Analysis of gas emissions from gas turbine

References

- https://archive.ics.uci.edu/ml/datasets/Gas+Turbine+CO+and+NOx+Emission+Data+Set (https://archive.ics.uci.edu/ml/datasets/Gas+Turbine+CO+and+NOx+Emission+Data+Set)
- https://journals.tubitak.gov.tr/elektrik/issues/elk-19-27-6/elk-27-6-54-1807-87.pdf (https://journals.tubitak.gov.tr/elektrik/issues/elk-19-27-6/elk-27-6-54-1807-87.pdf)
- https://towardsdatascience.com/build-your-own-neural-network-from-scratch-with-python-dbe0282bd9e3 (https://towardsdatascience.com/build-your-own-neural-network-from-scratch-with-python-dbe0282bd9e3)
- https://thecodacus.com/2017/08/14/neural-network-scratch-python-no-libraries/ (https://thecodacus.com/2017/08/14/neural-network-scratch-python-no-libraries/)
- https://machinelearningmastery.com/regression-tutorial-keras-deep-learning-library-python/ (https://machinelearningmastery.com/regression-tutorial-keras-deep-learning-library-python/)
- https://machinelearningmastery.com/deep-learning-models-for-multi-output-regression/ (https://machinelearningmastery.com/deep-learning-models-for-multi-output-regression/)
- https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/)
- https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html (https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html)
- https://stackoverflow.com/questions/12893492/choosing-eps-and-minpts-for-dbscan-r (https://stackoverflow.com/questions/12893492/choosing-eps-and-minpts-for-dbscan-r)
- https://towardsdatascience.com/explaining-dbscan-clustering-18eaf5c83b31 (https://towardsdatascience.com/explaining-dbscan-clustering-18eaf5c83b31)
- Machine Intelligence - Lecture 7 (Clustering, k-means, SOM) University of Waterloo : https://www.youtube.com/watch?v=lFbxTlD5R98 (https://www.youtube.com/watch?v=lFbxTlD5R98)
- SELF ORGANISING MAPS: INTRODUCTION: https://www.youtube.com/watch?v=0qtvb_Nx2tA (https://www.youtube.com/watch?v=0qtvb_Nx2tA)
- https://stackoverflow.com/questions/43160240/how-to-plot-a-k-distance-graph-in-python (https://stackoverflow.com/questions/43160240/how-to-plot-a-k-distance-graph-in-python)
- https://medium.com/machine-learning-researcher/self-organizing-map-som-c296561e2117 (https://medium.com/machine-learning-researcher/self-organizing-map-som-c296561e2117)
- som documentation - https://github.com/JustGlowing/minisom/blob/master/examples/Clustering.ipynb (https://github.com/JustGlowing/minisom/blob/master/examples/Clustering.ipynb)

About the dataset

The dataset contains 36733 instances of 11 sensor measures aggregated over one hour, from a gas turbine located in Turkey for the purpose of studying flue gas emissions, namely CO and NOx.

Why is this analysis even necessary ?

- We don't have much time to save our planet from irreversible changes
- Thus the analysis of gas emissions from industries is well needed so as to reduce the pollution and control the climate change.

- Flue gases here: carbon monoxide and nitrogen oxides are greenhouse gases and thus contribute majorly towards change in climate which needs to be controlled first before we can learn to reverse it.

Data is sourced from UCI ML repository and the dataset was uploaded on 29th November'2019.

Attribute Information (as uploaded on UCI ML website):

The explanations of sensor measurements and their brief statistics are given below.

- Variable (Abbr.) Unit Min Max Mean
- Ambient temperature (AT) C â€“6.23 37.10 17.71
- Ambient pressure (AP) mbar 985.85 1036.56 1013.07
- Ambient humidity (AH) (%) 24.08 100.20 77.87
- Air filter difference pressure (AFDP) mbar 2.09 7.61 3.93
- Gas turbine exhaust pressure (GTEP) mbar 17.70 40.72 25.56
- Turbine inlet temperature (TIT) C 1000.85 1100.89 1081.43
- Turbine after temperature (TAT) C 511.04 550.61 546.16
- Compressor discharge pressure (CDP) mbar 9.85 15.16 12.06
- Turbine energy yield (TEY) MWH 100.02 179.50 133.51
- Carbon monoxide (CO) mg/m3 0.00 44.10 2.37
- Nitrogen oxides (NOx) mg/m3 25.90 119.91 65.29

Importing libraries and dataset

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt
```

In [2]:

```
df1=pd.read_csv('gt_2011.csv')
df2=pd.read_csv('gt_2012.csv')
df3=pd.read_csv('gt_2013.csv')
df4=pd.read_csv('gt_2014.csv')
df5=pd.read_csv('gt_2015.csv')
```

In [3]:

```
df1.head()
```

Out[3]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	CO	NOX
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	134.67	11.898	0.32663	81.952
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	134.67	11.892	0.44784	82.377
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19	135.10	12.042	0.45144	83.776
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	135.03	11.990	0.23107	82.505
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00	134.67	11.910	0.26747	82.028

In [4]:

```
df1.shape
```

Out[4]:

```
(7411, 11)
```

We have 5 datasets in total each containing around 7 thousand instances

- Dates are not given in the data but the data is in chronological order.
- Here, the data is not temporal and neither it is supposed to be since gas emissions from a turbine is not a subject to time if other factors such as proper maintenance and calibrations are assumed to be conducted according to standard procedures.
- We will use first 3 datasets for training purpose and other 2 for testing purpose so as to follow protocol mentioned in the paper.

In [5]:

```
df1=df1.append(df2, ignore_index=True)
dftrain=df1.append(df3, ignore_index=True)
df1.head()
```

Out[5]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	CO	NOX
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	134.67	11.898	0.32663	81.952
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	134.67	11.892	0.44784	82.377
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19	135.10	12.042	0.45144	83.776
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	135.03	11.990	0.23107	82.505
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00	134.67	11.910	0.26747	82.028

In [6]:

```
dftrain.shape
```

Out[6]:

```
(22191, 11)
```

In [7]:

```
dftest=df4.append(df5, ignore_index=True)
dftest.head()
```

Out[7]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	CO	NOX
0	8.8358	1022.2	98.762	3.2422	23.152	1067.5	549.94	126.16	11.381	1.9157	68.292
1	9.0529	1021.8	97.464	3.2074	23.207	1067.9	549.80	126.51	11.476	2.0596	68.610
2	9.2464	1021.3	96.494	3.2051	23.296	1068.5	549.89	126.96	11.555	2.1621	68.324
3	9.3792	1020.9	95.912	3.2159	23.465	1069.6	549.94	127.83	11.539	2.1214	67.619
4	9.5757	1020.8	95.412	3.1898	23.205	1068.1	549.83	126.56	11.484	2.1549	66.499

Exploratory data analysis

In [8]:

```
dftest.shape
```

Out[8]:

```
(14542, 11)
```

Thus we have 60 % of total data as train data and 40 % as test data.

In [9]:

```
dftrain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22191 entries, 0 to 22190
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   AT      22191 non-null  float64
 1   AP      22191 non-null  float64
 2   AH      22191 non-null  float64
 3   AFDP    22191 non-null  float64
 4   GTEP    22191 non-null  float64
 5   TIT     22191 non-null  float64
 6   TAT     22191 non-null  float64
 7   TEY     22191 non-null  float64
 8   CDP     22191 non-null  float64
 9   CO      22191 non-null  float64
 10  NOX     22191 non-null  float64
dtypes: float64(11)
memory usage: 1.9 MB
```

In [10]:

```
dftrain.dtypes
```

Out[10]:

```
AT      float64
AP      float64
AH      float64
AFDP    float64
GTEP    float64
TIT     float64
TAT     float64
TEY     float64
CDP     float64
CO      float64
NOX     float64
dtype: object
```

In [11]:

```
l1=['Ambient temp','Ambient pressure','Ambient humidity','AFD pressure','GT exh pressure','
    'Turbine after temp','CD pressure',
    'Turbine energy yield','CO','NOX']
```

In [12]:

```
#renaming the columns
dftrain.columns=l1
dftrain.head(10)
```

Out[12]:

	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp	CD pressure	Turbine energy yield	C
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	134.67	11.898	0.326
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	134.67	11.892	0.447
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19	135.10	12.042	0.451
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	135.03	11.990	0.231
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00	134.67	11.910	0.267
5	3.8858	1017.7	83.946	3.5824	23.903	1086.0	549.98	134.67	11.868	0.234
6	3.6697	1018.0	84.114	3.5804	23.889	1085.9	550.04	134.68	11.877	0.444
7	3.5892	1018.2	83.867	3.5777	23.876	1086.0	549.88	134.66	11.893	0.799
8	3.7108	1018.5	84.948	3.6027	23.957	1086.3	549.98	134.65	11.870	0.689
9	4.8281	1018.5	85.346	3.5158	23.422	1083.1	549.80	132.67	11.694	1.028

In [13]:

```
dftest.columns=l1
dftest.head()
```

Out[13]:

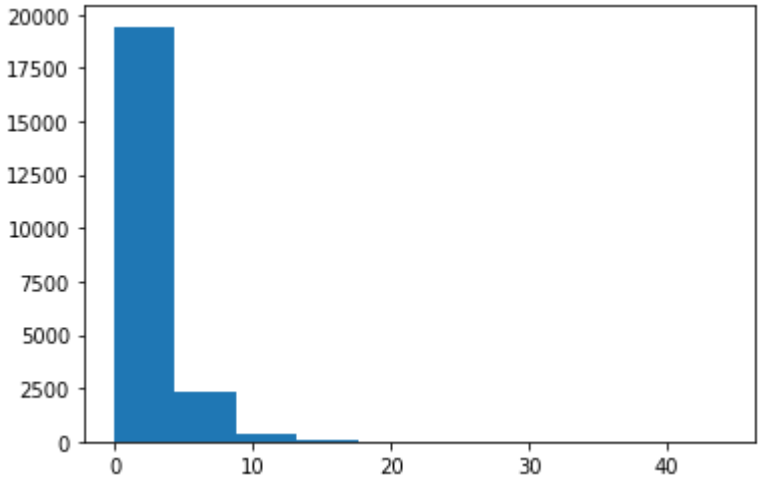
	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp	CD pressure	Turbine energy yield	
0	8.8358	1022.2	98.762	3.2422	23.152	1067.5	549.94	126.16	11.381	1.9
1	9.0529	1021.8	97.464	3.2074	23.207	1067.9	549.80	126.51	11.476	2.0
2	9.2464	1021.3	96.494	3.2051	23.296	1068.5	549.89	126.96	11.555	2.1
3	9.3792	1020.9	95.912	3.2159	23.465	1069.6	549.94	127.83	11.539	2.1
4	9.5757	1020.8	95.412	3.1898	23.205	1068.1	549.83	126.56	11.484	2.1

In [14]:

```
plt.hist(dftrain['CO'])
```

Out[14]:

```
(array([1.9435e+04, 2.3310e+03, 3.5800e+02, 3.4000e+01, 6.0000e+00,
        2.0000e+00, 8.0000e+00, 1.1000e+01, 2.0000e+00, 4.0000e+00]),
 array([3.87510000e-04, 4.41064876e+00, 8.82091001e+00, 1.32311713e+01,
        1.76414325e+01, 2.20516938e+01, 2.64619550e+01, 3.08722163e+01,
        3.52824775e+01, 3.96927388e+01, 4.41030000e+01]),
 <a list of 10 Patch objects>)
```

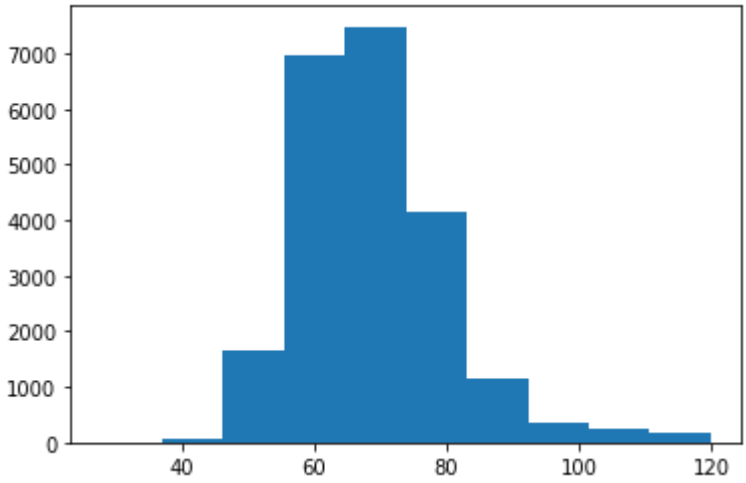


In [15]:

```
plt.hist(dftrain['NOX'])
```

Out[15]:

```
(array([1.000e+00, 5.600e+01, 1.639e+03, 6.981e+03, 7.482e+03, 4.130e+03,
        1.144e+03, 3.610e+02, 2.340e+02, 1.630e+02]),
 array([ 27.765 ,  36.9795,  46.194 ,  55.4085,  64.623 ,  73.8375,
         83.052 ,  92.2665, 101.481 , 110.6955, 119.91  ]),
 <a list of 10 Patch objects>)
```

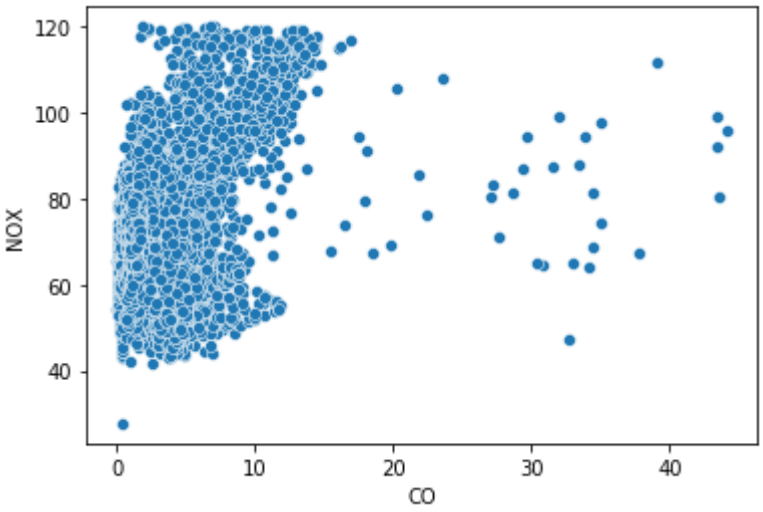


In [16]:

```
sns.scatterplot(data=dftrain,x='CO',y='NOX')
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0da38dc88>
```



Here it seems like there is no direct relationship between emissions of NOX and CO.

3D plot - CO , NOX and 'Turbine energy yield'

In [17]:

```
dffinal=dftrain.append(dftest,ignore_index=True)
```

In [18]:

```
dffinal.head()
```

Out[18]:

	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp	CD pressure	Turbine energy yield	C
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	134.67	11.898	0.326
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	134.67	11.892	0.447
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19	135.10	12.042	0.451
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	135.03	11.990	0.231
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00	134.67	11.910	0.267

In [19]:

```
import plotly.express as px
```

In [20]:

```
fig3d=px.scatter_3d(dffinal,x='CO',y='NOX',z='Turbine energy yield',size_max=2,opacity=0.7)
fig3d.show()
```

- From above 3d plot we can see that the datapoints need to be clustered since we can see there is one single big arbitrary shaped cluster and the points outside of it are to be put in a different cluster.
- Density based clustering algorithms might work here.

Sampling of data

Here we will do Random sampling so the sample data obtained is the sample representation of whole of the data Also by doing random sampling, the resulting sample we get will be free of any pattern in the dataset

In [21]:

```
dffinal.shape
```

Out[21]:

```
(36733, 11)
```

In [22]:

```
#sampling of data
from random import sample
list1=list(dffinal.index)
indexes=sample(list1,4000)
def rsampling(df):
    rsample=df.iloc[indexes]
    return rsample
```

In [23]:

```
sampled_df=rsampling(dffinal)
sampled_df.head()
```

Out[23]:

	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp	CD pressure	Turbine energy yield	
34368	22.7010	1010.1	71.326	3.7688	24.920	1080.3	550.02	131.26	11.956	2
15333	5.7108	1024.5	92.480	4.1969	34.919	1099.9	521.49	168.54	14.465	1
21826	17.4130	1003.7	90.046	3.2632	24.263	1082.3	550.08	129.72	11.717	1
18177	19.2290	1011.3	89.690	2.8226	19.062	1052.4	549.98	106.63	10.355	6
23208	15.6130	1005.8	80.452	3.5785	25.409	1077.9	550.00	133.57	11.955	0

In [24]:

```
sampled_df.shape
```

Out[24]:

```
(4000, 11)
```

Data visualizations

In [25]:

```
from matplotlib import pyplot
```

In [26]:

```
sns.set(rc={'figure.figsize':(15,10)})
sns.set(font_scale=1.5)
```

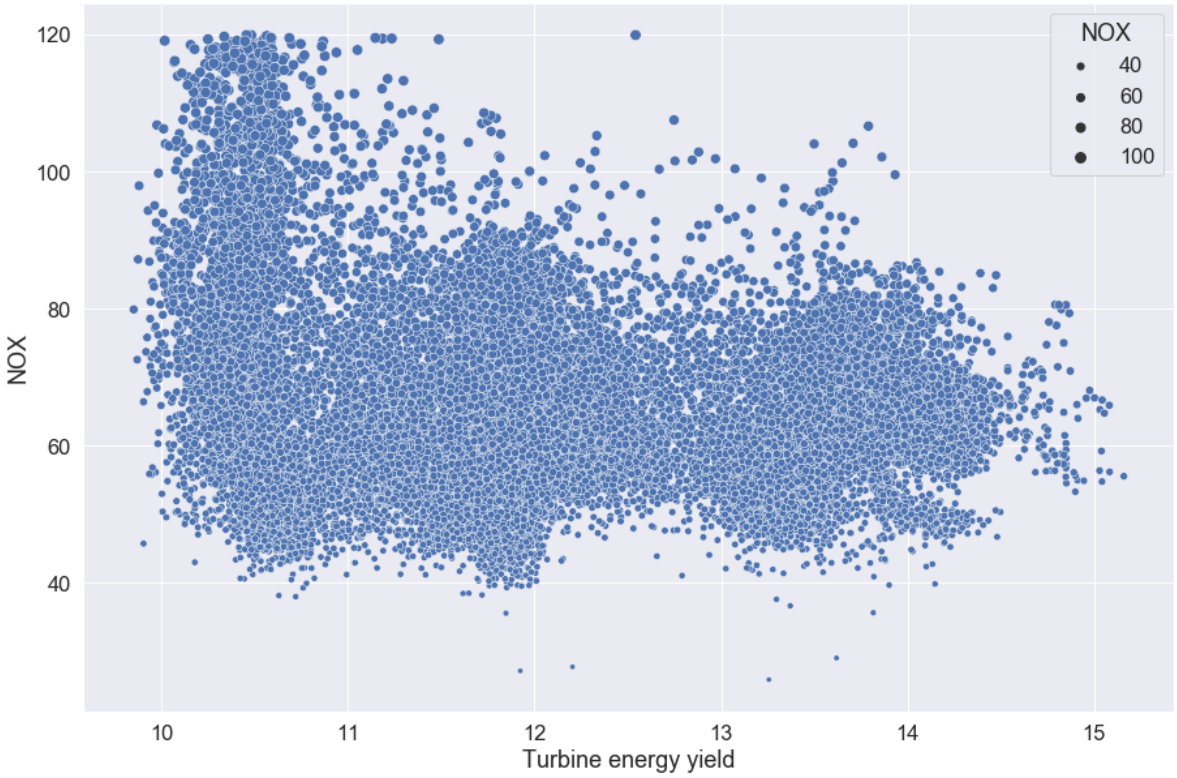
We use whole dataset so as to get better insights

In [27]:

```
sns.scatterplot(data=dffinal,x='Turbine energy yield',y='NOX',size='NOX')
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0d7d4da88>
```

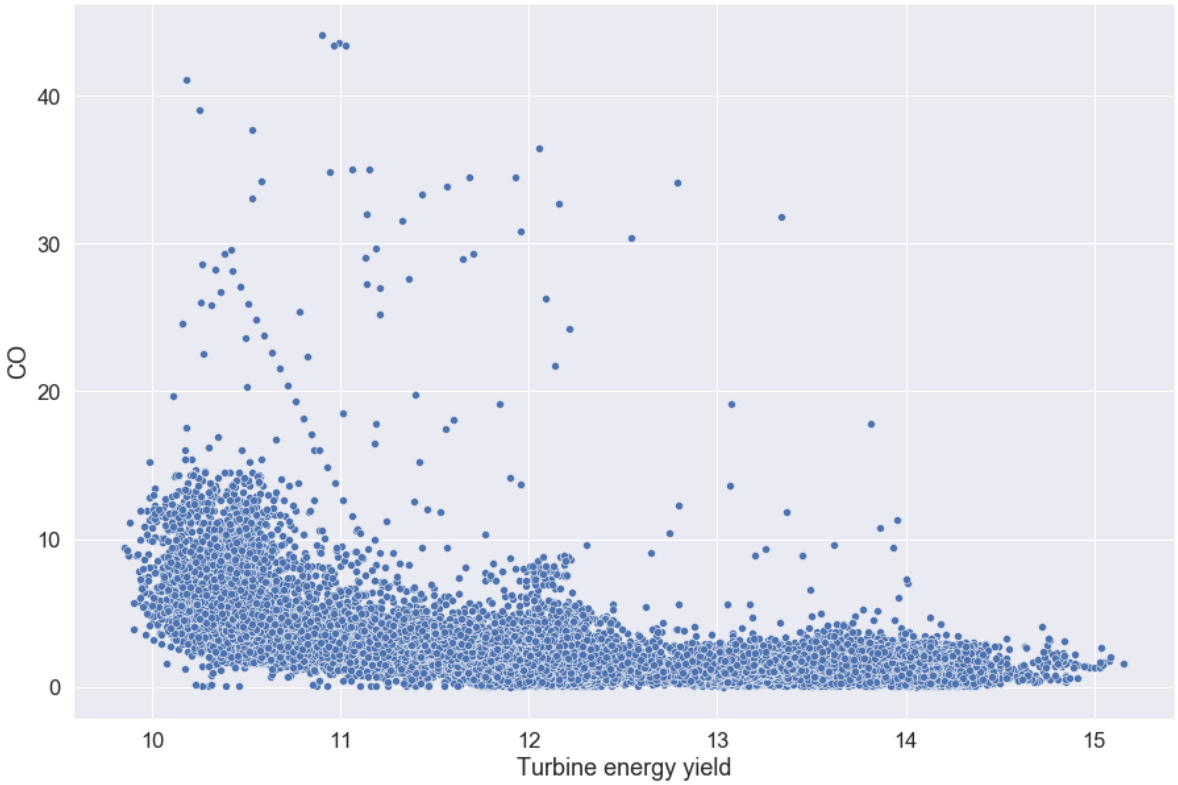


In [28]:

```
sns.scatterplot(data=dffinal,x='Turbine energy yield',y='CO')
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0df922f48>
```



- There is a very good insight from above plot, that with increase in Turbine energy yield ,the CO in the gas emissions reduces
- This means that when there is complete combustion in gas turbine , the energy yield is high and the greenhouse gases (CO) will be become negligible , thus using the basic definition of combustion, we can progress towards our goal

- The general equation for a complete combustion reaction is: Fuel + O2 → CO2 + H2O, but in case of incomplete combustion equation the release of CO and NO increases.
- Also there are many seemingly outliers when the turbine energy yield is low (incomplete combustion), these outliers are the instances which are supposed to trigger the maintenance of gas turbine.

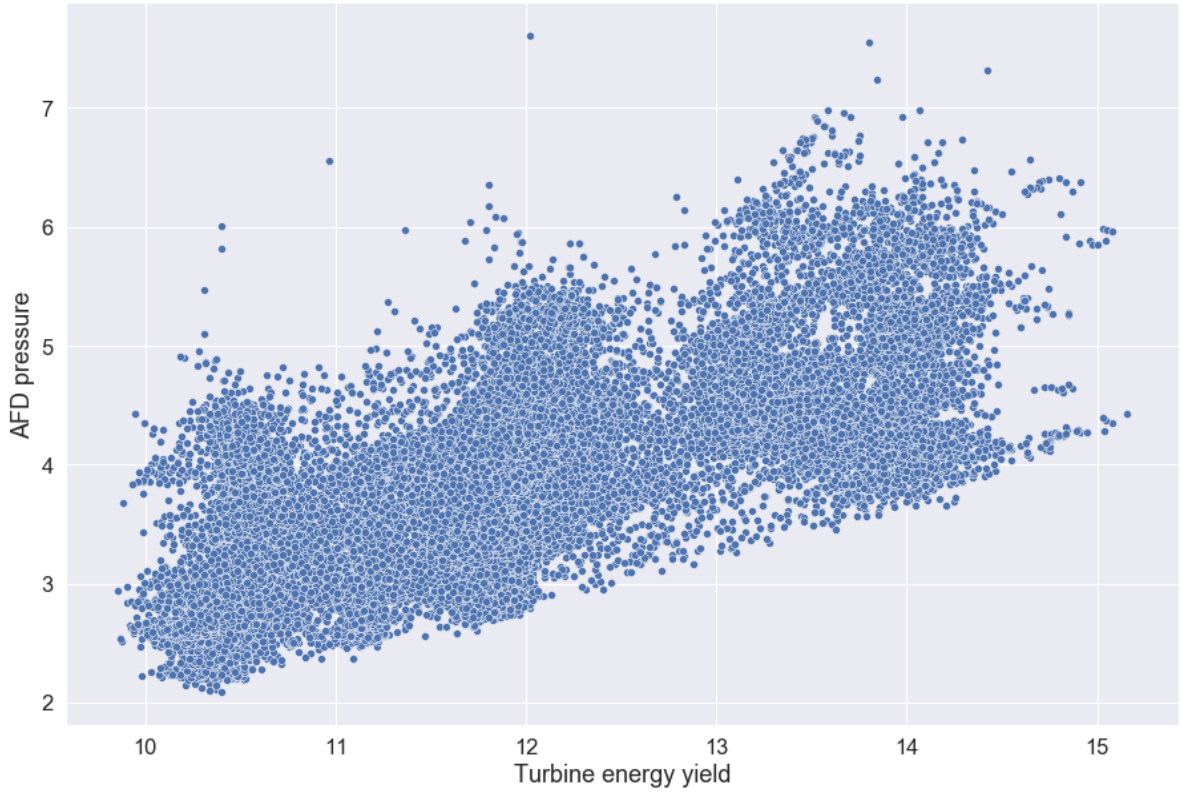
- **Plotting Turbine energy yield and other attributes.**

In [29]:

```
sns.scatterplot(data=dffinal,x='Turbine energy yield',y='AFD pressure')
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0e019b488>
```

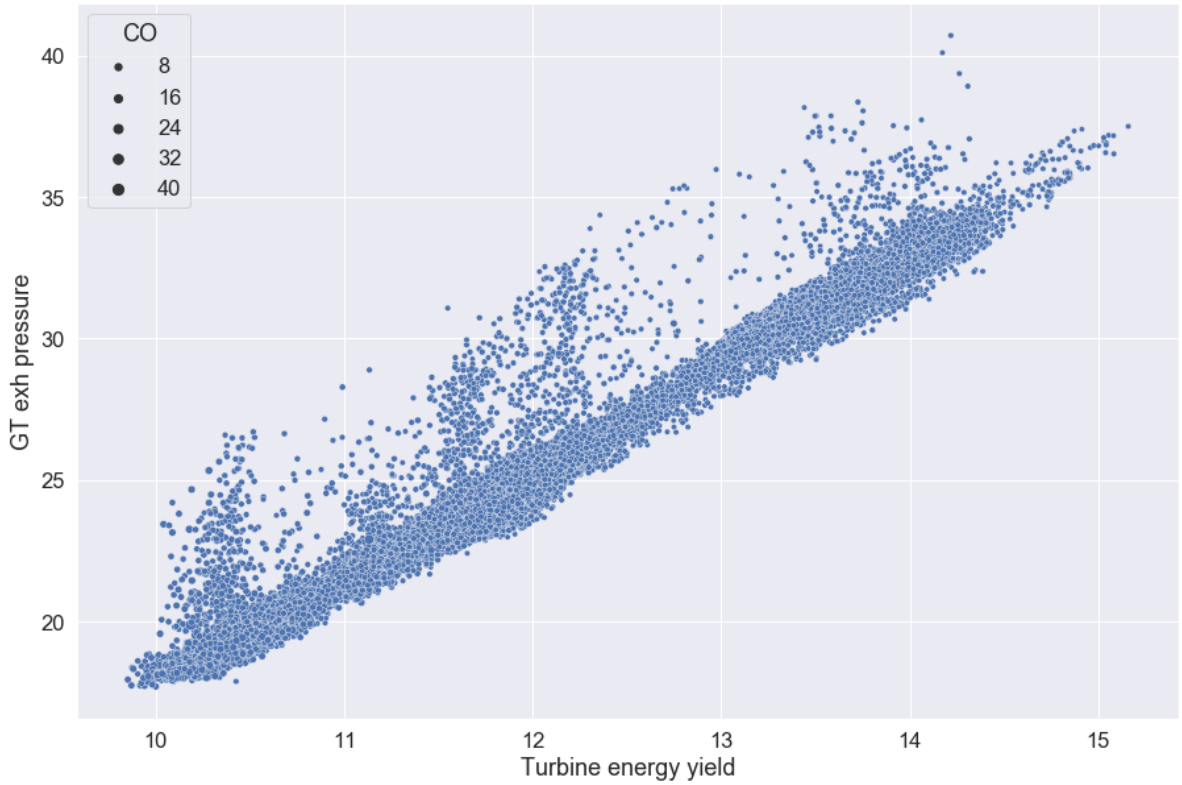


In [30]:

```
sns.scatterplot(data=dffinal,x='Turbine energy yield',y='GT exh pressure',size='CO')
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0df99eec8>
```

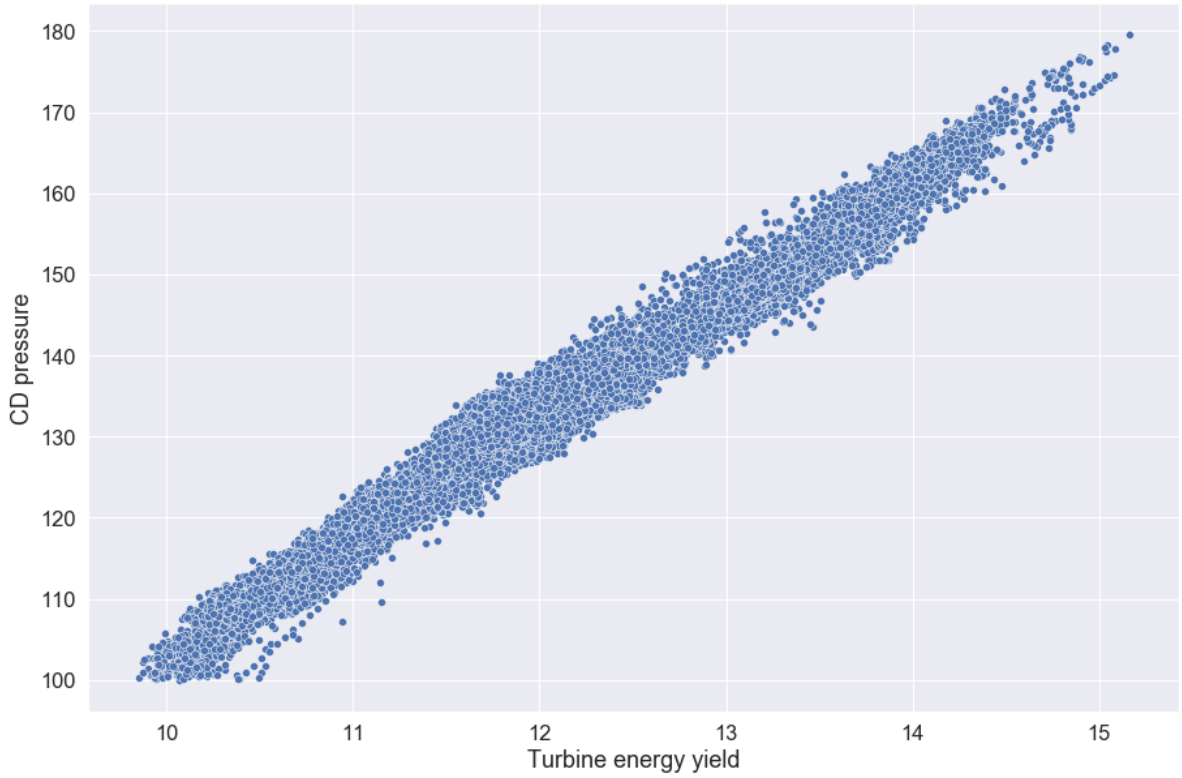


In [31]:

```
sns.scatterplot(data=dffinal,x='Turbine energy yield',y='CD pressure')
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0dfdc5088>
```



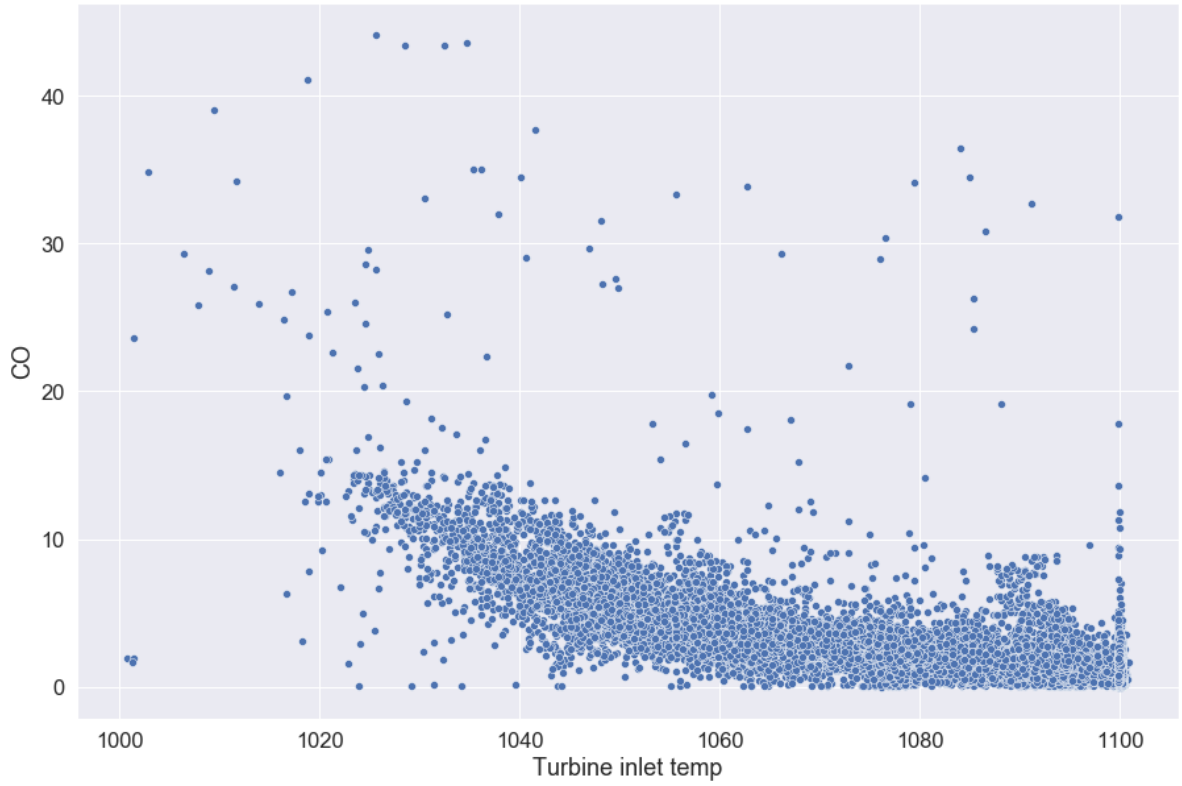
- **With increasing AFD pressure,GT exhaust pressure and CD pressure, the turbine energy yield also increases linearly, thought the increase is less steep in case of AFD pressure.**
- **Seems there is high correlation among turbine enrgy yield , CD pressure and GT exhaust pressure**

In [32]:

```
sns.scatterplot(data=dffinal,x='Turbine inlet temp',y='CO')
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0df9e5288>
```



In [33]:

```
dffinal.corr()
```

Out[33]:

	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp	CD pressure
Ambient temp	1.000000	-0.406601	-0.476291	0.251974	0.045851	0.183706	0.281869	-0.091152
Ambient pressure	-0.406601	1.000000	-0.015184	-0.040363	0.057533	-0.005390	-0.225601	0.118224
Ambient humidity	-0.476291	-0.015184	1.000000	-0.147840	-0.235153	-0.221809	0.022965	-0.137360
AFD pressure	0.251974	-0.040363	-0.147840	1.000000	0.678485	0.691292	-0.466882	0.665483
GT exh pressure	0.045851	0.057533	-0.235153	0.678485	1.000000	0.874234	-0.699703	0.964127
Turbine inlet temp	0.183706	-0.005390	-0.221809	0.691292	0.874234	1.000000	-0.380862	0.910297
Turbine after temp	0.281869	-0.225601	0.022965	-0.466882	-0.699703	-0.380862	1.000000	-0.682396
CD pressure	-0.091152	0.118224	-0.137360	0.665483	0.964127	0.910297	-0.682396	1.000000
Turbine energy yield	0.015287	0.102636	-0.196275	0.702568	0.978470	0.908469	-0.706438	0.988778
CO	-0.174326	0.067050	0.106586	-0.448425	-0.518909	-0.706275	0.058353	-0.569813
NOX	-0.558174	0.191938	0.164617	-0.188247	-0.201630	-0.213865	-0.092791	-0.116127

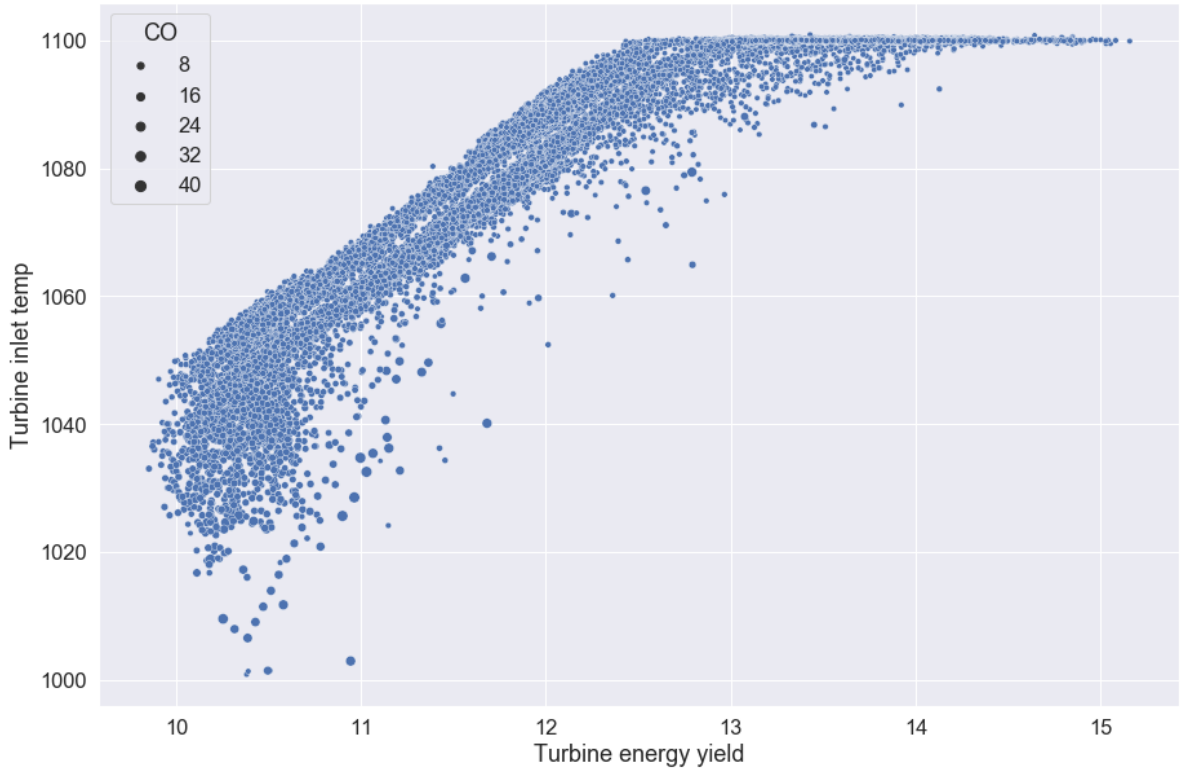
As evident from above table ,CD pressure is highly correlated with GT exh pressure and thus there is presence of Muliticollinearity , a problem since the variables are considered to independent of each other.

In [34]:

```
sns.scatterplot(data=dffinal,x='Turbine energy yield',y='Turbine inlet temp',size='CO')
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0e01319c8>
```



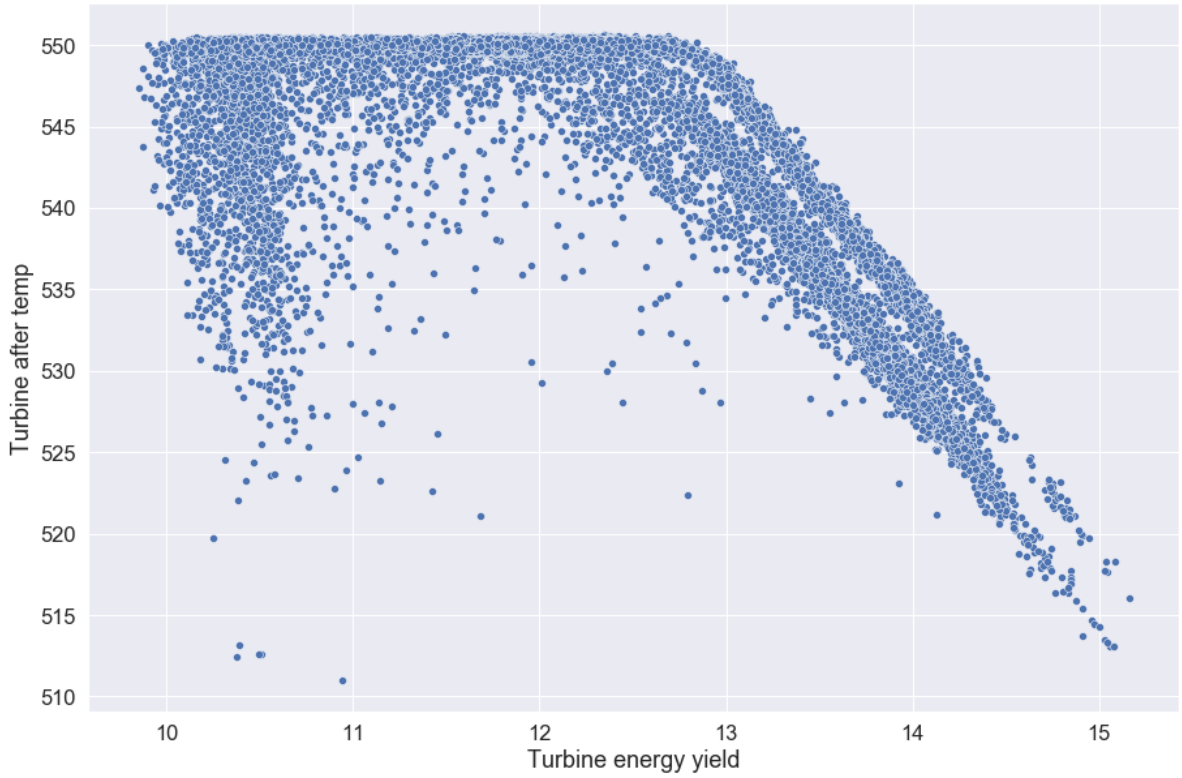
turbine inlet temp increases linearly with increases in TEY, upto a certain point (1100 C) and then the saturates at 1100 C , specifying the instances with low CO and NOX emissions (due to complete combustion).

In [35]:

```
sns.scatterplot(data=dffinal,x='Turbine energy yield',y='Turbine after temp')
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0dfa792c8>
```



Since the data we have contains approx 37k instances and around 7 predictor features,there is a high chance that the relationship between the predictor and target feature must be non linear since we have already visualized that the relationship of features with TEY non-linear, thus we will build and compare the neural network and the multiple regression models and compare them

Modelling of data

1.1 Deep Neural network - regression

Why are we using Neural network here ?

- Neural networks can learn non linearities in data much better compared to regular machine learning algorithms.

- A neural network algorithm can be interpreted as a bunch of linear regressions, where each node is an output of one linear regression.
- **above two reasons make neural networks much better choice when compared to conventional machine learning algos**

PREDICTING THE TURBINE ENERGY YIELD

- We'll drop 'CD pressure' to avoid multicollinearity
- CO and NOX will also be dropped since they are not predictor features here

In [36]:

```
df2=dffinal.drop(columns=['CD pressure','CO','NOX'])
```

In [37]:

```
df2.head()
```

Out[37]:

	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp	Turbine energy yield
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	11.898
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	11.892
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19	12.042
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	11.990
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00	11.910

In [38]:

```
df2.shape
```

Out[38]:

```
(36733, 8)
```

In [39]:

```
x=df2.drop(columns=['Turbine energy yield'])
```

In [*]:

```
x.shape
```

In [*]:

```
y=df2['Turbine energy yield']
```

In [*]:

```
y.shape
```

using library

In [43]:

```
from keras.models import Sequential
from keras.layers import Dense,Dropout
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

In [64]:

```
scaler=StandardScaler()
```

In [44]:

```
df2.head()
```

Out[44]:

	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp	Turbine energy yield
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	11.898
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	11.892
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19	12.042
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	11.990
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00	11.910

In [45]:

```
x.head()
```

Out[45]:

	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00

In [46]:

```
y.head()
```

Out[46]:

```
0    11.898
1    11.892
2    12.042
3    11.990
4    11.910
Name: Turbine energy yield, dtype: float64
```

In [47]:

```
dftemp=dffinal.drop(columns=['CD pressure'])
```

In [48]:

```
xn=dftemp.drop(columns=['Turbine energy yield','CO','NOX'])
```

In [49]:

```
yn=dftemp[['Turbine energy yield','CO','NOX']]
```

In [50]:

```
xn.head()
```

Out[50]:

	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00

In [51]:

```
yn.head()
```

Out[51]:

	Turbine energy yield	CO	NOX
0	11.898	0.32663	81.952
1	11.892	0.44784	82.377
2	12.042	0.45144	83.776
3	11.990	0.23107	82.505
4	11.910	0.26747	82.028

In [52]:

```
def baseline_model():
    model=Sequential()
    model.add(Dense(7,input_dim=7,kernel_initializer='normal',activation='relu'))
    model.add(Dense(3,kernel_initializer='normal'))
    model.compile(loss='mean_squared_error',optimizer='adam')
    return model
```

In [53]:

```
estimators=[]
estimators.append(('std',StandardScaler()))
estimators.append(('mlp',KerasRegressor(build_fn=baseline_model,epochs=1000,batch_size=5000
pipeline=Pipeline(estimators)
```

In [54]:

```
kfold = KFold(n_splits=10) # data is divided into 10 folds
results = cross_val_score(pipeline, xn, yn, cv=kfold)
print("Results: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
7/7 [==============================] - 0s 2ms/step - loss: 17.2466
Epoch 993/1000
7/7 [==============================] - 0s 3ms/step - loss: 17.2344
Epoch 994/1000
7/7 [==============================] - 0s 4ms/step - loss: 17.2226
Epoch 995/1000
7/7 [==============================] - 0s 3ms/step - loss: 17.2113
Epoch 996/1000
7/7 [==============================] - 0s 5ms/step - loss: 17.1995
Epoch 997/1000
7/7 [==============================] - 0s 2ms/step - loss: 17.1878
Epoch 998/1000
7/7 [==============================] - 0s 2ms/step - loss: 17.1762
Epoch 999/1000
7/7 [==============================] - 0s 2ms/step - loss: 17.1651
Epoch 1000/1000
7/7 [==============================] - 0s 2ms/step - loss: 17.1536
1/1 [==============================] - 0s 997us/step - loss: 41.6317
Results: -25.34 (10.52) MSE
```

- **RMSE is the standard deviation of prediction errors. Prediction errors are a measure of how far from the regression line the datapoints are.**
- Low RMSE is desirable

- we now have the estimate of models performance on problem of unseen data since we have used cross validation score
- and here we have Mean squared error ~ 11
- RMSE ~ 3.3
- we have low rmse , thus specifying that the neural networks were able to predict the multi output targets.

1.2 Multi output regressor using sklearn library

- **multi output regression is different from regression since in multi output regression, the target variables are dependent of predictor variables as well as dependent on each other**
- **so to exploit the correlation among the target variables we will use chained regressor model which will basically chain model in this way:**
  - given X (predictor features), predict y1(first target feature)
  - given X and y1, predict y2(second target feature)
  - given X,y1 and y2, predict y3(third target feature)
- **Thus by using chained regressor, we might achieve better results compared to neural network since the in neural network, the correlation among the target variables is not exploited**

In [55]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.multioutput import RegressorChain
```

In [56]:

```
reg_ob=LinearRegression(normalize=True)
```

In [57]:

```
chain=RegressorChain(reg_ob,order=[0,2,1])
```

In [58]:

```
chain.fit(xn,yn)
```

Out[58]:

```
RegressorChain(base_estimator=LinearRegression(normalize=True), order=[0, 2,
1])
```

In [59]:

```
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

In [60]:

```
results2 = cross_val_score(chain, xn, yn, cv=kfold)
print("Results: %.2f (%.2f) MSE" % (results2.mean(), results2.std()))
```

```
Results: 0.48 (0.13) MSE
```

We have the mse=0.4 which tells us that our model fits very well and will give us root mean squared error ~ 0.62

Thus we can see that the chained regressor gives us much better fit when we need to predict multiple target features

2.1 Clustering - Density based clustering

DBSCAN - density based spatial clustering of application with noise

In [61]:

```
dffinal.head()
```

Out[61]:

	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp	CD pressure	Turbine energy yield	C
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	134.67	11.898	0.326
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	134.67	11.892	0.447
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19	135.10	12.042	0.451
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	135.03	11.990	0.231
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00	134.67	11.910	0.267

- Since turbine energy yield is dependent highly on ambient features , so we will ignore the ambient features while clustering the data.
- Turbine energy yield attribute has already proved its importance when we need to know whether datapoint is dangereous or not (see visualizations)
- Also CO and NOX features if chosen alone do not give us seemingly valid clusters. (**by comparing 2d scatter plot of CO vs NOX and 3d scatter plot of CO vs NOX vs TEY**)
- **thus we will cluster datapoints based on TEY, CO and NOX**

In [62]:

```
df3=dffinal[['Turbine energy yield','CO','NOX']]
df3.head()
```

Out[62]:

	Turbine energy yield	CO	NOX
0	11.898	0.32663	81.952
1	11.892	0.44784	82.377
2	12.042	0.45144	83.776
3	11.990	0.23107	82.505
4	11.910	0.26747	82.028

We need to preprocess the data by standardizing the dataframe

In [65]:

```
df3_std=scaler.fit_transform(df3)
```

In [66]:

```
df3_std
```

Out[66]:

```
array([[-0.14927265, -0.90418162,  1.42649853],
       [-0.1547834 , -0.85061147,  1.46289113],
       [-0.01701458, -0.84902041,  1.58268701],
       ...,
       [-1.46817947,  3.99600838,  2.64201137],
       [-0.26591691,  0.41585555, -0.04753021],
       [-0.54972068,  4.24660051,  3.76316034]])
```

We will try to use Density based Clustering algorithm - DBSCAN here because :

- Data is clearly not linearly seperable (see 3d plot of CO vs NOX vs TEY).
- It is not vulnerable to outliers rather it handles them very well , either by declaring them as noise or clustering them togehter.
- It handles the noise datapoints effectively
- It forms clusters based on the density of datapoints
- Here above points are highly desirable in our case.

In [67]:

```
from sklearn.cluster import DBSCAN
```

In [68]:

```
db=DBSCAN(eps=0.54,min_samples=11).fit(df3_std)
core_samples_mask=np.zeros_like(db.labels_,dtype=bool)
core_samples_mask[db.core_sample_indices_]=True
labels=db.labels_
```

- epsilon is the maximum distance between the two points in a cluster
- min samples is the minimum number of the points which have to be inside the epsilon distance so as to be defined in a cluster

number of clusters we need =2 , one cluster should be the one with 'Not danger' values (High tey, Low CO and Low nox) and other cluser should consist of 'Danger' values (low tey, high CO and high NOX)

In [69]:

```
num_clusters=len(set(labels)) - (1 if -1 in labels else 0)
num_clusters
```

Out[69]:

```
2
```

In [70]:

```
print(labels)
```

```
[ 0  0  0 ...  0  0 -1]
```

We have chosen values epsilon and min_samples by heuristic approaches and techniques since we knew that no of clusters should be 2

- *Heuristically value of min_samples is taken as ln(n) where n= number of total datapoints to be clustered (used in Section 4.1 of [https://www.sciencedirect.com/science/article/pii/S0169023X06000218 (https://www.sciencedirect.com/science/article/pii/S0169023X06000218)](https://www.sciencedirect.com/science/article/pii/S0169023X06000218) – Shawn TIAN)*
- *so we have ln(36733)~10.5 , so we will assume min_samples either 10 or 11*
- *We choose the value of epsilon by hit and trial and by exploiting the fact that we need 2 clusters and the value of min_samples is already known to us._*

In [71]:

```
num_noise=list(labels).count(-1)
num_noise
```

Out[71]:

149

Lets visualize the clusters in order to see whether the clusters seem logical....

In [72]:

```
dffinal['Target']=labels
dffinal.head()
```

Out[72]:

	Ambient temp	Ambient pressure	Ambient humidity	AFD pressure	GT exh pressure	Turbine inlet temp	Turbine after temp	CD pressure	Turbine energy yield	C
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	134.67	11.898	0.326
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	134.67	11.892	0.447
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19	135.10	12.042	0.451
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	135.03	11.990	0.231
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00	134.67	11.910	0.267

In [73]:

```
fig3d=px.scatter_3d(dffinal,x='NOX',y='CO',z='Turbine energy yield',color='Target',opacity=
fig3d.show()
```

- By visual inspection , we can say that the goal we had in mind earlier (to cluster the datapoints into 2 clusters) is well completed.
- We will now evaluate the clustering results using some techniques and also we will compare the results with the other clustering approach results.

- Since the true cluster labels are unknown, the model itself will be used to evaluate performance.
- Silhouette coefficient (SC) will be calculated and we evaluate using it as:
  - SC is between -1 and 1
  - if SC=1 , we have the perfectly defined dense clusters
  - if SC=0, the datapoints are overlapping
  - if SC=-1 , the datapoints are assigned to wrong cluster

In [75]:

```
df3['Target']=labels
df3.head()
```

```
C:\Users\Anshul\AppData\Roaming\Python\Python37\site-packages\ipykernel_laun
cher.py:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
```

Out[75]:

	Turbine energy yield	CO	NOX	Target
0	11.898	0.32663	81.952	0
1	11.892	0.44784	82.377	0
2	12.042	0.45144	83.776	0
3	11.990	0.23107	82.505	0
4	11.910	0.26747	82.028	0

In [76]:

```
from sklearn import metrics
metrics.silhouette_score(df3, df3['Target'])
```

Out[76]:

```
0.5608920444000304
```

We have got a decent silhouette score ,indicating that our DBSCAN model does not have overlapping clusters or mislabelled datapoints

2.2 Clustering using Self Organizing maps (SOM)

- SOMs differ from other artificial neural networks because they apply competitive learning as opposed to error correlated learning, which involves backpropagation and gradient descent.
- In competitive learning, nodes compete for the right to respond to the input data subset.
- The training data usually has no labels and the map learns to differentiate and distinguish features based on similarities.

In [84]:

```
df3=df3.drop(columns=['Target'])
df3.head()
```

Out[84]:

	Turbine energy yield	CO	NOX
0	11.898	0.32663	81.952
1	11.892	0.44784	82.377
2	12.042	0.45144	83.776
3	11.990	0.23107	82.505
4	11.910	0.26747	82.028

In [85]:

```
from sklearn.preprocessing import MinMaxScaler
mm=MinMaxScaler(feature_range=(0,1))     #min mix scaler will transform the whole data in t
```

In [86]:

```
df3_mm=mm.fit_transform(df3)
df3_mm
```

Out[86]:

```
array([[0.3855517 , 0.00739735, 0.59621297],
       [0.38442116, 0.01014571, 0.600734  ],
       [0.41268465, 0.01022734, 0.61561619],
       ...,
       [0.11497588, 0.25879674, 0.74721557],
       [0.36162195, 0.07512055, 0.41309505],
       [0.30339916, 0.27165312, 0.8864954 ]])
```

In [87]:

```
df3_ss=scaler.fit_transform(df3)
df3_ss
```

Out[87]:

```
array([[-0.14927265, -0.90418162,  1.42649853],
       [-0.1547834 , -0.85061147,  1.46289113],
       [-0.01701458, -0.84902041,  1.58268701],
       ...,
       [-1.46817947,  3.99600838,  2.64201137],
       [-0.26591691,  0.41585555, -0.04753021],
       [-0.54972068,  4.24660051,  3.76316034]])
```

In [88]:

```
from minisom import MiniSom
import time
```

Thumb rule to set grid so as to get total nodes= sqrt(5*sqrt(N)) where N= num of datapoints

In [89]:

```
#setting of hyperparameters
som_grid_rows=30
som_grid_columns=30
iterations=2500 # iterations are generally taken as several thousand times the number of cl
sigma=0.5 # constant(default value)
lr=0.5 # constant(default value)
```

In [90]:

```
#initializing the som object
som_ob=MiniSom(x=som_grid_rows,
               y=som_grid_columns,
               input_len=3, #number of features in our input (3 in our case)
               sigma=sigma,
               learning_rate=lr,
               neighborhood_function='gaussian',
               random_seed=10)
som_ob.random_weights_init(df3_ss)  #assigning the random weights to neurons
```

In [91]:

```
#training process
start_time=time.time()
som_ob.train_batch(df3_ss,num_iteration=iterations)
train_time=time.time() - start_time
print('Time elapsed in the training process is :',train_time)
```

```
Time elapsed in the training process is : 0.5884253978729248
```

- Now we will consider all input mapped to a specific neuron as a single cluster.
- to identify each cluster more easily, we will translate the bi-dim indexes of the neurons into single dimension indexes

In [92]:

```
win_coords=np.array([som_ob.winner(x) for x in df3_ss]).T
```

In [93]:

```
cluster_index=np.ravel_multi_index(win_coords,(som_grid_rows,som_grid_columns))
```

In [94]:

```
cluster_index
```

Out[94]:

```
array([292, 293, 296, ..., 109, 810, 450], dtype=int64)
```

In [95]:

```
cluster_target=pd.DataFrame(cluster_index)
```

In [96]:

```
cluster_target.nunique()
```

Out[96]:

```
0    896
dtype: int64
```

In [97]:

```
df3['Target-som']=cluster_target
```

In [98]:

```
df3.head()
```

Out[98]:

	Turbine energy yield	CO	NOX	Target-som
0	11.898	0.32663	81.952	292
1	11.892	0.44784	82.377	293
2	12.042	0.45144	83.776	296
3	11.990	0.23107	82.505	292
4	11.910	0.26747	82.028	292

In [99]:

```
fig3d=px.scatter_3d(df3,x='NOX',y='CO',z='Turbine energy yield',color='Target-som',opacity=
fig3d.show()
```

As you can see from above 3d plot that self organizing maps do kind of soft clustering

- So basically we can cluster the above ~900 clusters into 2 clusters based upon the maximum values of CO and NOX and then classify whether a datapoint is benign or malignant

In [101]:

```
metrics.silhouette_score(df3, df3['Target-som'])
```

Out[101]:

```
0.6551812499384695
```

We can see that using the SOM , we get better defined clusters but the problem is that when we visually inspect SOM here don't seem to achieve our purpose very well because:

- SOM learn the similarities in input well but not the physical closeness in input data values.
- SOM are competitive neural networks and does not use the concept of backpropagation , thus the istribution of data will highly result of SOM.

In [ ]: