ME 228

REPORT

Pratyush Jagtap 22B2202
Shikhar Kunal 22B2201

Guided by: Prof. Shyamprasad Karagadde

Problem Statement

In astronomy, understanding the nature of celestial objects such as stars, galaxies, and quasars is crucial for unraveling the mysteries of the universe. Spectral characteristics serve as vital clues in classifying these objects, aiding astronomers in their quest to comprehend the cosmos.

This project aims to develop a machine learning model for classifying celestial objects based on their spectral features.

Other Applications

Spectral classification techniques, originally developed for astronomy, find versatile applications across diverse fields.

1. Medicine: Aid in disease diagnosis from medical imaging.
2. Environmental Monitoring: Enable disaster management and conservation through remote sensing.
3. Geology: Identify mineral deposits and map geological structures.
4. Oceanography: Monitor marine ecosystems and assess water quality.
5. Manufacturing: Detect defects and optimize production processes.

Dataset

The data consists of 100,000 observations of space taken by the SDSS (Sloan Digital Sky Survey).
Every observation is described by **17 feature columns and 1 class column** which identifies it to be either a **star, galaxy or quasar**.

ref https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17/data

Features

1. alpha = Right Ascension angle (at J2000 epoch)
2. delta = Declination angle (at J2000 epoch)
3. u = Ultraviolet filter in the photometric system
4. g = Green filter in the photometric system
5. r = Red filter in the photometric system
6. i = Near Infrared filter in the photometric system
7. z = Infrared filter in the photometric system
8. run_ID = Run Number used to identify the specific scan
9. rereun_ID = Rerun Number to specify how the image was processed
10. cam_col = Camera column to identify the scanline within the run
11. field_ID = Field number to identify each field
12. spec_obj_ID = Unique ID used for optical spectroscopic objects (this means that 2 different observations with the same spec_obj_ID must share the output class)
13. class = object class (galaxy, star or quasar object)
14. redshift = redshift value based on the increase in wavelength
15. plate = plate ID, identifies each plate in SDSS
16. MJD = Modified Julian Date, used to indicate when a given piece of SDSS data was taken
17. fiber_ID = fiber ID that identifies the fiber that pointed the light at the focal plane in each observation

ref https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17/data

Solution

1. Feature Engineering and Selection

- **Correlation Analysis**: Analyzing how strongly different features correlate with each other and the target classes (star, galaxy, quasar). **Removing highly correlated features** minimises redundancy and potential performance issues.
- **Domain-Specific Feature Engineering**: Leveraging the understanding of astrophysical properties embedded in the SDSS17 data:
  - **Magnitudes**: Brightness of a celestial object in different wavelength bands, a key characteristic in classification.
  - **Redshift**: Indicates the distance and expansion velocity of the object, which helps classify object types.
  - **Spectral Properties**: Analyzing absorption and emission lines in the spectrum reveals the object's chemical composition, which is crucial for determining stellar classes.

2. Handling Class Imbalance

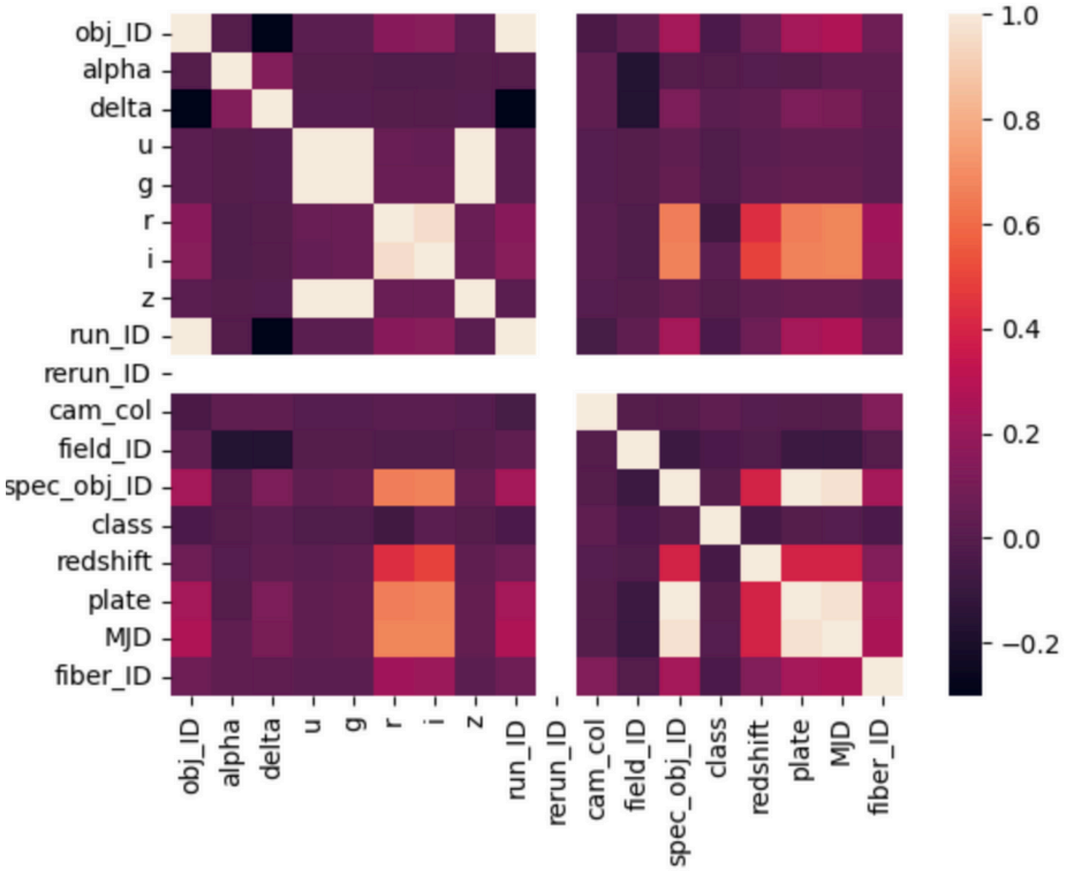
- **Undersampling**: Strategically removing samples from overrepresented classes to achieve a more balanced class distribution for training.

3. Machine Learning Algorithms

- Support Vector Classification (SVC)
- Multilayer Perceptron (MLP)
- Random Forest Classifier
- AdaBoost Classifier
- Gaussian Naive Bayes (GaussianNB)
- Logistic Regression

Data Preprocessing

```
import seaborn as sns
corr_matrix = df.corr() #calculating the correlation matrix
data_plot = sns.heatmap(corr_matrix) #heatmap
plt.show()
```



Correlation Analysis:

The heat map generated visually gives us an idea of the extent of correlation between features.

Correlation Analysis

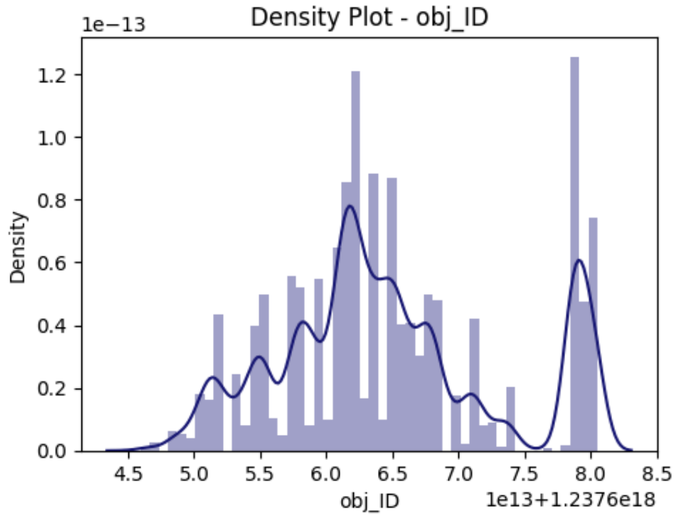
This code segment calculates the absolute correlation coefficients between pairs of features in a correlation matrix. It categorizes correlations as very high, high, moderate, or low based on predefined thresholds.

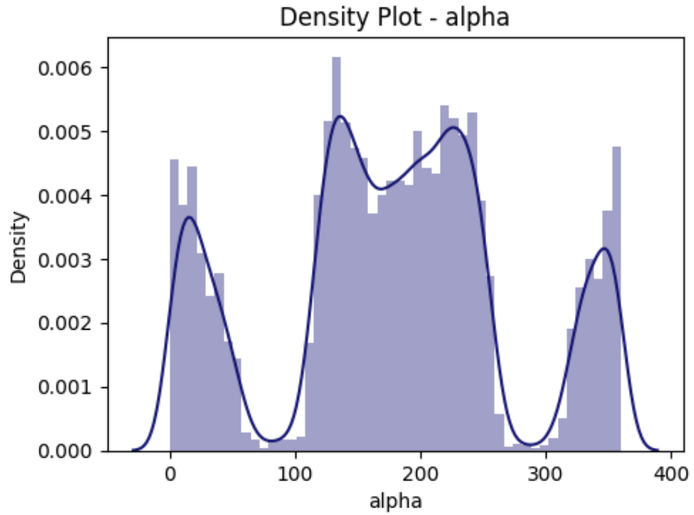
Features with very high correlation coefficients (exceeding 0.9) are marked for removal to address multicollinearity in the dataset.

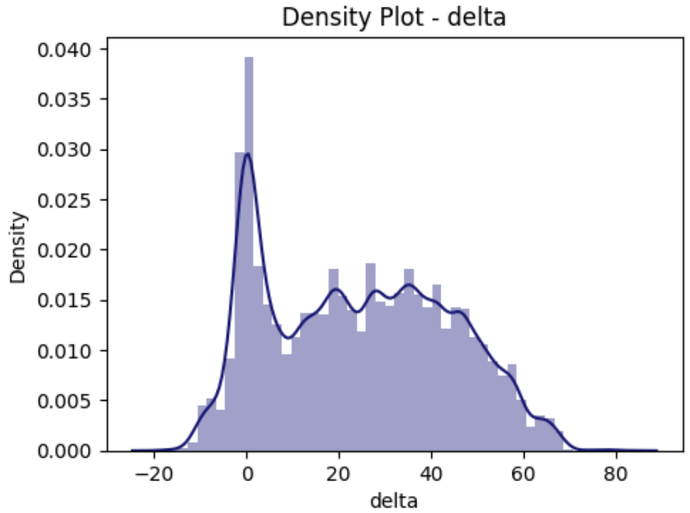
```python
corr_numpy = corr_matrix.to_numpy() #converting corr matrix to numpy for easy work ahead
high = 0
moderate = 0
low = 0
very_high = 0
unusable = []
for i in range(len(corr_numpy)):
  for j in range(len(corr_numpy[0])):
    if abs(corr_numpy[i][j])>0.9 and i!=j:
      very_high = very_high +1
      unusable.append([i,j])
    elif abs(corr_numpy[i][j])>0.5: high = high +1
    elif abs(corr_numpy[i][j])>0.2: moderate = moderate +1
    else: low = low +1
print(very_high, high, moderate, low)
print(unusable)

# drop 4,6,7,8,15,16,
```

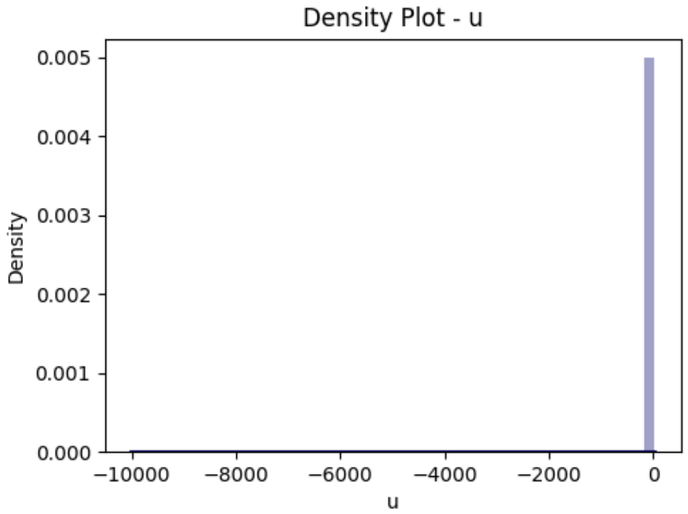
```
16 29 36 243
[[0, 8], [3, 4], [3, 7], [4, 3], [4, 7], [5, 6], [6, 5], [7, 3], [7, 4], [8, 0], [12, 15], [12
```

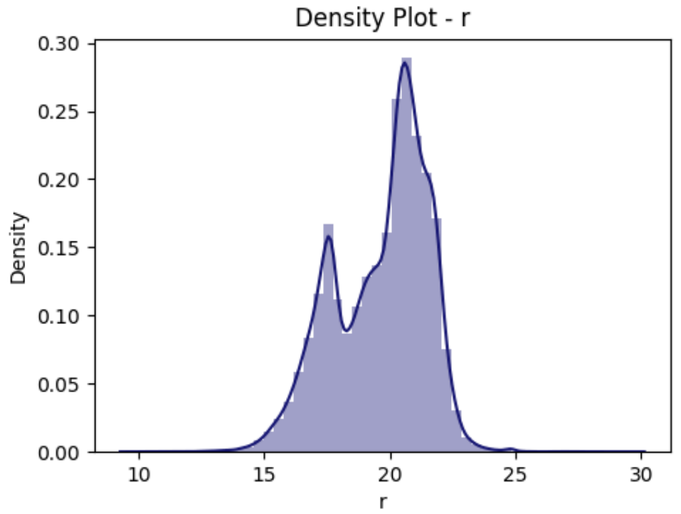
Data Visualisation

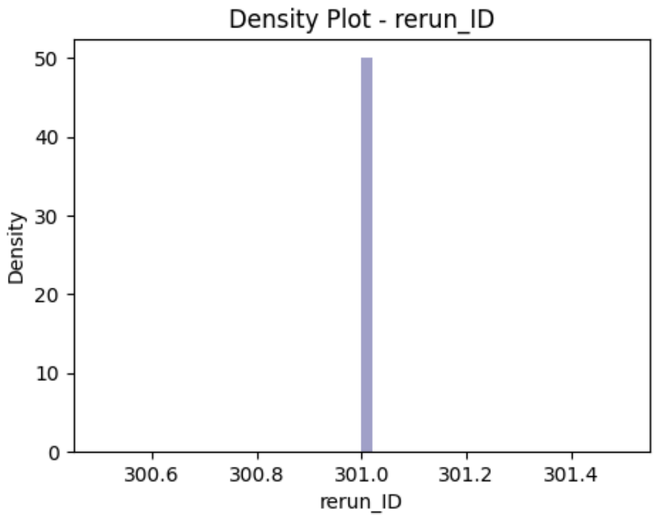




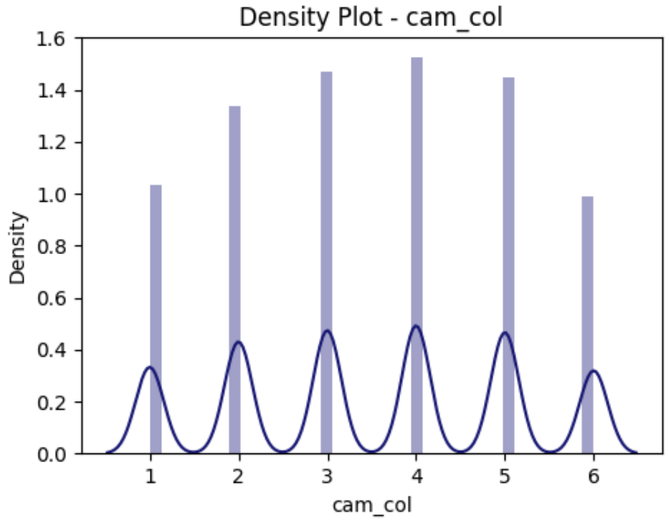


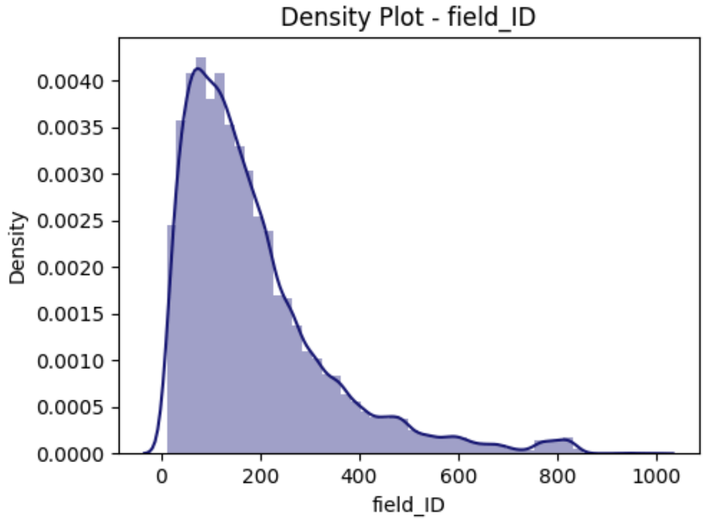


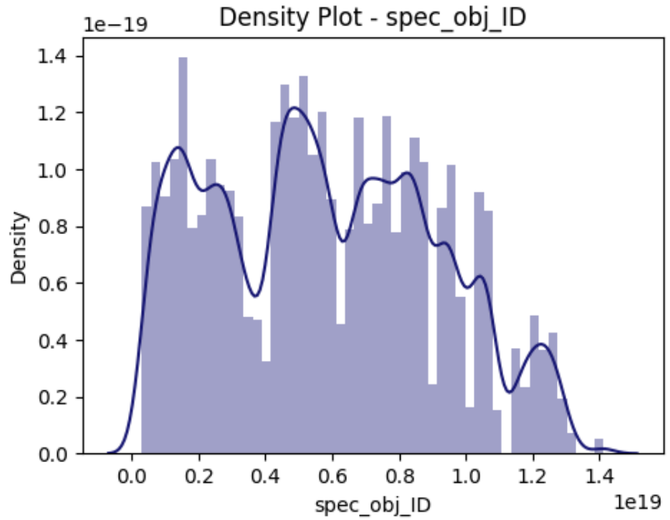


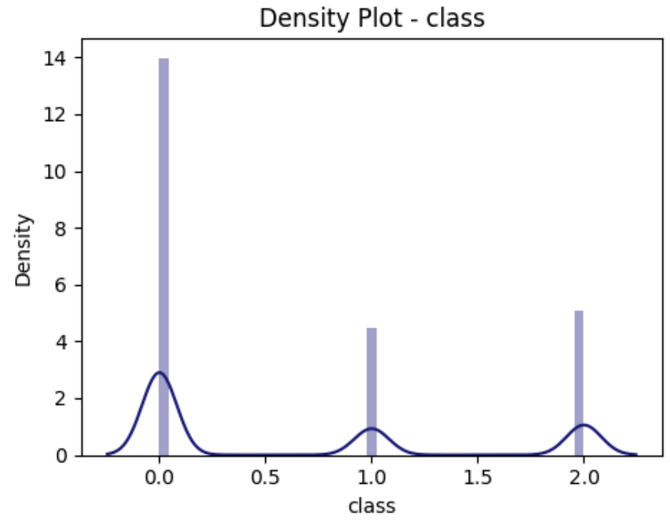


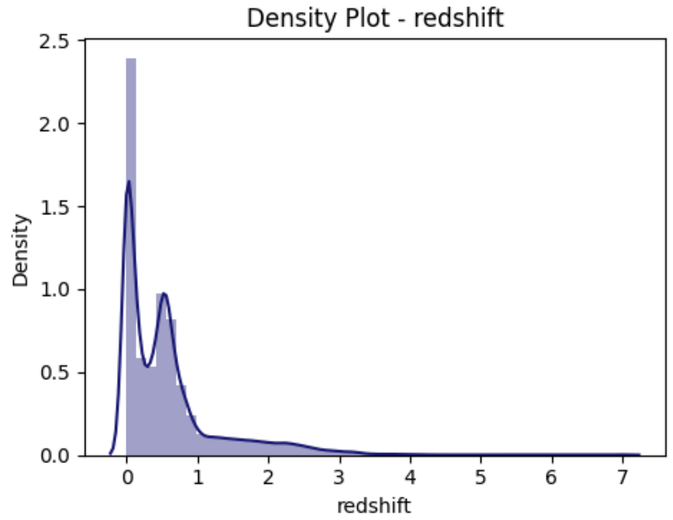
Data Visualisation

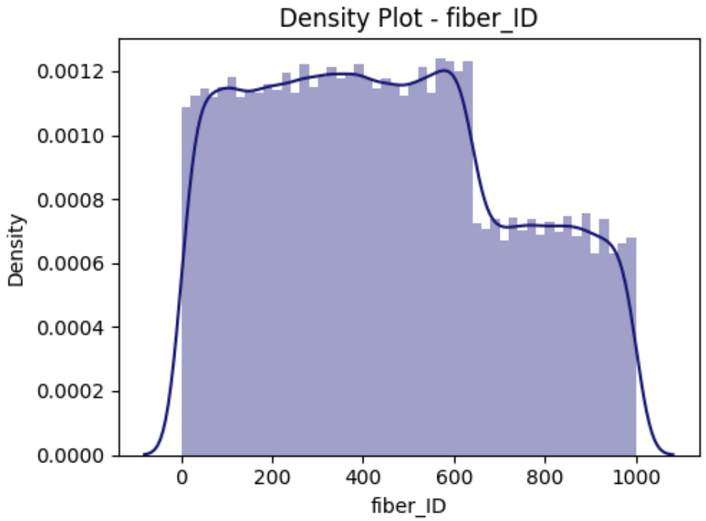




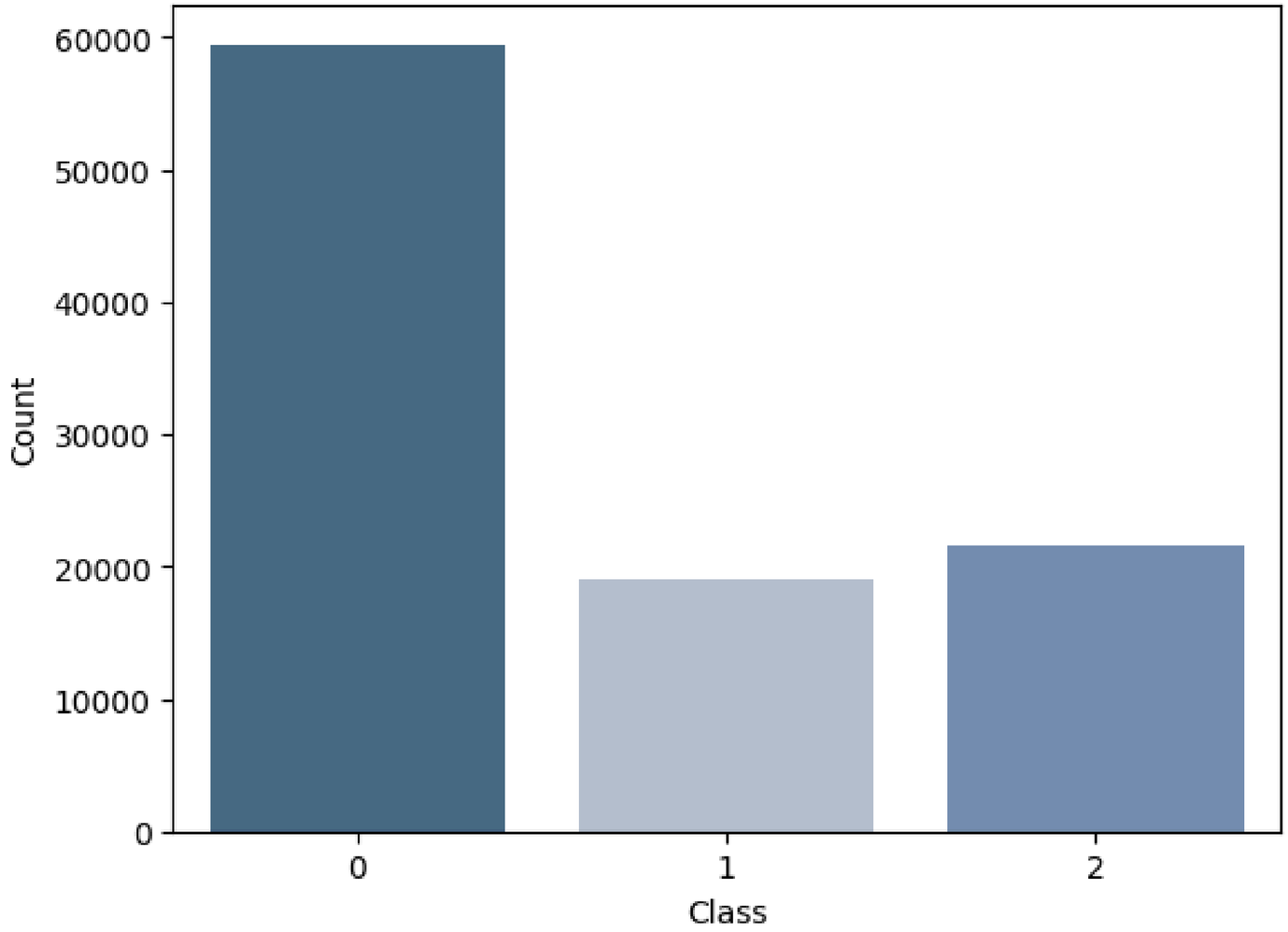








Data Distribution

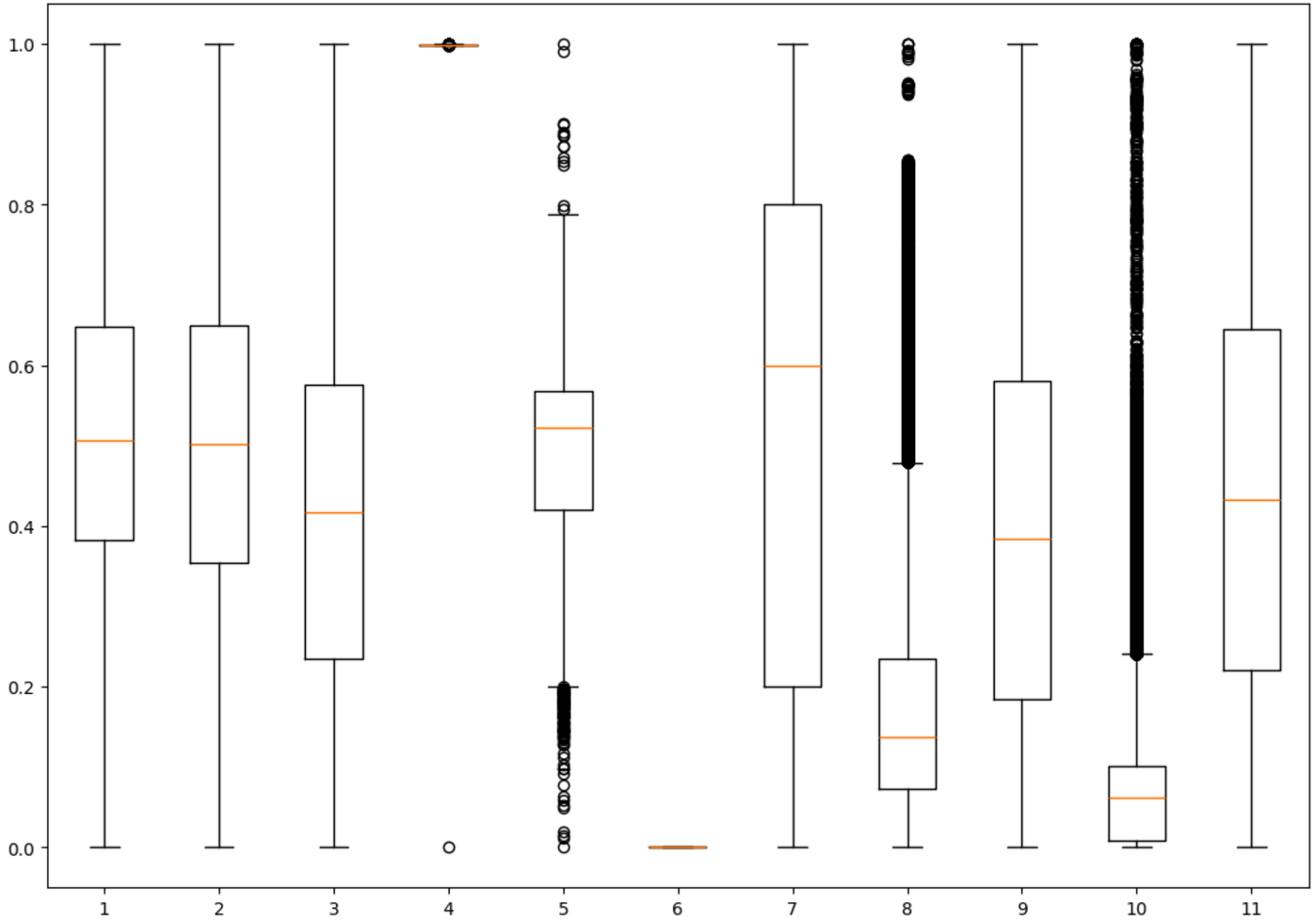


Class 1 - Galaxy

Class 2 - Quasar

Class 3 - Star

Data Scaling



Box Plot shows feature value ranges

```
#scaling data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

X = df[['obj_ID', 'alpha', 'delta', 'u', 'r', 'rerun_ID', 'cam_col', 'field_ID', 'spec_obj_ID', 'redshift', 'fiber_ID']]

Undersampling

Data distribution by class shows us our dataset is imbalanced as well as there are too many data points.
Large dataset will increase the computation cost by a lot.
To avoid all that, we will undersample the datasets to 5000 datapoints in each class.

```
from collections import Counter

class_counts = Counter(df['class'])

min_class_occurrence = min(class_counts.values())
min_class = [cls for cls, count in class_counts.items() if count == min_class_occurrence][0]

df = pd.concat([df[df['class'] == cls].sample(5000) for cls in class_counts.keys()])

df.reset_index(drop=True, inplace=True)

print(df.shape)
```

Data Splitting

Train Test split used is 2:1

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.33,shuffle=True, random_state=42)
```

Model Selection

The following models were tested

- Support Vector Classification (SVC)
- Multilayer Perceptron (MLP)
- Random Forest Classifier
- AdaBoost Classifier
- Gaussian Naive Bayes (GaussianNB)
- Logistic Regression

Results then can be compared to help in model selection

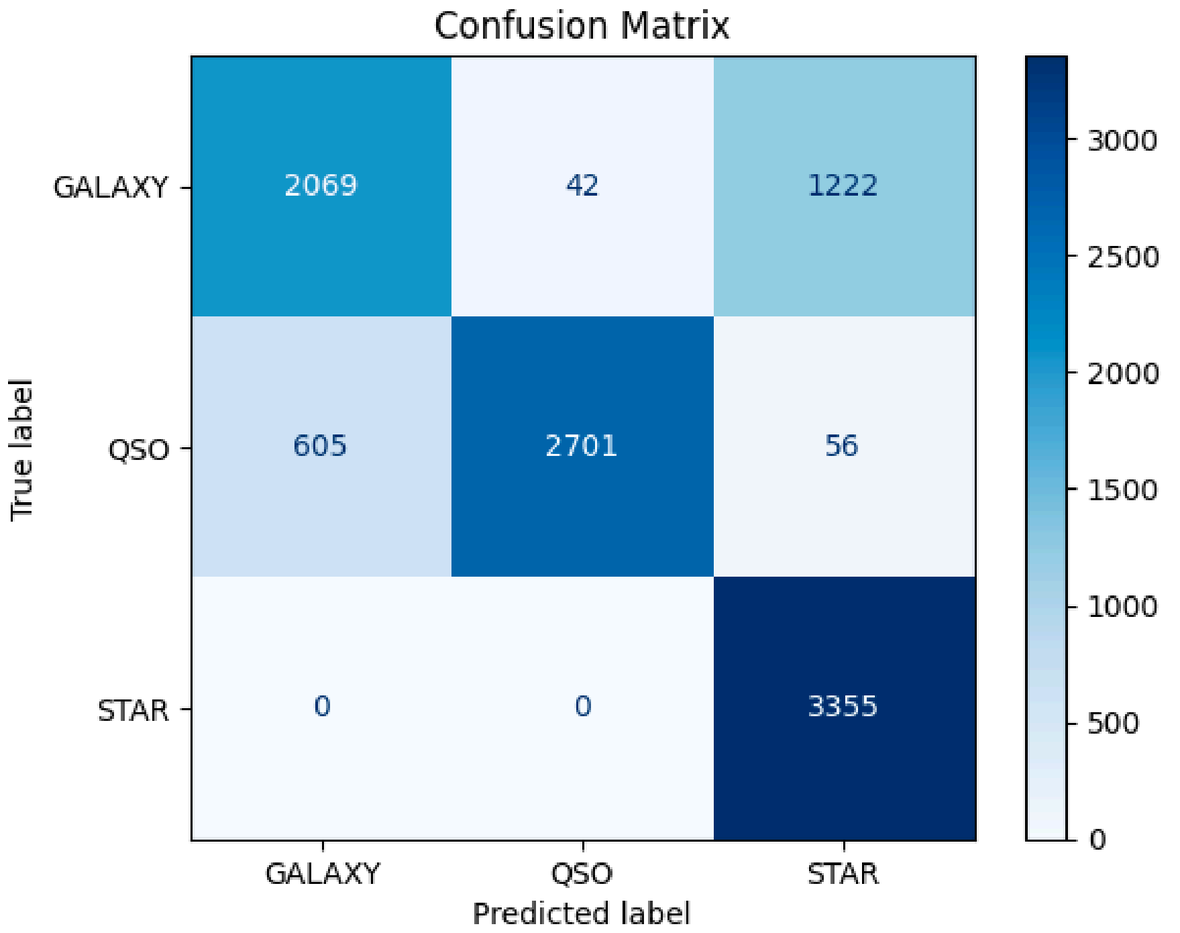
Support Vector Classifier

Hyperparameter: C

Values taken: 0.1, 1, 10

```
param_grid_1 = {'C': [0.1, 1, 10],
                'gamma': [0.01]}
grid_1 = GridSearchCV(SVC(), param_grid_1, refit = True, verbose = 3, cv=2)
grid_1.fit(x_train, y_train)

print(grid_1.best_params_)

print(grid_1.best_estimator_)


grid_predictions_1 = grid_1.predict(x_train)


print(classification_report(y_train, grid_predictions_1))
accuracy_1 = accuracy_score(y_train, grid_predictions_1)
print("Accuracy:", accuracy_1)
```

```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
[CV 1/2] END .................C=0.1, gamma=0.01;, score=0.527 total time=   1.5s
[CV 2/2] END .................C=0.1, gamma=0.01;, score=0.519 total time=   1.5s
[CV 1/2] END ...................C=1, gamma=0.01;, score=0.602 total time=   1.3s
[CV 2/2] END ...................C=1, gamma=0.01;, score=0.613 total time=   1.3s
[CV 1/2] END ..................C=10, gamma=0.01;, score=0.795 total time=   1.1s
[CV 2/2] END ..................C=10, gamma=0.01;, score=0.788 total time=   1.2s
{'C': 10, 'gamma': 0.01}
SVC(C=10, gamma=0.01)
              precision    recall  f1-score   support

           0       0.77      0.62      0.69      3333
           1       0.98      0.80      0.88      3362
           2       0.72      1.00      0.84      3355

    accuracy                           0.81     10050
   macro avg       0.83      0.81      0.80     10050
weighted avg       0.83      0.81      0.80     10050

Accuracy: 0.8084577114427861
```



Learned Hyperparameter

C = 10

Accuracy = 0.81

Multilayer Perceptron Classifier

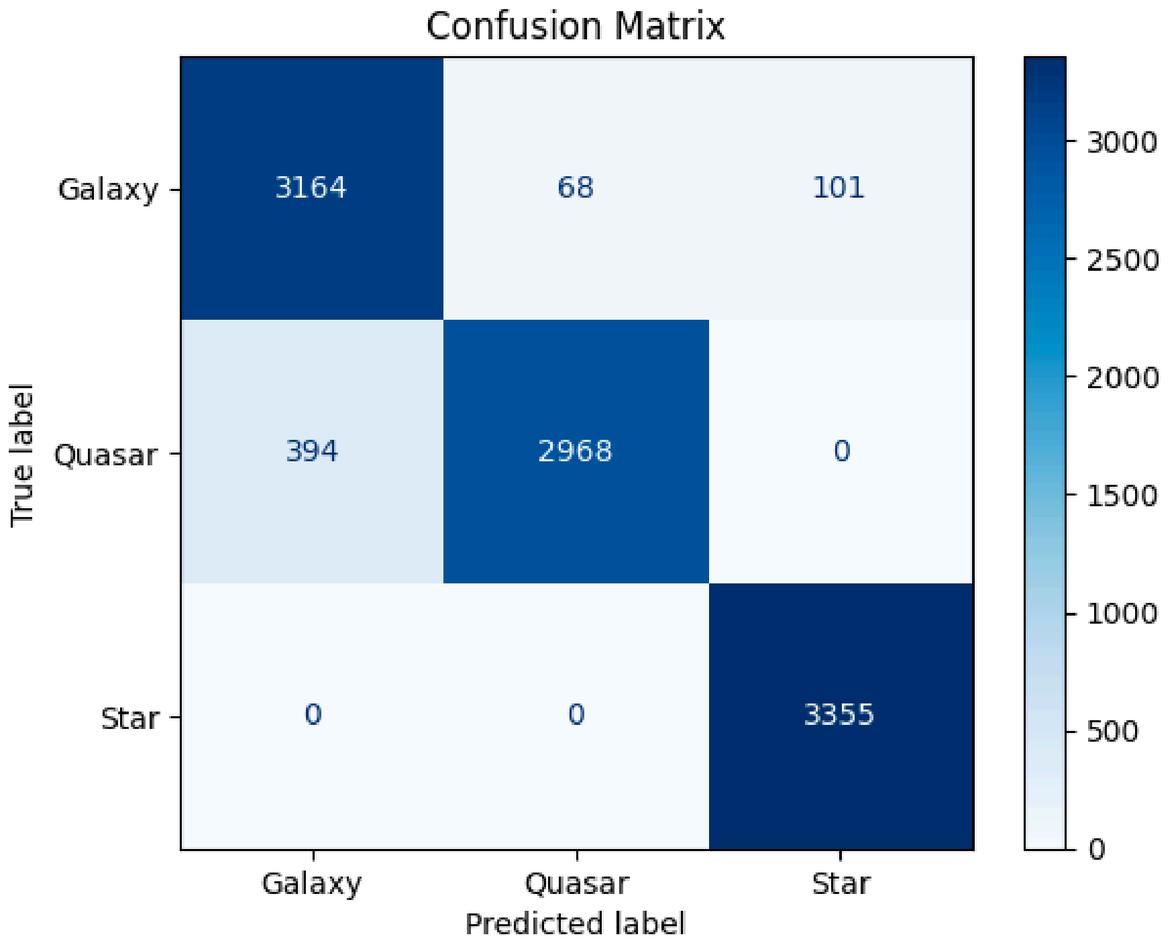
Hyperparameters: Hidden Layer Sizes and Learning Rate Initial

Hidden Layer Size: 32, 64

Learning Rate Initial: 0.1, 0.01, 0.001

```
#MLP
classifier = MLPClassifier(max_iter = 500)

param_grid_2 = {'hidden_layer_sizes': [32, 64],
                'learning_rate_init': [0.1, 0.01, 0.001]}
grid_2 = GridSearchCV(classifier, param_grid_2, refit = True, verbose = 3, error_score = 'raise', cv= 2)
grid_2.fit(x_train, y_train)

print(grid_2.best_params_)

print(grid_2.best_estimator_)

grid_predictions_2 = grid_2.predict(x_train)

print(classification_report(y_train, grid_predictions_2))
accuracy_2 = accuracy_score(y_train, grid_predictions_2)
print("Accuracy:", accuracy_2)
```

```
Fitting 2 folds for each of 6 candidates, totalling 12 fits
[CV 1/2] END hidden_layer_sizes=32, learning_rate_init=0.1;, score=0.932 total time=   0.2s
[CV 2/2] END hidden_layer_sizes=64, learning_rate_init=0.001;, score=0.940 total time=   2.2s
{'hidden_layer_sizes': 64, 'learning_rate_init': 0.001}
MLPClassifier(hidden_layer_sizes=64, max_iter=500)
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      3333
           1       0.98      0.88      0.93      3362
           2       0.97      1.00      0.99      3355

    accuracy                           0.94     10050
   macro avg       0.95      0.94      0.94     10050
weighted avg       0.95      0.94      0.94     10050

Accuracy: 0.9439800995024875
```



Learned Hyperparameters

Hidden Layers = 64

Learning Rate Initial = 0.001

Accuracy = 0.94

Random Forest Classifier

Hyperparameters: Max Depth and Max Leaf Nodes

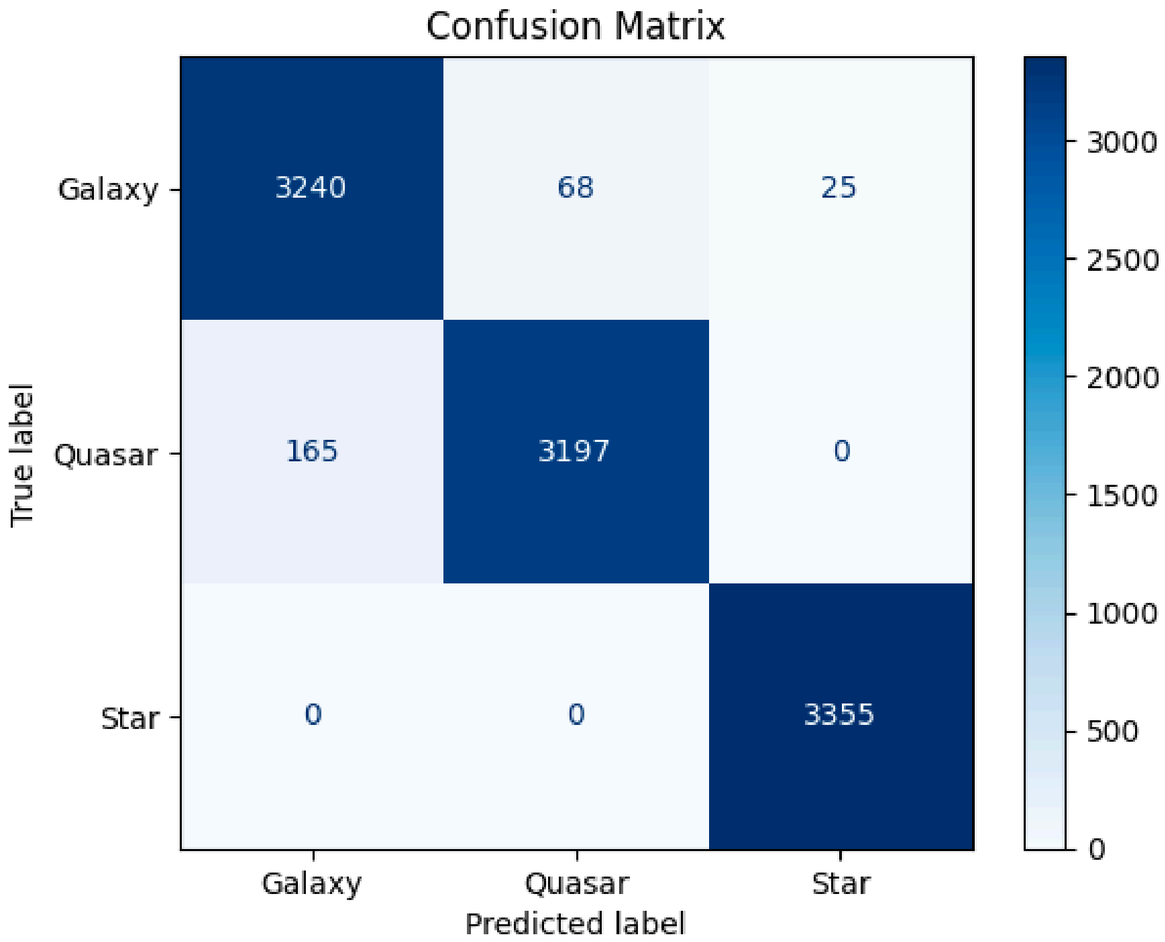
Max Depth: 5, 7, 9

Max Leaf Nodes: 50, 100, 150

+ Code + Markdown

```
#Random Forest
classifier = RandomForestClassifier()

param_grid_5 = {'max_depth': [5, 7, 9],
               'max_leaf_nodes': [50, 100, 150]}
grid_5 = GridSearchCV(classifier, param_grid_5, refit = True, verbose = 3, error_score = 'raise', cv=2)
grid_5.fit(x_train, y_train)

print(grid_5.best_params_)

print(grid_5.best_estimator_)

grid_predictions_5 = grid_5.predict(x_train)

print(classification_report(y_train, grid_predictions_5))
accuracy_5 = accuracy_score(y_train, grid_predictions_5)
print("Accuracy:", accuracy_5)
```

```
Output exceeds the size limit. Open the full output data in a text editor
Fitting 2 folds for each of 9 candidates, totalling 18 fits
[CV 1/2] END ....max_depth=5, max_leaf_nodes=50;, score=0.956 total time=   0.4s
 [CV 2/2] END ...max_depth=9, max_leaf_nodes=150;, score=0.966 total time=   0.6s
{'max_depth': 9, 'max_leaf_nodes': 150}
RandomForestClassifier(max_depth=9, max_leaf_nodes=150)
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      3333
           1       0.98      0.95      0.96      3362
...
   macro avg       0.97      0.97      0.97     10050
weighted avg       0.97      0.97      0.97     10050

Accuracy: 0.9743283582089552
```



Learned Hyperparameters

Max Depth = 9

Max Leaf Nodes = 150

Accuracy = 0.97

AdaBoostClassifier

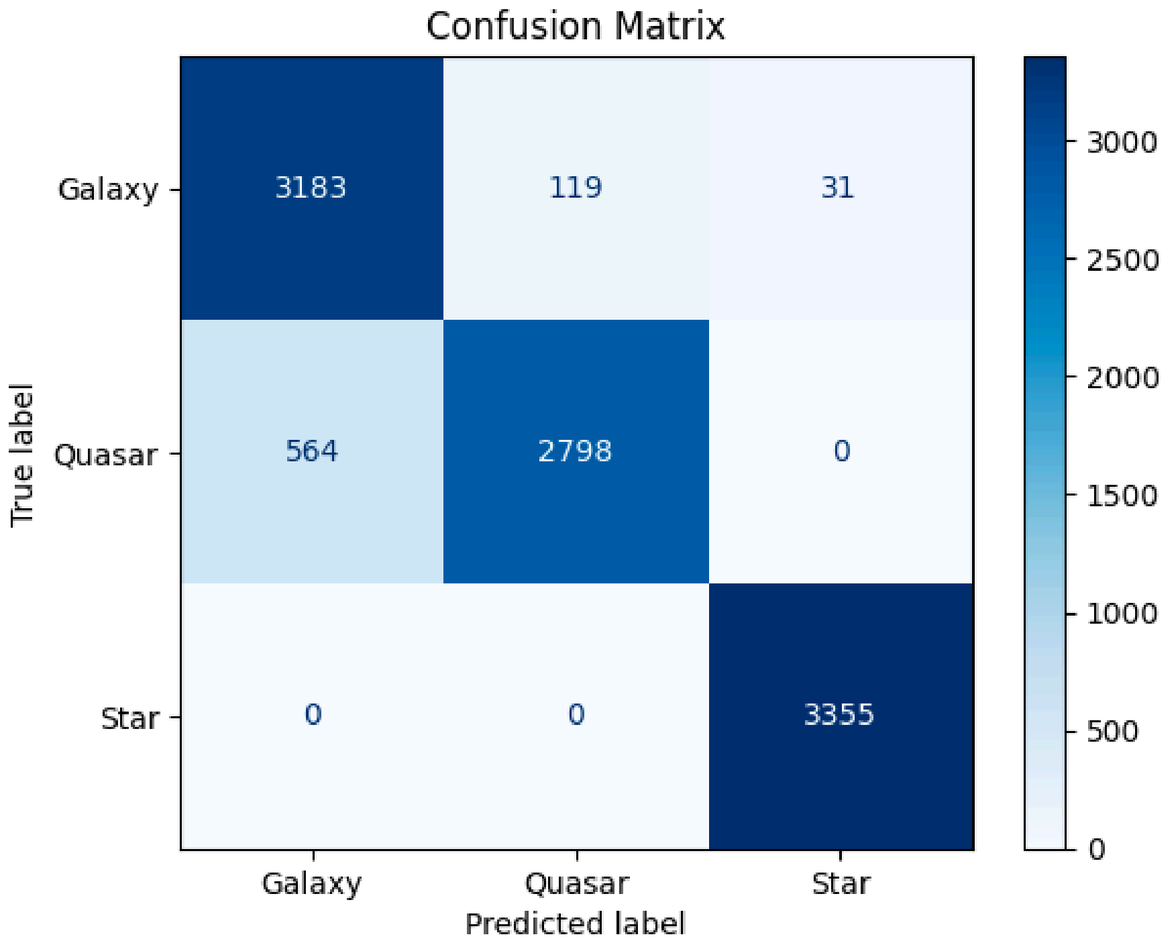
Hyperparameters: N estimators and Learning Rate

N_estimators: 50, 100, 150

Learning Rate: 0.1, 0.01

```
#AdaBoost
classifier = AdaBoostClassifier()

param_grid_3 = { 'n_estimators': [50, 100, 150],
                'learning_rate': [0.1, 0.01]}
grid_3 = GridSearchCV(classifier, param_grid_3, refit = True, verbose = 3, error_score = 'raise', cv=2)
grid_3.fit(x_train, y_train)

print(grid_3.best_params_)

print(grid_3.best_estimator_)

grid_predictions_3 = grid_3.predict(x_train)

print(classification_report(y_train, grid_predictions_3))
accuracy_3 = accuracy_score(y_train, grid_predictions_3)
print("Accuracy:", accuracy_3)
```

```
Fitting 2 folds for each of 6 candidates, totalling 12 fits
[CV 1/2] END learning_rate=0.1, n_estimators=50;, score=0.663 total time=   0.3s
[CV 2/2] END learning_rate=0.01, n_estimators=150;, score=0.917 total time=   0.8s
{'learning_rate': 0.01, 'n_estimators': 100}
AdaBoostClassifier(learning_rate=0.01, n_estimators=100)
              precision    recall  f1-score   support

           0       0.85      0.95      0.90      3333
           1       0.96      0.83      0.89      3362
           2       0.99      1.00      1.00      3355

    accuracy                           0.93     10050
   macro avg       0.93      0.93      0.93     10050
weighted avg       0.93      0.93      0.93     10050

Accuracy: 0.928955223880597
```



Learned Hyperparameters

N estimators = 100

Learning Rate = 0.01

Accuracy = 0.93

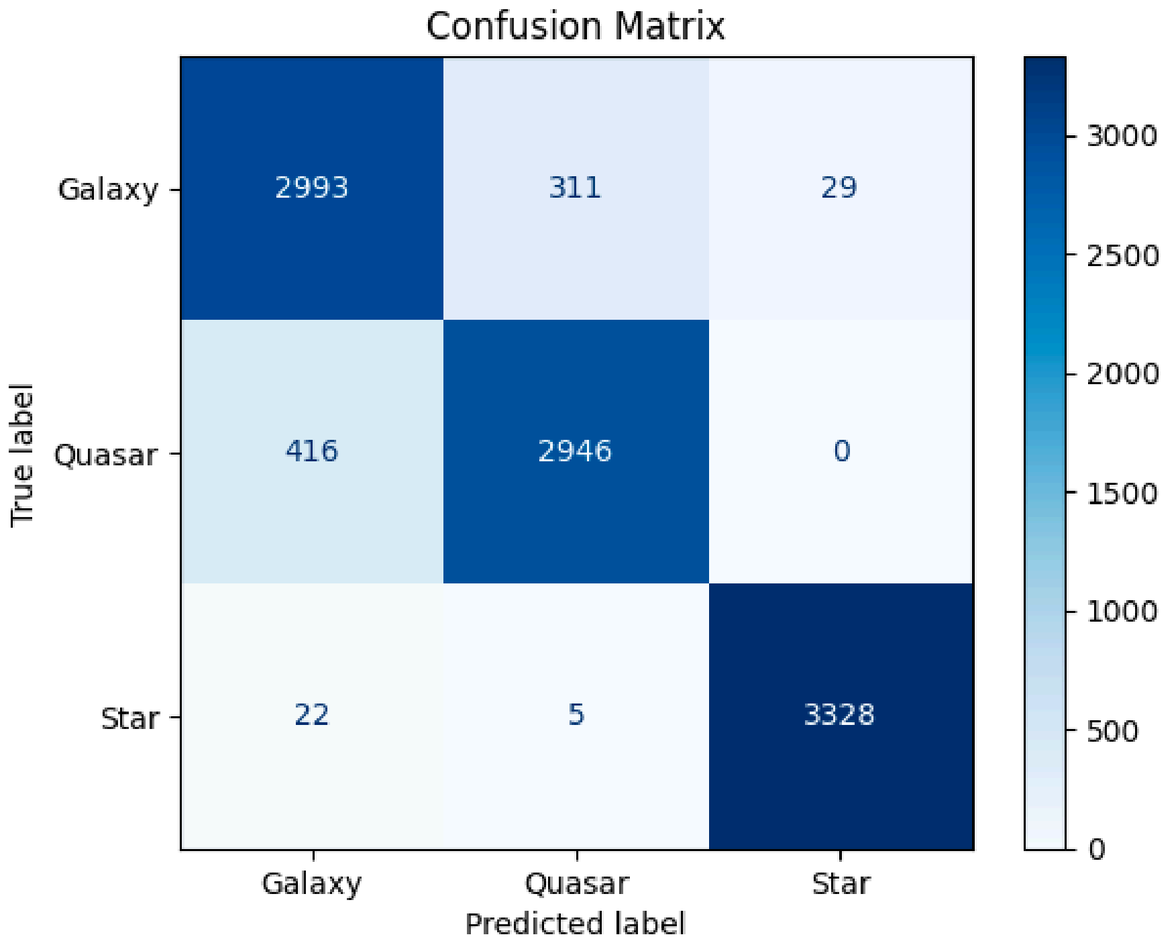
Gaussian Naive Bayes

Hyperparameter: Var Smoothing

Var_smoothing: 1e-9, 1e-10, 1e-11

```
#GaussianNB
classifier = GaussianNB()

param_grid_4 = {'var_smoothing': [1e-9, 1e-10, 1e-11]}
grid_4 = GridSearchCV(classifier, param_grid_4, refit = True, verbose = 3, error_score = 'raise', cv=2)
grid_4.fit(x_train, y_train)

print(grid_4.best_params_)

print(grid_4.best_estimator_)

grid_predictions_4 = grid_4.predict(x_train)

print(classification_report(y_train, grid_predictions_4))
accuracy_4 = accuracy_score(y_train, grid_predictions_4)
print("Accuracy:", accuracy_4)
```

```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
[CV 1/2] END ..............var_smoothing=1e-09;, score=0.925 total time=   0.0s
[CV 2/2] END ..............var_smoothing=1e-09;, score=0.919 total time=   0.0s
[CV 1/2] END ..............var_smoothing=1e-10;, score=0.925 total time=   0.0s
[CV 2/2] END ..............var_smoothing=1e-10;, score=0.919 total time=   0.0s
[CV 1/2] END ..............var_smoothing=1e-11;, score=0.925 total time=   0.0s
[CV 2/2] END ..............var_smoothing=1e-11;, score=0.919 total time=   0.0s
{'var_smoothing': 1e-09}
GaussianNB()
              precision    recall  f1-score   support

           0       0.87      0.90      0.88      3333
           1       0.90      0.88      0.89      3362
           2       0.99      0.99      0.99      3355

    accuracy                           0.92     10050
   macro avg       0.92      0.92      0.92     10050
weighted avg       0.92      0.92      0.92     10050

Accuracy: 0.922089552238806
```



Learned Hyperparameter

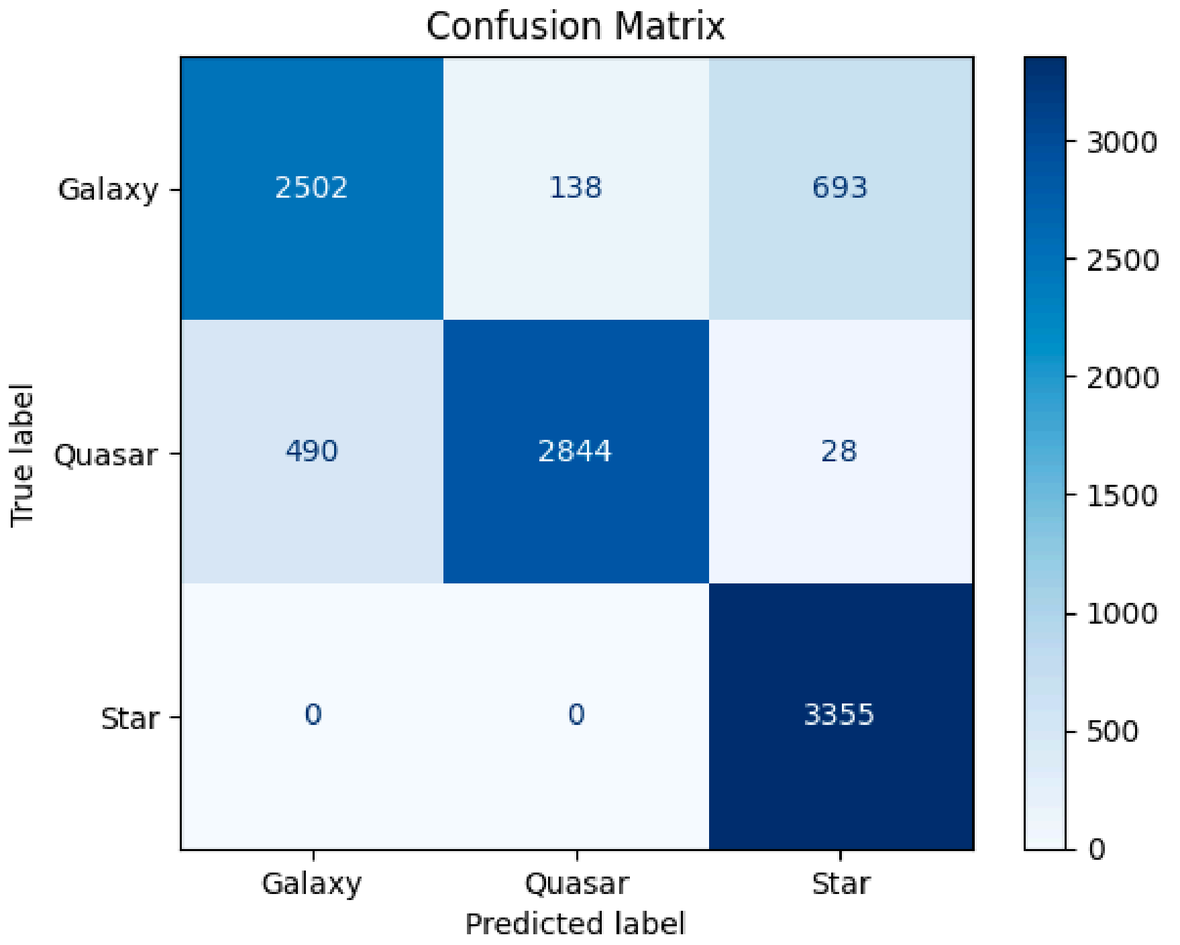
Var Smoothing = 1e-09

Accuracy = 0.92

Logistic Regression

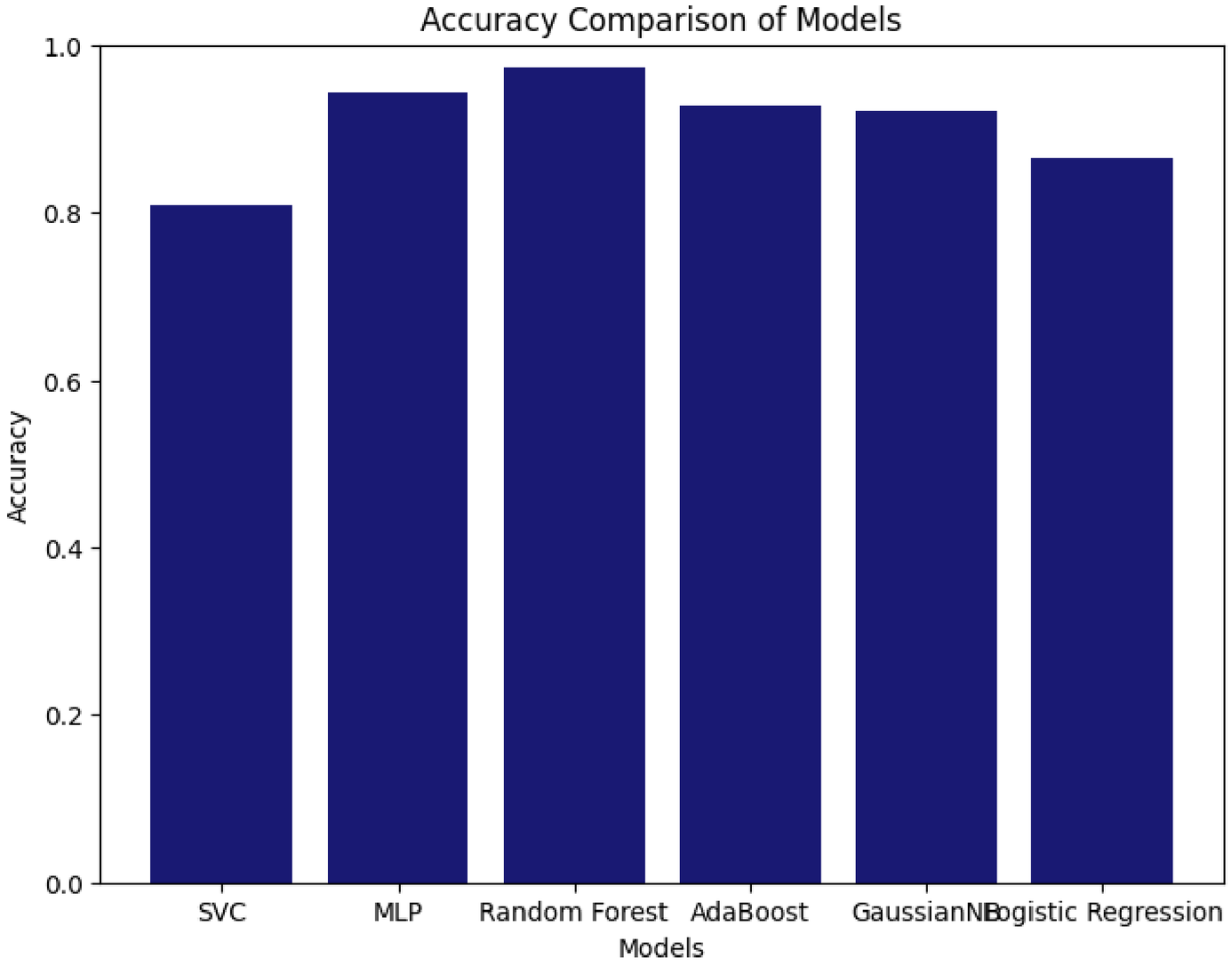
```
#logistici regression
classifier = LogisticRegression()

classifier.fit(x_train, y_train)

predictions = classifier.predict(x_train)

print(classification_report(y_train, predictions))
accuracy_6 = accuracy_score(y_train, predictions)
print("Accuracy:", accuracy_6)
```

```
              precision    recall  f1-score   support

           0       0.84      0.75      0.79      3333
           1       0.95      0.85      0.90      3362
           2       0.82      1.00      0.90      3355

    accuracy                           0.87     10050
   macro avg       0.87      0.87      0.86     10050
weighted avg       0.87      0.87      0.86     10050

Accuracy: 0.865771144278607
```



Accuracy = 0.87

What is best?



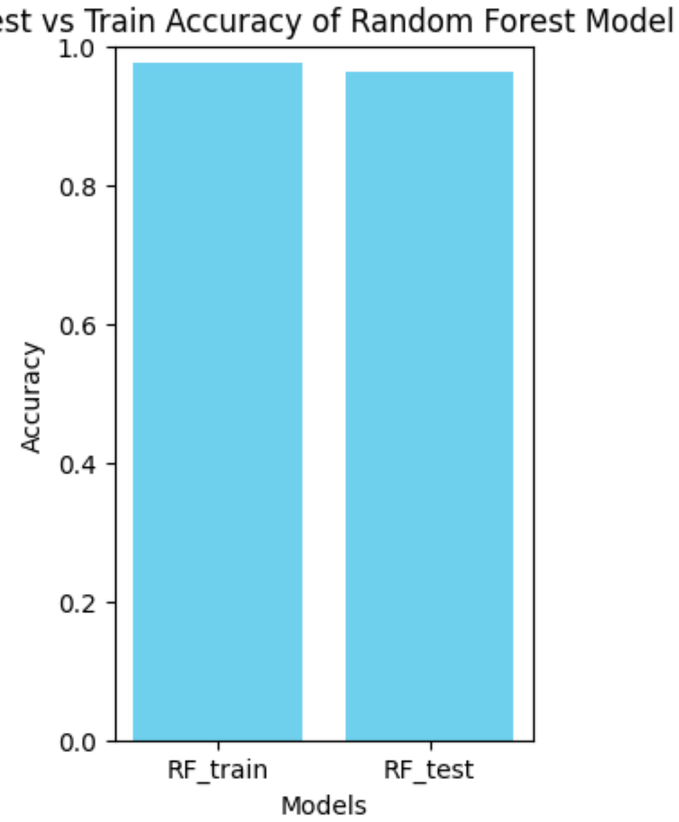
Random Forest gives **best** training accuracy of 0.97

SVC gives **worst** training accuracy of 0.81

Generalisation Error

Achieving a high training accuracy of 0.97 indicates effective learning from the training set. The slightly lower test accuracy of 0.96 suggests **good generalization performance**.

The small difference between training and test accuracies implies **minimal overfitting**. These results indicate low generalization error, highlighting the model's ability to make accurate predictions on new data.



Resources Used

Scikit Documentation
Kaggle Dataset

Thank You!