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_2012.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	statio
0	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	7.0	18.0	NaN	NaN	NaN	2.0	NaN	NaN	2807900
1	2012-09-01 01:00:00	0.3	0.3	0.7	NaN	3.0	18.0	55.0	10.0	9.0	1.0	NaN	2.4	2807900
2	2012-09-01 01:00:00	0.4	NaN	0.7	NaN	2.0	10.0	NaN	NaN	NaN	NaN	NaN	1.5	2807901
3	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	1.0	6.0	50.0	NaN	NaN	NaN	NaN	NaN	2807901
4	2012-09-01 01:00:00	NaN	NaN	NaN	NaN	1.0	13.0	54.0	NaN	NaN	3.0	NaN	NaN	2807901
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
210715	2012-03-01 00:00:00	NaN	0.6	NaN	NaN	37.0	84.0	14.0	NaN	NaN	NaN	NaN	NaN	2807905
210716	2012-03-01 00:00:00	NaN	0.4	NaN	NaN	5.0	76.0	NaN	17.0	NaN	7.0	NaN	NaN	2807905
210717	2012-03-01 00:00:00	NaN	NaN	NaN	0.34	3.0	41.0	24.0	NaN	NaN	NaN	1.34	NaN	2807905

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	statio
210718	2012-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	44.0	36.0	NaN	NaN	NaN	NaN	NaN	2807905
210719	2012-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	56.0	40.0	18.0	NaN	NaN	NaN	NaN	2807906

210720 rows × 14 columns

Data Cleaning and Data Preprocessing

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

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

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

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ MB
```

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

Out[6]:

	CO	station
6	0.2	28079024
30	0.2	28079024

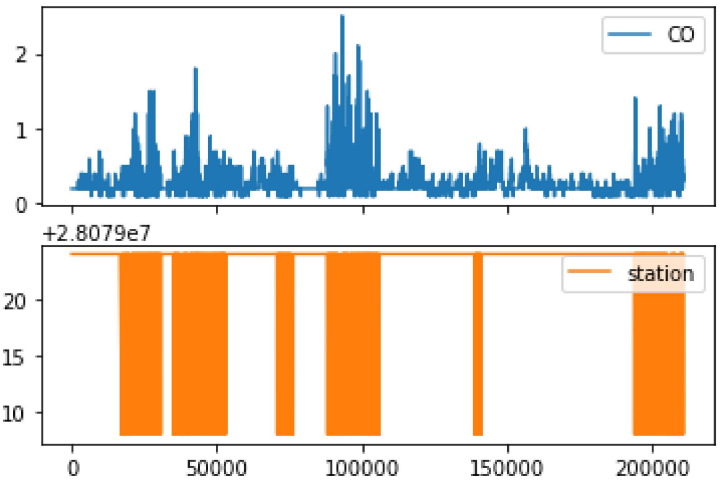
	CO	station
54	0.2	28079024
78	0.2	28079024
102	0.2	28079024
...	...	...
210654	0.3	28079024
210673	0.4	28079008
210678	0.3	28079024
210697	0.4	28079008
210702	0.3	28079024

10916 rows × 2 columns

Line chart

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

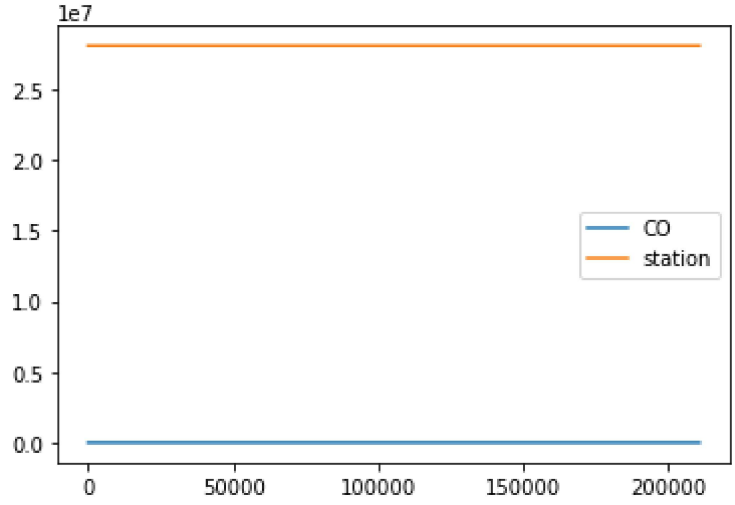
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

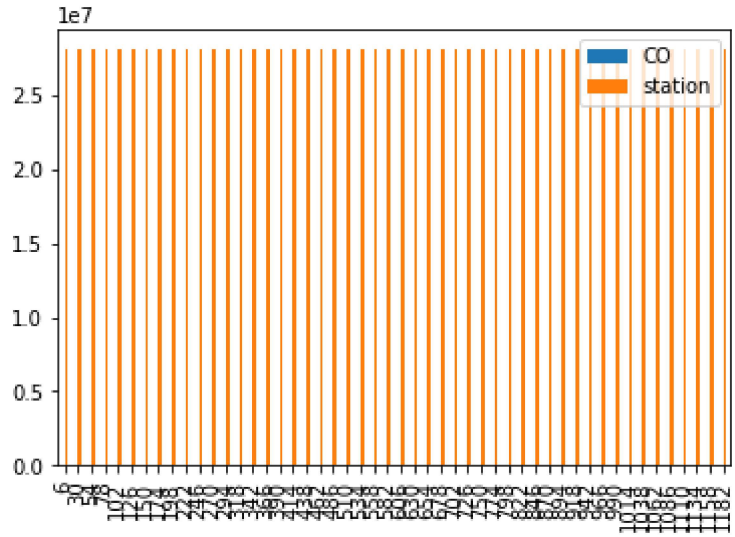


Bar chart

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

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

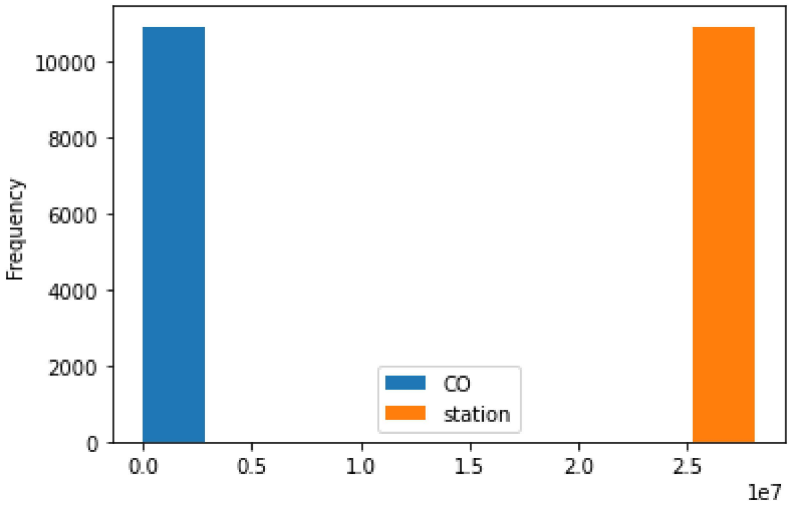
Out[10]: <AxesSubplot:>



Histogram

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

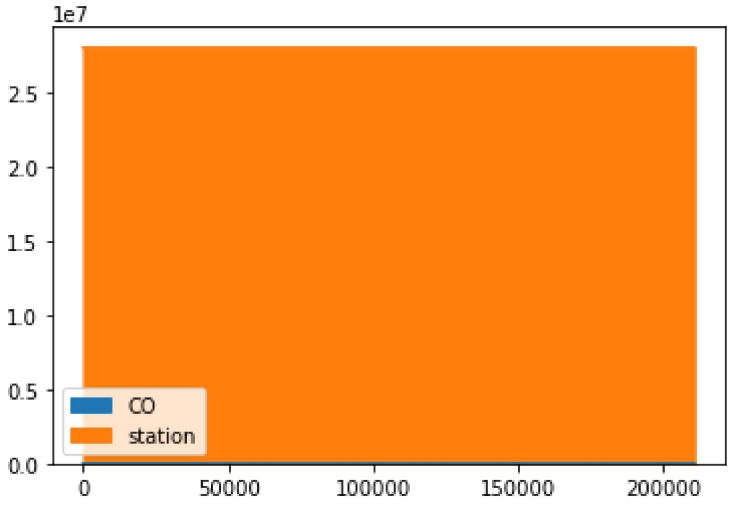
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

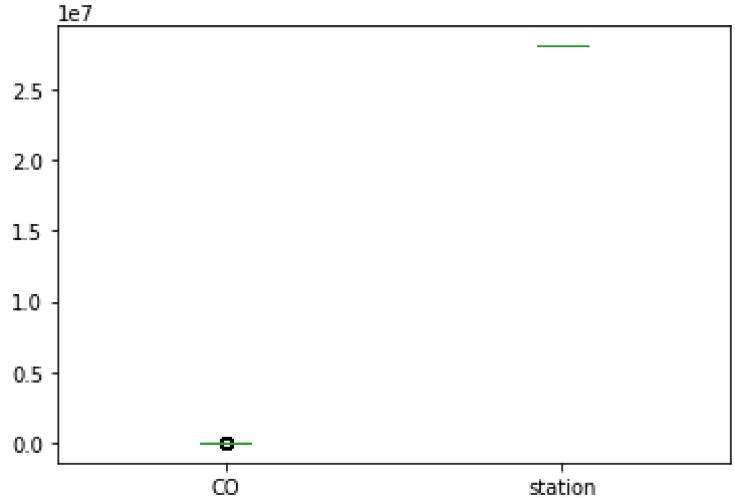
Out[12]: <AxesSubplot:>



Box chart

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

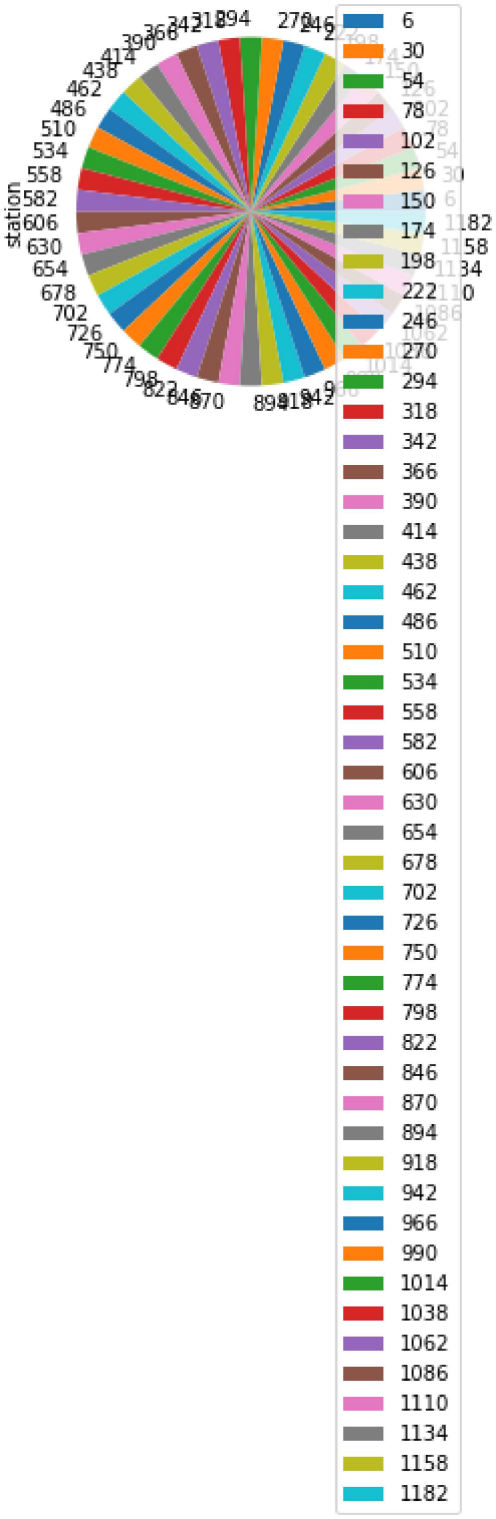
Out[13]: <AxesSubplot:>



Pie chart

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

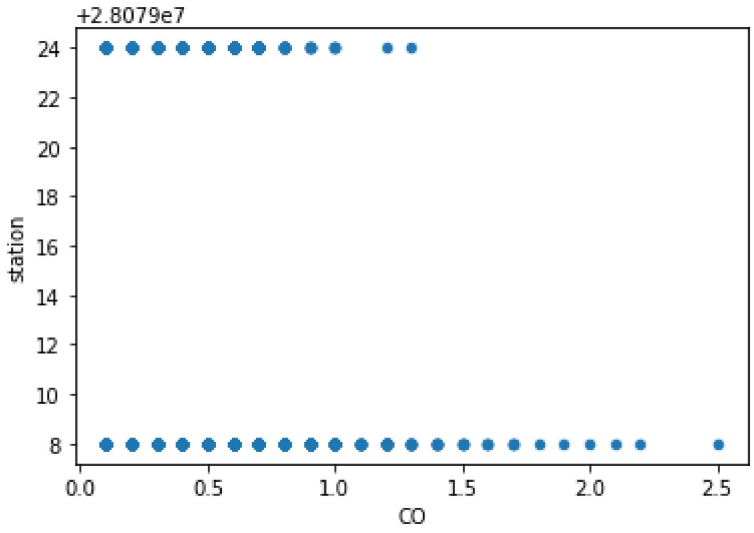
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ MB
```

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

Out[17]:

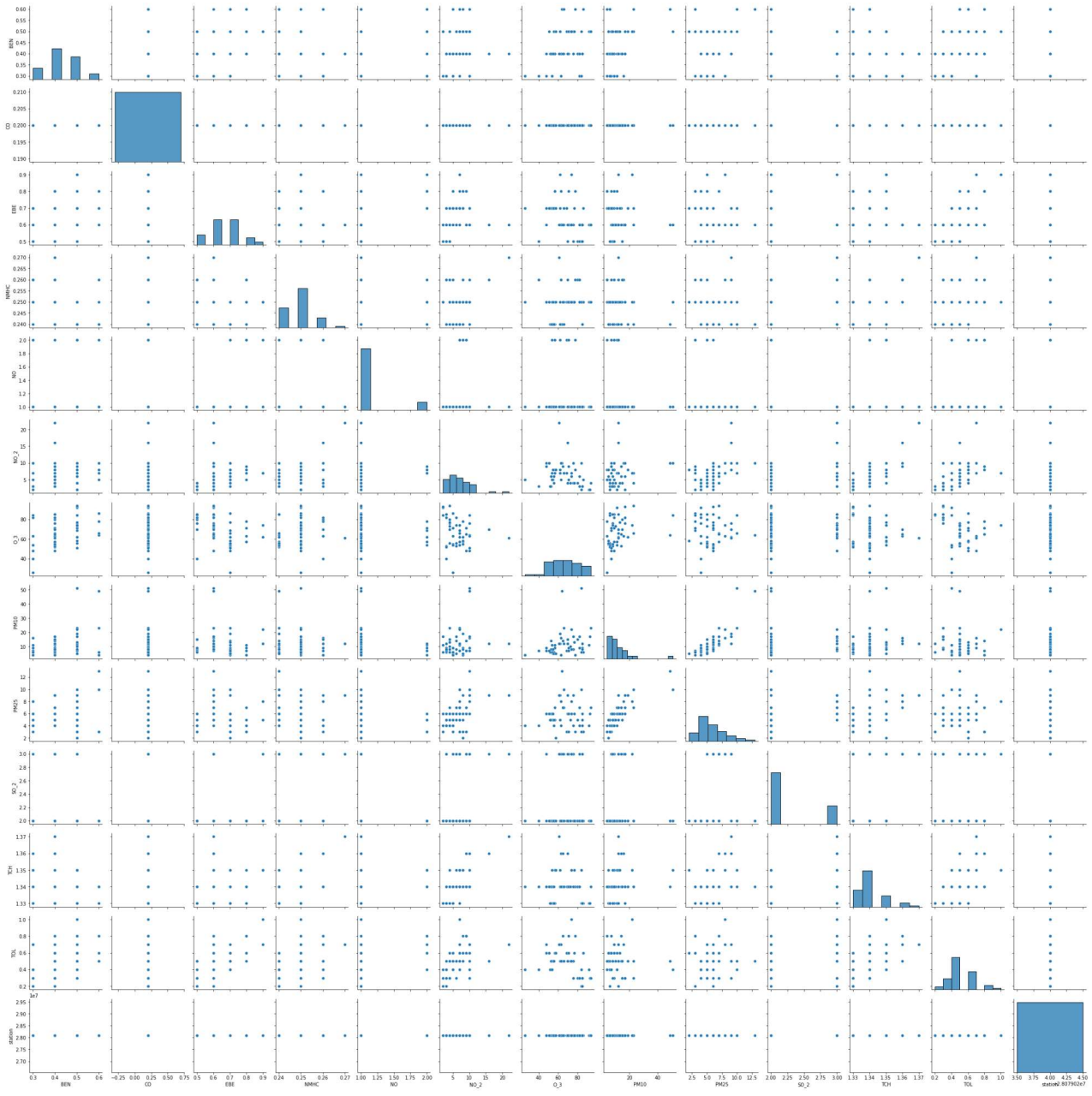
	BEN	CO	EBE	NMHC	NO	NO_2	O_3
count	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000
mean	0.784014	0.279333	0.992213	0.215755	18.795529	31.262642	44.23955
std	0.632755	0.167922	0.804554	0.075169	40.038872	27.234732	29.53556
min	0.100000	0.100000	0.100000	0.050000	0.000000	1.000000	1.000000
25%	0.400000	0.200000	0.500000	0.160000	1.000000	9.000000	18.000000
50%	0.600000	0.200000	0.800000	0.220000	3.000000	24.000000	44.000000
75%	0.900000	0.300000	1.200000	0.250000	18.000000	47.000000	65.000000

	BEN	CO	EBE	NMHC	NO	NO_2	O_3
max	7.000000	2.500000	9.700000	0.670000	525.000000	225.000000	157.00000

EDA AND VISUALIZATION

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

Out[18]: <seaborn.axisgrid.PairGrid at 0x26496cd2790>

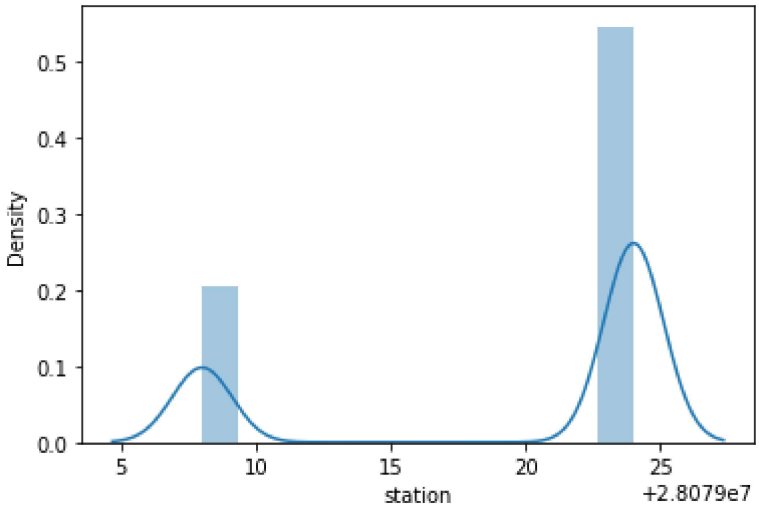


In [19]:
```
sns.distplot(df['station'])
```

```
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
```

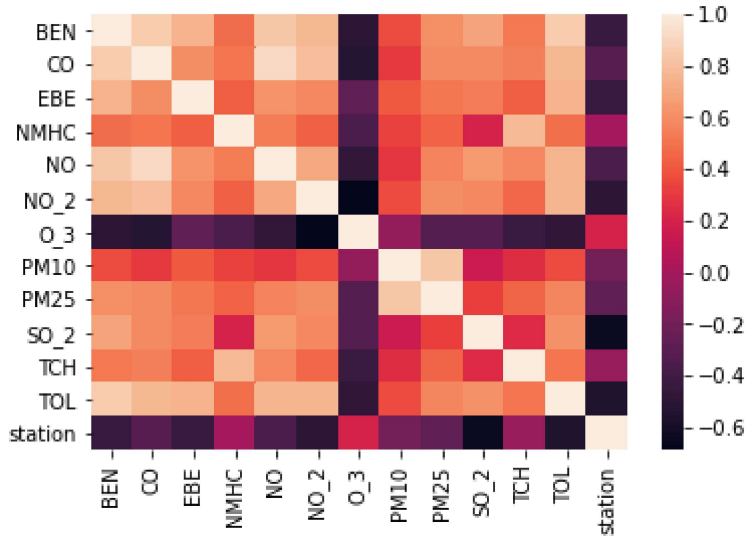
```
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

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]:
```
28079017.731460456
```

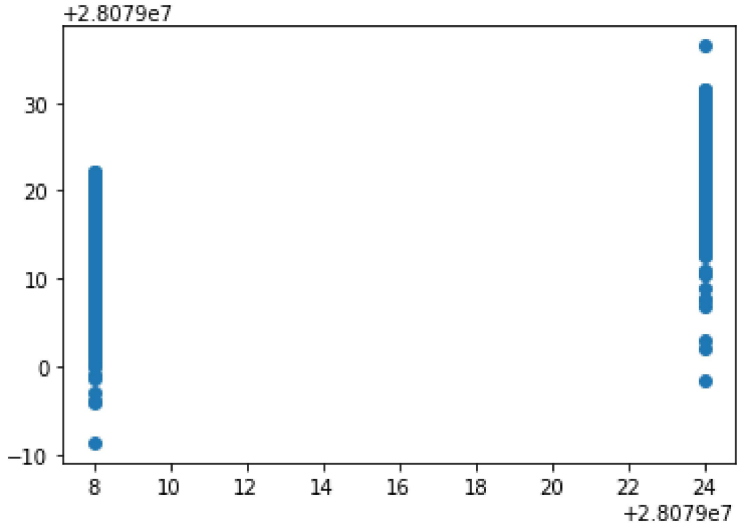
In [25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	3.853880
CO	22.100843
EBE	-0.487412
NMHC	17.444520
NO	-0.022994
NO_2	-0.127208
O_3	-0.031685
PM10	0.002833
PM25	-0.045314
SO_2	-0.674137
TCH	1.588794
TOL	-1.383102

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

Out[26]:
```
<matplotlib.collections.PathCollection at 0x264a1c19460>
```



ACCURACY

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

Out[27]: 0.6232492681080716

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

Out[28]: 0.6246877243192227

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]: 0.6199512231622704

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

Out[32]: 0.6204930833453566

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]:
```
0.3681684562128721
```

Accuracy(Lasso)

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

Out[35]:
```
0.36032335940494375
```

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.        ,  0.06749141,
       -0.09330053, -0.03899392, -0.        ,  0.0047338 , -0.69719731,
        0.        , -0.67547055])
```

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

Out[38]:
```
28079027.766384587
```

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

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

Out[40]:
```
0.5308256571193877
```

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)))
```

```
3.4683832535670915
23.292505075355432
4.8262309388751214
```

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]: (10916, 12)

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

Out[45]: (10916,)

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]: 0.9311102968120191

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

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

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

Out[54]:
```
array([[1.00000000e+00, 7.14919073e-33]])
```

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]:
```
0.9691135102101777
```

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

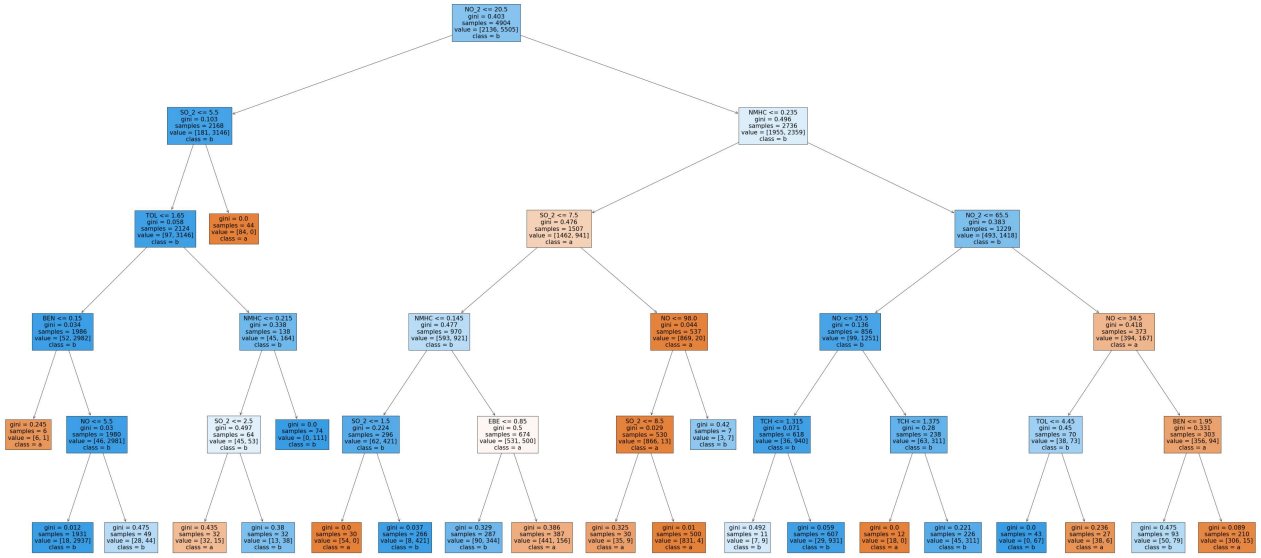
In [61]:
```
from sklearn.tree import plot_tree

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

Out[61]:
```
[Text(1734.3243243243244, 1993.2, 'NO_2 <= 20.5\ngini = 0.403\nsamples = 4904\nvalue =
[2136, 5505]\nclass = b'),
 Text(723.8918918918919, 1630.8000000000002, 'SO_2 <= 5.5\ngini = 0.103\nsamples = 2168
\nvalue = [181, 3146]\nclass = b'),
```

```
 Text(603.2432432432432, 1268.4, 'TOL <= 1.65\ngini = 0.058\nsamples = 2124\nvalue = [9
7, 3146]\nclass = b'),
 Text(241.2972972972973, 906.0, 'BEN <= 0.15\ngini = 0.034\nsamples = 1986\nvalue = [52,
2982]\nclass = b'),
 Text(120.64864864864865, 543.5999999999999, 'gini = 0.245\nsamples = 6\nvalue = [6, 1]
\nclass = a'),
 Text(361.94594594594594, 543.5999999999999, 'NO <= 5.5\ngini = 0.03\nsamples = 1980\nva
lue = [46, 2981]\nclass = b'),
 Text(241.2972972972973, 181.19999999999982, 'gini = 0.012\nsamples = 1931\nvalue = [18,
2937]\nclass = b'),
 Text(482.5945945945946, 181.19999999999982, 'gini = 0.475\nsamples = 49\nvalue = [28, 4
4]\nclass = b'),
 Text(965.1891891891892, 906.0, 'NMHC <= 0.215\ngini = 0.338\nsamples = 138\nvalue = [4
5, 164]\nclass = b'),
 Text(844.5405405405405, 543.5999999999999, 'SO_2 <= 2.5\ngini = 0.497\nsamples = 64\nva
lue = [45, 53]\nclass = b'),
 Text(723.8918918918919, 181.19999999999982, 'gini = 0.435\nsamples = 32\nvalue = [32, 1
5]\nclass = a'),
 Text(965.1891891891892, 181.19999999999982, 'gini = 0.38\nsamples = 32\nvalue = [13, 3
8]\nclass = b'),
 Text(1085.837837837838, 543.5999999999999, 'gini = 0.0\nsamples = 74\nvalue = [0, 111]
\nclass = b'),
 Text(844.5405405405405, 1268.4, 'gini = 0.0\nsamples = 44\nvalue = [84, 0]\nclass =
a'),
 Text(2744.7567567567567, 1630.8000000000002, 'NMHC <= 0.235\ngini = 0.496\nsamples = 27
36\nvalue = [1955, 2359]\nclass = b'),
 Text(1990.7027027027027, 1268.4, 'SO_2 <= 7.5\ngini = 0.476\nsamples = 1507\nvalue = [1
462, 941]\nclass = a'),
 Text(1568.4324324324325, 906.0, 'NMHC <= 0.145\ngini = 0.477\nsamples = 970\nvalue = [5
93, 921]\nclass = b'),
 Text(1327.135135135135, 543.5999999999999, 'SO_2 <= 1.5\ngini = 0.224\nsamples = 296\nv
alue = [62, 421]\nclass = b'),
 Text(1206.4864864864865, 181.19999999999982, 'gini = 0.0\nsamples = 30\nvalue = [54, 0]
\nclass = a'),
 Text(1447.7837837837837, 181.19999999999982, 'gini = 0.037\nsamples = 266\nvalue = [8,
421]\nclass = b'),
 Text(1809.7297297297296, 543.5999999999999, 'EBE <= 0.85\ngini = 0.5\nsamples = 674\nva
lue = [531, 500]\nclass = a'),
 Text(1689.081081081081, 181.19999999999982, 'gini = 0.329\nsamples = 287\nvalue = [90,
344]\nclass = b'),
 Text(1930.3783783783783, 181.19999999999982, 'gini = 0.386\nsamples = 387\nvalue = [44
1, 156]\nclass = a'),
 Text(2412.972972972973, 906.0, 'NO <= 98.0\ngini = 0.044\nsamples = 537\nvalue = [869,
20]\nclass = a'),
 Text(2292.324324324324, 543.5999999999999, 'SO_2 <= 8.5\ngini = 0.029\nsamples = 530\nv
alue = [866, 13]\nclass = a'),
 Text(2171.675675675676, 181.19999999999982, 'gini = 0.325\nsamples = 30\nvalue = [35,
9]\nclass = a'),
 Text(2412.972972972973, 181.19999999999982, 'gini = 0.01\nsamples = 500\nvalue = [831,
4]\nclass = a'),
 Text(2533.6216216216217, 543.5999999999999, 'gini = 0.42\nsamples = 7\nvalue = [3, 7]\n
class = b'),
 Text(3498.810810810811, 1268.4, 'NO_2 <= 65.5\ngini = 0.383\nsamples = 1229\nvalue = [4
93, 1418]\nclass = b'),
 Text(3016.2162162162163, 906.0, 'NO <= 25.5\ngini = 0.136\nsamples = 856\nvalue = [99,
1251]\nclass = b'),
 Text(2774.9189189189187, 543.5999999999999, 'TCH <= 1.315\ngini = 0.071\nsamples = 618
\nvalue = [36, 940]\nclass = b'),
 Text(2654.27027027027, 181.19999999999982, 'gini = 0.492\nsamples = 11\nvalue = [7, 9]
\nclass = b'),
 Text(2895.5675675675675, 181.19999999999982, 'gini = 0.059\nsamples = 607\nvalue = [29,
931]\nclass = b'),
 Text(3257.5135135135133, 543.5999999999999, 'TCH <= 1.375\ngini = 0.28\nsamples = 238\n
value = [63, 311]\nclass = b'),
 Text(3136.864864864865, 181.19999999999982, 'gini = 0.0\nsamples = 12\nvalue = [18, 0]
```

```
\nclass = a'),
 Text(3378.162162162162, 181.19999999999982, 'gini = 0.221\nsamples = 226\nvalue = [45,
311]\nclass = b'),
 Text(3981.4054054054054, 906.0, 'NO <= 34.5\ngini = 0.418\nsamples = 373\nvalue = [394,
167]\nclass = a'),
 Text(3740.108108108108, 543.5999999999999, 'TOL <= 4.45\ngini = 0.45\nsamples = 70\nval
ue = [38, 73]\nclass = b'),
 Text(3619.459459459459, 181.19999999999982, 'gini = 0.0\nsamples = 43\nvalue = [0, 67]
\nclass = b'),
 Text(3860.7567567567567, 181.19999999999982, 'gini = 0.236\nsamples = 27\nvalue = [38,
6]\nclass = a'),
 Text(4222.7027027027025, 543.5999999999999, 'BEN <= 1.95\ngini = 0.331\nsamples = 303\n
value = [356, 94]\nclass = a'),
 Text(4102.054054054054, 181.19999999999982, 'gini = 0.475\nsamples = 93\nvalue = [50, 7
9]\nclass = b'),
 Text(4343.351351351352, 181.19999999999982, 'gini = 0.089\nsamples = 210\nvalue = [306,
15]\nclass = a')]
```



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: 0.6232492681080716
Ridge Regression: 0.6199512231622704
Lasso Regression 0.36032335940494375
ElasticNet Regression: 0.5308256571193877
Logistic Regression: 0.9311102968120191
Random Forest: 0.9691135102101777
```

Random Forest is suitable for this dataset