```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
```

These lines import necessary libraries:

pandas for data manipulation and analysis.

train_test_split from sklearn.model_selection for splitting data into training and test sets.

MLPClassifier from sklearn.neural_network for creating a Multi-Layer Perceptron classifier.

StandardScaler and LabelEncoder from sklearn.preprocessing for scaling numeric features and encoding categorical features, respectively.

accuracy_score and confusion_matrix from sklearn.metrics for evaluating model performance.

matplotlib.pyplot as plt for data visualization.

```
data = pd.read_csv('nasa_asteroid.csv')
```

This line reads the dataset 'nasa_asteroid.csv' into a pandas DataFrame named data.

```
X = data.drop(['Neo Reference ID', 'Name', 'Close Approach Date', 'Epoch Date Close Approach', 'Orbit Determination Date', 'Equinox', 'Hazardous'], axis=1)
y = data['Hazardous']
```

These lines split the data into features (X) and the target variable (y). It drops columns that are deemed non-relevant for prediction and assigns the 'Hazardous' column to y.

```
label_encoder = LabelEncoder()
X['Orbiting Body'] = label_encoder.fit_transform(data['Orbiting Body'])
```

This line encodes the categorical feature 'Orbiting Body' using LabelEncoder, converting it into numeric values for model training.

```
X_train ،X_test ،y_train ،y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

This line splits the data into training and test sets with a 70-30 ratio. The stratify=y parameter ensures that the class distribution is preserved in the split.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

These lines standardize (scale) the numeric features in the training and test sets using StandardScaler.

```
mlp = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=1000, random_state=42)
```
This line initializes an MLPClassifier with a neural network architecture consisting of two hidden layers with 100 and 50 neurons, respectively. max_iter sets the maximum number of iterations for training, and random_state=42 ensures reproducibility.

```
mlp.fit(X_train_scaled, y_train)
```
This line trains the MLPClassifier model using the scaled training data (X_train_scaled and y_train).

```
y_pred = mlp.predict(X_test_scaled)
```
This line generates predictions (y_pred) on the scaled test data (X_test_scaled) using the trained model (mlp).

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Overall Accuracy: {accuracy:.4f}")
```
These lines calculate the accuracy of the model by comparing predicted labels (y_pred) with actual labels (y_test) and print the overall accuracy.

```
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```
These lines compute the confusion matrix to evaluate the performance of the model in terms of true positives, false positives, true negatives, and false negatives.

```
plt.figure(figsize=(8, 6))
plt.plot(mlp.loss_curve_)
plt.title('Loss Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```
This block of code plots the loss curve, showing how the loss (error) changes with each training epoch. It visualizes the training progress and convergence of the model.

```
hidden_layer_sizes = [(50,), (100,), (100, 50), (200, 100)]
results = []
```
These lines define different configurations of hidden layer sizes for hyperparameter tuning and initialize an empty list results to store the tuning results.

```
for hidden_layer in hidden_layer_sizes:
    mlp = MLPClassifier(hidden_layer_sizes=hidden_layer, max_iter=1000, random_state=42)
    mlp.fit(X_train_scaled, y_train)
    train_acc = mlp.score(X_train_scaled, y_train)
    test_acc = mlp.score(X_test_scaled, y_test)
    results.append({'Hidden Layers': hidden_layer, 'Train Accuracy': train_acc, 'Test Accuracy': test_acc})
```

This loop iterates over different hidden layer configurations, trains an MLPClassifier for each configuration, computes and stores the training and test accuracies in the results list along with the corresponding hidden layer configuration.

```
results_df = pd.DataFrame(results)
print("\nHyperparameter Tuning Results:")
print(results_df)
```

These lines create a DataFrame from the hyperparameter tuning results stored in results and print the tuning results, showing the performance of different hidden layer configurations on the training and test sets.

An **overall accuracy of 0.9780** indicates that your MLPClassifier model is performing quite well on the test data. It means that roughly 97.80% of the predictions made by the model align with the actual labels in the test set. This is a high accuracy score and suggests that model is making accurate predictions for classifying hazardous asteroids.

```
Confusion Matrix:

[[1171    9]

 [  22  205]]
```

True Negative (TN): 1171

False Positive (FP): 9

False Negative (FN): 22

True Positive (TP): 205

In the context of asteroid classification:

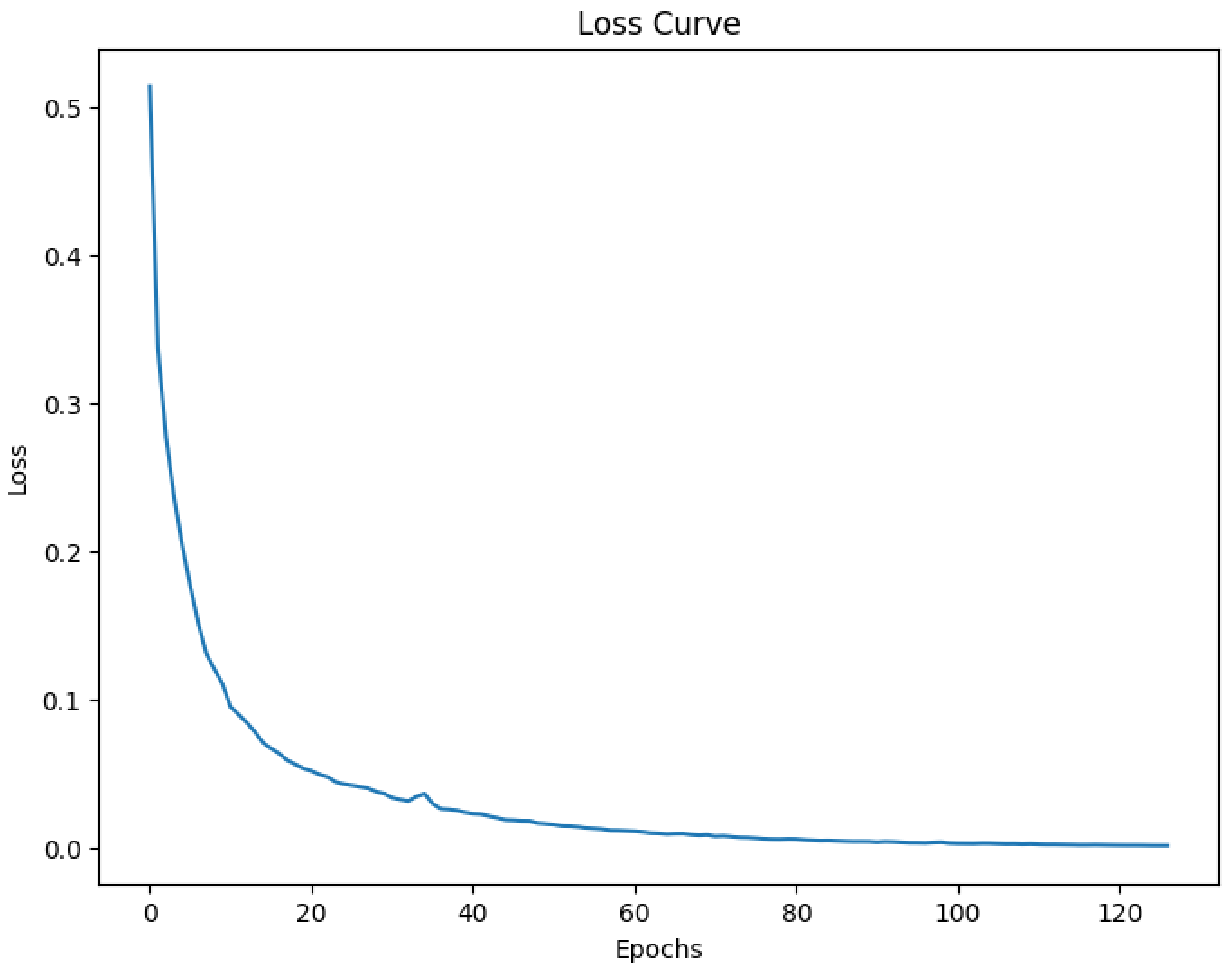
TN (1171) represents the number of correctly predicted non-hazardous asteroids.

FP (9) indicates the number of non-hazardous asteroids that were incorrectly classified as hazardous.

FN (22) represents the number of hazardous asteroids that were incorrectly classified as non-hazardous.

TP (205) indicates the number of correctly predicted hazardous asteroids.

From the confusion matrix, we can see that the model is performing well overall, with a high number of true positives and true negatives.



hyperparameter tuning results show the performance of different configurations of hidden layers in the MLPClassifier model:

For the configuration with a single hidden layer of 50 neurons ((50,)), the model achieved a high training accuracy of approximately 99.97% and a test accuracy of around 98.37%.

With a single hidden layer of 100 neurons ((100,)), the model achieved a perfect training accuracy of 100% but slightly lower test accuracy of about 98.15%.

The configuration with two hidden layers of 100 and 50 neurons ((100, 50)) achieved a perfect training accuracy of 100% but a test accuracy of around 97.80%.

Lastly, the configuration with two hidden layers of 200 and 100 neurons ((200, 100)) also achieved perfect training accuracy of 100% and a test accuracy of approximately 97.87%.