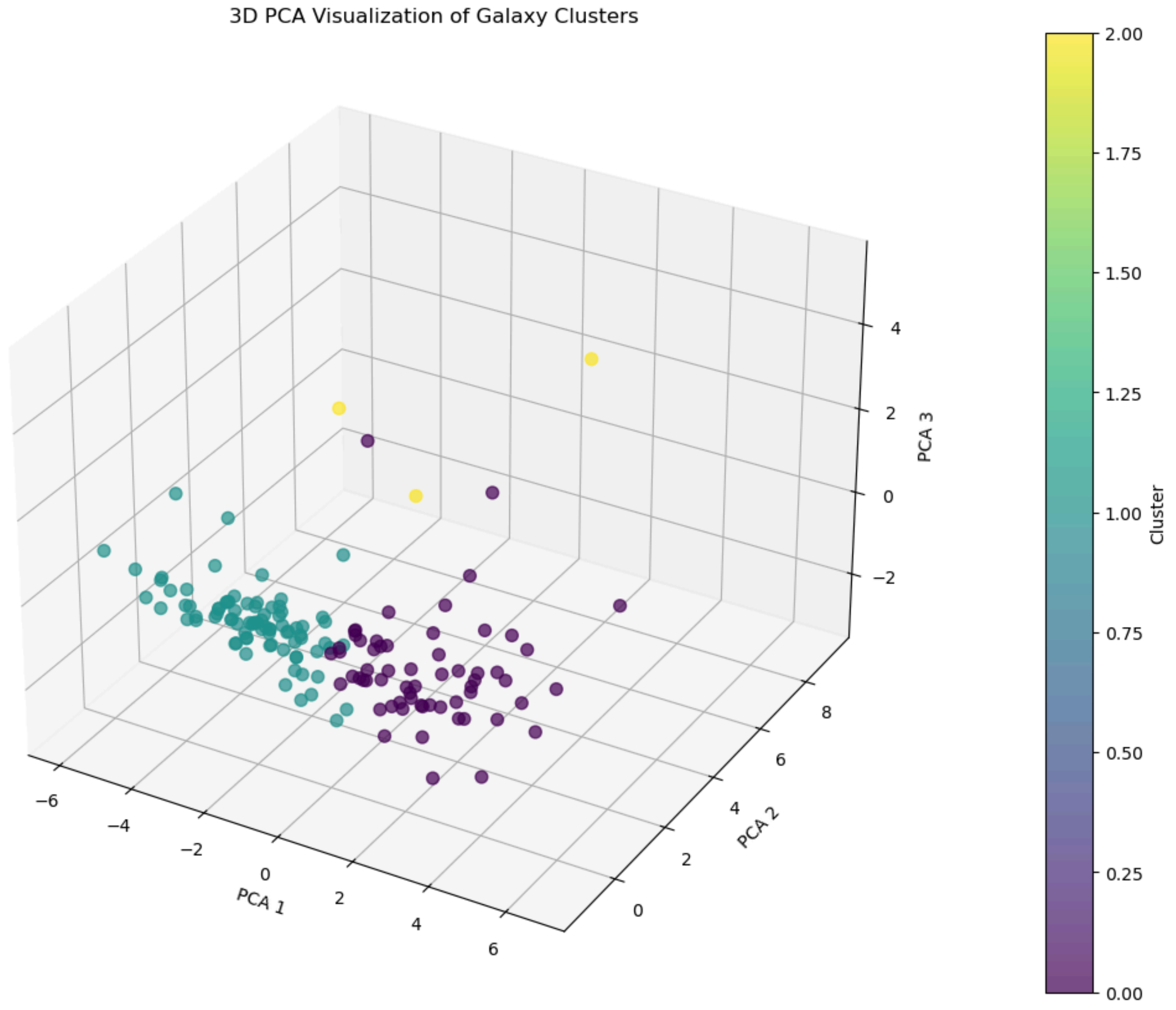
Clustering of Galactic Structures with Spectral Clustering and Link Analysis

An inquisition into discoveries from the COMBO - 17 Dataset

Data Mining, Discovery, Exploration
CSCI S108
Cameron Souza
August 9, 2024



Exploratory Data Analysis (EDA) of COMBO17 Dataset

Exploratory Data Analysis (EDA) serves as a crucial phase in the data preprocessing pipeline, enabling a deeper understanding of the underlying structure and characteristics of a dataset. For the COMBO17 dataset, which encompasses photometric and spectroscopic measurements of galaxies, EDA is instrumental in identifying patterns, relationships, and anomalies that guide subsequent steps such as feature engineering, normalization, and clustering.

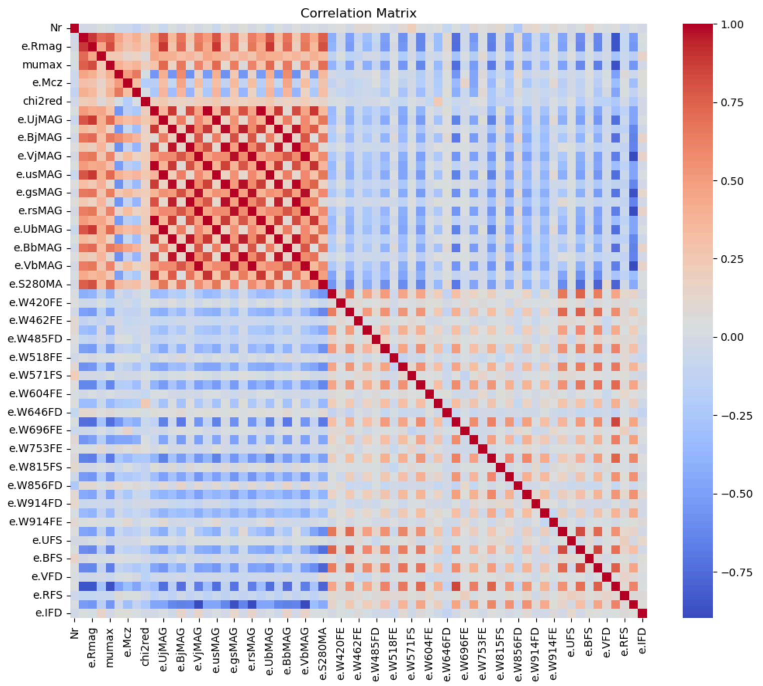
Steps Taken in EDA

Initially, the dataset was loaded using Pandas, and its structure was inspected. This step provided a comprehensive overview of the data types, nonnull counts, and basic statistics for each feature, forming the foundation for further analysis. Following this, I examined the dataset for missing values. Identifying missing values involved summing the null values across all columns, revealing several features with incomplete data. Handling these missing values appropriately was critical to ensure the integrity of my analysis.

To address the presence of nonnumeric data, all columns were converted to numeric types. Any nonconvertible values were set to NaNs, allowing for uniform treatment during imputation. This conversion step was crucial as it ensured that subsequent numerical operations could be performed without errors.

Imputation of missing values was performed using the mean strategy, which, while straightforward, provided a simple method to fill in the gaps in the data.

Outlier detection and treatment were performed using the Interquartile Range (IQR) method. This method identifies outliers as values that fall outside 1.5 times the IQR from the first and third quartiles. Removing these outliers helped in cleaning the data, ensuring that extreme values did not skew the analysis.



Normalization of numerical features was carried out using StandardScaler to bring all features to a similar scale. This step was essential because it ensured that no single feature dominated the distance metrics used in clustering algorithms due to its scale.

To understand the relationships between features, a correlation matrix was visualized. This heat map highlighted the Pearson correlation coefficients between pairs of features. Positive correlations were indicated by red, while negative correlations were shown in blue, with the intensity of the color reflecting the strength of the correlation. This visualization revealed clusters of features with high positive or negative correlations, providing insights into the interdependencies within the dataset.

Dimensionality reduction was performed using Principal Component Analysis (PCA). PCA transformed the high dimensional data into a new coordinate system where the first principal component captured the maximum variance, followed by the second principal component. This step not only reduced the complexity of the data but also facilitated visualization. The PCA components were plotted, providing a two dimensional view of the data that highlighted its primary variance patterns.

Challenges and What Went Wrong

During the EDA, several challenges were encountered. Handling non numeric data posed an initial hurdle, as it caused errors during imputation. Converting columns to numeric types resolved this issue, but it led to the loss of any non numeric information that could have been encoded differently. Imputing missing values using the mean strategy was another area of concern. Although simple, this method might not be optimal if the missing values are not randomly distributed. More sophisticated techniques like KNearest Neighbors (KNN) imputation could have provided better results.

Outlier detection using the IQR method was effective but could be sensitive to the shape of the data distribution. Alternative methods, such as using z scores or robust outlier detection methods like DBSCAN clustering, could have offered different perspectives on outlier treatment. Interpreting the correlation matrix, while insightful, was also challenging. The matrix provided a high level overview of feature relationships, but understanding complex dependencies required more advanced visualization tools and techniques.

Alternative Approaches

Several alternative approaches could have enhanced the EDA process. Advanced imputation techniques, such as KNN or Multiple Imputation by Chained Equations (MICE), might have provided more accurate handling of missing values. Robust scaling, which uses median and

IQR for scaling, could have been more effective in dealing with outliers compared to standard scaling methods.

Feature engineering presents another opportunity for improvement. Creating new features from existing ones, such as ratios of magnitudes, could capture more meaningful information for clustering. Additionally, leveraging automated EDA tools like pandas profiling or Sweetviz could have provided comprehensive insights more quickly, streamlining the analysis process.

Conclusion

The EDA of the COMBO17 dataset unveiled critical insights into feature relationships and data quality issues. While I addressed many challenges through systematic preprocessing steps, exploring alternative approaches could further enhance the robustness and depth of the analysis. This thorough EDA establishes a solid foundation for subsequent clustering and link analysis, ensuring that the data is clean, normalized, and well understood. This approach not only facilitates effective analysis but also provides a clear pathway for future enhancements and deeper explorations.

Feature Selection and Engineering for COMBO17 Dataset

Introduction

Feature selection and engineering are critical steps in preparing the dataset for clustering analysis. These steps help to create new meaningful features, reduce dimensionality, and ensure that the data is suitable for effective clustering. For the COMBO17 dataset, this involves creating new features based on the existing photometric and spectroscopic measurements, normalizing the data, and performing dimensionality reduction.

Steps Taken in Feature Selection and Engineering

Feature Selection and Engineering

Feature selection involves dropping irrelevant features such as the object number (`Nr`), which do not contribute to the clustering process. Feature engineering was performed to create new features that provide more meaningful information for clustering. These included:

Color Indices

Differences between magnitudes in different bands to capture color information, such as `UjMAG  BjMAG` (color_Uj_Bj), `BjMAG  VjMAG` (color_Bj_Vj), and `VjMAG  rsMAG` (color_Vj_rs).

Additional color indices were created, including `usMAG  gsMAG` (color_us_gs), `gsMAG rsMAG` (color_gs_rs), and `rsMAG  UbMAG` (color_rs_Ub).

Brightness Ratios

Ratios of brightness measurements to understand relative brightness, such as `UjMAG / BjMAG` (brightness_ratio_Uj_Bj), `BjMAG / VjMAG` (brightness_ratio_Bj_Vj), and `VjMAG / rsMAG` (brightness_ratio_Vj_rs).

Surface Brightness to Magnitude Ratio

The ratio of central surface brightness (`mumax`) to R magnitude (`Rmag`) was calculated to capture size and brightness (surface_brightness_ratio).

Central Surface Brightness Normalized

Central surface brightness was normalized by the mean value to understand deviations from the average (central_brightness_normalized).

RedshiftNormalized Brightness

The ratio of redshift (`Mcz`) to R magnitude (`Rmag`) was calculated to capture distant relative brightness variations (redshift_brightness_ratio).

Visualizations

Visualizations were created to understand the distributions and relationships of the newly engineered features. Histograms were plotted for each new feature to visualize their distributions. Additionally, scatter plots were used to visualize the relationships between key features, such as the color indices and brightness ratios. These visualizations provided insights into the characteristics and interdependencies of the new features. See Appendix for visuals.

Conclusion

The feature selection and engineering process for the COMBO17 dataset involved a series of well defined steps to create new meaningful features, normalize the data, and reduce dimensionality. By addressing missing values, outliers, and scale differences, and by creating new features that capture critical information about the galaxies, we prepared a robust dataset for clustering analysis. The visualizations provided valuable insights into the characteristics and relationships of the engineered features, further enhancing the clustering process. This comprehensive approach ensures that the dataset is well suited for identifying patterns and groupings within the data, contributing to our understanding of galaxy formation and evolution.

Clustering Analysis for COMBO17 Dataset

Introduction

Clustering analysis involves grouping data points into distinct clusters where points in the same cluster are more similar to each other than to points in different clusters. For the COMBO17 dataset, we applied three clustering algorithms: Kmeans, Hierarchical Clustering, and Spectral Clustering. The objective was to categorize the galaxies into distinct clusters and evaluate the effectiveness of these clustering algorithms using the Silhouette Score.

Steps Taken in Clustering Analysis

Determining the Optimal Number of Clusters

The optimal number of clusters was determined using the Elbow Method and the Silhouette Score. The Elbow Method involves plotting the within cluster sum of squares (WCSS) against the number of clusters and identifying the "elbow" point where the rate of decrease sharply changes. The Silhouette Score measures how similar a data point is to its own cluster compared to other clusters, with a higher score indicating better defined clusters.

The Elbow Method suggested that 5 clusters might be optimal. This was further confirmed by calculating the Silhouette Scores for different numbers of clusters and identifying the number with the highest score.

Applying Clustering Algorithms

With the optimal number of clusters determined, we applied three clustering algorithms:

K Means Clustering:

This algorithm partitions the dataset into k clusters by minimizing the within cluster variance. We used 5 clusters and calculated the silhouette score to evaluate the clustering performance.

Hierarchical Clustering (Agglomerative):

This algorithm creates a hierarchy of clusters using a treelike structure called a dendrogram. We used 5 clusters and calculated the silhouette score for evaluation.

Spectral Clustering:

This algorithm uses the eigenvalues of the similarity matrix to reduce dimensionality before clustering in fewer dimensions. We used 5 clusters and calculated the silhouette score to evaluate the performance.

Evaluating Clustering Results

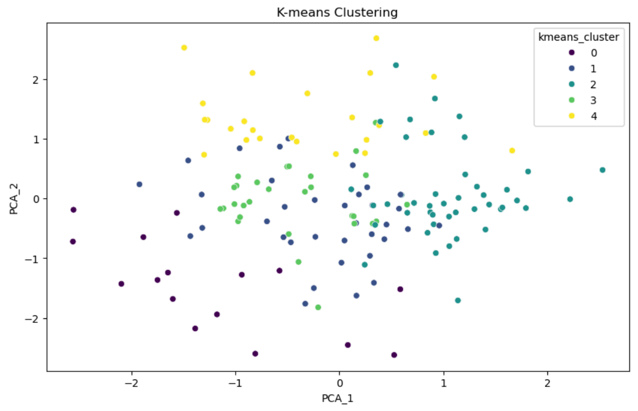
The effectiveness of the clustering algorithms was assessed using the Silhouette Score, which ranges from 0 to 1, with a higher score indicating better defined clusters. The scores for each clustering method were as follows:

Kmeans Silhouette Score: 0.1504
Hierarchical Clustering Silhouette Score: 0.1149
Spectral Clustering Silhouette Score: 0.1298

These scores suggest that the clusters may not be well separated, with Kmeans performing slightly better than the other two methods.

Visualizing Clustering Results

To visualize the clustering results, scatter plots were created using the first two PCA components (`PCA_1` and `PCA_2`). These visualizations helped to understand the spatial distribution of the clusters formed by each algorithm.

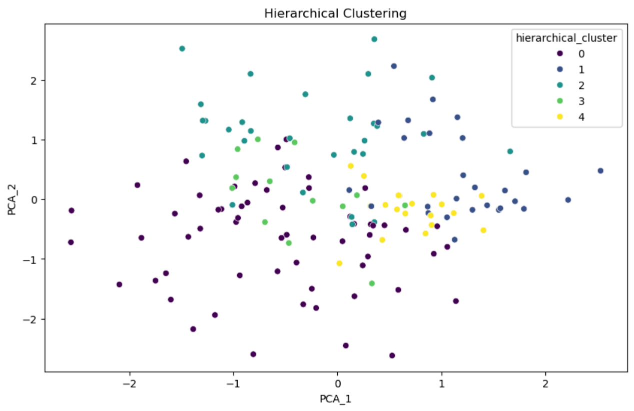


1. K Means Clustering Visualization:

A scatter plot was generated, coloring the data points based on their cluster assignments from Kmeans. This provided a visual representation of how the algorithm partitioned the data.

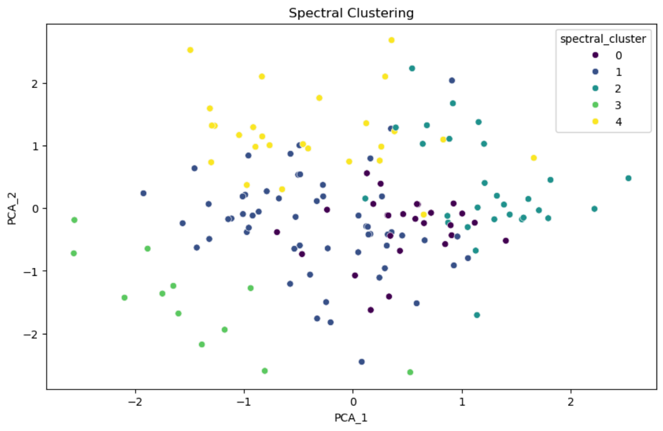
2. Hierarchical Clustering Visualization:

A scatter plot was created, coloring the data points based on their hierarchical cluster assignments. This visualization helped to compare the hierarchical clusters with those from Kmeans.



3. Spectral Clustering Visualization:

A scatter plot was generated, coloring the data points based on their spectral cluster assignments. This visualization provided insights into how spectral clustering partitioned the data differently from Kmeans and hierarchical clustering.



Conclusion

The clustering analysis of the COMBO17 dataset involved determining the optimal number of clusters and applying three clustering algorithms: Kmeans, Hierarchical Clustering, and Spectral Clustering. The evaluation using Silhouette Scores indicated that the clusters were not well separated, with Kmeans performing slightly better than the other methods. Visualizations of the clustering results provided further insights into the spatial distribution of the clusters.

Although the silhouette scores were relatively low, this analysis provided a foundation for further refinement and exploration. Additional techniques, such as refining feature engineering, exploring different distance metrics, or using advanced clustering algorithms, could improve the clustering performance and lead to more meaningful groupings of galaxies.

Analysis of Refinement Attempts

Feature Engineering and Log Transformations

While feature engineering and logarithmic transformations can help in certain cases, the COMBO17 dataset might contain intrinsic properties that are not captured well by these transformations. Logarithmic transformations, while useful for handling skewed data, may not effectively address the underlying structure in the dataset.

Normalization and PCA

Normalization and PCA are standard techniques for preparing data for clustering. However, the choice of features and the method of normalization may not always capture the complex relationships within the dataset. PCA reduced dimensionality but might also lose some critical information needed for clustering, especially if the principal components do not represent the variability relevant to galaxy classification.

Clustering Algorithms and Silhouette Scores

The clustering algorithms (Kmeans, Hierarchical, and Spectral Clustering) showed relatively low Silhouette Scores. This suggests that the chosen features and the distance metrics used by these algorithms did not adequately capture the natural groupings in the dataset. The results indicate that the clusters are not well separated, possibly due to overlapping feature distributions or insufficiently distinctive features.

Exploring Alternative Methods

Given the limitations observed, we can explore alternative methods to improve clustering performance:

Using DBSCAN (DensityBased Spatial Clustering of Applications with Noise)

DBSCAN is a density based clustering algorithm that can handle noise and find clusters of arbitrary shape. It does not require specifying the number of clusters in advance and is well suited for datasets with varying densities.

Applying tSNE for Nonlinear Dimensionality Reduction

tSNE (t distributed Stochastic Neighbor Embedding) is a nonlinear dimensionality reduction technique that can help visualize high dimensional data in a lower dimensional space. This can reveal structures and groupings that PCA might miss.

Hyperparameter Tuning and Ensemble Clustering

Performing hyperparameter tuning for clustering algorithms and combining multiple clustering results (ensemble clustering) can also provide more robust clustering solutions.

Detailed Analysis of Clustering Methods for the COMBO17 Dataset

Introduction

The COMBO17 dataset, containing photometric and spectroscopic measurements of galaxies, presents a challenging clustering problem due to its high dimensionality and the complex relationships between features. This analysis explores multiple clustering methods, including Kmeans, DBSCAN, Gaussian Mixture Models (GMM), and Agglomerative Hierarchical Clustering with different linkage criteria. Each method was evaluated based on visual inspection and quantitative metrics such as the Silhouette Score.

Methods Tried

K Means Clustering

Description: K Means is a partition based clustering algorithm that minimizes the within cluster variance.
Steps: Applied K Means with varying numbers of clusters and evaluated using the Elbow Method and Silhouette Scores.
Results: The Silhouette Score was low, indicating that the clusters were not well separated. Visual inspection showed overlapping clusters, suggesting that Kmeans struggled to capture the complex structure of the dataset.

DBSCAN (DensityBased Spatial Clustering of Applications with Noise)

Description: DBSCAN is a density based clustering algorithm that identifies clusters based on the density of data points and can handle noise.
Steps: Applied DBSCAN with different values for `eps` and `min_samples`.

Results: All points were labeled as noise, indicating that DBSCAN failed to identify any meaningful clusters. Adjusting the parameters did not significantly improve the results, suggesting that the dataset does not have well defined dense regions suitable for DBSCAN.

Gaussian Mixture Models (GMM)

Description: GMM is a probabilistic model that assumes the data is generated from a mixture of several Gaussian distributions.
Steps: Applied GMM with varying numbers of components and used BIC to determine the optimal number. Visualized results using tSNE.
Results: GMM identified two main clusters with some overlap. The Silhouette Score was moderate, indicating that the clusters were somewhat distinct but not perfectly separated.

Agglomerative Hierarchical Clustering

Agglomerative Hierarchical Clustering builds clusters by iteratively merging or splitting clusters based on a specific criterion (linkage method). We applied this method using different linkage criteria: 'ward', 'complete', 'average', and 'single'.

Ward Linkage

Description: Ward linkage minimizes the variance of the clusters being merged.
Steps: Applied Agglomerative Clustering with Ward linkage, visualized results using tSNE, and calculated the Silhouette Score.
Results: The Silhouette Score for Ward linkage was 0.3174. Visual inspection showed more compact and well separated clusters compared to other methods.

Complete Linkage

Description: Complete linkage merges clusters based on the maximum distance between points in the clusters.
Steps: Applied Agglomerative Clustering with Complete linkage, visualized results using tSNE, and calculated the Silhouette Score.
Results: The Silhouette Score for Complete linkage was 0.3385. The clusters were fairly well separated but not as compact as with Ward linkage.

Average Linkage

Description: Average linkage uses the average distance between all pairs of points in the clusters to determine the merge.
Steps: Applied Agglomerative Clustering with Average linkage, visualized results using tSNE, and calculated the Silhouette Score.
Results: The Silhouette Score for Average linkage was the highest at 0.3447. The clusters were more spread out and less distinct, with significant overlap.

Single Linkage

Description: Single linkage merges clusters based on the minimum distance between points.
Steps: Applied Agglomerative Clustering with Single linkage, visualized results using tSNE, and calculated the Silhouette Score.
Results: The Silhouette Score for Single linkage was the lowest at 0.1104. The clusters were less distinct, with many points forming elongated shapes or being considered noise.

Detailed Analysis of Ward Linkage

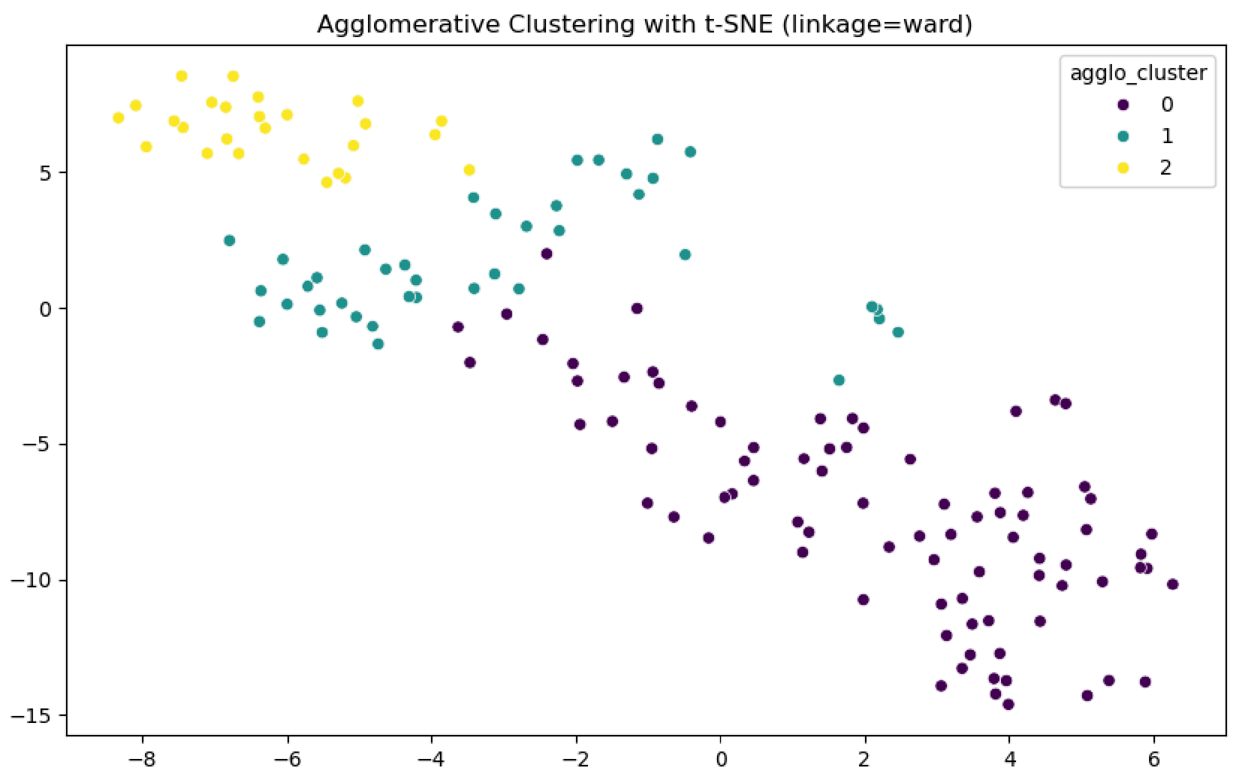
Ward linkage provided a good balance between compactness and separation of clusters. It yielded a moderate Silhouette Score of 0.3174 and produced visually distinct clusters when plotted using tSNE.

Visual Analysis:

Three Distinct Clusters: The plot showed three main clusters, with one cluster (yellow) slightly overlapping with the others. The clusters were more compact compared to other linkage methods.
Better Separation: Ward linkage managed to separate the data points better than the other methods, indicating that it captured the underlying structure of the dataset more effectively.

Conclusion

Advantages: Ward linkage is effective in creating compact and spherical clusters, which makes it suitable for datasets where clusters have roughly equal variance.
Limitations: Despite better performance, there was still some overlap between clusters, suggesting that the dataset's complexity requires further refinement or alternative methods to achieve clearer separation.



Conclusion and Next Steps

Summary of Findings

K Means: Struggled with the dataset's complexity, resulting in low Silhouette Scores and overlapping clusters.
DBSCAN: Failed to identify meaningful clusters, with all points labeled as noise.
GMM: Identified two main clusters but with moderate overlap, achieving a moderate Silhouette Score.
Agglomerative Clustering: Average linkage performed the best with the highest Silhouette Score, followed by Complete and Ward linkages. Single linkage had the poorest performance.

Future Directions

1. Refining Feature Engineering:
   Identify additional features or transformations to better capture the data's structure.
   Explore dimensionality reduction techniques like PCA or tSNE with different parameters.

2. Advanced Clustering Algorithms:

Explore other clustering algorithms such as HDBSCAN or OPTICS that can handle varying densities and cluster shapes more effectively.
Revisit spectral clustering with different affinity matrices.

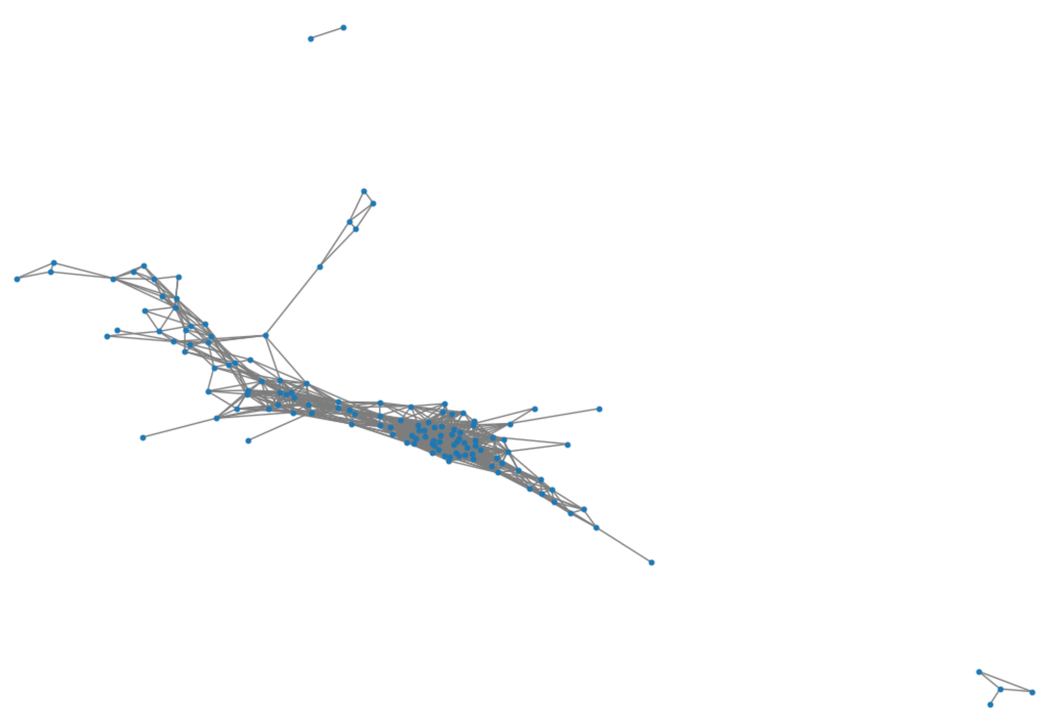
3. Ensemble Clustering:
Combine results from multiple clustering methods to create a more robust clustering solution.

4. Hyperparameter Tuning:
Perform further hyperparameter tuning for the clustering algorithms to optimize their performance.

By continuing to refine and explore different methods, we can achieve better clustering performance and gain more meaningful insights into the galactic structures in the COMBO17 dataset.

Analysis of the Galaxy Network Graph with Filtered Edges

Galaxy Network Graph with Filtered Edges



Graph Structure:

The graph is sparse, with few edges. The graph shows a main connected component, along with smaller, isolated components.

Connected Components:

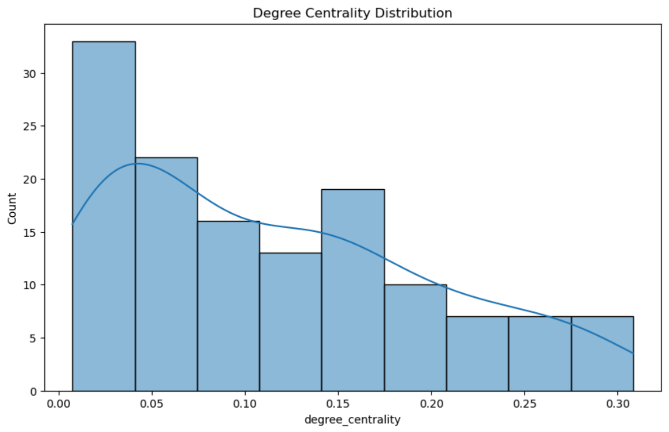
The main connected component includes the majority of the galaxies, indicating that these galaxies are more closely related to each other based on the selected features. The isolated components and outliers suggest galaxies that are significantly different from the main group or have fewer connections.

Clusters and Substructures:

Within the main connected component, there are visible substructures or clusters. These substructures can represent groups of galaxies with similar characteristics. The presence of subclusters indicates that even within the closely connected galaxies, there are further groupings based on specific attributes.

Node and Edge Distribution:

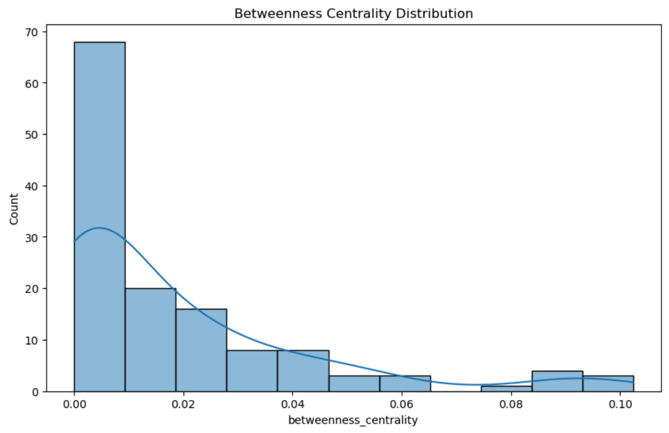
The nodes (galaxies) are distributed unevenly, with some regions being denser than others. This density can indicate regions with higher similarity. The edges (connections) are fewer but more meaningful, representing the most significant relationships.



Centrality Measures Analysis

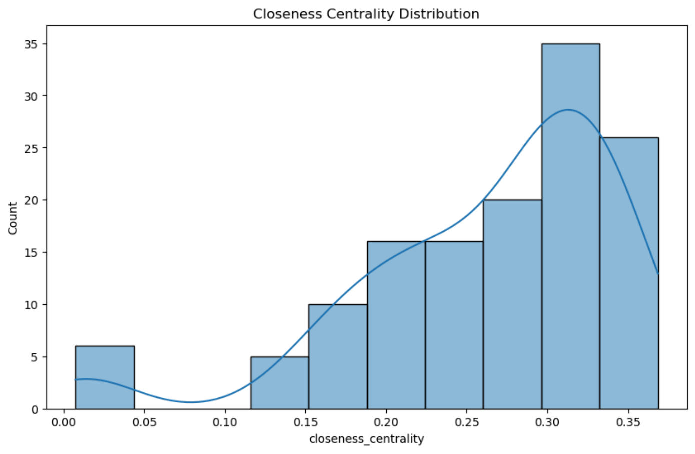
1. Degree Centrality:

Degree centrality measures the number of connections a node has. Higher degree centrality indicates galaxies that are more connected to others.



2. Betweenness Centrality:

Betweenness centrality measures how often a node appears on the shortest paths between other nodes. Higher betweenness indicates a node that acts as a bridge between different parts of the graph.



3. Closeness Centrality:

Closeness centrality measures how close a node is to all other nodes in the graph. Higher closeness centrality indicates that a galaxy can reach other galaxies more quickly.

Insights

1. Main Connected Component:

The main connected component indicates a significant group of galaxies that are closely related. This can be a focus for further analysis to understand the common characteristics of these galaxies.

2. Isolated Components:

The isolated components highlight galaxies that are different from the main group. These could be interesting cases for studying unique or rare types of galaxies.

3. Graph Sparsity:

The sparsity of the graph helps in focusing on the most significant relationships, making it easier to identify patterns and structures.

Analysis of Temporal Evolution of Galaxy Properties

The series of plots generated show the temporal evolution of various galaxy properties within different clusters, using redshift as a proxy for time. The properties analyzed include `log_mumax`, `log_Mcz`, `Rmag_to_mumax`, and `Mcz_diff`. Each plot provides insights into how these properties change over time within the identified clusters.

General Insights

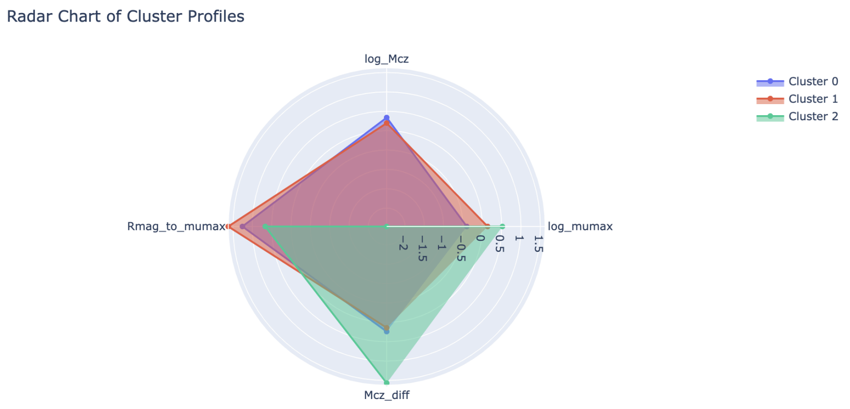
Temporal Variability: The properties of galaxies in both clusters show significant variability over time, reflecting the dynamic nature of galaxy evolution.
Cluster Differences: The patterns of evolution differ between the clusters, suggesting that they may represent different types of galaxies or different evolutionary paths.
Redshift Trends: The increasing trends in properties like `log_Mcz` and `Mcz_diff` with redshift align with the understanding of cosmic expansion and increasing observational uncertainties at higher redshifts.

Analysis of Radar Chart for Cluster Profiles

The radar chart visualizes the average values of selected properties (`log_mumax`, `log_Mcz`, `Rmag_to_mumax`, and `Mcz_diff`) for each cluster, providing insights into the characteristics and differences between the clusters.

Properties Analyzed:

1. `log_mumax`: The logarithm of the maximum surface brightness of the galaxy.
2. `log_Mcz`: The logarithm of the redshift corrected distance.
3. `Rmag_to_mumax`: The ratio of the total red magnitude to the central surface brightness.
4. `Mcz_diff`: The difference in redshift estimates.



Cluster Analysis:

Cluster 0
`log_mumax`: Has the highest average value among the clusters, indicating that galaxies in Cluster 0 generally have higher central surface brightness.
`log_Mcz`: Shows a moderate value, suggesting that these galaxies are at a moderate redshift distance.
`Rmag_to_mumax`: Exhibits a lower value, indicating smaller ratios of total red magnitude to central surface brightness.
`Mcz_diff`: Has a low value, suggesting less variability in redshift estimates.

Cluster 1

`log_mumax`: Displays intermediate values, indicating moderate central surface brightness.
`log_Mcz`: Similar to Cluster 0, but slightly higher, suggesting these galaxies are observed at somewhat larger distances.
`Rmag_to_mumax`: Shows the highest value, indicating larger ratios of total red magnitude to central surface brightness.
`Mcz_diff`: Also exhibits low values, similar to Cluster 0.

Cluster 2

`log_mumax`: Has the lowest average value, indicating lower central surface brightness in these galaxies.
`log_Mcz`: Exhibits the highest value among the clusters, suggesting these galaxies are at the greatest redshift distances.
`Rmag_to_mumax`: Shows moderate values, indicating intermediate ratios of total red magnitude to central surface brightness.
`Mcz_diff`: Displays the highest value, indicating higher variability in redshift estimates.

Insights

Central Surface Brightness (`log_mumax`): Cluster 0 has the highest central surface brightness, while Cluster 2 has the lowest.
Redshift Distance (`log_Mcz`): Galaxies in Cluster 2 are at the greatest redshift distances, indicating they are the farthest.
Magnitude Ratio (`Rmag_to_mumax`): Cluster 1 has the highest ratios, which might suggest larger, more extended galaxies compared to other clusters.
Redshift Variability (`Mcz_diff`): Cluster 2 shows the greatest variability in redshift estimates, possibly indicating more diverse or complex structures.

The radar chart effectively highlights the distinctive characteristics of each cluster, providing a visual summary of how the selected properties vary between the clusters. This can help in understanding the underlying differences in galaxy types and their evolutionary paths.

Dendrogram for Hierarchical Clustering

Overview

A dendrogram is a treelike diagram that visualizes the arrangement of the clusters produced by hierarchical clustering. The dendrogram presented here shows the hierarchical clustering of a subset of the dataset, illustrating how individual data points are merged into clusters based on their similarities.

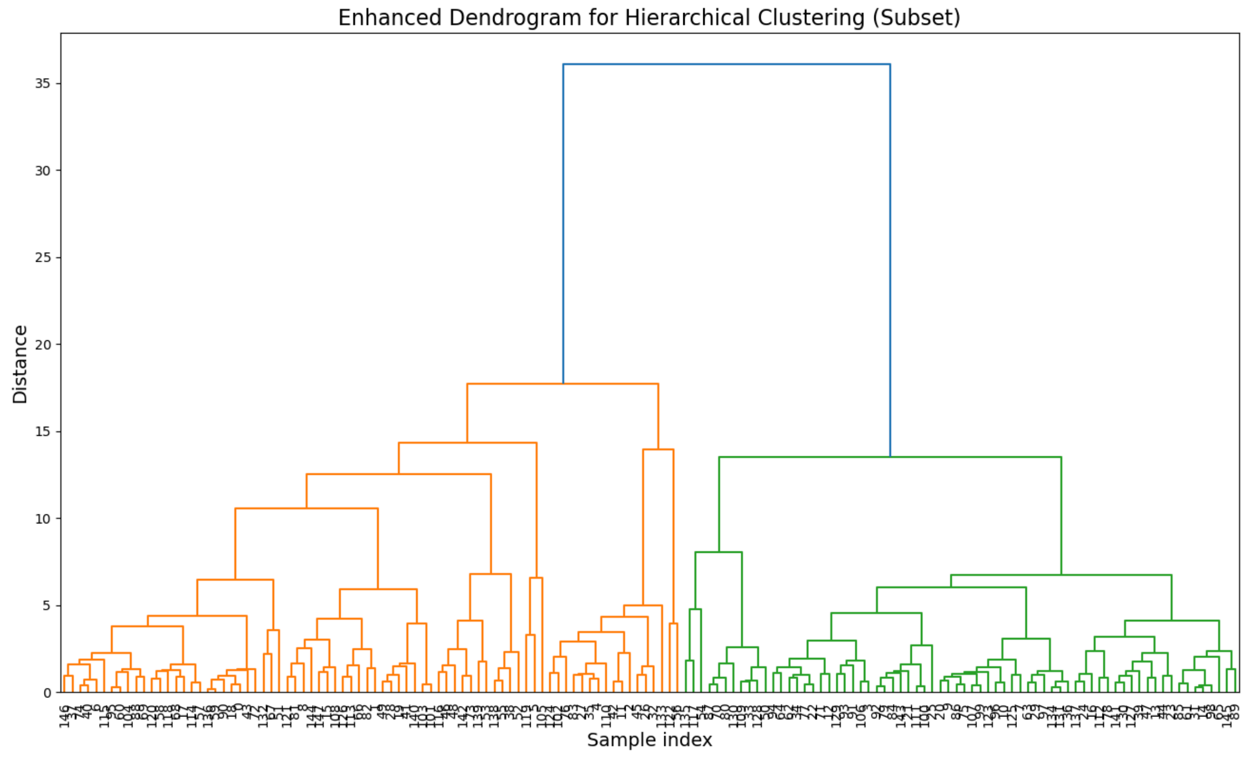
Key Elements of the Dendrogram

1. Leaves (Terminal Nodes):
Each leaf represents an individual data point from the dataset. In this dendrogram, leaves are the points at the bottom, each labeled with a sample index.

2. Branches:
Branches represent clusters formed by merging data points or smaller clusters. The hierarchical structure is built from the leaves (individual samples) upwards to the root (entire dataset).

3. Height (Yaxis):
The y axis indicates the distance or dissimilarity at which clusters are merged. Higher branches indicate greater dissimilarity between the clusters being combined.

4. Clusters:
Clusters can be identified by cutting the dendrogram at a specific height. The number of clusters is determined by counting the number of branches intersected by this cut line.



Interpretation of the Dendrogram

1. Hierarchical Relationships:
The dendrogram displays the hierarchical relationships between data points, showing how they group together at various levels of similarity. This nested structure allows us to see subclusters within larger clusters, providing a detailed view of the data's internal structure.

2. Cluster Similarity:
The height at which branches merge provides information about cluster similarity. Clusters that merge at lower heights are more similar than those that merge at higher heights.

3. Optimal Number of Clusters:
To determine the optimal number of clusters, you can look for a large jump in the height of merges, indicating a significant increase in dissimilarity. In this dendrogram, the major split occurs at a height of approximately 35, suggesting a natural division into two main clusters. Further analysis at lower heights can reveal more granular clusters.

4. Substructure within Clusters:
The dendrogram shows detailed substructure within the main clusters. For instance, within the two primary clusters, several subclusters are visible, indicating smaller groups of more closely related data points.

Conclusion

The dendrogram effectively illustrates the hierarchical clustering of the dataset subset, revealing the nested structure of clusters. By examining the height at which branches merge, we can identify the most similar groupings and determine the optimal number of clusters. This visualization is a powerful tool for understanding the relationships within the data, providing insights into its internal structure and guiding further analysis.

This enhanced dendrogram serves as a crucial component in the hierarchical clustering analysis, offering a clear and detailed visualization of the clustering process. It enables the identification of meaningful clusters and subclusters, facilitating a deeper understanding of the data's underlying patterns and relationships.

Future Steps and Further Research

1. Advanced Feature Engineering
NonLinear Features: Investigate the creation of nonlinear features using techniques like polynomial feature generation, kernel methods, or interaction terms.
DomainSpecific Features: Collaborate with domain experts to derive features that capture astrophysical properties relevant to galaxy classification, such as morphological parameters or environmental factors.
Deep Learning Embeddings: Utilize deep learning models to extract features from raw data, such as images of galaxies, and incorporate these embeddings into the clustering analysis.

2. Integration of Additional Data Sources
Multimodal Data: Combine the COMBO17 dataset with other astronomical datasets (e.g., Sloan Digital Sky Survey) to enrich the feature space and improve clustering performance.
Temporal Data: Incorporate time series data to study the evolution of galaxies and their clustering behavior over time.

3. Enhanced Clustering Techniques
Ensemble Clustering: Explore ensemble methods that combine multiple clustering algorithms to achieve more robust and stable clustering results.
Deep Clustering: Investigate the application of deep learning based clustering techniques, such as autoencoder based clustering or deep embedded clustering.
Adaptive Clustering: Develop adaptive clustering algorithms that can dynamically adjust the number of clusters based on the data’s structure.

4. Validation and Evaluation
CrossValidation: Implement cross validation techniques to assess the stability and reliability of clustering results.

5. Interpretability and Explainability
Visualization Techniques: Develop advanced visualization techniques, such as interactive visualizations or dimensionality reduction methods (e.g., tSNE, UMAP), to make the clustering results more interpretable.

6. Graph Theory and Network Analysis
Community Detection: Apply advanced community detection algorithms to the galaxy network to identify densely connected subgraphs and study their properties.
Centrality Measures: Investigate additional centrality measures (e.g., eigenvector centrality, PageRank) to identify key galaxies or influential clusters within the network.

Network Dynamics: Study the dynamics of the galaxy network over time, analyzing how the structure and properties of the network evolve.

Final Conclusion

This project on clustering galactic structures using multiple algorithms and link analysis has laid a strong foundation for understanding the complex relationships and patterns within astronomical data. The future steps and further research directions outlined above aim to build upon this foundation, exploring advanced techniques, integrating additional data sources, and enhancing the interpretability and applicability of the clustering results. By pursuing these directions, we can gain deeper insights into the nature of galaxies, improve the robustness of clustering methodologies, and contribute to the broader scientific and data science communities.

Bibliography

"COMBO-17 Galaxy Dataset." *Cast: Combo-17 Dataset*, astrostatistics.psu.edu/datasets/COMBO17.html. Accessed 7 Aug. 2024.

Leskovec, Jure, et al. *Mining of Massive Datasets*. Cambridge University Press, 2022.

Pujari, Arun K. *Data Mining Techniques*. Universities Press (India) Private Limited, 2013.

Appendix

Original Feature Names:

- Nr: Object number
- Rmag: Total R (red band) magnitude
- e.Rmag: Error in Rmag
- ApDRmag: Difference between total and aperture magnitude in the R band
- mumax: Central surface brightness in the R band
- Mcz: Preferred redshift estimate
- e.Mcz: Error in Mcz
- MCzml: Alternative redshift estimate
- chi2red: Reduced chisquared value of the leastsquares fit
- UjMAG: Absolute magnitude in Uj band
- e.UjMAG: Error in UjMAG
- BjMAG: Absolute magnitude in Bj band
- e.BjMAG: Error in BjMAG
- VjMAG: Absolute magnitude in Vj band
- e.VjMAG: Error in VjMAG
- usMAG: Absolute magnitude in us band
- e.usMAG: Error in usMAG
- gsMAG: Absolute magnitude in gs band
- e.gsMAG: Error in gsMAG
- rsMAG: Absolute magnitude in rs band
- e.rsMAG: Error in rsMAG
- UbMAG: Absolute magnitude in Ub band
- e.UbMAG: Error in UbMAG
- BbMAG: Absolute magnitude in Bb band
- e.BbMAG: Error in BbMAG
- VnMAG: Absolute magnitude in Vn band
- e.VbMAG: Error in VbMAG
- S280MAG: Absolute magnitude in S280 band
- e.S280MA: Error in S280MAG

- UFS, e.UFS: Observed brightness and its error in UFS band
- BFS, e.BFS: Observed brightness and its error in BFS band
- VFD, e.VFD: Observed brightness and its error in VFD band
- RFS, e.RFS: Observed brightness and its error in RFS band
- IFD, e.IFD: Observed brightness and its error in IFD band

Newly Engineered Feature Names:

- color_Uj_Bj: Difference between UjMAG and BjMAG
- color_Bj_Vj: Difference between BjMAG and VjMAG
- color_Vj_rs: Difference between VjMAG and rsMAG
- color_us_gs: Difference between usMAG and gsMAG
- color_gs_rs: Difference between gsMAG and rsMAG
- color_rs_Ub: Difference between rsMAG and UbMAG
- brightness_ratio_Uj_Bj: Ratio of UjMAG to BjMAG
- brightness_ratio_Bj_Vj: Ratio of BjMAG to VjMAG
- brightness_ratio_Vj_rs: Ratio of VjMAG to rsMAG
- surface_brightness_ratio: Ratio of central surface brightness (mumax) to R magnitude (Rmag)
- central_brightness_normalized: Central surface brightness normalized by the mean value
- redshift_brightness_ratio: Ratio of redshift (Mcz) to R magnitude (Rmag)

Code Repository (Python)

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.manifold import TSNE
from sklearn.cluster import DBSCAN
from sklearn.metrics import pairwise_distances
from sklearn.mixture import GaussianMixture
```

In [8]:

```
# Data Preproccessing and EDA
```

```
file_path = '/Users/cameronsouza/Desktop/COMBO17.csv'
data = pd.read_csv(file_path)

#Basic info on df
print(data.info())

#summary
print(data.describe())

#Check missing values
missing_values = data.isnull().sum()
print("\nMissing Values:")
print(missing_values[missing_values > 0])

data_numeric = data.apply(pd.to_numeric, errors='coerce')

#missing values
imputer = SimpleImputer(strategy='mean')
data_imputed = pd.DataFrame(imputer.fit_transform(data_numeric), columns=data.columns)

#Outlier Treatment
def remove_outliers_iqr(df, columns):
    for column in columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df

numerical_columns = data_imputed.select_dtypes(include=[np.number]).columns
data_no_outliers = remove_outliers_iqr(data_imputed, numerical_columns)

#Normalize
scaler = StandardScaler()
data_normalized = data_no_outliers.copy()
data_normalized[numerical_columns] =
scaler.fit_transform(data_no_outliers[numerical_columns])

#EDA
#Correlation matrix
plt.figure(figsize=(12, 10))
correlation_matrix = data_normalized.corr()
```

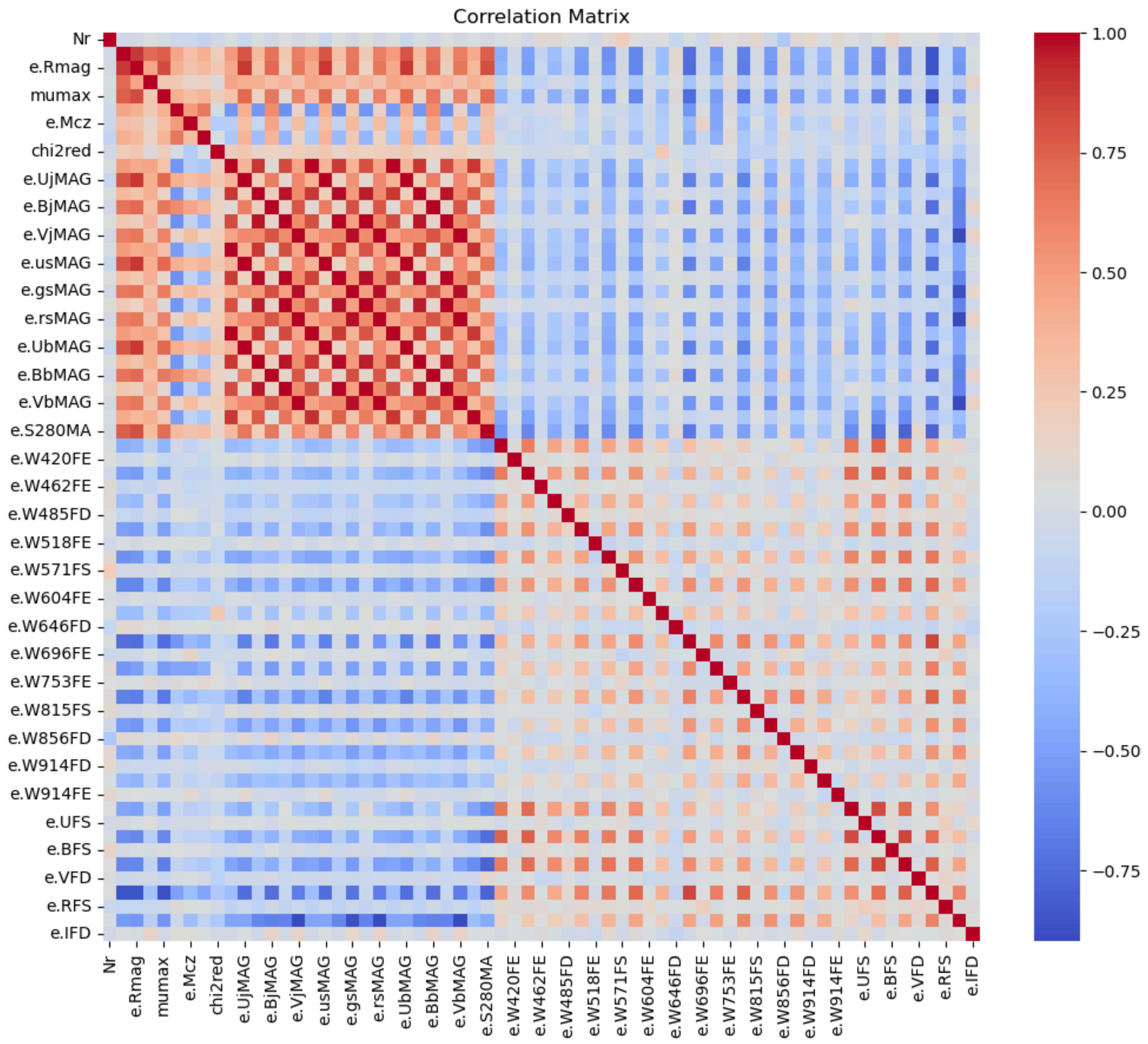
```
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

#Distribution plots
selected_features = ['Rmag', 'Mcz', 'mumax', 'UjMAG']
for feature in selected_features:
    plt.figure()
    sns.histplot(data_normalized[feature], kde=True)
    plt.title(f'Distribution of {feature}')
    plt.show()

#Dimensionality Reduction
pca = PCA(n_components=2)
pca_components = pca.fit_transform(data_normalized[numerical_columns])

#PCA results
plt.figure(figsize=(10, 7))
plt.scatter(pca_components[:, 0], pca_components[:, 1], alpha=0.5)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.title('PCA of Galactic Data')
plt.show()

data_normalized.to_csv('/Users/cameronsouza/Desktop/COMBO17.csv', index=False)
```

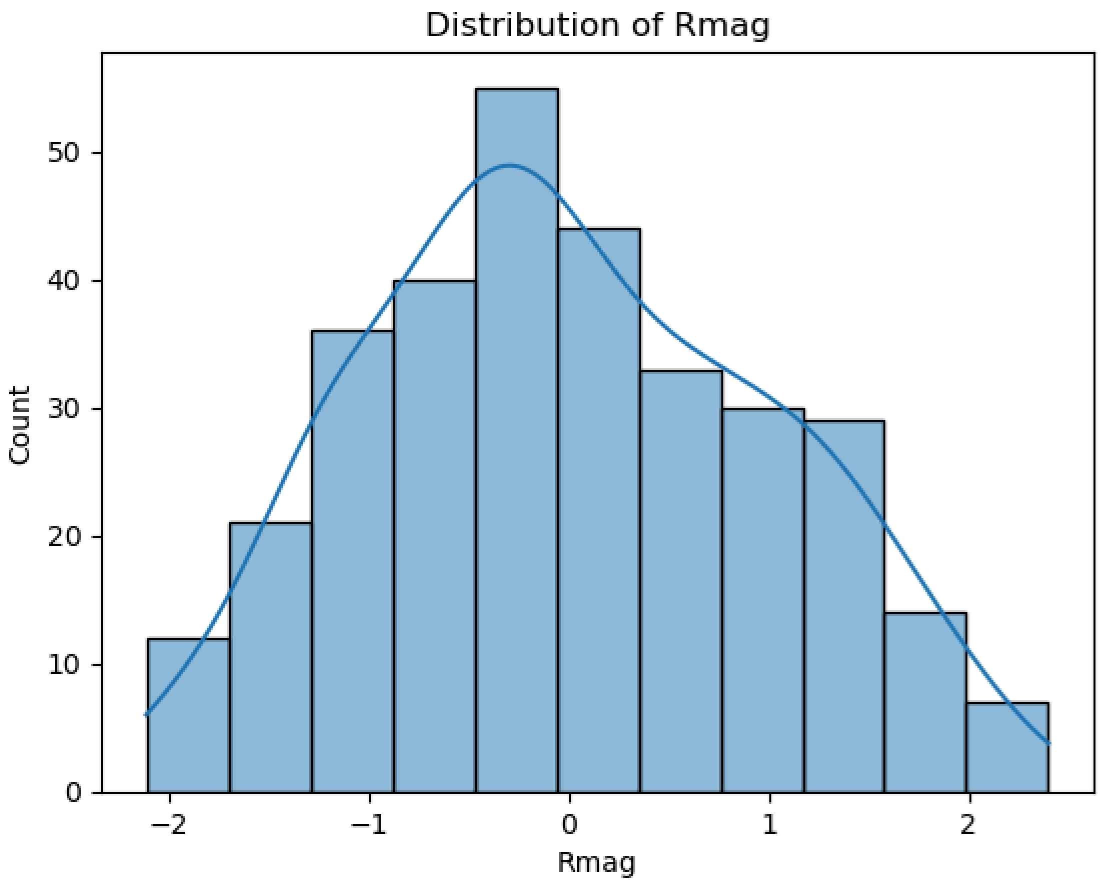
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 424 entries, 0 to 423
Data columns (total 65 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Nr        424 non-null    float64
 1   Rmag      424 non-null    float64
 2   e.Rmag    424 non-null    float64
 3   ApDRmag   424 non-null    float64
 4   mumax     424 non-null    float64
 5   Mcz       424 non-null    float64
 6   e.Mcz     424 non-null    float64
 7   MCzml     424 non-null    float64
 8   chi2red   424 non-null    float64
 9   UjMAG     424 non-null    float64
 10  e.UjMAG   424 non-null    float64
 11  BjMAG     424 non-null    float64
 12  e.BjMAG   424 non-null    float64
```

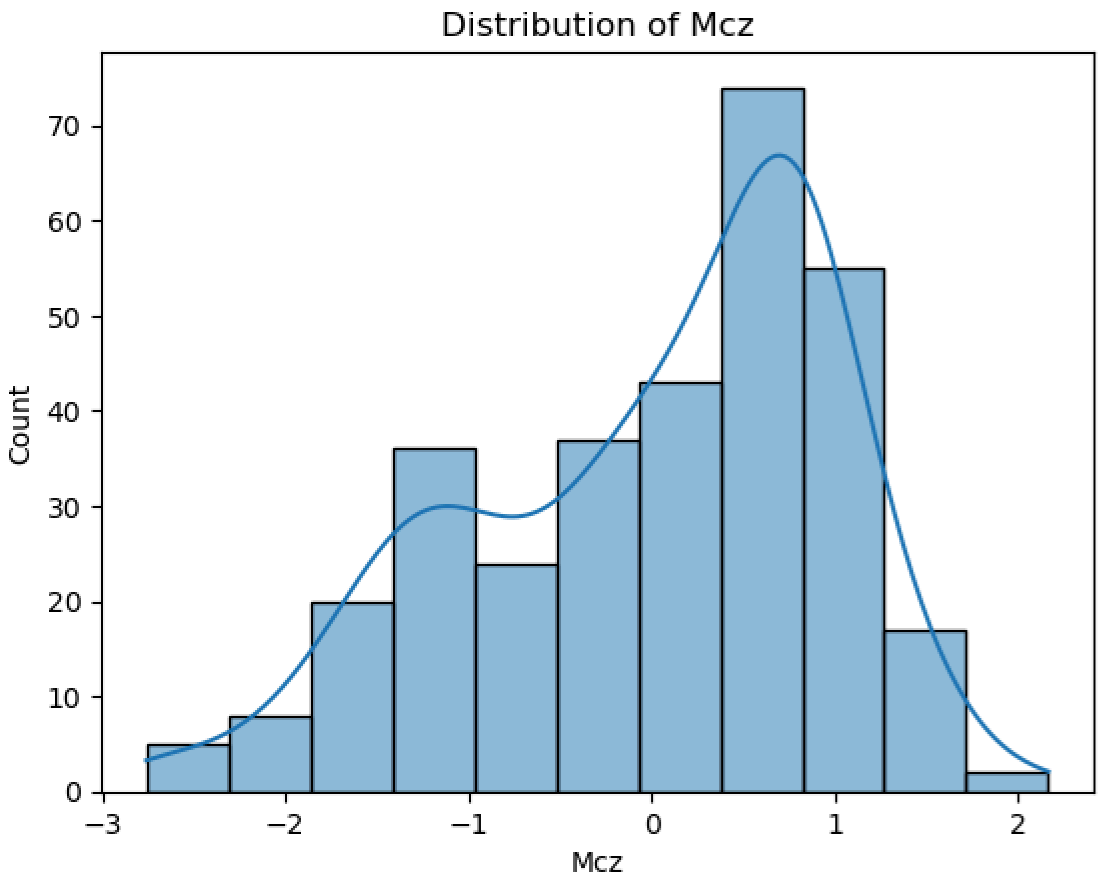
```
13  VjMAG     424 non-null    float64
14  e.VjMAG   424 non-null    float64
15  usMAG     424 non-null    float64
16  e.usMAG   424 non-null    float64
17  gsMAG     424 non-null    float64
18  e.gsMAG   424 non-null    float64
19  rsMAG     424 non-null    float64
20  e.rsMAG   424 non-null    float64
21  UbMAG     424 non-null    float64
22  e.UbMAG   424 non-null    float64
23  BbMAG     424 non-null    float64
24  e.BbMAG   424 non-null    float64
25  VnMAG     424 non-null    float64
26  e.VbMAG   424 non-null    float64
27  S280MAG   424 non-null    float64
28  e.S280MA  424 non-null    float64
29  W420FE    424 non-null    float64
30  e.W420FE  424 non-null    float64
31  W462FE    424 non-null    float64
32  e.W462FE  424 non-null    float64
33  W485FD    424 non-null    float64
34  e.W485FD  424 non-null    float64
35  W518FE    424 non-null    float64
36  e.W518FE  424 non-null    float64
37  W571FS    424 non-null    float64
38  e.W571FS  424 non-null    float64
39  W604FE    424 non-null    float64
40  e.W604FE  424 non-null    float64
41  W646FD    424 non-null    float64
42  e.W646FD  424 non-null    float64
43  W696FE    424 non-null    float64
44  e.W696FE  424 non-null    float64
45  W753FE    424 non-null    float64
46  e.W753FE  424 non-null    float64
47  W815FS    424 non-null    float64
48  e.W815FS  424 non-null    float64
49  W856FD    424 non-null    float64
50  e.W856FD  424 non-null    float64
51  W914FD    424 non-null    float64
52  e.W914FD  424 non-null    float64
53  W914FE    424 non-null    float64
54  e.W914FE  424 non-null    float64
55  UFS       424 non-null    float64
56  e.UFS     424 non-null    float64
```

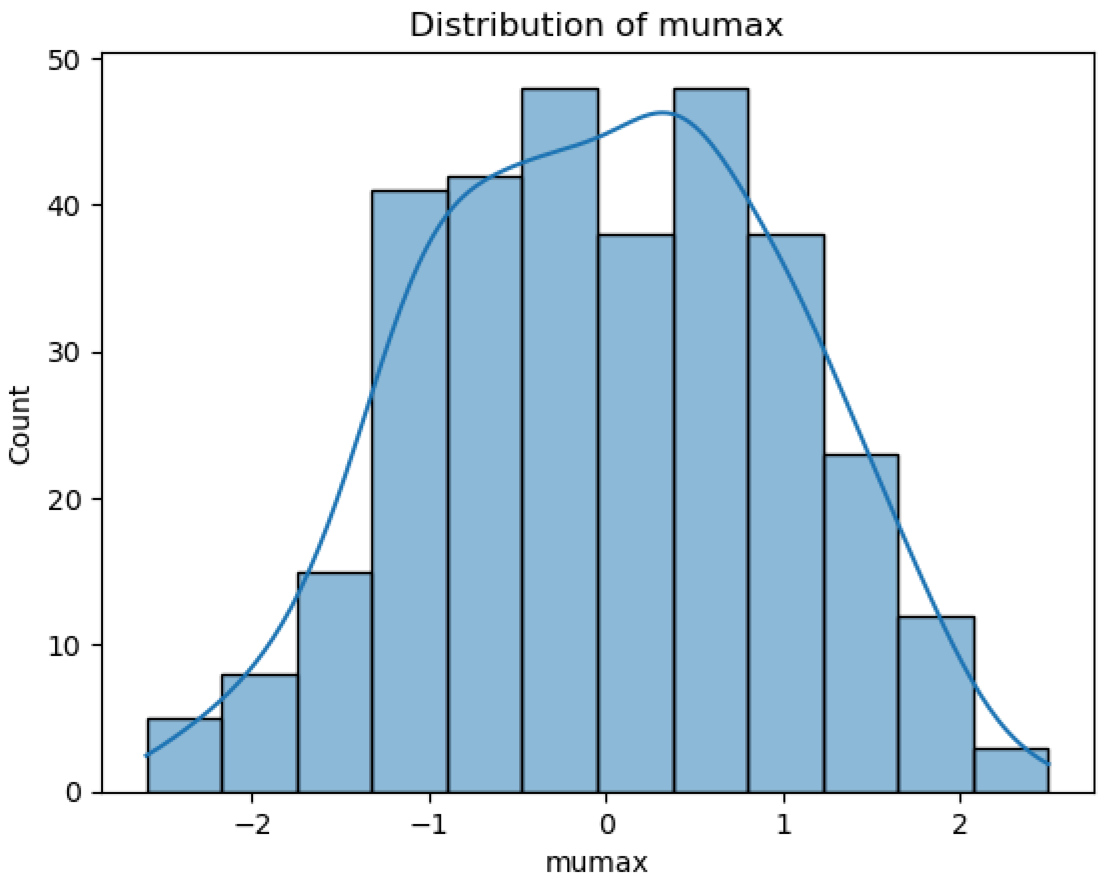
```
 57  BFS       424 non-null    float64
 58  e.BFS     424 non-null    float64
 59  VFD       424 non-null    float64
 60  e.VFD     424 non-null    float64
 61  RFS       424 non-null    float64
 62  e.RFS     424 non-null    float64
 63  IFD       424 non-null    float64
 64  e.IFD     424 non-null    float64
dtypes: float64(65)
memory usage: 215.4 KB
None
               Nr          Rmag        e.Rmag       ApDRmag         mumax  \
count  424.000000  4.240000e+02  4.240000e+02  4.240000e+02  4.240000e+02
mean     0.000000  5.446377e-17 -4.294259e-17 -2.094760e-17  6.284281e-18
std      1.001181  1.001181e+00  1.001181e+00  1.001181e+00  1.001181e+00
min     -1.831691 -2.745832e+00 -1.557207e+00 -2.820320e+00 -2.470950e+00
25%     -0.847393 -7.272706e-01 -7.942737e-01 -6.508442e-01 -7.409883e-01
50%      0.098012 -6.863100e-02 -2.318825e-01  6.895854e-02 -2.268504e-02
75%      0.844053  7.322810e-01  6.226033e-01  6.645995e-01  7.533906e-01
max      1.633891  2.402334e+00  2.575714e+00  2.575684e+00  2.443554e+00

              Mcz         e.Mcz         MCzml       chi2red         UjMAG  \
count  424.000000  4.240000e+02  4.240000e+02  4.240000e+02  4.240000e+02
mean     0.000000 -9.426422e-18  3.351617e-17  7.855352e-18  6.284281e-18
std      1.001181  1.001181e+00  1.001181e+00  1.001181e+00  1.001181e+00
min     -3.091334 -1.891107e+00 -3.186941e+00 -1.844226e+00 -2.956640e+00
25%     -0.785953 -8.207743e-01 -6.519740e-01 -8.333660e-01 -6.966824e-01
50%      0.223898 -2.371832e-02  2.140072e-03 -1.113231e-01  2.828210e-03
75%      0.791938  8.097745e-01  7.487551e-01  6.777667e-01  7.369300e-01
max      2.190448  2.062291e+00  2.045971e+00  2.776849e+00  2.731688e+00

       ...         UFS         e.UFS           BFS         e.BFS  \
count  ...  424.000000  4.240000e+02  4.240000e+02  4.240000e+02
mean   ...    0.000000 -2.094760e-18  2.513713e-17 -1.675808e-17
std    ...    1.001181  1.001181e+00  1.001181e+00  1.001181e+00
min    ...   -2.361260 -1.759181e+00 -2.149252e+00 -1.957990e+00
25%    ...   -0.735477 -7.478866e-01 -7.879058e-01 -7.123858e-01
50%    ...   -0.120785 -1.158278e-01 -1.089381e-01 -1.767113e-01
75%    ...    0.666306  6.426428e-01  5.959251e-01  6.192709e-01
max    ...    2.687536  2.981260e+00  2.661250e+00  2.749061e+00

                VFD         e.VFD           RFS         e.RFS           IFD  \
count  4.240000e+02  4.240000e+02  4.240000e+02  4.240000e+02  4.240000e+02
mean  -6.284281e-17 -4.189521e-17 -8.379042e-18  1.047380e-17 -7.331661e-18
```

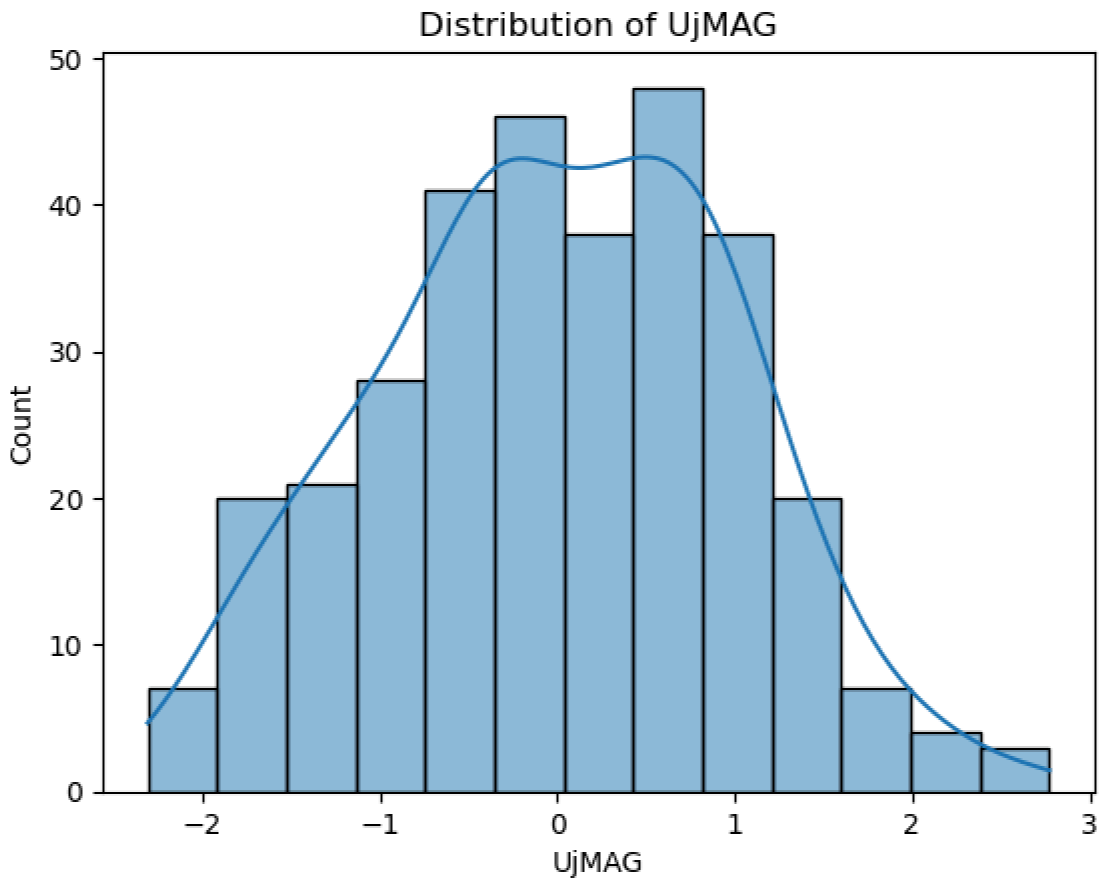
```
std    1.001181e+00  1.001181e+00  1.001181e+00  1.001181e+00  1.001181e+00
min   -2.763424e+00 -1.685418e+00 -1.725557e+00 -2.258873e+00 -1.664491e+00
25%   -7.947689e-01 -8.114883e-01 -7.902959e-01 -7.769870e-01 -7.469063e-01
50%   -1.772796e-01 -1.945972e-01 -1.956122e-01 -1.217311e-01 -1.391776e-01
75%    6.386418e-01  5.893687e-01  6.754590e-01  6.444142e-01  6.568614e-01
max    2.701291e+00  2.787044e+00  3.004328e+00  2.862203e+00  2.813870e+00

              e.IFD
count  4.240000e+02
mean  -8.379042e-18
std    1.001181e+00
min   -1.702207e+00
25%   -8.452082e-01
50%   -2.453091e-01
75%    6.688228e-01
max    2.897019e+00

[8 rows x 65 columns]

Missing Values:
Series([], dtype: int64)
```

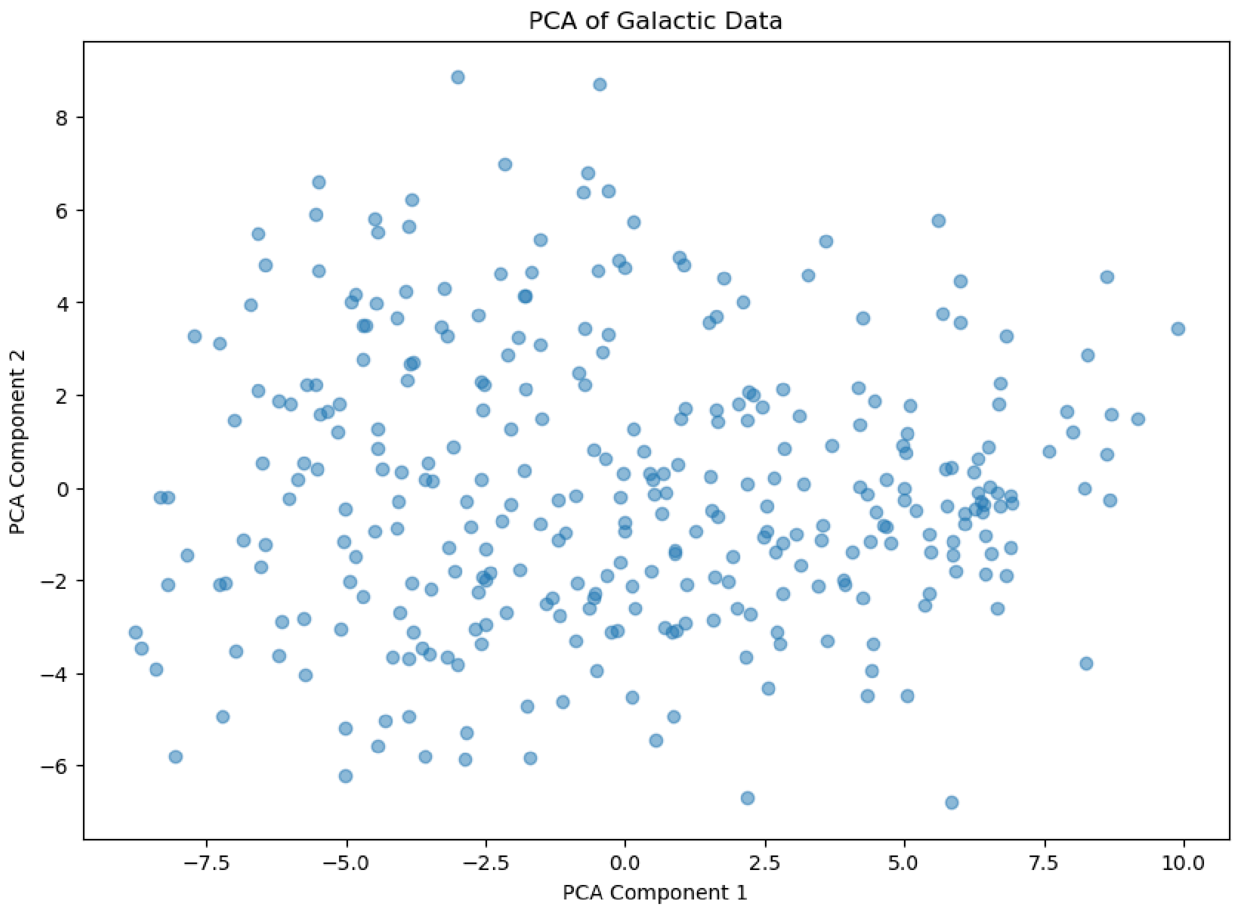

Correlation Matrix
Nr
e.Rmag
mumax
e.Mcz
chi2red
e.UjMAG
e.BjMAG
e.VjMAG
e.usMAG
e.gsMAG
e.rsMAG
e.UbMAG
e.BbMAG
e.VbMAG
e.S280MA
e.W420FE
e.W462FE
e.W485FD
e.W518FE
e.W571FS
e.W604FE
e.W646FD
e.W696FE
e.W753FE
e.W815FS
e.W856FD
e.W914FD
e.W914FE
e.UFS
e.BFS
e.VFD
e.RFS
e.IFD
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75


Distribution of Rmag
Count
Rmag
0
10
20
30
40
50
−2
−1
0
1
2


Distribution of Mcz
Count
Mcz
0
10
20
30
40
50
60
70
−3
−2
−1
0
1
2


Distribution of mumax
Count
mumax
0
10
20
30
40
50
−2
−1
0
1
2

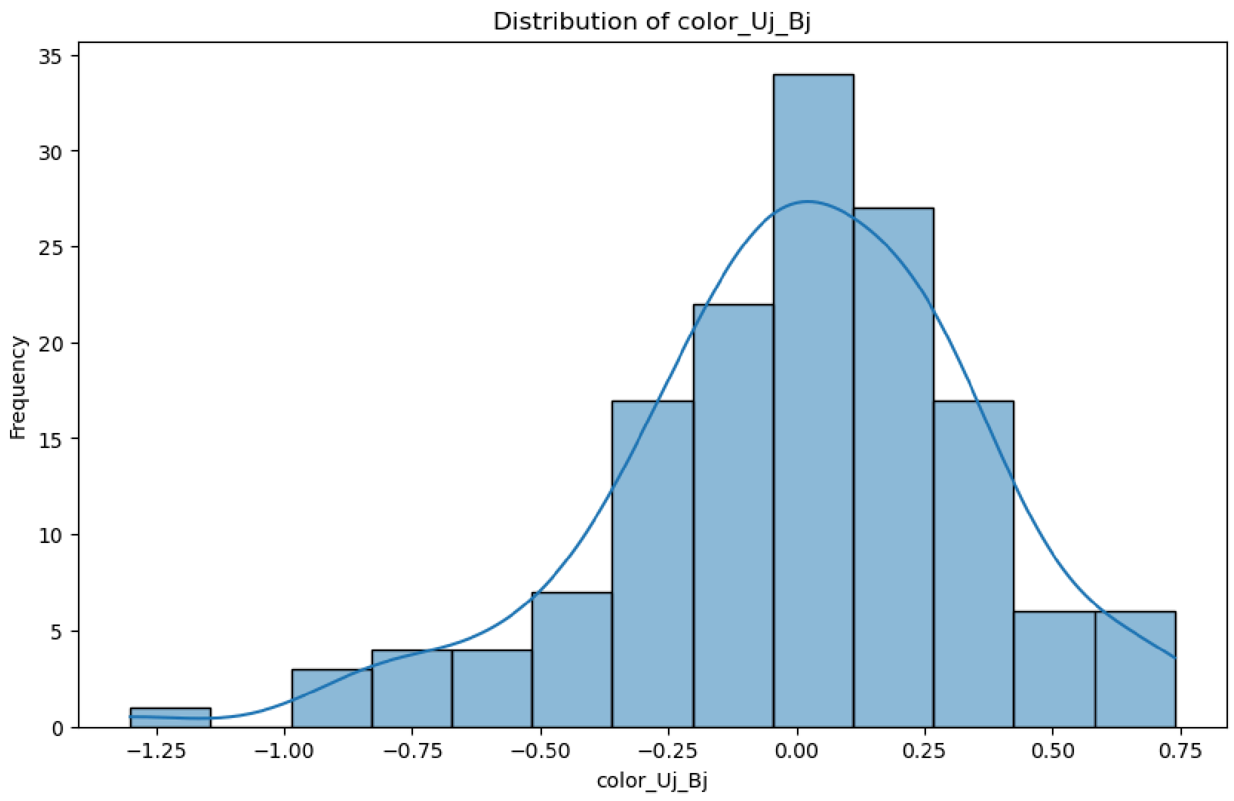

Distribution of UjMAG
Count
50
40
30
20
10
0
−2
−1
0
1
2
3
UjMAG

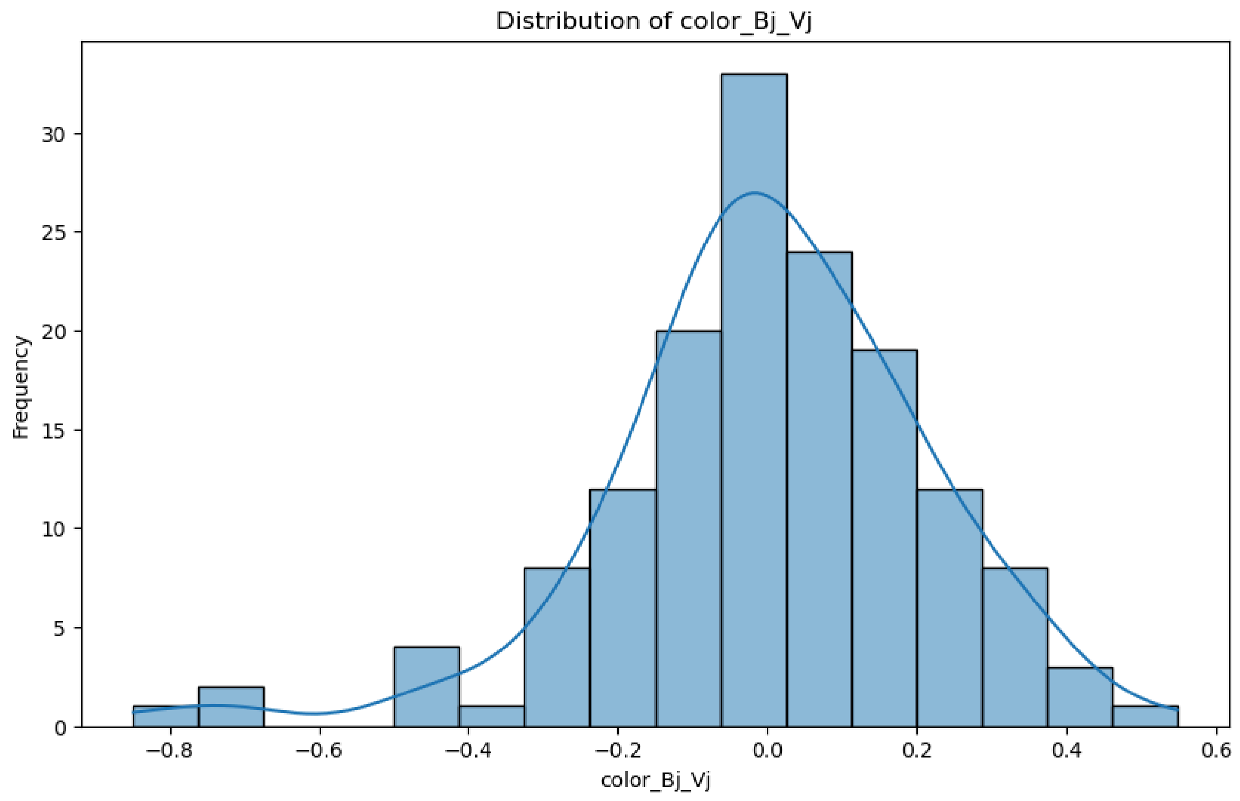


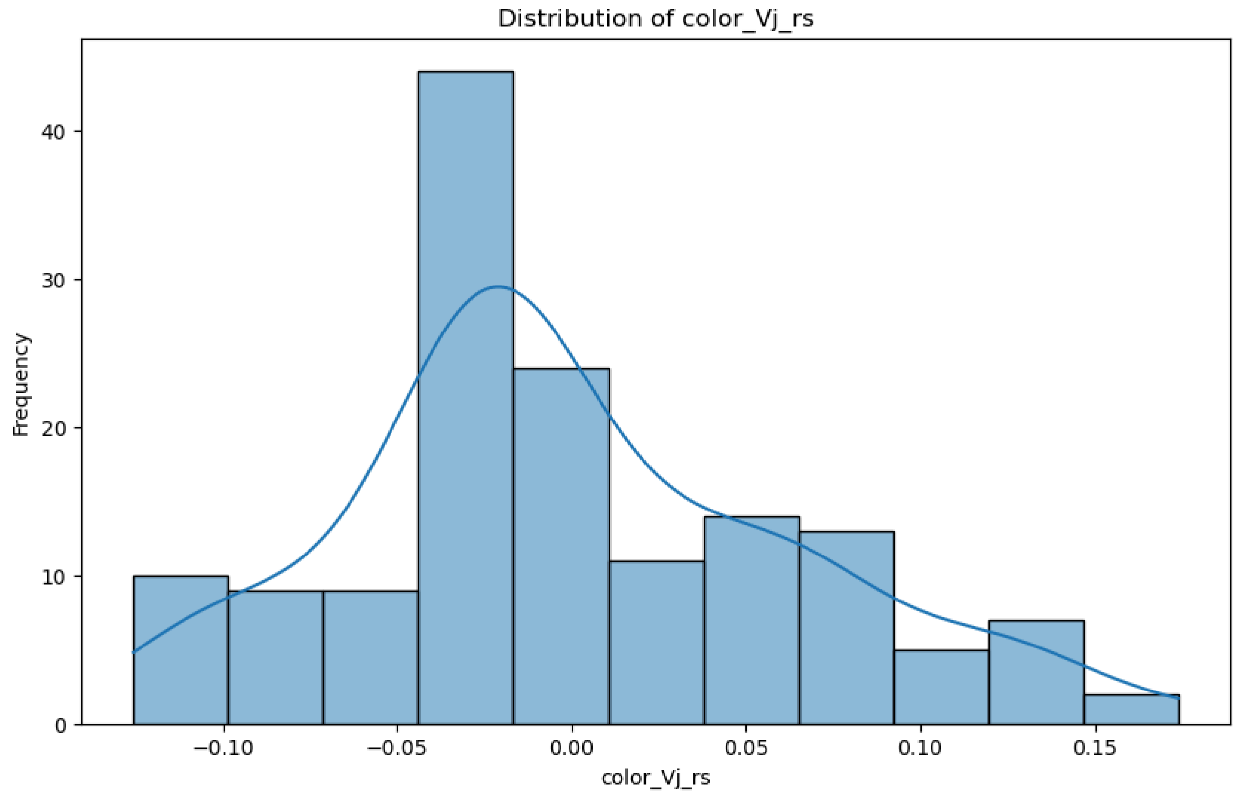
In [14]:

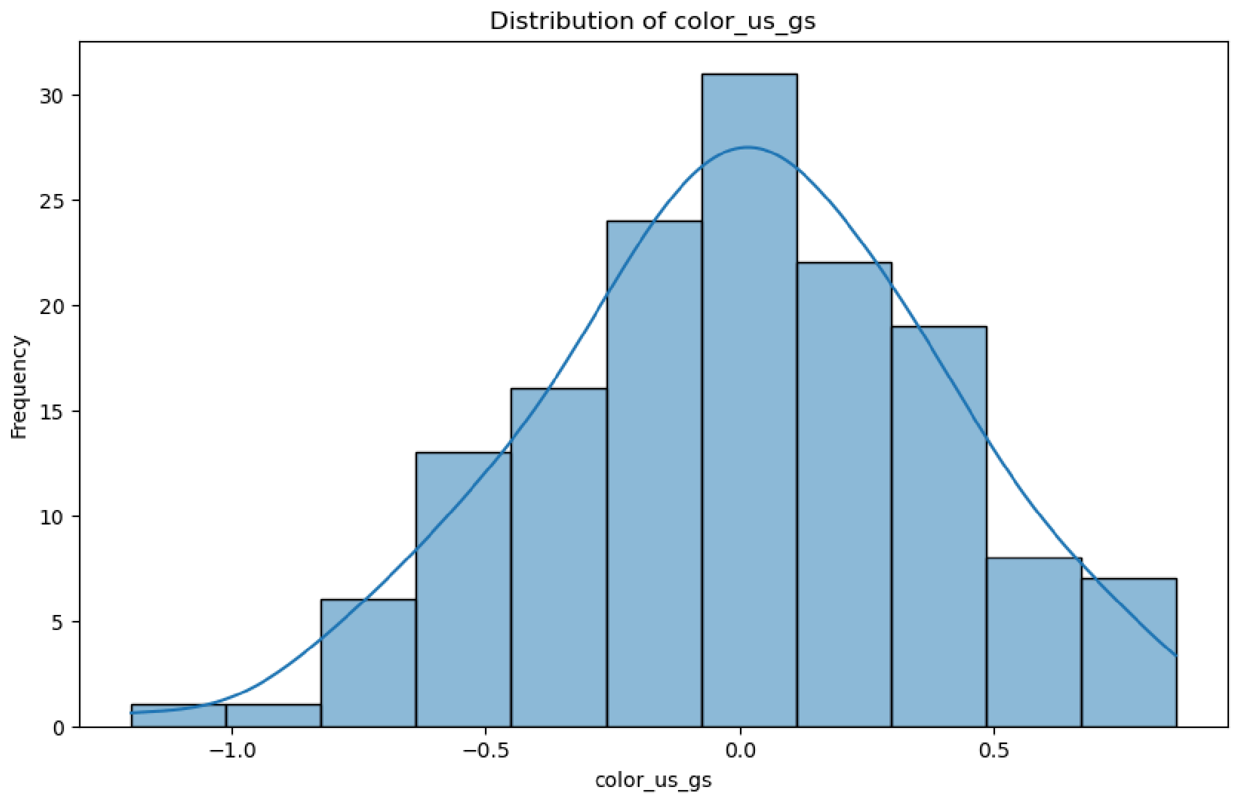
```
# Feature Selection

data = pd.read_csv(file_path)

features_to_drop = ['Nr']
data_features = data_normalized.drop(columns=features_to_drop)

# Create new features

data_features['color_Uj_Bj'] = data_features['UjMAG'] - data_features['BjMAG']
data_features['color_Bj_Vj'] = data_features['BjMAG'] - data_features['VjMAG']
data_features['color_Vj_rs'] = data_features['VjMAG'] - data_features['rsMAG']
data_features['color_us_gs'] = data_features['usMAG'] - data_features['gsMAG']
data_features['color_gs_rs'] = data_features['gsMAG'] - data_features['rsMAG']
data_features['color_rs_Ub'] = data_features['rsMAG'] - data_features['UbMAG']

#Brightness ratios
data_features['brightness_ratio_Uj_Bj'] = data_features['UjMAG'] / (data_features['BjMAG'] +
1e-10)
```

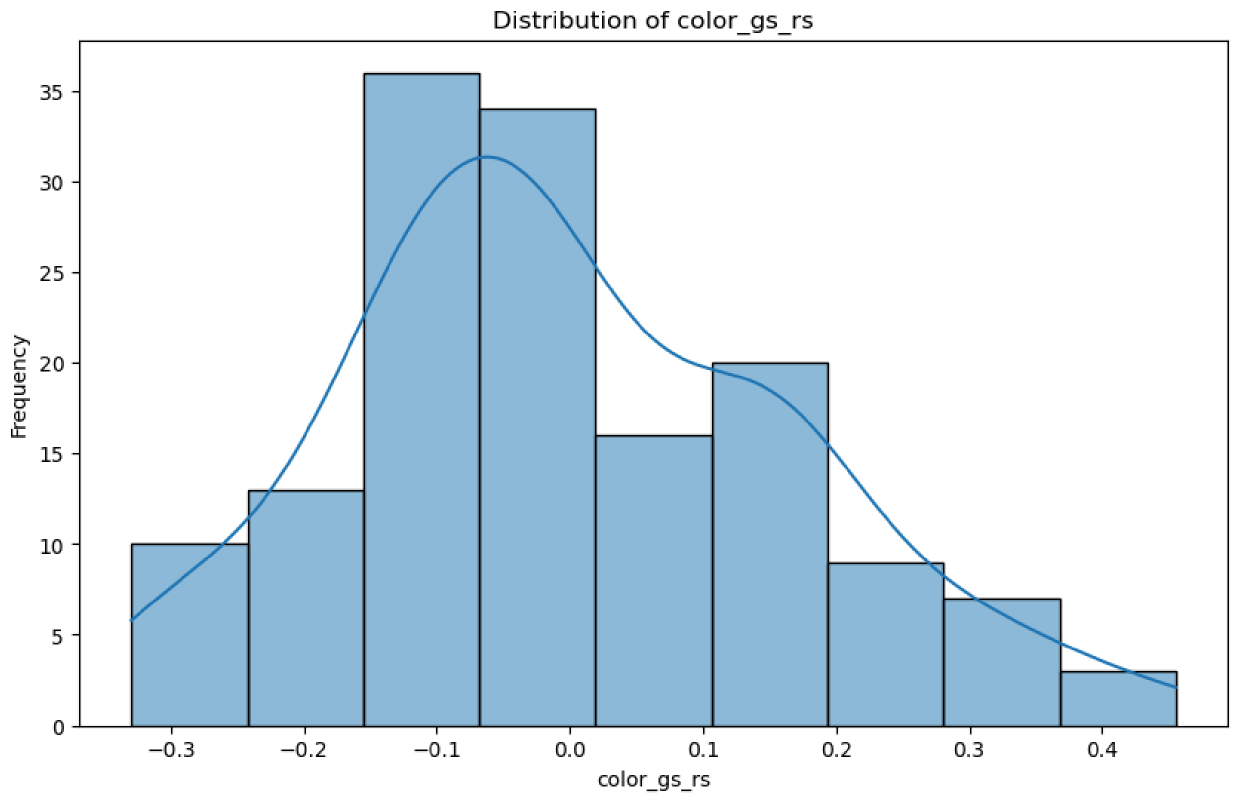
```
data_features['brightness_ratio_Bj_Vj'] = data_features['BjMAG'] / (data_features['VjMAG'] +
1e-10)
data_features['brightness_ratio_Vj_rs'] = data_features['VjMAG'] / (data_features['rsMAG'] +
1e-10)

#Surface brightness to magnitude ratio
data_features['surface_brightness_ratio'] = data_features['mumax'] / (data_features['Rmag'] +
1e-10)

#Central surface brightness normalized
data_features['central_brightness_normalized'] = data_features['mumax'] /
(data_features['mumax'].mean() + 1e-10)

#Redshift-normalized brightness
data_features['redshift_brightness_ratio'] = data_features['Mcz'] / (data_features['Rmag'] +
1e-10)

#Visualize the new features
new_features = [
   'color_Uj_Bj', 'color_Bj_Vj', 'color_Vj_rs', 'color_us_gs', 'color_gs_rs', 'color_rs_Ub',
   'brightness_ratio_Uj_Bj', 'brightness_ratio_Bj_Vj', 'brightness_ratio_Vj_rs',
   'surface_brightness_ratio', 'central_brightness_normalized', 'redshift_brightness_ratio'
]

for feature in new_features:
   plt.figure(figsize=(10, 6))
   sns.histplot(data_features[feature], kde=True)
   plt.title(f'Distribution of {feature}')
   plt.xlabel(feature)
   plt.ylabel('Frequency')
   plt.show()

#Visualize relationships
plt.figure(figsize=(10, 6))
sns.scatterplot(x='color_Uj_Bj', y='color_Bj_Vj', data=data_features)
plt.title('Scatter plot between color_Uj_Bj and color_Bj_Vj')
plt.xlabel('color_Uj_Bj')
plt.ylabel('color_Bj_Vj')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='brightness_ratio_Uj_Bj', y='brightness_ratio_Bj_Vj', data=data_features)
plt.title('Scatter plot between brightness_ratio_Uj_Bj and brightness_ratio_Bj_Vj')
plt.xlabel('brightness_ratio_Uj_Bj')
plt.ylabel('brightness_ratio_Bj_Vj')
plt.show()

#PCA
pca = PCA(n_components=10)
pca_components = pca.fit_transform(data_features)
pca_df = pd.DataFrame(pca_components, columns=[f'PCA_{i+1}' for i in
range(pca.n_components_)])
data_final = pd.concat([data_normalized.reset_index(drop=True),
pca_df.reset_index(drop=True)], axis=1)

data_final.to_csv('/Users/cameronsouza/Desktop/COMBO17.csv', index=False)
```

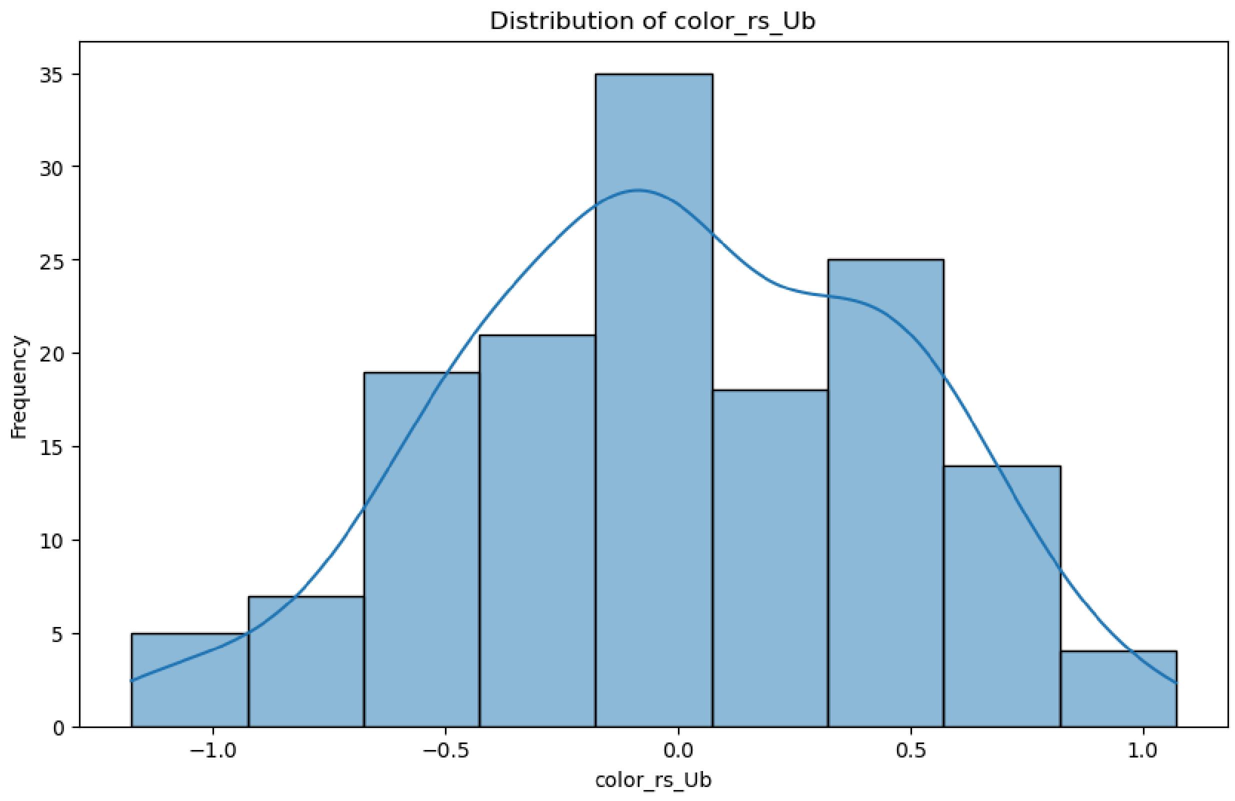


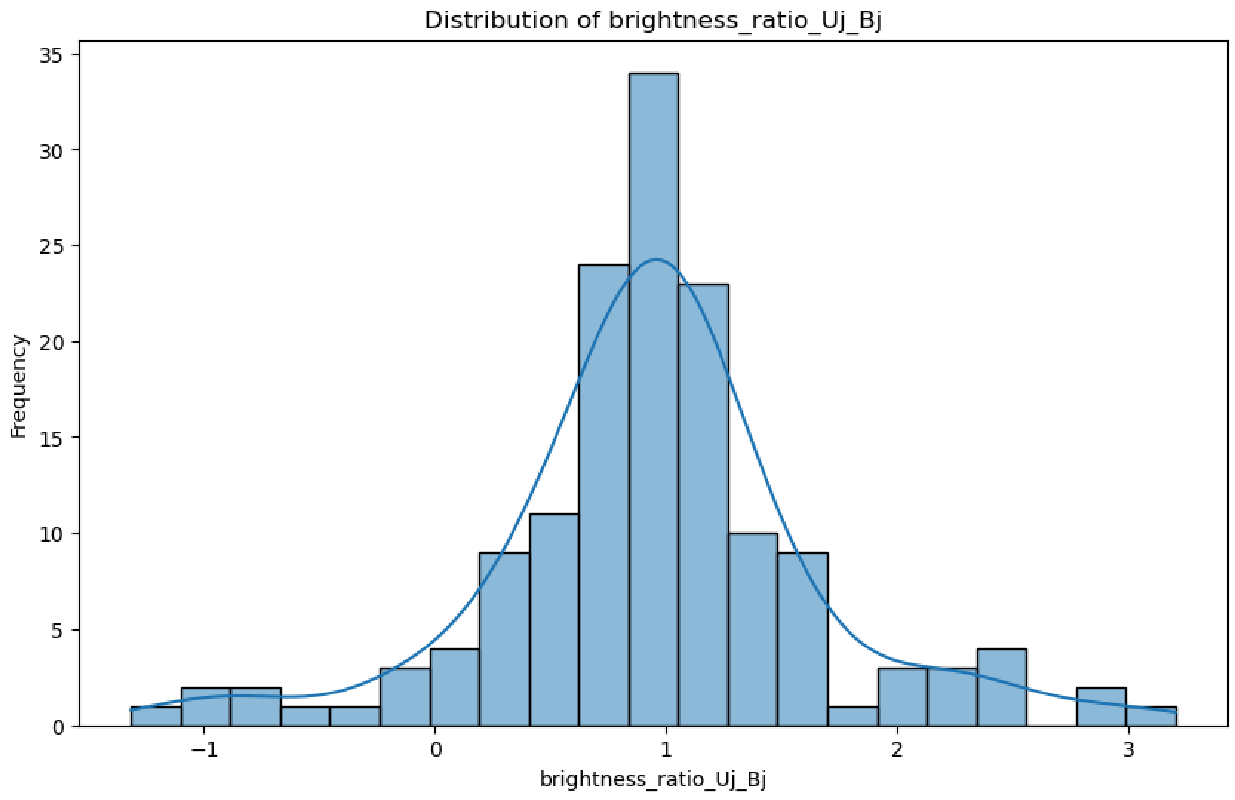

Distribution of color_Bj_Vj
Frequency
color_Bj_Vj
0
5
10
15
20
25
30
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

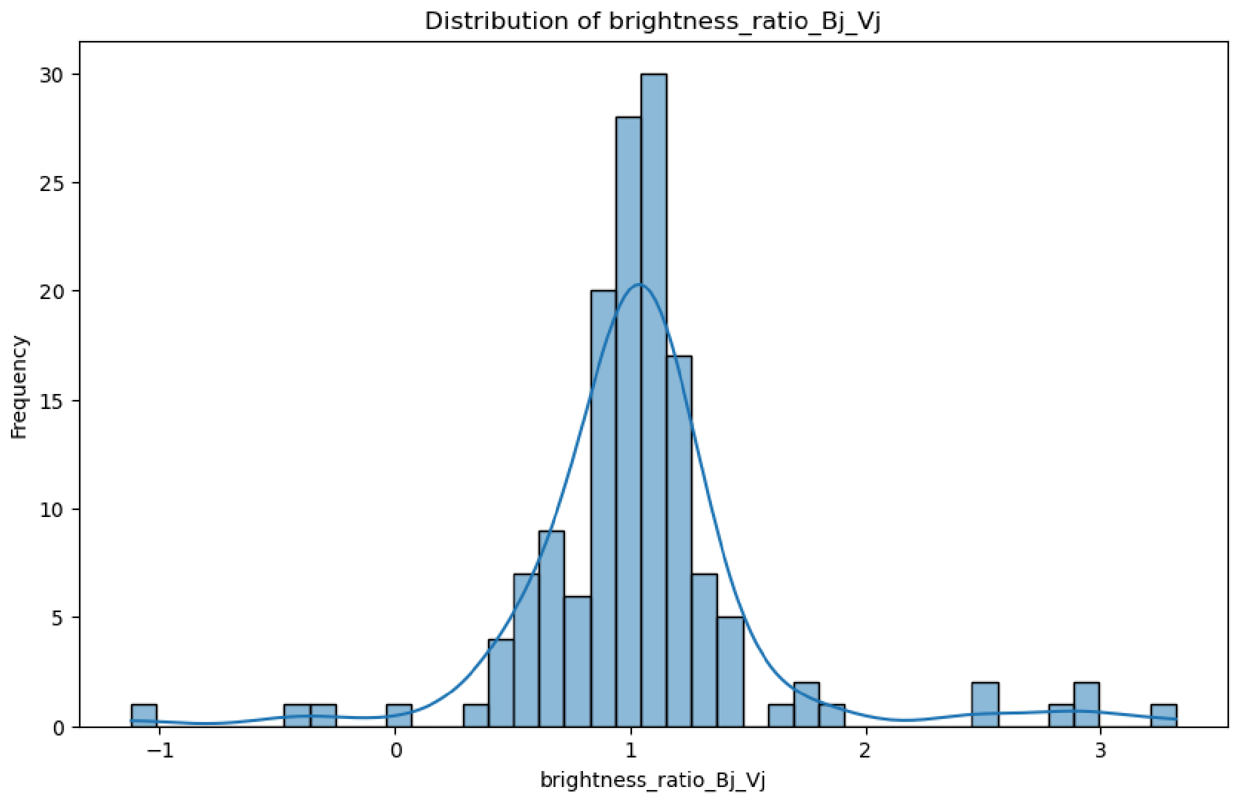

Distribution of color_Vj_rs
Frequency
color_Vj_rs
0
10
20
30
40
−0.10
−0.05
0.00
0.05
0.10
0.15

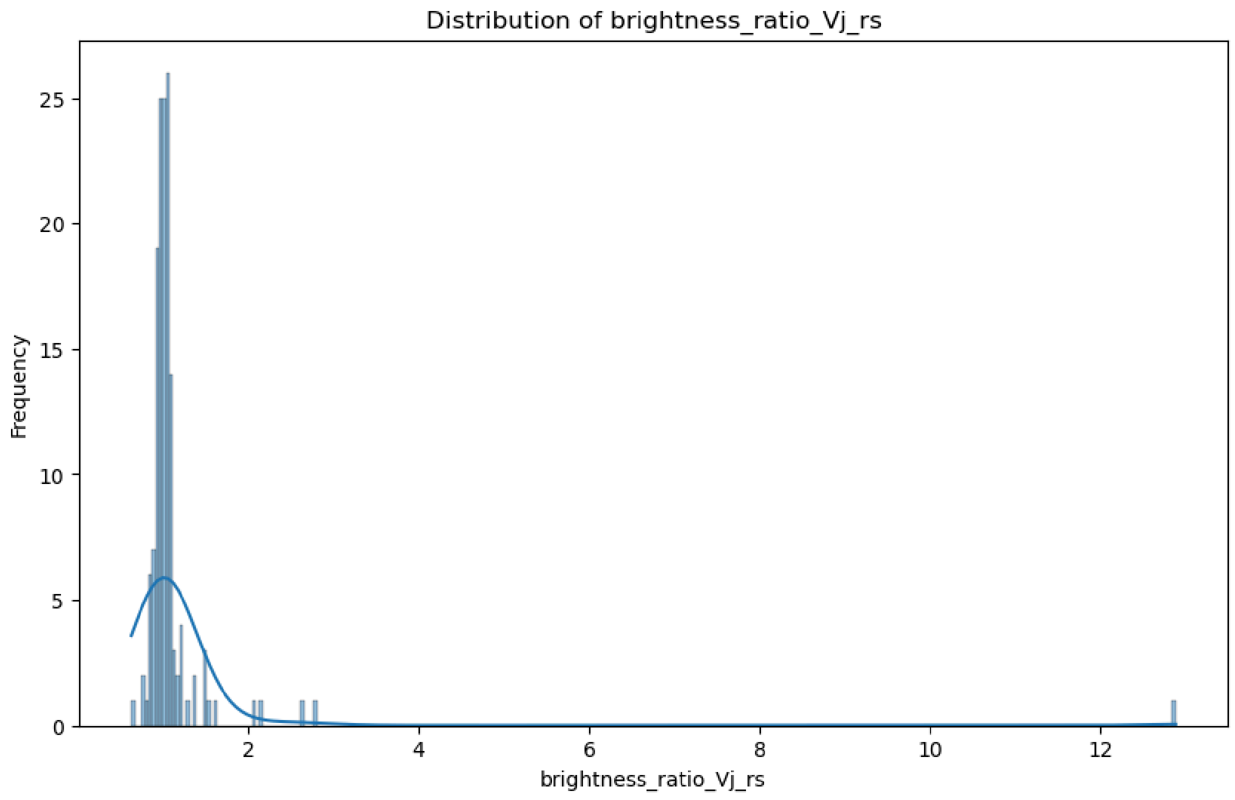

Distribution of color_us_gs
Frequency
color_us_gs
0
5
10
15
20
25
30
−1.0
−0.5
0.0
0.5

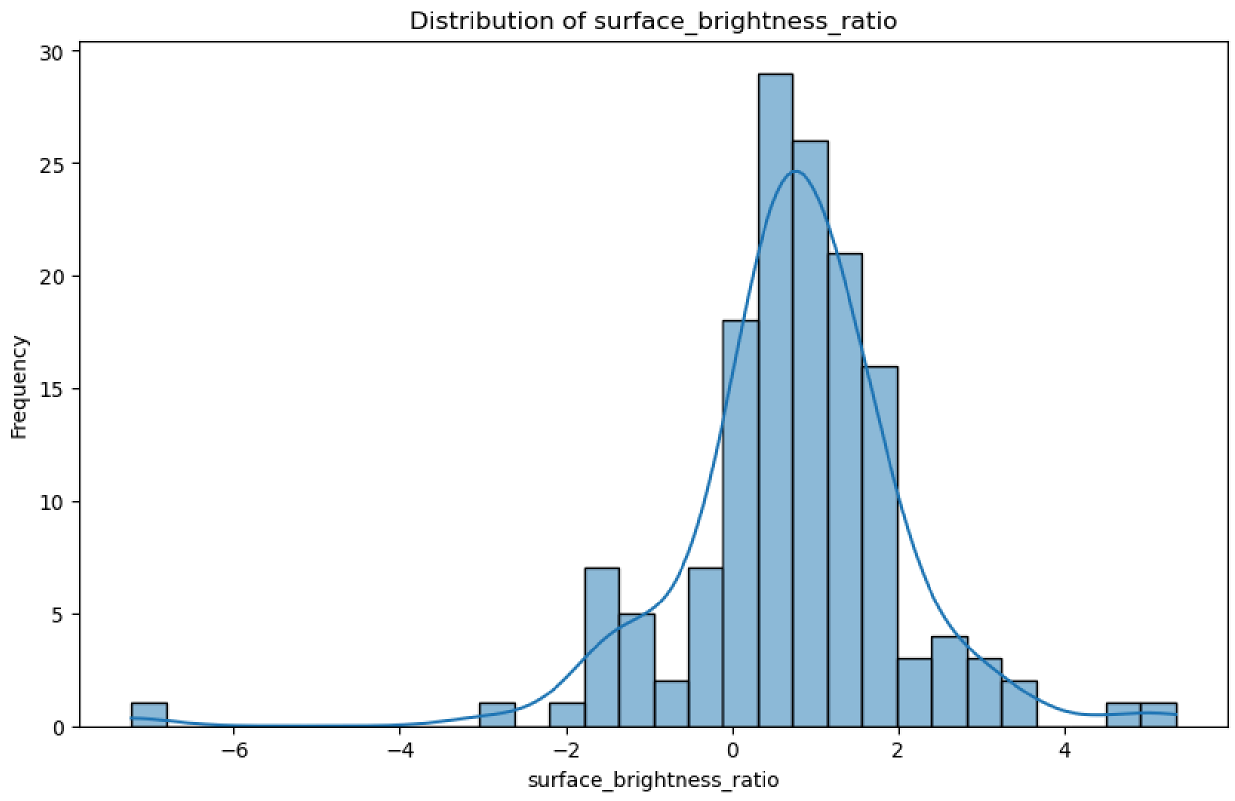

Distribution of color_gs_rs
Frequency
color_gs_rs
0
5
10
15
20
25
30
35
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4

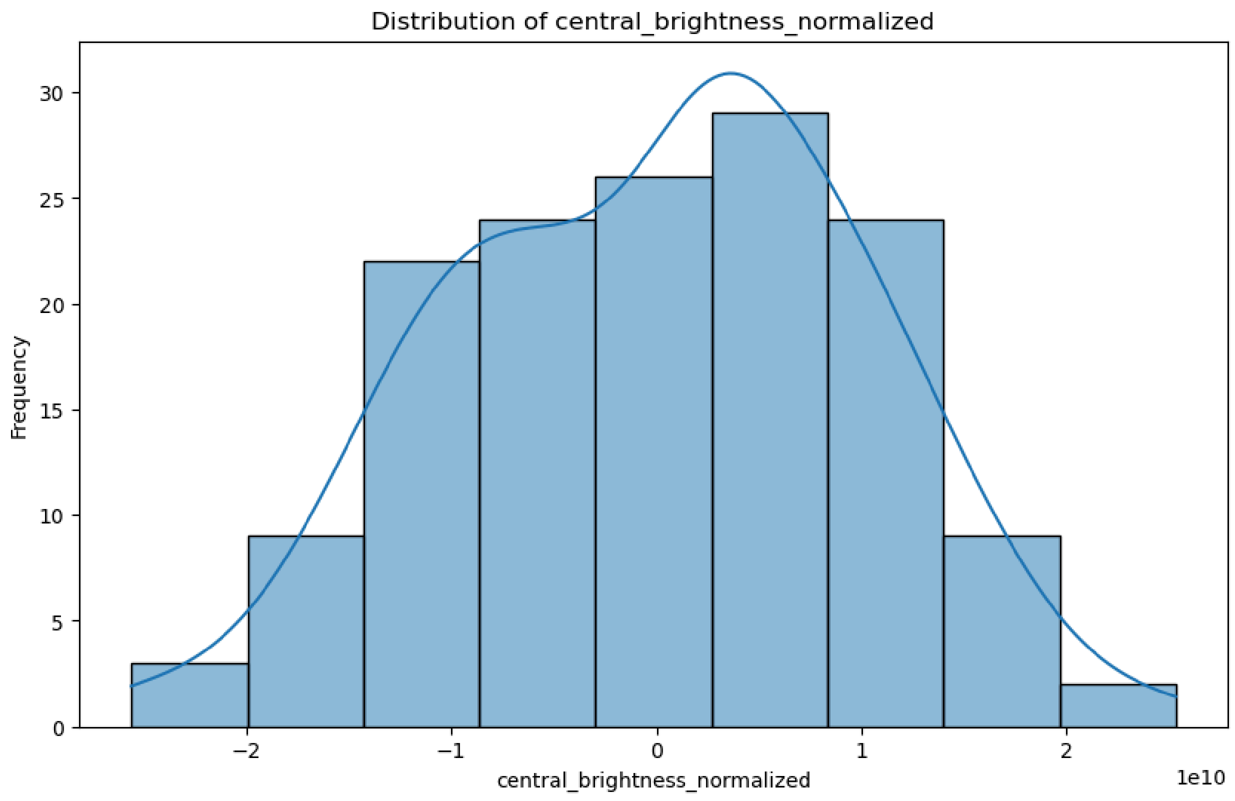

Distribution of color_rs_Ub
Frequency
color_rs_Ub
0
5
10
15
20
25
30
35
−1.0
−0.5
0.0
0.5
1.0

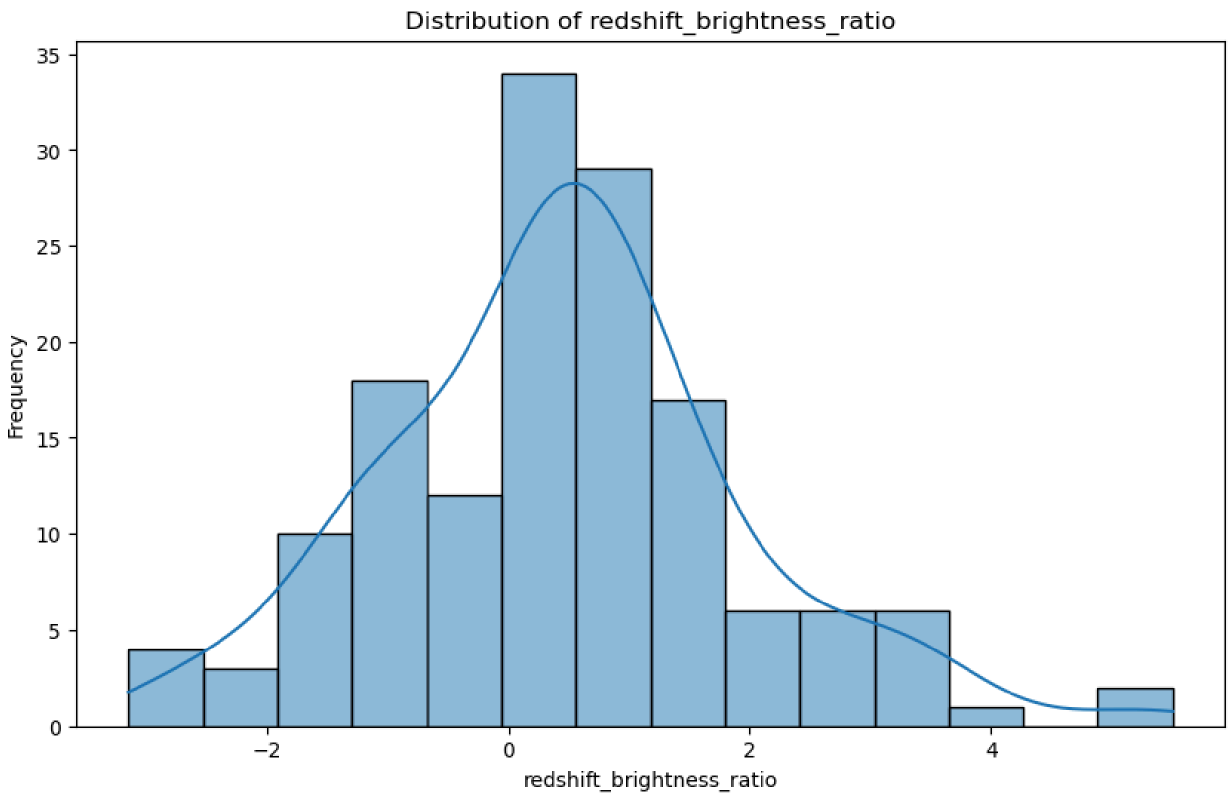

Distribution of brightness_ratio_Uj_Bj
Frequency
brightness_ratio_Uj_Bj
0
5
10
15
20
25
30
35
−1
0
1
2
3

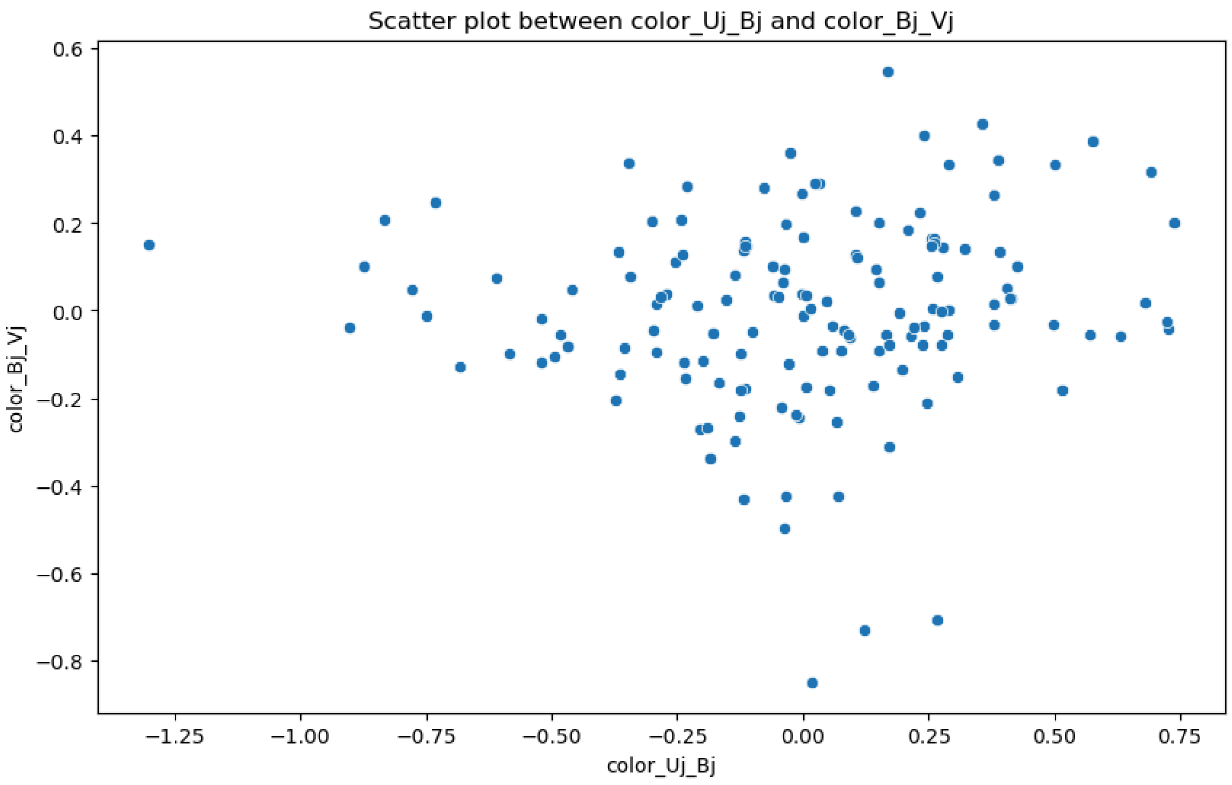

Distribution of brightness_ratio_Bj_Vj
Frequency
brightness_ratio_Bj_Vj
0
5
10
15
20
25
30
−1
0
1
2
3

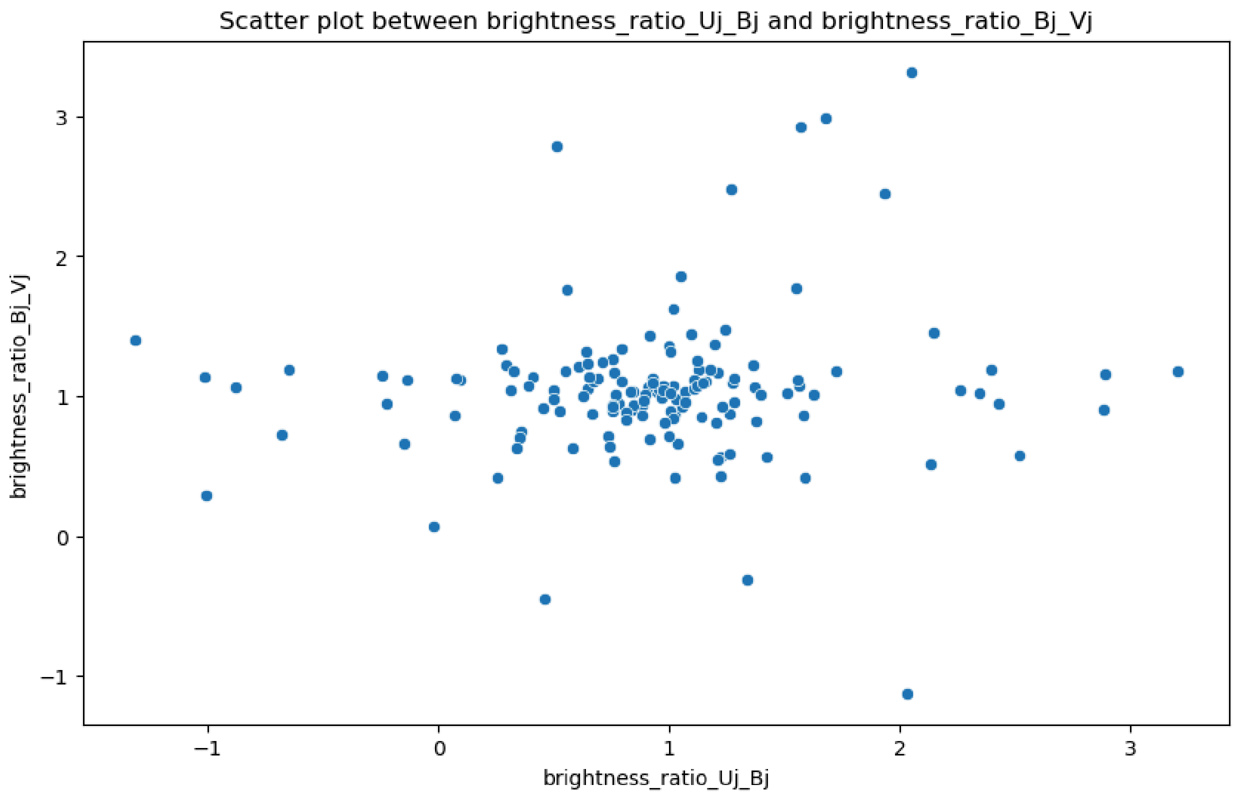

Distribution of brightness_ratio_Vj_rs
Frequency
brightness_ratio_Vj_rs
0
5
10
15
20
25
2
4
6
8
10
12


Distribution of surface_brightness_ratio
Frequency
0
5
10
15
20
25
30
−6
−4
−2
0
2
4
surface_brightness_ratio


Distribution of central_brightness_normalized
Frequency
0
5
10
15
20
25
30
−2
−1
0
1
2
1e10
central_brightness_normalized

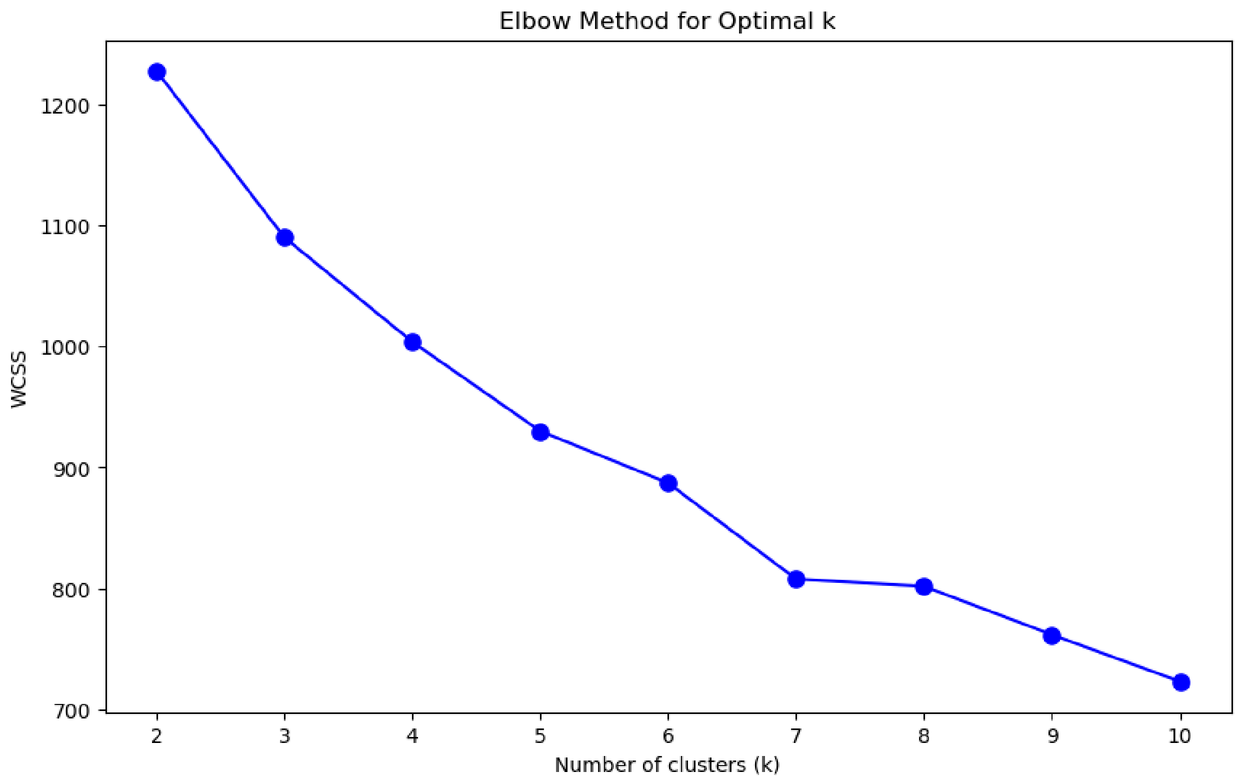

Distribution of redshift_brightness_ratio
Frequency
0
5
10
15
20
25
30
35
−2
0
2
4
redshift_brightness_ratio


Scatter plot between color_Uj_Bj and color_Bj_Vj
color_Bj_Vj
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.25
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
color_Uj_Bj

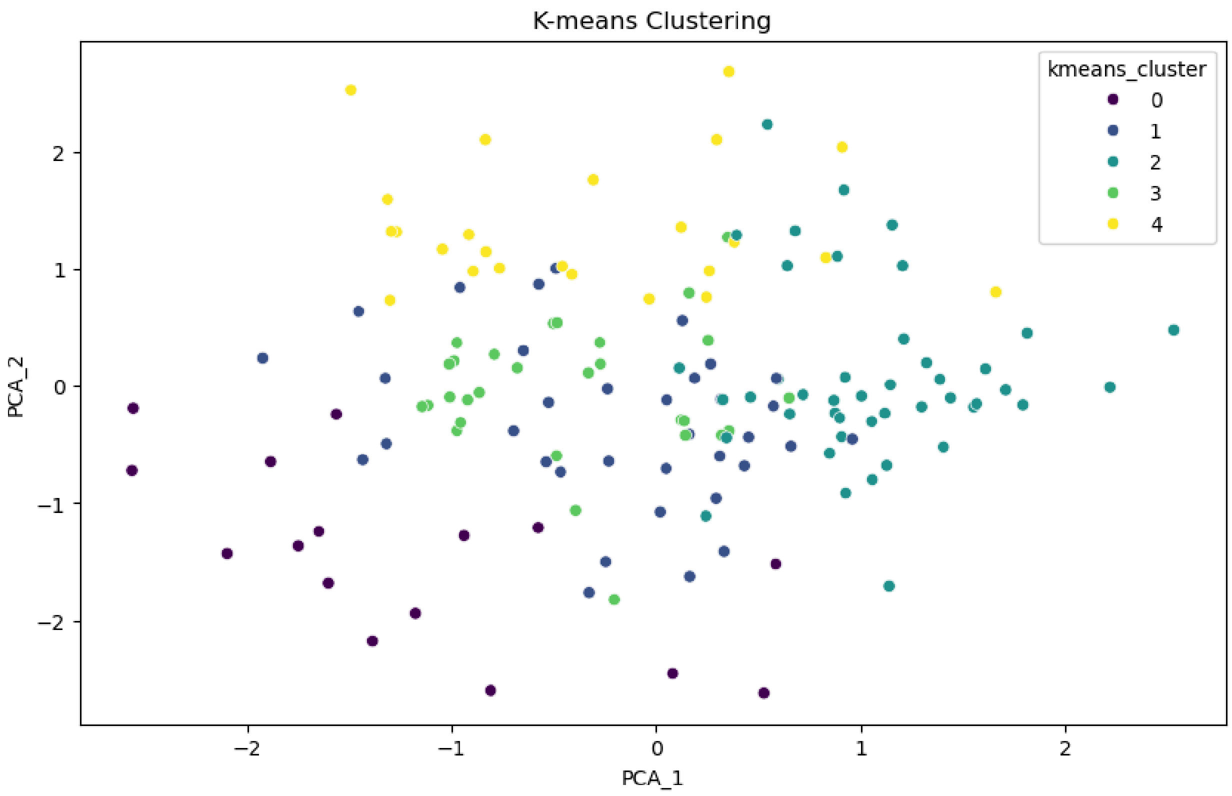


In [21]:

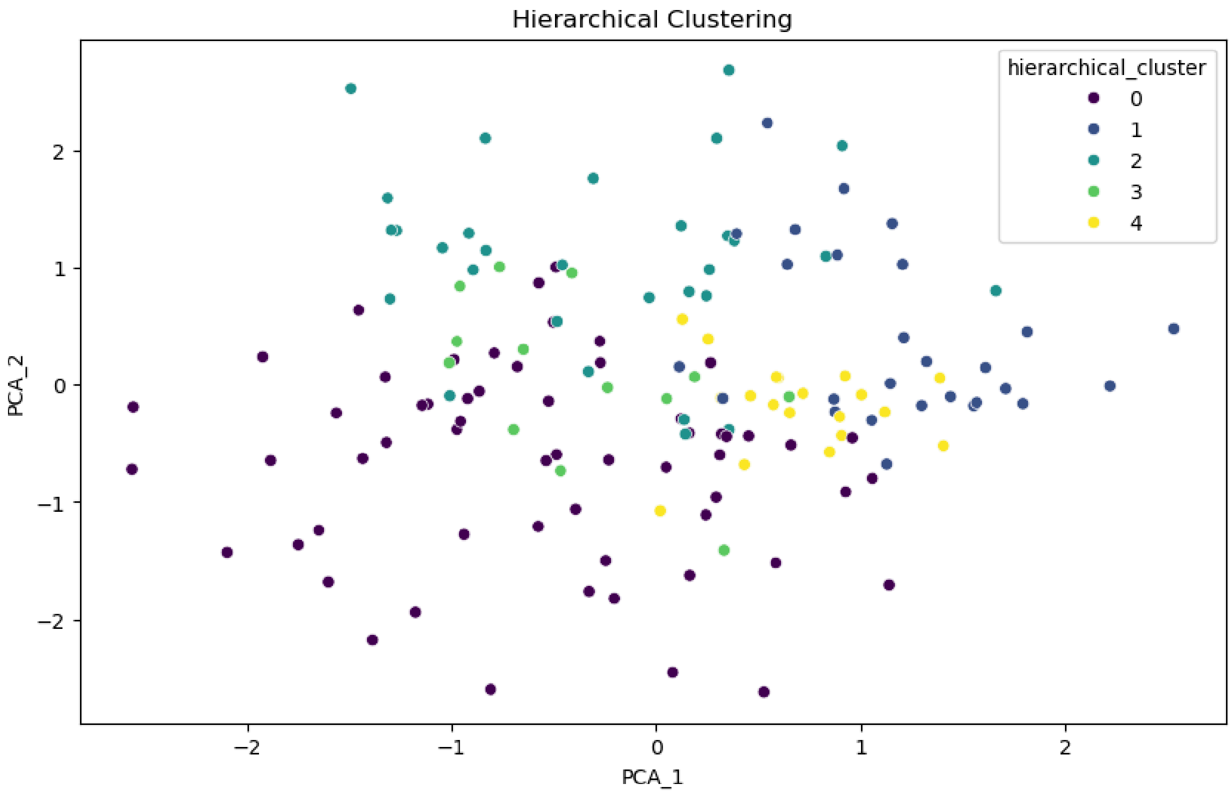
```
#Clustering

data = pd.read_csv(file_path)
features = data.drop(columns=[f'PCA_{i+1}' for i in range(10)])

#K-means
features = data[[f'PCA_{i+1}' for i in range(10)]]

#Determine the optimal number of clusters using the Elbow Method
wcss = []
silhouette_scores = []
K = range(2, 11)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(features)
    wcss.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(features, kmeans.labels_))

#Plot the Elbow Method
plt.figure(figsize=(10, 6))
plt.plot(K, wcss, 'bo-', markersize=8)
plt.xlabel('Number of clusters (k)')
plt.ylabel('WCSS')
plt.title('Elbow Method for Optimal k')
plt.show()

kmeans = KMeans(n_clusters=5, random_state=42)
kmeans_labels = kmeans.fit_predict(features)
data['kmeans_cluster'] = kmeans_labels
kmeans_silhouette = silhouette_score(features, kmeans_labels)

#Hierarchical Clustering
hierarchical = AgglomerativeClustering(n_clusters=5)
hierarchical_labels = hierarchical.fit_predict(features)
data['hierarchical_cluster'] = hierarchical_labels
hierarchical_silhouette = silhouette_score(features, hierarchical_labels)

#Spectral Clustering
spectral = SpectralClustering(n_clusters=5, random_state=42, affinity='nearest_neighbors')  #
Adjust the number of clusters as needed
spectral_labels = spectral.fit_predict(features)
data['spectral_cluster'] = spectral_labels
spectral_silhouette = silhouette_score(features, spectral_labels)

#silhouette scores
print(f"K-means Silhouette Score: {kmeans_silhouette}")
print(f"Hierarchical Clustering Silhouette Score: {hierarchical_silhouette}")
print(f"Spectral Clustering Silhouette Score: {spectral_silhouette}")

#Visualize the clustering results
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PCA_1', y='PCA_2', hue='kmeans_cluster', data=data, palette='viridis')
plt.title('K-means Clustering')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='PCA_1', y='PCA_2', hue='hierarchical_cluster', data=data, palette='viridis')
plt.title('Hierarchical Clustering')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='PCA_1', y='PCA_2', hue='spectral_cluster', data=data, palette='viridis')
plt.title('Spectral Clustering')
plt.show()
```



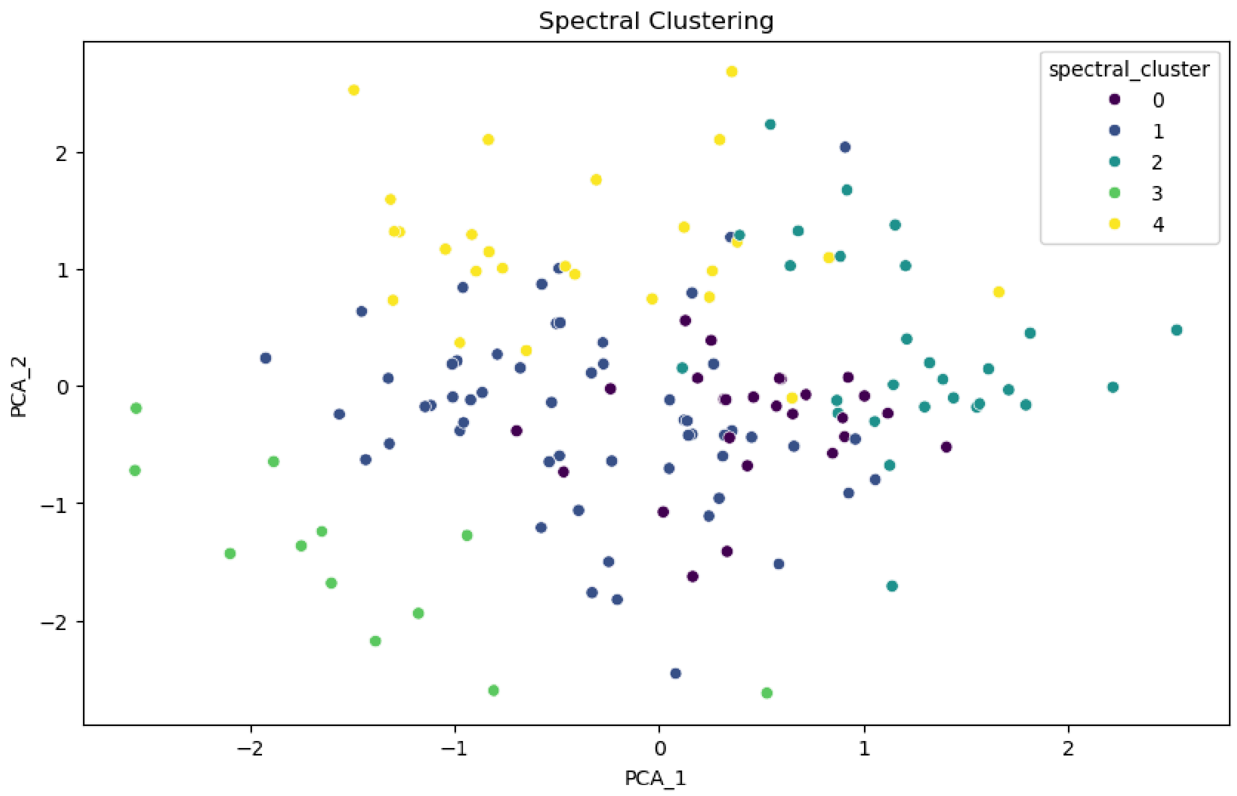
```
K-means Silhouette Score: 0.1504098540782459
Hierarchical Clustering Silhouette Score: 0.11491170716351616
Spectral Clustering Silhouette Score: 0.12978258837807546
```



K-means Clustering
kmeans_cluster
0
1
2
3
4
PCA_1
PCA_2
−2
−1
0
1
2



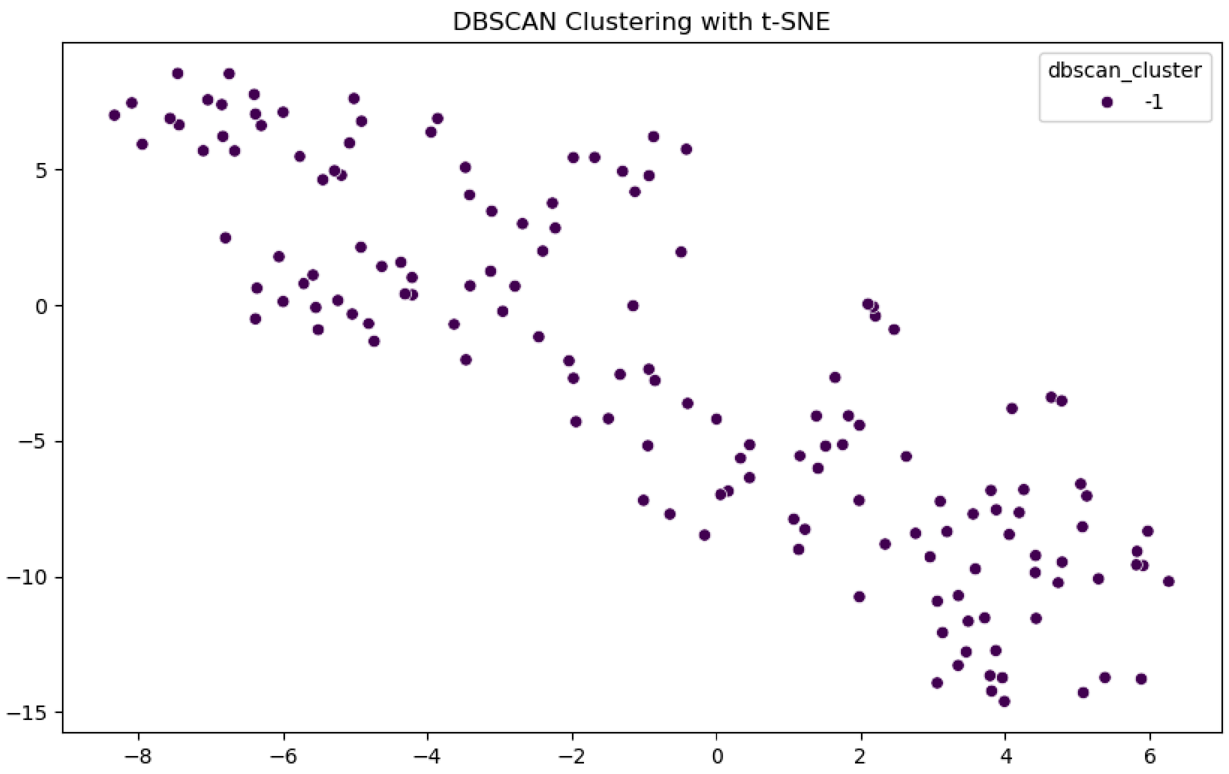
Hierarchical Clustering
hierarchical_cluster
0
1
2
3
4
PCA_1
PCA_2
−2
−1
0
1
2



In [31]:

```
data = pd.read_csv(file_path)
data_numeric = data.apply(pd.to_numeric, errors='coerce')
data_numeric = data_numeric.fillna(data_numeric.mean())

# Additional Feature Engineering
data_numeric['log_Rmag'] = np.log1p(data_numeric['Rmag'].replace({0: np.nan}).dropna())
data_numeric['log_mumax'] = np.log1p(data_numeric['mumax'].replace({0: np.nan}).dropna())
data_numeric['log_Mcz'] = np.log1p(data_numeric['Mcz'].replace({0: np.nan}).dropna())
features = data_numeric[['log_Rmag', 'log_mumax', 'log_Mcz', 'UjMAG', 'BjMAG', 'VjMAG',
'usMAG', 'gsMAG', 'rsMAG', 'UbMAG']]
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
features_scaled = pd.DataFrame(features_scaled, columns=features.columns).fillna(0)

#t-SNE for non-linear dimensionality reduction
tsne = TSNE(n_components=2, random_state=42)
tsne_components = tsne.fit_transform(features_scaled)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)  # Adjust eps and min_samples as needed
dbscan_labels = dbscan.fit_predict(features_scaled)
```

```
# Evaluate DBSCAN
dbscan_silhouette = silhouette_score(features_scaled, dbscan_labels) if
len(set(dbscan_labels)) > 1 else -1
data['dbscan_cluster'] = dbscan_labels

# Visualize the clustering results
plt.figure(figsize=(10, 6))
sns.scatterplot(x=tsne_components[:, 0], y=tsne_components[:, 1], hue='dbscan_cluster',
data=data, palette='viridis')
plt.title('DBSCAN Clustering with t-SNE')
plt.show()

#silhouette score for DBSCAN
print(f"DBSCAN Silhouette Score: {dbscan_silhouette}")

/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



```
DBSCAN Silhouette Score: -1
```

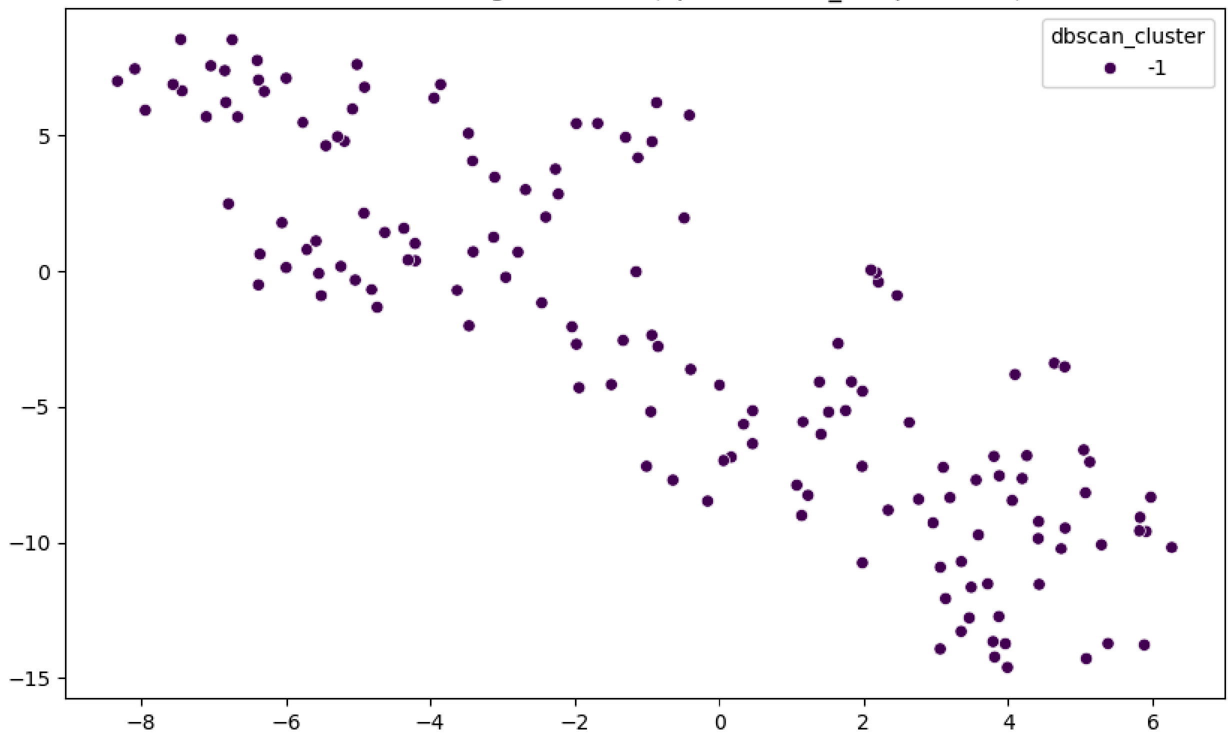
In [34]:

```
data = pd.read_csv(file_path)
data_numeric = data.apply(pd.to_numeric, errors='coerce')
data_numeric = data_numeric.fillna(data_numeric.mean())

# Additional Feature Engineering
data_numeric['log_Rmag'] = np.log1p(data_numeric['Rmag'].replace({0: np.nan}).dropna())
data_numeric['log_mumax'] = np.log1p(data_numeric['mumax'].replace({0: np.nan}).dropna())
data_numeric['log_Mcz'] = np.log1p(data_numeric['Mcz'].replace({0: np.nan}).dropna())
features = data_numeric[['log_Rmag', 'log_mumax', 'log_Mcz', 'UjMAG', 'BjMAG', 'VjMAG',
'usMAG', 'gsMAG', 'rsMAG', 'UbMAG']]

# Normalize
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
features_scaled = pd.DataFrame(features_scaled, columns=features.columns).fillna(0)

#t-SNE for non-linear dimensionality reduction
tsne = TSNE(n_components=2, random_state=42)
tsne_components = tsne.fit_transform(features_scaled)
```

```
#run DBSCAN
def run_dbscan(eps, min_samples):
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)
    dbscan_labels = dbscan.fit_predict(features_scaled)

    if len(set(dbscan_labels)) > 1:
        dbscan_silhouette = silhouette_score(features_scaled, dbscan_labels)
    else:
        dbscan_silhouette = -1

    data['dbscan_cluster'] = dbscan_labels

    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=tsne_components[:, 0], y=tsne_components[:, 1], hue='dbscan_cluster',
data=data, palette='viridis')
    plt.title(f'DBSCAN Clustering with t-SNE (eps={eps}, min_samples={min_samples})')
    plt.show()

    print(f"DBSCAN Silhouette Score (eps={eps}, min_samples={min_samples}):
{dbscan_silhouette}")

#different values for eps and min_samples
run_dbscan(eps=0.3, min_samples=100)
run_dbscan(eps=0.5, min_samples=100)
run_dbscan(eps=0.7, min_samples=100)
run_dbscan(eps=0.5, min_samples=100)
```

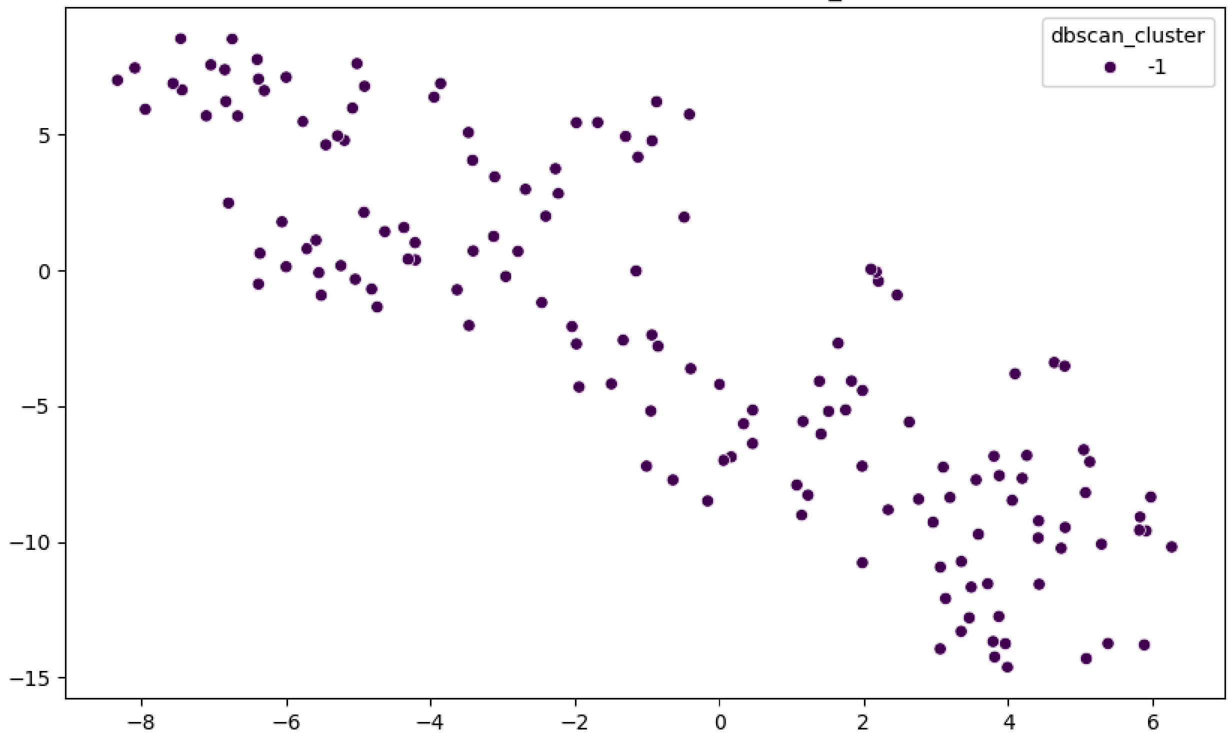
```
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

DBSCAN Clustering with t-SNE (eps=0.3, min_samples=100)

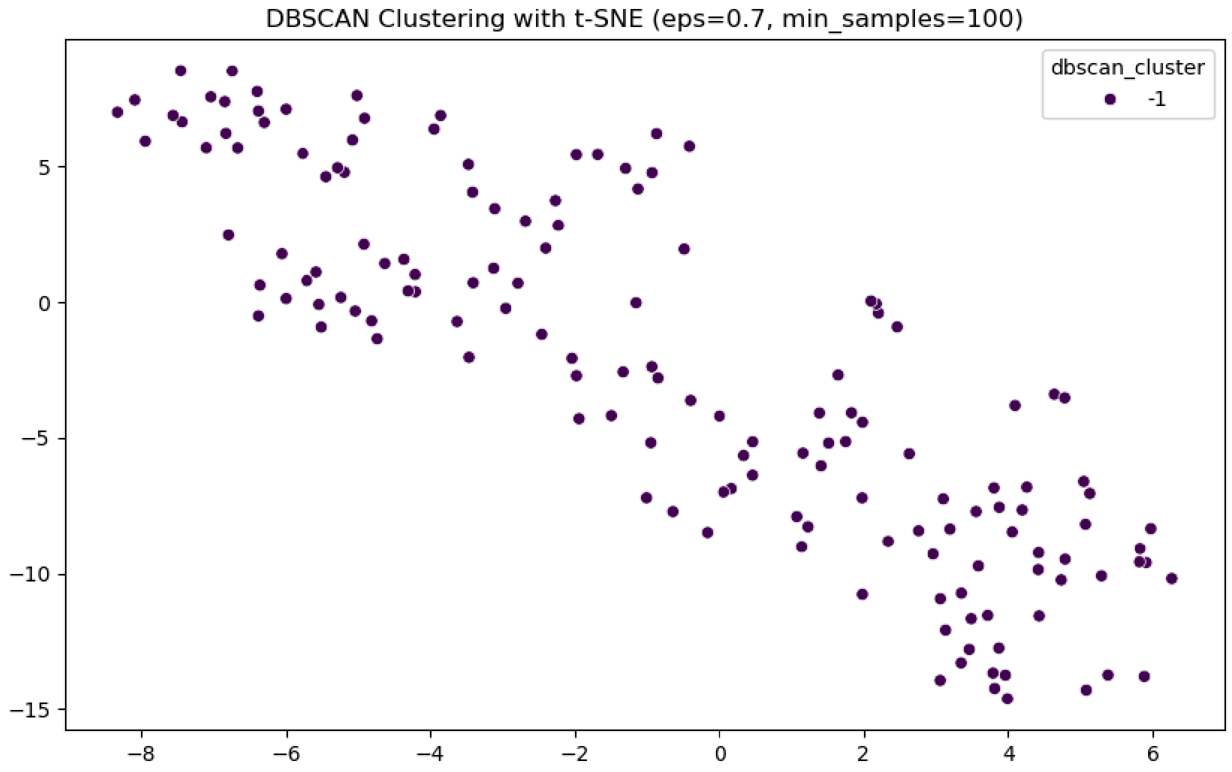


DBSCAN Silhouette Score (eps=0.3, min_samples=100): -1

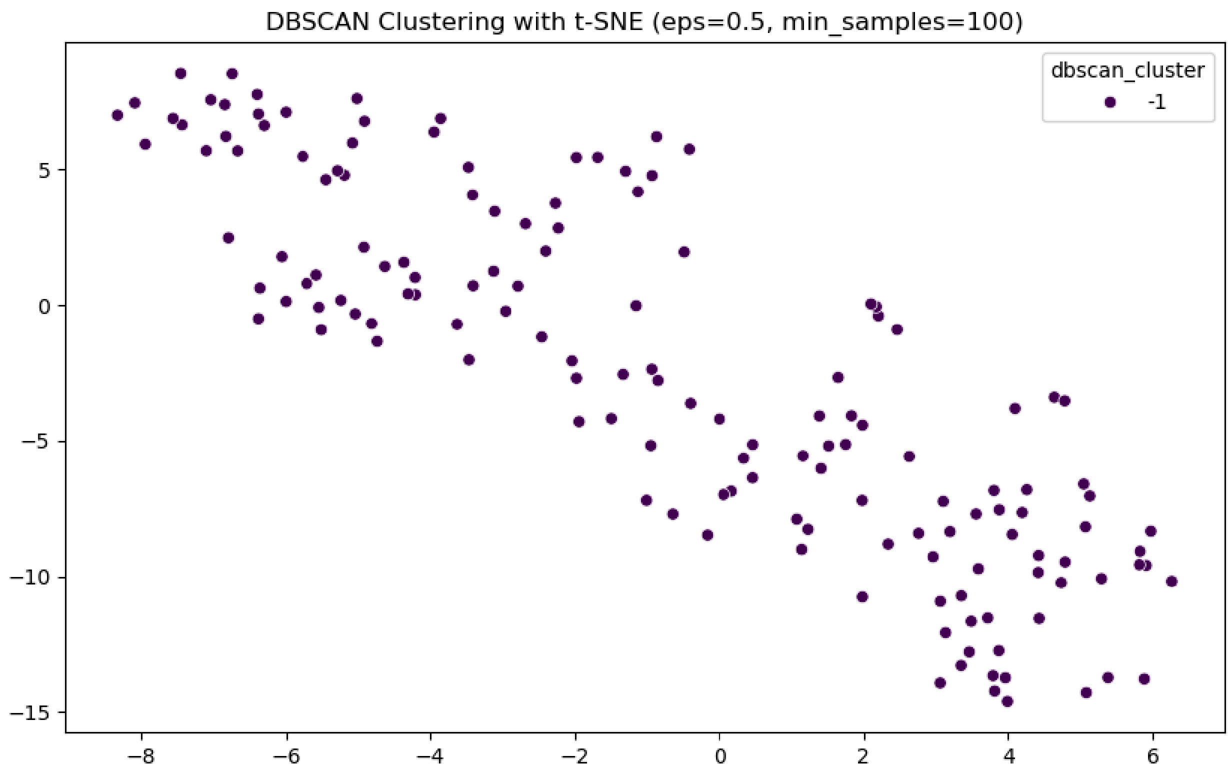
DBSCAN Clustering with t-SNE (eps=0.5, min_samples=100)



DBSCAN Silhouette Score (eps=0.5, min_samples=100): -1



DBSCAN Silhouette Score (eps=0.7, min_samples=100): -1



```
DBSCAN Silhouette Score (eps=0.5, min_samples=100): -1
```

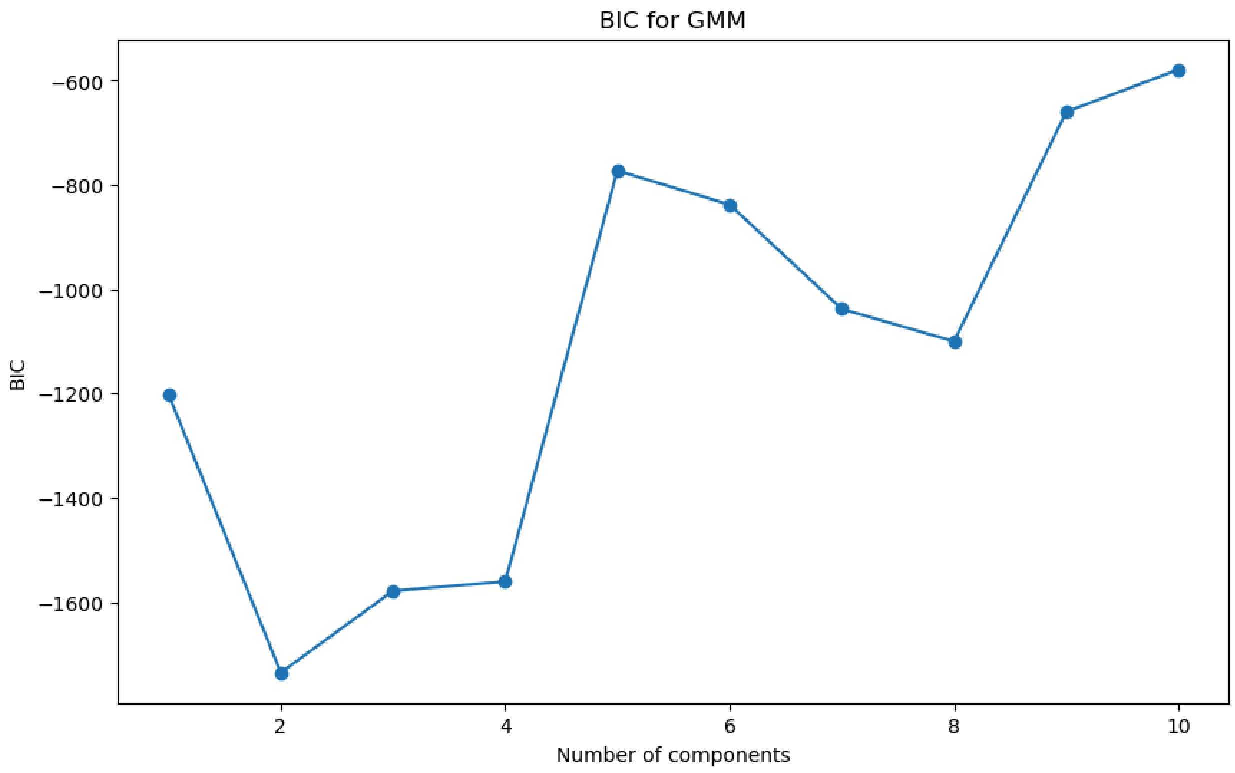
In [35]:

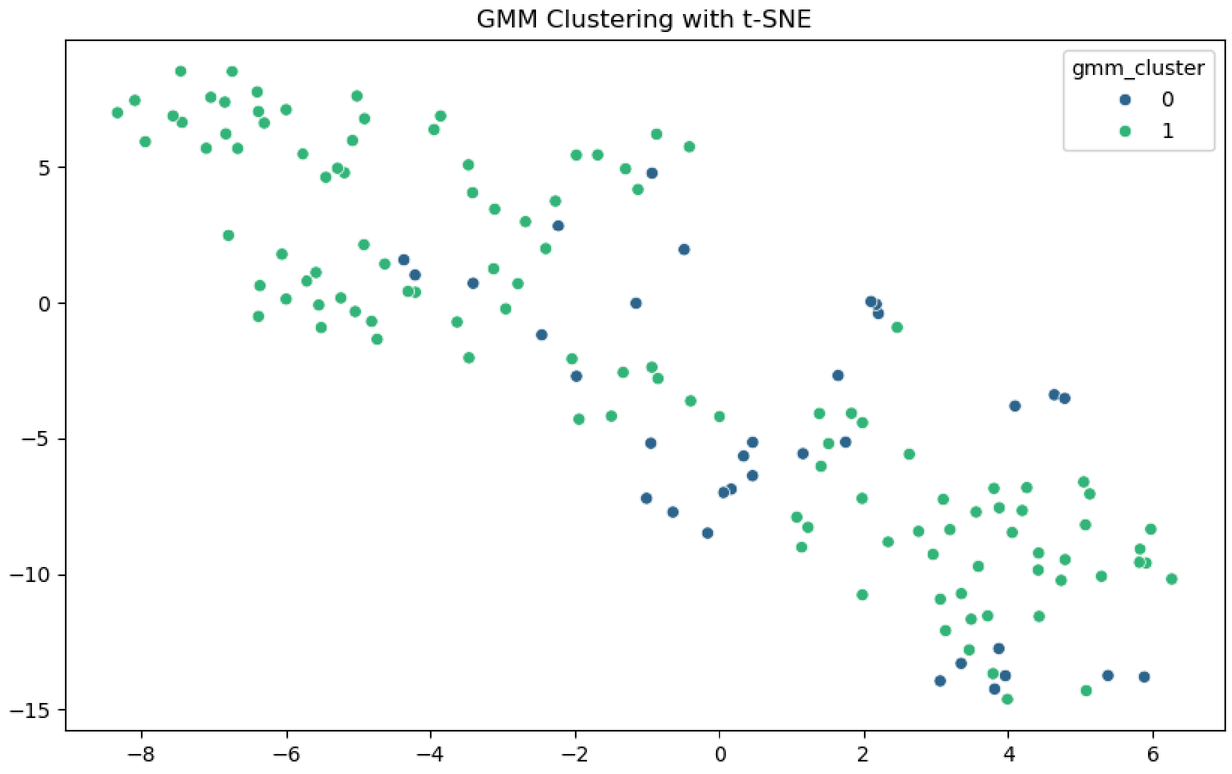
```
data = pd.read_csv(file_path)
data_numeric = data.apply(pd.to_numeric, errors='coerce')
data_numeric = data_numeric.fillna(data_numeric.mean())

# Additional Feature Engineering
data_numeric['log_Rmag'] = np.log1p(data_numeric['Rmag'].replace({0: np.nan}).dropna())
data_numeric['log_mumax'] = np.log1p(data_numeric['mumax'].replace({0: np.nan}).dropna())
data_numeric['log_Mcz'] = np.log1p(data_numeric['Mcz'].replace({0: np.nan}).dropna())

#relevant features
features = data_numeric[['log_Rmag', 'log_mumax', 'log_Mcz', 'UjMAG', 'BjMAG', 'VjMAG',
'usMAG', 'gsMAG', 'rsMAG', 'UbMAG']]

# Normalize
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
features_scaled = pd.DataFrame(features_scaled, columns=features.columns).fillna(0)

#t-SNE for non-linear dimensionality reduction
tsne = TSNE(n_components=2, random_state=42)
```

```
tsne_components = tsne.fit_transform(features_scaled)

# Determine the optimal number of clusters using BIC
n_components = np.arange(1, 11)
gmm_models = [GaussianMixture(n, covariance_type='full',
random_state=42).fit(features_scaled) for n in n_components]
bics = [model.bic(features_scaled) for model in gmm_models]

#BIC values
plt.figure(figsize=(10, 6))
plt.plot(n_components, bics, marker='o')
plt.xlabel('Number of components')
plt.ylabel('BIC')
plt.title('BIC for GMM')
plt.show()

# Use the optimal number of clusters
optimal_n_components = n_components[np.argmin(bics)]
gmm = GaussianMixture(n_components=optimal_n_components, covariance_type='full',
random_state=42)
gmm_labels = gmm.fit_predict(features_scaled)

#Evaluate GMM clustering
gmm_silhouette = silhouette_score(features_scaled, gmm_labels)
data['gmm_cluster'] = gmm_labels

#Visualize the clustering results
plt.figure(figsize=(10, 6))
sns.scatterplot(x=tsne_components[:, 0], y=tsne_components[:, 1], hue='gmm_cluster',
data=data, palette='viridis')
plt.title('GMM Clustering with t-SNE')
plt.show()

#silhouette score for GMM
print(f"GMM Silhouette Score: {gmm_silhouette}")
```

```
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
```

```
result = getattr(ufunc, method)(*inputs, **kwargs)
```





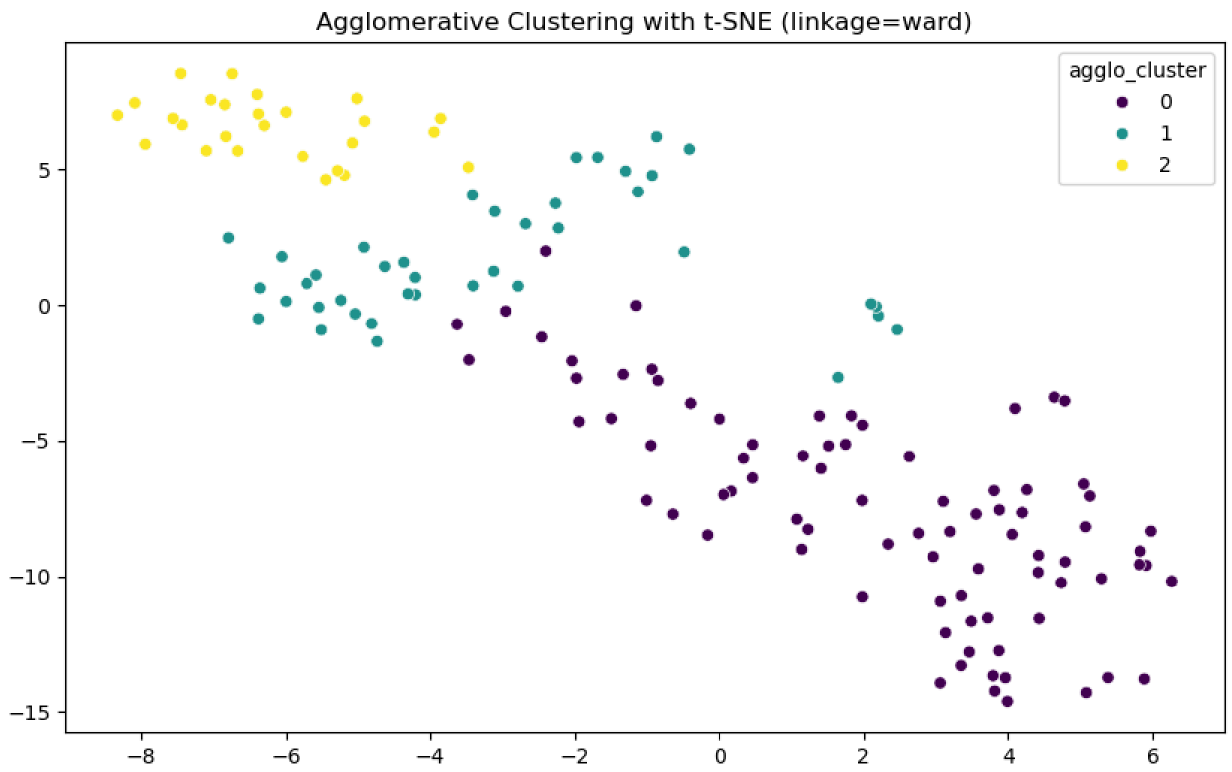
```
GMM Silhouette Score: 0.07724552023580593
```

In [55]:

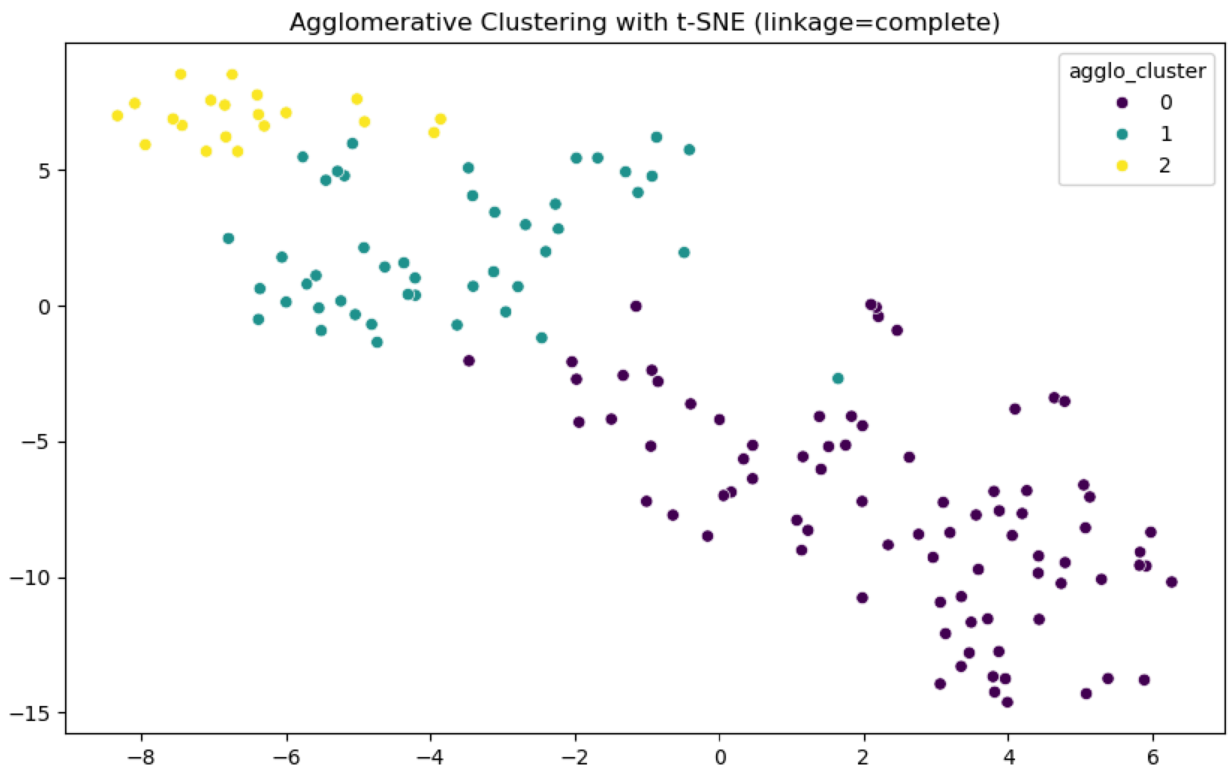
```
data = pd.read_csv(file_path)
data_numeric = data.apply(pd.to_numeric, errors='coerce')
data_numeric = data_numeric.fillna(data_numeric.mean())

# Additional Feature Engineering
data_numeric['log_Rmag'] = np.log1p(data_numeric['Rmag'].replace({0: np.nan}).dropna())
data_numeric['log_mumax'] = np.log1p(data_numeric['mumax'].replace({0: np.nan}).dropna())
data_numeric['log_Mcz'] = np.log1p(data_numeric['Mcz'].replace({0: np.nan}).dropna())

#relevant features
features = data_numeric[['log_Rmag', 'log_mumax', 'log_Mcz', 'UjMAG', 'BjMAG', 'VjMAG',
'usMAG', 'gsMAG', 'rsMAG', 'UbMAG']]

# Normalize
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
features_scaled = pd.DataFrame(features_scaled, columns=features.columns).fillna(0)

#t-SNE for non-linear dimensionality reduction
tsne = TSNE(n_components=2, random_state=42)
tsne_components = tsne.fit_transform(features_scaled)

#run Agglomerative Clustering
def run_agglomerative_clustering(linkage_criteria, n_clusters=3):
   agglo = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage_criteria)
   agglo_labels = agglo.fit_predict(features_scaled)

   agglo_silhouette = silhouette_score(features_scaled, agglo_labels)

   data['agglo_cluster'] = agglo_labels

   plt.figure(figsize=(10, 6))
   sns.scatterplot(x=tsne_components[:, 0], y=tsne_components[:, 1], hue='agglo_cluster',
data=data, palette='viridis')
   plt.title(f'Agglomerative Clustering with t-SNE (linkage={linkage_criteria})')
   plt.show()

   print(f"Agglomerative Clustering Silhouette Score (linkage={linkage_criteria}):
{agglo_silhouette}")

#different linkage criteria
```

```
run_agglomerative_clustering('ward', n_clusters=3)
run_agglomerative_clustering('complete', n_clusters=3)
run_agglomerative_clustering('average', n_clusters=3)
run_agglomerative_clustering('single', n_clusters=3)

#silhouette scores for each linkage method
def print_silhouette_scores(linkage_criteria, n_clusters=3):
    agglo = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage_criteria)
    agglo_labels = agglo.fit_predict(features_scaled)
    agglo_silhouette = silhouette_score(features_scaled, agglo_labels)
    print(f" (linkage={linkage_criteria}): {agglo_silhouette}")

print_silhouette_scores('ward', n_clusters=3)
print_silhouette_scores('complete', n_clusters=3)
print_silhouette_scores('average', n_clusters=3)
print_silhouette_scores('single', n_clusters=3)
```

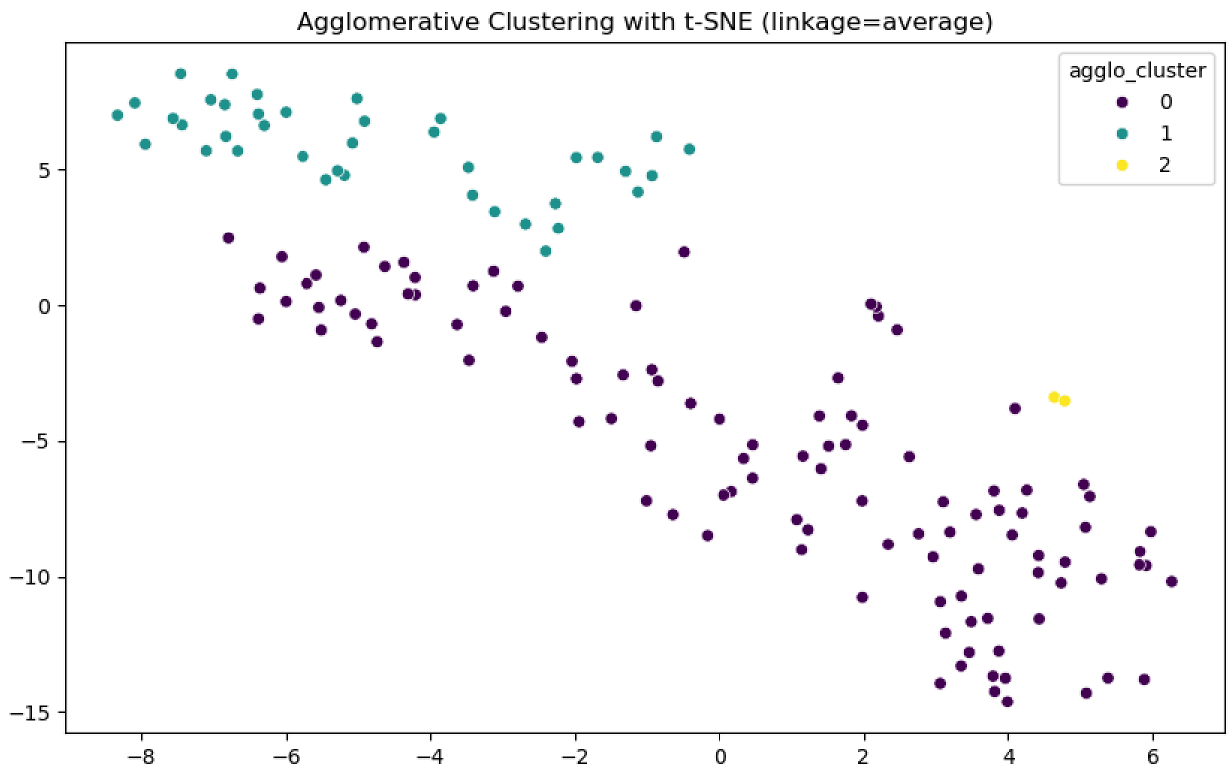
```
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



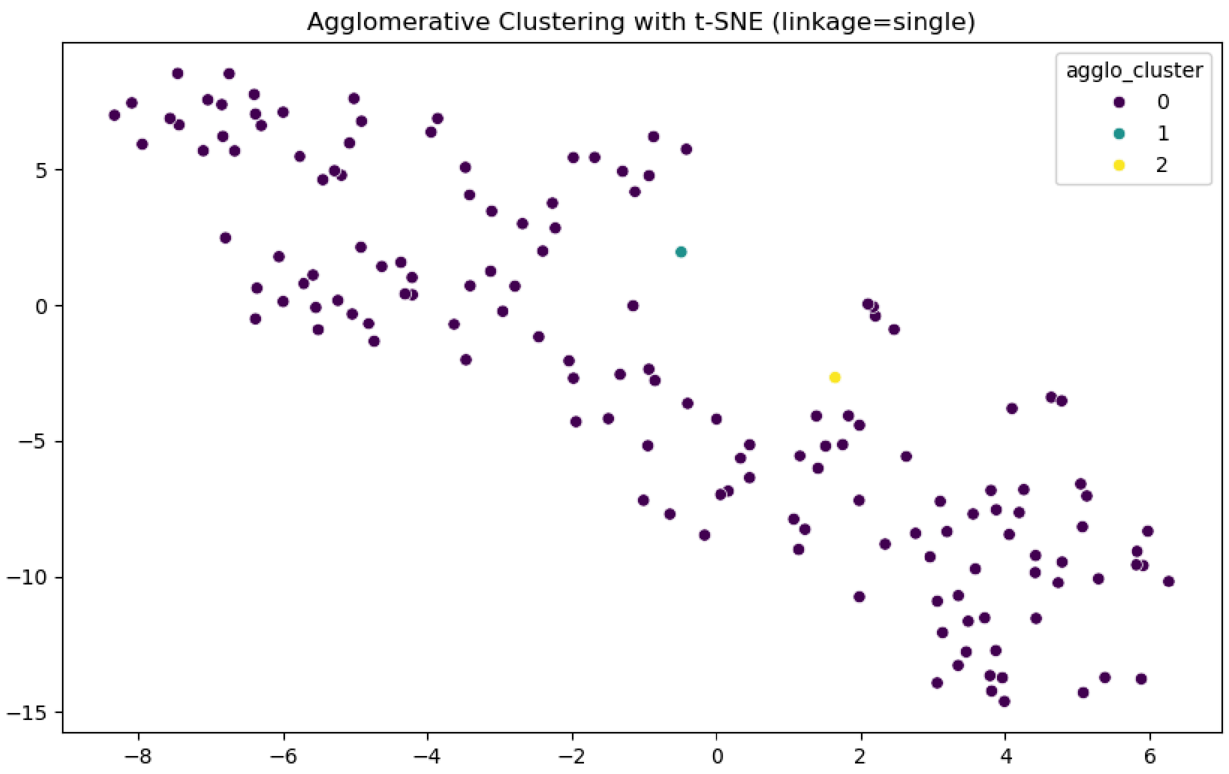
Agglomerative Clustering Silhouette Score (linkage=ward): 0.31739841120450185



Agglomerative Clustering Silhouette Score (linkage=complete): 0.3384611798451422



Agglomerative Clustering Silhouette Score (linkage=average): 0.34465057891510953



```
Agglomerative Clustering Silhouette Score (linkage=single): 0.11041841919998908
 (linkage=ward): 0.31739841120450185
 (linkage=complete): 0.3384611798451422
 (linkage=average): 0.34465057891510953
 (linkage=single): 0.11041841919998908
```

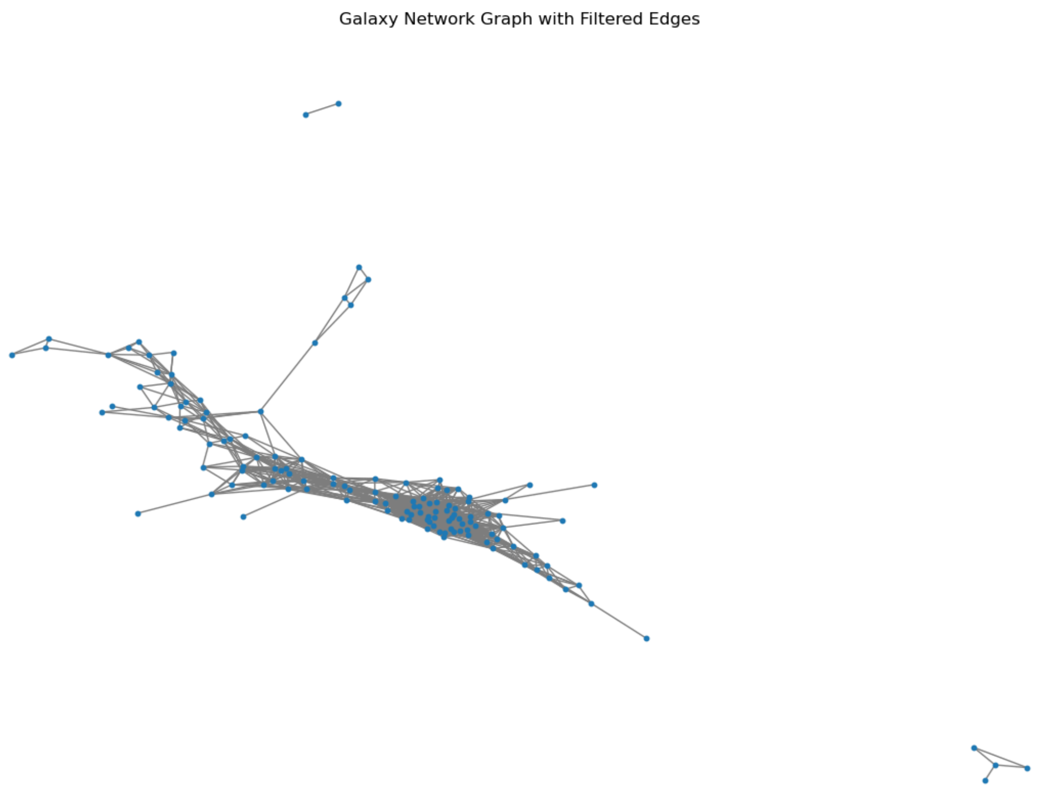
In [56]:

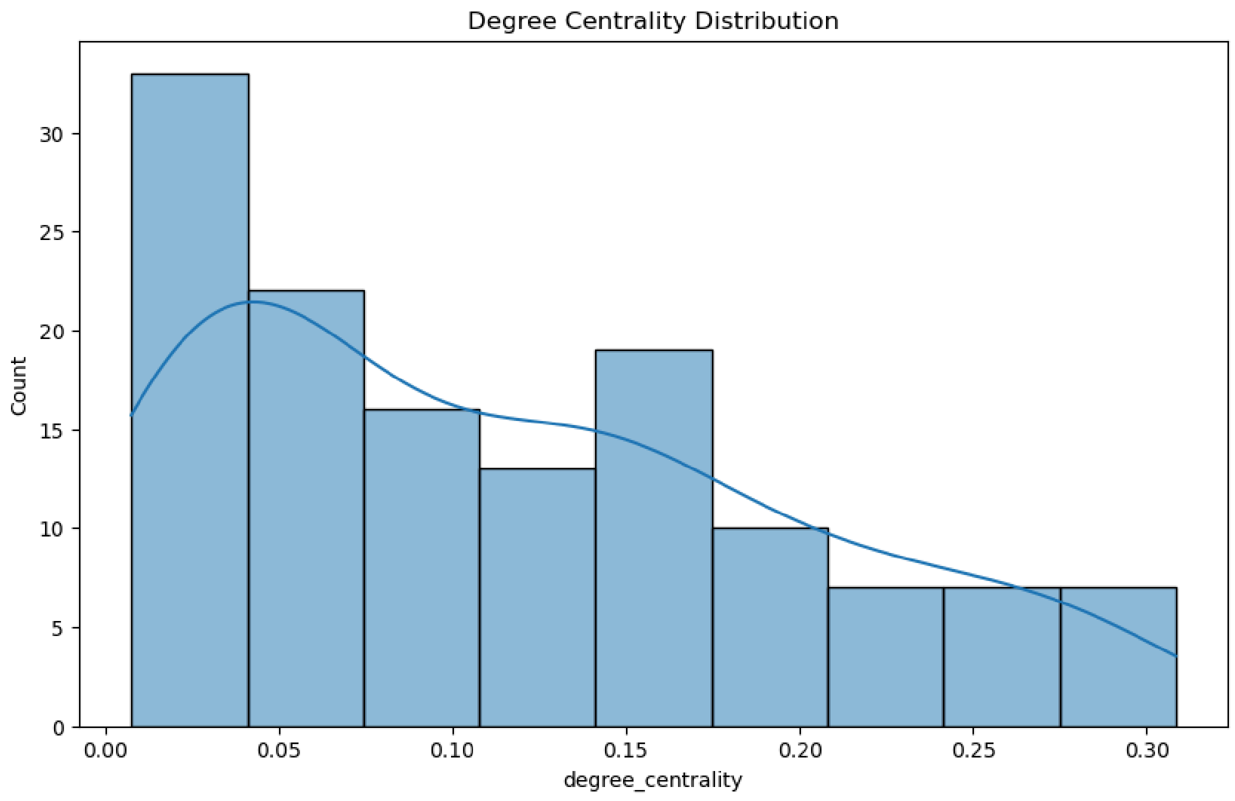
```
data = pd.read_csv(file_path)
data_numeric = data.apply(pd.to_numeric, errors='coerce')
data_numeric = data_numeric.fillna(data_numeric.mean())

#Additional Feature Engineering
data_numeric['log_mumax'] = np.log1p(data_numeric['mumax'].replace({0: np.nan}).fillna(1))
data_numeric['log_Mcz'] = np.log1p(data_numeric['Mcz'].replace({0: np.nan}).fillna(1))
data_numeric['Rmag_to_mumax'] = data_numeric['Rmag'] / data_numeric['mumax'].replace({0:
np.nan}).fillna(1)
data_numeric['Mcz_diff'] = data_numeric['Mcz'] - data_numeric['log_Mcz']

#relevant features
features = data_numeric[['log_mumax', 'log_Mcz', 'Rmag_to_mumax', 'Mcz_diff', 'UjMAG',
'BjMAG', 'VjMAG', 'usMAG', 'gsMAG', 'rsMAG', 'UbMAG']]

scaler = StandardScaler()
```

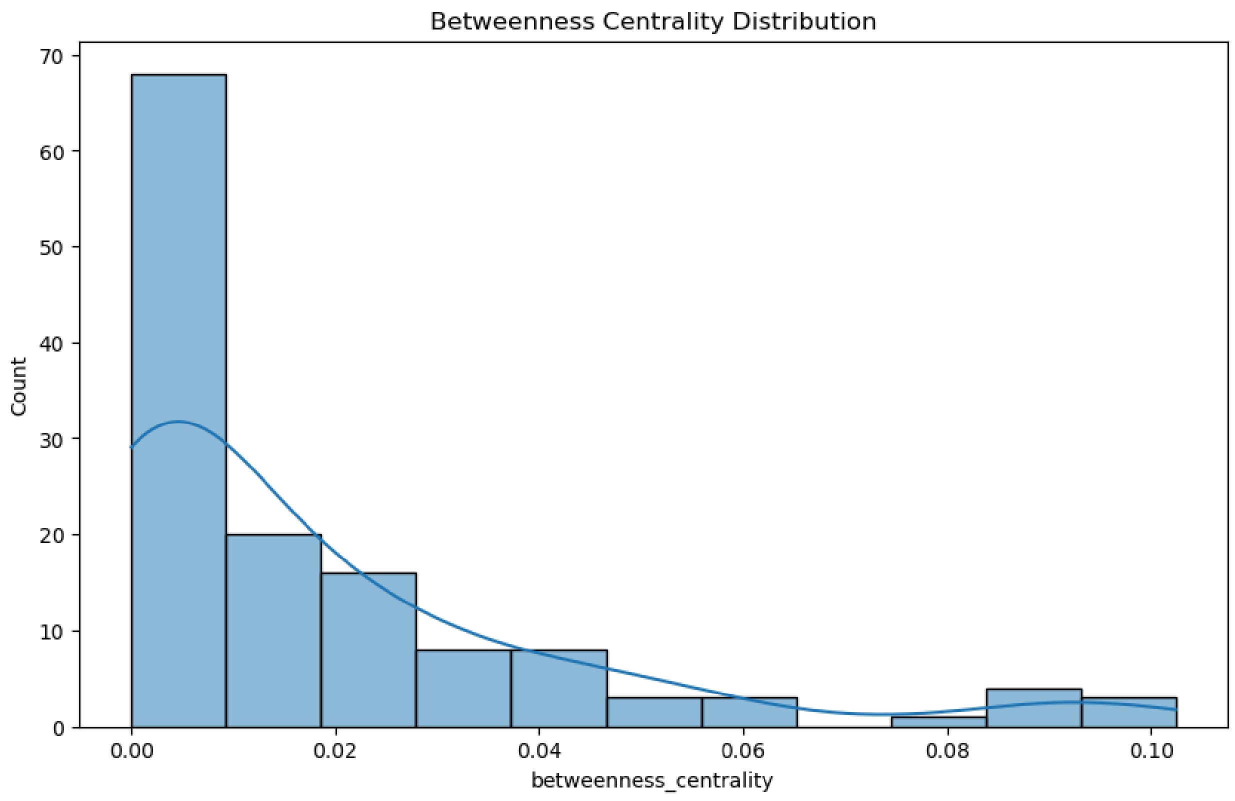
```
features_scaled = scaler.fit_transform(features)

#no NaNs
features_scaled = pd.DataFrame(features_scaled, columns=features.columns).fillna(0)

#similarity matrix using Euclidean distance
distance_matrix = pairwise_distances(features_scaled, metric='euclidean')

#filter edges
threshold = np.percentile(distance_matrix, 10)  # Only keep the closest 10% edges

#distance matrix with edges below the threshold
G = nx.Graph()
for i in range(len(distance_matrix)):
   for j in range(i + 1, len(distance_matrix)):
      if distance_matrix[i, j] < threshold:
         G.add_edge(i, j, weight=distance_matrix[i, j])

print("Number of nodes:", G.number_of_nodes())
print("Number of edges:", G.number_of_edges())

#Calculate centrality measures
degree_centrality = nx.degree_centrality(G)
betweenness_centrality = nx.betweenness_centrality(G)
closeness_centrality = nx.closeness_centrality(G)

#Add centrality measures
data['degree_centrality'] = pd.Series(degree_centrality)
data['betweenness_centrality'] = pd.Series(betweenness_centrality)
data['closeness_centrality'] = pd.Series(closeness_centrality)

plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G, seed=42)  # Layout for visualization
nx.draw(G, pos, node_size=10, with_labels=False, edge_color='gray')
plt.title("Galaxy Network Graph with Filtered Edges")
plt.show()

#centrality measures
plt.figure(figsize=(10, 6))
sns.histplot(data['degree_centrality'], kde=True)
plt.title('Degree Centrality Distribution')
plt.show()

plt.figure(figsize=(10, 6))
```

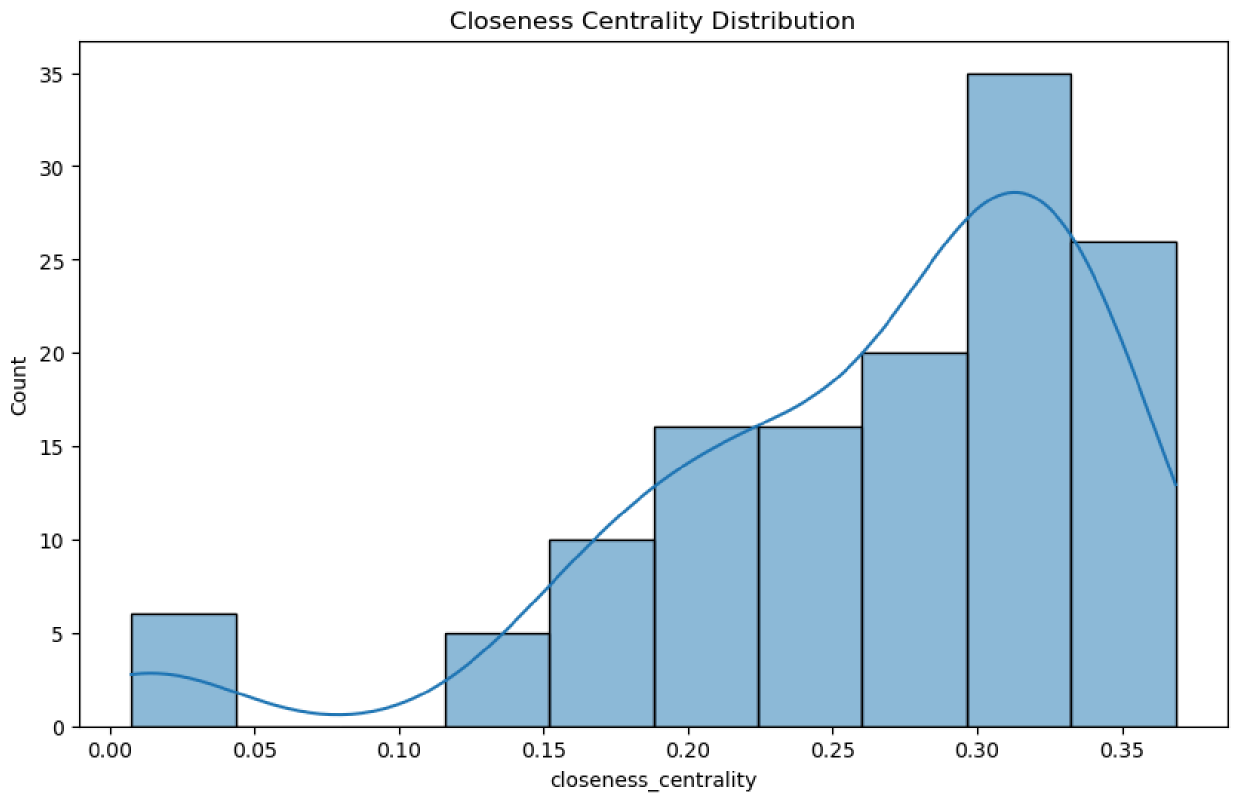
```
sns.histplot(data['betweenness_centrality'], kde=True)
plt.title('Betweenness Centrality Distribution')
plt.show()

plt.figure(figsize=(10, 6))
sns.histplot(data['closeness_centrality'], kde=True)
plt.title('Closeness Centrality Distribution')
plt.show()
```

```
Number of nodes: 134
Number of edges: 1021

/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/Users/cameronsouza/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
```




Degree Centrality Distribution
Count
degree_centrality
0
5
10
15
20
25
30
0.00
0.05
0.10
0.15
0.20
0.25
0.30


Betweenness Centrality Distribution
Count
betweenness_centrality
0
10
20
30
40
50
60
70
0.00
0.02
0.04
0.06
0.08
0.10



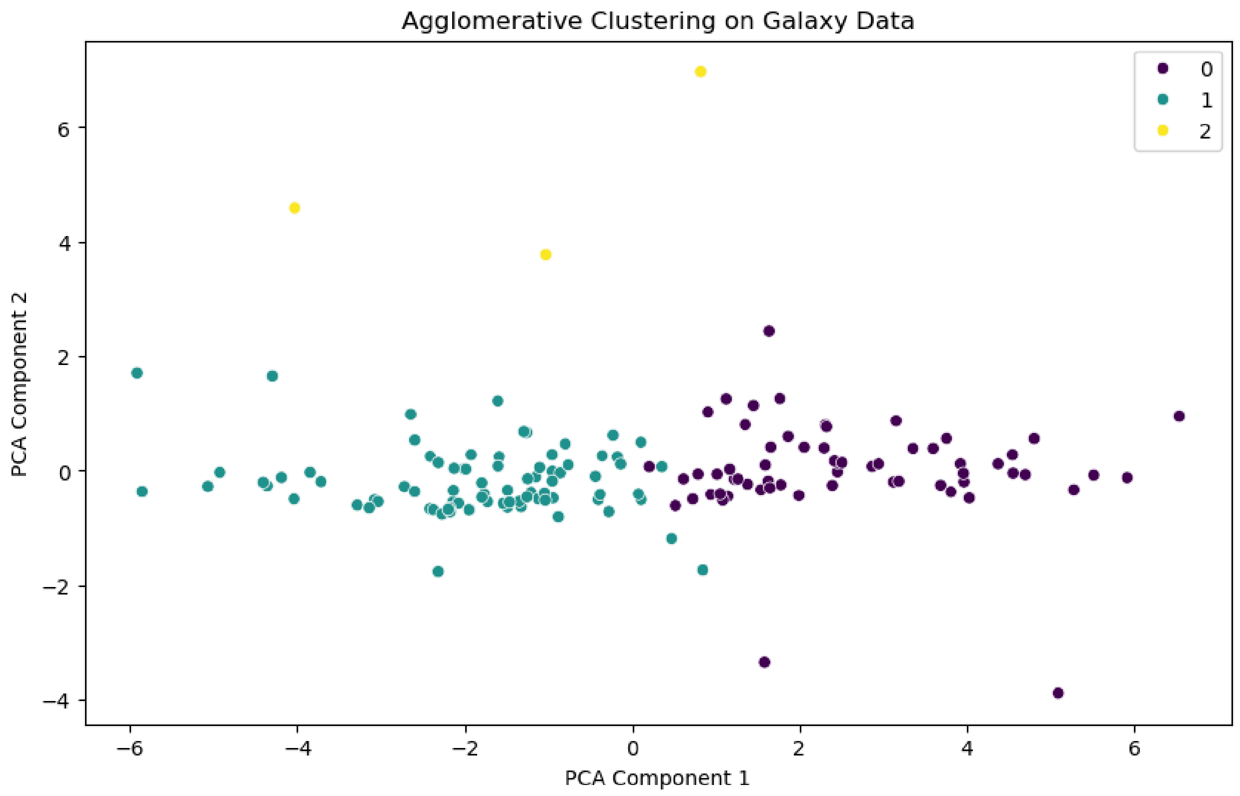
In [64]:

```
from sklearn.ensemble import GradientBoostingClassifier

# Train a Gradient Boosting classifier to determine feature importance
gbc = GradientBoostingClassifier(random_state=42)
gbc.fit(features_scaled, dbscan_labels)

# Get feature importances
feature_importances = gbc.feature_importances_

# Create a DataFrame for better visualization
feature_importance_df = pd.DataFrame({
    'feature': features.columns,
    'importance': feature_importances
})

# Sort features by importance
feature_importance_df = feature_importance_df.sort_values(by='importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance_df)
plt.title('Feature Importances for DBSCAN Clustering')
plt.show()

# Print the most important features
print("Most important features for clustering:")
print(feature_importance_df)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[64], line 5
      3 # Train a Gradient Boosting classifier to determine feature importance
      4 gbc = GradientBoostingClassifier(random_state=42)
----> 5 gbc.fit(features_scaled, dbscan_labels)
      7 # Get feature importances
      8 feature_importances = gbc.feature_importances_

File ~/anaconda3/lib/python3.11/site-packages/sklearn/base.py:1474, in
_fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1467     estimator._validate_params()
   1469 with config_context(
   1470     skip_parameter_validation=(
   1471         prefer_skip_nested_validation or global_skip_validation
   1472     )
   1473 ):
-> 1474     return fit_method(estimator, *args, **kwargs)

File ~/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:665, in
BaseGradientBoosting.fit(self, X, y, sample_weight, monitor)
    663 sample_weight = _check_sample_weight(sample_weight, X)
    664 if sample_weight_is_none:
--> 665     y = self._encode_y(y=y, sample_weight=None)
    666 else:
    667     y = self._encode_y(y=y, sample_weight=sample_weight)

File ~/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:1520, in
GradientBoostingClassifier._encode_y(self, y, sample_weight)
   1517     n_trim_classes = np.count_nonzero(np.bincount(encoded_y_int, sample_weight))
   1519 if n_trim_classes < 2:
-> 1520     raise ValueError(
   1521         "y contains %d class after sample_weight "
   1522         "trimmed classes with zero weights, while a "
   1523         "minimum of 2 classes are required." % n_trim_classes
   1524     )
   1525 return encoded_y
```

```
ValueError: y contains 1 class after sample_weight trimmed classes with zero weights, while a minimum of 2 classes are required.
```

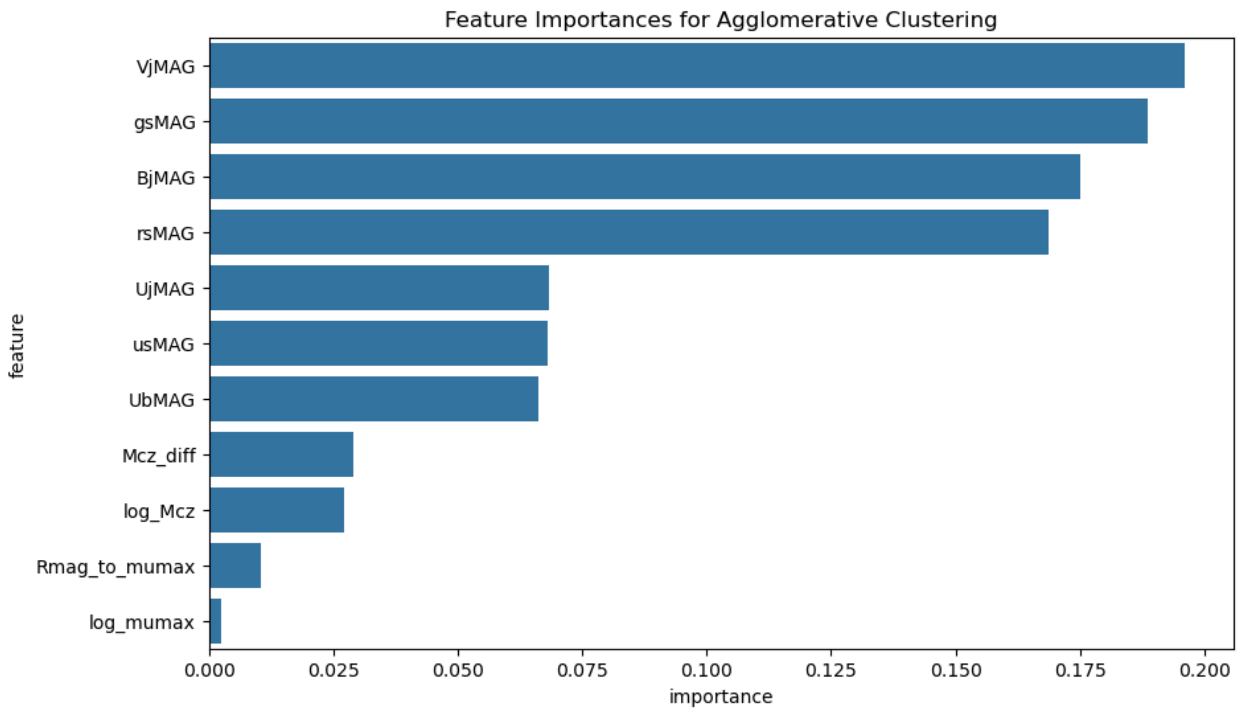
In [66]:

```
from sklearn.cluster import AgglomerativeClustering

# Apply Agglomerative Clustering
agglo = AgglomerativeClustering(n_clusters=3)
agglo_labels = agglo.fit_predict(features_scaled)

# Apply PCA for visualization
pca = PCA(n_components=2)
pca_components = pca.fit_transform(features_scaled)

plt.figure(figsize=(10, 6))
sns.scatterplot(x=pca_components[:, 0], y=pca_components[:, 1], hue=agglo_labels,
palette='viridis')
plt.title('Agglomerative Clustering on Galaxy Data')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()

# Add Agglomerative Clustering labels to the original data for further analysis
data['agglo_cluster'] = agglo_labels
```



In [67]:

```
from sklearn.ensemble import RandomForestClassifier

# Train a Random Forest classifier to determine feature importance
rf = RandomForestClassifier(random_state=42)
rf.fit(features_scaled, agglo_labels)

# Get feature importances
feature_importances = rf.feature_importances_

# Create a DataFrame for better visualization
feature_importance_df = pd.DataFrame({
    'feature': features.columns,
    'importance': feature_importances
})

# Sort features by importance
feature_importance_df = feature_importance_df.sort_values(by='importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance_df)
plt.title('Feature Importances for Agglomerative Clustering')
plt.show()

# Print the most important features
print("Most important features for clustering:")
print(feature_importance_df)
```



```
Most important features for clustering:
        feature  importance
6         VjMAG    0.195959
8         gsMAG    0.188482
5         BjMAG    0.175149
9         rsMAG    0.168760
4         UjMAG    0.068384
7         usMAG    0.068047
10        UbMAG    0.066177
3      Mcz_diff    0.028912
1       log_Mcz    0.027183
2  Rmag_to_mumax    0.010412
0     log_mumax    0.002535
```

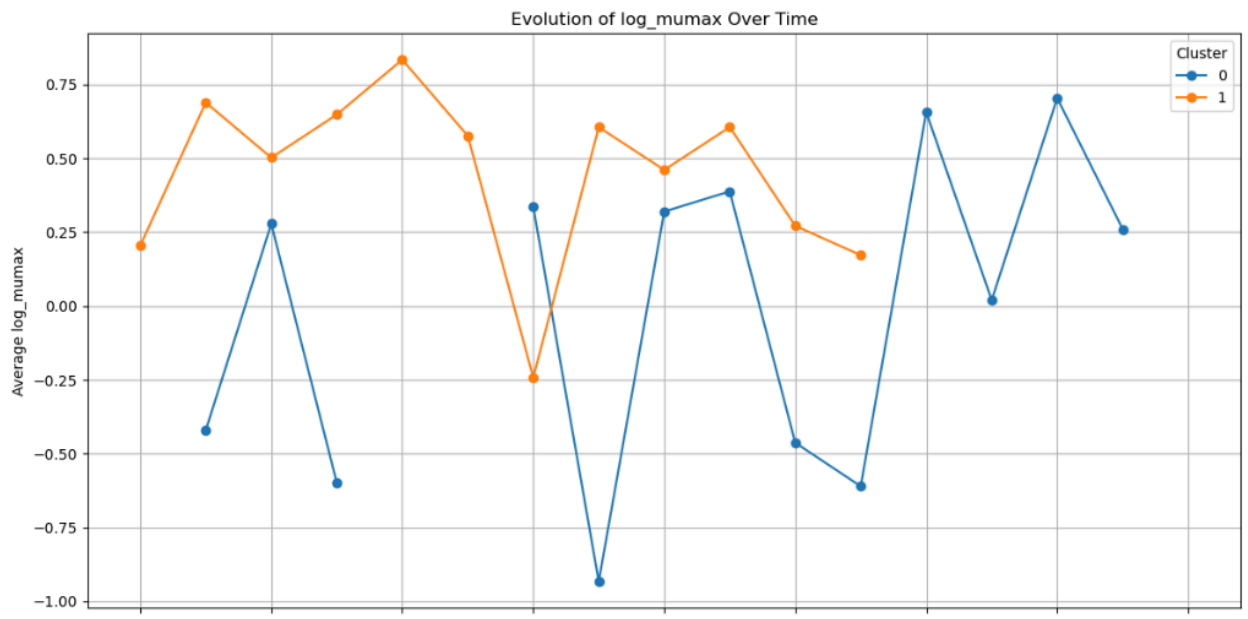
In [75]:

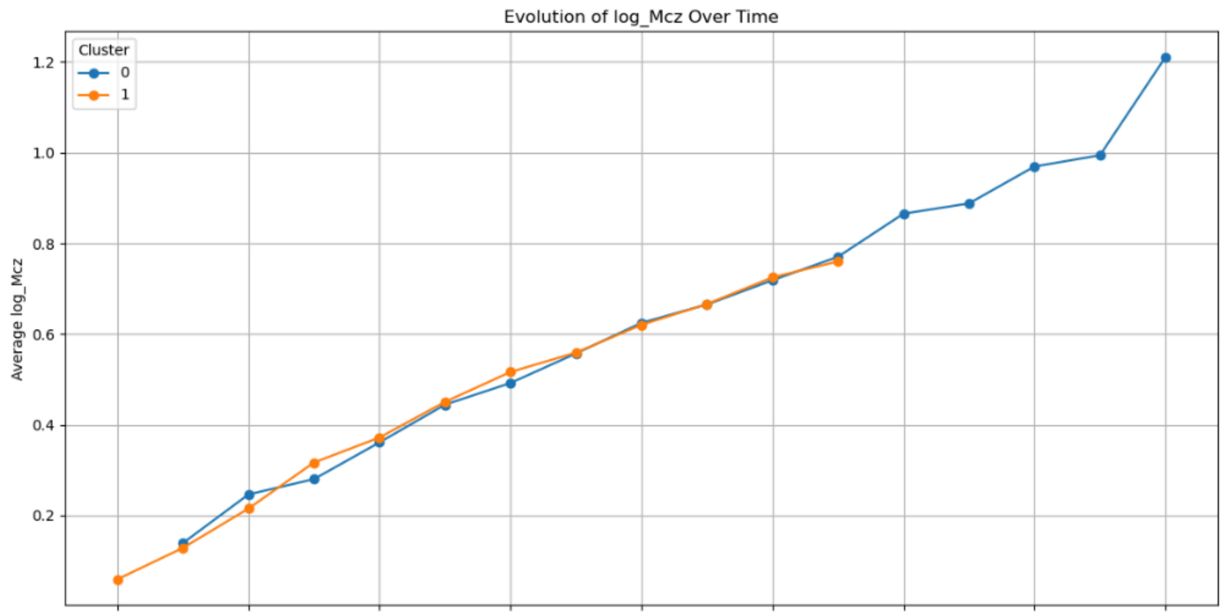
```
# Ensure the 'agglo_cluster' labels are present in the data
data['agglo_cluster'] = agglo_labels

# Extract relevant columns including redshift and cluster labels
```

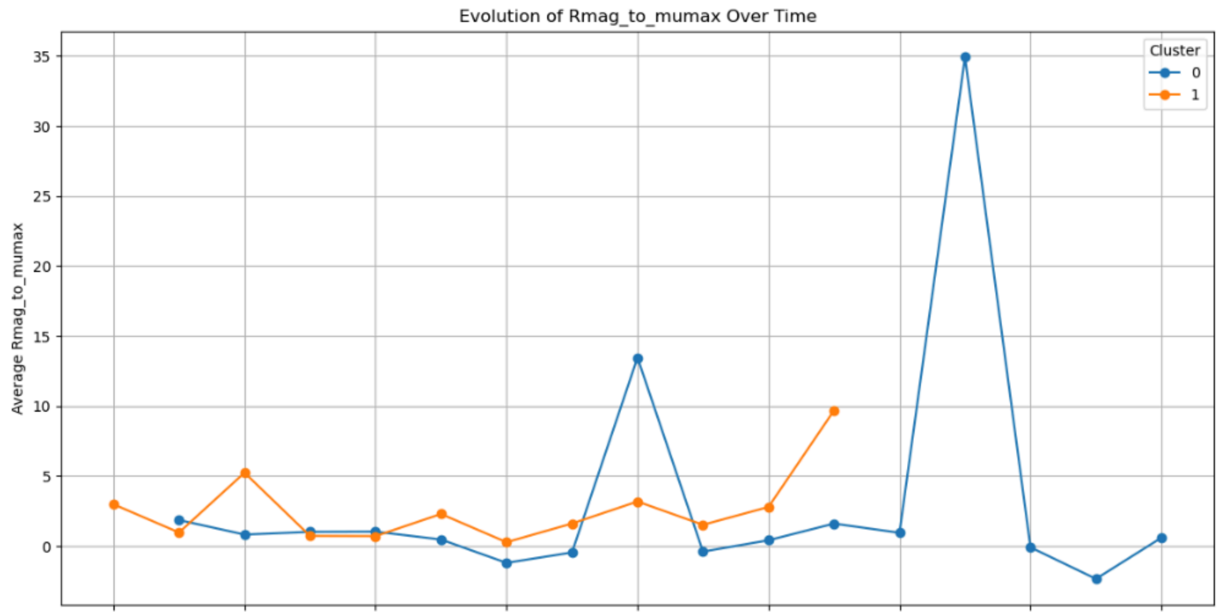
```
temporal_data = data[['Mcz', 'log_mumax', 'log_Mcz', 'Rmag_to_mumax', 'Mcz_diff',
'agglo_cluster']].copy()

# Bin the redshift values to create time intervals
temporal_data.loc[:, 'redshift_bin'] = pd.cut(temporal_data['Mcz'], bins=np.arange(0,
temporal_data['Mcz'].max() + 0.1, 0.1))

# Ensure no NaNs remain in the binned data
temporal_data = temporal_data.dropna(subset=['redshift_bin'])
```

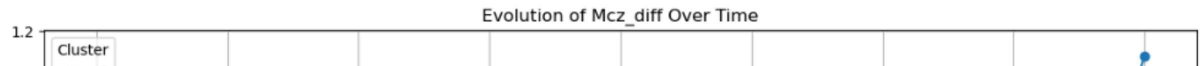
In [76]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Define the properties to be analyzed
properties = ['log_mumax', 'log_Mcz', 'Rmag_to_mumax', 'Mcz_diff']

# Create a figure for the subplots
fig, axes = plt.subplots(len(properties), 1, figsize=(12, 24), sharex=True)

for i, prop in enumerate(properties):
    # Aggregate the property by redshift bins and cluster labels
    agg_data = temporal_data.groupby(['redshift_bin', 'agglo_cluster'],
observed=True)[prop].mean().unstack()

    # Plot the aggregated data
    agg_data.plot(ax=axes[i], marker='o')
    axes[i].set_title(f'Evolution of {prop} Over Time')
    axes[i].set_xlabel('Redshift Bin')
    axes[i].set_ylabel(f'Average {prop}')
    axes[i].legend(title='Cluster')
    axes[i].grid(True)

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


Evolution of log_mumax Over Time
Average log_mumax
Cluster
0
1
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Evolution of log_Mcz Over Time
Average log_Mcz
Cluster
0
1
1.2
1.0
0.8
0.6
0.4
0.2


Evolution of Rmag_to_mumax Over Time
Average Rmag_to_mumax
Cluster
0
1
35
30
25
20
15
10
5
0


Evolution of Mcz_diff Over Time
1.2
Cluster

In [77]:

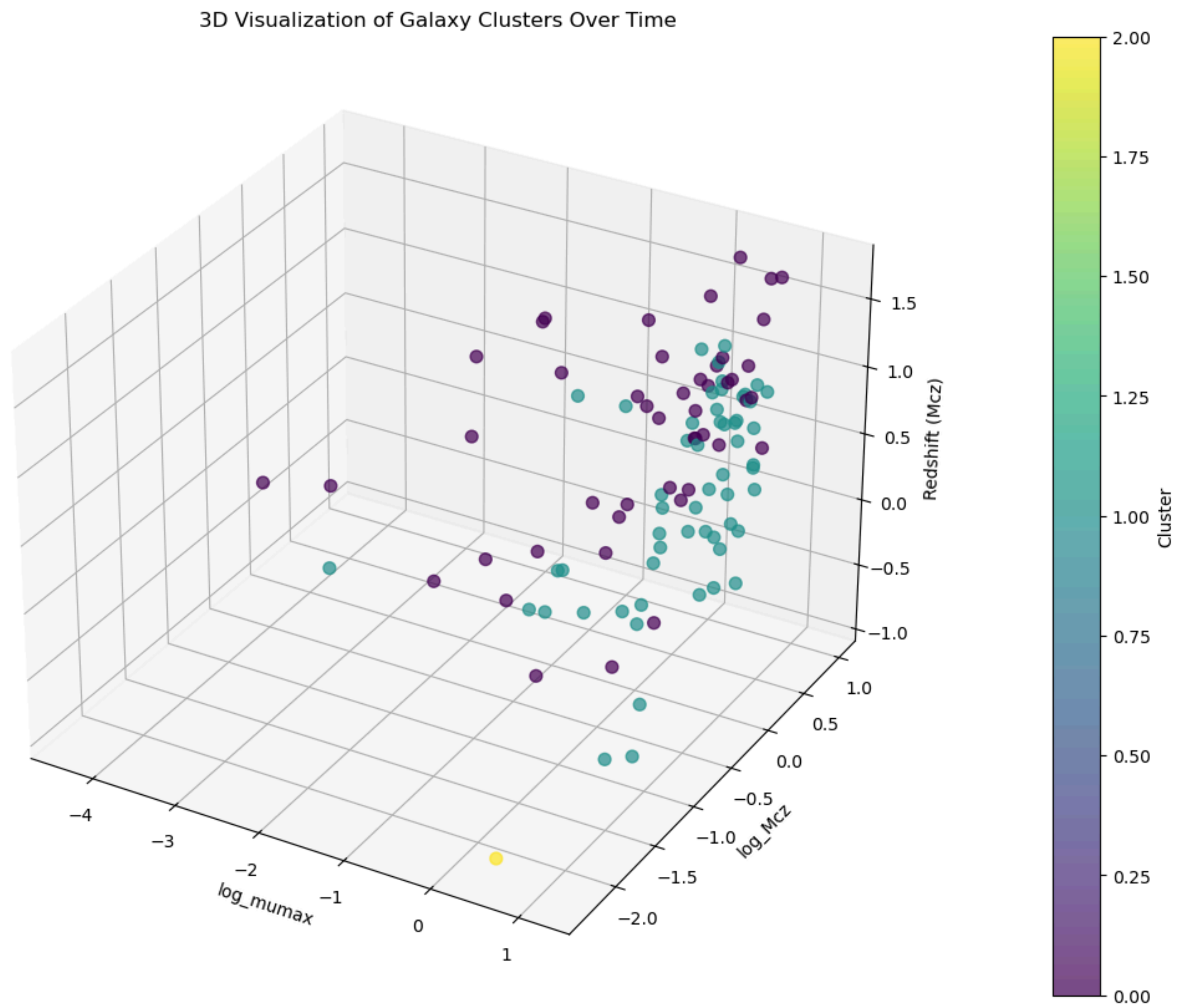
```
import plotly.graph_objects as go
import pandas as pd

# Calculate mean values of properties for each cluster
cluster_means = data.groupby('agglo_cluster').mean()[['log_mumax', 'log_Mcz',
'Rmag_to_mumax', 'Mcz_diff']]

# Create a radar chart for each cluster
fig = go.Figure()

for cluster in cluster_means.index:
    fig.add_trace(go.Scatterpolar(
        r=cluster_means.loc[cluster].values,
        theta=cluster_means.columns,
        fill='toself',
        name=f'Cluster {cluster}'
    ))

fig.update_layout(
    polar=dict(
        radialaxis=dict(
            visible=True,
            range=[cluster_means.values.min(), cluster_means.values.max()]
        )),
    showlegend=True,
    title="Radar Chart of Cluster Profiles"
)

fig.show()
```

```
log_mumaxlog_MczRmag_to_mumaxMcz_diff−2−1.5−1−0.500.511.5

Cluster 0Cluster 1Cluster 2Radar Chart of Cluster Profiles
```

In [81]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Create the 3D scatter plot
fig = plt.figure(figsize=(14, 10))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot with color representing clusters
```

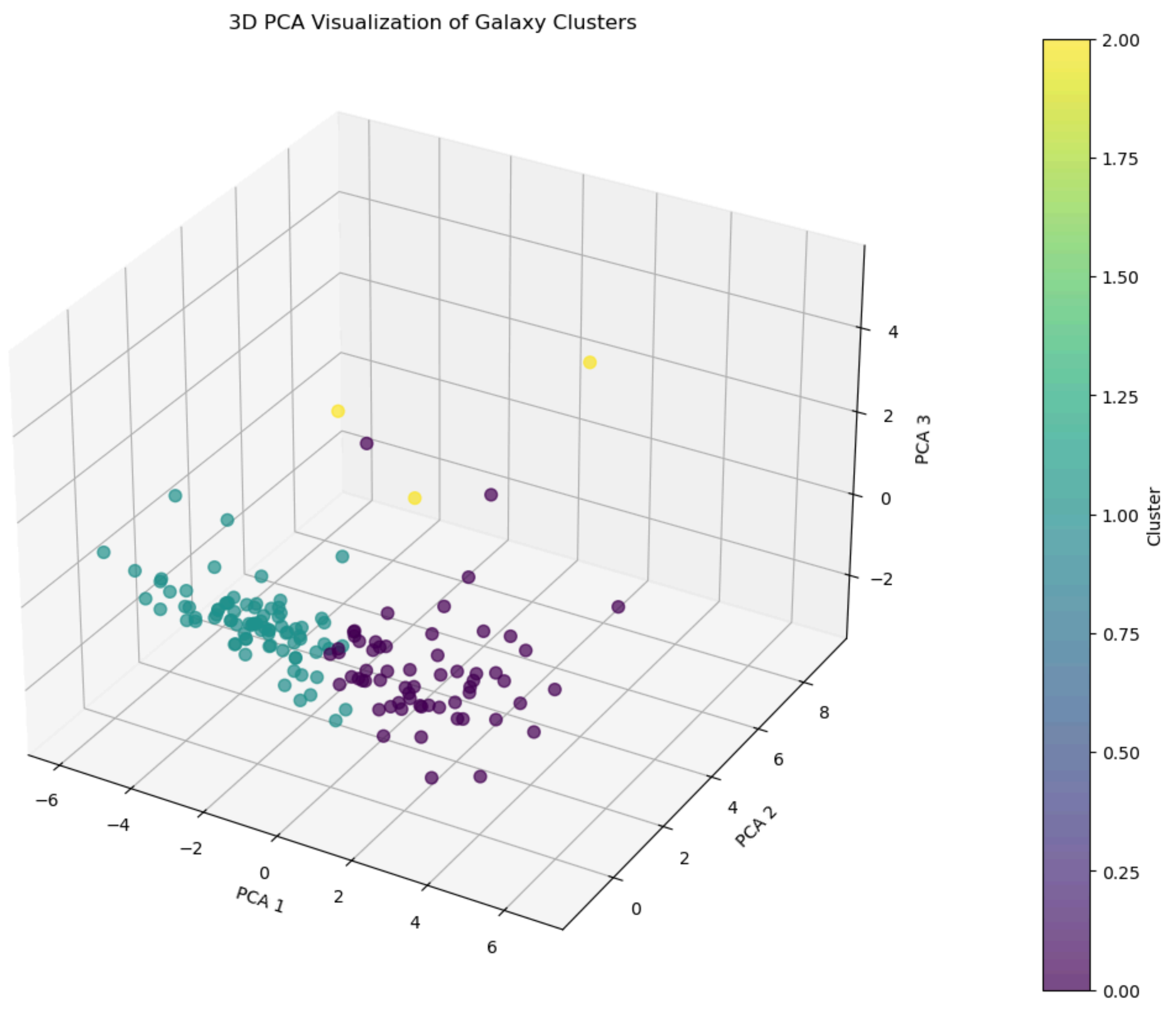
```
scatter = ax.scatter(data['log_mumax'], data['log_Mcz'], data['Mcz'], c=data['agglo_cluster'],
cmap='viridis', s=50, alpha=0.7)

# Add color bar
cbar = plt.colorbar(scatter, ax=ax, pad=0.1)
cbar.set_label('Cluster')

# Set labels
ax.set_title('3D Visualization of Galaxy Clusters Over Time')
ax.set_xlabel('log_mumax')
ax.set_ylabel('log_Mcz')
ax.set_zlabel('Redshift (Mcz)')

# Show the plot
plt.show()
```



In [83]:

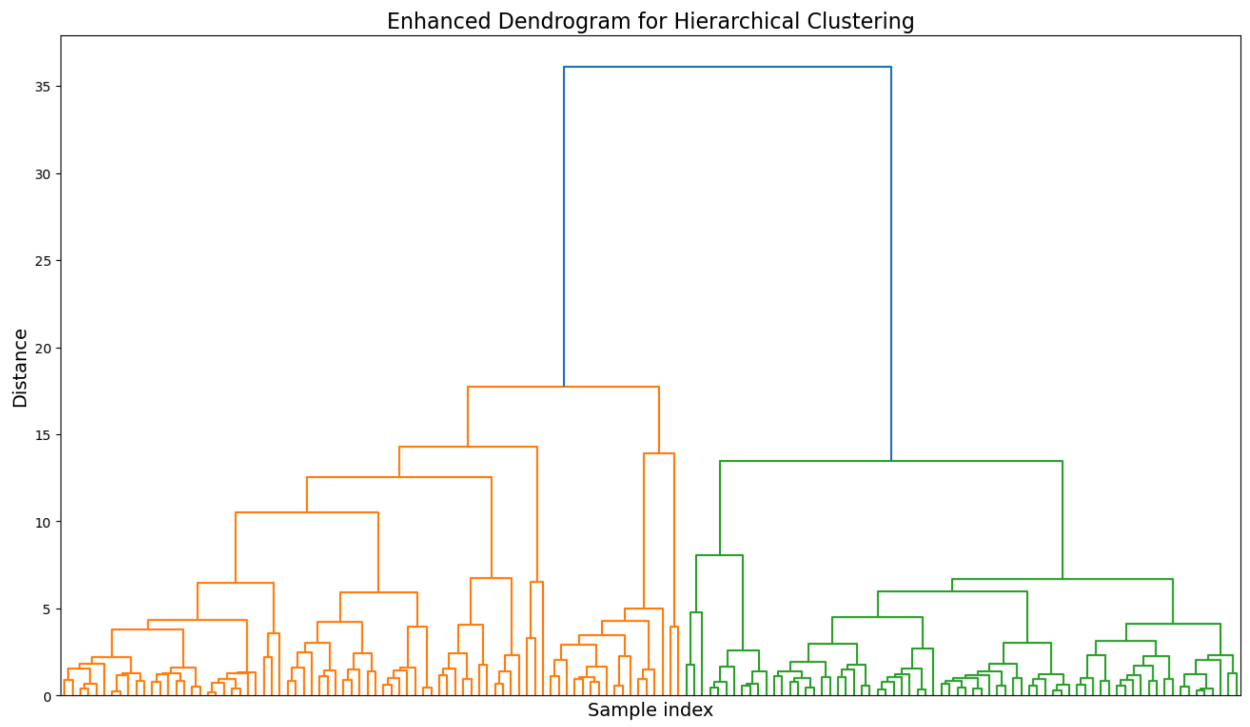
```
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features_scaled)

#reduce dimensions to 3D
pca = PCA(n_components=3)
pca_results = pca.fit_transform(features_scaled)

# Add the PCA
data['pca_1'] = pca_results[:, 0]
data['pca_2'] = pca_results[:, 1]
data['pca_3'] = pca_results[:, 2]

# Verify the added PCA components
print(data[['pca_1', 'pca_2', 'pca_3']].head())

fig = plt.figure(figsize=(14, 10))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
scatter = ax.scatter(data['pca_1'], data['pca_2'], data['pca_3'], c=data['agglo_cluster'],
cmap='viridis', s=50, alpha=0.7)
cbar = plt.colorbar(scatter, ax=ax, pad=0.1)
cbar.set_label('Cluster')
ax.set_title('3D PCA Visualization of Galaxy Clusters')
ax.set_xlabel('PCA 1')
ax.set_ylabel('PCA 2')
ax.set_zlabel('PCA 3')
plt.show()
```

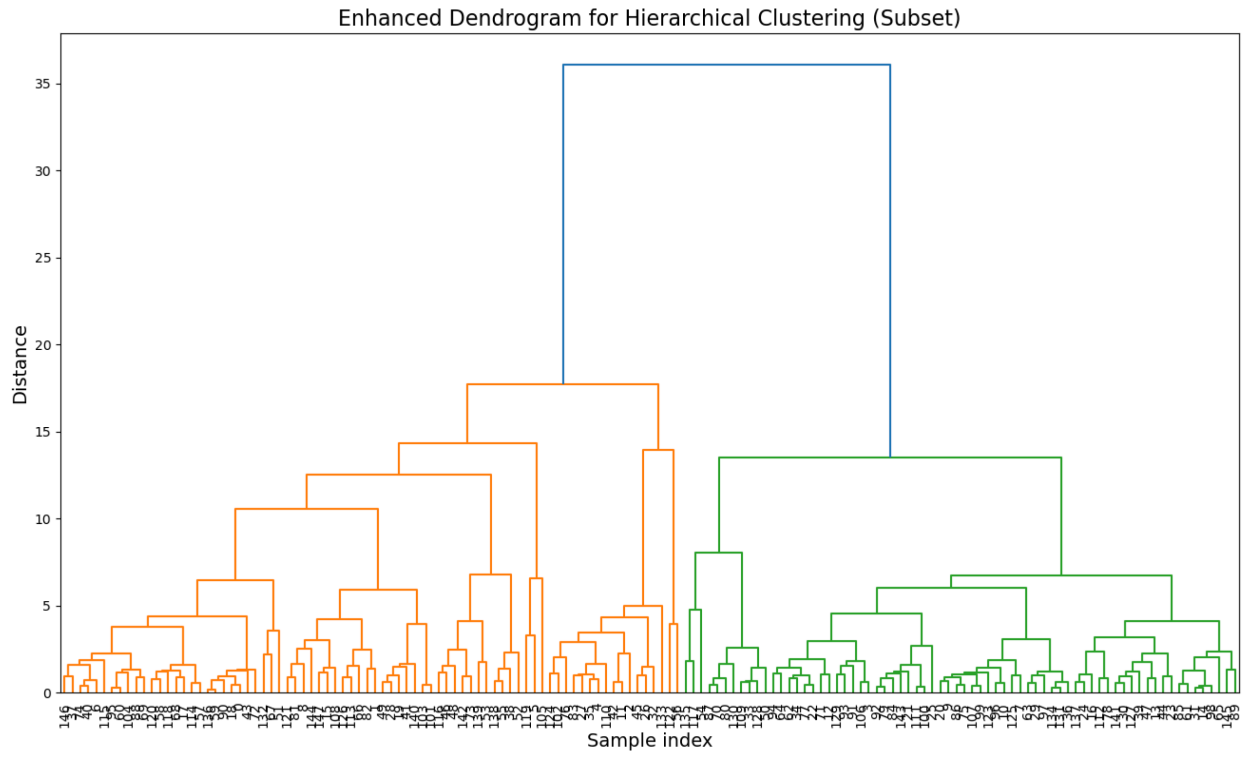
```
      pca_1     pca_2     pca_3
0  0.952662 -0.373512  0.629831
1  2.004353  0.225491 -0.318919
2  2.884568  0.257875  1.388798
3 -1.618907  0.010958 -0.625755
4  4.544640 -0.194846  0.544496
```



In [88]:

```
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Perform hierarchical/agglomerative clustering
linked = linkage(features_scaled, method='ward')

# Create a dendrogram without sample index labels
plt.figure(figsize=(16, 9))

dendrogram(
    linked,
    orientation='top',
    distance_sort='descending',
    show_leaf_counts=False,  # Hide leaf counts to reduce clutter
    no_labels=True,  # Hide sample index labels
    color_threshold=20  # Set color threshold for different clusters
)

# Add title and labels
plt.title('Enhanced Dendrogram for Hierarchical Clustering', fontsize=16)
plt.xlabel('Sample index', fontsize=14)
plt.ylabel('Distance', fontsize=14)

# Show the plot
plt.show()
# Use a smaller subset of data (first 200 samples)
subset_size = 200
features_subset = features_scaled[:subset_size]

# Perform hierarchical/agglomerative clustering on the subset
linked_subset = linkage(features_subset, method='ward')

# Create a dendrogram with the subset
plt.figure(figsize=(16, 9))

dendrogram(
    linked_subset,
    orientation='top',
    distance_sort='descending',
    show_leaf_counts=False,  # Hide leaf counts to reduce clutter
    leaf_rotation=90,  # Rotate x-axis labels for better readability
    leaf_font_size=10,  # Increase font size of labels
    color_threshold=20  # Set color threshold for different clusters
)

# Add title and labels
plt.title('Enhanced Dendrogram for Hierarchical Clustering (Subset)', fontsize=16)
plt.xlabel('Sample index', fontsize=14)
plt.ylabel('Distance', fontsize=14)

# Show the plot
plt.show()
```


Enhanced Dendrogram for Hierarchical Clustering
Distance
Sample index
0
5
10
15
20
25
30
35


Enhanced Dendrogram for Hierarchical Clustering (Subset)
Distance
Sample index
0
5
10
15
20
25
30
35

In [ ]: