20104169 - SUMESH R

Importing Libraries

In [1]:

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

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
df=pd.read_csv("/content/drive/MyDrive/mydatasets/csvs_per_year/madrid_2014.csv")
df
```

```
Mounted at /content/drive
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	2014-06-01 01:00:00	NaN	0.2	NaN	NaN	3.0	10.0	NaN	NaN	NaN	3.0	NaN	NaN	28079004
1	2014-06-01 01:00:00	0.2	0.2	0.1	0.11	3.0	17.0	68.0	10.0	5.0	5.0	1.36	1.3	28079008
2	2014-06-01 01:00:00	0.3	NaN	0.1	NaN	2.0	6.0	NaN	NaN	NaN	NaN	NaN	1.1	28079011
3	2014-06-01 01:00:00	NaN	0.2	NaN	NaN	1.0	6.0	79.0	NaN	NaN	NaN	NaN	NaN	28079016
4	2014-06-01 01:00:00	NaN	NaN	NaN	NaN	1.0	6.0	75.0	NaN	NaN	4.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210019	2014-09-01 00:00:00	NaN	0.5	NaN	NaN	20.0	84.0	29.0	NaN	NaN	NaN	NaN	NaN	28079056
210020	2014-09-01 00:00:00	NaN	0.3	NaN	NaN	1.0	22.0	NaN	15.0	NaN	6.0	NaN	NaN	28079057
210021	2014-09-01 00:00:00	NaN	NaN	NaN	NaN	1.0	13.0	70.0	NaN	NaN	NaN	NaN	NaN	28079058
210022	2014-09-01 00:00:00	NaN	NaN	NaN	NaN	3.0	38.0	42.0	NaN	NaN	NaN	NaN	NaN	28079059
210023	2014-09-01 00:00:00	NaN	NaN	NaN	NaN	1.0	26.0	65.0	11.0	NaN	NaN	NaN	NaN	28079060

210024 rows × 14 columns

Data Cleaning and Data Preprocessing

In [3]:

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

In [4]:

```
df.columns
```

Out[4]:

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

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13946 entries, 1 to 210006
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     13946 non-null  object 
 1   BEN      13946 non-null  float64
 2   CO       13946 non-null  float64
 3   EBE      13946 non-null  float64
 4   NMHC     13946 non-null  float64
 5   NO       13946 non-null  float64
 6   NO_2     13946 non-null  float64
 7   O_3      13946 non-null  float64
 8   PM10     13946 non-null  float64
 9   PM25     13946 non-null  float64
 10  SO_2     13946 non-null  float64
 11  TCH      13946 non-null  float64
 12  TOL      13946 non-null  float64
 13  station  13946 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.6+ MB
```

In [6]:

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

Out[6]:

	CO	station
1	0.2	28079008
6	0.2	28079024
25	0.2	28079008
30	0.2	28079024
49	0.2	28079008
...	...	...
209958	0.2	28079024
209977	0.7	28079008
209982	0.2	28079024
210001	0.4	28079008
210006	0.2	28079024

13946 rows × 2 columns

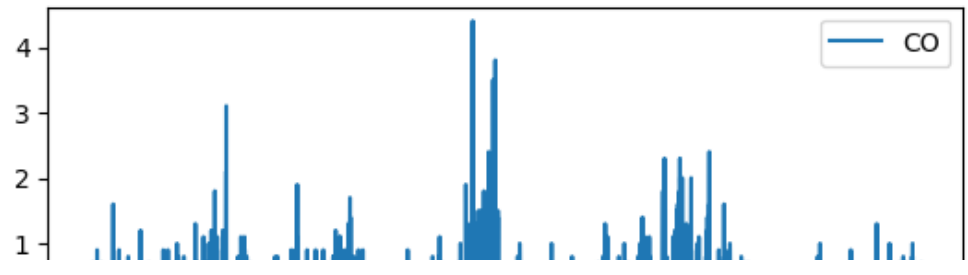
Line chart

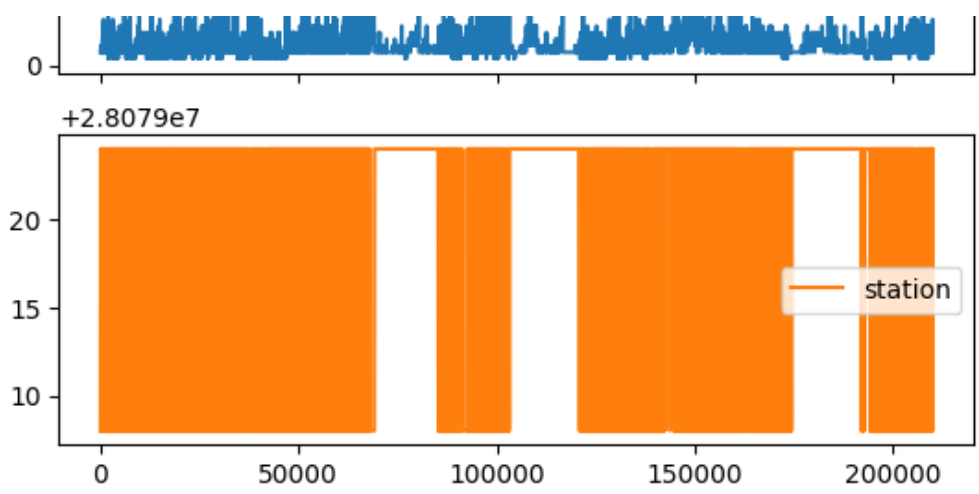
In [7]:

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

Out[7]:

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





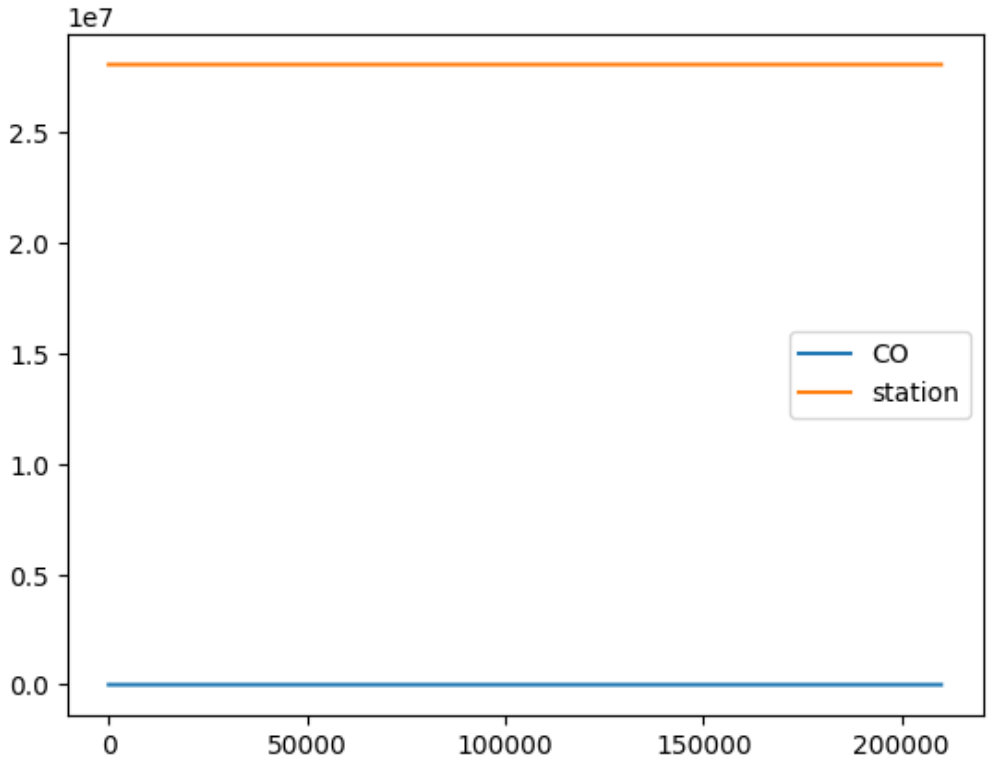
Line chart

In [8]:

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

Out[8]:

<Axes: >



Bar chart

In [9]:

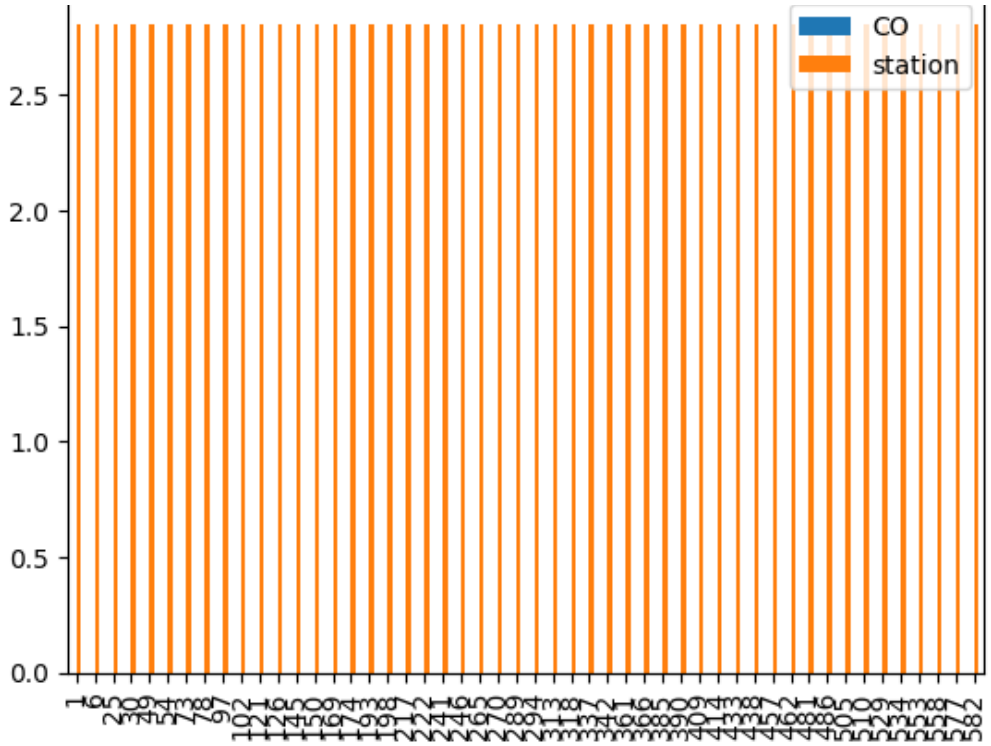
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<Axes: >



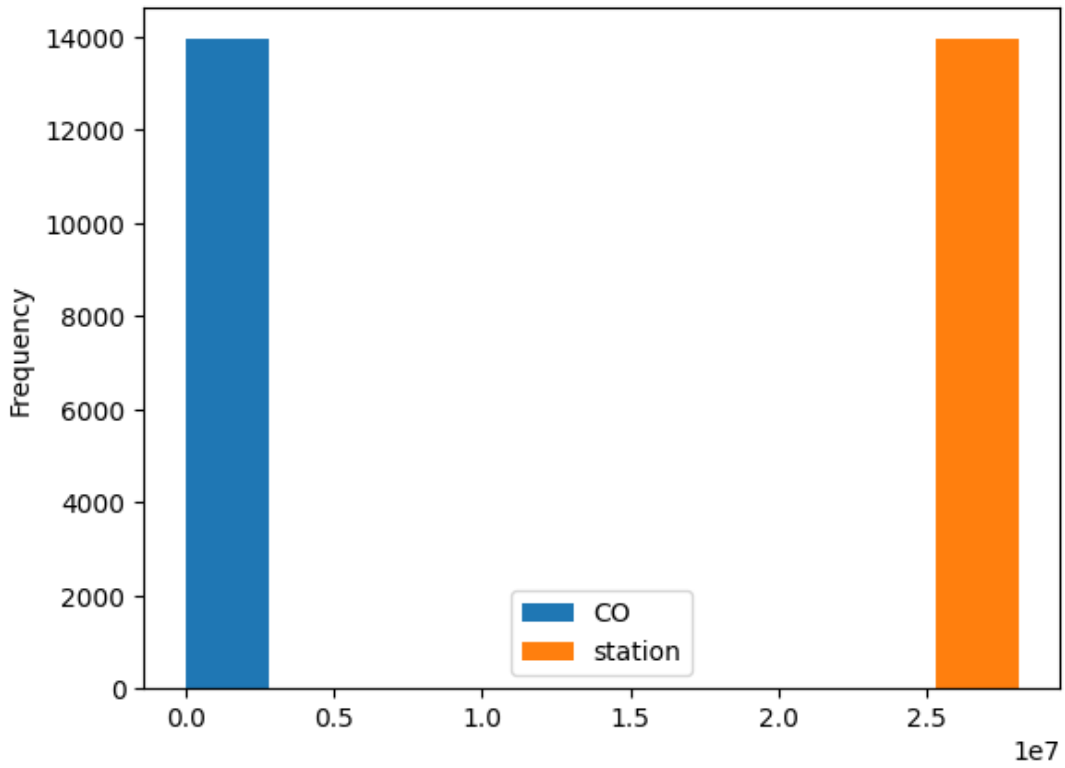
Histogram

In [11]:

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

Out[11]:

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



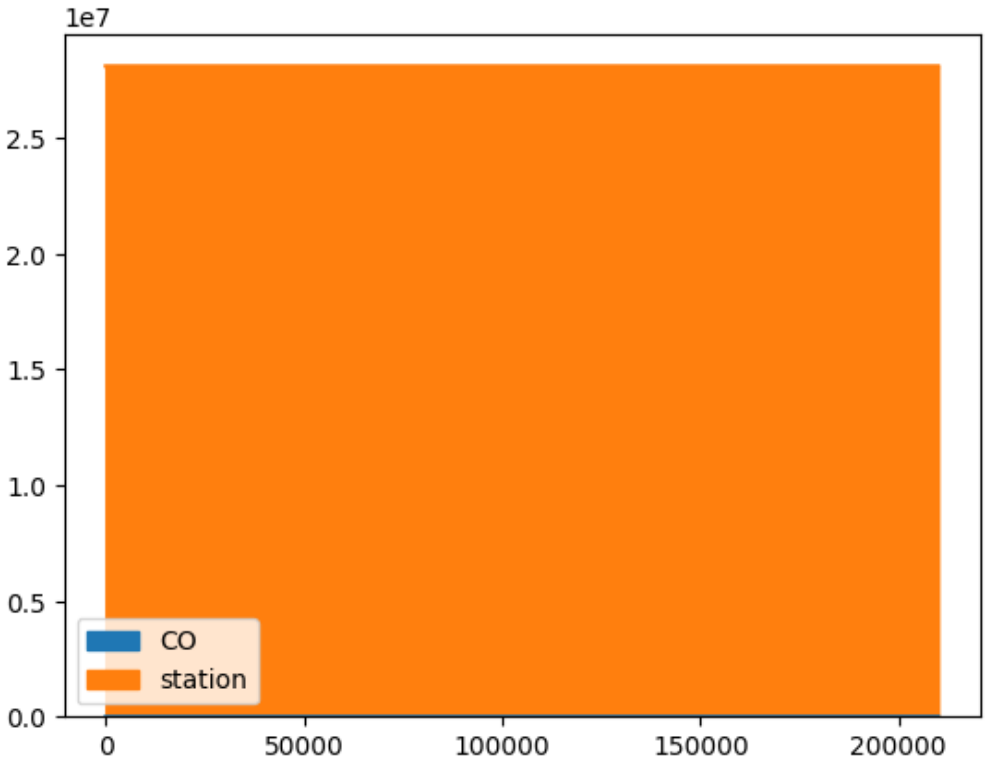
Area chart

In [12]:

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

Out[12]:

<Axes: >



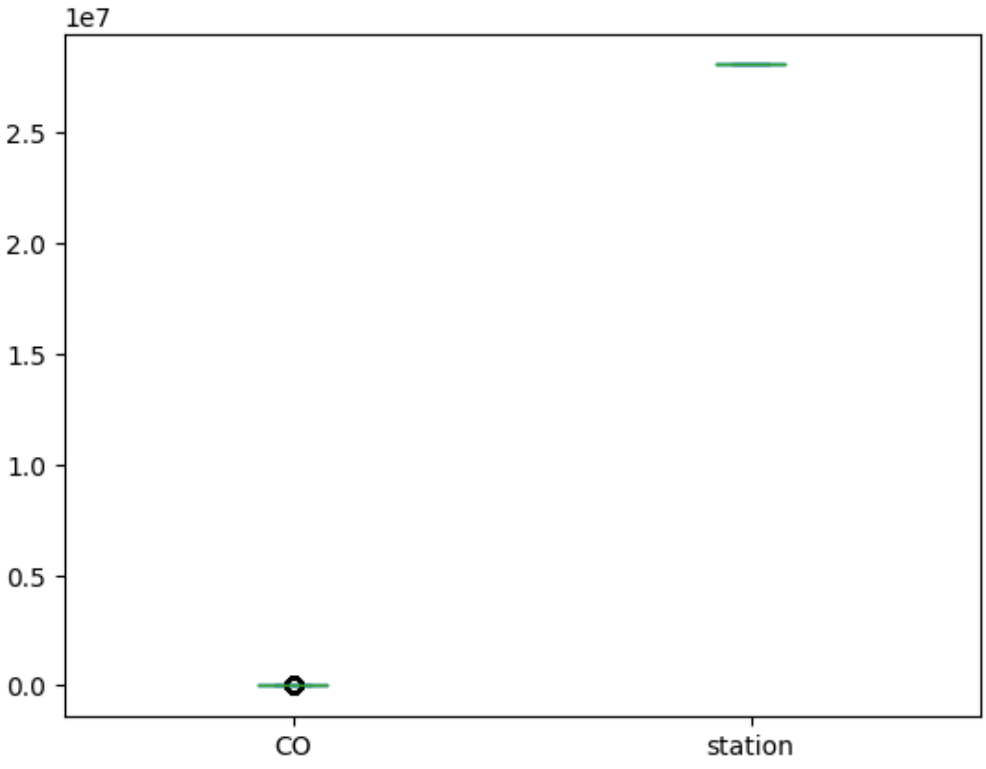
Box chart

In [13]:

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

Out[13]:

<Axes: >



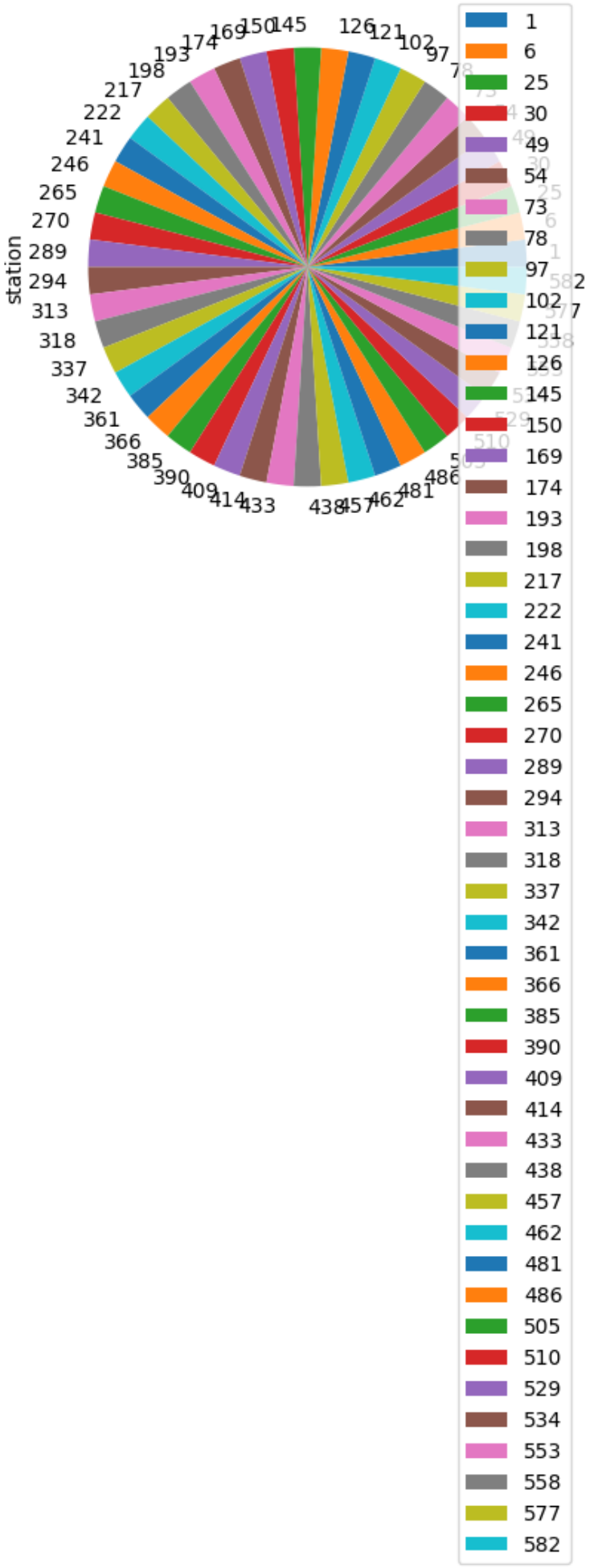
Pie chart

In [14]:

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

Out[14]:

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



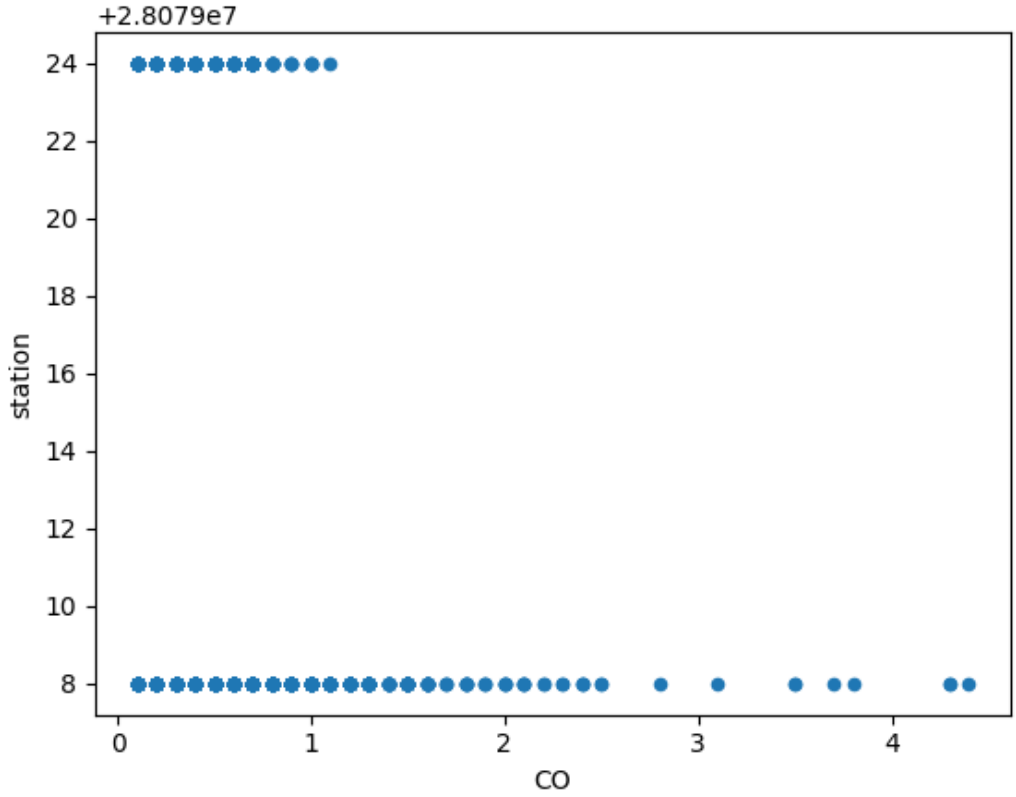
Scatter chart

In [15]:

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

Out[15]:

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



In [16]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13946 entries, 1 to 210006
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     13946 non-null  object 
 1   BEN      13946 non-null  float64
 2   CO       13946 non-null  float64
 3   EBE      13946 non-null  float64
 4   NMHC     13946 non-null  float64
 5   NO       13946 non-null  float64
 6   NO_2     13946 non-null  float64
 7   O_3      13946 non-null  float64
 8   PM10     13946 non-null  float64
 9   PM25     13946 non-null  float64
 10  SO_2     13946 non-null  float64
 11  TCH      13946 non-null  float64
 12  TOL      13946 non-null  float64
 13  station  13946 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.6+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10

	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	
count	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000	139
mean	0.375921	0.314793	0.306016	0.222302	17.589129	34.240929	53.082389	19.858526	
std	0.555093	0.207375	0.635475	0.082403	39.432216	30.654229	33.488167	15.613506	
min	0.100000	0.100000	0.100000	0.060000	1.000000	1.000000	1.000000	1.000000	
25%	0.100000	0.200000	0.100000	0.160000	1.000000	10.000000	25.000000	10.000000	
50%	0.200000	0.300000	0.100000	0.230000	4.000000	27.000000	53.000000	16.000000	
75%	0.400000	0.400000	0.300000	0.260000	18.000000	51.000000	75.000000	25.000000	
max	9.400000	4.400000	16.200001	1.290000	725.000000	346.000000	220.000000	197.000000	1

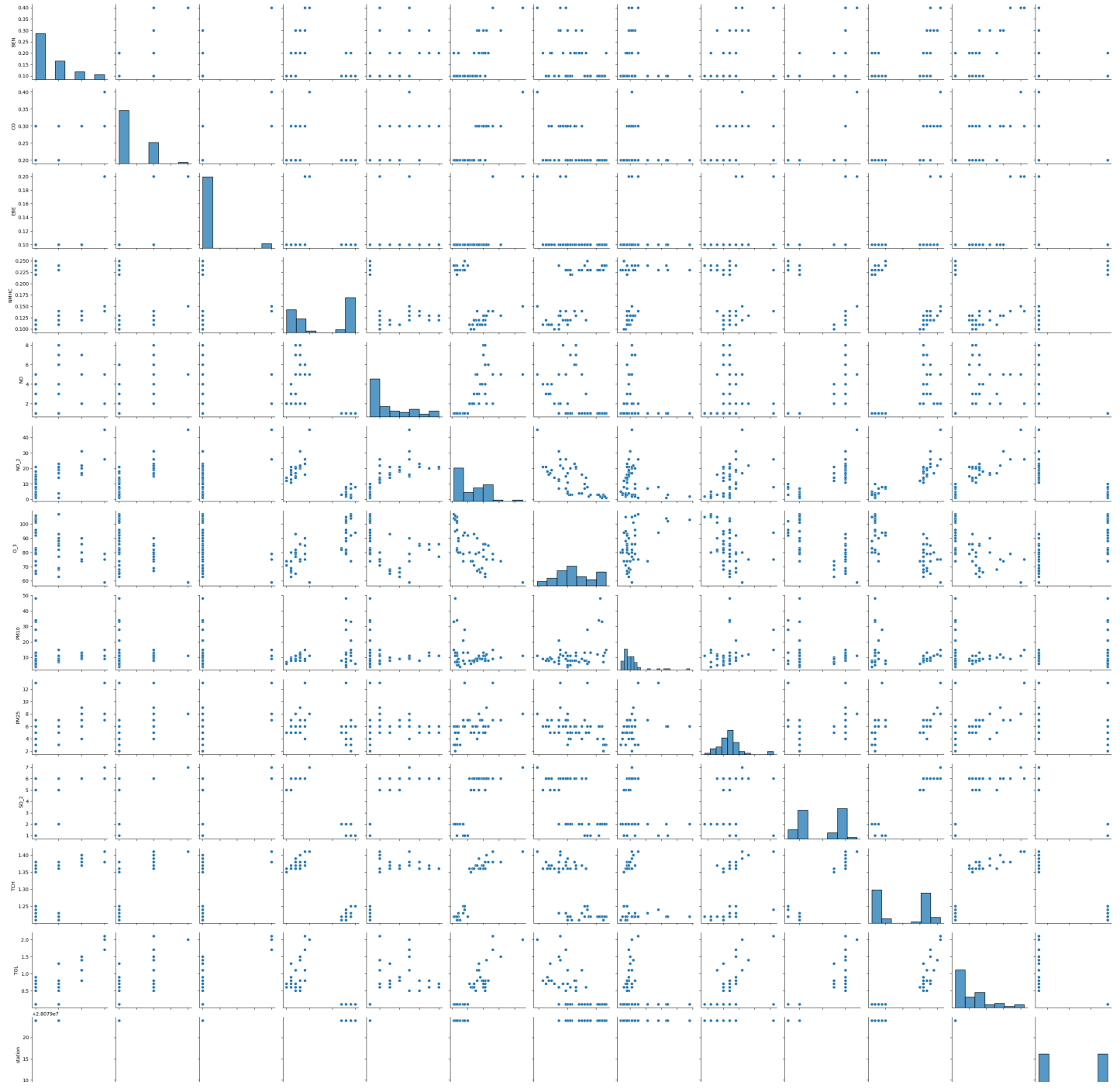
EDA AND VISUALIZATION

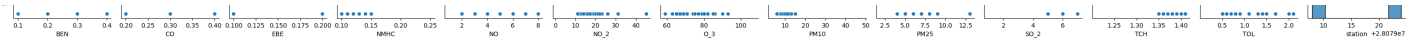
In [18]:

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

Out[18]:

```
<seaborn.axisgrid.PairGrid at 0x7c56f7838a90>
```





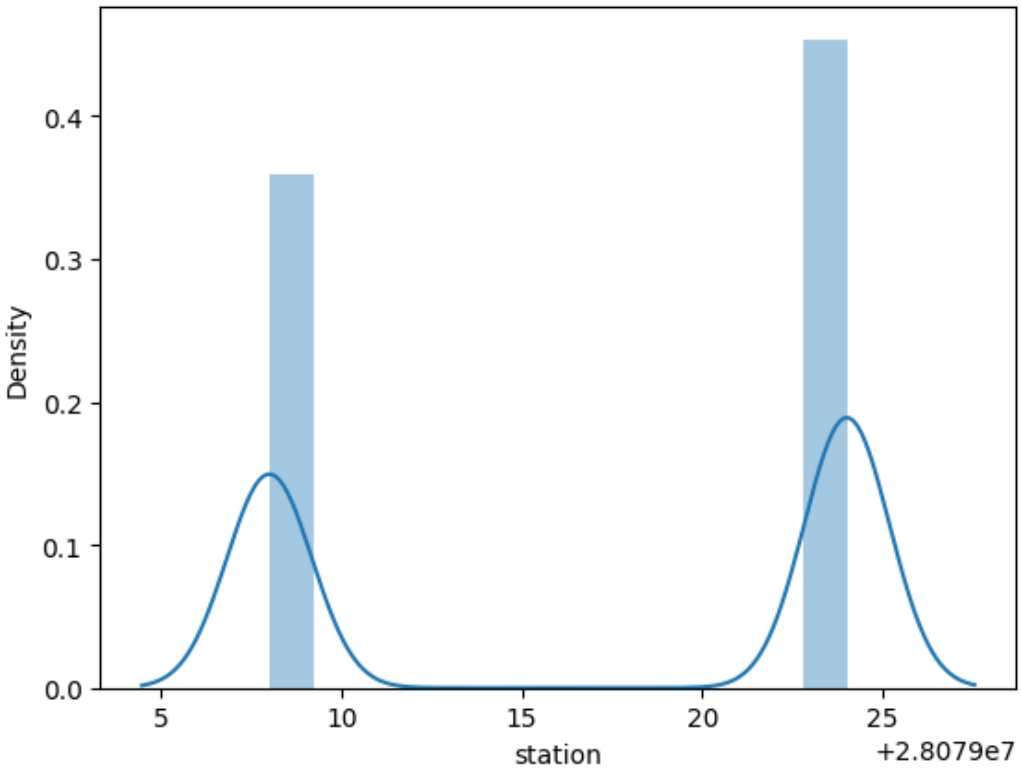
In [19]:

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

```
<ipython-input-19-6e2460d4583e>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

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

Out[19]:

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



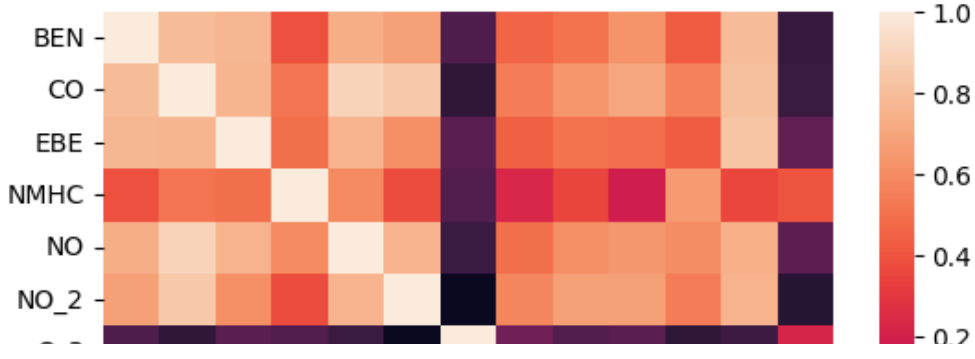
In [20]:

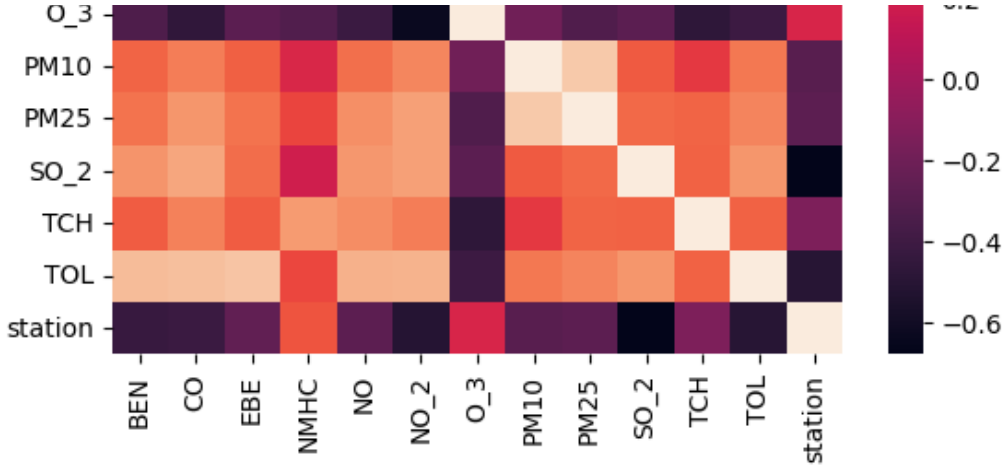
```
sns.heatmap(df.corr())
```

```
<ipython-input-20-aa4f4450a243>:1: FutureWarning: The default value of numeric_only in Da
taFrame.corr is deprecated. In a future version, it will default to False. Select only va
lid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(df.corr())
```

Out[20]:

```
<Axes: >
```





TO TRAIN THE MODEL AND MODEL BULDING

In [21]:

```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [22]:

```
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 [23]:

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

Out[23]:

▾ LinearRegression

LinearRegression()

In [24]:

```
lr.intercept_
```

Out[24]:

```
28079025.058953818
```

In [25]:

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

Out[25]:

	Co-efficient
BEN	-1.644805
CO	-8.959343
EBE	0.067268
NMHC	80.555308
NO	0.034487

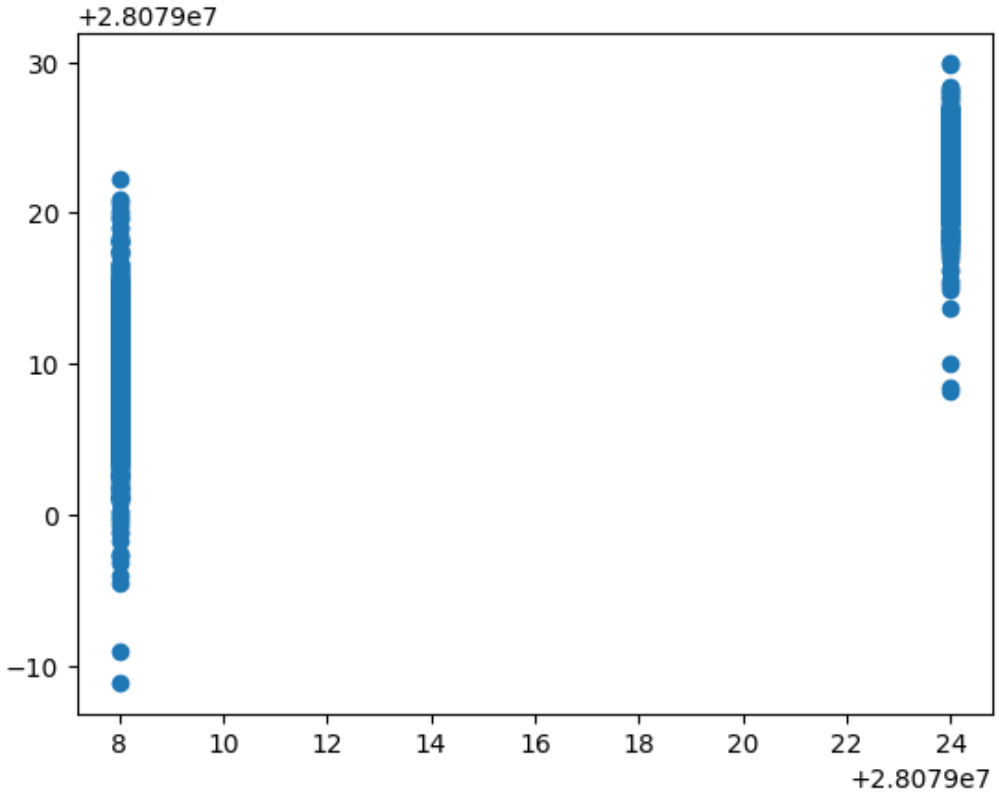
NO_2	**Coefficient**
O_3	**0.002819**
PM10	**-0.015507**
PM25	**0.106487**
SO_2	**-0.946053**
TCH	**-12.877728**
TOL	**-0.389355**

In [26]:

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

Out[26]:

```
<matplotlib.collections.PathCollection at 0x7c5739a30f10>
```



ACCURACY

In [27]:

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

Out[27]:

```
0.8893644128547189
```

In [28]:

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

Out[28]:

```
0.8916002720360299
```

Ridge and Lasso

In [ ]:

In [29]:

```
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]:

```
0.8645573359210382
```

In [32]:

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

Out[32]:

```
0.8689133208550619
```

In [33]:

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

Out[33]:

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

In [34]:

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

Out[34]:

```
0.30209027468061755
```

Accuracy(Lasso)

In [35]:

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

Out[35]:

```
0.2873813991872316
```

In [36]:

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

Out[36]:

▾ ElasticNet

ElasticNet()

In [37]:

```
en.coef_
```

Out[37]:

```
array([-0.00000000e+00, -0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        1.07103137e-01, -1.06771465e-01, -1.50489637e-02, -3.51822203e-04,
        6.51613371e-02, -1.47563281e+00,  0.00000000e+00, -5.23441253e-01])
```

In [38]:

```
en.intercept_
```

Out[38]:

```
28079026.78798942
```

In [39]:

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

In [40]:

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

Out[40]:

```
0.5864023545281206
```

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)))
```

```
4.172936021137526
26.16676886813824
5.115346407442828
```

Logistic Regression

In [42]:

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

In [43]:

```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [44]:

```
feature_matrix.shape
```

Out[44]:

```
(13946, 12)
```

In [45]:

```
target_vector.shape
```

Out[45]:

```
(13946,)
```

In [46]:

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

In [47]:

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

In [48]:

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

Out[48]:

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

In [49]:

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

In [50]:

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

```
[28079008]
```

In [51]:

```
logr.classes_
```

Out[51]:

```
array([28079008, 28079024])
```

In [52]:

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

Out[52]:

```
0.9930446006023232
```

In [53]:

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

Out[53]:

```
1.0
```

In [54]:

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

Out[54]:

```
array([[1.00000000e+00, 1.92542107e-22]])
```

Random Forest

In [55]:

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

In [56]:

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

Out[56]:

▾ RandomForestClassifier

RandomForestClassifier()

In [57]:

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

In [58]:

```
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[58]:

▸ GridSearchCV

▸ estimator: RandomForestClassifier

▸ RandomForestClassifier

In [59]:

```
grid_search.best_score_
```

Out[59]:

```
0.9960049170251998
```

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'],filled=True)
```

Out[61]:

```
[Text(0.359375, 0.9166666666666666, 'NMHC <= 0.185\ngini = 0.494\nsamples = 6205\nvalue = [4350, 5412]\nclass = b'),
 Text(0.125, 0.75, 'SO_2 <= 2.5\ngini = 0.044\nsamples = 1817\nvalue = [2790, 64]\nclass = a'),
 Text(0.0625, 0.5833333333333334, 'NMHC <= 0.125\ngini = 0.359\nsamples = 53\nvalue = [19, 62]\nclass = b'),
 Text(0.03125, 0.4166666666666667, 'gini = 0.0\nsamples = 13\nvalue = [19, 0]\nclass = a'),
 Text(0.09375, 0.4166666666666667, 'gini = 0.0\nsamples = 40\nvalue = [0, 62]\nclass = b'),
 Text(0.1875, 0.5833333333333334, 'NO_2 <= 7.5\ngini = 0.001\nsamples = 1764\nvalue = [2771, 2]\nclass = a'),
 Text(0.15625, 0.4166666666666667, 'gini = 0.375\nsamples = 6\nvalue = [6, 2]\nclass = a'),
 Text(0.21875, 0.4166666666666667, 'gini = 0.0\nsamples = 1758\nvalue = [2765, 0]\nclass
```

```
= a'),
 Text(0.59375, 0.75, 'TOL <= 1.85\ngini = 0.35\nsamples = 4388\nvalue = [1560, 5348]\ncla
ss = b'),
 Text(0.390625, 0.5833333333333334, 'BEN <= 0.25\ngini = 0.063\nsamples = 3276\nvalue = [
168, 4967]\nclass = b'),
 Text(0.28125, 0.4166666666666667, 'BEN <= 0.15\ngini = 0.01\nsamples = 3116\nvalue = [25
, 4860]\nclass = b'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 2090\nvalue = [0, 3264]\nclass = b'),
 Text(0.3125, 0.25, 'SO_2 <= 4.5\ngini = 0.03\nsamples = 1026\nvalue = [25, 1596]\nclass
= b'),
 Text(0.28125, 0.08333333333333333, 'gini = 0.001\nsamples = 1004\nvalue = [1, 1581]\ncla
ss = b'),
 Text(0.34375, 0.08333333333333333, 'gini = 0.473\nsamples = 22\nvalue = [24, 15]\nclass
= a'),
 Text(0.5, 0.4166666666666667, 'CO <= 0.25\ngini = 0.49\nsamples = 160\nvalue = [143, 107
]\nclass = a'),
 Text(0.4375, 0.25, 'NMHC <= 0.215\ngini = 0.078\nsamples = 63\nvalue = [4, 94]\nclass =
b'),
 Text(0.40625, 0.08333333333333333, 'gini = 0.444\nsamples = 5\nvalue = [4, 2]\nclass = a
'),
 Text(0.46875, 0.08333333333333333, 'gini = 0.0\nsamples = 58\nvalue = [0, 92]\nclass = b
'),
 Text(0.5625, 0.25, 'NO_2 <= 26.5\ngini = 0.156\nsamples = 97\nvalue = [139, 13]\nclass =
a'),
 Text(0.53125, 0.08333333333333333, 'gini = 0.331\nsamples = 31\nvalue = [34, 9]\nclass =
a'),
 Text(0.59375, 0.08333333333333333, 'gini = 0.071\nsamples = 66\nvalue = [105, 4]\nclass
= a'),
 Text(0.796875, 0.5833333333333334, 'TCH <= 1.335\ngini = 0.337\nsamples = 1112\nvalue =
[1392, 381]\nclass = a'),
 Text(0.71875, 0.4166666666666667, 'SO_2 <= 3.5\ngini = 0.497\nsamples = 54\nvalue = [44,
38]\nclass = a'),
 Text(0.6875, 0.25, 'BEN <= 0.15\ngini = 0.05\nsamples = 25\nvalue = [1, 38]\nclass = b')
,
 Text(0.65625, 0.08333333333333333, 'gini = 0.0\nsamples = 15\nvalue = [0, 23]\nclass = b
'),
 Text(0.71875, 0.08333333333333333, 'gini = 0.117\nsamples = 10\nvalue = [1, 15]\nclass =
b'),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 29\nvalue = [43, 0]\nclass = a'),
 Text(0.875, 0.4166666666666667, 'BEN <= 0.25\ngini = 0.323\nsamples = 1058\nvalue = [134
8, 343]\nclass = a'),
 Text(0.8125, 0.25, 'SO_2 <= 3.5\ngini = 0.092\nsamples = 198\nvalue = [15, 294]\nclass =
b'),
 Text(0.78125, 0.08333333333333333, 'gini = 0.0\nsamples = 127\nvalue = [0, 199]\nclass =
b'),
 Text(0.84375, 0.08333333333333333, 'gini = 0.236\nsamples = 71\nvalue = [15, 95]\nclass
= b'),
 Text(0.9375, 0.25, 'O_3 <= 3.5\ngini = 0.068\nsamples = 860\nvalue = [1333, 49]\nclass =
a'),
 Text(0.90625, 0.08333333333333333, 'gini = 0.0\nsamples = 12\nvalue = [0, 20]\nclass = b
'),
 Text(0.96875, 0.08333333333333333, 'gini = 0.042\nsamples = 848\nvalue = [1333, 29]\ncla
ss = a')]
```

Conclusion

Accuracy

In [63]:

```
print("Linear Regression:",lr.score(x_test,y_test))
print("Ridge Regression:",rr.score(x_test,y_test))
print("Lasso Regression",la.score(x_test,y_test))
print("ElasticNet Regression:",en.score(x_test,y_test))
print("Logistic Regression:",logr.score(fs,target_vector))
print("Random Forest:",grid_search.best_score_)
```

```
Linear Regression: 0.8893644128547189
```

```
Ridge Regression: 0.8645573359210382
Lasso Regression 0.2873813991872316
ElasticNet Regression: 0.5864023545281206
Logistic Regression: 0.9930446006023232
Random Forest: 0.9960049170251998
```

Random Forest is suitable for this dataset