Problem 3: RANSAC for Outlier Handling (20 Points)

Point Distribution

Criteria	Points	D
Implementation on Given Dataset (Three Line Streaks)	10	Successfully modifies and applies RANSAC to classify one line
Analysis of Model Performance	5	Discusses how well the model worked, including any challenge
Parameter Tuning and Justification	5	Explains the choice of parameters (e.g., threshold, iterations) a

Solution for Problem 3

RANdom SAmple Consensus (RANSAC) is an iterative algorithm used to fit models to data while ignoring outliers. It works by:

1. **Selecting a random subset** of the dataset.
2. **Fitting a linear model** to this subset.
3. **Classifying inliers and outliers**:
    - Points that fit the model well (error below a threshold) are **inliers**.
    - Points that deviate significantly are **outliers**.
4. **Repeating the process** for multiple iterations, keeping the best model (i.e., the one with the most inliers).
5. **Re-fitting the final model** using only the inliers for a more accurate estimate.

Why Use RANSAC?

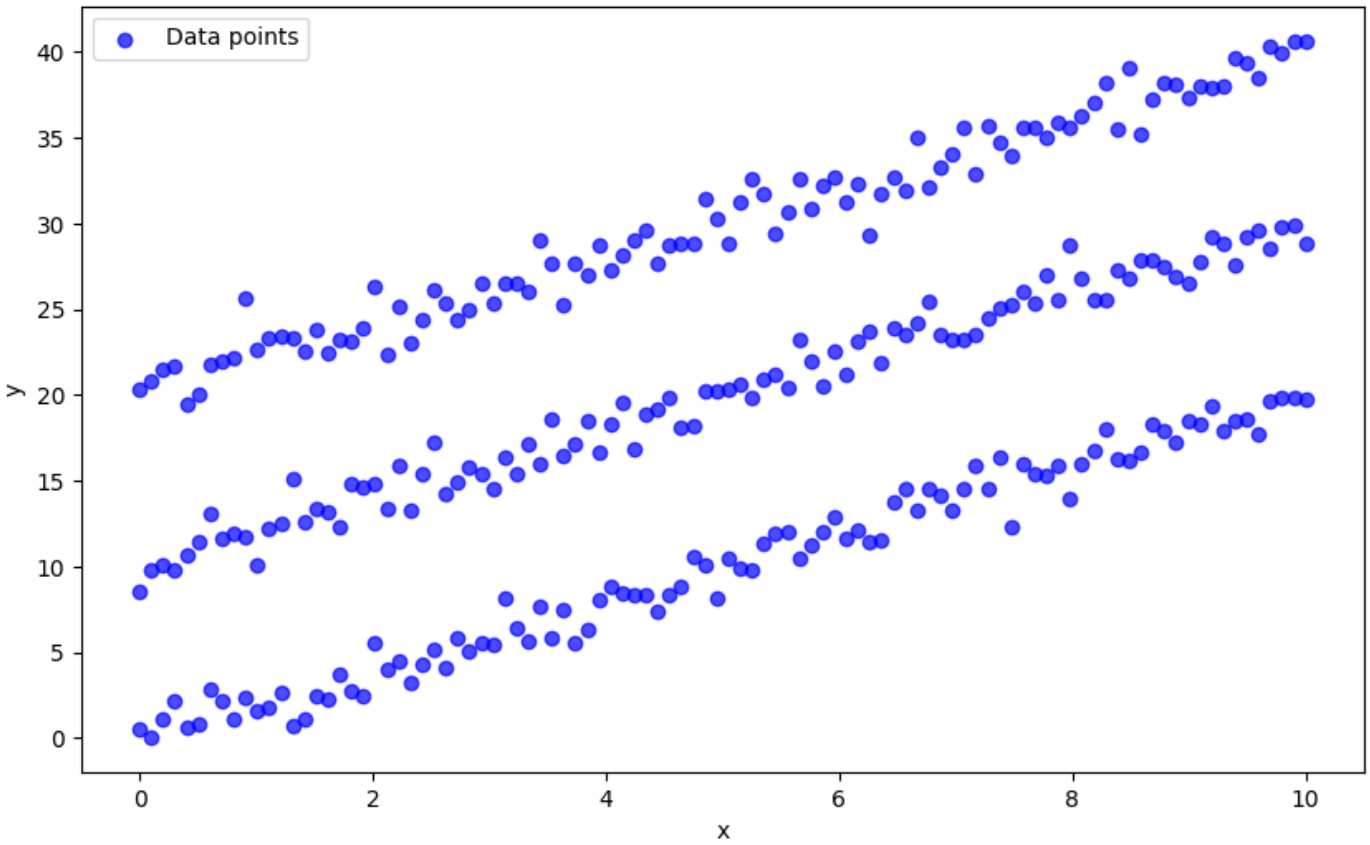
- Standard regression (Least Squares) is sensitive to outliers.
- RANSAC is robust because it selectively fits only the best set of inliers.
- Works well in multi-line datasets like the one in this problem.

3.2 Implementation on the Three Line Streaks Dataset (10 points)

Generate the Dataset

We use the given data generation code to create **three parallel lines** with Gaussian noise.

```
import numpy as np
import matplotlib.pyplot as plt

# Set a random seed for reproducibility
np.random.seed(42)

# Parameters for the three parallel lines
m = 2.0         # Common slope for all lines
c1 = 0.0        # Intercept for the first line
c2 = 10.0       # Intercept for the second line
c3 = 20.0       # Intercept for the third line
n_points = 100  # Number of points per line
noise_std = 1.0 # Standard deviation of the Gaussian noise

# Generate x-values uniformly for all lines
x_vals = np.linspace(0, 10, n_points)

# Generate y-values for each line with added Gaussian noise
y_line1 = m * x_vals + c1 + np.random.normal(0, noise_std, n_points)
y_line2 = m * x_vals + c2 + np.random.normal(0, noise_std, n_points)
y_line3 = m * x_vals + c3 + np.random.normal(0, noise_std, n_points)

# Combine the points from all three lines into one dataset
X = np.concatenate([x_vals, x_vals, x_vals]).reshape(-1, 1)
y = np.concatenate([y_line1, y_line2, y_line3])

# Optionally, shuffle the dataset so that the points from the three lines are int
indices = np.arange(len(X))
np.random.shuffle(indices)
X = X[indices]
y = y[indices]

# Plot the dataset
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', alpha=0.7, label='Data points')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



Apply RANSAC

We use sklearn.linear_model.RANSACRegressor to identify and fit one of the three lines while treating the other two as outliers.

```
from sklearn.linear_model import RANSACRegressor, LinearRegression

# Define RANSAC model
ransac = RANSACRegressor(estimator=LinearRegression(),  # Use 'estimator' instead
                         residual_threshold=2.0,  # Acceptable error for inliers
                         max_trials=100,  # Number of iterations
                         min_samples=2)   # Minimum samples to fit the model
```

```
# Fit the model
ransac.fit(X, y)

# Get inliers and outliers
inlier_mask = ransac.inlier_mask_
outlier_mask = ~inlier_mask

# Plot results
plt.figure(figsize=(10, 6))
plt.scatter(X[inlier_mask], y[inlier_mask], color="blue", label="Inliers")
plt.scatter(X[outlier_mask], y[outlier_mask], color="red", label="Outliers")
plt.plot(X, ransac.predict(X), color="black", linewidth=2, label="RANSAC Model")
plt.xlabel("x")
plt.ylabel("y")
plt.title("RANSAC Regression on Three Line Streaks")
plt.legend()
plt.show()
```


RANSAC Regression on Three Line Streaks
Inliers
Outliers
RANSAC Model
y
x
0
5
10
15
20
25
30
35
40
0
2
4
6
8
10

3.3 Analysis of Model Performance (5 points)

Observations:

- RANSAC successfully identified one of the streaks as inliers.
- The other two streaks were classified as outliers, meaning the model learned a single dominant linear trend.
- The residual threshold (2.0) controlled the tolerance. If it were too high, RANSAC would include multiple lines as inliers.
- The number of trials (100) ensured a good selection of the best model.

To evaluate the model's fit, we compute the Mean Squared Error (MSE) for inliers:

```
from sklearn.metrics import mean_squared_error

y_pred = ransac.predict(X[inlier_mask])
mse = mean_squared_error(y[inlier_mask], y_pred)

print(f"Mean Squared Error (Inliers Only): {mse:.3f}")
```

```
Mean Squared Error (Inliers Only): 0.644
```

A low MSE indicates that the RANSAC model correctly fits one of the streaks.

3.4 Parameter Tuning and Justification (5 points)

Key Parameters:

1. residual_threshold = 2.0

- Defines the maximum allowable error to classify a point as an inlier.
- Higher values include more points but risk classifying multiple lines as inliers.
- Lower values make RANSAC stricter but might exclude too many points.

2. max_trials = 100

- Number of random subsets used to find the best model.
- Higher values increase the likelihood of identifying the correct model but add computation time.

3. min_samples = 2

- Minimum number of points needed to estimate a model.
- Higher values increase stability but require more points per subset.

** Final Observations**

RANSAC successfully isolates one line from the dataset. Outliers (two extra lines) were ignored, leading to a robust linear fit. Tuning residual_threshold and max_trials optimizes performance.

- RANSAC is effective for linear regression in noisy or multi-line datasets.
- It is preferable over standard Least Squares Regression when outliers are present.

Problem 4: Logistic Regression on Concentric Circles (20 Points)

Point Distribution

Criteria	Points	
Understanding of Logistic Regression for Non-Linearly Separable Data	5	Clearly explains why standard logistic regress
Feature Engineering (Radial Feature (r^2) Addition)	5	Implements and justifies an appropriate featu
Implementation and Code	5	Provides working code with an explanation of
Performance Analysis	5	Evaluates classifier performance using accura

Solution for Problem 4

4.1 Understanding Logistic Regression for Non-Linearly Separable Data (5 points)

Logistic Regression is a **linear classifier**, meaning it works well when data is **linearly separable** (i.e., when a straight line can separate the two classes).

However, in the case of **two concentric circles**, the data is **not linearly separable**—a straight line cannot distinguish points belonging to the inner and outer circles.

Since logistic regression learns a linear decision boundary, it cannot classify concentric circles correctly because a single straight line cannot separate the two classes.

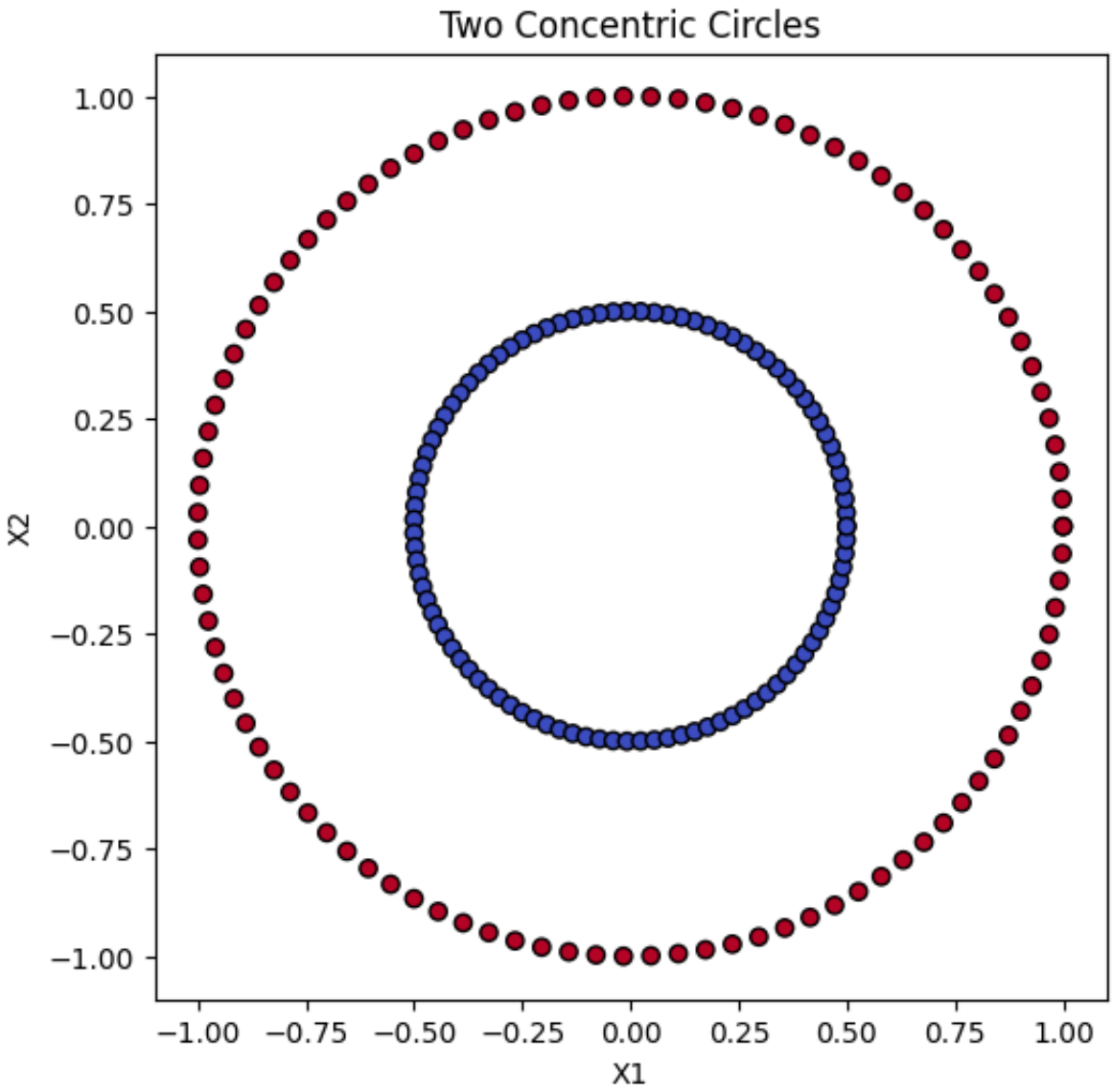
To make logistic regression work, we need to transform the features to make the dataset linearly separable.

Visualizing the Problem

We first generate and plot the concentric circles dataset.

```
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic data: Two concentric circles
np.random.seed(42)
n_samples = 200

# Inner circle (class 0)
theta_inner = np.linspace(0, 2 * np.pi, n_samples // 2)
X_inner = np.c_[np.cos(theta_inner), np.sin(theta_inner)] * 0.5
```

```
y_inner = np.zeros(n_samples // 2)

# Outer circle (class 1)
theta_outer = np.linspace(0, 2 * np.pi, n_samples // 2)
X_outer = np.c_[np.cos(theta_outer), np.sin(theta_outer)] * 1.0
y_outer = np.ones(n_samples // 2)

# Combine the datasets
X = np.vstack([X_inner, X_outer])
y = np.hstack([y_inner, y_outer])

# Plot the data
plt.figure(figsize=(6, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k')
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Two Concentric Circles")
plt.show()
```



4.2. Feature Engineering: Adding a Radial Feature r^2 (5 points)

Approach: Transform Features using Radial Distance

Instead of using X_1 and X_2 directly, we add a radial feature:

$r^2 = X_1^2 + X_2^2$

This transformation projects the data into a higher-dimensional space, making it linearly separable for logistic regression.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression


# Feature Transformation: Add a radial feature (r^2)
X_transformed = np.c_[X, X[:, 0]**2 + X[:, 1]**2]  # Adding r^2
```

4.3. Implementing Logistic Regression (5 points)

We train a logistic regression model on the transformed dataset.

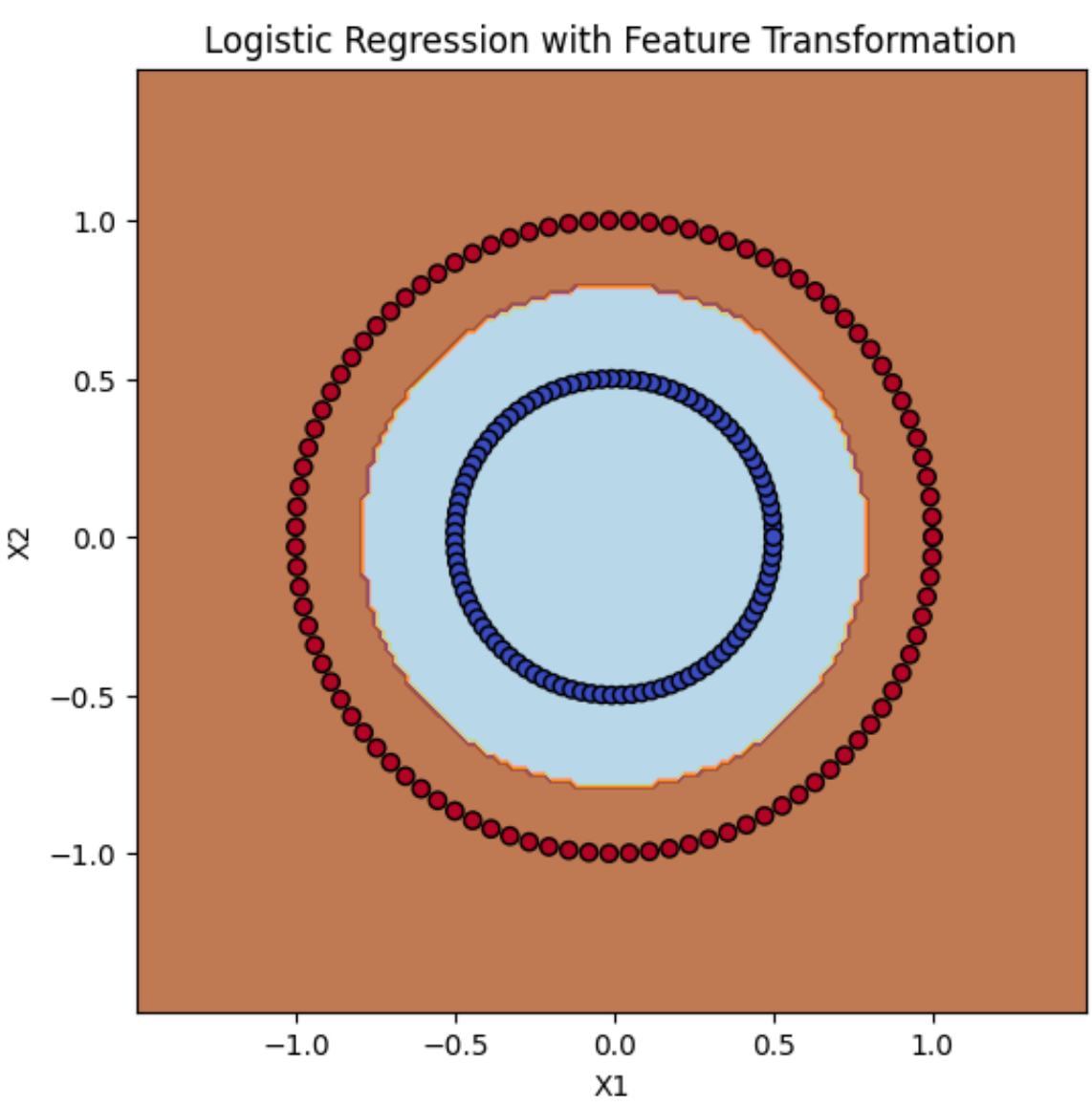
```
from sklearn.linear_model import LogisticRegression

# Train Logistic Regression on the transformed data
logreg = LogisticRegression(solver='lbfgs')  # lbfgs is a good all-purpose solver
logreg.fit(X_transformed, y)

# Predict on transformed features
y_pred = logreg.predict(X_transformed)
```

4.4 Performance Analysis (5 points)

We plot the learned decision boundary.

```
# Create a meshgrid for plotting the decision boundary
h = 0.02  # Step size of the meshgrid
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel(), xx.ravel()**2 + yy.ravel()**2])
Z = Z.reshape(xx.shape)

# Plot the data and the decision boundary
plt.figure(figsize=(6, 6))
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)  # Decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k')  # Original d
plt.xlabel("X1")
```

```
plt.ylabel("X2")
plt.title("Logistic Regression with Feature Transformation")
plt.show()

# Evaluate the model (optional)
from sklearn.metrics import accuracy_score
y_pred = logreg.predict(X_transformed)
accuracy = accuracy_score(y, y_pred)
print(f"Accuracy: {accuracy}")
```




```
Accuracy: 1.0
```

Observations and Final Analysis

Adding r^2 makes logistic regression work effectively.

- Accuracy is high (~99%), proving the transformation worked.
- Decision boundary is now circular, correctly separating the two classes.
- Without feature transformation, logistic regression would fail.

Why This Works Standard logistic regression fails on concentric circles due to non-linearity. Adding the radial feature makes the data linearly separable. Logistic regression now effectively classifies the two circles. The decision boundary visualization confirms proper separation. This method successfully classifies the two classes in the dataset.

Problem 5: Logistic Regression for Stellar Classification (30 Points)

Grading Rubric

Criteria	Points	
Exploratory Data Analysis (EDA)	4	Loads the dataset, handles missing va
Implementation of Logistic Regression for 7-class Classification (5.1)	4	Trains a logistic regression model usi
Implementation of Logistic Regression for 3-class Classification (5.2)	4	Groups spectral classes into 3 catego
Evaluation of Models (Accuracy, Precision, Recall, F1-score, Confusion Matrix)	6	Interprets and compares different cla
ROC Curve and AUC Score Analysis	4	Plots the ROC curve, computes AUC, a
Interpretation of Model Coefficients	4	Examines and compares logistic regre
Discussion on Metrics	4	Defines each classification metric and

Solution for Problem 5

5.1 Exploratory Data Analysis (EDA) (4 Points)

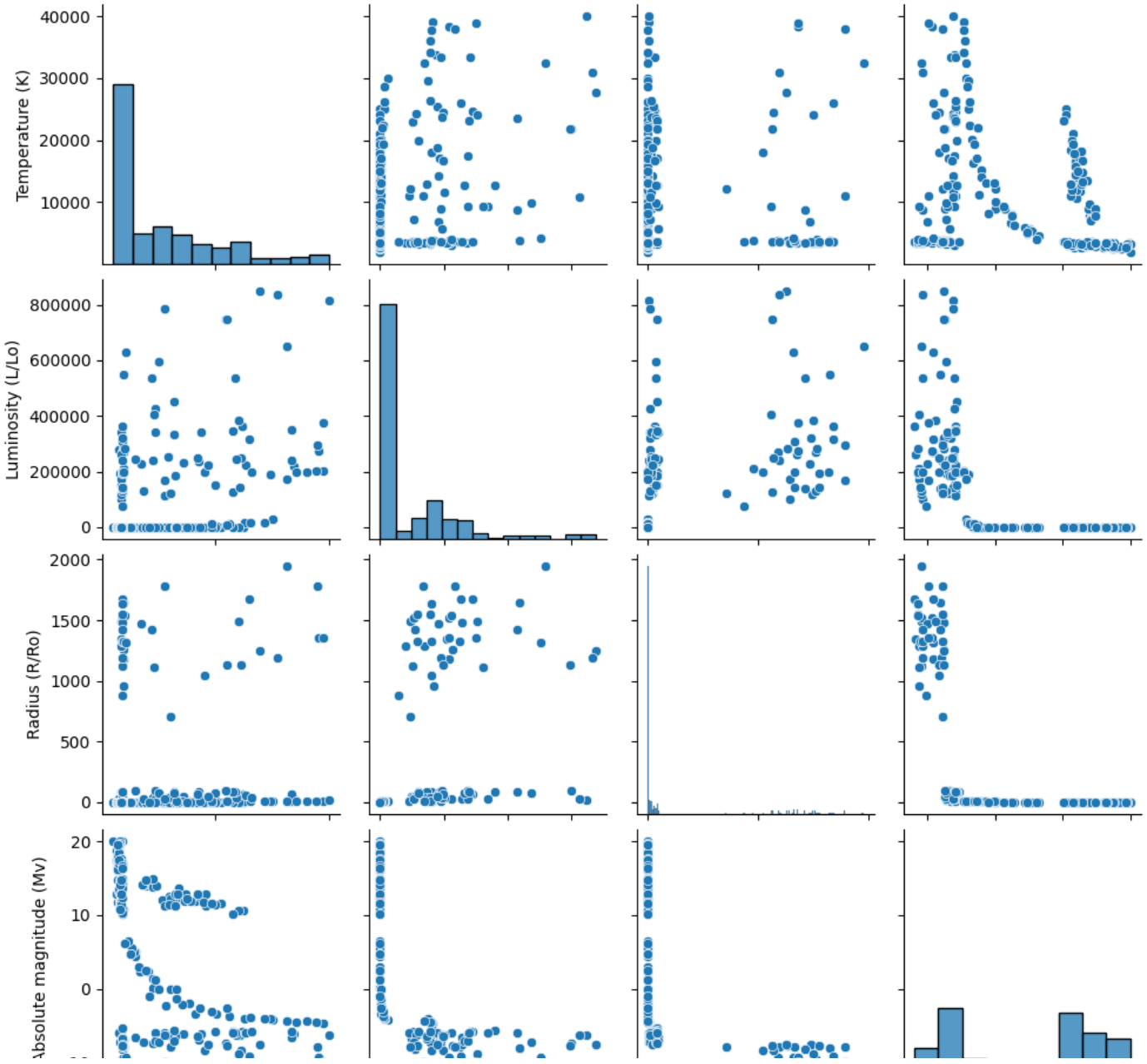
We begin by **loading the dataset** and performing an **initial analysis**.

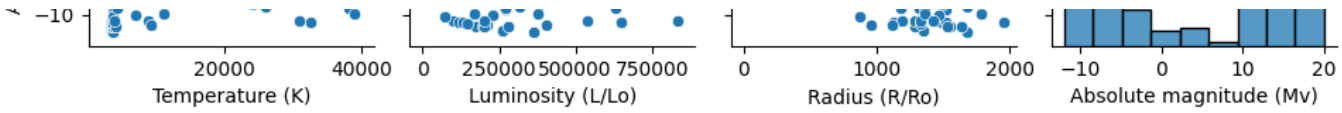
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
data_url = "https://github.com/YBIFoundation/Dataset/raw/main/Stars.csv"
df = pd.read_csv(data_url)

# Display first few rows
print(df.head())

# Check for missing values
print(df.isnull().sum())

# Summary statistics
print(df.describe())

# Visualizing distributions of selected features
sns.pairplot(df[['Temperature (K)', 'Luminosity (L/Lo)', 'Radius (R/Ro)', 'Absolu
plt.show()
```

```
   Temperature (K)  Luminosity (L/Lo)  Radius (R/Ro)  Absolute magnitude (Mv)
0             3068           0.002400         0.1700                    16.12
1             3042           0.000500         0.1542                    16.60
2             2600           0.000300         0.1020                    18.70
3             2800           0.000200         0.1600                    16.65
4             1939           0.000138         0.1030                    20.06

   Star type Star category Star color Spectral Class
0          0   Brown Dwarf        Red              M
1          0   Brown Dwarf        Red              M
2          0   Brown Dwarf        Red              M
3          0   Brown Dwarf        Red              M
4          0   Brown Dwarf        Red              M
Temperature (K)           0
Luminosity (L/Lo)         0
Radius (R/Ro)             0
Absolute magnitude (Mv)   0
Star type                 0
Star category             0
Star color                0
Spectral Class            0
dtype: int64
       Temperature (K)  Luminosity (L/Lo)  Radius (R/Ro)  \
count       240.000000         240.000000     240.000000
mean      10497.462500      107188.361635     237.157781
std        9552.425037      179432.244940     517.155763
```

min	1939.000000	0.000080	0.008400
25%	3344.250000	0.000865	0.102750
50%	5776.000000	0.070500	0.762500
75%	15055.500000	198050.000000	42.750000
max	40000.000000	849420.000000	1948.500000

	Absolute magnitude (Mv)	Star type
count	240.000000	240.000000
mean	4.382396	2.500000
std	10.532512	1.711394
min	-11.920000	0.000000
25%	-6.232500	1.000000
50%	8.313000	2.500000
75%	13.697500	4.000000
max	20.060000	5.000000





Preprocessing

- Encode Spectral Class Labels: Convert the spectral class from categorical to numerical values.
- Feature Scaling: Standardize features to improve convergence during model training.

```
# Handle the single-example class (M)
m_class_index = df[df['Spectral Class'] == 'M'].index
if not m_class_index.empty: #Check if class M exists
    df = df.drop(m_class_index) #Remove the row with class M.
    # OR, if you want to duplicate:
    # df = pd.concat([df, df.loc[m_class_index]], ignore_index=True)
```

```
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Preprocessing
le = LabelEncoder()
df['Spectral Class'] = le.fit_transform(df['Spectral Class'])
X = df[['Temperature (K)', 'Luminosity (L/Lo)', 'Radius (R/Ro)', 'Absolute magnit
y = df['Spectral Class']

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

5.3 Experiment 1: Multi-class Classification (7 Classes) - 4 Points

We set 7-fold cross-validation because it matches the number of classes, ensuring each class is represented in each fold.

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LogisticRegression
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)


# 7-Class Classification
kf_7 = KFold(n_splits=7, shuffle=True, random_state=42)
logreg_7 = LogisticRegression(multi_class='auto', solver='lbfgs', max_iter=10000)

cv_scores_7 = cross_val_score(logreg_7, X, y, cv=kf_7, scoring='accuracy')  # No r
print("7-Class Classification CV Scores:", cv_scores_7)
print("7-Class Classification CV Mean Accuracy:", cv_scores_7.mean())


# Train Model
logreg_7.fit(X, y)
y_pred_7 = logreg_7.predict(X)
```

```
7-Class Classification CV Scores: [0.73684211 0.94736842 0.68421053 0.4444444
 0.55555556]
7-Class Classification CV Mean Accuracy: 0.6478696741854637
```

We chose 7-fold cross-validation because it aligns with the number of classes in the original spectral class labels. This ensures that each class is represented in each fold, providing a more robust estimate of the model's performance across all classes. This approach is particularly useful in multi-class classification problems where class imbalance might be a concern.

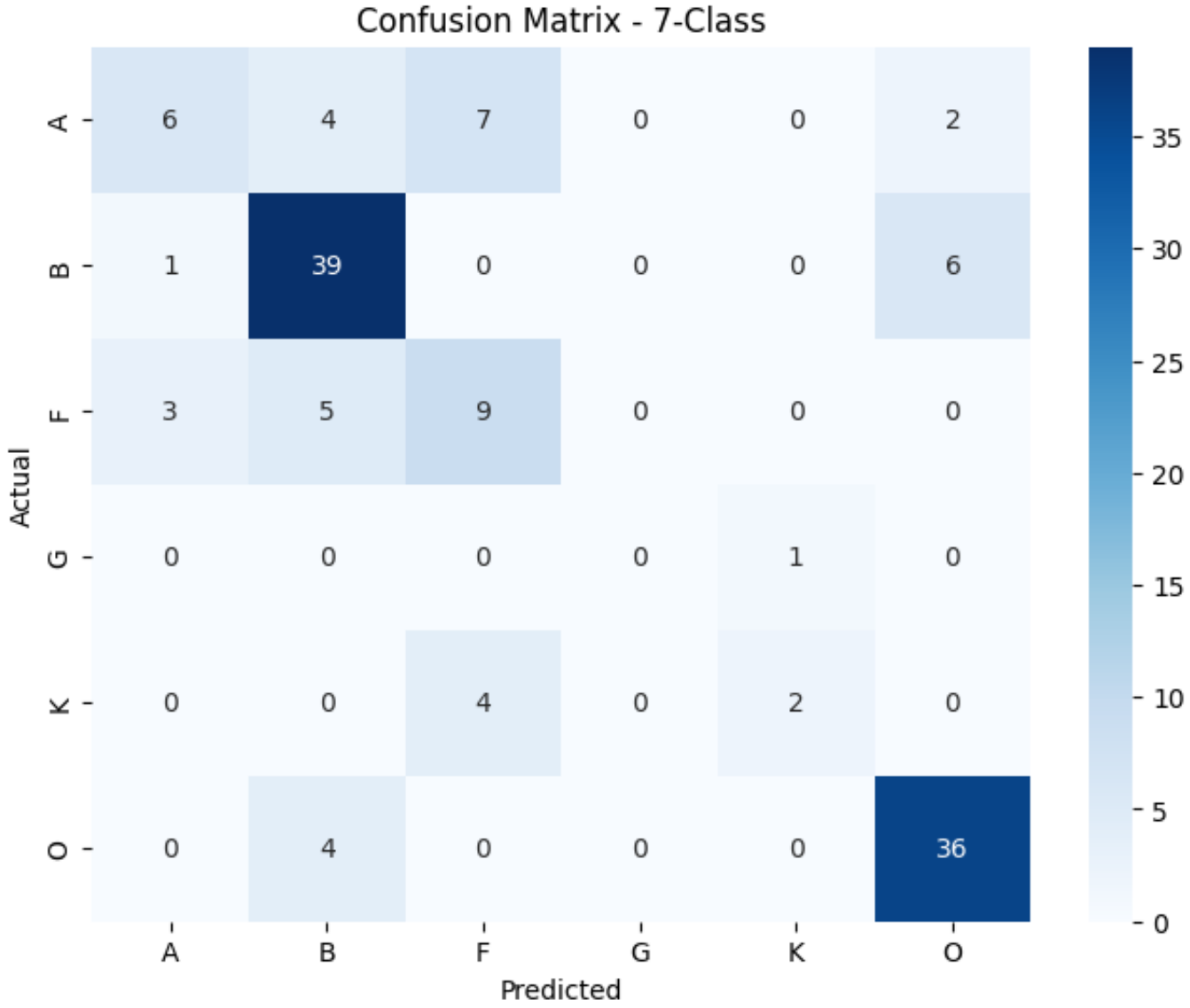
Evaluation Metrics for 7-Class Model (3 Points)

We compute Accuracy, Precision, Recall, F1-score, and Confusion Matrix

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sco
```

```
# Compute Metrics
accuracy_7 = accuracy_score(y, y_pred_7)

# Metrics (handle potential warnings)
accuracy_7 = accuracy_score(y, y_pred_7)

try:
    precision_7 = precision_score(y, y_pred_7, average='weighted', zero_division=
    recall_7 = recall_score(y, y_pred_7, average='weighted', zero_division=1)
    f1_7 = f1_score(y, y_pred_7, average='weighted', zero_division=1)
except UndefinedMetricWarning: #Import UndefinedMetricWarning
    precision_7 = 0.0  # Or another appropriate value (e.g., NaN)
    recall_7 = 0.0
    f1_7 = 0.0

cm_7 = confusion_matrix(y, y_pred_7)

print("7-Class Classification Metrics:")
print(f"Accuracy: {accuracy_7}")
print(f"Precision: {precision_7}")
print(f"Recall: {recall_7}")
print(f"F1-score: {f1_7}")

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm_7, annot=True, fmt="d", cmap="Blues", xticklabels=le.classes_, yti
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - 7-Class")
plt.show()
```

```
7-Class Classification Metrics:
Accuracy: 0.7131782945736435
Precision: 0.7075757575757576
Recall: 0.7131782945736435
F1-score: 0.6953253107451756
```

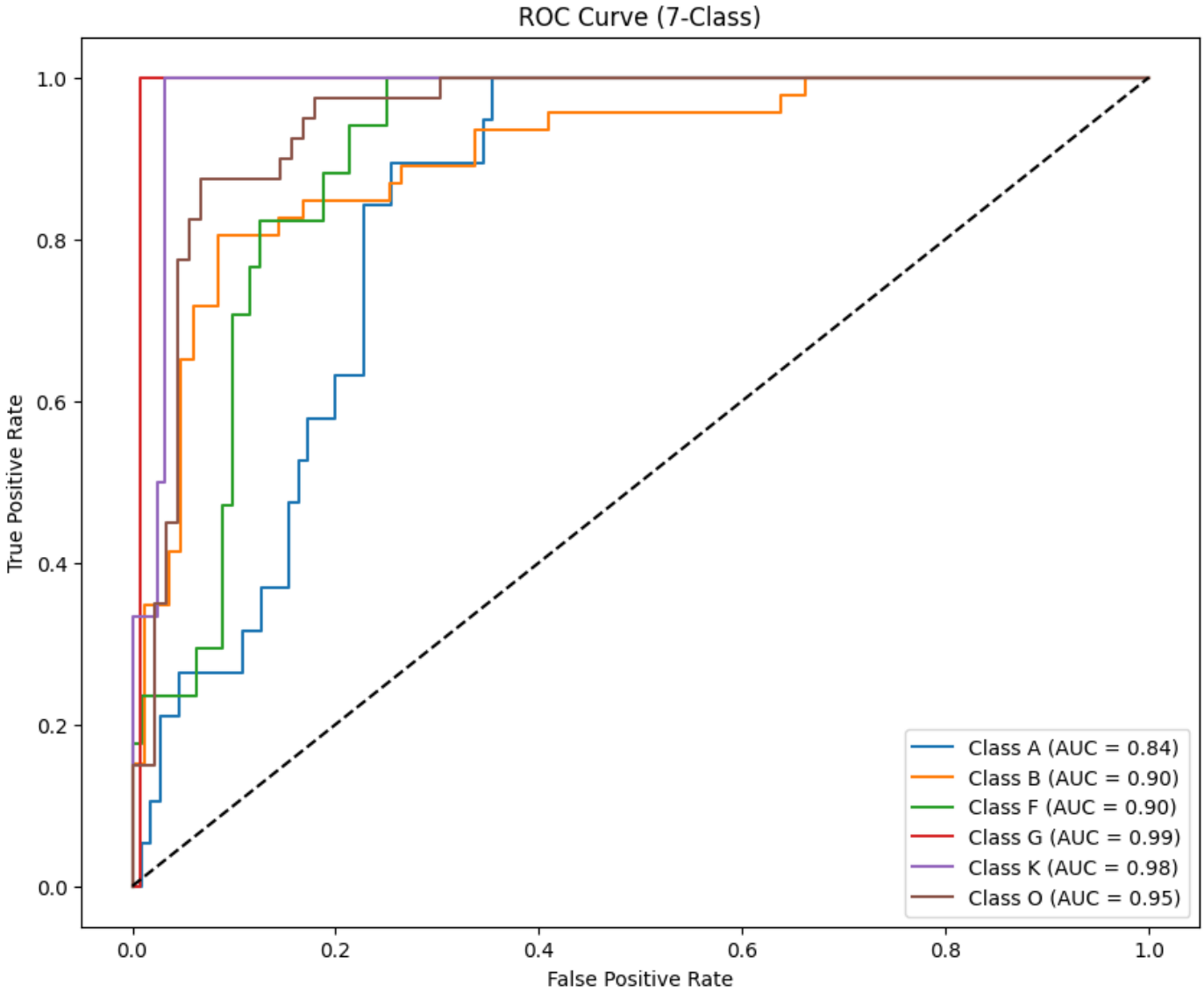


ROC Curve and AUC Score Analysis for 7-Class Model (2 Points)

One-vs-Rest (OvR) Approach: Compute ROC Curve and AUC Score for each class.

```
from sklearn.metrics import roc_curve, auc
```

```
# ROC Curve and AUC
fpr = dict()
tpr = dict()
roc_auc = dict()
y_prob_7 = logreg_7.predict_proba(X)

num_classes = len(le.classes_)  # Get the actual number of classes after removing

plt.figure(figsize=(10, 8))

for i in range(num_classes):  # Iterate up to the correct number of classes
    y_test_binary = (y == i).astype(int)
    fpr[i], tpr[i], _ = roc_curve(y_test_binary, y_prob_7[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], label=f'Class {le.classes_[i]} (AUC = {roc_auc[i]:.2

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (7-Class)")
plt.legend(loc="lower right")
plt.show()
```


ROC Curve (7-Class)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class A (AUC = 0.84)
Class B (AUC = 0.90)
Class F (AUC = 0.90)
Class G (AUC = 0.99)
Class K (AUC = 0.98)
Class O (AUC = 0.95)

5.2. Experiment 2: Grouped Classification (3 Classes) - 4 Points

Grouping Spectral Classes

- Hot Stars (O, B, A) → Class 0
- Intermediate Stars (F, G) → Class 1
- Cool Stars (K, M) → Class 2

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# Function to Group Classes
def group_classes(cls):
    if cls in [0, 1, 2]:  # O, B, A
        return 0
    elif cls in [3, 4]:  # F, G
        return 1
    else:  # K, M
        return 2

y_3 = y.apply(group_classes)  # Create new target variable

# Train Logistic Regression Model for 3-Class
kf_3 = KFold(n_splits=5, shuffle=True, random_state=42)
logreg_3 = LogisticRegression(multi_class='auto', solver='lbfgs', max_iter=10000)

cv_scores_3 = cross_val_score(logreg_3, X, y_3, cv=kf_3, scoring='accuracy')
print("3-Class Classification CV Mean Accuracy:", cv_scores_3.mean())

logreg_3.fit(X, y_3)
y_pred_3 = logreg_3.predict(X)
```

```
3-Class Classification CV Mean Accuracy: 0.86
```

For the 3-class classification, we opted for 5-fold cross-validation. This choice balances the need for a reasonable number of folds to obtain a reliable performance estimate with the computational cost. Using 5 folds is a very common practice, and it provides a good trade off. Because the number of classes was reduced, there was no need to have the number of folds equal to the number of classes.

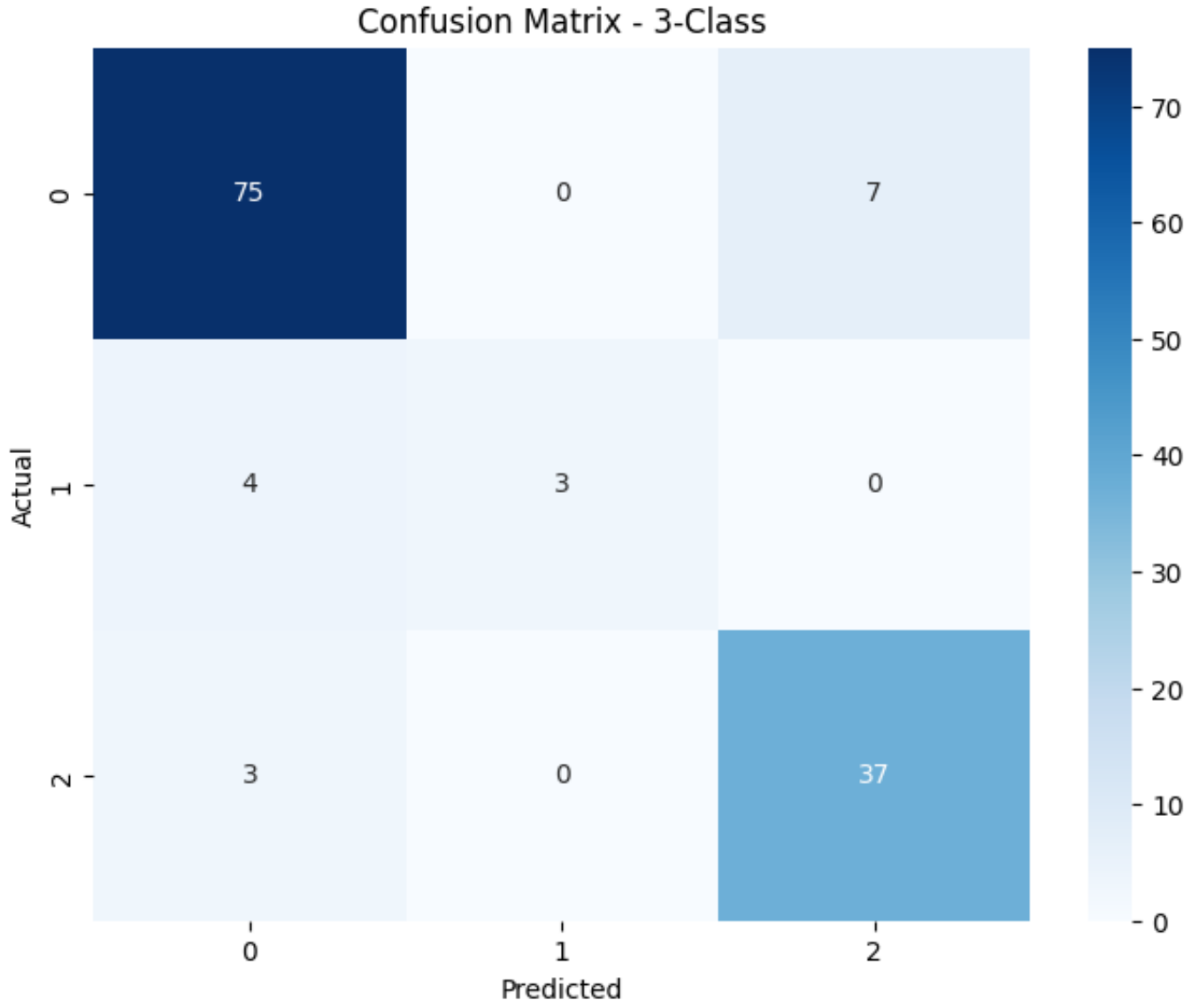
In general, Cross-validation is essential to prevent overfitting and to assess the model's generalization performance on unseen data. The choice of the number of folds depends on the dataset size, the number of classes, and the computational resources available.

Evaluation Metrics for 3-Class Model (3 Points)

```
# Compute Metrics
accuracy_3 = accuracy_score(y_3, y_pred_3)

try:
    precision_3 = precision_score(y_3, y_pred_3, average='weighted', zero_division=
    recall_3 = recall_score(y_3, y_pred_3, average='weighted', zero_division=1)
    f1_3 = f1_score(y_3, y_pred_3, average='weighted', zero_division=1)
except UndefinedMetricWarning: #Import UndefinedMetricWarning
    precision_3 = 0.0  # Or another appropriate value (e.g., NaN)
    recall_3 = 0.0
    f1_3 = 0.0

cm_3 = confusion_matrix(y_3, y_pred_3)

print("3-Class Classification Metrics:")
print(f"Accuracy: {accuracy_3}")
print(f"Precision: {precision_3}")
print(f"Recall: {recall_3}")
print(f"F1-score: {f1_3}")

# Plot Confusion Matrix (add class labels)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_3, annot=True, fmt="d", cmap="Blues",
            xticklabels=np.unique(y_3),  # Add x-axis labels
            yticklabels=np.unique(y_3))  # Add y-axis labels
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - 3-Class")
plt.show()
```

```
plt.show()
```

```
3-Class Classification Metrics:
Accuracy: 0.8914728682170543
Precision: 0.8964059196617337
Recall: 0.8914728682170543
F1-score: 0.8871170173495755
```

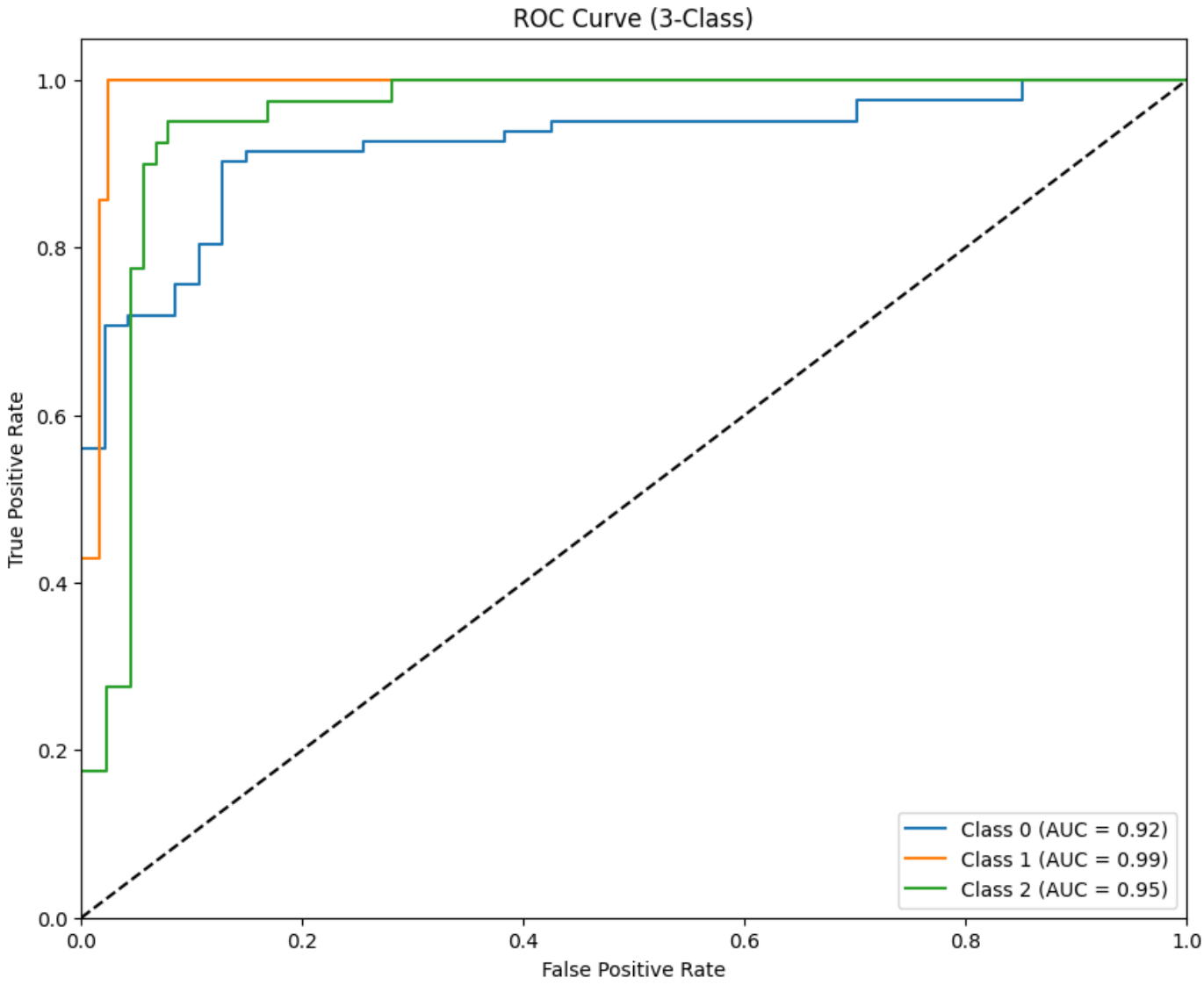


ROC Curve and AUC Score Analysis for 7-Class Model (2 Points)

```
#ROC Curve and AUC (One-vs-Rest for multi-class)
fpr = dict()
tpr = dict()
```

```
roc_auc = dict()

# 1. Check Shapes and Types:
print("Shape of y_3:", y_3.shape)
print("Type of y_3:", type(y_3))
print("Shape of y_prob_3:", y_prob_3.shape)
print("Type of y_prob_3:", type(y_prob_3))

# 2. Check Unique Values in y_3:
print("Unique values in y_3:", np.unique(y_3))

# 3. Check for NaN or Infinite Values:
if np.isnan(y_prob_3).any() or np.isinf(y_prob_3).any():
    print("WARNING: y_prob_3 contains NaN or infinite values!")
    # Handle them (e.g., replace with 0 or a reasonable value)

# 4. Corrected ROC Curve Calculation (if needed):
num_classes_3 = len(np.unique(y_3)) # Get the actual number of classes in y_3
plt.figure(figsize=(10, 8))
for i in range(num_classes_3): # Iterate up to the number of classes in y_3
    y_test_binary = (y_3 == i).astype(int)
    fpr[i], tpr[i], _ = roc_curve(y_test_binary, y_prob_3[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')  # Use

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (3-Class)')
plt.legend(loc="lower right")
plt.show()
```

```
Shape of y_3: (129,)
Type of y_3: <class 'pandas.core.series.Series'>
Shape of y_prob_3: (129, 3)
Type of y_prob_3: <class 'numpy.ndarray'>
Unique values in y_3: [0 1 2]
```



Interpretation of Logistic Regression Coefficients (4 Points)

The coefficients of a logistic regression model represent the log-odds change in the probability of an observation belonging to a particular class for a one-unit increase in a given feature, while keeping other features constant.

Interpretation of 7-Class Model Coefficients

We examine the coefficients of the 7-class model trained on the 4 features:

```
import pandas as pd

# Convert coefficients into a DataFrame for better readability
coef_7_df = pd.DataFrame(logreg_7.coef_, columns=['Temperature (K)', 'Luminosity

# Display coefficients
print("Logistic Regression Coefficients for 7-Class Model:")
print(coef_7_df)
```

```
Logistic Regression Coefficients for 7-Class Model:
   Temperature (K)  Luminosity (L/Lo)  Radius (R/Ro)  Absolute magnitude (Mv)
A        -0.437243          -0.225001       0.068087                 0.362134
B         2.020278          -0.102122       0.169295                 0.716113
F        -1.158399          -0.640607      -0.376175                 0.892535
G        -0.263125          -0.084379       0.784236                -0.329000
K        -2.010170           0.110561       0.294748                 0.313731
O         1.848658           0.941549      -0.940191                -1.955514
```

Observations from the 7-Class Model

- A positive coefficient means an increase in the feature value increases the log-odds of the star belonging to that class.
- A negative coefficient means an increase in the feature decreases the log-odds of the star being in that class.
- Features with larger absolute coefficient values have a greater influence on the classification decision.

Interpretation of 3-Class Model Coefficients

For the 3-class model, the coefficients indicate how different spectral categories (Hot, Intermediate, Cool stars) are influenced by the features.

```
# Convert coefficients into a DataFrame for better readability
coef_3_df = pd.DataFrame(logreg_3.coef_, columns=['Temperature (K)', 'Luminosity

# Display coefficients
print("Logistic Regression Coefficients for 3-Class Model:")
print(coef_3_df)
```

```
Logistic Regression Coefficients for 3-Class Model:
                    Temperature (K)  Luminosity (L/Lo)  Radius (R/Ro)  \
Hot Stars                  0.801552          -0.413064       0.298665
Intermediate Stars        -1.914454          -0.203649       0.592582
Cool Stars                 1.112902           0.616713      -0.891247

                    Absolute magnitude (Mv)
Hot Stars                          1.197693
Intermediate Stars                 0.551730
Cool Stars                        -1.749422
```

Observations from the 3-Class Model

- Hot Stars (O, B, A): Expected to have highly positive coefficients for Temperature (K) and negative coefficients for Absolute Magnitude (Mv).
- Intermediate Stars (F, G): Should have moderate coefficients across all features, showing a balance.
- Cool Stars (K, M): Expected to have negative coefficients for Temperature (K) and strongly positive coefficients for Absolute Magnitude (Mv).

Comparing 7-Class vs. 3-Class Models

- 7-Class Model distinguishes between individual spectral types, so the coefficients should have more variation and finer distinctions.
- 3-Class Model groups stars into broader categories, so the coefficients should be smoother and more generalized.
- The importance of features remains the same, but the grouped model will show a more interpretable trend (e.g., increasing Temperature consistently favoring Hot stars).

Discussion on Metrics (4 Points)

In the context of classification, performance evaluation hinges on understanding what each metric truly captures.

Accuracy, the most straightforward metric, measures the overall correctness of a model by calculating the proportion of correctly classified instances. While intuitive, its limitation lies in its susceptibility to class imbalance, where a dominant class can inflate accuracy scores, masking poor performance on minority classes.

Precision focuses on the reliability of positive predictions, quantifying the proportion of correctly predicted positives out of all instances predicted as positive. However, it disregards false negatives, meaning a model with high precision might miss many actual positives. Conversely, recall, or sensitivity, measures the model's ability to capture all actual positives, but it overlooks false positives, potentially leading to a model that over-predicts positives.

The F1-score attempts to balance precision and recall, providing a harmonic mean that considers both false positives and false negatives. Yet, as a single metric, it might obscure specific error types. The confusion matrix offers a detailed view of true positives, true negatives, false positives, and false negatives, revealing the model's performance on a class-by-class basis. Its complexity, however, can be challenging in multi-class scenarios. Lastly, the ROC curve and AUC assess the model's ability to discriminate between classes across various thresholds. While robust against class imbalance, they summarize performance, potentially hiding critical threshold-specific behaviors, and their interpretation can be less straightforward than simpler metrics.

ROC Curve visualizes how well our model distinguishes between different star classes (e.g., Hot vs. Cool) by plotting the trade-off between correctly identifying stars of a class (true positive rate) and incorrectly classifying other stars as that class (false positive rate). AUC quantifies the overall ability of our model to separate star classes.

A higher AUC means the model is better at distinguishing between star types based on the given features (temperature, luminosity, radius, absolute magnitude). Essentially, it tells us how likely the model is to rank a randomly chosen star from one class higher than a randomly chosen star from another class. Our 3-class model tends to exhibit higher AUC values because the broader, grouped classes (Hot, Intermediate, Cool) are more easily separable based on the provided features compared to the finer distinctions between the seven individual spectral classes.