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(r"C:\Users\user\Downloads\C10_air\csvs_per_year\csvs(Dataset)\ma
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.209
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.389
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.240
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.839
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.779
...	...	...	...	...	...	...	...	...	...	...	
243979	2003-10-01 00:00:00	0.20	0.16	2.01	3.17	0.02	31.799999	32.299999	1.68	34.049999	7.380
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.400
243981	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.830
243982	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.570
243983	2003-10-01 00:00:00	1.00	0.29	2.15	6.41	0.07	37.150002	56.849998	2.28	21.480000	12.350

243984 rows × 16 columns

Data Cleaning and Data Preprocessing

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

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

Out[4]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

In [7]:
```
data=df[['EBE', 'MXY', 'PXY']]
data
```

Out[7]:

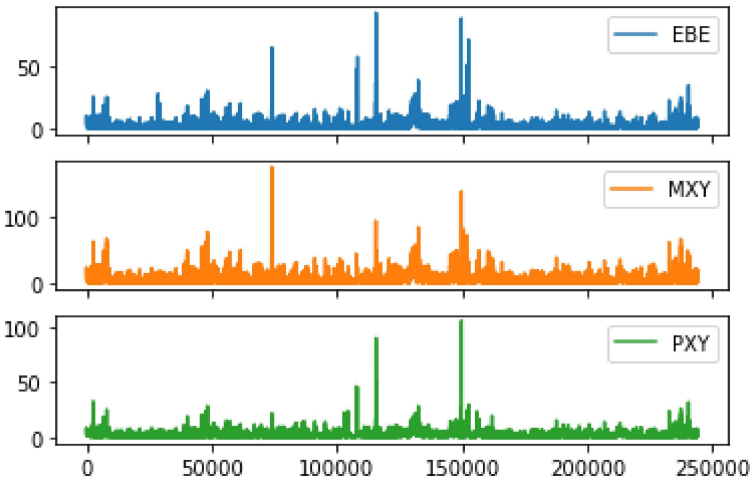
	EBE	MXY	PXY
5	9.83	21.49	7.94
23	3.43	7.08	2.62
27	5.75	10.88	4.24
33	10.63	24.73	8.93
51	3.20	7.08	2.70
...	...	...	...
243955	3.07	9.38	3.48
243957	3.88	10.86	3.89
243961	4.53	10.88	4.13
243979	2.01	3.17	1.20
243983	2.15	6.41	2.43

33010 rows × 3 columns

Line chart

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

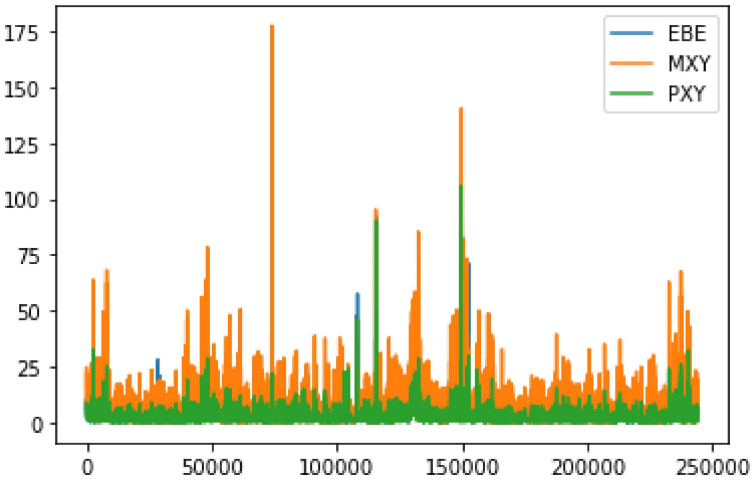
Out[8]:
```
array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [9]:
```
data.plot.line()
```

Out[9]: <AxesSubplot:>

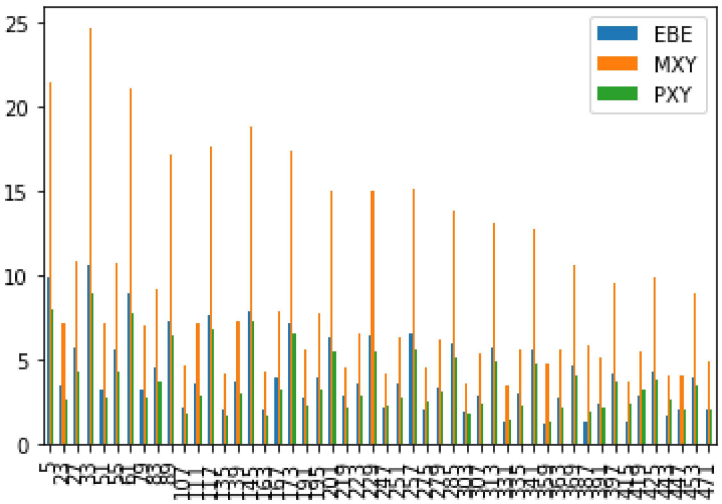


Bar chart

In [10]:
```
b=data[0:50]
```

In [11]:
```
b.plot.bar()
```

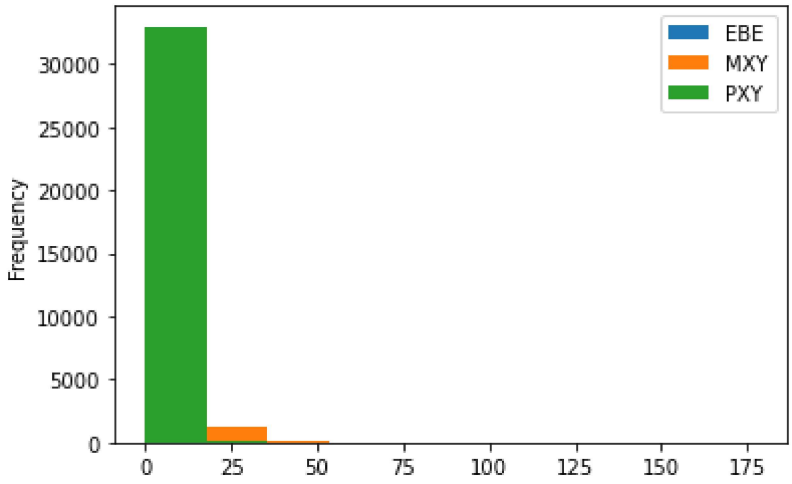
Out[11]: <AxesSubplot:>



Histogram

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

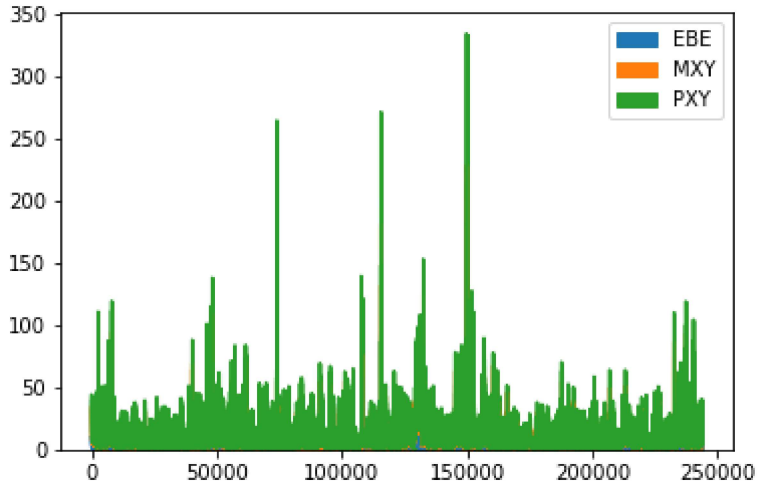
Out[12]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

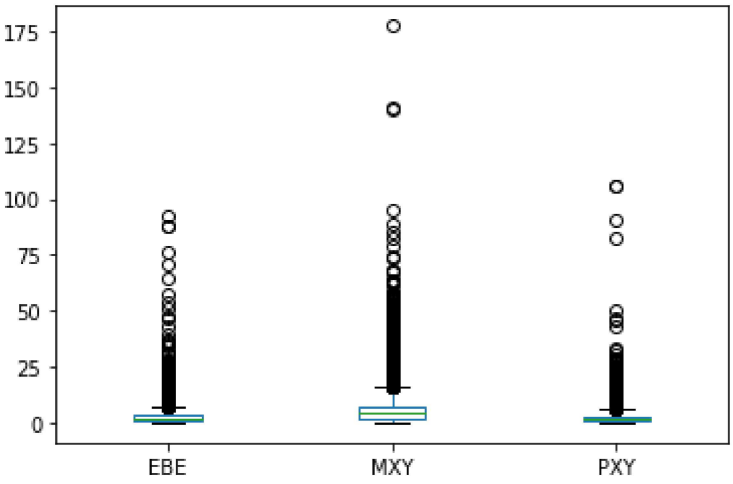
Out[13]: <AxesSubplot:>



Box chart

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

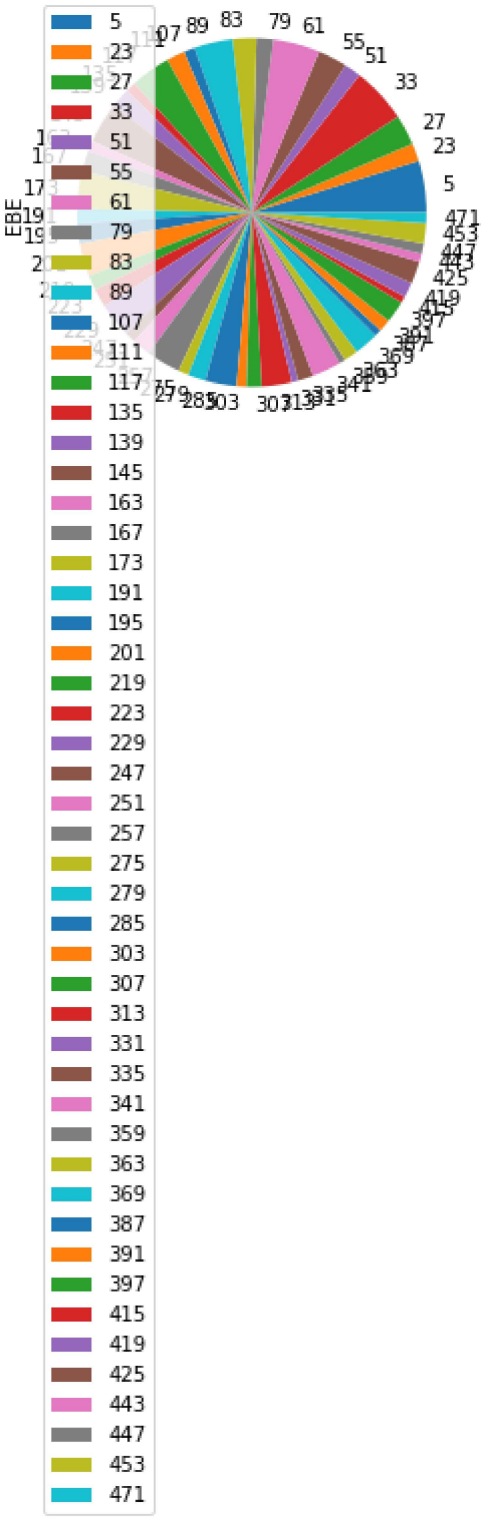
Out[14]: <AxesSubplot:>



Pie chart

In [16]:
```
b.plot.pie(y='EBE' )
```

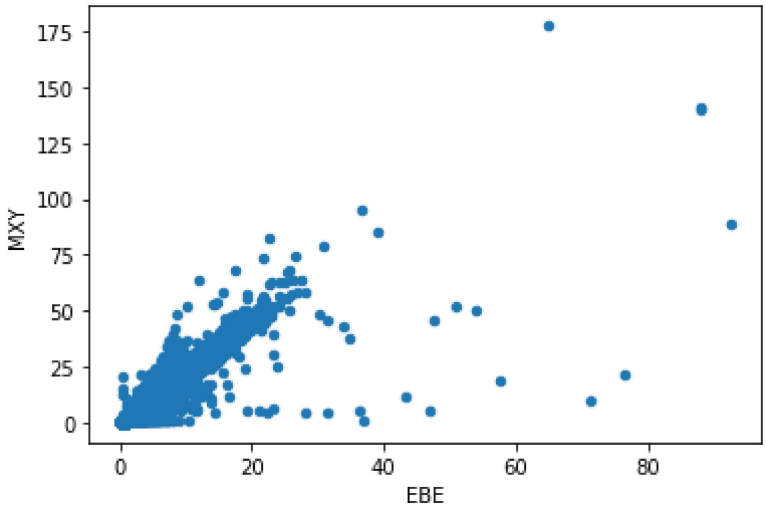
Out[16]:
```
<AxesSubplot:ylabel='EBE'>
```



Scatter chart

In [17]:
```
data.plot.scatter(x='EBE' ,y='MXY')
```

Out[17]:
```
<AxesSubplot:xlabel='EBE', ylabel='MXY'>
```



In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

In [19]:
```
df.describe()
```

Out[19]:

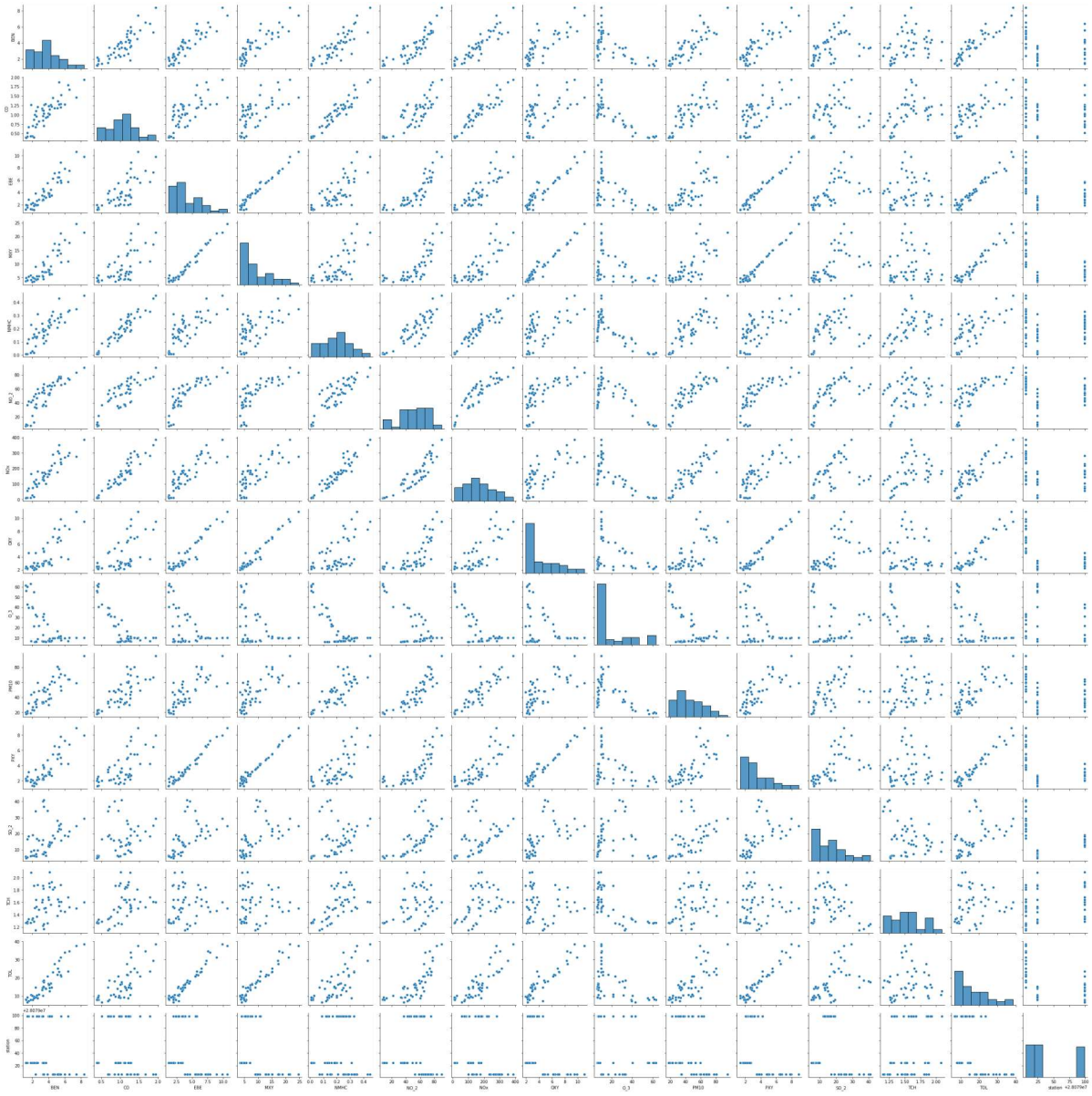
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	330
mean	2.192633	0.759868	2.639726	5.838414	0.137177	57.328049	1
std	2.064160	0.545999	2.825194	6.267296	0.127863	31.811082	1
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.900000	0.430000	1.010000	1.880000	0.060000	34.529999	
50%	1.610000	0.620000	1.890000	4.070000	0.110000	55.105000	
75%	2.810000	0.930000	3.300000	7.530000	0.170000	76.160004	1
max	66.389999	7.920000	92.589996	177.600006	2.180000	342.700012	12

In [20]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [21]:
```
sns.pairplot(df1[0:50])
```

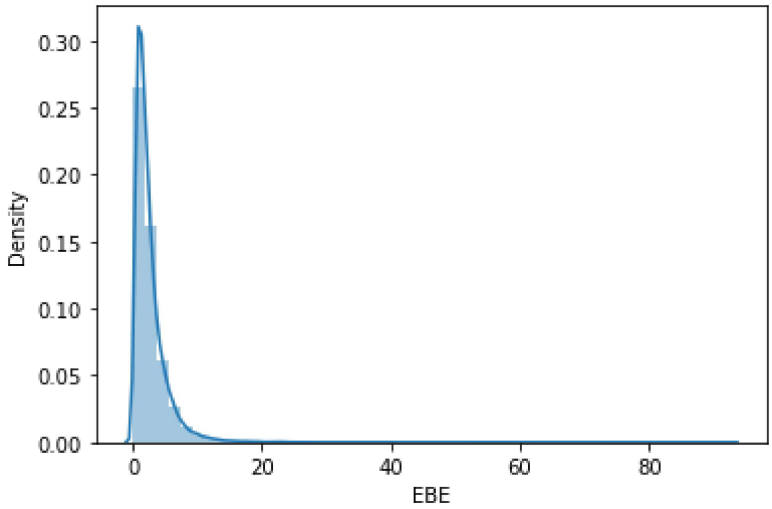
Out[21]: <seaborn.axisgrid.PairGrid at 0x18dd3ef6b50>



In [22]:
```
sns.distplot(df1['EBE'])
```

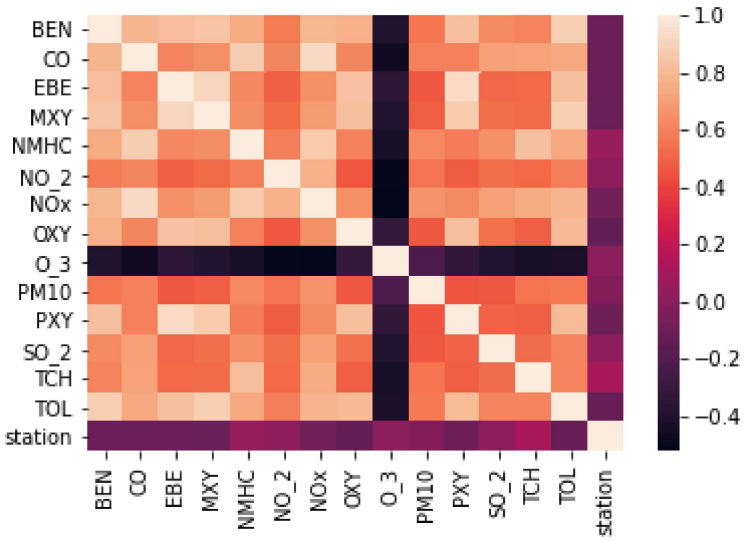
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function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

Out[22]:
```
<AxesSubplot:xlabel='EBE', ylabel='Density'>
```



In [23]:
```
sns.heatmap(df1.corr())
```

Out[23]:
```
<AxesSubplot:>
```



TO TRAIN THE MODEL AND MODEL BULDING

In [24]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [25]:
```
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 [26]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[26]:
```
LinearRegression()
```

In [27]:
```
lr.intercept_
```

Out[27]:
```
28079000.88287165
```

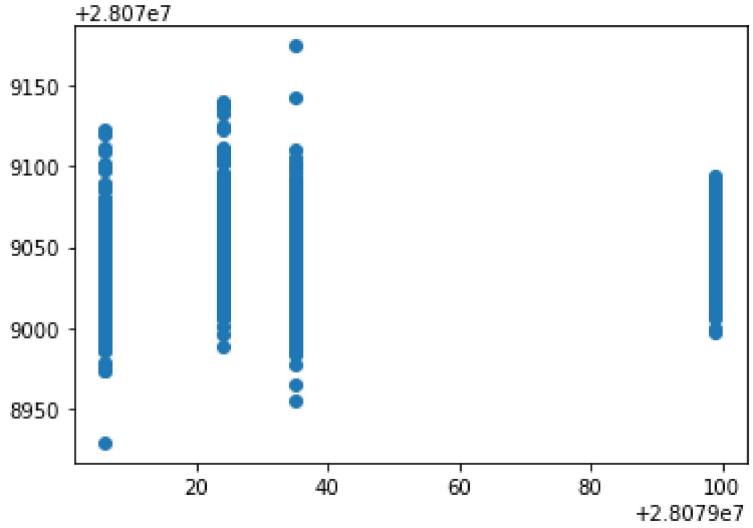
In [28]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[28]:

	Co-efficient
BEN	1.692702
CO	-38.977600
EBE	-1.867924
MXY	0.174105
NMHC	152.722698
NO_2	0.163994
NOx	-0.071702
OXY	-1.299032
O_3	-0.014141
PM10	-0.050933
PXY	2.076299
SO_2	0.865766
TCH	35.557182
TOL	-0.922173

In [29]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[29]: <matplotlib.collections.PathCollection at 0x18de1c99c70>



ACCURACY

In [30]:
```
lr.score(x_test,y_test)
```

Out[30]: 0.1724179923348299

In [31]:
```
lr.score(x_train,y_train)
```

Out[31]: 0.1775053138791619

Ridge and Lasso

In [32]:
```
from sklearn.linear_model import Ridge,Lasso
```

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

Out[33]: Ridge(alpha=10)

Accuracy(Ridge)

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

Out[34]: 0.17104202048880912

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

Out[35]:
```
0.17647021880331226
```

In [36]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[36]:
```
Lasso(alpha=10)
```

In [37]:
```
la.score(x_train,y_train)
```

Out[37]:
```
0.03743296904759286
```

Accuracy(Lasso)

In [38]:
```
la.score(x_test,y_test)
```

Out[38]:
```
0.03248687905869285
```

Accuracy(Elastic Net)

In [40]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[40]:
```
ElasticNet()
```

In [41]:
```
en.coef_
```

Out[41]:
```
array([ 0.        , -0.22243934,  0.        , -0.01685342,  0.14109596,
        0.15774441, -0.07187445, -1.12341392, -0.04543365,  0.08531711,
        0.33830415,  0.7394519 ,  1.57593375, -0.47188519])
```

In [42]:
```
en.intercept_
```

Out[42]:
```
28079037.19291707
```

In [43]:
```
prediction=en.predict(x_test)
```

In [44]:
```
en.score(x_test,y_test)
```

Out[44]:
```
0.04581467200955813
```

Evaluation Metrics

In [45]:
```
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)))
```

```
29.119161570718372
1184.5346313544042
34.41706889545367
```

Logistic Regression

In [46]:
```
from sklearn.linear_model import LogisticRegression
```

In [47]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [48]:
```
feature_matrix.shape
```

Out[48]:
```
(33010, 14)
```

In [49]:
```
target_vector.shape
```

Out[49]:
```
(33010,)
```

In [50]:
```
from sklearn.preprocessing import StandardScaler
```

In [51]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [52]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[52]:
```
LogisticRegression(max_iter=10000)
```

In [53]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [54]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079035]
```

In [55]:
```
logr.classes_
```

Out[55]:
```
array([28079006, 28079024, 28079035, 28079099], dtype=int64)
```

In [56]:
```
logr.score(fs,target_vector)
```

Out[56]:
```
0.7584974250227204
```

In [57]:
```
logr.predict_proba(observation)[0][0]
```

Out[57]:
```
2.3306153265290618e-23
```

In [58]:
```
logr.predict_proba(observation)
```

Out[58]:
```
array([[2.33061533e-23, 1.44436075e-55, 1.00000000e+00, 6.68457491e-16]])
```

Random Forest

In [59]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [60]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[60]:
```
RandomForestClassifier()
```

In [61]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [62]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="ac
grid_search.fit(x_train,y_train)
```

Out[62]:
```
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 [63]:
```
grid_search.best_score_
```

Out[63]:
```
0.7254074069545836
```

In [64]:
```
rfc_best=grid_search.best_estimator_
```

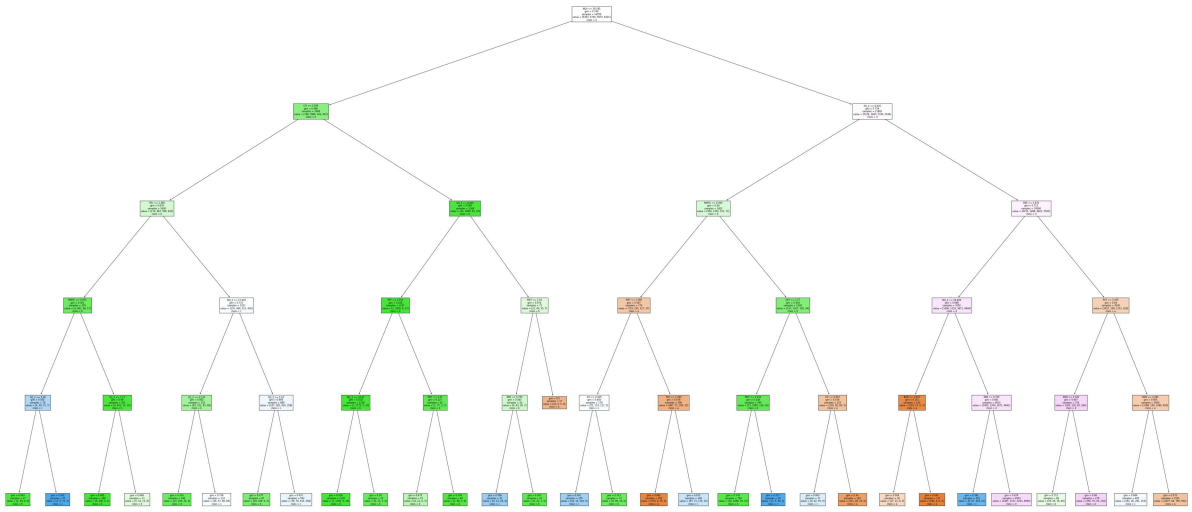
In [65]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['a','b'
```

Out[65]:
[Text(2222.7000000000003, 1993.2, 'NOx <= 38.185\ngini = 0.749\nsamples = 14550\nvalue = [5307, 5726, 5973, 6101]\nclass = d'),
 Text(1171.8000000000002, 1630.8000000000002, 'CO <= 0.385\ngini = 0.488\nsamples = 2668\nvalue = [198, 2886, 639, 467]\nclass = b'),
 Text(595.2, 1268.4, 'TOL <= 1.385\ngini = 0.679\nsamples = 1400\nvalue = [174, 987, 596, 443]\nclass = b'),
 Text(297.6, 906.0, 'NMHC <= 0.035\ngini = 0.301\nsamples = 379\nvalue = [4, 491, 84, 17]\nclass = b'),
 Text(148.8, 543.5999999999999, 'SO_2 <= 5.48\ngini = 0.545\nsamples = 78\nvalue = [4, 39, 72, 7]\nclass = c'),
 Text(74.4, 181.19999999999982, 'gini = 0.093\nsamples = 27\nvalue = [2, 39, 0, 0]\nclass = b'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.202\nsamples = 51\nvalue = [2, 0, 72, 7]\nclass = c'),
 Text(446.40000000000003, 543.5999999999999, 'SO_2 <= 7.27\ngini = 0.09\nsamples = 301\nvalue = [0, 452, 12, 10]\nclass = b'),
 Text(372.0, 181.19999999999982, 'gini = 0.005\nsamples = 280\nvalue = [0, 436, 0, 1]\nclass = b'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.649\nsamples = 21\nvalue = [0, 16, 12, 9]\nclass = b'),
 Text(892.8000000000001, 906.0, 'NO_2 <= 17.925\ngini = 0.721\nsamples = 1021\nvalue = [170, 496, 512, 426]\nclass = c'),
 Text(744.0, 543.5999999999999, 'SO_2 <= 6.335\ngini = 0.602\nsamples = 332\nvalue = [63, 311, 93, 68]\nclass = b'),
 Text(669.6, 181.19999999999982, 'gini = 0.314\nsamples = 199\nvalue = [27, 254, 25, 4]\nclass = b'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.738\nsamples = 133\nvalue = [36, 57, 68, 64]\nclass = c'),
 Text(1041.6000000000001, 543.5999999999999, 'SO_2 <= 5.19\ngini = 0.694\nsamples = 689\nvalue = [107, 185, 419, 358]\nclass = c'),
 Text(967.2, 181.19999999999982, 'gini = 0.377\nsamples = 97\nvalue = [29, 109, 5, 0]\nclass = b'),
 Text(1116.0, 181.19999999999982, 'gini = 0.637\nsamples = 592\nvalue = [78, 76, 414, 358]\nclass = c'),
 Text(1748.4, 1268.4, 'SO_2 <= 8.595\ngini = 0.089\nsamples = 1268\nvalue = [24, 1899, 43, 24]\nclass = b'),
 Text(1488.0, 906.0, 'PXY <= 1.015\ngini = 0.028\nsamples = 1197\nvalue = [2, 1850, 8, 17]\nclass = b'),
 Text(1339.2, 543.5999999999999, 'NO_2 <= 34.87\ngini = 0.018\nsamples = 1136\nvalue = [1, 1773, 5, 10]\nclass = b'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.016\nsamples = 1126\nvalue = [1, 1762, 3, 10]\nclass = b'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.26\nsamples = 10\nvalue = [0, 11, 2, 0]\nclass = b'),
 Text(1636.8000000000002, 543.5999999999999, 'MXY <= 2.51\ngini = 0.227\nsamples = 61\nvalue = [1, 77, 3, 7]\nclass = b'),
 Text(1562.4, 181.19999999999982, 'gini = 0.475\nsamples = 15\nvalue = [0, 11, 0, 7]\nclass = b'),
 Text(1711.2, 181.19999999999982, 'gini = 0.109\nsamples = 46\nvalue = [1, 66, 3, 0]\nclass = b'),
 Text(2008.8000000000002, 906.0, 'MXY <= 2.34\ngini = 0.674\nsamples = 71\nvalue = [22, 49, 35, 7]\nclass = b'),
 Text(1934.4, 543.5999999999999, 'EBE <= 0.755\ngini = 0.563\nsamples = 54\nvalue = [0, 43, 30, 7]\nclass = b'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.506\nsamples = 32\nvalue = [0, 12, 29, 4]\nclass = c'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.207\nsamples = 22\nva

```
lue = [0, 31, 1, 3]\nclass = b'),
 Text(2083.2000000000003, 543.5999999999999, 'gini = 0.5\nsamples = 17\nvalue
= [22, 6, 5, 0]\nclass = a'),
 Text(3273.6000000000004, 1630.8000000000002, 'SO_2 <= 6.625\ngini = 0.736\ns
amples = 11882\nvalue = [5109, 2840, 5334, 5634]\nclass = d'),
 Text(2678.4, 1268.4, 'NMHC <= 0.065\ngini = 0.64\nsamples = 1802\nvalue = [9
56, 1352, 512, 71]\nclass = b'),
 Text(2380.8, 906.0, 'MXY <= 2.085\ngini = 0.587\nsamples = 774\nvalue = [71
9, 185, 317, 37]\nclass = a'),
 Text(2232.0, 543.5999999999999, 'CO <= 0.425\ngini = 0.645\nsamples = 178\nv
alue = [52, 114, 117, 5]\nclass = c'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.541\nsamples = 105\nv
alue = [44, 15, 103, 5]\nclass = c'),
 Text(2306.4, 181.19999999999982, 'gini = 0.313\nsamples = 73\nvalue = [8, 9
9, 14, 0]\nclass = b'),
 Text(2529.6000000000004, 543.5999999999999, 'TCH <= 1.265\ngini = 0.478\nsam
ples = 596\nvalue = [667, 71, 200, 32]\nclass = a'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.081\nsamples = 356\nv
alue = [570, 0, 25, 0]\nclass = a'),
 Text(2604.0, 181.19999999999982, 'gini = 0.672\nsamples = 240\nvalue = [97,
71, 175, 32]\nclass = c'),
 Text(2976.0, 906.0, 'OXY <= 2.27\ngini = 0.454\nsamples = 1028\nvalue = [23
7, 1167, 195, 34]\nclass = b'),
 Text(2827.2000000000003, 543.5999999999999, 'MXY <= 5.225\ngini = 0.244\nsam
ples = 795\nvalue = [17, 1085, 126, 29]\nclass = b'),
 Text(2752.8, 181.19999999999982, 'gini = 0.176\nsamples = 756\nvalue = [16,
1080, 70, 27]\nclass = b'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.227\nsamples = 39\nva
lue = [1, 5, 56, 2]\nclass = c'),
 Text(3124.8, 543.5999999999999, 'CO <= 0.615\ngini = 0.576\nsamples = 233\nv
alue = [220, 82, 69, 5]\nclass = a'),
 Text(3050.4, 181.19999999999982, 'gini = 0.603\nsamples = 71\nvalue = [9, 4
2, 55, 5]\nclass = c'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.34\nsamples = 162\nva
lue = [211, 40, 14, 0]\nclass = a'),
 Text(3868.8, 1268.4, 'EBE <= 3.635\ngini = 0.713\nsamples = 10080\nvalue =
[4153, 1488, 4822, 5563]\nclass = d'),
 Text(3571.2000000000003, 906.0, 'NO_2 <= 25.495\ngini = 0.688\nsamples = 705
0\nvalue = [1696, 1219, 3671, 4644]\nclass = d'),
 Text(3422.4, 543.5999999999999, 'BEN <= 1.015\ngini = 0.151\nsamples = 135\n
value = [203, 15, 0, 3]\nclass = a'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.564\nsamples = 18\nva
lue = [17, 11, 0, 3]\nclass = a'),
 Text(3496.8, 181.19999999999982, 'gini = 0.041\nsamples = 117\nvalue = [186,
4, 0, 0]\nclass = a'),
 Text(3720.0000000000005, 543.5999999999999, 'EBE <= 0.725\ngini = 0.681\nsam
ples = 6915\nvalue = [1493, 1204, 3671, 4641]\nclass = d'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.381\nsamples = 352\nv
alue = [6, 67, 418, 51]\nclass = c'),
 Text(3794.4, 181.19999999999982, 'gini = 0.679\nsamples = 6563\nvalue = [148
7, 1137, 3253, 4590]\nclass = d'),
 Text(4166.400000000001, 906.0, 'PXY <= 3.435\ngini = 0.64\nsamples = 3030\nv
alue = [2457, 269, 1151, 919]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'BEN <= 2.345\ngini = 0.697\nsam
ples = 374\nvalue = [169, 129, 65, 264]\nclass = d'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.711\nsamples = 98\nva
lue = [19, 59, 30, 46]\nclass = b'),
```

```
 Text(4092.0000000000005, 181.19999999999982, 'gini = 0.66\nsamples = 276\nva
lue = [150, 70, 35, 218]\nclass = d'),
 Text(4315.200000000001, 543.5999999999999, 'BEN <= 3.285\ngini = 0.605\nsamp
les = 2656\nvalue = [2288, 140, 1086, 655]\nclass = a'),
 Text(4240.8, 181.19999999999982, 'gini = 0.684\nsamples = 495\nvalue = [261,
46, 291, 153]\nclass = c'),
 Text(4389.6, 181.19999999999982, 'gini = 0.572\nsamples = 2161\nvalue = [202
7, 94, 795, 502]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.1775053138791619

Ridge Regression:0.03743296904759286

Lasso Regression:0.03248687905869285

ElasticNet Regression:0.04581467200955813

Logistic Regression:0.7584974250227204

Random Forest:0.0.7254074069545836

Logistic Regression is suitable for this dataset