Spring 2025 CS4641/CS7641 Homework 2

Instructor: Dr. Mahdi Roozbahani

Deadline: Friday, March 7th, 11:59 pm EST

- No unapproved extension of the deadline is allowed. Late submission will lead to 0 credit.
- Discussion is encouraged on Ed as part of the Q/A. 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.
- 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 ALL CASES TO OSI**, which may, unfortunately, lead to a very harsh outcome. **Consequences can be severe, e.g., 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.
- To switch between cell for code and for markdown, see the menu -> Cell -> Cell Type
- You can directly type Latex equations into markdown cells.
- If a question requires a picture, you could use this syntax `<img src="" style="width: 300px;"/>` to include them within your ipython notebook.
- Your write up must be submitted in PDF form. You may use either Latex, markdown, or any word processing software. We will **NOT** accept handwritten work. Make sure

that your work is formatted correctly, for example submit $\sum_{i=0} x_i$ instead of \text{sum_{i=0} x_i}

- When submitting the non-programming part of your assignment, you must correctly map pages of your PDF to each question/subquestion to reflect where they appear. **Improperly mapped questions may not be graded correctly and/or will result in point deductions for the error.**
- All assignments should be done individually, and 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.
- **For the "Assignment 2 - Non-programming" part, 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 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", otherwise there may be a deduction in points for extra long sheets.**

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 [40pts: 37pts + 3pts Grad / 1.5% Bonus for Undergrad]

Deliverables: kmeans.py

- **pairwise_dist** [5 pts] - *programming*
- **KMeans Implementation** [30pts: 27pts + 3pts Grad / 1.5% Bonus for Undergrad] - *programming*
  - init_centers [2pts]
  - kmpp_init [3pts Grad / 1.5% Bonus for Undergrad]
  - update_assignment [5pts]
  - update_centers [5pts]
  - get_loss function [5pts]
  - train [10pts]
- **Fowlkes-Mallow Measure** [5 pts] - *programming*

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

Deliverables: Markdown Cell Text

- **2.1 Performing EM Update** [15 pts] - *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 [60pts + 2% Bonus for All]

Deliverables: gmm.py and Markdown Cell Text

- 3.1 Helper Functions [15pts] - *programming & non-programming*
    - 3.1.1. softmax [5pts]
    - 3.1.2. logsumexp [3pts + 2pts] - *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 [10pts] - *programming*
- 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 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 [no pts]
- 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]

Q6: Evaluating Data Representation in K-Means Clustering [4pts]

Deliverables: Markdown Cell Text

Points Totals:

- Total Base: 128 pts for grads / 116 pts for undergrads
- Total Undergrad Bonus: 6%
- Total Bonus for All: 10%

Gradescope Submission Deliverables:

- For any Non-Programming portion: HW2.pdf (Jupyter notebook converted to 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".

In [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: 3.11.11 (main, Dec 11 2024, 10:25:04) [Clang 14.0.6 ]
matplotlib: 3.10.0
numpy: 2.2.2
```

1. KMeans Clustering [40pts total: 37pts + 3pts Grad / 1.5% Bonus for Undergrad]

KMeans is trying to solve the following optimization problem:

$$\arg\min_{S} \sum_{i=1}^{K} \sum_{x_j \in S_i} ||x_j - \mu_i||^2 \tag{1}$$

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.

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 $dist_{i,j} = ||X_i - Y_j||_2$.

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. 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 [2]:
```
localtests.KMeansTests().test_pairwise_dist()
localtests.KMeansTests().test_pairwise_speed()
```

```
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 + 3pts Bonus for Undergrad]

The Kmeans algorithm is sensitive to how the centers are initialized. The naive approach is to randomly initialize the centers. However, a bad initialization can increase the time required for convergence or may even converge to a non-optimal solution.

- **init_centers** [2pts]: Here you will initialize the centers randomly **(Required for all)**

Hint: Please initialize centers by randomly sampling points (without repetition) from the data passed to the KMeans() object upon instantiation in case the autograder fails.

- **kmpp_init** [3pts Bonus for Undergrad]: Here you will use the intuition that points further away from each other will probably be better initial centers by implementing a version of KMeans++ **(Bonus for Undergrad, required for Grads)**

Hint: We need to initialize the centers without repetition.

KMeans++

The algorithm for KMPP that you will implement 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.

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.

Updating Centers Assignments: [5pts]

Since cluster memberships may have changed, you will need to update the cluster centers. You will implement this in **update_centers**. See docstring for more details.

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

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.

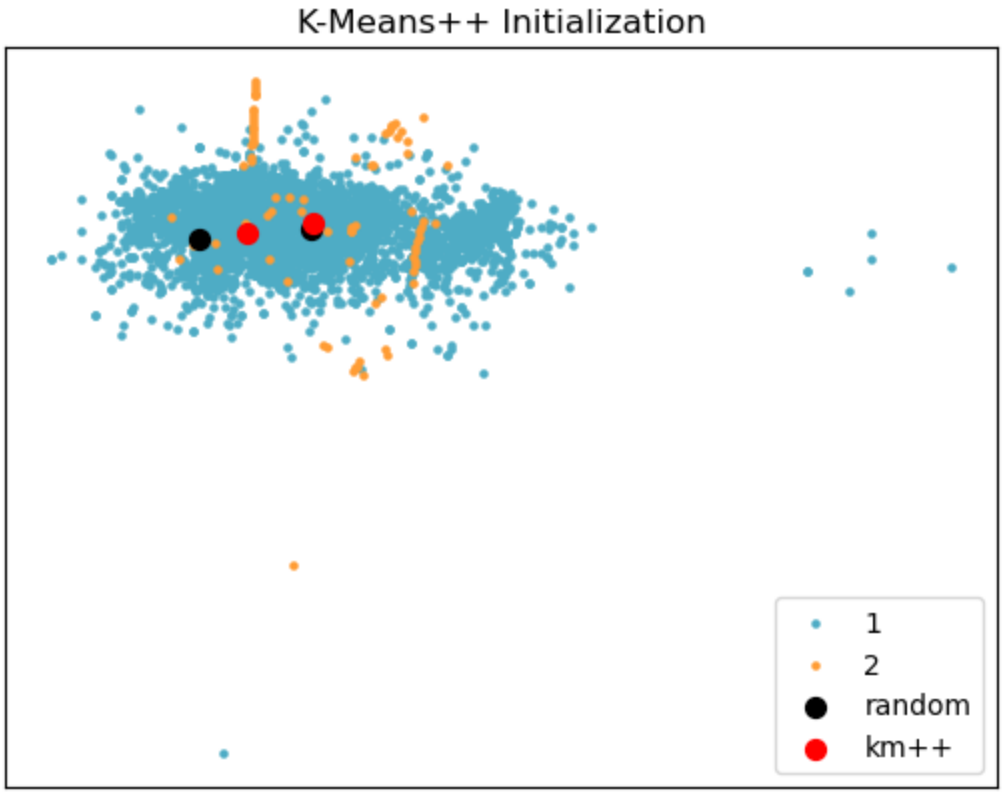
Train the model [10pts]

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!
```

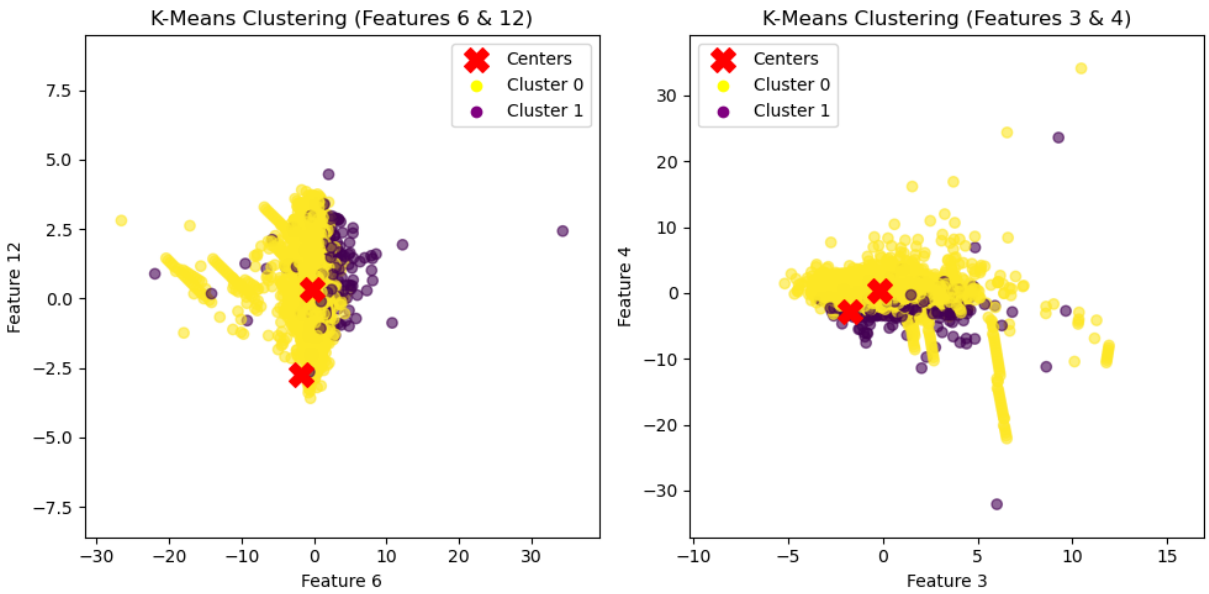
1.3 Visualize KMeans [0pts]

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 d
data = pd.read_csv("./data/creditcard.csv")
X = data.iloc[:, data.columns != "Class"].to_numpy()
k = 2
```

```
create_plots(X, k)
```



1.4 Fowlkes-Mallow Measure [5pts]

In this section, you will create a function to assess the quality of a clustering algorithm using Fowlkes-Mallow. The Fowlkes-Mallow Measure quantifies the goodness or quality of a clustering algorithm when compared to a ground truth.

As discussed in class, the computation is as follows:

$$FM = \frac{TP}{\sqrt{(TP + FN)(TP + FP)}}$$

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

TN (True Negative) is the count of pairs 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 pairs 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 pairs 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.

Refer to the class notes for more information on the Fowlkes-Mallow Measure

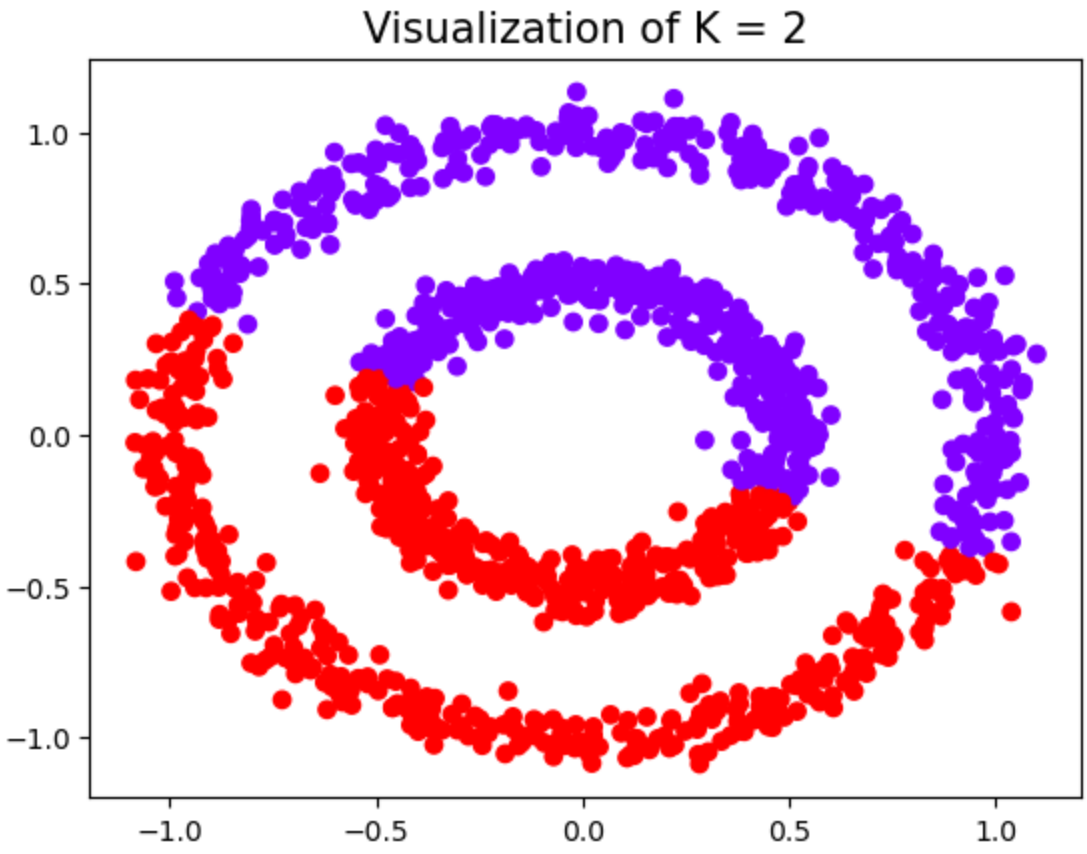
In [5]:
```
localtests.KMeansTests().test_fowlkes_mallow()
```

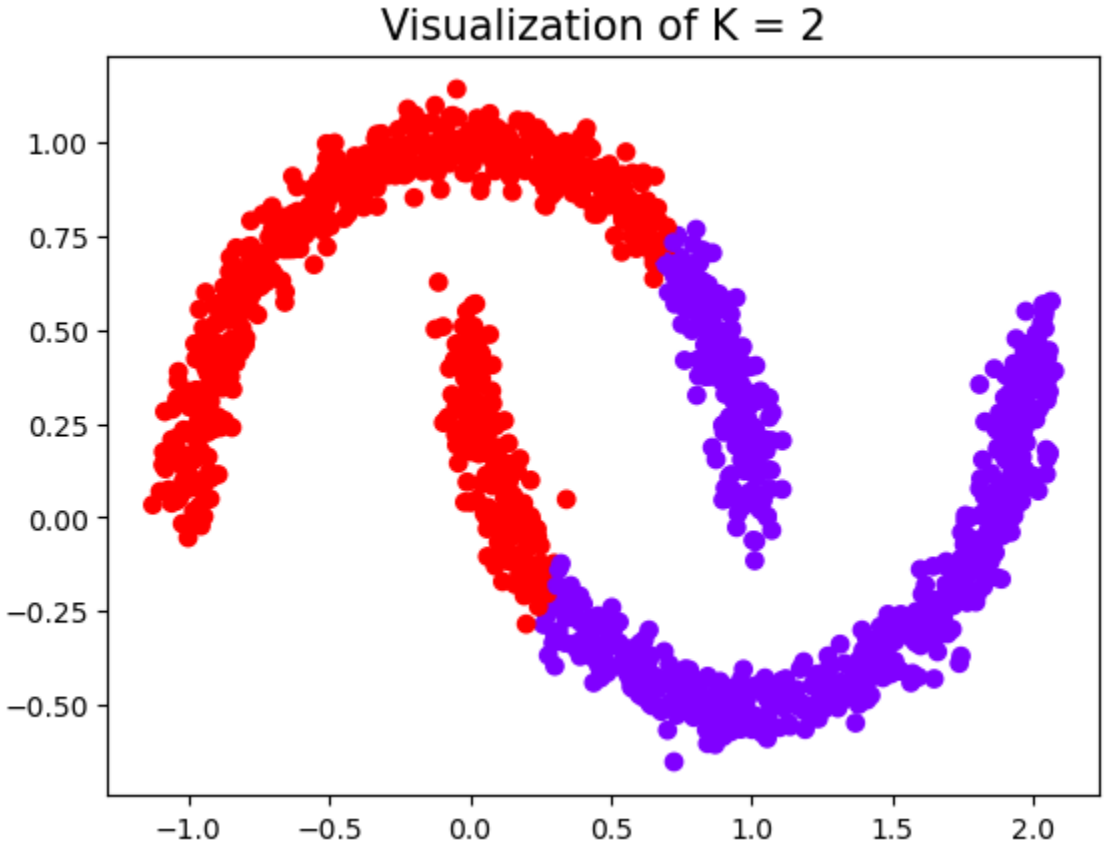
```
Expected value:  0.5852390484520126
Your value:  0.5852390484520126

UnitTest passed successfully!
```

1.5 Limitation of K-Means [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.

In [6]:
```
################################
### 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:

$$\mathbf{z} \sim Bernoulli(\alpha) = \begin{cases} \alpha & \text{if } z = 0 \\ 1 - \alpha & \text{if } z = 1 \end{cases}$$

$$p\left(\mathbf{x_n} \mid \mathbf{z} = \mathbf{0}\right) \sim \mathcal{N}(5\upsilon, 6\omega^2)$$

$$p\left(\mathbf{x_n} \mid \mathbf{z} = \mathbf{1}\right) \sim \mathcal{N}(3\upsilon, 5\omega^2)$$

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 $\mathcal{N}(x|5\upsilon, 6\omega^2)$ and $\mathcal{N}(x|3\upsilon, 5\omega^2)$ may be simpler
-- HINT: For this question suppose we have a Gaussian Distribution $\mathcal{N}(5\upsilon, 6\omega^2)$, it means $\mathcal{N}(\mu = 5\upsilon, \sigma^2 = 6\omega^2)$
-- 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 $\mathcal{N}(x|5\upsilon, 6\omega^2)$ and $\mathcal{N}(x|3\upsilon, 5\omega^2)$
-- 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 $\mathcal{N}(5\upsilon, 6\omega^2)$ and $\mathcal{N}(3\upsilon, 5\omega^2)$ 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

$$\ell(\theta|x) = \sum^{N} \sum_{k \in \{0,1\}}^{Z} p(z_k|x_n, \theta_{old}) ln\left[p(x_n, z_k|\theta)\right]$$

$$\ell(\upsilon, \omega^2, \alpha \mid x) = \sum^{N} \sum_{k \in \{0,1\}}^{Z} p(z_k \mid x_n, \theta_{old}) \ln\left[p(x_n, z_k \mid \mu_k, \sigma_k^2, \alpha)\right]$$

$$= \sum^{N} \sum_{k \in \{0,1\}}^{Z} p(z_k \mid x_n, \theta_{old}) \ln\left[p(z_k \mid \alpha) p(x_n \mid z_k, \upsilon, \omega^2)\right]$$

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. For your convenience, I have included the link below:

https://mahdi-roozbahani.github.io/CS46417641-spring2025/course/09-gaussian-mixture.pdf

2.1.1

$p(x) = \sum_z p(x \mid z)p(z)$

$p(x) = p(x \mid z=0)p(z=0) + p(x \mid z=1)p(z=1)$

$p(x) = \alpha\mathcal{N}(x \mid 5\upsilon, 6\omega^2) + (1-\alpha)\mathcal{N}(x \mid 3\upsilon, 5\omega^2)$

2.1.2

$p(z=0 \mid x) = \frac{p(x|z=0)p(z=0)}{p(x)}$

$\tau_0 = p(z=0 \mid x) = \frac{\alpha\mathcal{N}(x|5\upsilon,6\omega^2)}{\alpha\mathcal{N}(x|5\upsilon,6\omega^2)+(1-\alpha)\mathcal{N}(x|3\upsilon,5\omega^2)}$

$1-\tau_0 = p(z=1 \mid x) = \frac{(1-\alpha)\mathcal{N}(x|3\upsilon,5\omega^2)}{\alpha\mathcal{N}(x|5\upsilon,6\omega^2)+(1-\alpha)\mathcal{N}(x|3\upsilon,5\omega^2)}$

2.1.3

$c_0 = 6$ for the first Gaussian, $c_1 = 5$ for the second Gaussian

$\gamma_{n,0} = \tau_n$ and $\gamma_{n,1} = 1-\tau_n$

$\ln \mathcal{N}\big(x_n \mid \mu_k,\, c_k\, w^2\big) = -\frac{1}{2}\ln\big(2\pi\, c_k\, w^2\big) - \frac{(x_n-\mu_k)^2}{2\, c_k\, w^2}$

We only focus on:

$-\frac{1}{2}\ln(w^2) \;-\; \frac{(x_n-\mu_k)^2}{2\, c_k\, w^2}$

Now, take derivative: $\frac{\partial}{\partial w^2}\left[-\frac{1}{2}\ln(w^2)\right] = -\frac{1}{2}\,\frac{1}{w^2}, \qquad \frac{\partial}{\partial w^2}\left[-\frac{(x_n-\mu_k)^2}{2\, c_k\, w^2}\right] = \frac{(x_n-\mu_k)^2}{2\, c_k\, (w^2)^2}$

$\frac{\partial Q}{\partial w^2} = \sum_{n=1}^{N}\sum_{k=0}^{1}\gamma_{n,k}\left[-\frac{1}{2\, w^2} + \frac{(x_n-\mu_k)^2}{2\, c_k\, (w^2)^2}\right] = 0$

Multiply both sides by $2\omega^4$ and rearrange:

$\sum_{n=1}^{N}\sum_{k=0}^{1}\gamma_{n,k}\left[\frac{(x_n-\mu_k)^2}{c_k} - w^2\right] = 0 \quad \Longrightarrow \quad w^2 = \frac{\sum_{n=1}^{N}\sum_{k=0}^{1}\gamma_{n,k}\,\frac{(x_n-\mu_k)^2}{c_k}}{N}$

Finally, substitute c_0, μ_0 and c_1, μ_1

$$w^2 = \frac{1}{N} \sum_{n=1}^{N} \left[p(z=0 \mid x_n) \frac{(x_n - 5v)^2}{6} + p(z=1 \mid x_n) \frac{(x_n - 3v)^2}{5} \right]$$

$$w^2 = \frac{1}{N} \sum_{n=1}^{N} \left[\tau_n \frac{(x_n - 5\,v)^2}{6} + \left(1 - \tau_n\right) \frac{(x_n - 3\,v)^2}{5} \right]$$

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

2.2. 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. [5pts]
-- HINT: Think about the difference in updating parameters during each iteration. i.e. How many parameters need to be updated in gradient descent? What we did in for each iteration in EM algorithm to simplify it?

Instead of computing gradients and adjusting parameters slowly, EM can compute the expected value directly and update parameters in a single step. This makes iteration faster for EM than gradient ascent/descent. Also, gradient ascent needs to carefully determine step size because a small step size will result in slow convergence, and too big of a step size may cause a divergence.

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 $\mathcal{N}(\mu_k, \Sigma_k)$ 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:

$$\ln p(x_1, \ldots, x_N | \pi, \mu, \Sigma) = \sum_{i=1}^{N} \ln \left(\sum_{k=1}^{K} \pi(k) \mathcal{N}(x_i | \mu_k, \Sigma_k) \right) \tag{2}$$

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:

$$\tau\left(z_k\right) = \frac{\pi_k \mathcal{N}\left(x | \mu_k, \Sigma_k\right)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(x | \mu_j, \Sigma_j\right)}, \quad \text{for } k = 1, \ldots, K$$

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:

$$\mu_k^{new} = \frac{\sum_{n=1}^{N} \tau(z_k) x_n}{N_k} \tag{3}$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \tau(z_k)^T (x_n - \mu_k^{new})^T (x_n - \mu_k^{new}) \tag{4}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{5}$$

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
- The class notes assume that your dataset X is (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 $\pi(k)$ 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 $(N \times 3)$ 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 [15pts]

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
  - For an intuitive understanding on why this helps us, consider plotting e^{-x} and e^{x} on a graphing calculator when $x \geq 0$

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 + 2pts Written Questions]

Given $logit \in \mathbb{R}^{N \times D}$, calculate $s \in \mathbb{R}^{N}$, where $s_i = \log\left(\sum_{j=1}^{D} \exp(logit_{i,j})\right)$. Again, pay attention to the numerical problem. You may face similar conditions to the softmax function due to $logit_{i,j}$ being large. Therefore, 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 [2pts]:

1. Why should we add the maximum for each row of $logit$ to **logsumexp()** function? Show your reason by calculating and observing the relationship between F and s_1.
   -- Use a simple input like $logit \in \mathbb{R}^{1\times 3}$ and work through a mathematical example.
   -- Let N=1, D=3, $logit = \{logit_{11}, logit_{12}, logit_{13}\}$ and $max = logit_{13}$ is the maximum for this row. F is the array that subtracts the maximum for each row of $logits$.
   -- Start by subtracting the max of the row from each element in
   $s_1 = \log\left(\sum_{j=1}^{D} \exp(logit_{1,j})\right)$

$logit = \{logit_{11}, logit_{12}, logit_{13}\}$ and $max = logit_{13}$

$s_i = \log\left(\sum_{j=1}^{D} \exp(logit_{i,j})\right)$

$s_1 = \log\left(\sum_{j=1}^{3} \exp(logit_{1,j})\right)$

$s_1 = \log(exp(logit_{11}) + exp(logit_{12}) + exp(logit_{13}))$

$F_j = logit_{1,j} - max(logit) = logit_{1,j} - logit_{13}$

$s_1 = \max(\text{logit}) + \log\left(\sum_{j=1}^{3} \exp(\text{logit}_{1,j} - \max(\text{logit}))\right)$

Now, show why we should add the maximum

$\sum_j \exp(\text{logit}_{i,j} - \max(\text{logit}))$

$= \frac{1}{\exp(\max(\text{logit}))} \sum_j \exp(\text{logit}_{i,j})$

Take log

$\log\left(\frac{1}{\exp(\max(\text{logit}))} \sum_j \exp(\text{logit}_{i,j})\right)$

$= -\max(\text{logit}) + \log\left(\sum_j \exp(\text{logit}_{i,j})\right)$

Therefore we have to add max(logit) back to ensure mathematical correctness.

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_i^2 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:

$$\ell(\theta) = \sum_{i=1}^{N} \ln \left(\sum_{k=1}^{K} \pi(k) \mathcal{N}(x_i | \mu_k, \Sigma_k) \right) \tag{6}$$

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

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

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.
  $\text{log-likelihood}[i, k] = \log \left(\pi_k + \text{1e-32}\right) + \log \left(\mathcal{N}(x_i | \mu_k, \Sigma_k) + 1e\text{-}32\right)$. 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 [ ]:
```
gmm_tester.test_grad()
```

3.3 Image Compression and pixel clustering [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 [10]:
```
gmm_tester.test_undergrad_image_compression()
```

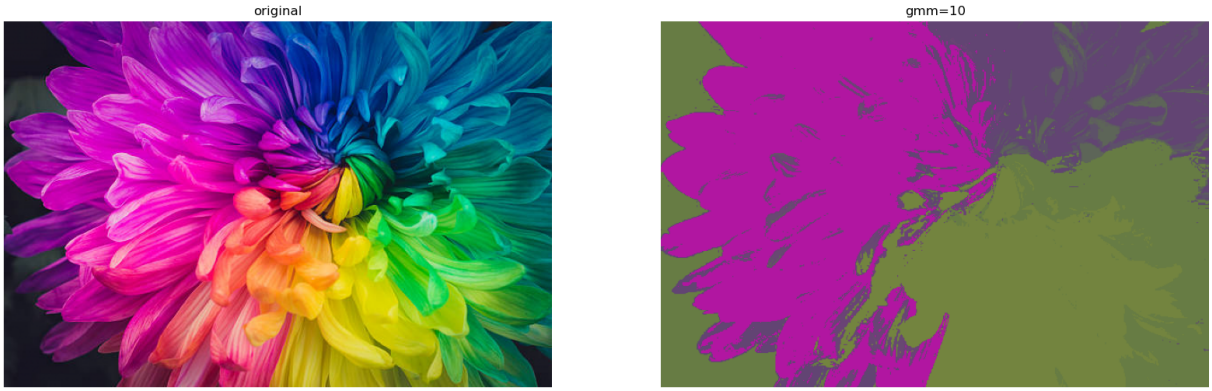
```
iter 9, loss: 7347731.1618: 100%|██████████| 10/10 [00:06<00:00,  1.46it/s]
Your image compression within the expected range:  True
```

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

```
iter 9, loss: 7347731.1618: 100%|██████████| 10/10 [00:07<00:00,  1.42it/s]
Your image compression within the expected range:  False
```

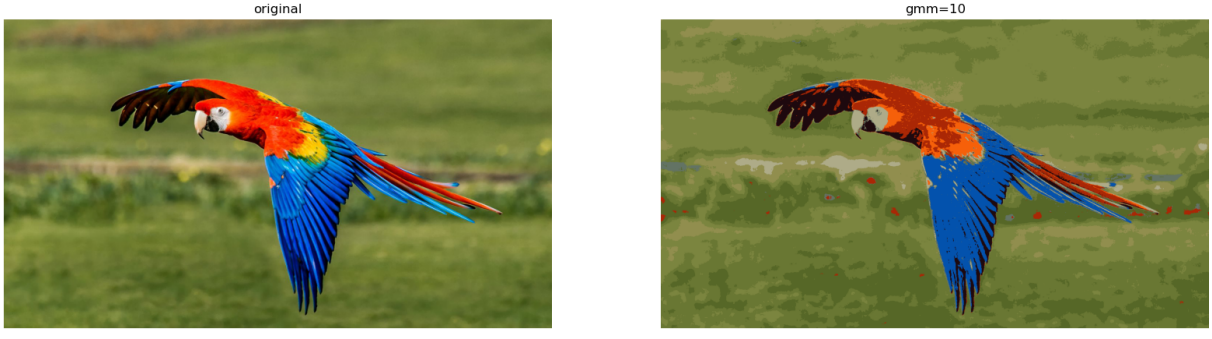
In [11]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################

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
        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)
```

```
/var/folders/br/15kx5hqn62v_m0r3lzbh_tsw0000gn/T/ipykernel_91748/162679580
5.py:15: DeprecationWarning: Starting with ImageIO v3 the behavior of this f
unction 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)
iter 9, loss: 4204443.7039: 100%|██████████| 10/10 [00:02<00:00,  4.20it/s]
```

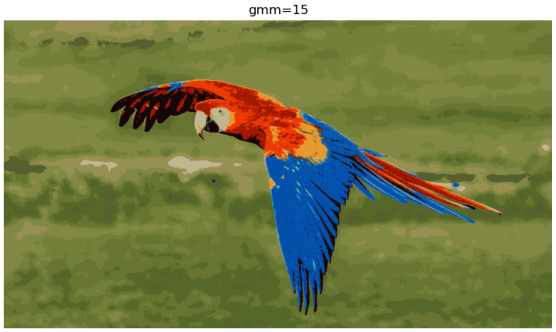



```
iter 9, loss: 4139754.2074: 100%|██████████| 10/10 [00:03<00:00,  2.87it/s]
```


```
/var/folders/br/15kx5hqn62v_m0r3lzbh_tsw0000gn/T/ipykernel_91748/162679580
5.py:18: DeprecationWarning: Starting with ImageIO v3 the behavior of this f
unction 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)
iter 9, loss: 15218212.0677: 100%|██████████| 10/10 [00:11<00:00,  1.14s/it]
```




```
iter 9, loss: 14928081.7858: 100%|██████████| 10/10 [00:17<00:00,  1.73s/it]
```



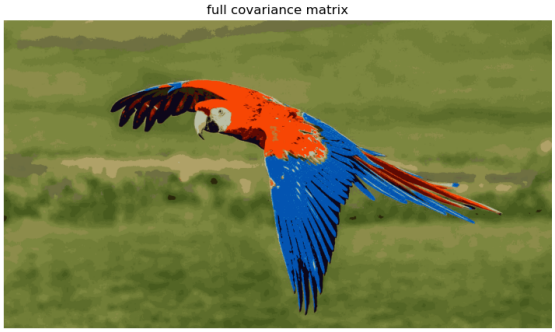
3.4 Compare full covariance matrix with diagonal covariance matrix [1% Bonus for All]

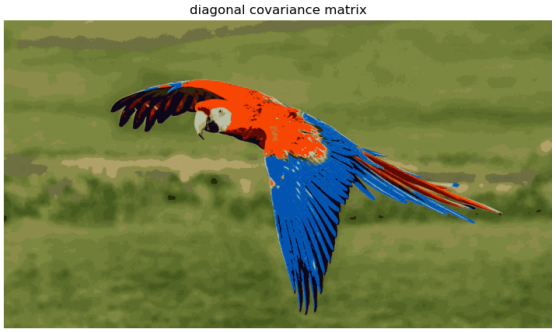
Compare the results of clustering an image with full covariance matrix and diagonal covariance matrix. Can you explain why the images are different with same clusters? Note: You will have to implement both multinormalPDF and normalPDF, and add a few arguments in the original _ll_joint(), _M_step(), _E_step()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

In [12]:
```
################################
### 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 covaria
    """
    # 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 [13]:
```
compare_matrix(image2, 20)
```

```
iter 9, loss: 14524020.8895: 100%|██████████| 10/10 [00:25<00:00,  2.51s/it]
iter 9, loss: 14524020.8895: 100%|██████████| 10/10 [00:25<00:00,  2.55s/it]
```





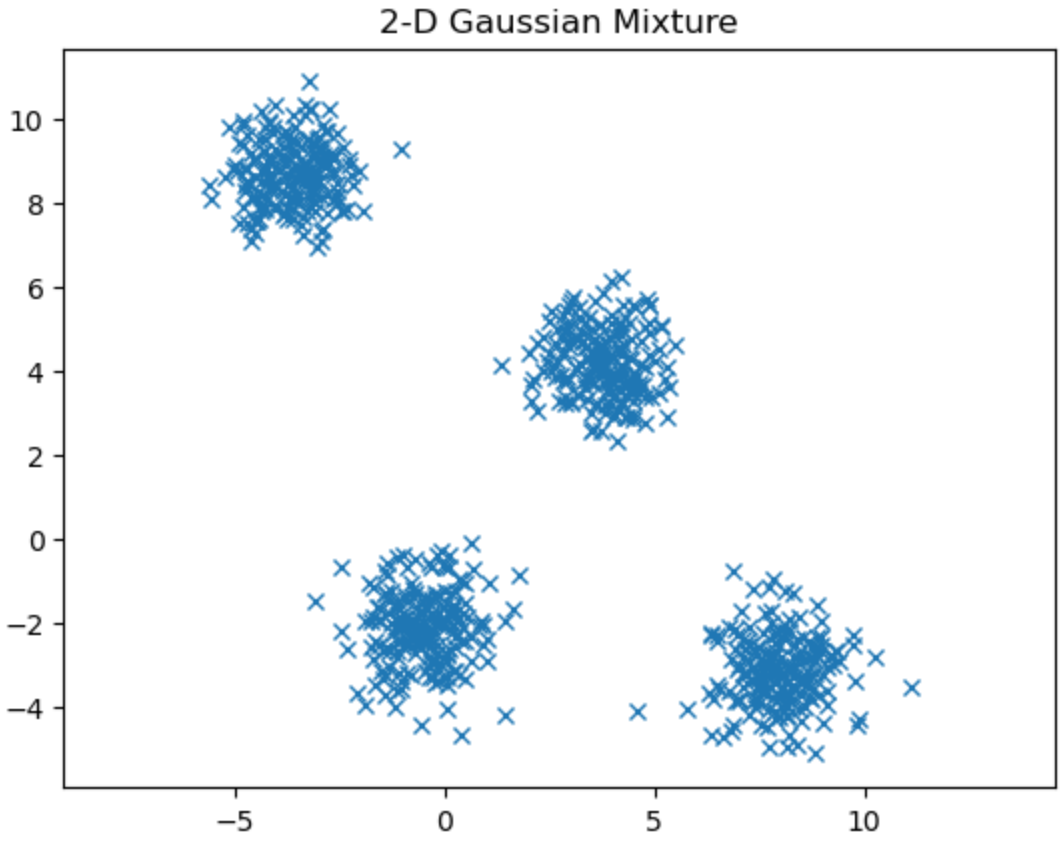
3.5 Generate samples from a Gaussian Mixture [5pts]

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 [14]:
```
################################
### 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 [15]:
```
# Fit your GMM implementation
K = 4
gmm = GMM(data, K)
tau, (pi, mu, sigma) = gmm()
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
iter 6, loss: 3021.7672:   7%|▋         | 7/100 [00:00<00:00, 1106.89it/s]
iter 6, loss: 3021.7672:   7%|▋         | 7/100 [00:00<00:00, 1106.89it/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=4 to represent 4 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 heve 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

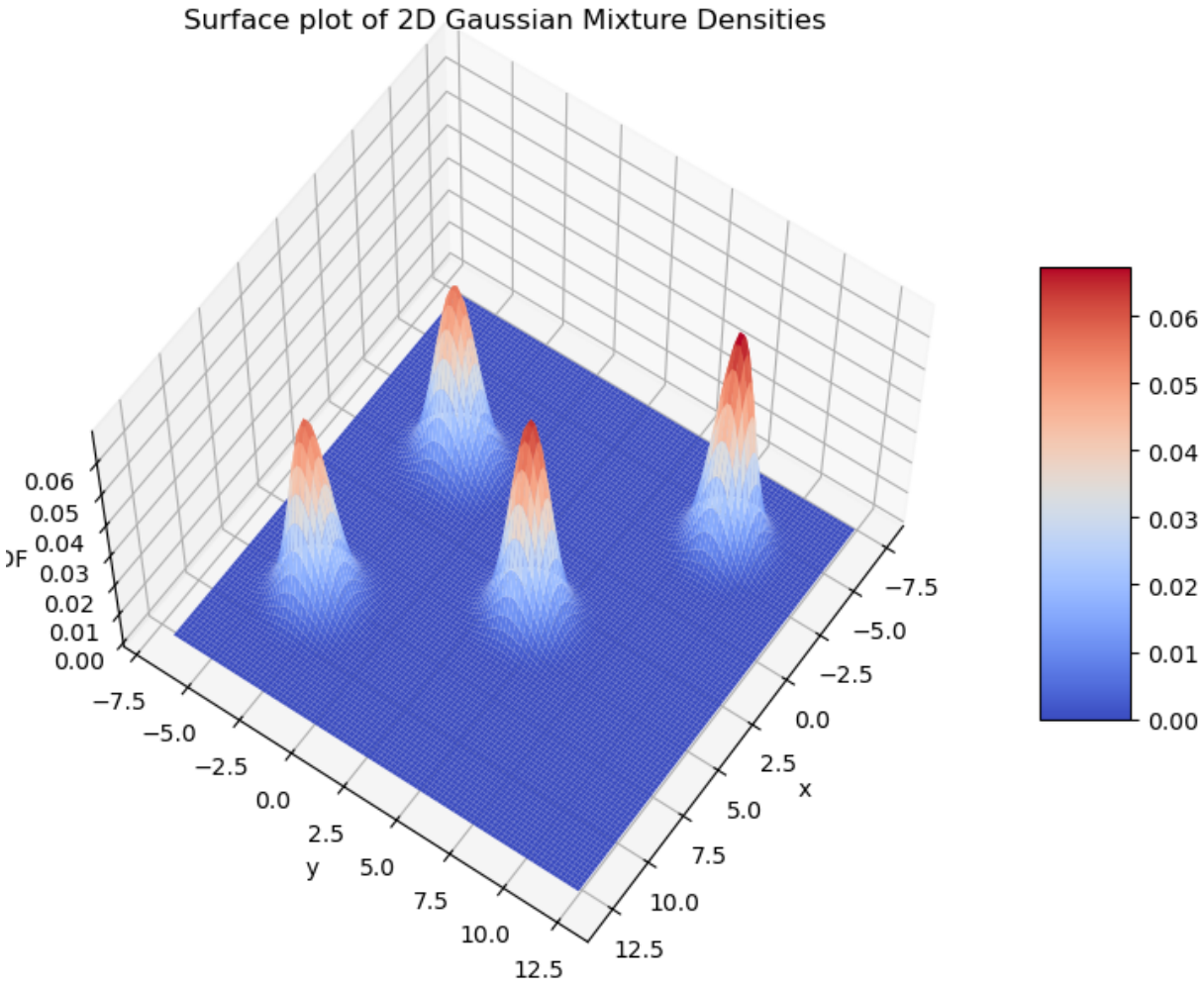
Conventionally we think of Gaussian Mixture Models as a form of soft clustering, but you can also think of them as an algorithm for estimating density of data points with gaussians. Thus we can take an arbitary data point and using the gaussian mixture model as an estimation for the density at a given 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 [16]:
```
################################
### 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
```

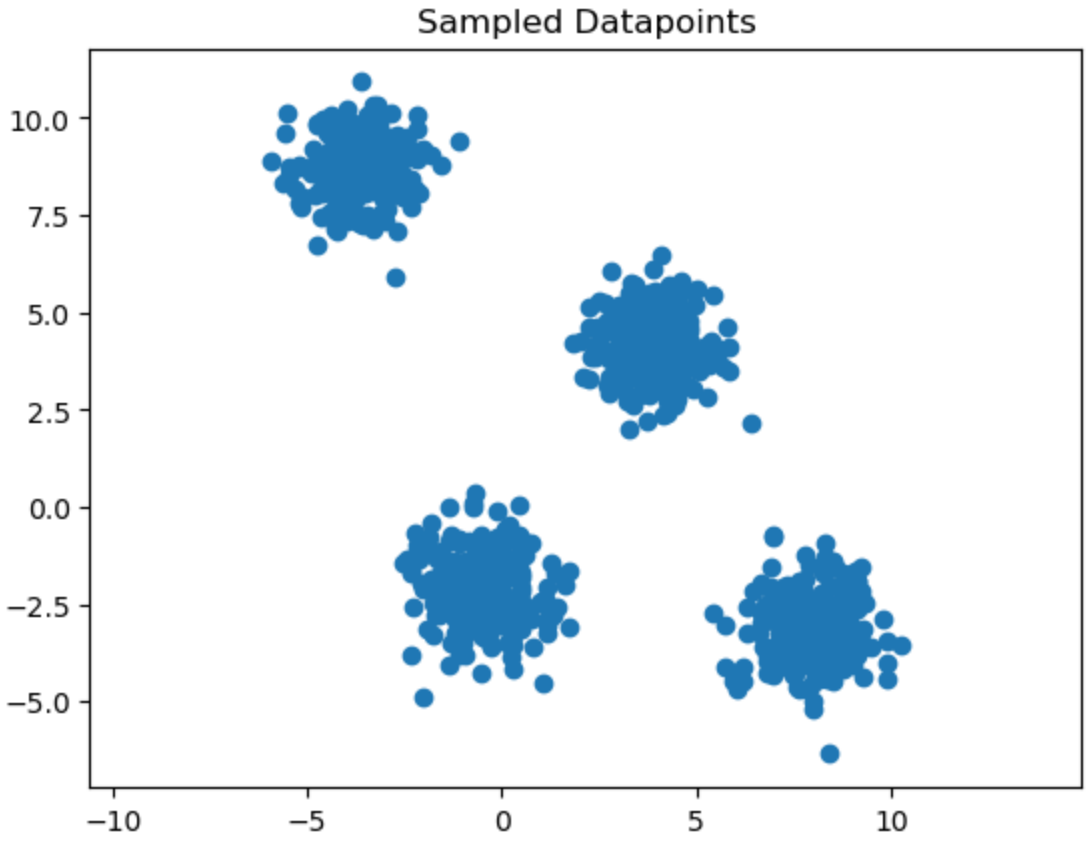
In [17]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################


# 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="non
)
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 [19]:
```
################################
### 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_sampl
        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 [20]:
```
################################
### 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 what if we used 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 [21]:
```
# Feel free to change the below parameters

full_matrix = False
K = 4
max_iters = 100
rel_tol = 1e-3
```

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

df = sns.load_dataset("iris")
```

```
data = np.array(df.drop("species", axis=1))

student_gmm_random = GMM(data, K, max_iters=max_iters)
student_gmm_random.__call__(full_matrix=full_matrix, kmeans_init=False, rel_
num_iters_random = student_gmm_random.num_iters

student_gmm_kmeans = GMM(data, K, max_iters=max_iters)
student_gmm_kmeans.__call__(full_matrix=full_matrix, kmeans_init=True, rel_t
num_iters_kmeans = student_gmm_kmeans.num_iters

print("Random num iterations: ", num_iters_random)
print("KMeans num iterations: ", num_iters_kmeans)
```

```
iter 8, loss: 276.0000:   9%|▉         | 9/100 [00:00<00:00, 1572.34it/s]
iter 9, loss: 265.1210:  10%|▉         | 10/100 [00:00<00:00, 1752.23it/s]
Random num iterations:  10
KMeans num iterations:  11
```

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 was slower or faster (~1-2 sentences)?

KMeans initialization should allow GMM to converge faster even if the result above does not represent that. The reason is that KMeans initialization provides more stable and well-separated initial cluster centers which effectively reduces the the number of iterations needed to converge.

4. (Bonus for All) Cleaning Messy data and semi-supervised learning [8% 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%]

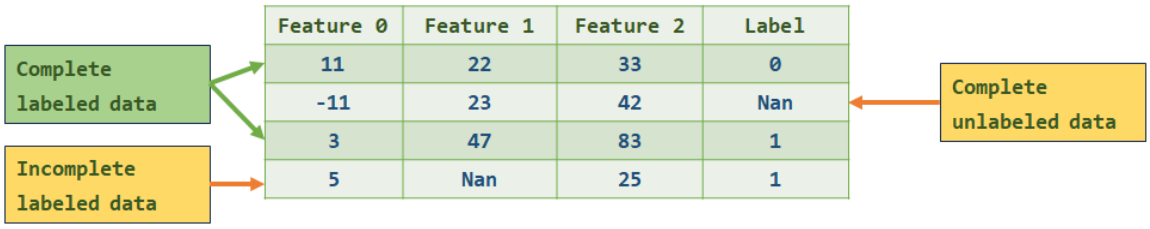
The first step is to break up the whole dataset into clear parts. All the data is randomly shuffled in one csv file. In order to move forward, the data needs to be split into three separate arrays:

- 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 [ ]:
```
###############################
### DO NOT CHANGE THIS CELL ###
###############################

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

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

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

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

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

4.2 Semi-supervised Learning [3.5%]

Semi-supervised learning is a type of machine learning that falls between supervised and unsupervised learning. It involves training a model on a dataset that contains both labeled and unlabeled data. Typically, a small portion of the dataset is labeled, while the majority remains unlabeled. This approach leverages the limited labeled data to guide the learning process while making use of the vast amount of unlabeled data to improve performance. Semi-supervised learning is particularly useful when labeling data is expensive or time-consuming, making it a powerful tool in many real-world AI applications. One such scenario is text labeling as you'll see in the paper provided 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 [ ]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
from semisupervised import ComparePerformance

ComparePerformance.accuracy_comparison()
```

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 [x points]

5.1 Hierarchical Clustering Implementation [x pts]

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:

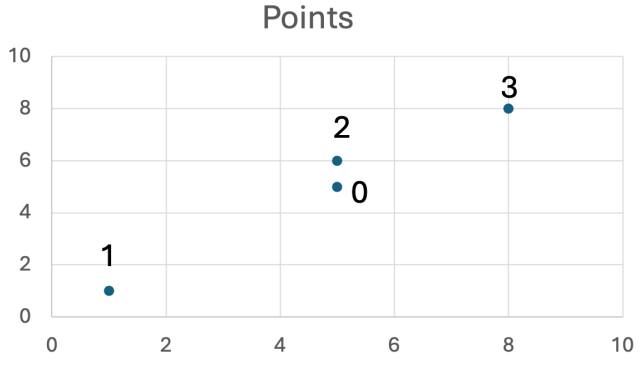
1. `points` : N x D numpy array where N is the number of points, and D is the dimensionality of each point. This is your dataset.
2. `distance` : N x N symmetric numpy array which stores pairwise distances between clusters. The distance between a cluster and itself should be `np.inf` in order to help 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`

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

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

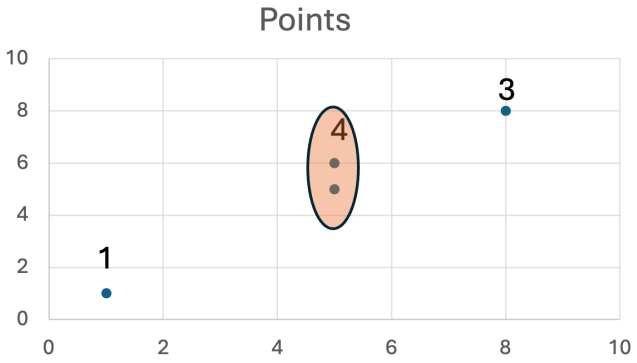
clustering = `[[0, 0, 0, 0], …, [0, 0, 0, 0]]`

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

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]]`

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

distances

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

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

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

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

In [ ]:
```
################################
### 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", "blac
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 [ ]:
```
! git clone https://github.com/koonimaru/radialtree.git
! pip install radialtree/
! rm -rf radialtree
```

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

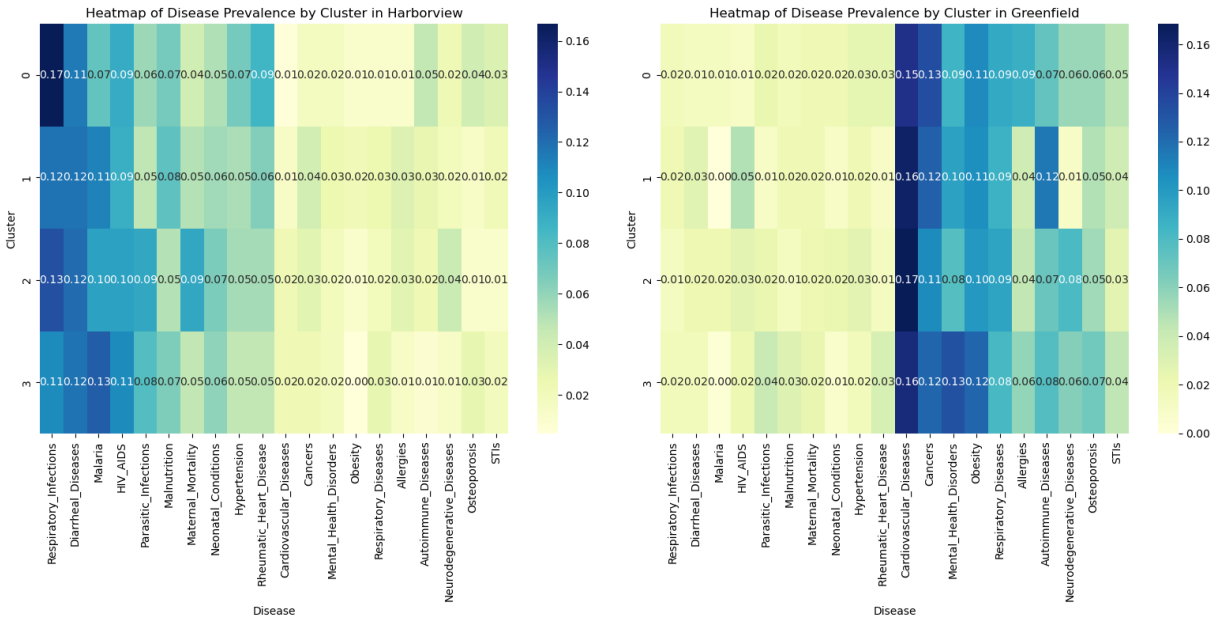
```
fig, axes = plt.subplots(2, 1, figsize=(20, 15))

# 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})
```

6. Evaluating Data Representation in K-Means Clustering [4pts]

A national healthcare system employs a K-means clustering algorithm to optimize healthcare resource distribution. Two datasets are used: one from Harborview, an underdeveloped city, and another from Greenfield, a developed city. The algorithm is applied to both datasets to identify healthcare resource allocation needs.

Datasets from vastly different settings might differ in incidence of chronic conditions, or preventive care and better healthcare access. Compare the heatmaps below:

In [23]:
```
%run Vis_DoNotChange.py
```



Question:

Which of the following statements are correct? (Select all that apply)

- A. Sensitivity to the scale and scope of the data might result in the urgency of

certain health conditions being overlooked.

- B. Uniform resource distribution between cities is the best ethical approach.
- C. Concentrating on general data trends, the algorithm may overlook the specific healthcare needs of smaller, underrepresented groups.
- D. If data contains underrepresentation of certain demographic groups, existing disparities (as shown by the plot) might be magnified.
- E. The outputs of unsupervised algorithms like K-means clustering are inherently unbiased, as they do not rely on pre-labeled data.

Answer: A, C, D