A Short Monograph on Clustering

TO SERVE AS A REFRESHER FOR PGP-DSBA

Table of Contents

List of Figures

List of Equations

1. Introduction

In today’s world of information explosion, knowledge is derived by classifying information. Analytics borrows its strength by eliminating the random perturbations and identifying the underlying structure. Every customer who walks in a departmental superstore is different. No two persons will buy identical items in identical quantities. Customers who are young and single are likely to buy certain items whereas elderly couples are likely to buy certain other items. To promote sales by utilizing resources in a cost-effective manner, superstores are to understand the requirements of different groups of customers. Instead of a blanket advertisement policy, where all items are being advertised to all customers, a much more efficient way of increasing sales is to classify customers into mutually exclusive groups and to come up with a tailor-made advertising policy for each group.

Clustering lies at the core of marketing strategy. In recent times many other application areas of clustering have emerged. Clustering documents on the web helps to extract and classify information. Clustering of genes helps to identify various properties, including their disease carrying propensity. Clustering is a powerful tool for data mining and pattern recognition for image analysis.

Clustering is an unsupervised technique. The underlying assumption is that, the observed data is coming from multiple populations. To elaborate the marketing strategy example, it is assumed that distinct populations exist among the customer base for a supermarket. The difference among populations may not be based on demographics only. A set of complex characteristics based on demography, socio-economic strata and other conditions delineate the populations and they form a partition of the customers. Once the clusters are identified, they can be studied in a better manner and possibly different strategies are applied to garner more business from the targeted groups.

Objective of clustering is to form homogeneous partitions out of heterogeneous observations.

Clustering may be done in many ways, not all of them are optimally data driven. It is possible to define clusters based on age-groups alone, or based on total amount spent or visit frequency. In subsequent sections we will discuss data driven clustering procedures only. The rationale behind such procedures is to define homogenous partition of the data.

Clustering is an unsupervised learning technique to partition the data into homogeneous segments. Within a cluster the observations may be assumed to come from one single population. Observations belonging to different clusters are assumed to represent different populations.

2. Clustering, Formally

Clustering is an example of unsupervised learning. For unsupervised learning problems there does not exist any response. A predictive modelling problem is a supervised learning (see the monograph on Predictive Modelling) since it contains a designated response variable which depends on the set of predictors. In clustering problems, the underlying populations are not identified beforehand, neither the number of possible populations is known. It is assumed that the observed data is heterogeneous and partitioning it into multiple populations is expected to make the clusters homogeneous. In other words, observations belonging to a cluster are more similar among themselves and observations belonging to different groups are dissimilar. Clustering is a process to find meaningful structure and features in a set of observations based on given measures of similarity.

The following important points need to be clarified at the outset.

I. It is assumed that the observations are not generated from a single population. However, there is no concrete evidence that it is so.
II. The rationale behind clustering is that, when the data set is partitioned into multiple groups, the sum of variations within each group (within sum of square) is substantially smaller than the total variation in the data.
III. Since the number of populations present, *k*, is unknown, it is also determined from the data.
IV. **Once the different populations are identified, they must be treated separately. This implies that the set of defining parameters must not be identical. If any predictive model is developed, a separate model needs to be developed for each population**.

.

3. Methods of Clustering

Clustering aims to determine the intrinsic grouping among the data points. The only criterion for grouping is that, the observations belonging to the same cluster, are closer among themselves than observations belonging to different groups. Hence a measure for closeness or *similarity* between pairs of points need to be developed. In subsequent sections we will discuss various options to measure similarity. Depending on the choice of similarity, the pair whose similarity measure is more than that of a second pair, is assumed to be closer with respect to *that* measure of similarity compared to the second pair.

There are two primary approaches to clustering; namely hierarchical or agglomerative clustering and *k*-means clustering. In hierarchical clustering, the closest points are combined in a pairwise manner to form the clusters. It is an iterative procedure, where at every step of the iteration, two points or two clusters are combined to form a bigger cluster. At the end of the process all points are combined into a single cluster. The number of clusters is not pre-determined.

In the *k*-means clustering process, *k* denotes the number of clusters, which has to be pre-determined. Once *k* is fixed, the observations are allocated to one and only one cluster so that the closest points belong to one cluster. The cluster size is not controlled.

3.1 Measures of Distance

Since the clustering process is completely controlled by the distance between two points and distance between two clusters, it is paramount that the concept of distance is clear before we can move on to the actual clustering process.

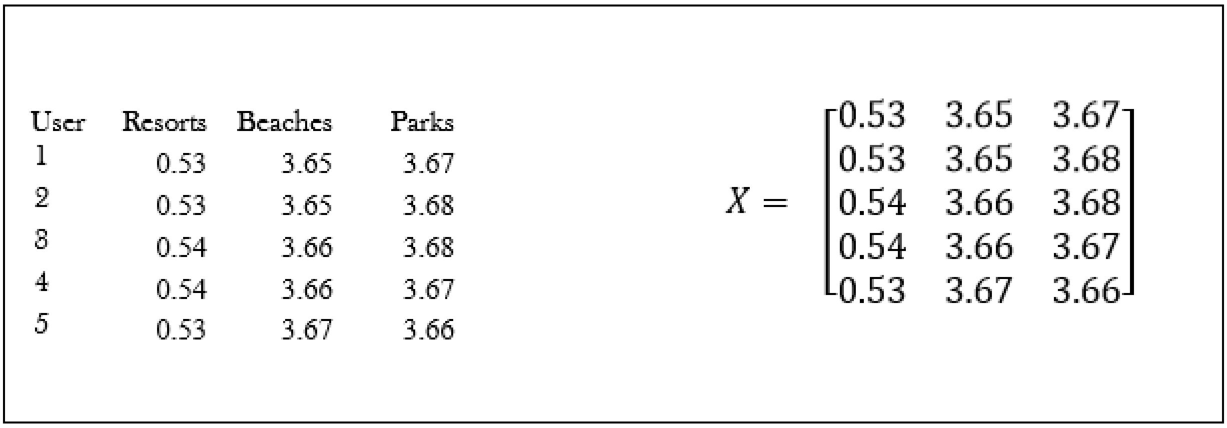
Clustering works only when the observations are multivariate. For univariate observations the concept of distance is trivial. We will also assume that the variables are only numeric. If the variables are categorical or mixed, this simple measure of distance will not work. A few alternative distance measures are given in Section 6.

Let us introduce matrix notation to describe the observations. Let X denote the data matrix of order *n x p*, i.e. there are *n* observations in the data and each observation has *p* dimensions. In the matrix each row denotes an observation and each column, one dimension or attribute.

$$\mathrm{X} = \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix}$$

Example: Let us consider a small data on Google user rating on various places of interest. Suppose there are 5 users and each one of them is giving rating on 3 tourist attractions. The rating may be any number between 1 and 5, where 1 is the lowest rating and 5 is the highest. (*Details of this data set are provided later*)

In Fig 1 below the left panel shows the observed data and the right panel shows the data arranged in matrix notation. Since there are 5 rows and 3 columns, the matrix is of dimension 5 *x* 3. Unless otherwise mentioned, each row represents one data point, each column represents one attribute. The (*i*, *j*)-th cell of a data matrix identifies the value of the *j*-th attribute on the *i*-th observation. If there is a missing value, that cell remains blank.



User	Resorts	Beaches	Parks
1	0.53	3.65	3.67
2	0.53	3.65	3.68
3	0.54	3.66	3.68
4	0.54	3.66	3.67
5	0.53	3.67	3.66

$$X = \begin{bmatrix} 0.53 & 3.65 & 3.67 \\ 0.53 & 3.65 & 3.68 \\ 0.54 & 3.66 & 3.68 \\ 0.54 & 3.66 & 3.67 \\ 0.53 & 3.67 & 3.66 \end{bmatrix}$$

Figure 1: Data Arrangement in Matrix Format

For the purpose of clustering, distance is defined between a pair of observations combining all the attributes. There are many possible distance measures. Distance is a non-negative quantity. Distance between two identical observations must be zero. Distance between two non-identical observations must always be positive.

Pairwise distance among a set of *n* *p*-dimensional observations can be arranged in a *n* x *n* symmetric (square) matrix whose principal diagonal is 0.

3.1.1 Common Measures of Distance

<u>Euclidean Distance</u>: Between two observations X_1 and X_2, each of dimension *p*, Euclidean distance is defined as

$$E_d\,(X_1, X_2) = \sqrt{\sum_{j=1}^{p}\left(X_{1j} - X_{2j}\right)^2}$$

Equation 1: Euclidean Distance

Example: Euclidean distance between Users 1 and 3 is

$$\sqrt{(0.53-0.54)^2+(3.65-3.66)^2+(3.67-3.68)^2} = \sqrt{0.0003} = 0.017$$

Equation 2: Calculating Euclidean Distance

Squared Euclidean Distance: It is the Euclidean distance without the square root and the distance between two observations X_1 and X_2 is defined as

$$\mathrm{SE_d}\,(X_1, X_2) = \sum_{j=1}^{p}\left(X_{1j} - X_{2j}\right)^2$$

Equation 3: Squared Euclidean Distance

Example: Squared Euclidean distance between Users 1 and 3 is 0.0003.

Manhattan Distance: This is also known as absolute value distance or L_1 distance. The Manhattan distance between two observations X_1 and X_2 is defined as

$$\mathrm{M_d}\,(X_1, X_2) = \sum_{j=1}^{p}\left|X_{1j} - X_{2j}\right|$$

Equation 4: Manhattan Distance

Example: Manhattan distance between Users 1 and 3 is 0.03.

Minkowski's Distance: This distance measure is a generalization of the Euclidean and Manhattan distance and is named after a German mathematician of the nineteenth century Hermann Minkowski. This is defined as

$$\mathrm{Min_d}\,(X_1, X_2) = \left(\sum_{j=1}^{p}\left|X_{1j} - X_{2j}\right|^q\right)^{1/q}$$

Equation 5: Minkowski's Distance

Where q is a positive integer. Note that for q = 1, Minkowski's distance coincide with Manhattan distance and for q = 2, it is identical to Euclidean distance.

There are many other important distance measures, among which the Mahalanobis Distance, the Cosine Distance and the Jaccard Distance deserve mention.

Following is an illustration of distance matrices computed for data given in Fig 1. Note that the matrices shown are all lower triangular since the principal diagonal is 0 and the distance matrices are symmetric.

Computing distances

Use sample data through manual input

```
import numpy as np
from scipy.spatial.distance import squareform,pdist
```

```
pts=np.array([[0.53,0.53,0.54,0.54,0.53],[3.65,3.65,3.66,3.66,3.67],[3.67,3.68,3.68,
3.67,3.66]]).transpose()
```

```
pts
array([[0.53, 3.65, 3.67],
       [0.53, 3.65, 3.68],
       [0.54, 3.66, 3.68],
       [0.54, 3.66, 3.67],
       [0.53, 3.67, 3.66]])
```

```
squareform(np.round(pdist(pts, metric='euclidean'),2))
array([[0.  , 0.01, 0.02, 0.01, 0.02],
       [0.01, 0.  , 0.01, 0.02, 0.03],
       [0.02, 0.01, 0.  , 0.01, 0.02],
       [0.01, 0.02, 0.01, 0.  , 0.02],
       [0.02, 0.03, 0.02, 0.02, 0.  ]])
```

```
squareform(np.round(pdist(pts, metric='cityblock'),2))
array([[0.  , 0.01, 0.03, 0.02, 0.03],
       [0.01, 0.  , 0.02, 0.03, 0.04],
       [0.03, 0.02, 0.  , 0.01, 0.04],
       [0.02, 0.03, 0.01, 0.  , 0.03],
       [0.03, 0.04, 0.04, 0.03, 0.  ]])
```

```
squareform(np.round(pdist(pts, metric='minkowski', p=2),2))
array([[0.  , 0.01, 0.02, 0.01, 0.02],
       [0.01, 0.  , 0.01, 0.02, 0.03],
       [0.02, 0.01, 0.  , 0.01, 0.02],
       [0.01, 0.02, 0.01, 0.  , 0.02],
       [0.02, 0.03, 0.02, 0.02, 0.  ]])
```

Note that the numerical values of the distances may be different depending on which method has been used to compute distances. The distance between User 1 and 2 is 0.01 using all three methods, while the distance between Users 1 and 3 depends on the method used. Euclidean distance between Users 1 and 3 is 0.02. However, the Manhattan or cityblock distance between Users 1 and 3 is 0.03. This observation indicates that the clustering may depend on the distance measure. In Python, the output for the Minkowski distance will be same as Euclidean Distance because of the parameter p=2 (by default). The parameter p=1 gives the Manhattan distance.

4. Hierarchical Clustering

4.1 Definition

Hierarchical clustering (also called **hierarchical cluster analysis** or **HCA**) is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

- **Agglomerative**: This is a "bottom-up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- **Divisive**: This is a "top-down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

In general, the merges and splits are determined in a greedy manner.

Agglomerative clustering is used more often than divisive clustering. In this monograph we will not discuss divisive clustering in any detail (see Section 6)

Clustering of observations is the objective here. However, it is also possible to apply hierarchical clustering to cluster attributes (variables).

4.2 Agglomerative Clustering

Figure 2 shows a graphical representation of agglomerative clustering. At each stage two units or two clusters or one unit and one cluster are combined into a larger cluster. The graphical representation is a tree diagram and is called a *dendrogram*. At the start of the algorithm, the number of clusters is equal to the number of observations *n*. At the end of the algorithm, the number of clusters is 1.

The optimum number of clusters is not pre-determined in this algorithm. Depending on which height the dendrogram is cut, the final number of clusters is determined.

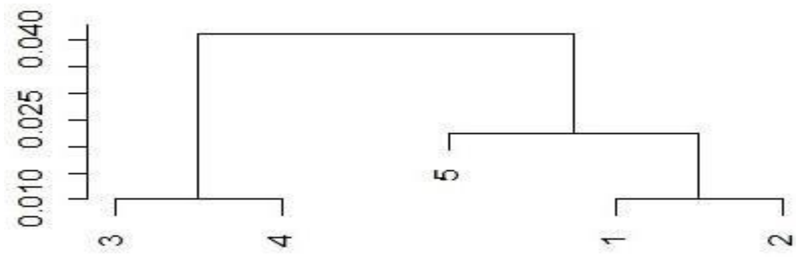


Figure 2: Diagram of an agglomerative clustering

This method builds the hierarchy from the individual elements by progressively merging clusters. In Fig 2, there are five elements {1} {2} {3} {4} and {5}. Each element represents one cluster of a single element, called *singleton*. The first step is to merge the two *closest* elements into a single cluster.

Closeness of two elements depends on the chosen distance.

Let us use Euclidean distance for illustration. In Fig 2, cutting after the first merge (from the bottom) of the dendrogram will yield clusters {1, 2} {5} {3, 4}. Cutting after the second merge will yield clusters {1, 2, 5} {3, 4}, which is a coarser clustering, with a smaller number of clusters but one or more clusters having larger size.

Once one or more clusters contain more than one element, the algorithm of merging two multi-element clusters need to be developed.

4.2.1 Concept of Linkage

Once at least one multiple-element cluster is formed, a process needs to be devised how to compute distances between a set of observations and a singleton, or between two sets of observations. This may be defined in various manner.

Single Linkage: Distance between two clusters is defined to be the smallest distance between a pair of observations (points), one from each cluster. This is the distance between two closest points of the two clusters.

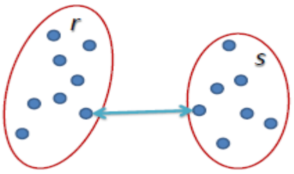


Figure 3: Single linkage clustering

Complete Linkage: Distance between two clusters is defined to be the largest distance between a pair of points, one from each cluster. This is the distance between two points belonging to two different clusters, which are farthest apart.

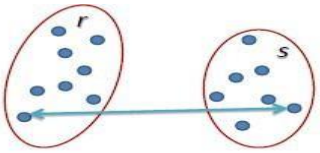


Figure 4: Complete linkage clustering

Average Linkage: Distance between two clusters is defined as the average distance between each pair of points, one from each cluster.

Centroid Linkage: Centroid of a cluster is defined as the vector of means of all attributes, over all observations within that cluster. Centroid linkage between two clusters is the distance between the respective centroids.

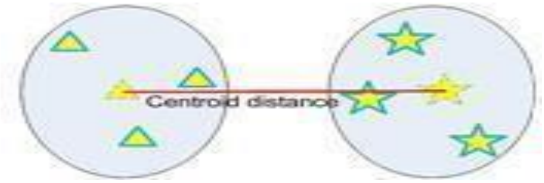


Figure 5: Centroid linkage clustering

Ward's Linkage: This is an alternative method of clustering, also known as minimum variance clustering method. In that sense it is similar to the *k*-means clustering (which is discussed later), an iterative method where after every merge, the distances are updated successively. Ward's method often creates compact and even-sized clusters. However, the drawback is, it may result in less than optimal clusters. Like all other clustering methods, this is also computationally intensive.

Note that the choice of distance and linkage method uniquely define the outcome of a clustering procedure. It is possible that the same set of observations may be clustered in different partitions based on the choice of distance and linkage method.

Case Study:

Review ratings by 5456 travelers (Users) on 24 tourist attractions across various sites in Europe are considered. This data is the primary input for an automatic recommender system. Each reviewer (User) has rated various attractions, such as churches, resorts, bakeries etc. and the given ratings are all between 0 (lowest) and 5 (highest). Objective is to partition the sample of travelers into clusters, so that tailored recommendations may be made for next travel destinations.

Statement of the problem: Cluster the Users into mutually exclusive groups according to their preferences for various tourist attractions.

EDA on Review Ratings Data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import dendrogram, linkage,fcluster
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
sns.set(context="notebook", palette="Spectral", style = 'darkgrid' ,font_scale =
1.5, color_codes=True)
```

```
df = pd.read_csv('ReviewRatings.csv')
```

```
round(df.describe().T,2)
```

	count	mean	std	min	25%	50%	75%	max
Churches	5456.0	1.46	0.83	0.00	0.92	1.34	1.81	5.0
Resorts	5456.0	2.32	1.42	0.00	1.36	1.90	2.68	5.0
Beaches	5456.0	2.49	1.25	0.00	1.54	2.06	2.74	5.0
Parks	5456.0	2.80	1.31	0.83	1.73	2.46	4.09	5.0
Theatres	5456.0	2.96	1.34	1.12	1.77	2.67	4.31	5.0
Museums	5456.0	2.89	1.28	1.11	1.79	2.68	3.84	5.0
Malls	5456.0	3.35	1.41	1.12	1.93	3.23	5.00	5.0
Zoo	5456.0	2.54	1.11	0.86	1.62	2.17	3.19	5.0
Restaurants	5456.0	3.13	1.36	0.84	1.80	2.80	5.00	5.0
Bars	5456.0	2.83	1.31	0.81	1.64	2.68	3.53	5.0
Local Services	5456.0	2.55	1.38	0.78	1.58	2.00	3.22	5.0
Fast Food	5456.0	2.08	1.25	0.78	1.29	1.69	2.28	5.0
Lodgings	5456.0	2.13	1.41	0.77	1.19	1.61	2.36	5.0
Juice Bars	5456.0	2.19	1.58	0.76	1.03	1.49	2.74	5.0
Art Galleries	5456.0	2.21	1.72	0.00	0.86	1.33	4.44	5.0
Dance Clubs	5456.0	1.19	1.11	0.00	0.69	0.80	1.16	5.0
Swimming Pools	5456.0	0.95	0.97	0.00	0.58	0.74	0.91	5.0

Gyms	5456.0	0.82	0.95	0.00	0.53	0.69	0.84	5.0
Bakeries	5456.0	0.97	1.20	0.00	0.52	0.69	0.86	5.0
Spas	5456.0	1.00	1.19	0.00	0.54	0.69	0.86	5.0
Cafes	5456.0	0.97	0.93	0.00	0.57	0.76	1.00	5.0
View Points	5456.0	1.75	1.60	0.00	0.74	1.03	2.07	5.0
Monuments	5456.0	1.53	1.32	0.00	0.79	1.07	1.56	5.0
Gardens	5456.0	1.56	1.17	0.00	0.88	1.29	1.66	5.0

There are 5456 reviewers' ratings in the data. In each of the attributes, the maximum value is 5, i.e. the highest possible score has been reached. However, not all attributes have been given the lowest (0) rating. The mean value is always greater than the median value, sometimes considerably, indicating that the distribution of the attributes is not symmetric, but positively skewed.

Histograms of the variables are shown below:

```
# checking distributions using histograms
x = df.copy()
x = x.drop(['User'],axis=1)
fig = plt.figure(figsize = (15,20))
ax = fig.gca()
x.hist(ax = ax)
plt.tight_layout()
plt.show()
```

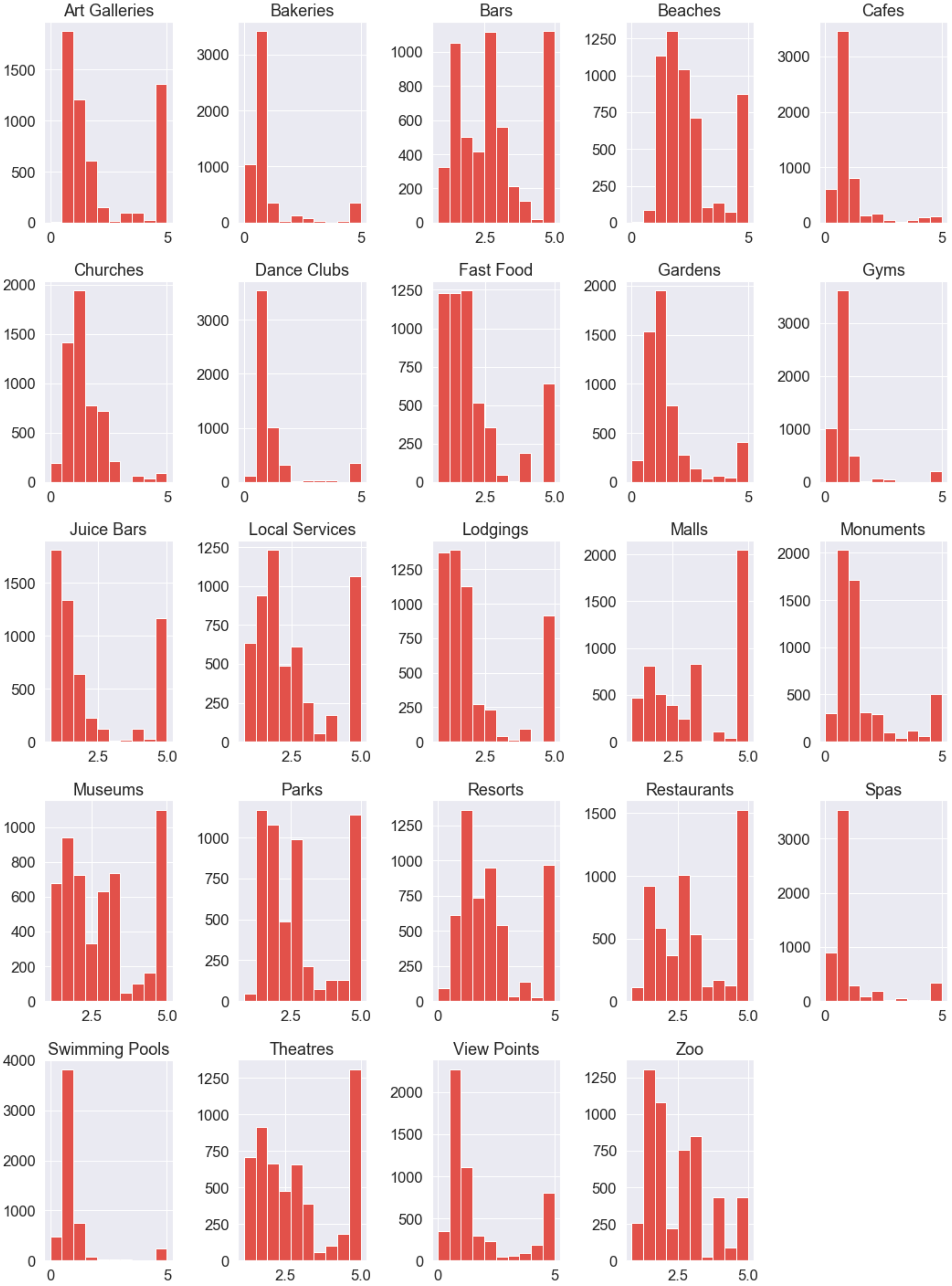


Figure 6: Histogram of the variables to be used to compute distance

The histograms bear proof of the inherent skewness in the attributes. Except for two attributes (churches and zoo) most of the data is concentrated towards the lower ranks with a disproportionate amount concentrated on rank 5.

Following figure gives a quick comparison of attribute means. Whiles malls, restaurants, theatres etc. have high average values, swimming pools and gyms have the lowest averages.

```
AvgR = df.mean()
AvgR = AvgR.sort_values()
plt.figure(figsize=(10,7))
plt.barh(np.arange(len(df.columns[1:])), AvgR.values, align='center')
plt.yticks(np.arange(len(df.columns[1:])), AvgR.index)
plt.ylabel('Categories')
plt.xlabel('Average Rating')
plt.title('Average Rating for every Category')
plt.show()
```

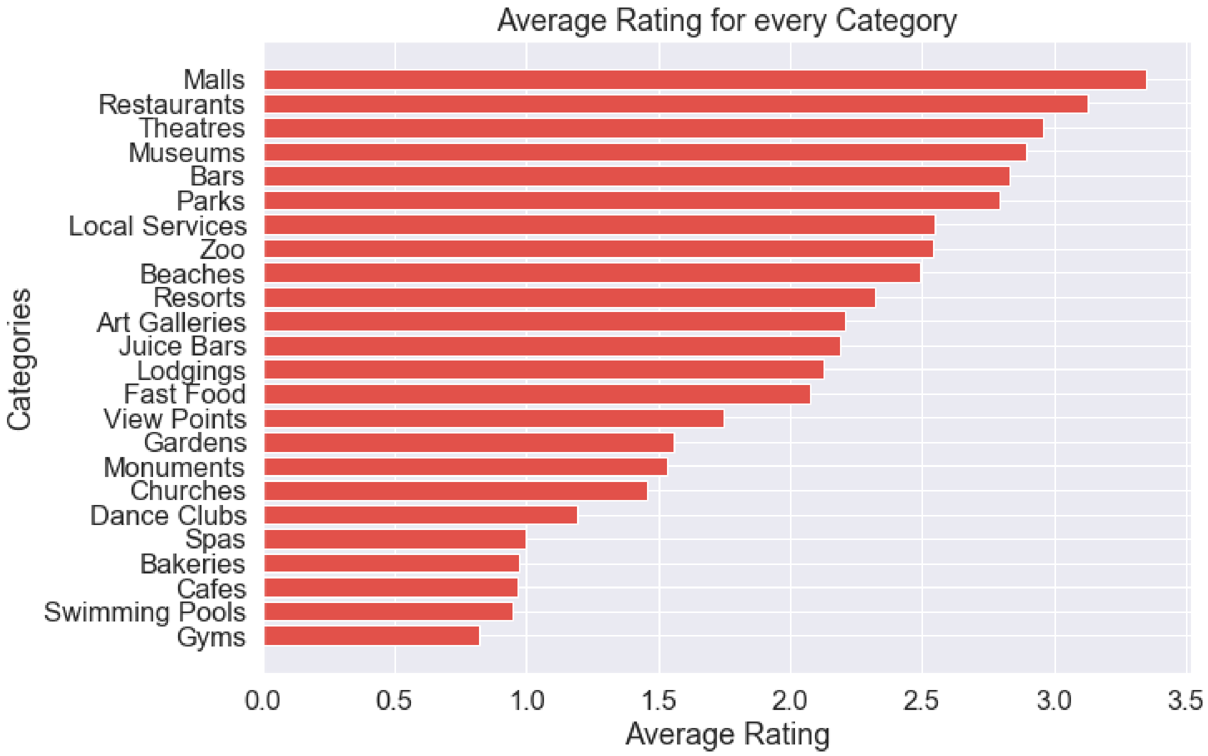


Figure 7: Visual comparison of attribute means

Clustering (Illustrative)

Visualization plays an important role in identifying the number of clusters in a set of observations. To illustrate the process only the users with serial number 100 – 120 are selected. Later the process is applied on the whole data set with 5456 users. All attributes are used in the hierarchical clustering process.

Both Euclidean distance and Manhattan distances are used with single and complete linkage. The codes are given in the following panel whereas the dendrograms are shown together later to facilitate comparison.

```
# Clustering Illustrative
#No Need for scaling as all variables have the same weightage.
#It’s a rating dataset
x=pd.DataFrame(x)
x_subset1 = x.loc[100:120 ,:].values
```

```
dendrogram = sch.dendrogram(sch.linkage(x_subset1, method  = "single", metric='euclidean'))
plt.title('Dendrogram- Eucildean Single')
plt.xlabel('Users')
plt.ylabel('Euclidean distances')
plt.show()
```

```
dendrogram = sch.dendrogram(sch.linkage(x_subset1, method  = "single", metric='cityblock'))
plt.title('Dendrogram - Manhattan Single')
plt.xlabel('Users')
plt.ylabel('Manhattan distances')
plt.show()
```

```
dendrogram = sch.dendrogram(sch.linkage(x_subset1, method  = "complete", metric='euclidean'))
plt.title('Dendrogram- Euclidean Complete')
plt.xlabel('Users')
plt.ylabel('Euclidean distances')
plt.show()
```

```
dendrogram = sch.dendrogram(sch.linkage(x_subset1, method  = "single", metric='cityblock'))
plt.title('Dendrogram- Manhattan Complete')
plt.xlabel('Users')
plt.ylabel('Manhattan distances')
plt.show()
```

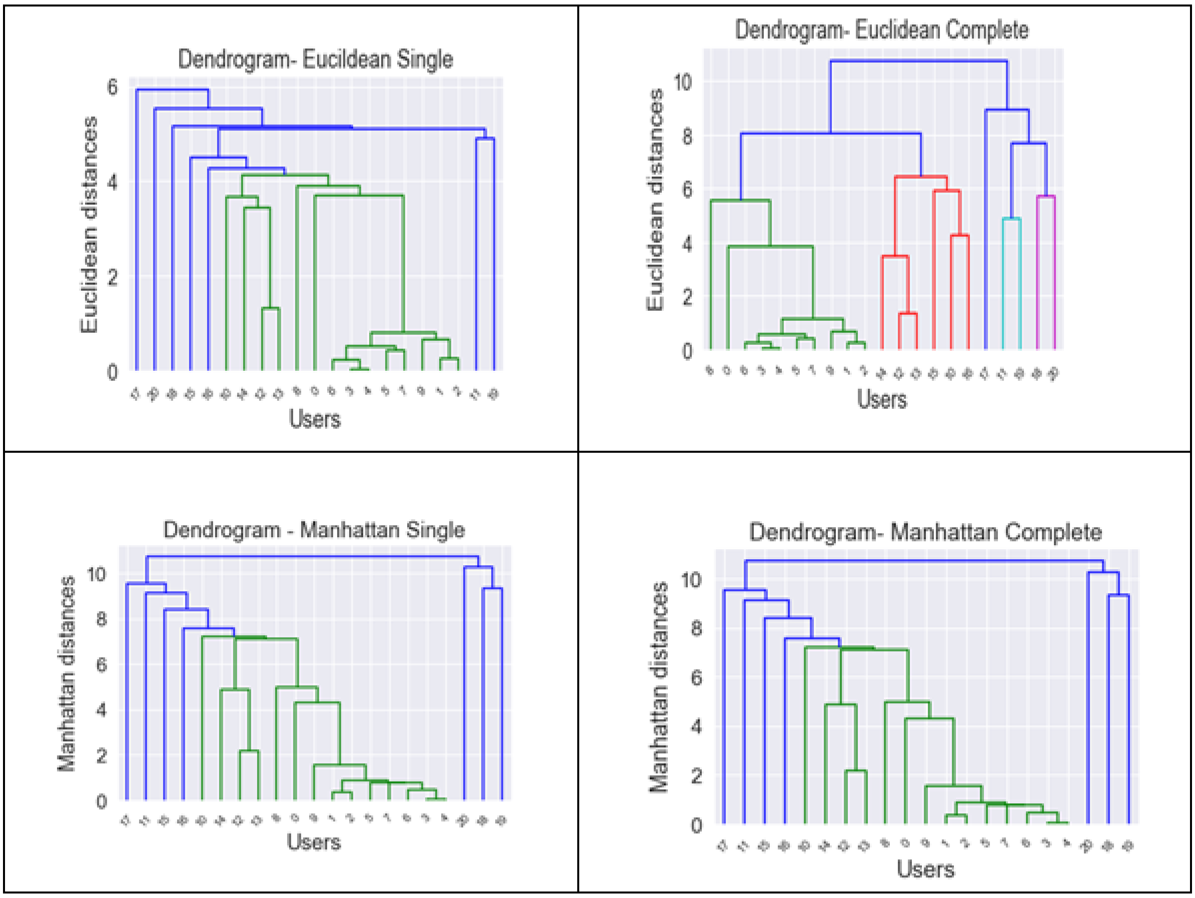


Figure 8: Comparison of distance and linkage method in constructing clusters

Illustrative clustering using Ward's method:

```
dendrogram = sch.dendrogram(sch.linkage(x_subset1, method  = "ward", metric='euc
lidean'))
plt.title('Dendrogram- Euclidean Ward')
plt.xlabel('Users')
plt.ylabel('Euclidean distances')
plt.show()
```

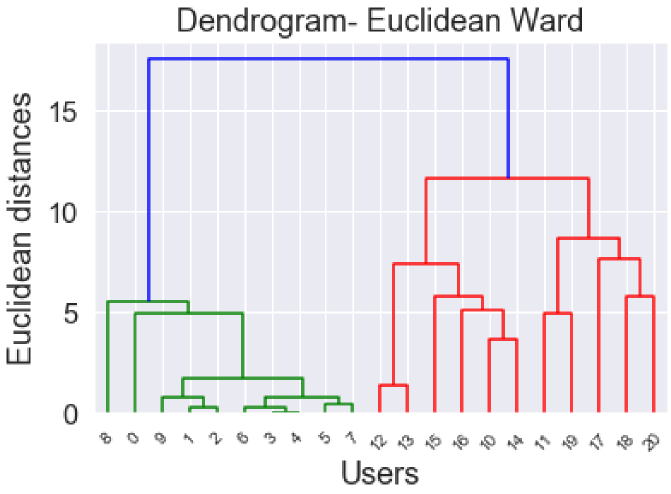


Figure 9: Ward's method of constructing clusters

A dendrogram is a visual representation of cluster-making. On the x-axis are the item names or item numbers. On the y-axis is the distance or height. The vertical straight lines denote the height where two items or two clusters combine. The higher the level of combining, the distant the individual items or clusters are. By definition of hierarchical clustering, all items must combine to make one cluster.

However, the problem of interest here is to determine the optimum number of clusters. Recall that each cluster is a representative of a different population. Since this is a problem of unsupervised learning, there cannot be any error measure to dictate the number of clusters. To some extent this is a subjective choice. After considering the dendrogram, one may decide the level where the resultant tree needs to be cut. This is another way to say that how close-knit or dispersed should the clusters be. If the number of clusters is large, the cluster size is small and the clusters homogeneous. If the number of clusters is small, each contains more items and hence clusters are more heterogeneous. Often practical consideration or domain knowledge plays a part in determining the number of clusters.

Note that depending on the distance measure and linkage used, the number of clusters and their composition may be different. Compare the dendrograms constructed by Euclidean distance and complete linkage and Manhattan distance and complete linkage in Figures 8 - 9. In the former there seems 3 clear clusters:

- Cluster 1: Users 100 – 109 with User 108 as a single sub-cluster
- Cluster 2: User 110, User 112 – 116
- Cluster 3: User 111, Users 117 – 120

In case of the other dendrogram (Manhattan and complete linkage), there seems to be 2 clusters

- Cluster 1: Users 118-120
- Cluster 2: Users 100-117 with many users appearing as a single cluster

In the above cluster segregation, there are many sub-clusters of a single user, which might be an indication of extreme values and/or outliers.

Ward's method also seems to indicate 2 clusters.

- Cluster 1: Users 100 – 109
- Cluster 2: Users 110 – 120

Both clusters are of approximately equal size and there does exist two users as individual sub-cluster.

This does create subjectivity in the process. Nevertheless, the items (in this case Users) that are really close, are combining a single cluster irrespective of the distance measure or linkage.

Now let us proceed to the final clustering involving all Users, Euclidean distance and ward linkage.

```
### Full Dataset
      x_full = x.loc[:].values
      plt.figure(figsize=(20, 20))
      plt.title("Customer Segmentation Dendograms")
      Z = sch.dendrogram(sch.linkage(x_full, method  = "ward", metric='euclidean
      ') ,color_threshold=85.25)
      plt.show()
```

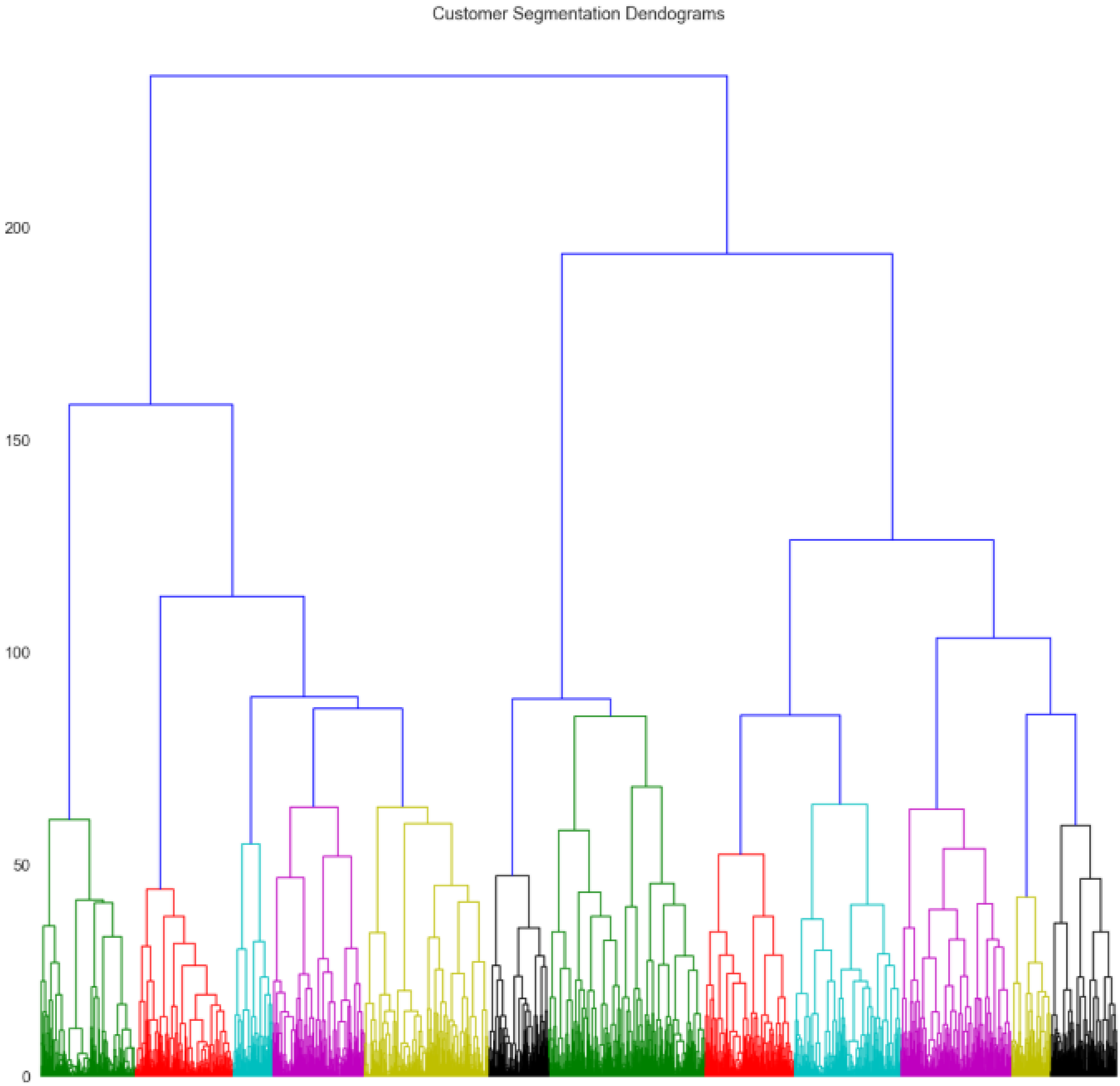


Figure 10: Dendrogram of the hierarchical clusters for the full data set

Dendrogram of Figure 10 includes all Users. The colours are given by color_threshold parameter and can be adjusted. However, going by color_threshold=None, there are 3 clusters. It is difficult for a business to understand segments from just 3 clusters. After multiple iterations, 12 was identified as the suitable number of clusters and hence using the function from *sklearn* library, 12 clusters were created in the full data set and Cluster label was included in the data set.

```
Y = sch.linkage(x_full, method  = "ward", metric='euclidean')

plt.title('Truncated cluster dendrogram ')
plt.xlabel('sample index or (cluster size)')
plt.ylabel('distance')
sch.dendrogram(
    Y,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=12,  # show only the last p merged clusters
    leaf_rotation=90.,
    leaf_font_size=12.,
)
plt.show()
```

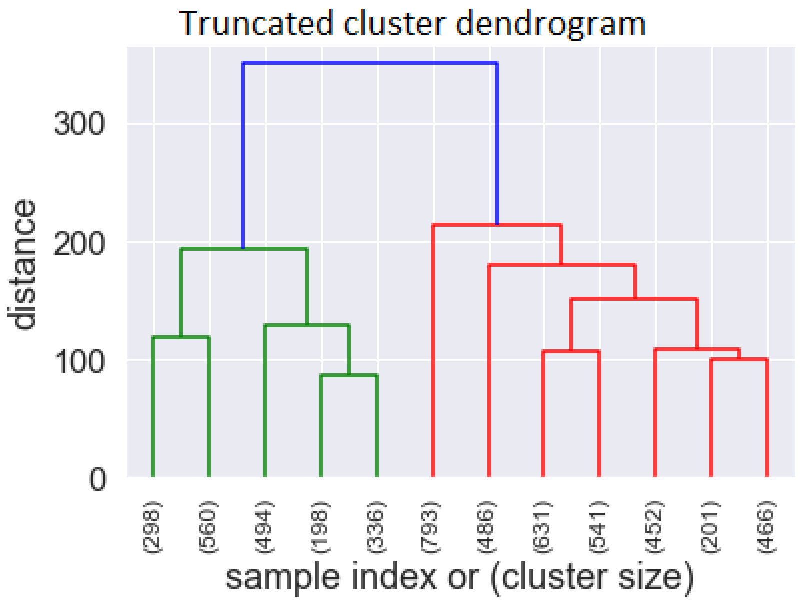


Figure 11: Truncated Cluster Plot

Figure 11 does not show the full grown tree but just the number of Users in each cluster. Cluster size is not equal for all 12 clusters, as indicated in the above diagram. It is important to note that based on the method selected for linkage and affinity, cluster membership and cluster size could vary.

```
cluster = AgglomerativeClustering(n_clusters=12, affinity='euclidean', linkage='ward
')
Cluster_No =cluster.fit_predict(x)
print(Cluster_No)
[5 5 5 ... 8 8 8]
x['Cluster'] = Cluster_No
```

Finally, to verify the differences between the identified clusters, a few important variables are chosen and their means and standard deviations are compared below.

```
### Let's select some specific features and find out how well clusters are
differentiating the same

x_analysis = x [['Lodgings', 'Fast Food', 'Restaurants','Cluster']]
```

```
x_analysis.groupby('Cluster').mean()
```

```
         Lodgings Fast Food Restaurants
Cluster
0            1.69      1.53      2.11
1            1.58      1.84      4.70
2            1.73      1.86      3.23
3            4.52      4.29      2.76
4            2.52      2.13      3.97
5            1.70      1.70      2.69
6            1.11      1.63      2.43
7            2.27      2.23      3.20
8            1.04      1.10      1.54
9            2.61      2.86      2.63
10           5.00      1.58      3.70
11           1.88      2.30      4.42
```

Recall that the clustering process here is an input to a recommender system. Instead of pushing all information indiscriminately to every user, based on the average ranking of various attributes in various groups, more efficient planning may be made. Consider for example Cluster 1 and Cluster 11 have given very high ranking to Restaurants whereas Clusters 0, 6 and especially 8 have given low ranking to Fast Food. It is clear that users belonging to Cluster 8 are discriminatory eaters and information about fast food to these users will not be useful at all. Compared to them, preference of Cluster 3 shows high ranks in Lodging and Fast Food but medium rank in Restaurant. It is possible that these users are more convenience loving and information about good accommodation and perhaps near-by fast food joints will be appreciated by this group.

As a practical consideration though, 12 may also be considered too many clusters.

5. *K*-Means Clustering

k-means clustering is the most used non-hierarchical clustering technique. It aims to partition n observations into *k* clusters in which each observation belongs to the cluster whose mean (centroid) is nearest to it, serving as a prototype of the cluster. It minimizes within-cluster variances (squared Euclidean distances).

The most common algorithm uses an iterative refinement technique. Due to its ubiquity, it is often called "the k-means algorithm".

Given an initial set of k means $m_{1(1)}$, …, $m_{k(1)}$, the algorithm proceeds by alternating between two steps
Assignment step: Assign each observation to the cluster whose mean has the least squared Euclidean distance; this is intuitively the "nearest" mean.

Update step: Calculate the new means (centroids) of the observations in the new clusters.

The algorithm has converged when the assignments no longer change.

One advantage of *k*-means clustering is that computation of distances among all pairs of observations is not necessary. However, the number of clusters *k* needs to be pre-specified. Once that is determined, an arbitrary partition of data into *k* clusters is the starting point. Then sequential assignment and update will find the clusters that are most separated. The cardinal rule of cluster building is that, at every step within-cluster variance will reduce (or stay the same) but between-cluster variance will increase (or stay the same).

Random assignment of initial centroids eliminate bias. However, it is not an efficient algorithmic process. An alternative way to form clusters is to start with a set of pre-specified starting points, though it is possible that such a set of initial clusters, if converges to a local optimality, may introduce bias. The following algorithm is a compromise between efficiency and randomness.

Leader Algorithm:

- **Step 1**. Select the first item from the list. This item forms the centroid of the first cluster.
- **Step 2**. Search through the subsequent items until an item is found that is at least distance δ away from any previously defined cluster centroid. This item will form the centroid of the next cluster.
- **Step 3**: Step 2 is repeated until all *k* cluster centroids are obtained or no further items can be assigned.
- **Step 4**: The initial clusters are obtained by assigning items to the nearest cluster centroids.

The distance δ along with the number of clusters *k* is the input to this algorithm. It is possible that domain knowledge or various other subjective considerations determine the value of *k*. If no other subjective input is available, *k* may be determined by empirical methods.

5.1 Determining the Number of Clusters

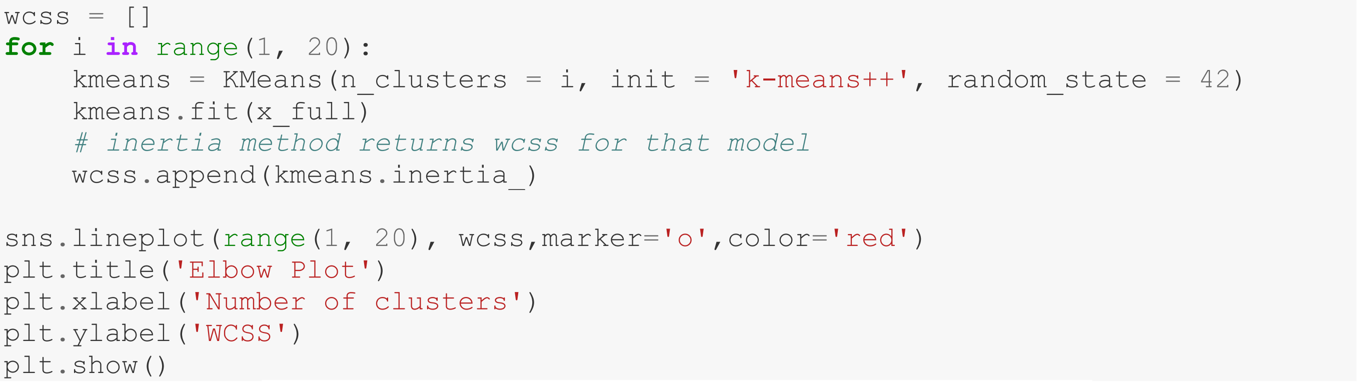
There are many methods that are recommended for determination of an optimal number of partitions. Unfortunately, however, there is no closed form solution to the problem of determining *k*. The choice is somewhat subjective and graphical methods are often employed.

Objective of partitioning is to separate out the observations or units so that the 'most' similar items are put together. Recall that singleton clusters will have the lowest value of WSS, but that is not useful. Hence finding *k* is striking a balance between WSS and cluster size.

5.1.1 Elbow method

For a given number of clusters, the total within-cluster sum of squares (WCSS) is computed. That value of *k* is chosen to be optimum, where addition of one more cluster does not lower the value of total WCSS appreciably.

The Elbow method looks at the total WCSS as a function of the number of clusters.



```
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(x_full)
    # inertia method returns wcss for that model
    wcss.append(kmeans.inertia_)

sns.lineplot(range(1, 20), wcss,marker='o',color='red')
plt.title('Elbow Plot')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

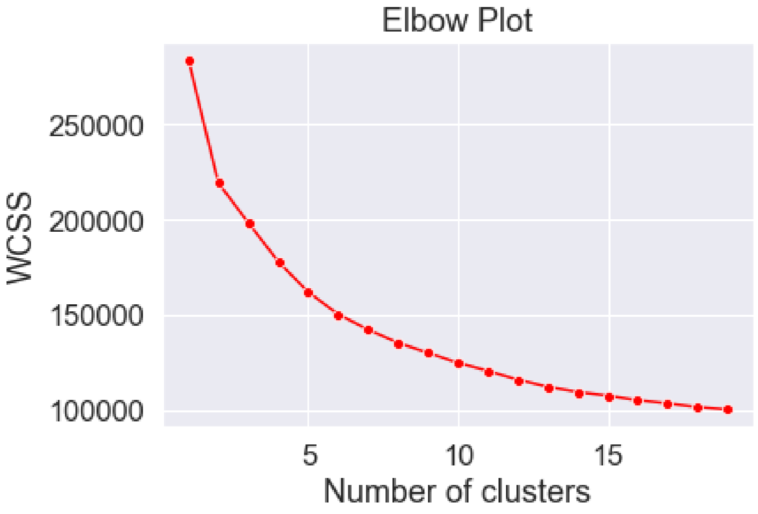


Figure 12: Elbow method to determine optimum number of clusters

Fig 12 does not indicate any clear break in the elbow after *k*=2. Hence one option for optimum number of clusters is 2 and thereafter dip is visible for *k*=15.

[Recall that hierarchical clustering of the same data suggested 12 clusters. There may be wide discrepancy in the number of clusters depending on the procedure applied. This will be clearer in subsequent analysis]

5.1.2 *Silhouette Method*

This method measures how tightly the observations are clustered and the average distance between clusters. For each observation a silhouette score is constructed which is a function of the average distance between the point and all other points in the cluster to which it belongs, and the distance between the point and all other points in all other clusters, that it does not belong to. The maximum value of the statistic indicates the optimum value of k.

```
ss={1:0}
for i in range(2, 20):
    clusterer = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    y=clusterer.fit_predict(x_full)
   # The higher (up to 1) the better
    s =silhouette_score(x_full, y )
    ss[i]=round(s,5)
    print("The Average Silhouette Score for {} clusters is {}".format(i,round(s,
5)))
```

```
The Average Silhouette Score for 2 clusters is 0.14426
The Average Silhouette Score for 3 clusters is 0.14453
The Average Silhouette Score for 4 clusters is 0.14711
The Average Silhouette Score for 5 clusters is 0.15228
The Average Silhouette Score for 6 clusters is 0.15008
The Average Silhouette Score for 7 clusters is 0.14728
The Average Silhouette Score for 8 clusters is 0.15676
The Average Silhouette Score for 9 clusters is 0.15885
The Average Silhouette Score for 10 clusters is 0.15272
The Average Silhouette Score for 11 clusters is 0.16131
The Average Silhouette Score for 12 clusters is 0.16544
The Average Silhouette Score for 13 clusters is 0.16522
The Average Silhouette Score for 14 clusters is 0.16211
The Average Silhouette Score for 15 clusters is 0.1624
The Average Silhouette Score for 16 clusters is 0.16136
The Average Silhouette Score for 17 clusters is 0.16388
The Average Silhouette Score for 18 clusters is 0.16246
The Average Silhouette Score for 19 clusters is 0.16035
```

```
maxkey= [key for key, value in ss.items() if value == max(ss.values())][0]
fig,ax = plt.subplots(figsize=(12,4))
sns.pointplot(list(ss.keys()),list(ss.values()))
plt.vlines(x=maxkey-1,ymax=0,ymin=0.25,linestyles='dotted')
ax.set(ylim=(0, 0.19))
ax.set_title('Silhouette Plot')
ax.set_xlabel('Number of clusters')
ax.set_ylabel('Average Silhouette Score')
plt.show()
```

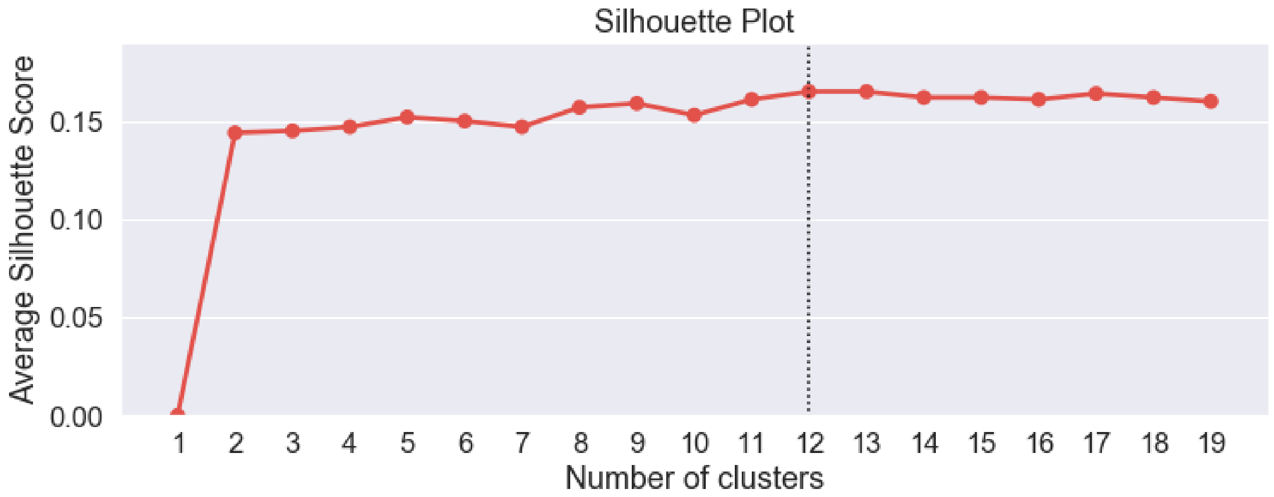


Figure 13: Silhouette Plot

It is clear from Fig 13 that the maximum value of average silhouette score is achieved for *k* = 12, which, therefore, is considered to be the optimum number of clusters for this data.

However, there are a number of merits for using a smaller number of clusters. The objective of this particular clustering effort is to devise a suitable recommendation system. It may not be practical to manage a very large number of tailor made recommendations. Hence, the final decision regarding an appropriate number of clusters must be taken after considering the within sum of squares and between sum of squares. Recall that within cluster sum of squares is the squared average Euclidean distance of all the points within a cluster from the cluster centroid and between cluster sum of squares is the average squared Euclidean distance between all cluster centroids.

Let us now proceed with 12 clusters. Clustering, like any other predictive modelling, is an iterative process. The final recommendation may be quite different from the initial starting point.

Case Study continued:

Let us decide to partition 5456 users into 12 partitions. Cluster plot is a helpful way to look at the spread and overlap of clusters. Ideally, for a perfect algorithm, the clusters will be well separated with no (or minimum) overlap.

The algorithm for *k*-means is a greedy algorithm and is sensitive to the random starting points.

k = 12

```
from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(x_full)
plt.figure(figsize=(12,7))
sns.scatterplot(x=plot_columns[:,1], y=plot_columns[:,0], hue=KMeans(n_clusters=
12, random_state=0).fit(x_full).labels_, palette='Dark2_r',legend=False)
plt.title('Cluster Plot for 12 Clusters')
plt.show()
```

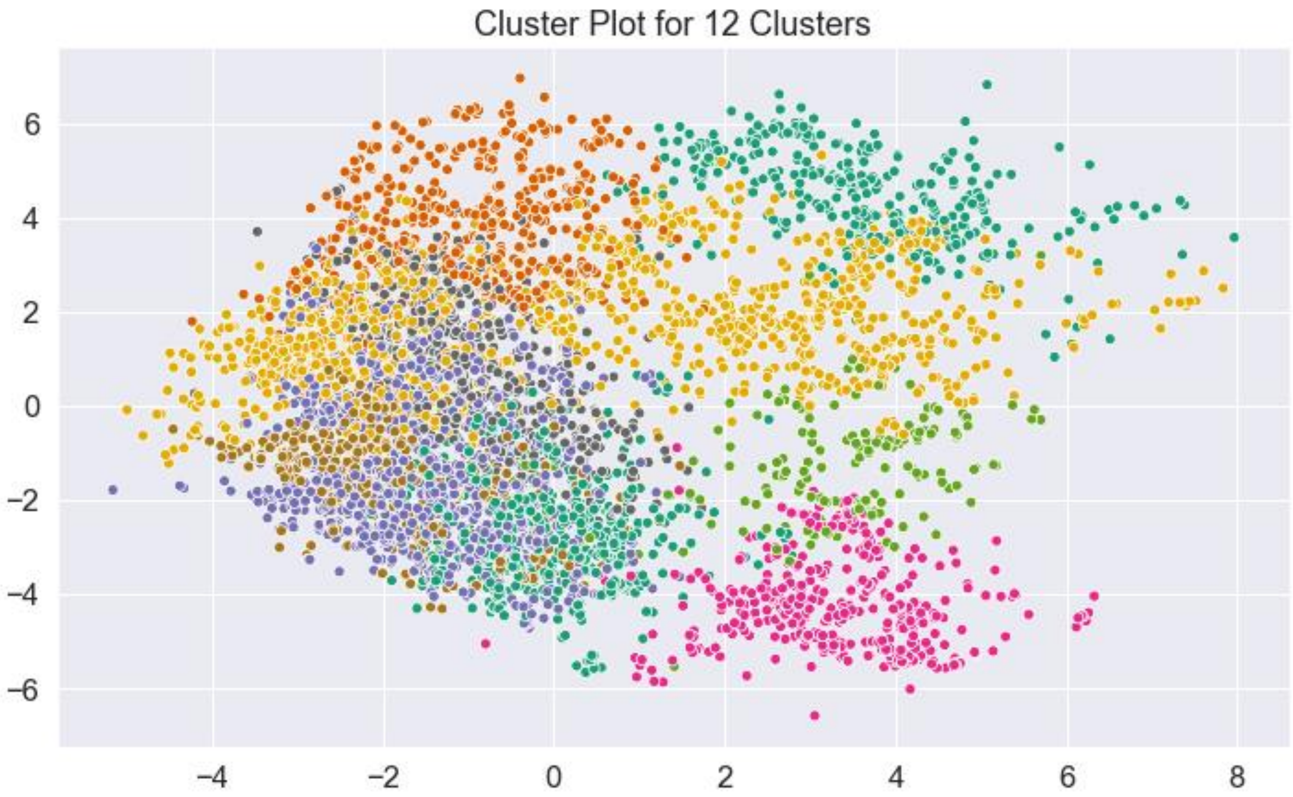


Figure 14: Cluster Plot for 12 Clusters

Note that elbow plot and silhouette plot both recommended a very large number of clusters for this data set. Though these two instruments are probably the most used for determination of *k*, the final decision must not be taken based solely on such automatic considerations. It is clear from Fig 14 that 12 clusters are too many and a considerable overlap exists among the clusters.

Now it is important to investigate the other outcomes of the clustering procedure, namely, the total sum of squares, the within sum of squares for each cluster, the total within sum of squares, the between cluster sum of squares and the cluster centers.

```
x=KMeans(n_clusters=12, random_state=0).fit(x_full).cluster_centers_
partition, euc_distance_to_centroids = vq(x_full, x)

TSS = np.sum((x_full-x_full.mean(0))**2)
SSW = np.sum(euc_distance_to_centroids**2)
SSB = TSS - SSW
print('Between Sum of Squares is ',round(SSB,2))
print('Total Sum of Squared is ',round(TSS,2))
WSS=pd.DataFrame([euc_distance_to_centroids**2,KMeans(n_clusters=12, random_stat
e=0).fit(x_full).labels_]).T
print('The Within Sum of Squares is',np.round(WSS.groupby(1).sum().values,2))
print('Total Within Sum of Squared is ',round(SSW,2))
print('The Cluster Size is',WSS.groupby(1).count().values)
```

```
Between Sum of Squares is  105417.56
Total Sum of Squared is  215751.36
The Within Sum of Squares is [[ 7204.03]
 [ 5191.59]
 [ 8958.72]
 [12457.79]
 [13401.97]
 [ 4298.14]
 [ 7827.57]
 [ 9236.24]
 [ 8107.17]
 [13530.54]
 [ 7696.57]
 [12423.48]]
Total Within Sum of Squared is  110333.8
The Cluster Size is [[288]
 [244]
 [485]
 [624]
 [575]
 [196]
 [482]
 [609]
 [474]
 [539]
 [337]
 [603]]
```

Since *k* = 12 is not a practical choice, let us consider several other values of *k* to determine a set of well-separated meaningful clusters.

Let us start with *k* =3

```
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(x_full)
plt.figure(figsize=(12,7))
sns.scatterplot(x=plot_columns[:,1], y=plot_columns[:,0], hue=KMeans(n_clusters=
3, random_state=0).fit(x_full).labels_,
            palette='Dark2_r',legend=False)
plt.title('Cluster Plot for 3 Clusters')
plt.show()
```

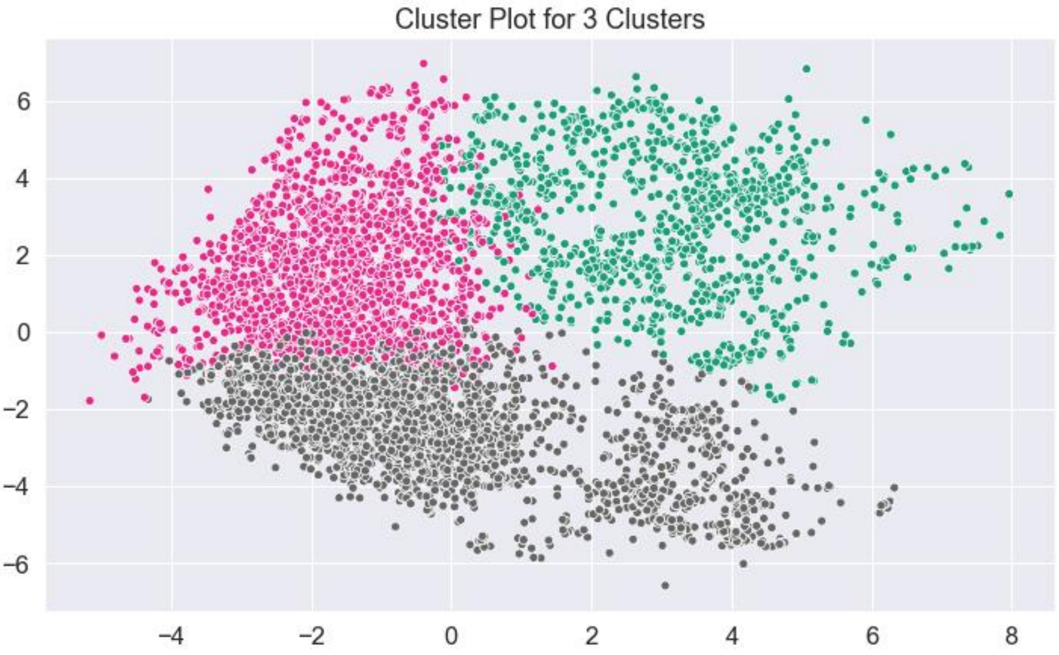


Figure 15:Cluster Plot for 3 Clusters

Fig 15 shows that the 3 clusters are much better separated with almost negligible overlap.

However, the final decision must be taken after considering the within sum of squares and between sum of squares. Recall that within cluster sum of squares is the squared average Euclidean distance of all the points within a cluster from the cluster centroid and between cluster sum of squares is the average squared Euclidean distance between all cluster centroids.

```
x=KMeans(n_clusters=3, random_state=0).fit(x_full).cluster_centers_
partition, euc_distance_to_centroids = vq(x_full, x)

TSS = np.sum((x_full-x_full.mean(0))**2)
SSW = np.sum(euc_distance_to_centroids**2)
SSB = TSS - SSW
print('Between Sum of Squared is ',round(SSB,2))
print('Total Sum of Squared is ',round(TSS,2))
WSS=pd.DataFrame([euc_distance_to_centroids**2,KMeans(n_clusters=3, random_state
=0).fit(x_full).labels_]).T
print('The Within Sum of Squares is',np.round(WSS.groupby(1).sum().values,2))
print('Total Within Sum of Squared is ',round(SSW,2))
print('The Cluster Size is',WSS.groupby(1).count().values)
```

```
Between Sum of Squared is  50294.06
Total Sum of Squared is  215751.36
The Within Sum of Squares is [[74659.32]
 [56898.67]
 [33899.31]]
Total Within Sum of Squared is  165457.3
The Cluster Size is [[2464]
 [1909]
 [1083]]
```

Note that the total sum of square in the data does not depend on the number of clusters. Total sum of squares for the current data set is 215751.36 which is a measure of total variability within the data. Each cluster has different within cluster sum of squares. For *k* = 12, the largest within cluster sum of squares is 13530.54 and the total within cluster sum of squares is 110333.8. The between cluster sum of squares is 105417.56. Let us now investigate the case with *k* = 3. The between cluster sum of squares is smaller than the within cluster sum of squares for two clusters. Hence *k* = 3 cannot be recommended.

Next a few other values of *k* are considered and *k* = 5 seems to be an optimal number of clusters.

```
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(x_full)
plt.figure(figsize=(12,7))
plt.scatter(x=plot_columns[:,1], y=plot_columns[:,0], c=KMeans(n_clusters=6, ran
dom_state=0).fit(x_full).labels_)
plt.title('Cluster Plot for 6 Clusters')
plt.show()
```

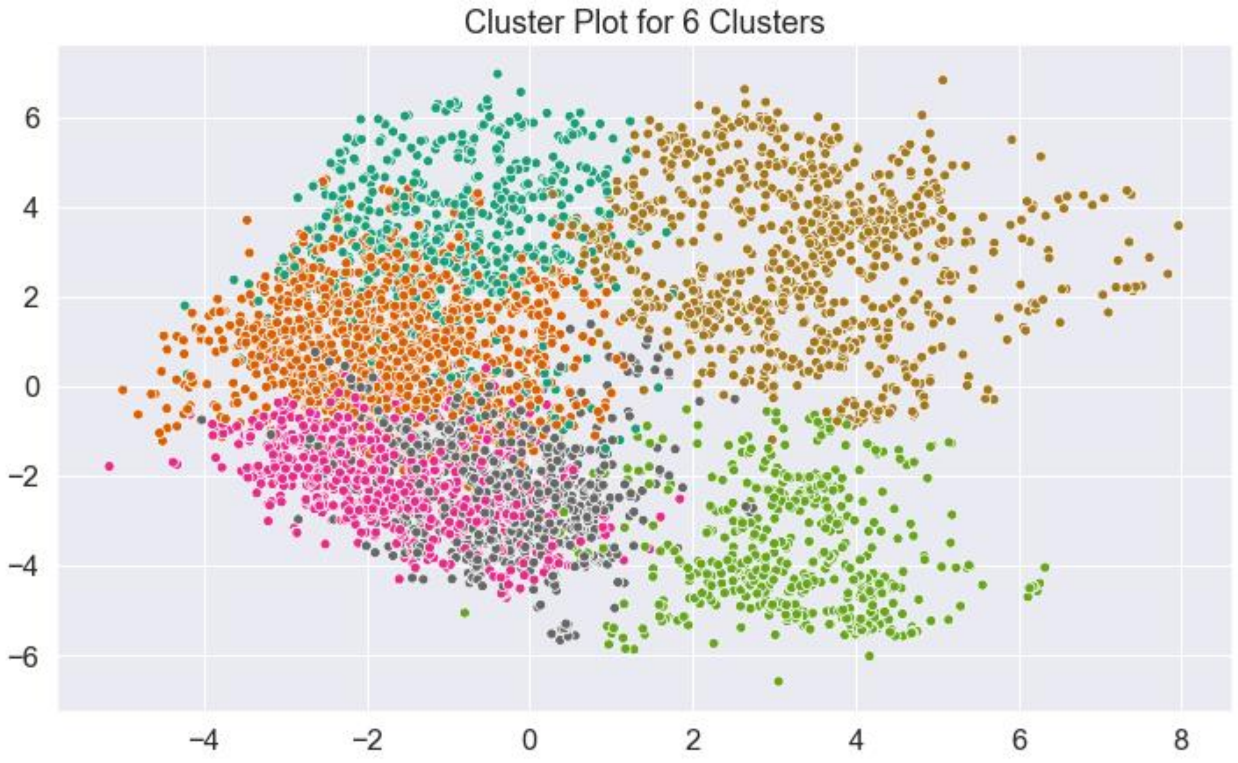


Figure 16: Cluster Plot for 6 Clusters

Comparison of various types of sums of squares are given below for alternative values of *k*.

```
import numpy as np
from scipy.cluster.vq import vq
#!pip install tabulate
from tabulate import tabulate

for i in range(3,7):
    x=KMeans(n_clusters=i, random_state=0).fit(x_full).cluster_centers_
    partition, euc_distance_to_centroids = vq(x_full, x)

    wss=((x_full-x_full.mean(0))**2).sum(0)
    TSS = np.sum((x_full-x_full.mean(0))**2)
    SSW = np.sum(euc_distance_to_centroids**2)
    SSB = TSS - SSW
    WSS=pd.DataFrame([euc_distance_to_centroids**2,KMeans(n_clusters=i, random_s
tate=0).fit(x_full).labels_]).T
    print(tabulate([[i,round(SSB,2),round(TSS,2), np.round(WSS.groupby(1).sum().
values,2),
                    round(SSW,2),WSS.groupby(1).count().values]],
                   headers=['Number of Cluster', 'B/w SS','Total SS','Within SS'
,'Total Within SS','Size']))
    print('\n')
```

```
Number of Cluster    B/w SS    Total SS  Within SS      Total Within SS      Size
-------------------  --------  ----------  ------------  -----------------  --------
        3            50294.1   215751 [[74659.32]         165457           [[2464]
                                       [56898.67]                          [1909]
                                       [33899.31]]                         [1083]]
```

```
Number of Cluster    B/w SS    Total SS  Within SS      Total Within SS       Size
-------------------  --------  ----------  ------------   -----------------       --------
        4          64864.6  215751  [[37960.81]       150887               [[1603]
                                   [54326.2 ]                              [1832]
                                   [31875.56]                              [1058]
                                   [26724.22]]                             [ 963]]


Number of Cluster    B/w SS    Total SS  Within SS      Total Within SS       Size
 -------------------  --------  ----------  ------------   -----------------    --------
        5            74241.2    215751       [[24993.55]     141510          [[1182]
                                             [20801.17]                       [ 743]
                                             [44472.73]                       [1677]
                                             [26995.21]                       [ 940]
                                             [24247.53]]                      [ 914]]


Number of Cluster    B/w SS    Total SS  Within SS  Total Within SS          Size
-------------------    --------  ----------  ------------  -----------------     --------
        6             81317.5     215751      [[23390.43]     134434          [[ 963]
                                             [27003.78]                       [ 935]
                                             [13512.81]                       [ 613]
                                             [15909.7 ]                       [ 884]
                                             [34360.41]                       [1336]
                                             [20256.76]]                      [ 725]]
```

Except for cluster 2 having small overlap with cluster 3, the clusters are well separated. For 6 clusters, all within cluster sums of squares are small compared to the between cluster sum of squares. Hence k = 6 is being recommended as the optimum number of clusters in this case. The last step of clustering procedure is cluster profiling. Based on the various mean values of the attributes in different clusters, the recommender system might be developed.

Let's profile the clusters, the way it was done in the hierarchical clustering to identify the suitable pattern for user segmentation.

```
kmeans = KMeans(n_clusters=6, random_state=0).fit(x)
labels = kmeans.labels_

#Glue back to original
x['kclusters'] = labels
```

```
round(x.groupby('kclusters').mean(),2).T
```

```
kclusters     0     1     2     3     4     5
Churches      1.50  0.79  1.09  1.30  2.09  1.94
Resorts       1.87  1.08  2.14  3.27  2.85  2.34
Beaches       1.96  1.68  1.97  3.49  2.67  2.96
Parks         2.85  1.67  2.13  3.63  2.48  3.63
Theatres      3.16  1.72  2.47  4.18  2.20  3.41
Museums       2.79  1.81  3.39  3.89  2.06  2.76
Malls         3.49  3.09  4.66  3.84  2.21  2.26
Zoo           3.31  1.94  3.49  2.48  1.69  1.78
Restaurants   3.83  2.87  4.59  2.77  2.09  2.08
Bars          3.73  2.85  3.54  2.70  1.57  2.21
```

Local Services	4.14	3.10	2.12	2.27	1.52	2.06
Fast Food	2.37	3.85	1.71	2.06	1.53	1.41
Lodgings	2.70	4.20	1.68	1.93	1.69	1.12
Juice Bars	1.69	4.83	2.15	2.17	2.15	1.02
Art Galleries	1.35	4.01	2.92	1.74	2.19	1.63
Dance Clubs	1.13	1.37	0.68	1.23	1.57	1.37
Swimming Pools	0.82	1.08	0.59	0.60	1.99	0.92
Gyms	0.53	0.83	0.56	0.43	2.01	0.88
Bakeries	0.50	1.20	0.65	0.43	2.56	0.96
Spas	0.48	0.99	0.79	0.59	2.18	1.29
Cafes	0.60	0.71	0.82	0.71	1.31	1.73
View Points	1.69	0.70	0.92	0.87	1.89	4.46
Monuments	1.89	0.68	0.95	0.93	1.86	2.83
Gardens	2.04	0.78	1.02	1.05	2.12	2.31
Cluster	10.02	3.60	1.69	4.14	0.01	6.85

6. Further Discussion and Considerations

6.1 Divisive Clustering Algorithm

This algorithm is rarely used. The basic principle of divisive clustering was published as the DIANA (DIvisive ANAlysis Clustering) algorithm. Initially, all data is in the same cluster, and the largest cluster is split until every object is separate. DIANA chooses the object with the maximum average dissimilarity and then moves all objects to this cluster that are more similar to the new cluster than to the remainder.
(https://github.com/div338/Divisive-Clustering-Analysis-Program-DIANA-)

6.2 Scaling

Standardization or scaling is an important aspect of data pre-processing. All machine learning algorithms are dependent on the scaling of data. Recall that scaling all numerical variables has been strongly recommended in case of PCA and FA. Similarly, for clustering too, scaling is usually applied.

However, in this case we have not applied scaling. The reason being all the variables here are scores, with values between 0 and 5. The variables here are naturally scaled. It was not necessary to smooth them any further.

6.3 Discrete Attributes

The main problem of using discrete attributes in clustering lies in computing distance between two items. All the distance measures defined (Euclidean, Minkowski's etc) are defined for continuous variables only. To use binary or nominal variables in clustering algorithm, a meaningful distance measure between a pair of discrete variables need to be defined. There are several options such as cosine similarity and Mahalanobis Distance. We will not go in depth with these cases. In *Python*, Gower Distance may be used to deal with mixed data type.

Gower distance: For each variable type, a different distance is used and scaled to fall between 0 and 1. For continuous variable Manhattan Distance is used. Ordinal variables are ranked first and then Manhattan distance is used with a special adjustment for ties. Nominal variables are converted into dummy variables and Dice Coefficient (concordance-discordance type coefficient) is computed. Then, a linear combination using user-specified weights (an average) is calculated to create the final distance matrix.(https://pypi.org/project/gower/)

6.4 K-Median Algorithm

This is a variant of *k*-means clustering where instead of minimizing the squared deviations, the absolute deviations from the medians is minimized. The difficulty is that for a multivariate data, there is no universally accepted definition of median. Another option is using a *k*-medoid partitioning.

6.5 K-Medoid Algorithm

In this algorithm *k* representative observations, called medoids, are chosen. Each observation is assigned to that cluster whose medoid is the closest and medoids are updated. A swapping cost is involved when a current medoid is replaced by a new medoid. Various algorithms are developed, namely PAM (partitioning around medoids), CLARA and randomized sampling. (https://github.com/letiantian/kmedoids)

6.6 K-Mode Clustering

Since for categorical data, mean is not defined, *k*-mode clustering is a sensible alternative. Once an optimal number of clusters is determined, initial cluster centres are arbitrarily selected. Each data object is assigned to that cluster, whose centre is closest to it by a suitably defined distance. Note that for cluster analysis the distance measure is all important. After each allocation, the cluster base is updated. A good discussion is given in *A review on k-mode clustering algorithm* by M. Goyal and S Aggarwal published in International Journal of Advanced Research in Computer Science. (https://www.ijarcs.info/index.php/Ijarcs/article/view/4301/4008)
(https://pypi.org/project/kmodes/)

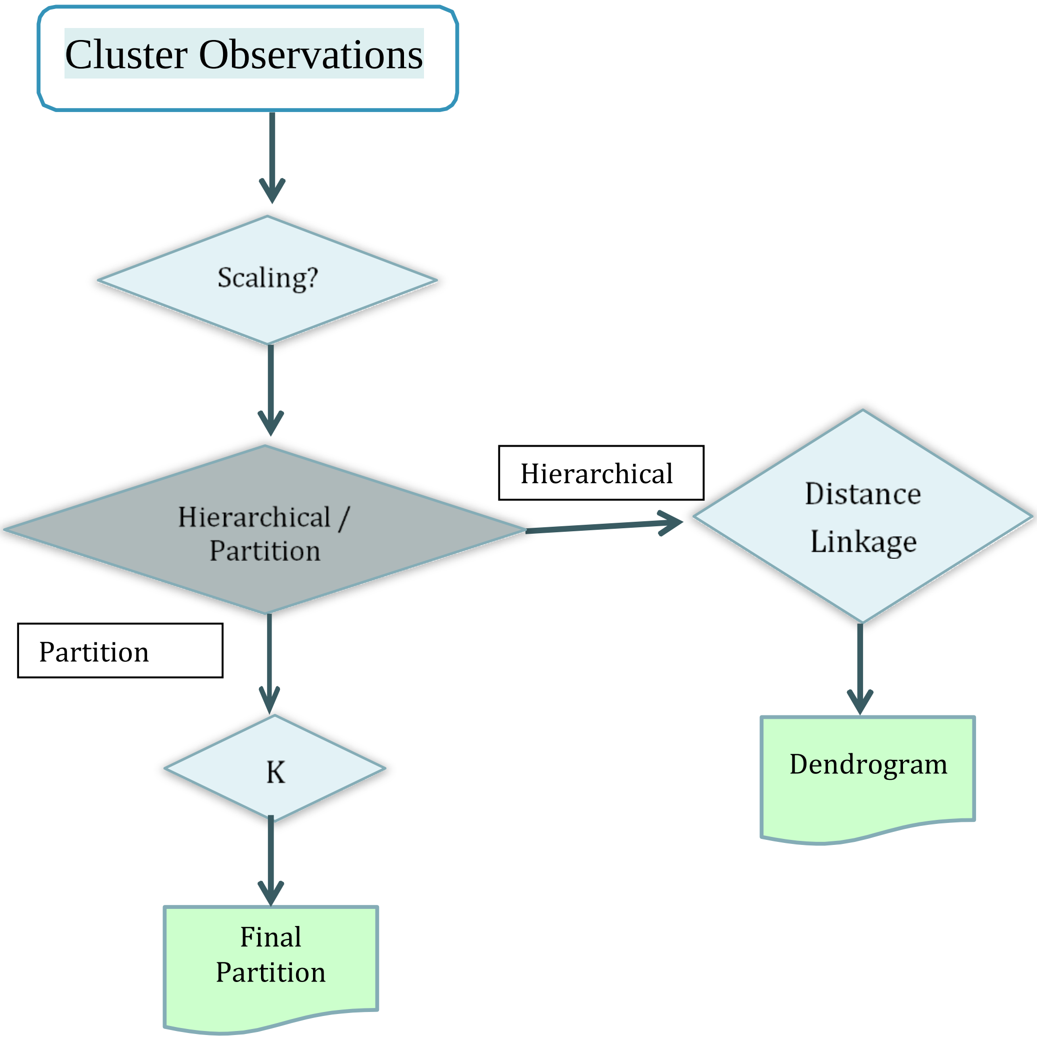


Figure 17: Clustering Flow chart

7. References

- Hennig, C., Meila M., Murtagh F. and Rocci, R (2015). Handbook of Cluster Analysis. Chapman and Hall/CRC.
- Everitt B. S., Landau S., Leese M. and Stahl, D. (2011). Cluster Analysis. 5th Ed. Wiley.

  Kassambara A. (2017). Practical Guide to Cluster Analysis in R. sthda.com.

  https://online.stat.psu.edu/stat505/lesson/14