In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
data = pd.read_parquet('OwnData/data.parquet')
```

In [6]:
```
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=['category']).columns.tolist(),'\n')

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

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

0 duplicate entries have been found in the dataset

All duplicates have been removed

Categorical columns: ['Label']
```

Out[6]:

	Protocol	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	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	22545	22	20	336.0	0.0	32.0	0.0	15.272727	9.207563	...	32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	TCPFlood-Evasive
1	6	1203699	7	5	413.0	11192.0	413.0	0.0	59.000000	156.099335	...	32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Hulk-NoDefense
2	6	776558	8	6	365.0	11192.0	365.0	0.0	45.625000	129.046982	...	32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Hulk-NoDefense
3	6	239234	11	6	225.0	4256.0	77.0	0.0	20.454546	35.041016	...	32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Benign
4	6	432672	8	6	376.0	11192.0	376.0	0.0	47.000000	132.936081	...	32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Hulk-NoDefense

5 rows × 78 columns

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

Out[7]:
```
Index(['Protocol', 'Flow Duration', 'Total Fwd Packet', 'Total Bwd packets',
       'Total Length of Fwd Packet', 'Total Length of Bwd Packet',
       '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', 'CWR Flag Count', 'ECE Flag Count', 'Down/Up Ratio',
       'Avg Packet Size', 'Fwd Segment Size Avg', 'Bwd Segment Size Avg',
       'Fwd Bytes/Bulk Avg', 'Fwd Packet/Bulk Avg', 'Fwd Bulk Rate Avg',
       'Bwd Bytes/Bulk Avg', 'Bwd Packet/Bulk Avg', 'Bwd Bulk Rate Avg',
       'Subflow Fwd Packets', 'Subflow Fwd Bytes', 'Subflow Bwd Packets',
       'Subflow Bwd Bytes', 'FWD Init Win Bytes', 'Bwd Init 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 [8]:
```
data['Label'].value_counts()
```

Out[8]:
```
Hulk-Reqtimeout            128346
Hulk-NoDefense             127671
Hulk-Security2             121956
Hulk-Evasive               113552
Benign                      45708
TCPFlood-Reqtimeout         45405
TCPFlood-Evasive            45390
TCPFlood-Security2          44547
TCPFlood-NoDefense          43762
Slowloris-Reqtimeout         1017
Slowhttptest-Security2       1003
Slowhttptest-Evasive         1002
Slowhttptest-NoDefense        998
Slowhttptest-Reqtimeout       852
Slowloris-Evasive             267
Slowloris-NoDefense           267
Slowloris-Security2           267
Name: Label, dtype: int64
```

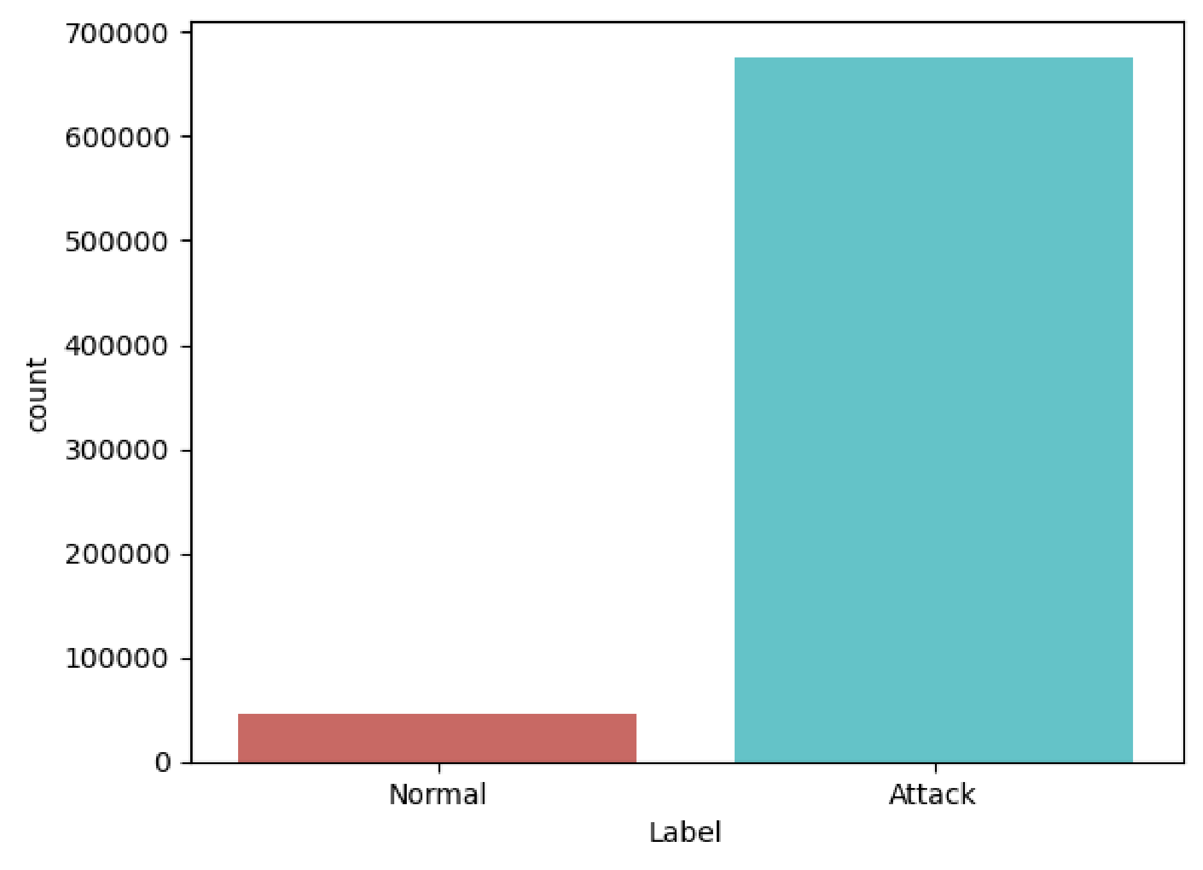
In [9]:
```
# changing attack labels to their respective attack class
def change_label(df):
    df['Label'].replace(['Hulk-Reqtimeout','Hulk-NoDefense','Hulk-Security2','Hulk-Evasive','TCPFlood-Reqtimeout','TCPFlood-Evasive','TCPFlood-Security2','TCPFlood-NoDefense','Slowloris-Reqtimeout','Slowhttptest-Security2',
                         'Slowhttptest-Evasive','Slowhttptest-NoDefense','Slowhttptest-Reqtimeout','Slowloris-Evasive','Slowloris-NoDefense','Slowloris-Security2'],'Attack',inplace=True)
    df['Label'].replace(['Benign'],'Normal',inplace=True)
```

In [10]:
```
change_label(data)
```

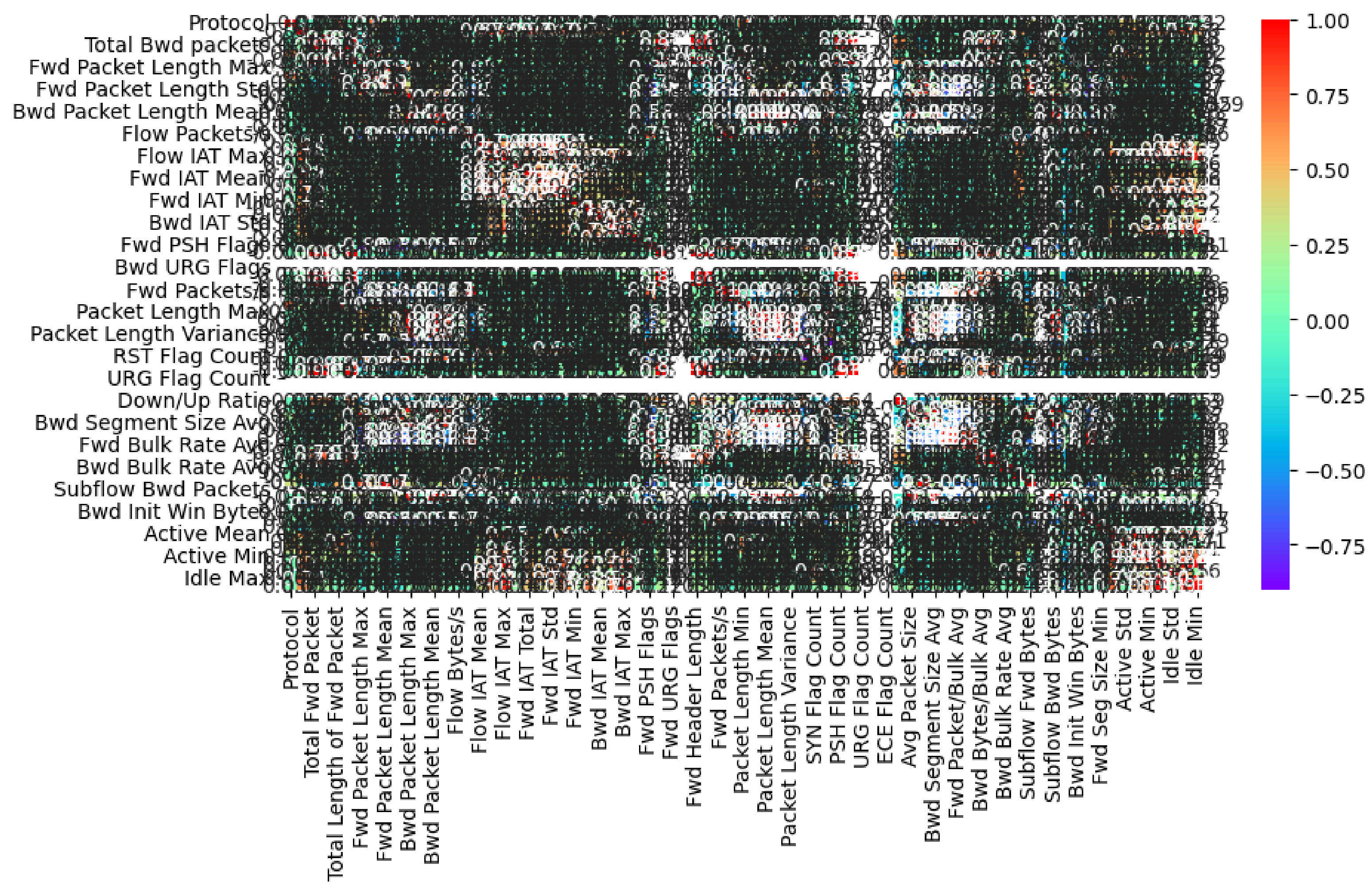
In [11]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 722010 entries, 0 to 722009
Data columns (total 78 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   Protocol                    722010 non-null  int8
 1   Flow Duration               722010 non-null  int32
 2   Total Fwd Packet            722010 non-null  int16
 3   Total Bwd packets           722010 non-null  int16
 4   Total Length of Fwd Packet  722010 non-null  float32
 5   Total Length of Bwd Packet  722010 non-null  float32
 6   Fwd Packet Length Max       722010 non-null  float32
 7   Fwd Packet Length Min       722010 non-null  float32
 8   Fwd Packet Length Mean      722010 non-null  float32
 9   Fwd Packet Length Std       722010 non-null  float32
 10  Bwd Packet Length Max       722010 non-null  float32
 11  Bwd Packet Length Min       722010 non-null  float32
 12  Bwd Packet Length Mean      722010 non-null  float32
 13  Bwd Packet Length Std       722010 non-null  float32
 14  Flow Bytes/s                722010 non-null  float32
 15  Flow Packets/s              722010 non-null  float32
 16  Flow IAT Mean               722010 non-null  float32
 17  Flow IAT Std                722010 non-null  float32
 18  Flow IAT Max                722010 non-null  float32
 19  Flow IAT Min                722010 non-null  float32
 20  Fwd IAT Total               722010 non-null  float32
 21  Fwd IAT Mean                722010 non-null  float32
 22  Fwd IAT Std                 722010 non-null  float32
 23  Fwd IAT Max                 722010 non-null  float32
 24  Fwd IAT Min                 722010 non-null  float32
 25  Bwd IAT Total               722010 non-null  float32
 26  Bwd IAT Mean                722010 non-null  float32
 27  Bwd IAT Std                 722010 non-null  float32
 28  Bwd IAT Max                 722010 non-null  float32
 29  Bwd IAT Min                 722010 non-null  float32
 30  Fwd PSH Flags               722010 non-null  int8
 31  Bwd PSH Flags               722010 non-null  int16
 32  Fwd URG Flags               722010 non-null  int8
 33  Bwd URG Flags               722010 non-null  int8
 34  Fwd Header Length           722010 non-null  int32
 35  Bwd Header Length           722010 non-null  int32
 36  Fwd Packets/s               722010 non-null  float32
 37  Bwd Packets/s               722010 non-null  float32
 38  Packet Length Min           722010 non-null  float32
 39  Packet Length Max           722010 non-null  float32
 40  Packet Length Mean          722010 non-null  float32
 41  Packet Length Std           722010 non-null  float32
 42  Packet Length Variance      722010 non-null  float32
 43  FIN Flag Count              722010 non-null  int8
 44  SYN Flag Count              722010 non-null  int8
 45  RST Flag Count              722010 non-null  int8
 46  PSH Flag Count              722010 non-null  int16
 47  ACK Flag Count              722010 non-null  int16
 48  URG Flag Count              722010 non-null  int8
 49  CWR Flag Count              722010 non-null  int8
 50  ECE Flag Count              722010 non-null  int8
 51  Down/Up Ratio               722010 non-null  float32
 52  Avg Packet Size             722010 non-null  float32
 53  Fwd Segment Size Avg        722010 non-null  float32
 54  Bwd Segment Size Avg        722010 non-null  float32
 55  Fwd Bytes/Bulk Avg          722010 non-null  int32
 56  Fwd Packet/Bulk Avg         722010 non-null  int8
 57  Fwd Bulk Rate Avg           722010 non-null  int32
 58  Bwd Bytes/Bulk Avg          722010 non-null  int32
 59  Bwd Packet/Bulk Avg         722010 non-null  int16
 60  Bwd Bulk Rate Avg           722010 non-null  int32
 61  Subflow Fwd Packets         722010 non-null  int8
 62  Subflow Fwd Bytes           722010 non-null  int16
 63  Subflow Bwd Packets         722010 non-null  int8
 64  Subflow Bwd Bytes           722010 non-null  int16
 65  FWD Init Win Bytes          722010 non-null  int16
 66  Bwd Init Win Bytes          722010 non-null  int16
 67  Fwd Act Data Packets        722010 non-null  int8
 68  Fwd Seg Size Min            722010 non-null  int8
 69  Active Mean                 722010 non-null  float32
 70  Active Std                  722010 non-null  float32
 71  Active Max                  722010 non-null  float32
 72  Active Min                  722010 non-null  float32
 73  Idle Mean                   722010 non-null  float32
 74  Idle Std                    722010 non-null  float32
 75  Idle Max                    722010 non-null  float32
 76  Idle Min                    722010 non-null  float32
 77  Label                       722010 non-null  category
dtypes: category(1), float32(45), int16(10), int32(7), int8(15)
memory usage: 168.0 MB
```

In [12]:
```
sns.countplot(x='Label',data=data, palette='hls')
plt.show()
#plt.savefig('count_plot') mal: the nodule malignancy, 0: benign, 1: malignant
```



In [13]:
```
plt.figure(figsize = (10,5))
sns.heatmap(data.corr(), annot = True, cmap="rainbow")
plt.show()
```



In [14]:
```
data['Label'].value_counts()
```

Out[14]:
```
Attack    676302
Normal     45708
Name: Label, dtype: int64
```

In [16]:
```
# 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'])
```

In [17]:
```
X = data.drop(["Label"],axis =1)
y = data["Label"]
```

FS

In [18]:
```
from sklearn.feature_selection import SelectKBest, SelectPercentile, mutual_info_classif
```

In [19]:
```
selector = SelectPercentile(mutual_info_classif, percentile=15)
X_reduced = selector.fit_transform(X, y)
#X_reduced.shape
```

In [20]:
```
cols = selector.get_support(indices=True)
selected_columns = X.iloc[:,cols].columns.tolist()
selected_columns
```

Out[20]:
```
['Total Length of Bwd Packet',
 'Fwd Packet Length Max',
 'Fwd Packet Length Mean',
 'Fwd Packet Length Std',
 'Bwd Packet Length Mean',
 'Fwd IAT Min',
 'Bwd PSH Flags',
 'Packet Length Mean',
 'Avg Packet Size',
 'Fwd Segment Size Avg',
 'Bwd Segment Size Avg',
 'FWD Init Win Bytes']
```

In [21]:
```
len(selected_columns)
```

Out[21]:
```
12
```

In [22]:
```
df = data[['Total Length of Bwd Packet',
 'Fwd Packet Length Max',
 'Fwd Packet Length Mean',
 'Fwd Packet Length Std',
 'Bwd Packet Length Mean',
 'Fwd IAT Min',
 'Bwd PSH Flags',
 'Packet Length Mean',
 'Avg Packet Size',
 'Fwd Segment Size Avg',
 'Bwd Segment Size Avg',
 'FWD Init Win Bytes','Label']]
```

In [23]:
```
df.columns
```

Out[23]:
```
Index(['Total Length of Bwd Packet', 'Fwd Packet Length Max',
       'Fwd Packet Length Mean', 'Fwd Packet Length Std',
       'Bwd Packet Length Mean', 'Fwd IAT Min', 'Bwd PSH Flags',
       'Packet Length Mean', 'Avg Packet Size', 'Fwd Segment Size Avg',
       'Bwd Segment Size Avg', 'FWD Init Win Bytes', 'Label'],
      dtype='object')
```

In [24]:
```
X = df.drop(["Label"],axis =1)
y = df["Label"]
```

In [25]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 42)
#X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

In [26]:
```
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 [27]:
```
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))
```

BernoulliNB

In [28]:
```
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB(alpha=1.0, binarize=0.0, fit_prior=True, class_prior=None)

bnb.fit(X_train, y_train)

y_pred = bnb.predict(X_test)

bnb_acc = accuracy_score(y_pred, y_test)
bnb_prec = precision_score(y_pred, y_test,average='weighted')
bnb_rec = recall_score(y_pred, y_test,average='weighted')
bnb_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [29]:
```
storeResults('BernoulliNB',bnb_acc,bnb_prec,bnb_rec,bnb_f1)
```

Passive Aggressive

In [30]:
```
from sklearn.linear_model import PassiveAggressiveClassifier

pa = PassiveAggressiveClassifier(C=1.0, fit_intercept=True, max_iter=1000, tol=0.001, early_stopping=False,
                                 validation_fraction=0.1, n_iter_no_change=5, shuffle=True, verbose=0,
                                 loss='hinge', n_jobs=None, random_state=None, warm_start=False,
                                 class_weight=None, average=False)

pa.fit(X_train, y_train)

y_pred = pa.predict(X_test)

pa_acc = accuracy_score(y_pred, y_test)
pa_prec = precision_score(y_pred, y_test,average='weighted')
pa_rec = recall_score(y_pred, y_test,average='weighted')
pa_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [31]:
```
storeResults('PassiveAggressive',pa_acc,pa_prec,pa_rec,pa_f1)
```

SGDClassifier

In [32]:
```
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
                   max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None,
                   random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False,
                   validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False)

sgd.fit(X_train, y_train)

y_pred = sgd.predict(X_test)

sgd_acc = accuracy_score(y_pred, y_test)
sgd_prec = precision_score(y_pred, y_test,average='weighted')
sgd_rec = recall_score(y_pred, y_test,average='weighted')
sgd_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [33]:
```
storeResults('SGDClassifier',sgd_acc,sgd_prec,sgd_rec,sgd_f1)
```

MLP Classifier

In [34]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto',
                    learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True,
                    random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True,
                    early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
                    n_iter_no_change=10, max_fun=15000)

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 [35]:
```
storeResults('MLPClassifier',mlp_acc,mlp_prec,mlp_rec,mlp_f1)
```

Ensemble

In [36]:
```
from sklearn.ensemble import VotingClassifier

eclf1 = VotingClassifier(estimators=[('BNB', bnb),('PA', pa),('SGD', sgd),('MLP', mlp)], voting='hard')


eclf1.fit(X_train, y_train)

y_pred = eclf1.predict(X_test)

stac_acc = accuracy_score(y_pred, y_test)
stac_prec = precision_score(y_pred, y_test,average='weighted')
stac_rec = recall_score(y_pred, y_test,average='weighted')
stac_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [37]:
```
storeResults('Ensemble',stac_acc,stac_prec,stac_rec,stac_f1)
```

Extension

In [38]:
```
from sklearn.ensemble import VotingClassifier, AdaBoostClassifier, RandomForestClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

brf = BaggingClassifier(RandomForestClassifier(),n_estimators=10, random_state=0,max_samples=1.0,max_features=1.0)

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

ext = VotingClassifier(estimators=[('BoostDT', bdt),('BagRF', brf)], voting='soft')
ext.fit(X_train, y_train)

y_pred    = ext.predict(X_test)

ml_acc = accuracy_score(y_pred, y_test)
ml_prec = precision_score(y_pred, y_test,average='weighted')
ml_rec = recall_score(y_pred, y_test,average='weighted')
ml_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [41]:
```
storeResults('Extension',ml_acc,ml_prec,ml_rec,ml_f1)
```

Comparison

In [42]:
```
#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                       'Precision': precision,
                       'Recall'    : recall,
                       'F1_score' : f1score
                      })
```

In [43]:
```
result
```

Out[43]:

	ML Model	Accuracy	Precision	Recall	F1_score
0	BernoulliNB	0.618	0.887	0.618	0.676
1	PassiveAggressive	0.991	0.991	0.991	0.991
2	SGDClassifier	0.980	0.980	0.980	0.980
3	MLPClassifier	0.989	0.989	0.989	0.989
4	Ensemble	0.992	0.992	0.992	0.992
5	Extension	1.000	1.000	1.000	1.000

Modelling

In [45]:
```
import joblib
filename = 'models/model_owndata.sav'
joblib.dump(ext, filename)
```

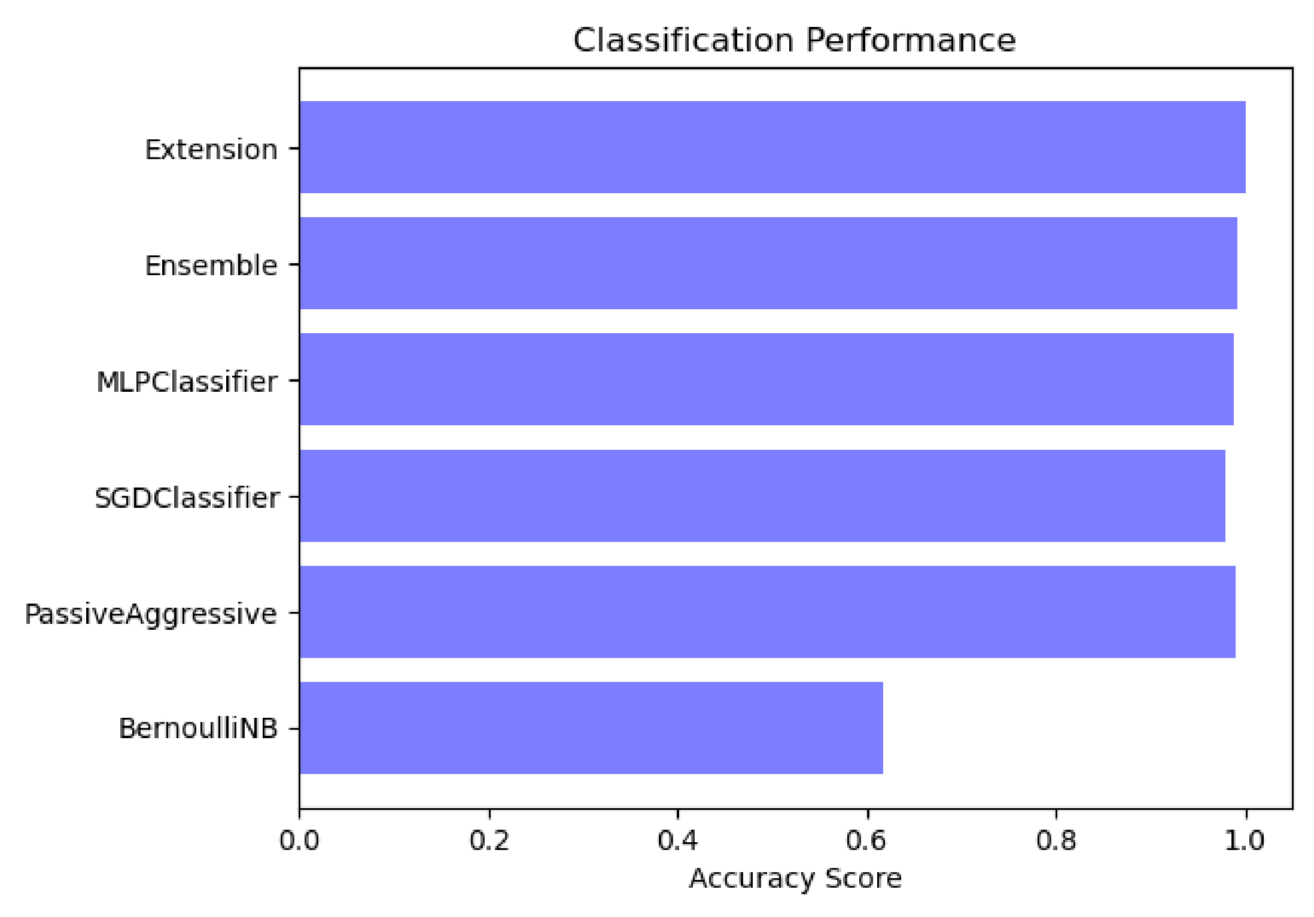
Out[45]:
```
['models/model_owndata.sav']
```

Graph

In [46]:
```
classifier = ML_Model
y_pos = np.arange(len(classifier))
```

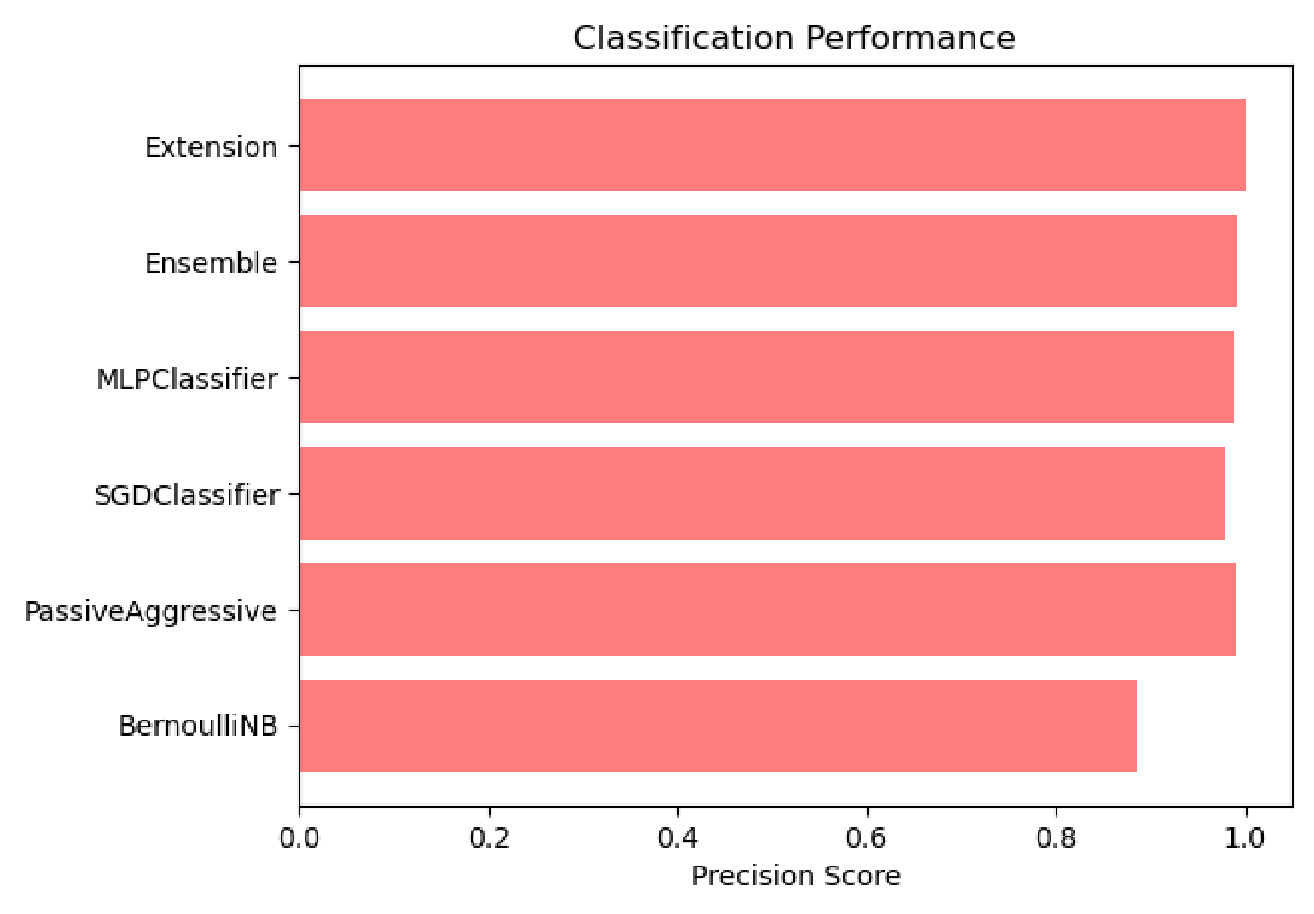
Accuracy

In [47]:
```
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()
```



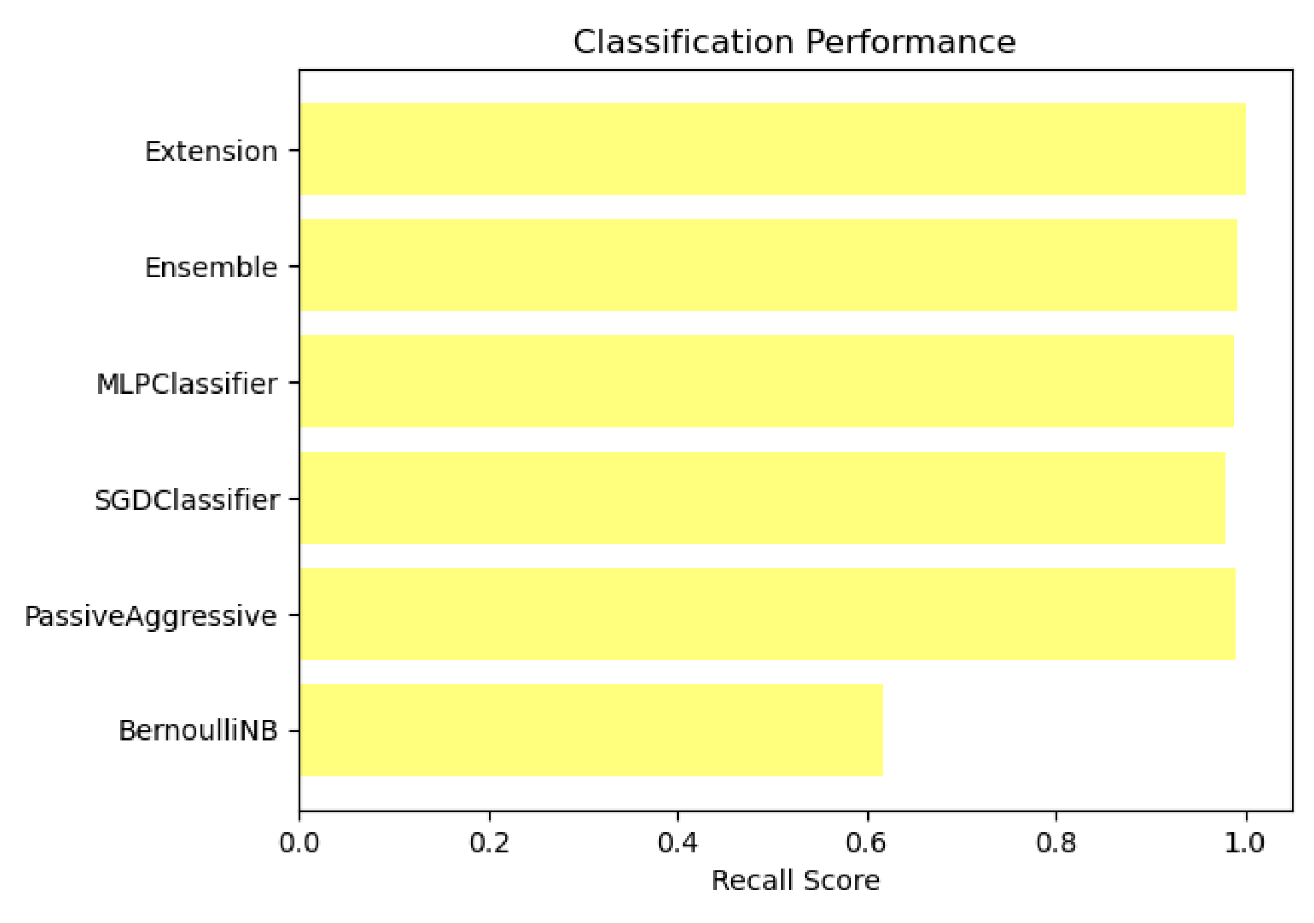
Precision

In [48]:
```
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()
```



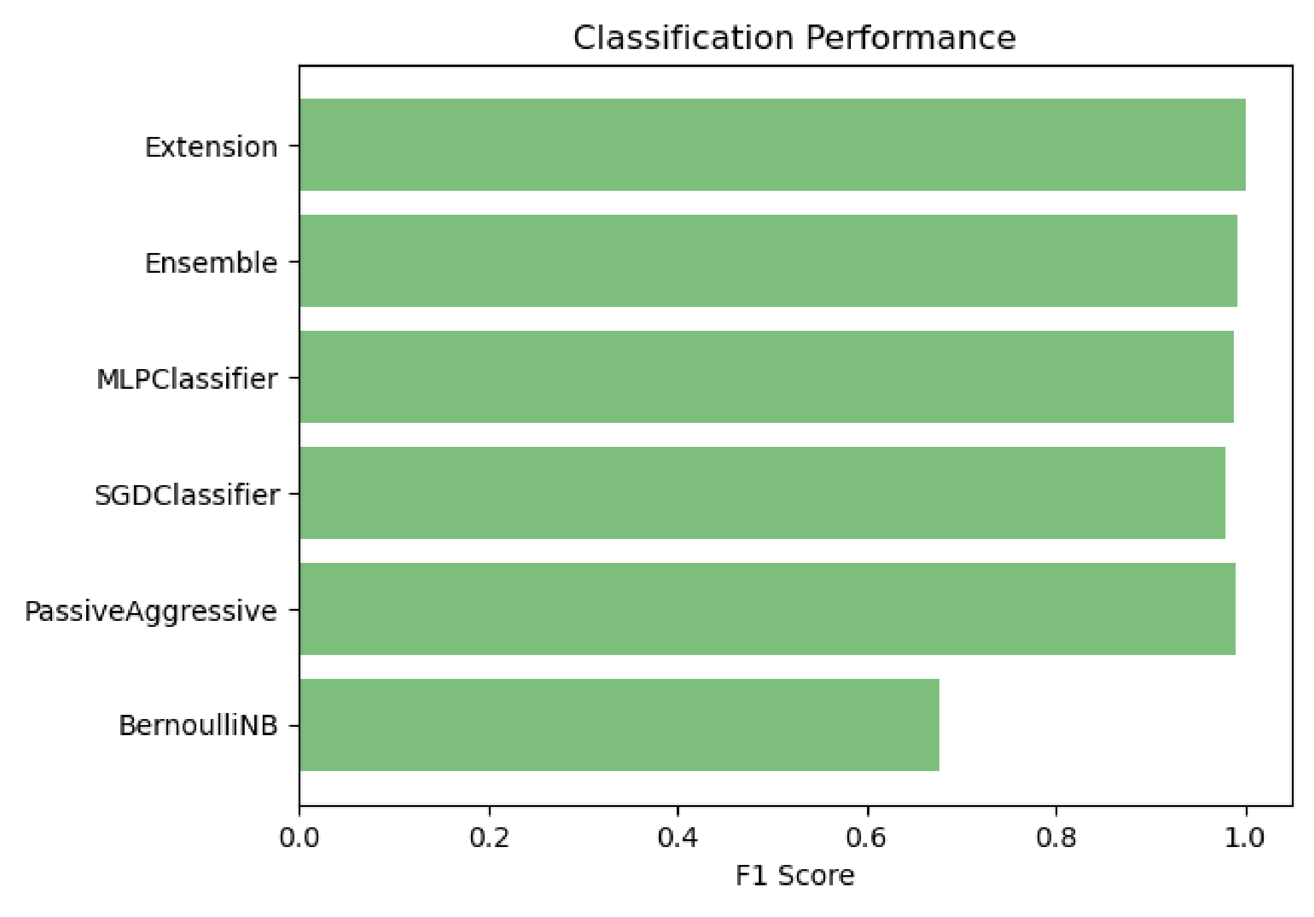
Recall

In [49]:
```
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 [50]:
```
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 [ ]: