

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

```
In [2]: df=pd.read_csv("madrid_2005.csv")
```

```
In [3]: df.head()
```

```
Out[3]:
```

	date	BEN	CO	EBE	MXV	NMHC	NO_2	NOx	OXY	O_3	PM10	PM2.5
0	2005-11-01 01:00:00	NaN	0.77	NaN	NaN	NaN	57.130001	128.699997	NaN	14.720000	14.91	10.6
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	NaN
2	2005-11-01 01:00:00	NaN	0.40	NaN	NaN	NaN	46.119999	53.000000	NaN	30.469999	14.60	NaN
3	2005-11-01 01:00:00	NaN	0.42	NaN	NaN	NaN	37.220001	52.009998	NaN	21.379999	15.16	NaN
4	2005-11-01 01:00:00	NaN	0.57	NaN	NaN	NaN	32.160000	36.680000	NaN	33.410000	5.00	NaN

```
In [4]: df=df.dropna()
```

```
In [5]: df.columns
```

```
Out[5]: Index(['date', 'BEN', 'CO', 'EBE', 'MXV', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
               'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
              dtype='object')
```

In [6]: 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 [7]: data=df[['CO', 'station']]
data

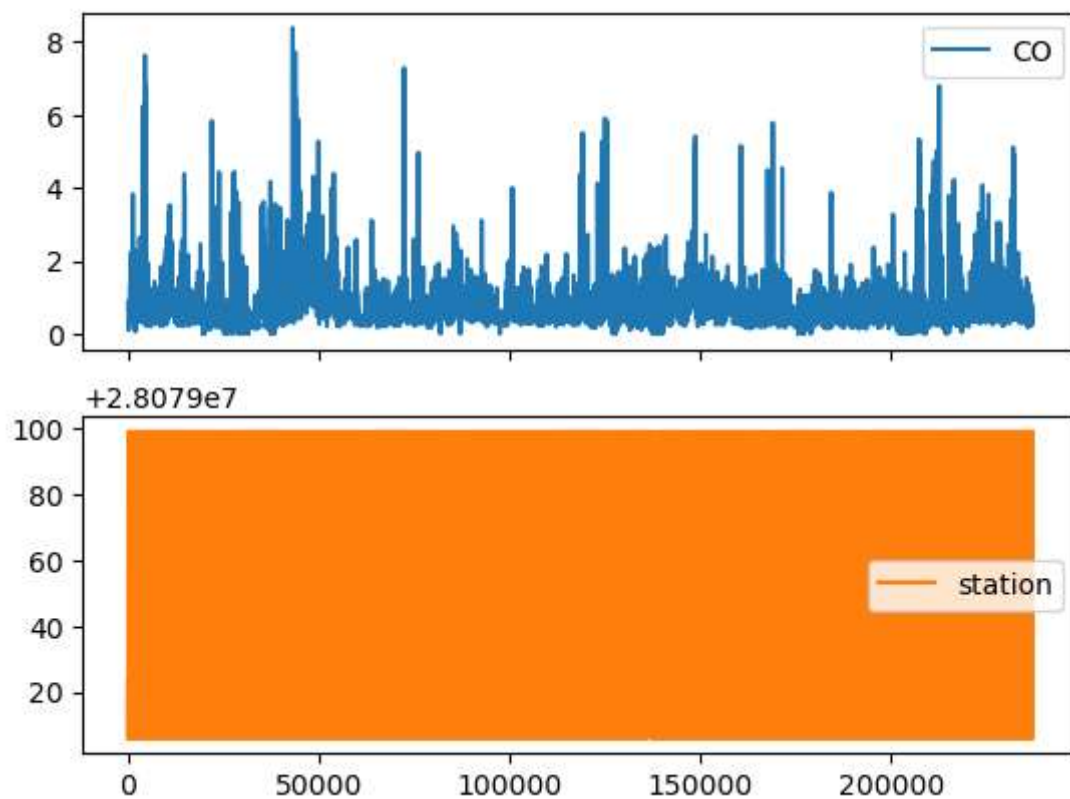
Out[7]:

	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

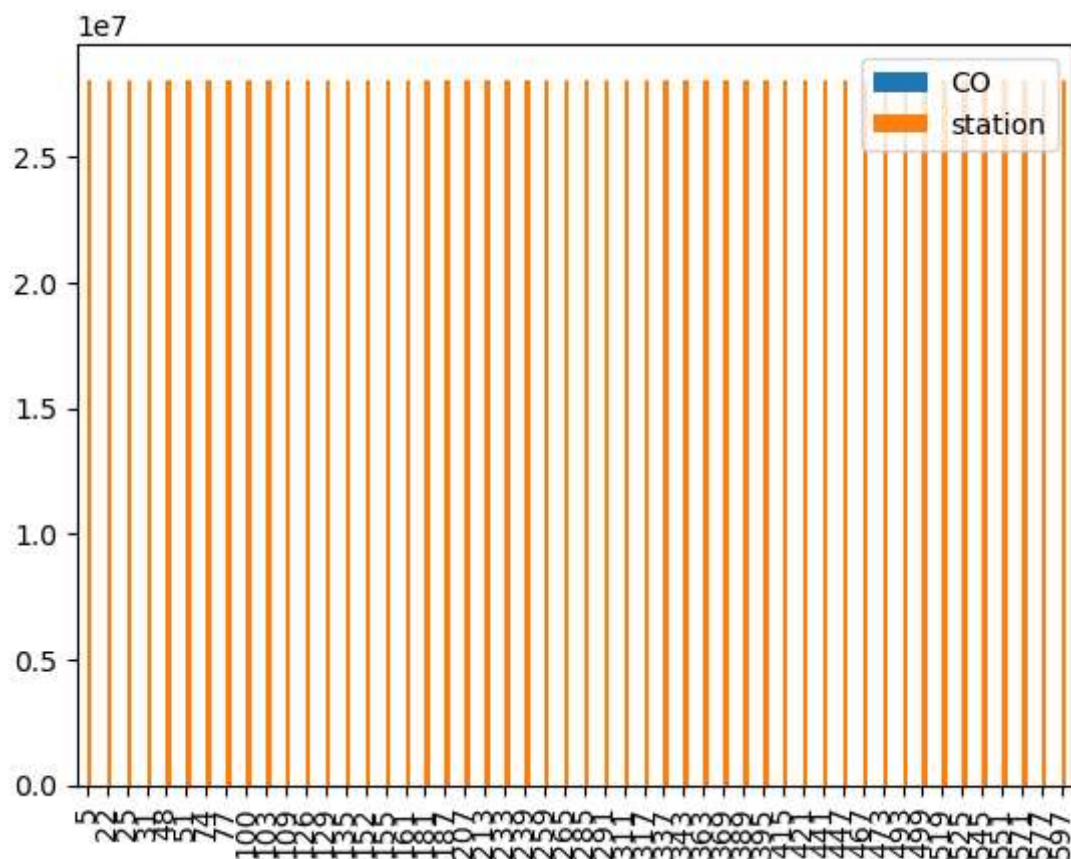
```
In [8]: data.plot.line(subplots=True)
```

```
Out[8]: array([<Axes: >, <Axes: >], dtype=object)
```



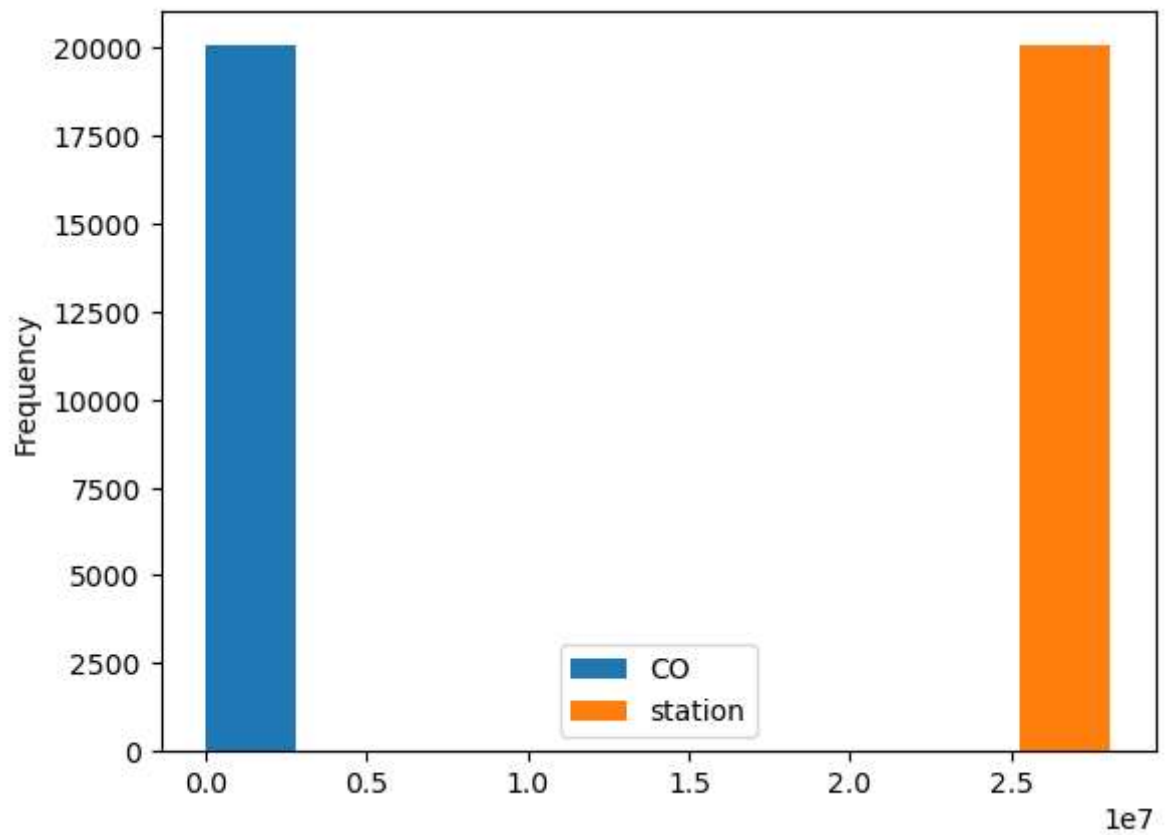
```
In [9]: b=data[0:50]  
b.plot.bar()
```

Out[9]: <Axes: >



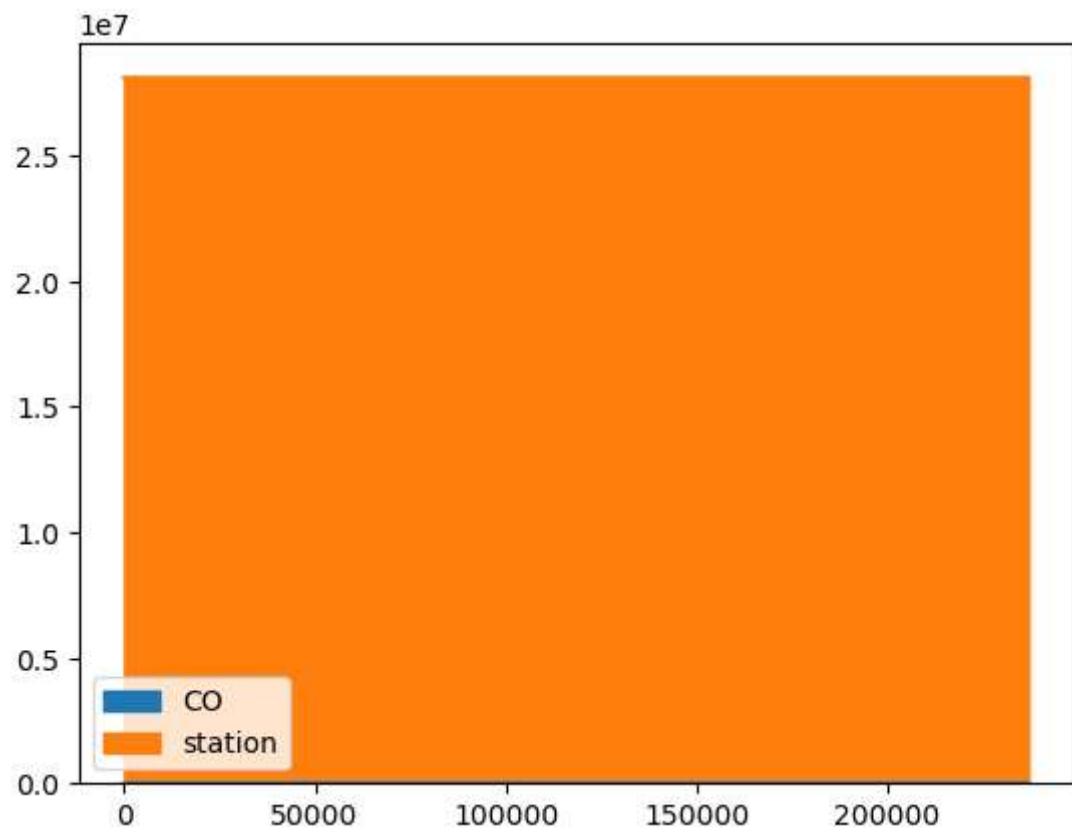
```
In [10]: data.plot.hist()
```

```
Out[10]: <Axes: ylabel='Frequency'>
```



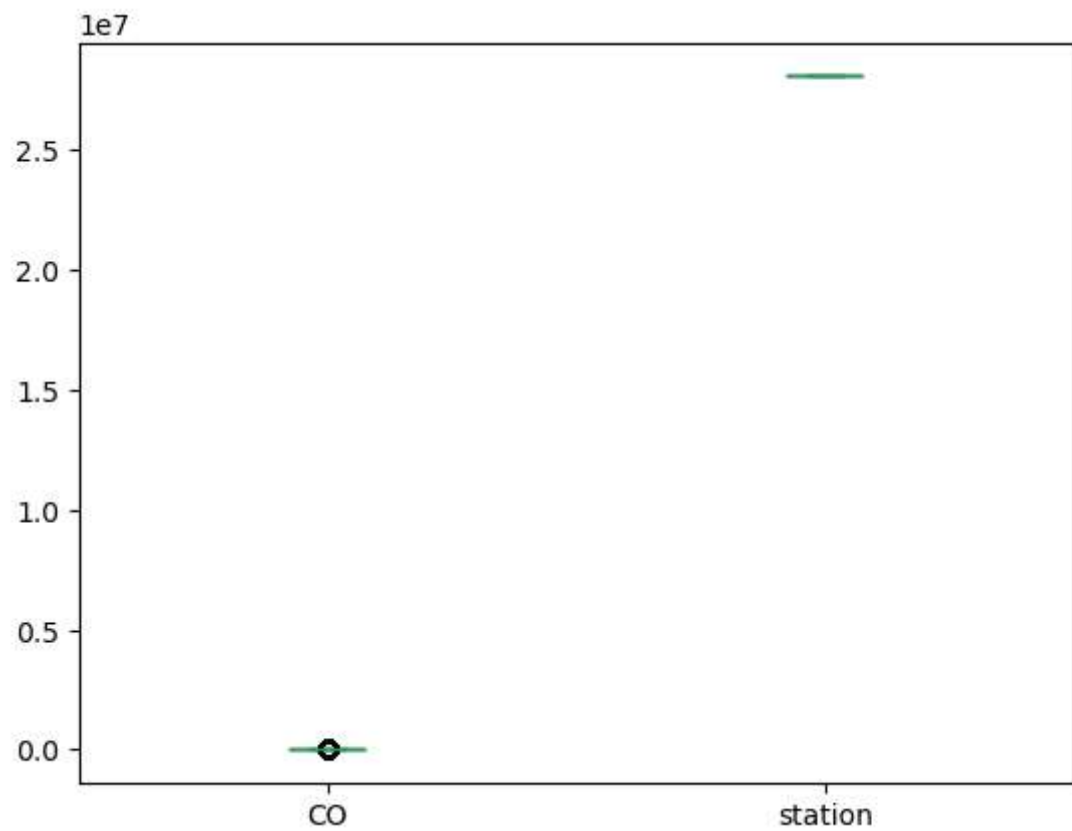
```
In [11]: data.plot.area()
```

```
Out[11]: <Axes: >
```



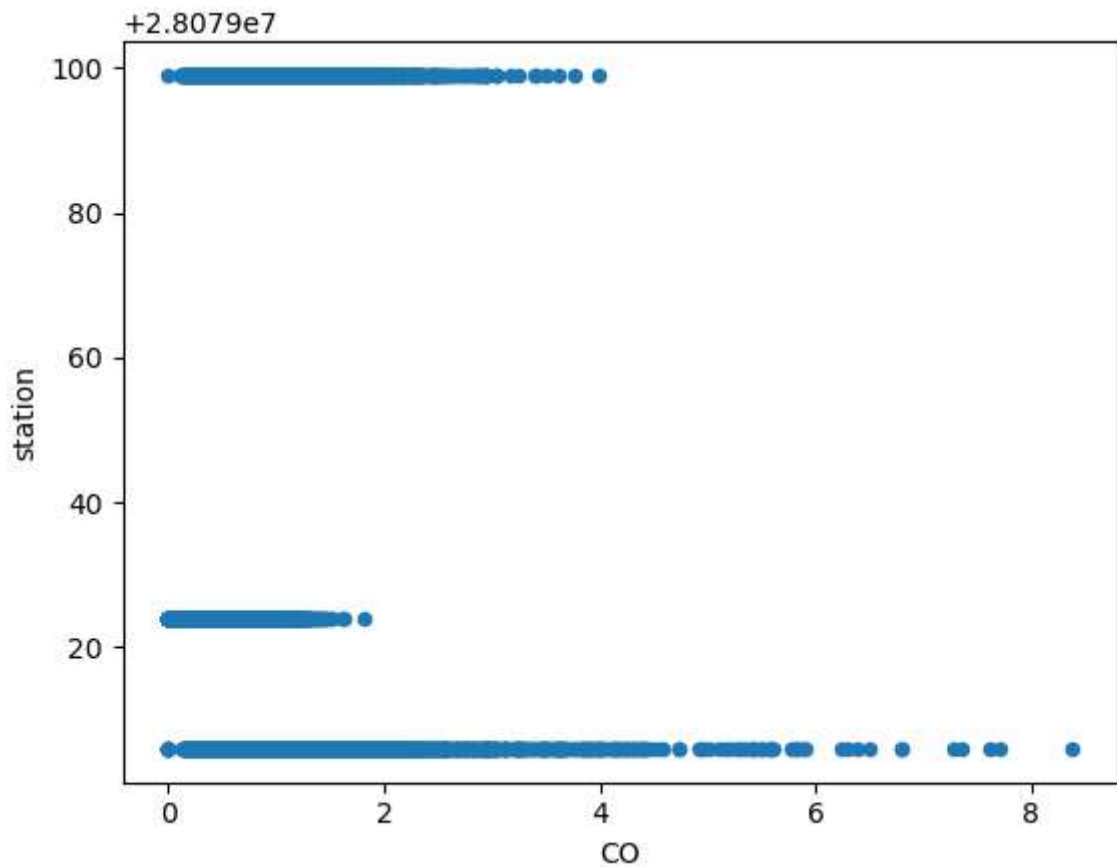
```
In [12]: data.plot.box()
```

```
Out[12]: <Axes: >
```



```
In [13]: data.plot.scatter(x='CO',y='station')
```

```
Out[13]: <Axes: xlabel='CO', ylabel='station'>
```



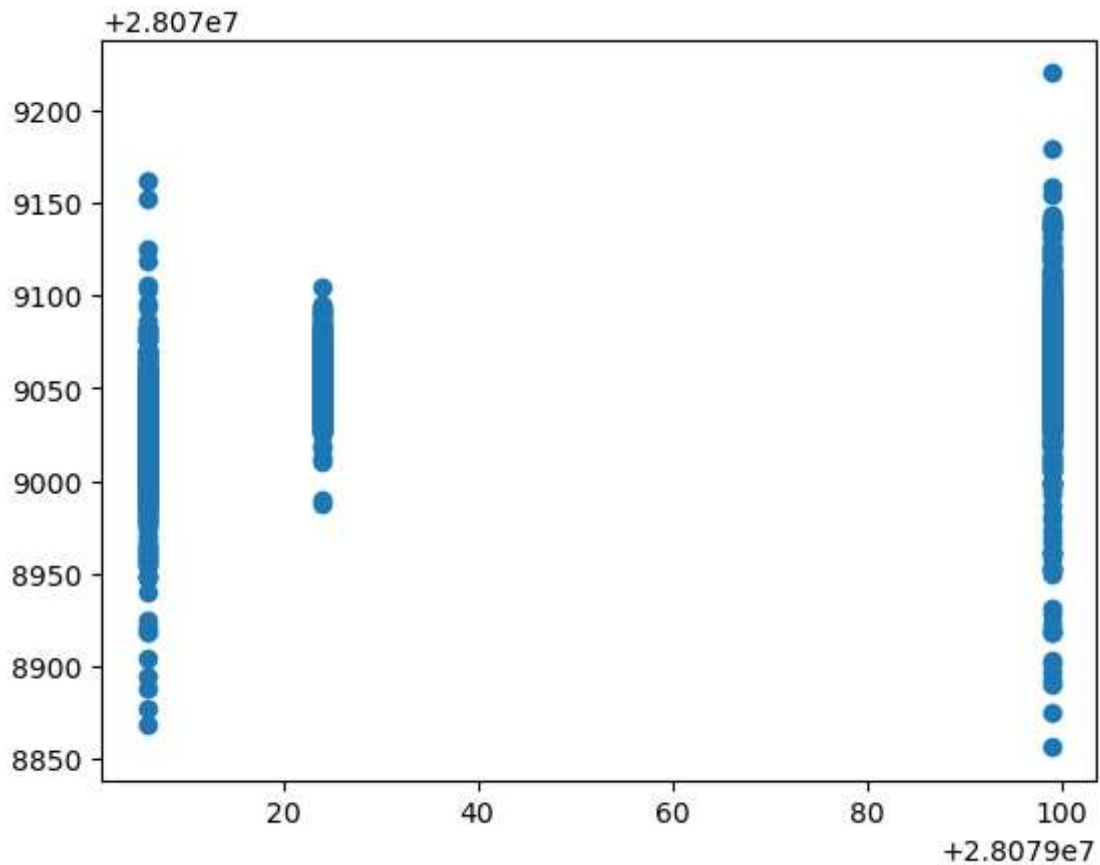
```
In [14]: x=df[['BEN', 'CO', 'EBE', 'MXV', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',  
              'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]  
y=df['station']
```

```
In [15]: 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 [16]: from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
lr.intercept_
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[16]: <matplotlib.collections.PathCollection at 0x251b61fb010>



```
In [17]: print(lr.score(x_test,y_test))
print(lr.score(x_train,y_train))
```

0.2931712221622176
0.3086854340059545

Ridge and Lasso

```
In [18]: from sklearn.linear_model import Ridge,Lasso
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
print(rr.score(x_test,y_test))
print(rr.score(x_train,y_train))
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

```
0.292621477952992
0.3084577884571391
```

```
Out[18]: 

▼ Lasso


Lasso(alpha=10)
```

```
In [19]: la.score(x_test,y_test)
```

```
Out[19]: 0.0643704470838089
```

ElasticNet

```
In [20]: from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

```
Out[20]: 

▼ ElasticNet


ElasticNet()
```

```
In [21]: en.coef_
```

```
Out[21]: array([-5.58629597,  1.53777176, -7.54592429,  2.77489042,  0.88405345,
                -0.04949124, -0.0098642 ,  2.00324766, -0.01301206,  0.24313767,
                 1.28287896,  0.14745343,  1.51680915, -0.83921788])
```

```
In [22]: en.intercept_
```

```
Out[22]: 28079049.054960202
```

```
In [23]: prediction=en.predict(x_test)
```

```
In [24]: en.score(x_test,y_test)
```

```
Out[24]: 0.17144799678940337
```

Evaluation Metrics

```
In [25]: 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)))
```

```
36.8653471896385
1548.7333223267033
39.35394925959405
```

Logistics Regression

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

```
In [27]: feature_matrix=df[['BEN', 'CO', 'EBE', 'MXV', 'NMHC', 'NO_2', 'NOx', 'OXY', 'C
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

```
In [28]: from sklearn.preprocessing import StandardScaler
fs=StandardScaler().fit_transform(feature_matrix)
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
logr.score(fs,target_vector)
```

```
Out[28]: 0.879023418036871
```

```
In [29]: observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
logr.predict_proba(observation)
```

```
Out[29]: array([[9.99896769e-01, 3.20761157e-30, 1.03230729e-04]])
```

Random Forest

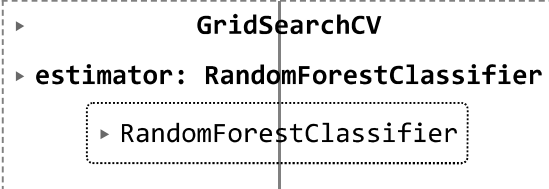
```
In [30]: from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

```
Out[30]: ▼ RandomForestClassifier
RandomForestClassifier()
```

```
In [31]: parameters={'max_depth':[1,2,3,4,5],  
                  'min_samples_leaf':[5,10,15,20,25],  
                  'n_estimators':[10,20,30,40,50]  
                }
```

```
In [32]: from sklearn.model_selection import GridSearchCV  
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="ac  
grid_search.fit(x_train,y_train)
```

```
Out[32]:
```



```
  ▶ GridSearchCV  
  ▶ estimator: RandomForestClassifier  
      ▶ RandomForestClassifier
```

```
In [33]: rfc_best=grid_search.best_estimator_  
         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'])
```

```

Out[33]: [Text(0.5270833333333333, 0.9166666666666666, 'BEN <= 1.525\ngini = 0.631\nsamples = 8869\nvalue = [5932, 2525, 5592]\nnclass = a'),
Text(0.2666666666666666, 0.75, 'OXY <= 1.005\ngini = 0.606\nsamples = 5127\nvalue = [1554, 2238, 4238]\nnclass = c'),
Text(0.1333333333333333, 0.5833333333333334, 'NO_2 <= 15.615\ngini = 0.561\nsamples = 2071\nvalue = [538, 1965, 800]\nnclass = b'),
Text(0.06666666666666667, 0.4166666666666667, 'NOx <= 16.41\ngini = 0.192\nsamples = 592\nvalue = [32, 847, 67]\nnclass = b'),
Text(0.03333333333333333, 0.25, 'EBE <= 0.785\ngini = 0.071\nsamples = 535\nvalue = [6, 836, 26]\nnclass = b'),
Text(0.016666666666666666, 0.08333333333333333, 'gini = 0.172\nsamples = 215\nvalue = [6, 311, 26]\nnclass = b'),
Text(0.05, 0.08333333333333333, 'gini = 0.0\nsamples = 320\nvalue = [0, 525, 0]\nnclass = b'),
Text(0.1, 0.25, 'BEN <= 0.37\ngini = 0.593\nsamples = 57\nvalue = [26, 11, 41]\nnclass = c'),
Text(0.08333333333333333, 0.08333333333333333, 'gini = 0.237\nsamples = 26\nvalue = [3, 2, 33]\nnclass = c'),
Text(0.11666666666666667, 0.08333333333333333, 'gini = 0.579\nsamples = 31\nvalue = [23, 9, 8]\nnclass = a'),
Text(0.2, 0.4166666666666667, 'MXY <= 1.025\ngini = 0.632\nsamples = 1479\nvalue = [506, 1118, 733]\nnclass = b'),
Text(0.16666666666666666, 0.25, 'PM10 <= 17.785\ngini = 0.468\nsamples = 458\nvalue = [103, 506, 117]\nnclass = b'),
Text(0.15, 0.08333333333333333, 'gini = 0.59\nsamples = 256\nvalue = [77, 221, 99]\nnclass = b'),
Text(0.18333333333333332, 0.08333333333333333, 'gini = 0.24\nsamples = 202\nvalue = [26, 285, 18]\nnclass = b'),
Text(0.23333333333333334, 0.25, 'PM10 <= 16.72\ngini = 0.656\nsamples = 1021\nvalue = [403, 612, 616]\nnclass = c'),
Text(0.21666666666666667, 0.08333333333333333, 'gini = 0.619\nsamples = 411\nvalue = [236, 106, 302]\nnclass = c'),
Text(0.25, 0.08333333333333333, 'gini = 0.607\nsamples = 610\nvalue = [167, 506, 314]\nnclass = b'),
Text(0.4, 0.5833333333333334, 'PXY <= 1.475\ngini = 0.421\nsamples = 3056\nvalue = [1016, 273, 3438]\nnclass = c'),
Text(0.3333333333333333, 0.4166666666666667, 'NMHC <= 0.055\ngini = 0.555\nsamples = 1451\nvalue = [688, 248, 1304]\nnclass = c'),
Text(0.3, 0.25, 'NOx <= 41.01\ngini = 0.191\nsamples = 246\nvalue = [336, 1, 39]\nnclass = a'),
Text(0.2833333333333333, 0.08333333333333333, 'gini = 0.499\nsamples = 53\nvalue = [42, 0, 38]\nnclass = a'),
Text(0.31666666666666665, 0.08333333333333333, 'gini = 0.013\nsamples = 193\nvalue = [294, 1, 1]\nnclass = a'),
Text(0.36666666666666664, 0.25, 'BEN <= 0.885\ngini = 0.486\nsamples = 1205\nvalue = [352, 247, 1265]\nnclass = c'),
Text(0.35, 0.08333333333333333, 'gini = 0.164\nsamples = 672\nvalue = [43, 49, 953]\nnclass = c'),
Text(0.38333333333333336, 0.08333333333333333, 'gini = 0.654\nsamples = 533\nvalue = [309, 198, 312]\nnclass = c'),
Text(0.4666666666666667, 0.4166666666666667, 'NO_2 <= 30.32\ngini = 0.246\nsamples = 1605\nvalue = [328, 25, 2134]\nnclass = c'),
Text(0.43333333333333335, 0.25, 'TCH <= 1.265\ngini = 0.536\nsamples = 98\nvalue = [61, 6, 74]\nnclass = c'),
Text(0.4166666666666667, 0.08333333333333333, 'gini = 0.137\nsamples = 37\nvalue = [50, 0, 4]\nnclass = a'),
Text(0.45, 0.08333333333333333, 'gini = 0.332\nsamples = 61\nvalue = [11, 6,

```

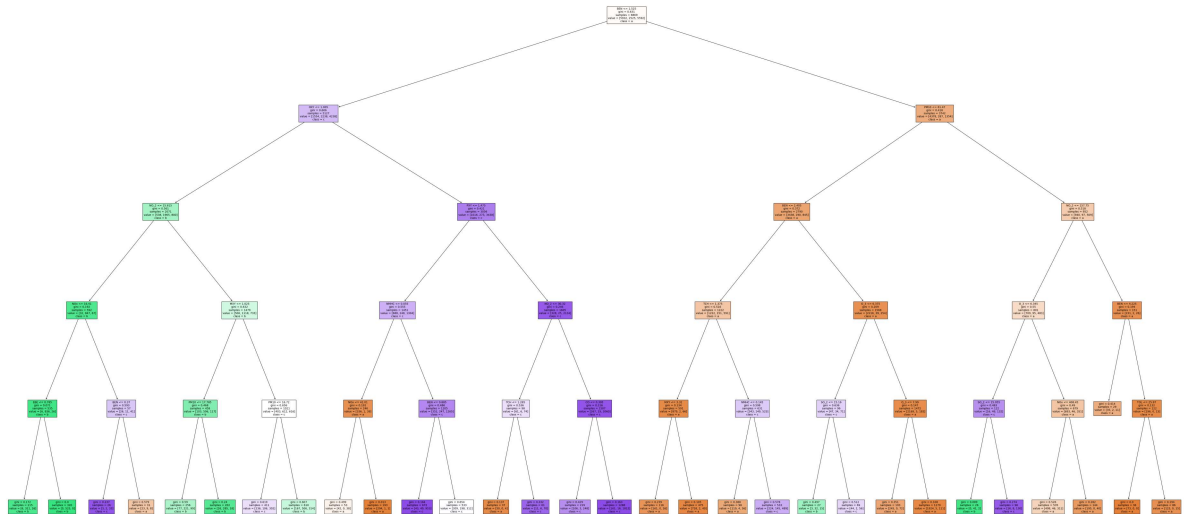
```

70]\nclasse = c'),
  Text(0.5, 0.25, 'CO <= 0.395\ngini = 0.216\nsamples = 1507\nvalue = [267, 1
9, 2060]\nclasse = c'),
  Text(0.4833333333333333, 0.0833333333333333, 'gini = 0.429\nsamples = 219
\nvalue = [106, 3, 248]\nclasse = c'),
  Text(0.5166666666666667, 0.0833333333333333, 'gini = 0.163\nsamples = 1288
\nvalue = [161, 16, 1812]\nclasse = c'),
  Text(0.7875, 0.75, 'PM10 <= 61.47\ngini = 0.418\nsamples = 3742\nvalue = [43
78, 287, 1354]\nclasse = a'),
  Text(0.6666666666666666, 0.5833333333333333, 'BEN <= 2.405\ngini = 0.372\nsa
mples = 2790\nvalue = [3438, 190, 845]\nclasse = a'),
  Text(0.6, 0.4166666666666667, 'TCH <= 1.375\ngini = 0.516\nsamples = 1222\nv
alue = [1222, 151, 591]\nclasse = a'),
  Text(0.5666666666666667, 0.25, 'MXV <= 3.32\ngini = 0.134\nsamples = 591\nva
lue = [879, 2, 66]\nclasse = a'),
  Text(0.55, 0.0833333333333333, 'gini = 0.239\nsamples = 116\nvalue = [161,
0, 26]\nclasse = a'),
  Text(0.5833333333333333, 0.0833333333333333, 'gini = 0.105\nsamples = 475\n
value = [718, 2, 40]\nclasse = a'),
  Text(0.6333333333333333, 0.25, 'NMHC <= 0.145\ngini = 0.598\nsamples = 631\n
value = [343, 149, 525]\nclasse = c'),
  Text(0.6166666666666667, 0.0833333333333333, 'gini = 0.388\nsamples = 98\nv
alue = [119, 4, 36]\nclasse = a'),
  Text(0.65, 0.0833333333333333, 'gini = 0.578\nsamples = 533\nvalue = [224,
145, 489]\nclasse = c'),
  Text(0.7333333333333333, 0.4166666666666667, 'O_3 <= 6.375\ngini = 0.209\nsa
mples = 1568\nvalue = [2216, 39, 254]\nclasse = a'),
  Text(0.7, 0.25, 'SO_2 <= 15.16\ngini = 0.636\nsamples = 96\nvalue = [47, 34,
71]\nclasse = c'),
  Text(0.6833333333333333, 0.0833333333333333, 'gini = 0.497\nsamples = 27\nv
alue = [3, 32, 15]\nclasse = b'),
  Text(0.7166666666666667, 0.0833333333333333, 'gini = 0.512\nsamples = 69\nv
alue = [44, 2, 56]\nclasse = c'),
  Text(0.7666666666666667, 0.25, 'O_3 <= 7.99\ngini = 0.147\nsamples = 1472\nv
alue = [2169, 5, 183]\nclasse = a'),
  Text(0.75, 0.0833333333333333, 'gini = 0.351\nsamples = 193\nvalue = [245,
0, 72]\nclasse = a'),
  Text(0.7833333333333333, 0.0833333333333333, 'gini = 0.108\nsamples = 1279
\nvalue = [1924, 5, 111]\nclasse = a'),
  Text(0.9083333333333333, 0.5833333333333333, 'NO_2 <= 157.75\ngini = 0.518\n
samples = 952\nvalue = [940, 97, 509]\nclasse = a'),
  Text(0.8666666666666667, 0.4166666666666667, 'O_3 <= 6.345\ngini = 0.55\nsam
ples = 801\nvalue = [709, 95, 483]\nclasse = a'),
  Text(0.8333333333333333, 0.25, 'SO_2 <= 15.055\ngini = 0.483\nsamples = 122
\nvalue = [16, 49, 132]\nclasse = c'),
  Text(0.8166666666666667, 0.0833333333333333, 'gini = 0.089\nsamples = 26\nv
alue = [0, 41, 2]\nclasse = b'),
  Text(0.85, 0.0833333333333333, 'gini = 0.274\nsamples = 96\nvalue = [16, 8,
130]\nclasse = c'),
  Text(0.9, 0.25, 'NOx <= 408.45\ngini = 0.49\nsamples = 679\nvalue = [693, 4
6, 351]\nclasse = a'),
  Text(0.8833333333333333, 0.0833333333333333, 'gini = 0.526\nsamples = 535\n
value = [498, 46, 311]\nclasse = a'),
  Text(0.9166666666666666, 0.0833333333333333, 'gini = 0.282\nsamples = 144\n
value = [195, 0, 40]\nclasse = a'),
  Text(0.95, 0.4166666666666667, 'BEN <= 4.225\ngini = 0.194\nsamples = 151\nv
alue = [231, 2, 26]\nclasse = a'),

```

```

Text(0.9333333333333333, 0.25, 'gini = 0.414\nsamples = 29\nvalue = [35, 2, 11]\nnclass = a'),
Text(0.9666666666666667, 0.25, 'TOL <= 25.97\ngini = 0.132\nsamples = 122\nvalue = [196, 0, 15]\nnclass = a'),
Text(0.95, 0.08333333333333333, 'gini = 0.0\nsamples = 36\nvalue = [73, 0, 0]\nnclass = a'),
Text(0.9833333333333333, 0.08333333333333333, 'gini = 0.194\nsamples = 86\nvalue = [123, 0, 15]\nnclass = a')]
```



Conclusion

```

In [34]: print("Linear Regression:",lr.score(x_test,y_test))
print("Ridge Regression:",rr.score(x_test,y_test))
print("Lasso Regression",la.score(x_test,y_test))
print("ElasticNet Regression:",en.score(x_test,y_test))
print("Logistic Regression:",logr.score(fs,target_vector))
print("Random Forest:",grid_search.best_score_)
```

```

Linear Regression: 0.2931712221622176
Ridge Regression: 0.292621477952992
Lasso Regression 0.0643704470838089
ElasticNet Regression: 0.17144799678940337
Logistic Regression: 0.879023418036871
Random Forest: 0.8663959966439416
```

Logistic Is Better!!!