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\Desktop\csvs_per_year\csvs_per_year\madrid_2001.
df
```

Out[2]:

	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	105.0(
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	100.5(
2	2001-08-01 01:00:00	NaN	0.28	NaN	NaN	NaN	50.660000	61.380001	NaN	46.310001	100.0(
3	2001-08-01 01:00:00	NaN	0.47	NaN	NaN	NaN	69.790001	73.449997	NaN	40.650002	69.7
4	2001-08-01 01:00:00	NaN	0.39	NaN	NaN	NaN	22.830000	24.799999	NaN	66.309998	75.1
...	...	...	...	...	...	...	...	...	...	...	
217867	2001-04-01 00:00:00	10.45	1.81	NaN	NaN	NaN	73.000000	264.399994	NaN	5.200000	47.8
217868	2001-04-01 00:00:00	5.20	0.69	4.56	NaN	0.13	71.080002	129.300003	NaN	13.460000	26.8(
217869	2001-04-01 00:00:00	0.49	1.09	NaN	1.00	0.19	76.279999	128.399994	0.35	5.020000	40.7
217870	2001-04-01 00:00:00	5.62	1.01	5.04	11.38	NaN	80.019997	197.000000	2.58	5.840000	37.8
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	35.3(

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 [7]:
```
data=df[['CO' ,'station']]
data
```

Out[7]:

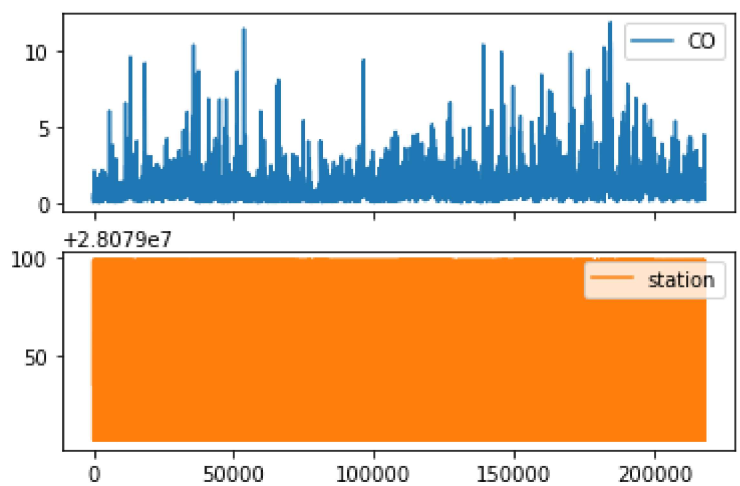
	CO	station
1	0.34	28079035
5	0.63	28079006
21	0.43	28079024
23	0.34	28079099
25	0.06	28079035
...	...	...
217829	4.48	28079006
217847	2.65	28079099
217849	1.22	28079035
217853	1.83	28079006
217871	1.62	28079099

29669 rows × 2 columns

Line chart

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

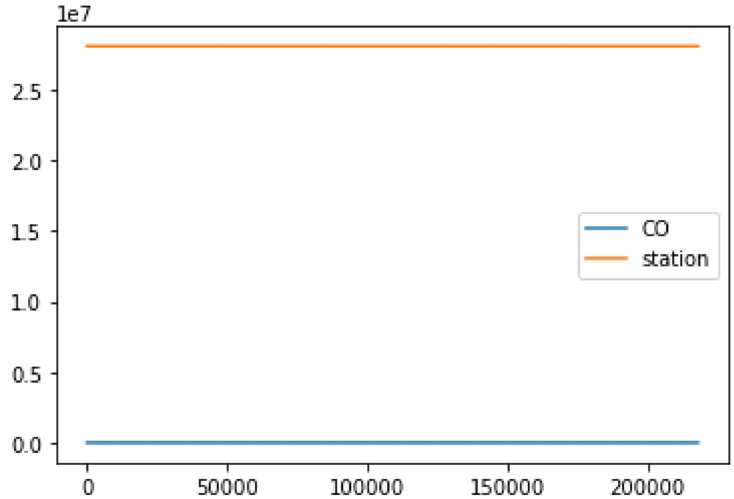
Out[8]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [9]:
```
data.plot.line()
```

Out[9]: <AxesSubplot:>

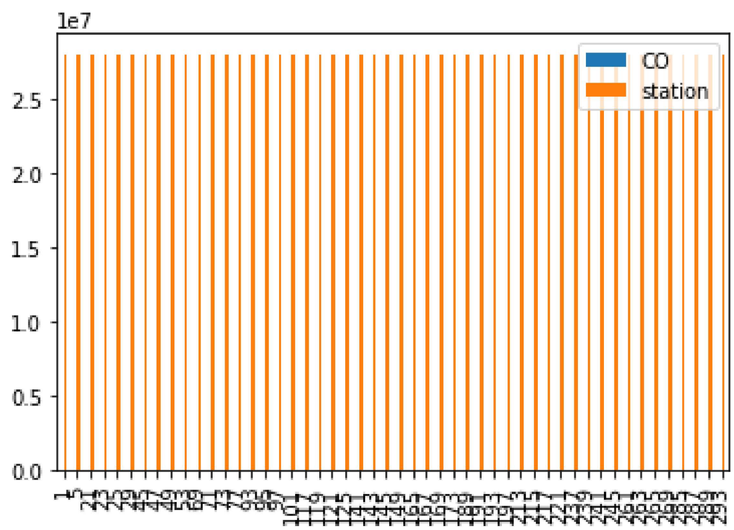


Bar chart

In [10]:
```
b=data[0:50]
```

In [11]:
```
b.plot.bar()
```

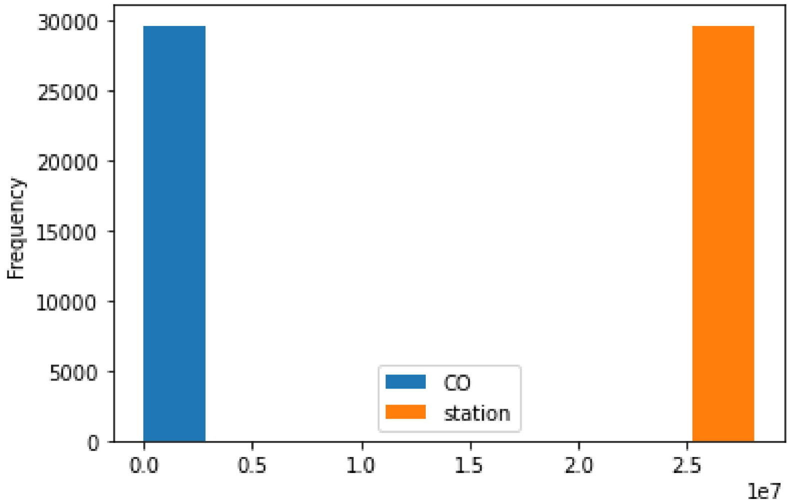
Out[11]: <AxesSubplot:>



Histogram

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

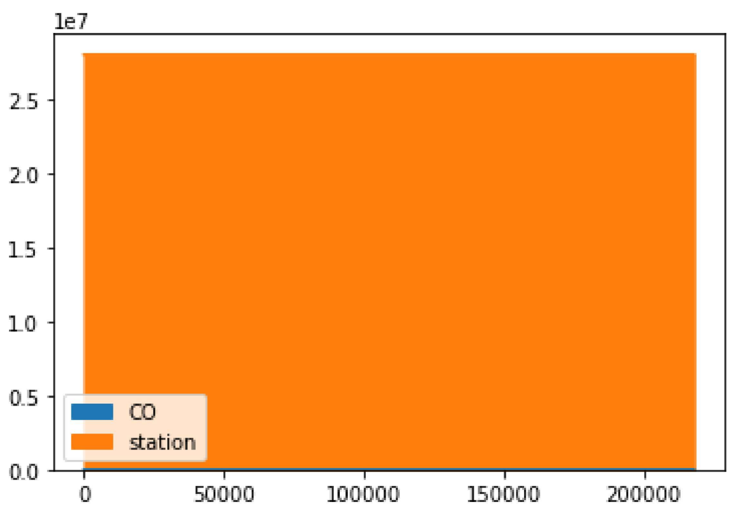
Out[12]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

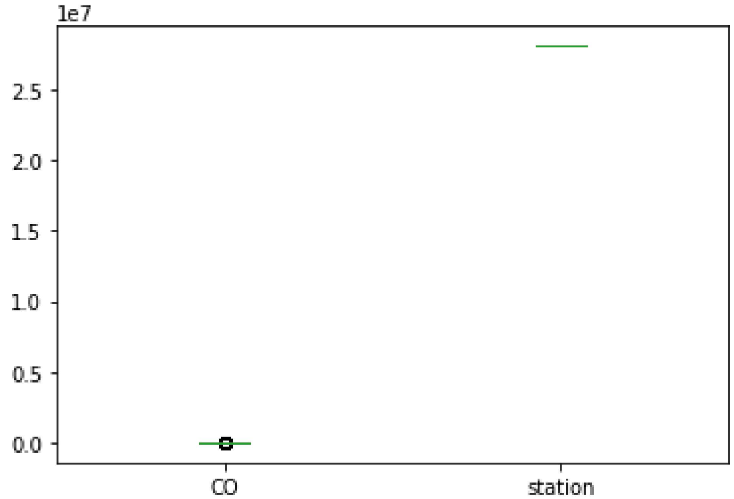
Out[13]: <AxesSubplot:>



Box chart

In [14]:
```
data.plot.box()
```

Out[14]: <AxesSubplot:>



Pie chart

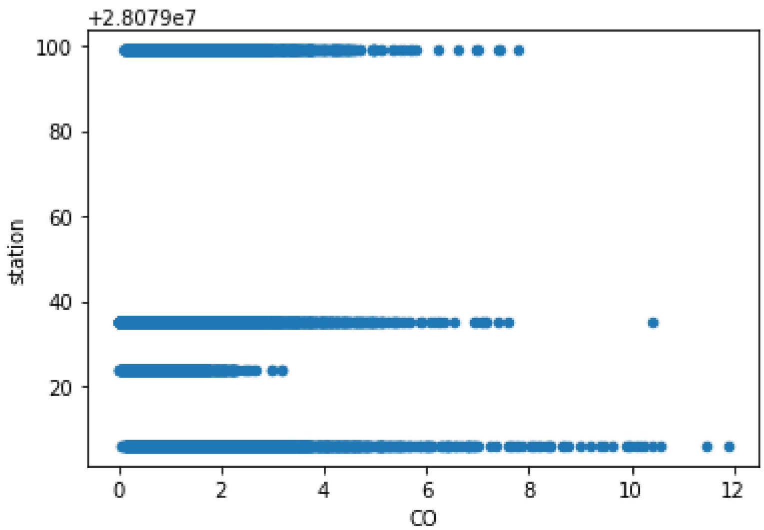
In [15]:
```
b.plot.pie(y='station' )
```

Out[15]: <AxesSubplot:ylabel='station'>

Scatter chart

In [16]:
```
data.plot.scatter(x='CO' ,y='station')
```

Out[16]:
```
<AxesSubplot:xlabel='CO', ylabel='station'>
```



In [17]:
```
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 [18]:
```
df.describe()
```

Out[18]:

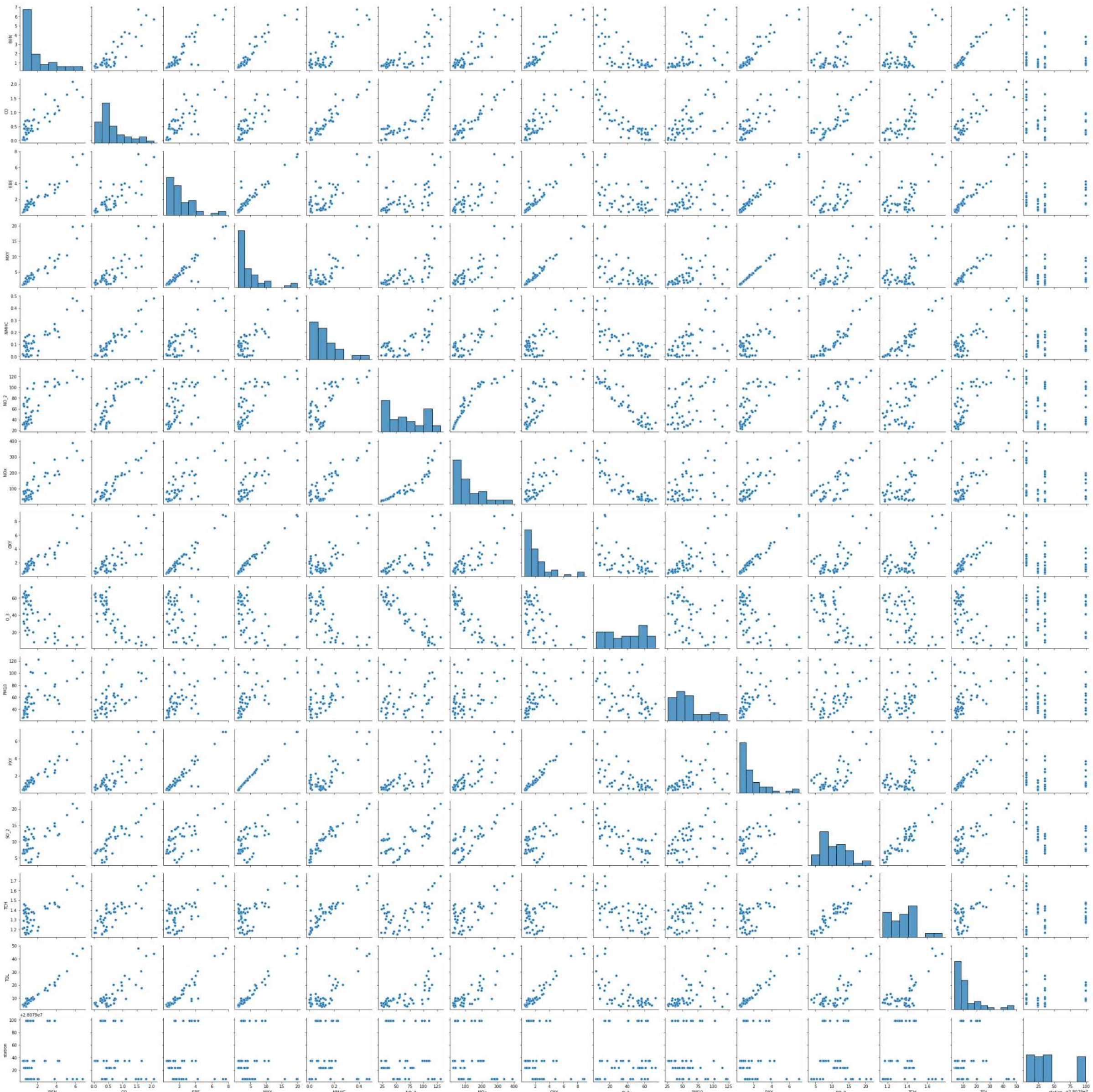
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000	296
mean	3.361895	1.005413	3.580229	8.113086	0.195222	67.652292	1
std	3.176669	0.863135	3.744496	7.909701	0.192585	34.003120	1
min	0.100000	0.000000	0.140000	0.210000	0.000000	1.180000	
25%	1.280000	0.470000	1.390000	3.040000	0.080000	44.299999	(
50%	2.510000	0.760000	2.600000	5.830000	0.140000	64.449997	1
75%	4.420000	1.270000	4.580000	10.640000	0.250000	86.540001	2
max	54.560001	11.890000	77.260002	150.600006	2.880000	292.700012	19

In [19]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [20]:
```
sns.pairplot(df1[0:50])
```

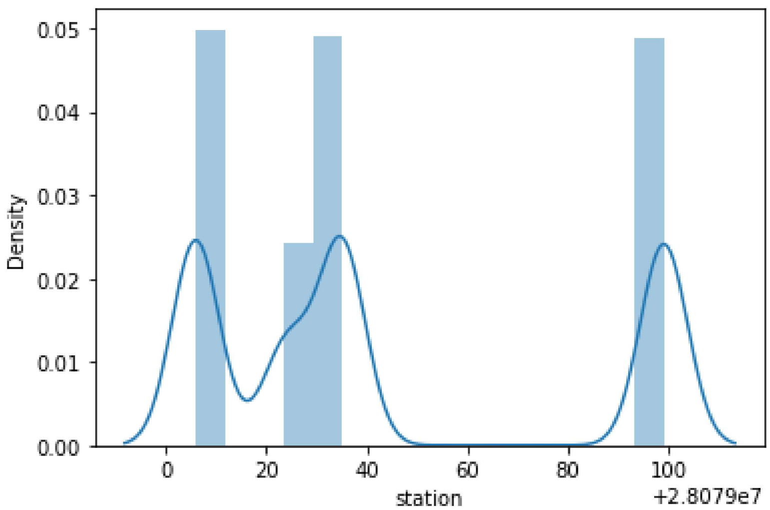
Out[20]: <seaborn.axisgrid.PairGrid at 0x25f079fb730>



In [21]:
```
sns.distplot(df1['station'])
```

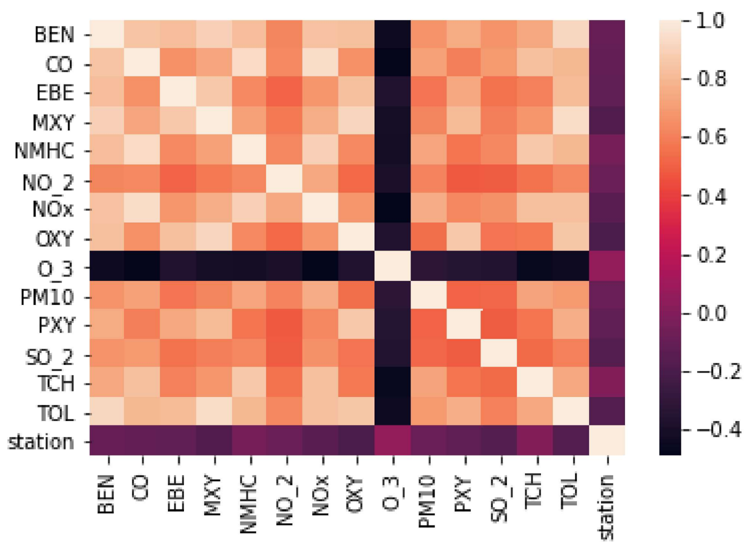
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  warnings.warn(msg, FutureWarning)
```

Out[21]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [22]:
```
sns.heatmap(df1.corr())
```

Out[22]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [23]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [24]:
```
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 [25]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[25]: LinearRegression()

In [26]:
```
lr.intercept_
```

Out[26]: 28079005.750684857

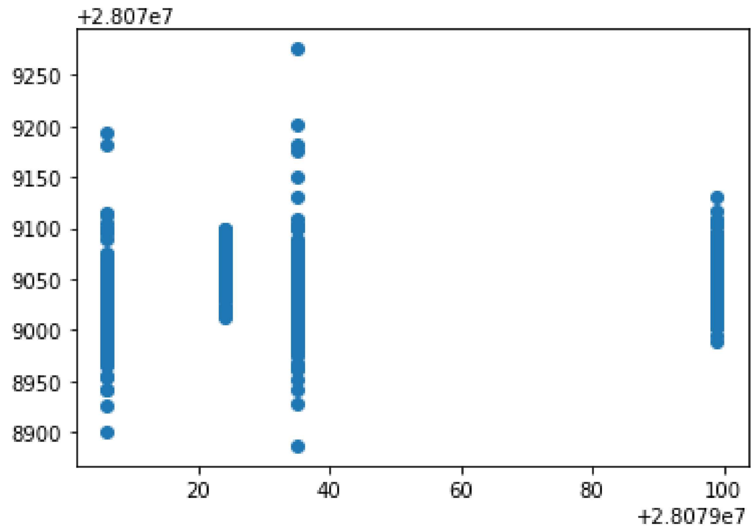
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	7.036777
CO	-15.503169
EBE	0.765946
MXY	-0.254371
NMHC	80.501035
NO_2	0.107271
NOx	-0.086776
OXY	-3.105663
O_3	-0.032001
PM10	-0.054825
PXY	1.583828
SO_2	-0.282500
TCH	38.496140
TOL	-1.178963

In [28]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[28]: <matplotlib.collections.PathCollection at 0x25f1811df10>



ACCURACY

In [29]:
```
lr.score(x_test,y_test)
```

Out[29]: 0.16251381576622992

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

Out[34]: 0.16590757797196254

Ridge and Lasso

In [31]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [32]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[32]: Ridge(alpha=10)

Accuracy(Ridge)

In [33]:
```
rr.score(x_test,y_test)
```
Out[33]: 0.1621791546763528

In [35]:
```
rr.score(x_train,y_train)
```
Out[35]: 0.16566844374256262

In [36]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```
Out[36]: Lasso(alpha=10)

In [37]:
```
la.score(x_train,y_train)
```
Out[37]: 0.03858125875748031

Accuracy(Lasso)

In [38]:
```
la.score(x_test,y_test)
```
Out[38]: 0.0408867933552266

In [39]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```
Out[39]: ElasticNet()

In [40]:
```
en.coef_
```
Out[40]:
```
array([ 4.91193074,  0.        ,  0.76167768, -0.3879746 ,  0.05769821,
        0.0527924 , -0.03142455, -2.49417495, -0.0392047 ,  0.08167882,
        1.00767918, -0.31493315,  1.21909083, -0.67553838])
```

In [41]:
```
en.intercept_
```
Out[41]: 28079049.384267006

In [42]:
```
prediction=en.predict(x_test)
```

In [43]:
```
en.score(x_test,y_test)
```
Out[43]: 0.10280994311474245

Evaluation Metrics

In [44]:
```
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)))
```

```
30.39035946931893
1218.54600288857
34.90767827983652
```

Logistic Regression

In [45]:
```
from sklearn.linear_model import LogisticRegression
```

In [46]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [47]:
```
feature_matrix.shape
```

Out[47]: (29669, 14)

In [48]:
```
target_vector.shape
```

Out[48]: (29669,)

In [49]:
```
from sklearn.preprocessing import StandardScaler
```

In [50]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [51]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[51]: LogisticRegression(max_iter=10000)

In [52]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [53]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079035]
```

In [54]:
```
logr.classes_
```

Out[54]:
```
array([28079006, 28079024, 28079035, 28079099], dtype=int64)
```

In [55]:
```
logr.score(fs,target_vector)
```

Out[55]:
```
0.8087229094340894
```

In [56]:
```
logr.predict_proba(observation)[0][0]
```

Out[56]:
```
1.7245277771444498e-43
```

In [57]:
```
logr.predict_proba(observation)
```

Out[57]:
```
array([[1.72452778e-43, 2.43756289e-56, 9.99998565e-01, 1.43537418e-06]])
```

Random Forest

In [58]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [59]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[59]:
```
RandomForestClassifier()
```

In [60]:
```
parameters={'max_depth':[1,2,3,4,5],
 'min_samples_leaf':[5,10,15,20,25],
 'n_estimators':[10,20,30,40,50]
}
```

In [61]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="acc
grid_search.fit(x_train,y_train)
```

Out[61]:
```
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 [62]:
```
grid_search.best_score_
```

Out[62]:
```
0.7254429892141756
```

In [63]:
```
rfc_best=grid_search.best_estimator_
```

In [64]:
```
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',
```

Out[64]:
```
[Text(2232.0, 1993.2, 'SO_2 <= 20.71\ngini = 0.732\nsamples = 13149\nvalue
= [6016, 2795, 6037, 5920]\nclass = c'),
 Text(1116.0, 1630.8000000000002, 'MXY <= 1.085\ngini = 0.729\nsamples = 91
42\nvalue = [2427, 2738, 4984, 4225]\nclass = c'),
 Text(558.0, 1268.4, 'NMHC <= 0.035\ngini = 0.395\nsamples = 994\nvalue =
[2, 1127, 347, 41]\nclass = b'),
 Text(279.0, 906.0, 'CO <= 0.305\ngini = 0.267\nsamples = 162\nvalue = [0,
21, 214, 17]\nclass = c'),
 Text(139.5, 543.5999999999999, 'PXY <= 0.515\ngini = 0.195\nsamples = 146
\nvalue = [0, 7, 201, 17]\nclass = c'),
 Text(69.75, 181.19999999999982, 'gini = 0.058\nsamples = 130\nvalue = [0,
2, 195, 4]\nclass = c'),
 Text(209.25, 181.19999999999982, 'gini = 0.601\nsamples = 16\nvalue = [0,
5, 6, 13]\nclass = d'),
 Text(418.5, 543.5999999999999, 'TOL <= 0.96\ngini = 0.499\nsamples = 16\nv
alue = [0, 14, 13, 0]\nclass = b'),
 Text(348.75, 181.19999999999982, 'gini = 0.0\nsamples = 8\nvalue = [0, 13,
0, 0]\nclass = b'),
 Text(488.25, 181.19999999999982, 'gini = 0.133\nsamples = 8\nvalue = [0,
```

Conclusion

Accuracy

Linear Regression:0.16520772737246636

Ridge Regression:0.16489635135341363

Lasso Regression:0.04002946671090035

ElasticNet Regression:0.10236914940351627

Logistic Regression:0.8087229094340894

Random Forest:0.7372399845916795

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