Lecture 3 Clustering Techniques

May 11, 2022

1 Clustering Workbook

Welcome to the Clustering Workbook. Let's start with importing the required libraries.

[1]:
```
import numpy as np
import pandas as pd

from matplotlib import pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set_style('darkgrid')

%matplotlib inline
```

First, let's read the sample data set.

[2]:
```
sample_df = pd.read_csv('clustering_dataset.csv')
```

[3]:
```
sample_df.describe()
```

[3]:

	x	y
count	300.000000	300.000000
mean	3.646046	2.938750
std	2.268252	1.840548
min	-1.714843	-0.680250
25%	1.355996	1.383061
50%	4.316122	2.691587
75%	5.425505	4.610866
max	7.392365	7.598304

[4]:
```
sample_df.head()
```

[4]:

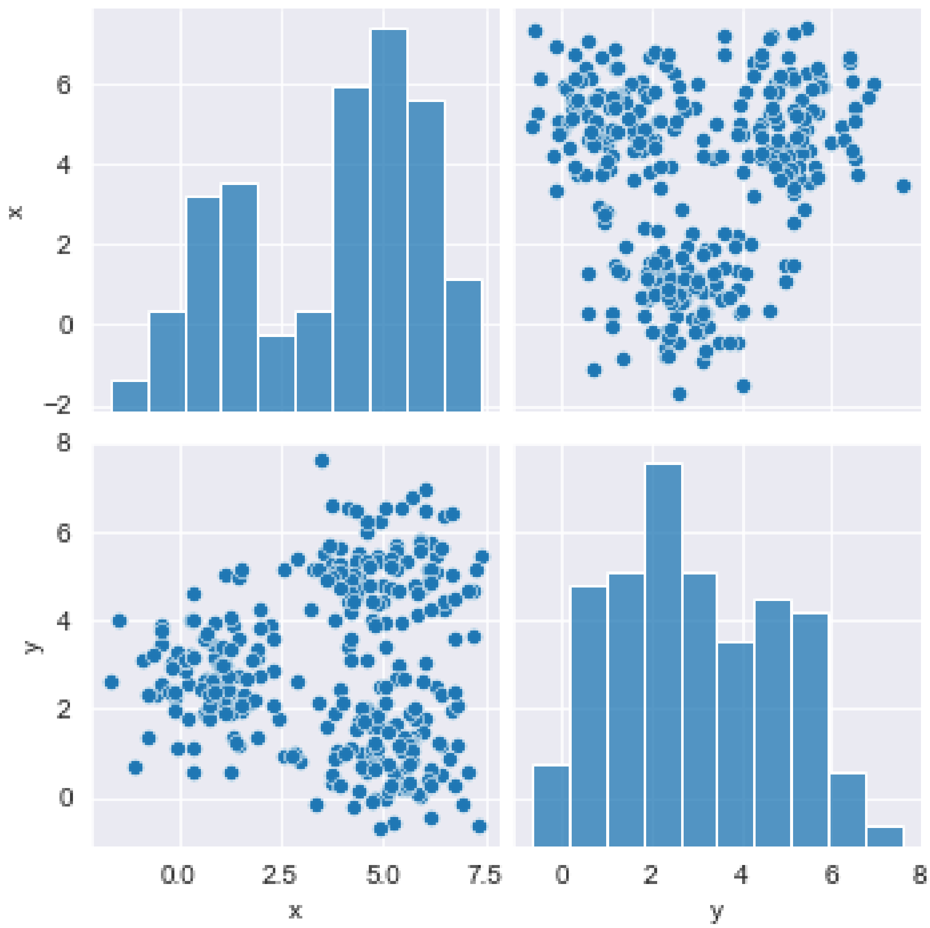
	x	y
0	3.914369	5.642055
1	5.997345	3.022112
2	5.282978	5.712265
3	3.493705	7.598304

```
4  4.421400  4.975374
```

Let's also plot the data set to try and understand how it looks like.

[5]:
```
sns.pairplot(data=sample_df)
```

[5]:
```
<seaborn.axisgrid.PairGrid at 0x135a67250>
```



1.1 KMeans Clustering

First, we will explore KMeans clustering.

[6]:
```
from sklearn.cluster import KMeans
```

[7]:
```
X = sample_df
```

I will start with a case of 3 clusters.

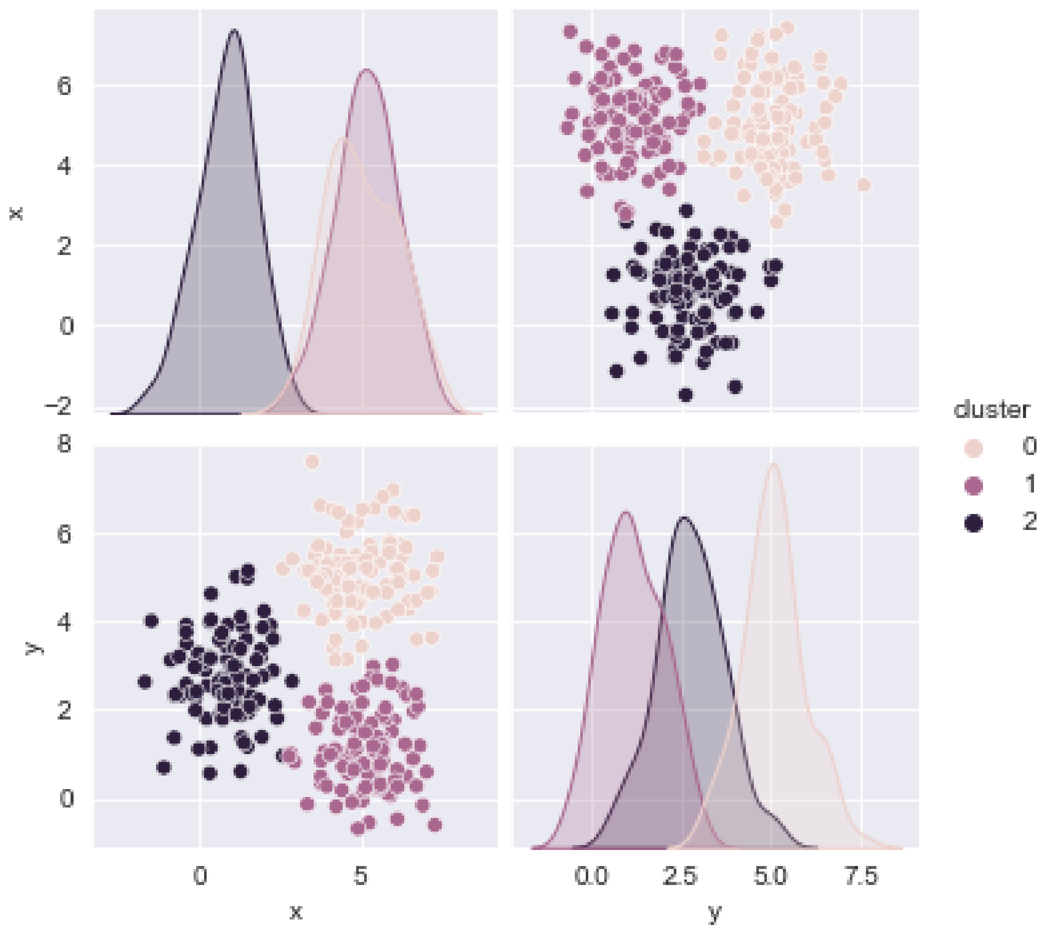
[8]:
```
kmeans = KMeans(n_clusters=3, random_state=123)
kmeans.fit(X)
```

[8]:
```
KMeans(n_clusters=3, random_state=123)
```

[9]:
```
clustered_data = sample_df.copy()
clustered_data['cluster'] = kmeans.predict(X)
```

[10]:
```
sns.pairplot(data=clustered_data, hue='cluster')
```

[10]:
```
<seaborn.axisgrid.PairGrid at 0x136c523a0>
```



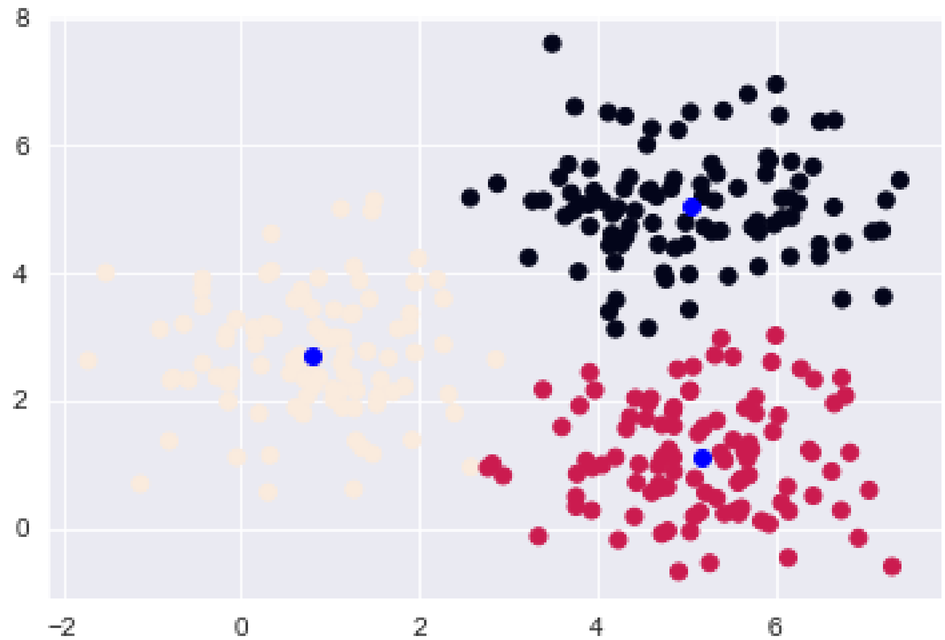
[11]:
```
clustered_data.head()
```

[11]:

	x	y	cluster
0	3.914369	5.642055	0
1	5.997345	3.022112	1
2	5.282978	5.712265	0

```
3  3.493705  7.598304        0
4  4.421400  4.975374        0
```

[12]:
```
# Scatterplot, colored by cluster
plt.figure()
plt.scatter(clustered_data.x, clustered_data.y, c=clustered_data.cluster)
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],c='b')
#sns.lmplot(x='x', y='y', hue='cluster', data=clustered_data, fit_reg=False)
plt.show()
```



[13]:
```
kmeans.cluster_centers_
```

[13]:
```
array([[5.03528393, 5.06129708],
       [5.15219323, 1.12755249],
       [0.790767  , 2.72939311]])
```

[14]:
```
kmeans.labels_
```

[14]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
```

```
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

Is this a good clustering? What are some problematic points?

1.1.1 Iteration Number

Can we go back to basics? How does the KMeans work so well? Let's make it so that there will only be one guess.

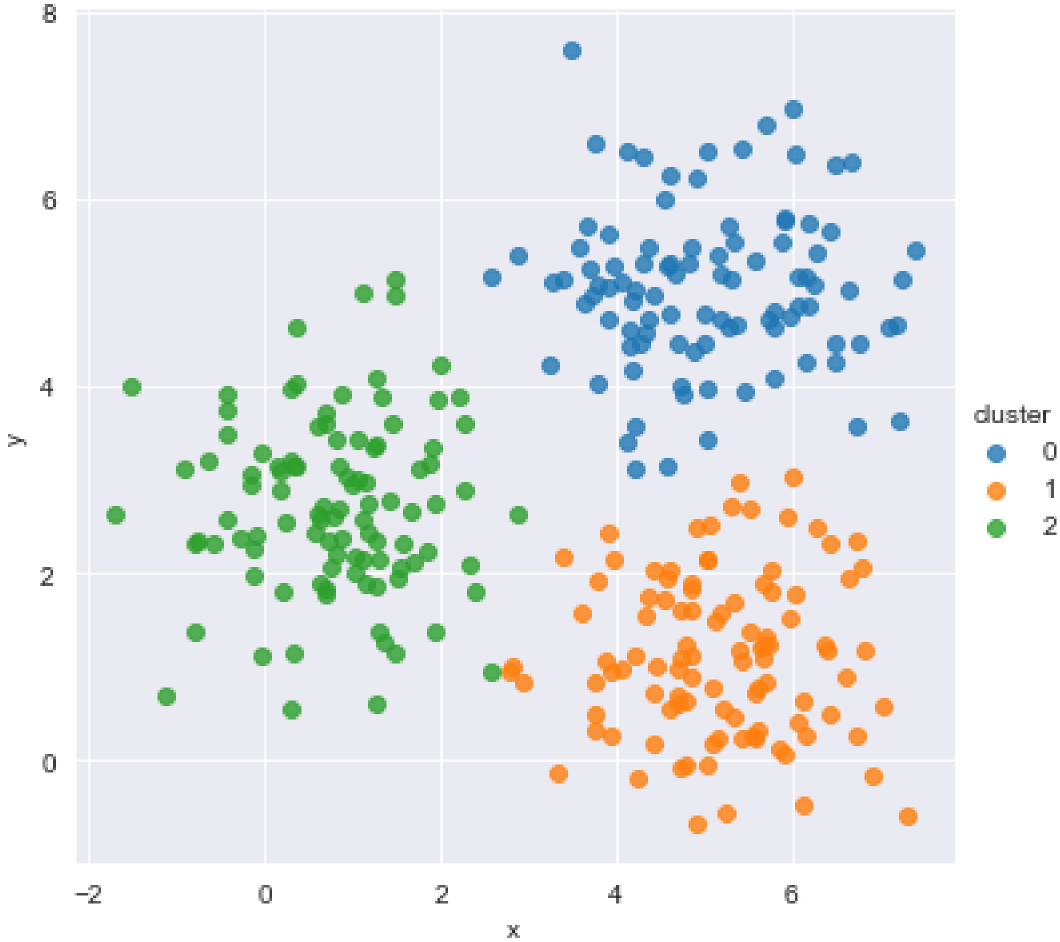
[15]:
```
# Fit K-Means (but only allow 1 iteration)
kmeans_m1 = KMeans(n_clusters=3, random_state=123, max_iter=1)
```

[16]:
```
kmeans_m1.fit(X)
```

[16]:
```
KMeans(max_iter=1, n_clusters=3, random_state=123)
```

[17]:
```
clustered_data = sample_df.copy()#X.copy()
clustered_data['cluster'] = kmeans_m1.predict(X)
```

[18]:
```
# Scatterplot, colored by cluster
sns.lmplot(x='x', y='y', hue='cluster', data=clustered_data, fit_reg=False)
plt.show()
```



This is still bull's eye at the first try. What are we missing?

[19]:
```
kmeans_m1.n_init
```

[19]: 10

Of course, initialization iterations.

[20]:
```
kmeans_m2 = KMeans(n_clusters=3,
                   random_state=5,
                   max_iter=1, # Only allow 1 max iteration
                   n_init=2, # Only allow 1 set of centroid starting seeds
                   init='random') # choose random initial centroids.
```

[21]:
```
kmeans_m2.fit(X)
```

[21]: KMeans(init='random', max_iter=1, n_clusters=3, n_init=2, random_state=5)

[22]:
```
clustered_data = sample_df.copy()#X.copy()
clustered_data['cluster'] = kmeans_m2.predict(X)
```

[23]:
```
# Scatterplot, colored by cluster
sns.lmplot(x='x', y='y', hue='cluster', data=clustered_data, fit_reg=False)
plt.show()
```

Not so good now. So we know, that the initialization also starts with iterations of guesses.

1.1.2 Number of Clusters

Now let's explore how different number of clusters change the result.

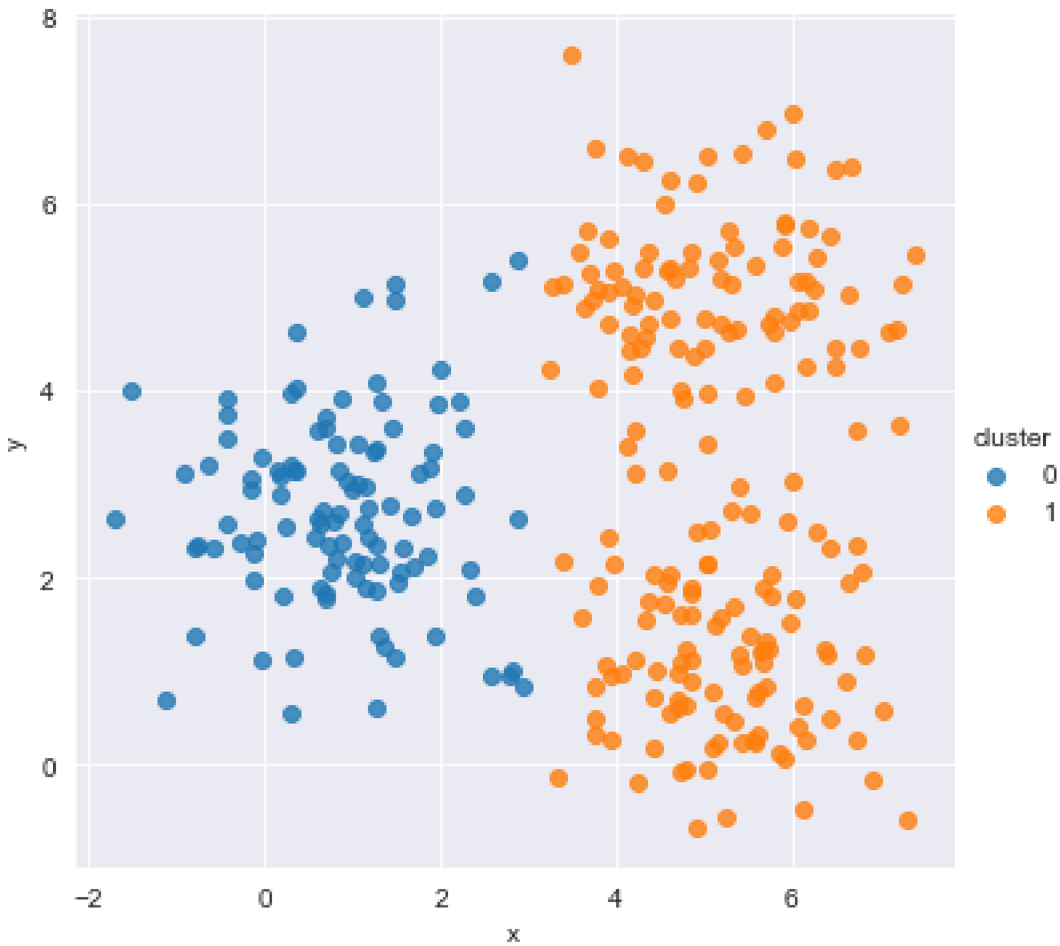
[24]:
```
kmeans_m3 = KMeans(n_clusters=2,
                   random_state=5,
                   max_iter=1, # Only allow 1 max iteration
                   n_init=1, # Only allow 1 set of centroid starting seeds
                   init='random') # choose random initial centroids.
```

[25]:
```
kmeans_m3.fit(X)
```

[25]:
```
KMeans(init='random', max_iter=1, n_clusters=2, n_init=1, random_state=5)
```

[26]:
```
clustered_data = sample_df.copy()#X.copy()
clustered_data['cluster'] = kmeans_m3.predict(X)
```

[27]:
```
# Scatterplot, colored by cluster
sns.lmplot(x='x', y='y', hue='cluster', data=clustered_data, fit_reg=False)
plt.show()
```



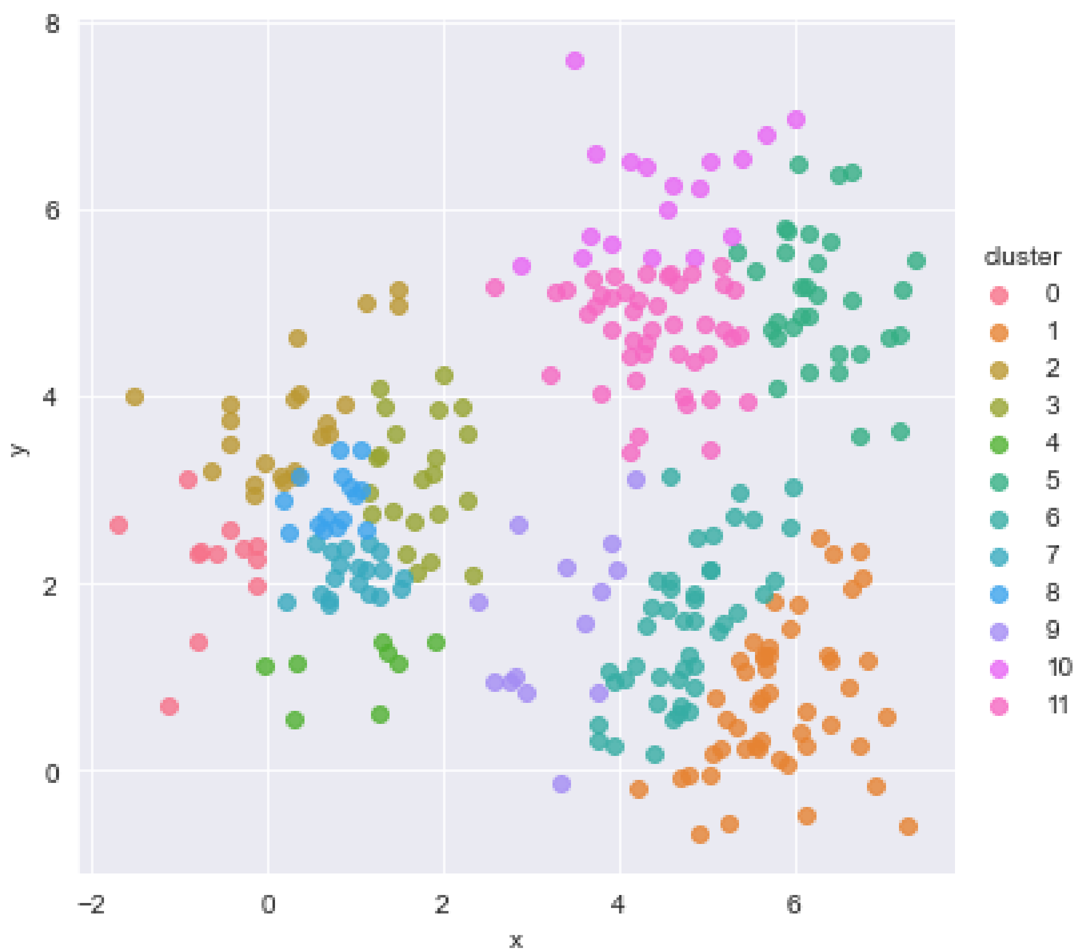
[28]:
```
kmeans_m4 = KMeans(n_clusters=12,
                   random_state=5,
                   max_iter=1, # Only allow 1 max iteration
                   n_init=1, # Only allow 1 set of centroid starting seeds
                   init='random') # choose random initial centroids.
```

[29]:
```
kmeans_m4.fit(X)
```

[29]:
```
KMeans(init='random', max_iter=1, n_clusters=12, n_init=1, random_state=5)
```

[30]:
```
clustered_data = sample_df.copy()#X.copy()
clustered_data['cluster'] = kmeans_m4.predict(X)
```

[31]:
```
# Scatterplot, colored by cluster
sns.lmplot(x='x', y='y', hue='cluster', data=clustered_data, fit_reg=False)
plt.show()
```



May be a bit too much.

1.1.3 MiniBatch KMeans

[32]:
```
from sklearn.cluster import MiniBatchKMeans
```

[33]:
```
mini_kmeans = MiniBatchKMeans(n_clusters=3)
```

[34]:
```
mini_kmeans.fit(X)
```

[34]: MiniBatchKMeans(n_clusters=3)

[35]:
```
clustered_data = sample_df.copy()#X.copy()
clustered_data['cluster'] = mini_kmeans.predict(X)
```

[36]:
```
# Scatterplot, colored by cluster
sns.lmplot(x='x', y='y', hue='cluster', data=clustered_data, fit_reg=False)
plt.show()
```

1.2 Evaluation Criteria

It is hard to understand this by looking. Let's explore some popular evaluation metrics for clustering.

1.2.1 Inertia

[37]:
```
mini_kmeans.inertia_
```

[37]:
```
534.4834618971341
```

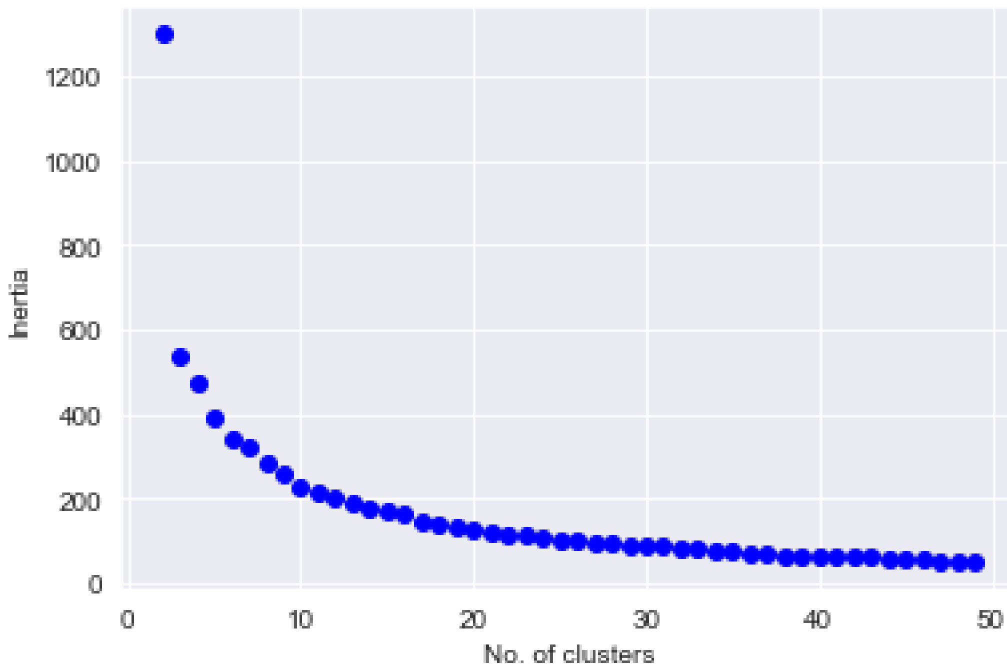
[38]:
```
kmeans_m2.inertia_
```

[38]:
```
539.6191222803174
```

[39]:
```
def fit_kmeans_model(n_clusters, X):
    model = KMeans(n_clusters=n_clusters,
                  random_state=5,
                  max_iter=1, # Only allow 1 max iteration
                  init='random') # choose random initial centroids.
    model.fit(X)

    clustered_data = sample_df.copy()
    clustered_data['cluster'] = model.predict(X)

    return (clustered_data, model)
```

[40]:
```
fig = plt.figure(figsize=[6,4])
ax = plt.subplot(111)

for n in range(2,50):
    clustered_data, model = fit_kmeans_model(n, X)
    ax.scatter(n, model.inertia_, color='b')

ax.set_ylabel('Inertia')
ax.set_xlabel('No. of clusters')
```

[40]:
```
Text(0.5, 0, 'No. of clusters')
```



There are three performance intervals here.

[41]:
```
clustered_data, model = fit_kmeans_model(6, X)
```

[42]:
```
# Scatterplot, colored by cluster
sns.lmplot(x='x', y='y', hue='cluster', data=clustered_data, fit_reg=False)
plt.show()
```

1.2.2 Silhouette Score

Now let's see what the silhouette score would tell for the model performance.

[43]:
```
from sklearn.metrics import silhouette_score
```

[44]:
```
silhouette_score(X, model.labels_)
```

[44]: 0.3416026475169336

[45]:
```
%matplotlib notebook
fig = plt.figure(figsize=[6,4])
ax = plt.subplot(111)

for n in range(2,50):
    clustered_data, model = fit_kmeans_model(n, X)
    ax.scatter(n, silhouette_score(X, model.labels_),color='b')
```

```
ax.set_ylabel('Silhouette Score')
ax.set_xlabel('No. of clusters')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

[45]:
```
Text(0.5, 0, 'No. of clusters')
```

The second highest value is 6.

1.2.3 KMeans Failure

We have seen that KMeans is doing okay with the data set we have looked at so far.

[46]:
```
from sklearn.datasets import make_moons
```

[47]:
```
X, y = make_moons(n_samples=300, noise=0.08, random_state=0)
moon_df = pd.DataFrame({'X1':X[:,0], 'X2':X[:,1], 'y':y})
```

[48]:
```
sns.lmplot(x='X1', y='X2', data=moon_df)
plt.show()
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

[49]:
```
moon_df
```

[49]:
```
           X1        X2  y
0    0.749212 -0.526487  1
1    0.212365 -0.253225  1
2    0.928320  0.404686  0
3    1.002225 -0.490274  1
4    1.202950 -0.386080  1
..        ...       ... ..
295  1.530946 -0.290321  1
296  0.221195 -0.249975  1
297 -0.719076  0.491739  0
298  0.583732  0.972254  0
299  0.104479  0.118310  1

[300 rows x 3 columns]
```

[50]:
```
kmeans = KMeans(n_clusters=2)
```

[51]:
```
kmeans.fit(X)
```

[51]: KMeans(n_clusters=2)

[52]:
```
y_pred=kmeans.predict(X)
moon_df['predicted_clusters'] = y_pred
```

[53]:
```
sns.lmplot(x='X1', y='X2', hue='predicted_clusters', data=moon_df,␣
↪fit_reg=False)
plt.show()
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[54]:
```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_pred=kmeans.predict(X)
moon_df['predicted_clusters2'] = y_pred
```

[55]:
```
sns.lmplot(x='X1', y='X2', hue='predicted_clusters2', data=moon_df,␣
↪fit_reg=False)
plt.show()
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[56]:
```
kmeans = KMeans(n_clusters=3, max_iter=100)
kmeans.fit(X)
y_pred=kmeans.predict(X)
moon_df['predicted_clusters3'] = y_pred
```

[57]:
```
sns.lmplot(x='X1', y='X2', hue='predicted_clusters3', data=moon_df,␣
↪fit_reg=False)
plt.show()
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[58]:
```
kmeans = KMeans(n_clusters=12)
kmeans.fit(X)
y_pred=kmeans.predict(X)
moon_df['predicted_clusters4'] = y_pred
```

[59]:
```
sns.lmplot(x='X1', y='X2', hue='predicted_clusters4', data=moon_df,␣
↪fit_reg=False)
plt.show()
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

Not getting any better.

1.3 DBSCAN

[60]:
```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
```

[61]:
```
dbscan = DBSCAN(eps=0.35)
```

DBSCAN requires scaling. When we are training a model from scratch, we will always scale+train.

[62]:
```
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

[63]:
```
clusters = dbscan.fit_predict(X_scaled)
moon_df['dbscan_clusters'] = clusters
```

[64]:
```
clusters
```

[64]:
```
array([0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0])
```

[65]:
```
fig = plt.figure(figsize=[6,4])
ax = plt.subplot(111)

ax.scatter(moon_df.X1, moon_df.X2, c=clusters, cmap='PiYG')

ax.set_ylabel('Silhouette Score')
ax.set_xlabel('Epsilon Value')
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[65]: Text(0.5, 0, 'Epsilon Value')

Pretty good!

[66]:
```
silhouette_score(X_scaled, clusters)
```

[66]: 0.38135450723457787

[67]:
```
def fit_dbscan(eps):
    dbscan = DBSCAN(eps=eps)
    clusters = dbscan.fit_predict(X_scaled)
    return clusters
```

[68]:
```
eps_list = list(np.arange(0.1,0.35,0.01))
```

[69]:
```
fig = plt.figure(figsize=[6,4])
ax = plt.subplot(111)

for eps in (eps_list):
    clusters = fit_dbscan(eps)
    ax.scatter(eps, silhouette_score(X_scaled, clusters), color='b')

ax.set_ylabel('Silhouette Score')
ax.set_xlabel('Epsilon Value')
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[69]: Text(0.5, 0, 'Epsilon Value')

DBSCAN does not have inertia, that's why we are only using silhouette score here. Congratulations, you have completed the Clustering Workbook!

[ ]:
```

```