```
import pandas as pd

df = pd.read_csv('/content/Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv')

# Display the first few rows to understand data
print(df.head())

# General info about the dataset
print(df.info())
```

```
    Destination Port   Flow Duration   Total Fwd Packets  \
0              54865               3                   2
1              55054             109                   1
2              55055              52                   1
3              46236              34                   1
4              54863               3                   2

    Total Backward Packets  Total Length of Fwd Packets  \
0                        0                           12
1                        1                            6
2                        1                            6
3                        1                            6
4                        0                           12

    Total Length of Bwd Packets   Fwd Packet Length Max  \
0                             0                       6
1                             6                       6
2                             6                       6
3                             6                       6
4                             0                       6

    Fwd Packet Length Min   Fwd Packet Length Mean   Fwd Packet Length Std  \
0                       6                      6.0                     0.0
1                       6                      6.0                     0.0
2                       6                      6.0                     0.0
3                       6                      6.0                     0.0
4                       6                      6.0                     0.0

   ...   min_seg_size_forward  Active Mean   Active Std   Active Max  \
0  ...                    20          0.0          0.0            0
1  ...                    20          0.0          0.0            0
2  ...                    20          0.0          0.0            0
3  ...                    20          0.0          0.0            0
4  ...                    20          0.0          0.0            0

    Active Min  Idle Mean   Idle Std   Idle Max   Idle Min   Label
0            0        0.0        0.0          0          0  BENIGN
1            0        0.0        0.0          0          0  BENIGN
2            0        0.0        0.0          0          0  BENIGN
3            0        0.0        0.0          0          0  BENIGN
4            0        0.0        0.0          0          0  BENIGN

[5 rows x 79 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 225745 entries, 0 to 225744
Data columns (total 79 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0    Destination Port             225745 non-null  int64
 1    Flow Duration                225745 non-null  int64
 2    Total Fwd Packets            225745 non-null  int64
 3    Total Backward Packets       225745 non-null  int64
 4   Total Length of Fwd Packets   225745 non-null  int64
 5    Total Length of Bwd Packets  225745 non-null  int64
 6    Fwd Packet Length Max        225745 non-null  int64
 7    Fwd Packet Length Min        225745 non-null  int64
 8    Fwd Packet Length Mean       225745 non-null  float64
 9    Fwd Packet Length Std        225745 non-null  float64
```

```
df.describe()
```

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Tc Length Bwd Pack
count	225745.00000	2.257450e+05	225745.000000	225745.000000	225745.000000	2.257450e
mean	8879.61946	1.624165e+07	4.874916	4.572775	939.463346	5.960477e
std	19754.64740	3.152437e+07	15.422874	21.755356	3249.403484	3.921834e
min	0.00000	-1.000000e+00	1.000000	0.000000	0.000000	0.000000e
25%	80.00000	7.118000e+04	2.000000	1.000000	26.000000	0.000000e
50%	80.00000	1.452333e+06	3.000000	4.000000	30.000000	1.640000e
75%	80.00000	8.805237e+06	5.000000	5.000000	63.000000	1.160100e
max	65532.00000	1.199999e+08	1932.000000	2942.000000	183012.000000	5.172346e

8 rows × 78 columns

```
print (df.columns)
```

```
Index([' Destination Port', ' Flow Duration', ' Total Fwd Packets',
       ' Total Backward Packets', 'Total Length of Fwd Packets',
       ' Total Length of Bwd Packets', ' Fwd Packet Length Max',
       ' Fwd Packet Length Min', ' Fwd Packet Length Mean',
       ' Fwd Packet Length Std', 'Bwd Packet Length Max',
       ' Bwd Packet Length Min', ' Bwd Packet Length Mean',
       ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s',
       ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min',
       'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max',
       ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std',
       ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Bwd PSH Flags',
       ' Fwd URG Flags', ' Bwd URG Flags', ' Fwd Header Length',
       ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s',
       ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean',
       ' Packet Length Std', ' Packet Length Variance', 'FIN Flag Count',
       ' SYN Flag Count', ' RST Flag Count', ' PSH Flag Count',
       ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count',
       ' ECE Flag Count', ' Down/Up Ratio', ' Average Packet Size',
       ' Avg Fwd Segment Size', ' Avg Bwd Segment Size',
       ' Fwd Header Length.1', 'Fwd Avg Bytes/Bulk', ' Fwd Avg Packets/Bulk',
       ' Fwd Avg Bulk Rate', ' Bwd Avg Bytes/Bulk', ' Bwd Avg Packets/Bulk',
       'Bwd Avg Bulk Rate', 'Subflow Fwd Packets', ' Subflow Fwd Bytes',
       ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward',
       ' Init_Win_bytes_backward', ' act_data_pkt_fwd',
       ' min_seg_size_forward', 'Active Mean', ' Active Std', ' Active Max',
       ' Active Min', 'Idle Mean', ' Idle Std', ' Idle Max', ' Idle Min',
       ' Label'],
      dtype='object')
```

```
df[' Label'].value_counts()
```

```
DDoS      128027
BENIGN     97718
Name:  Label, dtype: int64
```

```
correlation_matrix = df.corr()
```

```
<ipython-input-5-68bbfff3c4eb>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
  correlation_matrix = df.corr()
```

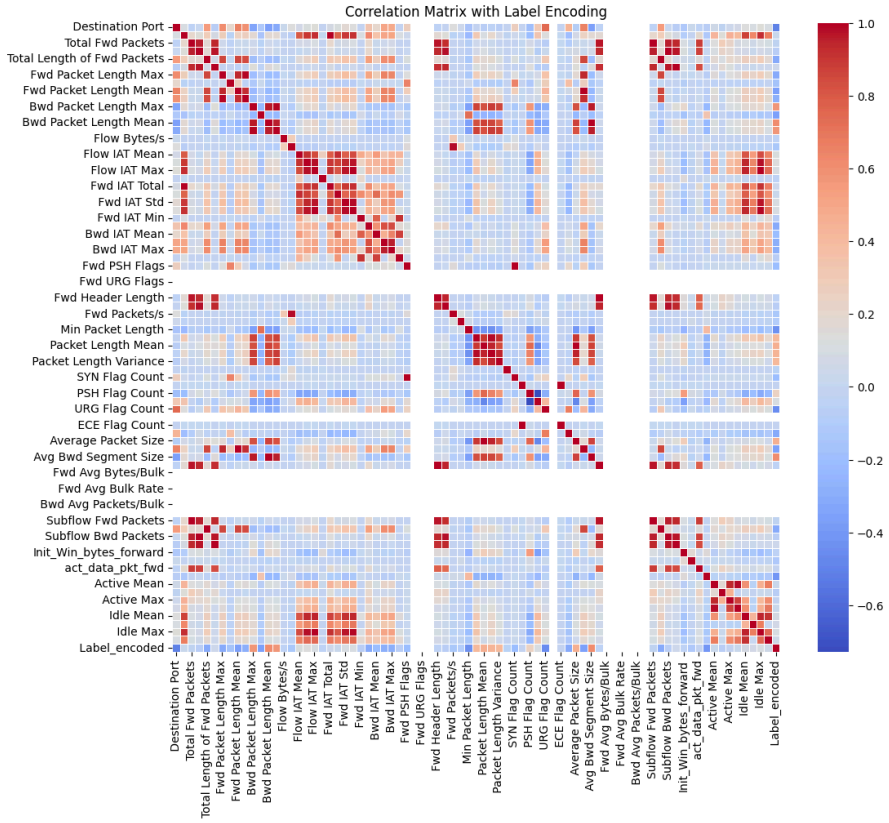
```
from sklearn.preprocessing import LabelEncoder

# Create a label encoder object
le = LabelEncoder()

# Fit and transform the 'Label' column
df[' Label_encoded'] = le.fit_transform(df[' Label'])

# Check the encoding
print(df[[' Label', ' Label_encoded']].head())
```

```
    Label   Label_encoded
0  BENIGN               0
1  BENIGN               0
2  BENIGN               0
3  BENIGN               0
4  BENIGN               0
```

```
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Isolate the correlations with 'Label_encoded'
label_correlations = correlation_matrix[' Label_encoded'].sort_values(ascending=False)

# Print the correlations with 'Label_encoded'
print(label_correlations)
```

```
<ipython-input-7-11937815093a>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
  correlation_matrix = df.corr()
 Label_encoded             1.000000
 Bwd Packet Length Mean    0.603299
 Avg Bwd Segment Size      0.603299
Bwd Packet Length Max      0.577323
 Bwd Packet Length Std     0.576155
                             ...
 Fwd Avg Packets/Bulk          NaN
 Fwd Avg Bulk Rate             NaN
 Bwd Avg Bytes/Bulk            NaN
 Bwd Avg Packets/Bulk          NaN
Bwd Avg Bulk Rate              NaN
Name:  Label_encoded, Length: 79, dtype: float64
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set the size of the figure
plt.figure(figsize=(12, 10))

# Generate a heatmap
sns.heatmap(correlation_matrix, cmap='coolwarm', linewidths=.5)

# Show the plot
plt.title('Correlation Matrix with Label Encoding')
plt.show()
```




```
# Assuming 'correlation_matrix' is already defined and includes 'Label_encoded'

# Get correlations with 'Label_encoded', sorted by absolute value in descending order, excluding 'Label_encoded' itself
sorted_correlations = correlation_matrix[' Label_encoded'].drop(' Label_encoded').abs().sort_values(ascending=False)

# Determine a threshold for selecting features. For example, you might start with 0.5.
threshold = 0.5

# Select features that have a correlation above this threshold
selected_features_above_threshold = sorted_correlations[sorted_correlations > threshold].index.tolist()

# Output the selected feature names
print("Features selected based on correlation with ' Label_encoded':")
print(selected_features_above_threshold)
```

```
Features selected based on correlation with ' Label_encoded':
[' Bwd Packet Length Mean', ' Avg Bwd Segment Size', 'Bwd Packet Length Max', ' Bwd Packet Length Std', ' Destination Port']
```

```
df[' Bwd Packet Length Mean']
```

```
0         0.0
1         6.0
2         6.0
3         6.0
4         0.0
         ...
225740    6.0
225741    6.0
225742    6.0
225743    0.0
225744    6.0
Name:  Bwd Packet Length Mean, Length: 225745, dtype: float64
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler

# Assume you have already created a dataframe 'df' with the necessary preprocessing
X = df[[' Bwd Packet Length Mean', ' Avg Bwd Segment Size', 'Bwd Packet Length Max', ' Bwd Packet Length Std', ' Destination Port']]
y = df[' Label_encoded']

# 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)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = rf_classifier.predict(X_test_scaled)

# Evaluate the classifier
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       1.00      0.96      0.98     19405
               1       0.97      1.00      0.98     25744

        accuracy                           0.98     45149
       macro avg       0.98      0.98      0.98     45149
    weighted avg       0.98      0.98      0.98     45149

    Confusion Matrix:
    [[18618   787]
     [   15 25729]]
    Accuracy Score: 0.9822365943874726
```

```
from sklearn.model_selection import cross_val_score
import numpy as np

# Make sure to import the classifier you are using
from sklearn.ensemble import RandomForestClassifier

# Initialize the classifier, assuming you have already done so
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(rf_classifier, X_scaled, y, cv=5)  # Use the correct variable for scaled features

# Print the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV Score:", np.mean(cv_scores))
```

```
Cross-validation scores: [0.97740814 0.9869986  0.99127334 0.98349908 0.97001041]
Mean CV Score: 0.981837914461007
```

```
# Assuming X_train_scaled and y_train are correctly defined and contain your training data:

# Initialize the RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the classifier with the training data
rf_classifier.fit(X_train_scaled, y_train)  # This step is crucial

# Now that the classifier has been fitted, you can access feature_importances_
feature_importances = rf_classifier.feature_importances_

# Create a DataFrame to display feature importance
features_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

# Sort the features by importance
features_df.sort_values(by='Importance', ascending=False, inplace=True)

# Display the feature importances
print(features_df)
```

```
                     Feature  Importance
4         Destination Port    0.366987
1     Avg Bwd Segment Size    0.190029
0   Bwd Packet Length Mean    0.181196
3    Bwd Packet Length Std    0.164499
2    Bwd Packet Length Max    0.097288
```

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    # Add other parameters you wish to tune
}

# Initialize the grid search
grid_search = GridSearchCV(rf_classifier, param_grid, cv=5)

# Perform the grid search on the scaled data
grid_search.fit(X_train_scaled, y_train)
print("Best parameters:", grid_search.best_params_)
```

```
Best parameters: {'max_depth': None, 'n_estimators': 200}
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

```
# Initialize the classifier
rf_classifier = RandomForestClassifier(random_state=42)
```

```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
```

```
grid_search.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
```

```
# Get the best parameters
print("Best parameters found: ", grid_search.best_params_)
```

```
Best parameters found:  {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}
```

```
# Retrieve the best model from the grid search
best_rf_classifier = grid_search.best_estimator_

# Predict on the test set
y_pred_final = best_rf_classifier.predict(X_test_scaled)

# Evaluate the final model
print(classification_report(y_test, y_pred_final))
print("Final model accuracy:", accuracy_score(y_test, y_pred_final))
```

```
                  precision    recall  f1-score   support

               0       1.00      0.96      0.98     19405
               1       0.97      1.00      0.98     25744

        accuracy                           0.98     45149
       macro avg       0.98      0.98      0.98     45149
    weighted avg       0.98      0.98      0.98     45149

    Final model accuracy: 0.9822587432722762
```

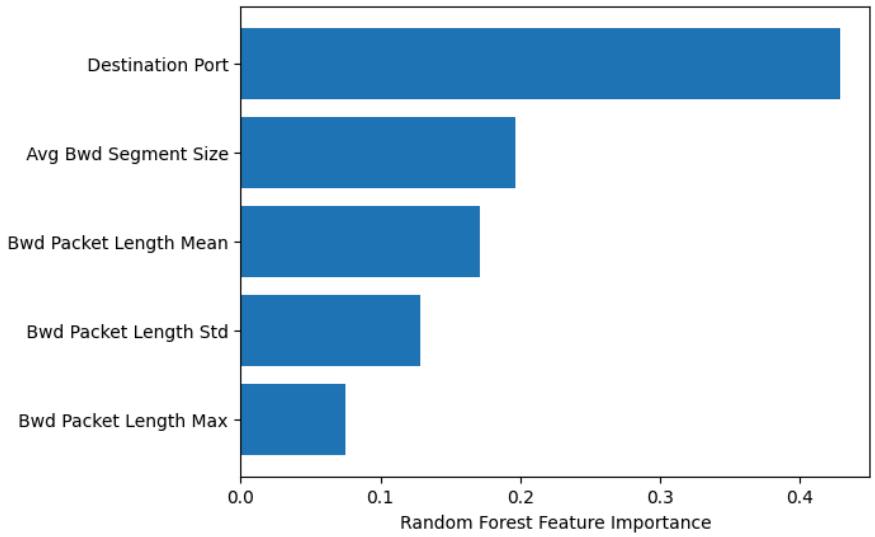
```
from sklearn.metrics import (
    classification_report,
    roc_auc_score,
    matthews_corrcoef,
    f1_score,
    precision_score,
    recall_score,
    fbeta_score,
    accuracy_score
)

def evaluate_model(y_true, y_pred, y_score=None):
    # Basic classification report
    print(classification_report(y_true, y_pred))

    # F1 Score
    print(f"F1 Score: {f1_score(y_true, y_pred)}")

    # F2 Score
    print(f"F2 Score: {fbeta_score(y_true, y_pred, beta=2)}")

    # Precision
    print(f"Precision: {precision_score(y_true, y_pred)}")

    # Recall
    print(f"Recall: {recall_score(y_true, y_pred)}")

    # ROC-AUC Score if the prediction scores are provided
    if y_score is not None:
        print(f"ROC-AUC Score: {roc_auc_score(y_true, y_score)}")

    # Matthews correlation coefficient
    print(f"Matthews correlation coefficient: {matthews_corrcoef(y_true, y_pred)}")

# Call the function with the true labels and predicted labels
# Note: If you have prediction probabilities, you can also pass them as y_score for ROC-AUC
evaluate_model(y_test, y_pred_final)
```

```
                  precision    recall  f1-score   support

               0       1.00      0.96      0.98     19405
               1       0.97      1.00      0.98     25744

        accuracy                           0.98     45149
       macro avg       0.98      0.98      0.98     45149
    weighted avg       0.98      0.98      0.98     45149

    F1 Score: 0.9846724966034559
    F2 Score: 0.9934667274173496
    Precision: 0.970356402036583
```

```
Recall: 0.9994173399627098
Matthews correlation coefficient: 0.9642171949039587
```

```
import joblib
from sklearn.inspection import permutation_importance

# Save the model to disk
joblib.dump(best_rf_classifier, 'best_random_forest_model.joblib')

# Load the model from disk (for future use)
# loaded_rf_classifier = joblib.load('best_random_forest_model.joblib')

# Optionally, get and plot feature importances
feature_importance = best_rf_classifier.feature_importances_
sorted_idx = feature_importance.argsort()
```

```
import matplotlib.pyplot as plt

plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), [X.columns[i] for i in sorted_idx])
plt.xlabel("Random Forest Feature Importance")
plt.show()
```




```
# Further analysis: Permutation importance
perm_importance = permutation_importance(best_rf_classifier, X_test_scaled, y_test)

sorted_perm_idx = perm_importance.importances_mean.argsort()

plt.barh(range(len(sorted_perm_idx)), perm_importance.importances_mean[sorted_perm_idx], align='center')
plt.yticks(range(len(sorted_perm_idx)), [X_test.columns[i] for i in sorted_perm_idx])
plt.xlabel("Permutation Importance")
plt.show()
```