In [1]:
```python
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
#importing dataset
data1=pd.read_csv(r"C:\Users\user\Downloads\madrid_2005.csv")
data1
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	P
0	2005-11-01 01:00:00	NaN	0.77	NaN	NaN	NaN	57.130001	128.699997	NaN	14.720000	14.91	1
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	30.93	
2	2005-11-01 01:00:00	NaN	0.40	NaN	NaN	NaN	46.119999	53.000000	NaN	30.469999	14.60	
3	2005-11-01 01:00:00	NaN	0.42	NaN	NaN	NaN	37.220001	52.009998	NaN	21.379999	15.16	
4	2005-11-01 01:00:00	NaN	0.57	NaN	NaN	NaN	32.160000	36.680000	NaN	33.410000	5.00	
...	...	...	...	...	...	...	...	...	...	...	...	
236995	2006-01-01 00:00:00	1.08	0.36	1.01	NaN	0.11	21.990000	23.610001	NaN	43.349998	5.00	
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	4.95	
236997	2006-01-01 00:00:00	0.19	NaN	0.26	NaN	0.08	26.730000	30.809999	NaN	43.840000	4.31	
236998	2006-01-01 00:00:00	0.14	NaN	1.00	NaN	0.06	13.770000	17.770000	NaN	NaN	5.00	
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	5.67	

237000 rows × 17 columns

In [3]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 237000 entries, 0 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     237000 non-null  object 
 1   BEN      70370 non-null   float64
 2   CO       217656 non-null  float64
 3   EBE      68955 non-null   float64
 4   MXY      32549 non-null   float64
 5   NMHC     92854 non-null   float64
 6   NO_2     235022 non-null  float64
 7   NOx      235049 non-null  float64
 8   OXY      32555 non-null   float64
 9   O_3      223162 non-null  float64
 10  PM10     232142 non-null  float64
 11  PM25     69407 non-null   float64
 12  PXY      32549 non-null   float64
 13  SO_2     235277 non-null  float64
 14  TCH      93076 non-null   float64
 15  TOL      70255 non-null   float64
 16  station  237000 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 30.7+ MB
```

In [4]:
```
data=data1.head(50000)
```

In [5]:
```
#filling null values
df=data.fillna(0)
df
```

Out[5]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10
0	2005-11-01 01:00:00	0.00	0.77	0.00	0.00	0.00	57.130001	128.699997	0.00	14.720000	14.910000
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	30.930000
2	2005-11-01 01:00:00	0.00	0.40	0.00	0.00	0.00	46.119999	53.000000	0.00	30.469999	14.600000
3	2005-11-01 01:00:00	0.00	0.42	0.00	0.00	0.00	37.220001	52.009998	0.00	21.379999	15.160000
4	2005-11-01 01:00:00	0.00	0.57	0.00	0.00	0.00	32.160000	36.680000	0.00	33.410000	5.000000
...	...	...	...	...	...	...	...	...	...	...	...
49995	2005-01-17 02:00:00	0.00	1.06	0.00	0.00	0.23	54.820000	137.699997	0.00	5.690000	38.389999
49996	2005-01-17 02:00:00	0.89	0.25	0.80	1.49	0.00	46.910000	58.160000	0.88	17.660000	23.250000
49997	2005-01-17 02:00:00	0.00	0.00	0.00	0.00	0.05	69.230003	100.599998	0.00	10.550000	3.890000
49998	2005-01-17 02:00:00	0.00	0.00	0.00	0.00	0.08	42.990002	54.290001	0.00	0.000000	4.530000
49999	2005-01-17 02:00:00	1.15	0.81	1.56	4.42	0.12	64.500000	133.399994	2.53	10.790000	18.250000

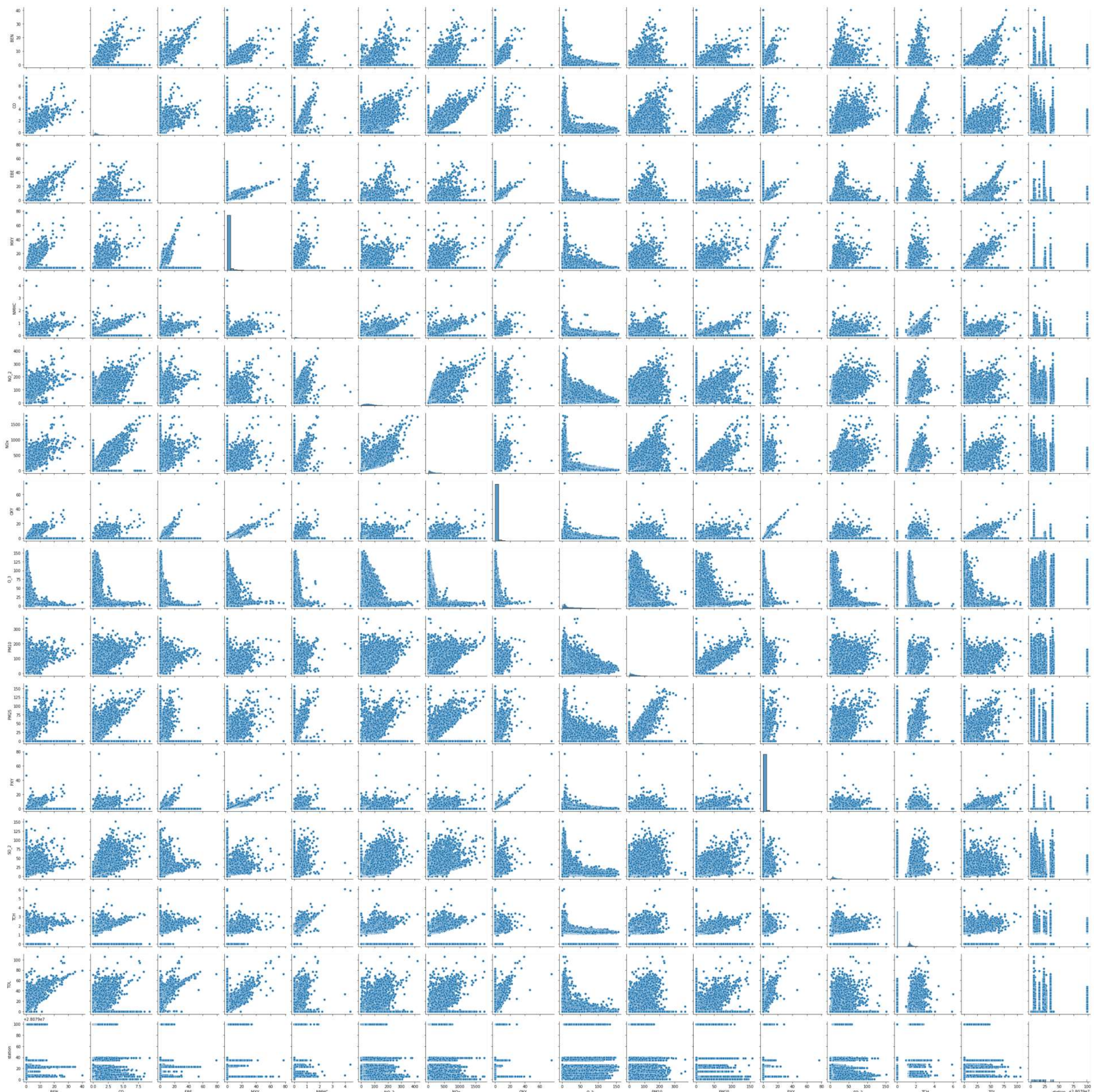
50000 rows × 17 columns

In [6]:
```
df.columns
```

Out[6]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [7]:
```
sns.pairplot(df)
```

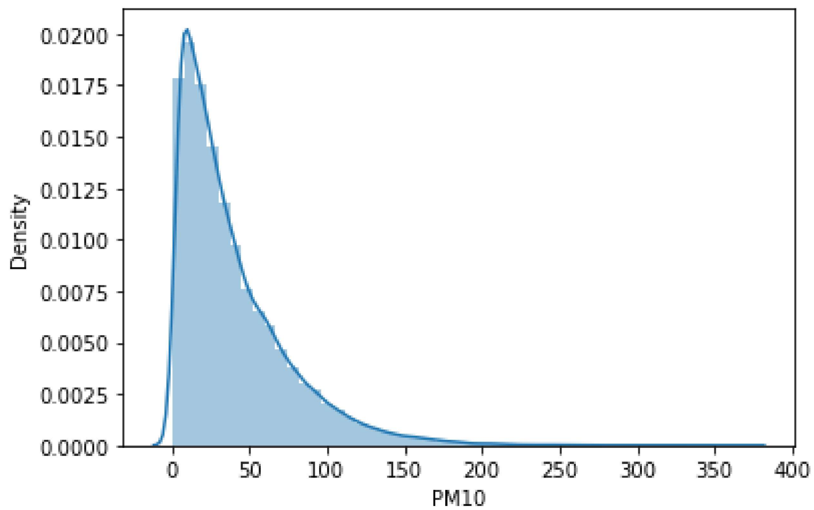
Out[7]: <seaborn.axisgrid.PairGrid at 0x1d226fe7e80>



In [8]:
```
sns.distplot(data["PM10"])
```

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

Out[8]: <AxesSubplot:xlabel='PM10', ylabel='Density'>



MODEL BUILDING

1.Linear Regression

In [9]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

In [10]:
```
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NOx', 'OXY','PM10', 'PXY', 'SO_2', 'T
y=df1[['station']]
```

In [11]:
```
#split the dataset into trainning and test
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)
```

In [12]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

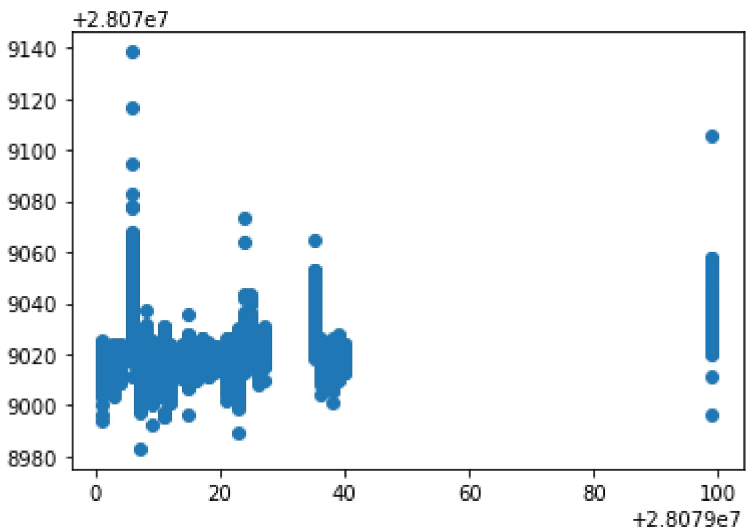
Out[12]:
```
LinearRegression()
```

In [13]:
```
print(lr.intercept_)
```

```
[28079020.74964394]
```

In [14]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[14]:
```
<matplotlib.collections.PathCollection at 0x1d2501af070>
```



In [15]:
```
print(lr.score(x_test,y_test))
```

```
0.095938837309 42909
```

2.Ridge Regression

In [16]:
```
from sklearn.linear_model import Ridge
```

In [17]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[17]:
```
Ridge(alpha=10)
```

In [18]:
```
rr.score(x_test,y_test)
```
Out[18]: 0.09577733230181018

3.Lasso Regression

In [19]:
```
from sklearn.linear_model import Lasso
```

In [20]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```
Out[20]: Lasso(alpha=10)

In [21]:
```
la.score(x_test,y_test)
```
Out[21]: 0.03244028208537508

4.ElasticNet Regression

In [22]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```
Out[22]: ElasticNet()

In [23]:
```
print(en.coef_)
```
```
[ 0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  2.87530861e-02 -1.48060739e-02  1.29506858e+00  2.48593348e-02
  1.05878845e+00 -1.95561452e-01  3.96656622e-01  1.27958520e-01
 -1.26791430e-04]
```

In [24]:
```
print(en.predict(x_test))
```
```
[28079024.36642105 28079024.9799531  28079034.60325968 ...
 28079022.12147274 28079022.75064977 28079022.17176772]
```

In [25]:
```
print(en.score(x_test,y_test))
```
```
0.07962296625413168
```

5.Logistic Regression

In [26]:
```
from sklearn.linear_model import LogisticRegression
```

In [27]:
```
feature_matrix = df1.iloc[:,0:16]
target_vector = df1.iloc[:,-1]
```

In [28]:
```
feature_matrix.shape
```

Out[28]:
```
(50000, 15)
```

In [29]:
```
target_vector.shape
```

Out[29]:
```
(50000,)
```

In [30]:
```
from sklearn.preprocessing import StandardScaler
```

In [31]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [32]:
```
logr = LogisticRegression()
logr.fit(fs,target_vector)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
3: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-
learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on)
  n_iter_i = _check_optimize_result(
```

Out[32]:
```
LogisticRegression()
```

In [33]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]]
```

In [34]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [35]:
```
logr.classes_
```

Out[35]:
```
array([28079001, 28079003, 28079004, 28079006, 28079007, 28079008,
       28079009, 28079011, 28079012, 28079014, 28079015, 28079016,
       28079017, 28079018, 28079019, 28079021, 28079022, 28079023,
       28079024, 28079025, 28079026, 28079027, 28079035, 28079036,
       28079038, 28079039, 28079040, 28079099], dtype=int64)
```

In [36]:
```
logr.score(fs,target_vector)
```

Out[36]:
```
0.85706
```

6.Random Forest

In [37]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3','PM10', 'P
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NOx', 'OXY','PM10', 'PXY', 'SO_2', 'T
y=df['station']
```

In [38]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=45)
```

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[39]:
```
RandomForestClassifier()
```

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

In [41]:
```
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[41]:
```
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 [42]:
```
grid_search.best_score_
```

Out[42]:
```
0.46589923102382547
```

In [43]:
```
rfc_best = grid_search.best_estimator_
```

In [44]:
```python
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,filled=True)
```

Out[44]:
```
[Text(2145.4137931034484, 1993.2, 'TOL <= 0.055\ngini = 0.963\nsamples = 3159
2\nvalue = [1860, 1896, 1774, 1849, 1831, 1882, 1616, 1912, 1921\n1819, 1835,
1880, 1755, 1853, 1812, 1753, 1794, 1830\n1865, 373, 1777, 1865, 1858, 1784,
1866, 1868, 1900\n1927]'),
 Text(1058.2758620689656, 1630.8000000000002, 'NO_2 <= 72.755\ngini = 0.95\ns
amples = 21806\nvalue = [1860, 1896, 1774, 17, 1831, 122, 1616, 1912, 1921\n1
819, 430, 1880, 1755, 1853, 1812, 1753, 1422, 6\n474, 29, 366, 407, 63, 1784,
1866, 1868, 1900, 1]'),
 Text(615.7241379310345, 1268.4, 'SO_2 <= 6.905\ngini = 0.946\nsamples = 1384
9\nvalue = [1093, 1127, 1281, 15, 1255, 79, 346, 1397, 1581\n1157, 193, 1280,
1072, 1314, 875, 1149, 1016, 6, 313\n29, 92, 223, 55, 1193, 947, 1173, 1586,
1]'),
 Text(307.86206896551727, 906.0, 'PM10 <= 0.56\ngini = 0.92\nsamples = 4402\n
value = [505, 525, 436, 8, 477, 63, 123, 5, 713, 330, 50\n43, 317, 30, 540, 5
4, 111, 6, 39, 29, 11, 177\n52, 650, 10, 984, 669, 0]'),
 Text(153.93103448275863, 543.5999999999999, 'CO <= 0.355\ngini = 0.898\nsamp
les = 288\nvalue = [5, 34, 31, 8, 0, 0, 94, 0, 3, 7, 50, 0, 60\n1, 8, 34, 1,
2, 1, 29, 0, 0, 32, 19, 0, 52\n8, 0]'),
 Text(76.96551724137932, 181.19999999999982, 'gini = 0.897\nsamples = 269\nva
```

Results

1.Linear regression : 0.11520628844535274

2.Ridge regression : 0.11519355940988374

3.Lasso regression : 0.03857460583675565

4.Elasticnet regression : 0.08515302878922548

5.Logistic regresssion : 0.9097

6.Random forest regression : 0.5368232977237609

Hence Logistic regression gives high accuracy for the madrid_2004 model.