

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("2008.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXV	NMHC	NO_2	NOx	OXY	O_3	
0	2008-06-01 01:00:00	NaN	0.47	NaN	NaN	NaN	83.089996	120.699997	NaN	16.990000	16
1	2008-06-01 01:00:00	NaN	0.59	NaN	NaN	NaN	94.820000	130.399994	NaN	17.469999	16
2	2008-06-01 01:00:00	NaN	0.55	NaN	NaN	NaN	75.919998	104.599998	NaN	13.470000	20
3	2008-06-01 01:00:00	NaN	0.36	NaN	NaN	NaN	61.029999	66.559998	NaN	23.110001	10
4	2008-06-01 01:00:00	1.68	0.80	1.70	3.01	0.30	105.199997	214.899994	1.61	12.120000	37
...
226387	2008-11-01 00:00:00	0.48	0.30	0.57	1.00	0.31	13.050000	14.160000	0.91	57.400002	5
226388	2008-11-01 00:00:00	NaN	0.30	NaN	NaN	NaN	41.880001	48.500000	NaN	35.830002	15
226389	2008-11-01 00:00:00	0.25	NaN	0.56	NaN	0.11	83.610001	102.199997	NaN	14.130000	17
226390	2008-11-01 00:00:00	0.54	NaN	2.70	NaN	0.18	70.639999	81.860001	NaN	NaN	11
226391	2008-11-01 00:00:00	0.75	0.36	1.20	2.75	0.16	58.240002	74.239998	1.64	31.910000	12

226392 rows × 17 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', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:

df.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25631 entries, 4 to 226391
Data columns (total 17 columns):
#   Column      Non-Null Count  Dtype
---  -
0   date        25631 non-null  object
1   BEN         25631 non-null  float64
2   CO          25631 non-null  float64
3   EBE         25631 non-null  float64
4   MXY         25631 non-null  float64
5   NMHC        25631 non-null  float64
6   NO_2        25631 non-null  float64
7   NOx         25631 non-null  float64
8   OXY         25631 non-null  float64
9   O_3         25631 non-null  float64
10  PM10        25631 non-null  float64
11  PM25        25631 non-null  float64
12  PXY         25631 non-null  float64
13  SO_2        25631 non-null  float64
14  TCH         25631 non-null  float64
15  TOL         25631 non-null  float64
16  station     25631 non-null  int64
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [6]:

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

Out[6]:

	CO	station
4	0.80	28079006
21	0.37	28079024
25	0.39	28079099
30	0.51	28079006
47	0.39	28079024
...
226362	0.35	28079024
226366	0.46	28079099
226371	0.53	28079006
226387	0.30	28079024
226391	0.36	28079099

25631 rows × 2 columns

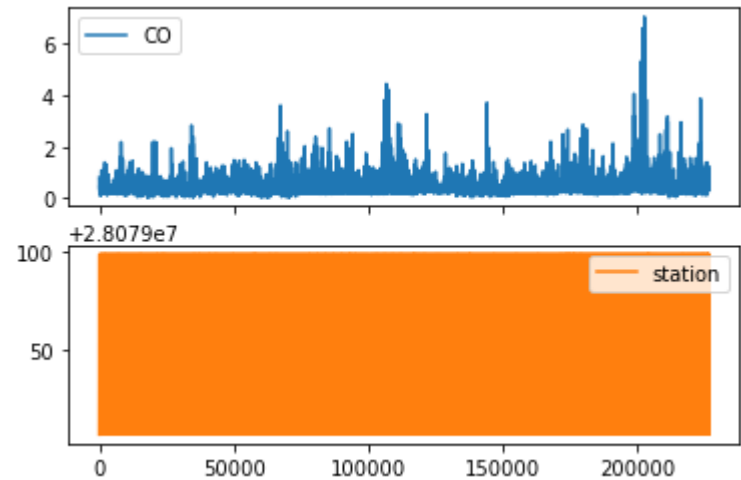
Line chart

In [7]:

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

Out[7]:

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



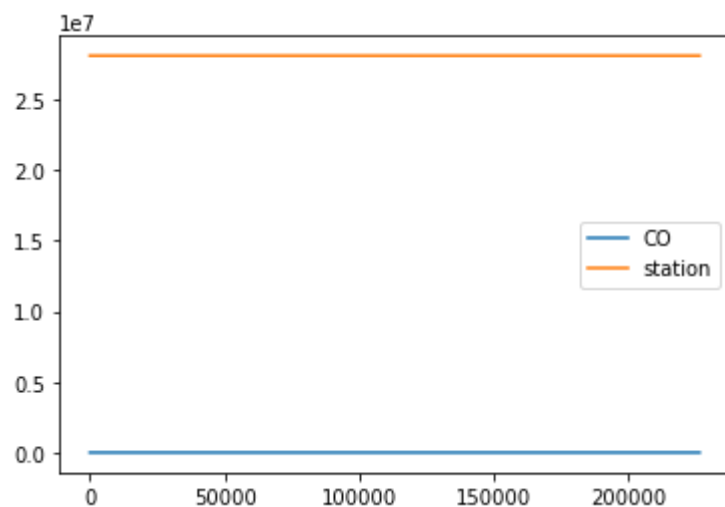
Line chart

In [8]:

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

Out[8]:

<AxesSubplot:>



Bar chart

In [9]:

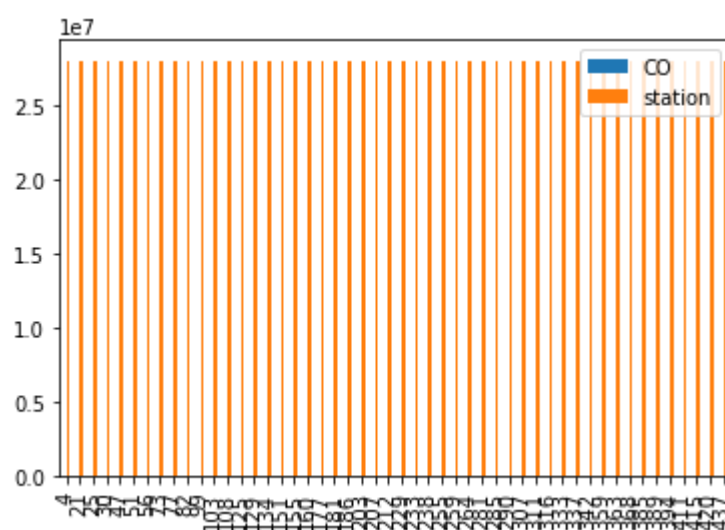
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<AxesSubplot:>



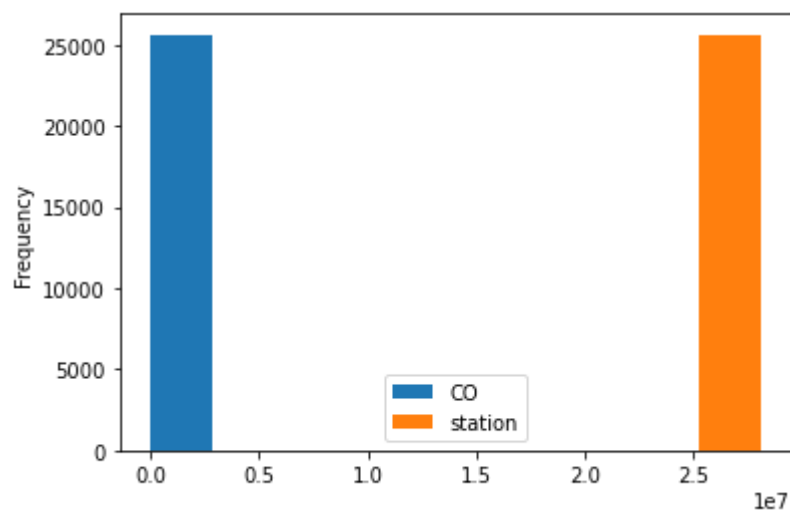
Histogram

In [11]:

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

Out[11]:

<AxesSubplot:ylabel='Frequency'>



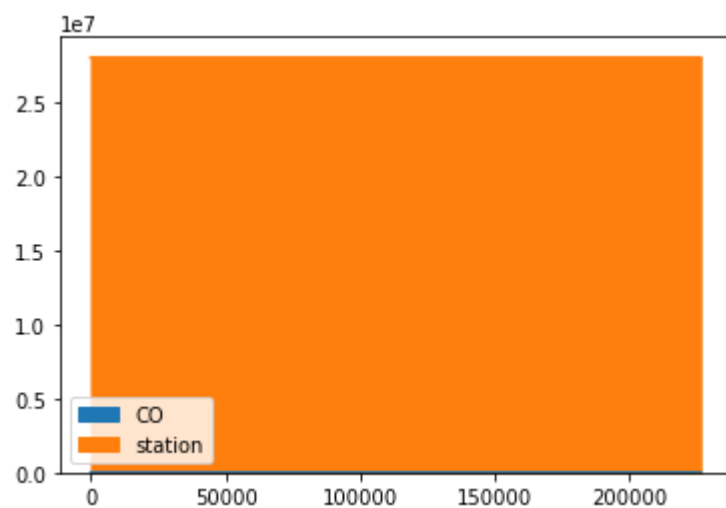
Area chart

In [12]:

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

Out[12]:

<AxesSubplot:>



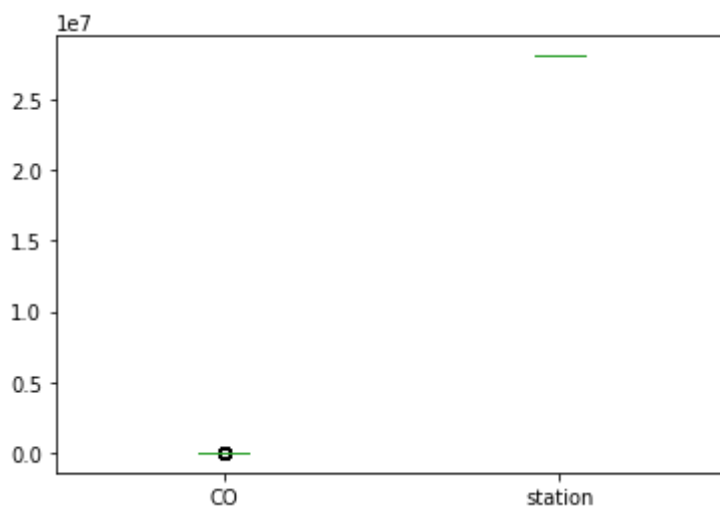
Box chart

In [13]:

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

Out[13]:

<AxesSubplot:>



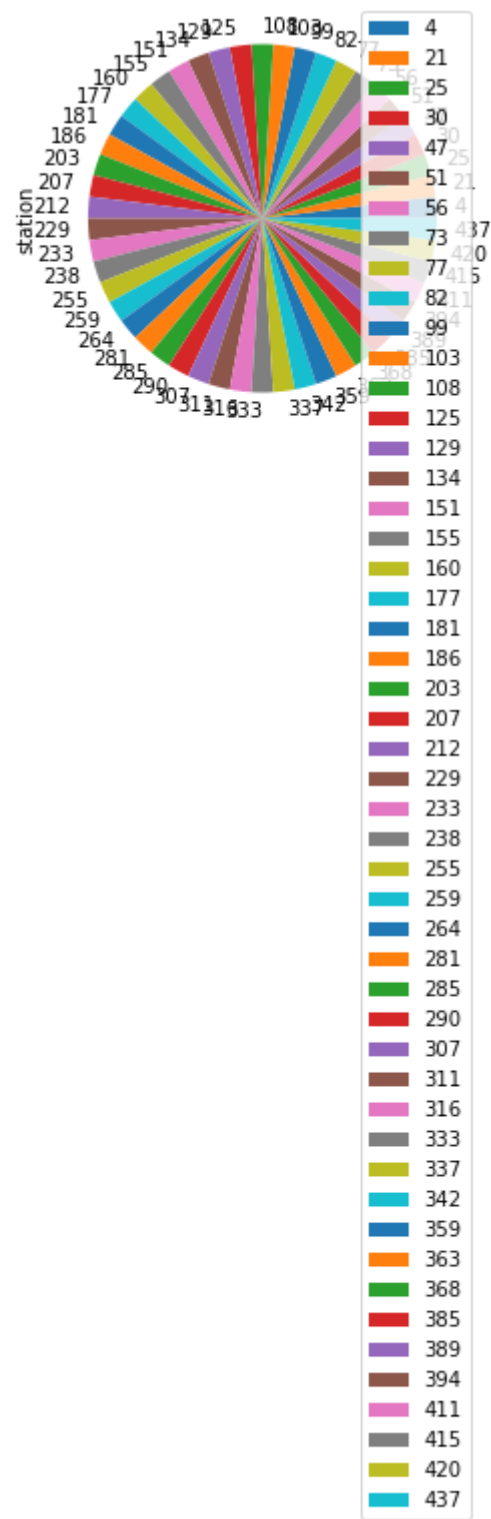
Pie chart

In [14]:

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

Out[14]:

<AxesSubplot:ylabel='station'>



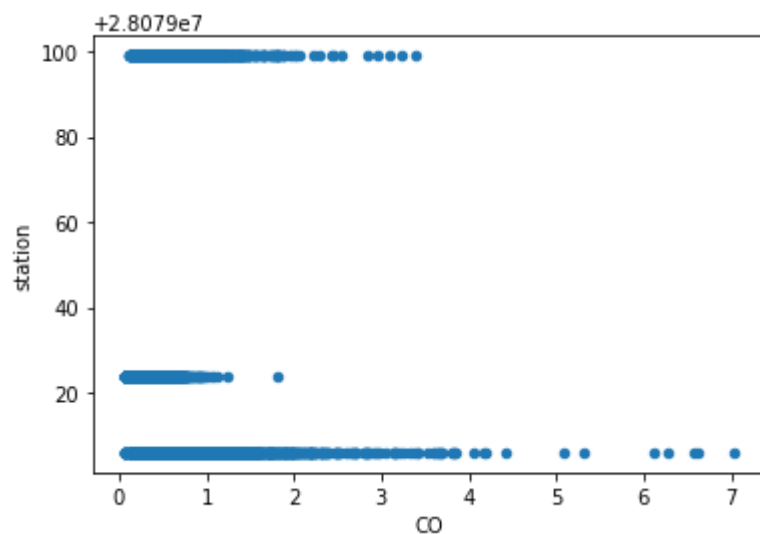
Scatter chart

In [15]:

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

Out[15]:

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



In [16]:

```
df.info()
```

```

---
0  date      25631 non-null  object
1  BEN       25631 non-null  float64
2  CO        25631 non-null  float64
3  EBE       25631 non-null  float64
4  MXY       25631 non-null  float64
5  NMHC      25631 non-null  float64
6  NO_2      25631 non-null  float64
7  NOx       25631 non-null  float64
8  OXY       25631 non-null  float64
9  O_3       25631 non-null  float64
10 PM10      25631 non-null  float64
11 PM25      25631 non-null  float64
12 PXY       25631 non-null  float64
13 SO_2      25631 non-null  float64
14 TCH       25631 non-null  float64
15 TOL       25631 non-null  float64
16 station   25631 non-null  int64
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB

```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000
mean	1.090541	0.440632	1.352355	2.446045	0.213323	54.225261
std	1.146461	0.317853	1.118191	2.390023	0.123409	38.164647
min	0.100000	0.060000	0.170000	0.240000	0.000000	0.240000
25%	0.430000	0.260000	0.740000	1.000000	0.130000	25.719999
50%	0.750000	0.350000	1.000000	1.620000	0.190000	48.000000
75%	1.320000	0.510000	1.580000	3.105000	0.270000	74.924999
max	27.230000	7.030000	26.740000	55.889999	1.760000	554.900024

In [18]:

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

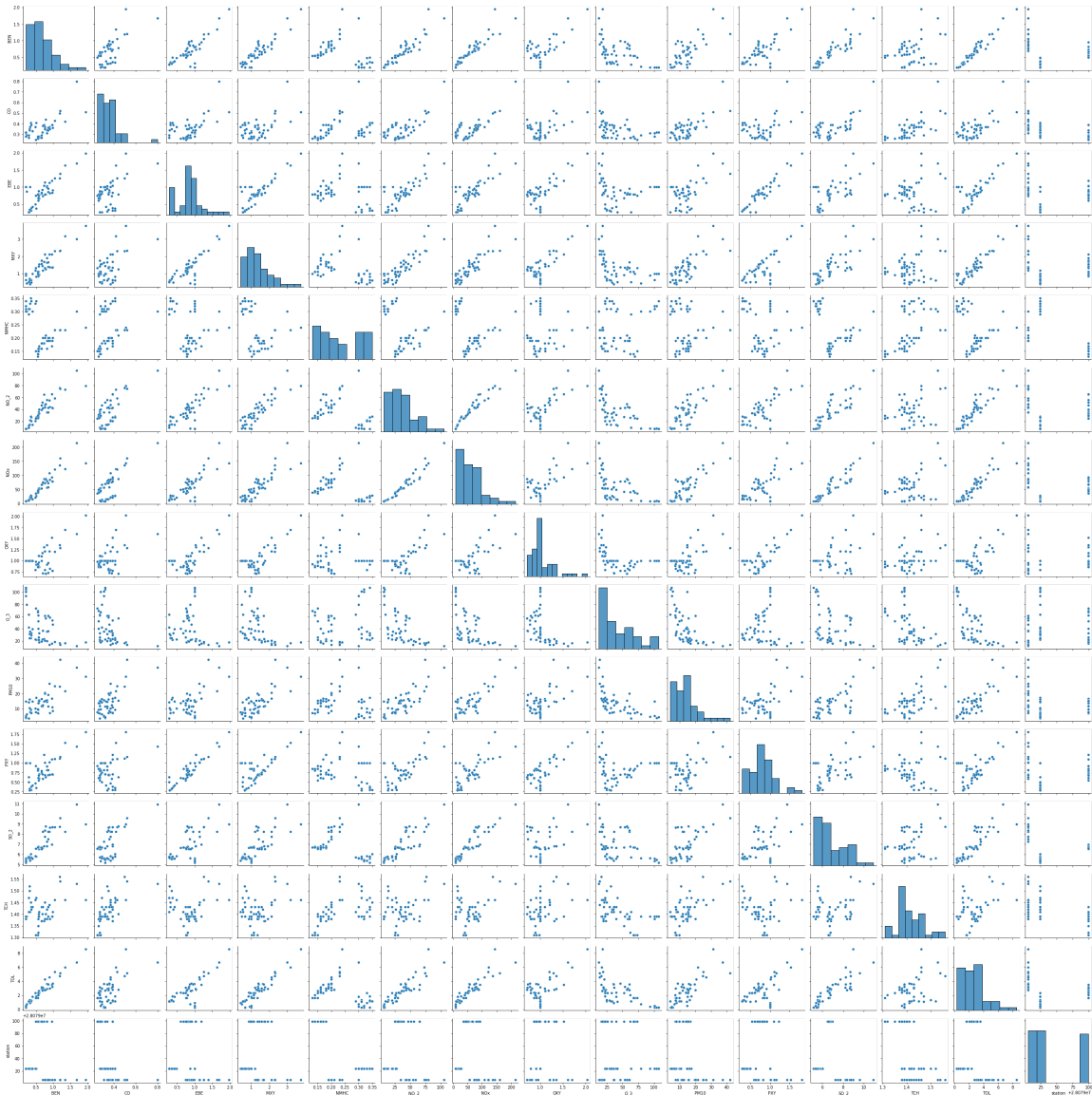
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

<seaborn.axisgrid.PairGrid at 0x20da6286b20>



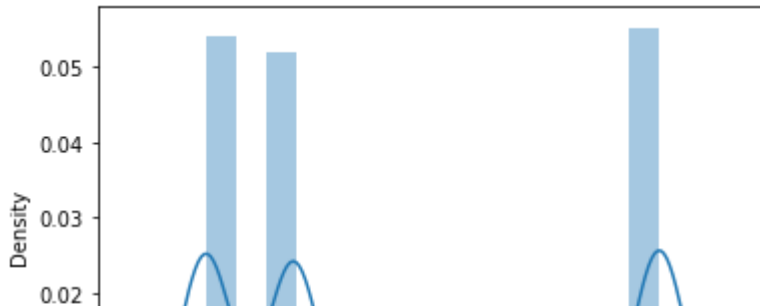
In [20]:

```
sns.distplot(df1['station'])
```

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:255
 7: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
 warnings.warn(msg, FutureWarning)

Out[20]:

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

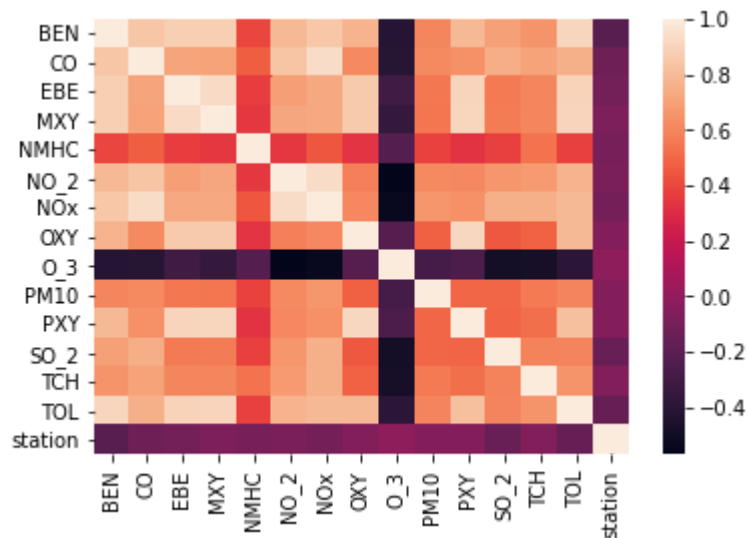


In [21]:

```
sns.heatmap(df1.corr())
```

Out[21]:

```
<AxesSubplot:>
```



TO TRAIN THE MODEL AND MODEL BUILDING

In [22]:

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

In [23]:

```
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 [24]:

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

Out[24]:

LinearRegression()

In [25]:

lr.intercept_

Out[25]:

28079035.12432534

In [26]:

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

Out[26]:

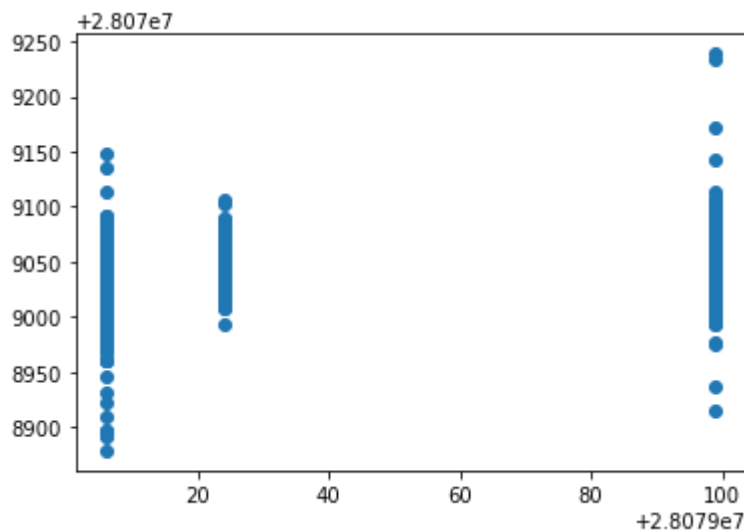
	Co-efficient
BEN	-25.677784
CO	0.039079
EBE	-0.975018
MXY	7.553562
NMHC	-27.887432
NO_2	-0.040768
NOx	0.125295
OXY	4.057897
O_3	-0.143160
PM10	0.140121
PXY	2.327563
SO_2	-0.637859
TCH	17.621654
TOL	-1.908789

In [27]:

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

Out[27]:

<matplotlib.collections.PathCollection at 0x20db4c95700>



ACCURACY

In [28]:

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

Out[28]:

0.1425413978965746

In [29]:

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

Out[29]:

0.14416558651112388

Ridge and Lasso

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)

Accuracy(Ridge)

In [32]:

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

Out[32]:

0.14254972762293971

In [33]:

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

Out[33]:

0.14414036663149943

In [34]:

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

Out[34]:

Lasso(alpha=10)

In [35]:

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

Out[35]:

0.04327523622896379

Accuracy(Lasso)

In [36]:

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

Out[36]:

0.04025531021711837

Elastic Net

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([-4.64079915, -0.          ,  0.          ,  3.19190927, -0.          ,
        0.06050465,  0.02579208,  1.57297725, -0.15712899,  0.13493911,
        1.54451818, -0.95966869,  0.          , -2.49989346])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079057.330976218
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.09278636545773644
```

Evaluation Metrics

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)))
```

```
35.88332447042623
```

```
1499.635061630427
```

```
38.72512184138905
```

Logistic Regression

In [43]:

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

In [44]:

```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXV', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
                  'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```


In [45]:

```
feature_matrix.shape
```

Out[45]:

```
(25631, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(25631,)
```

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

In [51]:

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

```
[28079099]
```

In [52]:

```
logr.classes_
```

Out[52]:

```
array([28079006, 28079024, 28079099], dtype=int64)
```

In [53]:

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

Out[53]:

```
0.794194530061254
```

In [54]:

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

Out[54]:

```
8.321803242555043e-09
```

In [55]:

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

Out[55]:

```
array([[8.32180324e-09, 1.19114634e-13, 9.99999992e-01]])
```

Random Forest

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]:

```
0.8517919874855023
```

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'],f
```

Out[62]:

```

[Text(2232.0, 1993.2, 'OXY <= 1.015\ngini = 0.666\nsamples = 11259\nvalue
= [5957, 5791, 6193]\nclasse = c'),
Text(1116.0, 1630.8000000000002, 'PXY <= 0.985\ngini = 0.605\nsamples = 6
026\nvalue = [2077, 5122, 2372]\nclasse = b'),
Text(558.0, 1268.4, 'NOx <= 21.665\ngini = 0.661\nsamples = 4363\nvalue =
[2012, 2692, 2196]\nclasse = b'),
Text(279.0, 906.0, 'TOL <= 0.85\ngini = 0.23\nsamples = 701\nvalue = [44,
991, 101]\nclasse = b'),
Text(139.5, 543.5999999999999, 'TOL <= 0.775\ngini = 0.047\nsamples = 214
\nvalue = [0, 323, 8]\nclasse = b'),
Text(69.75, 181.19999999999982, 'gini = 0.028\nsamples = 186\nvalue = [0,
276, 4]\nclasse = b'),
Text(209.25, 181.19999999999982, 'gini = 0.145\nsamples = 28\nvalue = [0,
47, 4]\nclasse = b'),
Text(418.5, 543.5999999999999, 'PXY <= 0.295\ngini = 0.295\nsamples = 487
\nvalue = [44, 668, 93]\nclasse = b'),
Text(348.75, 181.19999999999982, 'gini = 0.494\nsamples = 14\nvalue = [1
2, 4, 2]\nclasse = a'),
Text(488.25, 181.19999999999982, 'gini = 0.273\nsamples = 473\nvalue = [3
2, 664, 91]\nclasse = b'),
Text(837.0, 906.0, 'NOx <= 52.785\ngini = 0.664\nsamples = 3662\nvalue =
[1968, 1701, 2095]\nclasse = c'),
Text(697.5, 543.5999999999999, 'PXY <= 0.495\ngini = 0.64\nsamples = 1712
\nvalue = [547, 1069, 1102]\nclasse = c'),
Text(627.75, 181.19999999999982, 'gini = 0.603\nsamples = 453\nvalue = [3
04, 339, 95]\nclasse = b'),
Text(767.25, 181.19999999999982, 'gini = 0.59\nsamples = 1259\nvalue = [2
43, 730, 1007]\nclasse = c'),
Text(976.5, 543.5999999999999, 'CO <= 0.245\ngini = 0.633\nsamples = 1950
\nvalue = [1421, 632, 993]\nclasse = a'),
Text(906.75, 181.19999999999982, 'gini = 0.521\nsamples = 253\nvalue = [9
9, 253, 46]\nclasse = b'),
Text(1046.25, 181.19999999999982, 'gini = 0.602\nsamples = 1697\nvalue =
[1322, 379, 947]\nclasse = a'),
Text(1674.0, 1268.4, 'NO_2 <= 39.67\ngini = 0.167\nsamples = 1663\nvalue
= [65, 2430, 176]\nclasse = b'),
Text(1395.0, 906.0, 'MXV <= 1.29\ngini = 0.032\nsamples = 1332\nvalue =
[8, 2134, 27]\nclasse = b'),
Text(1255.5, 543.5999999999999, 'TOL <= 1.635\ngini = 0.017\nsamples = 12
87\nvalue = [5, 2074, 13]\nclasse = b'),
Text(1185.75, 181.19999999999982, 'gini = 0.008\nsamples = 1229\nvalue =
[4, 1993, 4]\nclasse = b'),
Text(1325.25, 181.19999999999982, 'gini = 0.198\nsamples = 58\nvalue =
[1, 81, 9]\nclasse = b'),
Text(1534.5, 543.5999999999999, 'CO <= 0.265\ngini = 0.358\nsamples = 45
\nvalue = [3, 60, 14]\nclasse = b'),
Text(1464.75, 181.19999999999982, 'gini = 0.574\nsamples = 15\nvalue =
[3, 16, 12]\nclasse = b'),
Text(1604.25, 181.19999999999982, 'gini = 0.083\nsamples = 30\nvalue =
[0, 44, 2]\nclasse = b'),
Text(1953.0, 906.0, 'OXY <= 0.595\ngini = 0.551\nsamples = 331\nvalue =
[57, 296, 149]\nclasse = b'),
Text(1813.5, 543.5999999999999, 'EBE <= 0.705\ngini = 0.098\nsamples = 11
1\nvalue = [0, 165, 9]\nclasse = b'),
Text(1743.75, 181.19999999999982, 'gini = 0.48\nsamples = 9\nvalue = [0,
12, 8]\nclasse = b'),
Text(1883.25, 181.19999999999982, 'gini = 0.013\nsamples = 102\nvalue =
[0, 153, 1]\nclasse = b'),
Text(2092.5, 543.5999999999999, 'CO <= 0.285\ngini = 0.628\nsamples = 220
\nvalue = [57, 131, 140]\nclasse = c'),
Text(2022.75, 181.19999999999982, 'gini = 0.427\nsamples = 62\nvalue = [1

```

```

2, 66, 12]\nclasse = b'),
Text(2162.25, 181.19999999999982, 'gini = 0.6\nsamples = 158\nvalue = [4
5, 65, 128]\nclasse = c'),
Text(3348.0, 1630.8000000000002, 'TOL <= 5.415\ngini = 0.57\nsamples = 52
33\nvalue = [3880, 669, 3821]\nclasse = a'),
Text(2790.0, 1268.4, 'SO_2 <= 8.055\ngini = 0.445\nsamples = 2158\nvalue
= [562, 417, 2476]\nclasse = c'),
Text(2511.0, 906.0, 'TOL <= 1.67\ngini = 0.212\nsamples = 978\nvalue = [9
9, 83, 1383]\nclasse = c'),
Text(2371.5, 543.5999999999999, 'TOL <= 1.055\ngini = 0.459\nsamples = 41
\nvalue = [1, 48, 24]\nclasse = b'),
Text(2301.75, 181.19999999999982, 'gini = 0.0\nsamples = 17\nvalue = [0,
30, 0]\nclasse = b'),
Text(2441.25, 181.19999999999982, 'gini = 0.513\nsamples = 24\nvalue =
[1, 18, 24]\nclasse = c'),
Text(2650.5, 543.5999999999999, 'NMHC <= 0.095\ngini = 0.165\nsamples = 9
37\nvalue = [98, 35, 1359]\nclasse = c'),
Text(2580.75, 181.19999999999982, 'gini = 0.483\nsamples = 39\nvalue = [3
9, 1, 23]\nclasse = a'),
Text(2720.25, 181.19999999999982, 'gini = 0.124\nsamples = 898\nvalue =
[59, 34, 1336]\nclasse = c'),
Text(3069.0, 906.0, 'BEN <= 0.495\ngini = 0.574\nsamples = 1180\nvalue =
[463, 334, 1093]\nclasse = c'),
Text(2929.5, 543.5999999999999, 'MXV <= 1.38\ngini = 0.34\nsamples = 131
\nvalue = [2, 158, 41]\nclasse = b'),
Text(2919.0, 181.19999999999982, 'gini = 0.029\nsamples = 87\nvalue =
[0, 133, 2]\nclasse = b'),
Text(2999.25, 181.19999999999982, 'gini = 0.506\nsamples = 44\nvalue =
[2, 25, 39]\nclasse = c'),
Text(3208.5, 543.5999999999999, 'TCH <= 1.275\ngini = 0.527\nsamples = 10
49\nvalue = [461, 176, 1052]\nclasse = c'),
Text(3138.75, 181.19999999999982, 'gini = 0.103\nsamples = 39\nvalue =
[3, 52, 0]\nclasse = b'),
Text(3278.25, 181.19999999999982, 'gini = 0.501\nsamples = 1010\nvalue =
[458, 124, 1052]\nclasse = c'),
Text(3439.0, 1268.4, 'TCH <= 1.605\ngini = 0.467\nsamples = 3075\nvalue =
[3318, 252, 1345]\nclasse = a'),
Text(3617.0, 906.0, 'SO_2 <= 8.505\ngini = 0.427\nsamples = 2329\nvalue =
[2694, 193, 857]\nclasse = a'),
Text(3487.5, 543.5999999999999, 'TOL <= 6.63\ngini = 0.512\nsamples = 314
0\nvalue = [1425, 1397, 895]\nclasse = c'),
Text(3417.75, 181.19999999999982, 'gini = 0.433\nsamples = 178\nvalue =
[83, 5, 202]\nclasse = c'),
Text(3557.25, 181.19999999999982, 'gini = 0.548\nsamples = 136\nvalue =
[109, 12, 92]\nclasse = a'),
Text(3766.5, 543.5999999999999, 'TOL <= 8.925\ngini = 0.371\nsamples = 20
15\nvalue = [2502, 176, 563]\nclasse = a'),
Text(4165.0, 181.19999999999982, 'gini = 0.523\nsamples = 1005\nvalue =
[1014, 150, 465]\nclasse = a'),
Text(3836.25, 181.19999999999982, 'gini = 0.144\nsamples = 1010\nvalue =
[1488, 26, 98]\nclasse = a'),
Text(4045.5, 543.5999999999999, 'CO <= 0.465\ngini = 0.105\nsamples = 171
\nvalue = [242, 7, 7]\nclasse = a'),
Text(3975.75, 181.19999999999982, 'gini = 0.245\nsamples = 5\nvalue = [1,
6, 0]\nclasse = b'),
Text(4115.25, 181.19999999999982, 'gini = 0.062\nsamples = 166\nvalue =
[241, 1, 37]\nclasse = a'),
Text(4324.5, 543.5999999999999, 'O_3 <= 11.85\ngini = 0.546\nsamples = 57
5\nvalue = [382, 52, 481]\nclasse = c'),

```

Conclusion

Scores

Linear Regression

```

In [64]:
In [65]:
In [66]:
In [67]:
In [68]:
In [69]:
In [70]:
In [71]:
In [72]:
In [73]:
In [74]:
In [75]:
In [76]:
In [77]:
In [78]:
In [79]:
In [80]:
In [81]:
In [82]:
In [83]:
In [84]:
In [85]:
In [86]:
In [87]:
In [88]:
In [89]:
In [90]:
In [91]:
In [92]:
In [93]:
In [94]:
In [95]:
In [96]:
In [97]:
In [98]:
In [99]:
In [100]:
In [101]:
In [102]:
In [103]:
In [104]:
In [105]:
In [106]:
In [107]:
In [108]:
In [109]:
In [110]:
In [111]:
In [112]:
In [113]:
In [114]:
In [115]:
In [116]:
In [117]:
In [118]:
In [119]:
In [120]:
In [121]:
In [122]:
In [123]:
In [124]:
In [125]:
In [126]:
In [127]:
In [128]:
In [129]:
In [130]:
In [131]:
In [132]:
In [133]:
In [134]:
In [135]:
In [136]:
In [137]:
In [138]:
In [139]:
In [140]:
In [141]:
In [142]:
In [143]:
In [144]:
In [145]:
In [146]:
In [147]:
In [148]:
In [149]:
In [150]:
In [151]:
In [152]:
In [153]:
In [154]:
In [155]:
In [156]:
In [157]:
In [158]:
In [159]:
In [160]:
In [161]:
In [162]:
In [163]:
In [164]:
In [165]:
In [166]:
In [167]:
In [168]:
In [169]:
In [170]:
In [171]:
In [172]:
In [173]:
In [174]:
In [175]:
In [176]:
In [177]:
In [178]:
In [179]:
In [180]:
In [181]:
In [182]:
In [183]:
In [184]:
In [185]:
In [186]:
In [187]:
In [188]:
In [189]:
In [190]:
In [191]:
In [192]:
In [193]:
In [194]:
In [195]:
In [196]:
In [197]:
In [198]:
In [199]:
In [200]:
In [201]:
In [202]:
In [203]:
In [204]:
In [205]:
In [206]:
In [207]:
In [208]:
In [209]:
In [210]:
In [211]:
In [212]:
In [213]:
In [214]:
In [215]:
In [216]:
In [217]:
In [218]:
In [219]:
In [220]:
In [221]:
In [222]:
In [223]:
In [224]:
In [225]:
In [226]:
In [227]:
In [228]:
In [229]:
In [230]:
In [231]:
In [232]:
In [233]:
In [234]:
In [235]:
In [236]:
In [237]:
In [238]:
In [239]:
In [240]:
In [241]:
In [242]:
In [243]:
In [244]:
In [245]:
In [246]:
In [247]:
In [248]:
In [249]:
In [250]:
In [251]:
In [252]:
In [253]:
In [254]:
In [255]:
In [256]:
In [257]:
In [258]:
In [259]:
In [260]:
In [261]:
In [262]:
In [263]:
In [264]:
In [265]:
In [266]:
In [267]:
In [268]:
In [269]:
In [270]:
In [271]:
In [272]:
In [273]:
In [274]:
In [275]:
In [276]:
In [277]:
In [278]:
In [279]:
In [280]:
In [281]:
In [282]:
In [283]:
In [284]:
In [285]:
In [286]:
In [287]:
In [288]:
In [289]:
In [290]:
In [291]:
In [292]:
In [293]:
In [294]:
In [295]:
In [296]:
In [297]:
In [298]:
In [299]:
In [300]:
In [301]:
In [302]:
In [303]:
In [304]:
In [305]:
In [306]:
In [307]:
In [308]:
In [309]:
In [310]:
In [311]:
In [312]:
In [313]:
In [314]:
In [315]:
In [316]:
In [317]:
In [318]:
In [319]:
In [320]:
In [321]:
In [322]:
In [323]:
In [324]:
In [325]:
In [326]:
In [327]:
In [328]:
In [329]:
In [330]:
In [331]:
In [332]:
In [333]:
In [334]:
In [335]:
In [336]:
In [337]:
In [338]:
In [339]:
In [340]:
In [341]:
In [342]:
In [343]:
In [344]:
In [345]:
In [346]:
In [347]:
In [348]:
In [349]:
In [350]:
In [351]:
In [352]:
In [353]:
In [354]:
In [355]:
In [356]:
In [357]:
In [358]:
In [359]:
In [360]:
In [361]:
In [362]:
In [363]:
In [364]:
In [365]:
In [366]:
In [367]:
In [368]:
In [369]:
In [370]:
In [371]:
In [372]:
In [373]:
In [374]:
In [375]:
In [376]:
In [377]:
In [378]:
In [379]:
In [380]:
In [381]:
In [382]:
In [383]:
In [384]:
In [385]:
In [386]:
In [387]:
In [388]:
In [389]:
In [390]:
In [391]:
In [392]:
In [393]:
In [394]:
In [395]:
In [396]:
In [397]:
In [398]:
In [399]:
In [400]:
In [401]:
In [402]:
In [403]:
In [404]:
In [405]:
In [406]:
In [407]:
In [408]:
In [409]:
In [410]:
In [411]:
In [412]:
In [413]:
In [414]:
In [415]:
In [416]:
In [417]:
In [418]:
In [419]:
In [420]:
In [421]:
In [422]:
In [423]:
In [424]:
In [425]:
In [426]:
In [427]:
In [428]:
In [429]:
In [430]:
In [431]:
In [432]:
In [433]:
In [434]:
In [435]:
In [436]:
In [437]:
In [438]:
In [439]:
In [440]:
In [441]:
In [442]:
In [443]:
In [444]:
In [445]:
In [446]:
In [447]:
In [448]:
In [449]:
In [450]:
In [451]:
In [452]:
In [453]:
In [454]:
In [455]:
In [456]:
In [457]:
In [458]:
In [459]:
In [460]:
In [461]:
In [462]:
In [463]:
In [464]:
In [465]:
In [466]:
In [467]:
In [468]:
In [469]:
In [470]:
In [471]:
In [472]:
In [473]:
In [474]:
In [475]:
In [476]:
In [477]:
In [478]:
In [479]:
In [480]:
In [481]:
In [482]:
In [483]:
In [484]:
In [485]:
In [486]:
In [487]:
In [488]:
In [489]:
In [490]:
In [491]:
In [492]:
In [493]:
In [494]:
In [495]:
In [496]:
In [497]:
In [498]:
In [499]:
In [500]:
In [501]:
In [502]:
In [503]:
In [504]:
In [505]:
In [506]:
In [507]:
In [508]:
In [509]:
In [510]:
In [511]:
In [512]:
In [513]:
In [514]:
In [515]:
In [516]:
In [517]:
In [518]:
In [519]:
In [520]:
In [521]:
In [522]:
In [523]:
In [524]:
In [525]:
In [526]:
In [527]:
In [528]:
In [529]:
In [530]:
In [531]:
In [532]:
In [533]:
In [534]:
In [535]:
In [536]:
In [537]:
In [538]:
In [539]:
In [540]:
In [541]:
In [542]:
In [543]:
In [544]:
In [545]:
In [546]:
In [547]:
In [548]:
In [549]:
In [550]:
In [551]:
In [552]:
In [553]:
In [554]:
In [555]:
In [556]:
In [557]:
In [558]:
In [559]:
In [560]:
In [561]:
In [562]:
In [563]:
In [564]:
In [565]:
In [566]:
In [567]:
In [568]:
In [569]:
In [570]:
In [571]:
In [572]:
In [573]:
In [574]:
In [575]:
In [576]:
In [577]:
In [578]:
In [579]:
In [580]:
In [581]:
In [582]:
In [583]:
In [584]:
In [585]:
In [586]:
In [587]:
In [588]:
In [589]:
In [590]:
In [591]:
In [592]:
In [593]:
In [594]:
In [595]:
In [596]:
In [597]:
In [598]:
In [599]:
In [600]:
In [601]:
In [602]:
In [603]:
In [604]:
In [605]:
In [606]:
In [607]:
In [608]:
In [609]:
In [610]:
In [611]:
In [612]:
In [613]:
In [614]:
In [615]:
In [616]:
In [617]:
In [618]:
In [619]:
In [620]:
In [621]:
In [622]:
In [623]:
In [624]:
In [625]:
In [626]:
In [627]:
In [628]:
In [629]:
In [630]:
In [631]:
In [632]:
In [633]:
In [634]:
In [635]:
In [636]:
In [637]:
In [638]:
In [639]:
In [640]:
In [641]:
In [642]:
In [643]:
In [644]:
In [645]:
In [646]:
In [647]:
In [648]:
In [649]:
In [650]:
In [651]:
In [652]:
In [653]:
In [654]:
In [655]:
In [656]:
In [657]:
In [658]:
In [659]:
In [660]:
In [661]:
In [662]:
In [663]:
In [664]:
In [665]:
In [666]:
In [667]:
In [668]:
In [669]:
In [670]:
In [671]:
In [672]:
In [673]:
In [674]:
In [675]:
In [676]:
In [677]:
In [678]:
In [679]:
In [680]:
In [681]:
In [682]:
In [683]:
In [684]:
In [685]:
In [686]:
In [687]:
In [688]:
In [689]:
In [690]:
In [691]:
In [692]:
In [693]:
In [694]:
In [695]:
In [696]:
In [697]:
In [698]:
In [699]:
In [700]:
In [701]:
In [702]:
In [703]:
In [704]:
In [705]:
In [706]:
In [707]:
In [708]:
In [709]:
In [710]:
In [711]:
In [712]:
In [713]:
In [714]:
In [715]:
In [716]:
In [717]:
In [718]:
In [719]:
In [720]:
In [721]:
In [722]:
In [723]:
In [724]:
In [725]:
In [726]:
In [727]:
In [728]:
In [729]:
In [730]:
In [731]:
In [732]:
In [733]:
In [734]:
In [735]:
In [736]:
In [737]:
In [738]:
In [739]:
In [740]:
In [741]:
In [742]:
In [743]:
In [744]:
In [745]:
In [746]:
In [747]:
In [748]:
In [749]:
In [750]:
In [751]:
In [752]:
In [753]:
In [754]:
In [755]:
In [756]:
In [757]:
In [758]:
In [759]:
In [760]:
In [761]:
In [762]:
In [763]:
In [764]:
In [765]:
In [766]:
In [767]:
In [768]:
In [769]:
In [770]:
In [771]:
In [772]:
In [773]:
In [774]:
In [775]:
In [776]:
In [777]:
In [778]:
In [779]:
In [780]:
In [781]:
In [782]:
In [783]:
In [784]:
In [785]:
In [786]:
In [787]:
In [788]:
In [789]:
In [790]:
In [791]:
In [792]:
In [793]:
In [794]:
In [795]:
In [796]:
In [797]:
In [798]:
In [799]:
In [800]:
In [801]:
In [802]:
In [803]:
In [804]:
In [805]:
In [806]:
In [807]:
In [808]:
In [809]:
In [810]:
In [811]:
In [812]:
In [813]:
In [814]:
In [815]:
In [816]:
In [817]:
In [818]:
In [819]:
In [820]:
In [821]:
In [822]:
In [823]:
In [824]:
In [825]:
In [826]:
In [827]:
In [828]:
In [829]:
In [830]:
In [831]:
In [832]:
In [833]:
In [834]:
In [835]:
In [836]:
In [837]:
In [838]:
In [839]:
In [840]:
In [841]:
In [842]:
In [843]:
In [844]:
In [845]:
In [846]:
In [847]:
In [848]:
In [849]:
In [850]:
In [851]:
In [852]:
In [853]:
In [854]:
In [855]:
In [856]:
In [857]:
In [858]:
In [859]:
In [860]:
In [861]:
In [862]:
In [863]:
In [864]:
In [865]:
In [866]:
In [867]:
In [868]:
In [869]:
In [870]:
In [871]:
In [872]:
In [873]:
In [874]:
In [875]:
In [876]:
In [877]:
In [878]:
In [879]:
In [880]:
In [881]:
In [882]:
In [883]:
In [884]:
In [885]:
In [886]:
In [887]:
In [888]:
In [889]:
In [890]:
In [891]:
In [892]:
In [893]:
In [894]:
In [895]:
In [896]:
In [897]:
In [898]:
In [899]:
In [900]:
In [901]:
In [902]:
In [903]:
In [904]:
In [905]:
In [906]:
In [907]:
In [908]:
In [909]:
In [910]:
In [911]:
In [912]:
In [913]:
In [914]:
In [915]:
In [916]:
In [917]:
In [918]:
In [919]:
In [920]:
In [921]:
In [922]:
In [923]:
In [924]:
In [925]:
In [926]:
In [927]:
In [928]:
In [929]:
In [930]:
In [931]:
In [932]:
In [933]:
In [934]:
In [935]:
In [936]:
In [937]:
In [938]:
In [939]:
In [940]:
In [941]:
In [942]:
In [943]:
In [944]:
In [945]:
In [946]:
In [947]:
In [948]:
In [949]:
In [950]:
In [951]:
In [952]:
In [953]:
In [954]:
In [955]:
In [956]:
In [957]:
In [958]:
In [959]:
In [960]:
In [961]:
In [962]:
In [963]:
In [964]:
In [965]:
In [966]:
In [967]:
In [968]:
In [969]:
In [970]:
In [971]:
In [972]:
In [973]:
In [974]:
In [975]:
In [976]:
In [977]:
In [978]:
In [979]:
In [980]:
In [981]:
In [982]:
In [983]:
In [984]:
In [985]:
In [986]:
In [987]:
In [988]:
In [989]:
In [990]:
In [991]:
In [992]:
In [993]:
In [994]:
In [995]:
In [996]:
In [997]:
In [998]:
In [999]:

```

Text(4254.75, 181.19999999999982, 'gini = 0.418\nsamples = 341\nvalue = [136, 17, 394]\nnclass = c'),
Text(404.25, 181.19999999999982, 'gini = 0.488\nsamples = 234\nvalue = [246, 35, 87]\nnclass = a')]

In [68]:

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

Out[68]:

0.14254972762293971

In [69]:

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

Out[69]:

0.14414036663149943

Elastic Net

In [71]:

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

Out[71]:

0.09278636545773644

Logistic Regression

In [70]:

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

Out[70]:

0.794194530061254

Random Forest

In [72]:

```
grid_search.best_score_
```

Out[72]:

0.8517919874855023

From the above data, we can conclude that random forest is preferable to other regression types

In []: