Configuration Manual

MSc Research Project
MSc in Data Analytics

Omkar Saurabh Parkar
Student ID: x22195777

School of Computing
National College of Ireland

Supervisor: Professor John Kelly

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	…….Omkar Saurabh Parkar…………………………………….
Student ID:	x22195777 …………………………………………………………………………………………….……
Programme:	MSc in Data Analytics ……………………………………………………… **Year:** 2023 …………………………..
Module:	……MSc Research Project………………………………………………………………………
Lecturer:	John Kelly ………………………………………………………………………………………………………
Submission Due Date:	……12/08/2024……………………………………………………………………………………….
Project Title:	… Deep Learning for Galaxy Morphology Classification in Large-Scale Surveys ……
Word Count:	………1274………… **Page Count:** …………19.……..………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……………………………Omkar Saurabh Parkar……………………………….

Date: ………………………………12/08/2024…………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual for Galaxy Morphology Classification

Omkar Saurabh Parkar
Student ID: x22195777
MSc in Data Analytics
Supervisor: John Kelly

National College of Ireland
School of Computing

1. Introduction

This configuration manual provides detailed instructions on setting up and running the deep learning framework for galaxy morphology classification using the provided code. The project uses the data from the Sloan Digital Sky Survey (SDSS) and uses the Convolutional Neural Networks (CNNs) to classify galaxies based on their morphological features.

2. System Requirements and Libraries

This section provides the details of Software and Hardware requirements to implement the research done.

Component	Specification
Operating System	Windows 10, macOS, or Linux
Processor	Intel i5 or higher
Memory	8 GB RAM or more
Storage	20 GB free disk space
Graphics	NVIDIA GPU with CUDA support (recommended)

Software/Library	Version/Command
Python	3.7 or higher
Jupyter Notebook	Latest
numpy	pip install numpy

pandas	pip install pandas
seaborn	pip install seaborn
matplotlib	pip install matplotlib
tqdm	pip install tqdm
xgboost	pip install xgboost
scikit-learn	pip install scikit-learn
imbalanced-learn	pip install imbalanced-learn
tensorflow/pytorch	pip install tensorflow or pip install torch

3. Data and Execution

3.1. Dataset

The dataset used in this research report is taken from large-scale astronomical surveys like the Sloan Digital Sky Survey (SDSS). The SDSS is one of the most extensive and detailed astronomical surveys available. This survey includes a multi-spectral photometric and spectroscopic data which covers a broad range of wavelengths. It also captures detailed images of millions of celestial objects. The key characteristics of the SDSS data include:

- Photometric Data: This includes the measurements of five spectral bands (u, g, r, i, z).
- Spectroscopic Data: This includes the detailed spectra of galaxies, stars, and quasars, and their redshift measurements.
- Positional Data: This includes the astronomical coordinates like right ascension and declination.
- Observational Metadata: This includes the information about the observation conditions, such as run number, camcol, field, and Modified Julian Date (MJD).

Data Component	**Description**
Data Files	galaxies.csv: Main dataset with attributes
	images/: Directory containing galaxy images

```python
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import tqdm
import xgboost as xgb
import matplotlib as mpl
import matplotlib.pyplot as plt
from collections import Counter
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


mpl.rcParams['figure.dpi'] = 300
warnings.filterwarnings("ignore")
```

Figure 1: Importing all the necessary libraries

```python
# Load the dataset into a pandas DataFrame
data = pd.read_csv("Skyserver_SQL7_13_2024 5_03_14 PM.csv", delimiter=',', header=0)

# Print the first few rows of the DataFrame to confirm it's loaded correctly
print("First few rows of the dataset:")
display(data.head())
```

Figure 2: Loading the data into 'data' variable and then displaying it

First few rows of the dataset:

	objid	ra	dec	u	g	r	i	z	run	rerun	camcol	field	specobjid	class	redshift	plate	mjd	fiberid
0	1.237660e+18	243.022574	4.385969	19.15551	17.43852	16.77859	16.57280	16.44750	3910	301	4	218	2.452360e+18	STAR	0.000036	2178	54629	550
1	1.237660e+18	243.432054	4.313188	19.03519	17.47085	16.86022	16.63442	16.49818	3910	301	4	221	2.452380e+18	STAR	-0.000138	2178	54629	634
2	1.237650e+18	148.591375	4.751633	19.05938	17.57685	17.09509	16.75525	16.62675	2126	301	5	197	6.430470e+17	GALAXY	0.082003	571	52286	576
3	1.237670e+18	257.734048	42.802318	19.20997	18.89720	18.81041	18.93487	18.85936	5327	301	3	12	9.606200e+18	QSO	0.815274	8532	58022	62
4	1.237670e+18	261.272024	36.439204	18.71752	17.09335	16.40921	16.08185	15.93252	5327	301	3	59	3.706590e+18	STAR	-0.000162	3292	54943	451

Figure 3: First few rows of the dataset

3.2. Data Preparation

Step	Description	Code Reference
Data Cleaning	Remove duplicates and handle missing values	data.drop_duplicates(), data.fillna()

Normalization	Scale features to [0, 1] or standardize	StandardScaler()
Data Augmentation	Apply transformations like rotation, flipping	ImageDataGenerator
Splitting the Data	Split into training (70%), validation (15%), test (15%)	train_test_split()
Class Imbalance Handling	Use SMOTE for synthetic sampling of minority classes	SMOTE()

```
# Print the shape of the dataset
print("Shape of the dataset:")
print(data.shape)
```

```
Shape of the dataset:
(500000, 18)
```

Figure 4: Exploring the shape of the dataset

```
# Print the column names
print("Columns in the dataset:")
print(data.columns)
```

```
Columns in the dataset:
Index(['objid', 'ra', 'dec', 'u', 'g', 'r', 'i', 'z', 'run', 'rerun', 'camcol',
       'field', 'specobjid', 'class', 'redshift', 'plate', 'mjd', 'fiberid'],
      dtype='object')
```

Figure 5: Printing the column names

```
# Print the sum of null values in each column, sorted by the number of nulls
print("Number of null values in each column, sorted:")
print(data.isnull().sum().sort_values(ascending=False))
```

```
Number of null values in each column, sorted:
objid        0
ra           0
mjd          0
plate        0
redshift     0
class        0
specobjid    0
field        0
camcol       0
rerun        0
run          0
z            0
i            0
r            0
g            0
u            0
dec          0
fiberid      0
dtype: int64
```

Figure 6: After cleaning the null values inside each column

```
# Print the count of exactly duplicate rows
print("Count of exactly duplicate rows:", data.duplicated().sum())
```

```
Count of exactly duplicate rows: 0
```

Figure 7: Number of duplicate rows

3.3. Model Training

Step	Description	Code Reference
Loading Data	Load dataset and images	pd.read_csv(), load_img()
Building Model	Define CNN architecture	tf.keras.models.Sequential
Training Model	Compile and train the model	model.fit()

Evaluation	Evaluate model on test set using various metrics	model.evaluate(), classification_report()

```python
# Setting the aesthetic style for the plots using Matplotlib
plt.style.use('seaborn-whitegrid')

# Sampling a subset of the data for quicker visualization, e.g., 10% of the data
sampled_data = data.sample(frac=0.1, random_state=1)

# Defining features
features = ['u', 'g', 'r', 'i', 'z', 'redshift']

# Initializing the figure
plt.figure(figsize=(15, 10))

# Generating histograms using Pandas' built-in function
for i, feature in enumerate(tqdm(features, desc="Generating Histograms")):
    ax = plt.subplot(2, 3, i + 1)  # Positioning the subplot
    sampled_data[feature].plot(kind='hist', bins=30, alpha=0.7, color='skyblue', ax=ax, density=True)
    sampled_data[feature].plot(kind='kde', color='darkblue', ax=ax)  # Adding a Kernel Density Estimate plot
    ax.set_title(f'Distribution of {feature}')
    ax.set_xlabel(f'{feature} Value')
    ax.set_ylabel('Density')

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

Figure 8: Univariate analysis of creating histograms and bar charts

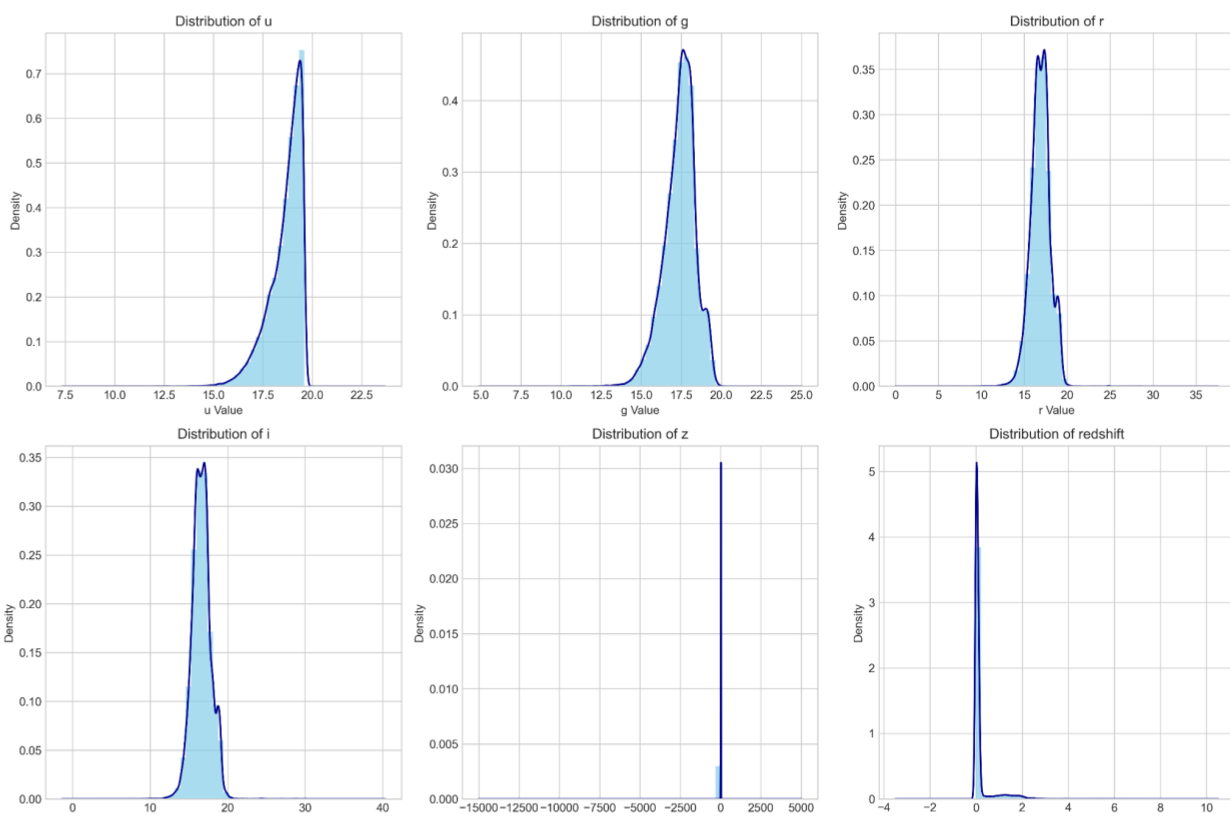

Figure 9: Histogram depicting the distribution (u, g, r, i, z)

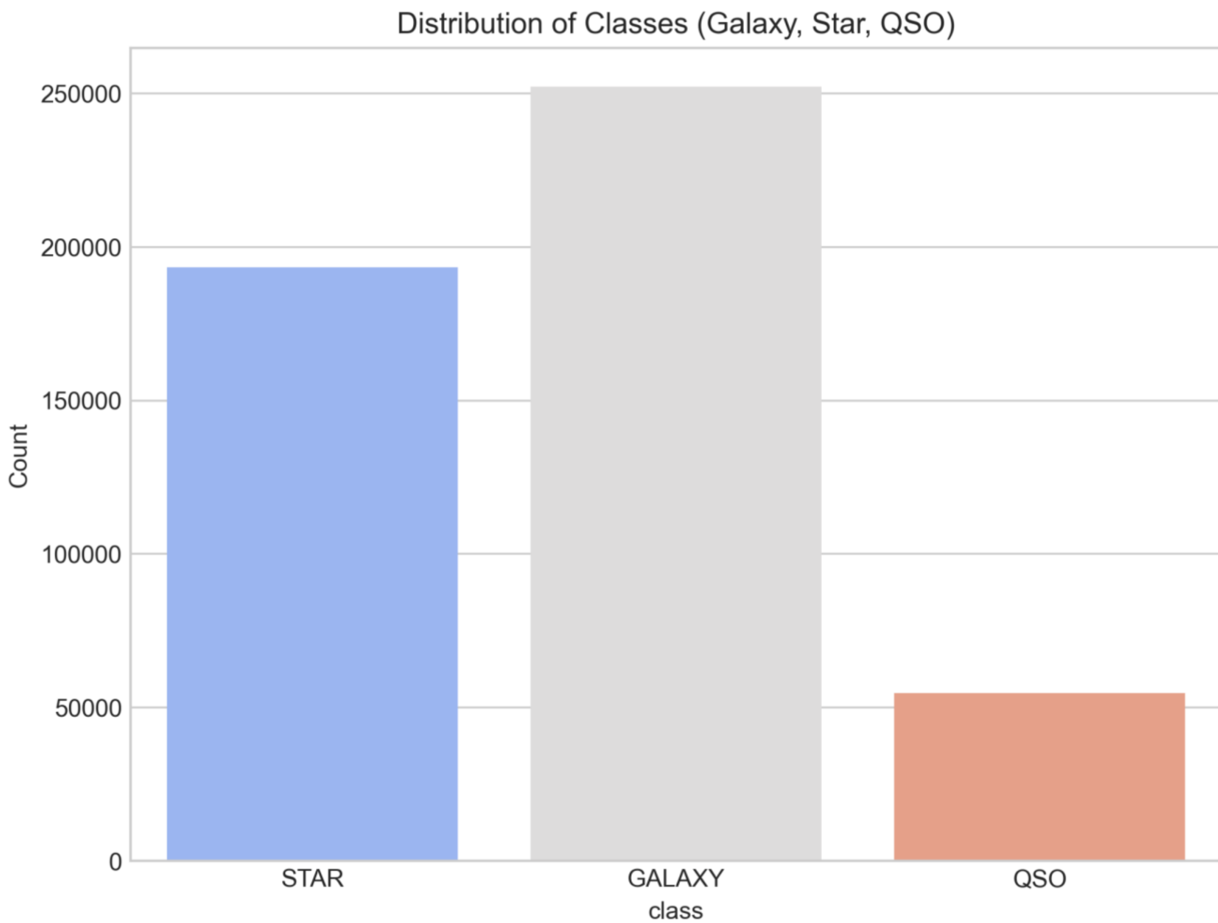

Figure 10: Bar chart of distribution of classes

```python
# Defining feature pairs for scatter plots
pairs = [('ra', 'dec'), ('u', 'redshift')]

# Generating scatter plots for each pair of features
for x, y in tqdm(pairs, desc="Generating Scatter Plots"):
    plt.figure(figsize=(10, 5))  # Specifying the size of the figure
    sns.scatterplot(x=x, y=y, data=sampled_data, hue='class', style='class', alpha=0.6)
    plt.title(f'{x} vs {y}')  # Setting the title to indicate which features are being plotted
    plt.xlabel(x)  # Labeling the x-axis
    plt.ylabel(y)  # Labeling the y-axis
    plt.show()
```

Figure 11: Bivariate analysis of the classes distribution

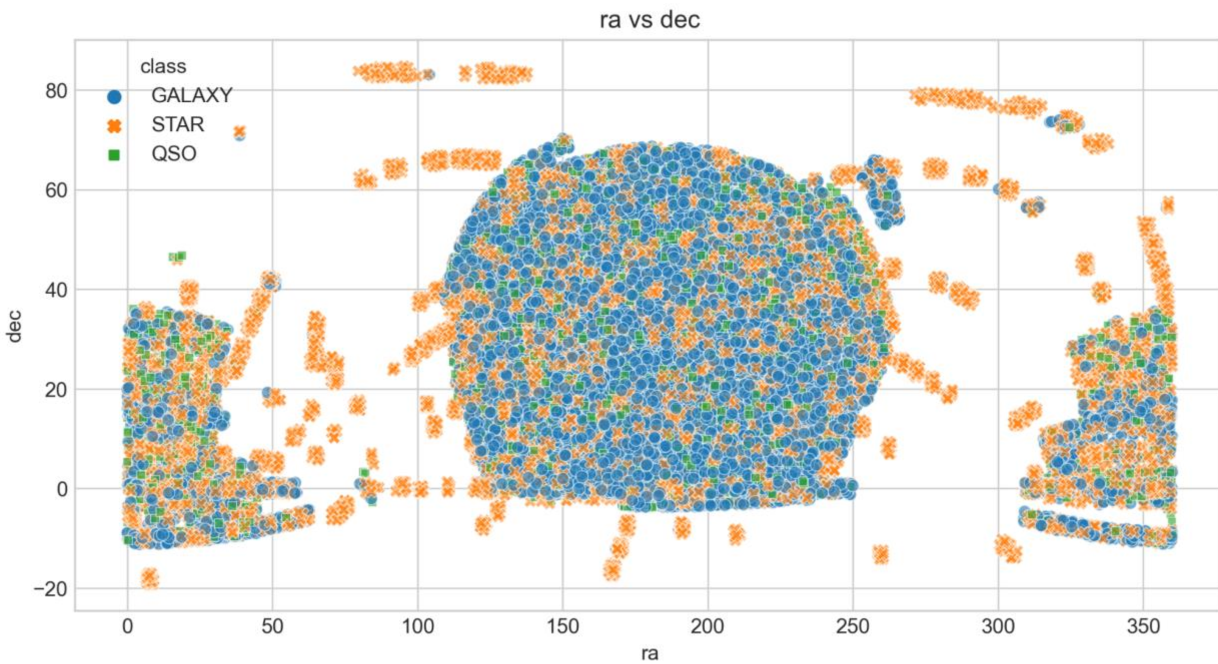

Figure 12: Scatter plot for the ra vs. dec

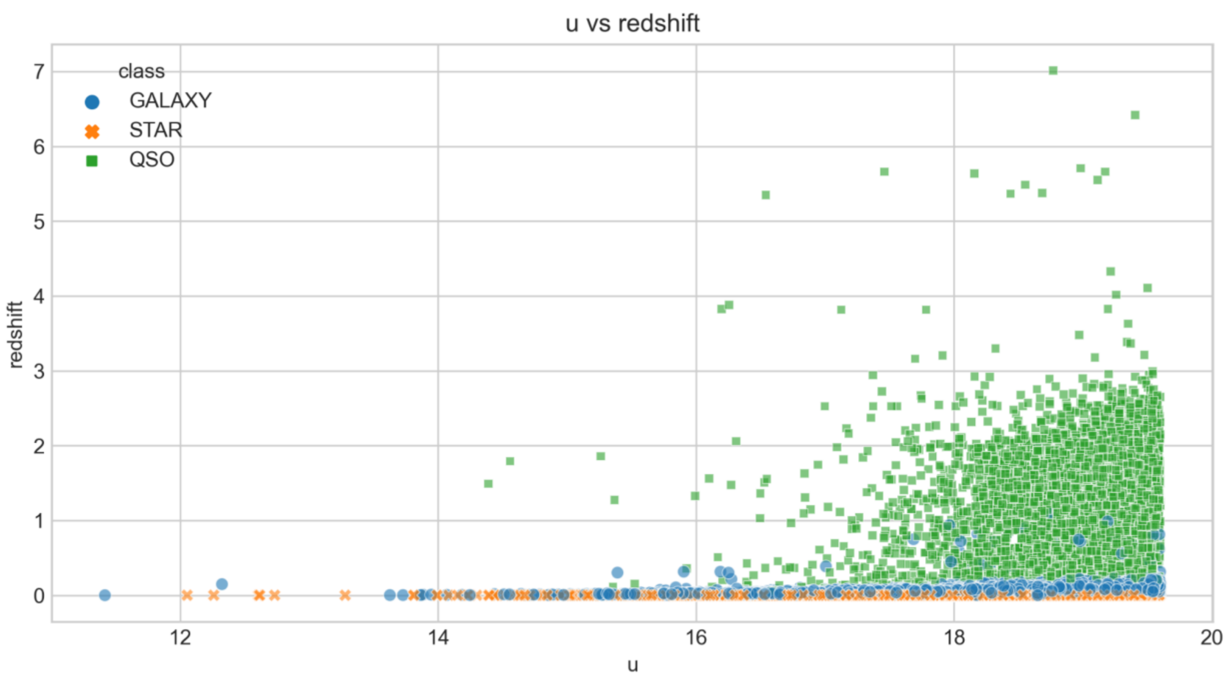

Figure 13: Scatter plot for the u vs. redshift

```python
# Box plots by class for spectral bands
plt.figure(figsize=(15, 10))
for i, feature in enumerate(tqdm(['u', 'g', 'r', 'i', 'z'], desc="Generating Box Plots")):
    plt.subplot(2, 3, i+1)
    sns.boxplot(x='class', y=feature, data=sampled_data, palette='coolwarm')
    plt.title(f'{feature} Distribution by Class')
plt.tight_layout()
plt.show()
```

Figure 14: Plotting the box plots for spectral bands

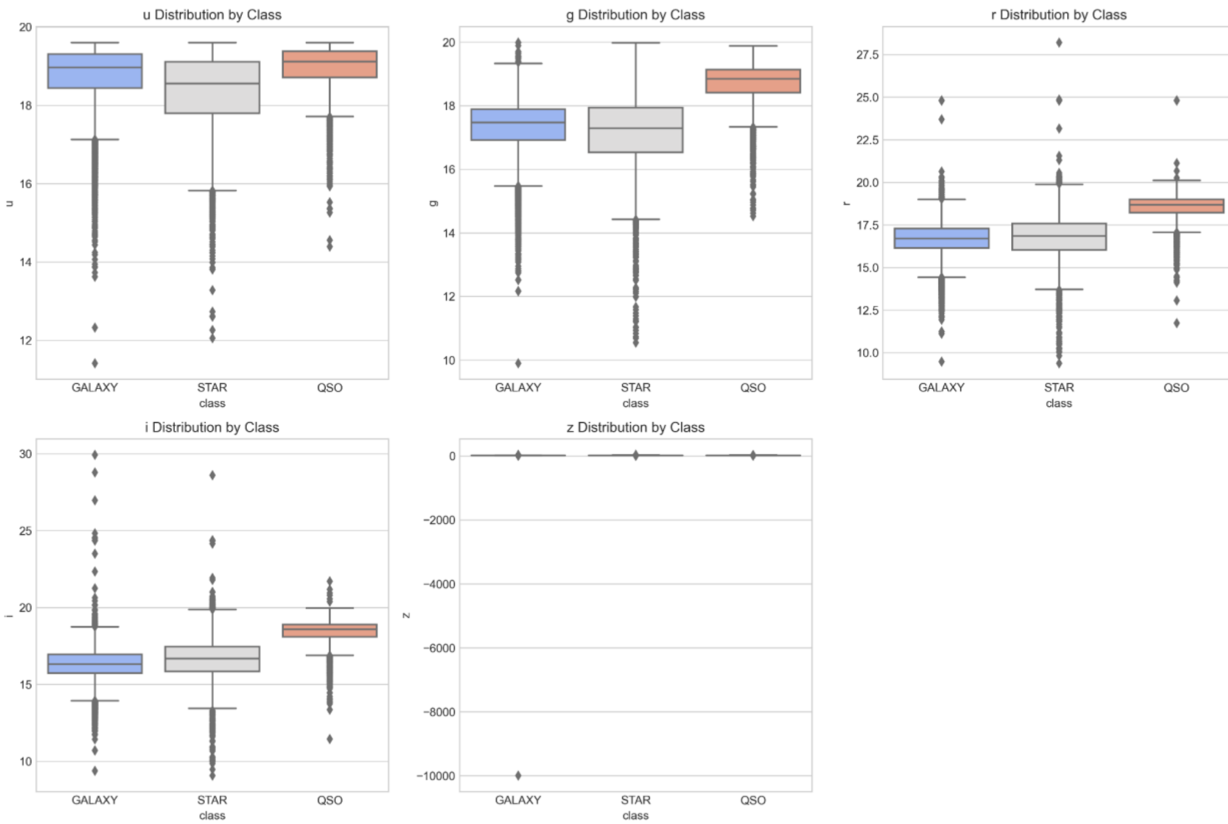

Figure 15: Box plot for various classes (u, g, r, i, z)

3.4. Running the Code

Step	Description
Set Up Environment	Install dependencies and ensure data placement
Execute Cells	Run each cell in the Jupyter Notebook sequentially
Model Training	Execute training cells to start the training process
Evaluation	Run evaluation cells to generate performance metrics

```python
# Heatmap of correlation matrix
plt.figure(figsize=(10, 8))
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.25, annot_kws={'size': 6})
plt.title('Correlation Matrix of Features')
plt.show()
```

Figure 16: Generating the heat map for the correlation of the matrix

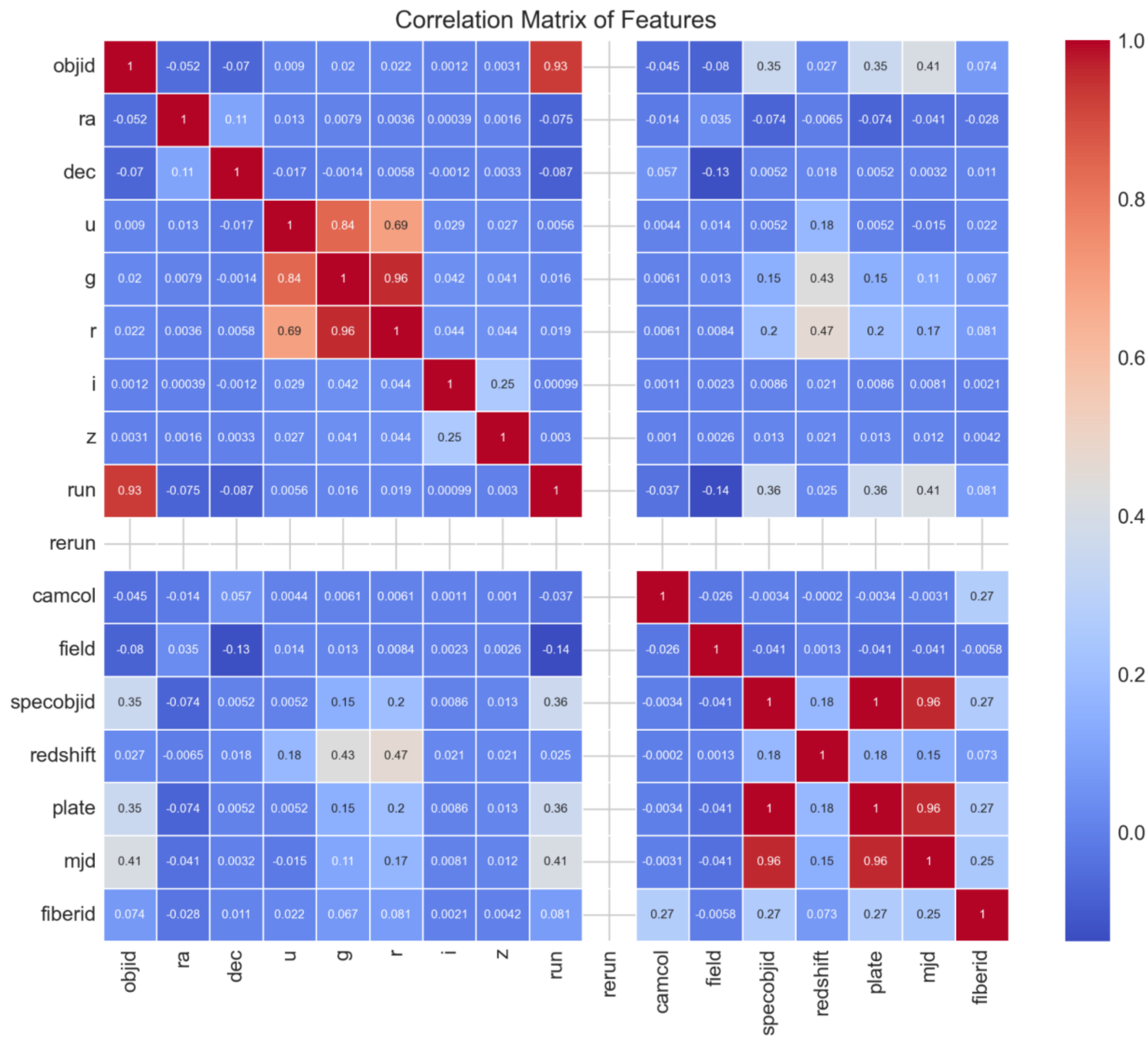

Figure 17: Heatmap for the correlation between matrix of features

```python
# Pair plot for selected features
sns.pairplot(sampled_data[['u', 'g', 'r', 'i', 'z', 'redshift', 'class']], hue='class', palette='coolwarm', diag_kind='kde')
plt.suptitle('Pairwise Relationships with Class Label', y=1.02)
plt.show()
```

Figure 18: Class specific graph generation for violin plots

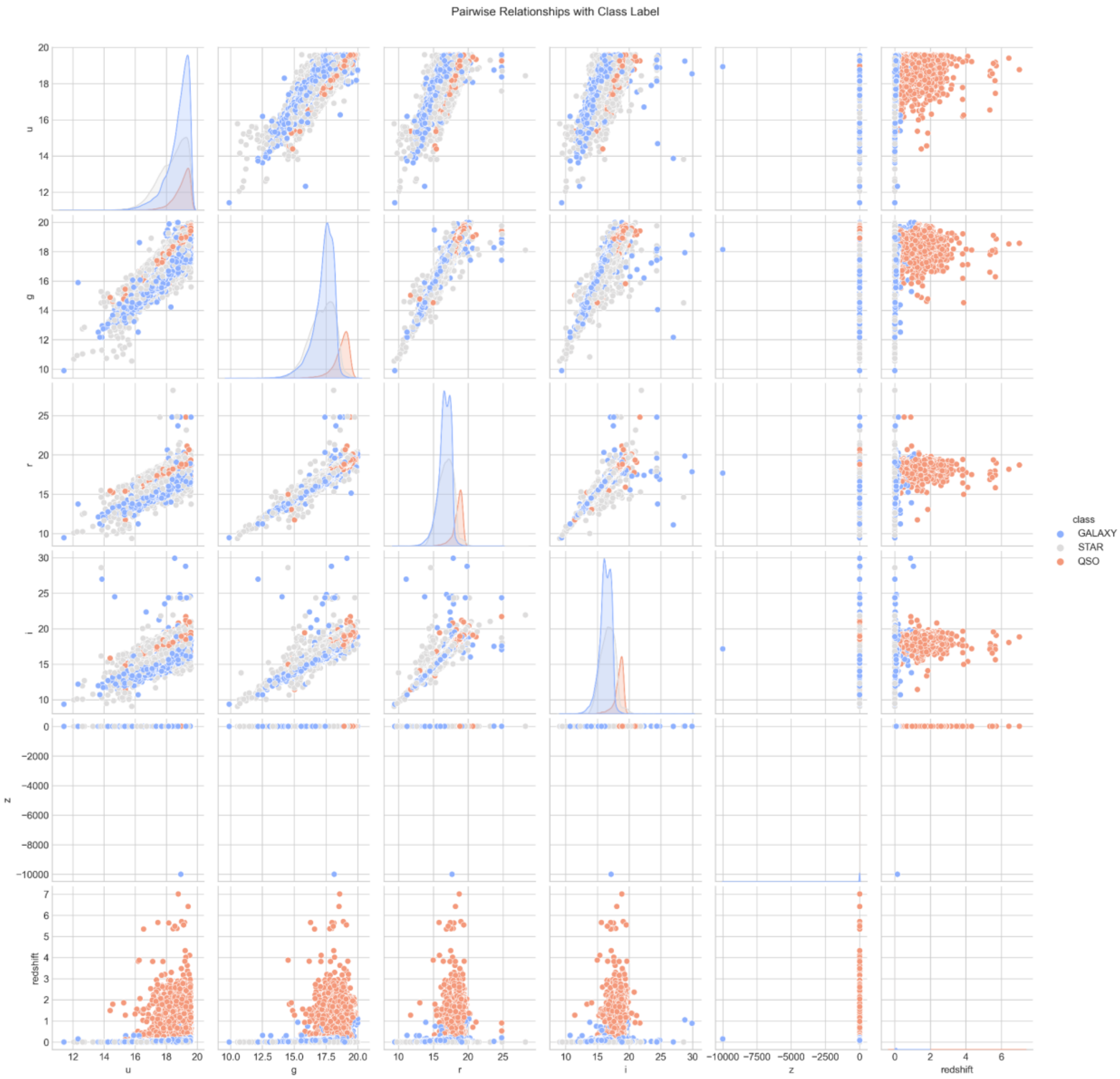

Figure 19: Violin plots for the distribution of classes

3.5. Results and Analysis

Step	Description
Review Evaluation	Analyze accuracy, precision, recall, F1 score, etc.
Visualization	Use tools like matplotlib and seaborn for plots

```python
X = data.drop('class', axis=1)  # Drop the class column to isolate features
y = data['class']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a SMOTE object
smote = SMOTE(random_state=42)

# Resample the training data
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Check the new class distribution
print("New class distribution:")
print(pd.Series(y_train_resampled).value_counts())
```

Figure 20: Data balancing and column isolation

```
New class distribution:
GALAXY    176571
STAR      176571
QSO       176571
Name: class, dtype: int64
```

Figure 21: New class distribution after removing the data imbalance

```python
# Initialize the Gradient Boosting classifier
gbm_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42, verbose=2)

# Train the model on the resampled training data
gbm_model.fit(X_train_resampled_scaled, y_train_resampled)

# Predict on the test data
y_pred_gbm = gbm_model.predict(X_test_scaled)

# Print the accuracy and the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_gbm, digits=4))

# Generate and display the confusion matrix
cm_gbm = confusion_matrix(y_test, y_pred_gbm)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_gbm, annot=True, fmt="d", cmap='Blues', xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for Gradient Boosting Machine')
plt.show()
```

Figure 22: Generating the classification report for the Gradient Boosting Machine

```
Classification Report:
              precision    recall  f1-score   support

      GALAXY     0.9897    0.9835    0.9866     75586
         QSO     0.9598    0.9562    0.9580     16299
        STAR     0.9897    0.9989    0.9943     58115

    accuracy                         0.9865    150000
   macro avg     0.9797    0.9795    0.9796    150000
weighted avg     0.9865    0.9865    0.9865    150000
```

Figure 23: Classification report for GBM

```python
# Initialize the XGBoost classifier
xgb_model = xgb.XGBClassifier(objective='multi:softmax', num_class=3, learning_rate=0.1, n_estimators=100, max_depth=4, seed=42, verbosity=2)

# Train the model on the resampled training data
xgb_model.fit(X_train_resampled_scaled, y_train_resampled)

# Predict on the test data
y_pred_xgb = xgb_model.predict(X_test_scaled)

print("Classification Report:")
print(classification_report(y_test, y_pred_xgb, digits=4))

# Generate and display the confusion matrix
cm_xgb = confusion_matrix(y_test, y_pred_xgb)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_xgb, annot=True, fmt="d", cmap='Blues', xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for XGBoost')
plt.show()
```

Figure 24: Generating the classification report for the XGBoost

```
Classification Report:
              precision    recall  f1-score   support

      GALAXY     0.9906    0.9910    0.9908     75586
         QSO     0.9642    0.9595    0.9618     16299
        STAR     0.9983    0.9990    0.9986     58115

    accuracy                         0.9907    150000
   macro avg     0.9843    0.9832    0.9838    150000
weighted avg     0.9907    0.9907    0.9907    150000
```

Figure 25: Classification report for the XGBoost

```
# Initialize the Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42, verbose=2)

# Train the model
rf_model.fit(X_train_resampled_scaled, y_train_resampled)

# Predict on the test data
y_pred_rf = rf_model.predict(X_test_scaled)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred_rf, digits=4))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

Figure 26: Generating the classification report for the Random Forest

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    3.4s finished
Classification Report:
              precision    recall  f1-score   support

      GALAXY     0.9917    0.9924    0.9921     75586
         QSO     0.9749    0.9652    0.9700     16299
        STAR     0.9970    0.9988    0.9979     58115

    accuracy                         0.9919    150000
   macro avg     0.9879    0.9855    0.9867    150000
weighted avg     0.9919    0.9919    0.9919    150000
```

Figure 27: Classification report for the Random Forest

```python
# Predict labels on the test set using each model
y_pred_gbm = gbm_model.predict(X_test_scaled)
y_pred_xgb = xgb_model.predict(X_test_scaled)
y_pred_rf = rf_model.predict(X_test_scaled)

# Define a function to calculate and return performance metrics
def compute_metrics(y_true, y_pred):
    """
    Computes and returns accuracy, precision, recall, and F1-score for the given true and predicted labels.
    """
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    return accuracy, precision, recall, f1

# Compute metrics for each model using the compute_metrics function
metrics_gbm = compute_metrics(y_test, y_pred_gbm)
metrics_xgb = compute_metrics(y_test, y_pred_xgb)
metrics_rf = compute_metrics(y_test, y_pred_rf)

# Compile the metrics into a DataFrame for better comparison
metrics_df = pd.DataFrame({
    'Model': ['GBM', 'XGBoost', 'Random Forest'],
    'Accuracy': [metrics_gbm[0], metrics_xgb[0], metrics_rf[0]],
    'Precision': [metrics_gbm[1], metrics_xgb[1], metrics_rf[1]],
    'Recall': [metrics_gbm[2], metrics_xgb[2], metrics_rf[2]],
    'F1-Score': [metrics_gbm[3], metrics_xgb[3], metrics_rf[3]]
})

# Print the DataFrame to display the metrics for each model
print("Performance Metrics for Each Model:")
display(metrics_df)
```

Figure 28: Model comparison and its depiction

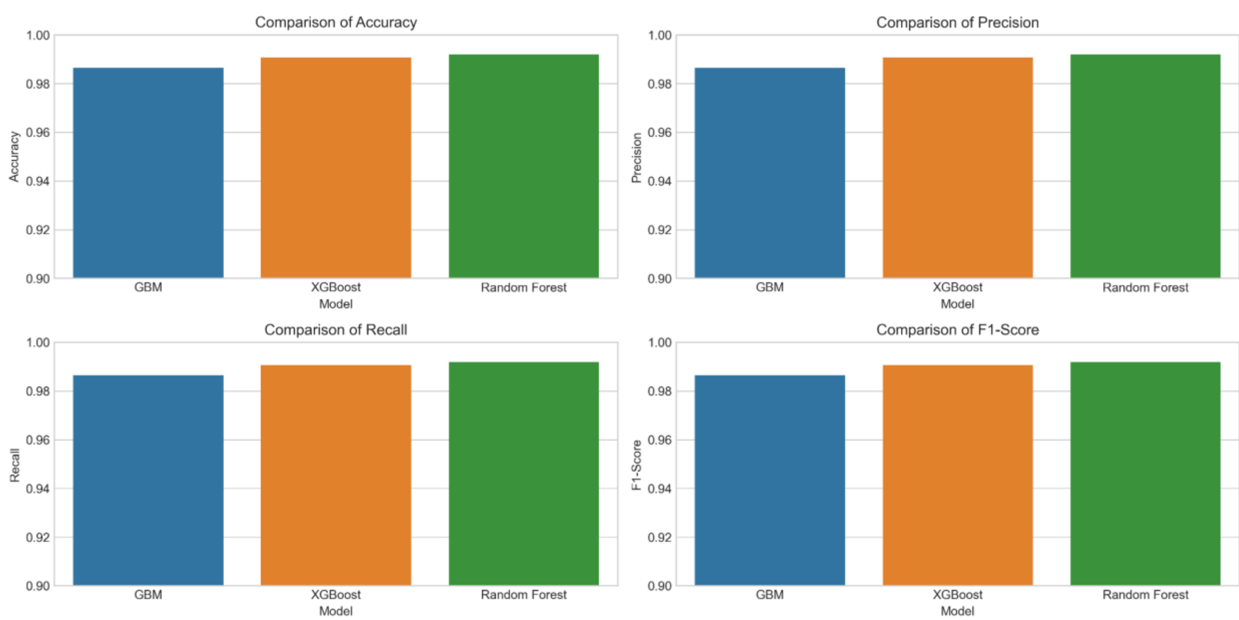

Figure 29: Comparison between the three models for eval matrices

```python
# Create a DataFrame with actual labels and predictions from each model
comparison_df = pd.DataFrame({
    'Actual': y_test,
    'GBM_Predictions': y_pred_gbm,
    'XGB_Predictions': y_pred_xgb,
    'RF_Predictions': y_pred_rf
})

# Function to return prediction if it's incorrect, otherwise return None
def mark_incorrect(actual, prediction):
    return prediction if actual != prediction else None

# Apply the function to mark incorrect predictions
comparison_df['GBM_Incorrect'] = comparison_df.apply(lambda x: mark_incorrect(x['Actual'], x['GBM_Predictions']), axis=1)
comparison_df['XGB_Incorrect'] = comparison_df.apply(lambda x: mark_incorrect(x['Actual'], x['XGB_Predictions']), axis=1)
comparison_df['RF_Incorrect'] = comparison_df.apply(lambda x: mark_incorrect(x['Actual'], x['RF_Predictions']), axis=1)

# Filter to show only rows where all models made incorrect predictions
all_incorrect_df = comparison_df[
    (comparison_df['GBM_Incorrect'].notna()) &
    (comparison_df['XGB_Incorrect'].notna()) &
    (comparison_df['RF_Incorrect'].notna())
]

# Display rows where all models were incorrect
print("Rows where all models made incorrect predictions:")
display(all_incorrect_df[['Actual', 'GBM_Incorrect', 'XGB_Incorrect', 'RF_Incorrect']])

# Generate a summary of incorrect predictions for each model by actual class
incorrect_summary = comparison_df[['Actual', 'GBM_Incorrect', 'XGB_Incorrect', 'RF_Incorrect']].melt(id_vars=['Actual'], value_name='Prediction').dropna()
incorrect_summary = incorrect_summary.groupby(['Actual', 'variable']).size().unstack(fill_value=0)
print("Summary of incorrect predictions for each model by actual class:")
print(incorrect_summary)
```

Figure 30: Calculating the inaccurate predictions by the models

```
Rows where all models made incorrect predictions:
```

	Actual	GBM_Incorrect	XGB_Incorrect	RF_Incorrect
140199	QSO	GALAXY	GALAXY	GALAXY
213764	QSO	GALAXY	GALAXY	GALAXY
158631	QSO	GALAXY	GALAXY	GALAXY
69080	QSO	GALAXY	GALAXY	GALAXY
467931	GALAXY	QSO	QSO	QSO
...	...	...	...	...
471517	QSO	GALAXY	GALAXY	GALAXY
495963	GALAXY	QSO	QSO	QSO
89576	GALAXY	QSO	QSO	QSO
76268	GALAXY	QSO	QSO	QSO
52542	QSO	GALAXY	GALAXY	GALAXY

971 rows × 4 columns

```
Summary of incorrect predictions for each model by actual class:
variable  GBM_Incorrect  RF_Incorrect  XGB_Incorrect
Actual
GALAXY             1248           573            677
QSO                 714           567            660
STAR                 65            68             56
```

Figure 31: Depiction of all the model inaccurate predictions

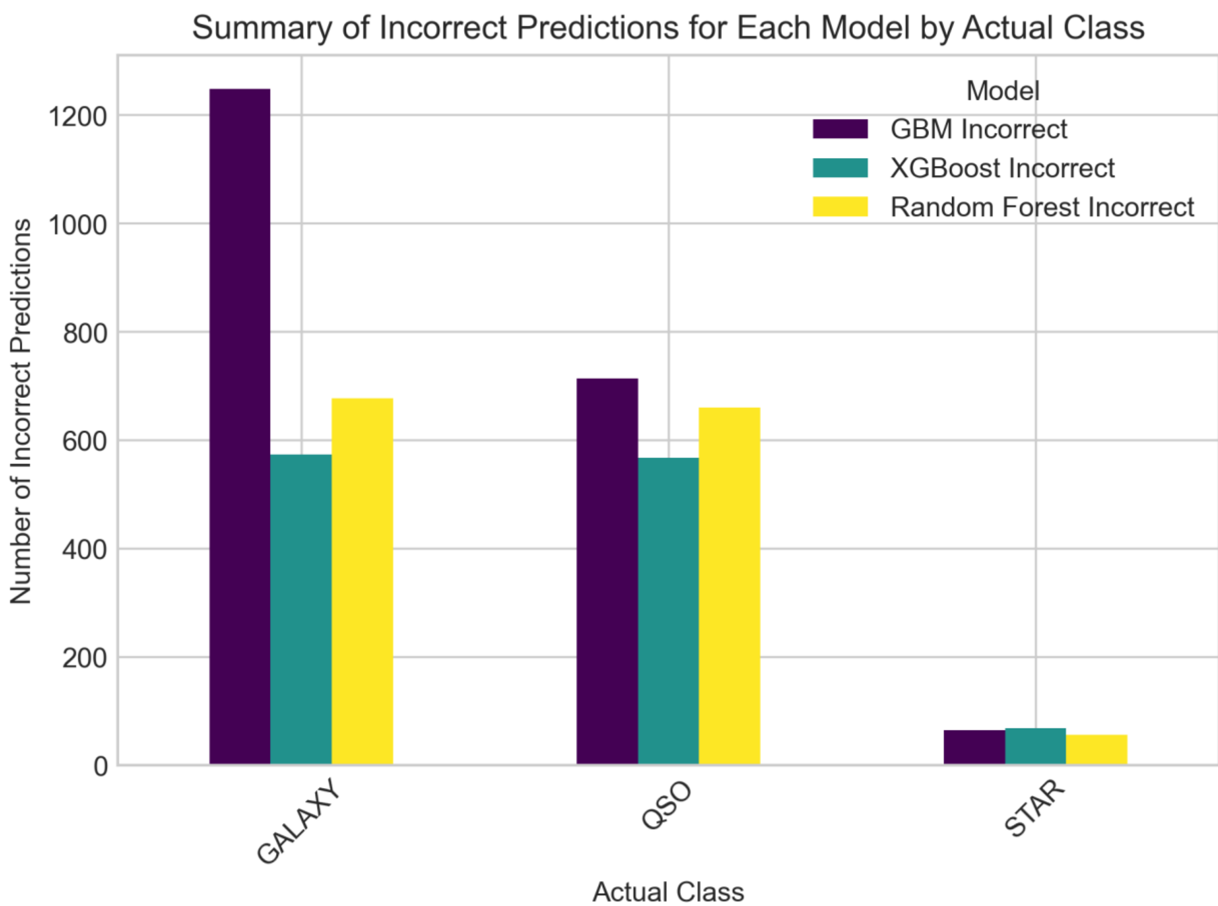

Figure 32: Bar chart for the inaccurate predictions of the models

References

1. Domínguez Sánchez, H., et al. "Deep learning for galaxy surface brightness profile fitting." *Monthly Notices of the Royal Astronomical Society* 484.1 (2019): 93-110.
2. Tuccillo, D., et al. "Deep learning for studies of galaxy morphology." *Monthly Notices of the Royal Astronomical Society* 475.4 (2018): 894-913.
3. Cheng, T., et al. "Machine learning-based morphological classification of galaxies." *Monthly Notices of the Royal Astronomical Society* 511.4 (2022): 5032-5045.
4. Willett, K. W., et al. "Galaxy Zoo: morphological classification and citizen science." *Monthly Notices of the Royal Astronomical Society* 435.4 (2013): 2835-2860.
5. Martin, G., et al. "Galaxy morphology classification in deep-wide surveys via unsupervised machine learning." *Monthly Notices of the Royal Astronomical Society* 491.1 (2020): 1408-1416.
6. Huertas-Company, M., et al. "A deep learning framework for galaxy morphology analysis: application to the CANDELS survey." *The Astrophysical Journal Supplement Series* 221.1 (2015): 8.
7. Dieleman, S., et al. "Rotation-invariant convolutional neural networks for galaxy morphology prediction." *Monthly Notices of the Royal Astronomical Society* 450.2 (2015): 1441-1459.

8. Domínguez Sánchez, H., et al. "Improving galaxy morphologies for SDSS with deep learning." *Monthly Notices of the Royal Astronomical Society* 476.3 (2018): 3661-3676.
9. Bottrell, C., et al. "Morphological classifications of galaxies in the Sloan Digital Sky Survey using convolutional neural networks." *Monthly Notices of the Royal Astronomical Society* 474.3 (2018): 2937-2950.
10. Walmsley, M., et al. "Galaxy Zoo: morphological classifications of galaxy mergers." *Monthly Notices of the Royal Astronomical Society* 491.2 (2020): 1554-1573.
11. Hezaveh, Y. D., et al. "Fast automated analysis of strong gravitational lenses with convolutional neural networks." *Nature* 548.7669 (2017): 555-557.
12. Khan, S. M., et al. "Galaxy morphology classification using deep learning." *Astrophysics and Space Science* 361.11 (2016): 1-11.
13. Schawinski, K., et al. "Galaxy Zoo: the effect of bar-driven secular evolution on the distribution of galaxies in the green valley." *Monthly Notices of the Royal Astronomical Society* 450.1 (2015): 297-308.
14. Huertas-Company, M., et al. "Morphologies of galaxies in the Hubble Ultra Deep Field using a support vector machine." *Astronomy & Astrophysics* 525 (2011): A157.
15. Gharat, S., et al. "Galaxy classification: a deep learning approach for classifying Sloan Digital Sky Survey images." *Monthly Notices of the Royal Astronomical Society* 511.4 (2022): 5120-5124.
16. Barchi, P. H., et al. "Galaxy morphology classification in S-PLUS using deep learning." *Monthly Notices of the Royal Astronomical Society* 481.3 (2018): 3095-3107.
17. Variawa, M. Z., et al. "Transfer learning and deep metric learning for automated galaxy morphology representation." *IEEE Access* 10 (2022): 19539-19549.
18. Graff, P., et al. "Deep learning for galaxy morphology." *Monthly Notices of the Royal Astronomical Society* 493.3 (2020): 3142-3156.
19. Davies, L. J., et al. "Deep learning galaxy morphology classification for large surveys: application to the SAMI Galaxy Survey." *Monthly Notices of the Royal Astronomical Society* 483.2 (2019): 5446-5457.
20. Guo, Y., et al. "Deep learning for galaxy morphology classification using large-scale data from the DECaLS survey." *Astrophysics and Space Science* 364.10 (2019): 1-9.