Importing Modules

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import datasets
from collections import Counter
```

Loading Data

In [2]:
```
iris = datasets.load_iris()
species = iris.target
data = pd.DataFrame(np.c_[iris.data, species.reshape((species.shape[0],1))], columns
data.head()
```

Out[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

In [3]:
```
data['Species'].value_counts()
```

Out[3]:
```
2.0    50
1.0    50
0.0    50
Name: Species, dtype: int64
```

Splitting into train and test

In [4]:
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size = 0.2, random_state = 123)
train.head()
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
130	7.4	2.8	6.1	1.9	2.0
119	6.0	2.2	5.0	1.5	2.0
29	4.7	3.2	1.6	0.2	0.0
0	5.1	3.5	1.4	0.2	0.0
62	6.0	2.2	4.0	1.0	1.0

Making K-Nearest neighbor function

In [5]:
```
class knn():
    def __init__(self,X, y, k_neighbors):
        self.k_neighbors = k_neighbors
        self.X_train = X
        self.Y_train = y
        self.target = set(y)

    # calculating Euclidean distance
```

```
    def euclidean_distance(self,row1,row2):
        distance = 0.0
        for i in range(len(row1)):
            distance += (row1[i]-row2[i])**2
        return np.sqrt(distance)

    def sort_distance(self,r):
        return r[2]

    #getting nearest neighbours
    def get_neighbors(self,row):
        dist = []
        for row_index in range(len(self.X_train)):
            d = self.euclidean_distance(self.X_train.iloc[row_index,:], row)
            dist.append((self.X_train.iloc[row_index,:],self.Y_train.iloc[row_index]
        dist.sort(key = self.sort_distance)

        neighbors = []
        for i in range(self.k_neighbors):
            neighbors.append(dist[i][1])
        return neighbors

    #counting the max output value that will be result
    def predict(self,row):
        neigh = self.get_neighbors(row)
        neighbors = Counter(neigh)
        count = 0
        pred = ""
        for i in self.target:
            if neighbors[i]>count:
                count = neighbors[i]
                pred = i
        return pred
```

In [6]:
```
X = train.drop('Species',axis = 1)
y = train['Species']
clf = knn(X, y, 5)
X.loc[0,:]
```

Out[6]:
```
sepal length (cm)    5.1
sepal width (cm)     3.5
petal length (cm)    1.4
petal width (cm)     0.2
Name: 0, dtype: float64
```

Predictions

In [7]:
```
predictions = []
Y_test = test['Species']
X_test = test.drop('Species',axis = 1)
for row in range(len(X_test)):
    pred = clf.predict(X_test.iloc[row,:])
    predictions.append(pred)
```

Accuracy

In [8]:
```
from sklearn.metrics import accuracy_score
accuracy_score(Y_test,predictions)
```

Out[8]:
```
0.9666666666666667
```

In [15]:
```
from sklearn.neighbors import KNeighborsClassifier
```

```
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X,y)
pred1=neigh.predict(X_test)
accuracy_score(Y_test,pred1)
```

Out[15]: 0.9666666666666667

In [14]:
```
from sklearn.neighbors import KNeighborsClassifier
neigh1 = KNeighborsClassifier(n_neighbors=5)
neigh1.fit(X,y)
pred2=neigh1.predict(X_test)
accuracy_score(Y_test,pred2)
```

Out[14]: 0.9666666666666667

In [18]:
```
neigh2 = KNeighborsClassifier(n_neighbors=7)
neigh2.fit(X,y)
pred3=neigh2.predict(X_test)
accuracy_score(Y_test,pred3)
```

Out[18]: 0.9333333333333333

In [19]:
```
neigh3 = KNeighborsClassifier(n_neighbors=9)
neigh3.fit(X,y)
pred4=neigh3.predict(X_test)
accuracy_score(Y_test,pred4)
```

Out[19]: 0.9666666666666667