Bone Supplement Market Segmentation

Team 1

Aquib Farhaan Hussain

Ankur Bhattacharjee

24/7/21

Github link: https://github.com/aquib97/Bone-Market-Segmentation
Github link: https://github.com/ankur302/Bone-Supplement-Segmentation

Abstract:

The outlook for children with bone problems varies greatly. Conditions that cause bone problems range from mild to severe in their effect on the body. Once the child has been diagnosed, the doctor will share more details about treatment and prognosis based on your child's specific case. Therefore, supplements plays a critical role in health but its role in the deficiency or extremes in bone is incompletely understood.

The intent of the report is to represent the work on segmentation of Supplements market with a complete understanding of region-based states of India specifying district level. The criteria are well achieved by using Unsupervised based Machine learning models to represent the clusters of data and evaluate the business implementation.

Data Source:

The dataset used for market segmentation is extracted from India - Annual Health Survey(AHS) 2012-13. The survey was conducted in Empowered Action Group (EAG) states Uttarakhand, Rajasthan, Uttar Pradesh, Bihar, Jharkhand, Odisha, Chhattisgarh & Madhya Pradesh and Assam.

As the survey contains vast data focusing on reliability, representation with respect to core indicators. The meta-data prepared is a subset from the survey for this report which will be focused to clarify problem statement on systematic segmentation.

The total number is records undertaken is: **284**

Data Insertion:

The data is in a csv file format “Dataset.csv” and the following is code are run on google colab.

Before importing the data the package libraries used are:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn import metrics
from sklearn.metrics import accuracy_score
import numpy as np
```

Importing the data using pandas library:

```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Feynnlabs/Dataset.csv")
```

The respective features in the dataset are:

1. State -- Assam, Bihar, Chhattisgarh, Jharkhand, Madhya-Pradesh, Odisha, Rajasthan, Uttar-Pradesh, Uttarakhand.

2. District -- (All districts records with respect to states)

3. Population

4. Children between 1 to 2 years

5. Total_Arthritis_Patient -- Rheumatoid arthritis contributes to joint pain and swelling, and cartilage damage, which then leads to erosion into the bone around the around the joint. Inflammation also affects bones, including contributing to the loss of bone mineralization.

6. Arthritis_Patient_Male.

7. Arthritis_Patient_Female

8. Children_Vaccinated

9. Children_Immunized

10. Children_Vaccinated_at_Birth

11. Children_Not_Vaccinated

12. Total_Children_Weighted

13. Children_Weighted_less_than_2.5_Kg

14. Women_Aged_15_49_years

15. Children_Breastfed_Total

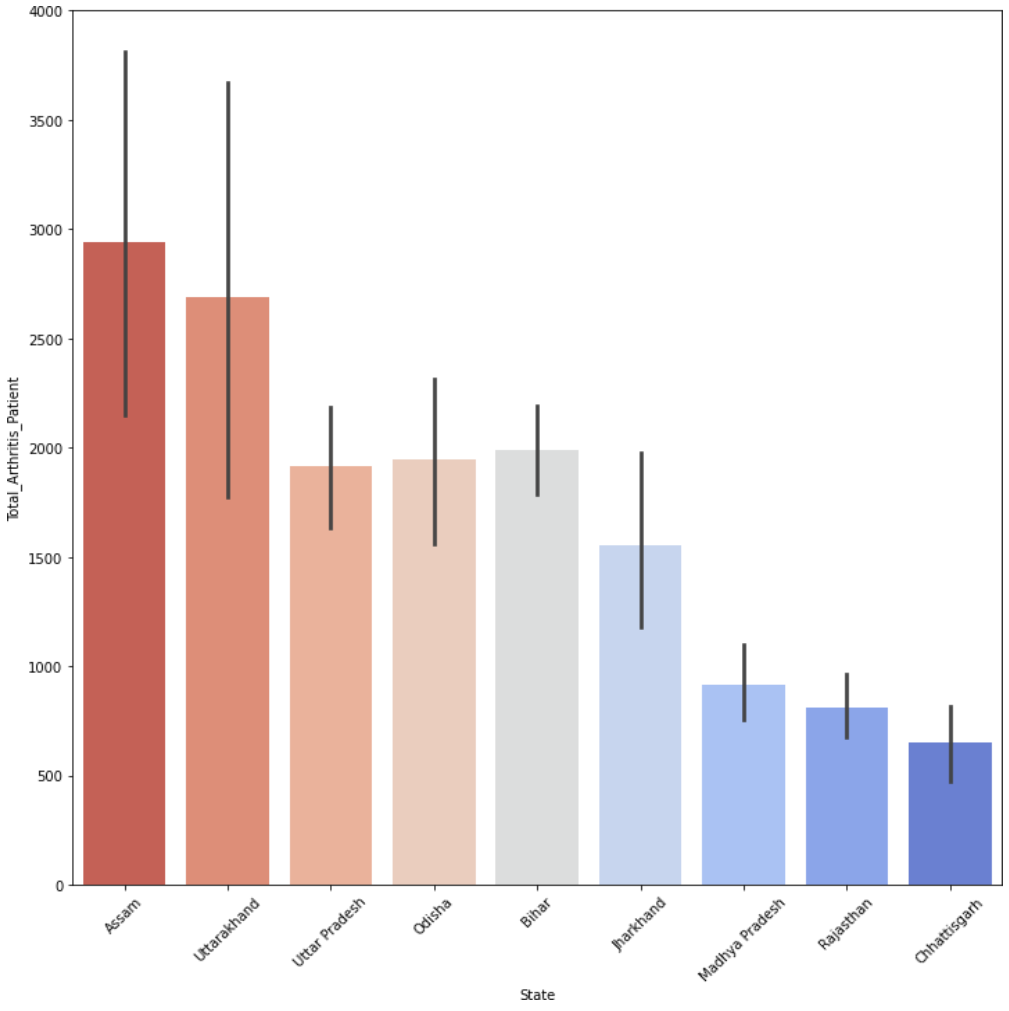
16. Children_Received_Foods_Other_Than_Breast_Milk_Total

17. Children_Received_Ifa_Tablets_Syrup_Last_3_Months_Total.

Exploratory data Analysis:

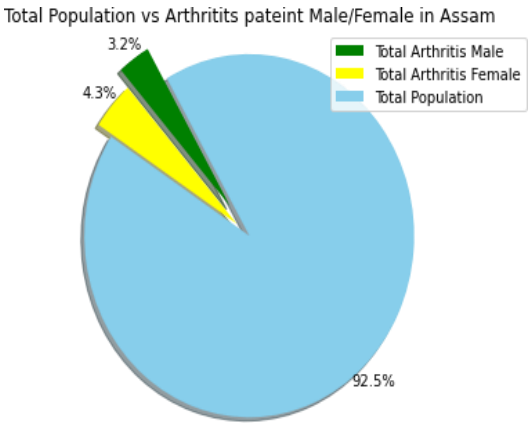
Total Population vs Total Arthritis patients

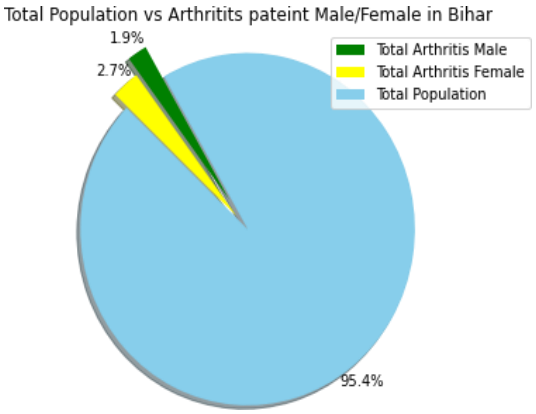
```
plt.figure(figsize=(12,12))
sns.barplot(x='State',data=data.sort_values("Total_Arthritis_Patient",ascending=False
),y="Total_Arthritis_Patient",palette='coolwarm_r')
plt.xticks(rotation=45)
plt.show()
```



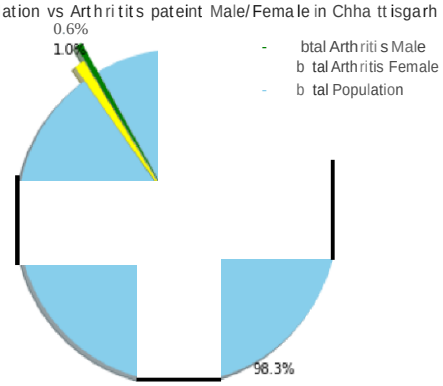
Total Population vs Arthritits pateint Male/Female

```
for state in data.State.unique():
  val1 = data[data['State']== state].Arthritis_Patient_Male.sum()/data[data['State']== st
ate].Population.sum()
  val2 = data[data['State']== state].Arthritis_Patient_Female.sum()/data[data['State']==
 state].Population.sum()
  x = [val1*100,val2*100,(1-val1-val2)*100]
  labels = ['Total Arthritis Male','Total Arthritis Female', 'Total Population']
  colors = ['green','yellow', 'skyblue']
  explode = (0.2, 0.1,0)
  plt.pie(x, colors=colors,explode=explode, shadow=True,startangle=120, pctdistance
=1.1,labeldistance=0.6, autopct='%1.1f%%')
  plt.legend(labels, loc="upper right")
  plt.axis('equal')
  plt.tight_layout()
  plt.title('Total Population vs Arthritits pateint Male/Female in {}'. format(state))
  plt.show()
```

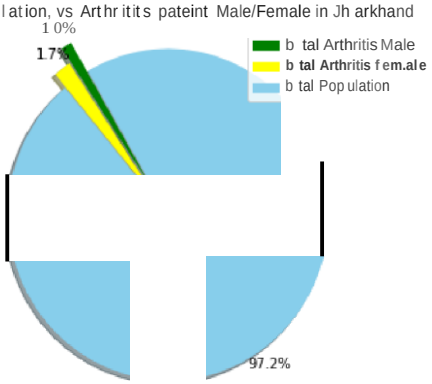




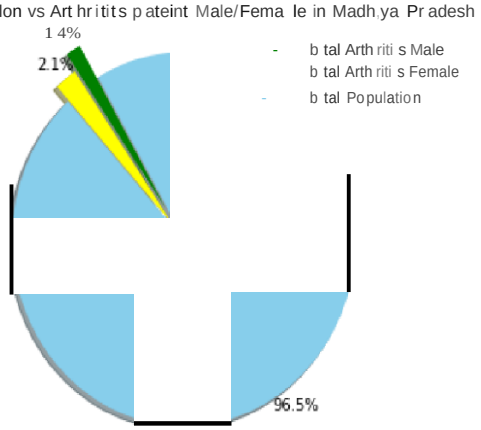
Total Popu lation vs Arthritits pateint Male/Fema le in Chha ttisgarh



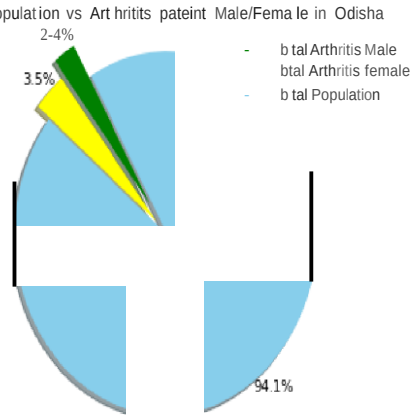
Total Pop'Ul ation, vs Arthr itits pateint Male/Female in Jh arkhand



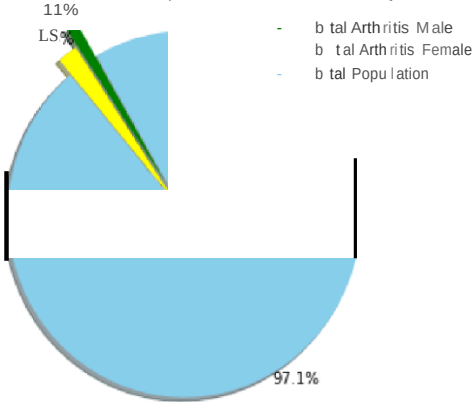
I otal Pop ulatilon vs Art hr itits p ateint Male/Fema le in Madh.ya Pr adesh



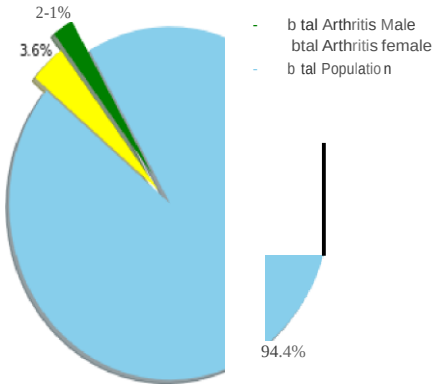
Total Population vs Art hritits pateint Male/Fema le in Odisha

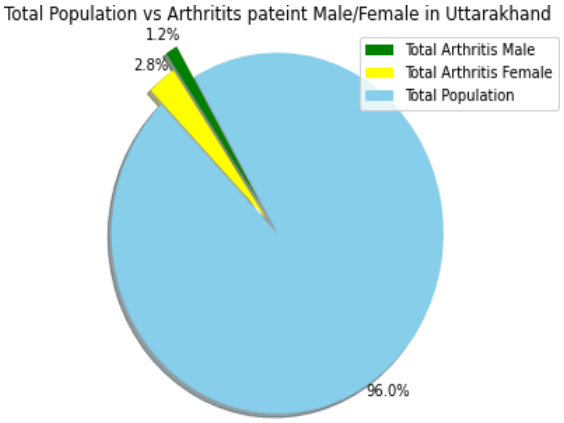


Total Popula t ion vs Arth ri tits pateint Male/f em ale in Raj asthan



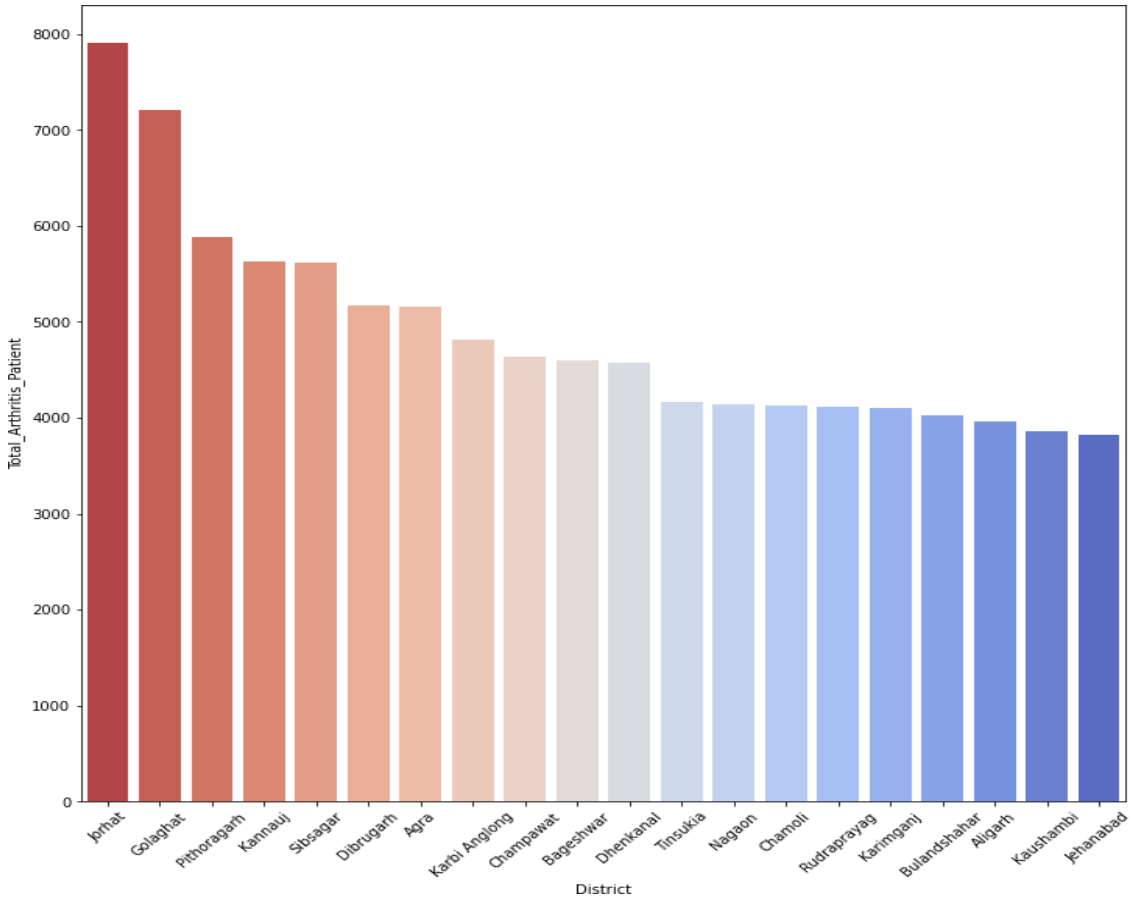
Total Populatio n vs Art h ri tits pateint Male/Female in Uttar Prades h





Top 20 districts to have most Arthritis Patients

```
#New data
new_data= data.sort_values(["Total_Arthritis_Patient"], axis=0,ascending=[False])
#Plot graph
plt.figure(figsize=(12,12))
sns.barplot(x="District",y='Total_Arthritis_Patient',data=new_data[:20],palette="cool warm_r")
plt.xticks(rotation=45)
plt.show()
```



Observations:

- From the first figures we see that **Assam, Bihar, Odisha, Uttar Pradesh,** has most number of Arthritis patients.
- From the 2nd figure we conclude that Female population is most affected by Arthritis compared to Male population
- 3rd figure shows the top 20 Districts with most Arthritis patients.

From the above analysis we understand that the above mentioned State along with top 20 districts can give us a good market hold for our supplement. Our target gender could possibly be women population as they are mostly affected by Arthritis.

Data Pre-processing:

In any Machine Learning process, Data Pre-processing is that step in which the data gets transformed, or Encoded, to bring it to such a state that now the machine can easily parse it. In other words, the features of the data can now be easily interpreted by the algorithm.

Transformed Features:

1. State – It is in the form nominal categorical variable. Therefore, One hot encoding is preformed to create new columns as a dummy variables filled up with zeros and ones (1 meaning TRUE, 0 meaning FALSE).

```
df=pd.get_dummies(data=df,columns=["State"],drop_first=True)
```

2. Population – This categorical feature has all distinct values, hence the values are encoded as per the number of counts in an index.

```
#enumerate data
dist={k:i for i,k in enumerate(df.District)}
#map data
df.District=df.District.map(dist)
```

Standard scaling:

Now all the variables are in numerical format with data type as either integer or float. For a machine learning algorithms to work better, the features should be on a relatively similar scale and close to normally distributed. Scale generally means to change the range of the values.

In this case StandardScaler is used where a feature is standardized by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by the standard

deviation. StandardScaler results in a distribution with a standard deviation equal to 1. The variance is equal to 1 also, because *variance = standard deviation squared*. And 1 squared = 1. StandardScaler makes the mean of the distribution 0. About 68% of the values will lie be between -1 and 1.

```
from sklearn.preprocessing import StandardScaler
#define scalar
scalar=StandardScaler()
#transform data
data=scalar.fit_transform(df)
```

The resultant dataset return is in the form of numpy array.

Clustering:

Clustering is an unsupervised approach which finds a structure/pattern in a collection of unlabeled data. A cluster is a collection of objects which are “similar” amongst themselves and are “dissimilar” to the objects belonging to a different cluster. The goal of clustering is to determine the intrinsic groups in unlabeled data. For customer segmentation the cluster of customers based on their purchases, their activity on your website, and so on is useful to understand who your customers are and what they need, so you can adapt your products and marketing campaigns to each segment.

In this report, the problem statement is approached by using clustering algorithm for modelling and showing the segmentation of the dataset. Here the algorithms used are:

1. K-Means Clustering.
2. Hierarchical Clustering.
3. DBSCAN(Density Based Spatial Clustering of Applications with Noise).

Evaluation of Clustering:

Aspects of cluster validation

- **External:** Compare your cluster to the ground truth.
- **Internal:** Evaluating the cluster without reference to external data.
- **Reliability:** The clusters are not formed by chance(randomly)- some statistical framework can be used.

The methods use to measure the quality of clusters without external references. There are two aspects to it.

- **Cohesion:** How closely the objects in the same cluster are related to each other. It is the within-cluster sum of squared distances. It is the same metric that we used to calculate for the K-Means algorithm.

$$WCSS=\sum\sum(x-mi)2WCSS=\sum\sum(x-mi)2$$

- **Separation:** How different the objects in different clusters are and how distinct a well-separated cluster is from other clusters. It is the between cluster sum of squared distances.

$$BSS=\sum Ci(m-mi)2BSS=\sum Ci(m-mi)2$$

Where **C** is the size of the individual cluster and **m** is the centroid of all the data points.

Note: BSS+WSS is always a constant.

The silhouette can be calculated as:

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}}$$

Where a(x) is the avarage distance of x from all the other points in the same cluster and b(x) is the avarage distance of x from all the other points in the other clusters.

And the Silhoeutte coefficient is given by:

$$SC=1/N\sum(x)$$

Unsupervised ML models:

K-Means Clustering

K-Means is a clustering approach in which the data is grouped into K distinct non-overlapping clusters based on their distances from the K centres. The value of **K** needs to be specified first and then the algorithm assigns the points to exactly one cluster.

Theory

The theory discussed above can be mathematically expressed as:

- Let C1, C2, Ck be the K clusters
- Then we can write: $C1UC2UC3U\ldots UCk$ = {1,2,3,…,n} i.e., each data-point has been assigned to a cluster.
- Also,

$$C_k \cap C_{k'} = \emptyset \text{ for all } k \neq k'.$$

  This means that the clusters are non-overlapping.

- The idea behind the K-Means clustering approach is that the within-cluster variation amongst the point should be minimum. The within-cluster variance is denoted by: W(Ck).Hence, according to the statement above, we need to minimize this variance for all the clusters. Mathematically it can be written as:

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W(C_k) \right\}.$$

- The next step is to define the criterion for measuring the within-cluster variance. Generally, the criterion is the Euclidean distance between two data points.

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2,$$

- The above formula says that we are calculating the distances between all the point in a cluster, then we are repeating it for all the K clusters (That's why two summation signs) and then we are dividing it by the number of observation in the clusters (Ck is the number of observations in the Kth cluster) to calculate the average.

So, ultimately our goal is to minimize:

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}.$$

The following algorithm steps are used to solve this problem.

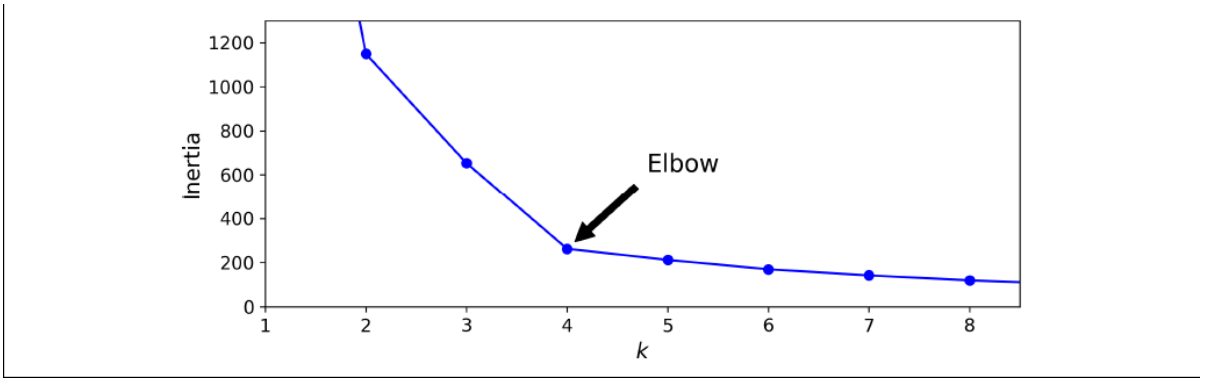
Algorithm Steps:

1. Randomly assign K centres.

2. Calculate the distance of all the points from all the K centres and allocate the points to cluster based on the shortest distance. The model's *inertia* is the mean squared distance between each instance and its closest centroid. The goal is to have a model with the lowes intertia.
3. Once all the points are assigned to clusters, recompute the centroids.
4. Repeat the steps 2 and 3 until the locations of the centroids stop changing and the cluster allocation of the points becomes constant

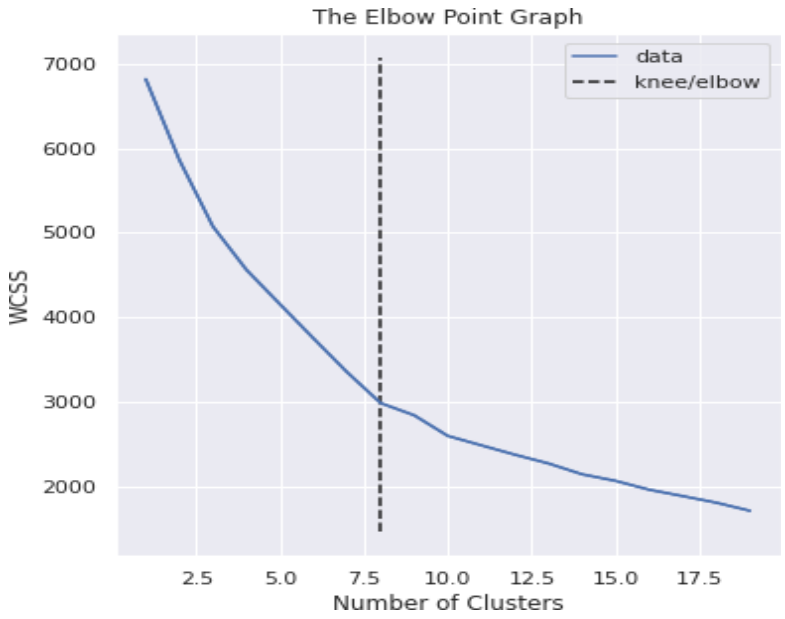
The Elbow-Method-(Finding the Optimal Number of Clusters)

This method is based on the relationship between the within-cluster sum of squared distances (WCSS or Inertia) and the number of clusters. It is observed that first with an increase in the number of clusters WCSS decreases steeply and then after a certain number of clusters the drop in WCSS is not that prominent. The point after which the graph between WCSS and the number of clusters becomes comparatively smother is termed as the elbow and the number of cluster at that point are the optimum number of clusters as even after increasing the clusters after that point the variation is not decreasing by much i.e., we have accounted for almost all the dissimilarity in the data. An elbow-curve looks like:




```
from sklearn.cluster import KMeans
from kneed import KneeLocator
#Data
X = data.iloc[:,:].values

#Cluster by wcss method
wcss = []
for i in range(1,20):
  kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
  kmeans.fit(X)
  wcss.append(kmeans.inertia_)

#Plot graph
kneedle = KneeLocator(range(1,20),wcss,curve='convex', direction='decreasing')
kneedle.plot_knee()
sns.set()
plt.plot(range(1,20), wcss)
plt.title('The Elbow Point Graph')
```

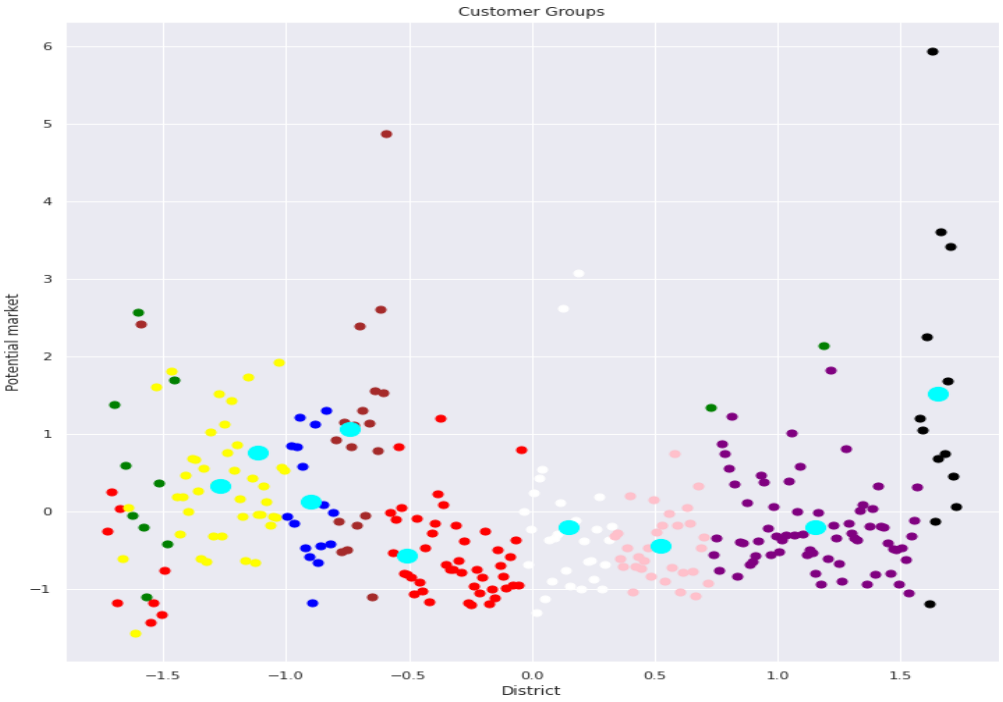
```
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```




```
print(kneedle.elbow)
```

Output: 8

```
kmeans = KMeans(n_clusters=9, init='k-means++', random_state=0)

# return a label for each data point based on their cluster
Y = kmeans.fit_predict(X)
#plot graph
plt.figure(figsize=(12,12))
plt.scatter(X[Y==0,0], X[Y==0,1], s=50, c='green', label='Cluster 1')
plt.scatter(X[Y==1,0], X[Y==1,1], s=50, c='red', label='Cluster 2')
plt.scatter(X[Y==2,0], X[Y==2,1], s=50, c='yellow', label='Cluster 3')
plt.scatter(X[Y==3,0], X[Y==3,1], s=50, c='Purple', label='Cluster 4')
plt.scatter(X[Y==4,0], X[Y==4,1], s=50, c='blue', label='Cluster 5')
plt.scatter(X[Y==5,0], X[Y==5,1], s=50, c='black', label='Cluster 6')
plt.scatter(X[Y==6,0], X[Y==6,1], s=50, c='white', label='Cluster 7')
plt.scatter(X[Y==7,0], X[Y==7,1], s=50, c='pink', label='Cluster 8')
plt.scatter(X[Y==8,0], X[Y==8,1], s=50, c='Brown', label='Cluster 9')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=200, c='cyan'
, label='Centroids')
plt.title('Customer Groups')
plt.xlabel('District')
plt.ylabel('Potential market')
plt.show()
```



The above graph shows the clusters segment by K-Means with along the centroids of each clusters.

Hierarchical clustering

One main disadvantage of K-Means is that it needs us to pre-enter the number of clusters (K). Hierarchical clustering is an alternative approach which does not need us to give the value of K beforehand and also, it creates a beautiful tree-based structure for visualization.

Here, we are going to discuss the bottom-up (or Agglomerative) approach of cluster building. We start by defining any sort of similarity between the data-points. Generally, we consider the Euclidean distance. The points which are closer to each are more similar than the points which re farther away. The Algorithms starts with considering all points as separate clusters and then grouping pints together to form clusters.

Algorithm Steps:

1. Begin with n observations and a measure (such as Euclidean distance) of all the $n(n-1)/2$ pairwise dissimilarities(or the Euclidean distances generally). Treat each observation as its own cluster. Initially, we have n-clusters.
2. Compare all the distances and put the two closest points/clusters in the same cluster. The dissimilarity (or the Euclidean distances) between these two clusters indicates the height in the dendrogram at which the fusion line should be placed.
3. Compute the new pairwise inter-cluster dissimilarities (or the Euclidean distances) among the remaining clusters.
4. Repeat steps 2 and 3 till we have only one cluster left.

The idea behind linkage clustering, or hierarchical clustering, is to put things that are close together into the same cluster. Initially, hierarchical clustering starts out with clusters consisting of individual points.

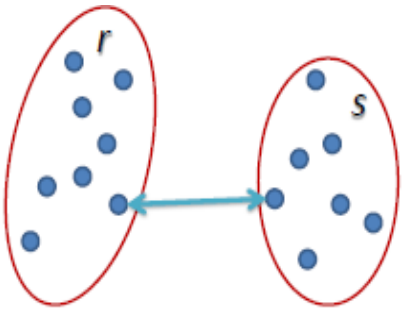
Later, it compares clusters with each other and merges the two "closest" clusters.

Since clusters are sets of points, there are many different kinds of linkage methods:

- **Single Linkage**: cluster distance = smallest pairwise distance
- **Complete Linkage**: cluster distance = largest pairwise distance
- **Average Linkage**: cluster distance = average pairwise distance.

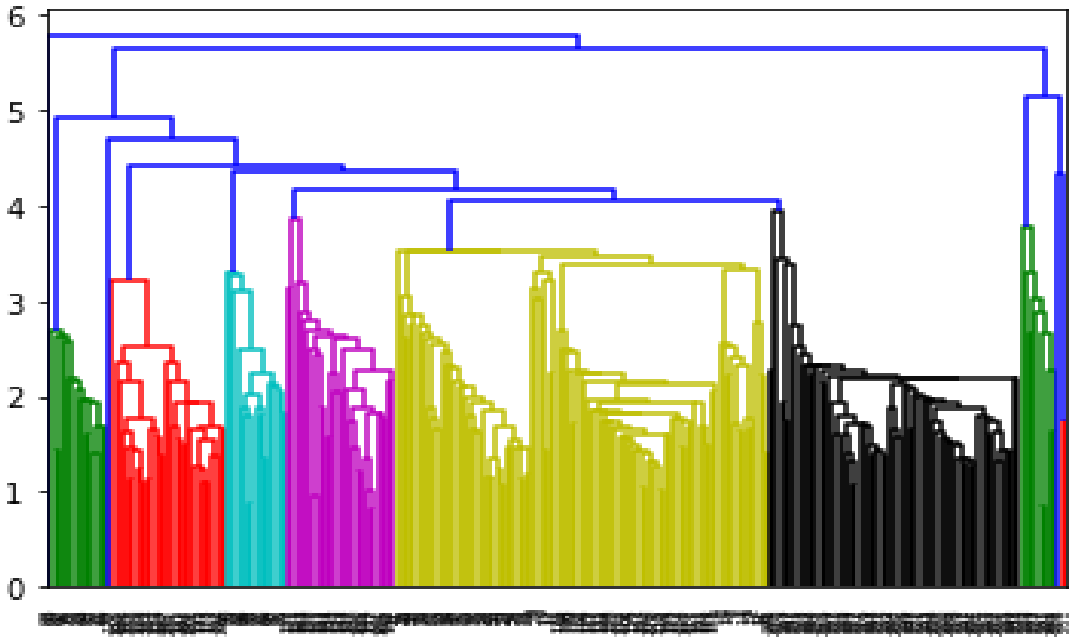
I. Single Linkage:

In single linkage hierarchical clustering, the distance between two clusters is defined as the shortest distance between two points in each cluster. For example, the distance between clusters “r” and “s” to the left is equal to the length of the arrow between their two closest points.



$$L(r,s) = \min(D(x_{ri}, x_{sj}))$$

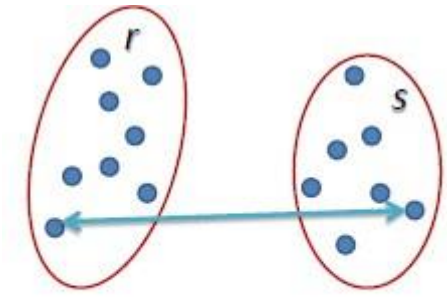
```
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cut_tree

# Single linkage:
mergings = linkage(data, method="single", metric='euclidean')
dendrogram(mergings)
plt.show()
```



Data points paired in dendrogram

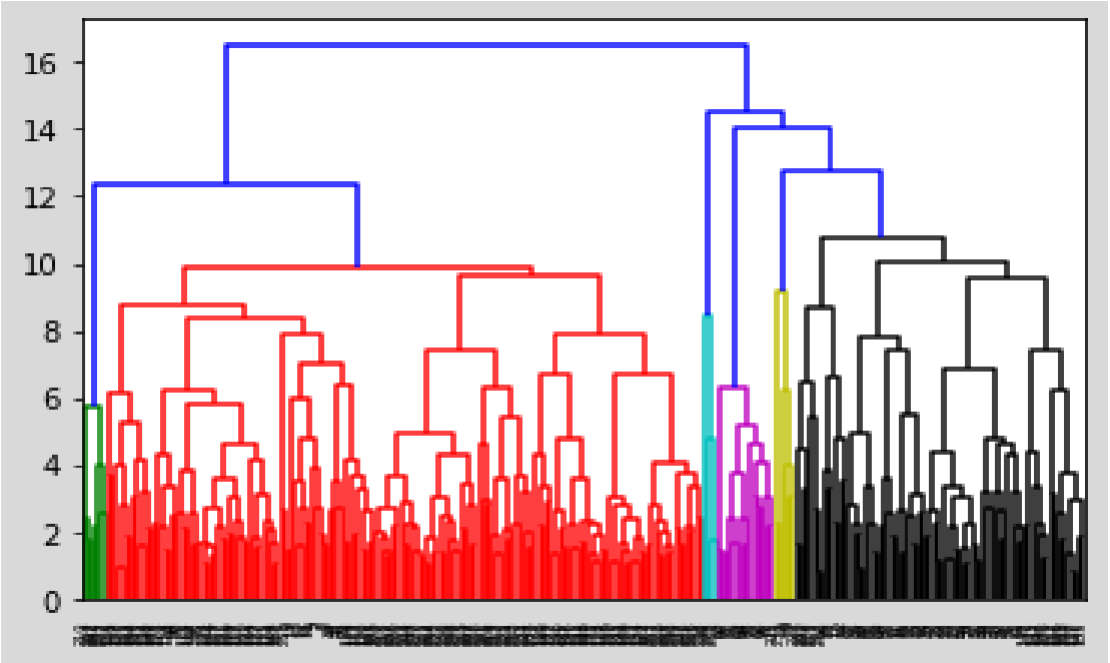
II. Complete Linkage

In complete linkage hierarchical clustering, the distance between two clusters is defined as the longest distance between two points in each cluster. For example, the distance between clusters “r” and “s” to the left is equal to the length of the arrow between their two furthest points.



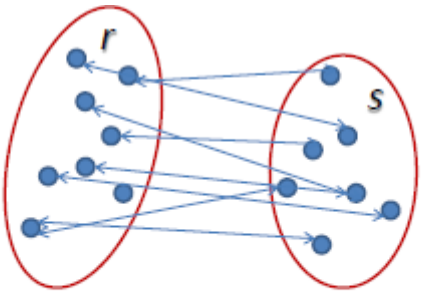
$$L(r,s) = \max(D(x_{ri}, x_{sj}))$$

```
# Complete linkage

mergings = linkage(data, method="complete", metric='euclidean')
dendrogram(mergings)
plt.show()
```



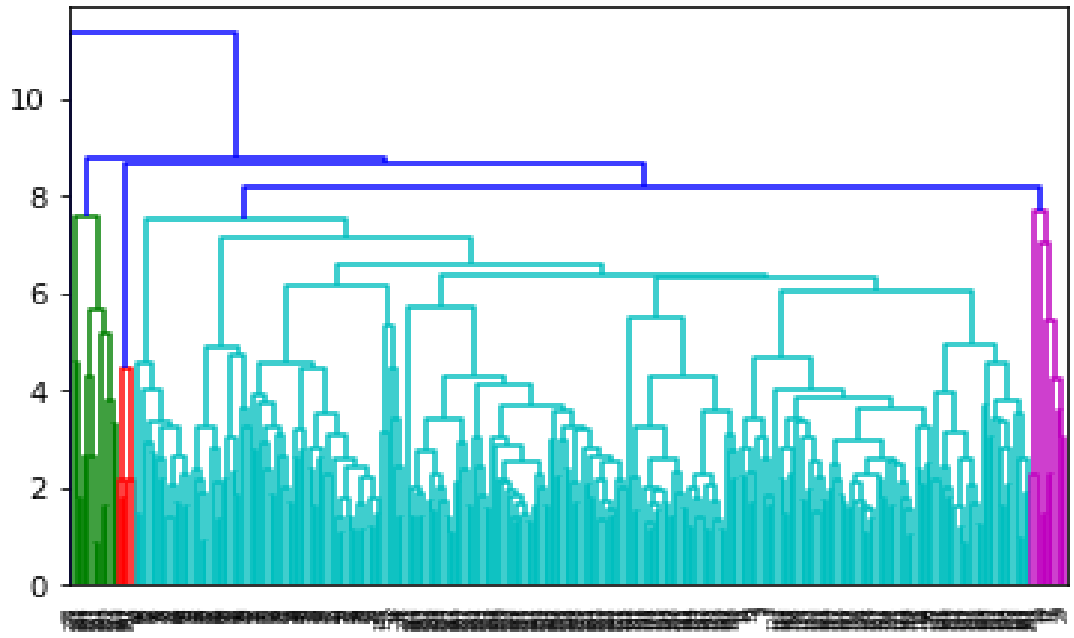
III. Average Linkage:

In average linkage hierarchical clustering, the distance between two clusters is defined as the average distance between each point in one cluster to every point in the other cluster. For example, the distance between clusters “r” and “s” to the left is equal to the average length each arrow between connecting the points of one cluster to the other.



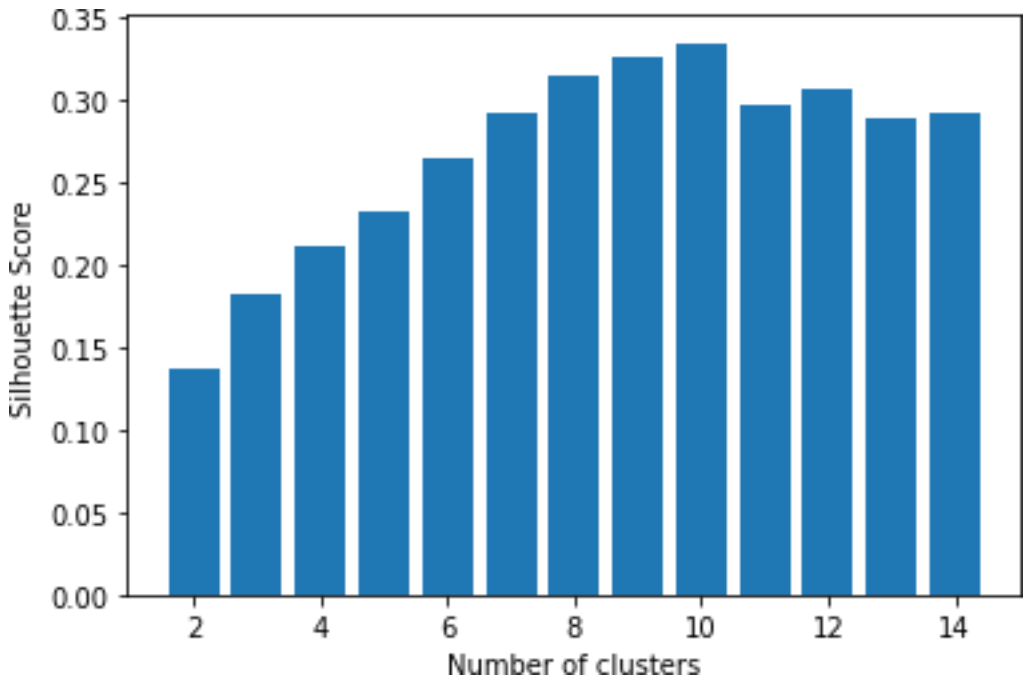
$$L(r,s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} D(x_{ri}, x_{sj})$$

```
# Average linkage
mergings = linkage(data, method="average", metric='euclidean')
dendrogram(mergings)
plt.show()
```



In order to get the best number of cluster for model fitting, Silhouette score as an evaluation criteria.

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc

#Silhoute score
silhouette_scores = []
for n_cluster in range(2, 15):
    silhouette_scores.append(silhouette_score(data, AgglomerativeClustering
(n_clusters = n_cluster).fit_predict(data)))

# Plotting a bar graph to compare the results
k = [2, 3, 4, 5, 6,7,8,9,10,11,12,13,14]
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 10)
plt.ylabel('Silhouette Score', fontsize = 10)
plt.show()
```



Number of Clusters=10 shows the highest score.

```
#Predict data
hc = AgglomerativeClustering(n_clusters = 10, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(data)
#Label
y_hc
```

Output: `array([2, 2, 1, 2, 2, 2, 8, 2, 8, 2, 8, 1, 8,. . . . . , 9, 9])`

DBSCAN(Density Based Spatial Clustering of Applications with Noise)

It is an unsupervised machine learning algorithm.This algorithm defines clusters as continuous regions of high density.

Some definitions first:

Epsilon: This is also called eps. This is the distance till which we look for the neighbouring points.

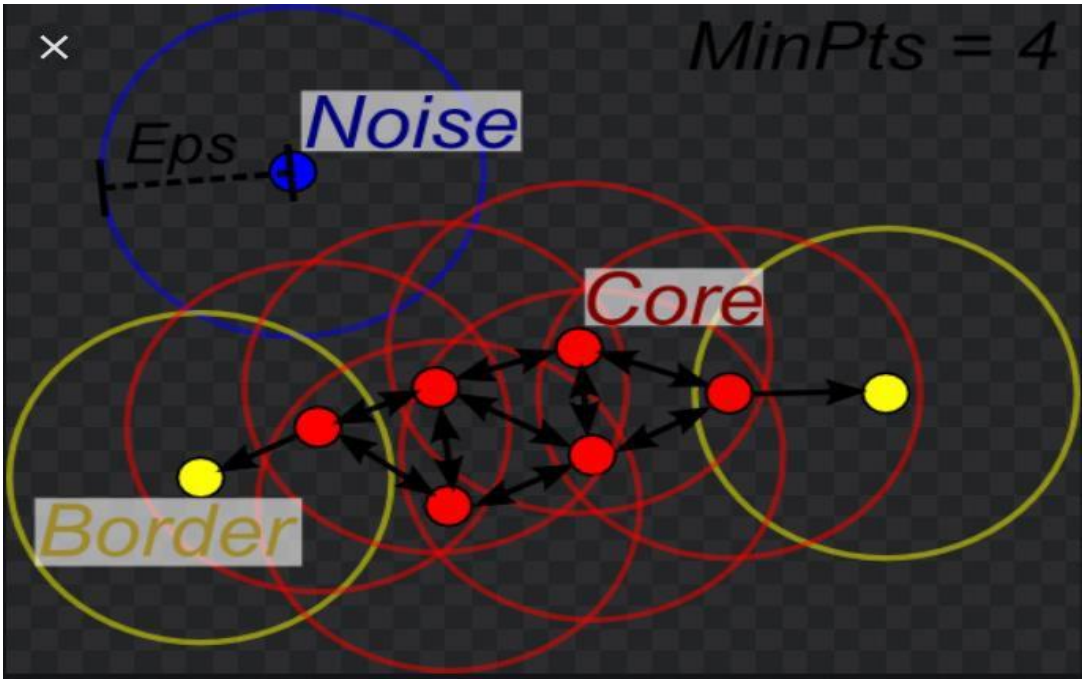
Min_points: The minimum number of points specified by the user.

Core Points: If the number of points inside the *eps radius* of a point is greater than or equal to the *min_points* then it's called a core point.

Border Points: If the number of points inside the *eps radius* of a point is less than the *min_points* and it lies within the *eps radius* region of a core point, it's called a border point.

Noise: A point which is neither a core nor a border point is a noise point.

Let's say if the eps=1 and min_points =4



Algorithm Steps:

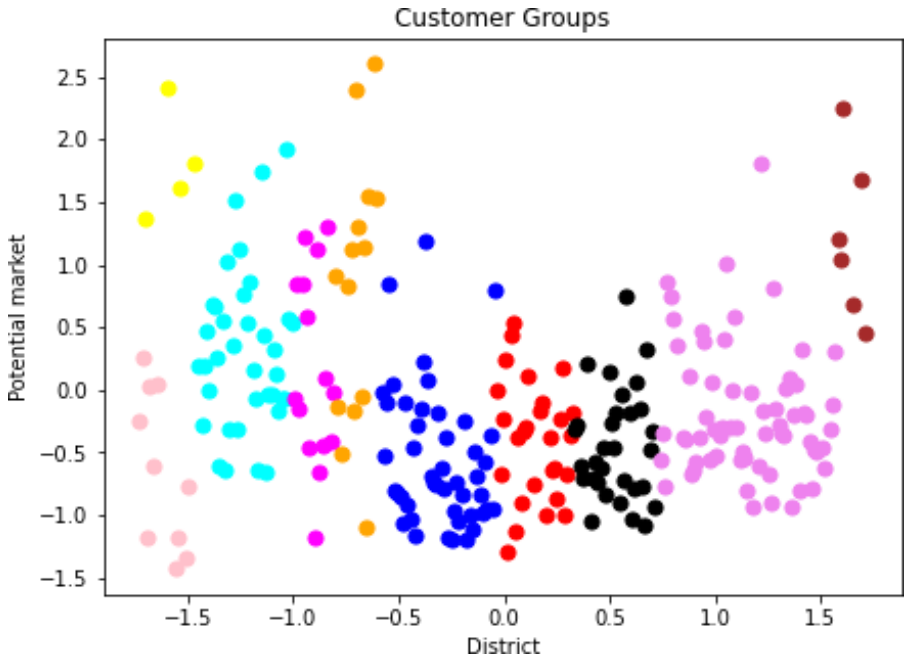
1. The algorithm starts with a random point in the dataset which has not been visited yet and its neighbouring points are identified based on the eps value.
2. If the point contains greater than or equal points than the min_pts, then the cluster formation starts and this point becomes a *core point*, else it's considered as noise. The thing to note here is that a point initially classified as noise can later become a border point if it's in the eps radius of a core point.
3. If the point is a core point, then all its neighbours become a part of the cluster. If the points in the neighbourhood turn out to be core points then their neighbours are also part of the cluster.
4. Repeat the steps above until all points are classified into different clusters or noises.

This algorithm works well if all the clusters are dense enough, and they are well separated by low-density regions.

```
from sklearn.cluster import DBSCAN

#Model fit
db=DBSCAN(eps=3,min_samples=4,metric='euclidean')
model=db.fit(data)

#label

label=model.labels_

#identifying the points which makes up our core points
sample_cores=np.zeros_like(label)
sample_cores[db.core_sample_indices_]=True

#Calculating the number of clusters
n_clusters=len(set(label))- (1 if -1 in label else 0)
print('No of clusters:',n_clusters)
```

Output: `No of clusters: 10`

```
#Predict data
y_means = db.fit_predict(data)

#Plotting Graph
plt.figure(figsize=(7,5))
plt.scatter(data[y_means == 0, 0], data[y_means == 0, 1], s = 50, c = 'pink')
plt.scatter(data[y_means == 1, 0], data[y_means == 1, 1], s = 50, c = 'yellow')
plt.scatter(data[y_means == 2, 0], data[y_means == 2, 1], s = 50, c = 'cyan')
plt.scatter(data[y_means == 3, 0], data[y_means == 3, 1], s = 50, c = 'magenta')
plt.scatter(data[y_means == 4, 0], data[y_means == 4, 1], s = 50, c = 'orange')
plt.scatter(data[y_means == 5, 0], data[y_means == 5, 1], s = 50, c = 'blue')
plt.scatter(data[y_means == 6, 0], data[y_means == 6, 1], s = 50, c = 'red')
plt.scatter(data[y_means == 7, 0], data[y_means == 7, 1], s = 50, c = 'black')
plt.scatter(data[y_means == 8, 0], data[y_means == 8, 1], s = 50, c = 'violet')
plt.scatter(data[y_means == 9, 0], data[y_means == 9, 1], s = 50, c = 'brown')

plt.title('Customer Groups')
plt.xlabel('District')
plt.ylabel('Potential market')
plt.show()
```



Total 10 cluster groups segmented using DBSCAN.

Conclusion:

The entire project is focused on Arthritis patients from all 9 states. The data is segmented with respect to district level. By performing Exploratory data analysis we can observe a clear picture of market understanding and put an estimation for future business opportunity. Clustering techniques are implanted to mark the division of the market prediction.

However we don't happen to find much information out of the rest of the data. Inclusion of Economic status, Accessibility to Healthcare data could possibly bring out more insights from the given dataset for better market segmentation.