Department of Engineering

Study program:

Renewable Energy Systems

Submitted by:

Sai Jithendra Kumar Gangireddy

Master’s thesis on:

Estimation of wind conditions in a wind park by using machine learning

Master thesis for obtaining the academic degree:

Master of Engineering (M. Eng.)

Submission date: 24.04.2023

[redacted]

First reviewer: Dr.-Ing. Birgit Lustermann
University of Applied Sciences Nordhausen

Second reviewer: Dr. rer. nat. Alexander Basse
University of Kassel

Abstract

In the wind industry, optimizing the power output of the wind park (or single turbines) in different atmospheric conditions is the key factor to achieve better performance and therefore, energy yield. For that purpose there is a need to monitor the performance of wind turbines continuously by detecting the abnormal behavior of the wind turbines at different wind conditions.

The use of machine learning in predicting the different wind conditions at different locations enables to reduce the measuring costs and maintenance costs. In this work, different machine learning models (linear regression, random forest, support vector regressor, and neural network) were implemented on different types of data (SCADA, met mast and weather model data) in order to predict the wind speeds at different locations in a wind park.

The thesis project will focus on the best possible use of machine learning models in predicting the wind conditions in a wind park and on comparing the different model results.

Acknowledgement

I would like to thank my friends, family, colleagues and Fraunhofer IEE for their support in this process. Dr. rer. nat. Alexander Basse and Dr.-Ing Birgit Lustermann helped to guide me in this path very patiently and encouraged me to strive for excellence. I count myself fortunate to have had the excellent guidance of Dr. rer. nat. Alexander and Dr.-Ing Birgit Lustermann, who never seemed to tired of my questions, no matter how absurd, and demystified this process for me. My colleagues of University Kassel and Fraunhofer IEE employees have been nothing but supportive. I could not have written this without my guides and my colleagues, who listened me patiently, provided well times distractions, and helped me with my thesis related thoughts.

Table of Contents

Abbreviations

RMSE	--	Root Mean Squared Error
MSE	--	Mean Squared Error
MAE	--	Mean Absolute Error
NN	--	Neural Network
SCADA	--	Supervisory Control and Data Acquisition
ICON	--	Icosahedral Nonhydrostatic

List of figures

C. MAE of models

List of tables

Chapter 3

Appendix

A. Error scores for different models

1. Introduction

1.1 Overview

Wind energy is a clean and renewable alternative for the production of electric energy, presenting the great social acceptance. Wind is the result of the displacement of air masses, caused by the effects of atmospheric pressure differences between two distinct regions and influenced by natural effects such as sea level, latitude, altitude and soil roughness among others [1].

Wind generation occurs through the contact of the wind with the blades of the wind device. When rotating, these blades convert wind speed into mechanical energy that drives the rotor of the wind generator, which produces electricity [1]. According to [2], the amount of energy generated by winds is a function of their speed and mass.

$$p = \frac{1}{2}\rho A V^3 \tag{1. 1}$$

Where ρ is density of air

A is swept area of turbine

V is velocity of wind

The above power equation (1. 1) shows that the power is directly proportional to the velocity to the power of three (V^3) Therefore, it is very important to make a good wind speed prediction. The wind turbines cannot utilize all the available power in the wind. In the wind industry, optimizing the power output of a wind park (or single turbines) in different atmospheric conditions is the key factor to achieve better performance and therefore, energy yield.

In order to achieve the maximum power output from the wind park, it is mandatory to monitor the performances of the wind turbines continuously, e.g., detecting the abnormal behavior of the wind turbines at different wind conditions in advance will help to fix the issue quickly and the downtime of the turbine can be minimized as a result there will be more electricity production.

There are two types of monitoring techniques involved for a wind turbine to run efficiently and safely. They are,

- Conditional monitoring (CM)
- Structural health monitoring (SHM)

Conditional monitoring:

It involves observing the mechanical components of a wind turbine to identify changes in operation that indicate developing a fault. The use of machine learning in detecting the abnormal behavior of the turbine components will be beneficial [3].

Machine learning techniques and artificial intelligence belong to the statistical approach. The model described in this project is usually a non-linear model. Many artificial intelligence methods are better than conventional methods and have a good development perspective [3]. Machine learning models use the previous history of wind data to perform a forecast over the next few minutes or a few hours. These models are easy to use but take computational effort, depending on the algorithm. The types of machine learning models used are explained in detail in the following chapters.

1.2 Task

The use of machine learning in predicting the wind conditions at different locations enables to reduce the measuring costs and maintenance costs. In this project, different machine learning models are implemented on different types of data (SCADA, met mast and weather model data) in order to estimate the wind speeds at different locations in a wind park as shown in Figure 1. 1.

The turbines are marked from 1-12 at different locations, the approach for predicting the wind speeds by using machine learning at a turbine are by training the models using the different data/cases as described below along with turbine wind speed data.

Case-1:

In case 1, only met mast data which is real time data measured in a park is considered while training the different machine learning models.

Case-2:

In case 2, ICON-D2 data which are weather model data (forecasted data) are considered while training the different machine learning models.

Case-3:

In case 3, a combination of met mast and ICON-D2 data are considered while training the different machine learning models.

Figure 1. 1: wind turbines in a wind park

This project will focus on the best possible use of machine learning models in estimating the wind conditions in a wind park. Model performance is evaluated by using different metrics (i.e., RMSE, change in bias and variance) scores and so target is identifying the models with high accuracy for the respective use cases.

2. Theoretical background

2.1 Wind conditions within a wind park

Wind

The wind is moving air caused by differences in air pressure within our atmosphere. The air tends to move from high pressure to low pressure [4]. The higher the pressure difference, the greater the airflow and, hence, higher wind speeds generally occur.

Wakes and Turbulence

Wind turbines extract energy from the wind. As a consequence, the wind is reduced for downstream of a turbine. This effect is called a wake. As the flow proceeds downstream, there is a spreading of the wake, and the wake recovers towards free stream conditions. This wake effect may influence the behavior of the other turbines that are downstream of the turbine. The wake interference effects, e.g., will highly decrease the power output and increase the fatigue loads of wind turbines that are sited in the wake region.

Influence of terrain and atmospheric stability on the wind conditions

Wind over complex terrain is influenced by changes in surface properties (such as roughness and land use) and the height elevation of the site above sea level (such as ridges, hills, and mountains). *Topography* is the term used to address the whole variation in surface properties and elevation. Orography is the term to address especially height elevation. [5]

The complexity of hilly and mountainous terrain does not allow for a straightforward application of the wind profile laws such as logarithmic law and power law [5].

In general, numerical or analytical flow models will be used to assess wind and turbulence conditions at a given site. Three-dimensional numerical wind field models can roughly be classified into three classes. The simplest ones are mass consistent flow models, which generate a divergence-free flow over orography from given measurements. They do not involve dynamic equations such as hydrostatic equations [5]. For reliable solutions, they need a more significant number of observations.

The next class is hydrostatic flow models, which solve the dynamic equations substituted by the hydrostatic (2. 1). They only work for larger scales of a few kilometers or more.

$$\frac{\partial p}{\partial z} = -g\rho = -\frac{gp}{RT} \tag{2. 1}$$

For smaller scales, full non-hydrostatic models have to be used.

2.2 Wind data generation

In the following sections, how the data are being generated and techniques used to measure are discussed in detail.

2.2.1 Met mast

A met mast is also called a meteorological tower or meteorological mast. It is a standing tower that carries meteorological instruments to measure wind speed, wind direction, and other meteorological parameters (e.g., temperature, relative humidity, air pressure, solar radiation) [6]. Met masts are the most commonly used measurement instrumentation for the planning and operation phase of wind farms. Typically, met masts have the following instruments attached:

- Wind vanes
- temperature sensors
- pressure sensors
- cup and /or ultrasonic anemometers
- humidity sensors and rain gauges

As shown in Figure 2.1, met mast consists of the above-mentioned sensors, which are placed at different heights on a tower. These instruments measure different atmospheric properties of wind continuously for every second, and this whole data are grouped differently based on different statistics (maximum value, minimum value, average value, standard deviation) for every 10 minutes.

Figure 2.1: Met mast tower [6]

2.2.2 SCADA data

SCADA (Supervisory Control and Data Acquisition) data is a type of data which contains all the turbine data i.e., wind speed, wind direction, power and so on for each individual turbine. These data are mainly useful in performance monitoring (conditional monitoring) of the wind turbine or park.

2.2.3 Weather model data ICON-D2

Weather model data are a type of data which use mathematical algorithms as well as measured data of the atmosphere to calculate or predict the weather conditions for certain time stamps. A number of global and regional forecast models are run in different countries worldwide. In this project, the ICON-D2 forecast model data are used along with the met mast data which is why this type of weather model data is briefly described here.

ICON-D2 data are created by the German Weather Service (German: DWD, Deutscher Wetterdienst), covering Germany, Austria, Switzerland and other neighbouring countries. It is both a regional and global weather forecast model [7]. Due to the high resolution (0.02°) of this model, it is considered to be more accurate than most global models. Due to the fine mesh size, the ICON-D2 especially provides for improved forecasts of hazardous weather conditions [7].

The DWD's ICON-D2 model is a forecast model which is operated for very short-range up to +27 hours. The regional ensemble forecasting system based on the DWD's numerical weather forecast model [8]. Based on the model runs at 00, 06, 09, 12, 15, 18 and 21 UTC, ICON-D2 provides new 27-hour forecast for every 3 hours [8]. It has a horizontal resolution of 2.2 km, in vertical the model defines 65 atmospheric levels.

The atmospheric parameters in ICON-D2 used for this project are wind speeds, wind direction, temperature, specific humidity at 10, 38, 78, 127, 184 m, diffusive flux, radiation, pressure, roughness and precipitation.

2.3 Machine learning

Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed. Machine learning often used for problems for which existing solutions require a lot of fine-tuning. Also, in case of complex problems for which using a traditional approach yields no good solution, the machine learning techniques can perhaps find a solution.

2.3.1 Different types of learning systems

Machine learning systems can be classified according to the amount and type of supervision they get during training. The different types of machine learning are:

- Supervised learning
- Unsupervised learning
- Semisupervised learning
- Reinforcement learning

These will shortly be described in the following sections.

Supervised learning

The supervised learning is defined as a model which is trained by using the labelled datasets. As an input the data is fed into the model with a specified labelled, the algorithm try to predict the best possible outcome. Label is the output whereas feature is the input [9].

The modules scikit-learn or keras tensorflow in the programming language python are widely used in developing the machine learning models. Any algorithm will solve either a regression problem or a classification problem.

Regression

Regression is a type of machine learning technique which uses to predict continuous (say y) outcomes based on the value of one or more predictor variables or independent data (say x1, x2, ...) and it is performed mainly to determine the correlations (relationship) between two or more variables

Classification

Classification is a type of machine learning technique which uses to predict the result as ‘Yes’ or ‘No’, ‘True’ or ‘False’, or may be with definite number of labels.

One real world example of the classification is a spam filter marking emails as ‘spam’ or ‘not spam’.

Unsupervised learning

Unsupervised learning is a type of machine learning where the model is not given any labeled data, only input data and the training set or the data is unlabeled (i.e., the data does not have any labels when training the model). The goal of unsupervised learning is to discover hidden patterns or structure in the data. It is used to uncover underlying relationships in the data, such as grouping similar examples together or identifying important features of the data. It can also be used to explore and understand a dataset, providing insights that can guide the development of supervised models. The system tries to learn on its own by observing specific patterns in the data [9].

Some of the most important and commonly used unsupervised learning algorithms are:

- Clustering
    - Hierarchical Cluster Analysis (HCA)
    - K-Means
- Dimensionality reduction and visualization
    - Principal Component Analysis (PCA)
    - Kernel PCA
    - Locally Linear Embedding (LLE)
- Association rule learning
    - Apriori
    - Eclat

Semisupervised Learning

Since labeling data is usually time-consuming and costly, you will often have plenty of unlabeled instances and few labeled instances. Some algorithms can deal with data that is partially labeled. This is called semi supervised learning [9].

Some photo hosting services are good examples of this. Most semisupervised algorithms are combinations of unsupervised and supervised algorithms. For example, deep belief networks (DBNs) are based on unsupervised components called restricted Boltzmann machines (RBMs) stacked on top of one another. RBMs are trained sequentially in an unsupervised manner, and then the whole system is fine-tuned using supervised learning techniques [9].

Reinforcement Learning

Reinforcement learning is a type of machine learning where the learning system (also called an agent) can observe the environment, select and perform actions, and get rewards or penalties in return. Many robots implement reinforcement learning algorithms to learn how to walk and to make hand movements.

2.3.2 Models

The type of machine learning models used in this project are supervised learning and regression algorithms (i.e., prediction of continuous data wind conditions) so that the algorithms, which solve the regression problems will be used. These are described in more detail in the following sections.

Linear Regression

The probably most widely used regression algorithm is linear regression, which trains the model by using the independent variables or features to predict the dependent variable or target. The most important prerequisite in order to have good predictions is to have the linear relationship between the features and target as shown in Figure 2.2 [10]. Features are the independent variables and target is the dependent variable in the data [11].

If the target or dependent variable is predicted by using only one feature, the approach is called linear regression. If it is predicted by using n number of features is called multivariate regression. linear regression line is of the form

$$y = mx + c \tag{2. 2}$$

Multivariate regression is formulated as in (2. 3) which takes n-independent variables (x1, x2, ...) to describe the dependent variable.

$$y = m_1x_1 + m_2x_2 + \ldots c \tag{2. 3}$$

Where m is slope of line and c is the intercept.

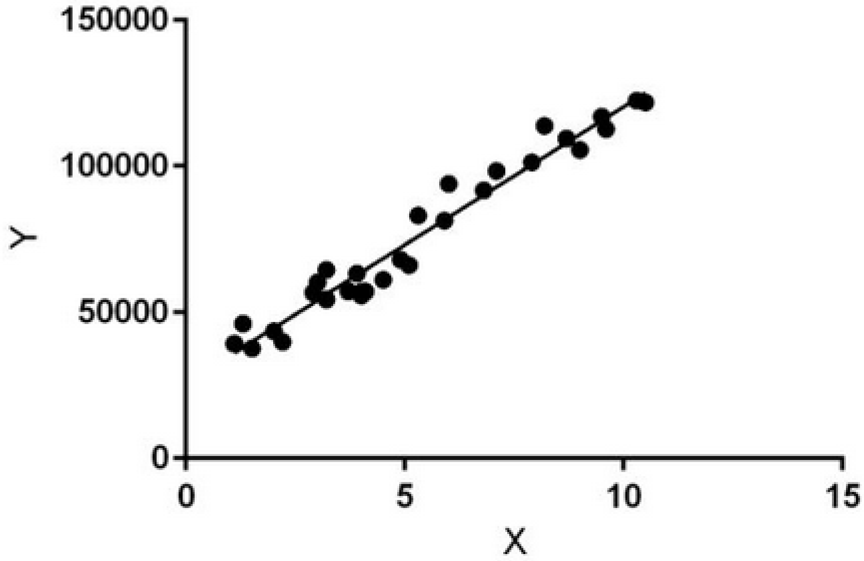

Figure 2.2: Regression line [10]

y is the dependent variable or in other words the predicted value and x, x1, x2.. are independent variables (features).

Support vector machine

Vapnik has first introduced support vector machine (svm). It is a supervised machine learning having two main categories of algorithms, support vector classification (SVC) used to classify the data and support vector regression (SVR) used for regression [12]. Svm is a learning system using a high dimensional feature space.

The main objective of svm algorithm is to find a hyperplane in an N-dimensional space (for N number of features) that classifies the data points. Hyperplanes are the decision boundaries that help classify the data points [13].

Support vector regression (SVR)

SVR algorithm will find an appropriate line (or hyperplane in higher dimensions) to fit the data by giving the flexibility to define how much error is acceptable in the model. In contrast to ordinary least square, the objective function of SVR is to reduce the coefficients i.e., l2 norm (Normalization technique used to reduce the noise) of the coefficient vector not the squared error. SVR projects training data with nonlinear structure-activity relationships in a given feature space into higher-dimensional space representations where a linear regression function may be derived [14].

Figure 2.3 shows the visual representation of the marginal error in a model with weights (wi), data x, y in x and y axis.

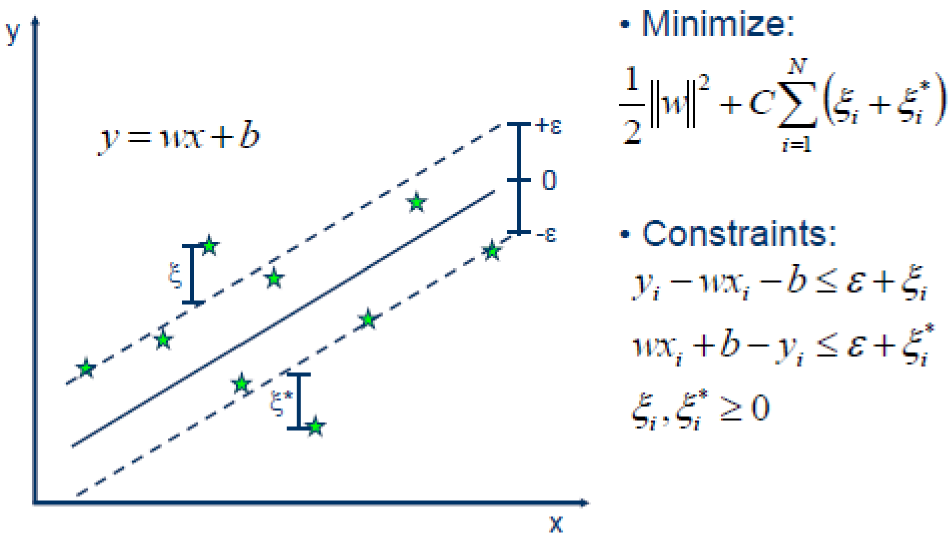

Figure 2.3: Support vectors with marginal errors, epsilon [13]

Random forest

Random forest is an ensemble learning. It is the process of using multiple models, trained over the same data, averaging the results of each model, and ultimately finding a more robust predictive/classification result. The requirement for ensemble learning is that the errors of each model are independent and different from tree to tree [15].

A random forest is a combination of N-decision trees, and these are used for regression and classification problems. They visually represent trees, and in the regression type, they start with the root of the tree and follow splits based on variable outcomes until a leaf node is reached and the result is given.

The random forest algorithm more accurately estimates the error rate than decision trees. More specifically, the error rate has been mathematically proven to always converge as the number of trees increases.

Random Forest Regression Model:

sklearn module for training the random forest regression model is used, precisely the RandomForestRegressor function. The RandomForestRegressor documentation shows many different parameters we can select for our model. Some of the crucial parameters [16] are explained below:

- **n_estimators** — the number of decision trees.
- **criterion** — The function which is used to measure the quality of a split. Supported criteria are “squared_error” for the mean squared error, which is equal to variance reduction as a feature selection criterion, “absolute_error” for the mean absolute error, and “poisson,” which uses reduction in Poisson deviance to find splits. Training using “absolute_error” is significantly slower than when using “squared_error.”
- **max_depth** — The maximum depth of the tree.
- **max_features** — The number of features to consider when looking for the best split.

Random forest is most widely used for classification type of problems but use of it in regression is also well known for its good accuracy, easy to handle, works well on large datasets and can handle multiple data types.

The predicted result is the average value of all the decision trees and input variables would not require any scaling because of its robustness. Some of the downsides of this model is overfitting, it is computationally expensive and does not work well with sparse datasets [17].

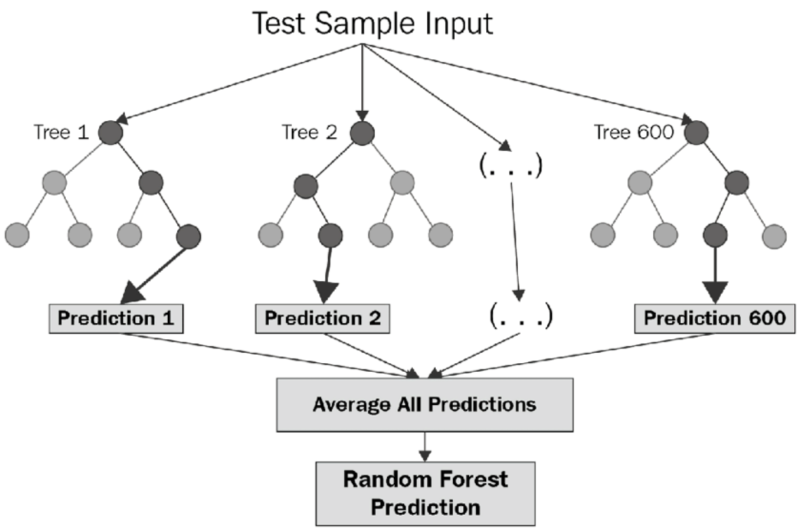

Figure 2.4: Random forest structure [17].

Neural network

Artificial neural networks are software implementations of the neuronal structure of human brains. They are generally comprised of neurons whose input values are passed through by an activation function and output the result, weights which carry values between neurons [18].

Neurons are grouped into three different layers.

- Input layer
- Hidden layer (s)
- Output layer

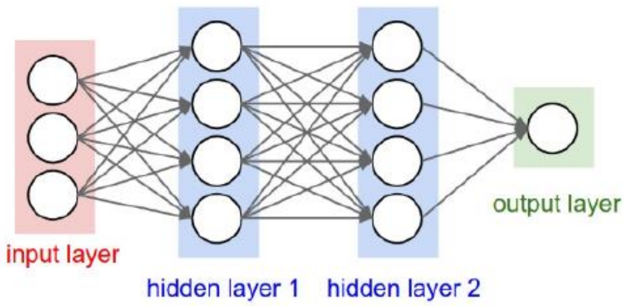

Figure 2.5: Basic neural network structure [18]

The functionality of a single neuron can be seen by breaking down the single neuron from neural network architecture. A single neuron takes the values of all the previous layer neurons. It is multiplied by another variable called weight, which determines the connection between the two neurons. Each neuron has its weight, which is modified again during the learning process, particularly in the backpropagation phase [19].

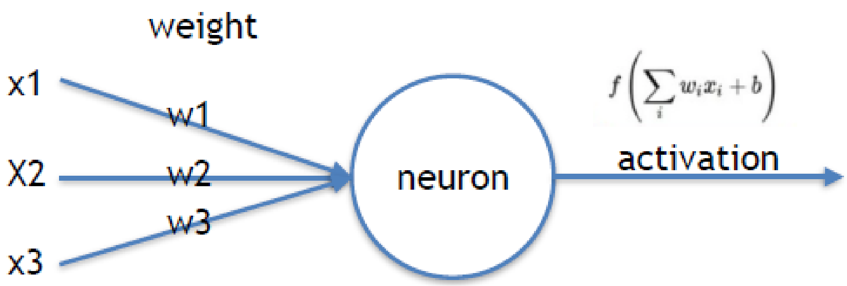

Figure 2.6: Breakdown of single neuron [18]

All these summations of neurons apply to a function called activation function which will convert the total value between 0 and 1. The most common activation functions are:

- ReLU: $$\mathrm{ReLU}(x) = \max(0,x) \tag{2.4}$$
- Sigmoid: $$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$
- Tanh: $$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2.6}$$

<u>Learning process:</u>

After defining all the neurons with a specific number of layers, input is given to the network and an output is returned. On the first try, it cannot get the correct output on its own, so during the learning phase, every input has its label explaining to the model what output it should have guessed. However, if the output does not match or comes close to the label, it will update its weights through the backpropagation technique [20]. The other parameter that controls the neural network is learning rate, which means how it will modify a weight, little by little or by more significant steps. One forward and backward phase is called an epoch, and one can run the model through an N-number of epochs until a low error between the actual value and predicted value is obtained.

3. Data basis and data preparation

In the chapter 2, there is a detailed explanation about the data (met mast, SCADA, ICON-D2) that were used in this project. In this chapter, there will be a detailed explanation of source of the data and a clear understanding of data (i.e., duration and frequency of data).

Fraunhofer IEE provides the weather model (ICON-D2) data and their project (WINDOW) partners provided met mast data and SCADA data.

3.1 MET MAST

The total available data recorded for this project is from 12-04-2021 11:40:00 to 08-01-2022 23:50:00 and the different measuring instruments measure their corresponding parameters of wind, i.e., wind speed, wind direction, humidity, temperature, and pressure at several heights (142, 136, 119, 99, 79, 59, and 39 metres) as shown in Table 3.1.

Sensor height (m)	**Sensor type**
Wind Speed (m/s)	
142.2	Ultrasonic
136.0	Anemometer
119.1	Anemometer
99.0	Anemometer
79.0	Anemometer
59.1	Anemometer
39.0	Anemometer
Wind Direction A°	
142.2	Ultrasonic
136	Wind vane

78.9	Wind vane
Humidity % **Temperature °C**	
136.5	Hygro thermo
98.9	Hygro thermo
30.0	Hygro thermo
Air pressure hpa	
16.0	Barometer
Precipitation mm	
134.5	Precipitation

Table 3.1: Different atmospheric parameters recorded by multiple sensors in met mast tower

3.2 SCADA

SCADA (Supervisory Control and Data Acquisition) data for wind turbines is collected and prepared through a combination of hardware and software systems.

First, sensors are placed on the wind turbine to collect data such as wind speed, turbine rotation speed, and power output. These sensors are connected to a data acquisition system, which is responsible for collecting and storing the data. The data acquisition system is typically connected to a programmable logic controller (PLC), which is a type of computer specifically designed for industrial control systems. The PLC acts as a bridge between the sensor data and the SCADA software, which is responsible for monitoring and controlling the wind turbine [21].

The total available SCADA data recorded is from 01-01-2021 00:00:00 to 31-12-2021 23:50:00 in a 10-minute interval. In general, the SCADA data contains recorded wind speeds, wind directions at a turbine and active power generated by a turbine for every second and this data is grouped based on different statistics (maximum value, minimum value, average value, standard deviation) for every 10 minutes and the average value is considered.

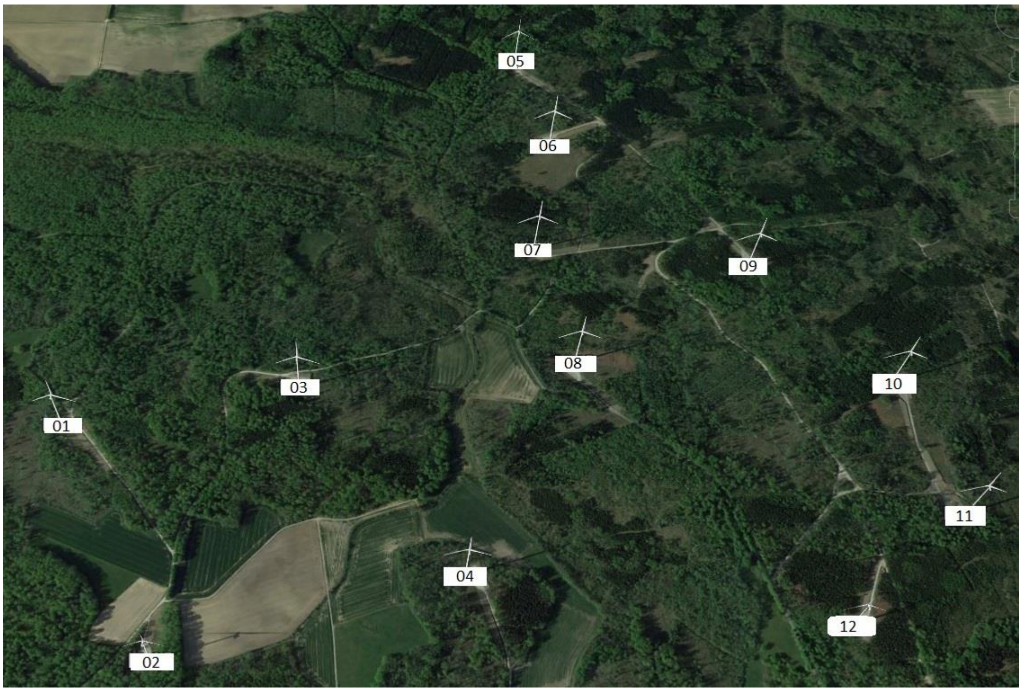

Figure 3.1: Geographic positions of the turbines in langenburg wind park

As shown in Figure 3.1, at these positions, wind speeds, wind directions and active power of twelve different turbines were recorded.

Only the wind speeds of each turbine are used for further work and these data are used as labelled data, which means that the machine learning model will try to predict the outcome of wind speed by looking at the other features of met mast and icon data.

3.2 ICON-D2

ICON-D2 data are weather model data. As shown in Figure 3.2, data is being recorded at each grid point with a resolution of 0.01° lat and 0.02° long. The green line boundary represents the border of the wind park. The geographic locations of the grid points in the wind park are shown in Figure 3.3.

Different atmospheric parameters, i.e.., wind speed (ws), wind direction (dir), pressure (press), temperature (temp), specific humidity (spec_hum), turbulent kinetic energy (tke), convective available potential energy (cape), precipitation (precp), diffusion flux (diff_flux), solar radiation (rad) and surface roughness (rh) are available at different heights from the time period of 2021-02-10 00:00:00 to 2022-01-30 23:00:00 in one-hour intervals.

For this project, the grid point nearest to or within the wind park as shown in Figure 3.2 is considered for further analysis. A direct linear interpolation would be sufficient to convert into 10-minute resolution in order to match the time stamps of other data (Met mast, SCADA).

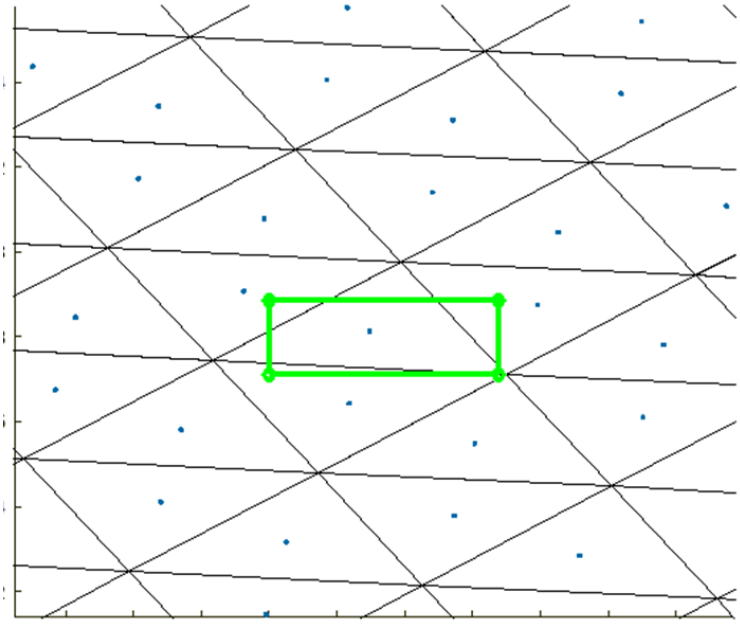

Figure 3.2: Grid points of ICON-D2 data

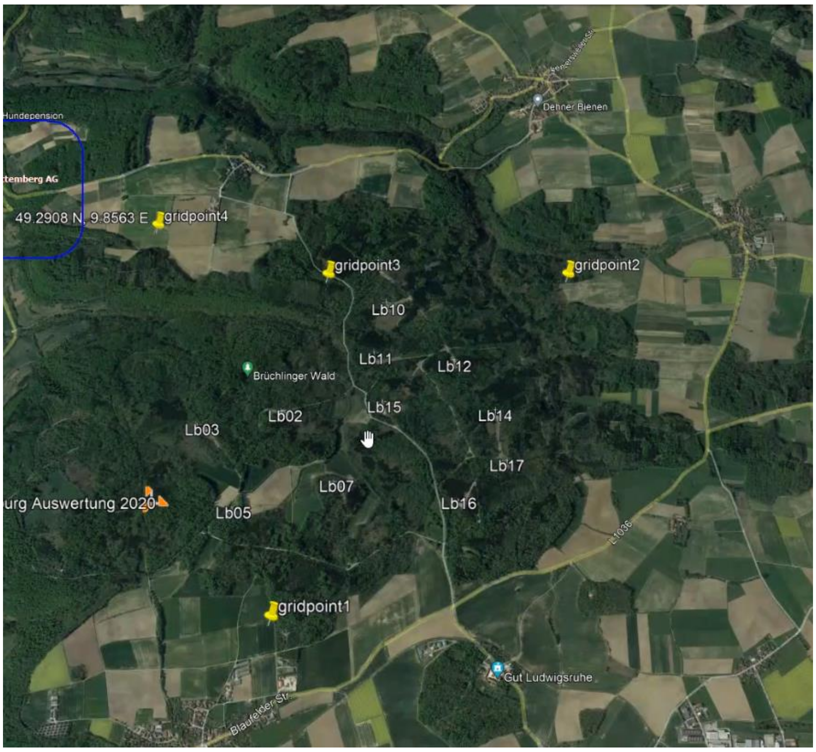

Figure 3.3: Geographic locations of the grid points in Langenburg wind park

4. Methodology

For this project, python programming language along with other modules like pandas, numpy, matplotlib, sklearn and tensor flow were used.
All these libraries have their own importance in the following aspects:

- Pandas is for data handling and data manipulation.
- Numpy for handling multi-dimensional arrays along with some high-level mathematical functions to operate on these arrays.
- matplotlib for data visualization in the form of plots.
- sklearn and tensor flow are the frameworks for machine learning.

4.1 Preparing the data

SCADA and met mast data are in csv format, which can be easily read using pandas in Python, whereas the ICON-D2 data were provided in mat format. These data were converted into csv format just like SCADA and the met mast data.

The input and output (Io) package in the scipy library is used to read the information from mat format files in Python. Different atmospheric parameters at different heights and their time stamps are written in structured arrays, and the data in these arrays are pulled out at their corresponding time stamps and re-written as numpy arrays. By using the Pandas library, all these numpy arrays and time stamps are converted into a pandas data frame and saved in csv format.

As shown in Figure 4.1, Figure 4.2. cell code, it performs pre-processing of the met mast and SCADA data which is extracting the necessary columns from raw data set then removing the null values, characters and at last converting the data into their corresponding data type i.e., values into float and date time index into pandas date time.

```
class Messmast:
    def __init__(self):
        self.adress = 'D:\\files\\test\\Messmast.csv'
        self.df = pd.read_csv(self.adress)

    def get_totaldata(self):
        return self.df

    def get_formatdata(self):

        '''pre processing the met mast data '''

        self.df = self.df.drop(columns='Unnamed: 0', axis=1)
        self.df['Date/time'] = pd.to_datetime(self.df['Date/time'])
        self.df = self.df.set_index(self.df['Date/time'])
        self.df = self.df.drop('Date/time', axis=1)
        self.df = self.df.replace(r'\W', np.NaN, regex=True)
        self.data = self.df.rename(
                    columns={
                            'VZ1-03210099;wind_speed;Avg':'ws_142',
                            'V01-11208098;wind_speed;Avg':'ws_136',
                            'V02-11208099;wind_speed;Avg':'ws_119',
                            'V03-11208100;wind_speed;Avg':'ws_99',
                            'V04-11208101;wind_speed;Avg':'ws_79',
                            'V05-11208102;wind_speed;Avg':'ws_59',
                            'V06-11208103;wind_speed;Avg':'ws_39',
                            'VZ1-03210099;wind_direction;Avg':'wd_142',
                            'D01-08170370;wind_direction;Avg':'wd_136',
                            'D02-08170369;wind_direction;Avg':'wd_79',
                            'HT1-236609;humidity;Avg':'H_136',
                            'HT2-236616;humidity;Avg':'H_99',
                            'HT3-236615;humidity;Avg':'H_30',
                            'VZ1-03210099;temperature;Avg':'Temp_142',
                            'HT1-236609;temperature;Avg':'Temp_136',
                            'HT2-236616;temperature;Avg':'Temp_99',
                            'HT3-236615;temperature;Avg':'Temp_30',
                            'BP1-150931338;air_pressure;Avg':'Press_16',
                            'VZ1-
03210099;wind_speed_vert;Avg':'ws_vert_142'})

        self.data =
self.data.loc[:,['ws_142','ws_136','ws_119','ws_99','ws_79','ws_59','ws_39'
,'ws_vert_142','wd_142','wd_136','wd_79','H_136', 'H_99',
'H_30','Temp_142','Temp_136','Temp_99','Temp_30','Press_16']]

        self.data['ws_142'] = self.data['ws_142'].astype('float')
        self.data['ws_vert_142'] = self.data['ws_vert_142'].astype('float')
        self.data['wd_142'] = self.data['wd_142'].astype('float')
        self.data['Temp_142'] = self.data['Temp_142'].astype('float')

        return self.data
```

Figure 4.1: code cell for pre-processing the met mast data

```
class Turbine:

    def __init__(self):
        self.df =
pd.read_csv('D:\\files\\turbine_power\\turbine_edited\\combined_data.csv')


    def get_data(self):
        '''Returns all the turbine data '''

        self.df['Zeitstempel (UTC)'] = pd.to_datetime(self.df['Zeitstempel
(UTC)'])
        self.df = self.df.set_index(self.df['Zeitstempel (UTC)'])
        self.df = self.df.drop('Zeitstempel (UTC)', axis=1)
        return self.df
```

Figure 4.2: code cell for pre-processing the scada data

4.2 Data transformation

As the frequency of the time stamps in ICON-D2 data is in hourly resolution, while the mast data and SCADA data are given in 10-minute intervals, the ICON-D2 data were converted into 10-minute values, ensuring the same, consistent temporal resolution. In order to do so, a simple interpolation was used.

For the parameters wind speed, temperature, pressure, precipitation, and humidity, a simple linear interpolation was applied for the interpolation. For the wind direction, linear interpolation cannot be used because the direction is in circular units (0° to 360°). For this reason, a circular interpolation was applied.

Linear interpolation

Linear interpolation is useful when looking for a value between given data points. It can be considered as “filling in the gaps” of a table of data. The strategy for linear interpolation is to use a straight line to connect the known data points on either side of the unknown point [22].

$$y(x) = y1 + (x - x1)\left(\frac{y2 - y1}{x2 - x1}\right) \tag{4. 1}$$

$x1, x2$ = 1, 6

$y1, y2$ = values at corresponding timestamps.

x is the independent variable and y is the interpolated value. The available data is in hourly resolution and it needs to be converted into 10-minute resolution, so in order to have five timestamps in a hour $x1, x2$ are chosen as 1, 6 Therefore, there will be five interpolated values in between one hour.

<u>Circular interpolation</u>

The circular interpolation for the circular data which is only in range of 0° to 360° can be achieved by using the unwrap function in the numpy module. It unwraps the data by changing the deltas between values to 2π complement [23].

Before the use of unwrap, a deg2rad function was applied which converted degrees into radians. Afterwards, direct linear interpolation can be done on these data points, which returns a 1-D array of interpolated values within the circular range. After all the interpolation, the frequency of the data converted to 10-min.

Below is the cell code as shown in Figure 4.3, linear interpolation and circular interpolation functions. As shown in Figure 4.4 the interpolation functions are used on every column in the data set (df) and converted into 10 minute resolution. A while loop is used to insert the additional interpolated values as a row in the existing data frame df.

```python
def y_interp(y1,y2):
        x = [0, 6]
        y = [y1, y2]

        # test value
        interpolate_x = [1, 2, 3, 4, 5]

        # Finding the interpolation
        y_interp = interp1d(x, y)

        return y_interp(interpolate_x)

def circular_interpl(y1, y2):

        x1 = 0
        x2 = 6

        data = np.array([[x1, x2], [y1, y2]])
        complement360 = np.rad2deg(np.unwrap(np.deg2rad(data[1])))
        f = interpolate.interp1d(data[0], complement360, kind='linear',
bounds_error=False, fill_value=None)

        return f(np.arange(7)[1:-1])%360
```

Figure 4.3: Code cell for interpolation functions

```python
def interpolate(df):

    #Tranforming the raw data into interpolated data
        i = 2
        while i <= df.shape[0]:
            row = {'ws_10m': y_interp(df.iloc[i-2,0], df.iloc[i-1,0]),
               'ws_38m':y_interp(df.iloc[i-2,1], df.iloc[i-1,1]),
               'ws_78m':y_interp(df.iloc[i-2,2], df.iloc[i-1,2]),
               'ws_127m':y_interp(df.iloc[i-2,3], df.iloc[i-1,3]),
               'ws_184m':y_interp(df.iloc[i-2,4], df.iloc[i-1,4]),
               'dir_10m':circular_interpl(df.iloc[i-2,5], df.iloc[i-1,5]),
               'dir_38m':circular_interpl(df.iloc[i-2,6], df.iloc[i-1,6]),
               'dir_78m':circular_interpl(df.iloc[i-2,7], df.iloc[i-1,7]),
               'dir_127m':circular_interpl(df.iloc[i-2,8], df.iloc[i-1,8]),
               'dir_184m':circular_interpl(df.iloc[i-2,9], df.iloc[i-1,9])
,
                'temp_2m':y_interp(df.iloc[i-2,10], df.iloc[i-1,10]),
               'temp_127m':y_interp(df.iloc[i-2,11], df.iloc[i-1,11]),
                'specHum_127m':y_interp(df.iloc[i-2,12], df.iloc[i-1,12]),
               'relHum_2m':y_interp(df.iloc[i-2,13], df.iloc[i-1,13]),
                'albedo':y_interp(df.iloc[i-2,14], df.iloc[i-1,14]),
               'diff_flux':y_interp(df.iloc[i-2,15], df.iloc[i-1,15]),
                'rad':y_interp(df.iloc[i-2,16], df.iloc[i-1,16]),
               'press':y_interp(df.iloc[i-2,17], df.iloc[i-1,17]),
                'roughness':y_interp(df.iloc[i-2,18], df.iloc[i-1,18]),
               'precipitation':y_interp(df.iloc[i-2,19], df.iloc[i-1,19]),
                'ws_max_10m':y_interp(df.iloc[i-2,20], df.iloc[i-1,20])}

            line = pd.DataFrame(row, index=[z for z in range(i,i+5)])
            data = pd.concat([df.iloc[:i-1], line, df.iloc[i-1:]])
            i = i+6

        return data
```

Figure 4.4: Code cell to convert one hour resolution into 10-min resolution

```
class Icon:

    def __init__(self):
        self.df=
pd.read_csv('D:\\files\\Icon_D2\\new\\Icon_gridcellcenter_interpolated.csv)


    def get_data(self):

        self.df = self.df.rename(columns={'Unnamed: 0':'Date_time'})
        self.df['Date_time'] = pd.to_datetime(self.df['Date_time'])

        self.df = self.df.set_index(self.df['Date_time'])
        self.df = self.df.drop('Date_time', axis=1)
        return self.df
```

Figure 4.5: Code cell for pre-processing the ICON-D2 data

4.3 Data splitting

After the preprocessing and data transformation, the data is split into training and testing sets. The training set of data is used in the learning process of the machine-learning algorithm, and the test set of data is used to test the model's performance by making predictions. The general convention for splitting the data is 70–30% (70% training and 30% testing) or 80–20% [24].

The target feature or parameter for a machine-learning model is the wind speed of different turbines (SCADA data). Met mast, ICON-D2 data are combined with SCADA data in order to train the model against different features and this can be done by using the pandas merge function, which merges two data frames at their corresponding time-stamps.

The samples of the data are in time series, so splitting of the data is not random. Instead, the last 20 % of the data is kept on the test side, and the remaining total available data are on the training set.

4.4 Training the model

After splitting up the total available data into a training set and a testing set, different machine learning models (Linear regression, Random Forest, Support Vector Regression, Neural Network) are trained by using the training set data and tested on the testing set.

For the linear regression, support vector regression and random forest, the in-built algorithms from sklearn library are used and for the neural network TensorFlow library is used.

The Linear Regression class from sklearn.model_selection [25] module is used to build a linear regression model. To train the model, it is to be called with fit method, which takes the features (x_train) and labels (y_train) as arguments. In total, there will be twelve different models for twelve turbines.

Similarly, for the support vector regressor (SVR class), random forest regressor (RandomForestRegressor) is chosen from skleran.svm and sklearn.ensemble. tensorflow.keras is used to build and train the neural network model. The machine learning model is trained by using x_train and y_train for all the turbines as shown in Figure 4.

```
In [8]: model_2 = lr.trainmodel(x_train_2, y_train_2)
        model_3 = lr.trainmodel(x_train_3, y_train_3)
        model_5 = lr.trainmodel(x_train_5, y_train_5)
        model_7 = lr.trainmodel(x_train_7, y_train_7)
        model_9 = lr.trainmodel(x_train_9, y_train_9)
        model_10 = lr.trainmodel(x_train_10, y_train_10)
        model_11 = lr.trainmodel(x_train_11, y_train_11)
        model_12 = lr.trainmodel(x_train_12, y_train_12)
        model_14 = lr.trainmodel(x_train_14, y_train_14)
        model_15 = lr.trainmodel(x_train_15, y_train_15)
        model_16 = lr.trainmodel(x_train_16, y_train_16)
        model_17 = lr.trainmodel(x_train_17, y_train_17)
```

Figure 4.6: Code cell for training the model

Models Parameters:

In this section, the parameters used in every model are discussed.

LinearRegression:

From sklearn.linear-model module LinearRegression model was used for the linear model in this project and default parameters were used. Some of the parameters in this model are.,

- fit_intercept: bool, True, whether to calculate the intercept for this model as discussed in Eq. (2. 2).
- normalize: bool, True, the regressors X will be normalized before regression
- n_jobs: None, No of jobs to use for the computation.

RandomForestRegressor:

As discussed previously, random forest is a collection of decision trees and it is an ensemble (Bagging) technique.

From sklearn.ensemble module RandomForestRegressor model was used for the random forest model in this project and default parameters were used. Some of the parameters in this model are.,

- n_estimators: default=100, number of trees in the forest.
- criterion: 'squared_error' function to measure the quality of split.
- max_depth: None, if None then nodes are expanded until all leaves are pure.
- min_samples_split: 2, minimum number of samples required to split an internal node.

Support Vector Regressor:

From sklearn.svm module SVR model was used for the support vector model in this project and default parameters were used. Some of the parameters in this model are.,

- kernal: 'rbf' is a kernal function used in the algorithm.
- degree: 3, degree of the polynomial function.
- epsilon: 1, epsilon in the epsilon-svr model. It specifies the epsilon tube within which no penalty is associated in the training loss function.

Neural Network:

Tensorflow keras, a deep learning library has been used for building the neural network model. Parameters used in this network are.,

- hidden layers: two hidden layers with 42 and 84 neurons along with dropout layer which will helps to drop the inactivated neurons.
- activation function: 'relu' , an activation function used on every neuron.
- optimizer: 'Adam' helps to find out the local minima of the errors in learning stage.
- learning_rate: 0.0001, learning rate which will be applied to descent in backward propagation.
- loss: 'mean_squared_error', loss function.

The source code for all the data-preprocessing, model training and plotting the errors can be found in Appendix D.

4.5 Predictions

After completion of the learning process, the model is ready to make some predictions by using the predict method as shown in Figure 4.7 that takes test features (x_test) as an input argument and gives the predicted results.

```
In [9]: predictions_2 = model_2.predict(x_test_2)
        predictions_3 = model_3.predict(x_test_3)
        predictions_5 = model_5.predict(x_test_5)
        predictions_7 = model_7.predict(x_test_7)
        predictions_9 = model_9.predict(x_test_9)
        predictions_10 = model_10.predict(x_test_10)
        predictions_11 = model_11.predict(x_test_11)
        predictions_12 = model_12.predict(x_test_12)
        predictions_14 = model_14.predict(x_test_14)
        predictions_15 = model_15.predict(x_test_15)
        predictions_16 = model_16.predict(x_test_16)
        predictions_17 = model_17.predict(x_test_17)
```

Figure 4.7: Code cell for making predictions

The predictions and real values (y_test) are then compared to evaluate the model performance. The evaluation of the model are identified by using different metrics from sklearn i.e., root mean squared error, mean absolute error, change in variance and bias.

4.6 Error scores

Root Mean Squared Error (RMSE):

The RMSE is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are scattered [26].

$$\text{RMSE} = \sqrt{\frac{\sum(y_i - \bar{y}_i)^2}{N}} \qquad (4.\ 2)$$

Where y_i = True value

$\bar{y}_i$ = predicted value

N = Total number of samples

Mean Absolute Error (MAE):

Mean absolute error of a model is the mean of the absolute values to the individual prediction errors on overall instances (samples) in the test set. Each prediction error is the difference between the true value and the predicted value for the instance [27].

$$\mathrm{MAE} = \frac{\sum \mathrm{abs}(y_i - \bar{y}_i)}{N} \tag{4. 3}$$

Where y_i = True value

$\bar{y}_i$ = predicted value

N = Total number of samples

Bias in mean and variance:

Bias in mean wind speed is the difference between the average of the model predicted value and true value which the model is trying to predict. Similarly bias in variance is the difference between variability of model prediction and true value for a given data point [28].

The RMSE (Root mean squared error) values, change in variance and bias for twelve different turbines are represented in a box plot.

Errors are calculated by using the methods as shown in code cell Figure 4.8.

```
def rmse(self, y_test, predictions):
    return np.sqrt(mean_squared_error(y_test, predictions))

def mae(self, y_test, predictions):
    return mean_absolute_error(y_test, predictions)

def variance(self, y_test, predictions):
    var = (np.var(predictions) - np.var(y_test))/np.var(y_test)
    return var

def bias(self, y_test, predictions):
    bias =  (np.mean(predictions) - np.mean(y_test))/np.mean(y_test)
    return bias
```

Figure 4.8: Code cell for calculating the errors

5. Results

In this chapter, the different error scores (RMSE, bias in variance, and bias in mean wind speed) are evaluated for every machine learning model and compared against different types of data. RMSE scores of the machine learning models indicate the absolute fit of the model to the data. In other words, how close the observed data points are to the model's predicted values so, the low value of RMSE implies that the model predicted values are closer to the real values.

<u>Dependence of met mast distance</u>:

Figure 5.1 below represents the root mean squared error of the linear regression model against a test data set for different turbines (from T1 to T12) in a wind park. In Appendix , B.3, B.4 RMSE for other models can be observed. This plot helps to understand how error values vary from one turbine to another. On the x-axis, the turbine numbering is sorted based on the distance between the met mast and a wind turbine from low to high, and their corresponding distances (meters) are indicated on the label.

By observing Figure 5.1, it is noted that the error scores RMSE are not dependent on the distance between the met mast and turbine. Another possibility of visualizing the errors by including all of the turbines and comparing them with other data sets (icon, icon+ met mast) is a box plot.

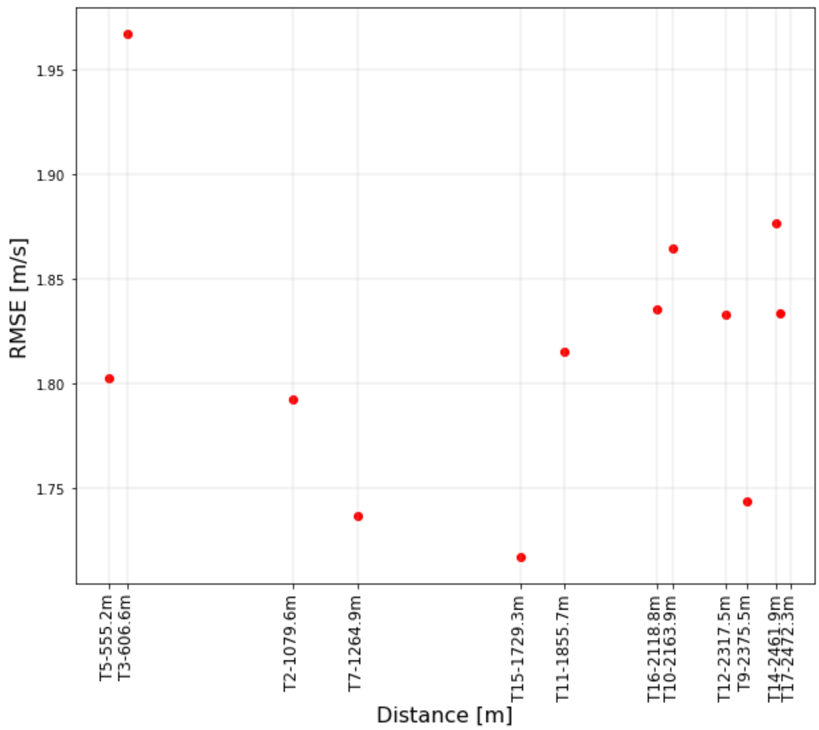

Figure 5.1: RMSE of linear regression model

A box plot is used to visualize the error score distribution of twelve different turbine data, as shown in Figure 5.2. Each dot represents one set of turbine data, so there will be twelve dots in a box plot (i.e., one for each turbine). Maximum, upper quartile, median, lower quartile range and minimum values are represented from left to right in every plot and with this, the distribution of the error scores in a wind park for every model can be identified.

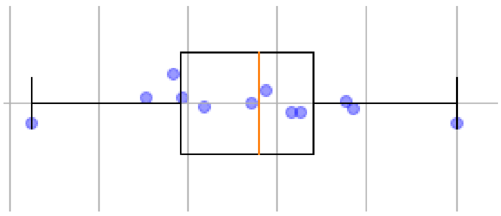

Figure 5.2: Box plot

5.1 Comparison of the different ML models: RMSE

The RMSE of different models (linear regression, random forest, random forest with selected features, support vector regressor, and neural network) for different data sets (met mast, icon, icon+mast) are discussed in this section. This enables the analysis of the accuracy of the models with regard to the estimates of the wind conditions at the location of the target turbine.

In Figure 5.3, it is observed that the RMSE values of the linear regression model for the twelve turbines are grouped based on the type of data set used in the model training. A simple box plot is used to visualize the errors of each turbine. Types of data sources are plotted on the x-axis, and RMSE values are on the y-axis. Figure 5.3 shows that the RMSE value when mast data has low error compared with error of icon and mast+icon data. RMSE of Mast+icon has very high errors between 18-21 m/s because of having more features in the data, and the linear regression model failed to find a linear relationship between all these features.

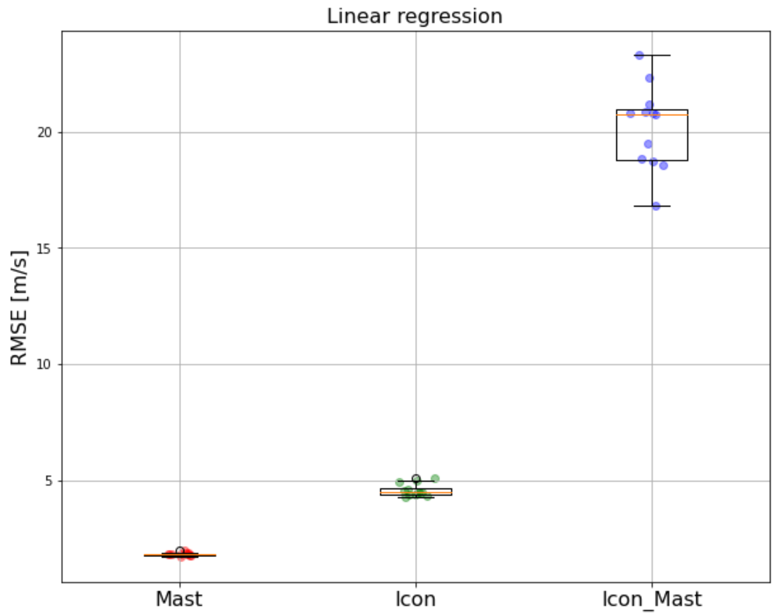

Figure 5.3: RMSE's of linear regression model.

In Figure 5.4, RMSE values of random forest model when using different data sets are shown. It has to be noted that the model is performing better when mast data is used between 1.9-2.2 m/s when compared to icon and icon+mast data.

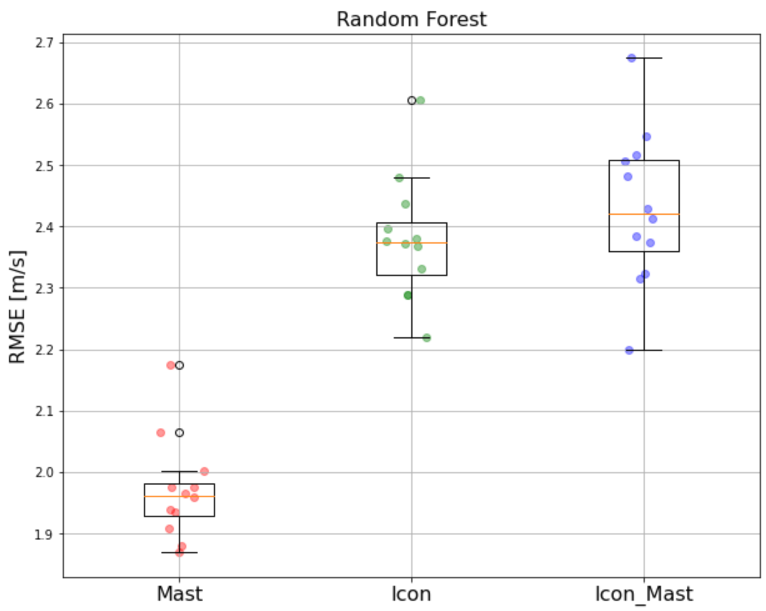

Figure 5.4: RMSE's of random forest model

In Figure 5.5, RMSE values of random forest with selected features model against different data sets are observed. In order to reduce the computational time the total number of features from individual data source are reduced. It is noted that the mast data is performing better when compared to icon and icon+ mast data.

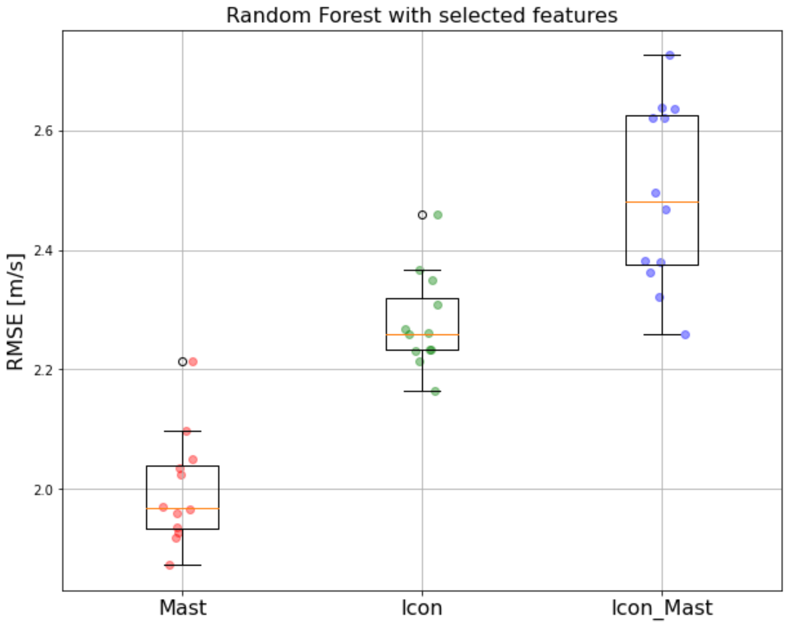

Figure 5.5: RMSE's of random forest model with selected features

Figure 5.6, It is noted that the RMSE values of support vector regressor model is showing similar trends when mast and icon+mast data are used, and are performing better when compared with icon data.

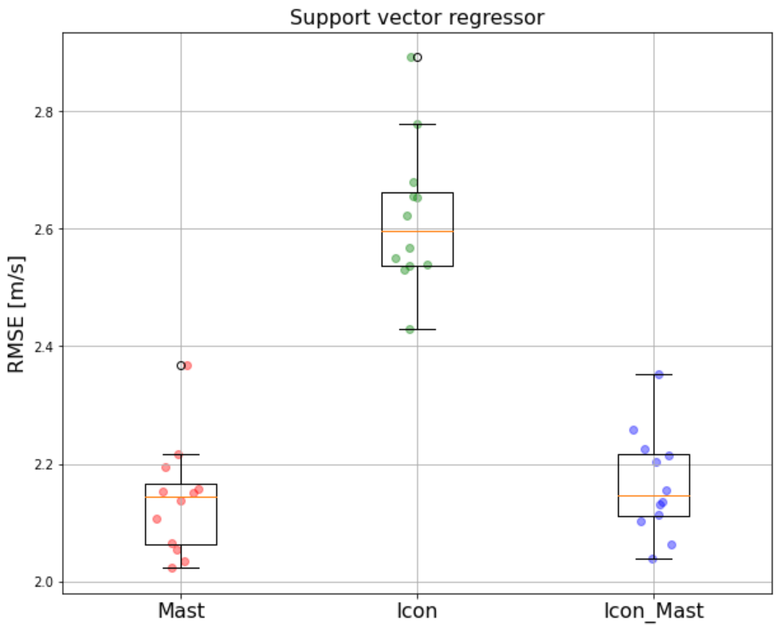

Figure 5.6: RMSE's of support vector regressor model

The RMSE of neural network model can be observed in Figure 5.7, it is to be noted that similar to support vector regrossor model the neural network model with mast and icon+mast data are performing better and shows similar trend when compared with icon data.

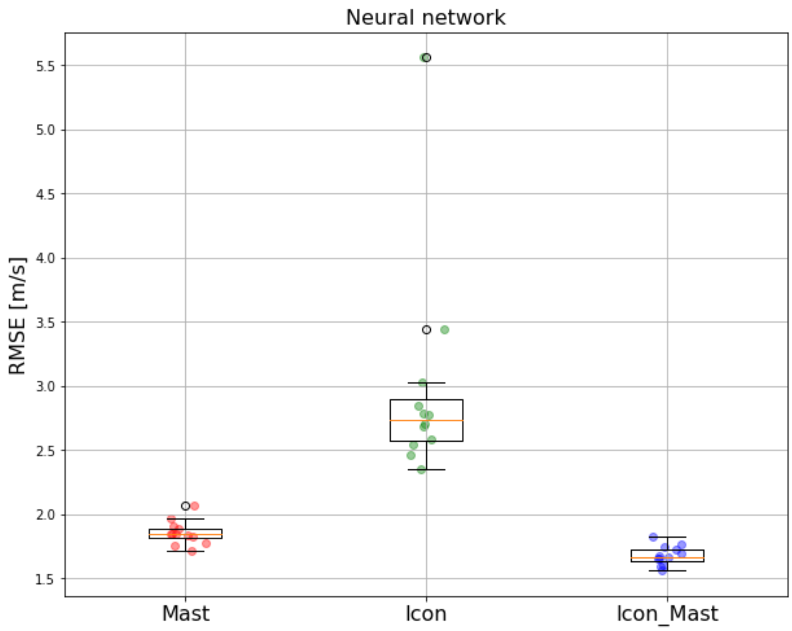

Figure 5.7: RMSE's of neural network model

By comparing all the models based on the RMSE metric, it is to be said that the random forest and neural network models have the lowest rmse values. Overall highest accuracy on average can be obtained by using neural network in particular for the met mast data, whereas the linear regression has a high rmse for icon+mast data. Therefore, the models random forest and neural network enable to best estimate the wind speeds using the met mast data.

5.2 Comparison of Bias in variance

The bias in variance of the different models (linear regression, random forest, random forest with selected features, support vector regressor, and neural network) for different data sets (Met mast, icon, icon+mast) are discussed in this section. It helps to analyze in which way the models deliver accurate variances.

The bias in the variance of linear regression model can be observed in Fig 5.8, in this plot the percentage of bias in variance on y-axis and types of data source used on x-axis. From Figure 5.8 it is to be concluded that there are strong systematic underestimations of the variance and the model is showing more negative variance when met mast is used compared to icon and icon+ mast.

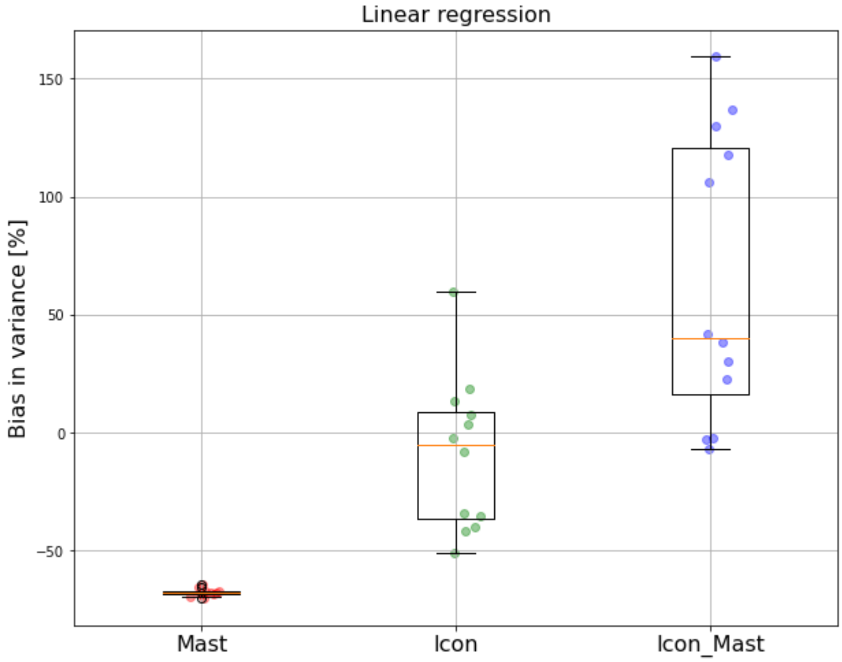

Figure 5.8: Bias in variance when linear regression model is used

In Figure 5.9, bias in variance of random forest model for all the data showing strong negative variance can be seen. Among them model is accurate when met mast data is used with percentage of -60 to -67 compared to the icon and icon+ mast data.

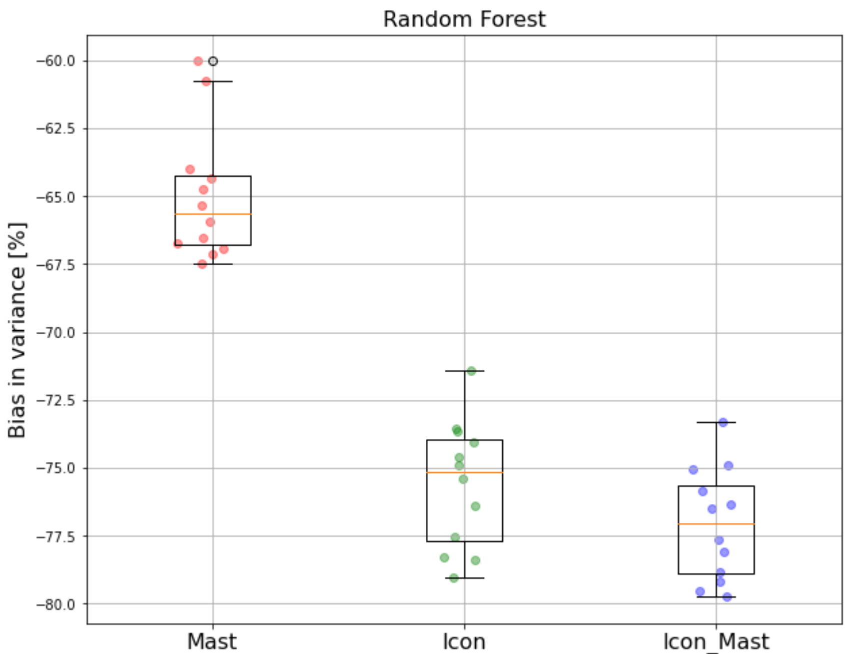

Figure 5.9: Bias in variance when random forest model is used

Figure 5.10, In contrast to the random forest with all the features, the bias in variance of random forest with selected features gives better results when icon is used compared to mast and icon+ mast data. Reducing the features from the icon data has some improvement in the model performance, but worse performance when using mast + icon data.

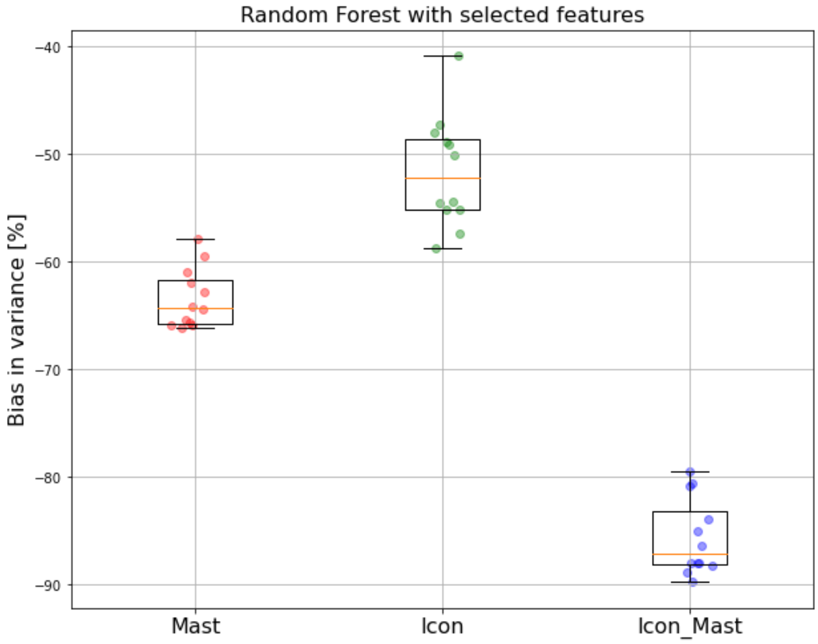

Figure 5.10: Bias in variance when random forest model with selected features is used

In Figure 5.11, bias in variance of support vector regressor model can be observed. Mast and icon+ mast data performing better compared to icon data.

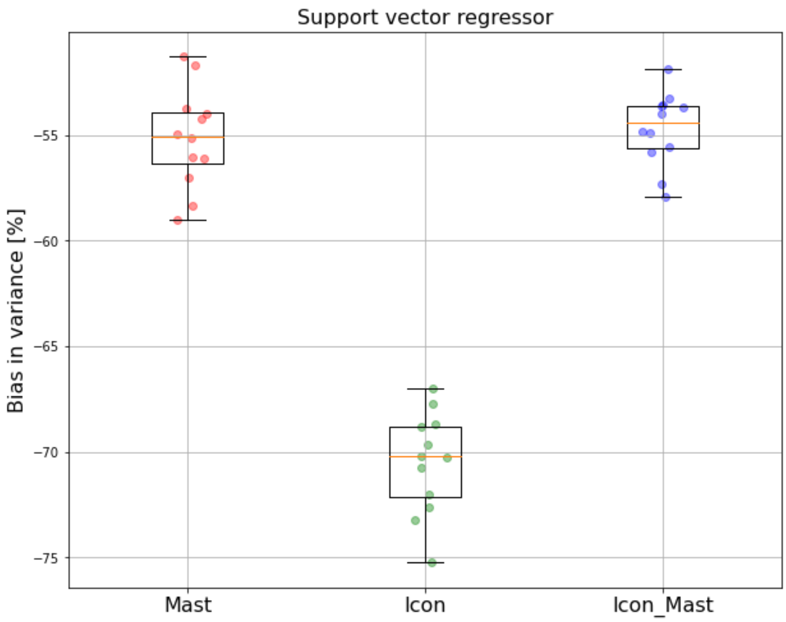

Figure 5.11: Bias in variance when support vector regressor model is used

Bias in variance of neural network model can be observed in Figure 5.12, all the three types of data shows similar results with average in between -50% to -60% bias in variance.

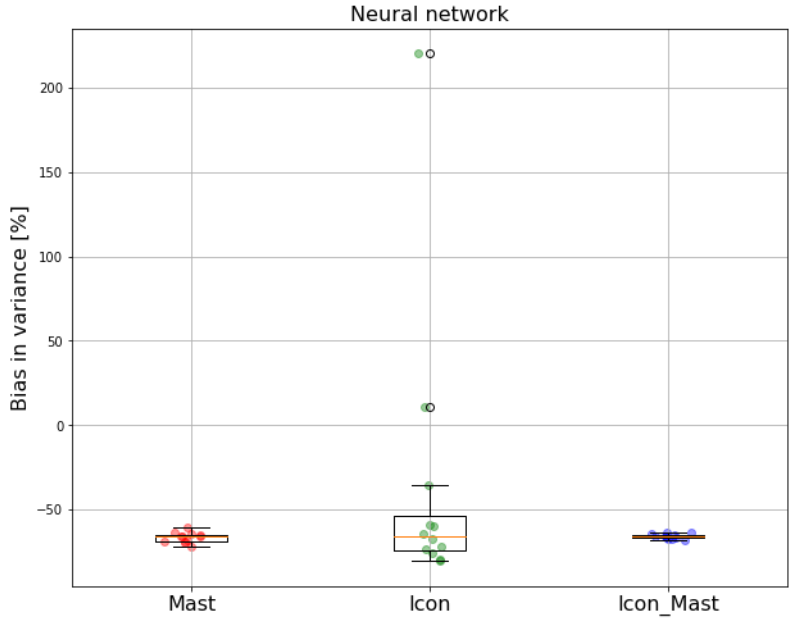

Figure 5.12: Bias in variance when neural network model is used

By comparing the bias in variance for all the models, it is to be concluded that all models strongly underestimate variance. The bias in variance is lowest for random forest, support vector regressor, and neural network. No clear dependence of the data set can be detected.

5.3 Comparison of Bias in mean wind speed

The bias in mean wind speed of different models (linear regression, random forest, random forest with selected features, support vector regressor, and neural network) for different data sets (met mast, icon, icon+ mast) are discussed in this section. This helps to analyze how much the output of a model is biased.

Figure 5.13. Bias in mean wind speed of the linear regression model can be observed, The biases for the mast and icon data are nearer to zero whereas icon+ mast are more biased with a value of -19%. The linear regression model is failed to find linear relationship between all the features i.e., combining the features from mast data and icon data.

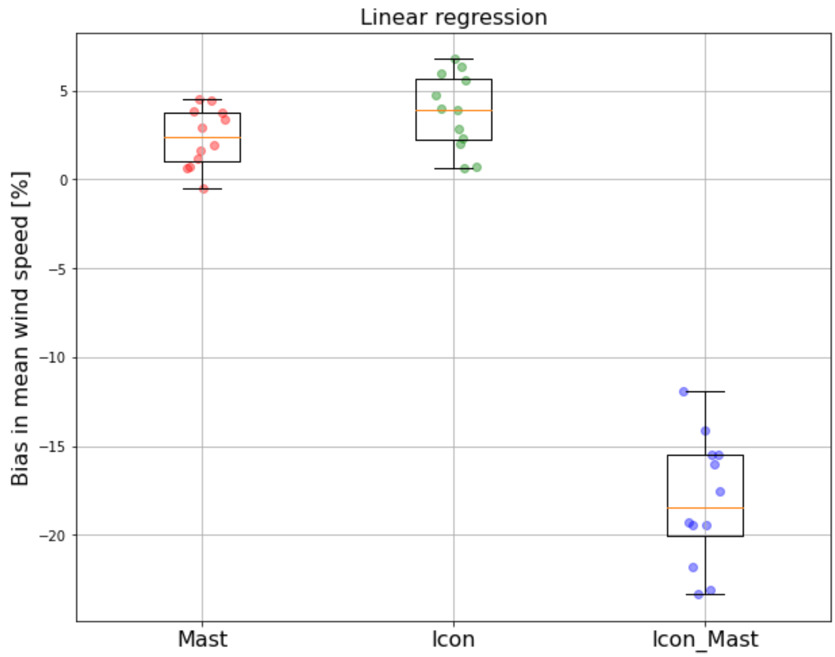

Figure 5.13: Bias in mean wind speed when linear regression model is used

Bias in mean wind speed of random forest model can be observed in Figure 5.14. The biases for the model when mast data is used are in between -2 to 3 %. For icon and icon+ mast data having around -17% on average.

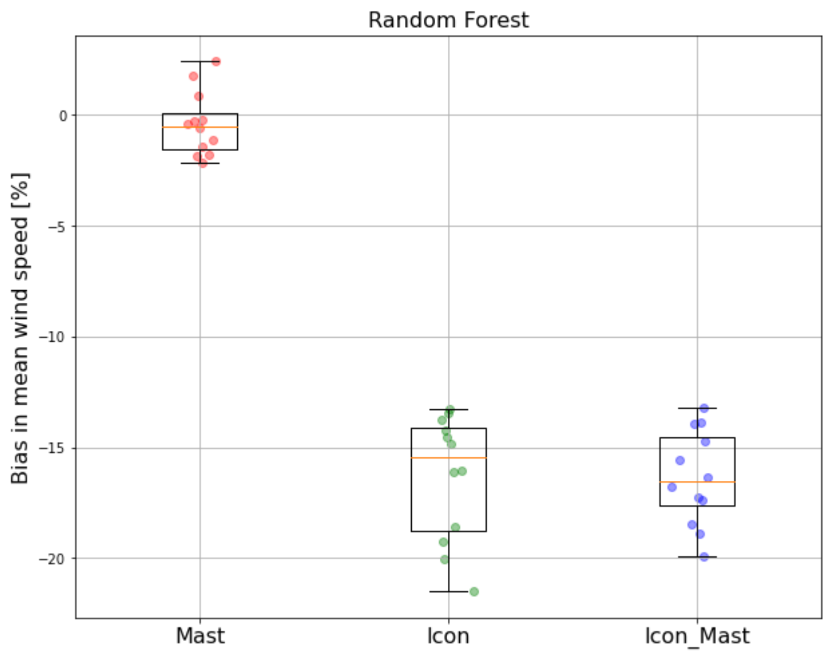

Figure 5.14: Bias in mean wind speed when the random forest model is used

In Figure 5.15, bias in mean wind speed can be observed for random forest model with selected features. The mast and icon data are performing better when compared to icon+ mast data. In contrast to random forest model, performance of icon data is improved. Hence, using selective features adds benefits when using the icon data.

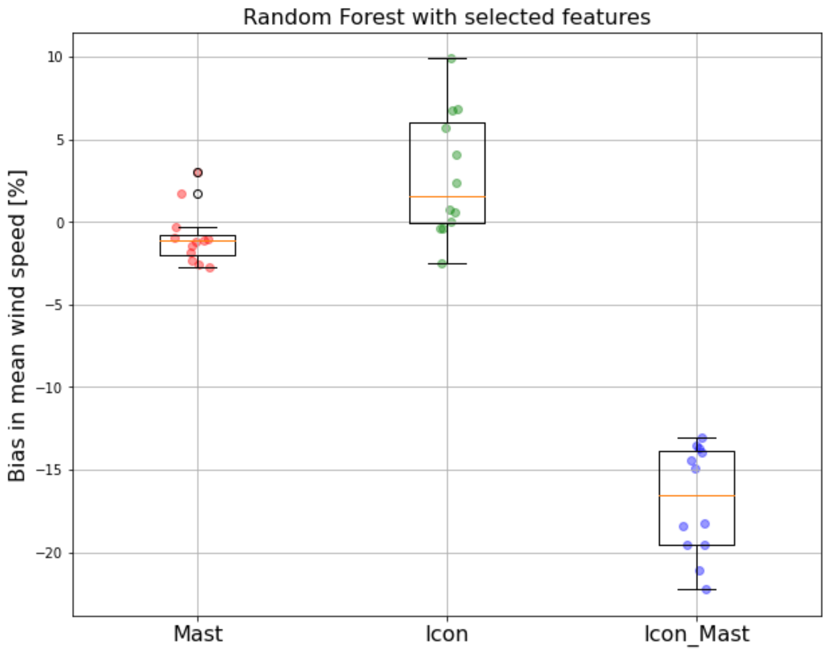

Figure 5.15: Bias in mean wind speed when random forest model with selected features is used

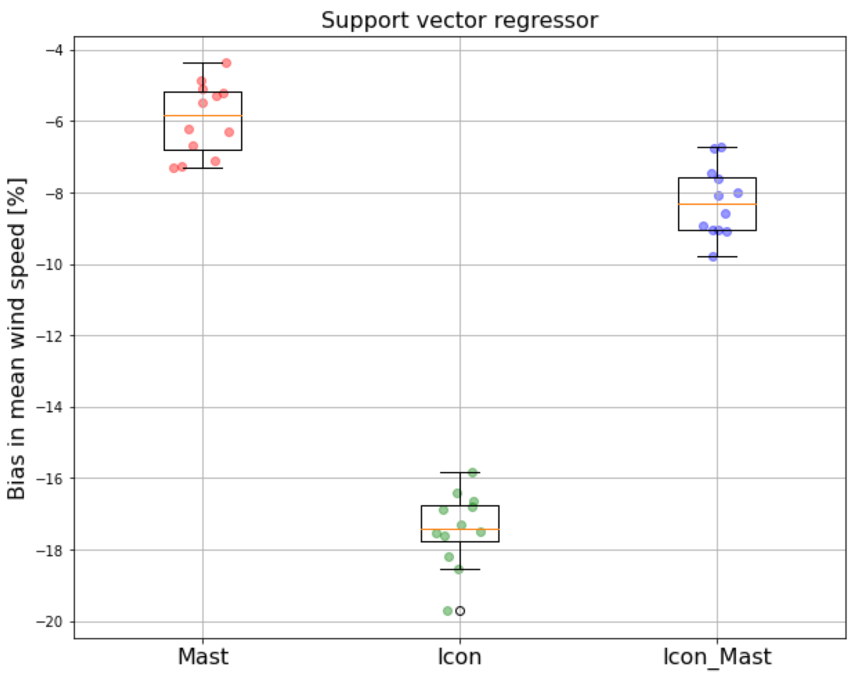

Figure 5.16: Bias in mean wind speed when support vector regressor model is used

In Figure 5.16, bias in mean wind speed can be observed for the support vector regressor model. There is a systematic negative bias throughout and using the mast data gives better results.

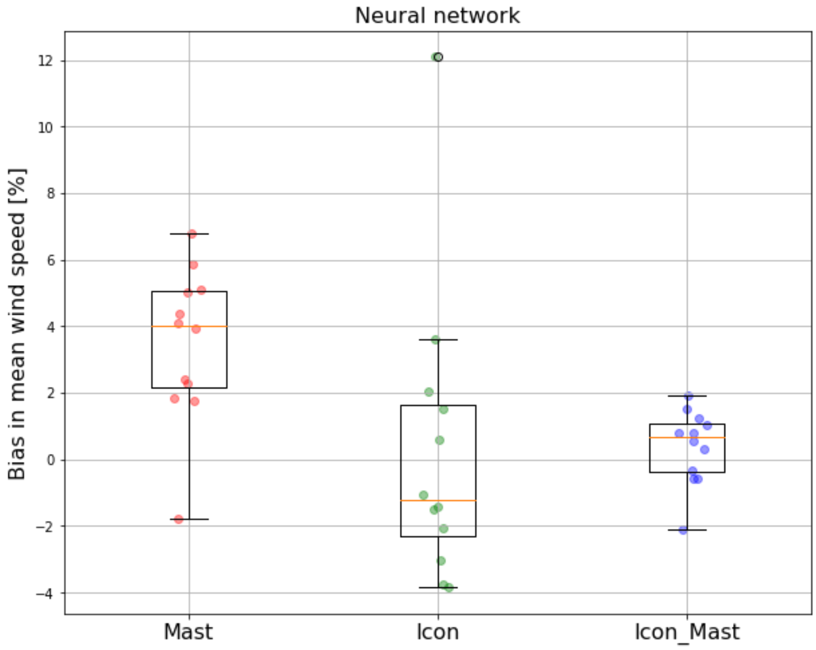

Figure 5.17: Bias in mean wind speed when neural network model is used

Bias in mean wind speed when using the neural network can be observed in Figure 5.17. By comparing all the models, linear regression, random forest, and neural network for met mast data having lower values (nearer to zero). In contrast, linear regression and random forest for icon+ mast data show negative values. Therefore, the model's random forest and linear regression are best for estimating the wind speeds using the met mast data for the bias metric.

5.4 Plots for actual vs predicted values.

In this section, model performances by using met mast data can be observed by comparing the plots for true values and predicted values. In Figure 5.18 plots for the linear regression model can be seen. The non linear change in the wind speed data cannot be catched by linear regression model, hence the predicted values are not matching with the true values.

In Figure 5.19 plots for the random forest model can be seen, unlike the linear regression model the random forest model able to capture trend in the wind speeds.

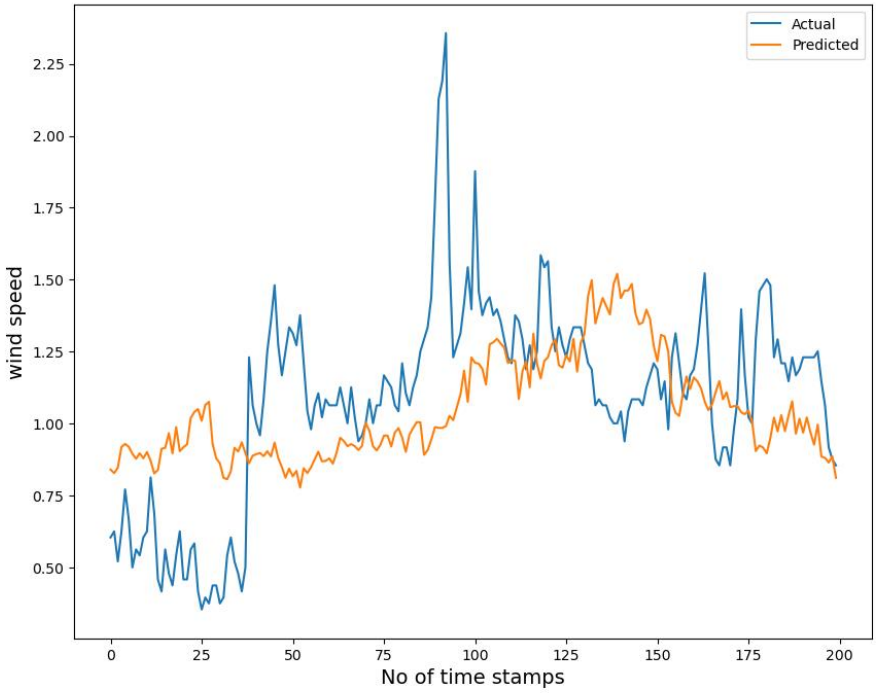

Figure 5.18: Comparison of true and predicted values for linear regression model

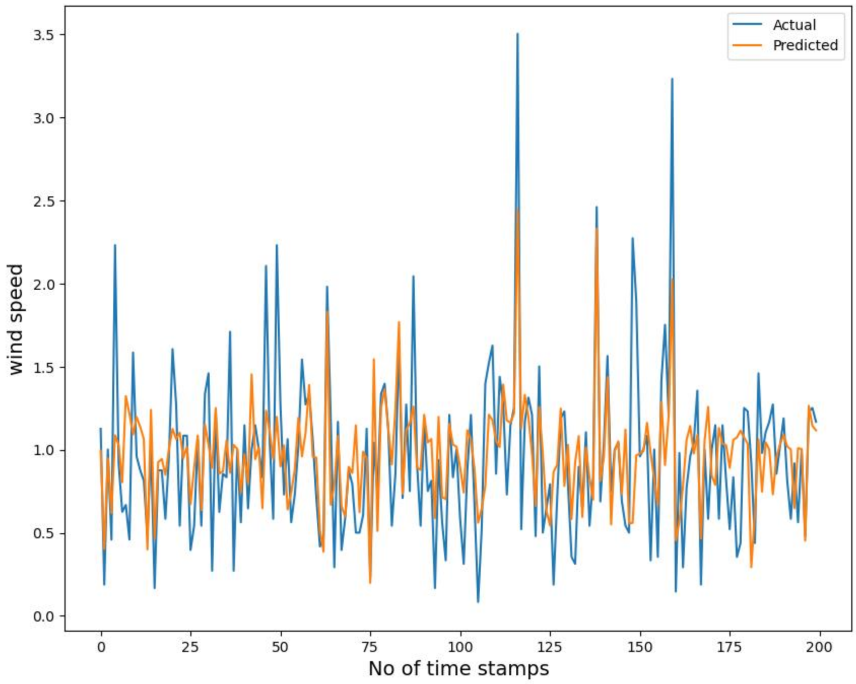

Figure 5.19: Comparison of true and predicted values for random forest model

Similarly the comparison of the predicted values for other models i.e., random forest with selected features, support vector regressor, and neural network can be seen in fig 5.20, fig 5.21, fig 5.22

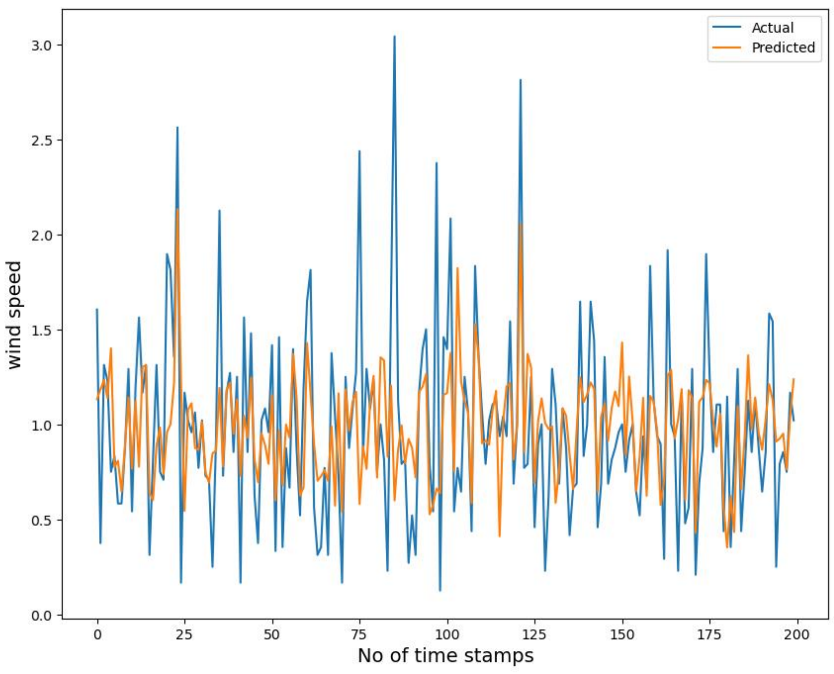

Figure 5.20: Comparison of true and predicted values for random forest model with selected features

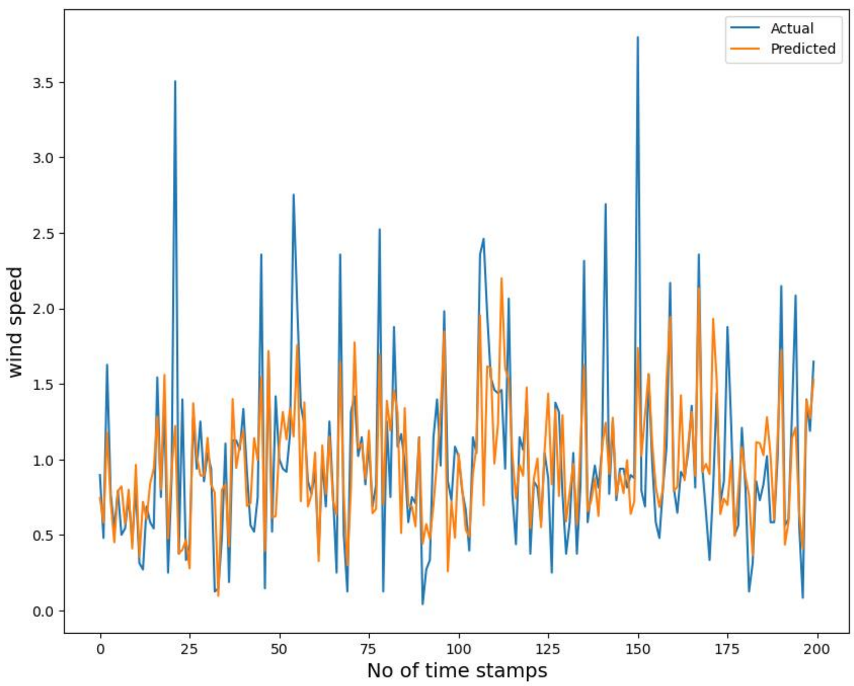

Figure 5.21: Comparison of true and predicted values for support vector regressor model

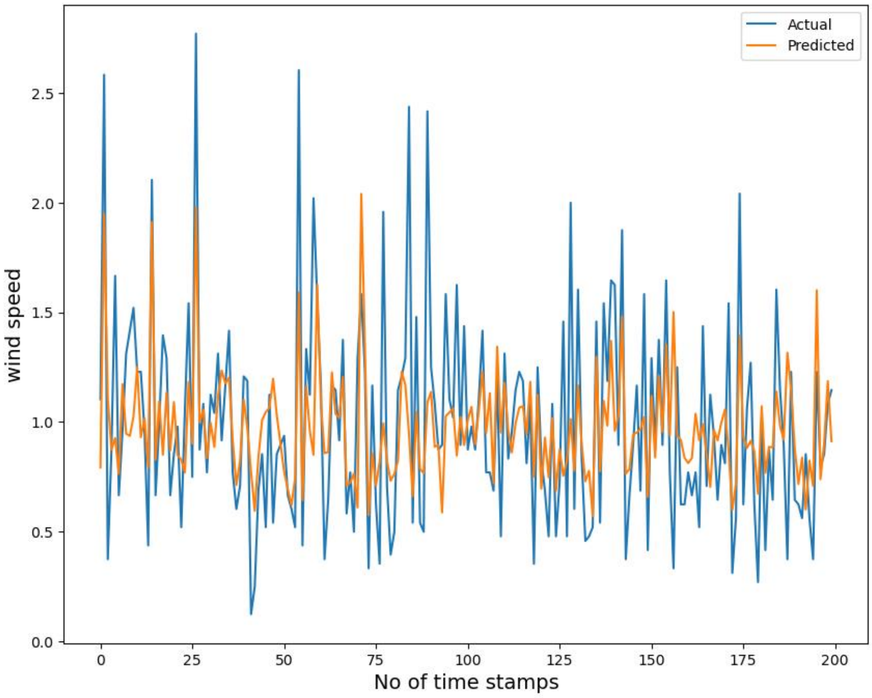

Figure 5.22: Comparison of true and predicted values for neural network

6. Summary

Based on the error scores, i.e., RMSE values, bias in variance, and bias in mean wind speed of different models against different data sets were discussed in chapter 5. It is to be concluded that the results highly depend on the combination of data set and machine learning model. For example, the model generally neural network performs better than model linear regression when using the Mast+ ICON-D2 data set

Figure 6.1. When using the Mast data set alone, however, model linear regression generally performs similar to neural network model. The overall lowest RMSE errors can be achieved by using neural network model with Mast+ ICON-D2 data.

RMSE (m/s)		Mean wind speed for 12 turbines					
	Models		Linear regression	Random forest	Random forest selected features	SVR	Neural network
	Data set	Mast	2	1,97	1,99	2,13	1,85
		ICON-D2	4,57	2,37	2,27	2,61	2,98
		Mast + ICON	21,62	2,43	2,49	2,16	1,68
Bias in variance %							
	Models		Linear regression	Random forest	Random forest selected features	SVR	Neural network
	Data set	Mast	-67,64	-65	-63,39	-55,13	-66,6
		ICON-D2	-9,12	-75,6	-51,61	-70,51	-36,53
		Mast + ICON	64,24	-77,04	-85,6	-54,7	-66,04
Bias in mean wind speed %							
	Models		Linear regression	Random forest	Random forest selected features	SVR	Neural network
	Data set	Mast	2,37	-0,4	-0,89	-5,92	3,48
		ICON-D2	3,82	-16,29	2,8	-17,41	0,26
		Mast + ICON	-18,08	-16,36	-16,89	-8,25	0,37

Figure 6.1: Error scores of all the models against different combinations of data

Additionally, it has to be noted that bias in variance and mean wind speed for all models systematically yield too low variances and strong negative biases. As a further research to this work, Performance of the models can be increased by tuning the model, i.e., adjusting the hyper parameters and building hybrid models. Thereafter power of a wind turbine by using the wind speeds of a turbine can be predicted by implementing the same techniques as in this project.

Appendix

A. Error scores for different models

The following Table A.1 shows the error scores for all the turbines and distance from met mast to the corresponding turbine for a linear regression model using the met mast data.

Table A.1: Error scores of linear regression model

Turbine no	MSE (m2/s2)	MAE (m/s)	RMSE (m/s)	variance	Bias	r2_score	Distance m
Turbine 5	3.249	1.238	1.803	-66.998	1.173	49.155	555.21
Turbine 3	3.869	1.424	1.967	-67.598	3.735	48.599	606.65
Turbine 2	3.214	1.266	1.793	-68.158	2.892	46.48	1079.68
Turbine 7	3.018	1.249	1.737	-70.011	0.627	49.258	1264.98
Turbine15	2.948	1.236	1.717	-68.5	3.849	45.418	1729.38
Turbine11	3.295	1.31	1.815	-67.42	1.958	46.644	1855.71
Turbine16	3.37	1.259	1.836	-67.505	-0.482	47.205	2118.87
Turbine10	3.477	1.33	1.865	-64.361	3.397	45.055	2163.94
Turbine12	3.359	1.327	1.833	-67.923	4.505	44.26	2317.55
Turbine 9	3.041	1.203	1.744	-68.185	1.657	46.52	2375.53
Turbine14	3.522	1.365	1.877	-69.38	0.719	47.847	2461.98
Turbine17	3.363	1.31	1.834	-65.699	4.437	44.341	2472.32

The following Table A.2: Error scores of the random forest model shows the error scores for all the turbines and distance from met mast to the corresponding turbine for a random forest model using the met mast data.

Table A.2: Error scores of the random forest model

Turbine no	MSE (m2/s2)	MAE (m/s)	RMSE (m/s)	variance	Bias	r2_score	Distance m
Turbine 5	3.9	1.387	1.975	-66.742	-1.795	38.973	555.21
Turbine 3	4.73	1.533	2.175	-66.921	-0.264	37.163	606.65
Turbine 2	3.859	1.393	1.964	-67.478	-1.127	35.74	1079.68
Turbine 7	3.497	1.359	1.87	-67.144	-0.597	41.198	1264.98
Turbine 15	3.534	1.338	1.88	-65.965	0.844	34.571	1729.38
Turbine 11	3.836	1.419	1.959	-64.367	-2.181	37.874	1855.71
Turbine 16	4.01	1.433	2.002	-64.767	-0.413	37.18	2118.87
Turbine 10	3.904	1.409	1.976	-60.04	-0.216	38.311	2163.94
Turbine 12	3.746	1.399	1.936	-60.786	2.399	37.839	2317.55
Turbine 9	3.645	1.335	1.909	-66.557	-1.838	35.886	2375.53
Turbine 14	4.26	1.503	2.064	-65.347	-1.447	36.917	2461.98
Turbine 17	3.76	1.391	1.939	-63.979	1.744	37.775	2472.32

The following Table A.3 shows the error scores for all the turbines and distance from met mast to the corresponding turbine for a random forest with selected features model using the met mast data.

Table A.3: Error scores of the random forest model with selected features

Turbine no	MSE (m2/s2)	MAE (m/s)	RMSE (m/s)	variance	Bias	r2_score	Distance m
Turbine 5	4.147	1.421	2.036	-65.854	-2.549	35.111	555.21
Turbine 3	4.899	1.56	2.213	-65.701	-1.177	34.924	606.65
Turbine 2	3.871	1.378	1.967	-66.197	-2.346	35.54	1079.68
Turbine 7	3.686	1.387	1.92	-65.938	-0.954	38.026	1264.98
Turbine 15	3.513	1.311	1.874	-65.34	-1.446	34.965	1729.38
Turbine 11	3.837	1.396	1.959	-61.907	-2.711	37.857	1855.71
Turbine 16	4.206	1.46	2.051	-64.45	-1.132	34.114	2118.87
Turbine 10	4.099	1.428	2.025	-57.92	-0.345	35.226	2163.94
Turbine 12	3.714	1.376	1.927	-59.477	3.029	38.38	2317.55
Turbine 9	3.752	1.345	1.937	-64.164	-1.887	34.011	2375.53
Turbine 14	4.403	1.502	2.098	-62.864	-1.019	34.802	2461.98
Turbine 17	3.887	1.416	1.972	-60.973	1.752	35.664	2472.32

The following Table A.4 shows the error scores for all the turbines and distance from met mast to the corresponding turbine for a support vector regressor model using the met mast data.

Table A.4: Error scores of the support vector regressor model

Turbine no	MSE (m2/s2)	MAE (m/s)	RMSE (m/s)	variance	Bias	r2_score	Distance m
Turbine 5	4.817	1.382	2.195	-53.998	-7.122	24.621	555.21
Turbine 3	5.61	1.537	2.369	-51.717	-5.292	25.475	606.65
Turbine 2	4.442	1.361	2.108	-51.307	-4.373	26.033	1079.68
Turbine 7	4.143	1.337	2.035	-59.018	-7.292	30.334	1264.98
Turbine 15	4.096	1.304	2.024	-56.076	-4.865	24.162	1729.38
Turbine 11	4.572	1.408	2.138	-56.997	-6.663	25.969	1855.71
Turbine 16	4.64	1.408	2.154	-58.352	-5.193	27.307	2118.87
Turbine 10	4.654	1.416	2.157	-53.743	-5.461	26.461	2163.94
Turbine 12	4.265	1.331	2.065	-56.115	-6.287	29.238	2317.55
Turbine 9	4.22	1.295	2.054	-55.169	-5.087	25.776	2375.53
Turbine 14	4.913	1.463	2.216	-54.947	-7.27	27.259	2461.98
Turbine 17	4.63	1.403	2.152	-54.217	-6.208	23.365	2472.32

The following Table A.5 shows the error scores for all the turbines and distance m from met mast to the corresponding turbine for neural network model using the met mast data.

Table A.5: Error scores of the neural network model

Turbine no	MSE (m2/s2)	MAE (m/s)	RMSE (m/s)	variance	Bias	r2_score	Distance m
Turbine 5	3.408	1.269	1.846	-65.798	2.419	46.671	555.21
Turbine 3	4.275	1.507	2.068	-68.691	4.109	43.215	606.65
Turbine 2	3.336	1.33	1.827	-69.51	5.118	44.448	1079.68
Turbine 7	3.151	1.299	1.775	-69.286	4.389	47.02	1264.98
Turbine 15	2.954	1.208	1.719	-66.299	1.838	45.318	1729.38
Turbine 11	3.359	1.297	1.833	-63.544	-1.803	45.611	1855.71
Turbine 16	3.394	1.286	1.842	-64.806	2.29	46.829	2118.87
Turbine 10	3.639	1.356	1.908	-60.844	5.033	42.503	2163.94
Turbine 12	3.551	1.399	1.885	-71.849	6.785	41.076	2317.55
Turbine 9	3.089	1.198	1.758	-65.031	1.777	45.667	2375.53
Turbine 14	3.889	1.444	1.972	-68.244	3.931	42.424	2461.98
Turbine 17	3.393	1.313	1.842	-66.223	5.875	43.839	2472.32

B. RMSE in dependence of met mast distance

The following Figure B.1, Figure B. 2, Figure B.3, Figure B.4 indicates the change in RMSE of individual turbines in dependence of distance between met mast tower and the corresponding turbine.

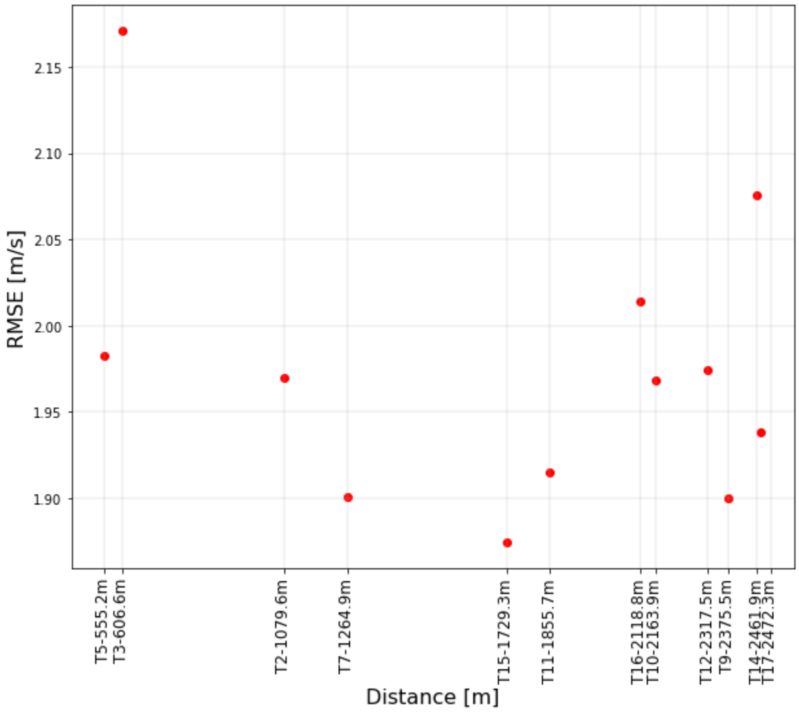

Figure B.1: RMSE of Random forest model.

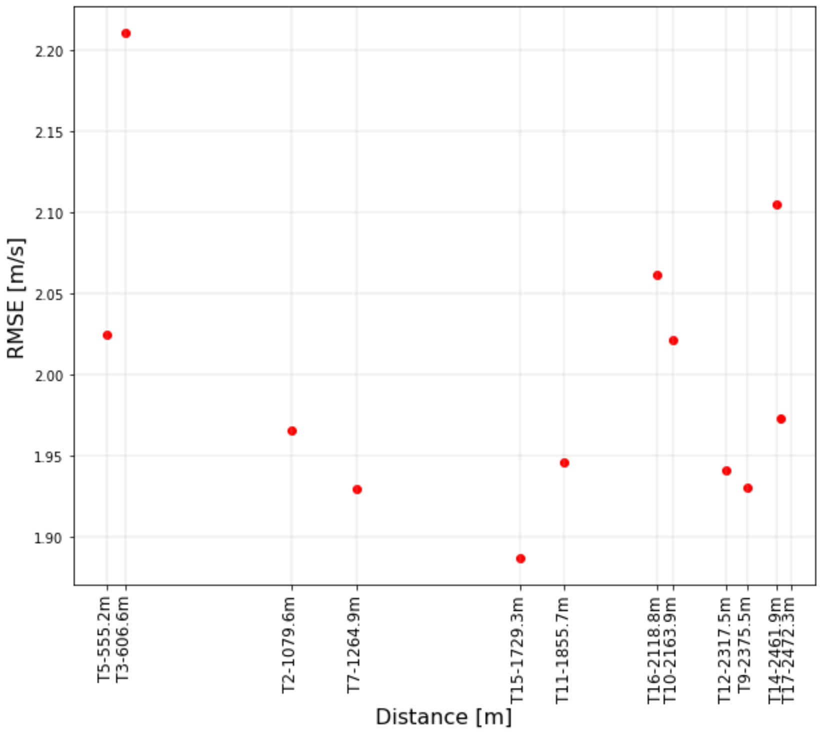

Figure B. 2: RMSE of Random forest model with selected features.

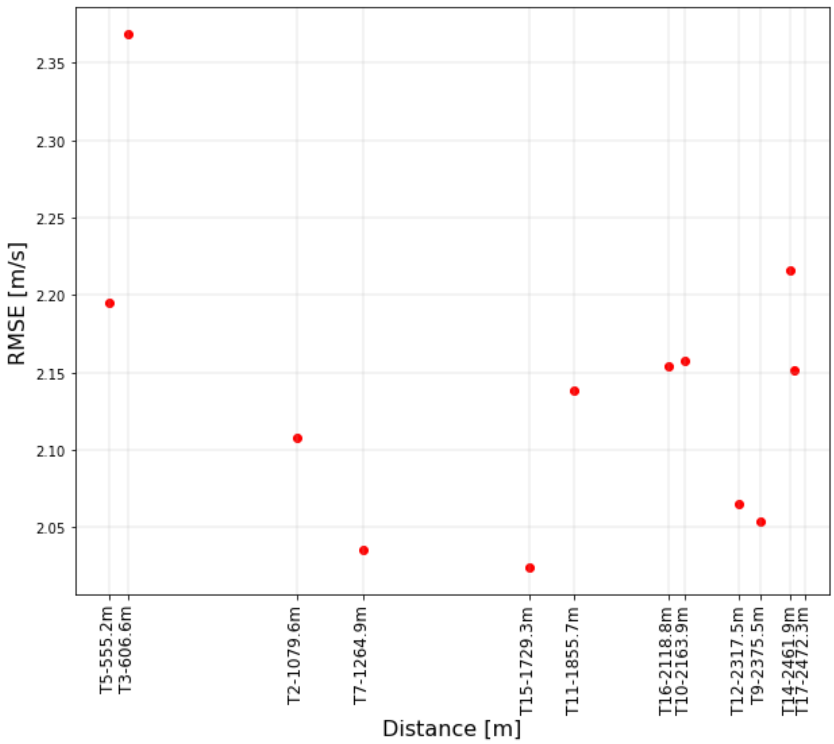

Figure B.3: RMSE of Support vector regressor model.

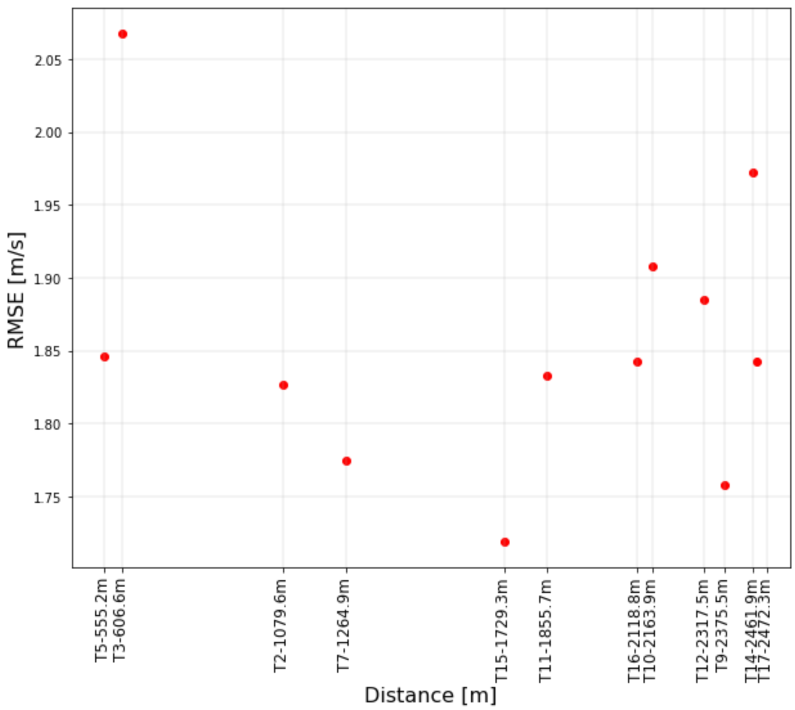

Figure B.4: RMSE of Neural network model.

C. MAE of models

The following Figure C.1 to Figure C.5 indicates the change in mean absolute error (mae) of individual turbines in dependence of distance between met mast tower and the corresponding turbine.

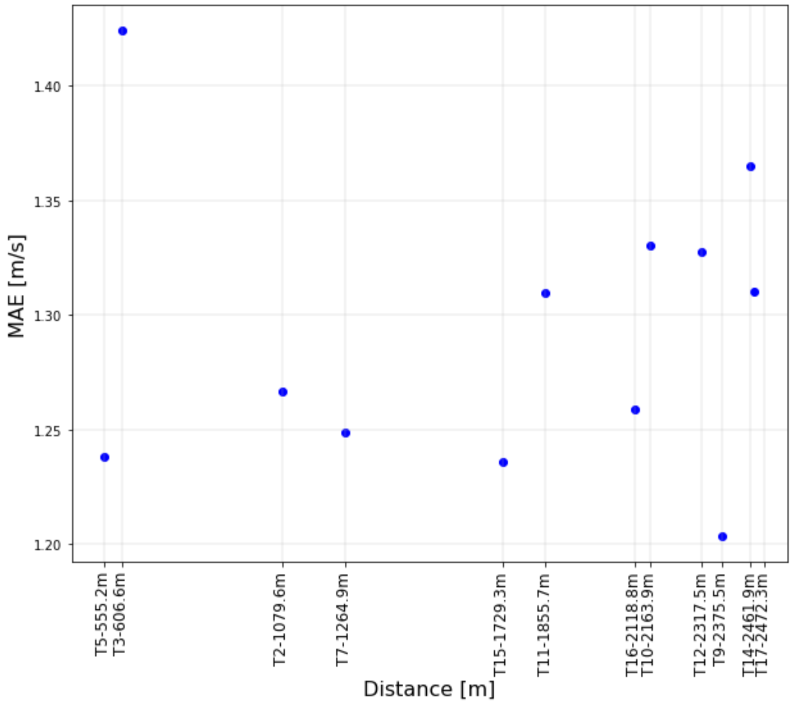

Figure C.1: MAE of linear regression model.

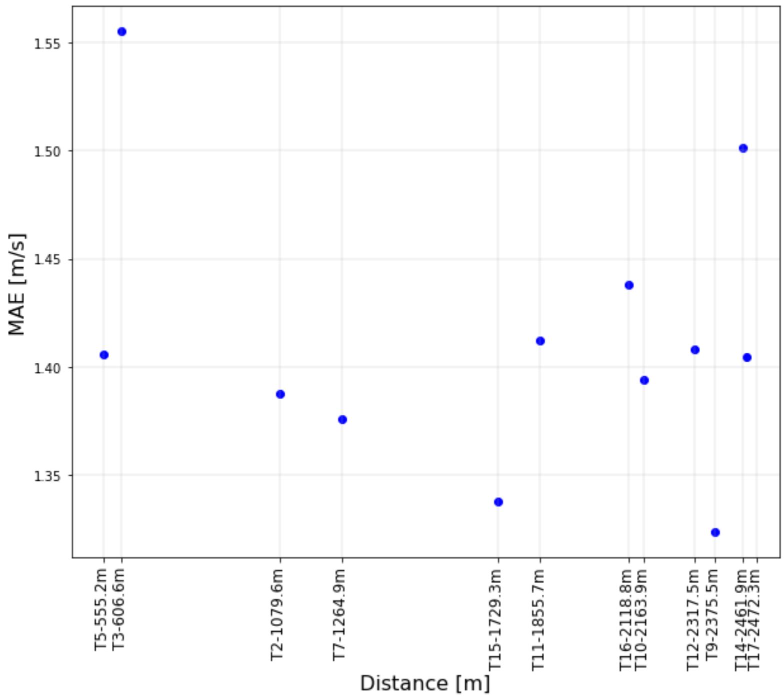

Figure C.2: MAE of random forest model.

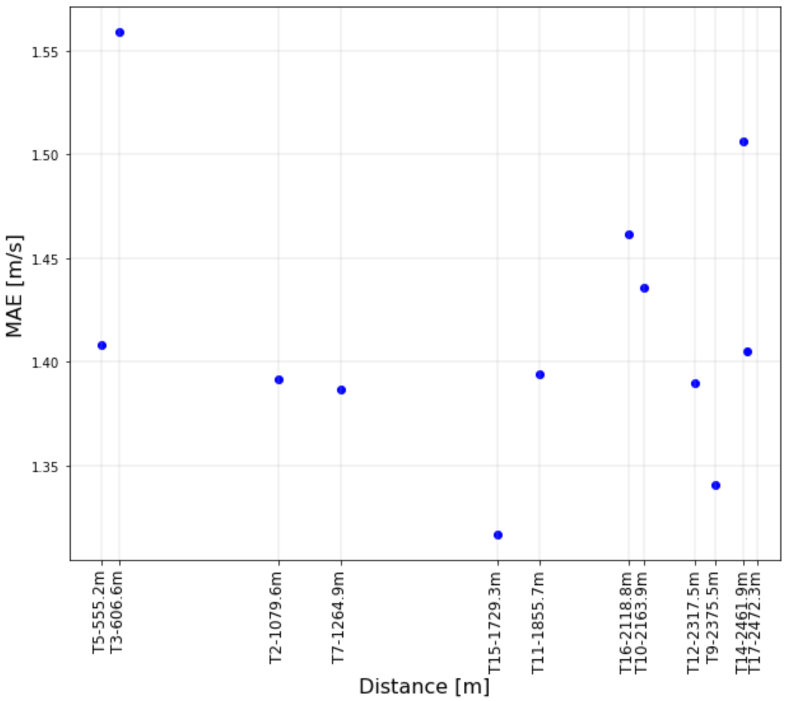

Figure C.3: MAE of random forest model with selected features.

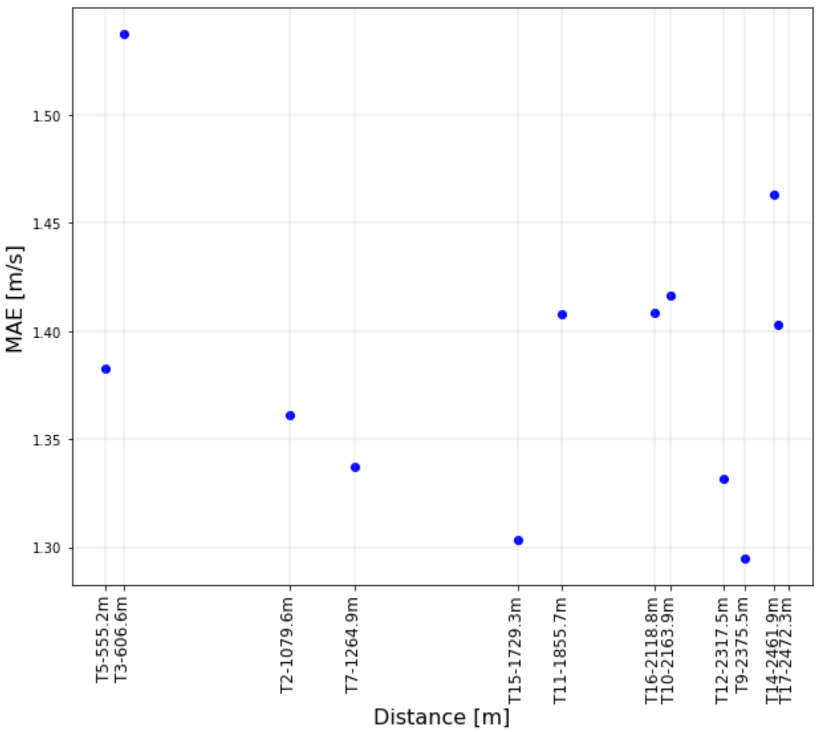

Figure C 4: MAE of support vector regressor model.

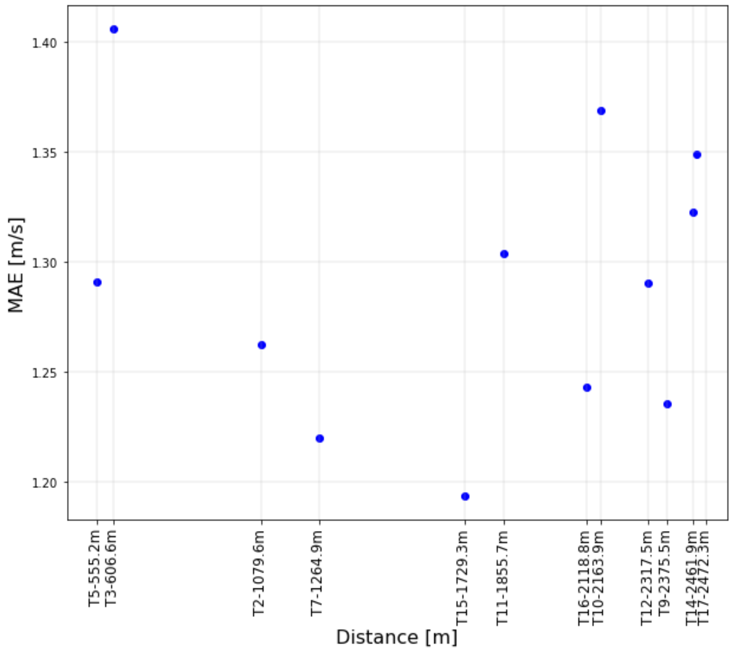

Figure C.5: MAE of Neural network model.

D. Code for the Linear regression model

Below is the code to preprocess the data, train the model and predicting the results plotting the errors.

```
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score

%matplotlib inline
from pylab import rcParams
rcParams['figure.figsize'] = 10,8

class Messmast:

    def __init__(self):

        self.adress = 'D:\\files\\test\\Messmast.csv'
        self.df = pd.read_csv(self.adress)

    def get_totaldata(self):
```

```
        return self.df

    def get_formatdata(self):

        self.df = self.df.drop(columns='Unnamed: 0',
axis=1)#,inplace=True)
        #self.data = self.data.set_index('Date/time')#,
inplace = True)

        self.df['Date/time'] =
pd.to_datetime(self.df['Date/time'])
        self.df = self.df.set_index(self.df['Date/time'])
        self.df = self.df.drop('Date/time', axis=1)

        self.df = self.df.replace(r'\W', np.NaN, regex=True)



        self.data = self.df.rename(columns={'VZ1-
03210099;wind_speed;Avg':'ws_142', 'V01-
11208098;wind_speed;Avg':'ws_136',
                   'V02-
11208099;wind_speed;Avg':'ws_119','V03-
11208100;wind_speed;Avg':'ws_99',
                   'V04-11208101;wind_speed;Avg':'ws_79','V05-
11208102;wind_speed;Avg':'ws_59',
                   'V06-11208103;wind_speed;Avg':'ws_39','VZ1-
03210099;wind_direction;Avg':'wd_142',
                   'D01-08170370;wind_direction;Avg':'wd_136',
'D02-08170369;wind_direction;Avg':'wd_79',
                   'HT1-236609;humidity;Avg':'H_136', 'HT2-
236616;humidity;Avg':'H_99', 'HT3-236615;humidity;Avg':'H_30',
                   'VZ1-
03210099;temperature;Avg':'Temp_142','HT1-
236609;temperature;Avg':'Temp_136','HT2-
236616;temperature;Avg':'Temp_99',
                   'HT3-
236615;temperature;Avg':'Temp_30','BP1-
150931338;air_pressure;Avg':'Press_16','VZ1-
03210099;wind_speed_vert;Avg':'ws_vert_142'
                    })#, inplace=True)
        self.data=
self.data.loc[:,['ws_142','ws_136','ws_119','ws_99','ws_79','w
s_59','ws_39','ws_vert_142',
                                   'wd_142','wd_136','wd_79',
                                   'H_136', 'H_99', 'H_30',

'Temp_142','Temp_136','Temp_99','Temp_30',
                                   'Press_16']]

        self.data['ws_142'] =
self.data['ws_142'].astype('float')
```

```
        self.data['ws_vert_142'] =
self.data['ws_vert_142'].astype('float')
        self.data['wd_142'] =
self.data['wd_142'].astype('float')
        self.data['Temp_142'] =
self.data['Temp_142'].astype('float')


        return self.data


class Turbine:

    def __init__(self):
        self.df =
pd.read_csv('D:\\files\\turbine_power\\turbine_edited\\combine
d_data.csv')#, index_col='Zeitstempel (UTC)')


    def get_data(self):
        '''Returns all the turbine data '''

        self.df['Zeitstempel (UTC)'] =
pd.to_datetime(self.df['Zeitstempel (UTC)'])
        self.df = self.df.set_index(self.df['Zeitstempel
(UTC)'])
        self.df = self.df.drop('Zeitstempel (UTC)', axis=1)
        return self.df


class Linear_regression:

    def get_data(self,feature_data, label_data):
        df = pd.merge(feature_data, label_data,
left_index=True, right_index=True)
        df = df.dropna()

        return df



    def processing(self, feature_data, label_data):

        df = pd.merge(feature_data, label_data,
left_index=True, right_index=True)
        df = df.dropna()
        features = df.iloc[:,:-1].values
        label = df.iloc[:,-1].values

        x_train, x_test, y_train, y_test =
train_test_split(features, label, test_size=0.20)
```

```
        return x_train,x_test,y_train, y_test

    def trainmodel(self, x_train, y_train):

        reggresion = LinearRegression()
        reggresion.fit(x_train, y_train)

        return reggresion

    def plot(self,y_test, predictions):

        plt.plot(y_test, color='g', label='Actual')
        plt.plot(predictions, color='b', label='Predicted')
        plt.legend()
        plt.show()

    def mse(self, y_test, predictions):
        return mean_squared_error(y_test, predictions)

    def mae(self, y_test, predictions):
        return mean_absolute_error(y_test, predictions)

    def variance(self, y_test, predictions):
        var = (np.var(predictions) -
np.var(y_test))/np.var(y_test)
        return var

    def bias(self, y_test, predictions):
        bias =  (np.mean(predictions) -
np.mean(y_test))/np.mean(y_test)
        return bias



mast_df = Messmast().get_formatdata()
turbine_df = Turbine().get_data()

mast_df = Messmast().get_formatdata()
lr = Linear_regression()
data_2 = lr.get_data(mast_df, turbine_df[['ws_2']])
data_3 = lr.get_data(mast_df, turbine_df[['ws_3']])
data_5 = lr.get_data(mast_df, turbine_df[['ws_5']])
data_7 = lr.get_data(mast_df, turbine_df[['ws_7']])
data_9 = lr.get_data(mast_df, turbine_df[['ws_9']])
data_10 = lr.get_data(mast_df, turbine_df[['ws_10']])
data_11 = lr.get_data(mast_df, turbine_df[['ws_11']])
data_12 = lr.get_data(mast_df, turbine_df[['ws_12']])
data_14 = lr.get_data(mast_df, turbine_df[['ws_14']])
data_15 = lr.get_data(mast_df, turbine_df[['ws_15']])
data_16 = lr.get_data(mast_df, turbine_df[['ws_16']])
data_17 = lr.get_data(mast_df, turbine_df[['ws_17']])
```

```
## 2021-04-12 11:40:00 ---> 2021-11-26 00:50:00
training_2 = data_2.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_3 = data_3.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_5 = data_5.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_7 = data_7.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_9 = data_9.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_10 = data_10.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_11 = data_11.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_12 = data_12.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_14 = data_14.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_15 = data_15.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_16 = data_16.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
training_17 = data_17.loc['2021-04-12 11:40:00':'2021-10-11
17:40:00']
testing_2 = data_2.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_3 = data_3.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_5 = data_5.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_7 = data_7.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_9 = data_9.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_10 = data_10.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_11 = data_11.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_12 = data_12.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_14 = data_14.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_15 = data_15.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_16 = data_16.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']
testing_17 = data_17.loc['2021-10-11 17:40:00':'2021-11-26
00:50:00']

x_train_2 = training_2.iloc[:,:-1]
```

```
x_train_3 = training_3.iloc[:,:-1]
x_train_5 = training_5.iloc[:,:-1]
x_train_7 = training_7.iloc[:,:-1]
x_train_9 = training_9.iloc[:,:-1]
x_train_10 = training_10.iloc[:,:-1]
x_train_11 = training_11.iloc[:,:-1]
x_train_12 = training_12.iloc[:,:-1]
x_train_14 = training_14.iloc[:,:-1]
x_train_15 = training_15.iloc[:,:-1]
x_train_16 = training_16.iloc[:,:-1]
x_train_17 = training_17.iloc[:,:-1]

y_train_2 = training_2.iloc[:,-1]
y_train_3 = training_3.iloc[:,-1]
y_train_5 = training_5.iloc[:,-1]
y_train_7 = training_7.iloc[:,-1]
y_train_9 = training_9.iloc[:,-1]
y_train_10 = training_10.iloc[:,-1]
y_train_11 = training_11.iloc[:,-1]
y_train_12 = training_12.iloc[:,-1]
y_train_14 = training_14.iloc[:,-1]
y_train_15 = training_15.iloc[:,-1]
y_train_16 = training_16.iloc[:,-1]
y_train_17 = training_17.iloc[:,-1]

x_test_2 = testing_2.iloc[:,:-1]
x_test_3 = testing_3.iloc[:,:-1]
x_test_5 = testing_5.iloc[:,:-1]
x_test_7 = testing_7.iloc[:,:-1]
x_test_9 = testing_9.iloc[:,:-1]
x_test_10 = testing_10.iloc[:,:-1]
x_test_11 = testing_11.iloc[:,:-1]
x_test_12 = testing_12.iloc[:,:-1]
x_test_14 = testing_14.iloc[:,:-1]
x_test_15 = testing_15.iloc[:,:-1]
x_test_16 = testing_16.iloc[:,:-1]
x_test_17 = testing_17.iloc[:,:-1]
y_test_2 = testing_2.iloc[:,-1]
y_test_3 = testing_3.iloc[:,-1]
y_test_5 = testing_5.iloc[:,-1]
y_test_7 = testing_7.iloc[:,-1]
y_test_9 = testing_9.iloc[:,-1]
y_test_10 = testing_10.iloc[:,-1]
y_test_11 = testing_11.iloc[:,-1]
y_test_12 = testing_12.iloc[:,-1]
y_test_14 = testing_14.iloc[:,-1]
y_test_15 = testing_15.iloc[:,-1]
y_test_16 = testing_16.iloc[:,-1]
y_test_17 = testing_17.iloc[:,-1]

from sklearn.utils import shuffle
x_train_2, y_train_2 = shuffle(x_train_2, y_train_2)
```

```
x_train_3, y_train_3 = shuffle(x_train_3, y_train_3)
x_train_5, y_train_5 = shuffle(x_train_5, y_train_5)
x_train_7, y_train_7 = shuffle(x_train_7, y_train_7)
x_train_9, y_train_9 = shuffle(x_train_9, y_train_9)
x_train_10, y_train_10 = shuffle(x_train_10, y_train_10)
x_train_11, y_train_11 = shuffle(x_train_11, y_train_11)
x_train_12, y_train_12 = shuffle(x_train_12, y_train_12)
x_train_14, y_train_14 = shuffle(x_train_14, y_train_14)
x_train_15, y_train_15 = shuffle(x_train_15, y_train_15)
x_train_16, y_train_16 = shuffle(x_train_17, y_train_17)
x_train_17, y_train_17 = shuffle(x_train_17, y_train_17)

model_2 = lr.trainmodel(x_train_2, y_train_2)
model_3 = lr.trainmodel(x_train_3, y_train_3)
model_5 = lr.trainmodel(x_train_5, y_train_5)
model_7 = lr.trainmodel(x_train_7, y_train_7)
model_9 = lr.trainmodel(x_train_9, y_train_9)
model_10 = lr.trainmodel(x_train_10, y_train_10)
model_11 = lr.trainmodel(x_train_11, y_train_11)
model_12 = lr.trainmodel(x_train_12, y_train_12)
model_14 = lr.trainmodel(x_train_14, y_train_14)
model_15 = lr.trainmodel(x_train_15, y_train_15)
model_16 = lr.trainmodel(x_train_16, y_train_16)
model_17 = lr.trainmodel(x_train_17, y_train_17)

predictions_2 = model_2.predict(x_test_2)
predictions_3 = model_3.predict(x_test_3)
predictions_5 = model_5.predict(x_test_5)
predictions_7 = model_7.predict(x_test_7)
predictions_9 = model_9.predict(x_test_9)
predictions_10 = model_10.predict(x_test_10)
predictions_11 = model_11.predict(x_test_11)
predictions_12 = model_12.predict(x_test_12)
predictions_14 = model_14.predict(x_test_14)
predictions_15 = model_15.predict(x_test_15)
predictions_16 = model_16.predict(x_test_16)
predictions_17 = model_17.predict(x_test_17)

mse_2 = lr.mse(y_test_2, predictions_2)
mse_3 = lr.mse(y_test_3, predictions_3)
mse_5 = lr.mse(y_test_5, predictions_5)
mse_7 = lr.mse(y_test_7, predictions_7)
mse_9 = lr.mse(y_test_9, predictions_9)
mse_10 = lr.mse(y_test_10, predictions_10)
mse_11 = lr.mse(y_test_11, predictions_11)
mse_12 = lr.mse(y_test_12, predictions_12)
mse_14 = lr.mse(y_test_14, predictions_14)
mse_15 = lr.mse(y_test_15, predictions_15)
mse_16 = lr.mse(y_test_16, predictions_16)
mse_17 = lr.mse(y_test_17, predictions_17)

mae_2 = lr.mae(y_test_2, predictions_2)
```

```
mae_3 = lr.mae(y_test_3, predictions_3)
mae_5 = lr.mae(y_test_5, predictions_5)
mae_7 = lr.mae(y_test_7, predictions_7)
mae_9 = lr.mae(y_test_9, predictions_9)
mae_10 = lr.mae(y_test_10, predictions_10)
mae_11 = lr.mae(y_test_11, predictions_11)
mae_12 = lr.mae(y_test_12, predictions_12)
mae_14 = lr.mae(y_test_14, predictions_14)
mae_15 = lr.mae(y_test_15, predictions_15)
mae_16 = lr.mae(y_test_16, predictions_16)
mae_17 = lr.mae(y_test_17, predictions_17)

def filterr(test,pred):
    new_test = []
    new_pred = []

    zipp = zip(test,pred)
    for test_val, pred_val in zipp:
        if pred_val<40 and test_val>0:
            new_pred.append(pred_val)
            new_test.append(test_val)
    new_pred = np.array(new_pred)
    new_test = np.array(new_test)
    return new_test, new_pred

new_test_2, new_pred_2 = filterr(y_test_2,predictions_2)
new_test_3, new_pred_3 = filterr(y_test_3,predictions_3)
new_test_5, new_pred_5 = filterr(y_test_5,predictions_5)
new_test_7, new_pred_7 = filterr(y_test_7,predictions_7)
new_test_9, new_pred_9 = filterr(y_test_9,predictions_9)
new_test_10, new_pred_10 = filterr(y_test_10,predictions_10)
new_test_11, new_pred_11 = filterr(y_test_11,predictions_11)
new_test_12, new_pred_12 = filterr(y_test_12,predictions_12)
new_test_14, new_pred_14 = filterr(y_test_14,predictions_14)
new_test_15, new_pred_15 = filterr(y_test_15,predictions_15)
new_test_16, new_pred_16 = filterr(y_test_16,predictions_16)
new_test_17, new_pred_17 = filterr(y_test_17,predictions_17)

var_2 =  lr.variance(new_test_2, new_pred_2)
var_3 =  lr.variance(new_test_3, new_pred_3)
var_5 =  lr.variance(new_test_5, new_pred_5)
var_7 =  lr.variance(new_test_7, new_pred_7)
var_9 =  lr.variance(new_test_9, new_pred_9)
var_10 = lr.variance(new_test_10, new_pred_10)
var_11 = lr.variance(new_test_11, new_pred_11)
var_12 = lr.variance(new_test_12, new_pred_12)
var_14 = lr.variance(new_test_14, new_pred_14)
var_15 = lr.variance(new_test_15, new_pred_15)
var_16 = lr.variance(new_test_16, new_pred_16)
var_17 = lr.variance(new_test_17, new_pred_17)

bias_2 = lr.bias(y_test_2, predictions_2)
```

```
bias_3 = lr.bias(y_test_3, predictions_3)
bias_5 = lr.bias(y_test_5, predictions_5)
bias_7 = lr.bias(y_test_7, predictions_7)
bias_9 = lr.bias(y_test_9, predictions_9)
bias_10 = lr.bias(y_test_10, predictions_10)
bias_11 =lr.bias(y_test_11, predictions_11)
bias_12 = lr.bias(y_test_12, predictions_12)
bias_14 = lr.bias(y_test_14, predictions_14)
bias_15 = lr.bias(y_test_15, predictions_15)
bias_16 = lr.bias(y_test_16, predictions_16)
bias_17 = lr.bias(y_test_17, predictions_17)

r2_2 = r2_score(y_test_2, predictions_2)
r2_3 = r2_score(y_test_3, predictions_3)
r2_5 = r2_score(y_test_5, predictions_5)
r2_7 = r2_score(y_test_7, predictions_7)
r2_9 = r2_score(y_test_9, predictions_9)
r2_10 = r2_score(y_test_10, predictions_10)
r2_11 = r2_score(y_test_11, predictions_11)
r2_12 = r2_score(y_test_12, predictions_12)
r2_14 = r2_score(y_test_14, predictions_14)
r2_15 = r2_score(y_test_15, predictions_15)
r2_16= r2_score(y_test_16, predictions_16)
r2_17 = r2_score(y_test_17, predictions_17)

distances = [1079.68, 606.65, 555.21, 1264.98, 2375.53,
2163.94, 1855.71, 2317.55, 2461.98, 1729.38, 2118.87,2472.32]

data={'Turbine no':['Turbine 2','Turbine 3','Turbine
5','Turbine 7','Turbine 9',
                   'Turbine 10','Turbine 11','Turbine
12','Turbine 14','Turbine 15',
                   'Turbine 16','Turbine 17'],
     'mean_squrd_err (m2/s2)':[mse_2, mse_3, mse_5, mse_7,
mse_9, mse_10, mse_11, mse_12, mse_14, mse_15,
                      mse_16, mse_17],
     'mean_absolute_err (m/s)':[mae_2, mae_3, mae_5, mae_7,
mae_9, mae_10, mae_11, mae_12, mae_14, mae_15,
                      mae_16, mae_17],
     'variance':[var_2, var_3, var_5, var_7, var_9, var_10,
var_11, var_12, var_14, var_15,
                      var_16, var_17],
     'bias':[bias_2, bias_3, bias_5, bias_7, bias_9, bias_10,
bias_11, bias_12, bias_14, bias_15,
                      bias_16, bias_17],
     'r2_score':[r2_2, r2_3, r2_5, r2_7, r2_9, r2_10, r2_11,
r2_12, r2_14, r2_15,
                      r2_16, r2_17],
     'Distance m':distances}

errors = pd.DataFrame(data)
```

```
errors['rmse (m/s)'] = np.sqrt(errors['mean_squrd_err
(m2/s2)'])
errors['variance'] = errors['variance']*100
errors['bias'] = errors['bias']*100
errors['r2_score']= errors['r2_score']*100
errors.set_index('Turbine no', inplace=True)
errors
Labels = ['T2-1079.6m','T3-606.6m','T5-555.2m','T7-
1264.9m','T9-2375.5m',
          'T10-2163.9m','T11-1855.7m','T12-2317.5m','T14-
2461.9m','T15-1729.3m',
         'T16-2118.8m','T17-2472.3m']
x_ticks = errors['Distance m']
x_ticks =[1079.68, 606.65, 555.21, 1264.98, 2375.53, 2163.94,
1855.71, 2317.55, 2461.98, 1729.38, 2118.87,2500.32]
minimum_mse =  min(errors['mean_squrd_err (m2/s2)'].values)
plt.scatter(errors['Distance m'], errors['rmse (m/s)'], c='r',
label='RMSE')

minimum_mae =  min(errors['mean_absolute_err (m/s)'].values)

plt.xticks(ticks=x_ticks, labels=Labels,rotation=90,
fontsize=12)
plt.xlabel('Distance [m]', fontsize=16)
plt.ylabel('RMSE [m/s]', fontsize=16)

plt.grid(linewidth=0.3)
plt.show()

minimum_var =  min(errors['variance'].values)
plt.scatter(errors['Distance m'], errors['variance'], c='r',
label='Variance')

minimum_bias =  min(errors['bias'].values)
plt.scatter(errors['Distance m'], errors['bias'],c='b',
label='Bias')


plt.xticks(ticks=x_ticks, labels=Labels,rotation=90,
fontsize=12)
plt.xlabel('Distance [m]', fontsize=16)
plt.ylabel('Errors [%]', fontsize=16)
plt.legend()
plt.grid(linewidth=0.3)
plt.show()

max_score =  max(errors['r2_score'].values)
plt.scatter(errors['Distance m'], errors['r2_score'],c='b',
label='R2_Score')

plt.xticks(ticks=x_ticks, labels=Labels,rotation=90,
fontsize=12)
```

```
plt.xlabel('Distance [m]', fontsize=16)
plt.ylabel('Score', fontsize=16)
plt.legend()
plt.grid(linewidth=0.3)
plt.show()

fig, ax = plt.subplots()

minimum_var =  min(errors['variance'].values)

ax.scatter(errors['Distance m'], errors['variance'], c='r',
label='Variance')

plt.xticks(ticks=x_ticks, labels=Labels,rotation=90,
fontsize=12)
ax.set_xlabel('Distance [m]', fontsize=16)
ax.set_ylabel('Variance error [%]', fontsize=16)
ax.legend(loc='best')
ax.grid(linewidth=0.3)


ax2 = ax.twinx()

minimum_bias =  min(errors['bias'].values)
ax2.scatter(errors['Distance m'], errors['bias'],c='b',
label='Bias')

ax2.set_ylabel('Bias error [%]', fontsize=16)
ax2.legend(loc=9)

ax2.grid(linewidth=0.3)

fig, ax = plt.subplots()

ax.scatter(errors['Distance m'], errors['mean_squrd_err
(m2/s2)'], c='r', label='MSE')
plt.xticks(ticks=x_ticks, labels=Labels,rotation=90,
fontsize=12)
ax.set_xlabel('Distance [m]', fontsize=16)
ax.set_ylabel('MSE [m2/s2]', fontsize=16)
ax.legend(loc='best')
ax.grid(linewidth=0.3)

ax2 = ax.twinx()

ax2.scatter(errors['Distance m'], errors['mean_absolute_err
(m/s)'],c='b', label='MAE')

ax2.set_ylabel('MAE [m/s]', fontsize=16)
ax2.legend(loc=9)

ax2.grid(linewidth=0.3)
```

```
#errors.to_csv('D:\\models\\metmast_new_20\\linear_regression.
csv')
plt.scatter(errors['Distance m'], errors['mean_absolute_err
(m/s)'],c='b', label='MAE')

plt.xticks(ticks=x_ticks, labels=Labels,rotation=90,
fontsize=12)
plt.xlabel('Distance [m]', fontsize=16)
plt.ylabel('MAE [m/s]', fontsize=16)
plt.grid(linewidth=0.3)
plt.show()
```

E. Code for Random forest model

Below is the code to preprorocess the data, train the model and predicting the results.

```
class RandomForest(Messmast, Turbine):

    def get_data(self,feature_data, label_data):
        df = pd.merge(feature_data, label_data, left_index=True,
right_index=True)
        df = df.dropna()

        return df


    def processing(self, feature_data, label_data):

        df = pd.merge(feature_data, label_data, left_index=True,
right_index=True)
        df = df.dropna()
        features = df.iloc[:,:-1].values
        label = df.iloc[:,-1].values

        x_train, x_test, y_train, y_test = train_test_split(features,
label, test_size=0.20)

        return x_train,x_test,y_train, y_test

    def featuredprocessing(self, feature_data, label_data):
        '''selects the certain features from the total features'''

#          'ws_142', 'ws_136', 'ws_119', 'ws_99', 'ws_79', 'ws_59', 'ws_39',
#         'ws_vert_142', 'wd_142', 'wd_136', 'wd_79', 'H_136', 'H_99',
'H_30',
#         'Temp_142', 'Temp_136', 'Temp_99', 'Temp_30', 'Press_16'

        df = pd.merge(feature_data, label_data, left_index=True,
right_index=True)
        df = df.dropna()
```

```
        parameters = ['ws_142','wd_142','H_136',
                      'Temp_142', 'Temp_30','Press_16']
        features = df.loc[:,parameters].values
        label = df.iloc[:,-1].values

        x_train, x_test, y_train, y_test = train_test_split(features,
label, test_size=0.20)

        return x_train,x_test,y_train, y_test


    def trainmodel(self, x_train, y_train):

        rf = RandomForestRegressor(n_estimators=50)
        model = rf.fit(x_train, y_train)
        return model


    def plot(self,y_test, predictions):

        plt.plot(y_test, color='g', label='Actual')
        plt.plot(predictions, color='b', label='Predicted')
        plt.legend()
        plt.show()

    def mse(self, y_test, predictions):
        return mean_squared_error(y_test, predictions)

    def mae(self, y_test, predictions):
        return mean_absolute_error(y_test, predictions)

    def variance(self, y_test, predictions):
        var = (np.var(predictions) - np.var(y_test))/np.var(y_test)
        return var

    def bias(self, y_test, predictions):
        bias =  (np.mean(predictions) - np.mean(y_test))/np.mean(y_test)
        return bias
```

F. Code for support vector regression model

Below is the code to preprocess the data, train the model and predicting the results.

```
class Svr(RandomForest):

    def get_data(self,feature_data, label_data):
        df = pd.merge(feature_data, label_data, left_index=True,
right_index=True)
        df = df.dropna()

        return df

    def trainmodel(self,x_train, y_train):
```

```
        '''Returns the SVR model by using x_trian, y_train'''

        '''Scaling of the trained and test data'''
        super()


        sc_X = StandardScaler()
        sc_y = StandardScaler()

        y_train_reshape = y_train.reshape(-1,1)
        x_sc = sc_X.fit_transform(x_train)
        y_sc = sc_y.fit_transform(y_train_reshape)

        ## Building the Model

        regressor = SVR(kernel='rbf')
        regressor.fit(x_sc, y_sc)

        return regressor


    def prediction(self, model, x_test, y_test):

        '''This method returns the predicted value'''
        '''scaling the x_test data to predict the target'''
        super()
        sc_X = StandardScaler()
        sc_y = StandardScaler()

        x_test_sc = sc_X.fit_transform(x_test)
        y_test_sc = sc_y.fit_transform(y_test.reshape(-1,1))

        y_pred = model.predict(x_test_sc)

        inv_ypred = sc_y.inverse_transform(y_pred.reshape(-1,1))

        return inv_ypred
```

G. Code for Neural network model

Below is the code to preprocess the data, train the model and predicting the results plotting the errors.

```
class Neuralnet():

    def get_data(self,feature_data, label_data):
        df = pd.merge(feature_data, label_data, left_index=True,
right_index=True)
        df = df.dropna()

        return df

    def processing(self, feature_data, label_data):
        '''Splits the data into Training and testing data'''
        super()
```

```
        df = pd.merge(feature_data, label_data, left_index=True,
right_index=True)
        df = df.dropna()

        def norm(df):
            return (df - df.mean())/df.std()

        norm_df = norm(df)

        train_data = norm_df.loc['2021-04-12 11:40:00':'2021-10-12
04:00:00']
        test_data = norm_df.loc['2021-10-12 04:00:00':'2021-11-26
00:50:00']

        train_data, test_data = train_test_split(norm_df, test_size=0.2)

        return train_data, test_data



        return train_data, test_data

    def get_mean(self, feature_data, label_data):
        df = pd.merge(feature_data, label_data, left_index=True,
right_index=True)
        df = df.dropna()
        mean = df.mean()
        std = df.std()

        return mean, std



    def buildmodel(self, train_data, test_data):
        '''Builds the neural network structure and returns the model'''

#         def norm(df):

#             return (df - df.mean())/df.std()

#         train_data = norm(train_data)
#         test_data = norm(test_data)



        degrees = ['dir_10m','dir_38m', 'dir_78m', 'dir_127m', 'dir_184m']

        feature_columns = []

        resolution_in_degrees = 6

        for i in train_data.columns[:-1]:

            if i not in degrees:

                #Represent ws11, ws12, ws10 as a floating point value.
                ws = tf.feature_column.numeric_column(i)
                feature_columns.append(ws)

            elif i in degrees:

                col_name = i
```

```
                # Create a bucket feature column for latitude.
                wd_as_a_numeric_column =
tf.feature_column.numeric_column(i)

                wd_boundaries =
list(np.arange(int(min(train_data[col_name])),

int(max(train_data[col_name])),
                                                 resolution_in_degrees))

                wd_feature =
tf.feature_column.bucketized_column(wd_as_a_numeric_column, wd_boundaries)

                feature_columns.append(wd_feature)

        my_feature_layer = tf.keras.layers.DenseFeatures(feature_columns)

        #Creating a neural network model

        model = tf.keras.Sequential()

        #Input layer
        model.add(my_feature_layer)

        #Hidden layer-1 with 6 neurons
        model.add(tf.keras.layers.Dense(units=42,
                                        activation='relu',
                                        name='Hidden1'))
        model.add(tf.keras.layers.Dropout(0.3))


        #Hidden layer-2 with 12 neurons
        model.add(tf.keras.layers.Dense(units=84,
                                        activation='relu',
                                        name='Hidden2'))
        model.add(tf.keras.layers.Dropout(0.3))


        #Define output layer
        model.add(tf.keras.layers.Dense(units=1,
                                         name='output'))


        model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.00001),
## good with total norm -- lr =0.00001
                      loss='mean_squared_error',
##  model.add(tf.keras.layers.Dropout(0.3))
                      metrics=[tf.keras.metrics.MeanSquaredError()])
## 1-42, 2-84



        return model


    def trainmodel(self, model, train_data, label_name):

        #Train the model by feeding it data
```

```
        feature = {name:np.array(value) for name, value in
train_data.items()}

        label = np.array(feature.pop(label_name))

        history = model.fit(x=feature, y=label, batch_size=500,
validation_split=0.2,    ## bs- 500, epochs--150
                              epochs=150, shuffle=True, verbose=1)

        epochs = history.epoch

        hist=pd.DataFrame(history.history)
        mse=hist['mean_squared_error']

        return history

    def mse(self, y_test, predictions):
        return mean_squared_error(y_test, predictions)

    def mae(self, y_test, predictions):
        return mean_absolute_error(y_test, predictions)

    def variance(self, y_test, predictions):
        var = (np.var(predictions) - np.var(y_test))/np.var(y_test)
        return var

    def bias(self, y_test, predictions):
        bias =  (np.mean(predictions) - np.mean(y_test))/np.mean(y_test)
        return bias
```

References

[1] D. B. d. Alencar and C. D. M. Affonso, "Different Models for Forecasting Wind Power," *Energies 2017,* pp. 2-4, 2017.

[2] M. Patel, "Wind and Solar Power Systems: Design, Analysis, and Operation," pp. 4-6, 2005.

[3] A. Stetco, F. Dinmohammadi and X. Zhao, "Machine learning methods for wind turbine condition monitoring: A review," *A. Stetco et al. / Renewable Energy ,* pp. 621-623, 2018.

[4] "National center for atmospheric research," University Corporation for Atmospheric Research., [Online]. Available: https://eo.ucar.edu/basics/wx_2_c.html.

[5] S. Emeis, Wind Energy Meteorology, Garmisch-Partenkirchen: Springer, 2013.

[6] F. Miceli, "Wind farms construction," 23 October 2017. [Online]. Available: https://www.windfarmbop.com/tag/met-mast-2/. [Accessed 08 june 2022].

[7] "LuckGrib," 23 july 2020. [Online]. Available: https://luckgrib.com/models/dwd_icon_d2/. [Accessed 15 june 2022].

[8] "dwd," [Online]. Available: https://www.dwd.de/EN/ourservices/nwp_forecast_data/nwp_forecast_data.html. [Accessed 05 june 2022].

[9] A. Geron, Hands on machine learning with Scikit-Learn, Keras and Tensorflow, Beijing: O REILLY, 2019.

[10] m. gupta, Geeks, 22 04 2022. [Online]. Available: https://www.geeksforgeeks.org/ml-linear-regression/. [Accessed 04 11 2022].

[11] G. K. Uyanik and N. Güler, "A study on multiple linear regression analysis," *Procedia-Social and Behavioral Sciences ,* pp. 234-240, 2013.

[12] D. Basak, S. Pal and D. C. Patranabis, "Support Vector Regression," *Neural information Processing,* vol. 11, pp. 203-206, 2007.

[13] Rohit, "Support Vector Machine — Introduction to Machine Learning Algorithms," 07 june 2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. [Accessed 20 july 2022].

[14] Rodríguez-Pérez, M. Vogt and J. Bajorath, "National Library of Medicine," 31 October 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6045367/. [Accessed 20 August 2022].

[15] Brieman, "Machine learning," *Random Forests,* pp. 5-32, 2001.

[16] "scikit-learn," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. [Accessed 15 July 2022].

[17] "An Empirical Study of Downstream Analysis Effects of Model Pre-Processing Choices," [Online]. Available: https://www.researchgate.net/figure/Random-forest-structure-15_fig3_346411178. [Accessed 29 August 2022].

[18] S. Sun, "Neural Networks," Beijing, 2017.

[19] K. Gurney, An introduction to neural networks, London: UCL Press, 1997.

[20] D. A. Thomas, An introduction to neural networks for beginners, 2005.

[21] T. adams, "SCADA System Fundamentals," New Jersey, 2014.

[22] FlexBooks, "ck-12," 19 july 2022. [Online]. Available: https://flexbooks.ck12.org/cbook/ck-12-basic-algebra-concepts/section/5.11/primary/lesson/linear-interpolation-and-extrapolation-bsc-alg/. [Accessed 25 August 2022].

[23] charris, "Github," 29 December 2019. [Online]. Available: https://github.com/numpy/numpy/blob/v1.16.6/numpy/lib/function_base.py#L1471-L1525. [Accessed 26 August 2022].

[24] "aws," Amazon, [Online]. Available: https://docs.aws.amazon.com/machine-learning/latest/dg/splitting-types.html. [Accessed 28 August 2022].

[25] Scikit Learn, "scikit-learn," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html. [Accessed 20 july 2022].

[26] S. Gen, "statisticshowto," [Online]. Available: https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/. [Accessed 01 september 2022].

[27] S. C, "Encyclopedia of machine learning," 2011. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_525. [Accessed 01 september 2022].

[28] s. singh, "towards data science," 21 May 2018. [Online]. Available: https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229. [Accessed 02 September 2022].

[29] seaborn, [Online]. Available: https://seaborn.pydata.org/generated/seaborn.boxplot.html. [Accessed 02 september 2022].

Declaration

"I here with declare that I have composed the present thesis myself and without use of any sources other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned.

In particular, I have not utilized commercial advice. Third parties have neither received direct nor indirect monetary benefits from me for actions which are related to the content of the submitted work.

The thesis in the same or similar form has not been submitted to any examination board and has not been published. This thesis was not yet, even in part, used in another examination or as a course performance, either in Germany or abroad."

Nordhausen,

20.04.2023