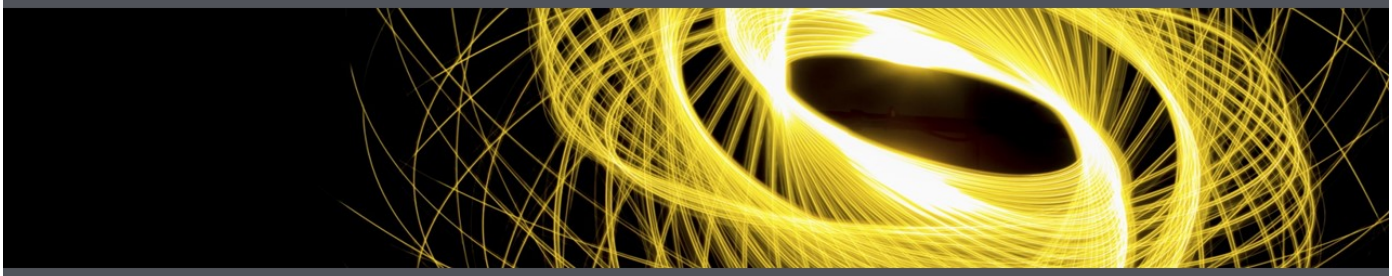
Department of Computer Science

University of Reading

CSMAD21 – Applied Data Science with Python



Clustering

University of Reading

Lecture Objectives

- Understand the characteristics and nature of K-means and Gaussian Mixture Models.
- Implement, evaluate and select the best model for clustering algorithms considering the nature of the dataset.

University of Reading

Outline

- Clustering algorithms
- K-Means
  - Definition
  - Expectation – Maximisation
  - Optimal Number of Clusters
- Gaussian Mixture Models
  - Definition
  - Expectation – Maximisation
  - Model Selection
- Q&A

University of Reading

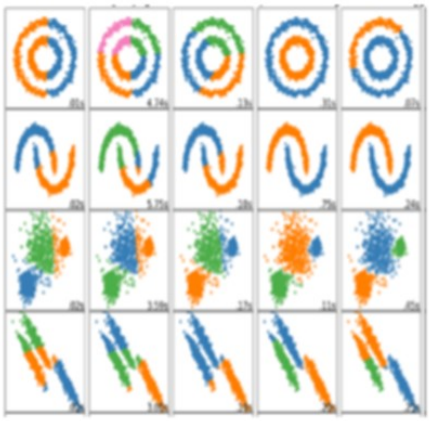
Clustering

- Clustering data can identify groups of similar data points within the data set.
  - Clustering is an unsupervised learning task, that is we typically do not have any 'correct' labelling of which data points belong in which cluster.
  - Today we will only consider quantitative data with continuous variables, but it is also possible to perform clustering in more complex data such as networks.

University of Reading

Clustering methods overview

- Various different clustering methods exist, some machine learning methods and some statistical approaches.
  - **k-means**
  - Agglomerative clustering
  - **Gaussian Mixture Models**
  - DBSCAN
  - Spectral clustering



- Different clustering tasks often require different clustering approaches.

University of Reading

K-means

- The k-means algorithm searches for a predetermined number of clusters within am unlabelled multidimensional dataset.
- Follows the simple conception of what the optimal cluster looks like:
  - The “cluster center” is the arithmetic mean of all the points belonging to the cluster.
  - Each point is closer to its own cluster than to other cluster centers.
- It is a very popular clustering algorithm given it’s simplicity and because follows basic logic providing solutions than can be inferred “by eye”.

University of Reading

K-means: Expectation – Maximisation

- The Expectation –Maximisation Approach (E-M) consist of the following procedure:
  - Guess some cluster centers
  - Repeat until covered:
    - E-Step: assign points to the nearest cluster center
    - M-Step: set the cluster center to the means.

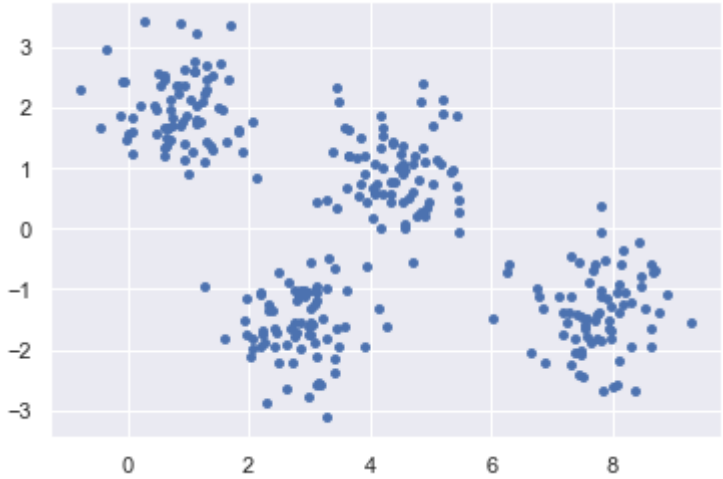
- The E-Step or Expectation step involves updating the expectation of which cluster each point belongs to.
- The M-Stem or Maximisation involves maximising some fitness function that defines the location of the cluster centers, in this case the mean of the data in each cluster.

In [3]:

```
## Importing libraries needed to create data and visualise it
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()  # for plot styling
import numpy as np
```

In [9]:

```
## Creating some datapoints to analyse
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.60, random_state=0)
X = X[:, ::-1] # flip axes for better plotting
plt.scatter(X[:, 0], X[:, 1], s=15);
```



We can identify 4 cluster without much analysis

In [10]:

```
##Import the library to implement K-means
from sklearn.cluster import KMeans

#Creating the deffinition of the model
kmeans = KMeans(n_clusters=4)

#Training the model to a particular dataset
kmeans.fit(X)

#Predicting the "labels" with the model defined and trained before
y_kmeans = kmeans.predict(X)
```

In [11]:

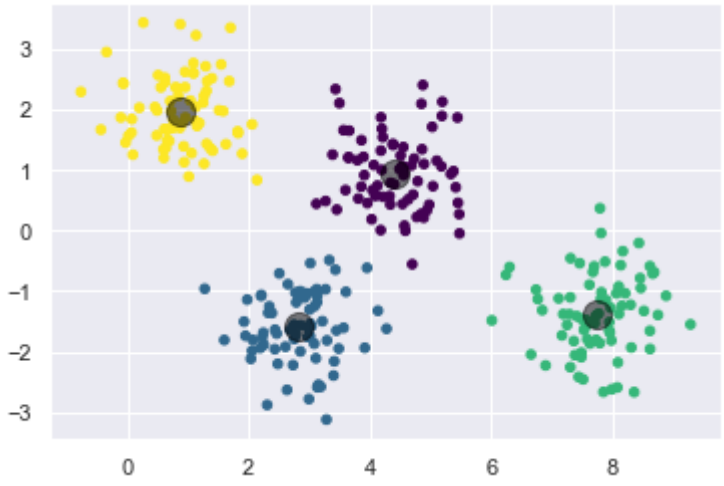
```
y_kmeans
```

Out[11]:

```
array([3, 2, 0, 2, 3, 3, 1, 0, 2, 2, 1, 2, 0, 2, 3, 0, 0, 3, 1, 1, 3, 3,
       0, 1, 1, 0, 3, 0, 1, 0, 2, 2, 0, 2, 2, 2, 2, 2, 1, 3, 0, 1, 0, 0,
       1, 1, 2, 1, 2, 3, 1, 3, 2, 3, 3, 1, 2, 1, 2, 3, 2, 0, 2, 1, 1, 1,
       2, 3, 2, 1, 0, 1, 2, 1, 1, 2, 1, 0, 3, 2, 3, 0, 3, 3, 2, 0, 3, 0,
       2, 2, 0, 3, 2, 1, 1, 0, 3, 3, 0, 1, 2, 3, 2, 3, 0, 3, 3, 0, 2, 0,
       1, 1, 3, 2, 3, 0, 2, 3, 3, 0, 1, 3, 1, 3, 3, 3, 3, 1, 3, 1, 2, 1,
       1, 3, 2, 1, 1, 2, 0, 2, 2, 1, 0, 1, 0, 1, 2, 0, 2, 2, 2, 0, 2, 0,
       3, 1, 2, 1, 3, 0, 2, 0, 0, 3, 0, 1, 1, 0, 3, 0, 0, 2, 3, 0, 1, 2,
       3, 3, 0, 1, 3, 0, 1, 1, 0, 0, 0, 0, 3, 2, 0, 1, 0, 0, 1, 1, 1, 0,
       1, 2, 0, 1, 3, 1, 0, 2, 1, 2, 0, 2, 0, 1, 0, 0, 2, 1, 1, 3, 3, 0,
       2, 3, 3, 1, 3, 1, 0, 2, 2, 0, 0, 2, 0, 3, 1, 0, 3, 1, 2, 1, 3, 0,
       3, 2, 2, 2, 2, 1, 1, 2, 0, 1, 3, 0, 1, 1, 1, 3, 3, 2, 0, 0, 1, 3,
       2, 1, 0, 2, 0, 3, 3, 1, 1, 0, 3, 3, 3, 0, 2, 2, 3, 3, 0, 3, 3, 3,
       2, 1, 2, 0, 3, 3, 2, 2, 2, 3, 3, 0, 2, 1])
```

In [12]:

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=20, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```

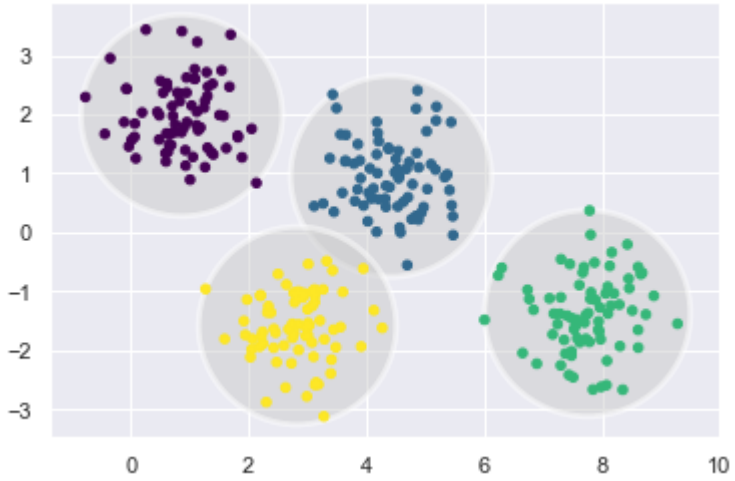


In [13]:

```
from scipy.spatial.distance import cdist

##Creating a method to visualise the centroids area of influence

def plot_kmeans(kmeans, X, n_clusters=4, rseed=0, ax=None):
    labels = kmeans.fit_predict(X)

    # plot the input data
    ax = ax or plt.gca()
    ax.axis('equal')
    ax.scatter(X[:, 0], X[:, 1], c=labels, s=20, cmap='viridis', zorder=2)

    # plot the representation of the KMeans model
    centers = kmeans.cluster_centers_
    radii = [cdist(X[labels == i], [center]).max()
             for i, center in enumerate(centers)]
    for c, r in zip(centers, radii):
        ax.add_patch(plt.Circle(c, r, fc='#CCCCCC', lw=3, alpha=0.5, zorder=1))
```

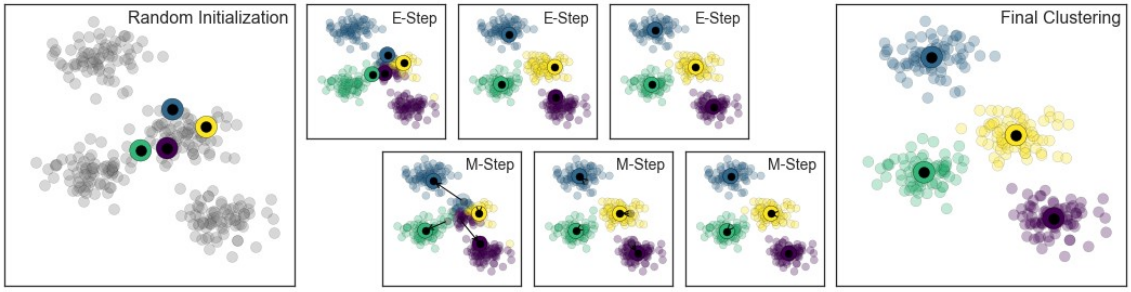
In [14]:

```
kmeans = KMeans(n_clusters=4, random_state=0)
plot_kmeans(kmeans, X)
```



University of Reading

K-means: Expectation – Maximisation



University of Reading

K-means: Expectation – Maximisation

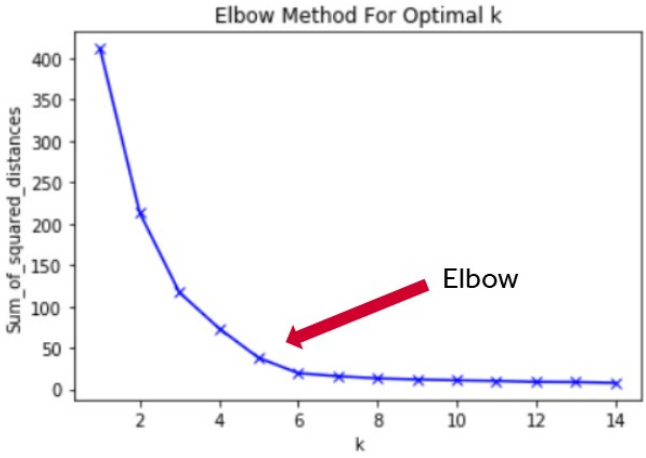
- Under typical circumstances, each repetition of the E-Step and M-Step will always result in a better estimate of the cluster characteristics.

- **Limitations:**
  - The global optimal result may not be achieved (randomness of the initial definition).
  - The number of clusters must be selected beforehand.
  - Is limited to linear cluster boundaries (not good clustering data with complex geometries).

University of Reading

K-means: Optimal Number of Clusters

- **Elbow method**
  - The objective is to analyse the behaviour of the “Inertia”(Mean of Sum of Squared Distances) value of the Cluster.
  - As k increases, the sum of squared distance tends to zero.



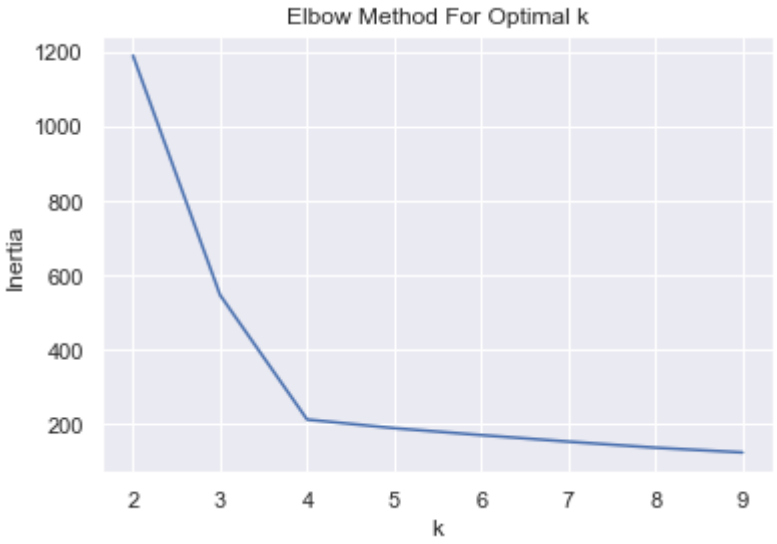
Elbow method

In [15]:

```
## List to store the metric value given different K values
inertia = []
#Range of the different values of K to analyse
K = range(2,10)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(X)
    inertia.append(km.inertia_)
```

In [16]:

```
plt.plot(K, inertia)
plt.xlabel('k')
plt.ylabel('Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```



University of Reading

K-means: Optimal Number of Clusters

- **Silhouette Scores**
  - The Silhouette score is the mean silhouette over all the instances.
  - An *silhouette coefficient* is equal to *(b -a) / max(a,b)*
    - Where *a* is the mean distance to the other instances in the same cluster (*mean intracluster distance*). *b* is the mean nearest-cluster distance (the mean distance to the instances of the next closest cluster, defined as the one that minimizes b, excluding the instance's own cluster).
  - Vary between -1 and 1.
    - Closer to 1 means that the instance is well inside its own cluster and far to other clusters.
    - 0 Means that is close to a cluster boundary.
    - -1 means that the instance may have been assigned to the wrong cluster.

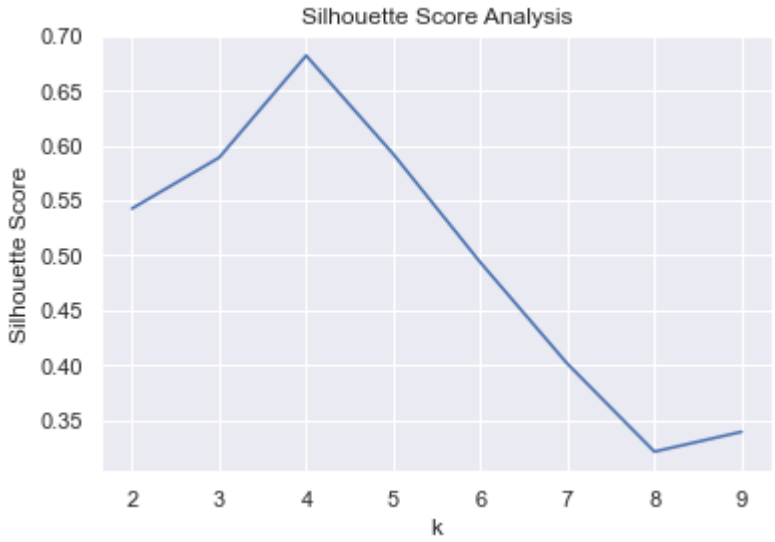
Silhouette Score

In [17]:

```
##Import the library to calculate the silhouette score
from sklearn.metrics import silhouette_score
## List to store the metric value given different K values
s_score = []
#Range of the different values of K to analyse
K = range(2,10)
for k in K:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    s_score.append(silhouette_score(X, kmeans.labels_))
```

In [18]:

```
##Plotting the values of the metric
plt.plot(K, s_score)
plt.xlabel('k')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score Analysis')
plt.show()
```



University of Reading

Gaussian Mixture Models

- Gaussian Mixture Models (GMM) are a way of representing data sets where the observations may have been sampled from a combination of multiple underlying probability distributions.
- In a mixture model the distribution of a variable is defined as a combination of multiple probability densities:

$$p(x) = w_1 p_1(x|\theta_1) + w_2 p_2(x|\theta_2) + \ldots + w_m p_m(x|\theta_m)$$

with $\sum_{i=1}^{m} w_i = 1$. Often in clustering the form of the distributions $p_1, p_2, \ldots, p_m$ is the same, but each has different parameters $\theta_1, \theta_2, \ldots, \theta_m$.

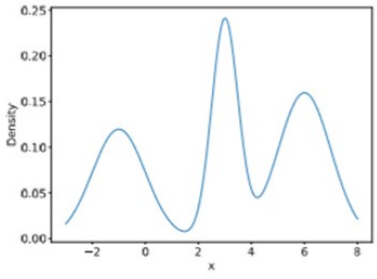
University of Reading

Gaussian Mixture Models

- In a GMM, each component i of the mixture follows a normal distribution with some parameters θi. In one dimension this would be:

$$p(x) = w_1 p_N(x|\mu_1, \sigma_1^2) + w_2 p_N(x|\mu_2, \sigma_2^2) + \ldots + w_m p_N(x|\mu_m, \sigma_m^2)$$

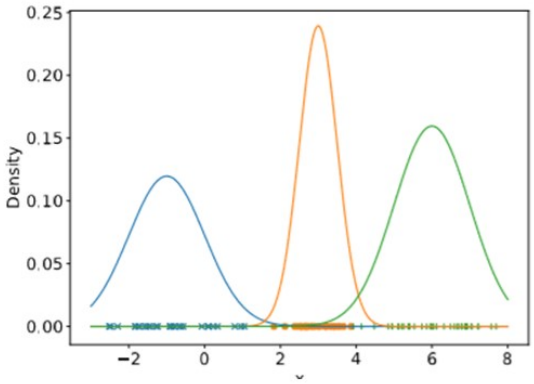
We can think of this as each data point being generated by a normal distribution with parameters μ_i, σ_i^2 with probability w_i.



University of Reading

Gaussian Mixture Models

- We can use these models to learn clustering of the data by fitting a mixture with a certain number of components to the observations. When we do this we can estimate the probability that each data point was generated by a particular component of the mixture.



University of Reading

Gaussian Mixture Models & Model Selection

- Using this approach we need to specify the number of mixture components in the model in advance. This means that if we do not know how many clusters to expect in the data we have to guess, or try multiple different numbers of clusters. Then we need a way to evaluate which best fits the data.

Model selection

One solution to this is to use something called *model selection*. This is where we compare multiple models to see which fits the data the best. Often this is done using the *likelihood* of the model:

$$\mathcal{L}(\theta|x) = p(x|\theta)$$

University of Reading

Model Selection

- Often, we will have multiple statistical models for a given data set, and would like to choose which is the best. This can occur when:
  - We want to see which probability distribution provides the best fit to the data.
  - We have multiple models for the process generating the data (e.g. are two groups of data points iid from the same distribution, or do they have two different parameter sets?)
- Using the Maximum Likelihood Estimate (MLE) of the model parameters, we can compare models using the maximum value of their likelihood:

$$\mathrm{MLE}(\theta) = \arg\max_{\theta} \mathcal{L}(\theta|x)$$

University of Reading

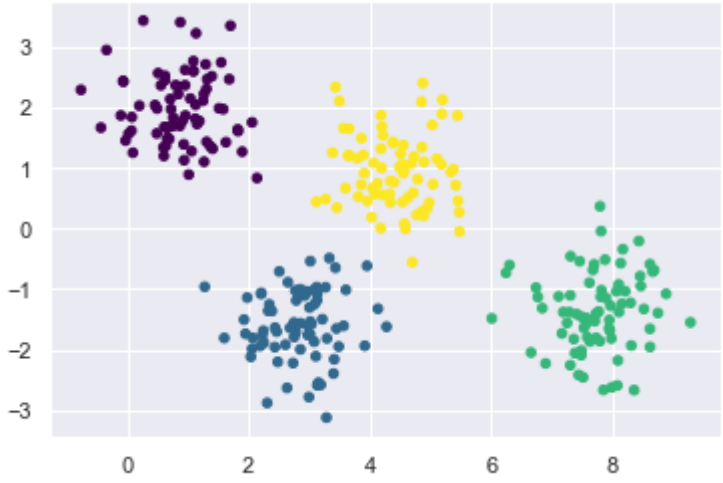
GMM: Expectation – Maximisation

- GMM is very similar to *k*-means: it uses an expectation–maximization approach which qualitatively does the following:
  - Choose starting guesses for the location and shape
  - Repeat until converged:
    - *E-Step*: for each point, find weights encoding the probability of membership in each cluster.
    - *M-Step*: for each cluster, update its location, normalization, and shape based on *all* data points, making use of the weights.
- The result of this is that each cluster is associated not with a hard-edged sphere, but with a smooth Gaussian model. Just as in the *k*-means expectation–maximization approach, this algorithm can sometimes miss the globally optimal solution, and thus in practice multiple random initializations are used.

Mixture Models

In [19]:

```
from sklearn.mixture import GaussianMixture as GMM

gmm = GMM(n_components=4).fit(X)
labels = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=20, cmap='viridis');
```



In [20]:

```
## Probability of each point
probs = gmm.predict_proba(X)
print(probs[:5].round(3))
```

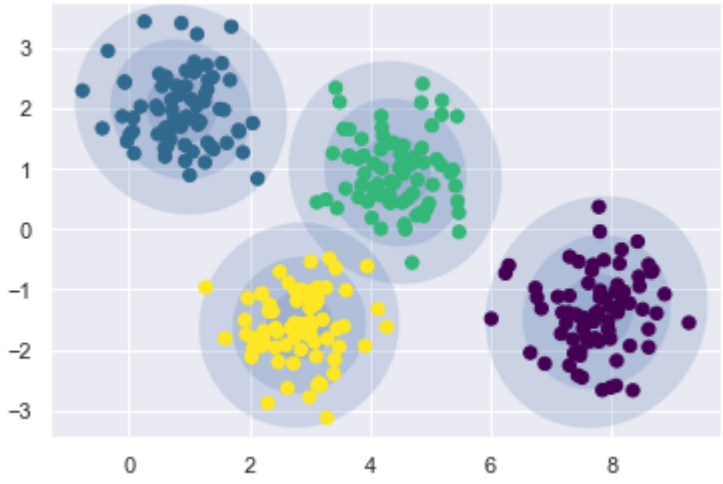
```
[[0.972 0.002 0.    0.026]
 [0.    0.    1.    0.   ]
 [0.    0.    0.    1.   ]
 [0.    0.    1.    0.   ]
 [0.999 0.    0.    0.001]]
```

In [21]:

```
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):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             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)
```

In [22]:

```
gmm = GMM(n_components=4, random_state=42)
plot_gmm(gmm, X)
```



University of Reading

Model Selection

- The Akaike Information Criterion (AIC) was created by statistician Hirotugu Akaike and gives a measure of the relative fit of several statistical models to data. It can be used to select which model best represents the data. A model with a smaller value of the AIC is considered better.
- The AIC is calculated as:

$$\mathrm{AIC} = 2k - 2\log(\mathcal{L}(\hat{\theta}|x))$$

where x is the observed data, k is the number of parameters in the model, $\hat{\theta}$ is the maximum likelihood estimate (MLE) of the parameters, and $\mathcal{L}$ is the likelihood of the model.

The addition of the number of parameters *penalises more complex models*.

University of Reading

Model Selection

- The Bayesian Information Criterion (BIC) is related to the AIC, but gives a stronger penalty based on the number of parameters.
- It was first published in a 1978 paper by Gideon E. Schwarz. The BIC is calculated as:

$$\mathrm{BIC} = k\log n - 2\log(\mathcal{L}(\hat{\theta}|x))$$

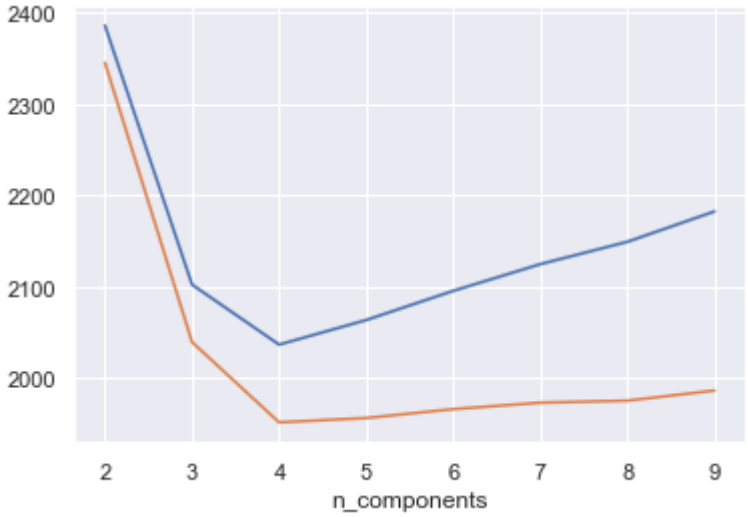
where k is the number of parameters in the model, n is the number of observations and $\hat{\theta}$ is the maximum likelihood estimate (MLE) of the parameters.

Again, a *smaller* value of the BIC is considered better.

AIC and BIC

In [23]:

```
n_components = range(2, 10)

models = [GMM(n, covariance_type='full', random_state=0).fit(X)
          for n in n_components]

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



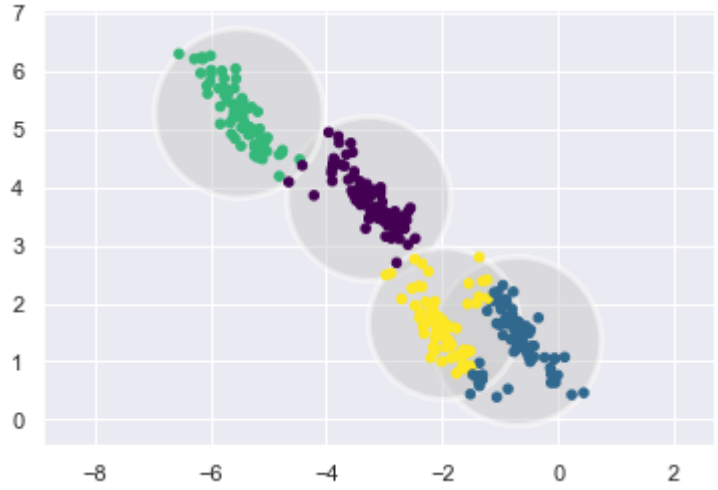
AIC vs BIC

- Both penalise models that have more parameters to learn (e.g. more clusters) and reward models that fit the data well. They often end up selecting the same model. When they differ, the model selected by BIC tends to be simpler (fewer parameters) than the one selected by the AIC, but tends to not fit the data quite as well (specially for larger datasets).

K-Means vs GMM

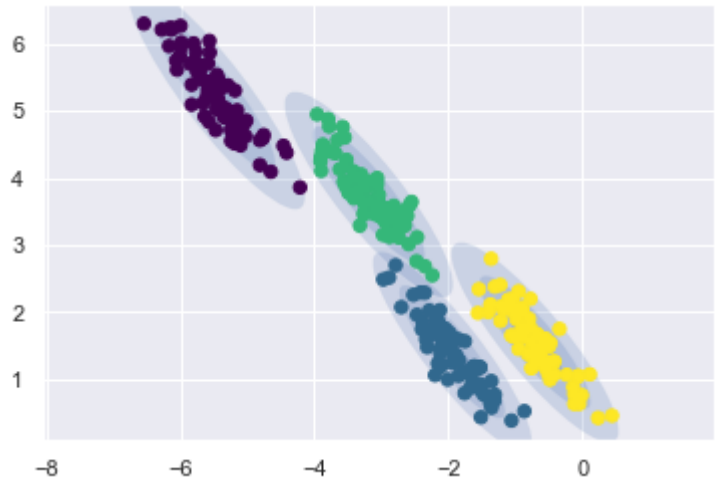
In [26]:

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

kmeans = KMeans(n_clusters=4, random_state=0)
plot_kmeans(kmeans, X_stretched)
```



In [27]:

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



- K-means is fast and scalable but it does not behave very well when the clusters have varying sizes, different densities, or nonspherical shapes. It is important to scale (normalise) the input features before K-

means; it generally improves things.

University of Reading

Summary

- Clustering algorithms are part of unsupervised machine learning. They identify groups of data point depending on their characteristic.
- Difference clustering tasks require different clustering approaches.
- K-Means it is known by its simplicity and consider the distances between the centroids and the data points to define clusters memberships. The Elbow Method and the Silhouette Scores are two approaches to define the optimal number of clusters.
- Gaussian Mixture Models (GMM) represents the data by combining multiple probabilistic distributions. AIC and BIC are two metrics that can be used to define the optimal number of clusters.

University of Reading

Questions

University of Reading

References

- VanderPlas, J. (2016). Python data science handbook: Essential tools for working with data. " O'Reilly Media, Inc.".
- Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.