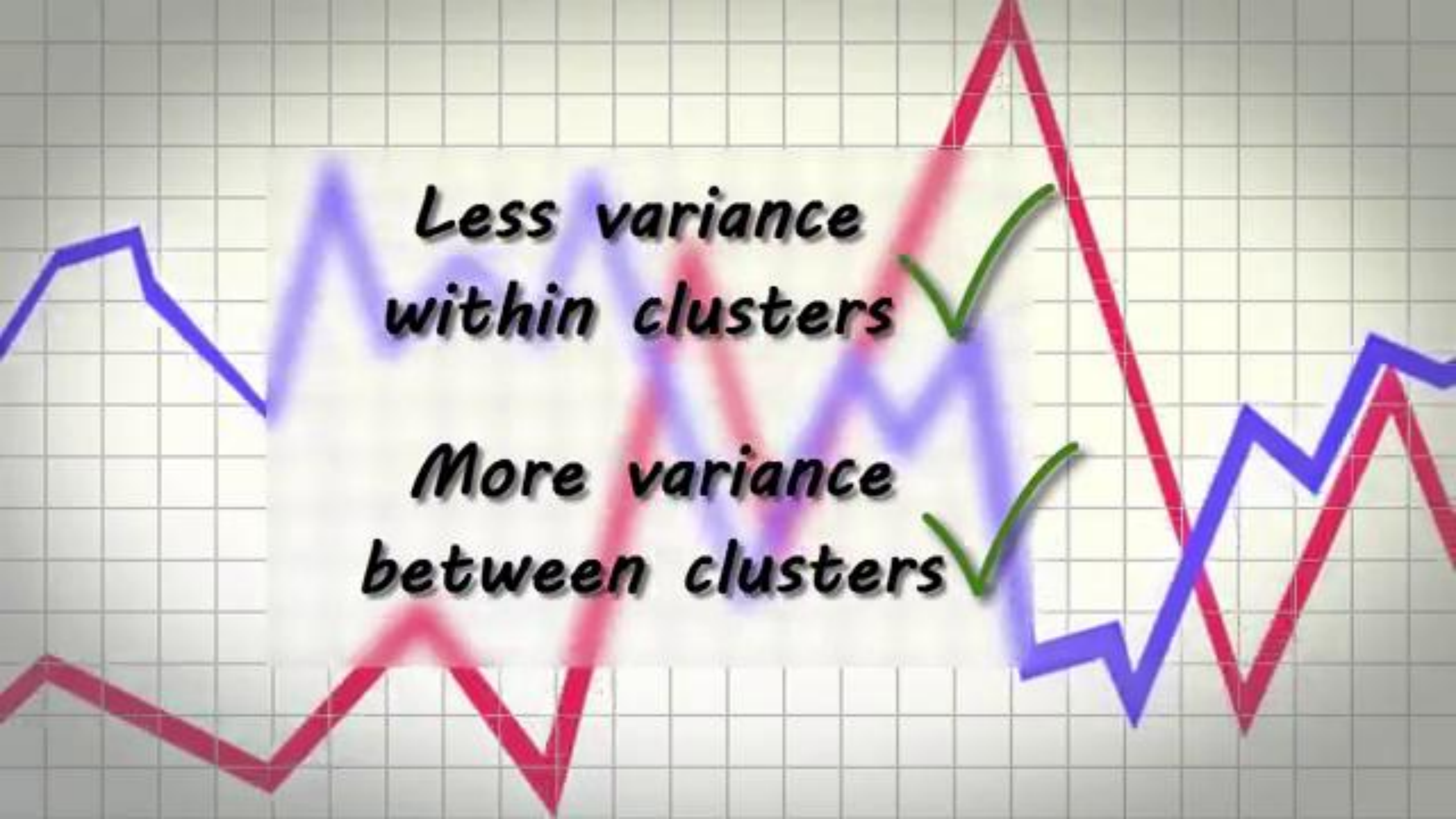
Module 4
Lesson 1 - What Is k-Means Cluster Analysis?

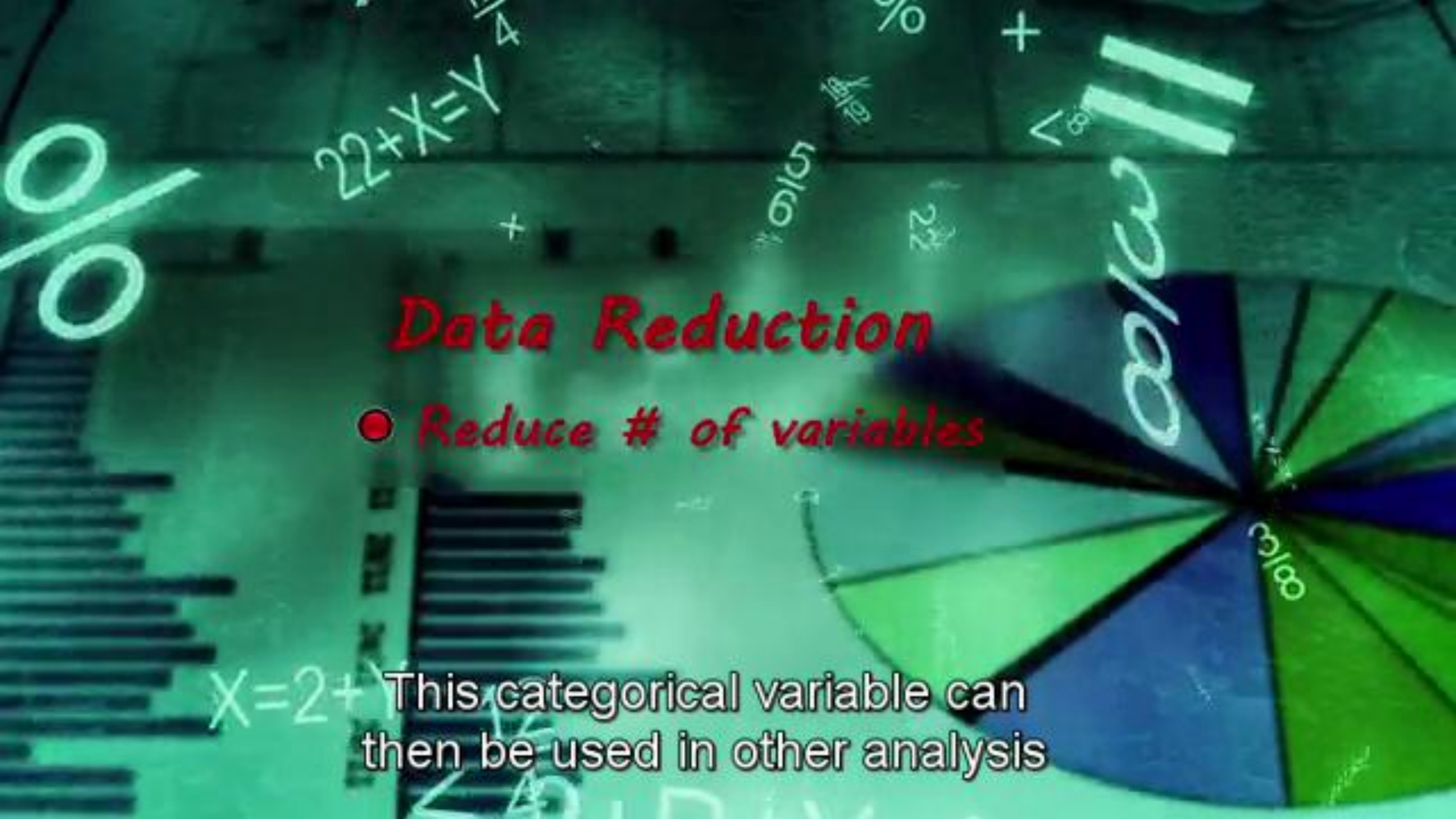
Cluster Analysis

Goal: group similar observations together

The goal of cluster analysis is to group or cluster observations into subsets based

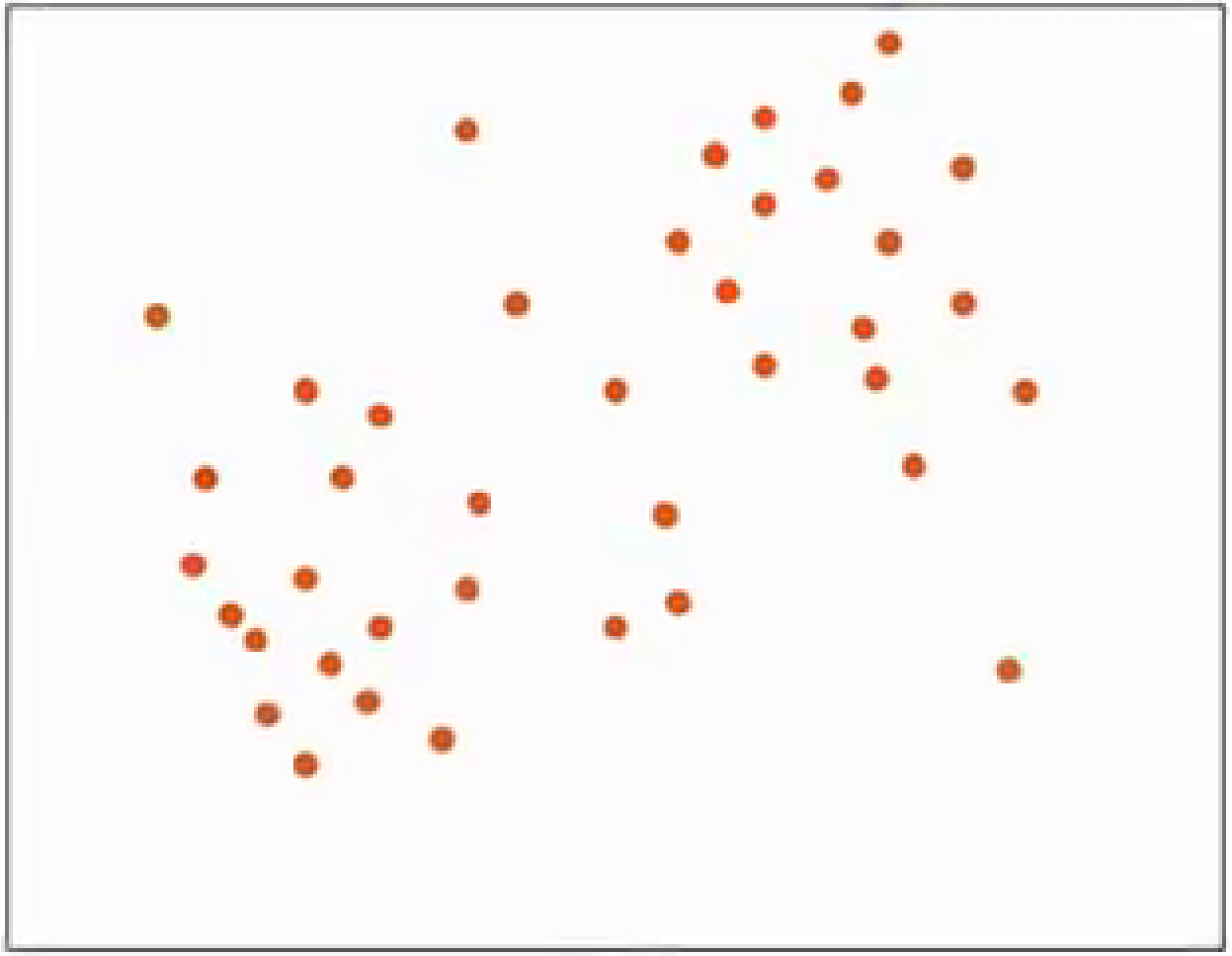

Market
Segmentation


Less variance
within clusters
More variance
between clusters


Data Reduction
Reduce # of variables
This categorical variable can
then be used in other analysis

K-Means Cluster Analysis

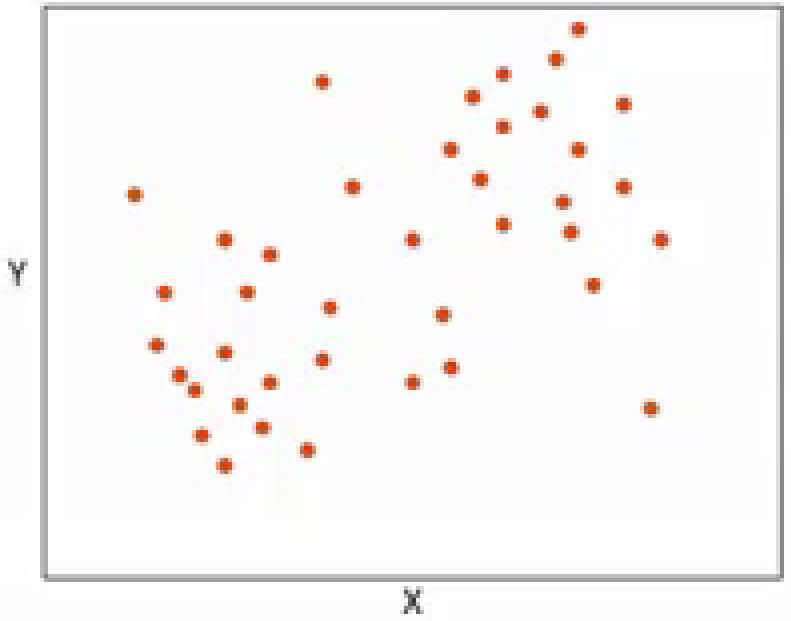
- Creates a p-dimensional space where p is the number of input variables
- Euclidean distance



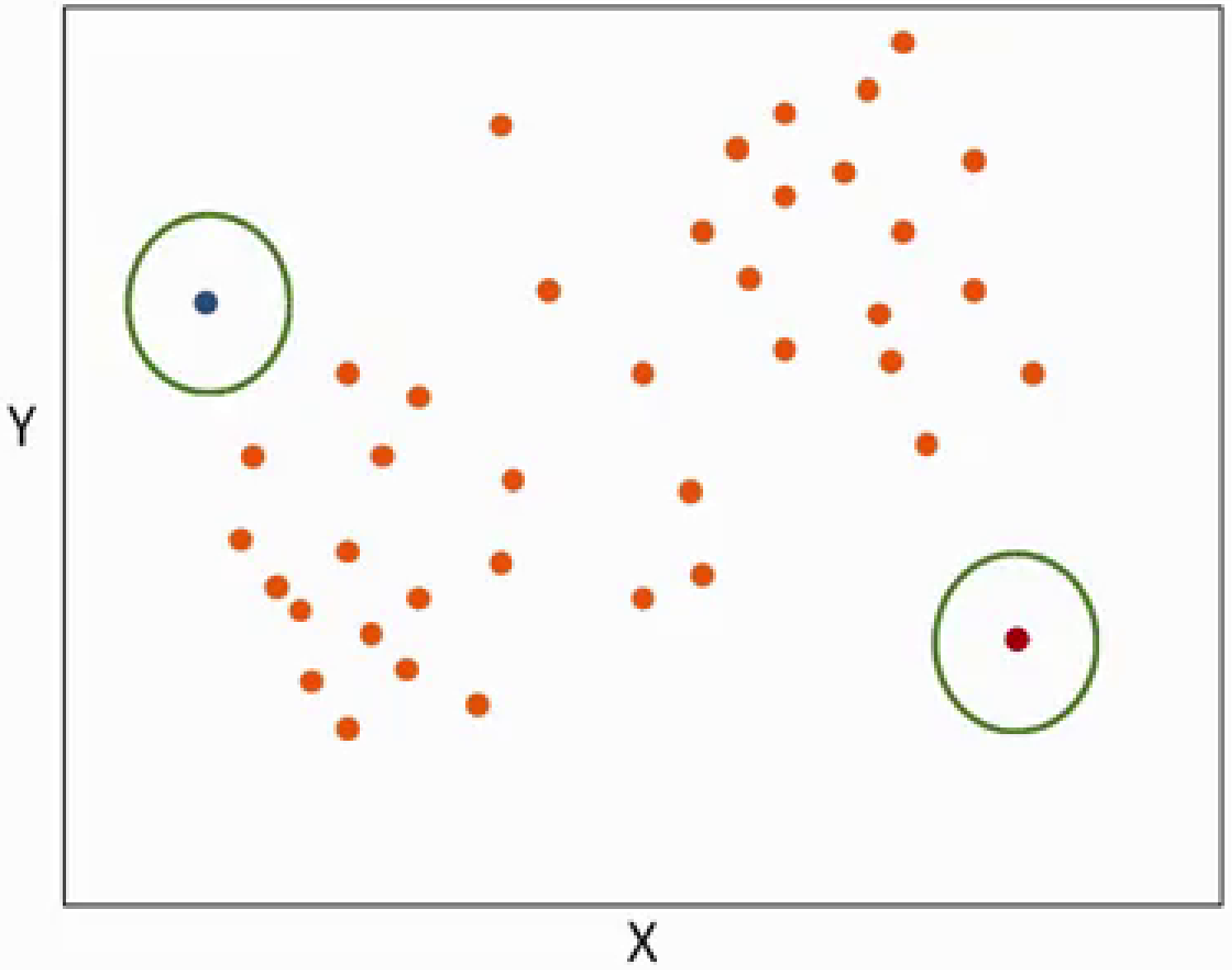
Scatterplot
p=2 variables

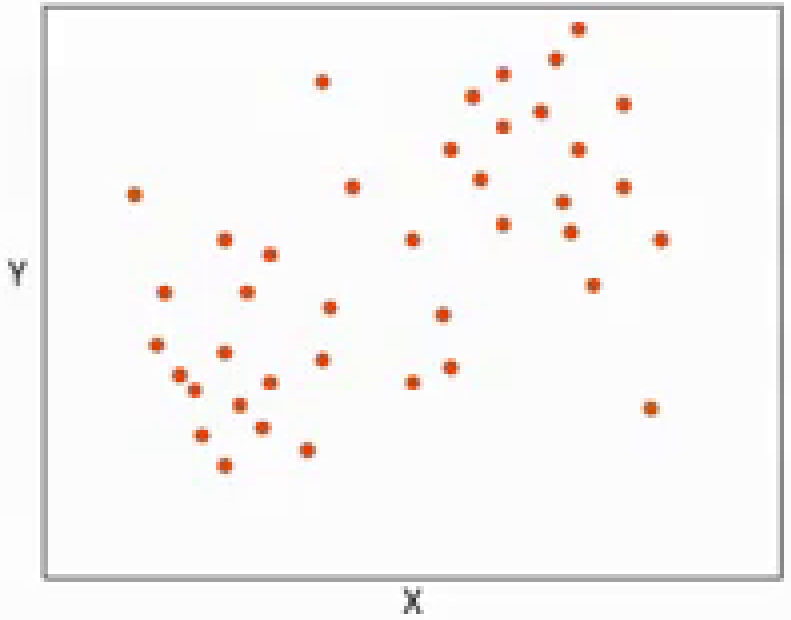
Each point is
an observation

where each of these points represents
an observation in the dataset.

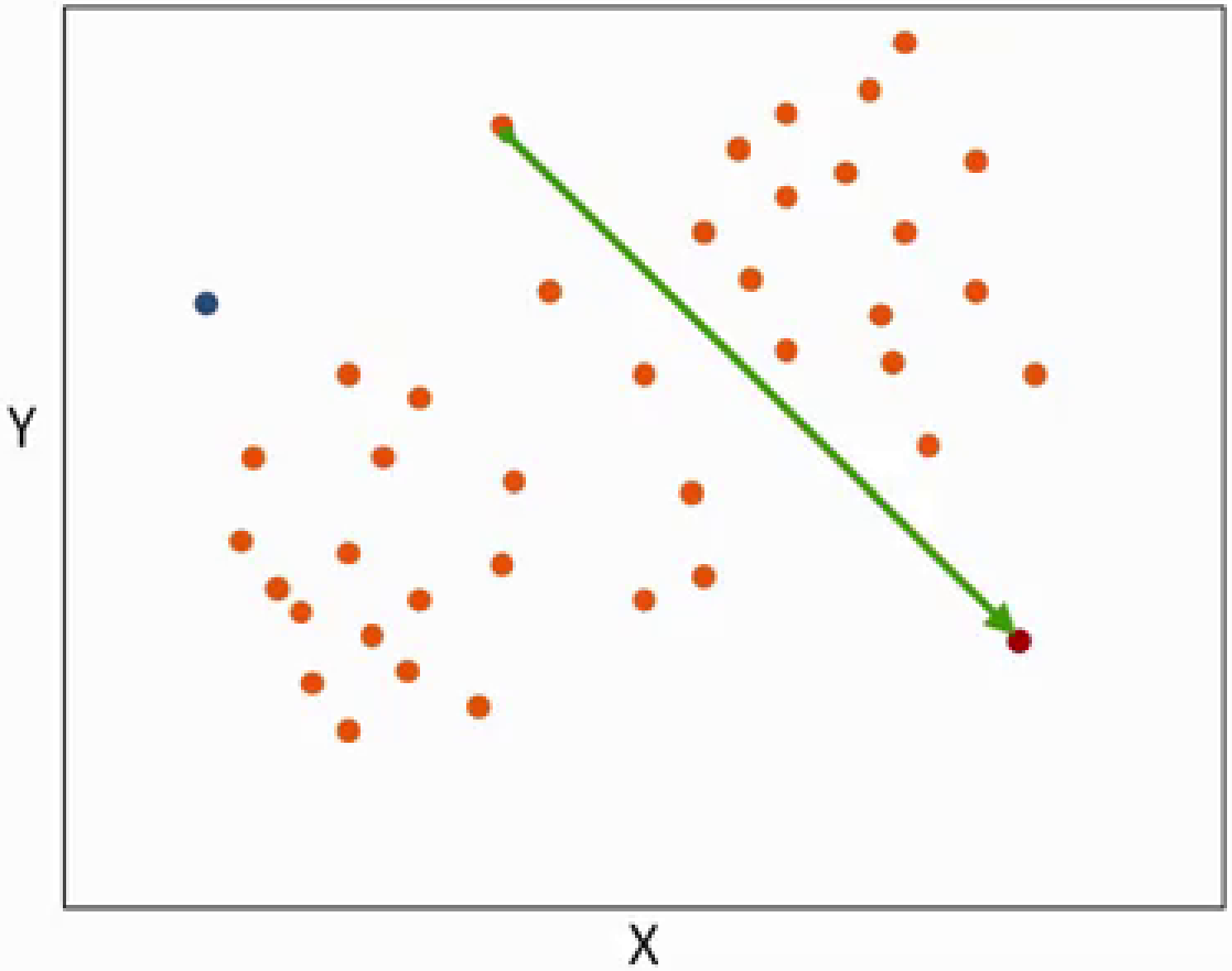


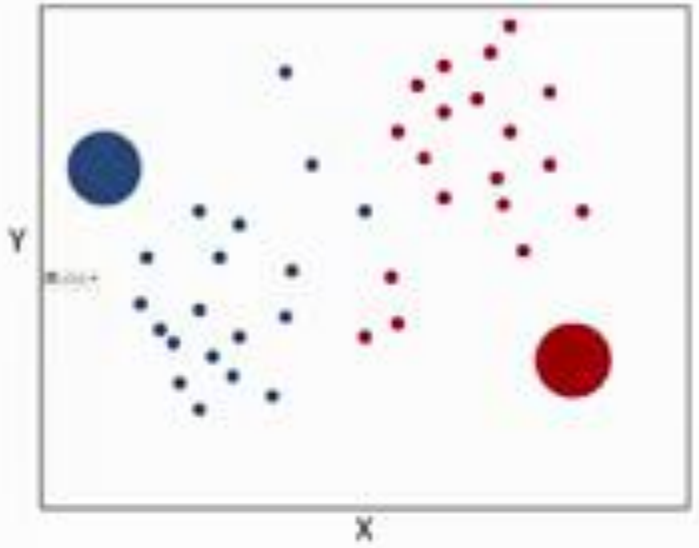
Step 1: randomly choose 2 points as the initial centroids (aka seeds) and calculate distance between points and the centroids



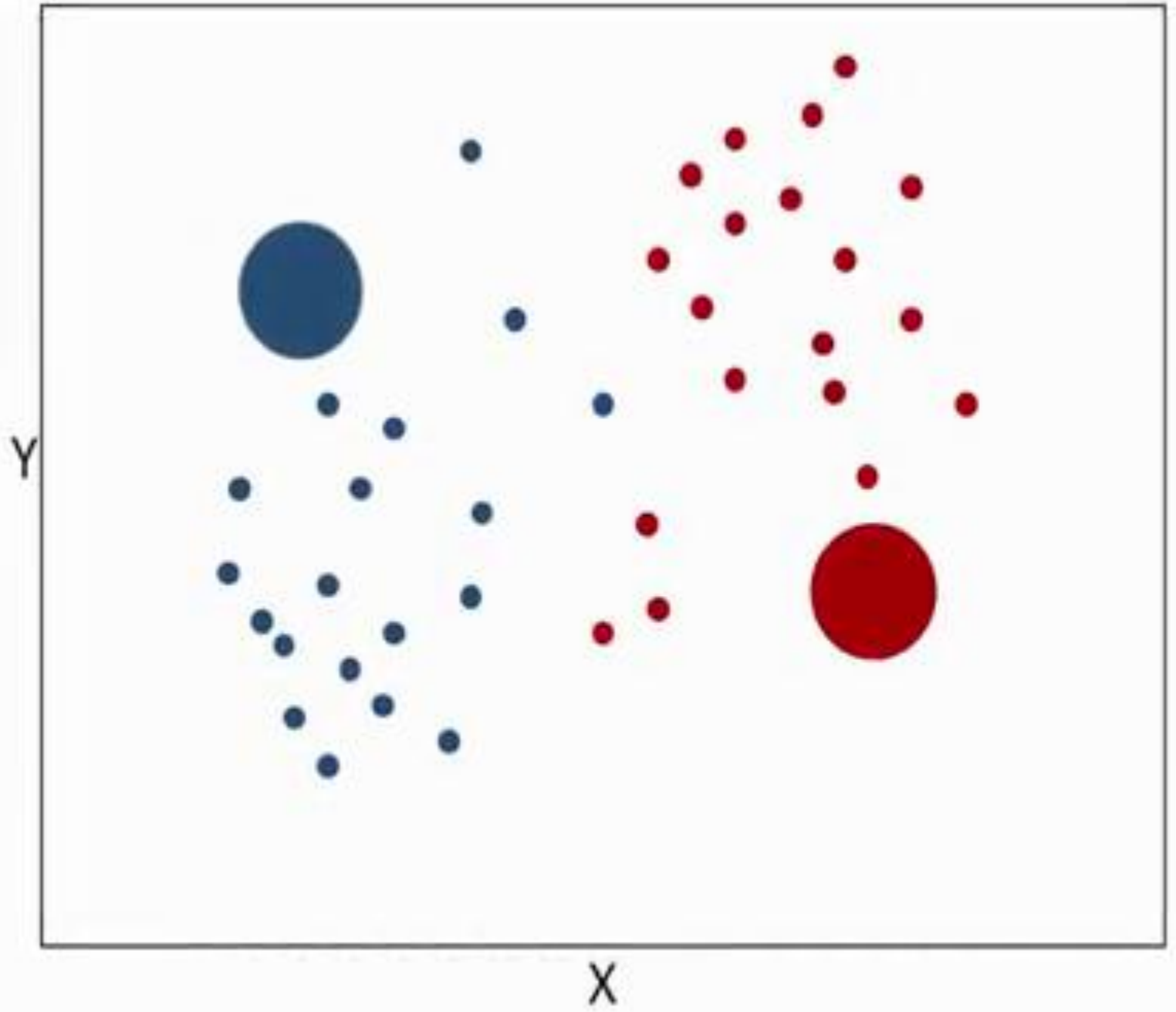


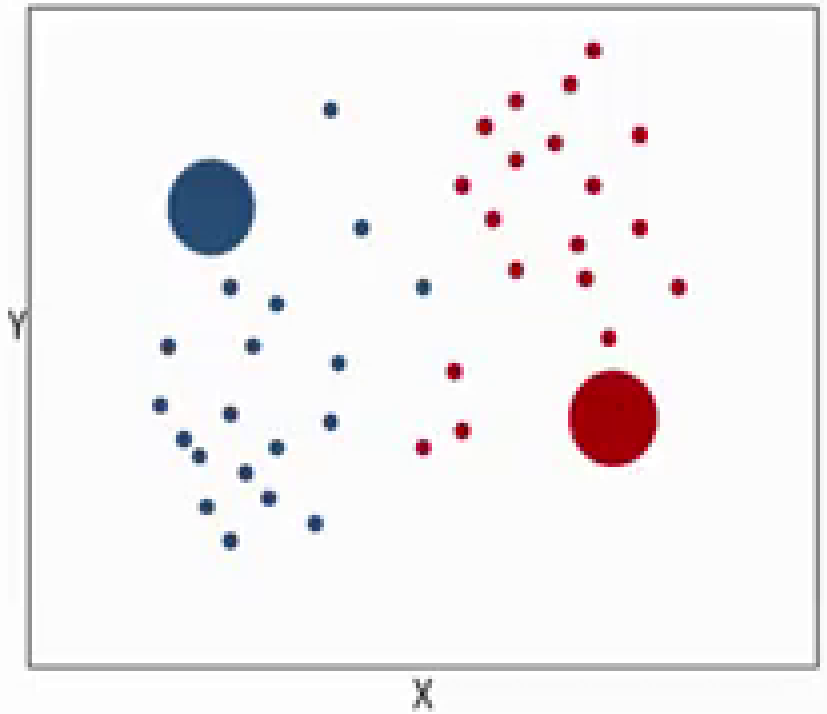
Step 1: randomly choose 2 points as the initial centroids (aka seeds) and calculate distance between points and the centroids



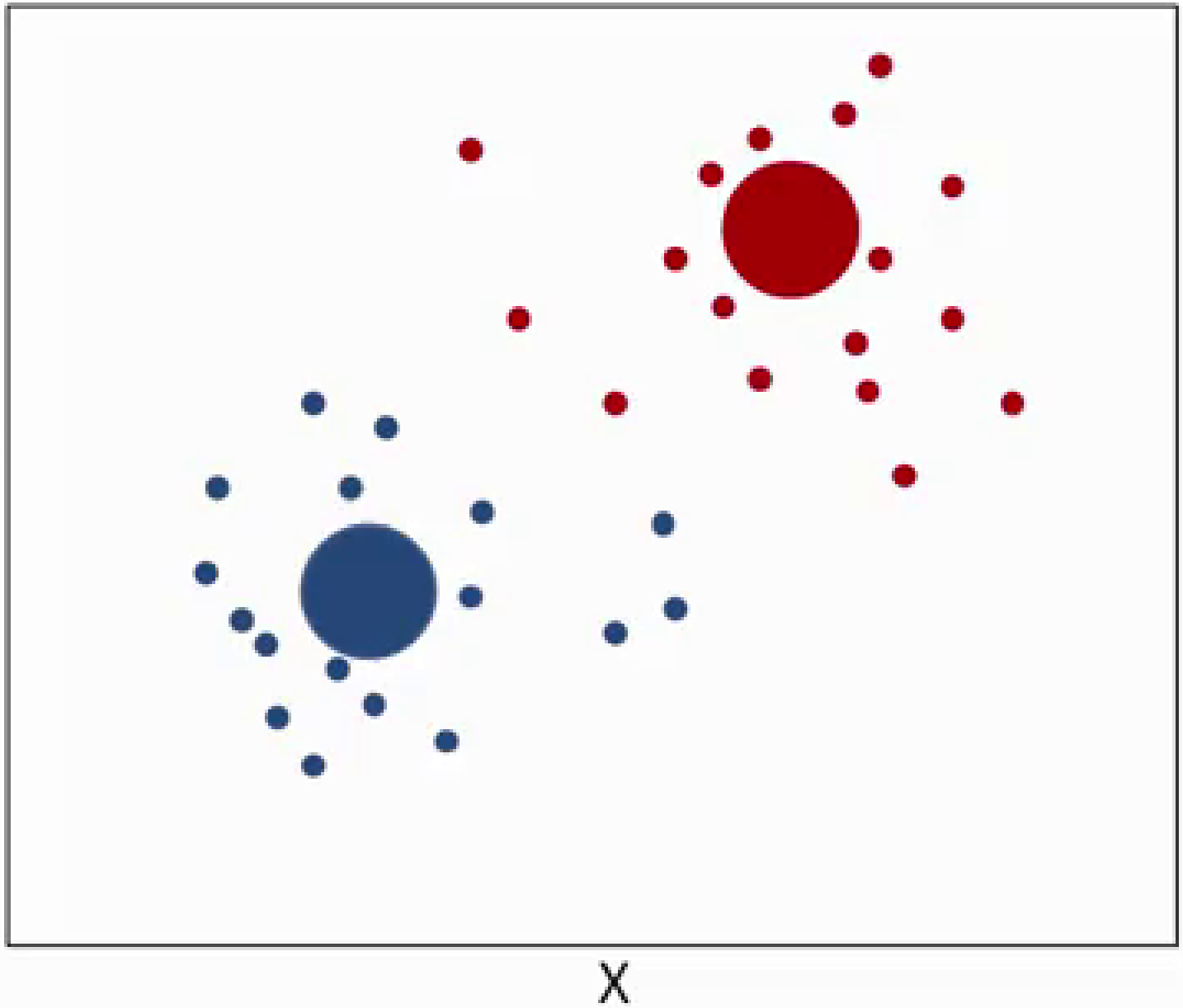


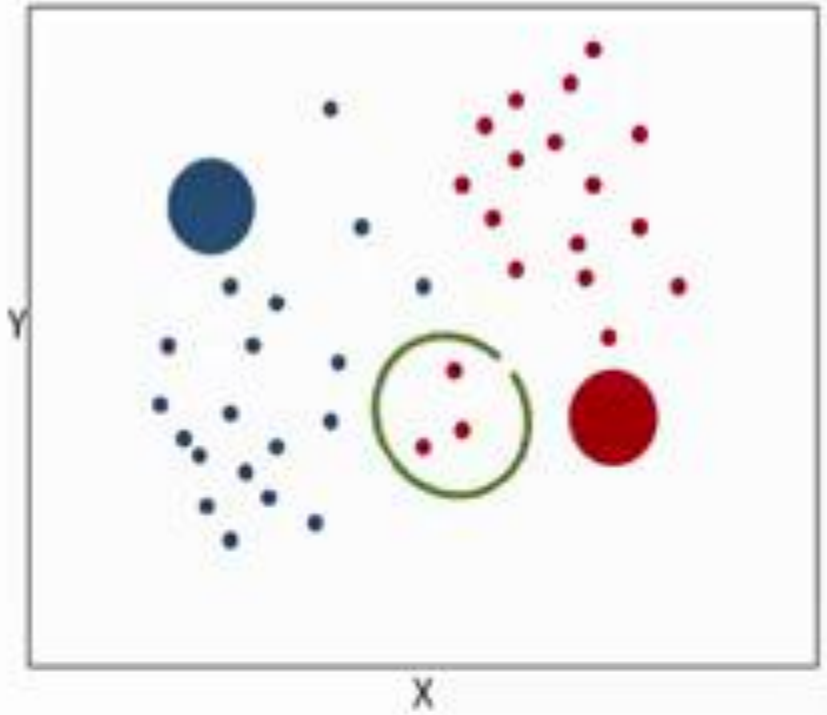
Step 2: each centroid is recalculated based on the location of the points that were assigned to it

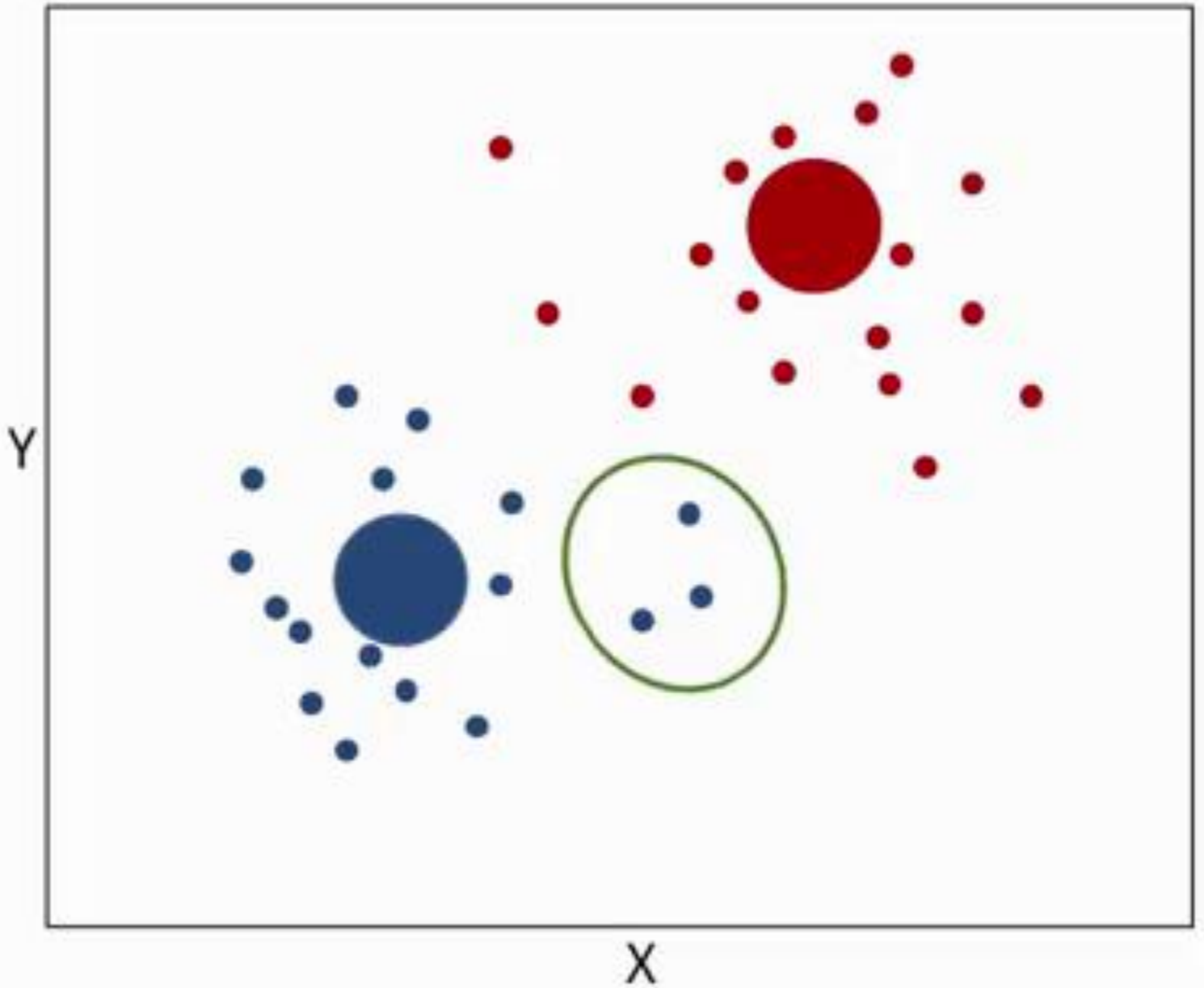




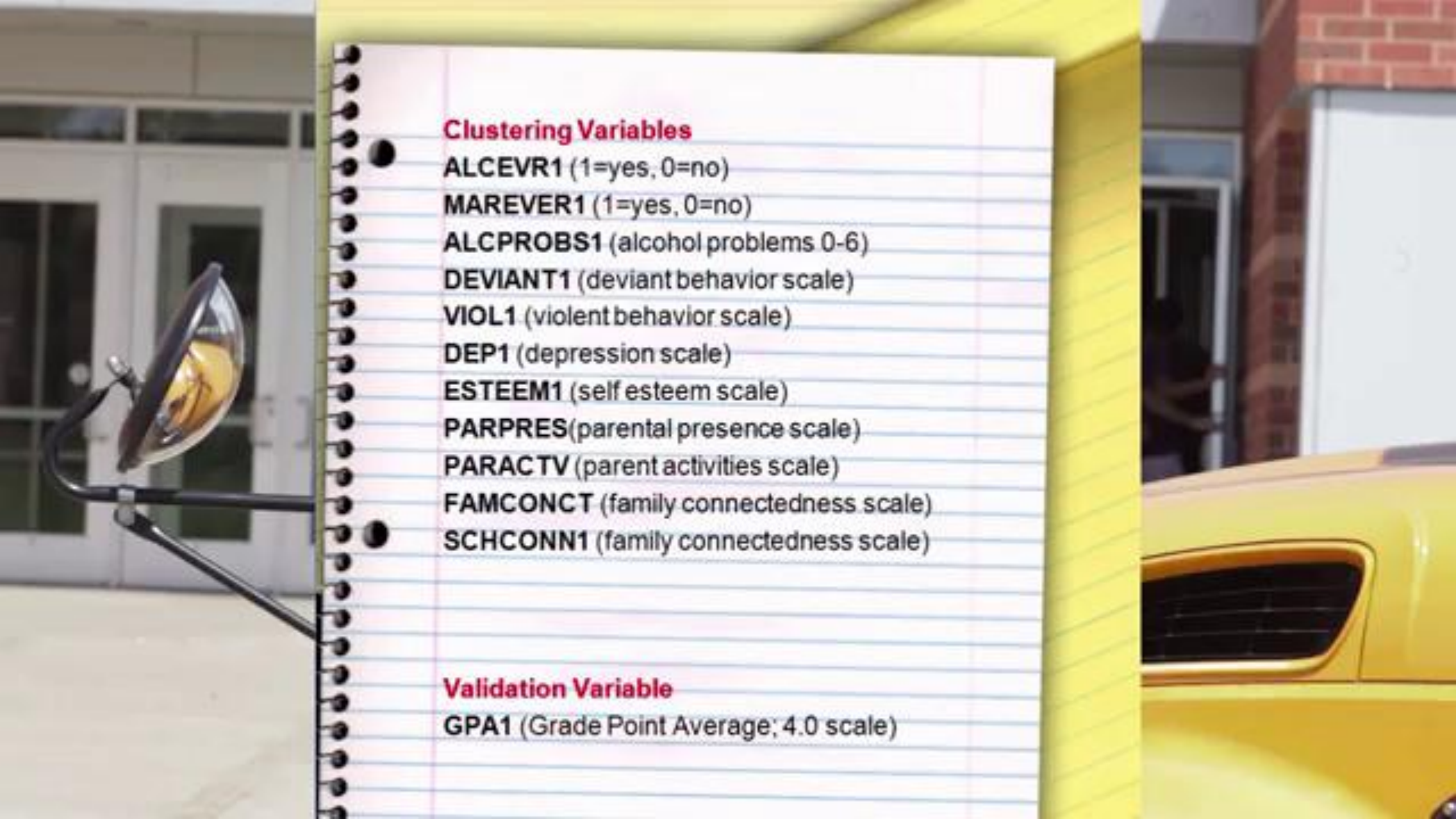
Step 3: process is repeated until the location of the centroids doesn't change very much







Step 3: process is repeated until the location of the centroids doesn't change very much


Clustering Variables
ALCEVR1 (1=yes, 0=no)
MAREVER1 (1=yes, 0=no)
ALCPROBS1 (alcohol problems 0-6)
DEVIANT1 (deviant behavior scale)
VIOL1 (violent behavior scale)
DEP1 (depression scale)
ESTEEM1 (self esteem scale)
PARPRES(parental presence scale)
PARACTV (parent activities scale)
FAMCONCT (family connectedness scale)
SCHCONN1 (family connectedness scale)
Validation Variable
GPA1 (Grade Point Average; 4.0 scale)


If we do find differences, then we have
some evidence that the clusters are valid

Module 4
Lesson 2 - Running a k-Means Cluster Analysis in Python, pt. 1

C4W4 python code for video examples.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 18 19:51:29 2016

@author: jrose01
"""

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
from sklearn.cross_validation import train_test_split
from sklearn import preprocessing
from sklearn.cluster import KMeans

"""
Data Management
"""
data = pd.read_csv("treeaddhealth.csv")

#data.reset_index(level=0, inplace=True)

#upper-case all DataFrame column names
data.columns = map(str.upper, data.columns)

# Data Management

data_clean = data.dropna()
#recode1 = {1:1, 2:0}
```

First, we will call in the libraries that we will need.

Source | Console | Object

Usage

Here you either o

Help can to an ob Inspecta

Object inspector | Variable explorer | Fi

IPython console

Console 1/A

64 bit (AMD64)]
Type "copyright", "credit

IPython 3.2.0 -- An enhan
Anaconda is brought to yo
Please check out: http://

```
"""
data = pd.read_csv("treeaddhealth.csv")

#data.reset_index(level=0, inplace=True)

#upper-case all DataFrame column names
data.columns = map(str.upper, data.columns)

# Data Management

data_clean = data.dropna()
#recode1 = {1:1, 2:0}
#data_clean['MALE']= data_clean['BIO_SEX'].map(recode1)

# subset clustering variables
cluster=data_clean[['ALCEVR1','MAREVER1','ALCPROBS1','DEVIANT1','VIOL1',
'DEP1','ESTEEM1','SCHCONN1','PARACTV', 'PARPRES','FAMCONCT']]
cluster.describe()


# standardize clustering variables to have mean=0 and sd=1
clustervar=cluster.copy()
clustervar['ALCEVR1']=preprocessing.scale(clustervar['ALCEVR1'].astype('float64'))
clustervar['ALCPROBS1']=preprocessing.scale(clustervar['ALCPROBS1'].astype('float64'))
clustervar['MAREVER1']=preprocessing.scale(clustervar['MAREVER1'].astype('float64'))
```

C4W4 python code for video examples.py

```
# subset clustering variables
cluster=data_clean[['ALCEVR1','MAREVER1','ALCPROBS1','DEVIANT1','VIOL1',
'DEP1','ESTEEM1','SCHCONN1','PARACTV', 'PARPRES','FAMCONCT']]
cluster.describe()


# standardize clustering variables to have mean=0 and sd=1
clustervar=cluster.copy()
clustervar['ALCEVR1']=preprocessing.scale(clustervar['ALCEVR1'].astype('float64'))
clustervar['ALCPROBS1']=preprocessing.scale(clustervar['ALCPROBS1'].astype('float64'))
clustervar['MAREVER1']=preprocessing.scale(clustervar['MAREVER1'].astype('float64'))
clustervar['DEP1']=preprocessing.scale(clustervar['DEP1'].astype('float64'))
clustervar['ESTEEM1']=preprocessing.scale(clustervar['ESTEEM1'].astype('float64'))
clustervar['VIOL1']=preprocessing.scale(clustervar['VIOL1'].astype('float64'))
clustervar['DEVIANT1']=preprocessing.scale(clustervar['DEVIANT1'].astype('float64'))
clustervar['FAMCONCT']=preprocessing.scale(clustervar['FAMCONCT'].astype('float64'))
clustervar['SCHCONN1']=preprocessing.scale(clustervar['SCHCONN1'].astype('float64'))
clustervar['PARACTV']=preprocessing.scale(clustervar['PARACTV'].astype('float64'))
clustervar['PARPRES']=preprocessing.scale(clustervar['PARPRES'].astype('float64'))

# split data into train and test sets
clus_train, clus_test = train_test_split(clustervar, test_size=.3, random_state=123)

# k-means cluster analysis for 1-9 clusters
from scipy.spatial.distance import cdist
clusters=range(1,10)
meandist=[]
```

We use the following code to standardize the clustering variables to have a mean of

Source Console Object

Object inspector Variable expl

IPython console

Console 1/A

```
...: Data Manage
...: """
...: data = pd.r
...:
```

C4W4 python code for video examples.py

```
#data_clean['MALE']= data_clean['BIO_SEX'].map(recode1)

# subset clustering variables
cluster=data_clean[['ALCEVR1','MAREVER1','ALCPROBS1','DEVIANT1','VIOL1',
'DEP1','ESTEEM1','SCHCONN1','PARACTV', 'PARPRES','FAMCONCT']]
cluster.describe()


# standardize clustering variables to have mean=0 and sd=1
clustervar=cluster.copy()
clustervar['ALCEVR1']=preprocessing.scale(clustervar['ALCEVR1'].astype('float64'))
clustervar['ALCPROBS1']=preprocessing.scale(clustervar['ALCPROBS1'].astype('float64'))
clustervar['MAREVER1']=preprocessing.scale(clustervar['MAREVER1'].astype('float64'))
clustervar['DEP1']=preprocessing.scale(clustervar['DEP1'].astype('float64'))
clustervar['ESTEEM1']=preprocessing.scale(clustervar['ESTEEM1'].astype('float64'))
clustervar['VIOL1']=preprocessing.scale(clustervar['VIOL1'].astype('float64'))
clustervar['DEVIANT1']=preprocessing.scale(clustervar['DEVIANT1'].astype('float64'))
clustervar['FAMCONCT']=preprocessing.scale(clustervar['FAMCONCT'].astype('float64'))
clustervar['SCHCONN1']=preprocessing.scale(clustervar['SCHCONN1'].astype('float64'))
clustervar['PARACTV']=preprocessing.scale(clustervar['PARACTV'].astype('float64'))
clustervar['PARPRES']=preprocessing.scale(clustervar['PARPRES'].astype('float64'))

# split data into train and test sets
clus_train, clus_test = train_test_split(clustervar, test_size=.3, random_state=123)

# k-means cluster analysis for 1-9 clusters
from scipy.spatial.distance import cdist
clusters=range(1,10)
```

a test data set consisting of
the other 30% of the observations.

Source | Console | Object

Object inspector | Variable explorer

IPython console

Console 1/A

```
clustervar['DEP1']=
   ...:
clustervar['ESTEEM1
   ...:
clustervar['VIOL1']
```

C4W4 python code for video examples.py

```
clustervar['MAREVER1']=preprocessing.scale(clustervar['MAREVER1'].astype('float64'))
clustervar['DEP1']=preprocessing.scale(clustervar['DEP1'].astype('float64'))
clustervar['ESTEEM1']=preprocessing.scale(clustervar['ESTEEM1'].astype('float64'))
clustervar['VIOL1']=preprocessing.scale(clustervar['VIOL1'].astype('float64'))
clustervar['DEVIANT1']=preprocessing.scale(clustervar['DEVIANT1'].astype('float64'))
clustervar['FAMCONCT']=preprocessing.scale(clustervar['FAMCONCT'].astype('float64'))
clustervar['SCHCONN1']=preprocessing.scale(clustervar['SCHCONN1'].astype('float64'))
clustervar['PARACTV']=preprocessing.scale(clustervar['PARACTV'].astype('float64'))
clustervar['PARPRES']=preprocessing.scale(clustervar['PARPRES'].astype('float64'))

# split data into train and test sets
clus_train, clus_test = train_test_split(clustervar, test_size=.3, random_state=123)

# k-means cluster analysis for 1-9 clusters
from scipy.spatial.distance import cdist
clusters=range(1,10)
meandist=[]

for k in clusters:
    model=KMeans(n_clusters=k)
    model.fit(clus_train)
    clusassign=model.predict(clus_train)
    meandist.append(su                                    ean'), axis=1))
    / clus_train.shape[0])

"""
Plot average distance from observations from the cluster centroid
```

we don't know how many clusters actually exist in the population for

C4W4 python code for video examples.py

```
clustervar['MAREVER1']=preprocessing.scale(clustervar['MAREVER1'].astype('float64'))
clustervar['DEP1']=preprocessing.scale(clustervar['DEP1'].astype('float64'))
clustervar['ESTEEM1']=preprocessing.scale(clustervar['ESTEEM1'].astype('float64'))
clustervar['VIOL1']=preprocessing.scale(clustervar['VIOL1'].astype('float64'))
clustervar['DEVIANT1']=preprocessing.scale(clustervar['DEVIANT1'].astype('float64'))
clustervar['FAMCONCT']=preprocessing.scale(clustervar['FAMCONCT'].astype('float64'))
clustervar['SCHCONN1']=preprocessing.scale(clustervar['SCHCONN1'].astype('float64'))
clustervar['PARACTV']=preprocessing.scale(clustervar['PARACTV'].astype('float64'))
clustervar['PARPRES']=preprocessing.scale(clustervar['PARPRES'].astype('float64'))

# split data into train and test sets
clus_train, clus_test = train_test_split(clustervar, test_size=.3, random_state=123)

# k-means cluster analysis for 1-9 clusters
from scipy.spatial.distance import cdist
clusters=range(1,10)
meandist=[]

for k in clusters:
    model=KMeans(n_clusters=k)
    model.fit(clus_train)
    clusassign=model.predict(clus_train)
    meandist.append(sum(np.min(cdist(clus_train, model.cluster_centers_, 'euclidean'), axis=1))
    / clus_train.shape[0])

"""
Plot average distance from observations from the cluster centroid
```

a range of values on the number of clusters before we begin we'll import

C4W4 python code for video examples.py

```
clustervar['MAREVER1']=preprocessing.scale(clustervar['MAREVER1'].astype('float64'))
clustervar['DEP1']=preprocessing.scale(clustervar['DEP1'].astype('float64'))
clustervar['ESTEEM1']=preprocessing.scale(clustervar['ESTEEM1'].astype('float64'))
clustervar['VIOL1']=preprocessing.scale(clustervar['VIOL1'].astype('float64'))
clustervar['DEVIANT1']=preprocessing.scale(clustervar['DEVIANT1'].astype('float64'))
clustervar['FAMCONCT']=preprocessing.scale(clustervar['FAMCONCT'].astype('float64'))
clustervar['SCHCONN1']=preprocessing.scale(clustervar['SCHCONN1'].astype('float64'))
clustervar['PARACTV']=preprocessing.scale(clustervar['PARACTV'].astype('float64'))
clustervar['PARPRES']=preprocessing.scale(clustervar['PARPRES'].astype('float64'))

# split data into train and test sets
clus_train, clus_test = train_test_split(clustervar, test_size=.3, random_state=123)

# k-means cluster analysis for 1-9 clusters
from scipy.spatial.distance import cdist
clusters=range(1,10)
meandist=[]

for k in clusters:
    model=KMeans(n_clusters=k)
    model.fit(clus_train)
    clusassign=model.predict(clus_train)
    meandist.append(sum(np.min(cdist(clus_train, model.cluster_centers_, 'euclidean'), axis=1))
    / clus_train.shape[0])

"""
Plot average distance from observations from the cluster centroid
```

the cdist function from the scipy.spatial.distance library.

C4W4 python code for video examples.py

```
clustervar['MAREVER1']=preprocessing.scale(clustervar['MAREVER1'].astype('float64'))
clustervar['DEP1']=preprocessing.scale(clustervar['DEP1'].astype('float64'))
clustervar['ESTEEM1']=preprocessing.scale(clustervar['ESTEEM1'].astype('float64'))
clustervar['VIOL1']=preprocessing.scale(clustervar['VIOL1'].astype('float64'))
clustervar['DEVIANT1']=preprocessing.scale(clustervar['DEVIANT1'].astype('float64'))
clustervar['FAMCONCT']=preprocessing.scale(clustervar['FAMCONCT'].astype('float64'))
clustervar['SCHCONN1']=preprocessing.scale(clustervar['SCHCONN1'].astype('float64'))
clustervar['PARACTV']=preprocessing.scale(clustervar['PARACTV'].astype('float64'))
clustervar['PARPRES']=preprocessing.scale(clustervar['PARPRES'].astype('float64'))

# split data into train and test sets
clus_train, clus_test = train_test_split(clustervar, test_size=.3, random_state=123)

# k-means cluster analysis for 1-9 clusters
from scipy.spatial.distance import cdist
clusters=range(1,10)
meandist=[]

for k in clusters:
    model=KMeans(n_clusters=k)
    model.fit(clus_train)
    clusassign=model.predict(clus_train)
    meandist.append(sum(np.min(cdist(clus_train, model.cluster_centers_, 'euclidean'), axis=1))
    / clus_train.shape[0])

"""
Plot average distance from observations from the cluster centroid
```

which will give us the cluster solutions
for k equals 1 to k equals 9 clusters.

C4W4 python code for video examples.py

```
clustervar['MAREVER1']=preprocessing.scale(clustervar['MAREVER1'].astype('float64'))
clustervar['DEP1']=preprocessing.scale(clustervar['DEP1'].astype('float64'))
clustervar['ESTEEM1']=preprocessing.scale(clustervar['ESTEEM1'].astype('float64'))
clustervar['VIOL1']=preprocessing.scale(clustervar['VIOL1'].astype('float64'))
clustervar['DEVIANT1']=preprocessing.scale(clustervar['DEVIANT1'].astype('float64'))
clustervar['FAMCONCT']=preprocessing.scale(clustervar['FAMCONCT'].astype('float64'))
clustervar['SCHCONN1']=preprocessing.scale(clustervar['SCHCONN1'].astype('float64'))
clustervar['PARACTV']=preprocessing.scale(clustervar['PARACTV'].astype('float64'))
clustervar['PARPRES']=preprocessing.scale(clustervar['PARPRES'].astype('float64'))

# split data into train and test sets
clus_train, clus_test = train_test_split(clustervar, test_size=.3, random_state=123)

# k-means cluster analysis for 1-9 clusters
from scipy.spatial.distance import cdist
clusters=range(1,10)
meandist=[]

for k in clusters:
    model=KMeans(n_clusters=k)
    model.fit(clus_train)
    clusassign=model.predict(clus_train)
    meandist.append(sum(np.min(cdist(clus_train, model.cluster_centers_, 'euclidean'), axis=1))
    / clus_train.shape[0])

"""
Plot average distance from observations from the cluster centroid
```

The formula first calculates the distance between each observation and the cluster

C4W4 python code for video examples.py

```
clustervar['MAREVER1']=preprocessing.scale(clustervar['MAREVER1'].astype('float64'))
clustervar['DEP1']=preprocessing.scale(clustervar['DEP1'].astype('float64'))
clustervar['ESTEEM1']=preprocessing.scale(clustervar['ESTEEM1'].astype('float64'))
clustervar['VIOL1']=preprocessing.scale(clustervar['VIOL1'].astype('float64'))
clustervar['DEVIANT1']=preprocessing.scale(clustervar['DEVIANT1'].astype('float64'))
clustervar['FAMCONCT']=preprocessing.scale(clustervar['FAMCONCT'].astype('float64'))
clustervar['SCHCONN1']=preprocessing.scale(clustervar['SCHCONN1'].astype('float64'))
clustervar['PARACTV']=preprocessing.scale(clustervar['PARACTV'].astype('float64'))
clustervar['PARPRES']=preprocessing.scale(clustervar['PARPRES'].astype('float64'))

# split data into train and test sets
clus_train, clus_test = train_test_split(clustervar, test_size=.3, random_state=123)

# k-means cluster analysis for 1-9 clusters
from scipy.spatial.distance import cdist
clusters=range(1,10)
meandist=[]

for k in clusters:
    model=KMeans(n_clusters=k)
    model.fit(clus_train)
    clusassign=model.predict(clus_train)
    meandist.append(su                                    dean'), axis=1))
    / clus_train.shape[0])

"""
Plot average distance from observations from the cluster centroid
```

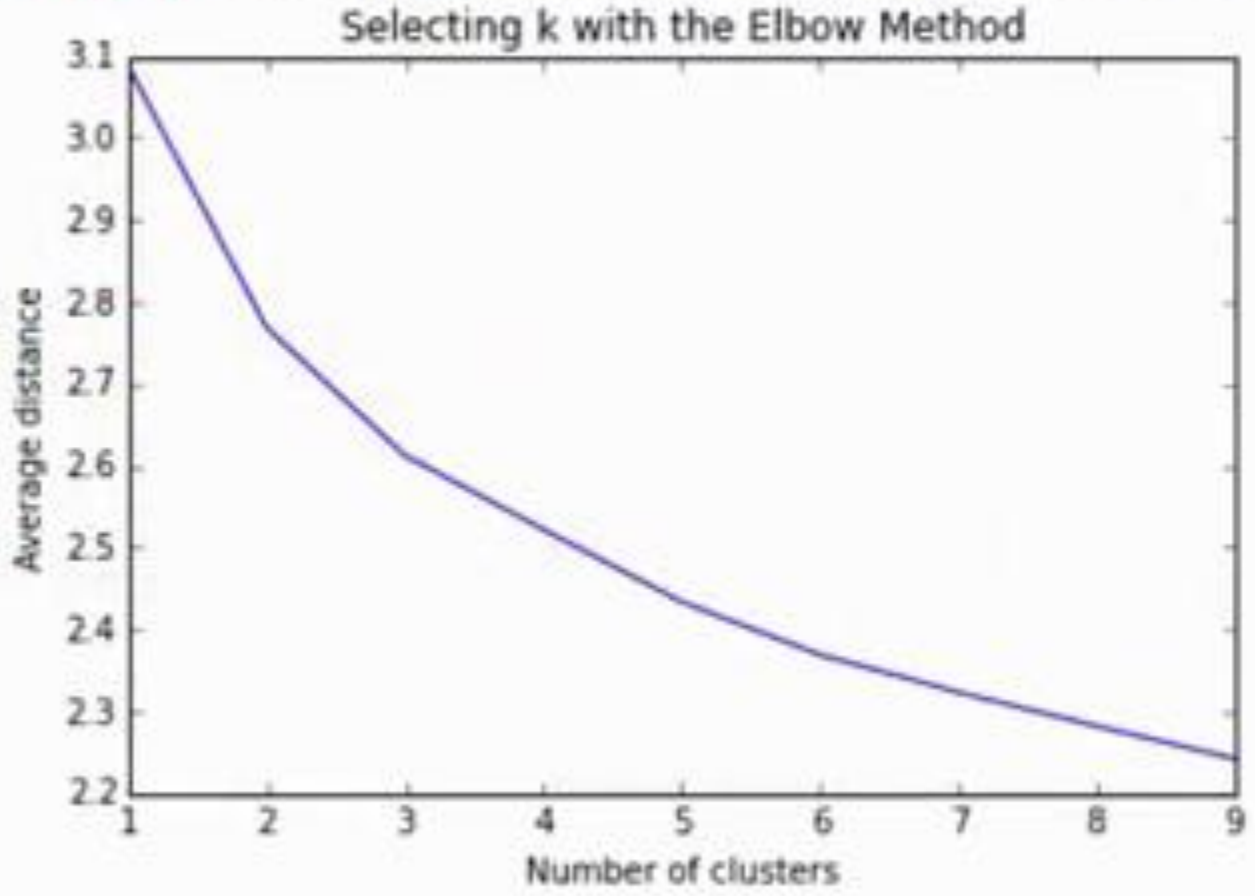
each centroid taking the smallest distance as the value of the minimum,

C4W4 python code for video examples.py

```
clustervar['MAREVER1']=preprocessing.scale(clustervar['MAREVER1'].astype('float64'))
clustervar['DEP1']=preprocessing.scale(clustervar['DEP1'].astype('float64'))
clustervar['ESTEEM1']=preprocessing.scale(clustervar['ESTEEM1'].astype('float64'))
clustervar['VIOL1']=preprocessing.scale(clustervar['VIOL1'].astype('float64'))
clustervar['DEVIANT1']=preprocessing.scale(clustervar['DEVIANT1'].astype('float64'))
clustervar['FAMCONCT']=preprocessing.scale(clustervar['FAMCONCT'].astype('float64'))
clustervar['SCHCONN1']=preprocessing.scale(clustervar['SCHCONN1'].astype('float64'))
clustervar['PARACTV']=preprocessing.scale(clustervar['PARACTV'].astype('float64'))
clustervar['PARPRES']=preprocessing.scale(clustervar['PARPRES'].astype('float64'))

# split data into train and test sets
clus_train, clus_test = train_test_split(clustervar, test_size=.3, random_state=123)

# k-means cluster analysis for 1-9 clusters
from scipy.spatial.distance import cdist
clusters=range(1,10)
meandist=[]

for k in clusters:
    model=KMeans(n_clusters=k)
    model.fit(clus_train)
    clusassign=model.predict(clus_train)
    meandist.append(sum(np.min(cdist(clus_train, model.cluster_centers_, 'euclidean'), axis=1))
    / clus_train.shape[0])

"""
Plot average distance from observations from the cluster centroid
```

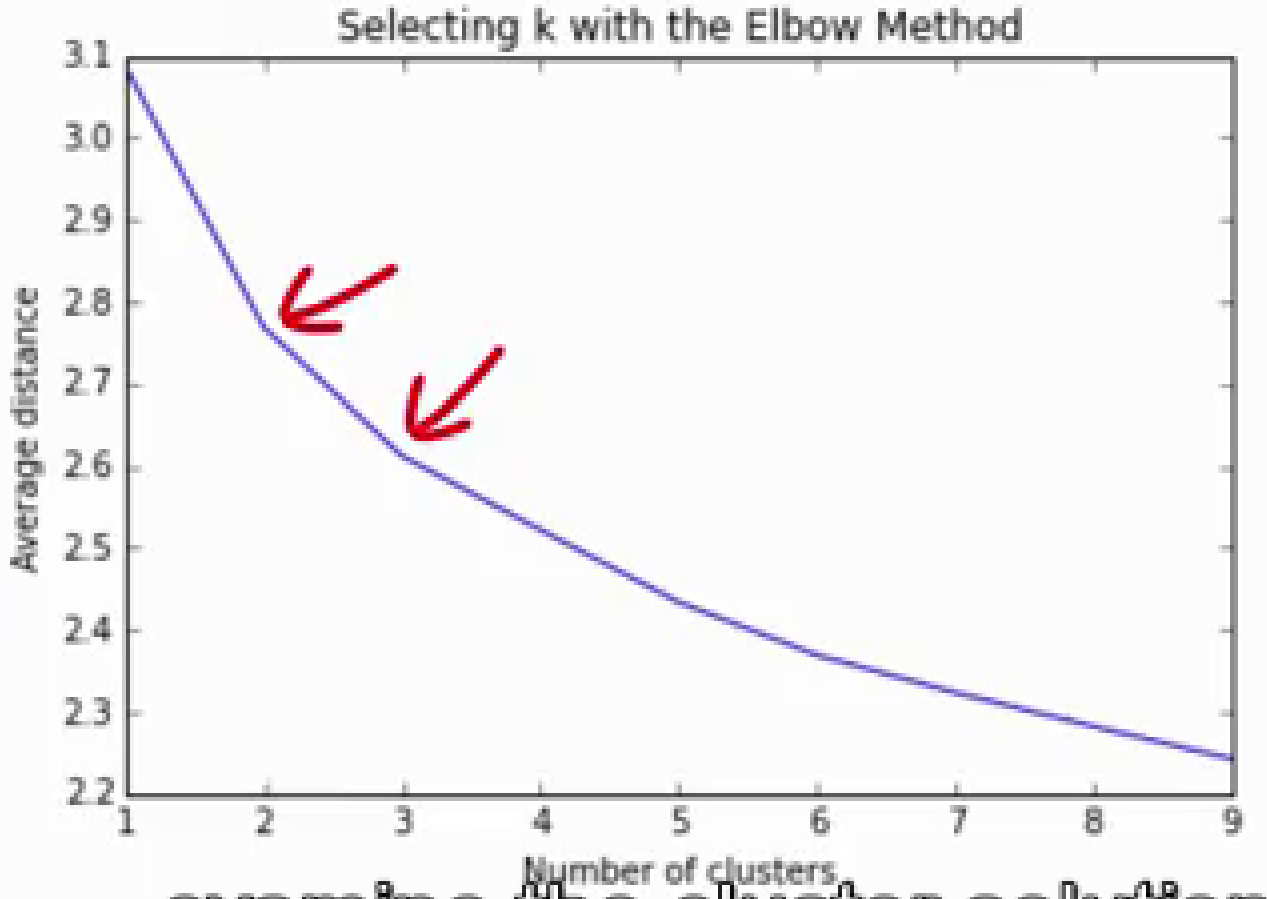
by the number of observations in the clus_train data set where the .shape with

```
...: plt.ylabel('Average distance')
...: plt.title('Selecting k with the Elbow Method')
...:
Out[5]: <matplotlib.text.Text at 0xd761588>
```




```
In [6]:
```

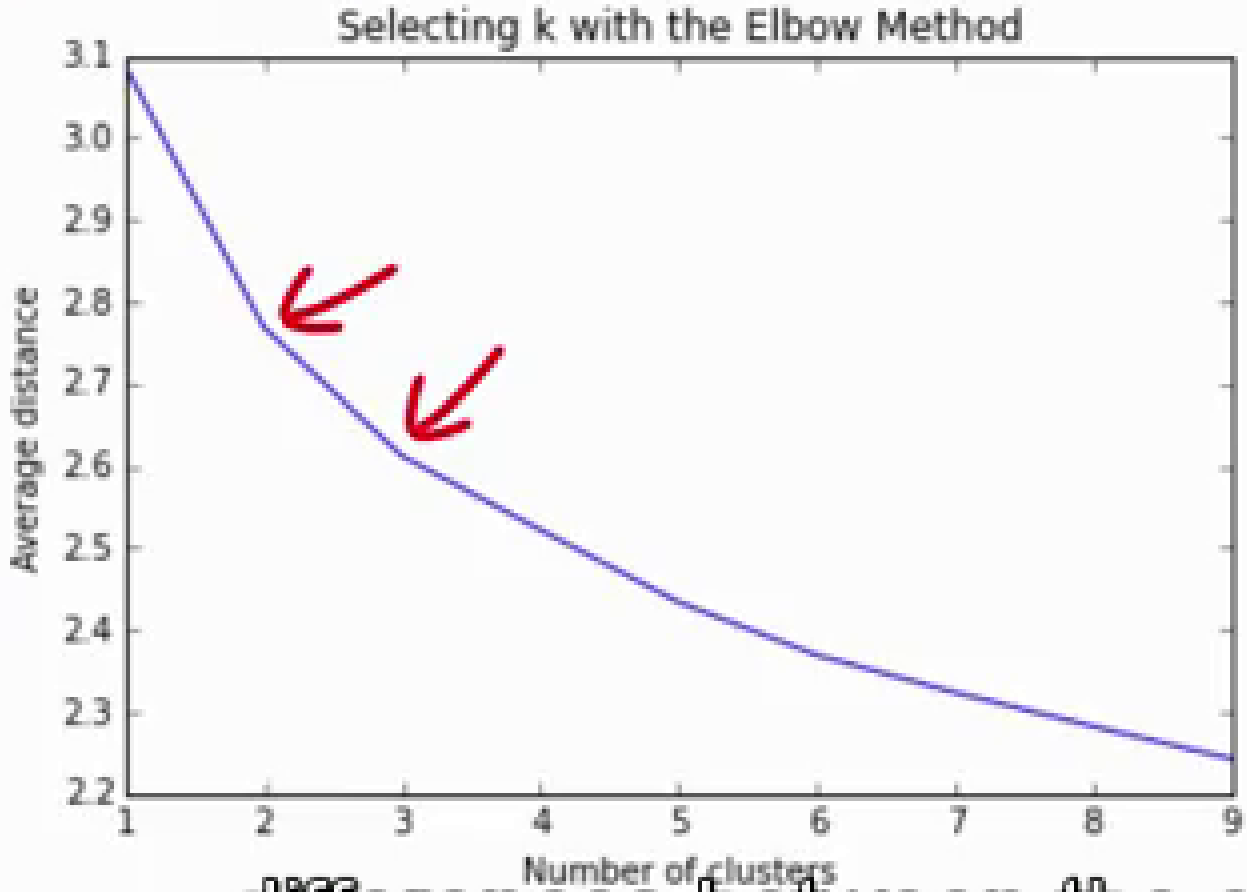
```
...: plt.ylabel('Average distance')
...: plt.title('Selecting k with the Elbow Method')
...:
Out[5]: <matplotlib.text.Text at 0xd761588>
```



examine the cluster solutions for at least

In [6 the two and three cluster solutions to see

```
...: plt.ylabel('Average distance')
...: plt.title('Selecting k with the Elbow Method')
...:
Out[5]: <matplotlib.text.Text at 0xd761588>
```

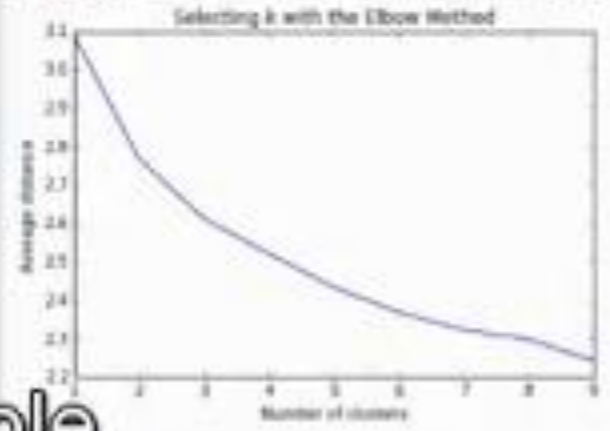


differences between the clusters on our external validation variable GPA.

In [6]:

Module 4
Lesson 3 - Running a k-Means Cluster Analysis in Python, pt. 2

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of clusters to choose
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=model3.labels_,)
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterplot of Canonical Variables for 3 Clusters')
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus
```



As an example,
let's interpret the 3 cluster solution.

C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of clusters to choose
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=model3.labels_,)
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterplot of Canonical Variables for 3 Clusters')
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

So we create an object, model 3, which will contain the results from the cluster

C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of clusters to choose
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=model3.labels_,)
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterplot
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

analysis with 3 clusters =KMeans,
and in parenthesis, n_clusters=3.

Source Console

Object inspector

IPython console

Console 1/A

C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of clusters to choose
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=model3.labels_,)
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterplot o
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

The first thing we want to try is to graph the clusters in a scatter plot to

Source Console

Object inspector

IPython console

Console 1/A

31
30
29

C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of clusters to choose
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=model3.labels_,)
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterplot of Canonical variables for 3 clusters')
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

see whether or not they overlap with each other in terms of their location in

Source Console

Object inspector

IPython console

Console 1/A

31
30
29

C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of clusters to choose
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=model3.labels_,)
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterpl
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

However, with 11 clustering variables
that means we have 11 dimensions,

Source Console

Object inspector

IPython console

Console 1/A

31
30
29

C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of clusters to choose
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=model3.labels_,)
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterplot of Canonical Variables for 3 Clusters')
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

which would be impossible to visualize.

Source Console

Object inspector

IPython console

Console 1/A

C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of clusters to choose
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=model3.labels_,)
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterplot
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

A scatter plot will work to visualize a few dimensions, but not 11 dimensions.

Source Console

Object inspector

IPython console

Console 1/A

31
30
29

C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_column
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatt
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

Canonical Discriminant Analysis

- Creates a smaller number of variables
- Linear combinations of clustering variables
- Canonical variables are ordered by proportion of variance accounted for

called canonical variables, are ordered in terms of the proportion of variance and

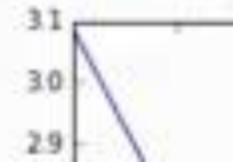
C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_column
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterplot of Canonical
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

Canonical Discriminant Analysis

- Creates a smaller number of variables
- Linear combinations of clustering variables
- Canonical variables are ordered by proportion of variance accounted for

So the first canonical variable will count for

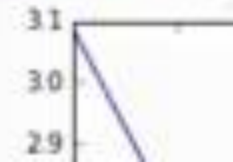


C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_column
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterplot of Canonical Variables
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

Canonical Discriminant Analysis

- Creates a smaller number of variables
- Linear combinations of clustering variables
- Canonical variables are ordered by proportion of variance accounted for
- Majority of variance is accounted for by first few canonical variables



C4W4 python code for video examples.py*

```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of clusters to choose
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')

# Interpret 3 cluster solution
model3=KMeans(n_clusters=3)
model3.fit(clus_train)
clusassign=model3.predict(clus_train)
# plot clusters

from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(clus_train)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=model3.labels_,)
plt.xlabel('Canonical variable 1')
plt.ylabel('Canonical variable 2')
plt.title('Scatterplot of
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
```

PCA(2) asks Python to return the two first canonical variables.

Source Console

Object inspector

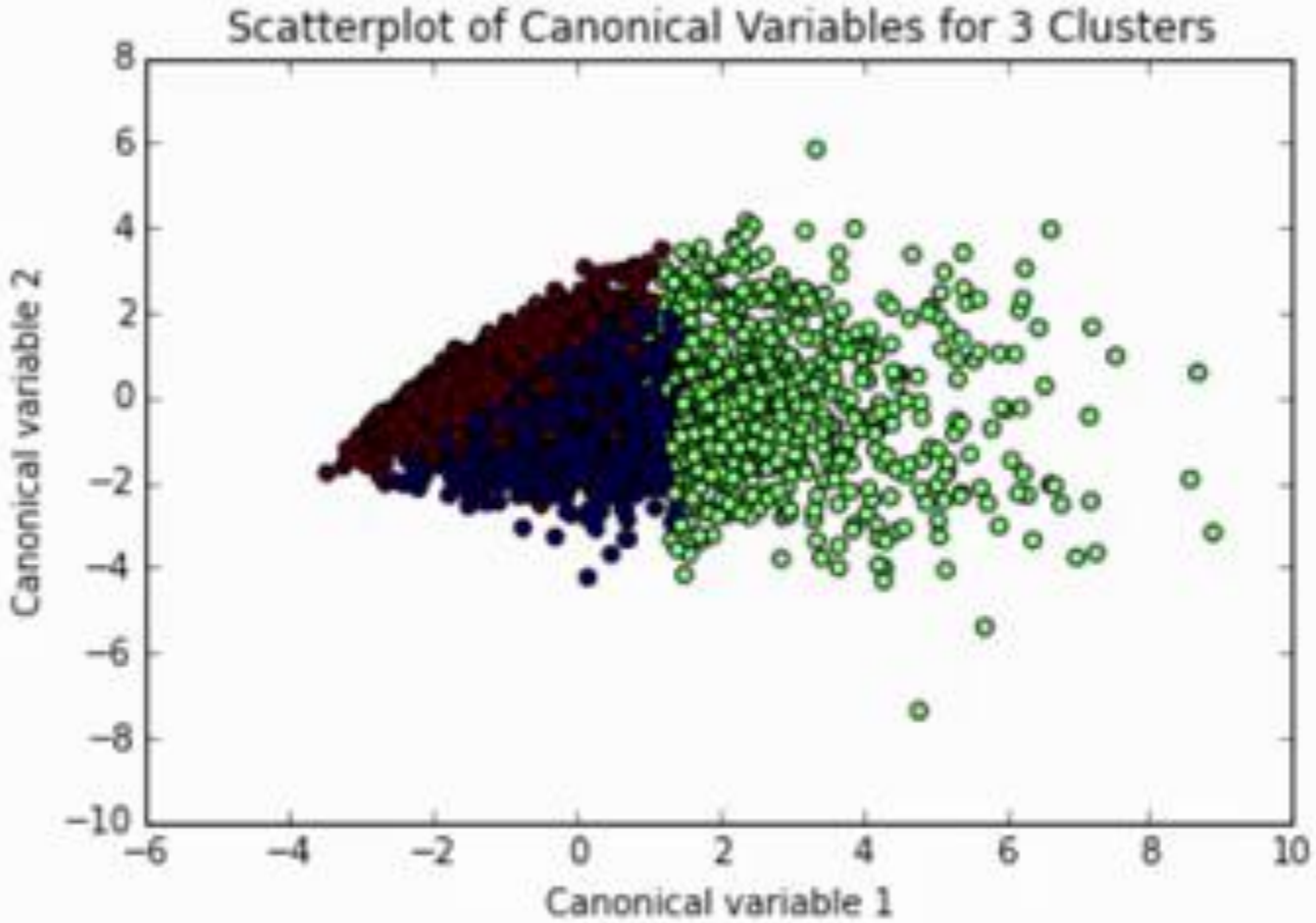
IPython console

Console 1/A

31
30
29

Console 1/A

```
if self._edgecolors == str('face'):
```

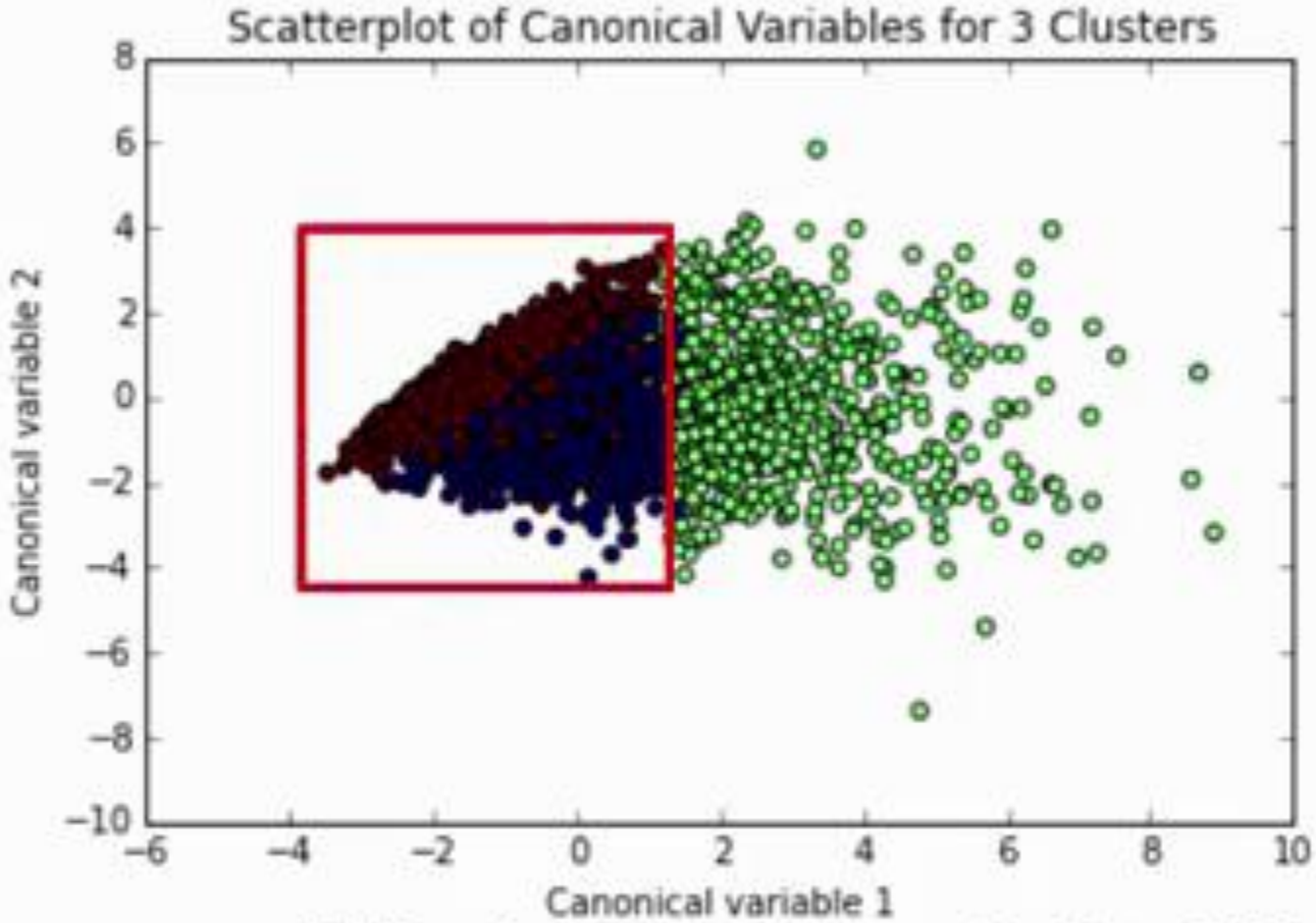


In [13]:

Here is the scatter plot.

Console 1/A

```
if self._edgecolors == str('face'):
```

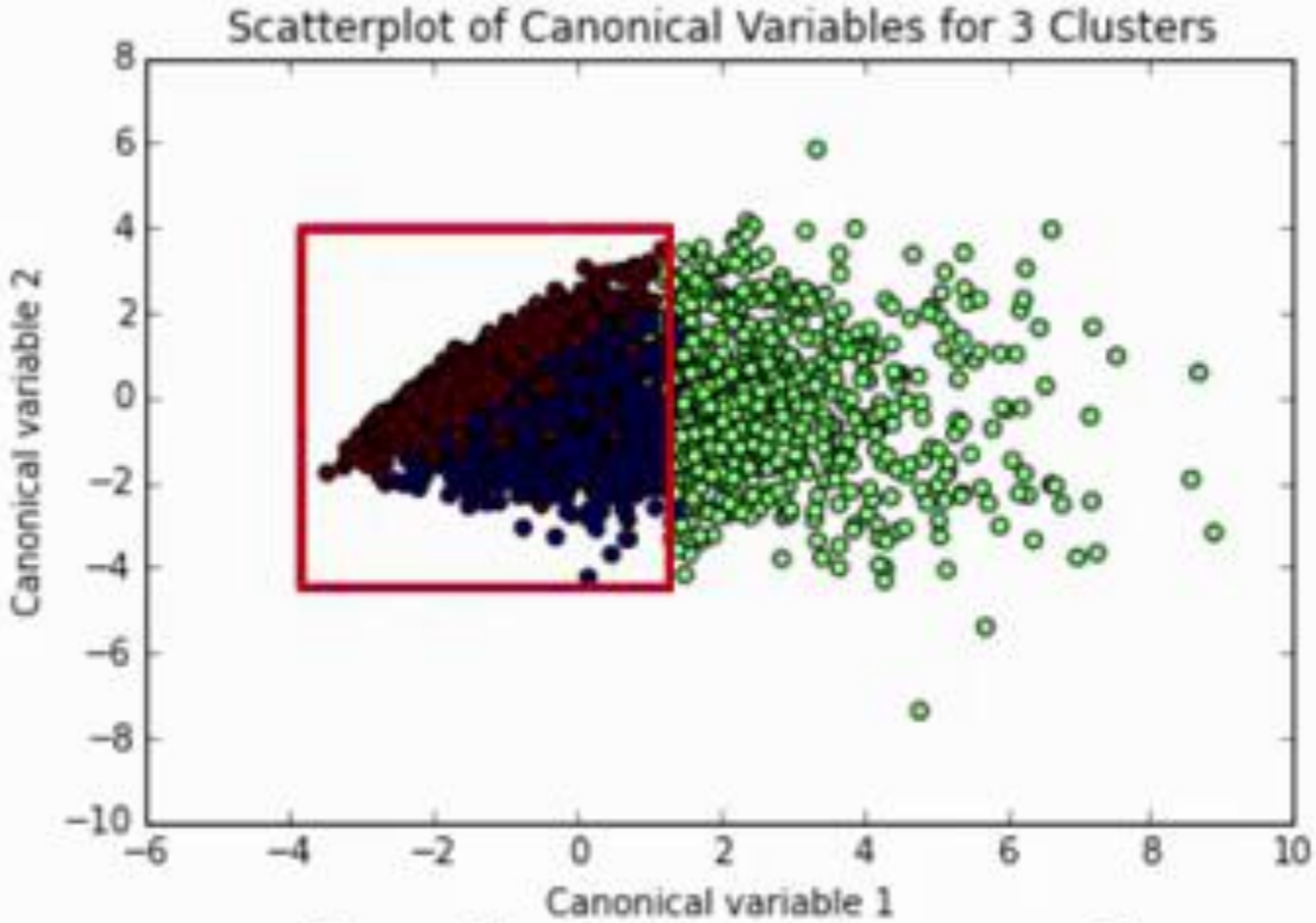


What this shows is that these two clusters are densely packed,

In [13]:

Console 1/A

```
if self._edgecolors == str('face'):
```

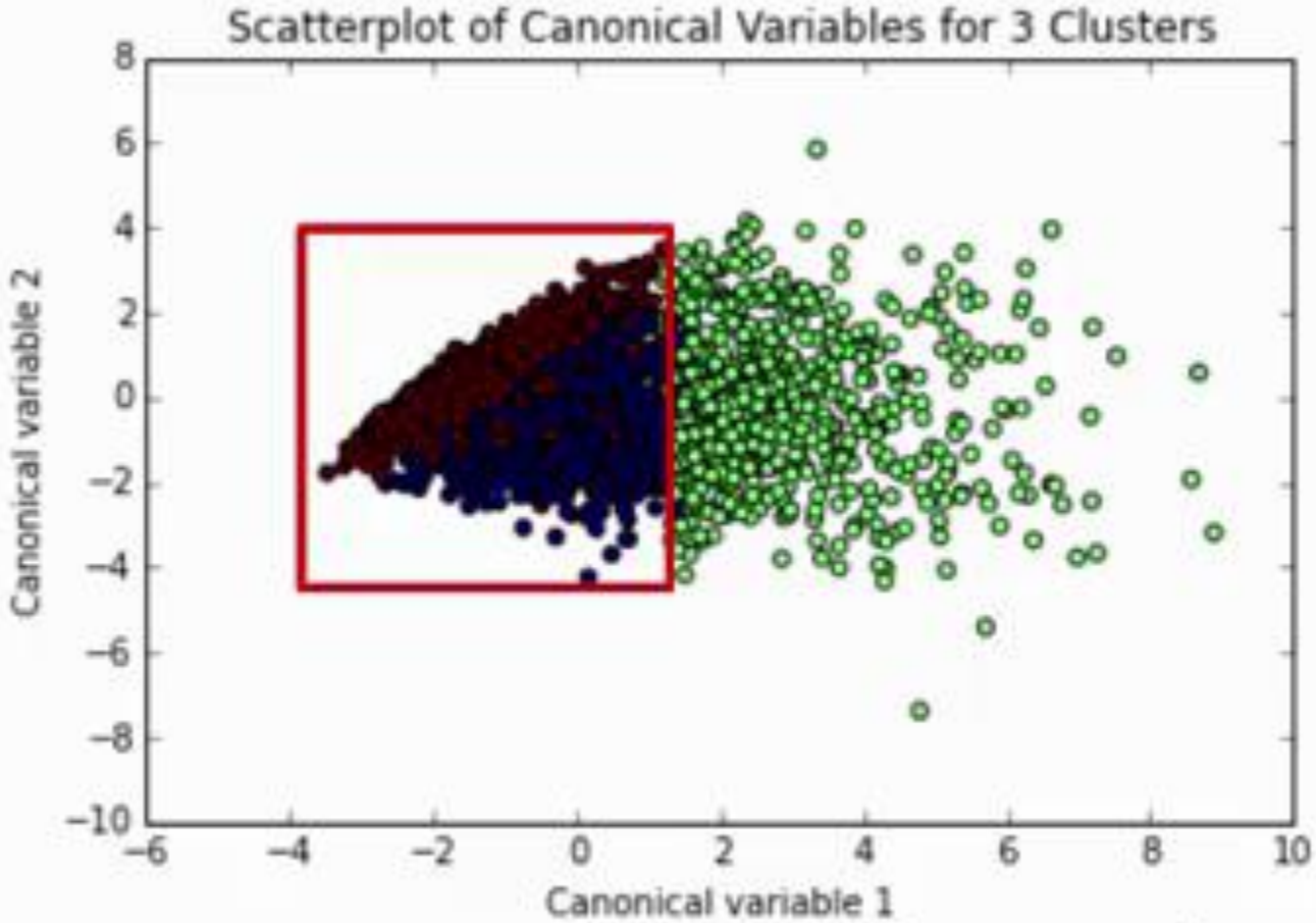


meaning that the observations within the clusters are pretty highly correlated with

In [13]:

Console 1/A

```
if self._edgecolors == str('face'):
```



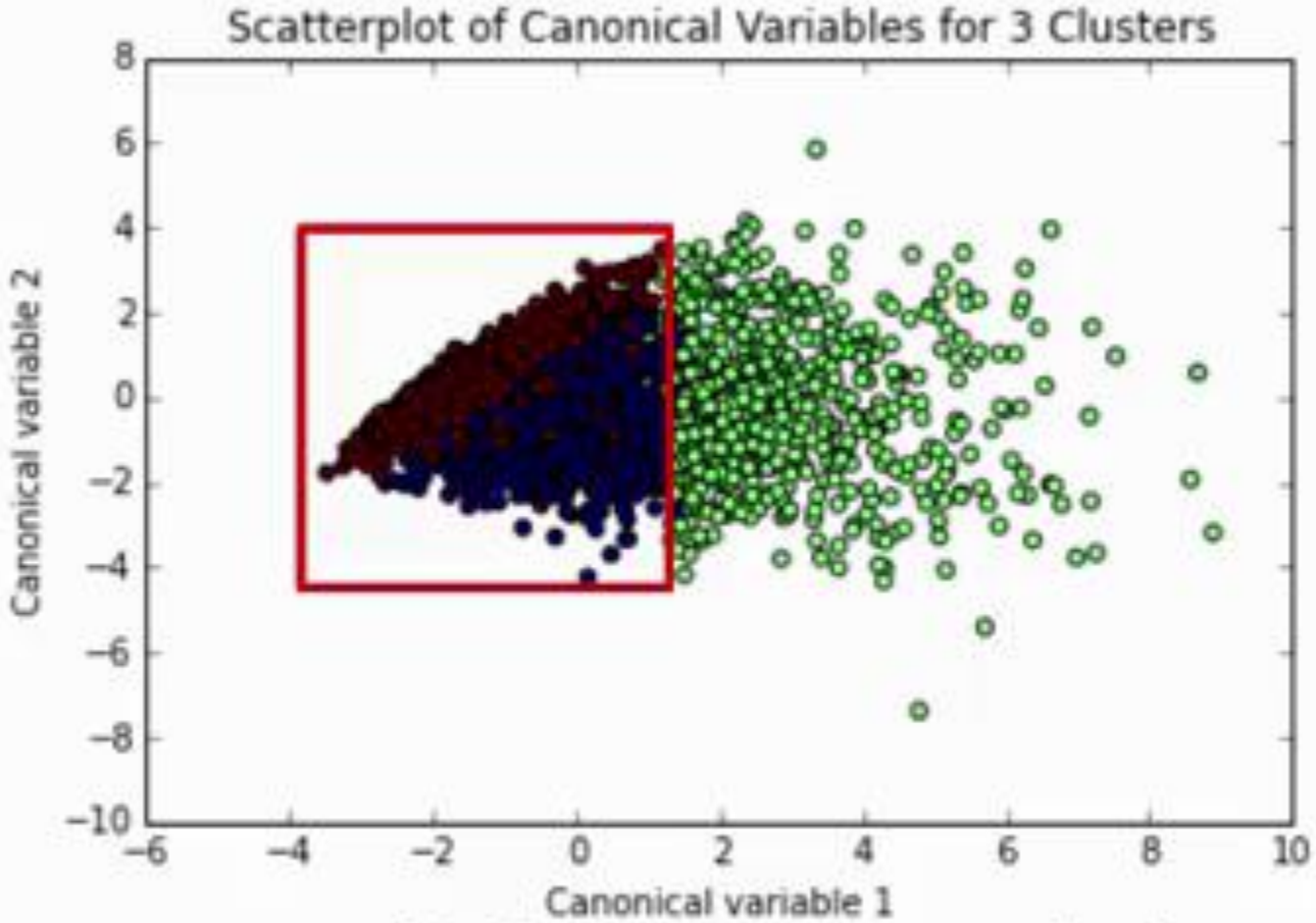
each other, and

In [13]:

within cluster variance is relatively low.

Console 1/A

```
if self._edgecolors == str('face'):
```

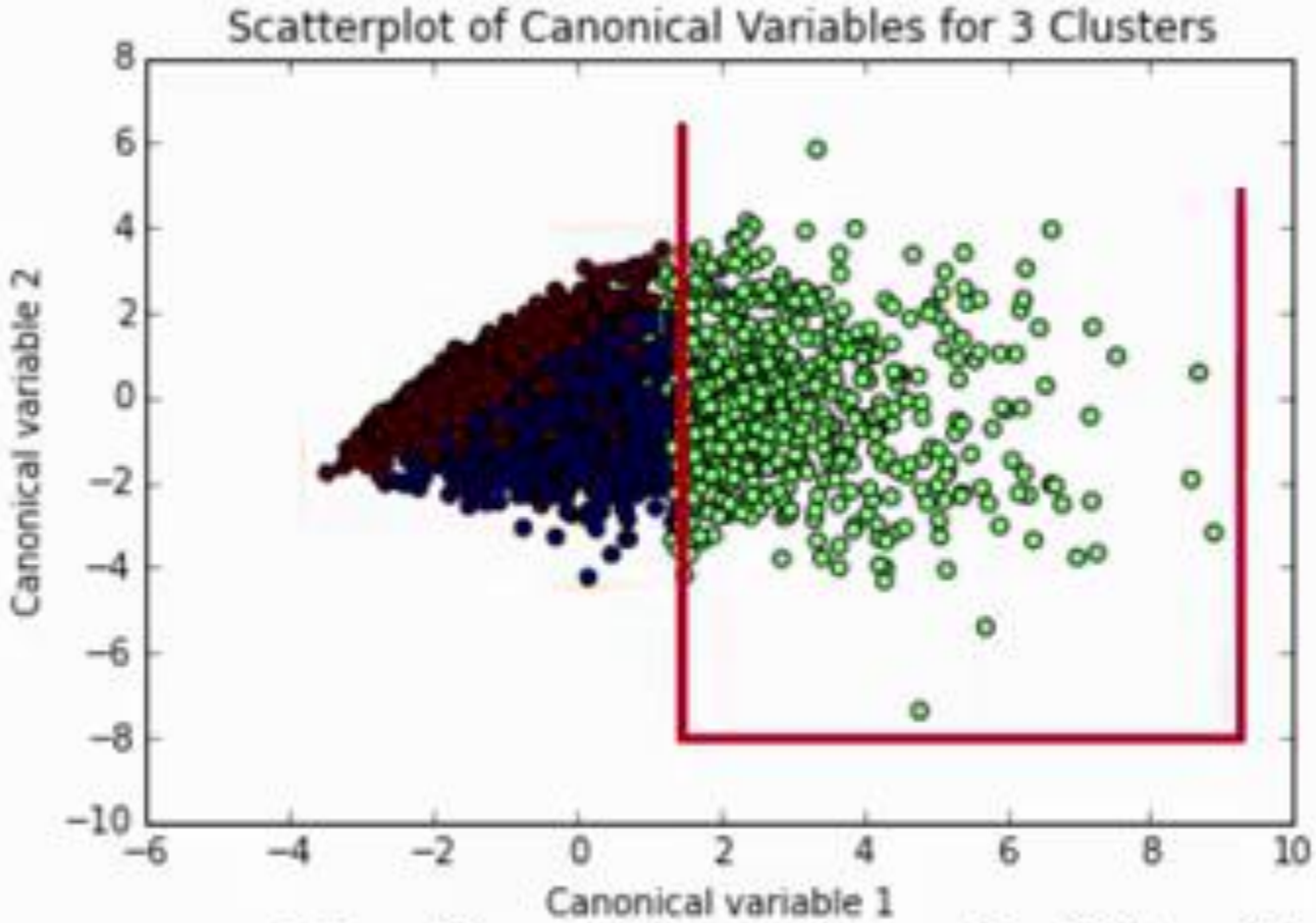


But they appear to have a good deal of overlap,

In [13]:

Console 1/A

```
if self._edgecolors == str('face'):
```

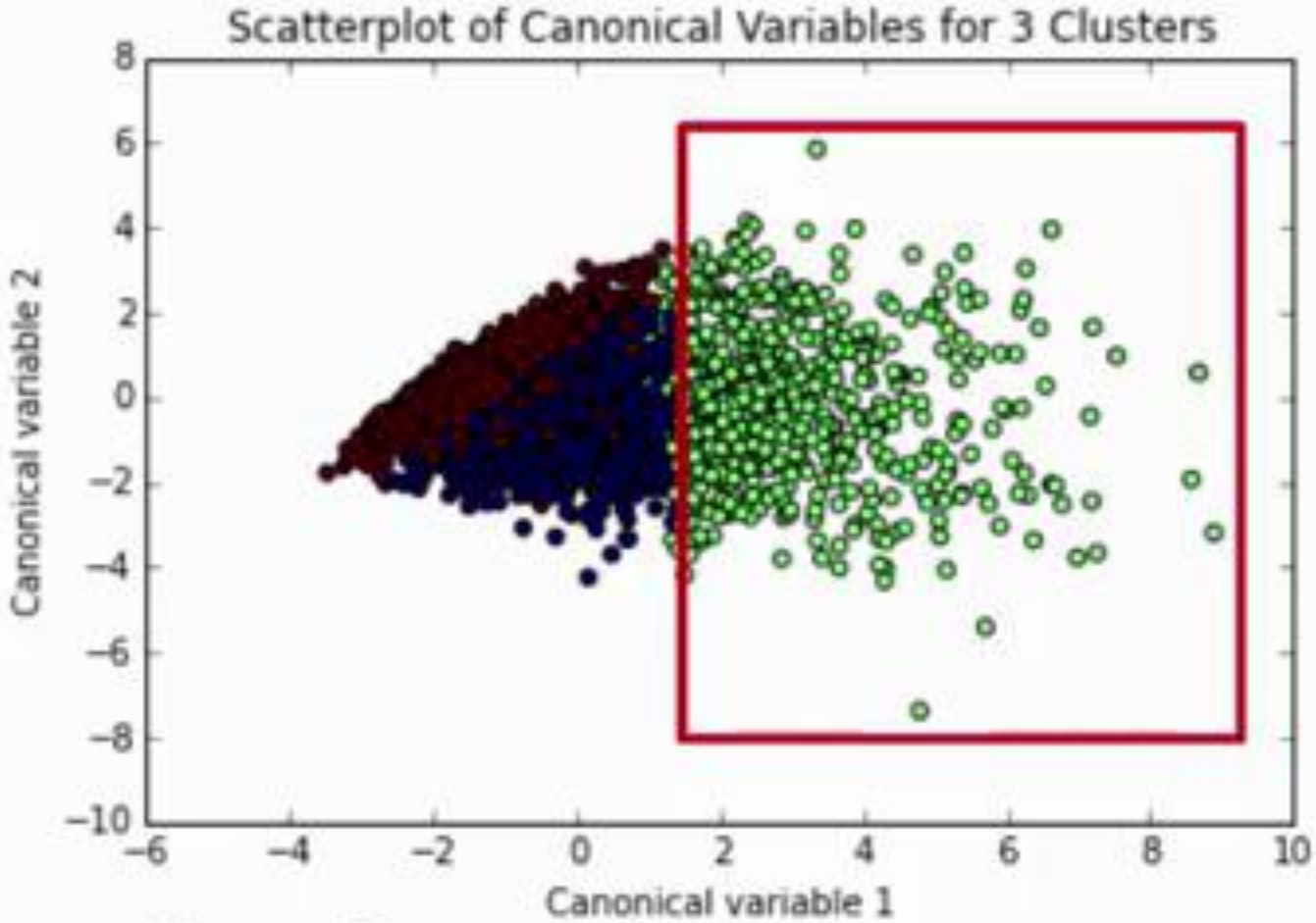


On the other hand, this cluster here shows better separation, but

In [13]:

Console 1/A

```
if self._edgecolors == str('face'):
```

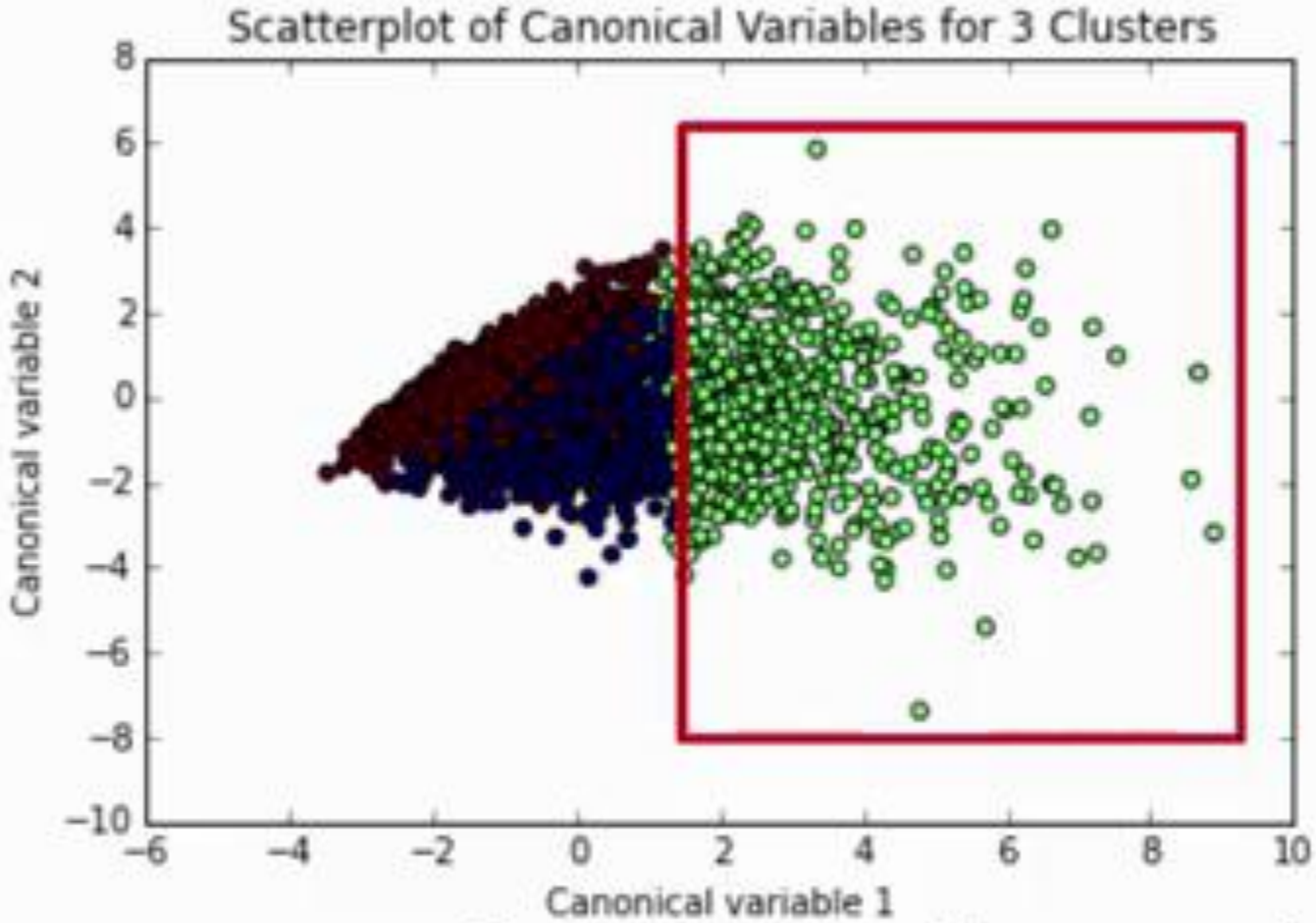


the observations are more spread out

In [13]: indicating less correlation among

Console 1/A

```
if self._edgecolors == str('face'):
```

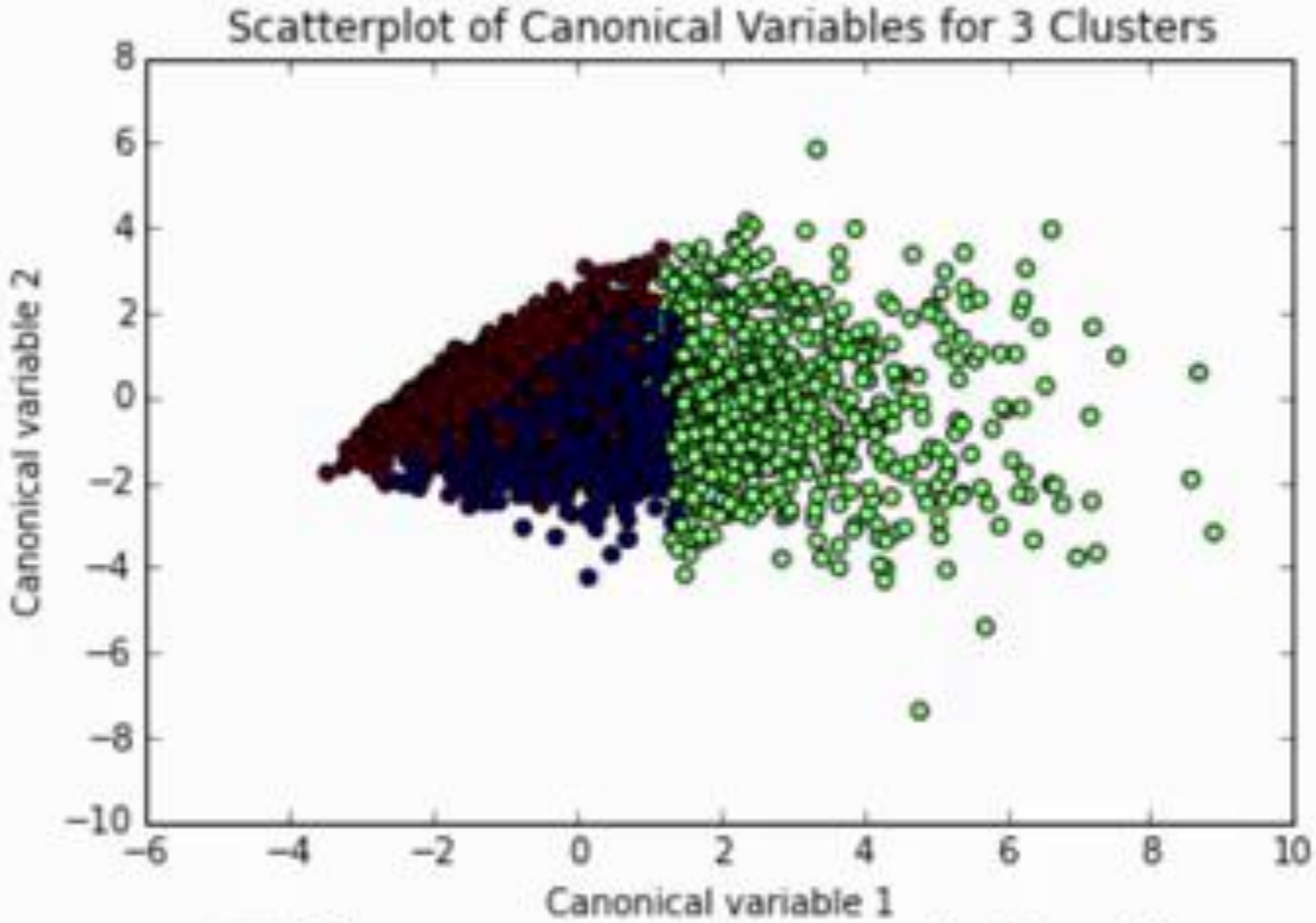


the observations and
higher within cluster variance.

In [13]:

Console 1/A

```
if self._edgecolors == str('face'):
```

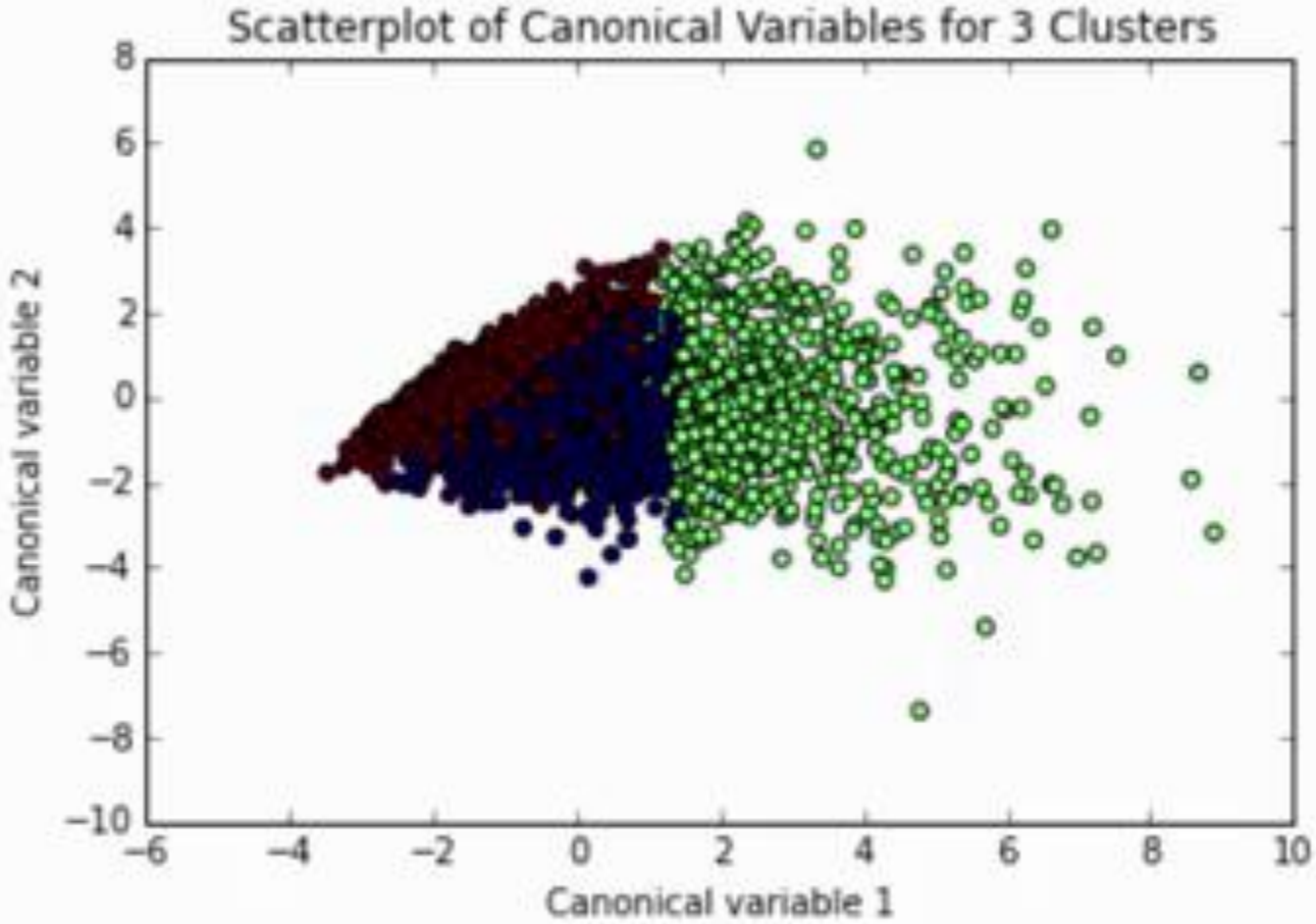


This suggests that the two cluster solution might be better, meaning that it

In [13]:

Console 1/A

```
if self._edgecolors == str('face'):
```



would be especially important to further evaluate the two cluster solution as well.

In [13]:

```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clu
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

Next we can take a look at the pattern of means on the clustering variables for

Usage

Here you can get help o
it, either on the Editor a

Help can also be shown
next to an object. You
Object Inspector.

New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```

Scatterplot of Canonical Variables for 3

```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_tra
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

each cluster to see whether they are distinct and meaningful.

Usage

Here you can get help o
it, either on the Editor a

Help can also be shown
next to an object. You
Object Inspector.

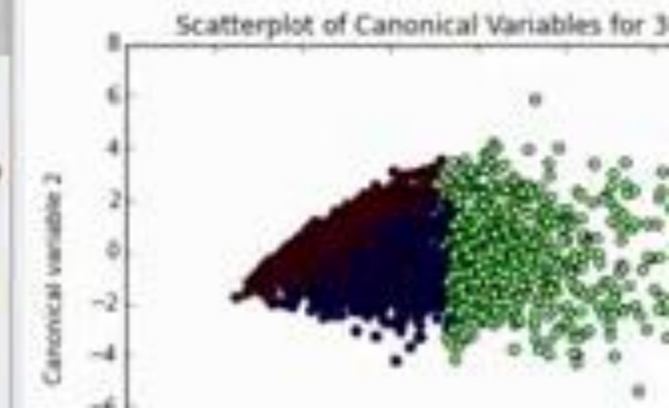
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(cl
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

we have to link the cluster assignment variable back to its corresponding

Usage

Here you can get help o
it, either on the Editor a

Help can also be shown
next to an object. You
Object Inspector.

New to

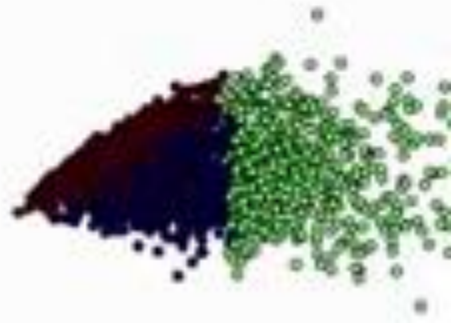
Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```

Scatterplot of Canonical Variables for 3



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

observation in the clus_train dataset that has the clustering variables.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

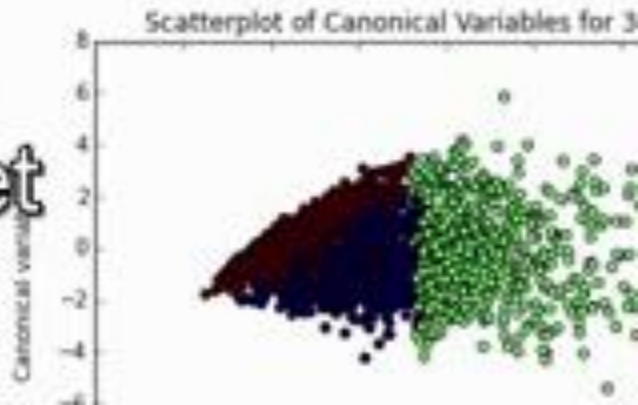
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train, newclus, on='index')
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

We will reset the index with the following code, clus_train.reset_index.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

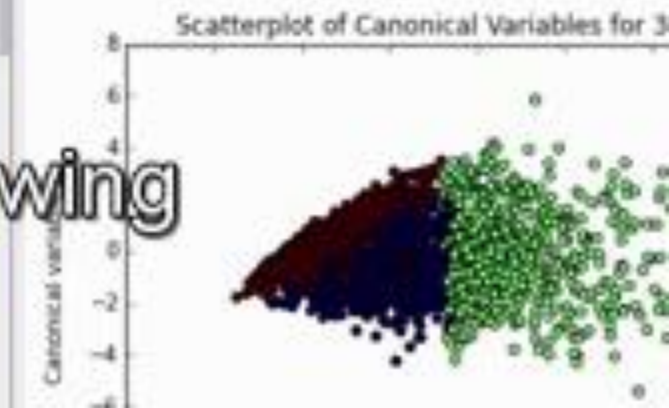
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_t
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

level=0 tells Python to only remove the given levels from the index, and

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

New to

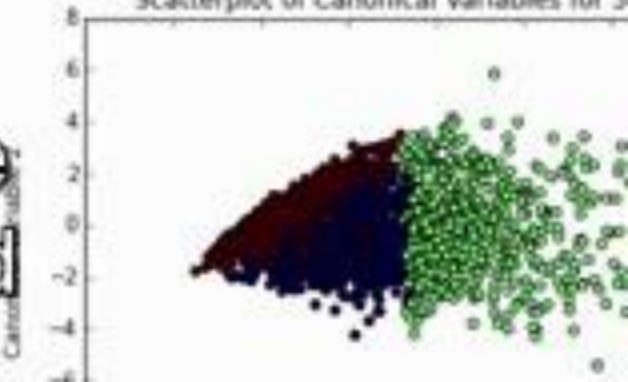
Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```

Scatterplot of Canonical Variables for 3



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train, newclus, on='index')
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

inplace=True, tells Python to add the new column to the existing clus_train dataset.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

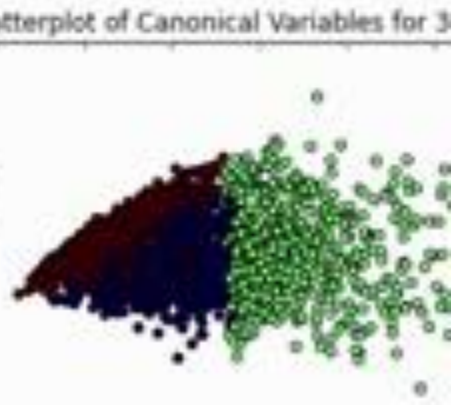
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train, newclu
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

use the list function to pull the new index variable

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

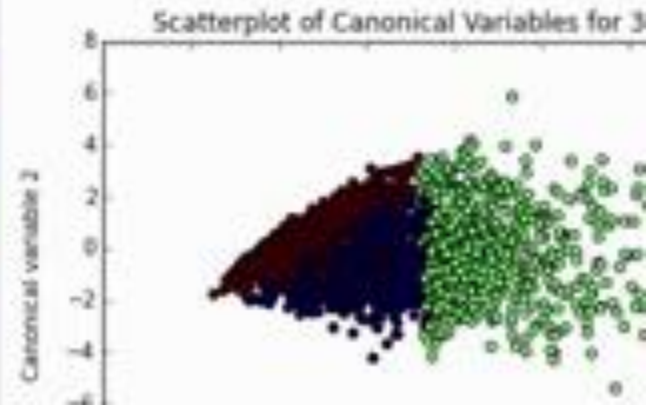
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus
merged_train.head(n=100)
# cluster frequencie
merged_train.cluster.value_counts()

"""
```

This will be combined with the cluster assignment variable, so that we can merge

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

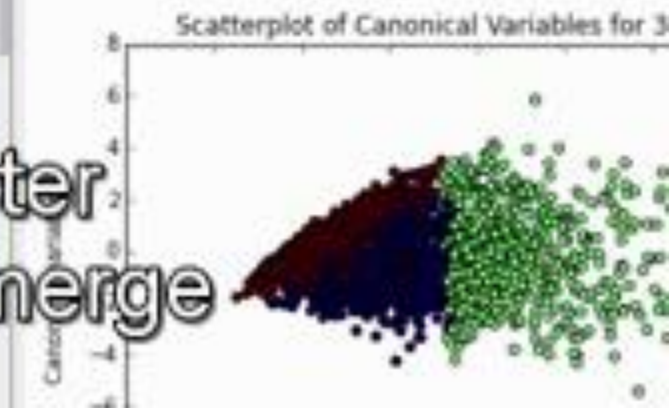
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_t
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

the two datasets together by each observation's unique identifier.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

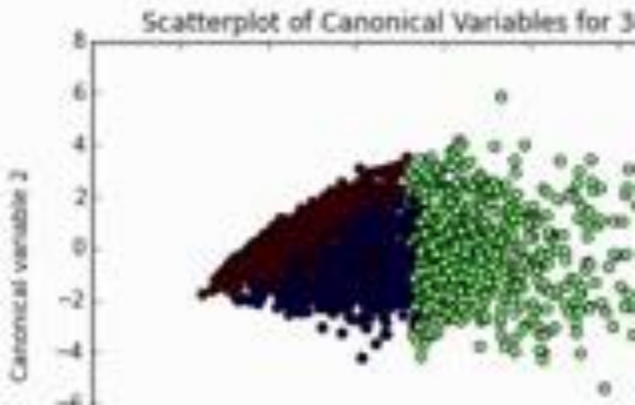
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train, newcl
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

and cluster assignment
together into a single list.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

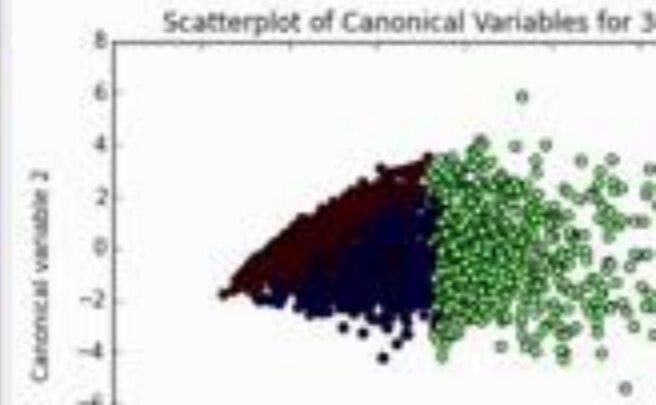
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train, newclus, on='index')
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

We'll call this list newlist using a dict command to create a dictionary.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

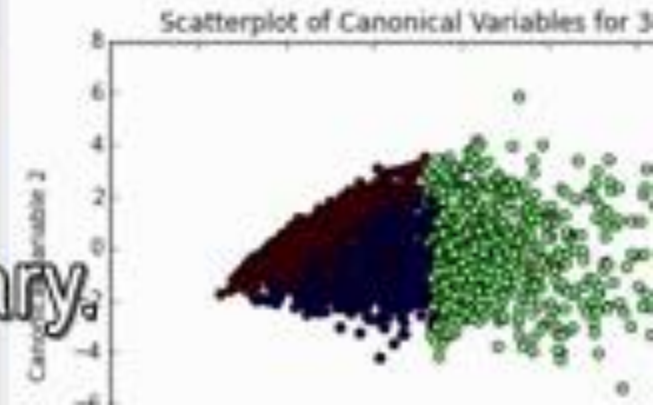
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index vari
merged_train=pd.merg
merged_train.head(n=100)
# cluster frequencie
merged_train.cluster.value_count

"""
```

The dict command creates a dictionary and the zip command is used to combine lists.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

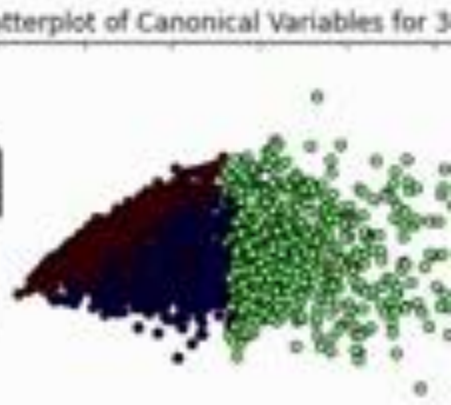
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train,
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.valu

"""
```

So here we type dict(zip) and the two lists that we want to combine.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

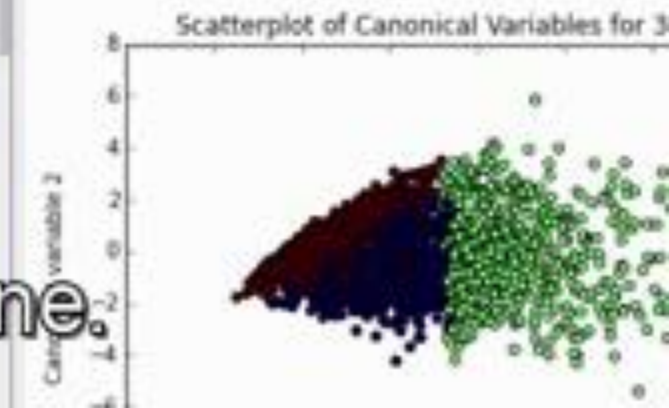
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train, newclus, on='index')
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

Finally, we convert the newlist dictionary to a data frame using

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

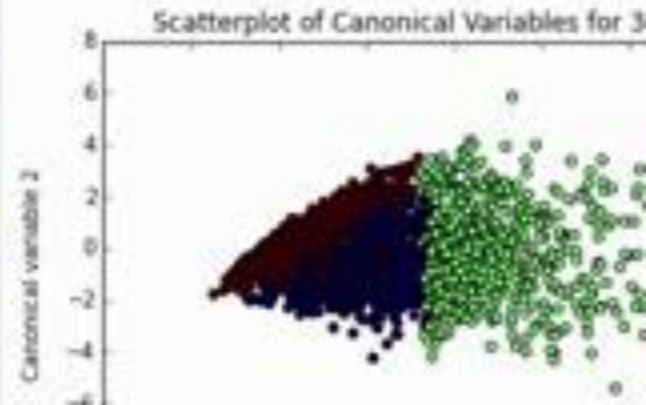
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clu
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

the data frame.from_dict function and rename the cluster assignment

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

New to

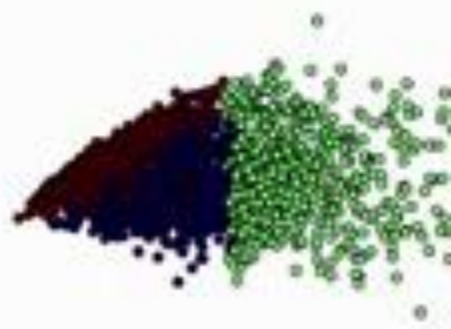
Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```

Scatterplot of Canonical Variables for 3



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train, n
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

variable to cluster by typing newclus.columns=cluster.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

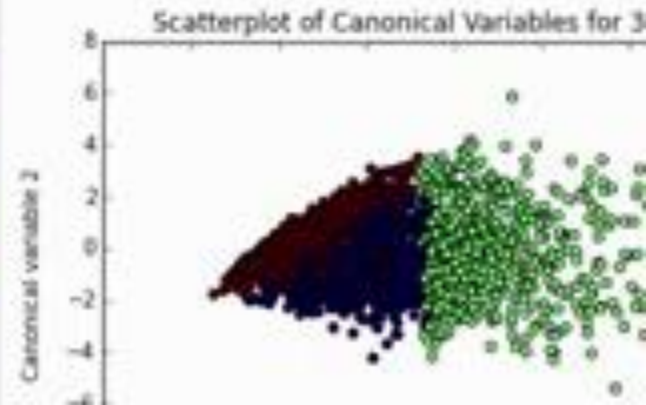
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train, newclus, on='index')
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

the newclus data frame using the model.visa_index function again.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

New to

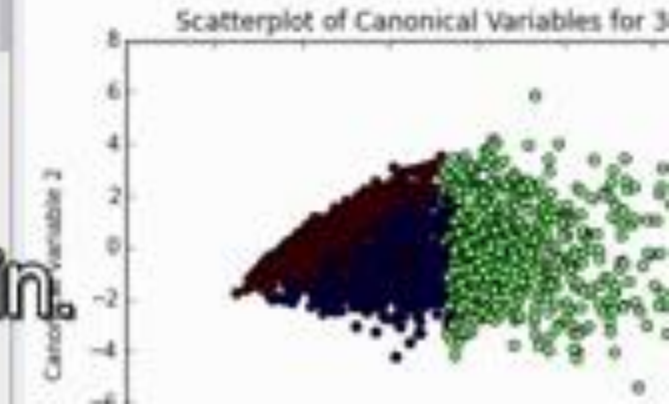
Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```

Scatterplot of Canonical Variables for 3



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train, newclus, on='index')
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

We do this by creating a new data frame, called merged_train, and

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

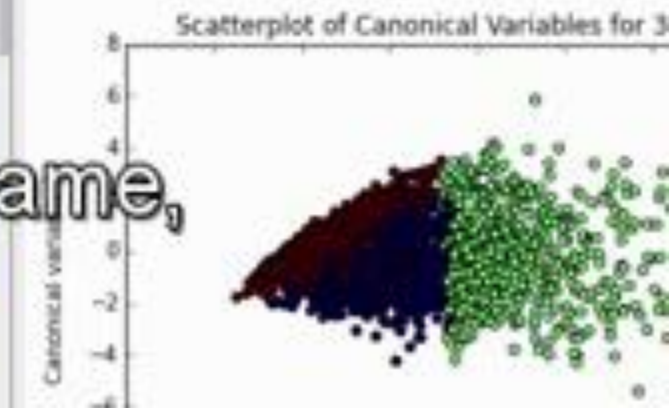
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_t
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

we use the pandas merge function to merge the two data sets.

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

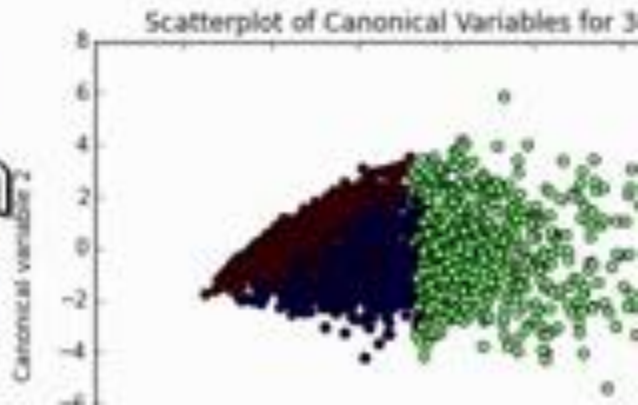
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
plt.show()

"""
BEGIN multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""
# create a unique identifier variable from the index for the
# cluster training data to merge with the cluster assignment variable
clus_train.reset_index(level=0, inplace=True)
# create a list that has the new index variable
cluslist=list(clus_train['index'])
# create a list of cluster assignments
labels=list(model3.labels_)
# combine index variable list with cluster assignment list into a dictionary
newlist=dict(zip(cluslist, labels))
newlist
# convert newlist dictionary to a dataframe
newclus=DataFrame.from_dict(newlist, orient='index')
newclus

# now do the same for the cluster assignment variable
# rename the cluster assignment column
newclus.columns = ['cluster']
# create a unique identifier variable from the index for the
# cluster assignment dataframe
# to merge with cluster training data
newclus.reset_index(level=0, inplace=True)
# merge the cluster assignment dataframe with the cluster training variable dataframe
# by the index variable
merged_train=pd.merge(clus_train, newclus, on='index')
merged_train.head(n=100)
# cluster frequencies
merged_train.cluster.value_counts()

"""
```

On='index' tells Python to merge them

Usage

Here you can get help o
it, either on the Editor o

Help can also be shown
next to an object. You
Object Inspector.

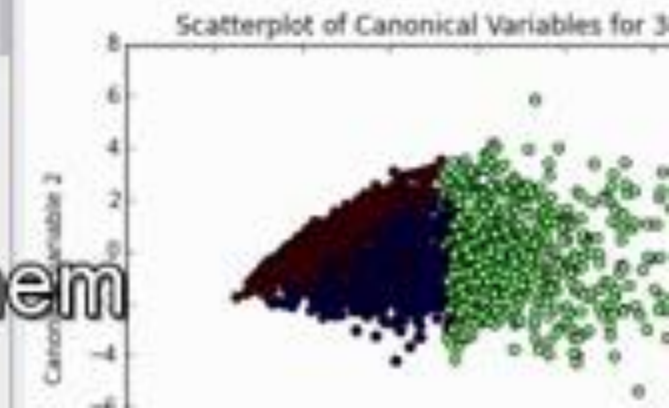
New to

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
if self._edgecolors == str('face
```



```
# cluster frequencies
merged_train.cluster.value_counts()

"""
END multiple steps to merge cluster assignment with clustering variables to e
cluster variable means by cluster
"""

# FINALLY calculate clustering variable means by cluster
clustergrp = merged_train.groupby('cluster').mean()
print ("Clustering variable means by cluster")
print(clustergrp)


# validate clusters in training data by examining cluster differences in GPA
# first have to merge GPA with clustering variables and cluster assignment da
gpa_data=data_cle['GPA1']
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=1
gpa_train1=pd.DataFrame(gpa_train)
```

Finally, we can get the means on the clustering variables for each cluster.

```
# cluster frequencies
merged_train.cluster.value_counts()

"""
END multiple steps to merge cluster assignment with clustering variables to e
cluster variable means by cluster
"""

# FINALLY calculate clustering variable means by cluster
clustergrp = merged_train.groupby('cluster').mean()
print ("Clustering variable means by cluster")
print(clustergrp)


# validate clusters in training data by examining cluster differences in GPA
# first have to merge GPA with clustering variables and cluster assignment da
gpa_data=data_clean['GPA1']
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=1
gpa_train1=pd.DataFrame(gpa_train)
```

all the clustering variables by cluster

IPython console

Console 1/A

```
    ...:
    ...: # FINALLY calculat
    ...: clustergrp = merge
    ...: print ("Clustering
    ...: print(clustergrp)
    ...:
Clustering variable means by cluster
                index   ALCEVR1  MAREVER1  ALCPROBS1       VIOL1  \
cluster
0        3328.930505  0.946562 -0.057109  -0.058550 -0.121192 -0.168252
1        3312.255193  0.661266  1.097891   0.906...   1.106490  0.791967
2        3239.829577 -1.056455 -0.474543  -0.412562 -0.451110 -0.264092

                DEP1   ESTEEM1  SCHCONN1   PARACTV   PARPRES  FAMCONCT
cluster
0          -0.196987  0.188367  0.131356  0.151205  0.099521  0.226842
1           0.853587 -0.655070 -0.927499 -0.409328 -0.473980 -0.955398
2          -0.292352  0.206912  0.341590  0.091216  0.156604  0.298288

In [14]:
```

Remember that the variables are standardized to be on the same scale with an overall sample mean of zero, and a standard deviation of 1

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 119 Column: 55 Memory

IPython console

Console 1/A

```
    ...:
    ...: # FINALLY calculate clustering variable means by cluster
    ...: clustergrp = merged_train.groupby('cluster').mean()
    ...: print ("Clustering variable means by cluster")
    ...: print(clustergrp)
    ...:
Clustering variable means by cluster
                index   ALCEVR1  MAREVER1  ALCPROBS1  DEVIANT1     VIOL1  \
cluster
0        3328.930505  0.946562 -0.057109  -0.058550 -0.121192 -0.168252
1        3312.255193  0.661266  1.097891   0.900592  1.106490  0.791967
2        3239.829577 -1.056455 -0.474543  -0.412562 -0.451110 -0.264092

                DEP1   ESTEEM1  SCHCONN1   PARACTV   PARPRES  FAMCONCT
cluster
0          -0.196987  0.188367  0.131356  0.151205  0.099521  0.226842
1           0.853587 -0.655070 -0.927499 -0.409328 -0.473980 -0.955398
2          -0.292352  0.206912  0.341590  0.091216  0.156604  0.298288

In [14]:
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 119 Column: 55 Memory

IPython console

Console 1/A

```
    ...:
    ...: # FINALLY calculate clustering variable means by cluster
    ...: clustergrp = merged_train.groupby('cluster').mean()
    ...: print ("Clustering variable means by cluster")
    ...: print(clustergrp)
    ...:
Clustering variable means by cluster
                index   ALCEVR1  MAREVER1  ALCPROBS1  DEVIANT1     VIOL1  \
cluster
0        3328.930505  0.946562 -0.057109  -0.058550 -0.121192 -0.168252
1        3312.255193  0.661266  1.097891   0.900592  1.106490  0.791967
2        3239.829577 -1.056455 -0.474543  -0.412562 -0.451110 -0.264092

                DEP1   ESTEEM1  SCHCONN1   PARACTV   PARPRES  FAMCONCT
cluster
0         -0.196987  0.188367  0.131356  0.151205  0.099521  0.226842
1          0.853587 -0.655070 -0.927499 -0.409328 -0.473980 -0.955398
2         -0.292352  0.206912  0.341590  0.091216  0.156604  0.298288

In [14]:
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 119 Column: 55 Memory

```
"""
END multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""

# FINALLY calculate clustering variable means by cluster
clustergrp = merged_train.groupby('cluster').mean()
print ("Clustering variable means by cluster")
print(clustergrp)


# validate clusters in training data by examining cluster differences in GPA using ANOVA
# first have to merge GPA with clustering variables and cluster assignment data
gpa_data=data_clean['GPA1']
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
```

Finally, let's see how the clusters differ on GPA.

Object inspector

IPython console

Console 1/

packages\pa
return

File "C:\
packages\pa
code, i

File "C:\

```
"""
END multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""

# FINALLY calculate clustering variable means by cluster
clustergrp = merged_train.groupby('cluster').mean()
print ("Clustering variable means by cluster")
print(clustergrp)


# validate clusters in training data by examining cluster differences in GPA using ANOVA
# first have to merge GPA with clustering variables and cluster assignment data
gpa_data=data_clean['GPA1']
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
```

We first have to extract the GPA variable from the original data set,

```
"""
END multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""

# FINALLY calculate clustering variable means by cluster
clustergrp = merged_train.groupby('cluster').mean()
print ("Clustering variable means by cluster")
print(clustergrp)


# validate clusters in training data by examining cluster differences in GPA using ANOVA
# first have to merge GPA with clustering variables and cluster assignment data
gpa_data=data_clean['GPA1']
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
```

Before merging, we need to reset the index for

```
"""
END multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""

# FINALLY calculate clustering variable means by cluster
clustergrp = merged_train.groupby('cluster').mean()
print ("Clustering variable means by cluster")
print(clustergrp)


# validate clusters in training data by examining cluster differences in GPA using ANOVA
# first have to merge GPA with clustering variables and cluster assignment data
gpa_data=data_clean['GPA1']
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary

print ('means for GPA by cluster')
```

the GPA variable data frame to create the unique identifier to link the datasets.

Object inspector

IPython console

Console 1/

packages\pa

return

File "C:\

packages\pa

code, i

File "C:\

```
"""
END multiple steps to merge cluster assignment with clustering variables to examine
cluster variable means by cluster
"""

# FINALLY calculate clustering variable means by cluster
clustergrp = merged_train.groupby('cluster').mean()
print ("Clustering variable means by cluster")
print(clustergrp)


# validate clusters in training data by examining cluster differences in GPA using ANOVA
# first have to merge GPA with clustering variables and cluster assignment data
gpa_data=data_clean['GPA1']
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
```

We then merge the datasets by the unique identifier index into a data frame

```
# FINALLY calculate clustering variable means by cluster
clustergrp = merged_train.groupby('cluster').mean()
print ("Clustering variable means by cluster")
print(clustergrp)


# validate clusters in training data by examining cluster differences in GPA using ANOVA
# first have to merge GPA with clustering variables and cluster assignment data
gpa_data=data_clean['GPA1']
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
m1= sub1.groupby('cluster').mean()
print (m1)

print ('standard deviations for GPA by cluster')
```

We'll use analysis of variance to test whether there are significant differences

```
gpa_data=data_clean['GPA1']
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
m1= sub1.groupby('cluster').mean()
print (m1)

print ('standard deviations for GPA by cluster')
m2= sub1.groupby('cluster').std()
print (m2)

mc1 = multi.MultiComparison(sub1
res1 = mc1.tukeyhsd()
print(res1.summary())
```

between clusters on
the quantitative GPA variable.

```
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
m1= sub1.groupby('cluster').mean()
print (m1)

print ('standard deviations for GPA by cluster')
m2= sub1.groupby('cluster').std()
print (m2)

mc1 = multi.MultiComparison(sub1['GPA1'], sub1['cluster'])
res1 = mc1.tukeyhsd()
print(res1.summary())
```

We use the ols function to test the analysis of variance.

```
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
m1= sub1.groupby('cluster').mean()
print (m1)

print ('standard deviations for GPA by cluster')
m2= sub1.groupby('cluster').std()
print (m2)

mc1 = multi.MultiComparison(sub1['GPA1'], sub1['cluster'])
res1 = mc1.tukeyhsd()
print(res1.summary())
```

The formula specifies the model, with GPA as the response variable and

```
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
m1= sub1.groupby('cluster').mean()
print (m1)

print ('standard deviations for GPA by cluster')
m2= sub1.groupby('cluster').std()
print (m2)

mc1 = multi.MultiComparison(sub1['GPA1'], sub1['cluster'])
res1 = mc1.tukeyhsd()
print(res1.summary())
```

The capital C tells Python that the cluster assignment variable is

```
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
m1= sub1.groupby('cluster').mean()
print (m1)

print ('standard deviations for GPA by cluster')
m2= sub1.groupby('cluster').std()
print (m2)

mc1 = multi.MultiComparison(sub1['GPA1'], sub1['cluster'])
res1 = mc1.tukeyhsd()
print(res1.summary())
```

categorical.

```
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
m1= sub1.groupby('cluster').mean()
print (m1)

print ('standard deviations for GPA by cluster')
m2= sub1.groupby('cluster').std()
print (m2)

mc1 = multi.MultiComparison(sub1['GPA1'], sub1['cluster'])
res1 = mc1.tukeyhsd()
print(res1.summary())
```

we will request a tukey test to evaluate post hot comparisons

```
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
m1= sub1.groupby('cluster').mean()
print (m1)

print ('standard deviations for GPA by cluster')
m2= sub1.groupby('cluster').std()
print (m2)

mc1 = multi.MultiComparison(sub1['GPA1'], sub1['cluster'])
res1 = mc1.tukeyhsd()
print(res1.summary())
```

```
# split GPA data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1=pd.DataFrame(gpa_train)
gpa_train1.reset_index(level=0, inplace=True)
merged_train_all=pd.merge(gpa_train1, merged_train, on='index')
sub1 = merged_train_all[['GPA1', 'cluster']].dropna()

import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula='GPA1 ~ C(cluster)', data=sub1).fit()
print (gpamod.summary())

print ('means for GPA by cluster')
m1= sub1.groupby('cluster').mean()
print (m1)

print ('standard deviations for GPA by cluster')
m2= sub1.groupby('cluster').std()
print (m2)

mc1 = multi.MultiComparison(sub1['GPA1'], sub1['cluster'])
res1 = mc1.tukeyhsd()
print(res1.summary())
```

between the clusters using
the multi comparison function from

```
nces in GPA using ANOVA
ssignment data

ndom_state=123)
```

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   GPA1   R-squared:                       0.077
Model:                            OLS   Adj. R-squared:                  0.076
Method:                 Least Squares   F-statistic:                     133.3
Date:                Fri, 22 Jan 2016   Prob (F-statistic):           2.50e-56
Time:                        14:24:27   Log-Likelihood:                -3599.3
No. Observations:                3202   AIC:                             7205.
Df Residuals:                    3199   BIC:                             7223.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
=
                 coef    std err          t      P>|t|      [95.0% Conf.
------------------------------------------------------------------------------
```

The analysis of variance summary table

```
ces in GPA using ANOVA
ssignment data


ndom_state=123)
```

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   GPA1   R-squared:                       0.077
Model:                            OLS   Adj. R-squared:                  0.076
Method:                 Least Squares   F-statistic:                     133.3
Date:                Fri, 22 Jan 2016   Prob (F-statistic):           2.50e-56
Time:                        14:24:27   Log-Likelihood:                -3599.3
No. Observations:                3202   AIC:                             7205.
Df Residuals:                    3199   BIC:                             7223.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf.
------------------------------------------------------------------------------
```

indicates that the clusters differed significantly on GPA.

```
nces in GPA using ANOVA
ssignment data


ndom_state=123)
```

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
means for GPA by cluster
              GPA1
cluster
0         2.829949
1         2.426063
2         2.994542
standard deviations for GPA by cluster
              GPA1
cluster
0         0.727230
1         0.738174
==========================================
```

adolescents in cluster 1, the most troubled group, had the lowest GPA,

```
nces in GPA using ANOVA
ssignment data


ndom_state=123)
```

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
means for GPA by cluster
              GPA1
cluster
0         2.829949
1         2.426063
2         2.994542
standard deviations for GPA by cluster
              GPA1
cluster
0         0.727230
2         0.738174
==========================================
```

and adolescents in cluster 2, the least troubled group, had the highest GPA.

```
ter differences in GPA using ANOVA
d cluster assignment data

_size=.3, random_state=123)

='index')

1).fit()

r'])
```

```
Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

means for GPA by cluster
             GPA1
cluster
0        2.829949
1        2.426063
2        2.994542
standard deviations for GPA by cluster
             GPA1
cluster
0        0.727230
1        0.786903
2        0.728174

Multiple Comparison of Means - Tukey HSD,FWER=0.05
==================================================
group1 group2 meandiff  lower   upper  reject
--------------------------------------------------
0      2      0.1646   0.0946  0.2346   True
```

The tukey test shows that the clusters differed significantly in mean GPA,

```
ter differences in GPA using ANOVA
d cluster assignment data

_size=.3, random_state=123)

='index')

1).fit()

r'])
```

Object inspector | Variable explorer | File explorer

IPython console

Console 1/A

```
means for GPA by cluster
              GPA1
cluster
0        2.829949
1        2.426063
2        2.994542
standard deviations for GPA by cluster
              GPA1
cluster
0        0.727230
1        0.786903
2        0.728174
Multiple Comparison of Means - Tukey HSD,FWER=0.05
=============================================
group1 group2 meandiff  lower   upper  reject
---------------------------------------------
  0      1    -0.4039  -0.4892 -0.3186  True
  0      2     0.1646   0.0946  0.2346  True
  1      2     0.5685   0.4868  0.6502  True
---------------------------------------------
```

```
ster differences in GPA using ANOVA
d cluster assignment data


_size=.3, random_state=123)


='index')


1).fit()


r'])
```

Object inspector | Variable explorer | File explorer

IPython console

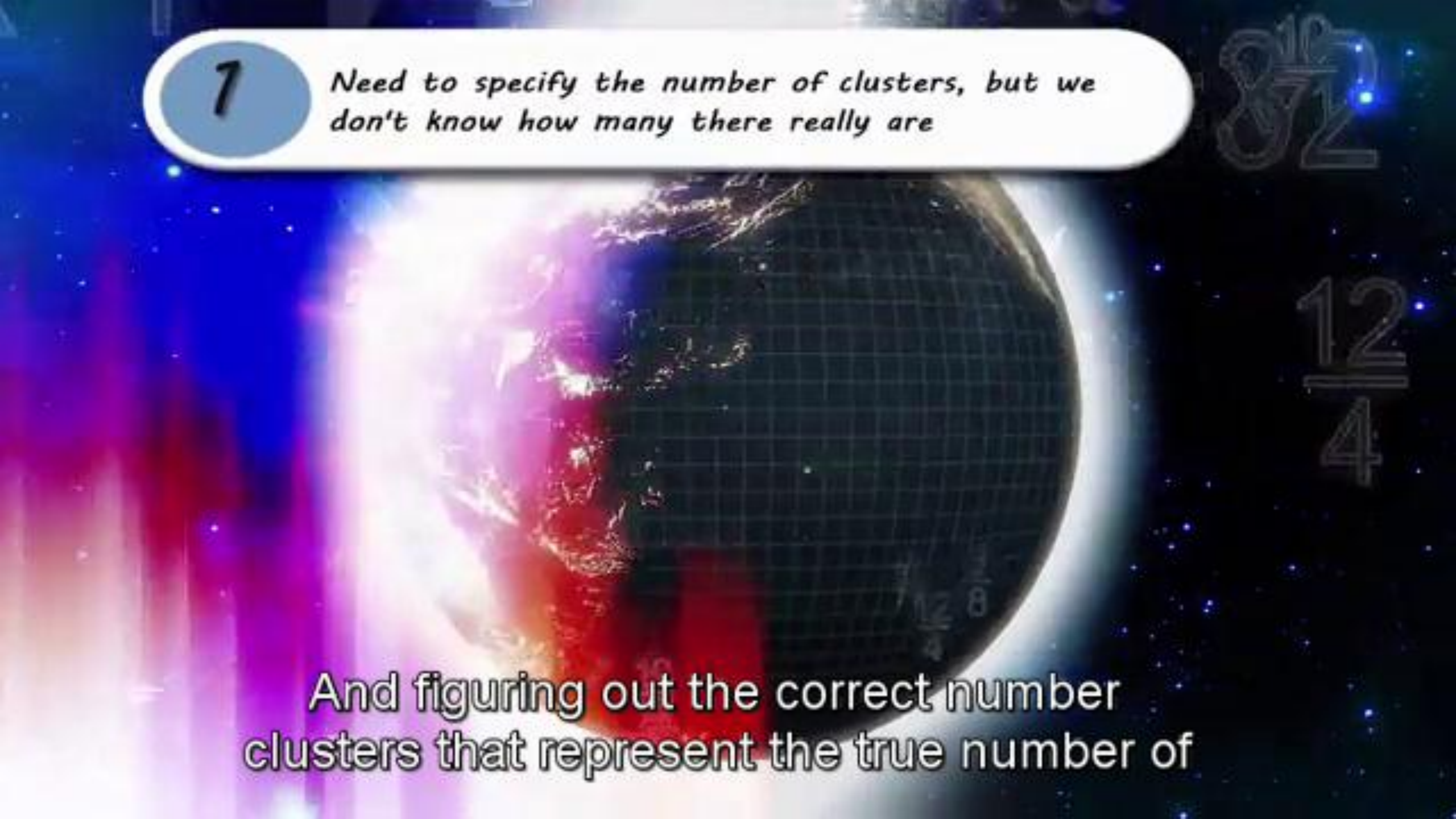
Console 1/A

```
means for GPA by cluster
              GPA1
cluster
0         2.829949
1         2.426063
2         2.994542
standard deviations for GPA by cluster
              GPA1
cluster
0         0.727230
1         0.786903
2         0.738174
Multiple Comparison of Means - Tukey HSD,FWER=0.05
=============================================
group1 group2 meandiff  lower   upper  reject
---------------------------------------------
1      2      0.5685   0.4868  0.6502  True
```

although the difference between cluster 0 and cluster 2 were smaller.

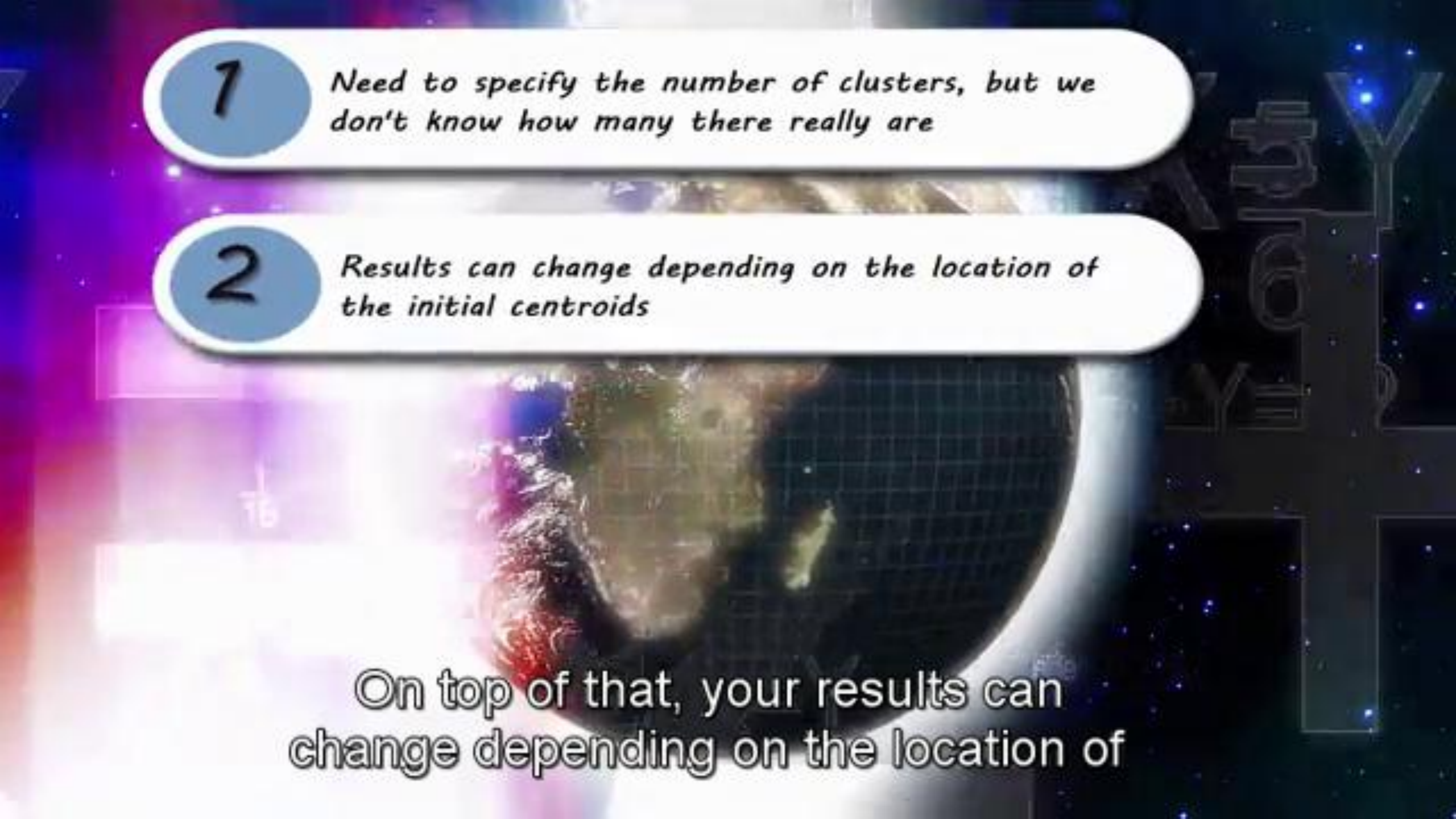
Module 4
Lesson 4 - k-Means Cluster Analysis Limitations


1
Need to specify the number of clusters, but we don't know how many there really are
But we don't know the true number of clusters.


1
Need to specify the number of clusters, but we don't know how many there really are
And figuring out the correct number
clusters that represent the true number of

1 Need to specify the number of clusters, but we don't know how many there really are

clusters in the population
is pretty subjective.


1
Need to specify the number of clusters, but we don't know how many there really are
2
Results can change depending on the location of the initial centroids
On top of that, your results can change depending on the location of

1. Need to specify the number of clusters, but we don't know how many there really are

2. Results can change depending on the location of the initial centroids

3. K-means cluster analysis isn't recommended if you have a lot of categorical variables

K-means cluster analysis
is not recommended if

1. Need to specify the number of clusters, but we don't know how many there really are

2. Results can change depending on the location of the initial centroids

3. K-means cluster analysis isn't recommended if you have a lot of categorical variables

If you have a lot of categorical variables, then you need to use

1. Need to specify the number of clusters, but we don't know how many there really are

2. Results can change depending on the location of the initial centroids

3. K-means cluster analysis isn't recommended if you have a lot of categorical variables

4. Assumes that clusters are spherical, distinct, and approximately equal in size

1. Need to specify the number of clusters, but we don't know how many there really are

2. Results can change depending on the location of the initial centroids

3. K-means cluster analysis isn't recommended if you have a lot of categorical variables

4. Assumes that clusters are spherical, distinct, and ap[illegible]

As a result, tends to identify clusters with these characteristics.

1. Need to specify the number of clusters, but we don't know how many there really are

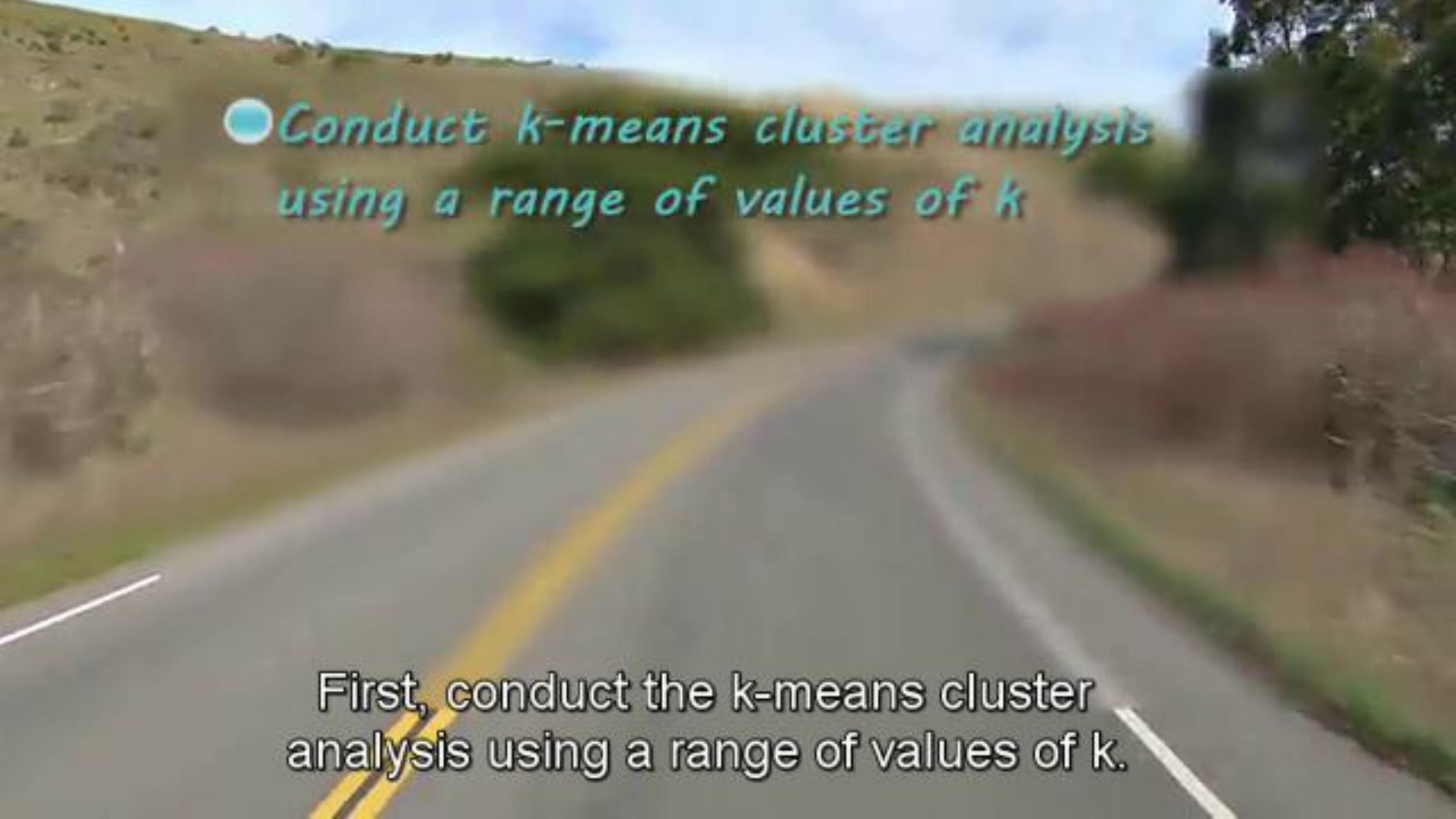
2. Results can change depending on the location of the initial centroids

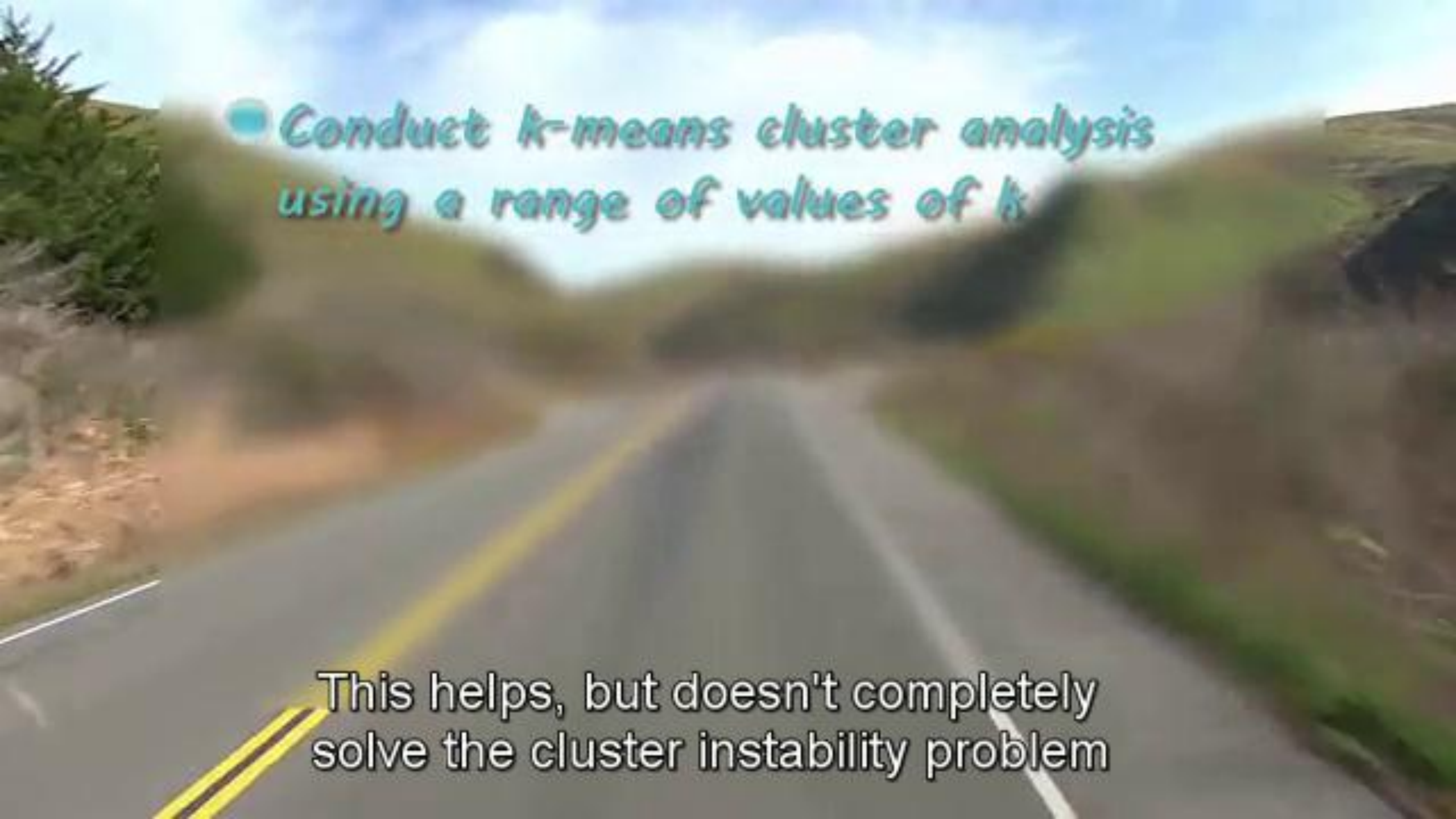
3. K-means cluster analysis isn't recommended if you have a lot of categorical variables

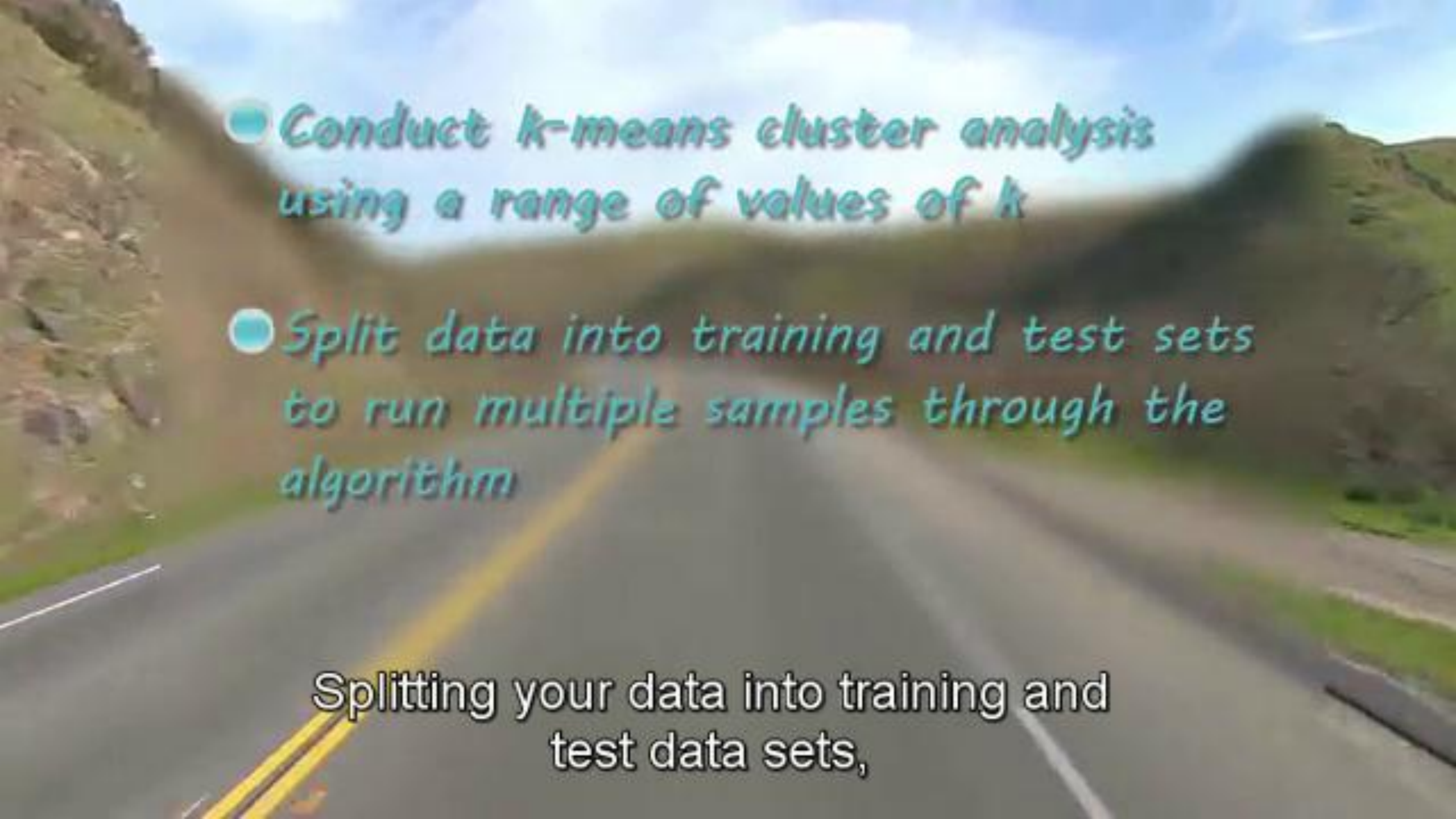
4. Assumes that clusters are spherical, distinct, and all, so, it won't work as well if clusters are elongated or not equal in size.

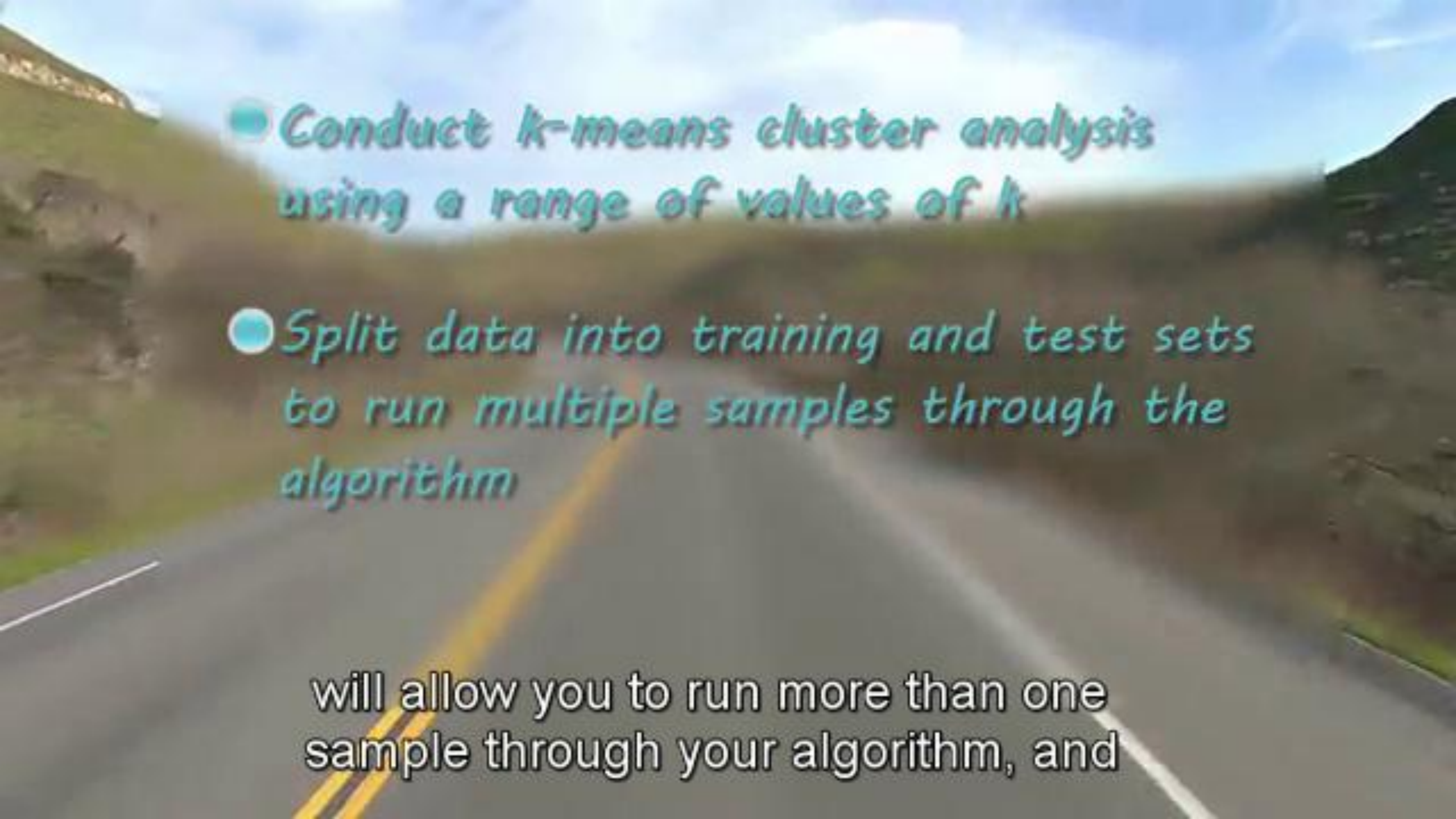

There are a few steps you can take to
help you feel more confident about


the reliability and
validity of your clusters.


Conduct k-means cluster analysis using a range of values of k
First, conduct the k-means cluster analysis using a range of values of k.


Conduct k-means cluster analysis using a range of values of k
This helps, but doesn't completely solve the cluster instability problem


Conduct k-means cluster analysis using a range of values of k
Split data into training and test sets to run multiple samples through the algorithm
Splitting your data into training and test data sets,

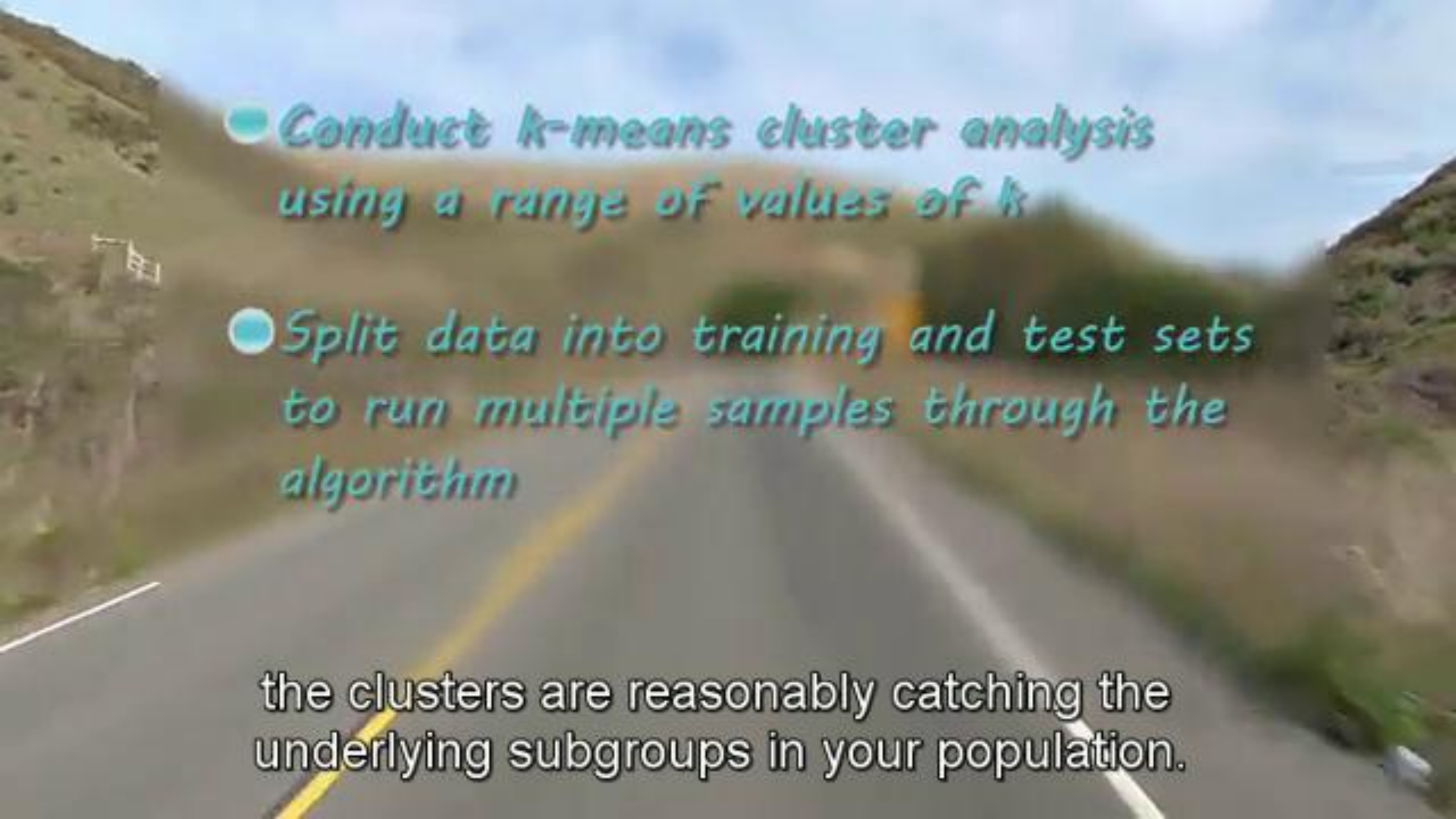

Conduct k-means cluster analysis using a range of values of k
Split data into training and test sets to run multiple samples through the algorithm
will allow you to run more than one sample through your algorithm, and

- Conduct k-means cluster analysis using a range of values of k
- Split data into training and test sets to run multiple samples through the algorithm

can be helpful in determining whether
the clusters you find are reliable.

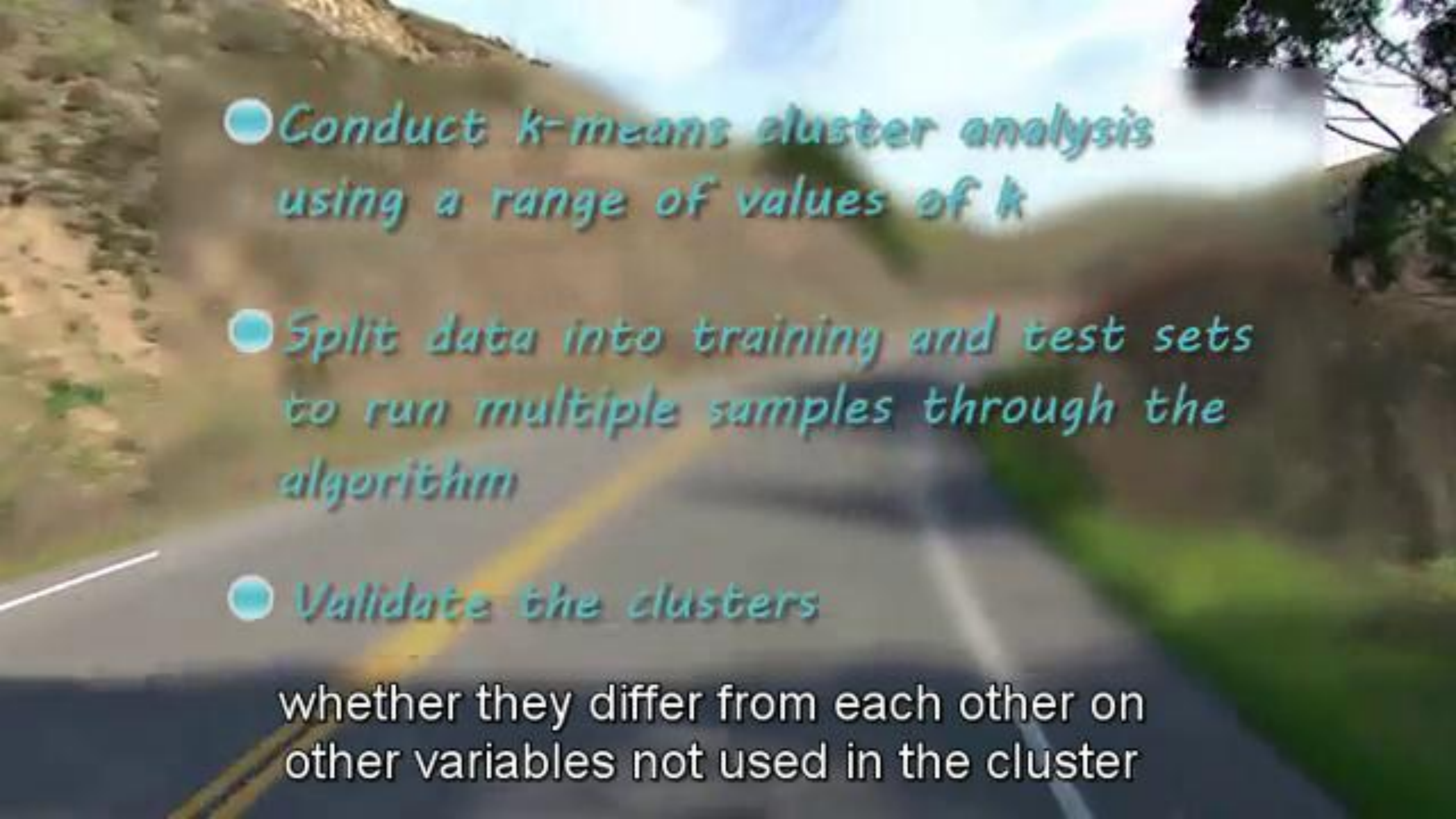
- Conduct k-means cluster analysis using a range of values of k
- Split data into training and test sets to run multiple samples through the algorithm

If you get the same results in different samples, you can be more confident that


Conduct k-means cluster analysis using a range of values of k
Split data into training and test sets to run multiple samples through the algorithm
the clusters are reasonably catching the underlying subgroups in your population.

- Conduct k-means cluster analysis using a range of values of k
- Split data into training and test sets to run multiple samples through the algorithm
- Validate the clusters

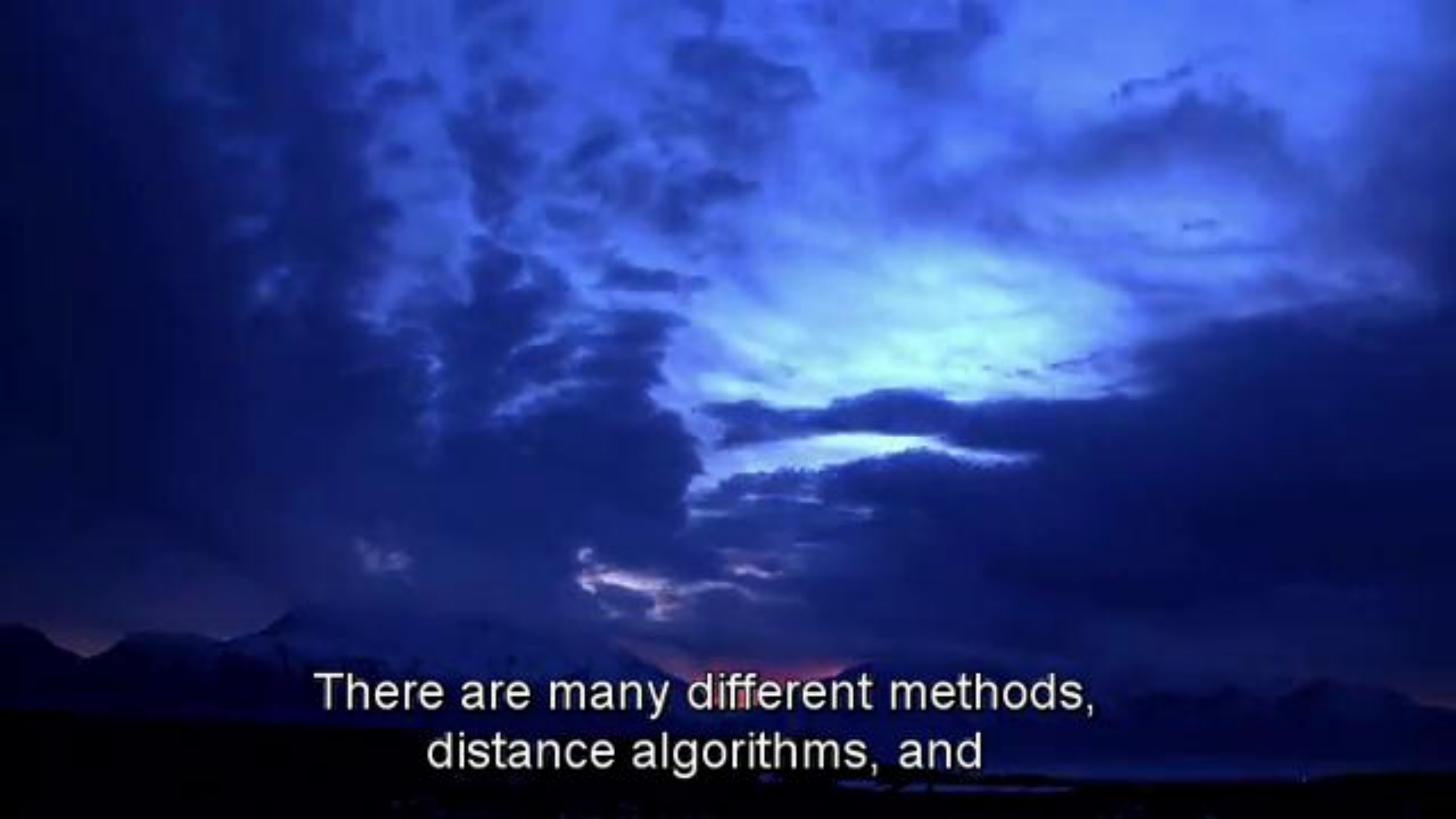
validating the clusters by determining whether they are interpretable, and

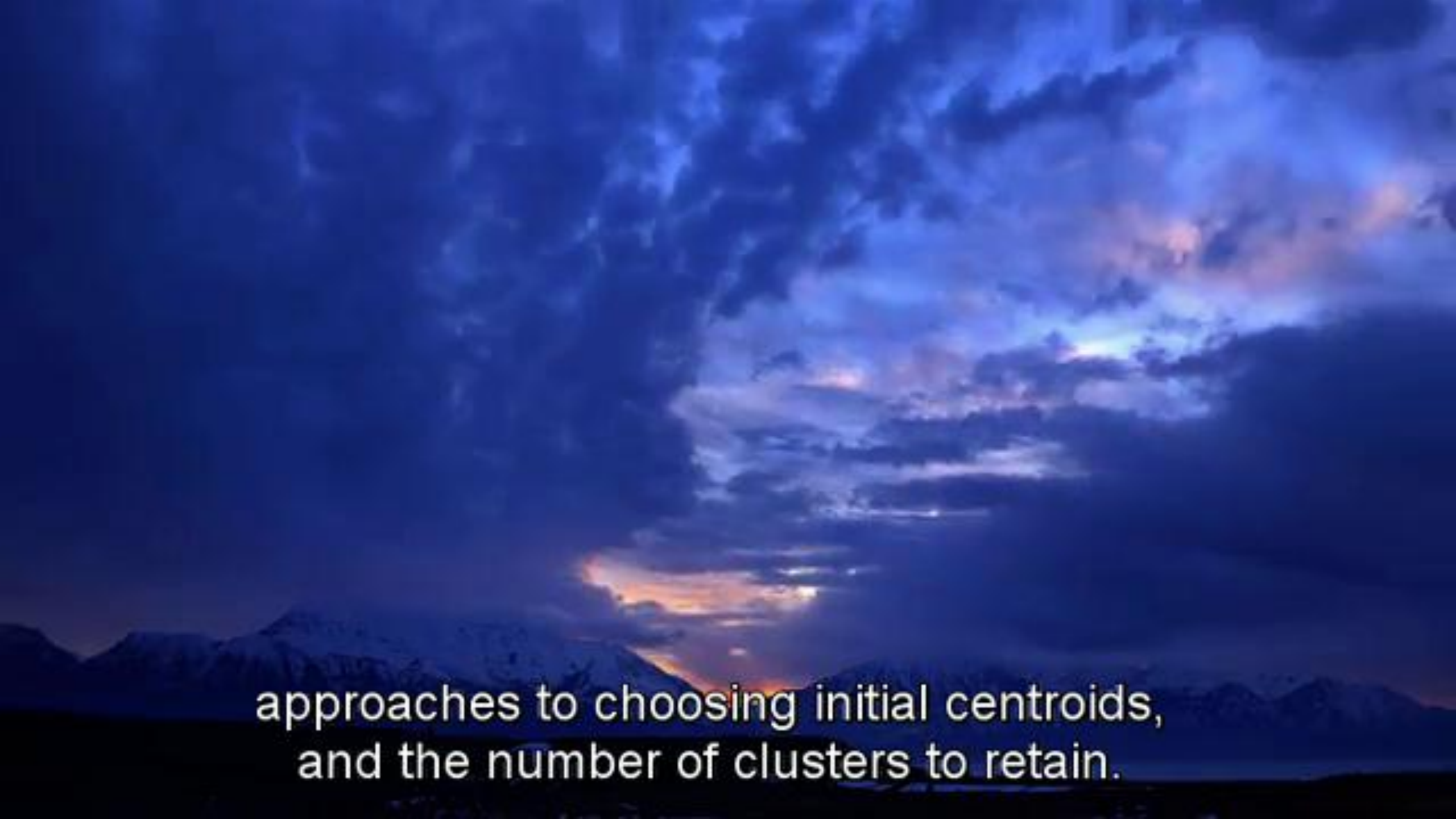

Conduct k-means cluster analysis using a range of values of k
Split data into training and test sets to run multiple samples through the algorithm
Validate the clusters
whether they differ from each other on other variables not used in the cluster

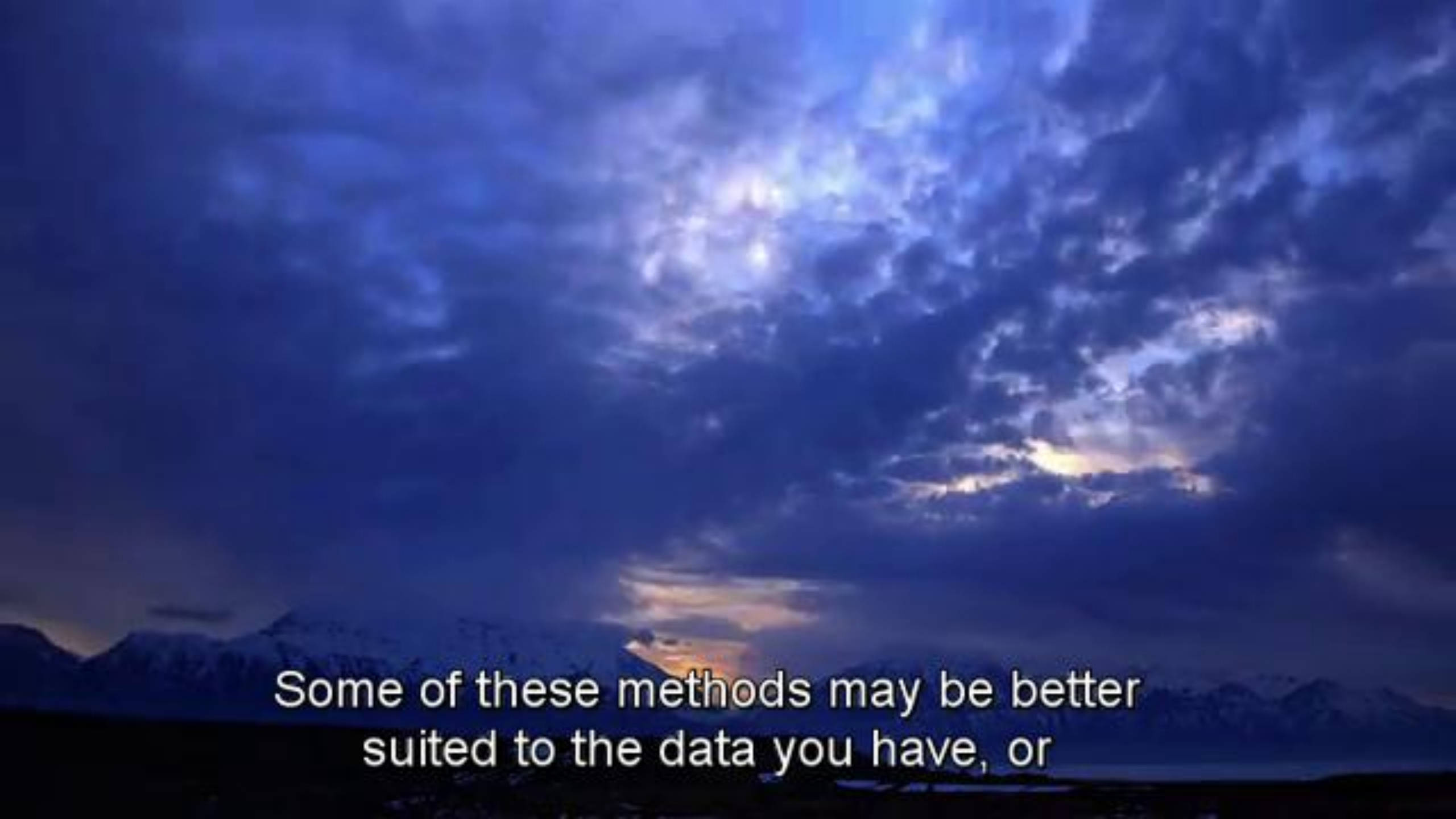
- Conduct k-means cluster analysis using a range of values of k
- Split data into training and test sets to run multiple samples through the algorithm
- Validate the clusters

analysis, can increase your confidence in the cluster solution that you choose.


In this course, we've just scratched
the surface of cluster analysis.


There are many different methods,
distance algorithms, and

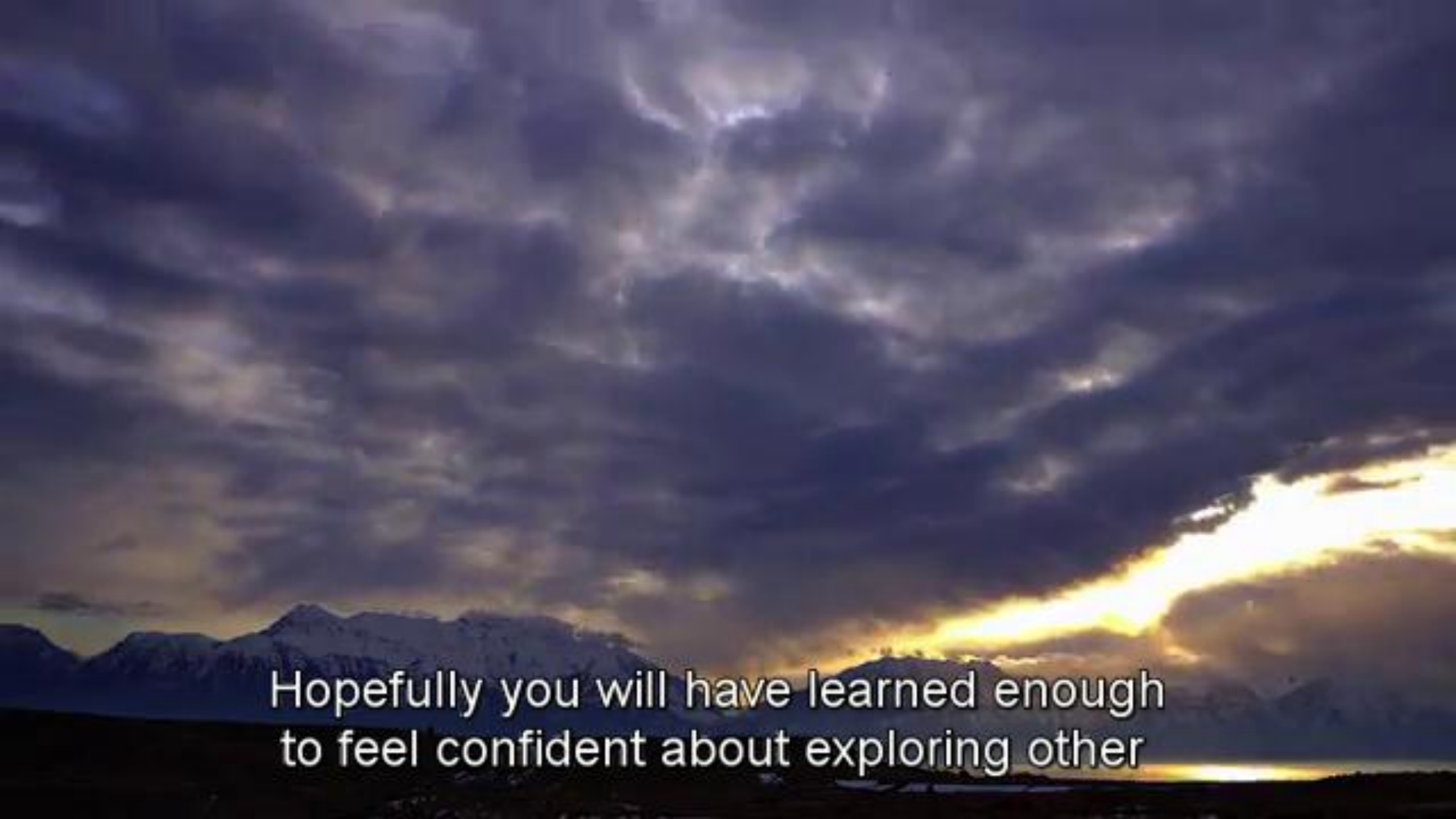

approaches to choosing initial centroids,
and the number of clusters to retain.


Some of these methods may be better
suited to the data you have, or


to the shapes and sizes of the clusters
you think might exist in your population.


K-means cluster analysis
is a good starting point


because its simplicity makes it
easier to convey the concepts.


Hopefully you will have learned enough
to feel confident about exploring other

methods.