University of Hamburg - Mathematical Modelling Camp

Predicting Wind Turbine Power Output Using Mathematical Models in Favor of Energy Generation in Germany

Authors:

- Rovanos Nzanguim Tsafack (7704600)
- Jessie Ighogho Ifeanyi (7704546)
- Saidi Atwaya Nchimbi (7704562)

Supervisors:

- Prof. Dr. Ingenuin Gasser
- Prof. Giuseppe Maria Coclite

Date: July 5, 2023

Abstract:

Wind power curves are essential for various applications such as wind power forecasting, wind turbine condition monitoring, wind energy potential estimation, and wind turbine selection. However, creating reliable wind power curves from raw wind data is challenging due to the presence of outliers resulting from unexpected conditions like wind curtailment and blade damage. This notebook presents an approach for modeling power curves using a cubic power curve model and 5PL models. It also exams the effect of cp decay on the power output prediction

This project proposes a 5PL model and cubic power curve model for predicting wind turbine power output and incorporates a cost model to assess the economic feasibility of wind energy projects. The cubic power curve and 5PL model captures the non-linear relationship between wind speed and power output, the 5PL describes sigmoidal or "S"-shaped curves that best fits a particular dataset, while the cost model considers capital costs, operation and maintenance expenses, and revenue generation. The findings contribute to improved wind power analysis and decision-making processes, enabling stakeholders to make informed choices regarding wind energy investments.

By leveraging the 5PL model and cubic power curve model, this study aims to enhance the accuracy and reliability of wind power curve modeling. The findings and insights from this research contribute to the development of improved methods for wind power analysis and decision-making in the field of wind energy.

Introduction:

The global pursuit of renewable energy sources has accelerated the development of wind power as a sustainable solution for electricity generation. Accurate estimation of wind turbine power output is crucial for optimizing energy production and evaluating the economic viability of wind projects. Power curves, which depict the relationship between wind speed and turbine power output, are fundamental tools in assessing wind turbine performance.

This project aims to enhance the accuracy of power curve modeling and evaluate the economic feasibility of wind energy projects by proposing a cubic power curve model and integrating a cost model.

The cubic power curve model accommodates the non-linear nature of the wind speed and power output relationship, providing improved predictions compared to traditional linear models. By incorporating cubic functions, the model captures variations in wind speed and enhances the accuracy of power curve estimation.

The 5PL (Five-Parameter Logistic) model is a mathematical model commonly used in various fields, including biology and pharmacology, to describe sigmoidal or "S"-shaped curves. In the context of wind speed and power output, the 5PL model can be employed to capture the non-linear relationship between these variables.

In addition to power curve modeling, this research incorporates a cost model to assess the financial viability of wind energy projects. The cost model considers capital costs, operation and maintenance expenses, and revenue generation to provide a comprehensive evaluation of project economics. By analyzing these cost factors, stakeholders can make informed decisions regarding the implementation and profitability of wind energy initiatives.

The primary objective of this study is to develop and validate mathematical power prediction models: the 5PL and the cubic power curve model for predicting wind turbine power output. To achieve this, historical wind data, including wind speed measurements and corresponding power output, will be utilized. The cubic power curve model will be calibrated and validated using this dataset, ensuring its accuracy and reliability.

Furthermore, a cost model will be integrated into the analysis to evaluate the economic feasibility of wind energy projects. By considering capital costs associated with wind turbine installation, operation and maintenance expenses, and potential revenue generation, the cost model provides insights into the financial viability of wind projects. This evaluation enables stakeholders to assess the potential returns on investment and make informed

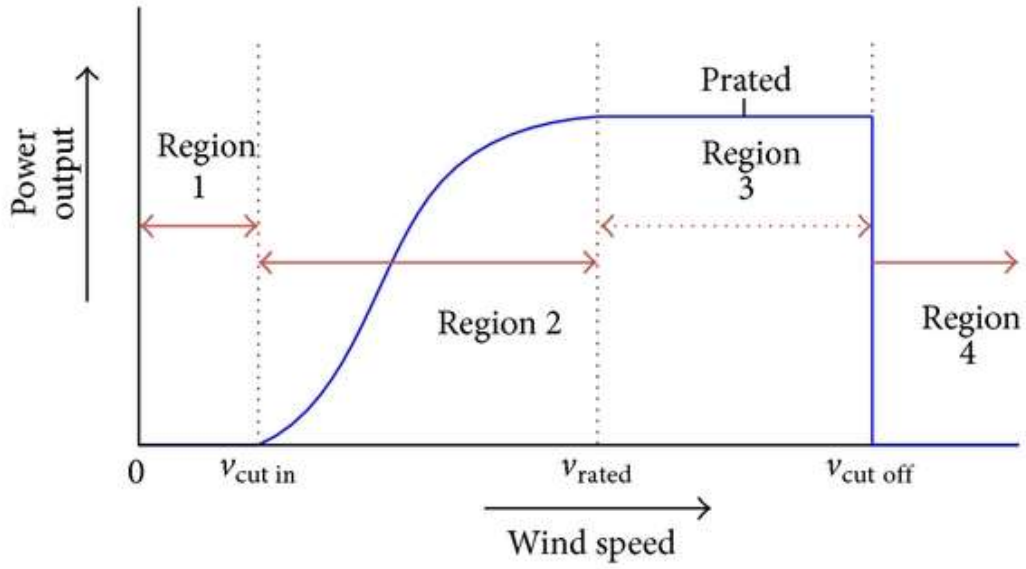
Overview

Wind turbines are designed to harness the power of the wind and convert it into electrical energy. Each wind turbine manufacturer provides an ideal energy production curve, commonly referred to as the power curve, which illustrates the electrical power output ratings of the turbine at different wind speeds. The power curve is a crucial parameter in wind power analysis, as it helps in predicting the turbine's performance, selecting appropriate turbines for specific locations, and estimating the wind energy potential of a site.

A typical wind turbine power curve exhibits three main characteristic speeds: the cut-in speed (Vc), the rated speed (Vr), and the cut-out speed (Vs). The cut-in speed represents the wind speed at which the turbine starts generating power. At this speed, the turbine's rotor blades begin to rotate and initiate energy production. The rated speed is the wind speed at which the turbine is designed to produce its maximum rated power output. Finally, the cut-out speed marks the wind speed at which the turbine's power generation is shut down to prevent potential defects and damages.

The 5PL model is defined by five parameters: lower asymptote, upper asymptote, slope, EC50 (the midpoint of the curve), and Hill's slope (a measure of the steepness of the curve). By fitting the 5PL model to wind speed and power output data, it allows for a more flexible representation of the relationship compared to linear or quadratic models.

In the case of wind speed and power output, the 5PL model can provide insights into the optimal wind speed range for maximum power production and help in analyzing the efficiency of wind turbines. By estimating the parameters of the 5PL model, it becomes possible to predict power output for a given wind speed and make informed decisions in wind energy applications.



It is important to note that the power curve provided by manufacturers is based on theoretical assumptions of ideal meteorological and topographical conditions. However, in practical applications, wind turbines are rarely operated under these ideal conditions. The empirical power curves, which reflect the actual performance of the turbines in real-world scenarios, can vary significantly from the theoretical curves.

Several factors contribute to the differences between theoretical and empirical power curves. The location of the turbine plays a significant role, as local wind characteristics, such as wind velocity distribution and wind direction, can vary widely. Other factors, including air density variations, mechanical and control issues, as well as uncertainties in measurements, also impact the actual power output of a wind turbine. This is why the 5PL model yields better results than the cubic power curve, which uses only historical windspeed and power output, because it takes other measurements embedded in the power output that would be otherwise difficult to model from the windspeeds alone.

This research aims to analyze the power curves of wind turbines in the context of real-world operating conditions, considering the aforementioned factors that deviate from the ideal assumptions. By comparing and evaluating the theoretical and empirical power curves, insights can be gained into the performance variations and discrepancies. This analysis will contribute to a better understanding of the factors influencing power output

Data description

In this wind turbine power curve and output analysis, we consider 3 datasets:

- Tdata1: Siemens and Nord Manufacturers wind turbine data
- Tdata2: Hamburg Onshore Wind Farm data
- Tdata3: Turkey wind farm data

Tdata1 Dataset details:: Siemens and Nordex Manufacturers wind turbine data:

We filtered Siemens & Nordex Manufacturers data from this dataset (https://openenergy-platform.org/dataedit/view/supply/wind_turbine_library)

Siemens (https://www.siemensgamesa.com/): Siemens is a multinational conglomerate based in Germany that operates in various sectors, including energy, healthcare, transportation, and industrial automation. In the wind power industry, Siemens is known for its high-quality wind turbines and comprehensive solutions for renewable energy projects. They offer a wide range of onshore and offshore wind turbines with different capacities and rotor diameters to suit various wind conditions and project requirements. Siemens wind turbines are known for their advanced technology, reliability, and performance.

Nordex (https://www.nordex-online.com/): Nordex is a German manufacturer specializing in the production of wind turbines. With several decades of experience, Nordex has established itself as a leading player in the global wind energy market. Nordex offers a range of onshore wind turbines designed for different wind conditions and power requirements. Their turbines are known for their efficiency, robustness, and high performance. Nordex focuses on delivering sustainable and reliable solutions for renewable energy projects worldwide.

Both Siemens and Nordex have made significant contributions to the wind power industry and have a strong track record in delivering innovative and reliable wind turbines. They continue to play a vital role in advancing the adoption of renewable energy and promoting a sustainable future.

Here is a table of description of the wind turbine data with the turbine ID included:

turbine_id	manufacturer	name	description
16	Nordex	N131/3900	Nordex wind turbine with a rotor diameter of 131 meters and a rated capacity of 3.9 MW.
17	Nordex	N131/3600	Nordex wind turbine with a rotor diameter of 131 meters and a rated capacity of 3.6 MW.
18	Nordex	N131/3300	Nordex wind turbine with a rotor diameter of 131 meters and a rated capacity of 3.3 MW.
19	Nordex	N131/3000	Nordex wind turbine with a rotor diameter of 131 meters and a rated capacity of 3.0 MW.
20	Nordex	N117/3600	Nordex wind turbine with a rotor diameter of 117 meters and a rated capacity of 3.6 MW.
21	Nordex	N117/3000	Nordex wind turbine with a rotor diameter of 117 meters and a rated capacity of 3.0 MW.
22	Nordex	N117/2400	Nordex wind turbine with a rotor diameter of 117 meters and a rated capacity of 2.4 MW.
23	Nordex	N100/3300	Nordex wind turbine with a rotor diameter of 100 meters and a rated capacity of 3.3 MW.
24	Nordex	N100/2500	Nordex wind turbine with a rotor diameter of 100 meters and a rated capacity of 2.5 MW.
25	Nordex	N90/2500	Nordex wind turbine with a rotor diameter of 90 meters and a rated capacity of 2.5 MW.
26	Nordex	N80/2500	Nordex wind turbine with a rotor diameter of 80 meters and a rated capacity of 2.5 MW.

turbine_id	manufacturer	name	description
27	Nordex	AW140/3000	Nordex wind turbine with a rotor diameter of 140 meters and a rated capacity of 3.0 MW.
28	Nordex	AW132/3000	Nordex wind turbine with a rotor diameter of 132 meters and a rated capacity of 3.0 MW.
29	Nordex	AW125/3000	Nordex wind turbine with a rotor diameter of 125 meters and a rated capacity of 3.0 MW.
30	Nordex	AW116/3000	Nordex wind turbine with a rotor diameter of 116 meters and a rated capacity of 3.0 MW.
31	Nordex	AW100/3000	Nordex wind turbine with a rotor diameter of 100 meters and a rated capacity of 3.0 MW.
32	Nordex	AW82/1500	Nordex wind turbine with a rotor diameter of 82 meters and a rated capacity of 1.5 MW.
33	Nordex	AW77/1500	Nordex wind turbine with a rotor diameter of 77 meters and a rated capacity of 1.5 MW.
34	Nordex	AW70/1500	Nordex wind turbine with a rotor diameter of 70 meters and a rated capacity of 1.5 MW.
81	Siemens	SWT-2.3-93	Siemens wind turbine with a rotor diameter of 93 meters and a rated capacity of 2.3 MW.
61	Siemens	SWT-7.0-154	Siemens wind turbine with a rotor diameter of 154 meters and a rated capacity of 7.0 MW.
62	Siemens	SWT-6.0-154	Siemens wind turbine with a rotor diameter of 154 meters and a rated capacity of 6.0 MW.
63	Siemens	SWT-6.0-120	Siemens wind turbine with a rotor diameter of 120 meters and a rated capacity of 6.0 MW.
64	Siemens	SWT-4.0-130	Siemens wind turbine with a rotor diameter of 130 meters and a rated capacity of 4.0 MW.
82	Siemens	SWT-2.3-82	Siemens wind turbine with a rotor diameter of 82 meters and a rated capacity of 2.3 MW.
65	Siemens	SWT-3.6-130	Siemens wind turbine with a rotor diameter of 130 meters and a rated capacity of 3.6 MW.
66	Siemens	SWT-3.6-120	Siemens wind turbine with a rotor diameter of 120 meters and a rated capacity of 3.6 MW.
67	Siemens	SWT-3.6-107	Siemens wind turbine with a rotor diameter of 107 meters and a rated capacity of 3.6 MW.
68	Siemens	SWT-3.3-130	Siemens wind turbine with a rotor diameter of 130 meters and a rated capacity of 3.3 MW.
69	Siemens	SWT-3.2-113	Siemens wind turbine with a rotor diameter of 113 meters and a rated capacity of 3.2 MW.
70	Siemens	SWT-3.2-108	Siemens wind turbine with a rotor diameter of 108 meters and a rated capacity of 3.2 MW.
71	Siemens	SWT-3.2-101	Siemens wind turbine with a rotor diameter of 101 meters and a rated capacity of 3.2 MW.
72	Siemens	SWT-3.15-142	Siemens wind turbine with a rotor diameter of 142 meters and a rated capacity of 3.15 MW.
73	Siemens	SWT-3.0-113	Siemens wind turbine with a rotor diameter of 113 meters and a rated capacity of 3.0 MW.
74	Siemens	SWT-3.0-108	Siemens wind turbine with a rotor diameter of 108 meters and a rated capacity of 3.0 MW.
75	Siemens	SWT-3.0-101	Siemens wind turbine with a rotor diameter of 101 meters and a rated capacity of 3.0 MW.
76	Siemens	SWT-2.5-120	Siemens wind turbine with a rotor diameter of 120 meters and a rated capacity of 2.5 MW.

turbine_id	manufacturer	name	description
77	Siemens	SWT-2.3-120	Siemens wind turbine with a rotor diameter of 120 meters and a rated capacity of 2.3 MW.
78	Siemens	SWT-2.3-113	Siemens wind turbine with a rotor diameter of 113 meters and a rated capacity of 2.3 MW.
79	Siemens	SWT-2.3-108	Siemens wind turbine with a rotor diameter of 108 meters and a rated capacity of 2.3 MW.
80	Siemens	SWT-2.3-101	Siemens wind turbine with a rotor diameter of 101 meters and a rated capacity of 2.3 MW.
83	Siemens	SWT-1.3-62	Siemens wind turbine with a rotor diameter of 62 meters and a rated capacity of 1.3 MW.

Both Siemens and Nordex provide power curves for their wind turbines. A power curve is a graphical representation that shows the relationship between the wind speed and the power output of a wind turbine. It helps in understanding the turbine's performance under different wind conditions.

Siemens (https://www.siemensgamesa.com/) and Nordex (https://www.nordex-online.com/) typically provide power curves for their turbines as part of their technical documentation. These power curves are generated through extensive testing and data analysis, taking into account factors such as wind speed, rotor diameter, and generator capacity. The power curves provide valuable information for wind farm developers, operators, and investors, as they can assess the expected power generation and optimize the turbine's performance for specific wind conditions. You can explore their "Products" section or look for specific turbine models to find detailed specifications, including power curves.

The power curves provided by Siemens and Nordex are typically based on actual measurements and are validated through field testing. They are an essential tool in estimating the energy production and optimizing the overall efficiency of wind energy projects.

Cleaned Columns:

Tdata2: Hamburg, Germany Wind Dataset

The dataset contains Hamburg wind speed data (https://www.visualcrossing.com/weather-history) for 2022-2023 merged with the data set captures hourly readings between 01.01.2022 and 16.06.2023 in Hamburg (https://montelgroup.com). Each data value belongs only to the relevant time period and the input variables transmitted in the data set for the time period. We extracted the following relevant columns for an Onshore wind farm.

Columns:

- Timestamp
- Wind_Speed
- Wind_Direction

Tdata3 Dataset details: Scada Dataset

Wind Turbines, this Scada Systems measure and save data's like wind speed, wind direction, generated power etc. for 10 minutes intervals. This file was taken from a wind turbine's scada system (https://www.kaggle.com/datasets/berkerisen/wind-turbine-scada-dataset) that is working and generating power in Turkey.

The data set consists of real-time SCADA data. Each data value belongs only to the relevant time period and the input variables transmitted in the data set for the time period to be predicted are prepared to be used to predict the power generation result in the same time period.

In The Table: The real-time power generation amount (Power(kW)) of a wind turbine between 01.01.2018 and 31.12.2018 is given on a 10-minute basis.

Content The data's in the file are:

- Date/Time (for 10 minutes intervals)
- LV ActivePower (kW): The actual power generated by the turbine for that moment
- Wind Speed (m/s): The wind speed at the hub height of the turbine
- Theoretical_Power_Curve (KWh): The theoretical power values that the turbine generates with that wind speed which is given by the turbine manufacturer
- Wind Direction (°): The wind direction at the hub height of the turbine

Cleaned Columns:

- Date/Time (for 10 minutes intervals)
- LV ActivePower (kW): The actual power generated by the turbine for that moment
- Wind Speed (m/s): The wind speed at the hub height of the turbine (the wind speed that turbine use for electricity generation)

Work plan 🤝

- 1- Data Exploration & Analysis
- 2- Building a Mathematical Model / Predict /calculate

Data Exploration & Analysis

In [1]:
```
import math
import numpy as np # linear algebra
import pandas as pd # data processing
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
import unittest
import os
%matplotlib inline
from scipy.optimize import curve_fit
from sklearn.metrics import mean_absolute_error

import datetime
import time
```

Tdata1

In [2]:
```
Tdata1=pd.read_csv("supply_wind_turbine_library_siemens_and_nord.csv")
Tdata1.sample(10)
```

Out[2]:

	turbine_id	Power_Coefficient_Values	power_curve_wind_speeds	power_curve_values	rotor_diameter	r
275	66	0.300	12.0	3588	120	
313	69	0.430	4.0	169	113	
125	22	0.460	7.0	1037	117	
288	66	0.033	25.0	3600	120	
170	24	0.302	12.0	2498	100	
328	69	0.076	19.0	3200	113	
245	65	0.414	5.0	421	130	
261	65	0.048	21.0	3600	130	
45	18	0.433	5.0	447	131	
349	72	0.066	17.0	3150	142	

In [3]:
```
pd.options.display.float_format = '{:.2f}'.format
Tdata1 = Tdata1.dropna()
# droping all the null values from the data
Tdata1.describe()
```

Out[3]:

	turbine_id	Power_Coefficient_Values	power_curve_wind_speeds	power_curve_values	rotor_diameter
count	379.00	379.00	379.00	379.00	379.00
mean	38.51	0.23	12.65	2189.08	118.46
std	23.59	0.17	6.40	1186.93	15.37
min	17.00	0.00	0.00	0.00	90.00
25%	19.00	0.07	7.00	1194.50	113.00
50%	24.00	0.18	12.50	2500.00	120.00
75%	66.00	0.41	18.00	3200.00	131.00
max	78.00	0.48	26.00	3600.00	142.00

In [4]:
```
Tdata1.nunique()
```

Out[4]:
```
turbine_id                12
Power_Coefficient_Values  219
power_curve_wind_speeds    52
power_curve_values        166
rotor_diameter              8
rotor_area                  8
dtype: int64
```

In [5]:
```
Tdata1['turbine_id'].unique()
```

Out[5]:
```
array([17, 18, 19, 22, 24, 25, 65, 66, 68, 69, 72, 78], dtype=int64)
```

Tdata2

In [6]:
```
Tdata2=pd.read_csv("wind_speed_direction_historical_forecast_data.csv")
Tdata2.sample(10)
```

Out[6]:

	timestamp	Wind_Speed	Wind_Direction	Actual_Power_Output
5146	03/08/2022 10:00	2.19	99.46	0.00
2551	17/04/2022 07:00	6.37	47.29	624.07
7251	30/10/2022 03:00	3.88	158.20	5.22
3015	06/05/2022 15:00	15.73	15.95	2820.47
10739	24/03/2023 11:00	18.58	215.54	2820.47
10613	19/03/2023 05:00	4.69	175.60	166.90
4076	19/06/2022 20:00	8.89	238.24	1808.90
6363	23/09/2022 03:00	4.61	141.34	166.90
1989	24/03/2022 21:00	10.82	3.81	2115.04
1559	06/03/2022 23:00	8.89	21.37	1808.90

In [7]:
```
Tdata2.head(5)
```

Out[7]:

	timestamp	Wind_Speed	Wind_Direction	Actual_Power_Output
0	01/01/2022 00:00	5.09	188.13	189.79
1	01/01/2022 01:00	3.98	174.81	5.22
2	01/01/2022 02:00	4.32	180.00	166.90
3	01/01/2022 03:00	3.26	173.66	0.00
4	01/01/2022 04:00	3.55	156.04	5.22

In [8]:
```
Tdata2.tail(5)
```

Out[8]:

	timestamp	Wind_Speed	Wind_Direction	Actual_Power_Output
13077	NaN	NaN	NaN	NaN
13078	NaN	NaN	NaN	NaN
13079	NaN	NaN	NaN	NaN
13080	NaN	NaN	NaN	NaN
13081	NaN	NaN	NaN	NaN

In [9]:
```
Tdata2 = Tdata2.dropna()
# droping all the null values from the data
Tdata2.describe()
```

Out[9]:

	Wind_Speed	Wind_Direction	Actual_Power_Output
count	12722.00	12722.00	12722.00
mean	7.79	157.99	1055.13
std	4.72	96.17	1051.71
min	0.00	1.10	0.00
25%	4.69	65.22	166.90
50%	6.61	160.71	624.07
75%	9.69	228.37	2038.90
max	34.80	360.00	3333.82

In [10]:
```
Tdata2.nunique()
```

Out[10]:
```
timestamp              12722
Wind_Speed              1182
Wind_Direction          2499
Actual_Power_Output       36
dtype: int64
```

In [11]:
```
# missing value

Tdata2.isnull().sum()
```

Out[11]:
```
timestamp              0
Wind_Speed             0
Wind_Direction         0
Actual_Power_Output    0
dtype: int64
```

In [12]:
```
# Display column names
print(Tdata2.columns)
```

```
Index(['timestamp', 'Wind_Speed', 'Wind_Direction', 'Actual_Power_Output'], dtype='obj
ect')
```

In [13]:
```
# Format timestamp labels
Tdata2["timestamp"] = pd.to_datetime(Tdata2["timestamp"], format = "%d/%m/%Y %H:%M", er
Tdata2
```

Out[13]:

	timestamp	Wind_Speed	Wind_Direction	Actual_Power_Output
0	2022-01-01 00:00:00	5.09	188.13	189.79
1	2022-01-01 01:00:00	3.98	174.81	5.22
2	2022-01-01 02:00:00	4.32	180.00	166.90
3	2022-01-01 03:00:00	3.26	173.66	0.00
4	2022-01-01 04:00:00	3.55	156.04	5.22
...	...	...	...	...
12717	2023-06-15 21:00:00	6.79	32.01	524.61
12718	2023-06-15 22:00:00	6.30	30.96	624.07
12719	2023-06-15 23:00:00	6.79	32.01	524.61
12720	2023-06-16 00:00:00	6.29	13.24	624.07
12721	2023-06-16 01:00:00	5.32	28.30	189.79

12722 rows × 4 columns

In [14]:
```
# Calculate z-scores for the "LV ActivePower (kW)" column
z_scores = (Tdata2['Actual_Power_Output'] - Tdata2['Actual_Power_Output'].mean()) / Tda

# Define a threshold for the z-score (e.g., 3 or any other value)
threshold = 2

# Remove rows where "LV ActivePower (kW)" has a z-score above the threshold
Tdata2 = Tdata2[(np.abs(z_scores) < threshold)]
```

Tdata3

In [15]:
```
Tdata3=pd.read_csv("T1.csv")
Tdata3.sample(10)
```

Out[15]:

	Date/Time	LV ActivePower (kW)	Wind Speed (m/s)
48625	18 12 2018 18:30	3486.29	12.45
45105	24 11 2018 05:00	2125.14	9.26
25788	04 07 2018 07:30	877.66	6.98
19176	18 05 2018 20:30	130.21	4.48
45214	24 11 2018 23:10	0.00	0.79
36266	15 09 2018 11:10	0.00	2.44
10047	16 03 2018 06:30	3197.42	11.61
2704	19 01 2018 22:20	3603.10	14.47
37479	23 09 2018 21:20	1372.15	8.37
20097	25 05 2018 06:00	2193.90	9.32

In [16]:
```
Tdata3.head(5)
```

Out[16]:

	Date/Time	LV ActivePower (kW)	Wind Speed (m/s)
0	01 01 2018 00:00	380.05	5.31
1	01 01 2018 00:10	453.77	5.67
2	01 01 2018 00:20	306.38	5.22
3	01 01 2018 00:30	419.65	5.66
4	01 01 2018 00:40	380.65	5.58

In [17]:
```
Tdata3.tail(5)
```

Out[17]:

	Date/Time	LV ActivePower (kW)	Wind Speed (m/s)
50525	31 12 2018 23:10	2963.98	11.40
50526	31 12 2018 23:20	1684.35	7.33
50527	31 12 2018 23:30	2201.11	8.44
50528	31 12 2018 23:40	2515.69	9.42
50529	31 12 2018 23:50	2820.47	9.98

In [18]:
```
Tdata3 = Tdata3[Tdata3['LV ActivePower (kW)'] >= 0].dropna()
```

In [19]:
```
Tdata3.describe()
```

Out[19]:

	LV ActivePower (kW)	Wind Speed (m/s)
count	50473.00	50473.00
mean	1309.16	7.56
std	1312.46	4.23
min	0.00	0.00
25%	52.74	4.21
50%	828.01	7.11
75%	2484.36	10.30
max	3618.73	25.21

In [20]:
```
# # Format timestamp labels
# Tdata3["timestamp"] = pd.to_datetime(Tdata2["timestamp"], format = "%d/%m/%Y %H:%M", 
Tdata3.rename(columns={'Date/Time':'Time'},inplace=True)
Tdata3.sample(15)
```

Out[20]:

	Time	LV ActivePower (kW)	Wind Speed (m/s)
23711	19 06 2018 15:30	3034.81	11.23
28076	20 07 2018 04:50	257.57	4.98
9947	15 03 2018 13:50	0.00	2.85
46129	01 12 2018 07:40	0.00	4.63
16296	28 04 2018 18:30	3596.47	15.38
3335	24 01 2018 07:30	0.00	9.19
19411	20 05 2018 11:40	352.03	5.36
47499	10 12 2018 21:10	2173.91	8.99
7049	23 02 2018 10:40	0.00	11.19
5429	12 02 2018 04:40	0.00	1.84
27511	16 07 2018 06:40	25.67	2.97

In [21]:
```
Tdata3.describe()
```

Out[21]:

	LV ActivePower (kW)	Wind Speed (m/s)
count	50473.00	50473.00
mean	1309.16	7.56
std	1312.46	4.23
min	0.00	0.00
25%	52.74	4.21
50%	828.01	7.11
75%	2484.36	10.30
max	3618.73	25.21

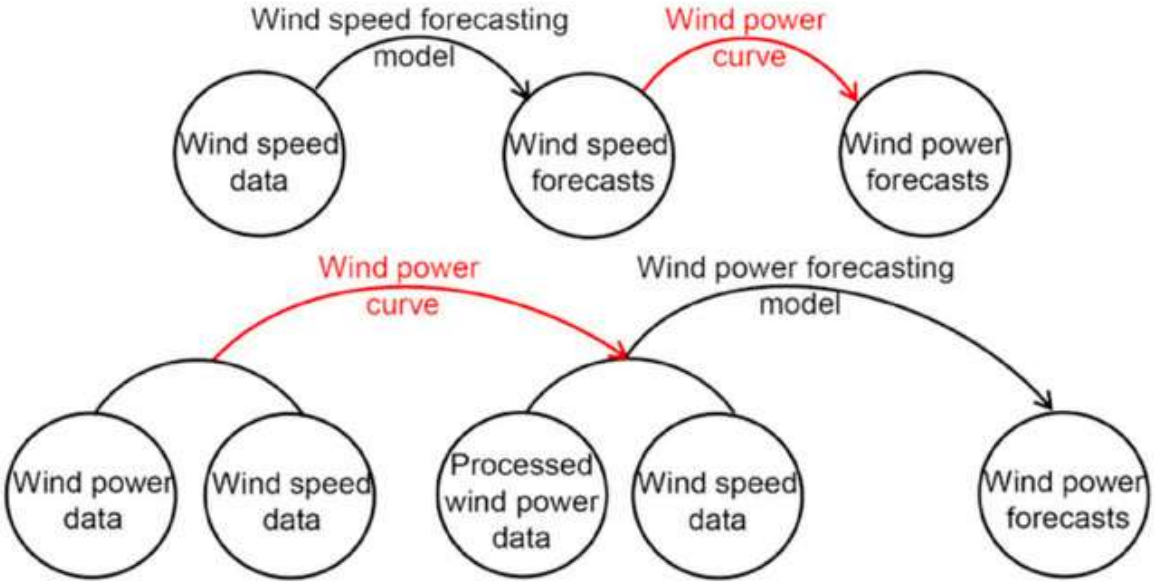
In [22]:
```
Tdata3.nunique()
```

Out[22]:
```
Time                 50473
LV ActivePower (kW)  38707
Wind Speed (m/s)     50248
dtype: int64
```

Power Prediction Model

Power output prediction can be classified based on fundamental equations of power available in the wind and models based on the concept of power curve of a wind turbine. This can be classified into parametric and non-parametric techniques, where the parametric techniques are based on Mathematical models such as the linearized segment model, the polynomial power curve, the probabilistic model, the ideal power curve, the 4-parametric logistic function, and the 5-parametric logistic function. The non-parametric techniques do not impose any pre-specified model. Modeling methods in which the actual power curve of an individual wind turbine is used for developing characteristic equations, by utilizing various curve fitting techniques, accurately

predict the power output of the wind turbine. The suitability of a particular model depends on the shape of the power curve of the selected wind turbin. In this work, we will use the cubic power approximation model of the power curve to estimate the production of electricity.



Cubic Power Curve Prediction Model

Power curves could either be provided by the wind turbine manufacturer or be approximated. The power delivered by a wind turbine is usually represented through its power curve, where a relation between the wind speed and the power is established. For the variable speed wind turbines, this relationship can be expressed in the following way, [Carrillo et al., 2013]:

$$P(v) = \begin{cases} 0 & \text{if } v < v_{ci} \text{ or } v > v_{co} \\ q(v) & \text{if } v_{ci} \leq v < v_r \\ P_r & \text{if } v_r \leq v \leq v_{co} \end{cases}$$

where $P(v)$ is the electric power in W, v_{ci} is the cut-in wind speed in m/s, v_{co} is the cut-out wind speed in m/s, v_r is the rated wind speed in m/s, P_r is the rated power in W, and $q(v)$ is the non-linear relationship between power and wind speed. The most typical mathematical equations for representing the non-linear part $q(v)$ of a power curve are: the polynomial power curve, the exponential power curve, the cubic power curve, and the approximate cubic power curve. In this work, we are going to use the approximate power curve which is best fitted for electricity production, see [Carrillo et al., 2013]. The approximate cubic power curve is an approximation of the cubic power curve:

$$q(v) = \frac{1}{2\rho C p_{\text{eq}} v^3}$$

where Cp_{eq} is a constant equivalent to the power coefficient. This is obtained by assuming that Cp_{eq} is equal to the maximum value of the effective power coefficient $Cp_{\max}$. We use the term "effective" to say that mechanical and electrical losses are included in this coefficient.

Thus, our cubic power curve function becomes:

$$q(v) = \frac{1}{2\rho C p_{\max} v^3}$$

Using the previous equations, we obtain:

$$P(v) = \begin{cases} 0 & \text{if } v < v_{ci} \text{ or } v > v_{co} \\ \dfrac{1}{2\rho C p_{\max} v^3} & \text{if } v_{ci} \le v < v_r \\ P_r & \text{if } v_r \le v \le v_{co} \end{cases}$$

To calculate the rated power of a wind turbine based on its parameters, there are a few common approaches. Here are two widely used methods:

1. Using the Swept Area: The rated power can be calculated by multiplying the swept area of the turbine rotor by the power coefficient at the rated wind speed. The formula is as follows:

   P_rated = 0.5 * rho * A * v_r^3 * C_p

   Where:

   - `P_rated` is the rated power output (Watts)
   - `rho` is the air density (kg/m^3)
   - `A` is the swept area of the rotor (m^2)
   - `v_r` is the rated wind speed (m/s)
   - `C_p` is the power coefficient at the rated wind speed
2. Using the Power Curve Data: The rated power can also be determined directly from the power curve data of the wind turbine. The rated power is the maximum power output achieved in the power curve. This can be obtained by finding the highest power value in the power curve data.

Both methods have their advantages and may be used depending on the available information and requirements. The first method allows for a theoretical calculation based on design parameters, while the second method considers the actual performance of the turbine.

It's important to note that the specific calculation may vary depending on the turbine model and design. It's recommended to refer to the manufacturer's specifications or consult engineering resources for precise calculations relevant to a specific wind turbine.

Using the Power Curve Data

Below we Test our Model against the data using Cp_max (Rated power coefficient) as 0.45 and setting P_r (Rated power output) as the maximum power output of each turbine

5 Parametric Logistic Function (5PL) Model

The cubic power curve neglects many other parameter that can affect the power output of a wind turbine/farm. This is why we choose to consider also the 5PL model for a higher level of accuracy. Due to aerodynamics over the surface of wind turbines blades, Cp varies with v near cut-in speed. The result is thus an acutely concave upwards curve near cut-in speed. However, as v increases, Cp becomes essentially constant which means the relationship between P and v should become cubic should start to resemble a cubic relationship. Eventually, as v approaches vr, Cp, starts to decrease. The result is thus a concave downwards curve near rated wind speed. The overall shape of Wind Power curve resembles the letter ‘S’ (Figure 1). Thus, it becomes prevalent to try to model it with logistic functions. In particular for a good fit of our Mathematical Model, we use the 5PLF for our approximated power curve because of its sinoidal feature. Thus our Mathematical model replaces q(v) in the previous model with $q(\theta, v)$:

$$q(\theta, v) = \frac{d + a - d}{1 + \left(1 + \left(\frac{v}{c}\right)^b\right)^g}$$

where $\theta = [a, b, c, d, g]$ is a parameter vector. d is the maximum (or upper) asymptote, a is the minimum or lower asymptote, c is the inflection point, b is the hill slope (which controls the rate of rise at point c), and g is the asymmetric try factor. In order to help the STIR algorithm minimize the objective function, we need to derive

some limits on the parameters of the 5PLF model. We define the main limits on θ as

$$a_{\min} \le a \le a_{\max}, \quad b_{\min} \le b \le b_{\max}, \quad c_{\min} \le c \le c_{\max}, \quad d_{\min} \le d \le d_{\max}, \quad g_{\min} \le g \le g_{\max}$$

In order to determine the values for these boundaries, we use the Monotonicity Condition on $q(\theta, v)$ and observe that $c_{\min} = 0$, $g_{\min} = 0$, $a_{\max} = d_{\min}$, and $b_{\min} = 0$. Also, using the asymptotes condition of the 5PLF and letting $v \to 0$ we have $a_{\min} = 0$, and taking $v \to \infty$ we have d. To estimate d, we allow the value of d to fluctuate between $\pm 10\%$ of P_r, resulting in $d_{\min} = 0.9P_r$, $d_{\max} = 1.1P_r$, and $a_{\max} = 0.9P_r$. The 3 limits $b_{\max}$, $c_{\max}$, and $g_{\max}$ which were not specified in our work, we check their ratings from the power output of the chosen data. Thus our new equation for $q(\theta, v)$ becomes:

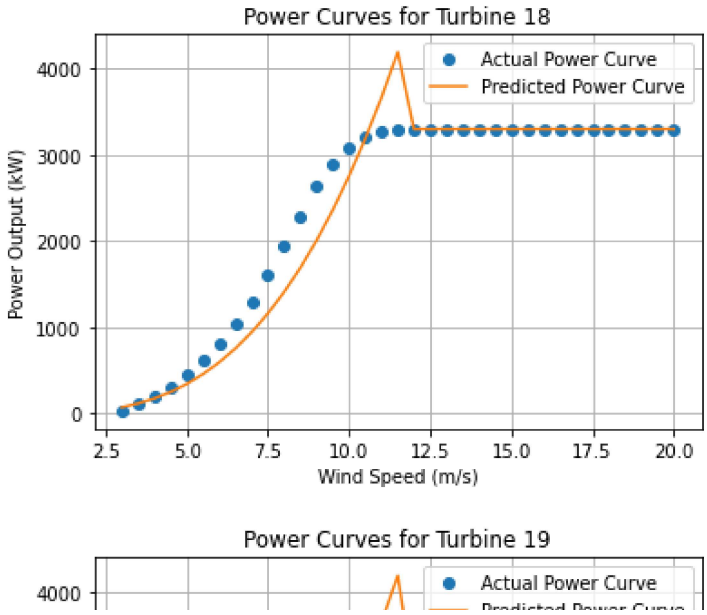
$$q(\theta, v) = \frac{d_n + a_n - d_n}{1 + \left(1 + \left(\frac{v}{c_n}\right)^{b_n}\right)^{g_n}}$$

Using the previous equations, we obtain:

$$P(v) = \begin{cases} 0 & \text{if } v < v_{ci} \text{ or } v > v_{co} \\ \dfrac{d_n + a_n - d_n}{1 + \left(1 + \left(\frac{v}{c_n}\right)^{b_n}\right)^{g_n}} & \text{if } v_{ci} \le v < v_r \\ P_r & \text{if } v_r \le v \le v_{co} \end{cases}$$

In [23]:

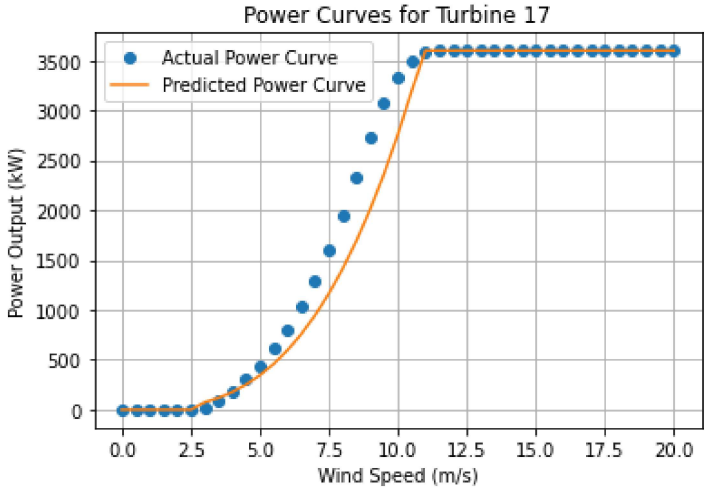
```
import matplotlib.pyplot as plt

# Define the parameters of the wind turbine model
v_ci = 3.0  # Cut-in wind speed (m/s)
v_r = 12.0  # Rated wind speed (m/s)
v_co = 26.0  # Cut-out wind speed (m/s)
rho = 1.225  # Air density (kg/m^3)
Cp_max = 0.45  # Rated power coefficient

df=Tdata1

# Iterate over each unique turbine_id
for turbine_id in df['turbine_id'].unique():
    # Create a new plot for the current turbine_id
    plt.figure()

    # Get the data for the current turbine_id
    turbine_data = df[df['turbine_id'] == turbine_id]

    # Extract the wind speeds and power values
    wind_speeds = turbine_data['power_curve_wind_speeds']
    power_values = turbine_data['power_curve_values']

    # Set P_r as the maximum rated power output in KW
    P_r = max(power_values)

    # Calculate the predicted power outputs for each wind speed
    power_outputs = []
    for wind_speed in wind_speeds:
        if wind_speed < v_ci:
            power_output = 0
        elif wind_speed < v_r:
            power_output = 0.5 * 10 * rho * Cp_max * wind_speed**3
        elif wind_speed <= v_co:
            power_output = P_r
        elif wind_speed > v_co:
            power_output = 0

        power_outputs.append(power_output)

    # Plot the actual and predicted power curves
    plt.plot(wind_speeds, power_values, 'o', label='Actual Power Curve')
    plt.plot(wind_speeds, power_outputs, label='Predicted Power Curve')

    # Set plot labels and title
    plt.xlabel('Wind Speed (m/s)')
    plt.ylabel('Power Output (kW)')
    plt.title(f'Power Curves for Turbine {turbine_id}')

    # Display legend and grid
    plt.legend()
    plt.grid(True)

    # Show the plot for the current turbine_id
    plt.show()
```



Testing our Model against the data using Cp_max (Rated power coefficient) as 0.45 and setting P_r (Rated power output) as the maximum power output of each turbine. Here we limit all calculated power output to P_r and Flatten the spikes to improve power output prediction

In [24]:

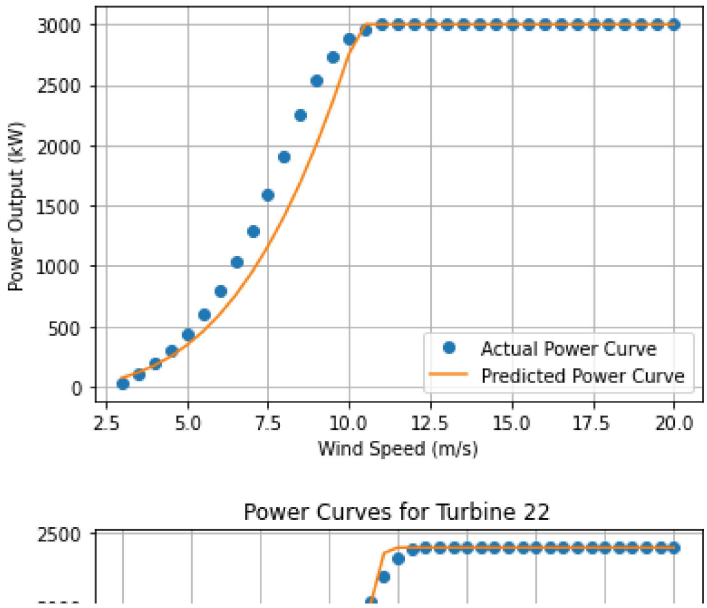
```
# Define the parameters of the wind turbine model
v_ci = 3.0  # Cut-in wind speed (m/s)
v_r = 12.0  # Rated wind speed (m/s)
v_co = 25.0  # Cut-out wind speed (m/s)
rho = 1.225  # Air density (kg/m^3)
Cp_max = 0.45  # Rated power coefficient

df=Tdata1

# Iterate over each unique turbine_id
for turbine_id in df['turbine_id'].unique():
    # Create a new plot for the current turbine_id
    plt.figure()

    # Get the data for the current turbine_id
    turbine_data = df[df['turbine_id'] == turbine_id]

    # Extract the wind speeds and power values
    wind_speeds = turbine_data['power_curve_wind_speeds']
    power_values = turbine_data['power_curve_values']

    # Set P_r as the maximum rated power output in KW
    P_r = max(power_values)

    # Calculate the predicted power outputs for each wind speed
    power_outputs = []
    for wind_speed in wind_speeds:
        if wind_speed < v_ci:
            power_output = 0
        elif wind_speed < v_r:
            power_output = 0.5 * 10 * rho * Cp_max * wind_speed**3
        elif wind_speed <= v_co:
            power_output = P_r
        elif wind_speed > v_co:
            power_output = 0

        power_output = min(power_output, P_r) # Flatten the spikes to improve power outp
        power_outputs.append(power_output)

    # Plot the actual and predicted power curves
    plt.plot(wind_speeds, power_values, 'o', label='Actual Power Curve')
    plt.plot(wind_speeds, power_outputs, label='Predicted Power Curve')

    # Set plot labels and title
    plt.xlabel('Wind Speed (m/s)')
    plt.ylabel('Power Output (kW)')
    plt.title(f'Power Curves for Turbine {turbine_id}')

    # Display legend and grid
    plt.legend()
    plt.grid(True)

    # Show the plot for the current turbine_id
    plt.show()
```



Type *Markdown* and LaTeX: α^2

In [25]:

```
import matplotlib.pyplot as plt

# Define the parameters of the wind turbine model
v_ci = 3.0  # Cut-in wind speed (m/s)
v_r = 12.0  # Rated wind speed (m/s)
v_co = 25.0  # Cut-out wind speed (m/s)
rho = 1.225  # Air density (kg/m^3)

df=Tdata1

# Iterate over each unique turbine_id
for turbine_id in df['turbine_id'].unique():
    # Create a new plot for the current turbine_id
    plt.figure()

    # Get the data for the current turbine_id
    turbine_data = df[df['turbine_id'] == turbine_id]

    # Extract the wind speeds and power values
    wind_speeds = turbine_data['power_curve_wind_speeds']
    power_values = turbine_data['power_curve_values']
    cp_max = turbine_data['Power_Coefficient_Values']

    # Set P_r as the maximum rated power output in KW
    P_r = max(power_values)

    # Calculate the predicted power outputs for each wind speed
    power_outputs = []
    for wind_speed in wind_speeds:
        if wind_speed < v_ci:
            power_output = 0
        elif wind_speed < v_r:
            power_output = 0.5  *10 * rho * Cp_max * wind_speed**3
        elif wind_speed <= v_co:
            power_output = P_r
        elif wind_speed > v_co:
            power_output = 0

        power_output = min(power_output, P_r) # Flatten the spikes to improve power outp
        power_outputs.append(power_output)  # Convert power output to kW

    # Plot the actual and predicted power curves
    plt.plot(wind_speeds, power_values, 'o', label='Actual Power Curve')
    plt.plot(wind_speeds, power_outputs, label='Predicted Power Curve')

    # Set plot labels and title
    plt.xlabel('Wind Speed (m/s)')
    plt.ylabel('Power Output (kW)')
    plt.title(f'Power Curves for Turbine {turbine_id}')

    # Display legend and grid
    plt.legend()
    plt.grid(True)

    # Show the plot for the current turbine_id
    plt.show()
```

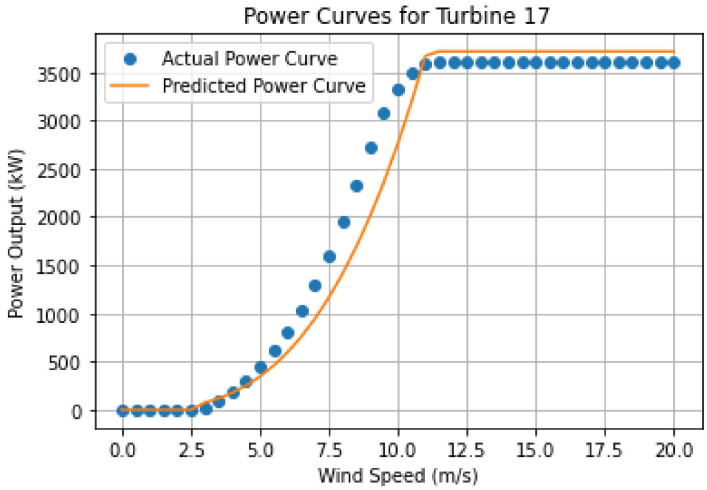


Using the Swept Area

Below we Test our Model against the data using Cp_max (Rated power coefficient) as 0.45 and setting P_r (Rated power output) as the swept area each turbine

In [26]:

```
# Define the parameters of the wind turbine model
v_ci = 3.0  # Cut-in wind speed (m/s)
v_r = 12.0  # Rated wind speed (m/s)
v_co = 25.0  # Cut-out wind speed (m/s)
rho = 1.225  # Air density (kg/m^3)
Cp_max = 0.45  # Rated power coefficient

df = Tdata1

# Iterate over each unique turbine_id
for turbine_id in df['turbine_id'].unique():
    # Create a new plot for the current turbine_id
    plt.figure()

    # Get the data for the current turbine_id
    turbine_data = df[df['turbine_id'] == turbine_id]

    # Extract the wind speeds, rotor swept area, and power values
    wind_speeds = turbine_data['power_curve_wind_speeds']
    power_values = turbine_data['power_curve_values']

    rotor_area = turbine_data['rotor_area']
    rotor_area= max(rotor_area)

    # Set P_r as the maximum rated power output in KW based on swept area
    P_r = 0.5 * Cp_max * rho * rotor_area


   # Calculate the predicted power outputs for each wind speed
    power_outputs = []
    for wind_speed in wind_speeds:
        if wind_speed < v_ci:
            power_output = 0
        elif wind_speed < v_r:
            power_output = 0.5  *10 * rho * Cp_max * wind_speed **3
        elif wind_speed <= v_co:
            power_output = P_r
        elif wind_speed > v_co:
            power_output = 0

        power_output = min(power_output, P_r) # Flatten the spikes to improve power outp
        power_outputs.append(power_output)  # Convert power output to kW

    # Plot the actual and predicted power curves
    plt.plot(wind_speeds, power_values, 'o', label='Actual Power Curve')
    plt.plot(wind_speeds, power_outputs, label='Predicted Power Curve')

    # Set plot labels and title
    plt.xlabel('Wind Speed (m/s)')
    plt.ylabel('Power Output (kW)')
    plt.title(f'Power Curves for Turbine {turbine_id}')

    # Display legend and grid
    plt.legend()
    plt.grid(True)

    # Show the plot for the current turbine_id
    plt.show()
```

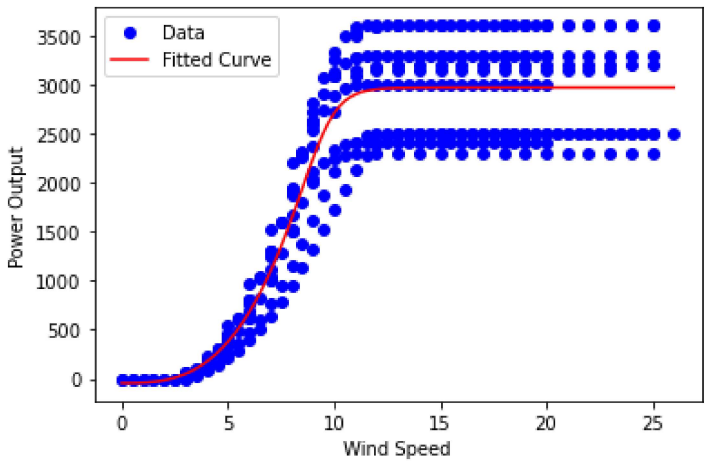

Power Curves for Turbine 17
Actual Power Curve
Predicted Power Curve
Power Output (kW)
Wind Speed (m/s)
0
500
1000
1500
2000
2500
3000
3500
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Power Curves for Turbine 18

In [27]:

```
# Load the dataset
data = Tdata1
dose = Tdata1['power_curve_wind_speeds']
response = Tdata1['power_curve_values']

# Define the 5PL model function
def fivepl_model(x, a, d, e, g, h):
    return a + (d - a) / (1 + (x / e) ** g) ** h

# Initial parameter guesses
initial_guess = [np.min(response), np.max(response), np.median(dose), 1, 1]

# Fit the model to the data
params, _ = curve_fit(fivepl_model, dose, response, p0=initial_guess)
a_fit, d_fit, e_fit, g_fit, h_fit = params

# Generate x values for the fitted curve
x_fit = np.linspace(np.min(dose), np.max(dose), 100)

# Calculate y values using the fitted parameters
y_fit = fivepl_model(x_fit, a_fit, d_fit, e_fit, g_fit, h_fit)

# Plot the data and the fitted curve
plt.plot(dose, response, 'bo', label='Data')
plt.plot(x_fit, y_fit, 'r-', label='Fitted Curve')
plt.xlabel('Wind Speed')
plt.ylabel('Power Output')
plt.legend()
plt.show()

# Calculate R-squared
residuals = response - fivepl_model(dose, a_fit, d_fit, e_fit, g_fit, h_fit)
ss_total = np.sum((response - np.mean(response)) ** 2)
ss_residual = np.sum(residuals ** 2)
r_squared = 1 - (ss_residual / ss_total)
print("R-squared:", r_squared)

# Predict the response for new doses
new_dose = np.array([1.0, 2.0, 3.0])  # Example new doses
predicted_response = fivepl_model(new_dose, a_fit, d_fit, e_fit, g_fit, h_fit)
```

```
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\1029153309.py:8: RuntimeWarning: d
ivide by zero encountered in power
  return a + (d - a) / (1 + (x / e) ** g) ** h
```

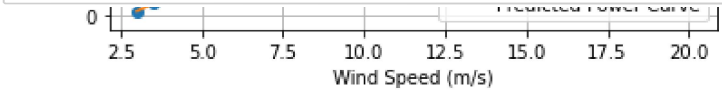



```
R-squared: 0.876999996749012
```

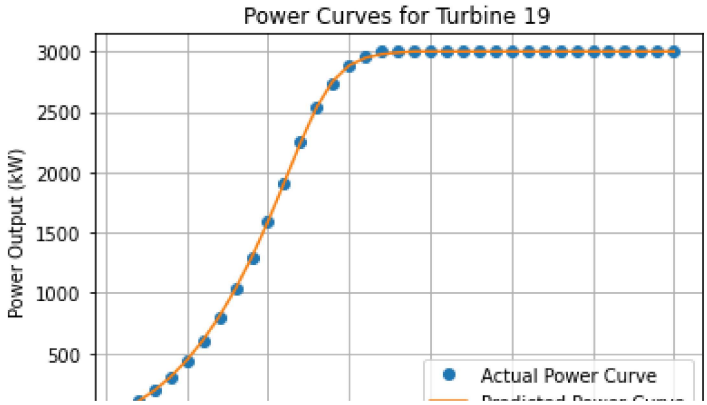
From the above, we see that the data affects the curve we get from the 5PL function. So we need to consider each turbine individually to determine accurate values for the 5 parameters and minimize the error of the prediction/expectation.

In [28]:

```
# Define the parameters of the wind turbine model
v_ci = 3.0  # Cut-in wind speed (m/s)
v_r = 12.0  # Rated wind speed (m/s)
v_co = 25.0  # Cut-out wind speed (m/s)
rho = 1.225  # Air density (kg/m^3)
Cp_max = 0.45  # Rated power coefficient

df = Tdata1

# Iterate over each unique turbine_id
for turbine_id in df['turbine_id'].unique():
    # Create a new plot for the current turbine_id
    plt.figure()

    # Get the data for the current turbine_id
    turbine_data = df[df['turbine_id'] == turbine_id]

    # Extract the wind speeds and power values
    wind_speeds = turbine_data['power_curve_wind_speeds']
    power_values = turbine_data['power_curve_values']

    # Fit the 5PL model to the data
    def fivepl_model(x, a, d, e, g, h):
        return a + (d - a) / (1 + (x / e) ** g) ** h

    initial_guess = [np.min(power_values), np.max(power_values), np.median(wind_speeds)
    params, _ = curve_fit(fivepl_model, wind_speeds, power_values, p0=initial_guess)
    a_fit, d_fit, e_fit, g_fit, h_fit = params

    print(a_fit, d_fit, e_fit, g_fit, h_fit)
    # Set P_r as the maximum rated power output in kW
    P_r = np.max(power_values)

    # Calculate the predicted power outputs for each wind speed
    power_outputs = []
    for wind_speed in wind_speeds:
        if wind_speed < v_ci:
            power_output = 0
        elif wind_speed < v_r:
            power_output = fivepl_model(wind_speed, a_fit, d_fit, e_fit, g_fit, h_fit)
        elif wind_speed <= v_co:
            power_output = P_r
        else:
            power_output = 0

        power_output = min(power_output, P_r)  # Flatten the spikes to improve power ou
        power_outputs.append(power_output)  # Convert power output to kW

    # Plot the actual and predicted power curves
    plt.plot(wind_speeds, power_values, 'o', label='Actual Power Curve')
    plt.plot(wind_speeds, power_outputs, label='Predicted Power Curve')

    # Set plot labels and title
    plt.xlabel('Wind Speed (m/s)')
    plt.ylabel('Power Output (kW)')
    plt.title(f'Power Curves for Turbine {turbine_id}')

    # Display legend and grid
    plt.legend()
    plt.grid(True)

    # Show the plot for the current turbine_id
    plt.show()
```



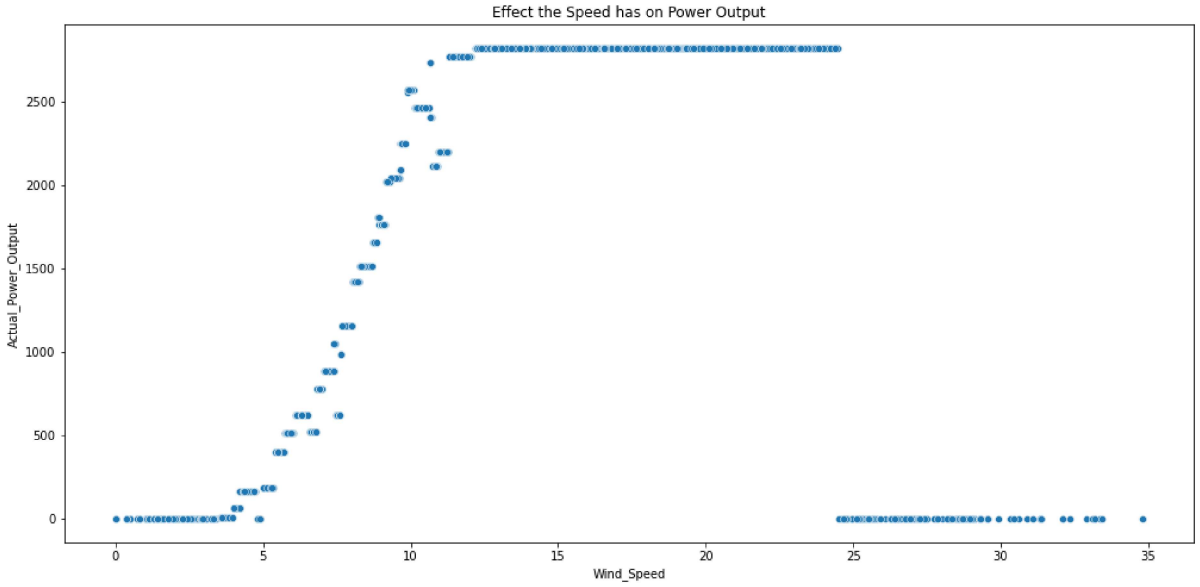

```
-92.36348834095605 3002.5529065443857 9.284579023201669 -17.461898274829068 0.16229
82134368792
```



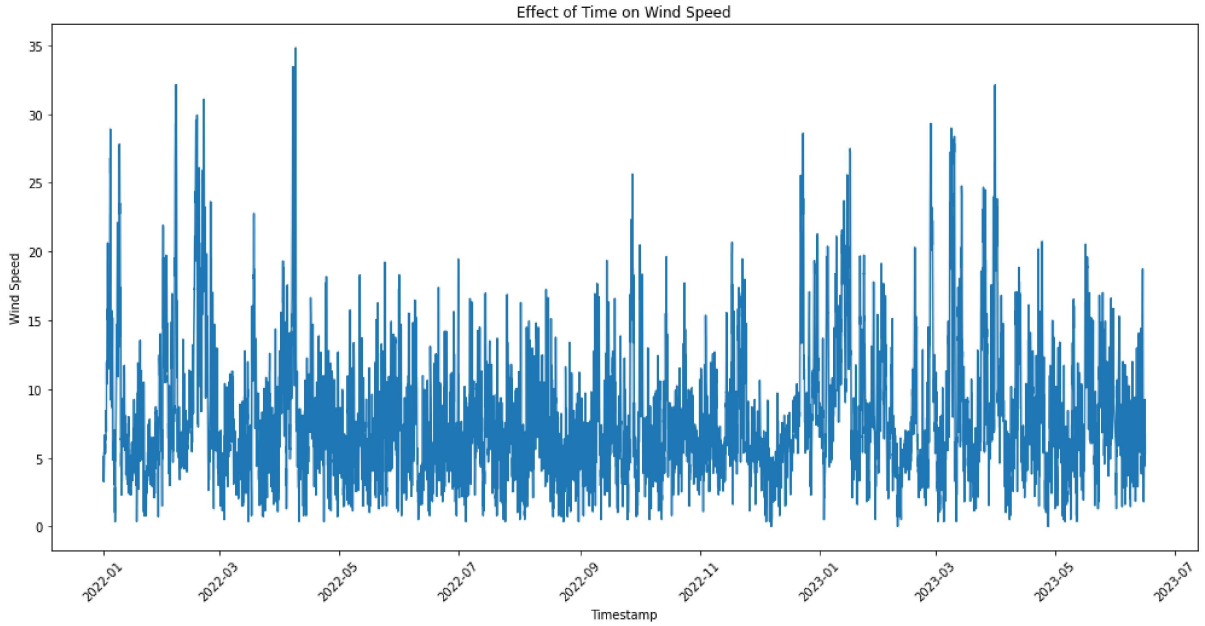
Tdata 2: Hamburg Wind Farm

In [29]:

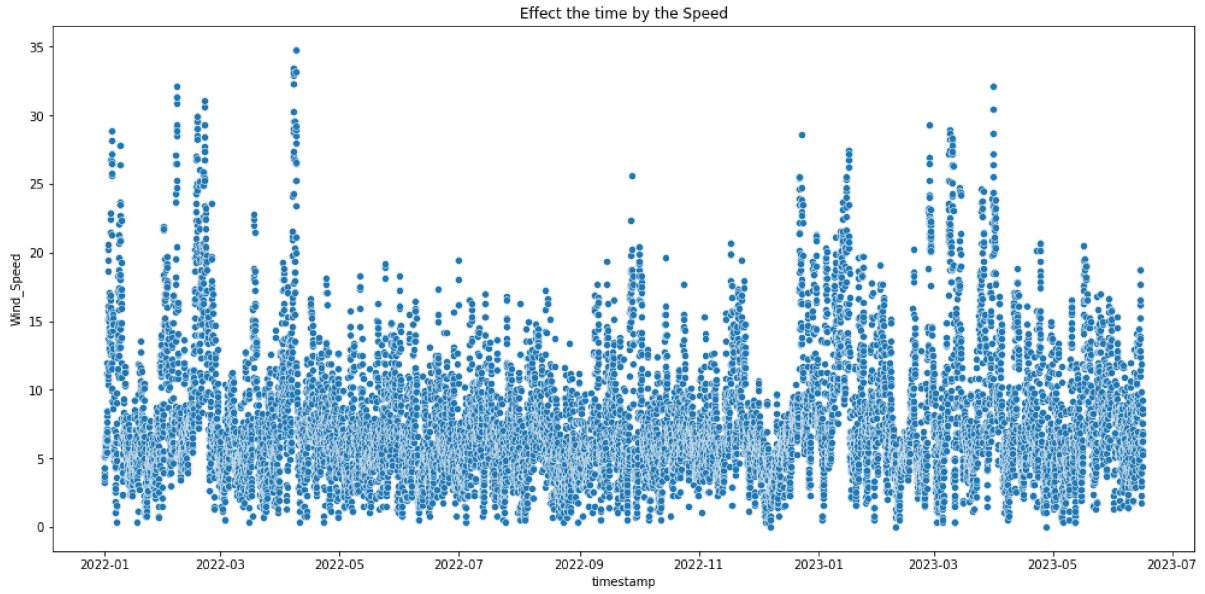
```
# the graph Actual Power Curve

plt.figure(figsize=(17,8))
sns.scatterplot(data=Tdata2,x="Wind_Speed",y="Actual_Power_Output")
plt.title("Effect the Speed has on Power Output")
plt.show()
```



In [30]:
```
plt.figure(figsize=(17, 8))

plt.plot(Tdata2['timestamp'], Tdata2['Wind_Speed'])

plt.title("Effect of Time on Wind Speed")
plt.xlabel("Timestamp")
plt.ylabel("Wind Speed")
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed


plt.show()
```



In [31]:
```
# the graph show effect the time by the Speed

plt.figure(figsize=(17,8))
sns.scatterplot(data=Tdata2,x="timestamp",y="Wind_Speed")
plt.title("Effect the time by the Speed")
plt.show()
```



In [32]:

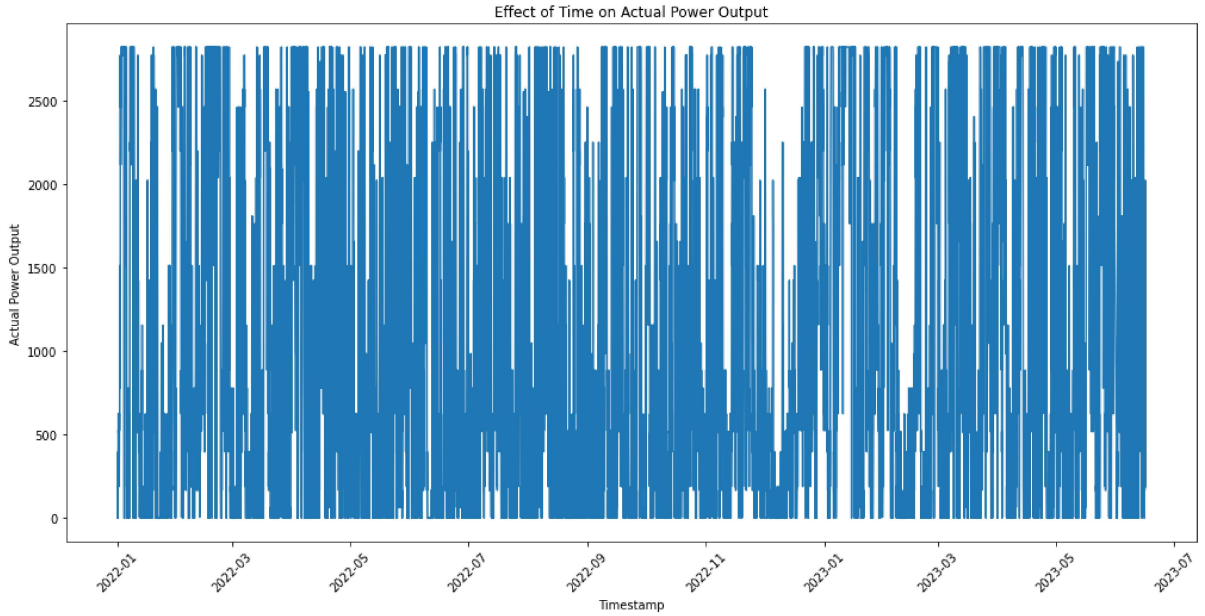
```
# Parameters
rho = 1.225  # Air density
Cp_max = 0.48  # Average power coefficient
v_ci = 3  # Cut-in wind speed
v_r = 12  # Rated wind speed
v_co = 25.0  # Cut-out wind speed
P_r = Tdata2['Actual_Power_Output'].max()  # Maximum actual power output
# P_r= 3000
# Calculate predicted power output
# Apply additional logic to handle wind speed ranges
power_outputs = []
for wind_speed in Tdata2['Wind_Speed']:
    if wind_speed < v_ci:
        power_output = 0
    elif wind_speed < v_r:
        power_output = 0.5 * 10 * rho * Cp_max * wind_speed ** 3
    elif wind_speed <= v_co:
        power_output = P_r
    elif wind_speed > v_co:
        power_output = 0

    power_output = min(power_output, P_r)  # Flatten the spikes to improve power output
    power_outputs.append(power_output)

# Assign the modified power outputs to the DataFrame
Tdata2['Predicted_Cubic_Power_Output'] = power_outputs

# Display the updated data
Tdata2.tail(15)
```

```
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\3498179207.py:26: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Tdata2['Predicted_Cubic_Power_Output'] = power_outputs
```

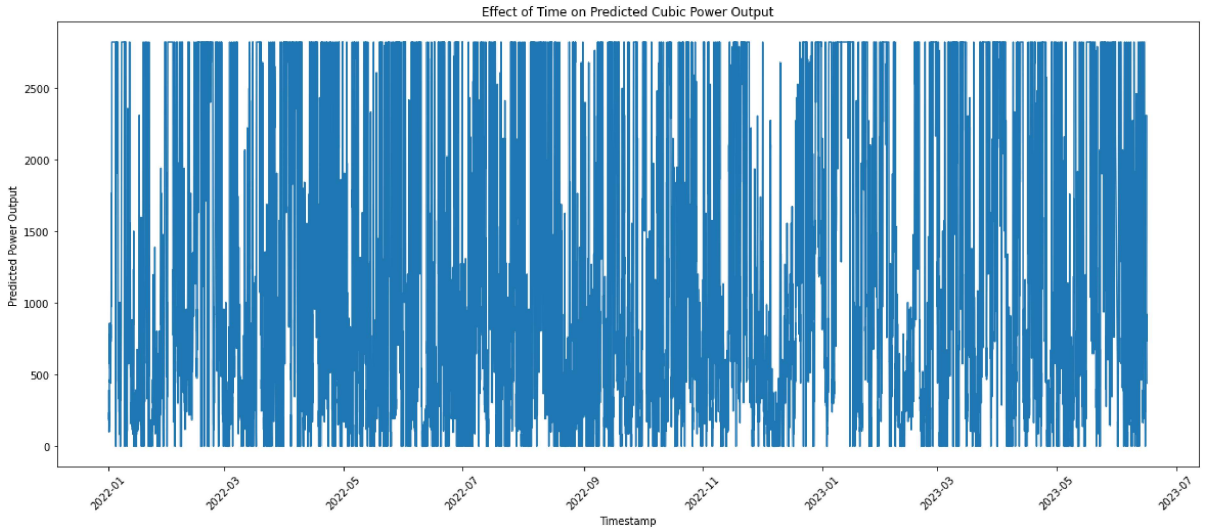
Out[32]:

	timestamp	Wind_Speed	Wind_Direction	Actual_Power_Output	Predicted_Cubic_Power_Output
12707	2023-06-15 11:00:00	3.98	5.19	5.22	184.84
12708	2023-06-15 12:00:00	6.29	23.63	624.07	730.64
12709	2023-06-15 13:00:00	5.24	15.95	189.79	423.41
12710	2023-06-15 14:00:00	4.38	9.46	166.90	246.97
12711	2023-06-15 15:00:00	5.51	11.31	397.65	491.00
12712	2023-06-15 16:00:00	6.92	8.97	779.03	976.24
12713	2023-06-15 17:00:00	7.59	5.44	624.72	1287.64
12714	2023-06-15 18:00:00	8.31	4.97	1511.74	1687.90
12715	2023-06-15 19:00:00	8.67	4.76	1511.74	1916.01
12716	2023-06-15 20:00:00	9.23	20.56	2022.32	2309.95
12717	2023-06-15 21:00:00	6.79	32.01	524.61	921.36
12718	2023-06-15 22:00:00	6.30	30.96	624.07	734.24
12719	2023-06-15 23:00:00	6.79	32.01	524.61	921.36
12720	2023-06-16 00:00:00	6.29	13.24	624.07	730.64
12721	2023-06-16 01:00:00	5.32	28.30	189.79	441.51

In [33]:
```
plt.figure(figsize=(17, 8))

plt.plot(Tdata2['timestamp'], Tdata2['Actual_Power_Output'])

plt.title("Effect of Time on Actual Power Output")
plt.xlabel("Timestamp")
plt.ylabel("Actual Power Output")
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed

plt.show()
```

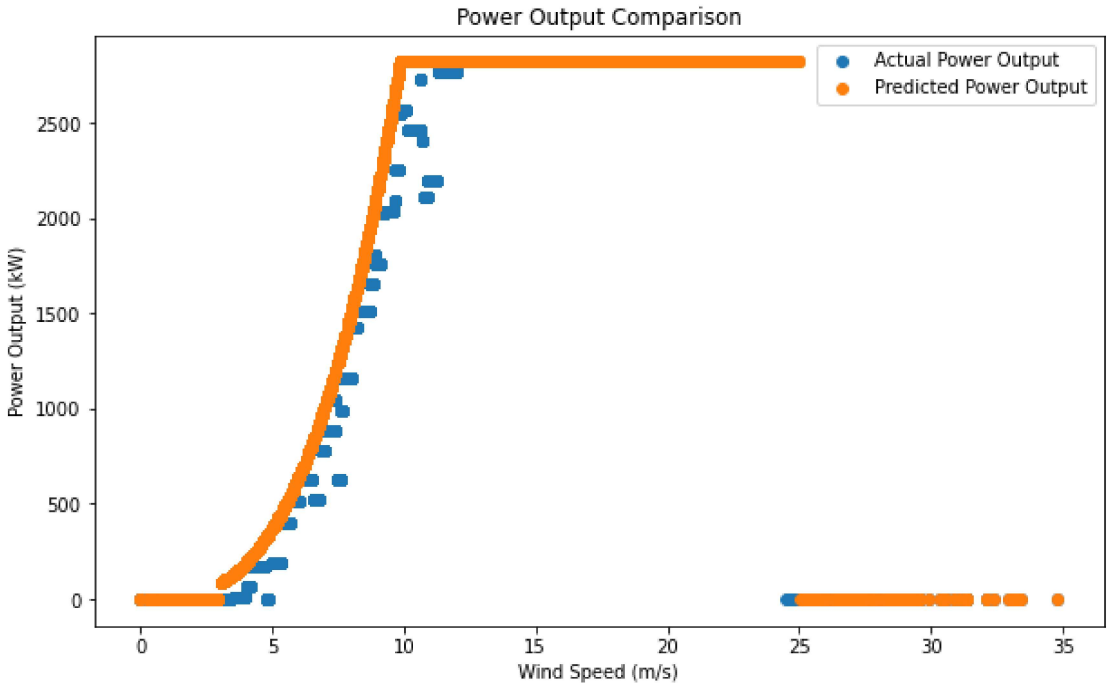


In [34]:
```
plt.figure(figsize=(20, 8))

plt.plot(Tdata2['timestamp'], Tdata2['Predicted_Cubic_Power_Output'])

plt.title("Effect of Time on Predicted Cubic Power Output")
plt.xlabel("Timestamp")
plt.ylabel("Predicted Power Output")
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed

plt.show()
```



In [35]:

```
plt.figure(figsize=(10, 6))

# Plotting the actual power output as a scatter plot
plt.scatter(Tdata2['Wind_Speed'], Tdata2['Actual_Power_Output'], label='Actual Power Ou
plt.scatter(Tdata2['Wind_Speed'], Tdata2["Predicted_Cubic_Power_Output"], label='Predic

plt.xlabel('Wind Speed (m/s)')
plt.ylabel('Power Output (kW)')
plt.title('Power Output Comparison')
plt.legend()
plt.show()
```



In [36]:

```
# Parameters
rho = 1.225  # Air density
r_annual = 0.018  # Annual Decay rate of Cp
r_hourly = r_annual / 8760  # Hourly Decay Rate of Cp
Cp_max = 0.48 # Initial power coefficient
v_ci = 3  # Cut-in wind speed
v_r = 12  # Rated wind speed
v_co = 25.0  # Cut-out wind speed
P_r = Tdata2['Actual_Power_Output'].max()  # Maximum actual power output

# Calculate predicted power output
# Apply additional logic to handle wind speed ranges
power_outputs = []
T = []
Cp_values = []
Power_Ratio = []
min_timestamp = Tdata2['timestamp'].min()
for wind_speed, timestamp in zip(Tdata2['Wind_Speed'], Tdata2['timestamp']):
    time_diff = (timestamp - min_timestamp).total_seconds() / 3600  # Calculate time di
    T.append(int(time_diff))

    Cp = Cp_max * (1 - r_hourly) ** T[-1]  # Calculate decayed Cp value based on the la
    Cp_formatted = "{:.5f}".format(Cp)  # Format Cp value with 5 decimal places
    Cp_values.append(Cp_formatted)  # Store the formatted Cp value

    if wind_speed < v_ci:
        power_output = 0
    elif wind_speed < v_r:
        power_output = 0.5 * 10 * rho * Cp * wind_speed ** 3
    elif wind_speed <= v_co:
        power_output = P_r
    elif wind_speed > v_co:
        power_output = 0

    power_output = min(power_output, P_r)  # Flatten the spikes to improve power output
    power_outputs.append(power_output)

# Assign the modified power outputs and Cp values to the DataFrame
Tdata2['Cp_values'] = Cp_values
Tdata2['Decaying_Cp_Power_Output'] = power_outputs
Tdata2['Actual_Power/Decaying_Cp_Power'] = Tdata2['Actual_Power_Output']/Tdata2['Decayi
K = Tdata2['Actual_Power/Decaying_Cp_Power'].mean()

Tdata2['K-Corrected_Decaying_Cp_Power']=K*Tdata2['Decaying_Cp_Power_Output']

# Display the updated data
Tdata2
```

```
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\566467168.py:39: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Tdata2['Cp_values'] = Cp_values
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\566467168.py:40: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Tdata2['Decaying_Cp_Power_Output'] = power_outputs
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\566467168.py:41: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Tdata2['Actual_Power/Decaying_Cp_Power'] = Tdata2['Actual_Power_Output']/Tdata2['Dec
aying_Cp_Power_Output']
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\566467168.py:44: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Tdata2['K-Corrected_Decaying_Cp_Power']=K*Tdata2['Decaying_Cp_Power_Output']
```

Out[36]:

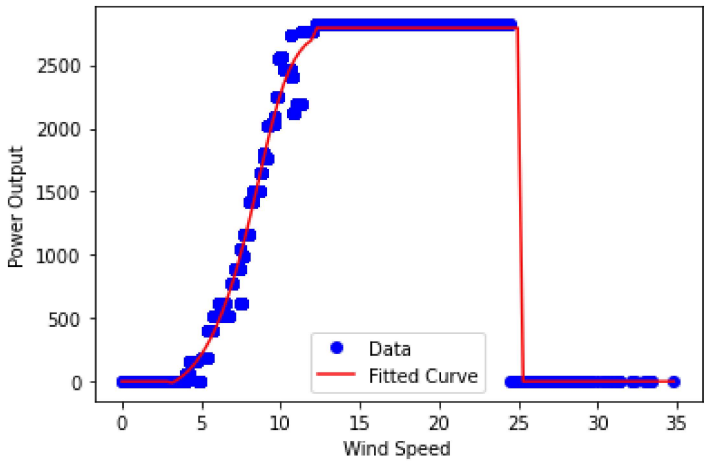
	timestamp	Wind_Speed	Wind_Direction	Actual_Power_Output	Predicted_Cubic_Power_Output	Cp_va
0	2022-01-01 00:00:00	5.09	188.13	189.79	387.97	0.4
1	2022-01-01 01:00:00	3.98	174.81	5.22	184.84	0.4
2	2022-01-01 02:00:00	4.32	180.00	166.90	237.03	0.4
3	2022-01-01 03:00:00	3.26	173.66	0.00	101.85	0.4
4	2022-01-01 04:00:00	3.55	156.04	5.22	131.04	0.4
...	...	...	...	...	...	
12717	2023-06-15 21:00:00	6.79	32.01	524.61	921.36	0.4
12718	2023-06-15 22:00:00	6.30	30.96	624.07	734.24	0.4
12719	2023-06-15 23:00:00	6.79	32.01	524.61	921.36	0.4
12720	2023-06-16 00:00:00	6.29	13.24	624.07	730.64	0.4
12721	2023-06-16 01:00:00	5.32	28.30	189.79	441.51	0.4

12681 rows × 9 columns

In [37]:

```
# Load the dataset
data = Tdata2
dose = Tdata2['Wind_Speed']
response = Tdata2['Actual_Power_Output']

# Define the 5PL model function with wind speed conditions
def fivepl_model(x, a, d, e, g, h, v_ci, v_r, v_co):
    y = np.zeros_like(x)
    for i, wind_speed in enumerate(x):
        if wind_speed < v_ci:
            y[i] = 0
        elif wind_speed < v_r:
            y[i] = a + (d - a) / (1 + (wind_speed / e) ** g) ** h
        elif wind_speed <= v_co:
            y[i] = d
        else:
            y[i] = 0
    return y

# Initial parameter guesses
initial_guess = [np.min(response), np.max(response), np.median(dose), 1, 1, 3.0, 12.0,

# Fit the model to the data
params, _ = curve_fit(fivepl_model, dose, response, p0=initial_guess)
a_fit, d_fit, e_fit, g_fit, h_fit, v_ci_fit, v_r_fit, v_co_fit = params

# Generate x values for the fitted curve
x_fit = np.linspace(np.min(dose), np.max(dose), 100)

# Calculate y values using the fitted parameters
y_fit = fivepl_model(x_fit, a_fit, d_fit, e_fit, g_fit, h_fit, v_ci_fit, v_r_fit, v_co_

# Calculate predicted power output for the dataset
power_outputs = fivepl_model(dose, a_fit, d_fit, e_fit, g_fit, h_fit, v_ci_fit, v_r_fit

# Assign the modified power outputs to the DataFrame
Tdata2['Predicted_5PL_Power_Output'] = power_outputs

# Plot the data and the fitted curve
plt.plot(dose, response, 'bo', label='Data')
plt.plot(x_fit, y_fit, 'r-', label='Fitted Curve')
plt.xlabel('Wind Speed')
plt.ylabel('Power Output')
plt.legend()
plt.show()

# Calculate R-squared
residuals = response - fivepl_model(dose, a_fit, d_fit, e_fit, g_fit, h_fit, v_ci_fit,
ss_total = np.sum((response - np.mean(response)) ** 2)
ss_residual = np.sum(residuals ** 2)
r_squared = 1 - (ss_residual / ss_total)
print("R-squared:", r_squared)

# Predict the response for new doses
new_dose = np.array([1.0, 2.0, 3.0])  # Example new doses
predicted_response = fivepl_model(new_dose, a_fit, d_fit, e_fit, g_fit, h_fit, v_ci_fit

print("Parameters:", a_fit, d_fit, e_fit, g_fit, h_fit, v_ci_fit, v_r_fit, v_co_fit)
print("Predicted Response:", predicted_response)
```

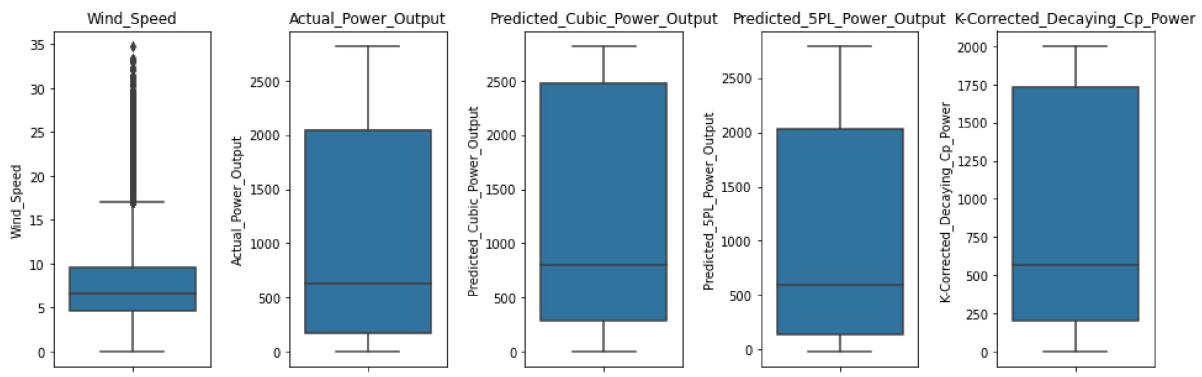
```
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\315957488.py:13: RuntimeWarning: i
nvalid value encountered in double_scalars
  y[i] = a + (d - a) / (1 + (wind_speed / e) ** g) ** h
C:\Users\Edukoya\anaconda3\lib\site-packages\scipy\optimize\minpack.py:833: OptimizeWa
rning: Covariance of the parameters could not be estimated
  warnings.warn('Covariance of the parameters could not be estimated',
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\315957488.py:37: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Tdata2['Predicted_5PL_Power_Output'] = power_outputs
```




```
R-squared: 0.9826397746641322
Parameters: -72.57276329304527 2793.3162454101703 10.02065720806091 -11.68992716015845
0.2846405873428665 3.0 12.0 25.0
Predicted Response: [  0.           0.         -20.76009296]
```

In [38]:

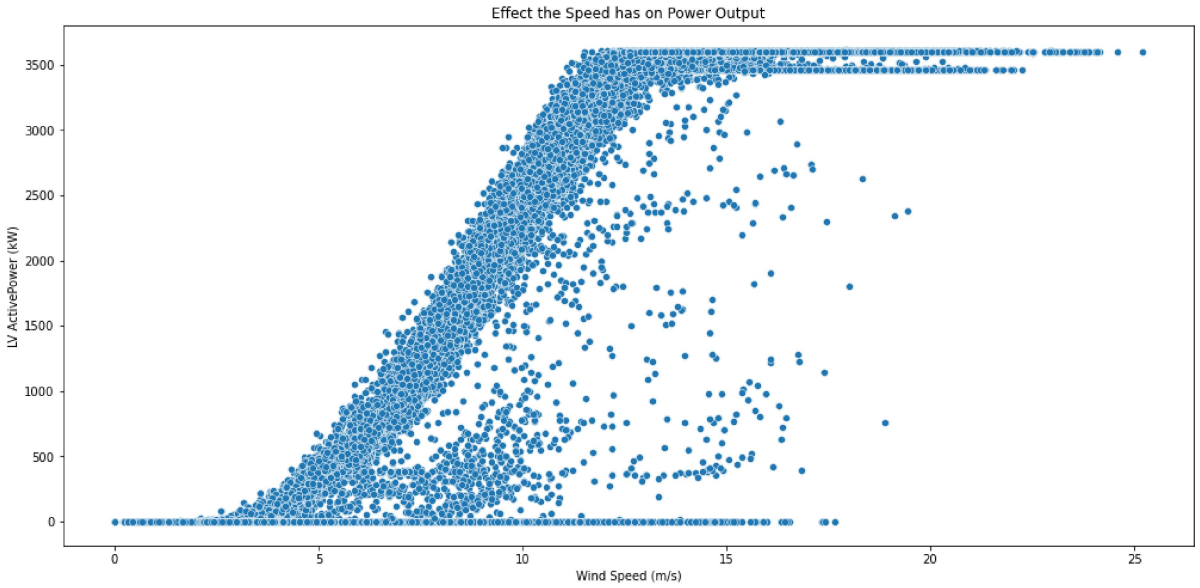
```
ind_Speed', 'Actual_Power_Output', 'Predicted_Cubic_Power_Output', 'Predicted_5PL_Power_

gsize=(16,5))
olumns:
ot(1,5,i)
ot(data = Tdata2, y= each)
(each)


adjust(wspace=0.5)  # Adjust the width space between subplots
```



Tdata3: Turkey Scada

In [39]:

```
# the graph show effect the time by the Speed

plt.figure(figsize=(17,8))
sns.scatterplot(data=Tdata3,x="Wind Speed (m/s)",y="LV ActivePower (kW)")
plt.title("Effect the Speed has on Power Output")
plt.show()
```



In [40]:

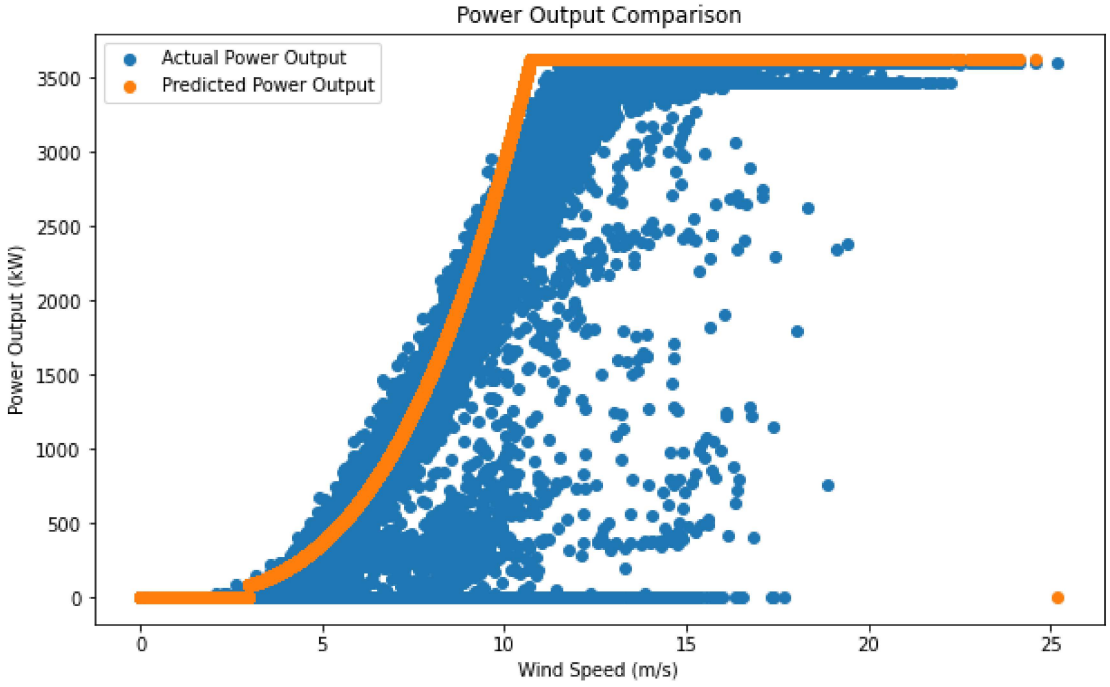
```
# Parameters
rho = 1.225  # Air density
Cp_max = 0.48  # Average power coefficient
v_ci = 3  # Cut-in wind speed
v_r = 12  # Rated wind speed
v_co = 25.0  # Cut-out wind speed
P_r = Tdata3['LV ActivePower (kW)'].max()  # Maximum actual power output

# Calculate predicted power output
# Apply additional logic to handle wind speed ranges
power_outputs = []
for wind_speed in Tdata3['Wind Speed (m/s)']:
    if wind_speed < v_ci:
        power_output = 0
    elif wind_speed < v_r:
        power_output = 0.5 * 10 * rho * Cp_max * wind_speed ** 3
    elif wind_speed <= v_co:
        power_output = P_r
    elif wind_speed > v_co:
        power_output = 0

    power_output = min(power_output, P_r)  # Flatten the spikes to improve power output
    power_outputs.append(power_output)

# Assign the modified power outputs to the DataFrame
Tdata3['Predicted_Power_Cubic_Output'] = power_outputs

# Display the updated data
Tdata3.tail(15)
```

Out[40]:

	Time	LV ActivePower (kW)	Wind Speed (m/s)	Predicted_Power_Cubic_Output
50515	31 12 2018 21:30	1814.36	8.47	1788.27
50516	31 12 2018 21:40	1992.35	8.43	1763.43
50517	31 12 2018 21:50	2554.38	9.98	2919.46
50518	31 12 2018 22:00	2681.27	10.42	3323.18
50519	31 12 2018 22:10	3019.89	10.71	3615.45
50520	31 12 2018 22:20	2771.11	10.15	3078.43
50521	31 12 2018 22:30	3333.82	12.07	3618.73
50522	31 12 2018 22:40	3455.28	12.20	3618.73
50523	31 12 2018 22:50	3429.02	12.49	3618.73
50524	31 12 2018 23:00	3514.27	12.56	3618.73
50525	31 12 2018 23:10	2963.98	11.40	3618.73
50526	31 12 2018 23:20	1684.35	7.33	1159.12
50527	31 12 2018 23:30	2201.11	8.44	1764.65
50528	31 12 2018 23:40	2515.69	9.42	2458.61
50529	31 12 2018 23:50	2820.47	9.98	2921.81

In [41]:

```
#Cubic Power Curve
plt.figure(figsize=(10, 6))

# Plotting the actual power output as a scatter plot
plt.scatter(Tdata3['Wind Speed (m/s)'], Tdata3['LV ActivePower (kW)'], label='Actual Pou
plt.scatter(Tdata3['Wind Speed (m/s)'], Tdata3['Predicted_Power_Cubic_Output'], label='

plt.xlabel('Wind Speed (m/s)')
plt.ylabel('Power Output (kW)')
plt.title('Power Output Comparison')
plt.legend()
plt.show()
```

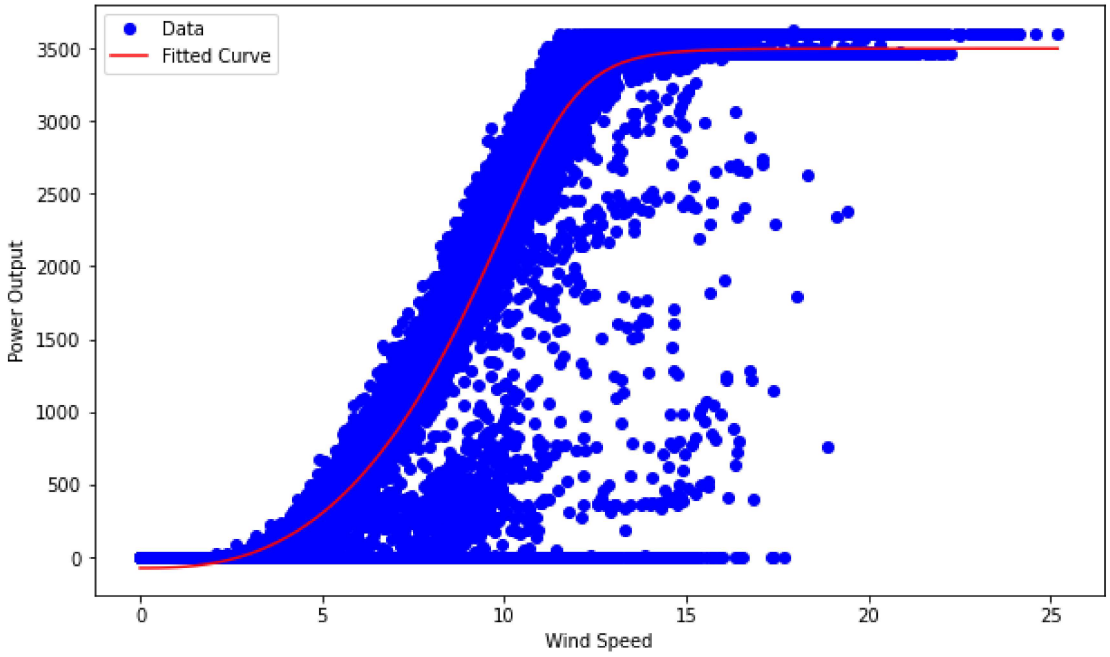


In [42]:

```
#5PL Power Curve

# Load the dataset
data = Tdata3
dose = Tdata3['Wind Speed (m/s)']
response = Tdata3['LV ActivePower (kW)']

# Define the 5PL model function
def fivepl_model(x, a, d, e, g, h):
    return a + (d - a) / (1 + (x / e) ** g) ** h

# Initial parameter guesses
initial_guess = [np.min(response), np.max(response), np.median(dose), 1, 1]

# Fit the model to the data
params, _ = curve_fit(fivepl_model, dose, response, p0=initial_guess)
a_fit, d_fit, e_fit, g_fit, h_fit = params

# Generate x values for the fitted curve
x_fit = np.linspace(np.min(dose), np.max(dose), 100)

# Calculate y values using the fitted parameters
y_fit = fivepl_model(x_fit, a_fit, d_fit, e_fit, g_fit, h_fit)
# Define the parameters of the wind turbine model
v_ci = 3.0  # Cut-in wind speed (m/s)
v_r = 12.0  # Rated wind speed (m/s)
v_co = 25.0  # Cut-out wind speed (m/s)

P_r = Tdata3['LV ActivePower (kW)'].max()  # Maximum actual power output

# Calculate predicted power output
# Apply additional logic to handle wind speed ranges
power_outputs = []
for wind_speed in Tdata3['Wind Speed (m/s)']:
    if wind_speed < v_ci:
        power_output = 0
    elif wind_speed < v_r:
        power_output = fivepl_model(wind_speed, a_fit, d_fit, e_fit, g_fit, h_fit)
    elif wind_speed <= v_co:
        power_output = P_r
    elif wind_speed > v_co:
        power_output = 0

    power_output = min(power_output, P_r)  # Flatten the spikes to improve power output
    power_outputs.append(power_output)

# Assign the modified power outputs to the DataFrame
Tdata3['Predicted_Power_5PL_Output'] = power_outputs
# Plot the data and the fitted curve

plt.figure(figsize=(10, 6))

plt.plot(dose, response, 'bo', label='Data')
plt.plot(x_fit, y_fit, 'r-', label='Fitted Curve')
plt.xlabel('Wind Speed')
plt.ylabel('Power Output')
plt.legend()
plt.show()

# Calculate R-squared
residuals = response - fivepl_model(dose, a_fit, d_fit, e_fit, g_fit, h_fit)
ss_total = np.sum((response - np.mean(response)) ** 2)
ss_residual = np.sum(residuals ** 2)
```

```
r_squared = 1 - (ss_residual / ss_total)
print("R-squared:", r_squared)

# Predict the response for new doses
new_dose = np.array([1.0, 2.0, 3.0])  # Example new doses
predicted_response = fivepl_model(new_dose, a_fit, d_fit, e_fit, g_fit, h_fit)

print(a_fit, d_fit, e_fit, g_fit, h_fit)
print("Predicted Response:", predicted_response)
```

```
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\4090972766.py:10: RuntimeWarning:
divide by zero encountered in power
  return a + (d - a) / (1 + (x / e) ** g) ** h
```



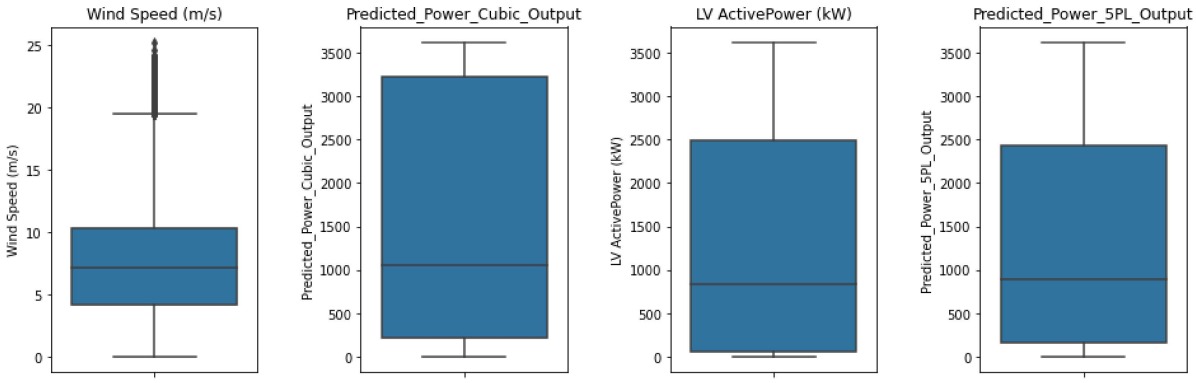

```
R-squared: 0.9102125137686787
-73.89206235492375 3496.9944212594028 11.617178678382068 -14.120686887753969 0.1890644
7068342747
Predicted Response: [-68.77223541 -41.31430837  22.2770326 ]
```

In [43]:
```
Tdata3.sample(15)
```

Out[43]:

	Time	LV ActivePower (kW)	Wind Speed (m/s)	Predicted_Power_Cubic_Output	Predicted_Power_5PL_Output
4559	06 02 2018 03:40	0.00	10.47	3372.49	2526.48
43664	10 11 2018 14:10	1005.30	7.31	1147.32	961.69
45227	25 11 2018 01:20	0.00	1.42	0.00	0.00
6770	21 02 2018 12:10	0.00	2.97	0.00	0.00
40239	17 10 2018 15:50	1728.38	8.60	1869.64	1521.48
31423	12 08 2018 11:10	2684.76	10.86	3618.73	2731.16
6715	21 02 2018 03:00	0.00	2.62	0.00	0.00
32271	18 08 2018 14:30	3190.45	11.86	3618.73	3140.90
43562	09 11 2018 21:10	533.51	5.62	521.88	439.97
9041	09 03 2018 06:40	0.00	0.46	0.00	0.00
7116	23 02 2018 21:50	1276.79	7.64	1310.50	1091.52
50005	28 12 2018 08:30	0.00	2.76	0.00	0.00
40880	22 10 2018 02:40	1383.06	8.06	1538.37	1269.46
35687	11 09 2018 08:00	0.00	1.93	0.00	0.00
22656	12 06 2018 07:30	2707.01	10.70	3596.73	2646.66

In [44]:
```
#Texas Data

columns = ['Wind Speed (m/s)', 'Predicted_Power_Cubic_Output', 'LV ActivePower (kW)', '
i=1
plt.figure(figsize=(16,5))
for each in columns:
    plt.subplot(1,4,i)
    sns.boxplot(data = Tdata3, y= each)
    plt.title(each)
    i += 1

plt.subplots_adjust(wspace=0.5)  # Adjust the width space between subplots
plt.show()
```



Calculating Mean Absolute Error

In [45]:
```
# Calculate mean absolute error
maec = mean_absolute_error(Tdata2['Actual_Power_Output'], Tdata2['Predicted_Cubic_Power_

# Calculate mean absolute error for 5PL

mae5 = mean_absolute_error(Tdata2['Actual_Power_Output'], Tdata2['Predicted_5PL_Power_O

# Display the mean error
print("Mean Error for Hamburg data:", maec, mae5)
```

```
Mean Error for Hamburg data: 168.9273722176038 70.42320122447751
```

In [46]:
```
# Calculate mean absolute error for Cubic Power curve Model
maec = mean_absolute_error(Tdata3['LV ActivePower (kW)'], Tdata3['Predicted_Power_Cubic_

# Calculate mean absolute error for 5PL model
mae5 = mean_absolute_error(Tdata3['LV ActivePower (kW)'], Tdata3['Predicted_Power_5PL_O


# Display the mean error
print("Mean Error for Texas data:", maec, mae5)
```

```
Mean Error for Texas data: 241.80473908617262 158.23845202712334
```

In [47]:

```
# Calculate the residuals
residuals = Tdata2['Actual_Power_Output'] - Tdata2['Predicted_Cubic_Power_Output']
residuals5 = Tdata2['Actual_Power_Output'] - Tdata2['Predicted_5PL_Power_Output']
# Calculate the squared residuals
squared_residuals = np.square(residuals)
squared_residuals5 = np.square(residuals5)
# Calculate the mean squared error (MSE)
mse = squared_residuals.mean()
mse5 = squared_residuals5.mean()
# Calculate the root mean square error (RMSE)
rmse = np.sqrt(mse)
rmse5 = np.sqrt(mse5)
# Append the residuals and RMSE to the DataFrame
Tdata2['Cubic_Residuals'] = residuals
Tdata2['5PL_Residuals'] = residuals5

# Display the RSME
print("The Root Mean Square Error for the Hamburg data using Cubic Poer curve and 5PL a

# Display the updated data
Tdata2.sample(10)
```

```
The Root Mean Square Error for the Hamburg data using Cubic Poer curve and 5PL are res
pectively: 245.82444135377216 137.73290373564453

C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\1358029544.py:14: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Tdata2['Cubic_Residuals'] = residuals
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\1358029544.py:15: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Tdata2['5PL_Residuals'] = residuals5
```

Out[47]:

er_Output	Cp_values	Decaying_Cp_Power_Output	Actual_Power/Decaying_Cp_Power	Corrected_Decaying_Cp_I
85.55	0.47439	84.55	0.00	
2820.47	0.47134	2820.47	1.00	2(
2820.47	0.47820	2820.47	1.00	2(
2820.47	0.46868	2820.47	1.00	2(
2820.47	0.46906	2820.47	0.87	2(
484.72	0.47516	479.83	0.83	3
2820.47	0.47866	2820.47	0.98	2(
1687.90	0.46850	1647.47	0.92	1'
2820.47	0.46767	2820.47	1.00	2(
1085.03	0.47417	1071.84	0.83	7

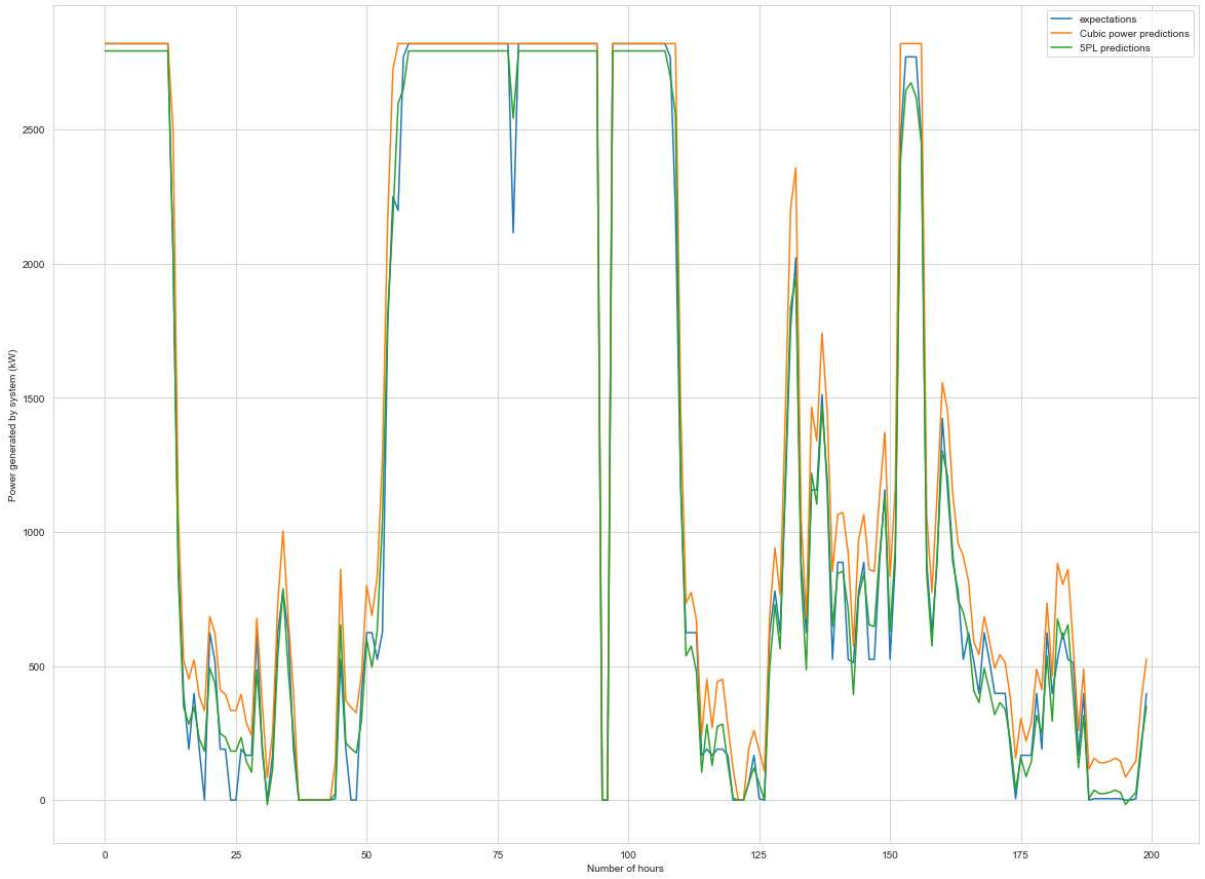
In [48]:

```
# Filter out rows with missing or invalid values
valid_data = Tdata2.dropna(subset=['Actual_Power_Output', 'Predicted_Cubic_Power_Output

expectations = np.array(valid_data['Actual_Power_Output'])
predictions = np.array(valid_data['Predicted_Cubic_Power_Output'])
predictions5= np.array(valid_data['Predicted_5PL_Power_Output'])
```

Final Prediction Plot for Tdata2 Hamburg Data

In [49]:

```
# line plot of observed vs predicted
sns.set_style("whitegrid")
plt.figure(figsize=(20,15))
plt.plot(expectations[100:300], label="expectations")
plt.plot(predictions[100:300], label="Cubic power predictions")
plt.plot(predictions5[100:300], label="5PL predictions")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW)")
plt.show()
```



In [50]:
```
Tdata2.tail(15)
```

Out[50]:

	timestamp	Wind_Speed	Wind_Direction	Actual_Power_Output	Predicted_Cubic_Power_Output	Cp_va
12707	2023-06-15 11:00:00	3.98	5.19	5.22	184.84	0.4
12708	2023-06-15 12:00:00	6.29	23.63	624.07	730.64	0.4
12709	2023-06-15 13:00:00	5.24	15.95	189.79	423.41	0.4
12710	2023-06-15 14:00:00	4.38	9.46	166.90	246.97	0.4
12711	2023-06-15 15:00:00	5.51	11.31	397.65	491.00	0.4
12712	2023-06-15 16:00:00	6.92	8.97	779.03	976.24	0.4
12713	2023-06-15 17:00:00	7.59	5.44	624.72	1287.64	0.4
12714	2023-06-15 18:00:00	8.31	4.97	1511.74	1687.90	0.4
12715	2023-06-15 19:00:00	8.67	4.76	1511.74	1916.01	0.4
12716	2023-06-15 20:00:00	9.23	20.56	2022.32	2309.95	0.4
12717	2023-06-15 21:00:00	6.79	32.01	524.61	921.36	0.4
12718	2023-06-15 22:00:00	6.30	30.96	624.07	734.24	0.4
12719	2023-06-15 23:00:00	6.79	32.01	524.61	921.36	0.4
12720	2023-06-16 00:00:00	6.29	13.24	624.07	730.64	0.4
12721	2023-06-16 01:00:00	5.32	28.30	189.79	441.51	0.4

Wind Power Prediction Model Calculator

Cubic Power Prediction

In [51]:

```
def cubic_power_output(wind_speed, v_ci, v_r, v_co, rho, Cp_max, P_r):

    v_ci = 3.0  # Cut-in wind speed (m/s)
    v_r = 12  # Rated wind speed (m/s)
    v_co = 25.0  # Cut-out wind speed (m/s)
    rho = 1.225  # Air density (kg/m^3)
    Cp_max = 0.45  # Approximated power coefficient
    P_r = 3000  # Rated power output (W) to be converted to fit data


    if wind_speed < v_ci:
            power_output = 0
    elif wind_speed < v_r:
            power_output = 0.5 * 10* rho * Cp_max * wind_speed**3
    elif wind_speed <= v_co:
            power_output = P_r
    elif windspeed > v_co:
            power_output = 0

    power_output = min(power_output, P_r) # Flatten the spikes to improve power output
    power_output = cubic_power_output(wind_speed, v_ci, v_r, v_co, rho, Cp_max)

    return power_output
wind_speed = float(input("Enter the wind speed (m/s): "))


print("Power Output: {} kW".format(power_output))
```

```
Enter the wind speed (m/s): 9
Power Output: 2256.850312561847 kW
```

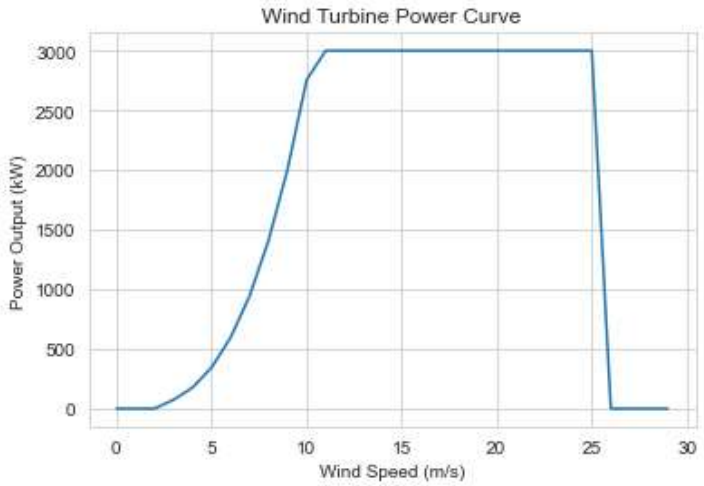
In [52]:

```
## Using 5PL prediction

def fivePL_power_output(wind_speed, v_ci, v_r, v_co, rho, Cp_max, P_r):

    v_ci = 3.0  # Cut-in wind speed (m/s)
    v_r = 12  # Rated wind speed (m/s)
    v_co = 25.0  # Cut-out wind speed (m/s)
    Cp_max = 0.45  # Approximated power coefficient
    P_r = 3000  # Rated power output (W) to be converted to fit data

    if wind_speed < v_ci:
            power_output = 0
    elif wind_speed < v_r:
            power_output = 48.101718895527256 + (2626.563931390021 - 48.101718895527256
    elif wind_speed <= v_co:
            power_output = P_r
    elif windspeed > v_co:
            power_output = 0

    power_output = min(power_output, P_r) # Flatten the spikes to improve power output
    power_output = fivePL_power_output(wind_speed, v_ci, v_r, v_co, Cp_max)

    return power_output

wind_speed = float(input("Enter the wind speed (m/s): "))


print("Power Output: {} kW".format(power_output))
```

```
Enter the wind speed (m/s): 9
Power Output: 2256.850312561847 kW
```

In [53]:

```
# Define the parameters of the wind turbine model
rated_power = 3000  # Rated power of the wind turbine in kilowatts
cut_in_speed = 3  # Wind speed at which the turbine starts operating in meters per secor
rated_speed = 12  # Rated wind speed for maximum power output in meters per second
cut_out_speed = 25  # Wind speed at which the turbine stops operating in meters per sec
Cp_max = 0.45  # Maximum power coefficient
rho = 1.225  # Air density (kg/m^3)

# Cubic Power Curve Calculation
def cubic_power_curve(wind_speed, rated_power, cut_in_speed, rated_speed, Cp_max, rho,
    if wind_speed < cut_in_speed or wind_speed > cut_out_speed:
        return 0
    elif wind_speed < rated_speed:
        return 0.5 * 10* rho * Cp_max * wind_speed**3
    else:
        return rated_power

# Generate wind speeds from 0 to 30 m/s
wind_speeds = np.arange(0, 30, 1)

# Calculate power outputs for each wind speed using the cubic power curve
power_outputs = [cubic_power_curve(wind_speed, rated_power, cut_in_speed, rated_speed,
power_outputs = np.minimum(power_outputs, rated_power)  # Flatten the spikes to improve


# Plot the power curve
plt.plot(wind_speeds, power_outputs)
plt.xlabel('Wind Speed (m/s)')
plt.ylabel('Power Output (kW)')
plt.title('Wind Turbine Power Curve')
plt.grid(True)
plt.show()
```



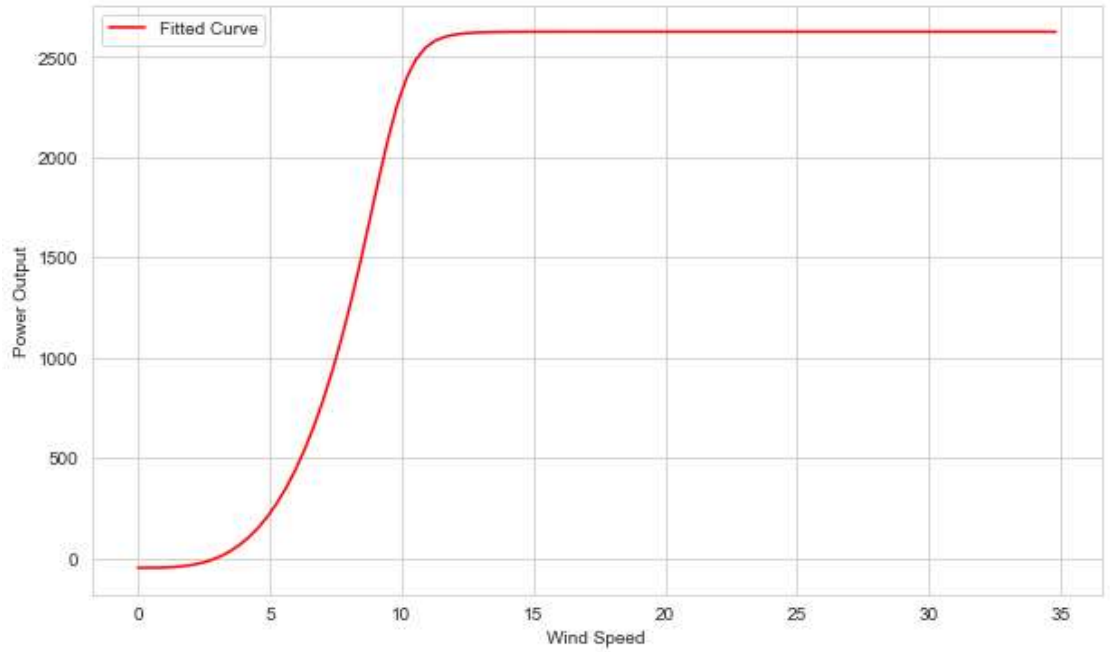
5PL Power Prediction

In [54]:

```
# Load the dataset
data = Tdata2
dose = data['Wind_Speed']
response = data['Actual_Power_Output']

# Define the 5PL model function
def fivepl_model(x, a, d, e, g, h):
    return a + (d - a) / (1 + (x / e) ** g) ** h

# Initial parameter guesses
initial_guess = [np.min(response), np.max(response), np.median(dose), 1, 1]

# Fit the model to the data
params, _ = curve_fit(fivepl_model, dose, response, p0=initial_guess)
a_fit, d_fit, e_fit, g_fit, h_fit = params

# Generate x values for the fitted curve
x_fit = np.linspace(np.min(dose), np.max(dose), 100)

# Calculate y values using the fitted parameters
y_fit = fivepl_model(x_fit, a_fit, d_fit, e_fit, g_fit, h_fit)

# Define the parameters of the wind turbine model
v_ci = 3.0  # Cut-in wind speed (m/s)
v_r = 12.0  # Rated wind speed (m/s)
v_co = 25.0  # Cut-out wind speed (m/s)
P_r = data['Actual_Power_Output'].max()  # Maximum actual power output

# Calculate predicted power output
power_outputs = []
for wind_speed in data['Wind_Speed']:
    if wind_speed < v_ci:
        power_output = 0
    elif wind_speed < v_r:
        power_output = fivepl_model(wind_speed, a_fit, d_fit, e_fit, g_fit, h_fit)
    elif wind_speed <= v_co:
        power_output = P_r
    else:
        power_output = 0

    power_output = min(power_output, P_r)  # Flatten the spikes to improve power output
    power_outputs.append(power_output)

# Assign the modified power outputs to the DataFrame
data['Predicted_5PL_Power_Output'] = power_outputs

# Plot the data and the fitted curve
plt.figure(figsize=(10, 6))
plt.plot(x_fit, y_fit, 'r-', label='Fitted Curve')
plt.xlabel('Wind Speed')
plt.ylabel('Power Output')
plt.legend()
plt.show()

# Calculate R-squared
residuals = response - fivepl_model(dose, a_fit, d_fit, e_fit, g_fit, h_fit)
ss_total = np.sum((response - np.mean(response)) ** 2)
ss_residual = np.sum(residuals ** 2)
r_squared = 1 - (ss_residual / ss_total)
print("R-squared:", r_squared)

# Predict the response for new doses
new_dose = np.array([1.0, 2.0, 3.0])  # Example new doses
```

```
predicted_response = fivepl_model(new_dose, a_fit, d_fit, e_fit, g_fit, h_fit)

print("Parameters:", a_fit, d_fit, e_fit, g_fit, h_fit)
print("Predicted Response:", predicted_response)
```

```
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\3732523031.py:8: RuntimeWarning: d
ivide by zero encountered in power
  return a + (d - a) / (1 + (x / e) ** g) ** h
C:\Users\Edukoya\AppData\Local\Temp\ipykernel_23556\3732523031.py:45: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Predicted_5PL_Power_Output'] = power_outputs
```




```
R-squared: 0.9219698282281301
Parameters: -48.101718895527256 2626.563931390021 9.967631207398195 -18.73421585882666
0.17740427689332583
Predicted Response: [-46.81815013 -35.25177094   1.34596499]
```

Notes

This was just a baseline trial of predicting the power output given wind speed and direction.

While power curves provide valuable information about the performance of a wind turbine, they have limitations and cannot accurately predict the power output at a specific average wind speed. Here are some pitfalls to consider when using power curves:

Wind Variability: Power curves are typically generated by collecting data over a range of wind speeds. However, if the wind at a specific site varies significantly or exhibits turbulent behavior, the power curve may not accurately represent the turbine's performance under those conditions.

Energy Content of Wind: The energy content of the wind is strongly dependent on wind speed. Power curves do not capture this variation adequately, as they only provide a snapshot of the turbine's performance at specific wind speeds. It's important to consider the wind speed distribution and how it affects the overall energy production.

Most Common Wind Speed vs. Available Energy: Power curves often show the turbine's performance at the most common wind speeds at a site. However, a significant amount of wind energy is available at higher wind speeds, typically around twice the most common wind speed. Power curves may not reflect the turbine's performance at these higher wind speeds.

Air Density Corrections: Power curves are typically generated under standard air pressure and temperature conditions. In practice, the operating conditions may deviate from these standards. To account for changes in air density, it's important to make corrections to the power curve when operating under non-standard conditions.

To mitigate these pitfalls and obtain more accurate predictions, advanced modeling techniques and site-specific data analysis are often employed. These approaches take into account factors such as wind speed distribution, turbulence intensity, wind directionality, and local environmental conditions to better estimate the power output of a wind turbine in specific operating conditions.

Cost Model: cost per unit of wind electricity produced

Levelized Cost of Energy (LCOE) Model using the annuity method

The Levelized Cost of Energy (LCOE) is a metric used to estimate the average cost of generating electricity over the lifetime of a power generation project. It represents the cost per unit of electricity produced and provides a standardized measure to compare different energy sources or projects.

We use the Levelized Cost of Energy (LCOE) for the Cost Model using the annuity method. The annuity method is one of the cost models used to calculate the LCOE (Levelized Cost of Energy). In the annuity method, the LCOE is determined by dividing the annualized investment and operating costs by the average electricity yield. It simplifies the calculation by considering the cash flows as equal annual payments over the project's lifetime.

The formula for calculating LCOE using the annuity method is:

$$LCOE = \frac{I_0 + \sum_{t=1}^{n} \frac{At}{(1+r)^t}}{\sum_{t=1}^{n} \frac{Mt,el}{(1+r)^t}}$$

where:

- I_0 is the investment expenditure
- A_t is the annual total cost in year t
- r is the discount rate or interest rate
- t is the year of the project's lifetime (1, 2, ..., n)
- $M_{t,el}$ is the electricity yield in year t
- n is the economic lifetime of the wind farm

The annuity method offers simplicity in calculation avoiding the complexity of cash flows and time value of money.

A simple LCOE Calculator for this:

This version of the calculator prompts the user to enter the required parameters, including investment expenditure, annual total costs, electricity yield, and the discount rate. It then calculates the LCOE using the annuity method based on the provided inputs and displays the result.

In [55]:

```python
def calculate_lcoe_annuity(I0, annual_costs, electricity_yield, discount_rate):
    """
    Calculate the Levelized Cost of Energy (LCOE) using the annuity method.

    Args:
        I0 (float): Investment expenditure.
        annual_costs (list): List of annual total costs.
        electricity_yield (list): List of electricity yield in each year.
        discount_rate (float): Discount rate or interest rate.

    Returns:
        float: Calculated LCOE.
    """
    numerator = I0 + sum([cost / ((1 + discount_rate) ** t) for t, cost in enumerate(an
    denominator = sum([yield_ / ((1 + discount_rate) ** t) for t, yield_ in enumerate(e

    lcoe = numerator / denominator
    return lcoe
```

In [56]:

```python
# Example inputs
I0 = 100000000  # Investment expenditure in EUR
annual_costs = [50000, 60000, 70000, 80000]  # List of annual total costs in EUR
electricity_yield = [1000000, 1200000, 1500000, 1800000]  # List of electricity yield i
discount_rate = 0.05  # Discount rate or interest rate

# Calculate LCOE using the annuity method
lcoe = calculate_lcoe_annuity(I0, annual_costs, electricity_yield, discount_rate)

# Print the calculated LCOE
print("LCOE using annuity method:", lcoe)
```

```
LCOE using annuity method: 20.805320738606042
```

In [57]:

```python
def calculator_lcoe_annuity():
    """
    Calculate the Levelized Cost of Energy (LCOE) using the annuity method.
    Requests input from the user for the required parameters.

    Returns:
        float: Calculated LCOE.
    """

    # Request input from the user
    I0 = float(input("Enter the investment expenditure in EUR: "))
    num_years = int(input("Enter the number of years for the project's lifetime (n): ")
    annual_costs = []
    electricity_yield = []

    for year in range(1, num_years + 1):
        try:
            cost = float(input(f"Enter the annual total operations and maintenance cost
            yield_ = float(input(f"Enter the electricity yield in year {year} in kWh pe
            annual_costs.append(cost)
            electricity_yield.append(yield_)

        except ValueError:
            print("Invalid input. Please enter numeric values.")
            exit(1)

    discount_rate = float(input("Enter the discount rate or interest rate (r) as a deci

    # Calculate LCOE using the annuity method
    numerator = I0 + sum([cost / ((1 + discount_rate) ** t) for t, cost in enumerate(an
    denominator = sum([yield_ / ((1 + discount_rate) ** t) for t, yield_ in enumerate(e

    # Check if denominator is zero
    if denominator == 0:
        print("Error: Division by zero. The calculation cannot be performed due to zero
        exit(1)

    lcoe = numerator / denominator

    return lcoe


# Calculate LCOE using the annuity method based on user input
lcoe_annuity = calculator_lcoe_annuity()

# Print the calculated LCOE
print("LCOE using annuity method:", lcoe_annuity)
```

```
Enter the investment expenditure in EUR: 10000000
Enter the number of years for the project's lifetime (n): 3
Enter the annual total operations and maintenance cost in year 1 in EUR: 100000
Enter the electricity yield in year 1 in kWh per year: 130000
Enter the annual total operations and maintenance cost in year 2 in EUR: 120000
Enter the electricity yield in year 2 in kWh per year: 160000
Enter the annual total operations and maintenance cost in year 3 in EUR: 150000
Enter the electricity yield in year 3 in kWh per year: 200000
Enter the discount rate or interest rate (r) as a decimal: 0.05
LCOE using annuity method: 23.395100963183886
```

Levelized Cost of Energy (LCOE) Model using the Annuity Method and

incorprating the Weighted Average Cost of Capital (WACC)

When the Weighted Average Cost of Capital (WACC) is used with the annuity method for calculating LCOE, it incorporates the financing costs and the mix of debt and equity into the analysis.

The WACC represents the average rate of return required by investors, taking into account the cost of debt and the cost of equity. It is used as the discount rate in the annuity method to calculate the present value of cash flows by incorporating the share of debt and equity into the analysis using the weighted average cost of capital (WACC). We consider the cost of debt, cost of equity, and the respective weights of debt and equity in the capital structure.

The formula to calculate the WACC is as follows:

$$WACC = (W_d * R_d) + (W_e * R_e)$$

where:

- W_d is the weight of debt in the capital structure (expressed as a decimal).
- R_d is the cost of debt (interest rate on debt).
- W_e is the weight of equity in the capital structure (expressed as a decimal).
- R_e is the cost of equity (required rate of return on equity).

Once we have the WACC, we use it as the discount rate (r) in the LCOE calculation.

The formula for calculating LCOE using the WACC and annuity method is similar to the basic annuity method, but the discount rate (r) is replaced with the WACC. The formula can be expressed as follows:

$$LCOE = \frac{I_0 + \sum_{t=1}^{n} \frac{At}{(1+WACC)^t}}{\sum_{t=1}^{n} \frac{Mt,el}{(1+WACC)^t}}$$

where:

- I_0 is the investment expenditure
- A_t is the annual total cost in year t
- WACC is the Weighted Average Cost of Capital
- t is the year of the project's lifetime (1, 2, ..., n)
- $M_{t,el}$ is the electricity yield in year t
- n is the economic lifetime of the wind farm

By incorporating the WACC, the LCOE calculation considers the financing structure of the project and reflects the cost of capital. It provides a more accurate assessment of the project's profitability and viability by factoring in the financing costs associated with the mix of debt and equity used to fund the wind farm.

A simple LCOE Calculator for this:

This version of the calculator prompts the user to enter the required parameters, including investment expenditure, annual total costs, electricity yield, weights of debt and equity, and costs of debt and equity. It then calculates the Weighted Average Cost of Capital (WACC) and uses it as the discount rate in the LCOE calculation using the annuity method.

Please note that the user inputs are assumed to be valid, and there is no input validation or error handling

In [58]:

```python
def calculate_lcoe_wacc():

    """
    Calculate the Levelized Cost of Energy (LCOE) using the annuity method and incorpora
    Cost of Capital (WACC).
    Requests input from the user for the required parameters.

    Returns:
        float: Calculated LCOE.
    """

    # Request input from the user
    I0 = float(input("Enter the investment expenditure (I0) in EUR: "))
    num_years = int(input("Enter the number of years for the project's lifetime (n): ")
    annual_costs = []
    electricity_yield = []


    # Get input for annual total costs and electricity yield for each year

    for year in range(1, num_years + 1):
        try:
            cost = float(input(f"Enter the annual total cost in year {year} in EUR: "))
            yield_ = float(input(f"Enter the electricity yield in year {year} in kWh per
            annual_costs.append(cost)
            electricity_yield.append(yield_)

        except ValueError:
            print("Invalid input. Please enter numeric values.")
            exit(1)

    weight_debt = float(input("Enter the weight of debt in the capital structure (Wd) a
    cost_debt = float(input("Enter the cost of debt (Rd) as a decimal: "))
    weight_equity = float(input("Enter the weight of equity in the capital structure (W
    cost_equity = float(input("Enter the cost of equity (Re) as a decimal: "))

    # Calculate the Weighted Average Cost of Capital (WACC)
    wacc = (weight_debt * cost_debt) + (weight_equity * cost_equity)

    # Calculate LCOE using the annuity method and WACC
    numerator = I0 + sum([cost / ((1 + wacc) ** t) for t, cost in enumerate(annual_cost
    denominator = sum([yield_ / ((1 + wacc) ** t) for t, yield_ in enumerate(electricit

    # Check if denominator is zero
    if denominator == 0:
        print("Error: Division by zero. The calculation cannot be performed due to zero
        exit(1)

    lcoe = numerator / denominator

    return lcoe


# Calculate LCOE using the annuity method and WACC based on user input
lcoe_wacc = calculate_lcoe_wacc()

# Print the calculated LCOE
print("LCOE using annuity method and WACC:", lcoe_wacc)
```

```
Enter the investment expenditure (I0) in EUR: 10000000
Enter the number of years for the project's lifetime (n): 3
Enter the annual total cost in year 1 in EUR: 100000
Enter the electricity yield in year 1 in kWh per year: 130000
Enter the annual total cost in year 2 in EUR: 120000
Enter the electricity yield in year 2 in kWh per year: 160000
Enter the annual total cost in year 3 in EUR: 150000
Enter the electricity yield in year 3 in kWh per year: 200000
Enter the weight of debt in the capital structure (Wd) as a decimal: 0.4
Enter the cost of debt (Rd) as a decimal: 0.05
Enter the weight of equity in the capital structure (We) as a decimal: 0.6
Enter the cost of equity (Re) as a decimal: 0.06
LCOE using annuity method and WACC: 23.668835584026148
```

This consideration of capital structure can help management decide the best way to finance the project at hand.

In [60]:
```
from subprocess import call
call(['jupyter', 'nbconvert', '--to', 'pdf', 'cubic-power-analysis-wind-turbine.ipynb'])
```

Out[60]: 1