In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
import numpy as np
import pandas as pd

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

In [3]:
```
df_data_1 = pd.read_parquet('cicids2017/Benign-Monday-no-metadata.parquet')
df_data_2 = pd.read_parquet('cicids2017/Botnet-Friday-no-metadata.parquet')
df_data_3 = pd.read_parquet('cicids2017/Bruteforce-Tuesday-no-metadata.parquet')
df_data_4 = pd.read_parquet('cicids2017/DDoS-Friday-no-metadata.parquet')
df_data_5 = pd.read_parquet('cicids2017/DoS-Wednesday-no-metadata.parquet')
df_data_7 = pd.read_parquet('cicids2017/Portscan-Friday-no-metadata.parquet')
df_data_8 = pd.read_parquet('cicids2017/WebAttacks-Thursday-no-metadata.parquet')
```

In [4]:
```
data = pd.concat([df_data_1, df_data_2, df_data_3, df_data_4,
                  df_data_5, df_data_7, df_data_8], axis=0, ignore_index=True)
```

In [6]:
```
# Load data
data.columns
```

Out[6]:
```
Index(['Protocol', 'Flow Duration', 'Total Fwd Packets',
       'Total Backward Packets', 'Fwd Packets Length Total',
       'Bwd Packets Length Total', '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',
       'Packet Length Min', 'Packet Length Max', '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',
       'Avg Packet Size', 'Avg Fwd Segment Size', 'Avg Bwd Segment Size',
       '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 Fwd Win Bytes', 'Init Bwd Win Bytes',
       'Fwd Act Data Packets', 'Fwd Seg Size Min', 'Active Mean', 'Active Std',
       'Active Max', 'Active Min', 'Idle Mean', 'Idle Std', 'Idle Max',
       'Idle Min', 'Label'],
      dtype='object')
```

In [7]:
```
null_counts = data.isnull().sum()
# Print the number of null values
print(f"{null_counts.sum()} null entries have been found in the dataset\n")
# Drop null values
data.dropna(inplace=True)          # or df_data = df_data.dropna()

# Find and handle duplicates
duplicate_count = data.duplicated().sum()
# Print the number of duplicate entries
print(f"{duplicate_count} duplicate entries have been found in the dataset\n")
# Remove duplicates
data.drop_duplicates(inplace=True)  # or df_data = df_data.drop_duplicates()
# Display relative message
print(f"All duplicates have been removed\n")

# Reset the indexes
data.reset_index(drop=True, inplace=True)

# Inspect the dataset for categorical columns
print("Categorical columns:",data.select_dtypes(include=['object']).columns.tolist(),'\n')

# Print the first 5 lines
data.head()
```

```
0 null entries have been found in the dataset

61963 duplicate entries have been found in the dataset

All duplicates have been removed

Categorical columns: ['Label']
```

Out[7]:

	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Fwd Packets Length Total	Bwd Packets Length Total	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
0	6	4	2	0	12	0	6	6	6.00000	0.000000	...	20	0.0	0.0	0	0	0.0	0.0	0	0	Benign
1	6	1	2	0	12	0	6	6	6.00000	0.000000	...	20	0.0	0.0	0	0	0.0	0.0	0	0	Benign
2	6	3	2	0	12	0	6	6	6.00000	0.000000	...	20	0.0	0.0	0	0	0.0	0.0	0	0	Benign
3	6	1	2	0	12	0	6	6	6.00000	0.000000	...	20	0.0	0.0	0	0	0.0	0.0	0	0	Benign
4	6	609	7	4	484	414	233	0	69.14286	111.967896	...	20	0.0	0.0	0	0	0.0	0.0	0	0	Benign

5 rows × 78 columns

In [8]:
```
drop_columns = [ # this list includes all spellings across CIC NIDS datasets
    "Flow ID",
    'Fwd Header Length.1',
    "Source IP", "Src IP",
    "Source Port", "Src Port",
    "Destination IP", "Dst IP",
    "Destination Port", "Dst Port",
    "Timestamp",
]
data.drop(columns=drop_columns, inplace=True, errors='ignore')
```

In [9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2044217 entries, 0 to 2044216
Data columns (total 78 columns):
 #   Column                    Dtype
---  ------                    -----
 0   Protocol                  int8
 1   Flow Duration             int32
 2   Total Fwd Packets         int32
 3   Total Backward Packets    int32
 4   Fwd Packets Length Total  int32
 5   Bwd Packets Length Total  int32
 6   Fwd Packet Length Max     int16
 7   Fwd Packet Length Min     int16
 8   Fwd Packet Length Mean    float32
 9   Fwd Packet Length Std     float32
 10  Bwd Packet Length Max     int16
 11  Bwd Packet Length Min     int16
 12  Bwd Packet Length Mean    float32
 13  Bwd Packet Length Std     float32
 14  Flow Bytes/s              float64
 15  Flow Packets/s            float64
 16  Flow IAT Mean             float32
 17  Flow IAT Std              float32
 18  Flow IAT Max              int32
 19  Flow IAT Min              int32
 20  Fwd IAT Total             int32
 21  Fwd IAT Mean              float32
 22  Fwd IAT Std               float32
 23  Fwd IAT Max               int32
 24  Fwd IAT Min               int32
 25  Bwd IAT Total             int32
 26  Bwd IAT Mean              float32
 27  Bwd IAT Std               float32
 28  Bwd IAT Max               int32
 29  Bwd IAT Min               int32
 30  Fwd PSH Flags             int8
 31  Bwd PSH Flags             int8
 32  Fwd URG Flags             int8
 33  Bwd URG Flags             int8
 34  Fwd Header Length         int64
 35  Bwd Header Length         int32
 36  Fwd Packets/s             float32
 37  Bwd Packets/s             float32
 38  Packet Length Min         int16
 39  Packet Length Max         int16
 40  Packet Length Mean        float32
 41  Packet Length Std         float32
 42  Packet Length Variance    float32
 43  FIN Flag Count            int8
 44  SYN Flag Count            int8
 45  RST Flag Count            int8
 46  PSH Flag Count            int8
 47  ACK Flag Count            int8
 48  URG Flag Count            int8
 49  CWE Flag Count            int8
 50  ECE Flag Count            int8
 51  Down/Up Ratio             int8
 52  Avg Packet Size           float32
 53  Avg Fwd Segment Size      float32
 54  Avg Bwd Segment Size      float32
 55  Fwd Avg Bytes/Bulk        int8
 56  Fwd Avg Packets/Bulk      int8
 57  Fwd Avg Bulk Rate         int8
 58  Bwd Avg Bytes/Bulk        int8
 59  Bwd Avg Packets/Bulk      int8
 60  Bwd Avg Bulk Rate         int8
 61  Subflow Fwd Packets       int32
 62  Subflow Fwd Bytes         int32
 63  Subflow Bwd Packets       int32
 64  Subflow Bwd Bytes         int32
 65  Init Fwd Win Bytes        int32
 66  Init Bwd Win Bytes        int32
 67  Fwd Act Data Packets      int32
 68  Fwd Seg Size Min          int32
 69  Active Mean               float32
 70  Active Std                float32
 71  Active Max                int32
 72  Active Min                int32
 73  Idle Mean                 float32
 74  Idle Std                  float32
 75  Idle Max                  int32
 76  Idle Min                  int32
 77  Label                     object
dtypes: float32(22), float64(2), int16(6), int32(26), int64(1), int8(20), object(1)
memory usage: 499.1+ MB
```

In [10]:
```
data['Label'].value_counts()
```

Out[10]:
```
Benign                      1707761
DoS Hulk                     172846
DDoS                         128014
DoS GoldenEye                 10286
FTP-Patator                    5931
DoS slowloris                  5385
DoS Slowhttptest               5228
SSH-Patator                    3219
PortScan                       1956
Web Attack ? Brute Force       1470
Bot                            1437
Web Attack ? XSS                652
Web Attack ? Sql Injection       21
Heartbleed                       11
Name: Label, dtype: int64
```

In [11]:
```
def change_label(df):
  df['Label'].replace(['DoS Hulk','DoS GoldenEye','DoS slowloris','DoS Slowhttptest'],'Dos',inplace=True)
  df['Label'].replace(['FTP-Patator','SSH-Patator','Infiltration','Heartbleed'],'Bot',inplace=True)
  df['Label'].replace(['Web Attack ? XSS','Web Attack ? Sql Injection'],'WebAttack',inplace=True)
  df['Label'].replace(['Web Attack ? Brute Force'],'BruteForce',inplace=True)
```

In [12]:
```
change_label(data)
```

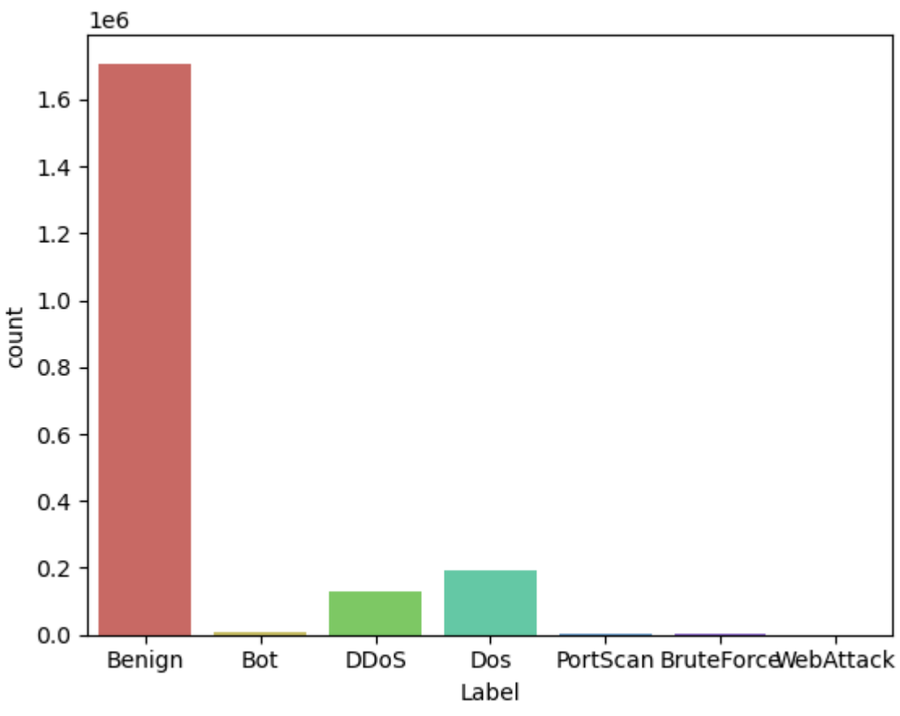
In [14]:
```
# distribution of attack classes
data['Label'].value_counts()
```

Out[14]:
```
Benign        1707761
Dos            193745
DDoS           128014
Bot             10598
PortScan         1956
BruteForce       1470
WebAttack         673
Name: Label, dtype: int64
```

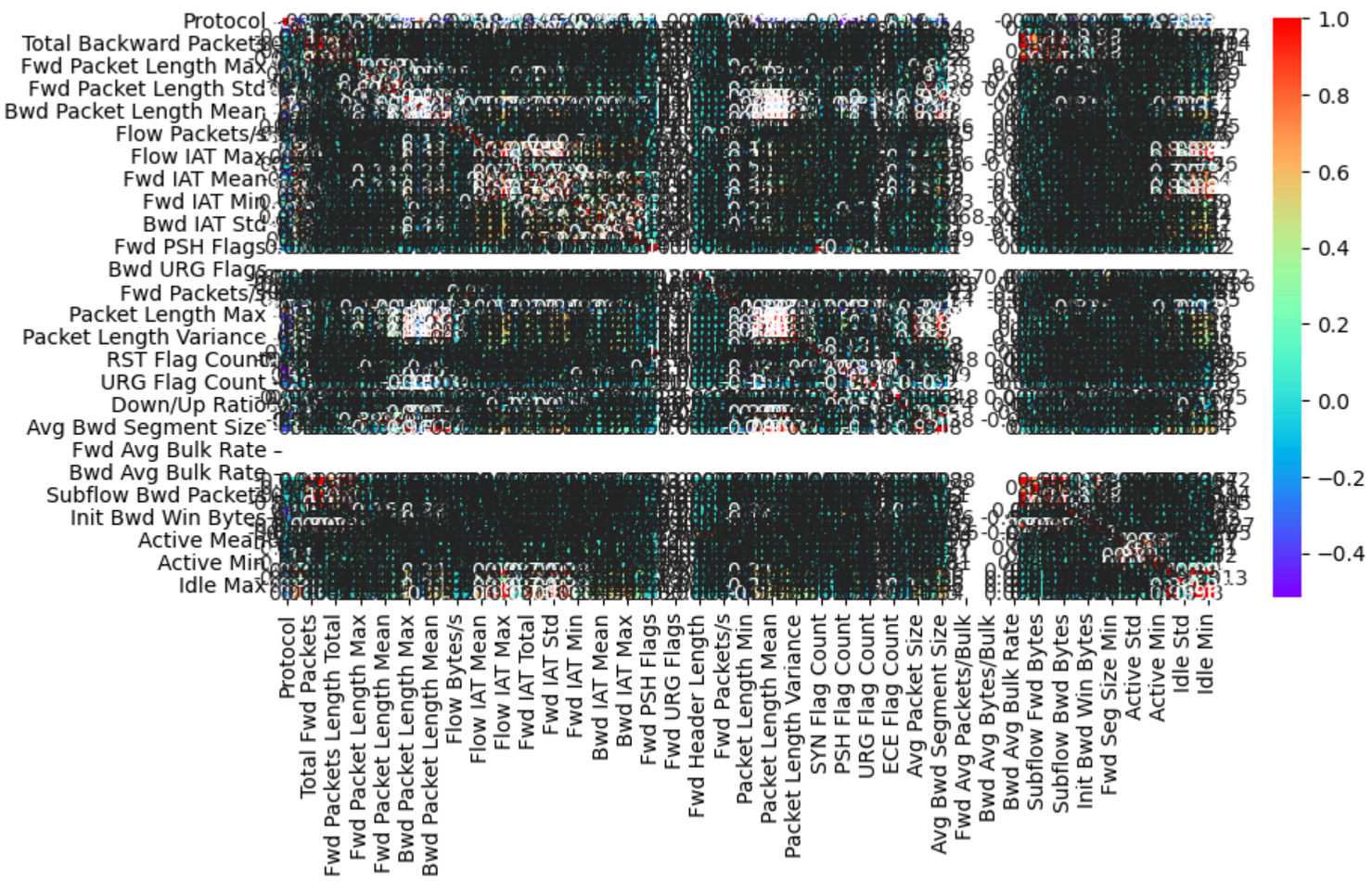
In [15]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2044217 entries, 0 to 2044216
Data columns (total 78 columns):
 #   Column                    Dtype
---  ------                    -----
 0   Protocol                  int8
 1   Flow Duration             int32
 2   Total Fwd Packets         int32
 3   Total Backward Packets    int32
 4   Fwd Packets Length Total  int32
 5   Bwd Packets Length Total  int32
 6   Fwd Packet Length Max     int16
 7   Fwd Packet Length Min     int16
 8   Fwd Packet Length Mean    float32
 9   Fwd Packet Length Std     float32
 10  Bwd Packet Length Max     int16
 11  Bwd Packet Length Min     int16
 12  Bwd Packet Length Mean    float32
 13  Bwd Packet Length Std     float32
 14  Flow Bytes/s              float64
 15  Flow Packets/s            float64
 16  Flow IAT Mean             float32
 17  Flow IAT Std              float32
 18  Flow IAT Max              int32
 19  Flow IAT Min              int32
 20  Fwd IAT Total             int32
 21  Fwd IAT Mean              float32
 22  Fwd IAT Std               float32
 23  Fwd IAT Max               int32
 24  Fwd IAT Min               int32
 25  Bwd IAT Total             int32
 26  Bwd IAT Mean              float32
 27  Bwd IAT Std               float32
 28  Bwd IAT Max               int32
 29  Bwd IAT Min               int32
 30  Fwd PSH Flags             int8
 31  Bwd PSH Flags             int8
 32  Fwd URG Flags             int8
 33  Bwd URG Flags             int8
 34  Fwd Header Length         int64
 35  Bwd Header Length         int32
 36  Fwd Packets/s             float32
 37  Bwd Packets/s             float32
 38  Packet Length Min         int16
 39  Packet Length Max         int16
 40  Packet Length Mean        float32
 41  Packet Length Std         float32
 42  Packet Length Variance    float32
 43  FIN Flag Count            int8
 44  SYN Flag Count            int8
 45  RST Flag Count            int8
 46  PSH Flag Count            int8
 47  ACK Flag Count            int8
 48  URG Flag Count            int8
 49  CWE Flag Count            int8
 50  ECE Flag Count            int8
 51  Down/Up Ratio             int8
 52  Avg Packet Size           float32
 53  Avg Fwd Segment Size      float32
 54  Avg Bwd Segment Size      float32
 55  Fwd Avg Bytes/Bulk        int8
 56  Fwd Avg Packets/Bulk      int8
 57  Fwd Avg Bulk Rate         int8
 58  Bwd Avg Bytes/Bulk        int8
 59  Bwd Avg Packets/Bulk      int8
 60  Bwd Avg Bulk Rate         int8
 61  Subflow Fwd Packets       int32
 62  Subflow Fwd Bytes         int32
 63  Subflow Bwd Packets       int32
 64  Subflow Bwd Bytes         int32
 65  Init Fwd Win Bytes        int32
 66  Init Bwd Win Bytes        int32
 67  Fwd Act Data Packets      int32
 68  Fwd Seg Size Min          int32
 69  Active Mean               float32
 70  Active Std                float32
 71  Active Max                int32
 72  Active Min                int32
 73  Idle Mean                 float32
 74  Idle Std                  float32
 75  Idle Max                  int32
 76  Idle Min                  int32
 77  Label                     object
dtypes: float32(22), float64(2), int16(6), int32(26), int64(1), int8(20), object(1)
memory usage: 499.1+ MB
```

In [16]:
```
sns.countplot(x='Label',data=data, palette='hls')
plt.show()
#plt.savefig('count_plot') mal: the nodule malignancy, 0: benign, 1: malignant
```



In [17]:
```
plt.figure(figsize = (10,5))
sns.heatmap(data.corr(), annot = True, cmap="rainbow")
plt.show()
```



In [19]:
```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
data['Label']= label_encoder.fit_transform(data['Label'])

data['Label'].unique()
```

Out[19]:
```
array([0, 4, 3, 1, 5, 2, 6])
```

In [20]:
```
X = data.drop(["Label"],axis =1)
y = data["Label"]
```

FS

In [21]:
```
from sklearn.feature_selection import SelectKBest, SelectPercentile, mutual_info_classif
```

In [22]:
```
selector = SelectPercentile(mutual_info_classif, percentile=25)
X_reduced = selector.fit_transform(X, y)
#X_reduced.shape
```

In [23]:
```
cols = selector.get_support(indices=True)
selected_columns = X.iloc[:,cols].columns.tolist()
selected_columns
```

Out[23]:
```
['Flow Duration',
 'Fwd Packets Length Total',
 'Bwd Packets Length Total',
 'Fwd Packet Length Max',
 'Fwd Packet Length Mean',
 'Flow IAT Max',
 'Fwd IAT Mean',
 'Fwd IAT Max',
 'Fwd Header Length',
 'Bwd Header Length',
 'Bwd Packets/s',
 'Packet Length Max',
 'Packet Length Mean',
 'Packet Length Std',
 'Packet Length Variance',
 'Avg Packet Size',
 'Avg Fwd Segment Size',
 'Subflow Fwd Bytes',
 'Init Bwd Win Bytes']
```

In [24]:
```
len(selected_columns)
```

Out[24]:
```
19
```

In [25]:
```
df = data[['Flow Duration',
           'Fwd Packets Length Total',
           'Bwd Packets Length Total',
           'Fwd Packet Length Max',
           'Fwd Packet Length Mean',
           'Flow IAT Max',
           'Fwd IAT Mean',
           'Fwd IAT Max',
           'Fwd Header Length',
           'Bwd Header Length',
           'Bwd Packets/s',
           'Packet Length Max',
           'Packet Length Mean',
           'Packet Length Std',
           'Packet Length Variance',
           'Avg Packet Size',
           'Avg Fwd Segment Size',
           'Subflow Fwd Bytes',
           'Init Bwd Win Bytes',
            'Label']]
```

In [26]:
```
df.to_csv('cic_processed.csv')
```

In [5]:
```
df = pd.read_csv('cic_processed.csv')
```

In [6]:
```
del df['Unnamed: 0']
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['Flow Duration', 'Fwd Packets Length Total', 'Bwd Packets Length Total',
       'Fwd Packet Length Max', 'Fwd Packet Length Mean', 'Flow IAT Max',
       'Fwd IAT Mean', 'Fwd IAT Max', 'Fwd Header Length', 'Bwd Header Length',
       'Bwd Packets/s', 'Packet Length Max', 'Packet Length Mean',
       'Packet Length Std', 'Packet Length Variance', 'Avg Packet Size',
       'Avg Fwd Segment Size', 'Subflow Fwd Bytes', 'Init Bwd Win Bytes',
       'Label'],
      dtype='object')
```

In [8]:
```
X = df[['Flow Duration', 'Fwd Packets Length Total', 'Bwd Packets Length Total',
       'Fwd Packet Length Max', 'Fwd Packet Length Mean', 'Flow IAT Max',
       'Fwd IAT Mean', 'Fwd IAT Max', 'Fwd Header Length', 'Bwd Header Length',
       'Bwd Packets/s', 'Packet Length Max', 'Packet Length Mean',
       'Packet Length Std', 'Packet Length Variance', 'Avg Packet Size',
       'Avg Fwd Segment Size', 'Subflow Fwd Bytes', 'Init Bwd Win Bytes']]
y = df["Label"]
```

In [9]:
```
from sklearn.model_selection import train_test_split
```

In [10]:
```
from sklearn.metrics import accuracy_score # for calculating accuracy of model
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

In [9]:
```
ML_Model = []
accuracy = []
precision = []
recall = []

f1score = []


#function to call for storing the results
def storeResults(model, a,b,c,d):
    ML_Model.append(model)
    accuracy.append(round(a, 3))
    precision.append(round(b, 3))
    recall.append(round(c, 3))
    f1score.append(round(d, 3))
```

In [17]:
```
# importing lime
import lime
from lime import lime_tabular
```

DNN

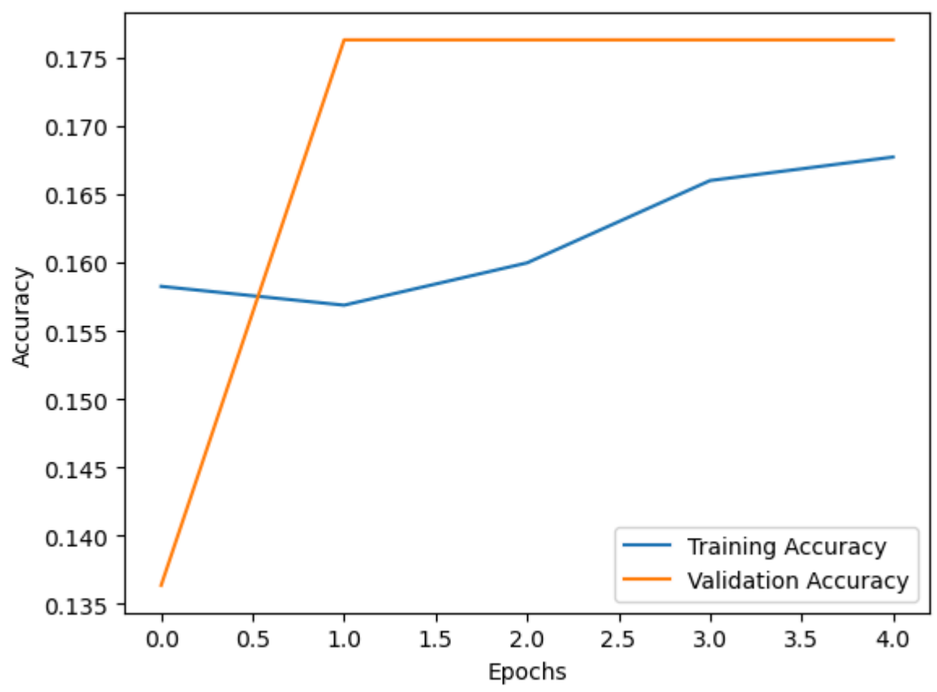
In [14]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

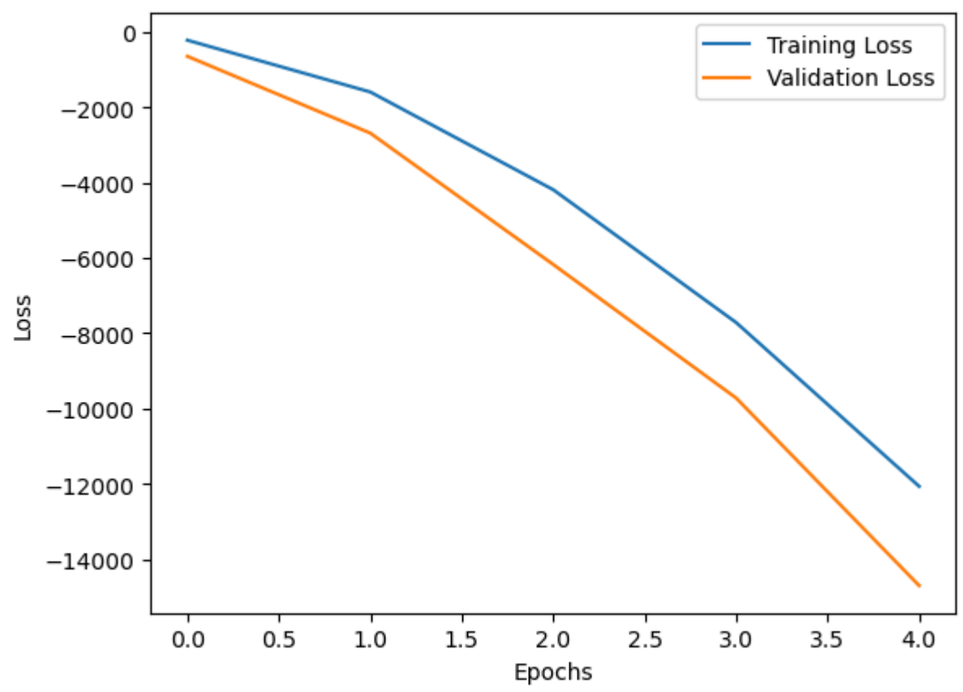
In [15]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)
# Scale the features using StandardScaler
scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)
```

In [16]:
```
# Build the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1024, activation='relu', input_dim=X_train.shape[1]),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=5, batch_size=4, validation_split=0.2)
```

```
Epoch 1/5
1452/1452 [==============================] - 12s 7ms/step - loss: -183.8006 - accuracy: 0.1507 - val_loss: -398.5809 - val_accuracy: 0.1763
Epoch 2/5
1452/1452 [==============================] - 9s 7ms/step - loss: -1451.5542 - accuracy: 0.1584 - val_loss: -1785.6713 - val_accuracy: 0.1763
Epoch 3/5
1452/1452 [==============================] - 9s 6ms/step - loss: -3825.5630 - accuracy: 0.1651 - val_loss: -3728.0132 - val_accuracy: 0.1763
Epoch 4/5
1452/1452 [==============================] - 10s 7ms/step - loss: -7149.9092 - accuracy: 0.1624 - val_loss: -6488.6616 - val_accuracy: 0.1763
Epoch 5/5
1452/1452 [==============================] - 9s 6ms/step - loss: -11145.7383 - accuracy: 0.1602 - val_loss: -8139.1475 - val_accuracy: 0.1763
```

In [13]:
```
import matplotlib.pyplot as plt

# Plotting akurasi
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting kerugian
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





In [14]:
```
# Evaluate the model on the testing set
predict_x=model.predict(X_test)
y_pred=np.argmax(predict_x,axis=1)

dl_acc = accuracy_score(y_pred, y_test)
dl_prec = precision_score(y_test,y_pred,average='weighted')
dl_rec = recall_score(y_test,y_pred,average='weighted')
dl_f1 = f1_score(y_test,y_pred,average='weighted')
```

```
152/152 [==============================] - 0s 2ms/step
```

In [15]:
```
storeResults('DNN',dl_acc,dl_prec,dl_rec,dl_f1)
```

In [25]:
```
lime_explainer = lime_tabular.LimeTabularExplainer(training_data=np.array(X_train), feature_names=X_train.columns,
            class_names=['0','1', '2', '3','4','5','6'], mode='classification')
```

In [28]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=model.predict, top_labels=6, num_features=19)
```

```
157/157 [==============================] - 0s 2ms/step
```

In [29]:
```
explanation.show_in_notebook()
```

ML

In [30]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 42)
```

AdaBoost

In [31]:
```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()

ada.fit(X_train, y_train)

y_pred    = ada.predict(X_test)

ada_acc = accuracy_score(y_pred, y_test)
ada_prec = precision_score(y_pred, y_test,average='weighted')
ada_rec = recall_score(y_pred, y_test,average='weighted')
ada_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [18]:
```
storeResults('AdaBoost',ada_acc,ada_prec,ada_rec,ada_f1)
```

LIME

In [32]:
```
lime_explainer = lime_tabular.LimeTabularExplainer(training_data=np.array(X_train), feature_names=X_train.columns,
            class_names=['0','1', '2', '3','4','5','6'], mode='classification')
```

In [33]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=ada.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

LightGBM

In [35]:
```
# build the lightgbm model
import lightgbm as lgb
clf = lgb.LGBMClassifier()
clf.fit(X_train, y_train)

y_pred    = clf.predict(X_test)

lgb_acc = accuracy_score(y_pred, y_test)
lgb_prec = precision_score(y_pred, y_test,average='weighted')
lgb_rec = recall_score(y_pred, y_test,average='weighted')
lgb_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000892 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 4463
[LightGBM] [Info] Number of data points in the train set: 9679, number of used features: 19
[LightGBM] [Info] Start training from score -1.803712
[LightGBM] [Info] Start training from score -1.784451
[LightGBM] [Info] Start training from score -2.111247
[LightGBM] [Info] Start training from score -1.809374
[LightGBM] [Info] Start training from score -1.807483
[LightGBM] [Info] Start training from score -1.816338
[LightGBM] [Info] Start training from score -2.884295
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

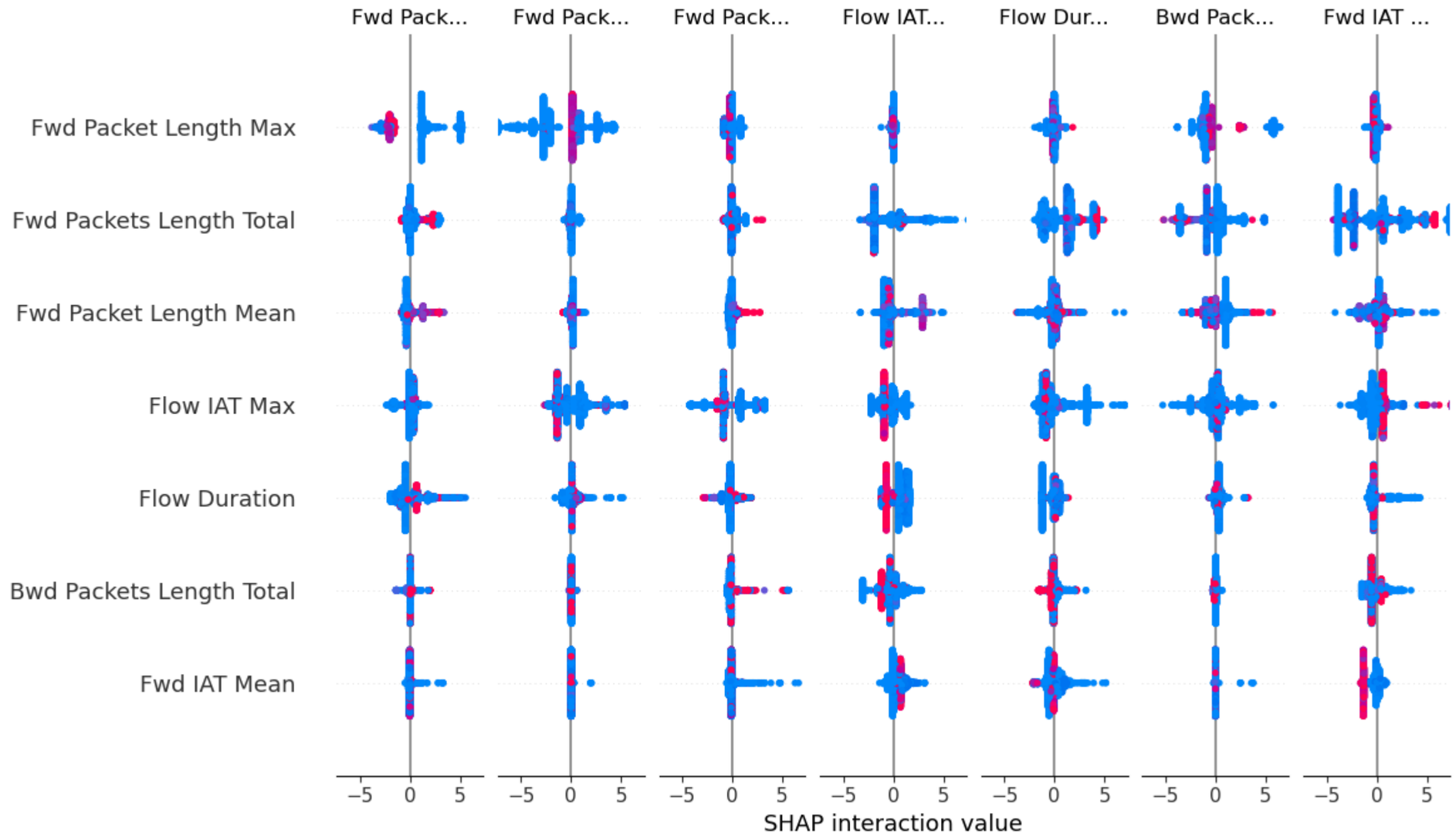
In [21]:
```
storeResults('LightGBM',lgb_acc,lgb_prec,lgb_rec,lgb_f1)
```

LIME

In [36]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=clf.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

SHAP

In [38]:
```
import shap
explainer = shap.Explainer(clf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



MLP

In [39]:
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(random_state=1, max_iter=300)

mlp.fit(X_train, y_train)

y_pred    = mlp.predict(X_test)

mlp_acc = accuracy_score(y_pred, y_test)
mlp_prec = precision_score(y_pred, y_test,average='weighted')
mlp_rec = recall_score(y_pred, y_test,average='weighted')
mlp_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [23]:
```
storeResults('MLP',mlp_acc,mlp_prec,mlp_rec,mlp_f1)
```

LIME

In [40]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=mlp.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

KNN

In [41]:
```
from sklearn.neighbors import KNeighborsClassifier
knn =  KNeighborsClassifier()
knn.fit(X_train, y_train)

y_pred    = knn.predict(X_test)

knn_acc = accuracy_score(y_pred, y_test)
knn_prec = precision_score(y_pred, y_test,average='weighted')
knn_rec = recall_score(y_pred, y_test,average='weighted')
knn_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [25]:
```
storeResults('KNN',knn_acc,knn_prec,knn_rec,knn_f1)
```

LIME

In [42]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=knn.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

Random Forest

In [43]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(criterion='entropy',max_features='log2',max_depth=20,n_estimators=600,min_samples_leaf=2)
rf.fit(X_train, y_train)

y_pred    = rf.predict(X_test)

rf_acc = accuracy_score(y_pred, y_test)
rf_prec = precision_score(y_pred, y_test,average='weighted')
rf_rec = recall_score(y_pred, y_test,average='weighted')
rf_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [27]:
```
storeResults('RandomForest',rf_acc,rf_prec,rf_rec,rf_f1)
```

LIME

In [44]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=rf.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

SVM

In [45]:
```
from sklearn import svm
svc = svm.SVC(decision_function_shape='ovo',probability=True)
svc.fit(X_train, y_train)

y_pred    = svc.predict(X_test)

svc_acc = accuracy_score(y_pred, y_test)
svc_prec = precision_score(y_pred, y_test,average='weighted')
svc_rec = recall_score(y_pred, y_test,average='weighted')
svc_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [29]:
```
storeResults('SVM',svc_acc,svc_prec,svc_rec,svc_f1)
```

LIME

In [48]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=svc.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

Extension

In [47]:
```
from sklearn.ensemble import VotingClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

brf = BaggingClassifier(RandomForestClassifier())

bdt = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), algorithm="SAMME", n_estimators=200
)

model = VotingClassifier(estimators= [('BoostDT', bdt),('BagRF', brf)], voting='soft')

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

ext_acc = accuracy_score(y_pred, y_test)
ext_prec = precision_score(y_pred, y_test,average='weighted')
ext_rec = recall_score(y_pred, y_test,average='weighted')
ext_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [32]:
```
storeResults('Extension',ext_acc,ext_prec,ext_rec,ext_f1)
```

In [49]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=model.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

Comparison

In [33]:
```
#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                       'Precision': precision,
                       'Recall'   : recall,
                       'F1_score' : f1score
                      })
```

In [34]:
```
result
```

Out[34]:

	ML Model	Accuracy	Precision	Recall	F1_score
0	DNN	0.166	0.028	0.166	0.047
1	AdaBoost	0.406	0.660	0.406	0.466
2	LightGBM	0.929	0.939	0.929	0.933
3	MLP	0.598	0.717	0.598	0.599
4	KNN	0.858	0.866	0.858	0.861
5	RandomForest	0.929	0.943	0.929	0.934
6	SVM	0.592	0.773	0.592	0.619
7	Extension	0.979	0.981	0.979	0.980

Modelling

In [35]:
```
import joblib
filename = 'models/model_cic.sav'
joblib.dump(model, filename)
```

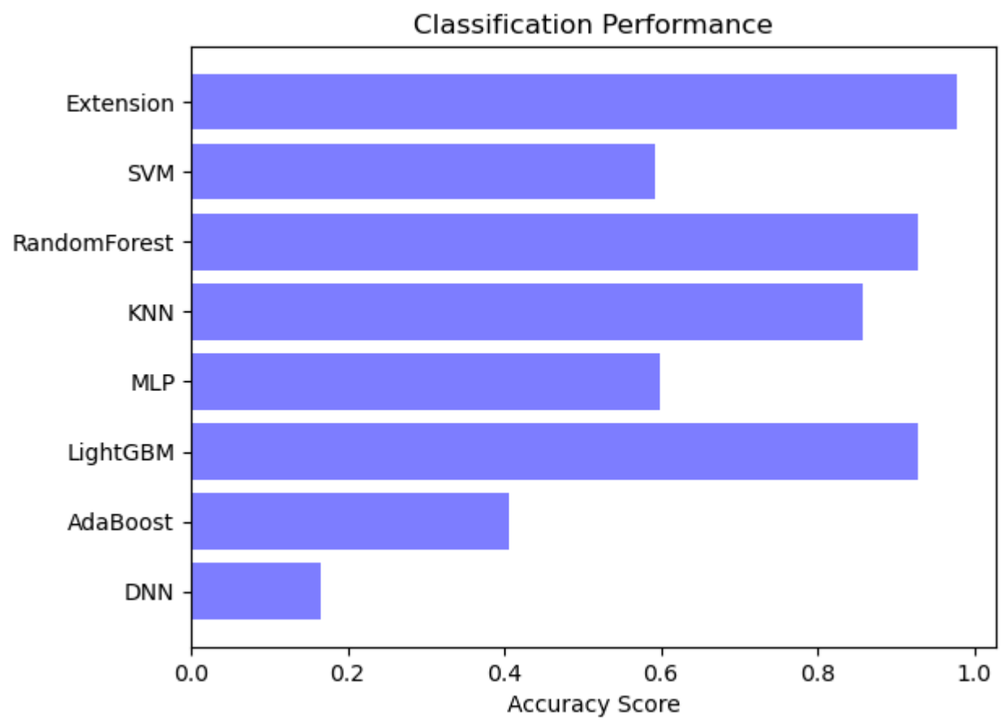
Out[35]:
```
['models/model_cic.sav']
```

Graph

In [37]:
```
classifier = ML_Model
y_pos = np.arange(len(classifier))
```

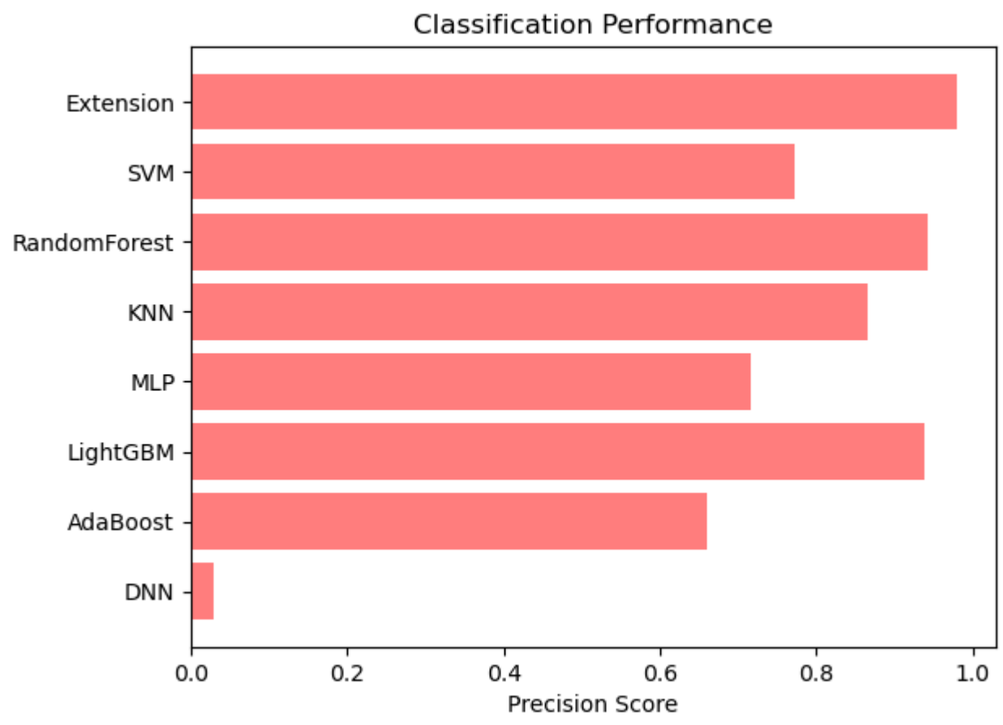
Accuracy

In [38]:
```
import matplotlib.pyplot as plt2
plt2.barh(y_pos, accuracy, align='center', alpha=0.5,color='blue')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Accuracy Score')
plt2.title('Classification Performance')
plt2.show()
```



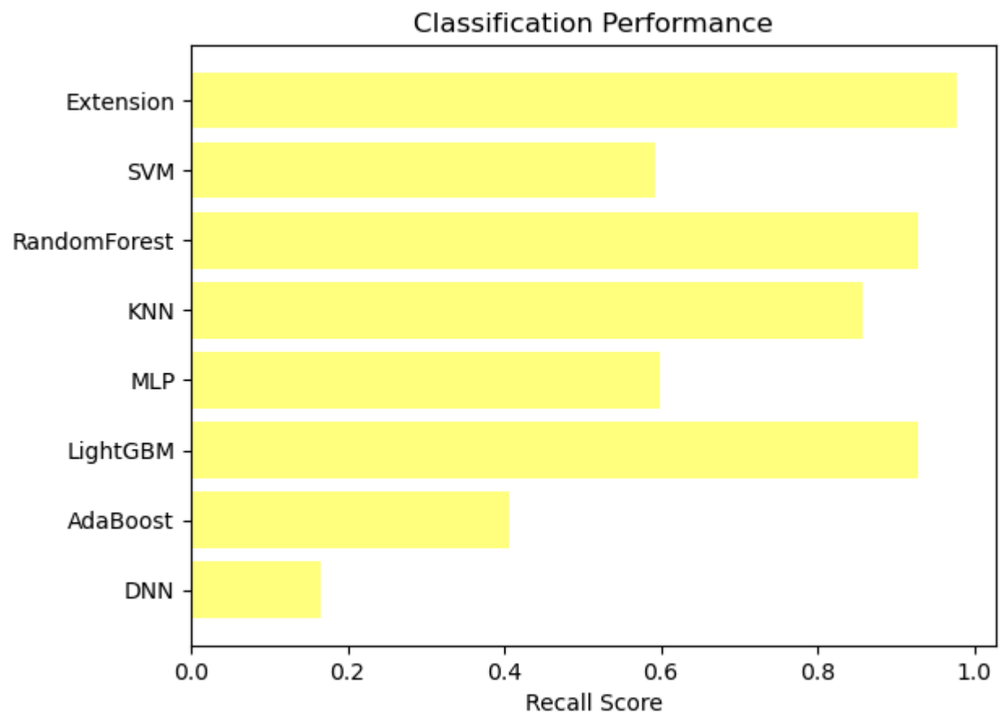
Precision

In [39]:
```
plt2.barh(y_pos, precision, align='center', alpha=0.5,color='red')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Precision Score')
plt2.title('Classification Performance')
plt2.show()
```



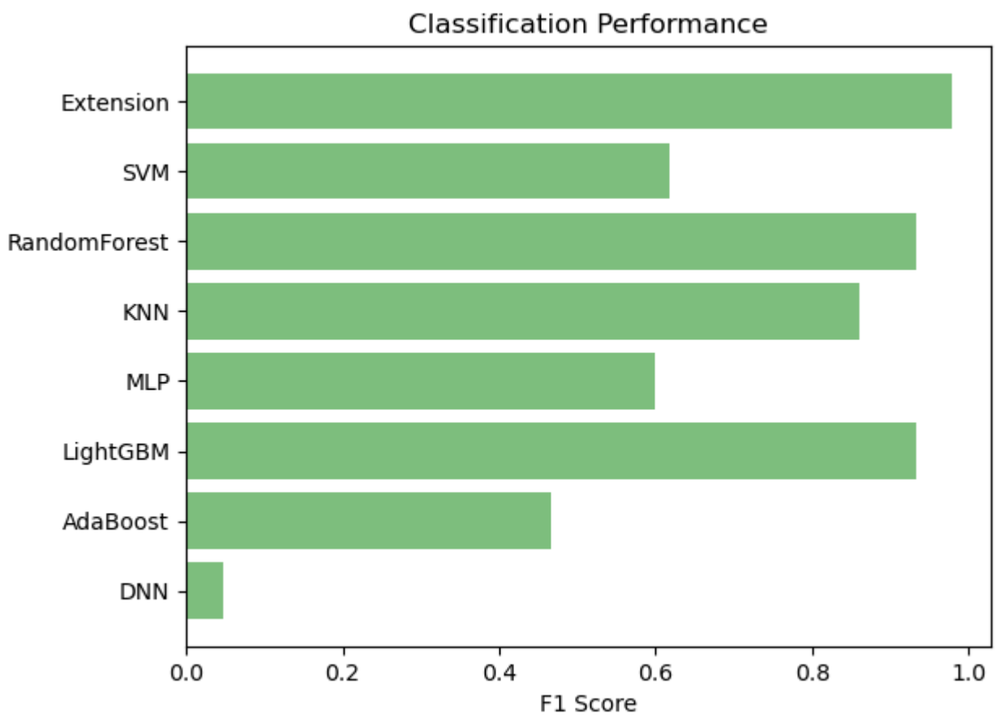
Recall

In [40]:
```
plt2.barh(y_pos, recall, align='center', alpha=0.5,color='yellow')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Recall Score')
plt2.title('Classification Performance')
plt2.show()
```



F1 Score

In [41]:
```
plt2.barh(y_pos, f1score, align='center', alpha=0.5,color='green')
plt2.yticks(y_pos, classifier)
plt2.xlabel('F1 Score')
plt2.title('Classification Performance')
plt2.show()
```



In [ ]: