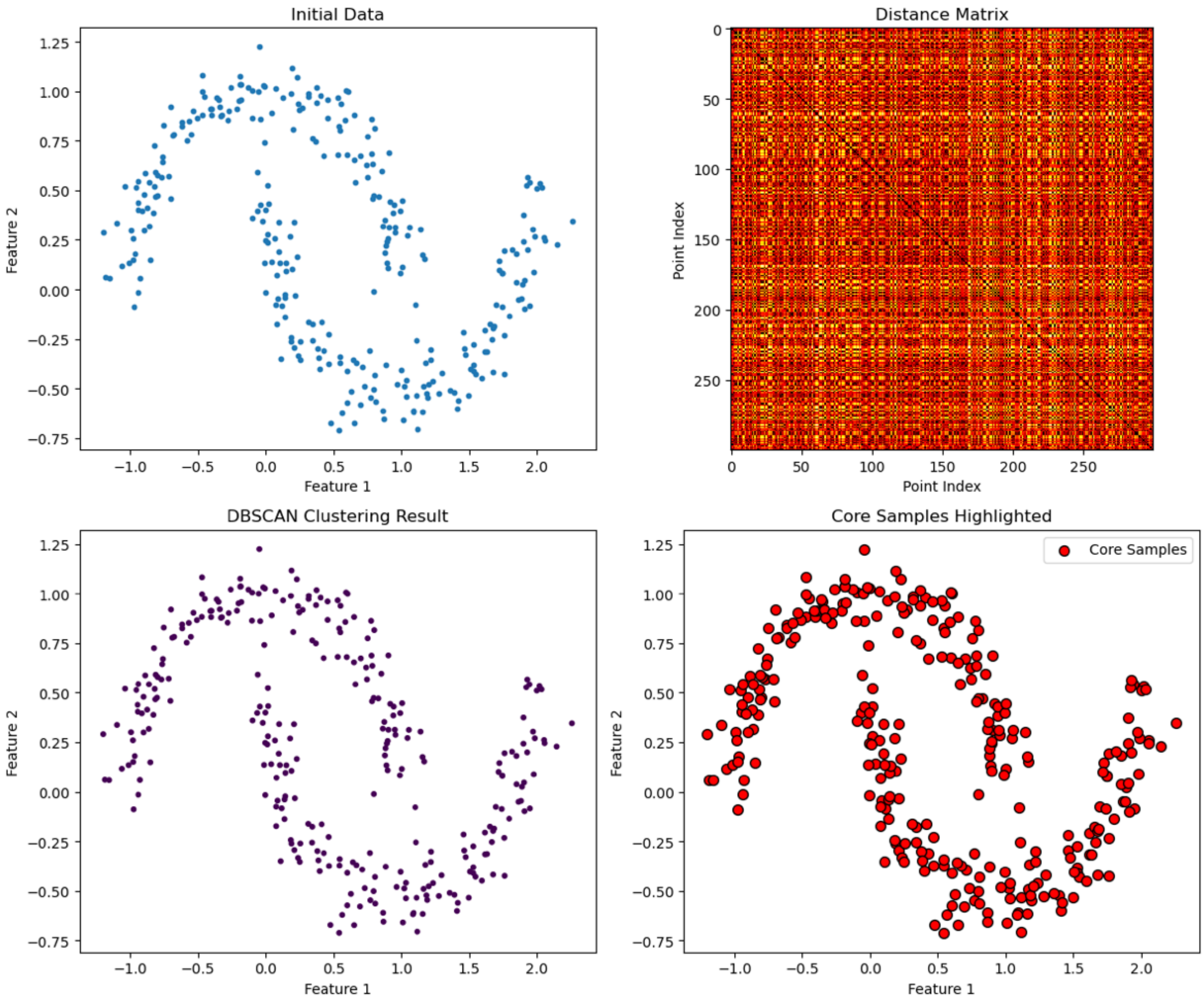
In [4]:
```
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from scipy.spatial.distance import pdist, squareform
```

In [6]:
```
# Generate sample data
X, y = make_moons(n_samples=300, noise=0.1, random_state=0)
# Function to plot data
def plot_data(X, title, ax):
    ax.scatter(X[:, 0], X[:, 1], s=10)
    ax.set_title(title)
    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
# Plot initial data
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
plot_data(X, 'Initial Data', axs[0, 0])

# Compute the distance matrix
dist_matrix = squareform(pdist(X))

# Plot distance matrix
axs[0, 1].imshow(dist_matrix, cmap='hot', interpolation='nearest')
axs[0, 1].set_title('Distance Matrix')
axs[0, 1].set_xlabel('Point Index')
axs[0, 1].set_ylabel('Point Index')

# Apply DBSCAN
eps = 0.2
min_samples = 2
dbscan = DBSCAN(eps=eps, min_samples=min_samples)
labels = dbscan.fit_predict(X)

# Plot clustering result
axs[1, 0].scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=10)
axs[1, 0].set_title('DBSCAN Clustering Result')
axs[1, 0].set_xlabel('Feature 1')
axs[1, 0].set_ylabel('Feature 2')

# Highlight core samples
core_samples_mask = np.zeros_like(labels, dtype=bool)
core_samples_mask[dbscan.core_sample_indices_] = True

axs[1, 1].scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=10)
axs[1, 1].scatter(X[core_samples_mask, 0], X[core_samples_mask, 1], c='red',
axs[1, 1].set_title('Core Samples Highlighted')
axs[1, 1].set_xlabel('Feature 1')
axs[1, 1].set_ylabel('Feature 2')
axs[1, 1].legend()
```

```python
plt.tight_layout()
plt.show()
```



In [5]:
```python
# Import libraries
import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

from scipy.integrate import odeint
from IPython.display import Image
from statistics import mode
from scipy.stats import pearsonr, spearmanr
from sklearn.feature_selection import mutual_info_regression
import statsmodels.api as sm
```

```python
# Suppress warnings
warnings.filterwarnings('ignore')

# Set number of decimals for np print options
np.set_printoptions(precision=3)

# Set the current working directory
os.chdir(sys.path[0])
```

In [8]:

```python
# load the dataset
df = pd.read_csv('moons_data.csv')
# display the first few rows of the dt
print(df)
```

```
           x1        x2  y
0    0.771744 -0.548086  1
1    0.189416 -0.261982  1
2    0.918359  0.443277  0
3    1.021213 -0.488523  1
4    1.178442 -0.369193  1
..        ...       ... ..
295  1.531153 -0.275921  1
296  0.184710 -0.243903  1
297 -0.703622  0.458507  0
298  0.592696  1.006177  0
299  0.104182  0.134738  1

[300 rows x 3 columns]
```

In [9]:

```python
# Extract features and labels
X1= df.iloc[:,0]
X2= df.iloc[:,1]
X = df.iloc[:, :2]
y_true = df.iloc[:,-1]
```

In [10]:

```python
X1
```

Out[10]:

```
0      0.771744
1      0.189416
2      0.918359
3      1.021213
4      1.178442
         ...
295    1.531153
296    0.184710
297   -0.703622
298    0.592696
299    0.104182
Name: x1, Length: 300, dtype: float64
```

In [11]:
```
X2
```

Out[11]:
```
0     -0.548086
1     -0.261982
2      0.443277
3     -0.488523
4     -0.369193
         ...
295   -0.275921
296   -0.243903
297    0.458507
298    1.006177
299    0.134738
Name: x2, Length: 300, dtype: float64
```

In [12]:
```
y_true
```

Out[12]:
```
0      1
1      1
2      0
3      1
4      1
      ..
295    1
296    1
297    0
298    0
299    1
Name: y, Length: 300, dtype: int64
```

In [13]:
```
X
```

Out[13]:

	x1	x2
0	0.771744	-0.548086
1	0.189416	-0.261982
2	0.918359	0.443277
3	1.021213	-0.488523
4	1.178442	-0.369193
...	...	...
295	1.531153	-0.275921
296	0.184710	-0.243903
297	-0.703622	0.458507
298	0.592696	1.006177
299	0.104182	0.134738

300 rows × 2 columns

In [14]:
```
# Perform agglomerative clustering
from sklearn.cluster import AgglomerativeClustering
```

In [15]:
```
clustering = AgglomerativeClustering(n_clusters = 7).fit(X)
```

In [16]:
```
#print labels
clustering.labels_
```

Out[16]:
```
array([0, 1, 6, 0, 0, 5, 0, 0, 4, 2, 3, 6, 1, 0, 6, 2, 3, 2, 0, 1, 1, 2,
       5, 5, 0, 5, 1, 3, 0, 3, 2, 4, 6, 5, 0, 1, 1, 1, 3, 6, 0, 2, 2, 4,
       4, 5, 0, 4, 6, 4, 2, 0, 1, 2, 4, 3, 1, 6, 3, 1, 5, 5, 4, 3, 3, 0,
       3, 5, 6, 4, 1, 0, 3, 6, 5, 0, 4, 2, 1, 5, 5, 1, 3, 0, 0, 2, 0, 0,
       0, 0, 3, 4, 3, 1, 1, 1, 2, 4, 1, 6, 3, 0, 5, 1, 4, 0, 0, 0, 3, 0,
       1, 1, 4, 2, 0, 4, 1, 3, 5, 6, 4, 5, 6, 4, 2, 2, 1, 1, 1, 6, 2, 1,
       4, 2, 5, 0, 3, 3, 4, 3, 6, 2, 0, 6, 1, 5, 6, 4, 2, 3, 0, 6, 0, 0,
       3, 2, 4, 6, 0, 2, 1, 2, 6, 1, 4, 4, 6, 1, 4, 3, 0, 5, 0, 4, 1, 0,
       5, 0, 0, 4, 2, 5, 3, 0, 5, 2, 0, 2, 0, 2, 0, 1, 4, 1, 5, 0, 2, 3,
       2, 3, 0, 0, 4, 4, 4, 2, 3, 5, 0, 6, 0, 5, 5, 2, 1, 2, 0, 5, 3, 3,
       1, 6, 2, 1, 2, 1, 5, 5, 3, 3, 5, 0, 2, 2, 3, 6, 5, 2, 4, 4, 1, 0,
       4, 6, 5, 3, 1, 4, 2, 5, 2, 6, 3, 0, 4, 5, 4, 0, 5, 4, 1, 1, 1, 0,
       0, 6, 4, 5, 1, 3, 3, 5, 3, 1, 2, 0, 0, 5, 3, 4, 6, 1, 3, 1, 3, 4,
       1, 2, 2, 2, 0, 3, 0, 6, 1, 0, 1, 5, 4, 1])
```

In [17]:
```
# Compare true labels with the ones from clustering
y_pred = clustering.labels_
```

```
result = pd.DataFrame(np.transpose(np.vstack((y_true, y_pred))), columns= ['
result.iloc[:20,:]
```

Out[17]:

	y_true	y_label
0	1	0
1	1	1
2	0	6
3	1	0
4	1	0
5	0	5
6	1	0
7	1	0
8	0	4
9	0	2
10	1	3
11	0	6
12	1	1
13	1	0
14	0	6
15	0	2
16	1	3
17	0	2
18	1	0
19	1	1

In [18]:
```
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from scipy.spatial.distance import pdist, squareform
```

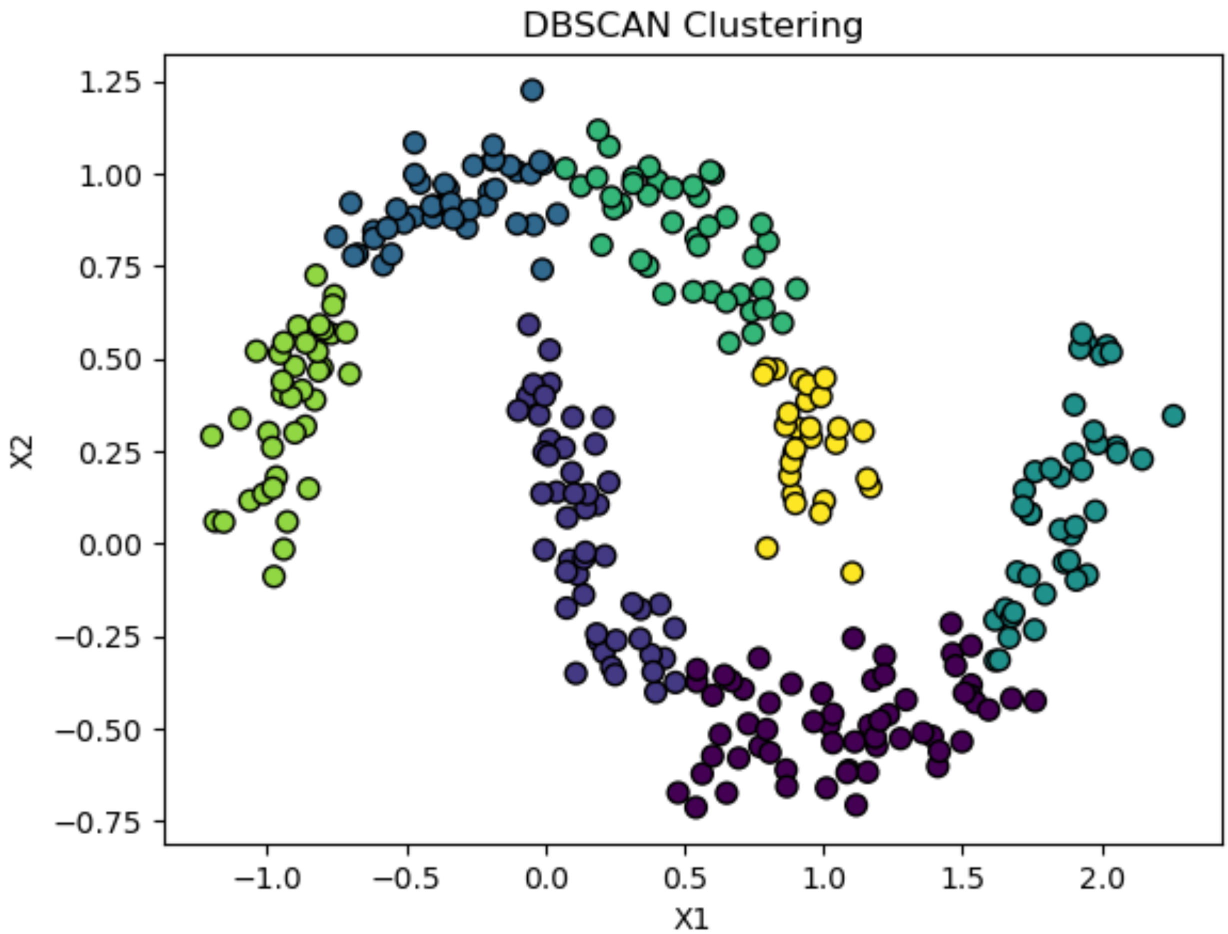
In [19]:
```
dbscan = DBSCAN(eps=0.3, min_samples=5)
cluster = dbscan.fit_predict(X)
```

In [20]:
```
# Add the predicted cluster labels to the DataFrame for comparison
```

```python
result_dbscan = pd.DataFrame({
    'X1': X1,
    'X2': X2,
    'y_true': y_true,
    'y_label_dbscan': y_pred
})
```

In [21]:
```python
# Show the first 20 rows of the results
print(result_dbscan.iloc[:20, :])

# Visualize the clustering results
plt.scatter(X1, X2, c=y_pred, cmap='viridis', edgecolor = 'k', s = 50)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('DBSCAN Clustering')
plt.show()
```

```
          X1        X2  y_true  y_label_dbscan
0   0.771744 -0.548086       1               0
1   0.189416 -0.261982       1               1
2   0.918359  0.443277       0               6
3   1.021213 -0.488523       1               0
4   1.178442 -0.369193       1               0
5  -1.062691  0.116202       0               5
6   1.234508 -0.461589       1               0
7   0.711839 -0.393012       1               0
8   0.542489  0.824464       0               4
9  -0.752471  0.828908       0               2
10  1.901708  0.242421       1               3
11  0.904850  0.312769       0               6
12  0.210737 -0.294526       1               1
13  1.411097 -0.602078       1               0
14  0.956856  0.287197       0               6
15 -0.099896  1.005161       0               2
16  1.744860  0.080245       1               3
17 -0.450309  0.973776       0               2
18  1.090810 -0.612312       1               0
19  0.095579  0.191930       1               1
```



In [22]:
```python
# Import KMeans from sklearn
from sklearn.cluster import KMeans
```

In [23]:
```python
kmeans = KMeans(n_clusters=7, random_state=42)
```

In [24]:
```python
y_pred_kmeans = kmeans.fit_predict(X)
```

In [25]:
```python
print(y_pred_kmeans)
```

```
[5 4 1 5 5 6 5 5 3 0 2 1 4 5 1 0 2 0 5 4 4 0 6 6 5 6 4 2 5 2 0 3 1 6 5 0 4
 4 2 1 5 0 0 1 3 6 5 3 1 3 0 5 4 0 3 2 4 1 2 4 6 6 3 2 2 5 2 6 1 3 4 5 2 1
 6 2 1 0 4 6 6 4 2 5 5 0 5 5 2 5 2 3 2 4 4 4 0 3 4 1 2 2 6 4 3 5 5 5 2 5 4
 4 3 0 5 3 4 2 6 1 1 6 1 3 0 0 4 4 4 1 0 4 3 0 6 5 2 2 3 2 1 0 5 1 4 6 1 3
 0 2 5 1 2 5 2 0 3 1 5 0 4 0 1 4 3 3 1 4 3 2 5 6 5 3 4 5 6 5 2 3 0 6 2 5 6
 0 5 0 5 0 5 4 1 4 6 5 0 2 0 2 2 5 3 3 3 0 2 6 5 1 5 6 6 0 4 0 5 6 2 2 4 1
 0 4 0 4 6 6 2 2 6 2 0 0 2 1 6 0 1 3 4 5 3 5 6 2 4 1 0 6 0 1 2 5 3 6 3 2 6
 1 4 4 0 2 5 1 3 6 4 2 2 6 2 4 0 5 2 6 2 3 1 4 2 4 2 3 4 0 0 0 5 2 5 1 4 2
 4 6 3 4]
```

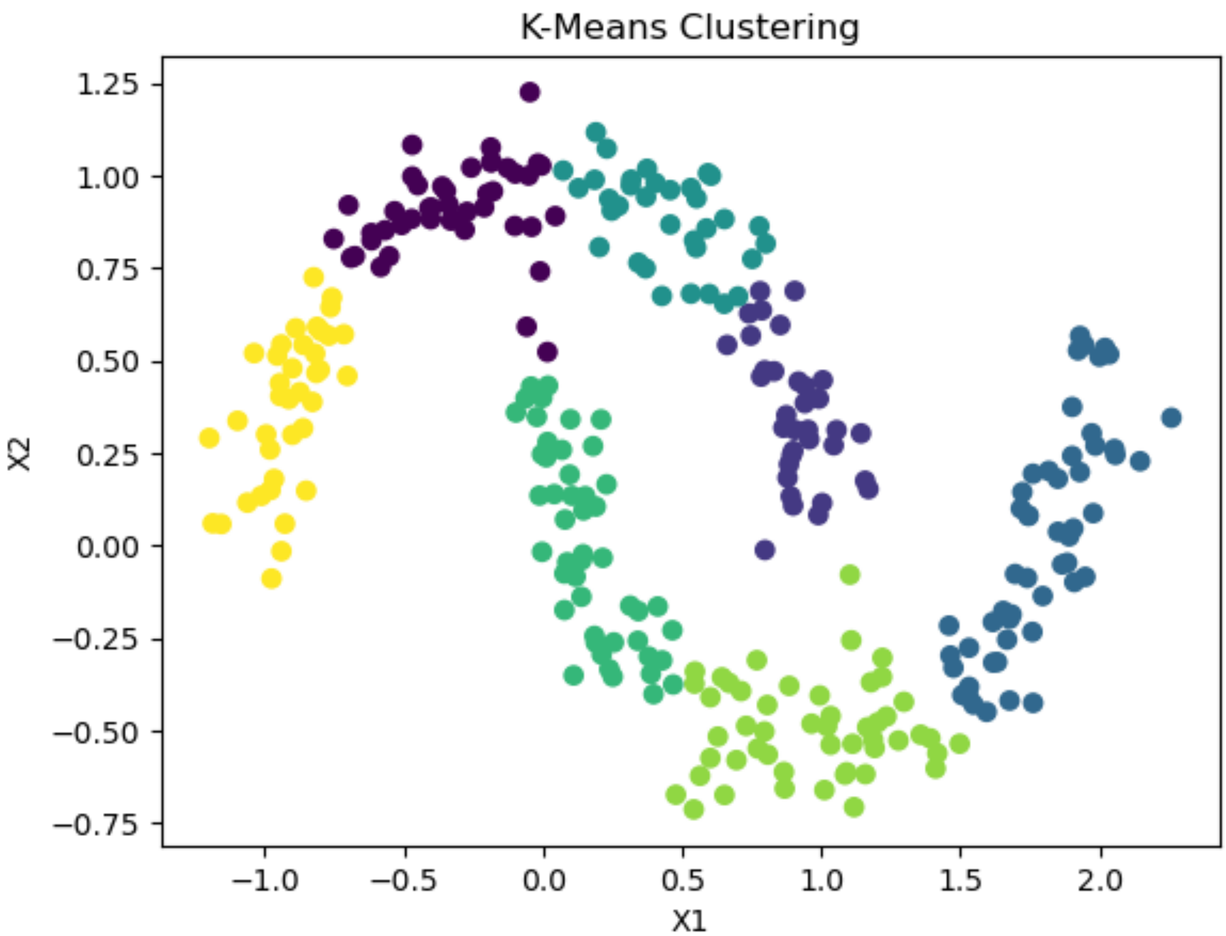
In [26]:
```python
# Add the predicted cluster labels to the DataFrame for comparison
result_kmeans = pd.DataFrame({
    'X1': X1,
```

```
    'X2': X2,
    'y_true': y_true,
    'y_label_kmeans': y_pred_kmeans
})
```

In [27]:
```
# Show the first 20 rows of the results
print(result_kmeans.iloc[:20, :])
```

```
          X1        X2  y_true  y_label_kmeans
0   0.771744 -0.548086       1               5
1   0.189416 -0.261982       1               4
2   0.918359  0.443277       0               1
3   1.021213 -0.488523       1               5
4   1.178442 -0.369193       1               5
5  -1.062691  0.116202       0               6
6   1.234508 -0.461589       1               5
7   0.711839 -0.393012       1               5
8   0.542489  0.824464       0               3
9  -0.752471  0.828908       0               0
10  1.901708  0.242421       1               2
11  0.904850  0.312769       0               1
12  0.210737 -0.294526       1               4
13  1.411097 -0.602078       1               5
14  0.956856  0.287197       0               1
15 -0.099896  1.005161       0               0
16  1.744860  0.080245       1               2
17 -0.450309  0.973776       0               0
18  1.090810 -0.612312       1               5
19  0.095579  0.191930       1               4
```

In [75]:
```
plt.scatter(X1, X2, c=y_pred_kmeans, cmap='viridis')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('K-Means Clustering')
plt.show()
```



Q1: How diffrent with DBScan and K-Means? Both are clustering algorithm. However, DBScan is a density-based algorithm which doesn't require the number of clusters and identifies clusters based on the density of points, and also excludes the outliers. In contrast, K-Means requires us to set the number of clusters (K) in advance, and is sensitive to outliers.

Q2:Why I choose n_clusters=7?

In [29]:
```
kmeans = KMeans(n_clusters=5, random_state=42)
y_pred_kmeans = kmeans.fit_predict(X)
print(y_pred_kmeans)
```

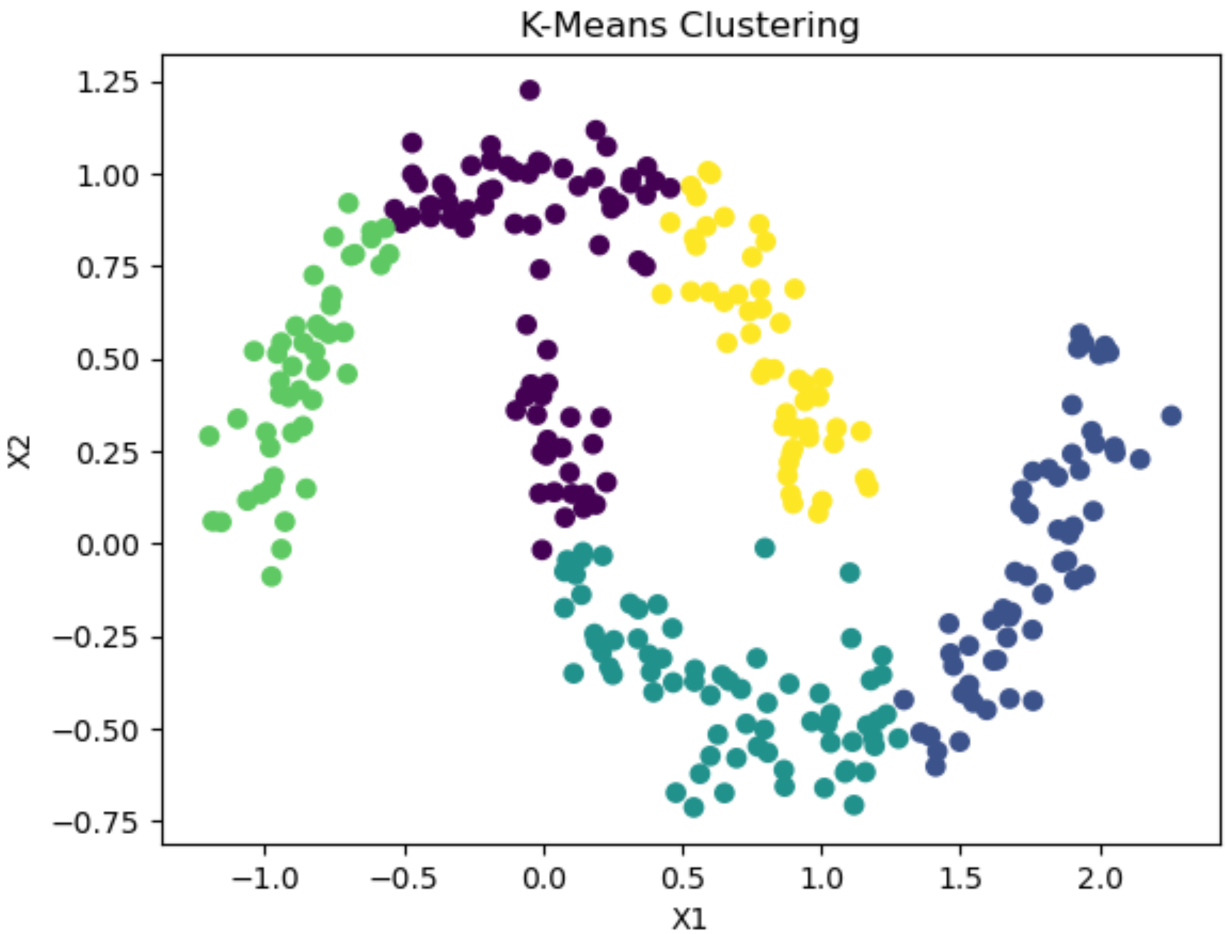
```
[2 2 4 2 2 3 2 2 4 3 1 4 2 1 4 0 1 0 2 0 2 0 3 3 2 3 2 1 2 1 0 0 4 3 2 0 2
 2 1 4 2 0 0 4 0 3 2 0 4 4 0 2 0 3 0 1 0 4 1 0 3 3 0 1 1 2 1 3 4 4 0 2 1 4
 3 1 4 0 2 3 3 0 1 2 2 0 2 2 1 2 1 4 1 2 0 0 0 0 2 4 1 1 3 0 0 2 1 2 1 2 0
 0 4 0 1 0 0 1 3 4 4 3 4 0 0 3 2 2 0 4 0 0 0 0 3 2 1 1 4 1 4 3 2 4 0 3 4 4
 0 1 2 4 1 1 1 0 0 2 2 0 2 0 4 2 4 4 4 0 4 1 2 3 2 0 2 2 3 2 1 4 3 3 1 2 3
 3 2 3 2 0 2 0 4 0 3 2 0 1 0 1 1 2 0 4 4 0 1 3 2 4 1 3 3 0 2 0 2 3 1 1 2 4
 0 2 0 2 3 3 1 1 3 1 0 0 1 4 3 3 4 0 0 2 0 2 3 1 2 4 3 3 0 4 1 2 4 3 0 1 3
 4 2 0 0 1 2 4 4 3 2 1 1 3 1 0 0 2 1 3 1 0 4 0 1 2 1 4 2 0 0 0 1 1 2 4 2 1
 2 3 4 0]
```

In [30]:
```
# Add the predicted cluster labels to the DataFrame for comparison
result_kmeans = pd.DataFrame({
    'X1': X1,
    'X2': X2,
    'y_true': y_true,
    'y_label_kmeans': y_pred_kmeans
})
```

In [31]:
```
# Show the first 20 rows of the results
print(result_kmeans.iloc[:20, :])
```

```
          X1        X2  y_true  y_label_kmeans
0   0.771744 -0.548086       1               2
1   0.189416 -0.261982       1               2
2   0.918359  0.443277       0               4
3   1.021213 -0.488523       1               2
4   1.178442 -0.369193       1               2
5  -1.062691  0.116202       0               3
6   1.234508 -0.461589       1               2
7   0.711839 -0.393012       1               2
8   0.542489  0.824464       0               4
9  -0.752471  0.828908       0               3
10  1.901708  0.242421       1               1
11  0.904850  0.312769       0               4
12  0.210737 -0.294526       1               2
13  1.411097 -0.602078       1               1
14  0.956856  0.287197       0               4
15 -0.099896  1.005161       0               0
16  1.744860  0.080245       1               1
17 -0.450309  0.973776       0               0
18  1.090810 -0.612312       1               2
19  0.095579  0.191930       1               0
```

In [32]:
```
plt.scatter(X1, X2, c=y_pred_kmeans, cmap='viridis')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('K-Means Clustering')
plt.show()
```



In [33]:
```
kmeans = KMeans(n_clusters=3, random_state=42)
y_pred_kmeans = kmeans.fit_predict(X)
print(y_pred_kmeans)
```

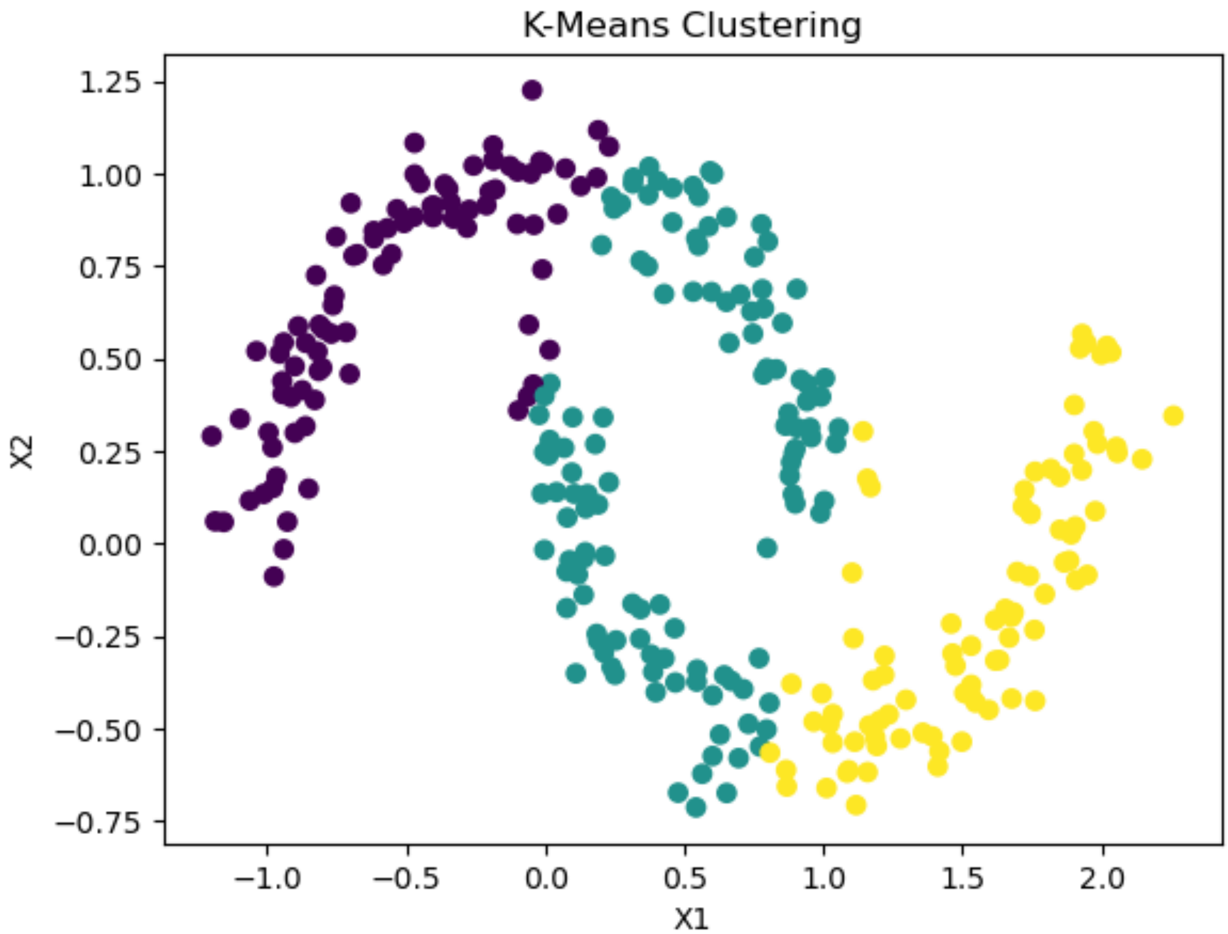
```
[1 1 1 2 2 0 2 1 1 0 2 1 1 2 1 0 2 0 2 1 1 0 0 0 2 0 1 2 2 2 0 1 1 0 1 0 1
 1 2 1 2 0 0 1 1 0 2 1 1 1 0 1 1 0 0 2 1 1 2 0 0 0 0 2 2 2 2 0 2 1 1 2 2 1
 0 2 1 0 1 0 0 1 2 2 2 0 1 2 2 2 2 1 2 1 0 1 0 1 1 1 2 2 0 1 0 2 2 1 2 1 1
 1 1 0 2 1 1 2 0 1 1 0 1 1 0 0 1 1 1 2 0 1 1 0 0 1 2 2 1 2 1 0 2 1 1 0 1 1
 0 2 1 1 2 2 2 0 1 1 2 0 1 0 1 1 1 1 1 0 1 2 2 0 1 1 1 2 0 1 2 1 0 0 2 1 0
 0 2 0 2 0 1 1 1 1 0 2 0 2 0 2 2 2 1 1 1 0 2 0 1 1 2 0 0 0 1 0 1 0 2 2 1 1
 0 1 0 1 0 0 2 2 0 2 0 0 2 1 0 0 1 1 1 2 0 2 0 2 1 1 0 0 0 1 2 2 1 0 1 2 0
 1 1 1 0 2 1 1 1 0 1 2 2 0 2 1 0 1 2 0 2 0 1 1 2 1 2 1 1 0 0 0 2 2 1 2 1 2
 1 0 1 1]
```

In [34]:
```
# Add the predicted cluster labels to the DataFrame for comparison
result_kmeans = pd.DataFrame({
    'X1': X1,
    'X2': X2,
    'y_true': y_true,
    'y_label_kmeans': y_pred_kmeans
})
# Show the first 20 rows of the results
print(result_kmeans.iloc[:20, :])
```

```
           X1        X2  y_true  y_label_kmeans
0    0.771744 -0.548086       1               1
1    0.189416 -0.261982       1               1
2    0.918359  0.443277       0               1
3    1.021213 -0.488523       1               2
4    1.178442 -0.369193       1               2
5   -1.062691  0.116202       0               0
6    1.234508 -0.461589       1               2
7    0.711839 -0.393012       1               1
8    0.542489  0.824464       0               1
9   -0.752471  0.828908       0               0
10   1.901708  0.242421       1               2
11   0.904850  0.312769       0               1
12   0.210737 -0.294526       1               1
13   1.411097 -0.602078       1               2
14   0.956856  0.287197       0               1
15  -0.099896  1.005161       0               0
16   1.744860  0.080245       1               2
17  -0.450309  0.973776       0               0
18   1.090810 -0.612312       1               2
19   0.095579  0.191930       1               1
```

In [35]:

```
plt.scatter(X1, X2, c=y_pred_kmeans, cmap='viridis')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('K-Means Clustering')
plt.show()
```



In [39]:
```
kmeans = KMeans(n_clusters=12, random_state=42)
y_pred_kmeans = kmeans.fit_predict(X)
print(y_pred_kmeans)
```

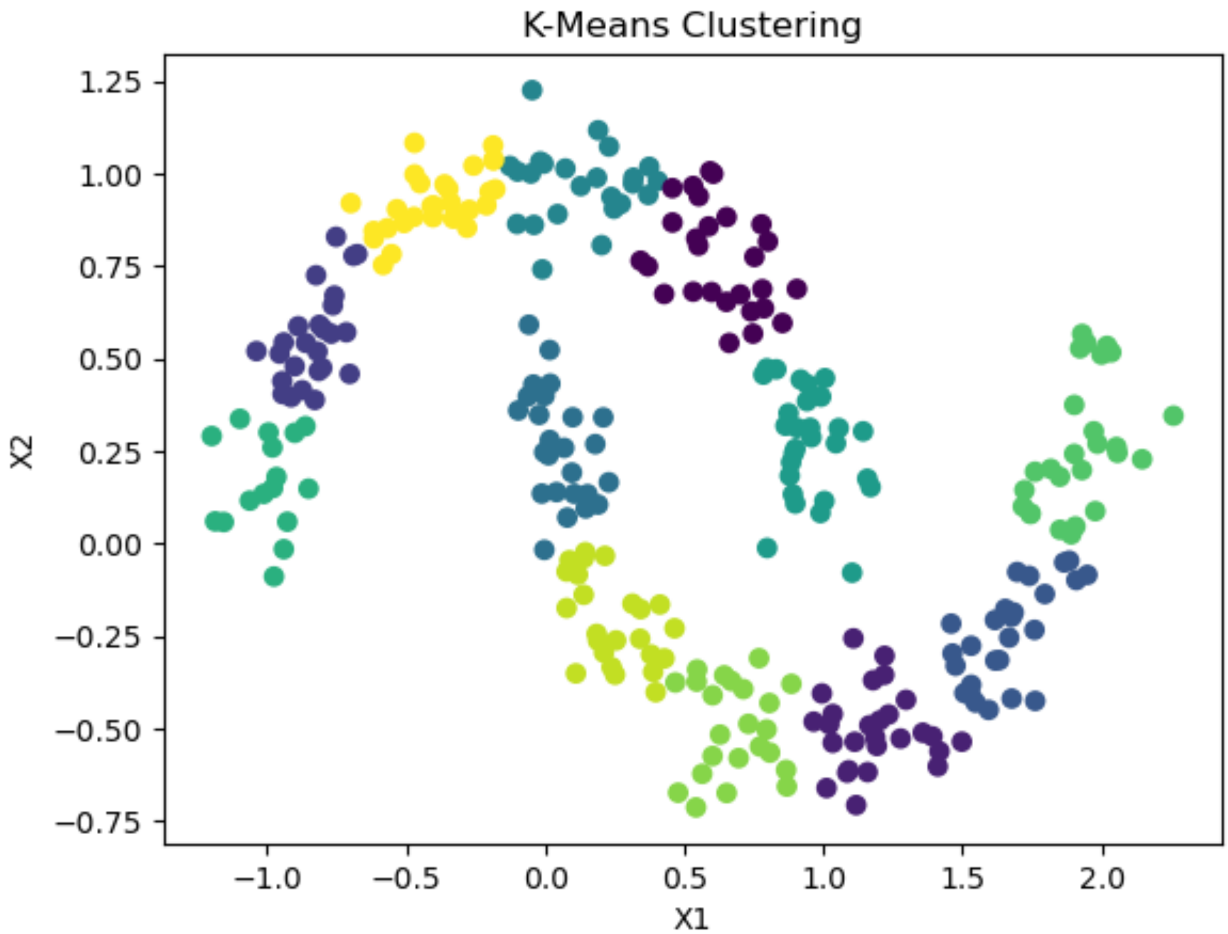
```
[ 9 10  6  1  1  7  1  9  0  2  8  6 10  1  6  5  8 11  1  4 10 11  2  2
  1  2 10  8  1  8 11  5  6  2  9  4 10 10  8  6  1 11 11  0  0  7  1  5
  6  0  5  9  4  2  5  3  4  6  8  4  7  7  5  8  8  9  3  2  6  0  4  1
  8  6  7  3  0 11 10  7  2  4  3  9  1  5  9  1  3  1  8  0  3 10  4  4
 11  0 10  6  8  3  2  4  5  1  1  9  8  9  4  4  0 11  1  5  4  8  2  6
  0  7  6  0 11 11 10 10  4  6 11  4  5  5  7  9  3  3  0  3  6 11  9  6
  4  2  6  0  5  8  9  6  3  1  8 11  5  6  1 11 10  5  6 10  0  0  6  4
  0  8  1  2  9  5 10  9  2  9  3  0  2  2  8  9  7 11  1 11  1 11  9  4
  0  4  2  1 11  8 11  8  3  1  5  0  0 11  8  7  9  6  1  7  7 11 10 11
  9  2  3  8 10  6  5  9 11 10  2  7  3  8  7  3 11 11  8  6  2 11  0  5
  4  1  5  6  2  3 10  0 11  2  5  6  3  1  0  2  5  3  7  0 10  4  4  3
  9  6  0  2 10  8  3  7  3  4 11  9  3  2  3  5  6  4  8 10  8  0 10 11
  5  5  1  3  9  6 10  3 10  2  0  4]
```

In [40]:
```
# Add the predicted cluster labels to the DataFrame for comparison
result_kmeans = pd.DataFrame({
    'X1': X1,
    'X2': X2,
    'y_true': y_true,
```

```
    'y_label_kmeans': y_pred_kmeans
})
# Show the first 20 rows of the results
print(result_kmeans.iloc[:20, :])
```

```
          X1        X2  y_true  y_label_kmeans
0   0.771744 -0.548086       1               9
1   0.189416 -0.261982       1              10
2   0.918359  0.443277       0               6
3   1.021213 -0.488523       1               1
4   1.178442 -0.369193       1               1
5  -1.062691  0.116202       0               7
6   1.234508 -0.461589       1               1
7   0.711839 -0.393012       1               9
8   0.542489  0.824464       0               0
9  -0.752471  0.828908       0               2
10  1.901708  0.242421       1               8
11  0.904850  0.312769       0               6
12  0.210737 -0.294526       1              10
13  1.411097 -0.602078       1               1
14  0.956856  0.287197       0               6
15 -0.099896  1.005161       0               5
16  1.744860  0.080245       1               8
17 -0.450309  0.973776       0              11
18  1.090810 -0.612312       1               1
19  0.095579  0.191930       1               4
```

In [41]:
```
plt.scatter(X1, X2, c=y_pred_kmeans, cmap='viridis')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('K-Means Clustering')
plt.show()
```



As the result of my tests, we could know if n_clusters<7, the algorithm may underfit the data, grouping distinct clusters into a single cluster and doesn't refluct the truth and important detail of our data. If n_clusters>7, the algorithm may overfit the data by creating many small clusters. This can lead to unmeaningful clusters.