Vijay(P15) 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_2015.csv")
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	NaN	0.8	NaN	NaN	90.0	82.0	NaN	NaN	NaN	10.0	NaN	NaN	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	NaN	1.8	NaN	29.0	97.0	NaN	NaN	NaN	NaN	NaN	7.1	28079011
3	2015-10-01 01:00:00	NaN	0.6	NaN	NaN	30.0	103.0	2.0	NaN	NaN	NaN	NaN	NaN	28079016
4	2015-10-01 01:00:00	NaN	NaN	NaN	NaN	95.0	96.0	2.0	NaN	NaN	9.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210091	2015-08-01 00:00:00	NaN	0.2	NaN	NaN	11.0	33.0	53.0	NaN	NaN	NaN	NaN	NaN	28079056
210092	2015-08-01 00:00:00	NaN	0.2	NaN	NaN	1.0	5.0	NaN	26.0	NaN	10.0	NaN	NaN	28079057
210093	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	1.0	7.0	74.0	NaN	NaN	NaN	NaN	NaN	28079058
210094	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	3.0	7.0	65.0	NaN	NaN	NaN	NaN	NaN	28079059
210095	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	1.0	9.0	54.0	29.0	NaN	NaN	NaN	NaN	28079060

210096 rows × 14 columns

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: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16026 non-null  object 
 1   BEN      16026 non-null  float64
 2   CO       16026 non-null  float64
 3   EBE      16026 non-null  float64
 4   NMHC     16026 non-null  float64
 5   NO       16026 non-null  float64
 6   NO_2     16026 non-null  float64
 7   O_3      16026 non-null  float64
 8   PM10     16026 non-null  float64
 9   PM25     16026 non-null  float64
 10  SO_2     16026 non-null  float64
 11  TCH      16026 non-null  float64
 12  TOL      16026 non-null  float64
 13  station  16026 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ MB
```

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

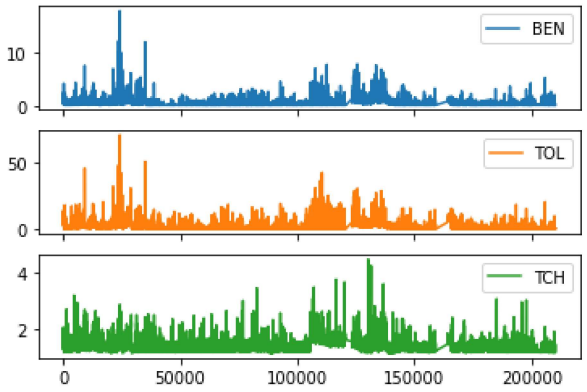
Out[6]:

	BEN	TOL	TCH
1	2.0	8.3	1.83
6	0.5	4.8	1.29
25	1.6	6.9	1.93
30	0.4	7.8	1.27
49	2.2	13.9	2.05
...	...	...	...
210030	0.1	0.2	1.18
210049	0.4	1.2	1.45
210054	0.1	0.2	1.18
210073	0.1	0.6	1.44
210078	0.1	0.4	1.18

16026 rows × 3 columns

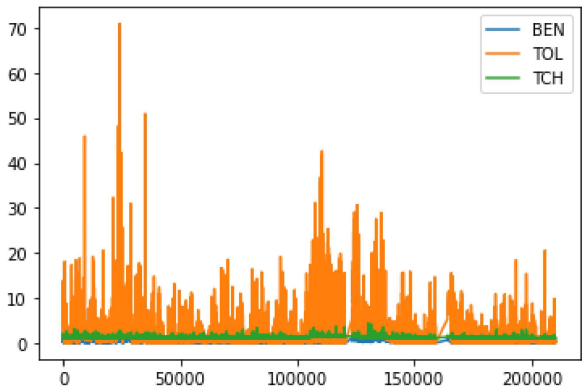
In [7]:
```
data.plot.line(subplots=True)
```

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



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

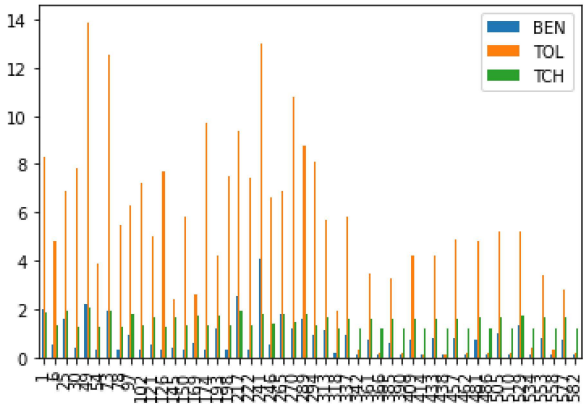
Out[8]:
```
<AxesSubplot:>
```



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

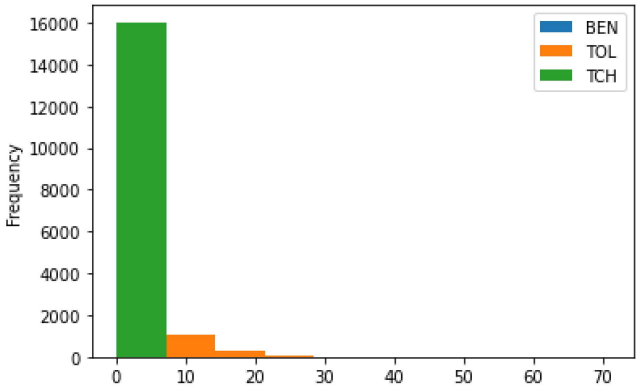
In [10]:
```
1 b.plot.bar()
```

Out[10]: <AxesSubplot:>



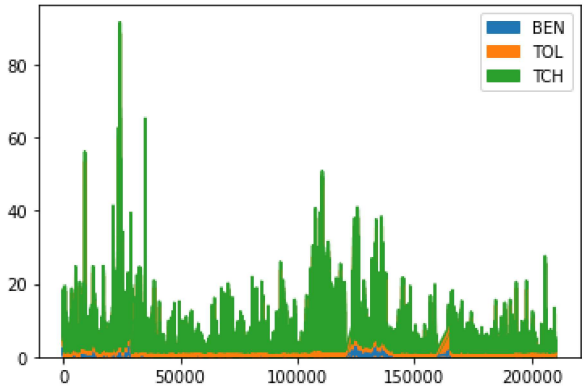
In [12]:
```
1 data.plot.hist()
```

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



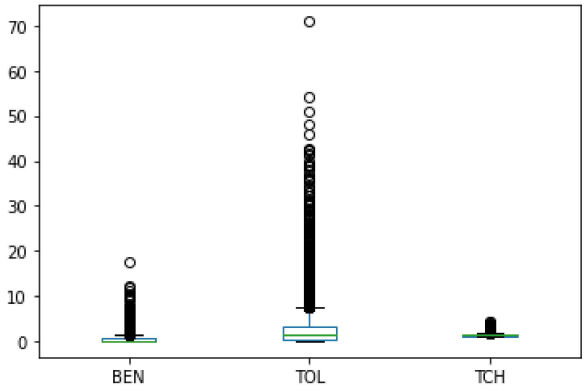
In [13]:
```
1 data.plot.area()
```

Out[13]: <AxesSubplot:>



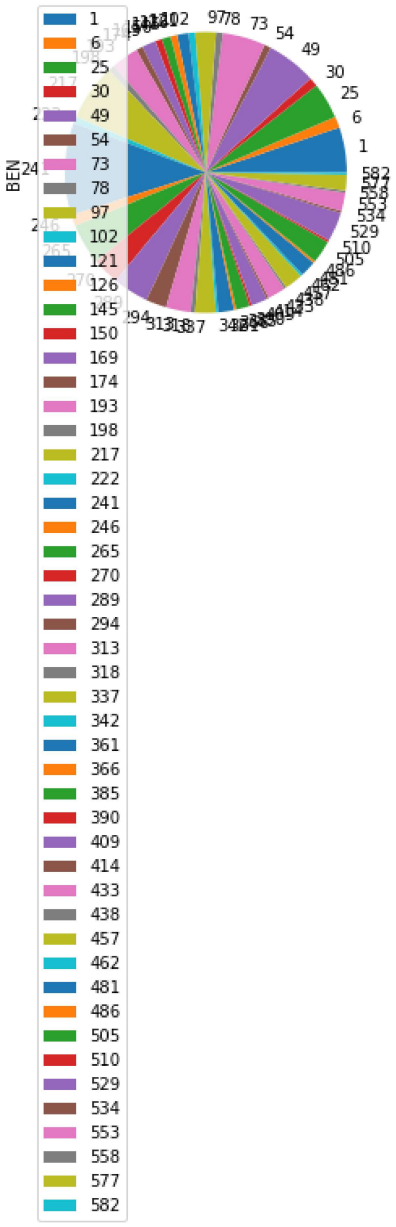
In [14]:
```
1 data.plot.box()
```

Out[14]: <AxesSubplot:>



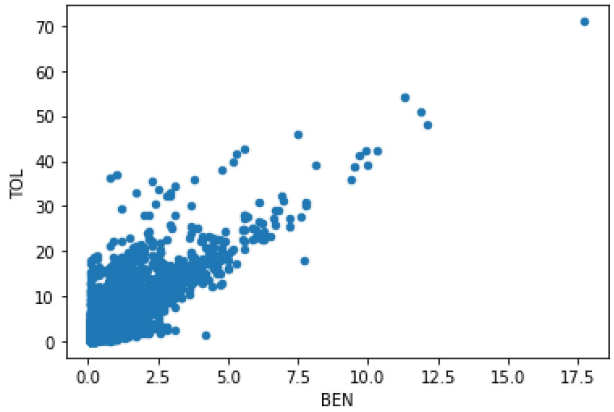
In [15]:
```
1 b.plot.pie(y='BEN' )
```

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



In [16]:
```
1 data.plot.scatter(x='BEN' ,y='TOL')
```

Out[16]: <AxesSubplot:xlabel='BEN', ylabel='TOL'>



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

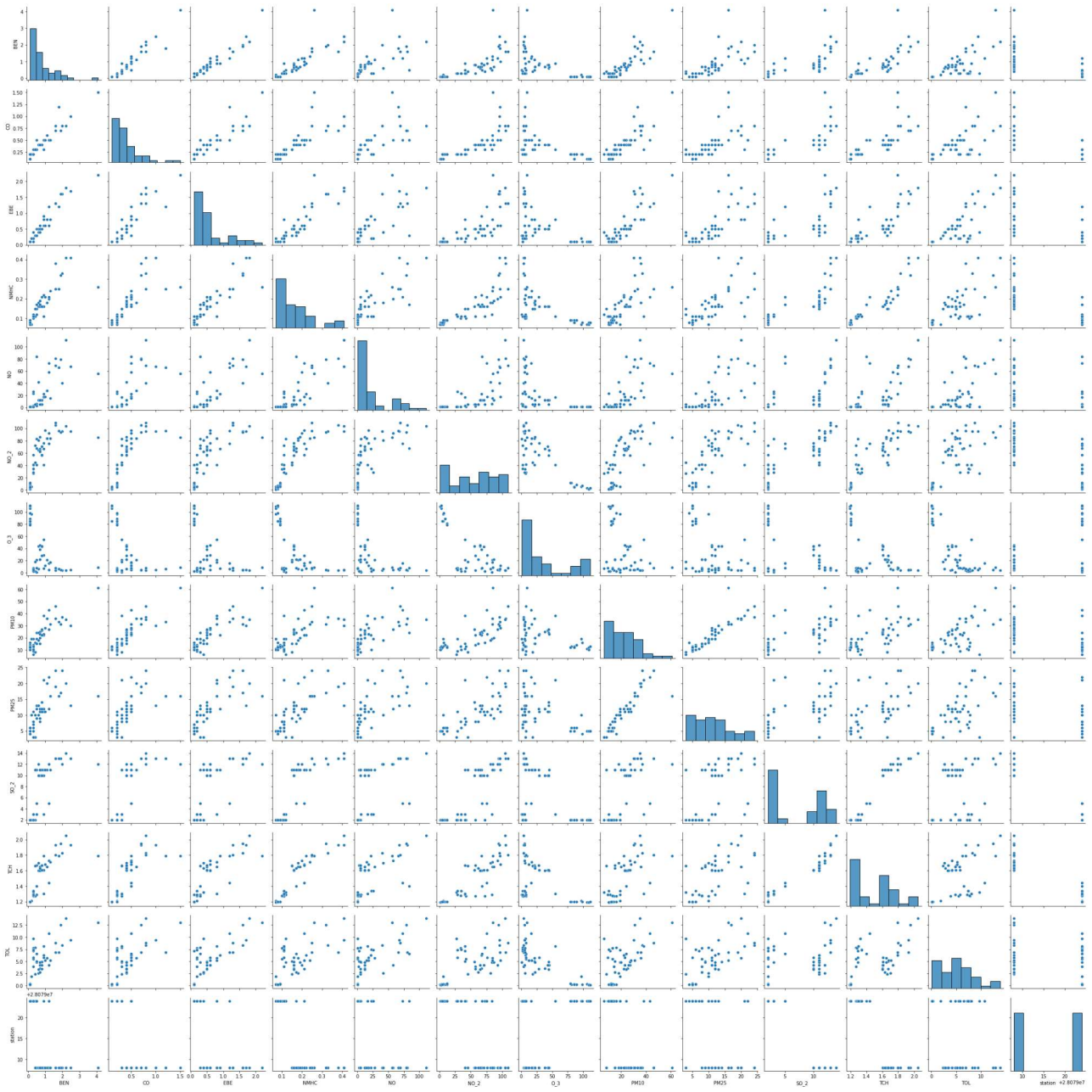
Out[17]:

	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	
count	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.
mean	0.504823	0.380594	0.394247	0.123099	23.842256	40.948771	48.089792	22.183764	11.
std	0.716896	0.260805	0.678592	0.092368	51.255660	33.236098	35.847298	15.993825	8.
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	1.000000	1.000000	0.
25%	0.100000	0.200000	0.100000	0.070000	1.000000	14.000000	15.000000	11.000000	6.
50%	0.200000	0.300000	0.100000	0.100000	6.000000	35.000000	46.000000	19.000000	10.
75%	0.700000	0.500000	0.400000	0.140000	24.000000	60.000000	73.000000	29.000000	16.
max	17.700001	4.500000	12.100000	1.090000	960.000000	369.000000	217.000000	196.000000	88.

In [18]:
```
1 df1=df[['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
2 'SO_2', 'TCH', 'TOL', 'station']]
```

In [19]:
```
1 sns.pairplot(df1[0:50])
```

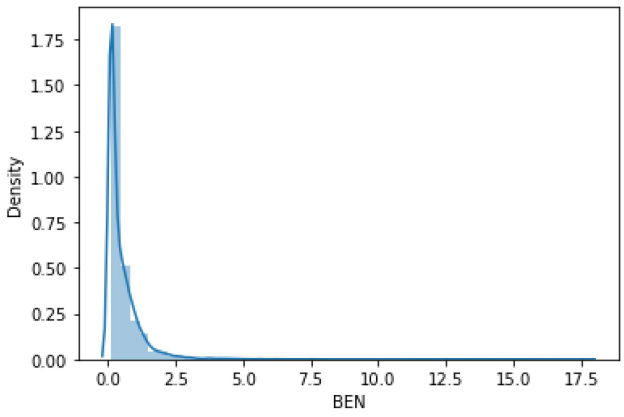
Out[19]: <seaborn.axisgrid.PairGrid at 0x1adc206fa00>



In [20]:
```
1 sns.distplot(df1['BEN'])
```

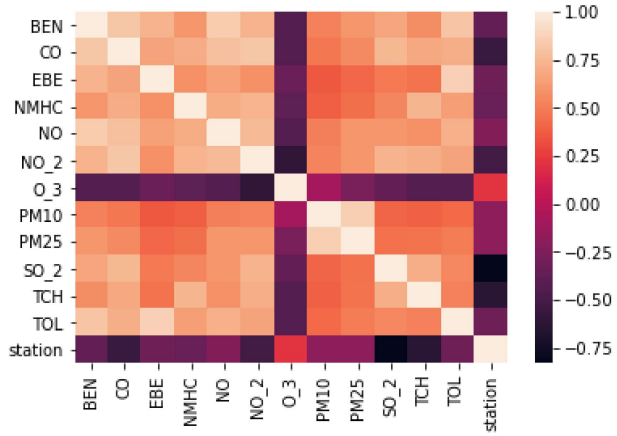
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is
a deprecated function and will be removed in a future version. Please adapt your code to use either `di
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function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

Out[20]: <AxesSubplot:xlabel='BEN', ylabel='Density'>



In [21]:
```
1 sns.heatmap(df1.corr())
```

Out[21]: <AxesSubplot:>



In [22]:
```
1 x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
2 'SO_2', 'TCH', 'TOL']]
3 y=df['station']
```

In [23]:
```
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 [24]:
```
1 from sklearn.linear_model import LinearRegression
2 lr=LinearRegression()
3 lr.fit(x_train,y_train)
```

Out[24]: LinearRegression()

In [25]:
```
1 lr.intercept_
```

Out[25]: 28079038.18771696

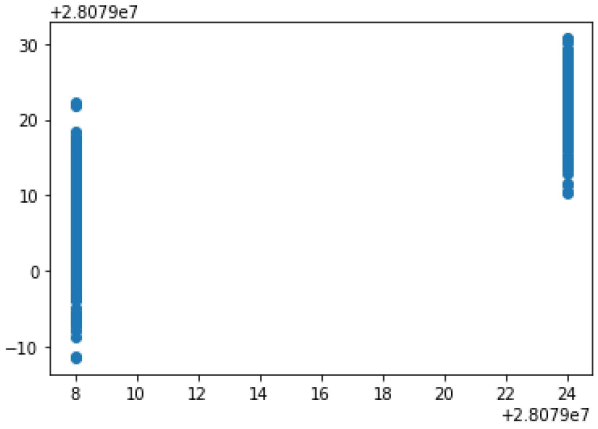
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	1.328184
CO	-9.525241
EBE	-0.680495
NMHC	13.550123
NO	0.081748
NO_2	-0.018620
O_3	-0.013259
PM10	0.006421
PM25	0.093514
SO_2	-1.120140
TCH	-9.621722
TOL	-0.125403

In [27]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[27]: <matplotlib.collections.PathCollection at 0x1adcf27a0d0>



In [28]:
```
lr.score(x_test,y_test)
```

Out[28]: 0.8776852904220687

In [29]:
```
lr.score(x_train,y_train)
```

Out[29]: 0.8691210440709395

In [30]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [31]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[31]: Ridge(alpha=10)

In [32]:
```
rr.score(x_test,y_test)
```

Out[32]: 0.8766762101663818

In [33]:
```
rr.score(x_train,y_train)
```

Out[33]: 0.8683060891347234

In [34]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[34]:
```
Lasso(alpha=10)
```

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

Out[35]:
```
0.7318409704608069
```

In [36]:
```
la.score(x_train,y_train)
```

Out[36]:
```
0.7284061635440928
```

In [37]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[37]:
```
ElasticNet()
```

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

Out[38]:
```
array([-0.        , -0.        , -0.        , -0.        ,  0.07630632,
       -0.05407965, -0.01126557,  0.0192436 ,  0.05271942, -1.31074793,
       -0.        , -0.09677337])
```

In [39]:
```
en.intercept_
```

Out[39]:
```
28079025.94073986
```

In [40]:
```
prediction=en.predict(x_test)
```

In [41]:
```
en.score(x_test,y_test)
```

Out[41]:
```
0.8255964162574249
```

In [42]:
```
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)))
```

```
2.5145796038979005
11.156404129066058
3.340120376433469
```

In [43]:
```
from sklearn.linear_model import LogisticRegression
```

In [44]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
'PM10', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

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

Out[45]:
```
(16026, 10)
```

In [46]:
```
target_vector.shape
```

Out[46]:
```
(16026,)
```

In [47]:
```
from sklearn.preprocessing import StandardScaler
```

In [48]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [49]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[49]:
```
LogisticRegression(max_iter=10000)
```

In [50]:
```
observation=[[1,2,3,4,5,6,7,8,9,10]]
```

In [51]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [52]:
```
logr.classes_
```

Out[52]:
```
array([28079008, 28079024], dtype=int64)
```

In [53]:
```
logr.score(fs,target_vector)
```

Out[53]:
```
0.9947585174092101
```

In [54]:
```
logr.predict_proba(observation)[0][0]
```

Out[54]:
```
1.0
```

In [55]:
```
logr.predict_proba(observation)
```

Out[55]:
```
array([[1.00000000e+00, 5.69793111e-39]])
```

In [56]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [57]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[57]:
```
RandomForestClassifier()
```

In [58]:
```
parameters={'max_depth':[1,2,3,4,5],
'min_samples_leaf':[5,10,15,20,25],
'n_estimators':[10,20,30,40,50]
}
```

In [59]:
```
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[59]:
```
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')
```

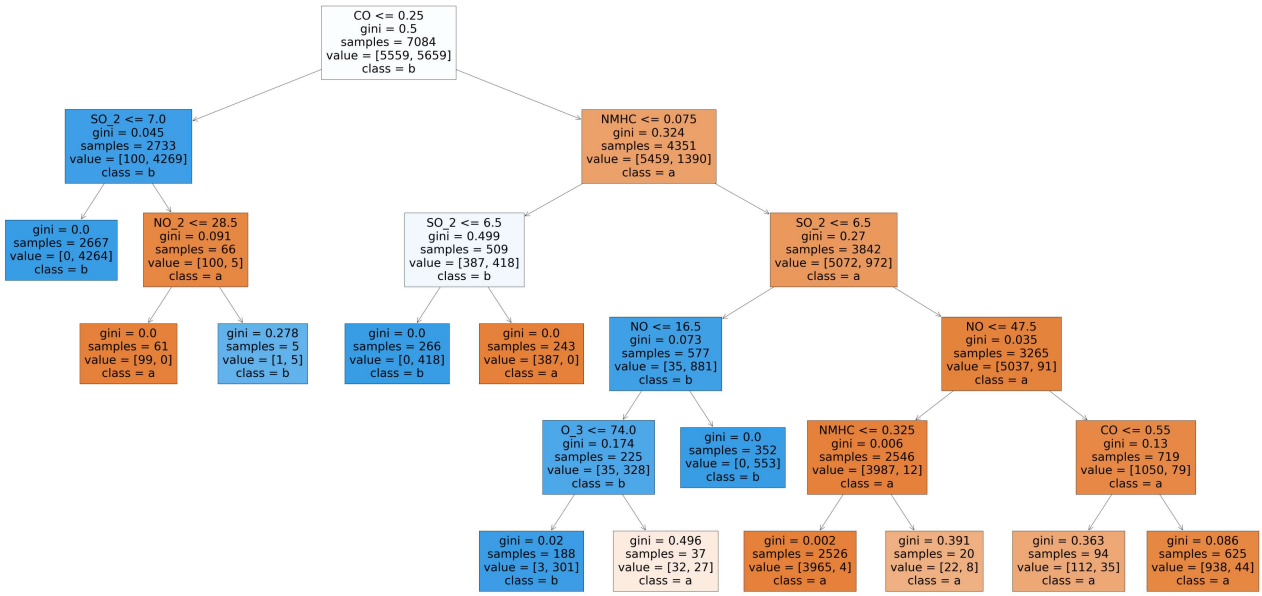
In [60]:
```
grid_search.best_score_
```

Out[60]:
```
0.994651453021929
```

In [61]:
```
rfc_best=grid_search.best_estimator_
```

In [62]:
```
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[62]:
```
[Text(1380.3157894736842, 1993.2, 'CO <= 0.25\ngini = 0.5\nsamples = 7084\nvalue = [5559, 5659]\nclass
= b'),
 Text(469.89473684210526, 1630.8000000000002, 'SO_2 <= 7.0\ngini = 0.045\nsamples = 2733\nvalue = [100,
4269]\nclass = b'),
 Text(234.94736842105263, 1268.4, 'gini = 0.0\nsamples = 2667\nvalue = [0, 4264]\nclass = b'),
 Text(704.8421052631579, 1268.4, 'NO_2 <= 28.5\ngini = 0.091\nsamples = 66\nvalue = [100, 5]\nclass =
a'),
 Text(469.89473684210526, 906.0, 'gini = 0.0\nsamples = 61\nvalue = [99, 0]\nclass = a'),
 Text(939.7894736842105, 906.0, 'gini = 0.278\nsamples = 5\nvalue = [1, 5]\nclass = b'),
 Text(2290.7368421052633, 1630.8000000000002, 'NMHC <= 0.075\ngini = 0.324\nsamples = 4351\nvalue = [54
59, 1390]\nclass = a'),
 Text(1644.6315789473683, 1268.4, 'SO_2 <= 6.5\ngini = 0.499\nsamples = 509\nvalue = [387, 418]\nclass
= b'),
 Text(1409.6842105263158, 906.0, 'gini = 0.0\nsamples = 266\nvalue = [0, 418]\nclass = b'),
 Text(1879.578947368421, 906.0, 'gini = 0.0\nsamples = 243\nvalue = [387, 0]\nclass = a'),
 Text(2936.842105263158, 1268.4, 'SO_2 <= 6.5\ngini = 0.27\nsamples = 3842\nvalue = [5072, 972]\nclass
= a'),
 Text(2349.4736842105262, 906.0, 'NO <= 16.5\ngini = 0.073\nsamples = 577\nvalue = [35, 881]\nclass =
b'),
 Text(2114.5263157894738, 543.5999999999999, 'O_3 <= 74.0\ngini = 0.174\nsamples = 225\nvalue = [35, 32
8]\nclass = b'),
 Text(1879.578947368421, 181.19999999999982, 'gini = 0.02\nsamples = 188\nvalue = [3, 301]\nclass =
b'),
 Text(2349.4736842105262, 181.19999999999982, 'gini = 0.496\nsamples = 37\nvalue = [32, 27]\nclass =
a'),
 Text(2584.4210526315787, 543.5999999999999, 'gini = 0.0\nsamples = 352\nvalue = [0, 553]\nclass = b'),
 Text(3524.2105263157896, 906.0, 'NO <= 47.5\ngini = 0.035\nsamples = 3265\nvalue = [5037, 91]\nclass =
a'),
 Text(3054.315789473684, 543.5999999999999, 'NMHC <= 0.325\ngini = 0.006\nsamples = 2546\nvalue = [398
7, 12]\nclass = a'),
 Text(2819.3684210526317, 181.19999999999982, 'gini = 0.002\nsamples = 2526\nvalue = [3965, 4]\nclass =
a'),
 Text(3289.2631578947367, 181.19999999999982, 'gini = 0.391\nsamples = 20\nvalue = [22, 8]\nclass =
a'),
 Text(3994.1052631578946, 543.5999999999999, 'CO <= 0.55\ngini = 0.13\nsamples = 719\nvalue = [1050, 7
9]\nclass = a'),
 Text(3759.157894736842, 181.19999999999982, 'gini = 0.363\nsamples = 94\nvalue = [112, 35]\nclass =
a'),
 Text(4229.0526315789475, 181.19999999999982, 'gini = 0.086\nsamples = 625\nvalue = [938, 44]\nclass =
a')]
```



Conclusion

Linear Regression=0.8691210440709395

Ridge Regression=0.8683060891347234

Lasso Regression=0.7284061635440928

ElasticNet Regression=0.8255964162574249

Logistic Regression=0.9947585174092101

Random Forest=0.994651453021929

Logistic Regression is suitable for this dataset

In [ ]:
```
1
```