Fall 2025 CS4641/CS7641 Homework 2 - Programming Section

Instructor: Dr. Max Mahdi Roozbahani, Dr. Nimisha Roy

Deadline: Friday, October 17th, 11:59 pm ET

- No unapproved extension of the deadline is allowed. For late submissions, please refer to the course website.
- Discussion is encouraged on Ed as part of the Q/A. We encourage whiteboard-level discussions about the homework. **However, all assignments should be done individually.**
- Plagiarism is a *serious offense*. **You are responsible for completing your own work.** You are not allowed to copy and paste, or paraphrase, or submit materials created or published by others, as if you created the materials. All materials submitted must be your own, and you must not collaborate with anyone or share your HW content except the ML instructional team.
- Working with a generative-AI platform *may constitute plagiarism*. In line with the Joyner Heuristic being used by many classes at GT, you should treat collaboration with generative-AI as collaboration with a knowledgeable peer. **Sharing the question or your work verbatim with an AI agent so as to generate an answer to the question is considered academic dishonesty and will be treated the same as any other incident of academic dishonesty.** If you find yourself turning to generative-AI for help answering a question, we suggest that you instead make use of Ed or TA office hours.
- Even using generative-AI for formatting your answers is *not permitted*. **While we understand that LaTeX has a learning curve, you may not use any generative-AI platform to improve your writing or LaTeX formatting.** These tools inherently produce output that may include a partial or complete solution to the question, including corrections to work you give it, even if purely prompted for syntactic use. Additionally, you have no custody over the data you give these platforms, which may leak or be used to inform subsequent training. Thus, while you are more than welcome to ask it about LaTeX commands and formatting tips, sharing the question or your answer to a question verbatim with an AI agent, even purely for syntactic use, is considered academic dishonesty and will be treated the same as any other incident of academic dishonesty. If you find yourself turning to generative-AI for help rewording your work to improve the language therein, remember that many of these questions will not be graded on language, but the content of your work. If you wish to improve it nonetheless, you can make use of the Georgia Tech Communications Lab or TA office hours. If you find yourself turning to generative-AI for help reformatting your LaTeX, we suggest that you instead use the resources in the instructions below or TA office hours. Ed is also an appropriate place to ask about LaTeX formatting, so long as your post doesn't reveal answers to a question (or make a private post if your question necessitates revealing answers).
- *All* incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures. **If we observe any (even small) similarities/plagiarisms detected by Gradescope or our TAs, we will directly report the case to OSI, which may, unfortunately, lead to a very harsh outcome, pending review. Consequences can be severe, including academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class.**

Instructions for the assignment

- This assignment consists of both programming and theory questions.
- Unless a theory question explicitly states that no work is required to be shown, you must provide an explanation, justification, or calculation for your answer.
- We will be using Gradescope for submission and grading of assignments.
- **Unless a question explicitly states that no work is required to be shown, you must provide an explanation, justification, or calculation for your answer.** Basic arithmetic can be combined (it does not need to each have its own step); your work should be at a level of detail that a TA can follow it.

- **For the "Assignment 2 - Non-programming" turn-in of this assignment, you will need to submit to Gradescope a PDF copy of your Jupyter Notebook with the cells ran.** Please refer to the Deliverables and Point Distribution section for an outline of the non-programming questions.
- When submitting your assignment on Gradescope, **you are required to correctly map pages of your PDF to each question/subquestion to reflect where your solutions appear in your PDF. Improperly mapped questions will receive a 0.** You are permitted to submit a regrade request in this event; however, the review of such a request is at the sole discretion of the instructional staff and is not likely to be accepted.
- **When submitting to Gradescope, please make sure to mark the page(s) corresponding to each problem/sub-problem. The pages in the PDF should be of size 8.5" x 11" (landscape or portrait), otherwise there may be a deduction in points for oversized sheets, up to and including full credit deducted.** Again, you are permitted to submit a regrade request in this event; however, the review of such a request is at the sole discretion of the instructional staff and is not likely to be accepted.
- All assignments should be done individually; each student must write up and submit their own answers.
- **Graduate Students**: You are required to complete any sections marked as Bonus for Undergrads.

Using the autograder

- Grads will find three assignments and Undergrads will find four assignments on Gradescope that correspond to HW2: "Assignment 2 Programming", "Assignment 2 - Non-programming", "Assignment 2 Programming - Bonus for all", and "Assignment 2 Programming - Bonus for Undergrad"
- You will submit your code for the autograder in the Assignment 2 Programming sections. Please refer to the Deliverables and Point Distribution section for what parts are considered required, bonus for undergrads, and bonus for all.
- We provided you different .py files and we added libraries in those files please DO NOT remove those lines and add your code after those lines. Note that these are the only allowed libraries that you can use for the homework.
- You are allowed to make as many submissions until the deadline as you like. Additionally, note that the autograder tests each function separately, therefore it can serve as a useful tool to help you debug your code if you are not sure of what part of your implementation might have an issue.

Using the local tests

- For some of the programming questions we have included a local test using a small toy dataset to aid in debugging. The local tests are all stored in localtests.py
- There are no points associated with passing or failing the local tests, you must still pass the autograder to get points.
- **It is possible to fail the local test and pass the autograder** since the autograder has a certain allowed error tolerance while the local test allowed error may be smaller. Likewise, passing the local tests does not guarantee passing the autograder.
- **You do not need to pass both local and autograder tests to get points, passing the Gradescope autograder is sufficient for credit.**
- It might be helpful to comment out the tests for functions that have not been completed yet.
- It is recommended to test the functions as it gets completed instead of completing the whole class and then testing. This may help in isolating errors. Do not solely rely on the local tests, continue to test on the autograder regularly as well.

Deliverables and Points Distribution

Q1: KMeans Clustering [45pts: 42pts + 3pts Grad / 1.5% Bonus for Undergrad]

Deliverables: kmeans.py

- **1.1 pairwise_dist** [5pts] - *programming*
- **1.2 KMeans Implementation** [30pts: 27pts + 3pts Grad / 1.5% Bonus for Undergrad] - *programming*
  - 1.2.1. init_centers [2pts]
  - 1.2.2. kmpp_init [3pts Grad / 1.5% Bonus for Undergrad]
  - 1.2.3. update_assignment [5pts]
  - 1.2.4. update_centers [5pts]
  - 1.2.5. get_loss [5pts]
  - 1.2.6. train [10pts]
- **1.3 Visualize KMeans** [0pts]
- **1.4 Clustering Metrics** [10pts] - *programming*
- **1.5 Limitation of KMeans** [0pts]

Q2: EM Algorithm [15pts + 1% Bonus for All]

Deliverables: Notebook Markdown Cell Text

- **2.1 Performing EM Update** [15pts] - *non-programming*
  - 2.1.1 [3pts] - *non-programming*
  - 2.1.2 [3pts] - *non-programming*
  - 2.1.3 [9pts] - *non-programming*
- **2.2 Gradient Descent and EM algorithm** [1% Bonus for All] - *non-programming*

Q3: GMM implementation [64pts + 2% Bonus for All]

Deliverables: gmm.py and Notebook Markdown Cell Text

- 3.1 Helper Functions [17pts] - *programming & non-programming*
  - 3.1.1. softmax [5pts]
  - 3.1.2. logsumexp [3pts + 4pts] - *programming & non-programming*
  - 3.1.3. normalPDF [5pts] - *for CS4641 students only*
  - 3.1.3. multinormalPDF [5pts] - *for CS7641 students only*
- 3.2 GMM Implementation [30pts] - *programming*
  - 3.2.1. init_components [5pts]
  - 3.2.2._ll_joint [10pts]

    - 3.2.3. Setup iterative steps for EM algorithm [15pts]
- 3.3 Image Compression and Pixel clustering [12pts] - *programming & non-programming*
    - 3.3.1. GMM Clustering in the RGB Space [10pts]
    - 3.3.2. Written Question [2pts]
- 3.4 Compare Full Convariance Matrix with Diagonal Covariance Matrix [1% Bonus for All] - *non-programming*
- 3.5 Generate samples from a Gaussian Mixture [5pts] - *non-programming*
- 3.6 Random vs. KMeans Initialization [1% Bonus for All] - *non-programming*

Q4: Cleaning Super Duper Messy data with semi-supervised learning [7.0% Bonus for All]

Deliverables: semisupervised.py and Notebook Markdown Cell Text

- 4.1: KNN [2.8% Bonus for All] - *programming*
    - 4.1.a. complete, *incomplete*, unlabeled_ [0.7% Bonus for All]
    - 4.1.b. CleanData __call__ [1.4% Bonus for All]
    - 4.1.c. MedianCleanData [0.7% Bonus for All]
- 4.2: Getting acquainted with semi-supervised learning approaches [3.5% Bonus for All] - *programming & non-programming*
    - 4.2.a. Write highlight summary [1.2% Bonus for All] - *non-programming*
    - 4.2.b. Implement EM algorithm [2.3% Bonus for All] - *programming*
- 4.3: Demonstrating the performance of the algorithm [0pts]
- 4.4: Interpretation of Results [0.7% Bonus for All] - *non-programming*

Note: It is highly recommended that you do Q4 (if not for the HW then before the project) as it teaches you imperfect data handling and a good understanding of how the models you have learnt can be used together for better results.

Q5: Hierarchical Clustering [9 pts Grad / 4.5% Bonus for Undergrad]

Deliverables: hierarchical_clustering.py

- 5.1 Hierarchical Clustering Implementation [9 pts Grad / 4.5% Bonus for Undergrad] - *programming*
- 5.2 Hierarchical Clustering Visualization [0 pts]
- 5.3 Hierarchical Clustering Large Dataset Visualization [0 pts]

Points Totals:

- Total Base: 133 pts for grads / 121 pts for undergrads
- Total Undergrad Bonus: 6%
- Total Bonus for All: 10%

Gradescope Submission Deliverables:

- For any Non-Programming portion: HW2.pdf
  - run all cells
  - use any utility to convert the Jupyter notebook to a pdf
  - on some systems, it may be easier to print it as an html (which VSCode natively supports), then render that to a pdf
- For 4641/7641 Programming Bonus for All: semisupervised.py
- For 7641 Programming: kmeans.py, gmm.py, hierarchical_clustering.py
- For 4641 Programming: kmeans.py, gmm.py
- For 4641 Programming Bonus For Undergrad: kmeans.py, hierarchical_clustering.py

0 Set up

This notebook is tested under python 3.11.**, and the corresponding packages can be downloaded from miniconda. You may also want to get yourself familiar with several packages:

- jupyter notebook
- numpy
- matplotlib

You can create a python conda environment with the necessary packages using the instructions in the `environment/environment_setup.md` file.

Please implement the functions that have "raise NotImplementedError", and after you finish the coding, please delete or comment "raise NotImplementedError".

```
[1]:
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from __future__ import absolute_import, division, print_function

%matplotlib inline

import sys

import localtests as localtests
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from mpl_toolkits.mplot3d import axes3d
from tqdm import tqdm

print("Version information")

print("python: {}".format(sys.version))
print("matplotlib: {}".format(matplotlib.__version__))
print("numpy: {}".format(np.__version__))

# Load image
import imageio

%load_ext autoreload
%autoreload 2
```

```
Version information
python: 3.13.5 | packaged by conda-forge | (main, Jun 16 2025, 08:27:50) [GCC 13.3.0]
matplotlib: 3.10.5
numpy: 2.3.2
```

1. KMeans Clustering [45pts total: 42pts + 3pts Grad / 1.5% Bonus for Undergrad]

KMeans is trying to solve the following optimization problem:

\begin{align} \arg \min_S \sum_{i=1}^K \sum_{x^{\{j\}} \in S_i} ||x^{\{j\}} - \mu_i||^2 \end{align} where one needs to partition the N observations into K clusters: $S = \{S_1, S_2, \ldots, S_K\}$ and each cluster has μ_i as its center.

1.1 Pairwise Distance [5pts]

In this section, you are asked to implement pairwise_dist function using the euclidean distance metric and the pairwise_dist_inf function using the infinity distance metric, also known as the chebyshev distance metric.

Given $X \in \mathbb{R}^{N \times D}$ and $Y \in \mathbb{R}^{M \times D}$, obtain the pairwise distance matrix $dist \in \mathbb{R}^{N \times M}$ using the euclidean distance metric, where $dist_{i, j} = ||X^{\{i\}} - Y^{\{j\}}||_2$.

For highly dimensional datasets, the pairwise distance matrix using the L_∞ (chebyshev) distance metric can be less impacted by the curse of dimensionality compared to the euclidean distance metric, and is effective for grid based systems. Obtain the L_∞ (chebyshev) distance $dist \in \mathbb{R}^{N \times M}$ using broadcasting, where $dist_{i, j} = ||X^{\{i\}} - Y^{\{j\}}||_\infty = \max^{d}_{k=1} |X^{\{i\}}_k - Y^{\{j\}}_k|$.

DO NOT USE LOOPS in your implementation, **using for-loops or while-loops will result in 0 credit for this portion.**

Hint: Use array broadcasting, but your implementation shouldn't create a third dimension (which would timeout). This can be achieved by using the $X^2 + Y^2 – 2XY$ shortcut calculation for the euclidean distance metric. Also notice that **a numpy array in shape $(N,1)$ is NOT the same as that in shape $(N,)$** so be careful and consistent on what you are using. You can see the detailed explanation here. Difference between numpy.array shape (R, 1) and (R,)

Hint: To calculate X^2 and Y^2 you can refer to the sum of squares function from assignment 1. For detailed explanation of pairwise distance function check this document - https://static.us.edusercontent.com/files/WLtSuk4PzW8e8M6VTsDq0BdJ

We have provided some unit tests in localtests.py for you to check your implementation. See Using the Local Tests for more details.

In [3]:
```
localtests.KMeansTests().test_pairwise_dist()
localtests.KMeansTests().test_pairwise_speed()
localtests.KMeansTests().test_pairwise_dist_inf()
localtests.KMeansTests().test_pairwise_speed_inf()
```

```
UnitTest passed successfully!
UnitTest passed successfully!
UnitTest passed successfully!
UnitTest passed successfully!
```

1.2 KMeans Implementation [30pts: 27pts + 3pts Grad / 1.5% Bonus for Undergrad]

In this section, you are asked to implement several methods in **kmeans.py**

You may use this visualization tool to refine your understanding of KMeans.

Initialization: [5pts: 2pts + 1.5% Bonus for Undergrad]

The Kmeans algorithm is sensitive to how the centers are initialized and a bad initialization can increase the time required for convergence or may even converge to a non-optimal solution, but we need to pick somehow.

1.2.1. **init_centers** [2pts]

As a naive approach, initialize the centers randomly by picking points from the dataset. Note that you should sample points without replacement/repetition, since we wouldn't want to pick the same point twice. This is actually a better approach than you might think. If there exists a large clustering of points in a region, and we're sampling all of the points uniformly, then there's a pretty good chance we'll pick a center in said region. The odds get worse with more clusters and more outliers, but it's a good place to start.

In **kmeans.py**, implement **init_centers**.

1.2.2. **kmpp_init** [3pts Grad / 1.5% Bonus for Undergrad]

One key intuition about ideal cluster centers is that a good starting point will have the centers be further away from each other.

In **kmeans.py**, implement **kmpp_init**, per the instructions below.

KMeans++

The algorithm for KMPP that you will implement (slightly different than the procedure described in its original paper) can be described as follows:

1. Sample 1% of the points from the dataset, uniformly at random (UAR) and without replacement. This sample will be the dataset the remainder of the algorithm uses to minimize initialization overhead.
2. From the above sample, select only one random point to be the first cluster center.
3. For each point in the sampled dataset, find the nearest cluster center and record the squared distance to get there.
4. Examine all the squared distances and take the point with the maximum squared distance as a new cluster center. In other words, we will choose the next center based on the maximum of the minimum calculated distance instead of sampling randomly like in step 2. You may break ties arbitrarily.
5. Repeat 3-4 until all k-centers have been assigned. You may use a loop over K to keep track of the data in each cluster.

1.2.3. Updating Cluster Assignments [5pts]

After you've chosen your centers, you will need to update the membership of each point based on the closest center. You will implement this in **update_assignment**. See docstring for more details.

1.2.4. Updating Centers Assignments [5pts]

Since cluster memberships may have changed, you will need to update the cluster centers.

In **kmeans.py**, implement **update_centers**. See docstring for more details.

Hint: You may use a loop over K to keep track of the data in each cluster, but avoid looping over N individual datapoints.

1.2.5. Loss & Convergence [5pts]

We will consider KMeans to be converged when the change in loss drops below a threshold value. The loss will be defined as the sum of the squared distances between each point and its respective center. Keep this in mind when writing the **train** function.

1.2.6. Train the model [10pts]

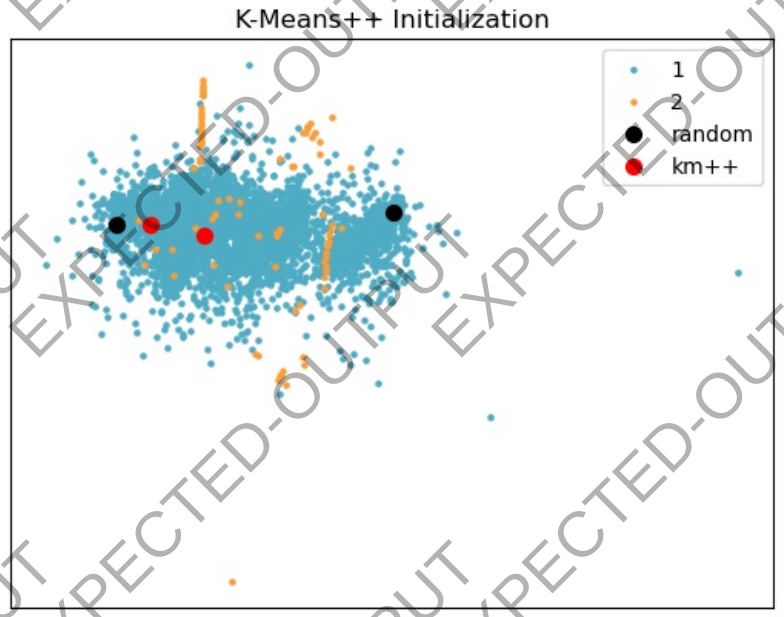
In the **train** method you will use all of the previously implemented steps to train your KMeans algorithm until convergence. Since the centers have already been initialized in the init function the general steps for the **train** method is as follows:

1. Update the cluster assignment for each point

2. Update the cluster centers based on the new assignments from Step 1
3. Check to make sure there is no mean without a cluster, i.e. no cluster center without any points assigned to it.
    - In the event of a cluster with no points assigned, pick a random point in the dataset to be the new center and update your cluster assignment accordingly.
4. Calculate the loss and check if the model has converged to break the loop early.
    - The convergence criteria is measured by whether the percentage of difference in loss with respect to the previous iteration's loss is less than the given relative tolerance threshold (self.rel_tol).
5. Iterate through steps 1 to 4 `max_iters` times. **Make sure to avoid infinite looping.**

We have provided the following local tests to help you check your implementation. Provided unit-tests are meant as a guide and are not intended to be comprehensive. See Using the Local Tests for more details.

In [3]:
```
localtests.KMeansTests().test_init()
localtests.KMeansTests().test_update_centers()
localtests.KMeansTests().test_kmeans_loss()
localtests.KMeansTests().test_train()
```




```
UnitTest passed successfully!
UnitTest passed successfully!
UnitTest passed successfully!
```

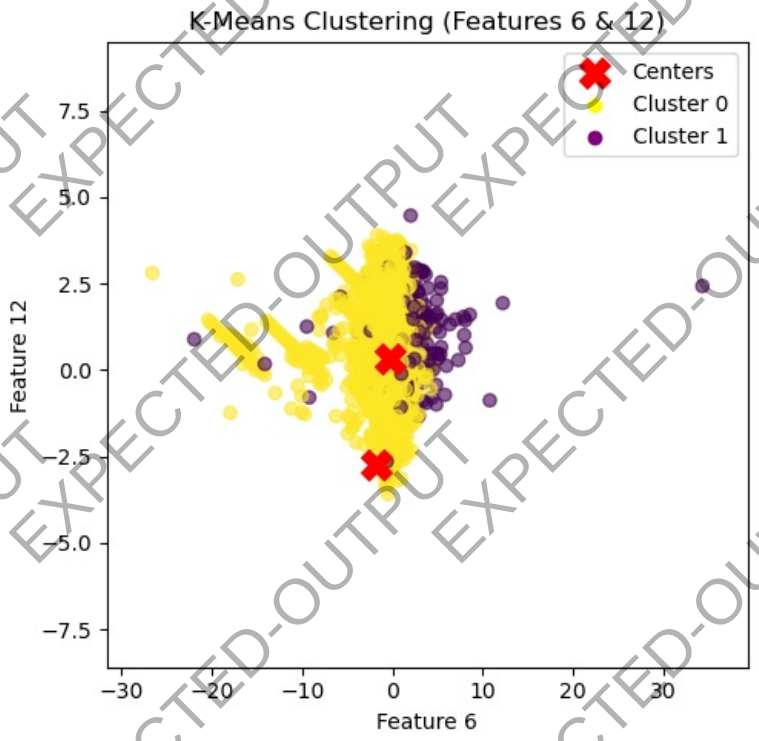
1.3 Visualize KMeans [0pts]

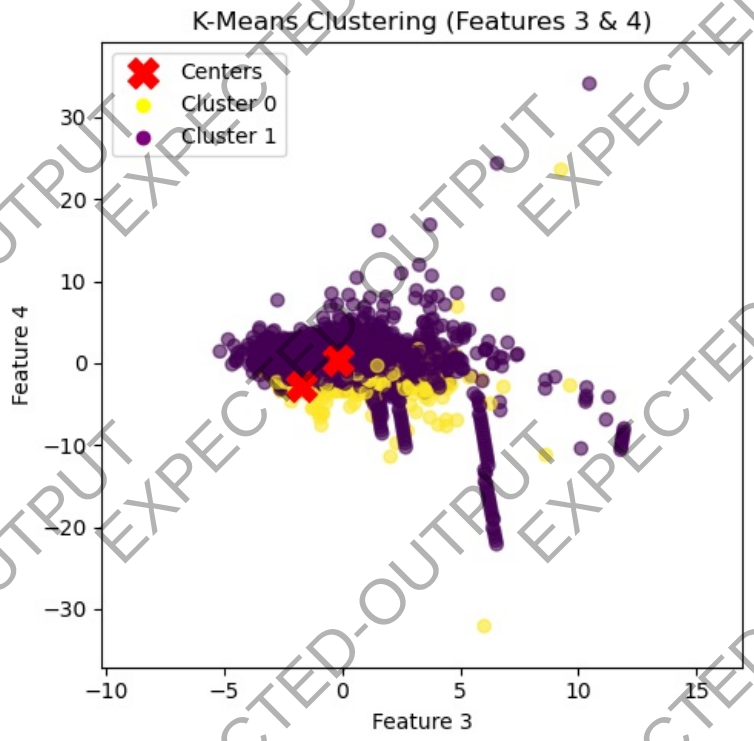
Cyber Sentinel Ava, a top fraud analyst in CyberHaven, is on a mission to protect the city's financial network from cybercriminals. The Central Credit Network (CCN) has detected unusual transaction patterns—small, frequent payments flowing through dormant accounts—suggesting that compromised accounts are being exploited. Remembering her expertise in Machine Learning, Ava knows that anomaly detection using K-Means clustering can uncover hidden fraud rings. Your task is to assist Ava in deploying this AI-driven system to identify compromised accounts and stop a billion-credit heist before it's too late

All you need to do is run the next cell. It should output different plots of a subset of selected features.

In [4]:
```
##############################
### DO NOT CHANGE THIS CELL ###
```

```
##############################

from utilities import *

# Note that because of a different file structure, students' paths will be different
data = pd.read_csv("./data/creditcard.csv")
X = data.iloc[:, data.columns != "Class"].to_numpy()
k = 2

create_plots(X, k)
```





1.4 Clustering Metrics [10pts]

In this section, you will calculate metrics to assess the quality of your clustering algorithm.

1. The **Adjusted Rand Index (ARI)** quantifies how consistently two clusterings group pairs of points, correcting for chance.
2. The **Silhouette Coefficient** is a measure of how well a data point fits within its assigned cluster compared to other clusters.

1.4.1 Adjusted Rand Index (ARI) [3 points]

As discussed in class, the computation for the Random Index is as follows:

$$RI = \frac{TP + TN}{TP + TN + FP + FN}$$

Using this value, we can calculate the Adjusted Random Index score, which is simply the random index score adjusted for chance.

$$ARI = \frac{RI - \mathbb{E}[\text{RI}]}{max(RI) - \mathbb{E}[\text{RI}]}$$

where

$$ max(RI) = 1 $$

$$\mathbb{E}[\text{RI}] = \frac{(TP+FP)(TP+FN) + (TN+FP)(TN+FN)}{(TP + TN + FP + FN)^2} $$

Substituting these values into the ARI formula, we see that

$$ARI = \frac{2(TP \cdot TN - FP \cdot FN)}{(TP + FN)(FN + TN) + (TP + FP)(FP + TN)}$$

TP (True Positive) represents the number of data points that are correctly clustered together in both the algorithm's result and the ground truth.

TN (True Negative) is the number of data points that are correctly placed in separate clusters in both the algorithm's result and the ground truth.

FP (False Positive) counts the number of data points that are incorrectly clustered together in the algorithm's result but correctly separated in the ground truth.

FN (False Negative) is the number of data points that are incorrectly separated in the algorithm's result but correctly clustered together in the ground truth.

We have provided the following local tests to help you check your implementation. Provided unit-tests are meant as a guide and are not intended to be comprehensive.
See Using the Local Tests for more details.

1.4.2 Silhouette Coefficient [5 points]

As discussed in class, the computation for the Silhouette Coefficient is as follows:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), \, b(i)\}}$$

where

$a(i)$ = average distance of point i to all other points in its own cluster

$b(i)$ = minimum average distance of point i to all points in any other cluster

Thus, the overall silhouette coefficient can be computed as the average of the silhouette coefficient for all points:

$$ S = \frac{1}{N} \sum_{i=1}^{N} s(i) $$

Refer to the class notes for more information on the Silhouette Coefficient

Refer to the class notes and sklearn documentation for more information on the Adjusted Random Index Measure

In [5]:
```
localtests.KMeansTests().test_silhouette_score()
localtests.KMeansTests().test_adjusted_rand_statistic()
```

```
Expected value:  -0.1121462906724291
Your value:  -0.11214629067242914

UnitTest passed successfully!
Expected value:  0.18331805682859761
Your value:  0.18331805682859761

UnitTest passed successfully!
```

1.4.3 Comparison (Written Question) [2 points]

Analyze the performance of these two evaluation metrics: Adjusted Rand Index (ARI) and Silhouette Coefficient (SC). Limit your response to 2-3 sentences MAX per

question.

a. Explain what each of these metrics measure. For each metric, describe what constitutes a “good” score and a “bad” score. Include the range of possible values (1 point)

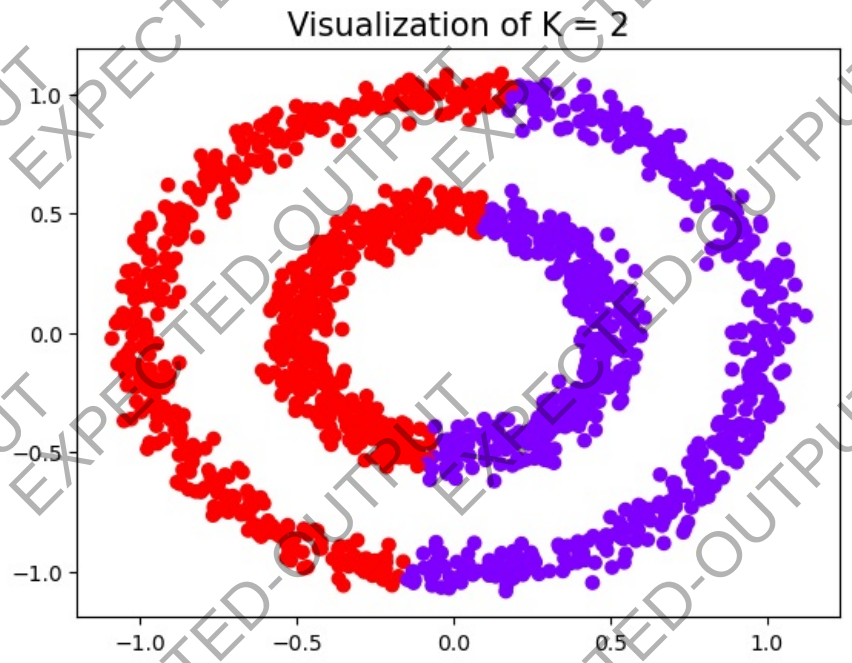
b. Identify the most notable difference between these two metrics and explain why this distinction is important when evaluating clustering results (1 point)

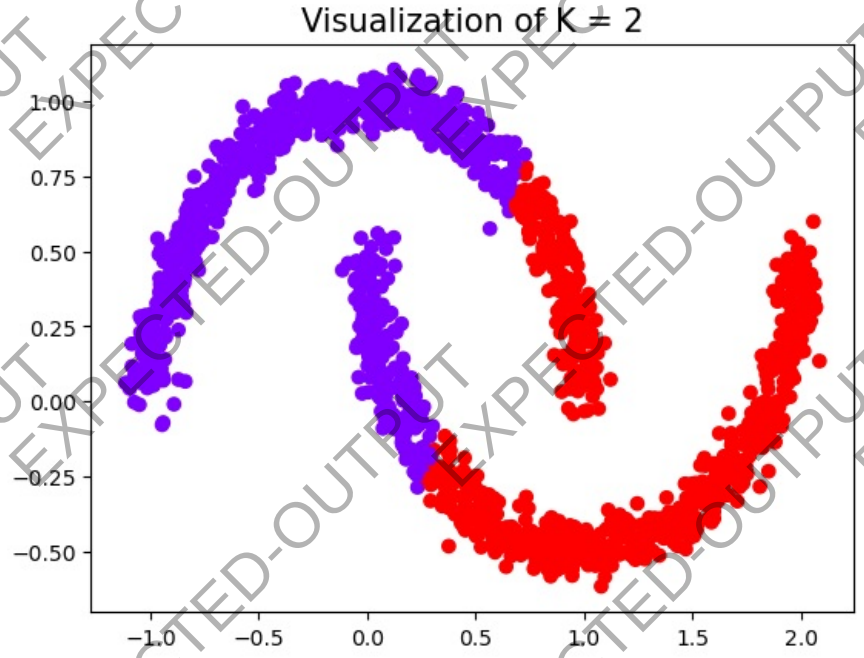
1.5 Limitation of KMeans [0pts]

You've now done the best you can selecting the perfect starting points and the right number of clusters. However, one of the limitations of K-Means Clustering is that it depends largely on the shape of the dataset. A common example of this is trying to cluster one circle within another (concentric circles). A K-means classifier will fail to do this and will end up effectively drawing a line that crosses the circles. You can visualize this limitation in the cell below.

```
[ ]:
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# visualize limitation of kmeans
from kmeans import *
from sklearn.datasets import make_circles, make_moons

X1, y1 = make_circles(factor=0.5, noise=0.05, n_samples=1500)
X2, y2 = make_moons(noise=0.05, n_samples=1500)


def visualise(
    X, C, K=None
):  # Visualization of clustering. You don't need to change this function
    fig, ax = plt.subplots()
    ax.scatter(X[:, 0], X[:, 1], c=C, cmap="rainbow")
    if K:
        plt.title("Visualization of K = " + str(K), fontsize=15)
    plt.show()
    pass


kmeans = KMeans(X1, 2)
centers1, cluster_idx1, loss1 = kmeans.train()
visualise(X1, cluster_idx1, 2)
kmeans = KMeans(X2, 2)
centers2, cluster_idx2, loss2 = kmeans.train()
visualise(X2, cluster_idx2, 2)
```





2. EM algorithm [15pts + 1% Bonus for All]

2.1 Performing EM Update [15 pts]

ANSWERS CANNOT BE HANDWRITTEN

A univariate Gaussian Mixture Model (GMM) has two components, both of which have their own mean and standard deviation. The model is defined by the following parameters: $$ \mathbf{z} \sim Bernoulli(\alpha) = \begin{cases} \alpha &\text{if} \, z=0 \\ 1-\alpha &\text{if} \, z=1 \\\end{cases}$$ $$ p\left(\mathbf{x_n \mid z=0}\right) \sim \mathcal{N}(3\upsilon, 4\omega^{2}) $$ $$ p\left(\mathbf{x_n \mid z=1}\right) \sim \mathcal{N}(2\upsilon, 7\omega^{2}) $$

For a dataset of N datapoints, find the following:

2.1.1. Write the marginal probability of x, i.e. $p(x)$. [3pts]
-- Express your answers in terms of $\mathcal{N}(x|3\upsilon, 4\omega^{2})$ and $\mathcal{N}(x|2\upsilon, 7\omega^{2})$ may be simpler.
-- HINT: For this question suppose we have a Gaussian Distribution $\mathcal{N}(3\upsilon, 4\omega^{2})$, it means $\mathcal{N}(\mu = 3\upsilon, \sigma^{2} = 4\omega^{2})$.
-- HINT: Start with the Sum Rule.

2.1.2. E-Step: Compute the posterior probabilities, i.e, $p(z_0|x), p(z_1|x)$ [3pts]
-- Express your answers in terms of $\mathcal{N}(x|3\upsilon, 4\omega^{2})$ and $\mathcal{N}(x|2\upsilon, 7\omega^{2})$.
-- HINT: Try to apply Bayes Rule.

2.1.3. M-Step: Compute the updated value of ω^{2}. (You can keep μ fixed when you calculate the derivative.) [9pts]
-- Note that ω^2 is a shared variable between the two distributions, your final answer should be one equation including both Gaussian distributions.
-- Express your answers in terms of τ, x, and υ (you will need to expand $\mathcal{N}(3\upsilon, 4\omega^{2})$ and $\mathcal{N}(2\upsilon, 7\omega^{2})$ into its PDF form).
-- HINT: Start from the below equation, note that θ is shorthand for various variables, and take the derivative w.r.t. ω^2.

$$ \begin{align*} \ell(\theta|x) &= \sum_n^{N} \sum^{Z}_{k \in \{0, 1\}}p(z_k \mid x^{\{n\}},\theta_{old})ln \left[p(x^{\{n\}},z_k \mid \theta)\right]\\ \ell(\upsilon, \omega^2, \alpha \mid x) &= \sum_n^{N} \sum^{Z}_{k \in \{0, 1\}} p(z_k \mid x^{\{n\}}, \theta_{old}) \ln\left[p(x^{\{n\}}, z_k \mid \mu_k, \sigma^2_k, \alpha)\right] \\ &= \sum_n^{N} \sum^{Z}_{k \in \{0, 1\}} p(z_k \mid x^{\{n\}}, \theta_{old}) \ln \left[p(z_k \mid \alpha) p(x^{\{n\}} \mid z_k, \upsilon, \omega^2)\right] \end{align*} $$

Recall that $p(x^{\{n\}} \mid z_k, \upsilon, \omega^2) \rightarrow \mathcal{N}(x^{\{n\}} \mid \mu_k, \sigma_k)$ has been defined at the beginning of the problem.

You can refer to this lecture to gain an understanding of the EM Algorithm.

2.2 Gradient Ascent and EM algorithm [1% Bonus for All]

What is the computational advantage of using the EM algorithm compared to the Gradient Ascent algorithm for the problem presented in 2.1? Please provide your own qualitative analysis. [1% Bonus for All]
-- HINT: Think about the difference in updating parameters during each iteration. i.e., How many parameters are updated (with a step size) in gradient ascent each iteration? What do we do in each EM iteration (E-step then M-step) to simplify updates?

3. GMM implementation [60pts total: 60pts + 2% Bonus for All]

Please make sure to read the problem setup in detail. Many questions for this section may have already been answered in the description and hints and docstrings.

A Gaussian Mixture Model (GMM) is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian Distribution. In a nutshell, GMM is a soft clustering algorithm in a sense that each data point is assigned to a cluster with a probability. In order to do that, we need to convert our clustering problem into an inference problem.

Given N samples $X = [x^{\{1\}}, x^{\{2\}}, \ldots, x^{\{N\}}]^T$, where $x^{\{i\}} \in \mathbb{R}^D$. Let π be a K-dimensional probability density function and $(\mu_k; \Sigma_k)$ be the mean and covariance matrix of the k^{th} Gaussian distribution in $\mathbb{R}^D$.

The GMM object implements EM algorithms for fitting the model and MLE for optimizing its parameters. It also has some particular hypothesis on how the data was generated:

- Each data point $x^{\{i\}}$ is assigned to a cluster k with probability of π_k where $\sum_{k=1}^K \pi_k = 1$
- Each data point $x^{\{i\}}$ is generated from Multivariate Normal Distribution $\cal{N}(\mu_k, \Sigma_k)$ where $\mu_k \in \mathbb{R}^D$ and $\Sigma_k \in \mathbb{R}^{D\times D}$

Our goal is to find a K-dimension Gaussian distributions to model our data X. This can be done by learning the parameters π, μ and Σ through likelihood function. Detailed derivation can be found in our slide of GMM. The log-likelihood function now becomes:

$$\begin{align} \text{ln } p(x^{\{1\}}, \dots, x^{\{N\}} | \pi, \mu, \Sigma) = \sum_{i=1}^N \text{ln } \big(\sum_{k=1}^{K} \pi_k \mathcal{N}(x^{\{i\}} | \mu_k, \Sigma_k)\big) \end{align}$$

From the lecture we know that MLEs for GMM all depend on each other and the responsibility τ. Thus, we need to use an iterative algorithm (the EM algorithm) to find the estimate of parameters that maximize our likelihood function. **All detailed derivations can be found in the lecture slide of GMM.**

- **E-step:** Evaluate the responsibilities

In this step, we need to calculate the responsibility τ, which is the conditional probability that a data point belongs to a specific cluster k if we are given the datapoint, i.e. $P(z_k|x)$. The formula for τ is given below:

$$\tau\left(z_k\right)=\frac{\pi_{k} \cal{N}\left(x | \mu_{k}, \Sigma_{k}\right)}{\sum_{j=1}^{K} \pi_{j} \cal{N}\left(x | \mu_{j}, \Sigma_{j}\right)}, \quad \text{for } k = 1, \dots, K$$

Note that each data point should have one probability for each component/cluster. For this homework, you will work with $\tau\left(z_k\right)$ which has a size of $N\times K$ and you should have all the responsibility values in one matrix.

- **M-step:** Re-estimate Paramaters

After we obtained the responsibility, we can find the update of parameters, which are given below:

$$\begin{align} \mu_k^{new} &= \dfrac{\sum_{n=1}^N \tau(z_k)x^{\{n\}}}{N_k} \\ \Sigma_k^{new} &= \dfrac{1}{N_k}\sum_{n=1}^N \tau (z_k)^T(x^{\{n\}} - \mu_k^{new})^T(x^{\{n\}}-\mu_k^{new}) \\ \pi_k^{new} &= \dfrac{N_k}{N} \end{align}$$ where $N_k = \sum_{n=1}^N \tau(z_k)$. Note that the updated value for μ_k is used when updating Σ_k. The multiplication of $\tau (z_k)^T(x^{\{n\}} - \mu_k^{new})^T$ is element-wise so it will preserve the dimensions of $(x^{\{n\}} - \mu_k^{new})^T$.

- We repeat E and M steps until the incremental improvement to the likelihood function is small.

Special Notes

- For undergraduate student: you may assume that the covariance matrix Σ is diagonal matrix, which means the features are independent. (i.e. the red intensity of a pixel is independent from its blue intensity, etc). Make sure you set **FULL_MATRIX = False** before you submit your code to Gradescope.
- For graduate student: please assume full covariance matrix. Make sure you set **FULL_MATRIX = True** before you submit your code to Gradescope
- Under numpy conventions, the class notes would have your dataset X as (D, N) but **the homework dataset is (N, D) as mentioned on the instructions, so the formula is a little different from the lecture note in order to obtain the right dimensions of parameters.**

Hints

1. **DO NOT USE FOR LOOPS OVER N.** You can always find a way to avoid looping over the observation datapoints in our homework problem. If you have to loop over D or K, that is fine.
2. You can initiate π_k the same for each k, i.e. $\pi_k = \frac{1}{K}, \forall k = 1, 2, \ldots, K$.
3. In part 3 you are asked to generate the model for pixel clustering of image. We will need to use a multivariate Gaussian because each image will hava N pixels and $D=3$ features which corresponds to red, green, and blue color intensities. It means that each image is a $(N\times3)$ dataset matrix. In the following parts, remember $D=3$ in this problem.
4. To avoid using for loops in your code, we recommend you take a look at the concept Array Broadcasting in Numpy. Also, certain calculations that required different shapes of arrays can also be achieved by broadcasting.
5. Be careful of the dimensions of your parameters. Before you test anything on the autograder, please look at the instructions below on the shapes of the variables you need to output and how to format your return statement. Print the shape of an array by print(array.shape) could enhance the functionality of your code and help you debugging. Also notice that **a numpy array in shape $(N,1)$ is NOT the same as that in shape $(N,)$** so be careful and consistent on what you are using. You can see the detailed explanation here. Difference between numpy.array shape (R, 1) and (R,)

- The dataset X: (N, D)
- μ: (K, D).
- Σ: (K, D, D)
- τ: (N, K)
- π: array of length K
- ll_joint: (N, K)

3.1 Helper functions [17pts]

To facilitate some of the operations in the GMM implementation, we would like you to implement the following three helper functions. In these functions, "logit" refers to an input array of size (N, D) that reperesents the unnormalized scores, that are passed to the softmax( ) or logsumexp( ) function. Remember the goal of helper functions is to facilitate our calculation so **DO NOT USE FOR LOOP OVER N**.

3.1.1. softmax [5pts]

Given $logit \in \mathbb{R}^{N \times D}$, calculate $prob \in \mathbb{R}^{N \times D}$, where $prob_{i, j} = \frac{\exp(logit_{i, j})}{\sum_{d=1}^D exp(logit_{i, d})}$.

Notes:

- $logit$ here refers to the unnormalized scores that are passed in as a parameter to the softmax function. The softmax operation normalizes these scores, resulting in them having values between 0 and 1. This allows us to interpret the normalized scores as a probability distribution over the classes.
- It is possible that $logit_{i, j}$ is very large, making $\exp(\cdot)$ of it to explode. To make sure it is numerically stable, for each row of $logits$ subtract the maximum of that row.
  - By property of Softmax equation, subtracting a constant value does not change the output. Refer to Mathematical properties

Special Notes

- Do not add back the maximum for each row.
- Add **keepdims=True** in your np.sum() function to avoid broadcast error.

3.1.2. logsumexp [3pts Programming + 4pts Written Questions]

Given $logit \in \mathbb{R}^{N \times D}$, calculate $s \in \mathbb{R}^N$, where $s_i = \log \big(\sum_{j=1}^D \exp(logit_{i, j}) \big)$. Again, pay attention to the numerical problem. You may face similar conditions to the softmax function due to $logit_{i, j}$ being large. However, unlike softmax, subtracting the row maxes will affect the result. To adjust the output for correctness, after calculating the logsumexp of each row, you should add the maximum for each row of $logit$ back for your functions before returning the final value.

Special Notes

- This function is used in the call() function, which is given, and helps calculate the loss of log-likelihood. You will not have to call it in functions that you are required to implement.

Written Questions [4pts]:

For each row of logits we subtract the max, then calculate the logsumexp, then add the max back. In this question, you'll explore why this works.

1. Show (algebraically) how adding the maximum for each row $logit$ will return a correct result. More formally:

Let l be a row of D logits: $$l = \{l_1, l_2, \cdots, l_D\}$$ Let s^* be the result of logsumexp applied directly to l: $$s^* = \log \left(\sum_{i=1}^D \exp(l_i) \right)$$ Let s' be the result of logsumexp with our technique, subtracting the max before, adding the max after. $$s' = \log \left(\sum_{j=1}^D \exp\left(l_i - \max_{j\in[1,D]}[l_j]\right) \right) + \max_{j\in[1,D]}[l_j]$$

You need to show that $s'=s^*$.

2. Why is numeric stability important when implementing functions like softmax and logsumexp in the GMM algorithm? Briefly describe how unstable computations could affect the model's outputs or training process.

3.1.3. Multivariate Gaussian PDF [5pts]

You should be able to write your own function based on the following formula, and you are **NOT allowed** to use outside resource packages other than those we provided.

(for undergrads only) normalPDF

Using the covariance matrix as a diagonal matrix with variances of the individual variables appearing on the main diagonal of the matrix and zeros everywhere else means that we assume the features are independent. In this case, the multivariate normal density function simplifies to the expression below: $$\mathcal{N}(x: \mu, \Sigma) = \prod_{i=1}^D \frac{1}{\sqrt{2\pi\sigma_i^2}}\exp{\left(-\frac{1}{2\sigma_i^2} (x_i-\mu_i)^2\right)}$$ where σ^2_i is the variance for the i^{th} feature, which is the diagonal element of the covariance matrix.

(for grads only) multinormalPDF

Given the dataset $X \in \mathbb{R}^{N \times D}$, the mean vector $\mu \in \mathbb{R}^{D}$ and covariance matrix $\Sigma \in \mathbb{R}^{D \times D}$ for a multivariate Gaussian distrubution, calculate the probability $p \in \mathbb{R}^{N}$ of each data. The PDF is given by $$\mathcal{N}(X: \mu, \Sigma) = \frac{1}{(2\pi)^{D/2}}|\Sigma|^{-1/2}\exp{\left(-\frac{1}{2}(X-\mu)\Sigma^{-1}(X-\mu)^T\right)}$$ where $|\Sigma|$ is the determinant of the covariance matrix.

Hints:

- If you encounter "LinAlgError", you can mitigate your number/array by summing a small value before taking the operation, e.g. np.linalg.inv(Σ_k + SIGMA_CONST). You can arrest and handle such error by using Try and Exception Block in Python. Please only add `SIGMA_CONST` to all elements when `sigma_i` is

not invertible.

- In the above calculation, you must avoid computing a (N,N) matrix. Using the above equation for large N will crash your kernel and/or give you a memory error on Gradescope. Instead, you can do this same operation by calculating $(X-\mu)\Sigma^{-1}$, a (N,D) matrix, transpose it to be a (D,N) matrix and do an element-wise multiplication with $(X-\mu)^T$, which is also a (D,N) matrix. Lastly, you will need to sum over the 0 axis to get a $(1,N)$ matrix before proceeding with the rest of the calculation. This uses the fact that doing an element-wise multiplication and summing over the 0 axis is the same as taking the diagonal of the (N,N) matrix from the matrix multiplication.

- In Numpy implementation for each individual μ, you can either use a 2-D array with dimension $(1,D)$ for each Gaussian Distribution, or a 1-D array with length D. Same to other array parameters. Both ways should be acceptable but pay attention to the shape mismatch problem and be **consistent all the time** when you implement such arrays.

- Please ~~**DO NOT**~~ use `self.D` in your implementation of `multinormalPDF()` .

3.2 GMM Implementation [30pts]

Things to do in this problem:

3.2.1. Initialize parameters in _init_components() [5pts]

Examples of how you can initialize the parameters.

1. `create_pi()` : Set the prior probability π the same for each class.
2. `create_mu()` : Initialize μ by randomly selecting K numbers of observations as the initial mean vectors. You should use np.random.choice() with replacement set to True for random selection of K numbers of observations.
3. `create_sigma()` : Initialize the covariance matrix with np.eye() for each k. For grads, you can also initialize the Σ by K diagonal matrices. It will become a full matrix after one iteration, as long as you adopt the correct computation.
4. You are expected to call these methods in the `_init_components()` method
5. The autograder will only test the shape of your π,μ, σ. Make sure you pass other evaluations in the autograder.

3.2.2. Formulate the log-likelihood function _ll_joint() [10pts]

The log-likelihood function is given by:

$$ \begin{align} \ell(\theta) = \sum_{i=1}^N \text{ln } \big(\sum_{k=1}^{K} \pi_k \mathcal{N}(x^{\{i\}} | \mu_k, \Sigma_k)\big) \end{align} $$

In this part, we will generate a (N,K) matrix where each datapoint $x^{\{i\}}, \forall i = 1, \dots, N$ has K log-likelihood numbers. Thus, for each $i = 1, \dots, N$ and $k = 1, \dots, K$,

$$ \text{log-likelihood}[i,k] = \log{\pi_k}+\log{\cal{N}(x^{\{i\}}|\mu_k, \Sigma_k)} $$

Hints:

- If you encounter "ZeroDivisionError" or "RuntimeWarning: divide by zero encountered in log", you can mitigate your number/array by summing a small value before taking the operation, e.g. $\text{log-likelihood}[i,k] = \log{(\pi_k + \text{1e-32})}+\log{(\cal{N}(x^{\{i\}}|\mu_k, \Sigma_k) + \text{1e-32})}$. If you pass the local test cases but fail the autograder, make sure you sum a small value like the example we given.
- You need to use the Multivariate Normal PDF function you created in the last part. Remember the PDF function is for each Gaussian Distribution (i.e. for each k) so you need to use a for loop over K.

3.2.3. Setup Iterative steps for EM Algorithm [5pts + 10pts]

You can find the detail instruction in the above description box.

Hints:

- For E steps, we already get the log-likelihood at _ll_joint() function. This is not the same as responsibilities (τ), but you should be able to finish this part with just a few lines of code by using _ll_joint() and softmax() defined above.
- For undergrads: Try to simplify your calculation for Σ in M steps as you assumed independent components. Make sure you are only taking the diagonal terms of your calculated covariance matrix.

Function Tests

Use these to test if your implementation of functions in GMM work as expected. See Using the Local Tests for more details.

In [7]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from gmm import *
from utilities import plot_images

gmm_tester = localtests.GMMTests()
```

In [8]:
```
gmm_tester.test_helper_functions()
gmm_tester.test_init_components()
```

```
Your softmax works within the expected range:  True
Your logsumexp works within the expected range:  True
Your _init_component's pi works within expected range:  True
Your _init_component's mu works within expected range:  True
Your _init_component's sigma works within the expected range:  True
```

In [9]:
```
gmm_tester.test_undergrad()
```

```
Your normal_pdf works within the expected range:  True
Your lljoint works within the expected range:  True
Your E step works within the expected range:  True
Your M step works within the expected range:  True
```

In [10]:
```
gmm_tester.test_grad()
```

```
Your multinormal_pdf works within the expected range:  True
Your lljoint works within the expected range:  True
Your E step works within the expected range:  True
Your M step works within the expected range:  True
```

3.3 Image Compression and pixel clustering [12pts]

3.3.1 GMM Clustering in the RGB Space [10pts]

Images typically need a lot of bandwidth to be transmitted over the network. In order to optimize this process, most image processors perform lossy compression of images (lossy implies some information is lost in the process of compression).

In this section, you will use your GMM algorithm implementation to do pixel clustering and compress the images. That is to say, you would develop a lossy image compression algorithm. This question is autograded based on your GMM implementation. (Hint: you can adjust the number of clusters formed and justify your answer

based on visual inspection of the resulting images or on a different metric of your choosing)

Implement the `cluster_pixels_gmm` function in gmm.py. Each pixel can be considered as a separate data point (of length 3), which you can then cluster using GMM. Then, process the outputs into the shape of the original image, where each pixel is its most likely value. What do μ and τ represent?

Special Notes

- Try to add a small value(e.g. SIGMA_CONST and LOG_CONST) before taking the operation if the output image is solid black.
- The output images may be slightly different due to different initialization methods in GMM() function.
- Sample with replacement in `create_mu`
- Undergrads are tested with FULL_MATRIX = False and Grads are tested with FULL_MATRIX = True.

In [11]:
```
gmm_tester.test_undergrad_image_compression()
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
iter 0, loss: 9075057.7382:   0%|          | 0/10 [00:03<?, ?it/s]
iter 0, loss: 9075057.7382:  10%|▉         | 1/10 [00:03<00:32,  3.62s/it]
iter 1, loss: 8733594.6897:  10%|▉         | 1/10 [00:07<00:32,  3.62s/it]
iter 1, loss: 8733594.6897:  20%|█▊        | 2/10 [00:07<00:29,  3.65s/it]
iter 2, loss: 8265664.0425:  20%|█▊        | 2/10 [00:10<00:29,  3.65s/it]
iter 2, loss: 8265664.0425:  30%|██▋       | 3/10 [00:10<00:25,  3.64s/it]
iter 3, loss: 7949217.6939:  30%|██▋       | 3/10 [00:14<00:25,  3.64s/it]
iter 3, loss: 7949217.6939:  40%|███▌      | 4/10 [00:14<00:21,  3.62s/it]
iter 4, loss: 7725034.0950:  40%|███▌      | 4/10 [00:18<00:21,  3.62s/it]
iter 4, loss: 7725034.0950:  50%|████▌     | 5/10 [00:18<00:18,  3.63s/it]
iter 5, loss: 7599263.2824:  50%|████▌     | 5/10 [00:21<00:18,  3.63s/it]
iter 5, loss: 7599263.2824:  60%|█████▍    | 6/10 [00:21<00:14,  3.63s/it]
iter 6, loss: 7516630.0913:  60%|█████▍    | 6/10 [00:25<00:14,  3.63s/it]
iter 6, loss: 7516630.0913:  70%|██████▎   | 7/10 [00:25<00:10,  3.58s/it]
iter 7, loss: 7449778.9646:  70%|██████▎   | 7/10 [00:28<00:10,  3.58s/it]
iter 7, loss: 7449778.9646:  80%|███████▏  | 8/10 [00:28<00:07,  3.58s/it]
iter 8, loss: 7395953.5072:  80%|███████▏  | 8/10 [00:32<00:07,  3.58s/it]
iter 8, loss: 7395953.5072:  90%|████████  | 9/10 [00:32<00:03,  3.59s/it]
iter 9, loss: 7347498.8695:  90%|████████  | 9/10 [00:36<00:03,  3.59s/it]
iter 9, loss: 7347498.8695: 100%|█████████| 10/10 [00:36<00:00,  3.58s/it]
iter 9, loss: 7347498.8695: 100%|█████████| 10/10 [00:36<00:00,  3.60s/it]
Your image compression within the expected range:  True
```

In [12]:
```
gmm_tester.test_grad_image_compression()
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
iter 0, loss: 8799310.1760:   0%|          | 0/10 [00:03<?, ?it/s]
iter 0, loss: 8799310.1760:  10%|▉         | 1/10 [00:03<00:33,  3.68s/it]
iter 1, loss: 8472039.4533:  10%|▉         | 1/10 [00:07<00:33,  3.68s/it]
iter 1, loss: 8472039.4533:  20%|█▊        | 2/10 [00:07<00:30,  3.81s/it]
iter 2, loss: 8047534.2466:  20%|█▊        | 2/10 [00:11<00:30,  3.81s/it]
iter 2, loss: 8047534.2466:  30%|██▋       | 3/10 [00:11<00:26,  3.79s/it]
iter 3, loss: 7771737.7790:  30%|██▋       | 3/10 [00:15<00:26,  3.79s/it]
iter 3, loss: 7771737.7790:  40%|███▌      | 4/10 [00:15<00:22,  3.76s/it]
```

```
iter 4, loss: 7543929.4043:  40%|████      | 4/10 [00:18<00:22,  3.76s/it]
iter 4, loss: 7543929.4043:  50%|█████     | 5/10 [00:18<00:18,  3.76s/it]
iter 5, loss: 7344542.8870:  50%|█████     | 5/10 [00:22<00:18,  3.76s/it]
iter 5, loss: 7344542.8870:  60%|██████    | 6/10 [00:22<00:15,  3.86s/it]
iter 6, loss: 7228331.4661:  60%|██████    | 6/10 [00:26<00:15,  3.86s/it]
iter 6, loss: 7228331.4661:  70%|███████   | 7/10 [00:26<00:11,  3.92s/it]
iter 7, loss: 7158305.0771:  70%|███████   | 7/10 [00:30<00:11,  3.92s/it]
iter 7, loss: 7158305.0771:  80%|████████  | 8/10 [00:30<00:07,  3.87s/it]
iter 8, loss: 7100391.7960:  80%|████████  | 8/10 [00:34<00:07,  3.87s/it]
iter 8, loss: 7100391.7960:  90%|█████████ | 9/10 [00:34<00:03,  3.81s/it]
iter 9, loss: 7052070.6312:  90%|█████████ | 9/10 [00:38<00:03,  3.81s/it]
iter 9, loss: 7052070.6312: 100%|██████████| 10/10 [00:38<00:00,  3.78s/it]
iter 9, loss: 7052070.6312: 100%|██████████| 10/10 [00:38<00:00,  3.81s/it]
Your image compression within the expected range:  True
```

In [13]:

```python
##############################
### DO NOT CHANGE THIS CELL ###
##############################

img1_dir = "./data/images/image_test_1.jpg"
img2_dir = "./data/images/image_test_2.jpg"


def perform_compression(image, min_clusters=10, max_clusters=20):
    for K in range(min_clusters, max_clusters + 1, 5):
        gmm_image_k = cluster_pixels_gmm(image, K, max_iters=10, full_matrix=True)
        plot_images([image, gmm_image_k], ["original", "gmm=" + str(K)])


image1 = imageio.imread(img1_dir)
perform_compression(image1, 10, 15)

image2 = imageio.imread(img2_dir)
perform_compression(image2, 10, 15)
```

```
/tmp/ipykernel_450976/1626795805.py:15: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the current behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image1 = imageio.imread(img1_dir)
  0%|          | 0/10 [00:00<?, ?it/s]
iter 0, loss: 4246975.5221:   0%|          | 0/10 [00:00<?, ?it/s]
iter 0, loss: 4246975.5221:  10%|▉         | 1/10 [00:00<00:08,  1.10it/s]
iter 1, loss: 4246907.0658:  10%|▉         | 1/10 [00:01<00:08,  1.10it/s]
iter 1, loss: 4246907.0658:  20%|█▉        | 2/10 [00:01<00:06,  1.18it/s]
iter 2, loss: 4246765.5457:  20%|█▉        | 2/10 [00:02<00:06,  1.18it/s]
iter 2, loss: 4246765.5457:  30%|██▉       | 3/10 [00:02<00:05,  1.24it/s]
iter 3, loss: 4246339.4788:  30%|██▉       | 3/10 [00:03<00:05,  1.24it/s]
iter 3, loss: 4246339.4788:  40%|███▉      | 4/10 [00:03<00:04,  1.27it/s]
iter 4, loss: 4245040.0694:  40%|███▉      | 4/10 [00:03<00:04,  1.27it/s]
iter 4, loss: 4245040.0694:  50%|████▉     | 5/10 [00:03<00:03,  1.29it/s]
iter 5, loss: 4241335.5441:  50%|████▉     | 5/10 [00:04<00:03,  1.29it/s]
iter 5, loss: 4241335.5441:  60%|█████▉    | 6/10 [00:04<00:03,  1.30it/s]
iter 6, loss: 4232434.5264:  60%|█████▉    | 6/10 [00:05<00:03,  1.30it/s]
```

```
iter 6, loss: 4232434.5264:  70%|███████   | 7/10 [00:05<00:02,  1.30it/s]
iter 7, loss: 4216176.4300:  70%|███████   | 7/10 [00:06<00:02,  1.30it/s]
iter 7, loss: 4216176.4300:  80%|████████  | 8/10 [00:06<00:01,  1.29it/s]
iter 8, loss: 4194238.7110:  80%|████████  | 8/10 [00:07<00:01,  1.29it/s]
iter 8, loss: 4194238.7110:  90%|█████████ | 9/10 [00:07<00:00,  1.30it/s]
iter 9, loss: 4170928.6893:  90%|█████████ | 9/10 [00:07<00:00,  1.30it/s]
iter 9, loss: 4170928.6893: 100%|██████████| 10/10 [00:07<00:00,  1.30it/s]
iter 9, loss: 4170928.6893: 100%|██████████| 10/10 [00:07<00:00,  1.28it/s]
```

original

gmm=10

```
  0%|          | 0/10 [00:00<?, ?it/s]
iter 0, loss: 4245793.5084:   0%|          | 0/10 [00:01<?, ?it/s]
iter 0, loss: 4245793.5084:  10%|▉         | 1/10 [00:01<00:14,  1.59s/it]
iter 1, loss: 4243296.2543:  10%|▉         | 1/10 [00:03<00:14,  1.59s/it]
iter 1, loss: 4243296.2543:  20%|█▉        | 2/10 [00:03<00:12,  1.52s/it]
iter 2, loss: 4237619.3819:  20%|█▉        | 2/10 [00:04<00:12,  1.52s/it]
iter 2, loss: 4237619.3819:  30%|██▉       | 3/10 [00:04<00:09,  1.41s/it]
iter 3, loss: 4226087.7724:  30%|██▉       | 3/10 [00:05<00:09,  1.41s/it]
iter 3, loss: 4226087.7724:  40%|███▉      | 4/10 [00:05<00:08,  1.34s/it]
iter 4, loss: 4208180.3429:  40%|███▉      | 4/10 [00:06<00:08,  1.34s/it]
iter 4, loss: 4208180.3429:  50%|████▉     | 5/10 [00:06<00:06,  1.31s/it]
iter 5, loss: 4186065.4621:  50%|████▉     | 5/10 [00:08<00:06,  1.31s/it]
iter 5, loss: 4186065.4621:  60%|█████▉    | 6/10 [00:08<00:05,  1.27s/it]
iter 6, loss: 4162329.8751:  60%|█████▉    | 6/10 [00:09<00:05,  1.27s/it]
iter 6, loss: 4162329.8751:  70%|██████▉   | 7/10 [00:09<00:03,  1.25s/it]
iter 7, loss: 4135415.3908:  70%|██████▉   | 7/10 [00:10<00:03,  1.25s/it]
iter 7, loss: 4135415.3908:  80%|███████▉  | 8/10 [00:10<00:02,  1.27s/it]
iter 8, loss: 4089370.1735:  80%|███████▉  | 8/10 [00:11<00:02,  1.27s/it]
iter 8, loss: 4089370.1735:  90%|████████▉ | 9/10 [00:11<00:01,  1.24s/it]
iter 9, loss: 4022958.2588:  90%|████████▉ | 9/10 [00:12<00:01,  1.24s/it]
```

```
iter 9, loss: 4022958.2588: 100%|██████████| 10/10 [00:12<00:00,  1.23s/it]
iter 9, loss: 4022958.2588: 100%|██████████| 10/10 [00:12<00:00,  1.29s/it]
```


```
/tmp/ipykernel_450976/1626795805.py:18: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To
keep the current behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image2 = imageio.imread(img2_dir)
  0%|          | 0/10 [00:00<?, ?it/s]
iter 0, loss: 17622615.1731:   0%|          | 0/10 [00:09<?, ?it/s]
iter 0, loss: 17622615.1731:  10%|▉         | 1/10 [00:09<01:27,  9.72s/it]
iter 1, loss: 15020855.7030:  10%|▉         | 1/10 [00:19<01:27,  9.72s/it]
iter 1, loss: 15020855.7030:  20%|█▊        | 2/10 [00:19<01:19,  9.88s/it]
iter 2, loss: 14508266.0502:  20%|█▊        | 2/10 [00:29<01:19,  9.88s/it]
iter 2, loss: 14508266.0502:  30%|██▋       | 3/10 [00:29<01:09,  9.89s/it]
iter 3, loss: 14274290.1530:  30%|██▋       | 3/10 [00:38<01:09,  9.89s/it]
iter 3, loss: 14274290.1530:  40%|███▌      | 4/10 [00:38<00:57,  9.66s/it]
iter 4, loss: 14184639.1932:  40%|███▌      | 4/10 [00:48<00:57,  9.66s/it]
iter 4, loss: 14184639.1932:  50%|████▌     | 5/10 [00:48<00:48,  9.69s/it]
iter 5, loss: 14110165.6856:  50%|████▌     | 5/10 [00:58<00:48,  9.69s/it]
iter 5, loss: 14110165.6856:  60%|█████▍    | 6/10 [00:58<00:38,  9.68s/it]
iter 6, loss: 14038772.9515:  60%|█████▍    | 6/10 [01:08<00:38,  9.68s/it]
iter 6, loss: 14038772.9515:  70%|██████▎   | 7/10 [01:08<00:29,  9.70s/it]
iter 7, loss: 13990458.4019:  70%|██████▎   | 7/10 [01:17<00:29,  9.70s/it]
iter 7, loss: 13990458.4019:  80%|███████▏  | 8/10 [01:17<00:19,  9.59s/it]
iter 8, loss: 13962194.2245:  80%|███████▏  | 8/10 [01:27<00:19,  9.59s/it]
iter 8, loss: 13962194.2245:  90%|████████  | 9/10 [01:27<00:09,  9.72s/it]
iter 9, loss: 13945100.0005:  90%|████████  | 9/10 [01:37<00:09,  9.72s/it]
iter 9, loss: 13945100.0005: 100%|██████████| 10/10 [01:37<00:00,  9.79s/it]
iter 9, loss: 13945100.0005: 100%|██████████| 10/10 [01:37<00:00,  9.74s/it]
```


```
  0%|          | 0/10 [00:00<?, ?it/s]
iter 0, loss: 17402519.3351:   0%|          | 0/10 [00:13<?, ?it/s]
iter 0, loss: 17402519.3351:  10%|▉         | 1/10 [00:13<01:57, 13.04s/it]
iter 1, loss: 14964608.7058:  10%|▉         | 1/10 [00:25<01:57, 13.04s/it]
iter 1, loss: 14964608.7058:  20%|█▉        | 2/10 [00:25<01:43, 12.90s/it]
iter 2, loss: 14518831.2542:  20%|█▉        | 2/10 [00:39<01:43, 12.90s/it]
iter 2, loss: 14518831.2542:  30%|██▊       | 3/10 [00:39<01:32, 13.28s/it]
iter 3, loss: 14284578.5614:  30%|██▊       | 3/10 [00:52<01:32, 13.28s/it]
iter 3, loss: 14284578.5614:  40%|███▊      | 4/10 [00:52<01:18, 13.09s/it]
iter 4, loss: 14172809.5719:  40%|███▊      | 4/10 [01:05<01:18, 13.09s/it]
iter 4, loss: 14172809.5719:  50%|████▊     | 5/10 [01:05<01:04, 12.99s/it]
iter 5, loss: 14095594.2905:  50%|████▊     | 5/10 [01:17<01:04, 12.99s/it]
iter 5, loss: 14095594.2905:  60%|█████▊    | 6/10 [01:17<00:51, 12.80s/it]
iter 6, loss: 14015931.1862:  60%|█████▊    | 6/10 [01:30<00:51, 12.80s/it]
iter 6, loss: 14015931.1862:  70%|██████▊   | 7/10 [01:30<00:38, 12.78s/it]
iter 7, loss: 13958758.3143:  70%|██████▊   | 7/10 [01:42<00:38, 12.78s/it]
iter 7, loss: 13958758.3143:  80%|███████▋  | 8/10 [01:42<00:25, 12.56s/it]
iter 8, loss: 13911618.8562:  80%|███████▋  | 8/10 [01:54<00:25, 12.56s/it]
iter 8, loss: 13911618.8562:  90%|████████▋ | 9/10 [01:54<00:12, 12.43s/it]
iter 9, loss: 13871185.4881:  90%|████████▋ | 9/10 [02:07<00:12, 12.43s/it]
iter 9, loss: 13871185.4881: 100%|██████████| 10/10 [02:07<00:00, 12.47s/it]
iter 9, loss: 13871185.4881: 100%|██████████| 10/10 [02:07<00:00, 12.72s/it]
```

3.3.2 Written Question [2pts]:

After completing the GMM-based image compression, consider how the clustering process would have been different if you had used K-means instead. How do the mixture weights (π) and pixel-cluster responsibilities (τ) in GMM influence the reconstructed image? In what situations might GMM provide advantages over K-means, and when might K-means be sufficient?

3.4 Compare full covariance matrix with diagonal covariance matrix [1% Bonus for All]

Compare the results of clustering an image with full covariance matrix and diagonal covariance matrix. Can you explain why the images are different with same clusters? Note: You will have to implement both multinormalPDF and normalPDF, and add a few arguments in the original _ll_joint(), _M_step(), _E_step() function. **You will earn full credit only if you implement all functions AND provide an explanation.**

In [14]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################


def compare_matrix(image, K):
    """
    Args:
        image: input image of shape(H, W, 3)
        K: number of components

    Return:
```

```
        plot: comparison between full covariance matrix and diagonal covariance matrix.
    """
    # full covariance matrix
    gmm_image_full = cluster_pixels_gmm(image, K, 10, full_matrix=True)
    # diagonal covariance matrix
    gmm_image_diag = cluster_pixels_gmm(image, K, 10, full_matrix=False)

    plot_images(
        [gmm_image_full, gmm_image_diag],
        ["full covariance matrix", "diagonal covariance matrix"],
    )
```

In [15]:
```
compare_matrix(image2, 20)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
iter 0, loss: 17485038.3306:   0%|          | 0/10 [00:19<?, ?it/s]
iter 0, loss: 17485038.3306:  10%|▉         | 1/10 [00:19<02:54, 19.44s/it]
iter 1, loss: 15001561.5684:  10%|▉         | 1/10 [00:39<02:54, 19.44s/it]
iter 1, loss: 15001561.5684:  20%|█▊        | 2/10 [00:39<02:39, 19.95s/it]
iter 2, loss: 14384132.8647:  20%|█▊        | 2/10 [01:00<02:39, 19.95s/it]
iter 2, loss: 14384132.8647:  30%|██▋       | 3/10 [01:00<02:20, 20.11s/it]
iter 3, loss: 14191722.0282:  30%|██▋       | 3/10 [01:20<02:20, 20.11s/it]
iter 3, loss: 14191722.0282:  40%|███▌      | 4/10 [01:20<02:02, 20.41s/it]
iter 4, loss: 14122921.7728:  40%|███▌      | 4/10 [01:41<02:02, 20.41s/it]
iter 4, loss: 14122921.7728:  50%|████▌     | 5/10 [01:41<01:42, 20.41s/it]
iter 5, loss: 14037326.7565:  50%|████▌     | 5/10 [02:02<01:42, 20.41s/it]
iter 5, loss: 14037326.7565:  60%|█████▍    | 6/10 [02:02<01:22, 20.54s/it]
iter 6, loss: 13925813.1390:  60%|█████▍    | 6/10 [02:21<01:22, 20.54s/it]
iter 6, loss: 13925813.1390:  70%|██████▎   | 7/10 [02:21<01:00, 20.30s/it]
iter 7, loss: 13847265.6077:  70%|██████▎   | 7/10 [02:42<01:00, 20.30s/it]
iter 7, loss: 13847265.6077:  80%|███████▏  | 8/10 [02:42<00:40, 20.32s/it]
iter 8, loss: 13807926.0906:  80%|███████▏  | 8/10 [03:00<00:40, 20.32s/it]
iter 8, loss: 13807926.0906:  90%|████████  | 9/10 [03:00<00:19, 19.72s/it]
iter 9, loss: 13786683.3982:  90%|████████  | 9/10 [03:21<00:19, 19.72s/it]
iter 9, loss: 13786683.3982: 100%|█████████| 10/10 [03:21<00:00, 20.12s/it]
iter 9, loss: 13786683.3982: 100%|█████████| 10/10 [03:21<00:00, 20.17s/it]
  0%|          | 0/10 [00:00<?, ?it/s]
iter 0, loss: 17770910.3781:   0%|          | 0/10 [00:11<?, ?it/s]
iter 0, loss: 17770910.3781:  10%|▉         | 1/10 [00:11<01:46, 11.85s/it]
iter 1, loss: 16503805.0689:  10%|▉         | 1/10 [00:23<01:46, 11.85s/it]
iter 1, loss: 16503805.0689:  20%|█▊        | 2/10 [00:23<01:34, 11.84s/it]
iter 2, loss: 15925014.9946:  20%|█▊        | 2/10 [00:35<01:34, 11.84s/it]
iter 2, loss: 15925014.9946:  30%|██▋       | 3/10 [00:35<01:21, 11.71s/it]
iter 3, loss: 15480525.3266:  30%|██▋       | 3/10 [00:46<01:21, 11.71s/it]
iter 3, loss: 15480525.3266:  40%|███▌      | 4/10 [00:46<01:10, 11.73s/it]
iter 4, loss: 15125192.9425:  40%|███▌      | 4/10 [00:58<01:10, 11.73s/it]
iter 4, loss: 15125192.9425:  50%|████▌     | 5/10 [00:58<00:57, 11.55s/it]
iter 5, loss: 14853710.9359:  50%|████▌     | 5/10 [01:09<00:57, 11.55s/it]
iter 5, loss: 14853710.9359:  60%|█████▍    | 6/10 [01:09<00:46, 11.57s/it]
```

```
iter 6, loss: 14696689.7979:  60%|██████    | 6/10 [01:21<00:46, 11.57s/it]
iter 6, loss: 14696689.7979:  70%|███████   | 7/10 [01:21<00:34, 11.58s/it]
iter 7, loss: 14597422.8008:  70%|███████   | 7/10 [01:32<00:34, 11.58s/it]
iter 7, loss: 14597422.8008:  80%|████████  | 8/10 [01:32<00:23, 11.52s/it]
iter 8, loss: 14530197.1573:  80%|████████  | 8/10 [01:44<00:23, 11.52s/it]
iter 8, loss: 14530197.1573:  90%|█████████ | 9/10 [01:44<00:11, 11.66s/it]
iter 9, loss: 14484523.6973:  90%|█████████ | 9/10 [01:56<00:11, 11.66s/it]
iter 9, loss: 14484523.6973: 100%|██████████| 10/10 [01:56<00:00, 11.64s/it]
iter 9, loss: 14484523.6973: 100%|██████████| 10/10 [01:56<00:00, 11.64s/it]
```

full covariance matrix

diagonal covariance matrix

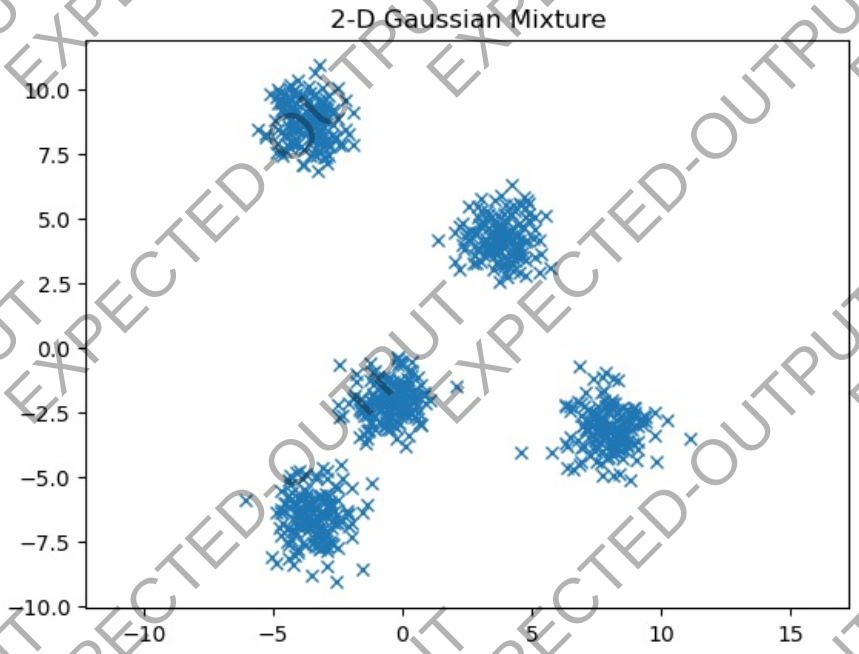
3.5 Generate samples from a Gaussian Mixture [5pts]

In this question, you will be fitting your GMM implementation on a 2D Gaussian Mixture to estimate the parameters of the distributions that make up the mixture, and then using these estimated parameters to generate samples.

Implement the `density` and `rejection_sample` functions in gmm.py. Then, fit your GMM implementation below by initializing and calling the class.

In [16]:
```
###############################
### DO NOT CHANGE THIS CELL ###
###############################

data = np.load("./data/gaussian_clusters.npy")
print(data.shape)

plt.plot(data[:, 0], data[:, 1], "x")
plt.axis("equal")
plt.title("2-D Gaussian Mixture")
plt.show()
```

```
(800, 2)
```



In [17]:
```
# Fit your GMM implementation
```

In [18]:
```
```

```
  0%|          | 0/30 [00:00<?, ?it/s]
iter 0, loss: 3918.4241:   0%|          | 0/30 [00:00<?, ?it/s]
iter 1, loss: 3608.2121:   0%|          | 0/30 [00:00<?, ?it/s]
iter 2, loss: 3472.6925:   0%|          | 0/30 [00:00<?, ?it/s]
iter 3, loss: 3318.3973:   0%|          | 0/30 [00:00<?, ?it/s]
iter 4, loss: 3196.2704:   0%|          | 0/30 [00:00<?, ?it/s]
iter 5, loss: 3192.9793:   0%|          | 0/30 [00:00<?, ?it/s]
iter 6, loss: 3192.9793:   0%|          | 0/30 [00:00<?, ?it/s]
iter 7, loss: 3192.9793:   0%|          | 0/30 [00:00<?, ?it/s]
iter 8, loss: 3192.9793:   0%|          | 0/30 [00:00<?, ?it/s]
iter 9, loss: 3192.9793:   0%|          | 0/30 [00:00<?, ?it/s]
iter 10, loss: 3192.9793:   0%|          | 0/30 [00:00<?, ?it/s]
iter 11, loss: 3192.9793:   0%|          | 0/30 [00:00<?, ?it/s]
iter 11, loss: 3192.9793:  40%|████      | 12/30 [00:00<00:00, 219.93it/s]
```

Now, you need to estimate the parameters of the Gaussian Mixture, and then use these estimated parameters to generate 1000 samples from the Gaussian Mixture. Plot the sampled datapoints. **You should notice that it resembles the original Gaussian Mixture.**

Steps

- First, to estimate the parameters of the Gaussian Mixture, you'll need to fit your GMM implementation to the dataset. You need to specify K=5 to represent 5

gaussians in our model, and run the EM algorithm. You'll have to choose the value for max_iters. If at the end of this section, your plot of the sampled datapoints doesn't look like the original distribution, you may need to increase max_iters to fit the GMM model better, and obtain better estimates of the parameters.

- Once you have the estimated parameters, we'll need to sample 1000 datapoints from the Gaussian Mixture. You will be using a technique called Rejection Sampling discussed below. Here are some external sources that may help: https://cosmiccoding.com.au/tutorials/rejection_sampling, https://towardsdatascience.com/rejection-sampling-with-python-d7a30cfc327b
  - We will be taking the approach from the first link, but extending it into the 2D space.
- The formula for the density function is $f(x^{\{i\}}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x^{\{i\}} | \mu_k, \Sigma_k)$

Generating vs Sampling To generate points directly from a given distribution is done via Inverse transform sampling. In inverse transform sampling, we require taking the inverse of cummulative distribution function for our gaussian mixture model. This operation in general can be expensive unless there is some known formula for inverting the CDF. It is also not always possible to take the inverse of the CDF of a gaussian mixture model. For these reasons we will implement a sampling method instead. This sampling method will give us points matching the gmm without the computation and mathematical concerns of generation.

Rejection Sampling

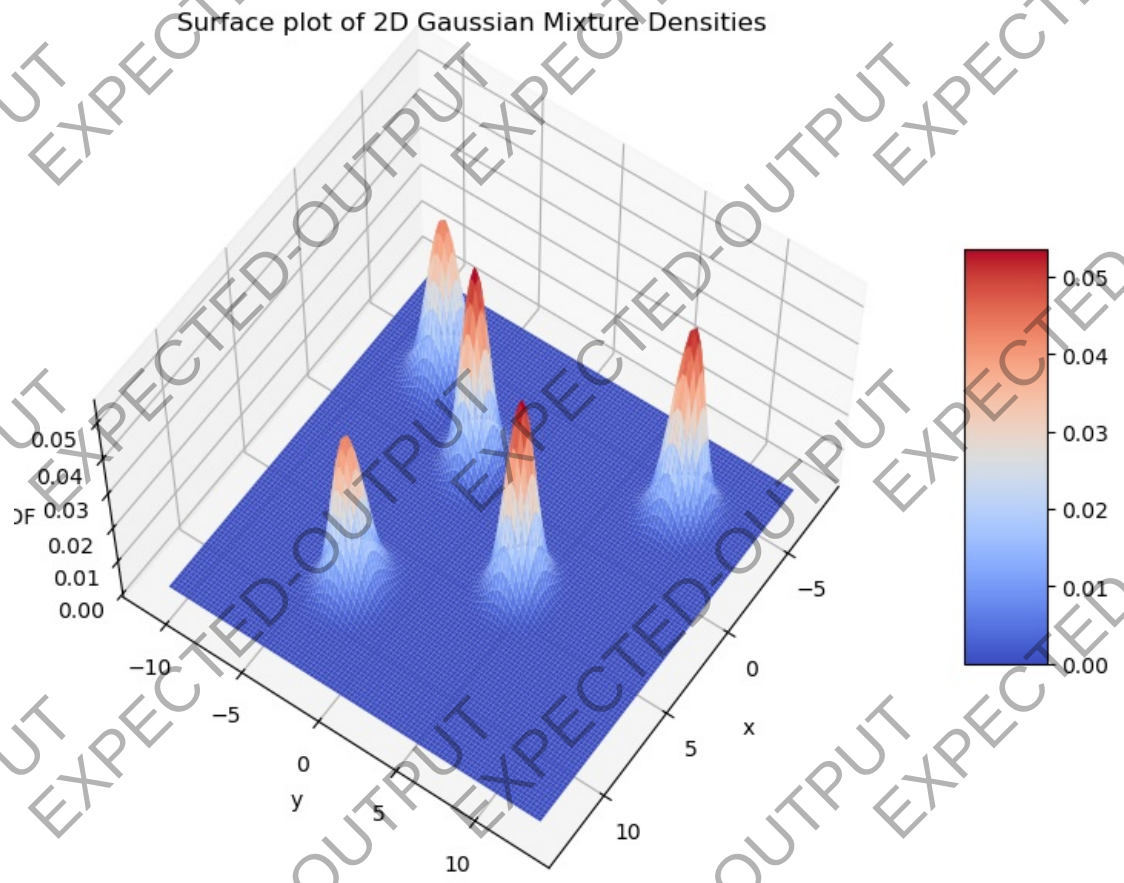
Conventionally we think of Gaussian Mixture Models as a form of soft clustering, but you can also think of them as an algorithm for estimating density of data points with gaussians. Thus we can take an arbitary data point and using the gaussian mixture model as an estimation for the density at a given location. From here we want the points that we sample to be proportional to the density at a given location.

We go about this by, choosing an arbitary point (x,y). Then we use the density formula function $f(x^{\{i\}}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x^{\{i\}} | \mu_k, \Sigma_k)$ to find out what the density of points is at (x,y). Now that we have the density, we can draw a random number between 0 and the maximum density to determine if we will keep or discard (x,y). If the random number drawn is less than the density, then (x,y) is our sample, otherwise we discard (x,y) and repeat. This method ensure that the samples we generate are proportional to the density predicted by our GMM at any given area.

In [19]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# Extract x and y
x = data[:, 0]
y = data[:, 1]

# Define the borders of the grid
deltaX = (max(x) - min(x)) / 10
deltaY = (max(y) - min(y)) / 10
xmin = min(x) - deltaX
xmax = max(x) + deltaX
ymin = min(y) - deltaY
ymax = max(y) + deltaY


# Create meshgrid
xx, yy = np.mgrid[xmin:xmax:100j, ymin:ymax:100j]
# coordinates of the points that make the grid
positions = np.vstack([xx.ravel(), yy.ravel()]).T
```

In [20]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################


# get the density at each coordinate on the grid
densities = np.reshape(density(positions, pi, mu, sigma, gmm), xx.shape)
```

```
fig = plt.figure(figsize=(13, 7), dpi=100)
ax = plt.axes(projection="3d")
surf = ax.plot_surface(
    xx, yy, densities, rstride=1, cstride=1, cmap="coolwarm", edgecolor="none"
)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("PDF")
ax.set_title("Surface plot of 2D Gaussian Mixture Densities")
fig.colorbar(surf, shrink=0.5, aspect=5)  # add color bar indicating the PDF
ax.view_init(60, 35)
plt.show()
```



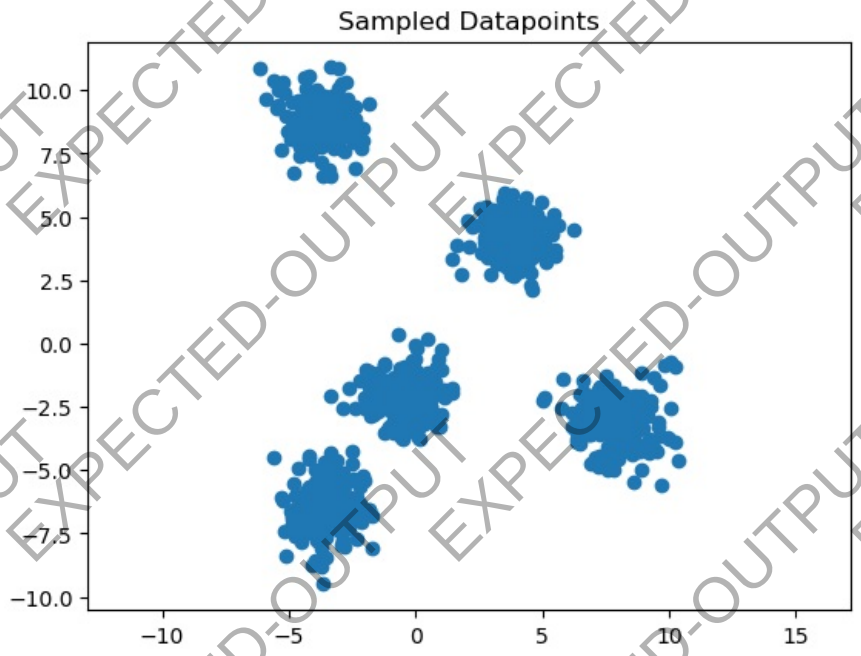
In [21]:

```
################################
### DO NOT CHANGE THIS CELL ###
################################


# Sample datapoints using Rejection Sampling
generated_datapoints = np.zeros((1000, 2))
i = 0
while i < 1000:
    generated_datapoints[i, 0], generated_datapoints[i, 1] = rejection_sample(
        xmin, xmax, ymin, ymax, pi, mu, sigma, gmm, dmax=1
    )
```

```
    if i % 100 == 0:
        print(i)
    i += 1
```

```
0
100
200
300
400
500
600
700
800
900
```

In [22]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

plt.scatter(generated_datapoints[:, 0], generated_datapoints[:, 1])
plt.axis("equal")
plt.title("Sampled Datapoints")
plt.show()
```



3.6 Random vs. KMeans Initialization [1% Bonus for All]

Initializing our GMM parameters randomly could yield a long training time, so we could try to use a heuristic for initialization. For example, we could use a trained KMeans model on our dataset, to initialize the centers (mu) of our GMM model.

Implement the `create_mu_kmeans` function in gmm.py. Then, call it in your `_init_components` function using the `kmeans_init` boolean to denote if you're using the kmeans initialization or random initialization. Finally, the below cells will run a KMeans initialization and a random initialization, and print out the number of iterations

GMM takes to converge on both.

In [23]:
```
# Experiment with the below parameters!
# num_trials may be particularly influential to your results
#   since the randomness is unseeded, so the intialization techniques may perform uncharacteristically poor or well on this few trials
# however, the other parameters may also be poorly conditioned

full_matrix = False
K = 5
max_iters = 100
rel_tol = 1e-5
num_trials = 40
```

In [24]:
```
###############################
### DO NOT CHANGE THIS CELL ###
###############################

df = sns.load_dataset("iris")
data = np.array(df.drop("species", axis=1))

random_iters = []
kmeans_iters = []

for _ in range(num_trials):
    student_gmm_random = GMM(data, K, max_iters=max_iters, seed=None)
    student_gmm_random(
        full_matrix=full_matrix, kmeans_init=False, rel_tol=rel_tol, disable_tqdm=True
    )
    random_iters.append(student_gmm_random.num_iters)

    student_gmm_kmeans = GMM(data, K, max_iters=max_iters, seed=None)
    student_gmm_kmeans(
        full_matrix=full_matrix, kmeans_init=True, rel_tol=rel_tol, disable_tqdm=True
    )
    kmeans_iters.append(student_gmm_kmeans.num_iters)

print("Random num iterations: ", np.mean(random_iters))
print("KMeans num iterations: ", np.mean(kmeans_iters))
```

```
Random num iterations:  28.725
KMeans num iterations:  23.725
```

Investigate the number of iterations of random initialization against KMeans initialization above. Which initialization allows GMM to converge faster during training? Why do you think KMeans initialization is slower or faster (~1-2 sentences)?

4. (Bonus for All) Cleaning Messy data and semi-supervised learning [7% Bonus for All]

Learning to work with messy data is a hallmark of a well-rounded data scientist. In most real-world settings the data given will usually have some issue, so it is important to learn skills to work around such impasses. This part of the assignment looks to expose you to clever ways to fix data using concepts that you have already learned in the prior questions.

Problem Scenario

Congratulations! you recently graduated with your shiny GT degree. You've decided to pursue your childhood dream to study astrophysics. Stationed at a cutting edge Gamma Ray Telescope facility as a Data Scientist, delving into the high-energy physics of the universe, your mission is to probe the enigmatic Sagittarius A, the*

supermassive black hole in the center of our galaxy, using a telescope that views the cosmos through gamma rays emitted by the most cataclysmic events in space like neutron star mergers, pulsars, and the voracious accretion disks of black holes. SUPER EXCITING!

The cutting edge telescope you are working with detects these really high energy gamma ray emissions from really small windows in the sky. You find that the telescope can be configured with a few parameters to detect these particles. These parameters, 10 in number, range from the telescope's orientation and timing precision to the sensitivity settings that find the faintest gamma signals against the cosmic background.

However, your cosmic quest faces an unexpected challenge. A data corruption incident has left a 15% void across your dataset, affecting both the intricate telescope parameters and the critical gamma ray detection records. This isn't just a minor hiccup; it's a significant obstacle in your path to unraveling the mysteries of our galaxy's heart.

But there's a silver lining. You remember that the machine learning techniques learnt in CS4641/7641 are the key to navigating this data loss issue. This challenge transforms into an opportunity to showcase the resilience and ingenuity of data science in the face of adversity. How will you leverage your skills to reconstruct the missing pieces and ensure that your exploration of Sagittarius A yields groundbreaking insights into the universe's most profound secrets?*

Task: Clean the data and implement a semi-supervised learning framework to classify the detection of gamma rays for your experiments. The data has 10 feature columns containing the telescope's parameters and one column containing a binary label containing either (0 or 1) representing the absence or a presence of a signal.

You are given two files for this task:

- data.csv: the entire dataset with complete and incomplete data
- validation.csv: a smaller, fully complete dataset made after the intern deleted the datapoints

4.1 Data Cleaning [2.8%]

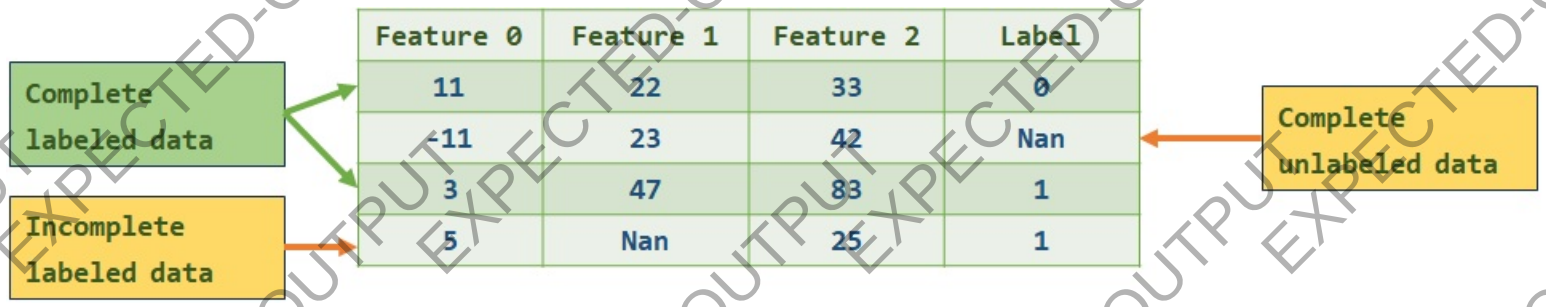
4.1.a Data Separating [0.7%]

The first step is to break up the whole dataset into clear parts. All the data is randomly shuffled in one csv file. In order to move forward, the data needs to be split into three separate arrays:

- labeled_complete: containing the complete characterization data and corresponding labels
- labeled_incomplete: containing partial characterization data (i.e., one of the features is NaN) and corresponding labels
- unlabeled_complete: containing complete characterization data but no corresponding labels (i.e., the label is NaN)

In **semisupervised.py**, implement the following methods:

- complete_
- incomplete_
- unlabeled_



In [25]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

localtests.SemisupervisedTests().test_data_separating_methods()
```

```
UnitTest passed successfully!
```

4.1.b KNN [1.4%]

The second step in this task is to clean the Labeled_incomplete dataset by filling in the missing values with probable ones derived from complete data. A useful approach to this type of problem is using a k-nearest neighbors (k-NN) algorithm. For this application, the method consists of replacing the missing value of a given point with the mean of the closest k-neighbors to that point. Given that you are focusing on neighbouring points, the margin of error from actual missing values should be limited.

In the **CleanData** class in **semisupervised.py**, implement the following methods:

- pairwise_dist
- __call__

The unit test is a good expectation of what the process should look like on a toy dataset. If your output matches the answer, you are on the right track. Run the following cell to check.

NOTE: Your rows of data should match with the expected output, although the order of the rows does not necessarily matter.

In [26]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

localtests.SemisupervisedTests().test_cleandata()
```

```
UnitTest passed successfully!
```

4.1.c Median of Features [0.7%]

Another method of filling the missing values is by using the median of individual features. Our goal with replacing NaN values is to insert values in their place while also minimally disturbing the overall distribution of each feature. Using the median of features helps avoid drastically changing the distribution of our data. This is also why while we could technically replace NaN values with 0, it is generally not advised to do so.

Implement the median_clean_data method in accordance with this rule. NOTE: There should be no NaN values in the n*d array that you return from median_clean_data.

In **semisupervised.py**, implement the following method:

- median_clean_data

In [27]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

localtests.SemisupervisedTests().test_median_clean_data()
```

```
UnitTest passed successfully!
```

4.2 Semi-supervised Learning [3.5%]

Semi-supervised learning is a machine learning approach that bridges the gap between supervised and unsupervised learning by training models on a dataset containing

both labeled and unlabeled data. Typically, only a small portion of the dataset is labeled often because labeling is expensive, time consuming, or requires expert knowledge while the majority remains unlabeled. Despite the scarcity of labeled data, semi-supervised learning leverages it to provide initial guidance, then uses patterns and structures within the unlabeled data to further refine and enhance the model's performance. This method is particularly effective in real-world scenarios where collecting large amounts of labeled data is impractical but raw data is abundant, such as in medical diagnosis, speech recognition, and natural language processing. A notable application is in text classification tasks, where a few annotated examples can be used to improve classification accuracy across large corpora of unlabelled text, as demonstrated in the paper discussed in the next section.

4.2.a Getting acquainted with semi-supervised learning approaches. [1.2%]

Take a look at the algorithm presented in Table 1 of the paper "Text Classification from Labeled and Unlabeled Documents using EM" by Nigam et al. (2000). While you are recommended to read the whole paper this assignment focuses on items 5.1, 5.2, and 6.1. Write a brief summary of three interesting highlights of the paper (50-words maximum).

4.2.b Implementing the EM algorithm. [2.3%]

Implement the EM algorithm proposed by Nigam et al. (2000) on Table 1, using a Gaussian Naive Bayes (GNB) classifier instead of a Naive Bayes (NB) classifier. What's the difference between the way of initialization in the paper and the way introduced in class?

(Hint: Using a GNB in place of an NB will enable you to reuse most of the implementation you developed for GMM in this assignment. Instead of building an initial naive Bayes classifier like it says in the second bullet point of Table 1, think about how we can implement this step with Gaussians. In fact, you can successfully solve the problem by simply modifying the __call__ and _init_components methods.)

In the **SemiSupervised** class in **semisupervised.py**, implement the following methods:

- _init_components
- __call__

4.3 Demonstrating the performance of the algorithm [0pts]

Let's compare the classification error based on the Gaussian Naive Bayes (GNB) classifier you implemented following the Nigam et al. (2000) approach to the performance of a GNB classifier trained using only labeled data.

In [28]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
from semisupervised import ComparePerformance

ComparePerformance.accuracy_comparison()
```

```
All Data shape:                 (19020, 11)
Labeled Complete shape:         (16167, 11)
Labeled Incomplete shape:       (2621, 11)
Labeled shape:                  (18788, 11)
Unlabeled shape:                (232, 11)
Cleaned data shape:             (18788, 11)
Cleaned + Unlabeled data shape: (19020, 11)
  0%|          | 0/100 [00:00<?, ?it/s]
iter 0, loss: 631907.0396:   0%|          | 0/100 [00:00<?, ?it/s]
iter 1, loss: 631904.0766:   0%|          | 0/100 [00:00<?, ?it/s]
iter 2, loss: 631904.0152:   0%|          | 0/100 [00:00<?, ?it/s]
iter 3, loss: 631904.0139:   0%|          | 0/100 [00:00<?, ?it/s]
```

```
iter 4, loss: 631904.0139:   0%|          | 0/100 [00:00<?, ?it/s]
iter 5, loss: 631904.0139:   0%|          | 0/100 [00:00<?, ?it/s]
iter 5, loss: 631904.0139:   6%|▌         | 6/100 [00:00<00:01, 52.99it/s]
iter 6, loss: 631904.0139:   6%|▌         | 6/100 [00:00<00:01, 52.99it/s]
iter 7, loss: 631904.0139:   6%|▌         | 6/100 [00:00<00:01, 52.99it/s]
iter 7, loss: 631904.0139:   8%|▊         | 8/100 [00:00<00:01, 46.93it/s]
===COMPARISON===
Supervised with only complete data, GNB Accuracy: 72.275%
Supervised with KNN clean data, GNB Accuracy:     72.425%
Supervised with Median clean data, GNB Accuracy:   72.3%
SemiSupervised Accuracy:                          72.2%
```

4.4 Interpretation of Results. [0.7%]

What are the differences in using the kNN method and the median method to fill NaN values? Explain in terms of the results you get from each. What would be some advantages of using the median method to fill in NaN values over using the **mean** of features?

5: Hierarchical Clustering [9 pts Grad / 4.5% Bonus for Undergrad]

5.1 Hierarchical Clustering Implementation [9pts Grad/4.5% Bonus for Undergrad]

Hierarchical Clustering is a bottom-up agglomerative clustering algorithm which iteratively combines the closest pair of clusters. Each point starts off as its own cluster, and in each iteration you'll find the closest clusters and update the distances to the new cluster using single-link clustering, keeping track of the order in which the clusters are combined. In this section, you'll implement the `create_distance_matrix`, `iterate`, and `fit` methods in **hierarchical_clustering.py**.

The `HierarchicalClustering` class has several instance variables that you may need to create and update in each iteration:

1. `points`: N x D numpy array where N is the number of points, and D is the dimensionality of each point. This is your dataset.
2. `distance`: N x N symmetric numpy array which stores pairwise distances between clusters. The distance between a cluster and itself should be `np.inf` in order to help us calculate the closest pair later
3. `cluster_ids`: (N,) numpy array where index_array[i] gives the cluster id of the i-th column and i-th row of distances. Initially, each point with index `points[i, :]` is assigned cluster id i, and new points are assigned cluster ids starting from `N` and incrementing.
4. `clustering`: (N - 1, 4) numpy array that keeps track of which clusters were merged in each iteration. `clustering[iteration_number]` keeps track of the first cluster id, second cluster id, distance between first and second clusters, and the size of new cluster
5. `cluster_sizes` (2N - 1,) numpy array that stores the number of points in each cluster, indexed by id. Because there are `N` original clusters corresponding to each point, and each iteration merges two clusters, there will be `2N-1` total clusters created.

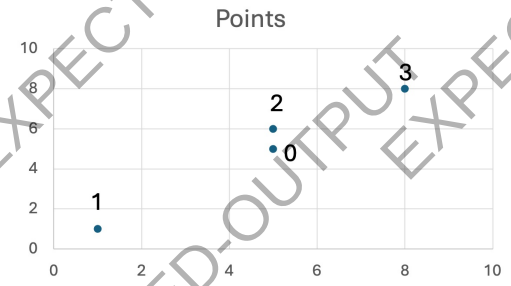
Theses are the following functions you'll have to implement in **hierarchical_clustering.py**:

1. `create_distances`: Creates the initial distance matrix and cluster ids
2. `iterate`: Merges the two closest clusters
3. `fit`: Calls `iterate` multiple times and returns the clusterings

An example of how the instance variables should be updated in an iteration is shown below:

Before:

current_iteration = `0`



Index	0	1	2	3
0	inf	5.6	1	4.2
1	5.6	inf	6.4	9.9
2	1	6.4	inf	3.6
3	4.2	9.9	3.6	inf

distances

cluster_ids = `[0, 1, 2, 3]`

cluster_sizes = `[1, 1, 1, 1, 0, 0, 0]`

clustering = `[[0, 0, 0, 0], …, [0, 0, 0, 0]]`

After calling `iterate`

current_iteration = `1`

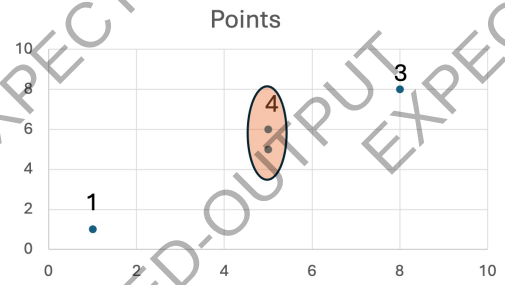
cluster_ids = `[4, 1, 3]`

The first row/col represents the cluster with id 4 (the new cluster), the second row/col represents the cluster with id 1, and the third row/col represents the cluster with id 3. Clusters with ids 0 and 2 are deleted from the distance matrix after being combined

cluster_sizes = `[1, 1, 1, 1, 2, 0, 0]`

clustering = `[[0, 2, 1, 2], …, [0, 0, 0, 0]]`

For the first iteration, we combined cluster ids 0 and 2, which had an inter-cluster distance of 1 and the new cluster contains 2 points



Index	0	1	2
0	inf	5.6	3.6
1	5.6	inf	9.9
2	3.6	9.9	inf

distances

In [29]:
```
from hierarchical_clustering import HierarchicalClustering
from scipy.cluster import hierarchy
```

In [30]:
```
localtests.HierarchicalClusteringTests().test_create_distance()
localtests.HierarchicalClusteringTests().test_iterate_1d()
localtests.HierarchicalClusteringTests().test_iterate_2d()
localtests.HierarchicalClusteringTests().test_fit()
```

```
UnitTest passed successfully!
UnitTest passed successfully!
UnitTest passed successfully!
UnitTest passed successfully!
```

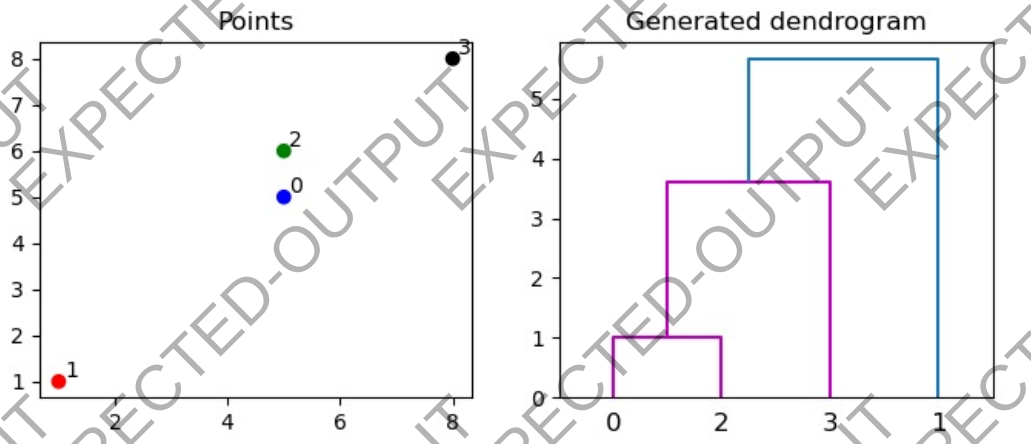
5.2 Hierarchical Clustering Visualization [0 pts]

In this section, you'll run Hierarchical Clustering on an example dataset and visualize it in a dendrogram using SciPy.

In [31]:
```
points = np.array([[5, 5], [1, 1], [5, 6], [8, 8]])
```

In [32]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################

hc = HierarchicalClustering(points)
clustering = hc.fit()

fig, axes = plt.subplots(1, 2, figsize=(8, 3))
```

```
axes[0].scatter(points[:, 0], points[:, 1], c=["blue", "red", "green", "black"])
for i in range(points.shape[0]):
    axes[0].annotate(i, points[i] + 0.1)
    axes[0].set_title("Points")

hierarchy.set_link_color_palette(["m", "c", "y", "k"])
dn1 = hierarchy.dendrogram(clustering, ax=axes[1], orientation="top")
axes[1].set_title("Generated dendrogram")
hierarchy.set_link_color_palette(None)
plt.show()
```



5.3 Hierarchical Clustering Large Dataset Visualization [0 pts]

Now you'll run Hierarchical Clustering on a larger dataset (Iris). Run the following code cell once in order to install the library that will allow you to visualize a radial dendrogram.

In [33]:
```
import matplotlib.pyplot as plt
import numpy as np
import radialtree as rt
import scipy.cluster.hierarchy as sch
from seaborn import load_dataset
```

In [34]:
```
#get a simple dataset
iris = load_dataset("iris")
species = iris.pop("species")

fig, axes = plt.subplots(2, 1, figsize=(20, 15))

# Compute and plot the dendrogram.
Y = sch.linkage(np.asarray(iris), method="average")
Z2 = sch.dendrogram(
    Y,
    # no_plot=True,
    ax=axes[0],
    color_threshold=1.0,
)

axes[1].set_aspect(1)
# plot a circular dendrogram
```

```
rt.radialTreee(Z2, ax=axes[1], sample_classes={"species": species})
```

```
0.15000000000000002
1.1500000000000001
[[0.     0.     0.     1.    ]
 [0.     0.7333 0.     1.    ]
 [0.8    0.8    0.8    1.    ]]
```

Out[34]: <Axes: >