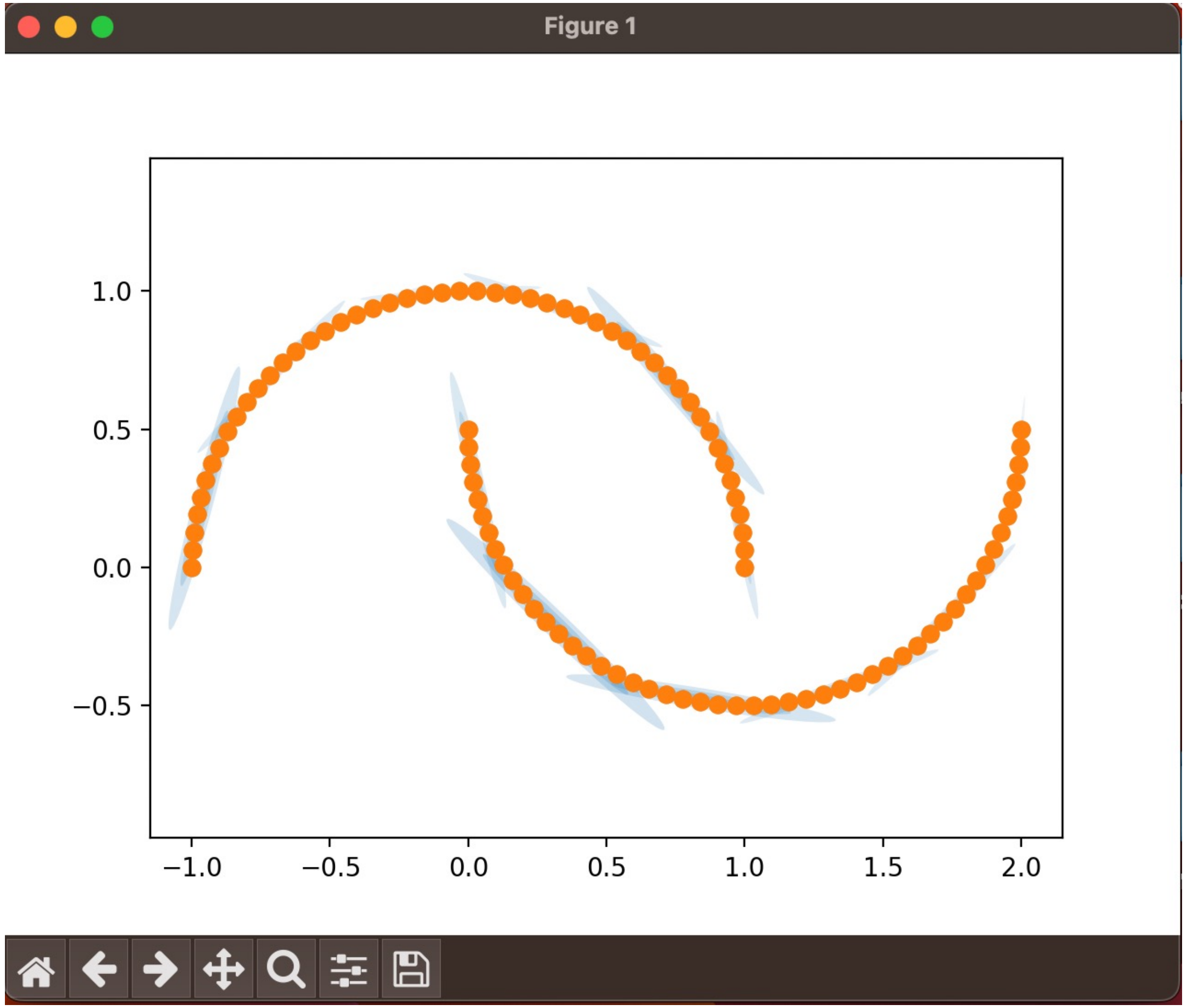
```
# Tashrif Apon
# 4330 Final

# sources:
https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html
# OpenAI
```

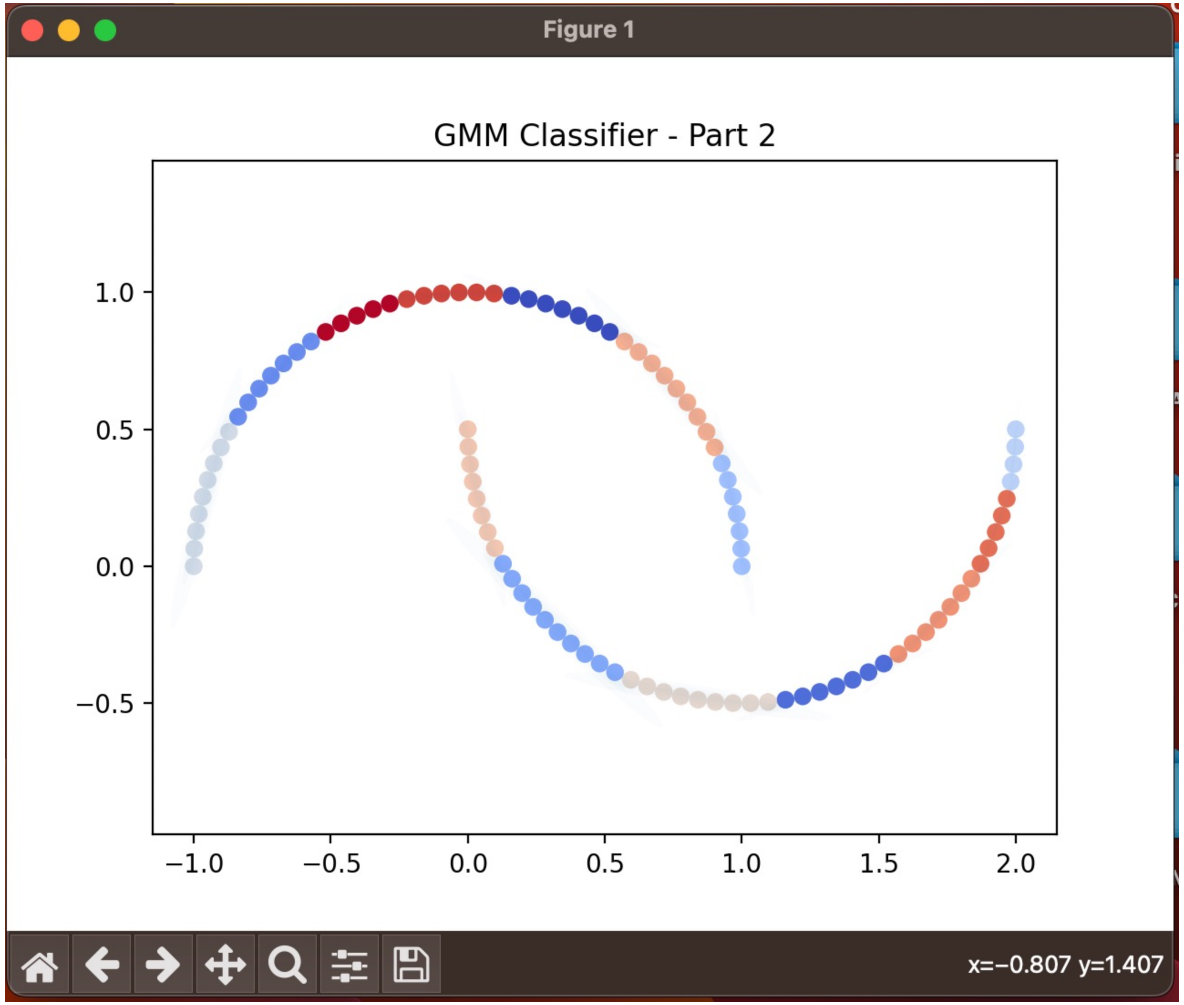
Use of BIC to figure out optimal component's

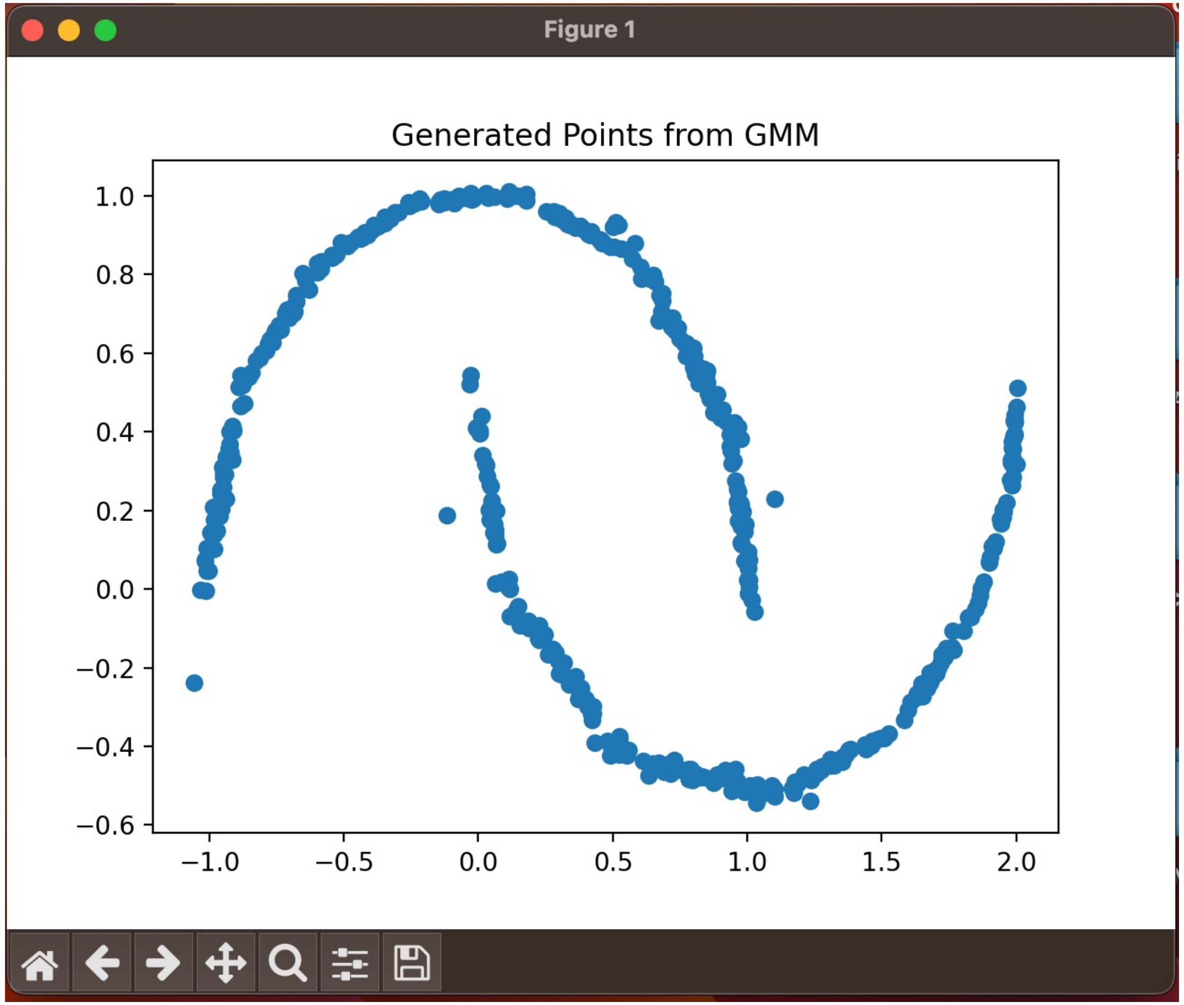
```
# optimal components
'''
n_components = np.arange(10, 16) # Prof.'s og recs
models = [GaussianMixture(n, covariance_type='full',
random_state=None).fit(Xmoon)
for n in n_components]
for m in models:
print(m.bic(Xmoon))

# 14 is the sweet spot # used lowest BIC
'''
```

Part 1 Graph



Part 2 Graphs – I think it is a good fit





Part 3: My computer chip is not powerful enough, so I used Amir for help. He said he got this after running for 10 minutes

Name	Type	Size	Value
accuracy	float64	1	0.44
accuracy_supervised	float64	1	0.2317857142857143
aic_semi_supervised	float64	1	2925245.3128358526
aic_unsupervised	float64	1	2925245.3128358526
ax	axes._axes.Axes	1	Axes object of matplotlib.axes._axes module
axes	Array of object	(10,)	ndarray object of numpy module
boosted_tree_clf	ensemble._gb.GradientBoostingClassifier	100	GradientBoostingClassifier object of sklearn.ensemble._gb module
cumulative_var_ratio	Array of float64	(80,)	[0.09746116 0.16901561 0.23051091 ... 0.88791333 0.88934768 0.89072103 ...
data	utils._bunch.Bunch	9	Bunch object of sklearn.utils._bunch module
fig	figure.Figure	1	Figure object of matplotlib.figure module
gb_classifier	ensemble._gb.GradientBoostingClassifier	100	GradientBoostingClassifier object of sklearn.ensemble._gb module
generated_samples	Array of float64	(1000, 42)	[[-2.84862016 -1.60566405 0.3779173 ... -0.01186339 0.16333381 0 ...
gmm	mixture._gaussian_mixture.GaussianMixture	1	GaussianMixture object of sklearn.mixture._gaussian_mixture module
gmm_semi_supervised	mixture._gaussian_mixture.GaussianMixture	1	GaussianMixture object of sklearn.mixture._gaussian_mixture module
gmm_supervised	mixture._gaussian_mixture.GaussianMixture	1	GaussianMixture object of sklearn.mixture._gaussian_mixture module
gmm_unsupervised	mixture._gaussian_mixture.GaussianMixture	1	GaussianMixture object of sklearn.mixture._gaussian_mixture module
i	int	1	9
image	Array of float64	(28, 28)	[[-3.62567239e-16 -1.06875290e-15 -1.23105166e-15 ... -2.22363583e-16 ...
labels_gmm	Array of int64	(1000,)	[0 0 0 ... 41 41 41]
labels_semi_supervised	Array of int64	(70000,)	[8 1 40 ... 23 30 27]
labels_true	Array of int32	(1000,)	[0 0 0 ... 41 41 41]
labels_unsupervised	Array of int64	(70000,)	[8 1 40 ... 23 30 27]
means	Array of float64	(10, 100)	[[-3.33288185e+00 -2.32456814e+00 8.84757817e-01 ... -2.95554853e-02 ...
means_images	Array of float64	(42, 28, 28)	[[[2.62244610e-15 4.55898150e-16 -1.96558673e-15 ... 1.09060478e-18 ...
means_original	Array of float64	(42, 784)	[[2.62244610e-15 4.55898150e-16 -1.96558673e-15 ... 0.00000000e+00 ...
means_pca	Array of float64	(42, 42)	[[-2.85783215 -2.19253968 1.12589721 ... 0.29701793 0.31454958 0 ...
mnist	utils._bunch.Bunch	9	Bunch object of sklearn.utils._bunch module
n_components	int	1	42
n_gaussians	int	1	42
n_samples	int	1	1000
num_classes	int	1	10
pca	decomposition._pca.PCA	1	PCA object of sklearn.decomposition._pca module
sample	Array of float64	(42,)	[0.45617351 1.89373547 0.2191235 ... -0.87972909 0.39891762 -0.0 ...
X	DataFrame	(70000, 784)	Column names: pixel1, pixel2, pixel3, pixel4, pixel5, pixel6, pixel7, ...
X_pca	Array of float64	(70000, 42)	[[0.47943237 -1.24013272 -0.20051698 ... -1.48883497 0.54804951 0 ...
X_test	Array of float64	(200, 42)	[[-2.96656921 -0.59626595 -0.78312011 ... 0.18782344 -0.0201581

```python
from sklearn.datasets import make_moons
from sklearn.mixture import GaussianMixture

from sklearn.decomposition import PCA
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt
import numpy as np
                                                         # Part 1
# get data & plot
Xmoon, ymoon = make_moons(100, noise=None, random_state=None)
plt.scatter(Xmoon[:, 0], Xmoon[:, 1])

# optimal components
'''
n_components = np.arange(10, 16) # Prof.'s og recs
models = [GaussianMixture(n, covariance_type='full', random_state=None).fit(Xmoon)
          for n in n_components]
for m in models:
    print(m.bic(Xmoon))

# 14 is the sweet spot # used lowest BIC
'''
```

```python
# GMM fitting & plot
from matplotlib.patches import Ellipse

def draw_ellipse(position, covariance, ax=None, **kwargs):
    ax = ax or plt.gca()

    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height, angle=angle, **kwargs))


def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)
    ax.axis('equal')

    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)

gmm = GaussianMixture(n_components=14, covariance_type='full', random_state=None)
gmm.fit(Xmoon)
plot_gmm(gmm, Xmoon, label=False)
plt.show()

```

```
                                                         # Part 2
# Assign labels based on the moon each point belongs to
labels = np.zeros(Xmoon.shape[0])
labels[Xmoon[:, 1] > 0.5] = 1 # Sometimes .5 isn't the best standard, but I played around with the ".AB" and this is what I am content with

# GMM fitting done in "Part 1"
# prob of point2moon (not a simple ratio; just a descriptor)
probs = gmm.predict_proba(Xmoon)

# Plotting the level sets of the GMM components and coloring the points according to the class assigned by the GMM
def plot_gmm_classification(gmm, X, labels, ax=None):
    ax = ax or plt.gca()
    labels_predict = gmm.predict(X)
    ax.scatter(X[:, 0], X[:, 1], c=labels_predict, cmap='coolwarm', marker='o')
    ax.axis('equal')
    ax.set_title('GMM Classifier - Part 2')
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * 0.2, ax=ax)

plot_gmm_classification(gmm, Xmoon, labels)
plt.show()

# Generating points from GMM
X_new, y_new = gmm.sample(500)
plt.scatter(X_new[:, 0], X_new[:, 1])
plt.title('Generated Points from GMM')
plt.show()

```

```
                                                         # Part 3
from sklearn.datasets import fetch_openml
# get data
mnist = fetch_openml('mnist_784', version=1, parser='auto')
X3, y = mnist['data'], mnist['target'].astype(int)

# Perform PCA first then adjust components
from sklearn.decomposition import PCA
pca = PCA(0.99, whiten=True)
X_pca = pca.fit_transform(X3)
#print(data.shape) # (70K, 331)
    #Test # I am just going with the minimum bc I am scared
'''
n_components = np.arange(30, 100)
models = [GaussianMixture(n, covariance_type='full', random_state=None)
          for n in n_components]
bics = [model.fit(data).bic(data) for model in models]
min_bic = min(bics)
print( bics.index(min_bic) )
'''


# GMM density estimator
gmm_density = GaussianMixture(n_components=30, covariance_type='full', random_state=None)
gmm_density.fit(X_pca)

# GMM classifier
gmm_classifier = GaussianMixture(n_components=30, covariance_type='full', random_state=None)
gmm_classifier.fit(X_pca, y)

```

```
# Plot means of GMM components for every class
def plot_means(gmm, pca, ax):
    means_proj = pca.inverse_transform(gmm.means_)
    for i in range(10):
        ax[i].imshow(means_proj[i].reshape(30, 30), cmap='gray')
        ax[i].set_title(f'Mean of Class {i}')
        ax[i].axis('off')

fig, axs = plt.subplots(2, 5, figsize=(15, 6))
plot_means(gmm_classifier, pca, axs.ravel())
plt.tight_layout()
plt.show()

# GMM samples
X_new3, y_new3 = gmm_density.sample(10)
X_new3 = pca.inverse_transform(X_new3)

# Plot sampled digits
plt.figure(figsize=(12, 6))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(X_new3[i].reshape(30, 30), cmap='gray')
    plt.title(f'Sample {i}')
    plt.axis('off')
plt.tight_layout()
plt.show()

# GMM classification vs. boosted gradient tree
            #   ACCURACY
labels_pred = gmm_classifier.predict(X_pca)
bgt_clsf = GradientBoostingClassifier(random_state=None)
bgt_clsf.fit(X_pca, y)
acc_gmm = accuracy_score(y, labels_pred) # compares list @ every index, which is fine for this
acc_bgt = accuracy_score(y, bgt_clsf.predict(X_pca))
print(f'Accuracy of GMM Classifier: {acc_gmm}')
print(f'Accuracy of Boosted Gradient Tree: {acc_bgt}')
```

```
# sorry, it took too long.
# my computer started heating up, and I have a MacBook Pro (M2 PRO chip) for reference
# before I gave up on part 3: I ran it 2 or 3 times, getting the same dimensionality error
# I looked through the link and asked OpenAI, but neither helped
```