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_2015").fillna(1)
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	2015-10-01 01:00:00	1.0	0.8	1.0	1.00	90.0	82.0	1.0	1.0	1.0	10.0	1.00	1.0	28079004
1	2015-10-01 01:00:00	2.0	0.8	1.6	0.33	40.0	95.0	4.0	37.0	24.0	12.0	1.83	8.3	28079008
2	2015-10-01 01:00:00	3.1	1.0	1.8	1.00	29.0	97.0	1.0	1.0	1.0	1.0	1.00	7.1	28079011
3	2015-10-01 01:00:00	1.0	0.6	1.0	1.00	30.0	103.0	2.0	1.0	1.0	1.0	1.00	1.0	28079016
4	2015-10-01 01:00:00	1.0	1.0	1.0	1.00	95.0	96.0	2.0	1.0	1.0	9.0	1.00	1.0	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210091	2015-08-01 00:00:00	1.0	0.2	1.0	1.00	11.0	33.0	53.0	1.0	1.0	1.0	1.00	1.0	28079056
210092	2015-08-01 00:00:00	1.0	0.2	1.0	1.00	1.0	5.0	1.0	26.0	1.0	10.0	1.00	1.0	28079057
210093	2015-08-01 00:00:00	1.0	1.0	1.0	1.00	1.0	7.0	74.0	1.0	1.0	1.0	1.00	1.0	28079058
210094	2015-08-01 00:00:00	1.0	1.0	1.0	1.00	3.0	7.0	65.0	1.0	1.0	1.0	1.00	1.0	28079059
210095	2015-08-01 00:00:00	1.0	1.0	1.0	1.00	1.0	9.0	54.0	29.0	1.0	1.0	1.00	1.0	28079060

210096 rows × 14 columns

Data Cleaning and Data Preprocessing

In [3]:
```
1 df.columns
```

Out[3]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [4]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210096 entries, 0 to 210095
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     210096 non-null  object 
 1   BEN      210096 non-null  float64
 2   CO       210096 non-null  float64
 3   EBE      210096 non-null  float64
 4   NMHC     210096 non-null  float64
 5   NO       210096 non-null  float64
 6   NO_2     210096 non-null  float64
 7   O_3      210096 non-null  float64
 8   PM10     210096 non-null  float64
 9   PM25     210096 non-null  float64
 10  SO_2     210096 non-null  float64
 11  TCH      210096 non-null  float64
 12  TOL      210096 non-null  float64
 13  station  210096 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 22.4+ MB
```

In [5]:
```
data=df[['CO' ,'station']]
data
```

Out[5]:

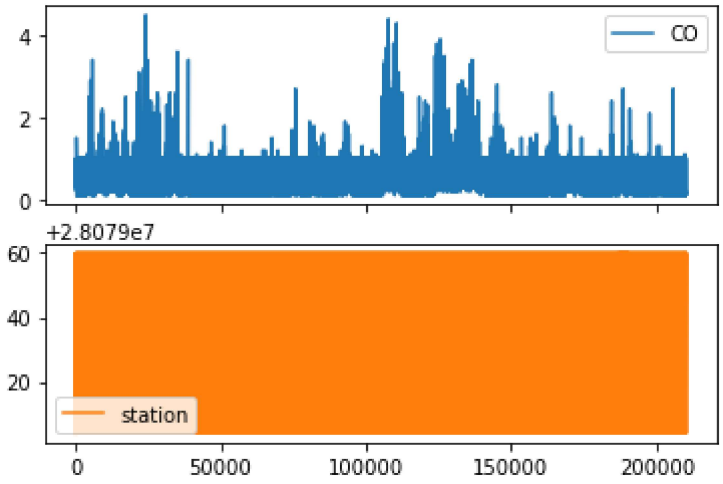
	CO	station
0	0.8	28079004
1	0.8	28079008
2	1.0	28079011
3	0.6	28079016
4	1.0	28079017
...	...	...
210091	0.2	28079056
210092	0.2	28079057
210093	1.0	28079058
210094	1.0	28079059
210095	1.0	28079060

210096 rows × 2 columns

Line chart

In [6]:
```
1 data.plot.line(subplots=True)
```

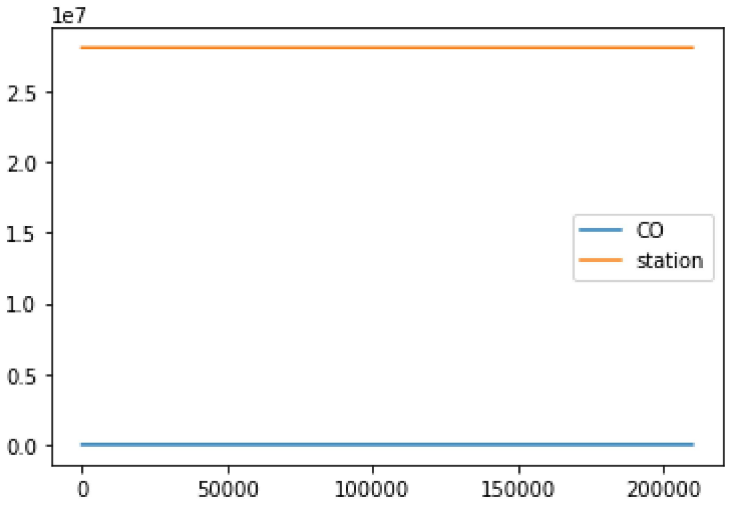
Out[6]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [7]:
```
1 data.plot.line()
```

Out[7]:
```
<AxesSubplot:>
```

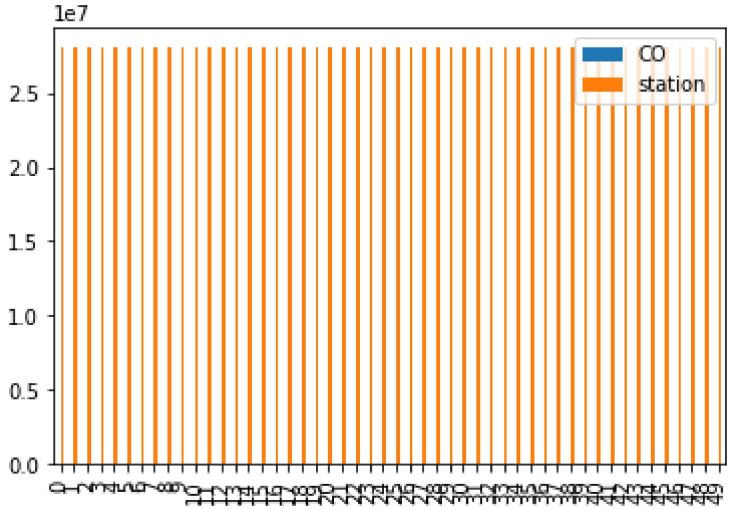


Bar chart

In [8]:
```
1 b=data[0:50]
```

In [9]:
```
1 b.plot.bar()
```

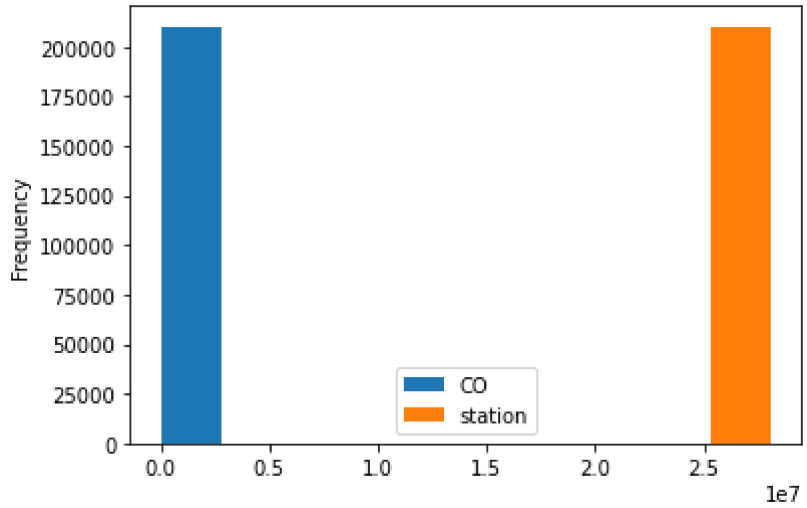
Out[9]: <AxesSubplot:>



Histogram

In [10]:
```
1 data.plot.hist()
```

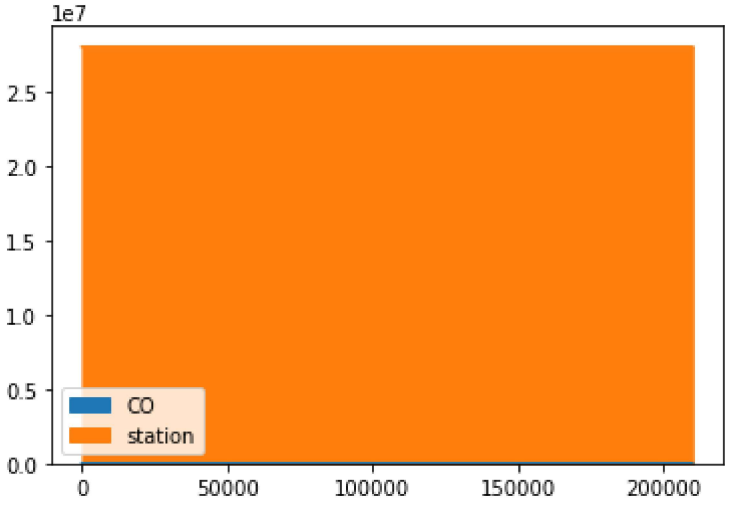
Out[10]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [11]:
```
1 data.plot.area()
```

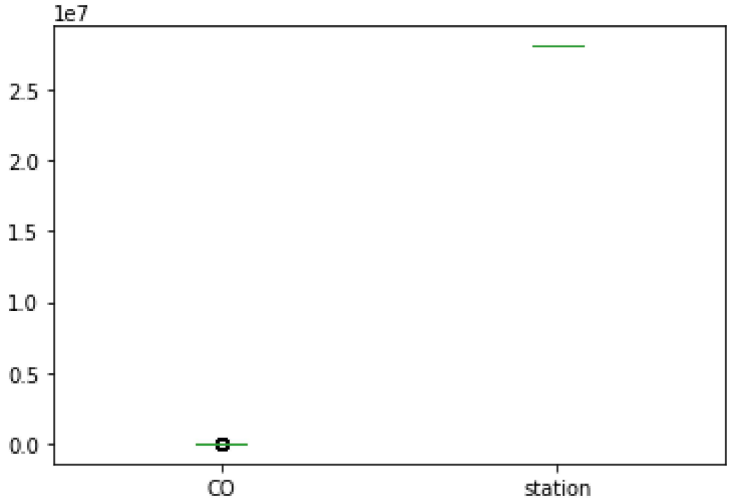
Out[11]: <AxesSubplot:>



Box chart

In [12]:
```
1 data.plot.box()
```

Out[12]: <AxesSubplot:>

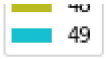


Pie chart

In [13]:
```
1 b.plot.pie(y='station' )
```

Out[13]:
```
<AxesSubplot:ylabel='station'>
```

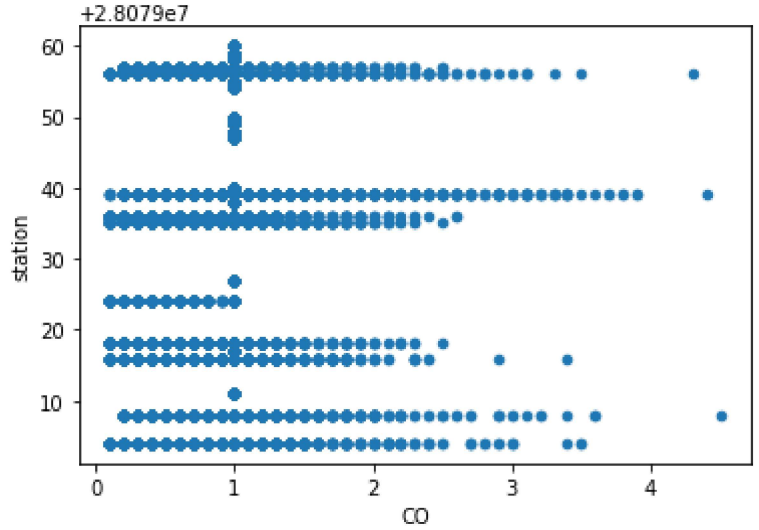

station
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47



Scatter chart

In [14]:
```
1 data.plot.scatter(x='CO' ,y='station')
```

Out[14]: <AxesSubplot:xlabel='CO', ylabel='station'>



In [15]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210096 entries, 0 to 210095
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     210096 non-null  object 
 1   BEN      210096 non-null  float64
 2   CO       210096 non-null  float64
 3   EBE      210096 non-null  float64
 4   NMHC     210096 non-null  float64
 5   NO       210096 non-null  float64
 6   NO_2     210096 non-null  float64
 7   O_3      210096 non-null  float64
 8   PM10     210096 non-null  float64
 9   PM25     210096 non-null  float64
 10  SO_2     210096 non-null  float64
 11  TCH      210096 non-null  float64
 12  TOL      210096 non-null  float64
 13  station  210096 non-null  int64
```

In [16]:
```
1 df.describe()
```

Out[16]:

	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	
count	210096.000000	210096.000000	210096.000000	210096.000000	210096.000000	210096.000000	210096.000000	210096.000000	2100
mean	0.940954	0.738216	0.877570	0.909471	26.578674	40.734631	29.691679	10.789782	
std	0.503139	0.361954	0.485907	0.263325	59.421024	32.779834	36.743137	15.114735	
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	1.000000	0.000000	
25%	1.000000	0.300000	1.000000	1.000000	2.000000	16.000000	1.000000	1.000000	
50%	1.000000	1.000000	1.000000	1.000000	6.000000	32.000000	7.000000	1.000000	
75%	1.000000	1.000000	1.000000	1.000000	22.000000	57.000000	56.000000	17.000000	
max	17.700001	4.500000	19.700001	2.800000	1146.000000	424.000000	236.000000	250.000000	8

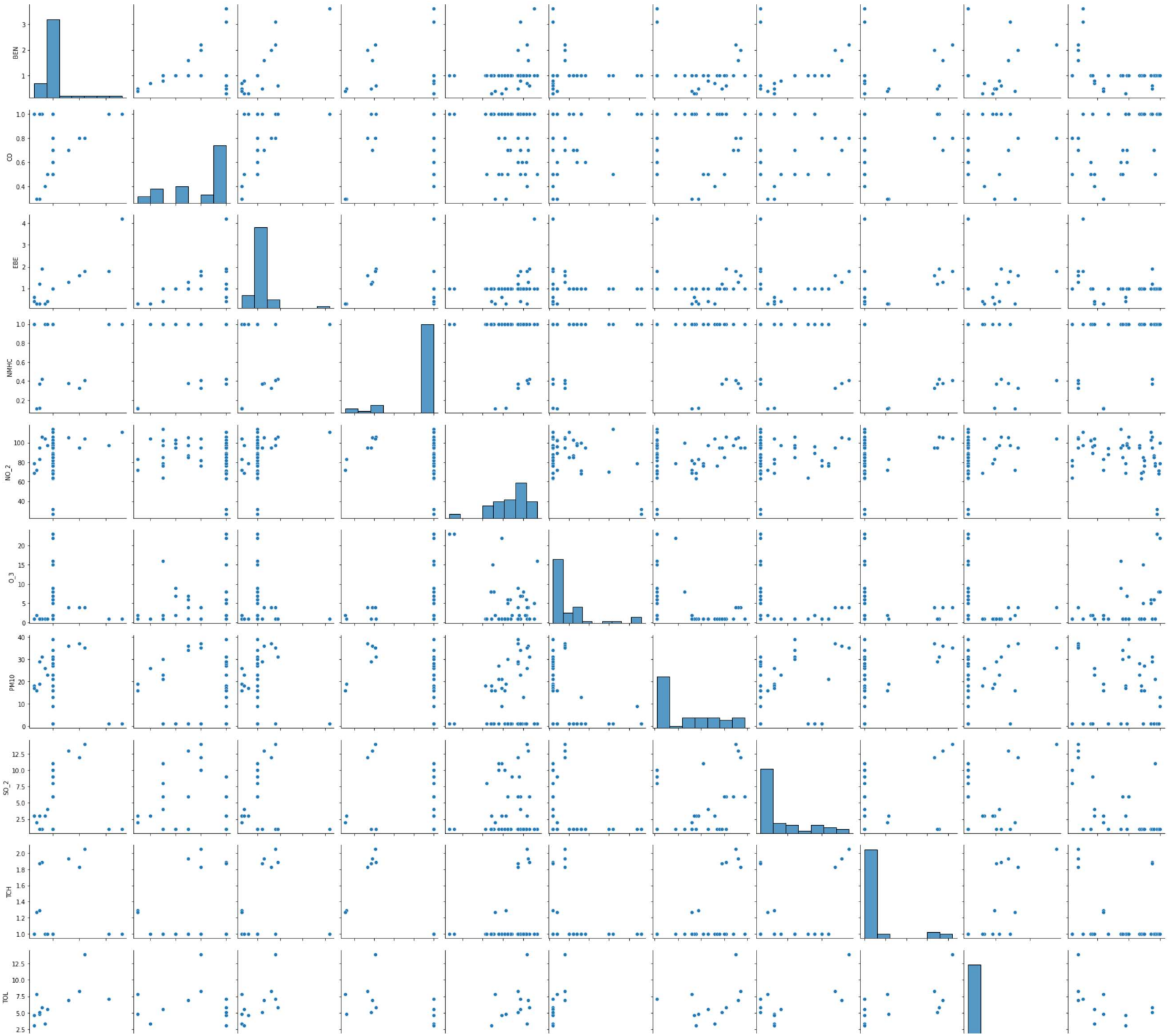
In [17]:

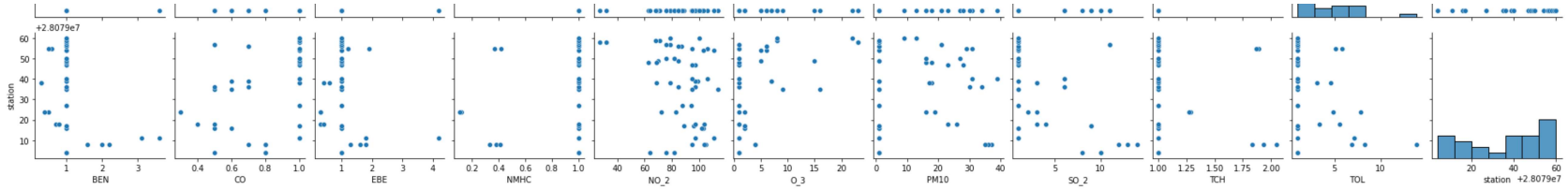
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [18]:
```
1 sns.pairplot(df1[0:50])
```

Out[18]:
```
<seaborn.axisgrid.PairGrid at 0x20301ec96d0>
```

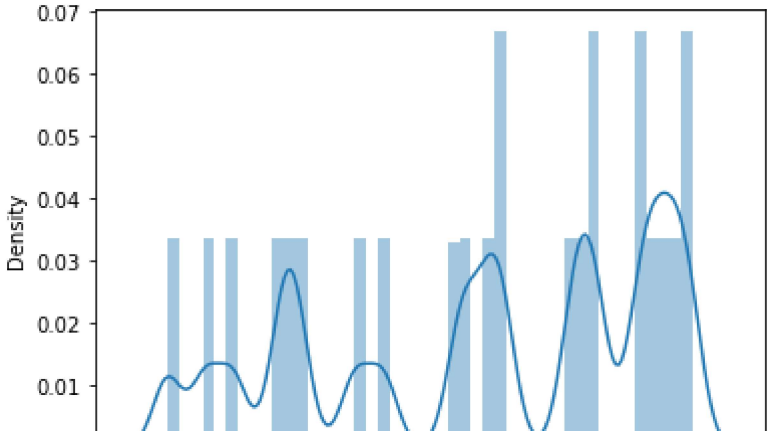

BEN
CO
EBE
NMHC
NO_2
O_3
PM10
SO_2
TCH
TOL



In [19]:
```
1 sns.distplot(df1['station'])
```

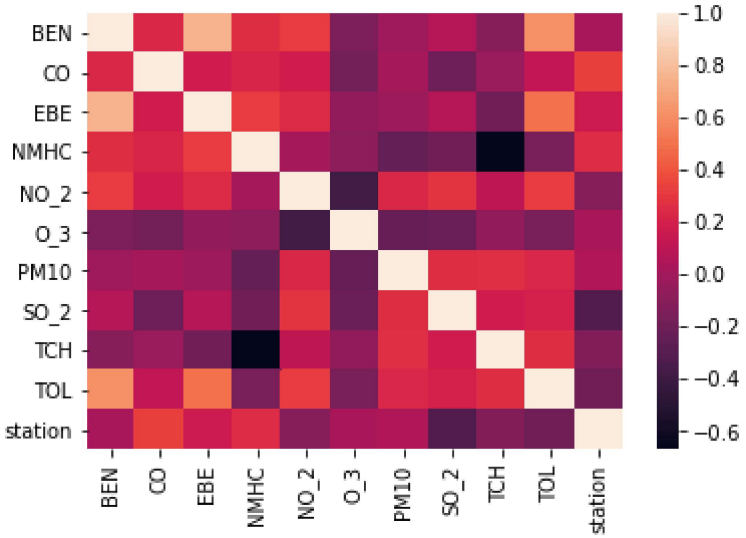
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function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

Out[19]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [20]:
```
1 sns.heatmap(df1.corr())
```

Out[20]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [21]:
```
1 x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
2        'PM10', 'SO_2', 'TCH', 'TOL']]
3 y=df['station']
```

In [22]:
```
1 from sklearn.model_selection import train_test_split
2 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]:
```
28079001.174014986
```

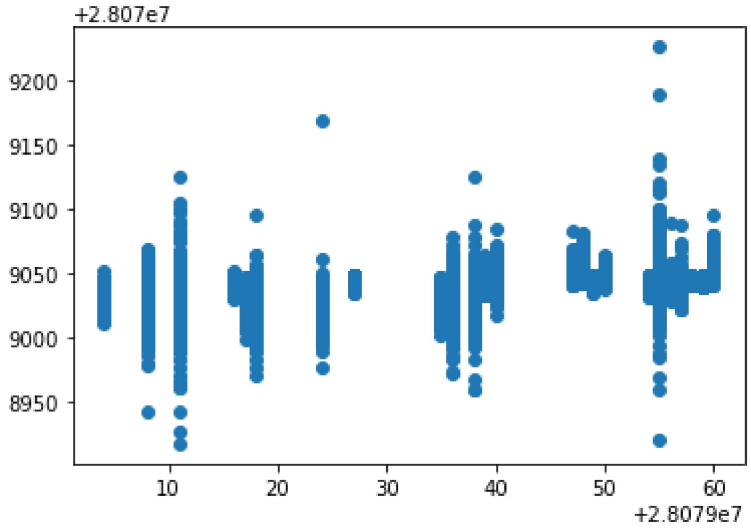
In [25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	-2.660254
CO	13.182744
EBE	13.851887
NMHC	10.320965
NO_2	-0.058883
O_3	0.012908
PM10	0.268334
SO_2	-0.969263
TCH	13.507064
TOL	-2.686796

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

Out[26]: <matplotlib.collections.PathCollection at 0x2030f597f70>



ACCURACY

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

Out[27]: 0.34911456095791904

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

Out[28]: 0.3479449527057812

Ridge and Lasso

In [29]:
```
1 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]: 0.349118748202651

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

Out[32]: 0.34794462246380553

In [33]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[33]: Lasso(alpha=10)

Accuracy(Lasso)

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

Out[34]: 0.10256230158121593

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

Out[35]: 0.10565688635021286

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([ 0.60135629,  2.06494749,  2.48508778,  0.59119365, -0.01861462,
       -0.00492022,  0.21558812, -1.24503397, -0.        , -1.41027512])
```

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

Out[38]:
```
28079037.959759142
```

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

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

Out[40]:
```
0.21870310308469598
```

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)))
```

```
13.07137681291404
242.67007677508545
15.577871381388583
```

Logistic Regression

In [42]:
```
from sklearn.linear_model import LogisticRegression
```

In [43]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [44]:
```
1 feature_matrix.shape
```

Out[44]:
```
(210096, 10)
```

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

Out[45]:
```
(210096,)
```

In [46]:
```
1 from sklearn.preprocessing import StandardScaler
```

In [47]:
```
1 fs=StandardScaler().fit_transform(feature_matrix)
```

In [48]:
```
1 logr=LogisticRegression(max_iter=10000)
2 logr.fit(fs,target_vector)
```

Out[48]:
```
LogisticRegression(max_iter=10000)
```

In [49]:
```
1 observation=[[1,2,3,4,5,6,7,8,9,10]]
```

In [50]:
```
1 prediction=logr.predict(observation)
2 print(prediction)
```

```
[28079018]
```

In [51]:
```
1 logr.classes_
```

Out[51]:
```
array([28079004, 28079008, 28079011, 28079016, 28079017, 28079018,
       28079024, 28079027, 28079035, 28079036, 28079038, 28079039,
       28079040, 28079047, 28079048, 28079049, 28079050, 28079054,
       28079055, 28079056, 28079057, 28079058, 28079059, 28079060],
      dtype=int64)
```

In [52]:
```
1 logr.score(fs,target_vector)
```

Out[52]:
```
0.6452050491204021
```

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

Out[53]:
```
1.7551372975127686e-190
```

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

Out[54]:
```
array([[1.75513730e-190, 1.26044787e-002, 1.32155783e-145,
        1.36891602e-127, 2.00525473e-069, 9.87395521e-001,
        6.42471645e-023, 5.53127694e-175, 2.07322318e-075,
        1.47224950e-065, 9.25290244e-036, 2.10198147e-133,
        1.22263020e-058, 4.67879125e-178, 6.17753358e-179,
        6.39931353e-175, 2.18050082e-183, 1.11012086e-167,
        6.26236550e-078, 3.29777962e-127, 2.36273295e-064,
        6.39796272e-184, 2.01416729e-179, 1.25963405e-078]])
```

Random Forest

In [55]:
```
1 from sklearn.ensemble import RandomForestClassifier
```

In [56]:
```
1 rfc=RandomForestClassifier()
2 rfc.fit(x_train,y_train)
```

Out[56]:
```
RandomForestClassifier()
```

In [57]:
```
1 parameters={'max_depth':[1,2,3,4,5],
2             'min_samples_leaf':[5,10,15,20,25],
3             'n_estimators':[10,20,30,40,50]
4 }
```

In [58]:
```
1 from sklearn.model_selection import GridSearchCV
2 grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="accuracy")
3 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]:
```
1 grid_search.best_score_
```

Out[59]:
```
0.6529609411306818
```

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','e','f','g','h','i
```

Out[61]:
```
[Text(2397.3333333333335, 1993.2, 'TCH <= 1.005\ngini = 0.958\nsamples = 92920\nvalue = [6141, 6101, 6217,
6234, 6028, 5991, 6225, 6033, 6182\n6075, 6075, 6309, 6101, 5962, 6123, 6199, 6081, 6023\n6184, 6207, 6129,
6231, 6095, 6121]\nclass = l'),
 Text(1322.6666666666667, 1630.8000000000002, 'O_3 <= 1.5\ngini = 0.953\nsamples = 81532\nvalue = [6141, 8
2, 6217, 6234, 6028, 5991, 96, 6033, 6182\n6075, 6075, 6309, 6101, 5962, 6123, 6199, 6081, 6023\n210, 6207,
6129, 6231, 6095, 6121]\nclass = l'),
 Text(661.3333333333334, 1268.4, 'PM10 <= 1.5\ngini = 0.899\nsamples = 36626\nvalue = [6141, 61, 6217, 168,
809, 548, 48, 425, 70, 6075\n6075, 54, 6101, 5962, 6123, 187, 6081, 23, 210, 54\n6129, 97, 37, 85]\nclass =
c'),
 Text(330.6666666666667, 906.0, 'EBE <= 0.95\ngini = 0.705\nsamples = 10400\nvalue = [6141, 52, 6217, 168,
809, 10, 40, 425, 70, 270, 81\n54, 70, 115, 122, 187, 854, 23, 11, 54, 382, 97\n37, 2]\nclass = c'),
 Text(165.33333333333334, 543.5999999999999, 'EBE <= 0.15\ngini = 0.021\nsamples = 3014\nvalue = [0, 0, 469
4, 0, 0, 0, 0, 0, 0, 0, 51, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = c'),
 Text(82.66666666666667, 181.19999999999982, 'gini = 0.06\nsamples = 804\nvalue = [0, 0, 1225, 0, 0, 0, 0,
0, 0, 0, 39, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = c'),
 Text(248.0, 181.19999999999982, 'gini = 0.007\nsamples = 2210\nvalue = [0, 0, 3469, 0, 0, 0, 0, 0, 0, 0, 1
2, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = c'),
 Text(496.0, 543.5999999999999, 'BEN <= 1.05\ngini = 0.685\nsamples = 7386\nvalue = [6141, 52, 1523, 168, 8
09, 10, 40, 425, 70, 270, 30\n54, 70, 115, 122, 187, 854, 23, 11, 54, 382, 97\n37, 2]\nclass = a'),
```

Conclusion

Accuracy

Linear Regression:0.3479449527057812

Ridge Regression:0.34794462246380553

Lasso Regression:0.10256230158121593

ElasticNet Regression:0.21870310308469598

Logistic Regression:0.6452050491204021

Random Forest:0.6529609411306818

Random Forest is suitable for this dataset