Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [2]:
```
df=pd.read_csv("madrid_2017.csv")
df
```

Out[2]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TC
0	2017-06-01 01:00:00	NaN	NaN	0.3	NaN	NaN	4.0	38.0	NaN	NaN	NaN	NaN	5.0	Na
1	2017-06-01 01:00:00	0.6	NaN	0.3	0.4	0.08	3.0	39.0	NaN	71.0	22.0	9.0	7.0	1.
2	2017-06-01 01:00:00	0.2	NaN	NaN	0.1	NaN	1.0	14.0	NaN	NaN	NaN	NaN	NaN	Na
3	2017-06-01 01:00:00	NaN	NaN	0.2	NaN	NaN	1.0	9.0	NaN	91.0	NaN	NaN	NaN	Na
4	2017-06-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	19.0	NaN	69.0	NaN	NaN	2.0	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
210115	2017-08-01 00:00:00	NaN	NaN	0.2	NaN	NaN	1.0	27.0	NaN	65.0	NaN	NaN	NaN	Na
210116	2017-08-01 00:00:00	NaN	NaN	0.2	NaN	NaN	1.0	14.0	NaN	NaN	73.0	NaN	7.0	Na
210117	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	4.0	NaN	83.0	NaN	NaN	NaN	Na
210118	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	11.0	NaN	78.0	NaN	NaN	NaN	Na
210119	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	14.0	NaN	77.0	60.0	NaN	NaN	Na

210120 rows × 16 columns

Data Cleaning and Data Preprocessing

In [3]:
```
df=df.dropna()
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4127 entries, 87457 to 158286
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4127 non-null   object 
 1   BEN      4127 non-null   float64
 2   CH4      4127 non-null   float64
 3   CO       4127 non-null   float64
 4   EBE      4127 non-null   float64
 5   NMHC     4127 non-null   float64
 6   NO       4127 non-null   float64
 7   NO_2     4127 non-null   float64
 8   NOx      4127 non-null   float64
 9   O_3      4127 non-null   float64
 10  PM10     4127 non-null   float64
 11  PM25     4127 non-null   float64
 12  SO_2     4127 non-null   float64
 13  TCH      4127 non-null   float64
 14  TOL      4127 non-null   float64
 15  station  4127 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 548.1+ KB
```

In [6]:
```
data=df[['CO' ,'station']]
data
```

Out[6]:

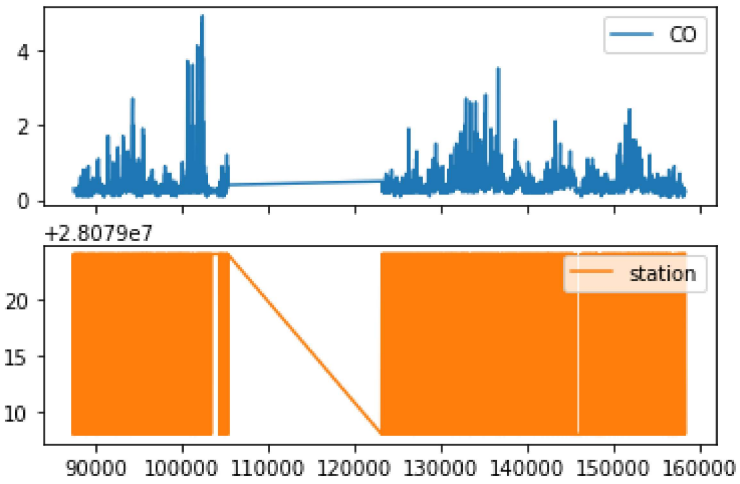
	CO	station
87457	0.3	28079008
87462	0.2	28079024
87481	0.2	28079008
87486	0.2	28079024
87505	0.2	28079008
...	...	...
158238	0.2	28079024
158257	0.3	28079008
158262	0.2	28079024
158281	0.2	28079008
158286	0.2	28079024

4127 rows × 2 columns

Line chart

In [7]:
```
data.plot.line(subplots=True)
```

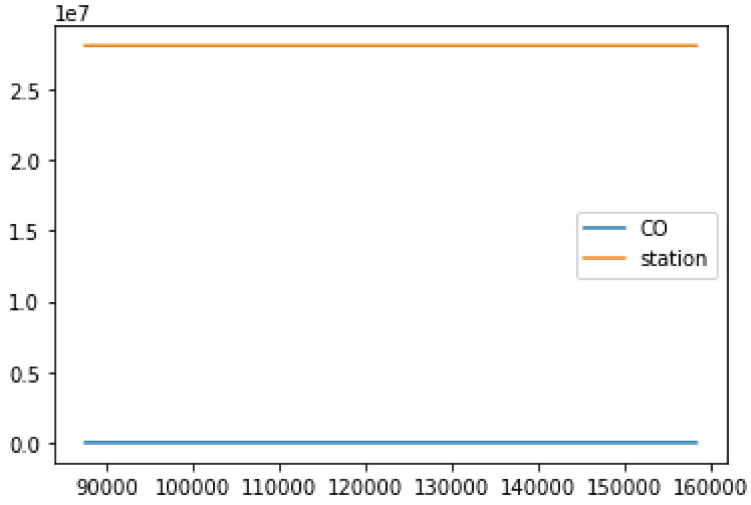
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [8]:
```
data.plot.line()
```

Out[8]: <AxesSubplot:>

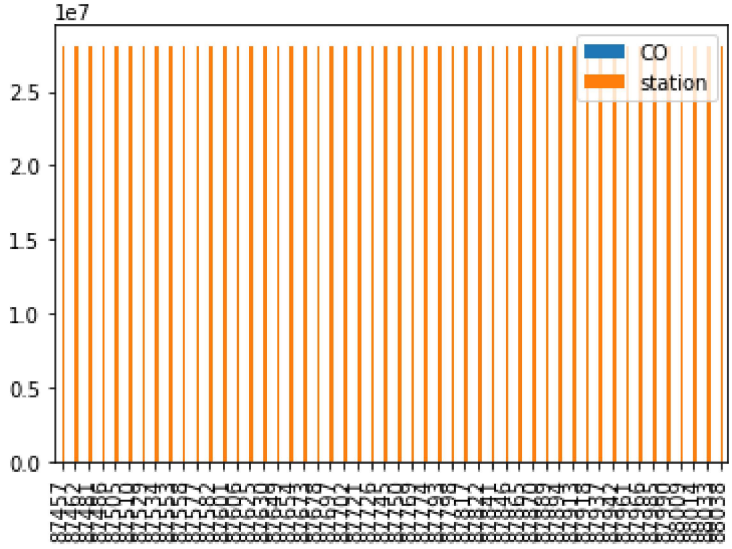


Bar chart

In [9]:
```
b=data[0:50]
```

In [10]:
```
b.plot.bar()
```

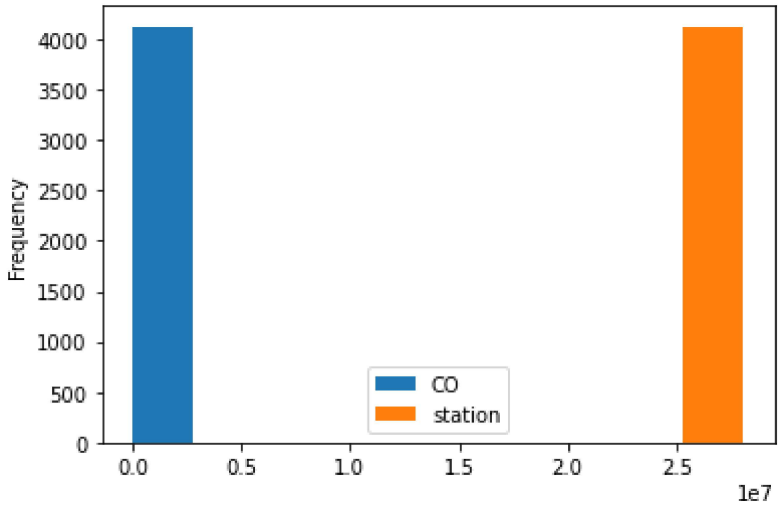
Out[10]: <AxesSubplot:>



Histogram

In [11]:
```
data.plot.hist()
```

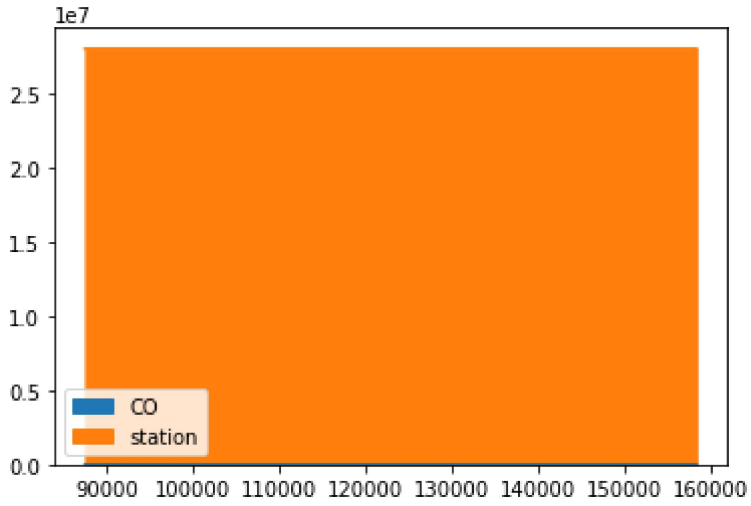
Out[11]:
```
<AxesSubplot:ylabel='Frequency'>
```



Area chart

In [12]:
```
data.plot.area()
```

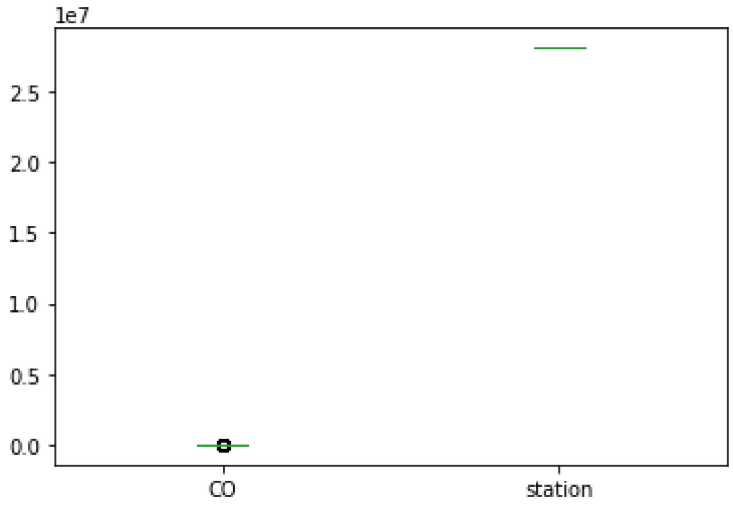
Out[12]:
```
<AxesSubplot:>
```



Box chart

In [13]:
```
data.plot.box()
```

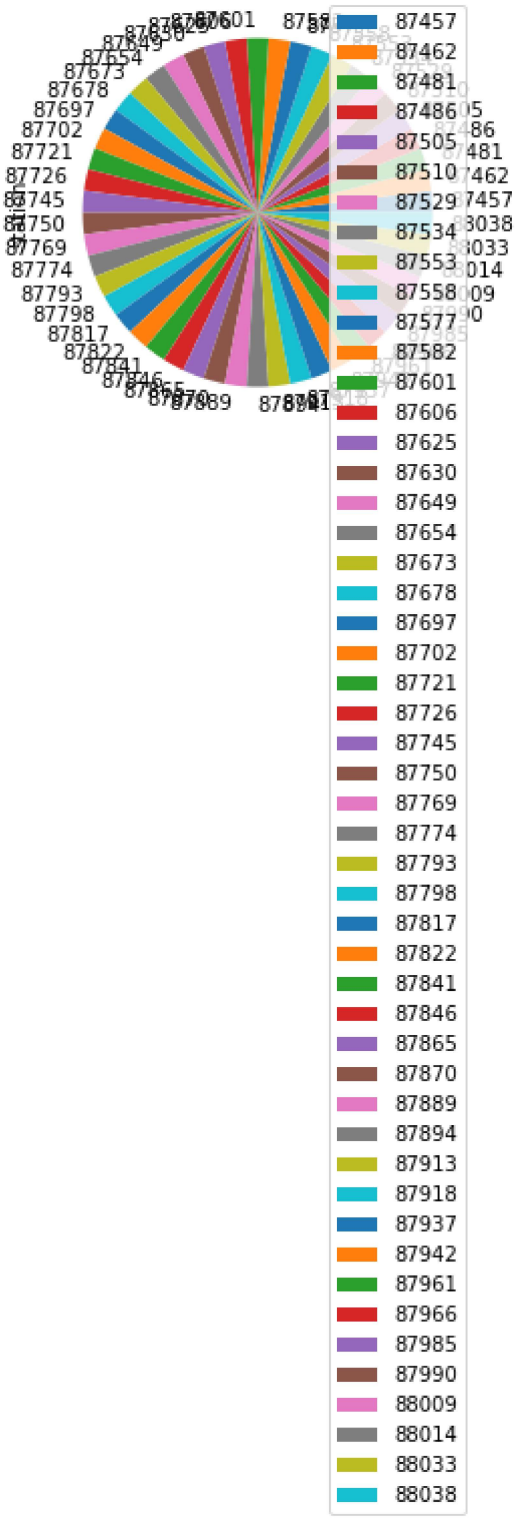
Out[13]:
```
<AxesSubplot:>
```



Pie chart

In [14]:
```
b.plot.pie(y='station' )
```

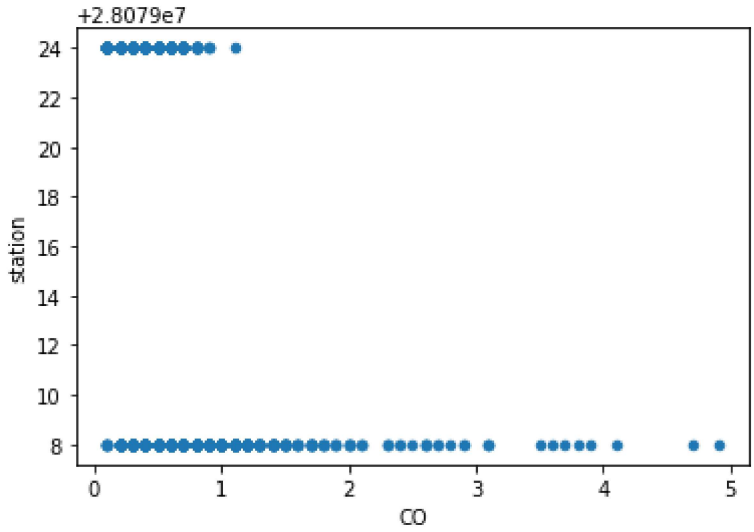
Out[14]: <AxesSubplot:ylabel='station'>



Scatter chart

In [15]:
```
data.plot.scatter(x='CO' ,y='station')
```

Out[15]: <AxesSubplot:xlabel='CO', ylabel='station'>



In [16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4127 entries, 87457 to 158286
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     4127 non-null   object
 1   BEN      4127 non-null   float64
 2   CH4      4127 non-null   float64
 3   CO       4127 non-null   float64
 4   EBE      4127 non-null   float64
 5   NMHC     4127 non-null   float64
 6   NO       4127 non-null   float64
 7   NO_2     4127 non-null   float64
 8   NOx      4127 non-null   float64
 9   O_3      4127 non-null   float64
 10  PM10     4127 non-null   float64
 11  PM25     4127 non-null   float64
 12  SO_2     4127 non-null   float64
 13  TCH      4127 non-null   float64
```

In [17]:
```
df.describe()
```

Out[17]:

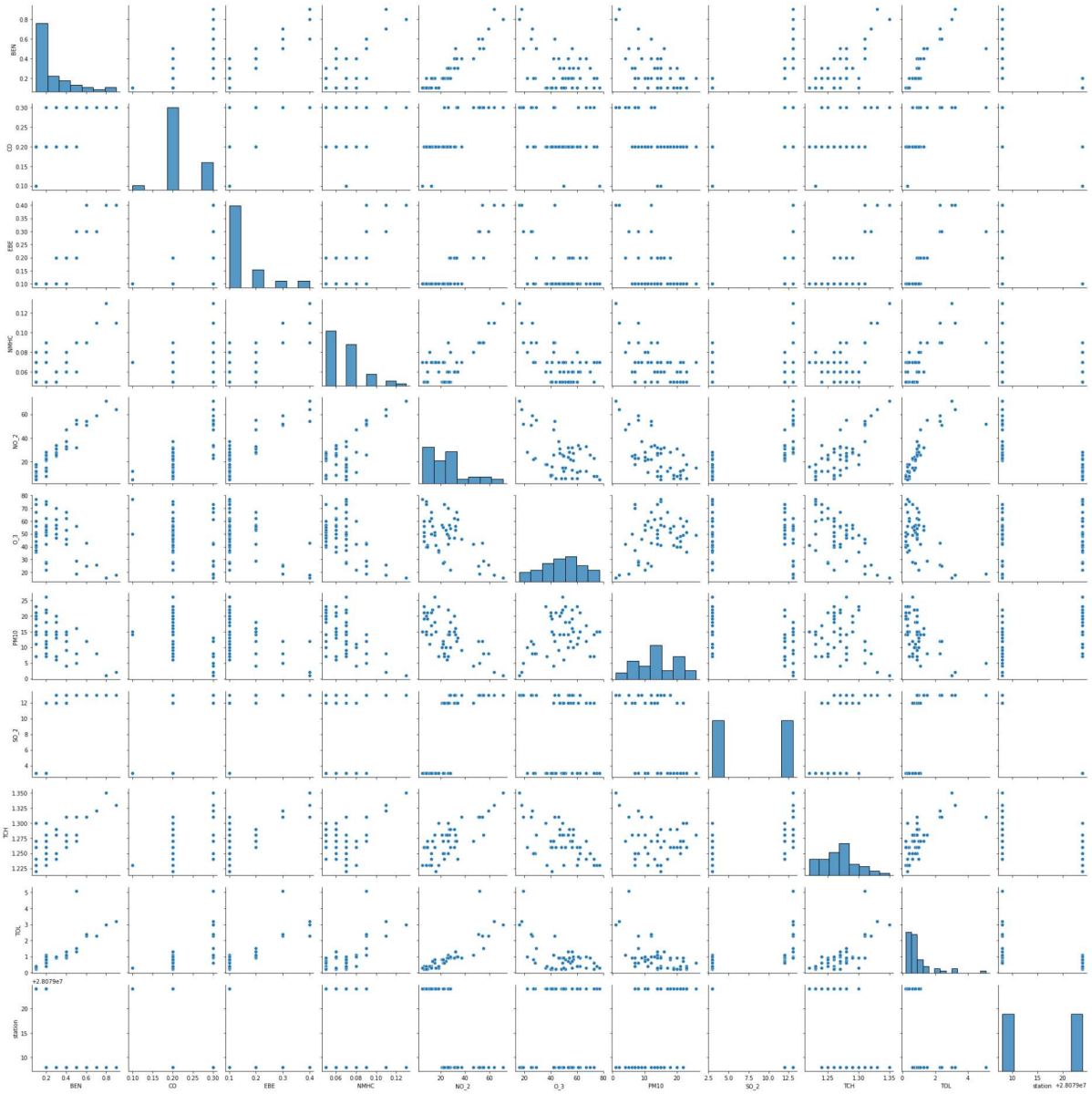
	BEN	CH4	CO	EBE	NMHC	NO	NO_
count	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4127.0000(
mean	0.919918	1.323732	0.417858	0.578168	0.097269	41.785316	58.0690!
std	1.123078	0.215742	0.342871	0.962000	0.094035	71.118499	38.9741'
min	0.100000	1.100000	0.100000	0.100000	0.000000	1.000000	1.0000(
25%	0.300000	1.180000	0.200000	0.100000	0.050000	3.000000	30.0000(
50%	0.600000	1.270000	0.300000	0.300000	0.080000	16.000000	54.0000(
75%	1.100000	1.400000	0.500000	0.700000	0.110000	50.000000	78.0000(
max	19.600000	3.630000	4.900000	16.700001	1.420000	879.000000	349.0000(

In [18]:
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL','station']]
```

EDA AND VISUALIZATION

In [19]:
```
sns.pairplot(df1[0:50])
```

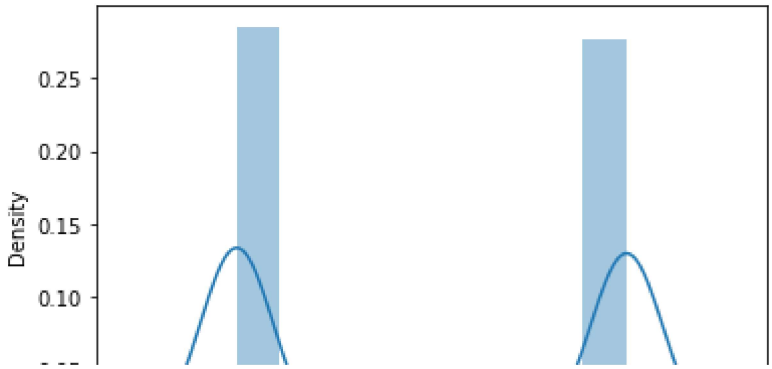
Out[19]: <seaborn.axisgrid.PairGrid at 0x2adbd166580>



In [20]:
```
sns.distplot(df1['station'])
```

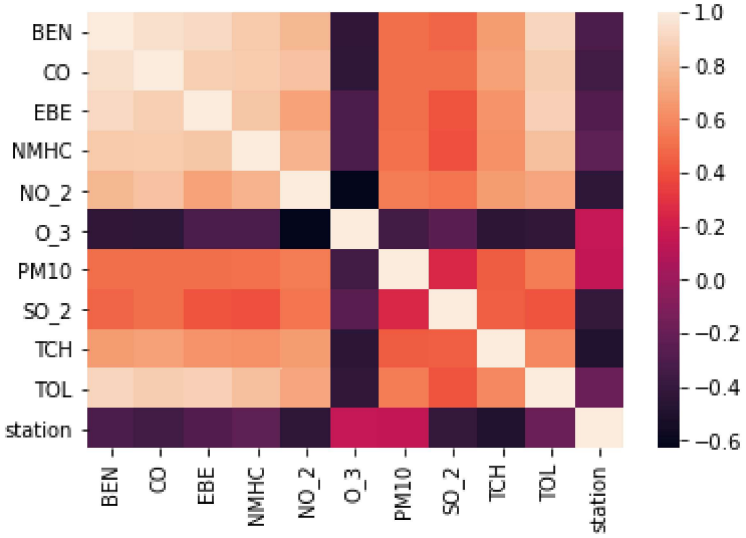
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [21]:
```
sns.heatmap(df1.corr())
```

Out[21]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [22]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [23]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

Linear Regression

In [24]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[24]:
```
LinearRegression()
```

In [25]:
```
lr.intercept_
```

Out[25]:
```
28079039.459184237
```

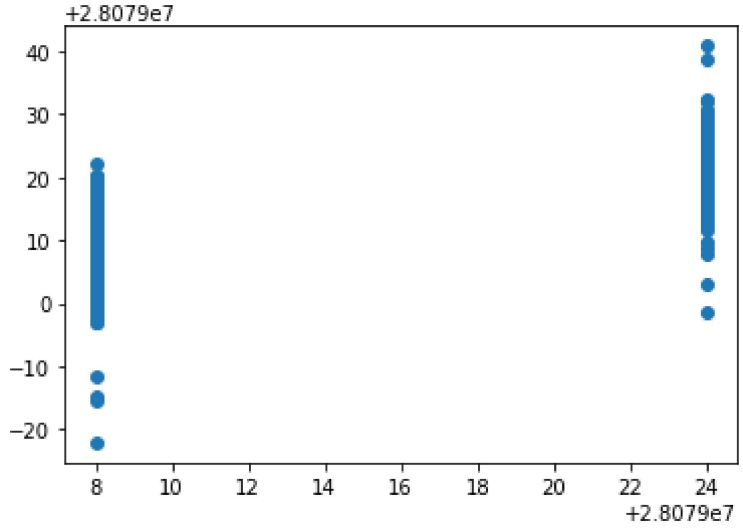
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	2.187711
CO	-3.492222
EBE	-3.622353
NMHC	36.153021
NO_2	-0.161764
O_3	-0.075336
PM10	0.342453
SO_2	-0.282252
TCH	-13.727623
TOL	0.256303

In [27]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[27]: <matplotlib.collections.PathCollection at 0x2adc5788250>



ACCURACY

In [28]:
```
lr.score(x_test,y_test)
```

Out[28]: 0.5687259910957732

In [29]:
```
lr.score(x_train,y_train)
```

Out[29]: 0.6178684385681026

Ridge and Lasso

In [30]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [31]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[31]: Ridge(alpha=10)

Accuracy(Ridge)

In [32]:
```
rr.score(x_test,y_test)
```

Out[32]: 0.5521871977947899

In [33]:
```
rr.score(x_train,y_train)
```

Out[33]:
```
0.6010735278768111
```

In [34]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[34]:
```
Lasso(alpha=10)
```

In [35]:
```
la.score(x_train,y_train)
```

Out[35]:
```
0.4064749701897894
```

Accuracy(Lasso)

In [36]:
```
la.score(x_test,y_test)
```

Out[36]:
```
0.38307427190670196
```

In [37]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[37]:
```
ElasticNet()
```

In [38]:
```
en.coef_
```

Out[38]:
```
array([-0.        , -0.        , -0.        ,  0.        , -0.16733424,
       -0.05592105,  0.32031324, -0.36344236, -0.        ,  0.13025385])
```

In [39]:
```
en.intercept_
```

Out[39]:
```
28079023.11159102
```

In [40]:
```
prediction=en.predict(x_test)
```

In [41]:
```
en.score(x_test,y_test)
```

Out[41]:
```
0.4332595621281703
```

Evaluation Metrics

In [42]:
```
from sklearn import metrics
print(metrics.mean_absolute_error(y_test,prediction))
print(metrics.mean_squared_error(y_test,prediction))
print(np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
5.196588886331968
36.22354191867041
6.018599664263308
```

Logistic Regression

In [43]:
```
from sklearn.linear_model import LogisticRegression
```

In [44]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [45]:
```
feature_matrix.shape
```

Out[45]:
```
(4127, 10)
```

In [46]:
```
target_vector.shape
```

Out[46]:
```
(4127,)
```

In [47]:
```
from sklearn.preprocessing import StandardScaler
```

In [48]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [49]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[49]:
```
LogisticRegression(max_iter=10000)
```

In [50]:
```
observation=[[1,2,3,4,5,6,7,8,9,10]]
```

In [51]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [52]:
```
logr.classes_
```

Out[52]:
```
array([28079008, 28079024], dtype=int64)
```

In [53]:
```
logr.score(fs,target_vector)
```

Out[53]:
```
0.9437848315968016
```

In [54]:
```
logr.predict_proba(observation)[0][0]
```

Out[54]:
```
0.9999999999725541
```

In [55]:
```
logr.predict_proba(observation)
```

Out[55]:
```
array([[1.00000000e+00, 2.74458959e-11]])
```

Random Forest

In [56]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [57]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[57]:
```
RandomForestClassifier()
```

In [58]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [59]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="ac
grid_search.fit(x_train,y_train)
```

Out[59]:
```
GridSearchCV(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'min_samples_leaf': [5, 10, 15, 20, 25],
                         'n_estimators': [10, 20, 30, 40, 50]},
             scoring='accuracy')
```

In [60]:
```
grid_search.best_score_
```

Out[60]:
```
0.9705678670360112
```

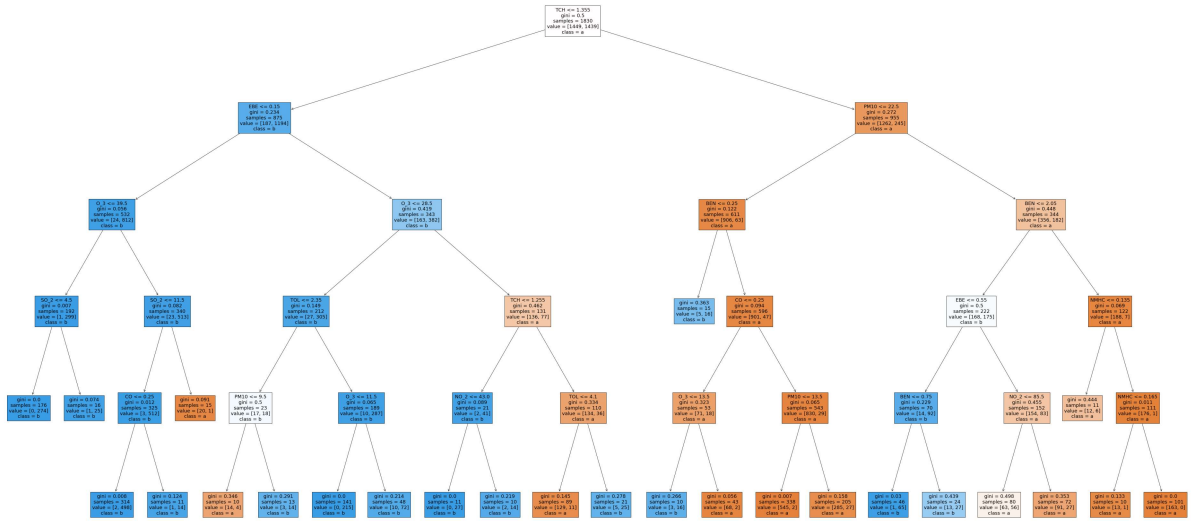
In [61]:
```
rfc_best=grid_search.best_estimator_
```

In [62]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['a','b'
```

Out[62]:
```
[Text(2141.1627906976746, 1993.2, 'TCH <= 1.355\ngini = 0.5\nsamples = 1830\n
value = [1449, 1439]\nclass = a'),
 Text(986.2325581395348, 1630.8000000000002, 'EBE <= 0.15\ngini = 0.234\nsamp
les = 875\nvalue = [187, 1194]\nclass = b'),
 Text(415.25581395348837, 1268.4, 'O_3 <= 39.5\ngini = 0.056\nsamples = 532\n
value = [24, 812]\nclass = b'),
 Text(207.62790697674419, 906.0, 'SO_2 <= 4.5\ngini = 0.007\nsamples = 192\nv
alue = [1, 299]\nclass = b'),
 Text(103.81395348837209, 543.5999999999999, 'gini = 0.0\nsamples = 176\nvalu
e = [0, 274]\nclass = b'),
 Text(311.4418604651163, 543.5999999999999, 'gini = 0.074\nsamples = 16\nvalu
e = [1, 25]\nclass = b'),
 Text(622.8837209302326, 906.0, 'SO_2 <= 11.5\ngini = 0.082\nsamples = 340\nv
alue = [23, 513]\nclass = b'),
 Text(519.0697674418604, 543.5999999999999, 'CO <= 0.25\ngini = 0.012\nsample
s = 325\nvalue = [3, 512]\nclass = b'),
 Text(415.25581395348837, 181.19999999999982, 'gini = 0.008\nsamples = 314\nv
alue = [2, 498]\nclass = b'),
 Text(622.8837209302326, 181.19999999999982, 'gini = 0.124\nsamples = 11\nval
ue = [1, 14]\nclass = b'),
 Text(726.6976744186046, 543.5999999999999, 'gini = 0.091\nsamples = 15\nvalu
e = [20, 1]\nclass = a'),
 Text(1557.2093023255813, 1268.4, 'O_3 <= 28.5\ngini = 0.419\nsamples = 343\n
value = [163, 382]\nclass = b'),
 Text(1141.953488372093, 906.0, 'TOL <= 2.35\ngini = 0.149\nsamples = 212\nva
lue = [27, 305]\nclass = b'),
 Text(934.3255813953489, 543.5999999999999, 'PM10 <= 9.5\ngini = 0.5\nsamples
= 23\nvalue = [17, 18]\nclass = b'),
 Text(830.5116279069767, 181.19999999999982, 'gini = 0.346\nsamples = 10\nval
ue = [14, 4]\nclass = a'),
 Text(1038.139534883721, 181.19999999999982, 'gini = 0.291\nsamples = 13\nval
ue = [3, 14]\nclass = b'),
 Text(1349.5813953488373, 543.5999999999999, 'O_3 <= 11.5\ngini = 0.065\nsamp
les = 189\nvalue = [10, 287]\nclass = b'),
 Text(1245.7674418604652, 181.19999999999982, 'gini = 0.0\nsamples = 141\nval
ue = [0, 215]\nclass = b'),
 Text(1453.3953488372092, 181.19999999999982, 'gini = 0.214\nsamples = 48\nva
lue = [10, 72]\nclass = b'),
 Text(1972.4651162790697, 906.0, 'TCH <= 1.255\ngini = 0.462\nsamples = 131\n
value = [136, 77]\nclass = a'),
 Text(1764.8372093023256, 543.5999999999999, 'NO_2 <= 43.0\ngini = 0.089\nsam
ples = 21\nvalue = [2, 41]\nclass = b'),
 Text(1661.0232558139535, 181.19999999999982, 'gini = 0.0\nsamples = 11\nvalu
e = [0, 27]\nclass = b'),
 Text(1868.6511627906978, 181.19999999999982, 'gini = 0.219\nsamples = 10\nva
lue = [2, 14]\nclass = b'),
 Text(2180.093023255814, 543.5999999999999, 'TOL <= 4.1\ngini = 0.334\nsample
s = 110\nvalue = [134, 36]\nclass = a'),
 Text(2076.279069767442, 181.19999999999982, 'gini = 0.145\nsamples = 89\nval
ue = [129, 11]\nclass = a'),
 Text(2283.906976744186, 181.19999999999982, 'gini = 0.278\nsamples = 21\nval
ue = [5, 25]\nclass = b'),
 Text(3296.093023255814, 1630.8000000000002, 'PM10 <= 22.5\ngini = 0.272\nsam
ples = 955\nvalue = [1262, 245]\nclass = a'),
 Text(2699.1627906976746, 1268.4, 'BEN <= 0.25\ngini = 0.122\nsamples = 611\n
value = [906, 63]\nclass = a'),
 Text(2595.3488372093025, 906.0, 'gini = 0.363\nsamples = 15\nvalue = [5, 16]
```

```
\nclass = b'),
 Text(2802.9767441860463, 906.0, 'CO <= 0.25\ngini = 0.094\nsamples = 596\nva
lue = [901, 47]\nclass = a'),
 Text(2595.3488372093025, 543.5999999999999, 'O_3 <= 13.5\ngini = 0.323\nsamp
les = 53\nvalue = [71, 18]\nclass = a'),
 Text(2491.5348837209303, 181.19999999999982, 'gini = 0.266\nsamples = 10\nva
lue = [3, 16]\nclass = b'),
 Text(2699.1627906976746, 181.19999999999982, 'gini = 0.056\nsamples = 43\nva
lue = [68, 2]\nclass = a'),
 Text(3010.6046511627906, 543.5999999999999, 'PM10 <= 13.5\ngini = 0.065\nsam
ples = 543\nvalue = [830, 29]\nclass = a'),
 Text(2906.7906976744184, 181.19999999999982, 'gini = 0.007\nsamples = 338\nv
alue = [545, 2]\nclass = a'),
 Text(3114.4186046511627, 181.19999999999982, 'gini = 0.158\nsamples = 205\nv
alue = [285, 27]\nclass = a'),
 Text(3893.0232558139533, 1268.4, 'BEN <= 2.05\ngini = 0.448\nsamples = 344\n
value = [356, 182]\nclass = a'),
 Text(3633.4883720930234, 906.0, 'EBE <= 0.55\ngini = 0.5\nsamples = 222\nval
ue = [168, 175]\nclass = b'),
 Text(3425.860465116279, 543.5999999999999, 'BEN <= 0.75\ngini = 0.229\nsampl
es = 70\nvalue = [14, 92]\nclass = b'),
 Text(3322.046511627907, 181.19999999999982, 'gini = 0.03\nsamples = 46\nvalu
e = [1, 65]\nclass = b'),
 Text(3529.6744186046512, 181.19999999999982, 'gini = 0.439\nsamples = 24\nva
lue = [13, 27]\nclass = b'),
 Text(3841.1162790697676, 543.5999999999999, 'NO_2 <= 85.5\ngini = 0.455\nsam
ples = 152\nvalue = [154, 83]\nclass = a'),
 Text(3737.3023255813955, 181.19999999999982, 'gini = 0.498\nsamples = 80\nva
lue = [63, 56]\nclass = a'),
 Text(3944.9302325581393, 181.19999999999982, 'gini = 0.353\nsamples = 72\nva
lue = [91, 27]\nclass = a'),
 Text(4152.558139534884, 906.0, 'NMHC <= 0.135\ngini = 0.069\nsamples = 122\n
value = [188, 7]\nclass = a'),
 Text(4048.7441860465115, 543.5999999999999, 'gini = 0.444\nsamples = 11\nval
ue = [12, 6]\nclass = a'),
 Text(4256.372093023256, 543.5999999999999, 'NMHC <= 0.165\ngini = 0.011\nsam
ples = 111\nvalue = [176, 1]\nclass = a'),
 Text(4152.558139534884, 181.19999999999982, 'gini = 0.133\nsamples = 10\nval
ue = [13, 1]\nclass = a'),
 Text(4360.186046511628, 181.19999999999982, 'gini = 0.0\nsamples = 101\nvalu
e = [163, 0]\nclass = a')]
```



Conclusion

Accuracy

In [63]:
```
lr.score(x_train,y_train)
```

Out[63]: 0.6178684385681026

In [64]:
```
rr.score(x_train,y_train)
```

Out[64]: 0.6010735278768111

In [65]:
```
la.score(x_train,y_train)
```

Out[65]: 0.4064749701897894

In [66]:
```
en.score(x_test,y_test)
```

Out[66]: 0.4332595621281703

In [67]:
```
logr.score(fs,target_vector)
```

Out[67]: 0.9437848315968016

In [68]:
```
grid_search.best_score_
```

Out[68]: 0.9705678670360112

Random Forest is suitable for this dataset

In [ ]: