Machine Learning for the Detection of Network Attacks

Analyse the machine learning algorithms on the [CICIDS 2017 Dataset] for clasification of network attacks. (https://www.unb.ca/cic/datasets/ids-2017.html):

- Support Vector Machine (SVM)
- Decision Tree
- Naive Bayes
- K Means Clustering
- K Nearest Neighbours

In [1]:

```
# from google.colab import drive
# drive.mount('/content/drive')
```

Import required libraries.

In [2]:

```
import glob
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sn
import time

from numpy import array

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler

from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import mutual_info_classif

from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.metrics import completeness_score, homogeneity_score, v_measure_score
```

```
from sklearn.model_selection import train_test_split
```

Loading the dataset

The implemented attacks include Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS.

Datasets is available in 8 different csv files.

- Monday-WorkingHours.pcap_ISCX.csv
- Tuesday-WorkingHours.pcap_ISCX.csv
- Wednesday-workingHours.pcap_ISCX.csv
- Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv
- Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv
- Friday-WorkingHours-Morning.pcap_ISCX.csv
- Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv
- Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv

8 different csv files of cicids dataset needs to be concatenated into a single csv file.

In [3]:

```
# # path to the all 8 files of CICIDS dataset.
# path = './datasets'
# all_files = glob.glob(path + "/*.csv")

# # concatenate the 8 files into 1.
# dataset = pd.concat((pd.read_csv(f) for f in all_files))
```

In [4]:

```
# # saving the combined dataset to disk named cicids.csv
# dataset.to_csv('cicids')
```

In [5]:

```
dataset=pd.read_csv('../datasets/cicids.csv')
# dataset=pd.read_csv('drive/MyDrive/datasets/cicids.zip')
```

In [6]:

```
# Dimenions of dataset.
print(dataset.shape)
```

```
(2827876, 79)
```

In [7]:

```
# column names as per dataset.

col_names = ["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",
                "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"
               ]
```

In [8]:

```
# Max rows and colummns to be shown in print console

pd.options.display.max_columns= 200
pd.options.display.max_rows= 200
```

In [9]:

```
# Assigning the column names.
dataset.columns = col_names

# first 5 records in the dataset.
dataset.head(5)
```

Out[9]:

	Destination_Port	Flow_Duration	Total_Fwd_Packets	Total_Backward_Packets	Total_Length_of_Fwd_Pacl
0	0	54865	3	2	
1	1	55054	109	1	
2	2	55055	52	1	
3	3	46236	34	1	
4	4	54863	3	2	

In [10]:

```
# check whether there is any categorical column are not if it is there it is to be enco
dataset.dtypes
```

Out[10]:

```
Destination_Port                 int64
Flow_Duration                    int64
Total_Fwd_Packets                int64
Total_Backward_Packets           int64
Total_Length_of_Fwd_Packets      int64
Total_Length_of_Bwd_Packets      int64
Fwd_Packet_Length_Max            int64
Fwd_Packet_Length_Min            int64
Fwd_Packet_Length_Mean           int64
Fwd_Packet_Length_Std          float64
Bwd_Packet_Length_Max          float64
Bwd_Packet_Length_Min            int64
Bwd_Packet_Length_Mean           int64
Bwd_Packet_Length_Std          float64
Flow_Bytes_s                   float64
Flow_Packets_s                 float64
Flow_IAT_Mean                  float64
Flow_IAT_Std                   float64
```

```
Flow_IAT_Max                 float64
Flow_IAT_Min                   int64
Fwd_IAT_Total                  int64
Fwd_IAT_Mean                   int64
Fwd_IAT_Std                  float64
Fwd_IAT_Max                  float64
Fwd_IAT_Min                    int64
Bwd_IAT_Total                  int64
Bwd_IAT_Mean                   int64
Bwd_IAT_Std                  float64
Bwd_IAT_Max                  float64
Bwd_IAT_Min                    int64
Fwd_PSH_Flags                  int64
Bwd_PSH_Flags                  int64
Fwd_URG_Flags                  int64
Bwd_URG_Flags                  int64
Fwd_Header_Length              int64
Bwd_Header_Length              int64
Fwd_Packets_s                  int64
Bwd_Packets_s                float64
Min_Packet_Length            float64
Max_Packet_Length              int64
Packet_Length_Mean             int64
Packet_Length_Std            float64
Packet_Length_Variance       float64
FIN_Flag_Count               float64
SYN_Flag_Count                 int64
RST_Flag_Count                 int64
PSH_Flag_Count                 int64
ACK_Flag_Count                 int64
URG_Flag_Count                 int64
CWE_Flag_Count                 int64
ECE_Flag_Count                 int64
Down_Up_Ratio                  int64
Average_Packet_Size            int64
Avg_Fwd_Segment_Size         float64
Avg_Bwd_Segment_Size         float64
Fwd_Header_Length            float64
Fwd_Avg_Bytes_Bulk             int64
Fwd_Avg_Packets_Bulk           int64
Fwd_Avg_Bulk_Rate              int64
Bwd_Avg_Bytes_Bulk             int64
Bwd_Avg_Packets_Bulk           int64
Bwd_Avg_Bulk_Rate              int64
Subflow_Fwd_Packets            int64
Subflow_Fwd_Bytes              int64
Subflow_Bwd_Packets            int64
Subflow_Bwd_Bytes              int64
Init_Win_bytes_forward         int64
Init_Win_bytes_backward        int64
act_data_pkt_fwd               int64
min_seg_size_forward           int64
Active_Mean                  float64
Active_Std                   float64
Active_Max                     int64
Active_Min                     int64
Idle_Mean                    float64
Idle_Std                     float64
Idle_Max                       int64
Idle_Min                       int64
Label                         object
dtype: object
```

Remove repeated columns, (NaN,Null,Infinite) values.

In [11]:
```
# Removing the duplicate columns (Header_Length is repeated)
dataset = dataset.loc[:, ~dataset.columns.duplicated()]
dataset.shape
```

Out[11]:
```
(2827876, 78)
```

In [12]:
```
# check if there are any Null values
dataset.isnull().any().any()
```

Out[12]:
```
False
```

In [13]:
```
# Replace Inf values with NaN
dataset = dataset.replace([np.inf, -np.inf], np.nan)

# Drop all occurences of NaN
dataset = dataset.dropna()

# Double check these are all gone
dataset.isnull().any()
```

Out[13]:
```
Destination_Port               False
Flow_Duration                  False
Total_Fwd_Packets              False
Total_Backward_Packets         False
Total_Length_of_Fwd_Packets    False
Total_Length_of_Bwd_Packets    False
Fwd_Packet_Length_Max          False
Fwd_Packet_Length_Min          False
Fwd_Packet_Length_Mean         False
Fwd_Packet_Length_Std          False
Bwd_Packet_Length_Max          False
Bwd_Packet_Length_Min          False
Bwd_Packet_Length_Mean         False
Bwd_Packet_Length_Std          False
Flow_Bytes_s                   False
Flow_Packets_s                 False
Flow_IAT_Mean                  False
Flow_IAT_Std                   False
Flow_IAT_Max                   False
Flow_IAT_Min                   False
Fwd_IAT_Total                  False
Fwd_IAT_Mean                   False
Fwd_IAT_Std                    False
Fwd_IAT_Max                    False
Fwd_IAT_Min                    False
Bwd_IAT_Total                  False
Bwd_IAT_Mean                   False
Bwd_IAT_Std                    False
Bwd_IAT_Max                    False
Bwd_IAT_Min                    False
Fwd_PSH_Flags                  False
Bwd_PSH_Flags                  False
Fwd_URG_Flags                  False
Bwd_URG_Flags                  False
```

```
Fwd_Header_Length              False
Bwd_Header_Length              False
Fwd_Packets_s                  False
Bwd_Packets_s                  False
Min_Packet_Length              False
Max_Packet_Length              False
Packet_Length_Mean             False
Packet_Length_Std              False
Packet_Length_Variance         False
FIN_Flag_Count                 False
SYN_Flag_Count                 False
RST_Flag_Count                 False
PSH_Flag_Count                 False
ACK_Flag_Count                 False
URG_Flag_Count                 False
CWE_Flag_Count                 False
ECE_Flag_Count                 False
Down_Up_Ratio                  False
Average_Packet_Size            False
Avg_Fwd_Segment_Size           False
Avg_Bwd_Segment_Size           False
Fwd_Avg_Bytes_Bulk             False
Fwd_Avg_Packets_Bulk           False
Fwd_Avg_Bulk_Rate              False
Bwd_Avg_Bytes_Bulk             False
Bwd_Avg_Packets_Bulk           False
Bwd_Avg_Bulk_Rate              False
Subflow_Fwd_Packets            False
Subflow_Fwd_Bytes              False
Subflow_Bwd_Packets            False
Subflow_Bwd_Bytes              False
Init_Win_bytes_forward         False
Init_Win_bytes_backward        False
act_data_pkt_fwd               False
min_seg_size_forward           False
Active_Mean                    False
Active_Std                     False
Active_Max                     False
Active_Min                     False
Idle_Mean                      False
Idle_Std                       False
Idle_Max                       False
Idle_Min                       False
Label                          False
dtype: bool
```

Analysing the attacks in dataset

In [14]:

```
# Distribution of Dataset
dataset['Label'].value_counts()
```

Out[14]:

```
BENIGN                        2271320
DoS Hulk                       230124
PortScan                       158804
DDoS                           128025
DoS GoldenEye                   10293
FTP-Patator                      7935
SSH-Patator                      5897
DoS slowloris                    5796
DoS Slowhttptest                 5499
Bot                              1956
Web Attack � Brute Force          1507
```

```
Web Attack � XSS                    652
Infiltration                         36
Web Attack � Sql Injection           21
Heartbleed                           11
Name: Label, dtype: int64
```

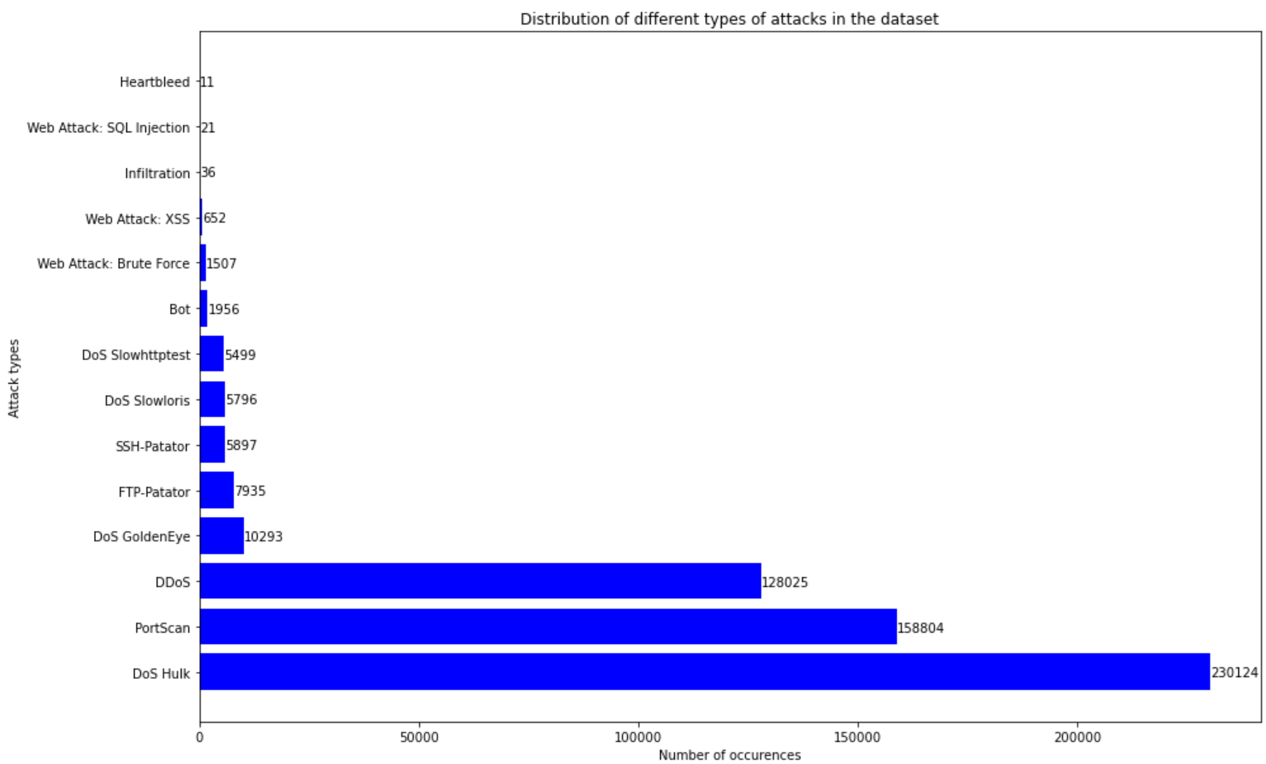
In [15]:

```
# Plotting the distribution of attacks in the dataset

plt.figure(figsize=(15,10))

attack = ('DoS Hulk', 'PortScan', 'DDoS', 'DoS GoldenEye', 'FTP-Patator', 'SSH-Patator'
          'DoS Slowhttptest', 'Bot', 'Web Attack: Brute Force', 'Web Attack: XSS', 'Inf
y_pos = np.arange(len(attack))
amount = dataset['Label'].value_counts()[1:]
plt.barh(y_pos, amount, align='center', color='blue' )
plt.yticks(y_pos, attack)
plt.title('Distribution of different types of attacks in the dataset')
plt.xlabel('Number of occurences')
plt.ylabel('Attack types')
for i, v in enumerate(amount):
    plt.text(v + 3, i-0.1 , str(v))

plt.show()
```



In [16]:

```
# There are only 11, 21, and 36 instances of Heartbleed, SQL injection and infiltration
# Remove 'Heartbleed', 'Web attack Sql Injection', 'Infiltration' as it's negligible.

dataset = dataset.replace(['Heartbleed', 'Web Attack � Sql Injection', 'Infiltration']
dataset = dataset.dropna()
dataset['Label'].value_counts()
```

Out[16]:

```
BENIGN                        2271320
DoS Hulk                       230124
PortScan                       158804
```

```
DDoS                          128025
DoS GoldenEye                  10293
FTP-Patator                     7935
SSH-Patator                     5897
DoS slowloris                   5796
DoS Slowhttptest                5499
Bot                             1956
Web Attack � Brute Force        1507
Web Attack � XSS                 652
Name: Label, dtype: int64
```

In [17]:

```
# Labelling Web Attack � Brute Force as Brute Force
# Labelling Web Attack � XSS as XSS

dataset.loc[dataset.Label == 'Web Attack � Brute Force', ['Label']] = 'Brute Force'
dataset.loc[dataset.Label == 'Web Attack � XSS', ['Label']] = 'XSS'
```

In [18]:

```
# Creating a attack column, containing binary labels for normal and attack to apply bin

dataset['Attack'] = np.where(dataset['Label'] == 'BENIGN','Normal' , 'Attack')
```

In [19]:

```
# Grouping attack labels in attack category as in dataset description for multi-class c

attack_group = {'BENIGN': 'benign',
                'DoS Hulk': 'dos',
                'PortScan': 'probe',
                'DDoS': 'ddos',
                'DoS GoldenEye': 'dos',
                'FTP-Patator': 'brute_force',
                'SSH-Patator': 'brute_force',
                'DoS slowloris': 'dos',
                'DoS Slowhttptest': 'dos',
                'Bot': 'botnet',
                'Brute Force': 'web_attack',
                'XSS': 'web_attack'}

# Create grouped label column

dataset['Label_Category'] = dataset['Label'].map(lambda x: attack_group[x])
dataset['Label_Category'].value_counts()
```

Out[19]:

```
benign         2271320
dos             251712
probe           158804
ddos            128025
brute_force      13832
web_attack        2159
botnet            1956
Name: Label_Category, dtype: int64
```

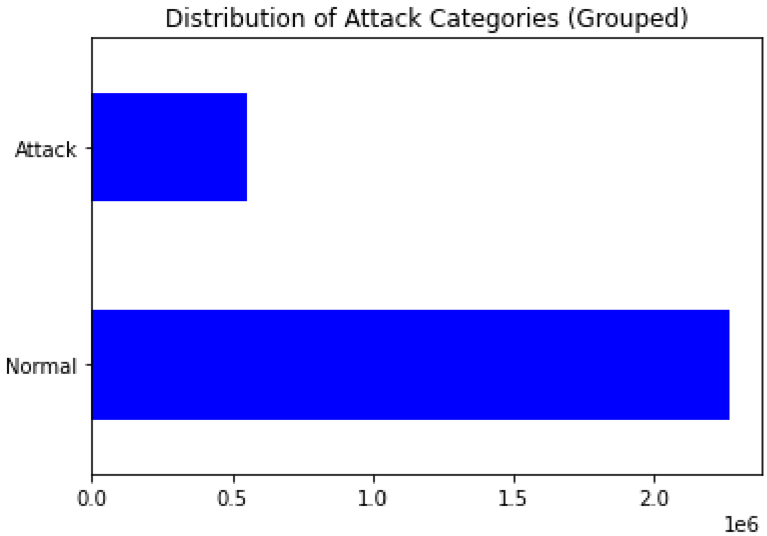
In [20]:

```
# Plotting binary grouped column Attack

train_attacks = dataset['Attack'].value_counts()
train_attacks.plot(kind='barh', color='blue')
plt.title('Distribution of Attack Categories (Grouped)')
```

Out[20]:

```
Text(0.5, 1.0, 'Distribution of Attack Categories (Grouped)')
```

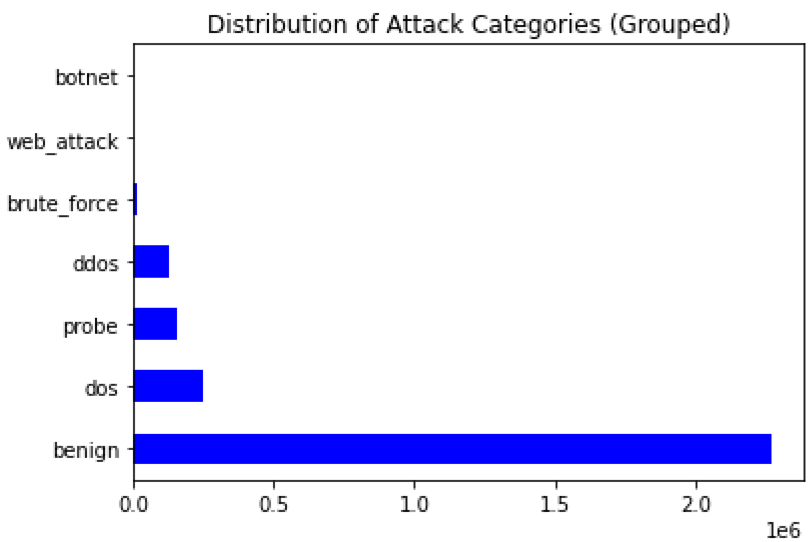


In [21]:

```
# Plotting multi-class grouped column Label_Category

train_attacks = dataset['Label_Category'].value_counts()
train_attacks.plot(kind='barh', color='blue')
plt.title('Distribution of Attack Categories (Grouped)')
```

Out[21]: Text(0.5, 1.0, 'Distribution of Attack Categories (Grouped)')

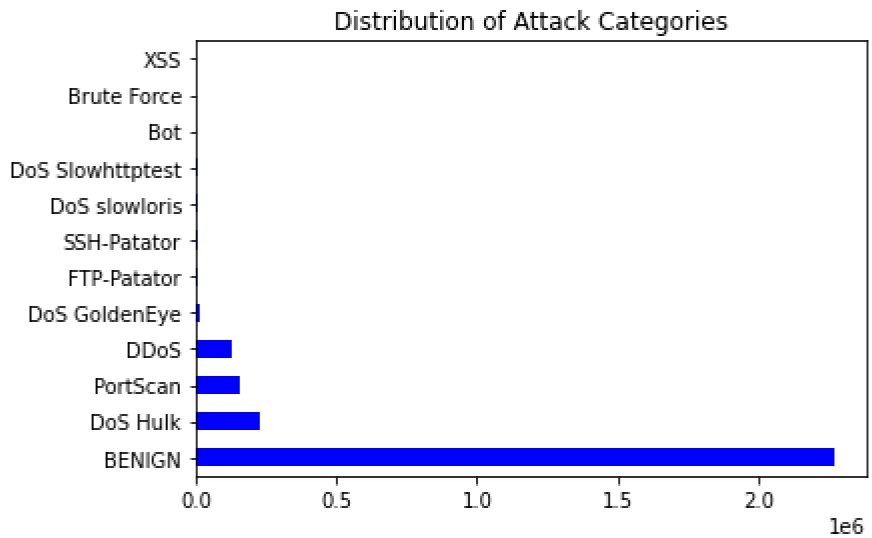


In [22]:

```
# Plotting multi-label column Label

train_attacks = dataset['Label'].value_counts()
train_attacks.plot(kind='barh', color='blue')
plt.title('Distribution of Attack Categories')
```

Out[22]: Text(0.5, 1.0, 'Distribution of Attack Categories')



In [23]:

```
print('Total number of all attack classes :',len(dataset.Label.unique()))
print('Total number of attack categories :',len(dataset.Label_Category.unique()))
```

```
Total number of all attack classes : 12
Total number of attack categories : 7
```

Splitting the dataset

Splitting dataset in 60:20:20 ratio, for training, testing and validation dataset. By stratifying with y label proportions of attacks remain the same throughout the 3 sets.

In [24]:

```
# 3 Different labeling options
attacks = ['Label', 'Label_Category', 'Attack']

# xs=feature vectors, ys=labels
xs = dataset.drop(attacks, axis=1)
ys = dataset[attacks]

# split dataset - stratified
x_train, x_temp, y_train, y_temp = train_test_split(xs, ys, test_size=0.4, random_state
x_test, x_validate, y_test, y_validate = train_test_split(x_temp, y_temp, test_size=0.5
```

Removing the columns with single unique values as it has no contribution in classification

In [25]:

```
column_names = np.array(list(x_train))
to_drop = []
for x in column_names:
    size = x_train.groupby([x]).size()
    # check for columns that only take one value
    if (len(size.unique()) == 1):
        to_drop.append(x)
to_drop
```

Out[25]: ['Fwd_URG_Flags',

```
 'Fwd_Header_Length',
 'Fwd_Avg_Bytes_Bulk',
 'Fwd_Avg_Packets_Bulk',
 'Fwd_Avg_Bulk_Rate',
 'Bwd_Avg_Bytes_Bulk',
 'Bwd_Avg_Packets_Bulk',
 'Bwd_Avg_Bulk_Rate']
```

In [26]:

```
x_train = x_train.drop(to_drop, axis=1)
x_validate = x_validate.drop(to_drop, axis=1)
x_test = x_test.drop(to_drop, axis=1)
dataset_copy = dataset.drop(to_drop, axis=1)
```

In [27]:

```
x_train.shape
```

Out[27]:

```
(1696684, 69)
```

Data Normalization

Min-max normalization technique is used to normalize the numerical values in dataset.

In [28]:

```
# Normalise
min_max_scaler = MinMaxScaler().fit(x_train)

# Apply normalisation to dataset
x_train = min_max_scaler.transform(x_train)
x_validate = min_max_scaler.transform(x_validate)
x_test = min_max_scaler.transform(x_test)
```

Feature Selection

Selecting K-best features by using chi2 scoring function for features

In [29]:

```
features = SelectKBest(score_func=chi2, k=x_train.shape[1])

#fit features to the training dataset
fit = features.fit(x_train, y_train.Label)
```

In [30]:

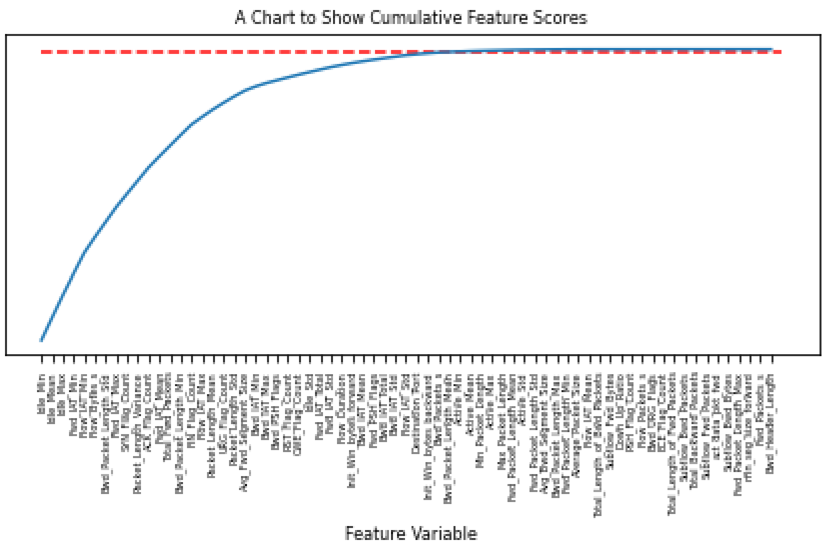
```
# sort the features by importance score
feature_importances = zip(dataset_copy.columns, features.scores_)
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
sorted_importances = [importance[1] for importance in feature_importances]
sorted_features = [importance[0] for importance in feature_importances]

x_values = list(range(len(feature_importances)))

# plot the cumulative scores
cumulative_importances = np.cumsum(sorted_importances)
plt.plot(x_values, cumulative_importances)

# Draw line at 99% of importance retained
value99 = cumulative_importances[-1]*0.99
```

```
plt.hlines(y = value99, xmin=0, xmax=len(sorted_importances), color = 'r', linestyles =
plt.xticks(x_values, sorted_features, rotation = 'vertical', fontsize=5)
plt.yticks([], [])
plt.xlabel('Feature Variable', fontsize=8)
plt.title('A Chart to Show Cumulative Feature Scores', fontsize=8)
#plt.figure(figsize=(500,200))
plt.tight_layout()
plt.savefig('cum_features.png', dpi=300)
```



In [31]:

```
# perform selectkbest with k=40

features = SelectKBest(score_func=chi2, k=40)
fit = features.fit(x_train, y_train.Label)

x_train = fit.transform(x_train)
x_test = fit.transform(x_test)
x_validate = fit.transform(x_validate)
```

In [32]:

```
new_features = dataset_copy.columns[features.get_support(indices=True)]
```

In [33]:

```
print('Number of features selected :',len(new_features))
new_features
```

```
Number of features selected : 40
```

Out[33]:

```
Index(['Destination_Port', 'Flow_Duration', 'Total_Fwd_Packets',
       'Bwd_Packet_Length_Min', 'Bwd_Packet_Length_Mean',
       'Bwd_Packet_Length_Std', 'Flow_Bytes_s', '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', 'Bwd_Packets_s', 'Packet_Length_Mean',
       'Packet_Length_Std', 'Packet_Length_Variance', 'FIN_Flag_Count',
       'SYN_Flag_Count', 'RST_Flag_Count', 'ACK_Flag_Count', 'URG_Flag_Count',
       'CWE_Flag_Count', 'Avg_Fwd_Segment_Size', 'Init_Win_bytes_forward',
       'Init_Win_bytes_backward', 'Active_Min', 'Idle_Mean', 'Idle_Std',
       'Idle_Max', 'Idle_Min'],
      dtype='object')
```

In [34]:
```
attack = np.array(['BENIGN', 'Bot', 'Brute Force', 'DDoS', 'DoS GoldenEye', 'DoS Hulk',
                   'DoS slowloris', 'FTP-Patator', 'PortScan', 'SSH-Patator', 'XSS'])
attack_groups = np.array(['benign', 'botnet', 'brute_force', 'ddos', 'dos', 'probe', 'w
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Applying Machine Learning classifier models

Each machine learning algorithm is applied in three different categories :

1. On all attack labels (12).
2. Binary Classifier (2).
3. Multi-class Classifier (7).

And then evaluate performance of each algorithm by confusion matrix plot. Evaluate Accuracy, Precision, Recall, F1-score.

1. Support Vector Machine (SVM)

In [35]:
```
classifier = LinearSVC()
```

1. a) On all attack labels.

In [36]:
```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Label)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  408.1688220500946
```

In [37]:
```
# predicting test results of SVM classifier on all labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  0.15294408798217773
```

In [38]:

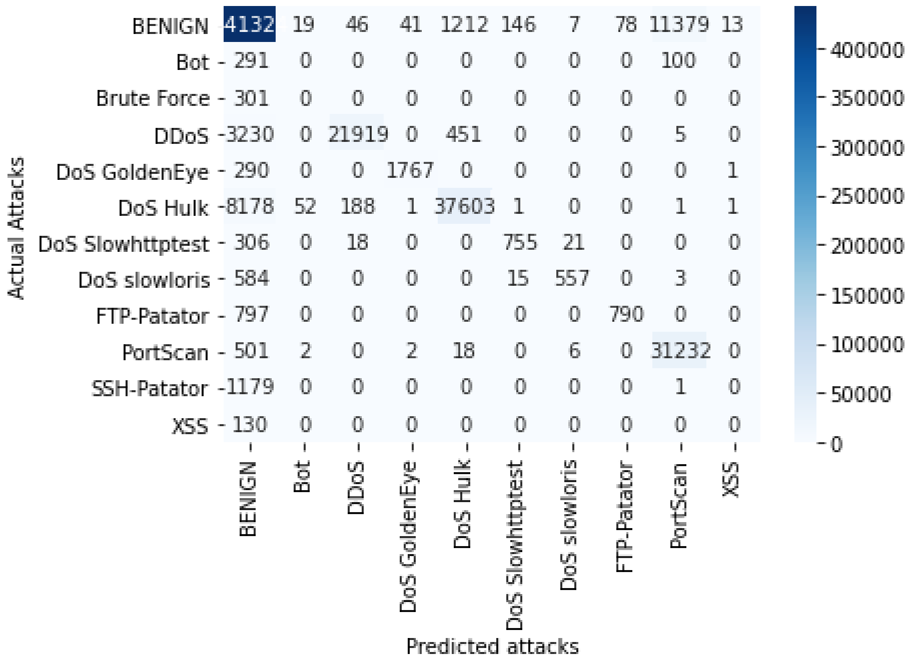
```
# Creating confusion matrix for SVM classifier on all labels.

confusion_svm_1 = pd.crosstab(y_validate.Label, y_predict, rownames=['Actual Attacks'],

print("Plotting Confusion Matrix of SVM classifier on all Labels ")

sn.heatmap(confusion_svm_1, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_svm_1
```

Plotting Confusion Matrix of SVM classifier on all Labels



Out[38]:

Predicted attacks / Actual Attacks	BENIGN	Bot	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	PortScan	XSS
BENIGN	441324	19	46	41	1212	146	7	78	11379	13
Bot	291	0	0	0	0	0	0	0	100	0
Brute Force	301	0	0	0	0	0	0	0	0	0
DDoS	3230	0	21919	0	451	0	0	0	5	0
DoS GoldenEye	290	0	0	1767	0	0	0	0	0	1
DoS Hulk	8178	52	188	1	37603	1	0	0	1	1
DoS Slowhttptest	306	0	18	0	0	755	21	0	0	0
DoS slowloris	584	0	0	0	0	15	557	0	3	0
FTP-Patator	797	0	0	0	0	0	0	790	0	0
PortScan	501	2	0	2	18	0	6	0	31232	0

Predicted attacks / Actual Attacks	BENIGN	Bot	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	PortScan	XSS
SSH-Patator	1179	0	0	0	0	0	0	0	1	0
XSS	130	0	0	0	0	0	0	0	0	0

In [39]:

```
# Precision,Recall,F1-score for SVM classifier on all labels.

precision, recall, fscore, support = score(y_validate.Label, y_predict)

d = {'attack': attack, 'precision': precision, 'recall' : recall, 'fscore': fscore}
results = pd.DataFrame(data=d)
results
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Undef
inedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[39]:

	attack	precision	recall	fscore
0	BENIGN	0.965464	0.971512	0.968478
1	Bot	0.000000	0.000000	0.000000
2	Brute Force	0.000000	0.000000	0.000000
3	DDoS	0.988634	0.856044	0.917574
4	DoS GoldenEye	0.975704	0.858601	0.913414
5	DoS Hulk	0.957209	0.817012	0.881572
6	DoS Slowhttptest	0.823337	0.686364	0.748637
7	DoS slowloris	0.942470	0.480587	0.636571
8	FTP-Patator	0.910138	0.497795	0.643585
9	PortScan	0.731069	0.983344	0.838646
10	SSH-Patator	0.000000	0.000000	0.000000
11	XSS	0.000000	0.000000	0.000000

In [40]:

```
# Average Accuracy,Precision,Recall,F1-score for SVM classifier on all labels.

precision_svm_1, recall_svm_1, fscore_svm_1, support = score(y_validate.Label, y_predic
accuracy_svm_1 = accuracy_score(y_validate.Label, y_predict)
print("Accuracy of SVM classifier on all labels : ", accuracy_svm_1)
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Undef
inedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Accuracy of SVM classifier on all labels :  0.9476361566017519
```

In [ ]:

1. b) Binary Classifier.

In [41]:

```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Attack)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  158.71935606002808
```

In [42]:

```
# predicting test results of SVM classifier on binary labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  0.07494568824768066
```

In [43]:

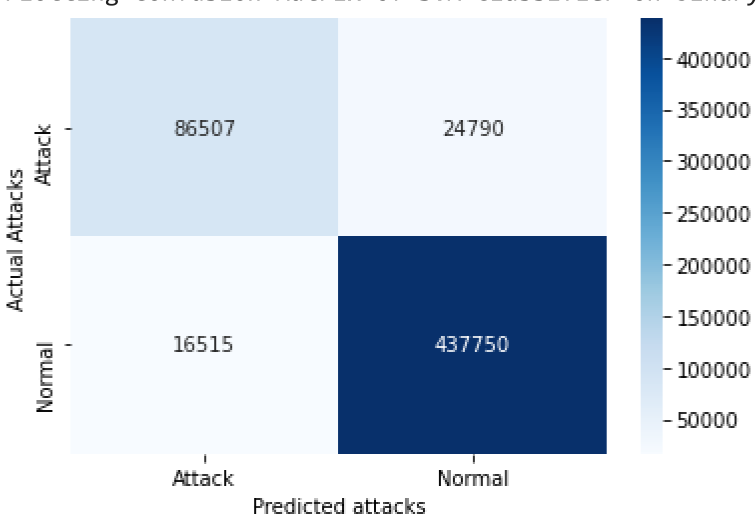
```
# Creating confusion matrix for SVM classifier on binary labels.

confusion_svm_2 = pd.crosstab(y_validate.Attack, y_predict, rownames=['Actual Attacks']

print("Plotting Confusion Matrix of SVM classifier on binary Labels ")

sn.heatmap(confusion_svm_2, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_svm_2
```

```
Plotting Confusion Matrix of SVM classifier on binary Labels
```



Out[43]: **Predicted attacks** **Attack** **Normal**

Predicted Attacks / Actual Attacks	Attack	Normal
Actual Attacks		
Attack	86507	24790
Normal	16515	437750

In [44]:

```
# Precision,Recall,F1-score for SVM classifier on binary labels.

precision, recall, fscore, support = score(y_validate.Attack, y_predict)
d = {'attack': [0,1], 'precision': precision, 'recall' : recall, 'fscore': fscore}
results = pd.DataFrame(data=d)
results
```

Out[44]:

	attack	precision	recall	fscore
0	0	0.839694	0.777263	0.807273
1	1	0.946405	0.963645	0.954947

In [45]:

```
# Average Accuracy,Precision,Recall,F1-score for SVM classifier on binary labels.

precision_svm_2, recall_svm_2, fscore_svm_2, n = score(y_validate.Attack, y_predict, av
accuracy_svm_2 = accuracy_score(y_validate.Attack, y_predict)
print("Accuracy of SVM classifier on binary labels : ", accuracy_svm_2)
```

```
Accuracy of SVM classifier on binary labels :  0.9269664510699093
```

In [ ]:

1. c) Multi-class Classifier.

In [46]:

```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Label_Category)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  352.8527202606201
```

In [47]:

```
# predicting test results of SVM classifier on multi-class labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  0.12475442886352539
```

In [48]:

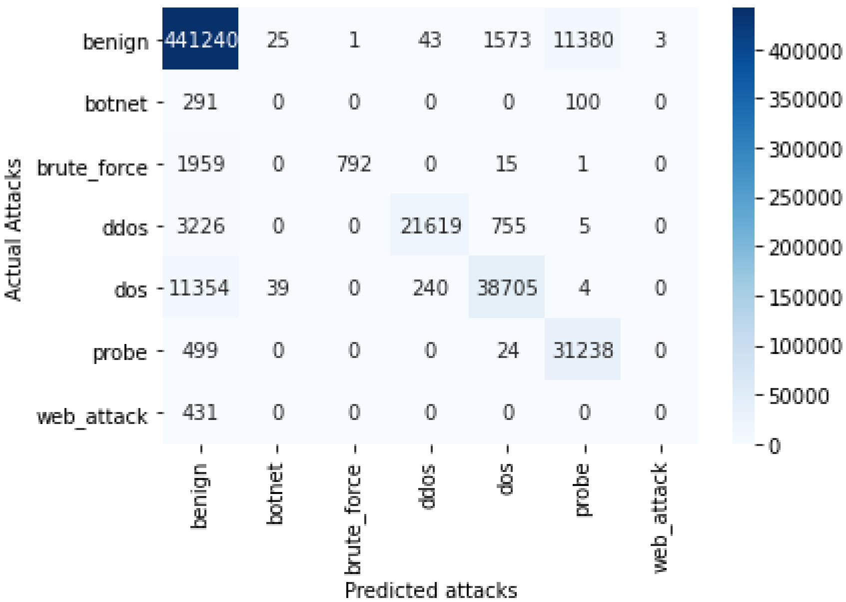
```python
# Creating confusion matrix for SVM classifier on multi-class labels.

confusion_svm_3 = pd.crosstab(y_validate.Label_Category, y_predict, rownames=['Actual A

print("Plotting Confusion Matrix of SVM classifier on multi-class Labels ")

sn.heatmap(confusion_svm_3, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_svm_3
```

```
Plotting Confusion Matrix of SVM classifier on multi-class Labels
```



Out[48]:

Predicted attacks	benign	botnet	brute_force	ddos	dos	probe	web_attack
Actual Attacks							
benign	441240	25	1	43	1573	11380	3
botnet	291	0	0	0	0	100	0
brute_force	1959	0	792	0	15	1	0
ddos	3226	0	0	21619	755	5	0
dos	11354	39	0	240	38705	4	0
probe	499	0	0	0	24	31238	0
web_attack	431	0	0	0	0	0	0

In [49]:

```python
# Precision,Recall,F1-score for SVM classifier on multi-class labels.

precision, recall, fscore, support = score(y_validate.Label_Category, y_predict)
d = {'attack': attack_groups, 'precision': precision, 'recall' : recall, 'fscore': fsco
results = pd.DataFrame(data=d)
results
```

Out[49]:

	attack	precision	recall	fscore

	attack	precision	recall	fscore
0	benign	0.961307	0.971327	0.966291
1	botnet	0.000000	0.000000	0.000000
2	brute_force	0.998739	0.286231	0.444944
3	ddos	0.987079	0.844327	0.910140
4	dos	0.942369	0.768841	0.846807
5	probe	0.731090	0.983533	0.838728
6	web_attack	0.000000	0.000000	0.000000

In [50]:

```
# Average Accuracy,Precision,Recall,F1-score for SVM classifier on multi-class labels.

precision_svm_3, recall_svm_3, fscore_svm_3, n = score(y_validate.Label_Category, y_pre
accuracy_svm_3 = accuracy_score(y_validate.Label_Category, y_predict)
print("Accuracy of SVM classifier on multi-class labels : ", accuracy_svm_3)
```

```
Accuracy of SVM classifier on multi-class labels :  0.9434756932042817
```

In [ ]:

Results for SVM:

In [51]:

```
print('Support Vector Machine: Precision / Recall / Fscore / Accuracy')

print('All Labels:', precision_svm_1, recall_svm_1, fscore_svm_1,  accuracy_svm_1)
print('Binary Labels:', precision_svm_2, recall_svm_2, fscore_svm_2, accuracy_svm_2)
print('Multi-class Labels:', precision_svm_3, recall_svm_3, fscore_svm_3, accuracy_svm_
```

```
Support Vector Machine: Precision / Recall / Fscore / Accuracy
All Labels: 0.6078354200082602 0.5126048603294849 0.5457063537203636 0.9476361566017519
Binary Labels: 0.8930495347366538 0.8704536041514149 0.8811100345781537 0.92696645106990
93
Multi-class Labels: 0.66008344887583 0.5506085077381077 0.5724156218459238 0.94347569320
42817
```

In [ ]:

In [ ]:

2. Decision Tree

In [52]:

```
classifier = DecisionTreeClassifier(random_state = 0)
```

1. a) On all attack labels.

In [53]:

```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Label)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  112.55840706825256
```

In [54]:

```
# predicting test results of Decision Tree classifier on all labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  0.24280858039855957
```

In [55]:

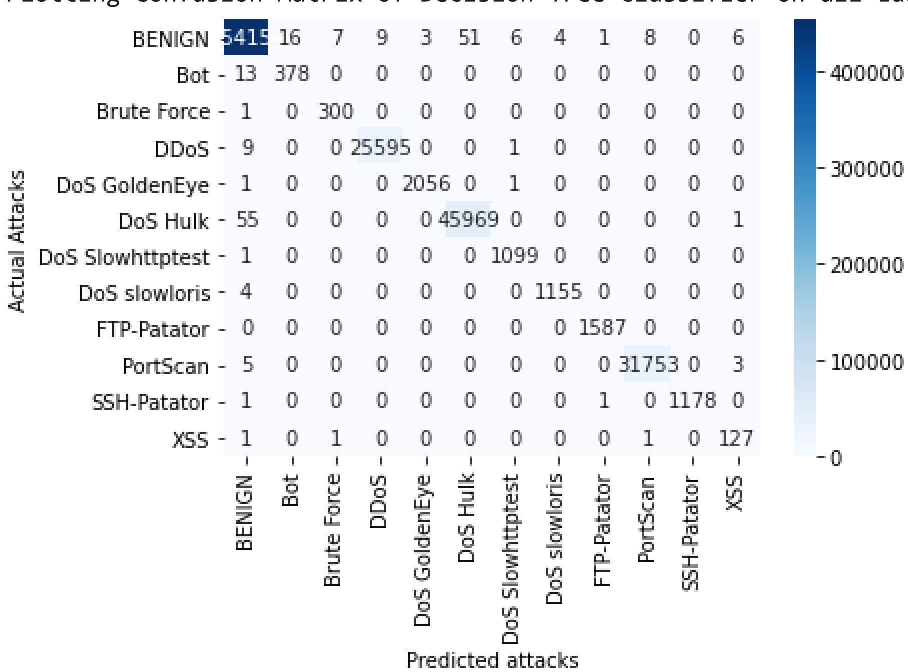
```
# Creating confusion matrix for Decision Tree classifier on all labels.

confusion_dt_1 = pd.crosstab(y_validate.Label, y_predict, rownames=['Actual Attacks'],

print("Plotting Confusion Matrix of Decision Tree classifier on all Labels ")

sn.heatmap(confusion_dt_1, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_dt_1
```

```
Plotting Confusion Matrix of Decision Tree classifier on all Labels
```



Out[55]:

Predicted attacks	BENIGN	Bot	Brute Force	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	PortScan
Actual Attacks										

Predicted attacks	BENIGN	Bot	Brute Force	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	PortScar
Actual Attacks										
BENIGN	454154	16	7	9	3	51	6	4	1	8
Bot	13	378	0	0	0	0	0	0	0	(
Brute Force	1	0	300	0	0	0	0	0	0	(
DDoS	9	0	0	25595	0	0	1	0	0	(
DoS GoldenEye	1	0	0	0	2056	0	1	0	0	(
DoS Hulk	55	0	0	0	0	45969	0	0	0	(
DoS Slowhttptest	1	0	0	0	0	0	1099	0	0	(
DoS slowloris	4	0	0	0	0	0	0	1155	0	(
FTP-Patator	0	0	0	0	0	0	0	0	1587	(
PortScan	5	0	0	0	0	0	0	0	0	3175
SSH-Patator	1	0	0	0	0	0	0	0	1	(
XSS	1	0	1	0	0	0	0	0	0	'

In [56]:

```python
# Precision,Recall,F1-score for Decision Tree classifier on all labels.

precision, recall, fscore, support = score(y_validate.Label, y_predict)

d = {'attack': attack, 'precision': precision, 'recall' : recall, 'fscore': fscore}
results = pd.DataFrame(data=d)
results
```

Out[56]:

	attack	precision	recall	fscore
0	BENIGN	0.999800	0.999756	0.999778
1	Bot	0.959391	0.966752	0.963057
2	Brute Force	0.974026	0.996678	0.985222
3	DDoS	0.999648	0.999609	0.999629
4	DoS GoldenEye	0.998543	0.999028	0.998786
5	DoS Hulk	0.998892	0.998783	0.998838
6	DoS Slowhttptest	0.992773	0.999091	0.995922
7	DoS slowloris	0.996549	0.996549	0.996549
8	FTP-Patator	0.998741	1.000000	0.999370
9	PortScan	0.999717	0.999748	0.999732

	attack	precision	recall	fscore
10	SSH-Patator	1.000000	0.998305	0.999152
11	XSS	0.927007	0.976923	0.951311

In [57]:

```
# Average Accuracy,Precision,Recall,F1-score for Decision Tree classifier on all labels

precision_dt_1, recall_dt_1, fscore_dt_1, support = score(y_validate.Label, y_predict,
accuracy_dt_1 = accuracy_score(y_validate.Label, y_predict)
print("Accuracy of Decision Tree classifier on all labels : ", accuracy_dt_1)
```

```
Accuracy of Decision Tree classifier on all labels :  0.9996269197718376
```

In [58]:

```
# 1. b) Binary Classifier.
```

In [59]:

```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Attack)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  103.06987261772156
```

In [60]:

```
# predicting test results of Decision Tree classifier on binary labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  0.18704795837402344
```

In [61]:

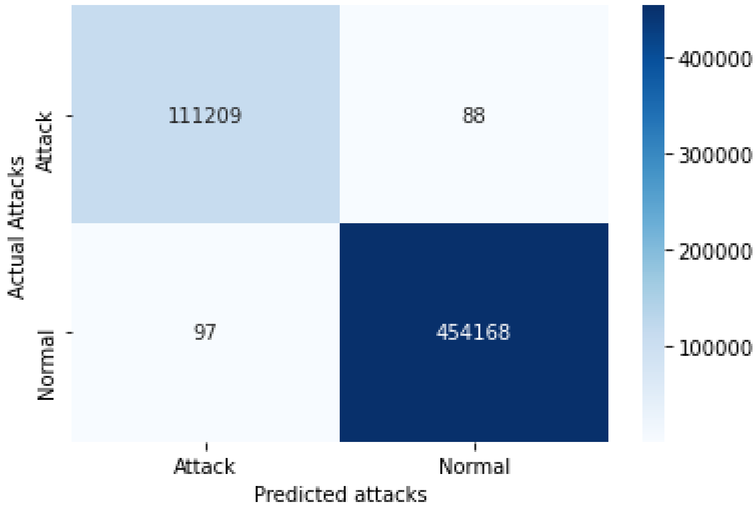
```
# Creating confusion matrix for Decision Tree classifier on binary labels.

confusion_dt_2 = pd.crosstab(y_validate.Attack, y_predict, rownames=['Actual Attacks'],

print("Plotting Confusion Matrix of Decision Tree classifier on binary Labels ")

sn.heatmap(confusion_dt_2, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_dt_2
```

```
Plotting Confusion Matrix of Decision Tree classifier on binary Labels
```



Out[61]:

Predicted attacks	Attack	Normal
Actual Attacks		
Attack	111209	88
Normal	97	454168

In [62]:

```
# Precision,Recall,F1-score for Decision Tree classifier on binary labels.

precision, recall, fscore, support = score(y_validate.Attack, y_predict)
d = {'attack': [0,1], 'precision': precision, 'recall' : recall, 'fscore': fscore}
results = pd.DataFrame(data=d)
results
```

Out[62]:

	attack	precision	recall	fscore
0	0	0.999129	0.999209	0.999169
1	1	0.999806	0.999786	0.999796

In [63]:

```
# Average Accuracy,Precision,Recall,F1-score for Decision Tree classifier on binary lab

precision_dt_2, recall_dt_2, fscore_dt_2, n = score(y_validate.Attack, y_predict, avera
accuracy_dt_2 = accuracy_score(y_validate.Attack, y_predict)
print("Accuracy of Decision Tree classifier on binary labels : ", accuracy_dt_2)
```

```
Accuracy of Decision Tree classifier on binary labels :  0.9996728917430804
```

In [64]:

```
# 1. c) Multi-class Classifier.
```

In [65]:

```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Label_Category)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  89.94037556648254
```

In [66]:

```
# predicting test results of Decision Tree classifier on multi-class labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  0.12569069862365723
```

In [67]:

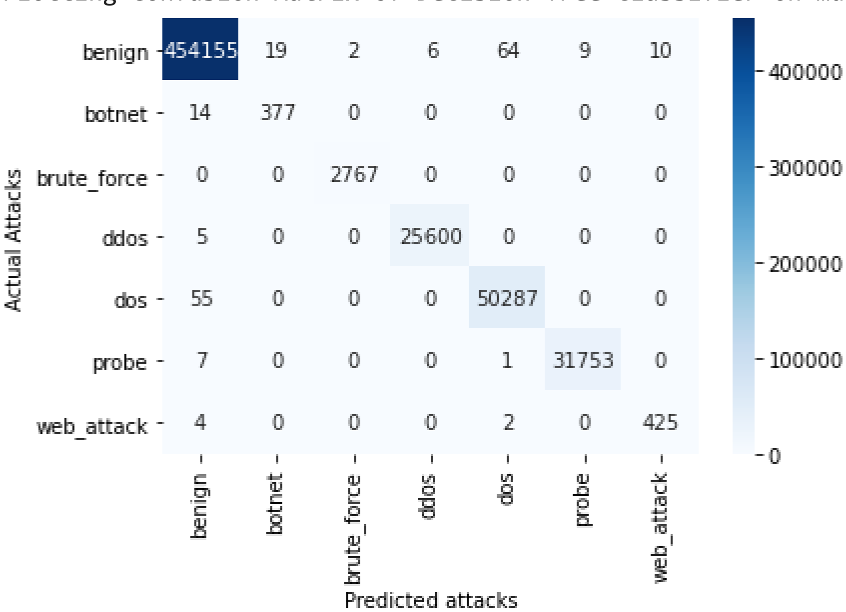
```
# Creating confusion matrix for Decision Tree classifier on multi-class labels.

confusion_dt_3 = pd.crosstab(y_validate.Label_Category, y_predict, rownames=['Actual At

print("Plotting Confusion Matrix of Decision Tree classifier on multi-class Labels ")

sn.heatmap(confusion_dt_3, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_dt_3
```

```
Plotting Confusion Matrix of Decision Tree classifier on multi-class Labels
```



Out[67]:

Predicted attacks	benign	botnet	brute_force	ddos	dos	probe	web_attack
Actual Attacks							
benign	454155	19	2	6	64	9	10
botnet	14	377	0	0	0	0	0
brute_force	0	0	2767	0	0	0	0
ddos	5	0	0	25600	0	0	0
dos	55	0	0	0	50287	0	0
probe	7	0	0	0	1	31753	0
web_attack	4	0	0	0	2	0	425

In [68]:
```
# Precision,Recall,F1-score for Decision Tree classifier on multi-class labels.

precision, recall, fscore, support = score(y_validate.Label_Category, y_predict)
d = {'attack': attack_groups, 'precision': precision, 'recall' : recall, 'fscore': fsco
results = pd.DataFrame(data=d)
results
```

Out[68]:

	attack	precision	recall	fscore
0	benign	0.999813	0.999758	0.999785
1	botnet	0.952020	0.964194	0.958069
2	brute_force	0.999278	1.000000	0.999639
3	ddos	0.999766	0.999805	0.999785
4	dos	0.998669	0.998907	0.998788
5	probe	0.999717	0.999748	0.999732
6	web_attack	0.977011	0.986079	0.981524

In [69]:
```
# Average Accuracy,Precision,Recall,F1-score for Decision Tree classifier on multi-clas

precision_dt_3, recall_dt_3, fscore_dt_3, n = score(y_validate.Label_Category, y_predic
accuracy_dt_3 = accuracy_score(y_validate.Label_Category, y_predict)
print("Accuracy of Decision Tree classifier on multi-class labels : ", accuracy_dt_3)
```

```
Accuracy of Decision Tree classifier on multi-class labels :  0.9996499057574589
```

Results for Decision Tree:

In [70]:
```
print('Decission Tree Classifier : Precision / Recall / Fscore / Accuracy')

print('All Labels:', precision_dt_1, recall_dt_1, fscore_dt_1,  accuracy_dt_1)
print('Binary Labels:', precision_dt_2, recall_dt_2, fscore_dt_2, accuracy_dt_2)
print('Multi-class Labels:', precision_dt_3, recall_dt_3, fscore_dt_3, accuracy_dt_3)
```

```
Decission Tree Classifier : Precision / Recall / Fscore / Accuracy
All Labels: 0.9870905886440647 0.9942685125536538 0.9906120696022297 0.9996269197718376
Binary Labels: 0.9994674025989333 0.9994978955269307 0.9994826481972332 0.99967289174308
04
Multi-class Labels: 0.9894677187232237 0.9926416325133209 0.9910461385301256 0.999649905
7574589
```

In [71]:
```
# 3. Naive Bayes Classifier
```

In [72]:
```
classifier = MultinomialNB()
```

In [73]:
```
# 1. a) On all attack labels.
```

In [74]:

```python
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Label)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  7.247187614440918
```

In [75]:

```python
# predicting test results of Naive Bayes classifier on all labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  0.11592531204223633
```

In [76]:

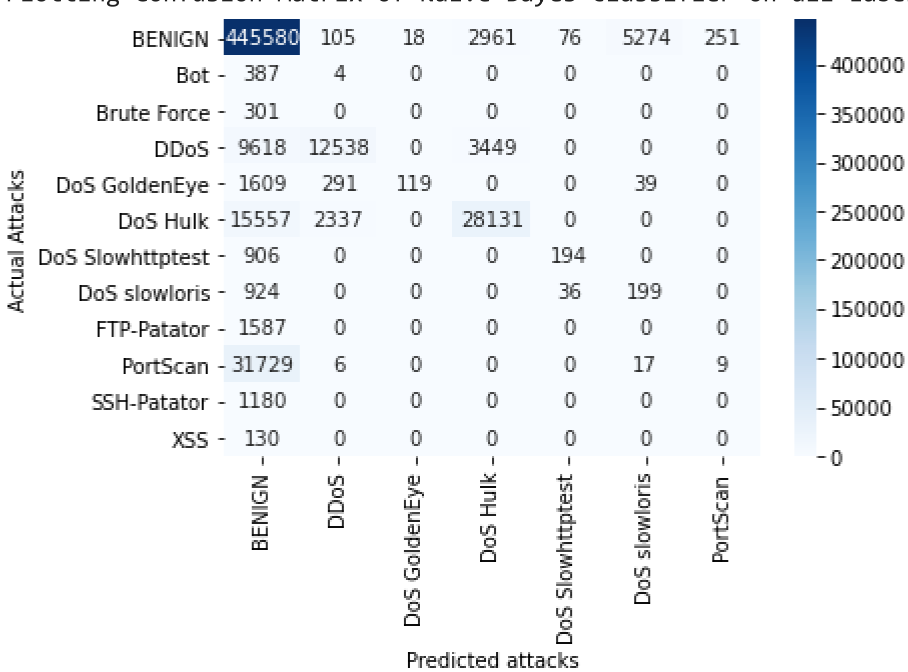
```python
# Creating confusion matrix for Naive Bayes classifier on all labels.

confusion_nb_1 = pd.crosstab(y_validate.Label, y_predict, rownames=['Actual Attacks'],

print("Plotting Confusion Matrix of Naive Bayes classifier on all Labels ")

sn.heatmap(confusion_nb_1, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_nb_1
```

```
Plotting Confusion Matrix of Naive Bayes classifier on all Labels
```



Out[76]:

Predicted attacks	BENIGN	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	PortScan
Actual Attacks							

Predicted attacks Actual Attacks	BENIGN	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	PortScan
BENIGN	445580	105	18	2961	76	5274	251
Bot	387	4	0	0	0	0	0
Brute Force	301	0	0	0	0	0	0
DDoS	9618	12538	0	3449	0	0	0
DoS GoldenEye	1609	291	119	0	0	39	0
DoS Hulk	15557	2337	0	28131	0	0	0
DoS Slowhttptest	906	0	0	0	194	0	0
DoS slowloris	924	0	0	0	36	199	0
FTP-Patator	1587	0	0	0	0	0	0
PortScan	31729	6	0	0	0	17	9
SSH-Patator	1180	0	0	0	0	0	0
XSS	130	0	0	0	0	0	0

In [77]:

```
# Precision,Recall,F1-score for Naive Bayes classifier on all labels.

precision, recall, fscore, support = score(y_validate.Label, y_predict)

d = {'attack': attack, 'precision': precision, 'recall' : recall, 'fscore': fscore}
results = pd.DataFrame(data=d)
results
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Undef
inedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[77]:

	attack	precision	recall	fscore
0	BENIGN	0.874530	0.980881	0.924658
1	Bot	0.000000	0.000000	0.000000
2	Brute Force	0.000000	0.000000	0.000000
3	DDoS	0.820496	0.489670	0.613315
4	DoS GoldenEye	0.868613	0.057823	0.108428
5	DoS Hulk	0.814423	0.611211	0.698334
6	DoS Slowhttptest	0.633987	0.176364	0.275960
7	DoS slowloris	0.035992	0.171700	0.059510
8	FTP-Patator	0.000000	0.000000	0.000000
9	PortScan	0.034615	0.000283	0.000562

	attack	precision	recall	fscore
10	SSH-Patator	0.000000	0.000000	0.000000
11	XSS	0.000000	0.000000	0.000000

In [78]:

```
# Average Accuracy,Precision,Recall,F1-score for Naive Bayes classifier on all labels.

precision_nb_1, recall_nb_1, fscore_nb_1, support = score(y_validate.Label, y_predict,
accuracy_nb_1 = accuracy_score(y_validate.Label, y_predict)
print("Accuracy of Naive Bayes classifier on all labels : ", accuracy_nb_1)
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Undef
inedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Accuracy of Naive Bayes classifier on all labels :  0.8606837093015443
```

In [79]:

```
# 1. b) Binary Classifier.
```

In [80]:

```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Attack)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  18.901024341583252
```

In [81]:

```
# predicting test results of Naive Bayes classifier on binary labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  0.05728888511657715
```

In [82]:

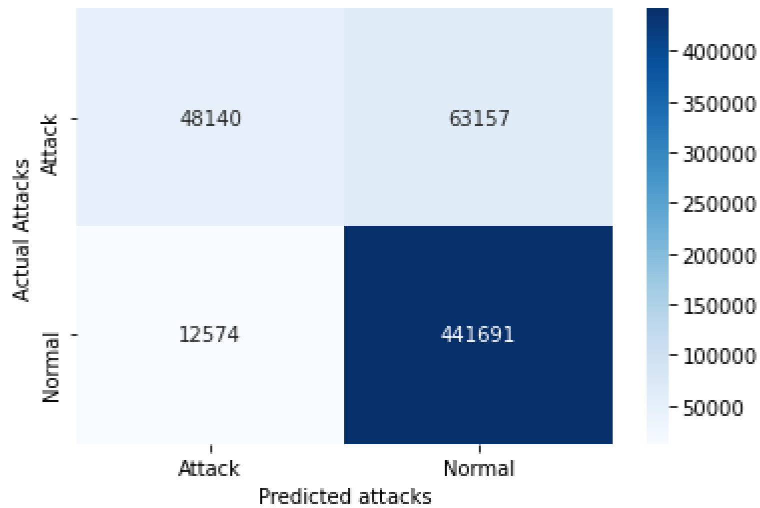
```
# Creating confusion matrix for Naive Bayes classifier on binary labels.

confusion_nb_2 = pd.crosstab(y_validate.Attack, y_predict, rownames=['Actual Attacks'],

print("Plotting Confusion Matrix of Naive Bayes classifier on binary Labels ")

sn.heatmap(confusion_nb_2, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_nb_2
```

```
Plotting Confusion Matrix of Naive Bayes classifier on binary Labels
```



Out[82]:

Predicted attacks	Attack	Normal
Actual Attacks		
Attack	48140	63157
Normal	12574	441691

In [83]:

```
# Precision,Recall,F1-score for Naive Bayes classifier on binary labels.

precision, recall, fscore, support = score(y_validate.Attack, y_predict)
d = {'attack': [0,1], 'precision': precision, 'recall' : recall, 'fscore': fscore}
results = pd.DataFrame(data=d)
results
```

Out[83]:

	attack	precision	recall	fscore
0	0	0.792898	0.432536	0.559732
1	1	0.874899	0.972320	0.921041

In [84]:

```
# Average Accuracy,Precision,Recall,F1-score for Naive Bayes classifier on binary label

precision_nb_2, recall_nb_2, fscore_nb_2, n = score(y_validate.Attack, y_predict, avera
accuracy_nb_2 = accuracy_score(y_validate.Attack, y_predict)
print("Accuracy of Naive Bayes classifier on binary labels : ", accuracy_nb_2)
```

```
Accuracy of Naive Bayes classifier on binary labels :  0.8660960248390097
```

In [85]:

```
# 1. c) Multi-class Classifier.
```

In [86]:

```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Label_Category)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  4.853246688842773
```

In [87]:

```
# predicting test results of Naive Bayes classifier on multi-class labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  0.07313156127929688
```

In [88]:

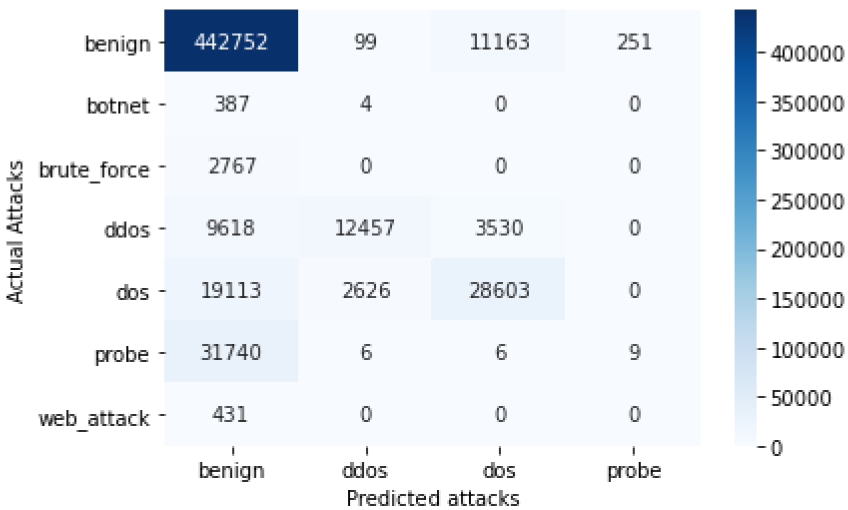
```
# Creating confusion matrix for Naive Bayes classifier on multi-class labels.

confusion_nb_3 = pd.crosstab(y_validate.Label_Category, y_predict, rownames=['Actual At

print("Plotting Confusion Matrix of Naive Bayes classifier on multi-class Labels ")

sn.heatmap(confusion_nb_3, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_nb_3
```

```
Plotting Confusion Matrix of Naive Bayes classifier on multi-class Labels
```



Out[88]:

Predicted attacks	benign	ddos	dos	probe
Actual Attacks				
benign	442752	99	11163	251
botnet	387	4	0	0
brute_force	2767	0	0	0
ddos	9618	12457	3530	0
dos	19113	2626	28603	0
probe	31740	6	6	9
web_attack	431	0	0	0

In [89]:

```
# Precision,Recall,F1-score for Naive Bayes classifier on multi-class labels.

precision, recall, fscore, support = score(y_validate.Label_Category, y_predict)
d = {'attack': attack_groups, 'precision': precision, 'recall' : recall, 'fscore': fsco
results = pd.DataFrame(data=d)
results
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Undef
inedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[89]:

	attack	precision	recall	fscore
0	benign	0.873609	0.974656	0.921370
1	botnet	0.000000	0.000000	0.000000
2	brute_force	0.000000	0.000000	0.000000
3	ddos	0.819971	0.486507	0.610682
4	dos	0.660547	0.568174	0.610888
5	probe	0.034615	0.000283	0.000562
6	web_attack	0.000000	0.000000	0.000000

In [90]:

```
# Average Accuracy,Precision,Recall,F1-score for Naive Bayes classifier on multi-class

precision_nb_3, recall_nb_3, fscore_nb_3, n = score(y_validate.Label_Category, y_predic
accuracy_nb_3 = accuracy_score(y_validate.Label_Category, y_predict)
print("Accuracy of Naive Bayes classifier on multi-class labels : ", accuracy_nb_3)
```

```
Accuracy of Naive Bayes classifier on multi-class labels :  0.8554694268709708
C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Undef
inedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [91]:

```
### Results for Naive Bayes:
```

In [92]:

```
print('Naive Bayes: Precision / Recall / Fscore / Accuracy')

print('All Labels:', precision_nb_1, recall_nb_1, fscore_nb_1,  accuracy_nb_1)
print('Binary Labels:', precision_nb_2, recall_nb_2, fscore_nb_2, accuracy_nb_2)
print('Multi-class Labels:', precision_nb_3, recall_nb_3, fscore_nb_3, accuracy_nb_3)
```

```
Naive Bayes: Precision / Recall / Fscore / Accuracy
All Labels: 0.3402214093082901 0.20732769678376894 0.22339725358984744 0.860683709301544
3
Binary Labels: 0.833898414212936 0.7024282440867775 0.7403861157638703 0.866096024839009
7
Multi-class Labels: 0.3412488885179302 0.289945623233776 0.3062146442943346 0.8554694268
709708
```

In [93]:

```
# 6. Random Forest Classifier
```

In [94]:
```
classifier =  RandomForestClassifier()
```

In [95]:
```
# 1. a) On all attack labels.
```

In [96]:
```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Label)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  934.3901979923248
```

In [97]:
```
# predicting test results of Random Forest classifier on all labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  9.01480770111084
```

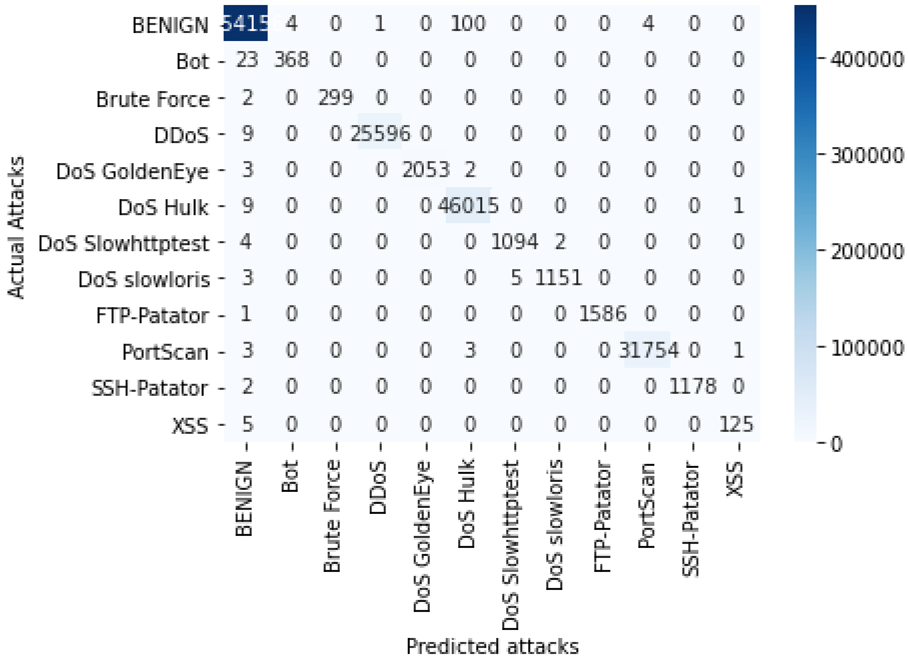
In [98]:
```
# Creating confusion matrix for Random Forest classifier on all labels.

confusion_rf_1 = pd.crosstab(y_validate.Label, y_predict, rownames=['Actual Attacks'],

print("Plotting Confusion Matrix of Random Forest classifier on all Labels ")

sn.heatmap(confusion_rf_1, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_rf_1
```

```
Plotting Confusion Matrix of Random Forest classifier on all Labels
```



Out[98]:

Predicted attacks / Actual Attacks	BENIGN	Bot	Brute Force	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	PortScar
BENIGN	454156	4	0	1	0	100	0	0	0	4
Bot	23	368	0	0	0	0	0	0	0	0
Brute Force	2	0	299	0	0	0	0	0	0	0
DDoS	9	0	0	25596	0	0	0	0	0	0
DoS GoldenEye	3	0	0	0	2053	2	0	0	0	0
DoS Hulk	9	0	0	0	0	46015	0	0	0	0
DoS Slowhttptest	4	0	0	0	0	0	1094	2	0	0
DoS slowloris	3	0	0	0	0	0	5	1151	0	0
FTP-Patator	1	0	0	0	0	0	0	0	1586	0
PortScan	3	0	0	0	0	3	0	0	0	3175
SSH-Patator	2	0	0	0	0	0	0	0	0	0
XSS	5	0	0	0	0	0	0	0	0	0

In [99]:

```
# Precision,Recall,F1-score for Random Forest classifier on all labels.

precision, recall, fscore, support = score(y_validate.Label, y_predict)

d = {'attack': attack, 'precision': precision, 'recall' : recall, 'fscore': fscore}
```

```
results = pd.DataFrame(data=d)
results
```

Out[99]:

	attack	precision	recall	fscore
0	BENIGN	0.999859	0.999760	0.999810
1	Bot	0.989247	0.941176	0.964613
2	Brute Force	1.000000	0.993355	0.996667
3	DDoS	0.999961	0.999649	0.999805
4	DoS GoldenEye	1.000000	0.997570	0.998784
5	DoS Hulk	0.997723	0.999783	0.998752
6	DoS Slowhttptest	0.995450	0.994545	0.994998
7	DoS slowloris	0.998265	0.993097	0.995675
8	FTP-Patator	1.000000	0.999370	0.999685
9	PortScan	0.999874	0.999780	0.999827
10	SSH-Patator	1.000000	0.998305	0.999152
11	XSS	0.984252	0.961538	0.972763

In [100…

```
# Average Accuracy,Precision,Recall,F1-score for Random Forest classifier on all labels

precision_rf_1, recall_rf_1, fscore_rf_1, support = score(y_validate.Label, y_predict,
accuracy_rf_1 = accuracy_score(y_validate.Label, y_predict)
print("Accuracy of Random Forest classifier on all labels : ", accuracy_rf_1)
```

```
Accuracy of Random Forest classifier on all labels :  0.9996693554376002
```

In [101…

```
# 1. b) Binary Classifier.
```

In [102…

```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Attack)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  1027.4533758163452
```

In [103…

```
# predicting test results of Random Forest classifier on binary labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  5.716964960098267
```

In [104…

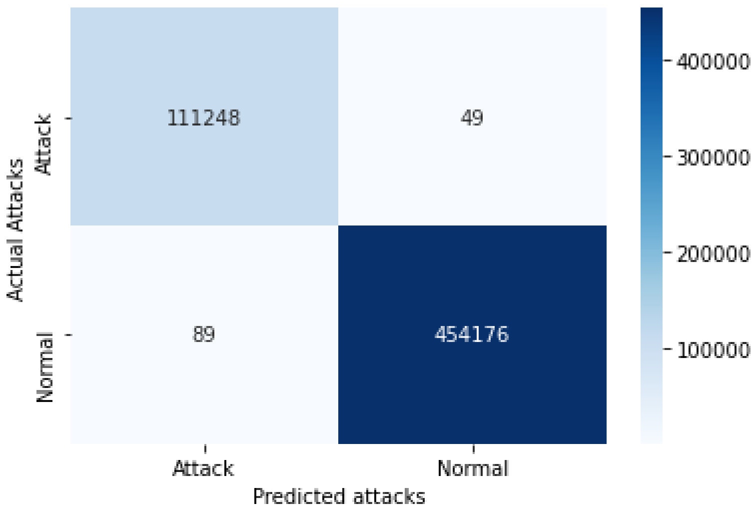
```
# Creating confusion matrix for Random Forest classifier on binary labels.

confusion_rf_2 = pd.crosstab(y_validate.Attack, y_predict, rownames=['Actual Attacks'],

print("Plotting Confusion Matrix of Random Forest classifier on binary Labels ")

sn.heatmap(confusion_rf_2, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_rf_2
```

```
Plotting Confusion Matrix of Random Forest classifier on binary Labels
```



Out[104…

Predicted attacks	Attack	Normal
Actual Attacks		
Attack	111248	49
Normal	89	454176

In [105…

```
# Precision,Recall,F1-score for Random Forest classifier on binary labels.

precision, recall, fscore, support = score(y_validate.Attack, y_predict)
d = {'attack': [0,1], 'precision': precision, 'recall' : recall, 'fscore': fscore}
results = pd.DataFrame(data=d)
results
```

Out[105…

	attack	precision	recall	fscore
0	0	0.999201	0.999560	0.999380
1	1	0.999892	0.999804	0.999848

In [106…

```
# Average Accuracy,Precision,Recall,F1-score for Random Forest classifier on binary lab

precision_rf_2, recall_rf_2, fscore_rf_2, n = score(y_validate.Attack, y_predict, avera
accuracy_rf_2 = accuracy_score(y_validate.Attack, y_predict)
print("Accuracy of Random Forest classifier on binary labels : ", accuracy_rf_2)
```

```
Accuracy of Random Forest classifier on binary labels :  0.9997559949218653
```

In [107…

```
# 1. c) Multi-class Classifier.
```

In [108…

```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Label_Category)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

```
Model Training Time is :  767.8004326820374
```

In [109…

```
# predicting test results of Random Forest classifier on multi-class labels.

start = time.time()
y_predict = classifier.predict(x_validate)
end = time.time()
testing_time = end  - start

print("Model Testing Time is : ", testing_time)
```

```
Model Testing Time is :  6.951459169387817
```

In [110…

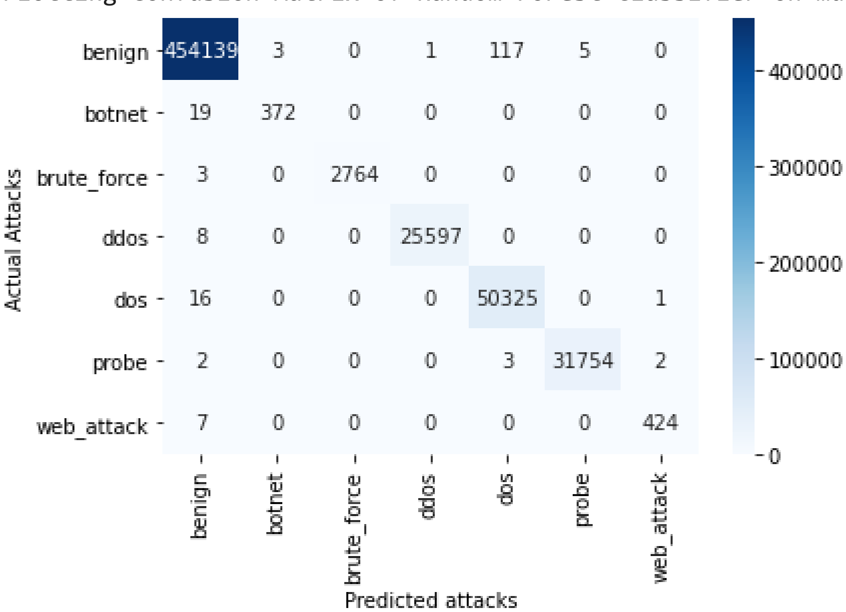
```
# Creating confusion matrix for Random Forest classifier on multi-class labels.

confusion_rf_3 = pd.crosstab(y_validate.Label_Category, y_predict, rownames=['Actual At

print("Plotting Confusion Matrix of Random Forest classifier on multi-class Labels ")

sn.heatmap(confusion_rf_3, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_rf_3
```

```
Plotting Confusion Matrix of Random Forest classifier on multi-class Labels
```



Out[110…

Predicted attacks	benign	botnet	brute_force	ddos	dos	probe	web_attack
Actual Attacks							
benign	454139	3	0	1	117	5	0
botnet	19	372	0	0	0	0	0
brute_force	3	0	2764	0	0	0	0
ddos	8	0	0	25597	0	0	0
dos	16	0	0	0	50325	0	1
probe	2	0	0	0	3	31754	2
web_attack	7	0	0	0	0	0	424

In [111…

```
# Precision,Recall,F1-score for Random Forest classifier on multi-class labels.

precision, recall, fscore, support = score(y_validate.Label_Category, y_predict)
d = {'attack': attack_groups, 'precision': precision, 'recall' : recall, 'fscore': fsco
results = pd.DataFrame(data=d)
results
```

Out[111…

	attack	precision	recall	fscore
0	benign	0.999879	0.999723	0.999801
1	botnet	0.992000	0.951407	0.971279
2	brute_force	1.000000	0.998916	0.999458
3	ddos	0.999961	0.999688	0.999824
4	dos	0.997621	0.999662	0.998641
5	probe	0.999843	0.999780	0.999811
6	web_attack	0.992974	0.983759	0.988345

In [112…

```
# Average Accuracy,Precision,Recall,F1-score for Random Forest classifier on multi-clas

precision_rf_3, recall_rf_3, fscore_rf_3, n = score(y_validate.Label_Category, y_predic
accuracy_rf_3 = accuracy_score(y_validate.Label_Category, y_predict)
print("Accuracy of Random Forest classifier on multi-class labels : ", accuracy_rf_3)
```

```
Accuracy of Random Forest classifier on multi-class labels :  0.9996693554376002
```

In [113…

```
### Results for Random Forest:
```

In [114…

```
print('Random Forest Classifier : Precision / Recall / Fscore / Accuracy')

print('All Labels:', precision_rf_1, recall_rf_1, fscore_rf_1,  accuracy_rf_1)
print('Binary Labels:', precision_rf_2, recall_rf_2, fscore_rf_2, accuracy_rf_2)
print('Multi-class Labels:', precision_rf_3, recall_rf_3, fscore_rf_3, accuracy_rf_3)
```

```
Random Forest Classifier : Precision / Recall / Fscore / Accuracy
```

```
All Labels: 0.9970527078650818 0.9898274730682263 0.9933773851343433 0.9996693554376002
Binary Labels: 0.9995463745382478 0.9996819078387745 0.9996141240892522 0.99975599492186
53
Multi-class Labels: 0.997468259368538 0.990419035313329 0.9938798193867859 0.99966935543
76002
```

In [ ]:

```

```