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("2005.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2005-11-01 01:00:00	NaN	0.77	NaN	NaN	NaN	57.130001	128.699997	NaN	14.720000	14.
1	2005-11-01 01:00:00	1.52	0.65	1.49	4.57	0.25	86.559998	181.699997	1.27	11.680000	30.
2	2005-11-01 01:00:00	NaN	0.40	NaN	NaN	NaN	46.119999	53.000000	NaN	30.469999	14.
3	2005-11-01 01:00:00	NaN	0.42	NaN	NaN	NaN	37.220001	52.009998	NaN	21.379999	15.
4	2005-11-01 01:00:00	NaN	0.57	NaN	NaN	NaN	32.160000	36.680000	NaN	33.410000	5.
...	...	...	...	...	...	...	...	...	...	...	
236995	2006-01-01 00:00:00	1.08	0.36	1.01	NaN	0.11	21.990000	23.610001	NaN	43.349998	5.
236996	2006-01-01 00:00:00	0.39	0.54	1.00	1.00	0.11	2.200000	4.220000	1.00	69.639999	4.
236997	2006-01-01 00:00:00	0.19	NaN	0.26	NaN	0.08	26.730000	30.809999	NaN	43.840000	4.
236998	2006-01-01 00:00:00	0.14	NaN	1.00	NaN	0.06	13.770000	17.770000	NaN	NaN	5.
236999	2006-01-01 00:00:00	0.50	0.40	0.73	1.84	0.13	20.940001	26.950001	1.49	48.259998	5.

237000 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: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     20070 non-null  object 
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
 14  TCH      20070 non-null  float64
 15  TOL      20070 non-null  float64
 16  station  20070 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.8+ MB
```

In [6]:

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

Out[6]:

	CO	station
5	0.88	28079006
22	0.22	28079024
25	0.49	28079099
31	0.84	28079006
48	0.20	28079024
...	...	...
236970	0.39	28079024
236973	0.45	28079099
236979	0.38	28079006
236996	0.54	28079024
236999	0.40	28079099

20070 rows × 2 columns

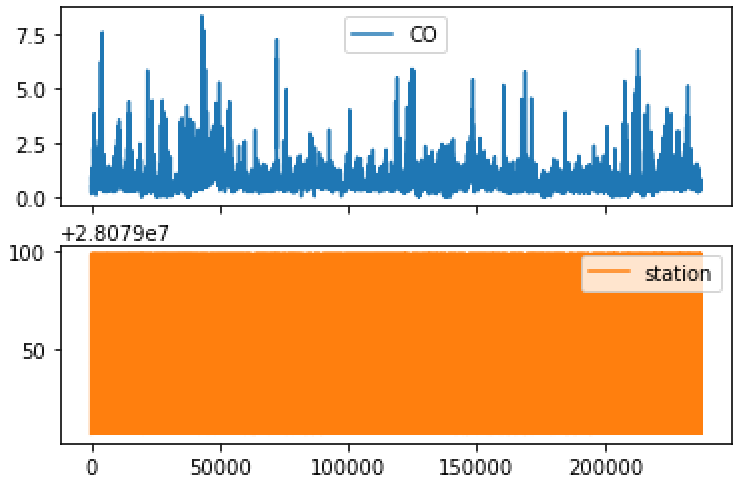
Line chart

In [7]:

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

Out[7]:

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



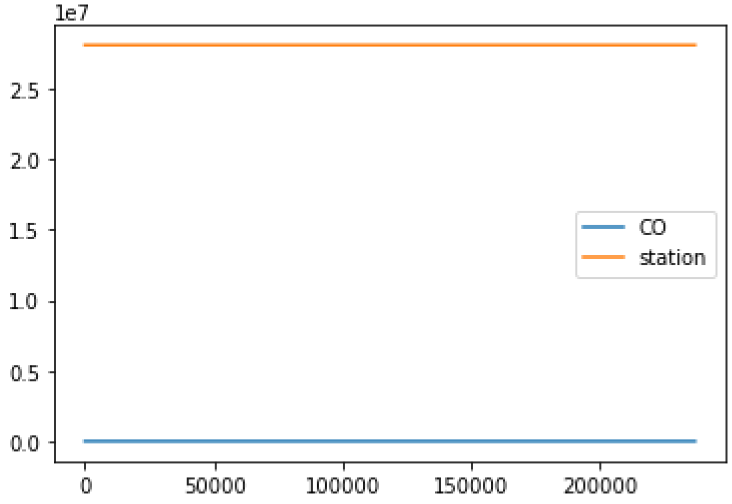
Line chart

In [8]:

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

Out[8]:

```
<AxesSubplot:>
```



Bar chart

In [9]:

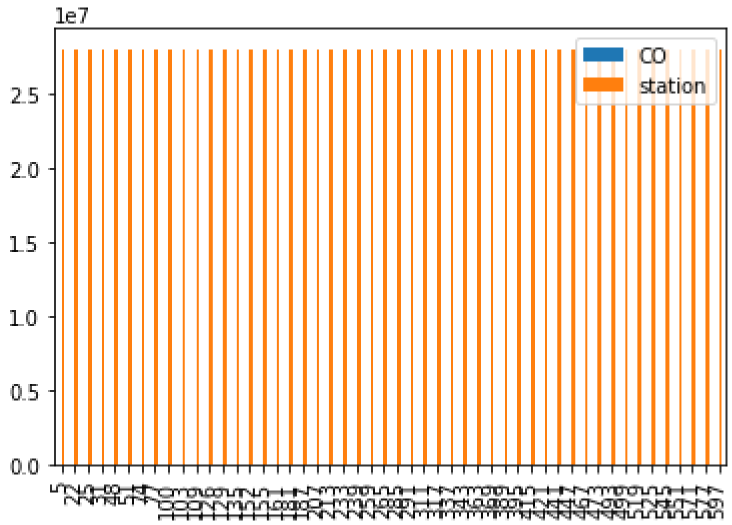
```
b=data[0:50]
```

In [10]:

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

Out[10]:

```
<AxesSubplot:>
```



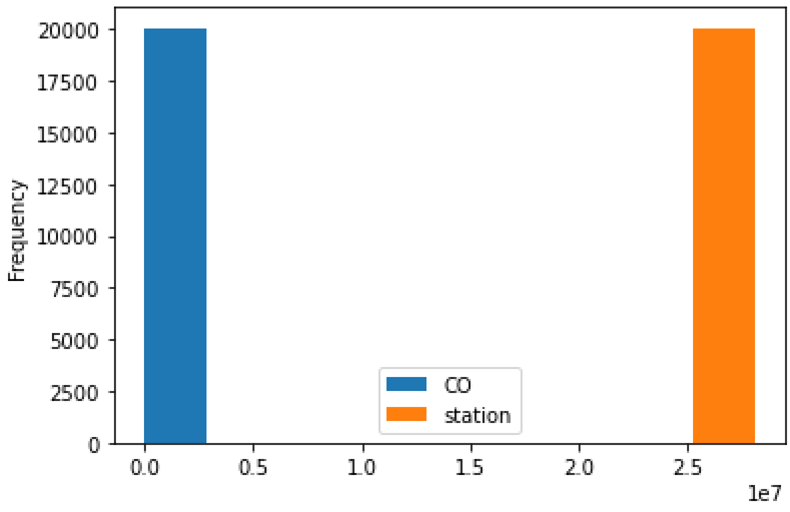
Histogram

In [11]:

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

Out[11]:

<AxesSubplot:ylabel='Frequency'>



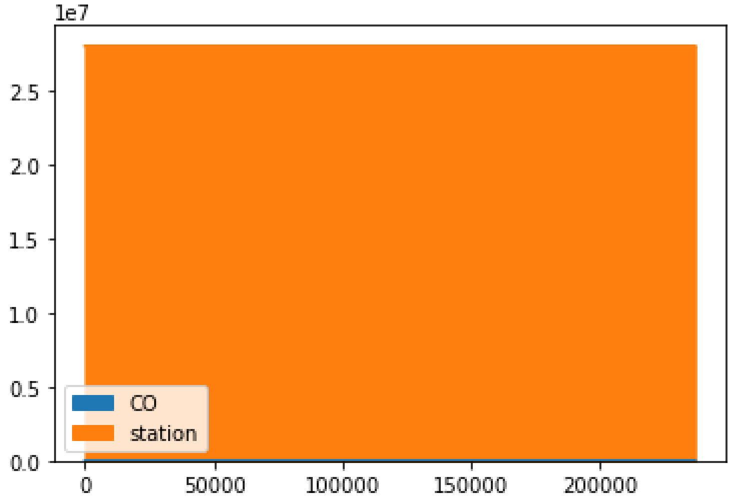
Area chart

In [12]:

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

Out[12]:

<AxesSubplot:>



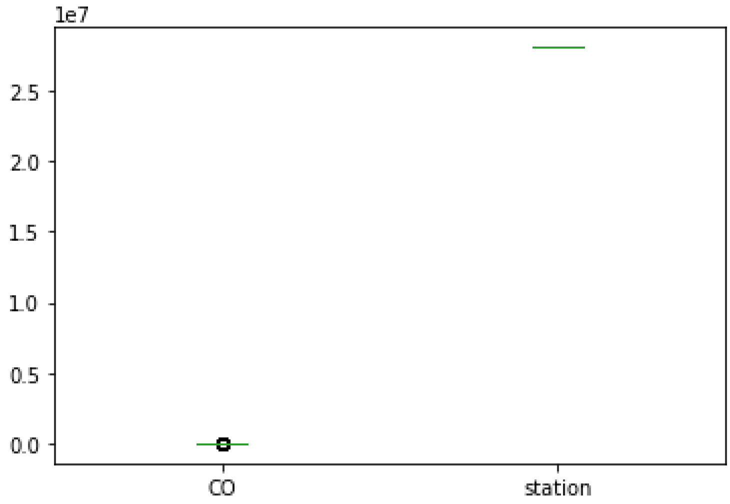
Box chart

In [13]:

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

Out[13]:

```
<AxesSubplot:>
```



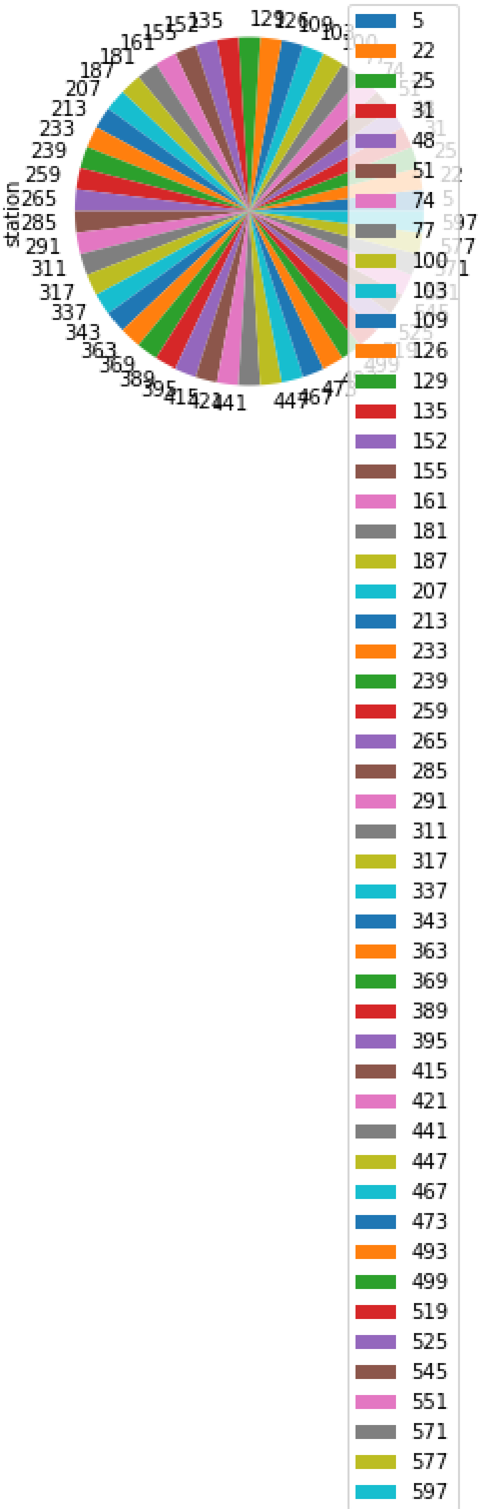
Pie chart

In [14]:

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

Out[14]:

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



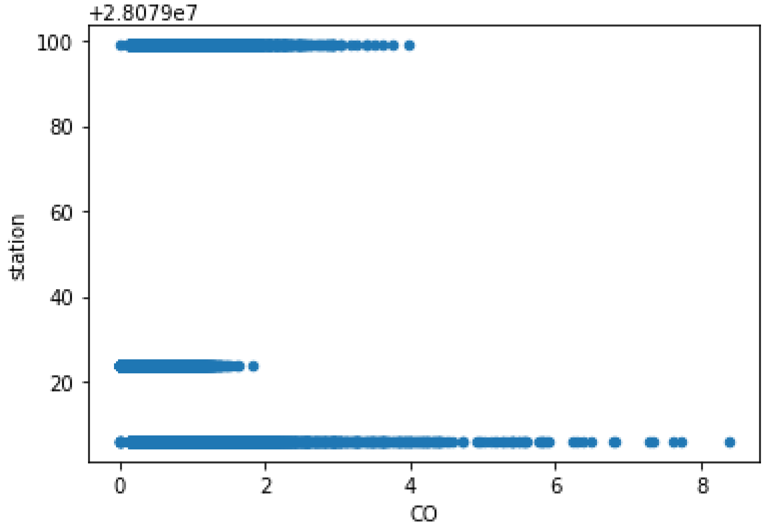
Scatter chart

In [15]:

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

Out[15]:

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



In [16]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     20070 non-null  object
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000
mean	1.923656	0.720657	2.345423	5.457855	0.179282	66.226924
std	2.019061	0.549723	2.379219	5.495147	0.152783	40.568197
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.690000	0.400000	0.950000	1.930000	0.090000	36.602499
50%	1.260000	0.580000	1.480000	3.800000	0.150000	60.525000
75%	2.510000	0.880000	2.950000	7.210000	0.220000	89.317499
max	26.570000	8.380000	29.870001	71.050003	1.880000	419.500000

In [18]:

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

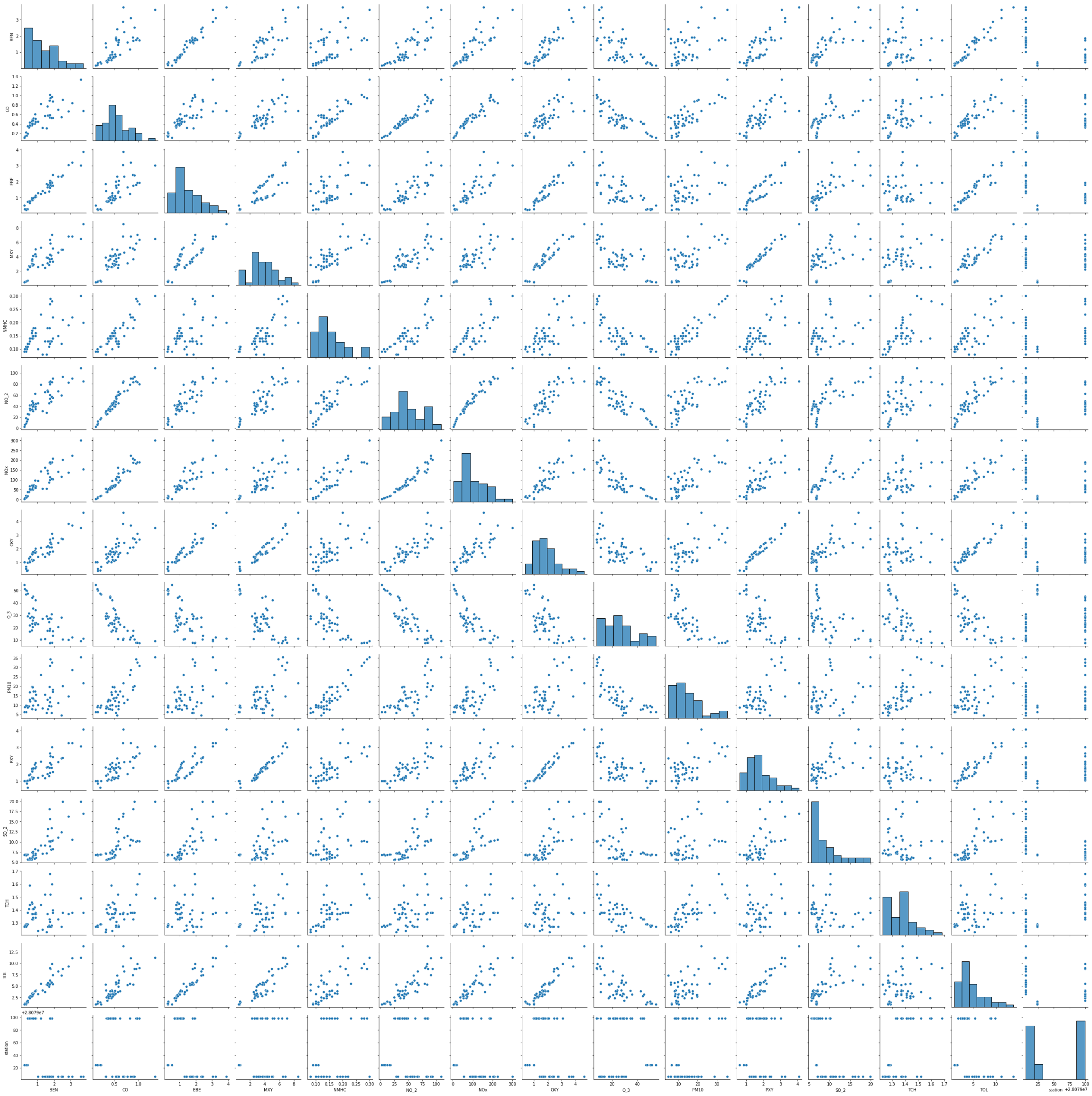
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x26050fd4d00>
```



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 remove
d in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-l
evel function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]:

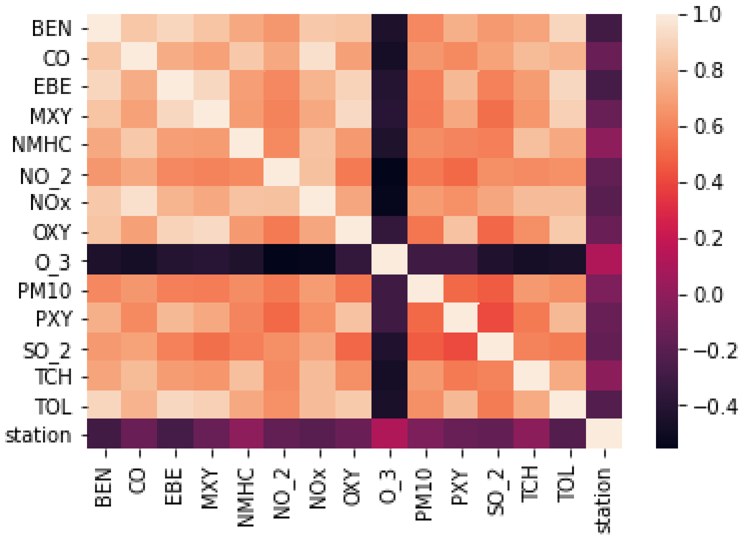
<AxesSubplot:xlabel='station', ylabel='Density'>

In [21]:

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

Out[21]:

<AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

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]:

```
28078961.370175403
```

In [26]:

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

Out[26]:

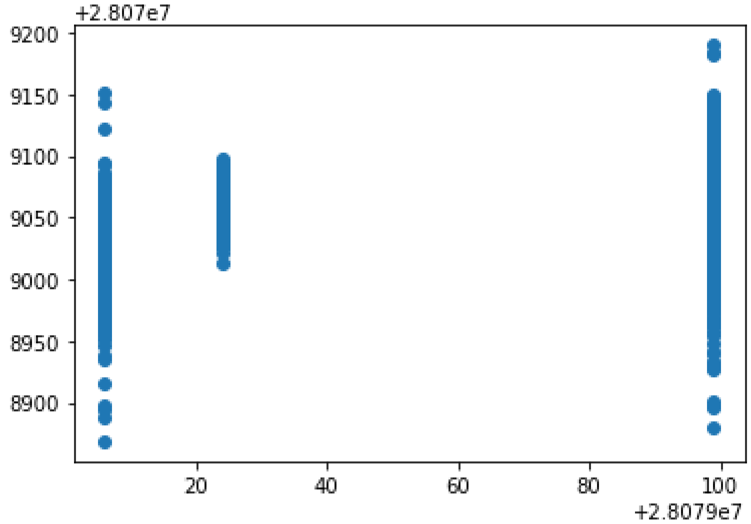
	Co-efficient
BEN	-9.503500
CO	36.721102
EBE	-12.736040
MXY	3.635589
NMHC	82.577619
NO_2	0.113992
NOx	-0.265849
OXY	3.052834
O_3	0.004354
PM10	0.059909
PXY	2.488550
SO_2	0.249597
TCH	61.105297
TOL	-0.599821

In [27]:

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

Out[27]:

<matplotlib.collections.PathCollection at 0x2605f946580>



ACCURACY

In [28]:

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

Out[28]:

0.312467100401984

In [29]:

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

Out[29]:

0.3004675395225376

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]:

```
0.3121714361922914
```

In [33]:

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

Out[33]:

```
0.3002120521351971
```

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]:

```
0.06553804816972864
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.06353386104215053
```

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([-5.5681588 ,  1.48284097, -7.23320507,  2.61112269,  0.88255271,
       -0.05970081, -0.00877309,  1.84002805, -0.02553792,  0.24402907,
        1.30669456,  0.17296177,  1.54189296, -0.81274396])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079049.829165973
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.17690741038500357
```

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)))
```

```
37.027742276968965
1545.3548703499223
39.3110018995945
```

Logistic Regression

In [43]:

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

In [44]:

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

In [45]:

```
feature_matrix.shape
```

Out[45]:

```
(20070, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(20070,)
```

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)
```

```
[28079006]
```

In [52]:

```
logr.classes_
```

Out[52]:

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

In [53]:

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

Out[53]:

```
0.879023418036871
```

In [54]:

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

Out[54]:

```
0.9998967601812779
```

In [55]:

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

Out[55]:

```
array([[9.99896760e-01, 3.21124597e-30, 1.03239819e-04]])
```

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]:

```
0.8634068653280262
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

In [62]:

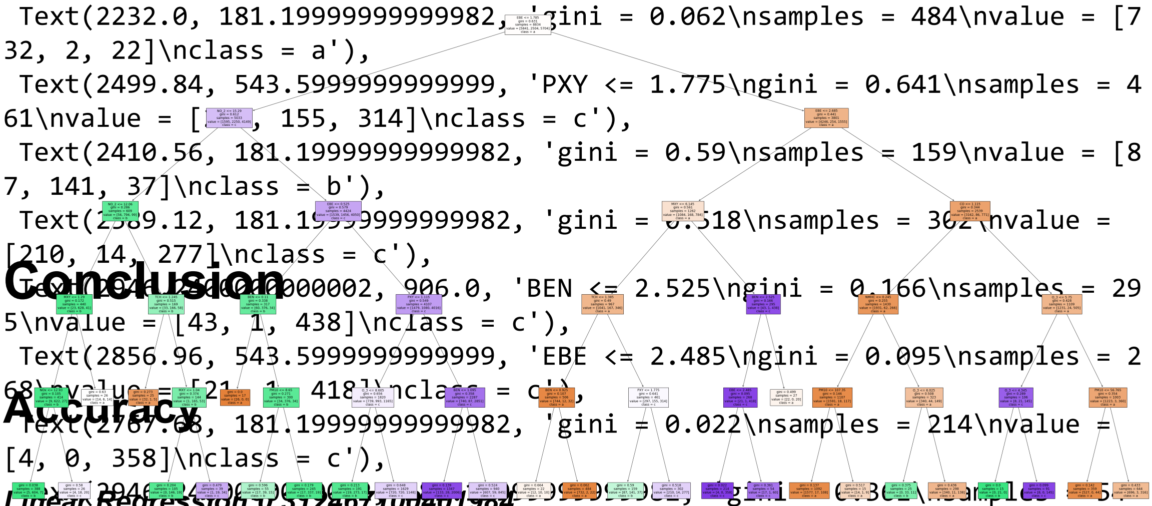
```
from sklearn.tree import plot_tree

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(2031.1200000000001, 1993.2, 'EBE <= 1.785\ngini = 0.631\nsamples = 8
834\nvalue = [5841, 2504, 5704]\nclass = a'),
 Text(870.48, 1630.8000000000002, 'NO_2 <= 15.29\ngini = 0.612\nsamples =
5033\nvalue = [1595, 2250, 4149]\nclass = c'),
 Text(446.4, 1268.4, 'NO_2 <= 12.06\ngini = 0.286\nsamples = 609\nvalue =
[56, 794, 99]\nclass = b'),
 Text(267.84000000000003, 906.0, 'MXY <= 1.29\ngini = 0.172\nsamples = 440
\nvalue = [23, 628, 41]\nclass = b'),
 Text(178.56, 543.5999999999999, 'NOx <= 12.93\ngini = 0.105\nsamples = 41
4\nvalue = [9, 622, 27]\nclass = b'),
 Text(89.28, 181.19999999999982, 'gini = 0.038\nsamples = 388\nvalue = [5,
604, 7]\nclass = b'),
 Text(267.84000000000003, 181.19999999999982, 'gini = 0.58\nsamples = 26\n
value = [4, 18, 20]\nclass = c'),
 Text(357.12, 543.5999999999999, 'gini = 0.63\nsamples = 26\nvalue = [14,
6, 14]\nclass = a'),
 Text(624.96, 906.0, 'TCH <= 1.245\ngini = 0.515\nsamples = 169\nvalue =
[33, 166, 58]\nclass = b'),
 Text(535.6800000000001, 543.5999999999999, 'gini = 0.273\nsamples = 25\nv
alue = [32, 1, 5]\nclass = a'),
 Text(714.24, 543.5999999999999, 'MXY <= 1.04\ngini = 0.374\nsamples = 144
\nvalue = [1, 165, 53]\nclass = b'),
 Text(624.96, 181.19999999999982, 'gini = 0.204\nsamples = 105\nvalue =
[0, 146, 19]\nclass = b'),
 Text(803.52, 181.19999999999982, 'gini = 0.479\nsamples = 39\nvalue = [1,
19, 34]\nclass = c'),
 Text(1294.56, 1268.4, 'EBE <= 0.525\ngini = 0.579\nsamples = 4424\nvalue
= [1539, 1456, 4050]\nclass = c'),
 Text(982.08, 906.0, 'BEN <= 0.11\ngini = 0.338\nsamples = 317\nvalue = [6
0, 376, 34]\nclass = b'),
 Text(892.8, 543.5999999999999, 'gini = 0.0\nsamples = 17\nvalue = [26, 0,
0]\nclass = a'),
 Text(1071.3600000000001, 543.5999999999999, 'PM10 <= 8.65\ngini = 0.271\n
samples = 300\nvalue = [34, 376, 34]\nclass = b'),
 Text(982.08, 181.19999999999982, 'gini = 0.596\nsamples = 55\nvalue = [1
7, 39, 15]\nclass = b'),
 Text(1160.64, 181.19999999999982, 'gini = 0.179\nsamples = 245\nvalue =
[17, 337, 19]\nclass = b'),
 Text(1607.04, 906.0, 'PXY <= 1.115\ngini = 0.549\nsamples = 4107\nvalue =
[1479, 1080, 4016]\nclass = c'),
 Text(1428.48, 543.5999999999999, 'O_3 <= 8.605\ngini = 0.656\nsamples = 1
820\nvalue = [739, 993, 1165]\nclass = c'),
 Text(1339.2, 181.19999999999982, 'gini = 0.213\nsamples = 191\nvalue = [1
9, 273, 17]\nclass = b'),
 Text(1517.76, 181.19999999999982, 'gini = 0.648\nsamples = 1629\nvalue =
[720, 720, 1148]\nclass = c'),
 Text(1785.6, 543.5999999999999, 'BEN <= 1.095\ngini = 0.358\nsamples = 22
87\nvalue = [740, 87, 2851]\nclass = c'),
 Text(1696.32, 181.19999999999982, 'gini = 0.139\nsamples = 1347\nvalue =
[133, 28, 2006]\nclass = c'),
 Text(1874.88, 181.19999999999982, 'gini = 0.524\nsamples = 940\nvalue =
[607, 59, 845]\nclass = c'),
 Text(3191.76, 1630.8000000000002, 'EBE <= 2.685\ngini = 0.441\nsamples =
3801\nvalue = [4246, 254, 1555]\nclass = a'),
 Text(2633.76, 1268.4, 'MXY <= 6.145\ngini = 0.561\nsamples = 1262\nvalue
= [1084, 168, 784]\nclass = a'),
 Text(2321.28, 906.0, 'TCH <= 1.385\ngini = 0.49\nsamples = 967\nvalue =
[1041, 167, 346]\nclass = a'),
 Text(2142.7200000000003, 543.5999999999999, 'BEN <= 0.925\ngini = 0.107\n
samples = 506\nvalue = [744, 12, 32]\nclass = a'),
 Text(2053.44, 181.19999999999982, 'gini = 0.664\nsamples = 22\nvalue = [1
```

```
2, 10, 10]\nclass = a'),
 Text(2232.0, 181.19999999999982, 'gini = 0.062\nsamples = 484\nvalue = [7
32, 2, 22]\nclass = a'),
 Text(2499.84, 543.5999999999999, 'PXY <= 1.775\ngini = 0.641\nsamples = 4
61\nvalue = [2, 155, 314]\nclass = c'),
 Text(2410.56, 181.19999999999982, 'gini = 0.59\nsamples = 159\nvalue = [8
7, 141, 37]\nclass = b'),
 Text(2589.12, 181.19999999999982, 'gini = 0.518\nsamples = 302\nvalue =
[210, 14, 277]\nclass = c'),
 Text(2946.2400000000002, 906.0, 'BEN <= 2.525\ngini = 0.166\nsamples = 29
5\nvalue = [43, 1, 438]\nclass = c'),
 Text(2856.96, 543.5999999999999, 'EBE <= 2.485\ngini = 0.095\nsamples = 2
68\nvalue = [21, 1, 418]\nclass = c'),
 Text(2767.68, 181.19999999999982, 'gini = 0.022\nsamples = 214\nvalue =
[4, 0, 358]\nclass = c'),
 Text(2946.24, 181.19999999999982, 'gini = 0.361\nsamples = 54
\nvalue = [17, 1, 60]\nclass = c'),
 Text(3035.52, 543.5999999999999, 'gini = 0.499\nsamples = 27\nvalue = [2
2, 0, 20]\nclass = a'),
 Text(3749.76, 1268.4, 'CO <= 1.115\ngini = 0.344\nsamples = 2539\nvalue =
[3162, 86, 771]\nclass = a'),
 Text(3392.64, 906.0, 'NMHC <= 0.245\ngini = 0.255\nsamples = 1430\nvalue
= [1931, 62, 266]\nclass = a'),
 Text(3214.08, 543.5999999999999, 'PM10 <= 107.35\ngini = 0.146\nsamples =
1107\nvalue = [1591, 18, 117]\nclass = a'),
 Text(3124.8, 181.19999999999982, 'gini = 0.137\nsamples = 1092\nvalue =
[1577, 17, 108]\nclass = a'),
 Text(3303.36, 181.19999999999982, 'gini = 0.517\nsamples = 15\nvalue = [1
4, 1, 9]\nclass = a'),
 Text(3571.2, 543.5999999999999, 'O_3 <= 6.025\ngini = 0.508\nsamples = 32
3\nvalue = [340, 44, 149]\nclass = a'),
 Text(3481.92, 181.19999999999982, 'gini = 0.375\nsamples = 25\nvalue =
[0, 33, 11]\nclass = b'),
 Text(3660.48, 181.19999999999982, 'gini = 0.436\nsamples = 298\nvalue =
[340, 11, 138]\nclass = a'),
 Text(4106.88, 906.0, 'O_3 <= 5.75\ngini = 0.428\nsamples = 1109\nvalue =
[1231, 24, 505]\nclass = a'),
 Text(3928.32, 543.5999999999999, 'O_3 <= 4.345\ngini = 0.289\nsamples = 1
06\nvalue = [8, 21, 145]\nclass = c'),
 Text(3839.04, 181.19999999999982, 'gini = 0.0\nsamples = 15\nvalue = [0,
21, 0]\nclass = b'),
 Text(4017.6, 181.19999999999982, 'gini = 0.099\nsamples = 91\nvalue = [8,
0, 145]\nclass = c'),
 Text(4285.4400000000005, 543.5999999999999, 'PM10 <= 56.765\ngini = 0.354
\nsamples = 1003\nvalue = [1223, 3, 360]\nclass = a')
```



Conclusion

Accuracy

Linear Regression:0.312467100401984

Ridge Regression:0.3121714361922914 ¶

Lasso Regression:0.06353386104215053

ElasticNet Regression:0.17690741038500357

Logistic Regression:0.8790234178036871

Random Forest:0.8634068653280262

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

In [ ]: