# Importing Libraries

In [491]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

# Importing Datasets

In [492]:
```
df=pd.read_csv(r"C:\Users\user\Desktop\csvs_per_year\csvs_per_year\madrid_2011.
df
```

Out[492]:

| | date | BEN | CO | EBE | NMHC | NO | NO_2 | O_3 | PM10 | PM25 | SO_2 | TCH | TOL | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2011-11-01 01:00:00 | NaN | 1.0 | NaN | NaN | 154.0 | 84.0 | NaN | NaN | NaN | 6.0 | NaN | NaN | 2 |
| **1** | 2011-11-01 01:00:00 | 2.5 | 0.4 | 3.5 | 0.26 | 68.0 | 92.0 | 3.0 | 40.0 | 24.0 | 9.0 | 1.54 | 8.7 | 2 |
| **2** | 2011-11-01 01:00:00 | 2.9 | NaN | 3.8 | NaN | 96.0 | 99.0 | NaN | NaN | NaN | NaN | NaN | 7.2 | 2 |
| **3** | 2011-11-01 01:00:00 | NaN | 0.6 | NaN | NaN | 60.0 | 83.0 | 2.0 | NaN | NaN | NaN | NaN | NaN | 2 |
| **4** | 2011-11-01 01:00:00 | NaN | NaN | NaN | NaN | 44.0 | 62.0 | 3.0 | NaN | NaN | 3.0 | NaN | NaN | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **209923** | 2011-09-01 00:00:00 | NaN | 0.2 | NaN | NaN | 5.0 | 19.0 | 44.0 | NaN | NaN | NaN | NaN | NaN | 2 |
| **209924** | 2011-09-01 00:00:00 | NaN | 0.1 | NaN | NaN | 6.0 | 29.0 | NaN | 11.0 | NaN | 7.0 | NaN | NaN | 2 |
| **209925** | 2011-09-01 00:00:00 | NaN | NaN | NaN | 0.23 | 1.0 | 21.0 | 28.0 | NaN | NaN | NaN | 1.44 | NaN | 2 |
| **209926** | 2011-09-01 00:00:00 | NaN | NaN | NaN | NaN | 3.0 | 15.0 | 48.0 | NaN | NaN | NaN | NaN | NaN | 2 |
| **209927** | 2011-09-01 00:00:00 | NaN | NaN | NaN | NaN | 4.0 | 33.0 | 38.0 | 13.0 | NaN | NaN | NaN | NaN | 2 |

209928 rows × 14 columns

# Data Cleaning and Data Preprocessing

In [493]:
```
df=df.dropna()
```

In [494]:
```
df.columns
```

Out[494]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM2
5',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [495]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16460 non-null  object 
 1   BEN      16460 non-null  float64
 2   CO       16460 non-null  float64
 3   EBE      16460 non-null  float64
 4   NMHC     16460 non-null  float64
 5   NO       16460 non-null  float64
 6   NO_2     16460 non-null  float64
 7   O_3      16460 non-null  float64
 8   PM10     16460 non-null  float64
 9   PM25     16460 non-null  float64
 10  SO_2     16460 non-null  float64
 11  TCH      16460 non-null  float64
 12  TOL      16460 non-null  float64
 13  station  16460 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

In [496]:
```
data=df[['CO' ,'station']]
data
```

Out[496]:

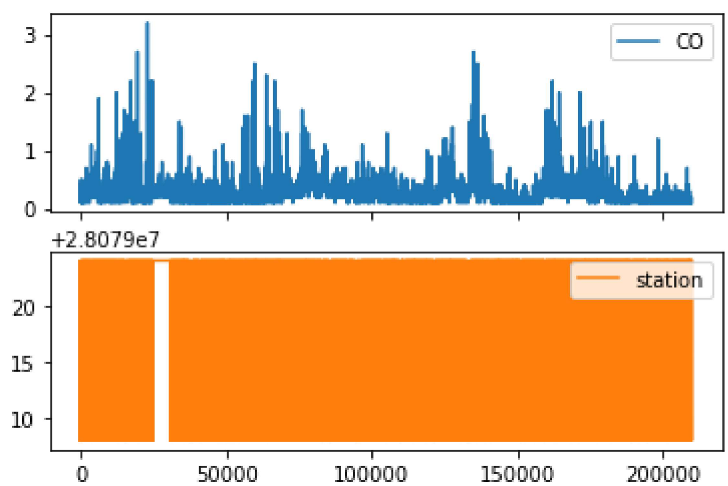
| | CO | station |
|---|---|---|
| **1** | 0.4 | 28079008 |
| **6** | 0.3 | 28079024 |
| **25** | 0.3 | 28079008 |
| **30** | 0.4 | 28079024 |
| **49** | 0.2 | 28079008 |
| **...** | ... | ... |
| **209862** | 0.1 | 28079024 |
| **209881** | 0.1 | 28079008 |
| **209886** | 0.1 | 28079024 |
| **209905** | 0.1 | 28079008 |
| **209910** | 0.1 | 28079024 |

16460 rows × 2 columns

# Line chart

In [497]:
```
data.plot.line(subplots=True)
```

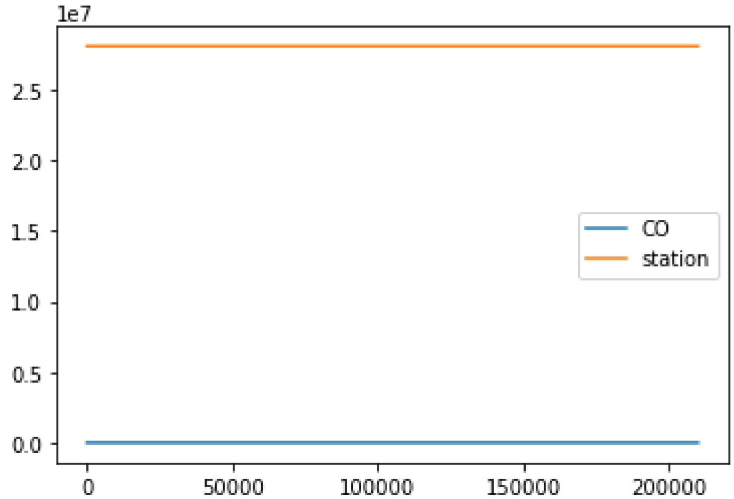
Out[497]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)




# Line chart

In [498]:
```
data.plot.line()
```

Out[498]: <AxesSubplot:>

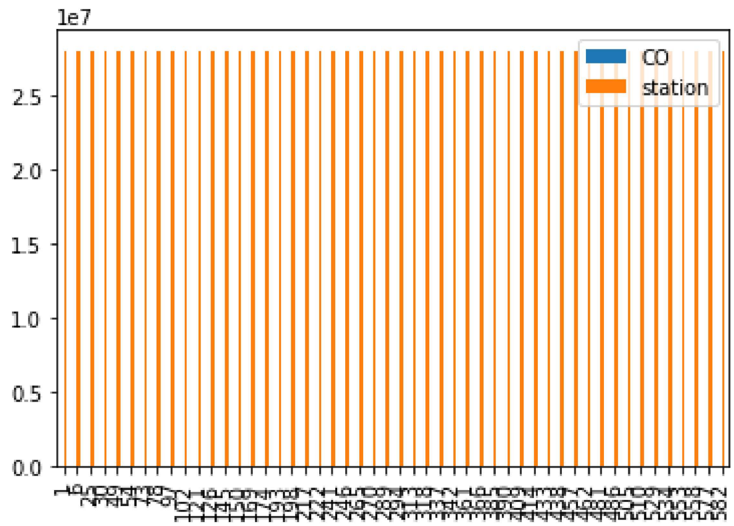



# Bar chart

In [499]:
```
b=data[0:50]
```

In [500]:
```
b.plot.bar()
```

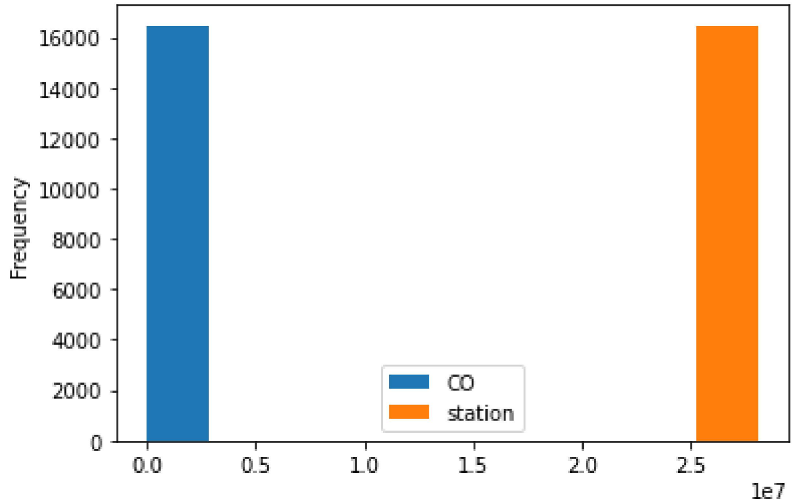
Out[500]: <AxesSubplot:>




# Histogram

In [501]:
```
data.plot.hist()
```

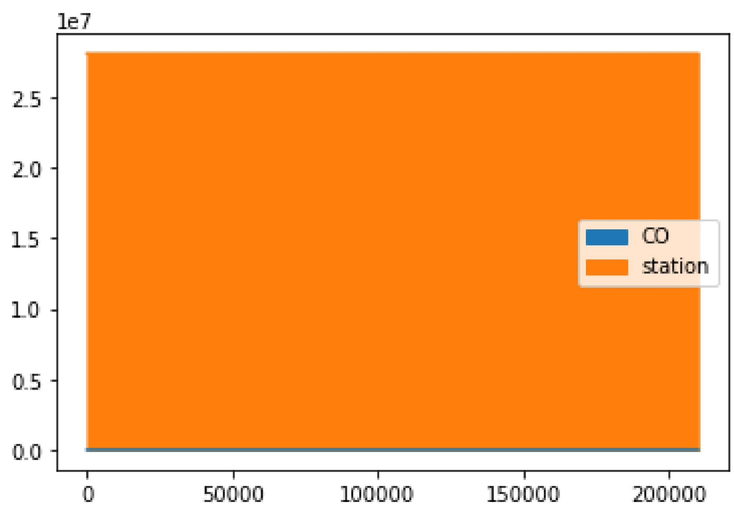
Out[501]: <AxesSubplot:ylabel='Frequency'>




# Area chart

In [502]:
```
data.plot.area()
```

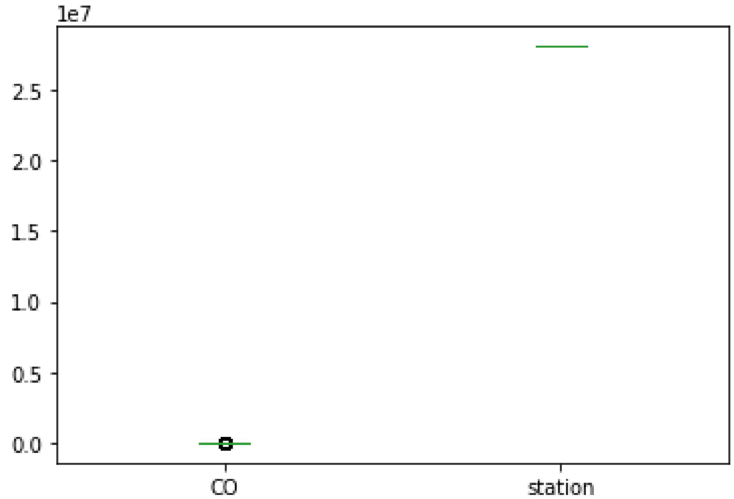
Out[502]: <AxesSubplot:>




# Box chart

In [503]:
```
data.plot.box()
```

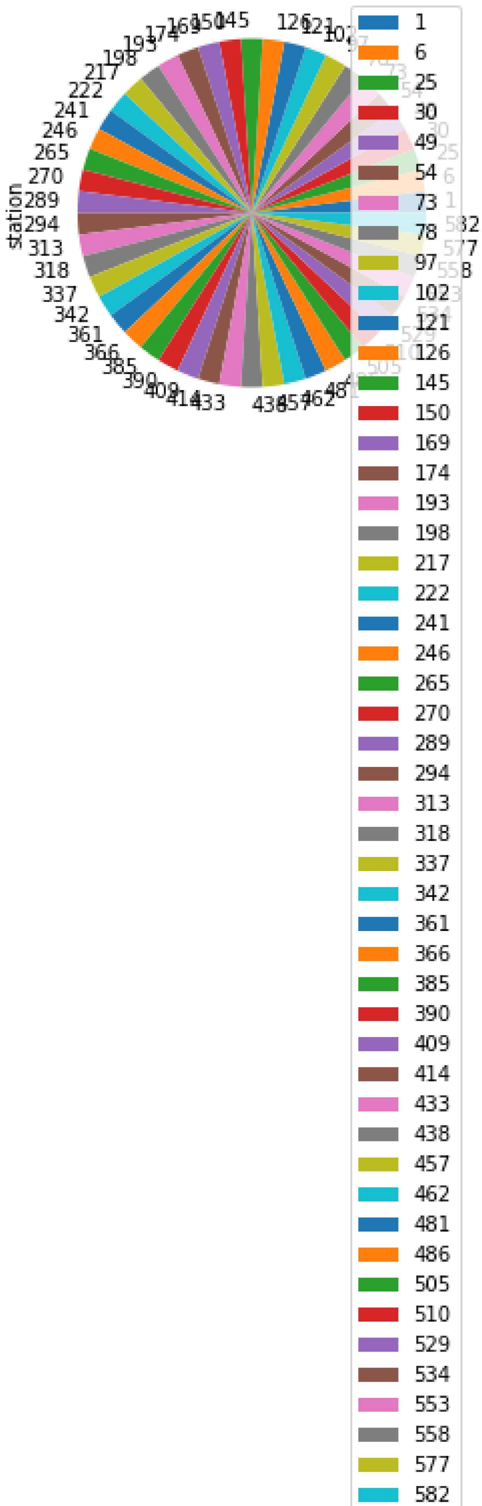
Out[503]: <AxesSubplot:>




# Pie chart

In [504]:
```
b.plot.pie(y='station' )
```

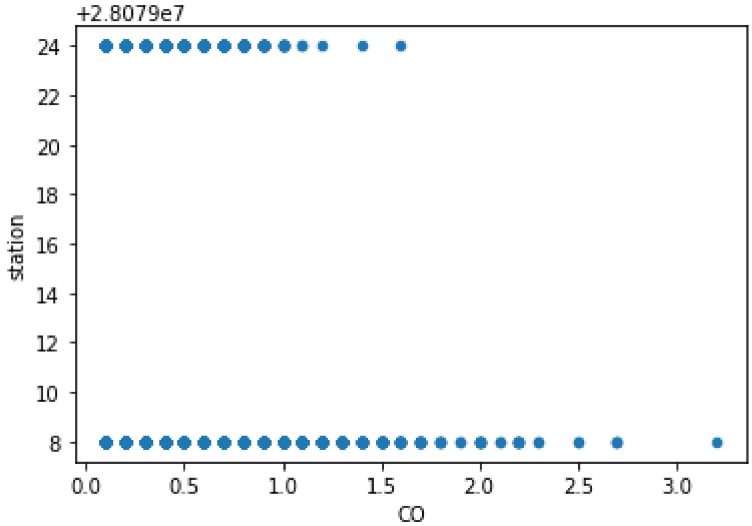
Out[504]: <AxesSubplot:ylabel='station'>




## Scatter chart

In [505]:
```
data.plot.scatter(x='CO' ,y='station')
```

Out[505]: <AxesSubplot:xlabel='CO', ylabel='station'>




In [506]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16460 non-null  object 
 1   BEN      16460 non-null  float64
 2   CO       16460 non-null  float64
 3   EBE      16460 non-null  float64
 4   NMHC     16460 non-null  float64
 5   NO       16460 non-null  float64
 6   NO_2     16460 non-null  float64
 7   O_3      16460 non-null  float64
 8   PM10     16460 non-null  float64
 9   PM25     16460 non-null  float64
 10  SO_2     16460 non-null  float64
 11  TCH      16460 non-null  float64
 12  TOL      16460 non-null  float64
 13  station  16460 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

In [507]:
```
df.describe()
```

Out[507]:

| | BEN | CO | EBE | NMHC | NO | NO_2 |
|---|---|---|---|---|---|---|
| **count** | 16460.000000 | 16460.000000 | 16460.000000 | 16460.000000 | 16460.000000 | 16460.000000 |
| **mean** | 0.900680 | 0.277758 | 1.471871 | 0.167043 | 23.671810 | 44.583961 |
| **std** | 0.768892 | 0.206143 | 1.051004 | 0.075068 | 44.362859 | 31.569185 |
| **min** | 0.100000 | 0.100000 | 0.200000 | 0.010000 | 1.000000 | 1.000000 |
| **25%** | 0.500000 | 0.200000 | 0.800000 | 0.120000 | 2.000000 | 19.000000 |
| **50%** | 0.700000 | 0.200000 | 1.200000 | 0.160000 | 7.000000 | 40.000000 |
| **75%** | 1.100000 | 0.300000 | 1.700000 | 0.200000 | 25.000000 | 63.000000 |
| **max** | 9.500000 | 3.200000 | 12.800000 | 0.840000 | 615.000000 | 289.000000 |

In [508]:
```
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-508-9c3e63dc22cd> 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 [ ]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

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

# ACCURACY

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

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

# Ridge and Lasso

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

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

# Accuracy(Ridge)

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

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

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

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

# Accuracy(Lasso)

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

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

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

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

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

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

# Evaluation Metrics

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

# Logistic Regression

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

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

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

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

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

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

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

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

In [509]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [510]:
```
logr.classes_
```

Out[510]:
```
array([28079006, 28079024, 28079099], dtype=int64)
```

In [511]:
```
logr.score(fs,target_vector)
```

Out[511]:
```
0.8951733624630821
```

In [512]:
```
logr.predict_proba(observation)[0][0]
```

Out[512]:
```
5.447205522232353e-13
```

In [513]:
```
logr.predict_proba(observation)
```

Out[513]:
```
array([[5.44720552e-13, 8.28692830e-44, 1.00000000e+00]])
```

# Random Forest

In [514]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [515]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[515]:
```
RandomForestClassifier()
```

In [516]:
```
parameters={'max_depth':[1,2,3,4,5],
 'min_samples_leaf':[5,10,15,20,25],
 'n_estimators':[10,20,30,40,50]
}
```

In [517]:
```
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[517]:
```
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 [518]:
```
grid_search.best_score_
```

Out[518]:
```
0.8990805901018493
```

In [519]:
```
rfc_best=grid_search.best_estimator_
```

In [520]:
```
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',
```

```
31, 18]\nclass = a'),
 Text(418.5, 543.5999999999999, 'TCH <= 1.435\ngini = 0.424\nsamples = 929
\nvalue = [1061, 416, 25]\nclass = a'),
 Text(348.75, 181.19999999999982, 'gini = 0.368\nsamples = 831\nvalue = [10
29, 291, 25]\nclass = a'),
 Text(488.25, 181.19999999999982, 'gini = 0.325\nsamples = 98\nvalue = [32,
125, 0]\nclass = b'),
 Text(837.0, 906.0, 'TOL <= 0.795\ngini = 0.582\nsamples = 3312\nvalue = [4
98, 2350, 2369]\nclass = c'),
 Text(697.5, 543.5999999999999, 'BEN <= 0.345\ngini = 0.245\nsamples = 761
\nvalue = [0, 1043, 174]\nclass = b'),
 Text(627.75, 181.19999999999982, 'gini = 0.479\nsamples = 187\nvalue = [0,
179, 118]\nclass = b'),
 Text(767.25, 181.19999999999982, 'gini = 0.114\nsamples = 574\nvalue = [0,
864, 56]\nclass = b'),
 Text(976.5, 543.5999999999999, 'TCH <= 1.325\ngini = 0.577\nsamples = 2551
\nvalue = [498, 1307, 2195]\nclass = c'),
 Text(906.75, 181.19999999999982, 'gini = 0.371\nsamples = 522\nvalue = [4
9, 646, 141]\nclass = b'),
 Text(1046.25, 181.19999999999982, 'gini = 0.515\nsamples = 2029\nvalue =
```

# Conclusion

# Accuracy

Linear Regression:0.440285701113897

Ridge Regression:0.17862590308954918

Lasso Regression:0.4106663168409096

ElasticNet Regression:0.2308125912534129

Logistic Regression:0.8660366036603661

Random Forest:0.9273467638234033

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

In [ ]: