Importing Libraries

In [575]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [576]:
```
df=pd.read_csv(r"C:\Users\user\Desktop\csvs_per_year\csvs_per_year\madrid_2013.
df
```

Out[576]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	
0	2013-11-01 01:00:00	NaN	0.6	NaN	NaN	135.0	74.0	NaN	NaN	NaN	7.0	NaN	NaN	2
1	2013-11-01 01:00:00	1.5	0.5	1.3	NaN	71.0	83.0	2.0	23.0	16.0	12.0	NaN	8.3	2
2	2013-11-01 01:00:00	3.9	NaN	2.8	NaN	49.0	70.0	NaN	NaN	NaN	NaN	NaN	9.0	2
3	2013-11-01 01:00:00	NaN	0.5	NaN	NaN	82.0	87.0	3.0	NaN	NaN	NaN	NaN	NaN	2
4	2013-11-01 01:00:00	NaN	NaN	NaN	NaN	242.0	111.0	2.0	NaN	NaN	12.0	NaN	NaN	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
209875	2013-03-01 00:00:00	NaN	0.4	NaN	NaN	8.0	39.0	52.0	NaN	NaN	NaN	NaN	NaN	2
209876	2013-03-01 00:00:00	NaN	0.4	NaN	NaN	1.0	11.0	NaN	6.0	NaN	2.0	NaN	NaN	2
209877	2013-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	4.0	75.0	NaN	NaN	NaN	NaN	NaN	2
209878	2013-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	11.0	52.0	NaN	NaN	NaN	NaN	NaN	2
209879	2013-03-01 00:00:00	NaN	NaN	NaN	NaN	1.0	10.0	75.0	3.0	NaN	NaN	NaN	NaN	2

209880 rows × 14 columns

Data Cleaning and Data Preprocessing

In [577]:
```
df=df.dropna()
```

In [578]:
```
df.columns
```

Out[578]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM2
5',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [579]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7315 entries, 17286 to 209718
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     7315 non-null   object 
 1   BEN      7315 non-null   float64
 2   CO       7315 non-null   float64
 3   EBE      7315 non-null   float64
 4   NMHC     7315 non-null   float64
 5   NO       7315 non-null   float64
 6   NO_2     7315 non-null   float64
 7   O_3      7315 non-null   float64
 8   PM10     7315 non-null   float64
 9   PM25     7315 non-null   float64
 10  SO_2     7315 non-null   float64
 11  TCH      7315 non-null   float64
 12  TOL      7315 non-null   float64
 13  station  7315 non-null   int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 857.2+ KB
```

In [580]:
```
data=df[['CO' ,'station']]
data
```

Out[580]:

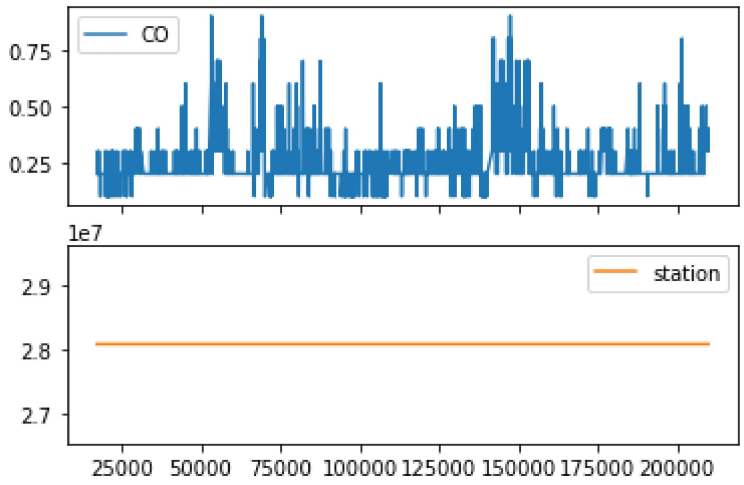
	CO	station
17286	0.2	28079024
17310	0.2	28079024
17334	0.2	28079024
17358	0.2	28079024
17382	0.2	28079024
...	...	...
209622	0.3	28079024
209646	0.4	28079024
209670	0.3	28079024
209694	0.3	28079024
209718	0.3	28079024

7315 rows × 2 columns

Line chart

In [581]:
```
data.plot.line(subplots=True)
```

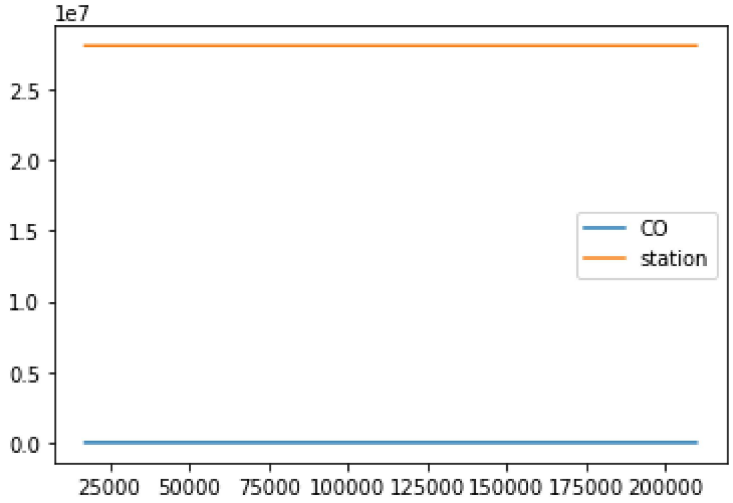
Out[581]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [582]:
```
data.plot.line()
```

Out[582]: <AxesSubplot:>

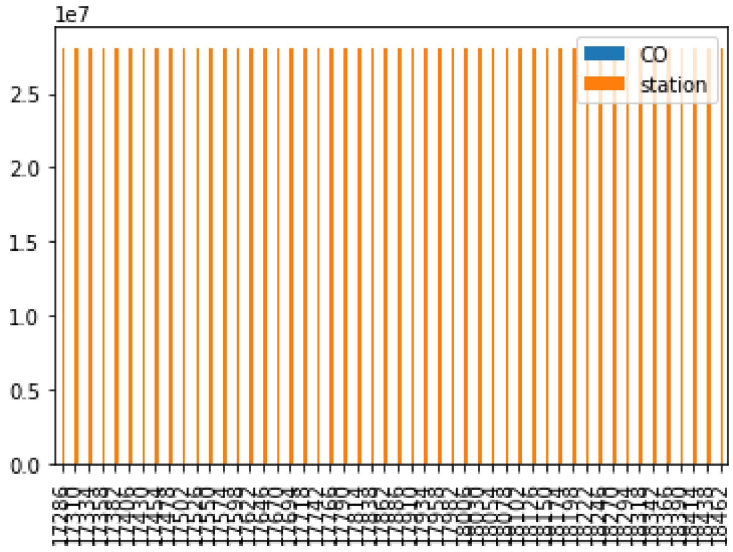


Bar chart

In [583]:
```
b=data[0:50]
```

In [584]:
```
b.plot.bar()
```

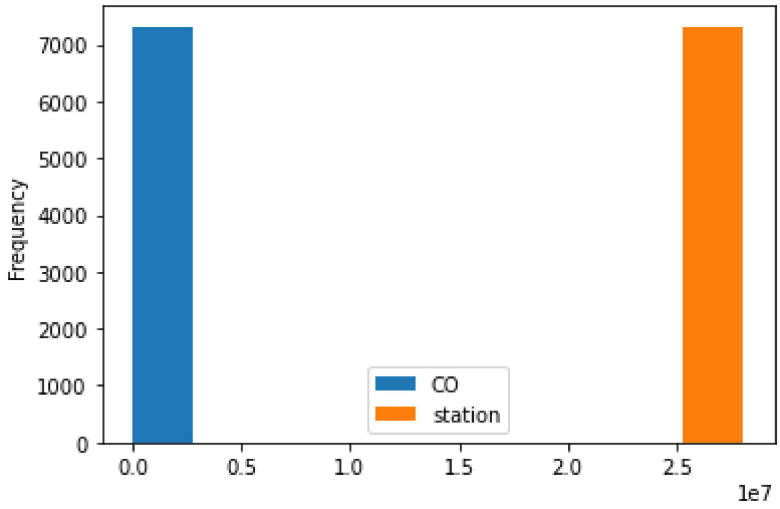
Out[584]: <AxesSubplot:>



Histogram

In [585]:
```
data.plot.hist()
```

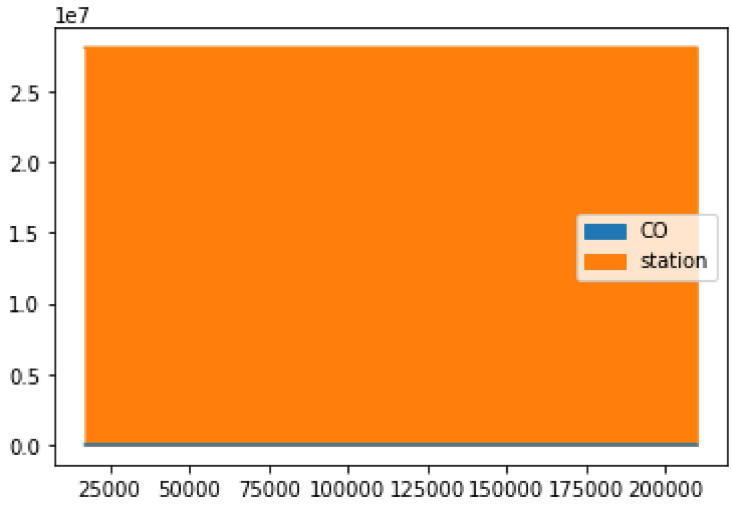
Out[585]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [586]:
```
data.plot.area()
```

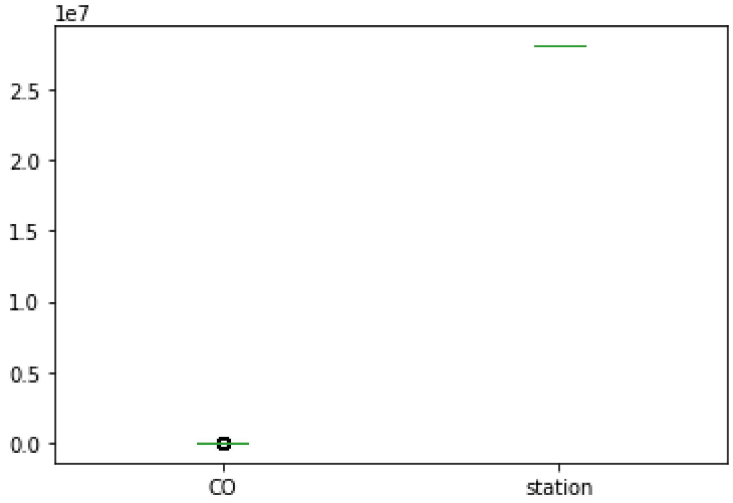
Out[586]: <AxesSubplot:>



Box chart

In [587]:
```
data.plot.box()
```

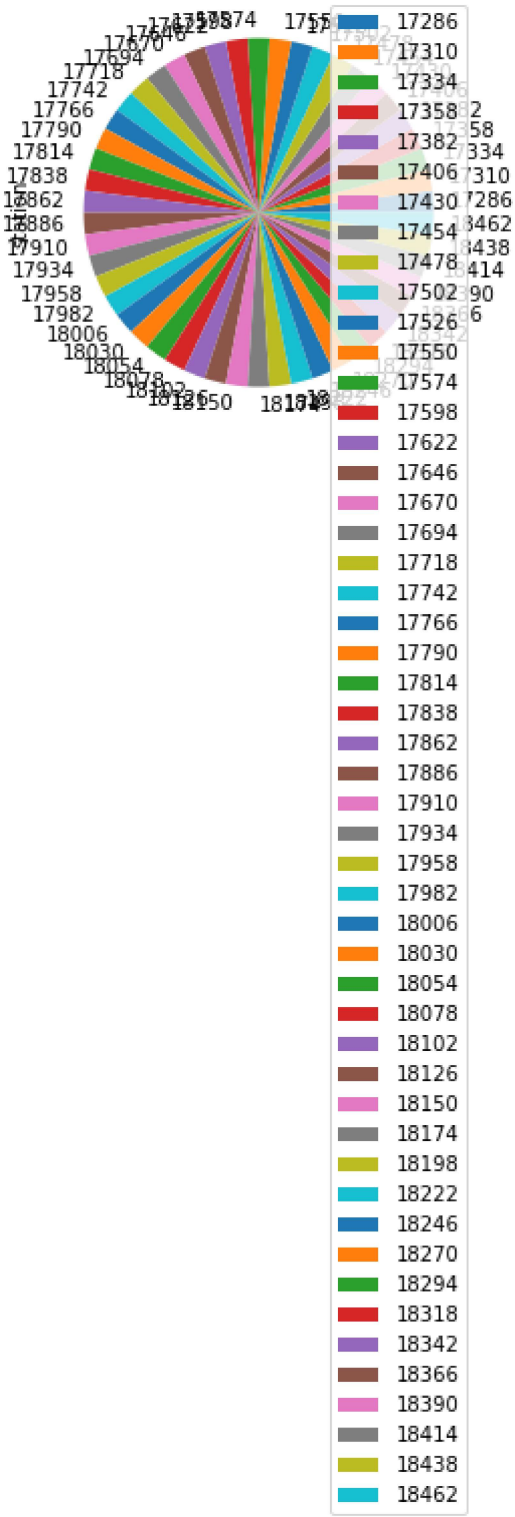
Out[587]: <AxesSubplot:>



Pie chart

In [588]:
```
b.plot.pie(y='station' )
```

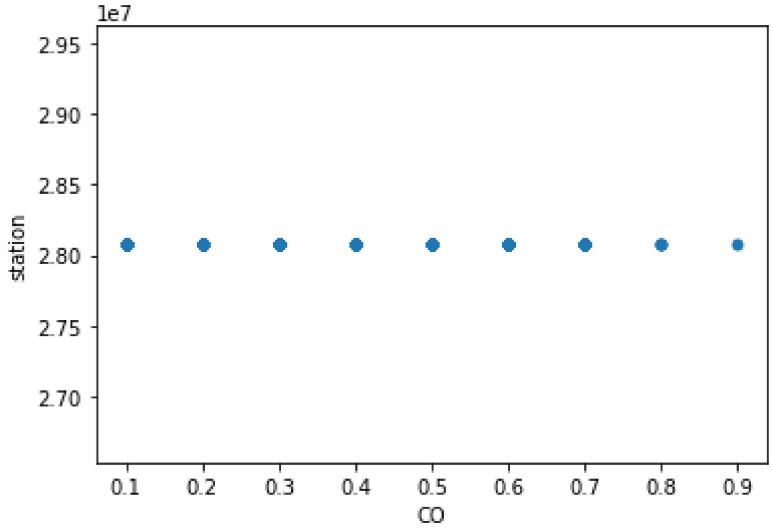
Out[588]: <AxesSubplot:ylabel='station'>



Scatter chart

In [589]:
```
data.plot.scatter(x='CO' ,y='station')
```

Out[589]: <AxesSubplot:xlabel='CO', ylabel='station'>



In [590]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7315 entries, 17286 to 209718
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     7315 non-null   object 
 1   BEN      7315 non-null   float64
 2   CO       7315 non-null   float64
 3   EBE      7315 non-null   float64
 4   NMHC     7315 non-null   float64
 5   NO       7315 non-null   float64
 6   NO_2     7315 non-null   float64
 7   O_3      7315 non-null   float64
 8   PM10     7315 non-null   float64
 9   PM25     7315 non-null   float64
 10  SO_2     7315 non-null   float64
 11  TCH      7315 non-null   float64
 12  TOL      7315 non-null   float64
 13  station  7315 non-null   int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 857.2+ KB
```

In [591]:
```
df.describe()
```

Out[591]:

	BEN	CO	EBE	NMHC	NO	NO_2	O_
count	7315.000000	7315.000000	7315.000000	7315.000000	7315.000000	7315.000000	7315.00000
mean	0.501928	0.236008	0.753247	0.255133	7.486808	19.742584	62.65399
std	0.275264	0.092865	0.386968	0.046754	18.386879	20.984539	35.82244
min	0.100000	0.100000	0.100000	0.170000	1.000000	1.000000	2.00000
25%	0.300000	0.200000	0.500000	0.230000	1.000000	5.000000	38.00000
50%	0.400000	0.200000	0.700000	0.240000	1.000000	12.000000	63.00000
75%	0.600000	0.200000	1.000000	0.270000	3.000000	27.000000	85.00000
max	2.600000	0.900000	3.600000	0.810000	234.000000	124.000000	215.00000

In [592]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-592-9c3e63dc22cd> in <module>
----> 1 df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
      2  'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem_
_(self, key)
   3028             if is_iterator(key):
   3029                 key = list(key)
-> 3030             indexer = self.loc._get_listlike_indexer(key, axis=1, rai
se_missing=True)[1]
   3031
   3032         # take() does not accept boolean indexers

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py in _get_li
stlike_indexer(self, key, axis, raise_missing)
   1264             keyarr, indexer, new_indexer = ax._reindex_non_unique(key
arr)
   1265
-> 1266         self._validate_read_indexer(keyarr, indexer, axis, raise_miss
ing=raise_missing)
   1267         return keyarr, indexer
   1268

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py in _valida
te_read_indexer(self, key, indexer, axis, raise_missing)
   1314             if raise_missing:
   1315                 not_found = list(set(key) - set(ax))
-> 1316                 raise KeyError(f"{not_found} not in index")
   1317
   1318             not_found = key[missing_mask]

KeyError: "['MXY', 'NOx', 'PXY', 'OXY'] not in index"
```

EDA AND VISUALIZATION

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

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

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

TO TRAIN THE MODEL AND MODEL BULDING

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

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

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

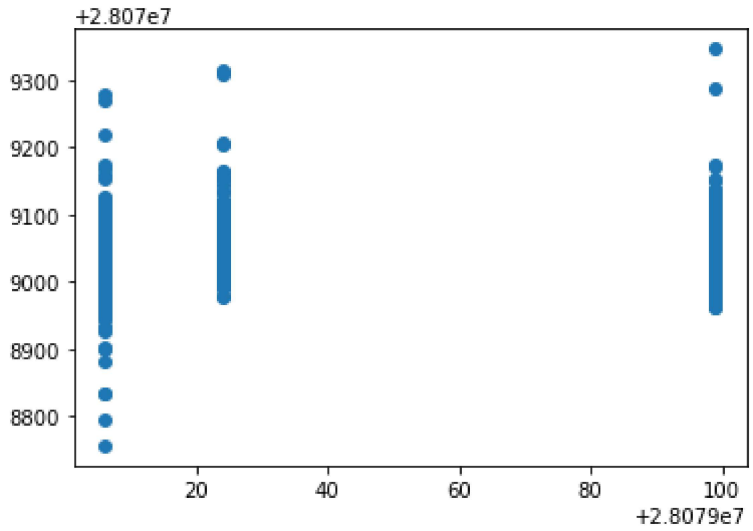
In [593]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[593]:

	Co-efficient
BEN	-37.039973
CO	-30.754126
EBE	7.531596
MXY	-1.633098
NMHC	-18.185915
NO_2	-0.183246
NOx	0.207947
OXY	14.764541
O_3	0.024889
PM10	-0.051958
PXY	2.828355
SO_2	-0.302665
TCH	124.896713
TOL	-1.148805

In [594]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[594]: <matplotlib.collections.PathCollection at 0x1cda5b071c0>



ACCURACY

In [595]:
```
lr.score(x_test,y_test)
```

Out[595]: 0.26828993596392614

In [596]:
```
lr.score(x_train,y_train)
```

Out[596]: 0.2946269921837963

Ridge and Lasso

In [597]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [598]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[598]: Ridge(alpha=10)

Accuracy(Ridge)

In [599]:
```
rr.score(x_test,y_test)
```

Out[599]:
```
0.2700143791476979
```

In [600]:
```
rr.score(x_train,y_train)
```

Out[600]:
```
0.2942743131240403
```

In [601]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[601]:
```
Lasso(alpha=10)
```

In [602]:
```
la.score(x_train,y_train)
```

Out[602]:
```
0.033961539134792496
```

Accuracy(Lasso)

In [603]:
```
la.score(x_test,y_test)
```

Out[603]:
```
0.03863380766863844
```

In [604]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[604]:
```
ElasticNet()
```

In [605]:
```
en.coef_
```

Out[605]:
```
array([-7.07079106, -0.68564137,  0.42393327,  2.12561565, -0.        ,
       -0.24862351,  0.13491165,  1.24030818, -0.13651085,  0.08671355,
        1.92554874, -0.71980373,  1.47185506, -1.99024297])
```

In [606]:
```
en.intercept_
```

Out[606]:
```
28079063.075263444
```

In [607]:
```
prediction=en.predict(x_test)
```

In [608]:
```
en.score(x_test,y_test)
```

Out[608]:
```
0.11209585600575955
```

Evaluation Metrics

In [609]:
```
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)))
```

```
35.84169726560099
1460.280829285163
38.21362099154126
```

Logistic Regression

In [610]:
```
from sklearn.linear_model import LogisticRegression
```

In [611]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-611-60ce984ebae0> in <module>
----> 1 feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx',
'OXY', 'O_3',
      2  'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
      3 target_vector=df[ 'station']

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem_
_(self, key)
   3028             if is_iterator(key):
   3029                 key = list(key)
-> 3030             indexer = self.loc._get_listlike_indexer(key, axis=1, rai
se_missing=True)[1]
   3031
   3032         # take() does not accept boolean indexers

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py in _get_li
stlike_indexer(self, key, axis, raise_missing)
   1264             keyarr, indexer, new_indexer = ax._reindex_non_unique(key
arr)
   1265
-> 1266         self._validate_read_indexer(keyarr, indexer, axis, raise_miss
ing=raise_missing)
   1267         return keyarr, indexer
   1268

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py in _valida
te_read_indexer(self, key, indexer, axis, raise_missing)
   1314             if raise_missing:
   1315                 not_found = list(set(key) - set(ax))
-> 1316                 raise KeyError(f"{not_found} not in index")
   1317
   1318             not_found = key[missing_mask]

KeyError: "['MXY', 'NOx', 'PXY', 'OXY'] not in index"
```

In [612]:
```
feature_matrix.shape
```

Out[612]: (24717, 14)

In [613]:
```
target_vector.shape
```

Out[613]: (24717,)

In [614]:
```
from sklearn.preprocessing import StandardScaler
```

In [615]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [616]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[616]:
```
LogisticRegression(max_iter=10000)
```

In [617]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [618]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [619]:
```
logr.classes_
```

Out[619]:
```
array([28079006, 28079024, 28079099], dtype=int64)
```

In [620]:
```
logr.score(fs,target_vector)
```

Out[620]:
```
0.8951733624630821
```

In [621]:
```
logr.predict_proba(observation)[0][0]
```

Out[621]:
```
5.447205522232353e-13
```

In [622]:
```
logr.predict_proba(observation)
```

Out[622]:
```
array([[5.44720552e-13, 8.28692830e-44, 1.00000000e+00]])
```

Random Forest

In [623]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [624]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[624]:
```
RandomForestClassifier()
```

In [625]:
```
parameters={'max_depth':[1,2,3,4,5],
 'min_samples_leaf':[5,10,15,20,25],
 'n_estimators':[10,20,30,40,50]
}
```

In [626]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="acc
grid_search.fit(x_train,y_train)
```

Out[626]:
```
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 [627]:
```
grid_search.best_score_
```

Out[627]:
```
0.8984449123125864
```

In [628]:
```
rfc_best=grid_search.best_estimator_
```

In [629]:
```
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',
```

```
 Text(2869.7142857142853, 906.0, 'TOL <= 3.055\ngini = 0.29\nsamples = 1602
\nvalue = [2127, 305, 123]\nclass = a'),
 Text(2710.285714285714, 543.5999999999999, 'PXY <= 0.705\ngini = 0.502\nsa
mples = 535\nvalue = [566, 188, 104]\nclass = a'),
 Text(2630.5714285714284, 181.19999999999982, 'gini = 0.034\nsamples = 318
\nvalue = [506, 0, 9]\nclass = a'),
 Text(2790.0, 181.19999999999982, 'gini = 0.592\nsamples = 217\nvalue = [6
0, 188, 95]\nclass = b'),
 Text(3029.142857142857, 543.5999999999999, 'PXY <= 0.865\ngini = 0.149\nsa
mples = 1067\nvalue = [1561, 117, 19]\nclass = a'),
 Text(2949.428571428571, 181.19999999999982, 'gini = 0.013\nsamples = 275\n
value = [444, 2, 1]\nclass = a'),
 Text(3108.8571428571427, 181.19999999999982, 'gini = 0.193\nsamples = 792
\nvalue = [1117, 115, 18]\nclass = a'),
 Text(3826.2857142857138, 1268.4, 'NOx <= 123.05\ngini = 0.616\nsamples = 5
687\nvalue = [2724, 1738, 4553]\nclass = c'),
 Text(3507.428571428571, 906.0, 'NMHC <= 0.285\ngini = 0.502\nsamples = 333
3\nvalue = [460, 1408, 3445]\nclass = c'),
 Text(3347.9999999999995, 543.5999999999999, 'SO_2 <= 6.695\ngini = 0.412\n
samples = 2858\nvalue = [459, 699, 3369]\nclass = c'),
```

Conclusion

Accuracy

Linear Regression:0.2982019580998285

Ridge Regression:0.29819884757718973

Lasso Regression:0.04508362771666252

ElasticNet Regression:0.15378968239891944

Logistic Regression:0.6612921669525443

Random Forest:0.6929265702850609

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

In [ ]: