mk 03/08/2023

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

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\C10_air\csvs_per_year\csvs_per_year\madrid_2018.csv")
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	NaN	NaN	0.3	NaN	NaN	1.0	29.0	31.0	NaN	NaN	NaN	2.0	NaN	NaN	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	NaN	NaN	0.2	NaN	4.0	41.0	47.0	NaN	NaN	NaN	NaN	NaN	1.1	28079011
3	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	35.0	37.0	54.0	NaN	NaN	NaN	NaN	NaN	28079016
4	2018-03-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	27.0	29.0	49.0	NaN	NaN	3.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
69091	2018-02-01 00:00:00	NaN	NaN	0.5	NaN	NaN	66.0	91.0	192.0	1.0	35.0	22.0	NaN	NaN	NaN	28079056
69092	2018-02-01 00:00:00	NaN	NaN	0.7	NaN	NaN	87.0	107.0	241.0	NaN	29.0	NaN	15.0	NaN	NaN	28079057
69093	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	28.0	48.0	91.0	2.0	NaN	NaN	NaN	NaN	NaN	28079058
69094	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	141.0	103.0	320.0	2.0	NaN	NaN	NaN	NaN	NaN	28079059
69095	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	69.0	96.0	202.0	3.0	26.0	NaN	NaN	NaN	NaN	28079060

69096 rows × 16 columns

In [3]:
```
df=df.dropna()
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4562 entries, 1 to 69078
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4562 non-null   object 
 1   BEN      4562 non-null   float64
 2   CH4      4562 non-null   float64
 3   CO       4562 non-null   float64
 4   EBE      4562 non-null   float64
 5   NMHC     4562 non-null   float64
 6   NO       4562 non-null   float64
 7   NO_2     4562 non-null   float64
 8   NOx      4562 non-null   float64
 9   O_3      4562 non-null   float64
 10  PM10     4562 non-null   float64
 11  PM25     4562 non-null   float64
 12  SO_2     4562 non-null   float64
 13  TCH      4562 non-null   float64
 14  TOL      4562 non-null   float64
 15  station  4562 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 605.9+ KB
```

In [6]:
```
1 data=df[['BEN', 'TOL', 'TCH']]
2 data
```

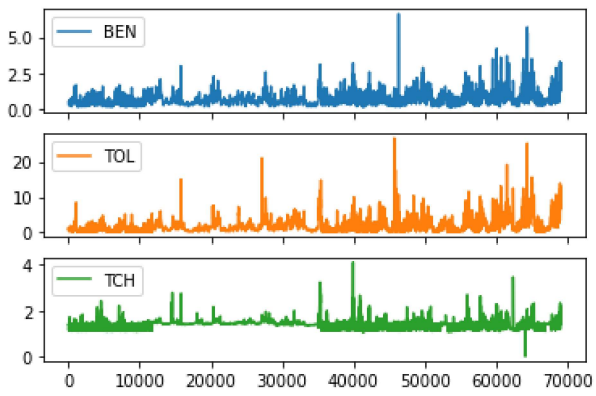
Out[6]:

	BEN	TOL	TCH
1	0.5	0.8	1.41
6	0.4	1.4	1.16
25	0.4	0.7	1.44
30	0.3	0.8	1.14
49	0.3	0.4	1.42
...	...	...	...
69030	1.8	11.9	1.40
69049	3.1	12.5	2.22
69054	1.6	10.3	1.32
69073	3.2	13.0	1.72
69078	1.3	6.8	1.24

4562 rows × 3 columns

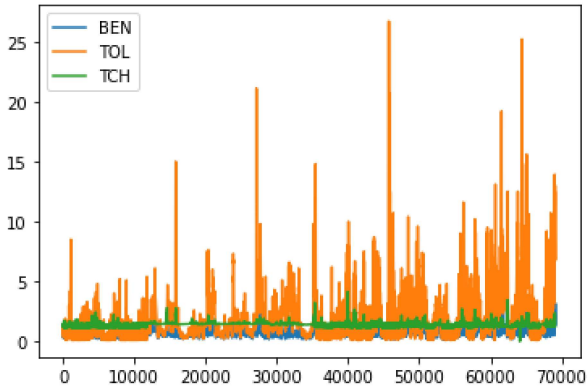
In [7]:
```
1 data.plot.line(subplots=True)
```

Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



In [8]:
```
1 data.plot.line()
```

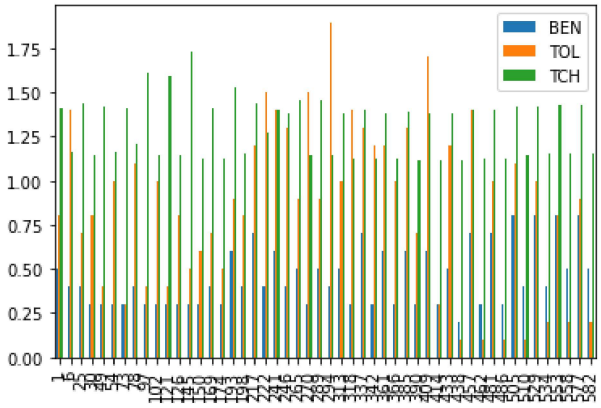
Out[8]: <AxesSubplot:>



In [9]:
```
1 b=data[0:50]
```

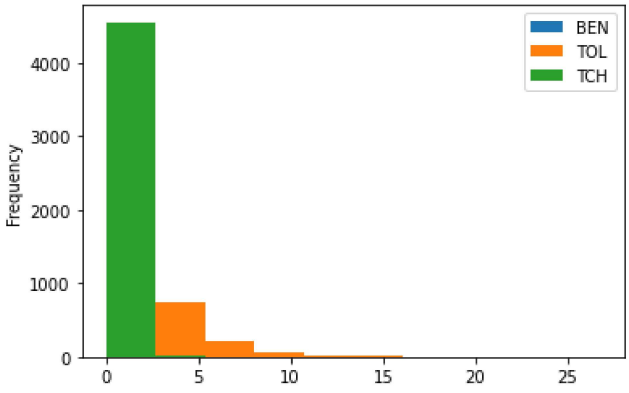
In [10]:
```
1 b.plot.bar()
```

Out[10]: <AxesSubplot:>



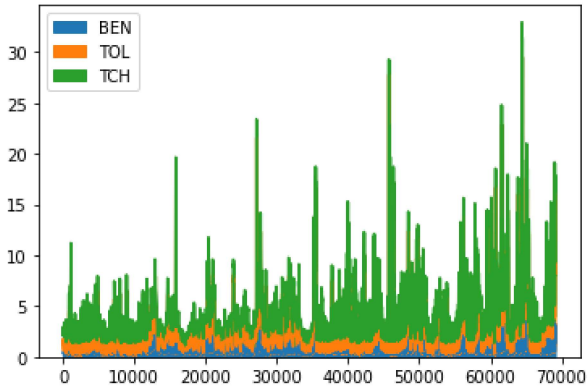
In [11]:
```
1 data.plot.hist()
```

Out[11]: <AxesSubplot:ylabel='Frequency'>



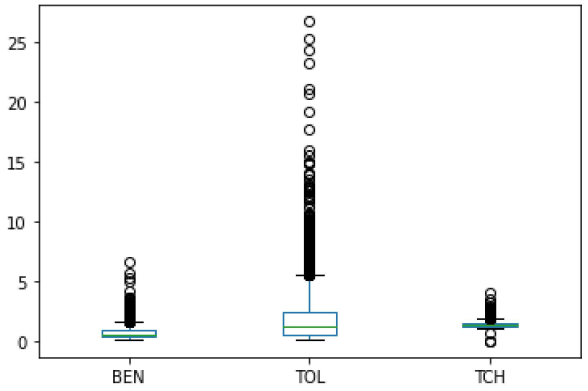
In [12]:
```
1 data.plot.area()
```

Out[12]: <AxesSubplot:>



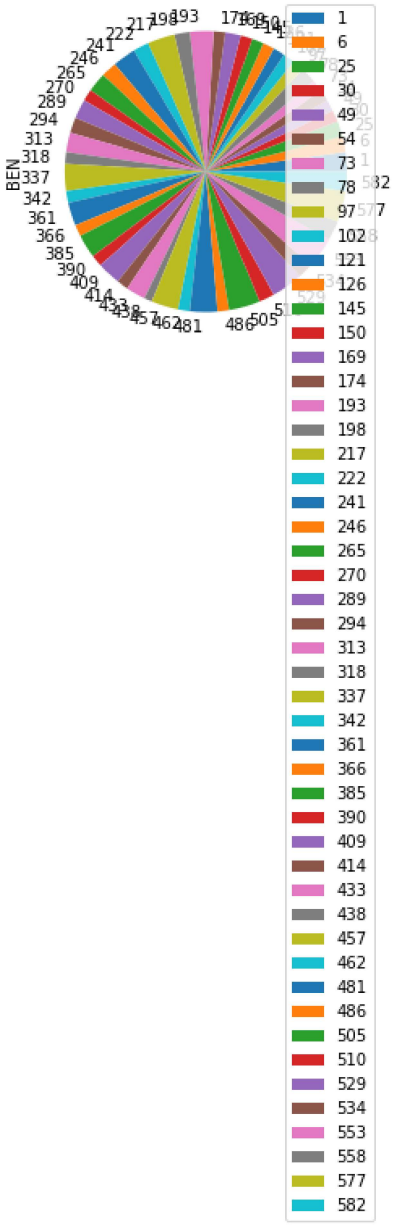
In [13]:
```
1 data.plot.box()
```

Out[13]: <AxesSubplot:>



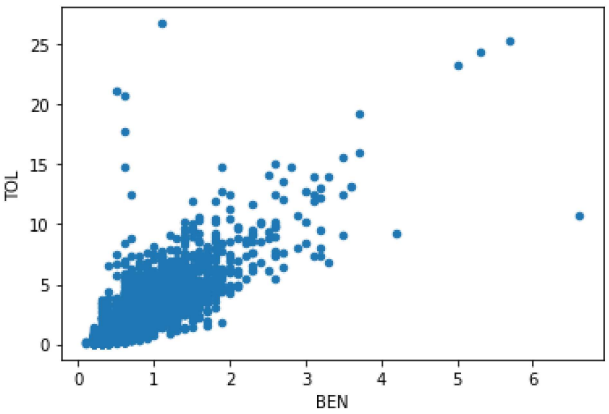
In [14]:
```
1 b.plot.pie(y='BEN' )
```

Out[14]: <AxesSubplot:ylabel='BEN'>



In [15]:
```
1 data.plot.scatter(x='BEN' ,y='TOL')
```

Out[15]: <AxesSubplot:xlabel='BEN', ylabel='TOL'>



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

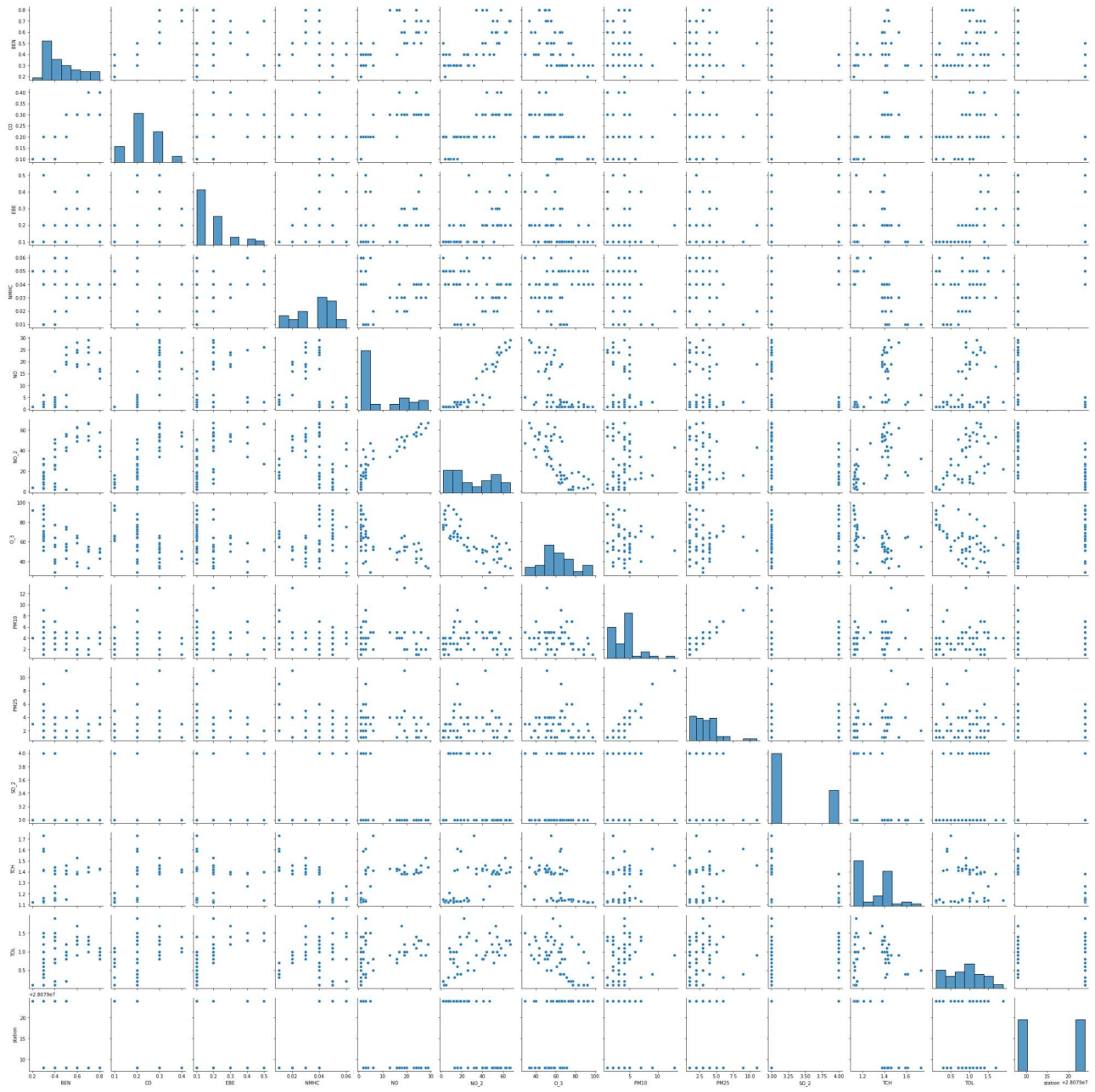
Out[16]:

	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	
count	4562.00000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	456
mean	0.69349	1.329163	0.330579	0.286782	0.056773	21.742218	44.152126	77.494739	41.279702	
std	0.46832	0.214399	0.161489	0.354442	0.037711	35.539531	30.234015	79.218558	26.298770	
min	0.10000	0.020000	0.100000	0.100000	0.000000	1.000000	1.000000	2.000000	1.000000	
25%	0.40000	1.120000	0.200000	0.100000	0.030000	1.000000	20.000000	24.000000	18.000000	
50%	0.60000	1.390000	0.300000	0.200000	0.050000	9.000000	41.000000	56.000000	42.000000	
75%	0.90000	1.420000	0.400000	0.300000	0.070000	27.000000	64.000000	106.000000	63.000000	
max	6.60000	3.920000	2.000000	7.400000	0.490000	431.000000	184.000000	844.000000	113.000000	

In [17]:
```
1 df1=df[['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
2 'SO_2', 'TCH', 'TOL', 'station']]
```

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

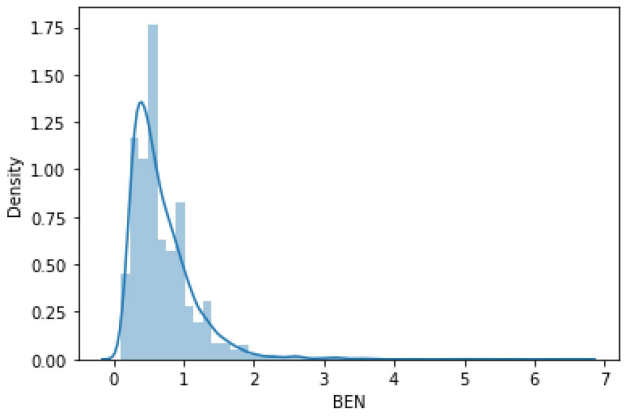
Out[18]: <seaborn.axisgrid.PairGrid at 0x1f74d26d550>



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

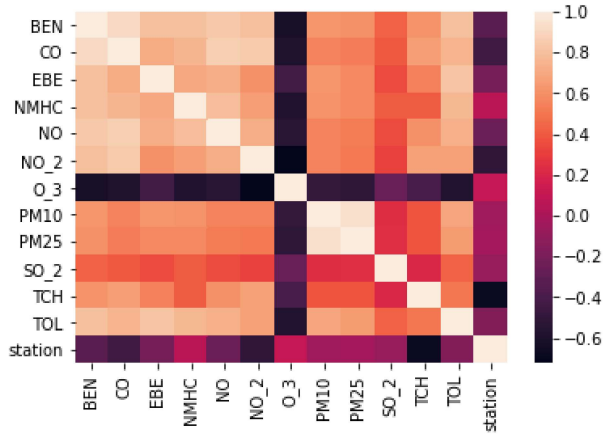
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is
a deprecated function and will be removed in a future version. Please adapt your code to use either `di
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function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

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



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

Out[20]: <AxesSubplot:>



In [21]:
```
1 x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
2 '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)
```

In [23]:
```
1 from sklearn.linear_model import LinearRegression
2 lr=LinearRegression()
3 lr.fit(x_train,y_train)
```

Out[23]: LinearRegression()

In [24]:
```
1 lr.intercept_
```

Out[24]: 28079045.756491568

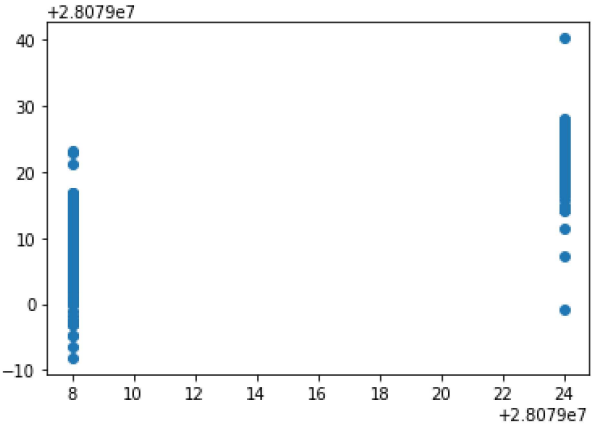
In [25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	-0.549441
CO	-22.024446
EBE	0.122078
NMHC	141.767134
NO	0.036296
NO_2	-0.149731
O_3	-0.077950
PM10	0.011811
PM25	0.147661
SO_2	-0.049968
TCH	-17.107525
TOL	-0.050521

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

Out[26]: <matplotlib.collections.PathCollection at 0x1f7570b8670>



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

Out[27]: 0.8158991759658195

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

Out[28]: 0.8069102549582842

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)

In [31]:
```
rr.score(x_test,y_test)
```

Out[31]: 0.702967782154945

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

Out[32]: 0.709408577546153

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

Out[33]:
```
Lasso(alpha=10)
```

In [34]:
```
la.score(x_test,y_test)
```

Out[34]:
```
0.4229829556738396
```

In [35]:
```
la.score(x_train,y_train)
```

Out[35]:
```
0.4113700257643299
```

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.        , -0.        , -0.        ,  0.        ,  0.03524078,
       -0.28709458, -0.1422162 ,  0.24799648, -0.0576301 ,  0.09137029,
       -0.13360456,  0.        ])
```

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

Out[38]:
```
28079029.658046696
```

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

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

Out[40]:
```
0.4901906715101415
```

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)))
```

```
4.731173390091293
31.405230207605218
5.604036956302592
```

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]:
```
(4562, 10)
```

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

Out[45]:
```
(4562,)
```

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([28079008, 28079024], dtype=int64)
```

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

Out[52]:
```
0.9888206926786497
```

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

Out[53]:
```
1.0
```

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

Out[54]:
```
array([[1.00000000e+00, 1.42669593e-19]])
```

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')
```

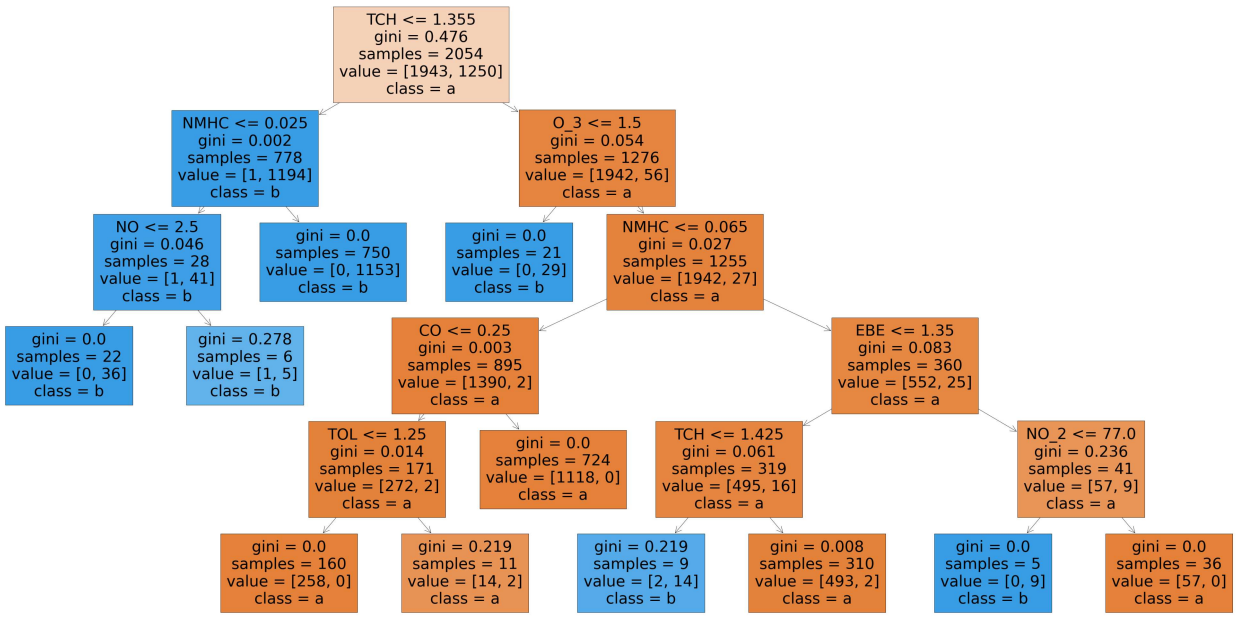
In [59]:
```
grid_search.best_score_
```

Out[59]:
```
0.9921700776675565
```

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(1539.3103448275863, 1993.2, 'TCH <= 1.355\ngini = 0.476\nsamples = 2054\nvalue = [1943, 1250]\ncl
ass = a'),
 Text(923.5862068965519, 1630.8000000000002, 'NMHC <= 0.025\ngini = 0.002\nsamples = 778\nvalue = [1, 1
194]\nclass = b'),
 Text(615.7241379310345, 1268.4, 'NO <= 2.5\ngini = 0.046\nsamples = 28\nvalue = [1, 41]\nclass = b'),
 Text(307.86206896551727, 906.0, 'gini = 0.0\nsamples = 22\nvalue = [0, 36]\nclass = b'),
 Text(923.5862068965519, 906.0, 'gini = 0.278\nsamples = 6\nvalue = [1, 5]\nclass = b'),
 Text(1231.448275862069, 1268.4, 'gini = 0.0\nsamples = 750\nvalue = [0, 1153]\nclass = b'),
 Text(2155.034482758621, 1630.8000000000002, 'O_3 <= 1.5\ngini = 0.054\nsamples = 1276\nvalue = [1942,
56]\nclass = a'),
 Text(1847.1724137931037, 1268.4, 'gini = 0.0\nsamples = 21\nvalue = [0, 29]\nclass = b'),
 Text(2462.896551724138, 1268.4, 'NMHC <= 0.065\ngini = 0.027\nsamples = 1255\nvalue = [1942, 27]\nclas
s = a'),
 Text(1693.2413793103449, 906.0, 'CO <= 0.25\ngini = 0.003\nsamples = 895\nvalue = [1390, 2]\nclass =
a'),
 Text(1385.3793103448277, 543.5999999999999, 'TOL <= 1.25\ngini = 0.014\nsamples = 171\nvalue = [272,
2]\nclass = a'),
 Text(1077.5172413793105, 181.19999999999982, 'gini = 0.0\nsamples = 160\nvalue = [258, 0]\nclass =
a'),
 Text(1693.2413793103449, 181.19999999999982, 'gini = 0.219\nsamples = 11\nvalue = [14, 2]\nclass =
a'),
 Text(2001.1034482758623, 543.5999999999999, 'gini = 0.0\nsamples = 724\nvalue = [1118, 0]\nclass =
a'),
 Text(3232.551724137931, 906.0, 'EBE <= 1.35\ngini = 0.083\nsamples = 360\nvalue = [552, 25]\nclass =
a'),
 Text(2616.8275862068967, 543.5999999999999, 'TCH <= 1.425\ngini = 0.061\nsamples = 319\nvalue = [495,
16]\nclass = a'),
 Text(2308.9655172413795, 181.19999999999982, 'gini = 0.219\nsamples = 9\nvalue = [2, 14]\nclass = b'),
 Text(2924.689655172414, 181.19999999999982, 'gini = 0.008\nsamples = 310\nvalue = [493, 2]\nclass =
a'),
 Text(3848.275862068966, 543.5999999999999, 'NO_2 <= 77.0\ngini = 0.236\nsamples = 41\nvalue = [57, 9]
\nclass = a'),
 Text(3540.4137931034484, 181.19999999999982, 'gini = 0.0\nsamples = 5\nvalue = [0, 9]\nclass = b'),
 Text(4156.137931034483, 181.19999999999982, 'gini = 0.0\nsamples = 36\nvalue = [57, 0]\nclass = a')]
```



Conclusion

Linear Regression=0.8069102549582842

Ridge Regression=0.709408577546153

Lasso Regression=0.4113700257643299

ElasticNet Regression=0.4901906715101415

Logistic Regression=0.9888206926786497

Random Forest=0.9921700776675565

Random Forest is suitable for this dataset