Predicting Solar Radiation Using Weather Data

https://github.com/lmh2/data1030-machine-learning.git

Leo Major
Brown University Data Science Institute
December 9, 2023

1. Introduction

Purpose

Being able to predict solar radiation levels based on weather can inform decisions pertaining to the usage of solar panels, such as how much power output one can expect based on certain conditions. As climate change necessitates the build-out of renewable energy sources such as solar, this question becomes increasingly important.

Kaggle Dataset

I used the Solar Radiation Prediction dataset[1] from Kaggle, obtained via NASA. This dataset contains meteorological data from the HI-SEAS weather station over four months (September-December 2016).

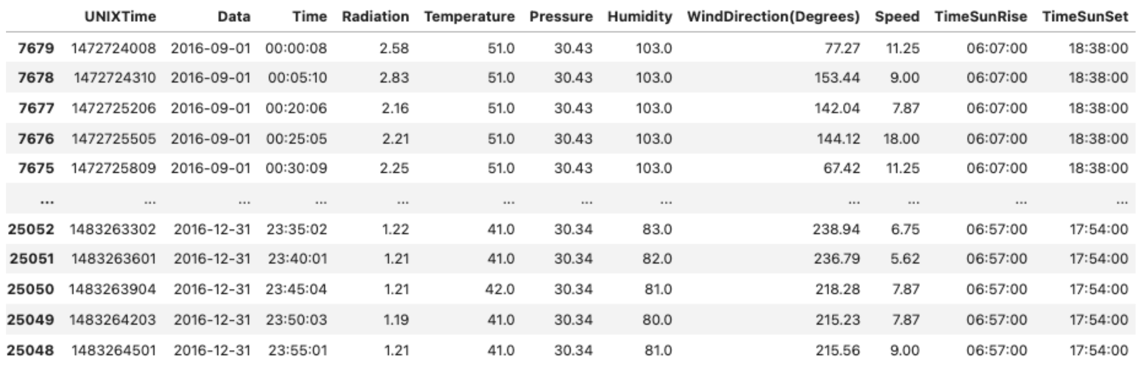
Features and Target Variable

This is a regression problem with time-series data. After dropping missing values and feature engineering, there are 33,738 rows and 11 features. The target variable that I predicted was "Radiation" (watts/meter squared), and my features included: 'Wind Direction', 'Wind speed', 'Humidity', 'Temperature', 'Pressure', 'Hour', 'Minute', 'Second', 'TotalSeconds', 'TotalMinutes', and 'Sunrise/sunset Time'. These time variables were extracted from an initial "Data" feature, which was then dropped. The sunrise/sunset time variable was created by combining two features. Over four months, I dropped four days which contained missing values. Finally, with the aim of predicting 3 hours ahead, I added three features lagged by 36, 24, and 12 points (one point = 5 mins, so the lags are approximately 3, 2, and 1 hours).

2. Exploratory Data Analysis

Feature Analysis

I first printed the unmodified dataframe head (Fig. 1). I then plotted my target variable, the mean radiation per hour of the day (Fig. 2). This shows a pattern reliant on the sun which is mirrored by other features, such as mean temperature per hour of the day (Fig. 3).



	UNIXTime	Data	Time	Radiation	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed	TimeSunRise	TimeSunSet
7679	1472724008	2016-09-01	00:00:08	2.58	51.0	30.43	103.0	77.27	11.25	06:07:00	18:38:00
7678	1472724310	2016-09-01	00:05:10	2.83	51.0	30.43	103.0	153.44	9.00	06:07:00	18:38:00
7677	1472725206	2016-09-01	00:20:06	2.16	51.0	30.43	103.0	142.04	7.87	06:07:00	18:38:00
7676	1472725505	2016-09-01	00:25:05	2.21	51.0	30.43	103.0	144.12	18.00	06:07:00	18:38:00
7675	1472725809	2016-09-01	00:30:09	2.25	51.0	30.43	103.0	67.42	11.25	06:07:00	18:38:00
...	...	...	...	...	...	...	...	...	...	...	...
25052	1483263302	2016-12-31	23:35:02	1.22	41.0	30.34	83.0	238.94	6.75	06:57:00	17:54:00
25051	1483263601	2016-12-31	23:40:01	1.21	41.0	30.34	82.0	236.79	5.62	06:57:00	17:54:00
25050	1483263904	2016-12-31	23:45:04	1.21	42.0	30.34	81.0	218.28	7.87	06:57:00	17:54:00
25049	1483264203	2016-12-31	23:50:03	1.19	41.0	30.34	80.0	215.23	7.87	06:57:00	17:54:00
25048	1483264501	2016-12-31	23:55:01	1.21	41.0	30.34	81.0	215.56	9.00	06:57:00	17:54:00

Fig 1. Unmodified Dataframe Head

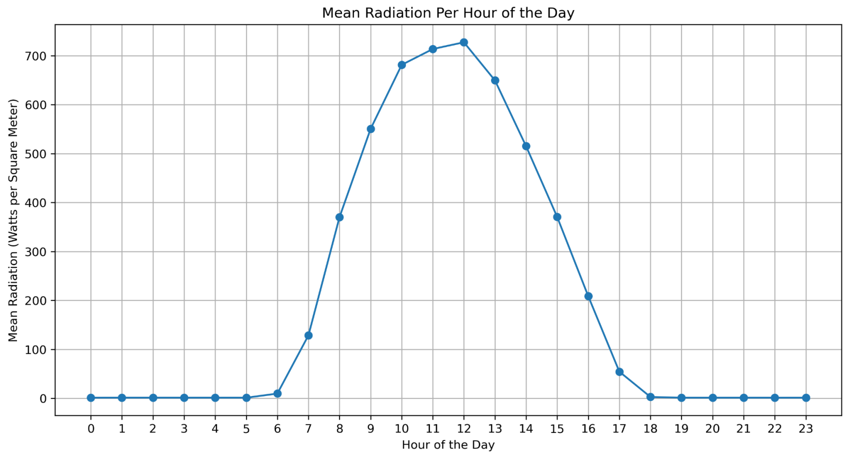


Fig 2. Mean Radiation Per Hour of the Day

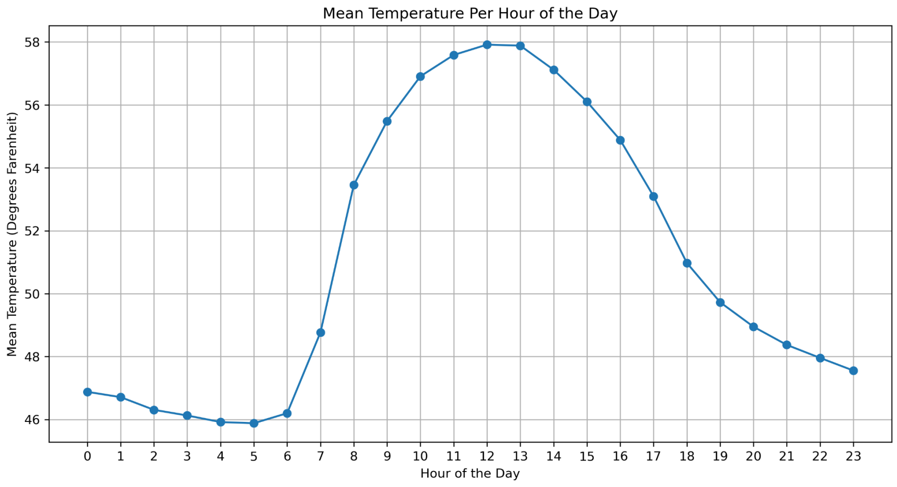


Fig 3. Mean Temperature Per Hour of the Day

When plotting mean temperature vs. mean radiation, we can see there is some correlation (Fig. 4).

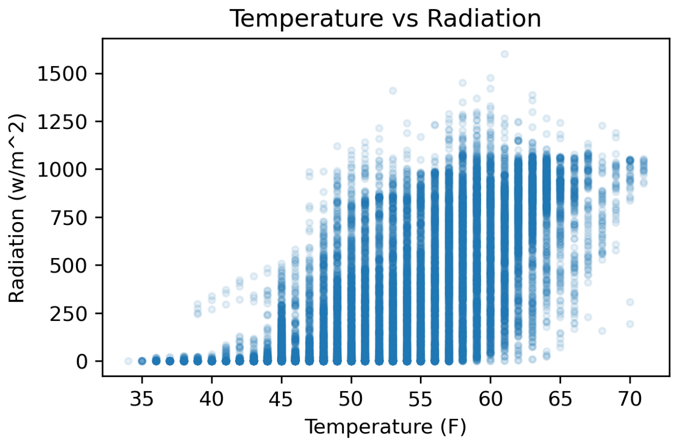


Fig 4. Mean Temperature Per Hour of the Day

Figure 5 shows the correlation matrix (where you can see that the time features overlap strongly), and figure 6 shows the autocorrelation between radiation and time.

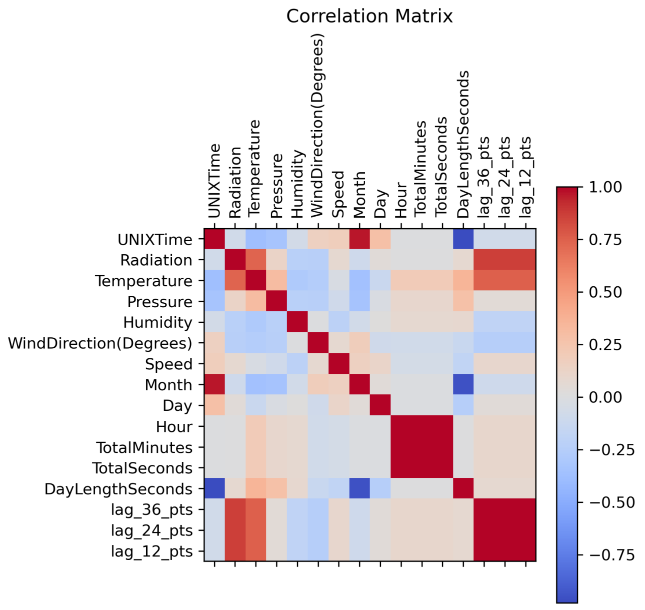


Fig. 5 Correlation Matrix

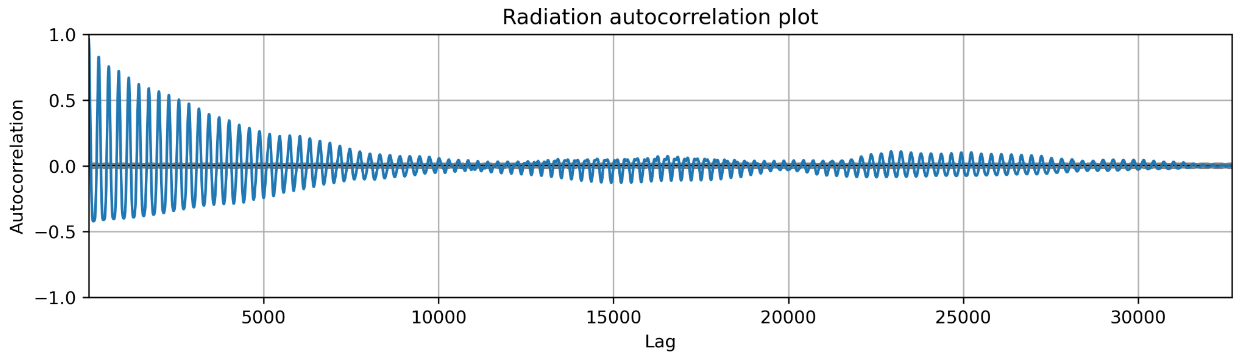


Fig. 6 Radiation (w/m^2) Autocorrelation Plot

3. Methods

Splitting

I first split my data into 'test' and 'other' using a simple train test split method. Then, I split 'other' into 'train' and 'validation' using a time series split with 5 folds, which allowed for a more robust splitting strategy. This is visualized in figure 7 below:

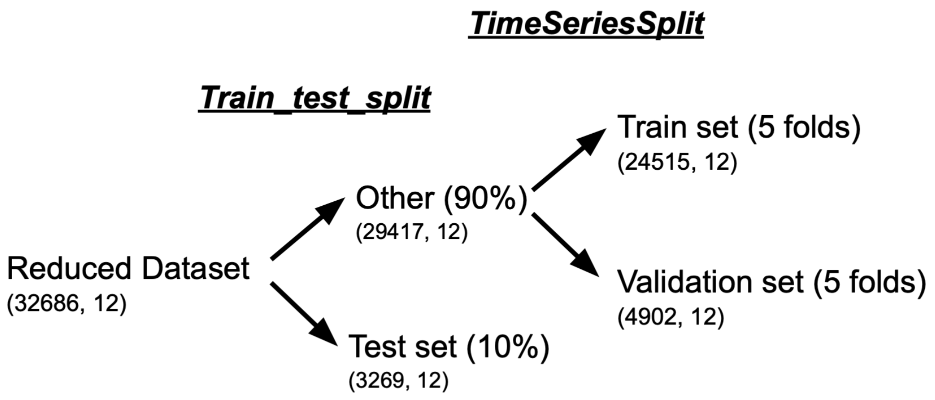


Fig. 7. Splitting Strategy

Preprocessing
All of my features were preprocessed as numerical features using StandardScaler, except for 'Month', which I treated as an ordinal feature (9, 10, 11, 12) using OrdinalEncoder.

Models
I use a function that after splitting and preprocessing runs GridSearchCV on four different models. These are: Ridge L2 Regression, Random Forest Regression, XGBoost Regression, and Support Vector Machine Regression. I run 3 random states for Random Forest Regression, and 1 random state for the rest of the models as they are deterministic when used on a time series dataset. GridSearchCV goes through the different hyperparameters (Fig. 8) and chooses the best ones. The evaluation metric I chose was Root Mean Squared Error (RMSE). It is easily interpretable because it shares the units of the target variable (watts per square meter). There were uncertainties due to Random Forest's non-deterministic behavior which led to multiple different outcomes depending on the random state, while the rest stayed the same.

Model	*Hyperparameters*
Ridge L2 Regression	'randomforestregressor__n_estimators': [25, 30, 50], 'randomforestregressor__max_depth': [20, 25, 30, 50]
Random Forest Regression	"xgbregressor__learning_rate": [0.01, 0.03], "xgbregressor__n_estimators": [1000], "xgbregressor__seed": [0], "xgbregressor__missing": [np.nan], "xgbregressor__max_depth": [3, 6], "xgbregressor__colsample_bytree": [0.8, 0.9, 1.0], "xgbregressor__subsample": [0.66, 0.8, 1.0]
XGBoost Regression	'svr__kernel': ['linear', 'rbf'], 'svr__gamma': [1e-3, 1e-1, 1e1], 'svr__C': [1e-1, 1e0, 1e1]
Support Vector Machine Regression	'ridge__alpha': [10, 100,1000], 'ridge__max_iter': [10000, 100000, 1000000]

Fig. 8. Models and Hyperparameters

4. Results

Model Scores

The RMSE results of the four models are shown below in Figure 9, as compared to the baseline RMSE (calculated using the mean). As three of the models were only run with one random state, they did not have standard deviations. Only Random Forest had a standard deviation of 1.84 over 3 random states, and this is too small to see in the figure.

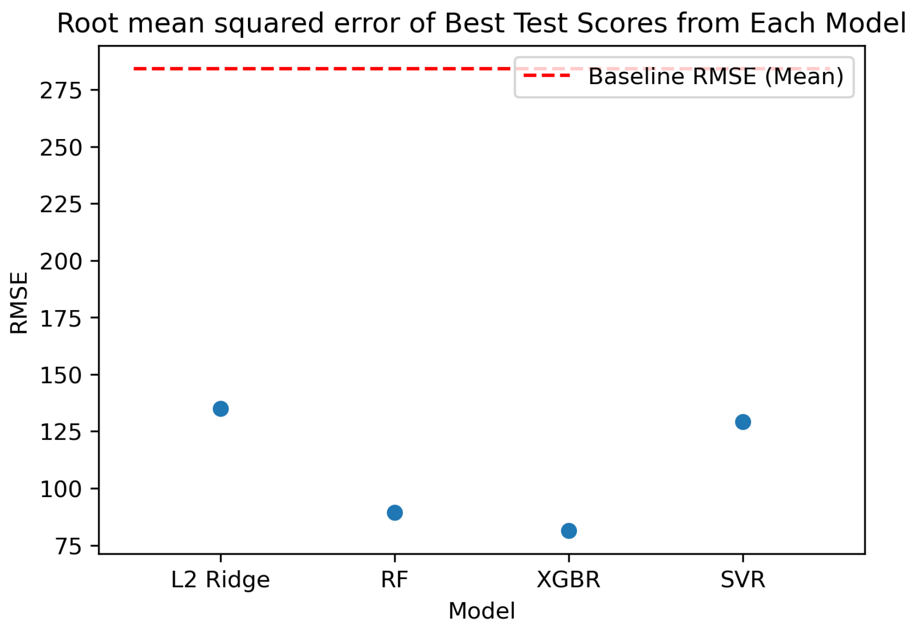


Fig. 9. RMSE for All Models, Compared to Baseline

The baseline RMSE score was 284. While all models got error scores significantly lower than the baseline, Random Forest and XGBoost performed the best (Fig. 10). XGBoost performed slightly better, making it my most predictive model.

Model	*RMSE Score*	*RMSE Standard Deviation*	*RMSE Baseline*
Ridge L2 Regression	135.12	0	284
Random Forest Regression	89.44	0	284
XGBoost Regression	81.36	1.847	284
Support Vector Machine Regression	129.27	0	284

Fig. 10. RMSE for All Models, Compared to Baseline

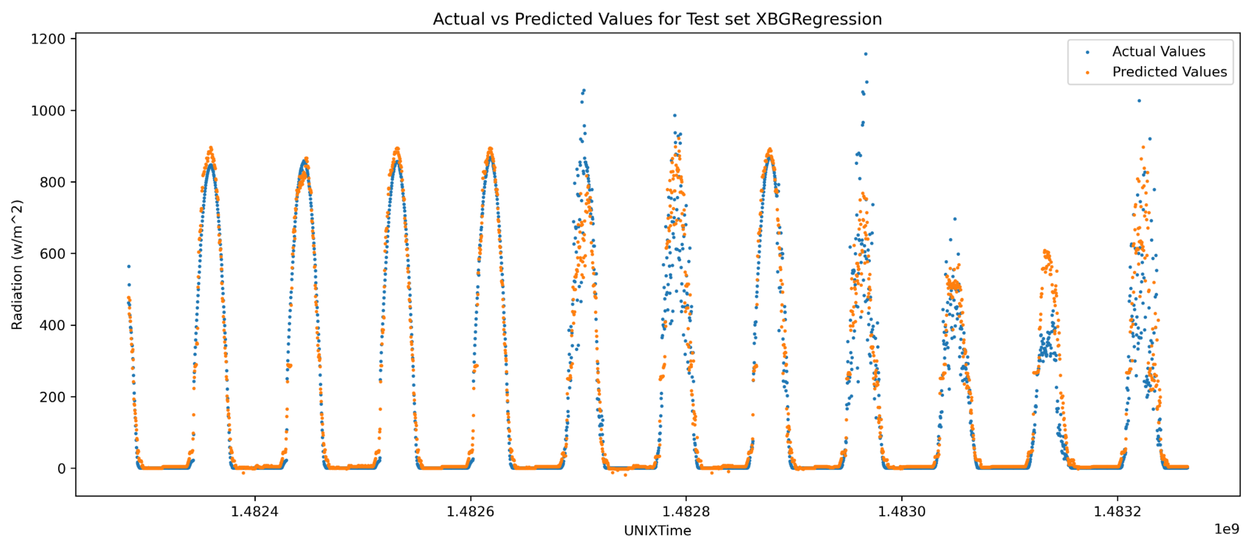


Fig. 11. XGBR Actual vs. Predicted Values

Global Feature Importances

I calculated global feature importances for my best model (XGBoost Regression) using three methods: permutation importance (Fig. 12), XGBoost's gain metric (Fig. 13) and SHAP with 200 data points (Fig. 14).

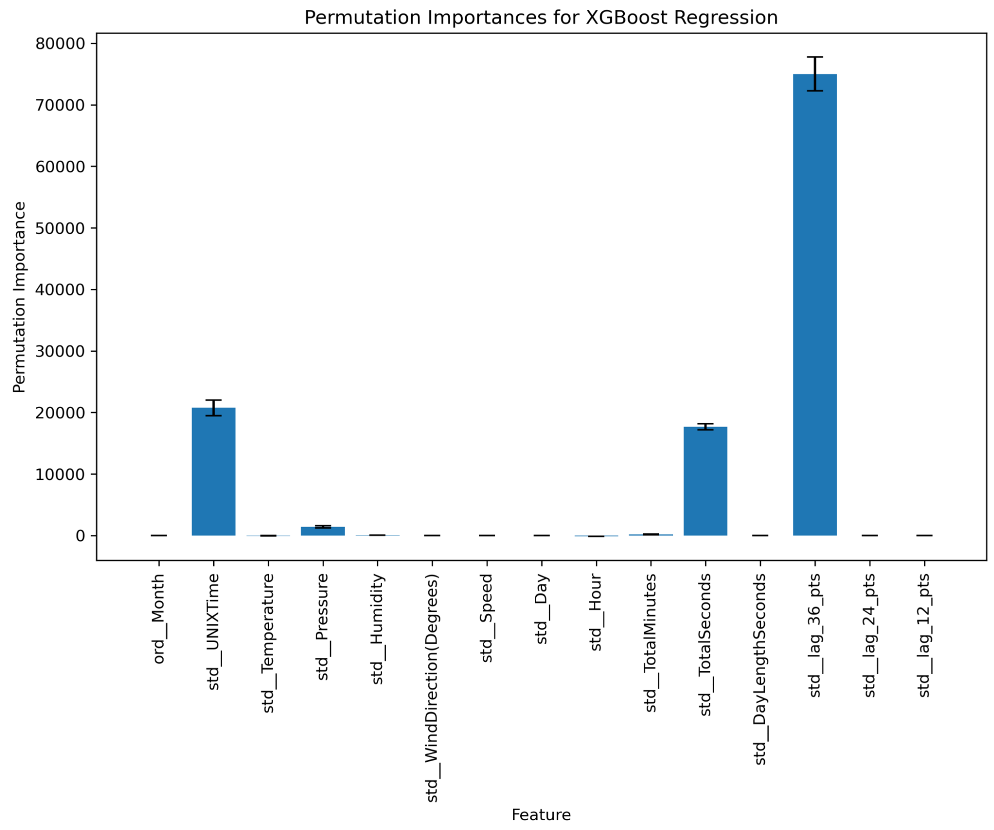


Fig. 12. XGBR Global Feature Importance- Permutation Importance

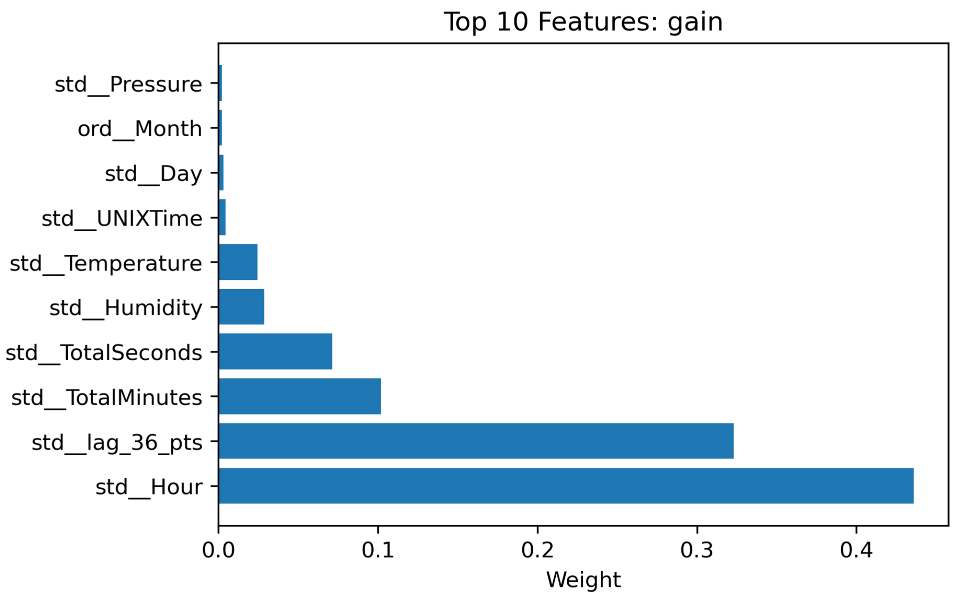


Fig. 13. XGBR Global Feature Importance- XGBoost's Gain

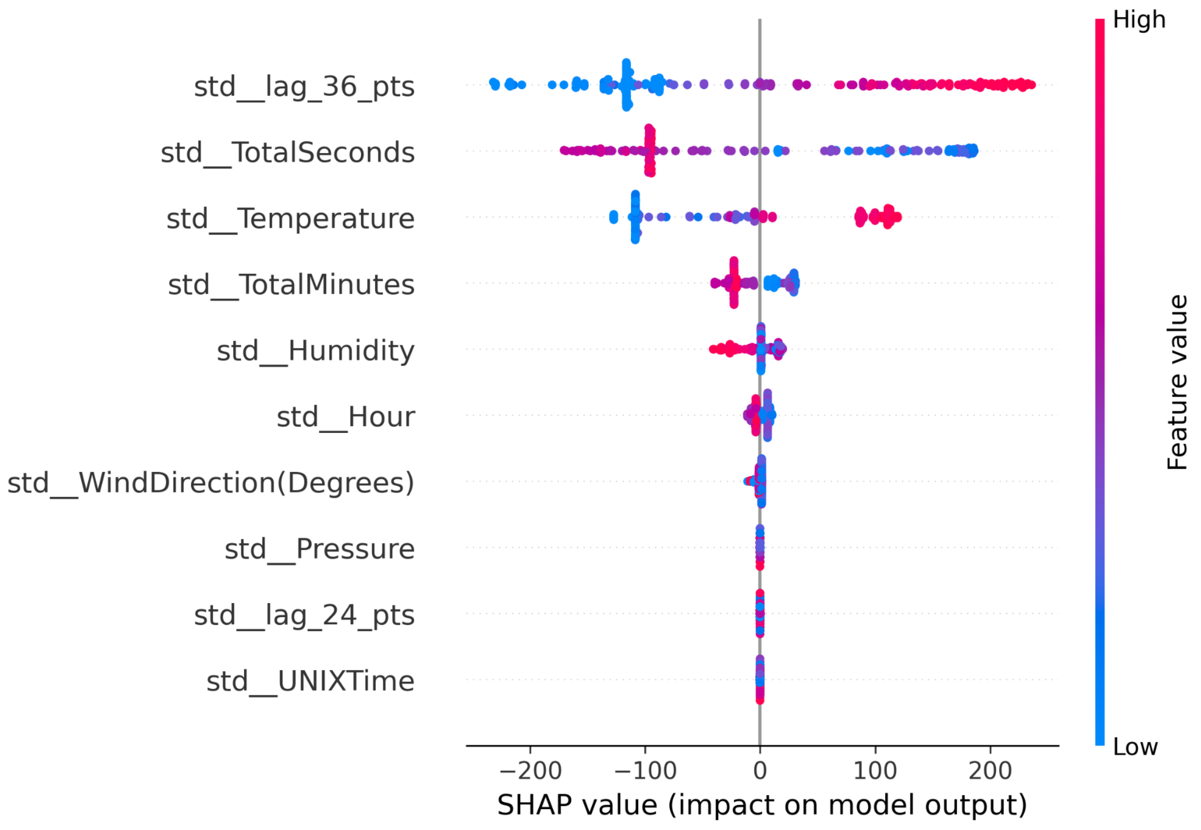


Fig. 14. XGBR Global Feature Importance- SHAP (200 samples)

Local Feature Importances

I calculated local feature importances using SHAP for two data points: 4 and 40. For data point 4, we can see that TotalSeconds influenced the prediction to be higher, while lag_36 and temperature influenced the prediction to be lower (Fig. 15). For data point 40, the number was only influenced to go higher, mainly by Total Seconds, lag_36, and Temperature (Fig. 16).

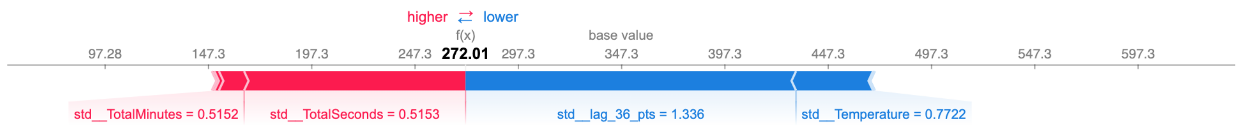


Fig. 15. SHAP Local Feature Importance- Data Point 4

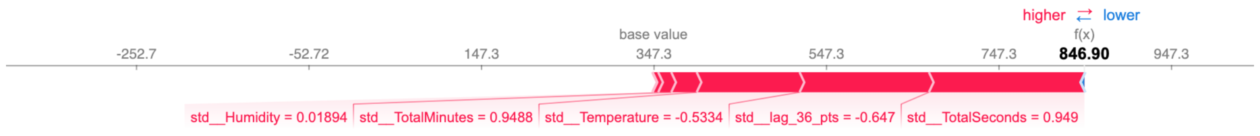


Fig. 16. SHAP Local Feature Importance- Data Point 40

Reflection

Looking at the local and global SHAP importances, we see that lag_36, Temperature, and TotalSeconds have the highest influences. However, while the permutation importance graph also shows TotalSeconds and lag_36 to be highly important, UNIXTime is also significant. XGBoost's Gain global metric also places a lot more importance on Hour than any of the other methods. The main overlap between the different methods are lag_36 and TotalSeconds. This tells us that time of day is critically important in predicting solar radiation levels. This is surprising, as one would expect that weather factors such as temperature would play a larger role.

5. Outlook

Limitations & Improvements

- *Strongly correlated features:* One limitation is the fact that many of the time features were strongly correlated. There is a possibility that this might reduce predictive power, so a future improvement could be to drop them.
- *Predicting different time scales:* The lagged features of 3, 2 and 1 hours existed for the purpose of predicting solar radiation 3 hours ahead of the current time. This gives the model a lot of information, thus increasing its predictive power. One alternative next step would be to predict solar radiation 12 hours ahead of the current time, therefore creating lagged features of 36 hours, 24 hours and 12 hours. This could be repeated for 1 day ahead of time.
- *Different Hyperparameters:* I tuned less hyperparameters than I could have because of time limitations (model taking a long time to run). In the future, I could experiment with more hyperparameters. These could be run on my top two models, XGBoost Regression and Random Forest for example.
- *Troubleshoot Feature Importances:* When comparing global feature importances for all four models, I noticed that the other three models exhibited a strange outcome. They seemed to place an exceedingly large importance on a single feature, while all the other features (even the similar

ones) showed little importance. One hypothesis for why this occurred is that the similar features canceled each other out and this could be solved by dropping them. Alternatively, there is something wrong with the way the models are running, which would necessitate further troubleshooting.

6. References

[1]: Dronio. (2017). Solar Energy Generation Data. Retrieved from Kaggle: https://www.kaggle.com/datasets/dronio/SolarEnergy/data
[2]: Github Repository: https://github.com/lmh2/data1030-machine-learning.git