PharmaSUG 2019 - Paper ST-183

Cluster Analysis: What It Is and How to Use It

Alyssa Wittle and Michael Stackhouse, Covance, Inc.

ABSTRACT

A Cluster Analysis is a great way of looking across several related data points to find possible relationships within your data which you may not have expected. The basic approach of a cluster analysis is to do the following: transform the results of a series of related variables into a standardized value such as Z-scores, then combine these values and determine if there are trends across the data which may lend the data to divide into separate, distinct groups, or "clusters". A cluster is assigned at a subject level, to be used as a grouping variable or even as a response variable. Once these clusters have been determined and assigned, they can be used in your analysis model to observe if there is a significant difference between the results of these clusters within various parameters. For example, is a certain age group more likely to give more positive answers across all questionnaires in a study or integration? Cluster analysis can also be a good way of determining exploratory endpoints or focusing an analysis on a certain number of categories for a set of variables. This paper will instruct on approaches to a clustering analysis, how the results can be interpreted, and how clusters can be determined and analyzed using several programming methods and languages, including SAS, Python and R. Examples of clustering analyses and their interpretations will also be provided.

INTRODUCTION

A cluster analysis is a multivariate data exploration method gaining popularity in the industry. Understanding the general purpose of a cluster analysis is an important starting point to understanding the process of how to construct a cluster analysis. In a cluster analysis, groups, or “clusters”, are identified based on the similarities between the data points, or a “natural” grouping. This can be done with a single data point or a combination of data points of interest, such a series of questionnaires. There are several ways to perform a cluster analysis, but the two primary methods will be explained in depth.

APPROACHES TO CLUSTERING

The two primary ways to determine clusters are K-means and Hierarchical. These approaches use opposite philosophies to determine clusters in data. There are pros and cons to each of these, one or the other might lend itself better to certain situations.

K-MEANS CLUSTERING

The K-Means approach is derived based on dividing data points into *k* number of groups. Figure 1 provides a basic illustration of this process. Step 1 shows a selection of random data points, in step 2 they are divided into arbitrary groups, or clusters. For this example, *k*=3 groups have been used. A mean for each of the *k* groups is determined, the red points in Figure 1, and literally or virtually plotted in the data field, as shown in step 3. Then, in step 4, the groups are reassigned determined by which of the group means each point lies closest to. From these new *k* groups, the group mean is re-calculated, shown in step 5 by the blue points, and the same process repeats for steps 4 and 5. Final clusters are determined when the process is repeated twice without any data points being assigned to a different group.

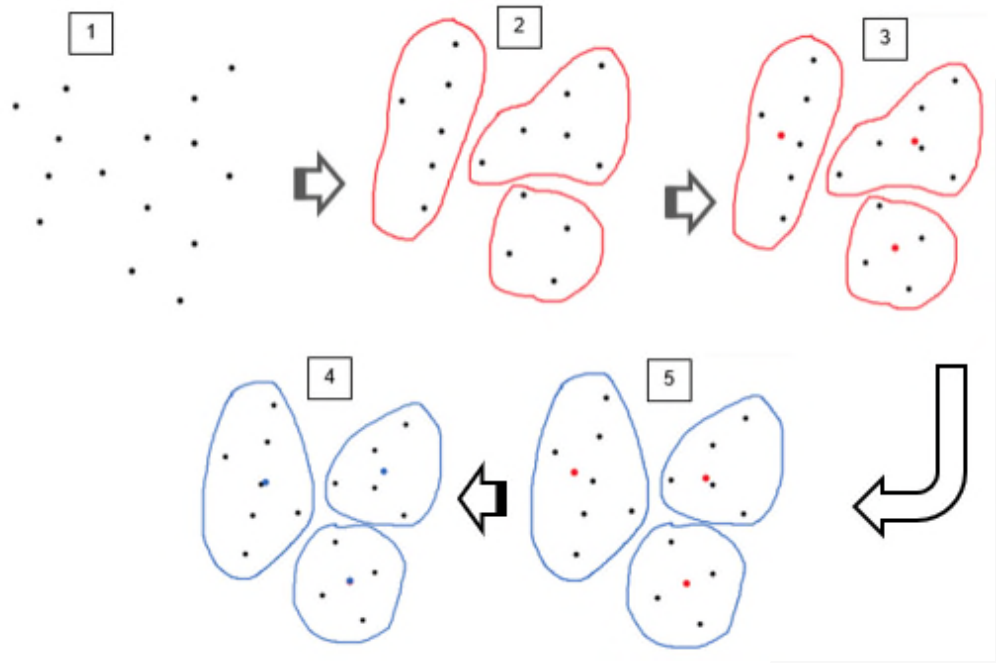


Figure 1. K-Means Clustering Process

In a K-Means cluster analysis, picking the right number of clusters is particularly important. Too few clusters may not effectively describe the data, however the other extreme of too many clusters may divide the data up too much beyond its natural groups. For this reason, with K-Means it is important to understand your data and have an expectation of the number of groups you are looking for. A straight-forward way to determine the number of clusters would be to view the data in a graphical form to look for natural divisions. When in doubt between several different *k* values, then it is always an option to try multiple different k-means analyses and determine later which number of clusters makes the most sense for your analysis.

HIERARCHICAL CLUSTERING

The second approach to a cluster analysis is the Hierarchical method. In contrast to K-means, in the hierarchical method clusters are merged based on distance from each other. It can be done in one of two ways: agglomerative or divisive. Agglomerative considers each data point as its own individual data point and then clusters data points based on the distance between each data point. Divisive considers all points as a single cluster initially and then divides the clusters based again on the distance of the data points. In this paper for simplicity we will focus on the agglomerative approach. This process is illustrated by Figure 2. In the first stage of merging, each data point is grouped with the data point closest to it as defined by one of the linkage methods for hierarchical clustering defined below:

- Single Linkage: the distance between the closest data points of the two clusters.
- Complete Linkage: the distance between the data points of the two clusters which are the farthest apart from each other.
- Average Linkage: comparing between all pairs and averages of all distances. Also called UPGMA – Unweighted Pair Group Mean Averaging.
- Centroid Method: finding the mean vector location for each of the clusters and taking the distance between the two centroids.
- Ward's Method: Uses statistical analysis methods such as error sum of squares and R-squared to determine groupings of data points.

Then, each of these groups is merged with the groups closest to its group mean, and so on. This continues until all groups have been merged.

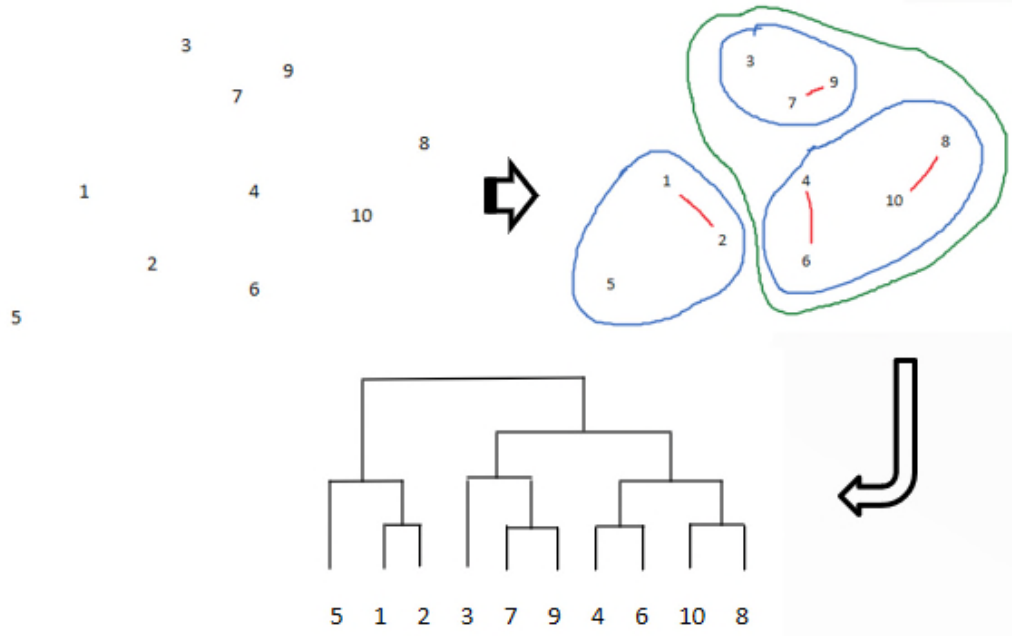


Figure 2. Hierarchical Clustering Process

The optimal number of clusters with the Hierarchical method is determined by the minimum number of groups with the maximum amount of distance between group means. Frequently, this is illustrated with a dendrogram of the merging clusters. Using a dendrogram, the ideal number of clusters is determined by the number of clusters intersected when drawing a horizontal line through the largest vertical distance between merging clusters. Similar to K-Means, the optimal *K* value must be chosen – but this method gives some perspective as to what the ideal *K* value may be.

METHOD PROS AND CONS

While both K-Means and Hierarchical clustering methods will result in the same general goal, there are important differences in usage to note between the two approaches. The first difference is processing power and time. Hierarchical clustering is a more complex way to determine clusters, working in a quadratic approach, while K-Means clustering is linear. Due to this, hierarchical clustering cannot handle big data efficiently.

Another difference is that with K-Means there is a factor of randomness. Since the clustering begins with a random choice of clusters and the results are made by determining the cluster definitions based on the existing group means, the result may vary based on where centroids are initially placed. Because hierarchical clustering is a fixed process, the results will always be identical.

Due to the use of group means in determining clusters, K-Means tends to the best approach when the clusters are circular or spherical.

Finally, the number of clusters requires different approaches depending on the method. To use K-Means clustering, the number of clusters is arbitrarily determined, either from existing knowledge of the data and the approximate number of groups you want to divide the data into. Of course, a different approach to K-Means would be to try several numbers of clusters and see which number best represents the data or produces any significant differences in analysis. Hierarchical clustering allows the user to determine the number of clusters which is most appropriate for their study by use of the dendrogram.

GENERAL PROCESS

As with any analysis, an important first step is to define the question of interest. This gives a good starting point for determining which variables are the most useful to use in order to answer your question. Once a list of variables has been formed, any outliers for each variable could be removed if desired. If these variables are scales of different directions and divisions (e.g. 0 to 10 with 0 best vs 10 being best in another scale, or scales of 0 to 10 and -2 to 2 being used), then a transformation to standardize should be considered in order to better facilitate logical and natural clusters. This could be done with something as simple as assigning Z-scores across all data values.

If the K-means approach is being used, then at this point the data should be considered and a desired number of clusters should be determined. The analysis can be done using multiple k-cluster values and analyzed to see which best reflects the data. After the clusters have been determined, a cluster is assigned to each subject of a study. This cluster assignment should be assigned as a constant categorical variable and will be used as a covariate in the formal analysis of the clusters using your modeling method of choice.

PROGRAMMING AND VISUALIZING CLUSTERS

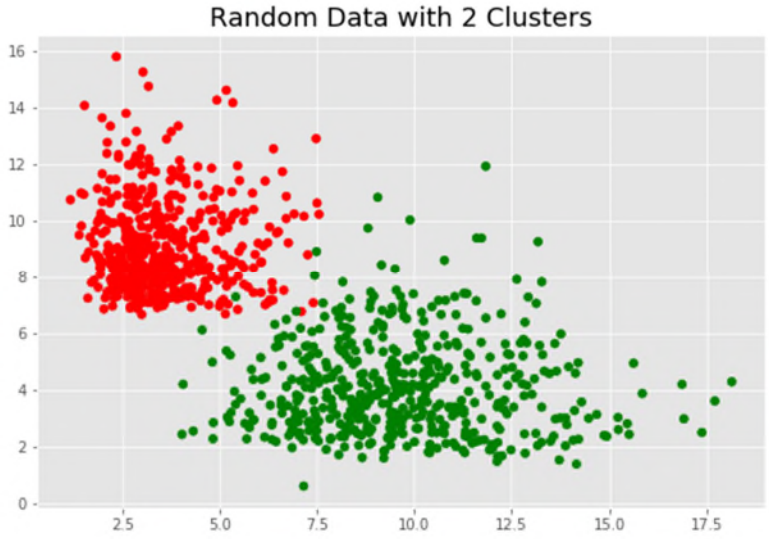
Clustering can be done programmatically in the popular programming formats, as well as most interfaces. We will take a closer look specifically at SAS, Python and R. We will use two datasets (generated for the purpose of this paper) for these examples: DATA_2 and DATA_6. Each contains 3 variables: X, Y and Z. X and Y are continuous variables that will be used for clustering, Z contains a randomly created starting group. DATA_2 will be used for all 2-cluster examples and DATA_6 will be used for all 6-cluster examples.

Full code examples are available on our Github link here: https://github.com/mstackhouse/pharmasug-st183. Some parts of the code have been removed for simplicity of presentation.

Let's start simple. To understand this concept, let's make a dataset with two distinct samples. We're going to choose two centers:

- x = 2, y =7
- x =7, y = 2

```
# Set a random seed for reproducibility
np.random.seed(0)

# Set the centers - These can be
centers = [(2,7), (7,2)]

# Generate the data
data_2 = gen_data(centers)

# Assign the z values to a color
c = [color_map[int(i)] for i in data_2.z.tolist()]

# Plot it
plt.figure(figsize=(9,6))
plt.scatter(data_2.x, data_2.y, color=c)
plt.title('Random Data with 2 Clusters', size=18)
plt.show()
```



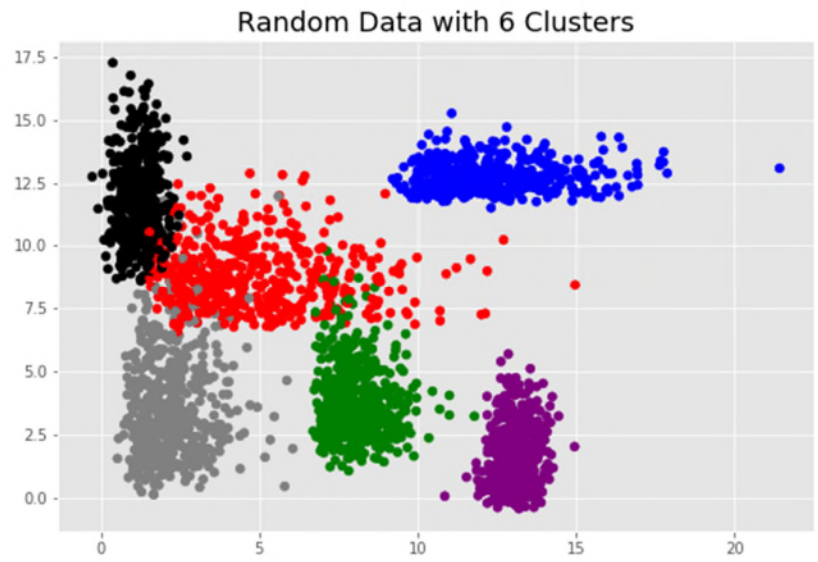
Output 1. Random Data with 2 Clusters from DATA_2

Great! Look at that. We have two clear groups of data. Now we let's make things a little more complicated and spice up our dataset. Let's make some more groups of data.

- x = 2, y = 7
- x = 7, y = 2
- x = 10, y = 12
- x = 13, y = 0
- x = 1, y = 10
- x = 1, y = 1

```
# Set a random seed for reproducibility
np.random.seed(6)

# Set the centers
centers = [(2,7), (7,2), (10,12), (13,0), (1,10), (1,1)]

# Generate the data
data_6 = gen_data(centers)

# Write out the data to a csv
data_6.to_csv('data_6.csv', index=False)

# Assign the z values to a color
c = [color_map[int(i)] for i in data_6.z.tolist()]
```

```
# Plot it
plt.figure(figsize=(9,6))
plt.scatter(data_6.x, data_6.y, color=c)
plt.title("Random Data with 6 Clusters", size=18)
plt.show()
```



Output 2. Random Data with 6 Clusters from DATA_6

Now let's go ahead and predict the clusters!

CLUSTER EXAMPLE - PYTHON

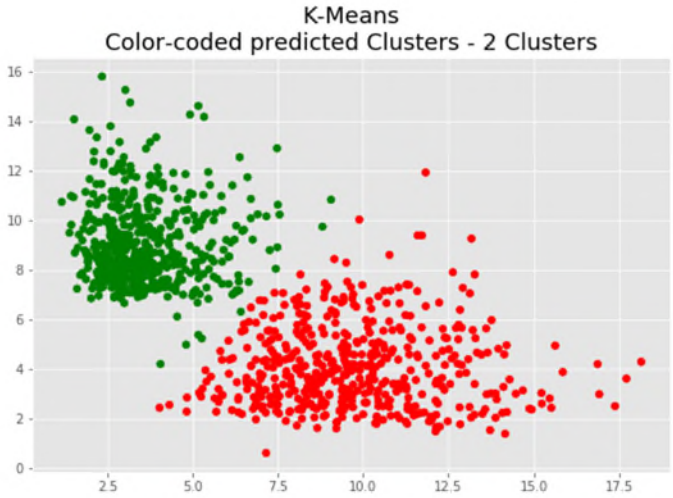
Let's look at using Python to do clustering. Below is a summary of the versions used in this demo:

```
Pandas: 0.20.3
NumPy: 1.14.2
Matplotlib: 2.1.0
SK-Learn: 0.19.1
SciPy: 1.0.0
```

K-Means Approach – 2 Cluster Example

```
# Separate the labels and the data
train = data_2[['x', 'y']]
labels = data_2['z']

# Create the K-means cluster object
km = KMeans(n_clusters=2)
km.fit(train)

# Get the centroids
```

```
centers = km.cluster_centers_

print("Centers:\n", centers)


# Get the points for each cluster
predicted = km.predict(train)

# Check how we did
# Assign the predicted values to a color
c = [color_map[int(i)] for i in predicted]
# Note the change here of using predicted instead of the Z variable

# Plot it
plt.figure(figsize=(9,6))
plt.scatter(data_2.x, data_2.y, color=c)
plt.title("K-Means\nColor-coded predicted Clusters - 2 Clusters", size=18)
plt.show()

 Centers:
  [[9.70550897 4.06574646]
  [3.59959631 9.04532532]]
```

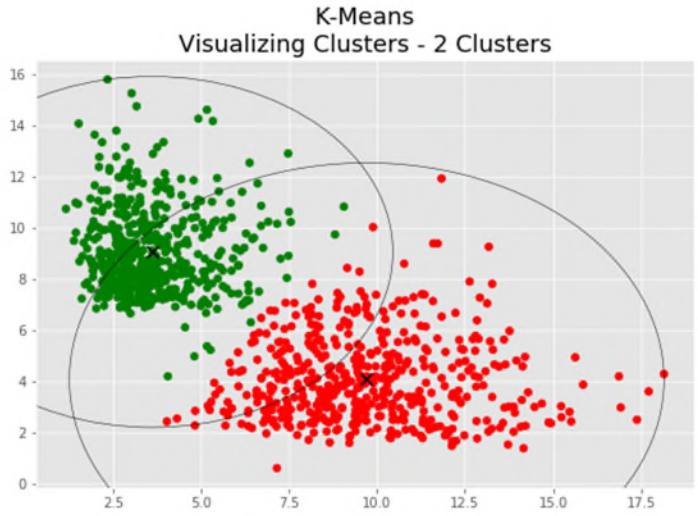


Output 3. K-Means Color-Coded Predicted Clusters from DATA_2

You can see here that for the most part, the predictions were successful. The centroids are slightly different, but they're reasonably close to the distributions we assigned. There is some slight misclassification in the boundaries between the red and green groups, which is understandable. Overall – the predictions were very successful.

Now – let's get a better understanding of how these values are being predicted. K-means is going to assign a data point to the nearest centroid. To visualize this, we can draw a circle around the centroid to look at how close each datapoint is to it is assigned cluster.

```
# Get the distance to the closest centroid
# What's happening here is that km.transform is giving us the distance to
# each of the two centroids - so we're taking the smallest of those two
# distances, as that's the centroid to which a pair of datapoints is
# assigned
clust_dist = [min(x) for x in km.transform(train)]

# Identify the max distance for each cluster
max_dist = np.zeros(2) # Initialize 0 as initial distance to compare

# Loop over each pair of data points, and evaluate
# the distance to the cluster
# Collect the largest as that's going to ultimately be our radius
for i in range(len(train)):
    # Pick out which label was assigned
    label = predicted[i]

    # Check if this distance is larger than the max distance
    if clust_dist[i] > max_dist[label]:
        # If the distance is farther, assign as the max distance
        # for that label
        max_dist[label] = clust_dist[i]

# Plot the scatter plot
plt.figure(figsize=(9,6))
plt.scatter(data_2.x, data_2.y, color=c)
plt.title("K-Means\nVisualizing Clusters - 2 Clusters",size=18)

# Get the figure info because we need to draw the centroid and a circl
fig = plt.gcf()
ax = plt.gca()

# Build the circles
for i in range(len(centers)):
    # Scatterplot the centroid - so just plotting one point
    plt.scatter(centers[i][0], centers[i][1],
                marker='x', s=80, color='black',
                alpha=1)

    ax.add_artist(plt.Circle((centers[i][0], centers[i][1]),
                  max_dist[i], facecolor='None', edgecolor='black'))
```



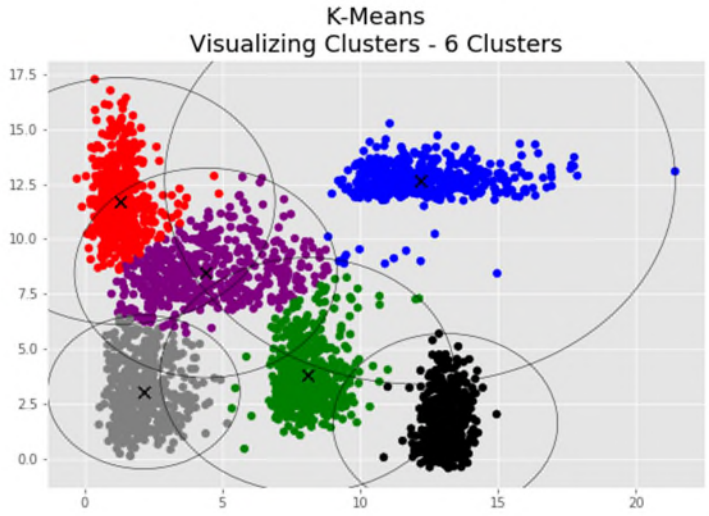
Output 4. K-Means Visualizing Clusters of DATA_2

6-Cluster Example

```
# Separate the labels and the data
train = data_6[['x', 'y']]
labels = data_6['z']

# Create the K-means cluster object
km = KMeans(n_clusters=6)
km.fit(train)

# Get the centroids
centers = km.cluster_centers_

print("Centers:\n", centers)

# Get the points for each cluster
predicted = km.predict(train)

# Check how we did
# Assign the predicted values to a color
c = [color_map[int(i)] for i in predicted]
# Note the change here of using predicted instead of the Z variable

# Get the distance to the closest centroid
# What's happening here is that km.transform is giving us the distance
# to each of the two centroids - so we're taking the smallest
# of those two distances, as that's the centroid to which a pair of
# datapoints is assigned
clust_dist = [min(x) for x in km.transform(train)]

# Identify the max distance for each cluster
max_dist = np.zeros(6) # Initialize 0 as initial distance to compare
```

```
# Loop over each pair of data points, and evaluate the distance
# to the cluster
# Collect the largest as that's going to ultimately be our radius
for i in range(len(train)):
    # Pick out which label was assigned
    label = predicted[i]

    # Check if this distance is larger than the max distance
    if clust_dist[i] > max_dist[label]:
        # If the distance is farther, assign as the max distance
        # for that label
        max_dist[label] = clust_dist[i]

# Plot the scatter plot
plt.figure(figsize=(9,6))
plt.scatter(data_6.x, data_6.y, color=c)
plt.title("K-Means\nVisualizing Clusters - 6 Clusters",size=18)

# Get the figure info because we need to draw the centroid and a circl
fig = plt.gcf()
ax = plt.gca()

# Build the circles
for i in range(len(centers)):
    # Scatterplot the centroid - so just plotting one point
    plt.scatter(centers[i][0], centers[i][1], marker='x', s=80,
                color='black', alpha=1)

    ax.add_artist(plt.Circle((centers[i][0], centers[i][1]), max_dist[i],
                  facecolor='None', edgecolor='black'))
```

```
Centers:
[[ 1.30871154 11.72308601]
[ 8.0907629   3.81038679]
[12.17115681 12.65926635]
[ 4.40554158  8.46152323]
[13.09093221  1.6279105 ]
[ 2.1532958   3.04194748]]
```



Output 5. K-Means Visualizing of DATA_6

Hierarchical Approach

Now let's look at a different method – Hierarchical clustering. Starting with each individual datapoint as a cluster, agglomerative hierarchical clustering will work backwards, taking the "nearest neighbor" to group datapoints (using some distance metric, i.e. Euclidian distance) into larger clusters unit you have one big cluster at the end. At that point, you can use a dendrogram to decide on optimal number of clusters for your problem.

Given that this is agglomerative clustering, start from the bottom of the dendrogram up. Each cluster is indicated by a U-shaped link. The legs of the U represent each of the individual clusters, and as the hierarchical clustering processes, the U represents the two clusters that were merged. The length of the legs in the U-shaped link represent the distance between those clusters. For more information on the dendrogram represented in this example, check this link. Another significant factor of hierarchical clustering is the linkage method. For more information on this, please see the prior sections. In all of the following examples, we will be using Ward's method.

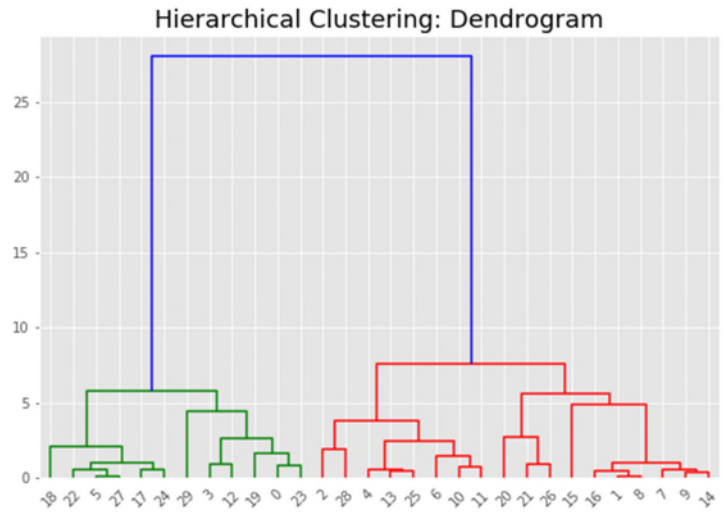
Let's look at an example.

Note: the code used below originated in the article here.

2-Cluster Example

```
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt

# Subset the dataset to a reasonable size for visualization
subset = data_2.iloc[0:30]

# Establish the linking
linked = linkage(subset, 'ward')

# X tick labels
labelList = range(len(subset))
```

```
# Make the figure
plt.figure(figsize=(9, 6))

# Draw the dendogram
dendrogram(linked,
           orientation='top',
           labels=labelList,
           distance_sort='descending',
           show_leaf_counts=True)
plt.title("Hierarchical Clustering: Dendrogram", size=18)
plt.show()
```



Output 6. Dendrogram of DATA_2 in Python

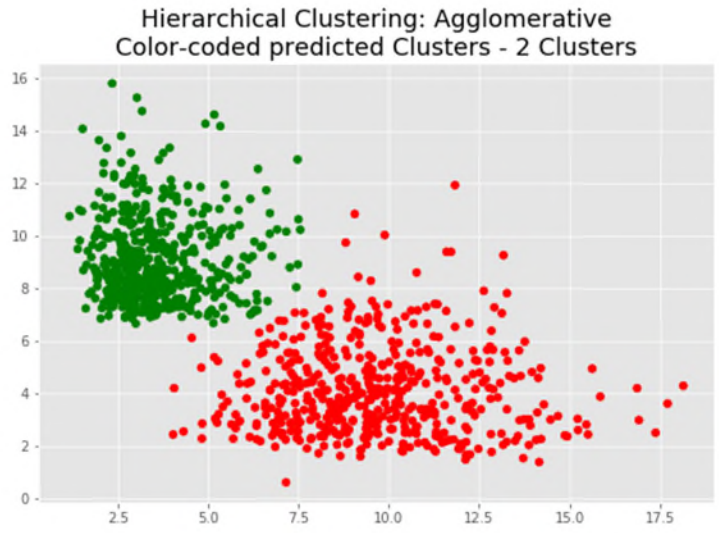
Looking at this example we can clearly see that optimally, we have two clusters in the dataset. Look at the top link – the legs of the U are much longer than the two clusters beneath it, and from there on the legs are quite short.

Let's use this method on the full dataset and see how it does.

Note: Here I'm moving back to using the sklearn module, despite the last example using scipy. Scipy offered the dendrogram visualization, while sklearn offers the extremely portable API that allows me to use the same code from the K-means example with minimal changes.

```
# Separate the labels and the data
train = data_2[['x', 'y']]
labels = data_2[['z']]

hc = AgglomerativeClustering(n_clusters=2)
hc.fit(train)

# Get the points for each cluster
predicted = hc.labels_
```

```
# Check how we did
# Assign the predicted values to a color
c = [color_map[int(i)] for i in predicted]
# Note the change here of using predicted instead of the Z variable


# Plot it
plt.figure(figsize=(9,6))
plt.scatter(data_2.x, data_2.y, color=c)
plt.title("Hierarchical Clustering: Agglomerative\nColor-coded predicted
Clusters - 2 Clusters", size=18)
plt.show()
```



Output 7. Hierarchical Clustering Color-Coded Predicted Clusters of DATA_2

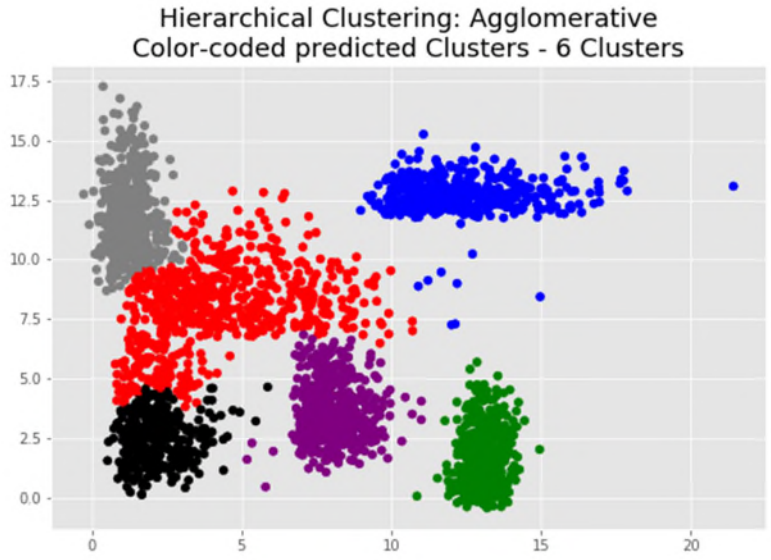
Another point to take notice of in this example is that we're not given centroids to the clusters like with K-Means. This is because the approach is fundamentally different; instead of measuring the distance from the center of the cluster, this method analyzes the distance from individual points – and this method differs based on the linkage method.

6-Cluster Example

Let's look at the same approach using the 6-group dataset:

```
# Separate the labels and the data
train = data_6[['x', 'y']]
labels = data_6[['z']]

hc = AgglomerativeClustering(n_clusters=6)
hc.fit(train)

# Get the points for each cluster
predicted = hc.labels_
```

```
# Check how we did
# Assign the predicted values to a color
c = [color_map[int(i)] for i in predicted] # Note the change here of using
predicted instead of the Z variable


# Plot it
plt.figure(figsize=(9,6))
plt.scatter(data_6.x, data_6.y, color=c)
plt.title("Hierarchical Clustering: Agglomerative\nColor-coded predicted
Clusters - 6 Clusters", size=18)
plt.show()
```



Output 8. Hierarchical Clustering Color-Coded Predicted Clusters of DATA_6

Note that K-Means did a better job at assigning datapoints to the right group in this example. Of course, these are using default settings for the clustering objects, so hyperparameters could be fine tuned to do a better job. Overall, it's up to you to take the time to ensure that you're choosing the better method applicable to your use case.

CLUSTER EXAMPLE – SAS

Now let's move over look at how to program a clustering in SAS:

K-Means Approach

SAS has a straight-forward approach for K-Means clustering, using a single procedure with a list of the variables you want to cluster. Let's look at how these results compare to Python. It is important to keep in mind that K-Means in general tends to vary a bit, by design of the clustering model. However, results in general should be similar, particularly if a predictor variable is included to use for initial clusters.

2-Cluster Example

```
***  2 Cluster Example ***;
***  PROC FASTCLUS is used for K-MEANS clustering ***;
***  The REPLACE= option defines how the pre-defined base
```

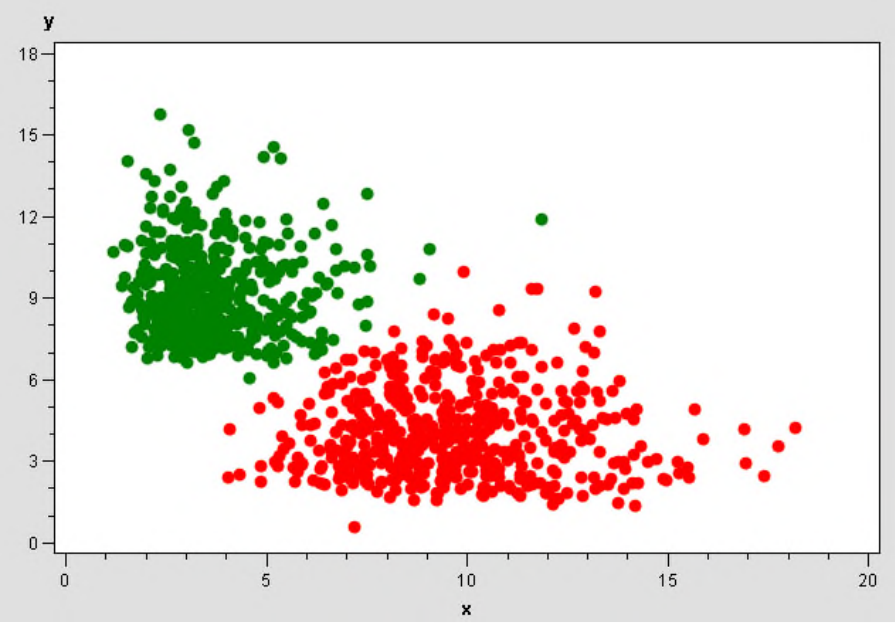
```
     cluster should be replaced, if it exists ***;
***  The variable "CLUSTER" is automatically created in the
     OUT= dataset using PROC FASTCLUS and amended on the input
     dataset.  This variable name can be altered using procedure
     statement options ***;

proc fastclus data=data_2 maxclusters=2 replace=random out=cluster2
   var x y;
run;
```

Cluster Means

Cluster	x	y
1	9.655086608	4.061433726
2	3.600656546	9.088994867

Output 9. PROC FASTCLUS Cluster Means Output in SAS of DATA_2

```
***  Define cluster colors for plot ***;

axis1 order=(0 to 25 by 5) label=("x");
axis2 order=(0 to 18 by 3) label=("y");

***  Plot CLUSTER2 dataset in order to visually review the clusters ***;
proc gplot data=cluster2;
   plot y*x=cluster / haxis=axis1 vaxis=axis2 overlay skipmiss;
run;
```



Output 10. PROC GPLOT for DATA_2 Divided into Two Clusters

The SAS output looks almost identical to the Python approach! Let's see how the 6-Cluster dataset compares.

6-Cluster Example

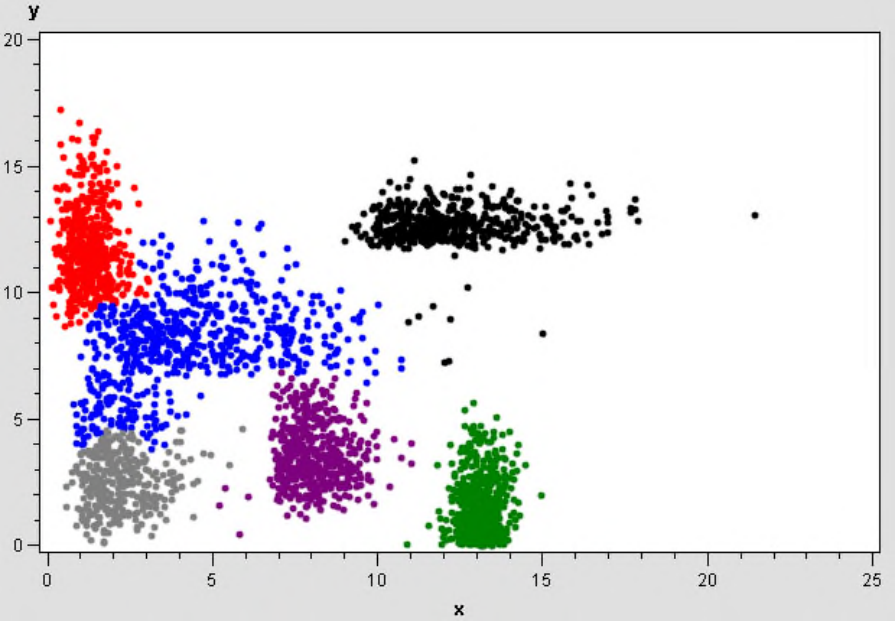
```
***  6 Cluster Example  ***;
***  PROC FASTCLUS is used for K-MEANS clustering ***;
***  The REPLACE= option defines how the pre-defined base
     cluster should be replaced, if it exists ***;
***  The variable "CLUSTER" is automatically created in the
     OUT= dataset using PROC FASTCLUS and amended on the input
     dataset.  This variable name can be altered using procedure
     statement options ***;

proc fastclus data=data_6 maxclusters=6 replace=random out=cluster6
   var x y;
run;
```

Cluster Means

Cluster	x	y
1	4.34837637	8.30404190
2	12.16450703	12.65314064
3	8.09076290	3.81038679
4	13.09093221	1.62791050
5	1.37926957	11.69232989
6	2.14537156	2.92449581

Output 11. PROC CLUSTER Cluster Means Output in SAS of DATA_6

```
***  Define cluster colors for plot ***;

symbol1 color=red value=dot width=.5 height=.5;
symbol2 color=green value=dot width=.5 height=.5;
symbol3 color=purple value=dot width=.5 height=.5;
symbol4 color=blue value=dot width=.5 height=.5;
symbol5 color=gray value=dot width=.5 height=.5;
symbol6 color=black value=dot width=.5 height=.5;
axis1 order=(0 to 25 by 5) label=("x");
axis2 order=(0 to 20 by 5) label=("y");

***  Plot CLUSTER6 dataset in order to visually review the clusters ***;
proc gplot data=cluster6;
   plot y*x=cluster / haxis=axis1 vaxis=axis2 overlay skipmiss;
run;
```



Output 12. PROC GPLOT for DATA_6 divided into two clusters

This clustering also appears to be extremely similar to what Python produced!

Note: even though the colors are different, that doesn't matter. Clustering helps us visualize groups when we're not sure what they are, rather than classify groups that we already know exist.

Hierarchical Approach

SAS uses a different procedure for hierarchical clustering. PROC CLUSTER performs clustering hierarchically, using the method specified in the PROC CLUSTER statement. The dendrogram can be produced using PROC TREE (though we have not included the dendrogram in the displays, but the code is included).

2-Cluster Example

```
***  2 Cluster Example ***;
***  A unique identifier can be assigned to each row
      so that it can be used as an ID variable for further
      investigation of clustering in the output file ***;

data data_2;
   set data_2;

   row = _N_;
run;

***  PROC CLUSTER is used for hierarchical clustering ***;
***  The Method is defined with the option method=.
     outtree= must be used to proceed with the dendrogram clustering ***;

proc cluster data=data_2 method=ward outtree=cluster2a;
   var x y;
   id row;
run;
```

The CLUSTER Procedure
Ward's Minimum Variance Cluster Analysis

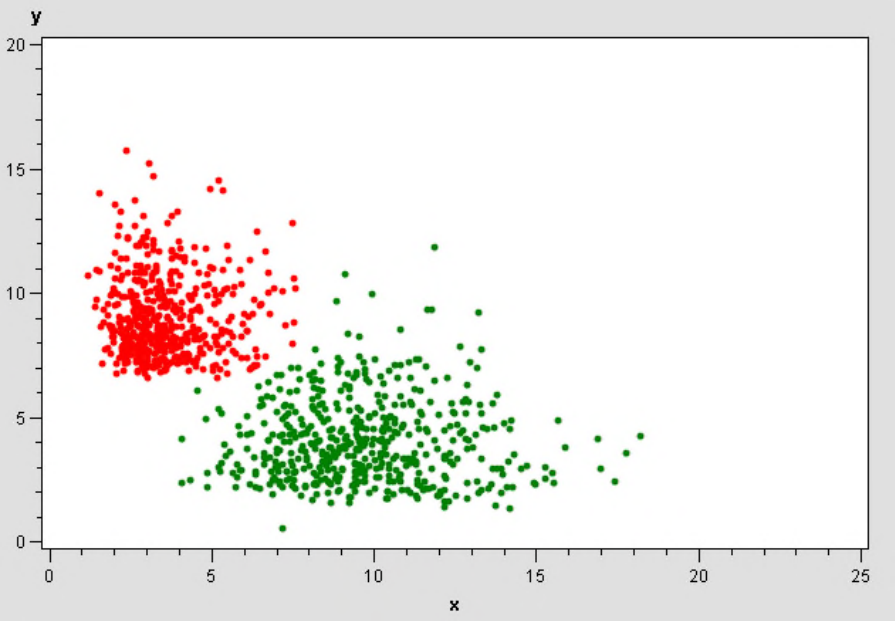
Eigenvalues of the Covariance Matrix				
	Eigenvalue	Difference	Proportion	Cumulative
1	18.8101417	15.7301716	0.8593	0.8593
2	3.0799701		0.1407	1.0000

Root-Mean-Square Total-Sample Standard Deviation	3.308331

Root-Mean-Square Distance Between Observations	6.616663

Output 13. PROC CLUSTER in SAS of DATA_2

```
***  PROC TREE is used to produce a dendrogram analysis for the
     hierarchical clustering.  The output from PROC TREE is a
     list of every record and how each is clustered together. ***;

proc tree horizontal nclusters=2 out=cluster2b noprint;
   id row;
run;
proc sort data=cluster 2b;
   by row;
run;

***  Merge the output dataset from PROC TREE onto the original dataset to
    assign the clusters on each row ***;

data cluster;
   merge data_2 cluster2b;
   by row;
run;

***  Define cluster colors for plot ***;

symbol1 color=red value=dot width=.5 height=.5;
symbol2 color=green value=dot width=.5 height=.5;
axis1 order=(0 to 25 by 5) label=(“x”);
axis2 order=(0 to 20 by 5) label=(”y”);

***  Plot CLUSTER dataset in order to visually review the clusters ***;
proc gplot data=cluster;
   plot y*x=cluster / haxis=axis1 vaxis=axis2 overlay skipmiss;
run;
```



Output 14. PROC GPLOT for DATA_2 divided into two clusters

This clustering appears to be identical to the Python output for the 2-cluster dataset! Let's examine the 6-cluster dataset and see if that has the same results as well.

6-Cluster Example

```
***  6 Cluster Example ***;
***  A unique identifier can be assigned to each row
     so that it can be used as an ID variable for further
     investigation of clustering in the output file ***;

data data_6;
   set data_6;

   row = _N_;
run;

***  PROC CLUSTER is used for hierarchical clustering ***;
***  The Method is defined with the option method=.
     outtree= must be used to proceed with the dendrogram clustering ***;

proc cluster data=data_6 method=ward outtree=cluster6a;
   var x y;
   id row;
run;
```

The CLUSTER Procedure
Ward's Minimum Variance Cluster Analysis

Eigenvalues of the Covariance Matrix

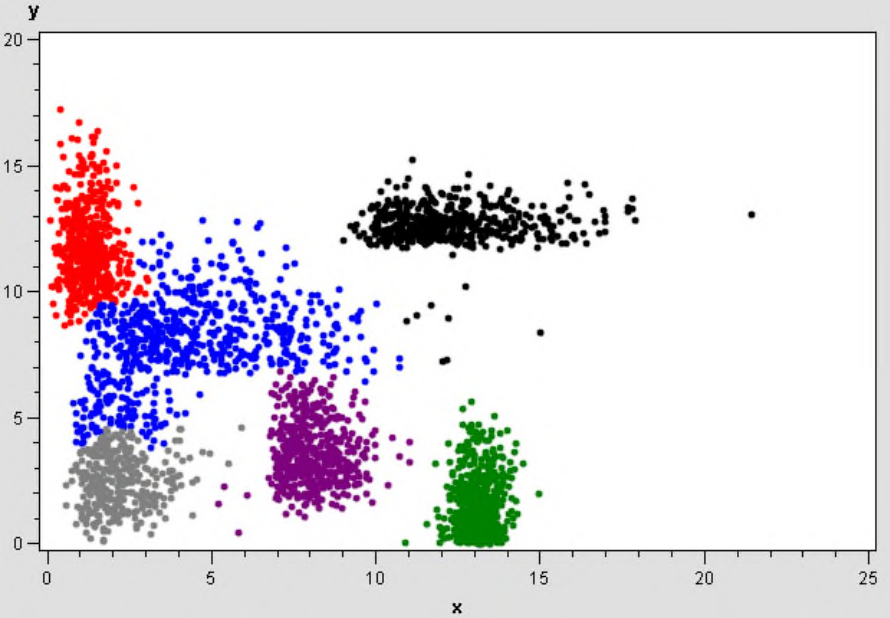
	Eigenvalue	Difference	Proportion	Cumulative
1	25.4115291	7.7480492	0.5899	0.5899
2	17.6634800		0.4101	1.0000

Root-Mean-Square Total-Sample Standard Deviation	4.640852

Root-Mean-Square Distance Between Observations	9.281703

Output 15. PROC CLUSTER in SAS of DATA_6

```
***  PROC TREE is used to produce a dendrogram analysis for the
     hierarchical clustering.  The output from PROC TREE is a
     list of every record and how each is clustered together. ***;

proc tree horizontal nclusters=6 out=cluster6b noprint;
   id row;
run;
proc sort data=cluster 6b;
   by row;
run;

***  Merge the output dataset from PROC TREE onto the original dataset to
     assign the clusters on each row ***;

data cluster;
   merge data_6 cluster6b;
   by row;
run;

***  Define cluster colors for plot ***;

symbol1 color=red value=dot width=.5 height=.5;
symbol2 color=green value=dot width=.5 height=.5;
symbol3 color=purple value=dot width=.5 height=.5;
symbol4 color=blue value=dot width=.5 height=.5;
symbol5 color=gray value=dot width=.5 height=.5;
symbol6 color=black value=dot width=.5 height=.5;
axis1 order=(0 to 25 by 5) label=(“x”);
axis2 order=(0 to 20 by 5) label=(”y”);

***  Plot CLUSTER dataset in order to visually review the clusters ***;
proc gplot data=cluster;
   plot y*x=cluster / haxis=axis1 vaxis=axis2 overlay skipmiss;
run;
```



Output 16. PROC GPLOT for DATA_6 divided into two clusters

Again, our results are almost identical to the Python approach!

CLUSTER EXAMPLE - R

Now let's look at how to cluster in R.

K-Means Approach

2-Cluster Example

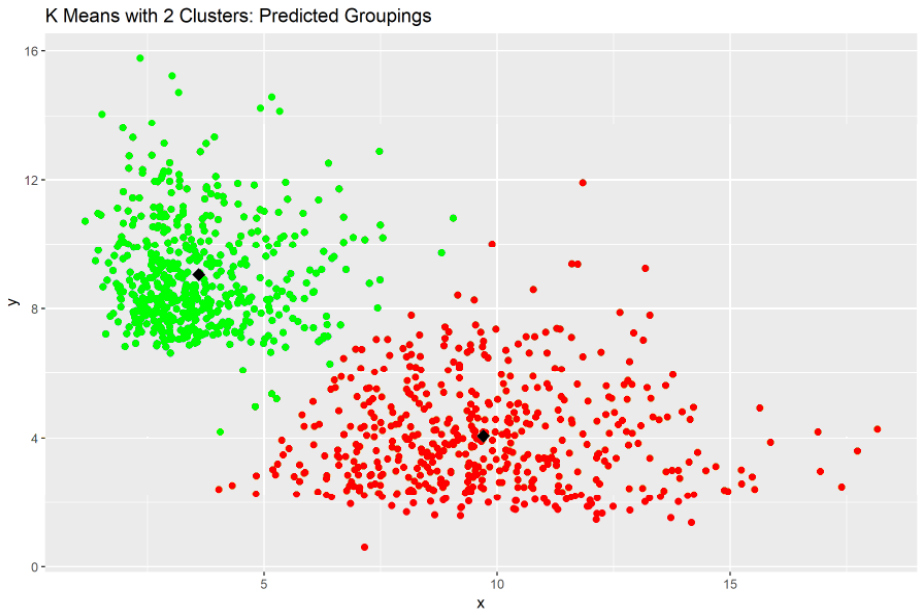
Let's test out K means for 2 groups:

```
# Fit and predict the clusters
km_2 <- kmeans(data_2[c('x', 'y')], 2)

# Display the centers
Print('Cluster centers: ')

## [1] 'Cluster centers: '

print(km_2$centers)

##
##          x        y
## 1 9.705509 4.065746
## 2 3.599596 9.045325

# Turn the centers into a dataset and assign a shape variable
km_2_centers <- as_tibble(km_2$centers) %>%
   Mutate(pred = 99, shape = 2, size = 2)
```

```
# Make a new tibble for the predicted values
data_2p <- data_2 %>%
   mutate(pred = km_2$cluster) %>%
   bind_rows(km_2_centers)
```

Let's plot the predicted values.

```
data_2p %>%
   ggplot(aes(x, y)) +
   geom_point(aes(color = factor(pred), shape = factor(shape), size =
                  factor(size))) +
   scale_color_manual(values = c('red', 'green', 'black')) +
   scale_shape_manual(values = c(16, 18)) +
   scale_size_manual(values = c(2, 4)) +
   theme(legend.position = 'none') +
   ggtitle('K Means with 2 Clusters: Predicted Groupings')
```



Output 17. K Means with 2 Clusters: Predicted Groupings

Great! For the most part, they look the same – and we can see the center of the clusters.

6-Cluster Example

Let's move on to look at the 6-cluster data now.

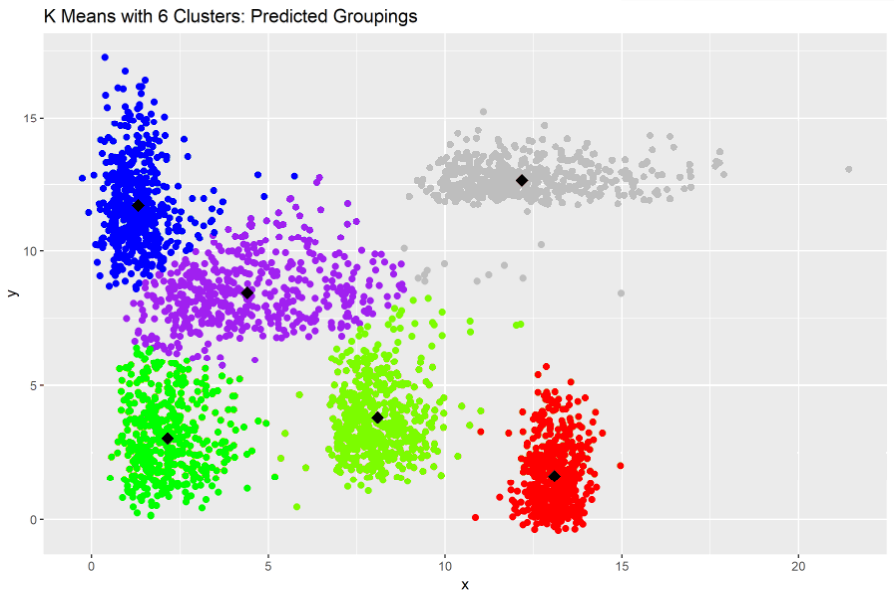
```
## K-Means: 6 Clusters ##
km_6 <- kmeans(data_6[c('x', 'y')], 6)

# Display the centers
Print('Cluster centers: ')

## [1] 'Cluster centers: '

print(km_6$centers)
```

```
##           x         y
## 1 13.090932  1.627911
## 2  2.153296  3.041947
## 3  1.316876 11.725101
## 4  4.402932  8.452970
## 5 12.171157 12.659266
## 6  8.090763  3.810387
```

```
# Add the predicted values back to data_6
# Turn the centers into a dataset and assign a shape variable
km_6_centers <- as_tibble(km_6$centers) %>%
   mutate(pred = 99, shape = 2, size = 2)

# Make a new tibble for the predicted values
data_6p <- data_6 %>%
   mutate(pred = km_6$cluster) %>%
   bind_rows(km_6_centers)
```

Same idea – Let's plot the predicted values:

```
# Plot the prdicted
data_6 %>%
   ggplot(aes(x, y)) +
   geom_point(aes(color = factor(pred), shape = factor(shape), size =
                  factor(size))) +
   scale_color_manual(values = c('red', 'green', 'blue', 'purple', 'grey',
                                 'lawngreen', 'black')) +
   scale_shape_manual(values = c(16, 18)) +
   scale_size_manual(values = c(2, 4)) +
   theme(legend.position = 'none') +
   ggtitle('K Means with 6 Clusters: Predicted Groupings')
```



Output 18. K Means with 6 Clusters: Predicted Groupings

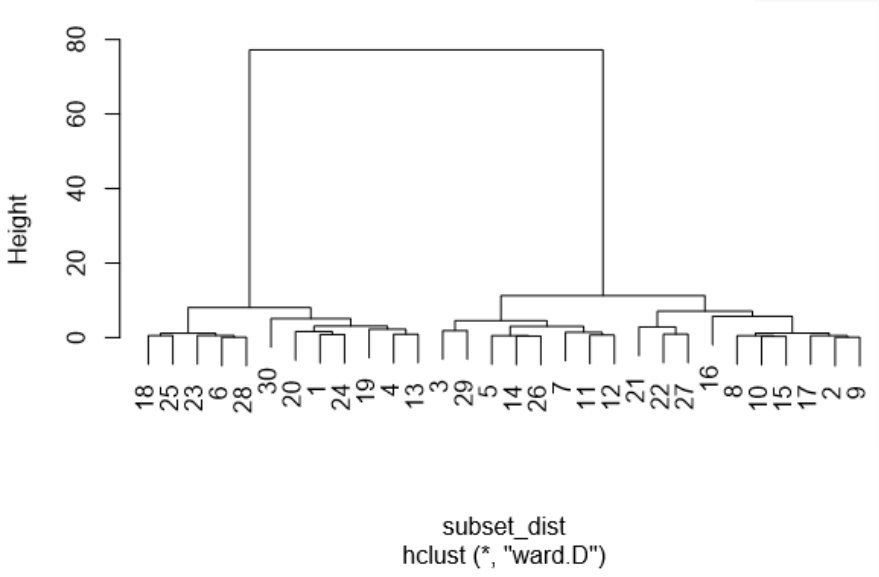
Hierarchical Approach

So we can see that K-means works the same as it does in SAS and in Python. So, let's now take a look at hierarchical clustering. There are some small differences here in that we need to take a few more steps: - Measure the distances of our independent variables (in this case, X and Y) – Create the cluster – Cut the tree.

Remember the dendrogram? That's what R forms first before we get the cluster assignments that we had in python and in SAS.

2-Cluster Example

```
# Subset the data so the dendrogram is a reasonable size
subset <- data_2[1:30, 1:2]
subset_dist <- dist(subset, method = 'euclidean')

# Create the cluster object
hc <- hclust(subset_dist, method = 'ward.D')

# Plot the dendrogram
plot(hc)
```



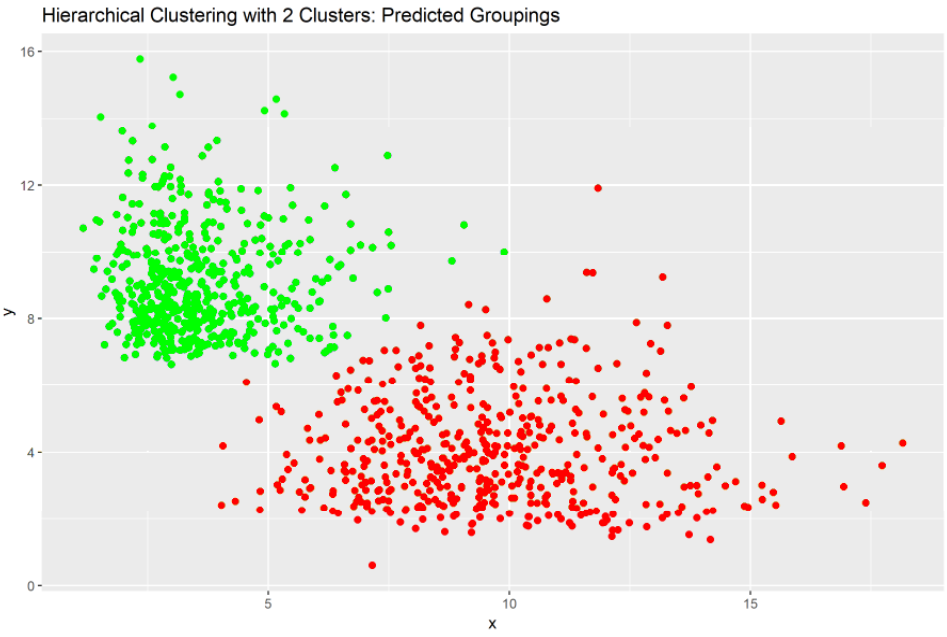
Output 19. Cluster Dendrogram for DATA_2

A nice benefit here is how easy it is to make the dendrogram. Build the cluster, and right away we can plot it with the simple command `plot()`.

Ok – so we know we want two clusters here. Let's do it.

```
# Measure the distances on the 2_group dataset
data_2_dist <- dist(data_2[, 1:2], method = 'euclidean')

# Cluster
```

```
hc_2 <- hclust(data_2_dist, method = 'ward.D')

# Now we cut the tree
data_2_cut <- cutree(hc_2, k = 2)

# Add in the predicted groupings
data_2_hc <- data_2 %>%
   mutate(pred = data_2_cut)

# Plot
data_2_hc %>%
   ggplot(aes(x, y)) +
   geom_point(aes(color = factor(pred), shape = factor(shape), size =
                  factor(size))) +
   scale_color_manual(values = c('red', 'green')) +
   scale_shape_manual(values = c(16)) +
   scale_size_manual(values = c(2)) +
   theme(legend.position = 'none') +
   ggtile('Hierarchical Clustering with 2 Clusters: Predicted Groupings')
```



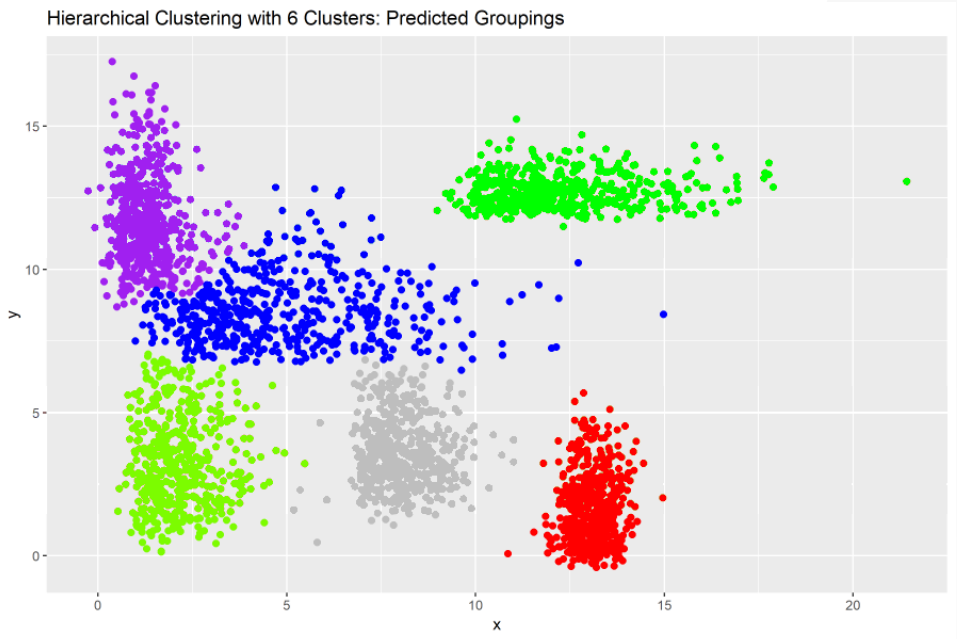
Output 20. Hierarchical Clustering with 2 Clusters: Predicted Groupings of DATA_2

Look at that! Overall it looks like we pretty much have the groups we wanted. Now let's check out 6 groups.

6-Cluster Example

```
# Measure the distance on the 6_group dataset
data_6_dist <- dist(data_6[, 1:2], method = 'euclidean')

# Cluster
hc_6 <- hclust(data_6_dist, method = 'ward.D')

# Now we cut the tree
```

```
data_6_cut <- cutree(hc_6, k = 6)

# Add in the predicted groupings
data_6_hc <- data_6 %>%
   mutate(pred = data_6_cut)

# Plot
data_6_hc %>%
   ggplot(aes(x, y)) +
   geom_point(aes(color = factor(pred), shape = factor(shape), size =
                  factor(size))) +
   scale_color_manual(values = c('red', 'green', 'blue', 'purple', 'grey',
                                 'lawngreen')) +
   scale_shape_manual(values = c(16)) +
   scale_size_manual(values = c(2)) +
   theme(legend.position = "none") +
   ggtitle('Hierarchical Clustering with 6 Clusters: Predicted Groupings')
```



Output 21. Hierarchical Clustering with 6 Clusters: Predicted Groupings of DATA_6

And there you have it!

APPLICATION OF CLUSTERS IN PHARMACEUTICAL INDUSTRY

Now that the clusters have been defined, what are they used for? What purpose is there in determining clusters, what good does it do for clinical trials?

Once clusters are defined, they can be used to better understand your data. A cluster definition is applied to each individual subject in a study selection. This new variable can then be used as a categorical grouping variable in any analysis model in order to see if there is a statistical response to any study variable. Analyzing your data in this way allows you to see if there is any significance gained from "natural" groupings of subjects (i.e. clusters) as opposed to pre-defined groups. It is particularly useful in exploratory analyses in preparation for a new trial, or as a subgroup to investigate during an integration.

There is a plethora of examples of questions which can be answered using cluster analysis, but for the sake of conciseness we will provide only a few:

- Do the clusters have a statistically significant impact on any baseline characteristics?
- Do the clusters have a statistically significant impact on any test results?
- Do the clusters have a statistically significant impact on the primary or secondary response variables?
- Do the groups being used for subgroup analysis best match how the clusters naturally group the data?

DEFINING CLUSTERS

If clusters do have a statistically significant impact on a variable when used as a covariate, the next thing to do is to determine how that cluster is defined. You can do this by examining the cluster means (which is provided in the output of procedure performing the cluster analysis). For instance, if we wanted to define the clusters of our analysis from our SAS examples above, we could do so by forming a table such as Table 1. This contains the cluster means from the K-Means approach cluster means for the SAS K=6 example above. The highest values of each row are in boldface in the table, indicating which clusters have the highest grouping of each variable. By looking down the columns, each cluster can be characterized by their general makeup of the combination of X and Y.

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
X	4.34	**12.16**	8.09	**13.09**	1.38	2.15
Y	8.30	**12.65**	3.81	1.63	**11.69**	2.92

Table 1. Cluster Means

By looking through these clusters we can see which variable makes up the majority of each cluster. For example, Cluster 5 is made up almost 90% by variable Y and Cluster 4 is made up almost 90% by variable X. Cluster 2 has almost equal values of X and Y which indicates this cluster contains many similarities between the two groups. With real data, it is helpful to have subject matter knowledge of the data in order to easily see the similarities between the variables in each cluster with high cluster means.

CONCLUSION

There are times in our work where we are required to look outside of the box for a solution. We need to examine data intently, to look for unexpected trends which may drive our research. Cluster analysis is a great option to achieve this. By looking across variables and subjects to show natural groupings of data rather than those we have pre-defined, we may notice new variables or subgroups of interest. This can easily lead to opening the door to exciting new research and help many people find answers in the process!

REFERENCES

Malik, Usman. “Hierarchical Clustering with Python and Scikit-Learn” Stack Abuse. July 12th, 2018. Available at https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Alyssa Wittle
Covance, Inc.
Alyssa.Wittle@Covance.com

https://www.linkedin.com/in/alyssa-wittle-a922638b

Michael Stackhouse
Covance, Inc.
Michael.Stackhouse@Covance.com
https://github.com/mstackhouse
https://www.linkedin.com/in/michael-s-stackhouse/