Predicting the Broyhill Wind Turbine

Eli Orians

Fall 2024

Contents

1 Project Overview

The goal of this project is to develop a machine learning model able to predict the power output of the Broyhill Wind Turbine using data collected from the turbine and weather forecast data. The Broyhill Turbine was built in 2009, funded by Appalachian State Renewable Energy Imitative (ASUREI) with help from New River Light and Power (NRLP). ASUREI is a student club that receives five dollars from every students tuition in order to reduce the environmental impact of Appalachian State University by implementing energy technologies, investing in energy efficiency projects, and promoting campus engagement.

This research is an important step toward improving how we predict wind energy production, which is especially useful for larger wind farms. While the small size of the Broyhill Wind Turbine limits the direct applicability of these predictions, scaling this research to larger turbine farms could have a significant impact. Accurate forecasts for larger operations help wind farm operators plan better—whether that's deciding when to store energy, managing how much electricity goes to the grid, or ensuring everything runs as efficiently as possible. It also means relying less on backup energy sources like fossil fuels and maintaining a more stable grid. By leveraging machine learning, this project lays the groundwork for improving prediction accuracy, which would be especially valuable for larger-scale applications, supporting the broader adoption of wind energy and reducing environmental impacts.

2 Data Collection

2.1 Turbine Data Collection

The data for the Broyhill Wind Turbine has been continuously recorded since its commissioning in 2009 and is maintained in the manufacturer's database. While ASUREI has its own database containing turbine information, it is quite limited, holding only timestamps and power output data from 2018 onward, with significant gaps in coverage. To access the more comprehensive manufacturer's database, I collaborated with Jim Dees, one of ASUREI's advisors and a faculty member in the Department of Sustainability, to establish contact with the manufacturer, Power Grid Partners.

After a series of discussions, we devised a practical solution to ensure continuous access to the data. A copy of the manufacturer's database was placed on the turbine's administrative computer, where new data is automatically appended. Periodically, Jim Dees and I retrieve the updated database by transferring its contents to a flash drive for use in this project.

The turbine records data at 15-minute intervals, with timestamps standardized to Coordinated Universal Time (UTC). The dataset contains nearly 200 columns, capturing a wide range of operational and environmental variables. Key columns include:

- Inverter power output (measured in kW) - the target metric for this project.
- Weather measurements such as wind speed, direction, and temperature, collected at the turbine.
- Turbine state indicators - operational data reflecting the turbine's status, such as error codes and system performance.

This comprehensive dataset forms the backbone of this research, providing the detailed information needed to develop a predictive model for the turbine's power output.

2.2 Forecast Data Collection

The forecast data paired with the turbine data was sourced from the National Oceanic and Atmospheric Administration (NOAA). Since historical weather forecasts were unavailable, I developed a

Python script to continuously collect forecast data in real time. This script runs on my personal desktop, utilizing NOAA's API to retrieve weather forecasts for the coordinates of the Broyhill Wind Turbine. However, the forecast data is generalized for the Boone area which you will later see caused some issues.

The script is designed to run hourly, retrieving forecasts for every hour over the next seven days. Key features of the forecast data include:

- Timestamp recorded in Eastern Standard Time (EST) to correspond with local time.
- Wind speed measured in miles per hour (mph).
- Wind direction
- Temperature
- Chance of precipitation

The collected data is stored in JSON files, with each file named according to the time it was retrieved (in EST). This process has generated thousands of JSON files, forming a robust repository of forecast data to support this project's analysis.

3 Data Preprocessing

3.1 Turbine Data Preprocessing

Preprocessing the turbine data was essential to ensure consistency and compatibility for subsequent machine learning tasks. The process began by converting the SQL dump into a pandas DataFrame, which served as the primary tool for data operations. To preserve the integrity of the original data and facilitate debugging, intermediate CSV files were saved throughout the preprocessing steps. While I initially considered using a database for this task, I ultimately determined that the volume of data did not require the additional complexity. The major data cleaning steps are as follows:

The dataset had missing values in various columns, likely due to data collection interruptions or sensor malfunctions. To prevent these gaps from introducing errors, I decided to drop any rows with null values. I also removed columns that were irrelevant to predicting power output, such as those related to administration or maintenance, to reduce unnecessary noise and improve computational efficiency.

The turbine data was originally collected every 15 minutes, but the forecast data from NOAA was provided hourly. To make the two datasets align, we aggregated the turbine data into hourly intervals. For numerical columns like wind speed and power output, I took the average of the four 15-minute readings within each hour. For categorical columns, such as the turbine's operational state, I kept the most recent value recorded within the hour to reflect the latest operational condition.

To ensure we were only analyzing relevant data, I excluded any periods where the turbine wasn't actively generating power, such as during maintenance, idling, or downtime. By focusing on periods when the turbine was operational, I ensured that the dataset used for training the model was accurate and reflective of real-world turbine performance.

I also experimented with removing values where the power output was negative. This is possible when the power output is low, and the devices within the turbine still need to be powered so it begins to use power rather than produce. However, this negatively impacted the model performance.

Turbine State Enumeration	
State	Integer Value
Init	0
Off	1
Stopped	2
System Test	3
Wait	4
Motor	5
Standby	6
Active	7
Decelerate	8
Service	9

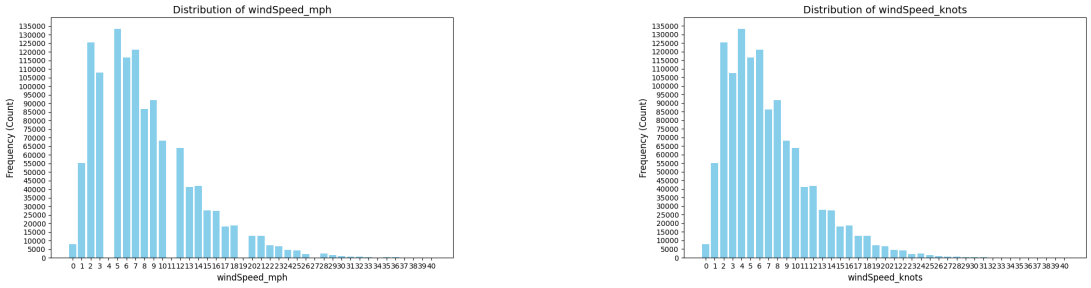
Figure 1: This chart can be referenced to see the possible turbine states, I kept only the 'Active' state.

3.2 Forecast Data Preprocessing

The forecast data required several adjustments to ensure it was suitable for analysis. Each JSON file was opened and converted to a dataframe for simplified cleaning. Each file was ultimately saved as a new CSV file. This simplified later steps and enabled seamless integration with the turbine data. It also allowed the original collected data to remain unchanged encase the process were to be updated.

The wind direction, originally recorded as cardinal directions (e.g., N, NE), was converted into two numerical columns representing the x and y components of the wind vector. This transformation made the data more suitable for machine learning algorithms, as numerical representations capture direction more effectively. For example, North would become $(x, y) = (0, 1)$.

Later in the project, I realized that there were gaps in the distribution of the wind speed (mph). After reaching out to NOAA, I discovered this was likely due to how they predict the initial wind speed in knots but convert it to mph and there are gaps when converting and rounding. For example, 3 knots converts to 3 mph but 4 knots converts to 5 mph. There were several of these cases, and converting the wind speed back to knots leveled out this distribution.



(a) Wind Speed in mph

(b) Wind Speed in knots

Figure 2: Comparison of the distribution of wind speed in mph and knots.

3.3 Combining Turbine and Forecast Data

The final step in the preprocessing pipeline was combining the turbine data with the forecast data. This alignment ensured that the turbine yield was matched with the corresponding forecast made a specified number of hours earlier, depending on the prediction horizon. To achieve this, we merged the datasets based on the timestamps, ensuring that the turbine data for a specific hour was paired with the forecast data made in the hour(s) prior.

For example, if we are forecasting 1 hour ahead and the current turbine row corresponds to 5:00 PM, the following steps were performed:

1. Identify the forecast made at 4:00 PM the same day, which was the forecast generated 1 hour before the current turbine data point.

2. Merge the row for 5:00 PM from the turbine dataset with the forecast data from 4:00 PM.

This process was repeated for each row of turbine data, aligning it with the relevant forecast data based on the desired forecast horizon. Additionally, to provide more context for the prediction, multiple forecasts from the previous hour were included in the merge, allowing the model to take into account any potential patterns in the forecast data leading up to the prediction time.

It's important to note that the combination of these datasets was limited to the period for which we had both turbine and forecast data, which resulted in using approximately six months worth of data (a little over 3000 data points after cleaning). This step was crucial for ensuring that the input data for the machine learning model was both relevant and aligned in time, enabling more accurate predictions.

4 Machine Learning

Thee final model selection process involved several iterations of training and validation, ultimately settling on the use of Nested Cross Validation.

Outer Loop: The outer loop is responsible for dividing the data into multiple splits, creating independent training and test sets for each iteration. Each split is used to train a model, while the test data is reserved for validation resulting in several models. The average of each models performance is used for validation. This ensures that each model is evaluated on unseen data, providing a more accurate estimate of its performance on new, real-world data.

Inner Loop: Within each iteration of the outer loop, the inner loop performs an exhaustive grid search over potential hyperparameters and feature selection methods. This allows us to identify the best combination of features and hyperparameters for each training dataset. The test set from the outer loop remains separate, ensuring that the hyperparameter tuning and feature selection do not influence the final validation results.

Secondary Loop after Validation: A final model is trained using the entire dataset (after validation) using a secondary loop that follows the same process as the inner loop, without any extra splits. This model, with the optimal hyperparameters and features, is then used for making predictions on new data and creating visualizations. It also outputs the selected features and hyper paramaeters, since I was unable to show these for each model in the nested cross validation loop.

Nested cross-validation is preferred for validation because it provides a more reliable estimate of a model's performance by effectively separating the process of model selection and evaluation. The outer loop ensures that the test data remains unseen during training, while the inner loop optimizes hyperparameters and selects features without contaminating the validation process. This reduces the risk of overfitting and provides a more robust assessment of the model's generalization ability. In contrast, a single validation set might lead to overly optimistic or biased performance estimates, as the same data could be used for both model optimization and evaluation.

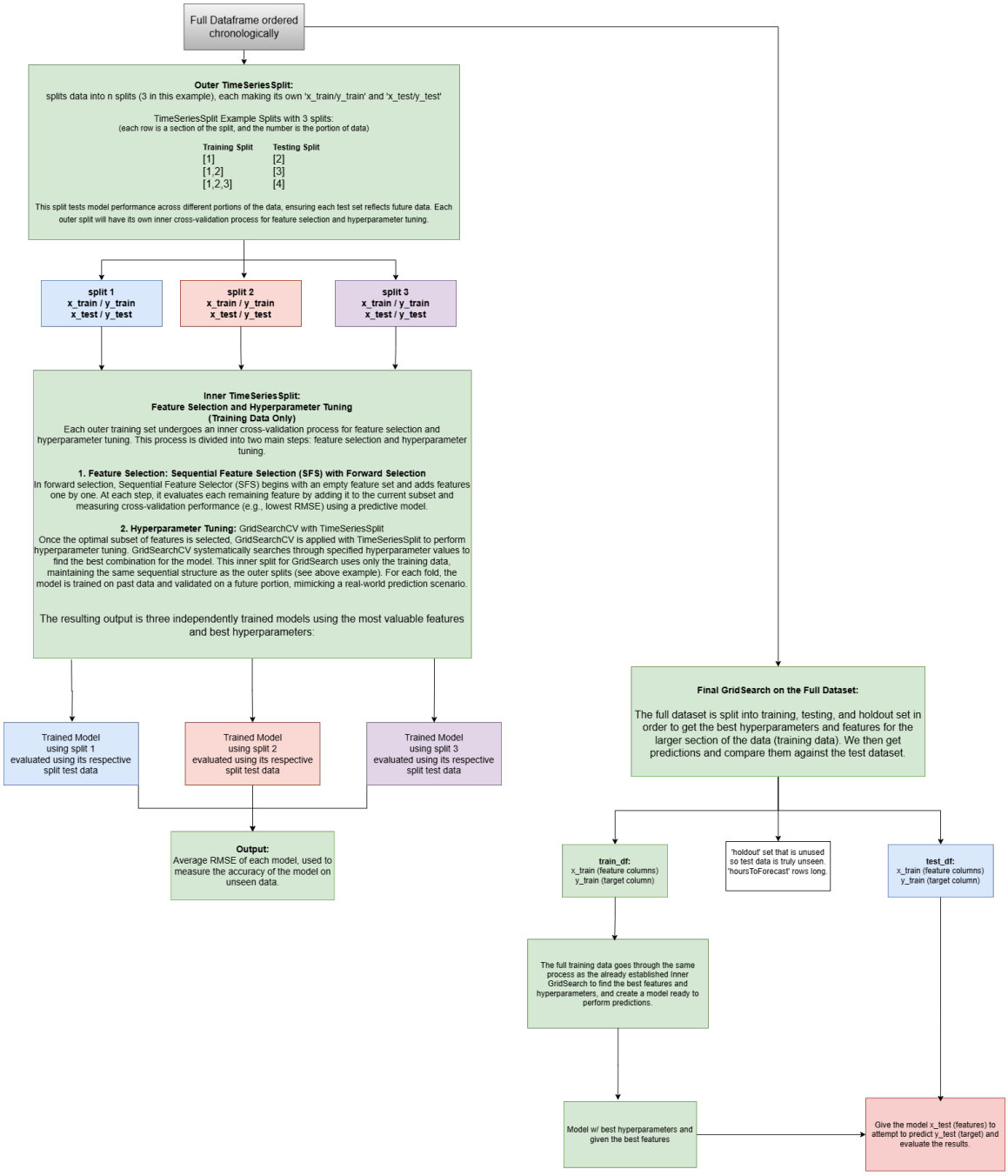


Figure 3: The machine learning process.

4.1 Time Series Split

For time series data, it is crucial to preserve the chronological order of observations, as future data points should not influence the model during training. To address this, I used a cross-validation method specifically designed for time series data, known as Time Series Split. This approach ensures that the training set always consists of data from the past, while the validation set is composed

of more recent data, aligning with the real-world scenario where past observations inform future predictions.

Time series split was applied at all stages of the cross-validation process, including the outer loop, inner grid search (feature selection & hyperparameter tuning), and the secondary loop. This preserved the integrity of the time-based dependencies and prevented any data leakage between training and validation sets.

For example, consider a dataset with 10 data points and 3 splits:

- **Split 1:** Train on [1, 2, 3], validate on [4, 5]
- **Split 2:** Train on [1, 2, 3, 4, 5], validate on [6, 7]
- **Split 3:** Train on [1, 2, 3, 4, 5, 6, 7], validate on [8, 9, 10]

This method allows for more realistic model training and validation, ensuring that the model's performance is assessed based on its ability to predict future values without "peeking" at future data during training. It is particularly useful in time series forecasting and predictive modeling, where temporal dependencies are key to accurate predictions.

4.2 Feature Selection

Feature selection plays a key role in enhancing model performance by identifying the most relevant features, reducing overfitting, and improving computational efficiency. It occurs first in the inner loop of the cross-validation process, where the goal is to identify the optimal subset of features for the model. Two main feature selection methods used are Sequential Feature Selection (SFS) and SelectKBest, each with its own advantages depending on the dataset.

Sequential Feature Selection (SFS): Sequential Feature Selection (SFS) works by initially evaluating each feature individually to determine the best one based on a performance metric. Once the best feature is selected, additional features are added one by one, testing each combination to check if the performance continues to improve. SFS can capture interactions between features as it evaluates combinations of features in each step. SFS offers two options, forward or backward selection. Forward selection starts with an empty set of features and iteratively adds the most significant features, one by one, based on the performance improvement. Backward elimination starts with all features and removes the least significant ones, one by one, to see if performance improves. This method is useful when dealing with a large number of features, as it can eliminate irrelevant features step by step.

SelectKBest: SelectKBest ranks all features independently based on their relevance to the target variable, then selects the top k features based on this ranking. This method is particularly efficient when dealing with datasets containing a large number of features. While it does not account for feature interactions, SelectKBest offers a straightforward way to reduce dimensionality by selecting only the most important features. To choose k, a created a parameter grid with several options so the model could choose the optimal number of features.

I typically chose to use SFS with forward selection. This is because I discovered that it would typically only use the forecasts for the current hour (rather than including forecasts from previous hours) so using a large number of features wasn't necessary.

4.3 Hyperparameter Tuning

Hyperparameter tuning is an essential step in the model optimization process, occurring second in the inner loop of the cross-validation. The goal of hyperparameter tuning is to systematically test different combinations of hyperparameters to identify the best settings that maximize model

performance. This is achieved through the use of cross-validation, which helps ensure that the selected hyperparameters generalize well to unseen data.

During hyperparameter tuning, a grid search approach is employed. A parameter grid is defined, containing the hyperparameters and their possible values that we wish to test. The model is then trained and evaluated using cross-validation for each combination of hyperparameters in the grid. This process ensures that each combination is evaluated independently on multiple data splits, reducing the risk of overfitting to a particular set of data.

The performance of each hyperparameter combination is assessed using mean squared error. The best performing set of hyperparameters is selected based on the evaluation results, which are averaged across the cross-validation folds. This helps in selecting hyperparameters that not only fit the training data but also provide strong performance on validation data, ensuring better generalization.

4.4 Validation

After selecting the best features and hyperparameters in the inner loop, the outer loop is responsible for validating each model on previously unseen data. This validation step ensures that the model's performance is tested on data it has not seen during training, providing a more realistic estimate of how the model will perform on new, unseen data.

Outer Loop Validation: In the outer loop, the data is split into training and test sets. The model is trained on the training set and evaluated on the test set. The process is repeated across multiple data splits, with the model being trained and validated on different portions of the dataset each time. This approach allows us to compute the average of the model's evaluation metrics across all splits. The final result provides a more reliable and unbiased estimate of the model's performance, as the evaluation is based on data that the model has not been exposed to during training.

Secondary Loop Validation: In contrast to the nested cross-validation approach, the secondary loop involves validating the model using a test dataset after the gridsearch has selected the best features and hyperparameters. While this method provides a performance estimate, it is less robust than nested cross-validation. The performance metrics derived from the secondary loop tend to be more optimistic, as the model has already been exposed to the training data and might exhibit some overfitting. Therefore, the results from the secondary loop should be interpreted with caution and are not as reliable as those obtained from the nested cross-validation process.

By using the outer loop in nested cross-validation, we ensure a more thorough and unbiased evaluation of the model's performance, providing stronger confidence in its ability to generalize to unseen data.

4.5 Evaluation Metrics

To assess the performance of the models, two key evaluation metrics were used: Root Mean Square Error (RMSE) and R^2. These metrics provide insight into the accuracy and explanatory power of the model, helping to determine how well it predicts and fits the data.

Root Mean Square Error (RMSE): RMSE measures the average deviation of the model's predictions from the actual values. It is calculated as the square root of the mean of the squared differences between predicted and actual values. RMSE is expressed in the same units as the target variable, making it intuitive to interpret. A lower RMSE indicates better model performance, with values closer to zero suggesting that the model's predictions are very close to the actual values.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

Where:

- $\hat{y}_i$: Predicted value for observation i
- y_i: Actual value for observation i
- n: Number of observations

R^2 (Coefficient of Determination): R^2 quantifies how well the model explains the variability in the target variable. It is a value between 0 and 1, where values closer to 1 indicate that the model accounts for a larger proportion of the variability in the data. An R^2 value of 1 means the model perfectly fits the data, while a value of 0 indicates that the model does not explain any of the variability.

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

Where:

- $\bar{y}$: Mean of the actual values
- Other terms are as defined above

By using both RMSE and R^2, the evaluation captures a comprehensive view of the model's performance, balancing the accuracy of predictions with its ability to explain the underlying data.

5 Results

5.1 Model Results

A detailed list of model results can be found here.

To establish a baseline for model performance, I evaluated a simple model that predicted the average power output for all data points. This baseline yielded an RMSE of 11.44 and an R^2 of -0.67. Most of the model evaluation focused on predictions made six hours in advance, providing a consistent benchmark for comparing performance across different models. If a model showed promise at six hours, I tested it at other prediction intervals.

Below is a table depicting several models that I tested with and their selected features. You can see how typically the wind speed and wind direction were the selected features. The _0 following these features indicates that they were the forecast for that hour, if it were to have a _1 following it would indicate that it was the forecast from 1 hour before said time. Typically, the feature selection would not use the forecasts from beforehand so in later tests I would exclude them to speed up computation time.

Model	Hours Out	Avg RMSE	Avg R^2	Features Selected
Polynomial Regression w/ Scaling	6	9.37	0.50	'windSpeed_knots_0', 'windDirection_x_0', 'windDirection_y_0'
SVR	6	9.55	0.47	'windSpeed_knots_0', 'windDirection_x_0', 'windDirection_y_0'
Linear Regression	6	9.81	0.44	'windSpeed_knots_0', 'windDirection_x_0', 'windDirection_y_0'
RidgeCV	6	9.82	0.44	'windSpeed_knots_0', 'windDirection_y_0', 'temperature_F_0'
LassoCV	6	10.01	0.43	'windSpeed_knots_0', 'windDirection_y_0', 'temperature_F_0'
Baseline	-	11.44	-0.67	-
Bagging	6	12.60	0.14	'windSpeed_knots_0', 'windDirection_x_0', 'windDirection_y_0'

Figure 4: Model Performance of several models focusing on 6 hours out

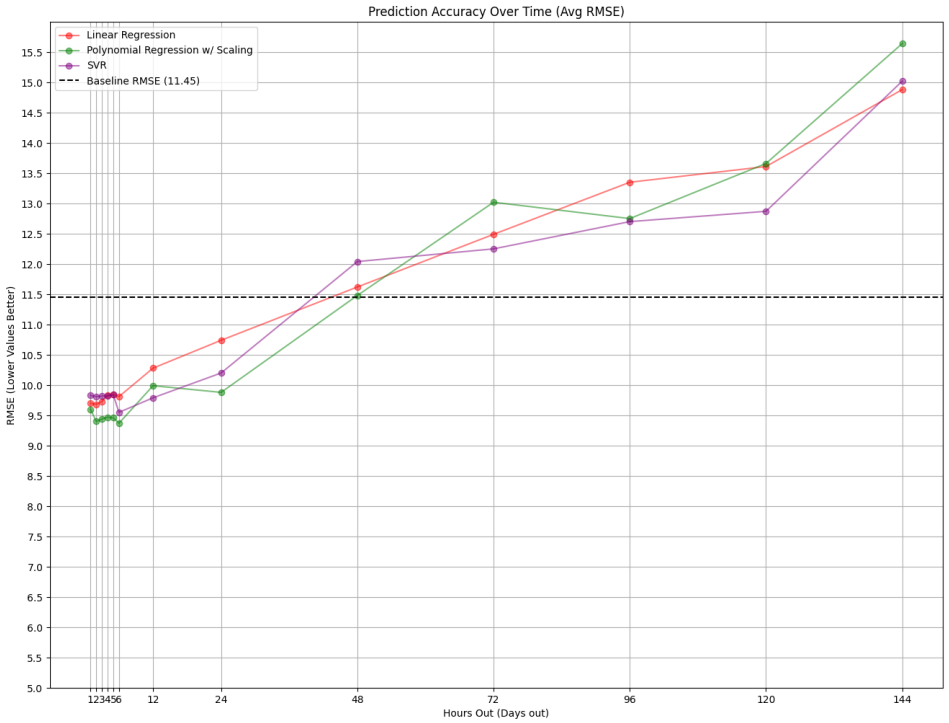


Figure 5: Model Performance of several models showing the performance when predicting different hours out.

As you can see from Figure 4, my best model was using Polynomial Regression. In the end, I also included scaling in the polynomial pipeline, since otherwise it would fail to converge when predicting more than 12 hours out. You can also see in Figure 5 that when predicting more than 48 hours in the future, most models fail to perform better than the baseline.

It is difficult to pin the exact RMSE that would be needed to call the model successful, but we can look at the range and average of the target metric to make the decision. The power output ranges from -.25 to 80.5 kW, with an average of 11.63. Because most of the target points fall around 11.63, and Polynomial Regression is on average 9.37 points off of the metric (the models RMSE), this is too significant of a difference to be accurate. We can also use the following scatter plot and line plot to see how accurate a model is.

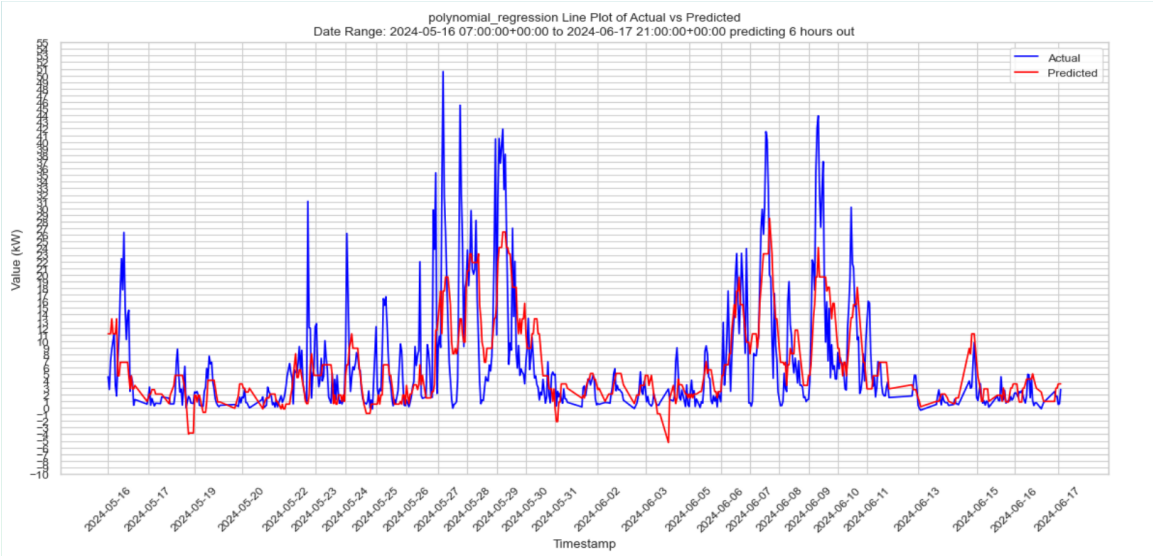


Figure 6: Line plot for Polynomial Regression predicting 6 hours out.

Figure 7: Scatter plot for Polynomial Regression predicting 6 hours out.

The scatter plot in Figure 7 highlights the model's inaccuracy, with points exhibiting significant variation from the ideal linear relationship.

5.2 Why is it inaccurate?

The next question to follow, is why is it inaccurate? One telling graph is when comparing the actual wind speed measured at the turbine with the foretasted wind speed:

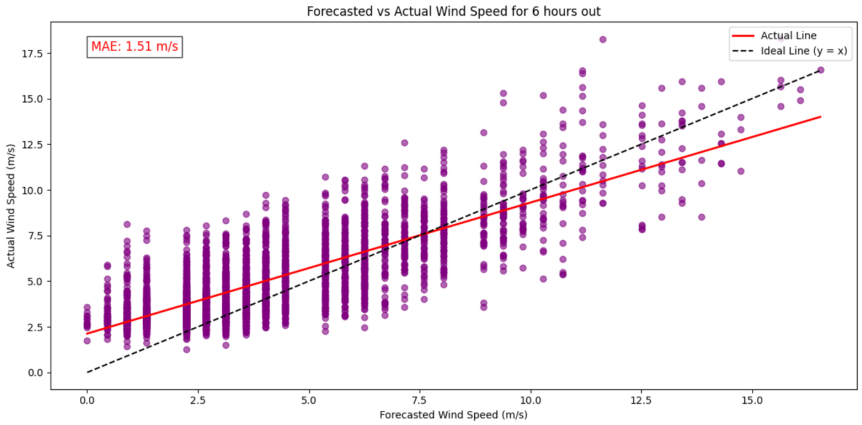


Figure 8: Scatter plot comparing actual wind speed against forecasted wind speed.

As you can see, there is a far too much variation to call the forecasts accurate. I tested a model trained on the actual wind speed collected at the turbine to see how accurate it would be if we had a more accurate forecast data:

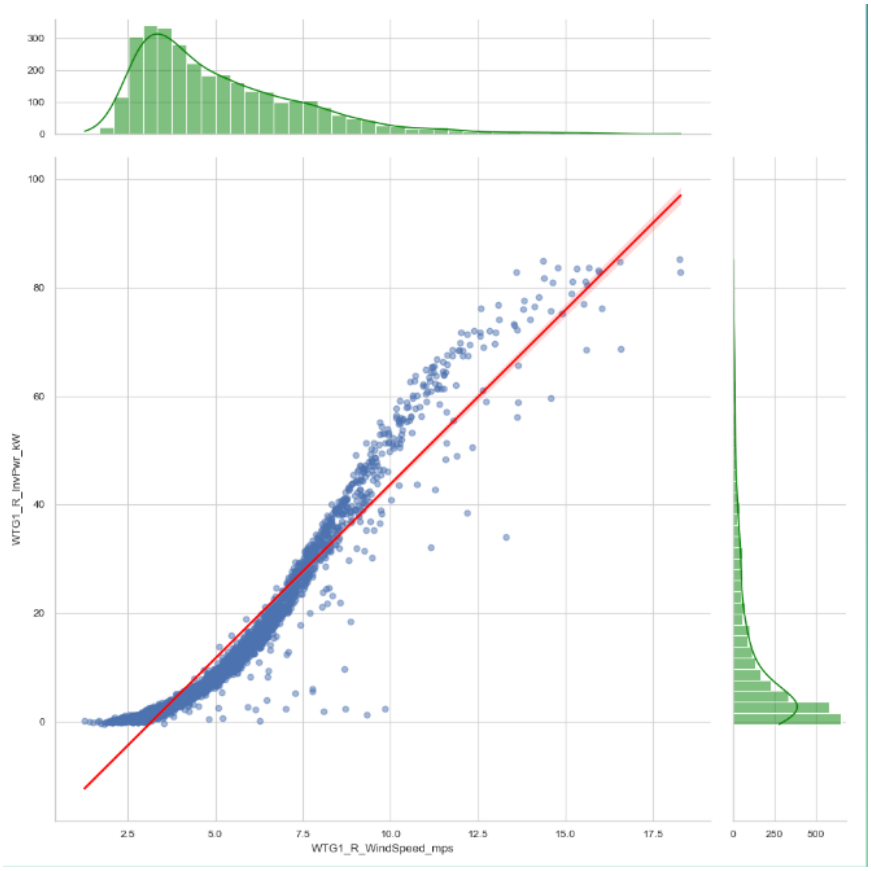


Figure 9: Correlation between actual wind speed and power output.

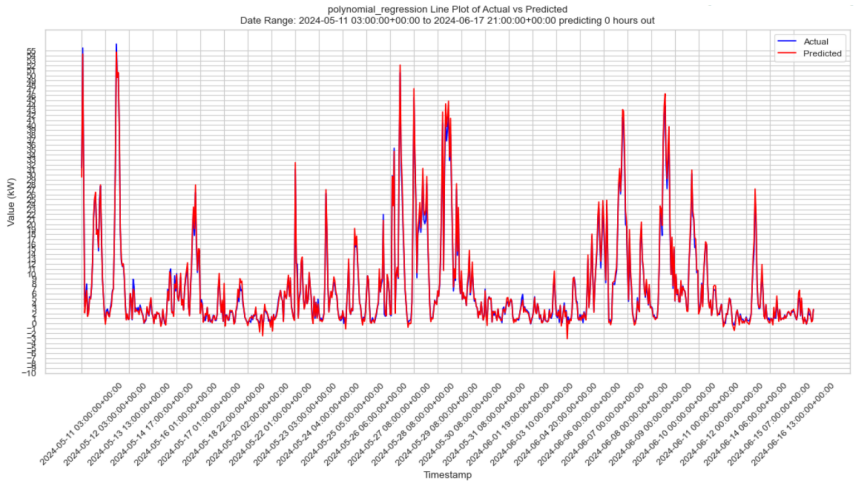


Figure 10: Line plot for Polynomial Regression trained using actual wind speed.

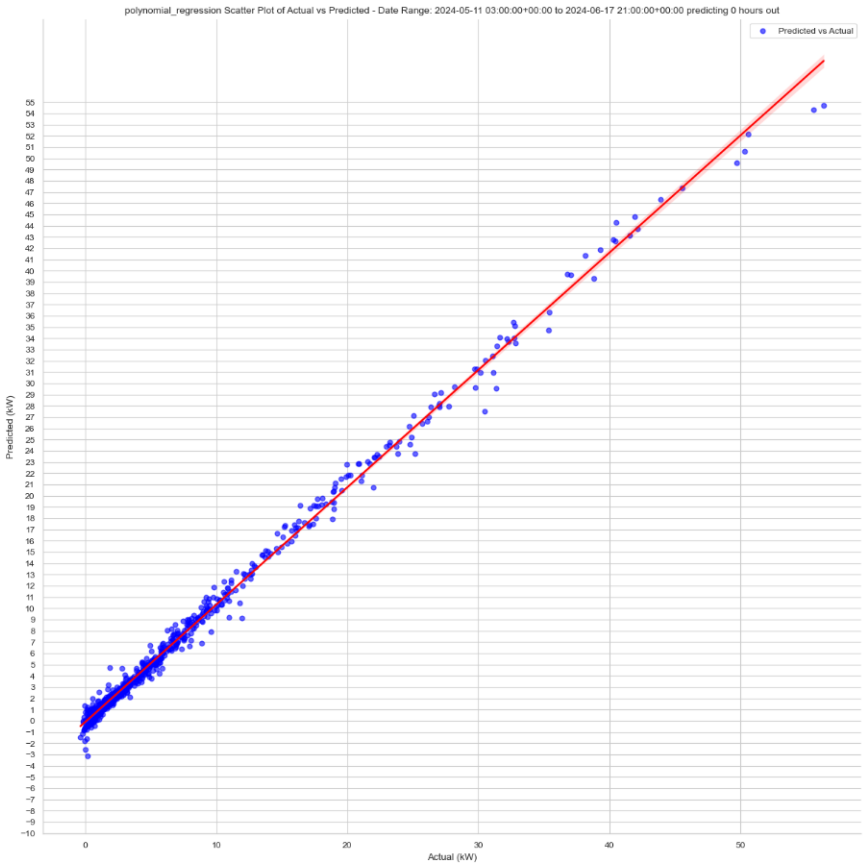


Figure 11: Scatter plot for Polynomial Regression trained using actual wind speed.

Polynomial regression had an accuracy of 2.93 average RMSE and had an R^2 of 0.95 when trained on the actual wind speed collected at the turbine. While these are great results there is no way to predict future power output since there is no future data with these, it simply showcases the best case scenario.

6 Conclusion & Future Work

Current weather forecasts aren't precise enough to reliably predict turbine power output. The problem lies in the significant variability in forecasted wind speeds, especially in complex mountain microclimates, where generalized forecasts struggle to capture local conditions. While models like Polynomial Regression show promise, their accuracy is limited by the quality of the input data.

To tackle this issue, we need more accurate local weather data. A practical solution could be deploying weather stations near turbine sites, such as mountain peaks, to monitor conditions directly. With better data feeding into the models, we can reduce errors and make predictions more reliable. Ultimately, improving weather forecasts and model inputs could unlock the full potential of predictive analytics for wind energy, especially in tricky regions like mountainous areas where conditions are hard to predict.

Special thanks to Dr. Parry for his guidance and mentorship throughout this project. I would also like to acknowledge the App State Renewable Energy Initiative and Jim Dees from the Department of Sustainability for their help in obtaining data and their valuable insights in renewable energy systems.

7 Project Links

- GitHub Repository
- Google Drive Folder
- ASUREI Wind Turbine Output Website