Week 4 Overview

This week's topic will cover the Data Modeling phase of a data mining process, as depicted in Figure 1.

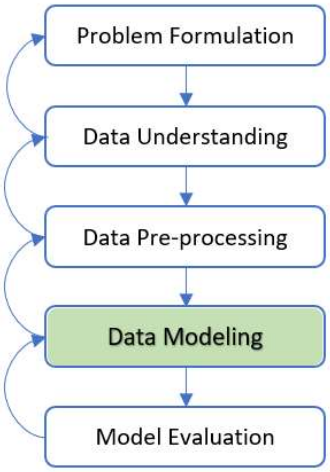

Figure 1 Data Modeling Phase

Data modeling methods may be categorized as either supervised or unsupervised learning methods for modeling data.

In supervised learning methods, there is a particular prespecified target attribute. The learning algorithm is given many training data as examples to provide the values of the target attribute. The algorithm may then learn which values of the target variable are associated with which values of the input attributes as the predictor. For instance, our mini case study of customer churn prespecified the Churn as the target attribute and other attributes such as payment transaction details and demographics as the predictor. In other words, the case study predicts if a customer will churn based on the input attributes (demographics and transactions) as the predictor. We will study supervised learning methods in Week 5.

In unsupervised methods, no target variable is specified. Instead, the data modeling algorithms search for structures and patterns among all the attributes. The most common unsupervised data modeling methods are Clustering and Association Rules. However, unsupervised learning does not imply that human involvement is not

required. On the contrary, realistic Clustering analysis and Association Rules demand substantial human evaluation with domain knowledge.

For the example case of Clustering, political advisors may use the Clustering unsupervised methods to analyze congressional zones to discover the locations of voter clusters that may be responsive to a particular candidate's statements. All relevant demographic attributes (i.e., age, income, ethnicity, and gender) would be input to the Clustering algorithm, with no target variable specified to develop voter profiles for specific purposes such as fund-raising and publicity.

Market basket analysis is a notable use case of Association Rules. For example, businesses may be interested in "which items are purchased together". Similar to Clustering as an unsupervised learning method, no target variable would be specified. The challenge for this use case is that there are too many sale items; searching for all possible associations among items is complicated due to the resulting combinatorial explosion. Nevertheless, association rules algorithms such as the a priori algorithm tackle this challenge effectively.

This week, we will learn the essential concepts of Clustering and Association Rules, then practice how to use Python to implement the unsupervised learning methods for the data modeling phase of data mining.

4.1 Clustering Concept

Clustering refers to grouping similar data points. Clustering is an unsupervised data modeling approach that involves splitting data into clusters or groups. In other words, similar data points, records, observations, objects, or cases (different authors may use different terms) are grouped in a cluster.

A cluster is a collection of records similar to and dissimilar to records in other clusters. Clustering does not predict or estimate the value of a target variable. Instead, Clustering algorithms segment a data set into relatively homogeneous

subgroups or clusters, where the algorithms attempt to maximize the similarity of the data within a cluster and minimise the similarity to records outside the cluster.

There are many categories of clustering algorithms, and we can perform Clustering using many different algorithms. Each of these categories has its unique strengths and weaknesses. Depending on the input data, specific clustering algorithms will result in more naturalistic cluster assignments.

Selecting an appropriate clustering algorithm for a dataset is often tricky due to the number of choices available. Some critical factors that affect this decision include the clusters' characteristics, the dataset's features, the number of outliers, and the number of data points.

Note: Different references may have a different categorization of Clustering algorithms. Examining each of these algorithms requires an extensive duration of time. Therefore, it is beyond the scope of this course to study each of them. Optionally, students can refer to available "Additional Reading Materials (optional)" of this week's content for a more detailed and extensive explanation of different Clustering algorithms.

Regardless of Clustering categories, all clustering algorithms address the identification of groups of data points such that similarity within a group is relatively very high. In contrast, the similarity to data points in other groups is relatively very low. In other words, as shown in Figure 2, Clustering algorithms aim to produce clusters of data points such that the between-cluster variation is enormous. In contrast, the within-cluster variation is as small as possible.

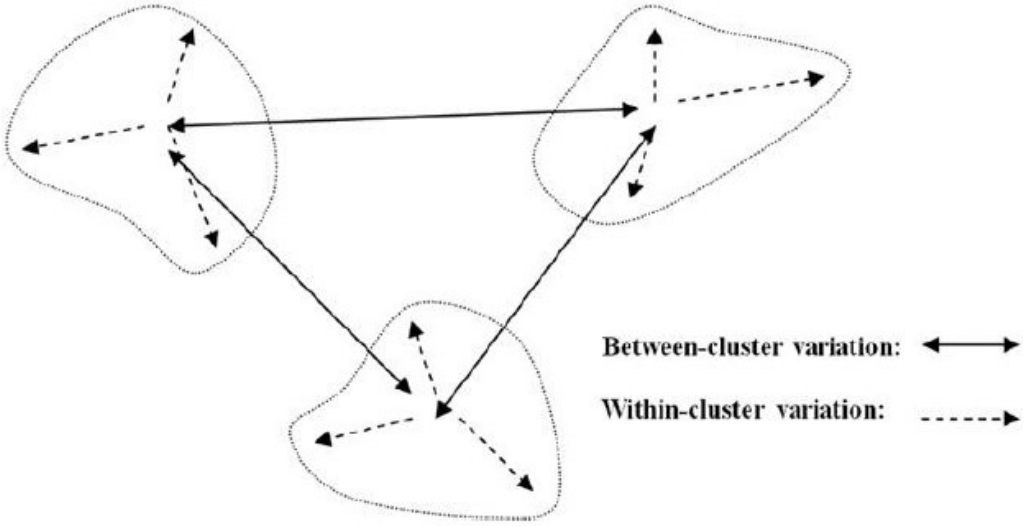

Figure 2 Clusters should have slight within-cluster variation compared to the between-cluster variation [1]

To set the scope for this week's topic, we will explore one of the most commonly used Clustering algorithms, the k-Means Clustering.

We will look at the k-Means algorithm in the following sections.

Reference:

[1] Larose, D.T. and Larose, C.D., 2014. Discovering knowledge in data: an introduction to data mining (Vol. 4). John Wiley & Sons.

4.2 k-Means Clustering

Overview of k-Means Clustering

k-Means Clustering is a method that divides a data set into k clusters. K-Means Clustering is one of the most straightforward and commonly used clustering algorithms. In this method, the user predefines the number of clusters (k) to indicate the number of groups for division in the data set.

The objective of the k-Means Clustering is to find a benchmark data point for each cluster; then assign all the data points to the nearest benchmark data point, i.e.,

denoted as the centroid (cluster's center), which then forms a cluster. The cluster centroid does not have to be an actual data point in the data set. It can be a simulated data point that represents the characteristics of all data points within the cluster.

The k-Means algorithm splits the whole data set into k divisions, where each division's centroid is the cluster's benchmark data point. The data points inside a division belong to the cluster. All data points are associated with the nearest centroid, and the points associated with the centroid form a unique division.

K-Means Clustering produces k splits in n-dimensional space, where n is the number of attributes in a data set. Splitting the data set requires a proximity measurement to be defined. The most commonly adopted measurement for a numeric attribute is the Euclidean distance. Figure 3 illustrates the commonly used example Clustering of the Iris data set with merely the petal length and width numerical type attributes with a k value predefined as 3. The outcome of k-Means Clustering provides a clear division zone for Cluster 1 and narrower zones for Cluster 2 and Cluster 3.

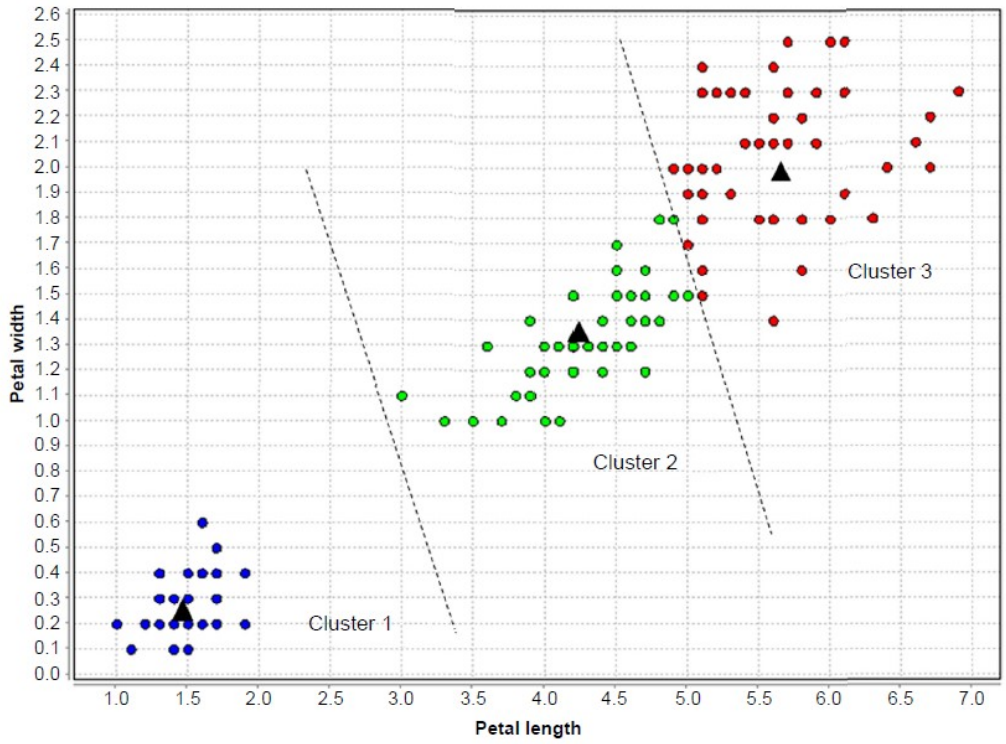

Figure 3 Iris dataset Clusters Example adapted from [1]

How the k-Means Algorithm Works

The k-Means Clustering algorithm is a straightforward algorithm for discovering clusters in data. In summary, the algorithm works as follows:

Step 1: look for the user's predefined number of clusters (k) to group the data points in a data set. Thus, there will be k clusters:

$$C_1, C_2, C_3, C_4, \ldots, C_k$$

Step 2: Randomly assign k records to the initial cluster center locations (i.e., initial seeds).

Step 3: For each data point, find the nearest cluster center.

Step 4: For each k cluster, find the cluster centroid, and update the location of each cluster center to the new centroid value.

Step 5: Repeat steps 3 to 5 until convergence or termination.

The "nearest" measure in step 3 is usually Euclidean distance (Equation 1, where x = x1, x2,…, xm and y = y1, y2,…, ym represent the m attribute values of two data points), although other measures may be applied, such as Minkowski distance (Equation 2, represents the general case of the foregoing two metrics for a general exponent q):

$$\textbf{Equation 1}: d_{Euclidean}(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

$$\textbf{Equation 2}: d_{Minkowski}(x, y) = \sum_i |x_i - y_i|\, q$$

The cluster centroid in step 4 is found as follows. Suppose that we have n data points (a_1, b_1, c_1), (a_2, b_2, c_2),…, (a_n, b_n, c_n), the centroid of these points is the center of gravity of these points and is located at point:

$$\sum a_i/n\,, \sum b_i/n\,, \sum c_i/n$$

For an instant, the points (1, 2, 1), (1, 1, 1), (1, 1, 2), and (2, 1, 1) would have a centroid as follows:

$$\left(\frac{1+1+1+2}{4}, \frac{2+1+1+1}{4}, \frac{1+1+2+1}{4}\right) = (1.25, 1.25, 1.25)$$

The k-Means Clustering algorithm terminates when the centroids no longer change. For all clusters, say C_1 till C_k, the algorithm terminates when all the data points belonging to each cluster center stay in that cluster. Or, the algorithm may terminate when some criterion is met, such as no significant reduction in the average squared errors.

Optionally, for a more detailed explanation of k-Means Clustering, students can refer to [2].

Considerations in using the k-Means Algorithm

Certain aspects can impact the usefulness of the final clusters constructed when using k-Means Clustering. The following elements are essential when solving problems using the K-means clustering algorithm.

1. **Number of clusters (k)**: The predefined number of clusters to group the data points.
2. **Initial Seeds**: Selection of the initial cluster centers can impact the final cluster formation. In other words, the clustering outcome can be different each time the algorithm is executed, even on the same data set.
3. **Outliers**: Cluster formation is susceptible to outliers. Outliers draw the cluster towards itself, thus influencing optimal cluster formation.
4. **Distance Measures**: Different measures that calculate the distance between a data point and cluster center might form different clusters.
5. **Data Type**: The K-Means algorithm only work with numerical data.

Evaluation of Clusters

Cluster validation is a procedure for evaluating the goodness of clustering algorithm results. Generally, the goal of clustering algorithms is to split the data set into clusters of data points, such that:

- The data points in the same cluster are similar as possible;
- The data points in different clusters are highly distinctive.

In other words, it is aimed to have the average distance within a cluster be as close as possible; the average distance between clusters be as far as possible.

The commonly used validation measures reflect the cluster partitions' compactness and separation.

- **Compactness** or cluster cohesion: Measures how close the objects are within the same cluster. A lower within-cluster deviation suggests good compactness, indicating good clustering.
- **Separation**: Measures how well-separated a cluster is from other clusters, such as the distances between cluster centers.

In the following, we will explore the two commonly used validation methods for assessing the goodness of clustering using k-Means algorithms: 1) the Elbow Curve and 2) the Silhouette analysis Methods. We can also use these methods to identify the optimal number of clusters (k) in the data.

1. **Elbow Curve Method**

The Elbow Curve method executes k-Means Clustering on a dataset for a range of k values.

The Elbow Curve method first calculates average distances to the centroid for each k value across all data points. For each value of k, the Elbow Curve method calculates the WCSS (Within-Cluster Sum of Square, the Sum of squared distance between each point and the centroid in a cluster). As the number of clusters increases, the WCSS value will decrease. WCSS value is largest when k = 1. Then, it plots these points and finds the point where the average distance from the centroid drops suddenly. It is called "Elbow" at this drop point.

Figure 4 shows that the graph rapidly drops at a point, resulting in an elbow shape. The graph starts to move nearly parallel to the X-axis after this point. The k value approximating this point is the optimal K value as the optimal number of clusters.

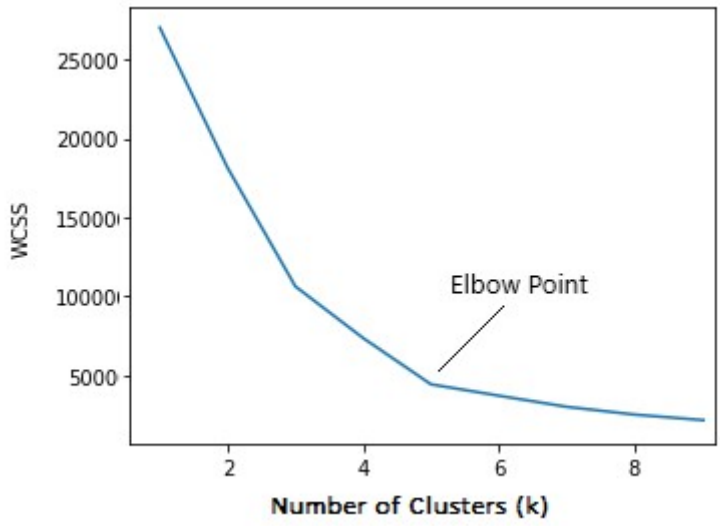

Figure 4 Example of Elbow Curve Method Plot

In the Figure 4 plot, the elbow is at k=5, which means the Sum of Squared distances drops suddenly, suggesting the optimal k for this dataset is 5.

2. Silhouette Analysis

The Silhouette coefficient measures how similar a data point is within a cluster (i.e., the cohesion between data points in a cluster) compared to other clusters (i.e., the separation between clusters). In other words, it indirectly estimates the average distance between clusters.

Figure 5 demonstrates the average distance between i (i.e., the data point with orange color) and all the other data points in the cluster to which i belongs.

Figure 5 The average distance between i and all the other data points in the cluster

Figure 6 presents the average distance from i (i.e. the data point with blue color) to all clusters to which i does not belong.

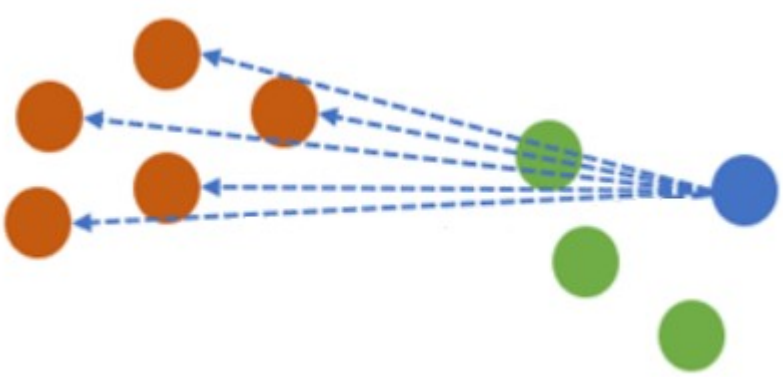

Figure 6 The average distance from i to all clusters

The Silhouette score for a set of sample data points is used to measure how dense and well-separated the clusters are. The Silhouette score considers the intra-cluster distance between the data point and other points within the same cluster (as in Figure 5) and the inter-cluster distance between the data point and the next nearest cluster (as in Figure 6).

The Silhouette method selects a range of k values and then plots Silhouette coefficient for each value of k. Steps to calculate the silhouette coefficient, S(i) of a data point (i):

- a_i: The average distance of that data point (i) with all other points in the same clusters.
- b_i: The average distance of that data point (i) with all the data points in the nearest cluster to its cluster.
- S_i: Silhouette coefficient using the following Equation 3.

$$\textbf{Equation 3}: S_i = \frac{b_i - a_i}{max(b_i\,, a_i)}$$

The method calculates the average Silhouette score for every k. Then plot the graph between the average Silhouette score and k. Figure 7 presents an example of a Silhouette plot.

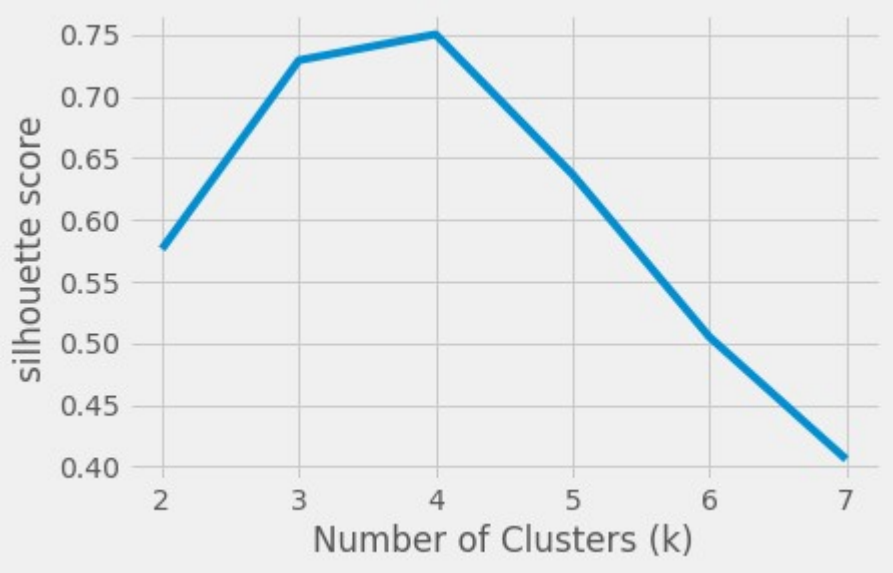

Figure 7 Example of a Silhouette Plot

In Figure 7, we observe that the Silhouette score is maximized at k = 4, suggesting 4 clusters is the optimum number to take. The silhouette method is often utilized with the Elbow Curve method for a more confident decision to select the k value.

The following explains the Silhouette coefficient value:

- The Silhouette coefficient value is between -1 and 1.
- A score of 1 denotes the best separation between clusters to which it belongs and far away from the other clusters.
- The worst value is -1. Values close to 0 indicate overlapping clusters.

In the next section, we will use the Iris example dataset to implement the k-Means clustering algorithm to apply the unsupervised learning concept for the data modeling phase in a data mining process.

References:

[1] Kotu, V. and Deshpande, B. 2014. Predictive analytics and data mining: concepts and practice with rapidminer. Morgan Kaufmann.

[2] Larose, D.T. and Larose, C.D. 2014. Discovering knowledge in data: an introduction to data mining. John Wiley & Sons.

4.2.1 k-Means Implementation using Python

To practically apply to the k-Means Clustering concept, we use a commonly used Iris dataset (iris.csv) to demonstrate the implementation. The Iris dataset has four attributes representing the length and width of three species' sepals and petals: Setosa, Virginica, and Versicolor.

Use the following Python codes to access the Iris dataset and transform the categorical Species attribute to numerical data type.

```
import pandas as pd
df = pd.read_csv('data/Iris.csv')

#transform categorical 'Species' data type to numeric
df.loc[df['Species'] == 'Iris-setosa', 'Species'] = 1
df.loc[df['Species'] == 'Iris-versicolor', 'Species'] = 2
df.loc[df['Species'] == 'Iris-virginica', 'Species'] = 3
```

using the KMeans() function, we first use the Elbow Curve method to identify the optimal k (number of clusters) for the Iris dataset. The following example codes produce an Elbow Curve plot. We use a user-defined variable named wcss[] to contain the WCSS (Within-Cluster Sum of Square, the Sum of squared distance between each point and the centroid in a cluster) generated at each k via the inertia_ object.

```
import numpy as np
import os
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans

# identify optimal k using the Elbow Curve method
wcss = []
for i in range(1, 11):   # specified a range for the Elbow Curve x-axis
   model_elbow = KMeans(
      n_clusters=i,
      init='k-means++',
      max_iter=300,
      n_init=10,
      random_state=10)
   model_elbow.fit(df)
   wcss.append(model_elbow.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('Elbow Curve Method')
plt.xlabel('Number of clusters')
```

```
plt.ylabel('WCSS')
plt.show()

strFile = "plot_elbow.png"
if os.path.isfile(strFile):
   os.remove(strFile)
plt.savefig(strFile)
plt.clf()
```

After running the above codes, we can observe from the Elbow Curve plot (plot_elbow.png) that the elbow is at k=3, which means the Sum of Squared distances drops more suddenly compared to other points, suggesting the optimal k for this dataset is 3.

Now we can further confirm if k=3 is also the optimal number of clusters using the Silhouette method using the silhouette_score() function after forming several k clusters for comparison. The Elbow Curve method indicates that k=3 is the optimal number. Therefore, we initiate the range of k to validate to include k=3, for example, in the range of [2, 3, 4].

The following Python codes calculate the Silhouette score using the silhouette_score() function for each k, then generate a Silhouette plot based on the score of each k in the range.

```
#identify the Silhouette scores according a range to include Elbow Curve finding
from sklearn.metrics import silhouette_samples, silhouette_score

range_n_clusters = [2,3,4]
# take a range to include ELbow Curve finding

silhouette_avg_n_clusters = []
for n_clusters in range_n_clusters:
    # Initialize the clusterer with n_clusters value and a random generator seed 10
for reproducibility.
    model_silhouette = KMeans(n_clusters=n_clusters, init='k-means++', max_iter=300,
n_init=10, random_state=10)
    model_silhouette = model_silhouette.fit_predict(df)

    # The silhouette_score gives the average value for all the samples,ie. density and
separation of the formed clusters
    silhouette_avg = silhouette_score(df, model_silhouette)
    print("For n_clusters =", n_clusters,"The average silhouette_score is :",
silhouette_avg.round(4))
    silhouette_avg_n_clusters.append(silhouette_avg.round(4))
```

```
# alternately we can plot the k and respecitve silhouette score
plt.plot(range_n_clusters, silhouette_avg_n_clusters)
plt.title('Silhouette Method')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.show()
strFile = "plot_silhouette.png"
if os.path.isfile(strFile):
   os.remove(strFile)
plt.savefig(strFile)
plt.clf()
```

After running the codes above, we can observe from the Silhouette plot generated (plot_silhouette.png) that the Silhouette score is maximized at k = 2 (having the highest Silhouette score), suggesting 2 clusters is the optimum number to take.

To decide if k=2 or k-3 is a better choice, we can use scatter plots to compare how the data points are distributed across different k numbers, taking the k=3 suggested by the Elbow Curve method, and k=2 indicated by the Silhouette method.

The following Python codes show examples of how we can produce the scatter plots for comparison.

```
# Plot Elbow Curve k clusters
k = 3 # k=3 suggested by the Elbow Curve
model_k = KMeans(n_clusters=k, init='k-means++', max_iter=300, n_init=10,
random_state=0) #random_state=42)
df_kmeans = model_k.fit_predict(df)

ax1 = fig.add_subplot(1, 2, 1, projection='3d')
ax1.scatter(df.loc[df_kmeans == 0]['SepalLengthCm'], df.loc[df_kmeans ==
0]['PetalLengthCm'], df.loc[df_kmeans == 0]['PetalWidthCm'],
            s=600, c='lightgreen', marker='s', edgecolor='black', label='Cluster 1')
ax1.scatter(df.loc[df_kmeans == 1]['SepalLengthCm'], df.loc[df_kmeans ==
1]['PetalLengthCm'], df.loc[df_kmeans == 1]['PetalWidthCm'],
            s=600, c='yellow', marker='o', edgecolor='black', label='Cluster 2')
ax1.scatter(df.loc[df_kmeans == 2]['SepalLengthCm'], df.loc[df_kmeans ==
2]['PetalLengthCm'], df.loc[df_kmeans == 2]['PetalWidthCm'],
            s=600, c='red', marker='*', edgecolor='black', label='Cluster 3')
#ax1.view_init(30, -70)
ax1.set_xlabel("SepalLengthCm", fontsize=50)
ax1.set_ylabel("PetalLengthCm", fontsize=50)
ax1.set_zlabel("PetalWidthCm", fontsize=50)
ax1.set_title("Elbow Curve Clusters", fontsize=70)
ax1.legend(loc='upper center', prop={'size':45}, bbox_to_anchor=(0.5, -0.08), ncol=3)
```

```
# Plot Silhouette k clusters
k = 2 # k=2 suggested by Silhouette
model_k = KMeans(n_clusters=k, init='k-means++', max_iter=300, n_init=10,
random_state=0) #random_state=42)
df_kmeans = model_k.fit_predict(df)

ax2 = fig.add_subplot(1, 2, 2, projection='3d')
ax2.scatter(df.loc[df_kmeans == 0]['SepalLengthCm'], df.loc[df_kmeans==
0]['PetalLengthCm'], df.loc[df_kmeans == 0]['PetalWidthCm'],
            s=600, c='lightgreen', marker='s', edgecolor='black', label='Cluster 1')
ax2.scatter(df.loc[df_kmeans == 1]['SepalLengthCm'], df.loc[df_kmeans==
1]['PetalLengthCm'], df.loc[df_kmeans == 1]['PetalWidthCm'],
            s=600, c='yellow', marker='o', edgecolor='black', label='Cluster 2')
#ax1.view_init(30, -70)
ax2.set_xlabel("SepalLengthCm", fontsize=50)
ax2.set_ylabel("PetalLengthCm", fontsize=50)
ax2.set_zlabel("PetalWidthCm", fontsize=50)
ax2.set_title("Silhouette Clusters", fontsize=70)
ax2.legend(loc='upper center', prop={'size':45}, bbox_to_anchor=(0.5, -0.08), ncol=3)

fig.show()

strFile = "plot_compare_clusters.png"
if os.path.isfile(strFile):
   os.remove(strFile)
plt.savefig(strFile)
plt.clf()
```

Run the above codes and observe the scatter plots generated and saved in the "plot_compare_clusters.png" file. The following Figure 8 shows the results of the scatter plots.

Figure 8. Data Points Distributions in Different k Clusters based on Elbow Curve and Silhouette Methods

We can see that k=3 (suggested by the Elbow Curve method) can separate different clusters farther than k=2 (suggested by the Silhouette method). Thus, for this Iris example, k=3 is a better choice as the optimal number of clusters.

To interpret the three clusters (k=3) in Figure 8, we observe that Cluster 1 contains the data points that have a shorter size in Petal length and width compared to the other two clusters, and its Sepal length is concentrated in a shorter length. Whereas Cluster 3 poses data points having larger Petal length and width than the other clusters, with various Sepal lengths. Cluster 2 demonstrates a group of data points with a larger Petal length than Cluster 1 but a smaller Petal width than Cluster 3.

References:
The following links from the official website of sci-kit learn that I find helpful for implementing a clustering approach using Python codes.

- To understand all the possible parameters of k-Means and their descriptions: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans
- Example of k-Means clustering: https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_iris.html?highlight=cluster
- For additional reference of different clustering algorithms (optional): https://scikit-learn.org/stable/modules/clustering.html?highlight=cluster

4.3 Association Rules

The association rule for data mining is discovering which data characteristics "go together", which means that if a particular attribute appears, the other attribute(s) also occurs with specific values. A common form of this application is called market

basket analysis, and the association aims to uncover rules for quantifying the relationship between two or more attributes.

In short, association rules explain data in the formation of:

$$\text{Antecedent} \Rightarrow \text{Consequent}$$

Which implies, "IF antecedent THEN consequent."

There are many possible association rules derivable from any given dataset, possibly most of them of little or no value, so it is typical for association rules to be expressed with some assessment measurement information indicating how reliable they are, for example:

$$(A > 45 \text{ and } B = \text{'Yes'}) \Rightarrow (C < 62) \qquad [\text{Confidence} = 0.7]$$

The above example implies that if $A > 45$ and $B =$ 'Yes' (i.e., Antecedent), then we are given 70% (confidence = 0.7) confidence in the value $C < 62$.

Assessment measurements will be associated with the rule to evaluate the reliability. The commonly used measures are Support, Confidence, and Lift.

Note: We will learn the mathematical equations of the Support, Confidence, and Lift measures in later sections.

For example, a particular supermarket may find that, of the 10000 customers shopping on a specific day, 2500 bought item X. Also, those 2500 who bought item X, 550 bought item Y. Thus, the association rule would be "If a customer buys item X, then buy item Y," with a support of 2500/10000 = 25% and a confidence of 550/2500 = 22%.

Suppose the marketing department knows the purchases made by all the customers at a supermarket within a certain period; it can find relationships to help the store promote its products more effectively.

For example, the rule "IF milk THEN cheese (confidence = 0.7)" can be notated as follows:

Milk ⇒ Cheese [confidence = 70%]

The above rule indicates that 70% of confidence that customers who buy milk also buy cheese. Thus, it would be sensible to place cheese closer to the milk counter if customer convenience were the primary concern or to separate them to encourage impulse buying of other products if profit were a more important concern.

Common examples of real-world association rules applications include:

- Investigating a company's subscribers proportion that responds positively to an offer or promotion
- Discovering items in a supermarket that customers bought together and which items they never bought together.

In the following, we will look at the concept of how association rules work and the assessment measurements for evaluating the reliability of the rules generated.

How Association Rules Work

Fundamental market basket association analysis deals with the occurrence of one item with another. The number of possible association rules grows exponentially in the number of attributes. For example, suppose that a small store has only a hundred different items, and a customer could buy or not buy any combination of those hundred items. Excluding the empty item set, the calculation equation will be:

$$2^d - 1$$

Where d = number of items within an itemset.

Then there will be enormous possible association rules that await the search algorithm. To provide a less complicated view to visualize the complexity of the association rules that may grow, we take an example with only five items shown in Figure 8.

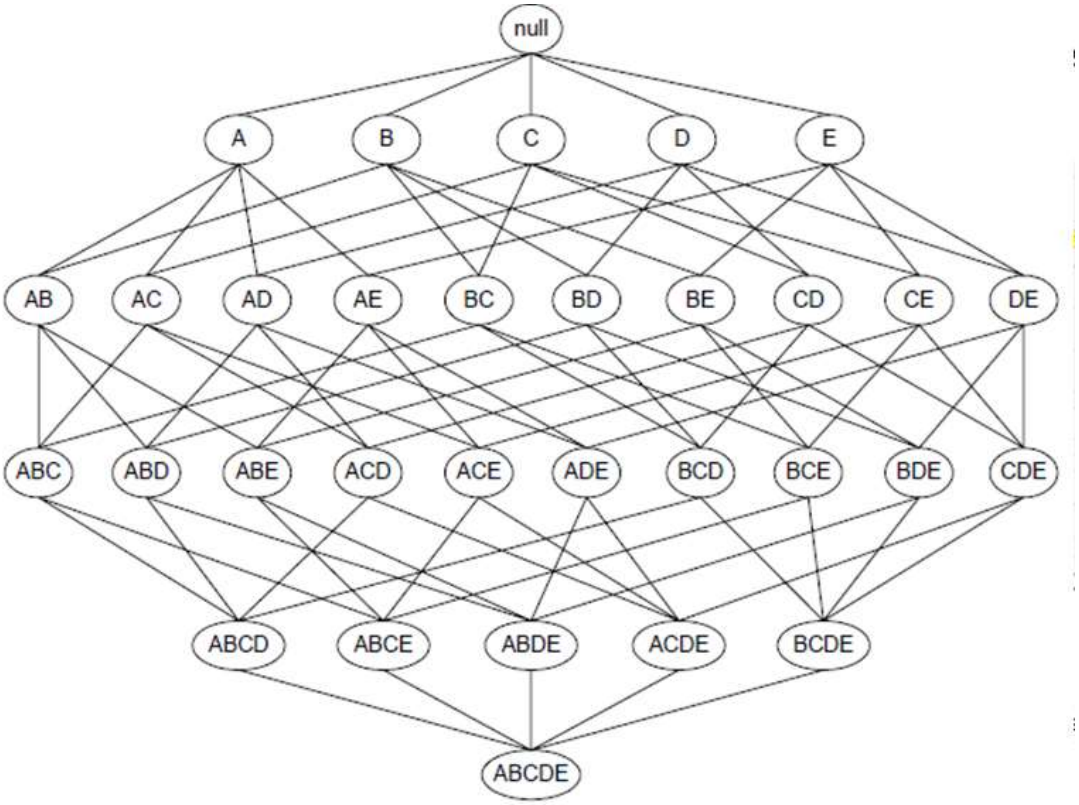

Figure 8 Frequent Sets Generation adapted from [1]

Let's say we have d = 5 means five items in an itemset, {A, B, C, D, E} as shown in Figure 8 as an example, the possible combination of itemsets based on these five items will be based on the equation 2^d-1 (excluding empty set), results in possible candidate item set = 2^5-1 = 25 - 1 = 32 - 1 = 31.

1. Association Rules' Fundamental Elements

The following terms represent the fundamental elements of an association rule:

- **Itemset**: A collection of one or more items; for example, {Bread, Milk, Diaper, Eggs, Beer, Coke} is an item set containing six items.
- **Support count**: Frequency of the occurrence of an item set
- **Frequent itemset**: An itemset whose support is greater than or equal to the user-predefined minimum threshold (denoted as 'minsup')

We will learn how these elements are utilized in the processing logic of Association Rules methods next.

2. Apriori algorithms

The commonly used association rules method is Apriori algorithms. The Apriori algorithms for mining association rules use structure within the rules themselves to reduce the search to a more manageable size. In summary, the fundamental logic of Apriori algorithms for generating frequent item sets is summarised as the following concise pseudo-algorithm:

```
# develop frequent itemsets based on support count
1. Let k = 1 (k is the number of items within an item set)
2. Develop frequent itemsets of length 1.
3. Develop frequent itemsets of length 1.
4. Repeat until no new frequent itemsets are found.
5. Develop length (k+1) candidate itemsets from length k frequent itemsets.
6. Eliminate candidate itemsets containing subsets of length k that are
infrequent.
7. Count the support of each candidate by scanning the database
8. Remove candidates that are infrequent, leaving only those that are
frequent.

# perform rules filtering basedon interst measures
9. Perform further generation of high confidence rules from each frequent
itemset,
   where each rule is a binary partitioning of a frequent itemset.
10.Finally, the algorithm generates and filters the rules based on the
interest measure.
```

The following sections further elaborate on the tasks of frequent itemsets and rules generation.

Apriori Algorithms: Frequent Itemset Generation

We begin with a simple sample to explain the fundamental Apriori algorithms concept. Suppose a local store offers the following items: {Bread, Milk, Diaper, Eggs, Beer, Coke}. Denote this item set as I. For explaining the association rules concept using this example, we are only interested in whether or not a particular item is purchased instead of how much each item is purchased.

Given the transactions listed in Table 1, find rules that will anticipate the occurrence of an item based on the occurrences of other items in the transaction within a specific period. For example, suppose Table 1 lists the transactions made.

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Table 1 Itemset data adapted from [1]

Let's say we want to find out association rules that contain frequent itemsets with a user-predefined minimum Support count (denotes as minsup) = 3.

Figure 9 diagrammatically illustrates the process of the Apriori algorithm generating frequent item sets. The process starts with counting the occurrence of each item in all transactions, then removing those items that do not have the occurrence satisfying the minsup threshold. The process will then repeat the similar task of counting-removing items based on the minsup threshold by adding one more item to the itemset for each iteration.

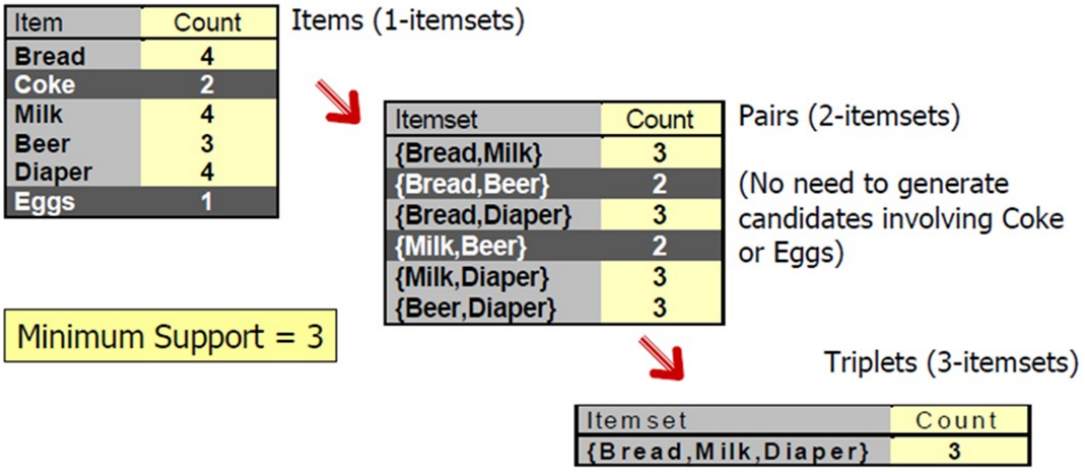

Figure 9 Process of Generating Frequent Itemsets adapted from [1]

After the process, there are five frequent itemsets (minsup >= 3) identified:

Frequent itemset 1: {Bread, Milk}

Frequent itemset 2:	(Bread, Diaper}
Frequent itemset 3:	{Milk, Diaper}
Frequent itemset 4:	{Beer, Diaper}
Frequent itemset 5:	{Bread, Milk, Diaper}

Apriori Algorithms: Rules Generation

A rule is an implication expression of the A ⇒ B, where A (as Antecedent) and B (as Consequent) are itemsets:

A ⇒ B

Antecedent Consequent

In rule generation, essential measures used to evaluate and qualify a rule are as follows:

- **Support**: the fraction of transactions containing items A and B (indicates popularity).
- **Confidence**: the conditional probability that a randomly selected transaction will contain item A, given that item B also exists.
- **Lift**: the probability of item A and B occurring together to the multiple of the two separate probabilities for itemsets A and B (indicates the power strength of an association rule).

The following expresses the equation of each measure stated above:

$$Support = \frac{Frequency\ of\ occurence\ (A \wedge B)}{Total\ number\ of\ transactions}$$

$$Confidence = \frac{Frequency\ of\ occurence\ (A \wedge B)}{Frequency\ \ occurence\ of\ A}$$

$$Lift = \frac{Probability\ of\ occurence\ (A \wedge B)}{Probability\ of\ occurence(A) \cdot Probability\ of\ occurence(B)}$$

Given the equations above to calculate the measures, let's say we evaluate one of the frequent itemsets identified that satisfies the minsup threshold>=3, i.e. Frequent itemset 4, {Beer, Diaper}. The {Beer, Diaper} itemset may possibly form the following rules:

Rule 1: {Beer} ⇒ {Diaper}

Rule 2: {Diaper} ⇒ {Beer}

Based on the two rules above, we can now apply the equations to derive the outcomes of the measures (i.e., Support, Confidence, and Lift) to evaluate the rules:

Based on Table 1 example transactions:

Total number of transactions in Table 1 = 5

Frequency of occurence (Beer and Diaper) = 3

Frequency of occurence (Beer) = 3

Frequency of occurence (Diaper) = 4

Thus, for Rule 1: {Beer} ⇒ {Diaper}

Support = 3/5 = 0.60

Confidence = 3/3 = 1.00

Lift = 3/(3/5 * 4/5) = 3/(0.6 * 0.8) = 6.25

And, for Rule 2: {Diaper} ⇒ {Beer}

Support = 3/5 = 0.60

Confidence = 3/4 = 0.75

Lift = 3/(4/5 * 3/5) = 6.25

Based on the measures calculated above, we can conclude that Rule 1 {Beer} ⇒ {Diaper} implies a 100% of confidence that customers who buy beers also buy diapers. Whereas Rule 2 shows lower confidence compared to Rule 1. Support and Lift measures are the same for both rules.

The acceptable reliability of a rule depends on the user-defined threshold for each measure. For example, let's say the local store owner expects the Support of 55%, Confidence of 80%, and a Lift of 5, then Rule 1 will be considered a useful rule to suggest strategies for improving item sales. Rule 2 will be deselected because it does not pass the 80% confidence threshold.

Based on Rule 1 {Beer} ⇒ {Diaper}, we can express and interpret it as follow:

Beer ⇒ Diaper [Support = 60%, Confidence = 100%, Lift = 6.25]

- 60% percent of total transactions contain both beer and diapers.

- 100% percent chance someone will buy diapers when they buy beer.
- There is a 6.25 times increase in the expectation that someone will buy diapers when we know they bought beer.

In the next section, we will use an example dataset to implement the Apriori algorithm to apply the Unsupervised learning concept for the market basket analysis.

References:

[1] Tan, P.N., Steinbach, M. and Kumar, V., 2016. Introduction to data mining. Pearson Education India.

4.3.1 Association Rules Implementation using Python

To practically apply the Apriori Algorithm concept, we use an example dataset containing retail transactions to demonstrate the implementation. The dataset has four attributes representing the length and width of three species' sepals and petals: Setosa, Virginica, and Versicolor.

Use the following Python codes to access the retail transaction dataset fretail.xlsx located in the /data/ folder. The first few lines of Python codes are meant to ignore any warning messages.

```
# ignore all warning messages
import warnings
warnings.filterwarnings('ignore')
def warn(*args, **kwargs):
    pass
warnings.warn = warn

# access retail dataset
import pandas as pd
df = pd.read_excel('data/fretail.xlsx')
```

We then need to select the attributes and set the data types correctly for further processing by the Apriori algorithm later. In this example, the Description and InvoiceNo attributes contain the items and invoice numbers of transactions that bear the transaction items. Based on the concept of Apriori Algorithms for market basket

analysis, we need these two attributes' information to form the dataset for further analysis.

```
# set the attributes and data types for asosciation rules analysis
df["Description"] = df["Description"].astype(str)
df["InvoiceNo"] = df["InvoiceNo"].astype(str)
```

To calculate the frequency of each item in a transaction, we use the sum() function to count to frequency and group by invoice number representing a transaction.

```
# calculate total item quantity of an InvoiceNo
df = df.groupby(["InvoiceNo","Description"])["Quantity"].sum()
```

After counting the frequency of each item in each transaction contained in the dataset, the structure requires a reshape so that the frequency of all the items within the dataset matching each transaction can be captured. Those items that do not appear in a particular transaction will have a NULL frequency. We can use the unstack() function to realize the structure reshape.

Figure 10 below diagrammatically elaborates the structure change after reshaping:

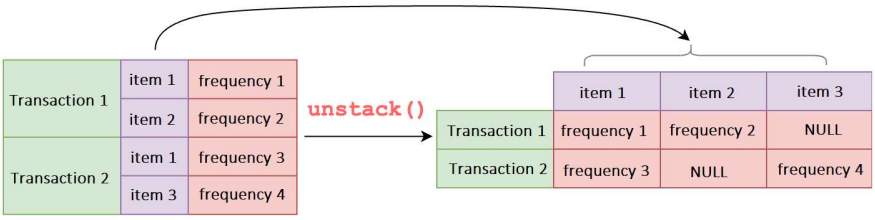

Figure 10 Restructure Dataset using the unstack() Python function

For those items that do not have any quantity (i.e. frequency), replace them with the value 0 using the fillna() function.

```
# to restrucutre the shape of dataset
df = df.unstack()

# to replace null with 0 value
df = df.fillna(0)
```

The concept of Apriori algorithms only concerns if an item occurs (i.e., becomes a binary value of either 0 or 1) in a transaction instead of the number of occurrences therefore, we convert the value to 1 if the occurrence is more or equal to 1, else 0 using the applymap() function.

```
# assign values to binary either 1 or 0
```

```
def reduce_to_binary(qty) :
    if qty >= 1 :
        return 1
    if qty <= 0 :
        return 0
df = df.applymap(reduce_to_binary)
```

Once the dataset format is probably processed to suit the Apriori Algorithms, we can now perform the Association Rules method for data modeling. The following Python codes show examples of how we can implement an Apriori Algorithm for the Association Rules method using the Apriori library's association_rules() function.

In this example, we user-defined the minsup = 0.1 as the measure threshold for generating frequent itemsets. We specify the columns to observe a rule, including the antecedents, consequent, support, confidence and lift. The result of the frequent itemsets is sorted by the lift measure descendingly.

```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

# specify minsup = 0.07 to generte frequent itemsets
frequent_items = apriori(df, min_support=0.1, use_colnames=True)

# perform association rules modeling
rules = association_rules(frequent_items, metric="lift", min_threshold=1)

# select columns to observe a rule that containing frequent itemsets
rules = rules[['antecedents','consequents', 'support','confidence','lift']]

# sort rules by lift measure
rules = rules.sort_values(by = ["lift"],ascending=False)

# print the rules that contian frequent itemsets with threshold of minsup = 0.1
print(rules)
```

After running the above codes, we can observe from the print(rules) result in Figure 11 that there are 14 rules with frequent itemsets satisfied the minsup = 0.1 threshold.

Figure 11 Rules with Frequent Itemsets

	antecedents	consequents	support	confidence	lift
12	(SET/6 RED SPOTTY PAPER PLATES)	(SET/6 RED SPOTTY PAPER CUPS)	0.104121	0.960000	8.195556
13	(SET/6 RED SPOTTY PAPER CUPS)	(SET/6 RED SPOTTY PAPER PLATES)	0.104121	0.888889	8.195556
10	(POSTAGE)	(ROUND SNACK BOXES SET OF4 WOODLAND)	0.125813	0.193333	1.437527
11	(ROUND SNACK BOXES SET OF4 WOODLAND)	(POSTAGE)	0.125813	0.935484	1.437527
2	(POSTAGE)	(PLASTERS IN TIN CIRCUS PARADE)	0.125813	0.193333	1.350404
3	(PLASTERS IN TIN CIRCUS PARADE)	(POSTAGE)	0.125813	0.878788	1.350404
7	(POSTAGE)	(RABBIT NIGHT LIGHT)	0.140998	0.216667	1.349775
6	(RABBIT NIGHT LIGHT)	(POSTAGE)	0.140998	0.878378	1.349775
8	(RED TOADSTOOL LED NIGHT LIGHT)	(POSTAGE)	0.134490	0.873239	1.341878
9	(POSTAGE)	(RED TOADSTOOL LED NIGHT LIGHT)	0.134490	0.206667	1.341878
4	(POSTAGE)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.117137	0.180000	1.238507
5	(PLASTERS IN TIN WOODLAND ANIMALS)	(POSTAGE)	0.117137	0.805970	1.238507
0	(LUNCH BAG RED RETROSPOT)	(POSTAGE)	0.104121	0.800000	1.229333
1	(POSTAGE)	(LUNCH BAG RED RETROSPOT)	0.104121	0.160000	1.229333

Once we have the list of rules containing frequent itemsets, we can use other measures such as Support, Confidence, and Lift to generate useful rules. The following example codes expect a threshold of Lift >=5, Confidence >= 0.9, and Support >=0.1.

```
rules = rules[ (rules["lift"] >= 5) & (rules["confidence"] >= 0.9) &
(rules["support"] >= 0.1)] print(rules)
```

After applying a tighter threshold for the measures above, we observe that there are only two rules qualified as useful rules that satisfy the threshold requirements, as shown in Figure 12.

	antecedents	consequents	support	confidence	lift
12	(SET/6 RED SPOTTY PAPER PLATES)	(SET/6 RED SPOTTY PAPER CUPS)	0.104121	0.96	8.195556

Figure 12. The rule is generated based on a user-defined threshold for each measure

The rule generated from the example retail transaction dataset in Figure 12 implies the following:

- 10.4% percent of total transactions contain both itemsets (SET/6 RED SPOTTY PAPER PLATES) and (SET/6 RED SPOTTY PAPER CUPS).
- 96.0% percent chance someone will buy (SET/6 RED SPOTTY PAPER CUPS) when they picked (SET/6 RED SPOTTY PAPER PLATES).
- There is an 8.96 times increase in the expectation that someone will buy (SET/6 RED SPOTTY PAPER CUPS) when we know they bought (SET/6 RED SPOTTY PAPER PLATES).

4.4 Discussion Forum Activity

Time: 30 minutes

Purpose: The purpose of this activity is to determine optimal k, and plot clusters to observe the data points.

Task: By using the dataset(s) that the you have chosen in Week 2 Activity 1, perform the following tasks:

- Use any three numerical attributes with rationale (according to the problem and data understanding) from the dataset chosen, apply k_Means Clustering to determine optimal k, and plot clusters to observe the data points. Share your findings in the Discussion Board.