

Elements Of Data Science - F2024

**Week 11: Clustering, Recommendation Systems and Imbalanced
Classes**

12/02/2024

TODOs

- Readings:
 - PDSH: Chap 3.11 Working with Time Series
 - PDSH: Chap 5.06 Example: Predicting Bicycle Traffic
 - Optional: Python for Data Analysis: Chap 11: Time Series
 - Optional: PML: Chap 9: Embedding a Machine Learning Model into a Web Application
- Quiz 10: due Monday Dec 9th, 11:59pm ET
- HW4: due Monday Dec 16th 11:59pm ET

Today

- Clustering
- Recommendation Systems
- Imbalanced Data

Questions?

Environment Setup

Clustering

- Can we group our data based on the features alone?
- **Unsupervised:** There is no label/target y
- Use similarity to group X into k clusters

Why do Clustering?

- Exploratory data analysis
- Group media: images, music, news articles,...
- Group people: social network
- Science applications: gene families, psychological groups,...
- Image segmentation: group pixels, regions, ...
- ...

Clustering Methods

- k-Means
- **Heirarchical Agglomerative Clustering**
- Spectral Clustering
- DBScan
- ...

Clustering: k -Means

- Not to be confused with k-NN!
- Idea:
 - Finds k points in space as cluster centers (means)
 - Assigns datapoints to their closest cluster mean
- Need to specify the number of clusters k up front
- sklearn uses euclidean distance to judge similarity

***k*-Means: How it works**

k -Means: How it works

FIRST: choose initial k points (means), randomly.

A: fix means \rightarrow assign all datapoints to their closest mean

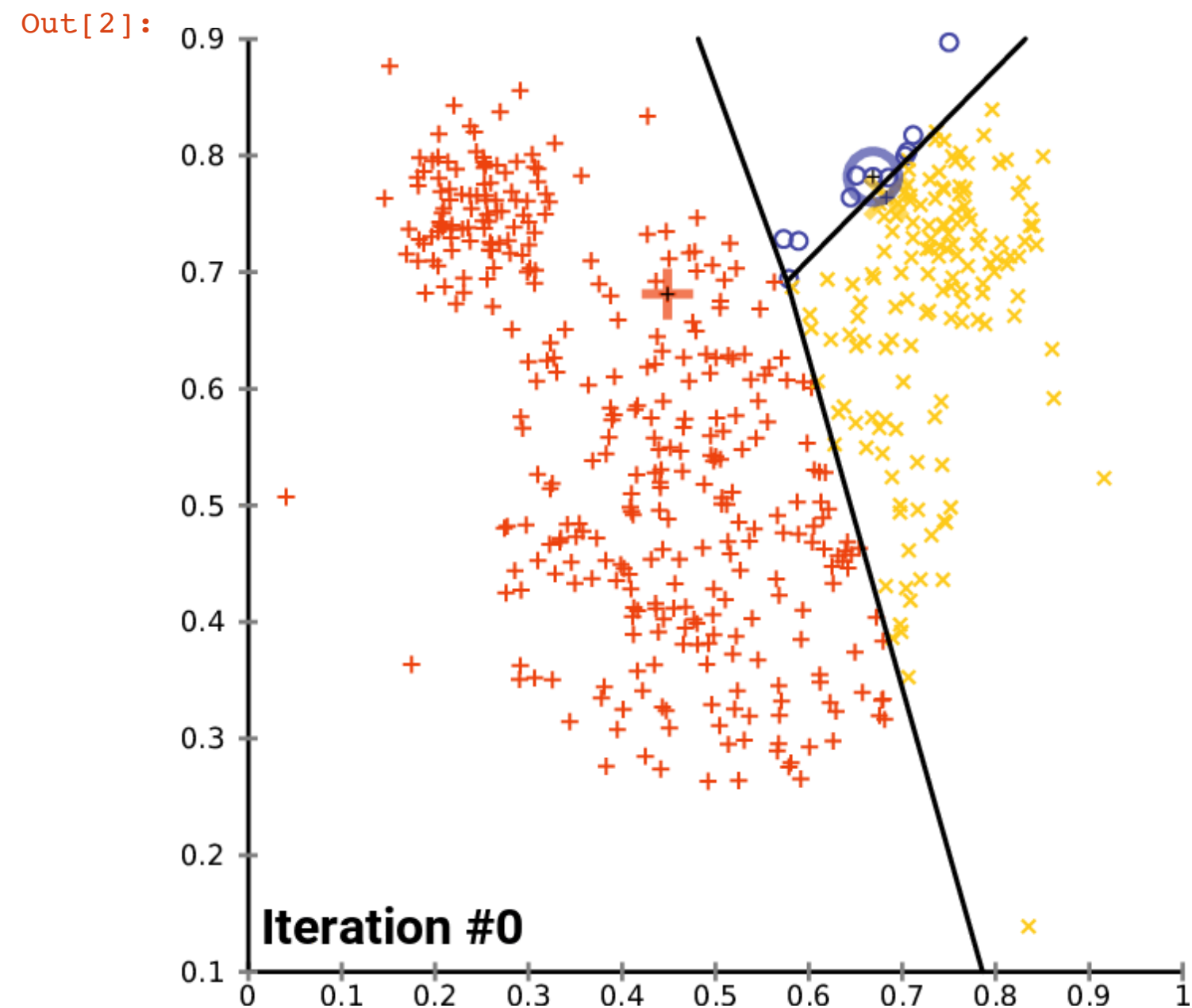
B: fix cluster assignments \rightarrow recalculate means

RETURN TO A and Repeat until convergence!

k -Means: How it works

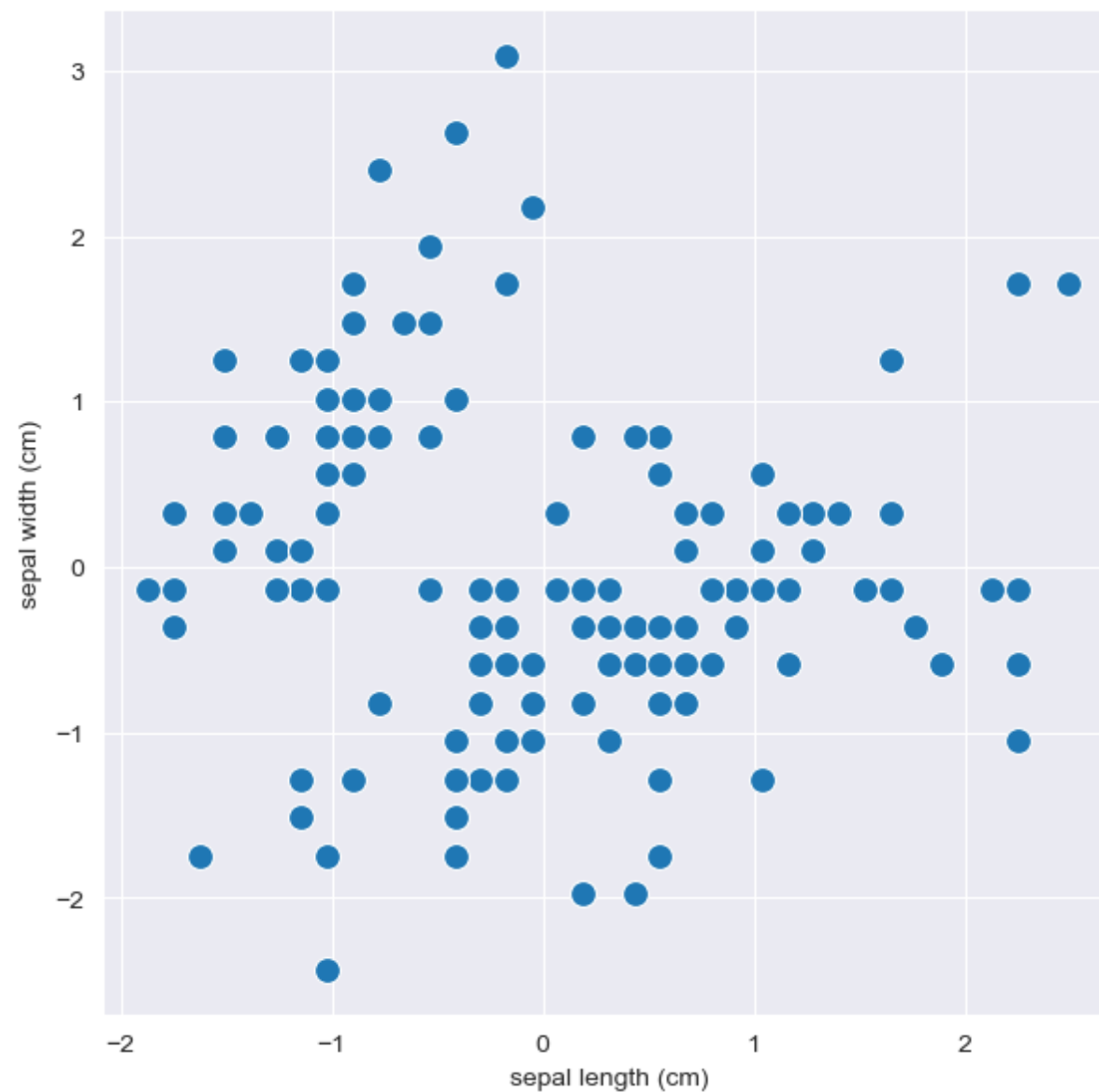
k -Means: How it works

```
In [2]: 1  
2 from IPython.display import Image  
3 Image(url='https://upload.wikimedia.org/wikipedia/commons/e/ea/K-means_convergence.gif')  
4
```



Load Example Data

```
In [3]: 1 from sklearn.datasets import load_iris
2 from sklearn.preprocessing import StandardScaler
3 iris = load_iris()
4 X_iris = StandardScaler().fit_transform(iris.data[:, :2])
5 X_iris = pd.DataFrame(X_iris, columns=iris.feature_names[:2])
6
7 fig, ax = plt.subplots(1, 1, figsize=(7, 7))
8 sns.scatterplot(x='sepal length (cm)', y='sepal width (cm)', data=X_iris, s=100);
```



KMeans in sklearn

KMeans in sklearn

```
In [4]: 1 from sklearn.cluster import KMeans
        2
        3 km = KMeans(n_clusters=2, init='random', random_state=0) # default init=k-means++
        4
        5 c = km.fit_predict(X_iris)
```

KMeans in sklearn

```
In [4]: 1 from sklearn.cluster import KMeans
        2
        3 km = KMeans(n_clusters=2, init='random', random_state=0) # default init=k-means++
        4
        5 c = km.fit_predict(X_iris)
```

```
In [5]: 1 # cluster assignments
        2 tmp = X_iris.copy()
        3 tmp['cluster_assignments'] = c
        4 tmp.sample(5, random_state=0).round(2)
```

Out[5]:

	sepal length (cm)	sepal width (cm)	cluster_assignments
114	-0.05	-0.59	1
62	0.19	-1.97	1
33	-0.42	2.63	0
107	1.77	-0.36	1
7	-1.02	0.79	0

KMeans in sklearn

```
In [4]: 1 from sklearn.cluster import KMeans
        2
        3 km = KMeans(n_clusters=2, init='random', random_state=0) # default init=k-means++
        4
        5 c = km.fit_predict(X_iris)
```

```
In [5]: 1 # cluster assignments
        2 tmp = X_iris.copy()
        3 tmp['cluster_assignments'] = c
        4 tmp.sample(5, random_state=0).round(2)
```

Out[5]:

	sepal length (cm)	sepal width (cm)	cluster_assignments
114	-0.05	-0.59	1
62	0.19	-1.97	1
33	-0.42	2.63	0
107	1.77	-0.36	1
7	-1.02	0.79	0

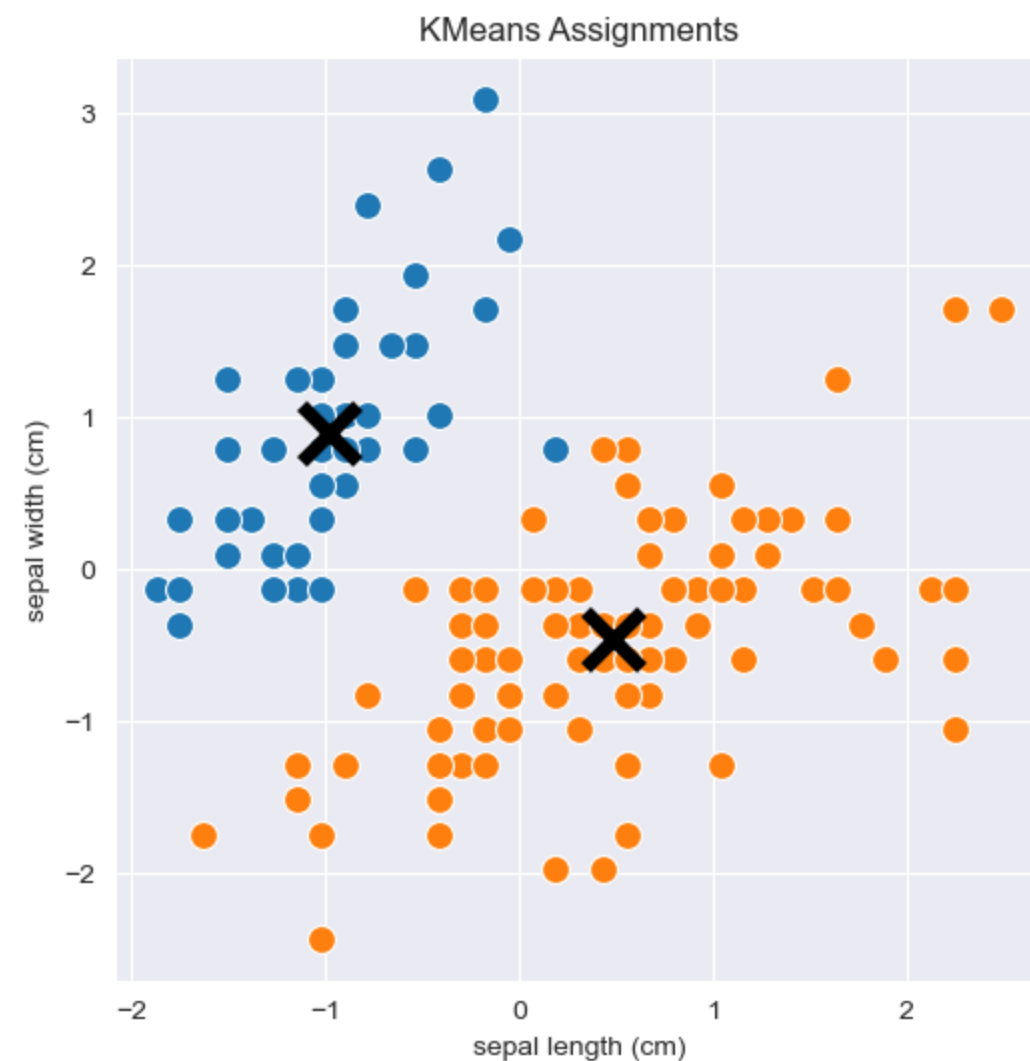
```
In [6]: 1 # cluster centers
        2 km.cluster_centers_.round(2)
```

Out[6]: array([[-0.98, 0.9],
 [0.49, -0.45]])

Plotting clusters and centers

Plotting clusters and centers

```
In [7]: 1 def plot_clusters(X,c=None,km=None,title=None,ax=None,marker_size=100):
2         '''Plot data colored by cluster assignment'''
3         _,ax = plt.subplots(1,1,figsize=(6,6)) if ax is None else (None,ax)
4         c = km.fit_predict(X) if c is None else c
5         for i in range(np.max(c)+1):
6             sns.scatterplot(x=X.loc[c == i].iloc[:,0],y=X.loc[c == i].iloc[:,1],s=marker_size,ax=ax);
7             if km:
8                 ax.plot(km.cluster_centers_[i,0],km.cluster_centers_[i,1], marker='x',c='k', ms=20, mew=5)
9         ax.set_title(title)
10
11 plot_clusters(X_iris,km=km,title="KMeans Assignments")
```



K-Means: How good are the clusters?

- One way: **Within Cluster Sum of Squared Distances (SSD)**
- How close is every point to it's assigned cluster center?

$$SSD = \sum_{k=1}^K \sum_{x_i \in C_k} ||x_i - \mu_k||_2^2$$

$$\text{where } ||x - \mu||_2 = \sqrt{\sum_{j=1}^d (x_j - \mu_j)^2}$$

- If this is high, items in cluster are far from their means.
- If this is low, items in cluster are close to their means.

K-Means: How good are the clusters?

- One way: **Within Cluster Sum of Squared Distances (SSD)**
- How close is every point to it's assigned cluster center?

$$SSD = \sum_{k=1}^K \sum_{x_i \in C_k} ||x_i - \mu_k||_2^2$$

$$\text{where } ||x - \mu||_2 = \sqrt{\sum_{j=1}^d (x_j - \mu_j)^2}$$

- If this is high, items in cluster are far from their means.
- If this is low, items in cluster are close to their means.

```
In [11]: 1 # SSD stored in KMeans as `.inertia_`  
         2 round(km.inertia_,2)
```

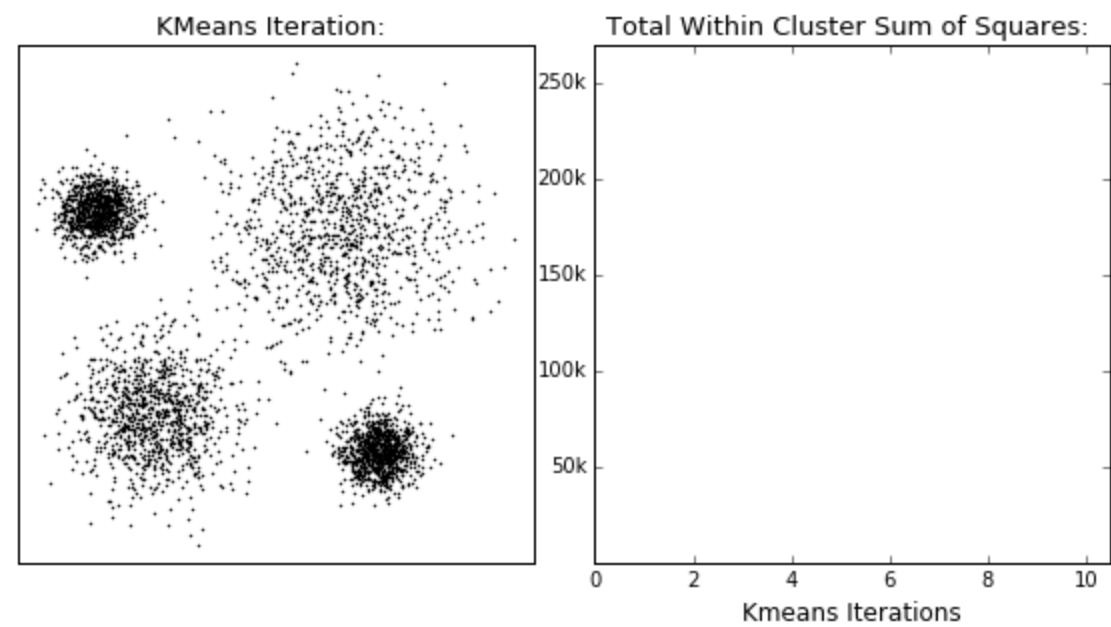
```
Out[11]: 166.95
```

KMeans in Action

KMeans in Action

```
In [12]: 1 Image(url='https://dashee87.github.io/images/kmeans.gif')  
        2
```

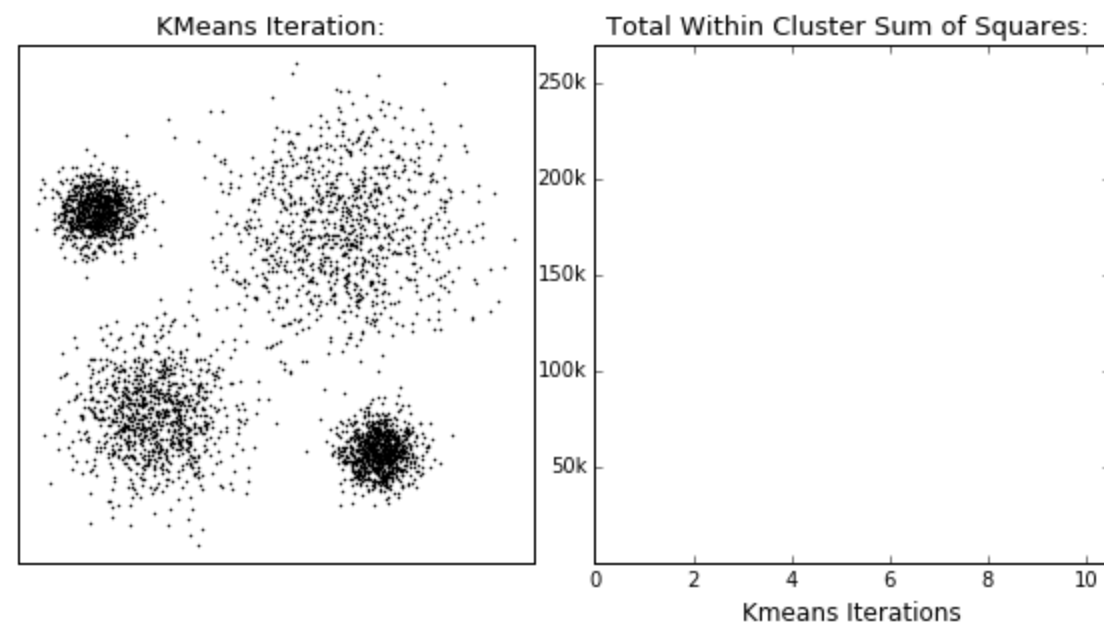
Out[12]:



KMeans in Action

```
In [12]: 1 Image(url='https://dashee87.github.io/images/kmeans.gif')  
        2
```

Out[12]:



From <https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/>

Things you need to define for KMeans

- number of clusters k or `n_clusters`
- initial locations of means
 - random
 - k-means++ (pick starting points far apart from each other)

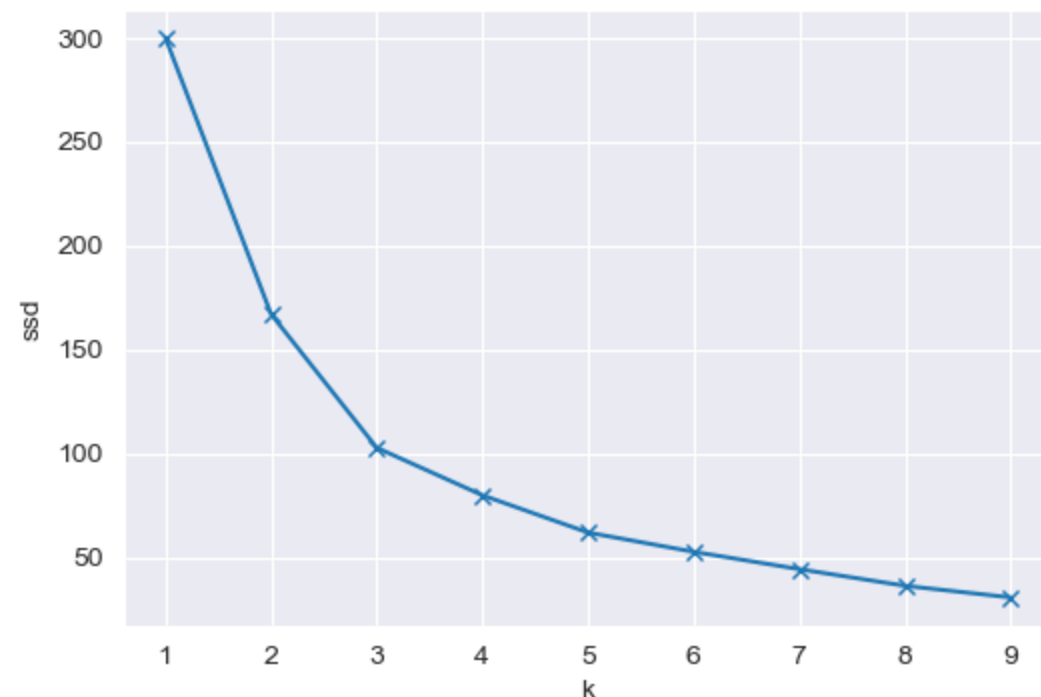
How to choose k or `n_clusters`?

- One way: use "elbow" in SSD or `KMeans.inertia_`
- "elbow" is where SSD ceases to drop rapidly

How to choose k or `n_clusters`?

- One way: use "elbow" in SSD or `KMeans.inertia_`
- "elbow" is where SSD ceases to drop rapidly

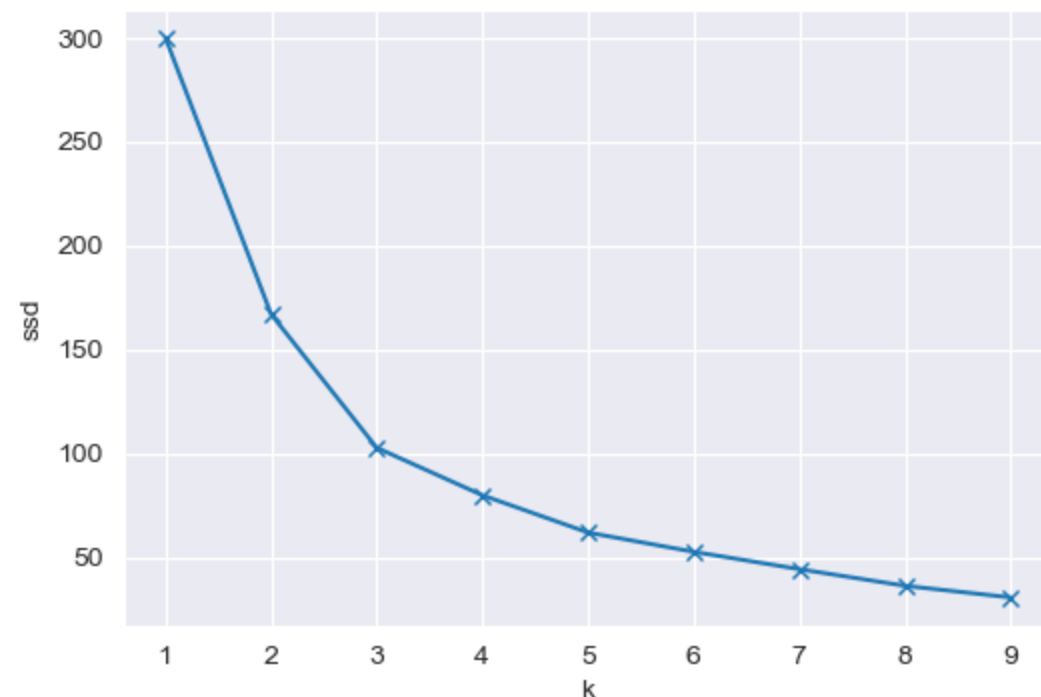
```
In [13]: 1 ssd = []
          2 for i in range(1,10):
          3     ssd.append(KMeans(n_clusters=i).fit(X_iris).inertia_)
          4 fig,ax=plt.subplots(1,1,figsize=(6,4))
          5 ax.plot(range(1,10),ssd,marker='x');
          6 ax.set_xlabel('k');ax.set_ylabel('ssd');
```



How to choose k or `n_clusters`?

- One way: use "elbow" in SSD or `KMeans.inertia_`
- "elbow" is where SSD ceases to drop rapidly

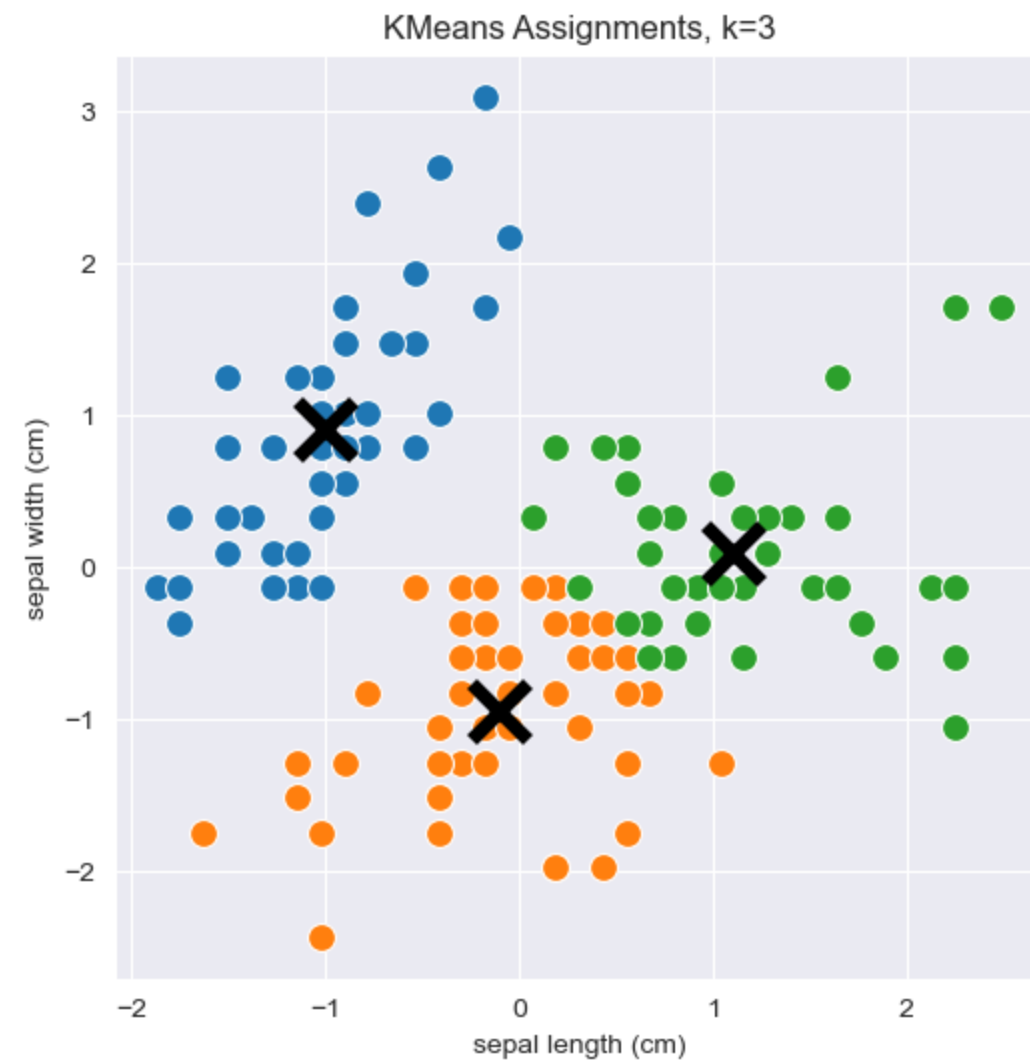
```
In [13]: 1 ssd = []  
2 for i in range(1,10):  
3     ssd.append(KMeans(n_clusters=i).fit(X_iris).inertia_)  
4 fig,ax=plt.subplots(1,1,figsize=(6,4))  
5 ax.plot(range(1,10),ssd,marker='x');  
6 ax.set_xlabel('k');ax.set_ylabel('ssd');
```



Refitting with $k=3$

Refitting with k=3

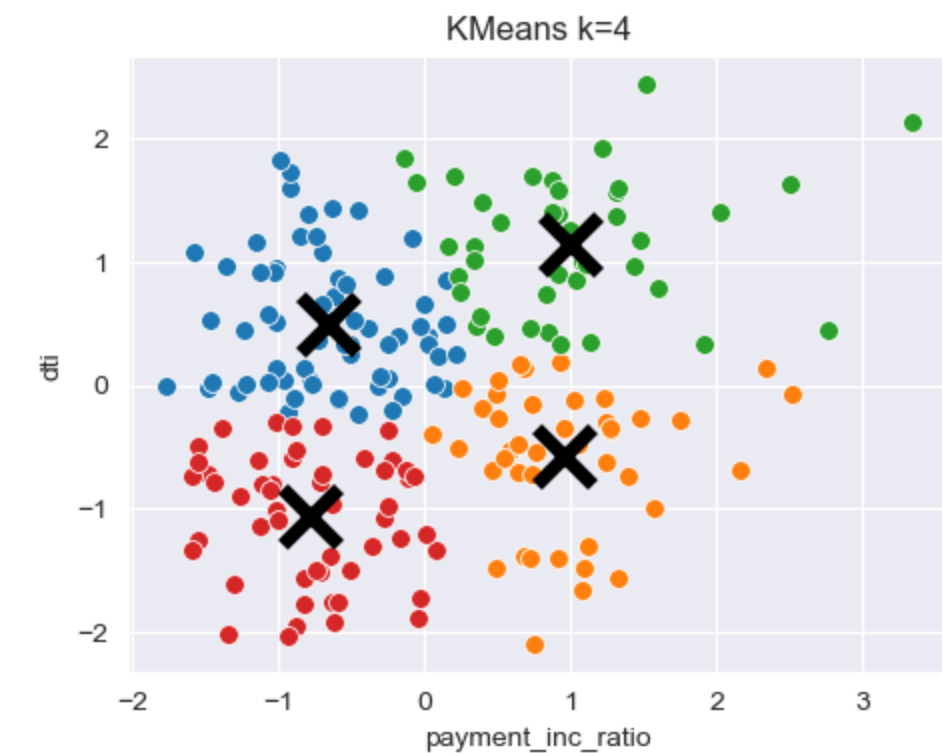
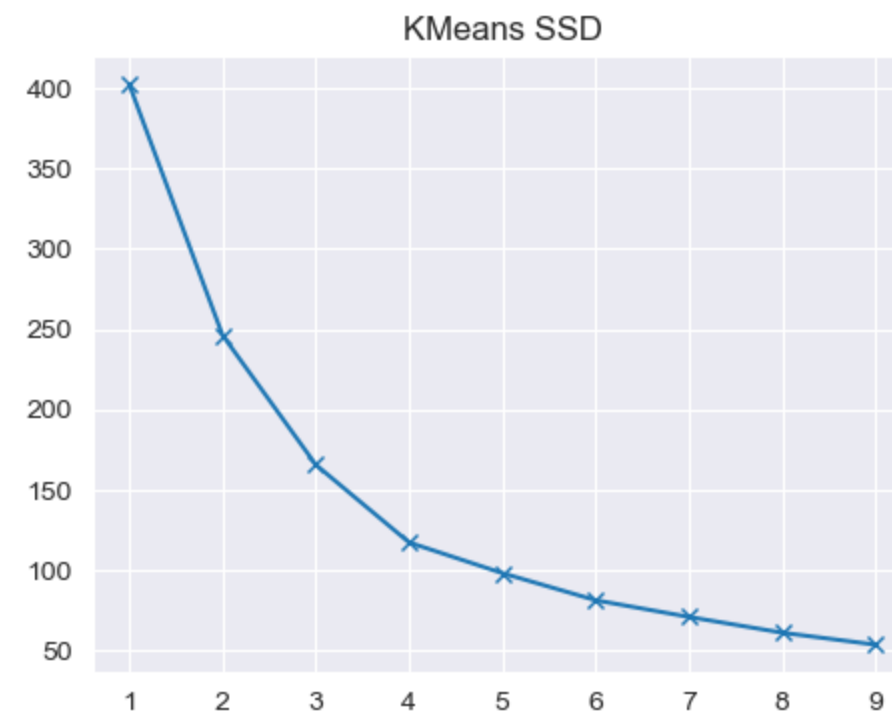
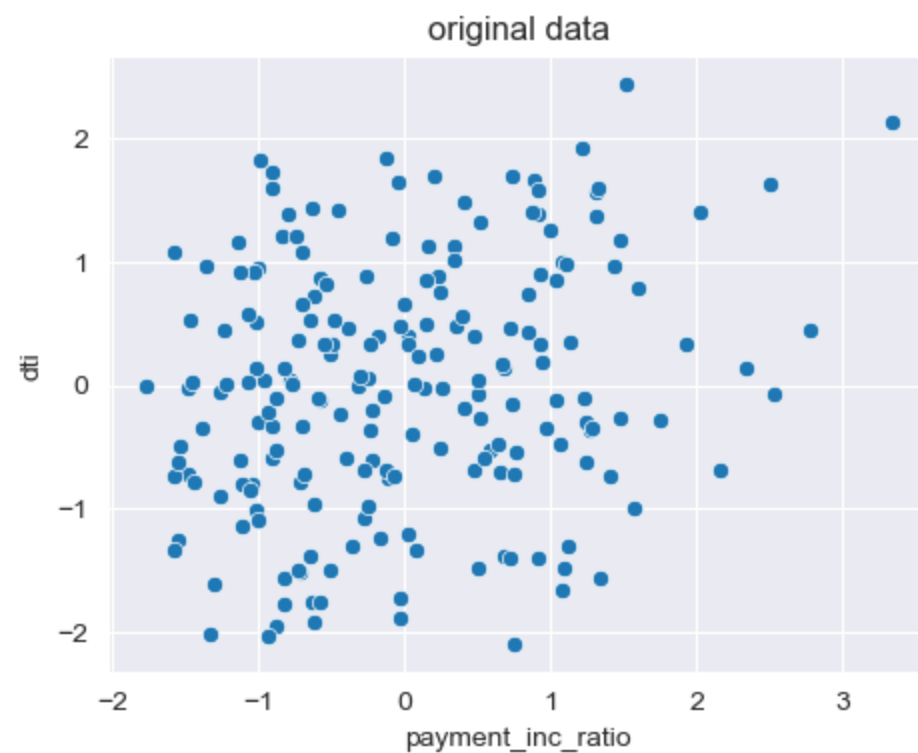
```
In [14]: 1 plot_clusters(X_iris, km=KMeans(n_clusters=3, random_state=0), title="KMeans Assignments, k=3")
```



KMeans: Another Example

KMeans: Another Example

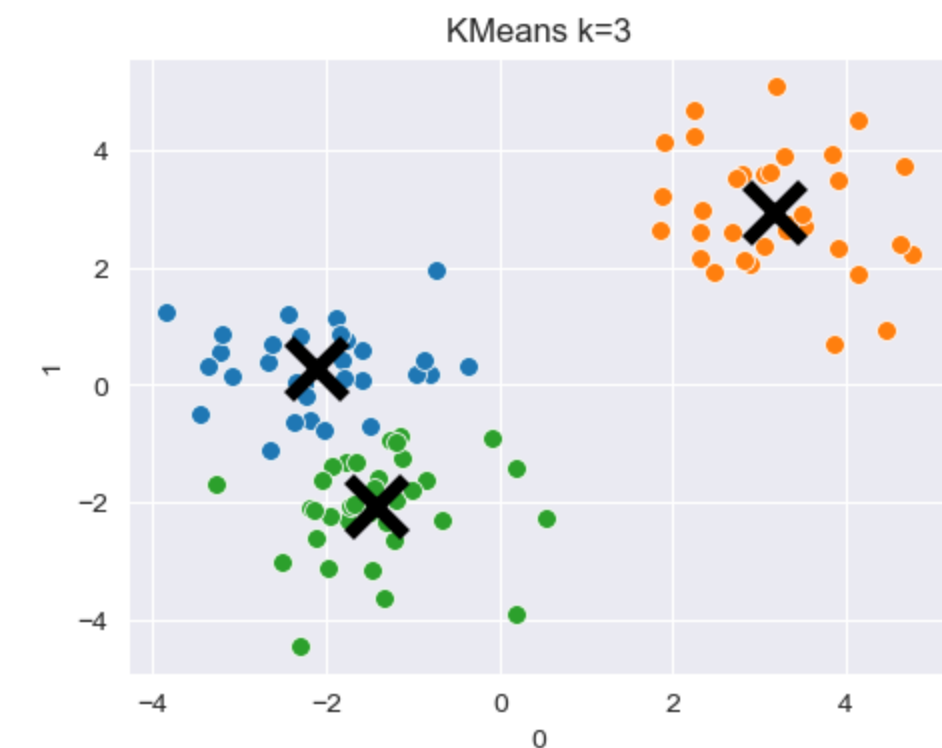
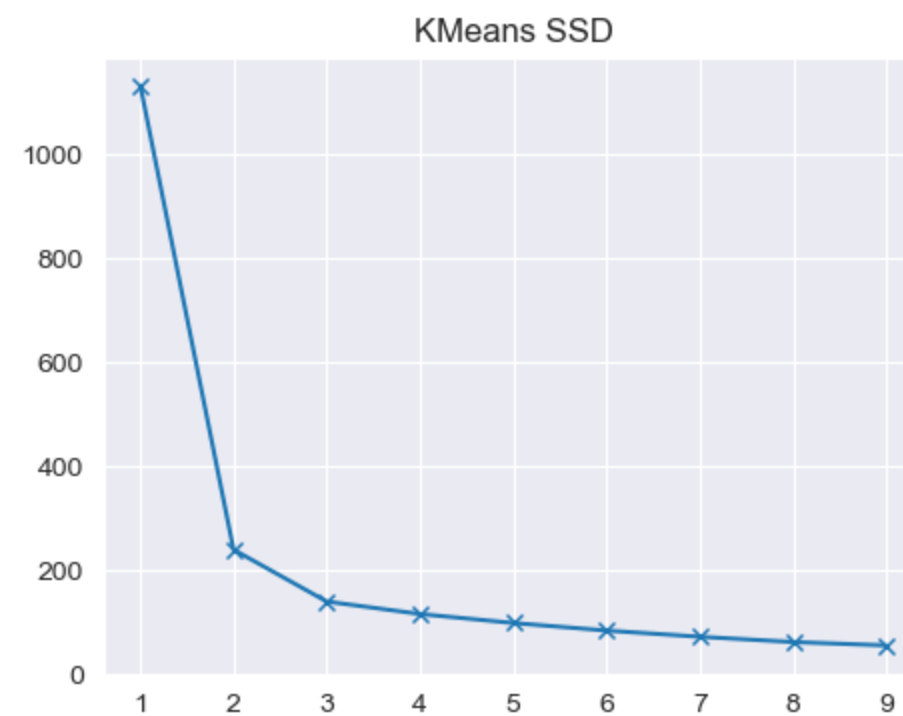
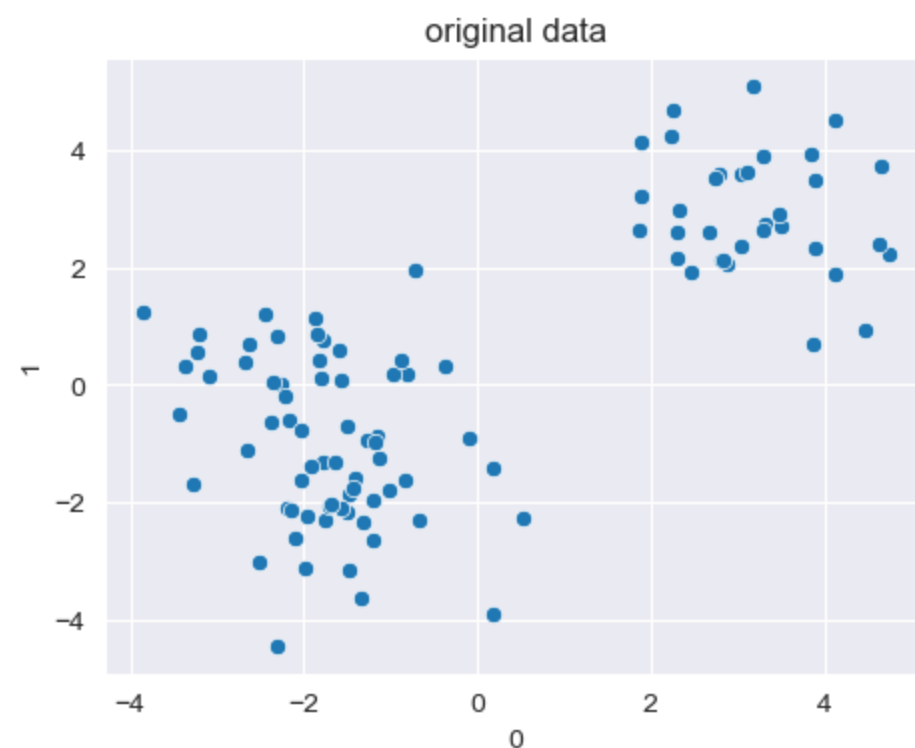
```
In [16]: 1 # loading and plotting the data
2 data = pd.read_csv('../data/loan200.csv')[['payment_inc_ratio','dti']]
3 from sklearn.preprocessing import StandardScaler
4 X_loan = pd.DataFrame(StandardScaler().fit_transform(data),columns=data.columns)
5
6 fig,ax = plt.subplots(1,3,figsize=(18,4))
7 sns.scatterplot(x=X_loan.iloc[:,0],y=X_loan.iloc[:,1],ax=ax[0]);
8 ax[0].set_title('original data');
9
10 ssd = [KMeans(n_clusters=i).fit(X_loan).inertia_ for i in range(1,10)]
11 ax[1].plot(range(1,10),ssd,marker='x');
12 ax[1].set_title('KMeans SSD');
13
14 plot_clusters(X_loan,km=KMeans(n_clusters=4, random_state=0),title='KMeans k=4',marker_size=50,ax=ax[2])
```



KMeans: Synthetic Example

KMeans: Synthetic Example

```
In [17]: 1 from sklearn.datasets import make_blobs
2 X_blobs,y_blobs = make_blobs(centers=[(3,3),(-2,0),(-2,-2)],random_state=1)
3 X_blobs = pd.DataFrame(X_blobs)
4
5 fig,ax = plt.subplots(1,3,figsize=(18,4))
6
7 sns.scatterplot(x=X_blobs.iloc[:,0],y=X_blobs.iloc[:,1],ax=ax[0]);
8 ax[0].set_title('original data');
9
10 ssd = [KMeans(n_clusters=i).fit(X_blobs).inertia_ for i in range(1,10)]
11 ax[1].plot(range(1,10),ssd,marker='x');
12 ax[1].set_title('KMeans SSD')
13
14 plot_clusters(X_blobs,km=KMeans(n_clusters=3, random_state=0),title='KMeans k=3',marker_size=50,ax=ax[2])
```



Hierarchical Agglomerative Clustering (HAC)

- group clusters together from the bottom up
- don't have to specify number of clusters up front
- generates binary tree over data

HAC: How it works

FIRST: every point is it's own cluster

A: Find pair of clusters that are "closest"

B: Merge into single cluster

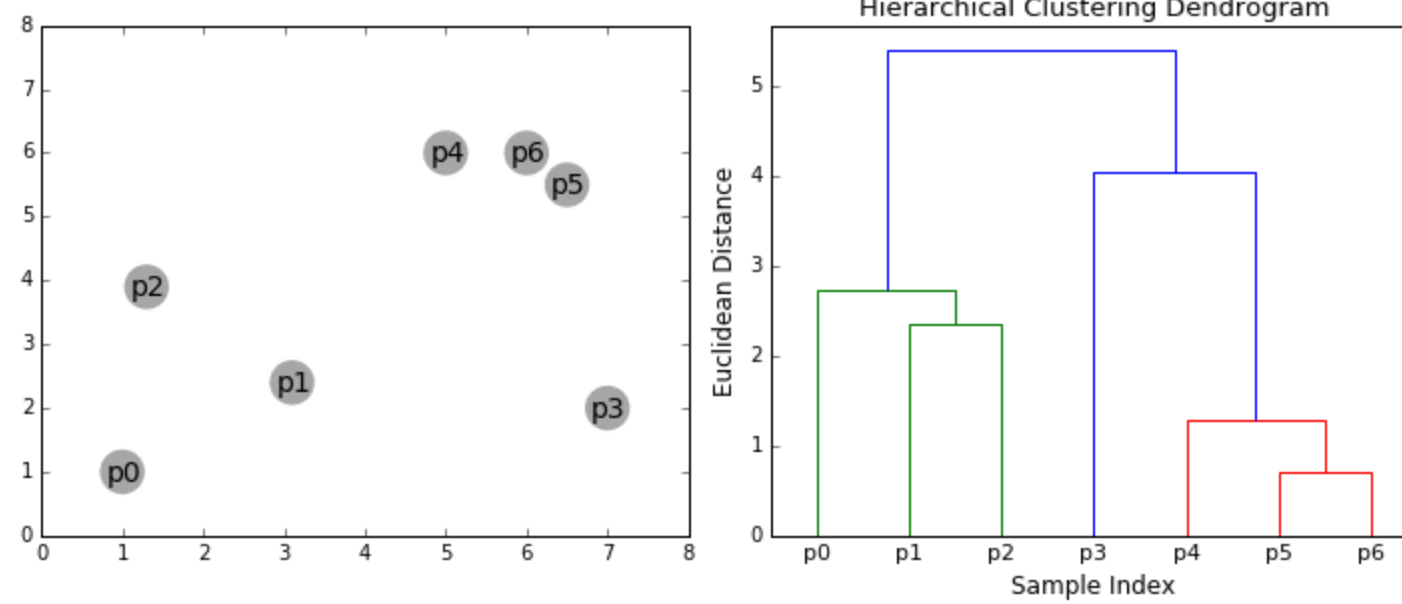
GOTO A and Repeat till there is a single cluster

HAC in Action

HAC in Action

```
In [18]: 1 Image(url='https://dashee87.github.io/images/hierarch.gif')
```

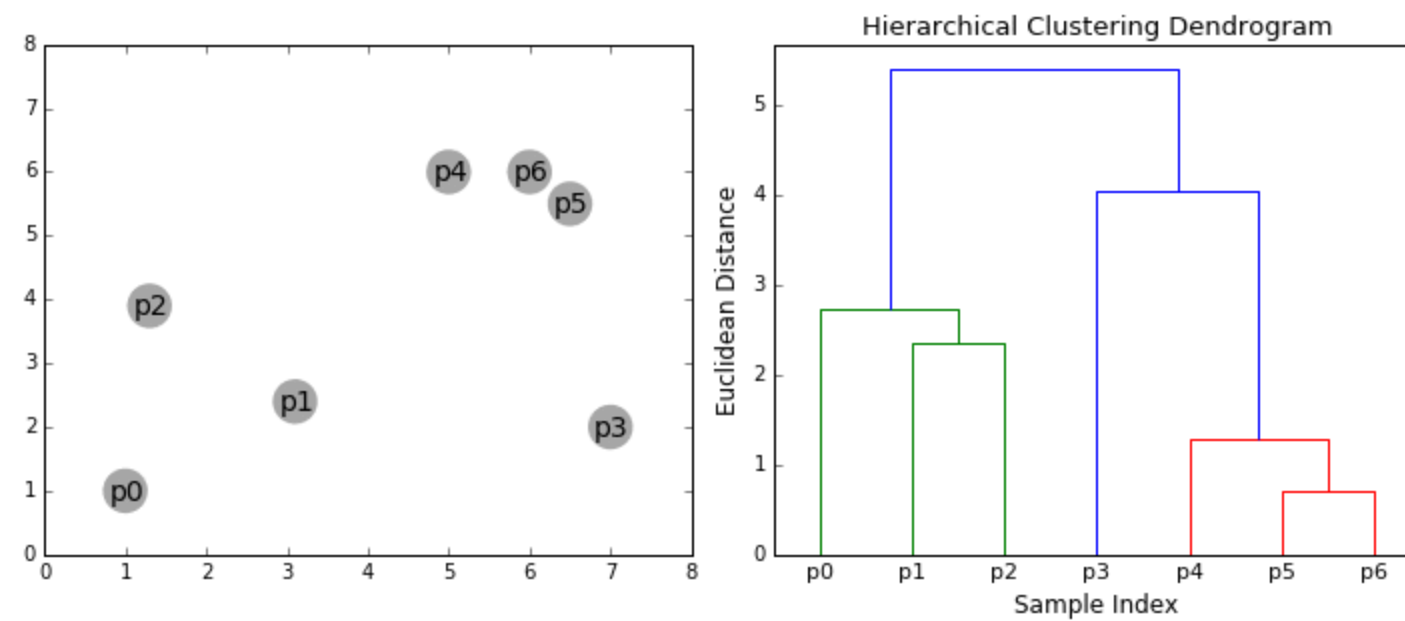
Out[18]:



HAC in Action

```
In [18]: 1 Image(url='https://dashee87.github.io/images/hierarch.gif')
```

Out[18]:



From <https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/>

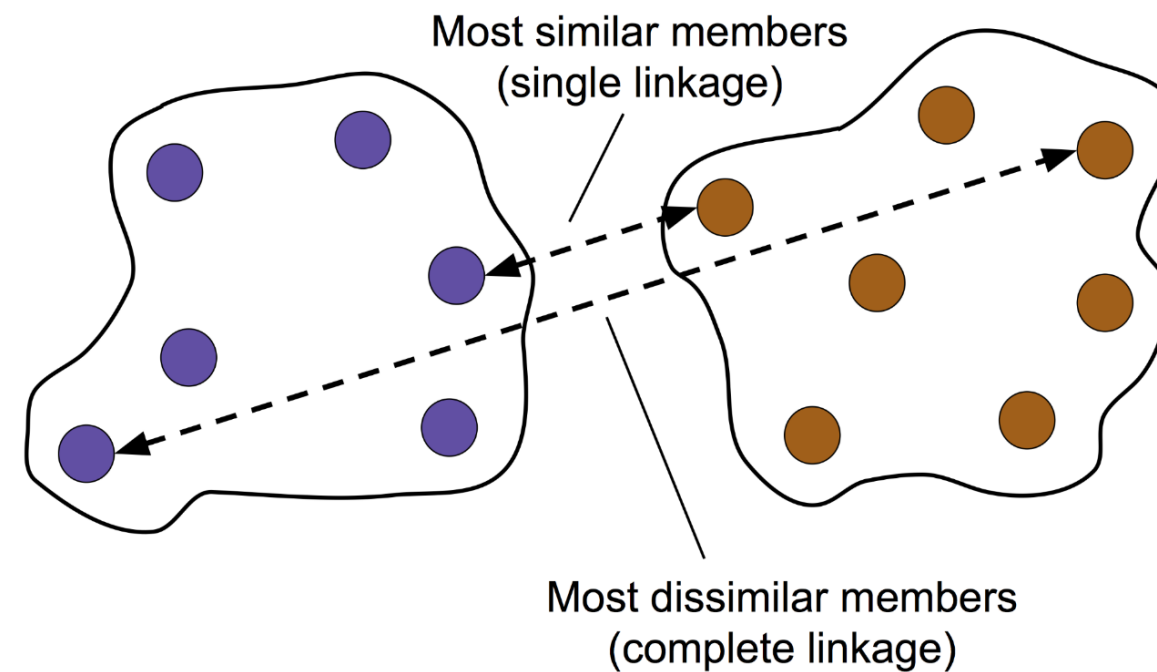
What is "close"?

- Need to define what we mean by "closeness" by choosing
 - distance metric (how to measure distance)
 - linkage criteria (how to compare clusters)

Need to define: Distance Metric

- Euclidean : $\sqrt{\sum_{i=1}^n (a_i - b_i)^2}$
 - easy to use analytically, sensitive to outliers
- Manhattan : $\sum_{i=1}^n |a_i - b_i|$
 - more difficult to use analytically, robust to outliers
- Cosine : $1 - \frac{\sum a_i b_i}{\|a_i\|_2 \|b_i\|_2}$
 - angle between vectors while ignoring their scale
- many more (see <https://numerics.mathdotnet.com/Distance.html>)

Need to define: Linkage



single : shortest distance from item of one cluster to item of the other

complete : greatest distance from item of one cluster to item of the other

average : average distance of items in one cluster to items in the other

ward : minimize variance of clusters being merged (only euclidean metric)

HAC in sklearn

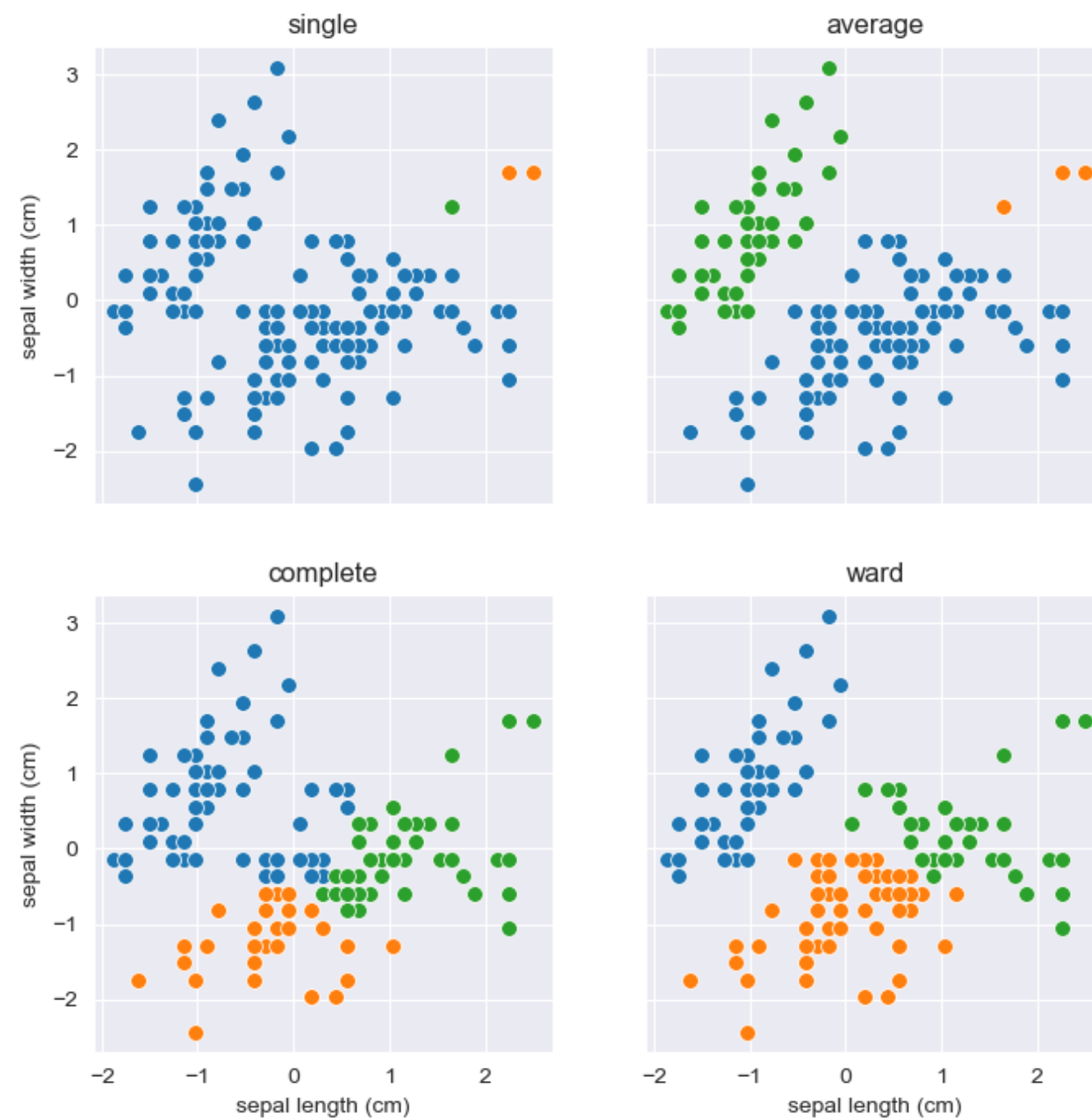
HAC in sklearn

```
In [19]: 1 from sklearn.cluster import AgglomerativeClustering
2
3 hac = AgglomerativeClustering(linkage='single',      # ward by default
4                               affinity='euclidean',  # default
5                               n_clusters=4)         # 2 by default
6 c_single = hac.fit_predict(X_iris)
7
8 # generate models and assignments for all linkages
9 models,assignments = [],[]
10 linkages = ['single','average','complete','ward']
11 for linkage in linkages:
12     models.append(AgglomerativeClustering(linkage=linkage,affinity='euclidean',n_clusters=3))
13     assignments.append(models[-1].fit_predict(X_iris))
14
15 # plot on the next slide
```

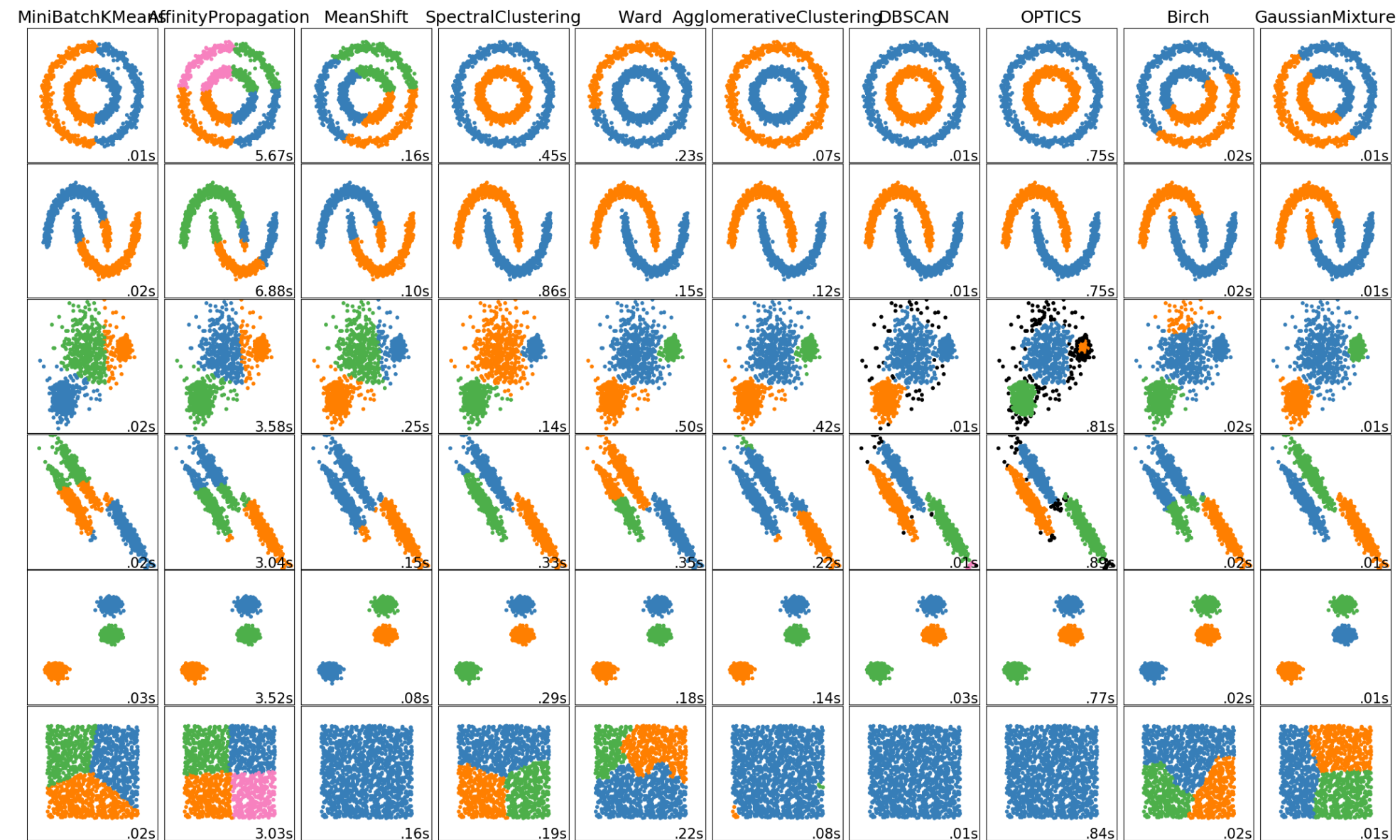
HAC in sklearn

HAC in sklearn

```
In [20]: 1 fig,ax = plt.subplots(2,2,figsize=(8,8),sharex=True,sharey=True)
2         axs = ax.flatten()
3         for i in range(len(linkage)):
4             plot_clusters(X_iris,c=assignments[i],title=linkages[i],ax=axs[i],marker_size=50)
```



Clustering: Many Other Methods



From <https://scikit-learn.org/stable/modules/clustering.html>

How to evaluate clustering?

- **Within Cluster Sum of Squared Distances (SSD)**
- If we have labels:
 - **Homogeneity:** each cluster contains only members of a single class
 - **Completeness:** all members of a given class are assigned to the same cluster
 - **V-score:** harmonic mean of Homogeneity and Completeness
- Silhouette plots (see PML)
- many others ([see sklearn](#))

Clustering Review

- k-Means
- Heirarchical Agglomerative Clustering
 - linkages
 - distance metrics
- Evaluating

Questions re Clustering?

Recommendation Engines

- Given a user and a set of items to recommend (or rank):
 - **Content-Based Filtering:** Recommend things **similar to the things I've liked**
 - **Collaborative Filtering:** Recommend things **that people with similar tastes have liked**
 - Hybrid/Ensemble
 - Recommendation as Classification

Example: Housing Data

Example: Housing Data

```
In [21]: 1 df_house = pd.read_csv('../data/house_sales_subset.csv')
2 df_house = df_house.iloc[:10].loc[:, ['SqFtTotLiving', 'SqFtLot', 'AdjSalePrice']]
3 X_house_scaled = StandardScaler().fit_transform(df_house)
4 df_house_scaled = pd.DataFrame(X_house_scaled, columns=['SqFtTotLiving_scaled', 'SqFtLot_scaled', 'AdjSalePrice_scaled'])
5 df_house_scaled.head().round(2)
```

Out[21]:

	SqFtTotLiving_scaled	SqFtLot_scaled	AdjSalePrice_scaled
0	0.40	-0.47	-0.70
1	2.03	0.65	2.48
2	-0.01	1.26	1.19
3	1.36	-0.54	-0.12
4	-0.41	-0.54	-0.71

Content-Based Filtering

- Find other things similar to the things I've liked
 - Assume: If I like product A, and product B is like product A, I'll like product B
 - Use similarity of items
-
- Matrix: items x items
 - Values: Similarity of items

Calculate Distances

- to maximize similarity \rightarrow minimize distance

Calculate Distances

- to maximize similarity → minimize distance

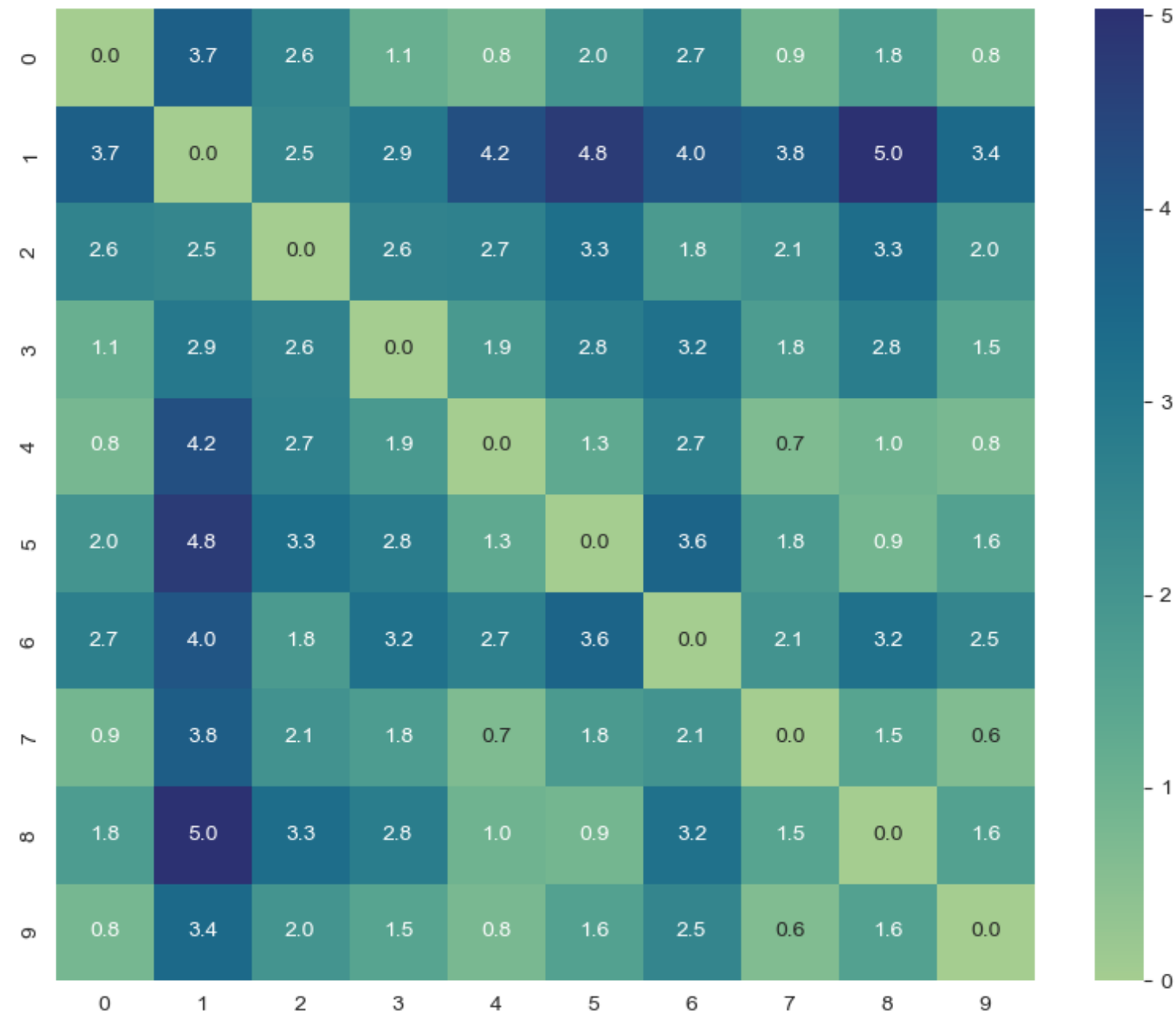
```
In [22]: 1 # using euclidean distance
          2 from sklearn.metrics.pairwise import euclidean_distances
          3
          4 # calculate all pairwise distances between houses
          5 dists = euclidean_distances(X_house_scaled)
          6
          7 np.round(dists,2)
```

```
Out[22]: array([[0.   , 3.74, 2.59, 1.12, 0.82, 2.01, 2.73, 0.87, 1.76, 0.84],
                [3.74, 0.   , 2.49, 2.94, 4.19, 4.78, 4.01, 3.79, 5.03, 3.44],
                [2.59, 2.49, 0.   , 2.61, 2.65, 3.25, 1.83, 2.07, 3.31, 2.01],
                [1.12, 2.94, 2.61, 0.   , 1.87, 2.83, 3.19, 1.76, 2.8 , 1.47],
                [0.82, 4.19, 2.65, 1.87, 0.   , 1.32, 2.69, 0.68, 0.97, 0.78],
                [2.01, 4.78, 3.25, 2.83, 1.32, 0.   , 3.59, 1.81, 0.87, 1.61],
                [2.73, 4.01, 1.83, 3.19, 2.69, 3.59, 0.   , 2.05, 3.2 , 2.51],
                [0.87, 3.79, 2.07, 1.76, 0.68, 1.81, 2.05, 0.   , 1.5 , 0.64],
                [1.76, 5.03, 3.31, 2.8 , 0.97, 0.87, 3.2 , 1.5 , 0.   , 1.61],
                [0.84, 3.44, 2.01, 1.47, 0.78, 1.61, 2.51, 0.64, 1.61, 0.   ]])
```

Visualizing Distances With a Heatmap

Visualizing Distances With a Heatmap

```
In [23]: 1 fig,ax = plt.subplots(1,1,figsize=(10,8))
          2 sns.heatmap(dists,annot=True,fmt=".1f",cmap='crest');
```



Query For Similarity

- Imagine I like house 5
- What houses are similar to house 5?

Query For Similarity

- Imagine I like house 5
- What houses are similar to house 5?

```
In [24]: 1 query_idx = 5  
        2 df_house.iloc[query_idx]
```

```
Out[24]: SqFtTotLiving      930.0  
         SqFtLot          1012.0  
         AdjSalePrice     411781.0  
         Name: 5, dtype: float64
```

Query For Similarity

- Imagine I like house 5
- What houses are similar to house 5?

```
In [24]: 1 query_idx = 5
         2 df_house.iloc[query_idx]
```

```
Out[24]: SqFtTotLiving      930.0
         SqFtLot           1012.0
         AdjSalePrice      411781.0
         Name: 5, dtype: float64
```

```
In [25]: 1 # Distances to house 5
         2 [f'{x:0.1f}' for x in dists[query_idx]]
```

```
Out[25]: ['2.0', '4.8', '3.3', '2.8', '1.3', '0.0', '3.6', '1.8', '0.9', '1.6']
```

Query For Similarity Cont.

Query For Similarity Cont.

```
In [26]: 1 # find indexes of best scores (for distances, want ascending)
          2 best_idx asc = np.argsort(dists[query_idx])
          3 best_idx asc
```

```
Out[26]: array([5, 8, 4, 9, 7, 0, 3, 2, 6, 1])
```

Query For Similarity Cont.

```
In [26]: 1 # find indexes of best scores (for distances, want ascending)
2 best_idx asc = np.argsort(dists[query_idx])
3 best_idx asc
```




















```
Out[26]: array([5, 8, 4, 9, 7, 0, 3, 2, 6, 1])
```

```
In [27]: 1 # the top 10 recommendations with their distances
2 list(zip(['house ' + str(x) for x in best_idx asc],
3         np.round(dists[query_idx][best_idx asc], 2)
4         ))
5
```

```
Out[27]: [('house 5', 0.0),
('house 8', 0.87),
('house 4', 1.32),
('house 9', 1.61),
('house 7', 1.81),
('house 0', 2.01),
('house 3', 2.83),
('house 2', 3.25),
('house 6', 3.59),
('house 1', 4.78)]
```

(User Based) Collaborative Filtering

- Recommend things that people with similar tastes have liked
- Assume: If both you and I like Movie A, and you like Movie B, I'll like movie B
- Use similarity of user preferences
- Matrix: Users x Items
- Values: Rankings

					
A					
B					
C					
D					
E					

Example: User Interests

Can we recommend topics based on a users existing interests?

Example: User Interests

Can we recommend topics based on a users existing interests?

```
In [28]: 1 # from Data Science from Scratch by Joel Grus
          2 #https://github.com/joelgrus/data-science-from-scratch.git
          3
          4 users_interests = [
          5     ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandra"],
          6     ["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"],
          7     ["Python", "scikit-learn", "scipy", "numpy", "statsmodels", "pandas"],
          8     ["R", "Python", "statistics", "regression", "probability"],
          9     ["machine learning", "regression", "decision trees", "libsvm"],
         10     ["Python", "R", "Java", "C++", "Haskell", "programming languages"],
         11     ["statistics", "probability", "mathematics", "theory"],
         12     ["machine learning", "scikit-learn", "Mahout", "neural networks"],
         13     ["neural networks", "deep learning", "Big Data", "artificial intelligence"],
         14     ["Hadoop", "Java", "MapReduce", "Big Data"],
         15     ["statistics", "R", "statsmodels"],
         16     ["C++", "deep learning", "artificial intelligence", "probability"],
         17     ["pandas", "R", "Python"],
         18     ["databases", "HBase", "Postgres", "MySQL", "MongoDB"],
         19     ["libsvm", "regression", "support vector machines"]
         20 ]
```

Example: User Interests

Can we recommend topics based on a users existing interests?

```
In [28]: 1 # from Data Science from Scratch by Joel Grus
2 #https://github.com/joelgrus/data-science-from-scratch.git
3
4 users_interests = [
5     ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandra"],
6     ["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"],
7     ["Python", "scikit-learn", "scipy", "numpy", "statsmodels", "pandas"],
8     ["R", "Python", "statistics", "regression", "probability"],
9     ["machine learning", "regression", "decision trees", "libsvm"],
10    ["Python", "R", "Java", "C++", "Haskell", "programming languages"],
11    ["statistics", "probability", "mathematics", "theory"],
12    ["machine learning", "scikit-learn", "Mahout", "neural networks"],
13    ["neural networks", "deep learning", "Big Data", "artificial intelligence"],
14    ["Hadoop", "Java", "MapReduce", "Big Data"],
15    ["statistics", "R", "statsmodels"],
16    ["C++", "deep learning", "artificial intelligence", "probability"],
17    ["pandas", "R", "Python"],
18    ["databases", "HBase", "Postgres", "MySQL", "MongoDB"],
19    ["libsvm", "regression", "support vector machines"]
20 ]
```

```
In [29]: 1 # interests of user0
2 sorted(users_interests[0])
```

```
Out[29]: ['Big Data', 'Cassandra', 'HBase', 'Hadoop', 'Java', 'Spark', 'Storm']
```

All Unique Interests

All Unique Interests

```
In [30]: 1 # get a sorted list of unique interests (here using set)
2 unique_interests = sorted({interest
3                             for user_interests in users_interests
4                             for interest in user_interests})
5
6 # the first 20 unique interests
7 unique_interests[:20]
```

```
Out[30]: ['Big Data',
          'C++',
          'Cassandra',
          'HBase',
          'Hadoop',
          'Haskell',
          'Java',
          'Mahout',
          'MapReduce',
          'MongoDB',
          'MySQL',
          'NoSQL',
          'Postgres',
          'Python',
          'R',
          'Spark',
          'Storm',
          'artificial intelligence',
          'databases',
          'decision trees']
```

Transform User Interest Matrix

Transform User Interest Matrix

```
In [31]: 1 # Transform between lists of strings and fixed length lists of ints
2 from sklearn.preprocessing import MultiLabelBinarizer
3
4 mlb = MultiLabelBinarizer(classes=unique_interests)
5
6 # a matrix of "user" rows and "interest" columns
7 user_interest_matrix = mlb.fit_transform(users_interests)
8
9 # The interests for user0
10 user_interest_matrix[0]
```

```
Out[31]: array([1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Transform User Interest Matrix

```
In [31]: 1 # Transform between lists of strings and fixed length lists of ints
2 from sklearn.preprocessing import MultiLabelBinarizer
3
4 mlb = MultiLabelBinarizer(classes=unique_interests)
5
6 # a matrix of "user" rows and "interest" columns
7 user_interest_matrix = mlb.fit_transform(users_interests)
8
9 # The interests for user0
10 user_interest_matrix[0]
```

```
Out[31]: array([1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
In [32]: 1 # transforming back from interest matrix to list of interests
2 mlb.inverse_transform(user_interest_matrix)[0]
```

```
Out[32]: ('Big Data', 'Cassandra', 'HBase', 'Hadoop', 'Java', 'Spark', 'Storm')
```


Calculate Similarity

Calculate Similarity

```
In [33]: 1 from sklearn.metrics.pairwise import cosine_similarity
          2
          3 # using similarity, higher values are better
          4 user_similarities = cosine_similarity(user_interest_matrix)
          5
          6 # what are the similarites for user0 to other users?
          7 user_similarities[0].round(1)
```

```
Out[33]: array([1. , 0.3, 0. , 0. , 0. , 0.2, 0. , 0. , 0.2, 0.6, 0. , 0. , 0. ,
                0.2, 0. ])
```

Calculate Similarity

```
In [33]: 1 from sklearn.metrics.pairwise import cosine_similarity
          2
          3 # using similarity, higher values are better
          4 user_similarities = cosine_similarity(user_interest_matrix)
          5
          6 # what are the similarites for user0 to other users?
          7 user_similarities[0].round(1)
```

```
Out[33]: array([1. , 0.3, 0. , 0. , 0. , 0.2, 0. , 0. , 0.2, 0.6, 0. , 0. , 0. ,
                0.2, 0. ])
```

```
In [34]: 1 # what users does user0 share interests with?
          2 np.where(user_similarities[0])[0]
```

```
Out[34]: array([ 0,  1,  5,  8,  9, 13])
```

Find Similar Users

Find Similar Users

```
In [35]: 1 # return a sorted list of users based on similarity
2 # skip query user and similarity == 0
3 def most_similar_users_to(query_idx):
4     users_scores = [(idx,np.round(sim,2))
5                     for idx,sim in enumerate(user_similarities[query_idx])
6                     if idx != query_idx and sim > 0]
7     return sorted(users_scores, key=lambda x:x[1])
8
9 pd.DataFrame(most_similar_users_to(0),columns=['user','similarity'])
```

Out[35]:

	user	similarity
0	5	0.15
1	13	0.17
2	8	0.19
3	1	0.34
4	9	0.57

Recommend Based On User Similarity

- Want to return items liked by other users sorted by the similarity of those users

Recommend Based On User Similarity

- Want to return items liked by other users sorted by the similarity of those users

```
In [36]: 1 from collections import defaultdict
2
3 def user_based_suggestions(user_idx):
4     suggestions = defaultdict(float)
5
6     # iterate over interests of similar users
7     for other_idx, sim in most_similar_users_to(user_idx): # for each similar user
8         for interest in users_interests[other_idx]: # for each interest of that user
9             suggestions[interest] += sim # add weight based on the similarity of that user
10
11     # sort suggestions based on weight
12     suggestions = sorted(suggestions.items(),
13                          key=lambda x:x[1],
14                          reverse=True)
15
16     # return only new interests
17     return [(suggestion,weight.round(2))
18             for suggestion,weight in suggestions
19             if suggestion not in users_interests[user_idx]] # weed out existing interests
```

Recommend Based On User Similarity

Recommend Based On User Similarity

```
In [37]: 1 # reminder: original interests  
        2 users_interests[0]
```

```
Out[37]: ['Hadoop', 'Big Data', 'HBase', 'Java', 'Spark', 'Storm', 'Cassandra']
```

Recommend Based On User Similarity

```
In [37]: 1 # reminder: original interests
          2 users_interests[0]
```

Out[37]: ['Hadoop', 'Big Data', 'HBase', 'Java', 'Spark', 'Storm', 'Cassandra']

```
In [38]: 1 # top 5 new recommended interests
          2 pd.DataFrame(user_based_suggestions(0)[:5],columns=['interest','weight'])
```

Out[38]:

	interest	weight
0	MapReduce	0.57
1	Postgres	0.51
2	MongoDB	0.51
3	NoSQL	0.34
4	neural networks	0.19

Issues with Collab. Filtering

- **the cold start problem** : What if it's your first time?
- **sparsity** : How to recommend movies no one's seen?

Recommendation as Classification

- set1 features + set2 features -> label
- generate label based on history
- Examples
 - user + item -> purchased or not
 - candidate + job -> hired or not
 - ...
- Feature Engineering!

Recommendation as Classification: Example

Recommendation as Classification: Example

```
In [39]: 1 person_features = ['Age', 'Country', 'Interest']
          2 book_features = ['Price', 'Language', 'Topic']
          3
          4 features = ['Person_Age',
          5                 'Person_Country',
          6                 'Person_Interest',
          7                 'Book_Price',
          8                 'Book_Language',
          9                 'Book_Topic',
         10                 'Interest_Topic_Match',
         11                 'Country_Language_Match',
         12                 ]
          13
         14 # target: "Did person purchase book? 1 == yes, 0 == no"
         15
         16 # dataset: Generate all person x book pairs and calculate target
```

Recommendation as Classification: Prediction

Recommendation as Classification: Prediction

```
1 train classifier on dataset using one of our Classification Models
2
3 then, for a query_person:
4     1. generate all query_person x book pairs
5     2. calculate  $P(y=1|X)$  for all pairs using .predict_proba()
6     3. rank by  $P(y=1|X)$ 
7     4. return the top N books
```


Issues with Recommendation as Classification

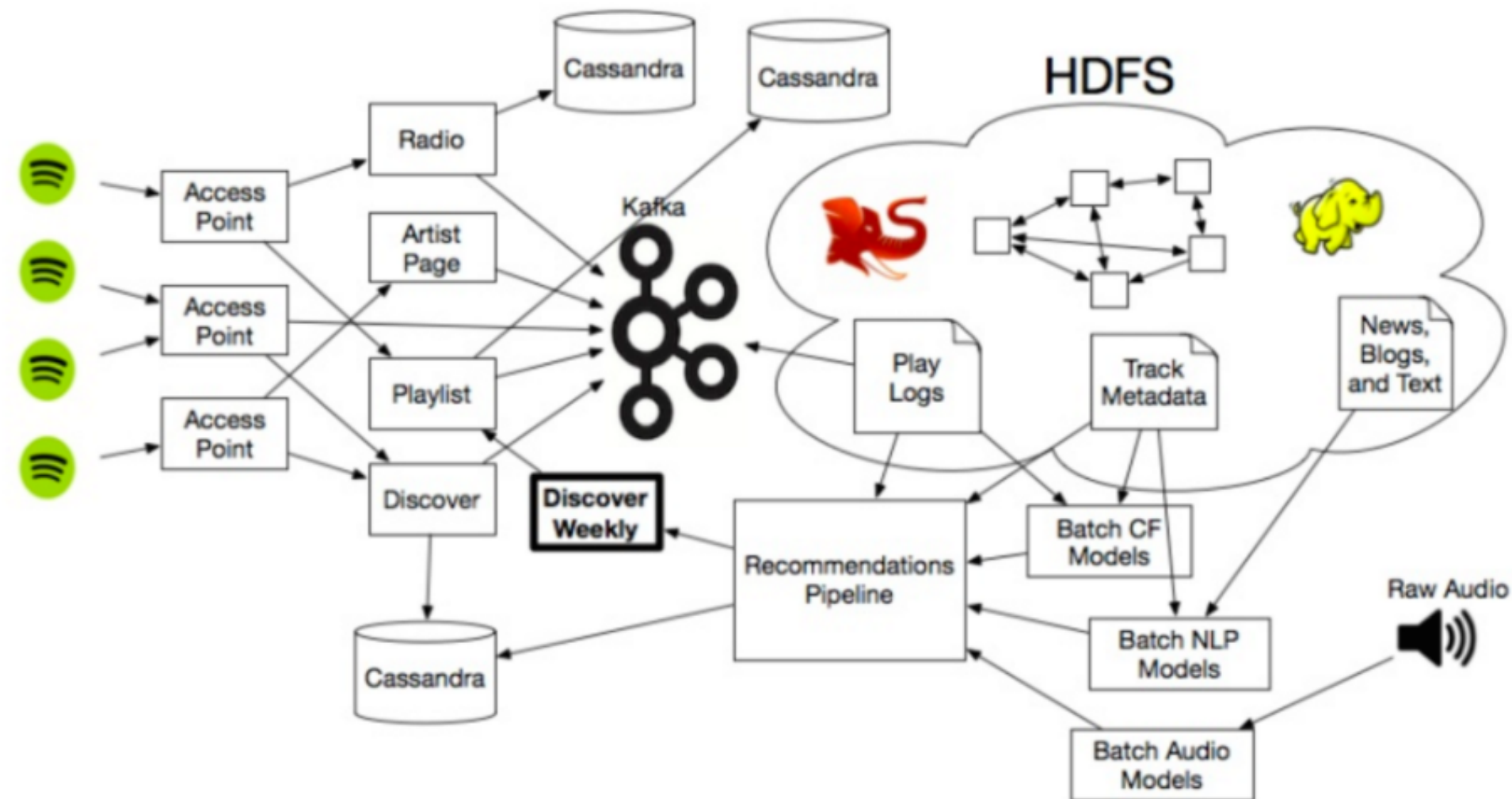
- Imbalanced classes
 - Example: each person bought different 1 of 100 books -> 1 pos to 99 neg
- False Negatives
 - Example: a person+book pair may be a good match even though it wasn't purchased

Evaluating Recommendation Systems

- **Precision At K:** Out of top K, how many were true/good? TP / K
- **Recall At K:** Out of all true/good, how many were in top K? $TP / (TP + FN)$
- Surprise/Novelty?
- Diversity?

Spotify's Recommendation Engine

How Does Spotify Know You So Well?



Recommendation Engines Review

- Content-Based
- User-Based Collaborative Filtering
- Recommendation as Classification
- Issues
- Evaluating

Questions re Recommendation Engines?

Imbalanced Classes

- **Imbalanced classes:**
 - when there is significantly more of one class than another in a classification task
- common in real world datasets
- Ex: credit card fraud
 - very small number of fraud transactions relative to total transactions

Dealing With Imbalanced Classes

- Stratified Sampling
- Random Undersampling
- Random Oversampling
- Oversample Synthetic Minority Items
 - SMOTE
 - ADASYN
- Other methods

Stratified Sampling

Stratified Sampling

```
In [40]: 1 from sklearn.model_selection import StratifiedKFold
2
3 x = np.ones(9)
4 y = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1])
5
6 skf = StratifiedKFold(n_splits=3)
7 for train_idx, test_idx in skf.split(X, y):
8     print(f"indices : {train_idx} {test_idx}")
9     print(f"values  : {y[train_idx]} {y[test_idx]}")
10    print()
```

```
indices : [2 3 4 5 7 8] [0 1 6]
values  : [0 0 0 0 1 1] [0 0 1]
```

```
indices : [0 1 4 5 6 8] [2 3 7]
values  : [0 0 0 0 1 1] [0 0 1]
```

```
indices : [0 1 2 3 6 7] [4 5 8]
values  : [0 0 0 0 1 1] [0 0 1]
```

Random Sampling

- Randomly Oversample minority class
- Randomly Undersample majority class

Example Dataset

Example Dataset

```
In [41]: 1 from sklearn.datasets import make_classification
          2 from collections import Counter
          3 X_imb, y_imb = make_classification(n_samples=5000, n_features=2, n_informative=2, n_redundant=0, n_repeated=0, n_classes=3,
          4                                     n_clusters_per_class=1, weights=[0.01, 0.05, 0.94], class_sep=0.8, random_state=0)
          5 df_imb = pd.DataFrame(X_imb); df_imb['y'] = y_imb; df_imb.y.value_counts()
```

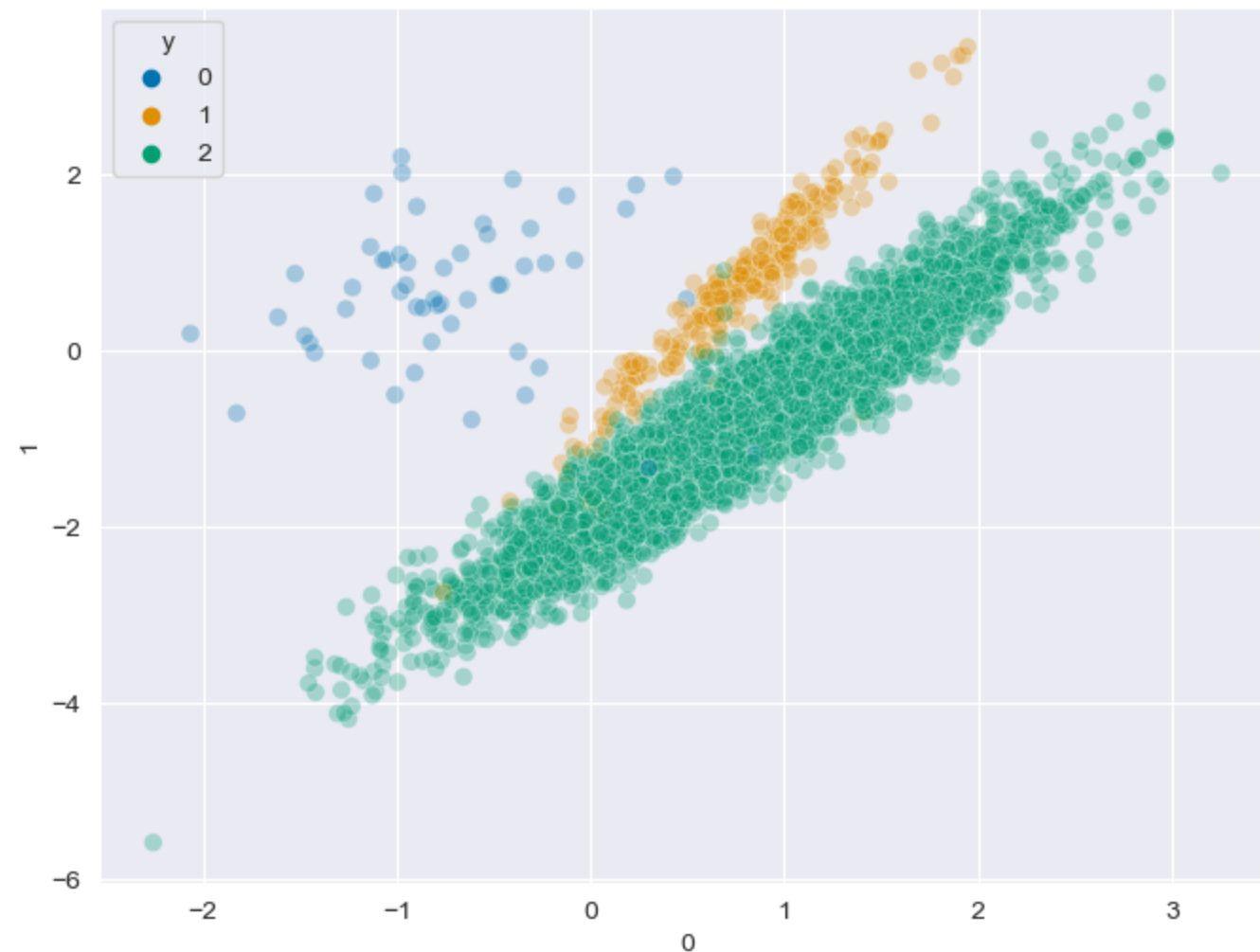
```
Out[41]: 2      4674
          1       262
          0        64
          Name: y, dtype: int64
```

Example Dataset

```
In [41]: 1 from sklearn.datasets import make_classification
2 from collections import Counter
3 X_imb, y_imb = make_classification(n_samples=5000, n_features=2, n_informative=2, n_redundant=0, n_repeated=0, n_classes=3,
4                                   n_clusters_per_class=1, weights=[0.01, 0.05, 0.94], class_sep=0.8, random_state=0)
5 df_imb = pd.DataFrame(X_imb); df_imb['y'] = y_imb; df_imb.y.value_counts()
```

```
Out[41]: 2    4674
1     262
0       64
Name: y, dtype: int64
```

```
In [42]: 1 fig,ax=plt.subplots(1,1,figsize=(8,6)); sns.scatterplot(x=0,y=1,hue='y',data=df_imb,palette="colorblind",alpha=.3,s=50);
```



Using imblearn

- `imblearn` is library created to deal with imbalanced classes
- need to install from `conda-forge` as `imbalanced-learn`
- import from `imblearn`

Random Oversampling of minority class

Random Oversampling of minority class

```
In [44]: 1 from imblearn.over_sampling import RandomOverSampler
          2
          3 ros = RandomOverSampler(random_state=0)
          4 X_ros, y_ros = ros.fit_resample(X_imb, y_imb)
          5 df_ros = pd.DataFrame(X_ros); df_ros['y'] = y_ros; df_ros.y.value_counts()
```

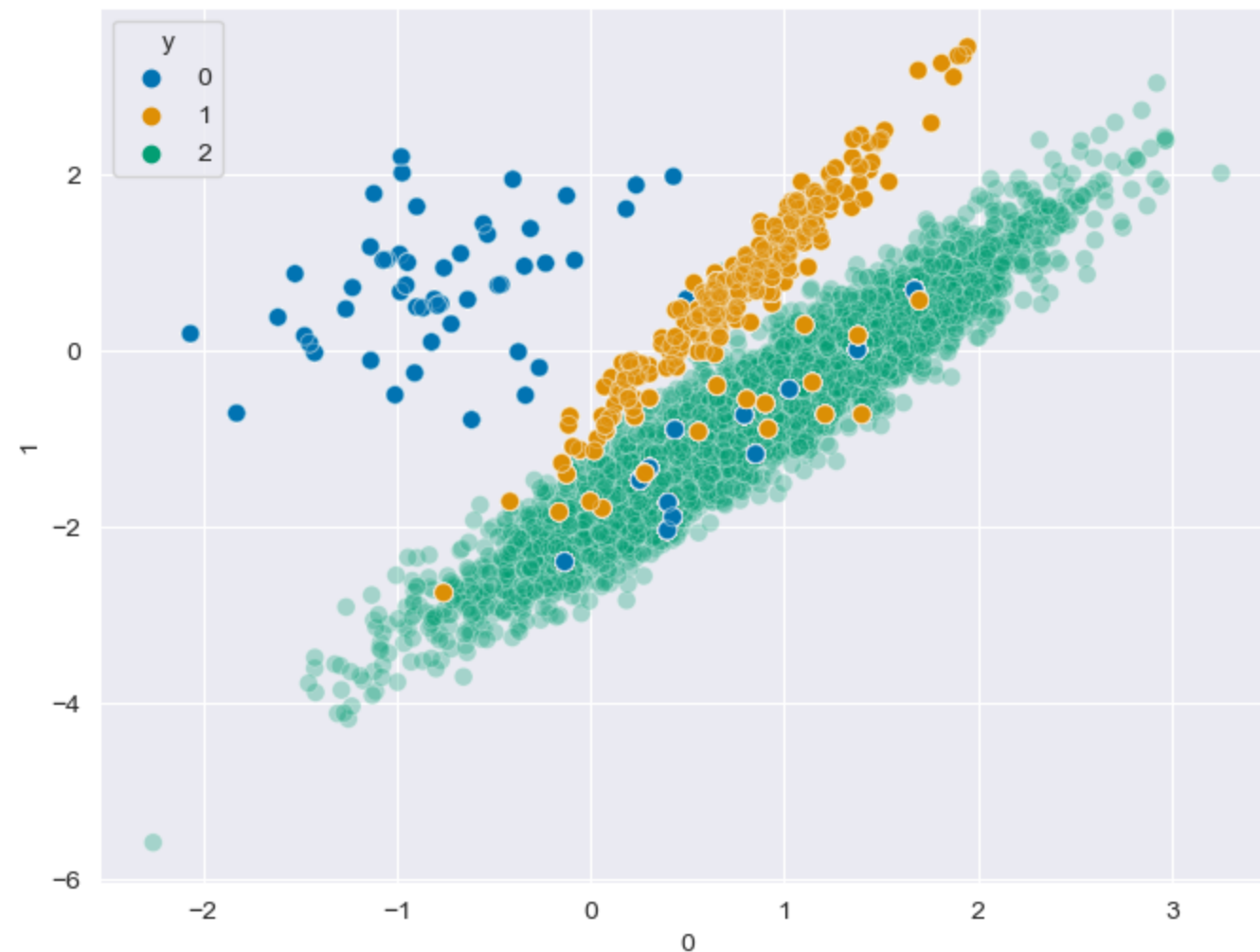
```
Out[44]: 2      4674
          1      4674
          0      4674
          Name: y, dtype: int64
```


Random Oversampling of minority class

```
In [44]: 1 from imblearn.over_sampling import RandomOverSampler
2
3 ros = RandomOverSampler(random_state=0)
4 X_ros, y_ros = ros.fit_resample(X_imb, y_imb)
5 df_ros = pd.DataFrame(X_ros); df_ros['y'] = y_ros; df_ros.y.value_counts()
```

```
Out[44]: 2    4674
1    4674
0    4674
Name: y, dtype: int64
```

```
In [45]: 1 fig,ax=plt.subplots(1,1,figsize=(8,6)); sns.scatterplot(x=0,y=1,hue='y',data=df_ros,palette="colorblind",alpha=.3,s=50);
```



Random Undersampling of majority class

Random Undersampling of majority class

```
In [46]: 1 from imblearn.under_sampling import RandomUnderSampler
          2
          3 rus = RandomUnderSampler(random_state=0)
          4 X_rus, y_rus, = rus.fit_resample(X_imb, y_imb)
          5 df_rus = pd.DataFrame(X_rus); df_rus['y'] = y_rus; df_rus.y.value_counts()
```

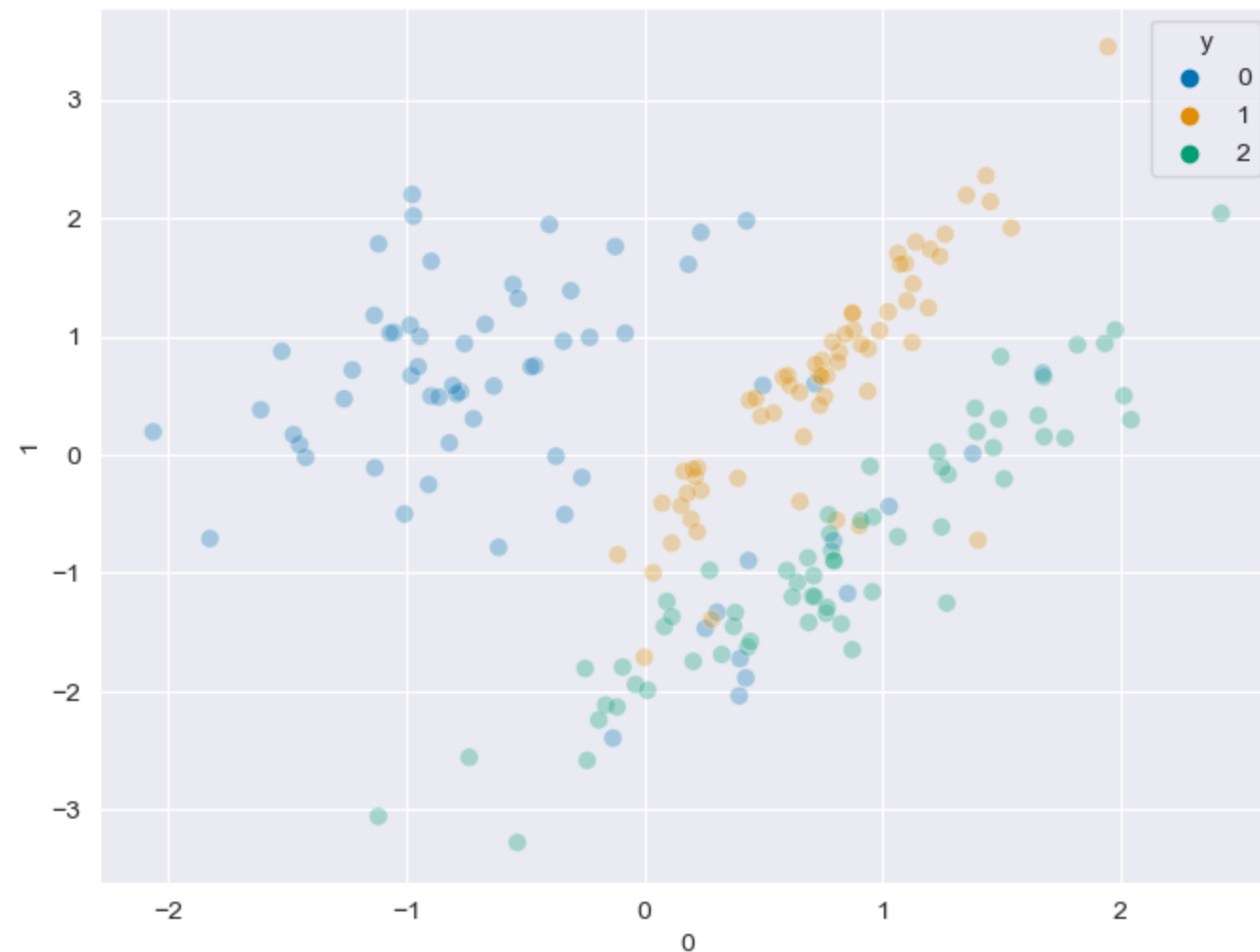
```
Out[46]: 0    64
          1    64
          2    64
          Name: y, dtype: int64
```

Random Undersampling of majority class

```
In [46]: 1 from imblearn.under_sampling import RandomUnderSampler
2
3 rus = RandomUnderSampler(random_state=0)
4 X_rus, y_rus, = rus.fit_resample(X_imb, y_imb)
5 df_rus = pd.DataFrame(X_rus); df_rus['y'] = y_rus; df_rus.y.value_counts()
```

```
Out[46]: 0    64
1    64
2    64
Name: y, dtype: int64
```

```
In [47]: 1 fig,ax=plt.subplots(1,1,figsize=(8,6)); sns.scatterplot(x=0,y=1,hue='y',data=df_rus,palette="colorblind",alpha=.3,s=50);
```



Oversample Synthetic Minority Items

- SMOTE: Synthetic Minority Oversampling
- ADASYN: Adaptive Synthetic Minority Oversampling

SMOTE: Synthetic Minority Oversampling

- Create new synthetic points between existing points

SMOTE: Synthetic Minority Oversampling

- Create new synthetic points between existing points

```
In [48]: 1 from imblearn.over_sampling import SMOTE
          2 X_smote, y_smote = SMOTE().fit_resample(X_imb, y_imb)
          3 df_smote = pd.DataFrame(X_smote); df_smote['y'] = y_smote; df_smote.y.value_counts()
```

```
Out[48]: 2    4674
          1    4674
          0    4674
          Name: y, dtype: int64
```

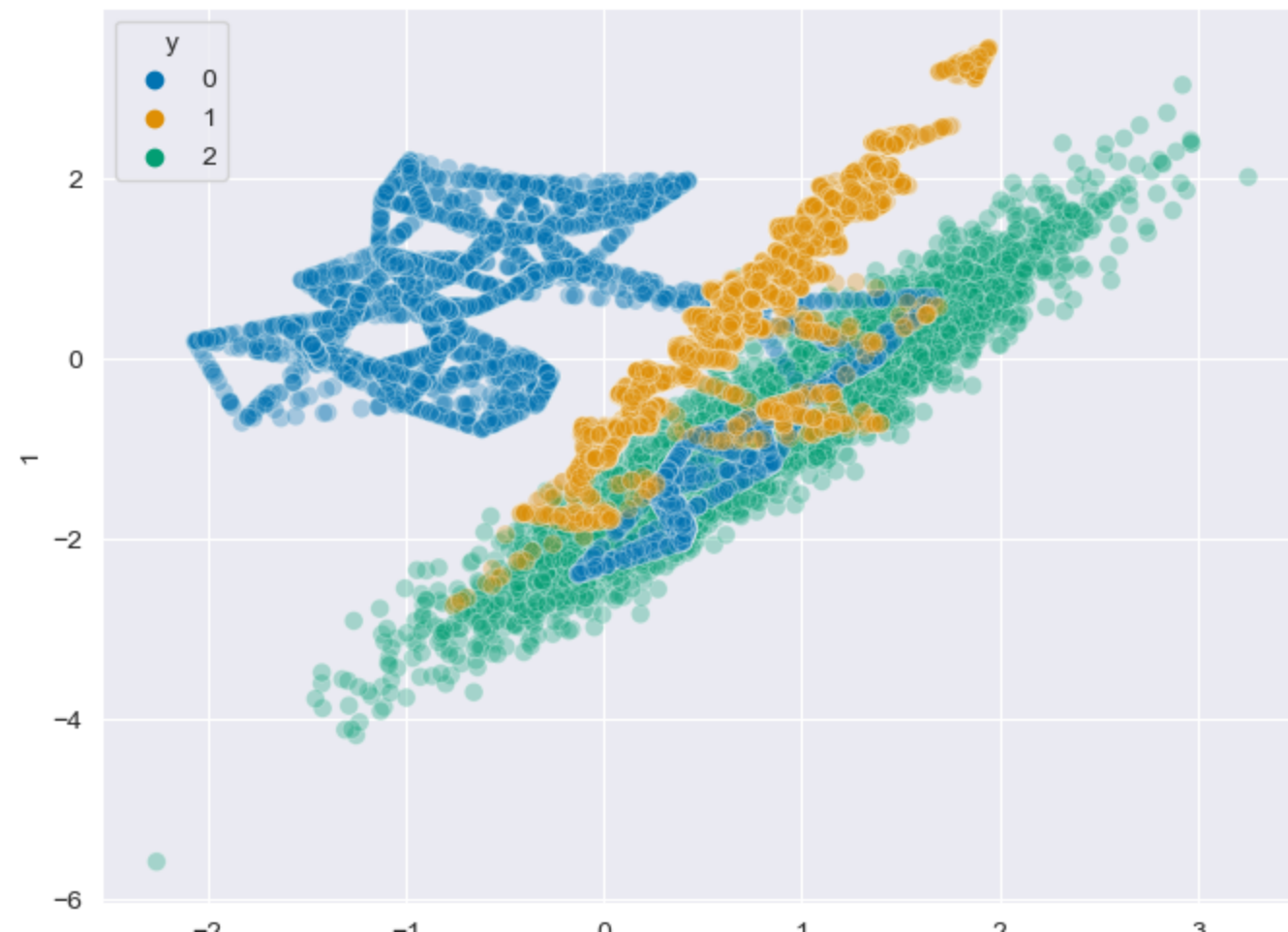
SMOTE: Synthetic Minority Oversampling

- Create new synthetic points between existing points

```
In [48]: 1 from imblearn.over_sampling import SMOTE
2 X_smote, y_smote = SMOTE().fit_resample(X_imb, y_imb)
3 df_smote = pd.DataFrame(X_smote); df_smote['y'] = y_smote; df_smote.y.value_counts()
```

```
Out[48]: 2    4674
1    4674
0    4674
Name: y, dtype: int64
```

```
In [49]: 1 fig,ax=plt.subplots(1,1,figsize=(8,6)); sns.scatterplot(x=0,y=1,hue='y',data=df_smote,palette="colorblind",alpha=.3,s=50);
```



ADASYN: Adaptive Synthetic Minority Oversampling

- Create new synthetic points between existing points *where classes overlap*

ADASYN: Adaptive Synthetic Minority Oversampling

- Create new synthetic points between existing points *where classes overlap*

```
In [50]: 1 from imblearn.over_sampling import ADASYN
          2 X_adasyn, y_adasyn = ADASYN().fit_resample(X_imb, y_imb)
          3 df_adasyn = pd.DataFrame(X_adasyn); df_adasyn['y'] = y_adasyn; df_adasyn.y.value_counts()
```

```
Out[50]: 2    4674
          0    4673
          1    4662
          Name: y, dtype: int64
```

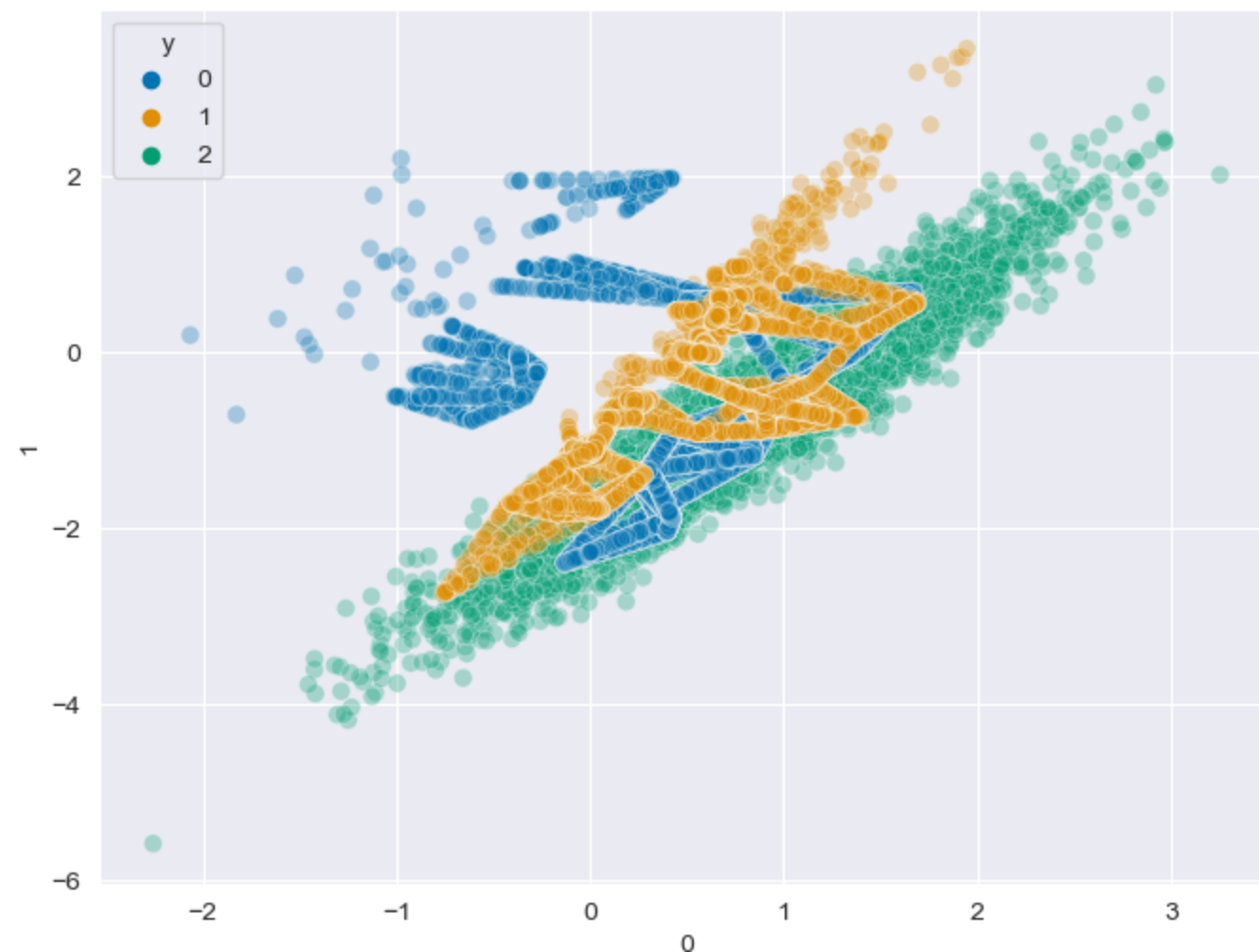
ADASYN: Adaptive Synthetic Minority Oversampling

- Create new synthetic points between existing points *where classes overlap*

```
In [50]: 1 from imblearn.over_sampling import ADASYN
2 X_adasyn, y_adasyn = ADASYN().fit_resample(X_imb, y_imb)
3 df_adasyn = pd.DataFrame(X_adasyn); df_adasyn['y'] = y_adasyn; df_adasyn.y.value_counts()
```

```
Out[50]: 2    4674
0    4673
1    4662
Name: y, dtype: int64
```

```
In [51]: 1 fig,ax=plt.subplots(1,1,figsize=(8,6)); sns.scatterplot(x=0,y=1,hue='y',data=df_adasyn,palette="colorblind",alpha=.3,s=50);
```



Other methods for dealing with imbalanced classes

- Adjust class weight (sklearn)
 - Adjust decision threshold (sklearn)
 - Treat as anomaly detection
 - Generate/buy more labels
-
- See https://imbalanced-learn.readthedocs.io/en/stable/auto_examples/over-sampling/plot_comparison_over_sampling.html

Questions re Imbalanced Classes?