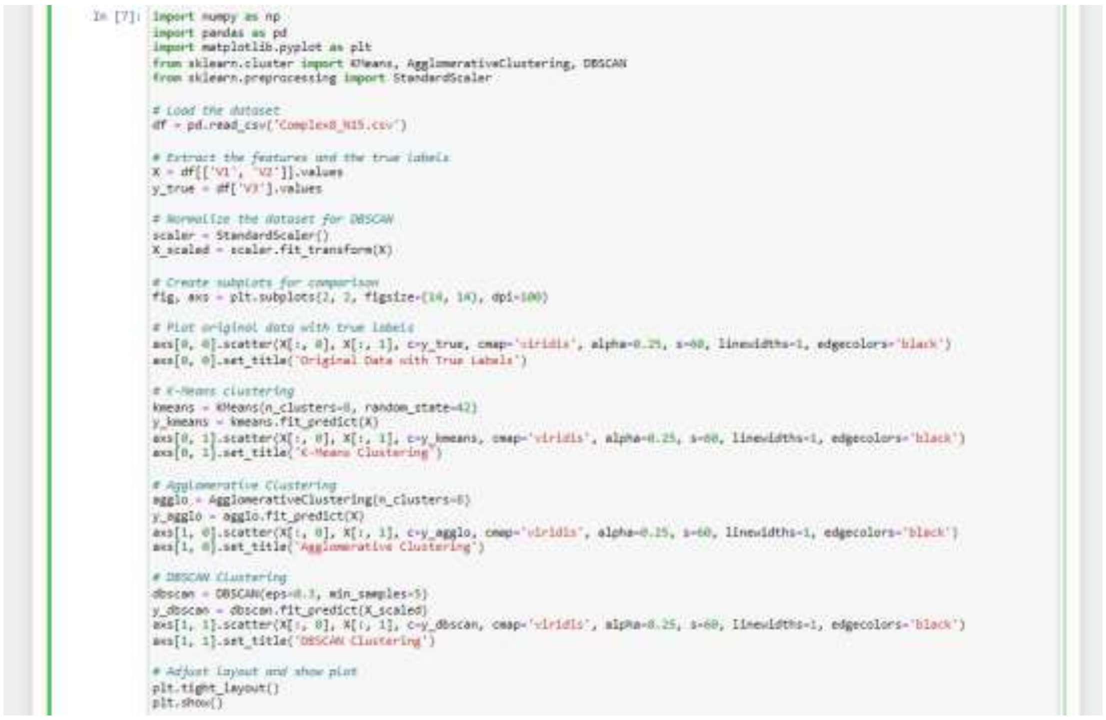
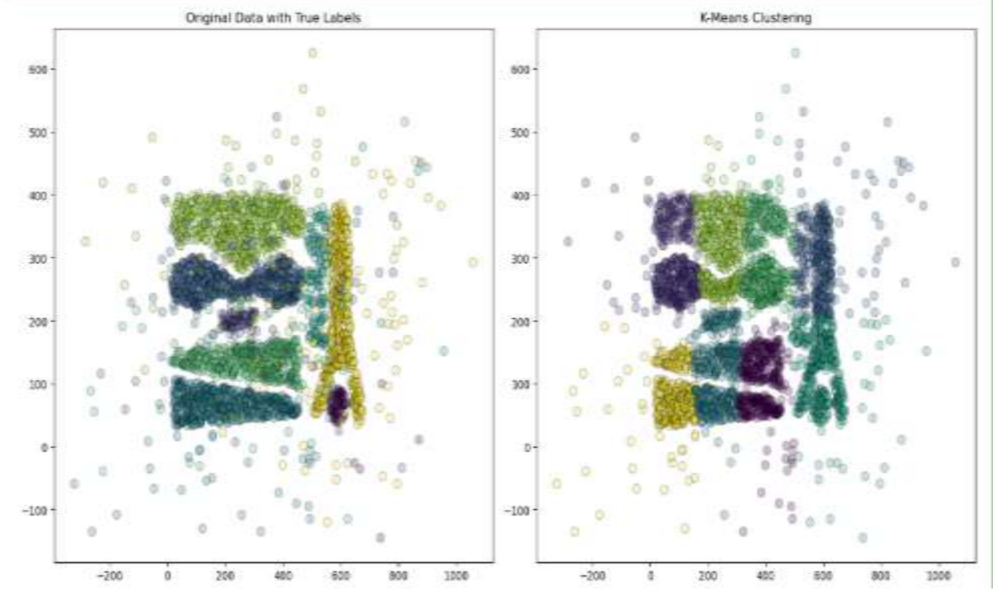
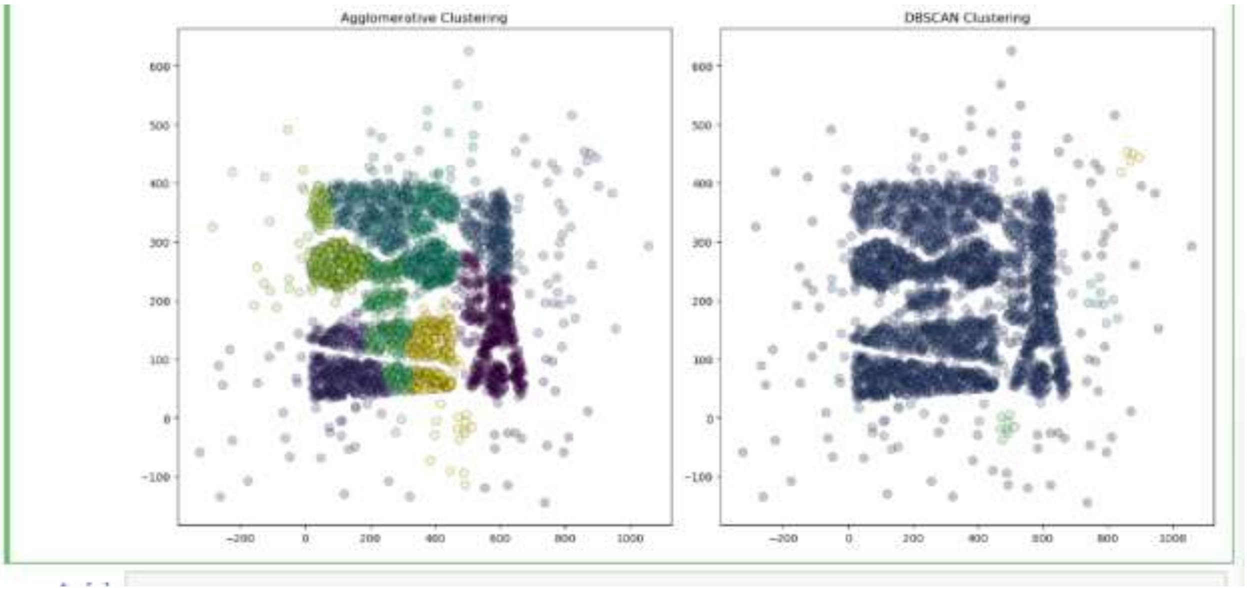
# 8.2D
# Comparing unsupervised clustering algorithms



```
In [7]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
        from sklearn.preprocessing import StandardScaler

        # Load the dataset
        df = pd.read_csv('Complex8_N15.csv')

        # Extract the features and the true labels
        X = df[['V1', 'V2']].values
        y_true = df['V3'].values

        # Normalize the dataset for DBSCAN
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)

        # Create subplots for comparison
        fig, axs = plt.subplots(2, 2, figsize=(14, 14), dpi=100)

        # Plot original data with true labels
        axs[0, 0].scatter(X[:, 0], X[:, 1], c=y_true, cmap='viridis', alpha=0.25, s=60, linewidths=1, edgecolors='black')
        axs[0, 0].set_title('Original Data with True Labels')

        # K-Means clustering
        kmeans = KMeans(n_clusters=8, random_state=42)
        y_kmeans = kmeans.fit_predict(X)
        axs[0, 1].scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis', alpha=0.25, s=60, linewidths=1, edgecolors='black')
        axs[0, 1].set_title('K-Means Clustering')

        # Agglomerative Clustering
        agglo = AgglomerativeClustering(n_clusters=8)
        y_agglo = agglo.fit_predict(X)
        axs[1, 0].scatter(X[:, 0], X[:, 1], c=y_agglo, cmap='viridis', alpha=0.25, s=60, linewidths=1, edgecolors='black')
        axs[1, 0].set_title('Agglomerative Clustering')

        # DBSCAN Clustering
        dbscan = DBSCAN(eps=0.3, min_samples=5)
        y_dbscan = dbscan.fit_predict(X_scaled)
        axs[1, 1].scatter(X[:, 0], X[:, 1], c=y_dbscan, cmap='viridis', alpha=0.25, s=60, linewidths=1, edgecolors='black')
        axs[1, 1].set_title('DBSCAN Clustering')

        # Adjust layout and show plot
        plt.tight_layout()
        plt.show()
```





Using a provided 2-dimensional dataset (Complex8_N15.csv), this Python script compares three unsupervised clustering algorithms: K-Means, Agglomerative Clustering, and DBSCAN. The panda's package is used to import the dataset, and then the pertinent features (V1 and V2) and their matching true labels (V3) are retrieved. To guarantee that every feature contributes equally to the clustering process, the dataset is normalised using the StandardScaler before the clustering algorithms are applied. This is crucial for DBSCAN because it relies heavily on distance-based computations.
In order to visually compare the original data distribution and the clustering outcomes from each technique, the script generates a collection of subplots. The first subplot uses a scatter plot to display the actual data with true labels, with different colours denoting different classes (V3). The K-Means clustering algorithm is used in the following subplot, with 8 clusters specified (n_clusters=8). To see how K-Means divides the data points into clusters, the outcomes are plotted.
Similar to this, eight clusters are used for agglomerative clustering, and the outcomes are shown in a different subplot. Data points are grouped using the hierarchical Agglomerative Clustering methodology according to how similar they are; this is represented in the clustering result.
Lastly, the normalised data (X_scaled) is subjected to the DBSCAN algorithm, which has two parameters: min_samples=5 (the minimum number of samples in a neighbourhood for a point to be considered a core point) and eps=0.3 (the maximum distance between two samples for one to be considered as in the neighbourhood of the other). The final subplot shows the resulting clusters.
The script makes it possible to compare the clustering results side by side with the original labelled data by setting up all of the plots in a 2x2 grid format. This makes it simpler to see the similarities and differences amongst the three clustering algorithms. The visual comparison is guaranteed to be understandable and intuitive through the use of standard plot styles (colour maps, marker sizes, and transparency levels). Plt.tight_layout () is used to improve readability of the layout, while plt.show () is used to display the plots.