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("2009.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2009-10-01 01:00:00	NaN	0.27	NaN	NaN	NaN	39.889999	48.150002	NaN	50.680000	18.
1	2009-10-01 01:00:00	NaN	0.22	NaN	NaN	NaN	21.230000	24.260000	NaN	55.880001	10.
2	2009-10-01 01:00:00	NaN	0.18	NaN	NaN	NaN	31.230000	34.880001	NaN	49.060001	25.
3	2009-10-01 01:00:00	0.95	0.33	1.43	2.68	0.25	55.180000	81.360001	1.57	36.669998	26.
4	2009-10-01 01:00:00	NaN	0.41	NaN	NaN	0.12	61.349998	76.260002	NaN	38.090000	23.
...	...	...	...	...	...	...	...	...	...	...	
215683	2009-06-01 00:00:00	0.50	0.22	0.39	0.75	0.09	22.000000	24.510000	1.00	82.239998	10.
215684	2009-06-01 00:00:00	NaN	0.31	NaN	NaN	NaN	76.110001	101.099998	NaN	41.220001	9.
215685	2009-06-01 00:00:00	0.13	NaN	0.86	NaN	0.23	81.050003	99.849998	NaN	24.830000	12.
215686	2009-06-01 00:00:00	0.21	NaN	2.96	NaN	0.10	72.419998	82.959999	NaN	NaN	13.
215687	2009-06-01 00:00:00	0.37	0.32	0.99	1.36	0.14	54.290001	64.480003	1.06	56.919998	15.

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

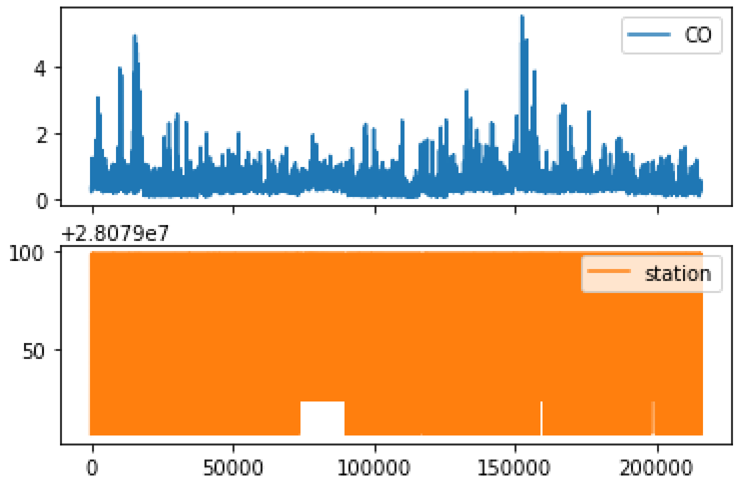
Line chart

In [7]:

```
data.plot.line(subplots=True)
```

Out[7]:

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



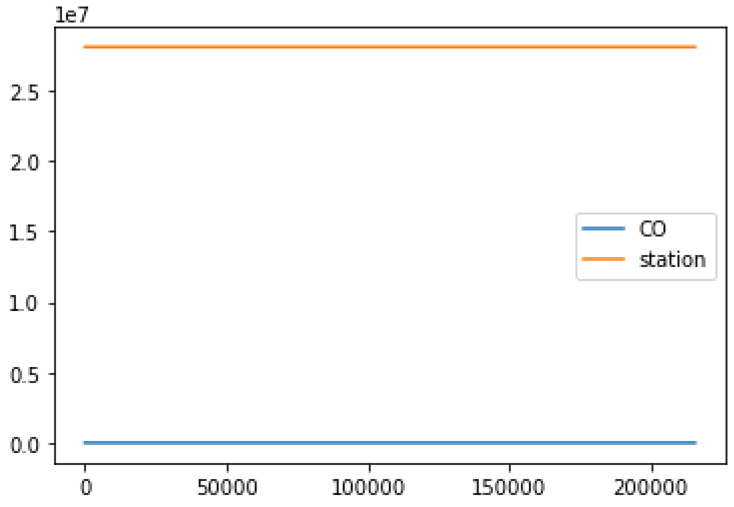
Line chart

In [8]:

```
data.plot.line()
```

Out[8]:

<AxesSubplot:>



Bar chart

In [9]:

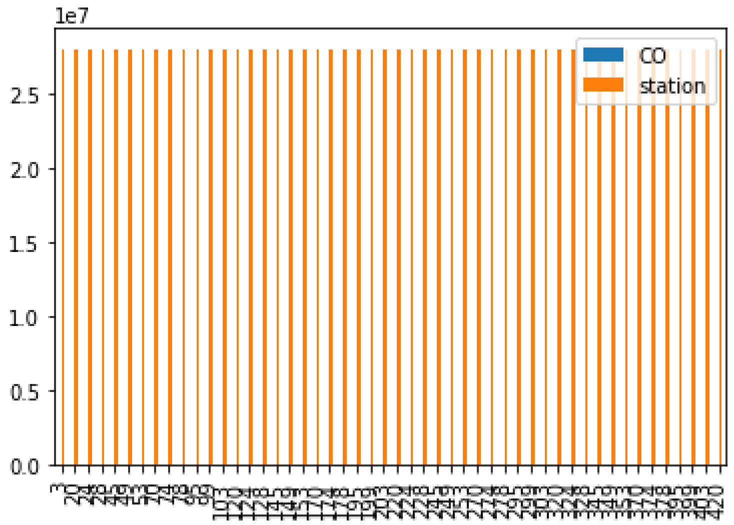
```
b=data[0:50]
```

In [10]:

```
b.plot.bar()
```

Out[10]:

<AxesSubplot:>



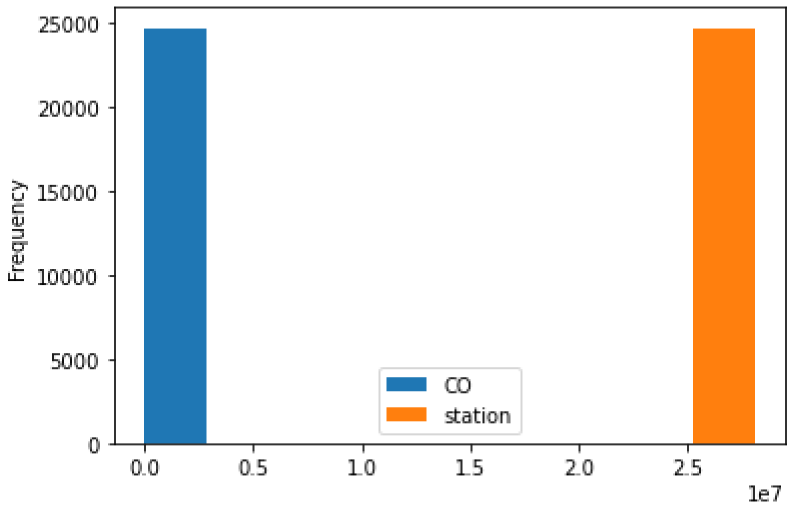
Histogram

In [11]:

```
data.plot.hist()
```

Out[11]:

```
<AxesSubplot:ylabel='Frequency'>
```



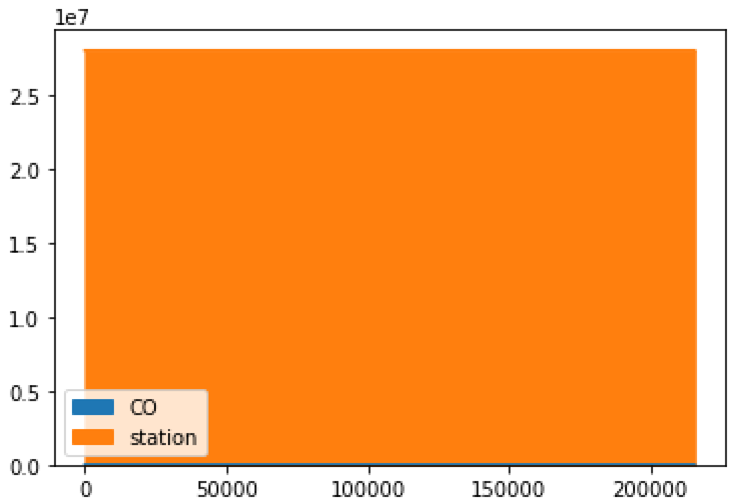
Area chart

In [12]:

```
data.plot.area()
```

Out[12]:

```
<AxesSubplot:>
```



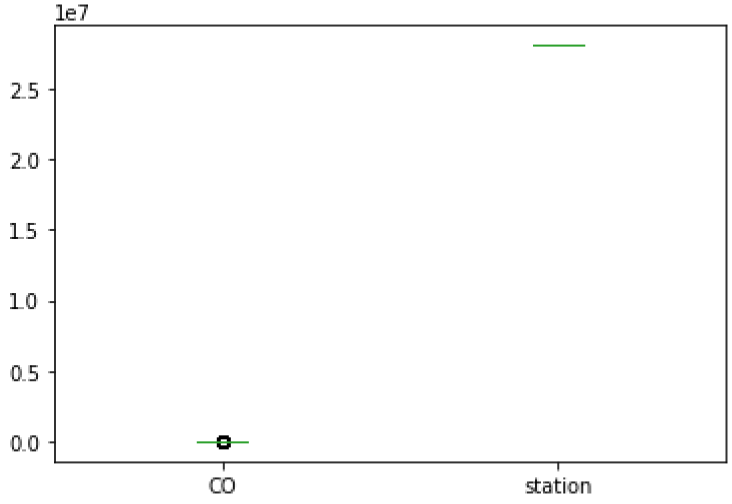
Box chart

In [13]:

```
data.plot.box()
```

Out[13]:

<AxesSubplot:>



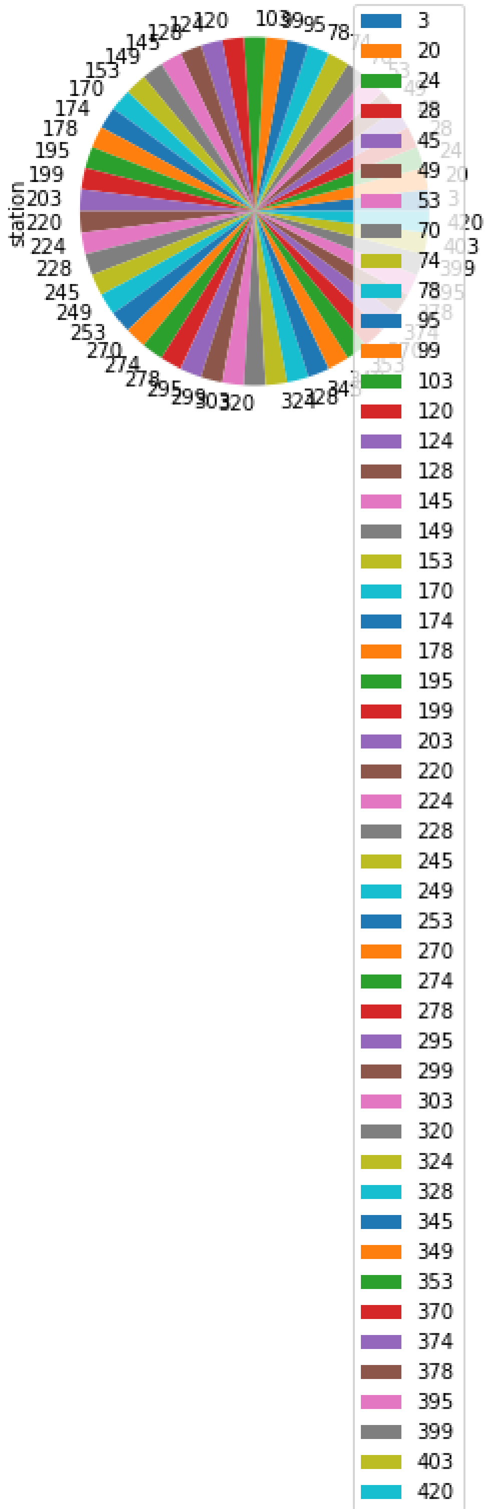
Pie chart

In [14]:

```
b.plot.pie(y='station' )
```

Out[14]:

<AxesSubplot:ylabel='station'>



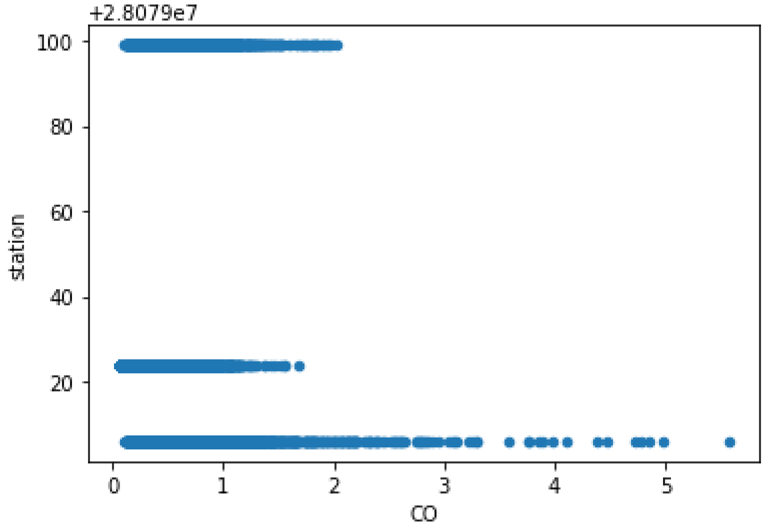
Scatter chart

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
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000
mean	1.010583	0.448056	1.262430	2.244469	0.219582	55.563929
std	1.007345	0.291706	1.074768	2.242214	0.141661	38.911677
min	0.170000	0.060000	0.250000	0.240000	0.000000	0.600000
25%	0.460000	0.270000	0.720000	0.990000	0.140000	26.510000
50%	0.670000	0.370000	1.000000	1.490000	0.190000	47.930000
75%	1.180000	0.570000	1.430000	2.820000	0.260000	76.269997
max	22.379999	5.570000	47.669998	56.500000	2.580000	477.399994

In [18]:

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

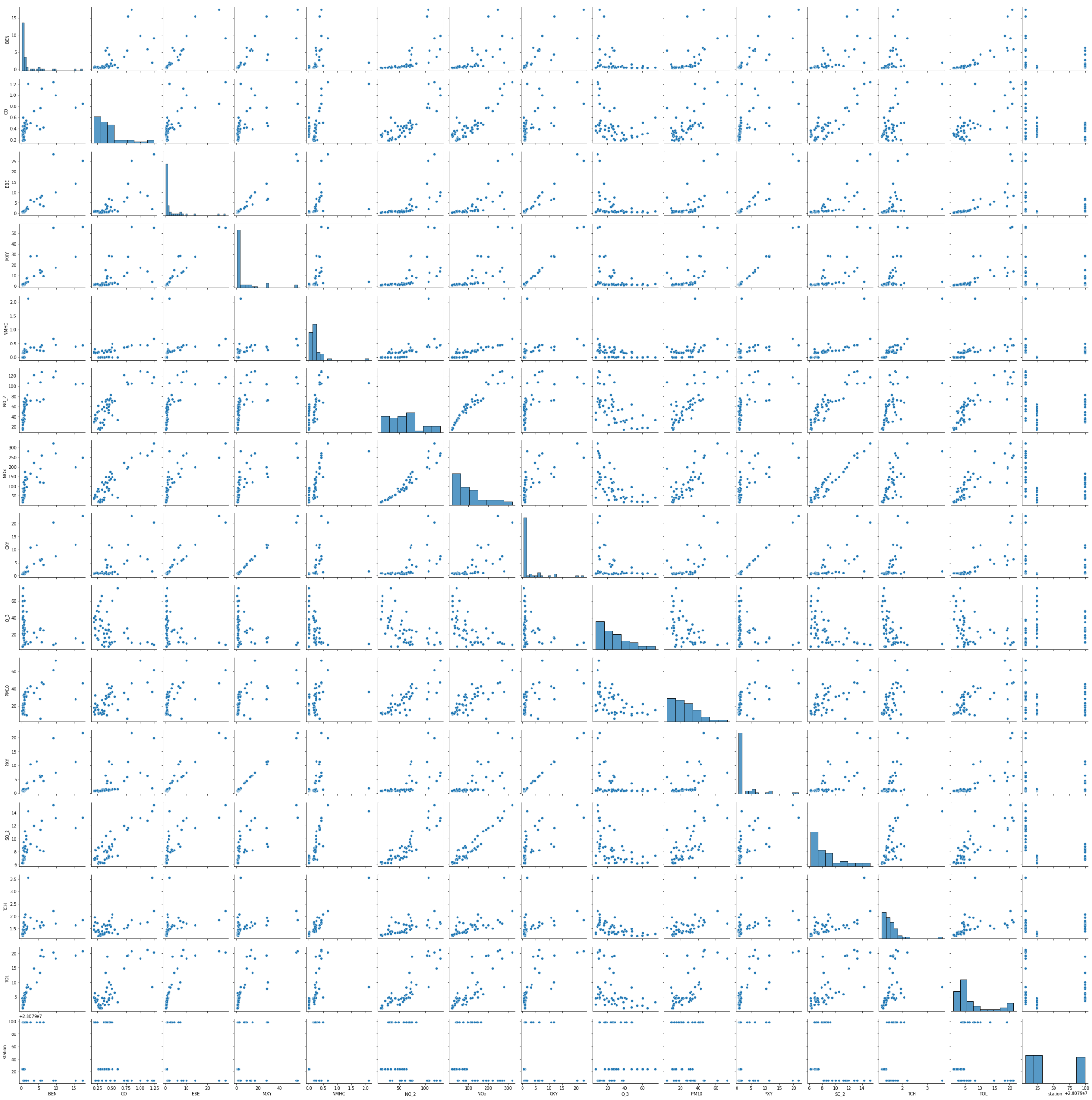
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x1c686fd3880>
```



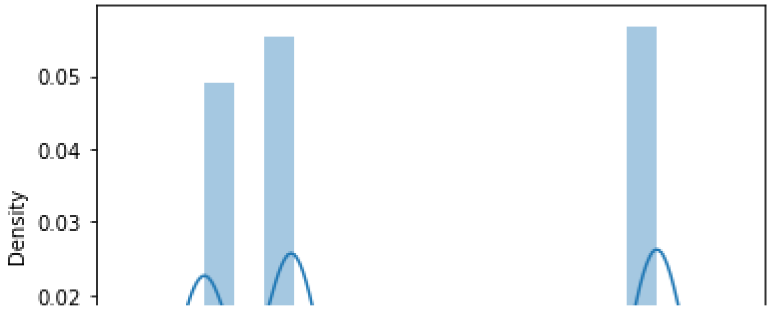
In [20]:

```
sns.distplot(df1['station'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:255
7: FutureWarning: `distplot` is a deprecated function and will be remove
d in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-l
evel function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]:

```
<AxesSubplot:xlabel='station', ylabel='Density'>
```

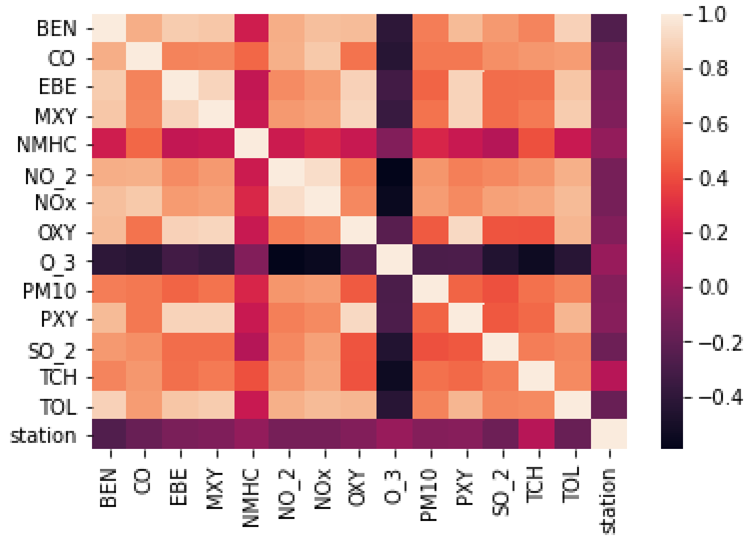


In [21]:

```
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]:

```
28078897.431059588
```

In [26]:

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

Out[26]:

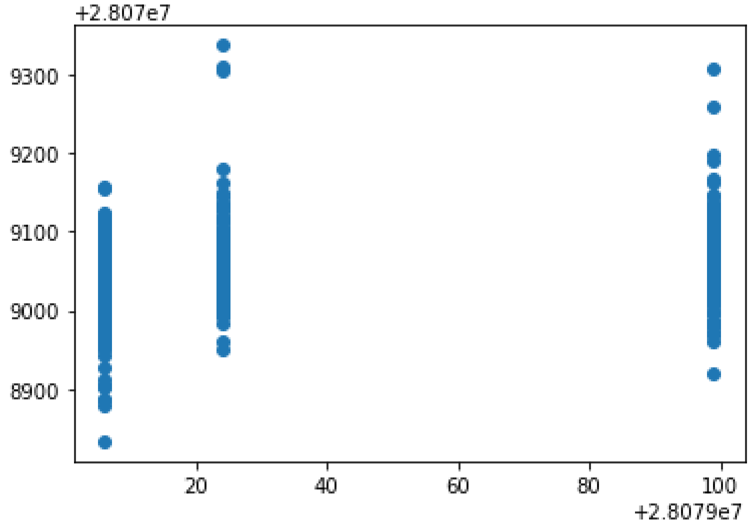
	Co-efficient
BEN	-35.490221
CO	-28.569434
EBE	6.280297
MXY	-0.514729
NMHC	-17.687543
NO_2	-0.182322
NOx	0.202518
OXY	13.116387
O_3	0.018908
PM10	-0.054712
PXY	2.181191
SO_2	-0.365033
TCH	121.852124
TOL	-1.192129

In [27]:

```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[27]:

```
<matplotlib.collections.PathCollection at 0x1c695866550>
```



ACCURACY

In [28]:

```
lr.score(x_test,y_test)
```

Out[28]:

```
0.28907770904191143
```

In [29]:

```
lr.score(x_train,y_train)
```

Out[29]:

```
0.2866577907132045
```

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]:

```
0.2895379984598364
```

In [33]:

```
rr.score(x_train,y_train)
```

Out[33]:

```
0.2863332584799254
```

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]:

```
0.036780247807414734
```

Accuracy(Lasso)

In [36]:

```
la.score(x_test,y_test)
```

Out[36]:

```
0.03515746933141528
```

Elastic Net

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([-7.07154932, -0.65489885,  0.31774118,  2.09529124,  0.        ,
       -0.22374956,  0.12687927,  1.1911961 , -0.14587028,  0.08206418,
        1.72109062, -0.78899961,  1.494183  , -1.93521422])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079064.228753425
```

In [40]:

```
prediction=en.predict(x_test)
```

In [41]:

```
en.score(x_test,y_test)
```

Out[41]:

```
0.10785348119567184
```

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)))
```

```
35.8310913665405
1454.217905501406
38.13420912384844
```

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]:

```
5.447205522232353e-13
```

In [55]:

```
logr.predict_proba(observation)
```

Out[55]:

```
array([[5.44720552e-13, 8.28692830e-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]:

```
0.8911047204272553
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

In [62]:

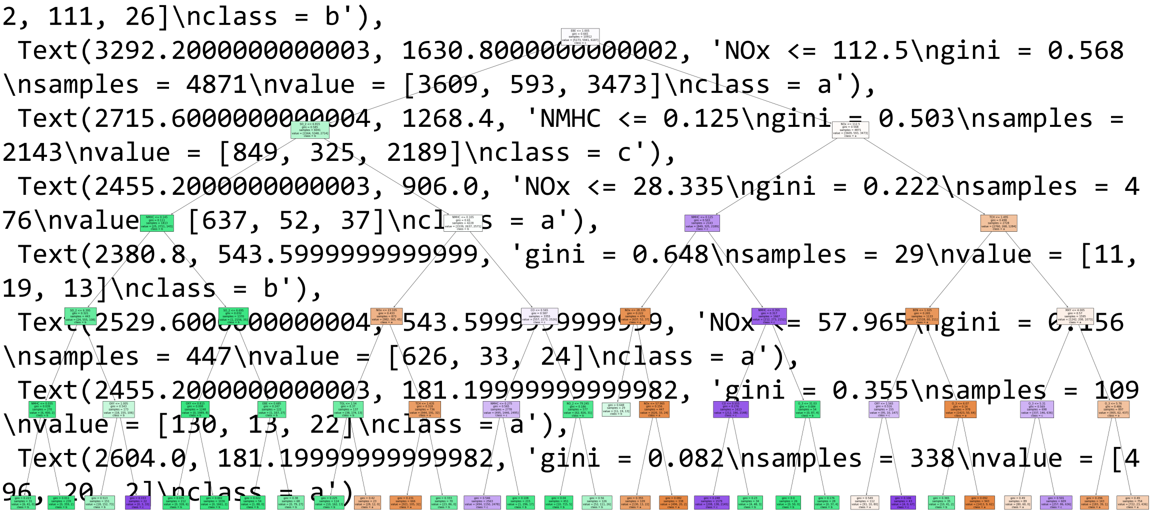
```
from sklearn.tree import plot_tree

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

Out[62]:

```
[Text(2241.3, 1993.2, 'EBE <= 1.005\ngini = 0.665\nsamples = 10912\nvalue
= [5173, 5941, 6187]\nclass = c'),
 Text(1190.4, 1630.8000000000002, 'SO_2 <= 6.815\ngini = 0.585\nsamples =
6041\nvalue = [1564, 5348, 2714]\nclass = b'),
 Text(595.2, 1268.4, 'NMHC <= 0.145\ngini = 0.111\nsamples = 1813\nvalue =
[25, 2711, 143]\nclass = b'),
 Text(297.6, 906.0, 'SO_2 <= 6.395\ngini = 0.321\nsamples = 443\nvalue =
[24, 555, 108]\nclass = b'),
 Text(148.8, 543.5999999999999, 'NMHC <= 0.035\ngini = 0.039\nsamples = 27
0\nvalue = [6, 400, 2]\nclass = b'),
 Text(74.4, 181.19999999999982, 'gini = 0.223\nsamples = 31\nvalue = [6, 4
1, 0]\nclass = b'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.011\nsamples = 239
\nvalue = [0, 359, 2]\nclass = b'),
 Text(446.40000000000003, 543.5999999999999, 'OXY <= 1.005\ngini = 0.543\n
samples = 173\nvalue = [18, 155, 106]\nclass = b'),
 Text(372.0, 181.19999999999982, 'gini = 0.513\nsamples = 151\nvalue = [1
8, 152, 73]\nclass = b'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.153\nsamples = 22\n
value = [0, 3, 33]\nclass = c'),
 Text(892.8000000000001, 906.0, 'SO_2 <= 6.695\ngini = 0.032\nsamples = 13
70\nvalue = [1, 2156, 35]\nclass = b'),
 Text(744.0, 543.5999999999999, 'OXY <= 0.815\ngini = 0.008\nsamples = 124
8\nvalue = [0, 1989, 8]\nclass = b'),
 Text(669.6, 181.19999999999982, 'gini = 0.035\nsamples = 212\nvalue = [0,
328, 6]\nclass = b'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.002\nsamples = 1036
\nvalue = [0, 1661, 2]\nclass = b'),
 Text(1041.6000000000001, 543.5999999999999, 'EBE <= 0.685\ngini = 0.247\n
samples = 122\nvalue = [1, 167, 27]\nclass = b'),
 Text(967.2, 181.19999999999982, 'gini = 0.022\nsamples = 54\nvalue = [1,
88, 0]\nclass = b'),
 Text(1116.0, 181.19999999999982, 'gini = 0.38\nsamples = 68\nvalue = [0,
79, 27]\nclass = b'),
 Text(1785.6000000000001, 1268.4, 'NMHC <= 0.105\ngini = 0.65\nsamples = 4
228\nvalue = [1539, 2637, 2571]\nclass = b'),
 Text(1488.0, 906.0, 'NOx <= 23.105\ngini = 0.433\nsamples = 873\nvalue =
[982, 365, 45]\nclass = a'),
 Text(1339.2, 543.5999999999999, 'TOL <= 1.54\ngini = 0.37\nsamples = 137
\nvalue = [38, 174, 13]\nclass = b'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.225\nsamples = 114
\nvalue = [10, 162, 13]\nclass = b'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.42\nsamples = 23\n
value = [28, 12, 0]\nclass = a'),
 Text(1636.8000000000002, 543.5999999999999, 'TCH <= 1.435\ngini = 0.318\n
samples = 736\nvalue = [944, 191, 32]\nclass = a'),
 Text(1562.4, 181.19999999999982, 'gini = 0.231\nsamples = 666\nvalue = [9
21, 105, 32]\nclass = a'),
 Text(1711.2, 181.19999999999982, 'gini = 0.333\nsamples = 70\nvalue = [2
3, 86, 0]\nclass = b'),
 Text(2083.2000000000003, 906.0, 'CO <= 0.565\ngini = 0.587\nsamples = 335
5\nvalue = [557, 2272, 2526]\nclass = c'),
 Text(1934.4, 543.5999999999999, 'NMHC <= 0.275\ngini = 0.565\nsamples = 2
778\nvalue = [495, 1446, 2495]\nclass = c'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.546\nsamples = 256
3\nvalue = [494, 1150, 2478]\nclass = c'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.108\nsamples = 215
\nvalue = [1, 296, 17]\nclass = b'),
 Text(2232.0, 543.5999999999999, 'NO_2 <= 79.245\ngini = 0.186\nsamples =
577\nvalue = [62, 826, 31]\nclass = b'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.04\nsamples = 451
```

```
\nvalue = [10, 715, 5]\nclass = b'),
 Text(2306.4, 181.19999999999982, 'gini = 0.56\nsamples = 126\nvalue = [5
2, 111, 26]\nclass = b'),
 Text(3292.2000000000003, 1630.8000000000002, 'NOx <= 112.5\ngini = 0.568
\nsamples = 4871\nvalue = [3609, 593, 3473]\nclass = a'),
 Text(2715.6000000000004, 1268.4, 'NMHC <= 0.125\ngini = 0.503\nsamples =
2143\nvalue = [849, 325, 2189]\nclass = c'),
 Text(2455.2000000000003, 906.0, 'NOx <= 28.335\ngini = 0.222\nsamples = 4
76\nvalue = [637, 52, 37]\nclass = a'),
 Text(2380.8, 543.5999999999999, 'gini = 0.648\nsamples = 29\nvalue = [11,
19, 13]\nclass = b'),
 Text(2529.6000000000004, 543.5999999999999, 'NOx <= 57.965\ngini = 0.156
\nsamples = 447\nvalue = [626, 33, 24]\nclass = a'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.355\nsamples = 109
\nvalue = [130, 13, 22]\nclass = a'),
 Text(2604.0, 181.19999999999982, 'gini = 0.082\nsamples = 338\nvalue = [4
96, 20, 2]\nclass = a'),
 Text(2976.0, 906.0, 'NMHC <= 0.355\ngini = 0.317\nsamples = 1667\nvalue =
[212, 273, 2152]\nclass = c'),
 Text(2827.2000000000003, 543.5999999999999, 'CO <= 0.555\ngini = 0.276\ns
amples = 1613\nvalue = [212, 186, 2148]\nclass = c'),
 Text([illegible], 181.19999999999982, 'gini = 0.248\nsamples = 1579\nvalue =
[208, 138, 2145]\nclass = c'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.23\nsamples = 34\n
value = [4, 48, 3]\nclass = b'),
 Text(3124.8, 543.5999999999999, 'O_3 <= 31.03\ngini = 0.084\nsamples = 54
\nvalue = [0, 87, 4]\nclass = b'),
 Text(3050.4, 181.19999999999982, 'gini = 0.0\nsamples = 26\nvalue = [0, 5
0, 0]\nclass = b'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.176\nsamples = 28
\nvalue = [0, 37, 4]\nclass = b'),
 Text(3868.8, 1268.4, 'TCH <= 1.495\ngini = 0.498\nsamples = 2728\nvalue =
[2760, 268, 1284]\nclass = a'),
 Text(3571.2000000000003, 906.0, 'BEN <= 1.025\ngini = 0.265\nsamples = 11
33\nvalue = [1518, 60, 211]\nclass = a'),
 Text(3422.4, 543.5999999999999, 'OXY <= 1.565\ngini = 0.516\nsamples = 15
5\nvalue = [95, 10, 147]\nclass = c'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.549\nsamples = 112
\nvalue = [91, 10, 80]\nclass = a'),
 Text(3496.8, 181.19999999999982, 'gini = 0.106\nsamples = 43\nvalue = [4,
0, 67]\nclass = c'),
 Text(3720.0000000000005, 543.5999999999999, 'O_3 <= 6.07\ngini = 0.14\nsa
mples = 978\nvalue = [1423, 50, 64]\nclass = a'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.365\nsamples = 35
\nvalue = [10, 41, 2]\nclass = b'),
 Text(3794.4, 181.19999999999982, 'gini = 0.092\nsamples = 943\nvalue = [1
413, 9, 62]\nclass = a'),
 Text(4166.400000000001, 906.0, 'MXY <= 4.005\ngini = 0.57\nsamples = 1595
\nvalue = [1242, 208, 1073]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'O_3 <= 5.31\ngini = 0.569\ns
amples = 698\nvalue = [337, 146, 636]\nclass = c'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.49\nsamples = 89\n
value = [80, 50, 0]\nclass = a'),
 Text(4092.0000000000005, 181.19999999999982, 'gini = 0.501\nsamples = 609
\nvalue = [257, 86, 636]\nclass = c'),
 Text(4315.200000000001, 543.5999999999999, 'O_3 <= 5.76\ngini = 0.486\nsa
mples = 897\nvalue = [905, 62, 437]\nclass = a'),
 Text(4240.8, 181.19999999999982, 'gini = 0.296\nsamples = 143\nvalue = [1
83, 39, 1]\nclass = a'),
 Text([illegible].6, 181.19999999999982, 'gini = 0.49\nsamples = 754\nvalue = [72
2, 23, 436]\nclass = a')]
```



Conclusion

Scores

Linear Regression

In [63]:

```
lr.score(x_test,y_test)
```

Out[63]:

```
0.28967770904191143
```

In [64]:

```
lr.score(x_train,y_train)
```

Out[64]:

```
0.2866577907132045
```

Lasso

In [65]:

```
lasso.score(x_test,y_test)
```

Out[65]:

```
0.03515746933141528
```

Ridge

In [66]:

```
rr.score(x_test,y_test)
```

Out[66]:

0.2895379984598364

In [67]:

```
rr.score(x_train,y_train)
```

Out[67]:

0.2863332584799254

Elastic Net

In [68]:

```
en.score(x_test,y_test)
```

Out[68]:

0.10785348119567184

Logistic Regression

In [69]:

```
logr.score(fs,target_vector)
```

Out[69]:

0.8951733624630821

Random Forest

In [70]:

```
grid_search.best_score_
```

Out[70]:

0.8911047204272553

From the above data, we can conclude that logistic regression is preferrable to other regression types

In [ ]:

```

```