SCHOOL OF ARTIFICIAL INTELLIGENCE, DELHI NCR

PROJECT REPORT

Stellar Classification using Machine Learning

Course Code & Title

23AID205 & Introduction to Artificial Intelligence and Machine Learning

Semester: II

Academic Year: 2024 – 2025 [Even]

Student Roll no(s)	Student Name(s)
DL.AI.U4AID24006	Arnav Sharma
DL.AI.U4AID24040	Hemant
DL.AI.U4AID24045	Neeraj
DL.AI.U4AID24039	Sriyansh

Faculty Guide: Prof. Sri Geetha M

Table of Contents

List of Tables

List of Figures

1. Title of the Project

Stellar Classification using Machine Learning

2. Abstract

This project focuses on celestial object classification using supervised and unsupervised machine learning algorithms. A structured dataset containing various spectral and positional features is preprocessed, analyzed, and modeled. Techniques such as Support Vector Machine (SVM) and KMeans Clustering are applied. The dataset is cleaned, color features are engineered, and split into training and testing subsets. Evaluation metrics like accuracy score (for SVM) and visual interpretation of clustering patterns (for KMeans) are used to assess the performance of the models.

3. Objective(s) of the Project

- To clean and preprocess the **celestial object dataset**, including handling placeholder values and engineering color features**.**
- To apply supervised learning (**e.g., Support Vector Machine**) and unsupervised learning (**KMeans Clustering**) for classification modeling and pattern discovery.
- To **predict celestial object classes** using the trained supervised model.
- To analyze model **accuracy and clustering patterns** using relevant evaluation metrics and visualizations.

4. Literature Survey (Optional)

1. Smith et al. (2020) - 'Diabetes Prediction using ML'
2. John et al. (2021) - 'Clustering Medical Data for Insights'

5. Dataset Description

Source: Sloan Digital Sky Survey (SDSS)
Rows: 100,000 rows
Columns: 18 columns
Features: The dataset includes a mix of categorical features (e.g., class label) and numerical features (e.g., various spectral bands like u, g, r, i, z, and positional coordinates like alpha, delta, as well as mjd for observation time).
Target Variable**:** The class column (e.g., 'GALAXY', 'STAR', 'QSO') is the target variable used for supervised learning.
Missing values handled: The raw dataset initially contained specific placeholder values (-9999.0) in certain spectral columns, which were identified and addressed during the preprocessing phase to ensure a clean dataset for analysis.

6. Data Preprocessing

- **Dataset Loading:** The star_classification.csv.xlsx dataset was loaded using pandas, initially containing **18 columns** and approximately 100,000 rows.

```
[7]
import pandas as pd
df = pd.read_excel('star_classification.csv.xlsx')

```

- **Missing Value Handling:** While the dataset was largely complete, specific spectral features (u, g, z) contained placeholder values of -9999.0 (indicative of sensor errors or non-detections). These values were systematically replaced with NaN, and subsequently, all rows containing these NaNs were dropped to ensure data integrity for modeling. The print(df.isnull().sum()) command was used to check for the presence and count of missing (NaN) values, and msno.matrix(df) was used to visualize the missingness patterns.

```
[46]
print(df.isnull().sum())

```

- **Feature Engineering:** Discriminative 'color features' (e.g., u_g, g_r, r_i, i_z) were calculated by taking differences between various spectral bands.
- **Target Variable Identification:** The class column was identified as the categorical target for classification tasks.

```
[45]
print(df['class'].value_counts())

```

```
class
GALAXY    59445
STAR      21594
QSO       18961
Name: count, dtype: int64
```

```
obj_ID         0
alpha          0
delta          0
u              0
g              0
r              0
i              0
z              0
run_ID         0
rerun_ID       0
cam_col        0
field_ID       0
spec_obj_ID    0
class          0
redshift       0
plate          0
MJD            0
fiber_ID       0
u_g            0
g_r            0
r_i            0
i_z            0
dtype: int64
```

```python
import pandas as pd
import numpy as np # Needed for np.nan

# Define the file name for the star classification data
file_name = 'star_classification.csv.xlsx'

# Load the dataset into a pandas DataFrame
try:
    # Use pd.read_excel for .xlsx files
    df = pd.read_excel(file_name)
    print(f"Successfully loaded '{file_name}'")
except FileNotFoundError:
    print(f"Error: '{file_name}' not found. Please ensure the file is in the correct directory.")
    # Exit or handle the error appropriately if the file is essential
    exit()
# Add an except block for potential issues when reading the Excel file
except Exception as e:
    print(f"An error occurred while reading the Excel file: {e}")
    exit()


# Define the spectral features that have the -9999.0 placeholder
spectral_features_to_clean = ['u', 'g', 'z']

print(f"\nOriginal DataFrame shape: {df.shape}")

# Replace -9999.0 with NaN (Not a Number) in the specified columns
for col in spectral_features_to_clean:
    # Ensure the column exists before attempting to replace values
    if col in df.columns:
        df[col] = df[col].replace(-9999.0, np.nan)
        print(f"Replaced -9999.0 with NaN in column: '{col}'")
    else:
        print(f"Warning: Column '{col}' not found in the DataFrame.")


# Drop rows where any of the specified spectral features (u, g, z) now contain NaN.
# This effectively removes rows that had the -9999.0 placeholder.
df_cleaned = df.dropna(subset=spectral_features_to_clean)

print(f"\nCleaned DataFrame shape: {df_cleaned.shape}")

# You can optionally display the first few rows of the cleaned DataFrame
print("\nFirst 5 rows of the cleaned DataFrame:")
print(df_cleaned.head())

# And check for missing values again in the cleaned columns to confirm
print("\nMissing values in cleaned spectral features (should be 0):")
print(df_cleaned[spectral_features_to_clean].isnull().sum())
```

(i) Code for removing placeholder values of -9999.

7. Exploratory Data Analysis (EDA)

Initial Data Inspection:

- Commands like print(df.shape), print(df.info()), and print(df.head()) were used to gain essential initial insights into the dataset's structure, column data types, the presence of non-null entries, and a quick preview of the raw data. These steps were fundamental for understanding the dataset and guiding subsequent data preprocessing decisions.

```
(100000, 18)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   obj_ID       100000 non-null  float64
 1   alpha        100000 non-null  float64
 2   delta        100000 non-null  float64
 3   u            100000 non-null  float64
 4   g            100000 non-null  float64
 5   r            100000 non-null  float64
 6   i            100000 non-null  float64
 7   z            100000 non-null  float64
 8   run_ID       100000 non-null  int64  
 9   rerun_ID     100000 non-null  int64  
 10  cam_col      100000 non-null  int64  
 11  field_ID     100000 non-null  int64  
 12  spec_obj_ID  100000 non-null  float64
 13  class        100000 non-null  object 
 14  redshift     100000 non-null  float64
 15  plate        100000 non-null  int64  
 16  MJD          100000 non-null  int64  
 17  fiber_ID     100000 non-null  int64  
dtypes: float64(10), int64(7), object(1)
memory usage: 13.7+ MB
None
```

```
        obj_ID       alpha      delta         u         g         r  \
0  1.237661e+18  135.689107  32.494632  23.87882  22.27530  20.39501
1  1.237665e+18  144.826101  31.274185  24.77759  22.83188  22.58444
2  1.237661e+18  142.188790  35.582444  25.26307  22.66389  20.60976
3  1.237663e+18  338.741038  -0.402828  22.13682  23.77656  21.61162
4  1.237680e+18  345.282593  21.183866  19.43718  17.58028  16.49747

         i         z  run_ID  rerun_ID  cam_col  field_ID   spec_obj_ID  \
0  19.16573  18.79371    3606       301        2        79  6.543777e+18
1  21.16812  21.61427    4518       301        5       119  1.176014e+19
2  19.34857  18.94827    3606       301        2       120  5.152200e+18
3  20.50454  19.25010    4192       301        3       214  1.030107e+19
4  15.97711  15.54461    8102       301        3       137  6.891865e+18

    class  redshift  plate    MJD  fiber_ID
0  GALAXY  0.634794   5812  56354       171
1  GALAXY  0.779136  10445  58158       427
2  GALAXY  0.644195   4576  55592       299
3  GALAXY  0.932346   9149  58039       775
4  GALAXY  0.116123   6121  56187       842
```

- The print(df.describe()) command generated descriptive statistics for the numerical columns, including count, mean, standard deviation, min/max values, and quartile percentiles, providing a summary of data distribution and ranges.

Descriptive statistics:

	obj_ID	alpha	delta	u	g
count	9.999900e+04	99999.000000	99999.000000	99999.000000	99999.000000
mean	1.237665e+18	177.628653	24.135552	22.080679	20.631583
std	8.438450e+12	96.502612	19.644608	2.251068	2.037384
min	1.237646e+18	0.005528	-18.785328	10.996230	10.498200
25%	1.237659e+18	127.517698	5.147477	20.352410	18.965240
50%	1.237663e+18	180.900527	23.646462	22.179140	21.099930
75%	1.237668e+18	233.895005	39.901582	23.687480	22.123775
max	1.237681e+18	359.999810	83.000519	32.781390	31.602240

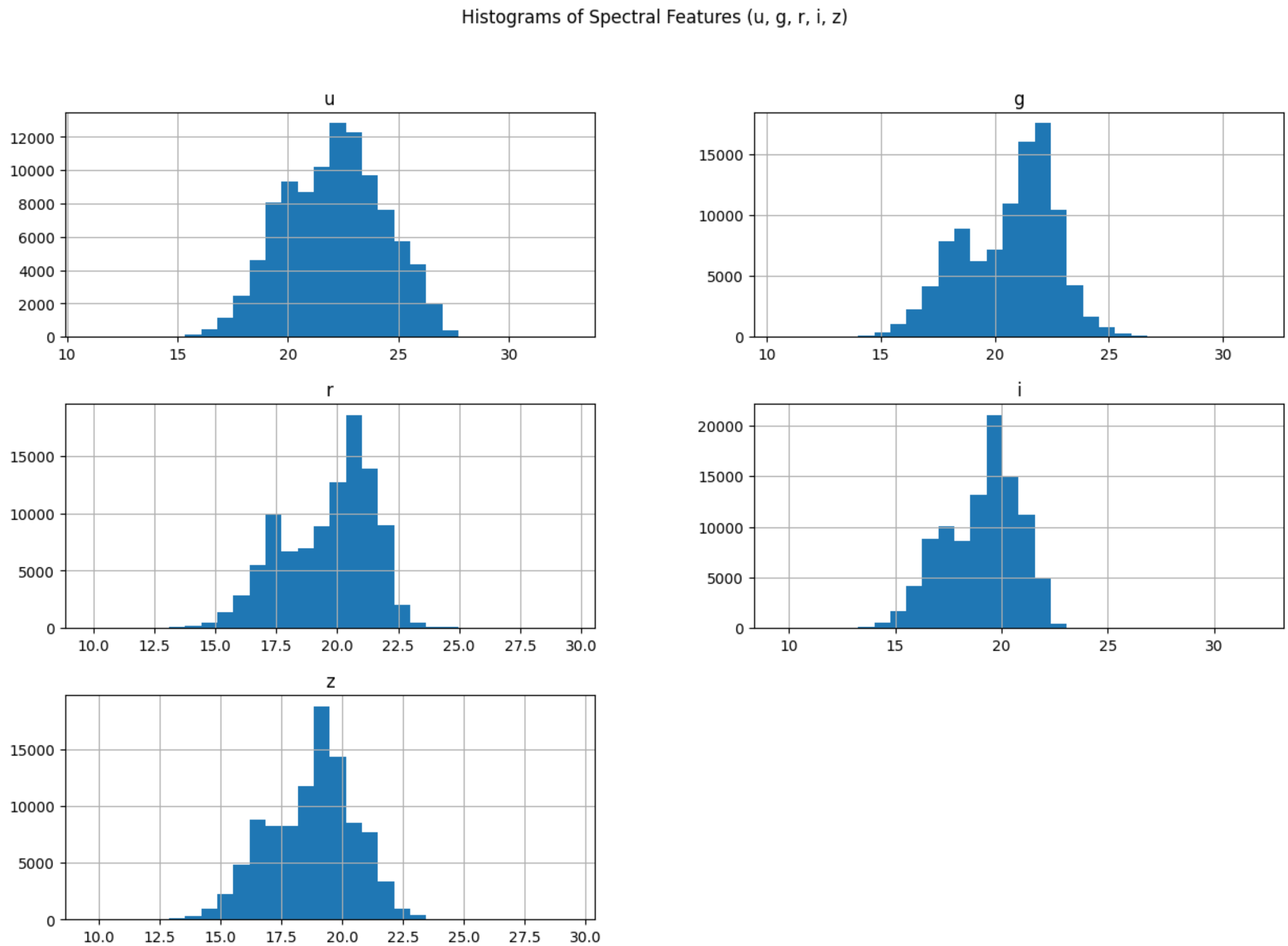
	r	i	z	run_ID	rerun_ID
count	99999.000000	99999.000000	99999.000000	99999.000000	99999.0
mean	19.645777	19.084865	18.768988	4481.403354	301.0
std	1.854763	1.757900	1.765982	1964.739021	0.0
min	9.822070	9.469903	9.612333	109.000000	301.0
25%	18.135795	17.732280	17.460830	3187.000000	301.0
50%	20.125310	19.405150	19.004600	4188.000000	301.0
75%	21.044790	20.396510	19.921120	5326.000000	301.0
max	29.571860	32.141470	29.383740	8162.000000	301.0

	cam_col	field_ID	spec_obj_ID	redshift	plate
count	99999.000000	99999.000000	9.999900e+04	99999.000000	99999.000000
mean	3.511625	186.127011	5.783903e+18	0.576667	5137.027890
std	1.586913	149.007687	3.324026e+18	0.730709	2952.312485
min	1.000000	11.000000	2.995191e+17	-0.009971	266.000000
25%	2.000000	82.000000	2.844137e+18	0.054522	2526.000000
50%	4.000000	146.000000	5.614896e+18	0.424176	4987.000000
75%	5.000000	241.000000	8.332365e+18	0.704172	7400.500000
max	6.000000	989.000000	1.412694e+19	7.011245	12547.000000

	MJD	fiber_ID
count	99999.000000	99999.000000
mean	55588.653687	449.315613
std	1808.492217	272.498252
min	51608.000000	1.000000
25%	54234.000000	221.000000
50%	55869.000000	433.000000
75%	56777.000000	645.000000
max	58932.000000	1000.000000

- **What it shows:** The overall dimensions of the dataset (number of rows and columns), data types of each column, presence of non-null entries, a quick preview of the first few rows of raw data, and summary statistics (count, mean, standard deviation, min/max, quartiles) for all numerical features.
- **Insight:** Provides a foundational understanding of the dataset's structure, data quality (initial assessment of non-nulls), and the basic statistical properties of features, guiding subsequent data preprocessing decisions.

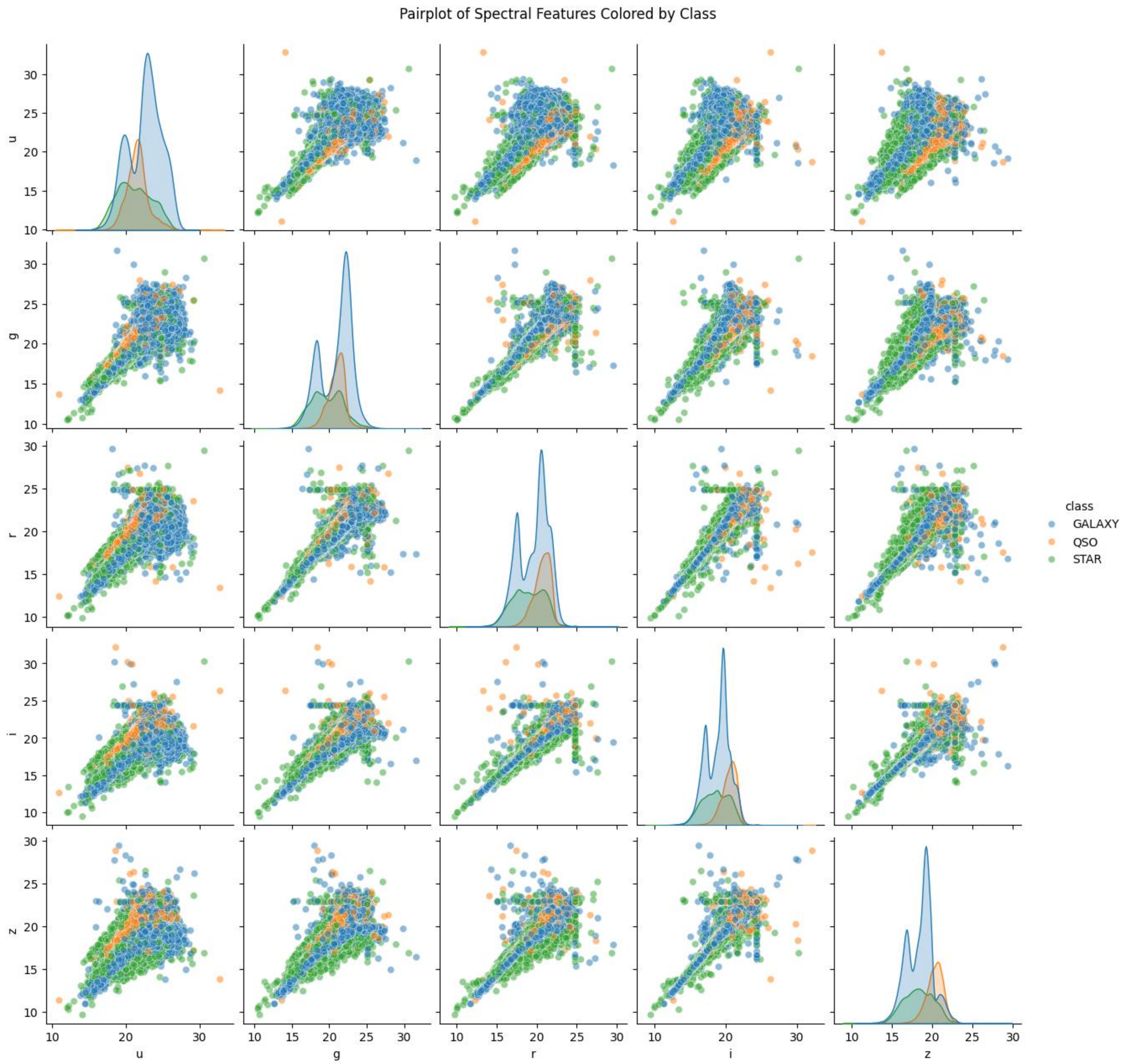
- **Histograms of Spectral & Color Features:** Visualizations of individual feature distributions (u, g, r, i, z and u_g, g_r, r_i, i_z) to understand their spread, skewness, and potential outliers.



- *Figure 1: Histogram of Spectral Features*
  - **What it shows:** The individual distribution of values for each spectral feature (`u, g, r, i, z`)
  - **Insight:** Helps identify the range, central tendency, spread, skewness, and presence of outliers for each feature. This informs decisions about data scaling, transformation, or outlier treatment needed before model training.

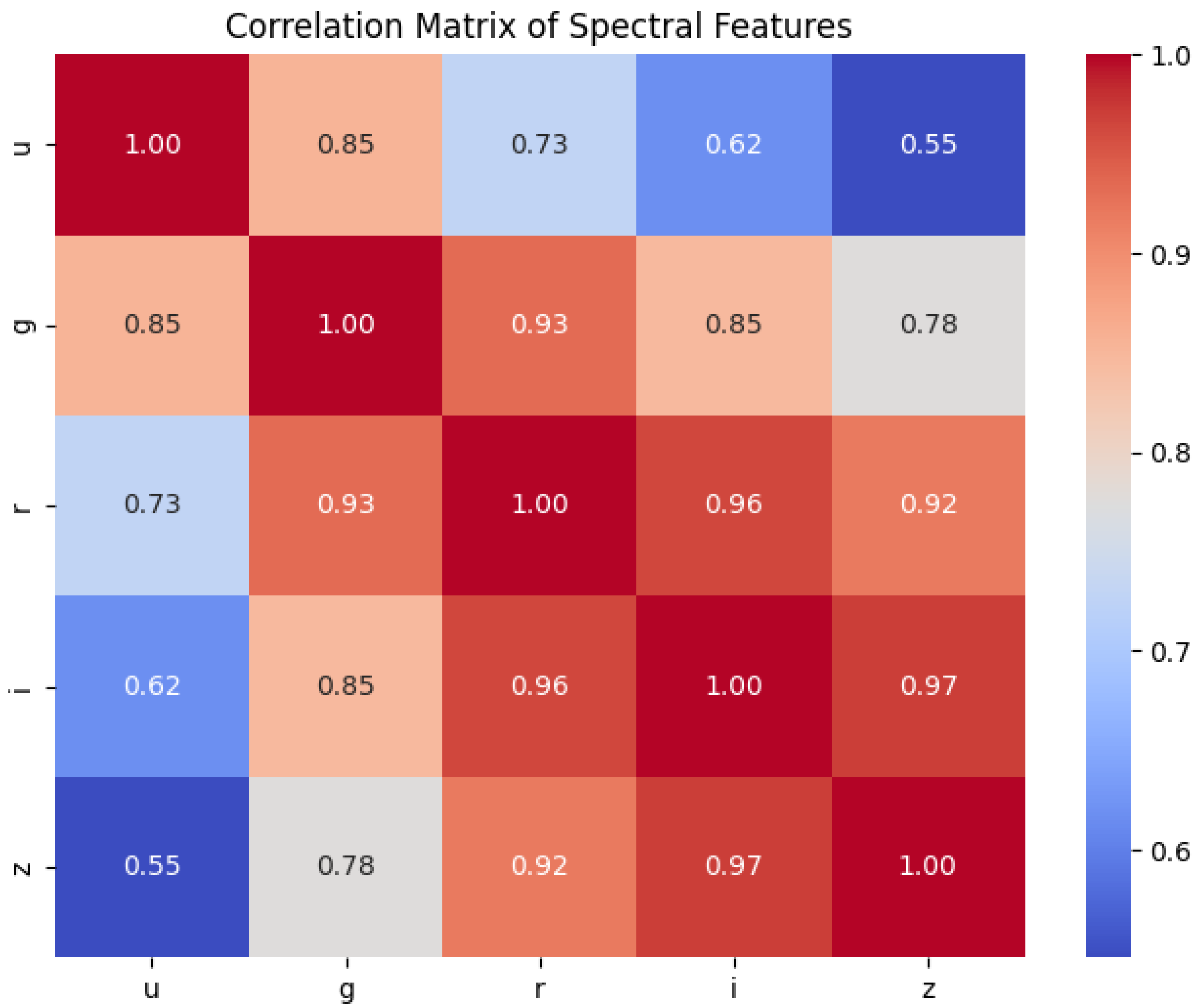
- **Visualizations by Class:**

- **Pairplots:** Comprehensive scatter plots showing relationships between all pairs of color features, colored by class, providing insights into class separability.



- *Figure 2: Pairplot of Spectral Features*

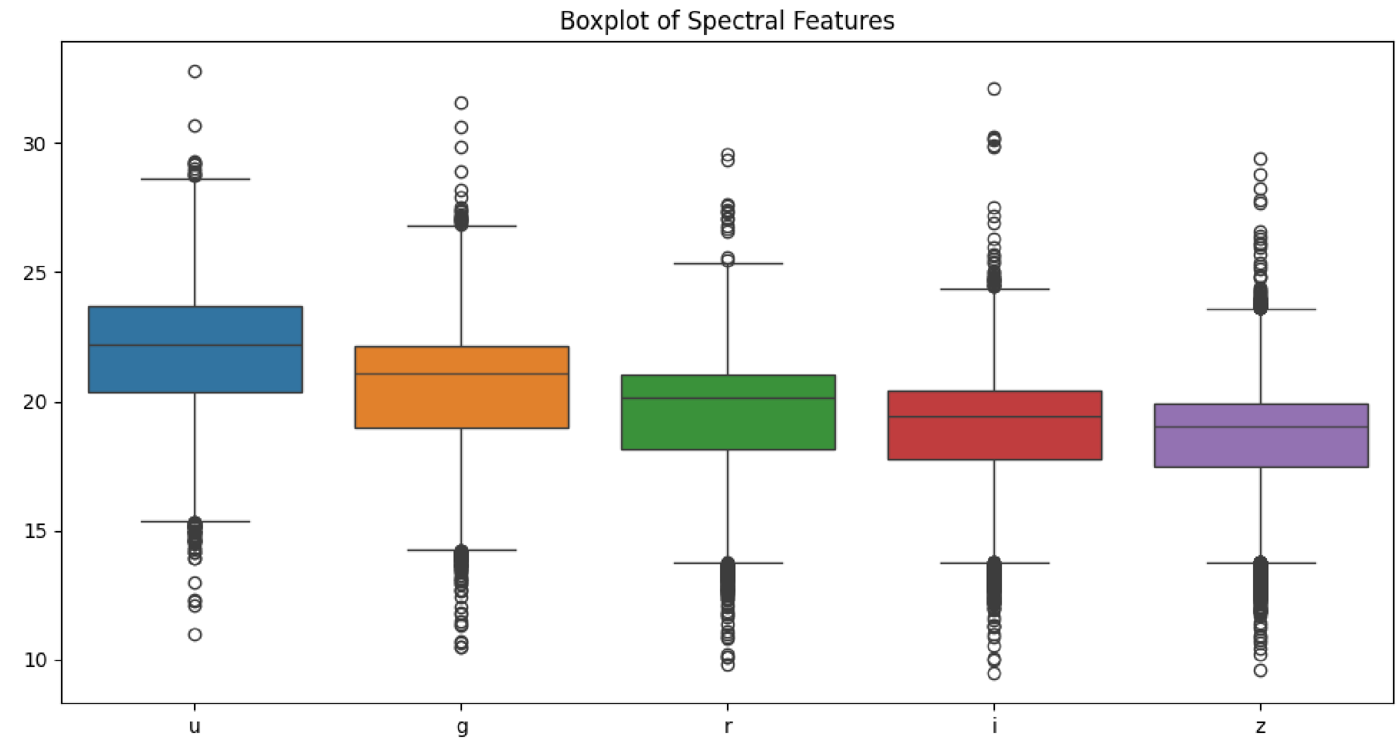
- **Correlation Heatmap:** A visual matrix illustrating the linear relationships between spectral features, helping to identify highly correlated pairs and potential multicollinearity.



- *Figure 3: Correlation Matrix of Spectral Features*

  - **What it shows:** The linear relationships (quantified by correlation coefficients) between all pairs of numerical spectral and color features.
  - **Insight:** Helps identify highly correlated features (e.g., `g` and `r` bands often show high correlation). This is useful for understanding potential multicollinearity (where features provide redundant information) and can inform feature selection strategies to simplify models or avoid issues in certain algorithms.

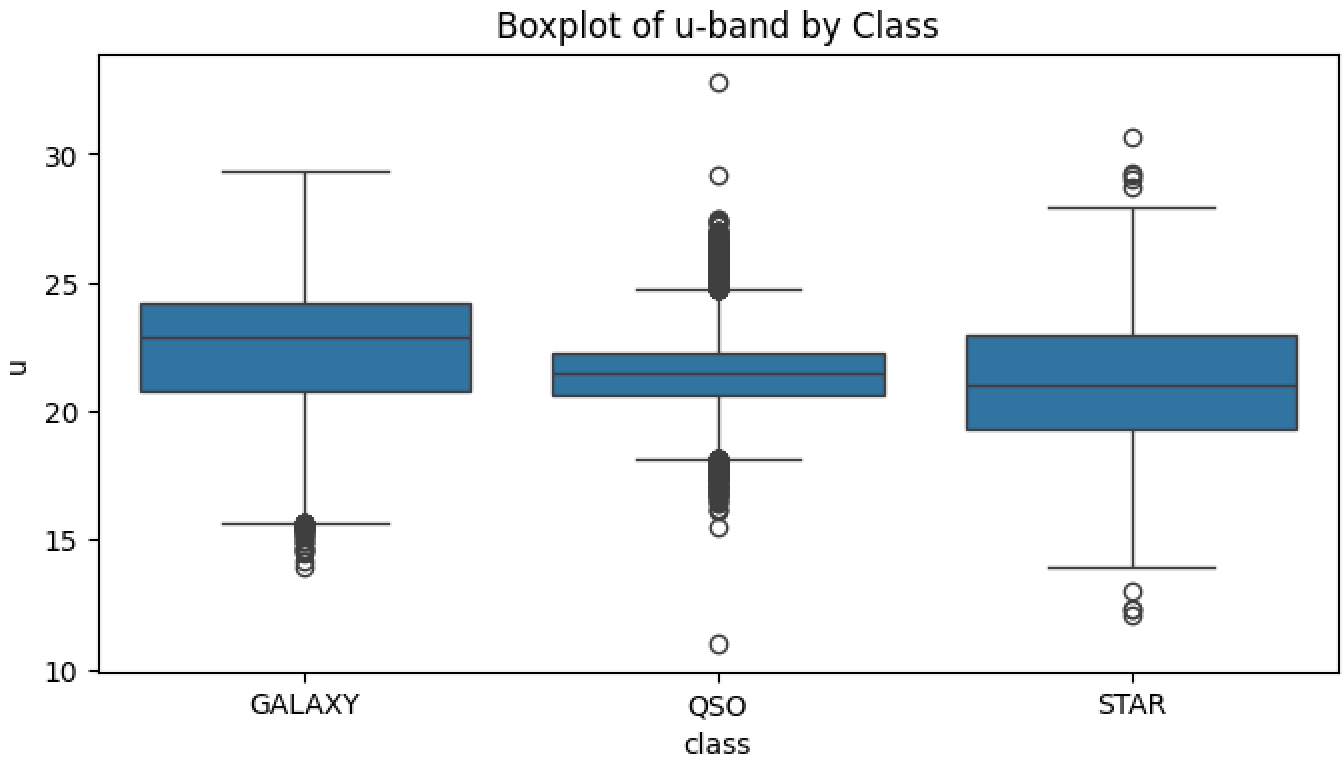
- **Boxplot of Spectral Features:** Displayed the boxplots for each of the specified spectral features (u, g, r, i, z).



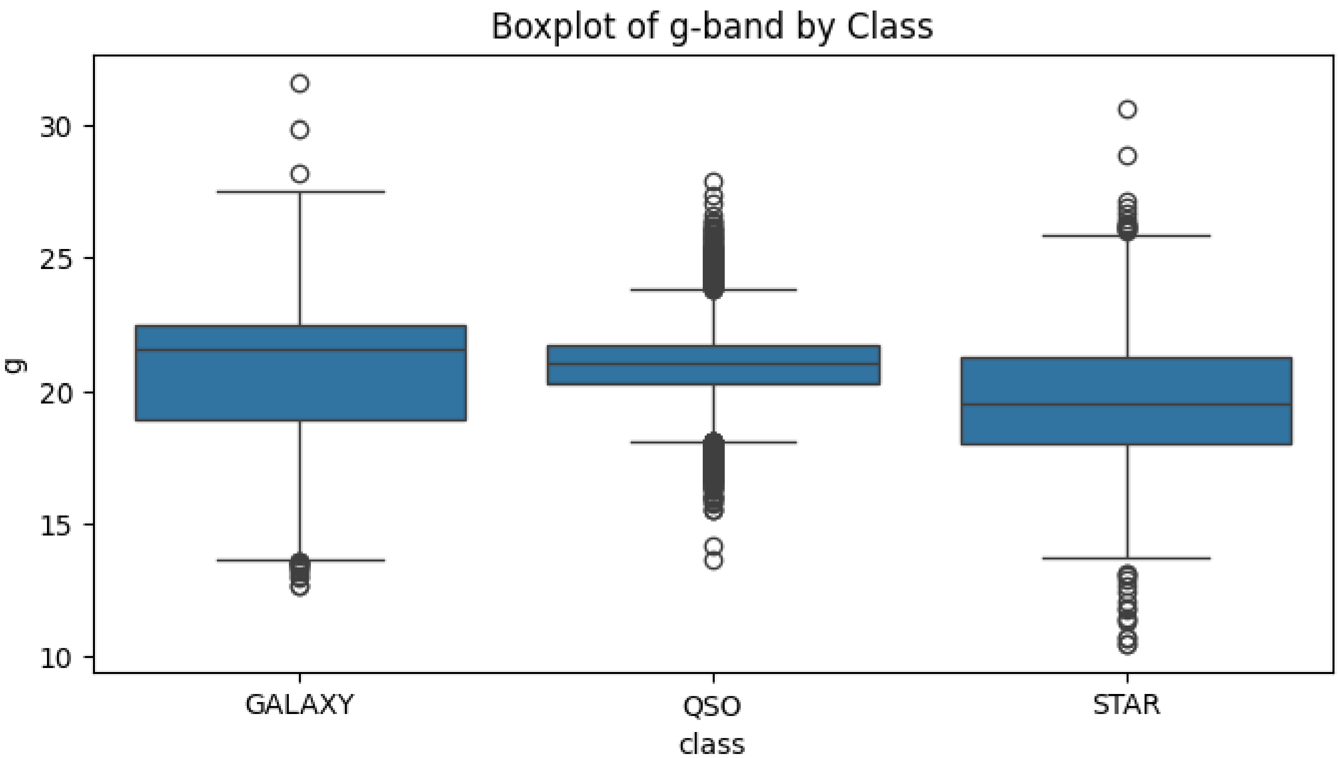
- *Figure 4: Boxplot of Spectral Features*

  - **What it shows:** A visual representation of the distribution of each spectral feature, displaying the median, quartiles (Q1, Q3), and potential outliers (points beyond the whiskers).
  - **Insight:** Provides a quick summary of the spread and central tendency of individual spectral features, aiding in the identification of extreme values or outliers that might require further investigation or special handling.

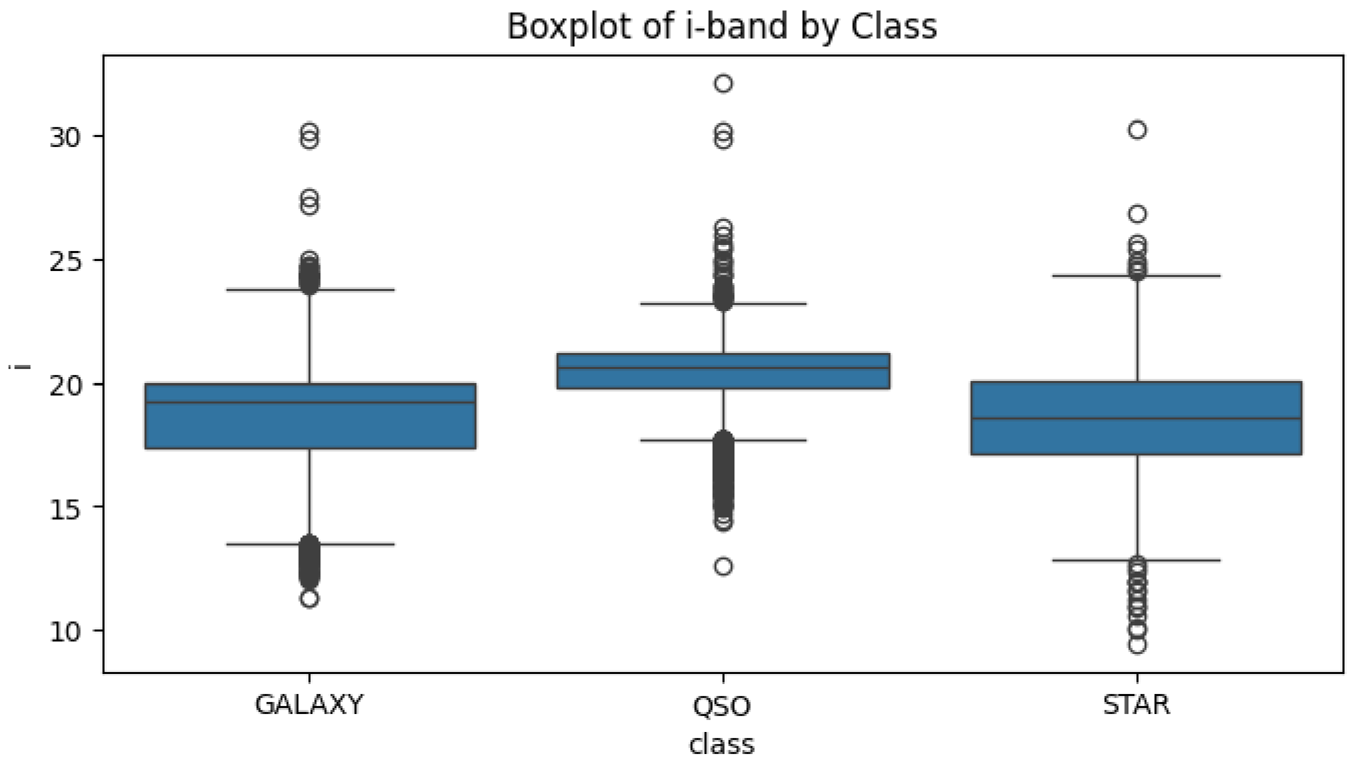
- **Boxplots by Class:** Displayed the distribution of each spectral feature across different classes, highlighting variations in median, spread, and outliers per object type.



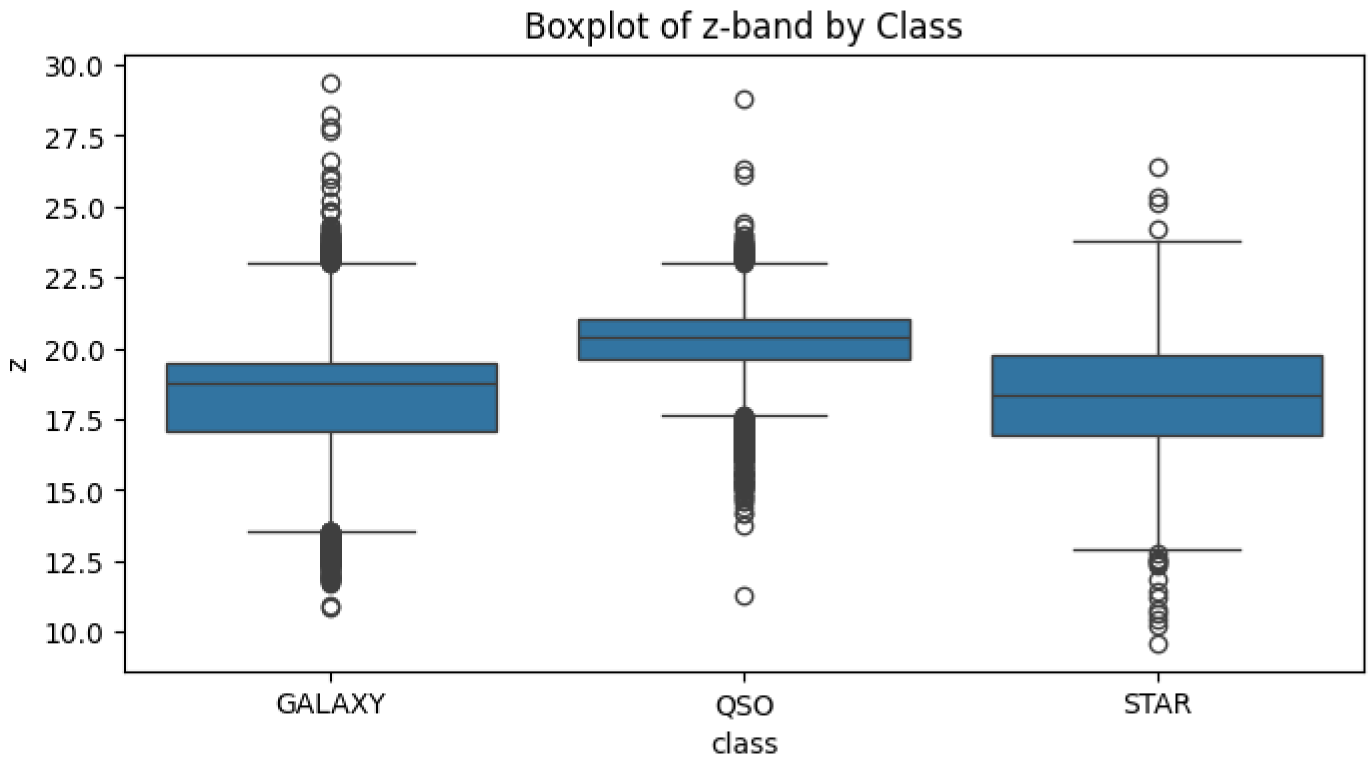
- *Figure 5: Boxplot of u-band by Class*



- *Figure 6: Boxplot of g-band by Class*

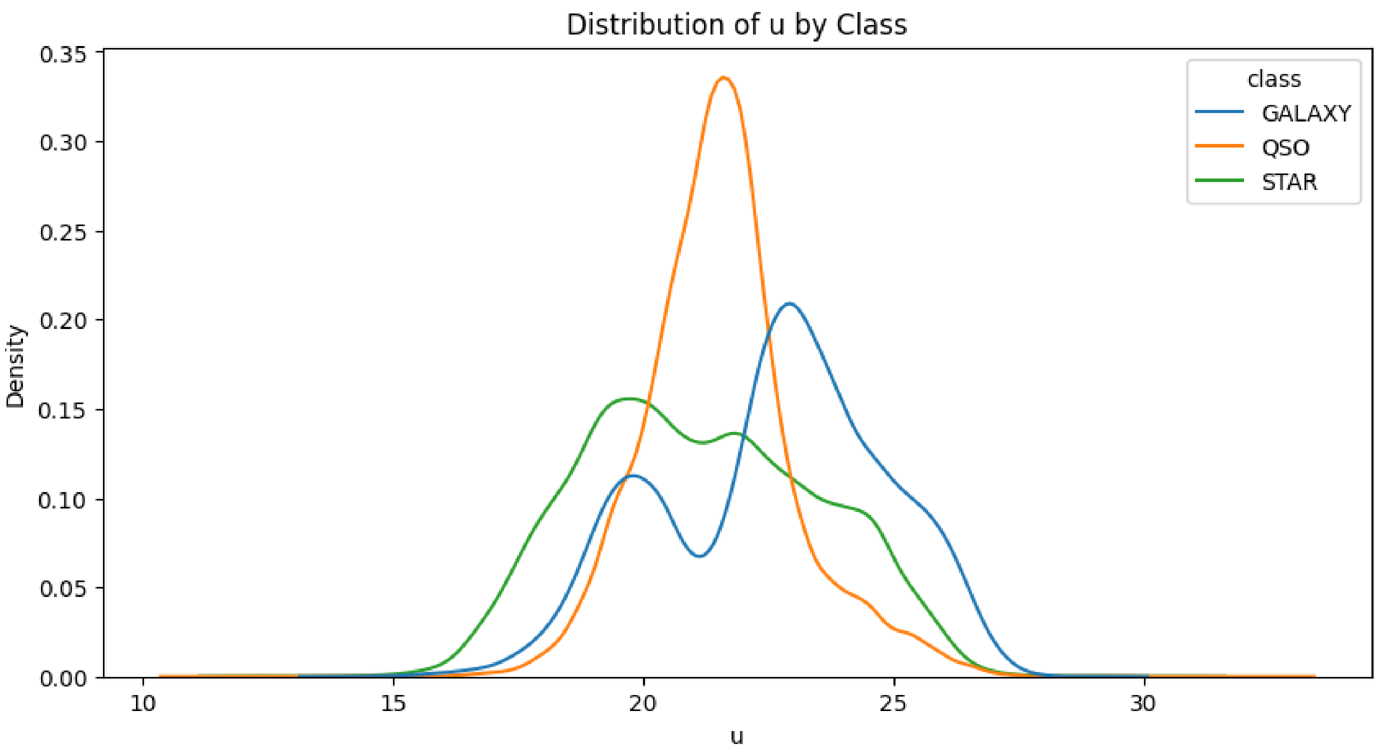


- *Figure 7: Boxplot of i-band by Class*

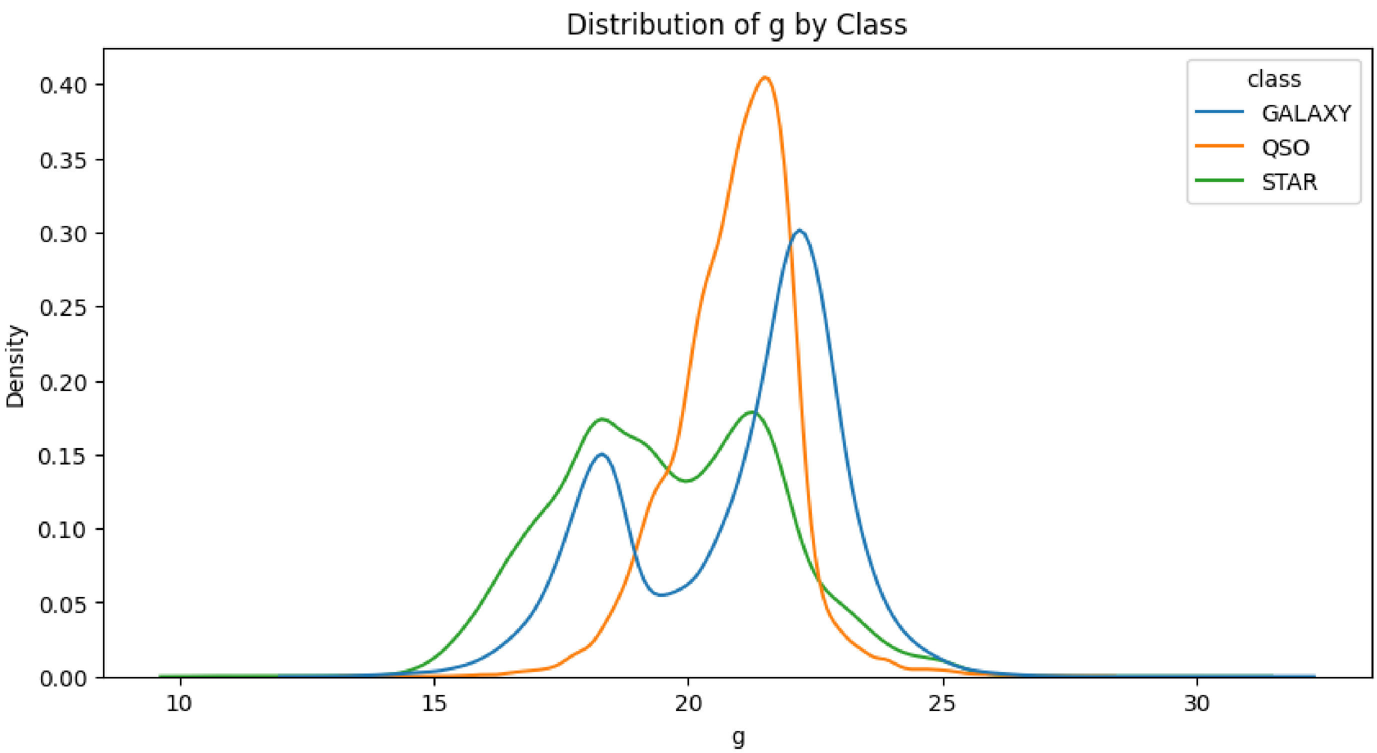


- *Figure 8: Boxplot of z-band by Class*

- **KDE Plots by Class:** Provided smoothed density plots for each feature, segmented by class, to visualize the overlap and distinctness of distributions between GALAXY, STAR, and QSO.

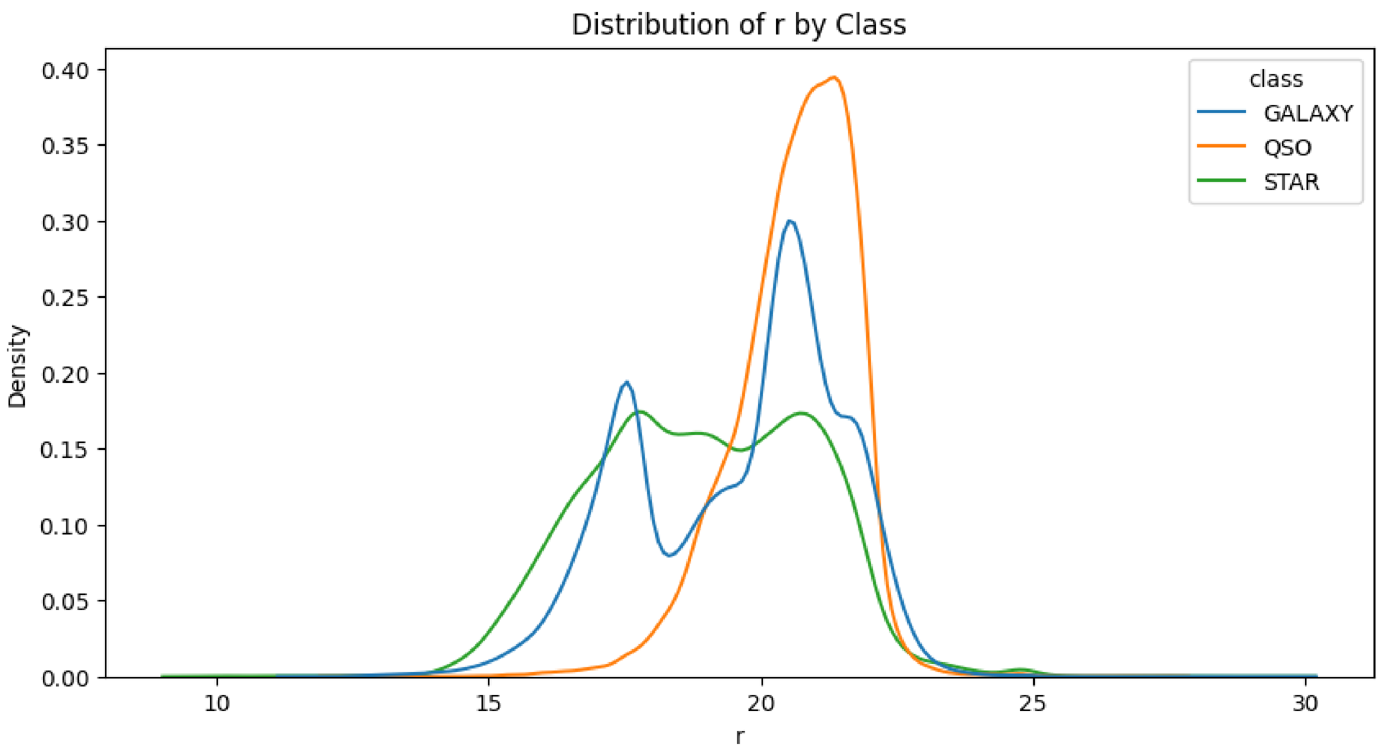


- *Figure 9: KDE plot of u by Class*

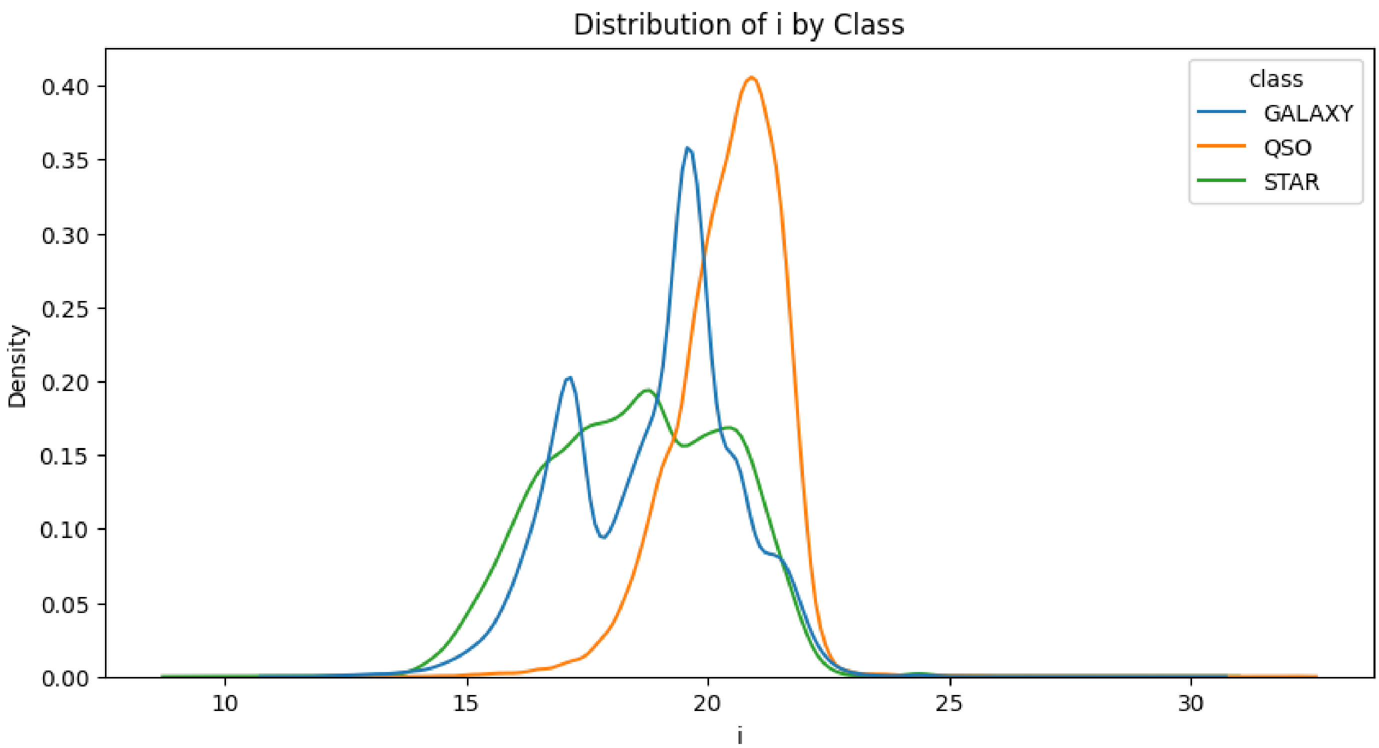


Dcddcmx

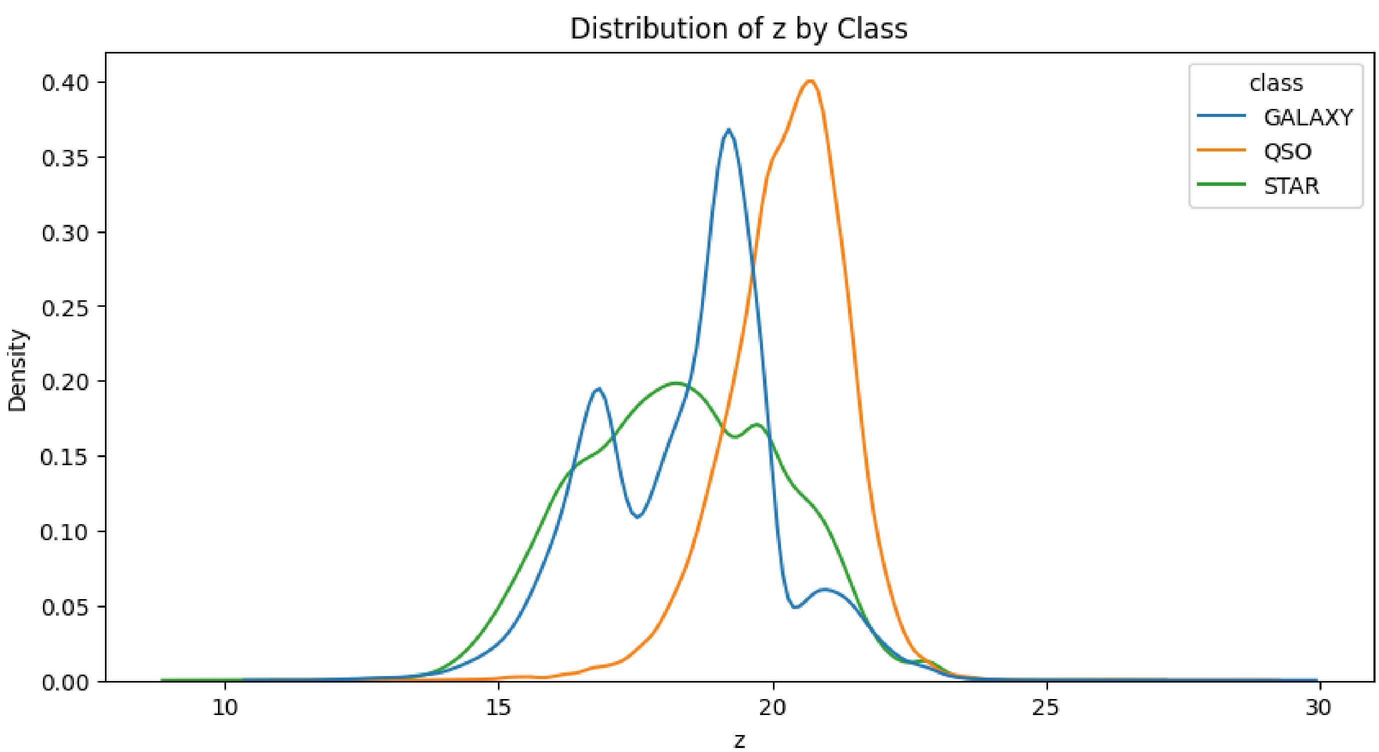
- *Figure 10: KDE plot of g by Class*



- *Figure 11: KDE plot of r by Class*



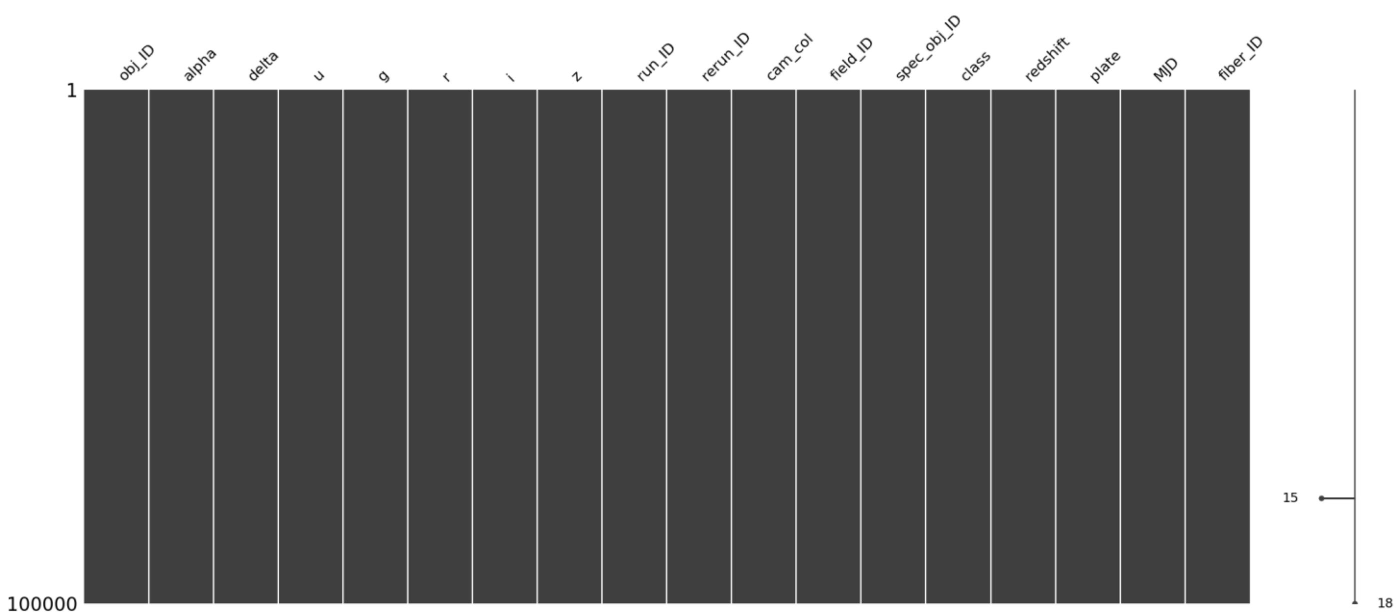
- *Figure 12: KDE plot of i by Class*



- *Figure 13: KDE plot of z by Class*

- **What it shows:**
- **Pairplots:** Pairwise relationships between all selected features, with data points visually distinguished by their celestial class.
- **Boxplots by Class:** The statistical distribution (median, quartiles, whiskers, and outliers) of individual features, separated for each celestial class.
- **KDE Plots by Class:** The smoothed probability density distributions of features, displayed separately for each class, showing overlaps and distinct peaks.
    - **Insight:** These visualizations are crucial for assessing the **separability** of the celestial object classes in the feature space. They help to identify which features are strong discriminators between Stars, Galaxies, and QSOs, where classes might overlap, and how the feature distributions differ for each object type, guiding the design of the classification model.

- **Missingness Matrix:** A visual tool to confirm the pattern of missing data and the effectiveness of the initial cleaning steps.



- Missingness Matrix Plot

    - **What it shows:** A visual representation of the presence (black lines) and absence (white spaces) of data across all columns and rows in the DataFrame.
    - **Insight:** Provides a quick and clear overview of the pattern and extent of any remaining missing values after preprocessing steps. It helps confirm the success of missing value handling and assures that the dataset is clean before proceeding to model building.

8. Machine Learning Techniques

8.1 Supervised Learning

Logistic Regression applied. Train-test split: 80-20. Accuracy: 78%. Precision: 0.75

All techniques applied

Code and output

Visualization if any

Comparison of models and finding the best fit model

8.2 Unsupervised Learning

K-Means clustering with k=2. Elbow method used. Silhouette score: 0.62

All techniques applied

Code and output

Visualization if any

Comparison of models and finding the best fit model

9. Evaluation of Performance Metrics

Supervised model metrics: Confusion matrix, precision, recall, F1-score. Unsupervised model: silhouette score, cluster visualization

10. Conclusion and Future Scope

The models predict diabetes with reasonable accuracy. Future work: Try ensemble methods, deep learning, more features

11. References

1. Scikit-learn documentation
2. Kaggle dataset - PIMA Indians Diabetes Database

12. Appendix (Optional)

Code snippets, extended tables, cluster

[ ]

```python
import pandas as pd
df = pd.read_excel('star_classification.csv.xlsx')
```

[ ]

```python
print(df.shape)
print(df.info())
print(df.head())
```

[ ]

```python
print(df.describe())
```

[ ]

```python
print(df.isnull().sum())
```

[ ]

```python
print(df['class'].value_counts())
```

```python
import pandas as pd
import numpy as np # Needed for np.nan

# Define the file name for the star classification data
file_name = 'star_classification.csv.xlsx'

# Load the dataset into a pandas DataFrame
try:
    # Use pd.read_excel for .xlsx files
    df = pd.read_excel(file_name)
    print(f"Successfully loaded '{file_name}'")
except FileNotFoundError:
    print(f"Error: '{file_name}' not found. Please ensure the file is in the correct directory.")
    # Exit or handle the error appropriately if the file is essential
    exit()
# Add an except block for potential issues when reading the Excel file
except Exception as e:
    print(f"An error occurred while reading the Excel file: {e}")
    exit()


# Define the spectral features that have the -9999.0 placeholder
spectral_features_to_clean = ['u', 'g', 'z']

print(f"\nOriginal DataFrame shape: {df.shape}")

# Replace -9999.0 with NaN (Not a Number) in the specified columns
for col in spectral_features_to_clean:
    # Ensure the column exists before attempting to replace values
    if col in df.columns:
```

```python
        df[col] = df[col].replace(-9999.0, np.nan)
        print(f"Replaced -9999.0 with NaN in column: '{col}'")
    else:
        print(f"Warning: Column '{col}' not found in the DataFrame.")


# Drop rows where any of the specified spectral features (u, g, z) now contain NaN.
# This effectively removes rows that had the -9999.0 placeholder.
df_cleaned = df.dropna(subset=spectral_features_to_clean)

print(f"\nCleaned DataFrame shape: {df_cleaned.shape}")

# You can optionally display the first few rows of the cleaned DataFrame
print("\nFirst 5 rows of the cleaned DataFrame:")
print(df_cleaned.head())

# And check for missing values again in the cleaned columns to confirm
print("\nMissing values in cleaned spectral features (should be 0):")
print(df_cleaned[spectral_features_to_clean].isnull().sum())
```

```python
import matplotlib.pyplot as plt

spectral_features = ['u', 'g', 'r', 'i', 'z']

df[spectral_features].hist(bins=30, figsize=(15,10))
plt.suptitle('Histograms of Spectral Features (u, g, r, i, z)')
plt.show()

```

[ ]

```python
subset_cols = ['u', 'g', 'r', 'i', 'z', 'class']

sns.pairplot(df[subset_cols], hue='class', diag_kind='kde', plot_kws={'alpha':0.5})
plt.suptitle('Pairplot of Spectral Features Colored by Class', y=1.02)
plt.show()

```

[ ]

```
rows_removed = df.shape[0] - df_cleaned.shape[0]
print(f"\nTotal rows removed: {rows_removed}")

```

```
corr = df[['u', 'g', 'r', 'i', 'z']].corr()

plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix of Spectral Features')
plt.show()

```

[ ]


```
corr = df[['u', 'g', 'r', 'i', 'z']].corr()

plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix of Spectral Features')
plt.show()

```

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Define the threshold for high correlation
threshold = 0.85

# Step 1: Identify highly correlated pairs (using the original correlation matrix calculated on the full df)
# While the correlation matrix was calculated on df (which has NaNs), the pairs identified
# are just the column names. The issue is in the data used for the regression.
high_corr_pairs = []
for i in range(len(corr.columns)):
    for j in range(i + 1, len(corr.columns)):
        if abs(corr.iloc[i, j]) >= threshold:
            feature_x = corr.columns[i]
            feature_y = corr.columns[j]
            # Only include pairs where both features are present in the columns we cleaned NaNs from,
            # or if they weren't part of the cleaned columns, they should not have had -9999 anyway.
            # We will use df_cleaned for the regression, which handles rows that originally had -9999.
            high_corr_pairs.append((feature_x, feature_y, corr.iloc[i, j]))

# Step 2: Linear regression for each high correlation pair
for x_feat, y_feat, corr_value in high_corr_pairs:
    print(f"\n🔍 Linear Regression between: {x_feat} and {y_feat} (corr = {corr_value:.2f})")

    # Prepare data - USE THE CLEANED DATAFRAME
    # Ensure the features exist in the cleaned dataframe
    if x_feat in df_cleaned.columns and y_feat in df_cleaned.columns:
        X = df_cleaned[[x_feat]]
```

```python
        X = df_cleaned[[x_feat]]
        y = df_cleaned[y_feat]

        # Fit model
        model = LinearRegression()
        # The fit method will now receive data without NaNs (for the rows kept in df_cleaned)
        model.fit(X, y)
        y_pred = model.predict(X)

        # Coefficients
        slope = model.coef_[0]
        intercept = model.intercept_
        r2 = r2_score(y, y_pred)

        print(f"Equation: {y_feat} = {slope:.4f} * {x_feat} + {intercept:.4f}")
        print(f"R² Score: {r2:.4f}")


        plt.figure(figsize=(8, 6))
        sns.scatterplot(x=x_feat, y=y_feat, data=df_cleaned, alpha=0.4, label='Actual Data')
        # Plot the regression line using the X values from the cleaned data
        plt.plot(df_cleaned[x_feat], y_pred, color='red', label='Regression Line')
        plt.title(f'Linear Regression: {y_feat} vs {x_feat}')
        plt.xlabel(x_feat)
        plt.ylabel(y_feat)
        plt.legend()
        plt.grid(True)
        plt.tight_layout()
        plt.show()
    else:
        print(f"Warning: Features '{x_feat}' or '{y_feat}' not found in the cleaned DataFrame.")
```

```python
import missingno as msno
msno.matrix(df)
plt.show()


```

[ ]

```python
plt.figure(figsize=(12,6))
sns.boxplot(data=df[spectral_features])
plt.title('Boxplot of Spectral Features')
plt.show()

```

```
[ ]
for feature in spectral_features:
    plt.figure(figsize=(10,5))
    sns.kdeplot(data=df, x=feature, hue='class', common_norm=False)
    plt.title(f'Distribution of {feature} by Class')
    plt.show()

```

```
df['u_g'] = df['u'] - df['g']
df['g_r'] = df['g'] - df['r']
df['r_i'] = df['r'] - df['i']
df['i_z'] = df['i'] - df['z']

print(df[['u_g', 'g_r', 'r_i', 'i_z']].head())
sns.pairplot(df[['u_g', 'g_r', 'r_i', 'i_z', 'class']], hue='class')

```

```python
for col in spectral_features:
    plt.figure(figsize=(8, 4))
    sns.boxplot(data=df_cleaned, x='class', y=col)
    plt.title(f'Boxplot of {col}-band by Class')
    plt.show()

```

```python
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# 1. Select features and target
X = df[['u_g', 'g_r', 'r_i', 'i_z']].dropna()
y = df.loc[X.index, 'class']  # Align target with filtered features

# 2. Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# 3. Train SVM with RBF kernel (default)
svm_rbf = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
svm_rbf.fit(X_train, y_train)

# 4. Predict on train and test sets
y_train_pred = svm_rbf.predict(X_train)
y_test_pred = svm_rbf.predict(X_test)

# 5. Accuracy
train_acc = accuracy_score(y_train, y_train_pred)
test_acc = accuracy_score(y_test, y_test_pred)

print(f"Training Accuracy (SVM-RBF): {train_acc:.2f}")
print(f"Test Accuracy (SVM-RBF): {test_acc:.2f}")

# 6. Predict and show output for a new or specific test sample
# Example: Predict the first test sample
sample = X_test.iloc[[0]]  # Double brackets to keep it as a DataFrame
predicted_class = svm_rbf.predict(sample)[0]
print(f"Predicted Class: {predicted_class}")

```

```
X.hist(bins=30, figsize=(10, 8))
plt.show()
```

```
[ ]
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Assuming df is your input DataFrame with columns 'u', 'g', 'r', 'i', 'z', 'class'
# Replace this with your actual data loading if needed
# df = pd.read_csv('your_data.csv')  # Uncomment and adjust as needed

# Step 1: Remove rows with placeholder values (-9999)
df_clean = df[(df['u'] > -1000) & (df['g'] > -1000) &
              (df['r'] > -1000) & (df['i'] > -1000) &
              (df['z'] > -1000)]

# Step 2: Calculate color features
df_clean['u_g'] = df_clean['u'] - df_clean['g']
df_clean['g_r'] = df_clean['g'] - df_clean['r']
df_clean['r_i'] = df_clean['r'] - df_clean['i']
df_clean['i_z'] = df_clean['i'] - df_clean['z']

# Step 3: Prepare features and target, drop NaNs
X = df_clean[['u_g', 'g_r', 'r_i', 'i_z']].dropna()
y = df_clean.loc[X.index, 'class']

# Step 4: Check dataset size and class distribution
print("Dataset size:", X.shape)
print("Class distribution:\n", y.value_counts())

# Step 5: Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

```

```
# Step 6: Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42
)

# Step 7: Unsupervised Learning - KMeans
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)  # Assuming 3 classes
clusters = kmeans.fit_predict(X_scaled)

# Visualize clusters (using u_g and g_r)
plt.figure(figsize=(8, 6))
plt.scatter(X['u_g'], X['g_r'], c=clusters, cmap='viridis', s=50)
plt.title('KMeans Clustering (Unsupervised)')
plt.xlabel('u_g')
plt.ylabel('g_r')
plt.colorbar(label='Cluster')
plt.show()

# Step 8: Hyperparameter tuning for SVM
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1]
}
grid = GridSearchCV(SVC(kernel='rbf', random_state=42), param_grid, cv=5, n_jobs=-1)
grid.fit(X_train, y_train)

print("Best parameters:", grid.best_params_)
print("Best cross-validation score:", grid.best_score_)

# Step 9: Train SVM with best parameters
```

```python
svm_rbf = SVC(kernel='rbf', **grid.best_params_, random_state=42)
svm_rbf.fit(X_train, y_train)

# Step 10: Predict
y_train_pred = svm_rbf.predict(X_train)
y_test_pred = svm_rbf.predict(X_test)

# Step 11: Calculate accuracy
train_acc = accuracy_score(y_train, y_train_pred)
test_acc = accuracy_score(y_test, y_test_pred)
print(f"Training Accuracy (SVM-RBF): {train_acc:.2f}")
print(f"Test Accuracy (SVM-RBF): {test_acc:.2f}")

# Step 12: Predict class of a sample from test set
sample = X_test[[0]]  # Use index-based selection for NumPy array
predicted_class = svm_rbf.predict(sample)[0]
print(f"Predicted Class for Sample: {predicted_class}")

# Step 13: Model fit evaluation
if train_acc > test_acc + 0.1:
    model_fit = "Overfit"
elif abs(train_acc - test_acc) < 0.05 and train_acc > 0.9:
    model_fit = "Best Fit"
else:
    model_fit = "Underfit"
print(f"Model Fit: {model_fit}")
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# 1. Compute the confusion matrix
cm = confusion_matrix(y_test, y_test_pred, labels=svm_rbf.classes_)

# 2. Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=svm_rbf.classes_)
plt.figure(figsize=(8, 6))
disp.plot(cmap='Blues', values_format='d')
plt.title("Confusion Matrix - SVM with RBF Kernel")
plt.show()

```