SVM:support Vector machanisam goal is to create the best line or decision boundray that can sepaaret the n-dimensional space into classes so we can easily put the new data point in the correct category in the future this new data points are will not effect the decision boundry due to it took the help of support vectors at the end of the off the streets.

In [95]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn import svm
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

In [96]:
```
cd \Users\758449\Downloads
```

```
C:\Users\758449\Downloads
```

In [ ]:

In [83]:
```
from sklearn import datasets
```

In [97]:
```
cc_data = pd.read_csv("waterPollution.csv")
```

In [98]:
```
cc_data.head().head()
```

Out[98]:

	parameterWaterBodyCategory	observedPropertyDeterminandCode	procedureAnalysedFraction	procedureAnalysedMedia	resultUom	phe
0	RW	CAS_14797-65-0	total	water	mg{NO2}/L	
1	RW	CAS_14797-65-0	total	water	mg{NO2}/L	
2	RW	EEA_3164-07-6	total	water	{massRatio}	
3	RW	CAS_14797-55-8	total	water	mg{NO3}/L	
4	RW	EEA_3151-01-7	total	water	mmol/L	

5 rows × 29 columns

In [99]:
```
cc_data.drop(['parameterWaterBodyCategory','observedPropertyDeterminandCode','resultUom','parameterSamp
lingPeriod','waterBodyIdentifier'],axis =1,inplace =True)
```

In [100]:
```
cc_data.isnull().sum()
```

Out[100]:
```
procedureAnalysedFraction                    0
procedureAnalysedMedia                       0
phenomenonTimeReferenceYear                  0
resultMeanValue                              0
Country                                      0
PopulationDensity                          107
TerraMarineProtected_2016_2018             107
TouristMean_1990_2020                      107
VenueCount                                   0
netMigration_2011_2018                     107
droughts_floods_temperature                107
literacyRate_2010_2018                     107
combustibleRenewables_2009_2014            107
gdp                                        107
composition_food_organic_waste_percent     107
composition_glass_percent                  107
composition_metal_percent                  107
composition_other_percent                  107
composition_paper_cardboard_percent        107
composition_plastic_percent                107
composition_rubber_leather_percent         107
composition_wood_percent                   107
composition_yard_garden_green_waste_percent 107
waste_treatment_recycling_percent          107
dtype: int64
```

In [101]:
```
cc_data['PopulationDensity'].fillna(cc_data.PopulationDensity.median(),inplace =True)
cc_data['TerraMarineProtected_2016_2018'].fillna(cc_data.TerraMarineProtected_2016_2018.median(),inpla
ce =True)
cc_data['TouristMean_1990_2020'].fillna(cc_data.TouristMean_1990_2020.median(),inplace =True)
cc_data['netMigration_2011_2018'].fillna(cc_data.netMigration_2011_2018.median(),inplace =True)
cc_data['droughts_floods_temperature'].fillna(cc_data.droughts_floods_temperature.median(),inplace =Tr
ue)
cc_data['literacyRate_2010_2018'].fillna(cc_data.literacyRate_2010_2018.median(),inplace =True)
cc_data['combustibleRenewables_2009_2014'].fillna(cc_data.combustibleRenewables_2009_2014.median(),inp
lace =True)
cc_data['gdp'].fillna(cc_data.gdp.median(),inplace =True)
cc_data['composition_food_organic_waste_percent'].fillna(cc_data.composition_food_organic_waste_percen
t.median(),inplace =True)
cc_data['composition_glass_percent'].fillna(cc_data.composition_glass_percent.median(),inplace =True)
cc_data['composition_metal_percent'].fillna(cc_data.composition_metal_percent.median(),inplace =True)
cc_data['composition_other_percent'].fillna(cc_data.composition_other_percent.median(),inplace =True)
cc_data['composition_paper_cardboard_percent'].fillna(cc_data.composition_paper_cardboard_percent.medi
an(),inplace =True)
cc_data['composition_plastic_percent'].fillna(cc_data.composition_plastic_percent.median(),inplace =Tr
ue)
cc_data['composition_rubber_leather_percent'].fillna(cc_data.composition_rubber_leather_percent.median
(),inplace =True)
cc_data['composition_wood_percent'].fillna(cc_data.composition_wood_percent.median(),inplace =True)
cc_data['composition_yard_garden_green_waste_percent'].fillna(cc_data.composition_yard_garden_green_wa
ste_percent.median(),inplace =True)
cc_data['waste_treatment_recycling_percent'].fillna(cc_data.waste_treatment_recycling_percent.median
(),inplace =True)
```

In [102]:
```
#checking whether null data removed or not
cc_data.isnull().sum()
```

Out[102]:
```
procedureAnalysedFraction                    0
procedureAnalysedMedia                       0
phenomenonTimeReferenceYear                  0
resultMeanValue                              0
Country                                      0
PopulationDensity                            0
TerraMarineProtected_2016_2018               0
TouristMean_1990_2020                        0
VenueCount                                   0
netMigration_2011_2018                       0
droughts_floods_temperature                  0
literacyRate_2010_2018                       0
combustibleRenewables_2009_2014              0
gdp                                          0
composition_food_organic_waste_percent       0
composition_glass_percent                    0
composition_metal_percent                    0
composition_other_percent                    0
composition_paper_cardboard_percent          0
composition_plastic_percent                  0
composition_rubber_leather_percent           0
composition_wood_percent                     0
composition_yard_garden_green_waste_percent  0
waste_treatment_recycling_percent            0
dtype: int64
```

In [104]:
```
cat_data_column = cc_data.select_dtypes(include =['object']).columns
```

In [105]:
```
cat_data_column
```

Out[105]:
```
Index(['procedureAnalysedFraction', 'procedureAnalysedMedia', 'Country'], dtype='object')
```

In [106]:
```
columns = ('procedureAnalysedFraction', 'procedureAnalysedMedia', 'Country')
for i in columns:
  le = LabelEncoder()
  cc_data[i] = le.fit_transform(cc_data[i])
```

In [107]:
```
columns
```

Out[107]:
```
('procedureAnalysedFraction', 'procedureAnalysedMedia', 'Country')
```

In [108]:
```
cc_data
```

Out[108]:

	procedureAnalysedFraction	procedureAnalysedMedia	phenomenonTimeReferenceYear	resultMeanValue	Country	PopulationDensity
0	2	1	2009	0.063310	8	122.299437
1	2	1	2009	0.046733	8	122.299437
2	2	1	2009	132.859000	8	122.299437
3	2	1	2009	11.578376	8	122.299437
4	2	1	2009	0.206800	24	93.677197
...	...	...	...	...	...	...
19995	2	1	2009	0.092466	8	122.299437
19996	2	1	2009	89.908300	8	122.299437
19997	2	1	2009	18.901608	8	122.299437
19998	2	1	2009	307.307000	8	122.299437
19999	2	1	2009	7.954790	8	122.299437

20000 rows × 24 columns

In [109]:
```
cc_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 24 columns):
 #   Column                                       Non-Null Count  Dtype  
---  ------                                       --------------  -----  
 0   procedureAnalysedFraction                    20000 non-null  int32  
 1   procedureAnalysedMedia                       20000 non-null  int32  
 2   phenomenonTimeReferenceYear                  20000 non-null  int64  
 3   resultMeanValue                              20000 non-null  float64
 4   Country                                      20000 non-null  int32  
 5   PopulationDensity                            20000 non-null  float64
 6   TerraMarineProtected_2016_2018               20000 non-null  float64
 7   TouristMean_1990_2020                        20000 non-null  float64
 8   VenueCount                                   20000 non-null  float64
 9   netMigration_2011_2018                       20000 non-null  float64
 10  droughts_floods_temperature                  20000 non-null  float64
 11  literacyRate_2010_2018                       20000 non-null  float64
 12  combustibleRenewables_2009_2014              20000 non-null  float64
 13  gdp                                          20000 non-null  float64
 14  composition_food_organic_waste_percent       20000 non-null  float64
 15  composition_glass_percent                    20000 non-null  float64
 16  composition_metal_percent                    20000 non-null  float64
 17  composition_other_percent                    20000 non-null  float64
 18  composition_paper_cardboard_percent          20000 non-null  float64
 19  composition_plastic_percent                  20000 non-null  float64
 20  composition_rubber_leather_percent           20000 non-null  float64
 21  composition_wood_percent                     20000 non-null  float64
 22  composition_yard_garden_green_waste_percent  20000 non-null  float64
 23  waste_treatment_recycling_percent            20000 non-null  float64
dtypes: float64(20), int32(3), int64(1)
memory usage: 3.4 MB
```

In [120]:
```
cc_data.columns
```

Out[120]:
```
Index(['procedureAnalysedFraction', 'procedureAnalysedMedia',
       'phenomenonTimeReferenceYear', 'resultMeanValue', 'Country',
       'PopulationDensity', 'TerraMarineProtected_2016_2018',
       'TouristMean_1990_2020', 'VenueCount', 'netMigration_2011_2018',
       'droughts_floods_temperature', 'literacyRate_2010_2018',
       'combustibleRenewables_2009_2014', 'gdp',
       'composition_food_organic_waste_percent', 'composition_glass_percent',
       'composition_metal_percent', 'composition_other_percent',
       'composition_paper_cardboard_percent', 'composition_plastic_percent',
       'composition_rubber_leather_percent', 'composition_wood_percent',
       'composition_yard_garden_green_waste_percent',
       'waste_treatment_recycling_percent'],
      dtype='object')
```

In [123]:
```
X = new_data.drop(["resultMeanValue"],axis = 1)
y = new_data["resultMeanValue"].values
```

In [124]:
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.33, random_state=42)
```

In [125]:
```
X_train.shape
```

Out[125]:
```
(13400, 23)
```

In [126]:
```
X_test.shape
```

Out[126]:
```
(6600, 23)
```

In [140]:
```
X_train
```

Out[140]:
```
array([[1.        , 0.1878453 , 1.        , ..., 0.04067403, 0.        ,
        0.52560017],
       [1.        , 0.90055249, 0.5       , ..., 0.        , 0.        ,
        0.45676652],
       [1.        , 0.75138122, 1.        , ..., 0.        , 0.        ,
        0.45676652],
       ...,
       [1.        , 0.86187845, 1.        , ..., 0.        , 0.        ,
        0.45676652],
       [1.        , 0.96132597, 1.        , ..., 0.        , 0.        ,
        0.45676652],
       [1.        , 0.93370166, 1.        , ..., 0.44160372, 0.08864084,
        0.56277884]])
```

In [121]:
```
# scaler = MinMaxScaler()
# new_data = pd.DataFrame(scaler.fit_transform(cc_data),columns=['procedureAnalysedFraction', 'procedu
reAnalysedMedia',
#        'phenomenonTimeReferenceYear', 'resultMeanValue', 'Country',
#        'PopulationDensity', 'TerraMarineProtected_2016_2018',
#        'TouristMean_1990_2020', 'VenueCount', 'netMigration_2011_2018',
#        'droughts_floods_temperature', 'literacyRate_2010_2018',
#        'combustibleRenewables_2009_2014', 'gdp',
#        'composition_food_organic_waste_percent', 'composition_glass_percent',
#        'composition_metal_percent', 'composition_other_percent',
#        'composition_paper_cardboard_percent', 'composition_plastic_percent',
#        'composition_rubber_leather_percent', 'composition_wood_percent',
#        'composition_yard_garden_green_waste_percent',
#        'waste_treatment_recycling_percent'])
```

In [131]:
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(x_train)
X_test = scaler.fit_transform(x_test)
```

In [161]:
```
from sklearn.svm import SVR # "Support vector Regr"
reg = SVR(kernel='rbf')
reg.fit(X_train, y_train)
```

Out[161]:
```
SVR()
```

In [162]:
```
pred = reg.predict(X_test)
```

In [163]:
```
pred
```

Out[163]:
```
array([0.09990458, 0.08484328, 0.09936665, ..., 0.09512323, 0.09546222,
       0.0968252 ])
```

SVM classification :

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM. The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane. We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

In [175]:
```
iris = datasets.load_iris()
```

In [180]:
```
iris.feature_names
```

Out[180]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [182]:
```
df = pd.DataFrame(iris.data ,columns =iris.feature_names)
```

In [184]:
```
df.head()
```

Out[184]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [187]:
```
df['target']=iris.target
df.head()
```

Out[187]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [189]:
```
iris.target_names
```

Out[189]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [191]:
```
df['flower_name']=df.target.apply(lambda x:iris.target_names[x])
```

In [193]:
```
df.head()
```

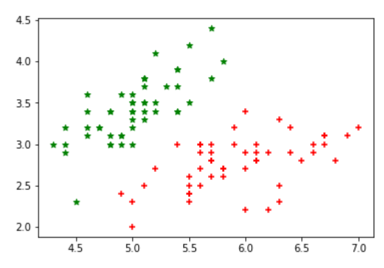
Out[193]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

In [194]:
```
df0 = df[df.target==0]
df1 = df[df.target==1]
df2 = df[df.target==2]
```

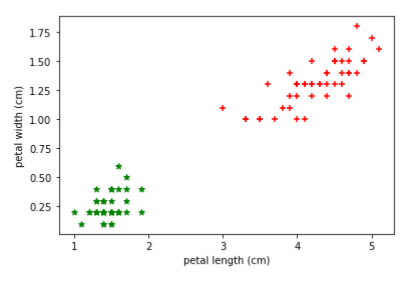
In [202]:
```
plt.scatter(df0['sepal length (cm)'],df0['sepal width (cm)'],color ='Green',marker = '*')
plt.scatter(df1['sepal length (cm)'],df1['sepal width (cm)'],color ='red',marker = '+')
#plt.scatter(df2['sepal length (cm)'],df2['sepal width (cm)'],color ='yellow',marker = '+')
```

Out[202]:
```
<matplotlib.collections.PathCollection at 0x27386ee4610>
```



In [205]:
```
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)')
plt.scatter(df0['petal length (cm)'],df0['petal width (cm)'],color ='Green',marker = '*')
plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],color ='red',marker = '+')
```

Out[205]:
```
<matplotlib.collections.PathCollection at 0x27387316f10>
```



In [206]:
```
from sklearn.model_selection import train_test_split
```

In [209]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [208]:
```
X= df.drop(['target','flower_name'],axis = 'columns')
y = df.target
```

In [211]:
```
X_train.head()
```

Out[211]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
96	5.7	2.9	4.2	1.3
105	7.6	3.0	6.6	2.1
66	5.6	3.0	4.5	1.5
0	5.1	3.5	1.4	0.2
122	7.7	2.8	6.7	2.0

In [226]:
```
from sklearn.svm import SVC
model = SVC(kernel='rbf',C=100)
```

In [229]:
```
model.fit(X_train,y_train)
```

Out[229]:
```
SVC(C=100)
```

In [231]:
```
pred = model.predict(X_test)
```

In [234]:
```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, pred)
```

In [235]:
```
cm
```

Out[235]:
```
array([[19,  0,  0],
       [ 0, 15,  0],
       [ 0,  1, 15]], dtype=int64)
```

In [ ]:

In [228]:
```
model.score(X_test,y_test)
```

Out[228]:
```
0.98
```