turbine_erosion_analysis

July 1, 2021

0.0.1 Please note

Due to the sensitive nature of the original data, the dataset has been anonimized and each column (except for ID and DateTime) has been both multiplied and added with random noise. Internal and/or autocorrelation in the dataset is still present, though the absolute numbers may sometimes look a bit off to domain experts. Some bits of documentation have been removed as well. These often have to do with the characteristics of the turbines themselves. Some "magical" numbers to denote their performance might therefore appear to come out of the blue.

1 Detecting erosion in turbine blades

This notebook – a document in which you can see computer code along with its output and documentation – tries to answer the following question:

Can we see that the turbines are generating less energy in comparable external conditions if time progresses?

To answer this question, the following data sources where used:

1. SCADA data from three turbines (anonimized)
2. Dutch Offshore Wind Atlas
3. Experts from the organization that operates the turbines (anonimized)

The notebook can be read from top to bottom. In summary: the data is read, cleaned up and prepared for training of two machine learning models. Those models are then trained and evaluated. Lastly, the conclusions and further avenues of research are expanded upon.

2 Libraries

These are libraries (1st or 3rd party code) that we'll use throughout this notebook.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error

sns.set_theme()
```

3 Reading the data

Let's start with the SCADA data itself, and then match it with the wind data.

3.1 SCADA data

[2]:
```
df = pd.read_csv("processed_data/anonimized_turbine_data.csv", parse_dates =␣
↪["DateTime"])
display(df.head())
display(df.info())
```

```
   Unnamed: 0  turbineId            DateTime  \
0           0          1 2017-01-01 00:00:00
1           1          1 2017-01-01 00:10:00
2           2          1 2017-01-01 00:20:00
3           3          1 2017-01-01 00:30:00
4           4          1 2017-01-01 00:40:00

   Blade angle (pitch position) (°)  Capacity factor (%)  Current L1 / U (A)  \
0                         -0.472515             0.844930         1585.229435
1                          0.661520             0.902685         1692.747727
2                          4.158128             0.930028         1762.306084
3                          2.646081             0.929191         1766.790410
4                          1.323041             0.920867         1745.203369

   Current L2 / V (A)  Current L3 / W (A)  Data Availability (%)  \
0         2019.395496         2118.400349               0.764859
1         2155.276646         2261.788784               0.764859
2         2242.534799         2355.039430               0.764859
3         2248.448387         2361.926484               0.764859
4         2221.245945         2333.827438               0.764859

   Energy Export (kWh)  …  Rotor inverter temperature L3 (°C)  \
0           234.790181  …                            36.549650
1           253.294400  …                            36.549650
2           253.294400  …                            37.486821
3           259.462473  …                            37.486821
4           253.294400  …                            37.486821

   Rotor speed (RPM)  Time-based System Avail. (%)  Top controller temp. (°C)  \
0          16.880372                      0.860397                         NaN
1          16.880372                      0.860397                         NaN
2          16.880372                      0.860397                         NaN
3          16.880372                      0.860397                         NaN
4          16.880372                      0.860397                         NaN

   Virtual Production (kWh)  Voltage L1 / U (V)  Voltage L2 / V (V)  \
0                225.437879          551.269312          502.424628
```

```
1                243.180819          551.688030          502.678700
2                243.180819          546.245042          497.724571
3                249.095132          544.709829          496.073220
4                243.180819          546.105469          497.470538

   Voltage L3 / W (V)  Wind direction (°)  Wind speed (m/s)
0          343.489128          212.121123         14.880649
1          343.749648          210.731684         15.510894
2          340.189082          207.860169         16.771385
3          338.886428          207.582293         16.393238
4          339.928535          207.767548         16.015091

[5 rows x 56 columns]

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 473040 entries, 0 to 473039
Data columns (total 56 columns):
 #   Column                                        Non-Null Count   Dtype
---  ------                                        --------------   -----
 0   Unnamed: 0                                    473040 non-null  int64
 1   turbineId                                     473040 non-null  int64
 2   DateTime                                      473040 non-null
datetime64[ns]
 3   Blade angle (pitch position) (°)              457098 non-null  float64
 4   Capacity factor (%)                           457097 non-null  float64
 5   Current L1 / U (A)                            457099 non-null  float64
 6   Current L2 / V (A)                            457098 non-null  float64
 7   Current L3 / W (A)                            457099 non-null  float64
 8   Data Availability (%)                         473040 non-null  float64
 9   Energy Export (kWh)                           473040 non-null  float64
 10  Energy Theoretical (kWh)                      426275 non-null  float64
 11  First alarm in 10 min frame ()                354389 non-null  float64
 12  First alarm parameter 1 in 10 min frame ()    354389 non-null  float64
 13  First alarm parameter 2 in 10 min frame ()    354389 non-null  float64
 14  Gear bearing temp. (°C)                       457088 non-null  float64
 15  Gear oil temperature (°C)                     457089 non-null  float64
 16  Generator RPM (RPM)                           457098 non-null  float64
 17  Generator bearing front temperature (°C)      457089 non-null  float64
 18  Generator phase 1 temp (°C)                   457096 non-null  float64
 19  Generator phase 2 temp (°C)                   457096 non-null  float64
 20  Generator phase 3 temp (°C)                   457093 non-null  float64
 21  Generator slipring temp (°C)                  457092 non-null  float64
 22  Grid busbar temperature (°C)                  457088 non-null  float64
 23  Grid frequency (Hz)                           426168 non-null  float64
 24  Grid inverter temperature L1 (°C)             456956 non-null  float64
 25  Hub controller temp. (°C)                     269822 non-null  float64
 26  Hydraulic oil pressure (bar)                  269822 non-null  float64
 27  Lost Production (Contractual) (kWh)           424452 non-null  float64
```

```
 28  Lost Production (Time-based IEC B.2.2) (kWh)  423651 non-null  float64
 29  Lost Production Total (kWh)                   426275 non-null  float64
 30  Lost Production to Performance (kWh)          426275 non-null  float64
 31  Nacelle position (°)                          457084 non-null  float64
 32  Operating state ()                            354389 non-null  float64
 33  Operating sub state ()                        354389 non-null  float64
 34  Pending Operating State ()                    354389 non-null  float64
 35  Performance Index (%)                         411363 non-null  float64
 36  Phase 1 temperature (°C)                      457085 non-null  float64
 37  Phase 2 temperature (°C)                      457086 non-null  float64
 38  Phase 3 temperature (°C)                      457085 non-null  float64
 39  Power Reference (kW)                          269820 non-null  float64
 40  Power factor (cosphi) ()                      354389 non-null  float64
 41  Production-based Contractual Avail. (%)       424452 non-null  float64
 42  Production-based System Avail. (%)            426275 non-null  float64
 43  Reactive power (kvar)                         457099 non-null  float64
 44  Rotor inverter temperature L1 (°C)            457092 non-null  float64
 45  Rotor inverter temperature L2 (°C)            457091 non-null  float64
 46  Rotor inverter temperature L3 (°C)            457091 non-null  float64
 47  Rotor speed (RPM)                             457096 non-null  float64
 48  Time-based System Avail. (%)                  466420 non-null  float64
 49  Top controller temp. (°C)                     269822 non-null  float64
 50  Virtual Production (kWh)                      473040 non-null  float64
 51  Voltage L1 / U (V)                            457099 non-null  float64
 52  Voltage L2 / V (V)                            457099 non-null  float64
 53  Voltage L3 / W (V)                            457099 non-null  float64
 54  Wind direction (°)                            457091 non-null  float64
 55  Wind speed (m/s)                              426275 non-null  float64
dtypes: datetime64[ns](1), float64(53), int64(2)
memory usage: 202.1 MB
```

```
None
```

There are a lot of columns in the dataset. We will use the following ones:

- turbineId
- DateTime
- Blade angle
- Rotor speed
- Energy export
- Wind direction
- Wind speed

[3]:
```
df = pd.read_csv("processed_data/anonimized_turbine_data.csv", parse_dates =␣
↪["DateTime"], usecols = ["turbineId", "DateTime", "Blade angle (pitch␣
↪position) (°)", "Rotor speed (RPM)", "Energy Export (kWh)", "Wind direction␣
↪(°)", "Wind speed (m/s)"])
df.head()
```

[3]:
```
   turbineId             DateTime  Blade angle (pitch position) (°)  \
0          1  2017-01-01 00:00:00                          -0.472515
1          1  2017-01-01 00:10:00                           0.661520
2          1  2017-01-01 00:20:00                           4.158128
3          1  2017-01-01 00:30:00                           2.646081
4          1  2017-01-01 00:40:00                           1.323041

   Energy Export (kWh)  Rotor speed (RPM)  Wind direction (°)  \
0           234.790181          16.880372          212.121123
1           253.294400          16.880372          210.731684
2           253.294400          16.880372          207.860169
3           259.462473          16.880372          207.582293
4           253.294400          16.880372          207.767548

   Wind speed (m/s)
0         14.880649
1         15.510894
2         16.771385
3         16.393238
4         16.015091
```

3.2 Splitting the data for the three turbines

There are three (different) turbines in the dataset. Let's split the dataset into three, based on the `turbineId`.

[4]:
```
df_tub_1 = df[df["turbineId"] == 1]
df_tub_2 = df[df["turbineId"] == 2]
df_tub_3 = df[df["turbineId"] == 3]
```

3.3 Wind data

The wind data is all measured in the Dutch Offshore Wind Atlas.

[5]:
```
wind_df = pd.read_csv("processed_data/dowa-2008-2018-hourly.csv",␣
 ↪parse_dates=["Datetime"])
wind_df = wind_df.append(pd.read_csv("processed_data/dowa-2018-2019-hourly.
 ↪csv", parse_dates=["Datetime"]))
wind_df = wind_df.reset_index()
wind_df = wind_df.drop(["Unnamed: 0", "index"], axis="columns")
wind_df
```

[5]:
```
                 Datetime  wspeed (m/s)  wdir (deg)  air pressure (Pa)
0     2008-01-01 00:00:00      3.017058  139.518890      102050.476562
1     2008-01-01 01:00:00      3.160662  136.922974      102027.328125
2     2008-01-01 02:00:00      3.601591  122.715172      102004.906250
3     2008-01-01 03:00:00      4.507411  113.727936      101968.695312
4     2008-01-01 04:00:00      4.215354  116.094360      101947.960938
```

```
…                    …           …          …                …
96429 2018-12-31 20:00:00     10.395692  259.509277      102488.523438
96430 2018-12-31 21:00:00     10.335975  271.005768      102464.070312
96431 2018-12-31 22:00:00     10.146530  264.667816      102330.890625
96432 2018-12-31 23:00:00     11.404872  265.993774      102309.804688
96433 2019-01-01 00:00:00     11.968995  265.476349      102273.820312

[96434 rows x 4 columns]
```

3.3.1 Removing unnecessary data

The wind data starts from 2014-01-01, which is unnecessary since the SCADA data starts from 2017-01-01. So let's remove all the data before that.

[6]:
```
start_date = pd.to_datetime("2017-01-01 00:00:00")
wind_df = wind_df[wind_df["Datetime"] >= start_date].reset_index()
wind_df = wind_df.drop("index", axis="columns") # Make sure we have an ID-like␣
↪index instead of it being the old index
wind_df = wind_df.sort_values(by="Datetime")
wind_df
```

[6]:
```
                 Datetime  wspeed (m/s)  wdir (deg)  air pressure (Pa)
0     2017-01-01 00:00:00     13.368198  231.824738      101490.335938
1     2017-01-01 01:00:00     12.213772  233.784348      101377.070312
2     2017-01-01 02:00:00     12.209412  231.856689      101306.453125
3     2017-01-01 03:00:00     12.360149  228.361801      101272.976562
4     2017-01-01 04:00:00     12.777927  229.301239      101162.203125
…                    …           …          …                …
17517 2018-12-31 20:00:00     10.395692  259.509277      102488.523438
17518 2018-12-31 21:00:00     10.335975  271.005768      102464.070312
17519 2018-12-31 22:00:00     10.146530  264.667816      102330.890625
17520 2018-12-31 23:00:00     11.404872  265.993774      102309.804688
17521 2019-01-01 00:00:00     11.968995  265.476349      102273.820312

[17522 rows x 4 columns]
```

3.4 Averaging the SCADA data

Wind data is only available per hour. It is the mean of all the measurements in that hour. So a wind speed of 8.0 m/s at 14:00 means that between 13:00 and 14:00, the average wind speed was 8.0 m/s.

Let's aggregate the SCADA data according to the same principle as above so it nicely fits the wind data timestamps. This means we will average all columns except for the turbineId and Energy Export. The ID because it is an ID, so we can take the first one of the hour. The Energy Export because it is a cumulative value over time. As such, we will add it to its previous value.

[7]:
```
def resample_df_on_hour(dataframe, column_name):
    return dataframe.groupby(
        pd.Grouper(key = "DateTime", freq = "H")).agg({
            "turbineId": "first",
            "Blade angle (pitch position) (°)": "mean",
            "Energy Export (kWh)": "sum",
            "Rotor speed (RPM)": "mean",
            "Wind direction (°)": "mean",
            "Wind speed (m/s)": "mean"
        }).reset_index()

df_tub_1 = resample_df_on_hour(df_tub_1, "DateTime")
df_tub_2 = resample_df_on_hour(df_tub_2, "DateTime")
df_tub_3 = resample_df_on_hour(df_tub_3, "DateTime")

df_tub_1.head()
```

[7]:

	DateTime	turbineId	Blade angle (pitch position) (°)	Energy Export (kWh)	Rotor speed (RPM)	Wind direction (°)	Wind speed (m/s)
0	2017-01-01 00:00:00	1	1.606549	1507.430254	16.880372	209.218738	15.910050
1	2017-01-01 01:00:00	1	1.386043	1507.430254	16.880372	208.091750	15.868034
2	2017-01-01 02:00:00	1	0.488265	1470.421816	16.880372	202.642056	15.510895
3	2017-01-01 03:00:00	1	4.646393	1476.589889	16.880372	201.298931	17.086508
4	2017-01-01 04:00:00	1	3.638362	1532.102546	16.880372	206.594241	16.813402

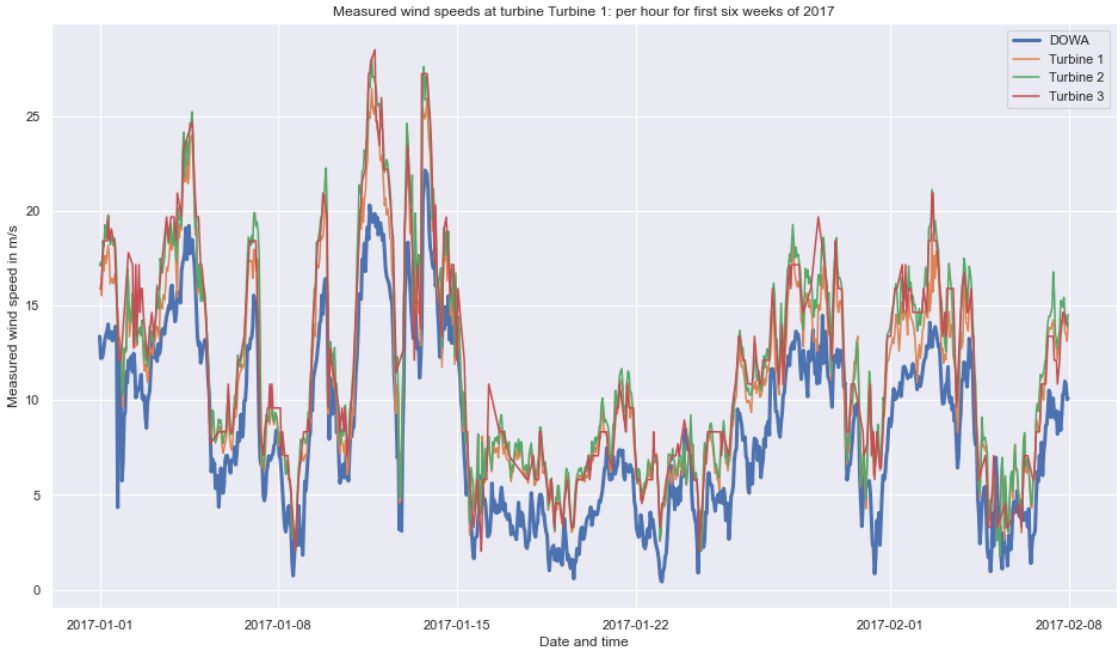
4 Comparison of SCADA data vs DOWA

The data mostly lines up nicely*. The organization has regularly said that the "SCADA wind data is unreliable at best".

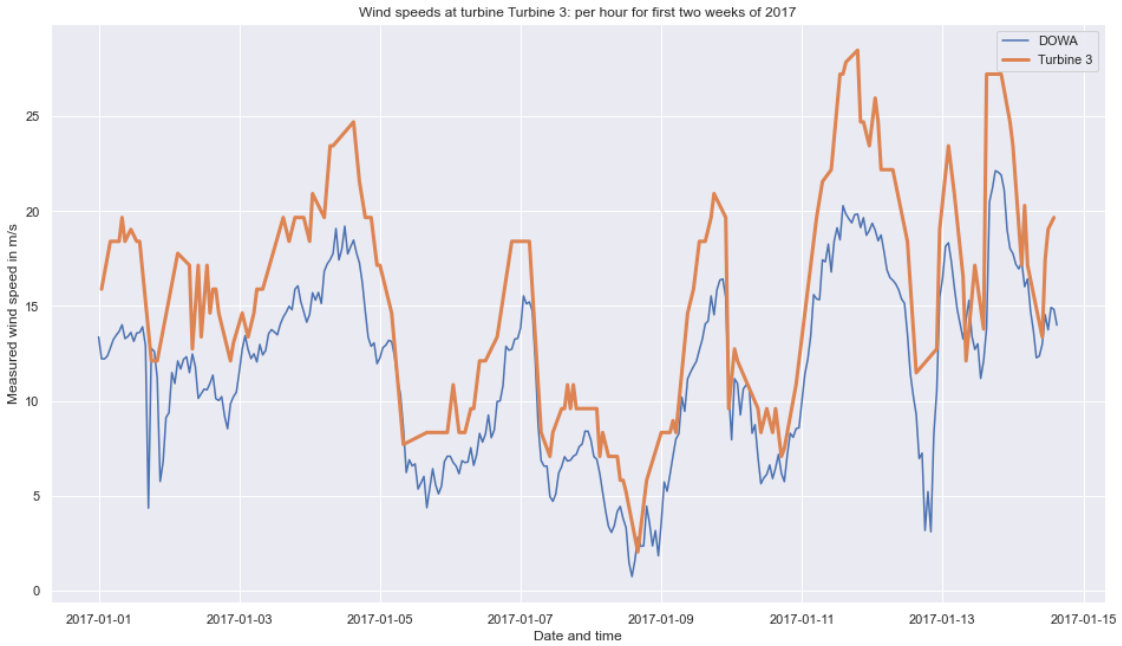
DOWA is proved to be accurate within 0.5 m/s at the hour level. From now on, let's only use the DOWA data for wind measurements.

*Due to the anomization of the data, the results are a bit off here.

[8]:
```
plt.figure(figsize=(16,9))
plt.title('Measured wind speeds at turbine Turbine 1: per hour for first six␣
 ↪weeks of 2017')
plt.xlabel('Date and time')
plt.ylabel('Measured wind speed in m/s')
sns.lineplot(x = "Datetime", y = "wspeed (m/s)", data = wind_df.head(912),␣
 ↪label = "DOWA", linewidth=3)
sns.lineplot(x = "DateTime", y = "Wind speed (m/s)", data = df_tub_1.head(912),␣
 ↪label="Turbine 1")
sns.lineplot(x = "DateTime", y = "Wind speed (m/s)", data = df_tub_2.head(912),␣
 ↪label="Turbine 2")
sns.lineplot(x = "DateTime", y = "Wind speed (m/s)", data = df_tub_3.head(912),␣
 ↪label="Turbine 3")
plt.show()
```



[9]:
```
plt.figure(figsize=(16,9))
plt.title('Wind speeds at turbine Turbine 3: per hour for first two weeks of␣
 ↪2017')
plt.xlabel('Date and time')
plt.ylabel('Measured wind speed in m/s')
sns.lineplot(x = "Datetime", y = "wspeed (m/s)", data = wind_df.head(328),␣
 ↪label = "DOWA")
sns.lineplot(x = "DateTime", y = "Wind speed (m/s)", data = df_tub_3.head(328),␣
 ↪label="Turbine 3", linewidth=3)
plt.show()
```



5 Merging the datasets

Let's merge the two datasets:

1. Remove the wind data from the SCADA dataset
2. Define a key to merge on: Datetime
3. Merge the two sets so we have 1 set with all the data

Important: we're using an inner join, which means that any timestamps that do not occur in both datasets will be removed. Practically: all data from 2019 is not used for the rest of this analysis.

[10]:
```
def remove_rename_merge_scada_with_dowa(scada_dataframe, dowa_dataframe):
    scada_dataframe = scada_dataframe.drop(columns = ["Wind direction (°)",␣
 ↪"Wind speed (m/s)"])
    scada_dataframe = scada_dataframe.rename(columns = {"DateTime": "Datetime"})
    return pd.merge(scada_dataframe, dowa_dataframe, on="Datetime")

df_tub_1 = remove_rename_merge_scada_with_dowa(df_tub_1, wind_df)
df_tub_2 = remove_rename_merge_scada_with_dowa(df_tub_2, wind_df)
df_tub_3 = remove_rename_merge_scada_with_dowa(df_tub_3, wind_df)
```

[11]:
```
display(df_tub_1.head())
display(df_tub_1.tail())
```

```
             Datetime  turbineId  Blade angle (pitch position) (°)  \
0 2017-01-01 00:00:00          1                          1.606549
1 2017-01-01 01:00:00          1                          1.386043
```

```
2 2017-01-01 02:00:00          1                          0.488265
3 2017-01-01 03:00:00          1                          4.646393
4 2017-01-01 04:00:00          1                          3.638362

   Energy Export (kWh)  Rotor speed (RPM)  wspeed (m/s)  wdir (deg)  \
0          1507.430254          16.880372     13.368198  231.824738
1          1507.430254          16.880372     12.213772  233.784348
2          1470.421816          16.880372     12.209412  231.856689
3          1476.589889          16.880372     12.360149  228.361801
4          1532.102546          16.880372     12.777927  229.301239

   air pressure (Pa)
0      101490.335938
1      101377.070312
2      101306.453125
3      101272.976562
4      101162.203125

                 Datetime  turbineId  Blade angle (pitch position) (°)  \
17517 2018-12-31 20:00:00          1                         338.978872
17518 2018-12-31 21:00:00          1                         338.903552
17519 2018-12-31 22:00:00          1                         338.705954
17520 2018-12-31 23:00:00          1                         338.852346
17521 2019-01-01 00:00:00          1                         338.793705

       Energy Export (kWh)  Rotor speed (RPM)  wspeed (m/s)  wdir (deg)  \
17517           589.929385          15.919258     10.395692  259.509277
17518           699.412682          16.154580     10.335975  271.005768
17519           943.822578          16.823080     10.146530  264.667816
17520           815.835062          16.695397     11.404872  265.993774
17521          1044.053765          16.890140     11.968995  265.476349

       air pressure (Pa)
17517      102488.523438
17518      102464.070312
17519      102330.890625
17520      102309.804688
17521      102273.820312
```

[12]:
```
print(f"Turbine 1 contains {len(df_tub_1)} rows, Turbine 2 contains␣
↪{len(df_tub_2)} rows, Turbine 3 contains {len(df_tub_3)} rows. This totals␣
↪{len(df_tub_1) + len(df_tub_2) + len(df_tub_3)} rows to use.")
```

```
Turbine 1 contains 17522 rows, Turbine 2 contains 17522 rows, Turbine 3 contains
17522 rows. This totals 52566 rows to use.
```

6 Cut off at 10 m/s

Since wind speeds above 10 m/s give too much noise to really say something about erosion, let's keep only the measurements smaller than or equal to 10 m/s.

[13]:
```
df_tub_1 = df_tub_1[df_tub_1["wspeed (m/s)"] <= 10.0]
df_tub_2 = df_tub_2[df_tub_2["wspeed (m/s)"] <= 10.0]
df_tub_3 = df_tub_3[df_tub_3["wspeed (m/s)"] <= 10.0]
```

[14]:
```
print(f"Turbine Turbine 1 contains {len(df_tub_1)} rows, Turbine 2 contains␣
↪{len(df_tub_2)} rows, Turbine 3 contains {len(df_tub_3)} rows. This totals␣
↪{len(df_tub_1) + len(df_tub_2) + len(df_tub_3)} rows to use.")
```

```
Turbine Turbine 1 contains 10948 rows, Turbine 2 contains 10948 rows, Turbine 3
contains 10948 rows. This totals 32844 rows to use.
```

7 Visualizing the data up to now

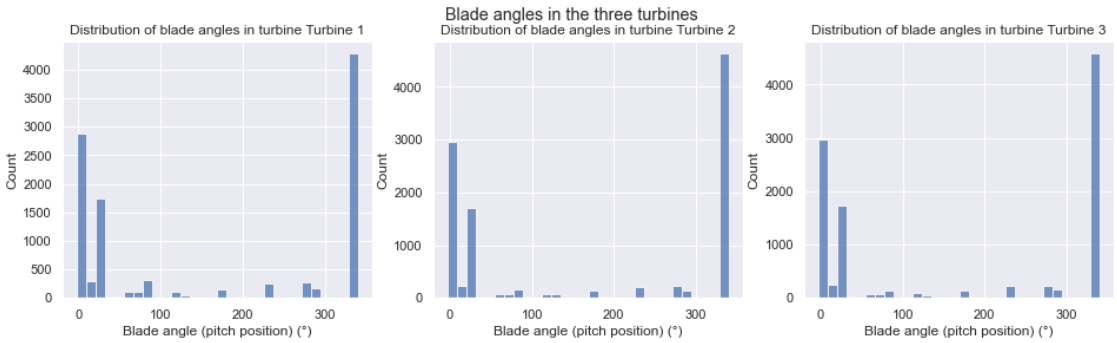
[15]:
```
df_tub_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10948 entries, 17 to 17516
Data columns (total 8 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Datetime                          10948 non-null  datetime64[ns]
 1   turbineId                         10948 non-null  int64
 2   Blade angle (pitch position) (°)  10781 non-null  float64
 3   Energy Export (kWh)               10948 non-null  float64
 4   Rotor speed (RPM)                 10781 non-null  float64
 5   wspeed (m/s)                      10948 non-null  float64
 6   wdir (deg)                        10948 non-null  float64
 7   air pressure (Pa)                 10948 non-null  float64
dtypes: datetime64[ns](1), float64(6), int64(1)
memory usage: 769.8 KB
```

7.1 Blade angles

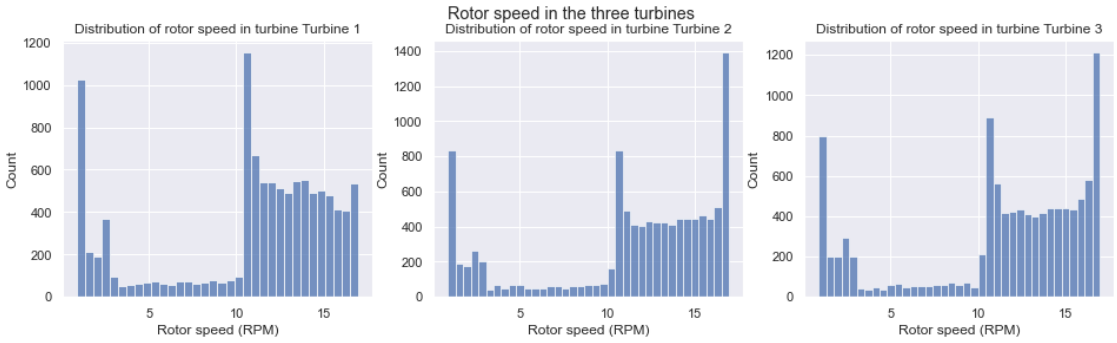
[16]:
```
def blade_angle_histograms():
    figure, axes = plt.subplots(1, 3, figsize=(16, 4))
    figure.suptitle("Blade angles in the three turbines")

    sns.histplot(df_tub_1, x = 'Blade angle (pitch position) (°)', ax =␣
↪axes[0], bins = 30)
    axes[0].set_title("Distribution of blade angles in turbine Turbine 1")

    sns.histplot(df_tub_2, x = 'Blade angle (pitch position) (°)', ax =␣
↪axes[1], bins = 30)
```

```
    axes[1].set_title("Distribution of blade angles in turbine Turbine 2")

    sns.histplot(df_tub_3, x = 'Blade angle (pitch position) (°)', ax =␣
↪axes[2], bins = 30)
    axes[2].set_title("Distribution of blade angles in turbine Turbine 3")

blade_angle_histograms()
```



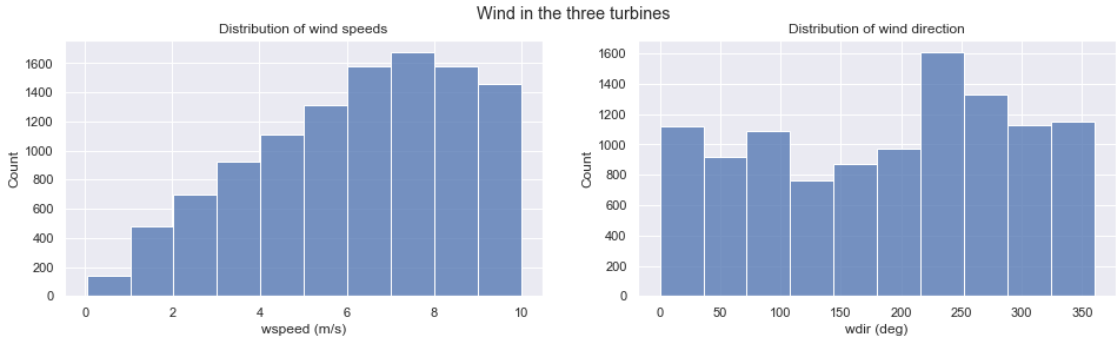
7.2 Rotor speed

[17]:
```
def rotor_speed_histograms():
    figure, axes = plt.subplots(1, 3, figsize=(16, 4))
    figure.suptitle("Rotor speed in the three turbines")

    sns.histplot(df_tub_1, x = 'Rotor speed (RPM)', ax = axes[0])
    axes[0].set_title("Distribution of rotor speed in turbine Turbine 1")

    sns.histplot(df_tub_2, x = 'Rotor speed (RPM)', ax = axes[1])
    axes[1].set_title("Distribution of rotor speed in turbine Turbine 2")

    sns.histplot(df_tub_3, x = 'Rotor speed (RPM)', ax = axes[2])
    axes[2].set_title("Distribution of rotor speed in turbine Turbine 3")

rotor_speed_histograms()
```



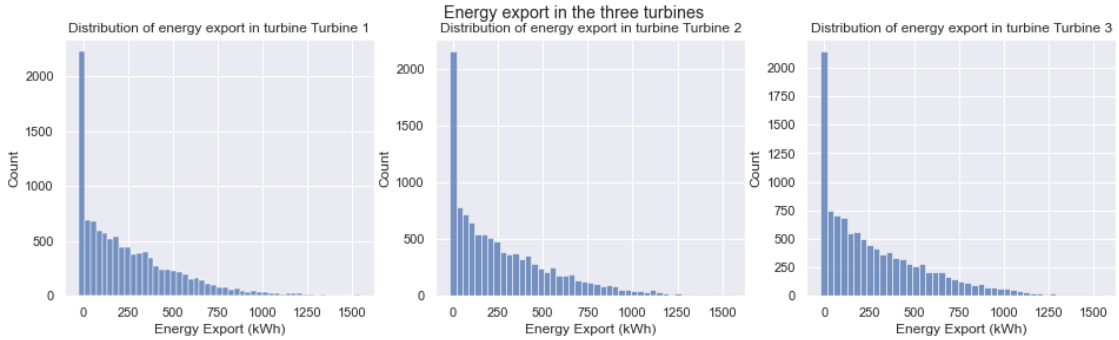
7.3 Wind

[18]:
```
def wind_histograms():
    figure, axes = plt.subplots(1, 2, figsize=(16, 4))
    figure.suptitle("Wind in the three turbines")

    sns.histplot(df_tub_1, x = "wspeed (m/s)", ax = axes[0], bins = 10)
    axes[0].set_title("Distribution of wind speeds")

    sns.histplot(df_tub_1, x = "wdir (deg)", ax = axes[1], bins = 10)
    axes[1].set_title("Distribution of wind direction")

wind_histograms()
```



7.4 Energy export

[19]:
```
def energy_export_histograms():
    figure, axes = plt.subplots(1, 3, figsize=(16, 4))
    figure.suptitle("Energy export in the three turbines")

    sns.histplot(df_tub_1, x = 'Energy Export (kWh)', ax = axes[0])
    axes[0].set_title("Distribution of energy export in turbine Turbine 1")

    sns.histplot(df_tub_2, x = 'Energy Export (kWh)', ax = axes[1])
    axes[1].set_title("Distribution of energy export in turbine Turbine 2")

    sns.histplot(df_tub_3, x = 'Energy Export (kWh)', ax = axes[2])
    axes[2].set_title("Distribution of energy export in turbine Turbine 3")

energy_export_histograms()
```



There are quite some energy exports of 0, let's investigate that a bit more. I expect this to be because of the wind speeds below 4 m/s.

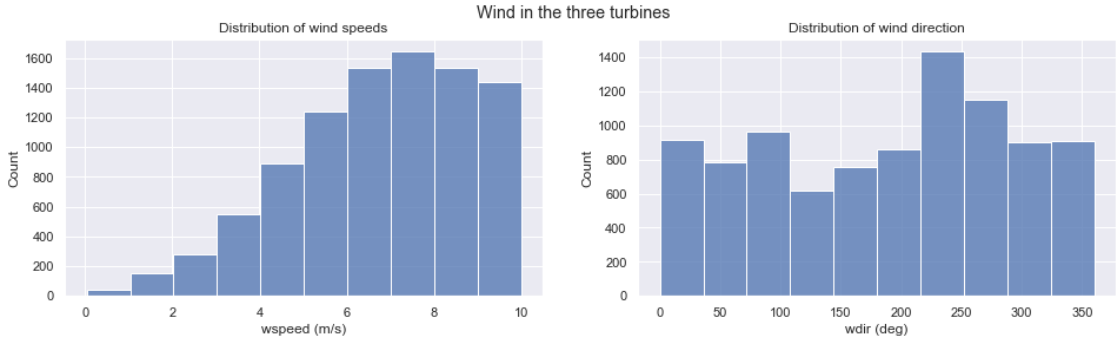
Turns out this removes about 2000 entries. Some 500 of those were of wind speeds below 4 m/s, but also above those speeds. The other energy exports of 0 are probably wrong measurements, repairs or inspections.

[20]:
```
display(df_tub_1[df_tub_1["Energy Export (kWh)"] > 0].info())

figure, axes = plt.subplots(1, 2, figsize=(16, 4))
figure.suptitle("Wind in the three turbines")

sns.histplot(df_tub_1[df_tub_1["Energy Export (kWh)"] > 0], x = "wspeed (m/s)",␣
 ↪ax = axes[0], bins = 10)
axes[0].set_title("Distribution of wind speeds")

sns.histplot(df_tub_1[df_tub_1["Energy Export (kWh)"] > 0], x = "wdir (deg)",␣
 ↪ax = axes[1], bins = 10)
axes[1].set_title("Distribution of wind direction")
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9294 entries, 17 to 17516
Data columns (total 8 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Datetime                          9294 non-null   datetime64[ns]
 1   turbineId                         9294 non-null   int64
 2   Blade angle (pitch position) (°)  9144 non-null   float64
 3   Energy Export (kWh)               9294 non-null   float64
 4   Rotor speed (RPM)                 9144 non-null   float64
 5   wspeed (m/s)                      9294 non-null   float64
 6   wdir (deg)                        9294 non-null   float64
 7   air pressure (Pa)                 9294 non-null   float64
dtypes: datetime64[ns](1), float64(6), int64(1)
memory usage: 653.5 KB
```

```
None
```

[20]:
```
Text(0.5, 1.0, 'Distribution of wind direction')
```



7.5 Blade angle correction

There are some really strange blade angles. For example 180 or 90. Some blade angles are defined as negative numbers, and some are numbers close to 360. Those mean the same thing, but we need to tell our computer that.

Let's define any blade angle more than 30 degrees as *nonsense* and delete it.

[21]:
```
def get_blade_angle_from_zero(angle):
    if (angle > 180):
        return 360 - angle
    if (angle < 0):
        return -angle
    return angle

def remove_nonsense_blade_angles(dataframe):
    dataframe = dataframe[dataframe["Blade angle (pitch position) (°)"] < 30]
    return dataframe

df_tub_1["Blade angle (pitch position) (°)"] = df_tub_1.apply(lambda x:␣
 ↪get_blade_angle_from_zero(x["Blade angle (pitch position) (°)"]), axis=1)
df_tub_2["Blade angle (pitch position) (°)"] = df_tub_2.apply(lambda x:␣
 ↪get_blade_angle_from_zero(x["Blade angle (pitch position) (°)"]), axis=1)
df_tub_3["Blade angle (pitch position) (°)"] = df_tub_3.apply(lambda x:␣
 ↪get_blade_angle_from_zero(x["Blade angle (pitch position) (°)"]), axis=1)

df_tub_1 = remove_nonsense_blade_angles(df_tub_1)
df_tub_2 = remove_nonsense_blade_angles(df_tub_2)
df_tub_3 = remove_nonsense_blade_angles(df_tub_3)
```

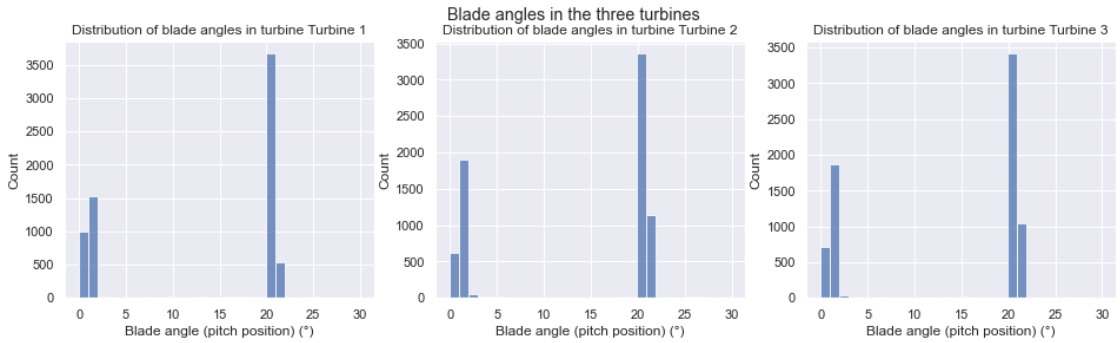
7.6 Low wind speed removal

We should remove all speeds below 4 m/s. The turbines will not work with such low wind speeds, so there is no use keeping them in the dataset.

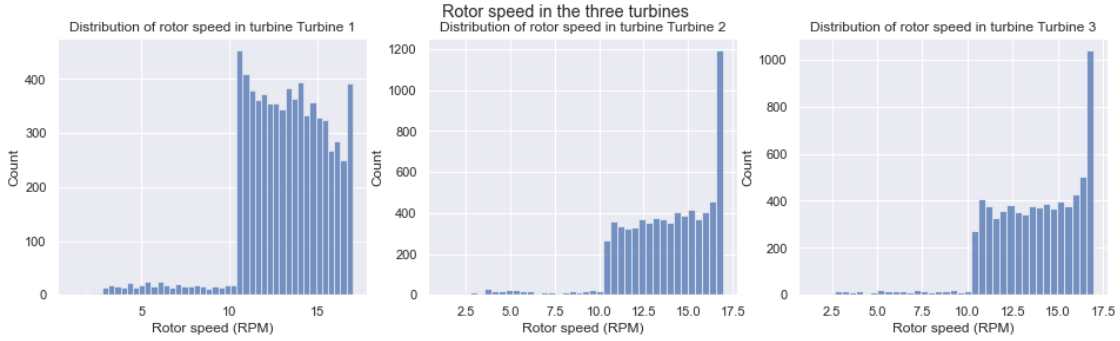
[22]:
```
def remove_low_speeds(dataframe):
    return dataframe[dataframe["wspeed (m/s)"] >= 4]

df_tub_1 = remove_low_speeds(df_tub_1)
df_tub_2 = remove_low_speeds(df_tub_2)
df_tub_3 = remove_low_speeds(df_tub_3)
```

8 Revisualizing the data after preparation

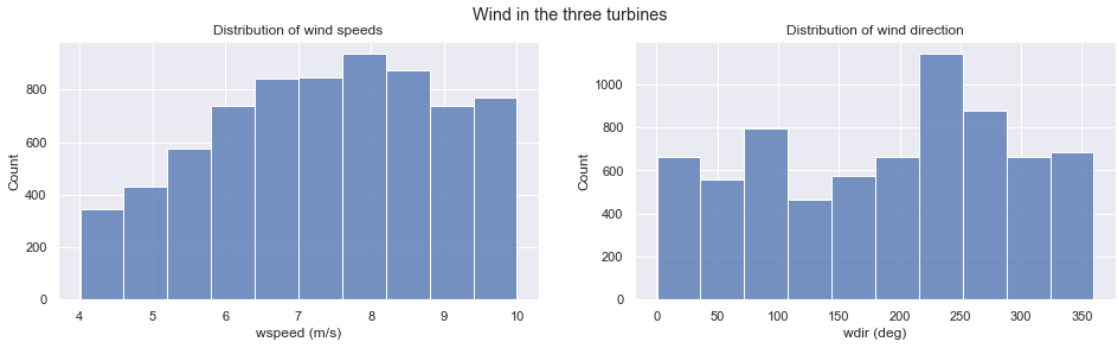
[23]:
```
blade_angle_histograms()
```



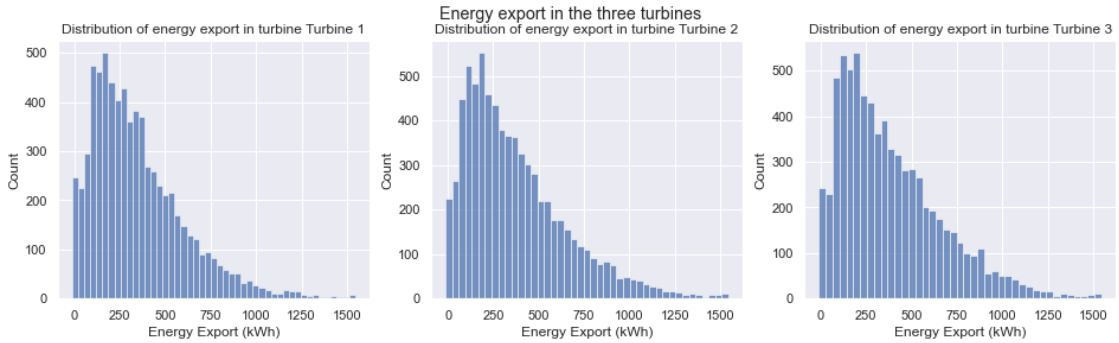
[24]:
```
rotor_speed_histograms()
```



[25]:
```
wind_histograms()
```



[26]:
```
energy_export_histograms()
```



Still some energy exports of 0, but way less than the 2000 we had earlier. Now it's only double digits. They will have to be repairs, inspections or just wrong measurements. Since there are few of them, we'll let them reside in the data.

Note: this cell has no output due to the anonimization process

[27]:
```
display(df_tub_1[df_tub_1["Energy Export (kWh)"] == 0])
display(df_tub_2[df_tub_2["Energy Export (kWh)"] == 0])
display(df_tub_3[df_tub_3["Energy Export (kWh)"] == 0])
```

```
Empty DataFrame
Columns: [Datetime, turbineId, Blade angle (pitch position) (°), Energy Export␣
 ↪(kWh), Rotor speed (RPM), wspeed (m/s), wdir (deg), air pressure (Pa)]
Index: []

Empty DataFrame
Columns: [Datetime, turbineId, Blade angle (pitch position) (°), Energy Export␣
 ↪(kWh), Rotor speed (RPM), wspeed (m/s), wdir (deg), air pressure (Pa)]
Index: []

Empty DataFrame
```

```
Columns: [Datetime, turbineId, Blade angle (pitch position) (°), Energy Export␣
↪(kWh), Rotor speed (RPM), wspeed (m/s), wdir (deg), air pressure (Pa)]
Index: []
```

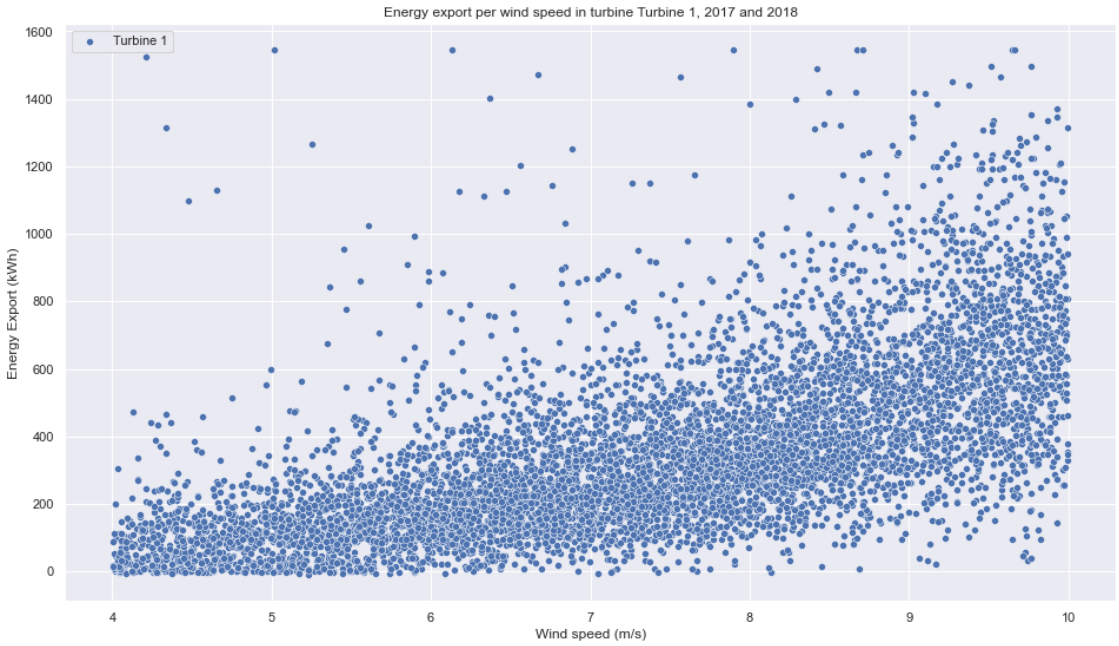
9 Impossible outputs

Let's remove any truly impossible outputs. For example, 1900 kWh energy export at an average windspeed of 4 m/s... The formula for generated power between 4 and 10 m/s for our turbine type is `power = 22.39 * x^2 - 96.50x + 50.1` with an R-squared of 0.995.

So let's remove any values that lie more than 30% above the theoretical max output.

[28]:
```
plt.figure(figsize = (16, 9))
plt.figure(figsize=(16,9))
plt.title('Energy export per wind speed in turbine Turbine 1, 2017 and 2018')
plt.xlabel('Wind speed (m/s)')
plt.ylabel('Energy Export (kWh)')
sns.scatterplot(x = "wspeed (m/s)", y = "Energy Export (kWh)", data = df_tub_1,␣
↪label = "Turbine 1")
```

[28]:
```
<AxesSubplot:title={'center':'Energy export per wind speed in turbine Turbine 1,
2017 and 2018'}, xlabel='Wind speed (m/s)', ylabel='Energy Export (kWh)'>

<Figure size 1152x648 with 0 Axes>
```

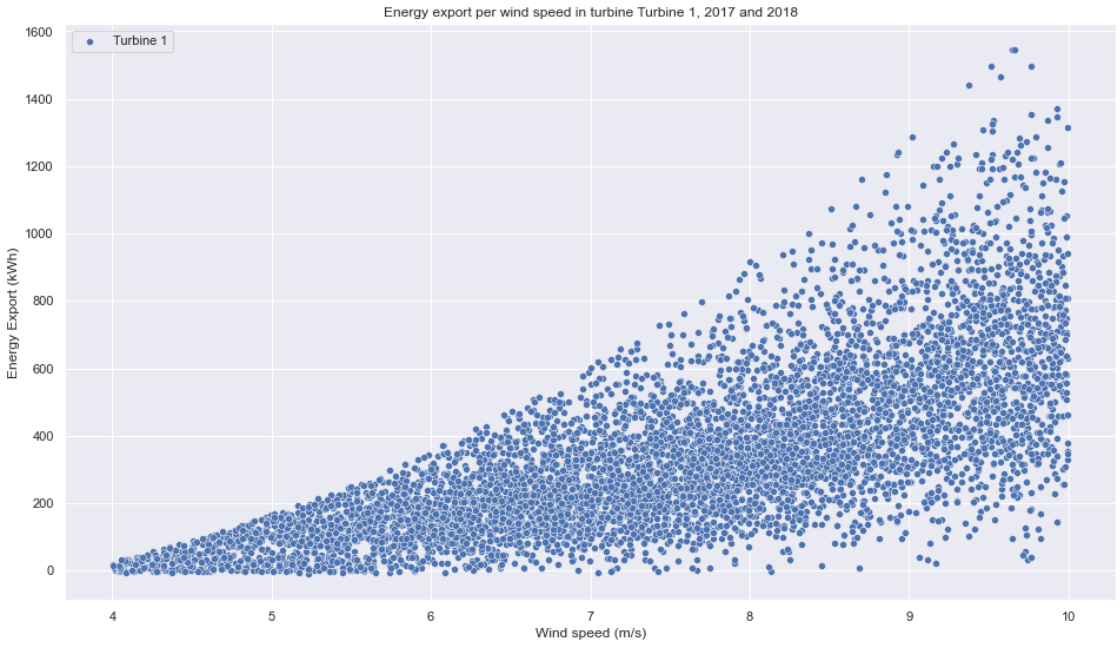


[29]:
```
factor = 1.3
```

```
df_tub_1 = df_tub_1[df_tub_1["Energy Export (kWh)"] < (factor * (22.39 *␣
 ↪df_tub_1["wspeed (m/s)"] * df_tub_1["wspeed (m/s)"] - 96.5 *␣
 ↪df_tub_1["wspeed (m/s)"] + 50.1))]
df_tub_2 = df_tub_2[df_tub_2["Energy Export (kWh)"] < (factor * (22.39 *␣
 ↪df_tub_2["wspeed (m/s)"] * df_tub_2["wspeed (m/s)"] - 96.5 *␣
 ↪df_tub_2["wspeed (m/s)"] + 50.1))]
df_tub_3 = df_tub_3[df_tub_3["Energy Export (kWh)"] < (factor * (22.39 *␣
 ↪df_tub_3["wspeed (m/s)"] * df_tub_3["wspeed (m/s)"] - 96.5 *␣
 ↪df_tub_3["wspeed (m/s)"] + 50.1))]
```

[30]:
```
plt.figure(figsize = (16, 9))
plt.figure(figsize=(16,9))
plt.title('Energy export per wind speed in turbine Turbine 1, 2017 and 2018')
plt.xlabel('Wind speed (m/s)')
plt.ylabel('Energy Export (kWh)')
sns.scatterplot(x = "wspeed (m/s)", y = "Energy Export (kWh)", data = df_tub_1,␣
 ↪label = "Turbine 1")
```

[30]:
```
<AxesSubplot:title={'center':'Energy export per wind speed in turbine Turbine 1,
2017 and 2018'}, xlabel='Wind speed (m/s)', ylabel='Energy Export (kWh)'>

<Figure size 1152x648 with 0 Axes>
```



10 Correlations

Correlation between wind speed and energy export is nice and high. Good, now we can continue.

[31]:
```
display(df_tub_1.corr())
display(df_tub_2.corr())
display(df_tub_3.corr())
```

	turbineId	Blade angle (pitch position) (°)	\
turbineId	NaN	NaN	
Blade angle (pitch position) (°)	NaN	1.000000	
Energy Export (kWh)	NaN	0.016165	
Rotor speed (RPM)	NaN	-0.029018	
wspeed (m/s)	NaN	-0.010758	
wdir (deg)	NaN	-0.091374	
air pressure (Pa)	NaN	-0.030182	

	Energy Export (kWh)	Rotor speed (RPM)	\
turbineId	NaN	NaN	
Blade angle (pitch position) (°)	0.016165	-0.029018	
Energy Export (kWh)	1.000000	0.854039	
Rotor speed (RPM)	0.854039	1.000000	
wspeed (m/s)	0.724208	0.740978	
wdir (deg)	-0.102579	-0.097967	
air pressure (Pa)	-0.096714	-0.099493	

	wspeed (m/s)	wdir (deg)	air pressure (Pa)
turbineId	NaN	NaN	NaN
Blade angle (pitch position) (°)	-0.010758	-0.091374	-0.030182
Energy Export (kWh)	0.724208	-0.102579	-0.096714
Rotor speed (RPM)	0.740978	-0.097967	-0.099493
wspeed (m/s)	1.000000	0.030126	-0.123062
wdir (deg)	0.030126	1.000000	0.003463
air pressure (Pa)	-0.123062	0.003463	1.000000

	turbineId	Blade angle (pitch position) (°)	\
turbineId	NaN	NaN	
Blade angle (pitch position) (°)	NaN	1.000000	
Energy Export (kWh)	NaN	-0.012105	
Rotor speed (RPM)	NaN	-0.052028	
wspeed (m/s)	NaN	-0.027316	
wdir (deg)	NaN	-0.087370	
air pressure (Pa)	NaN	-0.027825	

	Energy Export (kWh)	Rotor speed (RPM)	\
turbineId	NaN	NaN	
Blade angle (pitch position) (°)	-0.012105	-0.052028	
Energy Export (kWh)	1.000000	0.819068	
Rotor speed (RPM)	0.819068	1.000000	
wspeed (m/s)	0.737678	0.743201	
wdir (deg)	-0.008510	-0.025042	
air pressure (Pa)	-0.115844	-0.112907	

```
                                wspeed (m/s)  wdir (deg)  air pressure (Pa)
turbineId                                NaN         NaN                NaN
Blade angle (pitch position) (°)   -0.027316   -0.087370          -0.027825
Energy Export (kWh)                 0.737678   -0.008510          -0.115844
Rotor speed (RPM)                   0.743201   -0.025042          -0.112907
wspeed (m/s)                        1.000000    0.045334          -0.117925
wdir (deg)                          0.045334    1.000000           0.020563
air pressure (Pa)                  -0.117925    0.020563           1.000000

                                turbineId  Blade angle (pitch position) (°)  \
turbineId                             NaN                               NaN
Blade angle (pitch position) (°)      NaN                          1.000000
Energy Export (kWh)                   NaN                         -0.010081
Rotor speed (RPM)                     NaN                         -0.034485
wspeed (m/s)                          NaN                         -0.013819
wdir (deg)                            NaN                         -0.079229
air pressure (Pa)                     NaN                         -0.040939

                                Energy Export (kWh)  Rotor speed (RPM)  \
turbineId                                       NaN                NaN
Blade angle (pitch position) (°)          -0.010081          -0.034485
Energy Export (kWh)                        1.000000           0.844806
Rotor speed (RPM)                          0.844806           1.000000
wspeed (m/s)                               0.750796           0.758217
wdir (deg)                                 0.113075           0.084541
air pressure (Pa)                         -0.113303          -0.107789

                                wspeed (m/s)  wdir (deg)  air pressure (Pa)
turbineId                                NaN         NaN                NaN
Blade angle (pitch position) (°)   -0.013819   -0.079229          -0.040939
Energy Export (kWh)                 0.750796    0.113075          -0.113303
Rotor speed (RPM)                   0.758217    0.084541          -0.107789
wspeed (m/s)                        1.000000    0.044098          -0.125003
wdir (deg)                          0.044098    1.000000           0.010314
air pressure (Pa)                  -0.125003    0.010314           1.000000
```

11 Total rows remaining

We now have three datasets with relatively clean data to use.

[32]:
```
print(f"Turbine 1 contains {len(df_tub_1)} rows, Turbine 2 contains␣
↪{len(df_tub_2)} rows, Turbine 3 contains {len(df_tub_3)} rows. This totals␣
↪{len(df_tub_1) + len(df_tub_2) + len(df_tub_3)} rows to use.")
```

```
Turbine 1 contains 6534 rows, Turbine 2 contains 6710 rows, Turbine 3 contains
6663 rows. This totals 19907 rows to use.
```

12 Diverging paths

There are now two ways (well… a lot more, but let's stay with two) to try and find an answer to our question: *Can we see that the turbines are generating less energy in comparable external conditions if time progresses?* If yes, that's possibly due to erosion and a good reason to send out the drone for inspection.

12.1 The first way

Create *bins* with discrete conditions of wind speed, wind direction and blade angle. Then visualize these bins on the time axis and see if we can spot downward trends. This is hard, manual work. See last cells of this notebook for an indication of what could be possible.

12.2 The second way

Consists of modelling a regression function of (wind speed, air pressure, wind direction) vs energy export. Both linear and polynomial regressions can be used, since the power curve for the turbine is exponential up until appr. 15 m/s. Then we'll take the raw, real data of the turbines and calculate the error for the regression line. When we have calculated all the errors, we'll visualize those errors on the time axis in a moving average. If we see a downward trend as time progresses, we might very well see erosion happening.

We have chosen the second way: using a linear and second degree polynomial regression

13 Turbine 1

[33]:
```
train_x = df_tub_1[["wspeed (m/s)", "air pressure (Pa)", "wdir (deg)"]]
print(train_x.shape)
train_y = df_tub_1["Energy Export (kWh)"]
print(train_y.shape)
```

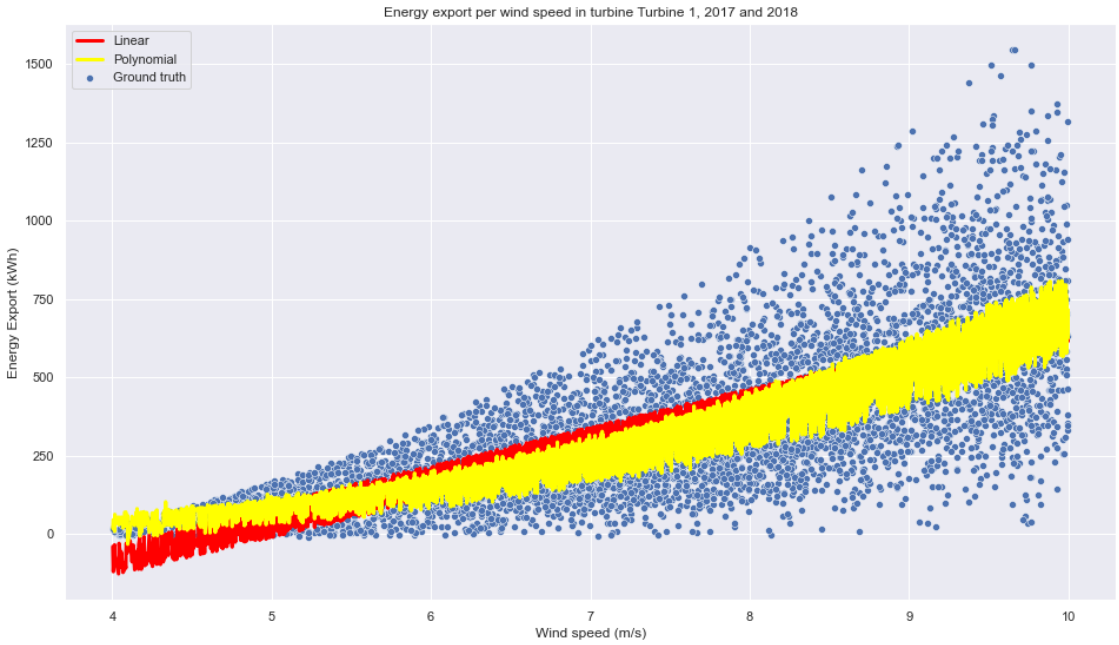
```
(6534, 3)
(6534,)
```

[34]:
```
linear_model = LinearRegression()
linear_model.fit(train_x, train_y)
linear_predictions = linear_model.predict(train_x)

print(mean_absolute_error(train_y, linear_predictions))

poly = PolynomialFeatures(degree=2)
poly_x = poly.fit_transform(train_x)

poly_model = LinearRegression()
poly_model.fit(poly_x, train_y)
poly_predictions = poly_model.predict(poly_x)

print(mean_absolute_error(train_y, poly_predictions))
```

```
124.01574114367796
117.93953579868506
```

[35]:
```
plt.figure(figsize = (16, 9))
plt.figure(figsize=(16,9))
plt.title('Energy export per wind speed in turbine Turbine 1, 2017 and 2018')
plt.xlabel('Wind speed (m/s)')
plt.ylabel('Energy Export (kWh)')
sns.scatterplot(x = "wspeed (m/s)", y = "Energy Export (kWh)", data = df_tub_1,␣
↪label = "Ground truth")
sns.lineplot(x = "wspeed (m/s)", y = linear_predictions, data = df_tub_1,␣
↪linewidth = 3, color = "red", label = "Linear")
sns.lineplot(x = "wspeed (m/s)", y = poly_predictions, data = df_tub_1,␣
↪linewidth = 3, color = "yellow", label = "Polynomial")
```

[35]:
```
<AxesSubplot:title={'center':'Energy export per wind speed in turbine Turbine 1,
2017 and 2018'}, xlabel='Wind speed (m/s)', ylabel='Energy Export (kWh)'>

<Figure size 1152x648 with 0 Axes>
```



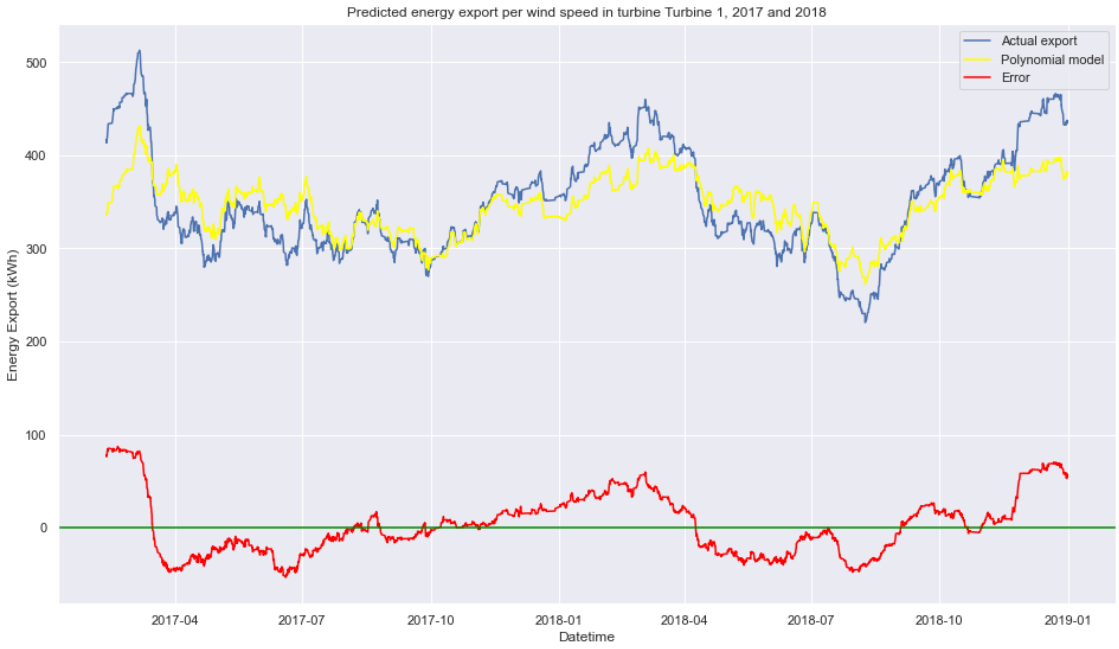
[36]:
```
df_tub_1["energy_prediction"] = poly_predictions
df_tub_1["error"] = df_tub_1["Energy Export (kWh)"] -␣
↪df_tub_1["energy_prediction"]
```

[37]:
```
window_size = 2 * 7 * 24 # 2 weeks of 7 days of 24 hours

plt.figure(figsize = (16, 9))
```

```
plt.figure(figsize=(16,9))
plt.title('Predicted energy export per wind speed in turbine Turbine 1, 2017␣
 ↪and 2018')
plt.xlabel('Datetime')
plt.ylabel('Energy Export (kWh)')
sns.lineplot(x = "Datetime", y = df_tub_1["Energy Export (kWh)"].rolling(window␣
 ↪= window_size).mean(), data = df_tub_1, label = "Actual export")
sns.lineplot(x = "Datetime", y = df_tub_1["energy_prediction"].rolling(window =␣
 ↪window_size).mean(), data = df_tub_1, color = "yellow", label = "Polynomial␣
 ↪model")
sns.lineplot(x = "Datetime", y = df_tub_1["error"].rolling(window =␣
 ↪window_size).mean(), data = df_tub_1, color = "red", label = "Error")
plt.axhline(0, color = "green")
```

[37]: <matplotlib.lines.Line2D at 0x212866d0280>

<Figure size 1152x648 with 0 Axes>



14 Turbine 2

[38]:
```
train_x = df_tub_2[["wspeed (m/s)", "air pressure (Pa)", "wdir (deg)"]]
print(train_x.shape)
train_y = df_tub_2["Energy Export (kWh)"]
print(train_y.shape)
```

(6710, 3)

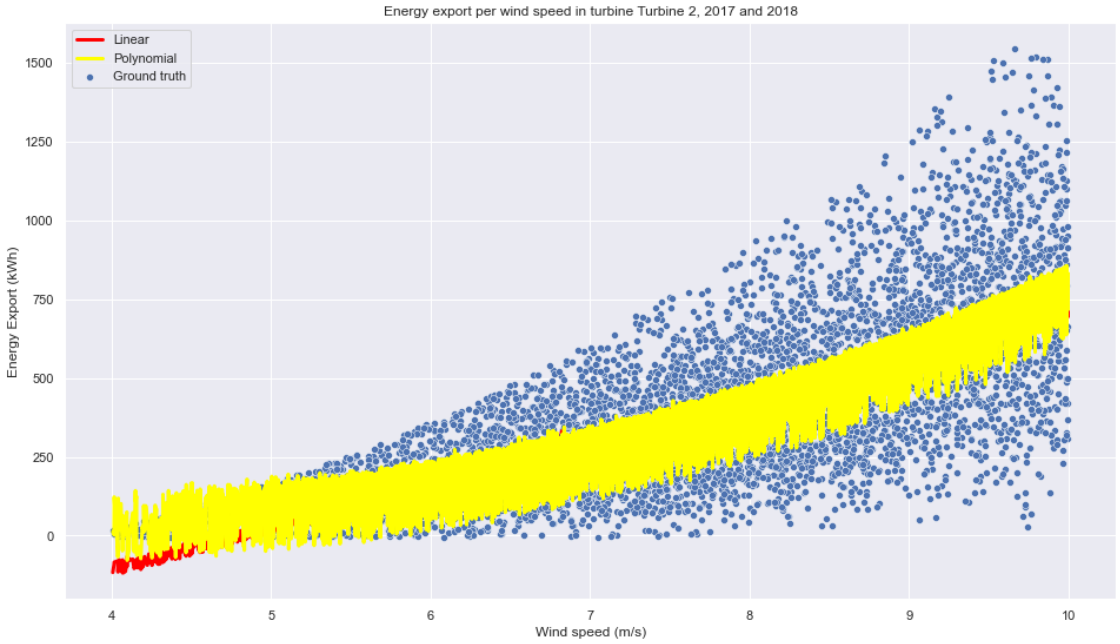
```
(6710,)
```

[39]:
```
linear_model = LinearRegression()
linear_model.fit(train_x, train_y)
linear_predictions = linear_model.predict(train_x)

print(mean_absolute_error(train_y, linear_predictions))

poly = PolynomialFeatures(degree=2)
poly_x = poly.fit_transform(train_x)

poly_model = LinearRegression()
poly_model.fit(poly_x, train_y)
poly_predictions = poly_model.predict(poly_x)

print(mean_absolute_error(train_y, poly_predictions))
```

```
137.51450281643486
123.95899954190739
```

[40]:
```
plt.figure(figsize = (16, 9))
plt.figure(figsize=(16,9))
plt.title('Energy export per wind speed in turbine Turbine 2, 2017 and 2018')
plt.xlabel('Wind speed (m/s)')
plt.ylabel('Energy Export (kWh)')
sns.scatterplot(x = "wspeed (m/s)", y = "Energy Export (kWh)", data = df_tub_2,␣
↪label = "Ground truth")
sns.lineplot(x = "wspeed (m/s)", y = linear_predictions, data = df_tub_2,␣
↪linewidth = 3, color = "red", label = "Linear")
sns.lineplot(x = "wspeed (m/s)", y = poly_predictions, data = df_tub_2,␣
↪linewidth = 3, color = "yellow", label = "Polynomial")
```

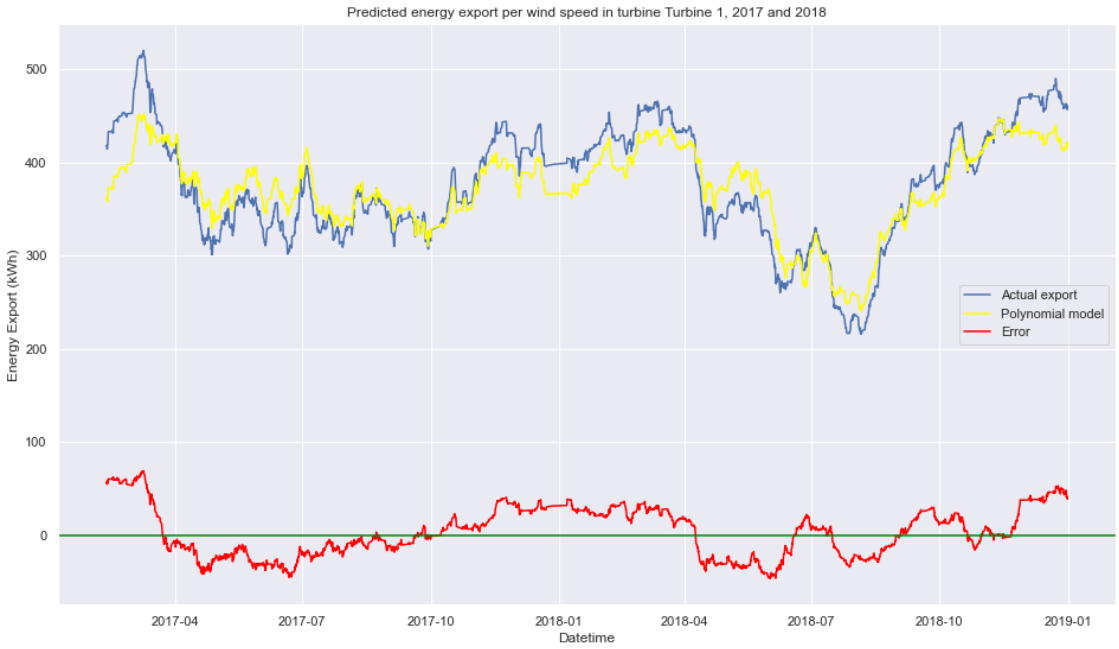
[40]:
```
<AxesSubplot:title={'center':'Energy export per wind speed in turbine Turbine 2,
2017 and 2018'}, xlabel='Wind speed (m/s)', ylabel='Energy Export (kWh)'>

<Figure size 1152x648 with 0 Axes>
```



[41]:
```
df_tub_2["energy_prediction"] = poly_predictions
df_tub_2["error"] = df_tub_2["Energy Export (kWh)"] -␣
↪df_tub_2["energy_prediction"]
```

[42]:
```
window_size = 2 * 7 * 24 # 2 weeks of 7 days of 24 hours

plt.figure(figsize = (16, 9))
plt.figure(figsize=(16,9))
plt.title('Predicted energy export per wind speed in turbine Turbine 1, 2017␣
↪and 2018')
plt.xlabel('Datetime')
plt.ylabel('Energy Export (kWh)')
sns.lineplot(x = "Datetime", y = df_tub_2["Energy Export (kWh)"].rolling(window␣
↪= window_size).mean(), data = df_tub_2, label = "Actual export")
sns.lineplot(x = "Datetime", y = df_tub_2["energy_prediction"].rolling(window =␣
↪window_size).mean(), data = df_tub_2, color = "yellow", label = "Polynomial␣
↪model")
sns.lineplot(x = "Datetime", y = df_tub_2["error"].rolling(window =␣
↪window_size).mean(), data = df_tub_2, color = "red", label = "Error")
plt.axhline(0, color = "green")
```

[42]:
```
<matplotlib.lines.Line2D at 0x212f42a2460>

<Figure size 1152x648 with 0 Axes>
```



15 Turbine 3

[43]:
```
train_x = df_tub_3[["wspeed (m/s)", "air pressure (Pa)", "wdir (deg)"]]
print(train_x.shape)
train_y = df_tub_3["Energy Export (kWh)"]
print(train_y.shape)
```

```
(6663, 3)
(6663,)
```

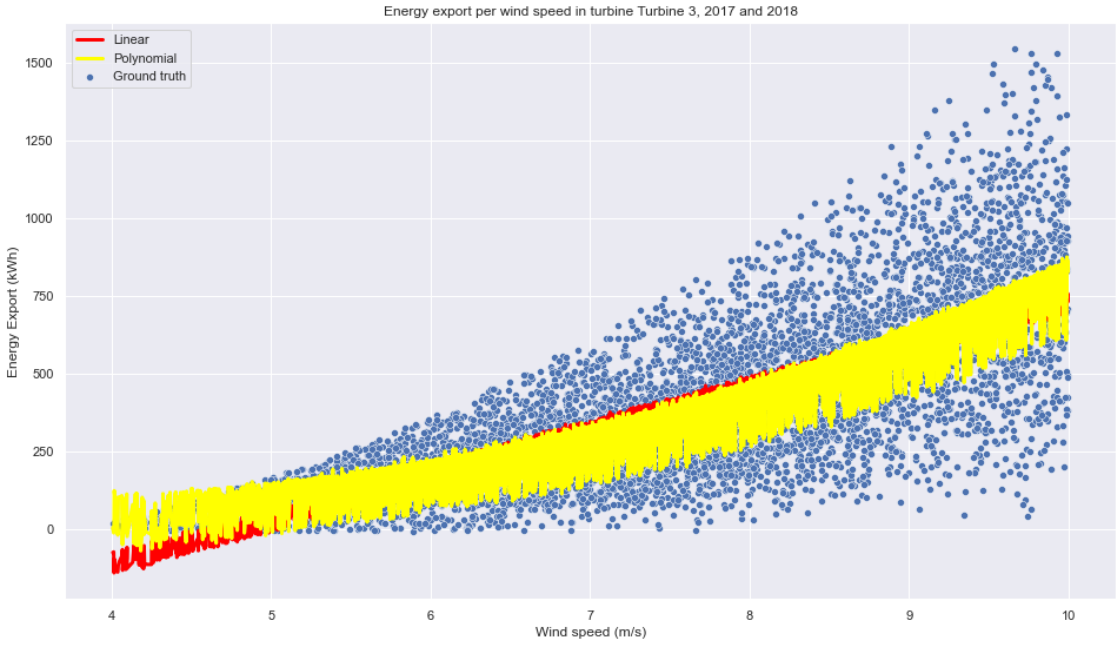
[44]:
```
linear_model = LinearRegression()
linear_model.fit(train_x, train_y)
linear_predictions = linear_model.predict(train_x)

print(mean_absolute_error(train_y, linear_predictions))

poly = PolynomialFeatures(degree=2)
poly_x = poly.fit_transform(train_x)

poly_model = LinearRegression()
poly_model.fit(poly_x, train_y)
poly_predictions = poly_model.predict(poly_x)

print(mean_absolute_error(train_y, poly_predictions))
```

```
135.32997984429196
```

```
124.63008989021846
```

[45]:
```
plt.figure(figsize = (16, 9))
plt.figure(figsize=(16,9))
plt.title('Energy export per wind speed in turbine Turbine 3, 2017 and 2018')
plt.xlabel('Wind speed (m/s)')
plt.ylabel('Energy Export (kWh)')
sns.scatterplot(x = "wspeed (m/s)", y = "Energy Export (kWh)", data = df_tub_3,␣
↪label = "Ground truth")
sns.lineplot(x = "wspeed (m/s)", y = linear_predictions, data = df_tub_3,␣
↪linewidth = 3, color = "red", label = "Linear")
sns.lineplot(x = "wspeed (m/s)", y = poly_predictions, data = df_tub_3,␣
↪linewidth = 3, color = "yellow", label = "Polynomial")
```

[45]:
```
<AxesSubplot:title={'center':'Energy export per wind speed in turbine Turbine 3,
2017 and 2018'}, xlabel='Wind speed (m/s)', ylabel='Energy Export (kWh)'>

<Figure size 1152x648 with 0 Axes>
```



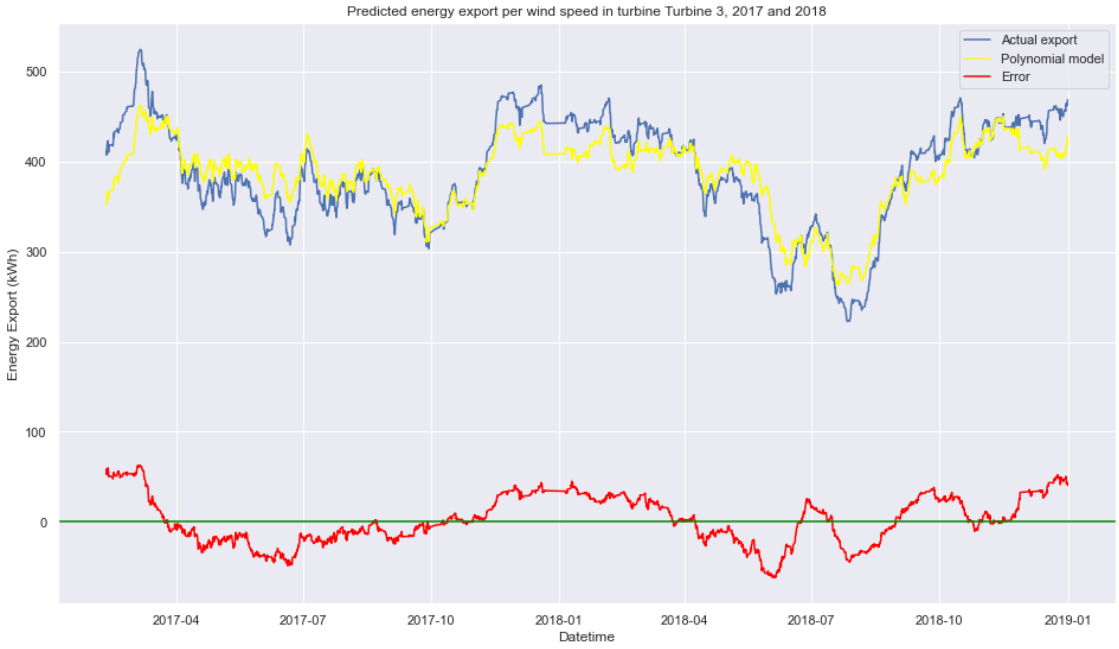
[46]:
```
df_tub_3["energy_prediction"] = poly_predictions
df_tub_3["error"] = df_tub_3["Energy Export (kWh)"] -␣
↪df_tub_3["energy_prediction"]
```

[47]:
```
window_size = 2 * 7 * 24 # 2 weeks of 7 days of 24 hours

plt.figure(figsize = (16, 9))
plt.figure(figsize=(16,9))
```

```
plt.title('Predicted energy export per wind speed in turbine Turbine 3, 2017␣
 ↪and 2018')
plt.xlabel('Datetime')
plt.ylabel('Energy Export (kWh)')
sns.lineplot(x = "Datetime", y = df_tub_3["Energy Export (kWh)"].rolling(window␣
 ↪= window_size).mean(), data = df_tub_3, label = "Actual export")
sns.lineplot(x = "Datetime", y = df_tub_3["energy_prediction"].rolling(window =␣
 ↪window_size).mean(), data = df_tub_3, color = "yellow", label = "Polynomial␣
 ↪model")
sns.lineplot(x = "Datetime", y = df_tub_3["error"].rolling(window =␣
 ↪window_size).mean(), data = df_tub_3, color = "red", label = "Error")
plt.axhline(0, color = "green")
```

[47]: <matplotlib.lines.Line2D at 0x21286aee220>

<Figure size 1152x648 with 0 Axes>



16 Conclusions

16.1 Main question

Can we see that the turbines are generating less energy in comparable external conditions if time progresses?

Based on the models: no. There is not enough data to support such a conclusion. We can see a couple of other things:

16.2 Other conclusions

1. Wind varies throughout the seasons. Autumn and Winter generally show more exported energy than Spring and Summer.
2. The summer of 2018 broke records for low precipitation, low wind speeds and generally extremely hot weather. We can see that summer in all turbines.
3. The SCADA data is generally not of a good enough quality to do this kind of analysis on. We started with some 470,000 rows. Since we needed to resample on the hour level and cut off at 10 m/s wind speeds, that meant some 35,000 rows left. In the end I could only use a little more than 18,000 rows. That means about 50% of all the resampled data was not usable. See the data processing sections to get an indication of the problems. The energy export and blade pitch problems are especially worrisome.

16.3 Further avenues of research

1. Use better models, for example Support Vector Machines with RBF kernels. Though without a better dataset, this will probably not lead to better results.
2. Continue with the manual creation of bins. Below contains some example code of how this can be tackled.

[ ]:

[ ]:

17 Creating bins

17.1 Creating wind rose categories

Koninklijke Nederlandse Windatlas uses a wind rose that resembles N - NNE - ENE - E - ESE - SSE - S - SSW - WSW - W - WNW - NNW: https://www.knmiprojects.nl/projects/knw-atlas/image-library/f060-windrose. The categories are 30°.

N = 345° - 15°, E = 15° - 45°, etc etc

Let's reuse those categories.

[48]:
```
def create_wind_rose_categories(direction):
    if ((direction > 345 and direction <= 360) or (direction > 0 and direction␣
↪<= 15)):
        return "N"
    if (direction > 15 and direction <= 45):
        return "NNE"
    if (direction > 45 and direction <= 75):
        return "ENE"
    if (direction > 75 and direction <= 105):
        return "E"
    if (direction > 105 and direction <= 135):
        return "ESE"
    if (direction > 135 and direction <= 165):
```

```
        return "SSE"
    if (direction > 165 and direction <= 195):
        return "S"
    if (direction > 195 and direction <= 225):
        return "SSW"
    if (direction > 225 and direction <= 255):
        return "WSW"
    if (direction > 255 and direction <= 285):
        return "W"
    if (direction > 285 and direction <= 315):
        return "WNW"
    if (direction > 315 and direction <= 345):
        return "NNW"

# df_tub_1["wind_direction"] = df_tub_1.apply(lambda x:␣
 ↪create_wind_rose_categories(x["wdir (deg)"]), axis = 1)
# df_tub_2["wind_direction"] = df_tub_2.apply(lambda x:␣
 ↪create_wind_rose_categories(x["wdir (deg)"]), axis = 1)
# df_tub_3["wind_direction"] = df_tub_3.apply(lambda x:␣
 ↪create_wind_rose_categories(x["wdir (deg)"]), axis = 1)
```

17.2 Creating wind speed categories

Koninklijke Nederlandse Windatlas uses wind speed categories of 3 m/s, found in the same wind rose images as above. Since the turbine is only supposed to start working at 4 m/s, we can ignore all lower values.

Assumption: categories of 1.5 m/s, starting at 4. So 4-5.5 m/s, 5.5-7 m/s, 7-8.5 and 8.5-10 m/s.

[49]:
```
def create_wind_speed_categories(speed):
    if (speed <= 5.5):
        return "low"
    if (speed <= 7):
        return "medium-low"
    if (speed <= 8.5):
        return "medium-high"
    if (speed <= 10):
        return "high"

# df_tub_1["wind_speed_category"] = df_tub_1.apply(lambda x:␣
 ↪create_wind_speed_categories(x["wspeed (m/s)"]), axis=1)
# df_tub_2["wind_speed_category"] = df_tub_2.apply(lambda x:␣
 ↪create_wind_speed_categories(x["wspeed (m/s)"]), axis=1)
# df_tub_3["wind_speed_category"] = df_tub_3.apply(lambda x:␣
 ↪create_wind_speed_categories(x["wspeed (m/s)"]), axis=1)
```

17.3 Creating rotor speed categories

Assumption: categories of 5 RPM

[50]:
```
def create_wind_speed_categories(speed):
    if (speed <= 5):
        return "low"
    if (speed <= 10):
        return "medium-low"
    if (speed <= 15):
        return "medium-high"
    if (speed <= 20):
        return "high"

# df_tub_1["rotor_speed_category"] = df_tub_1.apply(lambda x:␣
 ↪create_wind_speed_categories(x["Rotor speed (RPM)"]), axis=1)
# df_tub_2["rotor_speed_category"] = df_tub_2.apply(lambda x:␣
 ↪create_wind_speed_categories(x["Rotor speed (RPM)"]), axis=1)
# df_tub_3["rotor_speed_category"] = df_tub_3.apply(lambda x:␣
 ↪create_wind_speed_categories(x["Rotor speed (RPM)"]), axis=1)
```

17.4 Creating blade angle categories

Assumption: categories of 5 degrees

[51]:
```
def create_wind_speed_categories(angle):
    if (angle <= 5):
        return "very low"
    if (angle <= 10):
        return "low"
    if (angle <= 15):
        return "medium-low"
    if (angle <= 20):
        return "medium-high"
    if (angle <= 25):
        return "high"
    if (angle <= 30):
        return "very high"

# df_tub_1["angle_category"] = df_tub_1.apply(lambda x:␣
 ↪create_wind_speed_categories(x["Blade angle (pitch position) (°)"]), axis=1)
# df_tub_2["angle_category"] = df_tub_2.apply(lambda x:␣
 ↪create_wind_speed_categories(x["Blade angle (pitch position) (°)"]), axis=1)
# df_tub_3["angle_category"] = df_tub_3.apply(lambda x:␣
 ↪create_wind_speed_categories(x["Blade angle (pitch position) (°)"]), axis=1)
```

[ ]:
```

```

[52]:
```
# plt.figure(figsize=(16,9))
# plt.title('Energy export per hour, 2017 and 2018')
# plt.xlabel('Date and time')
# plt.ylabel('Energy Export (kWh)')
# sns.scatterplot(x = "Datetime", y = "Energy Export (kWh)", data =␣
 ↪df_tub_2[(df_tub_2["rotor_speed_category"] == "high")], hue =␣
 ↪"wind_speed_category")
```

[53]:
```
# plt.figure(figsize=(16,9))
# graph = sns.FacetGrid(df_tub_1, col = "wind_speed_category")
# graph.map(sns.scatterplot, "Datetime", "Energy Export (kWh)")
```

[ ]:
```
```

[54]:
```
# plt.figure(figsize=(16,9))
# plt.title('Energy export per hour, 2017 and 2018')
# plt.xlabel('Wind speed (m/s)')
# plt.ylabel('Energy Export (kWh)')
# sns.scatterplot(x = "Datetime", y = "Energy Export (kWh)", data = df_tub_1,␣
 ↪label = "Turbine 1")
# sns.scatterplot(x = "wspeed (m/s)", y = "Energy Export (kWh)", data =␣
 ↪df_tub_2, label = "Turbine 2")
# sns.scatterplot(x = "Datetime", y = "Energy Export (kWh)", data = df_058,␣
 ↪label = "Turbine 3")
```