Homework Assignment 3

CSE 151A: Introduction to Machine Learning

Due: May 18th, 2023, 23:59pm (Pacific Time)

Instructions: Please answer the questions below, attach your code in the document, and insert figures to create **a single PDF file**.

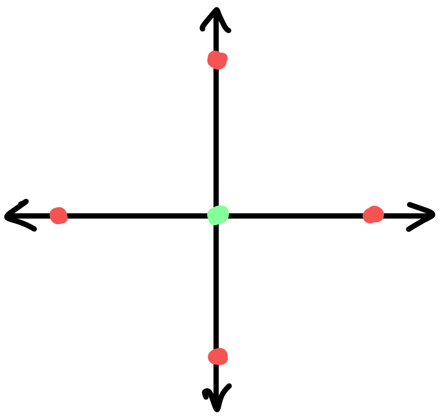
Grade: ____ out of 100 points

1 (20 points) SVM and kernel

1. (Linearly separability)(5 points) In lecture, we mentioned **Hard SVM** only works for **linearly separable** data points. Now you are given dataset $S = \{(\mathbf{x}_i, y_i), i = 1, .., N\}$ where each data point $(\mathbf{x}, y)$ contains a feature vector $\mathbf{x} \in \mathbb{R}^2$ and a label $y \in \{+, -\}$. Is the dataset linearly separable? If yes, please specify a decision boundary that linearly separates the data points with positive label from the data points with negative label.

   - Case 1:

label (y)	features ($\mathbf{x}$)
Positive (+)	$(0, 0)$
Negative (−)	$(1, 0), (0, 1), (-1, 0), (0, -1)$

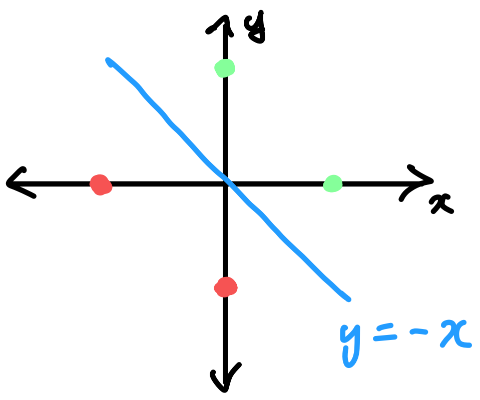


No, the positive class is surrounded by the negative class so cannot linearly separate in 2D space.

   - Case 2:

label (y)	features ($\mathbf{x}$)
Positive (+)	$(0, 1), (1, 0)$
Negative (−)	$(-1, 0), (0, -1)$

Yes, we can use $y = -x$ as the decision boundary



2. (Feature map and kernel)(10 points) Sometimes, data points that are not linearly separable in one feature space might be linearly separable in another. Therefore, we need a **transformation function (feature map)** ϕ, that maps features from one space to another space. Suppose for feature vector $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ we define **feature map** $\phi : \mathbb{R}^2 \to \mathbb{R}^3$ as: $\phi(\mathbf{x}) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right)$. Given the **feature map** ϕ, the corresponding **kernel** is defined as: $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$, where $\mathbf{x}, \mathbf{z} \in \mathbb{R}^2$. Prove that $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$ is the **kernel** for ϕ. In other words, prove that $\phi(\mathbf{x}) \cdot \phi(\mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$.

Hint: for vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n, \mathbf{x} \cdot \mathbf{y} = x_1y_1 + x_2y_2 + \ldots + x_ny_n$.

$$
\begin{aligned}
(\vec{x} \cdot \vec{z})^2 &= \phi(x) \cdot \phi(z) \\
&= (x_1^2, x_2^2, \sqrt{2}x_1x_2) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2) \\
&= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\
&= (x_1z_1 + x_2z_2)^2 \\
&= (\vec{x} \cdot \vec{z})^2
\end{aligned}
$$

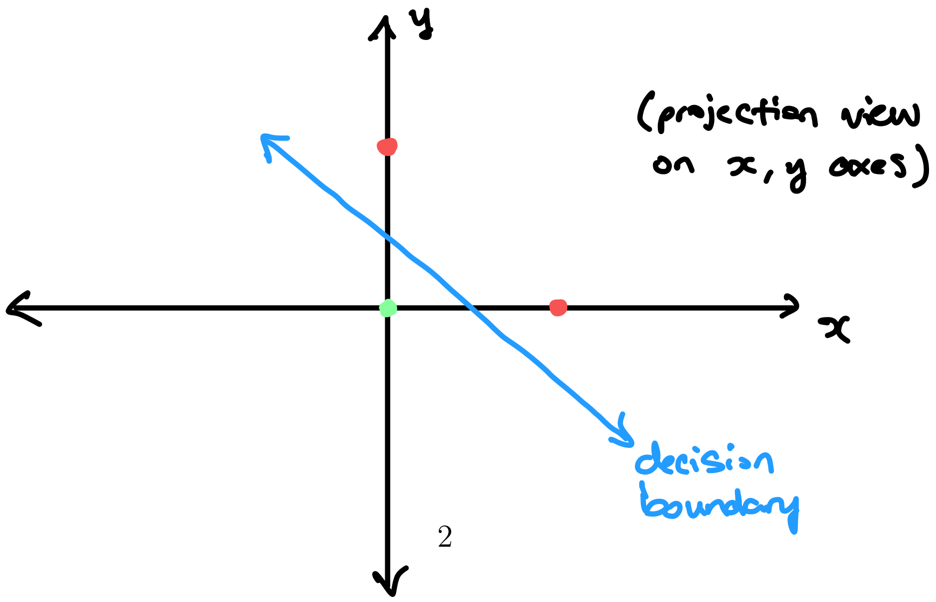
3. (5 points) Recall Case 1 in part 1. Apply ϕ defined in part 2 to the 5 data points in Case 1 and compute the new features in $\mathbb{R}^3$. Are they now linearly separable in $\mathbb{R}^3$? (Optional: specify a valid decision boundary in $\mathbb{R}^3$ if linearly separable)

Positive : $(0^2, 0^2, \sqrt{2}(0)(0)) \rightarrow (0,0,0)$

Negative : $(1^2, 0^2, \sqrt{2}(1)(0)), (0^2, 1^2, \sqrt{2}(0)(1)),$
$(-1^2, 0^2, \sqrt{2}(-1)(0)), (0^2, -1^2, \sqrt{2}(0)(-1))$

$\hookrightarrow (1,0,0), (0,1,0), (1,0,0), (0,1,0)$

Yes, as shown below, it is now linearly seperable



2 (10 points) K-means

Suppose you are given 4 data points: $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \in \mathbb{R}^2$, where $\mathbf{x}_1 = (1, 0), \mathbf{x}_2 = (1, 1)$, $\mathbf{x}_1 = (-1, 0), \mathbf{x}_1 = (-1, -1)$. Assume we know these points can be put into 2 clusters (k = 2). Suppose we start with **centroids** $\mathbf{c}_1 = (2, 0), \mathbf{c}_2 = (0, -1)$. Calculate where the new centroids would be after 1 iteration of **K-means**.

1) Classify points using Euclidean distance to initial centroids:

$$x_1: \operatorname{argmin}(c_1 = 1, c_2 = \sqrt{2}) \rightarrow c_1$$

$$x_2: \operatorname{argmin}(c_1 = \sqrt{2}, c_2 = \sqrt{5}) \rightarrow c_1$$

$$x_3: \operatorname{argmin}(c_1 = 3, c_2 = \sqrt{2}) \rightarrow c_2$$

$$x_4: \operatorname{argmin}(c_1 = \sqrt{10}, c_2 = 1) \rightarrow c_2$$

2) Update centroids:

$$c_1' = \frac{1}{|c_1|} \sum_{x_i \in c_1} x_i = \frac{1}{2} \cdot (2, 1)$$

$$\boxed{= (1, 0.5)}$$

$$c_2' = \frac{1}{|c_2|} \sum_{x_i \in c_2} x_i = \frac{1}{2} \cdot (-2, -1)$$

$$\boxed{= (-1, -0.5)}$$

3 (10 points) Gaussian Mixture Models

Suppose we have a set of probabilistic clusters, $\mathbf{C} = (C_1, C_2)$ .C_1 has probability density function $f_1 = \mathcal{N}(0,1)$, that is, a Gaussian distribution with mean 0 and variance 1 , and C_2 has probability density function $f_2 = \mathcal{N}(1,1)$. The weights of these clusters are given as: $\mathbb{P}(C_1) = \mathbb{P}(C_2) = 0.5$. Given the following data points in 1D, calculate the probability $\mathbb{P}(x \mid \mathbf{C})$ that a data point x is generated by this set of clusters $\mathbf{C}$, for the following 2 data points: (1)$x = 0.7$, (2)$x = 1.5$.

Hint: The probability density function of $\mathcal{N}(\mu, \sigma)$ is: $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-(x-\mu)^2}{2\sigma^2}$

$$\mathbb{P}(x_1) = \pi_1 N(x_1 | c_1) + \pi_2 N(x_1 | c_2)$$

$$= 0.5\left(\frac{1}{1\sqrt{2\pi}} \exp \frac{-(0.7-0)^2}{2(1)^2}\right)$$

$$+0.5\left(\frac{1}{1\sqrt{2\pi}} \exp \frac{-(0.7-1)^2}{2(1)^2}\right)$$

$$= \boxed{0.347}$$

$$\mathbb{P}(x_2) = \pi_1 N(x_2 | c_1) + \pi_2 N(x_2 | c_2)$$

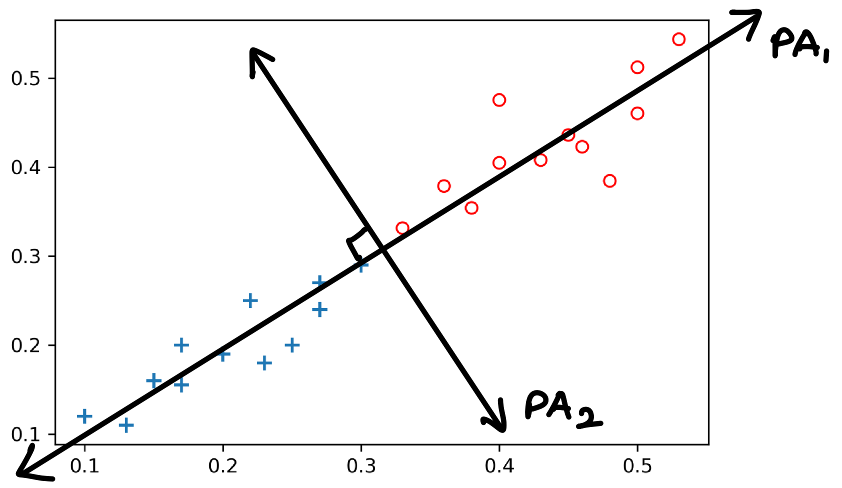
$$= 0.5\left(\frac{1}{1\sqrt{2\pi}} \exp \frac{-(1.5-0)^2}{2(1)^2}\right)$$

$$+0.5\left(\frac{1}{1\sqrt{2\pi}} \exp \frac{-(1.5-1)^2}{2(1)^2}\right)$$

$$= \boxed{0.241}$$

4 (10 points) Principle Component Analysis (PCA)

You are now given some data points in 2D, with 2 kinds of labels ("+" and "o").

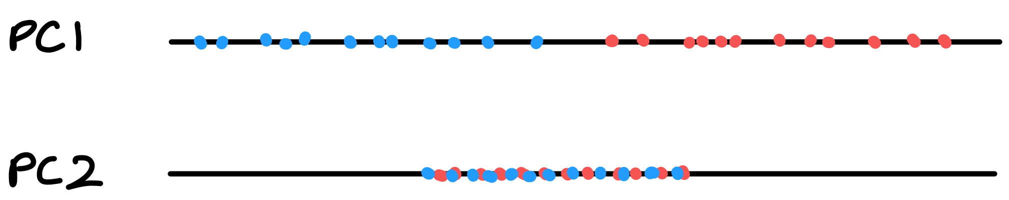


(1) (5 points) On the scatter plot, roughly draw the direction for the 1st and 2nd principle axes for this dataset.

(2) (5 points) Which principle axis would you choose to project onto, if we want to project the 2D features onto 1D, and still want good separation between the 2 labels?

I would choose to project onto $\boxed{PA_1}$ for the objective of having good separation between the two labels. This can be observed in the approximate projections onto PA_1 and PA_2 in 1D space illustrated below.

Of important note, while PA_1 does maximize the variance between points in the projected space by definition of the first principle axis, this is not the same as maximizing separability between labels (objective of LDA). In this particular case, the first principle axis, PA_1, also happens to be the vector that maintains better separability.



5 (50 points) Implementing the K-means algorithm

Now, you will implement a K-Means model from scratch. We have provided a skeleton code file (i.e. KMeans.py) for you to implement the algorithm as well as a notebook file (i.e. KMeans.ipynb) for you to conduct experiments and answer relevant questions. Libraries such as numpy and pandas may be used for auxiliary tasks (such as matrix multiplication, matrix inversion, and so on), but not for the algorithms. That is, you can use numpy to implement your model, but cannot directly call libraries such as scikit-learn to get a K-Means model for your skeleton code. We will grade this question based on the three following criteria:

1. Your implementation in code. Please do not change the structure of our skeleton code.
2. Your model's performance (we check if your model behaves correctly based on the results from multiple experiments in the notebook file).
3. Your written answers for questions in the notebook file.

```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)

        ##### TODO 1 ######
        #
        # Task: initialize self.centers
        #
        # Instruction:
        # For each dimension (feature) in X, use the 10th percentile and
        # the 90th percentile to form a uniform distribution. Then, we will initialize
        # the values of each center by randomly selecting values from the distributions.
        #
        # Note:
        # This method is by no means the best initialization method. However, we would
        # like you to follow our guidelines in this HW. We will ask you to discuss some better
        # initializaiton methods in the notebook.
        #
        # Hint:
        # 1. np.random.uniform(), np.percentile() might be useful
        # 2. make sure to look over its parameters if you're not sure
        ###################
        for i in range(n):
            lower, upper = np.percentile(X[:,i], [10, 90])
            self.centers[:,i] = np.random.uniform(lower, upper, self.k)
        ##### END TODO 1 #####

        for i in range(self.num_iter):
            # new_centers are a 2d-array of
            # (number of clusters, number of dimensions of our input data)
            new_centers = np.zeros((self.k, n))

            ##### TODO 2 ######
            #
            # Task: calculate the distance and create cluster index for each observation
            #
            # Instruction:
```

```
            # You should calculate the distance between each observation and each centroid
            # using specified self.order. Then, you should derive the cluster index for
            # each observation based on the minimum distance between an observation and
            # each of the centers.
            #
            # Hint:
            # 1. np.linalg.norm() might be useful, along with parameter axis, ord
            # for that function
            # 2. You can transpose an array using .T at the end
            # 3. np.argmin() might be useful along with parameter axis in finding
            # the desired cluster index of all observations
            #
            # IMPORTANT:
            # Copy-paste this part of your implemented code
            # to the predict function, and return cluster_idx in that function
            ####################
            distances = np.array([np.linalg.norm(X - center, ord=self.order, axis=1) for center in self.centers])
            cluster_idx = np.argmin(distances.T, axis=1)
            ##### END TODO 2 #####

            ##### TODO 3 ######
            #
            # Task: calculate the coordinates of new_centers based on cluster_idx
            #
            # Instruction:
            # You should assign the coordinates of the new_center by calculating
            # mean/median of the coordinates of observations belonging to the same
            # cluster.
            #
            # Hint:
            # 1. np.mean(), np.median() with axis might be helpful
            ####################
            for idx in range(self.k):
                cluster_coordinates = X[cluster_idx == idx]
                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
            ##### END TODO 3 #####

            ##### TODO 4 ######
            #
            # Task: determine early stop and update centers and cluster_idx
            #
            # Instructions:
            # You should stop tranining as long as cluster index for all
            # observations is the same as the previous iteration
            # Hint:
            # 1. .all() might be helpful
            ####################
            if (cluster_idx == self.cluster_idx).all():
                print(f"Early Stopped at Iteration {i}")
                return self
            self.centers = new_centers
            self.cluster_idx = cluster_idx
            ##### END TODO 4 #####
        return self


    # This function makes predictions with input data
    # Copy-paste your code from TODO 2 and return cluster_idx
    def predict(self, X):
        distances = np.array([np.linalg.norm(X - center, ord=self.order, axis=1) for center in self.centers])
        return np.argmin(distances.T, axis=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`

In [ ]:
```
%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
```

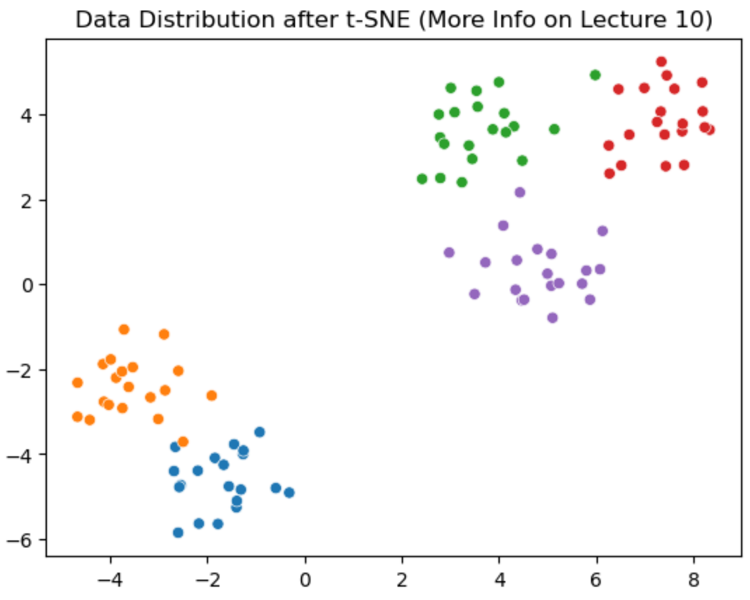
ATTENTION: THERE ARE A TOTAL OF 6 QUESTIONS THAT NEED YOUR ANSWERS

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.

In [ ]:
```
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)');
```

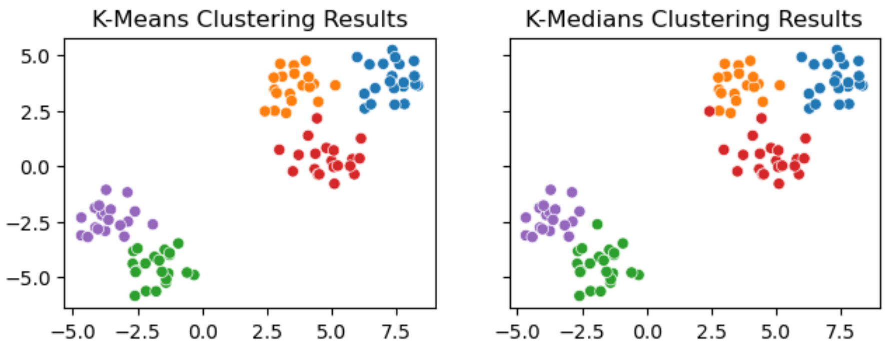


Now, let's see how our algorithm performs compared to the ground truth.

In [ ]:
```
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');
```

```
/Users/viren/anaconda3/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Early Stopped at Iteration 3
Early Stopped at Iteration 3
```



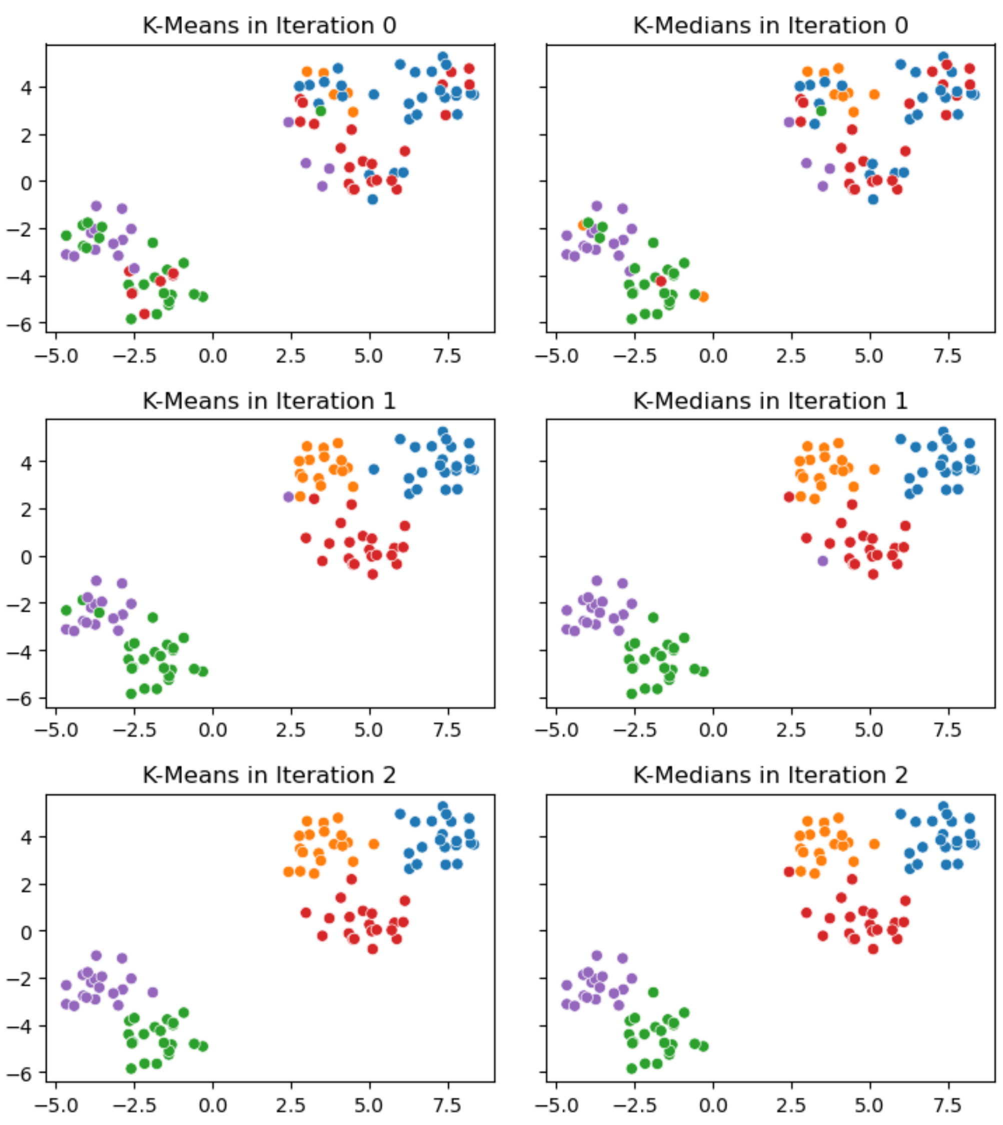
Question 1: From the above two figures, which one seems better compared to the original data distribution with actual cluster indices? Can you list some possible reasons why one way performs better than the other way?

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

Answer: K-Means clustering performs slightly better than K-Medians (0.948 vs 0.904 respectively for K-X vs ground truth). Sklearn make_blobs constructs clusters using a Gaussian distribution and the standard deviation function for a Gaussian distribution more closely resembles Euclidean distance of K-Means, which could explain the slightly superior performance of K-Means. However, the performance of K-Medians is not too far behind. We have fairly symmetric data with how we constructed the synthetic dataset. Since we don't really have outliers, noise or skew to our data, distance measurements taken by mean vs median approaches should be similar. Thus, we have similar model performance and classification as well, albiet with K-Means performing slightly better due to the nature of the synethic data construction.

Let's see how the clustering goes over each iteration

```
In [ ]: 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.

```
In [ ]: 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
```

Out[ ]:

	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

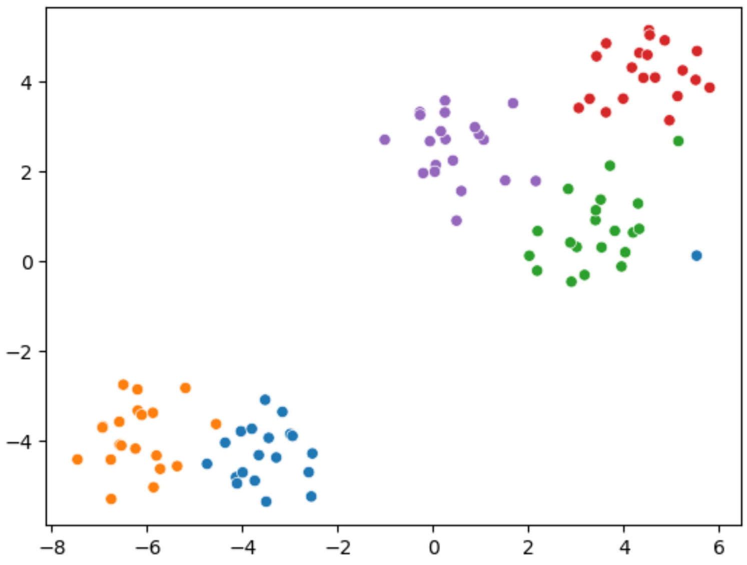
Wait... What happens when we have just one outlier?

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.

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

        dims = TSNE(random_state=42).fit_transform(X)
```

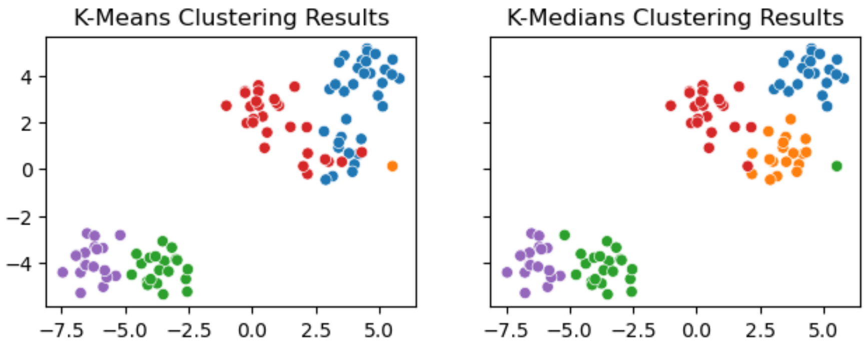
```
dim1, dim2 = dims[:, 0], dims[:, 1]
sns.scatterplot(x=dim1, y=dim2, hue=y, palette='tab10', legend=False);
```



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

In [ ]:
```
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');
```

```
/Users/viren/anaconda3/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Early Stopped at Iteration 4
Early Stopped at Iteration 3
```



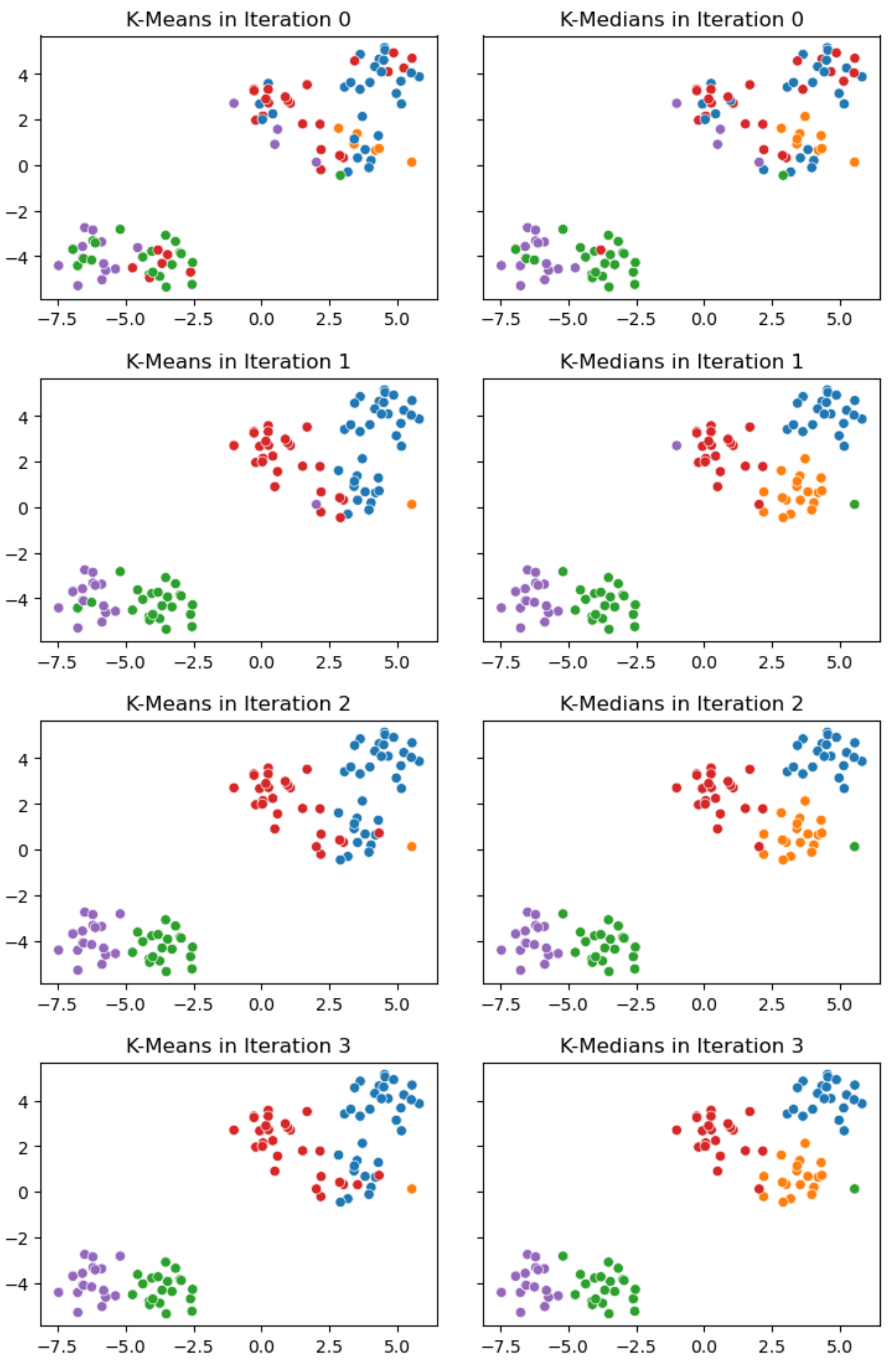
Question 2: From the above two figures, which one seems better compared to the original data distribution with actual cluster indices? Can you list some possible reasons why one way performs better than the other way?

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-Medians clustering performs better than K-Means in this case with the one outlier. We can see that K-Means's clusters differ significantly, espeically with the classes in the top right, compared to the ground truth. A reason for this is the nature of how distance is found between the two models and how they are affected by outliers. Mean is a measurement that is more susceptible to change from introduction of an outlier. Median on the other hand is typically not as affected by outliers since the introduction of one more data point would either mean the median stays the same or shifts by one index to a new data point which is probably quite similar to the original median. So if we have skewed data, as we do in this case with the outlier, median will provide a more meaningful interpretation of distance compared to mean. Thus, K-Medians clustering performs better in this case.

Let's see how the clustering goes over each iteration

In [ ]:
```python
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);
```



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.

```
In [ ]: 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
```

Out[ ]:

	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. The benefits of K-Medians are when we have non-symmetric data and/or data with many outliers since medians and Manhattan distance is more robust to these types of irregularities. In other cases, especially with data preprocessing, K-Means is preferred since it is usually faster with high-dimension data and has better convergance properties from the smooth nature of Euclidean distance gradient.

2. If a model performs better on K-Medians than K-Means, that would suggest that our dataset has some skew or outliers for the same reasoning elucidated in Question 2.

3. If we want to achieve better performance with K-Means, we could introduce some sort of threshold to limit the influence of outliers on our data. This could be done with filtering or adding a penalty to reduce the weight of outliers in SSE.

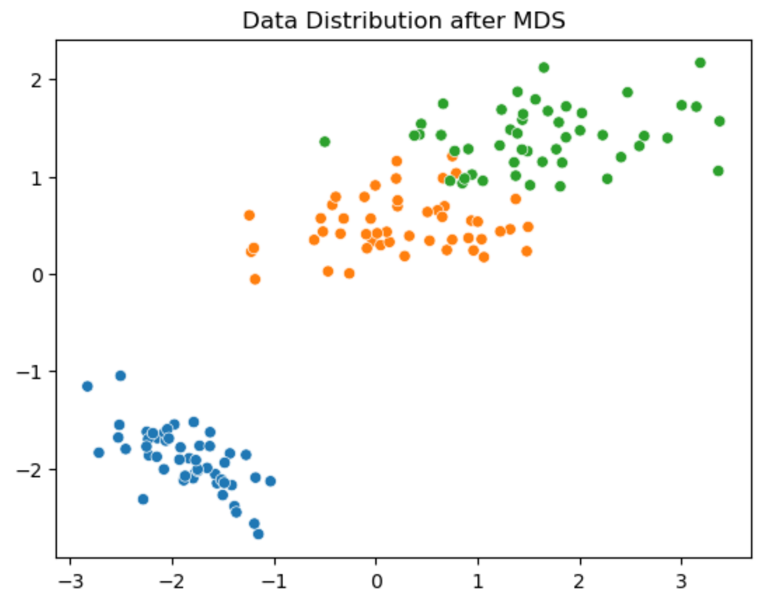
4. Two primary reasons for why Sklearn may perform better than our model are preprocessing and centroid intialization. Sklearn may use data preprocessing to identify outliers and use filtering or a penalty to reduce their influence on clustering. It may also have a more intelligent approach for centroid intializaition or repeat the clustering process with different centroid initializations to find a better SSE minimization.

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.

```
In [ ]: 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');
```

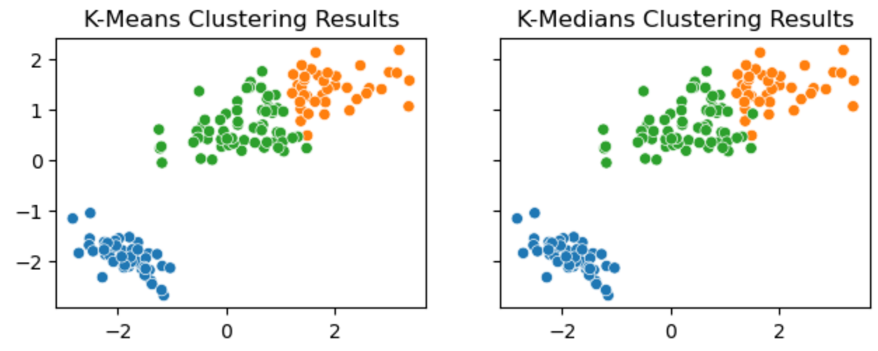
```
/Users/viren/anaconda3/lib/python3.10/site-packages/sklearn/manifold/_mds.py:299: FutureWarning: The default value of `normalized_stre
ss` will change to `'auto'` in version 1.4. To suppress this warning, manually set the value of `normalized_stress`.
  warnings.warn(
```



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

```
In [ ]: 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');
```

```
/Users/viren/anaconda3/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Early Stopped at Iteration 5
Early Stopped at Iteration 4
```



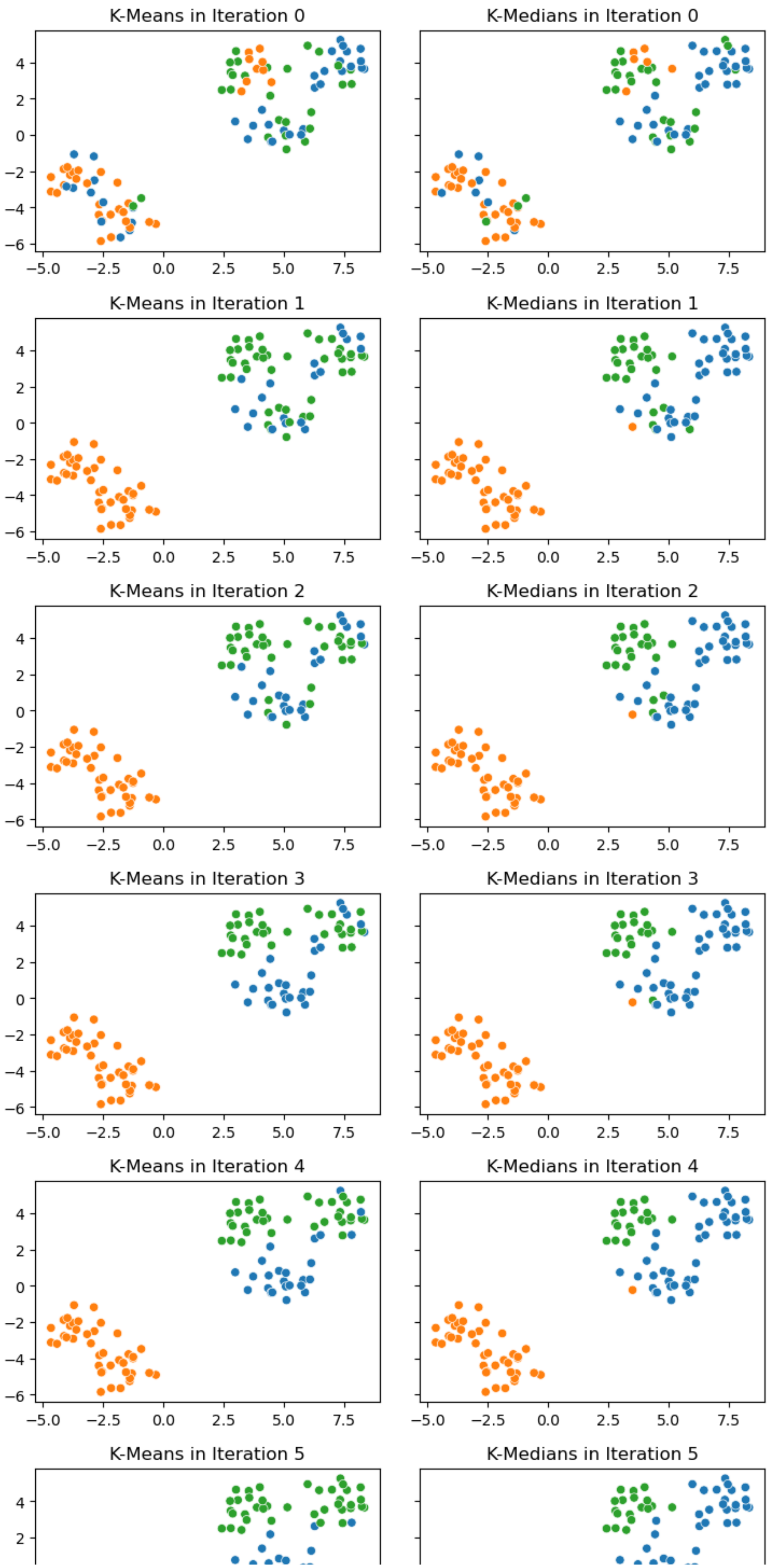
Question 4: From the above results, do our models still perform that well compared to the expereiment where we used the synthetic data? What makes the difference in real-life data?

Answer: Our models perform decently compared to the ground truth, but not as well as we observed with synthetic data. One of the main reasons for this is that there isn't a hard boundary between the green and orange classes in our ground truth. In our models' clustering output, we constructed a hard boundary between the two classes even though there is some overlap in the ground truth. So, the difference with real-world data is that it's likely to have outliers, noise, skewed variable distribution, or colinearity, which would work against the assumptions/requirements for K-Means/Medians clustering.

Let's see how the clustering goes over each iteration

```
In [ ]: 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);
```


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
4
2
0
−2
−4
−6
−5.0
−2.5
0.0
2.5
5.0
7.5

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.

In [ ]:
```
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
```

Out[ ]:

	Mutual Info Score
Reference KMeans from Sklearn vs Ground Truth	0.947515
Our KMeans vs Ground Truth	0.947515
Our KMedians vs Ground Truth	0.904241

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

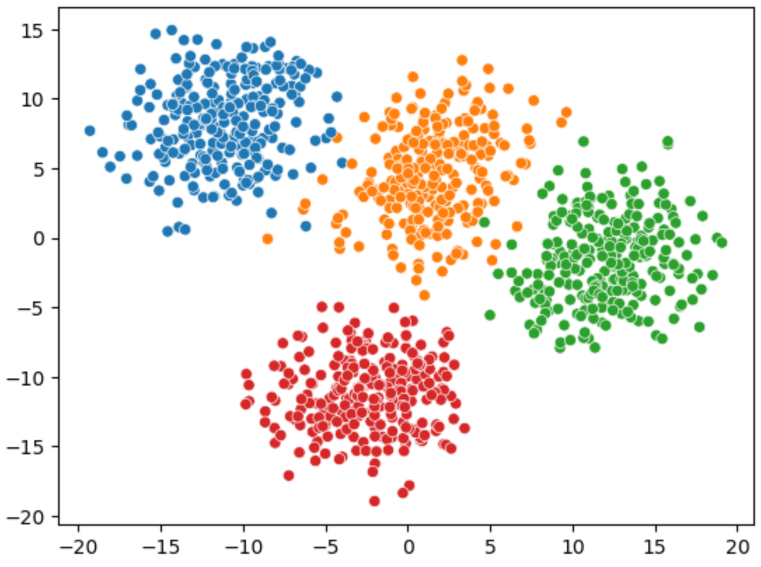
Let us turn back to the synthetic data. For this part of the experiment, there are 1000 points in our data, and it should have 4 features and 4 centers (or 4 different types/labels).

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

In [ ]:
```
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)
```

```
/Users/viren/anaconda3/lib/python3.10/site-packages/sklearn/manifold/_mds.py:299: FutureWarning: The default value of `normalized_stre
ss` will change to `'auto'` in version 1.4. To suppress this warning, manually set the value of `normalized_stress`.
  warnings.warn(
```

Out[ ]: `<Axes: >`



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.

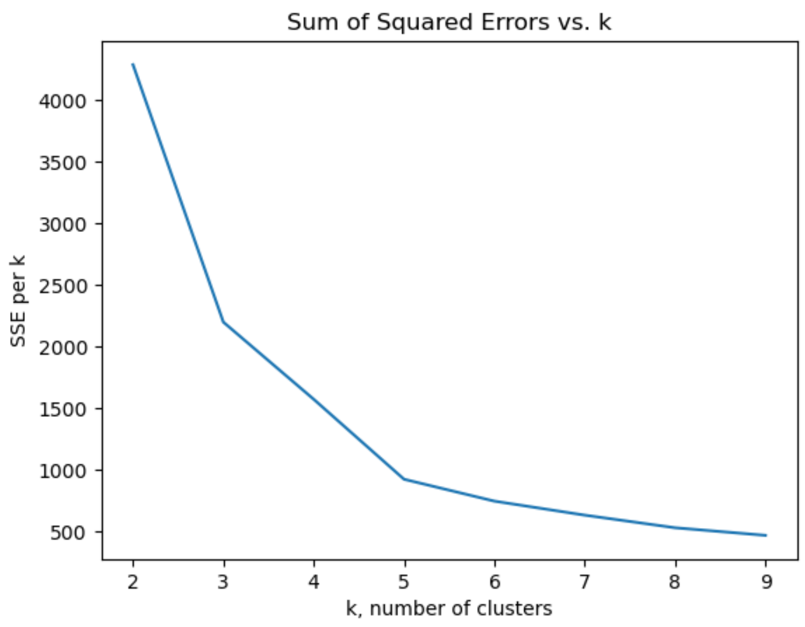
In [ ]:
```
SSE = []
y_preds = []

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.

```
In [ ]: 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')
```

```
Out[ ]: 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: Based on this metric, it suggests we should have 9 clusters, rather than the 4 clusters we were expecting. We cannot trust this metric because it is SSE per k, which would often lead us to pick the largest k in our range. The takeaway is that to find the optimal number of clusters, we need to identify the correct metric to measure the difference in performance.

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: One of the largest influences on clustering performance is centroid initialization. We know that K-Means will coverge to some local minimum, but there could be some other centroid initialization that would get us to a lower minimum. By running the clustering algorithm multiple times with different centroid initializations (perhaps with sampling to reduce computational complexity), we may be able to find better centroids. Another optimization would be to do some data preprocessing to do dimensionality reduction, penalize/drop outliers, or tranform the data to make it symmetric. One challenge with having many dimensions is that it may be hard to find strict differences in distance between data points. Processing outliers and making our data symmetric can be beneficial for the reasons stated in Questions 2 and 3.