1. Loading and Reading the data

1.1 loading the obtained DDOS attack datset from Kaggles # each csv hold the diffrent type of DDOS attacks collected on diffrent dates

In [ ]:
```
import pandas as pd

# df1=pd.read_csv('./02-14-2018.csv')
# df2=pd.read_csv('./02-15-2018.csv')
# df3=pd.read_csv('./02-16-2018.csv')
# df4=pd.read_csv('./02-20-2018.csv')
# df5=pd.read_csv('./02-21-2018.csv')
df6=pd.read_csv('./02-22-2018.csv')
df7=pd.read_csv('./02-23-2018.csv')
# df8=pd.read_csv('./02-28-2018.csv')
# df9=pd.read_csv('./03-01-2018.csv')
# df10=pd.read_csv('./03-02-2018.csv')
```

In [ ]:
```
# print('the shape of dataframe1',df1.shape) #reading the file shape for each data f
# print('the shape of dataframe2',df2.shape)
# print('the shape of dataframe3',df3.shape)
# print('the shape of dataframe4',df4.shape)
# print('the shape of dataframe5',df5.shape)
print('the shape of dataframe6',df6.shape)
print('the shape of dataframe7',df7.shape)
# print('the shape of dataframe8',df8.shape)
# print('the shape of dataframe9',df9.shape)
# print('the shape of dataframe10',df10.shape)
```

```
the shape of dataframe6 (1048575, 80)
the shape of dataframe7 (1048575, 80)
```

1.2 Reading the dataset and removing the outliers

In [ ]:
```
# print('1.The Type and quantity of attach present in df1 \n',df1['Label'].value_cou
# print('\n 2.The Type and quantity of attach present in df2 \n ',df2['Label'].value_
# print('\n 3.The Type and quantity of attach present in df3 \n',df3['Label'].value_
# print('\n 4. The Type and quantity of attach present in df4 \n',df4['Label'].value_
# print('\n 5. The Type and quantity of attach present in df5 \n',df5['Label'].value_
print('\n 6. The Type and quantity of attach present in df6 \n',df6['Label'].value_c
print('\n 7. The Type and quantity of attach present in df7 \n',df7['Label'].value_c
# print('\n 8. The Type and quantity of attach present in df8 \n',df8['Label'].value_
# print('\n 9. The Type and quantity of attach present in df9 \n',df9['Label'].value_
# print('\n 10. The Type and quantity of attach present in df10 \n',df10['Label'].va
```

```
 6. The Type and quantity of attach present in df6
 Benign              1048213
Brute Force -Web        249
Brute Force -XSS         79
SQL Injection            34
Name: Label, dtype: int64

 7. The Type and quantity of attach present in df7
 Benign              1048009
Brute Force -Web        362
Brute Force -XSS        151
SQL Injection            53
Name: Label, dtype: int64
```

next codebox we will use df.drop(df.loc[df['Label']=='Label'].index,inplace=True) # the column value = 'Label' does not account for any type of DDOS attack, hence removing this

In [ ]:

```
# df3.drop(df3.loc[df3['Label']=='Label'].index,inplace=True)
# df8.drop(df8.loc[df8['Label']=='Label'].index,inplace=True)
# df9.drop(df9.loc[df9['Label']=='Label'].index,inplace=True)
```

1.3 Using Stratified Sampling to sample data from the population

In [ ]:

```
# Strat_df1=df1.groupby('Label', group_keys=False).apply(lambda x: x.sample(10000))
# Strat_df2=df2.groupby('Label', group_keys=False).apply(lambda x: x.sample(10000))
# Strat_df3=df3.groupby('Label', group_keys=False).apply(lambda x: x.sample(10000))
# Strat_df4=df4.groupby('Label', group_keys=False).apply(lambda x: x.sample(10000))
# del df1,df2,df3,df4

# print('The Stratified Sample of Dataset1 \n ', Strat_df1['Label'].value_counts())
# print('='*100)
# print('The Stratified Sample of Dataset2 \n ', Strat_df2['Label'].value_counts())
# print('='*100)
# print('The Stratified Sample of Dataset3 \n ', Strat_df3['Label'].value_counts())
# print('='*100)
# print('The Stratified Sample of Dataset4 \n ', Strat_df4['Label'].value_counts())

# print('*'*100)
# print('*'*100)

# #web attcks in these dataframes are less than 10000 hence we are randomly sampling
N=10000
# Strat_df5=df5.head(N)
Strat_df6=df6.head(N)
Strat_df7=df7.head(N)
# del df5,df6,df7


# print('The Stratified Sample of Dataset5 \n ', Strat_df5['Label'].value_counts())
# print('='*100)
print('The Stratified Sample of Dataset6 \n ', Strat_df6['Label'].value_counts())
print('='*100)
print('The Stratified Sample of Dataset7 \n ', Strat_df7['Label'].value_counts())

print('*'*100)
print('*'*100)
# Applying Stratified Sampling on the datasets
# Strat_df8=df8.groupby('Label', group_keys=False).apply(lambda x: x.sample(10000))
# Strat_df9=df9.groupby('Label', group_keys=False).apply(lambda x: x.sample(10000))
# Strat_df10=df10.groupby('Label', group_keys=False).apply(lambda x: x.sample(10000)
# del df8,df9,df10
# #obtaning equal no of class sample to process EDA
# print('The Stratified Sample of Dataset8 \n ', Strat_df8['Label'].value_counts())
# print('='*100)
# print('The Stratified Sample of Dataset9 \n ', Strat_df9['Label'].value_counts())
# print('='*100)
# print('The Stratified Sample of Dataset10 \n ', Strat_df10['Label'].value_counts()
# print('='*100)
```

```
The Stratified Sample of Dataset6 
  Benign              9638
Brute Force -Web     249
Brute Force -XSS      79
SQL Injection         34
Name: Label, dtype: int64
====================================================================================================
```

```
===============
The Stratified Sample of Dataset7
  Benign              9434
Brute Force -Web      362
Brute Force -XSS      151
SQL Injection          53
Name: Label, dtype: int64
*********************************************************************************************
*********************************************************************************************
```

1.4 Concatenating all the sampled to a single dataset

In [ ]:

```
final_dataset=pd.concat([Strat_df6,Strat_df7])
del Strat_df6,Strat_df7
# final_dataset=pd.concat([final_dataset,Strat_df3])
# del Strat_df3
# final_dataset=pd.concat([final_dataset,Strat_df4])
# del Strat_df4
# final_dataset=pd.concat([final_dataset,Strat_df5])
# del Strat_df5
# final_dataset=pd.concat([final_dataset,Strat_df6])
# del Strat_df6
# final_dataset=pd.concat([final_dataset,Strat_df7])
# del Strat_df7
# final_dataset=pd.concat([final_dataset,Strat_df8])
# del Strat_df8
# final_dataset=pd.concat([final_dataset,Strat_df9])
# del Strat_df9
# final_dataset=pd.concat([final_dataset,Strat_df10])
# del Strat_df10
```

In [ ]:

```
final_dataset.shape
```

Out[ ]: (20000, 80)

In [ ]:

```
final_dataset.sample(20)
```

Out[ ]:

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min	Act
1871	500	17	23/02/2018 04:58:14	89479780	6	0	3000	0	500	500	...	8	4.00
1747	500	17	22/02/2018 04:51:48	89479474	6	0	3000	0	500	500	...	8	4.00
4767	443	6	22/02/2018 10:10:39	60140916	13	12	1052	1682	394	0	...	20	2.93
867	500	17	22/02/2018 12:11:12	89479575	6	0	3000	0	500	500	...	8	4.00
4990	3389	6	23/02/2018 10:10:45	4057303	13	8	1440	1731	725	0	...	20	0.00
6295	443	6	23/02/2018 10:44:13	180	2	0	0	0	0	0	...	20	0.00

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min	Act
7575	443	6	23/02/2018 11:15:49	172	2	0	0	0	0	0	...	20	0.00
9913	80	6	23/02/2018 01:05:54	118065210	20	16	1307	2118	432	0	...	20	2.81
1114	80	6	23/02/2018 01:19:11	56797401	203	104	56330	189999	680	0	...	20	0.00
451	80	6	22/02/2018 10:38:39	5001360	5	3	646	364	646	0	...	20	0.00
858	500	17	23/02/2018 11:58:59	89479630	6	0	3000	0	500	500	...	8	4.00
1645	500	17	23/02/2018 03:35:42	89479759	6	0	3000	0	500	500	...	8	4.00
5556	445	6	22/02/2018 10:37:50	263733	3	1	0	0	0	0	...	20	0.00
8556	80	6	23/02/2018 11:45:39	79	2	0	0	0	0	0	...	20	0.00
9326	443	6	22/02/2018 01:39:56	115844786	19	16	1823	425	1213	0	...	20	4.68
6306	80	6	23/02/2018 10:44:15	139	2	0	0	0	0	0	...	20	0.00
7696	443	6	23/02/2018 11:18:30	1730	3	0	77	0	46	0	...	20	0.00
9100	445	6	22/02/2018 01:29:59	829508	7	5	364	582	103	0	...	20	0.00
3656	443	6	22/02/2018 09:07:41	118168610	16	14	1007	3521	361	0	...	20	2.69
1104	500	17	23/02/2018 01:15:24	89479730	6	0	3000	0	500	500	...	8	4.00

20 rows × 80 columns

In [ ]:

```
final_dataset.sample(10)
```

Out[ ]:

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min	Ac
2946	80	6	23/02/2018 08:15:40	116058683	16	14	459	913	448	0	...	20	2.7
1589	49238	6	23/02/2018 03:17:19	5012529	2	2	0	0	0	0	...	32	0.0

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min	A
1179	80	6	23/02/2018 01:32:38	56654964	203	104	56330	190002	680	0	...	20	0.0
9799	80	6	22/02/2018 01:46:36	115515545	16	14	445	706	434	0	...	20	1.3
3918	80	6	22/02/2018 09:17:03	115939289	16	14	447	788	436	0	...	20	1.6
5075	80	6	22/02/2018 10:24:21	852	3	4	148	243	148	0	...	20	0.0
577	80	6	23/02/2018 10:42:42	56723607	153	104	54995	72494	646	0	...	20	0.0
6536	80	6	22/02/2018 11:08:30	5917468	4	4	97	231	97	0	...	20	0.0
8896	3389	6	23/02/2018 11:57:50	4126955	14	8	1441	1731	725	0	...	20	0.0
1852	500	17	23/02/2018 04:50:36	89479623	6	0	3000	0	500	500	...	8	4.0

10 rows × 80 columns

2.Data PreProcessing

In [ ]:

```
final_dataset.columns
```

Out[ ]:

```
Index(['Dst Port', 'Protocol', 'Timestamp', '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', 'Label'],
      dtype='object')
```

In [ ]:

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

Out[ ]:
```
Benign              19072
Brute Force -Web      611
Brute Force -XSS      230
SQL Injection          87
Name: Label, dtype: int64
```

In [ ]:
```
final_dataset.dtypes
```

Out[ ]:
```
Dst Port            int64
Protocol            int64
Timestamp          object
Flow Duration       int64
Tot Fwd Pkts        int64
                    ...
Idle Mean         float64
Idle Std          float64
Idle Max            int64
Idle Min            int64
Label              object
Length: 80, dtype: object
```

Clearly all the numerical features are listing as object here, hence we should convert them into numerical category

In [ ]:
```
import numpy as np #converting the timestamp is int

final_dataset['Timestamp'] = pd.to_datetime(final_dataset['Timestamp']).astype(np.in
```

In [ ]:
```
#converting all the numerical features datatype from object to float
final_dataset = final_dataset.astype({'Dst Port': 'float', 'Protocol': 'float','Dst
'Tot Bwd Pkts':'float', 'TotLen Fwd Pkts':'float', 'TotLen Bwd Pkts':'float', 'Fwd P

final_dataset = final_dataset.astype({'Fwd Pkt Len Min':'float', 'Fwd Pkt Len Mean':
'Bwd Pkt Len Std':'float', 'Flow Byts/s':'float', 'Flow Pkts/s':'float', 'Flow IAT M

final_dataset = final_dataset.astype({'Flow IAT Std':'float', 'Flow IAT Max':'float'
'Fwd IAT Min':'float','Bwd IAT Tot':'float', 'Bwd IAT Mean':'float', 'Bwd IAT Std':'
```

In [ ]:
```
final_dataset = final_dataset.astype({'Bwd IAT Min':'float', 'Fwd PSH Flags':'float'
'Fwd Pkts/s':'float','Bwd Pkts/s':'float', 'Pkt Len Min':'float', 'Pkt Len Max':'flo

final_dataset = final_dataset.astype({'Pkt Len Std':'float', 'Pkt Len Var':'float',
'URG Flag Cnt':'float','CWE Flag Count':'float', 'ECE Flag Cnt':'float', 'Down/Up Ra

final_dataset = final_dataset.astype({'Fwd Seg Size Avg':'float', 'Bwd Seg Size Avg'
'Bwd Pkts/b Avg':'float', 'Bwd Blk Rate Avg':'float', 'Subflow Fwd Pkts':'float','Su


final_dataset = final_dataset.astype({'Init Fwd Win Byts':'float', 'Init Bwd Win Byt
'Active Min':'float', 'Idle Mean':'float', 'Idle Std':'float', 'Idle Max':'float', '
```

In [ ]:
```
pd.set_option('display.max_rows',None) #listing all the rows
final_dataset.dtypes
```

Out[ ]:
```
Dst Port             float64
Protocol             float64
Timestamp            float64
Flow Duration        float64
Tot Fwd Pkts         float64
```

```
Tot Bwd Pkts         float64
TotLen Fwd Pkts      float64
TotLen Bwd Pkts      float64
Fwd Pkt Len Max      float64
Fwd Pkt Len Min      float64
Fwd Pkt Len Mean     float64
Fwd Pkt Len Std      float64
Bwd Pkt Len Max      float64
Bwd Pkt Len Min      float64
Bwd Pkt Len Mean     float64
Bwd Pkt Len Std      float64
Flow Byts/s          float64
Flow Pkts/s          float64
Flow IAT Mean        float64
Flow IAT Std         float64
Flow IAT Max         float64
Flow IAT Min         float64
Fwd IAT Tot          float64
Fwd IAT Mean         float64
Fwd IAT Std          float64
Fwd IAT Max          float64
Fwd IAT Min          float64
Bwd IAT Tot          float64
Bwd IAT Mean         float64
Bwd IAT Std          float64
Bwd IAT Max          float64
Bwd IAT Min          float64
Fwd PSH Flags        float64
Bwd PSH Flags        float64
Fwd URG Flags        float64
Bwd URG Flags        float64
Fwd Header Len       float64
Bwd Header Len       float64
Fwd Pkts/s           float64
Bwd Pkts/s           float64
Pkt Len Min          float64
Pkt Len Max          float64
Pkt Len Mean         float64
Pkt Len Std          float64
Pkt Len Var          float64
FIN Flag Cnt         float64
SYN Flag Cnt         float64
RST Flag Cnt         float64
PSH Flag Cnt         float64
ACK Flag Cnt         float64
URG Flag Cnt         float64
CWE Flag Count       float64
ECE Flag Cnt         float64
Down/Up Ratio        float64
Pkt Size Avg         float64
Fwd Seg Size Avg     float64
Bwd Seg Size Avg     float64
Fwd Byts/b Avg       float64
Fwd Pkts/b Avg       float64
Fwd Blk Rate Avg     float64
Bwd Byts/b Avg       float64
Bwd Pkts/b Avg       float64
Bwd Blk Rate Avg     float64
Subflow Fwd Pkts     float64
Subflow Fwd Byts     float64
Subflow Bwd Pkts     float64
Subflow Bwd Byts     float64
Init Fwd Win Byts    float64
Init Bwd Win Byts    float64
Fwd Act Data Pkts    float64
Fwd Seg Size Min     float64
Active Mean          float64
Active Std           float64
Active Max           float64
```

```
Active Min          float64
Idle Mean           float64
Idle Std            float64
Idle Max            float64
Idle Min            float64
Label                object
dtype: object
```

In [ ]:

```
# final_dataset.drop(['Flow ID', 'Src IP', 'Src Port', 'Dst IP'], axis=1, inplace=Tr
```

checking for Null and Inf values

In [ ]:

```
final_dataset[final_dataset.isnull().any(axis=1)]
```

Out[ ]:

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min
3445	49738.0	6.0	1.519290e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
3767	49812.0	6.0	1.519291e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
3819	49862.0	6.0	1.519291e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
3916	49844.0	6.0	1.519291e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
4060	49971.0	6.0	1.519291e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
4065	49972.0	6.0	1.519291e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
4080	49985.0	6.0	1.519291e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
4180	50026.0	6.0	1.519292e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
4204	50017.0	6.0	1.519292e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
4223	50028.0	6.0	1.519292e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
5034	50351.0	6.0	1.519295e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
5058	50403.0	6.0	1.519295e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
5248	50495.0	6.0	1.519295e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
5467	50516.0	6.0	1.519295e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
5860	50779.0	6.0	1.519296e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
5905	50702.0	6.0	1.519296e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
5926	50740.0	6.0	1.519296e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
6320	51063.0	6.0	1.519297e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
6373	51004.0	6.0	1.519297e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
6793	51198.0	6.0	1.519299e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
7364	51431.0	6.0	1.519302e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
7365	51432.0	6.0	1.519302e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
7439	51396.0	6.0	1.519303e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
7441	51400.0	6.0	1.519303e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
7568	51549.0	6.0	1.519303e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min	A
7635	51525.0	6.0	1.519303e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
7654	51530.0	6.0	1.519303e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
7687	51554.0	6.0	1.519303e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
8380	51863.0	6.0	1.519261e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
8451	51747.0	6.0	1.519261e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
8475	51824.0	6.0	1.519261e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
8771	52009.0	6.0	1.519262e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
8940	52095.0	6.0	1.519263e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
9023	52111.0	6.0	1.519263e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
9743	52469.0	6.0	1.519264e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
9870	52445.0	6.0	1.519264e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
9977	52494.0	6.0	1.519264e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
2969	49490.0	6.0	1.519374e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
3265	49582.0	6.0	1.519375e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
3269	49584.0	6.0	1.519375e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
3275	49590.0	6.0	1.519375e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
3305	49596.0	6.0	1.519375e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
3646	49891.0	6.0	1.519376e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
3728	49888.0	6.0	1.519376e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
4149	50032.0	6.0	1.519379e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
4201	50054.0	6.0	1.519379e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
4332	50135.0	6.0	1.519379e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
4440	50133.0	6.0	1.519379e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
4614	50233.0	6.0	1.519380e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
4616	50236.0	6.0	1.519380e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
4624	50244.0	6.0	1.519380e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
5443	50360.0	6.0	1.519381e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
5866	50537.0	6.0	1.519382e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
6486	50745.0	6.0	1.519383e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
6539	50719.0	6.0	1.519383e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
6546	50696.0	6.0	1.519383e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
6741	50838.0	6.0	1.519383e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
8517	51231.0	6.0	1.519386e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	
8532	51243.0	6.0	1.519386e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0	

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min
8629	51291.0	6.0	1.519387e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
8759	51308.0	6.0	1.519387e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
9926	51640.0	6.0	1.519348e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
9976	51579.0	6.0	1.519348e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0
9998	51603.0	6.0	1.519348e+18	0.0	2.0	0.0	0.0	0.0	0.0	0.0	...	20.0

64 rows × 80 columns

In [ ]:

```
final_dataset['Flow Byts/s']=final_dataset['Flow Byts/s'].replace([np.inf, -np.inf],
final_dataset['Flow Pkts/s']=final_dataset['Flow Pkts/s'].replace([np.inf, -np.inf],
```

In [ ]:

```
#final_dataset.replace("Infinity", 0, inplace=True)
final_dataset=final_dataset.replace([np.inf, -np.inf], np.nan)
```

In [ ]:

```
constant_features=['Bwd PSH Flags','Bwd URG Flags','Fwd Byts/b Avg','Fwd Pkts/b Avg'
#these contant features have been identified by using the SelectKBest sklearn librar
#these features just account to constant feature value of 0, which will not help in
constant_features=final_dataset[constant_features]
constant_features.head()
```

Out[ ]:

	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
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [ ]:

```
#these features just account to a constant  feature value of 0, which will not help

final_dataset.drop(['Bwd PSH Flags'],axis=1,inplace=True)
final_dataset.drop(['Bwd URG Flags'],axis=1,inplace=True)
final_dataset.drop(['Fwd Byts/b Avg'],axis=1,inplace=True)
final_dataset.drop(['Fwd Pkts/b Avg'],axis=1,inplace=True)
final_dataset.drop(['Fwd Blk Rate Avg'],axis=1,inplace=True)
final_dataset.drop(['Bwd Byts/b Avg'],axis=1,inplace=True)
final_dataset.drop(['Bwd Pkts/b Avg'],axis=1,inplace=True)
final_dataset.drop(['Bwd Blk Rate Avg'],axis=1,inplace=True)
```

In [ ]:

```
final_dataset.shape
```

Out[ ]: (20000, 72)

In [ ]:
```
# drop duplicate rows
final_dataset =  final_dataset.drop_duplicates(keep="first")
final_dataset.shape
```

Out[ ]: (19989, 72)

In [ ]:
```
#final_dataset=final_dataset.replace(',,', np.nan, inplace=False) #replace blanks in
import numpy as np #converting the blank spaces into NaN values

final_dataset.replace(r'^\s*$', np.nan, regex=True)
```

In [ ]:
```
final_dataset.isnull().sum()
```

Out[ ]:
```
Dst Port              0
Protocol              0
Timestamp             0
Flow Duration         0
Tot Fwd Pkts          0
Tot Bwd Pkts          0
TotLen Fwd Pkts       0
TotLen Bwd Pkts       0
Fwd Pkt Len Max       0
Fwd Pkt Len Min       0
Fwd Pkt Len Mean      0
Fwd Pkt Len Std       0
Bwd Pkt Len Max       0
Bwd Pkt Len Min       0
Bwd Pkt Len Mean      0
Bwd Pkt Len Std       0
Flow Byts/s         140
Flow Pkts/s         140
Flow IAT Mean         0
Flow IAT Std          0
Flow IAT Max          0
Flow IAT Min          0
Fwd IAT Tot           0
Fwd IAT Mean          0
Fwd IAT Std           0
Fwd IAT Max           0
Fwd IAT Min           0
Bwd IAT Tot           0
Bwd IAT Mean          0
Bwd IAT Std           0
Bwd IAT Max           0
Bwd IAT Min           0
Fwd PSH Flags         0
Fwd URG Flags         0
Fwd Header Len        0
Bwd Header Len        0
Fwd Pkts/s            0
Bwd Pkts/s            0
Pkt Len Min           0
Pkt Len Max           0
Pkt Len Mean          0
Pkt Len Std           0
Pkt Len Var           0
FIN Flag Cnt          0
SYN Flag Cnt          0
RST Flag Cnt          0
PSH Flag Cnt          0
ACK Flag Cnt          0
```

```
URG Flag Cnt          0
CWE Flag Count        0
ECE Flag Cnt          0
Down/Up Ratio         0
Pkt Size Avg          0
Fwd Seg Size Avg      0
Bwd Seg Size Avg      0
Subflow Fwd Pkts      0
Subflow Fwd Byts      0
Subflow Bwd Pkts      0
Subflow Bwd Byts      0
Init Fwd Win Byts     0
Init Bwd Win Byts     0
Fwd Act Data Pkts     0
Fwd Seg Size Min      0
Active Mean           0
Active Std            0
Active Max            0
Active Min            0
Idle Mean             0
Idle Std              0
Idle Max              0
Idle Min              0
Label                 0
dtype: int64
```

In [ ]:

```
final_dataset=final_dataset.replace(np.nan, 0)
```

In [ ]:

```
final_dataset.isnull().any() #Now we get no null features on these dataset
```

Out[ ]:

```
Dst Port             False
Protocol             False
Timestamp            False
Flow Duration        False
Tot Fwd Pkts         False
Tot Bwd Pkts         False
TotLen Fwd Pkts      False
TotLen Bwd Pkts      False
Fwd Pkt Len Max      False
Fwd Pkt Len Min      False
Fwd Pkt Len Mean     False
Fwd Pkt Len Std      False
Bwd Pkt Len Max      False
Bwd Pkt Len Min      False
Bwd Pkt Len Mean     False
Bwd Pkt Len Std      False
Flow Byts/s          False
Flow Pkts/s          False
Flow IAT Mean        False
Flow IAT Std         False
Flow IAT Max         False
Flow IAT Min         False
Fwd IAT Tot          False
Fwd IAT Mean         False
Fwd IAT Std          False
Fwd IAT Max          False
Fwd IAT Min          False
Bwd IAT Tot          False
Bwd IAT Mean         False
Bwd IAT Std          False
Bwd IAT Max          False
Bwd IAT Min          False
Fwd PSH Flags        False
Fwd URG Flags        False
Fwd Header Len       False
Bwd Header Len       False
```

```
Fwd Pkts/s           False
Bwd Pkts/s           False
Pkt Len Min          False
Pkt Len Max          False
Pkt Len Mean         False
Pkt Len Std          False
Pkt Len Var          False
FIN Flag Cnt         False
SYN Flag Cnt         False
RST Flag Cnt         False
PSH Flag Cnt         False
ACK Flag Cnt         False
URG Flag Cnt         False
CWE Flag Count       False
ECE Flag Cnt         False
Down/Up Ratio        False
Pkt Size Avg         False
Fwd Seg Size Avg     False
Bwd Seg Size Avg     False
Subflow Fwd Pkts     False
Subflow Fwd Byts     False
Subflow Bwd Pkts     False
Subflow Bwd Byts     False
Init Fwd Win Byts    False
Init Bwd Win Byts    False
Fwd Act Data Pkts    False
Fwd Seg Size Min     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
```

In [ ]:

```
final_dataset.shape
```

Out[ ]:

```
(19989, 72)
```

3.Feature Engineering

encoding the Anomalous and Normal values as 0 and 1 to visualize

In [ ]:

```
final_dataset.replace(to_replace=['Infilteration','Bot','DoS attacks-GoldenEye','DoS
```

In [ ]:

```
final_dataset.replace(to_replace=['SQL Injection'],value=1,inplace=True)
final_dataset['Label'].value_counts()
```

Out[ ]:

```
0    19902
1       87
Name: Label, dtype: int64
```

Splitting the Xi and Yi to apply the Permutation Importance model and identify the importanat features to further apply model

In [ ]:

```
y = final_dataset['Label']
X = final_dataset.drop(['Label'],axis=1)
```

In [ ]:
```
# pip install eli5
```

In [ ]:
```
from sklearn.model_selection import  train_test_split
from sklearn.ensemble import RandomForestClassifier
from eli5.sklearn import PermutationImportance
from sklearn.feature_selection import SelectFromModel
```

In [ ]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)
```

3.1 Using Random Forest Feature Importance to identify the imporatant features

In [ ]:
```
model = RandomForestClassifier() #checking Randome forest feature importance
model.fit(X_train,Y_train)
columns = X_train.columns
coefficients = model.feature_importances_.reshape(X_train.columns.shape[0], 1)
absCoefficients = abs(coefficients)
fullList = pd.concat((pd.DataFrame(columns, columns = ['Variable']), pd.DataFrame(ab
print('RandomForestClassifier - Feature Importance:')
print('\n',fullList,'\n')
```

```
RandomForestClassifier - Feature Importance:

              Variable  absCoefficient
15      Bwd Pkt Len Std    1.146406e-01
2             Timestamp    9.044231e-02
59    Init Fwd Win Byts    5.124514e-02
3         Flow Duration    3.450993e-02
36            Fwd Pkts/s    3.287459e-02
21         Flow IAT Min    3.260145e-02
18        Flow IAT Mean    3.021814e-02
17           Flow Pkts/s    2.793091e-02
22          Fwd IAT Tot    2.771458e-02
26          Fwd IAT Min    2.690570e-02
23         Fwd IAT Mean    2.667217e-02
12      Bwd Pkt Len Max    2.626049e-02
31          Bwd IAT Min    2.513716e-02
20         Flow IAT Max    2.498070e-02
25          Fwd IAT Max    2.420785e-02
39          Pkt Len Max    2.365301e-02
0              Dst Port    2.337180e-02
41          Pkt Len Std    2.197412e-02
35       Bwd Header Len    1.957807e-02
60    Init Bwd Win Byts    1.920716e-02
42          Pkt Len Var    1.856343e-02
19         Flow IAT Std    1.823351e-02
37           Bwd Pkts/s    1.780867e-02
6       TotLen Fwd Pkts    1.640153e-02
10     Fwd Pkt Len Mean    1.615546e-02
53     Fwd Seg Size Avg    1.302869e-02
24          Fwd IAT Std    1.274016e-02
58     Subflow Bwd Byts    1.250281e-02
7       TotLen Bwd Pkts    1.221033e-02
56     Subflow Fwd Byts    1.166906e-02
8       Fwd Pkt Len Max    1.105418e-02
5          Tot Bwd Pkts    1.055562e-02
45         RST Flag Cnt    1.003097e-02
57     Subflow Bwd Pkts    9.134310e-03
54     Bwd Seg Size Avg    7.907639e-03
30          Bwd IAT Max    7.356841e-03
34       Fwd Header Len    7.189914e-03
27          Bwd IAT Tot    7.179748e-03
4          Tot Fwd Pkts    6.780407e-03
```

```
50        ECE Flag Cnt    6.667199e-03
52        Pkt Size Avg    6.508464e-03
11     Fwd Pkt Len Std    6.503977e-03
55    Subflow Fwd Pkts    6.497728e-03
14    Bwd Pkt Len Mean    6.384657e-03
40        Pkt Len Mean    6.139636e-03
16         Flow Byts/s    5.404535e-03
29         Bwd IAT Std    5.282424e-03
28        Bwd IAT Mean    4.878254e-03
48        URG Flag Cnt    4.380429e-03
51       Down/Up Ratio    4.165590e-03
61   Fwd Act Data Pkts    3.467172e-03
66          Active Min    8.628283e-04
46        PSH Flag Cnt    6.773740e-04
67           Idle Mean    3.160971e-04
47        ACK Flag Cnt    3.006927e-04
65          Active Max    2.972738e-04
68            Idle Std    2.061140e-04
62    Fwd Seg Size Min    1.876389e-04
70            Idle Min    1.071147e-04
44        SYN Flag Cnt    9.588249e-05
32       Fwd PSH Flags    3.191866e-05
1             Protocol    7.551260e-06
43        FIN Flag Cnt    2.905247e-07
13     Bwd Pkt Len Min    0.000000e+00
9      Fwd Pkt Len Min    0.000000e+00
49      CWE Flag Count    0.000000e+00
63         Active Mean    0.000000e+00
64          Active Std    0.000000e+00
38         Pkt Len Min    0.000000e+00
69            Idle Max    0.000000e+00
33       Fwd URG Flags    0.000000e+00
```

3.2 Computing the feature importance using Permuatation Importance

In [ ]:

```
X_train=np.asarray(X_train)
X_test=np.asarray(X_test)
Y_train=np.asarray(Y_train)
Y_test=np.asarray(Y_test)

sel = SelectFromModel(PermutationImportance(RandomForestClassifier(), cv=5)).fit(X_t
X_train2 = sel.transform(X_train)
X_test2 = sel.transform(X_test)
```

In [ ]:

```
model = RandomForestClassifier()
model.fit(X_train2,Y_train) # Needed to initialize coef_ or feature_importances_
coefficients = model.feature_importances_
absCoefficients = abs(coefficients)
Perm_imp = pd.concat((pd.DataFrame(columns, columns = ['Variable']), pd.DataFrame(ab

print('\n',Perm_imp,'\n')
```

```
              Variable  absCoefficient
1             Protocol        0.116096
4         Tot Fwd Pkts        0.073925
19        Flow IAT Std        0.065110
16         Flow Byts/s        0.064782
9      Fwd Pkt Len Min        0.062916
26         Fwd IAT Min        0.056848
6      TotLen Fwd Pkts        0.055630
7      TotLen Bwd Pkts        0.051923
10    Fwd Pkt Len Mean        0.050018
11     Fwd Pkt Len Std        0.037165
0             Dst Port        0.035942
25         Fwd IAT Max        0.031597
```

```
8     Fwd Pkt Len Max        0.031362
17         Flow Pkts/s       0.028984
14    Bwd Pkt Len Mean       0.028864
18      Flow IAT Mean        0.027855
3       Flow Duration        0.026688
2            Timestamp       0.025919
15     Bwd Pkt Len Std       0.024939
5        Tot Bwd Pkts        0.024111
24        Fwd IAT Std        0.023531
20       Flow IAT Max        0.020475
12     Bwd Pkt Len Max       0.014050
22        Fwd IAT Tot        0.008888
13     Bwd Pkt Len Min       0.007268
23       Fwd IAT Mean        0.003550
21       Flow IAT Min        0.001564
27        Bwd IAT Tot             NaN
28       Bwd IAT Mean             NaN
29        Bwd IAT Std             NaN
30        Bwd IAT Max             NaN
31        Bwd IAT Min             NaN
32      Fwd PSH Flags             NaN
33      Fwd URG Flags             NaN
34     Fwd Header Len             NaN
35     Bwd Header Len             NaN
36         Fwd Pkts/s             NaN
37         Bwd Pkts/s             NaN
38        Pkt Len Min             NaN
39        Pkt Len Max             NaN
40       Pkt Len Mean             NaN
41        Pkt Len Std             NaN
42        Pkt Len Var             NaN
43       FIN Flag Cnt             NaN
44       SYN Flag Cnt             NaN
45       RST Flag Cnt             NaN
46       PSH Flag Cnt             NaN
47       ACK Flag Cnt             NaN
48       URG Flag Cnt             NaN
49     CWE Flag Count             NaN
50       ECE Flag Cnt             NaN
51      Down/Up Ratio             NaN
52       Pkt Size Avg             NaN
53    Fwd Seg Size Avg            NaN
54    Bwd Seg Size Avg            NaN
55    Subflow Fwd Pkts            NaN
56    Subflow Fwd Byts            NaN
57    Subflow Bwd Pkts            NaN
58    Subflow Bwd Byts            NaN
59   Init Fwd Win Byts            NaN
60   Init Bwd Win Byts            NaN
61   Fwd Act Data Pkts            NaN
62    Fwd Seg Size Min            NaN
63        Active Mean             NaN
64         Active Std             NaN
65         Active Max             NaN
66         Active Min             NaN
67          Idle Mean             NaN
68           Idle Std             NaN
69           Idle Max             NaN
70           Idle Min             NaN
```

Dropping the least important features from the dataset

In [ ]:

```
least_features=Perm_imp.iloc[50 :,0] #selecting the bottom 21 features
least_features
```

Out[ ]:

```
50          ECE Flag Cnt
51         Down/Up Ratio
```

```
52          Pkt Size Avg
53      Fwd Seg Size Avg
54      Bwd Seg Size Avg
55      Subflow Fwd Pkts
56      Subflow Fwd Byts
57      Subflow Bwd Pkts
58      Subflow Bwd Byts
59     Init Fwd Win Byts
60     Init Bwd Win Byts
61     Fwd Act Data Pkts
62      Fwd Seg Size Min
63           Active Mean
64            Active Std
65            Active Max
66            Active Min
67             Idle Mean
68              Idle Std
69              Idle Max
70              Idle Min
Name: Variable, dtype: object
```

In [ ]:

```
data=least_features.tolist() #concerting the pandas series to list
for i in data:#dropping the list of features from the dataset
  final_dataset.drop(labels=[i],axis=1,inplace=True)
```

In [ ]:

```
final_dataset.shape
```

Out[ ]:

```
(19989, 51)
```

In [ ]:

```
Y_Labels = final_dataset['Label'] #Splitting the Xi and Yi to apply the Permutation
X_data = final_dataset.drop(['Label'],axis=1)
```

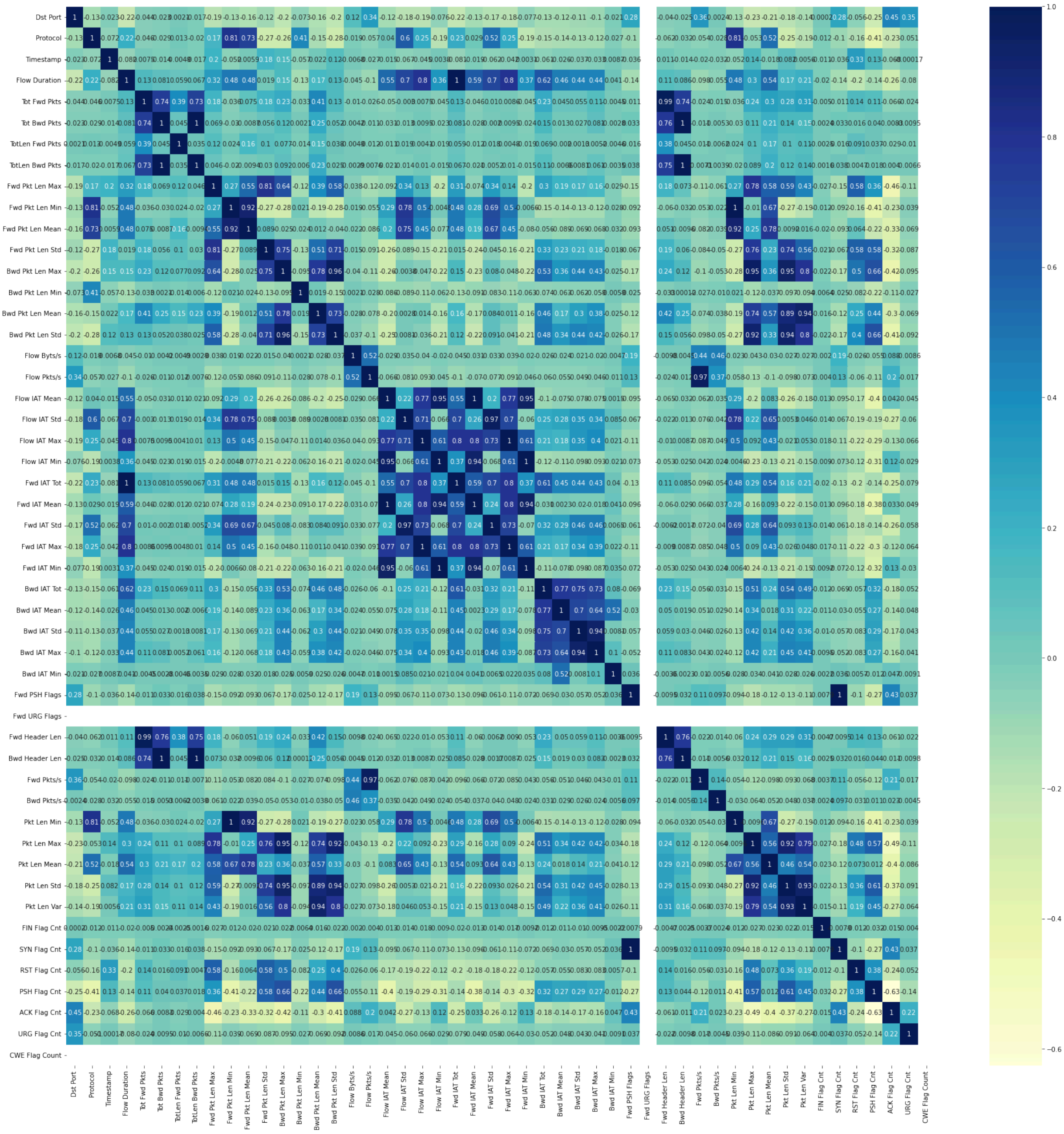
3.3 Computing corelation between the features and drop the highly corealted features

In [ ]:

```
from matplotlib import pyplot as plt
import seaborn as sns
fig= plt.figure(figsize=(30,30))
sns.heatmap(X_data.corr(), annot=True,cmap="YlGnBu")
```

Out[ ]:

```
<AxesSubplot:>
```



In [ ]:

```
def get_correlation_high(X_data,threshold): #Findout the features with the correlati
    corr_col=set()
    corrmat=X_data.corr()
    for i in range(len(corrmat.columns)):
        for j in range(i):
            if abs(corrmat.iloc[i,j])>threshold:
                colname=corrmat.columns[i]
                corr_col.add(colname)
    return corr_col
```

In [ ]:

```
corelated_features=get_correlation_high(X_data,0.95) #spot the highly co realted fea
corelated_features
```

Out[ ]:

```
{'Bwd Header Len',
 'Bwd Pkt Len Std',
 'Flow IAT Min',
 'Fwd Header Len',
 'Fwd IAT Max',
 'Fwd IAT Mean',
 'Fwd IAT Min',
 'Fwd IAT Std',
```

```
 'Fwd IAT Tot',
 'Fwd Pkts/s',
 'Pkt Len Max',
 'Pkt Len Min',
 'SYN Flag Cnt',
 'TotLen Bwd Pkts'}
```

In [ ]:

```
corr=list(corelated_features)
corr
```

Out[ ]:

```
['Flow IAT Min',
 'Fwd IAT Tot',
 'Fwd IAT Min',
 'SYN Flag Cnt',
 'Pkt Len Min',
 'Fwd IAT Mean',
 'TotLen Bwd Pkts',
 'Bwd Header Len',
 'Pkt Len Max',
 'Fwd IAT Std',
 'Fwd Header Len',
 'Fwd Pkts/s',
 'Fwd IAT Max',
 'Bwd Pkt Len Std']
```

In [ ]:

```
for i in corr: #dropping the highly corelated features from the dataset
  final_dataset.drop(labels=[i],axis=1,inplace=True)
```

In [ ]:

```
final_dataset.shape
```

Out[ ]:

```
(19989, 37)
```

In [ ]:

```
final_dataset.columns
```

Out[ ]:

```
Index(['Dst Port', 'Protocol', 'Timestamp', 'Flow Duration', 'Tot Fwd Pkts',
       'Tot Bwd Pkts', 'TotLen Fwd 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', 'Flow Byts/s', 'Flow Pkts/s',
       'Flow IAT Mean', 'Flow IAT Std', 'Flow IAT Max', 'Bwd IAT Tot',
       'Bwd IAT Mean', 'Bwd IAT Std', 'Bwd IAT Max', 'Bwd IAT Min',
       'Fwd PSH Flags', 'Fwd URG Flags', 'Bwd Pkts/s', 'Pkt Len Mean',
       'Pkt Len Std', 'Pkt Len Var', 'FIN Flag Cnt', 'RST Flag Cnt',
       'PSH Flag Cnt', 'ACK Flag Cnt', 'URG Flag Cnt', 'CWE Flag Count',
       'Label'],
      dtype='object')
```

4. Modeling

4.1 Splitting the datset to train and test

In [ ]:

```
y = final_dataset['Label'] #Splitting the Xi and Yi to apply the Permutation Importa
X = final_dataset.drop(['Label'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.30, random_sta
```

In [ ]:

```
from xgboost import XGBClassifier

xgb = XGBClassifier(n_estimators=100)
xgb.fit(X_train, y_train)
```

```
c:\Program Files\Python36\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The
use of label encoder in XGBClassifier is deprecated and will be removed in a future r
elease. To remove this warning, do the following: 1) Pass option use_label_encoder=Fa
lse when constructing XGBClassifier object; and 2) Encode your labels (y) as integers
starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[23:51:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/
learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with t
he objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set
eval_metric if you'd like to restore the old behavior.
```

Out[ ]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, make_scorer, accuracy_score
XGBpredictions = xgb.predict(X_test)
print('the accuracy of SCG Classfier with hinge loss:',accuracy_score(y_test,XGBpred
```

```
the accuracy of SCG Classfier with hinge loss: 0.9984992496248124
```

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
ADBClassifier = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),n_estimators=
ADBClassifier.fit(X_train, y_train)
```

Out[ ]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   n_estimators=200)
```

In [ ]:
```
ADBCpredictions = ADBClassifier.predict(X_test)
print('the accuracy of SCG Classfier with hinge loss:',accuracy_score(y_test,ADBCpre
```

```
the accuracy of SCG Classfier with hinge loss: 0.9998332499583125
```

4.2 Applying the Standard SVC model and checking the performance metrices

In [ ]:
```
from sklearn.svm import SVC
model = SVC()
model.fit(X_train, y_train)
```

Out[ ]:
```
SVC()
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, make_scorer, accuracy_score
predictions = model.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print('the accuracy of Standard SVC:',accuracy_score(y_test,predictions))

print(classification_report(y_test, predictions))
```

```
the accuracy of Standard SVC: 0.9953309988327497
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5969
           1       0.00      0.00      0.00        28
```

```
     accuracy                           1.00      5997
    macro avg       0.50      0.50      0.50      5997
 weighted avg       0.99      1.00      0.99      5997

c:\Program Files\Python36\lib\site-packages\sklearn\metrics\_classification.py:1248:
UndefinedMetricWarning: 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 behav
ior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Program Files\Python36\lib\site-packages\sklearn\metrics\_classification.py:1248:
UndefinedMetricWarning: 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 behav
ior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Program Files\Python36\lib\site-packages\sklearn\metrics\_classification.py:1248:
UndefinedMetricWarning: 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 behav
ior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Observation:

1. The standard SVC model is behaving worse with this data, the overall accuarcy has been just 0.5337
2. And clealry this model is overfitting by seeing the Precison and recall values.
3. Hence we should perform some hypertuning or use Kernal SVMs to deal with this data

4.3 Applying the SGD Classifier with Hinge Loss

In [ ]:

```python
from sklearn.linear_model import SGDClassifier
SGDmodel = SGDClassifier(loss="hinge", penalty="l2")
SGDmodel.fit(X_train, y_train)
```

Out[ ]:

```
SGDClassifier()
```

In [ ]:

```python
from sklearn.metrics import confusion_matrix, make_scorer, accuracy_score
SGDpredictions = SGDmodel.predict(X_test)
print('the accuracy of SCG Classfier with hinge loss:',accuracy_score(y_test,SGDpred
```

```
the accuracy of SCG Classfier with hinge loss: 0.9953309988327497
```

Observations:

The model performance is worse by appying the SGD classifier with hinge loss. Appyling SGD for this dataset will not be the best idea here

4.4 Applying the SGD Classifier with Hinge Loss on different iterations

In [ ]:

```python
n_iters = [5, 10, 20, 50, 100, 1000]
scores = []
for n_iter in n_iters:
    SGDmodel = SGDClassifier(loss="hinge", penalty="l2", max_iter=n_iter)
    SGDmodel.fit(X_train, y_train)
    SGDpredictions = SGDmodel.predict(X_test)
    scores.append(accuracy_score(y_test,SGDpredictions))

plt.title("Effect of n_iter")
plt.xlabel("n_iter")
plt.ylabel("score")
plt.plot(n_iters, scores)
```

Observations:

1. The performance of the SVM SGD with hinge loss on diifrent iteration is also very poor.
2. With different iterations the performance is not improving , with increasing iterations the model score is staying stagnent here.

4.5 Applying the Logistic regression SGD Classifier with Log Loss on different iterations

In [ ]:
```
n_iters = [5, 10, 20, 50, 100, 1000]
scores = []
for n_iter in n_iters:
    SGDmodel = SGDClassifier(loss="log", penalty="l2", max_iter=n_iter)
    SGDmodel.fit(X_train, y_train)
    SGDpredictions = SGDmodel.predict(X_test)
    scores.append(accuracy_score(y_test,SGDpredictions))

plt.title("Effect of n_iter")
plt.xlabel("n_iter")
plt.ylabel("score")
plt.plot(n_iters, scores)
```

Observations:

1. With SGD Log loss the score is still worse, the accuracy is poor with diffrent iterations
2. The Accuracy score has stopped increasing after iterations

4.6 Applying the Decsion Trees

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
DT_clf = DecisionTreeClassifier(random_state=0)

DT_clf.fit(X_train, y_train)
DT_pred=DT_clf.predict(X_test)
print('the accuracy',accuracy_score(y_test,DT_pred))
```

```
the accuracy 0.9978322494580624
```

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
confusion_matrix(y_test,DT_pred)
```

Out[ ]:
```
array([[5966,    3],
       [  10,   18]], dtype=int64)
```

In [ ]:
```
print(classification_report(y_test, DT_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5969
           1       0.86      0.64      0.73        28

    accuracy                           1.00      5997
   macro avg       0.93      0.82      0.87      5997
weighted avg       1.00      1.00      1.00      5997
```

Observations:

1. The DT Algorithm seems to be performing better than the other SVM and LR models.
2. The model is producing better accuracy score that by applying SVM and LR

3. With confusion matrix we could see that the model is separating the bening and anamlous requests in a decent way.

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier as DT
from sklearn.model_selection import GridSearchCV

parameters = {'max_depth':[1,5,10,50],'min_samples_split':[5,10,100,500]}
DTclf= GridSearchCV(DT(),parameters)
DTclf.fit(X_train, y_train)
```

Out[ ]:
```
GridSearchCV(estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]})
```

In [ ]:
```
print('Best score: ',DTclf.best_score_)
print('Parameters with best score: ',DTclf.best_params_)
```

```
Best score:  0.9979273461547203
Parameters with best score:  {'max_depth': 50, 'min_samples_split': 5}
```

In [ ]:
```
Best_DT_clf = DT(max_depth=50, min_samples_split=10, random_state=0)
Best_DT_clf.fit(X_train, y_train)
H_DT_pred=Best_DT_clf.predict(X_test)
print('the accuracy',accuracy_score(y_test,H_DT_pred))
```

```
the accuracy 0.9976654994163748
```

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
confusion_matrix(y_test,DT_pred)
```

Out[ ]:
```
array([[5966,    3],
       [  10,   18]], dtype=int64)
```

In [ ]:
```
print(classification_report(y_test, DT_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5969
           1       0.86      0.64      0.73        28

    accuracy                           1.00      5997
   macro avg       0.93      0.82      0.87      5997
weighted avg       1.00      1.00      1.00      5997
```

Observation:

1. Post the Hyperparamater tuning and applying the best parameters the model accuracy had improved in a very small amount.
2. There is not a bigger but Hyperparameter tuning should be done for better optimized results

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
RF_model = RandomForestClassifier(n_estimators=100,random_state=0).fit(X_train, y_tr
RF_pred=RF_model.predict(X_test)
print('the accuracy of RF model',accuracy_score(y_test,RF_pred))
```

```
the accuracy of RF model 0.9983324995831249
```

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
confusion_matrix(y_test,RF_pred)
```

Out[ ]:
```
array([[5969,    0],
       [  10,   18]], dtype=int64)
```

In [ ]:
```
print(classification_report(y_test, RF_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5969
           1       1.00      0.64      0.78        28

    accuracy                           1.00      5997
   macro avg       1.00      0.82      0.89      5997
weighted avg       1.00      1.00      1.00      5997
```

Observation:

1. The DT based Ensemble model seems to be producing better results.
2. Comparing to the RF results with the DT results both are more are less the same. But DT seems to be producing better results in terms of seperating the Being and anamalous requests.
3. This is interpreted clearly via the confusion matrix
4. Hence DT will be the better algorithm here

Summary:

1. The DDOS dataset used here is a multiclass dataset due to severe imbalence in the class label we have built the primary model as a Binary classification model converting all the attacks as 0 and Bening requests to be 1.
2. We have come to a conclusion that the DT Algorithm can be the best algorithm for the Binary classification model.

Ensembling

In [ ]:
```
y = final_dataset['Label'] #Splitting the Xi and Yi for ensembling
X = final_dataset.drop(['Label'],axis=1)
```

In [ ]:
```
X1_train, X1_test, y1_train, y1_test = train_test_split(X,y,test_size = 0.2, random_
```

In [ ]:
```
data = pd.concat([X1_train, y1_train], axis=1) #taking the 80 split alone
data.shape
```

Out[ ]:
```
(15991, 37)
```

In [ ]:
```
EightyPer_Y=data['Label'] #seperating the class labels from 80% split
EightyPer_X=data.drop(['Label'],axis=1)
D1_X, D2_X, D1_Y, D2_Y = train_test_split(EightyPer_X,EightyPer_Y,test_size = 0.5, r
```

In [ ]:
```
D1=pd.concat([D1_X,D1_Y],axis=1) #Forming D1 and D2
D1.shape
```

```
print('the shape of D1',D1.shape)
D2=pd.concat([D2_X,D2_Y],axis=1)
print('the shape of D2',D2.shape)
```

```
the shape of D1 (7995, 37)
the shape of D2 (7996, 37)
```

In [ ]:

```
S1=D1.sample(n=60908, replace=True, random_state=1) # sampling with replacement on D
S2=D1.sample(n=60908, replace=True, random_state=1)
S3=D1.sample(n=60908, replace=True, random_state=1)
S4=D1.sample(n=60908, replace=True, random_state=1)
S5=D1.sample(n=60908, replace=True, random_state=1)
S6=D1.sample(n=60908, replace=True, random_state=1)
```

In [ ]:

```
S1y = S1['Label']  #samples and its coressponding class labels
S1X = S1.drop(['Label'],axis=1)
S2y = S2['Label']
S2X = S2.drop(['Label'],axis=1)
S3y = S3['Label']
S3X = S3.drop(['Label'],axis=1)
S4y = S4['Label']
S4X = S4.drop(['Label'],axis=1)
S5y = S5['Label']
S5X = S5.drop(['Label'],axis=1)
S6y = S6['Label']
S6X = S6.drop(['Label'],axis=1)
```

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier #fitting the base model DT with then
Base_model=DecisionTreeClassifier()

BS1=Base_model.fit(S1X,S1y)
BS2=Base_model.fit(S2X,S2y)
BS3=Base_model.fit(S3X,S3y)
BS4=Base_model.fit(S4X,S4y)
BS5=Base_model.fit(S5X,S5y)
BS6=Base_model.fit(S6X,S6y)
```

In [ ]:

```
model1=Base_model.predict(D2_X) #passing the D2 train to get model predict
model2=Base_model.predict(D2_X)
model3=Base_model.predict(D2_X)
model4=Base_model.predict(D2_X)
model5=Base_model.predict(D2_X)
model6=Base_model.predict(D2_X)
```

In [ ]:

```
BS1=model1.reshape(-1,1)
BS2=model2.reshape(-1,1)
BS3=model3.reshape(-1,1)
BS4=model4.reshape(-1,1)
BS5=model5.reshape(-1,1)
BS6=model6.reshape(-1,1)
```

In [ ]:

```
D_meta = np.vstack((BS1,BS2,BS3,BS4,BS5,BS6)) #creating Dataset out of the n-models ,
print('The length of the D_meta data',len(D_meta))
D_meta_data=D_meta[:60908] #sampling only the required datapoints
print('Lenght of meta data post Sampling ',len(D_meta_data))
```

```
The length of the D_meta data 47976
Lenght of meta data post Sampling  47976
```

In [ ]:

```
from xgboost import XGBClassifier #taking the XGB as the meta model and passing the 

xgb = XGBClassifier(n_estimators=100)
xgb.fit(D_meta_data, D2_Y)
```

In [ ]:

```
test_pred1=Base_model.predict(X1_test) #passing 20 percent test data which was split
test_pred2=Base_model.predict(X1_test)
test_pred3=Base_model.predict(X1_test)
test_pred4=Base_model.predict(X1_test)
test_pred5=Base_model.predict(X1_test)
test_pred6=Base_model.predict(X1_test)
```

In [ ]:

```
TP1=test_pred1.reshape(-1,1) #test prediction results
TP2=test_pred2.reshape(-1,1)
TP3=test_pred3.reshape(-1,1)
TP4=test_pred4.reshape(-1,1)
TP5=test_pred5.reshape(-1,1)
TP6=test_pred6.reshape(-1,1)
```

In [ ]:

```
D_test_meta = np.vstack((TP1,TP2,TP3,TP4,TP5,TP6)) #Stacking up the test prediction
```

In [ ]:

```
D_test_meta=D_test_meta[:30455]
```

In [ ]:

```
pred_final = xgb.predict(D_test_meta) #passing the final dataset to the meta model
```

Custom Implementation

In [ ]:

```
Y=pd.DataFrame(final_dataset['Label'],columns=['Label']) #Splitting the Xi and Yi fo
X = final_dataset.drop(['Label'],axis=1)
```

In [ ]:

```
X1_train, X1_test, y1_train, y1_test = train_test_split(X,y,test_size = 0.2, random_
```

In [ ]:

```
def sampling(n_estimators,D1_data):#sampling Randomly
  size=len(D1_data)
  samples=[]
  for i in range(n_estimators):
    S=D1_data.sample(n=size, replace=True, random_state=60000)
    samples.append(S)
  return samples

def splitting(samples,n_estimators):#splitting Xi and Yi
  Xi=[]
  Yi=[]
  for i in range(n_estimators):
    a=samples[i]
    S_Yi=a['Label']
    S_Xi=a.drop(['Label'],axis=1)

    Xi.append(S_Xi)
    Yi.append(S_Yi)
```

```
    return Xi,Yi

def modeling(n_estimators,Xi,Yi): #training on the base model
    from sklearn.tree import DecisionTreeClassifier
    Base_model=DecisionTreeClassifier()

    for i in range(n_estimators):
      BS=Base_model.fit(Xi[i],Yi[i])

    return BS

def base_pred(BS,D2_X_test,n_estimators):#pred DT with D2_X_test
    samples_prediction=[]

    for i in range(n_estimators):
      predictions=BS.predict(D2_X_test)
      samples_prediction.append(predictions)

    return samples_prediction



def reshape(samples_prediction,n_estimators):#reshape
    SP_meta=[]
    for i in range(n_estimators):
      SP_meta=samples_prediction[i].reshape(-1,1)
    return SP_meta

def xgb_model(meta_samples,D2_Y_test): #XGB
    from xgboost import XGBClassifier

    meta= XGBClassifier(n_estimators=100)
    meta.fit(meta_samples, D2_Y_test)

    return meta



def test_pred(Base_models,twenty_test_pred,n_estimators):#test pred

    test_prediction=[]
    for i in range(n_estimators):
      predictions=Base_models.predict(X_test)
      test_prediction.append(predictions)

    return test_prediction



def custom_ensemble(X_train,y_train,X_test,y1_test,n_estimators):

    D1_X_train,D2_X_test,D1_Y_train,D2_Y_test=train_test_split(X_train,y_train,test_si


    D1_data=pd.concat([D1_X_train,D1_Y_train],axis=1)
    D2_data=pd.concat([D2_X_test,D2_Y_test],axis=1)

    D_samples=sampling(n_estimators,D1_data)#sampling from D1
    Xi,Yi=splitting(D_samples,n_estimators)#split xi and yi of the samples
    Base_models=modeling(n_estimators,Xi,Yi)#training the DT model with the samples
    samples_prediction=base_pred(Base_models,D2_X_test,n_estimators)#precicting using
    meta_samples=reshape(samples_prediction,n_estimators)#reshape
    meta_model=xgb_model(meta_samples,D2_Y_test)#calling XGB model
    twenty_test_pred=base_pred(Base_models,X_test,n_estimators)
```

```
    tp_samples=reshape(twenty_test_pred,n_estimators)#reshape
    pred_final = meta_model.predict(tp_samples)
    from sklearn.metrics import accuracy_score
    acc=accuracy_score(y1_test,pred_final)

    print('the accuracy for the custom ensemble implementation',acc)
```

In [ ]:

```
custom_ensemble(X1_train,y1_train,X1_test,y1_test,40)
```

```
c:\Program Files\Python36\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The
use of label encoder in XGBClassifier is deprecated and will be removed in a future r
elease. To remove this warning, do the following: 1) Pass option use_label_encoder=Fa
lse when constructing XGBClassifier object; and 2) Encode your labels (y) as integers
starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[23:56:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/
learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with t
he objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set
eval_metric if you'd like to restore the old behavior.
the accuracy for the custom ensemble implementation 0.9964982491245623
```