Importing Libraries

In [1]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [2]:
```python
df=pd.read_csv("madrid_2018").fillna(1)
df
```

Out[2]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	2018-03-01 01:00:00	1.0	1.00	0.3	1.0	1.00	1.0	29.0	31.0	1.0	1.0	1.0	2.0	1.00	1.0	28079004
1	2018-03-01 01:00:00	0.5	1.39	0.3	0.2	0.02	6.0	40.0	49.0	52.0	5.0	4.0	3.0	1.41	0.8	28079008
2	2018-03-01 01:00:00	0.4	1.00	1.0	0.2	1.00	4.0	41.0	47.0	1.0	1.0	1.0	1.0	1.00	1.1	28079011
3	2018-03-01 01:00:00	1.0	1.00	0.3	1.0	1.00	1.0	35.0	37.0	54.0	1.0	1.0	1.0	1.00	1.0	28079016
4	2018-03-01 01:00:00	1.0	1.00	1.0	1.0	1.00	1.0	27.0	29.0	49.0	1.0	1.0	3.0	1.00	1.0	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
69091	2018-02-01 00:00:00	1.0	1.00	0.5	1.0	1.00	66.0	91.0	192.0	1.0	35.0	22.0	1.0	1.00	1.0	28079056
69092	2018-02-01 00:00:00	1.0	1.00	0.7	1.0	1.00	87.0	107.0	241.0	1.0	29.0	1.0	15.0	1.00	1.0	28079057
69093	2018-02-01 00:00:00	1.0	1.00	1.0	1.0	1.00	28.0	48.0	91.0	2.0	1.0	1.0	1.0	1.00	1.0	28079058
69094	2018-02-01 00:00:00	1.0	1.00	1.0	1.0	1.00	141.0	103.0	320.0	2.0	1.0	1.0	1.0	1.00	1.0	28079059
69095	2018-02-01 00:00:00	1.0	1.00	1.0	1.0	1.00	69.0	96.0	202.0	3.0	26.0	1.0	1.0	1.00	1.0	28079060

69096 rows × 16 columns

Data Cleaning and Data Preprocessing

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

Out[3]:
```
Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69096 entries, 0 to 69095
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     69096 non-null  object 
 1   BEN      69096 non-null  float64
 2   CH4      69096 non-null  float64
 3   CO       69096 non-null  float64
 4   EBE      69096 non-null  float64
 5   NMHC     69096 non-null  float64
 6   NO       69096 non-null  float64
 7   NO_2     69096 non-null  float64
 8   NOx      69096 non-null  float64
 9   O_3      69096 non-null  float64
 10  PM10     69096 non-null  float64
 11  PM25     69096 non-null  float64
 12  SO_2     69096 non-null  float64
 13  TCH      69096 non-null  float64
 14  TOL      69096 non-null  float64
 15  station  69096 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 8.4+ MB
```

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

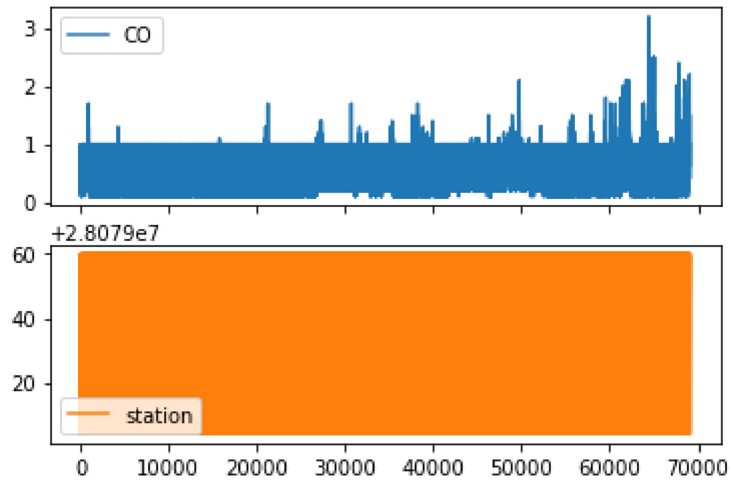
Out[5]:

	CO	station
0	0.3	28079004
1	0.3	28079008
2	1.0	28079011
3	0.3	28079016
4	1.0	28079017
...	...	...
69091	0.5	28079056
69092	0.7	28079057
69093	1.0	28079058
69094	1.0	28079059
69095	1.0	28079060

69096 rows × 2 columns

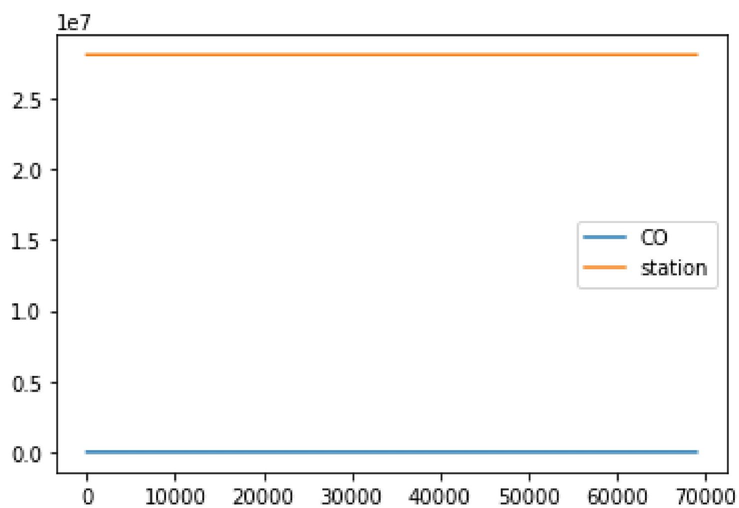
Line chart

In [6]:
```
data.plot.line(subplots=True)
```
Out[6]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

In [7]:
```
data.plot.line()
```
Out[7]: <AxesSubplot:>

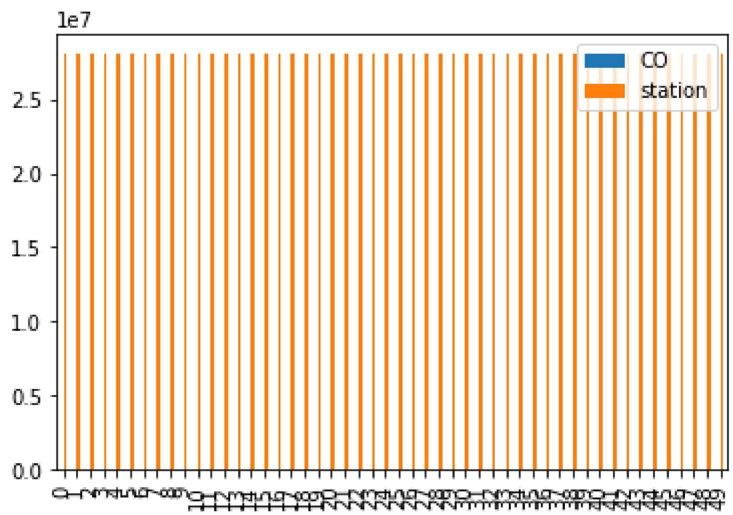


Bar chart

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

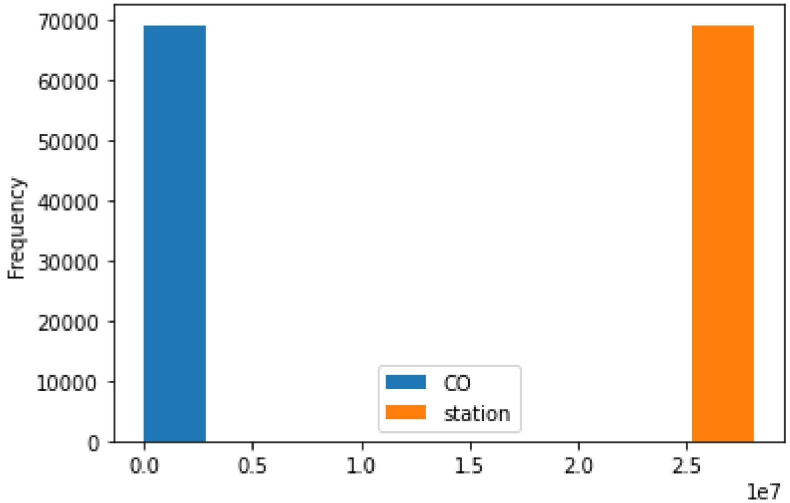
In [9]:
```
b.plot.bar()
```

Out[9]:
```
<AxesSubplot:>
```



Histogram

In [10]:
```
1 data.plot.hist()
```
Out[10]: <AxesSubplot:ylabel='Frequency'>

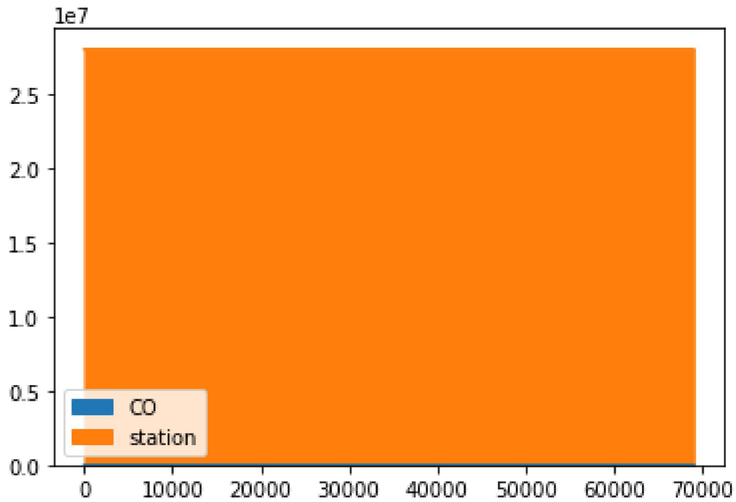


Area chart

In [11]:

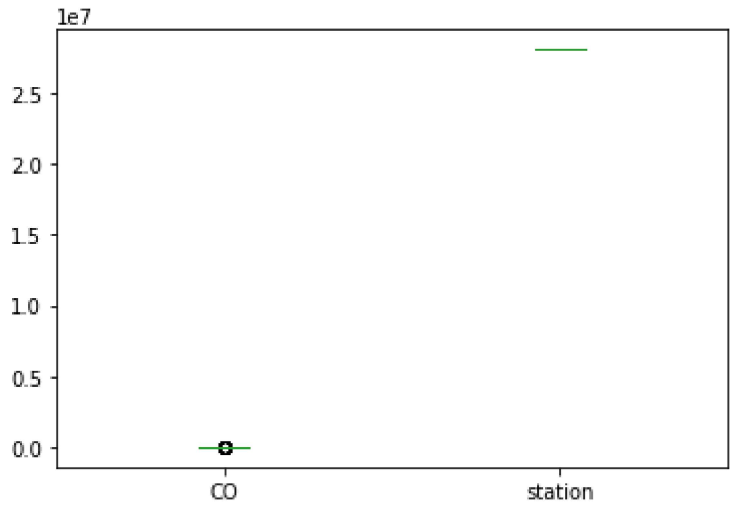
```
data.plot.area()
```

Out[11]: <AxesSubplot:>



Box chart

In [12]:
```
data.plot.box()
```
Out[12]: <AxesSubplot:>



Pie chart

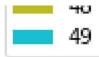
In [13]:

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

Out[13]:

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

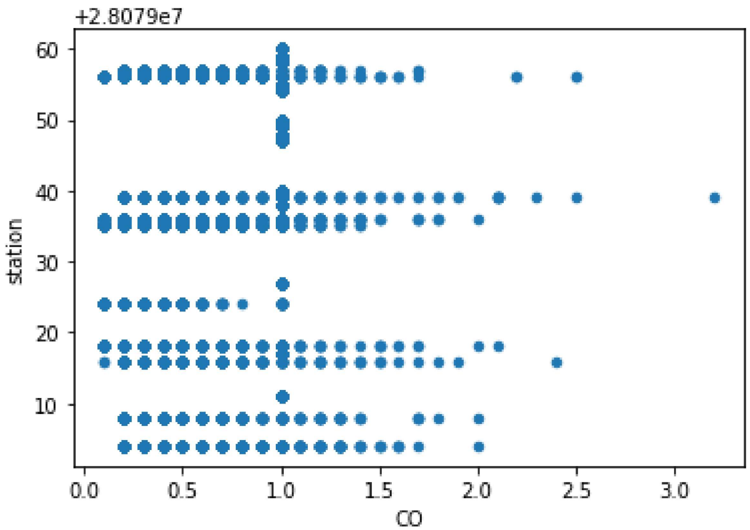
station
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47



Scatter chart

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

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



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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69096 entries, 0 to 69095
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   date    69096 non-null  object 
 1   BEN     69096 non-null  float64
 2   CH4     69096 non-null  float64
 3   CO      69096 non-null  float64
 4   EBE     69096 non-null  float64
 5   NMHC    69096 non-null  float64
 6   NO      69096 non-null  float64
 7   NO_2    69096 non-null  float64
 8   NOx     69096 non-null  float64
 9   O_3     69096 non-null  float64
 10  PM10    69096 non-null  float64
 11  PM25    69096 non-null  float64
 12  SO_2    69096 non-null  float64
 13  TCH     69096 non-null  float64
```

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

Out[16]:

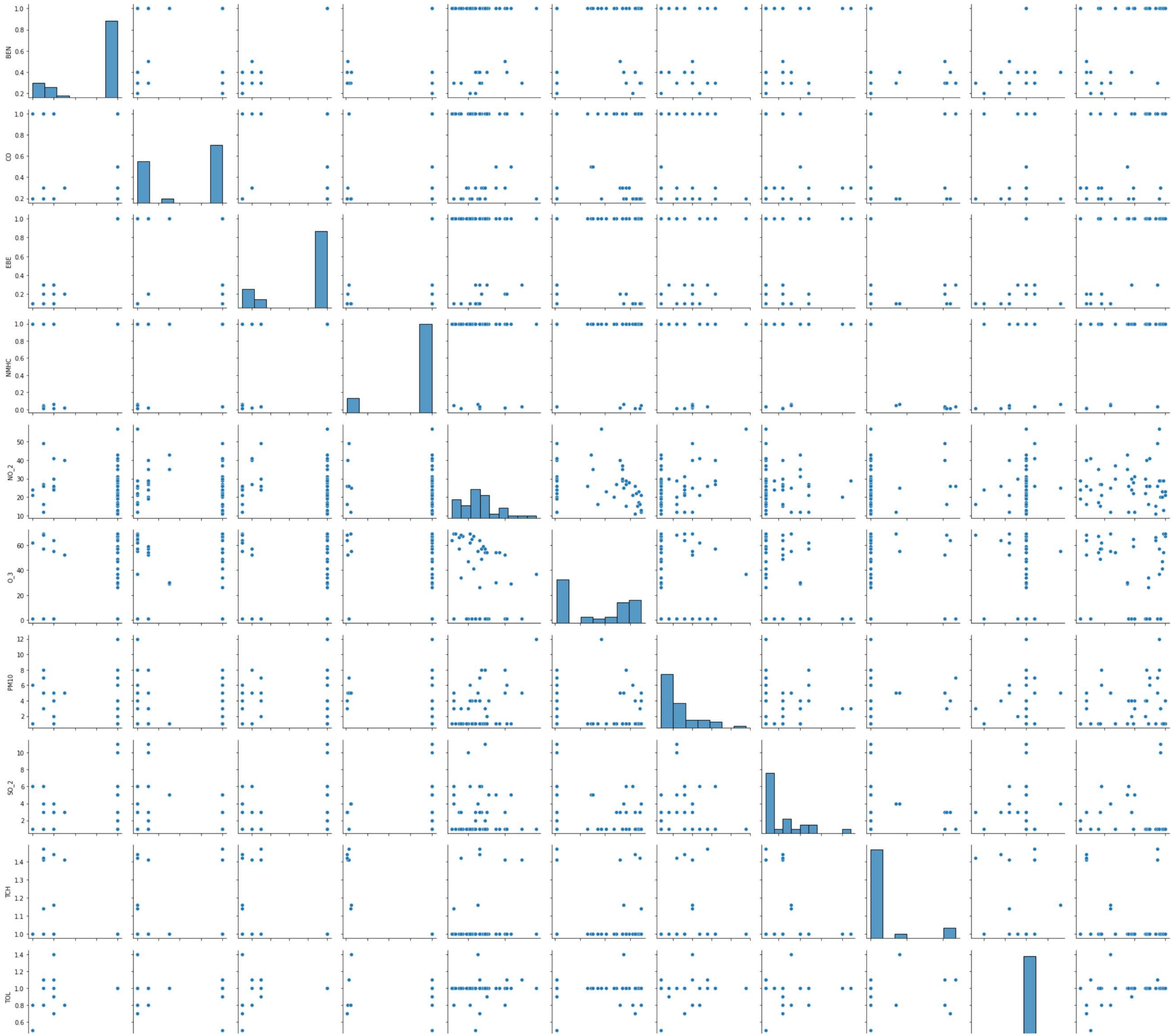
	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3
count	69096.000000	69096.000000	69096.000000	69096.000000	69096.000000	69096.000000	69096.000000	69096.000000	69096.000000
mean	0.891049	1.034859	0.728669	0.828423	0.885822	19.819425	38.485151	68.870123	26.423932
std	0.295475	0.114176	0.348123	0.360883	0.306434	40.579600	28.532232	85.230974	30.707335
min	0.100000	0.020000	0.100000	0.100000	0.000000	1.000000	1.000000	1.000000	1.000000
25%	1.000000	1.000000	0.300000	1.000000	1.000000	1.000000	16.000000	20.000000	1.000000
50%	1.000000	1.000000	1.000000	1.000000	1.000000	5.000000	32.000000	41.000000	7.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	18.000000	55.000000	84.000000	53.000000
max	8.400000	3.920000	3.200000	14.900000	1.000000	774.000000	276.000000	1422.000000	133.000000

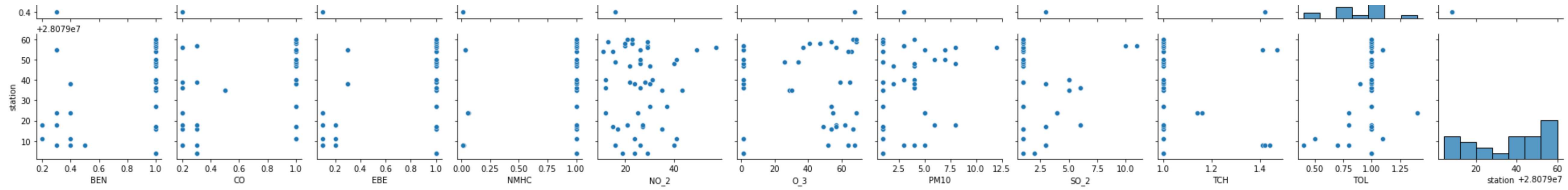
In [17]:
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
        'PM10', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

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

Out[18]: <seaborn.axisgrid.PairGrid at 0x1ac405261c0>

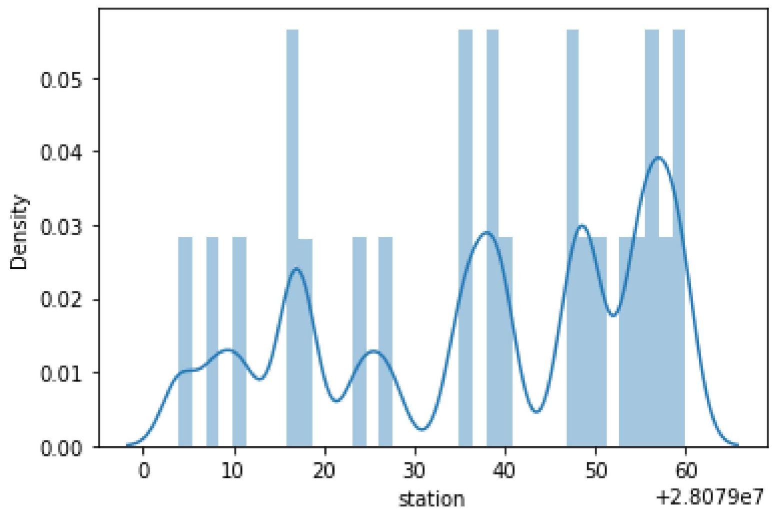

BEN
CO
EBE
NMHC
NO_2
O_3
PM10
SO_2
TCH
TOL



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

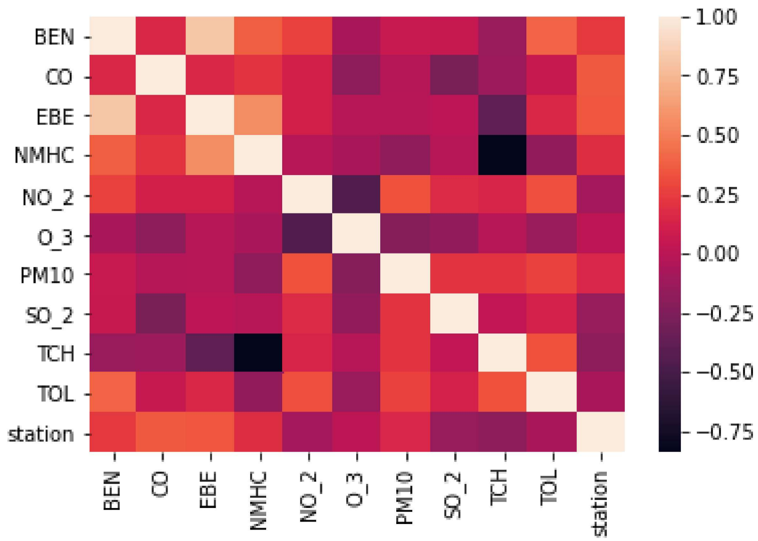
```
(a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

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



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

Out[20]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [21]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', '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()

In [24]:

```
lr.intercept_
```

Out[24]: 28079055.860388223

In [25]:

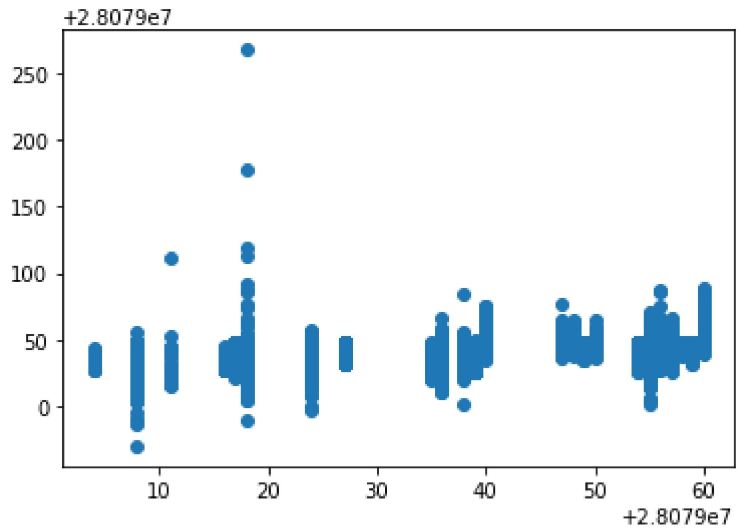
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	3.542848
CO	17.357554
EBE	17.943335
NMHC	-15.918930
NO_2	-0.118764
O_3	0.004224
PM10	0.451379
SO_2	-0.338340
TCH	-28.980378
TOL	-2.165710

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

Out[26]: <matplotlib.collections.PathCollection at 0x1ac48443e80>



ACCURACY

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

Out[27]: 0.3338200820046149

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

Out[28]: 0.3426886551551521

Ridge and Lasso

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]: 0.3338245601758344

In [32]:
```
rr.score(x_train,y_train)
```
Out[32]: 0.34265154106894136

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]: 0.04608736375660516

In [35]:
```
la.score(x_test,y_test)
```
Out[35]: 0.04747180808354623

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([ 1.29364991,  2.47466501,  2.72316987,  0.72303036, -0.10131896,
       -0.01457371,  0.40012824, -0.68643046, -0.        , -0.68969608])
```

In [38]:
```
en.intercept_
```
Out[38]:
```
28079035.814630482
```

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

In [40]:
```
en.score(x_test,y_test)
```
Out[40]:
```
0.1604545384821412
```

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)))
```
```
13.705578675622542
259.34240055641703
16.10411129359261
```

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]:
```
feature_matrix.shape
```

Out[44]:
```
(69096, 10)
```

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

Out[45]:
```
(69096,)
```

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]]
```

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

```
[28079008]
```

In [51]:
```
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]:
```
logr.score(fs,target_vector)
```

Out[52]:
```
0.6794894060437652
```

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

Out[53]:
```
1.5372578801045127e-124
```

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

Out[54]:
```
array([[1.53725788e-124, 9.99999638e-001, 1.57962753e-119,
        5.70337989e-117, 1.17048776e-053, 3.61536912e-007,
        2.96025442e-016, 1.45984975e-121, 1.03922905e-062,
        5.13099564e-057, 1.86164835e-026, 1.43373175e-115,
        3.71763643e-056, 1.10272203e-126, 4.98061564e-129,
        1.12343211e-125, 2.50362413e-130, 5.41204368e-123,
        8.41883922e-063, 1.80233350e-069, 3.62069053e-058,
        4.29638443e-133, 1.96532180e-126, 4.86877434e-062]])
```

Random Forest

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

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

Out[56]:
```
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(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]:
```
grid_search.best_score_
```

Out[59]:
```
0.66880314404135
```

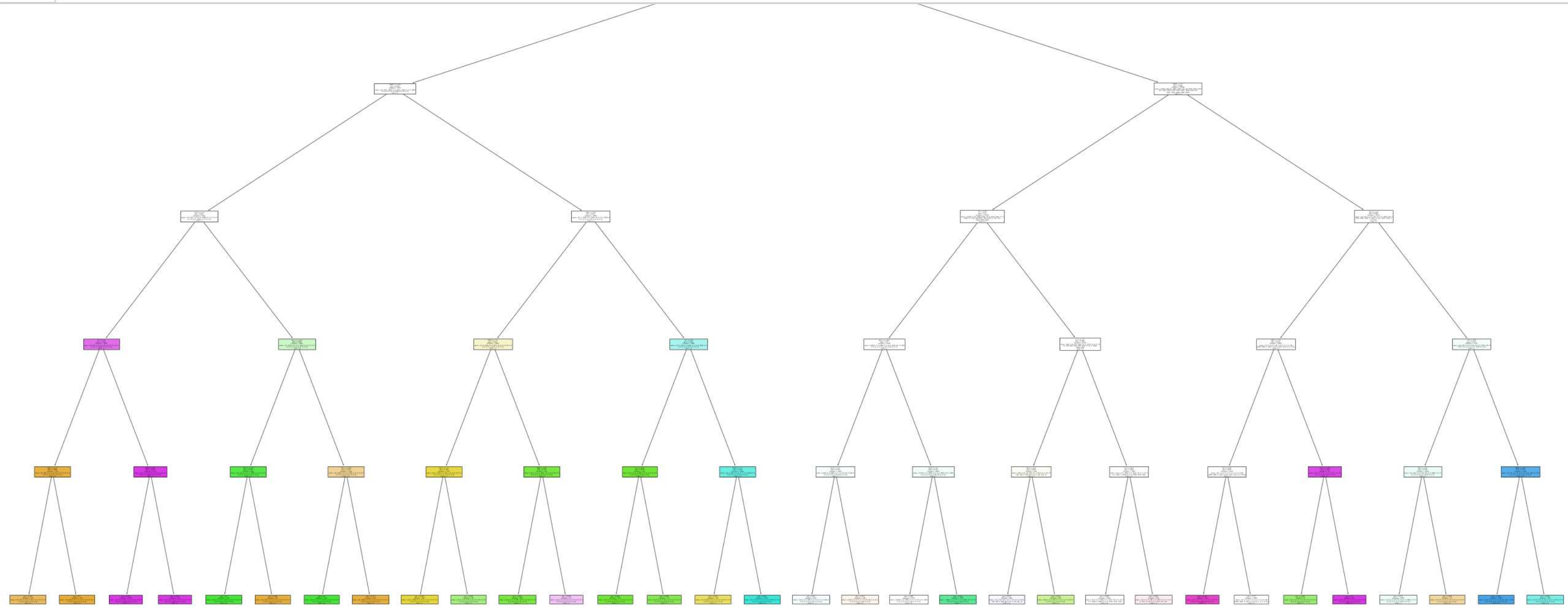
In [60]:

```
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
```



Conclusion

Accuracy

Linear Regression:0.3426886551551521

Ridge Regression:0.34265154106894136

Lasso Regression:0.046087363756605l6

ElasticNet Regression:0.16045453384821412

Logistic Regression:0.6794894060437652

Random Forest:0.66880314404135

Logistic Regression is suitable for this dataset

In [ ]:

```

```