Trojan Guard (Team 3)

Name	Section	BN	Contribution
Abdelrahman Mohamed Salem	2	1	Analysis of the feature
Youssef Said Ibrahim Rabie	2	40	Analysis of the feature
Ahmed Fawzy	1	7	Analysis of the models
Ahmed Alaa	1	6	Analysis of the models

Problem Motivation

Detection of Trojan Traffic using Binary Classification.

Problem Definition

In today's technologically advanced world, the threat of cyberattacks has become a significant concern for individuals, businesses, and governments alike. One of the most insidious types of attacks is the Trojan horse, a malicious software that disguises itself as a harmless file or program. Detecting and preventing these attacks is crucial to maintaining the security of our digital systems.

The Trojan horse can penetrate the system or network when they are invited by the users unknowingly through the visiting of unknown and malicious websites or installing the software. So, Trojan horse can be better understood and realized by monitoring the network and capturing the network packet. The present dataset contains the malicious (Trojan category) and benign network packets which can be used to detect Trojan horse.

Data-Set Link

https://www.kaggle.com/datasets/subhajournal/trojan-detection

Evaluation Metrics

1. Accuracy
2. Precision
3. Recall
4. F1-Score
5. Training Time
6. Model Size
7. Prediction Time
8. AUC-ROC graph

Results

Dataset shape

At the first sight, we have 85 feature with 177482 row, some of which are just constant values, some are in string format, some are just combination of different other included features, others are following a normal distribution, but most don't follow any kind of distribution as we will see.

Unimportant Features at first sight

After examining the dataset, we found that the following features are unimportant:

1. **Timestamp**: it's a digital record of the date and time when the frame transmitted but for simplicity, we decided to drop this feature as it has nothing to do with today's attacks, so we didn't want the dates of the past affect the prediction of today's attacks.
2. **Flow ID:** it's just an identifier in string format for the transmission and it's written as "<source/destination ip> - <source/destination ip> - <destination/source port> - <destination/source port> - <protocol>". Which makes this feature a string representation for a mix between the features named "source ip", "destination ip", "destination port", "source port", "protocol".
3. **record_id:** it represents the id of the record and it has nothing to do with the class of the record.

The above 3 features are marked as non-important as they are either some identifier to the record or combination of other records.

Train-Split criteria

We decided as a rule of thumb to first divide the data into 75% training and 25% testing where the analysis will be done only to the training data and we will use the results of the analysis of training data to deal with testing data. We used stratify strategy which make ratio of Class "Trojan" to the ratio of Class "Benign" in training data is equal to that of testing data.

Training Data at first Sight

As we examine the training data initially, we see that there are some features that have a normal distribution while many others have the value 0 as their Q1, Q2 and Q3 which means that the vast majority of this feature just have the constant value 0.

	Source Port	Destination Port	Protocol	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	...	act_data_pkt_fwd	min_seg_size_forward	Active Mean	Active Std
count	133111.00	133111.00	133111.00	133111.00	133111.00	133111.00	133111.00	133111.00	133111.00	133111.00	...	133111.00	133111.00	133111.00	133111.00
mean	38677.25	5728.70	8.29	11627116.90	6.76	9.82	812.06	10099.82	230.86	13.43	...	1.68	27.80	142253.96	20455.36
std	18396.81	15112.31	4.54	23012414.79	30.88	108.27	5990.73	154521.47	392.28	68.89	...	6.89	6.66	784640.82	247120.01
min	0.00	0.00	0.00	2.00	1.00	0.00	0.00	0.00	0.00	0.00	...	0.00	-1.00	0.00	0.00
25%	34210.00	80.00	6.00	47124.50	1.00	0.00	0.00	0.00	0.00	0.00	...	0.00	20.00	0.00	0.00
50%	43105.00	443.00	6.00	489011.00	2.00	1.00	31.00	31.00	31.00	0.00	...	0.00	32.00	0.00	0.00
75%	52145.00	443.00	6.00	10621120.00	5.00	4.00	456.00	495.00	360.00	19.00	...	1.00	32.00	0.00	0.00
max	65530.00	64872.00	17.00	119999100.00	4502.00	12951.00	846614.00	18684972.00	1460.00	1460.00	...	586.00	60.00	30594213.00	15999508.03

Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	...	act_data_pkt_fwd	min_seg_size_forward	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min
133111.00	133111.00	133111.00	133111.00	...	133111.00	133111.00	133111.00	133111.00	133111.00	133111.00	133111.00	133111.00	133111.00	133111.00
812.06	10099.82	230.86	13.43	...	1.68	27.80	142253.96	20455.36	161871.32	126604.05	4527301.58	396093.59	4876552.46	4225439.01
5990.73	154521.47	392.28	68.89	...	6.89	6.66	784640.82	247120.01	880394.66	756249.31	15417940.75	3209320.84	16253085.52	15043254.66
0.00	0.00	0.00	0.00	...	0.00	-1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	...	0.00	20.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
31.00	31.00	31.00	0.00	...	0.00	32.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
456.00	495.00	360.00	19.00	...	1.00	32.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
846614.00	18684972.00	1460.00	1460.00	...	586.00	60.00	30594213.00	15999508.03	30594213.00	30594213.00	119773643.00	74451824.53	119773643.00	119773643.00

Upon examining the datatypes of the features of the dataset, we found that nearly all the feature have either numerical or decimal values except for only 2 features (Source and destination IP)

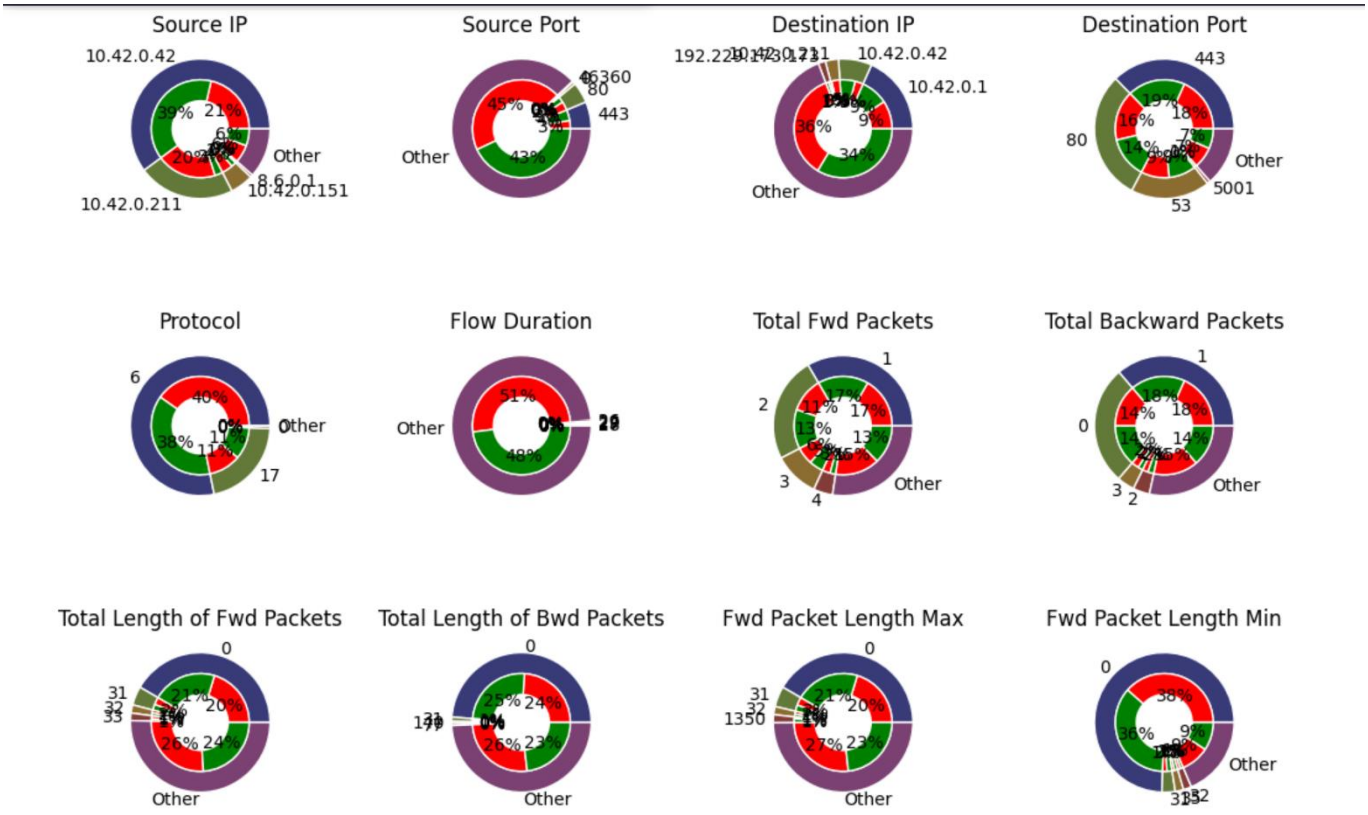
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 133111 entries, 0 to 133110
Data columns (total 82 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   Source IP                    133111 non-null  object
 1   Source Port                  133111 non-null  int64
 2   Destination IP               133111 non-null  object
 3   Destination Port             133111 non-null  int64
 4   Protocol                     133111 non-null  int64
 5   Flow Duration                133111 non-null  int64
 6   Total Fwd Packets            133111 non-null  int64
 7   Total Backward Packets       133111 non-null  int64
 8   Total Length of Fwd Packets  133111 non-null  int64
 9   Total Length of Bwd Packets  133111 non-null  int64
 10  Fwd Packet Length Max        133111 non-null  int64
 11  Fwd Packet Length Min        133111 non-null  int64
 12  Fwd Packet Length Mean       133111 non-null  float64
 13  Fwd Packet Length Std        133111 non-null  float64
 14  Bwd Packet Length Max        133111 non-null  int64
 15  Bwd Packet Length Min        133111 non-null  int64
 16  Bwd Packet Length Mean       133111 non-null  float64
 17  Bwd Packet Length Std        133111 non-null  float64
 18  Flow Bytes/s                 133111 non-null  float64
 19  Flow Packets/s               133111 non-null  float64
 20  Flow IAT Mean                133111 non-null  float64
 21  Flow IAT Std                 133111 non-null  float64
 22  Flow IAT Max                 133111 non-null  int64
 23  Flow IAT Min                 133111 non-null  int64
 24  Fwd IAT Total                133111 non-null  int64
 25  Fwd IAT Mean                 133111 non-null  float64
 26  Fwd IAT Std                  133111 non-null  float64
 27  Fwd IAT Max                  133111 non-null  int64
 28  Fwd IAT Min                  133111 non-null  int64
 29  Bwd IAT Total                133111 non-null  int64
 30  Bwd IAT Mean                 133111 non-null  float64
 31  Bwd IAT Std                  133111 non-null  float64
 32  Bwd IAT Max                  133111 non-null  int64
 33  Bwd IAT Min                  133111 non-null  int64
 34  Fwd PSH Flags                133111 non-null  int64
 35  Bwd PSH Flags                133111 non-null  int64
 36  Fwd URG Flags                133111 non-null  int64
 37  Bwd URG Flags                133111 non-null  int64
 38  Fwd Header Length            133111 non-null  int64
 39  Bwd Header Length            133111 non-null  int64
 40  Fwd Packets/s                133111 non-null  float64
 41  Bwd Packets/s                133111 non-null  float64
 42  Min Packet Length            133111 non-null  int64
 43  Max Packet Length            133111 non-null  int64
 44  Packet Length Mean           133111 non-null  float64
 45  Packet Length Std            133111 non-null  float64
 46  Packet Length Variance       133111 non-null  float64
 47  FIN Flag Count               133111 non-null  int64
 48  SYN Flag Count               133111 non-null  int64
 49  RST Flag Count               133111 non-null  int64
 50  PSH Flag Count               133111 non-null  int64
 51  ACK Flag Count               133111 non-null  int64
 52  URG Flag Count               133111 non-null  int64
 53  CWE Flag Count               133111 non-null  int64
 54  ECE Flag Count               133111 non-null  int64
 55  Down/Up Ratio                133111 non-null  int64
 56  Average Packet Size          133111 non-null  float64
 57  Avg Fwd Segment Size         133111 non-null  float64
 58  Avg Bwd Segment Size         133111 non-null  float64
 59  Fwd Header Length.1          133111 non-null  int64
 60  Fwd Avg Bytes/Bulk           133111 non-null  int64
 61  Fwd Avg Packets/Bulk         133111 non-null  int64
 62  Fwd Avg Bulk Rate            133111 non-null  int64
 63  Bwd Avg Bytes/Bulk           133111 non-null  int64
 64  Bwd Avg Packets/Bulk         133111 non-null  int64
 65  Bwd Avg Bulk Rate            133111 non-null  int64
 66  Subflow Fwd Packets          133111 non-null  int64
 67  Subflow Fwd Bytes            133111 non-null  int64
 68  Subflow Bwd Packets          133111 non-null  int64
 69  Subflow Bwd Bytes            133111 non-null  int64
 70  Init_Win_bytes_forward       133111 non-null  int64
 71  Init_Win_bytes_backward      133111 non-null  int64
 72  act_data_pkt_fwd             133111 non-null  int64
 73  min_seg_size_forward         133111 non-null  int64
 74  Active Mean                  133111 non-null  float64
 75  Active Std                   133111 non-null  float64
 76  Active Max                   133111 non-null  int64
 77  Active Min                   133111 non-null  int64
 78  Idle Mean                    133111 non-null  float64
 79  Idle Std                     133111 non-null  float64
 80  Idle Max                     133111 non-null  int64
 81  Idle Min                     133111 non-null  int64
dtypes: float64(24), int64(56), object(2)
memory usage: 83.3+ MB
```

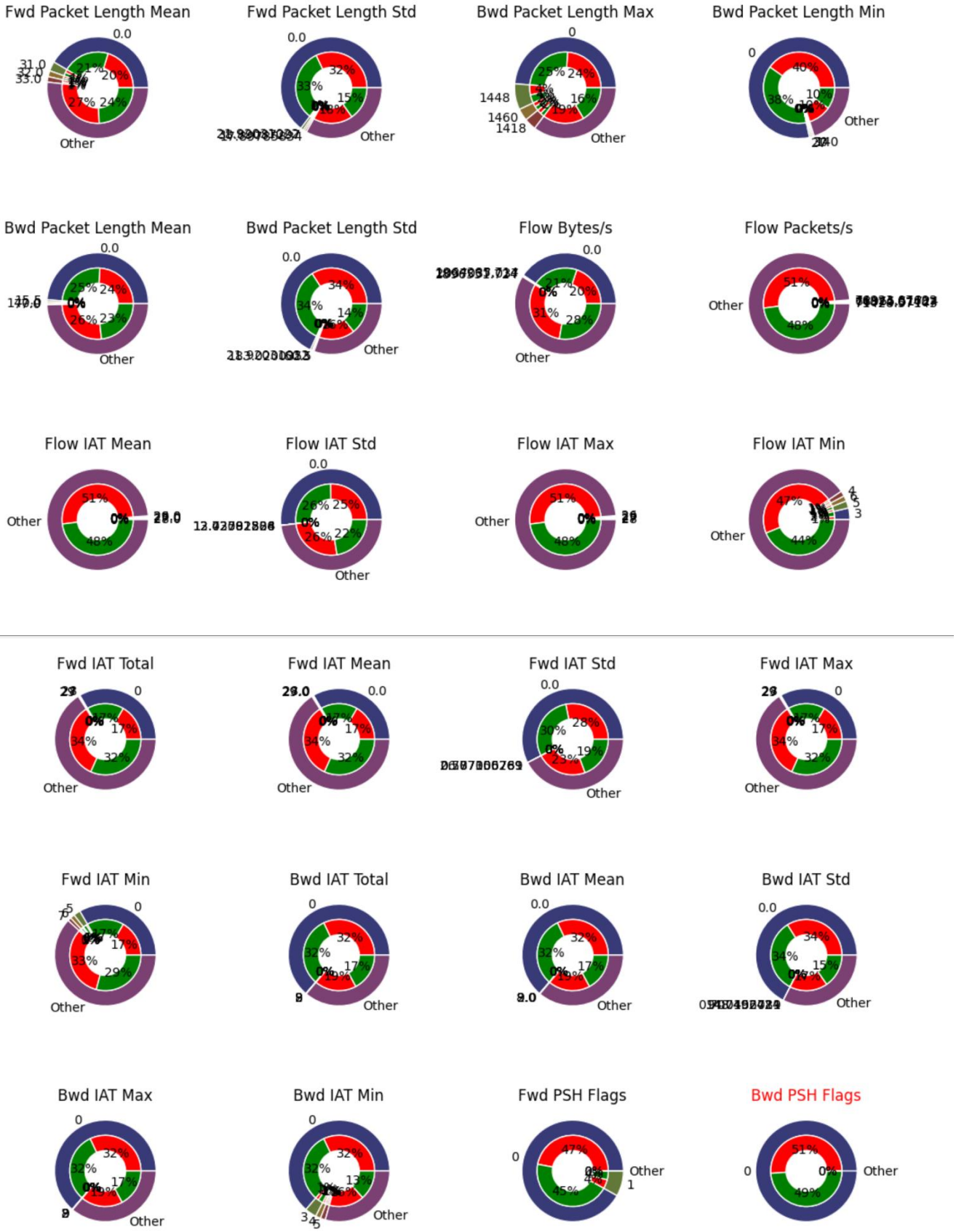
	0	1	2	3	4	5	6	7	8	9
Source IP	10.42.0.42	10.42.0.42	10.42.0.42	10.42.0.42	10.42.0.211	10.42.0.42	10.42.0.42	10.42.0.42	10.42.0.42	10.42.0.42
Source Port	48983	34014	1976	60140	57976	57903	60833	44359	34505	22638
Destination IP	123.125.29.220	172.217.12.202	10.42.0.1	202.77.129.161	216.58.219.227	63.250.200.101	10.42.0.1	174.35.73.148	202.77.129.150	10.42.0.1
Destination Port	80	443	53	80	443	443	53	80	80	53
Protocol	6	6	17	6	6	6	17	6	6	17
...	...	...	...	...	...	...	...	...	...	...
Active Min	0	1955451	0	0	241715	0	0	0	0	0
Idle Mean	0.00	58576706.00	0.00	0.00	14201349.00	0.00	0.00	0.00	0.00	0.00
Idle Std	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Idle Max	0	58576706	0	0	14201349	0	0	0	0	0
Idle Min	0	58576706	0	0	14201349	0	0	0	0	0

82 rows × 10 columns

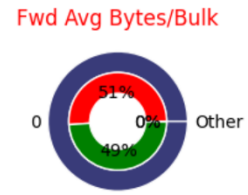
Non-Important features Due to Analysis

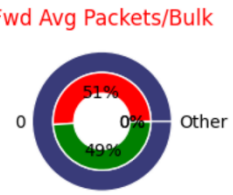
We plotted every feature in a pie manner where we choose the most common 4 values for every feature and the fifth is all other features. Then the values of every feature is split into 2 halves (red and green) where red represents the Trojan class and green represents Benign Class. We then colored the Title of every non-important feature that has nearly constant value equally distributed between the 2 classes to be red and next is the visualization.

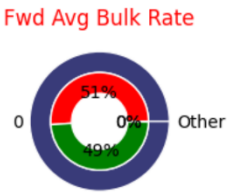


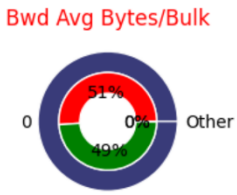

Fwd Packet Length Mean
Fwd Packet Length Std
Bwd Packet Length Max
Bwd Packet Length Min
Bwd Packet Length Mean
Bwd Packet Length Std
Flow Bytes/s
Flow Packets/s
Flow IAT Mean
Flow IAT Std
Flow IAT Max
Flow IAT Min
Fwd IAT Total
Fwd IAT Mean
Fwd IAT Std
Fwd IAT Max
Fwd IAT Min
Bwd IAT Total
Bwd IAT Mean
Bwd IAT Std
Bwd IAT Max
Bwd IAT Min
Fwd PSH Flags
Bwd PSH Flags
Other

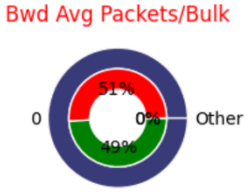

Fwd URG Flags
Bwd URG Flags
Fwd Header Length
Bwd Header Length
Fwd Packets/s
Bwd Packets/s
Min Packet Length
Max Packet Length
Packet Length Mean
Packet Length Std
Packet Length Variance
FIN Flag Count
SYN Flag Count
RST Flag Count
PSH Flag Count
ACK Flag Count
URG Flag Count
CWE Flag Count
ECE Flag Count
Down/Up Ratio
Average Packet Size
Avg Fwd Segment Size
Avg Bwd Segment Size
Fwd Header Length.1
Other

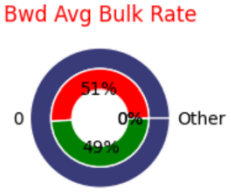

Fwd Avg Bytes/Bulk
51%
0%
49%
0
Other

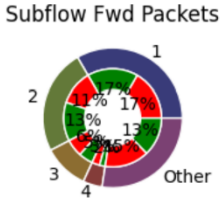

Fwd Avg Packets/Bulk
51%
0%
49%
0
Other

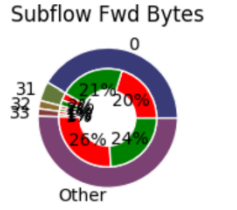

Fwd Avg Bulk Rate
51%
0%
49%
0
Other

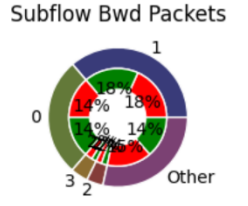

Bwd Avg Bytes/Bulk
51%
0%
49%
0
Other

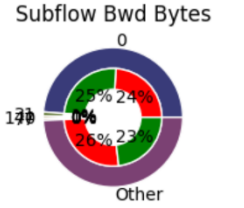

Bwd Avg Packets/Bulk
51%
0%
49%
0
Other

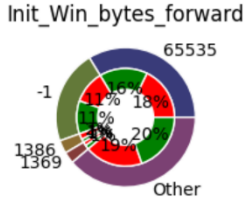

Bwd Avg Bulk Rate
51%
0%
49%
0
Other

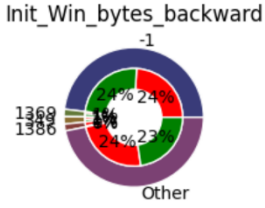

Subflow Fwd Packets
1
2
3
4
17%
17%
13%
13%
11%
Other

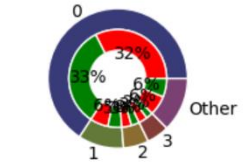

Subflow Fwd Bytes
0
31
21%
20%
26%
24%
Other

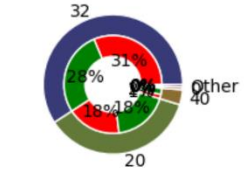

Subflow Bwd Packets
1
0
18%
18%
14%
14%
14%
3
2
Other

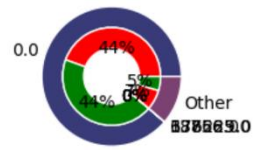

Subflow Bwd Bytes
0
25%
24%
26%
23%
Other

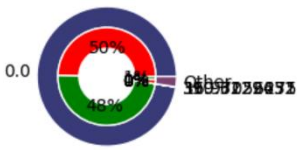

Init_Win_bytes_forward
65535
-1
16%
18%
11%
20%
19%
1386
1369
Other

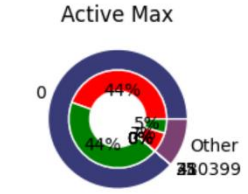

Init_Win_bytes_backward
-1
24%
24%
1369
1386
24%
23%
Other

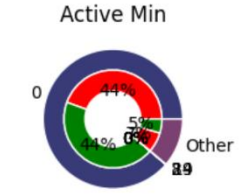

act_data_pkt_fwd
0
32%
33%
Other
1
2
3

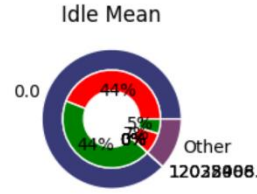

min_seg_size_forward
32
31%
28%
Other
40
18%
18%
20

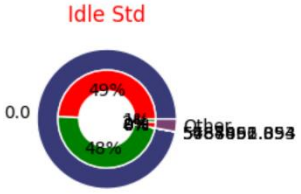

Active Mean
0.0
44%
5%
44%
Other

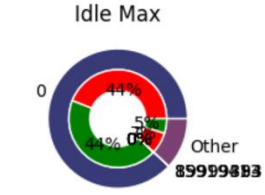

Active Std
0.0
50%
48%
Other

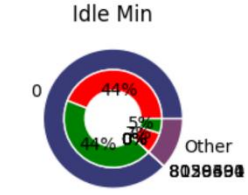

Active Max
0
44%
5%
44%
Other

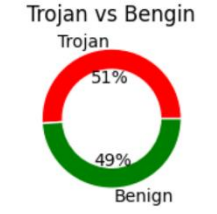

Active Min
0
44%
5%
44%
Other

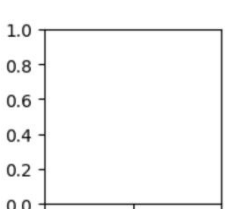

Idle Mean
0.0
44%
5%
44%
Other


Idle Std
0.0
49%
48%
Other


Idle Max
0
44%
5%
44%
Other


Idle Min
0
44%
5%
44%
Other


Trojan vs Bengin
Trojan
51%
49%
Benign


1.0
0.8
0.6
0.4
0.2
0.0

From the above graph we decided to exclude the following for having nearly constant values equally distributed between the 2 classes and we considered the distribution of the constant values because some classifiers like oneR depends on the frequency of a class per a feature:

1. Bwd PSH Flags
2. Fwd URG Flags
3. Bwd URG Flags
4. FIN Flag Count
5. RST Flag Count
6. CWE Flag Count
7. ECE Flag Count
8. Fwd Avg Bytes/Bulk
9. Fwd Avg Packets/Bulk
10. Fwd Avg Bulk Rate
11. Bwd Avg Bytes/Bulk
12. Bwd Avg Packets/Bulk
13. Bwd Avg Bulk Rate
14. Active Std
15. Idle Std

Note that the distribution of the 2 classes (Trojan and Benign) overall the dataset is nearly equal (51:49)

Conversion of String objects into int

We only have 2 object columns in our dataset which are Source IP and destination IP which are in the format 'A.B.C.D'. We dealt with these columns as labels and since using one hot encoder variables to represent such a column will result in a huge vector space so we use OrdinalEncoder which gives a label for each IP where for example an address of '10.4.255.1' will be substituted with 0 for example. Label Encoder which is used with training is used with training data and new IP addresses in test data which is wasn't seen before will be given a label -1. Classes that has the value Trojan is converted to 1 and Benign is converted to 0.

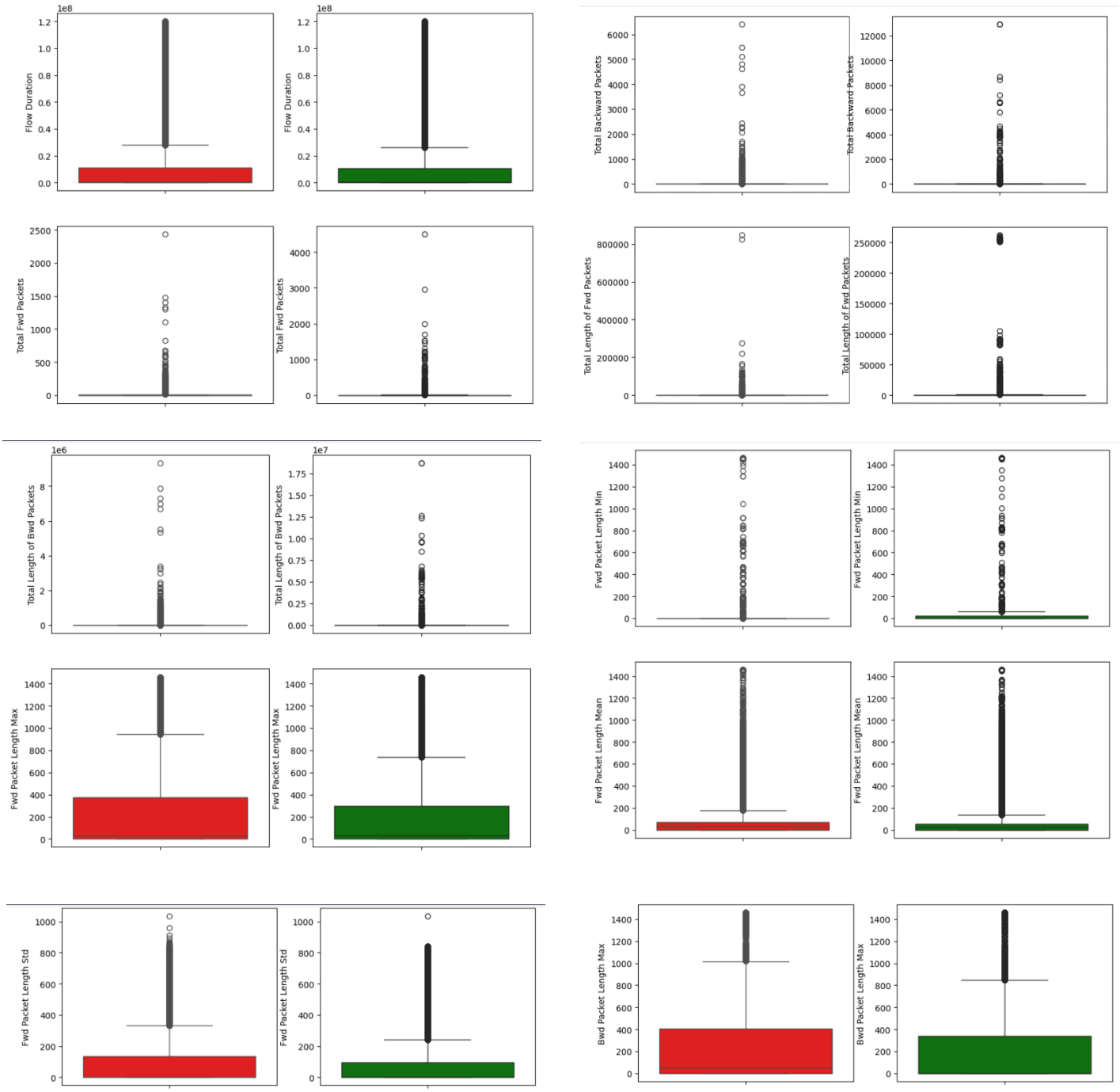
	Source IP	Source Port	Destination IP	Destination Port	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	...	Init_Win_bytes_forward	Init_Win_bytes_backward	act_data_pkt_fw
0	7.00	48983	353.00	80	6	290032	1	1	0	0	...	1386	137	
1	7.00	34014	813.00	443	6	60631432	88	98	4826	127271	...	65535	439	
2	7.00	1976	7.00	53	17	1172343	1	1	33	153	...	-1	-1	
3	7.00	60140	1276.00	80	6	30484454	2	2	0	106	...	65535	115	
4	6.00	57976	1417.00	443	6	14443064	9	7	504	4755	...	65535	349	

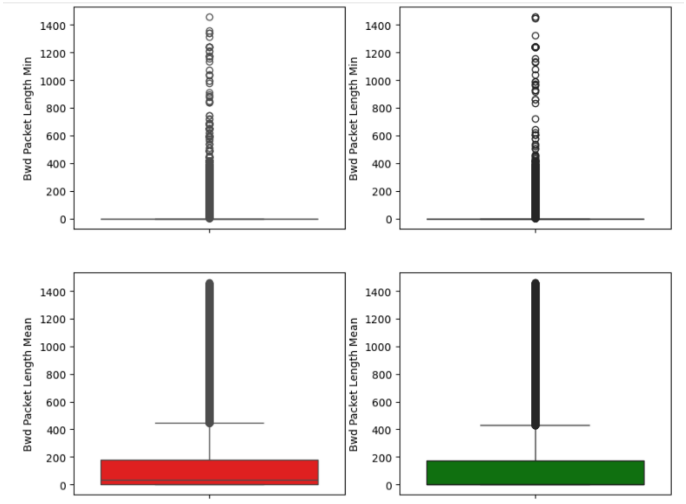
Outlier Detection

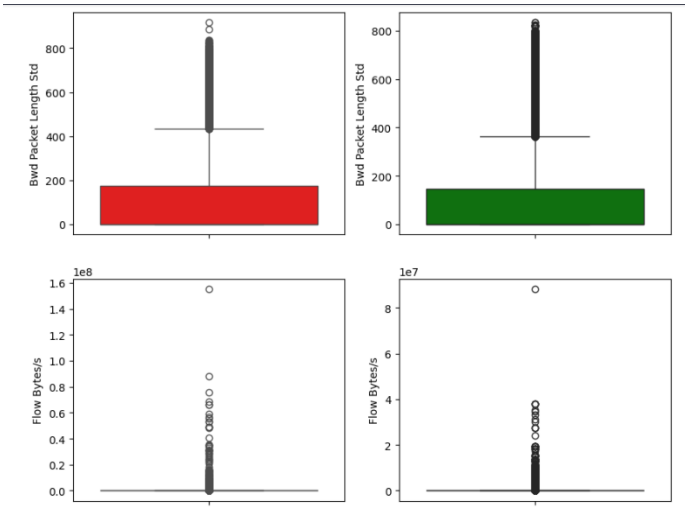
We used a voting system where we used around 5 approach to vote for each point to be an outlier or and at the end we consider the point to be an outlier if more than 2 approaches voted for that point to be an outlier. But there are some feature like 'Protocol', 'Source IP', 'Source Port', 'Destination IP', 'Destination Port' where application of outliers on such features doesn't have a meaning where outliers of such features can be indicated for Trojan points.

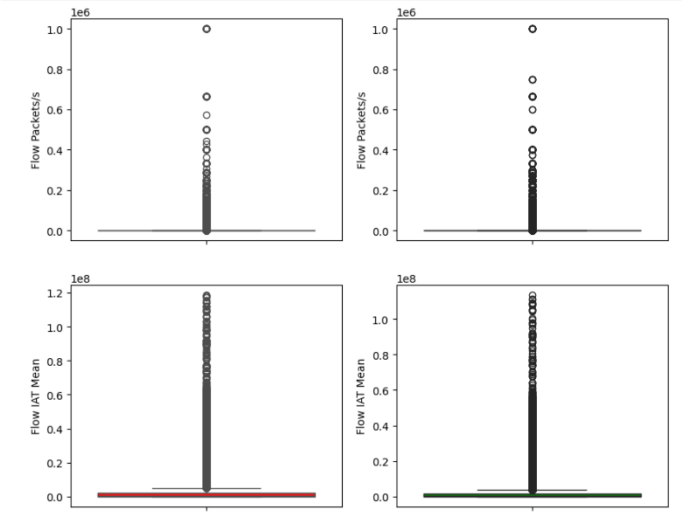
1st Approach

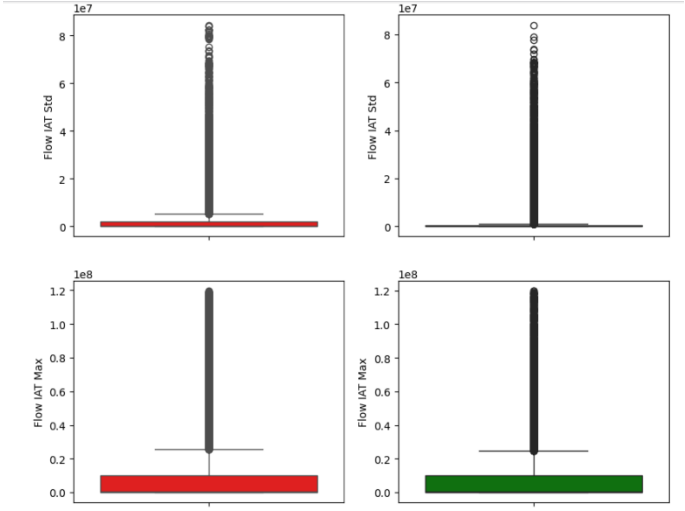
We used IQR (interquartile range) to get the outlier where it uses Boxplot and plot Q1, Q2, Q3 for each range per feature per class. Where each point is considered outlier if its feature value is less than (Q1 – IQR) or greater than (Q3 + IQR) where equation of IQR is (Q3 – Q1). (notice how many graph have Q1, Q2 and Q3 nearly same constant value)

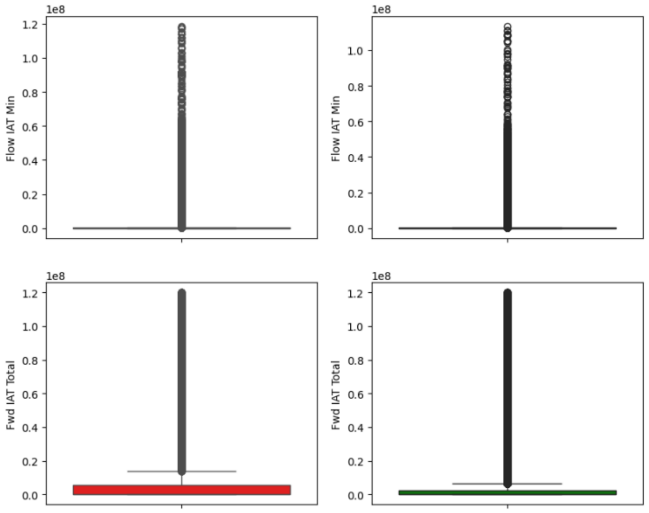


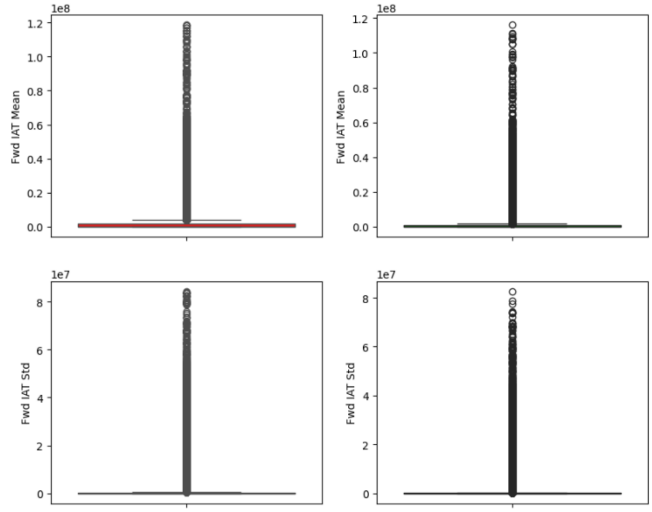

Bwd Packet Length Min
Bwd Packet Length Min
Bwd Packet Length Mean
Bwd Packet Length Mean

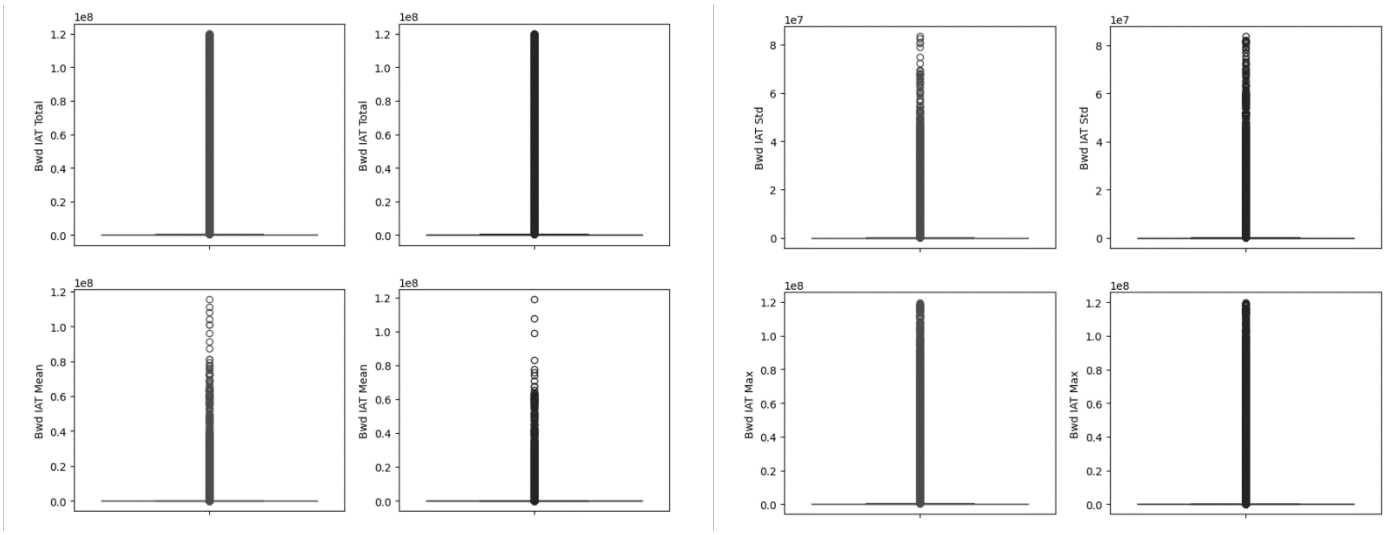

Bwd Packet Length Std
Bwd Packet Length Std
Flow Bytes/s
Flow Bytes/s

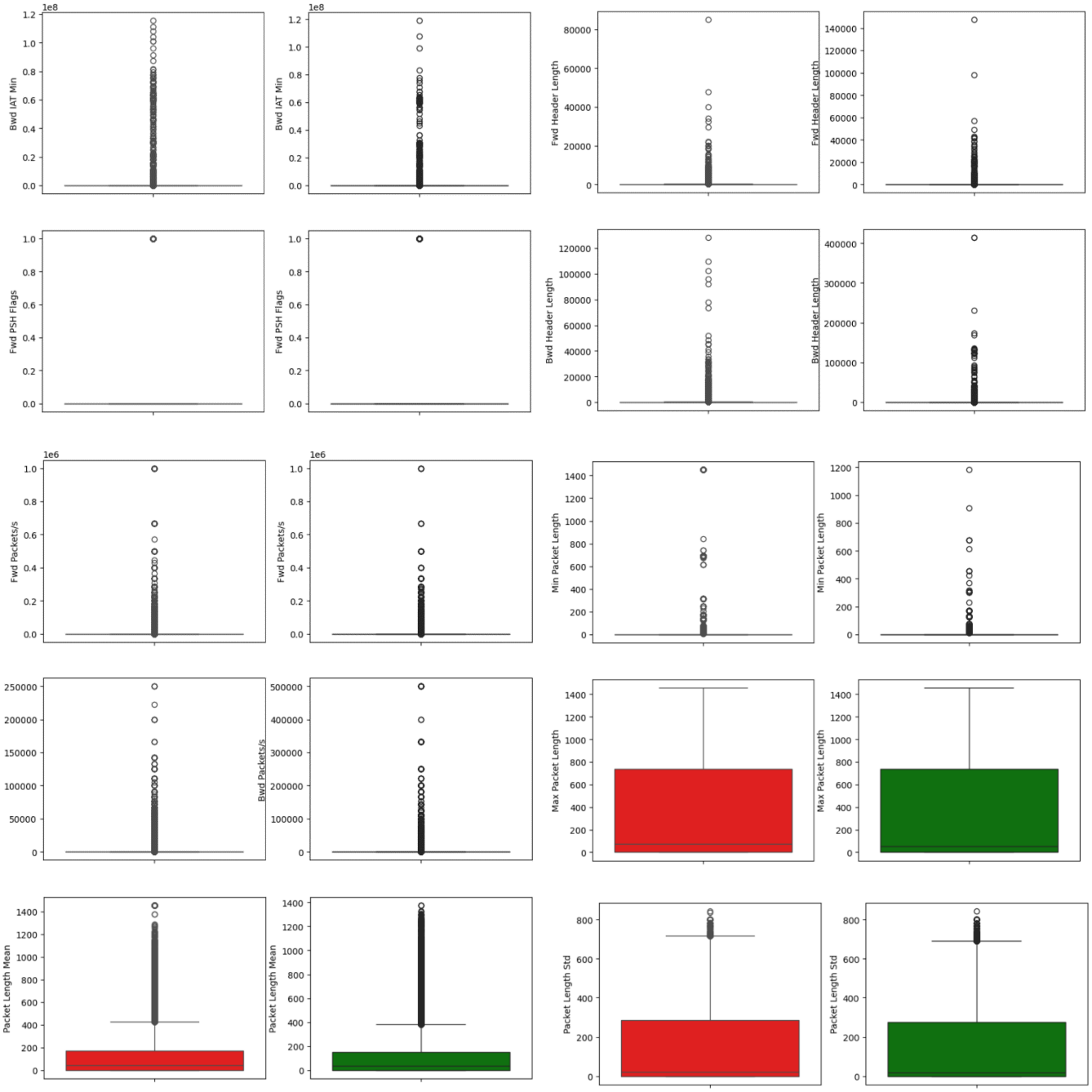

Flow Packets/s
Flow Packets/s
Flow IAT Mean
Flow IAT Mean

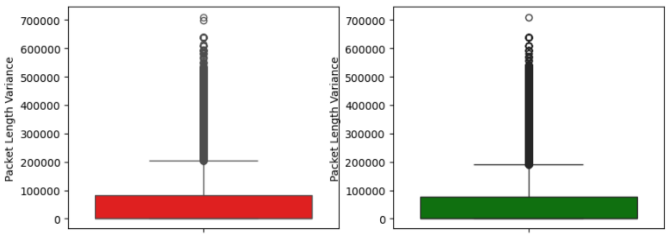

Flow IAT Std
Flow IAT Std
Flow IAT Max
Flow IAT Max

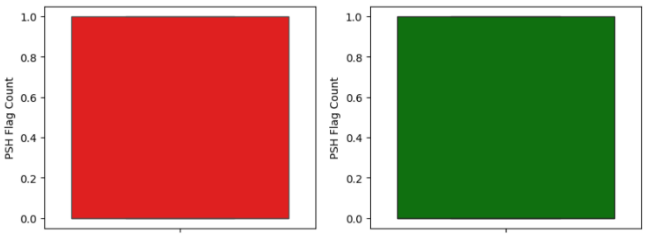

Flow IAT Min
Flow IAT Min
Fwd IAT Total
Fwd IAT Total
Fwd IAT Max
Fwd IAT Max

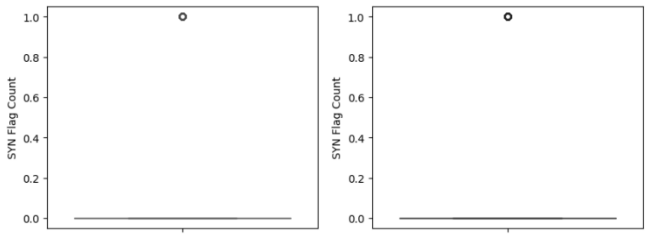

Fwd IAT Mean
Fwd IAT Mean
Fwd IAT Std
Fwd IAT Std
Fwd IAT Min
Fwd IAT Min

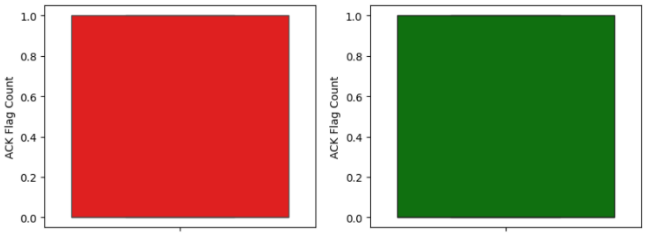

Bwd IAT Total
Bwd IAT Total
Bwd IAT Std
Bwd IAT Std
Bwd IAT Mean
Bwd IAT Mean
Bwd IAT Max
Bwd IAT Max

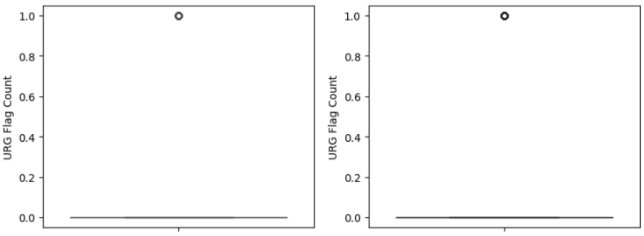

Bwd IAT Min
Bwd IAT Min
Fwd Header Length
Fwd Header Length
Fwd PSH Flags
Fwd PSH Flags
Bwd Header Length
Bwd Header Length
Fwd Packets/s
Fwd Packets/s
Min Packet Length
Min Packet Length
Bwd Packets/s
Bwd Packets/s
Max Packet Length
Max Packet Length
Packet Length Mean
Packet Length Mean
Packet Length Std
Packet Length Std

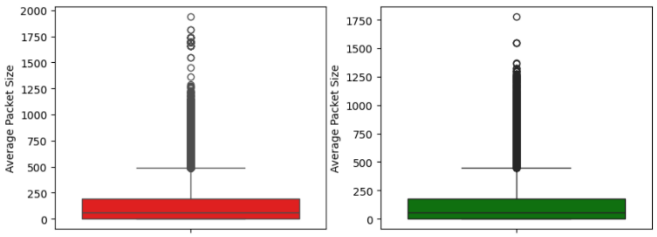

Packet Length Variance
Packet Length Variance

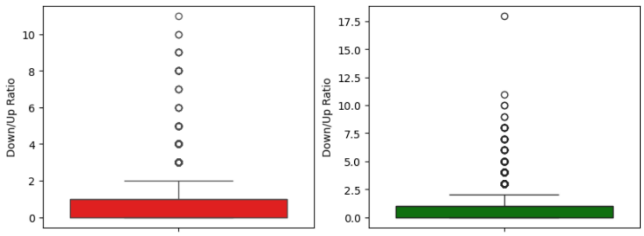

PSH Flag Count
PSH Flag Count

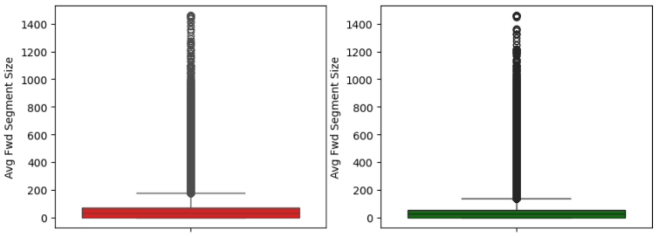

SYN Flag Count
SYN Flag Count

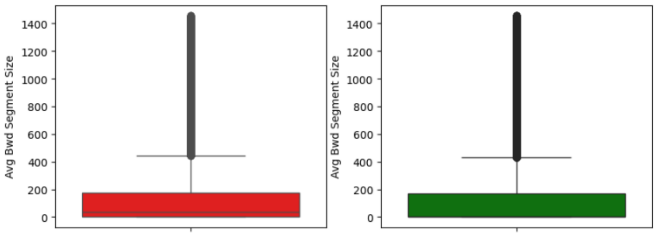

ACK Flag Count
ACK Flag Count

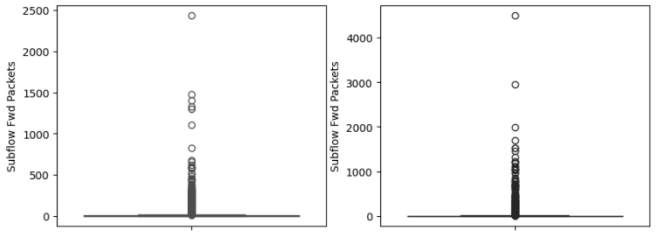

URG Flag Count
URG Flag Count

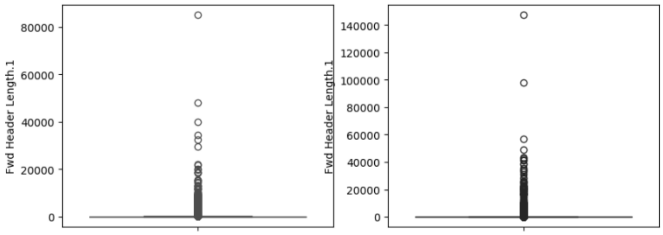

Average Packet Size
Average Packet Size

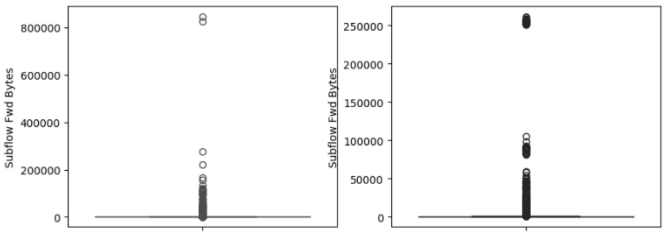

Down/Up Ratio
Down/Up Ratio

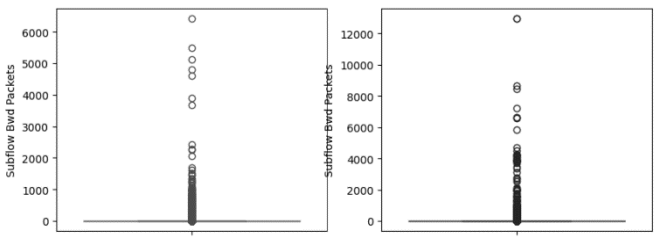

Avg Fwd Segment Size
Avg Fwd Segment Size

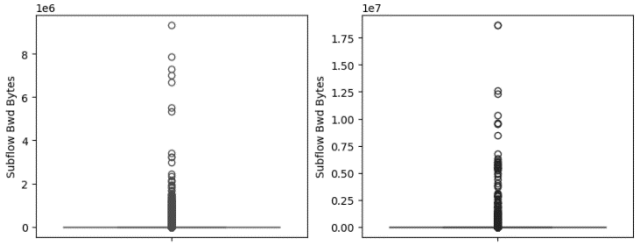

Avg Bwd Segment Size
Avg Bwd Segment Size

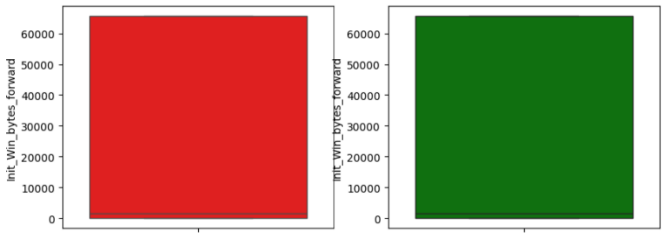

Subflow Fwd Packets
Subflow Fwd Packets

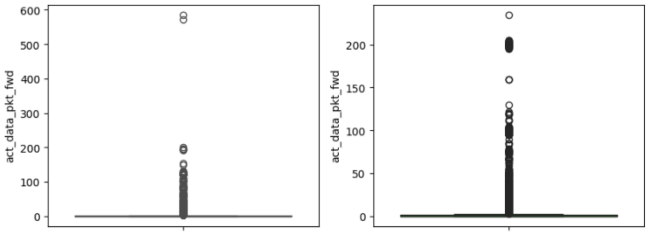

Fwd Header Length.1
Fwd Header Length.1

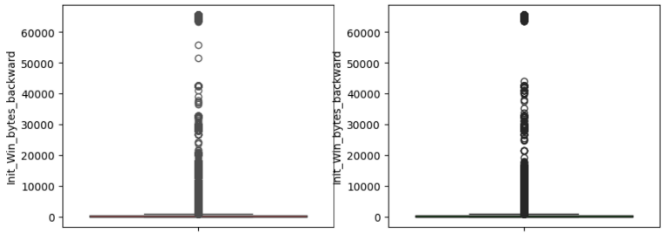

Subflow Fwd Bytes
Subflow Fwd Bytes

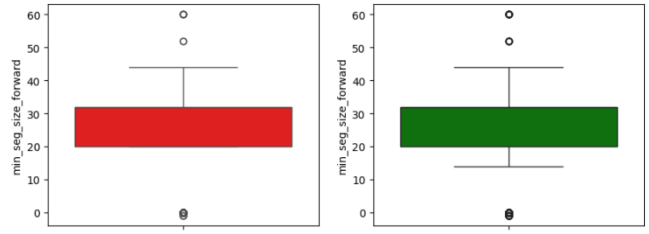

Subflow Bwd Packets
Subflow Bwd Packets

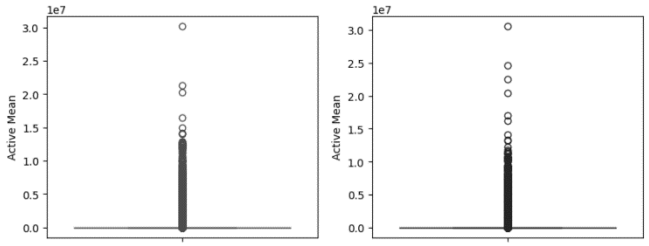

1e6
Subflow Bwd Bytes
1e7
Subflow Bwd Bytes

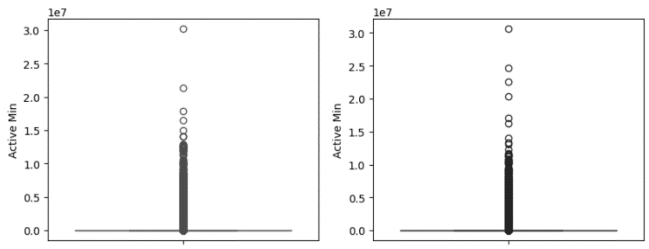

Init_Win_bytes_forward
Init_Win_bytes_forward

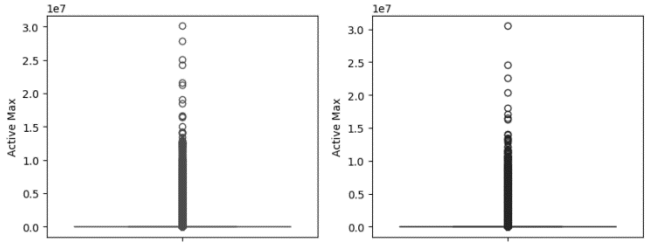

act_data_pkt_fwd
act_data_pkt_fwd

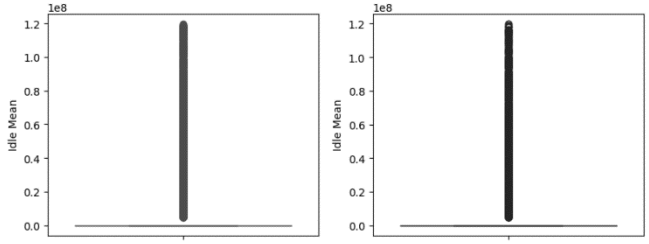

Init_Win_bytes_backward
Init_Win_bytes_backward

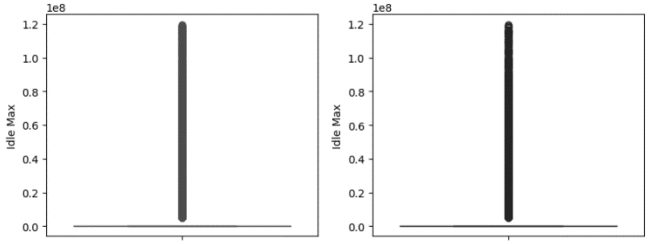

min_seg_size_forward
min_seg_size_forward

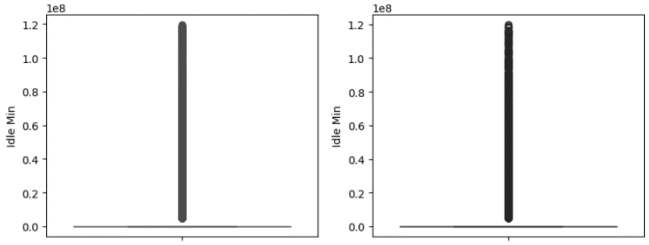

1e7
Active Mean
1e7
Active Mean


1e7
Active Min
1e7
Active Min


1e7
Active Max
1e7
Active Max

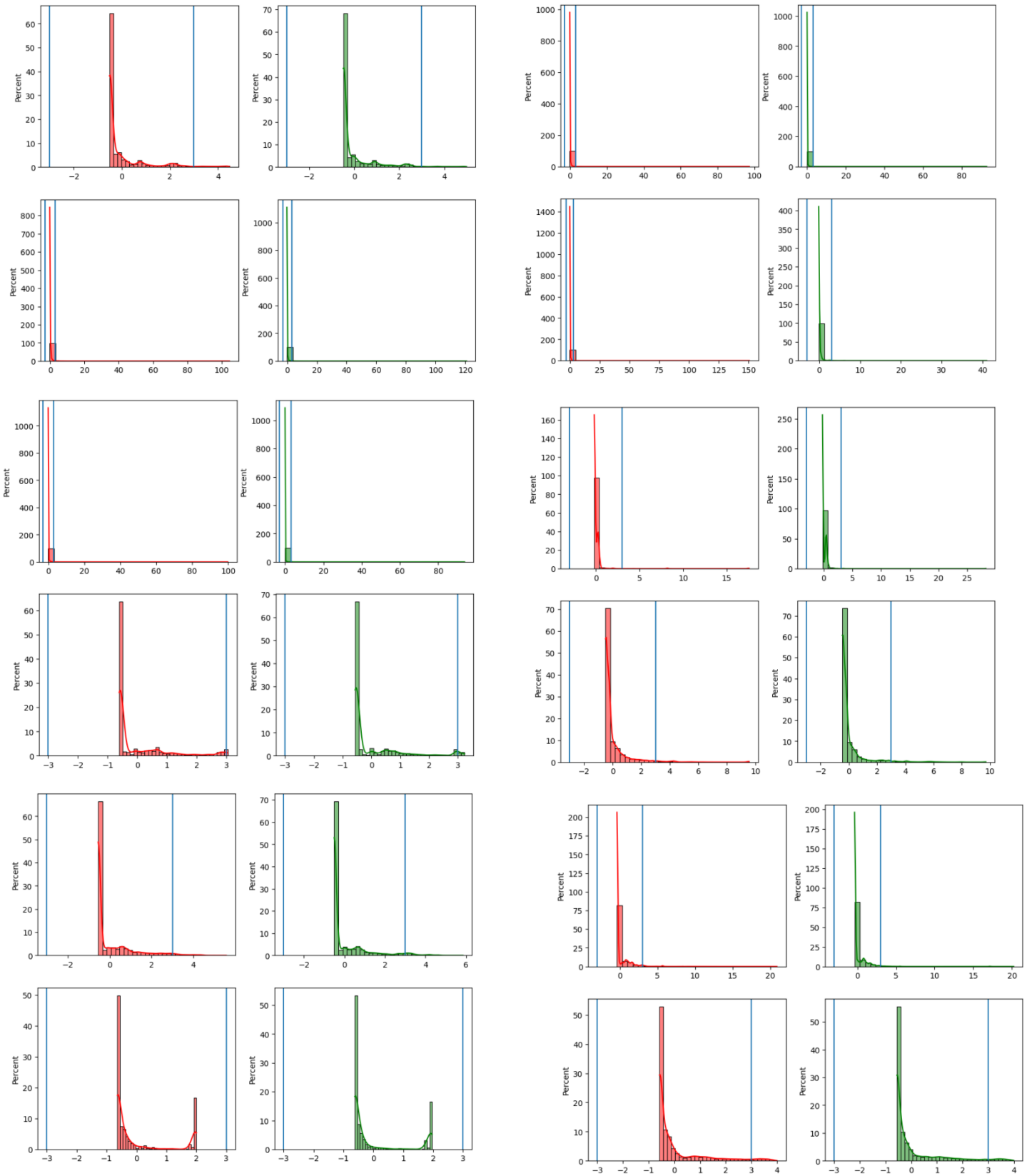

1e8
Idle Mean
1e8
Idle Mean

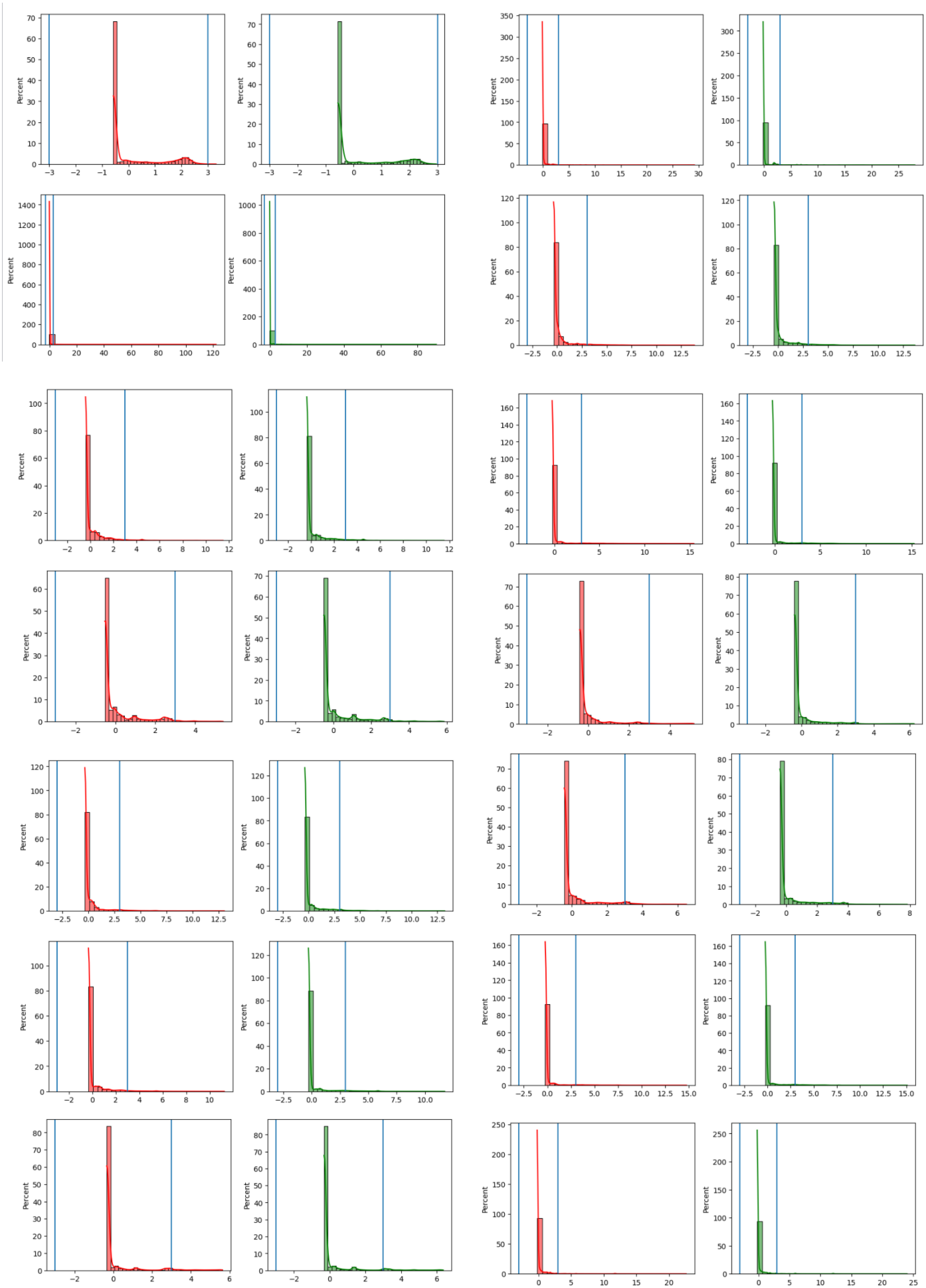

1e8
Idle Max
1e8
Idle Max

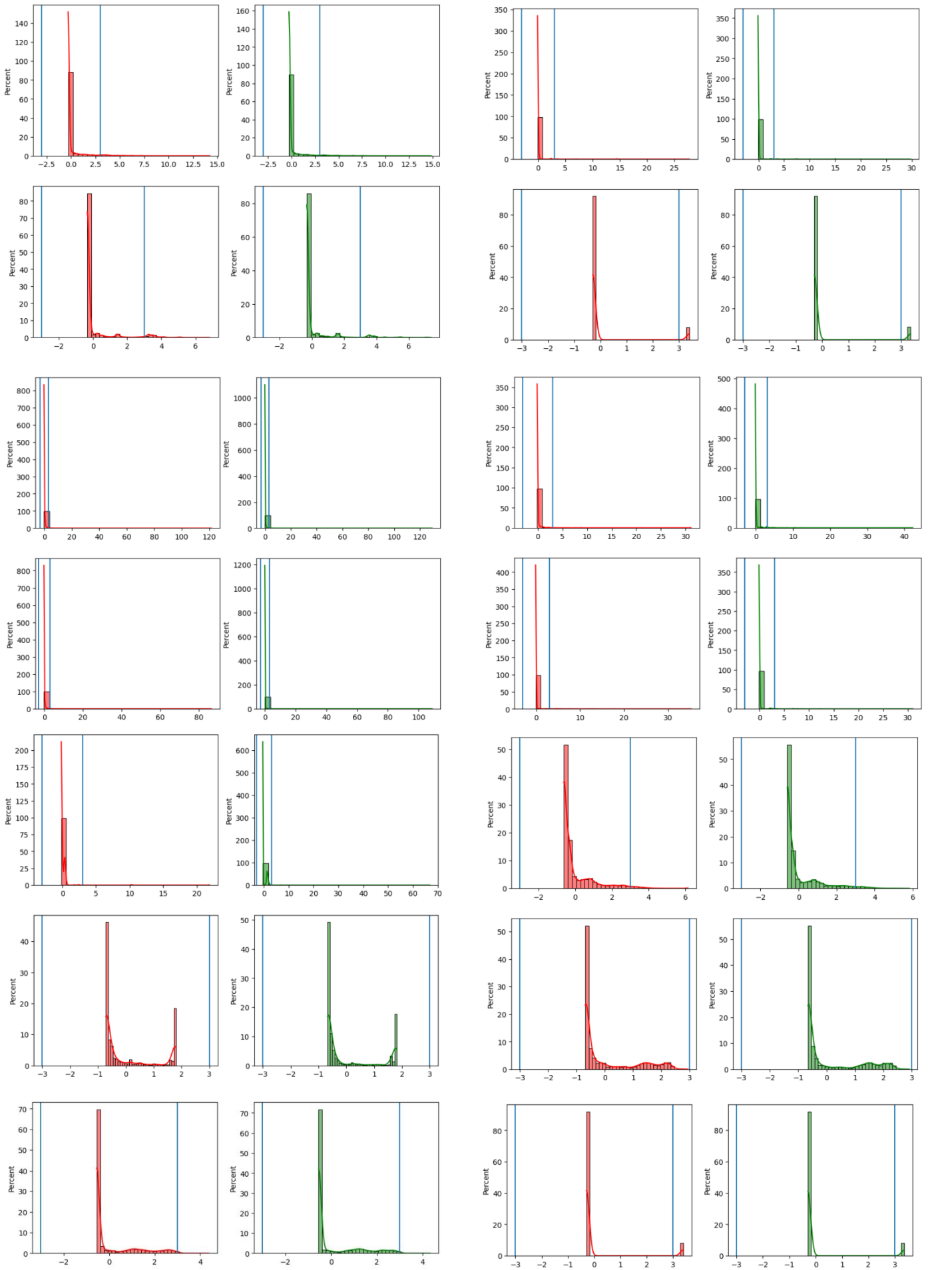

1e8
Idle Min
1e8
Idle Min

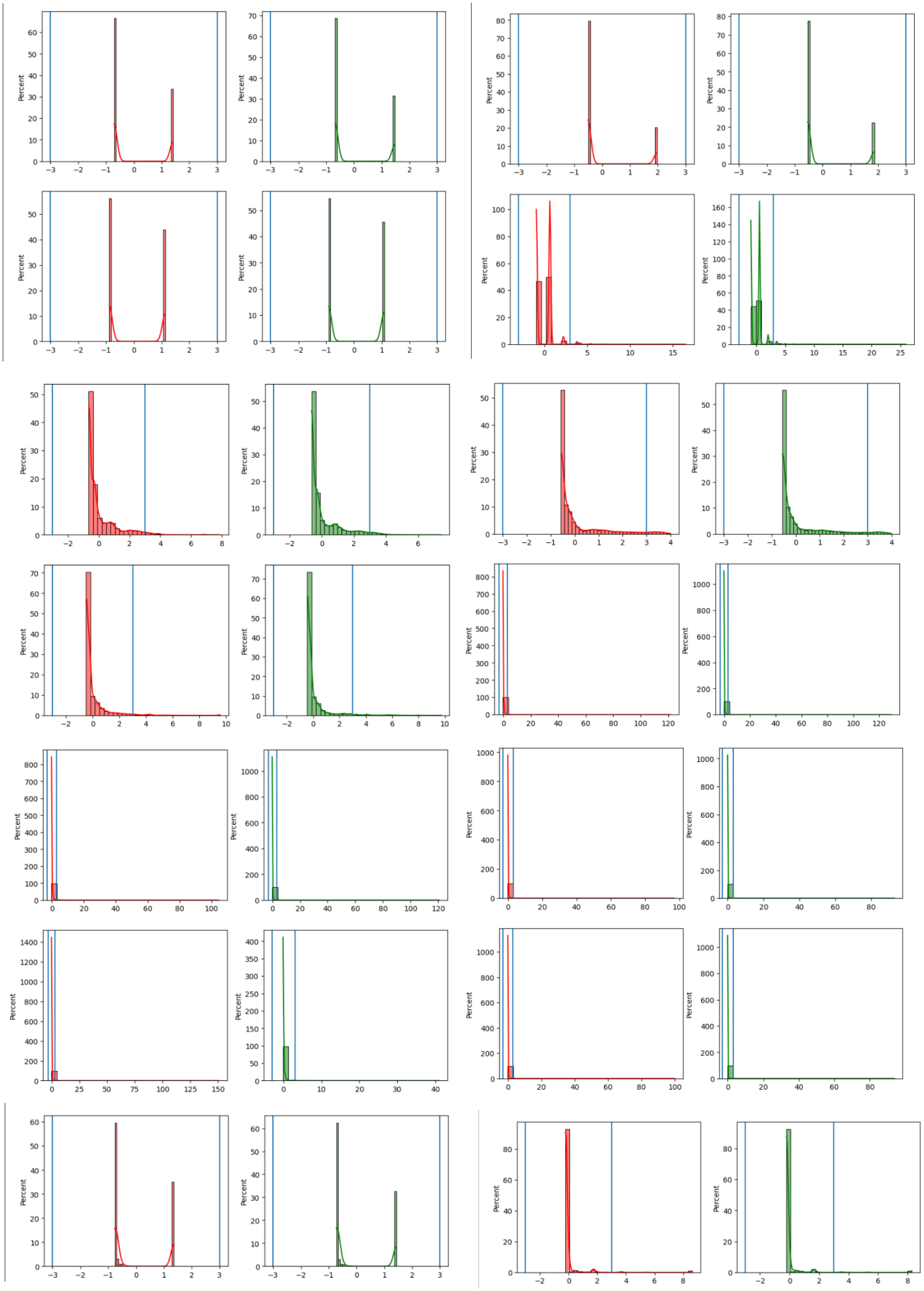
2nd Approach

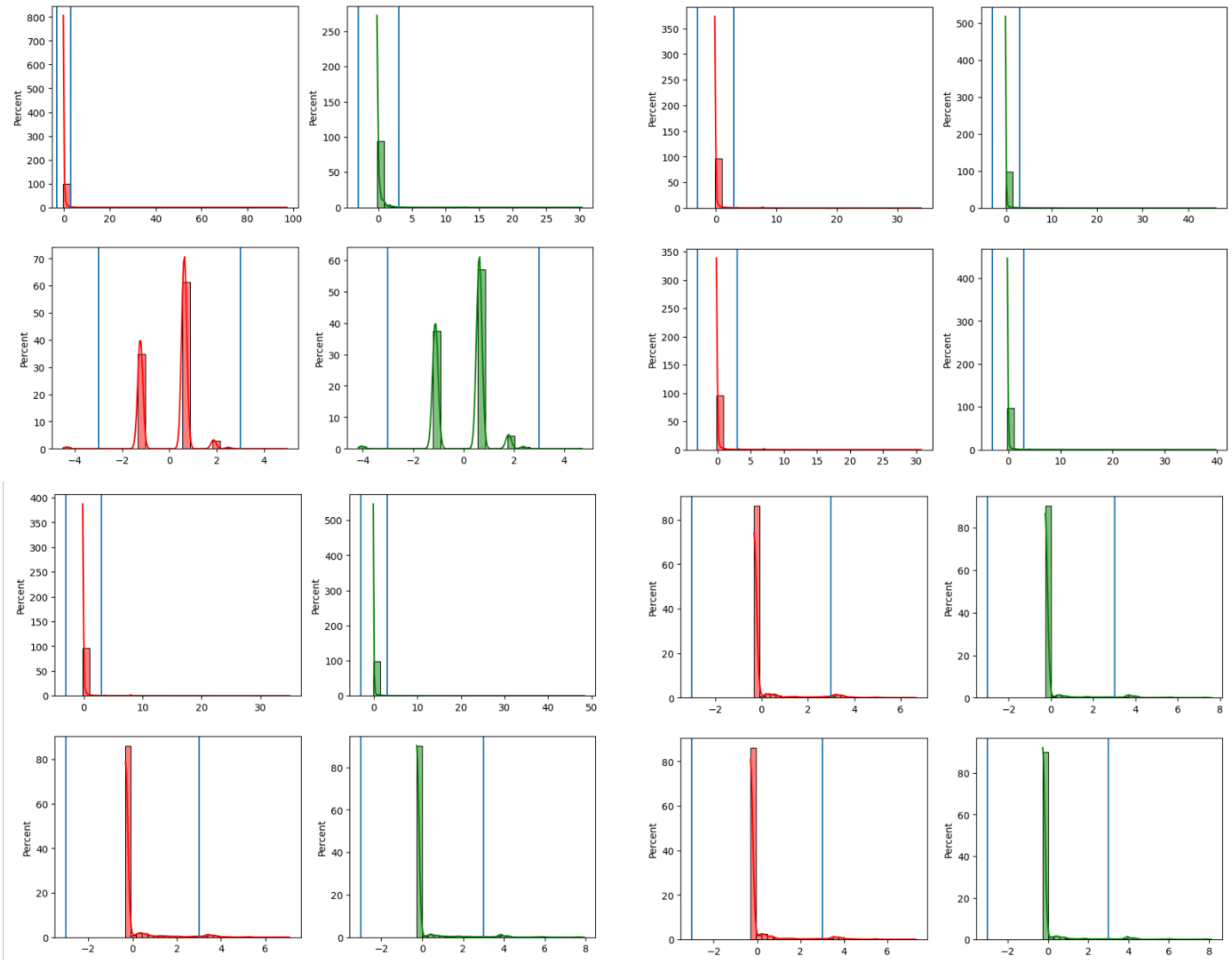
We used Z-score where we standardized the features and assumed they follow normal distribution and every value that's greater than 3 or less than -3 is considered an outlier.




Percent


Percent

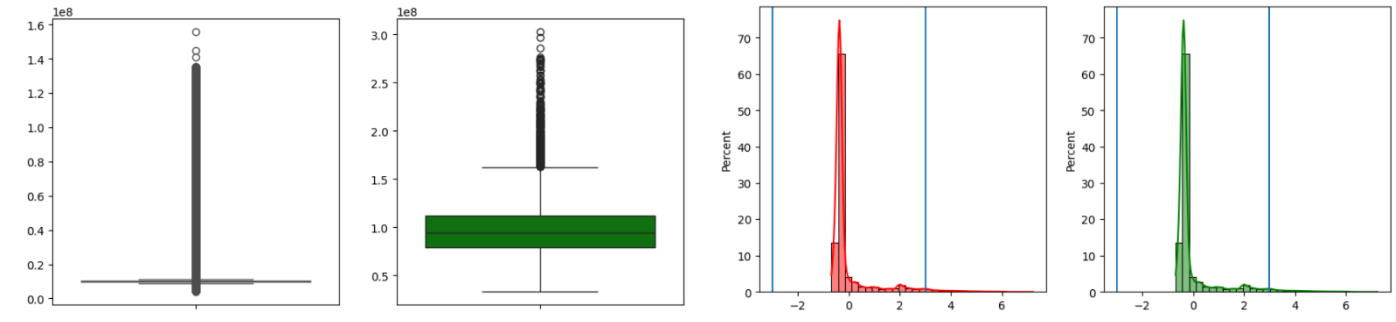

Percent



3rd Approach

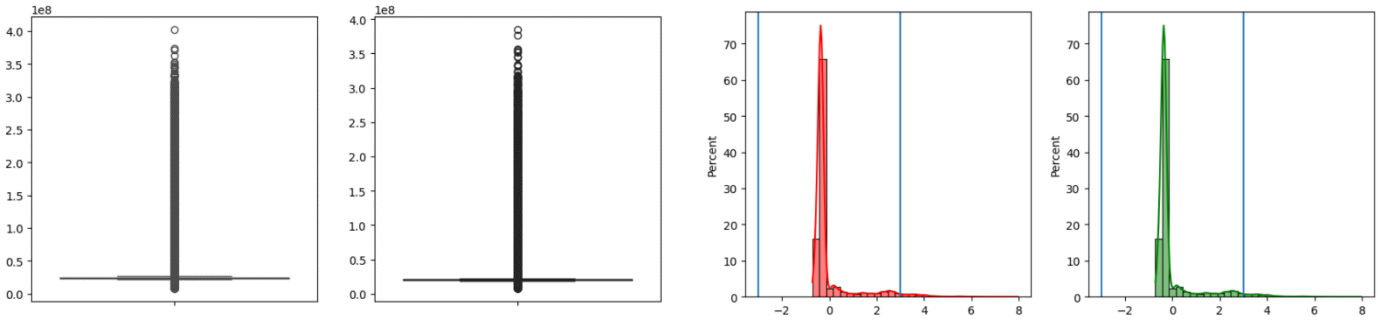
We applied Kmeans to the our data with K=2 and then applied approach number 1&2 on the distance from each point to the center of its cluster.

We found that number of points in 1st cluster = 120876 and number of points in 2nd cluster = 12235 where percentage of cluster 2 to cluster 1 = 0.1.



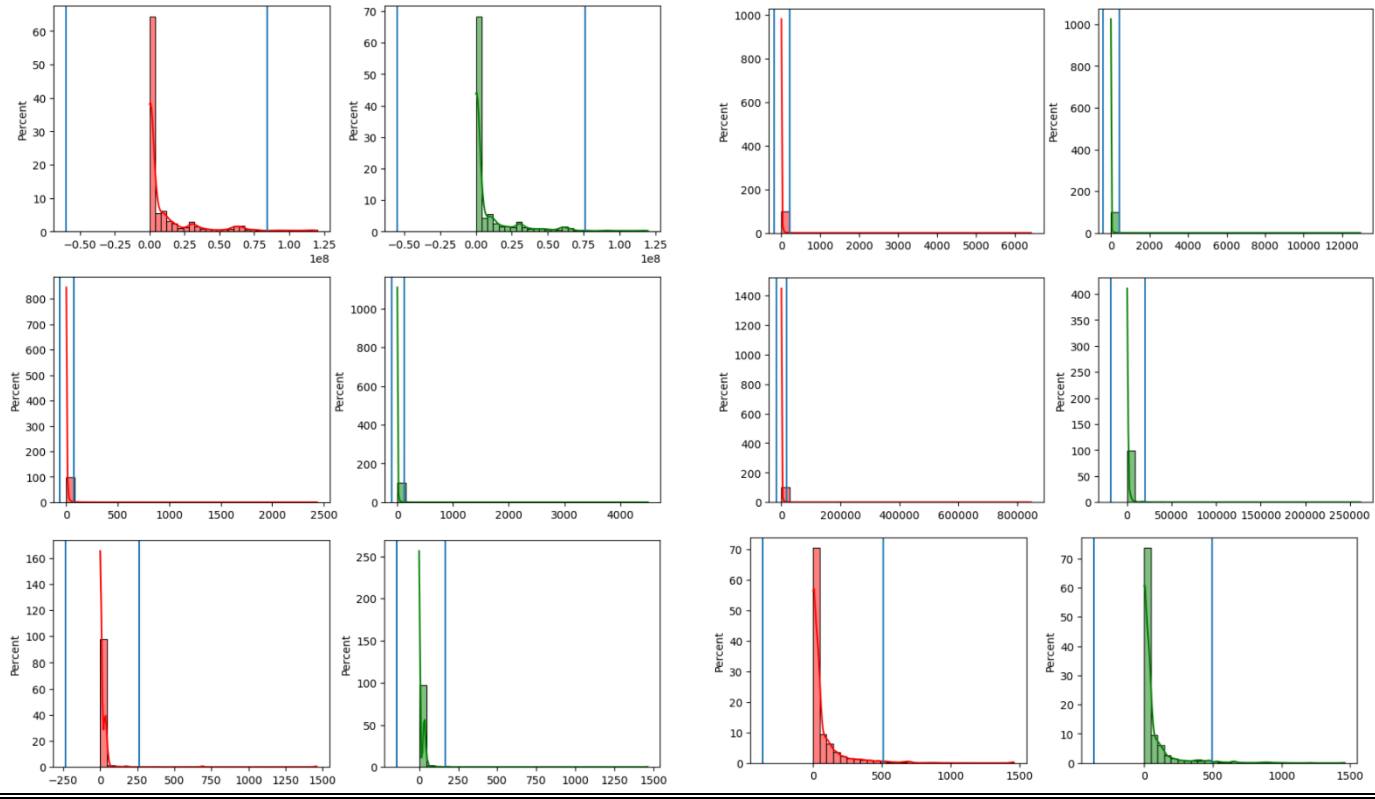
4th Approach

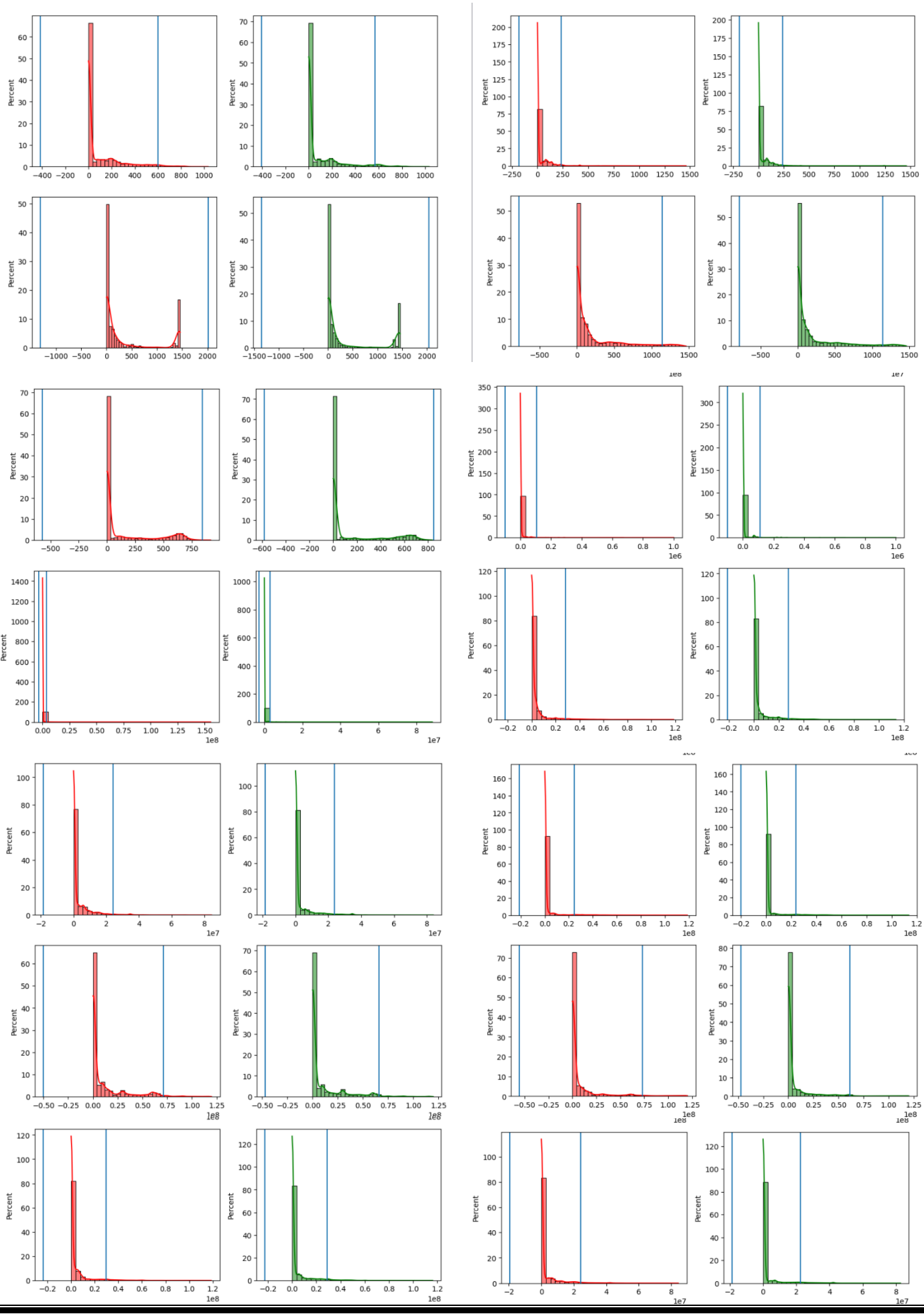
We applied 3rd approach but instead of using Kmeans we considered Trojan points as a cluster and Benign points as another cluster. Then applied approach number 1&2 on the distance from each point to the center of its cluster where number of points in Trojan cluster = 68012 and number of points in Benign cluster = 65099.

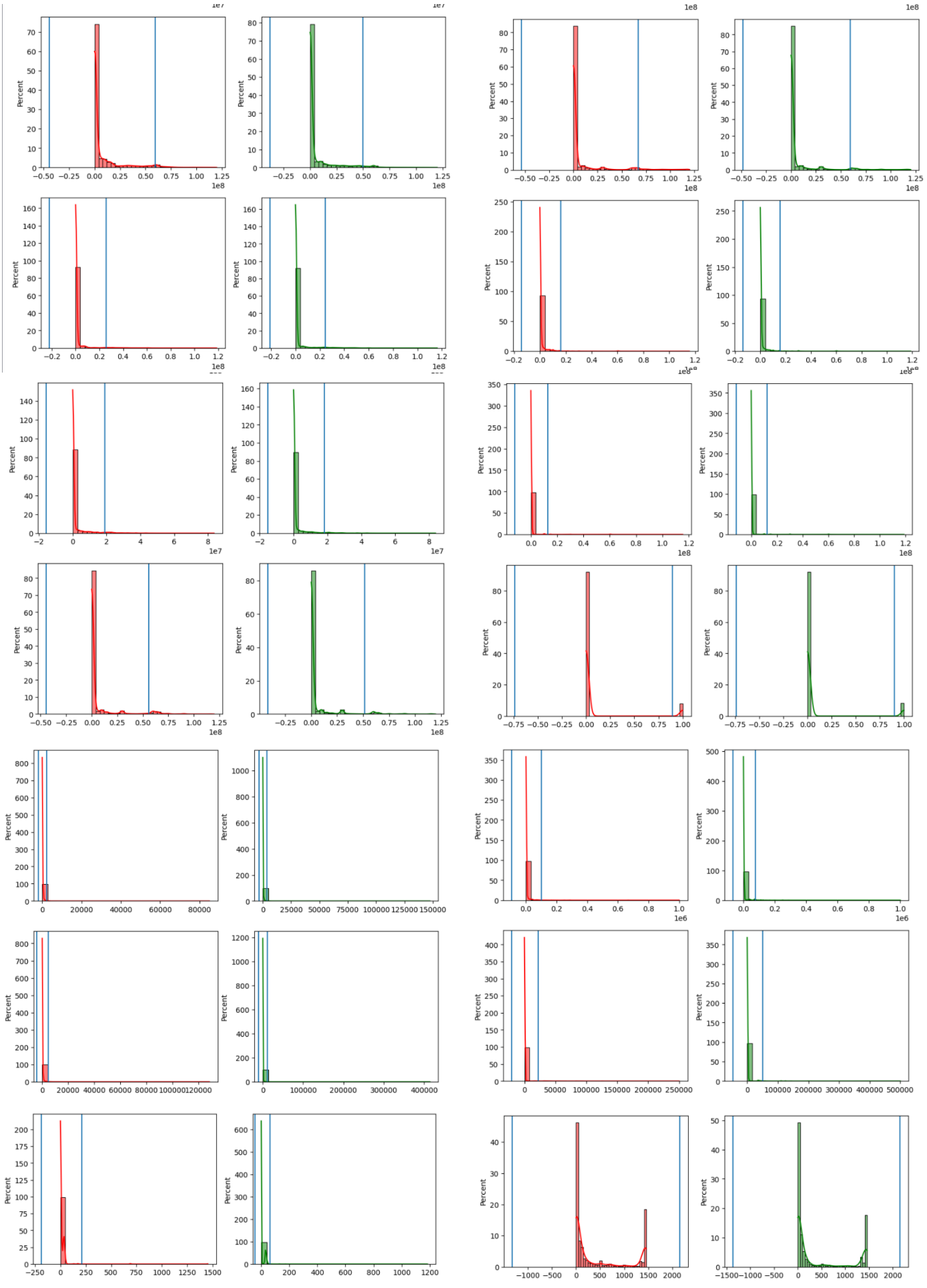


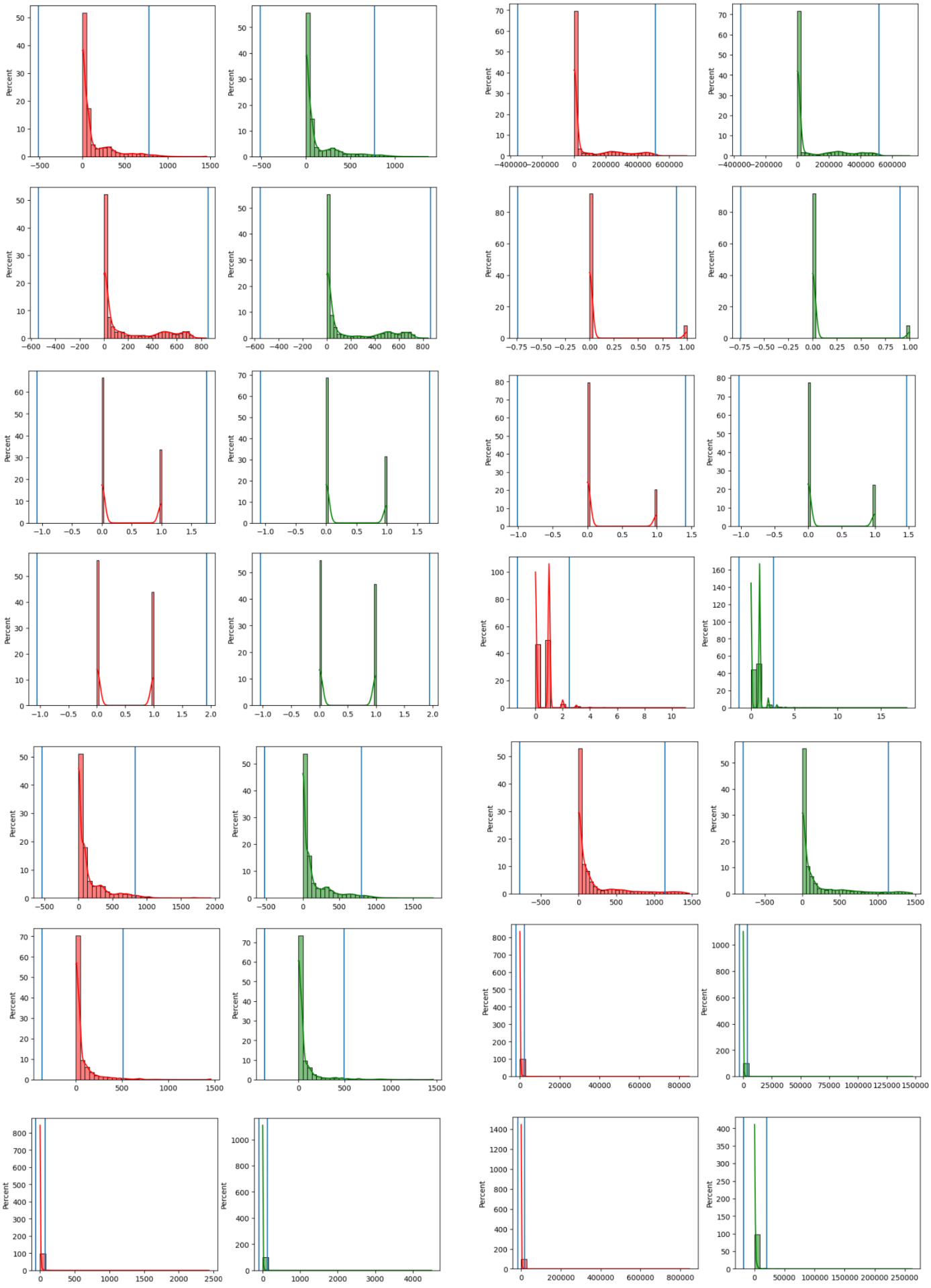
5th Approach

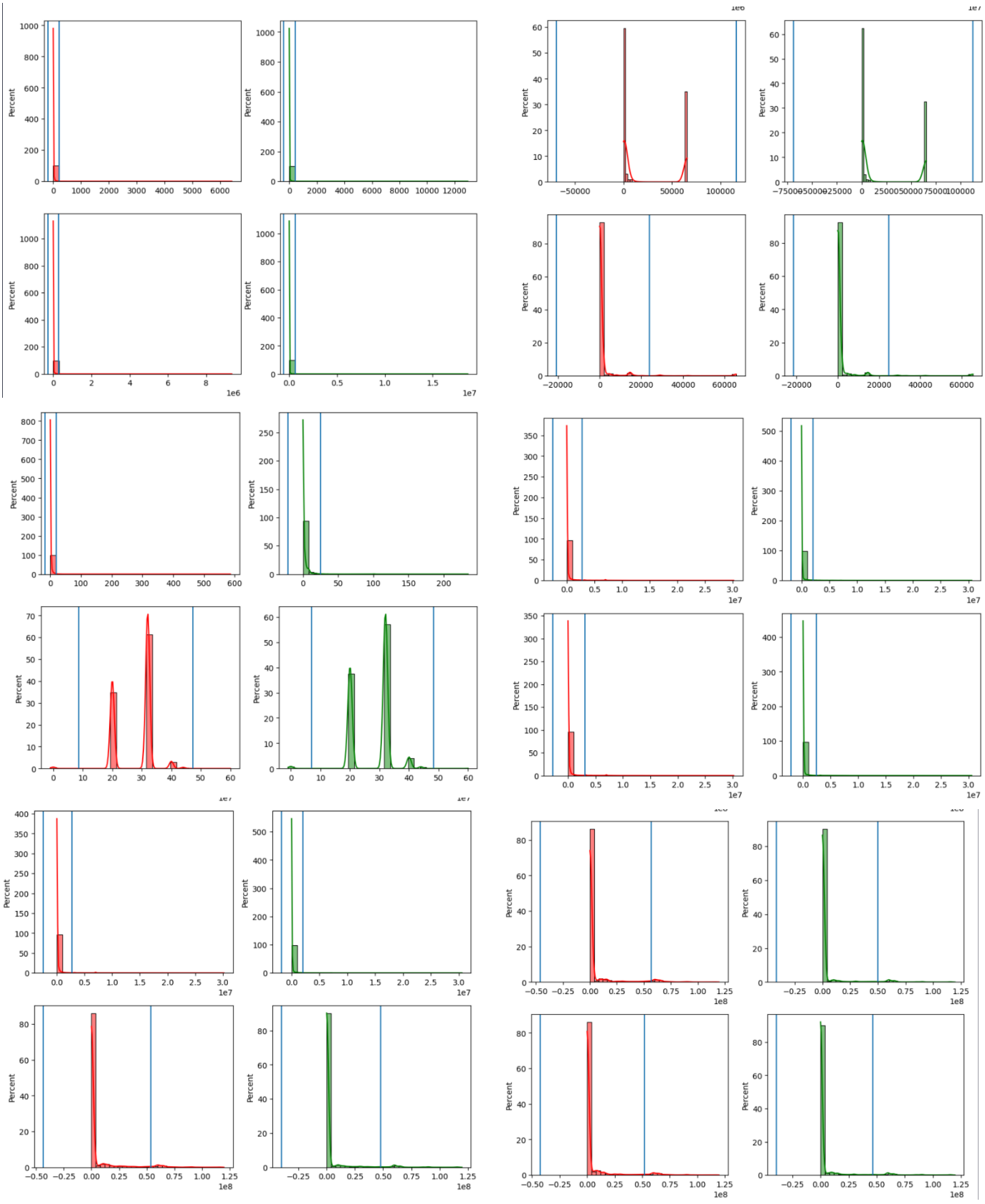
For every feature per class, we get its mean and standard deviation. Where any point is greater than mean() + std() or less than mean() – std() will be considered an outlier.




Percent

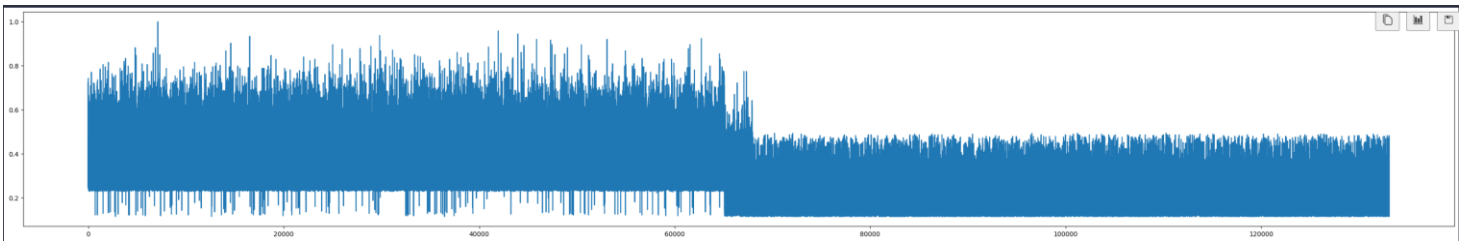

Percent
1e7
1e8
1e6


Percent


Percent
1e6
1e7
1e8

Voting Result

After using the above approaches, the voting graph probability of points is as follows:

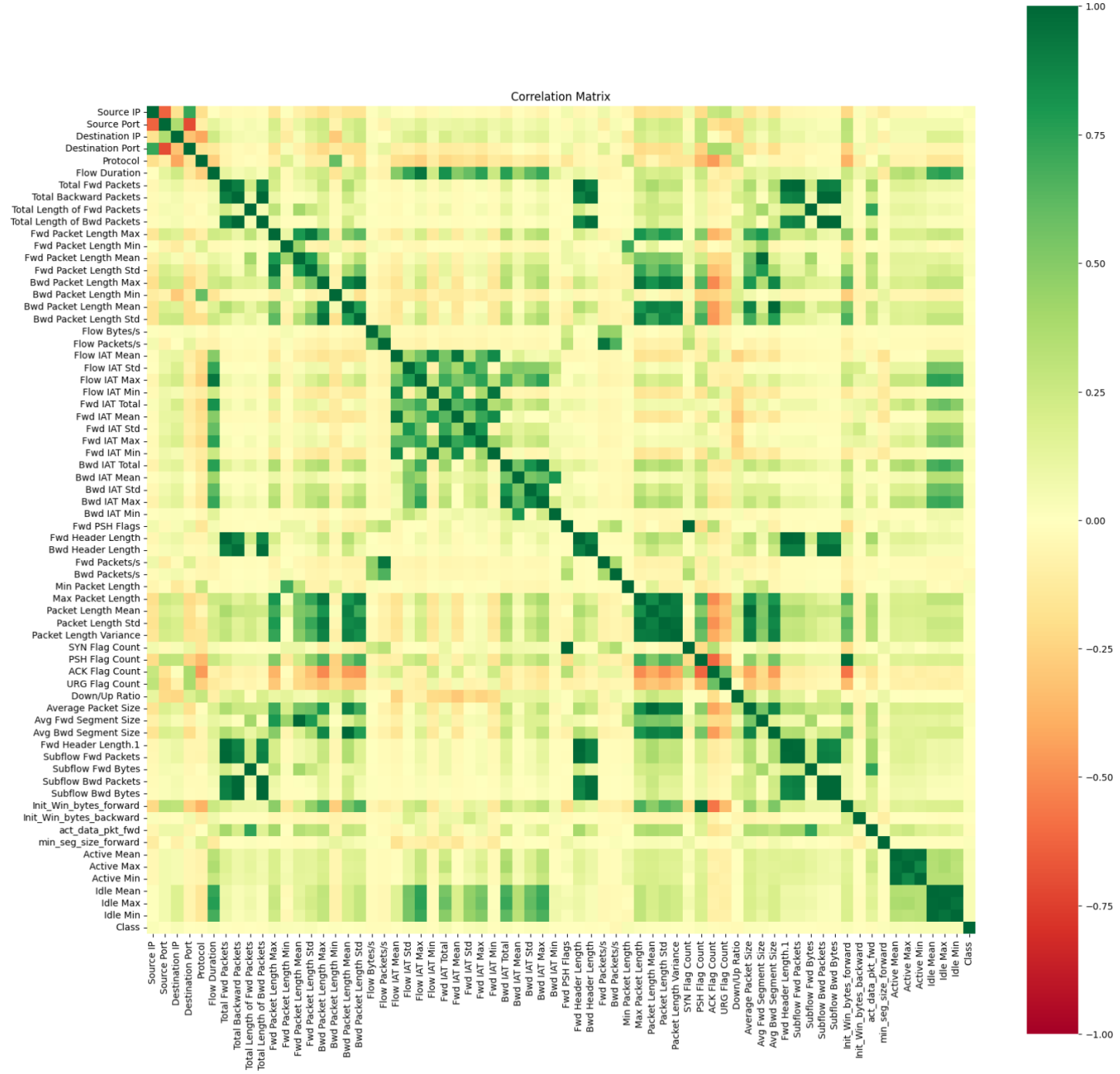


Where we put a threshold around 0.6 that any point has a voting greater than or equal to this threshold will be considered an outlier.

thus the number of points to be removed = 3818 and it's proportion of the training data = 2.86%.

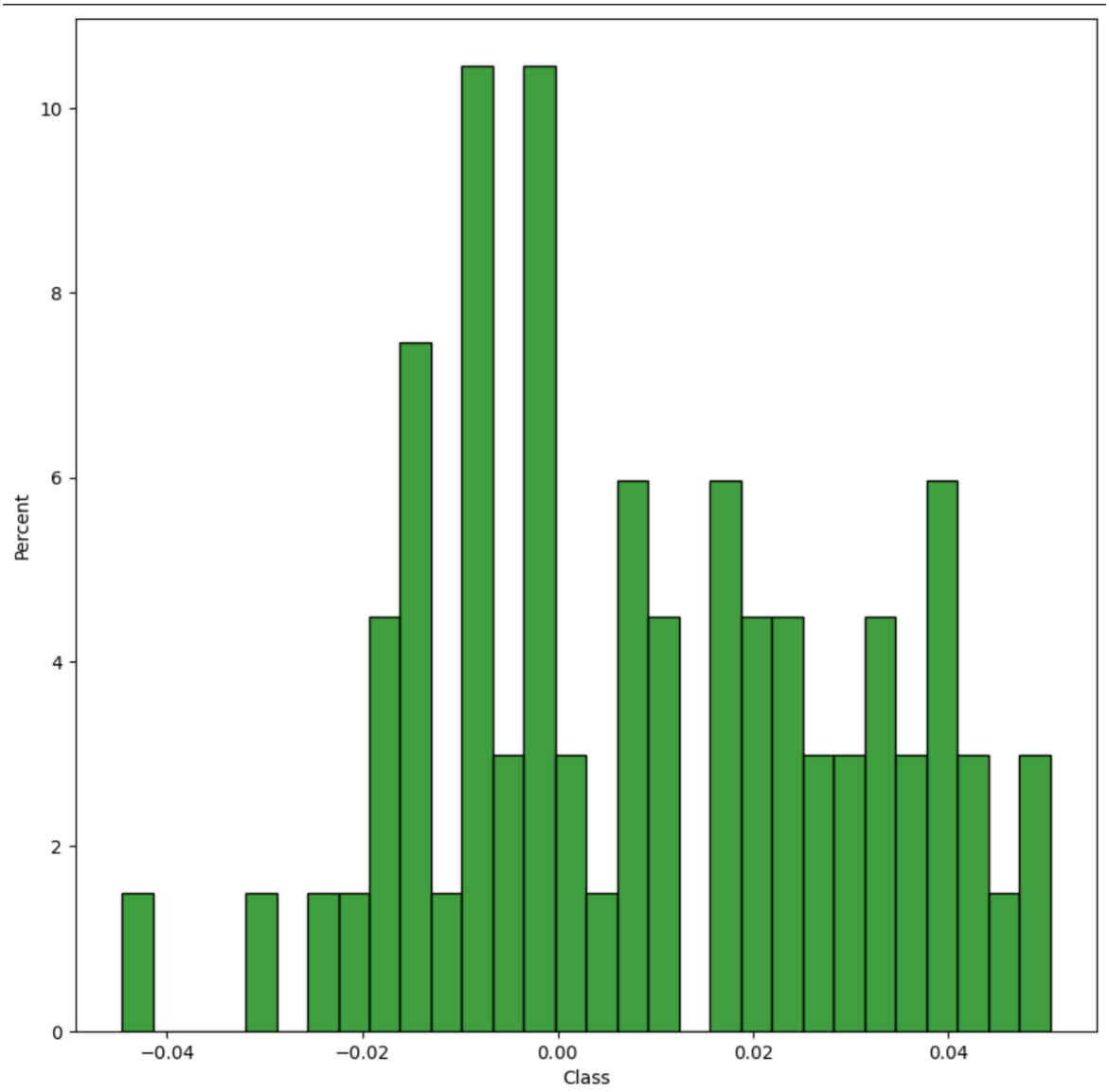
Feature Correlation

We applied feature correlation to discover that there are many features that are correlated to each other so many features are redundant but more importantly we found that all of these features have a corresponding correlation to the class around 0. Which indicates that these features aren't good estimate for this problem.



Correlated Feature to the Class

We computed correlation of each feature to the class then we only kept features that absolute correlation of more than 0.02:

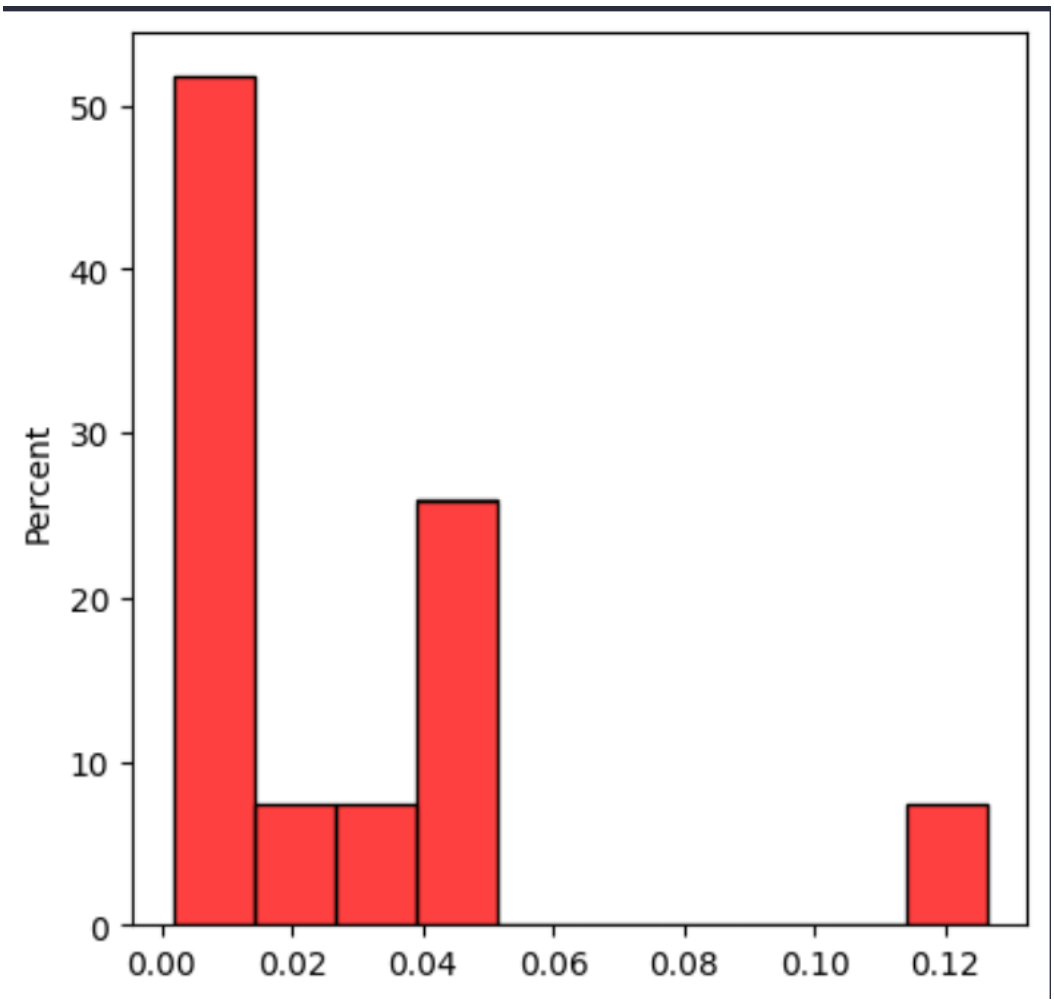


The resultant features we kept have the following correlation with the class (we only kept 27 features from that step):

```
...   Number of features that are not zero correlated to the class are 27
      Source IP                0.03
      Destination IP           0.04
      Flow Duration            0.04
      Fwd Packet Length Max    0.03
      Fwd Packet Length Min    0.02
      Fwd Packet Length Mean   0.02
      Fwd Packet Length Std    0.03
      Flow IAT Max             0.03
      Fwd IAT Total            0.05
      Fwd IAT Std              0.03
      Fwd IAT Max              0.05
      Bwd IAT Total            0.03
      Bwd IAT Std              0.02
      Bwd IAT Max              0.02
      Bwd Packets/s           -0.04
      Min Packet Length        0.04
      PSH Flag Count           0.02
      URG Flag Count          -0.02
      Down/Up Ratio           -0.03
      Avg Fwd Segment Size     0.02
      Init_Win_bytes_forward   0.03
      Active Mean              0.04
      Active Max               0.04
      Active Min               0.04
      Idle Mean                0.04
      Idle Max                 0.04
      Idle Min                 0.04
      Name: Class, dtype: float64
```

Important Features

We then computed information gain for every of the above 27 features kept only the features that have information gain higher than 0.02, so we are left with only 12 features.



And these 12 features are:

'Source IP', 'Destination IP', 'Flow Duration', 'Fwd Packet Length Max',

'Fwd Packet Length Mean', 'Fwd Packet Length Std', 'Flow IAT Max',

'Fwd IAT Total', 'Fwd IAT Max', 'Bwd Packets/s', 'Avg Fwd Segment Size',

'Init_Win_bytes_forward'

Feature Normalization

We computed the mean and standard deviation of the training data then normalized it and normalized the testing data using the mean and standard deviation of the training data. Encoding of IP addresses of the test data is encoded using the same encoding of the training data where unseen IP addresses will be given an ID of -1. Some features had a standard deviation of 0, so we added a small value to it to avoid divide by zero.

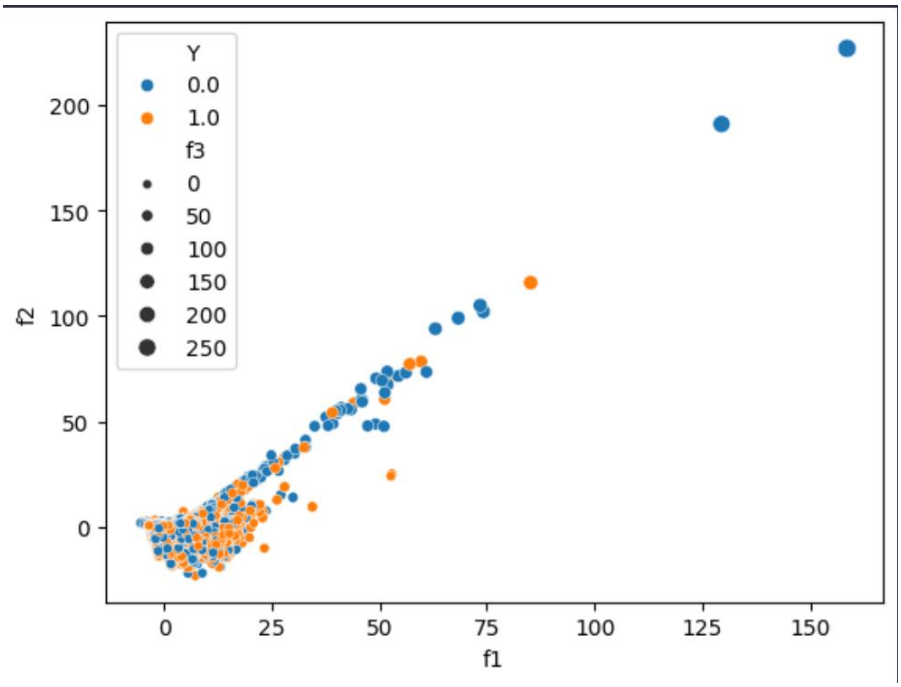
Data separability

from the above we can see that the interference between data points is very big that finding any separator that can split the classes of the points is very hard and we can visualize the data as follows:

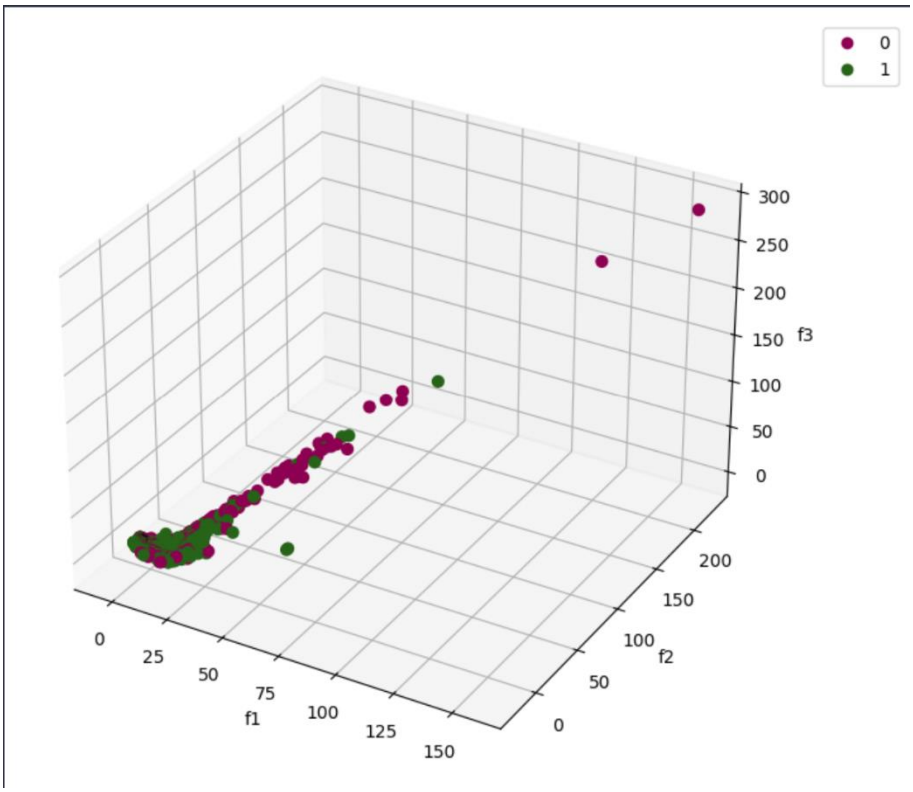
we computed the PCA of the feature vector space and only kept the highest 3 vectors representing the highest 3 variance in data preserving around 45% of the variance in data and applied that transformation:

```
original variance ratio explained by each feature: [2.16688502e-01 1.28211880e-01 1.08513482e-01 5.99161864e-02
 5.37825377e-02 4.71213394e-02 3.95796086e-02 3.77148799e-02
 3.63208666e-02 2.92822137e-02 2.81302683e-02 2.60239325e-02
 2.19051260e-02 1.90227092e-02 1.55935030e-02 1.39957024e-02
 1.30833064e-02 1.13923595e-02 1.07736854e-02 9.06517270e-03
 8.69111537e-03 7.57683344e-03 7.26260244e-03 6.36498405e-03
 5.55816062e-03 5.13255068e-03 4.83470158e-03 4.63552049e-03
 3.68212360e-03 3.43941676e-03 2.94003852e-03 2.70207849e-03
 1.80615628e-03 1.44098455e-03 1.21425228e-03 1.06168390e-03
 8.86454569e-04 7.86933023e-04 6.66956843e-04 6.12755235e-04
 4.70221268e-04 4.33253652e-04 3.01627010e-04 2.87749296e-04
 2.50426545e-04 2.27189759e-04 1.79441680e-04 1.47363131e-04
 1.05865205e-04 5.61777252e-05 3.18342008e-05 3.13009751e-05
 1.70794988e-05 1.45440365e-05 1.29132166e-05 8.48677714e-06
 5.98349307e-06 4.97764905e-06 6.66574708e-23 1.11934320e-24
 4.27342720e-32 8.08127897e-33 5.54133690e-33 4.48739675e-33
 9.99059813e-34 2.87234132e-34 1.85086134e-34] and total = 1.0000000000000002
cumulative sum of variance ratio = [0.2166885  0.34490038 0.45341386 0.51333005 0.56711259 0.61423393
 0.65381354 0.69152842 0.72784928 0.7571315  0.78526177 0.8112857
 0.83319082 0.85221353 0.86780704 0.88180274 0.89488604 0.9062784
 0.91705209 0.92611726 0.93480838 0.94238521 0.94964781 0.9560128
 0.96157096 0.96670351 0.97153821 0.97617373 0.97985585 0.98329527
 0.98623531 0.98893739 0.99074354 0.99218453 0.99339878 0.99446047
 0.99534692 0.99613385 0.99680081 0.99741356 0.99788379 0.99831704
 0.99861867 0.99890642 0.99915684 0.99938403 0.99956347 0.99971084
 0.9998167  0.99987288 0.99990471 0.99993602 0.99995309 0.99996764
 0.99998055 0.99998904 0.99999502 1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.        ]
new number of dimensions = 3
```

the data has become a 3 dimensional data that can be visualized, we plotted 2 dimensional data as x and y coordinates where the size of the point is the third dimensional feature. The Color of the point represents its class:



We then plotted the same graph but instead of size of point representing 3rd dimension, we used z axis to represent the 3rd dimension:



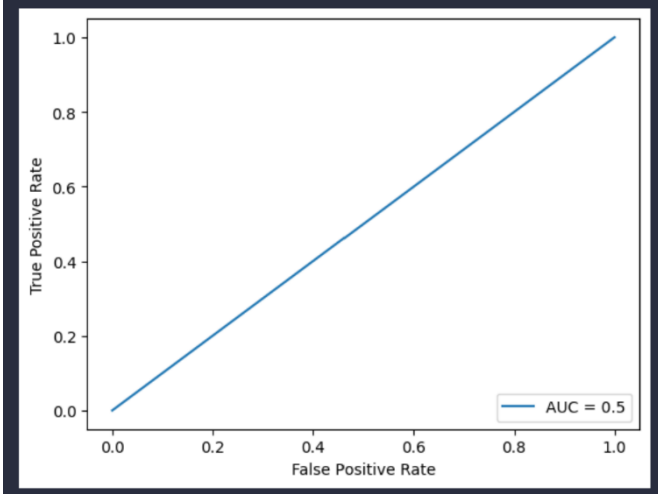
From the above 2 graphs, we can see there is a huge interference between the 2 classes data points. And finding that separator would be nearly impossible that’s why our traditional models has resulted in poor accuracy.

ZeroR Model

This model always predicts the test data to have a class of the most frequently seen class in the training data. This model does not have any parameters to optimize.

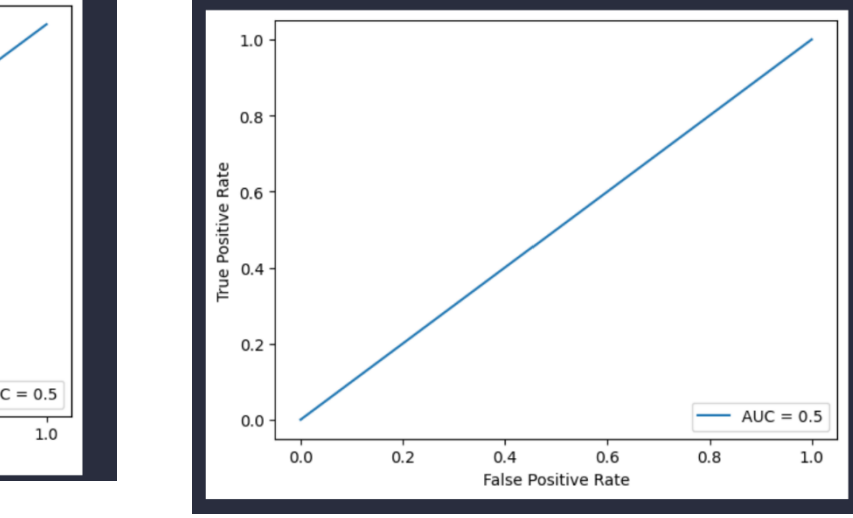
Dataset (un-modified)

```
============================
model Name: ZeroR
============================
best hyper parameters are:
-
----------------------------
confusion matrix:
+-----------------+-------------+-----------------+
|                 | Trojan(Gold) | Not Trojan(Gold) |
+-----------------+-------------+-----------------+
|   Trojan(Pred)  |    22671    |      21700      |
| Not Trojan(Pred) |     0      |        0        |
+-----------------+-------------+-----------------+
----------------------------
Accuracy = 0.51
Recall = 1.0
Precision = 0.51
F1-Score = 0.68
Model Size = 215 bytes
Training Time = 0.02 sec
Prediction Time = 0.0 sec
----------------------------
```



Dataset (modified)

```
============================
model Name: ZeroR
============================
best hyper parameters are:
-
----------------------------
confusion matrix:
+-----------------+-------------+-----------------+
|                 | Trojan(Gold) | Not Trojan(Gold) |
+-----------------+-------------+-----------------+
|   Trojan(Pred)  |    22671    |      21700      |
| Not Trojan(Pred) |     0      |        0        |
+-----------------+-------------+-----------------+
----------------------------
Accuracy = 0.51
Recall = 1.0
Precision = 0.51
F1-Score = 0.68
Model Size = 215 bytes
Training Time = 0.02 sec
Prediction Time = 0.0 sec
----------------------------
```



As you notice, zeroR will always result in 100% recall. our feature extraction and analysis didn't result in improvements and it shouldn't.

Logistic Regression Model

logistic regression is just some combination of weights multiplied to the data points features to predict the class of the model.

We applied Grid Search with K - cross Validation of K=5 on the following parameters with following values:

1. **Penalty:** ‘l2’, ‘l1’, ‘elasticnet’.
2. **C:** ‘0.01’, ‘1’, ‘100’.
3. **Solver:** ‘lbfgs’, ‘liblinear’, ‘newton-cg’, ‘newton-cholesky’, ‘sag’, ‘saga’.
4. **max_iter:** ‘100’, ‘1000’, ‘10000’.
5. **multi_class:** ‘auto’, ‘ovr’, ‘multinomial’.

This resulted in around 486 candidates with 2430 fits. Grid Search Completed in around 10 minutes.

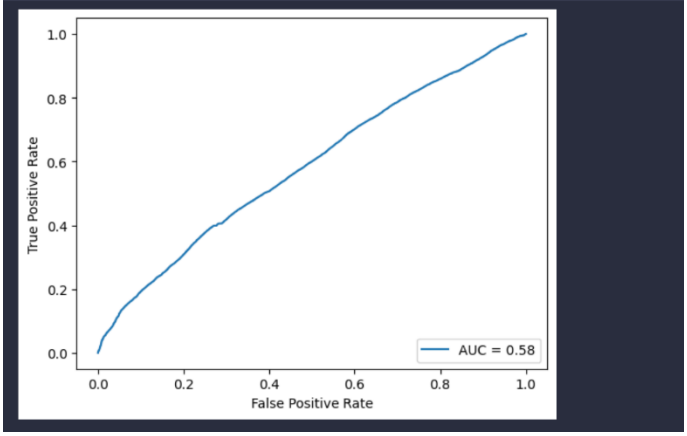
The best validation accuracy we could get is 0.53.

The following were found to be the best parameters:

- **C:** 0.01
- **max_iter:** 100
- **multi_class:** auto
- **penalty:** l1
- **solver:** liblinear

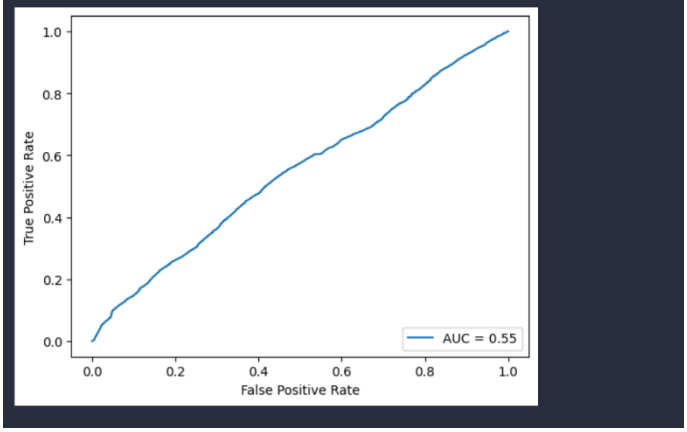
Dataset (un-modified)

```
============================
model Name: Logistic Regression
============================
best hyper parameters are:
{'C': 0.01, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling':
----------------------------
confusion matrix:
+-----------------+--------------+-----------------+
|                 | Trojan(Gold) | Not Trojan(Gold) |
+-----------------+--------------+-----------------+
|   Trojan(Pred)  |    12898     |      10070      |
| Not Trojan(Pred) |     9773     |      11630      |
+-----------------+--------------+-----------------+
----------------------------
Accuracy = 0.55
Recall = 0.57
Precision = 0.56
F1-Score = 0.57
Model Size = 1465 bytes
Training Time = 16.19 sec
Prediction Time = 0.01 sec
----------------------------
```



Dataset (modified)

```
============================
model Name: Logistic Regression
============================
best hyper parameters are:
{'C': 0.01, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1.
----------------------------
confusion matrix:
+-----------------+--------------+-----------------+
|                 | Trojan(Gold) | Not Trojan(Gold) |
+-----------------+--------------+-----------------+
|   Trojan(Pred)  |    13285     |      11137      |
| Not Trojan(Pred) |     9386     |      10563      |
+-----------------+--------------+-----------------+
----------------------------
Accuracy = 0.54
Recall = 0.59
Precision = 0.54
F1-Score = 0.56
Model Size = 902 bytes
Training Time = 0.8 sec
Prediction Time = 0.0 sec
----------------------------
```



As you can see, our feature analysis and extraction results in:

- 1% lower accuracy
- 2% higher recall
- 2% lower Precision
- 1% lower F1-Score
- 62% lower model size
- 1900% faster training time
- 1000% faster prediction time
- 7% lower AUC

Support Vector Machine Model

Support vector machine is just finding a linear separator or converting non-linear separable data into linear separable where this linear separator will have the highest margin between its support vectors.

We applied Grid Search with K - cross Validation of K=5 on the following parameters with following values:

1. **degree:** '3', '7', 'll'.
2. **C:** '0.01', '1', '100'.
3. **kernel:** 'linear, 'poly', 'rbf', 'sigmoid'.
4. **Gamma:** 'scale', 'auto'.
5. **max_iter:** '100', '500', '1000'.
6. **decision_function_shape:** 'ovo, 'ovr'.

This resulted in around 432 candidates with 2160 fits. Grid Search Completed in around 289 minutes.

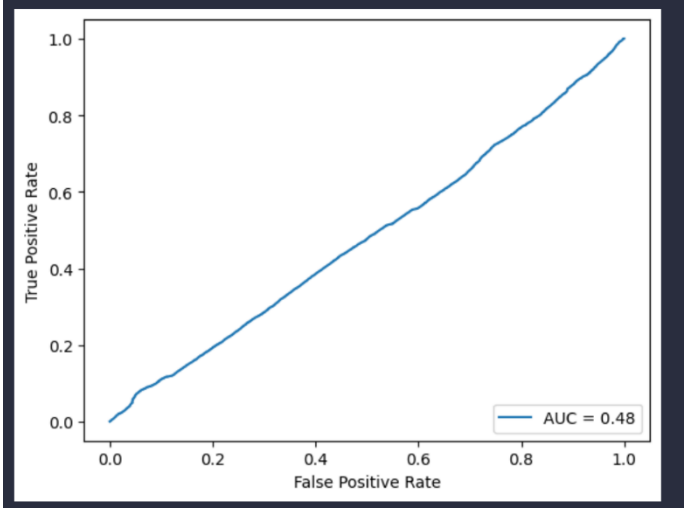
The best validation accuracy we could get is 0.51.

The following were found to be the best parameters:

- **C:** 1
- **decision_function_shape:** ovo
- **degree:** 3
- **gamma:** auto
- **kernel:** poly
- **max_iter:** 100

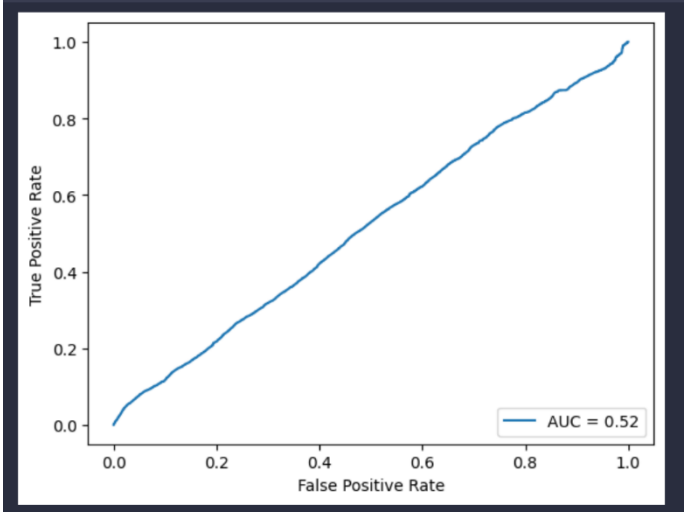
Dataset (un-modified)

```
============================
model Name: Support Vector Machine
============================
best hyper parameters are:
{'C': 1, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0
----------------------------
confusion matrix:
+-----------------+--------------+------------------+
|                 | Trojan(Gold) | Not Trojan(Gold) |
+-----------------+--------------+------------------+
|   Trojan(Pred)  |    22669     |      21692       |
| Not Trojan(Pred) |      2       |        8         |
+-----------------+--------------+------------------+
----------------------------
Accuracy = 0.51
Recall = 1.0
Precision = 0.51
F1-Score = 0.68
Model Size = 97294 bytes
Training Time = 21.37 sec
Prediction Time = 0.31 sec
----------------------------
```



Dataset (modified)

```
============================
model Name: Support Vector Machine
============================
best hyper parameters are:
{'C': 1, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0
----------------------------
confusion matrix:
+-----------------+--------------+------------------+
|                 | Trojan(Gold) | Not Trojan(Gold) |
+-----------------+--------------+------------------+
|   Trojan(Pred)  |    22654     |      21696       |
| Not Trojan(Pred) |      17      |        4         |
+-----------------+--------------+------------------+
----------------------------
Accuracy = 0.51
Recall = 1.0
Precision = 0.51
F1-Score = 0.68
Model Size = 24377 bytes
Training Time = 12.98 sec
Prediction Time = 0.27 sec
----------------------------
```



As you can see, our feature analysis and extraction results in:

- 0% change in accuracy
- 0% change in recall
- 0% change in Precision
- 0% change in F1-Score
- 300% lower model size
- 64% faster training time
- 15% faster prediction time
- 8% higher AUC

Adaptive boosting Classifier Model

Adaboost is just using many classifiers where every classifier tries to improve the result of the preceding classifier during training and at the end, every classifier votes to the classification of a point using a weight given to it during training.

In our Adaboost model, we are using Decision trees with depth=1 (stamp) as our basic classifiers.

We applied Grid Search with K - cross Validation of K=5 on the following parameters with following values:

1. **n_estimators:** '10', '100', '1000'.
2. **learning_rate:** '0.01', '1', '100'.
3. **algorithm:** 'SAMME, 'SAMME.R'.

This resulted in around 18 candidates with 90 fits. Grid Search Completed in around 15 minutes.

The best validation accuracy we could get is 0.73.

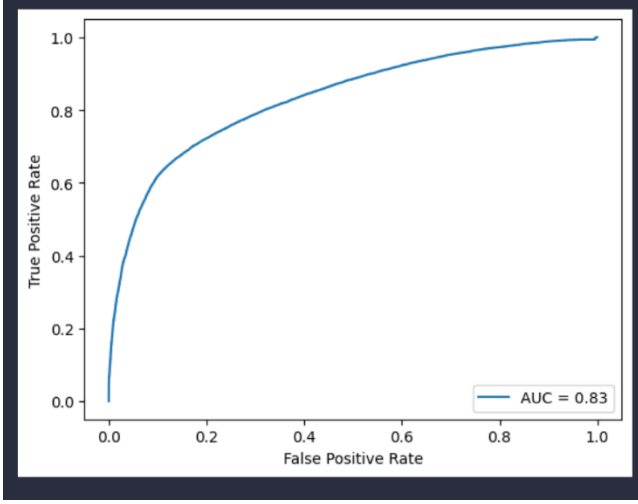
The following were found to be the best parameters:

- **algorithm:** SAMME.R
- **learning_rate:** 1
- **n_estimators:** 1000

we notices the increasing the number of estimators increases the test accuracy. So we put n_estimators to be 5000 not 1000 as after 5000 we noticed only a small change in the accuracy.

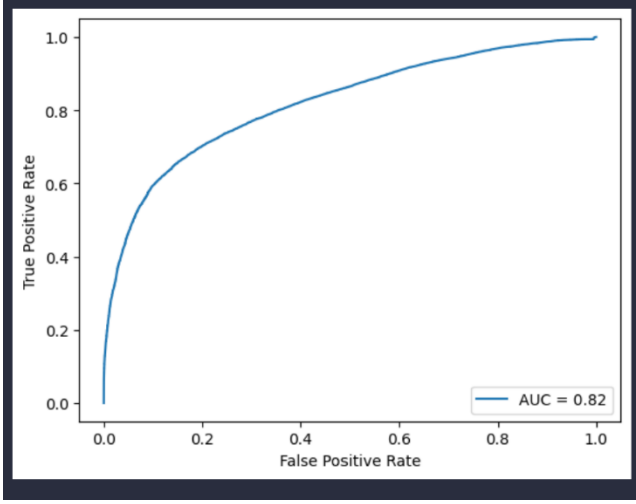
Dataset (un-modified)

```
============================
model Name: Adaptive Boosting Classifier
============================
best hyper parameters are:
{'algorithm': 'SAMME.R', 'estimator': None, 'learning_rate': 1, 'n_estimators': 5
------------------------------
confusion matrix:
+------------------+--------------+------------------+
|                  | Trojan(Gold) | Not Trojan(Gold) |
+------------------+--------------+------------------+
|   Trojan(Pred)   |    15207     |       3081       |
| Not Trojan(Pred) |     7464     |      18619       |
+------------------+--------------+------------------+
------------------------------
Accuracy = 0.76
Recall = 0.67
Precision = 0.83
F1-Score = 0.74
Model Size = 2731712 bytes
Training Time = 3433.29 sec
Prediction Time = 72.56 sec
------------------------------
```



Dataset (modified)

```
============================
model Name: Adaptive Boosting Classifier
============================
best hyper parameters are:
{'algorithm': 'SAMME.R', 'estimator': None, 'learning_rate': 1, 'n_estimators': 5
------------------------------
confusion matrix:
+------------------+--------------+------------------+
|                  | Trojan(Gold) | Not Trojan(Gold) |
+------------------+--------------+------------------+
|   Trojan(Pred)   |    14440     |       2918       |
| Not Trojan(Pred) |     8231     |      18782       |
+------------------+--------------+------------------+
------------------------------
Accuracy = 0.75
Recall = 0.64
Precision = 0.83
F1-Score = 0.72
Model Size = 2731712 bytes
Training Time = 969.3 sec
Prediction Time = 23.45 sec
------------------------------
```



As you can see, our feature analysis and extraction results in:

- 1% lower accuracy
- 3% lower recall
- 0% change in Precision
- 2% lower F1-Score
- 0% change in model size
- 250% faster training time
- 213% faster prediction time
- 1% lower AUC

Other Models

We used a package called LazyClassifier that trains multiple different Sklearn models on our models and give some insights on what models would be better for our data and here are the results:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
RandomForestClassifier	0.72	0.72	None	0.72	145.78
ExtraTreesClassifier	0.71	0.71	None	0.71	35.30
BaggingClassifier	0.71	0.71	None	0.71	80.34
KNeighborsClassifier	0.69	0.69	None	0.69	6.89
DecisionTreeClassifier	0.68	0.68	None	0.68	15.22
ExtraTreeClassifier	0.67	0.67	None	0.67	1.42
LGBMClassifier	0.65	0.65	None	0.65	1.74
NuSVC	0.62	0.63	None	0.61	29369.69
SVC	0.61	0.61	None	0.61	2095.41
AdaBoostClassifier	0.59	0.59	None	0.59	46.81
LogisticRegression	0.54	0.54	None	0.53	1.28
LinearSVC	0.54	0.53	None	0.53	43.33
CalibratedClassifierCV	0.54	0.53	None	0.53	3.42
NearestCentroid	0.53	0.53	None	0.53	1.02
RidgeClassifierCV	0.54	0.53	None	0.53	1.55
LinearDiscriminantAnalysis	0.53	0.53	None	0.53	1.52
RidgeClassifier	0.53	0.53	None	0.53	1.35
BernoulliNB	0.53	0.53	None	0.53	1.43
QuadraticDiscriminantAnalysis	0.52	0.53	None	0.44	1.20
SGDClassifier	0.52	0.52	None	0.52	1.71
PassiveAggressiveClassifier	0.52	0.52	None	0.52	1.20
GaussianNB	0.50	0.51	None	0.44	1.11
DummyClassifier	0.51	0.50	None	0.35	0.94
Perceptron	0.49	0.49	None	0.49	1.25

Experimental Results

1. We tried to convert IP addresses from Format ‘A.B.C.D’ into the Format ‘AAABBBCCCDDD’ so for example an IP address having a value 10.42.0.4 will be 010042000004 and so on but this yielded very big numbers and we decided the number isn’t representative of the address and using other techniques as label encoder (we dealt with the IP address as categories) and using one hot encoder for IP address would just increase the feature vector space dramatically.
2. We tried PCA to reduce the dimensions of our feature vector space from 67 to 20 preserving around 95% of the variance but it didn’t improve the accuracy so we used other feature selections approach.

Other Approaches

We only found another one notebook that achieved an accuracy around 95% using SVC but we found 2 problems with that notebook like:

- He used training data with test in feature analysis and extraction.
- He considered around 75% of training data to be outliers.

After fixing these problems in his notebook, the SVC he used resulted in around 49.7% accuracy.

Conclusion

1. Either the distribution of the data is very hard to separate from each other or the labeling of the data is incorrect.
2. Using feature engineering on such data didn't improve accuracy (decreased the accuracy by a very small number) but made both training and prediction faster significantly. Whereas the model size decreased significantly.
3. Many features are correlated with each other meaning that they have the same meaning.