In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
from sklearn.preprocessing import normalize
```

In [2]:
```
data= pd.read_csv('C:\\Users\Dell\Downloads\heart1.csv')
data.head()
```

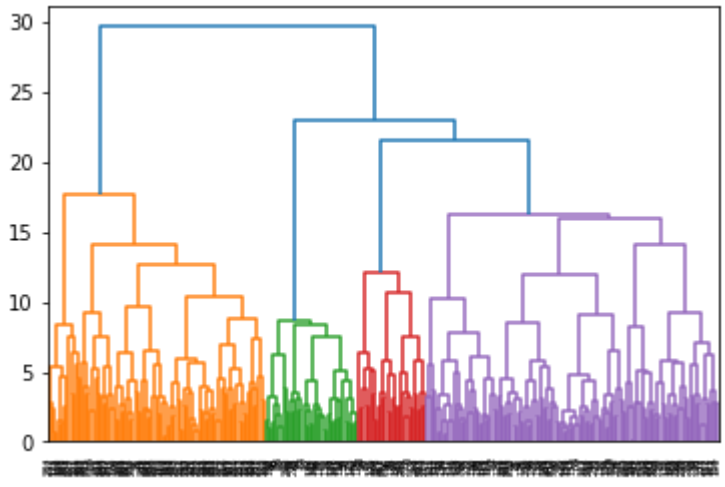
Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [3]:
```
X = normalize(data.iloc[:,:-1])
X=sc.fit_transform(X)
```

Hierarchical clustering

In [4]:
```
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))
```

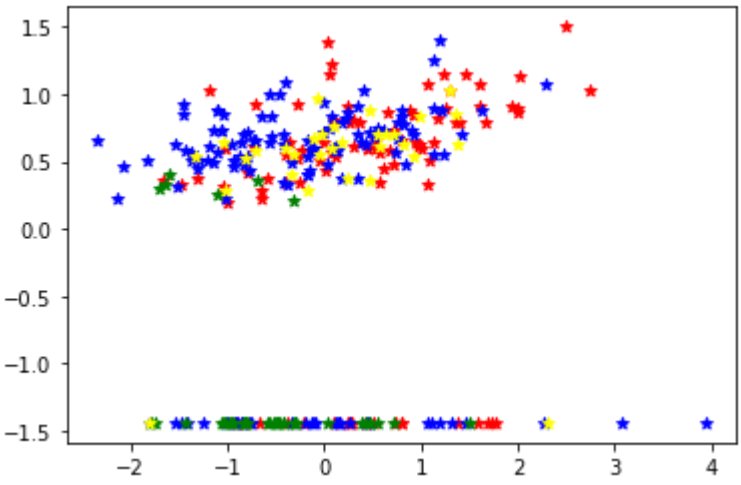


In [5]:
```
model = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward
model.fit(X)
labels = model.labels_
```

In [6]:
```
labels
```

Out[6]:
```
array([3, 1, 1, 1, 2, 0, 2, 1, 3, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 0,
       1, 3, 1, 2, 3, 1, 2, 3, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 1, 0, 2,
       1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 0, 3, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 3, 0, 3, 1, 1, 1, 2, 3, 2, 2, 1, 3,
       1, 2, 3, 1, 1, 2, 1, 0, 2, 3, 2, 3, 1, 1, 1, 3, 1, 1, 3, 0, 2, 2,
       0, 3, 2, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 2, 1, 1, 1, 2, 1, 0, 1, 1,
       2, 1, 2, 2, 3, 3, 0, 0, 2, 2, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 0, 0, 0, 0, 0, 3, 1, 1, 1, 0, 0,
       3, 2, 0, 0, 0, 0, 2, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 3,
       0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 3, 3, 0, 3, 0, 3,
       2, 0, 3, 0, 0, 0, 0, 0, 1, 1, 1, 3, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 2, 0, 1, 1, 0, 3, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 3, 1, 1, 0, 1, 0, 1, 0, 1, 2, 0, 0, 0, 3, 1, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1], dtype=int64)
```

In [7]:
```
plt.scatter(X[labels==0, 0], X[labels==0, 1], marker='*', color='red')
plt.scatter(X[labels==1, 0], X[labels==1, 1], marker='*', color='blue')
plt.scatter(X[labels==2, 0], X[labels==2, 1], marker='*', color='green')
plt.scatter(X[labels==3, 0], X[labels==3, 1], marker='*', color='yellow')
plt.show()
```



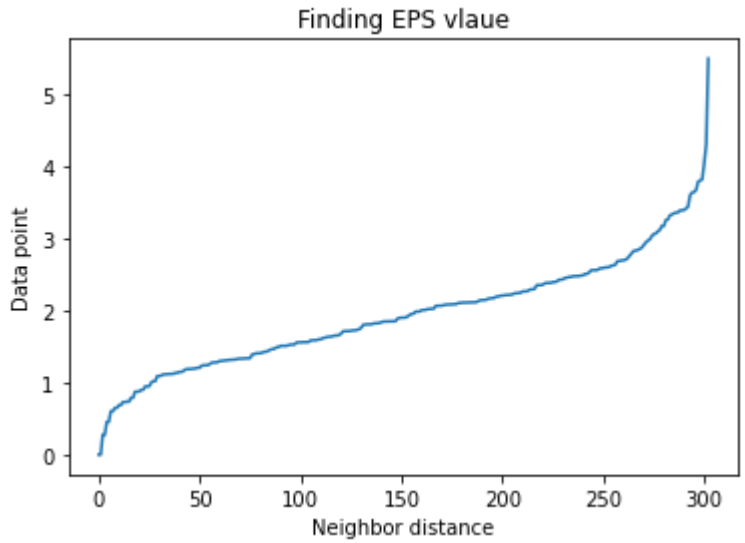
Advantages are

1. No apriori information about the number of clusters required.
2. Easy to implement

DBSCAN

In [8]:
```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=2)
nbrs = neigh.fit(X)
distances, indices = nbrs.kneighbors(X)
```

In [9]:
```
distance=np.sort(distances,axis=0)
distance=distance[:,1]
plt.plot(distance)
plt.xlabel("Neighbor distance")
plt.ylabel("Data point")
plt.title("Finding EPS vlaue")
plt.show()
```

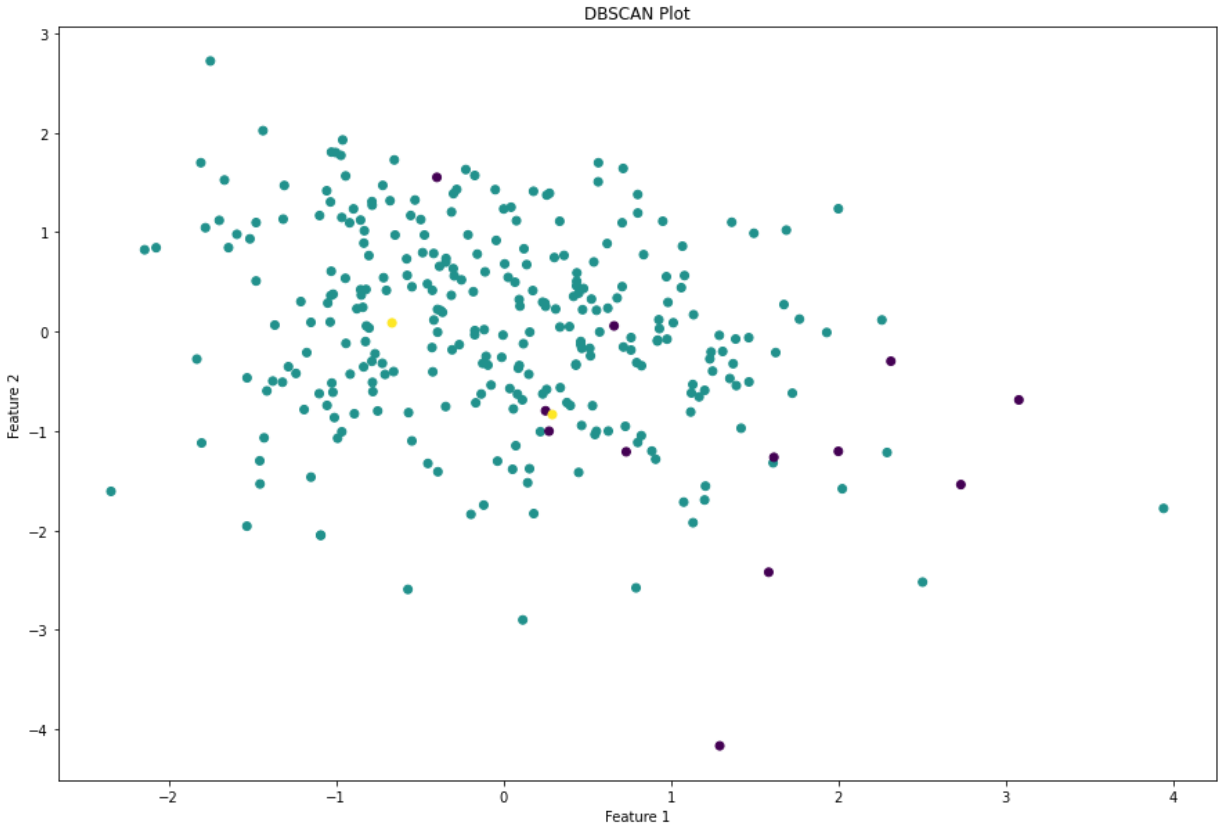


In [10]:
```
clustering = DBSCAN(eps=3.4, min_samples=3).fit(X)
clustering.labels_
clustering
```

Out[10]:
```
DBSCAN(eps=3.4, min_samples=3)
```

In [11]:
```
dbscan = DBSCAN(eps=3.4, min_samples = 2)
clusters = dbscan.fit_predict(X)
```

In [12]:
```
plt.figure(figsize=(15,10))
X=pd.DataFrame(X)
plt.scatter(X.iloc[:,0], X.iloc[:,4], c=clusters)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("DBSCAN Plot")
plt.show()
```



DBSCAN is particularly well suited for problems which require:

1. Minimal domain knowledge to determine the input parameters (i.e. K in k-means and Dmin in hierarchical clustering)
2. Discovery of clusters with arbitrary shapes
3. Good efficiency on large databases

In [ ]: