20104028

Heamnath N

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_2018.csv")
df
```

Out[2]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH	T
0	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	29.0	31.0	NaN	NaN	NaN	2.0	NaN	N
1	2018-03-01 01:00:00	0.5	1.39	0.3	0.2	0.02	6.0	40.0	49.0	52.0	5.0	4.0	3.0	1.41	
2	2018-03-01 01:00:00	0.4	NaN	NaN	0.2	NaN	4.0	41.0	47.0	NaN	NaN	NaN	NaN	NaN	
3	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	35.0	37.0	54.0	NaN	NaN	NaN	NaN	N
4	2018-03-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	27.0	29.0	49.0	NaN	NaN	3.0	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
69091	2018-02-01 00:00:00	NaN	NaN	0.5	NaN	NaN	66.0	91.0	192.0	1.0	35.0	22.0	NaN	NaN	N
69092	2018-02-01 00:00:00	NaN	NaN	0.7	NaN	NaN	87.0	107.0	241.0	NaN	29.0	NaN	15.0	NaN	N
69093	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	28.0	48.0	91.0	2.0	NaN	NaN	NaN	NaN	N

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH	T
69094	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	141.0	103.0	320.0	2.0	NaN	NaN	NaN	NaN	N
69095	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	69.0	96.0	202.0	3.0	26.0	NaN	NaN	NaN	N

69096 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: 4562 entries, 1 to 69078
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4562 non-null   object 
 1   BEN      4562 non-null   float64
 2   CH4      4562 non-null   float64
 3   CO       4562 non-null   float64
 4   EBE      4562 non-null   float64
 5   NMHC     4562 non-null   float64
 6   NO       4562 non-null   float64
 7   NO_2     4562 non-null   float64
 8   NOx      4562 non-null   float64
 9   O_3      4562 non-null   float64
 10  PM10     4562 non-null   float64
 11  PM25     4562 non-null   float64
 12  SO_2     4562 non-null   float64
 13  TCH      4562 non-null   float64
 14  TOL      4562 non-null   float64
 15  station  4562 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 605.9+ KB
```

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

Out[6]:

	CO	station
1	0.3	28079008

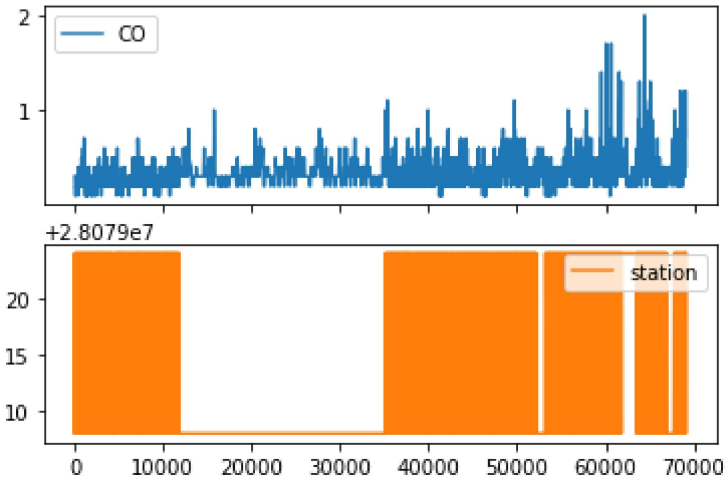
	CO	station
6	0.2	28079024
25	0.2	28079008
30	0.2	28079024
49	0.2	28079008
...	...	...
69030	0.7	28079024
69049	1.2	28079008
69054	0.6	28079024
69073	1.0	28079008
69078	0.4	28079024

4562 rows × 2 columns

Line chart

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

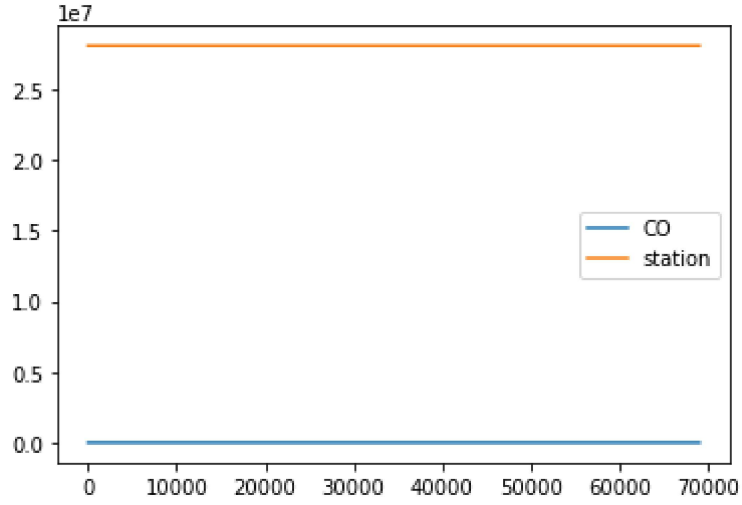
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

Out[8]:
```
<AxesSubplot:>
```

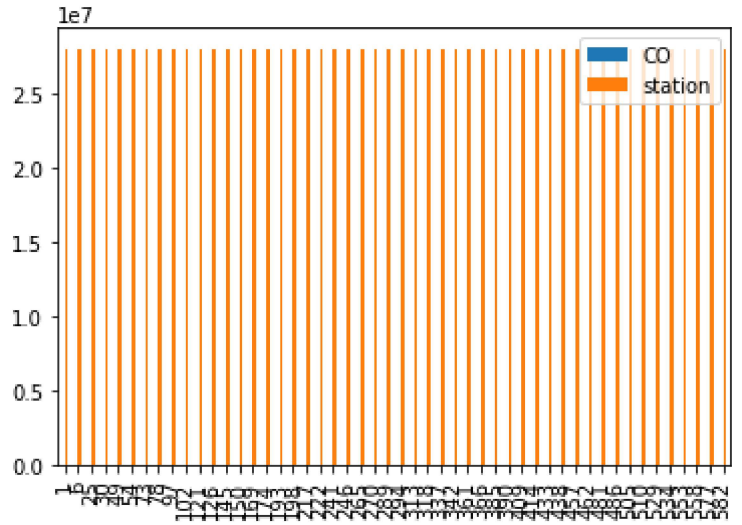


Bar chart

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

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

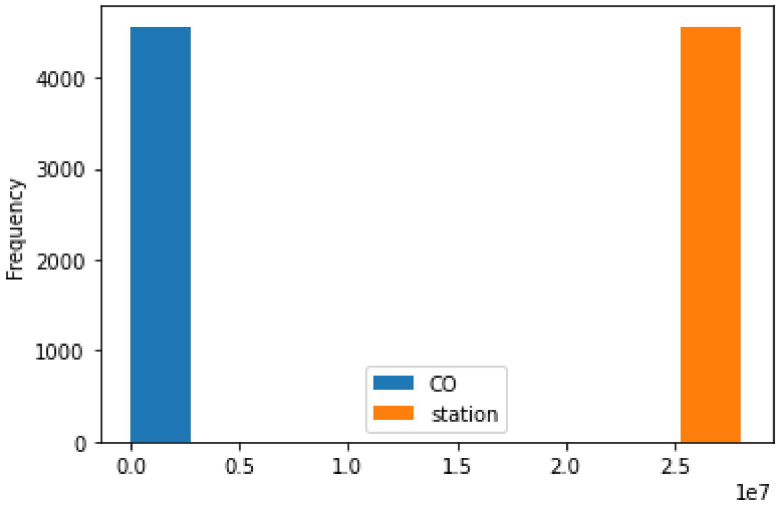
Out[10]: <AxesSubplot:>



Histogram

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

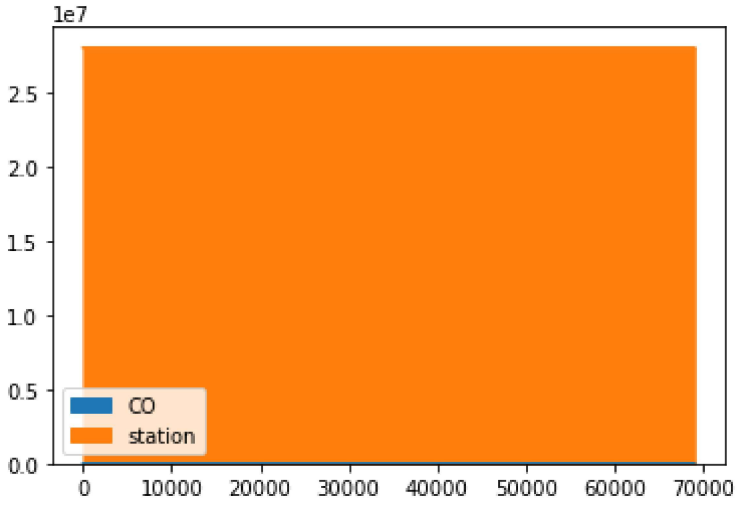
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

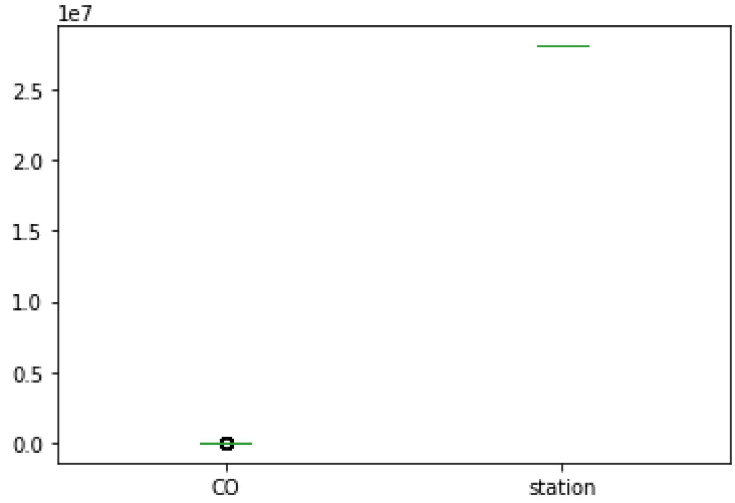
Out[12]: <AxesSubplot:>



Box chart

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

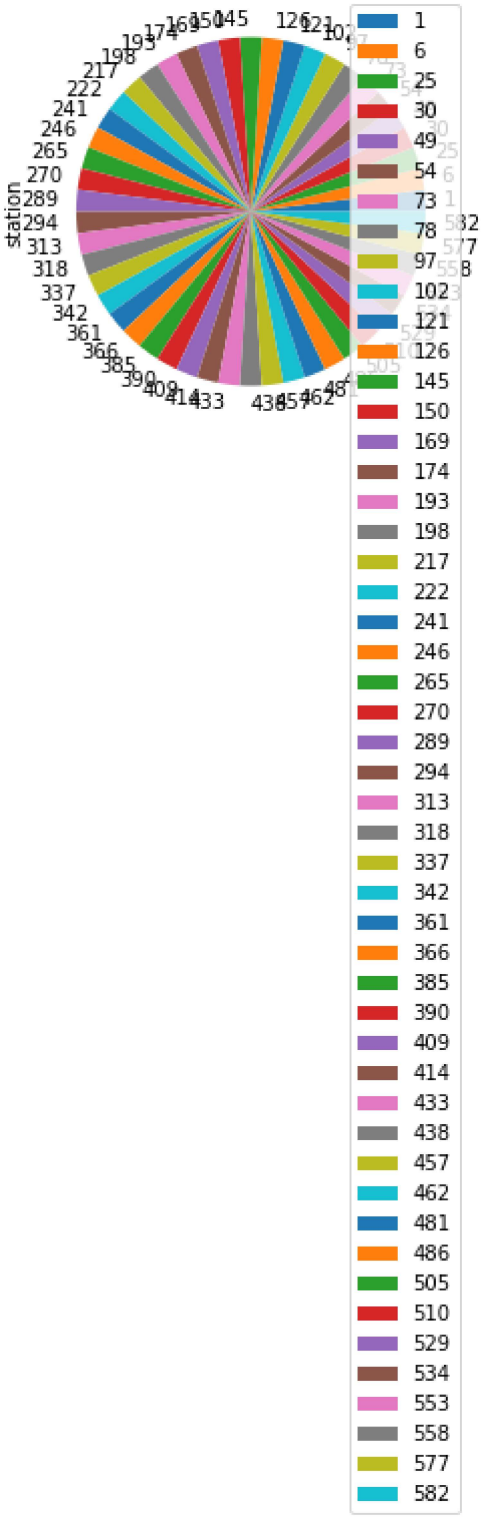
Out[13]: <AxesSubplot:>



Pie chart

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

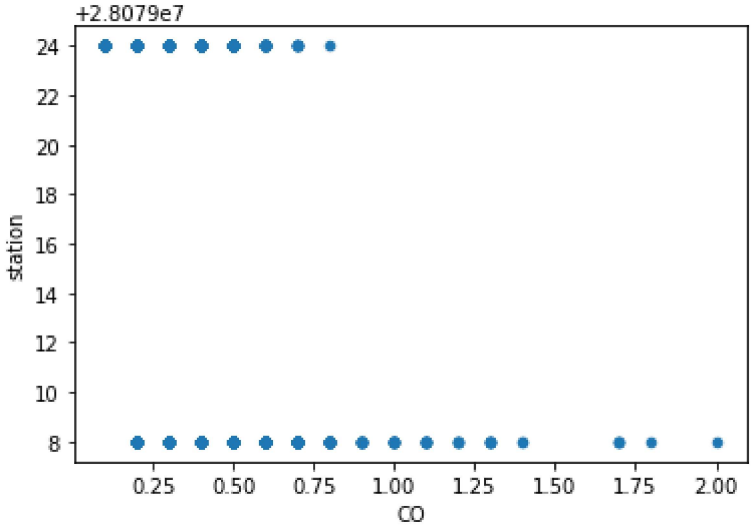
Out[14]: <AxesSubplot:ylabel='station'>



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: 4562 entries, 1 to 69078
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4562 non-null   object 
 1   BEN      4562 non-null   float64
 2   CH4      4562 non-null   float64
 3   CO       4562 non-null   float64
 4   EBE      4562 non-null   float64
 5   NMHC     4562 non-null   float64
 6   NO       4562 non-null   float64
 7   NO_2     4562 non-null   float64
 8   NOx      4562 non-null   float64
 9   O_3      4562 non-null   float64
 10  PM10     4562 non-null   float64
 11  PM25     4562 non-null   float64
 12  SO_2     4562 non-null   float64
 13  TCH      4562 non-null   float64
 14  TOL      4562 non-null   float64
 15  station  4562 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 605.9+ KB
```

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

Out[17]:

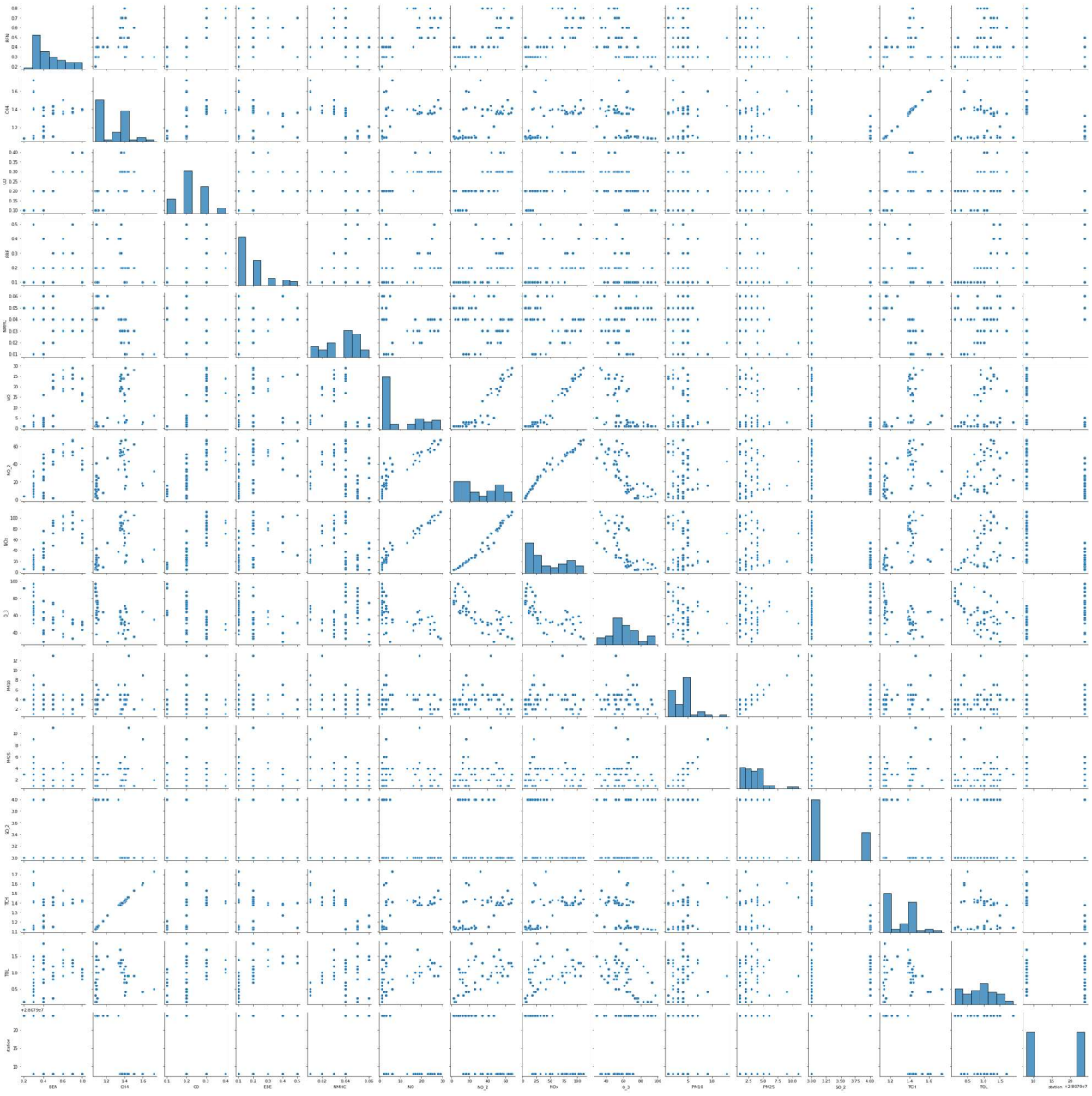
	BEN	CH4	CO	EBE	NMHC	NO	NO_2	
count	4562.00000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	4562.0
mean	0.69349	1.329163	0.330579	0.286782	0.056773	21.742218	44.152126	77.4
std	0.46832	0.214399	0.161489	0.354442	0.037711	35.539531	30.234015	79.2
min	0.10000	0.020000	0.100000	0.100000	0.000000	1.000000	1.000000	2.0
25%	0.40000	1.120000	0.200000	0.100000	0.030000	1.000000	20.000000	24.0
50%	0.60000	1.390000	0.300000	0.200000	0.050000	9.000000	41.000000	56.0
75%	0.90000	1.420000	0.400000	0.300000	0.070000	27.000000	64.000000	106.0

	BEN	CH4	CO	EBE	NMHC	NO	NO_2	
max	6.60000	3.920000	2.000000	7.400000	0.490000	431.000000	184.000000	844.0

EDA AND VISUALIZATION

In [18]:
```
sns.pairplot(df[0:50])
```

Out[18]: <seaborn.axisgrid.PairGrid at 0x2020ebf93a0>

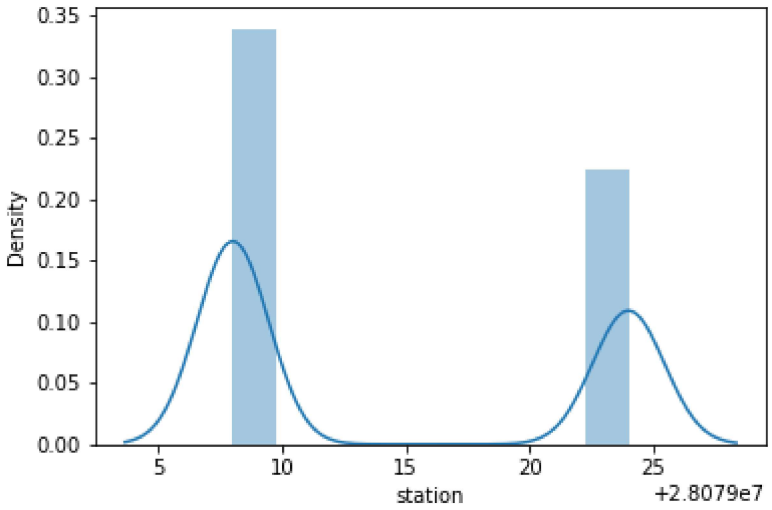


In [19]:
```
sns.distplot(df['station'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
```

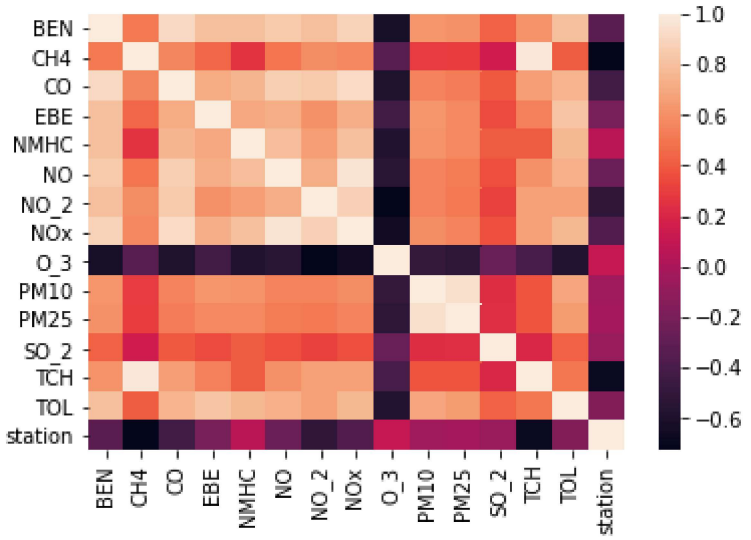
```
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[19]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [20]:
```
sns.heatmap(df.corr())
```

Out[20]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [21]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [22]:
```
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 [23]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[23]:
```
LinearRegression()
```

In [24]:
```
lr.intercept_
```

Out[24]:
```
28079045.423624586
```

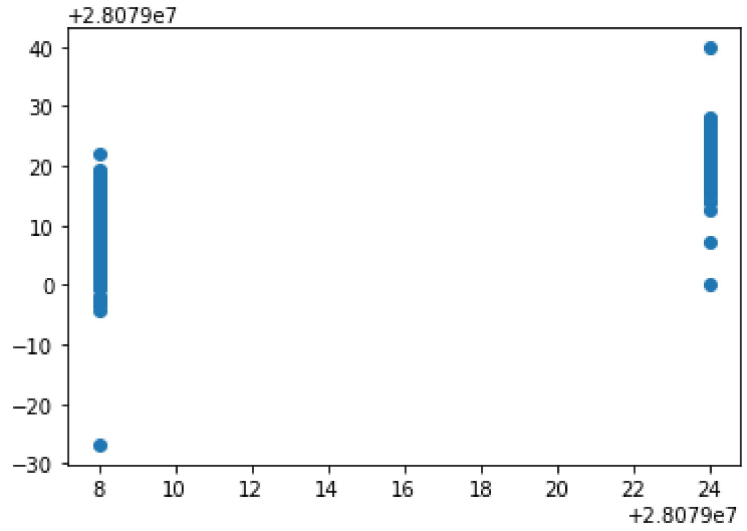
In [25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	-1.568866
CO	-20.679861
EBE	0.800929
NMHC	148.652061
NO	0.031730
NO_2	-0.148463
O_3	-0.082006
PM10	0.052993
PM25	0.090646
SO_2	-0.030789
TCH	-16.799550
TOL	-0.193482

In [26]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[26]:
```
<matplotlib.collections.PathCollection at 0x2021b8030d0>
```



ACCURACY

In [27]:
```
lr.score(x_test,y_test)
```

Out[27]: 0.8110075514813904

In [28]:
```
lr.score(x_train,y_train)
```

Out[28]: 0.8096385926089058

Ridge and Lasso

In [29]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [30]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[30]: Ridge(alpha=10)

Accuracy(Ridge)

In [31]:
```
rr.score(x_test,y_test)
```

Out[31]: 0.7136620338564642

In [32]:
```
rr.score(x_train,y_train)
```

Out[32]: 0.7055993253893034

In [33]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[33]:
```
Lasso(alpha=10)
```

In [34]:
```
la.score(x_train,y_train)
```

Out[34]:
```
0.4133909143336273
```

Accuracy(Lasso)

In [35]:
```
la.score(x_test,y_test)
```

Out[35]:
```
0.43117592115261916
```

In [36]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[36]:
```
ElasticNet()
```

In [37]:
```
en.coef_
```

Out[37]:
```
array([-0.        , -0.        , -0.        ,  0.        ,  0.0322608 ,
       -0.28886481, -0.14691916,  0.27114545, -0.07808131,  0.04482872,
       -0.11152484,  0.        ])
```

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

Out[38]:
```
28079030.103471804
```

In [39]:
```
prediction=en.predict(x_test)
```

In [40]:
```
en.score(x_test,y_test)
```

Out[40]:
```
0.4882985369903171
```

Evaluation Metrics

In [41]:
```
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)))
```

```
4.728265913792494
31.309586326777108
5.59549696870413
```

Logistic Regression

In [42]:
```
from sklearn.linear_model import LogisticRegression
```

In [43]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [44]:
```
feature_matrix.shape
```

Out[44]: (4562, 12)

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

Out[45]: (4562,)

In [46]:
```
from sklearn.preprocessing import StandardScaler
```

In [47]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [48]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[48]: LogisticRegression(max_iter=10000)

In [49]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12]]
```

In [50]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [51]:
```
logr.classes_
```

Out[51]: array([28079008, 28079024], dtype=int64)

In [52]:
```
logr.score(fs,target_vector)
```

Out[52]: 0.9890398947829899

In [53]:
```
logr.predict_proba(observation)[0][0]
```

Out[53]:
```
1.0
```

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

Out[54]:
```
array([[1.00000000e+00, 1.09190927e-22]])
```

Random Forest

In [55]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [56]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[56]:
```
RandomForestClassifier()
```

In [57]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [58]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="accuracy")
grid_search.fit(x_train,y_train)
```

Out[58]:
```
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 [59]:
```
grid_search.best_score_
```

Out[59]:
```
0.9934226219901664
```

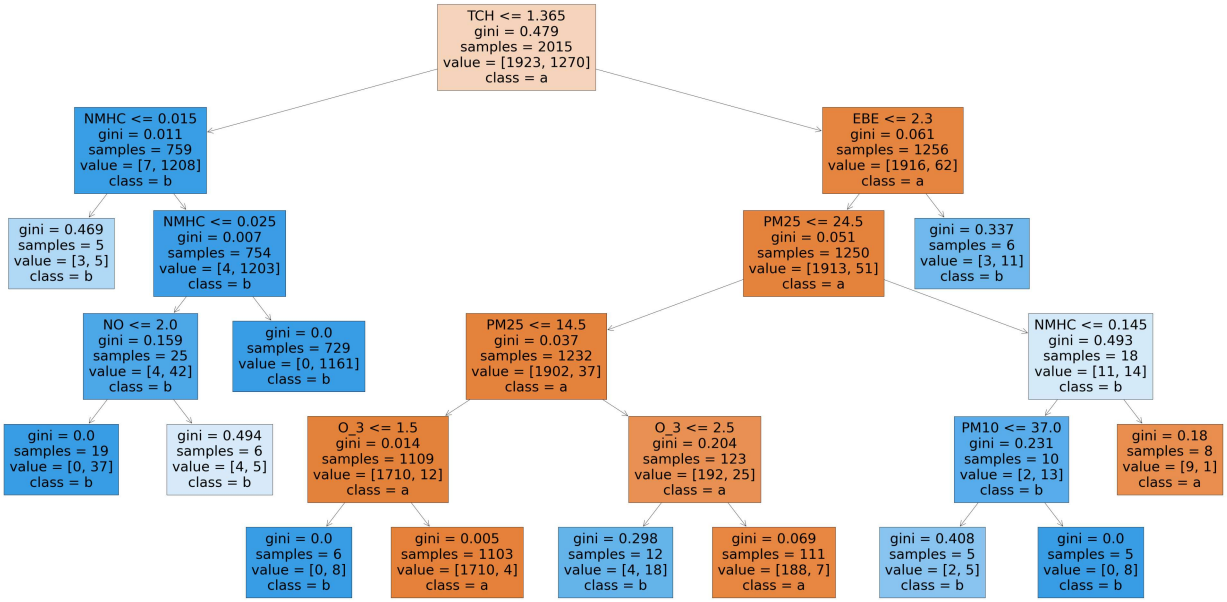
In [60]:
```
rfc_best=grid_search.best_estimator_
```

In [61]:
```
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','c','d']
```

Out[61]:
```
[Text(1883.25, 1993.2, 'TCH <= 1.365\ngini = 0.479\nsamples = 2015\nvalue = [1923, 1270]
\nclass = a'),
 Text(558.0, 1630.8000000000002, 'NMHC <= 0.015\ngini = 0.011\nsamples = 759\nvalue =
[7, 1208]\nclass = b'),
```

```
 Text(279.0, 1268.4, 'gini = 0.469\nsamples = 5\nvalue = [3, 5]\nclass = b'),
 Text(837.0, 1268.4, 'NMHC <= 0.025\ngini = 0.007\nsamples = 754\nvalue = [4, 1203]\ncla
ss = b'),
 Text(558.0, 906.0, 'NO <= 2.0\ngini = 0.159\nsamples = 25\nvalue = [4, 42]\nclass =
b'),
 Text(279.0, 543.5999999999999, 'gini = 0.0\nsamples = 19\nvalue = [0, 37]\nclass = b'),
 Text(837.0, 543.5999999999999, 'gini = 0.494\nsamples = 6\nvalue = [4, 5]\nclass = b'),
 Text(1116.0, 906.0, 'gini = 0.0\nsamples = 729\nvalue = [0, 1161]\nclass = b'),
 Text(3208.5, 1630.8000000000002, 'EBE <= 2.3\ngini = 0.061\nsamples = 1256\nvalue = [19
16, 62]\nclass = a'),
 Text(2929.5, 1268.4, 'PM25 <= 24.5\ngini = 0.051\nsamples = 1250\nvalue = [1913, 51]\nc
lass = a'),
 Text(1953.0, 906.0, 'PM25 <= 14.5\ngini = 0.037\nsamples = 1232\nvalue = [1902, 37]\ncl
ass = a'),
 Text(1395.0, 543.5999999999999, 'O_3 <= 1.5\ngini = 0.014\nsamples = 1109\nvalue = [171
0, 12]\nclass = a'),
 Text(1116.0, 181.19999999999982, 'gini = 0.0\nsamples = 6\nvalue = [0, 8]\nclass = b'),
 Text(1674.0, 181.19999999999982, 'gini = 0.005\nsamples = 1103\nvalue = [1710, 4]\nclas
s = a'),
 Text(2511.0, 543.5999999999999, 'O_3 <= 2.5\ngini = 0.204\nsamples = 123\nvalue = [192,
25]\nclass = a'),
 Text(2232.0, 181.19999999999982, 'gini = 0.298\nsamples = 12\nvalue = [4, 18]\nclass =
b'),
 Text(2790.0, 181.19999999999982, 'gini = 0.069\nsamples = 111\nvalue = [188, 7]\nclass
= a'),
 Text(3906.0, 906.0, 'NMHC <= 0.145\ngini = 0.493\nsamples = 18\nvalue = [11, 14]\nclass
= b'),
 Text(3627.0, 543.5999999999999, 'PM10 <= 37.0\ngini = 0.231\nsamples = 10\nvalue = [2,
13]\nclass = b'),
 Text(3348.0, 181.19999999999982, 'gini = 0.408\nsamples = 5\nvalue = [2, 5]\nclass =
b'),
 Text(3906.0, 181.19999999999982, 'gini = 0.0\nsamples = 5\nvalue = [0, 8]\nclass = b'),
 Text(4185.0, 543.5999999999999, 'gini = 0.18\nsamples = 8\nvalue = [9, 1]\nclass = a'),
 Text(3487.5, 1268.4, 'gini = 0.337\nsamples = 6\nvalue = [3, 11]\nclass = b')]
```



Conclusion

Accuracy

In [62]:
```
print("Linear Regression:",lr.score(x_test,y_test))
print("Ridge Regression:",rr.score(x_test,y_test))
print("Lasso Regression",la.score(x_test,y_test))
print("ElasticNet Regression:",en.score(x_test,y_test))
print("Logistic Regression:",logr.score(fs,target_vector))
print("Random Forest:",grid_search.best_score_)
```

```
Linear Regression: 0.8110075514813904
Ridge Regression: 0.7136620338564642
Lasso Regression 0.43117592115261916
ElasticNet Regression: 0.4882985369903171
Logistic Regression: 0.9890398947829899
Random Forest: 0.9934226219901664
```

Random Forest is suitable for this dataset