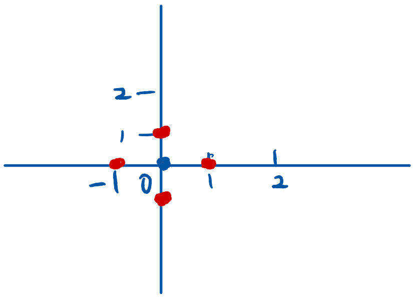
1

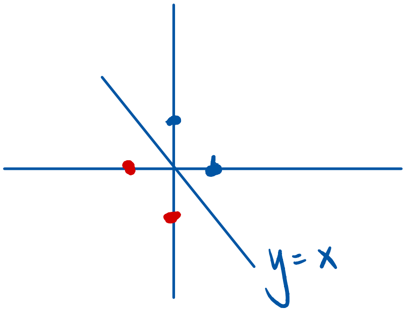
(1) case 1 : Not linear separable



1 0.5

-1 -0.5

Case 2 :



(2) $(X \cdot Z)^2 = (X_1 Z_1 + X_2 Z_2)^2$

$= X_1^2 Z_1^2 + 2 X_1 X_2 Z_1 Z_2 + X_2^2 Z_2^2$

$\Phi(X) = \langle X_1^2, \sqrt{2} X_1 X_2, X_2^2 \rangle$

$\Phi(Z) = \langle Z_1^2, \sqrt{2} Z_1 Z_2, Z_2^2 \rangle$

(3) $(0,0), (1,0), (0,1), (-1,0), (0,-1)$

$\Rightarrow (0,0,0), (1,0,0), (0,0,1)$ $(1,0,0), (0,0,1)$

plain $2x+2y=1$ can separate the two groups

2.

	$c_1 = (2, 0)$	$c_2 = (0, -1)$
x_1	1	$\sqrt{2}$
x_2	$\sqrt{2}$	$\sqrt{3}$
x_3	3	$\sqrt{2}$
x_4	$\sqrt{10}$	1

C_1: x_1 x_2

C_2 x_3 x_4

New C_1: $\left(\frac{1}{2}(1+1), \frac{1}{2}(0+1)\right) = (1, \frac{1}{2})$

New C_2: $\left(\frac{1}{2}(-1-1), \frac{1}{2}(0-1)\right) = (-1, -\frac{1}{2})$

(3)

$f_1 = N(0,1)$ $\qquad$ $f_2 = N(1,1)$

$= \frac{1}{\sqrt{2\pi}} e^{\frac{-(x)^2}{2}}$ $\qquad$ $= \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-1)^2}{2}}$

$x_1 = 0.7$

$$P(x_1 | C) = 0.5 \times \frac{1}{\sqrt{2\pi}} e^{\frac{-0.49}{2}} + 0.5 \times \frac{1}{\sqrt{2\pi}} e^{\frac{-0.09}{2}}$$

$$= 0.347$$

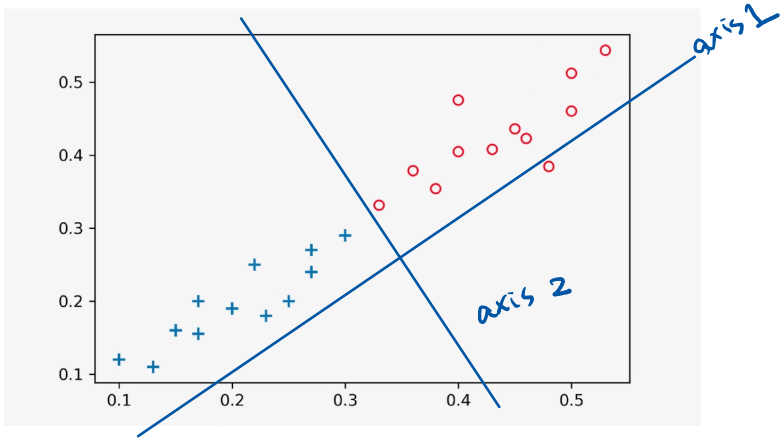
$x_2 = 1.5$

$$P(x_2 | C) = 0.5 \times \frac{1}{\sqrt{2\pi}} e^{\frac{-2.25}{2}} + 0.5 \times \frac{1}{\sqrt{2\pi}} e^{\frac{-0.25}{2}}$$

$$= 0.24$$

(4)

(1)



(2) axis 1

```python
import numpy as np

class KMeans():
    # This function initializes the KMeans class
    def __init__(self, k = 3, num_iter = 1000, order = 2):
        # Set a seed for easy debugging and evaluation
        np.random.seed(42)

        # This variable defines how many clusters to create
        # default is 3
        self.k = k

        # This variable defines how many iterations to recompute centroids
        # default is 1000
        self.num_iter = num_iter

        # This variable stores the coordinates of centroids
        self.centers = None

        # This variable defines whether it's K-Means or K-Medians
        # an order of 2 uses Euclidean distance for means
        # an order of 1 uses Manhattan distance for medians
        # default is 2
        if order == 1 or order == 2:
            self.order = order
        else:
            raise Exception("Unknown Order")

    # This function fits the model with input data (training)
    def fit(self, X):
        # m, n represent the number of rows (observations)
        # and columns (positions in each coordinate)
        m, n = X.shape

        # self.centers are a 2d-array of
        # (number of clusters, number of dimensions of our input data)
        self.centers = np.zeros((self.k, n))

        # self.cluster_idx represents the cluster index for each observation
        # which is a 1d-array of (number of observations)
        self.cluster_idx = np.zeros(m)


        for i in range(n):
            feature = X[:,i]
            ten_percentile = np.percentile(feature,10)
            ninety_percentile = np.percentile(feature, 90)
            self.centers[:,i] =
             np.random.uniform(ten_percentile,ninety_percentile, size=self.k)

        for i in range(self.num_iter):
            cluster_idx = np.zeros(m)
            distances = np.zeros((m,self.k))
            for ob_ind in range(m):
                for center_ind in range(self.k):
                    distances[ob_ind][center_ind] = np.linalg.norm(X[ob_ind] -
                     self.centers[center_ind], ord=self.order)
            cluster_idx = np.argmin(distances, axis=1)

            new_centers = np.zeros((self.k, n))
            for idx in range(self.k):
                cluster_coordinates = X[cluster_idx == idx]  # first
                 cluster_idx== idx return a bool array, then X[bool_array]
                 keeps only the row with 'True', which in this case, the row
                 in X belongs to the current cluster ind
                if self.order == 2:
                    cluster_center = np.mean(cluster_coordinates,axis=0)
                elif self.order == 1:
                    cluster_center = np.median(cluster_coordinates,axis=0)
                new_centers[idx, :] = cluster_center

            if np.all(cluster_idx == self.cluster_idx):
                print(f"Early Stopped at Iteration {i}")
                return self
            self.centers = new_centers
            self.cluster_idx = cluster_idx
        return self


    def predict(self, X):
        m = len(X)
        distances = np.zeros((m,self.k))
        for ob_ind in range(m):
            for center_ind in range(self.k):
                distances[ob_ind][center_ind] = np.linalg.norm(X[ob_ind] -
                 self.centers[center_ind], ord=self.order)
            self.cluster_idx[ob_ind] = np.argmin(distances[ob_ind])
        return self.cluster_idx
```

KMeans

May 15, 2023

0.1 Import necessary packages

You'll be implement your model in `KMeans.py` which should be put under the same directory as the location of `KMeans.ipynb`. Since we have enabled `autoreload`, you only need to import these packages once. You don't need to restart the kernel of this notebook nor rerun the next cell even if you change your implementation for `KMeans.py` in the meantime.

A suggestion for better productivity if you never used jupyter notebook + python script together: you can split your screen into left and right parts, and have your left part displaying this notebook and have your right part displaying your `KMeans.py`

[145]:
```
%load_ext autoreload
%autoreload 2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from KMeans import KMeans

from sklearn import datasets
from sklearn.datasets import make_blobs
from sklearn.metrics.cluster import adjusted_mutual_info_score
from sklearn.cluster import KMeans as Ref
from sklearn.manifold import TSNE, MDS
from sklearn.metrics import mean_squared_error
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

0.1.1 ATTENTION: THERE ARE A TOTAL OF 6 QUESTIONS THAT NEED YOUR ANSWERS

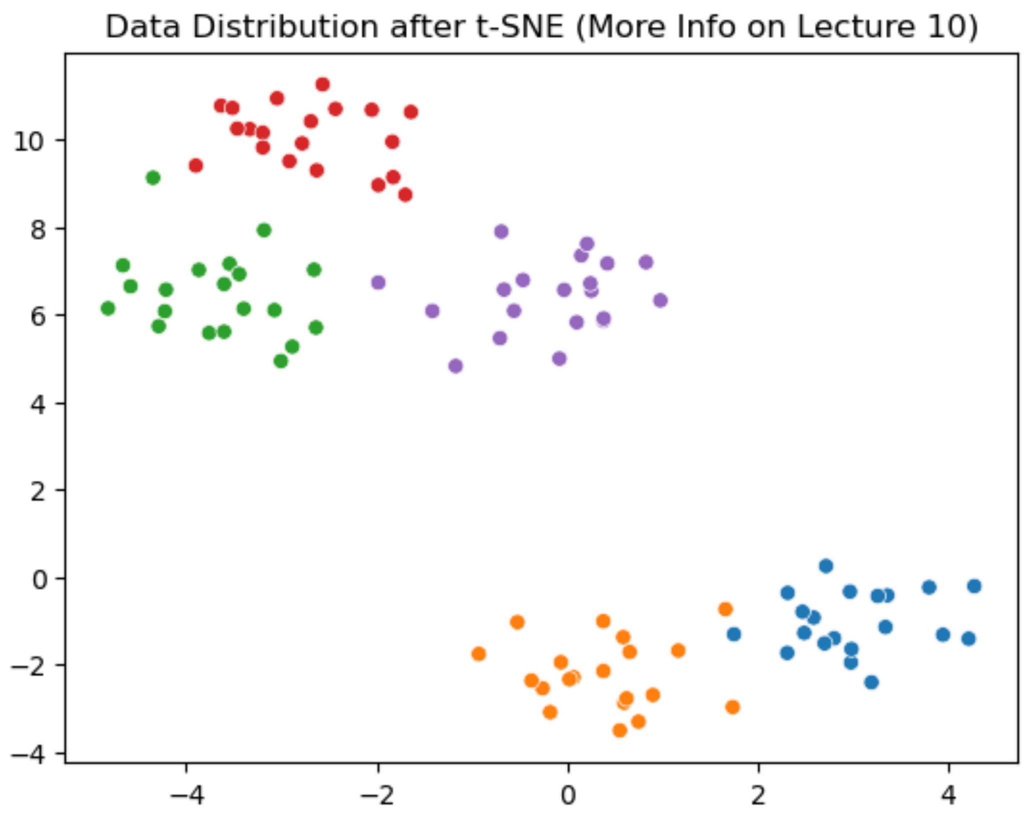
1 Experiment: Synthetic Data

First, let's play with our model on some synthetic data that have clear separation for different clusters. Here, let's make a dataset of 100 elements in 5 different clusters with 10 dimensions and visualize it by manifolding it into a 2D space with t-SNE.

Note: The distance that you can observe from the t-SNE visualization may be significantly different from the real distance due to the manifold embedding. Please refer to the PCA and T-SNE lecture for more details.

```
[146]: X, y = make_blobs(n_samples=100, centers=5, n_features=10, random_state=42,␣
        ↪cluster_std=2, center_box=(0, 10))

       dims = TSNE(random_state=42).fit_transform(X)
       dim1, dim2 = dims[:, 0], dims[:, 1]
       sns.scatterplot(x=dim1, y=dim2, hue=y, palette='tab10', legend=False)
       plt.title('Data Distribution after t-SNE (More Info on Lecture 10)');
```

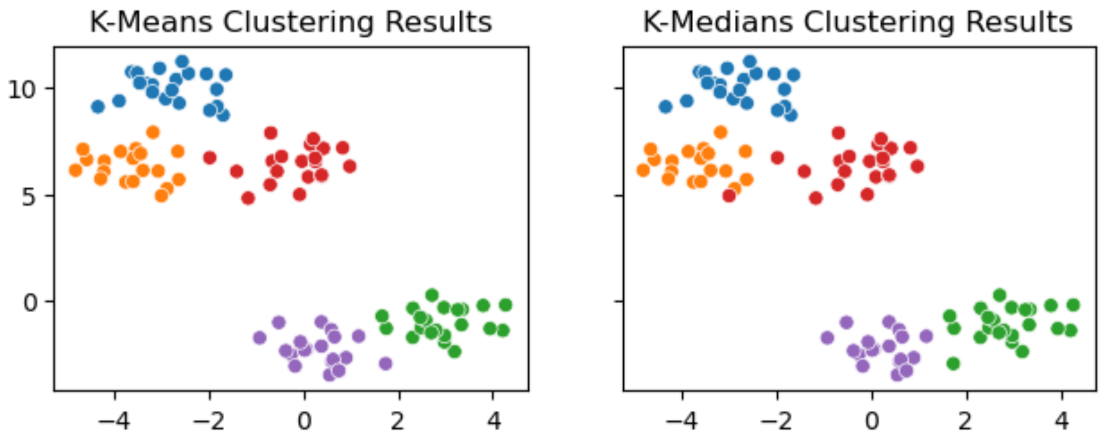
```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/manifold/_t_sne.py:780:
FutureWarning: The default initialization in TSNE will change from 'random' to
'pca' in 1.2.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/manifold/_t_sne.py:790:
FutureWarning: The default learning rate in TSNE will change from 200.0 to
'auto' in 1.2.
  warnings.warn(
```



Now, let's see how our algorithm performs compared to the ground truth.

```
[147]: fig, axes = plt.subplots(1, 2, figsize=(7.5, 2.5), sharey=True)

       # This is a reference of KMeans from sklearn's implementation, which we will be␣
        ↪using later to evaluate our model
       ref_kmeans = Ref(5, init='random').fit(X).predict(X)

       # This is to evaluate our KMeans model predictions
       y_pred_kmeans = KMeans(5, order=2).fit(X).predict(X)
       sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmeans, palette='tab10', ax=axes[0],␣
        ↪legend=False)
       axes[0].set_title('K-Means Clustering Results')

       # This is to evaluate our KMedians model predictions
       y_pred_kmedians = KMeans(5, order=1).fit(X).predict(X)
       sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmedians, palette='tab10',␣
        ↪ax=axes[1], legend=False)
       axes[1].set_title('K-Medians Clustering Results');
```

```
Early Stopped at Iteration 3
Early Stopped at Iteration 3
```



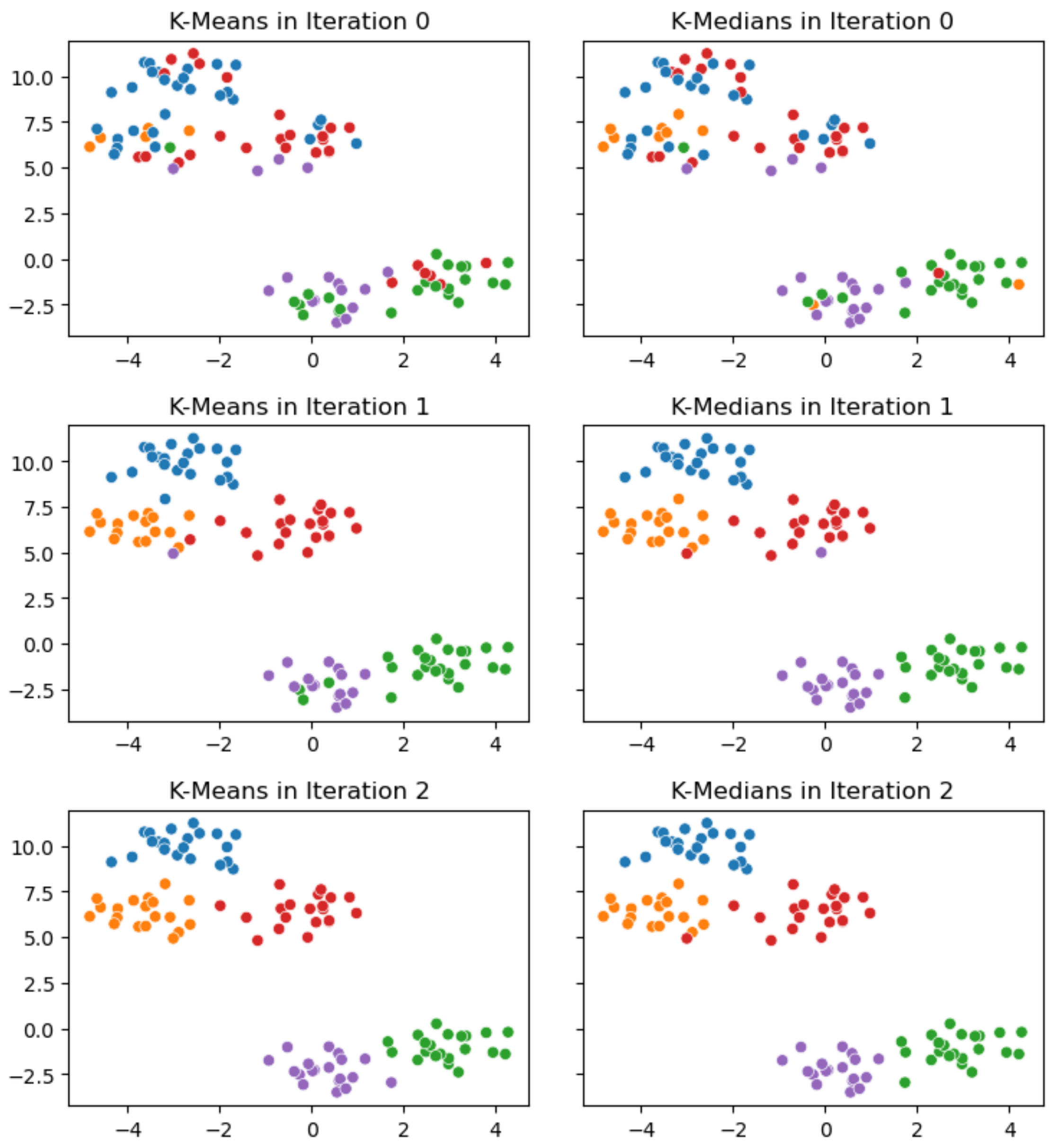
Question 1: From the above two figures, which one seems better compared to the original data distribution with actual cluster indices? Can you list some possible reasons why one way performs better than the other way?

Hint: Think of how we make the synthetic data. Also, next cell block shows the detailed clustering progress over each iteration.

Answer:K-Means seems to preform better than K-Medians. This synthetic data is generated around several cluster center, so in generally the cluster is within a radius to its center. In another word, the synthetic data is more compact to centers. So K-Means perform better here.

Let's see how the clustering goes over each iteration

```
[148]: fig, axes = plt.subplots(3, 2, figsize=(7.5, 8), sharey=True)
       fig.tight_layout()
       plt.subplots_adjust(hspace=0.3)

       # Don't worry about the fact that we train a separate model for each iteration
       # since we used a fixed random seed to ensure initialization consistency
       for i in range(3):
           y_pred = KMeans(5, num_iter=i, order=2).fit(X).predict(X)
           ax = axes[i][0]
           ax.title.set_text(f'K-Means in Iteration {i}')
           sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax,␣
        ↪legend=False)

           y_pred = KMeans(5, num_iter=i, order=1).fit(X).predict(X)
           ax = axes[i][1]
           ax.title.set_text(f'K-Medians in Iteration {i}')
           sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax,␣
        ↪legend=False);
```



Let's now evaluate our models with respect to sklearn's model. Here, we will be using adjusted mutual information score as our metric to evaluate the performance of clustering.

Hint: If your model is correctly implemented, you should have one of the models (K-Means, K-Medians) to have the same mutual info score as sklearn's implementation.

```
[149]: pd.DataFrame({'Reference K-Means from Sklearn vs Ground Truth':␣
         ↪adjusted_mutual_info_score(ref_kmeans, y),
                      'Our K-Means vs Ground Truth':␣
         ↪adjusted_mutual_info_score(y_pred_kmeans, y),
              'Our K-Medians vs Ground Truth':␣
↪adjusted_mutual_info_score(y_pred_kmedians, y)},
              index=['Mutual Info Score']).T
```

[149]:
```
                                               Mutual Info Score
Reference K-Means from Sklearn vs Ground Truth          0.947515
Our K-Means vs Ground Truth                             0.947515
Our K-Medians vs Ground Truth                           0.904241
```

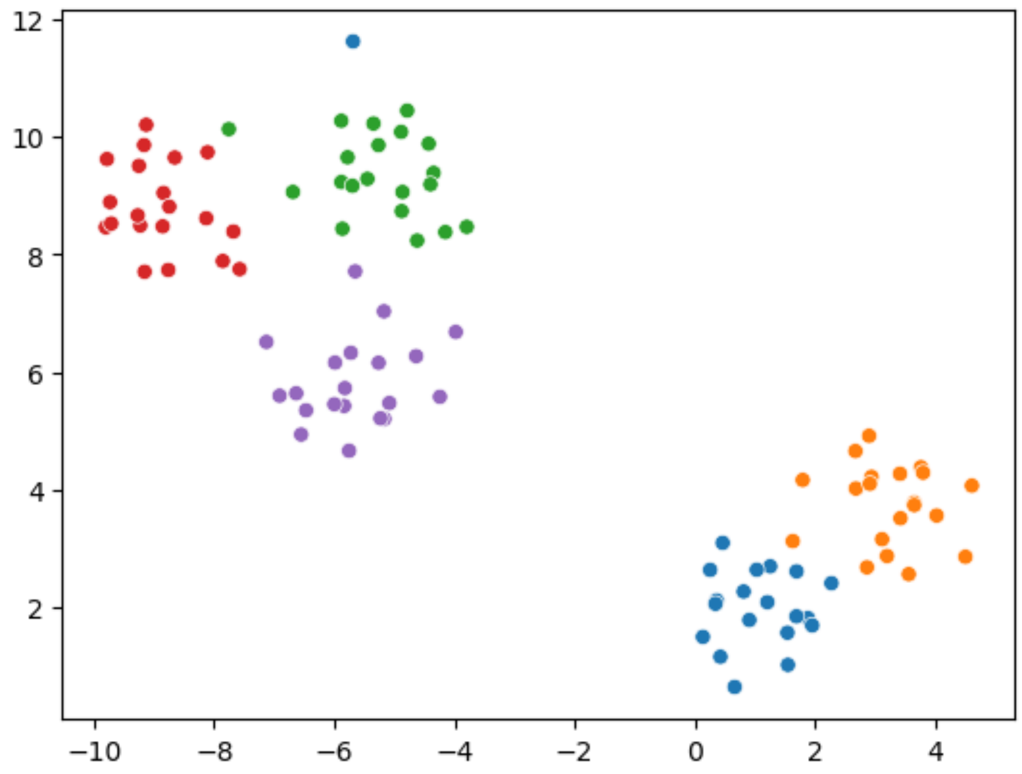
1.1 Wait... What happens when we have just one outlier?

Now, let's change one observation in the dataset to be an outlier. That is, we'll set the value of the first dimension of the first point in the dataset to be 100. Other than that, the rest of the dataset is kept completely the same as before.

As you can see in the below visualization, the manifolded figrue with t-SNE shows that there started to have points belonging to a particular cluster (according to the ground truth) appearing in the side of another cluster. However, if we discard the coloring of the below figure, we can still see that our dataset roughly has 5 clusters and each cluster contains an equal size of observations. We will see if this will affect the performance of our models.

[133]:
```
# let's make an outlier here
X[0][0] = 100

dims = TSNE(random_state=42).fit_transform(X)
dim1, dim2 = dims[:, 0], dims[:, 1]
sns.scatterplot(x=dim1, y=dim2, hue=y, palette='tab10', legend=False);
```

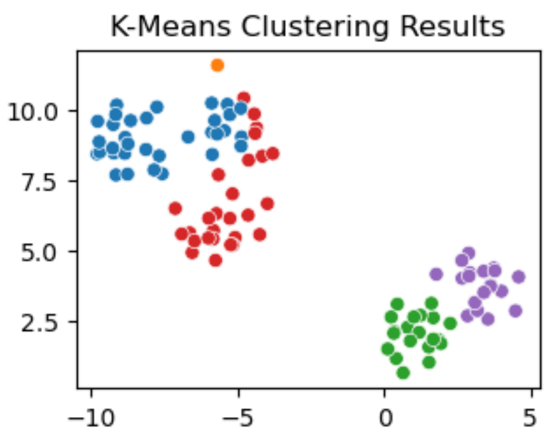
```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/manifold/_t_sne.py:780:
FutureWarning: The default initialization in TSNE will change from 'random' to
'pca' in 1.2.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/manifold/_t_sne.py:790:
FutureWarning: The default learning rate in TSNE will change from 200.0 to
'auto' in 1.2.
  warnings.warn(
```

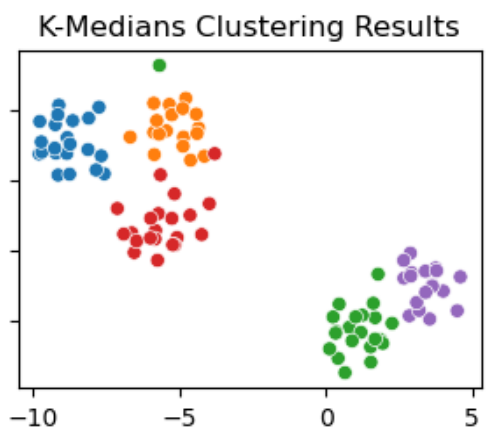


Now, let's see how our algorithm performs compared to the ground truth.

[134]:
```
fig, axes = plt.subplots(1, 2, figsize=(7.5, 2.5), sharey=True)

# This is a reference of KMeans from sklearn's implementation, which we will be␣
↪using later to evaluate our model
ref_kmeans = Ref(5, init='random').fit(X).predict(X)

# This is to evaluate our KMeans model predictions
y_pred_kmeans = KMeans(5, order=2).fit(X).predict(X)
sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmeans, palette='tab10', ax=axes[0],␣
↪legend=False)
axes[0].set_title('K-Means Clustering Results')

# This is to evaluate our KMedians model predictions
y_pred_kmedians = KMeans(5, order=1).fit(X).predict(X)
sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmedians, palette='tab10',␣
↪ax=axes[1], legend=False)
axes[1].set_title('K-Medians Clustering Results');
```

```
Early Stopped at Iteration 4
Early Stopped at Iteration 3
```





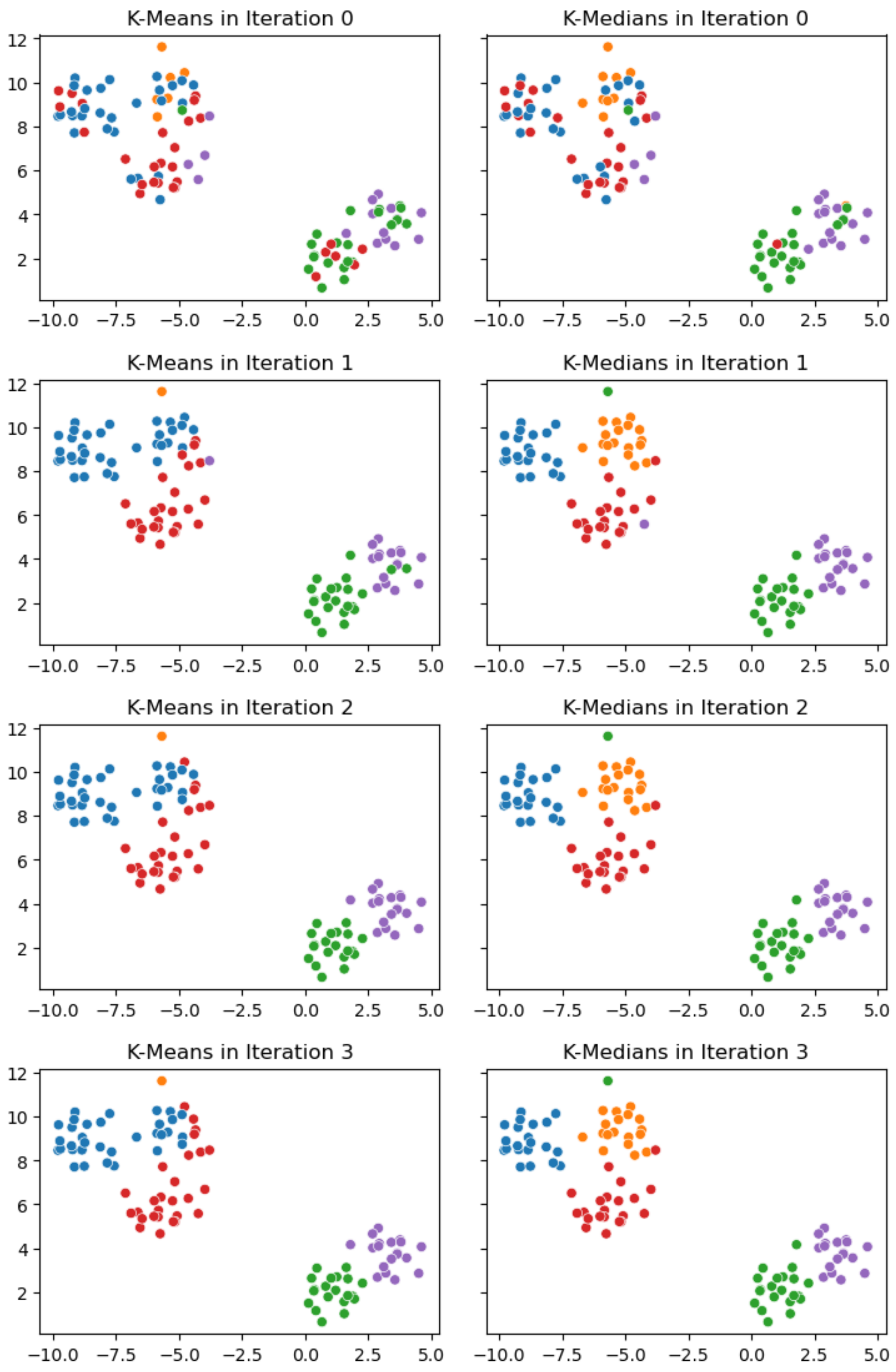
Question 2: From the above two figures, which one seems better compared to the original data distribution with actual cluster indices? Can you list some possible reasons why one way performs better than the other way?

Hint: Think of how we make the synthetic data. Also, think of the consequences of using means vs using medians in finding the centers.

Answer:K-Median seems to perform better than K-Mean. In K-means, the mean of each cluster is used to calculate distances, which can be greatly affected by outliers. On the other hand, K-median is more robust to outliers as they are less affected by extreme values since median is used.

Let's see how the clustering goes over each iteration

[135]:
```
fig, axes = plt.subplots(4, 2, figsize=(7.5, 11), sharey=True)
fig.tight_layout()
plt.subplots_adjust(hspace=0.3)

for i in range(4):
    y_pred = KMeans(5, num_iter=i, order=2).fit(X).predict(X)
    ax = axes[i][0]
    ax.title.set_text(f'K-Means in Iteration {i}')
    sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax,␣
 ↪legend=False)

    y_pred = KMeans(5, num_iter=i, order=1).fit(X).predict(X)
    ax = axes[i][1]
    ax.title.set_text(f'K-Medians in Iteration {i}')
    sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax,␣
 ↪legend=False);
```


K-Means in Iteration 0
K-Medians in Iteration 0
K-Means in Iteration 1
K-Medians in Iteration 1
K-Means in Iteration 2
K-Medians in Iteration 2
K-Means in Iteration 3
K-Medians in Iteration 3
12
10
8
6
4
2
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0

Let's now evaluate our models with respect to sklearn's model. Here, we will be using adjusted mutual information score as our metric to evaluate the performance of clustering.

Hint: If your model is correctly implemented, you should one of the scores higher than the reference score, and the other score lower than the reference score.

[136]:
```
pd.DataFrame({'Reference KMeans from Sklearn vs Ground Truth':␣
  ↪adjusted_mutual_info_score(ref_kmeans, y),
              'Our KMeans vs Ground Truth':␣
  ↪adjusted_mutual_info_score(y_pred_kmeans, y),
              'Our KMedians vs Ground Truth':␣
  ↪adjusted_mutual_info_score(y_pred_kmedians, y)},
              index=['Mutual Info Score']).T
```

[136]:
```
                                               Mutual Info Score
Reference KMeans from Sklearn vs Ground Truth           0.862091
Our KMeans vs Ground Truth                              0.781036
Our KMedians vs Ground Truth                            0.904241
```

Question 3: After the above experiments, (1)Can you summarize when is better to use Euclidean distance for K-Means, and when is better to use Manhattan distance for K-Medians? (2)If a model performs better on K-Medians than the most popular K-Means, what does that mean for the dataset? (3)Are there ways you can manipulate the dataset a little bit to make the model achieve a better performance on K-Means? (4)You may noticed that Sklearn's KMeans algorithms performs better than our KMeans algorithm. What could be the cause here?

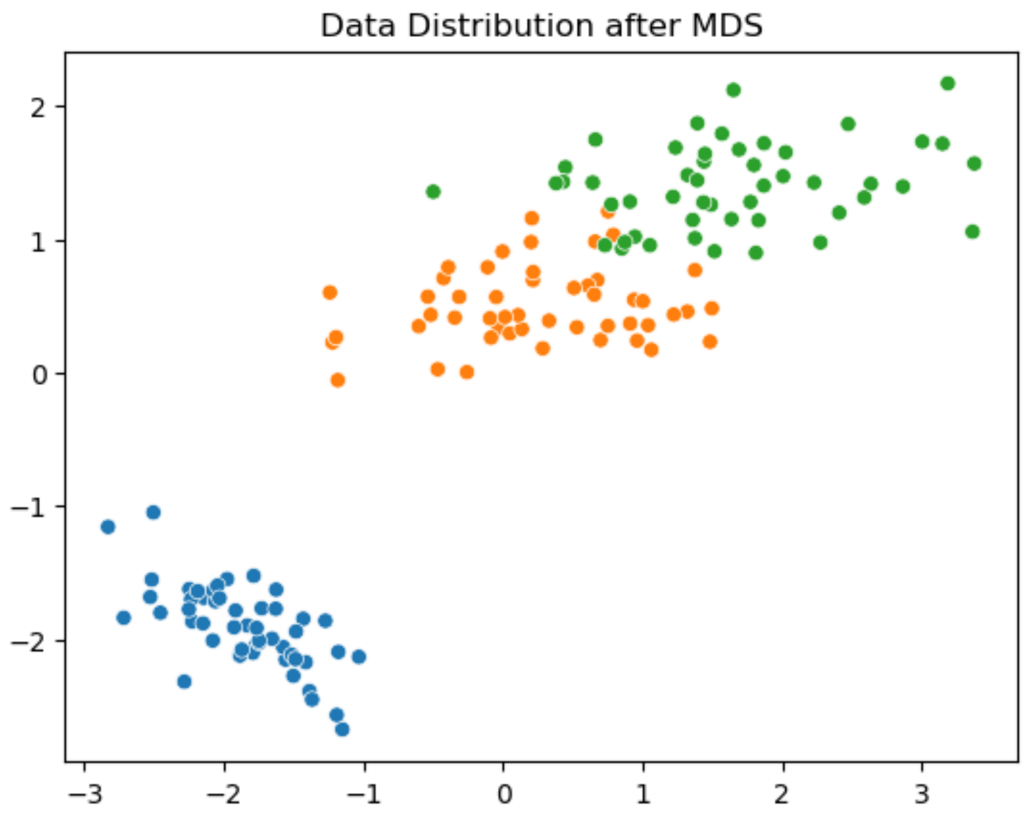
Answer: (1)When the data forms a more tight, well-separated clusters that follows a normal distribution, it is better to use K-means because it calculate Euclidiean distance. When the data has some outliers or not a spherical shape around center, the K-medians is better since it is more tolerable to outliers and different distribution within cluster using manhattan distance. (2) It may means the dataset has some outliers, or an indicator of within a cluster the distribution is not normal (3) Do some pre data processing that removes the outlier, or rescaling/ normalizing the features in the dataset can help improve the performance of K-means (4) They may perform a different machanism to find the first round of centroids of the dataset, which better helps the clustering later.

2 Experiment: Real-World Data

Now, after that we have dealt with some synthetic data, which come from a normal distribution at different centers, we will evaluate our model's performance on real-world data. Here, we will be using the iris dataset. Let's first visualize our data using Multi-Dimensional Scaling, which is another way to visualize multi-dimensional data into a 2D space.

[137]:
```
data = datasets.load_iris()
X, y = data['data'], data['target']
```

```
dims = MDS(random_state=42).fit_transform(X)
dim1, dim2 = dims[:, 0], dims[:, 1]
sns.scatterplot(x=dim1, y=dim2, hue=y, palette='tab10', legend=False)
plt.title('Data Distribution after MDS');
```

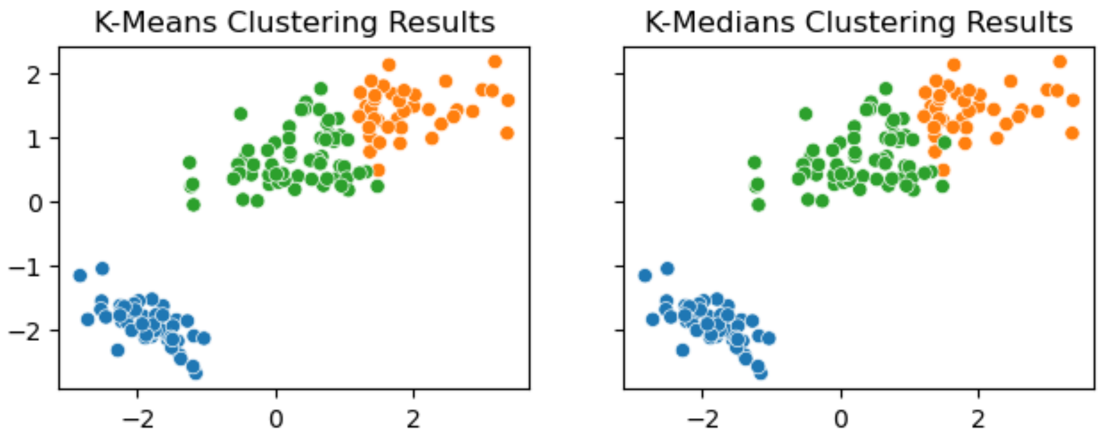


Now, let's see how our algorithm performs compared to the ground truth.

[138]:
```
fig, axes = plt.subplots(1, 2, figsize=(7.5, 2.5), sharey=True)

# This is a reference of KMeans from sklearn's implementation, which we will be␣
↪using later to evaluate our model
ref_kmeans = Ref(3, init='random').fit(X).predict(X)

# This is to evaluate our KMeans model predictions
y_pred_kmeans = KMeans(3, order=2).fit(X).predict(X)
sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmeans, palette='tab10', ax=axes[0],␣
↪legend=False)
axes[0].set_title('K-Means Clustering Results')

# This is to evaluate our KMedians model predictions
y_pred_kmedians = KMeans(3, order=1).fit(X).predict(X)
```

```
sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmedians, palette='tab10',␣
 ↪ax=axes[1], legend=False)
axes[1].set_title('K-Medians Clustering Results');
```

```
Early Stopped at Iteration 5
Early Stopped at Iteration 4
```



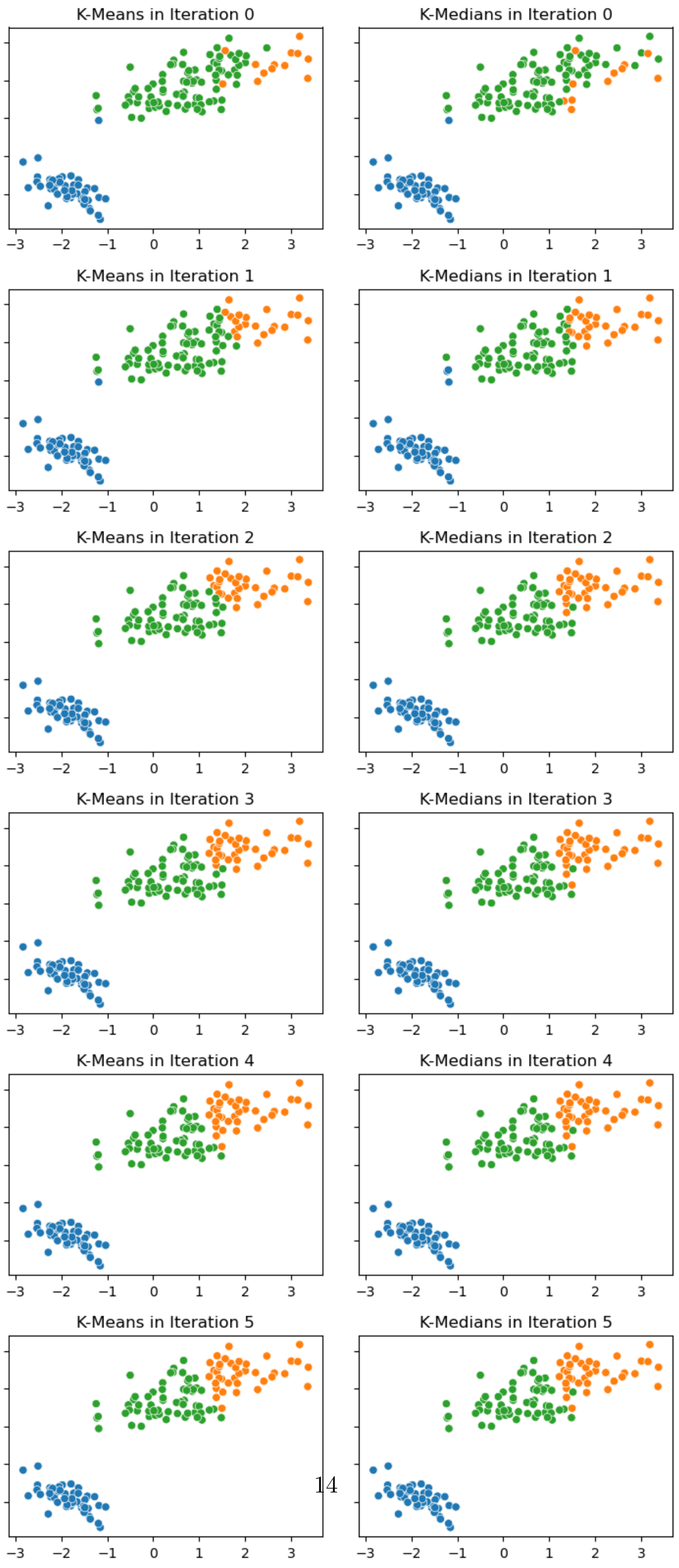
Question 4: From the above results, do our models still perform that well compared to the expereiment where we used the synthetic data? What makes the difference in real-life data?

Answer: The model perform well with the cluster that is far away from other cluster, but doesn't perform as well when the clusters are close to each other. Like the figure aboove, it mis-classified a lot of data between the green and orange cluster. The real-life data sometimes doesn't have a well-separated centroids; the boundries between different clusters are not so clear, which make the classification more tricky using the K-Mean/K-median model.

Let's see how the clustering goes over each iteration

```
[139]: fig, axes = plt.subplots(6, 2, figsize=(7.5, 16), sharey=True)
fig.tight_layout()
plt.subplots_adjust(hspace=0.3)

for i in range(6):
    y_pred = KMeans(3, num_iter=i, order=2).fit(X).predict(X)
    ax = axes[i][0]
    ax.title.set_text(f'K-Means in Iteration {i}')
    sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax,␣
 ↪legend=False)

    y_pred = KMeans(3, num_iter=i, order=1).fit(X).predict(X)
    ax = axes[i][1]
    ax.title.set_text(f'K-Medians in Iteration {i}')
    sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax,␣
↪legend=False);
```

```
Early Stopped at Iteration 4
```


K-Means in Iteration 0
K-Medians in Iteration 0
K-Means in Iteration 1
K-Medians in Iteration 1
K-Means in Iteration 2
K-Medians in Iteration 2
K-Means in Iteration 3
K-Medians in Iteration 3
K-Means in Iteration 4
K-Medians in Iteration 4
K-Means in Iteration 5
K-Medians in Iteration 5

Let's now evaluate our models with respect to sklearn's model. Here, we will be using adjusted mutual information score as our metric to evaluate the performance of clustering.

Hint: If your model is correctly implemented, you should have one of the models (K-Means, K-Medians) to have the same mutual info score as sklearn's implementation.

```
[140]: pd.DataFrame({'Reference KMeans from Sklearn vs Ground Truth':␣
        ↪adjusted_mutual_info_score(ref_kmeans, y),
                     'Our KMeans vs Ground Truth':␣
        ↪adjusted_mutual_info_score(y_pred_kmeans, y),
                     'Our KMedians vs Ground Truth':␣
        ↪adjusted_mutual_info_score(y_pred_kmedians, y)},
                     index=['Mutual Info Score']).T
```

```
[140]:                                               Mutual Info Score
       Reference KMeans from Sklearn vs Ground Truth           0.755119
       Our KMeans vs Ground Truth                              0.755119
       Our KMedians vs Ground Truth                            0.747583
```

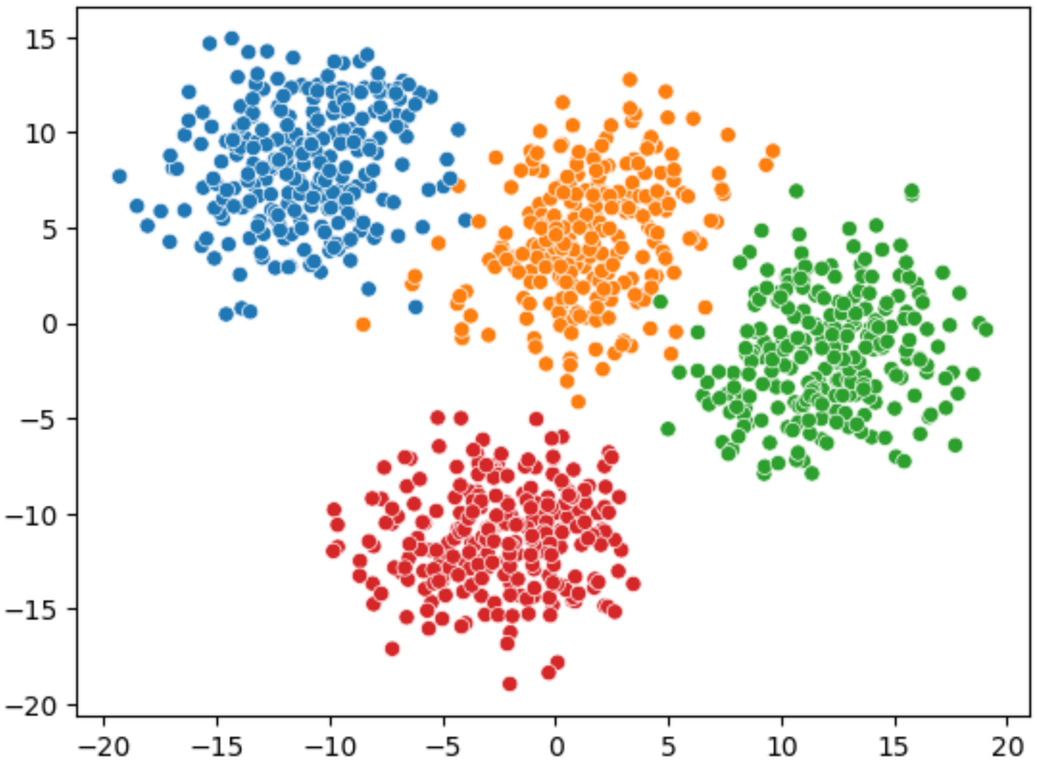
2.1 But most often time... we don't know how many clusters are there beforehand

Let us turn back to the synthetic data. For this part of the experiment, there are 1000 points in our data, and it should have 4 features and 4 centers (or 4 different types/labels).

```
[141]: X, y = make_blobs(n_samples=1000, n_features=4, centers=4, cluster_std=2.5,␣
        ↪random_state = 15)
```

```
[142]: dims = MDS(random_state=42).fit_transform(X)
       dim1, dim2 = dims[:, 0], dims[:, 1]
       sns.scatterplot(x=dim1, y=dim2, hue=y, palette='tab10', legend=False)
```

```
[142]: <AxesSubplot:>
```



We will train our model using k from 2 to 10, and store the Sum of Squares Error per cluster for each k.

SSE per cluster is the squared distances between points in a cluster and the cluster center. It indicates how "compact" each cluster is.

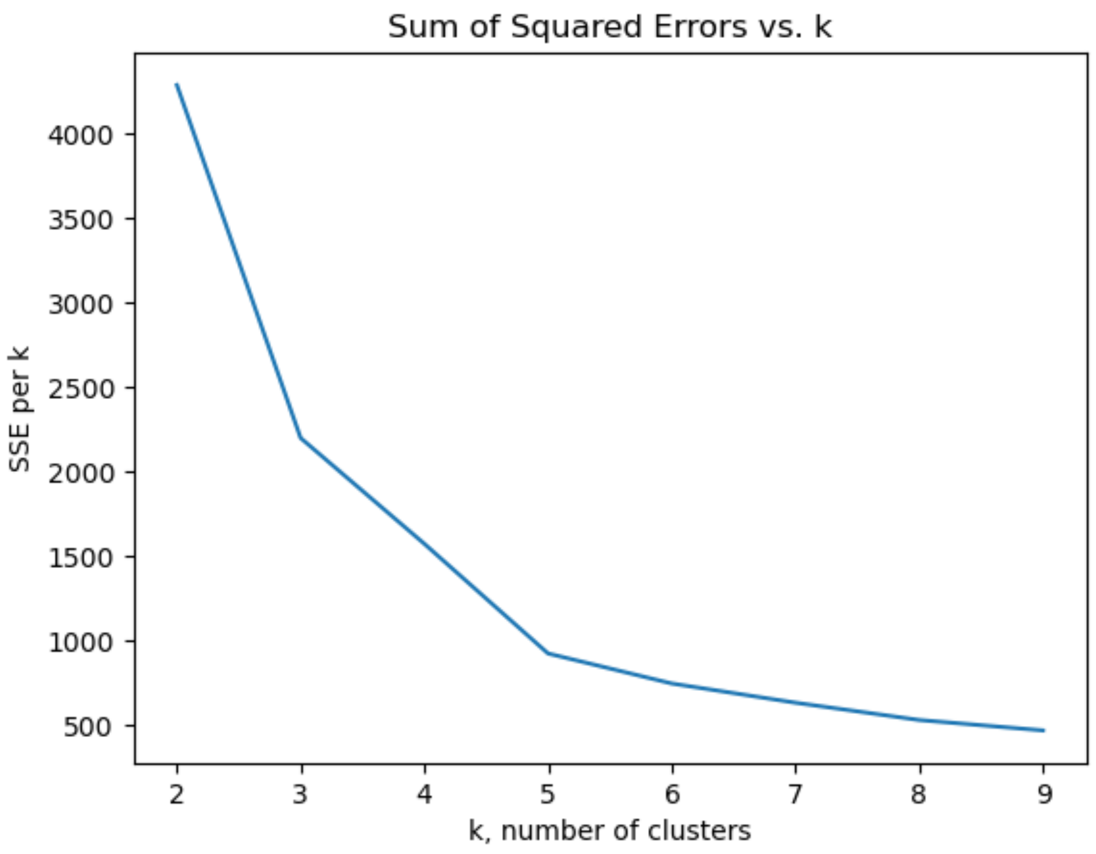
[143]:
```
SSE = []
y_preds = []

for i in range(2, 10):
    clf = KMeans(i, order=2)
    clf.fit(X)
    y_pred = clf.predict(X)
    SSE_jlst = []
    for j in range(i):
        SSE_j = 0
        idx = np.array(y_pred == j)
        for xj in X[idx]:
            se = np.linalg.norm((xj - clf.centers[j]), ord = 2)
            SSE_j += se
        SSE_jlst.append(SSE_j)
    SSE.append(sum(SSE_jlst) / i)
    y_preds.append(y_pred)
```

```
Early Stopped at Iteration 5
Early Stopped at Iteration 9
Early Stopped at Iteration 13
Early Stopped at Iteration 18
Early Stopped at Iteration 18
Early Stopped at Iteration 17
Early Stopped at Iteration 22
Early Stopped at Iteration 22
```

After training our model, let's plot SSE against k. Pay attention to the trend and see if you can find a tipping point. A tipping point sometimes indicates a balance point for our model; increasing k furthur would lead to overfitting.

[144]:
```
x = range(2, 10)
plt.plot(x, SSE)
plt.xlabel("k, number of clusters")
plt.ylabel("SSE per k")
plt.title('Sum of Squared Errors vs. k')
```

[144]:
```
Text(0.5, 1.0, 'Sum of Squared Errors vs. k')
```



Question 5: Can you find the most reasonable k based on this metric? Recall that we should have 4 clusters in this dataset. With that in mind, can you completely trust some metrics that help you determine how many clusters to use? What are the takeaways from this part of the experiment?

Answer: I will use K=3 or K=5 based on this metric, because in the graph there is a knee (change of slope) when K=3 and K=5, usually it is a good indicator of whether the current cluster number is effective on clustering the data. No we can only use the metric as a reference to choose the cluster number, since it is always hard to find a balance of overfitting and low SSE. The lesson is determining the cluster number is very hard depending on the matric because natually the error rate will always go down when having more cluster.Need to combine more graph and dataset analysis to decide the cluster number.

Question 6: In designing our model, we simplified some steps to make this implementation process easier. Can you list some potential improvements that can possibly make our model better when encountering data with noises or outliers, data with different types of distributions, or data where clusters have various distances of separation? (Hint: consider how we can work on initialization, assigning centers, data processing within the model, etc to make it better).

Answer:We can use K-means++ initialization method. It is a way to select cluster centers that makes better dispersion and improves convergence. We also should have run K-means with multiple rounds of different initial center selection and select the model with lowest error. We can use DBSCAN to pre-process the data so we can detect ourlier and remove them before running model. Instead of hard cluster assignment, we can also employ Gaussian Mixture Models, which allows more flexiblility with complicated real-word datasets and handle outliers better.

(6)

(1) $P(z_1|X=1) = \frac{P(X=1|z_1)\cdot P(z_1)}{P(X=1)}$

$$= \frac{\beta_1 e^{-\beta_1}\cdot \pi_1}{\beta_1 e^{-\beta_1}\cdot\pi_1 + \beta_2 e^{-\beta_2}\cdot\pi_2 + \beta_3 e^{-\beta_3}\cdot\pi_3}$$

(2) Given the current π, β, compute the cluster posterior for each x_i with each z_i by:

$$\begin{cases} P(z_i|x_j) = \dfrac{\pi_i \beta_i e^{-\beta_i x_j}}{\sum_{n=1}^{K} \pi_n \beta_n e^{-\beta_n x_j}} & \text{if } x_j \geq 0 \\ P(z_i|x_j) = 0, & \text{if } x_j < 0 \end{cases}$$