Networks and Systems Security
Assignment - 4

Name: Suraj Kiran Mate
Entry No: 2021JCS2387

Task - 1:

```
dos["Message Identifier"].nunique(dropna = True)
```

```
27
```

In **dos_dataset 27** different message identifiers are recorded.

```
dos["Message Identifier"].unique()
```

```
array(['018f', '0260', '02a0', '0329', '0545', '0002', '0153', '02c0',
       '0130', '0131', '0140', '0350', '043f', '0370', '0440', '0316',
       '04f0', '0430', '04b1', '01f1', '05f0', '00a0', '00a1', '0690',
       '05a0', '05a2', '0000'], dtype=object)
```

These identifiers are shown in the above image.

```
[28]
ids["Message Identifier"].nunique(dropna = True)
```

```
26
```

In **gear_dataset 26** different message identifiers are recorded.

```
[8]
impersonation["Message Identifier"].unique()
✓ 0.1s
```

```
array(['02c0', '0350', '0370', '043f', '0440', '0316', '018f', '0002',
       '0153', '0260', '0130', '0131', '0140', '02a0', '0329', '0545',
       '04f0', '0430', '04b1', '01f1', '05f0', '00a0', '00a1', '0690',
       '05a0', '05a2'], dtype=object)
```

These identifiers are shown in the above image.

Task - 2:

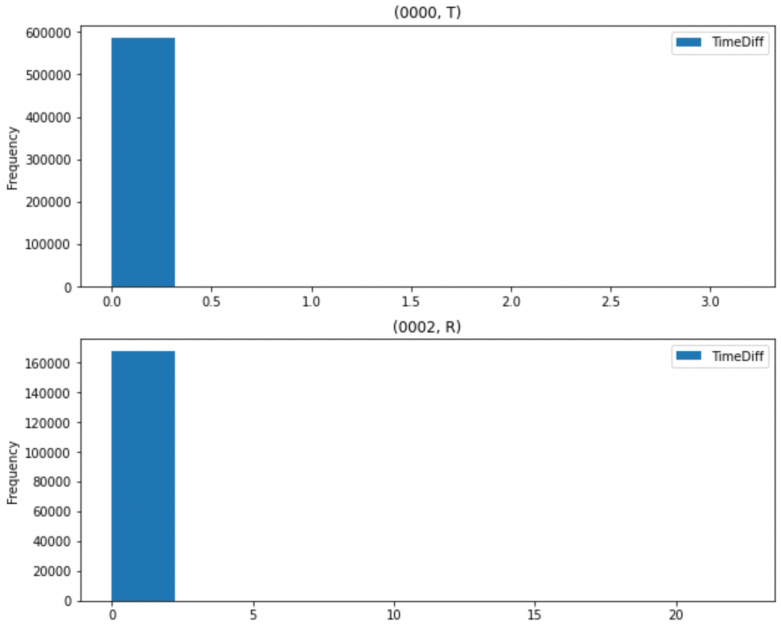


Fig 2.1: Histogram for dos attack messages

We plotted the above histograms using matplotlib. We can see that in dos_dataset, the attacker is using ID **0000** and the benign unit is using ID **0002**. For an attacker the signals are also periodic but the frequency of these signals is very high in just **0-0.3 time units** while the benign messages are also periodic but the frequency of these signals is very high in **0-2.5 time units** range.

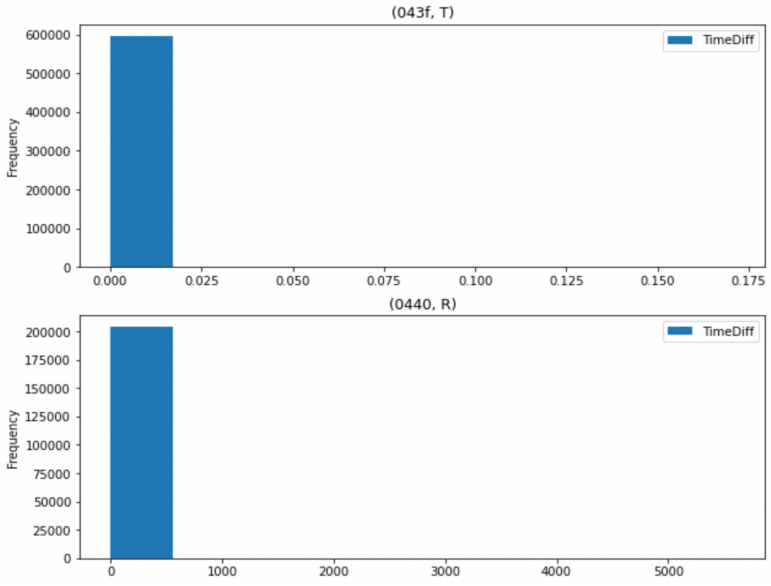


Fig 2.2: Histogram for impersonation attack messages

Similarly, in impersonation attack the attackers signals are more dense in the range **0.0000-0.0015 time units** and benign signals are periodic and frequent in the range **0-250 time units**.

Task - 3:

In our given datasets we have a timestamp column representing the time of arrival of the messages. Then the Message Identifier, Data Length, Message content and TimeDiff. We are going to use all above features to detect the Target of the newly given datapoint. This is called classification problem in machine learning. In such classification problems we are always given multiple independent variables which are used as training vectors by the machine learning model. The models get trained using these vectors and then classify the new data point when all these independent variable values are given to it.

There are many classification algorithms that can be used for this purpose, but KNN (K-nearest neighbors) is the perfect fit for this problem case. We can also use ANN (Artificial neural Networks) which are universal for all types of problems but complex to implement. Logistic regression is the most simplest classification algorithm with decent accuracy. Random Forest and Decision Trees can also be used for this classification.

But we will stick to most fitting algorithm ie. KNN. In KNN based on the N number of independent variables the data points are mapped in N dimensional space. While training the data points are mapped in N dimensional space and when the ned data point is given for classification then simply this new datapoint is also mapped in this space and distance of this point from the constellation of other points is computed. When the distance from some specific constellation is minimum then that datapoint is mapped to that constellation.

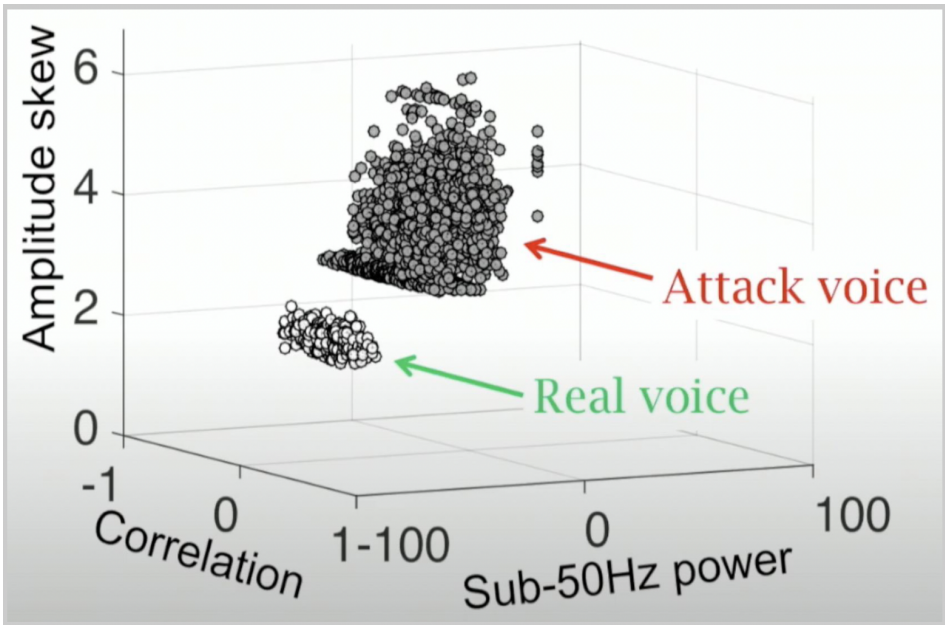


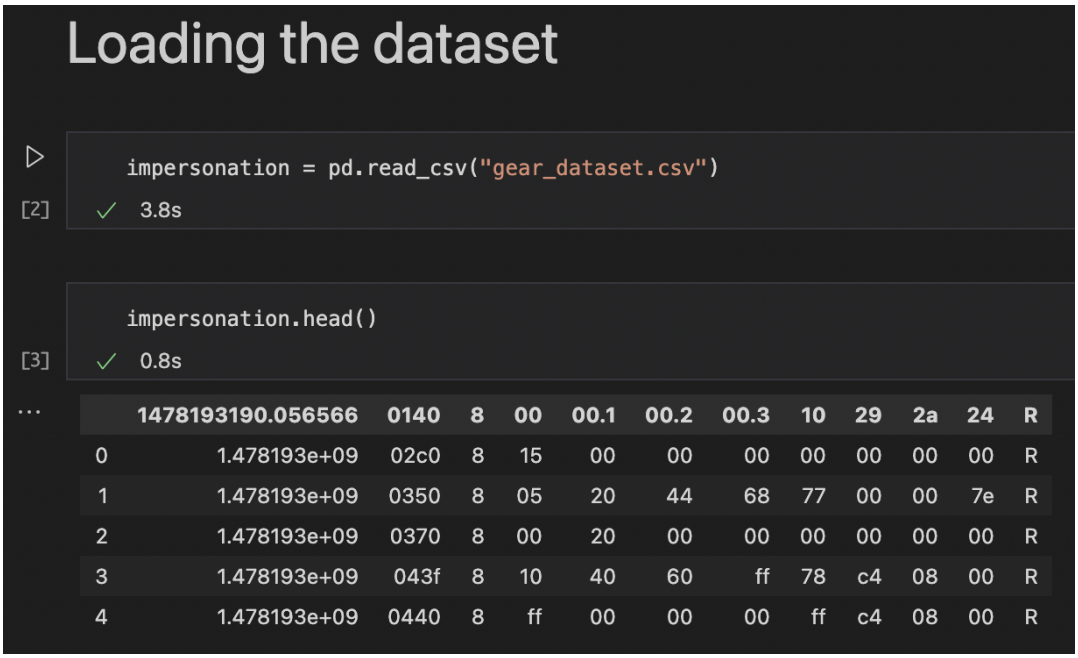
Fig 3.1: Different Data constellations

Above is the image from my research paper that I have implemented in Wireless Networks course where we have implemented KNN for classification of attacker's voice command and benign user's voice command to voice enabled devices. Here we are using Amplitude skew, correlation and sub-50Hz power as our spatial vectors to map the data points and then classify this model outcome. We can clearly see the constellation of points in 3 dimensional space (because of 3 features).

In our dataset we have 12 different independent variables hence each datapoint will be mapped in 12 dimensional space.

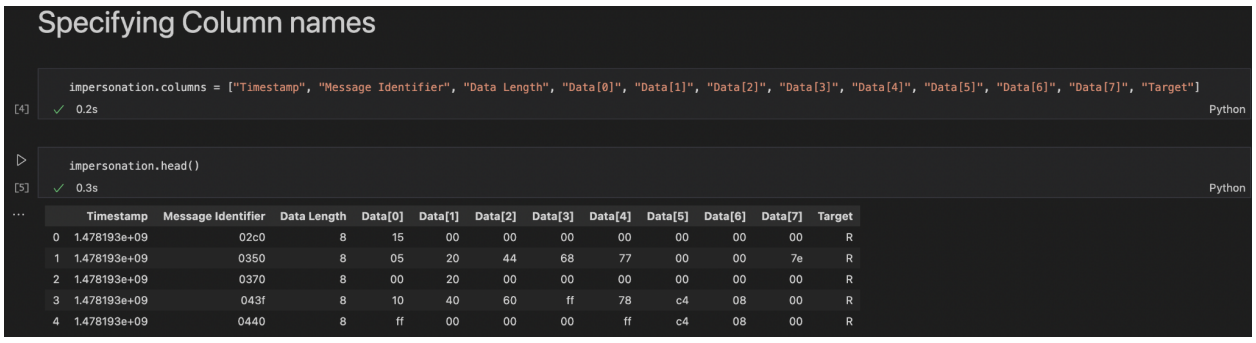
Starting with our problem we are going to explain only **ids_impersonation** all settings are exactly the same for **ids_dos**.

We are first loading the dataset as below:



	1478193190.056566	0140	8	00	00.1	00.2	00.3	10	29	2a	24	R
0	1.478193e+09	02c0	8	15	00	00	00	00	00	00	00	R
1	1.478193e+09	0350	8	05	20	44	68	77	00	00	7e	R
2	1.478193e+09	0370	8	00	20	00	00	00	00	00	00	R
3	1.478193e+09	043f	8	10	40	60	ff	78	c4	08	00	R
4	1.478193e+09	0440	8	ff	00	00	00	ff	c4	08	00	R

After looking at the data we can see that the columns are not named. Hence we have to first label the columns for easy understanding and training our model.



	Timestamp	Message Identifier	Data Length	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	Target
0	1.478193e+09	02c0	8	15	00	00	00	00	00	00	00	R
1	1.478193e+09	0350	8	05	20	44	68	77	00	00	7e	R
2	1.478193e+09	0370	8	00	20	00	00	00	00	00	00	R
3	1.478193e+09	043f	8	10	40	60	ff	78	c4	08	00	R
4	1.478193e+09	0440	8	ff	00	00	00	ff	c4	08	00	R

These are the column names we have specified. Now it is easy to understand which column belongs to what.

Then we are checking NULL or NA values in the data frame. This checking is important to avoid any further errors arising due to presence of them in our dataset. Then we are checking the unique values in Message Identifier and also Unique values in Target.

Since the most important measure here for classification is the time difference between two consecutive messages coming from the same Message Identifier. Hence we first need to compute this time difference.

Creating time difference column

```
# Finding time difference between all consecutive entries belonging to same message identifier and storing it in TimeDiff column
impersonation["TimeDiff"] = impersonation.groupby("Message Identifier")["Timestamp"].diff()
```

[14] ✓ 0.8s

```
impersonation
```

[15] ✓ 0.3s

	Timestamp	Message Identifier	Data Length	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	Target	TimeDiff
0	1.478193e+09	02c0	8	15	00	00	00	00	00	00	00	R	NaN
1	1.478193e+09	0350	8	05	20	44	68	77	00	00	7e	R	NaN
2	1.478193e+09	0370	8	00	20	00	00	00	00	00	00	R	NaN
3	1.478193e+09	043f	8	10	40	60	ff	78	c4	08	00	R	NaN
4	1.478193e+09	0440	8	ff	00	00	00	ff	c4	08	00	R	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4443136	1.478201e+09	018f	8	fe	59	00	00	00	41	00	00	R	0.009850
4443137	1.478201e+09	0260	8	18	21	21	30	08	8f	6d	19	R	0.009847
4443138	1.478201e+09	02a0	8	24	00	9a	1d	97	02	bd	00	R	0.009845
4443139	1.478201e+09	0329	8	dc	b7	7f	14	11	20	00	14	R	0.009844
4443140	1.478201e+09	0545	8	d8	00	00	8b	00	00	00	00	R	0.009847

4402976 rows × 13 columns

The similar code will also work for dos_dataset. We can see that a new column named **TimeDiff** at the end is added. This column stores the values of time difference between consecutive data entries belonging to same Message Identifier. Hence values at the start are NaN ie. no earlier values were present to find the difference. In code the **groupby** takes names of column by which we have to group the value differences. Here we are passing a Message Identifier.

These NaN values can corrupt our model training hence we are going to remove these values from our dataset.

```
impersonation = impersonation.dropna(subset="TimeDiff")
```

[16] ✓ 0.6s

```
impersonation
[17] ✓ 0.2s
```

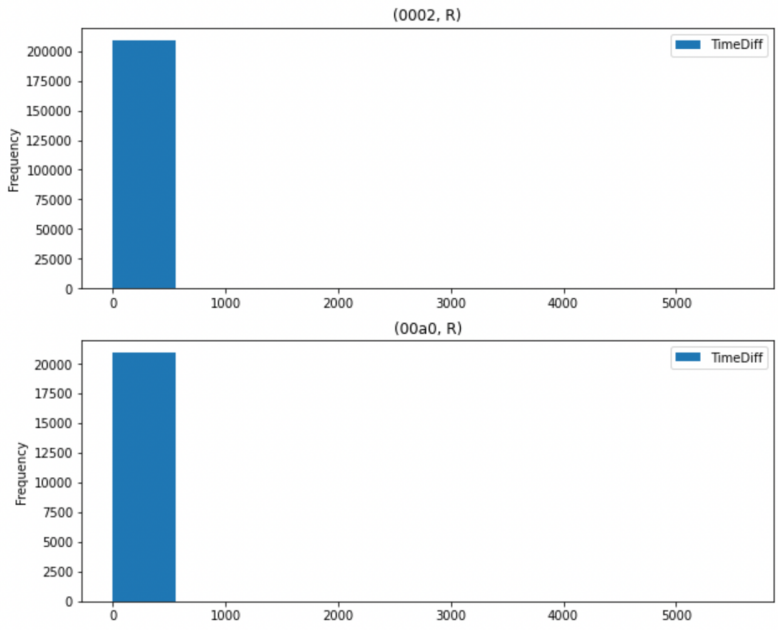
	Timestamp	Message Identifier	Data Length	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	Target	TimeDiff
16	1.478193e+09	043f	8	10	40	60	ff	78	c3	08	00	R	0.008940
17	1.478193e+09	02c0	8	15	00	00	00	00	00	00	00	R	0.009914
18	1.478193e+09	0350	8	05	20	54	68	77	00	00	6e	R	0.009909
19	1.478193e+09	0370	8	00	20	00	00	00	00	00	00	R	0.009913
20	1.478193e+09	0440	8	ff	00	00	00	ff	c3	08	00	R	0.009677
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4443136	1.478201e+09	018f	8	fe	59	00	00	00	41	00	00	R	0.009850
4443137	1.478201e+09	0260	8	18	21	21	30	08	8f	6d	19	R	0.009847
4443138	1.478201e+09	02a0	8	24	00	9a	1d	97	02	bd	00	R	0.009845
4443139	1.478201e+09	0329	8	dc	b7	7f	14	11	20	00	14	R	0.009844
4443140	1.478201e+09	0545	8	d8	00	00	8b	00	00	00	00	R	0.009847

4402951 rows × 13 columns

This is the cleaned version of our dataset.

Now we are going to plot a histogram to detect the time range in which the difference between arrival of two consecutive commands belonging to the same Message Identifier are lying. So for that we are using Histogram in matplotlib library.

```
[18] ✓ 15.1s
ax = impersonation.plot.hist(column = ["TimeDiff"], by=["Message Identifier", "Target"], figsize = (10,120))
```



Using the above command we are able to plot the histograms. We are plotting these histograms on a TimeDiff column and using **Message Identifier** and **Target** Column for differentiating different figures. Thus multiple plots are plotted based on the Target label and Message Identifier of the message. The first plot in the above image

shows the histogram for **0002** Message Identifier and message source is similar for the second histogram.

Creating X and y (X-independent variables, y- dependent target value)

```
X = impersonation.drop("Target", axis = 1)
```
[19] ✓ 0.4s

```
X.head()
```
[20] ✓ 0.3s

	Timestamp	Message Identifier	Data Length	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	TimeDiff
16	1.478193e+09	043f	8	10	40	60	ff	78	c3	08	00	0.008940
17	1.478193e+09	02c0	8	15	00	00	00	00	00	00	00	0.009914
18	1.478193e+09	0350	8	05	20	54	68	77	00	00	6e	0.009909
19	1.478193e+09	0370	8	00	20	00	00	00	00	00	00	0.009913
20	1.478193e+09	0440	8	ff	00	00	00	ff	c3	08	00	0.009677

```
y = impersonation["Target"]
```
[21] ✓ 0.3s

```
y
```
[22] ✓ 0.2s

Creating X and y sections of the dataset. Here the X section has all the independent variables in the dataset and the y section has only the target values. Ie. Model will use the X section and predict the corresponding y section.

Encoding Complete Data

```
# Since KNN cant work well with non numeric data. Hence it becomes necessary to encode the data into numeric values.
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```
[23] ✓ 0.7s

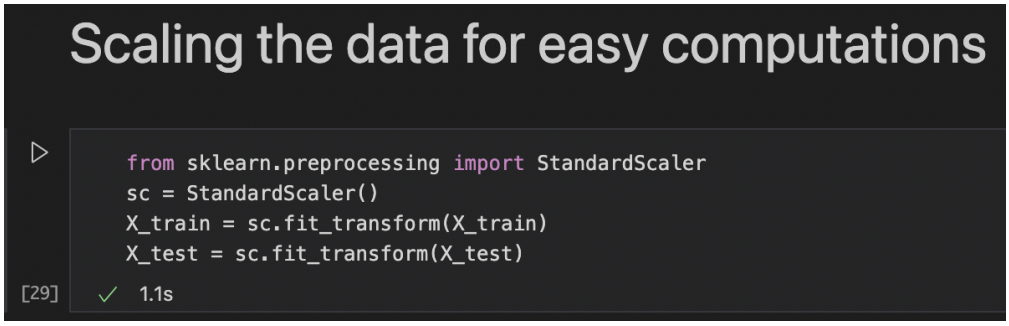
```
X["Message Identifier"] = le.fit_transform(X["Message Identifier"])
```
[24] ✓ 0.7s

```
X["Data[0]"] = le.fit_transform(X["Data[0]"])
X["Data[1]"] = le.fit_transform(X["Data[1]"])
X["Data[2]"] = le.fit_transform(X["Data[2]"])
X["Data[3]"] = le.fit_transform(X["Data[3]"])
X["Data[4]"] = le.fit_transform(X["Data[4]"])
X["Data[5]"] = le.fit_transform(X["Data[5]"])
X["Data[6]"] = le.fit_transform(X["Data[6]"])
X["Data[7]"] = le.fit_transform(X["Data[7]"])
```
[26] ✓ 6.6s

```
# Data completely Encoded into numeric values
X.head()
```
[27] ✓ 0.4s

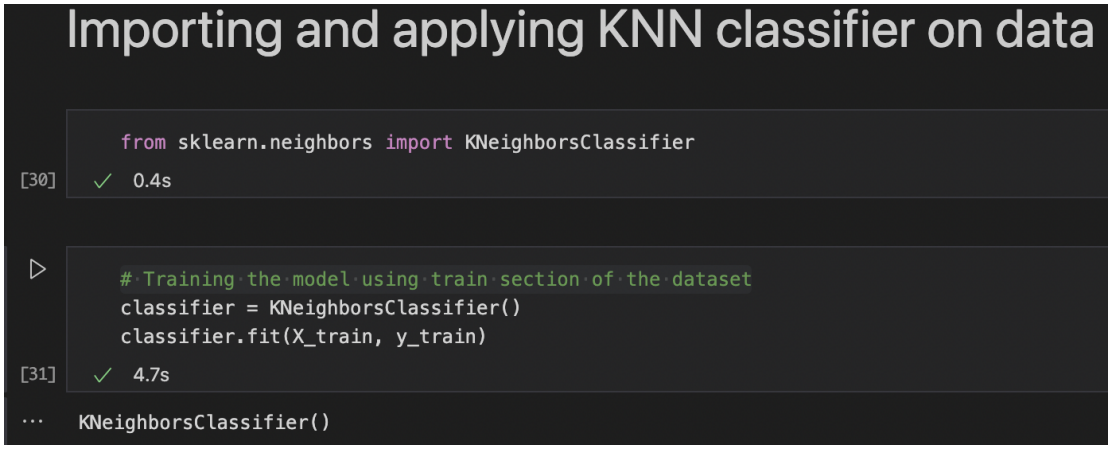
	Timestamp	Message Identifier	Data Length	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	TimeDiff
16	1.478193e+09	17	8	16	38	50	35	120	195	8	0	0.008940
17	1.478193e+09	11	8	21	0	0	0	0	0	0	0	0.009914
18	1.478193e+09	14	8	5	13	47	14	119	0	0	110	0.009909
19	1.478193e+09	15	8	0	13	0	0	0	0	0	0	0.009913
20	1.478193e+09	18	8	149	0	0	0	220	195	8	0	0.009677

Since most of the ML algorithms do not work on categorical data and also for KNN it is difficult to plot this categorical data in space. Hence it is always better to encode the data into numeric values. For that we are LabelEncoder from **sklearn** library. We can see the encoded version of data only has numeric values.



Then we are going to split the data into train and test sections. For this are using **train_test_split** from **sklearn** library. We are taking **20%** of data for testing and the rest **80%** will be used for model training.

Because of scaling all the data points will be closer to each other but the relative distance between two different data points will still remain the same.



Then we are importing and applying KNN classifier. The cell 31 represents the training of the model. X_train and y_train parameters are passed for training the KNN classifier.

Now we are going to predict the values using this classifier. For this we are using X_test data and getting y_pred. We can see that prediction takes too much time, around **7mins**. The y_pred variable contains all the predictions of 20% of the data.

```
# Calculating confusion matrix and accuracy values
from sklearn.metrics import confusion_matrix, accuracy_score
confMat = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
```

[33] ✓ 0.2s

Now we are going to compute the confusion matrix and accuracy values. For that we are importing **confusion_matrix** and **accuracy_score** from sklearn library and we are passing **y_test** and **y_pred** to compute both of these values.

```
# Confusion matrix is the best measure tounderstand the true positive and false positive detections of our model.
confMat
sns.heatmap(confMat, annot=True, linecolor="yellow", fmt="d")
```

[39] ✓ 0.2s

From the above confusion matrix we can see that there are 0 true negative and false positive entries. ie. Almost all the predictions are exactly correct and hence we expect that our model will give 100% accuracy.

```
# Getting accuracy
accuracy
```

[35] ✓ 0.4s

1.0

Yes, our model accuracy is 100%. ie. None of the predictions were wrong.

Task - 4:

We have divided the given dataset into 80% for training and 20% for testing. In this setting we are predicting the outcomes for 20% of the data points.

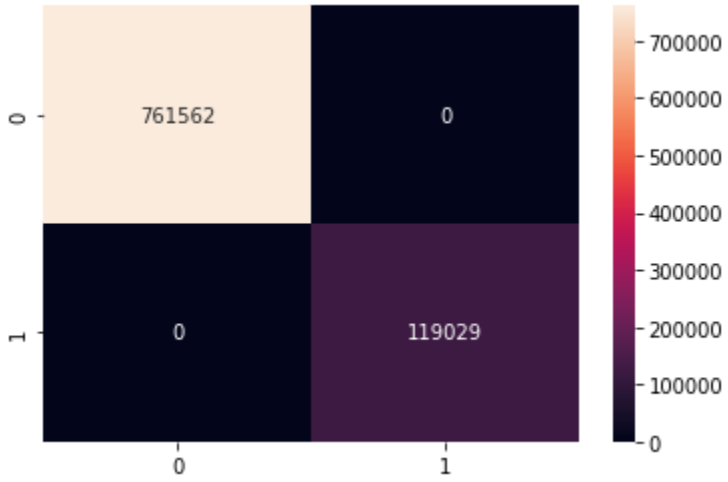


Fig 4.1: Confusion matrix for impersonation Attack

Above is the confusion matrix drawn for impersonation attack.

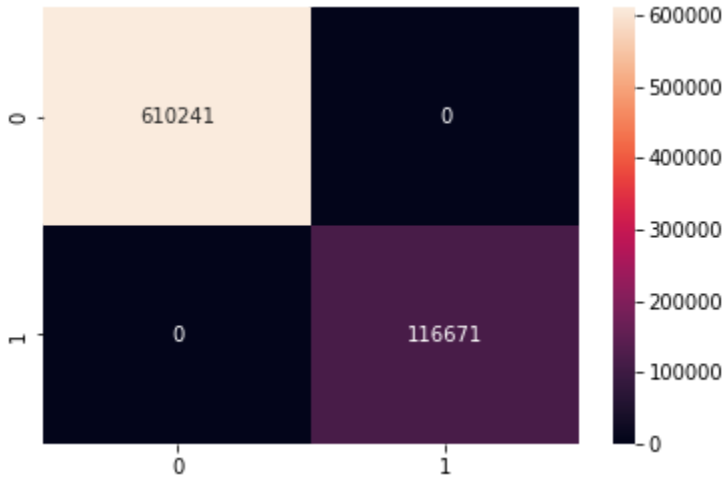


Fig 4.2: Confusion matrix for dos Attack

Above is the confusion matrix drawn for dos attack.

We can see that both **false positive** and **false negative** in both the datasets are exactly 0. Hence our model was 100% accurate. Thus the accuracy values for both datasets were 100%.

Question: How are we getting 100% accuracy in both the datasets?

Ans:

As already mentioned we are using the K nearest neighbors algorithm for this prediction problem. We have seen in **Task-2,** from the screenshots we explained that the density of data points for benign messages were lying in between **0-2.5 time units** for dos attack and in between **0-250 time units** for impersonation attack. And also we have observed that the density of time difference values was maximum in between **0-0.3 time units** in attackers case in dos attack and **0-0.015 time units** in impersonation attack.

This is a huge value difference and our KNN model could easily plot these values far from the benign messages values using **TimeDiff** dimension thus it was very easy for our model to differentiate between these two values since the distance between the constellation of benign messages and the constellation of attackers messages is too high.

KNN model gets inaccurate when the spatial distance in any dimension is not too high and can't be clearly interpreted for classification. In our problem case due to the big difference between these values it was easy classification for our model and it could perform with 100% accuracy.

How to Run this Code:

1. Run each notebook cell one by one.

2. The data should be present in the same folder where this notebook is present. Otherwise the data address should be changed.

References:

1. https://matplotlib.org/stable/users/index.html **-** Matplotlib data visualisation tool documentation.

2. https://seaborn.pydata.org - Seaborn data visualisation tool documentation.

3. https://stackoverflow.com - It is the best reference for quick query resolution.

Thank You !