Machine-Learning-Based Network security management system

- This code showcases how IDS can be designed and intergrated in computer systems.
- THe code is based on CICIDS2017 dataset and it serves a purpose as a final Year project.

Get all necessary libraries

In [1]:
```python
import os
import re
import shutil
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from itertools import cycle
from sklearn.preprocessing import label_binarize
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import (
    classification_report,confusion_matrix,accuracy_score,precision_recall_fscore_support, f1_score,
    recall_score, precision_recall_curve, precision_score, roc_curve,auc
)

from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from  xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from lightgbm import LGBMClassifier
from sklearn.decomposition import PCA

#for features selection
from sklearn.feature_selection import SelectKBest,chi2
from sklearn.feature_selection import f_classif

# for getting feature importances
from xgboost import plot_importance


import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
# some settings
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

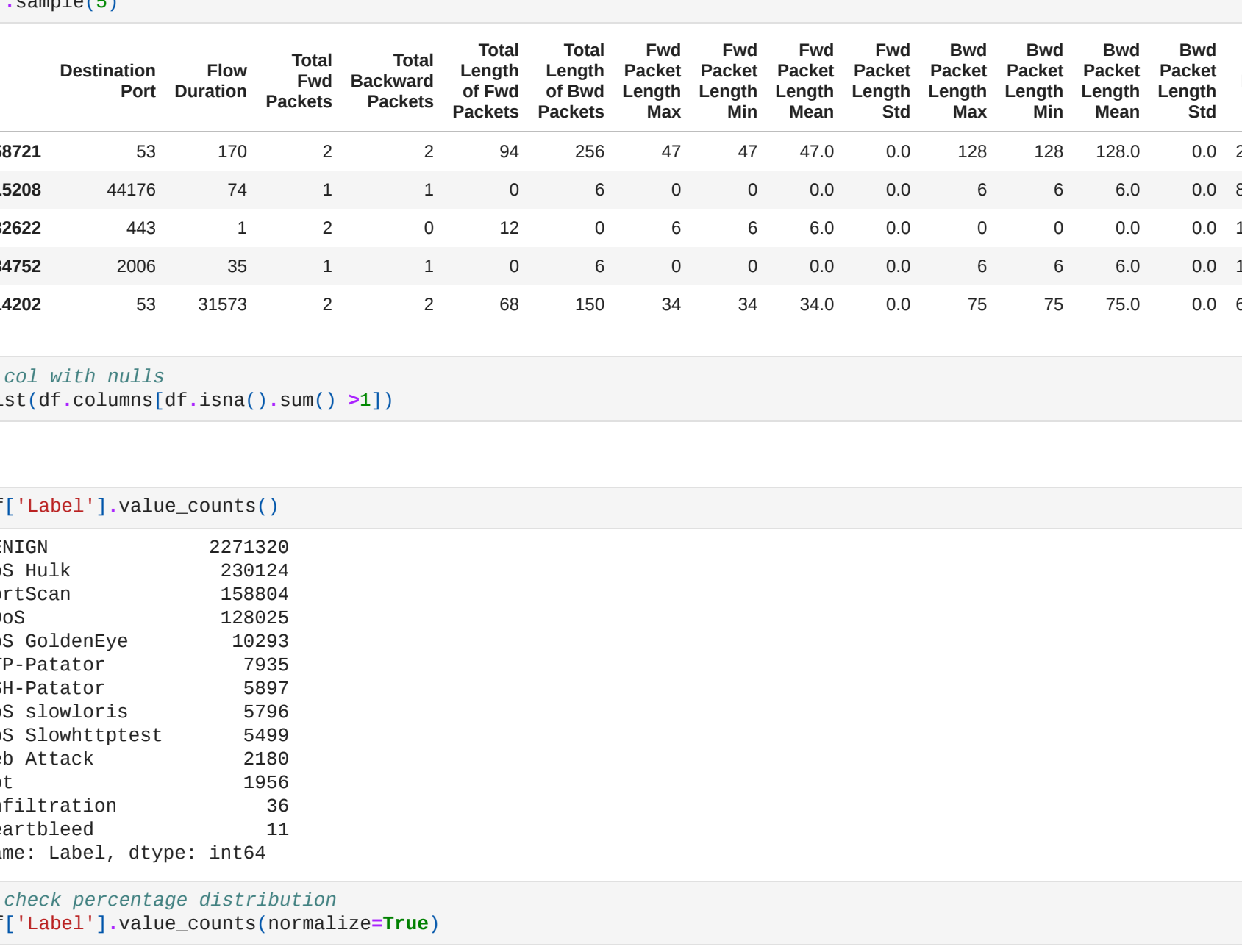
The CICIDS2017 dataset

- The CICIDS2017 dataset can be found in kaggle through https://www.kaggle.com/datasets/cicdataset/cicids2017 or pulicly at: https://www.unb.ca/cic/datasets/ids-2017.html

`COPIED FROM DESCRIPTION`.

```
About Dataset
Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs) are the most
important defense tools against the sophisticated and ever-growing network attacks. Due to the
lack of reliable test and validation datasets, anomaly-based intrusion detection approaches
are suffering from consistent and accurate performance evolutions.

Our evaluations of the existing eleven datasets since 1998 show that most are out of date and
unreliable. Some of these datasets suffer from the lack of traffic diversity and volumes, some
do not cover the variety of known attacks, while others anonymize packet payload data, which
cannot reflect the current trends. Some are also lacking feature set and metadata.

CICIDS2017 dataset contains benign and the most up-to-date common attacks, which resembles the
true real-world data (PCAPs). It also includes the results of the network traffic analysis
using CICFlowMeter with labeled flows based on the time stamp, source, and destination IPs,
source and destination ports, protocols and attack (CSV files). Also available is the
extracted features definition.

Generating realistic background traffic was our top priority in building this dataset. We have
used our proposed B-Profile system (Sharafaldin, et al. 2016) to profile the abstract behavior
of human interactions and generates naturalistic benign background traffic. For this dataset,
we built the abstract behavior of 25 users based on the HTTP, HTTPS, FTP, SSH, and email
protocols.

The data capturing period started at 9 a.m., Monday, July 3, 2017, and ended at 5 p.m. on
Friday, July 7, 2017, for a total of 5 days. Monday is the normal day and only includes benign
```

Data Selection

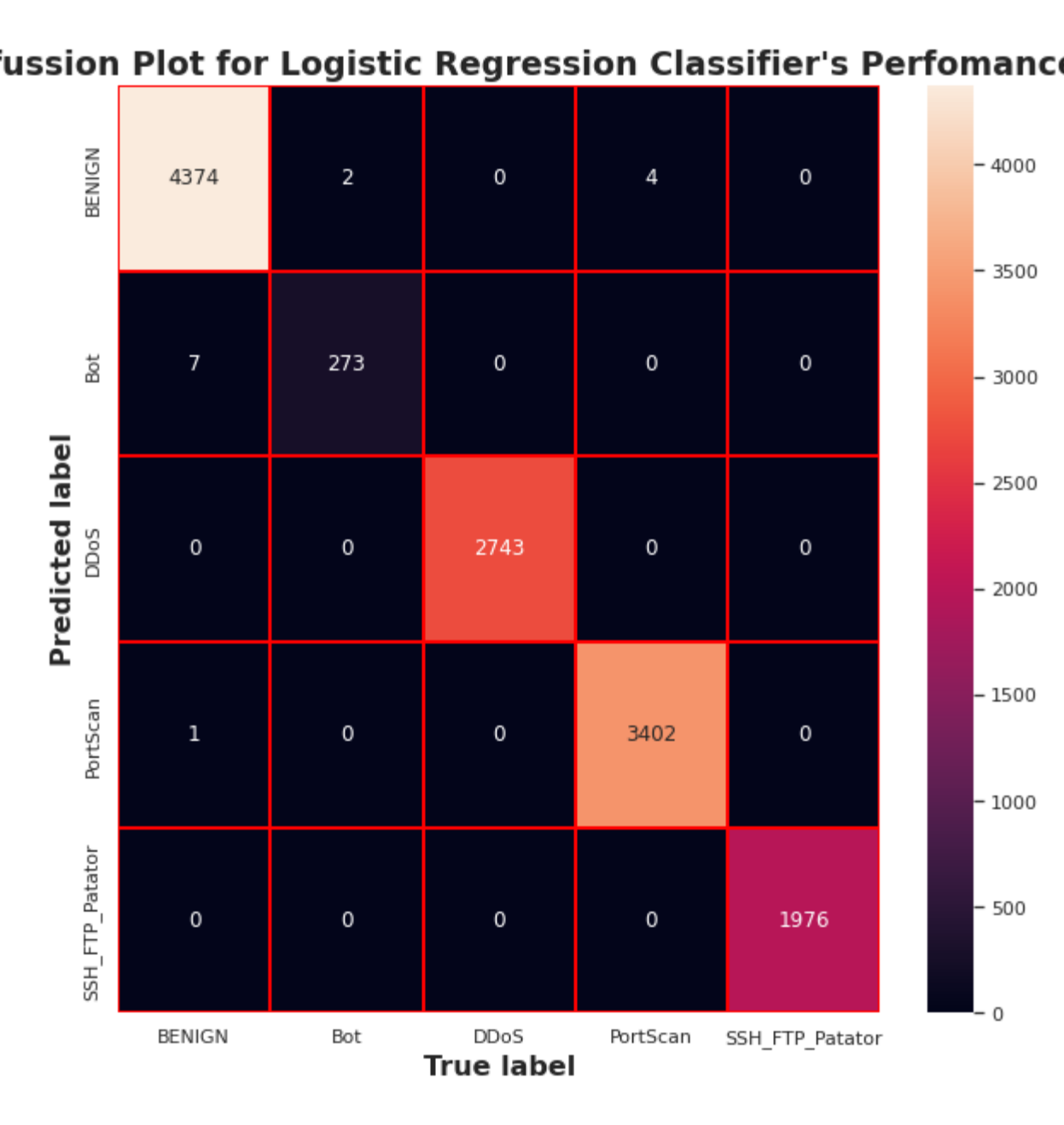


Target Scoring Distribution

Receiver operating characteristic for Logistic Regression Classifier

```
Logistic Regression Classifier Metrics Analysis Results

        Accuracy is   0.9989047097480832
        F1 Score is   0.9989017492373268
        Recall Score is   0.9989047097480832
        Precision Score is   0.9989024261350944

        Outputting Classification Report Matrix Plot

         CLASSIFICATION REPORT FOR THE LOGISTIC REGRESSION CLASSIFIER
                 precision    recall  f1-score   support

          BENIGN       1.00      1.00      1.00      4380
             Bot       0.99      0.97      0.98       280
            DDoS       1.00      1.00      1.00      2743
        PortScan       1.00      1.00      1.00      3403
SSH_FTP_Patator       1.00      1.00      1.00      1976

        accuracy                           1.00     12782
       macro avg       1.00      0.99      1.00     12782
    weighted avg       1.00      1.00      1.00     12782
```

Confussion Plot for Logistic Regression Classifier's Perfomance



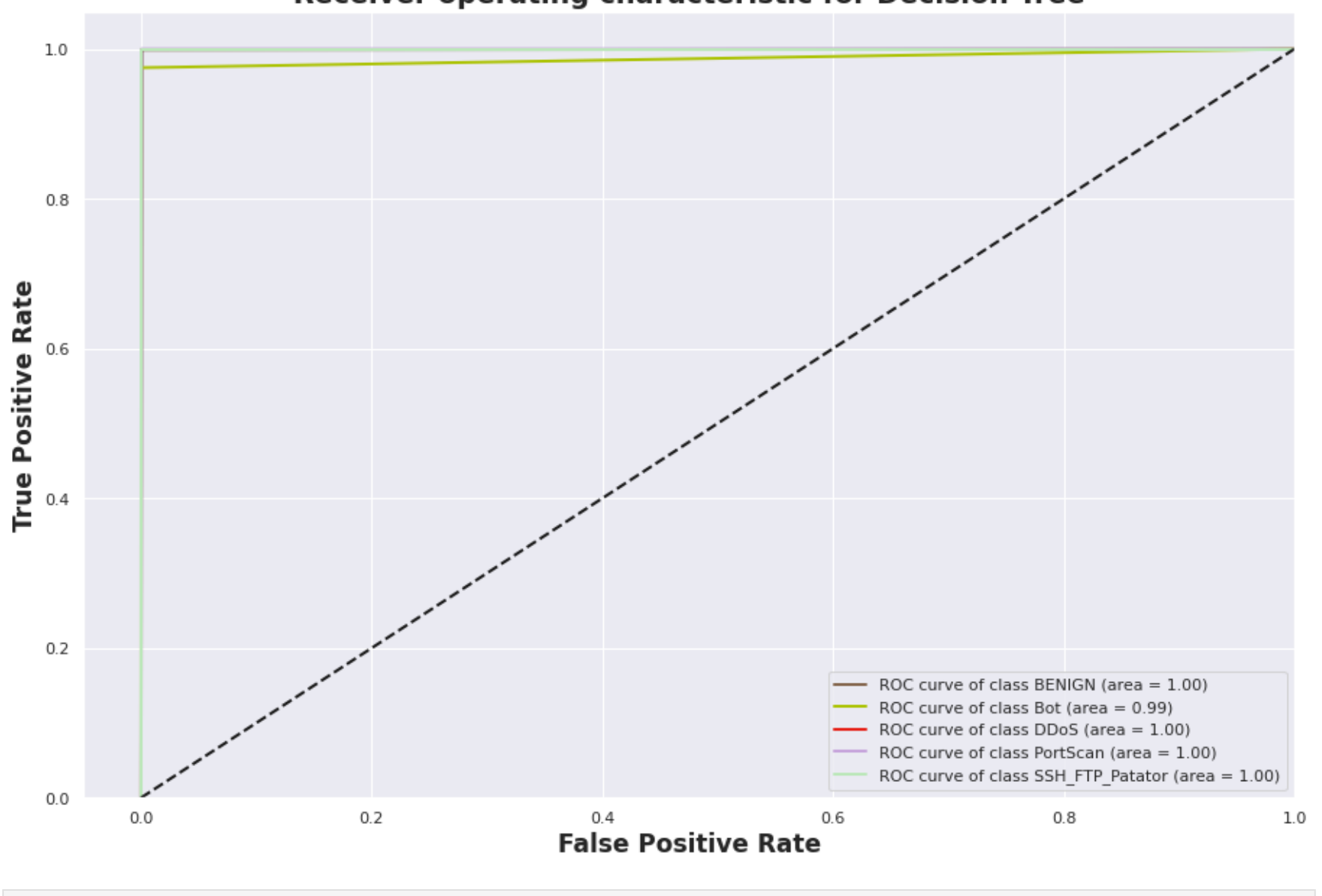
2. Decision Tree Classifier.

In [54]:
```python
# Decision tree training and prediction
decision_model = DecisionTreeClassifier(random_state = 0)
# train the model
decision_model.fit(Xtrain,Ytrain)


# get model score and predictions
dt_score=decision_model.score(Xtest,Ytest)
decision_pred=decision_model.predict(Xtest)
print('Accuracy of DT: '+ str(dt_score))
precision,recall,fscore,none= precision_recall_fscore_support(Ytest, decision_pred, average='weighted')
print('Precision of DT: '+(str(precision)))
print('Recall of DT: '+(str(recall)))
print('F1-score of DT: '+(str(fscore)))
```

```
Accuracy of DT: 0.9989047097480832
Precision of DT: 0.9989024261350944
Recall of DT: 0.9989047097480832
F1-score of DT: 0.9989017492373268
```

In [55]:
```python
# get the analysis of roc auc curve
draw_roc_auc_curve(decision_model, "Decision Tree", Ytest, Xtest)
```

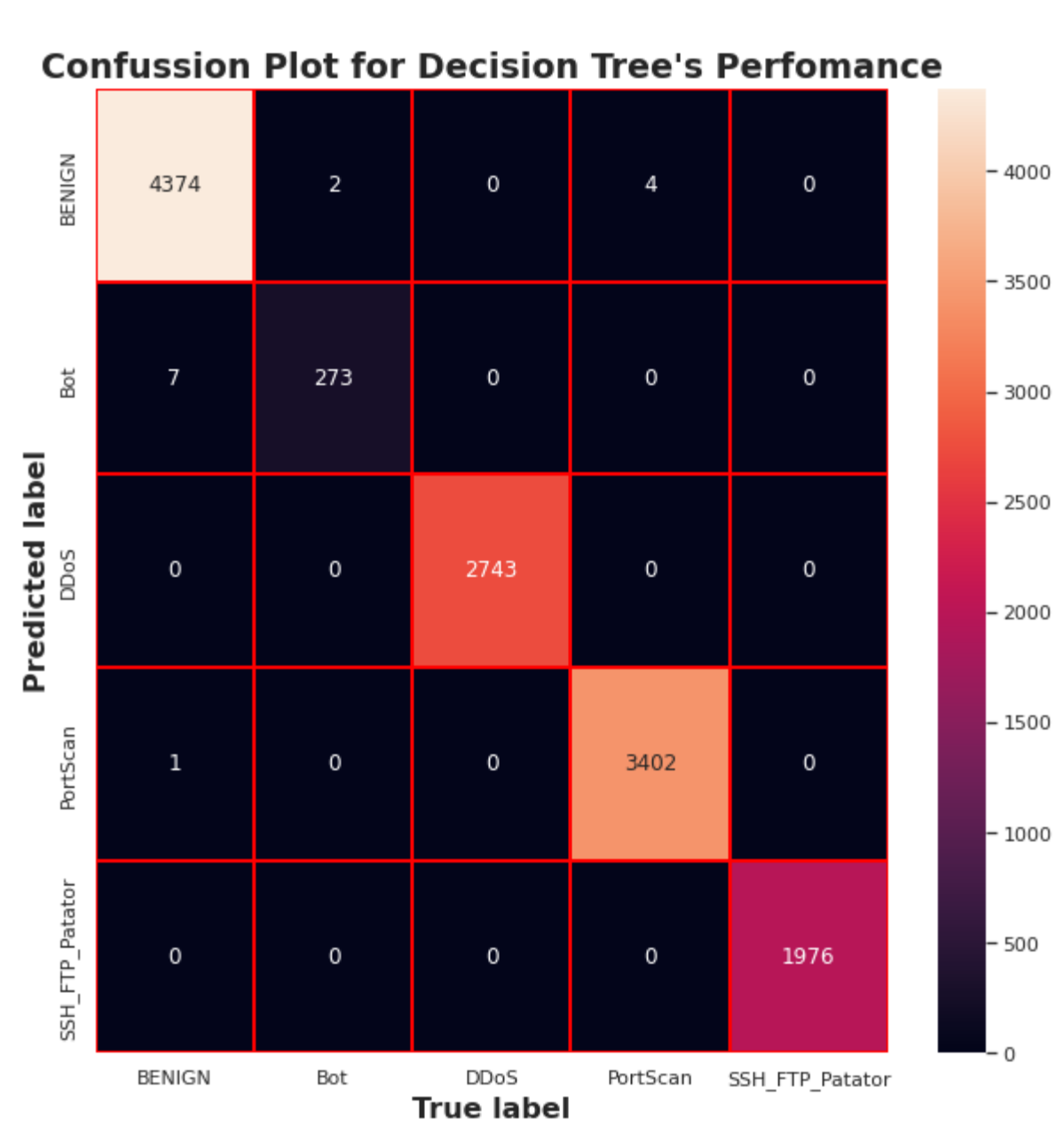
Receiver operating characteristic for Decision Tree



In [56]:
```python
# confussion matric analysis
draw_confusion_matrix(Ytest, decision_model.predict(Xtest), "Decision Tree")
```

```
Decision Tree Metrics Analysis Results

        Accuracy is   0.9989047097480832
        F1 Score is   0.9989017492373268
        Recall Score is   0.9989047097480832
        Precision Score is   0.9989024261350944

        Outputting Classification Report Matrix Plot

         CLASSIFICATION REPORT FOR THE DECISION TREE
                 precision    recall  f1-score   support

          BENIGN       1.00      1.00      1.00      4380
             Bot       0.99      0.97      0.98       280
            DDoS       1.00      1.00      1.00      2743
        PortScan       1.00      1.00      1.00      3403
SSH_FTP_Patator       1.00      1.00      1.00      1976

        accuracy                           1.00     12782
       macro avg       1.00      0.99      1.00     12782
    weighted avg       1.00      1.00      1.00     12782
```

Confussion Plot for Decision Tree's Perfomance



3. Train A RandomForest Model

In [57]:
```python
# Random Forest training and prediction
rf_clf = RandomForestClassifier(random_state = 0)
rf_clf.fit(Xtrain,Ytrain)
rf_score=rf_clf.score(Xtest,Ytest)
rf_clf_predict=rf_clf.predict(Xtest)
print('Accuracy of RF: '+ str(rf_score))
precision,recall,fscore,none= precision_recall_fscore_support(Ytest, rf_clf_predict, average='weighted')
print('Precision of RF: '+(str(precision)))
print('Recall of RF: '+(str(recall)))
print('F1-score of RF: '+(str(fscore)))
```

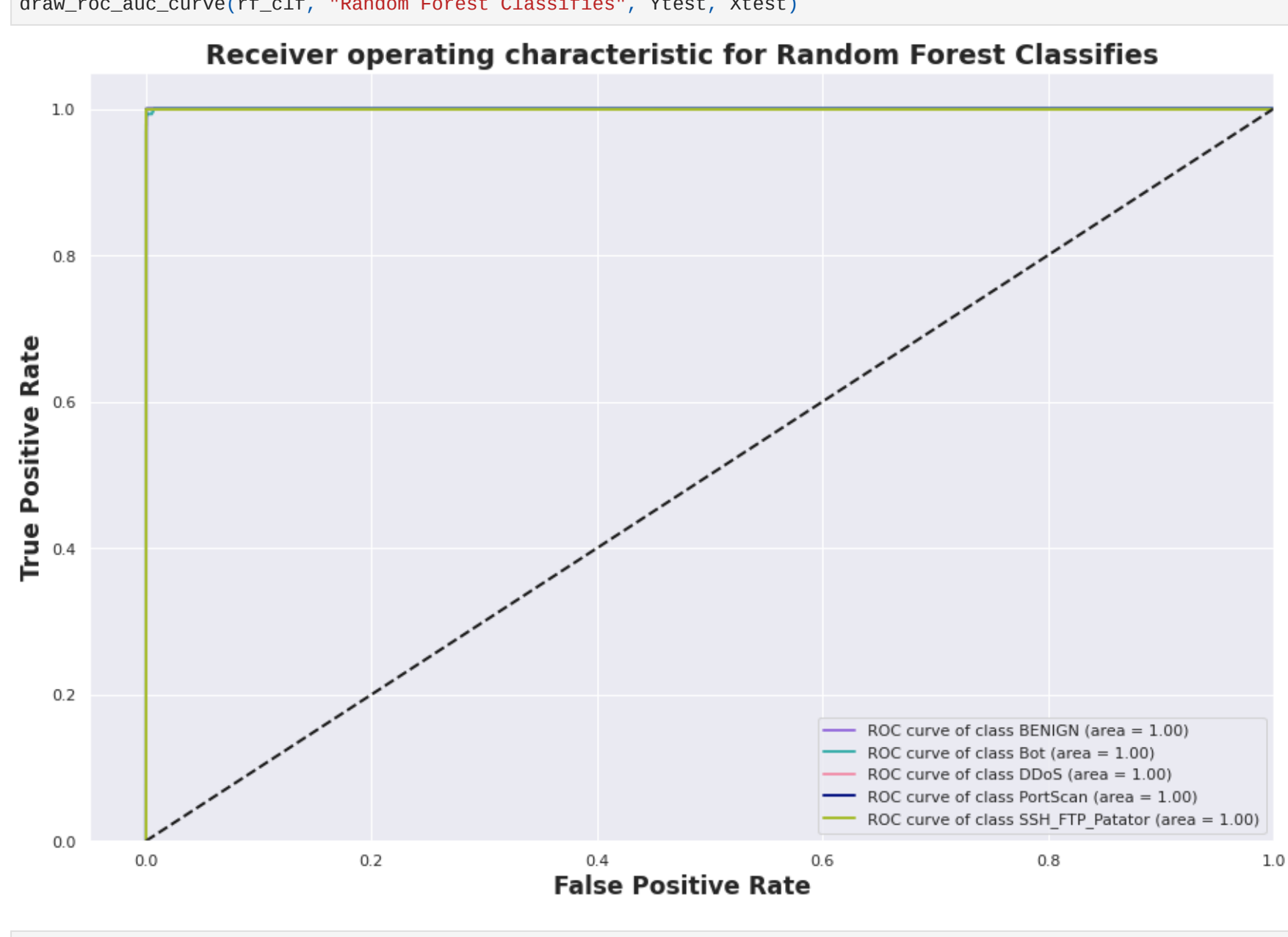
```
Accuracy of RF: 0.9993741198560475
Precision of RF: 0.9993734792732641
Recall of RF: 0.9993741198560475
F1-score of RF: 0.9993731181269735
```

In [58]:
```python
print("Hey there")
```

```
Hey there
```

In [59]:
```python
# get the analysis of roc auc curve fpr random forest
draw_roc_auc_curve(rf_clf, "Random Forest Classifies", Ytest, Xtest)
```

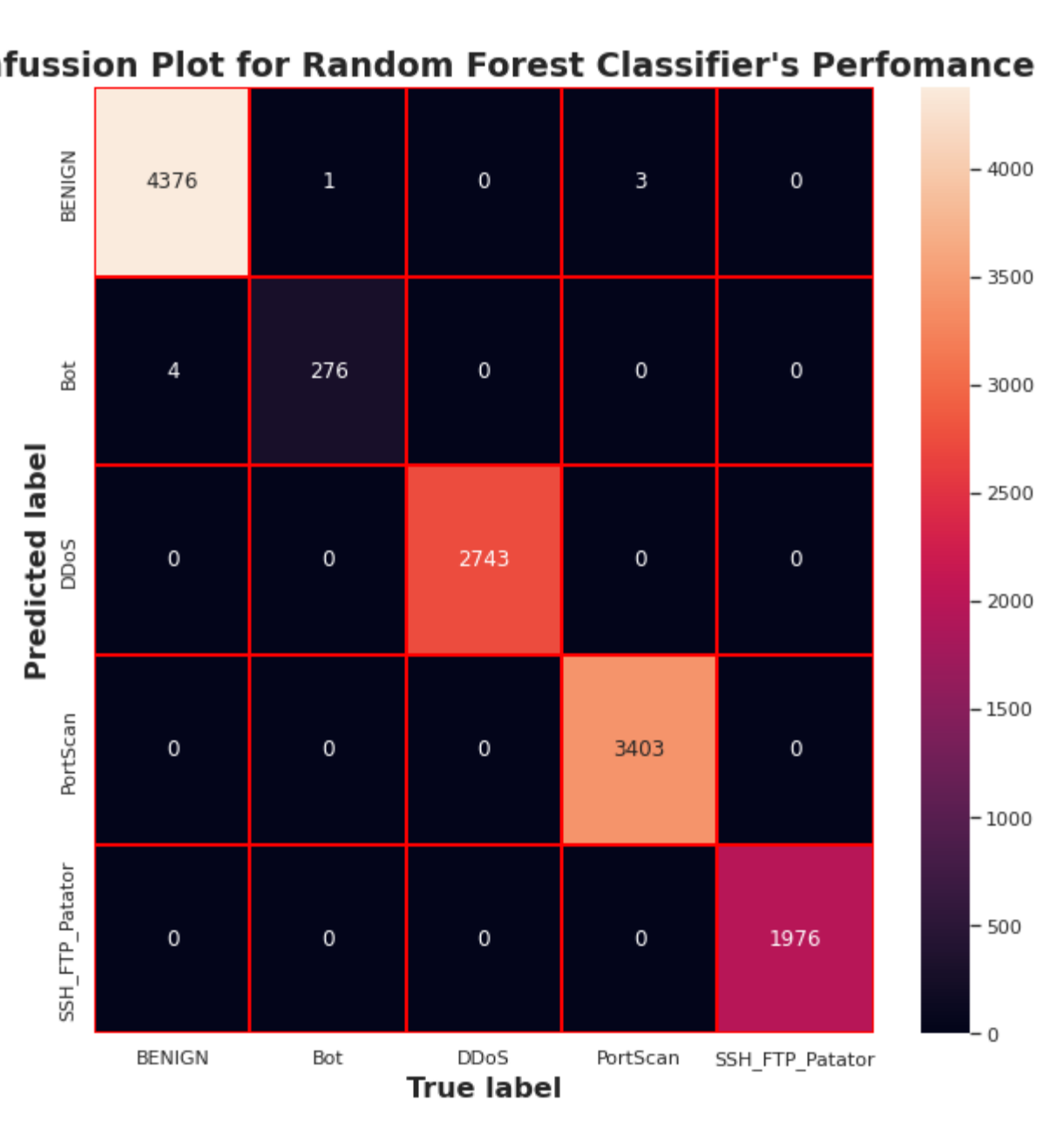
Receiver operating characteristic for Random Forest Classifies



In [60]:
```python
# confussion matric analysis for random forest
draw_confusion_matrix(Ytest, rf_clf.predict(Xtest), "Random Forest Classifier")
```

```
Random Forest Classifier Metrics Analysis Results

        Accuracy is   0.9993741198560475
        F1 Score is   0.9993731181269735
        Recall Score is   0.9993741198560475
        Precision Score is   0.9993734792732641

        Outputting Classification Report Matrix Plot

         CLASSIFICATION REPORT FOR THE RANDOM FOREST CLASSIFIER
                 precision    recall  f1-score   support

          BENIGN       1.00      1.00      1.00      4380
             Bot       1.00      0.99      0.99       280
            DDoS       1.00      1.00      1.00      2743
        PortScan       1.00      1.00      1.00      3403
SSH_FTP_Patator       1.00      1.00      1.00      1976

        accuracy                           1.00     12782
       macro avg       1.00      1.00      1.00     12782
    weighted avg       1.00      1.00      1.00     12782
```

Confussion Plot for Random Forest Classifier's Perfomance



4. XGBOOST Model

In [61]:
```python
# XGboost training and prediction
xgb_clf = XGBClassifier(n_estimators = 500)
xgb_clf.fit(Xtrain,Ytrain)
xgb_score=xgb_clf.score(Xtest,Ytest)
xgb_predict=xgb_clf.predict(Xtest)
print('Accuracy of XGBoost: '+ str(xgb_score))
precision,recall,fscore,none= precision_recall_fscore_support(Ytest, xgb_predict, average='weighted')
print('Precision of XGBoost: '+(str(precision)))
print('Recall of XGBoost: '+(str(recall)))
print('F1-score of XGBoost: '+(str(fscore)))
```

```
Accuracy of XGBoost: 0.9994523548740416
Precision of XGBoost: 0.9994541367653242
Recall of XGBoost: 0.9994523548740416
F1-score of XGBoost: 0.9994528070309752
```

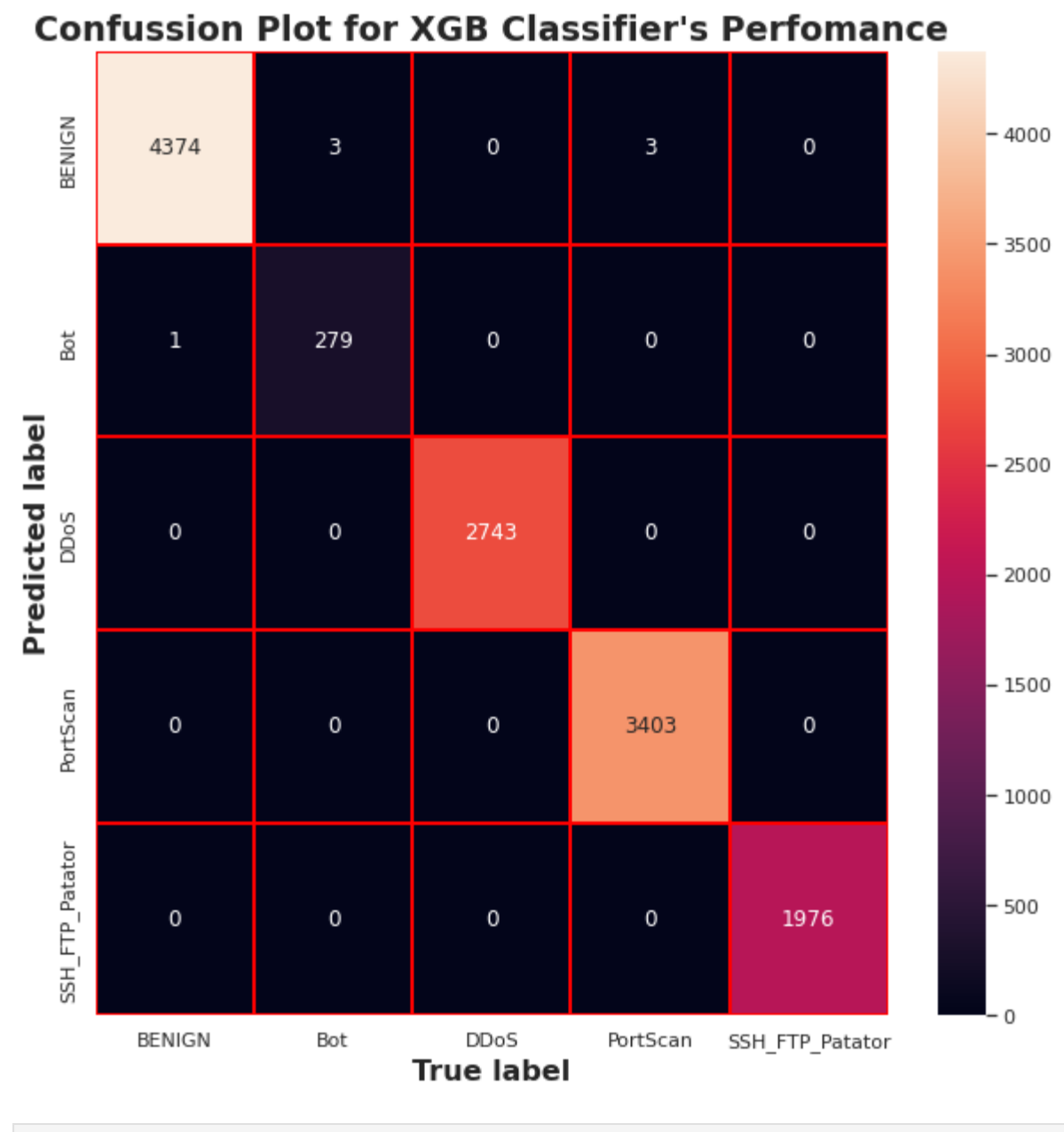
In [62]:
```python
# get the analysis of roc auc curve for xgboost model
draw_roc_auc_curve(xgb_clf, "XGB Classifies", Ytest, Xtest)
```

Receiver operating characteristic for XGB Classifies

In [63]:
```python
# confussion matric analysis for xgboost classifier
draw_confusion_matrix(Ytest, xgb_clf.predict(Xtest), "XGB Classifier")
```

```
XGB Classifier Metrics Analysis Results

        Accuracy is   0.9994523548740416
        F1 Score is   0.9994528070309752
        Recall Score is   0.9994523548740416
        Precision Score is   0.9994541367653242

        Outputting Classification Report Matrix Plot

         CLASSIFICATION REPORT FOR THE XGB CLASSIFIER
                 precision    recall  f1-score   support

          BENIGN       1.00      1.00      1.00      4380
             Bot       0.99      1.00      0.99       280
            DDoS       1.00      1.00      1.00      2743
        PortScan       1.00      1.00      1.00      3403
SSH_FTP_Patator       1.00      1.00      1.00      1976

        accuracy                           1.00     12782
       macro avg       1.00      1.00      1.00     12782
    weighted avg       1.00      1.00      1.00     12782
```

Confussion Plot for XGB Classifier's Perfomance



In [64]:
```python
# same random forest model

pd.to_pickle(rf_clf, "rf_clf.pkl")
```

Feature Selection and Dimension Reduction for CICIDS2017 DATASET.

- In Machine Learning, feature selection and dimension reduction are important techniques used to improve the performance and efficiency of a model by reducing the number of input variables. The CICIDS2017 dataset provides a large number of features that can be used to train a machine learning model, however, not all of these features may be useful or necessary.

Feature Selection

- Feature selection is the process of selecting a subset of the most relevant features from the available feature set, based on a certain criterion. This helps to remove irrelevant and redundant features, and can improve the accuracy, speed, and interpretability of the model.
- One common method of feature selection is to use a univariate statistical test, such as the chi-squared test or ANOVA, to identify the most relevant features. Another method is to use feature importances generated by tree-based algorithms, such as Random Forest or XGBoost, to select the most important features. In this project, I will use CHI_SQUARED TEST to select the top 40 features that best descript the output of a certain data packets. THose other Featiures will be ignored.

Dimension Reduction

- Dimension reduction is the process of changing a high-dimensional feature space into a lower-dimensional representation, while preserving the relevant information i.e We try to reduce Curve of dimensionality by reducing the number of features while keeping as much as possible information about the daatset.
- This can be achieved through techniques such as principal component analysis (PCA), linear discriminant analysis (LDA), and t-distributed stochastic neighbor embedding (t-SNE).
- One common use of dimension reduction is to reduce the computational complexity of the model and to remove multicollinearity in the feature set. It can also be used for data visualization purposes, allowing patterns and relationships in the data to be more easily understood.
- FOr this project, PCA will be used as it it the eeasier method available. All Features from Feature selection Stage will be reduced into 20 Features i.e 40 features will be transafomed into 40 features

---

- By applying feature selection and dimension reduction techniques, we can improve the performance and efficiency of machine learning models (Beconmes more fast to train and Inference than running on all features), while also making the data more interpretable.

LETS DO FEATURE SELECTIONS TO GET TOP 40 Features.

In [65]:
```python
# check the data available
X.shape, Y.shape
```

Out[65]:
```
((89476, 70), (89476,))
```

In [66]:
```python
# scaler the training features
scaler = StandardScaler()
scaler.fit(X)
```

Out[66]:
```
StandardScaler()
```

In [67]:
```python
df_s.shape
```

Out[67]:
```
(89476, 71)
```

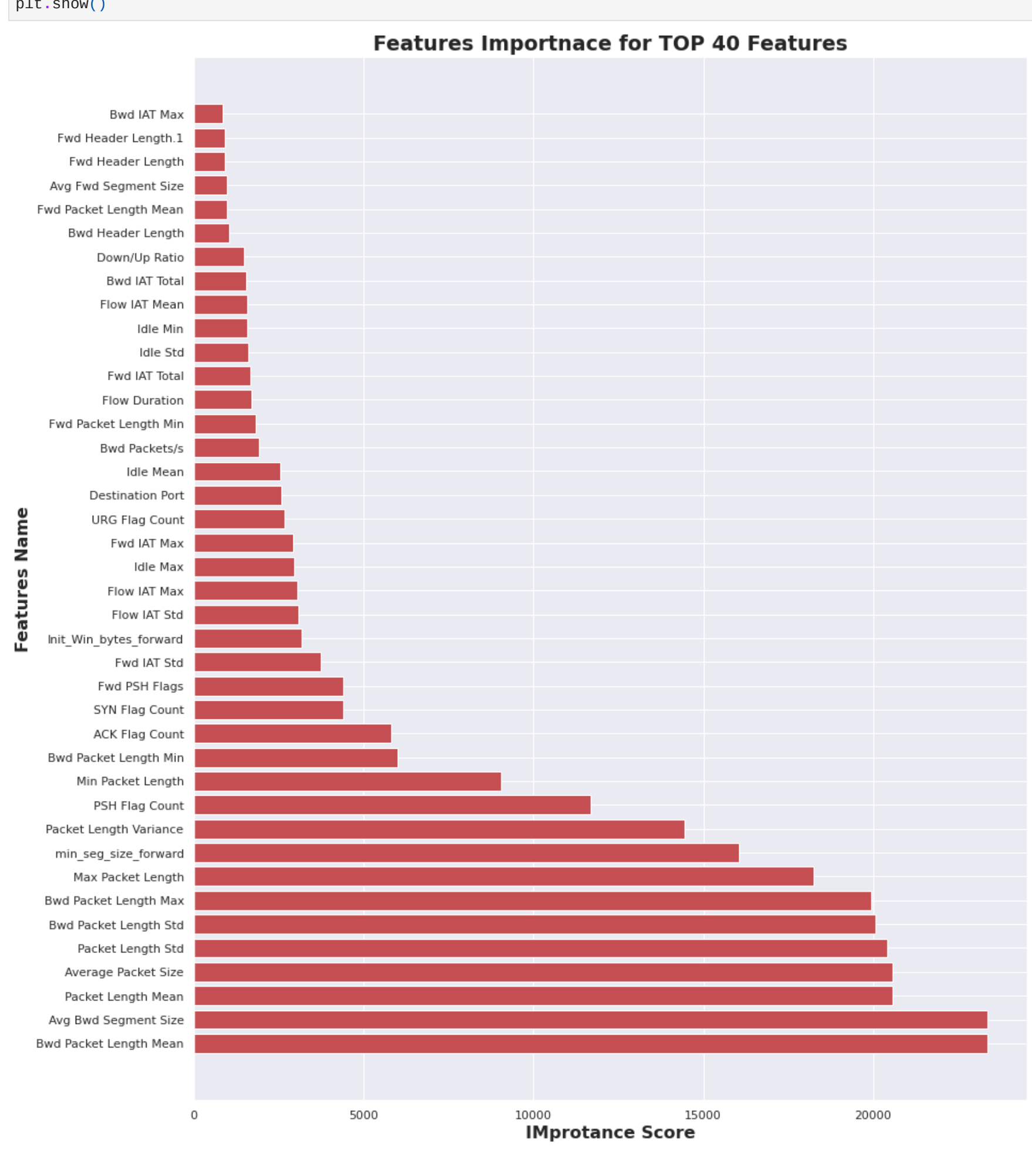
In [68]:
```python
# Apply SelectKBest class to extract top 40 best features
bestfeatures = SelectKBest(score_func=f_classif, k=40)
# fit the algorithm for selection
bestfeatures_fit = bestfeatures.fit(scaler.transform(X), Y)

# get the scores for each featuire
bestfeatures_scores = pd.DataFrame(bestfeatures_fit.scores_)
bestfeatures_cols = pd.DataFrame(df.drop("Label", axis=1).columns)

# Concatenate two dataframes for better visualization
featureScores = pd.concat([bestfeatures_cols, bestfeatures_scores], axis=1)
# rename columsn
featureScores.columns = ['Specs', 'Score']

# get the tiop 10
print(featureScores.nlargest(10, 'Score'))
```

```
                      Specs         Score
12  Bwd Packet Length Mean  23377.776158
52    Avg Bwd Segment Size  23377.776158
38      Packet Length Mean  20575.791615
50     Average Packet Size  20561.120649
39       Packet Length Std  20407.351124
13   Bwd Packet Length Std  20072.057802
10   Bwd Packet Length Max  19959.085347
37       Max Packet Length  18237.738136
61    min_seg_size_forward  16050.970228
40  Packet Length Variance  14444.168514
```

In [ ]:

In [69]:
```python
# Get the indices sorted by most important to least important
indices = np.argsort(bestfeatures_fit.scores_)[::-1]
# To get your top 40 feature names
features = []
for i in range(40):
    features.append(df.drop("Label", axis=1).columns[indices[i]])
# Now plot
plt.figure(figsize=(14,18))
plt.barh(features, bestfeatures_fit.scores_[indices[range(40)]], color='r', align='center')
plt.ylabel('Features Name', fontsize=16, fontweight='bold')
plt.xlabel('IMprotance Score', fontsize=16, fontweight='bold')
plt.title(f"Features Importnace for TOP 40 Features", fontsize=19, fontweight='bold')
plt.show()
```

Features Importnace for TOP 40 Features



Results

- From the above, we have seen the top 40 features that contributes greatly to the type of packed.
- One of them is AVG BWD segment size which s the topmost features in determining what is the target type.
- I will only use thse 40 features to the next process of dimensionality reduction.
- The value 40 is selected randomly.

In [70]:
```python
TOP_N_FEATURES = features
print(TOP_N_FEATURES)
```

```
['Bwd Packet Length Mean', 'Avg Bwd Segment Size', 'Packet Length Mean', 'Average Packet Size', 'Packet Length Std', 'Bwd Packet Length Std', 'Bwd Packet Length Max', 'Max Packet Length', 'min_seg_size_forward', 'Packet Length Variance', 'PSH Flag Count', 'Min Packet Length', 'Bwd Packet Length Min', 'ACK Flag Count', 'SYN Flag Count', 'Fwd PSH Flags', 'Fwd IAT Std', 'Init_Win_bytes_forward', 'Flow IAT Std', 'Flow IAT Max', 'Idle Max', 'Fwd IAT Max', 'URG Flag Count', 'Destination Port', 'Idle Mean', 'Bwd Packets/s', 'Fwd Packet Length Min', 'Flow Duration', 'Fwd IAT Total', 'Idle Std', 'Idle Min', 'Flow IAT Mean', 'Bwd IAT Total', 'Down/Up Ratio', 'Bwd Header Length', 'Fwd Packet Length Mean', 'Avg Fwd Segment Size', 'Fwd Header Length', 'Fwd Header Length.1', 'Bwd IAT Max']
```

In [71]:
```python
len(TOP_N_FEATURES)
```

Out[71]:
```
40
```

PCA FOR DIMESNIONS REDUCTION.

- Below is implimentation of PCA that takes the whole 40 features data selected above and reduce it into an array of 20 features (just a random selection).
- Here is the code

In [72]:
```python
# define scaler to transform the dataset
scaler = StandardScaler()
scaler.fit(df[TOP_N_FEATURES])

pca_model = PCA(n_components=20)
principalComponents = pca_model.fit_transform(scaler.transform(df[TOP_N_FEATURES]))
principalDf = pd.DataFrame(data=principalComponents, columns=[f'PCA_{x}' for x in range(pca_model.n_components_
```

In [73]:
```python
# check the selected Data frame
principalDf.head()
```

Out[73]:

	PCA_0	PCA_1	PCA_2	PCA_3	PCA_4	PCA_5	PCA_6	PCA_7	PCA_8	PCA_9	PCA_10	PCA_11	PCA_1
0	-1.561199	-0.136754	0.149967	0.252891	-0.876953	-0.017584	-0.066270	-0.043347	-0.670671	0.627482	0.088956	-0.375503	0.80415
1	-1.712553	1.361358	-2.138056	-3.776224	2.654966	0.025330	-2.763876	-0.295349	0.806062	0.165943	-0.053345	0.244204	0.14730
2	12.505827	4.571741	-2.542435	-0.929006	-1.586993	1.396585	1.335626	5.150638	1.133649	0.613578	-3.629255	-0.827866	0.79968
3	-0.514642	1.466967	-1.312680	-0.591605	0.317364	0.658886	0.605404	0.210551	-1.638283	0.057827	0.553909	0.359204	0.33649
4	-1.712434	1.361346	-2.138079	-3.776181	2.655332	0.025573	-2.764217	-0.295252	0.805109	0.163531	-0.053620	0.243391	0.14749

SPLIT THE NEW DATASET FOR TRAINING AND VALIDAITION.

- Same code as above will be used..

In [74]:
```python
# we copied same code from above but these time we use training features that is transformed
for train_index, test_index in skf.split(principalDf, Y):
    Xtrain, Xtest = X[train_index], X[test_index]
    Ytrain, Ytest = Y[train_index], Y[test_index]

    reshaped_y_train = np.asarray(Ytrain).reshape(-1, 1)
    reshaped_y_test = np.asarray(Ytest).reshape(-1, 1)

print('X_train length: ', len(Xtrain) ) # To check if splits worked
print('y_train length: ', len(Ytrain) )
print('X_test length: ', len(Xtest) )
print('y_test length: ', len(Ytest) )
```

```
X_train length:  76694
y_train length:  76694
X_test length:  12782
y_test length:  12782
```

Retraining Models to observe their Perfomacne.

- I willl use similar models as above Except that I will only Test Logistic Regresision and Random Forest and Later Build An ENsemble with the four models and check with the same metrics.

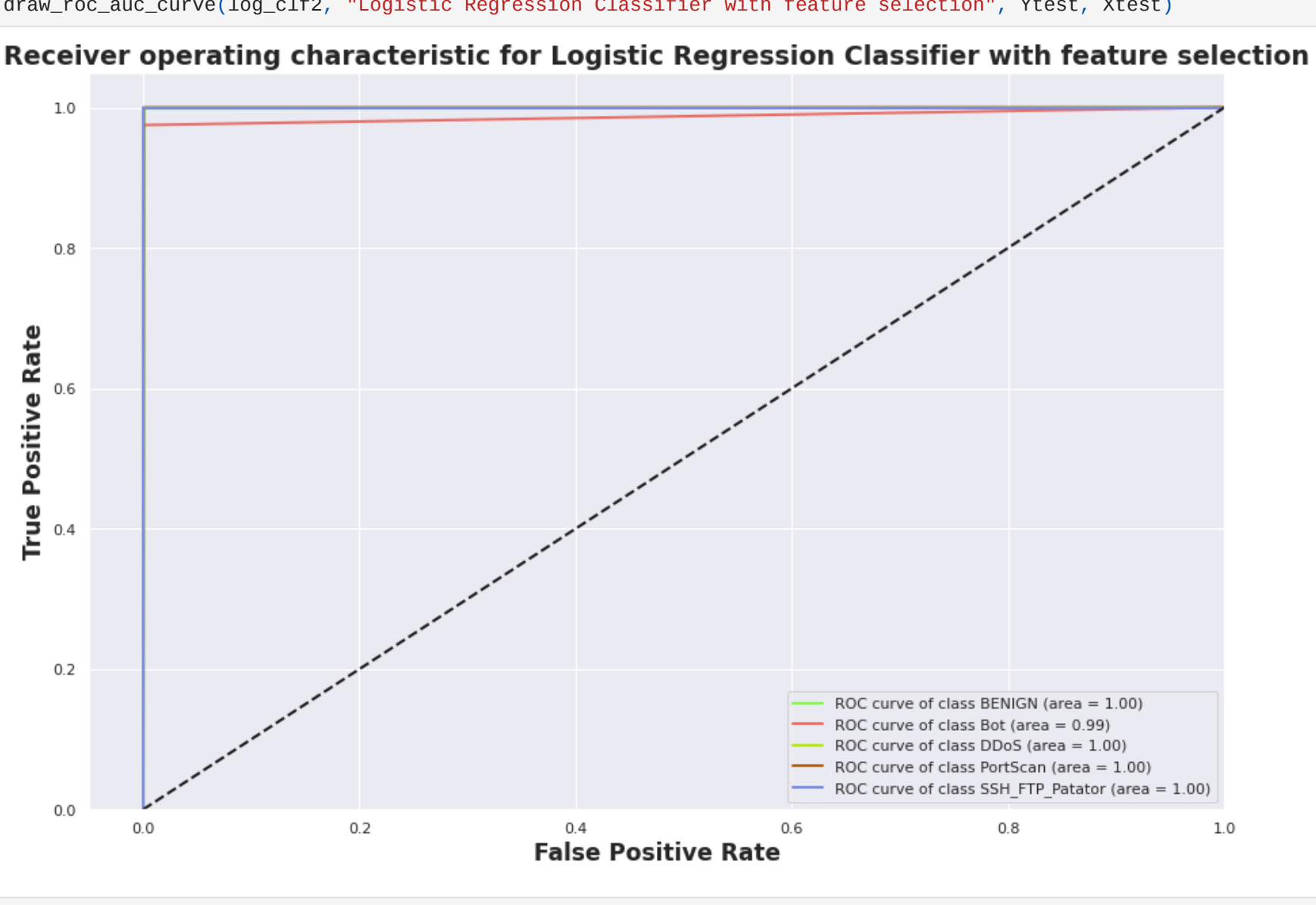
1b. Logistci regression with Feature selection and Dimensionality reduction

In [75]:
```python
# Logistic Regression Model
log_clf2 = DecisionTreeClassifier(random_state = 0)
# train the model
log_clf2.fit(Xtrain,Ytrain)


# get model score and predictions
log_score=log_clf2.score(Xtest,Ytest)
log_preds=log_clf2.predict(Xtest)
print('Accuracy of Logistic Regression Classifier with Feature selection: '+ str(log_score))
precision,recall,fscore,none= precision_recall_fscore_support(Ytest, log_preds, average='weighted')
print('Precision of Logistic Regression Classifier  with Feature selection: '+(str(precision)))
print('Recall of Logistic Regression Classifier with Feature selection: '+(str(recall)))
print('F1-score of Logistic Regression Classifier with Feature selection: '+(str(fscore)))
```

```
Accuracy of Logistic Regression Classifier with Feature selection: 0.9989047097480832
Precision of Logistic Regression Classifier  with Feature selection: 0.9989024261350944
Recall of Logistic Regression Classifier with Feature selection: 0.9989047097480832
F1-score of Logistic Regression Classifier with Feature selection: 0.9989017492373268
```

In [76]:
```python
# Feature reduction resule
# get the analysis of roc auc curve for logistic regression model
draw_roc_auc_curve(log_clf2, "Logistic Regression Classifier with feature selection", Ytest, Xtest)
```

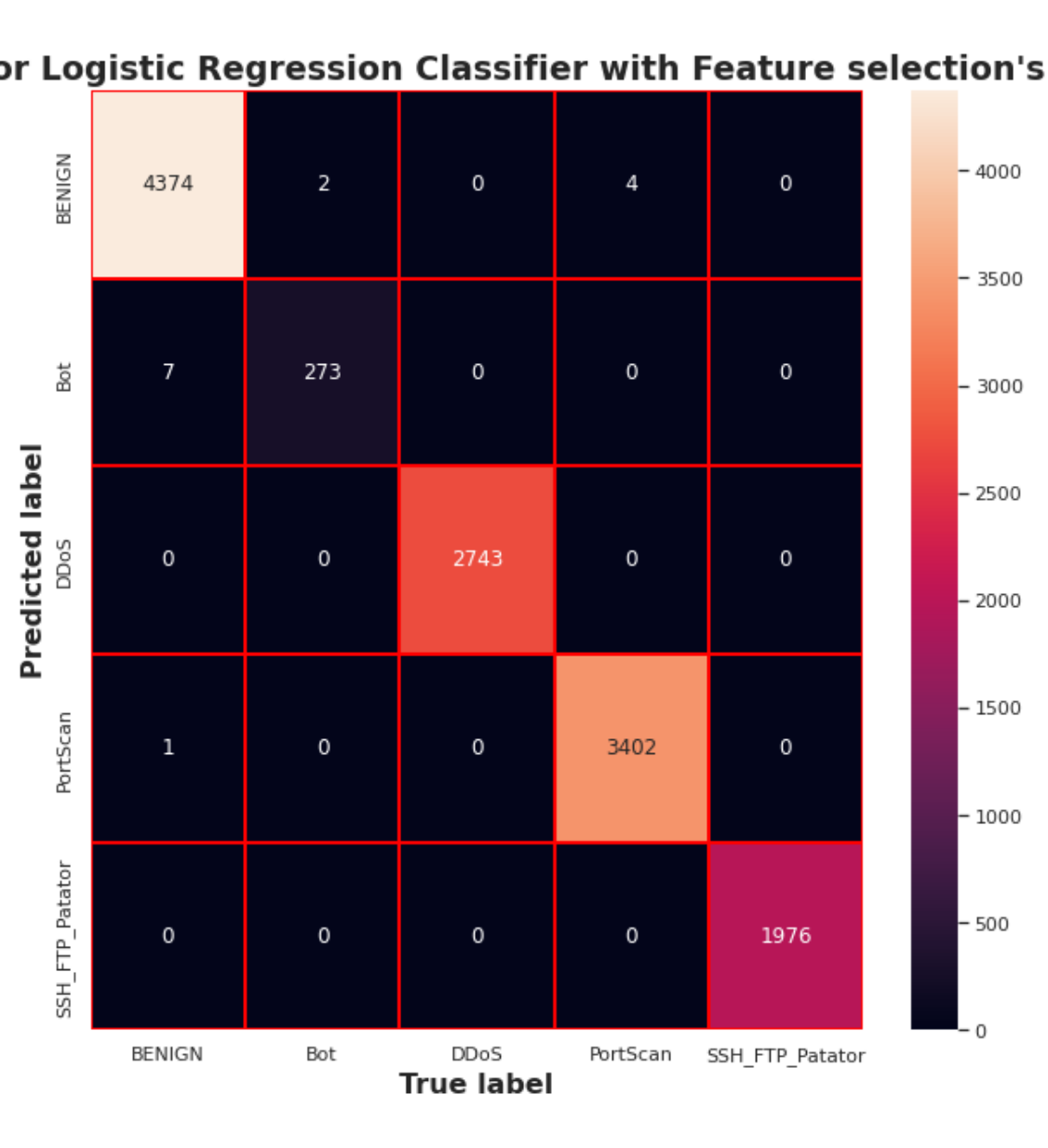
Receiver operating characteristic for Logistic Regression Classifier with feature selection



In [77]:
```python
# confussion matric analysis for logistic regression model
draw_confusion_matrix(Ytest, log_clf2.predict(Xtest), "Logistic Regression Classifier with Feature selection")
```

```
Logistic Regression Classifier with Feature selection Metrics Analysis Results

        Accuracy is   0.9989047097480832
        F1 Score is   0.9989017492373268
        Recall Score is   0.9989047097480832
        Precision Score is   0.9989024261350944

        Outputting Classification Report Matrix Plot

         CLASSIFICATION REPORT FOR THE LOGISTIC REGRESSION CLASSIFIER WITH FEATURE SELECTION
                 precision    recall  f1-score   support

          BENIGN       1.00      1.00      1.00      4380
             Bot       0.99      0.97      0.98       280
            DDoS       1.00      1.00      1.00      2743
        PortScan       1.00      1.00      1.00      3403
SSH_FTP_Patator       1.00      1.00      1.00      1976

        accuracy                           1.00     12782
       macro avg       1.00      0.99      1.00     12782
    weighted avg       1.00      1.00      1.00     12782
```

Confussion Plot for Logistic Regression Classifier with Feature selection's Perfomance



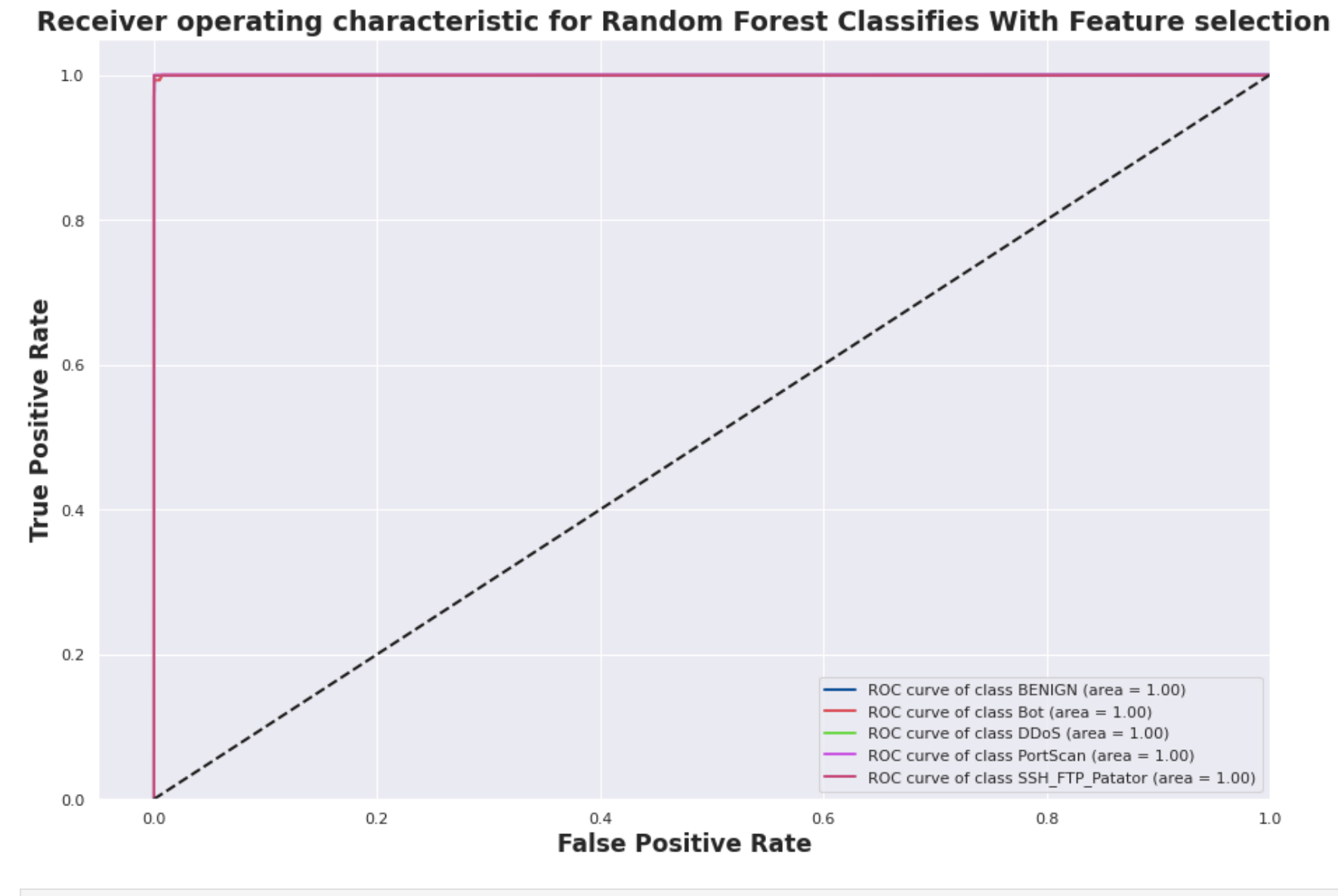
2b Random Forest with PCA and Feature selection

In [78]:
```python
# Random Forest training and prediction
rf_clf2 = RandomForestClassifier(random_state = 0)
rf_clf2.fit(Xtrain,Ytrain)
rf_score=rf_clf2.score(Xtest,Ytest)
rf_clf_predict=rf_clf2.predict(Xtest)
print('Accuracy of RF With Feature selection: '+ str(rf_score))
precision,recall,fscore,none= precision_recall_fscore_support(Ytest, rf_clf_predict, average='weighted')
print('Precision of RF With Feature selection: '+(str(precision)))
print('Recall of RF With Feature selection: '+(str(recall)))
print('F1-score of RF With Feature selection: '+(str(fscore)))
```

```
Accuracy of RF With Feature selection: 0.9993741198560475
Precision of RF With Feature selection: 0.9993734792732641
Recall of RF With Feature selection: 0.9993741198560475
F1-score of RF With Feature selection: 0.9993731181269735
```

In [79]:
```python
# RF WITH FEATURE SELECTION
# get the analysis of roc auc curve fpr random forest
draw_roc_auc_curve(rf_clf, "Random Forest Classifies With Feature selection", Ytest, Xtest)
```

Receiver operating characteristic for Random Forest Classifies With Feature selection



In [80]:
```
draw_confusion_matrix(Ytest, rf_clf.predict(Xtest), "Random Forest Classifier With Feature selection")
```

```
Random Forest Classifier With Feature selection Metrics Analysis Results

        Accuracy is   0.9993741198560475
        F1 Score is   0.9993731181269735
        Recall Score is   0.9993741198560475
        Precision Score is   0.9993734792732641

        Outputting Classification Report Matrix Plot

         CLASSIFICATION REPORT FOR THE RANDOM FOREST CLASSIFIER WITH FEATURE SELECTION
                 precision    recall  f1-score   support

         BENIGN       1.00      1.00      1.00      4380
            Bot       1.00      0.99      0.99       280
           DDoS       1.00      1.00      1.00      2743
       PortScan       1.00      1.00      1.00      3403
SSH_FTP_Patator       1.00      1.00      1.00      1976

       accuracy                           1.00     12782
      macro avg       1.00      1.00      1.00     12782
   weighted avg       1.00      1.00      1.00     12782
```

Confussion Plot for Random Forest Classifier With Feature selection's Perfomance

FINALLY.

ENSEMBLING THROUGH STACKING.

- A stacking model is an ensemble learning technique where multiple models are combined to make a final prediction. In this case, I will stack random forest, logistic regression, XGBoost, and decision trees to build a more powerful model.
- The four base models (random forest, logistic regression, XGBoost, and decision trees) will be trained on the training data, and their predictions are generated for both the training and testing data. These predictions are then concatenated and used as input for the meta model (logistic regression in this case), which is also trained on the training data. Finally, the accuracy of the stacking model is calculated using the testing data. Below is the implimentation

In [81]:
```
# Initialize the base models
rf = RandomForestClassifier(n_estimators=100, random_state=0)
lr = LogisticRegression(random_state=0)
xgb = XGBClassifier(random_state=0)
dt = DecisionTreeClassifier(random_state=0)

# Fit the base models on the training data
rf.fit(Xtrain, Ytrain)
lr.fit(Xtrain, Ytrain)
xgb.fit(Xtrain, Ytrain)
dt.fit(Xtrain, Ytrain)
print("Done Predicting Base Models")
```

```
Done Predicting Base Models
```

In [86]:
```
# get probs
rf.predict_proba(Xtrain)
```

Out[86]:
```
array([[0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0.],
       ...,
       [0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0.]])
```

In [87]:
```
# Generate predictions for the training data using the base models
train_preds_rf = rf.predict_proba(Xtrain)[:]
train_preds_lr = lr.predict_proba(Xtrain)[:]
train_preds_xgb = xgb.predict_proba(Xtrain)[:]
train_preds_dt = dt.predict_proba(Xtrain)[:]

# Concatenate the predictions from the base models
train_preds = np.column_stack((train_preds_rf, train_preds_lr, train_preds_xgb, train_preds_dt))

# Generate predictions for the testing data using the base models
test_preds_rf = rf.predict_proba(Xtest)[:]
test_preds_lr = lr.predict_proba(Xtest)[:]
test_preds_xgb = xgb.predict_proba(Xtest)[:]
test_preds_dt = dt.predict_proba(Xtest)[:]

# Concatenate the predictions from the base models
test_preds = np.column_stack((test_preds_rf, test_preds_lr, test_preds_xgb, test_preds_dt))

print("Done getting concantenated predicitions from base models")
```

```
Done getting concantenated predicitions from base models
```

In [89]:
```
test_preds.shape, train_preds.shape
```

Out[89]:
```
((12782, 20), (76694, 20))
```

In [90]:
```
test_preds
```

Out[90]:
```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 1., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

In [91]:
```
# BUILDING A STACKER MODELS AS RandomForestClassifier
# Initialize the meta model
meta_model = RandomForestClassifier(random_state=0, n_estimators=1000)

# Fit the meta model on the predictions from the base models
meta_model.fit(train_preds, Ytrain)

# Generate predictions for the testing data using the meta model
final_preds = meta_model.predict(test_preds)
```

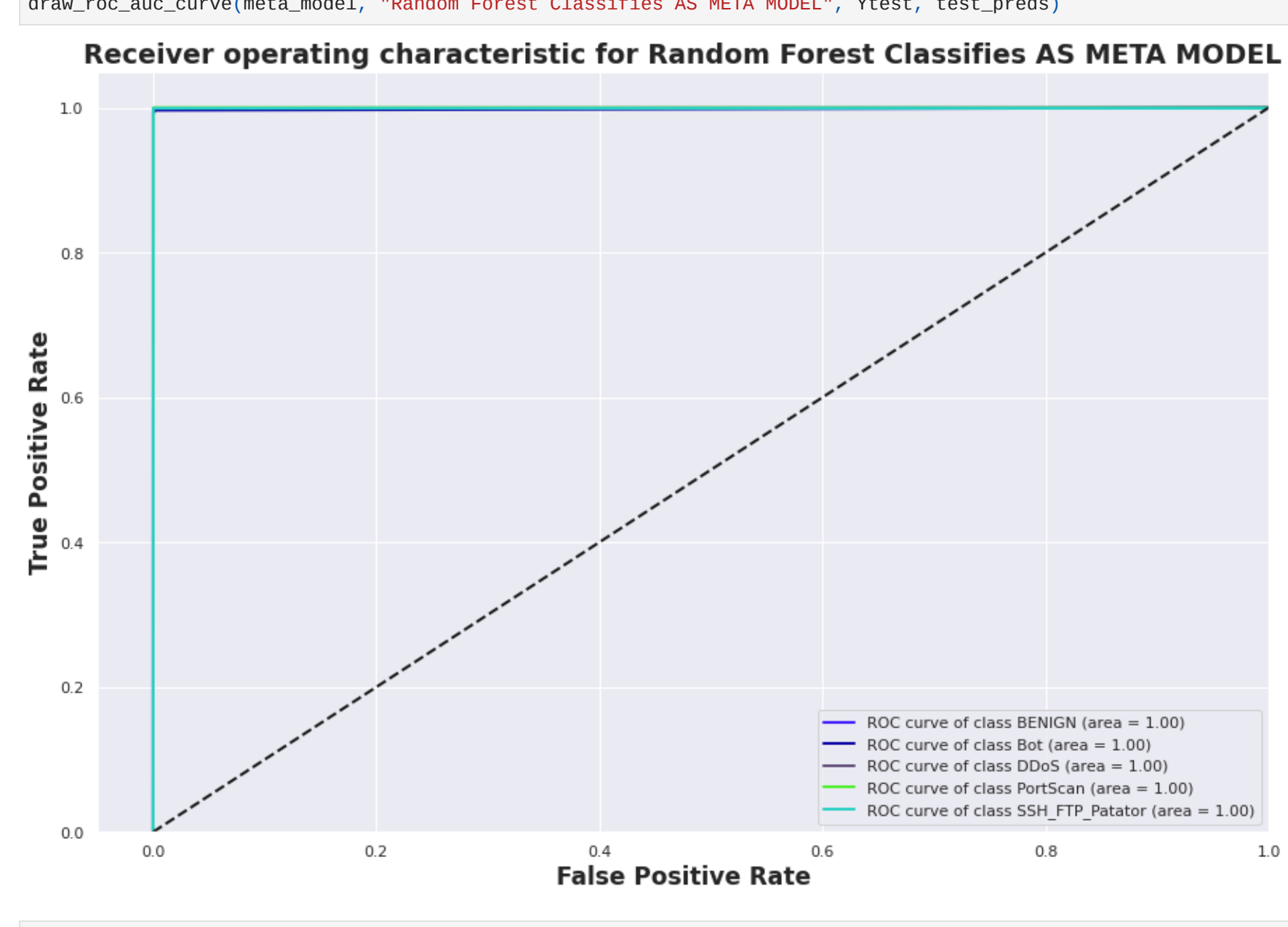
In [95]:
```
# EVALUATATE THE META MODEL TO SEE ITS PERFOMANXE

# Calculate the accuracy of the

accuracy = accuracy_score(Ytest, final_preds)
print("Accuracy of Stacking Model: {:.2f}%".format(accuracy*100))
```

```
Accuracy of Stacking Model: 99.96%
```

EVALUATIONG STACKED MODELS

In [93]:
```
# STACKED MODEL EVALUATION WITH FEATURE SELECTION
# FPR analysis for meta model
draw_roc_auc_curve(meta_model, "Random Forest Classifies AS META MODEL", Ytest, test_preds)
```

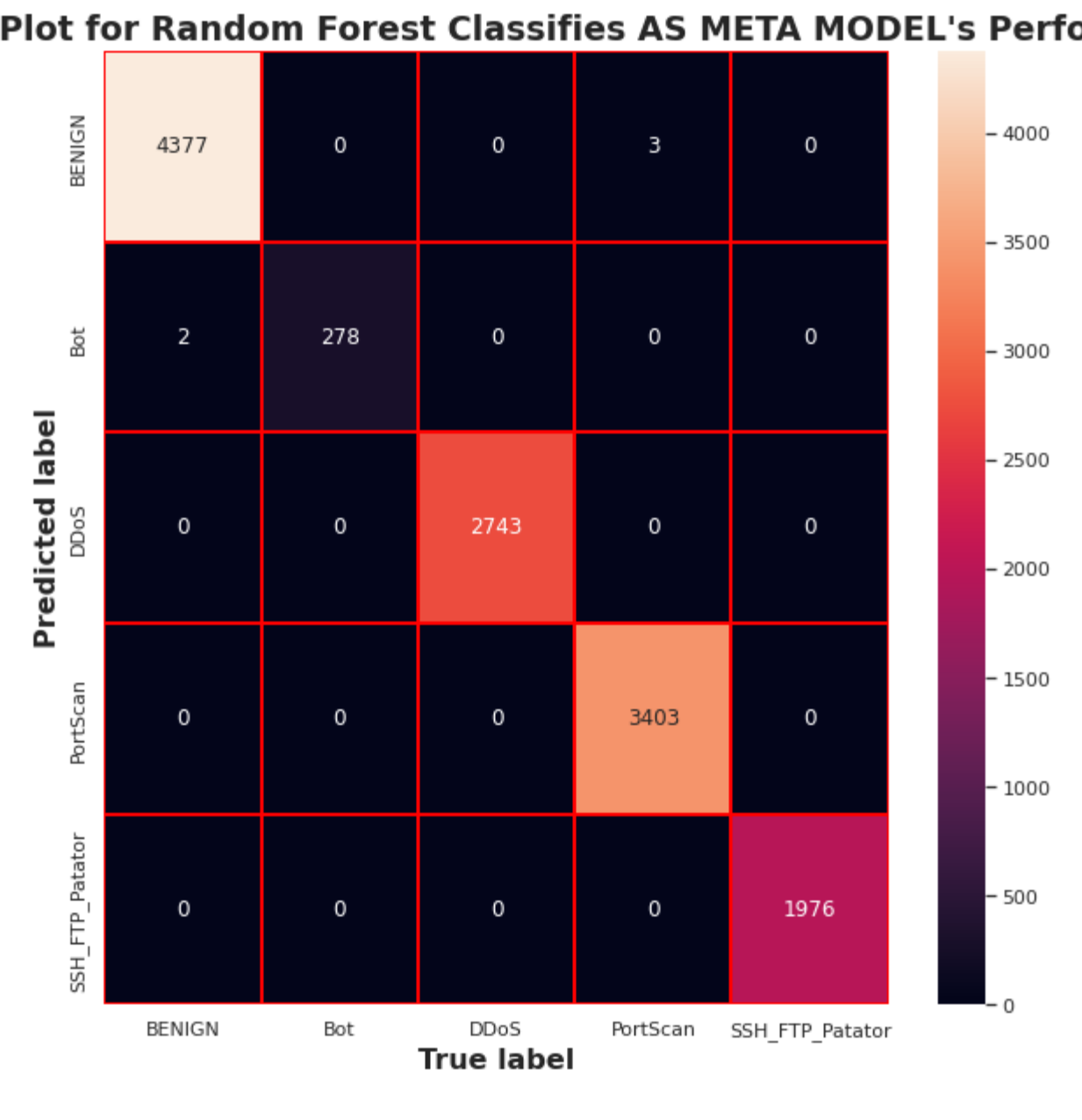
Receiver operating characteristic for Random Forest Classifies AS META MODEL



In [94]:
```
# plot confunction matrix for meta model
draw_confusion_matrix(Ytest, final_preds, "Random Forest Classifies AS META MODEL")
```

```
Random Forest Classifies AS META MODEL Metrics Analysis Results

        Accuracy is   0.9996088249100298
        F1 Score is   0.9996085738726254
        Recall Score is   0.9996088249100298
        Precision Score is   0.9996089959059999

        Outputting Classification Report Matrix Plot

         CLASSIFICATION REPORT FOR THE RANDOM FOREST CLASSIFIES AS META MODEL
                 precision    recall  f1-score   support

         BENIGN       1.00      1.00      1.00      4380
            Bot       1.00      0.99      1.00       280
           DDoS       1.00      1.00      1.00      2743
       PortScan       1.00      1.00      1.00      3403
SSH_FTP_Patator       1.00      1.00      1.00      1976

       accuracy                           1.00     12782
      macro avg       1.00      1.00      1.00     12782
   weighted avg       1.00      1.00      1.00     12782
```

Confussion Plot for Random Forest Classifies AS META MODEL's Perfomance



CONCLUSION.

- The results of a stacking model using Random Forest as the meta model show that the individual models (Random Forest, Logistic Regression, XGBoost, Decision tree clasisifer) were outperformed better by the stacking model. The accuracy, F1 score, recall score, and precision score of the individual models were abit lower compared to the stacking model. The accuracy of the Random Forest model was 0.999, the Logistic Regression model was 0.998, the Decision treee model was 0.999,and the XGBoost model was 0.999. However, the stacking model had an accuracy of only 1.000 Where only 3 classes were not predicted accurately.
- Based on the results of the individual models and the stacking model, it is clear that the Random Forest, Logistic Regression, and XGBoost and Metal Model models performed better. To determine which model to use, it would be necessary to consider additional factors such as the business problem and the specific requirements for the model.
- Since the goal was to achieve the best model, it seems that using the Metal Model of Random Forest or XGBoost model would be the best choice.
- Normally it is important to thoroughly evaluate and compare the performance of multiple models to make an informed decision as we have done above with different situations in order ot come into conclusion on the best selection. Also we could include trying different meta models, feature selection techniques, and algorithms to ensure that the best possible model is chosen for the given problem but since we hace achieved a good score we let it go.
- In conclusion, the meta model in this case did perform as well as the individual models with a slight improvement. More perfomance can be achieved though in various ways, i.e such as the choice of meta model or the way in which the predictions from the base models were combined. It is important to carefully consider the model selection and implementation when using stacking to improve the performance.

DONE

In [ ]: