Customer Segmentation using Clustering

This mini-project is based on this blog post (http://blog.yhat.com/posts/customer-segmentation-using-python.html) by yhat. Please feel free to refer to the post for additional information, and solutions.

Customer Segmentation using Clustering

Applying clustering algorithm on dataset containing information on marketing newsletters/e-mail campaigns (e-mail offers sent to customers) and transaction level data from customers for customer segmentation.

Approach

- Data Wrangling.
- Apply KMean algorithm. Also apply different methods for choosing K.
  - Elbow method
  - Silhouette method
- Visualizing clusters using PCA.

In [1]:

```
%matplotlib inline
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns

# Setup Seaborn
sns.set_style("whitegrid")
sns.set_context("poster")

# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
# ignore all user warnings
simplefilter(action='ignore', category=UserWarning)
```

Data

The dataset contains information on marketing newsletters/e-mail campaigns (e-mail offers sent to customers) and transaction level data from customers. The transactional data shows which offer customers responded to, and what the customer ended up buying. The data is presented as an Excel workbook containing two worksheets. Each worksheet contains a different dataset.

In [2]:

```
df_offers = pd.read_excel("./WineKMC.xlsx", sheetname=0)
df_offers.columns = ["offer_id", "campaign", "varietal", "min_qty", "discount", "origi
n", "past_peak"]
df_offers.head()
```

Out[2]:

	offer_id	campaign	varietal	min_qty	discount	origin	past_peak
0	1	January	Malbec	72	56	France	False
1	2	January	Pinot Noir	72	17	France	False
2	3	February	Espumante	144	32	Oregon	True
3	4	February	Champagne	72	48	France	True
4	5	February	Cabernet Sauvignon	144	44	New Zealand	True

We see that the first dataset contains information about each offer such as the month it is in effect and several attributes about the wine that the offer refers to: the variety, minimum quantity, discount, country of origin and whether or not it is past peak. The second dataset in the second worksheet contains transactional data -- which offer each customer responded to.

In [3]:

```
df_transactions = pd.read_excel("./WineKMC.xlsx", sheetname=1)
df_transactions.columns = ["customer_name", "offer_id"]
df_transactions['n'] = 1
df_transactions.head()
```

Out[3]:

	customer_name	offer_id	n
0	Smith	2	1
1	Smith	24	1
2	Johnson	17	1
3	Johnson	24	1
4	Johnson	26	1

Data wrangling

We're trying to learn more about how our customers behave, so we can use their behavior (whether or not they purchased something based on an offer) as a way to group similar minded customers together. We can then study those groups to look for patterns and trends which can help us formulate future offers.

The first thing we need is a way to compare customers. To do this, we're going to create a matrix that contains each customer and a 0/1 indicator for whether or not they responded to a given offer.

Checkup Exercise Set I

Exercise: Create a data frame where each row has the following columns (Use the pandas [`merge`](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html) and [`pivot_table`](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.pivot_table.html) functions for this purpose):

- customer_name
- One column for each offer, with a 1 if the customer responded to the offer

Make sure you also deal with any weird values such as `NaN`. Read the documentation to develop your solution.

In [4]:

```
df = pd.merge(df_transactions[['customer_name', 'offer_id', 'n']], df_offers[['offer_id']], on = 'offer_id')
df.head()
```

Out[4]:

	customer_name	offer_id	n
0	Smith	2	1
1	Rodriguez	2	1
2	Martin	2	1
3	Jackson	2	1
4	Campbell	2	1

In [5]:

```
table = pd.pivot_table(df, values = 'n', index = ['customer_name'], columns = ['offer_id'], fill_value = 0.0)
table.head()
```

Out[5]:

offer_id	1	2	3	4	5	6	7	8	9	10	...	23	24	25	26	27	28	29	30	31	32
customer_name																					
Adams	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	1	1	0	0
Allen	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	1	0	0	0	0	0
Anderson	0	0	0	0	0	0	0	0	0	0	...	0	1	0	1	0	0	0	0	0	0
Bailey	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	1	0	0
Baker	0	0	0	0	0	0	1	0	0	1	...	0	0	0	0	0	0	0	0	1	0

5 rows × 32 columns

K-Means Clustering

Recall that in K-Means Clustering we want to *maximize* the distance between centroids and *minimize* the distance between data points and the respective centroid for the cluster they are in. True evaluation for unsupervised learning would require labeled data; however, we can use a variety of intuitive metrics to try to pick the number of clusters K. We will introduce two methods: the Elbow method, the Silhouette method and the gap statistic.

Choosing K: The Elbow Sum-of-Squares Method

The first method looks at the sum-of-squares error in each cluster against K. We compute the distance from each data point to the center of the cluster (centroid) to which the data point was assigned.

$$SS = \sum_k \sum_{x_i \in C_k} \sum_{x_j \in C_k} (x_i - x_j)^2 = \sum_k \sum_{x_i \in C_k} (x_i - \mu_k)^2$$

where x_i is a point, C_k represents cluster k and μ_k is the centroid for cluster k. We can plot SS vs. K and choose the *elbow point* in the plot as the best value for K. The elbow point is the point at which the plot starts descending much more slowly.

Checkup Exercise Set II

Exercise:

- What values of SS do you believe represent better clusterings? Why?
- Create a numpy matrix `x_cols` with only the columns representing the offers (i.e. the 0/1 colums)
- Write code that applies the [`KMeans`](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html) clustering method from scikit-learn to this matrix.
- Construct a plot showing SS for each K and pick K using this plot. For simplicity, test $2 \leq K \leq 10$.
- Make a bar chart showing the number of points in each cluster for k-means under the best K.
- What challenges did you experience using the Elbow method to pick K?

Code that applies the KMeans clustering method from scikit-learn to this matrix.

In [6]:

```
x_cols = table.as_matrix()
x_cols
```

Out[6]:

```
array([[0, 0, 0, ..., 1, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [1, 0, 0, ..., 0, 1, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 1, 1]], dtype=int64)
```

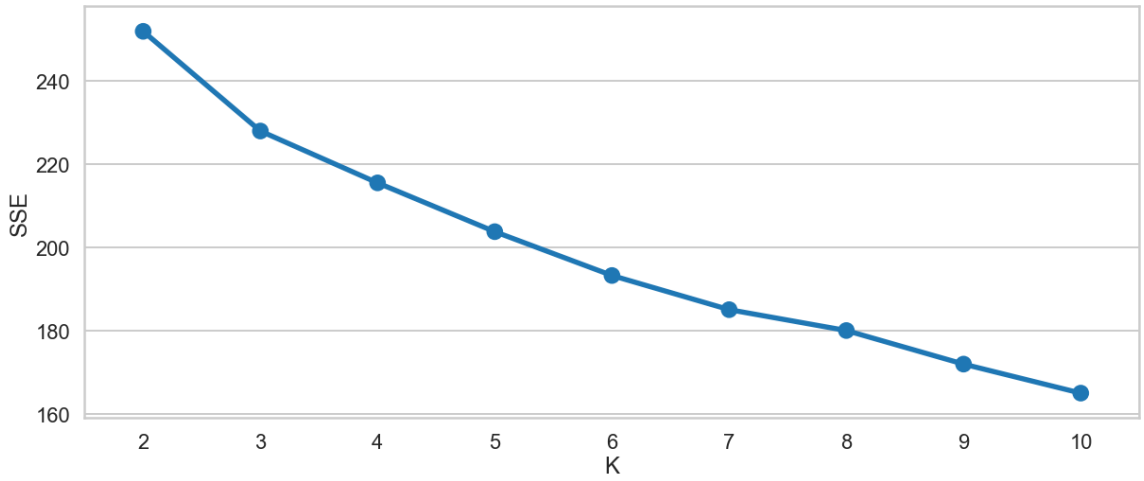
In [7]:

```
from sklearn.cluster import KMeans
import numpy as np

ks = np.arange(2, 11)
scores = []

# Apply kmeans modelling
for k in ks:
    kmeans = KMeans(n_clusters = k, random_state = 10)
    kmeans.fit(x_cols)
    scores.append(kmeans.inertia_)
```

Construct a plot showing SS for each K and pick K using this plot. For simplicity, test $2 \leq K \leq 10$

In [8]:

```
fig, ax = plt.subplots(figsize = (20,8))
sns.pointplot(x = ks, y = scores, ax = ax)
plt.xlabel('K')
plt.ylabel('SSE')
plt.show()
```

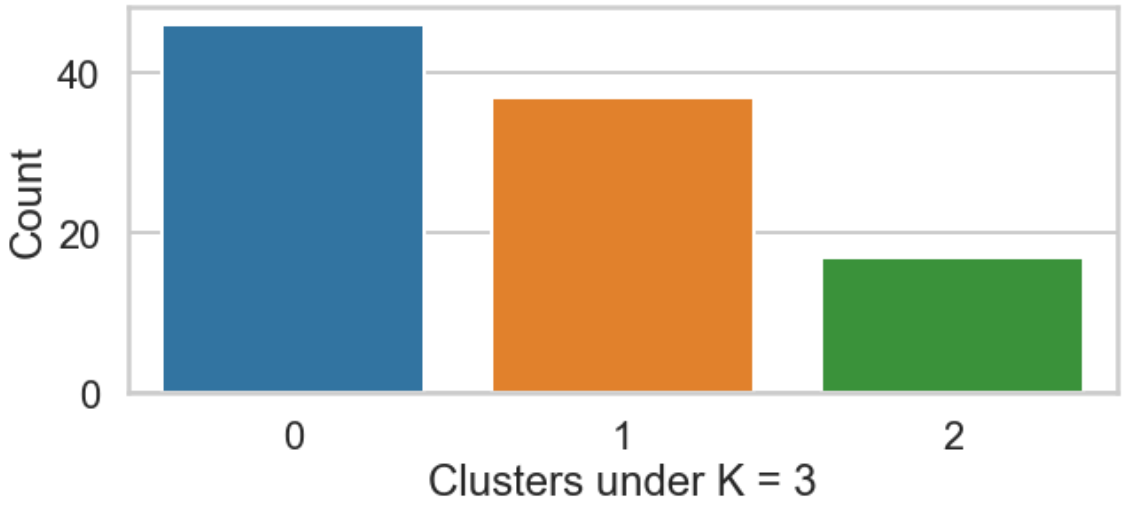


Make a bar chart showing the number of points in each cluster for k-means under the best K.

The best K seems to be 3.

In [9]:

```
kmeans = KMeans(n_clusters = 3, random_state = 10)
kmeans.fit(x_cols)
labels = kmeans.predict(x_cols)

fig, ax = plt.subplots(figsize = (10, 4))
sns.countplot(x = labels, ax = ax)
plt.xlabel('Clusters under K = 3')
plt.ylabel('Count')
plt.show()
```



What challenges did you experience using the Elbow method to pick K?

It's hard to pick K when the plot starts descending 'much more slowly' as in the above plot.

Choosing K: The Silhouette Method

There exists another method that measures how well each datapoint x_i "fits" its assigned cluster *and also* how poorly it fits into other clusters. This is a different way of looking at the same objective. Denote a_{x_i} as the *average* distance from x_i to all other points within its own cluster k. The lower the value, the better. On the other hand b_{x_i} is the minimum average distance from x_i to points in a different cluster, minimized over clusters. That is, compute separately for each cluster the average distance from x_i to the points within that cluster, and then take the minimum. The silhouette $s(x_i)$ is defined as

$$s(x_i) = \frac{b_{x_i} - a_{x_i}}{\max\left(a_{x_i}, b_{x_i}\right)}$$

The silhouette score is computed on *every datapoint in every cluster*. The silhouette score ranges from -1 (a poor clustering) to +1 (a very dense clustering) with 0 denoting the situation where clusters overlap. Some criteria for the silhouette coefficient is provided in the table below.

Range	Interpretation
0.71 - 1.0	A strong structure has been found.
0.51 - 0.7	A reasonable structure has been found.
0.26 - 0.5	The structure is weak and could be artificial.
< 0.25	No substantial structure has been found.

</pre>
Source: http://www.stat.berkeley.edu/~spector/s133/Clus.html (http://www.stat.berkeley.edu/~spector/s133/Clus.html)

Fortunately, scikit-learn provides a function to compute this for us (phew!) called `sklearn.metrics.silhouette_score` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html). Take a look at this article (http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html) on picking K in scikit-learn, as it will help you in the next exercise set.

Checkup Exercise Set III

Exercise: Using the documentation for the `silhouette_score` function above, construct a series of silhouette plots like the ones in the article linked above.

Exercise: Compute the average silhouette score for each K and plot it. What K does the plot suggest we should choose? Does it differ from what we found using the Elbow method?

In [10]:

```
from sklearn.metrics import silhouette_samples, silhouette_score

range_n_clusters = list(range(2,11))
silhouette_scores = []

for n_clusters in range_n_clusters:

    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(x_cols)

    silhouette_avg = silhouette_score(x_cols, cluster_labels)
    silhouette_scores.append(silhouette_avg)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)
```

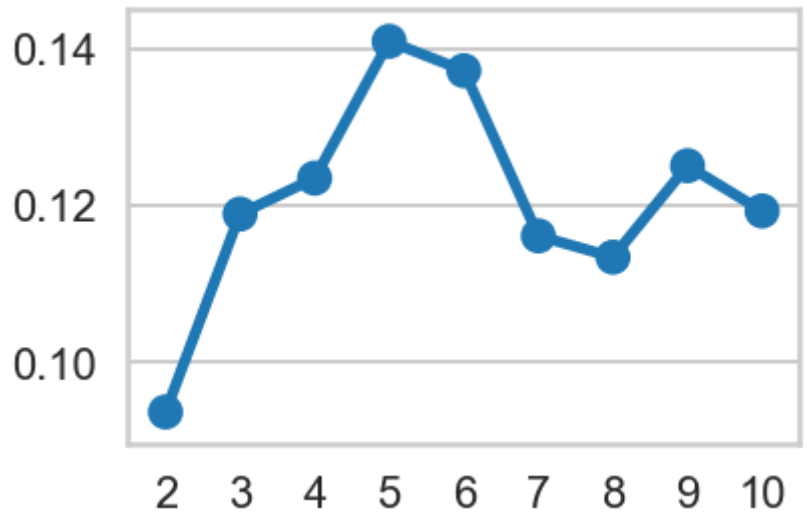
```
For n_clusters = 2 The average silhouette_score is : 0.09365573283492713
For n_clusters = 3 The average silhouette_score is : 0.11889942863638556
For n_clusters = 4 The average silhouette_score is : 0.12347053919571699
For n_clusters = 5 The average silhouette_score is : 0.14092516241984757
For n_clusters = 6 The average silhouette_score is : 0.1371798939109807
For n_clusters = 7 The average silhouette_score is : 0.1161092456616906
For n_clusters = 8 The average silhouette_score is : 0.11339573832632867
For n_clusters = 9 The average silhouette_score is : 0.12505960527779877
For n_clusters = 10 The average silhouette_score is : 0.11928332134753233
```

In [11]:

```
sns.pointplot(x=range_n_clusters, y=silhouette_scores)
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1822d4aa8d0>
```



From the values of silhouette score obtained, we will choose the cluster correponding to the highest silhouette score. We see that the cluster size value is 5,

This is different from the result obtained through the Elbow Method. From the table below,n_clusters = 5 has the best silhouette_score 0.14092, which is different from 3, where we found using Elbow method.

Choosing K: The Gap Statistic

There is one last method worth covering for picking K, the so-called Gap statistic. The computation for the gap statistic builds on the sum-of-squares established in the Elbow method discussion, and compares it to the sum-of-squares of a "null distribution," that is, a random set of points with no clustering. The estimate for the optimal number of clusters K is the value for which $\log SS$ falls the farthest below that of the reference distribution:

$$G_k = E_n^*\{\log SS_k\} - \log SS_k$$

In other words a good clustering yields a much larger difference between the reference distribution and the clustered data. The reference distribution is a Monte Carlo (randomization) procedure that constructs B random distributions of points within the bounding box (limits) of the original data and then applies K-means to this synthetic distribution of data points.. $E_n^*\{\log SS_k\}$ is just the average SS_k over all B replicates. We then compute the standard deviation σ_{SS} of the values of SS_k computed from the B replicates of the reference distribution and compute

$$s_k = \sqrt{1 + 1/B}\sigma_{SS}$$

Finally, we choose $K = k$ such that $G_k \geq G_{k+1} - s_{k+1}$.

Aside: Choosing K when we Have Labels

Unsupervised learning expects that we do not have the labels. In some situations, we may wish to cluster data that is labeled. Computing the optimal number of clusters is much easier if we have access to labels. There are several methods available. We will not go into the math or details since it is rare to have access to the labels, but we provide the names and references of these measures.

- Adjusted Rand Index
- Mutual Information
- V-Measure
- Fowlkes–Mallows index

See this article (http://scikit-learn.org/stable/modules/clustering.html) for more information about these metrics.

Visualizing Clusters using PCA

How do we visualize clusters? If we only had two features, we could likely plot the data as is. But we have 100 data points each containing 32 features (dimensions). Principal Component Analysis (PCA) will help us reduce the dimensionality of our data from 32 to something lower. For a visualization on the coordinate plane, we will use 2 dimensions. In this exercise, we're going to use it to transform our multi-dimensional dataset into a 2 dimensional dataset.

This is only one use of PCA for dimension reduction. We can also use PCA when we want to perform regression but we have a set of highly correlated variables. PCA untangles these correlations into a smaller number of features/predictors all of which are orthogonal (not correlated). PCA is also used to reduce a large set of variables into a much smaller one.

Checkup Exercise Set IV

Exercise: Use PCA to plot your clusters:

- Use scikit-learn's [`PCA`](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html) function to reduce the dimensionality of your clustering data to 2 components
- Create a data frame with the following fields:
  - customer name
  - cluster id the customer belongs to
  - the two PCA components (label them `x` and `y`)
- Plot a scatterplot of the `x` vs `y` columns
- Color-code points differently based on cluster ID
- How do the clusters look?
- Based on what you see, what seems to be the best value for K? Moreover, which method of choosing K seems to have produced the optimal result visually?

Exercise: Now look at both the original raw data about the offers and transactions and look at the fitted clusters. Tell a story about the clusters in context of the original data. For example, do the clusters correspond to wine variants or something else interesting?

Use scikit-learn's PCA function to reduce the dimensionality of your clustering data to 2 components

In [30]:

```
pca = PCA(n_components = 2).fit(x_cols)
x_2d = pca.transform(x_cols)
```

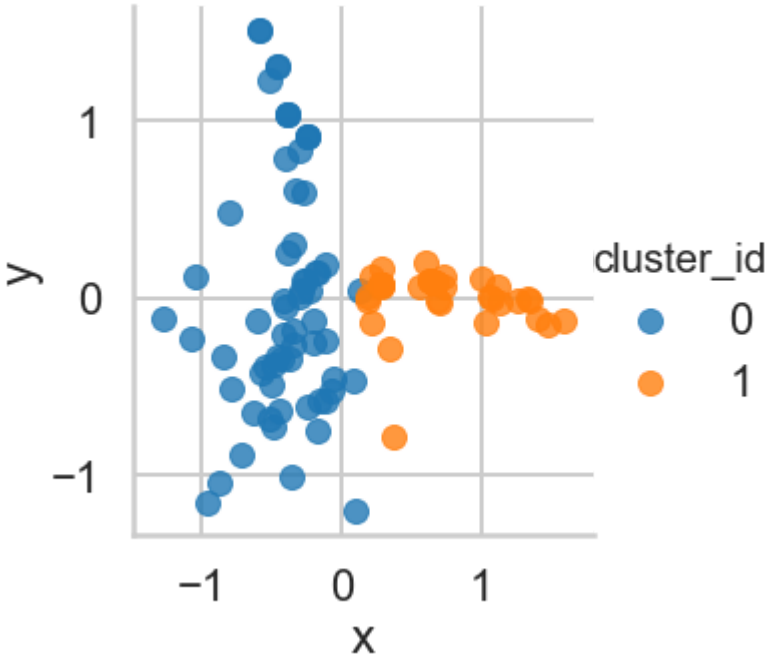
Create a data frame with the following fields:

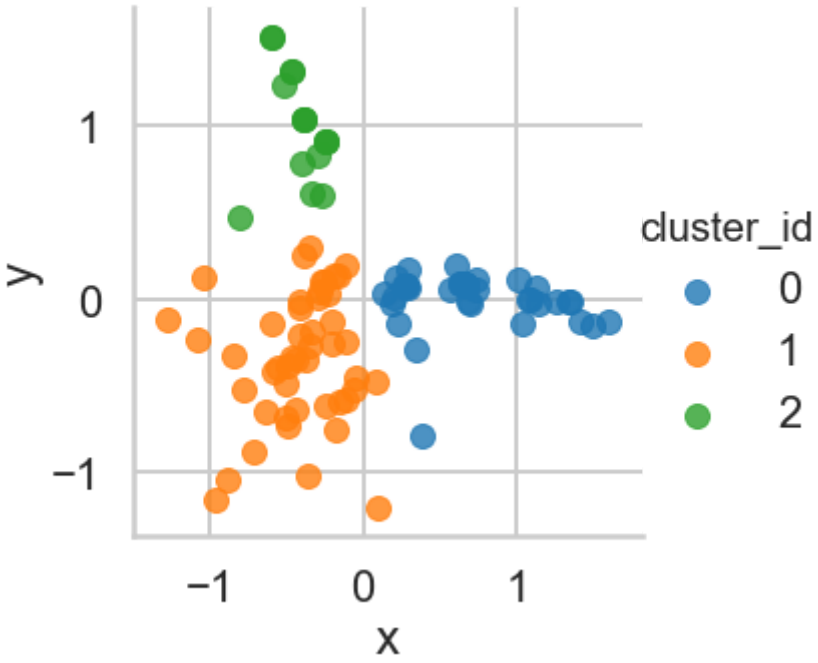
customer name cluster id the customer belongs to the two PCA components (label them x and y) Plot a scatterplot of the x vs y columns

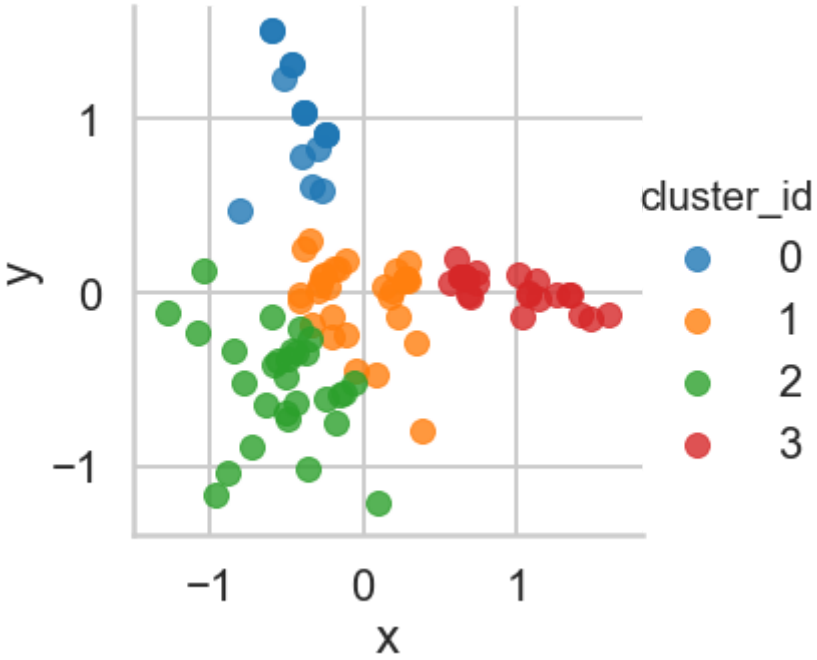
Color-code points differently based on cluster ID

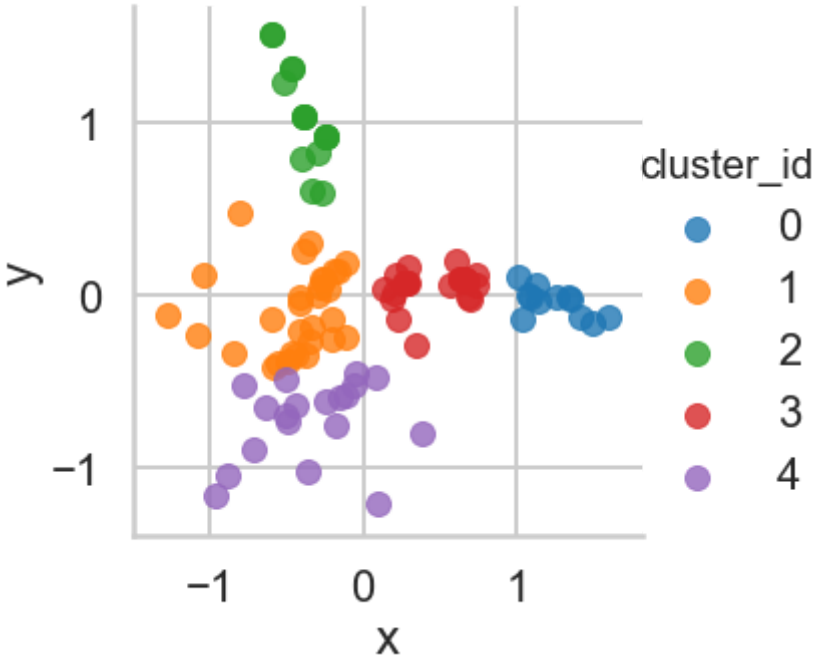
In [31]:

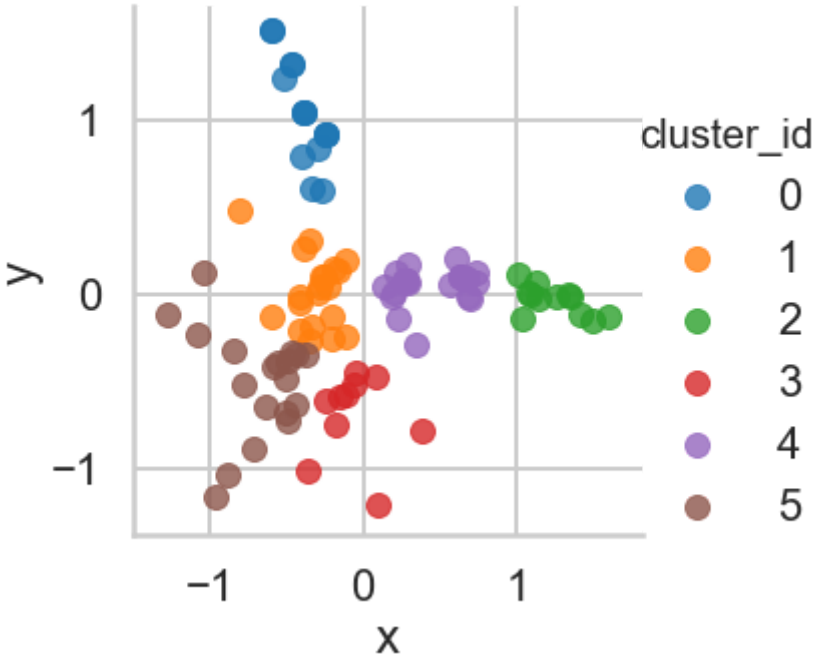
```
for k in ks:
    kmeans = KMeans(n_clusters = k, random_state = 10)
    kmeans.fit(x_2d)
    labels = kmeans.predict(x_2d)

    df_2d = pd.DataFrame({'customer_name' : table.index, 'cluster_id' : labels, 'x' : x
_2d[:, 0], 'y' : x_2d[:, 1]})

    sns.lmplot(x = 'x', y = 'y', hue = 'cluster_id', data = df_2d, fit_reg = False)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
```

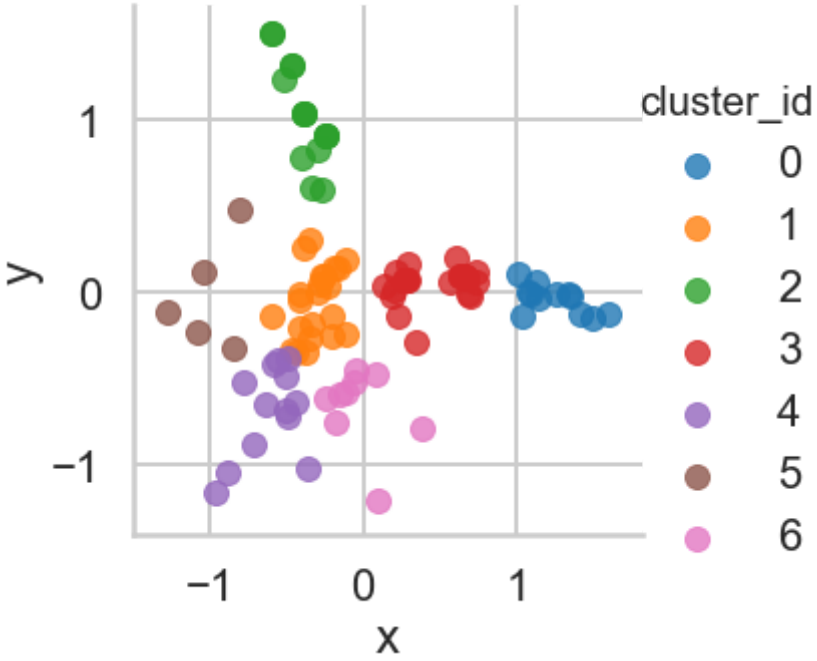

1
0
−1
y
−1
0
1
x
cluster_id
0
1

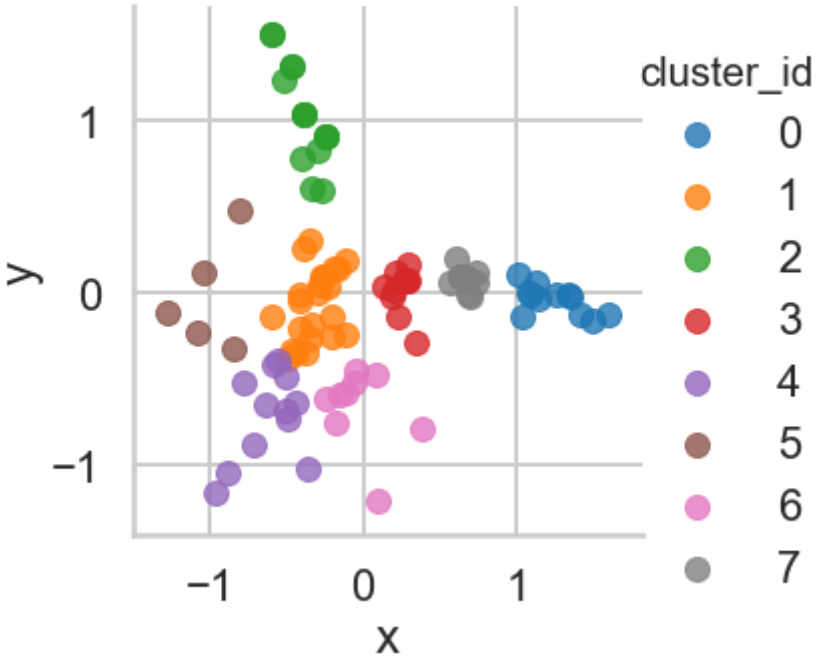

1
0
−1
y
−1
0
1
x
cluster_id
0
1
2

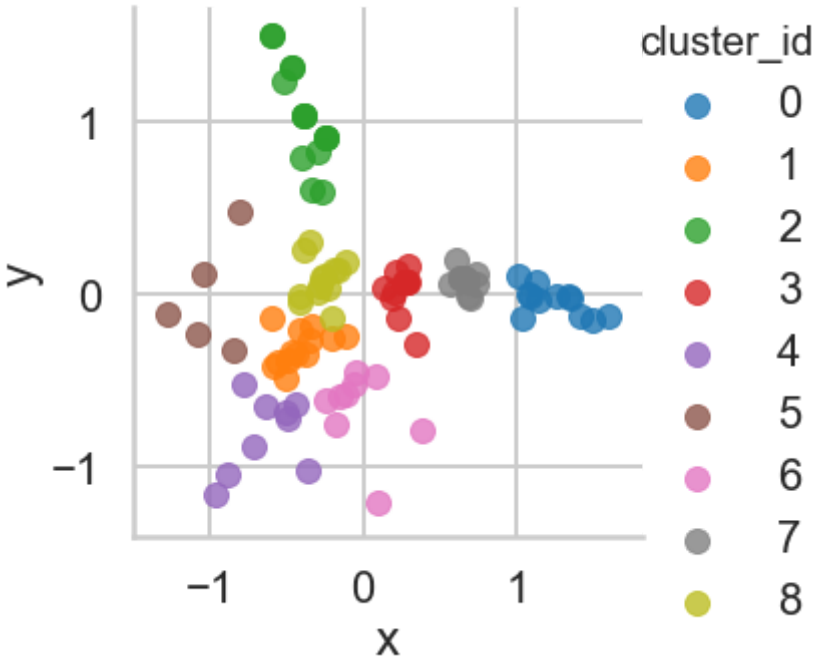

cluster_id
0
1
2
3
y
x
1
0
−1
−1
0
1

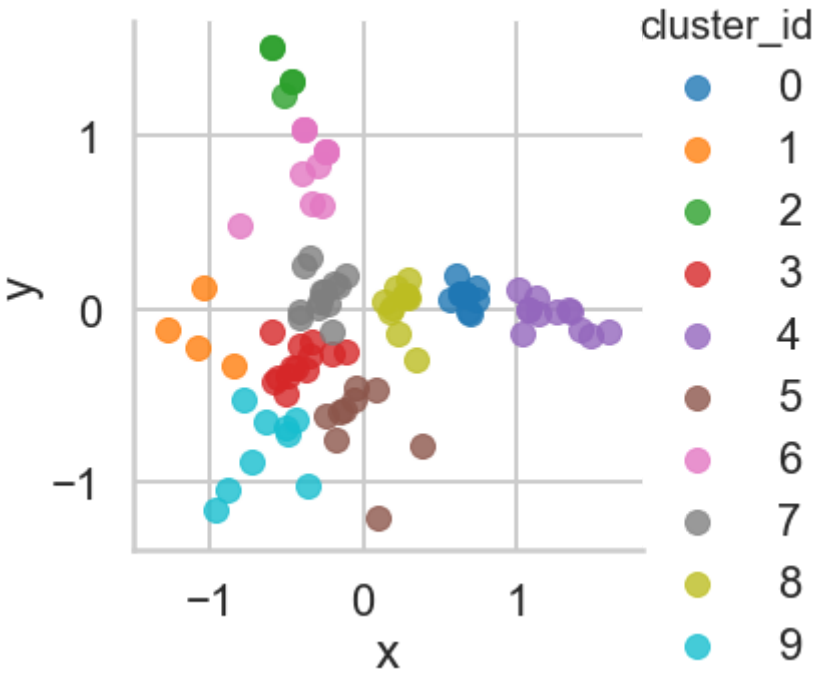

cluster_id
0
1
2
3
4
y
x
1
0
−1
−1
0
1


cluster_id
0
1
2
3
4
5
y
x
1
0
−1
−1
0
1


cluster_id
0
1
2
3
4
5
6
y
x
1
0
−1
−1
0
1


cluster_id
0
1
2
3
4
5
6
7
y
x
1
0
−1
−1
0
1


cluster_id
0
1
2
3
4
5
6
7
8
y
x
1
0
−1
−1
0
1



How do the clusters look?

The clusters look good when K is 3 ~ 5 and seems a little bit overfit when K >= 6.

Based on what you see, what seems to be the best value for K ? Moreover, which method of choosing K seems to have produced the optimal result visually?

Looking at the plots above, it seems that 3 ~ 5 are all good values for K.

In addition, PCA seems to be a good method as we can map the data into 2D and visulize the clusters.

What we've done is we've taken those columns of 0/1 indicator variables, and we've transformed them into a 2-D dataset. We took one column and arbitrarily called it `x` and then called the other `y`. Now we can throw each point into a scatterplot. We color coded each point based on it's cluster so it's easier to see them.

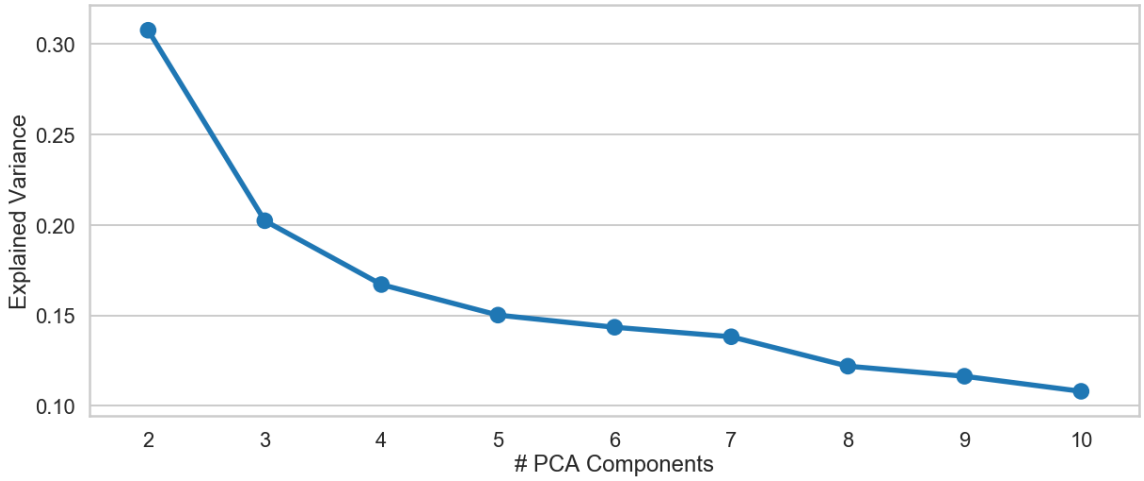
Exercise Set V

As we saw earlier, PCA has a lot of other uses. Since we wanted to visualize our data in 2 dimensions, restricted the number of dimensions to 2 in PCA. But what is the true optimal number of dimensions?

Exercise: Using a new PCA object shown in the next cell, plot the `explained_variance_` field and look for the elbow point, the point where the curve's rate of descent seems to slow sharply. This value is one possible value for the optimal number of dimensions. What is it?

In [13]:

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 10)
pca.fit(x_cols)
x_trans = pca.transform(x_cols)
variences = pca.explained_variance_

fig, ax = plt.subplots(figsize = (20,8))
sns.pointplot(x = ks, y = variences[1:], ax = ax)
plt.xlabel('# PCA Components')
plt.ylabel('Explained Variance')
plt.show()
```



Based on the above plot, 3 seems to be the elbow point, where the curve's rate of descent seems to slow sharply.

In [14]:

```
pca = PCA(n_components=2)
x_cols_2 = pca.fit_transform(x_cols)

df_pca = pd.DataFrame(x_cols_2)
df_pca.columns = ['x', 'y']
df_pca.head()
```

Out[14]:

	x	y
0	1.007580	0.108215
1	-0.287539	0.044715
2	-0.392032	1.038391
3	0.699477	-0.022542
4	0.088183	-0.471695

In [28]:

```
x_cols_2.shape
```

Out[28]:

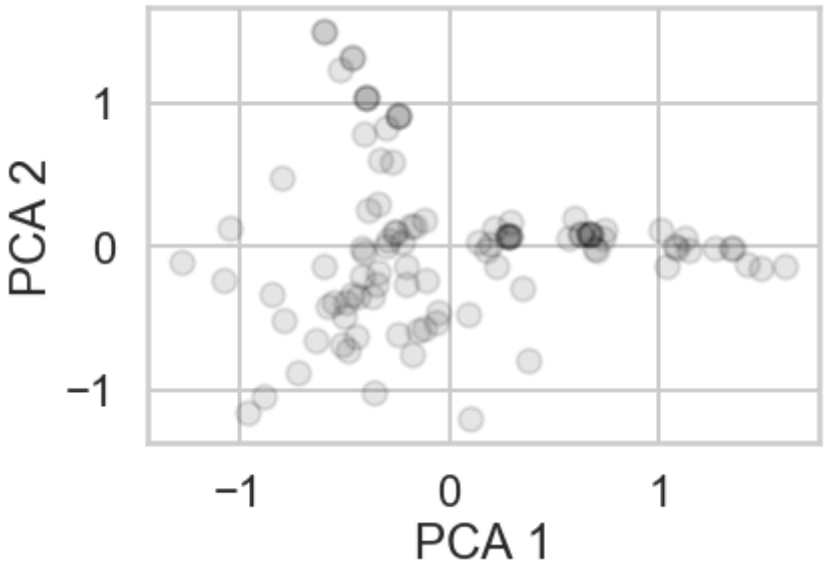
```
(100, 2)
```

In [29]:

```
plt.scatter(df_pca['x'], df_pca['y'], alpha=.1, color='black')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
```

Out[29]:

```
Text(0, 0.5, 'PCA 2')
```



In [ ]:

```

```

In [15]:

```
df_merged = df_pca.merge(table.reset_index(), left_index=True, right_index=True)
df_merged = df_merged.drop(list(range(1,33)), axis=1)
df_merged['label'] = clusterer.labels_
df_merged.head()
```

Out[15]:

	x	y	customer_name	label
0	1.007580	0.108215	Adams	1
1	-0.287539	0.044715	Allen	2
2	-0.392032	1.038391	Anderson	0
3	0.699477	-0.022542	Bailey	1
4	0.088183	-0.471695	Baker	3

In [16]:

```
df_merged['label_3'] = KMeans(n_clusters=3, random_state=42).fit(x_cols).labels_
```

In [17]:

```
df_merged['label'] = df_merged['label_3']
df_merged = df_merged.drop('label_3', axis=1)
```

In [18]:

```
df = df_merged.merge(df_transactions, on='customer_name').merge(df_offers, on='offer_id').drop('n', axis=1)
df.head()
```

Out[18]:

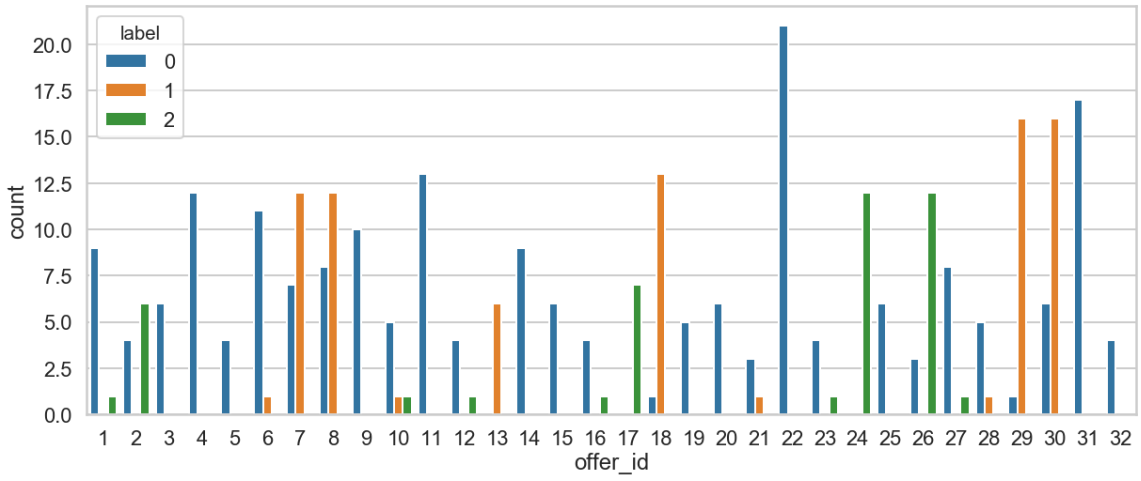
	x	y	customer_name	label	offer_id	campaign	varietal	min_qty	disc
0	1.007580	0.108215	Adams	1	18	July	Espumante	6	
1	0.346529	-0.288514	Gutierrez	1	18	July	Espumante	6	
2	1.140585	-0.029993	Hill	1	18	July	Espumante	6	
3	1.484258	-0.155233	James	1	18	July	Espumante	6	
4	1.125346	0.065148	King	1	18	July	Espumante	6	

In [19]:

```
plt.figure(figsize=(20,8))
sns.countplot(x='offer_id', hue='label', data=df)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1822f81db70>
```



From the grouped counplot above, we find some very interesting observations.

Class 0: Offer 22, 31 , 11 are most popular. **Class 1:** Offer 29, 30, 18 are most popular. **Class 2:** Offer 25,26,17 are most most popular.

Other Clustering Algorithms

k-means is only one of a ton of clustering algorithms. Below is a brief description of several clustering algorithms, and the table provides references to the other clustering algorithms in scikit-learn.

- **Affinity Propagation** does not require the number of clusters K to be known in advance! AP uses a "message passing" paradigm to cluster points based on their similarity.
- **Spectral Clustering** uses the eigenvalues of a similarity matrix to reduce the dimensionality of the data before clustering in a lower dimensional space. This is tangentially similar to what we did to visualize k-means clusters using PCA. The number of clusters must be known a priori.
- **Ward's Method** applies to hierarchical clustering. Hierarchical clustering algorithms take a set of data and successively divide the observations into more and more clusters at each layer of the hierarchy. Ward's method is used to determine when two clusters in the hierarchy should be combined into one. It is basically an extension of hierarchical clustering. Hierarchical clustering is *divisive*, that is, all observations are part of the same cluster at first, and at each successive iteration, the clusters are made smaller and smaller. With hierarchical clustering, a hierarchy is constructed, and there is not really the concept of "number of clusters." The number of clusters simply determines how low or how high in the hierarchy we reference and can be determined empirically or by looking at the dendogram (https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.cluster.hierarchy.dendrogram.html).
- **Agglomerative Clustering** is similar to hierarchical clustering but but is not divisive, it is *agglomerative*. That is, every observation is placed into its own cluster and at each iteration or level or the hierarchy, observations are merged into fewer and fewer clusters until convergence. Similar to hierarchical clustering, the constructed hierarchy contains all possible numbers of clusters and it is up to the analyst to pick the number by reviewing statistics or the dendogram.
- **DBSCAN** is based on point density rather than distance. It groups together points with many nearby neighbors. DBSCAN is one of the most cited algorithms in the literature. It does not require knowing the number of clusters a priori, but does require specifying the neighborhood size.

Clustering Algorithms in Scikit-learn

</colgroup>

</tr> </thead>

</tr>

</tr>

</tr>

</tr>

</tr>

</tr>

</tr>

</tr>

</tr> </tbody> </table> Source: http://scikit-learn.org/stable/modules/clustering.html (http://scikit-learn.org/stable/modules/clustering.html)

Exercise Set VI

Exercise: Try clustering using the following algorithms.

1. Affinity propagation
2. Spectral clustering
3. Agglomerative clustering
4. DBSCAN

How do their results compare? Which performs the best? Tell a story why you think it performs the best.

In [20]:

```
Silhouette Score is: 0.12346523604478911
Silhouette Score is: 0.10611539040197304
Silhouette Score is: 0.11625878863607858
```

In [21]:

Out[21]:

```
array([-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, -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],
      dtype=int64)
```

Affinity Propagation achieved the highest silhouette score. It is also observed that the difference in scores is extremely minimal and therefore, the 3 algorithms' performances are comparable.

In [ ]:

Method name	Parameters	Scalability	Use Case	Geometry (metric used)
K-Means	number of clusters	Very largen_samples, medium n_clusters with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium n_samples, small n_clusters	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters	Large n_samples and n_clusters	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters, linkage type, distance	Large n_samples and n_clusters	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large n_samples, medium n_clusters	Non-flat geometry, uneven cluster sizes	Distances between nearest points