new target was created:

(target / installed_capacity)

This gave the energy production for unit cell.

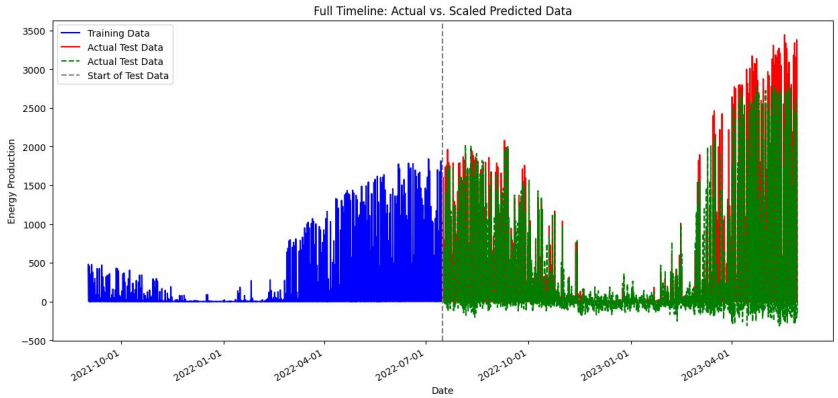
In theory this should improve the model as it would be easier to latch on to seasonal and weather trends.



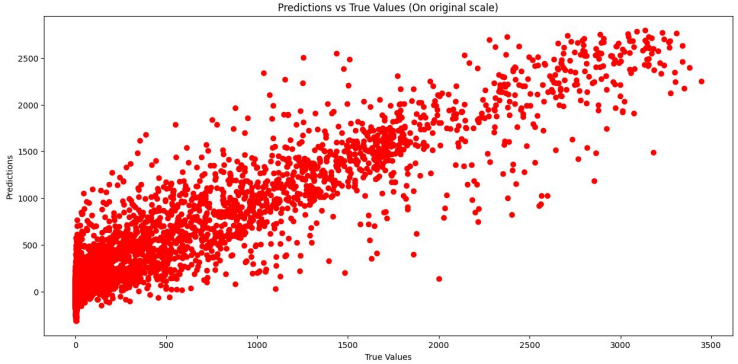
R2 Score (unit targets): 0.881494

CONGRUSION 👆

The predictions were then multiplied by the installed capacity in order to get real target values for a comparison.



Mean Absolute Error for rescaled predictions: 5.0241



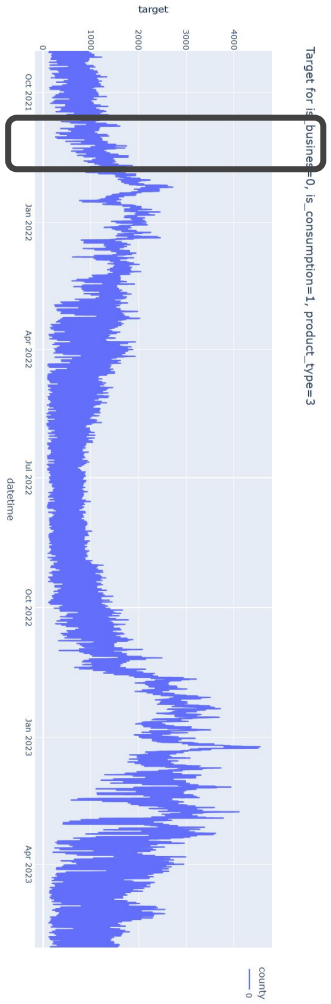
R2 Score (rescaled): 0.878696

From the tests we can see that gated recurrent units are extremely useful for predicting future data in a time sequence.

They do not need much training data. It successfully managed to notice patterns in the weather and other variables.

Time Series Forecasting

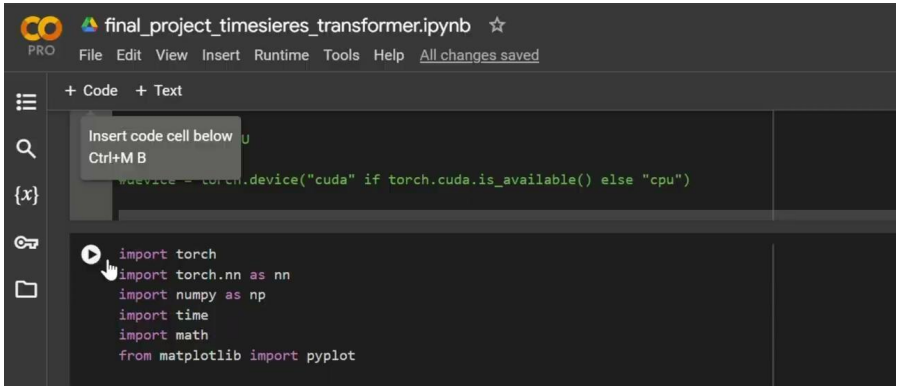
- Our next idea was to train a transformer for time series forecasting
- Transformers are effective for sequence modeling due to their ability to handle long-range dependencies
- We only train on the target timeseries (univariate)
- Univariate models often outperform multivariate models due to the difficulty in estimating cross-series correlations and the added variance from these estimates.
- Inspiration on a github page for long term river flow modeling



GPU acceleration

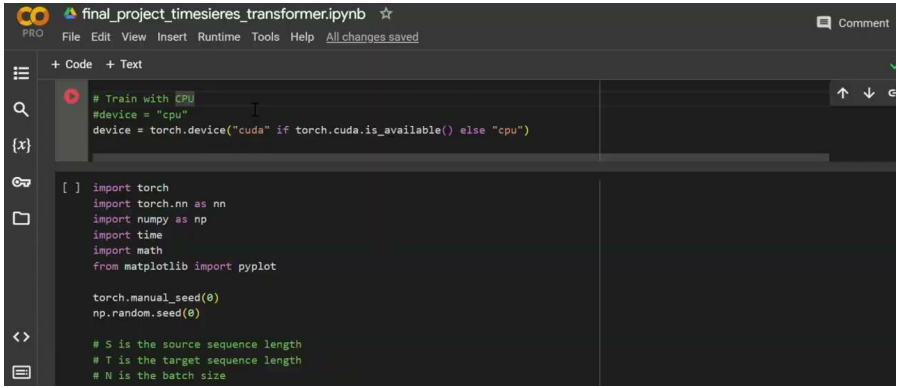
Run on “CPU”

- CPU is a Xeon Processors
- Relatively slow



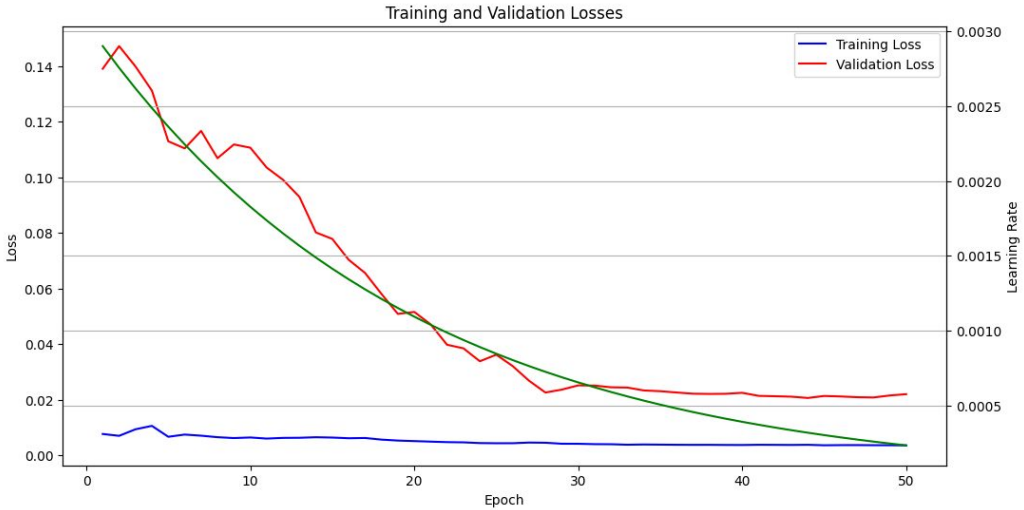
Run on “GPU”

- NVIDIA T4 Tensor Core GPU
- Benefits of parallel computing



Training and Validation Loss

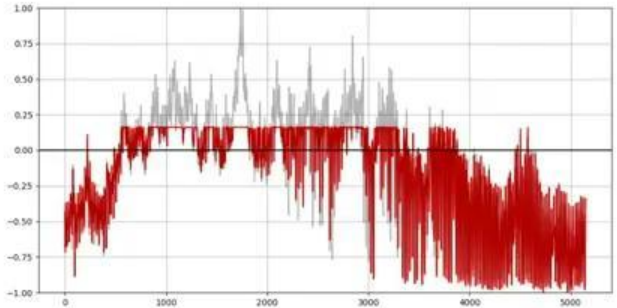
- We didn’t train on all the data
- We trained for 50 Epochs with Learning rate decay
- Validation loss is going down and stagnating in the end
- Training loss starts very low because we started with a high learning rate



Results (See a transformer learn)

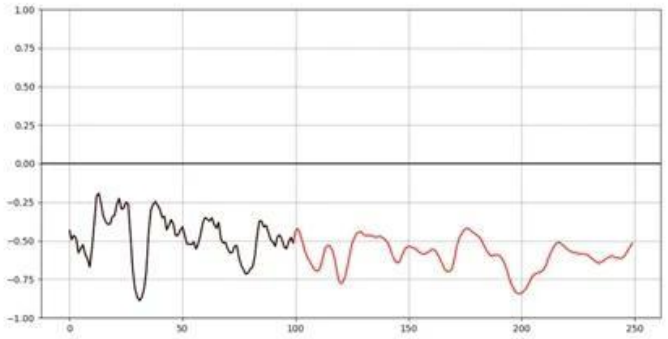
Training Predictions:

Grey is the Actual Time Series and red is the prediction while training

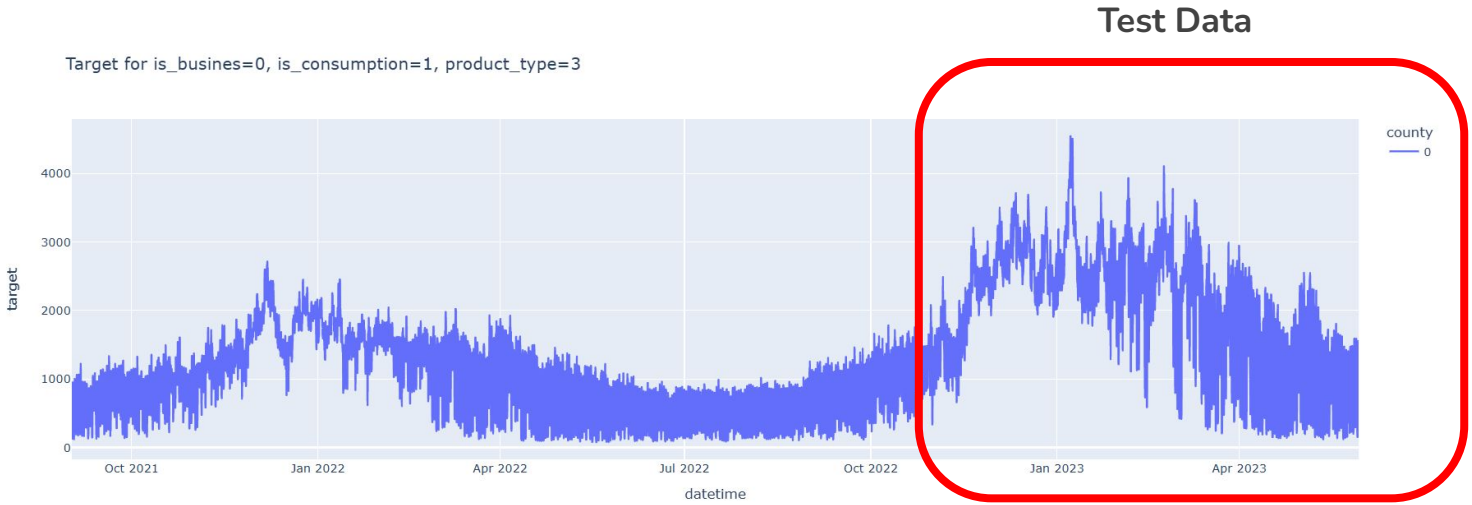


Future Prediction:

Here we give it a input window from the validation set and it predicts the next step based on the previous prediction



Comparison



Transformer: Mean Absolute Error (MAE) for the test dataset: 0.1075*

LightGBM: Mean Absolute Error (MAE) for the test dataset (LightGBM): 0.2129*

*(normalized targets)

Comparison with other results

1st Place:

- Model: XGBoost, GRU, XGBoost+GRU
- All models have 600 features

5th Place:

- Model: various but settled on LightGBM
- 75 production features & 85 for consumption

6th Place:

- Model: LightGBM and some ‘baseline models’, final was weighed combination of models
- Number of features used not directly given

7th Place:

- Model: few others tested but ended with LightGBM and XGBoost
- 192 features used for both

Conclusion

What was most important:

- *Data augmentation* to make the data more machine learning friendly,
  -> highly ranking features derived from ‘datetime’ object & GRU prediction scores

All models gave unique insights to the data and were useful for predicting different aspects.

- Tree based algorithms: best at full predictions of the data and could be run on all data seamlessly.
- GRU/Transformer: very useful for predicting a *single* business’ production/consumption in the future.

R^2 scores on predicted values against true values:

Production,Consumption

LGBM	GRU	Transformer
0.9881, 0.9967	0.8786*, N/A	0.8197*, 0.8479*

*Single business

Future Expansions

- Find how Kaggle competition ‘score’ is calculated & how to submit
  - *lot* of time spent investigating/trying but did not succeed
- Include extra data, eg. Estonian Holidays or create lagged features in model training
- Consider additional augmentation of other features, like the target/solar capacity

Thanks for listening 😀

Appendix

- Competition info
- Data/LGBM info
- Gru info
- Transformer info
- Issues with Kaggle

Extra info on competition

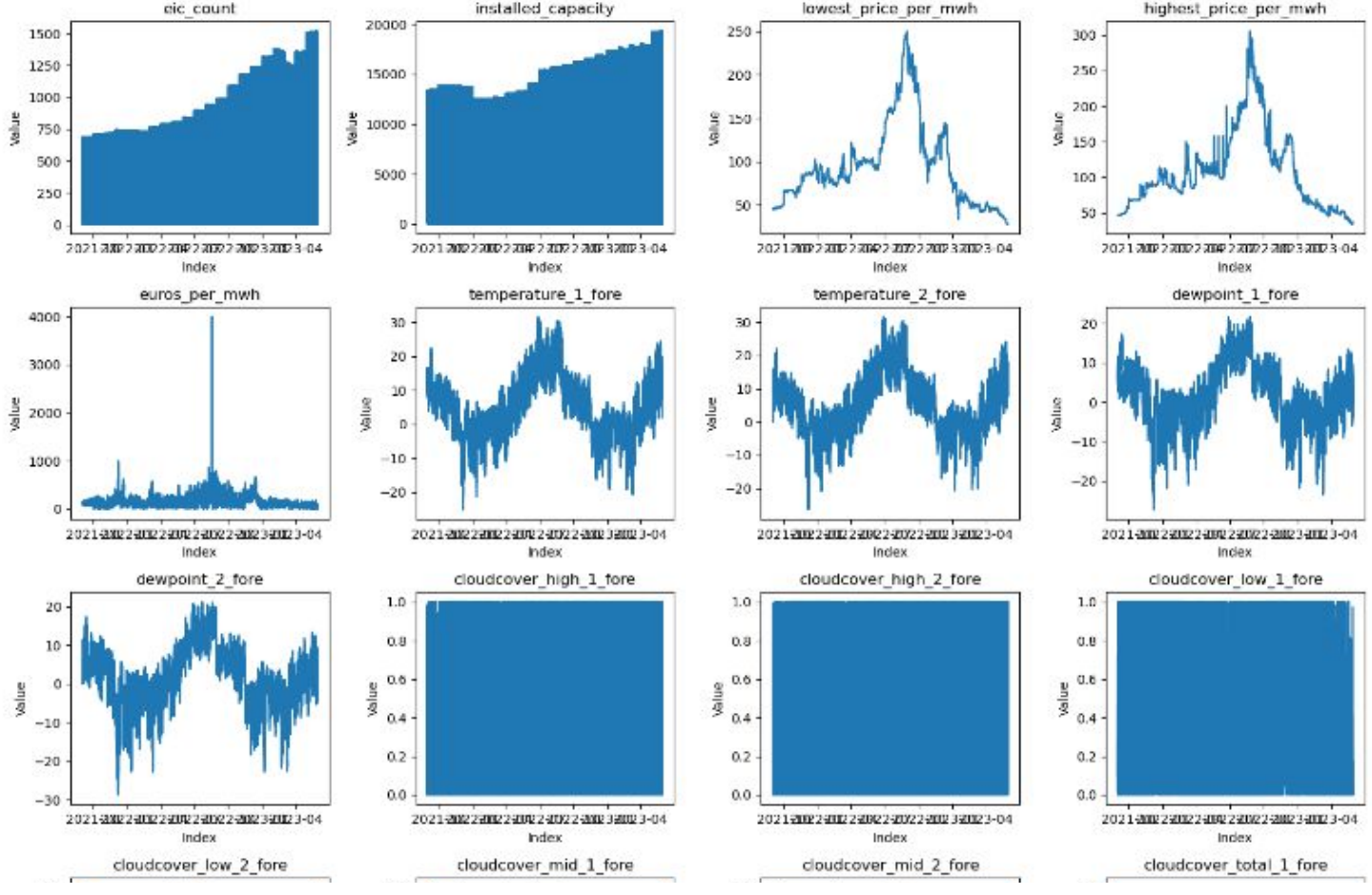
Enefit - Predict Energy Behavior of Prosumers

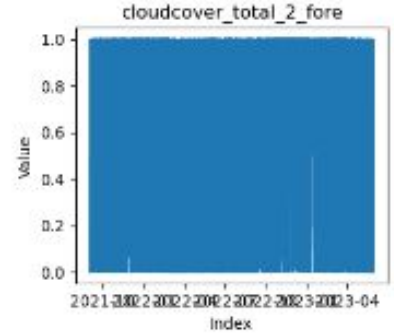
Enefit: one of biggest energy companies in the Baltic

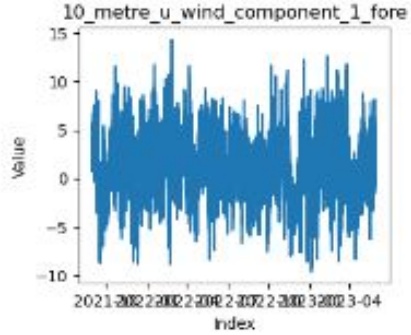
To improve efficiency [*eg. operation costs*, *grid instability*, *energy use*, *etc*] with rising amounts of consumers switching to being prosumers Enefit wished to better be able to predict the energy produced and consumed.

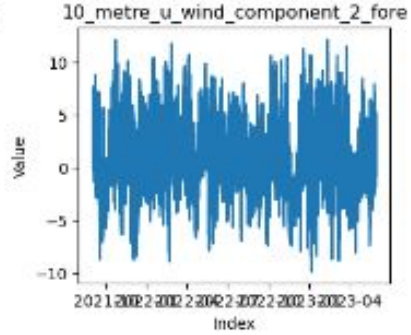
Competition worked with Estonia specific data

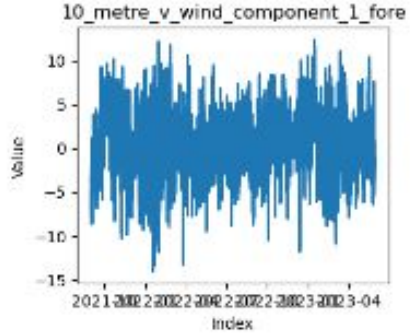
Data visualisation of *interesting* features

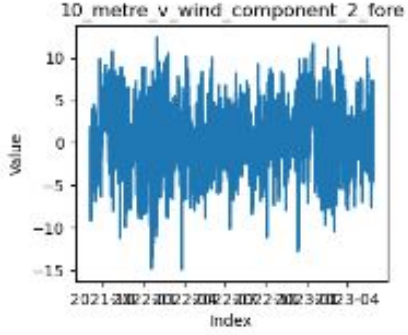


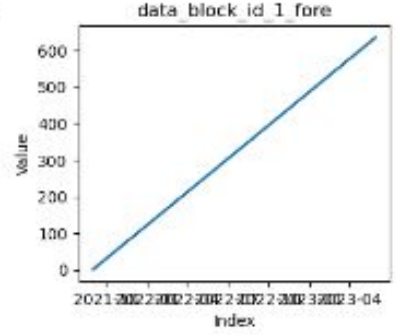

cloudcover_total_2_fore
Value
Index

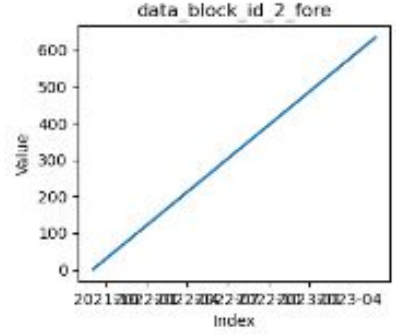

10_metre_u_wind_component_1_fore
Value
Index

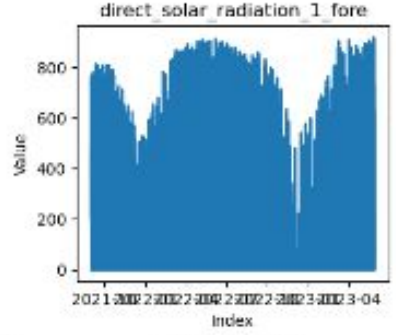

10_metre_u_wind_component_2_fore
Value
Index

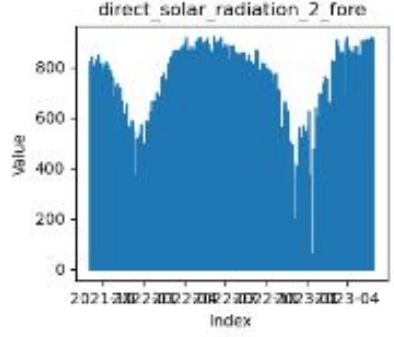

10_metre_v_wind_component_1_fore
Value
Index

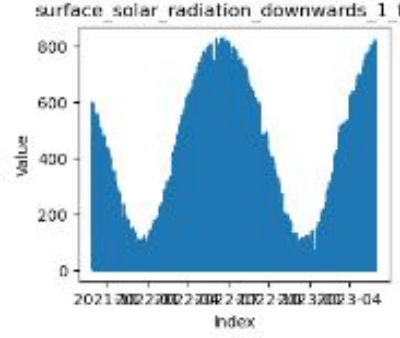

10_metre_v_wind_component_2_fore
Value
Index

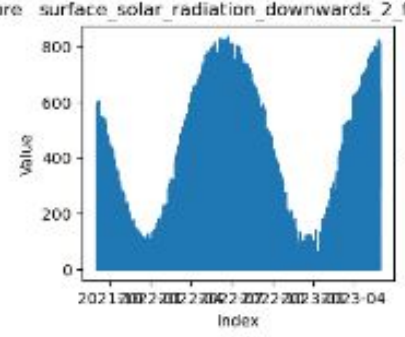

data_block_id_1_fore
Value
Index

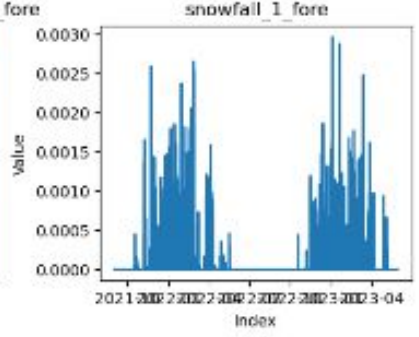

data_block_id_2_fore
Value
Index

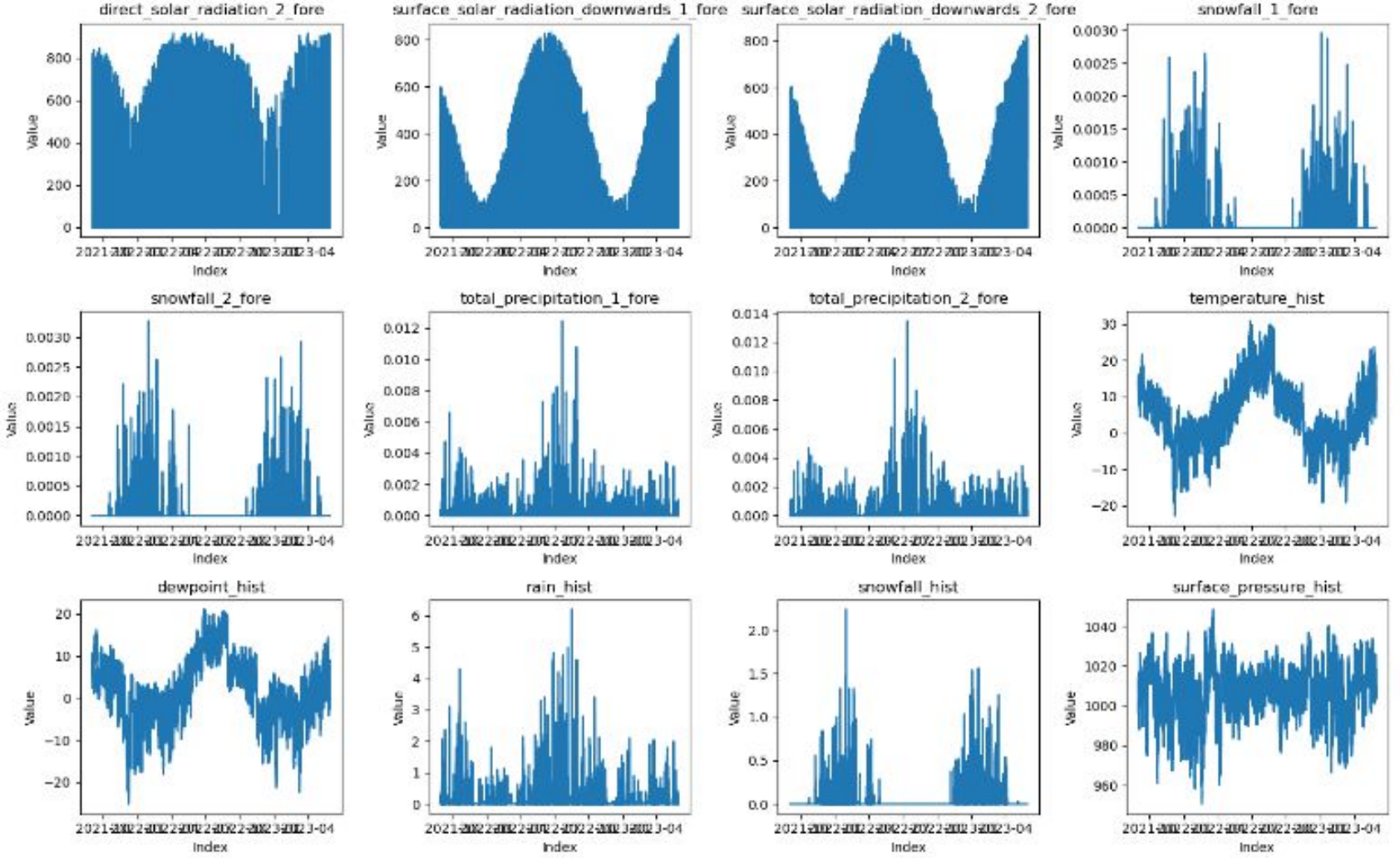

direct_solar_radiation_1_fore
Value
Index


direct_solar_radiation_2_fore
Value
Index


surface_solar_radiation_downwards_1_fore
Value
Index


surface_solar_radiation_downwards_2_fore
Value
Index


snowfall_1_fore
Value
Index


direct_solar_radiation_2_fore
surface_solar_radiation_downwards_1_fore
surface_solar_radiation_downwards_2_fore
snowfall_1_fore
snowfall_2_fore
total_precipitation_1_fore
total_precipitation_2_fore
temperature_hist
dewpoint_hist
rain_hist
snowfall_hist
surface_pressure_hist
Value
Index

LGBM info

To balance compute time and accuracy we used:

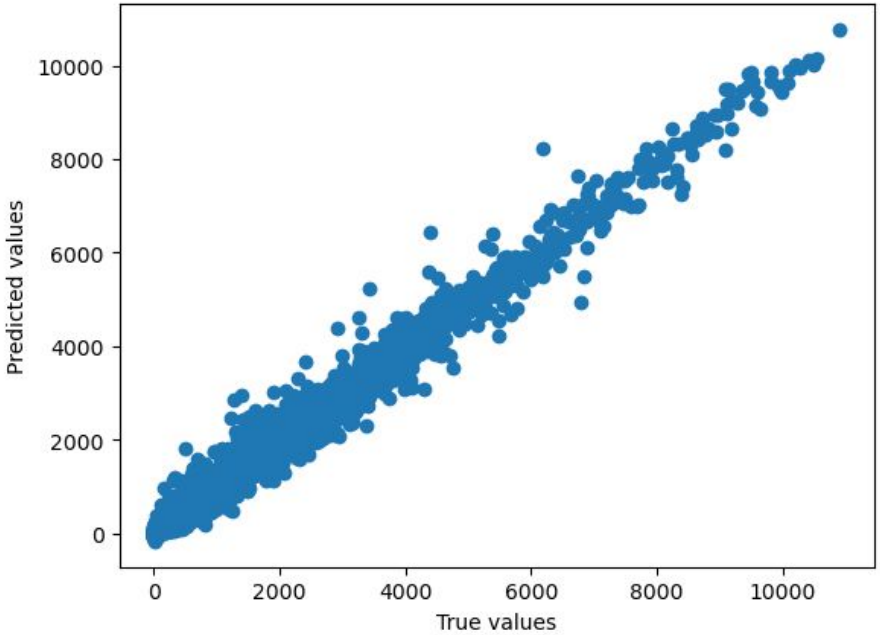
- n_estimators=1000
- num_leaves=100
- learning_rate=0.1

For all the LGBM-related data/plots/shap

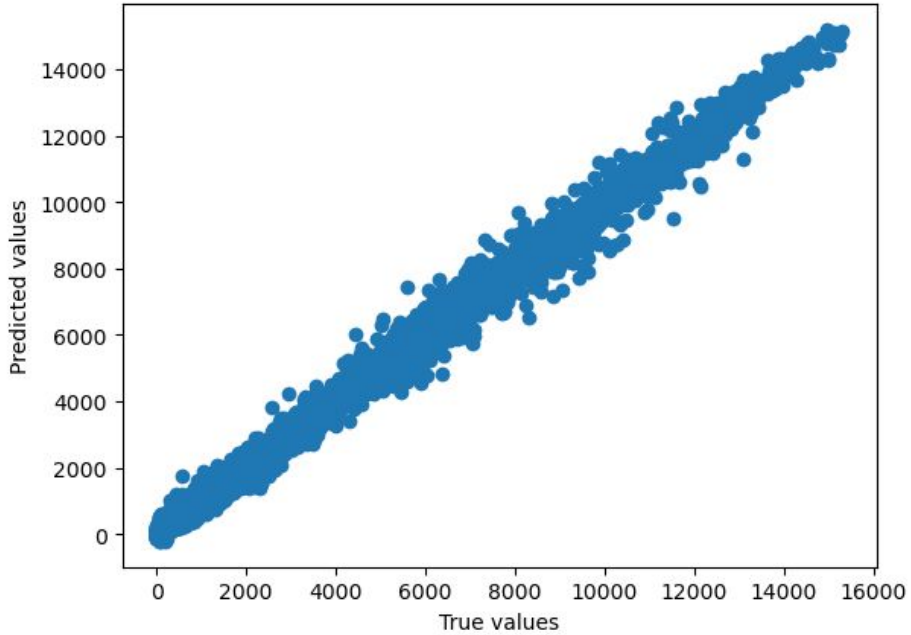
When trying out different parameters, significant improvements were made when making the model extremely complex (large number of leaves) even on the error of the validation set.

LGBM Tuner was investigated for hyperparameter optimisation however this was later discarded and focus was changed to the NN as the tree based algo was already performing very well and so didn't need improvement

Production: R2 = 0.9881



Consumption: R2 = 0.9967



Mean Absolute Error comparison with different number of features

	MAE (All features)	Top 10	Top 6	Top 5	Top 3
Production	11.97	18.72	24.72	24.73	31.42
Consumption	30.47	46.15	59.56	102.81	121.23

The jump from Top 6 to Top 5 for consumption in an interesting observation, this happened because that 6th feature was ‘hours’ aka time of day which evidently is very important, it is odd that shap didn't rank it higher. This further emphasizes the random/unpredictable nature of the consumption without better features.

GRU

One companies data found using one production_unit_id. Production data extracted from this.

The target changed to target/installed_capacity and the original target and capacity was held in an array for later conversion.

Categorical features preprocessed using OneHotEncoder. Numerical features preprocessed using StandardScaler.

Create_sequences function used to create X sequences over 24 hours and the Y value being the following target.

Training data used as the first half of the data and the test data used as the last half (in order of time).

Sequences converted to torch tensors.

Algorithm: PyTorch GRUNet(nn.Module)

Input dimension: features per time step

50 hidden units, 2 layers, and predicts one output per time step.

Idea for 2 layers taken from kaggle winner hyd.

Criterion = nn.L1Loss().

Learning rate = 0.001. Different learning rates attempted with this seeming most plausible.

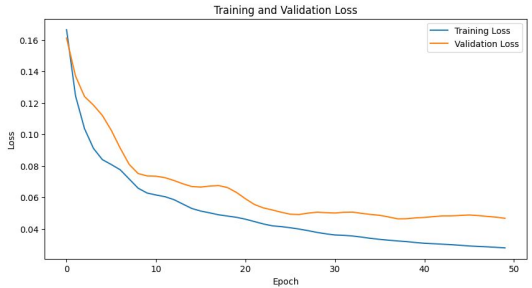
Initially ran over 50 epochs, as didn't improve much afterwards. More training and optimisation could be done to improve accuracy slightly.

predictions = model(X_test_tensor)

t/c mae = mean_abolsute_error(Y_test_tensor, predictions)

converted mae =mae(held_targets[test:],held_capacity[test:]*predictions)

torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1) -> gradient clipping applied



Early GRU attempt

At first a GRU was built to try and handle all data. The rows of data were in an order of each companies production and consumption values for that hour, and then it would move onto the next hour. This was hard to sequence effectively. It was attempted to group them like so:

```
[ ]  grouped = data.groupby(['is_business', 'product_type', 'county'])
     production_X_seqs, production_y_seqs = [], []
     for name, group in grouped:
         group.sort_values('datetime', inplace=True)
         X, y = group[X_variables], group['target']
         X_seq, y_seq = create_sequences(X, y, sequence_length=24)
         production_X_seqs.extend(X_seq)
         production_y_seqs.extend(y_seq)
```

This then gave rise to many other issues while training and proved very difficult to get a proper model working. It also took an immense amount of computational power. It should have been possible, and I think it would still be a valid way to handle this problem and it is still an idea to keep in mind when tackling sequential data like this.

Time Series Transformer

Used PyTorch

Algorithm: Transformer from https://github.com/oliverguhr/transformer-time-series-prediction

Hyperparameters:

batch_size = 10

input_window = 100 and 50

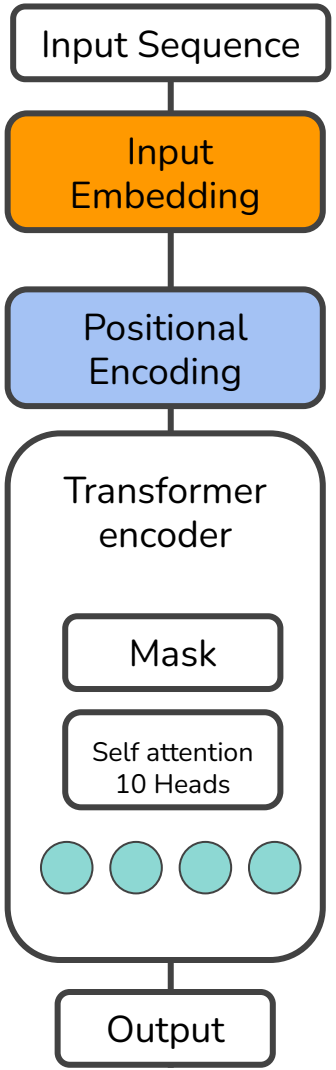
output_window = 1

starting learning rate = 0.0029 and 0.0005

Learning rate decay = scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1, gamma=0.95)

Number of heads = 10

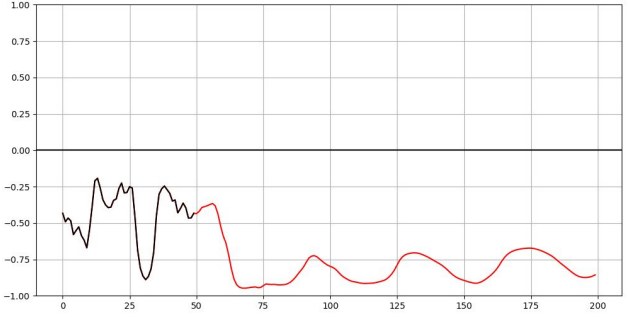
Number of features = 250



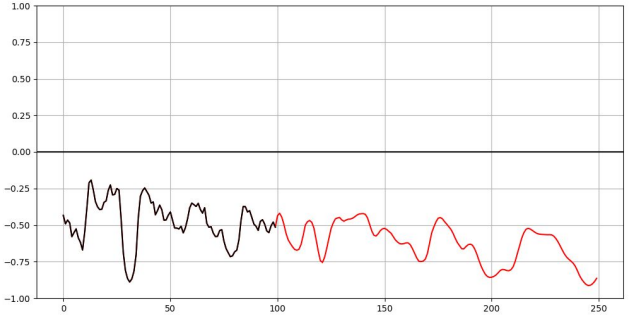
Importance of Hyperparameters

- The input window size is crucial for model performance.
- An input window of 50 steps is insufficient to learn long cycles in the data.
- An input window of 100 steps provides better context for learning.
- Learning rate is one of the most important parameters for model optimization.

Window Size: 50 Steps Epoch 50



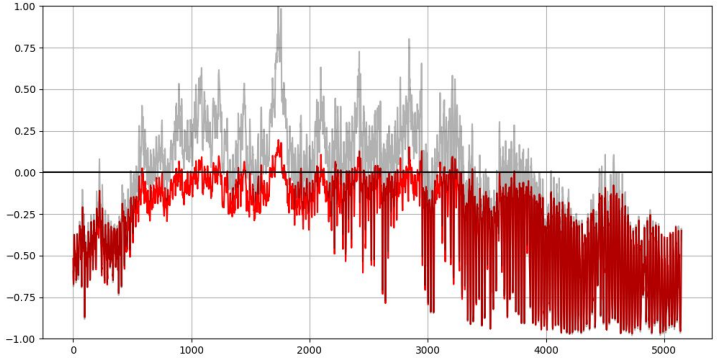
Window Size: 100 Steps, Epoch 50



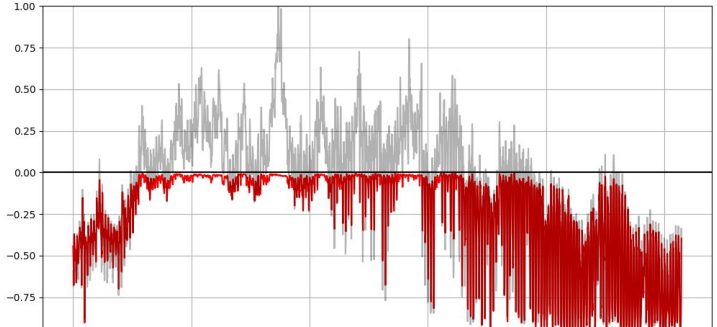
Importance of Learning Rate

- The image shows training predictions with varying learning rates.
- A low learning rate allows for detailed weight adjustments but results in slower convergence.
- A high learning rate speeds up convergence but may prevent detailed adjustments for cases outside the training norm.

Learning rate 0.0005 Epoch 20



Learning rate 0.0029 Epoch 20



Kaggle Submission Issues

- Initially attempted in Google Collab, upon finding Kaggle had own environment switched to attempting there.
- General struggles with the Kaggle interface as it is not straight forward (eg. no bulk imports?).
- Notebook necessary for submission uses a custom ‘enefit’ python module.
- Could only get module to work when forking the provided submission template.
- Initial Kaggle attempts resulted in this error.
- Retires with locally downloaded Kaggle files worked better.
- Ultimately despite finding examples (*more that listed in Kaggle notes*) we were not able to make a functioning submission..

My own Kaggle fork of the submission template with some notes:
https://www.kaggle.com/code/silverstarstorm/enefit-submission-template

‘Scoring’ and Comparison Attempts

- Kaggle leaderboard provides a ‘score’ which we could only guess may have corresponded to MAE, however the actual numbers (especially looking at lowest score) did not make sense as MAE.
- Could not find what the ‘score’ could correspond to elsewhere.
- Scoured through all leaderboard solutions and did not find any form of internal ‘score’ value we could compare to (eg. MAE or MSE).
- Went managed to track down and and went through all the solution codes (including the ones of the leaderboard we found) and did not find any means of comparison either.
- Theoretically we may have, given more time, been able to run the solutions ourselves and then compare. This however would have required: use of only openly accessible code files, only openly accessible imports (including data), would have had to have a Kaggle ranking to be worth something for us, and would have needed to be feasibly runnable on our hardware. Considering not many python notebooks were openly accessible, and of those a very few had some reference to Kaggle ranking, and if we only have one point of comparison we do not get much information; the (very poignant) time limitation would not have had the chance to be relevant.