FINAL REPORT

Mehmet Yiğit Turalı, Mehmet Efe Lorasdağı

Introduction

The energy crisis has made it possible for researchers to investigate new ideas and for businesspeople to invest in alternative energy sources. Because wind energy can provide power on a large scale, it is becoming more and more popular. Due to the current electricity crisis, wind energy must compete with alternative energy sources. Numerous European nations have begun switching to this cleaner alternative for their energy needs. Recently, with the Russian invasion of Ukraine, which caused a political crisis between the rest of Europe and Russia, European countries accelerated their search for an alternative to natural gas even more. Considering the possible consequences of using nuclear energy to produce electricity, such as the Chernobyl accident, wind energy can be a vital step in finding an alternative source. But several recognized problems demand attention. The fluctuating nature of wind energy has brought up the issue of the reliability of the power distribution system. Therefore, addressing wind power generation's unpredictable nature and variability represents the industry's main problem.

Problem

Research is currently being done to create models that accurately anticipate future electricity generated from wind power over the short and long term. Creating a short-term wind power forecast model is the main emphasis of this project. The prediction of short-term electricity generated from wind power ranges in time from a few minutes to a day. A balance between electricity generation and consumption must be maintained in a grid, necessitating the development of a reliable mechanism for forecasting wind power on a short-term basis.

Dataset

The data consists of time-related internal features which can be generalized under Real-time SCADA (Supervisory Control and Data Acquisition) data. SCADA system acts as a nerve center for the wind turbines. It connects the individual turbines, substations, and meteorological stations to a central computer. Hence the data is time-series data which has information about internal features of wind turbine which are given with a specific “timestamp”. Moreover, the dataset has features such as temperature of gearboxes, bearings, batterybox and transformers, tower acceleration normal and lateral information, torque, blade angles, proximity sensor degrees and similar internal features about wind turbines. Although the dataset has various internal features, there are only three external features in the dataset which are “Ambient Temperature” and “Wind Deviation in 1 second and 10 seconds”. In the below list, the one can find all features that belongs to the dataset:

[Timestamp, Gearbox_T1_High_Speed_Shaft_Temperature, Gearbox_T3_High_Speed_Shaft_Temperature, Gearbox_T1_Intermediate_Speed_Shaft_Temperature, Temperature Gearbox Bearing Hollow Shaft, Tower Acceleration Normal, Gearbox_Oil-2_Temperature, Tower Acceleration Lateral, Temperature Bearing_A, Temperature Trafo-3, Gearbox_T3_Intermediate_Speed_Shaft_Temperature, Gearbox_Oil-1_Temperature, Gearbox_Oil_Temperature, Torque, Converter Control Unit Reactive Power, Temperature Trafo-2, Reactive Power, Temperature Shaft Bearing-1, Gearbox_Distributor_Temperature, Moment D Filtered, Moment D Direction, N-set 1,Operating State, Power Factor, Temperature Shaft Bearing-2, Temperature_Nacelle, Voltage A-N, Temperature Axis Box-3, Voltage C-N, Temperature Axis Box-2, Temperature Axis Box-1, Voltage B-N, Nacelle Position_Degree, Converter Control Unit Voltage, Temperature Battery Box-3, Temperature Battery Box-2, Temperature Battery Box-1, Hydraulic Prepressure, Angle Rotor Position, Temperature Tower Base, Pitch Offset-2 Asymmetric Load Controller, Pitch Offset Tower Feedback, Line Frequency ,Internal Power Limit, Circuit Breaker cut-ins, Particle Counter, Tower Accelaration Normal Raw, Torque Offset Tower Feedback, External Power Limit, Blade-2 Actual Value_Angle-B, Blade-1 Actual Value_Angle-B, Blade-3 Actual Value_Angle-B, Temperature Heat Exchanger Converter Control Unit, Tower Accelaration Lateral Raw, Temperature Ambient, Nacelle Revolution, Pitch Offset-1 Asymmetric Load Controller, Tower Deflection, Pitch Offset-3 Asymmetric Load Controller, Wind Deviation 1 seconds, Wind Deviation 10 seconds, Proxy Sensor_Degree-135, State and Fault, Proxy Sensor_Degree-225, Blade-3 Actual Value_Angle-A, Scope CH 4, Blade-2 Actual Value_Angle-A, Blade-1 Actual Value_Angle-A, Blade-2 Set

Value_Degree, Pitch Demand Baseline_Degree, Blade-1 Set Value_Degree, Blade-3 Set Value_Degree, Moment Q Direction, Moment Q Filltered, Proxy Sensor_Degree-45, Turbine State, Proxy Sensor_Degree-315, Power]

The dataset link, description and definition can be found on Appendix A.

The input variables conveyed in the data set for the time period to be forecasted are ready to be utilized to predict the power generation results in the same period, and each data value solely pertains to the pertinent period.

For preprocessing, firstly the data is analyzed and Explanatory Data Analysis (EDA) is performed. After that feature engineering was performed. For EDA, firstly, the dataset opened as dataframe and analyzed the type of data:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 136730 entries, 0 to 136729
Columns: 107 entries, Timestamp to rolling_24_power_mean
dtypes: datetime64[ns](1), float64(95), int64(9), object(2)
memory usage: 112.7+ MB
```

Figure 1: Data Type, Memory Usage Analysis

From the Figure 1, it can be seen that we have 107 column with different datatypes such as datetime64,float64. This dataset can be computationally handled easily since it is memory usage is not very large.

Next, to see date interval and number of days in dataset, the data analyzed via Pandas:

```
The data is available from 2019-01-01 00:00:00 to 2021-08-14 23:50:00

The number of days of wind power in the dataset is 949
```

Figure 2: Date Interval and the Number of Days in the Dataset

Since data issampled 6 samples per hour, which means that the sampling rate is 10 minutes, we have $949 \times 144 = 136656 + 74(remaining\ time) = 136730$ data points in data set.

Accordingly, the statistical analysis of feature columns is made. The number of feature columns, the mean, standard deviation, minimum and maximum values of the feature columns are analyzed:

	Gearbox_T1_High_Speed_Shaft_Temperature	...	rolling_24_power_mean
count	136730.000000	...	136730.000000
mean	896.733880	...	1138.392954
std	9011.509039	...	1026.318033
min	-273.000000	...	-45.100694
25%	44.020000	...	142.950969
50%	57.049166	...	859.129385
75%	63.570000	...	2176.361853
max	99999.000000	...	2773.057210

[8 rows x 104 columns]

Figure 3: Statistical Analysis of Feature Columns

To display there are any missing values or not we use isna().sum() from Pandas:

```
[1 rows x 107 columns]
Timestamp                                          0
Gearbox_T1_High_Speed_Shaft_Temperature            0
Gearbox_T3_High_Speed_Shaft_Temperature            0
Gearbox_T1_Intermediate_Speed_Shaft_Temperature    0
Temperature Gearbox Bearing Hollow Shaft           0
                                                  ..
rolling_4_power_mean                               0
rolling_6_power_mean                               0
rolling_12_power_mean                              0
rolling_18_power_mean                              0
rolling_24_power_mean                              0
Length: 107, dtype: int64
```

Figure 4: The Number of Missing Values for Every Feature Column in Dataset

From above, the one can see that there are not any missing values in the dataset.

For EDA, firstly the plots of power generation trend across time, yearly average power production, weekly power generation trend across years are plotted and outlier detection is made.

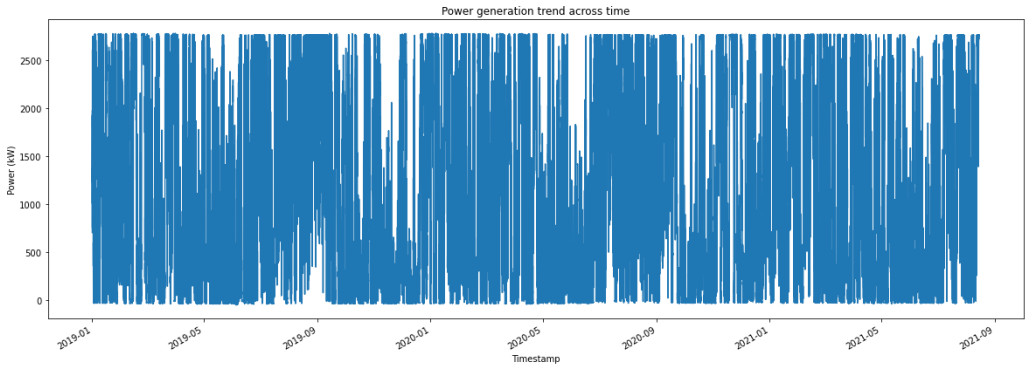


Figure 5: Power Generation Trend Across Time

As it can be seen on the graph, the power production across time is very unstable, therefore the forecasting task is going to be hard to implement.

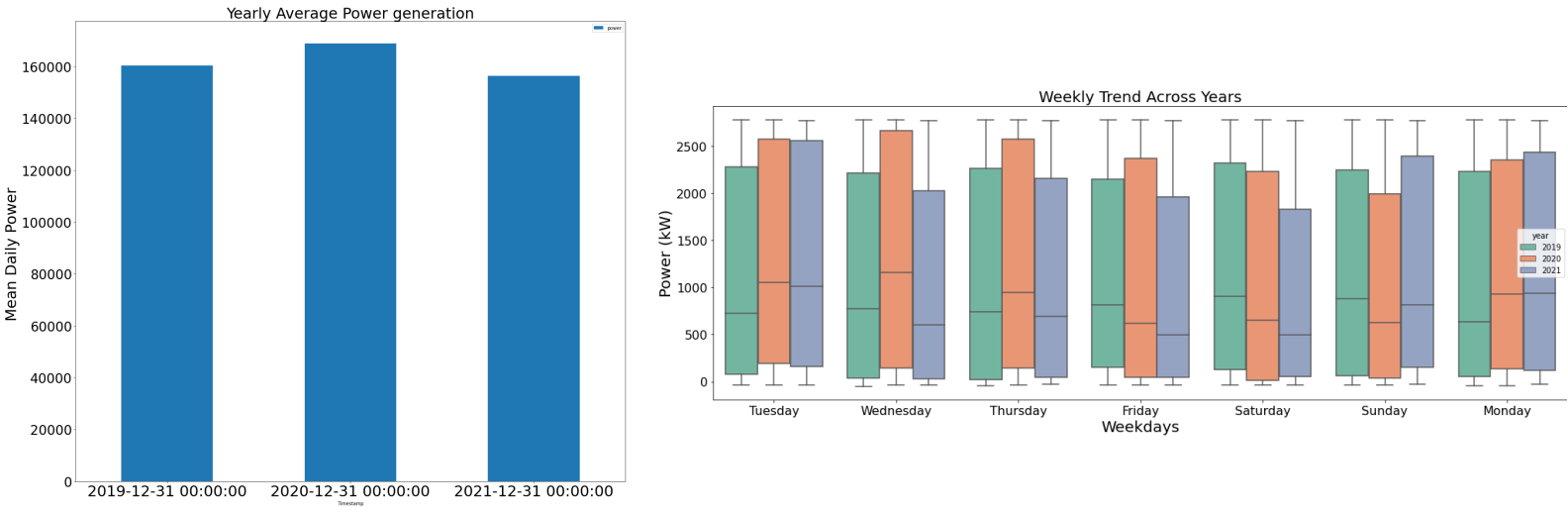


Figure 6: Yearly and Weekly Average Power Production

From the bar graph, 2020 is the most productive year in the wind power plant which is followed by 2019 and 2021.

The weekly trend nearly follows the yearly average, therefore, the dataset is in seasonal trend on average.

For outlier detection, we used the outlier detection formula:

1. Sort the data from low to high
2. Identify the first quartile (Q1) and the third quartile (Q3).
3. IQR = Q3 – Q1
4. Upper Bound = Q3 + (1.5 * IQR)
5. Lower Bound= Q1 – (1.5 * IQR):

```
The upper bound is 5655.516842842102 , lower bound is  -3264.6794385910034
We have following number of outliers : 0
```

Figure 7: Outlier Detection

We will use different machine learning models in the next part of the report which are Lasso Regression, MLP and Decision Tree-based Algorithms. Hence, before machine learning can be used, time series forecasting problems must be re-framed as supervised learning problems. Therefore for feature engineering part, we added weekday, month, year and hour data to dataset, since datetime data cannot be used as a feature. Also “timeblock” number is added as float and “morning”, ”afternoon”, ”evening”, “night”, “midnight to dawn” data are added as binary features since average wind power production is affected from the which part of the day are the data in.

Moreover, time series forecasting issues are traditionally converted into supervised learning issues using lag features. Also, the values in the sliding window can be used to compute summary statistics, which we can then use as features in our dataset. The rolling mean, often known as the mean of the last several values, is possibly the most helpful. For this purpose we added 1, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72 lagging features and 4, 6, 12, 18, 24 rolling mean windows. In the below table, one can see the engineered additional features and their descriptions:

Features	Definitions
MonthName	Month Names which are encoded as Sin/Cos Function
tb	Time Blocks
Morning	Morning or Not (binary Encoded)
Afternoon	Afternoon or Not (binary Encoded)
Evening	Evening or Not (binary Encoded)
Night	Night or Not (binary Encoded)
MidnighttoDawn	Midnight to Dawn or Not (binary Encoded)
feature_lagging_power_1,6,12,18,24,30,36,42,48,54,60,66,72	Power is Lagged with Given Timeblocks
Rolling_4,6,12,18,24_power_mean	Rolling Mean of Power with given Timeblocks
Ramp	The Change in Power Between Current and Given Time Blocks Ahead

Table 1: Engineered Additional Features

After that we use partial autocorrelation and autocorrelation and correlation matrix to decide which lagging and rolling features are we going to use.

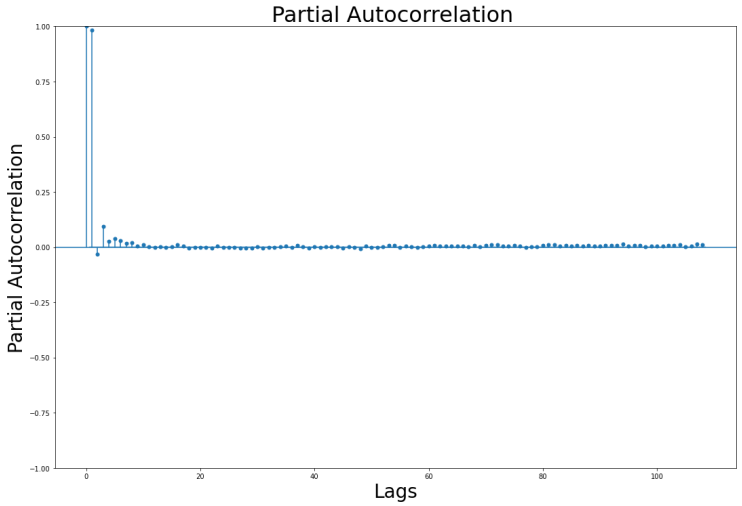


Figure 8: Partial Autocorrelation For Lagging Features

As it can be seen from the graph, lagging power feature 1 and 6 have strong correlation with power generation.

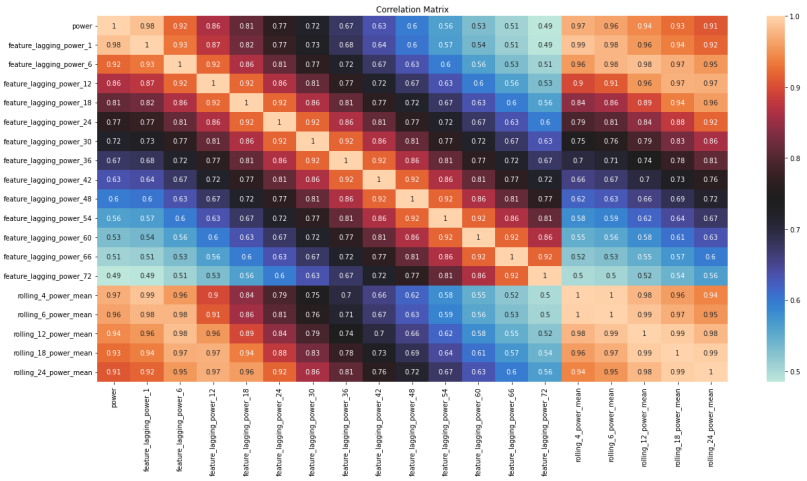


Figure 9: Correlation Matrix for Lagging and Rolling Power Features

From matrix feature_lagging_power_1 6, and 12 also rolling_4_power_mean 6, 12, 18 and 24 have strong correlations with power generation data.

For different target time block forecasting, we will use different lagging and rolling features in order to predict without giving the same target time block lagging and rolling features. Finally, to prevent data leakage, we partitioned the last 8600 samples for the test set before any training. The remaining data was custom partitioned according to the demand of the below models.

Machine Learning Methods

1) LASSO Regression

The first model we will use is L1 Regression, Least Absolute Shrinkage and Selection Operator (LASSO) regression. When there are many features, LASSO is frequently employed since it does regularization and variable selection. The goal is to decrease over-fitting and increase forecast accuracy. The LASSO regression approach is appropriate for models with high degrees of multicollinearity, just like Ridge regression. The goal of LASSO regression, which employs L1 regularization, is to obtain the subset of predictors that reduce the prediction error rate for a target attribute [1].

To start our algorithm, we started with data normalization and regularization. Regularization is a crucial idea that helps prevent overfitting of the data, especially when the learned and test sets of data differ significantly. Regularization also limits the effect of predictor variables over the output variable by compressing their coefficients.

First, the dataset is standardized by using below formula:

$$x_{standard} = \frac{x-\mu}{\sigma}$$

for all features.

Our dataset has over 80 features, in order to prevent overtraining, LASSO regression is a suitable choice. The reason, the LASSO method was chosen is that as we have a lot of features, we assumed pushing more irrelevant features would be a benefit as it would prevent overfitting.

Since Lasso Regression cannot use gradient descent directly, we used the Coordinate Descent algorithm. To update model weights, firstly the coordinate descent for L1 regression will be explained. We take the gradient of cost function with respect to weights of the model:

$$\frac{dJ}{dw} = \frac{-2}{m}\left[\sum_{i=1}^{m} x_j\left(y^{(i)} - \hat{y}^{(i)}\right) + \lambda\right] if\ w_i > 0$$

$$\frac{dJ}{dw} = \frac{-2}{m}\left[\sum_{i=1}^{m} x_j\left(y^{(i)} - \hat{y}^{(i)}\right) - \lambda\right] if\ w_i \leq 0$$

for weights and

$$\frac{dJ}{db} = \frac{-2}{m}\left[\sum_{i=1}^{m}\left(y^{(i)} - \hat{y}^{(i)}\right)\right]$$

for bias term.

To update weights and bias following equation is used:

$$w_{l+1} = w_l - \alpha \times \Delta w_l$$

$$b_{l+1} = b_l - \alpha \times \Delta b_l$$

where α is the learning rate. With the above equations the parameters will be updated.

Next, to find the best λ and α parameter, GridSearchCV(Cross Validation) is used. Since we use time series as a dataset we cannot shuffle the dataset, therefore we used Time Series Split CV to solve this problem. Cross-validation on a rolling basis is a technique that can be applied to cross-validate the time-series model. The algorithm starts with a small subset of data for training purposes, makes predictions for subsequent data points, and then assesses the precision of the predictions. The following training dataset contains the same predicted data points, and subsequent data points are forecasted after that.

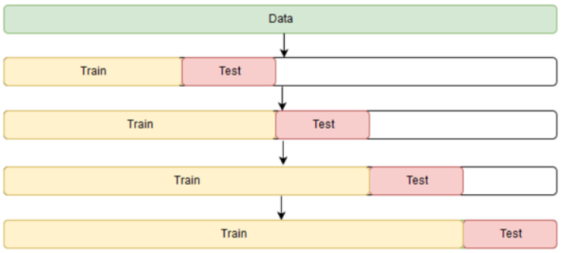


Figure 10: Time Series Cross Validation [4]

On the assumption that the validation set is always in front of the training set, the idea behind time series splits is to divide the training set into two folds at each iteration.

Moreover, GridSearchCV is a method for adjusting hyperparameters to find the best values for a particular model. Technically, only the parameters that we put in your parameter grid could produce the best parameters that GridSearchCV identifies.

By integrating the Time Series Split CV and GridSearch algorithm, we have tuned the hyperparameters for the lasso regression. With 4 folds for CV the training has been done. Firstly to find best λ and α parameter list are made for these parameters. The model is trained for λ = 0.05, 0.1,0.15, 0.2, 1 and $\alpha = 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$ for 500 epochs which will be forecast 24 hours.

The hyperparameter table is for Lasso Regression is

Hyperparameters	Value-1	Value-2	Value-3	Value-4
λ	0.05	0.1	0.15	0.2
α	10^{-4}	10^{-3}	10^{-2}	10^{-1}

Table 2: Hyperparameter Table For Lasso GridSearchCV

For forecast 24 hours best learning rate is 10^{-2} in 500 epochs and best λ is 1 since at 10^{-1} the model overfits and the MSE is smallest at this step :

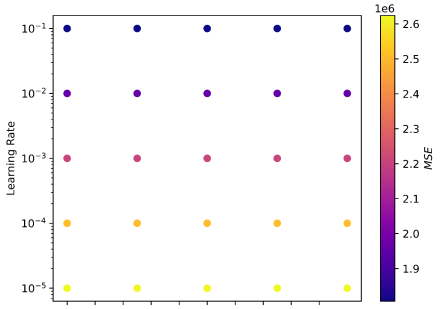


Figure 12: GridSearchCV Results for 24 Hours Forecasting

Since MSE is too large for forecasting, 4, 6, 8, 12 hours ahead are also tested. Firstly GridsearchCV is applied to all models with λ = 0.001, 0.01, 0.1, 0.5, 1, 2, 10 and $\alpha = 10^{-2}$ for 500 epochs (See Appendix A,B,C).

Time blocks will be tested	4, 6, 8, 12 hours	4, 6, 8, 12 hours	4, 6, 8, 12 hours	4, 6, 8, 12 hours	4, 6, 8, 12 hours	4, 6, 8, 12 hours	4, 6, 8, 12 hours
λ	0.001	0.01	0.1	0.5	1	2	10

Table 3: Hyperparameter Table For Lasso Second GridSearchCV

After hyperparameter tuning, Forecasting 4 Hours Ahead Model is trained and tested.

```
R^2 = 0.936525908365205
RMSE = 264.2629385720416
```

Figure 11: R-Square and RMSE Results of Testing 4 Hours Forecast Ahead Model

R-squared is a statistical indicator of how well a regression model fits the data. R-square metric's maximum value is 1. The model is better suited if the r-square value is close to 1. Since the R-Square result is nearly equal to 1 the model fits the data successfully, and forecasts 4 hours ahead successfully. Moreover, since RMSE is not large, the predictions are consistent enough to forecast 4 hours ahead (For Loss Graph and Forecast Results see Appendix E,F).

2) <u>Decision Tree Regressor and Random Forest Algorithm</u>

Since decision trees and Random Forest are machine learning models that can be used for regression, we chose it as our third model in our project. However, since Decision Trees and Random Forest tend to overfit over data, we chose top 15 features from 80 features from the dataset. To select features, we apply "Feature Importance" by using our Random Forest algorithm. The bar graph below displays that the sorted feature importances for our algorithm to give maximize accurate forecasting on validation set:

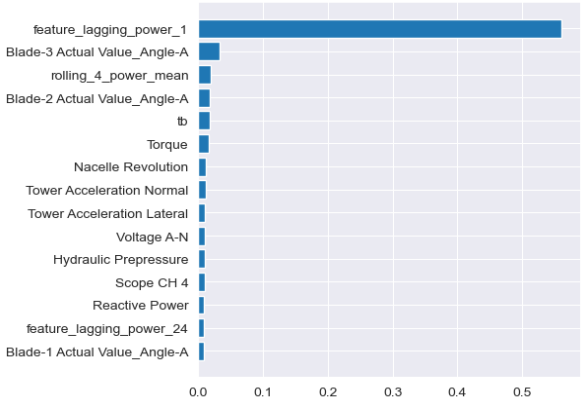


Figure 12: Feature Importance for Decision Tree Based Algorithms

After feature importances obtained, top 15 and intuitively chosen 2 features which are "Power" and "Ramp" features are chosen to train our decision tree first. Then Training, Validation and test set partitioned where, test set contains 8600 data samples. For training and validation sets, in our first experiment we tried to do 5-fold time series cross validation which contains of [22554 *(fold_number+1)] training samples and 22554 validation samples in each fold. However, since the training and validation folds contain large amounts of samples, the Decision Tree became extremely slow and cannot complete the GridSearchCV in 10 consecutive days.Therefore, we make 10-fold which contains 4000 training samples and 2000 validation samples in each fold for Cross Validation. In other words, we used Blocking 10-folds Cross Validation as in figure below:

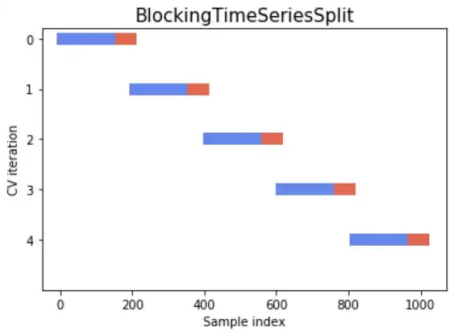


Figure 13: Blocking Cross Validation Schematic [4]

It operates by introducing margins in two places. In order to prevent the model from noticing lag values that are utilized twice-once as a regressor and again as a response-the first is between the

training and validation folds. In order to avoid the model from memorizing patterns from one iteration to the next, there are two folds employed at each iteration. With using Minimum Number of Split and Max Depth hyperparameters 10 Fold Blocking GridSearchCV algorithm is applied:

Hyperparameters	Value-1	Value-2	Value-3	Value-4	Value-5
min_samples_split	3	4	5	7	-
max_depth	3	6	9	12	15

Table 4: Hyperparameter Table For Decision Trees GridSearchCV

After 12 hours of GridSearchCV with 20 combinations, the Best hyperparameters and the hyperparameter plane are:

```
Minimum MSE: 1200.3164
Best max_depth: 3
Best min_samples_split: 5
```

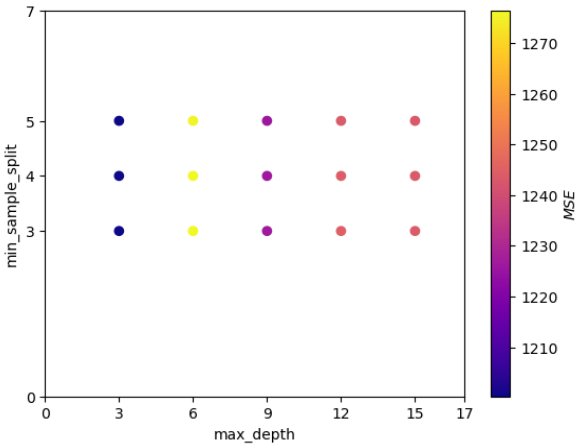


Figure 14: 10-Fold Blocking GridSearchCV results. The one does not pay regard to MSE loss since the dataset sizes are too small in order to run the hyperparameter analysis.

By using above hyperparameters whose combination is the most successful in validation set, the decision tree is trained and tested for forecasting 4 hours ahead:

```
R^2 = 0.6836
RMSE = 385.7239
MAPE = 18.7782
```

Figure 15: Decision Tree Algorithm with max_depth: 3 min_samples_split: 5 Test Results

After obtaining the test results, one can see that since it requires extremely amount of time and computational power we cannot train with whole dataset; therefore, the forecast is not very accurate, but still reasonable. However, from R-squared test, one can see that our model can predict the trend with nearly 18.78% MAPE since the mean of the power generation is nearly 2500-2550 kW per day.

To improve our predictor, we implemented a Random Forest Regressor onto the Decision Tree algorithm since the Random Forest is an ensemble algorithm that is a forecasting tool superior to single Decision Tree algorithm. Firstly we tried to apply 10-Fold Blocking GridSearchCV algorithm however, training with 10 folds with different tree numbers takes too much computational power, therefore cross-validation is failed. Accordingly, we restricted the number of trees to 2 since we do not have enough computational power to train unoptimised algorithm, max features to 10. By using best hyperparameters of Decision Tree hyperparameters, (max_depth:3, min_samples_split: 5**),** we tried to train model again, but the model the 16 GB RAM was not computationally strong enough . Therefore, the algorithm was mainly useful for obtaining feature importance only, rather than forecasting.

3) Multi-Layer Perceptron

The second model we will employ is Neural networks. Neural networks can be utilized in a variety of tasks, including regression. We will use a fully connected network (Multi-Layer Perceptron). As we have used feature engineering to obtain features that include information from past states, using MLPs is an adequate approach [2]. Fully Connected networks are powerful models that can represent and learn many patterns. For this reason, we chose FCNs as a method. We decided to utilize an architecture of 1 input, 2 hidden, and 1 output layer. Rectified linear unit (ReLU) was used as the activation function.

There is no analytic derivation for this approach, but intuitively we deemed this model is complex enough to learn and not too complex to overfit. The hyperparameters to be tuned are layer sizes of hidden layer 1 and hidden layer 2.

Hyperparameters	Value-1	Value-2	Value-3
Hidden Layer 1 Size	12	10	8
Hidden Layer 2 Size	6	4	2

Table 5: Hyperparameter Table For MLP GridSearchCV

Furthermore, 2 experiments were conducted. Firstly, the 17 most important features determined in Decision Trees section were used in training the model. Then, the model was trained on all of the features. Then, for these 2 different feature sizes, the above hyperparameters were tried.

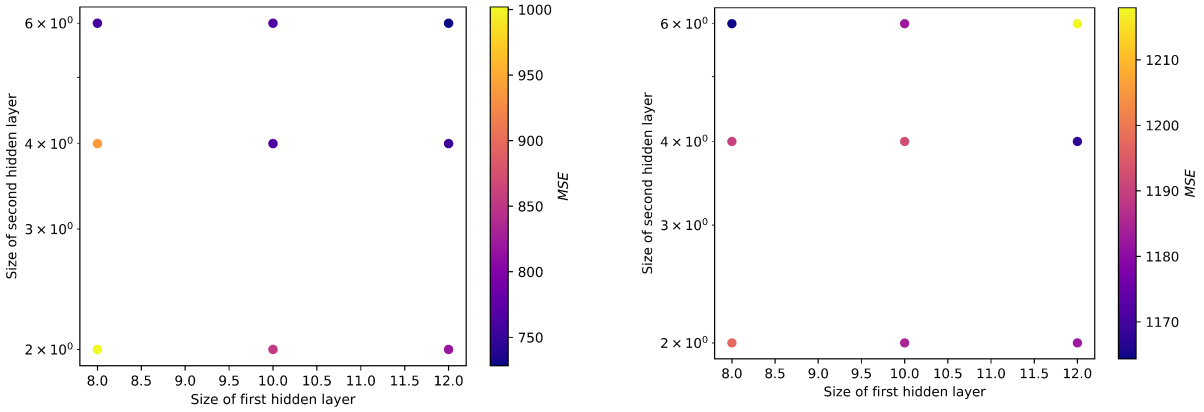


Figure 16,17: Validation MSE results of the model with feature selection and Validation MSE results of the model with all features respectively.

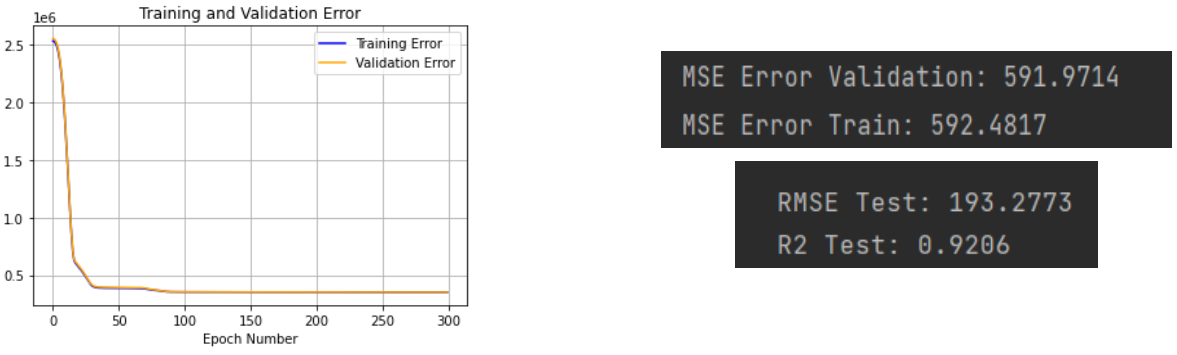


Figure 18: Training and Validation Loss and Training,Validation and Test Results

Forecasting results of the trained MLP model can be seen in Appendix G.

Conclusion

In this project, our aim was to predict wind energy from some data that includes SCADA data and corresponding wind energy generation. The goal was to predict a real number, so this was a regression problem. For this goal, we have worked with 3 different machine learning models: LASSO regression, decision tree-based algorithms, and neural networks. Further, we used extensive feature engineering and selection in the project. Our data, wind energy prediction, is a time series, but our machine learning models do not inherently have previous context information. We used feature engineering to obtain and use extra features that give information about previous data, namely lag features and rolling window mean.

For the first method, LASSO regression, we used all features of the dataset. The LASSO algorithm pushes irrelevant features to 0, thus we did not need to perform feature selection. The LASSO model's R-squared metric on the test set was 0.936 and RMSE was 264.3 kW. According to our results, the one can understand that from R-squared score LASSO model can track the trend easily but with a little error since the RMSE score is 264.3 kW (Average Power Production is 2500 kW per day), in other words calculated MAPE is 12.3% which is very accurate prediction. Thus this model is suitable for forecasting Wind Power Production.

The second method, Decision Based Algorithms were most problematic algorithms for us since writing them by using only NumPy and Pandas makes them unoptimized for our current computers. The algorithm requires extremely amount of time and computational power since we cannot make batch training and our dataset has nearly 110.000 data samples. However, with a small training dataset and with nearly 36 hours training and cross validation time, we obtain nearly good forecaster for our problem. With a 0.6836 R-Squared score, the one can understand that the model can nearly track the trend deviation of the test set. Moreover, with 385.72 kW RMSE and 18.77% MAPE score, the model can be classified as a intermediate forecaster for this problem. Moreover, we tried to train the Random Forest algorithm which we implemented for only obtaining feature importances, but it failed since we cannot do batch training and it requires extremely amount of time and computational power

For the third method, fully connected neural networks, we experimented with 2 possible feature sets: 17 selected features from the second method or using all features. The validation and training errors were smaller for selected features, so we proceeded with it. GridSearchCV was used to tune the hyperparameters which were hidden layer sizes for this method. 12 and 6 were found as the best ones for first and second hidden layers, respectively. Then, after the features and layer sizes were selected, the network was trained for 300 epochs, which was selected by using the validation costs of epochs. The multi-layer perceptron model's R-Squared metric on the test set was 0.921 and RMSE was 193.3 kW.

R-Squared metric explains how well the trends in the data are captured, and RMSE explains the difference between predicted values and target values. R-Squared of LASSO and MLP is similar, however, RMSE of MLP is significantly lower. Thus, it can be concurred that under our experiment settings, the trained MLP model is the best model for predicting wind energy from the given dataset and our features.

Gantt Chart

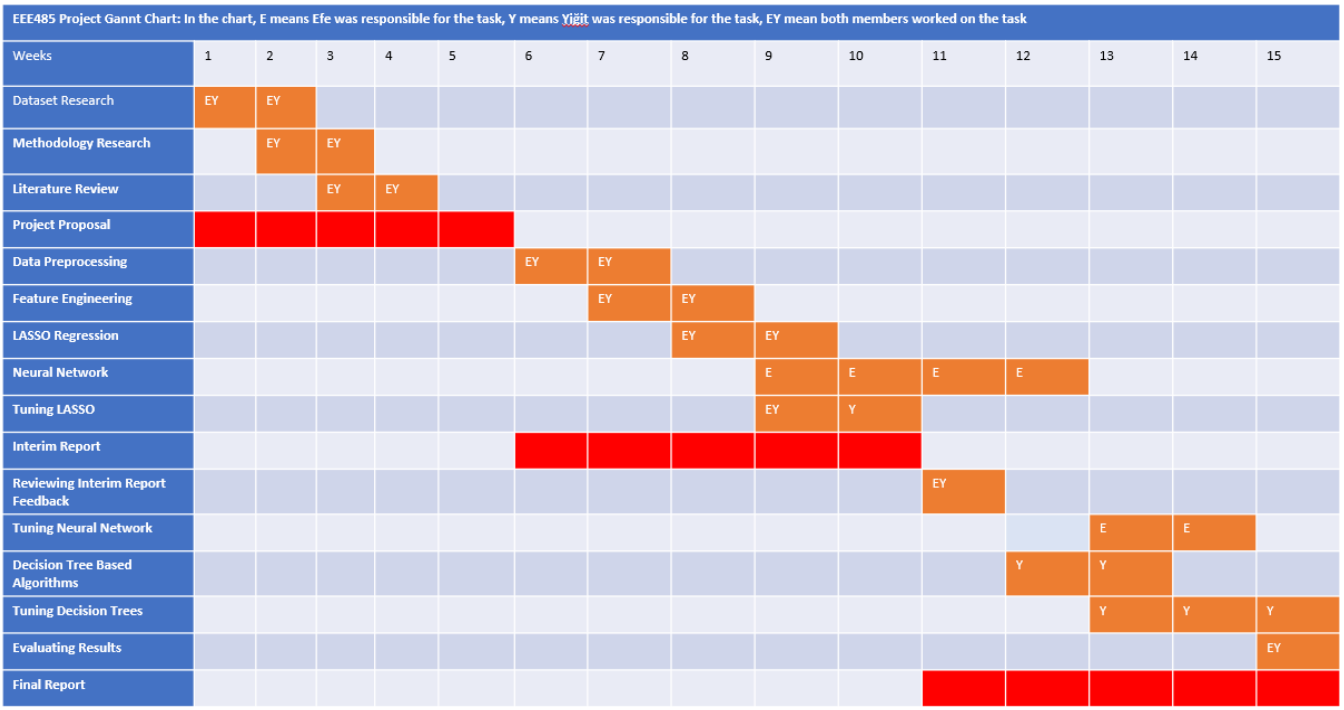


Figure: Gantt Chart of the Project.

It should be noted that E means Efe was responsible for the task, Y means Yiğit was responsible for the task, EY means both members worked on the task

Encountered and Solved Challenges:

As our original data is a time series data, we used some feature engineering to maintain a modified dataset that includes time dependent features in itself. This approach caused some unexpected problems since the importance of feature engineered features have exceeded 50% of the other data.

For both Neural Networks and Decision Tree Based Algorithms , using 100 features and large training dataset would be computationally expensive and prone to overfitting. Thus, feature selection was very important, especially for the decision tree, for the training of these 2 models. We solved this problem by using feature importance with Random Forest algorithm implemented by us.

References

[1] Wind power generation prediction using machine learning algorithms. Retrieved October 18, 2022, from https://dergipark.org.tr/tr/download/article-file/1052210

[2] J. Brownlee, “Crash course on multi-layer perceptron neural networks,” *MachineLearningMastery.com*, 02-Aug-2022. [Online]. Available: https://machinelearningmastery.com/neural-networks-crash-course/. [Accessed: 20-Nov-2022].

[3] “Machine learning basics: Decision tree regression.” [Online]. Available: https://towardsdatascience.com/machine-learning-basics-decision-tree-regression-1d73ea003fda. [Accessed: 20-Nov-2022].

[4]S. Shrivastava, “Cross validation in Time Series,” *Medium*, 17-Jan-2020. [Online]. Available: https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4. [Accessed: 19-Dec-2022].

Appendix

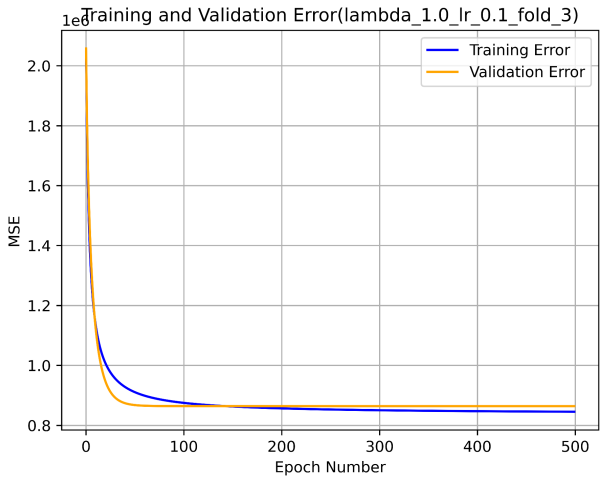
A:Dataset Definitions:

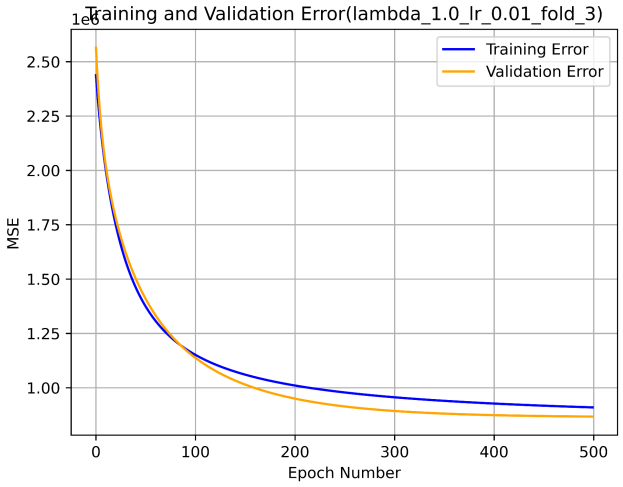
Dataset link:
https://www.kaggle.com/datasets/psycon/wind-turbine-energy-kw-generation-data?select=features.csv

Features	Definitions
Timestamp	Date-Time Information of Sample
Gearbox_T1_High_Speed_Shaft_Temperature	The Temperature of Gearbox T1 at High Speed Shaft
Gearbox_T3_High_Speed_Shaft_Temperature	The Temperature of Gearbox T3 at High Speed Shaft
Gearbox_T1_Intermediate_Speed_Shaft_Temperature	The Temperature of Gearbox T1 at Medium Speed Shaft
Temperature Gearbox Bearing Hollow Shaft	The Temperature of Gearbox Bearing Hollow Shaft
Tower Acceleration Normal	Tower Acceleration in the Normal Axis of Tower(to calculate oscillation)
Gearbox_Oil-2_Temperature	The Temperature of Gearbox 2 Oil
Tower Acceleration Lateral	Tower Acceleration in the Lateral Axis of Tower(to calculate oscillation)
Temperature Bearing_A	The Temperature of Bearing A
Temperature Trafo-3	The Temperature of the transformer
Gearbox_T3_Intermediate_Speed_Shaft_Temperature	The Temperature of Gearbox T3 at Medium Speed Shaft
Gearbox_Oil-1_Temperature	The Temperature of Gearbox 1 Oil
Gearbox_Oil_Temperature	The Temperature of Gearbox 0 Oil
Torque	Torque of the Turbines
Converter Control Unit Reactive Power	Converter Control Unit Reactive Power
Temperature Trafo-2	Temperature Trafo-2
Reactive Power	Reactive Power
Temperature Shaft Bearing-1	The Temperature of Bearing 1
Gearbox_Distributor_Temperature	The Temperature of Gearbox Temperature
Moment D Filtered	Filtered D Moment Value
Moment D Direction	Moment D Direction
N-set 1	N-set 1 (rpm)
Operating State	Operating State
Power Factor	Power Factor
Temperature Shaft Bearing-2	The Temperature of Shaft Bearing
Temperature_Nacelle	The Temperature of Nacelle (kind of cover of wind turbine)
Voltage A-N	Voltage A-N
Temperature Axis Box-3	The Temperature of Axis Box-3
Voltage C-N	Voltage C-N
Temperature Axis Box-2	The Temperature of Temperature Axis Box-2
Temperature Axis Box-1	The Temperature of Temperature Axis Box-1
Voltage B-N	Voltage B-N
Nacelle Position_Degree	Nacelle Position_Degree
Converter Control Unit Voltage	Converter Control Unit Voltage
Temperature Battery Box-3	The Temperature of Battery Box-3
Temperature Battery Box-2	The Temperature of Battery Box-2
Temperature Battery Box-1	The Temperature of Battery Box-1
Hydraulic Prepressure	Hydraulic Prepressure
Angle Rotor Position	Angle Rotor Position
Temperature Tower Base	The Temperature of Tower Base
Pitch Offset-2 Asymmetric Load Controller	Pitch Offset-2 Asymmetric Load Controller
Pitch Offset Tower Feedback	Pitch Offset Tower Feedback

Features	Definitions
Line Frequency	Line Frequency
Internal Power Limit	Internal Power Limit
Circuit Breaker cut-ins	Circuit Breaker cut-ins
Particle Counter	Counts Unwanted Particles in Gearbox Unit
Tower Accelaration Normal Raw	Tower Acceleration in the Normal Axis of Tower in raw form(to calculate oscillation)
Torque Offset Tower Feedback	Torque Offset of Tower Feedback
External Power Limit	External Power Limit
Blade-2 Actual Value_Angle-B	Actual Angle-B of Blade 2
Blade-1 Actual Value_Angle-B	Actual Angle-B of Blade 1
Blade-3 Actual Value_Angle-B	Actual Angle-B of Blade 3
Temperature Heat Exchanger Converter Control Unit	Temperature of Heat Exchanger Converter Control Unit
Tower Accelaration Lateral Raw	Tower Acceleration in the Lateral Axis of Tower in raw form(to calculate oscillation)
Temperature Ambient	Ambient Temperature
Nacelle Revolution	The Revolution of Gearbox Cover
Pitch Offset-1 Asymmetric Load Controller	Pitch Offset-1 Asymmetric Load Controller
Tower Deflection	Tower Deflection
Pitch Offset-3 Asymmetric Load Controller	Pitch Offset-3 Asymmetric Load Controller
Wind Deviation 1 seconds	Wind Deviation in 1 seconds
Wind Deviation 10 seconds	Wind Deviation in 10 seconds
Proxy Sensor_Degree-135	Proxy Sensor Degree 135 (mm)
State and Fault	State and Fault
Proxy Sensor_Degree-225	Proxy Sensor Degree 225 (mm)
Blade-3 Actual Value_Angle-A	Actual Angle-A of Blade 3
Scope CH 4	Scope CH 4
Blade-2 Actual Value_Angle-A	Actual Angle-A of Blade 1
Blade-1 Actual Value_Angle-A	Actual Angle-A of Blade 2
Blade-2 Set Value_Degree	Default Angle of Blade 3
Pitch Demand Baseline_Degree	Degree of Pitch Demand Baseline
Blade-1 Set Value_Degree	Default Angle of Blade 1
Blade-3 Set Value_Degree	Default Angle of Blade 2
Moment Q Direction	Filtered Q Moment Direction
Moment Q Filtered	Filtered Q Moment Value
Proxy Sensor_Degree-45	Proxy Sensor Degree 45 (mm)
Turbine State	State of Turbine (Fault or No-Fault)
Proxy Sensor_Degree-315	Proxy Sensor Degree 315 (mm)
Power	Power Produced at that Moment

B: At LR = 0.1, model overfits





C: GridSearchCV Results for Different Forecast Models

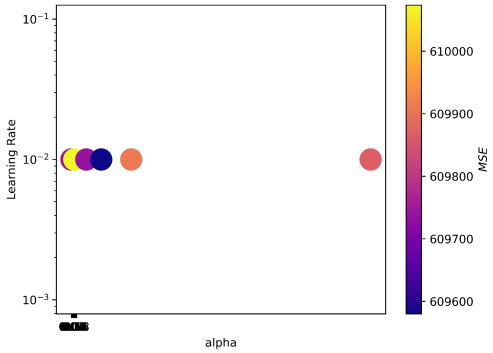
Model	λ Parameter	MSE
4-hrs Ahead Forecasting	0.01	486468.0198
6-hrs Ahead Forecasting	1.0	609580.1

8-hrs Ahead Forecasting	10.0	723177.7558
12-hrs Ahead Forecasting	0.5	887243.6362

D: 4 Hours Ahead GridSearchCV

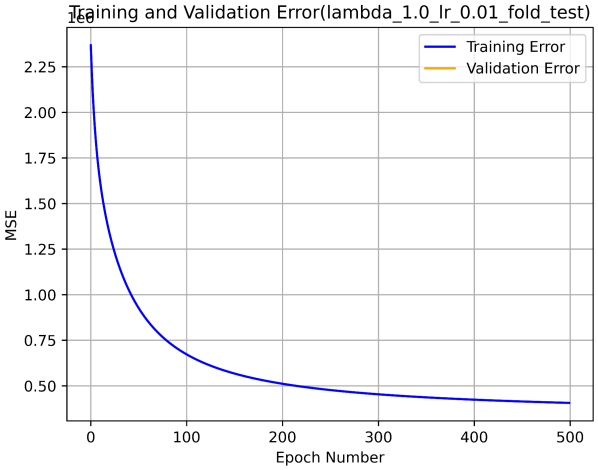
```
---------fold finished---------
[0.001, 0.01, 0.1, 0.5, 1, 2, 10]
[[353252.569  352300.0147 351773.972  339802.8106]]
Minimum MSE: 486468.0198
Best alpha: 0.01
Best gamma: 0.01
```

E: 6 Hours Ahead GridSearchCV (Purple = 1.0, counted from beginning)

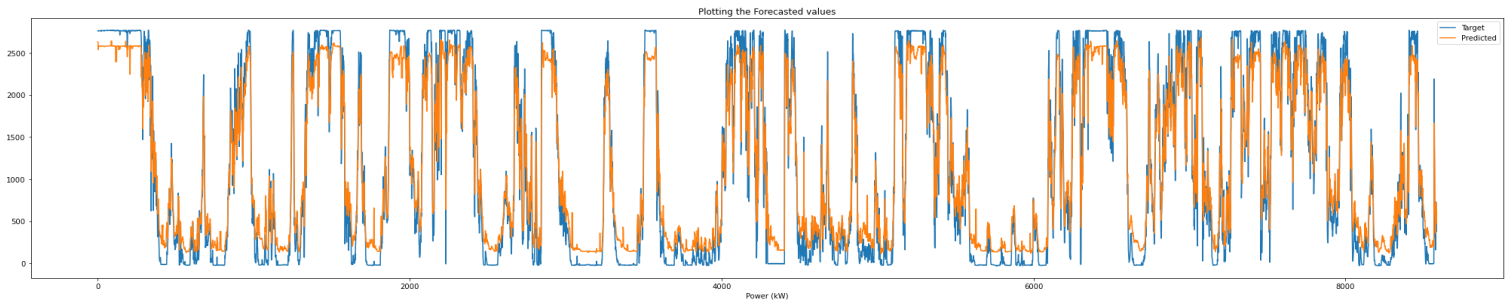



```
---------fold finished---------
[0.001, 0.01, 0.1, 0.5, 1, 2, 10]
[[453556.6968 450740.8363 453742.1446 440505.2525]]
Minimum MSE: 609580.0134
Best alpha: 1.0
Best gamma: 0.01
griffin:~/yigit_mefe/6hrs>
```

F: Training Error For 4-Hours Ahead Forecasting Model



G: Test Results of 4-Hours Ahead Forecasting Model with Neural Network Model



CODES:

Project_Script_DataProcessing.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 15 13:47:59 2022

@author: Mehmet Yigit Turali, Mehmet Efe Lorasdagi
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

wind_power_data = pd.read_csv('C:/Users/Mehmet Yigit Turali/Desktop/BILKENT/BILKENT FALL 2022-2023/EEE485/Project/archive (1)/power.csv')
wind_power_variables = pd.read_csv('C:/Users/Mehmet Yigit Turali/Desktop/BILKENT/BILKENT FALL 2022-2023/EEE485/Project/archive (1)/features.csv')
wind_power_all = pd.merge(wind_power_variables ,wind_power_data, how='inner', on='Timestamp')

#fill the null columns with mean of the column:
wind_power_all.fillna(wind_power_all.mean(),inplace=True)

#add year-month-day and time to dataframe
wind_power_all['Timestamp'] = pd.to_datetime(wind_power_all['Timestamp'])
wind_power_all['weekday'] = wind_power_all['Timestamp'].dt.day_name()
wind_power_all['month'] = wind_power_all['Timestamp'].dt.month_name()
wind_power_all['year'] = wind_power_all['Timestamp'].dt.year.astype('string')
wind_power_all['hour'] = wind_power_all['Timestamp'].dt.strftime('%H').astype('string')
#Detecting the outlier power
sb.boxplot(data=wind_power_all['Power(kW)'])
plt.title('Box Plot of Power(kW)')
plt.ylabel('Power(kW)')
plt.show()
#no outliers detected

#feature enginnering
feat_eng_data = wind_power_all.copy()
feat_eng_data['MonthName'] = feat_eng_data['Timestamp'].dt.month
tb = np.arange(1,feat_eng_data.shape[0]+1)
feat_eng_data['tb'] = tb
feat_eng_data['Morning'] = ((feat_eng_data['Timestamp'].dt.hour>6) &
(feat_eng_data['Timestamp'].dt.hour<=12) ).astype(int)
feat_eng_data['Afternoon'] = ((feat_eng_data['Timestamp'].dt.hour>12) &
(feat_eng_data['Timestamp'].dt.hour<=18) ).astype(int)
feat_eng_data['Evening'] = ((feat_eng_data['Timestamp'].dt.hour>18) &
(feat_eng_data['Timestamp'].dt.hour<=21) ).astype(int)
feat_eng_data['Night'] = (feat_eng_data['Timestamp'].dt.hour>21).astype(int)
```

```
feat_eng_data['MidnighttoDawn'] = ((feat_eng_data['Timestamp'].dt.hour<=6) &
(feat_eng_data['Timestamp'].dt.hour>=0) ).astype(int)

for x in [1,6,12,18,24,30,36,42,48,54,60,66,72]:
    feat_eng_data['feature_lagging_power_{}'.format(x)] = feat_eng_data['Power(kW)'].shift(x).bfill()

# build some rolling features like rolling mean
feat_eng_data['rolling_4_power_mean'] = feat_eng_data['Power(kW)'].rolling(4,
1).mean().shift().bfill()
feat_eng_data['rolling_6_power_mean'] = feat_eng_data['Power(kW)'].rolling(6,
1).mean().shift().bfill()
feat_eng_data['rolling_12_power_mean'] = feat_eng_data['Power(kW)'].rolling(12,
1).mean().shift().bfill()
feat_eng_data['rolling_18_power_mean'] = feat_eng_data['Power(kW)'].rolling(18,
1).mean().shift().bfill()
feat_eng_data['rolling_24_power_mean'] = feat_eng_data['Power(kW)'].rolling(24,
1).mean().shift().bfill()

pd.DataFrame.to_csv(feat_eng_data,'feat_eng_data.csv', sep=',', encoding='utf-8')
pd.DataFrame.to_csv(wind_power_all,'wind_power_all.csv', sep=',', encoding='utf-8')
```

EDA_script.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 20 15:35:20 2022

@author: Mehmet Yigit Turali
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import math

feat_eng_data = pd.read_csv('C:/Users/Mehmet Yigit Turali/Desktop/BILKENT/BILKENT FALL
2022-2023/EEE485/Project/feat_eng_data.csv',index_col=0)
feat_eng_data.rename(columns={'Power(kW)':'power'},inplace=True)
feat_eng_data['Timestamp'] = pd.to_datetime(feat_eng_data['Timestamp'])
feat_eng_data.info()
print("")
print(f'The data is starts from {feat_eng_data["Timestamp"].min()} ends at
{feat_eng_data["Timestamp"].max()}')
days = feat_eng_data.shape[0]//144
print("")
print(f'The number of days is {days}')
print("")
print(feat_eng_data.describe())
print("")
```

```
print(f' The maximum power production is on
\n{feat_eng_data.loc[feat_eng_data.power==feat_eng_data.power.max()]}')
print(feat_eng_data.isna().sum())

g_data = (feat_eng_data.set_index('Timestamp')).resample('H')['power'].sum()
g_data.plot(kind='line')
plt.ylabel('Power')
plt.title('Sampled Power Production to 1 HRS')
plt.show()

fig , axes = plt.subplots(1,1,figsize=(20,7))
feat_eng_data.set_index('Timestamp')['power'].plot()
plt.ylabel('Power (kW)')
plt.title('Power Production Trend Across Time')
plt.show()

year_data =
pd.DataFrame(feat_eng_data.set_index('Timestamp').resample('1D')['power'].sum()).resample('1Y').m
ean()
year_data.plot.bar()
plt.title("Yearly Average Power Generation",fontsize=32)
plt.xticks(fontsize=32,rotation = "horizontal")
plt.yticks(fontsize=28,rotation = "horizontal")
plt.ylabel("Mean Daily Power ",fontsize=32)
plt.show()

fig,axes = plt.subplots(1,1,figsize=(20,7))
sb.boxplot(data = feat_eng_data,x='weekday',y='power',hue='year',palette='Set2',ax=axes)
plt.xticks(fontsize=16,rotation = "horizontal")
plt.yticks(fontsize=16,rotation = "horizontal")
plt.xlabel("Weekdays",fontsize=20)
plt.ylabel("Power (kW)",fontsize=20)
plt.title('Weekly Trend Across Years',fontsize=20)
plt.show()

sb.boxplot(data=feat_eng_data['power'])
plt.title('Box Plot of Power(kW)',fontsize=32)
plt.ylabel('Power(kW)',fontsize=28)
plt.xticks(fontsize=16,rotation = "horizontal")
plt.yticks([0,1000,3000,5000,7000],fontsize=16,rotation = "horizontal")
plt.show()

Q1= feat_eng_data.power.quantile(0.25)
Q3= feat_eng_data.power.quantile(0.75)
IQR = Q3-Q1
lower_b = Q1-IQR*1.5
feat_eng_data['IQR_OUTLIER_FLAG'] = (feat_eng_data.power>(Q3+IQR*1.5)) |
(feat_eng_data.power<(lower_b))
```

```
print(f"The upper bound is {Q3+IQR*1.5} , lower bound is  {lower_b} ")
print(f"We have following number of outliers : {feat_eng_data['IQR_OUTLIER_FLAG'].sum()}")
```

training_test_processing.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 15 18:37:39 2022

@author: Mehmet Yigit Turali
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import math


def standard_dataset(dataset):
    for i in dataset.columns:
        try:
            dataset[i] = (dataset[i] -dataset[i].mean(skipna = True)) / dataset[i].std(skipna = True)
        except:
            print("No Standartization for {}".format(i))
feat_eng_data = pd.read_csv('C:/Users/Mehmet Yigit Turali/Desktop/BILKENT/BILKENT FALL
2022-2023/EEE485/Project/feat_eng_data.csv',index_col=0)
train_data = pd.DataFrame()
target_ahead_timeblocks = 12*4 # forecasting 24 hours ahead.
feat_eng_data.rename(columns={'Power(kW)':'power'},inplace=True)
feat_eng_data['ramp'] = feat_eng_data['power'] -
feat_eng_data['power'].shift(target_ahead_timeblocks)
features_data = feat_eng_data.copy()
#from correlation matrix no need for below features:

features_data.drop(['feature_lagging_power_18', 'feature_lagging_power_24',
'feature_lagging_power_30', 'feature_lagging_power_36',
'feature_lagging_power_42','feature_lagging_power_48',
'feature_lagging_power_54','feature_lagging_power_60',
'feature_lagging_power_66','feature_lagging_power_72','rolling_12_power_mean',
'rolling_18_power_mean', 'rolling_24_power_mean'], axis = 1, inplace=True)
#feat_eng_data = pd.get_dummies(feat_eng_data,drop_first=True)
train_data = features_data.copy()
for feature in train_data.columns:
    train_data[feature] = train_data[feature].shift(-target_ahead_timeblocks)

train_data['target'] = train_data['power'].shift(-target_ahead_timeblocks)

validation_start_date = '2020-08-14 23:50:00'
test_start_date = '2021-06-14 23:50:00'
```

```
x = train_data.drop('target',1).copy()
ground_truth = train_data[['Timestamp','target']].copy()
ground_truth = ground_truth.rename_axis('index').reset_index()


x_before = x
#standartized dataset
standard_dataset(x)
x = x.rename_axis('index').reset_index()

x_train_list = []
x_val_list = []
y_train_list = []
y_val_list = []
for i in range (22154,110770,22154):
    #idx = int(i/22154)-1
    x_train_list.append(x[x['index'] < i].iloc[:,1:].copy())
    x_val_list.append (x[(x['index']>= i) & (x['index']< i+22154)].iloc[:,1:].copy())
    y_train_list.append ( ground_truth[ground_truth['index'] < i].iloc[:,1:].copy())
    y_val_list.append ( ground_truth[(ground_truth['index']>= i) & (ground_truth['index']<
i+22154)].iloc[:,1:].copy())
for i in range (0,4):
    pd.DataFrame.to_csv(x_train_list[i],'x_train_f{}.csv'.format(i), sep=',', encoding='utf-8')
    pd.DataFrame.to_csv(y_train_list[i],'y_train_f{}.csv'.format(i), sep=',', encoding='utf-8')
    pd.DataFrame.to_csv(x_val_list[i],'x_val_f{}.csv'.format(i), sep=',', encoding='utf-8')
    pd.DataFrame.to_csv(y_val_list[i],'y_val_f{}.csv'.format(i), sep=',', encoding='utf-8')

x_train = x[x['Timestamp'] < test_start_date].iloc[:,1:].copy()
y_train = ground_truth[ground_truth['Timestamp'] < test_start_date].iloc[:,1:].copy()
# x_val = x[(x['Timestamp']>= validation_start_date) & (x['Timestamp']<
test_start_date)].iloc[:,1:].copy()
# y_val = ground_truth[(ground_truth['Timestamp']>= validation_start_date) &
(ground_truth['Timestamp']< test_start_date)].iloc[:,1:].copy()
x_test = x[(x['Timestamp']>= test_start_date)] .iloc[:,1:].copy()
y_test = ground_truth[(ground_truth['Timestamp']>= test_start_date)] .iloc[:,1:].copy()

pd.DataFrame.to_csv(x_train,'x_train.csv', sep=',', encoding='utf-8')
pd.DataFrame.to_csv(y_train,'y_train.csv', sep=',', encoding='utf-8')
pd.DataFrame.to_csv(x_test,'x_test.csv', sep=',', encoding='utf-8')
pd.DataFrame.to_csv(y_test,'y_test.csv', sep=',', encoding='utf-8')
```

correlations.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 15 17:59:47 2022
```

```
@author: Mehmet Yigit Turali, Mehmet Efe Lorasdagi
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from statsmodels.graphics.tsaplots import plot_pacf,plot_acf

feat_eng_data = pd.read_csv('C:/Users/Mehmet Yigit Turali/Desktop/BILKENT/BILKENT FALL
2022-2023/EEE485/Project/feat_eng_data.csv')
#wind_power_variables = pd.read_csv('C:/Users/Mehmet Yigit Turali/Desktop/BILKENT/BILKENT
FALL 2022-2023/EEE485/Project/wind_power_all.csv')
feat_eng_data.rename(columns={'Power(kW)':'power'},inplace=True)
dataframe = feat_eng_data.set_index('Timestamp')
plt.rc("figure", figsize=(18, 12))
plot_pacf(dataframe.power, lags=6*18)

#plt.annotate('Strong correlation at lag = 1', xy=(1, 0.6),  xycoords='data', \
#            xytext=(0.17, 0.75), textcoords='axes fraction', \
#            arrowprops=dict(color='red', shrink=0.05, width=1))

plt.xlabel('Lags', fontsize=28)
plt.ylabel('Partial Autocorrelation', fontsize=28)
plt.title(f'Partial Autocorrelation ', fontsize=32)
plt.show()

plt.rc("figure", figsize=(18, 12))
plot_acf(dataframe.power, lags=6*18)
plt.xlabel('Lags', fontsize=28)
plt.ylabel(' Autocorrelation', fontsize=28)
plt.title(f' Autocorrelation', fontsize=32)
plt.show()

cor = feat_eng_data.filter(regex='power').corr()
fig,axes = plt.subplots(1,1,figsize=(21,10))
sb.heatmap(cor,annot=True, cmap ='icefire')
plt.title('Correlation Matrix')
plt.show()
```

lasso_reg.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 15 19:35:23 2022

@author: Mehmet Yigit Turali
"""
import numpy as np
```

```
import gridsearch_fortimeseries as grdsrch
import matplotlib.pyplot as plt
import time


class LASSO_Regression():
    #hyperparameter initialization
    def __init__(self,learning_rate,lambda_param,epoch_num,batch_size):
        self.learning_rate = learning_rate
        self.lambda_param = lambda_param
        self.epoch_num = epoch_num
        self.batch_size = batch_size

    def cost_function(self,y,y_pred):
        sum_error = 0.0
        for i in range(len(y)):
            prediction_error = y_pred[i] - y[i]
            sum_error += (prediction_error ** 2)

        mean_err = sum_error / float(len(y))
        return mean_err

    #dataset fitting
    def fit(self,X_train,Y_train,X_val,Y_val,fold_num):
        start = time.time()
        self.number_of_input = X_train.shape[0]
        self.number_of_ftrs = X_train.shape[1]

        self.w = np.random.randn(self.number_of_ftrs)
        self.b = 0
        self.cost_batch_arr = []
        self.val_cost_arr = []
        self.cost_arr = []
        self.X_train = X_train
        self.Y_train = Y_train

        batches_x,batches_y = self.create_batches(self.X_train, self.Y_train)
        print(len(batches_x))
        for epoch in range(self.epoch_num):
            for batch in range (len(batches_x)):
                self.X = batches_x[batch]#.to_numpy(dtype = "float32")
                self.Y = batches_y[batch]#.to_numpy(dtype = "float32").squeeze()
                print("Batch:{}/Total Batch:{}".format(batch,len(batches_x)))
                self.update_weights(epoch)
            self.cost_arr.append(np.mean(np.array(self.cost_batch_arr)))
            Y_hat_val = self.predict(X_val)
            cost_val = self.cost_function(Y_val, Y_hat_val)
            self.val_cost_arr.append(cost_val)
```

```
        cont = time.time()
        print("Time Elapsed:",cont-start)

np.save("numpy_files/train_cost_per_epoch_lambda_{}_lr_{}_fold_{}".format(self.lambda_param,s
elf.learning_rate,fold_num),np.array(self.cost_arr))

np.save("numpy_files/val_cost_per_epoch_lambda_{}_lr_{}_fold_{}".format(self.lambda_param,sel
f.learning_rate,fold_num),np.array(self.val_cost_arr))
        plt.figure()
        plt.plot(np.arange(0,len(self.cost_arr)),self.cost_arr,color="blue",label ="Training Error" )
        plt.plot(np.arange(0,len(self.val_cost_arr)),self.val_cost_arr,color="orange",label ="Validation
Error")
        plt.xlabel("Epoch Number")
        plt.ylabel("MSE")
        plt.legend(["Training Error","Validation Error"])
        plt.title("Training and Validation
Error(lambda_{}_lr_{}_fold_{})".format(self.lambda_param,self.learning_rate,fold_num))
        plt.grid()

plt.savefig("graphs/cost_per_epoch_lambda_{}_lr_{}_fold_{}.png".format(self.lambda_param,self.le
arning_rate,fold_num),dpi =800)
        plt.show()
        plt.clf()
        plt.close()

    def fit_wo_val(self,X_train,Y_train):
        start = time.time()
        self.number_of_input = X_train.shape[0]
        self.number_of_ftrs = X_train.shape[1]

        self.w = np.random.randn(self.number_of_ftrs)
        self.b = 0
        self.cost_batch_arr = []
        self.cost_arr = []
        self.X_train = X_train
        self.Y_train = Y_train

        for epoch in range(self.epoch_num):
            batches_x,batches_y = self.create_batches(self.X_train, self.Y_train)
            print(len(batches_x))
            for batch in range (len(batches_x)):
                self.X = batches_x[batch]#.to_numpy(dtype = "float32")
                self.Y = batches_y[batch]#.to_numpy(dtype = "float32").squeeze()
                print("Batch:{}/Total Batch:{}".format(batch,len(batches_x)))
                self.update_weights(epoch)
            self.cost_arr.append(np.mean(np.array(self.cost_batch_arr)))

        cont = time.time()
```

```
        print("Time Elapsed:",cont-start)

np.save("numpy_files/train_cost_per_epoch_lambda_{}_lr_{}".format(self.lambda_param,self.learni
ng_rate),np.array(self.cost_arr))
        plt.figure()
        plt.plot(np.arange(0,len(self.cost_arr)),self.cost_arr,color="blue",label ="Training Error" )
        plt.xlabel("Epoch Number")
        plt.ylabel("MSE")
        plt.legend("Training Error(lambda_{}_lr_{})".format(self.lambda_param,self.learning_rate))
        plt.grid()

plt.savefig("cost_per_epoch_lambda_{}_lr_{}.png".format(self.lambda_param,self.learning_rate),dpi
=800)
        plt.show()
        plt.clf()
        plt.close()

    def create_batches(self,x_train,y_train):
        mini_batches_x = []
        mini_batches_y = []
        #dataset_processed = np.stack((x_train,y_train),axis=1)
        batch_num = x_train.shape[0]//1440
        print("Number of batches is {}".format(batch_num))
        time.sleep(5)
        for b in range(batch_num):
            mini_batch_x = x_train[b*self.batch_size:(b+1)*self.batch_size]
            mini_batch_y = y_train[b*self.batch_size:(b+1)*self.batch_size]
            mini_batches_x.append(mini_batch_x)
            mini_batches_y.append(mini_batch_y)

        if x_train.shape[0] % self.batch_size != 0:
            mini_batch_x = x_train[b*self.batch_size:]
            mini_batch_y = y_train[b*self.batch_size:]
            mini_batches_x.append(mini_batch_x)
            mini_batches_y.append(mini_batch_y)

        return mini_batches_x,mini_batches_y


    #update function
    def update_weights(self,epoch):
        #print(self.X)
        Y_hat = self.predict(self.X)
        #print(Y_hat.shape)
        #print(self.Y.shape)
        cost = self.cost_function(self.Y,Y_hat)
        self.cost_batch_arr.append(cost)
        print("The Cost function for the epoch {}----->{} )".format(epoch, cost))
```

```
        dw = np.zeros(self.number_of_ftrs)
        db = 0
        #gradient descent algorithm
        for i in range(self.number_of_ftrs):
            if self.w[i] > 0:
                dw[i] = -2*(np.dot(self.X[:,i],(self.Y-Y_hat))+self.lambda_param)/self.number_of_input
            else:
                dw[i] = -2*(np.dot(self.X[:,i],(self.Y-Y_hat))-self.lambda_param)/self.number_of_input

            db = -2*np.sum(self.Y-Y_hat)/self.number_of_input

        self.w = self.w - self.learning_rate*dw
        self.b = self.b - self.learning_rate*db



    #Target Prediction
    def predict(self,X):
        yhat = np.dot(X,self.w)+self.b
        return yhat

    def save_model(self):
        weights = np.array(self.w)
        bias = np.array(self.b)
        lambda_ = self.lambda_param

        return weights,bias,lambda_

    def model_load(self,weights,bias,lambda_):
        self.w = weights
        self.b = bias
        self.lambda_param = lambda_

#comment below when model_test_script.py is running
X_train =
[["x_train_f0.csv","x_val_f0.csv"],["x_train_f1.csv","x_val_f1.csv"],["x_train_f2.csv","x_val_f2.csv"
],["x_train_f3.csv","x_val_f3.csv"]]
Y_train =
[["y_train_f0.csv","y_val_f0.csv"],["y_train_f1.csv","y_val_f1.csv"],["y_train_f2.csv","y_val_f2.csv"
],["y_train_f3.csv","y_val_f3.csv"]]
lambda_param = np.linspace(0,0.2,9)
learning_rates = np.logspace(-5,-1,num=9)
#lambda_param = [0.002]
#learning_rates = [0.001]
batch_size = 1440
epoch_num = 500
```

```
x,y,z,train_mse_score,cross_val_mse_score = grdsrch.GridSearch_Lasso(X_train, Y_train,
lambda_param, learning_rates,epoch_num,batch_size,LASSO_Regression)
grdsrch.plot_hyperparameters(x, y, z)
```

model_test_script.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 19 00:17:35 2022

@author: Mehmet Yigit Turali
"""
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import lasso_reg

def mean_squared_error(y,y_pred):
    sum_error = 0.0
    for i in range(len(y)):
        prediction_error = y_pred[i] - y[i]
        sum_error += (prediction_error ** 2)

    mean_err = sum_error / float(len(y))
    return mean_err

def r_squared(y_test,y_hat):
    y_mean = np.mean(y_test)
    ss_total = np.sum(np.power((y_test-y_mean),2))
    ss_res = np.sum(np.power((y_test-y_hat),2))

    return 1-(ss_res/ss_total)
target_ahead_timeblocks = 16
x_train_fold = pd.read_csv("x_train_4hrs.csv",header=None, index_col=False)
x_train_fold = x_train_fold.iloc[1: , 2:].to_numpy(dtype = "float32")

x_test_fold = pd.read_csv("x_test_4hrs.csv",header=None, index_col=False)
x_test_fold = x_test_fold.iloc[1: , 2:].to_numpy(dtype = "float32")

y_train_fold = pd.read_csv("y_train_4hrs.csv",header=None, index_col=False)
y_train_fold = y_train_fold.iloc[1: , 2:].to_numpy(dtype = "float32").squeeze()

y_test_fold = pd.read_csv("y_test_4hrs.csv", index_col=False)
y_test_fold.target = y_test_fold.target.shift(target_ahead_timeblocks)
y_test_fold = y_test_fold.iloc[0: , 2:].to_numpy(dtype = "float32").squeeze()

lambda_param = 1.0
learning_rate = 0.01
#lambda_param = [0.002]
```

```
#learning_rates = [0.001]
batch_size = 1440
epoch_num = 500
model = lasso_reg.LASSO_Regression(learning_rate,lambda_param,epoch_num,batch_size)
#model.fit_wo_val(x_train_fold,y_train_fold)
model.fit(x_train_fold,y_train_fold,x_test_fold,y_test_fold,"test")
predictions = np.array(model.predict(x_test_fold))
np.save("predictions.npy",np.array(predictions))

y_test_fold = pd.read_csv("y_test_4hrs.csv", index_col=False)
y_test_fold.target = y_test_fold.target.shift(target_ahead_timeblocks)
y_test_fold_r = y_test_fold.iloc[0: , 2:].to_numpy(dtype = "float32").squeeze()

rmse =
np.sqrt(mean_squared_error(y_test_fold_r[target_ahead_timeblocks:],predictions[target_ahead_timeb
locks:]))
r_squared =
r_squared(y_test_fold_r[target_ahead_timeblocks:],predictions[target_ahead_timeblocks:])
print("R^2 =",r_squared)
print("RMSE =",rmse)
y_test_fold['lasso_prediction'] = predictions



pd.DataFrame.to_csv(y_test_fold,'y_tested.csv', sep=',', encoding='utf-8')

y_test_fold = y_test_fold.iloc[0: , 2:].to_numpy(dtype = "float32").squeeze()
fig,axes = plt.subplots(1,1,figsize=(35,7))
plt.plot(y_test_fold[len(y_test_fold)-24*target_ahead_timeblocks:len(y_test_fold)-4*target_ahead_ti
meblocks])
plt.title('Plotting the Forecasted values')
plt.xlabel('Power (kW)')
plt.legend(["Target","Predicted"])
plt.savefig("test_forecast.png",dpi =800)
plt.show()

plt.plot(y_test_fold[len(y_test_fold)-8*target_ahead_timeblocks:len(y_test_fold)-4*target_ahead_tim
eblocks])
plt.title('Plotting the Forecasted values')
plt.xlabel('Power (kW)')
plt.legend(["Target","Predicted"])
plt.savefig("test_forecast_6hrs.png",dpi =800)
plt.show()
```

mlp_part.py

```
import pandas as pd
import numpy as np
from math import floor
from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error,mean_absolute_error
import matplotlib.pyplot as plt

x_train_fold = pd.read_csv(
   "x_train_4hrs.csv",
   index_col="Timestamp")
y_train_fold = pd.read_csv(
   "y_train_4hrs.csv",
   index_col="Timestamp")
y_train = np.array(y_train_fold.iloc[:, 1:])
X_train = x_train_fold[['feature_lagging_power_1', 'Blade-3 Actual Value_Angle-A',
'rolling_4_power_mean', 'tb',
                'Blade-2 Actual Value_Angle-A', 'Torque', 'Nacelle Revolution', 'Voltage A-N',
                'Tower Acceleration Normal', 'Scope CH 4', 'Hydraulic Prepressure',
                'Tower Acceleration Lateral', 'Reactive Power', 'Blade-1 Actual Value_Angle-A',
                'feature_lagging_power_24', 'feature_lagging_power_72',"power"]]

# X_train = x_train_fold
x_test_fold = pd.read_csv("x_test_4hrs.csv",index_col="Timestamp")
x_test_fold = x_test_fold.iloc[:, 2:]
X_test = x_test_fold[['feature_lagging_power_1', 'Blade-3 Actual Value_Angle-A',
'rolling_4_power_mean', 'tb',
                'Blade-2 Actual Value_Angle-A', 'Torque', 'Nacelle Revolution', 'Voltage A-N',
                'Tower Acceleration Normal', 'Scope CH 4', 'Hydraulic Prepressure',
                'Tower Acceleration Lateral', 'Reactive Power', 'Blade-1 Actual Value_Angle-A',
                'feature_lagging_power_24', 'feature_lagging_power_72',"power"]]
# y_test = pd.read_csv(
#     "y_test_4hrs.csv",
#     index_col="Timestamp")
# y_test = np.array(y_test.iloc[:, 1:])
y_test_fold = pd.read_csv("y_test_4hrs.csv", index_col=False)
y_test_fold.target = y_test_fold.target.shift(16)
y_test_fold_r = y_test_fold.iloc[0: , 2:].to_numpy(dtype = "float32")


# X_test = x_test_fold
def r_squared(y_hat,y_test):
   y_mean = np.mean(y_test)
   ss_total = np.sum(np.power((y_test-y_mean),2))
   ss_res = np.sum(np.power((y_test-y_hat),2))
```

```
    return 1-(ss_res/ss_total)

class two_hidden_layer_MLP():
    def __init__(self):
        pass
    def leaky_relu(self,x):
        return np.where(x > 0, x, x * 0.01)
    def xavier_uniform(self,s_in, s_out):
        limit = np.sqrt(6 / float(s_in + s_out))
        W = np.random.uniform(low=-limit, high=limit, size=(s_in, s_out))
        return W
    def init_parameters(self, layer_size1, layer_size2, data):
        w1 =self.xavier_uniform(np.shape(data)[1],layer_size1 ) *0.001
        b1 = self.xavier_uniform(1, layer_size1)*0.001
        w2 =self.xavier_uniform(layer_size1, layer_size2)*0.001
        b2=self.xavier_uniform(1, layer_size2)*0.001
        w3=self.xavier_uniform(layer_size2, 1)*0.001
        b3=self.xavier_uniform(1,1)*0.001
        return w1,w2,w3, b1,b2,b3
    def linear_forward(self,data_batch,w1,w2,w3, b1,b2,b3):
        z1=np.matmul(data_batch, w1) + b1
        a1=self.leaky_relu(z1)
        # print(np.shape(a1))
        z2=np.matmul(a1, w2) + b2
        a2=self.leaky_relu(z2)
        z3=np.matmul(a2, w3) + b3
        a3=self.leaky_relu(z3)

        cache = {"z1":z1,"a1":a1,"z2":z2,"a2":a2,"z3":z3,"a3":a3}
        return a3, cache
    def compute_loss(self,yhat,y):
        error = yhat-y
        return np.square(error)/len(error)

    def back_pass(self, yhat, y,x, cache, w1,w2,w3, b1,b2,b3):
        dyhat = 2*(yhat-y)
        m = len(yhat)
        z1=cache["z1"]
        z2=cache["z2"]
        z3=cache["z3"]
        a1=cache["a1"]
        a2=cache["a2"],
        a3=cache["a3"]
        temp1 = np.zeros_like(a3)
        dz3 = np.where(a3>temp1, dyhat, 0.1*dyhat)
        # print(np.shape(dz3))
        # print(np.shape(a2))
```

```
    a2= np.squeeze(a2, axis=0)
    dw3=np.matmul(dz3.T, a2).T
    # print(np.shape(dw3))
    db3=np.sum(dz3, axis=0)
    da2=np.matmul(dz3, w3.T)
    temp2 = np.zeros_like(a2)
    dz2 = np.where(a2>temp2, da2, 0.1*da2)

    dw2=np.matmul(dz2.T,a1).T
    db2=np.sum(dz2, axis=0)

    da1=np.matmul(dz2,w2.T)
    temp3 = np.zeros_like(a1)
    dz1 = np.where(a1>temp3, da1, 0.1*da1)

    dw1=np.matmul(dz1.T, x).T
    db1=np.sum(dz1 , axis=0)

    dw1=dw1.clip(-2,2)
    dw2=dw2.clip(-2,2)
    dw3=dw3.clip(-2,2)
    db1=db1.clip(-2,2)
    db2 = db2.clip(-2,2)
    db3=db3.clip(-2,2)
    grads={"dw3":dw3/m,"db3":db3/m,"dw2":dw2/m,"db2":db2/m, "dw1":dw1/m,"db1":db1/m}
    return grads

def update_params(self, grads, prev_grads, learning_rate, momentum, w1,w2,w3, b1,b2,b3):
    dw1=grads["dw1"]
    dw2=grads["dw2"]
    dw3=grads["dw3"]
    db1=grads["db1"]
    db2=grads["db2"],
    db3=grads["db3"]
    pdw1=prev_grads["dw1"]
    pdw2=prev_grads["dw2"]
    pdw3=prev_grads["dw3"]
    pdb1=prev_grads["db1"]
    pdb2=prev_grads["db2"],
    pdb3=prev_grads["db3"]
    w1 -= learning_rate*dw1 + momentum * pdw1
    w2 -= learning_rate*dw2 + momentum * pdw2
    w3 -= learning_rate*dw3 + momentum * pdw3
    b1 -= learning_rate*db1 + momentum * pdb1
    b2 -= learning_rate*np.asarray(db2) + momentum *np.asarray(pdb2)
    b3 -= learning_rate*db3 + momentum * pdb3
    return w1,w2,w3, b1,b2,b3
def create_batches(self,x_train,y_train, batch_size):
```

```
        mini_batches_x = []
        mini_batches_y = []
        #dataset_processed = np.stack((x_train,y_train),axis=1)
        batch_num = x_train.shape[0]//batch_size
        print("Number of batches is {}".format(batch_num))
        for b in range(batch_num):
            mini_batch_x = x_train[b*batch_size:(b+1)*batch_size]
            mini_batch_y = y_train[b*batch_size:(b+1)*batch_size]
            mini_batches_x.append(mini_batch_x)
            mini_batches_y.append(mini_batch_y)

        if x_train.shape[0] % batch_size != 0:
            mini_batch_x = x_train[b*batch_size:]
            mini_batch_y = y_train[b*batch_size:]
            mini_batches_x.append(mini_batch_x)
            mini_batches_y.append(mini_batch_y)

        return mini_batches_x,mini_batches_y, batch_num
    def plot_loss(self,J_train_arr,J_val_arr) :
        plt.figure()
        plt.plot(np.arange(0,len(J_train_arr)),J_train_arr,color="blue",label ="Training Error" )
        plt.plot(np.arange(0,len(J_val_arr)),J_val_arr,color="orange",label ="Validation Error")
        plt.xlabel("Epoch Number")
        plt.ylabel("Categorical Cross-Entropy Loss")
        plt.legend(["Training Error","Validation Error"])
        plt.title("Training and Validation Error")
        plt.grid()
        plt.show()
        plt.clf()
        plt.close()
    def training(self, data, labels,lr, momentum, epochs, batch_size, lsize1, lsize2):
        w1,w2,w3, b1,b2,b3 = self.init_parameters(lsize1, lsize2, data)
        prev_grads = {"dw3": 0, "db3": 0, "dw2": 0, "db2":0, "dw1": 0, "db1": 0}
        J_train_arr = []
        J_val_arr = []
        for e in range(epochs):

            data_tr, data_val, label_tr, label_val =train_test_split(data, labels, test_size=.2,
random_state=41)
            data_batches, label_batches, nb = self.create_batches(data_tr, label_tr, batch_size)
            for i in range(nb):
                batch_x=  data_batches[i]
                batch_y= label_batches[i]
                pred, cache = self.linear_forward(batch_x, w1,w2,w3, b1,b2,b3)
                loss = self.compute_loss(pred, batch_y)
                grads = self.back_pass(pred, batch_y,batch_x, cache, w1,w2,w3, b1,b2,b3)
                w1,w2,w3, b1,b2,b3 = self.update_params(grads, prev_grads, lr, momentum, w1,w2,w3,
b1,b2,b3)
```

```
            # print(w1)
            pred_tr, _ = self.linear_forward(data_tr, w1,w2,w3, b1,b2,b3)
            pred_val, _ = self.linear_forward(data_val, w1,w2,w3, b1,b2,b3)
            J_val = mean_squared_error(pred_val,label_val)
            J_val_arr.append(J_val)
            print('------------------------')
            print("Validation Cost after iteration %i: %f" % (e, J_val))
            print('------------------------')

            J_train = mean_squared_error(pred_tr,label_tr)
            J_train_arr.append(J_train)
            print("Training Cost after epoch %i: %f" % (e, J_train))

        self.plot_loss(J_train_arr, J_val_arr)

        return w1,w2,w3, b1,b2,b3, J_train_arr, J_val_arr

import time
from sklearn.metrics import mean_squared_error,mean_absolute_error


def GridSearch_MLP(X,Y,layer1,layer2, batch_size, lr, momentum, epochs):
    train_mse_score_mean = []
    cross_val_mse_score_mean = []
    g_x = []
    g_y = []
    g_z = []
    X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=.2, random_state=41)
    print(Y_val.shape)
    print(Y_val.shape)
    for layer_size_1 in layer1:
        train_mse_score_fold = []
        cross_val_mse_score_fold = []
        x_row = []
        y_row = []
        z_row = []
        for layer_size_2 in layer2:
            train_mse_score_lr = []
            cross_val_mse_score_lr = []
            model = two_hidden_layer_MLP()
            # data_train, label_train, nb = model.create_batches(X_train, Y_train, batch_size)
            # data_val, label_val, nb = model.create_batches(X_val, Y_val, batch_size)
            # for batch in range(nb):
            #     cur_x =  data_train[batch]
            #     cur_y = label_train[batch]
            model = two_hidden_layer_MLP()
```

```
        w1,w2,w3, b1,b2,b3, _,_ = model.training(X_train,Y_train, lr, momentum, epochs, batch_size,
layer_size_1, layer_size_2)
        y_predicted_val, _ = model.linear_forward(X_val, w1,w2,w3, b1,b2,b3)
        # print(np.shape(y_predicted_val))
        print(np.sqrt(mean_squared_error(Y_val[16:],y_predicted_val[16:])))
        mse_err_val = np.sqrt(mean_squared_error(Y_val[16:], y_predicted_val[16:]))
        cross_val_mse_score_lr.append(round(mse_err_val,4))
        y_predicted_train,_ = model.linear_forward(X_train, w1,w2,w3, b1,b2,b3)
        print(np.sqrt(mean_squared_error(Y_train[16:],y_predicted_train[16:])))
        mse_err_train = np.sqrt(mean_squared_error(Y_train[16:],y_predicted_train[16:]))
        train_mse_score_lr.append(round(mse_err_train,4))
        print("---------batch finished---------")

        time.sleep(3)
        train_mse_score_fold.append(train_mse_score_lr)
        cross_val_mse_score_fold.append(cross_val_mse_score_lr)
        x_row.append(layer_size_1)
        y_row.append(layer_size_2)
        z_row.append((np.mean(np.array(cross_val_mse_score_fold))))

    print(np.array(train_mse_score_fold))
    train_mse_score_mean =
0#np.concatenate((np.expand_dims(lambda_param,axis=1),np.array(train_mse_score_fold).squeeze())
,axis=1)
    cross_val_mse_score_mean =
0#np.concatenate((np.expand_dims(lambda_param,axis=1),np.array(cross_val_mse_score_fold).sque
eze()),axis=1)
    g_x.append(x_row)
    g_y.append(y_row)
    g_z.append(z_row)

  graph_x=np.array(g_x)
  graph_y=np.array(g_y)
  graph_z=np.array(g_z)
  min_z = np.min(graph_z)
  position_min_z = np.argwhere(graph_z == np.min(graph_z))[0]
  print('Minimum MSE: {:.4f}'.format(min_z))
  print('Best max_depth: {}' .format(graph_x[position_min_z[0],position_min_z[1]]))
  print('Best min_samples_split: {}'.format(graph_y[position_min_z[0],position_min_z[1]]))
  return graph_x,graph_y,graph_z,train_mse_score_mean,cross_val_mse_score_mean


# graph_x,graph_y,graph_z,train_mse_score_mean,cross_val_mse_score_mean =
GridSearch_MLP(X_train,y_train,[12,10,8],[6,4,2], batch_size=512 ,lr=0.075, momentum =
0.5,epochs=50)
# np.save( "graph_x_dene", graph_x)
# np.save( "graph_y_dene", graph_y)
# np.save( "graph_z_dene", graph_z)
```

```
# np.save( "train_mse_score_mean_dene", train_mse_score_mean)
# np.save( "cross_val_mse_score_mean_dene", cross_val_mse_score_mean)
# x= np.load("graph_x_dene.npy")
# y= np.load("graph_y_dene.npy")
# z= np.load("graph_z_dene.npy")
# # # a=np.load("graph_x_dene")
# # # a=np.load("graph_x_dene")
# # print(z)
def plot_hyperparameters(x,y,z):
    plt.yscale('log')
    points=plt.scatter(x, y, c=z, cmap='plasma') #,s = (z*30/pow(10, 6))) # 3,s = (z*30/pow(10, 6)))
    color_bar=plt.colorbar(points)
    color_bar.set_label("$MSE$")
    plt.xlabel('Size of first hidden layer')
    plt.ylabel('Size of second hidden layer')
    # plt.xticks(np.arange(0,0.2,0.02))
    plt.savefig('plot_hyperparameters.png',format='png',dpi=1600)

x_train, X_val, Y_train, Y_val = train_test_split(X_train, y_train, test_size=.2, random_state=41)

model = two_hidden_layer_MLP()
w1, w2, w3, b1, b2, b3, train_err, val_err = model.training(X_train, y_train, 0.2, 0.25, 300, 512, 12, 6)
np.save( "w1_300epoch", w1)
np.save( "w2_300epoch", w2)
np.save( "w3_300epoch", w3)
np.save( "b1_300epoch", b1)
np.save( "b2_300epoch", b2)
np.save( "b3_300epoch", b3)

# w1=np.load( "w1.npy" )
# w2=np.load( "w2.npy" )
# w3=np.load( "w3.npy" )
# b1=np.load( "b1.npy" )
# b2=np.load( "b2.npy" )
# b3=np.load( "b3.npy" )
# print(w1)
y_predicted_val, _ = model.linear_forward(X_val, w1, w2, w3, b1, b2, b3)
# print(np.shape(y_predicted_val))
# print("RMSE Error "np.sqrt(mean_squared_error(Y_val[16:], y_predicted_val[16:])))
mse_err_val = np.sqrt(mean_squared_error(Y_val[16:], y_predicted_val[16:]))
print("MSE Error Validation: " + str(round(mse_err_val, 4)))
y_predicted_train, _ = model.linear_forward(x_train, w1, w2, w3, b1, b2, b3)
# print(np.sqrt(mean_squared_error(Y_train[16:], y_predicted_train[16:])))
mse_err_train = np.sqrt(mean_squared_error(Y_train[16:], y_predicted_train[16:]))
print("MSE Error Train: " + str(round(mse_err_train, 4)))

# train_mse_score_lr.append(round(mse_err_train, 4))
print("---------Training finished---------")
```

```
y_predicted_test, _ = model.linear_forward(X_test, w1, w2, w3, b1, b2, b3)
# print(np.sqrt(mean_squared_error(Y_train[16:], y_predicted_train[16:])))
# mse_err_train = np.sqrt(mean_squared_error(y_predicted_test[16:416], y_predicted_train[16:416]))
# print("MSE Error Test: " + str(round(mse_err_train, 4)))
score = np.sqrt(mean_squared_error(y_test_fold_r[16:416],y_predicted_test[16:416]))
r2 = r_squared(y_predicted_test[16:416], y_test_fold_r[16:416])
print("RMSE Test: " + str(round(score, 4)))
print("R2 Test: " + str(round(r2, 4)))
fig,axes = plt.subplots(1,1,figsize=(35,7))
plt.plot(y_test_fold_r[16:416])
plt.plot(y_predicted_test[16:416])

plt.title('Plotting the Forecasted values')
plt.xlabel('Power (kW)')
plt.legend(["Target","Predicted"])
plt.savefig("test_forecast_4hrs.png",dpi =800)
plt.show()
```

decisiontree.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
def mean_squared_error(y,y_pred):
    sum_error = 0.0
    for i in range(len(y)):
        prediction_error = y_pred[i] - y[i]
        sum_error += (prediction_error ** 2)

    mean_err = sum_error / float(len(y))
    return mean_err

def MAPE(y_gt,y_hat):
    mape = np.mean(np.abs((y_gt - y_hat)/y_gt))*100
    return mape

def mae_metric(y_gt,y_hat):
    sum_error = 0.0
    for i in range(len(y_gt)):
        sum_error += abs(y_hat[i] - y_gt[i])
    return sum_error / float(len(y_gt))

def r_squared(y_test,y_hat):
    y_mean = np.mean(y_test)
    ss_total = np.sum(np.power((y_test-y_mean),2))
    ss_res = np.sum(np.power((y_test-y_hat),2))
```

```
    return 1-(ss_res/ss_total)

class Node():
    def __init__(self,feat_idx = None,threshold =None,left_n = None, right_n = None,var_reduc =
None,node_val = None):
        self.feat_idx =  feat_idx
        self.threshold = threshold
        self.left_n = left_n
        self.right_n = right_n
        self.var_reduc = var_reduc
        self.node_val = node_val

class DecisionTreeRegressor():
    def __init__(self,min_samples_split = 2,max_depth = 2):
        self.root = None
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth

    def grow_tree(self,dataset,current_depth= 0 ):
        X,y = dataset[:,:-1],dataset[:,-1]
        num_samples, num_features = np.shape(X)
        best_split = {}
        if num_samples >= self.min_samples_split and current_depth <= self.max_depth:
            best_split = self.get_best_split(dataset, num_samples, num_features)
            if best_split["var_reduc"] >0:
                left_node = self.grow_tree(best_split["dataset_left"],current_depth+1)
                right_node = self.grow_tree(best_split["dataset_right"],current_depth+1)

                return
Node(best_split["feature_idx"],best_split["threshold"],left_node,right_node,best_split["var_reduc"])

        leaf_val = self.calc_leaf_val(y)
        return Node(node_val=leaf_val)


    def get_best_split(self,dataset,num_samples,num_features):

        best_split = {}
        max_var_reduc = -float("inf")
        for f_idx in range(num_features):
            feature_vals = dataset[:,f_idx]
            possib_th = np.unique(feature_vals)
            for th  in possib_th:
                dataset_left,dataset_right = self.split(dataset,f_idx,th)
                if len(dataset_left) > 0 and len(dataset_right) > 0:
                    y,left_y,right_y = dataset[:,-1],dataset_left[:,-1],dataset_right[:,-1]
                    current_var_reduc = self.variance_reduction(y,left_y,right_y)
```

```
                if current_var_reduc > max_var_reduc:
                    best_split["feature_idx"] = f_idx
                    best_split["threshold"] = th
                    best_split["dataset_left"] = dataset_left
                    best_split["dataset_right"] = dataset_right
                    best_split["var_reduc"] = current_var_reduc

                    max_var_reduc = current_var_reduc

        return best_split

    def split(self,dataset,feat_idx,threshold):
        dataset_left = np.array([r for r in dataset if r[feat_idx]<=threshold])
        dataset_right = np.array([r for r in dataset if r[feat_idx]>threshold])

        return dataset_left,dataset_right

    def variance_reduction(self,parent,left_child,right_child):

        weight_left = len(left_child)/len(parent)
        weight_right = len(right_child)/len(parent)

        reduct = np.var(parent) - ( weight_left*np.var(left_child) + weight_right * np.var(right_child))
        return reduct

    def calc_leaf_val(self,y):
        val = np.mean(y)
        return val


    def print_tree(self, tree = None, indent = " "):

        if not tree:
            tree = self.root

        if tree.node_val is not None:
            print(tree.node_val)

        else:
            print("X_"+str(tree.feat_idx),"<=",tree.threshold,"?",tree.var_reduc)
            print("%sleft:" % (indent), end="")
            self.print_tree(tree.left_n,indent + indent)
            print("%sright:" % (indent), end="")
            self.print_tree(tree.right_n,indent + indent)

    def fit(self,X,y):
        dataset = np.concatenate((X,y),axis=1)
```

```
        self.root = self.grow_tree(dataset)

    def make_predictions(self,x,tree):

        if tree.node_val != None:
            return tree.node_val

        feat_val = x[tree.feat_idx]

        if feat_val<=tree.threshold:
            return self.make_predictions(x,tree.left_n)

        else:
            return self.make_predictions(x,tree.right_n)

    def predict(self,X):

        preds = [self.make_predictions(x,self.root) for x in X]
        return preds

x_train_fold = pd.read_csv(
  "C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_4hrs_train.csv",
  index_col="Timestamp")
y_train_fold = pd.read_csv(
  "C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_4hrs_train.csv",
  index_col="Timestamp")
y_train = np.array(y_train_fold.iloc[:, 1:])
X_train = x_train_fold[['feature_lagging_power_1', 'Blade-3 Actual Value_Angle-A',
'rolling_4_power_mean', 'tb',
                  'Blade-2 Actual Value_Angle-A', 'Torque', 'Nacelle Revolution', 'Voltage A-N',
                  'Tower Acceleration Normal', 'Scope CH 4', 'Hydraulic Prepressure',
                  'Tower Acceleration Lateral', 'Reactive Power', 'Blade-1 Actual Value_Angle-A',
                  'feature_lagging_power_24', 'feature_lagging_power_72',"power"]]

x_test_fold = pd.read_csv(
  "C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_4hrs_test.csv",
  index_col="Timestamp")
x_test_fold = x_test_fold.iloc[:, 2:]
X_test = x_test_fold[['feature_lagging_power_1', 'Blade-3 Actual Value_Angle-A',
'rolling_4_power_mean', 'tb',
                  'Blade-2 Actual Value_Angle-A', 'Torque', 'Nacelle Revolution', 'Voltage A-N',
                  'Tower Acceleration Normal', 'Scope CH 4', 'Hydraulic Prepressure',
                  'Tower Acceleration Lateral', 'Reactive Power', 'Blade-1 Actual Value_Angle-A',
                  'feature_lagging_power_24', 'feature_lagging_power_72',"power"]]
```

```
regressor = DecisionTreeRegressor(min_samples_split=3,max_depth=5)
regressor.fit(X_train.values[-8000:],y_train[-8000:])
y_test_fold = pd.read_csv("C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/yigit_mefe/4hrs/y_test_4hrs.csv", index_col=False)
y_test_fold.target = y_test_fold.target.shift(16)
y_test_fold_r = y_test_fold.iloc[0: , 2:].to_numpy(dtype = "float32").squeeze()
Y_pred = regressor.predict(X_test.values)
rmse = np.sqrt(mean_squared_error(y_test_fold_r[16:216],Y_pred[16:216]))
r_squared = r_squared(y_test_fold_r[16:216],Y_pred[16:216])
print("R^2 =",r_squared)
print("RMSE =",rmse)
```

GridSearchCV_Decision_Trees.py

```
def plot_hyperparameters(x,y,z):
    points=plt.scatter(x, y, c=z, cmap='plasma')
    color_bar=plt.colorbar(points)
    color_bar.set_label("$MSE$")
    plt.ylabel('max_depth')
    plt.xlabel('min_sample_split')
    plt.yticks([0,3,4,5])
    plt.xticks([0,3,6,9,12,15])
    plt.savefig('plot_hyperparameters.png',format='png',dpi=800)

def GridSearch_Regression_Trees(X_train,Y_train,min_split,max_depth):
    train_mse_score_mean = []
    cross_val_mse_score_mean = []
    g_x = []
    g_y = []
    g_z = []
    for md in max_depth:
        print("Testing Max_depth:",md)
        train_mse_score_fold = []
        cross_val_mse_score_fold = []
        x_row = []
        y_row = []
        z_row = []
        for ms in min_split:
            print("Testing Min_split:",ms)
            train_mse_score_lr = []
            cross_val_mse_score_lr = []
            for fold in range(len(X_train)):
                print("Fold number:{}".format(fold))

                x_train_fold = pd.read_csv(X_train[fold][0],index_col="Timestamp")
                x_train_fold = x_train_fold[['feature_lagging_power_1', 'Blade-3 Actual Value_Angle-A',
'rolling_4_power_mean', 'tb',
                    'Blade-2 Actual Value_Angle-A', 'Torque', 'Nacelle Revolution', 'Voltage A-N',
```

```
                'Tower Acceleration Normal', 'Scope CH 4', 'Hydraulic Prepressure',
                'Tower Acceleration Lateral', 'Reactive Power', 'Blade-1 Actual Value_Angle-A',
                'feature_lagging_power_24', 'feature_lagging_power_72',"power","ramp"]]

            x_val_fold = pd.read_csv(X_train[fold][1],index_col="Timestamp")
            x_val_fold = x_val_fold[['feature_lagging_power_1', 'Blade-3 Actual Value_Angle-A',
'rolling_4_power_mean', 'tb',
                'Blade-2 Actual Value_Angle-A', 'Torque', 'Nacelle Revolution', 'Voltage A-N',
                'Tower Acceleration Normal', 'Scope CH 4', 'Hydraulic Prepressure',
                'Tower Acceleration Lateral', 'Reactive Power', 'Blade-1 Actual Value_Angle-A',
                'feature_lagging_power_24', 'feature_lagging_power_72',"power","ramp"]]

            y_train_fold = pd.read_csv(Y_train[fold][0],index_col="Timestamp")
            y_train_fold.target = y_train_fold.target.shift(16)
            y_train_fold = np.array(y_train_fold.iloc[:, 1:])

            y_val_fold = pd.read_csv(Y_train[fold][1],index_col="Timestamp")
            y_val_fold.target = y_val_fold.target.shift(16)
            y_val_fold = np.array(y_val_fold.iloc[:, 1:])


            model = DecisionTreeRegressor(min_samples_split=ms,max_depth=md)
            print(model.min_samples_split,model.max_depth)
            s = time.time()
            model.fit(x_train_fold.values[-3500:],y_train_fold[-3500:])
            e = time.time()
            print("Time Passed:",s-e)
            y_predicted_val = model.predict(x_val_fold.values)
            print(np.sqrt(mean_squared_error(y_val_fold[16:],y_predicted_val[16:])))
            mse_err_val = np.sqrt(mean_squared_error(y_val_fold[16:],y_predicted_val[16:]))
            cross_val_mse_score_lr.append(round(mse_err_val,4))

            y_predicted_train = model.predict(x_train_fold.values)
            print(np.sqrt(mean_squared_error(y_train_fold[16:],y_predicted_train[16:])))
            mse_err_train = np.sqrt(mean_squared_error(y_train_fold[16:],y_predicted_train[16:]))
            train_mse_score_lr.append(round(mse_err_train,4))
        print("---------fold finished---------")

        time.sleep(3)
        train_mse_score_fold.append(train_mse_score_lr)
        cross_val_mse_score_fold.append(cross_val_mse_score_lr)
        x_row.append(md)
        y_row.append(ms)
        z_row.append((np.mean(np.array(cross_val_mse_score_fold))))

    print(np.array(train_mse_score_fold))
```

```
        train_mse_score_mean =
0#np.concatenate((np.expand_dims(lambda_param,axis=1),np.array(train_mse_score_fold).squeeze())
,axis=1)
        cross_val_mse_score_mean =
0#np.concatenate((np.expand_dims(lambda_param,axis=1),np.array(cross_val_mse_score_fold).sque
eze()),axis=1)
        g_x.append(x_row)
        g_y.append(y_row)
        g_z.append(z_row)

    graph_x=np.array(g_x)
    graph_y=np.array(g_y)
    graph_z=np.array(g_z)
    min_z = np.min(graph_z)
    position_min_z = np.argwhere(graph_z == np.min(graph_z))[0]
    print('Minimum MSE: {:.4f}'.format(min_z))
    print('Best max_depth: {}' .format(graph_x[position_min_z[0],position_min_z[1]]))
    print('Best min_samples_split: {}'.format(graph_y[position_min_z[0],position_min_z[1]]))
    return graph_x,graph_y,graph_z,train_mse_score_mean,cross_val_mse_score_mean
X_train = [["C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_train_4hrs_f0.csv","C:/Users/Mehme
t Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_val_4hrs_f0.csv"],
["C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_train_4hrs_f1.csv","C:/Users/Mehme
t Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_val_4hrs_f1.csv"],
["C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_train_4hrs_f2.csv","C:/Users/Mehme
t Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_val_4hrs_f2.csv"],
["C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_train_4hrs_f3.csv","C:/Users/Mehme
t Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_val_4hrs_f3.csv"],
["C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_train_4hrs_f4.csv","C:/Users/Mehme
t Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_val_4hrs_f4.csv"],
["C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_train_4hrs_f5.csv","C:/Users/Mehme
t Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_val_4hrs_f5.csv"],
["C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_train_4hrs_f6.csv","C:/Users/Mehme
t Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_val_4hrs_f6.csv"],
["C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_train_4hrs_f7.csv","C:/Users/Mehme
t Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_val_4hrs_f7.csv"],
["C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_train_4hrs_f8.csv","C:/Users/Mehme
t Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_val_4hrs_f8.csv"],
```

["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_train_4hrs_f9.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_val_4hrs_f9.csv"]]

Y_train = [["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_train_4hrs_f0.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_val_4hrs_f0.csv"],
["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_train_4hrs_f1.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_val_4hrs_f1.csv"],
["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_train_4hrs_f2.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_val_4hrs_f2.csv"],
["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_train_4hrs_f3.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_val_4hrs_f3.csv"],
["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_train_4hrs_f4.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_val_4hrs_f4.csv"],
["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_train_4hrs_f5.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_val_4hrs_f5.csv"],
["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_train_4hrs_f6.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_val_4hrs_f6.csv"],
["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_train_4hrs_f7.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_val_4hrs_f7.csv"],
["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_train_4hrs_f8.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_val_4hrs_f8.csv"],
["C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_train_4hrs_f9.csv","C:/Users/Mehmet Yigit Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_val_4hrs_f9.csv"]]
min_split = [3,4,5]
max_depth = [3,6,9,12]
x,y,z,train_mse_score,cross_val_mse_score = GridSearch_Regression_Trees(X_train,Y_train,min_split,max_depth)
plot_hyperparameters(x, y, z)

RandomForest.py

```
import numpy as np
import pandas as pd
import math

class DecisionNode:
```

```
    """
    Class for a parent/leaf node in the decision tree.
    A Node with node information about it's left and right nodes if any. it has the impurity info also.
    """
    def __init__(self, imp=None, threshold_decision=None, feat_idx=None, threshold=None,
                 left_sbtree=None, right_sbtree=None):
        """
        :param
        """
        self.imp = imp
        self.threshold_decision = threshold_decision
        self.feat_idx = feat_idx
        self.threshold = threshold
        self.left_subtree = left_sbtree
        self.right_subtree = right_sbtree

class leaf_node:
    def __init__(self, val):
        self.pred_val = val
class DecisionTree:
    def __init__(self, min_sample_split=3, min_imp=1e-8, max_depth=float('inf'),
                 imp_func=None, leaf_node_calc=None):
        """
        """
        self.root = None
        self.min_sample_split = min_sample_split
        self.min_imp = min_imp
        self.max_depth = max_depth
        self.imp_func = imp_func
        self.leaf_node_calc = leaf_node_calc

    def _split_datamatrix(self, Xy, feat_idx, threshold):

        left_split = np.array([r for r in Xy if r[feat_idx] <= threshold])
        right_split = np.array([r for r in Xy if r[feat_idx] > threshold])

        return left_split, right_split

    def _best_split_finder(self, Xy):

        best_threshold_decision = tuple()
        best_datasplit = {}
        max_imp = 0
        n_features = (Xy.shape[1] - 1)
        for feat_idx in range(n_features):
            unique_val = set(s for s in Xy[:,feat_idx])
            for threshold in unique_val:
                left_datamatrix, right_datamatrix = self._split_datamatrix(Xy, feat_idx, threshold)
```

```
            if len(left_datamatrix) > 0 and len(right_datamatrix) > 0:

                y = Xy[:, -1]
                true_y = left_datamatrix[:, -1]
                false_y = right_datamatrix[:, -1]

                imp = self.imp_func(y, true_y, false_y)

                if imp > max_imp:
                    max_imp= imp
                    best_threshold_decision = (feat_idx, threshold)
                    best_datasplit = {
                            "leftX": left_datamatrix[:, :n_features],
                            "lefty": left_datamatrix[:, n_features:],
                            "rightX": right_datamatrix[:, :n_features],
                            "righty": right_datamatrix[:, n_features:]
                    }

        return max_imp, best_threshold_decision, best_datasplit

    def _build_tree(self, X, y, current_depth=0):
        n_samples , n_features = X.shape
        Xy = np.concatenate((X, y), axis=1)
        if (n_samples >= self.min_sample_split) and (current_depth <= self.max_depth):
            imp, decision_q, best_datasplit = self._best_split_finder(Xy)
            if imp > self.min_imp:
                left_branch = self._build_tree(best_datasplit["leftX"], best_datasplit["lefty"], current_depth
+ 1)
                right_branch = self._build_tree(best_datasplit["rightX"], best_datasplit["righty"],
current_depth + 1)
                return DecisionNode( imp=imp, threshold_decision=decision_q, feat_idx=decision_q[0],
threshold=decision_q[1],
                            left_sbtree=left_branch, right_sbtree=right_branch)

        leaf_val = self._leaf_val_calc(y)
        return leaf_node(val=leaf_val)


    def train(self, X, y):
        self.root = self._build_tree(X, y, current_depth=0)

    def predict_sample(self, x, tree=None):
        if tree is None:
            tree = self.root
        if isinstance(tree, leaf_node):

            return tree.pred_val
        feature_val = x[tree.feat_idx]
```

```
        branch = tree.right_subtree

        if isinstance(feature_val, int) or isinstance(feature_val, float):

            if feature_val >= tree.threshold:
                branch = tree.left_subtree

            elif feature_val == tree.threshold:
                branch = tree.left_subtree

        return self.predict_sample(x, branch)

    def predict(self, test_X):
        test = np.array(test_X)
        y_pred = [self.predict_sample(x) for x in test]
        return y_pred

    def print_tree(self, tree = None, indents = " "):
        if tree is None:
            tree = self.root

        def print_threshold_decision(threshold_decision, indents):
            feat_idx = threshold_decision[0]
            threshold = threshold_decision[1]
            condition = "=="
            if isinstance(threshold, int) or isinstance(threshold, float):
                condition = ">="
            print(indents,"Is {col} {condition} {val}?".format(col=feat_idx, condition=condition,
val=threshold))

        if isinstance(tree , leaf_node):
            print(indents,"The predicted val -->", tree.pred_val)
            return

        else:

            print_threshold_decision(tree.threshold_decision,indents)
            if tree.left_subtree  is not None:
.
                print (indents + '----- True branch :)')
                self.print_tree(tree.left_subtree , indents + "  ")
            if tree.right_subtree is not None:

                print (indents + '----- False branch :)')
                self.print_tree(tree.right_subtree, indents + "  ")

class DecisionTreeRegressor(DecisionTree):
```

```
    def __init__(self, min_sample_split=3, min_imp=1e-8, max_depth=float('inf'),
                 ):
        self._imp_func = self._calc_var_reduct
        self._leaf_val_calc = self._calc_col_m
        super(DecisionTreeRegressor, self).__init__(min_sample_split=min_sample_split,
min_imp=min_imp, max_depth=max_depth,
                      imp_func=self._imp_func, leaf_node_calc=self._leaf_val_calc)


    def _calc_var_reduct(self, y, y1, y2):

        var = np.var(y)
        var_y1 = np.var(y1)
        var_y2 = np.var(y2)

        y_len = len(y)
        part_1 = len(y1) / y_len
        part_2 = len(y2) / y_len
        var_reduction = var - (part_1 * var_y1 + part_2 * var_y2)
        return  var_reduction

    def _calc_col_m(self, y):
        """
        calculate the prediction val for that leaf node using mean.
        """
        mean = np.mean(y, axis=0)
        return mean

    def train(self, X, y):
        """
        Build the tree.
        """

        super(DecisionTreeRegressor, self).train(X, y)


class RandomForest:
    def __init__(self, trees, n_trees, max_feature ,
                 prediction_aggrigation_calculation):
        self.n_estimators = n_trees
        self.max_features = max_feature
        self.tree_feat_idxes = []
        self.prediction_aggrigation_calculation = prediction_aggrigation_calculation
        # Initialize the trees.
        self.trees = trees

    def _create_randsubs(self, X, y, n_subsets, replacement=True):
        subset = []
```

```
        sample_size = (X.shape[0] if replacement else (X.shape[0] // 2))
        Xy = np.concatenate((X, y), axis=1)
        np.random.shuffle(Xy)
        for i in range(n_subsets):
            idx = np.random.choice(range(sample_size), size=np.shape(range(sample_size)),
replace=replacement)
            X = Xy[idx][:, :-1]
            y = Xy[idx][: , -1]
            subset.append({"X" : X, "y": y})
        return subset

    def train(self, X, y):
        n_features = X.shape[1]
        if self.max_features == None:
            self.max_features = int(math.sqrt(n_features))

        subsets = self._create_randsubs(X, y, self.n_estimators)

        for i, subset in enumerate(subsets):
            X_subset , y_subset = subset["X"], subset["y"]
            # bagging
            idx = np.random.choice(range(n_features), size=self.max_features, replace=True)
            # track this for prediction.
            self.tree_feat_idxes.append(idx)
            #X with the selected features
            X_subset = X_subset[:, idx]

            y_subset = np.expand_dims(y_subset, axis =1)
            # build the tree.
            self.trees[i].train(X_subset, y_subset)

    def predict(self, test_X):
        y_preds = np.empty((test_X.shape[0], self.n_estimators))
        # pred from each tree
        for i, tree in enumerate(self.trees):
            features_idx = self.tree_feat_idxes[i]
            X_selected_features = np.array(test_X[:, features_idx])
            if isinstance(tree, DecisionTreeRegressor):
                y_preds[:, i] = np.array(tree.predict(X_selected_features)).reshape((-1,))
            else:
                y_preds[:, i] = tree.predict(X_selected_features)

        y_pred = self.prediction_aggrigation_calculation(y_preds)

        return y_pred

class RandomForestRegression(RandomForest):
    def __init__(self, max_feature, max_depth, n_trees=100, min_sample_split=2, min_imp=1e-8):
```

```
        self.prediction_aggrigation_calculation = self.calc_the_mean


        self.trees = []
        for _ in range(n_trees):
            self.trees.append(DecisionTreeRegressor(min_sample_split=min_sample_split,
                                        min_imp=min_imp, max_depth=max_depth))

        super().__init__(trees=self.trees, n_trees=n_trees,max_feature=max_feature,
                    prediction_aggrigation_calculation=self.prediction_aggrigation_calculation)

    def calc_the_mean(self, y_preds):
        y_pred = np.empty((y_preds.shape[0], 1))
        for i, sample_predictions in enumerate(y_preds):
            y_pred[i] = np.mean(sample_predictions)

        return y_pred

def mean_squared_error(y,y_pred):
    sum_error = 0.0
    for i in range(len(y)):
        pred_err = y_pred[i] - y[i]
        sum_error += (pred_err ** 2)

    mean_err = sum_error / float(len(y))
    return mean_err[0]

def MAPE(y_gt,y_hat):
    mape = np.mean(np.abs((y_gt - y_hat)/y_gt))*100
    return mape

def mae_metric(y_gt,y_hat):
    sum_error = 0.0
    for i in range(len(y_gt)):
        sum_error += abs(y_hat[i] - y_gt[i])
    return sum_error / float(len(y_gt))

def r_squared(y_test,y_hat):
    y_mean = np.mean(y_test)
    ss_total = np.sum(np.power((y_test-y_mean),2))
    ss_res = np.sum(np.power((y_test-y_hat),2))

    return 1-(ss_res/ss_total)

x_train_fold = pd.read_csv(
  "C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_4hrs_train.csv",
  index_col="Timestamp")
```

```
y_train_fold = pd.read_csv(
  "C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/y_4hrs_train.csv",
  index_col="Timestamp")
y_train = np.array(y_train_fold.iloc[:, 1:])
X_train = x_train_fold[['feature_lagging_power_1', 'Blade-3 Actual Value_Angle-A',
'rolling_4_power_mean', 'tb',
                'Blade-2 Actual Value_Angle-A', 'Torque', 'Nacelle Revolution', 'Voltage A-N',
                'Tower Acceleration Normal', 'Scope CH 4', 'Hydraulic Prepressure',
                'Tower Acceleration Lateral', 'Reactive Power', 'Blade-1 Actual Value_Angle-A',
                'feature_lagging_power_24', 'feature_lagging_power_72',"power"]]

x_test_fold = pd.read_csv(
  "C:/Users/Mehmet Yigit
Turali/PycharmProjects/485project/yigit_mefe/Decision_Tree/x_4hrs_test.csv",
  index_col="Timestamp")
x_test_fold = x_test_fold.iloc[:, 2:]
X_test = x_test_fold[['feature_lagging_power_1', 'Blade-3 Actual Value_Angle-A',
'rolling_4_power_mean', 'tb',
                'Blade-2 Actual Value_Angle-A', 'Torque', 'Nacelle Revolution', 'Voltage A-N',
                'Tower Acceleration Normal', 'Scope CH 4', 'Hydraulic Prepressure',
                'Tower Acceleration Lateral', 'Reactive Power', 'Blade-1 Actual Value_Angle-A',
                'feature_lagging_power_24', 'feature_lagging_power_72',"power"]]

y_test_fold = pd.read_csv("/auto/k2/yturali/yigit_mefe/Decision_Tree/y_4hrs_test.csv",
index_col=False)
y_test_fold.target = y_test_fold.target.shift(16)
y_test_fold_r = y_test_fold.iloc[0: , 2:].to_numpy(dtype = "float32")
print(y_test_fold_r)
random_forest_reg = RandomForestRegression(n_trees=2, max_feature=10, min_sample_split=3,
max_depth=5)

random_forest_reg.train(X_train[-1000:].values, y_train[-1000:])
y_pred = random_forest_reg.predict(X_test.values)
print(y_test_fold_r[16:].shape)
print(y_pred[16:])
score = np.sqrt(mean_squared_error(y_test_fold_r[16:216],y_pred[16:216]))
print("The RMSE score of the trained model", score)
```