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"E:\154\C10_air\csvs_per_year\csvs_per_year\madrid_2013.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	2013-11-01 01:00:00	NaN	0.6	NaN	NaN	135.0	74.0	NaN	NaN	NaN	7.0	NaN	NaN	28079004
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	28079008
2	2013-11-01 01:00:00	3.9	NaN	2.8	NaN	49.0	70.0	NaN	NaN	NaN	NaN	NaN	9.0	28079011
3	2013-11-01 01:00:00	NaN	0.5	NaN	NaN	82.0	87.0	3.0	NaN	NaN	NaN	NaN	NaN	28079016
4	2013-11-01 01:00:00	NaN	NaN	NaN	NaN	242.0	111.0	2.0	NaN	NaN	12.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
209875	2013-03-01 00:00:00	NaN	0.4	NaN	NaN	8.0	39.0	52.0	NaN	NaN	NaN	NaN	NaN	28079056
209876	2013-03-01 00:00:00	NaN	0.4	NaN	NaN	1.0	11.0	NaN	6.0	NaN	2.0	NaN	NaN	28079057
209877	2013-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	4.0	75.0	NaN	NaN	NaN	NaN	NaN	28079058
209878	2013-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	11.0	52.0	NaN	NaN	NaN	NaN	NaN	28079059
209879	2013-03-01 00:00:00	NaN	NaN	NaN	NaN	1.0	10.0	75.0	3.0	NaN	NaN	NaN	NaN	28079060

209880 rows × 14 columns

Data Cleaning and Data Preprocessing

In [3]:
```
df=df.fillna(1)
```

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

Out[4]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209880 entries, 0 to 209879
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     209880 non-null  object 
 1   BEN      209880 non-null  float64
 2   CO       209880 non-null  float64
 3   EBE      209880 non-null  float64
 4   NMHC     209880 non-null  float64
 5   NO       209880 non-null  float64
 6   NO_2     209880 non-null  float64
 7   O_3      209880 non-null  float64
 8   PM10     209880 non-null  float64
 9   PM25     209880 non-null  float64
 10  SO_2     209880 non-null  float64
 11  TCH      209880 non-null  float64
 12  TOL      209880 non-null  float64
 13  station  209880 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 22.4+ MB
```

In [6]:
```
data=df[['CO' ,'station']]
data
```

Out[6]:

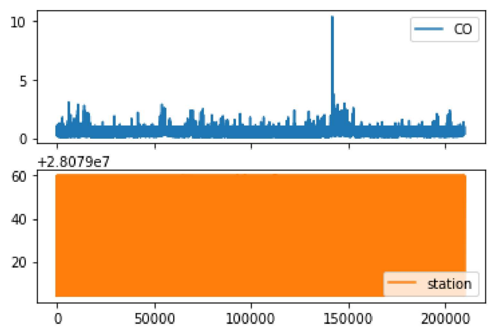
	CO	station
0	0.6	28079004
1	0.5	28079008
2	1.0	28079011
3	0.5	28079016
4	1.0	28079017
...	...	...
209875	0.4	28079056
209876	0.4	28079057
209877	1.0	28079058
209878	1.0	28079059
209879	1.0	28079060

209880 rows × 2 columns

Line chart

In [7]:
```
data.plot.line(subplots=True)
```

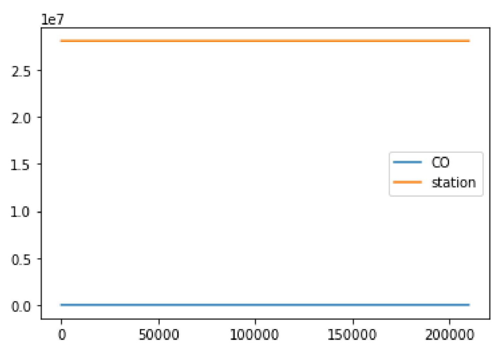
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

Out[8]:
```
<AxesSubplot:>
```

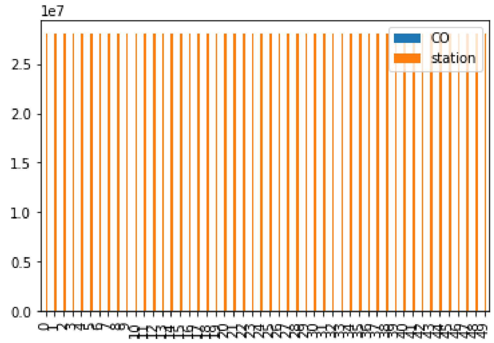


Bar chart

In [9]:
```
b=data[0:50]
```

In [10]:
```
b.plot.bar()
```

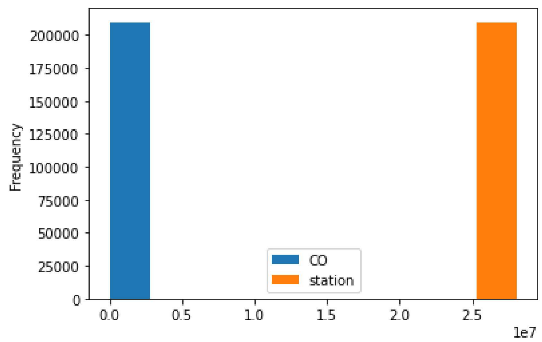
Out[10]: <AxesSubplot:>



Histogram

In [11]:
```
data.plot.hist()
```

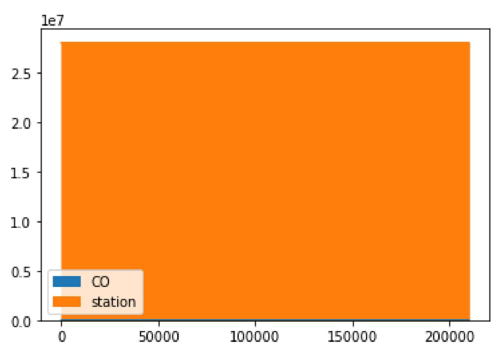
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

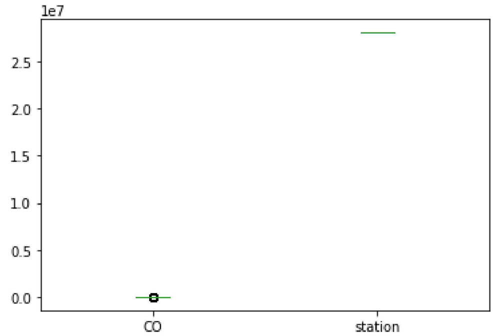
Out[12]: <AxesSubplot:>



Box chart

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

Out[13]: <AxesSubplot:>



Pie chart

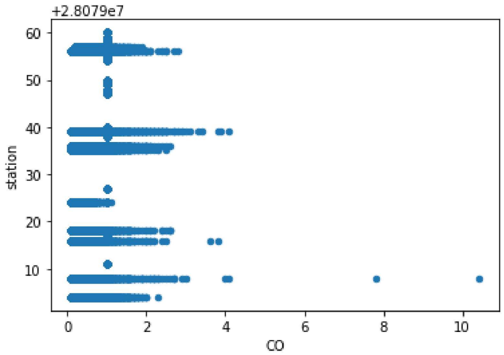
In [14]:
```
b.plot.pie(y='station' )
```

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'>
RangeIndex: 209880 entries, 0 to 209879
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   date     209880 non-null  object
 1   BEN      209880 non-null  float64
 2   CO       209880 non-null  float64
 3   EBE      209880 non-null  float64
 4   NMHC     209880 non-null  float64
 5   NO       209880 non-null  float64
 6   NO_2     209880 non-null  float64
 7   O_3      209880 non-null  float64
 8   PM10     209880 non-null  float64
 9   PM25     209880 non-null  float64
 10  SO_2     209880 non-null  float64
 11  TCH      209880 non-null  float64
 12  TOL      209880 non-null  float64
 13  station  209880 non-null  int64
dtypes: float64(13), int64(1), object(1)
```

In [17]:
```
df.describe()
```

Out[17]:

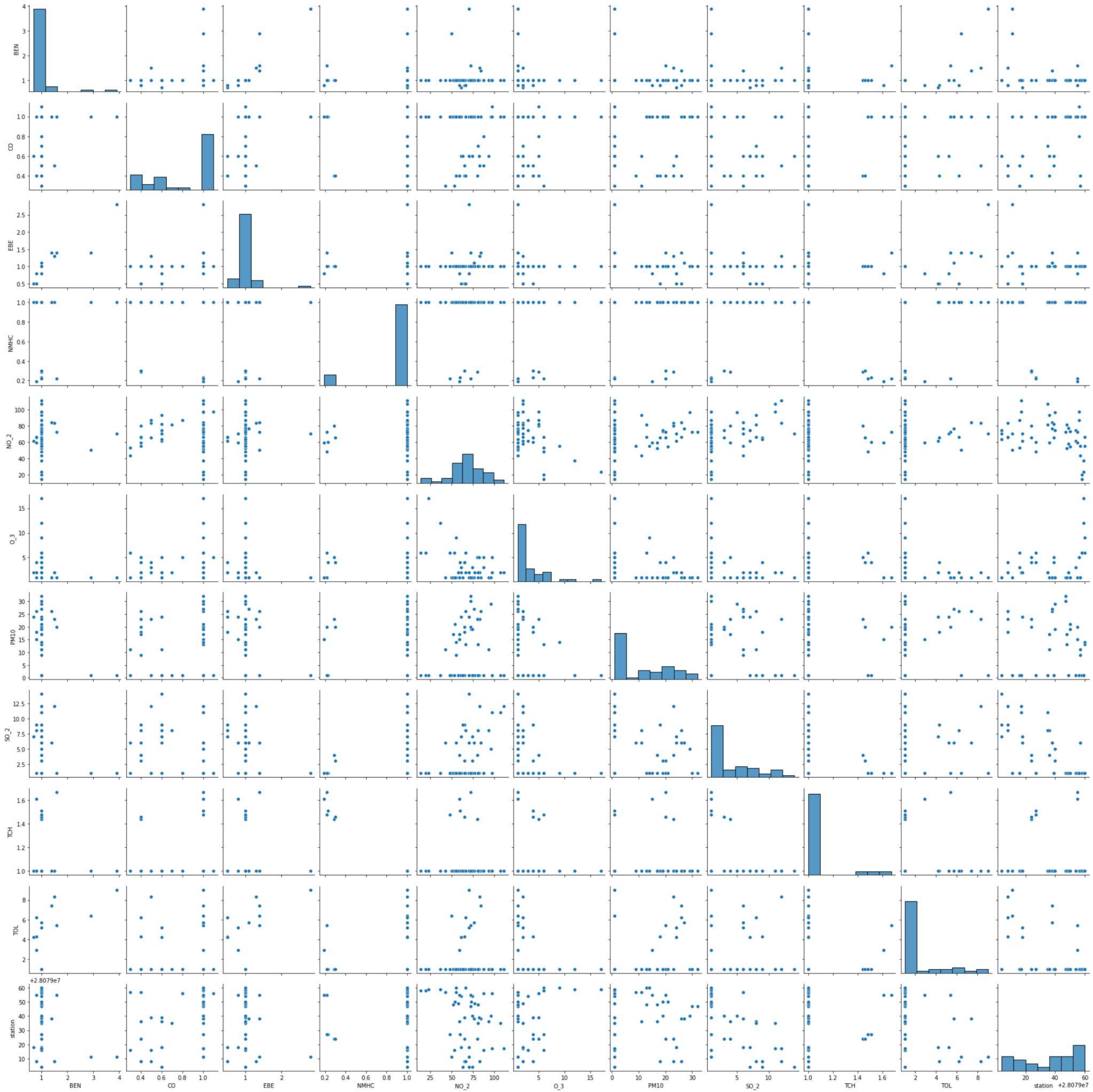
	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_
count	209880.000000	209880.000000	209880.000000	209880.000000	209880.000000	209880.000000	209880.000000	209880.000000	209880.000000	209880.00000
mean	0.931014	0.721695	0.954744	0.900223	20.101401	34.586402	29.461235	9.636635	3.213098	2.41724
std	0.430684	0.361528	0.301074	0.267139	44.319112	27.866588	35.362880	13.492716	5.044685	3.09325
min	0.100000	0.100000	0.100000	0.040000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000
25%	1.000000	0.300000	1.000000	1.000000	2.000000	14.000000	1.000000	1.000000	1.000000	1.00000
50%	1.000000	1.000000	1.000000	1.000000	5.000000	27.000000	8.000000	1.000000	1.000000	1.00000
75%	1.000000	1.000000	1.000000	1.000000	17.000000	48.000000	54.000000	14.000000	1.000000	3.00000
max	12.100000	10.400000	11.800000	1.000000	1081.000000	388.000000	226.000000	232.000000	63.000000	89.00000

In [18]:
```
df1=df[['BEN', 'CO', 'EBE','NMHC', 'NO_2','O_3',
       'PM10', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [19]:
```
sns.pairplot(df1[0:50])
```

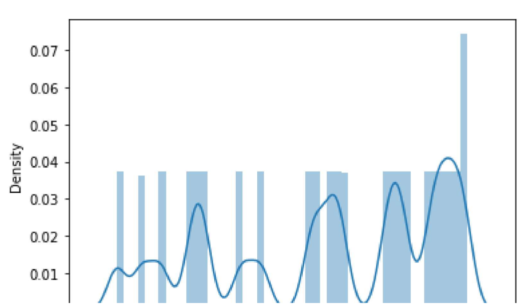
Out[19]: <seaborn.axisgrid.PairGrid at 0x20305a5d790>



In [20]:
```
sns.distplot(df1['station'])
```

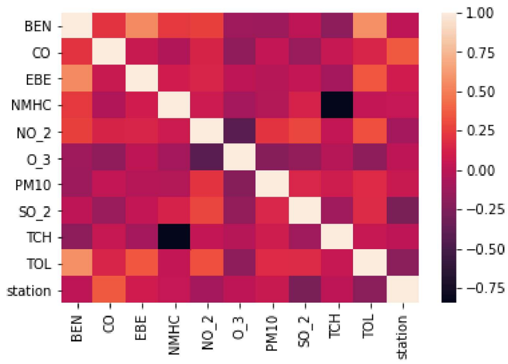
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function 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 BULDING

In [22]:
```
x=df[['BEN', 'CO', 'EBE','NMHC', 'NO_2','O_3',
       'PM10', 'SO_2', 'TCH', 'TOL', 'station']]
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]: -1.30385160446167e-07

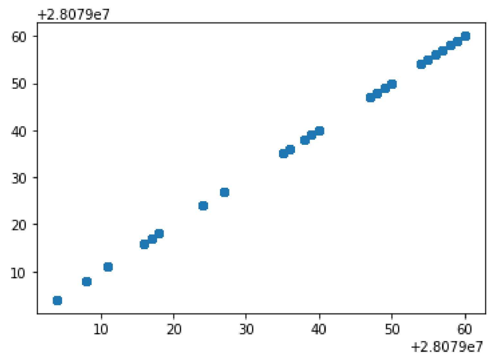
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	8.337672e-15
CO	-8.574377e-14
EBE	-2.330254e-14
NMHC	-1.339406e-13
NO_2	3.252273e-16
O_3	-6.054380e-17
PM10	-8.865941e-16
SO_2	3.914959e-15
TCH	-1.782208e-13
TOL	9.249893e-15
station	1.000000e+00

In [27]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[27]: <matplotlib.collections.PathCollection at 0x20313eb3fd0>



ACCURACY

In [28]:
```
lr.score(x_test,y_test)
```

Out[28]: 1.0

In [29]:
```
lr.score(x_train,y_train)
```

Out[29]: 1.0

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]: 0.9999999999999309

In [33]:
```
rr.score(x_train,y_train)
```

Out[33]: 0.9999999999999309

In [34]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[34]: Lasso(alpha=10)

Accuracy(Lasso)

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

Out[35]: 0.9989554186948196

In [36]:
```
la.score(x_test,y_test)
```

Out[36]: 0.9989553902378367

ElasticNet

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([ 0.        ,  0.        ,  0.        ,  0.        , -0.        ,
        0.        ,  0.        , -0.        ,  0.        , -0.        ,
        0.99677322])
```

In [39]:
```
en.intercept_
```

Out[39]:
```
90604.98503045738
```

In [40]:
```
prediction=en.predict(x_test)
```

In [41]:
```
en.score(x_test,y_test)
```

Out[41]:
```
0.9999895875824748
```

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)))
```

```
0.0487597770780679
0.003228860993378997
0.05682306744077617
```

Logistic Regression

In [43]:
```
from sklearn.linear_model import LogisticRegression
```

In [44]:
```
feature_matrix=df[['BEN', 'CO', 'EBE','NMHC', 'NO_2','O_3',
       'PM10', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [45]:
```
feature_matrix.shape
```

Out[45]:
```
(209880, 10)
```

In [46]:
```
target_vector.shape
```

Out[46]:
```
(209880,)
```

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 [52]:
```
observation=[[1,2,3,4,5,6,7,8,9,10]]
```

In [53]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [54]:
```
logr.classes_
```

Out[54]:
```
array([28079004, 28079008, 28079011, 28079016, 28079017, 28079018,
       28079024, 28079027, 28079035, 28079036, 28079038, 28079039,
       28079040, 28079047, 28079048, 28079049, 28079050, 28079054,
       28079055, 28079056, 28079057, 28079058, 28079059, 28079060],
      dtype=int64)
```

In [55]:
```
logr.score(fs,target_vector)
```
Out[55]:
```
0.6612921669525443
```

In [56]:
```
logr.predict_proba(observation)[0][0]
```
Out[56]:
```
9.49253547859177e-217
```

In [57]:
```
logr.predict_proba(observation)
```
Out[57]:
```
array([[9.49253548e-217, 6.03969072e-001, 1.69773000e-169,
        1.44179094e-134, 1.71060740e-074, 3.96021369e-001,
        9.55808997e-006, 5.22717178e-089, 5.48319507e-081,
        1.32436170e-079, 1.07294134e-076, 3.50636612e-129,
        1.69529056e-079, 3.82520459e-158, 4.22872970e-161,
        3.57928159e-187, 2.10845766e-164, 8.33937392e-188,
        1.12752042e-082, 7.42692411e-129, 7.66872499e-080,
        6.30044443e-191, 4.32093567e-191, 3.26054498e-071]])
```

Random Forest

In [58]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [59]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```
Out[59]:
```
RandomForestClassifier()
```

In [60]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [61]:
```
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[61]:
```
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 [62]:
```
grid_search.best_score_
```
Out[62]:
```
0.9546475537041574
```

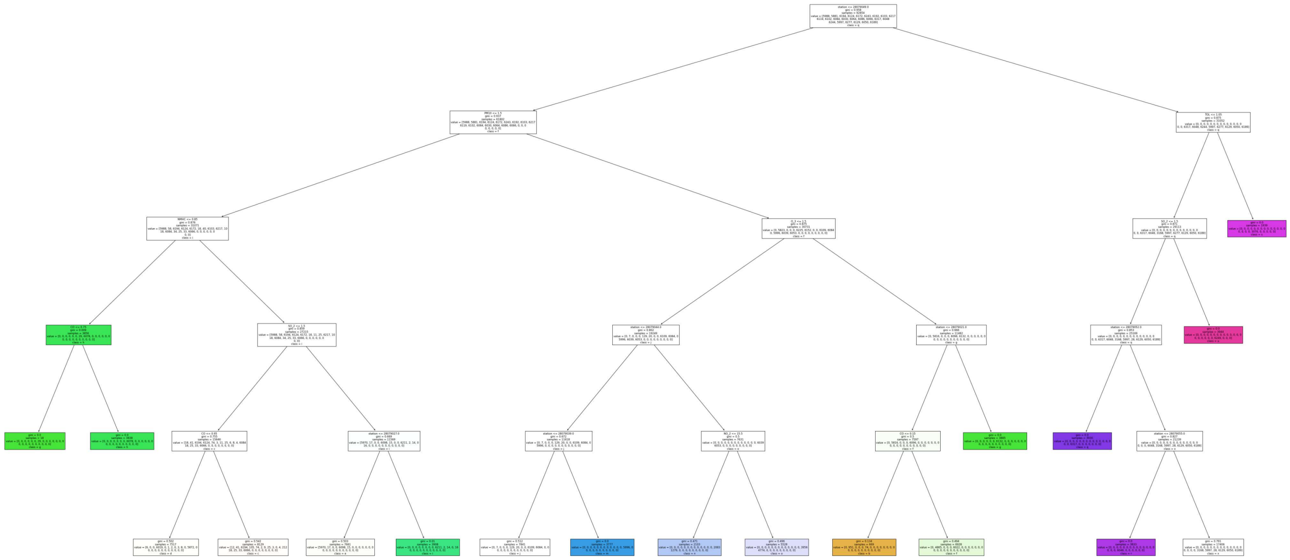
In [63]:
```
rfc_best=grid_search.best_estimator_
```

In [65]:

```
s=x.columns,class_names=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x'],filled=
```

Out[65]:

```
[Text(2938.8, 1993.2, 'station <= 28079049.0\ngini = 0.958\nsamples = 92854\nvalue = [5988, 5881, 6194, 6124, 6172, 6243, 6192, 6103, 6217\n6119, 6102, 6084, 6030, 6064, 6086, 6066, 6317, 6048\n6244, 5997, 6277, 6129, 6050, 6189]\nclass = q'),
 Text(1711.2, 1630.8000000000002, 'PM10 <= 1.5\ngini = 0.937\nsamples = 61802\nvalue = [5988, 5881, 6194, 6124, 6172, 6243, 6192, 6103, 6217\n6119, 6102, 6084, 6030, 6064, 6086, 6066, 0, 0, 0\n0, 0, 0, 0, 0, 0]\nclass = f'),
 Text(669.6, 1268.4, 'NMHC <= 0.85\ngini = 0.876\nsamples = 31071\nvalue = [5988, 58, 6194, 6124, 6172, 18, 40, 6103, 6217, 10\n18, 6084, 34, 25, 33, 6066, 0, 0, 0, 0, 0, 0\n0, 0]\nclass = i'),
 Text(297.6, 906.0, 'CO <= 0.75\ngini = 0.009\nsamples = 3856\nvalue = [0, 0, 0, 0, 0, 0, 29, 6078, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = h'),
 Text(148.8, 543.5999999999999, 'gini = 0.0\nsamples = 18\nvalue = [0, 0, 0, 0, 0, 0, 29, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = g'),
 Text(446.40000000000003, 543.5999999999999, 'gini = 0.0\nsamples = 3838\nvalue = [0, 0, 0, 0, 0, 0, 0, 6078, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = h'),
 Text(1041.6000000000001, 906.0, 'SO_2 <= 1.5\ngini = 0.859\nsamples = 27215\nvalue = [5988, 58, 6194, 6124, 6172, 18, 11, 25, 6217, 10\n18, 6084, 34, 25, 33, 6066, 0, 0, 0, 0, 0, 0\n0, 0]\nclass = i'),
 Text(744.0, 543.5999999999999, 'CO <= 0.95\ngini = 0.755\nsamples = 15646\nvalue = [18, 41, 6194, 6124, 74, 3, 11, 25, 6, 8, 4, 6084\n18, 25, 33, 6066, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = c'),
 Text(595.2, 181.19999999999982, 'gini = 0.502\nsamples = 7517\nvalue = [6, 0, 0, 6019, 0, 1, 2, 0, 3, 8, 0, 5872, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = d'),
 Text(892.8000000000001, 181.19999999999982, 'gini = 0.542\nsamples = 8129\nvalue = [12, 41, 6194, 105, 74, 2, 9, 25, 3, 0, 4, 212\n18, 25, 33, 6066, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = c'),
 Text(1339.2, 543.5999999999999, 'station <= 28079027.0\ngini = 0.669\nsamples = 11569\nvalue = [5970, 17, 0, 0, 6098, 15, 0, 0, 6211, 2, 14, 0\n16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = i'),
 Text(1190.4, 181.19999999999982, 'gini = 0.503\nsamples = 7661\nvalue = [5970, 17, 0, 0, 6098, 15, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = e'),
 Text(1488.0, 181.19999999999982, 'gini = 0.01\nsamples = 3908\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 6211, 2, 14, 0, 16\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = i'),
 Text(2752.8, 1268.4, 'O_3 <= 1.5\ngini = 0.875\nsamples = 30731\nvalue = [0, 5823, 0, 0, 0, 6225, 6152, 0, 0, 6109, 6084\n0, 5996, 6039, 6053, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = f'),
 Text(2232.0, 906.0, 'station <= 28079044.0\ngini = 0.802\nsamples = 19249\nvalue = [0, 7, 0, 0, 0, 129, 20, 0, 0, 6109, 6084, 0\n5996, 6039, 6053, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = j'),
 Text(1934.4, 543.5999999999999, 'station <= 28079039.0\ngini = 0.672\nsamples = 11618\nvalue = [0, 7, 0, 0, 0, 129, 20, 0, 0, 6109, 6084, 0\n5996, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = j'),
 Text(1785.6000000000001, 181.19999999999982, 'gini = 0.512\nsamples = 7841\nvalue = [0, 7, 0, 0, 0, 129, 20, 0, 0, 6109, 6084, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = j'),
 Text(2083.2000000000003, 181.19999999999982, 'gini = 0.0\nsamples = 3777\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5996, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = m'),
 Text(2529.6000000000004, 543.5999999999999, 'NO_2 <= 15.5\ngini = 0.5\nsamples = 7631\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6039\n6053, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = o'),
 Text(2380.8, 181.19999999999982, 'gini = 0.471\nsamples = 2103\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2083\n1279, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = n'),
 Text(2678.4, 181.19999999999982, 'gini = 0.496\nsamples = 5528\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3956\n4774, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = o'),
 Text(3273.6000000000004, 906.0, 'station <= 28079021.0\ngini = 0.666\nsamples = 11482\nvalue = [0, 5816, 0, 0, 0, 6096, 6132, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = g'),
 Text(3124.8, 543.5999999999999, 'CO <= 0.15\ngini = 0.5\nsamples = 7597\nvalue = [0, 5816, 0, 0, 0, 6096, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = f'),
 Text(2976.0, 181.19999999999982, 'gini = 0.134\nsamples = 669\nvalue = [0, 951, 0, 0, 0, 74, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = b'),
 Text(3273.6000000000004, 181.19999999999982, 'gini = 0.494\nsamples = 6928\nvalue = [0, 4865, 0, 0, 0, 6022, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = f'),
 Text(3422.4, 543.5999999999999, 'gini = 0.0\nsamples = 3885\nvalue = [0, 0, 0, 0, 0, 0, 6132, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = g'),
 Text(4166.400000000001, 1630.8000000000002, 'TOL <= 1.05\ngini = 0.875\nsamples = 31052\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 6317, 6048, 6244, 5997, 6277, 6129, 6050, 6189]\nclass = q'),
 Text(4017.6000000000004, 1268.4, 'SO_2 <= 1.5\ngini = 0.871\nsamples = 29113\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 6317, 6048, 3168, 5997, 6277, 6129, 6050, 6189]\nclass = q'),
 Text(3868.8, 906.0, 'station <= 28079052.0\ngini = 0.853\nsamples = 25169\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 6317, 6048, 3168, 5997, 28, 6129, 6050, 6189]\nclass = q'),
 Text(3720.0000000000005, 543.5999999999999, 'gini = 0.0\nsamples = 3930\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 6317, 0, 0, 0, 0, 0, 0, 0]\nclass = q'),
 Text(4017.6000000000004, 543.5999999999999, 'station <= 28079055.0\ngini = 0.827\nsamples = 21239\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 6048, 3168, 5997, 28, 6129, 6050, 6189]\nclass = x'),
 Text(3868.8, 181.19999999999982, 'gini = 0.0\nsamples = 3833\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 6048, 0, 0, 0, 0, 0, 0]\nclass = r'),
 Text(4166.400000000001, 181.19999999999982, 'gini = 0.791\nsamples = 17406\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 3168, 5997, 28, 6129, 6050, 6189]\nclass = x'),
 Text(4166.400000000001, 906.0, 'gini = 0.0\nsamples = 3944\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 6249, 0, 0, 0]\nclass = u'),
 Text(4315.200000000001, 1268.4, 'gini = 0.0\nsamples = 1939\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 3076, 0, 0, 0, 0, 0]\nclass = s')]
```



Accuracy

Linear Regression

In [66]:
```
lr.score(x_train,y_train)
```
Out[66]: 1.0

Ridge Regression

In [67]:
```
rr.score(x_train,y_train)
```
Out[67]: 0.9999999999999309

Lasso Regression

In [68]:
```
la.score(x_test,y_test)
```
Out[68]: 0.9989553902378367

ElasticNet Regression

In [69]:
```
en.score(x_test,y_test)
```
Out[69]: 0.9998895875824748

Logistic Regression

In [70]:
```
logr.score(fs,target_vector)
```
Out[70]: 0.6612921669525443

Random Forest ¶

In [71]:
```
grid_search.best_score_
```
Out[71]: 0.9546475537041574

Conclusion

Random Forest is suitable for this dataset