Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [3]:

```
df=pd.read_csv(r"C:\Users\user\Downloads\C10_air\csvs_per_year\csvs(Dataset)\madrid_2001.
df
```

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2001-08-01 01:00:00	NaN	0.37	NaN	NaN	NaN	58.400002	87.150002	NaN	34.529999	10
1	2001-08-01 01:00:00	1.50	0.34	1.49	4.10	0.07	56.250000	75.169998	2.11	42.160000	10
2	2001-08-01 01:00:00	NaN	0.28	NaN	NaN	NaN	50.660000	61.380001	NaN	46.310001	10
3	2001-08-01 01:00:00	NaN	0.47	NaN	NaN	NaN	69.790001	73.449997	NaN	40.650002	6
4	2001-08-01 01:00:00	NaN	0.39	NaN	NaN	NaN	22.830000	24.799999	NaN	66.309998	7
...	...	...	...	...	...	...	...	...	...	...	
217867	2001-04-01 00:00:00	10.45	1.81	NaN	NaN	NaN	73.000000	264.399994	NaN	5.200000	4
217868	2001-04-01 00:00:00	5.20	0.69	4.56	NaN	0.13	71.080002	129.300003	NaN	13.460000	2
217869	2001-04-01 00:00:00	0.49	1.09	NaN	1.00	0.19	76.279999	128.399994	0.35	5.020000	4
217870	2001-04-01 00:00:00	5.62	1.01	5.04	11.38	NaN	80.019997	197.000000	2.58	5.840000	3
217871	2001-04-01 00:00:00	8.09	1.62	6.66	13.04	0.18	76.809998	206.300003	5.20	8.340000	3

217872 rows × 16 columns

Data Cleaning and Data Preprocessing

In [4]:

```
df=df.dropna()
```

In [5]:

```
df.columns
```

Out[5]:

```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O
_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29669 entries, 1 to 217871
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     29669 non-null  object 
 1   BEN      29669 non-null  float64
 2   CO       29669 non-null  float64
 3   EBE      29669 non-null  float64
 4   MXY      29669 non-null  float64
 5   NMHC     29669 non-null  float64
 6   NO_2     29669 non-null  float64
 7   NOx      29669 non-null  float64
 8   OXY      29669 non-null  float64
 9   O_3      29669 non-null  float64
 10  PM10     29669 non-null  float64
 11  PXY      29669 non-null  float64
 12  SO_2     29669 non-null  float64
 13  TCH      29669 non-null  float64
 14  TOL      29669 non-null  float64
 15  station  29669 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 3.8+ MB
```

In [17]:

```
data=df[['BEN', 'CO', 'station']]
data
```

Out[17]:

	BEN	CO	station
1	1.50	0.34	28079035
5	2.11	0.63	28079006
21	0.80	0.43	28079024
23	1.29	0.34	28079099
25	0.87	0.06	28079035
...	...	...	...
217829	11.76	4.48	28079006
217847	9.79	2.65	28079099
217849	5.86	1.22	28079035
217853	14.47	1.83	28079006
217871	8.09	1.62	28079099

29669 rows × 3 columns

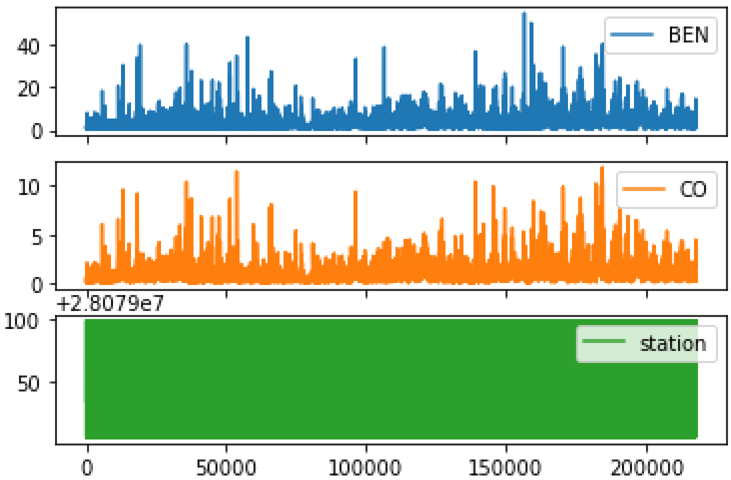
Line chart

In [18]:

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

Out[18]:

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



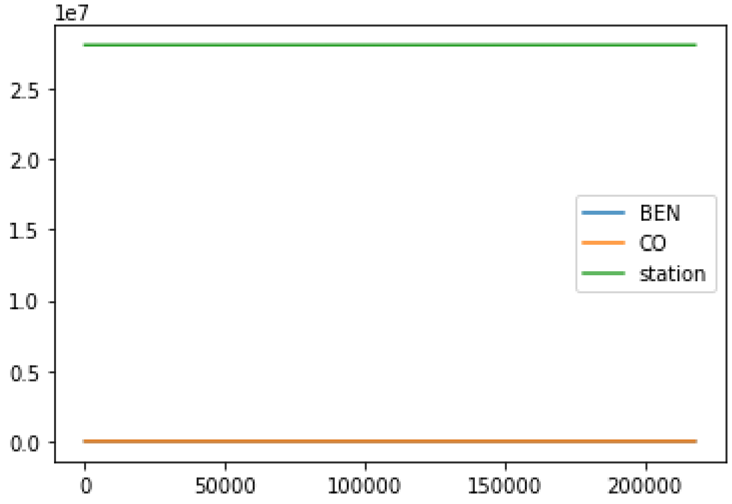
Line chart

In [19]:

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

Out[19]:

<AxesSubplot:>



Bar chart

In [20]:

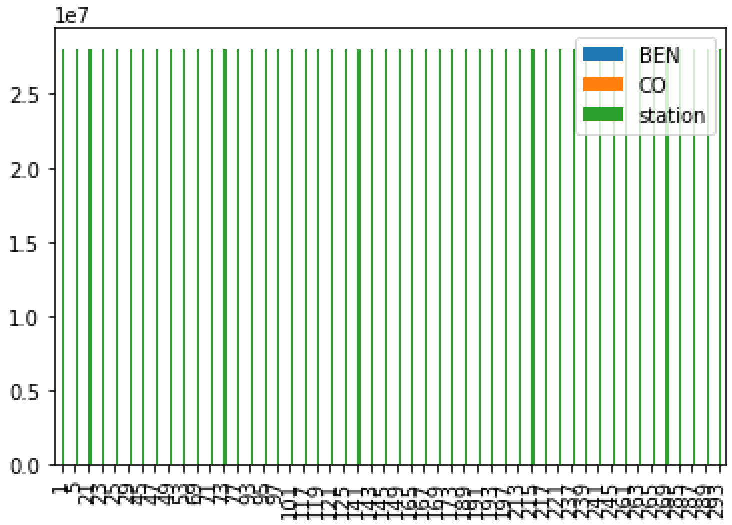
```
b=data[0:50]
```

In [21]:

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

Out[21]:

<AxesSubplot:>



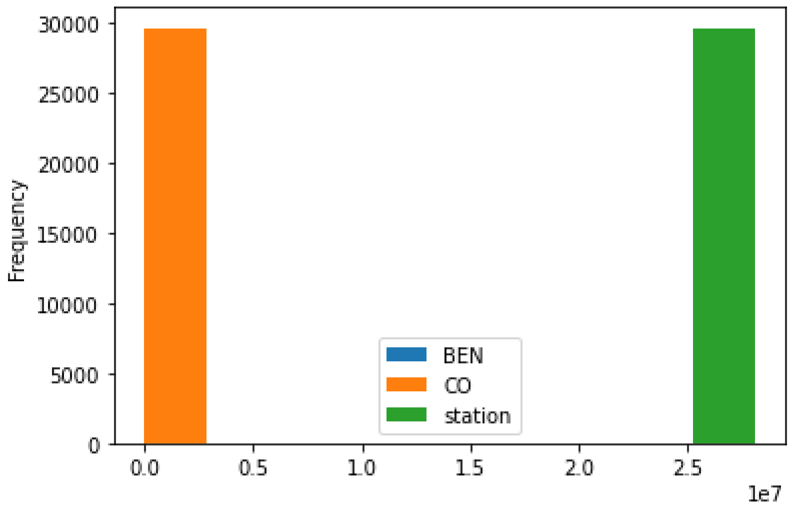
Histogram

In [22]:

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

Out[22]:

<AxesSubplot:ylabel='Frequency'>



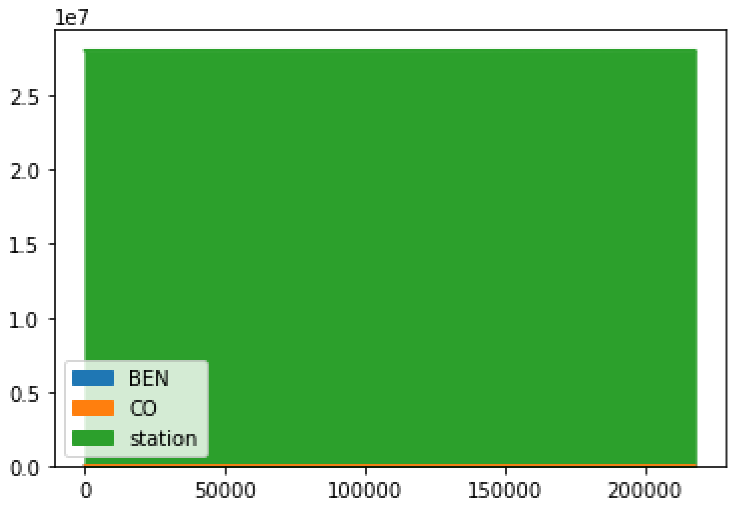
Area chart

In [23]:

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

Out[23]:

<AxesSubplot:>



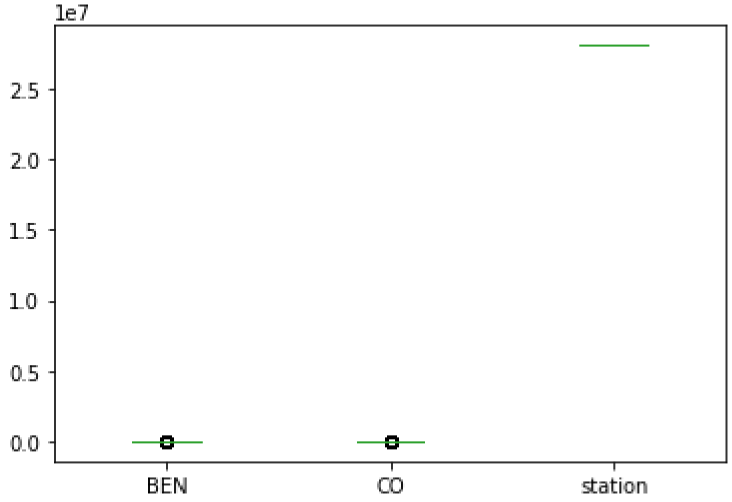
Box chart

In [24]:

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

Out[24]:

```
<AxesSubplot:>
```



Pie chart

In [25]:

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

Out[25]:

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

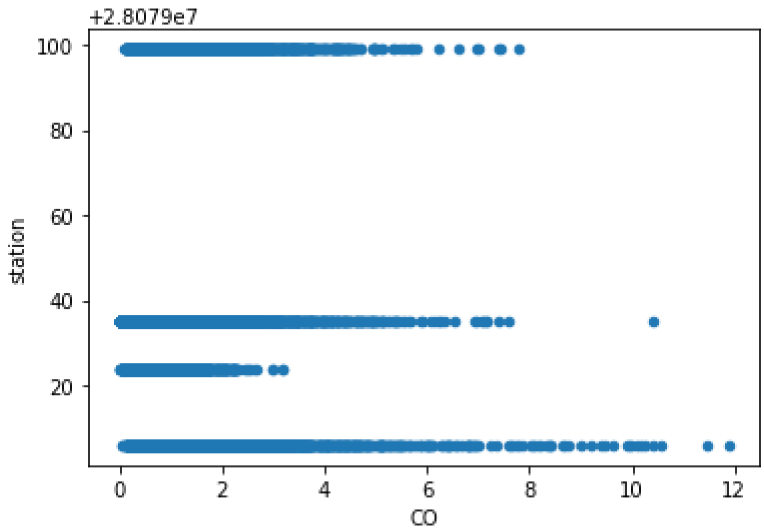
Scatter chart

In [26]:

```
data.plot.scatter(x='CO' ,y='station')
```

Out[26]:

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



In [27]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29669 entries, 1 to 217871
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     29669 non-null  object 
 1   BEN      29669 non-null  float64
 2   CO       29669 non-null  float64
 3   EBE      29669 non-null  float64
 4   MXY      29669 non-null  float64
 5   NMHC     29669 non-null  float64
 6   NO_2     29669 non-null  float64
 7   NOx      29669 non-null  float64
 8   OXY      29669 non-null  float64
 9   O_3      29669 non-null  float64
 10  PM10     29669 non-null  float64
 11  PXY      29669 non-null  float64
 12  SO_2     29669 non-null  float64
 13  TCH      29669 non-null  float64
 14  TOL      29669 non-null  float64
 15  station  29669 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 3.8+ MB
```

In [28]:

```
df.describe()
```

Out[28]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000
mean	3.361895	1.005413	3.580229	8.113086	0.195222	67.652292
std	3.176669	0.863135	3.744496	7.909701	0.192585	34.003120
min	0.100000	0.000000	0.140000	0.210000	0.000000	1.180000
25%	1.280000	0.470000	1.390000	3.040000	0.080000	44.299999
50%	2.510000	0.760000	2.600000	5.830000	0.140000	64.449997
75%	4.420000	1.270000	4.580000	10.640000	0.250000	86.540001
max	54.560001	11.890000	77.260002	150.600006	2.880000	292.700012

In [30]:

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

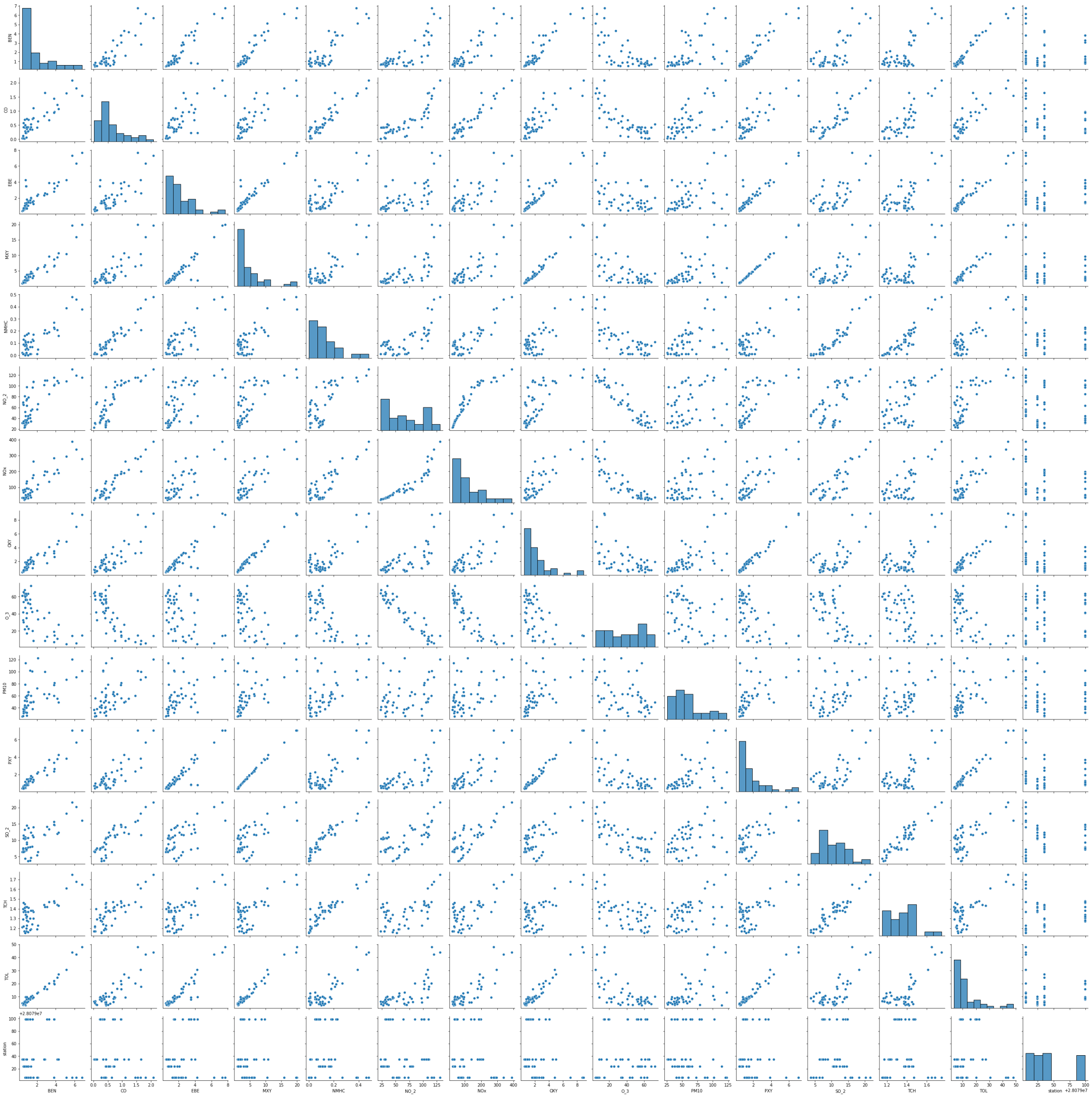
EDA AND VISUALIZATION

In [31]:

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

Out[31]:

```
<seaborn.axisgrid.PairGrid at 0x248f6e09250>
```



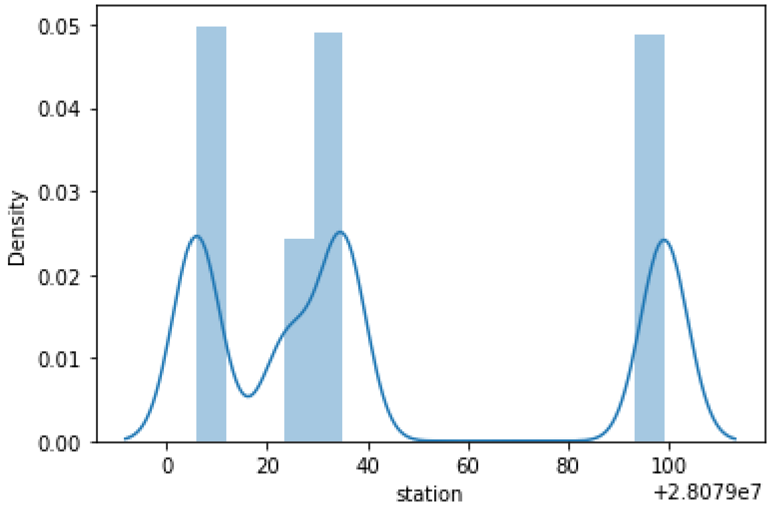
In [32]:

```
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[32]:

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

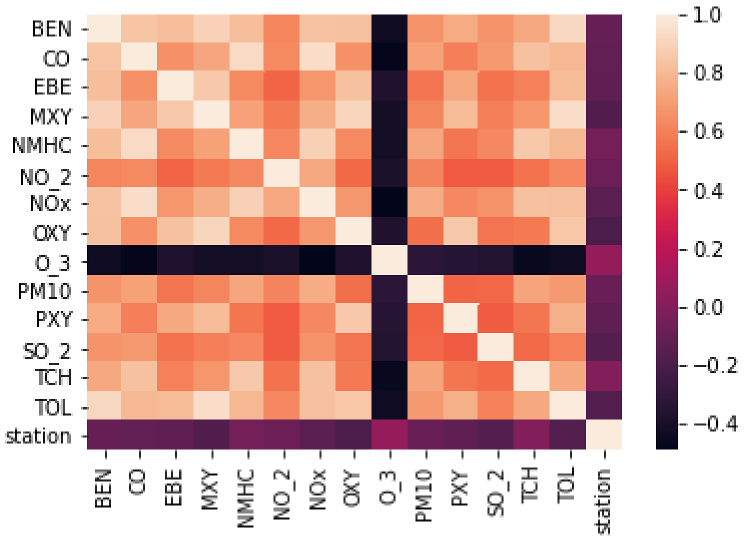


In [33]:

```
sns.heatmap(df1.corr())
```

Out[33]:

```
<AxesSubplot:>
```



TO TRAIN THE MODEL AND MODEL BULDING

In [34]:

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

In [35]:

```
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 [36]:

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[36]:

```
LinearRegression()
```

In [37]:

```
lr.intercept_
```

Out[37]:

```
28079007.64496446
```

In [38]:

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

Out[38]:

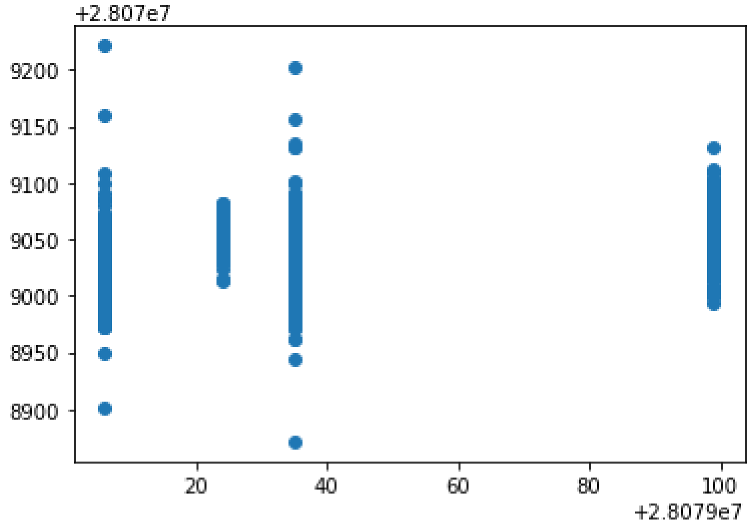
	Co-efficient
BEN	6.970464
CO	-15.513928
EBE	0.740165
MXY	-0.177168
NMHC	80.626169
NO_2	0.112355
NOx	-0.081561
OXY	-3.117769
O_3	-0.024178
PM10	-0.063713
PXY	1.304098
SO_2	-0.315505
TCH	36.870386
TOL	-1.132227

In [39]:

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

Out[39]:

```
<matplotlib.collections.PathCollection at 0x24882d7cac0>
```



ACCURACY

In [40]:

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

Out[40]:

```
0.1647074187443297
```

In [41]:

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

Out[41]:

```
0.16493490828376267
```

Ridge and Lasso

In [42]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [43]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[43]:

```
Ridge(alpha=10)
```

Accuracy(Ridge)

In [44]:

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

Out[44]:

```
0.16425017603940917
```

In [45]:

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

Out[45]:

```
0.1646970650280114
```

In [46]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[46]:

```
Lasso(alpha=10)
```

In [47]:

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

Out[47]:

```
0.040014295690649515
```

Accuracy(Lasso)

In [48]:

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

Out[48]:

```
0.0376097730714563
```

In [49]:

```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[49]:

```
ElasticNet()
```

In [50]:

```
en.coef_
```

Out[50]:

```
array([ 4.84124189,  0.        ,  0.71064843, -0.29880189,  0.0723964 ,
        0.06242194, -0.03254308, -2.52507789, -0.033144  ,  0.07237582,
        0.81792277, -0.33712439,  1.21467131, -0.63262354])
```

In [51]:

```
en.intercept_
```

Out[51]:

```
28079049.346572097
```

In [52]:

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

In [53]:

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

Out[53]:

```
0.09943610174409923
```

Evaluation Metrics

In [54]:

```
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)))
```

```
30.369818605860505
1208.4702603479132
34.76305884625105
```

Logistic Regression

In [55]:

```
from sklearn.linear_model import LogisticRegression
```

In [56]:

```
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 [57]:

```
feature_matrix.shape
```

Out[57]:

```
(29669, 14)
```

In [58]:

```
target_vector.shape
```

Out[58]:

```
(29669,)
```

In [59]:

```
from sklearn.preprocessing import StandardScaler
```

In [60]:

```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [61]:

```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[61]:

```
LogisticRegression(max_iter=10000)
```

In [62]:

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

In [63]:

```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079035]
```

In [64]:

```
logr.classes_
```

Out[64]:

```
array([28079006, 28079024, 28079035, 28079099], dtype=int64)
```

In [65]:

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

Out[65]:

```
0.8087229094340894
```

In [66]:

```
logr.predict_proba(observation)[0][0]
```

Out[66]:

```
1.724527777144498e-43
```

In [67]:

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

Out[67]:

```
array([[1.72452778e-43, 2.43756289e-56, 9.99998565e-01, 1.43537418e-06]])
```

Random Forest

In [68]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [69]:

```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[69]:

```
RandomForestClassifier()
```

In [70]:

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

In [71]:

```
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[71]:

```
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 [72]:

```
grid_search.best_score_
```

Out[72]:

```
0.7301617873651772
```

In [73]:

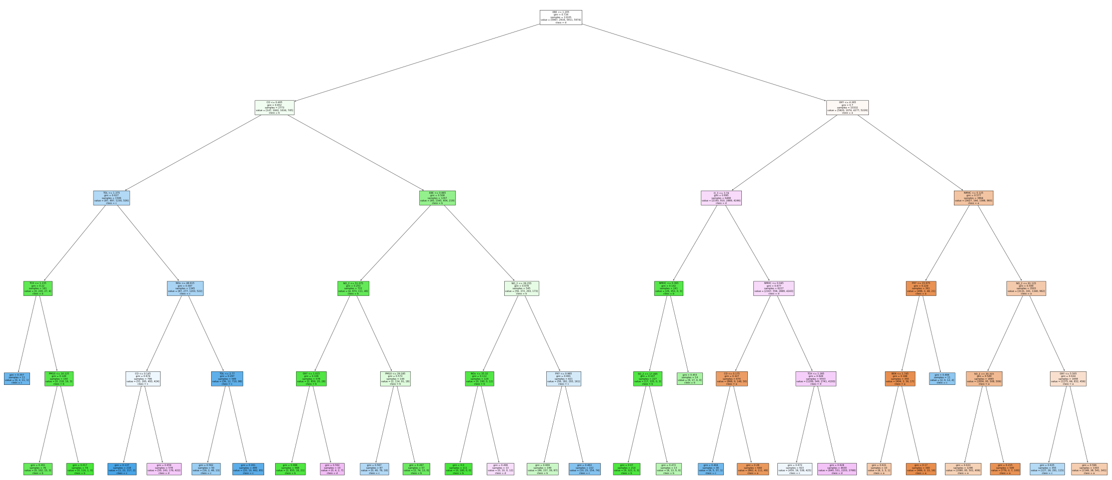
```
rfc_best=grid_search.best_estimator_
```

In [74]:

```
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
```

```
5\nvalue = [127, 28, 291, 115]\nclass = c'),
 Text(4379.773584905661, 181.19999999999982, 'gini = 0.596\nsamples = 13
03\nvalue = [1148, 38, 541, 341]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.15333059191475773

Ridge Regression:0.15336555741871216

Lasso Regression:0.03896350073644961

ElasticNet Regression:0.09871426228846358

Logistic Regression:0.8087229094340894

Random Forest:0.7331953004622496

Logistic Regression is suitable for this dataset