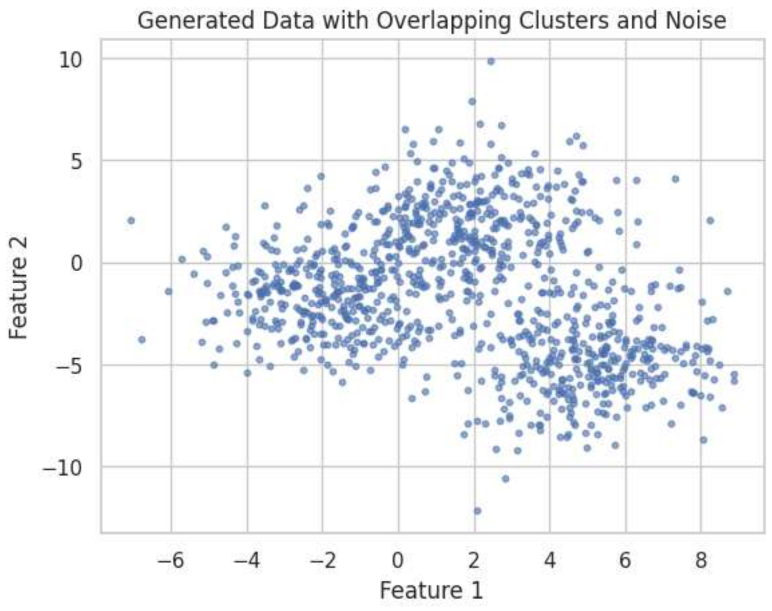
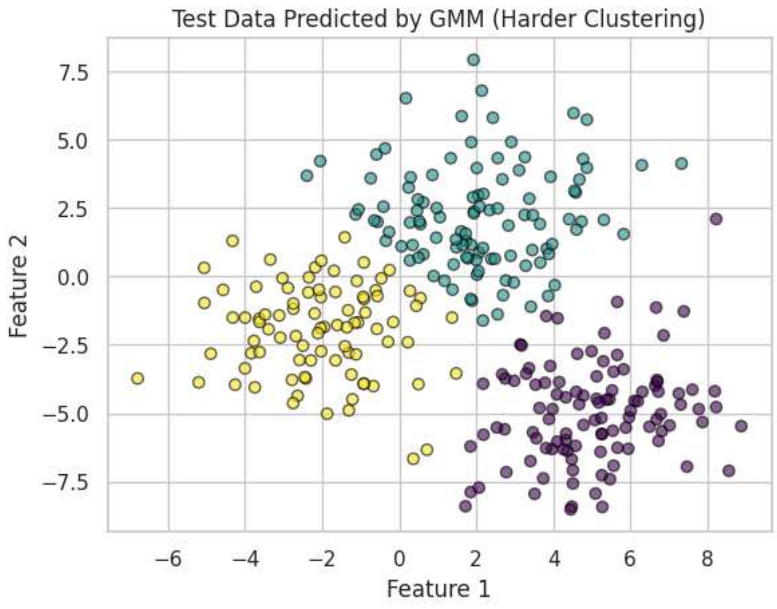
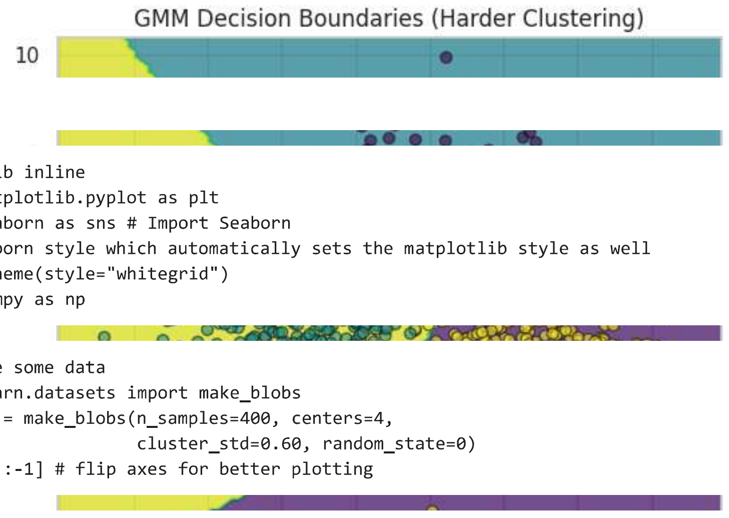
Gaussian Mixture Model

```python
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 1. Generate Data with Overlap and Noise
# Generate data with three centers but add noise and adjust covariance to make it harder to cluster
np.random.seed(42)

# Cluster centers
centers = [[2, 2], [-2, -2], [5, -5]]
# Covariances that introduce overlap
covariances = [
    [[1, 0.8], [0.8, 1]],   # Elliptical, high correlation between features
    [[1, -0.6], [-0.6, 1]],  # Another elliptical cluster, but with negative correlation
    [[3, 1.5], [1.5, 3]]     # Larger cluster with higher spread
]

# Create blobs with overlapping clusters and noise
X, y = make_blobs(n_samples=1000, centers=centers, cluster_std=1.5, random_state=42)

# Add noise to the data to make the clusters harder to separate
X += np.random.normal(0, 1, size=X.shape)

# Visualize the generated data
plt.scatter(X[:, 0], X[:, 1], s=10, alpha=0.6)
plt.title("Generated Data with Overlapping Clusters and Noise")
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# 2. Train/Test Split
# Split the dataset into training and testing sets
X_train, X_test = train_test_split(X, test_size=0.3, random_state=42)

# 3. Gaussian Mixture Model
# Let's set n_components=3 (since we know there are 3 clusters)
gmm = GaussianMixture(n_components=3, covariance_type='full', random_state=42)
gmm.fit(X_train)

# 4. Make Predictions
# Predict the clusters for the test data
y_pred = gmm.predict(X_test)

# 5. Evaluate the Model (optional)
# Since this is a clustering problem, accuracy doesn't match well with true labels.
# But for the sake of demonstration, let's check the match.
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# 6. Visualize the GMM Results
# Visualize the predicted clusters on the test data
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', marker='o', edgecolors='black', alpha=0.6)
plt.title("Test Data Predicted by GMM (Harder Clustering)")
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# Optionally, let's visualize the GMM's learned decision boundaries
# To do this, let's create a grid of points and predict for each point
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
Z = gmm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.contourf(xx, yy, Z, alpha=0.75, cmap='viridis')
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap='viridis', alpha=0.6)
plt.title("GMM Decision Boundaries (Harder Clustering)")
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



```
Accuracy: 0.3000
```



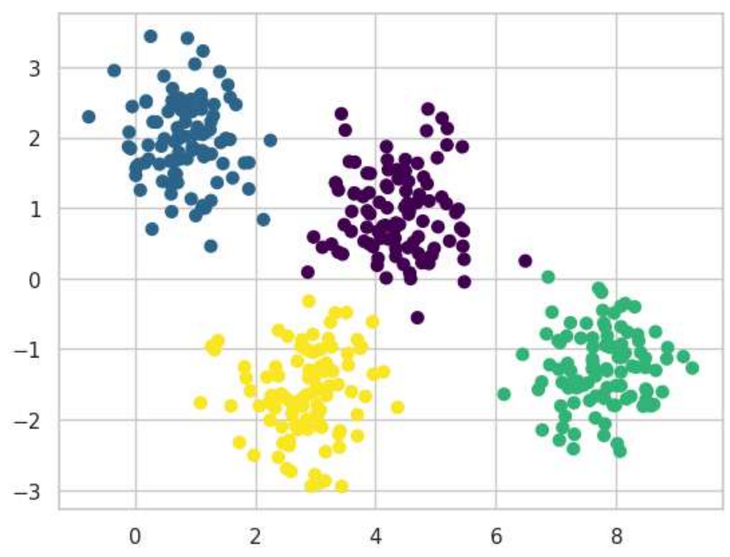


```
#GMM
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns # Import Seaborn
# Set Seaborn style which automatically sets the matplotlib style as well
sns.set_theme(style="whitegrid")
import numpy as np
```

```
# Generate some data
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=400, centers=4,
                       cluster_std=0.60, random_state=0)
X = X[:, ::-1] # flip axes for better plotting
```

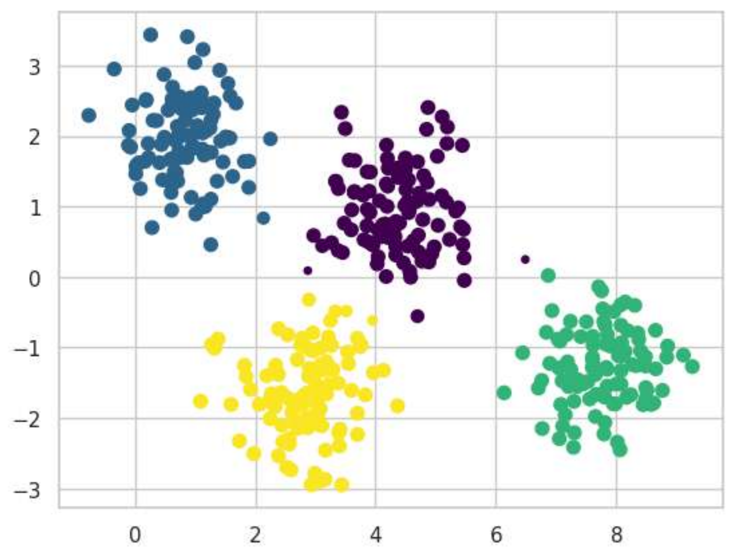
```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=4).fit(X)
labels = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis');
```



```
probs = gmm.predict_proba(X)
print(probs[:5].round(3))
```

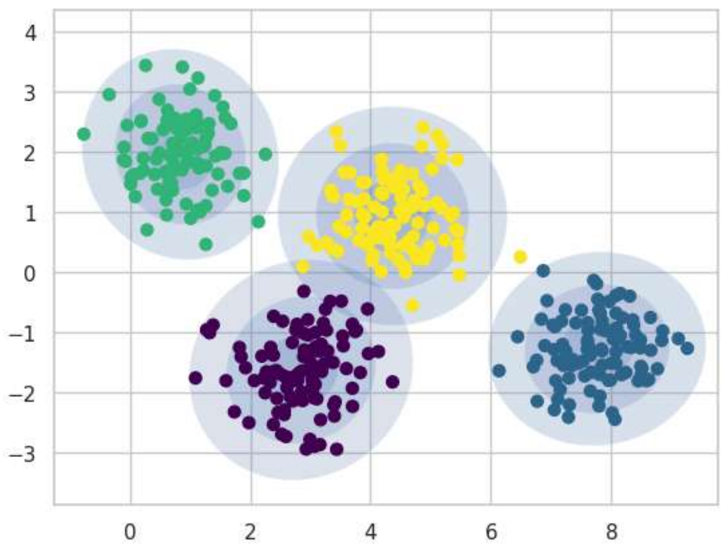
```
[[0.531 0.    0.469 0.   ]
 [0.    0.    0.    1.   ]
 [0.    0.    0.    1.   ]
 [1.    0.    0.    0.   ]
 [0.    0.    0.    1.   ]]
```

```
size = 50 * probs.max(1) ** 2  # square emphasizes differences
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=size);
```



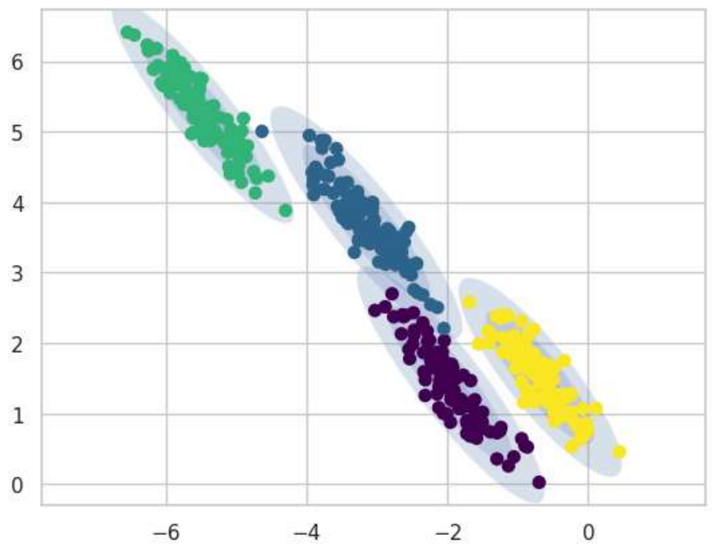
```
from matplotlib.patches import Ellipse

def draw_ellipse(position, covariance, ax=None, **kwargs):
    """Draw an ellipse with a given position and covariance"""
    ax = ax or plt.gca()

    # Convert covariance to principal axes
    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)

    # Draw the ellipse
    for nsig in range(1, 4):
        # Pass position as a single argument, unpacking its elements for the center
        ax.add_patch(Ellipse(xy=position, width=nsig * width, height=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=4, random_state=42)
plot_gmm(gmm, X)
```

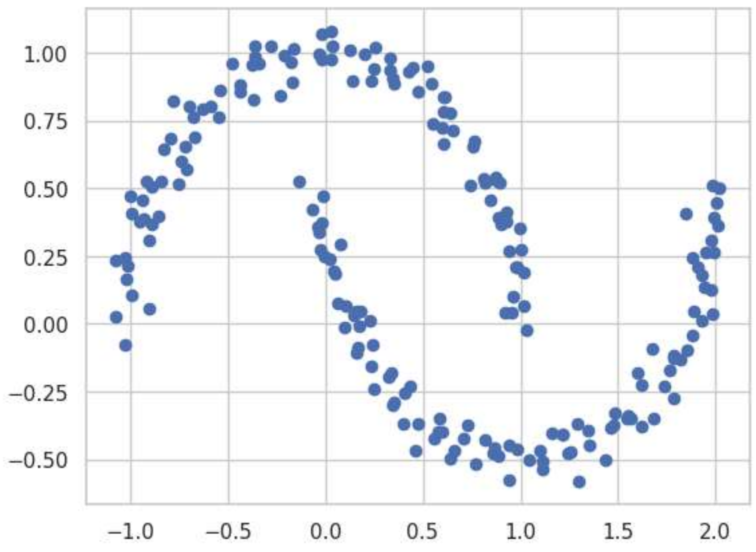


```
rng = np.random.RandomState(13)
X_stretched = np.dot(X, rng.randn(2, 2))
```

```
gmm = GaussianMixture(n_components=4, covariance_type='full', random_state=42)
plot_gmm(gmm, X_stretched)
```

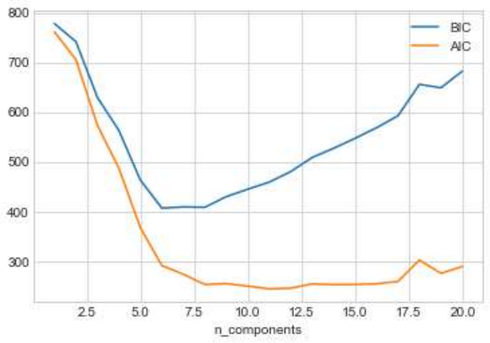


```
from sklearn.datasets import make_moons
Xmoon, ymoon = make_moons(200, noise=.05, random_state=0)
plt.scatter(Xmoon[:, 0], Xmoon[:, 1]);
```



A GMM is convenient as a flexible means of modeling an arbitrary multidimensional distribution of data.

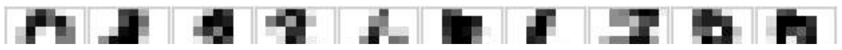
```
n_components = np.arange(1, 21)
models = [GaussianMixture(n, covariance_type='full', random_state=0).fit(Xmoon)
          for n in n_components]

plt.plot(n_components, [m.bic(Xmoon) for m in models], label='BIC')
plt.plot(n_components, [m.aic(Xmoon) for m in models], label='AIC')
plt.legend(loc='best')
plt.xlabel('n_components');
```



```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

```
(1797, 64)
```

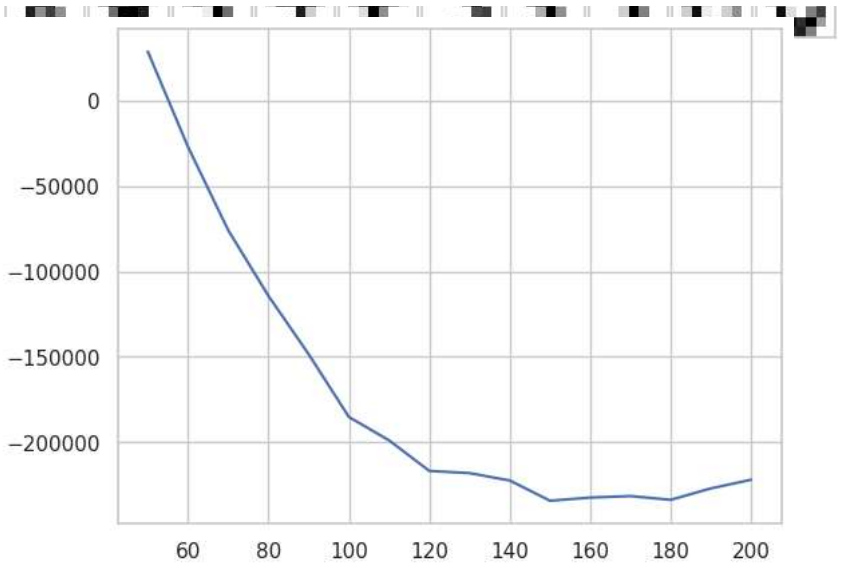
```
def plot_digits(data):
    fig, ax = plt.subplots(5, 10, figsize=(8, 4),
                           subplot_kw=dict(xticks=[], yticks=[]))
    fig.subplots_adjust(hspace=0.05, wspace=0.05)
    for i, axi in enumerate(ax.flat):
        im = axi.imshow(data[i].reshape(8, 8), cmap='binary')
        im.set_clim(0, 16)
plot_digits(digits.data)
```



```
from sklearn.decomposition import PCA
pca = PCA(0.99, whiten=True)
data = pca.fit_transform(digits.data)
data.shape
```

```
(1797, 41)
```

```
n_components = np.arange(50, 210, 10)
models = [GaussianMixture(n, covariance_type='full', random_state=0)
          for n in n_components]
aics = [model.fit(data).aic(data) for model in models]
plt.plot(n_components, aics);
```



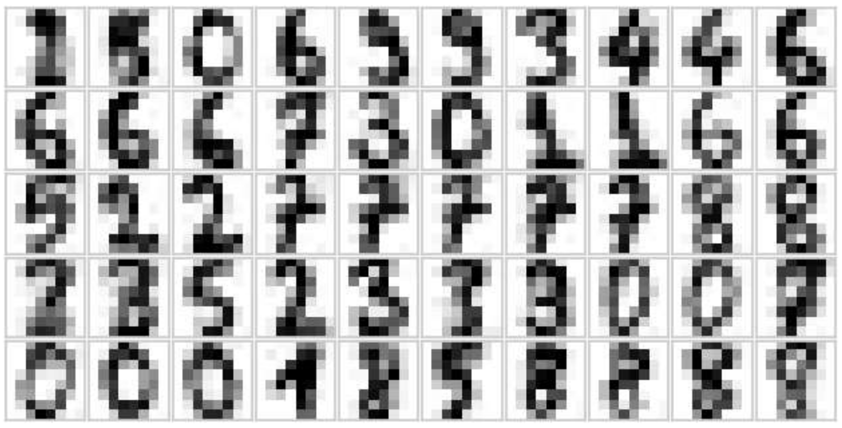
```
gmm = GaussianMixture(140, covariance_type='full', random_state=0)
gmm.fit(data)
print(gmm.converged_)
```

```
True
```

```
data_new, label_new = gmm.sample(100)
data_new.shape
```

```
(100, 41)
```

```
digits_new = pca.inverse_transform(data_new)
plot_digits(digits_new)
```



Start coding or generate with AI.