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(r"C:\Users\user\Downloads\madrid_2009.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25
0	01-10-2009 01:00	NaN	0.27	NaN	NaN	NaN	39.889999	48.150002	NaN	50.680000	18.260000	NaN
1	01-10-2009 01:00	NaN	0.22	NaN	NaN	NaN	21.230000	24.260000	NaN	55.880001	10.580000	NaN
2	01-10-2009 01:00	NaN	0.18	NaN	NaN	NaN	31.230000	34.880001	NaN	49.060001	25.190001	NaN
3	01-10-2009 01:00	0.95	0.33	1.43	2.68	0.25	55.180000	81.360001	1.57	36.669998	26.530001	6.82
4	01-10-2009 01:00	NaN	0.41	NaN	NaN	0.12	61.349998	76.260002	NaN	38.090000	23.760000	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
215683	01-06-2009 00:00	0.50	0.22	0.39	0.75	0.09	22.000000	24.510000	1.00	82.239998	10.830000	7.15
215684	01-06-2009 00:00	NaN	0.31	NaN	NaN	NaN	76.110001	101.099998	NaN	41.220001	9.920000	NaN
215685	01-06-2009 00:00	0.13	NaN	0.86	NaN	0.23	81.050003	99.849998	NaN	24.830000	12.460000	6.77

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25
215686	01-06-2009 00:00	0.21	NaN	2.96	NaN	0.10	72.419998	82.959999	NaN	NaN	13.030000	NaN
215687	01-06-2009 00:00	0.37	0.32	0.99	1.36	0.14	54.290001	64.480003	1.06	56.919998	15.360000	11.6

215688 rows × 17 columns

Data Cleaning and Data Preprocessing

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

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

Out[4]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24717 non-null  object 
 1   BEN      24717 non-null  float64
 2   CO       24717 non-null  float64
 3   EBE      24717 non-null  float64
 4   MXY      24717 non-null  float64
 5   NMHC     24717 non-null  float64
 6   NO_2     24717 non-null  float64
 7   NOx      24717 non-null  float64
 8   OXY      24717 non-null  float64
 9   O_3      24717 non-null  float64
 10  PM10     24717 non-null  float64
 11  PM25     24717 non-null  float64
 12  PXY      24717 non-null  float64
 13  SO_2     24717 non-null  float64
 14  TCH      24717 non-null  float64
 15  TOL      24717 non-null  float64
 16  station  24717 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

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

Out[6]:

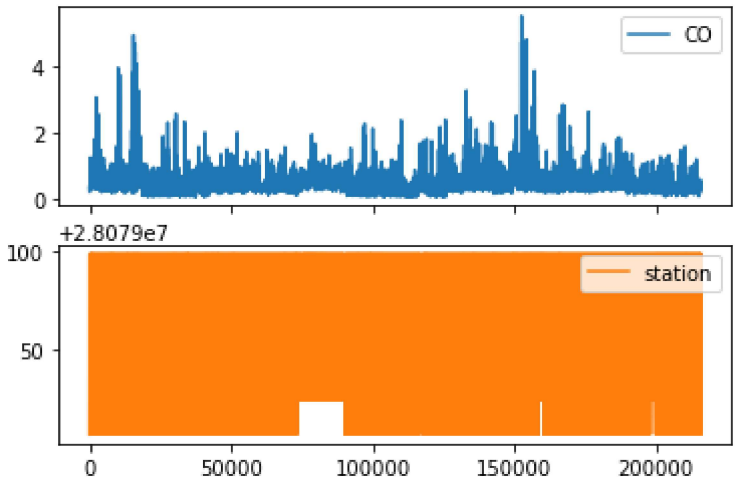
	CO	station
3	0.33	28079006
20	0.32	28079024
24	0.24	28079099
28	0.21	28079006
45	0.30	28079024
...	...	...
215659	0.27	28079024
215663	0.35	28079099
215667	0.29	28079006
215683	0.22	28079024
215687	0.32	28079099

24717 rows × 2 columns

Line chart

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

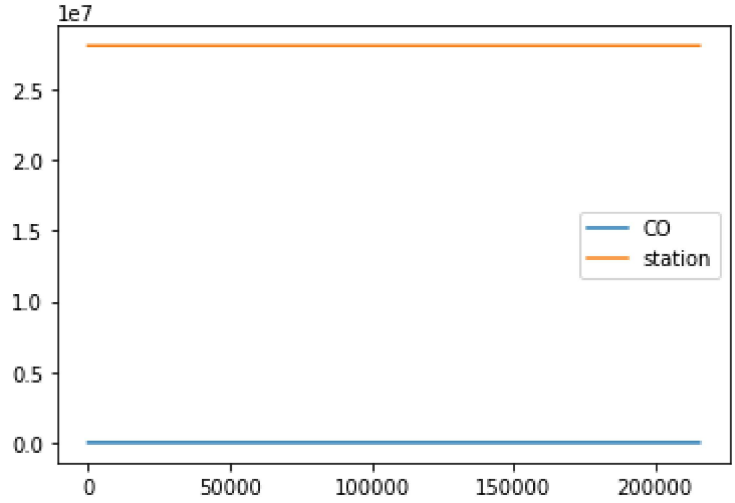
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

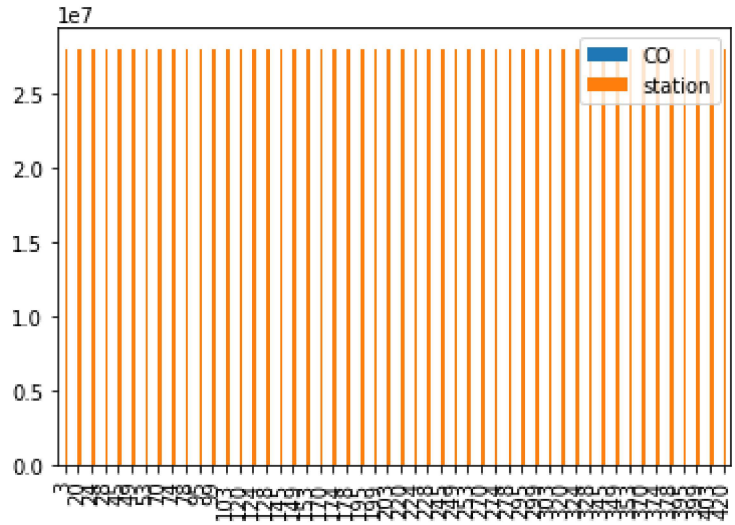


Bar chart

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

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

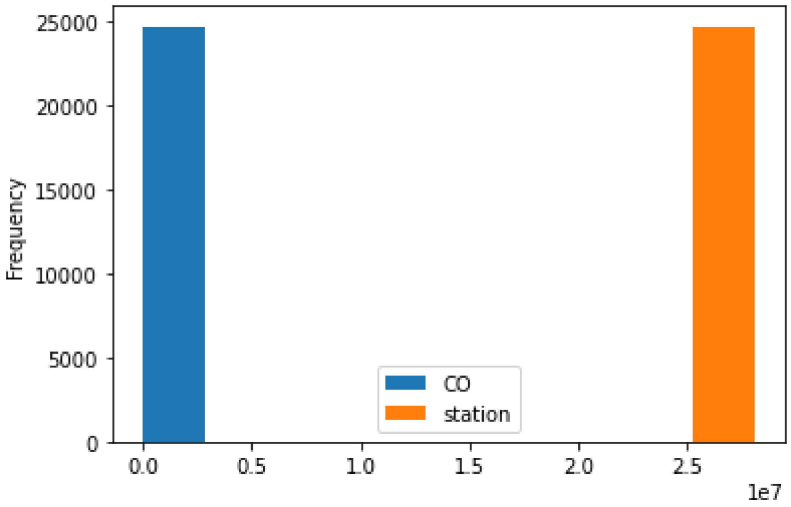
Out[10]: <AxesSubplot:>



Histogram

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

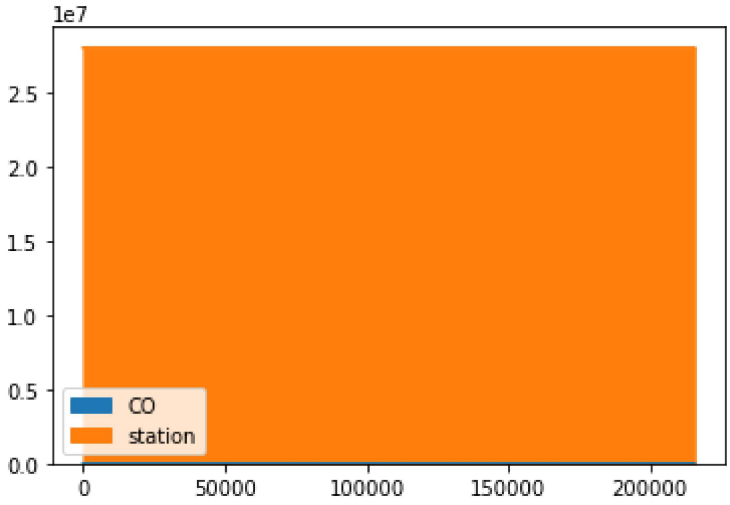
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

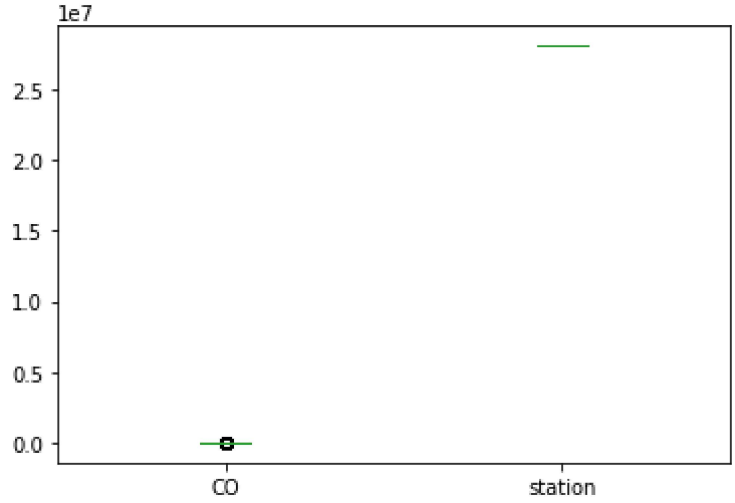
Out[12]: <AxesSubplot:>



Box chart

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

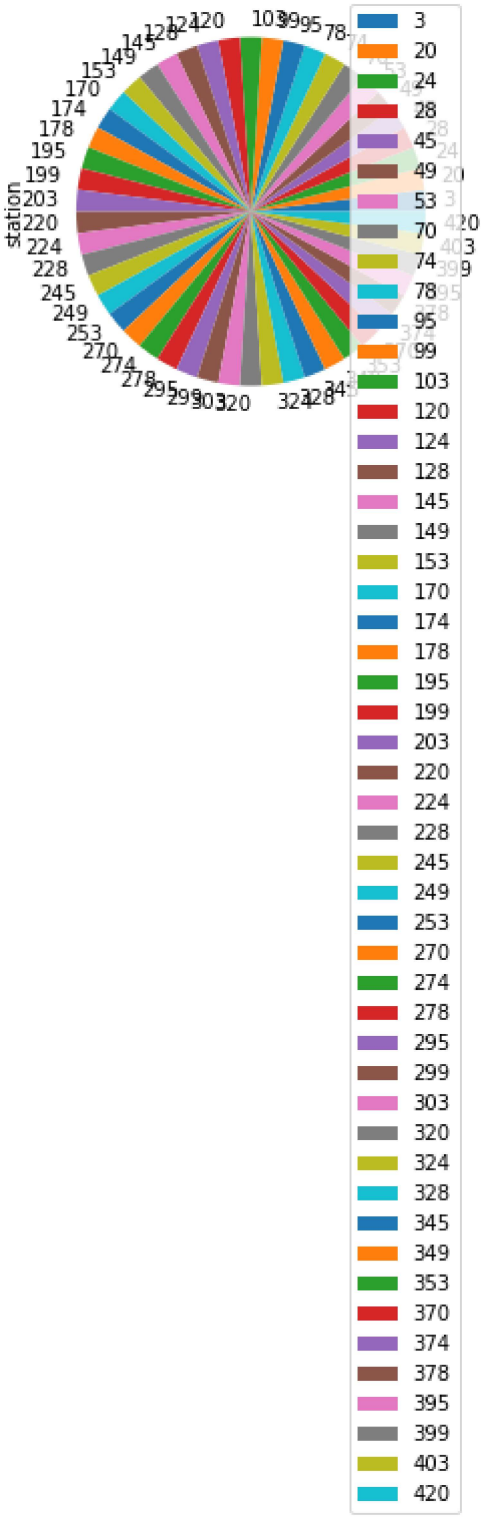
Out[13]: <AxesSubplot:>



Pie chart

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

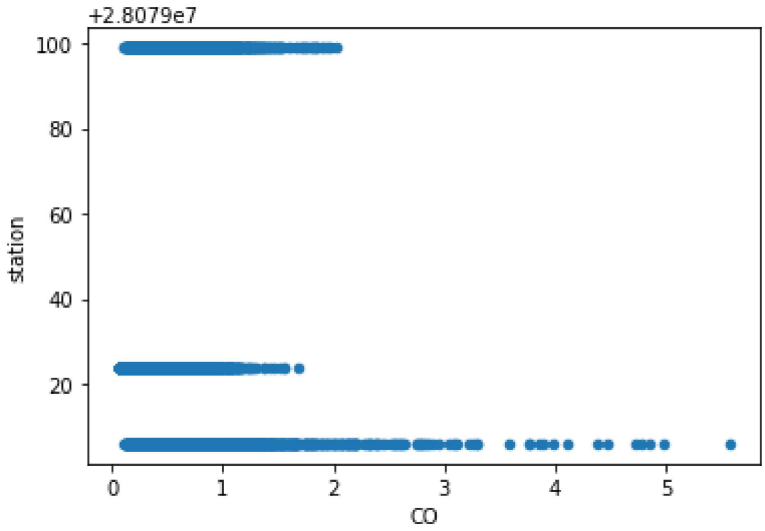
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24717 non-null  object 
 1   BEN      24717 non-null  float64
 2   CO       24717 non-null  float64
 3   EBE      24717 non-null  float64
 4   MXY      24717 non-null  float64
 5   NMHC     24717 non-null  float64
 6   NO_2     24717 non-null  float64
 7   NOx      24717 non-null  float64
 8   OXY      24717 non-null  float64
 9   O_3      24717 non-null  float64
 10  PM10     24717 non-null  float64
 11  PM25     24717 non-null  float64
 12  PXY      24717 non-null  float64
 13  SO_2     24717 non-null  float64
 14  TCH      24717 non-null  float64
 15  TOL      24717 non-null  float64
 16  station  24717 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

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

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NO
count	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	24717.00000
mean	1.010583	0.448056	1.262430	2.244469	0.219582	55.563929	92.90718
std	1.007345	0.291706	1.074768	2.242214	0.141661	38.911677	91.98535
min	0.170000	0.060000	0.250000	0.240000	0.000000	0.600000	2.25000
25%	0.460000	0.270000	0.720000	0.990000	0.140000	26.510000	33.00999
50%	0.670000	0.370000	1.000000	1.490000	0.190000	47.930000	67.01000

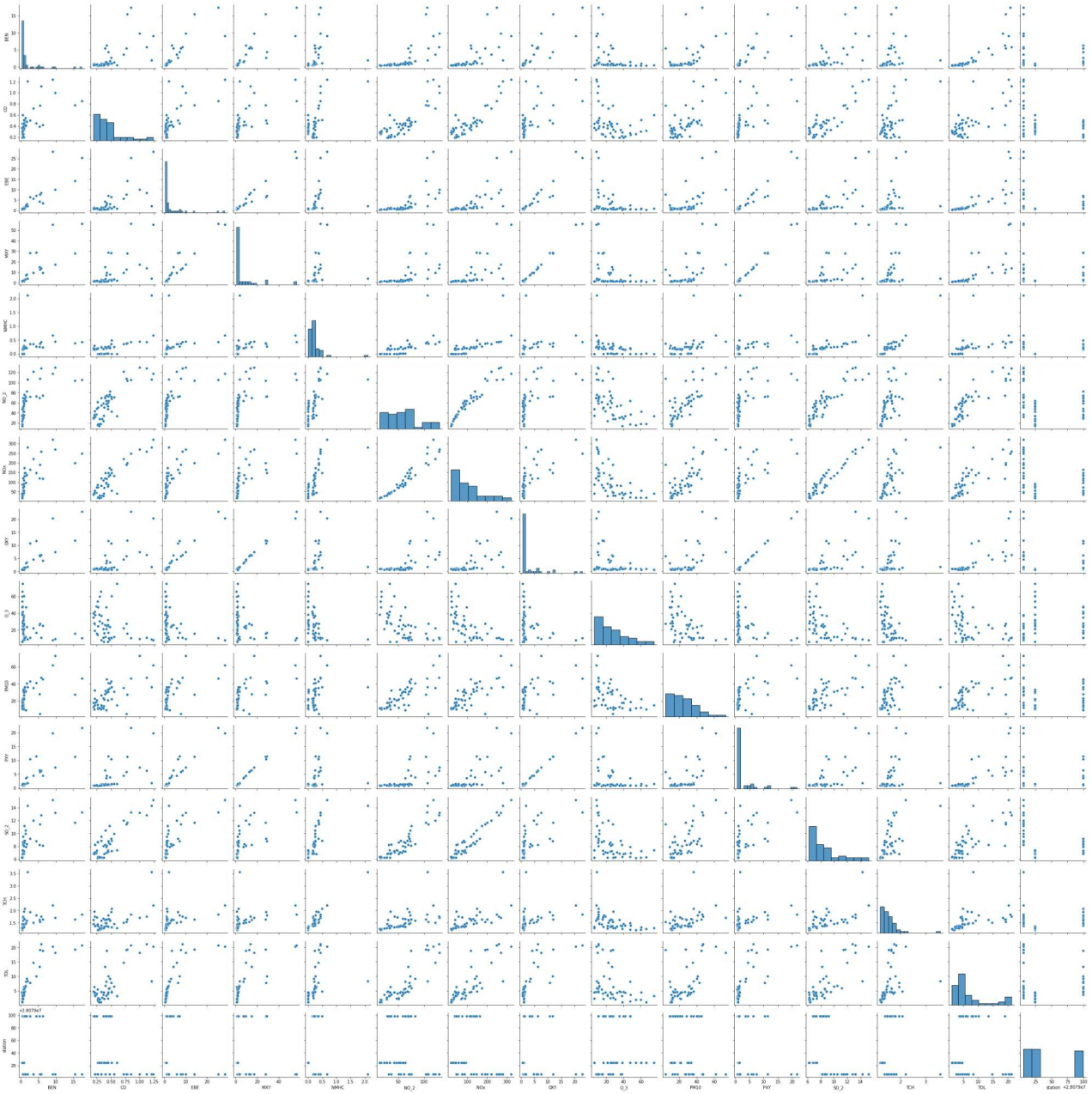
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
75%	1.180000	0.570000	1.430000	2.820000	0.260000	76.269997	124.69999
max	22.379999	5.570000	47.669998	56.500000	2.580000	477.399994	1438.00000

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

EDA AND VISUALIZATION

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

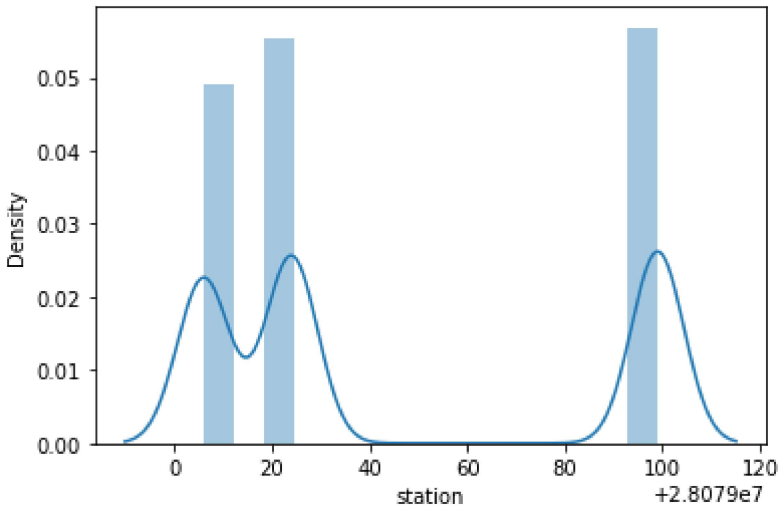
Out[19]: <seaborn.axisgrid.PairGrid at 0x206770ee550>



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

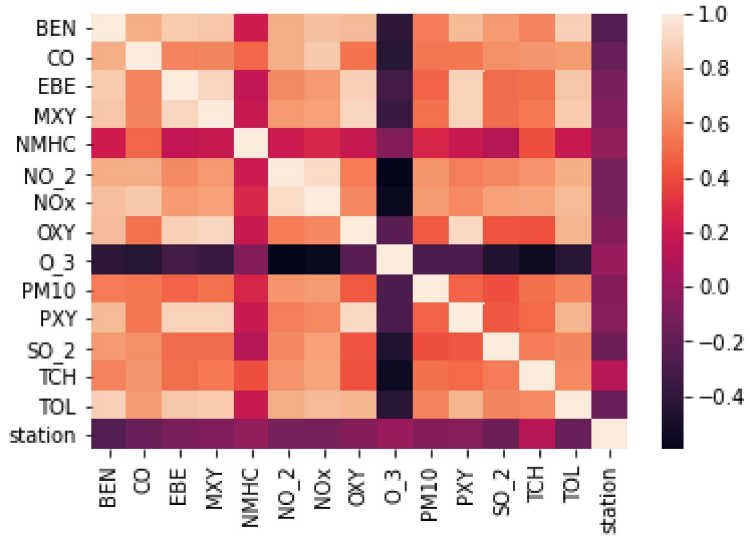
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function 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', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', '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]:
```
28078899.77857116
```

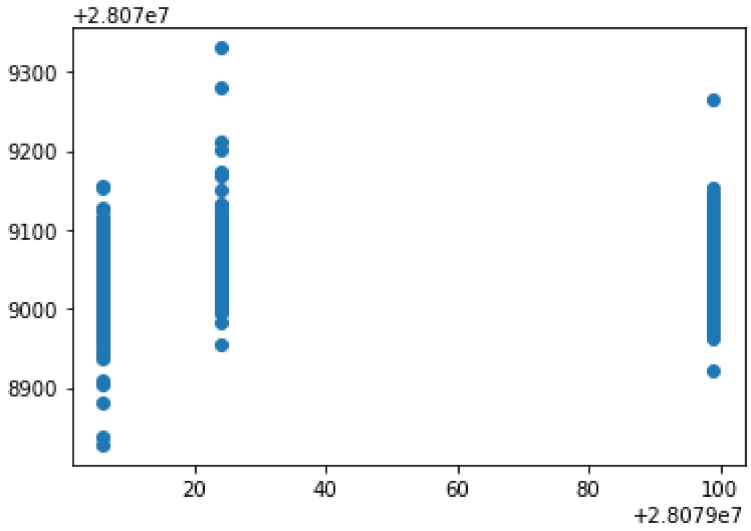
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-34.777298
CO	-29.277698
EBE	5.234704
MXY	-0.564850
NMHC	-18.299222
NO_2	-0.174441
NOx	0.202177
OXY	13.039875
O_3	0.019674
PM10	-0.050444
PXY	2.779321
SO_2	-0.331558
TCH	119.587582
TOL	-1.159765

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x2060626c0a0>
```



ACCURACY

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

Out[28]: 0.29531586104927077

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

Out[29]: 0.2837197692763801

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]: 0.29518568539614765

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

Out[33]: 0.2834080618622108

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]:
```
0.03586873067275287
```

Accuracy(Lasso)

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

Out[36]:
```
0.0375592043443409
```

Elastic Net regression

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([-6.94976412, -0.67049718,  0.34248156,  2.14969383, -0.        ,
       -0.2330618 ,  0.13271996,  1.1337234 , -0.14789524,  0.07863545,
        1.91180828, -0.74941148,  1.49787945, -2.02567564])
```

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

Out[39]:
```
28079063.53660987
```

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

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

Out[41]:
```
0.10867786906519561
```

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)))
```

```
36.01575405326016
1477.5335903680946
38.438699124295226
```

Logistic Regression

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

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

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

Out[45]: (24717, 14)

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

Out[46]: (24717,)

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=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

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

```
[28079099]
```

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

Out[52]: array([28079006, 28079024, 28079099], dtype=int64)

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

Out[53]:
```
0.8951733624630821
```

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

Out[54]:
```
5.433879549400576e-13
```

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

Out[55]:
```
array([[5.43387955e-13, 8.29911869e-44, 1.00000000e+00]])
```

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="accuracy")
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]:
```
0.8971736636414114
```

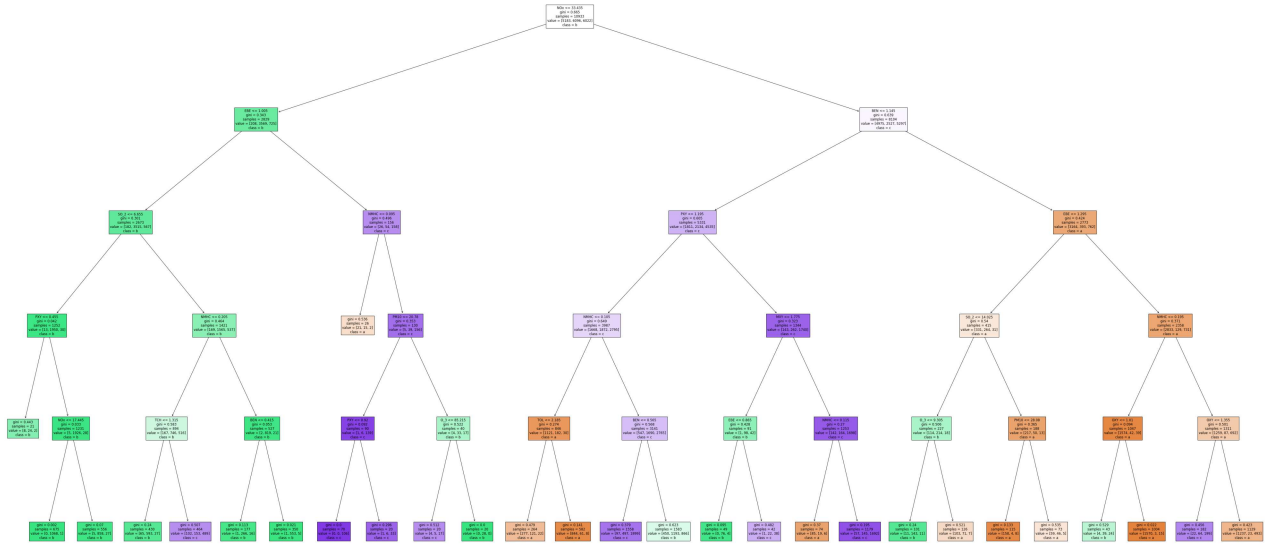
In [61]:
```
rfc_best=grid_search.best_estimator_
```

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','c','d']
```

Out[62]: [Text(2010.9056603773586, 1993.2, 'NOx <= 33.435\ngini = 0.665\nsamples = 10933\nvalue = [5183, 6096, 6022]\nclass = b'),
 Text(905.433962264151, 1630.8000000000002, 'EBE <= 1.005\ngini = 0.343\nsamples = 2829\nvalue = [208, 3569, 725]\nclass = b'),
 Text(463.24528301886795, 1268.4, 'SO_2 <= 6.655\ngini = 0.301\nsamples = 2673\nvalue = [182, 3515, 567]\nclass = b'),
 Text(168.45283018867926, 906.0, 'PXY <= 0.455\ngini = 0.042\nsamples = 1252\nvalue = [13, 1950, 30]\nclass = b'),
 Text(84.22641509433963, 543.5999999999999, 'gini = 0.443\nsamples = 21\nvalue = [8, 24, 2]\nclass = b'),
 Text(252.67924528301887, 543.5999999999999, 'NOx <= 17.445\ngini = 0.033\nsamples = 1231\nvalue = [5, 1926, 28]\nclass = b'),
 Text(168.45283018867926, 181.19999999999982, 'gini = 0.002\nsamples = 675\nvalue = [0, 1068, 1]\nclass = b'),
 Text(336.9056603773585, 181.19999999999982, 'gini = 0.07\nsamples = 556\nvalue = [5, 858, 27]\nclass = b'),
 Text(758.0377358490566, 906.0, 'NMHC <= 0.205\ngini = 0.464\nsamples = 1421\nvalue = [169, 1565, 537]\nclass = b'),
 Text(589.5849056603774, 543.5999999999999, 'TCH <= 1.315\ngini = 0.583\nsamples = 894\nvalue = [167, 746, 516]\nclass = b'),
 Text(505.35849056603774, 181.19999999999982, 'gini = 0.24\nsamples = 430\nvalue = [65, 593, 27]\nclass = b'),
 Text(673.811320754717, 181.19999999999982, 'gini = 0.507\nsamples = 464\nvalue = [102, 153, 489]\nclass = c'),
 Text(926.4905660377359, 543.5999999999999, 'BEN <= 0.415\ngini = 0.053\nsamples = 527\nvalue = [2, 819, 21]\nclass = b'),
 Text(842.2641509433963, 181.19999999999982, 'gini = 0.113\nsamples = 177\nvalue = [1, 266, 16]\nclass = b'),
 Text(1010.7169811320755, 181.19999999999982, 'gini = 0.021\nsamples = 350\nvalue = [1, 553, 5]\nclass = b'),
 Text(1347.622641509434, 1268.4, 'NMHC <= 0.095\ngini = 0.496\nsamples = 156\nvalue = [26, 54, 158]\nclass = c'),
 Text(1263.3962264150944, 906.0, 'gini = 0.536\nsamples = 26\nvalue = [21, 15, 2]\nclass = a'),
 Text(1431.8490566037738, 906.0, 'PM10 <= 20.78\ngini = 0.353\nsamples = 130\nvalue = [5, 39, 156]\nclass = c'),
 Text(1263.3962264150944, 543.5999999999999, 'PXY <= 0.92\ngini = 0.092\nsamples = 90\nvalue = [1, 6, 139]\nclass = c'),
 Text(1179.169811320755, 181.19999999999982, 'gini = 0.0\nsamples = 70\nvalue = [0, 0, 106]\nclass = c'),
 Text(1347.622641509434, 181.19999999999982, 'gini = 0.296\nsamples = 20\nvalue = [1, 6, 33]\nclass = c'),
 Text(1600.301886792453, 543.5999999999999, 'O_3 <= 85.215\ngini = 0.522\nsamples = 40\nvalue = [4, 33, 17]\nclass = b'),
 Text(1516.0754716981132, 181.19999999999982, 'gini = 0.512\nsamples = 20\nvalue = [4, 5, 17]\nclass = c'),
 Text(1684.5283018867926, 181.19999999999982, 'gini = 0.0\nsamples = 20\nvalue = [0, 28, 0]\nclass = b'),
 Text(3116.377358490566, 1630.8000000000002, 'BEN <= 1.145\ngini = 0.639\nsamples = 8104\nvalue = [4975, 2527, 5297]\nclass = c'),
 Text(2442.566037735849, 1268.4, 'PXY <= 1.195\ngini = 0.605\nsamples = 5331\nvalue = [1811, 2134, 4535]\nclass = c'),
 Text(2105.6603773584907, 906.0, 'NMHC <= 0.105\ngini = 0.649\nsamples = 3987\nvalue = [1668, 1872, 2795]\nclass = c'),
 Text(1937.2075471698115, 543.5999999999999, 'TOL <= 2.185\ngini = 0.274\nsamples = 846\nvalue = [1121, 182, 30]\nclass = a'),
 Text(1852.9811320754718, 181.19999999999982, 'gini = 0.479\nsamples = 264\nvalue = [277, 121, 22]\nclass = a'),
 Text(2021.433962264151, 181.19999999999982, 'gini = 0.141\nsamples = 582\nvalue = [844, 61, 8]\nclass = a'),
 Text(2274.11320754717, 543.5999999999999, 'BEN <= 0.565\ngini = 0.568\nsamples = 3141\nvalue = [547, 1690, 2765]\nclass = c'),
 Text(2189.8867924528304, 181.19999999999982, 'gini = 0.379\nsamples = 1558\nvalue = [97, 497, 1899]\nclass = c'),
 Text(2358.33962264151, 181.19999999999982, 'gini = 0.623\nsamples = 1583\nvalue = [450,

```
1193, 866]\nclass = b'),
 Text(2779.471698113208, 906.0, 'MXY <= 1.775\ngini = 0.323\nsamples = 1344\nvalue = [14
3, 262, 1740]\nclass = c'),
 Text(2611.0188679245284, 543.5999999999999, 'EBE <= 0.865\ngini = 0.428\nsamples = 91\n
value = [1, 98, 42]\nclass = b'),
 Text(2526.7924528301887, 181.19999999999982, 'gini = 0.095\nsamples = 49\nvalue = [0, 7
6, 4]\nclass = b'),
 Text(2695.245283018868, 181.19999999999982, 'gini = 0.482\nsamples = 42\nvalue = [1, 2
2, 38]\nclass = c'),
 Text(2947.924528301887, 543.5999999999999, 'NMHC <= 0.115\ngini = 0.27\nsamples = 1253
\nvalue = [142, 164, 1698]\nclass = c'),
 Text(2863.6981132075475, 181.19999999999982, 'gini = 0.37\nsamples = 74\nvalue = [85, 1
9, 6]\nclass = a'),
 Text(3032.1509433962265, 181.19999999999982, 'gini = 0.195\nsamples = 1179\nvalue = [5
7, 145, 1692]\nclass = c'),
 Text(3790.1886792452833, 1268.4, 'EBE <= 1.295\ngini = 0.424\nsamples = 2773\nvalue =
[3164, 393, 762]\nclass = a'),
 Text(3453.283018867925, 906.0, 'SO_2 <= 14.025\ngini = 0.54\nsamples = 415\nvalue = [33
1, 264, 31]\nclass = a'),
 Text(3284.8301886792456, 543.5999999999999, 'O_3 <= 9.305\ngini = 0.506\nsamples = 227
\nvalue = [114, 214, 18]\nclass = b'),
 Text(3200.603773584906, 181.19999999999982, 'gini = 0.24\nsamples = 101\nvalue = [11, 1
43, 11]\nclass = b'),
 Text(3369.0566037735853, 181.19999999999982, 'gini = 0.521\nsamples = 126\nvalue = [10
3, 71, 7]\nclass = a'),
 Text(3621.735849056604, 543.5999999999999, 'PM10 <= 28.08\ngini = 0.365\nsamples = 188
\nvalue = [217, 50, 13]\nclass = a'),
 Text(3537.509433962264, 181.19999999999982, 'gini = 0.133\nsamples = 115\nvalue = [158,
4, 8]\nclass = a'),
 Text(3705.9622641509436, 181.19999999999982, 'gini = 0.535\nsamples = 73\nvalue = [59,
46, 5]\nclass = a'),
 Text(4127.094339622642, 906.0, 'NMHC <= 0.195\ngini = 0.371\nsamples = 2358\nvalue = [2
833, 129, 731]\nclass = a'),
 Text(3958.6415094339627, 543.5999999999999, 'OXY <= 1.01\ngini = 0.094\nsamples = 1047
\nvalue = [1574, 42, 39]\nclass = a'),
 Text(3874.415094339623, 181.19999999999982, 'gini = 0.529\nsamples = 43\nvalue = [4, 3
9, 24]\nclass = b'),
 Text(4042.867924528302, 181.19999999999982, 'gini = 0.022\nsamples = 1004\nvalue = [157
0, 3, 15]\nclass = a'),
 Text(4295.5471698113215, 543.5999999999999, 'OXY <= 1.355\ngini = 0.501\nsamples = 1311
\nvalue = [1259, 87, 692]\nclass = a'),
 Text(4211.320754716981, 181.19999999999982, 'gini = 0.456\nsamples = 182\nvalue = [22,
64, 199]\nclass = c'),
 Text(4379.773584905661, 181.19999999999982, 'gini = 0.423\nsamples = 1129\nvalue = [123
7, 23, 493]\nclass = a')]
```



Conclusion

Accuracy

linear regression

In [63]:
```
lr.score(x_test,y_test)
```

Out[63]: 0.29531586104927077

Ridge regression

In [64]:
```
rr.score(x_test,y_test)
```

Out[64]: 0.29518568539614765

Lasso regression

In [65]:
```
la.score(x_test,y_test)
```

Out[65]: 0.0375592043443409

Elastic net regression

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

Out[66]: 0.10867786906519561

Logistic regression

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

Out[67]: 0.8951733624630821

Random forest

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

Out[68]: 0.8971736636414114

Accuracy for random forest is higher so it is the best fit model