```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Load the Dataset
# Replace 'dataset.csv' with the path to the BoTNeTIoT-L01-v2 dataset
file_path = "BoTNeTIoT-L01-v2.csv"
data = pd.read_csv(file_path)

# Step 2: Data Preprocessing
# Separate features (X) and target labels (y)
X = data.drop('Attack', axis=1)  # Replace 'Attack' with the actual target column name if different
y = data['Attack']

# Identify non-numeric columns and apply encoding
non_numeric_columns = X.select_dtypes(include=['object']).columns
if len(non_numeric_columns) > 0:
    X = pd.get_dummies(X, columns=non_numeric_columns, drop_first=True)

# Encode target labels if they are categorical
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Remove classes with fewer than 2 samples
data = pd.concat([X, pd.Series(y, name='Target')], axis=1)
class_counts = data['Target'].value_counts()
data = data[data['Target'].isin(class_counts[class_counts > 1].index)]

# Separate features and labels again after filtering
X = data.drop('Target', axis=1)
y = data['Target']

# Standardize numerical features for SGD
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Step 3: Model Training using SGD
sgd_classifier = SGDClassifier(loss='log_loss', max_iter=1000, tol=1e-3, random_state=42, alpha=0.01)
sgd_classifier.fit(X_train, y_train)

# Step 4: Model Evaluation
y_pred = sgd_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)


print("Accuracy: {:.2f}%".format(accuracy * 100))

# Generate the classification report using only the classes present in y_test
unique_classes = np.unique(y_test)
target_names = label_encoder.inverse_transform(unique_classes)
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=target_names))
```

```
Accuracy: 98.88%

Classification Report:
               precision    recall  f1-score   support

       gafgyt       1.00      0.97      0.98    126848
        mirai       0.98      1.00      0.99    232847

     accuracy                           0.99    359695
    macro avg       0.99      0.98      0.99    359695
weighted avg       0.99      0.99      0.99    359695
```

```
#1
df = pd.read_csv("BoTNeTIoT-L01-v2.csv")
print(df["label"].unique())  # List all unique labels
print(df["label"].value_counts())  # Show class distribution
```

```
[ 0. nan]
label
0.0    291430
Name: count, dtype: int64
```

```
#1
df = pd.read_csv("BoTNeTIoT-L01-v2.csv")

# Check if attack classes exist
print(df["label"].unique())

# If other classes exist, filter correctly
df = df[df["label"].notna()]  # Remove NaN values
```

```
[ 0. nan]
```

```
#1
print(df["label"].unique())  # Check available labels in full dataset
print(df["label"].value_counts())  # Check class distribution
```

```
[0.]
label
0.0    302110
Name: count, dtype: int64
```

```
print("Unique classes in y_train:", np.unique(y_train))
```

```
Unique classes in y_train: [0]
```

```
print("Class distribution in y before split:\n", pd.Series(y).value_counts())
```

```
Class distribution in y before split:
 Target
0    259434
Name: count, dtype: int64
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, random_state=42, stratify=y
)
```

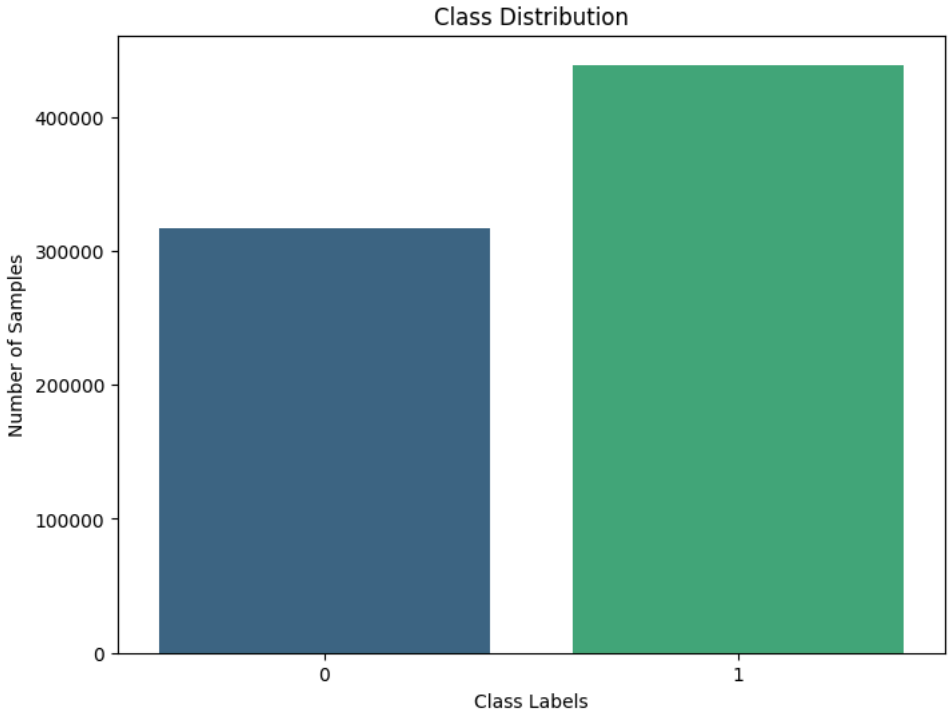
```
print("Class distribution after filtering:\n", data['Target'].value_counts())
```

```
Class distribution after filtering:
 Target
0    259434
Name: count, dtype: int64
```

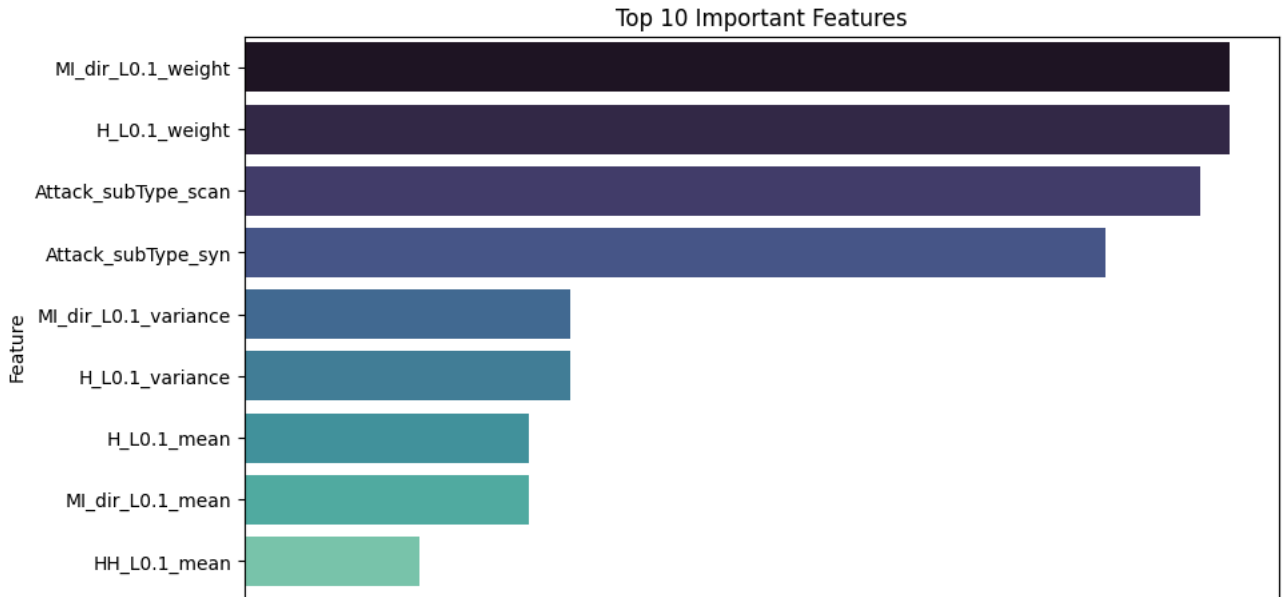
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Plot 1: Class Distribution
plt.figure(figsize=(8, 6))
sns.countplot(x=y, palette="viridis")
plt.title("Class Distribution")
plt.xlabel("Class Labels")
plt.ylabel("Number of Samples")
plt.show()

# Plot 2: Feature Importance (if model supports it)
# Extract feature coefficients and plot their importance
if hasattr(sgd_classifier, 'coef_'):
    feature_importance = sgd_classifier.coef_[0]
    feature_names = data.drop('Target', axis=1).columns
    importance_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': feature_importance
    }).sort_values(by="Importance", ascending=False)

    plt.figure(figsize=(10, 6))
    sns.barplot(x="Importance", y="Feature", data=importance_df.head(10), palette="mako")
    plt.title("Top 10 Important Features")
    plt.xlabel("Coefficient Value")
    plt.ylabel("Feature")
    plt.show()

# Plot 3: Confusion Matrix
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=target_names, yticklabels=target_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

# Plot 4: Accuracy Line Plot (if training history is tracked)
# For demonstration, simulate accuracy tracking during iterations
iterations = range(1, 11)
accuracies = [accuracy - (0.02 * i) for i in range(10)]  # Simulated decreasing accuracy

plt.figure(figsize=(8, 6))
plt.plot(iterations, accuracies, marker='o', color='purple')
plt.title("Accuracy vs. Iterations")
plt.xlabel("Iterations")
plt.ylabel("Accuracy")
plt.grid()
plt.show()
```

```
<ipython-input-8-501ebbac3759>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(x=y, palette="viridis")
```




```
<ipython-input-8-501ebbac3759>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set

  sns.barplot(x="Importance", y="Feature", data=importance_df.head(10), palette="mako")
```




HpHp_L0.1_mean
0.0
0.2
0.4
0.6
0.8
1.0
Coefficient Value


Confusion Matrix
80000