Explanation of the Code:

1. **Import Libraries**:
   - We import `pandas` for data manipulation, `numpy` for mathematical operations, `matplotlib` for visualization, and `StandardScaler` from `sklearn` for feature scaling.
   - The `load_iris()` function from `sklearn` loads the Iris dataset, which we use to test our implementation.

2. **Load and Preprocess the Data**:
   - We load the Iris dataset into `X` (features) and `y` (actual labels).
   - To ensure that all features contribute equally to the clustering process, we scale the data using `StandardScaler`. This standardizes the features to have a mean of 0 and a standard deviation of 1, making it easier for the K-Means algorithm to perform effectively.

3. **Define Euclidean Distance Function**:
   - The `euclidean_distance()` function calculates the distance between two points in space. We will use this to measure the proximity of each data point to the centroids during clustering.

4. **Implement the K-Means Algorithm**:
   - **Step 4a**: The centroids are initialized randomly by selecting `K` random data points from the dataset.
   - **Step 4b**: We enter the iterative process:
     - **Step 4b1**: Each data point is assigned to the nearest centroid based on the Euclidean distance.
     - **Step 4b2**: After assigning all points to clusters, we update each centroid by calculating the mean of the points assigned to that cluster.
     - **Step 4b3**: We check for convergence. If the centroids no longer change, the algorithm stops early.

5. **Apply K-Means to the Iris Dataset**:
   - We apply the `kmeans()` function to the scaled Iris data with `K=3` clusters (since the Iris dataset has 3 distinct species).

- The final centroids and labels for each data point are printed out. The centroids represent the central point of each cluster, while the labels indicate the assigned cluster for each data point.

6. **Visualize the Clustering**:

   - Since we are using a 4-dimensional dataset (Iris dataset has 4 features), we visualize the first two features (sepal length vs sepal width) for simplicity.
   - We use `matplotlib` to create a scatter plot where each point is colored according to its assigned cluster. The centroids are marked with red 'X' markers.

7. **Compare Clustering Results with True Labels**:

   - Finally, we compare the K-Means clustering results (`labels`) with the true class labels (`y`). This comparison helps assess the performance of the algorithm. While K-Means doesn't know the true labels, you can visually inspect or measure how closely the clustering matches the actual species.

Key Notes:

- **Initialization**: The random initialization of centroids can sometimes lead to suboptimal clustering, especially if the initial centroids are chosen poorly. This can be mitigated using methods like **K-Means++**, but that's not implemented here.
- **Convergence**: The algorithm stops when the centroids do not change significantly after an iteration (convergence). If there is no convergence within the maximum number of iterations (`max_iters`), the algorithm will stop anyway.
- **Visualization**: Since we used the first two features for plotting, the clusters may look more distinct. In a higher-dimensional space, K-Means works in the full feature space, and visualizations can be more complex. Dimensionality reduction techniques like **PCA** or **t-SNE** are often used for such cases.

By running this code, you should see how the K-Means algorithm clusters the Iris dataset into 3 groups, and the visual representation should give you an idea of how well the algorithm performed.