Lab-Assignment

Nigel Sjölin Grech MA661E - VT2021

Wind Power Forecasting

Imports

- os - operating system function, used to make platform independent paths
- Pandas - for data manipulation
- Numpy - for array and mathematical functions
- missingno - specific library for visualizing missing data
- plotly - for generating visualization
- sklearn - for clustering and dimension reduction
- scipy.stats - for statistical testing
- yellowbrick - for model visualization (elbow testing)

In [100…

```
import pandas as pd
import numpy as np
import missingno as msno
import os
from plotly import express as px
from plotly import graph_objects as go
import seaborn as sns
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import matplotlib.pyplot as plt

import scipy.stats as stats

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from yellowbrick.cluster import KElbowVisualizer


pd.set_option('display.max_columns', 50)

import plotly.io as pio
pio.renderers.default="notebook"
```

1. Data Preparation

1.1 Reading data

Here data is read using pandas' read_csv method. One note is that the os.path.join function is used for platform independence. The head and tail of the data is displayed and we can make an observation: that the top of the file is missing data, while the bottom not so much this may indicate inconsistent data collection at the beginning of the process.

In [101…]
```
raw_turbine_path = os.path.join(os.pardir, 'data', 'Turbine_Data.csv')
raw_turbine_data = pd.read_csv(raw_turbine_path)
```

In [102…]
```
raw_turbine_data.head()
```

Out[102]:

	Unnamed: 0	ActivePower	AmbientTemperatue	BearingShaftTemperature	Blade1PitchAngle
0	2017-12-31 00:00:00+00:00	NaN	NaN	NaN	NaN
1	2017-12-31 00:10:00+00:00	NaN	NaN	NaN	NaN
2	2017-12-31 00:20:00+00:00	NaN	NaN	NaN	NaN
3	2017-12-31 00:30:00+00:00	NaN	NaN	NaN	NaN
4	2017-12-31 00:40:00+00:00	NaN	NaN	NaN	NaN

In [103…]
```
raw_turbine_data.tail()
```

Out[103]:

	Unnamed: 0	ActivePower	AmbientTemperatue	BearingShaftTemperature	Blade1PitchA
118219	2020-03-30 23:10:00+00:00	70.044465	27.523741	45.711129	1.51
118220	2020-03-30 23:20:00+00:00	40.833474	27.602882	45.598573	1.70
118221	2020-03-30 23:30:00+00:00	20.777790	27.560925	45.462045	1.70
118222	2020-03-30 23:40:00+00:00	62.091039	27.810472	45.343827	1.57
118223	2020-03-30 23:50:00+00:00	68.664425	27.915828	45.231610	1.49

Explaining the column names

Column Name	Description
Time Stamp (Unnamed 0)	Time stamp of the data recording, from Jan 2018 - March 2020
ActivePower	The power generated by the turbine (KiloWats)
Ambient temperature	The ambient temperature around the turbine
BearingShaftTemperature	The temperature of the turbine's bearing shaft
Blade1PitchAngle	The pitch angle for the turbine's blade 1
Blade2PitchAngle	The pitch angle for the turbine's blade 2
Blade3PitchAngle	The pitch angle for the turbine's blade 3
ControlBoxTemperature	The temperature of the turbine's control box
GearboxBearingTemperature	The temperature of the turbine's gearbox bearing

Column Name	Description
GearboxOilTemperature	The temperature of the turbine's gearbox oil
GeneratorRPM	Rotations per min for the generator
GeneratorWinding1Temperature	Generator Winding 1 Temperature sensor
GeneratorWinding2Temperature	Generator Winding 2 Temperature sensor
HubTemperature	Hub Temperature
MainBoxTemperature	Main Box Temperature
NacellePosition	The nacelle is the casing on top of the tower that contains components necessary to move the turbine into the wind.
ReactivePower	Power that flows back from a destination toward the grid in an alternating current scenario.
RotorRPM	Rotations per min for the Rotator
TurbineStatus	Turbine statuses, no description of their meaning
WTG	Turbine Name
WindDirection	Wind direction in degrees from north
WindSpeed	Wind Speed

1.2. Manipulating Data

1.2.1 Finding and handling Missing Values

In [104…

```
# defining a na info display function
# we can alternatively use the df.info here but this has a prettier output

def get_info_pdf(df):
    info_df = pd.concat([df.dtypes, df.count(), df.isna().sum()], axis=1).reset_in
            .rename(columns={'index':'feature', 0:'dtype', 1:'# values', 2:'# na'}
    info_df['% missing'] = np.ceil((info_df['# na']*100)/len(df))
    return info_df

get_info_pdf(raw_turbine_data)
```

Out[104]:

	feature	dtype	# values	# na	% missing
0	Unnamed: 0	object	118224	0	0.0
1	ActivePower	float64	94750	23474	20.0
2	AmbientTemperatue	float64	93817	24407	21.0
3	BearingShaftTemperature	float64	62518	55706	48.0
4	Blade1PitchAngle	float64	41996	76228	65.0
5	Blade2PitchAngle	float64	41891	76333	65.0
6	Blade3PitchAngle	float64	41891	76333	65.0
7	ControlBoxTemperature	float64	62160	56064	48.0
8	GearboxBearingTemperature	float64	62540	55684	48.0
9	GearboxOilTemperature	float64	62438	55786	48.0
10	GeneratorRPM	float64	62295	55929	48.0
11	GeneratorWinding1Temperature	float64	62427	55797	48.0
12	GeneratorWinding2Temperature	float64	62449	55775	48.0
13	HubTemperature	float64	62406	55818	48.0
14	MainBoxTemperature	float64	62507	55717	48.0
15	NacellePosition	float64	72278	45946	39.0
16	ReactivePower	float64	94748	23476	20.0
17	RotorRPM	float64	62127	56097	48.0
18	TurbineStatus	float64	62908	55316	47.0
19	WTG	object	118224	0	0.0
20	WindDirection	float64	72278	45946	39.0
21	WindSpeed	float64	94595	23629	20.0

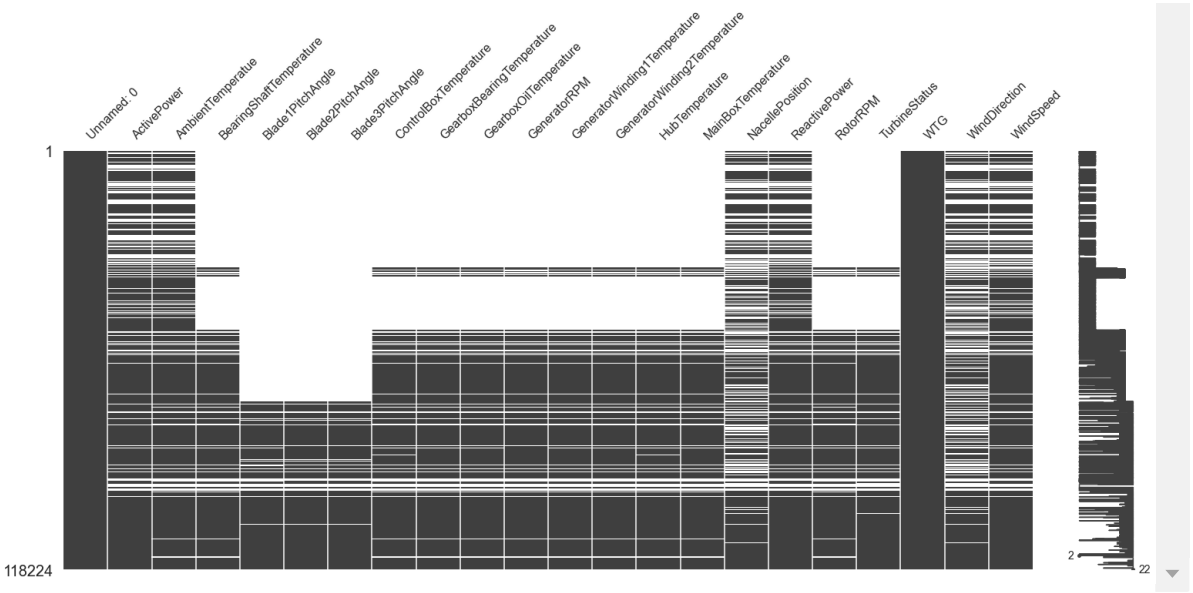
1.2.1.1 Visualizing Missing Data

In [105…

```
msno.matrix(raw_turbine_data)
```

Out[105]:

```
<AxesSubplot:>
```



1.2.1.2 Fixing time series index and value range

We see there is a lot of missing data at the top of the set so we will also look at the missing data by year.

To do this we:

- rename the time stamp col
- cast it to pandas datetime format
- set it as the index

In [106…
```
tmp_1 = raw_turbine_data.rename(columns={'Unnamed: 0': 'TimeStamp'})
tmp_1['TimeStamp'] = pd.to_datetime(tmp_1['TimeStamp'])
missing_turbine_data = tmp_1.set_index('TimeStamp')
missing_turbine_data = missing_turbine_data.tz_localize(None)
```

We can also drop the 'WTG' column since it is just the wind turbine name and there is only one turbine.

In [107…
```
missing_turbine_data.WTG.value_counts()
```

Out[107]:
```
G01    118224
Name: WTG, dtype: int64
```

In [108…
```
missing_turbine_data = missing_turbine_data.drop('WTG', axis=1)
```

We can now use loc to slice the data by year

In [109…
```
msno.matrix(missing_turbine_data.loc['2017'], figsize=(20, 5))
```

Out[109]:
```
<AxesSubplot:>
```

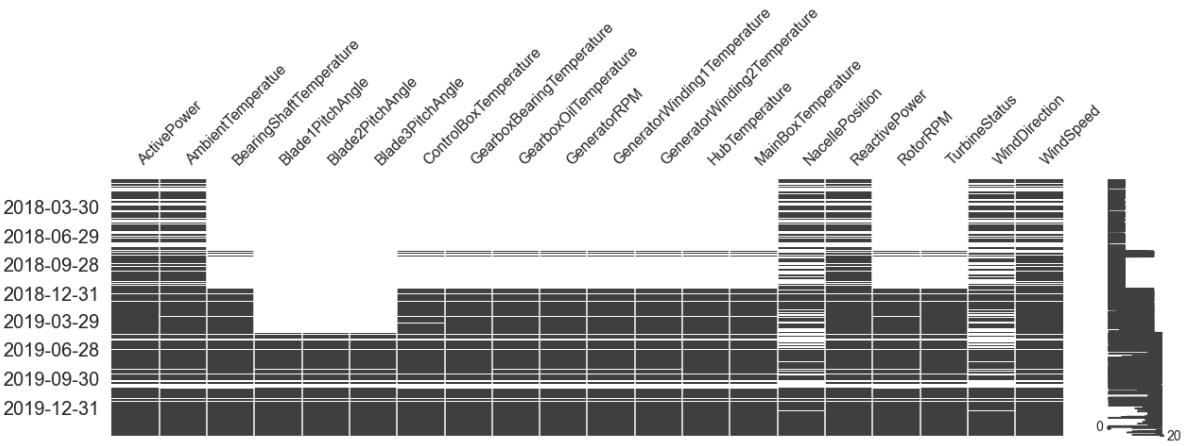
When we look at the 2017 data we see none of the columns have any values so we can safely drop all these rows.

In [110…
```
before_shape = missing_turbine_data.shape
print(f'Shape before dropping 2017 rows: {before_shape}')

missing_turbine_data_v2 = missing_turbine_data.loc['2018':]

print(f'Shape after dropping 2017 rows: {missing_turbine_data_v2.shape}, we loose
```

```
Shape before dropping 2017 rows: (118224, 20)
Shape after dropping 2017 rows: (118080, 20), we loose 144 rows
```

In [111…
```
msno.matrix(missing_turbine_data_v2, figsize=(20, 5), freq='BQ')
```

Out[111]:
```
<AxesSubplot:>
```



In [112…
```
# looking at the % of missing data again

get_info_pdf(missing_turbine_data_v2)
```

Out[112]:

	feature	dtype	# values	# na	% missing
0	ActivePower	float64	94750	23330	20.0
1	AmbientTemperatue	float64	93817	24263	21.0
2	BearingShaftTemperature	float64	62518	55562	48.0
3	Blade1PitchAngle	float64	41996	76084	65.0
4	Blade2PitchAngle	float64	41891	76189	65.0
5	Blade3PitchAngle	float64	41891	76189	65.0
6	ControlBoxTemperature	float64	62160	55920	48.0
7	GearboxBearingTemperature	float64	62540	55540	48.0
8	GearboxOilTemperature	float64	62438	55642	48.0
9	GeneratorRPM	float64	62295	55785	48.0
10	GeneratorWinding1Temperature	float64	62427	55653	48.0
11	GeneratorWinding2Temperature	float64	62449	55631	48.0
12	HubTemperature	float64	62406	55674	48.0
13	MainBoxTemperature	float64	62507	55573	48.0
14	NacellePosition	float64	72278	45802	39.0
15	ReactivePower	float64	94748	23332	20.0
16	RotorRPM	float64	62127	55953	48.0
17	TurbineStatus	float64	62908	55172	47.0
18	WindDirection	float64	72278	45802	39.0
19	WindSpeed	float64	94595	23485	20.0

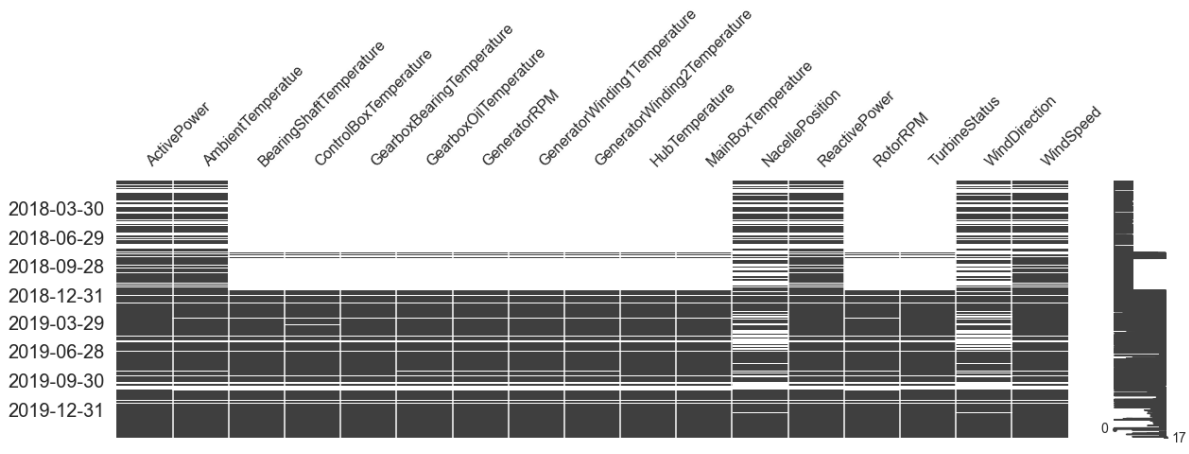
We se the blade pitch columns are still missing in 65% of the data so we will drop these columns.

In [113…

```
missing_turbine_data_v3 = missing_turbine_data_v2.drop(['Blade1PitchAngle', 'Blade2
msno.matrix(missing_turbine_data_v3, figsize=(20, 5), freq='BQ')
```

Out[113]:

```
<AxesSubplot:>
```

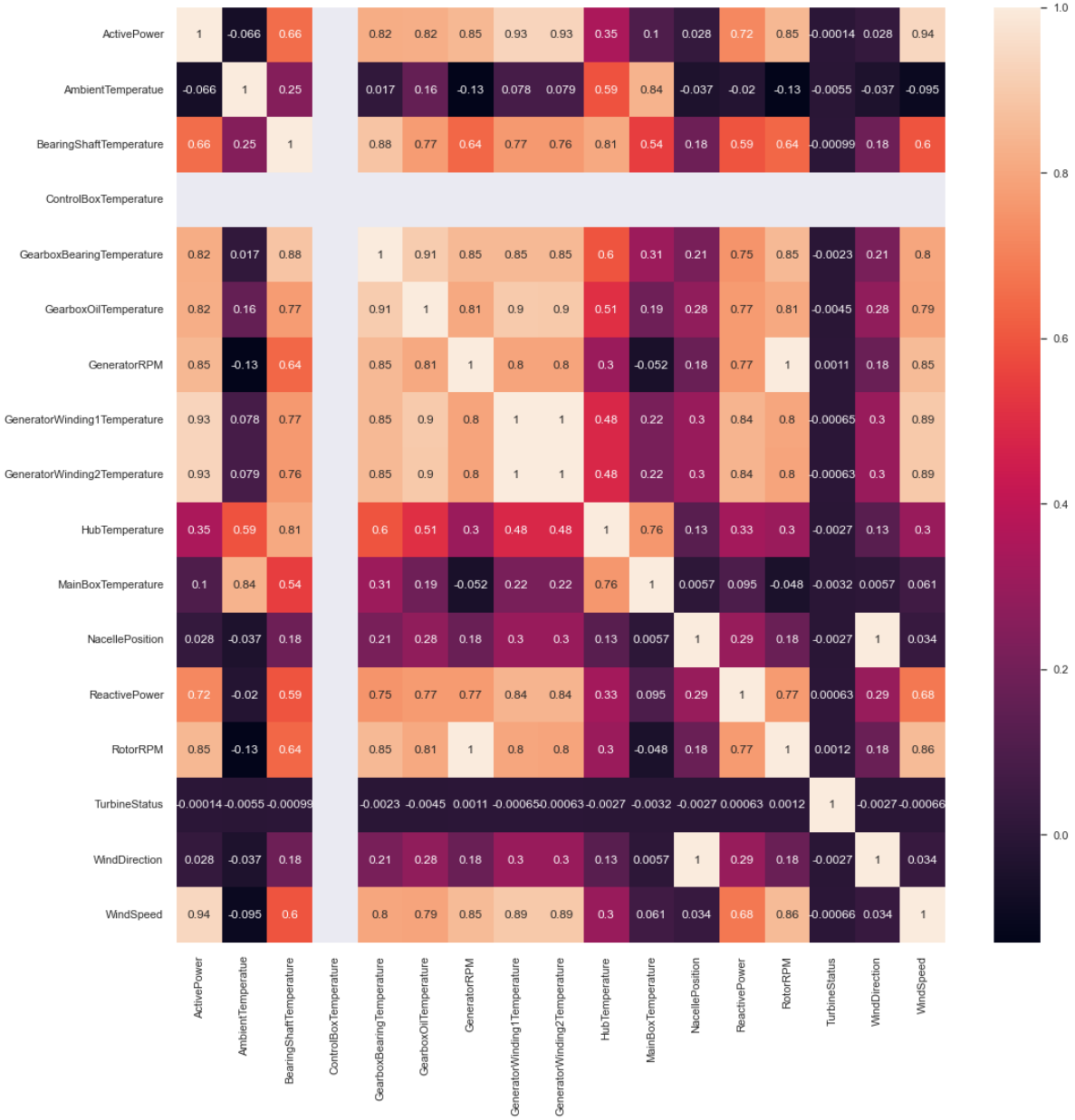


This now leaves us with 11 columns missing ~48% of their data and 6 columns missing ~20%.

1.2.1.3 Dropping uncorrelated columns

We now look at the correlations in the data to remove some of the columns and reduce the amount of missing values we need to fill.

In [114…]
```
sns.set(rc={'figure.figsize':(17,17)})
_ = sns.heatmap(missing_turbine_data_v3.corr(), annot=True)
```



If we look at Active power being the target values for this data set we can start by removing those columns in the range -0.5 to 0.5:

In [115…]
```
cols_to_drop = [missing_turbine_data_v3.columns[i] for i, corr in enumerate(missing
cols_to_drop
```

Out[115]:
```
['AmbientTemperatue',
 'HubTemperature',
 'MainBoxTemperature',
 'NacellePosition',
 'TurbineStatus',
 'WindDirection']
```

We also look into the control box temperature since is has a NA correlation

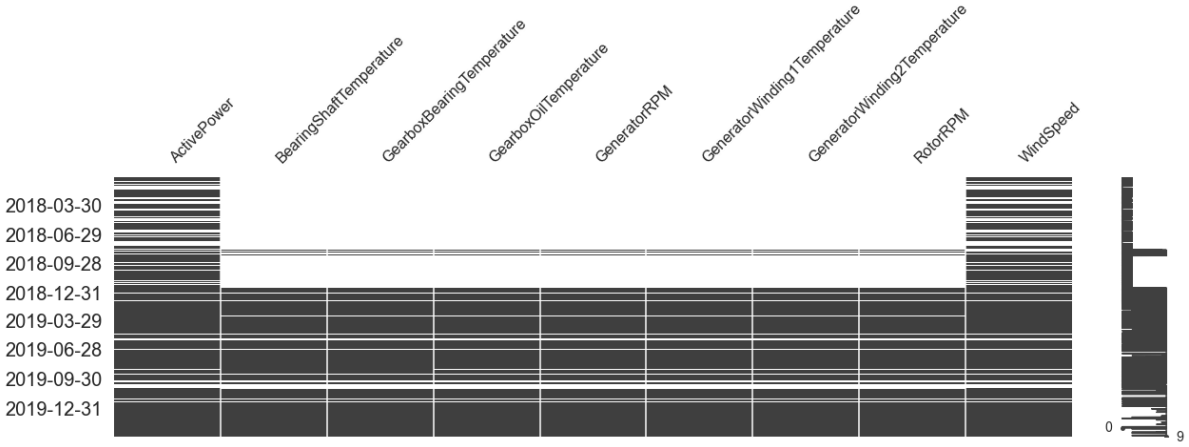
In [116…
```
# Checking control box temperature values
missing_turbine_data_v3[['ControlBoxTemperature']].value_counts()
```

Out[116]:
```
ControlBoxTemperature
0.0                     62160
dtype: int64
```

We see that it has a only one value if 0.0 so we can drop the column, We will also drop Reactive Power since it is feedback from the power grid and (as far as I understand) wont affect power production.

In [117…
```
cols_to_drop += ['ControlBoxTemperature', 'ReactivePower']
```

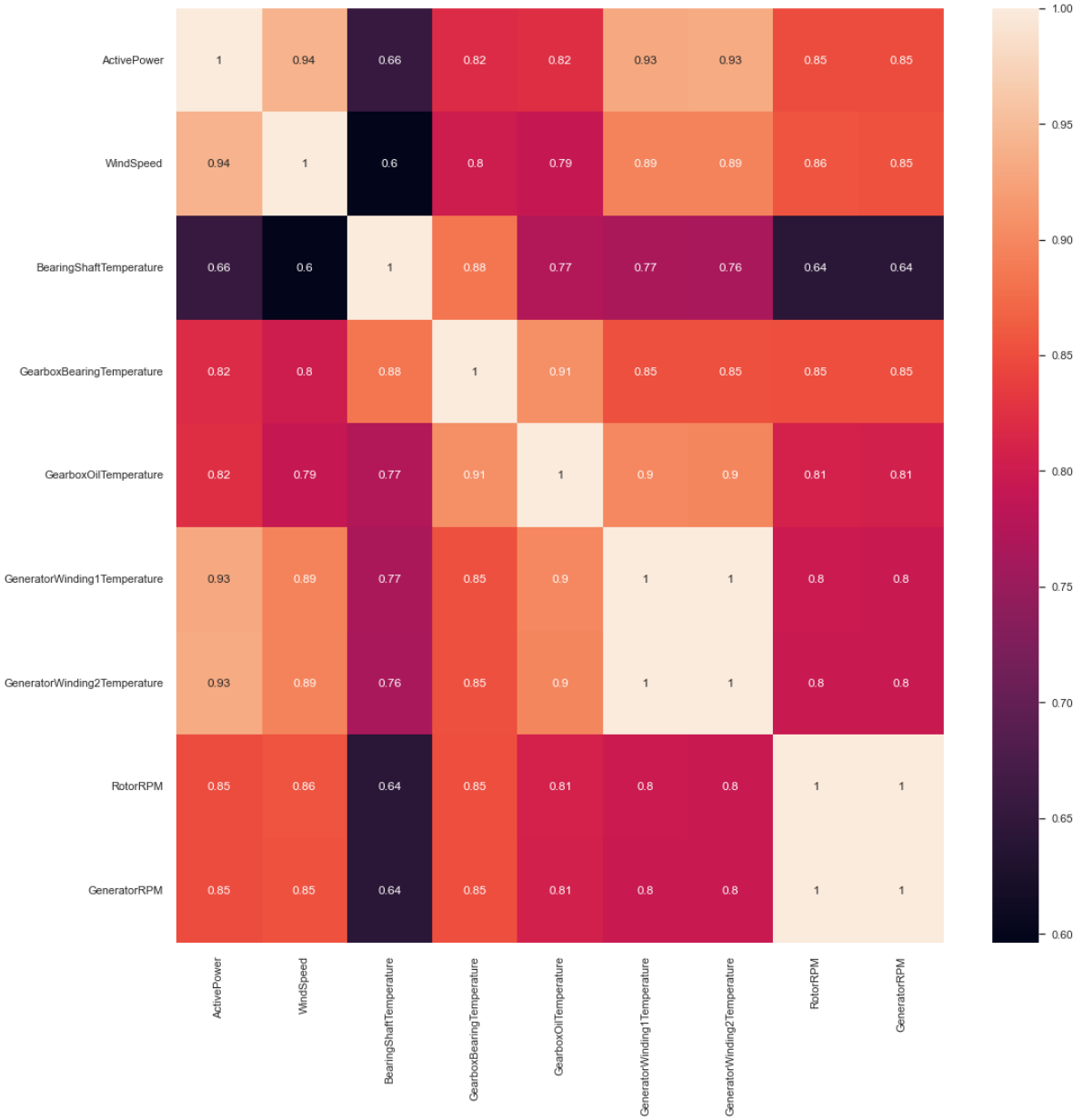
In [118…
```
missing_turbine_data_v4 = missing_turbine_data_v3.drop(cols_to_drop, axis=1)
_ = msno.matrix(missing_turbine_data_v4, figsize=(20, 5), freq='BQ')
```



Reorder cols to keep RPM cols near each other (this will make the corr matrix easier to read)

In [119…
```
cols = ['ActivePower', 'WindSpeed', 'BearingShaftTemperature', 'GearboxBearingTempe
        'GeneratorWinding1Temperature', 'GeneratorWinding2Temperature', 'RotorRPM'
missing_turbine_data_v4 = missing_turbine_data_v4[cols]
```

In [120…
```
_ = sns.heatmap(missing_turbine_data_v4.corr(), annot=True)
```



Of the remaining columns we also see that some are highly correlate with each other. We will keep these columns for now.

1.2.1.4 Filling Missing Values

To fill missing values we will use the mean of that value foe the day of the year. This may not fill all of the missing values since there are some days of the year with no values in 2018-2020, in those cases we use the mean of the week.

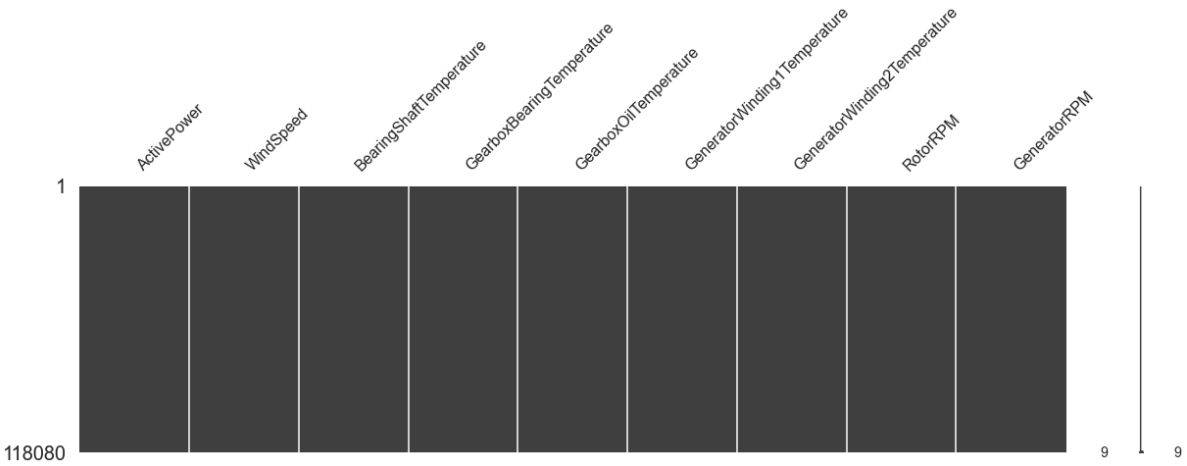
In [121…

```
# Filling missing values using daily mean
missing_turbine_data_v5 = missing_turbine_data_v4.fillna(missing_turbine_data_v4.gr
# there is still some missing day worth of data we can fill them using a weekly avg
turbine_data = missing_turbine_data_v5.fillna(missing_turbine_data_v4.groupby(miss:

get_info_pdf(turbine_data)
```

Out[121]:

	feature	dtype	# values	# na	% missing
0	ActivePower	float64	118080	0	0.0
1	WindSpeed	float64	118080	0	0.0
2	BearingShaftTemperature	float64	118080	0	0.0
3	GearboxBearingTemperature	float64	118080	0	0.0
4	GearboxOilTemperature	float64	118080	0	0.0
5	GeneratorWinding1Temperature	float64	118080	0	0.0
6	GeneratorWinding2Temperature	float64	118080	0	0.0
7	RotorRPM	float64	118080	0	0.0
8	GeneratorRPM	float64	118080	0	0.0

In [122…
```
_ = msno.matrix(turbine_data, figsize=(20, 5))
```



In [123…
```
n_raw_rows = raw_turbine_data.shape[0]
n_rows_with_missing_data = n_raw_rows - raw_turbine_data.dropna(how='any').shape[0

print(f'The data originally had {n_raw_rows} rows, of which {n_rows_with_missing_da
print(f'In handle the missing data a number of records were removed resulting in {
```

```
The data originally had 118224 rows, of which 85496 had missing values
In handle the missing data a number of records were removed resulting in 118080 ro
ws
```

In addition the column were reduced from 21 dimensions to 9, by discarding columns with no meaning or no correlation to the target.

1.2.2 Aggregating Data

Power production per 10-min is not an easy unit it work with so we will resample the data per hour.

For Active Power we will resample by taking the sum of the 10 min power production where as for the remainder fo the fields we can take the mean.

In [124…
```
pdf1 = turbine_data[['ActivePower']].resample('H').sum()
pdf2 = turbine_data.loc[:, 'WindSpeed':].resample('H').mean()
hourly_turbine_data = pd.concat([pdf1, pdf2], axis=1)
```

In [125…
```
print(f'After this resampling we went from {turbine_data.shape[0]} rows to {hourly_
```

```
After this resampling we went from 118080 rows to 19680 rows, 144 samples per day
to 24 samples per day.
```

2. Exploration of Data

2.1 Analyzing feasibility of values

We see that most of the values are within the expected ranges, however we see that the min value for Active Power is -ve. for the purpose of this analysis we will assume that that is a mistake since and correct it to 0.

In [126…
```
hourly_turbine_data.describe()
```

Out[126]:

	ActivePower	WindSpeed	BearingShaftTemperature	GearboxBearingTemperature	Gearbo
count	19680.000000	19680.000000	19680.000000	19680.000000	
mean	3706.709783	5.869944	42.805310	63.738307	
std	3423.663228	2.434869	4.622542	8.997237	
min	-63.166574	1.051559	18.648041	26.947778	
25%	877.290390	4.104073	39.951595	59.142322	
50%	2512.626093	5.428579	42.776467	63.481402	
75%	6139.751181	7.345256	46.124657	69.250039	
max	10369.139318	19.054201	54.844991	82.010276	

In [127…
```
hourly_turbine_data.loc[hourly_turbine_data['ActivePower']<=0, 'ActivePower'].count
```

Out[127]:
```
1717
```

In [128…
```
hourly_turbine_data.loc[hourly_turbine_data['ActivePower']<0, 'ActivePower'] = 0
hourly_turbine_data.describe()
```

Out[128]:

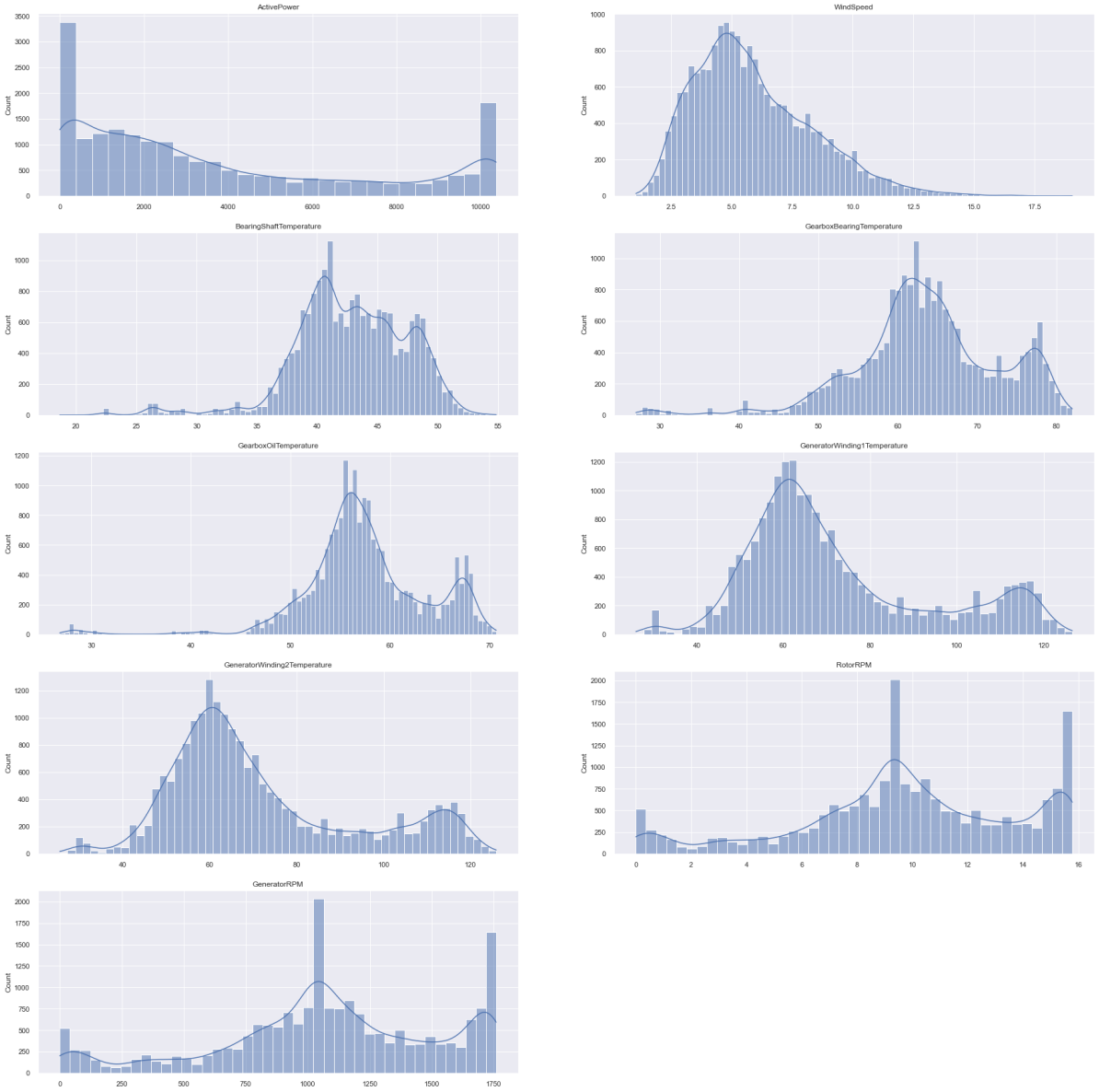
	ActivePower	WindSpeed	BearingShaftTemperature	GearboxBearingTemperature	Gearbo
count	19680.000000	19680.000000	19680.000000	19680.000000	
mean	3709.464346	5.869944	42.805310	63.738307	
std	3420.664468	2.434869	4.622542	8.997237	
min	0.000000	1.051559	18.648041	26.947778	
25%	877.290390	4.104073	39.951595	59.142322	
50%	2512.626093	5.428579	42.776467	63.481402	
75%	6139.751181	7.345256	46.124657	69.250039	
max	10369.139318	19.054201	54.844991	82.010276	

2.2 Univariate analysis

2.2.1 Dist plots

In [129…

```
fig = plt.figure(figsize=(30,50))
for i, col in enumerate(hourly_turbine_data.columns):
    plt.subplot(8,2,i+1)
    sns.histplot(x=hourly_turbine_data[col], kde=True).set(xlabel=None)
    plt.title(col)
```



What is very striking about this data is that we see the similarity in the histograms for example:

- GearBox oil and bearing temperatures
- Generator winding temperatures
- The rotor and generator RPM

Not just in the shape of the respective curves but also in the modality of the data.

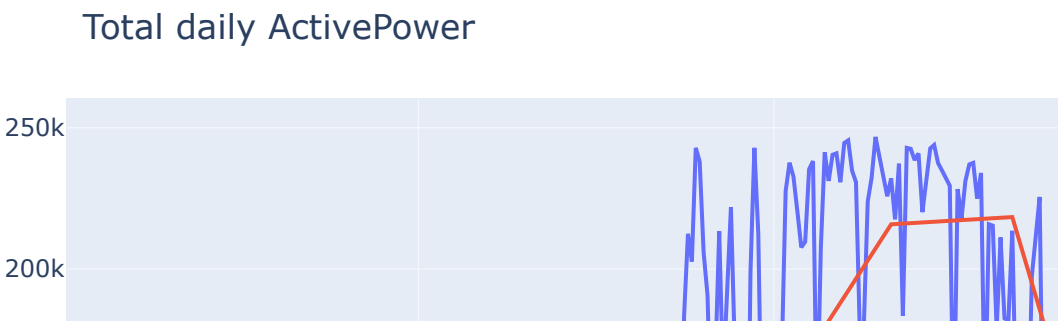
Time series plots

We will also make some time series plots to see how the data shifts according to the season.

Hourly data is too much for a time series so we will resample daily and monthly

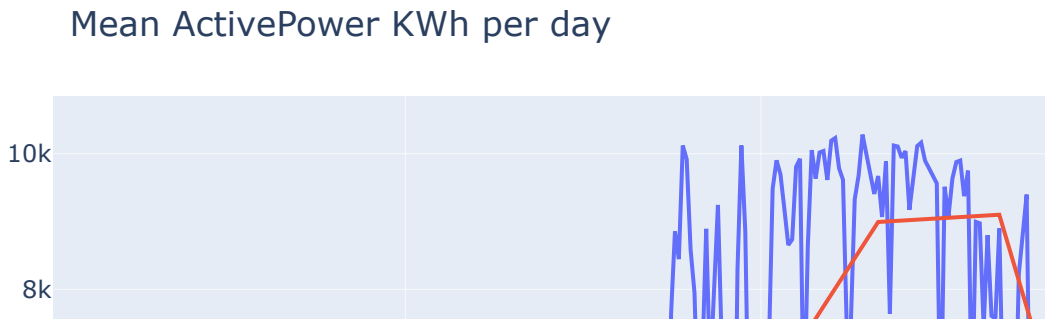
In [130…

```
# Active Power produced per day

daily_ap = hourly_turbine_data.loc[:,['ActivePower']].resample('D').sum()
monthly_ap = daily_ap.loc[:,['ActivePower']].resample('M').mean()

fig = px.line(daily_ap, y='ActivePower', x=daily_ap.index)
fig.add_trace(go.Scatter(y=monthly_ap['ActivePower'], x=monthly_ap.index,  mode='l
fig.update_layout(legend=dict(orientation="h",yanchor="top", y=1.02,xanchor="right'
                  title_text="Total daily ActivePower")
fig.show()
```



In [131…

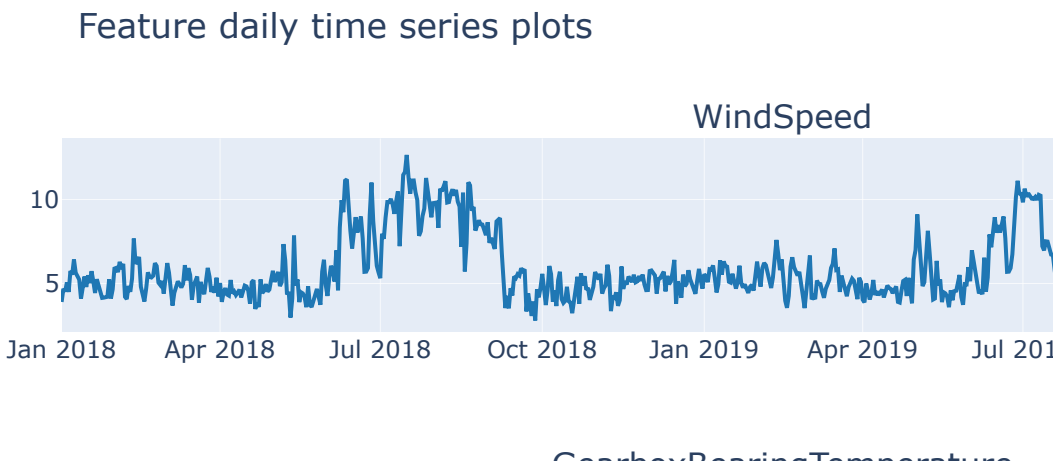
```
# Mean KW/h Active Power

daily_ap = hourly_turbine_data.loc[:,['ActivePower']].resample('D').mean()
monthly_ap = daily_ap.loc[:,['ActivePower']].resample('M').mean()

fig = px.line(daily_ap, y='ActivePower', x=daily_ap.index)
fig.add_trace(go.Scatter(y=monthly_ap['ActivePower'], x=monthly_ap.index,  mode='l
fig.update_layout(legend=dict(orientation="h",yanchor="top", y=1.02,xanchor="right'
                  title_text="Mean ActivePower KWh per day")
fig.show()
```



Now we can look at the mean daily values for our features

In [132…

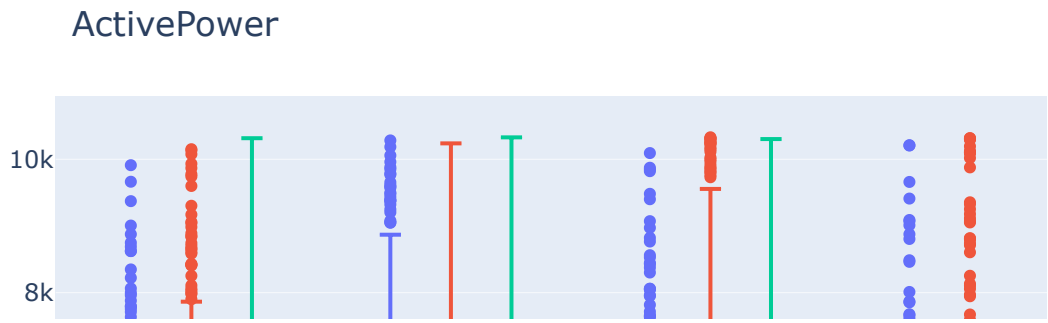
```
# Hourly data is too much for a time series so we will take daily
daily_means = hourly_turbine_data.loc[:, 'WindSpeed':].resample('D').mean()
monthly_means = hourly_turbine_data.loc[:, 'WindSpeed':].resample('M').mean()

fig = make_subplots(rows=4,  cols=2, subplot_titles=daily_means.columns)
for i, c in enumerate(daily_means.columns):

    fig.add_trace(go.Scatter(y=daily_means[c], x=daily_means.index,  mode='lines',
    #fig.add_trace(go.Scatter(y=monthly_means[c], x=monthly_means.index,  mode='li

fig.update_layout(showlegend=False, title_text='Feature daily time series plots', h
fig.show()
```

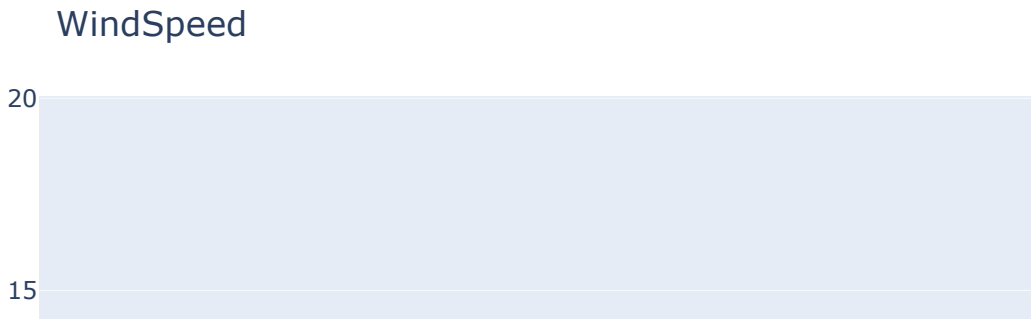


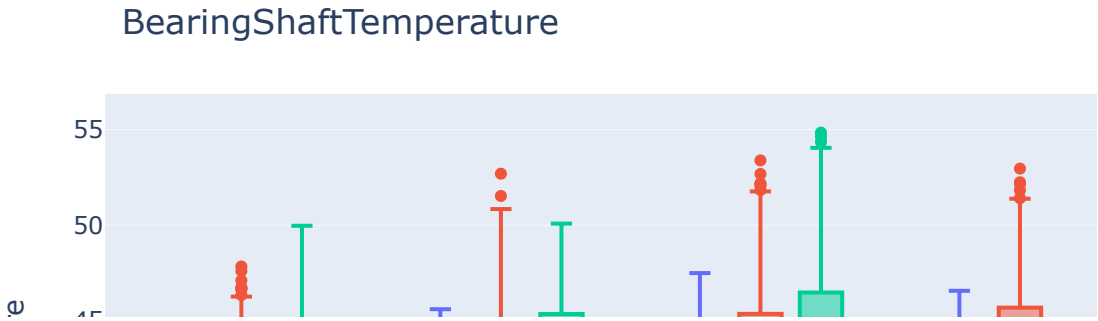
2.2.2 Box Plots

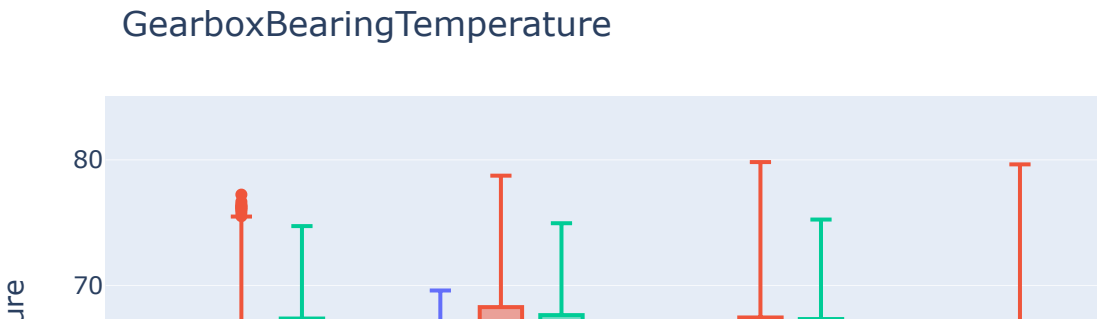
In [133…

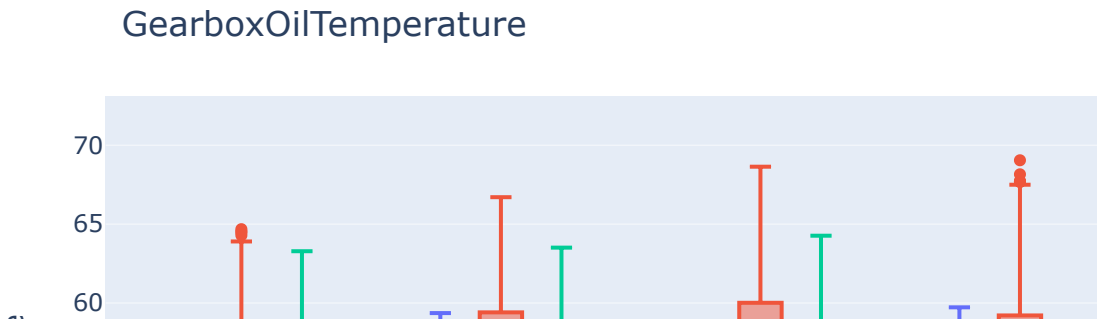
```
for c in hourly_turbine_data.columns:
    fig = px.box(hourly_turbine_data, y=c, x=hourly_turbine_data.index.month, colo
    fig.update_layout(title_text=c)
    fig.show()
```

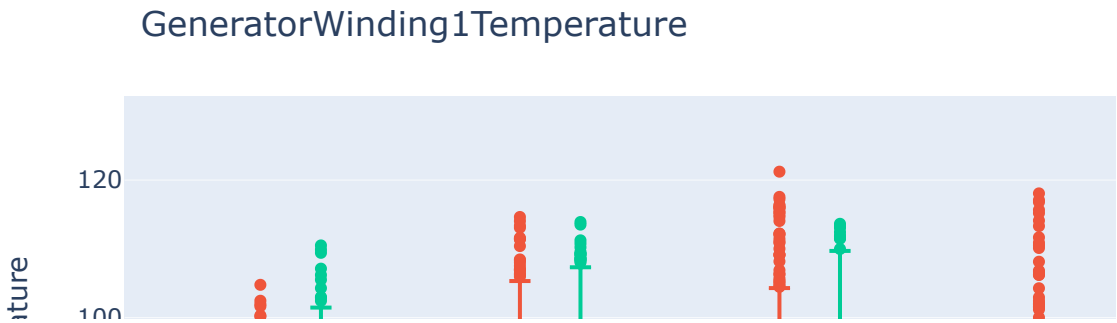

ActivePower
10k
8k

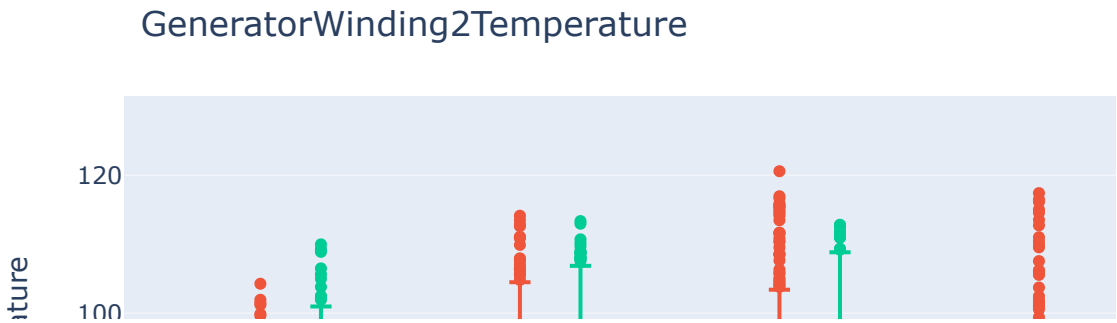

WindSpeed
20
15

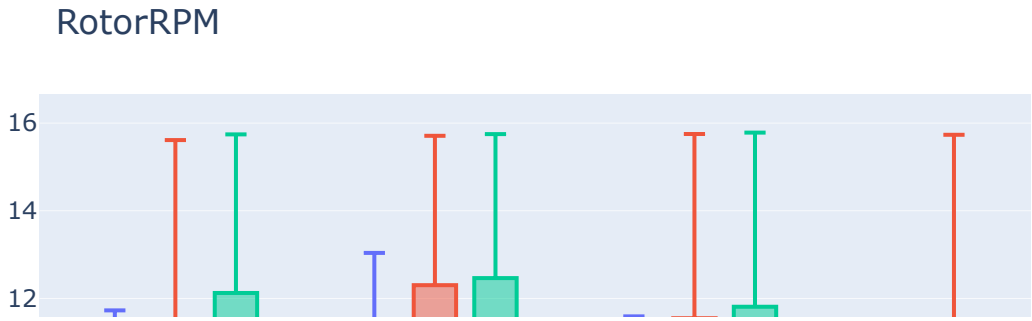

BearingShaftTemperature
55
50

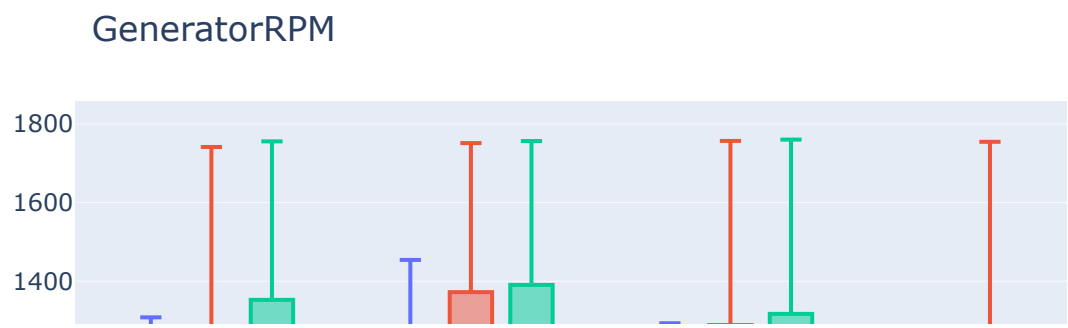

GearboxBearingTemperature
80
70


GearboxOilTemperature
70
65
60


GeneratorWinding1Temperature
120
100


GeneratorWinding2Temperature
120
100


RotorRPM
16
14
12



As we see from the box plots of the 2018 data, the large amount of missing values that were filled skews the data towards the mean. As a result we see a large number of samples which are marked as outliers for 2018 but fall in the IQ range of 2019 and 2020.

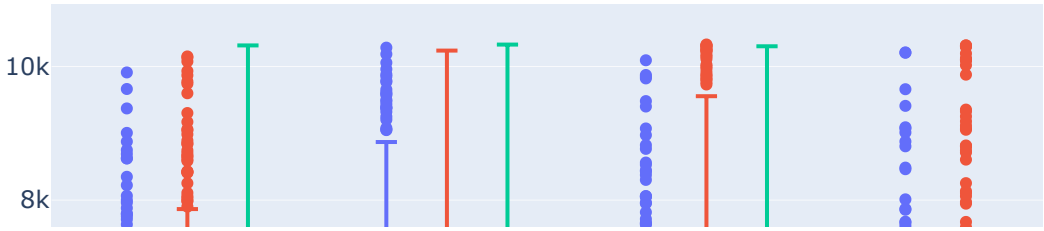
Based on this we will use the data from 2019 & 2020 to remove outliers.

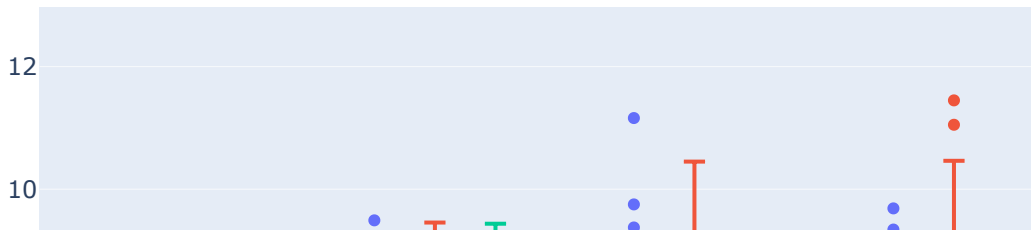
2.2.3 Removal of outliers

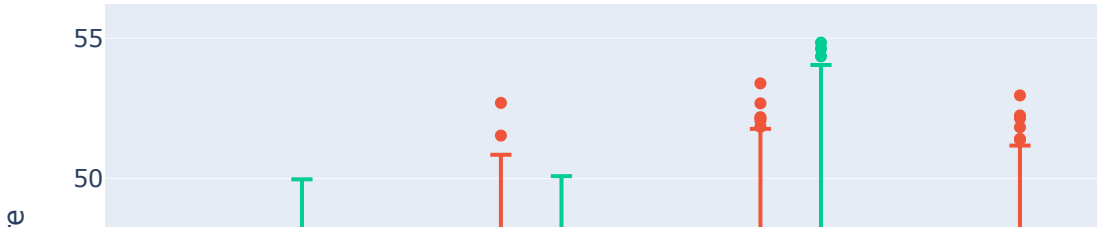
In [134…
```
q1 = hourly_turbine_data.loc['2019':].quantile(0.25)
q3 = hourly_turbine_data.loc['2019':].quantile(0.75)
iqr = q3 - q1

scrubbed_hourly_td = hourly_turbine_data[~((hourly_turbine_data < (q1 - 1.5 * iqr)
print(f'{scrubbed_hourly_td.shape[0]} from {hourly_turbine_data.shape[0]} rows, los
```

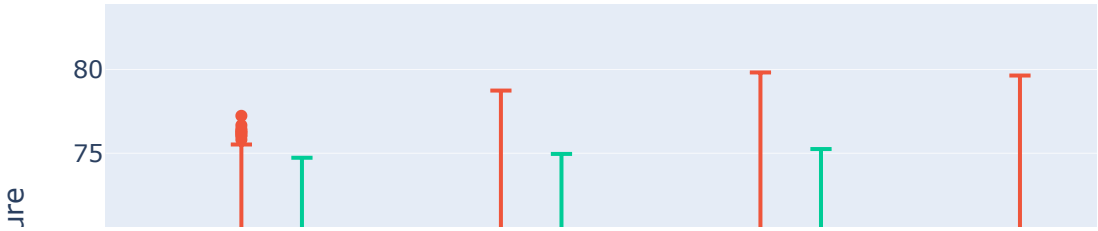
```
18884 from 19680 rows, loss of 796 rows
```

In [135…
```
for c in scrubbed_hourly_td.columns:
    fig = px.box(scrubbed_hourly_td, y=c, x=scrubbed_hourly_td.index.month, color=
    fig.show()
```

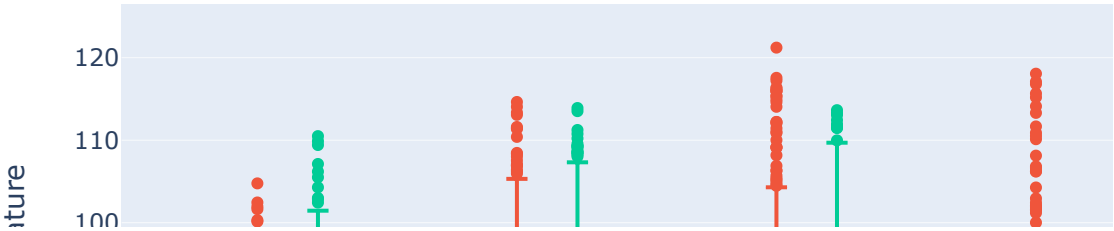

10k
8k

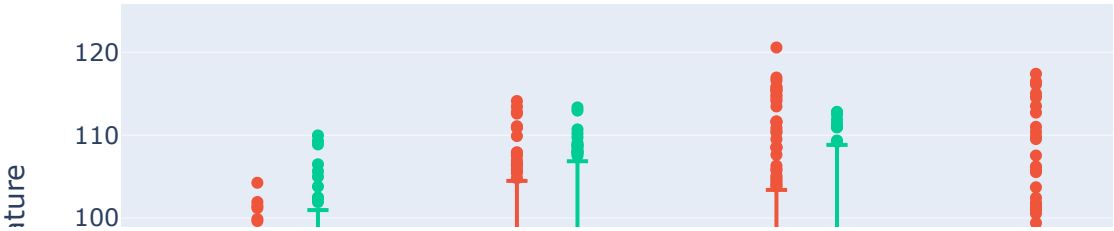

12
10

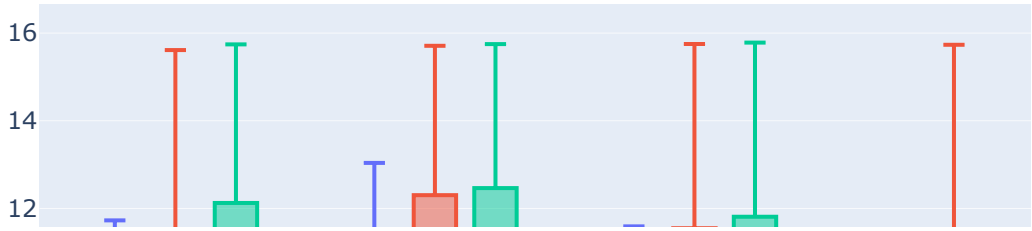

55
50
φ

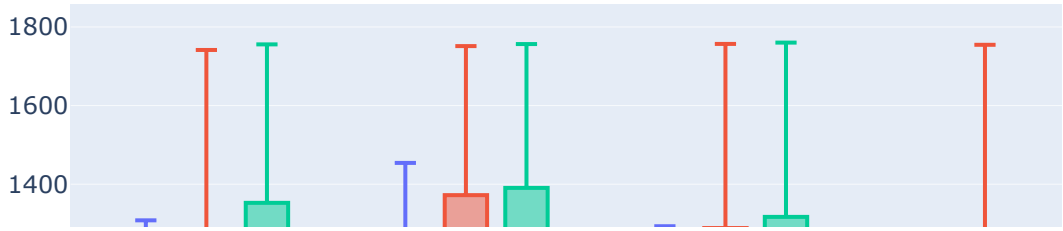

80
75


70
65


ature
120
110
100


ıture
120
110
100


16
14
12



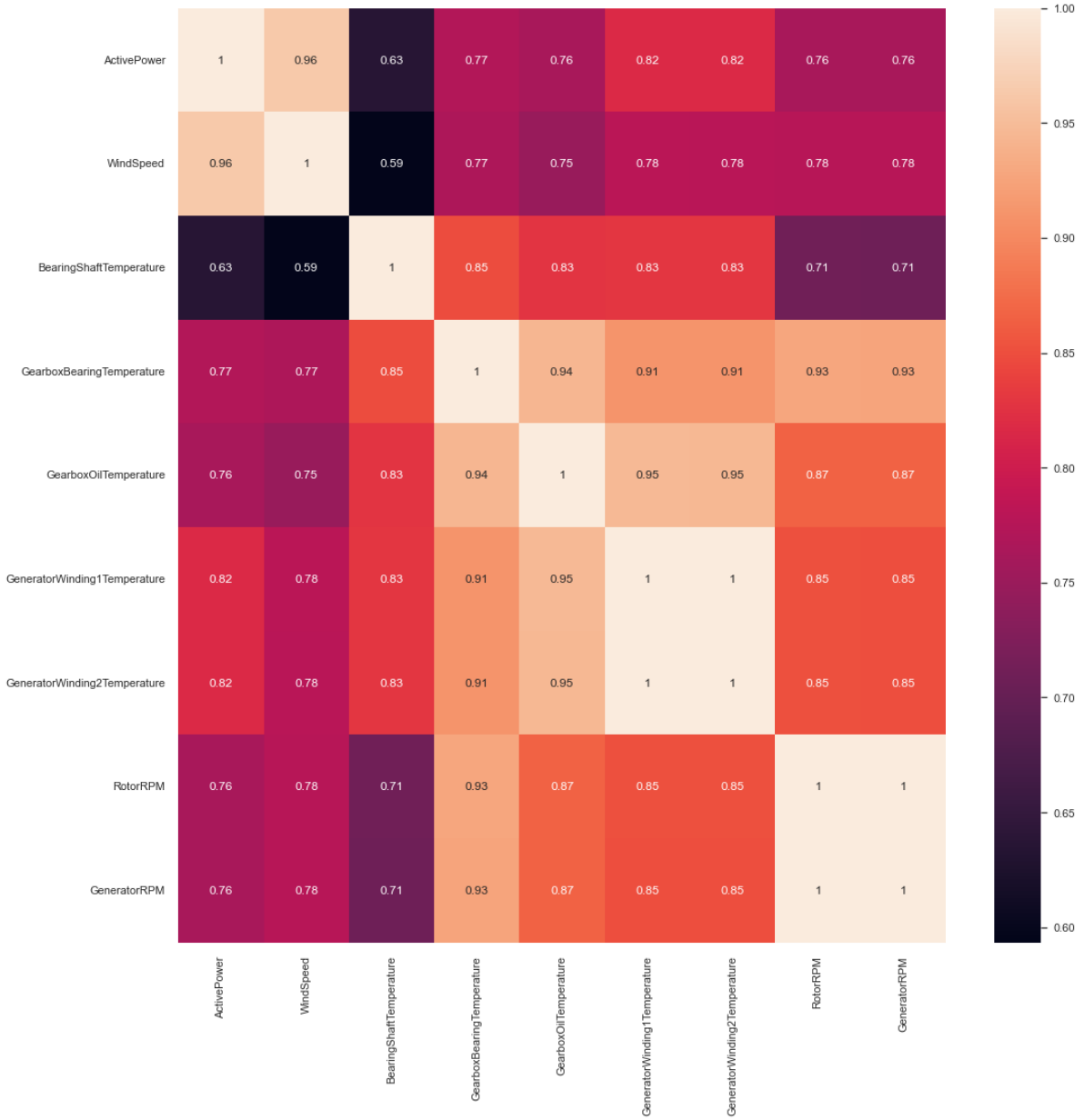
We see that this process of outlier removal drops our data by 796 rows, we also still see a number of outliers this is because we are splitting the data by month in the box plots but calculated the IQR based on the yearly data.

2.3 Bivariate analysis

2.3.1 Correlation heat maps

In [136…

```
_ = sns.heatmap(scrubbed_hourly_td.corr(), annot=True)
```



We see that the following pairs of fields are very closely related:

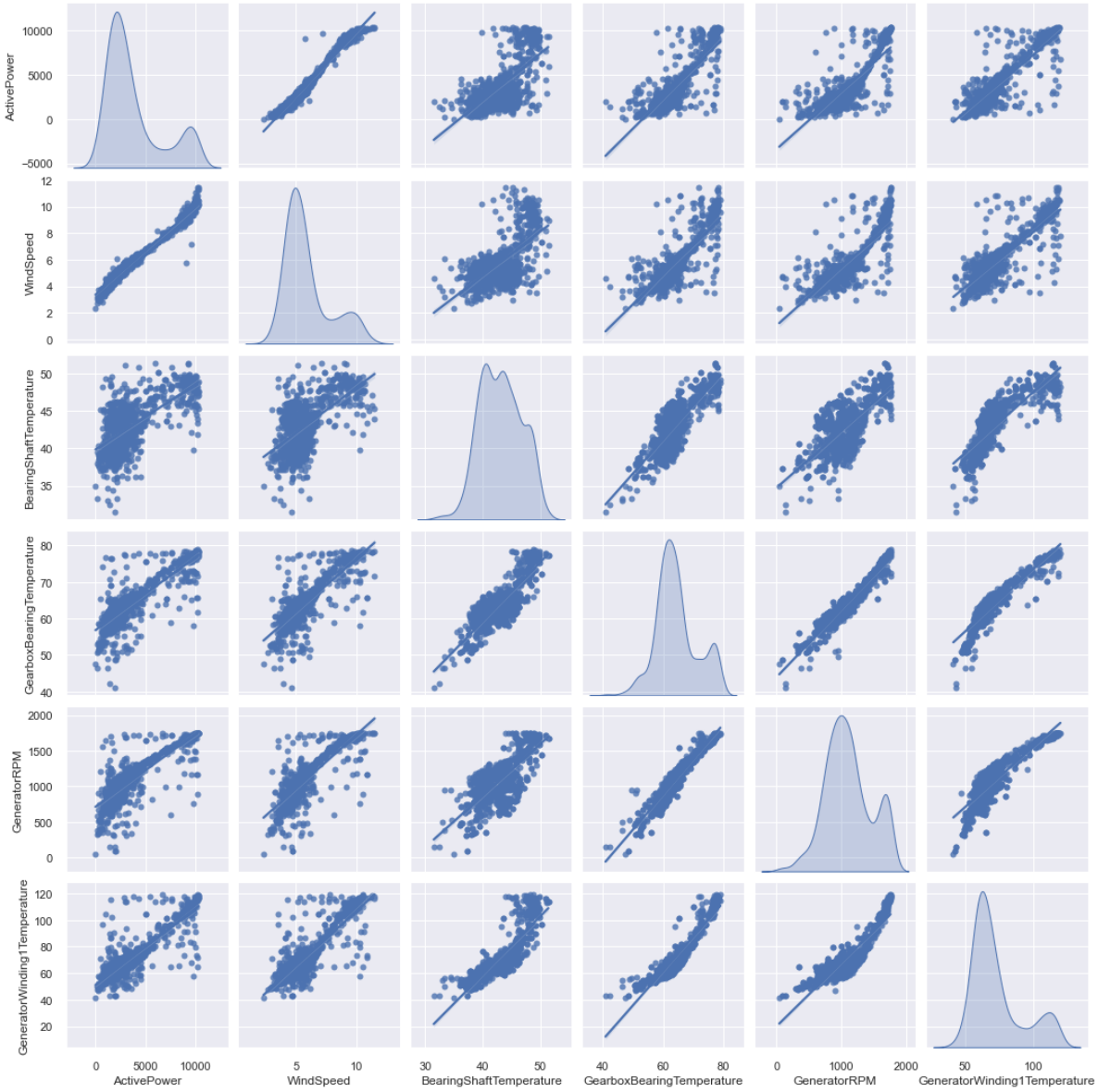
- GeneratorWinding1Temperature & GeneratorWinding2Temperature
- GeneratorRPM & RotorRPM
- GearboxBearingTemperature & GearboxOilTemperature

Since the pairs of fields both have the same correlation score with active power, we will keep only the first of the pair for the following plots.

2.3.2 Scatter plots

In [137…
```
selected_cols = ['WindSpeed', 'BearingShaftTemperature', 'GearboxBearingTemperature
daily_means = scrubbed_hourly_td.resample('D').mean()
```

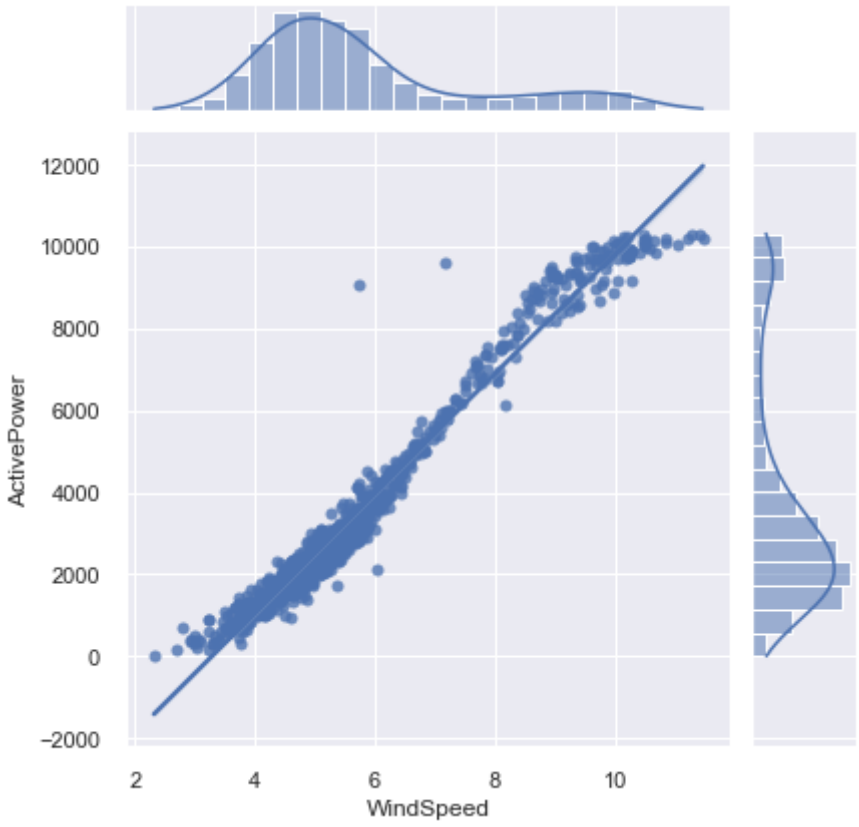
In [138…
```
_ = sns.pairplot(daily_means.loc[:, ['ActivePower'] + selected_cols ], kind='reg',
```

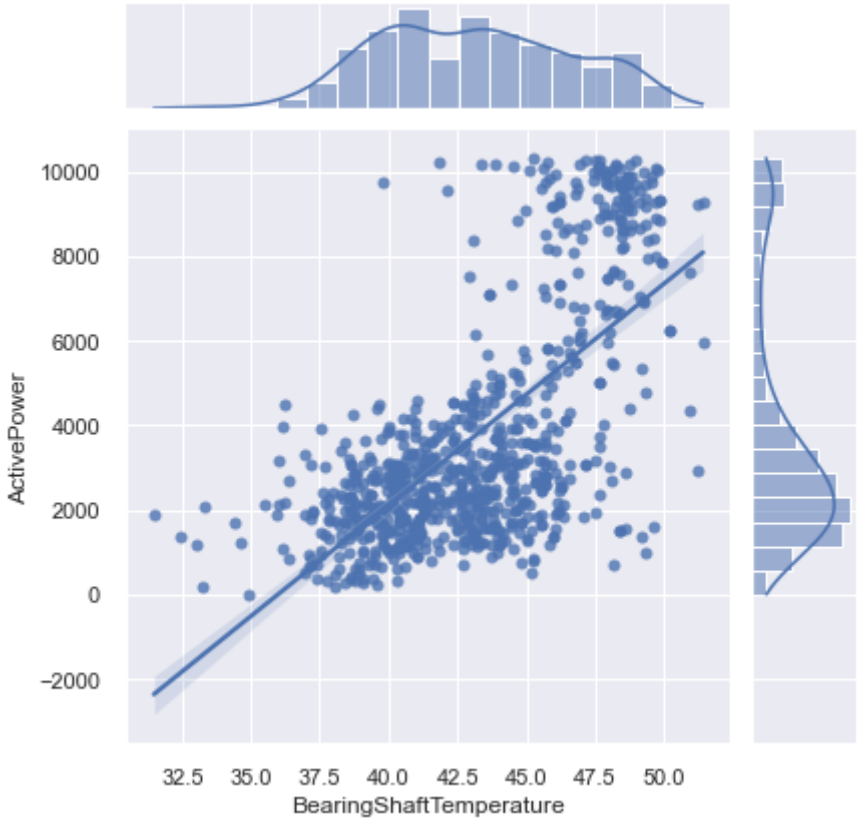


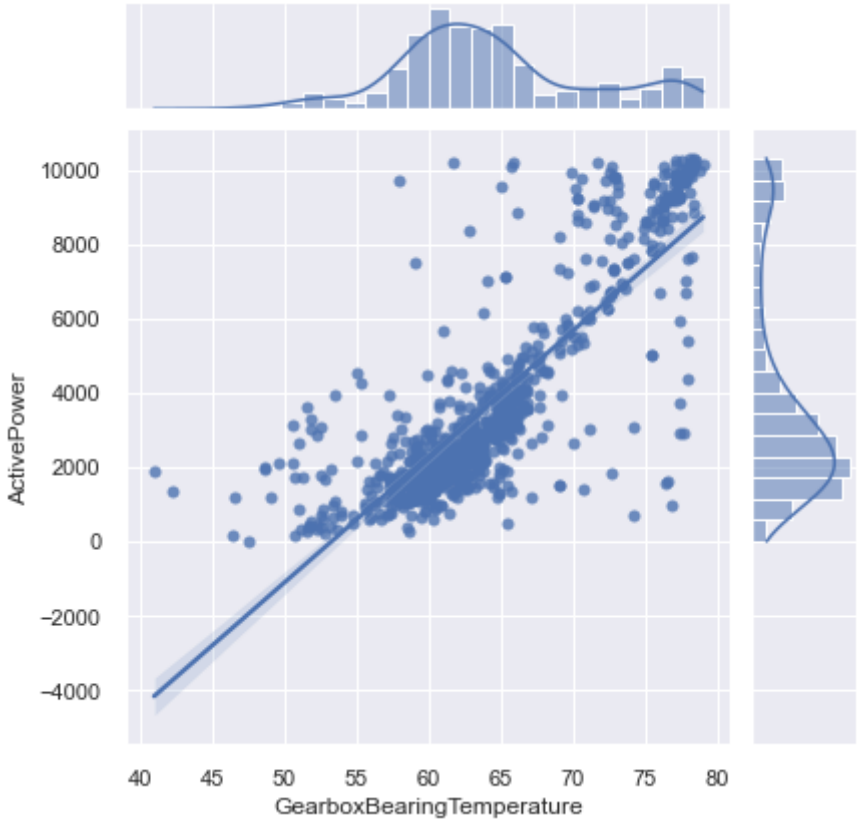
2.3.3 Joint Distribution plots

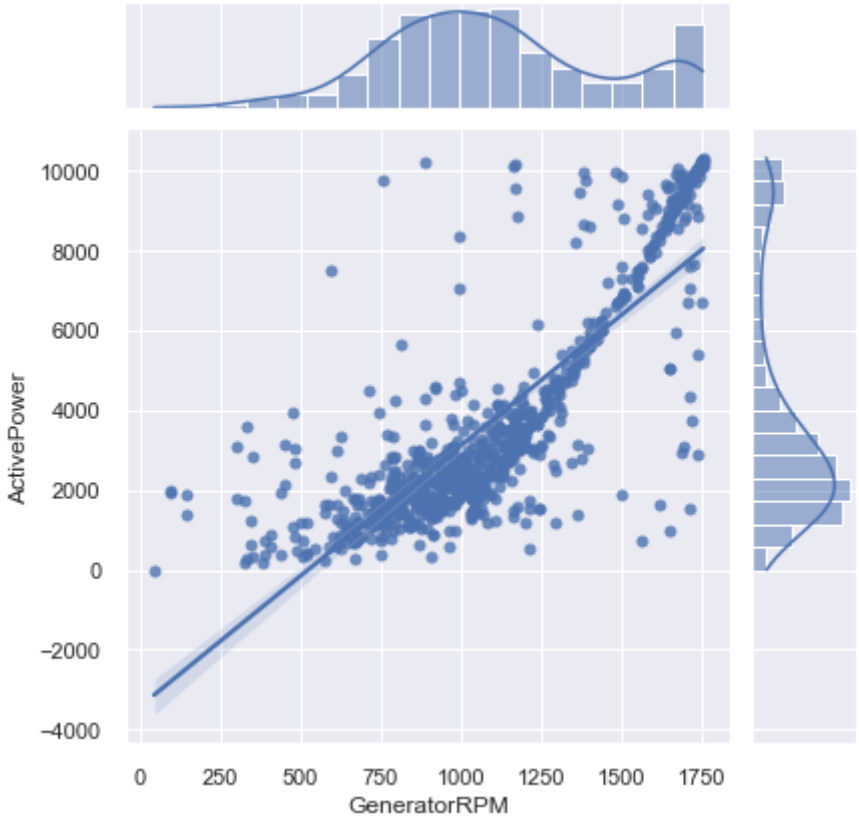
In [139…

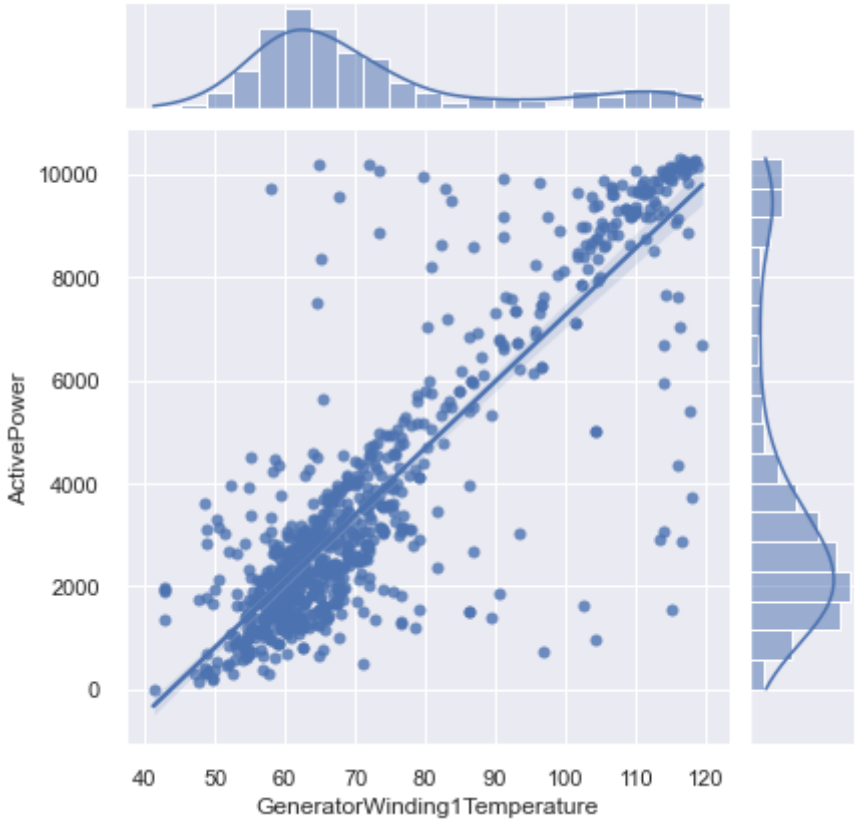
```
for c in selected_cols:
    sns.jointplot(data=daily_means, x=c, y='ActivePower', kind='reg')
```


12000
10000
8000
6000
4000
2000
0
−2000
ActivePower
2
4
6
8
10
WindSpeed


10000
8000
6000
4000
2000
0
−2000
ActivePower
32.5
35.0
37.5
40.0
42.5
45.0
47.5
50.0
BearingShaftTemperature


10000
8000
6000
4000
2000
0
−2000
−4000
ActivePower
40
45
50
55
60
65
70
75
80
GearboxBearingTemperature


10000
8000
6000
4000
2000
0
−2000
−4000
ActivePower
0
250
500
750
1000
1250
1500
1750
GeneratorRPM



As we can see from the Joint plots all we can fit a line quite well to describe the chosen variables relationship with active power.

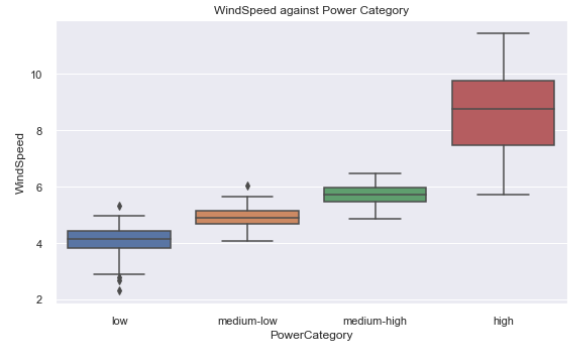
2.3.4 Category plot

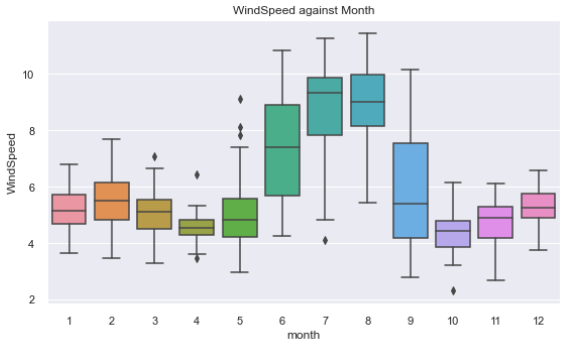
To do cat plots we will need some category for this we will bucket active power according to the quartiles, creating 4 categories of low, medium-low, medium-high and high power production.

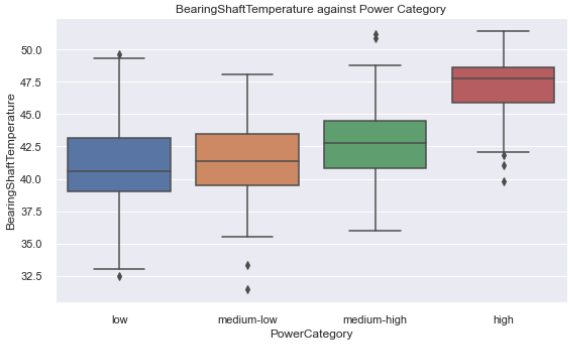
For this plot we look at the daily averages of each metric categorised by power production and the month.

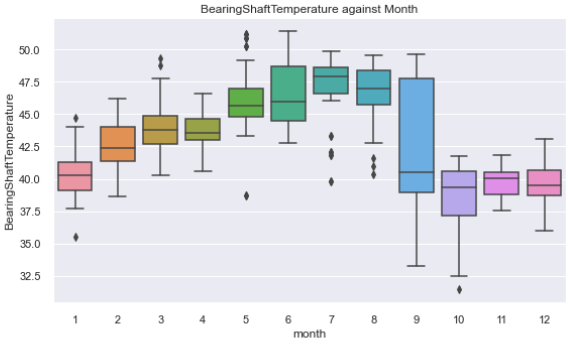
In [140…
```
bucketed_turbine_data = daily_means.copy()
bucketed_turbine_data['PowerCategory']  = pd.qcut(daily_means.ActivePower, 4, labe

# I also add month as columns since seaborn throws errors when using the dt index
bucketed_turbine_data['month'] = bucketed_turbine_data.index.month
```

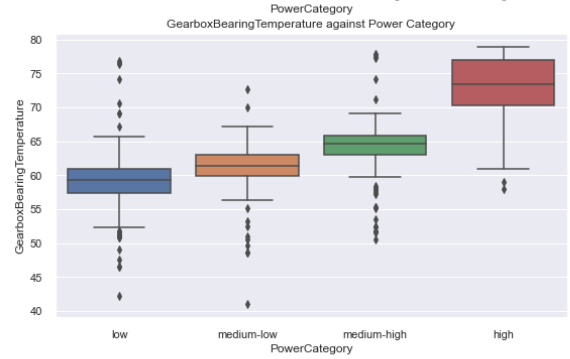
In [141…
```
fig, axis = plt.subplots(5, 2, figsize=(20,30))
#plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=0.15, hsp

for i, col in enumerate(selected_cols):
    ax = sns.boxplot(data=bucketed_turbine_data, y=col,  x="PowerCategory", ax=axis
    ax.set_title(f'{col} against Power Category')
    ax = sns.boxplot(data=bucketed_turbine_data, y=col,  x="month", ax=axis[i, 1])
    ax.set_title(f'{col} against Month')
```

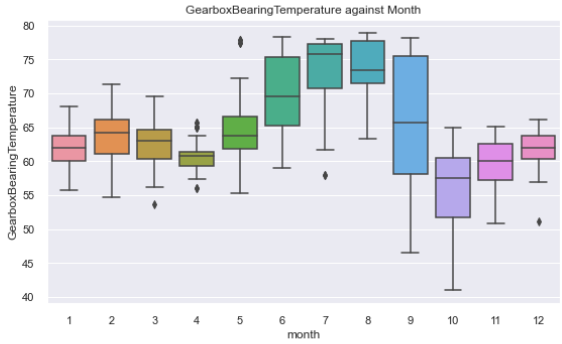


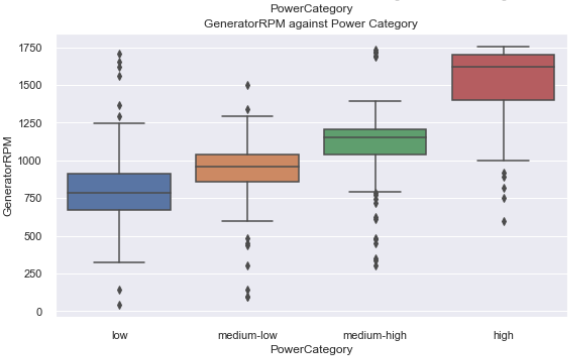


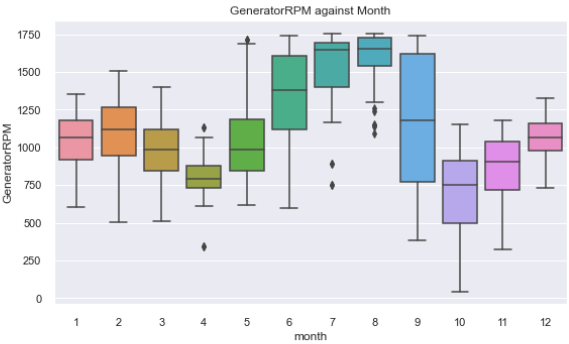


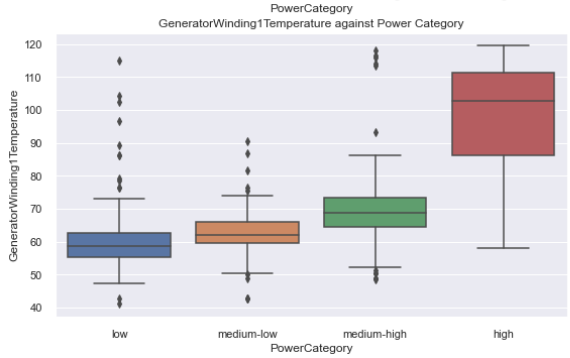


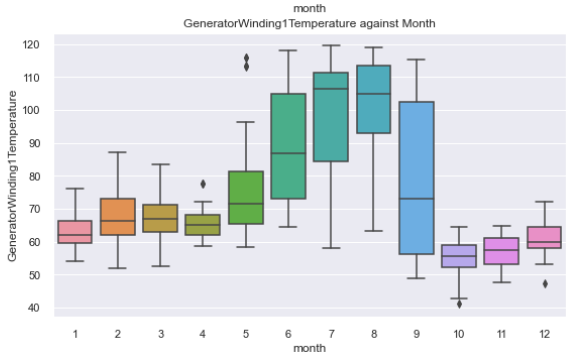












2.3.5 Hypothesis testing

We will test the following hypothesis:

1. High power levels are a result of higher winds
   A. H_0: There is no difference in means of wind speed between high power level and other classes

B. H_a: The mean wind speed of high power levels is *greater* than that of other power levels

2. Summer months (6,7,8) produce higher wind speeds
   A. H_0: There is no difference in means of wind speed between summer months and other months
   B. H_a: The mean wind speed in summer months is *greater* than that of other months
3. Higher power levels produce higher component temperatures
   A. H_0: There is no difference in means of temperature high power level and other classes
   B. H_a: The mean of temperature of high power level is *greater* than that of lower power levels
4. Wind speed greater than 10m/s has no effect on active power produced
   A. H_0: The mean active power produced between 9-10m/s is the same as that produces by wind speeds greater than 10m/s
   B. H_a: The mean power produced by wind speeds greater than 10m/s than of that produced by wind speeds between 9-10m/s

In [142…
```
# Testing case 1 high power levels are a result of high windspeeds
print(f"Mean wind speed for high power level class: {bucketed_turbine_data.loc[bucl
print(f"Mean wind speed for lower power level class: {bucketed_turbine_data.loc[bu

score, p_val = stats.ttest_ind(bucketed_turbine_data.loc[bucketed_turbine_data['Pow
                               bucketed_turbine_data.loc[bucketed_turbine_data['Power
                               equal_var=False)
if p_val<0.05:
    print(f'P-val of {p_val:0.2e} < 0.05 therefore we reject the null hypothesis.'
else:
    print(f'P-val of {p_val:0.2e} > 0.05 therefore we accept the null hypothesis.'
```

```
Mean wind speed for high power level class: 8.64 m/s
Mean wind speed for lower power level class: 4.91 m/s
P-val of 4.58e-104 < 0.05 therefore we reject the null hypothesis.
```

In [143…
```
# Testing case 2 Summer months (6,7,8) produce higher wind speeds
summer_months = bucketed_turbine_data.loc[bucketed_turbine_data['month'].isin([6,7
other_months = bucketed_turbine_data.loc[~(bucketed_turbine_data['month'].isin([6,
print(f"Mean wind speed for summer months: {summer_months.mean():0.2f} m/s")
print(f"Mean wind speed for other months: {other_months.mean():0.2f} m/s")

score, p_val = stats.ttest_ind(summer_months,
                               other_months,
                               equal_var=False)
if p_val<0.05:
    print(f'P-val of {p_val:0.2e} < 0.05 therefore we reject the null hypothesis.'
else:
    print(f'P-val of {p_val:0.2e} > 0.05 therefore we accept the null hypothesis.'
```

```
Mean wind speed for summer months: 8.34 m/s
Mean wind speed for other months: 5.10 m/s
P-val of 4.22e-62 < 0.05 therefore we reject the null hypothesis.
```

In [144…
```
# Testing case 3 Higher power levels produce higher component temperatures

for comp_temp in ['BearingShaftTemperature', 'GearboxBearingTemperature','Generato


    high_pl = bucketed_turbine_data.loc[bucketed_turbine_data['PowerCategory']=='h
    other_pl = bucketed_turbine_data.loc[bucketed_turbine_data['PowerCategory']!='
```

```
    print(f'\nSub-case: {comp_temp} ---:')
    print(f"Mean {comp_temp} for high power level class: {high_pl.mean():0.2f} c")
    print(f"Mean {comp_temp} for lower power level class: {other_pl.mean():0.2f} c

    score, p_val = stats.ttest_ind(high_pl,
                                   other_pl,
                                   equal_var=False)
    if p_val<0.05:
        print(f'P-val of {p_val:0.2e} < 0.05 therefore we reject the null hypothes
    else:
        print(f'P-val of {p_val:0.2e} > 0.05 therefore we accept the null hypothes
```

```
Sub-case: BearingShaftTemperature ---:
Mean BearingShaftTemperature for high power level class: 47.19 c
Mean BearingShaftTemperature for lower power level class: 41.78 c
P-val of 4.46e-107 < 0.05 therefore we reject the null hypothesis.

Sub-case: GearboxBearingTemperature ---:
Mean GearboxBearingTemperature for high power level class: 73.03 c
Mean GearboxBearingTemperature for lower power level class: 61.40 c
P-val of 1.05e-102 < 0.05 therefore we reject the null hypothesis.

Sub-case: GeneratorWinding1Temperature ---:
Mean GeneratorWinding1Temperature for high power level class: 98.40 c
Mean GeneratorWinding1Temperature for lower power level class: 64.15 c
P-val of 5.61e-86 < 0.05 therefore we reject the null hypothesis.
```

In [145…

```
# Wind speed greater than 10m/s has no effect on active power produced
lower_bound_filter = ((bucketed_turbine_data['WindSpeed'] >= 9) & (bucketed_turbine
lower_bound_ws = bucketed_turbine_data.loc[lower_bound_filter, 'ActivePower'].dropn
upper_bound_ws = bucketed_turbine_data.loc[bucketed_turbine_data['WindSpeed'] > 10

print(f"Mean active power for 9-10 m/s wind speeds: {lower_bound_ws.mean():0.2f} kW
print(f"Mean active power for >10 m/s wind speeds: {upper_bound_ws.mean():0.2f} kWs

score, p_val = stats.ttest_ind(lower_bound_ws,
                               upper_bound_ws,
                               equal_var=False)
if p_val<0.05:
    print(f'P-val of {p_val:0.2e} < 0.05 therefore we reject the null hypothesis.')
else:
    print(f'P-val of {p_val:0.2e} > 0.05 therefore we accept the null hypothesis.')
```

```
Mean active power for 9-10 m/s wind speeds: 9309.85 kWs
Mean active power for >10 m/s wind speeds: 9976.89 kWs
P-val of 1.75e-12 < 0.05 therefore we reject the null hypothesis.
```

3. Clustering

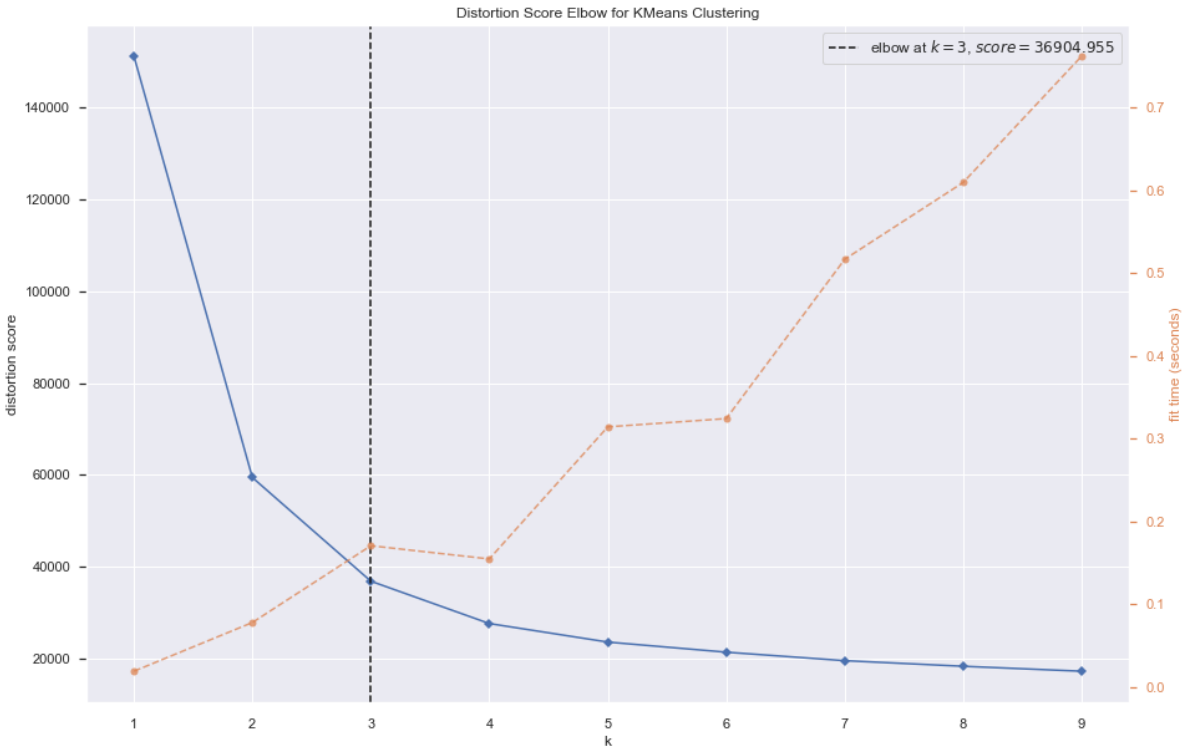
3.1 Identifying the number of clusters

Here will will use the Yellowbrick visualization tools to determine the appropriate number of clusters using the elbow method. We use two evaluation metrics, dispersion and silhouette score.

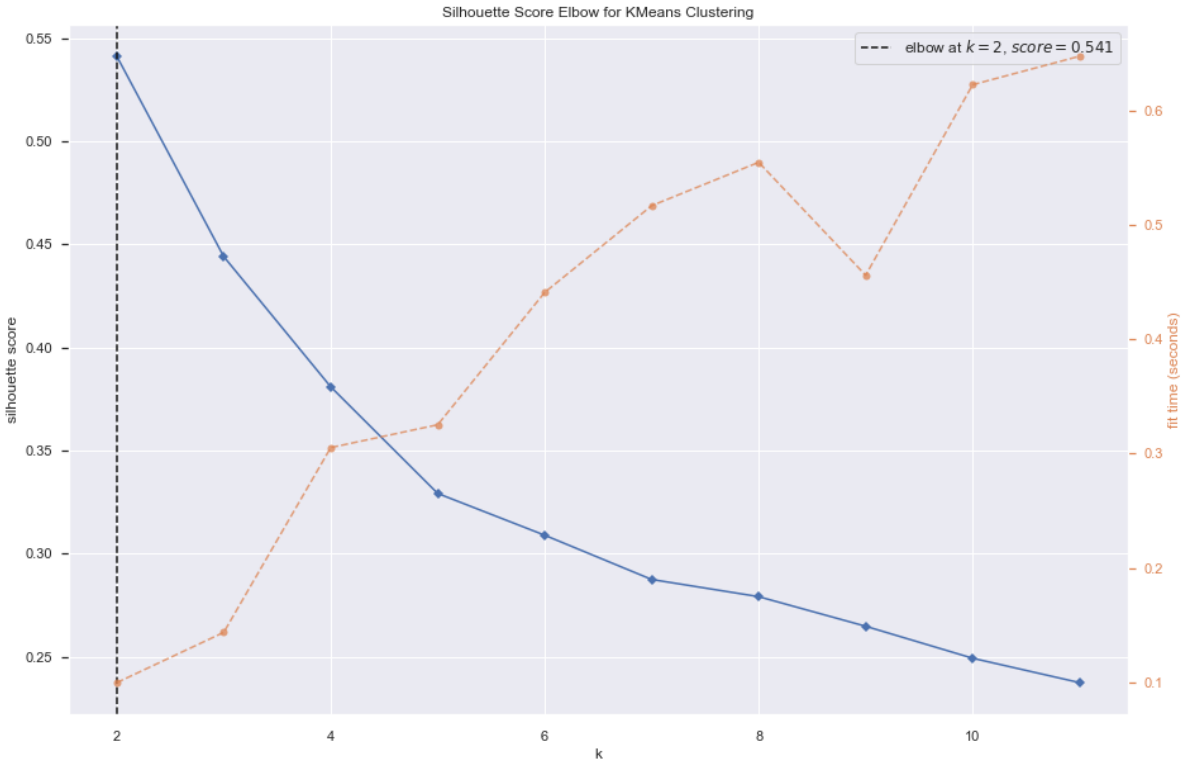
In [146…

```
# pre processing the data
X = scrubbed_hourly_td.drop(['ActivePower'], axis=1)
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [147…
```
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,10), size=(1080, 720))

visualizer.fit(X)

_ = visualizer.show()
```



In [148…
```
model = KMeans()
visualizer = KElbowVisualizer(model, k=(2,12), metric='silhouette', size=(1080, 72

visualizer.fit(X)
_ = visualizer.show()
```



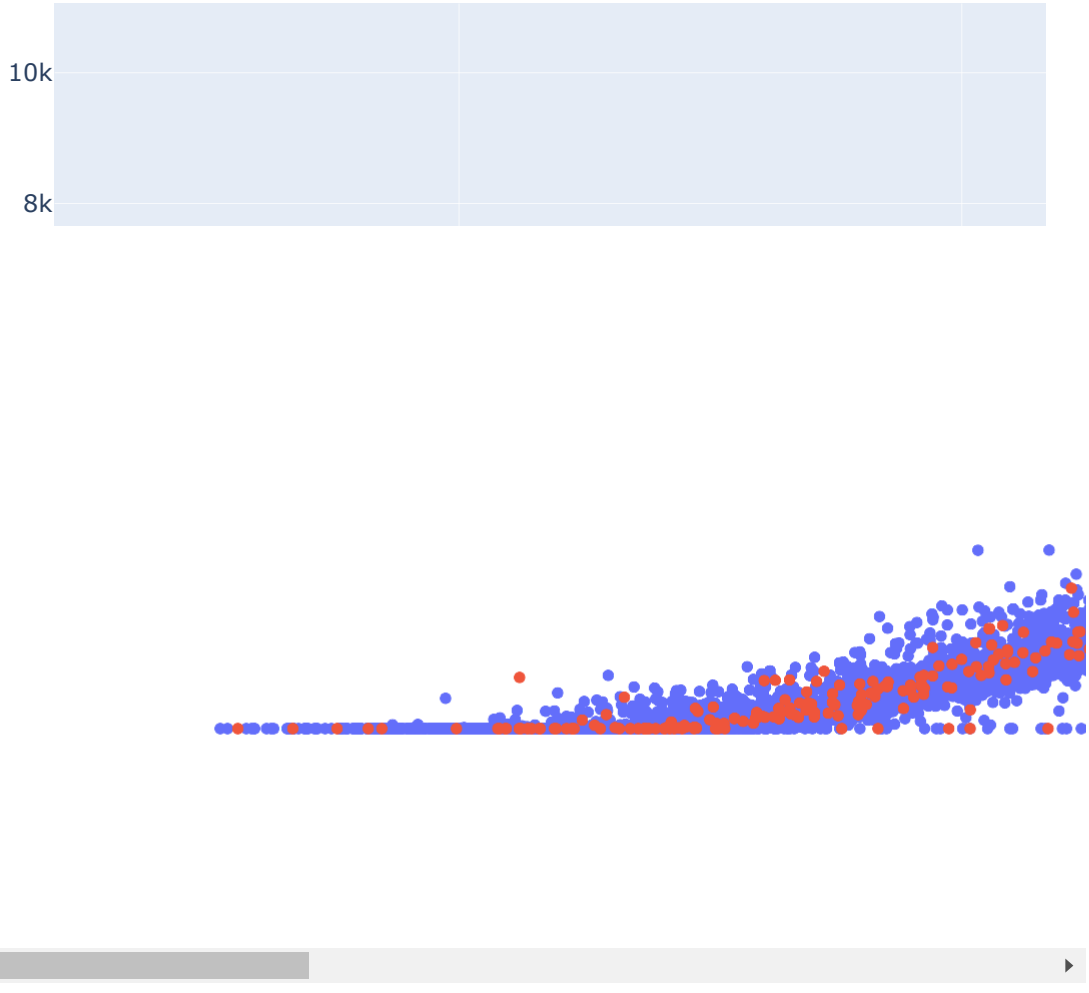
We see that using dispersion we get 3 clusters while silhouette gets indicates 2 would be better, we now cluster each point and make coloured scatter plots to see the effect of

clustering.

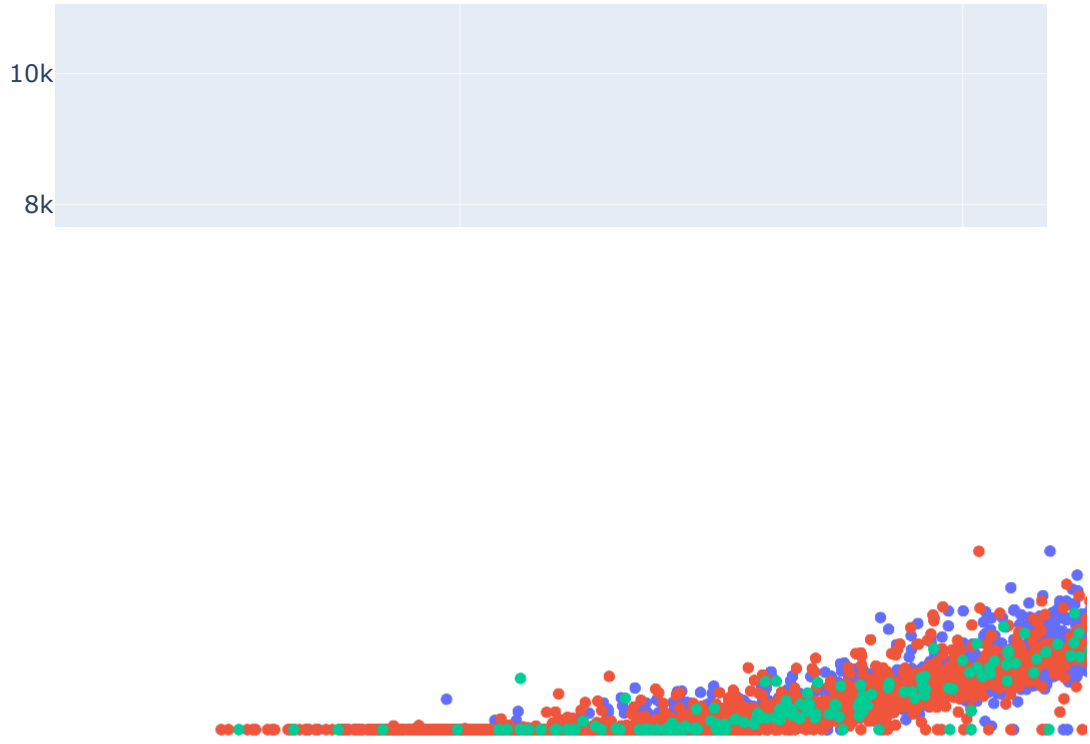
3.2 Clustering with K-means

In [149…
```
scrubbed_hourly_td['kmeans_2_cluster'] = KMeans(n_clusters=2).fit_predict(X)
scrubbed_hourly_td['kmeans_3_cluster'] = KMeans(n_clusters=3).fit_predict(X)
```

In [150…
```
px.scatter(scrubbed_hourly_td, y='ActivePower', x='WindSpeed', color=scrubbed_hour
```



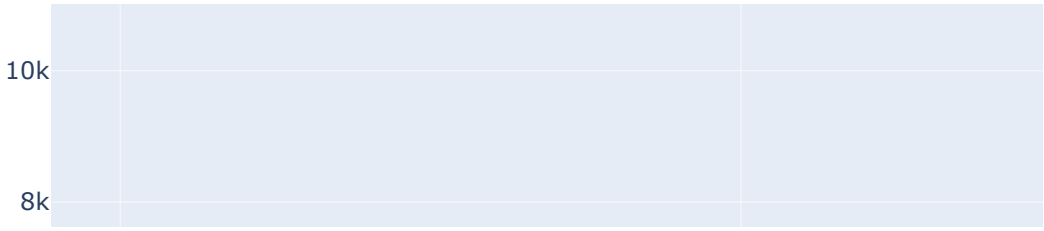
In [151…
```
px.scatter(scrubbed_hourly_td, y='ActivePower', x='WindSpeed', color=scrubbed_hour
```



We see that with the hourly data we have very poor results in both the 3 and 2 cluster case. We now try considering the daily avg power level with 2 clusters

In [152…
```
# removing missing days caused by resampling
daily_means = daily_means.dropna()

X_dm = daily_means.drop(['ActivePower'], axis=1)
scaler = StandardScaler()
X_dm = scaler.fit_transform(X_dm)
daily_means['kmeans_2_cluster'] = KMeans(n_clusters=2).fit_predict(X_dm)
```

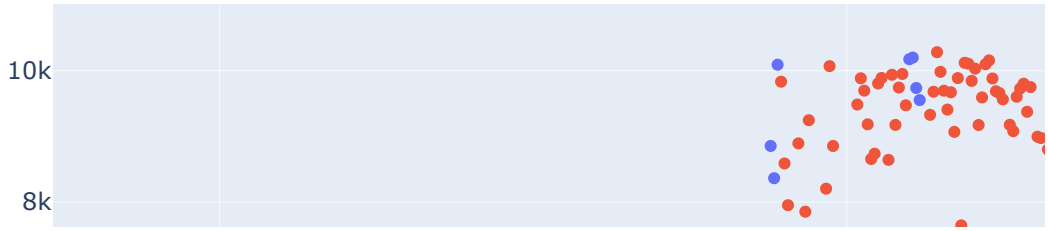
In [153…
```
px.scatter(daily_means, y='ActivePower', x='WindSpeed', color=daily_means['kmeans_
```



This produces a much better result, from which we can conclude that K means is not capable of handling the noisy hourly level data, but can be used to cluster the daily averages. We now consider the cluster in the time dimension.

In [154…

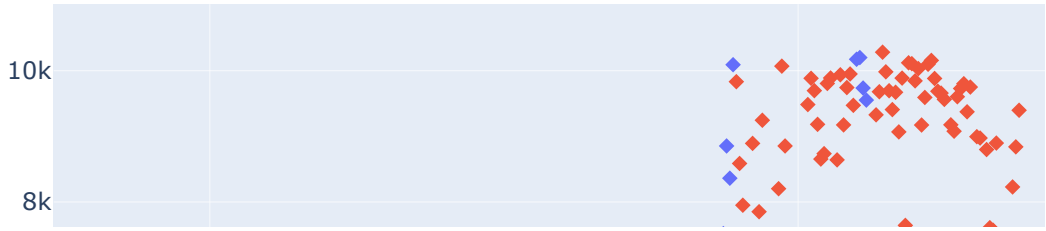
```
px.scatter(daily_means, y='ActivePower', x=daily_means.index, color=daily_means['kr
```



From this we can see that with out reference to the time KMeans is capable of identifying the periods of high power production. From this result we can also conclude that the 4 artificial categories we made above are too many. We will categorise them into two classes, high power vs everything else and compare them with the results of Kmeans.

In [155…

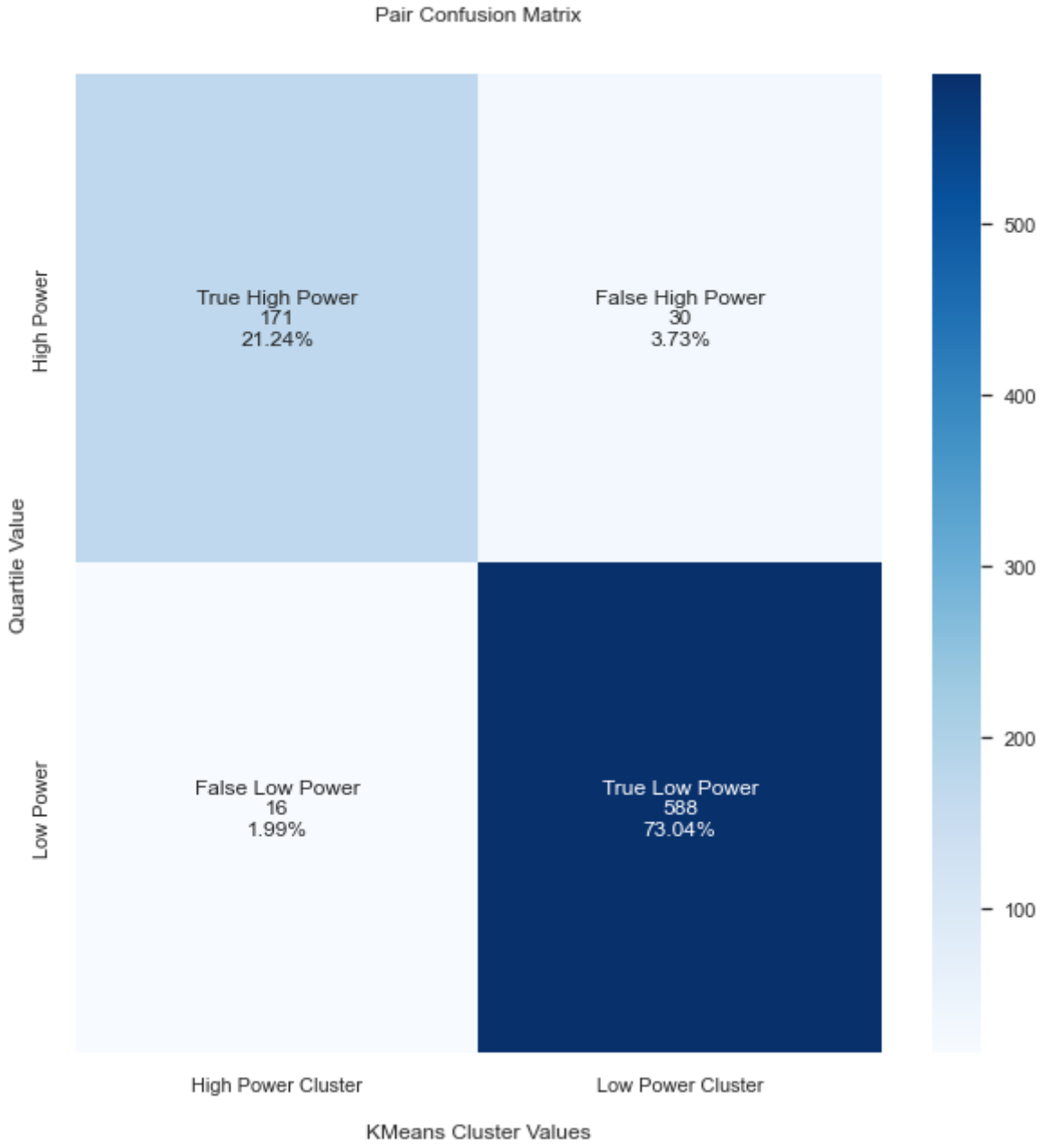
```
# Adding the manually labelled data to daily aggregate
daily_means['HighPowerProd'] = bucketed_turbine_data['PowerCategory'] == 'high'
px.scatter(daily_means, y='ActivePower', x=daily_means.index, color=daily_means['kr
```



From this plot we see that the majority of the high power data points are fit into a single class.

In [156...

```
thp = sum((daily_means['HighPowerProd'] == True) & (daily_means['kmeans_2_cluster']
fhp = sum((daily_means['HighPowerProd'] == True) & (daily_means['kmeans_2_cluster']
tlp = sum((daily_means['HighPowerProd'] == False) & (daily_means['kmeans_2_cluster
flp = sum((daily_means['HighPowerProd'] == False) & (daily_means['kmeans_2_cluster

# confusion like matrix
pcf = np.array([
                    [thp, fhp],
                    [flp, tlp]
                ])

group_names = ['True High Power','False High Power','False Low Power','True Low Pou
group_counts = ["{0:0.0f}".format(value) for value in pcf.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in pcf.flatten()/np.sum(pcf

labels = np.array([f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,group_cou
fig, ax = plt.subplots(figsize=(10,10))
ax = sns.heatmap(pcf, annot=labels, fmt='', cmap='Blues', ax=ax)

ax.set_title('Pair Confusion Matrix\n\n')
ax.set_xlabel('\nKMeans Cluster Values')
ax.set_ylabel('Quartile Value')

ax.xaxis.set_ticklabels(['High Power Cluster','Low Power Cluster'])
ax.yaxis.set_ticklabels(['High Power','Low Power'])
plt.show()
```

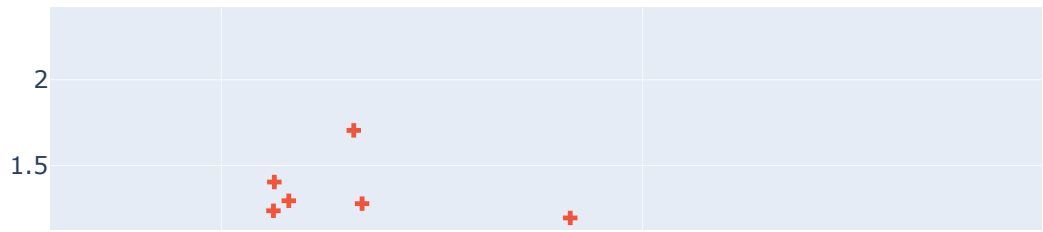


From the pair confusion matrix we can see that, if we take the quartile label as bing correct, KMeans can correctly cluster 94.28% of the data as high power or low.

4. PCA

In [157…
```
X_pca=PCA(n_components=2).fit_transform(X_dm)
```

In [158…
```
fig = px.scatter(x=X_pca[:, 0], y=X_pca[:, 1], color=daily_means['kmeans_2_cluster
legend_names = {'0, False': 'True Low Power', '0, True': 'False High Power', '1, Fa
fig.for_each_trace(lambda t: t.update(name=legend_names[t.name], legendgroup =leger
fig.update_layout(legend_title='Classes', title_text='PCA with Kmeans and Quartile
fig.show()
```



From PCA we can see two groups that have been highlighted by the KMeans clustering (shown by color). This also helps explain why some of the high power points (crosses) are clustered as low power (circles) and vice versa.

In [159…
```
# Adding PC components to daily aggregate
daily_means['PC1'] = X_pca[:, 0]
daily_means['PC2'] = X_pca[:, 1]
```

In [160…
```
# Saving the data
save_path = os.path.join(os.pardir, 'data', 'processed_daily_avg_turbine_data.csv')
daily_means.to_csv(save_path)
```

In [161…
```
# Saving the data
save_path = os.path.join(os.pardir, 'data', 'processed_hourly_avg_turbine_data.csv
scrubbed_hourly_td.to_csv(save_path)
```