Import Necessary Library

Commonly used libraries are imported here. Libraries for models may be imported when they will be used in their cell.

```
import numpy as np
import pandas as pd
import os, re, time, math, tqdm, itertools
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import plotly.express as px
import plotly.offline as pyo
import seaborn as sns

from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE

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

from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import SequentialFeatureSelector

from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_scoreS
import lime
import lime.lime_tabular
from lime.lime_tabular import LimeTabularExplainer
import graphviz
import shap

import pickle

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

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[1], line 44
     42 from sklearn.metrics import classification_report
     43 from sklearn.metrics import confusion_matrix
---> 44 from sklearn.metrics import roc_auc_scoreS
     45 import lime
     46 import lime.lime_tabular

ImportError: cannot import name 'roc_auc_scoreS' from 'sklearn.metrics'
(C:\Users\asus\anaconda3\lib\site-packages\sklearn\metrics\__init__.py)
```

Start coding or generate with AI.

Load Dataset

```
for dirname, _, filenames in os.walk('./archive'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
./archive\02-14-2018.csv
./archive\02-15-2018.csv
./archive\02-16-2018.csv
./archive\02-20-2018.csv
./archive\02-21-2018.csv
./archive\02-22-2018.csv
./archive\02-23-2018.csv
./archive\02-28-2018.csv
./archive\03-01-2018.csv
./archive\03-02-2018.csv
```

Read Dataset

```
%%time
network_data_d1 = pd.read_csv("./archive/02-14-2018.csv", low_memory=False)
network_data_d2 = pd.read_csv("./archive/02-15-2018.csv", low_memory=False)
network_data_d3 = pd.read_csv("./archive/02-16-2018.csv", low_memory=False)
network_data_d4 = pd.read_csv("./archive/02-20-2018.csv", low_memory=False)
network_data_d5 = pd.read_csv("./archive/02-21-2018.csv", low_memory=False)
network_data_d6 = pd.read_csv("./archive/02-22-2018.csv", low_memory=False)
network_data_d7 = pd.read_csv("./archive/02-23-2018.csv", low_memory=False)
network_data_d8 = pd.read_csv("./archive/02-28-2018.csv", low_memory=False)
network_data_d9 = pd.read_csv("./archive/03-01-2018.csv", low_memory=False)
network_data_d10 = pd.read_csv("./archive/03-02-2018.csv", low_memory=False)
```

```
CPU times: total: 2min
Wall time: 5min 47s
```

```
network_data_d4.drop(columns=['Flow ID', 'Src IP', 'Src Port', 'Dst IP'], axis=1,inplace=True)
```

Data Type Fix

```
def fixDataType(df_dataset):

    df_dataset = df_dataset[df_dataset['Dst Port'] != 'Dst Port']

    df_dataset['Dst Port'] = df_dataset['Dst Port'].astype(int)
    df_dataset['Protocol'] = df_dataset['Protocol'].astype(int)
    df_dataset['Flow Duration'] = df_dataset['Flow Duration'].astype(int)
    df_dataset['Tot Fwd Pkts'] = df_dataset['Tot Fwd Pkts'].astype(int)
    df_dataset['Tot Bwd Pkts'] = df_dataset['Tot Bwd Pkts'].astype(int)
    df_dataset['TotLen Fwd Pkts'] = df_dataset['TotLen Fwd Pkts'].astype(int)
    df_dataset['TotLen Bwd Pkts'] = df_dataset['TotLen Bwd Pkts'].astype(int)
    df_dataset['Fwd Pkt Len Max'] = df_dataset['Fwd Pkt Len Max'].astype(int)
    df_dataset['Fwd Pkt Len Min'] = df_dataset['Fwd Pkt Len Min'].astype(int)
    df_dataset['Fwd Pkt Len Mean'] = df_dataset['Fwd Pkt Len Mean'].astype(float)
    df_dataset['Fwd Pkt Len Std'] = df_dataset['Fwd Pkt Len Std'].astype(float)
    df_dataset['Bwd Pkt Len Max'] = df_dataset['Bwd Pkt Len Max'].astype(int)
    df_dataset['Bwd Pkt Len Min'] = df_dataset['Bwd Pkt Len Min'].astype(int)
    df_dataset['Bwd Pkt Len Mean'] = df_dataset['Bwd Pkt Len Mean'].astype(float)
    df_dataset['Bwd Pkt Len Std'] = df_dataset['Bwd Pkt Len Std'].astype(float)
    df_dataset['Flow Byts/s'] = df_dataset['Flow Byts/s'].astype(float)
    df_dataset['Flow Pkts/s'] = df_dataset['Flow Pkts/s'].astype(float)
    df_dataset['Flow IAT Mean'] = df_dataset['Flow IAT Mean'].astype(float)
    df_dataset['Flow IAT Std'] = df_dataset['Flow IAT Std'].astype(float)
    df_dataset['Flow IAT Max'] = df_dataset['Flow IAT Max'].astype(int)
    df_dataset['Flow IAT Min'] = df_dataset['Flow IAT Min'].astype(int)
    df_dataset['Fwd IAT Tot'] = df_dataset['Fwd IAT Tot'].astype(int)
    df_dataset['Fwd IAT Mean'] = df_dataset['Fwd IAT Mean'].astype(float)
    df_dataset['Fwd IAT Std'] = df_dataset['Fwd IAT Std'].astype(float)
    df_dataset['Fwd IAT Max'] = df_dataset['Fwd IAT Max'].astype(int)
    df_dataset['Fwd IAT Min'] = df_dataset['Fwd IAT Min'].astype(int)
    df_dataset['Bwd IAT Tot'] = df_dataset['Bwd IAT Tot'].astype(int)
    df_dataset['Bwd IAT Mean'] = df_dataset['Bwd IAT Mean'].astype(float)
    df_dataset['Bwd IAT Std'] = df_dataset['Bwd IAT Std'].astype(float)
    df_dataset['Bwd IAT Max'] = df_dataset['Bwd IAT Max'].astype(int)
    df_dataset['Bwd IAT Min'] = df_dataset['Bwd IAT Min'].astype(int)
    df_dataset['Fwd PSH Flags'] = df_dataset['Fwd PSH Flags'].astype(int)
    df_dataset['Bwd PSH Flags'] = df_dataset['Bwd PSH Flags'].astype(int)
    df_dataset['Fwd URG Flags'] = df_dataset['Fwd URG Flags'].astype(int)
    df_dataset['Bwd URG Flags'] = df_dataset['Bwd URG Flags'].astype(int)
    df_dataset['Fwd Header Len'] = df_dataset['Fwd Header Len'].astype(int)
    df_dataset['Bwd Header Len'] = df_dataset['Bwd Header Len'].astype(int)
    df_dataset['Fwd Pkts/s'] = df_dataset['Fwd Pkts/s'].astype(float)
    df_dataset['Bwd Pkts/s'] = df_dataset['Bwd Pkts/s'].astype(float)
    df_dataset['Pkt Len Min'] = df_dataset['Pkt Len Min'].astype(int)
    df_dataset['Pkt Len Max'] = df_dataset['Pkt Len Max'].astype(int)
    df_dataset['Pkt Len Mean'] = df_dataset['Pkt Len Mean'].astype(float)
    df_dataset['Pkt Len Std'] = df_dataset['Pkt Len Std'].astype(float)
    df_dataset['Pkt Len Var'] = df_dataset['Pkt Len Var'].astype(float)
    df_dataset['FIN Flag Cnt'] = df_dataset['FIN Flag Cnt'].astype(int)
    df_dataset['SYN Flag Cnt'] = df_dataset['SYN Flag Cnt'].astype(int)
    df_dataset['RST Flag Cnt'] = df_dataset['RST Flag Cnt'].astype(int)
    df_dataset['PSH Flag Cnt'] = df_dataset['PSH Flag Cnt'].astype(int)
    df_dataset['ACK Flag Cnt'] = df_dataset['ACK Flag Cnt'].astype(int)
    df_dataset['URG Flag Cnt'] = df_dataset['URG Flag Cnt'].astype(int)
    df_dataset['CWE Flag Count'] = df_dataset['CWE Flag Count'].astype(int)
    df_dataset['ECE Flag Cnt'] = df_dataset['ECE Flag Cnt'].astype(int)
    df_dataset['Down/Up Ratio'] = df_dataset['Down/Up Ratio'].astype(int)
    df_dataset['Pkt Size Avg'] = df_dataset['Pkt Size Avg'].astype(float)
    df_dataset['Fwd Seg Size Avg'] = df_dataset['Fwd Seg Size Avg'].astype(float)
    df_dataset['Bwd Seg Size Avg'] = df_dataset['Bwd Seg Size Avg'].astype(float)
    df_dataset['Fwd Byts/b Avg'] = df_dataset['Fwd Byts/b Avg'].astype(int)
    df_dataset['Fwd Pkts/b Avg'] = df_dataset['Fwd Pkts/b Avg'].astype(int)
    df_dataset['Fwd Blk Rate Avg'] = df_dataset['Fwd Blk Rate Avg'].astype(int)
    df_dataset['Bwd Byts/b Avg'] = df_dataset['Bwd Byts/b Avg'].astype(int)
    df_dataset['Bwd Pkts/b Avg'] = df_dataset['Bwd Pkts/b Avg'].astype(int)
    df_dataset['Bwd Blk Rate Avg'] = df_dataset['Bwd Blk Rate Avg'].astype(int)
    df_dataset['Subflow Fwd Pkts'] = df_dataset['Subflow Fwd Pkts'].astype(int)
    df_dataset['Subflow Fwd Byts'] = df_dataset['Subflow Fwd Byts'].astype(int)
    df_dataset['Subflow Bwd Pkts'] = df_dataset['Subflow Bwd Pkts'].astype(int)
    df_dataset['Subflow Bwd Byts'] = df_dataset['Subflow Bwd Byts'].astype(int)
    df_dataset['Init Fwd Win Byts'] = df_dataset['Init Fwd Win Byts'].astype(int)
    df_dataset['Init Bwd Win Byts'] = df_dataset['Init Bwd Win Byts'].astype(int)
    df_dataset['Fwd Act Data Pkts'] = df_dataset['Fwd Act Data Pkts'].astype(int)
    df_dataset['Fwd Seg Size Min'] = df_dataset['Fwd Seg Size Min'].astype(int)
    df_dataset['Active Mean'] = df_dataset['Active Mean'].astype(float)
    df_dataset['Active Std'] = df_dataset['Active Std'].astype(float)
    df_dataset['Active Max'] = df_dataset['Active Max'].astype(int)
```

```
    df_dataset['Active Min'] = df_dataset['Active Min'].astype(int)
    df_dataset['Idle Mean'] = df_dataset['Idle Mean'].astype(float)
    df_dataset['Idle Std'] = df_dataset['Idle Std'].astype(float)
    df_dataset['Idle Max'] = df_dataset['Idle Max'].astype(int)
    df_dataset['Idle Min'] = df_dataset['Idle Min'].astype(int)

    return df_dataset
```

```
%%time
network_data_d3 = fixDataType(network_data_d3)
network_data_d8 = fixDataType(network_data_d8)
network_data_d9 = fixDataType(network_data_d9)
```

```
 CPU times: total: 24.1 s
 Wall time: 43.9 s
```

EDA

Data Properties

```
def dataProperties(df, day):
    print(day)
    df.shape
    print ('Number of rows (Samples): ' , df.shape[0])
    print ('Number of columns (Features): ' , df.shape[1])
    print(df['Label'].value_counts())
    print("\n\n\n")
```

```
%%time
dataProperties(network_data_d1, "Day 1")
dataProperties(network_data_d2, "Day 2")
dataProperties(network_data_d3, "Day 3")
dataProperties(network_data_d4, "Day 4")
dataProperties(network_data_d5, "Day 5")
dataProperties(network_data_d6, "Day 6")
dataProperties(network_data_d7, "Day 7")
dataProperties(network_data_d8, "Day 8")
dataProperties(network_data_d9, "Day 9")
dataProperties(network_data_d10, "Day 10")
```

```
Day 9
Number of rows (Samples):  331100
Number of columns (Features):  80
Benign           238037
Infilteration     93063
Name: Label, dtype: int64
```

```
Day 10
Number of rows (Samples):  1048575
Number of columns (Features):  80
Benign    762384
Bot       286191
Name: Label, dtype: int64
```

```
CPU times: total: 312 ms
Wall time: 515 ms
```

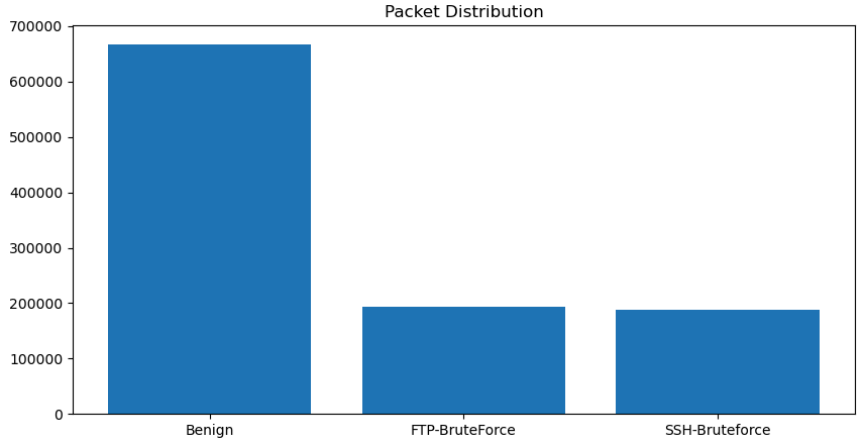
Data Visualizations

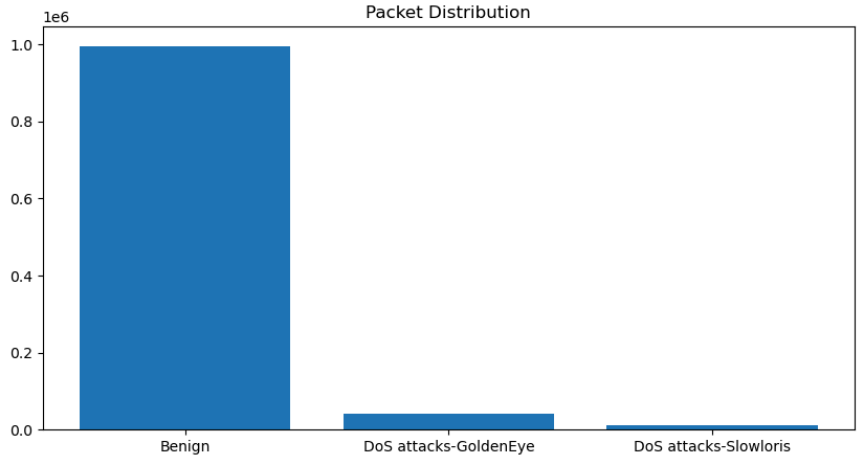
Bar Chart

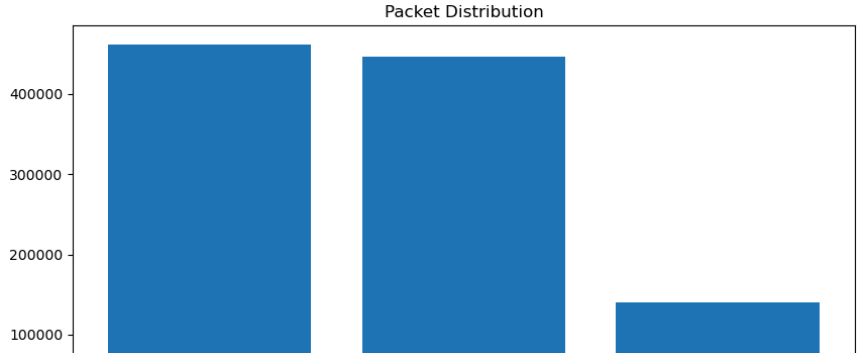
```
# %%time
def visualizeBar(df):
    # bar chart of packets label
    plt.figure(figsize=(10, 5))
    plt.title('Packet Distribution')
    # plt.bar(x=['Benign', 'FTP-BruteForce', 'SSH-Bruteforce'], height=network_data['Label'].value_counts(), color=['blue', 'magenta', 'cyan
    plt.bar(x=df['Label'].unique(), height=df['Label'].value_counts())
    p = plt.gcf()
```

```
%%time
# network_data = dropInfinateNull(network_data)
visualizeBar(network_data_d1)
visualizeBar(network_data_d2)
visualizeBar(network_data_d3)
visualizeBar(network_data_d4)
visualizeBar(network_data_d5)
visualizeBar(network_data_d6)
visualizeBar(network_data_d7)
visualizeBar(network_data_d8)
visualizeBar(network_data_d9)
visualizeBar(network_data_d10)
```

```
CPU times: total: 562 ms
Wall time: 1.26 s
```





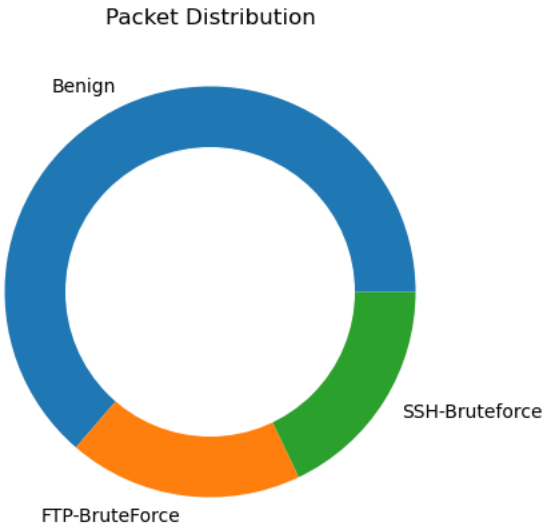


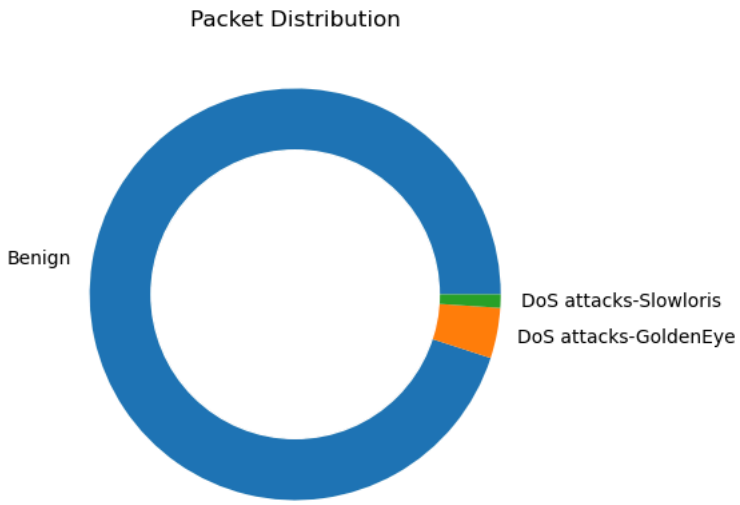
Pie Chart

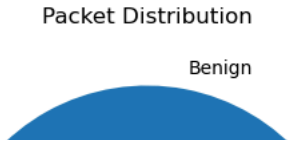
```
# %%time
def visualizePie(df):
    # pie chart of packets label
    plt.figure(figsize=(5, 5))
    circle = plt.Circle((0, 0), 0.7, color='white')
    plt.title('Packet Distribution')
    # plt.pie(network_data_d1['Label'].value_counts(), labels=['Benign', 'FTP-BruteForce', 'SSH-Bruteforce'], colors=['blue', 'magenta', 'cy
    plt.pie(df['Label'].value_counts(), labels=df['Label'].unique())
    p = plt.gcf()
    p.gca().add_artist(circle)


%%time
# network_data = dropInfinateNull(network_data)
visualizePie(network_data_d1)
visualizePie(network_data_d2)
visualizePie(network_data_d3)
visualizePie(network_data_d4)
visualizePie(network_data_d5)
visualizePie(network_data_d6)
visualizePie(network_data_d7)
visualizePie(network_data_d8)
visualizePie(network_data_d9)
visualizePie(network_data_d10)
```

```
CPU times: total: 812 ms
Wall time: 1.09 s
```







Data Preprocessing

Drop Infinate and Null

```
def dropInfinateNull(df):
    print (df.shape)

    # replace infinity value as null value
    df = df.replace(["Infinity", "infinity"], np.inf)
    df = df.replace([np.inf, -np.inf], np.nan)

    # drop all null values
    df.dropna(inplace=True)

    print (df.shape)

    return df
```

```
%%time
# network_data = dropInfinateNull(network_data)
network_data_d1 = dropInfinateNull(network_data_d1)
network_data_d2 = dropInfinateNull(network_data_d2)
network_data_d3 = dropInfinateNull(network_data_d3)
network_data_d4 = dropInfinateNull(network_data_d4)
network_data_d5 = dropInfinateNull(network_data_d5)
network_data_d6 = dropInfinateNull(network_data_d6)
network_data_d7 = dropInfinateNull(network_data_d7)
network_data_d8 = dropInfinateNull(network_data_d8)
network_data_d9 = dropInfinateNull(network_data_d9)
network_data_d10 = dropInfinateNull(network_data_d10)
```

```
(1048575, 80)
(1044751, 80)
(1048575, 80)
(1040548, 80)
(1048574, 80)
(1048574, 80)
(7948748, 80)
(7889295, 80)
(1048575, 80)
(1048575, 80)
(1048575, 80)
(1042965, 80)
(1048575, 80)
(1042867, 80)
(613071, 80)
(606902, 80)
(331100, 80)
(328181, 80)
(1048575, 80)
(1044525, 80)
CPU times: total: 22 s
Wall time: 1min 4s
```

Drop Unnecessary Column

```
def dropUnnecessaryColumn(df):
    df.drop(columns="Timestamp", inplace=True)
    print (df.shape)
    return df
```

```
%%time
# network_data = dropInfinateNull(network_data)
network_data_d1 = dropUnnecessaryColumn(network_data_d1)
network_data_d2 = dropUnnecessaryColumn(network_data_d2)
network_data_d3 = dropUnnecessaryColumn(network_data_d3)
network_data_d4 = dropUnnecessaryColumn(network_data_d4)
network_data_d5 = dropUnnecessaryColumn(network_data_d5)
network_data_d6 = dropUnnecessaryColumn(network_data_d6)
network_data_d7 = dropUnnecessaryColumn(network_data_d7)
network_data_d8 = dropUnnecessaryColumn(network_data_d8)
network_data_d9 = dropUnnecessaryColumn(network_data_d9)
network_data_d10 = dropUnnecessaryColumn(network_data_d10)
```

```
(1044751, 79)
(1040548, 79)
```

```
(1048574, 79)
(7889295, 79)
(1048575, 79)
(1042965, 79)
(1042867, 79)
(606902, 79)
(328181, 79)
(1044525, 79)
CPU times: total: 3.75 s
Wall time: 16.8 s
```

Transform Target Label into Binary Class

```
%%time
def transformTargetLabelToBinary(df):
    # encode the target feature
    df['Label'] = df['Label'].apply(lambda x: "Benign" if x == 'Benign' else "Malicious")
    print(df['Label'].unique())
    print(df['Label'].value_counts())
    return df
```

```
CPU times: total: 0 ns
Wall time: 0 ns
```

```
%%time
# network_data = dropInfinateNull(network_data)
network_data_d1 = transformTargetLabelToBinary(network_data_d1)
network_data_d2 = transformTargetLabelToBinary(network_data_d2)
network_data_d3 = transformTargetLabelToBinary(network_data_d3)
network_data_d4 = transformTargetLabelToBinary(network_data_d4)
network_data_d5 = transformTargetLabelToBinary(network_data_d5)
network_data_d6 = transformTargetLabelToBinary(network_data_d6)
network_data_d7 = transformTargetLabelToBinary(network_data_d7)
network_data_d8 = transformTargetLabelToBinary(network_data_d8)
network_data_d9 = transformTargetLabelToBinary(network_data_d9)
network_data_d10 = transformTargetLabelToBinary(network_data_d10)
```

```
['Benign' 'Malicious']
Benign       663808
Malicious    380943
Name: Label, dtype: int64
['Benign' 'Malicious']
Benign       988050
Malicious     52498
Name: Label, dtype: int64
['Benign' 'Malicious']
Malicious    601802
Benign       446772
Name: Label, dtype: int64
['Benign' 'Malicious']
Benign       7313104
Malicious     576191
Name: Label, dtype: int64
['Benign' 'Malicious']
Malicious    687742
Benign       360833
Name: Label, dtype: int64
['Benign' 'Malicious']
Benign       1042603
Malicious        362
Name: Label, dtype: int64
['Benign' 'Malicious']
Benign       1042301
Malicious        566
Name: Label, dtype: int64
['Benign' 'Malicious']
Benign       538666
Malicious     68236
Name: Label, dtype: int64
['Benign' 'Malicious']
Benign       235778
Malicious     92403
Name: Label, dtype: int64
['Benign' 'Malicious']
Benign       758334
Malicious    286191
Name: Label, dtype: int64
CPU times: total: 1.31 s
Wall time: 2.51 s
```

Data Balancing

```
%%time
def balanceData(df):
    # split data into features and target
    X=df.drop(["Label"], axis=1)
    y=df["Label"]

    # applying oversampling
    rus = RandomUnderSampler()
    X_balanced, y_balanced = rus.fit_resample(X, y) # insted of X, y use the direct syntex

    df = pd.concat([X_balanced, y_balanced], axis=1)
    del X, y, X_balanced, y_balanced
    print (df.shape)
    print(df['Label'].value_counts())

    return df

     CPU times: total: 0 ns
     Wall time: 0 ns


%%time
# network_data = dropInfinateNull(network_data)
network_data_d1 = balanceData(network_data_d1)
network_data_d2 = balanceData(network_data_d2)
network_data_d3 = balanceData(network_data_d3)
network_data_d4 = balanceData(network_data_d4)
network_data_d5 = balanceData(network_data_d5)
network_data_d6 = balanceData(network_data_d6)
network_data_d7 = balanceData(network_data_d7)
network_data_d8 = balanceData(network_data_d8)
network_data_d9 = balanceData(network_data_d9)
network_data_d10 = balanceData(network_data_d10)

     (761886, 79)
     Benign       380943
     Malicious    380943
     Name: Label, dtype: int64
     (104996, 79)
     Benign       52498
     Malicious    52498
     Name: Label, dtype: int64
     (893544, 79)
     Benign       446772
     Malicious    446772
     Name: Label, dtype: int64
     (1152382, 79)
     Benign       576191
     Malicious    576191
     Name: Label, dtype: int64
     (721666, 79)
     Benign       360833
     Malicious    360833
     Name: Label, dtype: int64
     (724, 79)
     Benign       362
     Malicious    362
     Name: Label, dtype: int64
     (1132, 79)
     Benign       566
     Malicious    566
     Name: Label, dtype: int64
     (136472, 79)
     Benign       68236
     Malicious    68236
     Name: Label, dtype: int64
     (184806, 79)
     Benign       92403
     Malicious    92403
     Name: Label, dtype: int64
     (572382, 79)
     Benign       286191
     Malicious    286191
     Name: Label, dtype: int64
     CPU times: total: 32.5 s
     Wall time: 1min 13s
```

Data Concatenation

```
%%time
network_data = pd.concat([network_data_d1, network_data_d2], axis=0)
network_data.reset_index(drop=True, inplace=True)
del network_data_d1, network_data_d2

network_data = pd.concat([network_data, network_data_d3], axis=0)
network_data.reset_index(drop=True, inplace=True)
del network_data_d3

network_data = pd.concat([network_data, network_data_d4], axis=0)
network_data.reset_index(drop=True, inplace=True)
del network_data_d4

network_data = pd.concat([network_data, network_data_d5], axis=0)
network_data.reset_index(drop=True, inplace=True)
del network_data_d5

network_data = pd.concat([network_data, network_data_d6], axis=0)
network_data.reset_index(drop=True, inplace=True)
del network_data_d6

network_data = pd.concat([network_data, network_data_d7], axis=0)
network_data.reset_index(drop=True, inplace=True)
del network_data_d7

network_data = pd.concat([network_data, network_data_d8], axis=0)
network_data.reset_index(drop=True, inplace=True)
del network_data_d8

network_data = pd.concat([network_data, network_data_d9], axis=0)
network_data.reset_index(drop=True, inplace=True)
del network_data_d9

network_data = pd.concat([network_data, network_data_d10], axis=0)
network_data.reset_index(drop=True, inplace=True)
del network_data_d10
```

```
CPU times: total: 2.98 s
Wall time: 5.32 s
```

```
network_data['Label'].value_counts()
```

```
Benign       2264995
Malicious    2264995
Name: Label, dtype: int64
```

Temporarily Hold Data

This helps to code while manupulating the data. If something unexpected operation happend on data, strat a run from this cell is enough. Don't have to run full notebook.

```
%%time
# while working comment one line and uncomment another as needed
# temp_network_data = network_data
# network_data = temp_network_data
```

```
CPU times: total: 0 ns
Wall time: 0 ns
```

Feature Selection

Drop Constant Column

```
%%time
# drop the constant columns (which varience is 0)
variances = network_data.var(numeric_only=True)
constant_columns = variances[variances == 0].index
network_data = network_data.drop(constant_columns, axis=1)

print(constant_columns)
print (network_data.shape)
```

```
Index(['Bwd PSH Flags', 'Bwd URG Flags', 'Fwd Byts/b Avg', 'Fwd Pkts/b Avg',
       'Fwd Blk Rate Avg', 'Bwd Byts/b Avg', 'Bwd Pkts/b Avg',
       'Bwd Blk Rate Avg'],
      dtype='object')
(4529990, 71)
CPU times: total: 734 ms
Wall time: 1.63 s
```

Check and Drop Duplicate Column

```
%%time
duplicates = set()
for i in range(0, len(network_data.columns)):
    col1 = network_data.columns[i]
    for j in range(i+1, len(network_data.columns)):
        col2 = network_data.columns[j]
        if(network_data[col1].equals(network_data[col2])):
            duplicates.add(col2)

print (duplicates)
network_data.drop(duplicates, axis=1, inplace=True)
print (network_data.shape)
```

```
{'Subflow Fwd Pkts', 'CWE Flag Count', 'Subflow Bwd Pkts', 'SYN Flag Cnt'}
(4529990, 67)
CPU times: total: 6.5 s
Wall time: 12.4 s
```

Encode Target Label

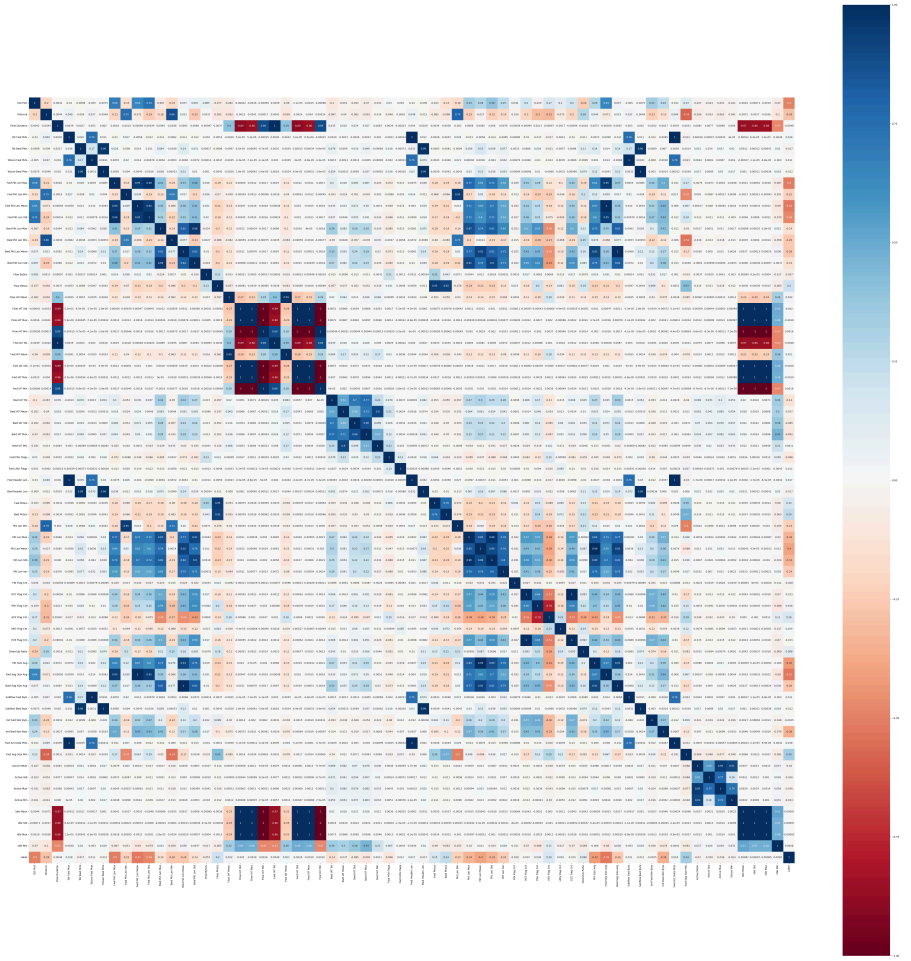
```
%%time
# encode the target feature
network_data['Label'] = network_data['Label'].apply(lambda x: 0 if x == 'Benign' else 1)
print(network_data['Label'].unique())
```

```
[0 1]
CPU times: total: 484 ms
Wall time: 1 s
```

Drop Column Based on Correlations

Correlations of Data

```
%%time
# pearson correlation heatmap
plt.figure(figsize=(70, 70))
corr = network_data.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap='RdBu', vmin=-1, vmax=1, square=True) # annot=True
plt.show()
```



```
CPU times: total: 21.3 s
Wall time: 38.7 s
```

Drop Columns

```
%%time
correlated_col = set()
is_correlated = [True] * len(corr.columns)
threshold = 0.90
for i in range (len(corr.columns)):
    if(is_correlated[i]):
        for j in range(i):
            if (corr.iloc[i, j] >= threshold) and (is_correlated[j]):
                colname = corr.columns[j]
                is_correlated[j]=False
                correlated_col.add(colname)

print(correlated_col)
print(len(correlated_col))
```

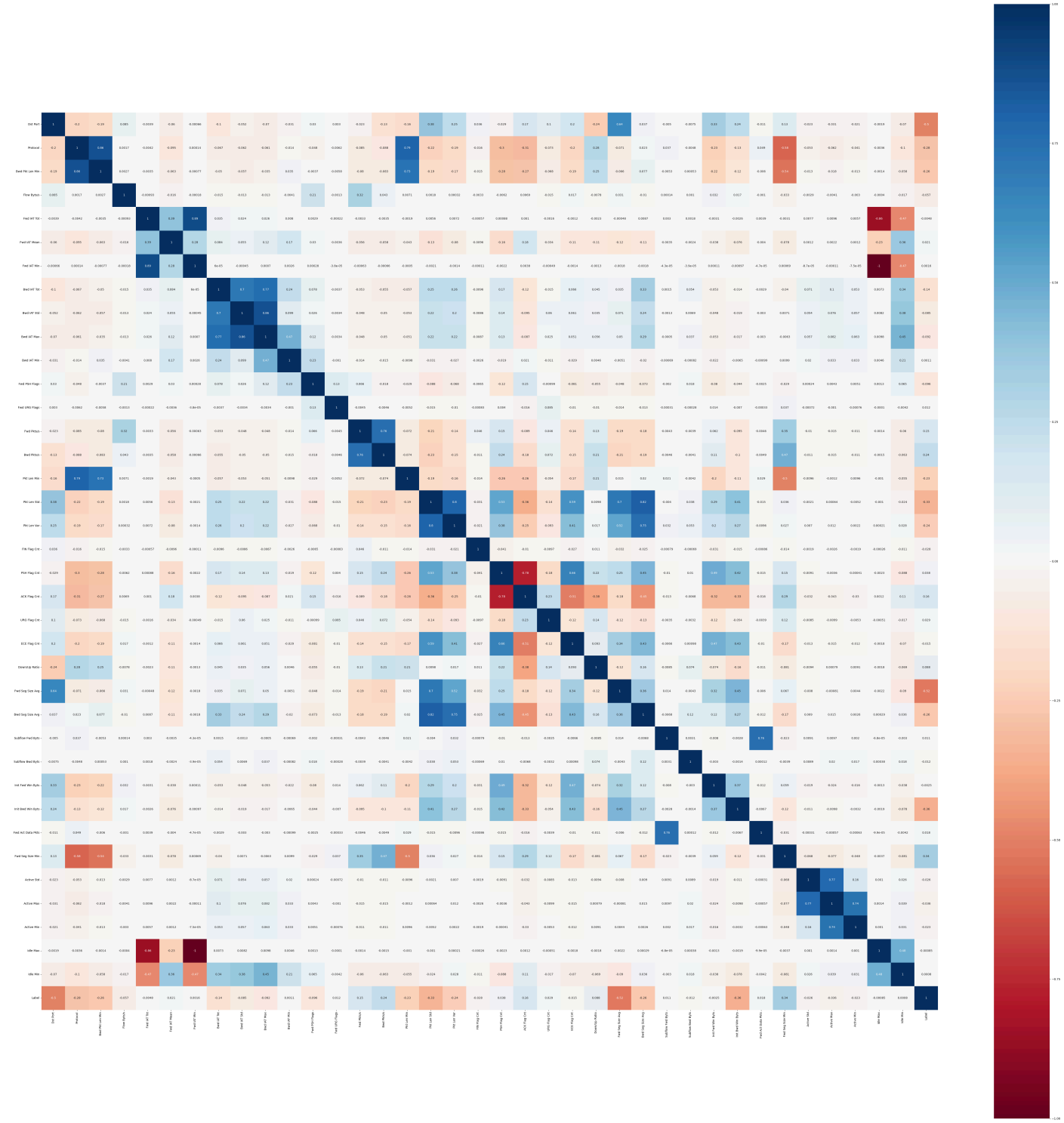
```
{'Flow IAT Mean', 'Fwd Header Len', 'TotLen Fwd Pkts', 'Bwd Pkt Len Mean', 'Flow IAT Max', 'TotLen Bwd Pkts', 'Bwd Pkt Len Max', 'Flow D
29
CPU times: total: 15.6 ms
Wall time: 38.1 ms
```

```
%%time
network_data.drop(correlated_col, axis=1, inplace=True)
print (network_data.shape)
```

```
(4529990, 38)
CPU times: total: 266 ms
Wall time: 778 ms
```

Correlations of Data

```
%%time
# pearson correlation heatmap
plt.figure(figsize=(70, 70))
corr = network_data.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap='RdBu', vmin=-1, vmax=1, square=True) # annot=True
plt.show()
```



```
CPU times: total: 6.95 s
Wall time: 13.2 s
```

Save Data

```
network_data.to_csv('./kaggle/working/cic-ids-2.csv', index=False)
```

Temporarily Hold Data

This helps to code while manupulating the data. If something unexpected operation happend on data, strat a run from this cell is enough. Don't have to run full notebook.

```
%%time
# while working comment one line and uncomment another as needed
# temp_network_data = network_data
# network_data = temp_network_data
```

```
CPU times: total: 0 ns
Wall time: 0 ns
```

Sequential Feature Selection

```
%%time
# split data into features and target
X=network_data.drop(["Label"], axis=1)
y=network_data["Label"]
```

```
CPU times: total: 312 ms
Wall time: 483 ms
```

Backward Feature Selection

```
%%time
# applying backward feature selection
dt = DecisionTreeClassifier()
sfs_backward = SequentialFeatureSelector(
    dt, n_features_to_select=30, direction="backward"
).fit(X, y)
```

```
CPU times: total: 16h 53min 18s
Wall time: 19h 15min 19s
```

```
selected_features = X.columns[sfs_backward.get_support()]
print(selected_features)
```

```
Index(['Protocol', 'Bwd Pkt Len Min', 'Flow Byts/s', 'Fwd IAT Tot',
       'Fwd IAT Mean', 'Bwd IAT Std', 'Bwd IAT Min', 'Fwd PSH Flags',
       'Fwd URG Flags', 'Fwd Pkts/s', 'Pkt Len Min', 'Pkt Len Std',
       'Pkt Len Var', 'FIN Flag Cnt', 'PSH Flag Cnt', 'ACK Flag Cnt',
       'ECE Flag Cnt', 'Down/Up Ratio', 'Fwd Seg Size Avg', 'Bwd Seg Size Avg',
       'Subflow Fwd Byts', 'Subflow Bwd Byts', 'Init Fwd Win Byts',
       'Fwd Act Data Pkts', 'Fwd Seg Size Min', 'Active Std', 'Active Max',
       'Active Min', 'Idle Max', 'Idle Min'],
      dtype='object')
```

```
# add target label
selected_features = selected_features.tolist()
selected_features.append('Label')
selected_features = pd.Index(selected_features)
print(selected_features)
```

```
Index(['Protocol', 'Bwd Pkt Len Min', 'Flow Byts/s', 'Fwd IAT Tot',
       'Fwd IAT Mean', 'Bwd IAT Std', 'Bwd IAT Min', 'Fwd PSH Flags',
       'Fwd URG Flags', 'Fwd Pkts/s', 'Pkt Len Min', 'Pkt Len Std',
       'Pkt Len Var', 'FIN Flag Cnt', 'PSH Flag Cnt', 'ACK Flag Cnt',
       'ECE Flag Cnt', 'Down/Up Ratio', 'Fwd Seg Size Avg', 'Bwd Seg Size Avg',
       'Subflow Fwd Byts', 'Subflow Bwd Byts', 'Init Fwd Win Byts',
       'Fwd Act Data Pkts', 'Fwd Seg Size Min', 'Active Std', 'Active Max',
       'Active Min', 'Idle Max', 'Idle Min', 'Label'],
      dtype='object')
```

```
%%time
network_data = network_data.loc[:, selected_features]
del X, y
network_data
```

```
CPU times: total: 297 ms
Wall time: 1.98 s
```

	Protocol	Bwd Pkt Len Min	Flow Byts/s	Fwd IAT Tot	Fwd IAT Mean	Bwd IAT Std	Bwd IAT Mir
0	6	0.0	1959.508300	1387593.0	1.982276e+05	3.591041e+05	61461.0
1	6	0.0	1081.272711	733315.0	1.466630e+05	1.781080e+04	162246.0
2	17	67.0	289926.289926	0.0	0.000000e+00	0.000000e+00	0.0
3	6	0.0	39.829581	118680636.0	7.912042e+06	2.069812e+07	1.0
4	17	139.0	12554.355450	0.0	0.000000e+00	0.000000e+00	0.0
...	...	...	...	...	...	...	..
4529985	0	0.0	0.000000	9487.0	1.054111e+03	0.000000e+00	0.0
4529986	6	0.0	0.000000	1030059.0	3.433530e+05	0.000000e+00	515615.0
4529987	6	0.0	0.000000	1029998.0	3.433327e+05	0.000000e+00	515621.0
4529988	6	0.0	0.000000	1030017.0	3.433390e+05	0.000000e+00	515643.0
4529989	6	0.0	0.000000	1030836.0	3.436120e+05	0.000000e+00	515608.0

4529990 rows × 31 columns

Temporarily Hold Data

This helps to code while manupulating the data. If something unexpected operation happend on data, strat a run from this cell is enough. Don't have to run full notebook.

```
%%time
# while working comment one line and uncomment another as needed
# temp_network_data = network_data
# network_data = temp_network_data
```

```
CPU times: total: 0 ns
Wall time: 0 ns
```

Data Normalization

```
%%time
# list numeric columns

numeric_cols = ['Flow Duration', 'Tot Fwd Pkts',
       'Tot Bwd Pkts', 'TotLen Fwd Pkts', 'TotLen Bwd Pkts', 'Fwd Pkt Len Max',
       'Fwd Pkt Len Min', 'Fwd Pkt Len Mean', 'Fwd Pkt Len Std',
       'Bwd Pkt Len Max', 'Bwd Pkt Len Min', 'Bwd Pkt Len Mean',
       'Bwd Pkt Len Std', 'Flow Byts/s', 'Flow Pkts/s', 'Flow IAT Mean',
       'Flow IAT Std', 'Flow IAT Max', 'Flow IAT Min', 'Fwd IAT Tot',
       'Fwd IAT Mean', 'Fwd IAT Std', 'Fwd IAT Max', 'Fwd IAT Min',
       'Bwd IAT Tot', '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 Len', 'Bwd Header Len', 'Fwd Pkts/s',
       'Bwd Pkts/s', 'Pkt Len Min', 'Pkt Len Max', 'Pkt Len Mean',
       'Pkt Len Std', 'Pkt Len Var', 'FIN Flag Cnt', 'SYN Flag Cnt',
       'RST Flag Cnt', 'PSH Flag Cnt', 'ACK Flag Cnt', 'URG Flag Cnt',
       'CWE Flag Count', 'ECE Flag Cnt', 'Down/Up Ratio', 'Pkt Size Avg',
       'Fwd Seg Size Avg', 'Bwd Seg Size Avg', 'Fwd Byts/b Avg',
       'Fwd Pkts/b Avg', 'Fwd Blk Rate Avg', 'Bwd Byts/b Avg',
       'Bwd Pkts/b Avg', 'Bwd Blk Rate Avg', 'Subflow Fwd Pkts',
       'Subflow Fwd Byts', 'Subflow Bwd Pkts', 'Subflow Bwd Byts',
       'Init Fwd Win Byts', 'Init Bwd Win Byts', 'Fwd Act Data Pkts',
       'Fwd Seg Size Min', 'Active Mean', 'Active Std', 'Active Max',
       'Active Min', 'Idle Mean', 'Idle Std', 'Idle Max', 'Idle Min']

print(numeric_cols)
```

```
['Flow Duration', 'Tot Fwd Pkts', 'Tot Bwd Pkts', 'TotLen Fwd Pkts', 'TotLen Bwd Pkts', 'Fwd Pkt Len Max', 'Fwd Pkt Len Min', 'Fwd Pkt L
CPU times: total: 0 ns
```

```
Wall time: 0 ns
```

```
network_data
```

	Protocol	Bwd Pkt Len Min	Flow Byts/s	Fwd IAT Tot	Fwd IAT Mean	Bwd IAT Std	Bwd IAT Mir
0	6	0.0	1959.508300	1387593.0	1.982276e+05	3.591041e+05	61461.0
1	6	0.0	1081.272711	733315.0	1.466630e+05	1.781080e+04	162246.0
2	17	67.0	289926.289926	0.0	0.000000e+00	0.000000e+00	0.0
3	6	0.0	39.829581	118680636.0	7.912042e+06	2.069812e+07	1.0
4	17	139.0	12554.355450	0.0	0.000000e+00	0.000000e+00	0.0
...	...	...	...	...	...	...	..
4529985	0	0.0	0.000000	9487.0	1.054111e+03	0.000000e+00	0.0
4529986	6	0.0	0.000000	1030059.0	3.433530e+05	0.000000e+00	515615.0
4529987	6	0.0	0.000000	1029998.0	3.433327e+05	0.000000e+00	515621.0
4529988	6	0.0	0.000000	1030017.0	3.433390e+05	0.000000e+00	515643.0
4529989	6	0.0	0.000000	1030836.0	3.436120e+05	0.000000e+00	515608.0

4529990 rows × 31 columns

Classification

Here many algoritms will be used to classify the data.

Split Data

```
%%time
X=network_data.drop(["Label"], axis=1)
y=network_data["Label"]
# K-fold
kf = KFold(n_splits=10, shuffle=True, random_state=42)
```

```
CPU times: total: 141 ms
Wall time: 371 ms
```

```
print(X)
```

```
          Protocol  Bwd Pkt Len Min    Flow Byts/s  Fwd IAT Tot  Fwd IAT Mean  \
0                6              0.0    1959.508300    1387593.0  1.982276e+05
1                6              0.0    1081.272711     733315.0  1.466630e+05
2               17             67.0  289926.289926          0.0  0.000000e+00
3                6              0.0      39.829581  118680636.0  7.912042e+06
4               17            139.0   12554.355450          0.0  0.000000e+00
...            ...              ...            ...          ...           ...
4529985          0              0.0       0.000000       9487.0  1.054111e+03
4529986          6              0.0       0.000000    1030059.0  3.433530e+05
4529987          6              0.0       0.000000    1029998.0  3.433327e+05
4529988          6              0.0       0.000000    1030017.0  3.433390e+05
4529989          6              0.0       0.000000    1030836.0  3.436120e+05

           Bwd IAT Std  Bwd IAT Min  Fwd PSH Flags  Fwd URG Flags   Fwd Pkts/s  \
0        3.591041e+05      61461.0              0              0     5.765379
1        1.781080e+04     162246.0              0              0     8.181130
2        0.000000e+00          0.0              0              0  2457.002457
3        2.069812e+07          1.0              0              0     0.134816
4        0.000000e+00          0.0              0              0    70.136064
...               ...          ...            ...            ...          ...
4529985  0.000000e+00          0.0              0              0  1054.073996
4529986  0.000000e+00     515615.0              0              0     3.883273
4529987  0.000000e+00     515621.0              0              0     3.883503
4529988  0.000000e+00     515643.0              0              0     3.883431
4529989  0.000000e+00     515608.0              0              0     3.880346
```