In [26]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, classification_report, f1_score, roc
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_pr
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, LogisticRegression
import numpy as np
from sklearn.svm import SVC, LinearSVC
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import loguniform # For sampling C and gamma
```

In [27]:
```
raw_train = pd.read_csv('train.csv')
raw_test = pd.read_csv('test.csv')

raw_train.head()
```

Out[27]:

	obj_ID	alpha	delta	u	g	r	i	
0	1.237661e+18	135.689107	32.494632	23.87882	22.27530	20.39501	19.16573	18.7937
1	1.237665e+18	144.826101	31.274185	24.77759	22.83188	22.58444	21.16812	21.6142
2	1.237661e+18	142.188790	35.582444	25.26307	22.66389	20.60976	19.34857	18.9482
3	1.237663e+18	338.741038	-0.402828	22.13682	23.77656	21.61162	20.50454	19.2501
4	1.237680e+18	345.282593	21.183866	19.43718	17.58028	16.49747	15.97711	15.5446

In [28]:
```
class_encoder = LabelEncoder()


raw_train['class'] = class_encoder.fit_transform(raw_train['class'])
#raw_test['class'] = class_encoder.transform(raw_test['class'])

raw_train.head(2)
```

Out[28]:

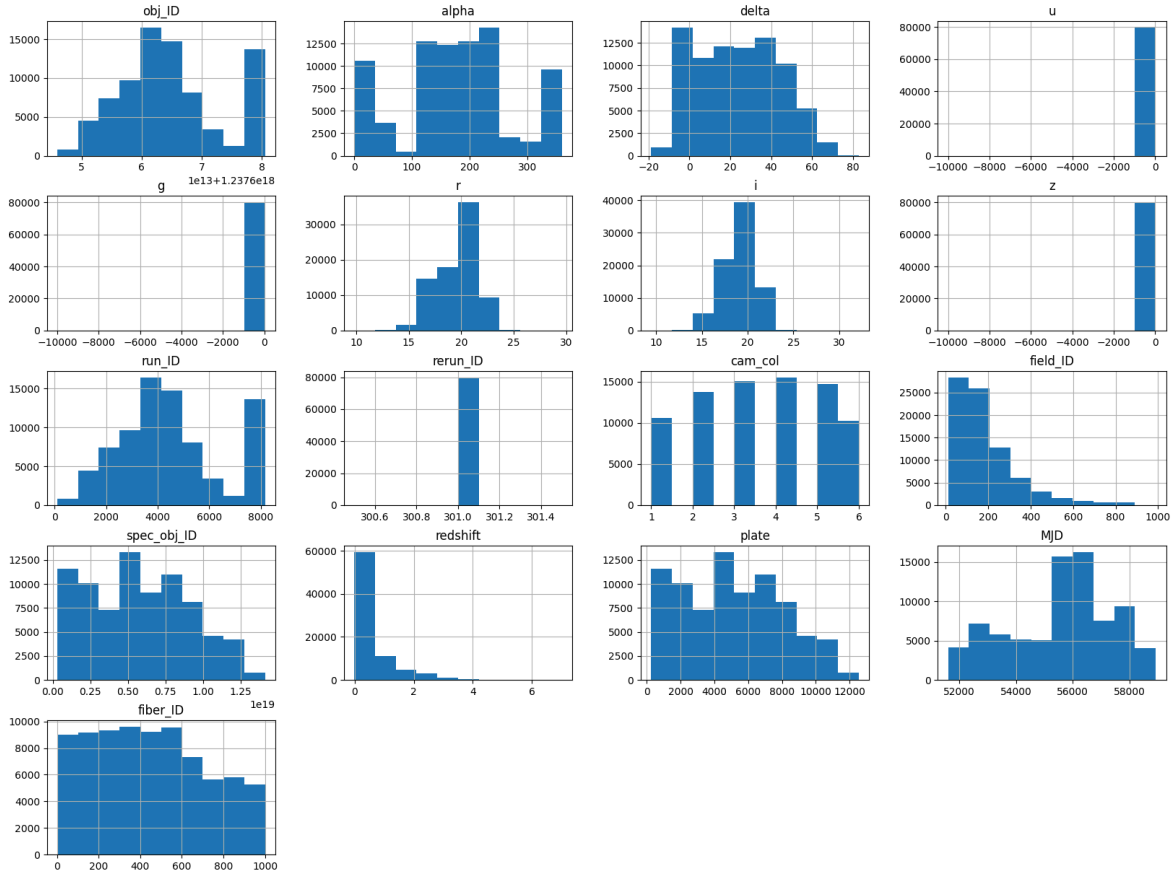
	obj_ID	alpha	delta	u	g	r	i	
0	1.237661e+18	135.689107	32.494632	23.87882	22.27530	20.39501	19.16573	18.7937
1	1.237665e+18	144.826101	31.274185	24.77759	22.83188	22.58444	21.16812	21.6142

In [ ]:
```

```

In [29]:
```
x = raw_train.drop(columns=['class'])
y = raw_train['class']
print(x.shape, y.shape)
```

```
(80000, 17) (80000,)
```

In [30]:
```
x.hist(figsize=(20, 15))
plt.show()
```



In [31]:
```python
# Check for missing values in x
print("Missing values before handling:")
print(x.isnull().sum())

# Fill missing values in 'u' column with its mean
mean_u = x['u'].mean()
x['u'].fillna(mean_u, inplace=True)
raw_test['u'].fillna(mean_u, inplace=True)
# Verify that missing values in x have been handled
print("\nMissing values after handling:")
print(x.isnull().sum())

# y does not have missing values based on the provided info, but adding a check
print("\nMissing values in y:")
print(y.isnull().sum())
```

```
Missing values before handling:
obj_ID           0
alpha            0
delta            0
u              362
g                0
r                0
i                0
z                0
run_ID           0
rerun_ID         0
cam_col          0
field_ID         0
spec_obj_ID      0
redshift         0
plate            0
MJD              0
fiber_ID         0
dtype: int64

Missing values after handling:
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
redshift       0
plate          0
MJD            0
fiber_ID       0
dtype: int64

Missing values in y:
0
```

```
C:\Users\madst\AppData\Local\Temp\ipykernel_20432\152278541.py:7: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  x['u'].fillna(mean_u, inplace=True)
C:\Users\madst\AppData\Local\Temp\ipykernel_20432\152278541.py:8: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  raw_test['u'].fillna(mean_u, inplace=True)
```

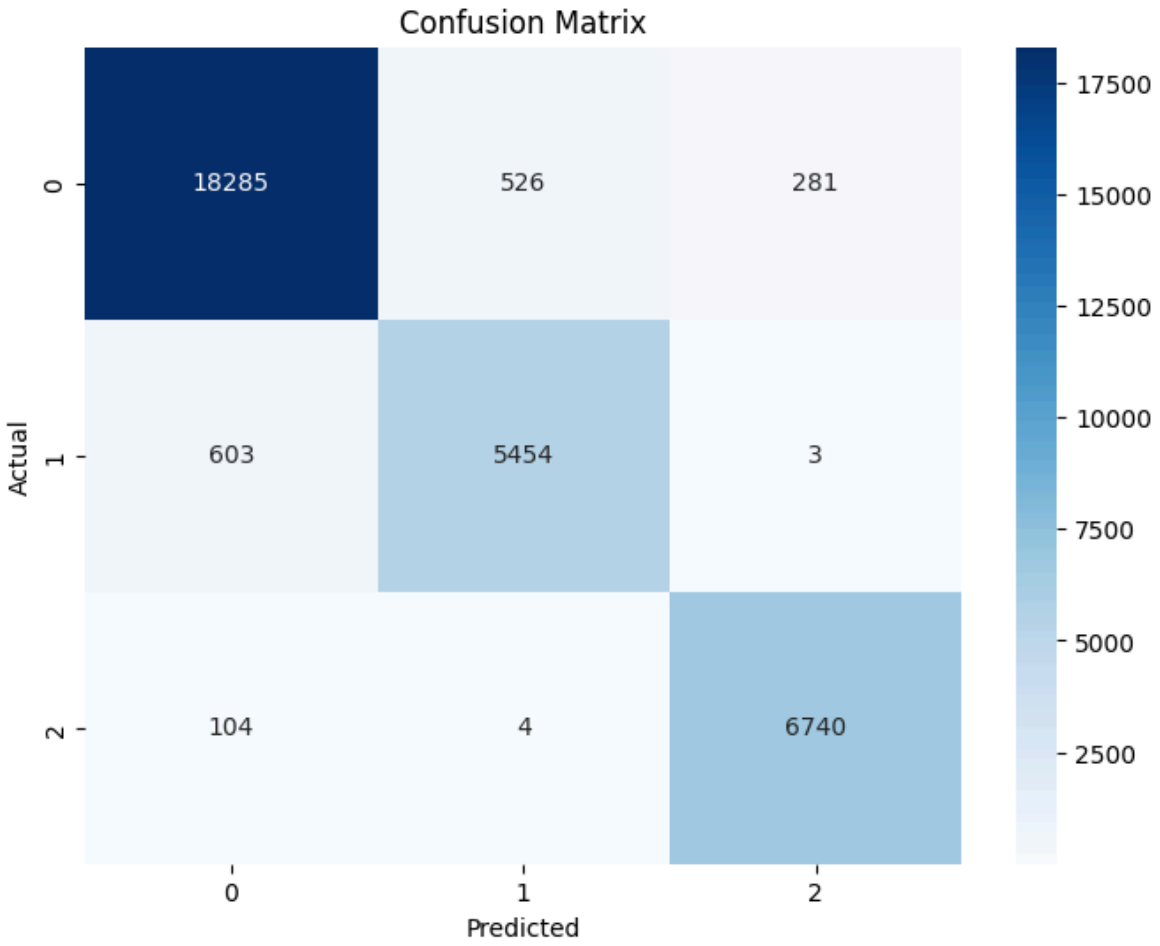
In [ ]:

```python
from sklearn.model_selection import GridSearchCV

# Create a pipeline with StandardScaler, PCA, and SVC. NOTE: this takes a long t
pipeline1 = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=10)),
    ('classifier', SVC(random_state=42, probability=True))
])

# Create a pipeline with StandardScaler, PCA, and Logistic Regression
pipeline2 = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=10)),
    ('classifier', LogisticRegression(penalty='l2', solver='liblinear', max_iter
])

# Define a small parameter grid for grid search
param_grid_lr = {
    'classifier__C': [0.01, 1.0, 100.0]
}

# Create GridSearchCV
grid_search = GridSearchCV(
    pipeline2,
    param_grid=param_grid_lr,
    cv=3,
    scoring='accuracy',
    n_jobs=-1
)

# Fit GridSearchCV to the data
print("Starting Grid Search...")
grid_search.fit(x, y)
```

```
print("Grid Search finished.")

# Print the best parameters found by GridSearchCV
print(f"Best parameters found: {grid_search.best_params_}")

# Print the best cross-validated score (mean accuracy)
print(f"Best cross-validated accuracy: {grid_search.best_score_:.4f}")

# Set the best model for use in subsequent cells
best_model = grid_search.best_estimator_
```

```
Starting Grid Search...
Grid Search finished.
Best parameters found: {'classifier__C': 100.0}
Best cross-validated accuracy: 0.9523
```

In [33]:
```
# Split the data into train and test sets with 60/40 ratio
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_

# Train the best model on the training data
best_model.fit(X_train, y_train)

# Predict the classes on the test data
y_pred = best_model.predict(X_test)

# Create and plot the confusion matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Print classification report for more detailed metrics
print(classification_report(y_test, y_pred))
```

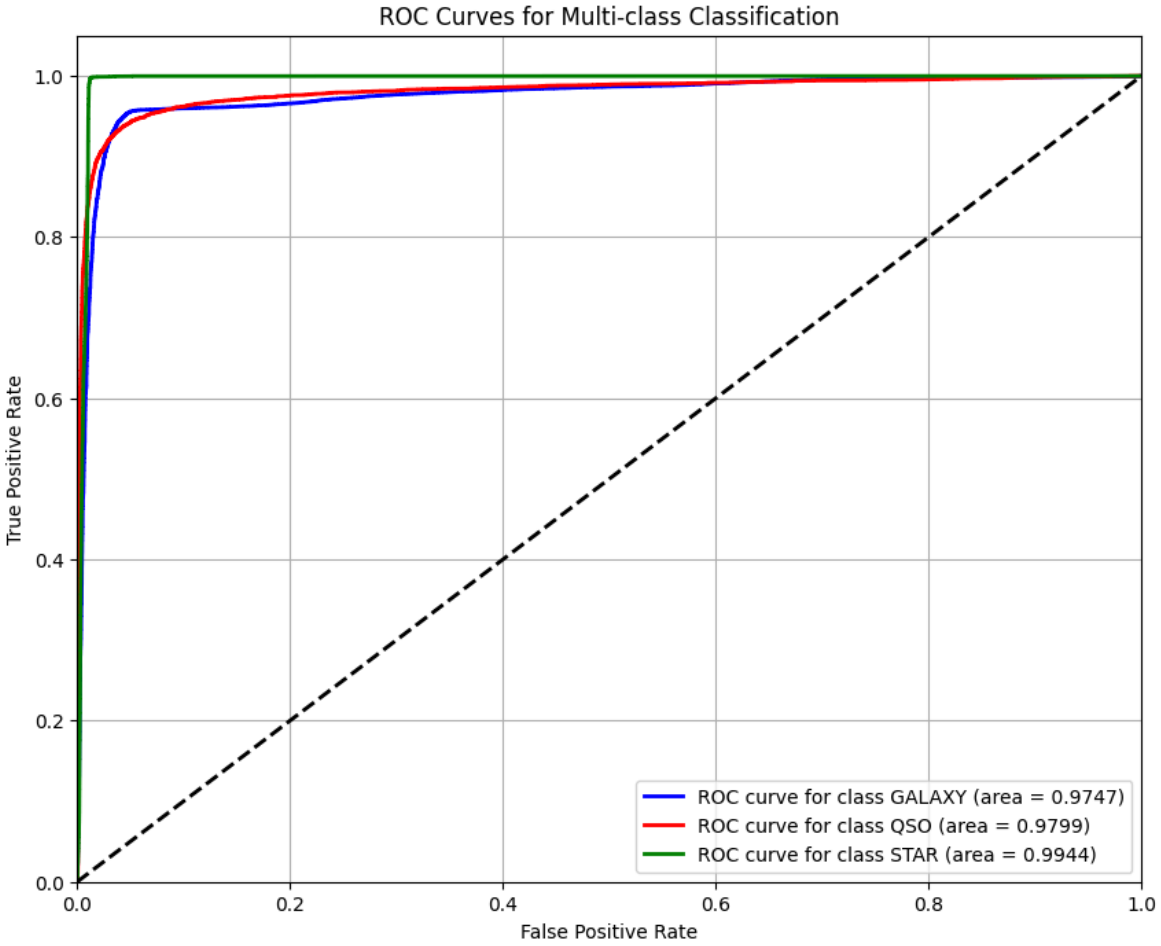


	precision	recall	f1-score	support
0	0.96	0.96	0.96	19092
1	0.91	0.90	0.91	6060
2	0.96	0.98	0.97	6848
accuracy			0.95	32000
macro avg	0.94	0.95	0.95	32000
weighted avg	0.95	0.95	0.95	32000

In [36]:
```
# Calculate and print the macro-averaged F1 score
macro_f1 = f1_score(y_test, y_pred, average='macro')
print(f"Macro-averaged F1 score: {macro_f1:.4f}")

# Calculate and print other evaluation metrics
weighted_f1 = f1_score(y_test, y_pred, average='weighted')
print(f"Weighted F1 score: {weighted_f1:.4f}")

# Generate ROC curve for multi-class classification (one-vs-rest)
y_prob = best_model.predict_proba(X_test)

plt.figure(figsize=(10, 8))
colors = ['blue', 'red', 'green']
class_names = class_encoder.inverse_transform([0, 1, 2])

# Plot ROC curves for each class
for i, color in enumerate(colors):
    fpr, tpr, _ = roc_curve(y_test == i, y_prob[:, i])
    plt.plot(fpr, tpr, color=color, lw=2,
             label=f'ROC curve for class {class_names[i]} (area = {np.trapezoid(
```

```
plt.plot([0, 1], [0, 1], 'k--', lw=2)
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 Curves for Multi-class Classification')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

```
Macro-averaged F1 score: 0.9459
Weighted F1 score: 0.9524
```



In [35]:
```
# Process the test data the same way as training data
# Ensure test features have the same columns as training features
test_features = raw_test[x.columns]

# Make predictions on the test dataset
test_predictions = best_model.predict(test_features)


# Load the sample submission file
sample_submission = pd.read_csv('sample_submission.csv')

# Add the predictions to the appropriate column
sample_submission['class'] = test_predictions

# Save the updated submission file
sample_submission.to_csv('submission.csv', index=False)
```

```
# Display the first few rows of the submission
sample_submission.head()
```

Out[35]:

	ID	class
0	0	0
1	1	2
2	2	2
3	3	2
4	4	2