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 [7]:

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

In [8]:

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

```
(2827876, 79)
```

In [9]:

```
# 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 [10]:

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

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

In [11]:

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

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

Out[11]:

	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 [12]:

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

Out[12]:

```
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 [13]:

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

Out[13]:

```
(2827876, 78)
```

In [14]:

```
# check if there are any Null values
dataset.isnull().any().any()
```

Out[14]:

```
False
```

In [15]:

```
# 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[15]:

```
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 [16]:

```
# Distribution of Dataset
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
Infiltration                     36
Web Attack � Sql Injection       21
Heartbleed                       11
Name: Label, dtype: int64
```

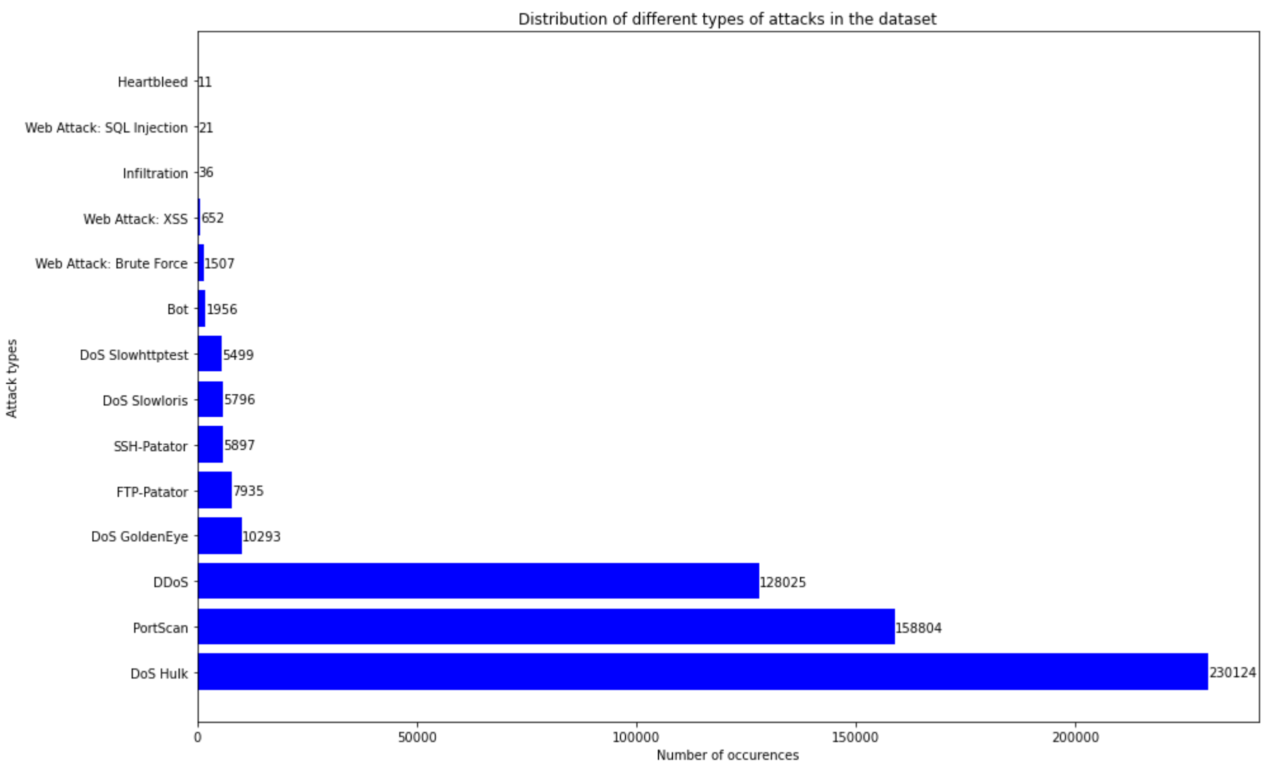
In [17]:

```
# 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 [18]:

```
# 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[18]:

```
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 [19]:

```
# 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 [20]:

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

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

In [21]:

```
# 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[21]:

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

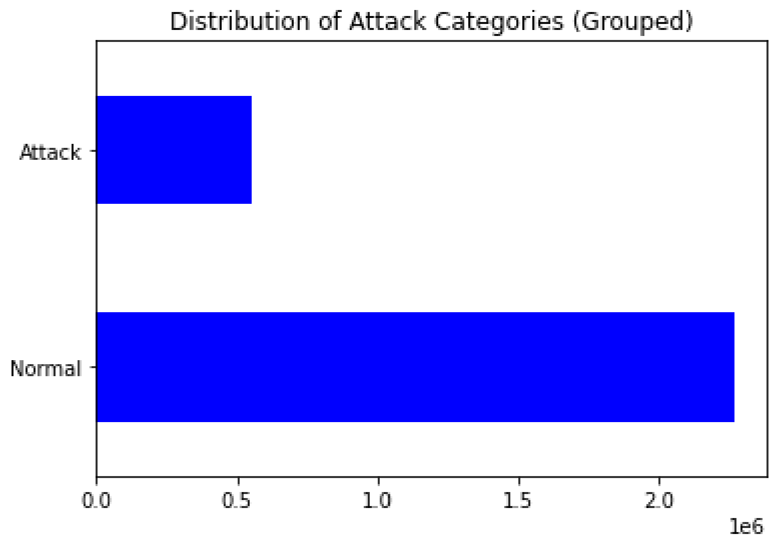
In [22]:

```
# 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[22]:

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

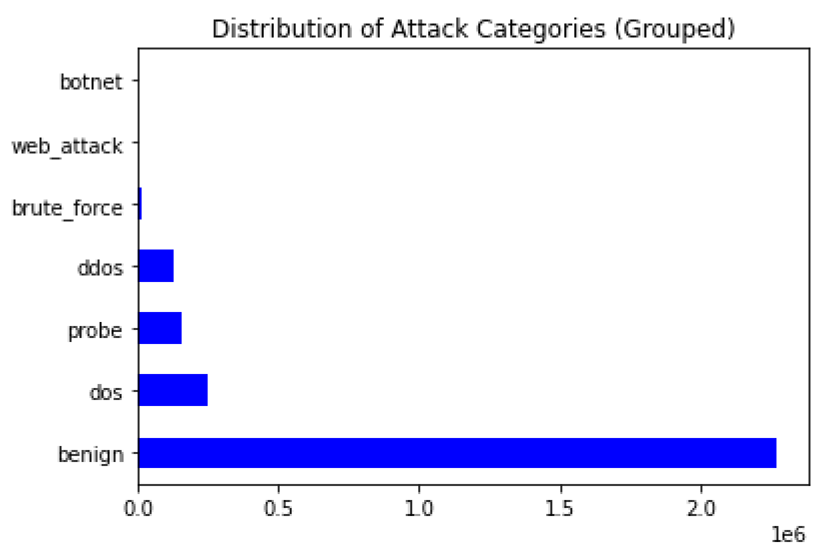


In [23]:

```
# 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[23]: Text(0.5, 1.0, 'Distribution of Attack Categories (Grouped)')

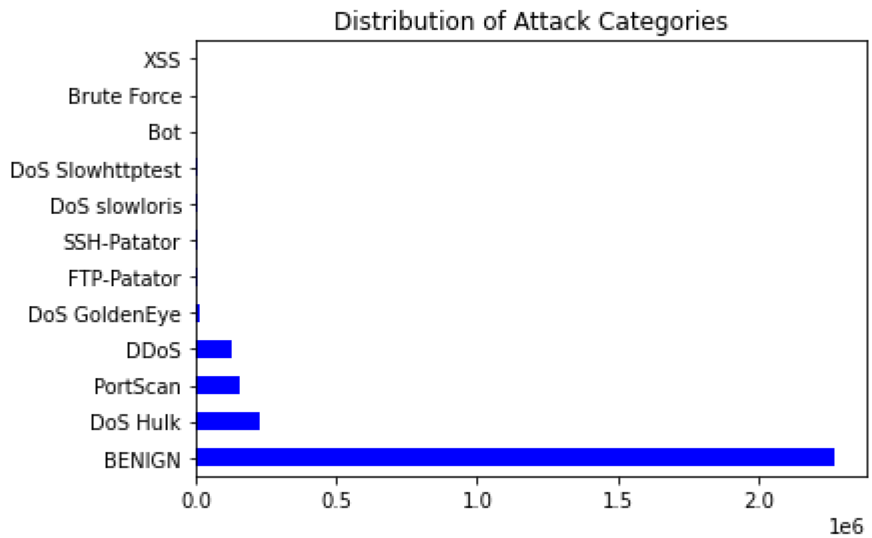


In [24]:

```
# 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[24]: Text(0.5, 1.0, 'Distribution of Attack Categories')



In [25]:

```
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 [26]:

```
# 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_test, y_train, y_test = train_test_split(xs, ys, test_size=0.3, random_state
```

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

In [27]:

```
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[27]:

```
['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 [28]:

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

In [29]:

```
x_train.shape
```

Out[29]:

```
(1979465, 69)
```

Data Normalization

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

In [30]:

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

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

Feature Selection

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

In [31]:

```
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 [32]:

```
# 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)
```

In [33]:

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

In [34]:

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

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

Out[34]:
```
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 [35]:
```
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 [36]:
```
classifier = LinearSVC()
```

1. a) On all attack labels.

In [37]:
```
# 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 :  475.13808488845825
```

In [38]:

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

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

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

```
Model Testing Time is :  0.1943984031677246
```

In [39]:

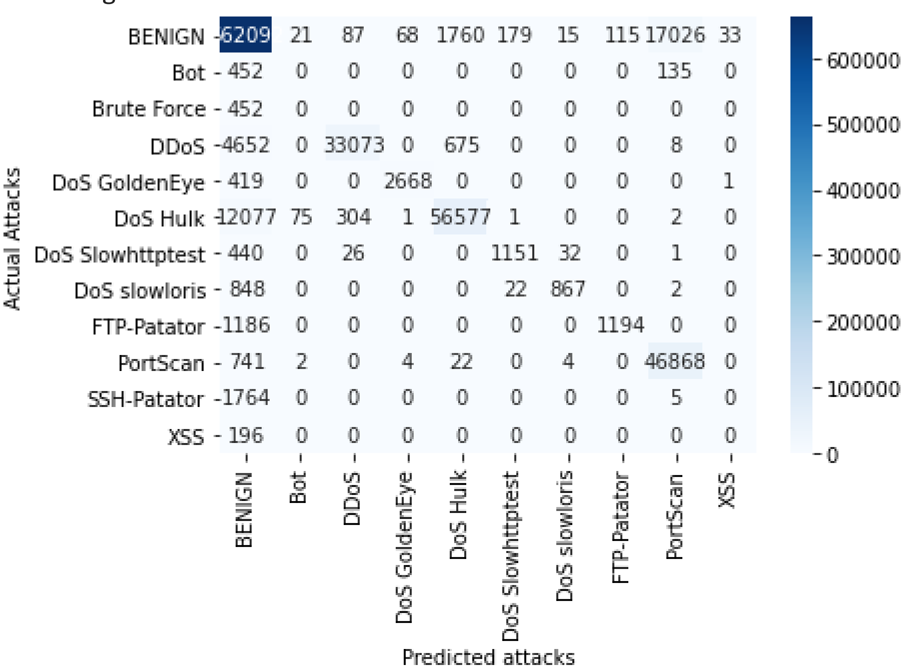
```python
# Creating confusion matrix for SVM classifier on all labels.

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

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[39]:

Predicted attacks / Actual Attacks	BENIGN	Bot	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	PortScan	XSS
BENIGN	662092	21	87	68	1760	179	15	115	17026	33
Bot	452	0	0	0	0	0	0	0	135	0
Brute Force	452	0	0	0	0	0	0	0	0	0
DDoS	4652	0	33073	0	675	0	0	0	8	0

Predicted attacks Actual Attacks	BENIGN	Bot	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	PortScan	XSS
DoS GoldenEye	419	0	0	2668	0	0	0	0	0	1
DoS Hulk	12077	75	304	1	56577	1	0	0	2	0
DoS Slowhttptest	440	0	26	0	0	1151	32	0	1	0
DoS slowloris	848	0	0	0	0	22	867	0	2	0
FTP-Patator	1186	0	0	0	0	0	0	1194	0	0
PortScan	741	2	0	4	22	0	4	0	46868	0
SSH-Patator	1764	0	0	0	0	0	0	0	5	0
XSS	196	0	0	0	0	0	0	0	0	0

In [40]:

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

precision, recall, fscore, support = score(y_test.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[40]:

	attack	precision	recall	fscore
0	BENIGN	0.966108	0.971670	0.968881
1	Bot	0.000000	0.000000	0.000000
2	Brute Force	0.000000	0.000000	0.000000
3	DDoS	0.987549	0.861097	0.919998
4	DoS GoldenEye	0.973367	0.863990	0.915423
5	DoS Hulk	0.958380	0.819517	0.883526
6	DoS Slowhttptest	0.850702	0.697576	0.766567
7	DoS slowloris	0.944444	0.498562	0.652616
8	FTP-Patator	0.912147	0.501681	0.647330
9	PortScan	0.731775	0.983774	0.839267
10	SSH-Patator	0.000000	0.000000	0.000000
11	XSS	0.000000	0.000000	0.000000

In [41]:

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

precision_svm_1, recall_svm_1, fscore_svm_1, support = score(y_test.Label, y_predict, a
accuracy_svm_1 = accuracy_score(y_test.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 :  0.9483074652587455
```

In [ ]:

1. b) Binary Classifier.

In [42]:

```
# 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 :  272.09957218170166
```

In [43]:

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

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

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

```
Model Testing Time is :  0.14150285720825195
```

In [44]:

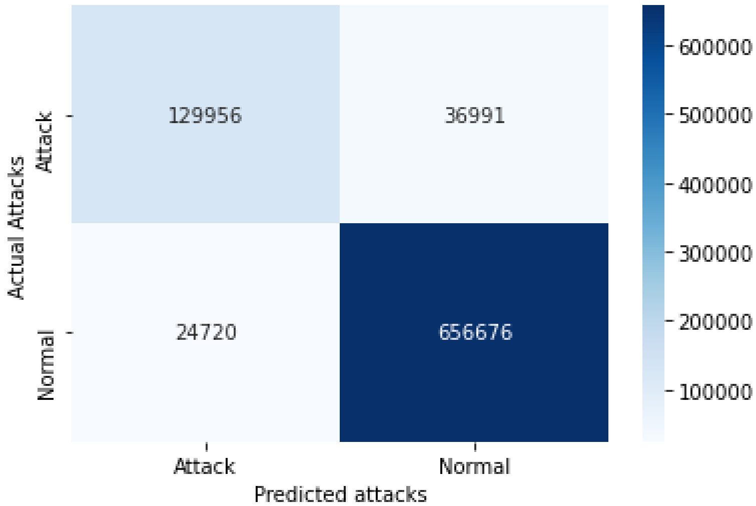
```
# Creating confusion matrix for SVM classifier on binary labels.

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

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[44]:

Predicted attacks	Attack	Normal
Actual Attacks		
Attack	129956	36991
Normal	24720	656676

In [45]:

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

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

Out[45]:

	attack	precision	recall	fscore
0	0	0.840182	0.778427	0.808126
1	1	0.946673	0.963722	0.955121

In [46]:

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

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

```
Accuracy of SVM classifier on binary labels :  0.9272570175035334
```

In [ ]:

1. c) Multi-class Classifier.

In [47]:

```
# 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 :  421.0585870742798
```

In [48]:

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

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

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

```
Model Testing Time is :  0.14888381958007812
```

In [49]:

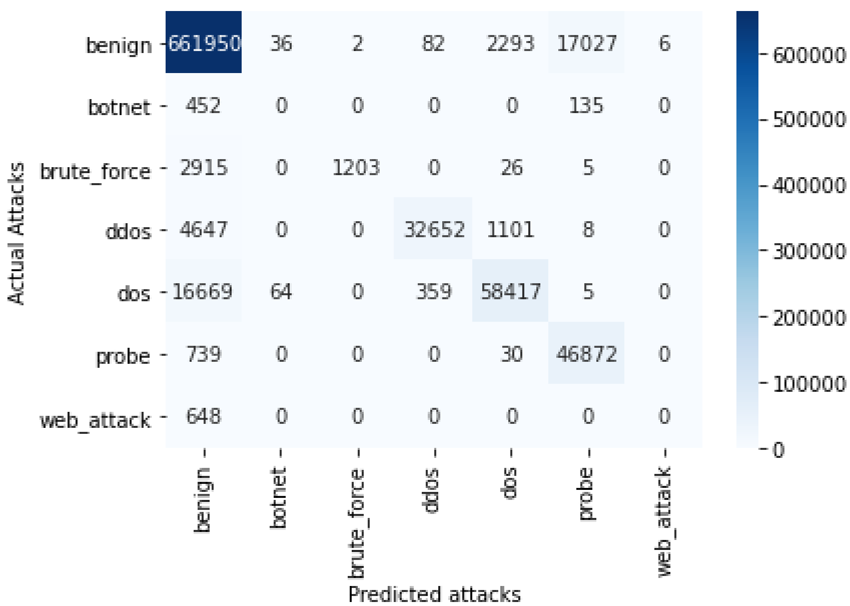
```
# Creating confusion matrix for SVM classifier on multi-class labels.

confusion_svm_3 = pd.crosstab(y_test.Label_Category, y_predict, rownames=['Actual Attac

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[49]:

Predicted attacks / Actual Attacks	benign	botnet	brute_force	ddos	dos	probe	web_attack
benign	661950	36	2	82	2293	17027	6
botnet	452	0	0	0	0	135	0
brute_force	2915	0	1203	0	26	5	0
ddos	4647	0	0	32652	1101	8	0
dos	16669	64	0	359	58417	5	0

Predicted attacks	benign	botnet	brute_force	ddos	dos	probe	web_attack
Actual Attacks							
probe	739	0	0	0	30	46872	0
web_attack	648	0	0	0	0	0	0

In [50]:
```
# Precision,Recall,F1-score for SVM classifier on multi-class labels.

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

Out[50]:

	attack	precision	recall	fscore
0	benign	0.962109	0.971462	0.966762
1	botnet	0.000000	0.000000	0.000000
2	brute_force	0.998340	0.289949	0.449384
3	ddos	0.986674	0.850135	0.913330
4	dos	0.944235	0.773592	0.850438
5	probe	0.731780	0.983858	0.839301
6	web_attack	0.000000	0.000000	0.000000

In [51]:
```
# 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_test.Label_Category, y_predict
accuracy_svm_3 = accuracy_score(y_test.Label_Category, y_predict)
print("Accuracy of SVM classifier on multi-class labels : ", accuracy_svm_3)
```

```
Accuracy of SVM classifier on multi-class labels :  0.9443043674551449
```

In [ ]:

Results for SVM:

In [52]:
```
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: 0.6103726620259674 0.5164888817555738 0.5494671656464464 0.9483074652587455
Binary Labels: 0.8934276588134502 0.871074111344472 0.8816238067435893 0.927257017503533
4
Multi-class Labels: 0.6604483528860229 0.5527137708530362 0.5741734874496957 0.944304367
4551449
```

In [ ]:

In [ ]:

2. Decision Tree

In [53]:

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

1. a) On all attack labels.

In [54]:

```
# 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 :  120.84963846206665
```

In [55]:

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

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

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

```
Model Testing Time is :  0.19506096839904785
```

In [56]:

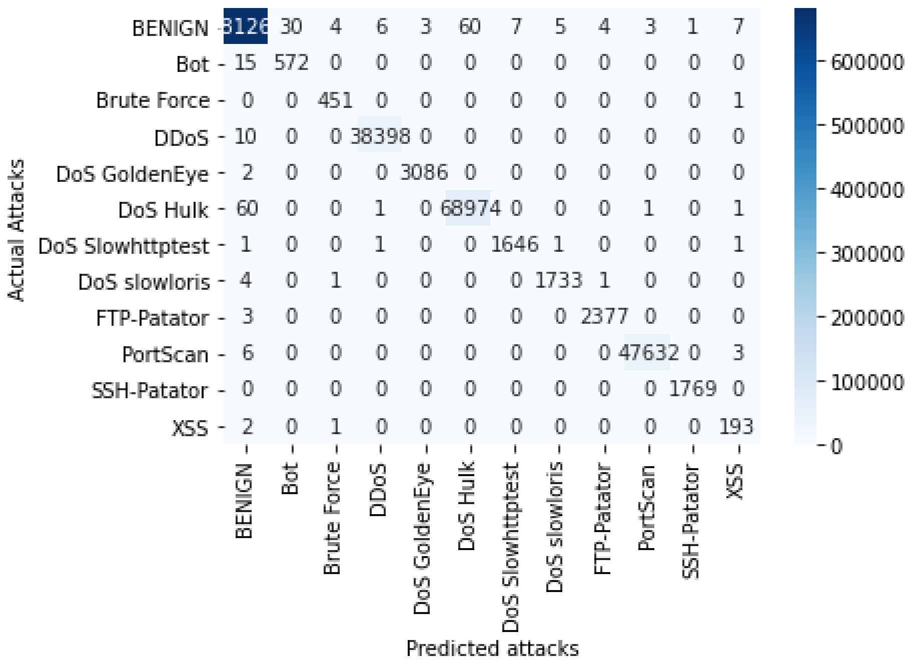
```
# Creating confusion matrix for Decision Tree classifier on all labels.

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

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[56]:

Predicted attacks / Actual Attacks	BENIGN	Bot	Brute Force	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	PortScan
BENIGN	681266	30	4	6	3	60	7	5	4	3
Bot	15	572	0	0	0	0	0	0	0	0
Brute Force	0	0	451	0	0	0	0	0	0	0
DDoS	10	0	0	38398	0	0	0	0	0	0
DoS GoldenEye	2	0	0	0	3086	0	0	0	0	0
DoS Hulk	60	0	0	1	0	68974	0	0	0	1
DoS Slowhttptest	1	0	0	1	0	0	1646	1	0	0
DoS slowloris	4	0	1	0	0	0	0	1733	1	0
FTP-Patator	3	0	0	0	0	0	0	0	2377	0
PortScan	6	0	0	0	0	0	0	0	0	4763
SSH-Patator	0	0	0	0	0	0	0	0	0	0
XSS	2	0	1	0	0	0	0	0	0	0

In [57]:

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

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

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

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

Out[57]:

	attack	precision	recall	fscore
0	BENIGN	0.999849	0.999809	0.999829
1	Bot	0.950166	0.974446	0.962153
2	Brute Force	0.986871	0.997788	0.992299
3	DDoS	0.999792	0.999740	0.999766
4	DoS GoldenEye	0.999029	0.999352	0.999191
5	DoS Hulk	0.999131	0.999087	0.999109
6	DoS Slowhttptest	0.995765	0.997576	0.996670
7	DoS slowloris	0.996550	0.996550	0.996550
8	FTP-Patator	0.997901	0.998739	0.998320
9	PortScan	0.999916	0.999811	0.999864
10	SSH-Patator	0.999435	1.000000	0.999717
11	XSS	0.936893	0.984694	0.960199

In [58]:

```
# 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_test.Label, y_predict, aver
accuracy_dt_1 = accuracy_score(y_test.Label, y_predict)
print("Accuracy of Decision Tree classifier on all labels : ", accuracy_dt_1)
```

```
Accuracy of Decision Tree classifier on all labels :  0.9997100229506226
```

In [59]:

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

In [60]:

```
# 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 :  119.4067771434784
```

In [61]:

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

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

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

```
Model Testing Time is :  0.19147992134094238
```

In [62]:

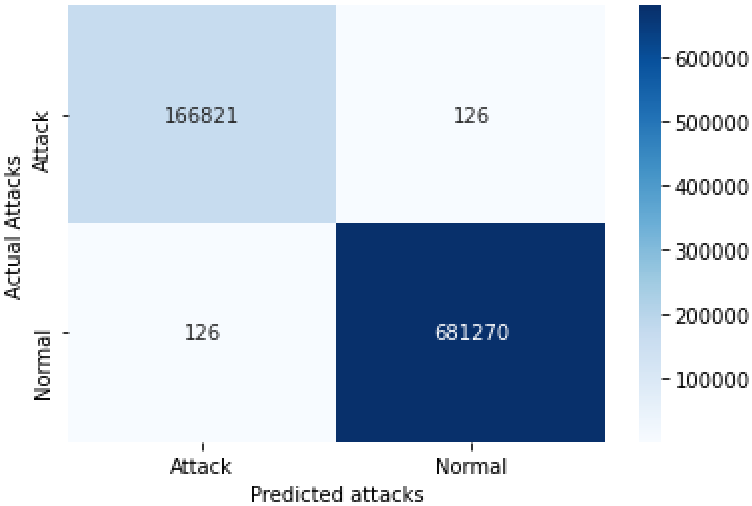
```
# Creating confusion matrix for Decision Tree classifier on binary labels.

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

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[62]:

Predicted attacks	Attack	Normal
Actual Attacks		
Attack	166821	126
Normal	126	681270

In [63]:

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

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

Out[63]:

	attack	precision	recall	fscore
0	0	0.999245	0.999245	0.999245
1	1	0.999815	0.999815	0.999815

In [64]:

```
# 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_test.Attack, y_predict, average='
accuracy_dt_2 = accuracy_score(y_test.Attack, y_predict)
print("Accuracy of Decision Tree classifier on binary labels : ", accuracy_dt_2)
```

```
Accuracy of Decision Tree classifier on binary labels :  0.9997029503396622
```

In [65]:
```
# 1. c) Multi-class Classifier.
```

In [66]:
```
# 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 :  126.87579131126404
```

In [67]:
```
# predicting test results of Decision Tree classifier on multi-class labels.

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

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

```
Model Testing Time is :  0.20798730850219727
```

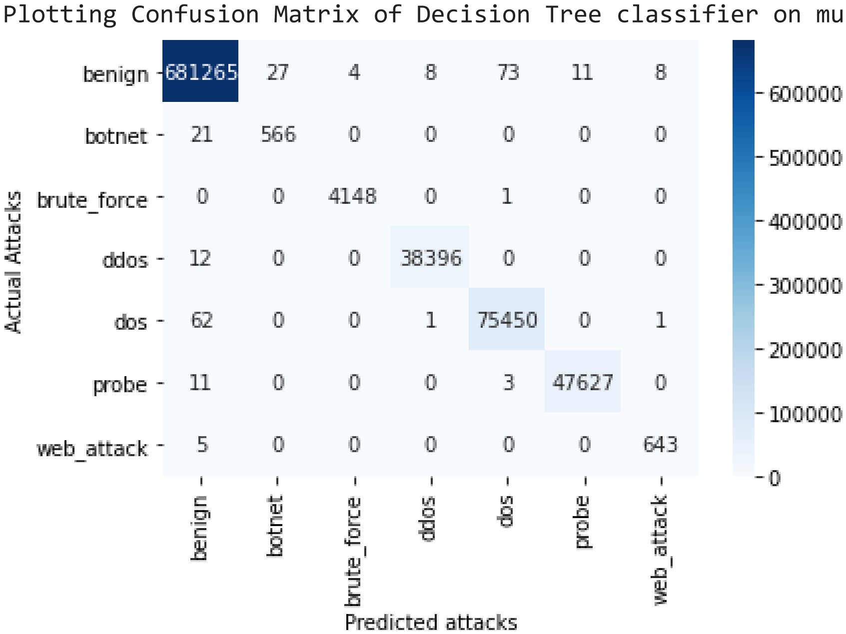
In [68]:
```
# Creating confusion matrix for Decision Tree classifier on multi-class labels.

confusion_dt_3 = pd.crosstab(y_test.Label_Category, y_predict, rownames=['Actual Attack

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[68]:

Predicted attacks	benign	botnet	brute_force	ddos	dos	probe	web_attack
Actual Attacks							
benign	681265	27	4	8	73	11	8
botnet	21	566	0	0	0	0	0
brute_force	0	0	4148	0	1	0	0
ddos	12	0	0	38396	0	0	0
dos	62	0	0	1	75450	0	1
probe	11	0	0	0	3	47627	0
web_attack	5	0	0	0	0	0	643

In [69]:

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

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

Out[69]:

	attack	precision	recall	fscore
0	benign	0.999837	0.999808	0.999822
1	botnet	0.954469	0.964225	0.959322
2	brute_force	0.999037	0.999759	0.999398
3	ddos	0.999766	0.999688	0.999727
4	dos	0.998980	0.999152	0.999066
5	probe	0.999769	0.999706	0.999738
6	web_attack	0.986196	0.992284	0.989231

In [70]:

```
# 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_test.Label_Category, y_predict, a
accuracy_dt_3 = accuracy_score(y_test.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 :  0.9997076654136358
```

Results for Decision Tree:

In [71]:

```
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: 0.9884414516786525 0.9956327114513193 0.9919721795956459 0.9997100229506226
```

```
Binary Labels: 0.999530177478858 0.999530177478858 0.999530177478858 0.9997029503396622
Multi-class Labels: 0.9911505813408323 0.9935173891628378 0.9923290838491127 0.999707665
4136358
```

In [72]:

```
# 3. Naive Bayes Classifier
```

In [73]:

```
classifier = MultinomialNB()
```

In [74]:

```
# 1. a) On all attack labels.
```

In [75]:

```
# 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 :  14.33758020401001
```

In [76]:

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

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

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

```
Model Testing Time is :  0.7225887775421143
```

In [77]:

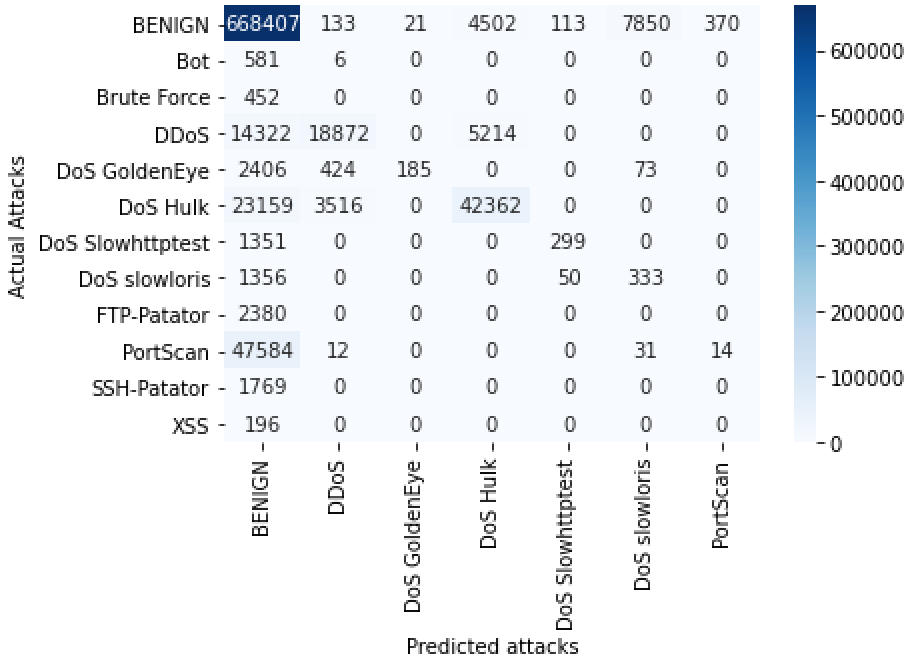
```
# Creating confusion matrix for Naive Bayes classifier on all labels.

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

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[77]:

Predicted attacks / Actual Attacks	BENIGN	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	PortScan
BENIGN	668407	133	21	4502	113	7850	370
Bot	581	6	0	0	0	0	0
Brute Force	452	0	0	0	0	0	0
DDoS	14322	18872	0	5214	0	0	0
DoS GoldenEye	2406	424	185	0	0	73	0
DoS Hulk	23159	3516	0	42362	0	0	0
DoS Slowhttptest	1351	0	0	0	299	0	0
DoS slowloris	1356	0	0	0	50	333	0
FTP-Patator	2380	0	0	0	0	0	0
PortScan	47584	12	0	0	0	31	14
SSH-Patator	1769	0	0	0	0	0	0
XSS	196	0	0	0	0	0	0

In [78]:

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

precision, recall, fscore, support = score(y_test.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[78]:

	attack	precision	recall	fscore
0	BENIGN	0.874921	0.980938	0.924901
1	Bot	0.000000	0.000000	0.000000
2	Brute Force	0.000000	0.000000	0.000000
3	DDoS	0.821844	0.491356	0.615014
4	DoS GoldenEye	0.898058	0.059909	0.112325
5	DoS Hulk	0.813434	0.613613	0.699534
6	DoS Slowhttptest	0.647186	0.181212	0.283144
7	DoS slowloris	0.040183	0.191489	0.066427
8	FTP-Patator	0.000000	0.000000	0.000000
9	PortScan	0.036458	0.000294	0.000583
10	SSH-Patator	0.000000	0.000000	0.000000
11	XSS	0.000000	0.000000	0.000000

In [79]:

```
# 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_test.Label, y_predict, aver
accuracy_nb_1 = accuracy_score(y_test.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 :  0.8610573789139534
```

In [80]:

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

In [81]:

```
# 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 :  44.21700477600098
```

In [82]:

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

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

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

```
Model Testing Time is :  2.2370920181274414
```

In [83]:

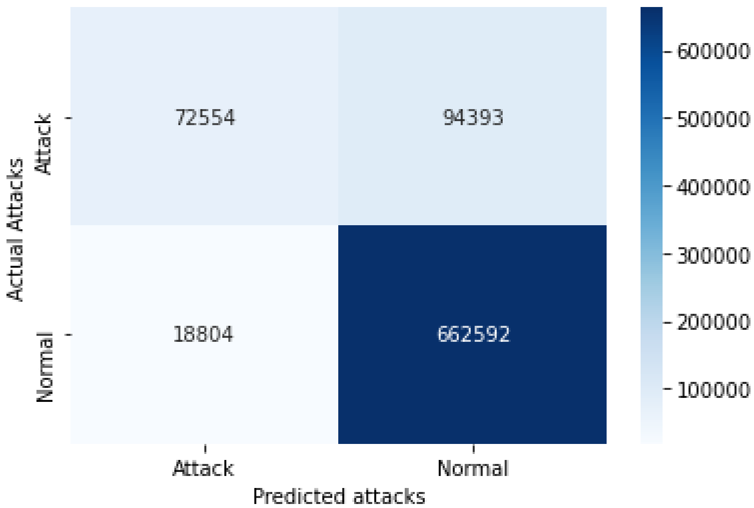
```
# Creating confusion matrix for Naive Bayes classifier on binary labels.

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

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[83]:

Predicted attacks	Attack	Normal
Actual Attacks		
Attack	72554	94393
Normal	18804	662592

In [84]:

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

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

Out[84]:

	attack	precision	recall	fscore
0	0	0.794172	0.434593	0.561770
1	1	0.875304	0.972404	0.921302

In [85]:

```
# 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_test.Attack, y_predict, average='
```

```
accuracy_nb_2 = accuracy_score(y_test.Attack, y_predict)
print("Accuracy of Naive Bayes classifier on binary labels : ", accuracy_nb_2)
```

```
Accuracy of Naive Bayes classifier on binary labels :  0.8665669428521247
```

In [86]:

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

In [87]:

```
# 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 :  8.498613119125366
```

In [88]:

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

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

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

```
Model Testing Time is :  0.1821749210357666
```

In [89]:

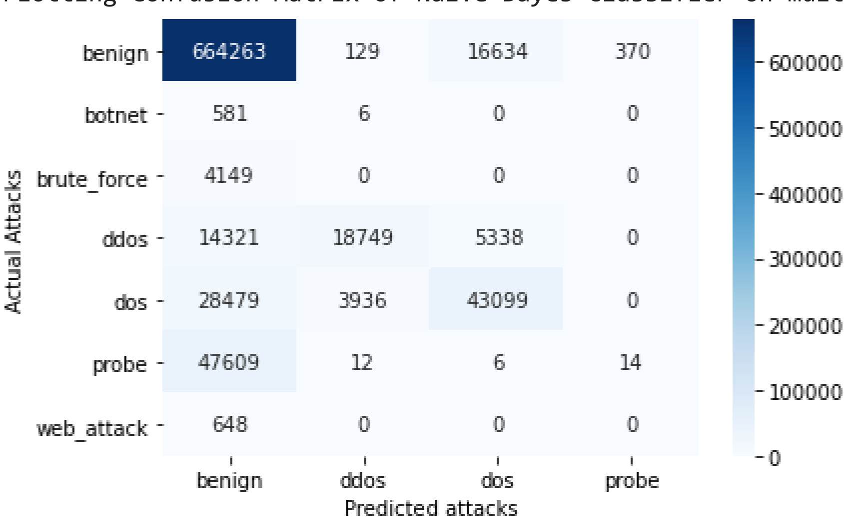
```
# Creating confusion matrix for Naive Bayes classifier on multi-class labels.

confusion_nb_3 = pd.crosstab(y_test.Label_Category, y_predict, rownames=['Actual Attack

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[89]:

Predicted attacks Actual Attacks	benign	ddos	dos	probe
benign	664263	129	16634	370
botnet	581	6	0	0
brute_force	4149	0	0	0
ddos	14321	18749	5338	0
dos	28479	3936	43099	0
probe	47609	12	6	14
web_attack	648	0	0	0

In [90]:

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

precision, recall, fscore, support = score(y_test.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[90]:

	attack	precision	recall	fscore
0	benign	0.873973	0.974856	0.921662
1	botnet	0.000000	0.000000	0.000000
2	brute_force	0.000000	0.000000	0.000000
3	ddos	0.821172	0.488154	0.612312
4	dos	0.662277	0.570742	0.613112
5	probe	0.036458	0.000294	0.000583
6	web_attack	0.000000	0.000000	0.000000

In [91]:

```
# 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_test.Label_Category, y_predict, a
accuracy_nb_3 = accuracy_score(y_test.Label_Category, y_predict)
print("Accuracy of Naive Bayes classifier on multi-class labels : ", accuracy_nb_3)
```

```
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 multi-class labels :  0.8559332722731253
```

In [92]:

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

In [93]:
```
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: 0.3443403606808013 0.20990094123148562 0.22516065133780527 0.861057378913953
4
Binary Labels: 0.834738189432076 0.7034983634641163 0.7415362463788386 0.866566942852124
7
Multi-class Labels: 0.34198287606723227 0.2905778934659667 0.3068098620024685 0.85593327
22731253
```

In [94]:
```
# 6. Random Forest Classifier
```

In [95]:
```
classifier =  RandomForestClassifier()
```

In [96]:
```
# 1. a) On all attack labels.
```

In [97]:
```
# 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 :  1213.830884218216
```

In [98]:
```
# predicting test results of Random Forest classifier on all labels.

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

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

```
Model Testing Time is :  13.517748594284058
```

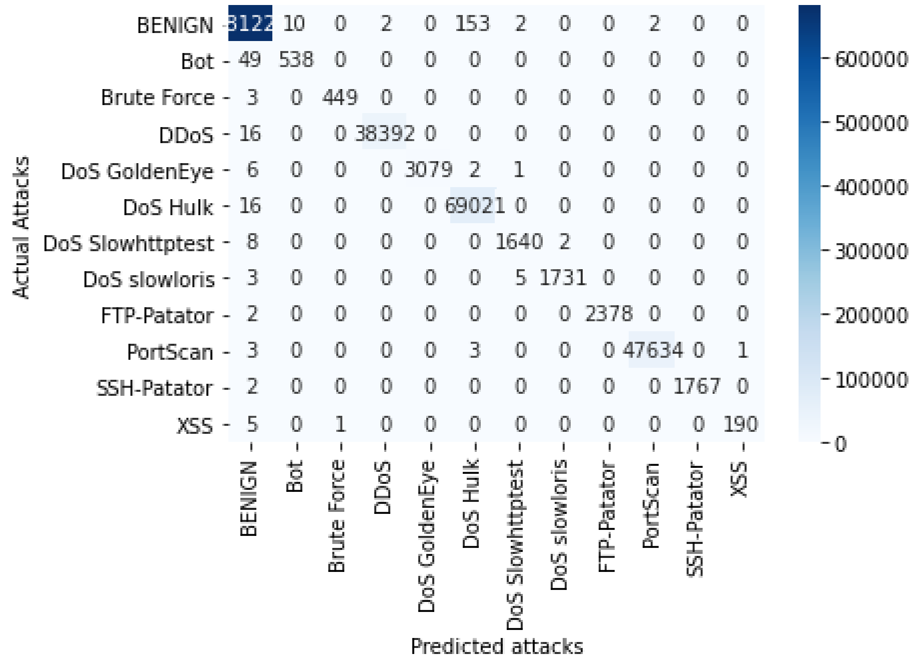
In [99]:
```
# Creating confusion matrix for Random Forest classifier on all labels.

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

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[99]:

Predicted attacks	BENIGN	Bot	Brute Force	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	FTP-Patator	PortScan
Actual Attacks										
BENIGN	681227	10	0	2	0	153	2	0	0	2
Bot	49	538	0	0	0	0	0	0	0	0
Brute Force	3	0	449	0	0	0	0	0	0	0
DDoS	16	0	0	38392	0	0	0	0	0	0
DoS GoldenEye	6	0	0	0	3079	2	1	0	0	0
DoS Hulk	16	0	0	0	0	69021	0	0	0	0
DoS Slowhttptest	8	0	0	0	0	0	1640	2	0	0
DoS slowloris	3	0	0	0	0	0	5	1731	0	0
FTP-Patator	2	0	0	0	0	0	0	0	2378	0
PortScan	3	0	0	0	0	3	0	0	0	4763
SSH-Patator	2	0	0	0	0	0	0	0	0	0
XSS	5	0	1	0	0	0	0	0	0	0

In [100…

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

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

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

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

Out[100…

	attack	precision	recall	fscore
0	BENIGN	0.999834	0.999752	0.999793
1	Bot	0.981752	0.916525	0.948018
2	Brute Force	0.997778	0.993363	0.995565
3	DDoS	0.999948	0.999583	0.999766
4	DoS GoldenEye	1.000000	0.997085	0.998541
5	DoS Hulk	0.997716	0.999768	0.998741
6	DoS Slowhttptest	0.995146	0.993939	0.994542
7	DoS slowloris	0.998846	0.995400	0.997120
8	FTP-Patator	1.000000	0.999160	0.999580
9	PortScan	0.999958	0.999853	0.999906
10	SSH-Patator	1.000000	0.998869	0.999434
11	XSS	0.994764	0.969388	0.981912

In [101…

```
# 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_test.Label, y_predict, aver
accuracy_rf_1 = accuracy_score(y_test.Label, y_predict)
print("Accuracy of Random Forest classifier on all labels : ", accuracy_rf_1)
```

```
Accuracy of Random Forest classifier on all labels :  0.9996499057574589
```

In [102…

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

In [103…

```
# 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 :  1065.3459088802338
```

In [104…

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

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

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

```
Model Testing Time is :  11.010573148727417
```

In [105…

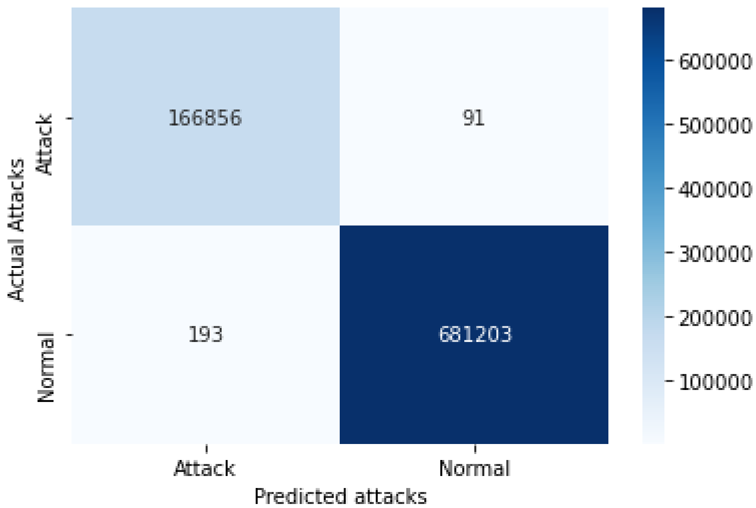
```
# Creating confusion matrix for Random Forest classifier on binary labels.

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

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[105…

Predicted attacks	Attack	Normal
Actual Attacks		
Attack	166856	91
Normal	193	681203

In [106…

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

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

Out[106…

	attack	precision	recall	fscore
0	0	0.998845	0.999455	0.999150
1	1	0.999866	0.999717	0.999792

In [107…

```
# 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_test.Attack, y_predict, average='
accuracy_rf_2 = accuracy_score(y_test.Attack, y_predict)
print("Accuracy of Random Forest classifier on binary labels : ", accuracy_rf_2)
```

```
Accuracy of Random Forest classifier on binary labels :  0.9996652297478732
```

In [108…

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

In [109…

```python
# 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 :  1315.117155790329
```

In [110…

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

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

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

```
Model Testing Time is :  14.242902278900146
```

In [111…

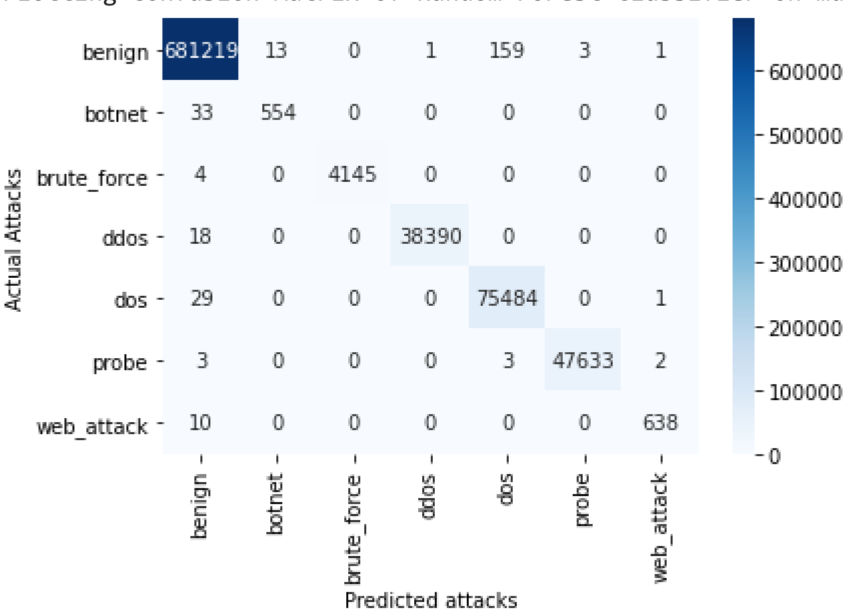
```python
# Creating confusion matrix for Random Forest classifier on multi-class labels.

confusion_rf_3 = pd.crosstab(y_test.Label_Category, y_predict, rownames=['Actual Attack

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[111…

Predicted attacks Actual Attacks	benign	botnet	brute_force	ddos	dos	probe	web_attack
benign	681219	13	0	1	159	3	1
botnet	33	554	0	0	0	0	0
brute_force	4	0	4145	0	0	0	0
ddos	18	0	0	38390	0	0	0
dos	29	0	0	0	75484	0	1
probe	3	0	0	0	3	47633	2
web_attack	10	0	0	0	0	0	638

In [112…

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

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

Out[112…

	attack	precision	recall	fscore
0	benign	0.999858	0.999740	0.999799
1	botnet	0.977072	0.943782	0.960139
2	brute_force	1.000000	0.999036	0.999518
3	ddos	0.999974	0.999531	0.999753
4	dos	0.997858	0.999603	0.998730
5	probe	0.999937	0.999832	0.999885
6	web_attack	0.993769	0.984568	0.989147

In [113…

```
# 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_test.Label_Category, y_predict, a
accuracy_rf_3 = accuracy_score(y_test.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 :  0.9996699448218468
```

In [114…

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

In [115…

```
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: 0.9971451422674532 0.9885571349521968 0.9927430946915207 0.9996499057574589
Binary Labels: 0.99935554050863 0.999585837389874 0.9994706395556712 0.9996652297478732
Multi-class Labels: 0.9954955471214486 0.9894417346370281 0.9924242228340139 0.999669944
8218468
```