



# Prediction of Malicious traffic in IoT network

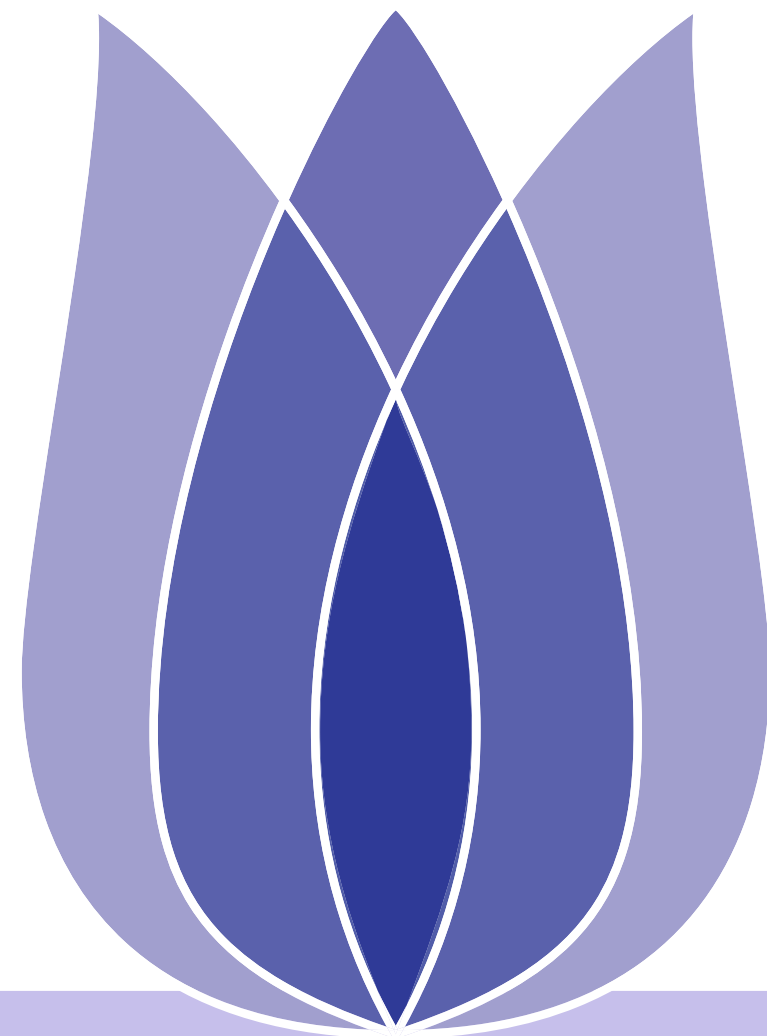
Spoorthy Reddy Jarugu

Vellore Institute of Technology

Vellore

India

May 6, 2023





# Overview

- [Problem Definition](#)
- [Data Preprocessing and visualization](#)
- [Model built and prediction](#)
- [Conclusion](#)

## Problem Definition

Prediction of Malicious traffic in IoT network

## Data Preprocessing and visualization

- Data Preprocessing
  - Cleaning the Dataset
  - IP address to int
  - Label Encoding
  - Correlation

## Model built and prediction

- Splitting the data
- Decision Tree Classifier
- Evaluating the model

## Conclusion



**Problem Definition**

Prediction of Malicious traffic in IoT network

Data Preprocessing and visualization

Model built and prediction

Conclusion

# Problem Definition



# Prediction of Malicious traffic in IoT network

Problem Definition  
Prediction of Malicious traffic in IoT network

Data Preprocessing and visualization

Model built and prediction

Conclusion

Defn

IoT devices are growing rapidly and due to there less computational power these devices are getting compromised easily. Task is to predict malicious traffic in the IoT network. Prediction is done based on the lables that are assigned is the dataset.

- In the data generated have 22 columns where time, IP address for originating point and responding point, protocol used for communication, duration, connection state, no of packets, label are recorded.
- With the combination of all the columns finally labels says the particular communication happened is **Benign** or **Malicious**.



- [Problem Definition](#)
- [Data Preprocessing and visualization](#)**
- [Data Preprocessing](#)
- [Cleaning the Dataset](#)
- [IP address to int](#)
- [Label Encoding](#)
- [Correlation](#)
- [Model built and prediction](#)
- [Conclusion](#)

# Data Preprocessing and visualization

# Data Preprocessing

- Problem Definition
- Data Preprocessing and visualization
  - Data Preprocessing
  - Cleaning the Dataset
    - IP address to int
    - Label Encoding
    - Correlation
- Model built and prediction
- Conclusion

- Dataset is in the form of CSV file named as `iot23_combined_new.csv`
- Which contains `6046623` rows and `22` columns
- We have to check for missing or NAN values.
- There are some values which is represented with special character `'_'` that have to replaced.

```
In [4]: dd.isnull().sum()
Out[4]: Unnamed: 0      0
        ts              0
        uid             0
        id.orig_h       0
        id.orig_p       0
        id.resp_h       0
        id.resp_p       0
        proto           0
        service         0
        duration        0
        orig_bytes      0
        resp_bytes      0
        conn_state      0
        local_orig      0
        local_resp      0
        missed_bytes    0
        history         0
        orig_pkts       0
        orig_ip_bytes   0
        resp_pkts       0
        resp_ip_bytes   0
        label           0
        dtype: int64
```

Figure 1: Checking for missing values

```
In [6]: dd.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6046623 entries, 0 to 6046622
Data columns (total 22 columns):
#   Column      Dtype
---  -----  ---
0   Unnamed: 0   int64
1   ts           float64
2   uid          object
3   id.orig_h    object
4   id.orig_p    float64
5   id.resp_h    object
6   id.resp_p    float64
7   proto        object
8   service      object
9   duration     object
10  orig_bytes   object
11  resp_bytes   object
12  conn_state   object
13  local_orig   object
14  local_resp   object
15  missed_bytes float64
16  history      object
17  orig_pkts    float64
18  orig_ip_bytes float64
19  resp_pkts    float64
20  resp_ip_bytes float64
21  label        object
dtypes: float64(8), int64(1), object(13)
memory usage: 1014.9+ MB
```

Figure 2: Data type of each column



# Cleaning the Dataset

[Problem Definition](#)

[Data Preprocessing and visualization](#)

[Data Preprocessing](#)

[Cleaning the Dataset](#)

[IP address to int](#)

[Label Encoding](#)

[Correlation](#)

[Model built and prediction](#)

[Conclusion](#)

- In the process of cleaning the dataset we have seen that 'uid', 'ts', 'Unnamed: 0' are having unique values so we are removing it.

```
In [9]: # We can see that uid and ts have unique value so we can remove it  
dd=dd.drop(columns=['uid','ts','Unnamed: 0'])
```

Figure 3: Dropping the values



**TULIP**

Team for Universal Learning and Intelligent Processing



# IP address to int

Problem Definition
Data Preprocessing and visualization
Data Preprocessing
Cleaning the Dataset
IP address to int
Label Encoding
Correlation
Model built and prediction
Conclusion

- Using IP address in network traffic dataset as a feature can be used for detecting anomalies or identifying network patterns.
- With the help of the below code we are converting it to int data type which is easy for machine learning to process.

Converting originator IP address string to its corresponding integer representation

```
In [12]: ip_col_name = "id.orig_h"

# define a function to convert IP addresses to integers
def ip_to_int(ip):
    try:
        return struct.unpack("!I", socket.inet_aton(ip))[0]
    except socket.error:
        return None

dd[ip_col_name] = dd[ip_col_name].apply(ip_to_int)

dd.to_csv("iot23_combined_new.csv", index=False)
```

Figure 4: Converting Ip address to int



Problem Definition
Data Preprocessing and visualization
Data Preprocessing
Cleaning the Dataset
IP address to int
Label Encoding
Correlation
Model built and prediction
Conclusion

- Protocol: With `dd['proto'].values_counts()` we can display the individual count of each protocol used in the IoT network traffic.
- With the value displayed below the `values_count` we can see that TCP is maximum used.

```
In [17]: dd['proto'].value_counts()

Out[17]: tcp      6026584
         udp       18429
         icmp      1610
         Name: proto, dtype: int64
```

Figure 5: Individual count of protocol





- [Problem Definition](#)
- [Data Preprocessing and visualization](#)
- [Data Preprocessing](#)
- [Cleaning the Dataset](#)
- [IP address to int](#)**
- [Label Encoding](#)
- [Correlation](#)
- [Model built and prediction](#)
- [Conclusion](#)

## ■ Visual representation of protocol in bar graph

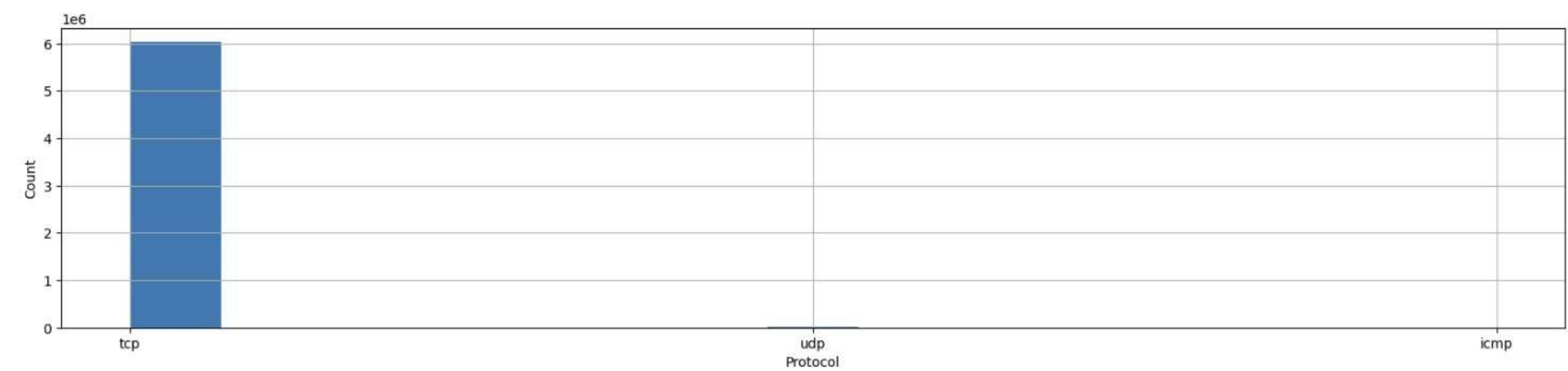


Figure 6: Protocol Visualization



[Problem Definition](#)

[Data Preprocessing and visualization](#)

[Data Preprocessing](#)

[Cleaning the Dataset](#)

[IP address to int](#)

[Label Encoding](#)

[Correlation](#)

[Model built and prediction](#)

[Conclusion](#)

- Coming to 'Service', 'duration', 'orig\_bytes', 'resp\_bytes' when using `values_counts()` we can see that

```
In [20]: dd['service']=dd['service'].str.replace('-', 'nil')
```

Figure 7: Service

```
In [25]: dd['duration']=dd['duration'].str.replace('-', '0')
```

Figure 8: Duration



**TULIP**

*Team for Universal Learning and Intelligent Processing*



[Problem Definition](#)

[Data Preprocessing and visualization](#)

[Data Preprocessing](#)

[Cleaning the Dataset](#)

[IP address to int](#)

[Label Encoding](#)

[Correlation](#)

[Model built and prediction](#)

[Conclusion](#)

```
In [25]: dd['duration']=dd['duration'].str.replace('-', '0')
```

Figure 9: Originator bytes

```
In [29]: dd['resp_bytes']=dd['resp_bytes'].str.replace('-', '0')
```

Figure 10: Respondent bytes





# Connection State

- Problem Definition
- Data Preprocessing and visualization
  - Data Preprocessing
  - Cleaning the Dataset
  - IP address to int
  - Label Encoding
  - Correlation
- Model built and prediction
- Conclusion

- Conn\_state: Connection state of the network traffic is displayed in pi chart. Where we can see 'S0' and 'OTH' occupais the maximum portion.

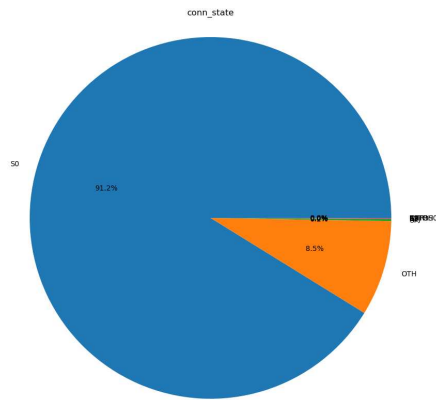


Figure 11: Connection state pi chart representation

- To understand the split up of other values we have ignored those two values and represented it in bar chart for reaming values.

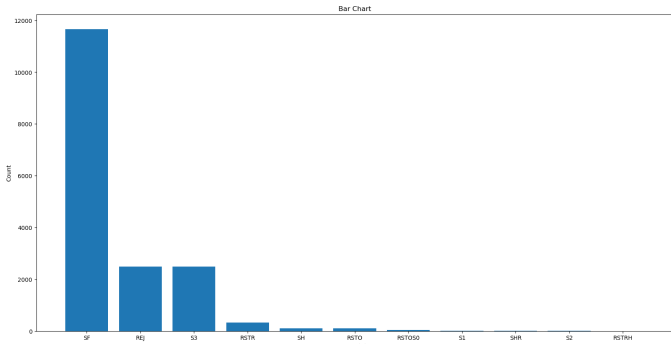


Figure 12: Connection state filtered bar chart



# Label

- [Problem Definition](#)
- [Data Preprocessing and visualization](#)
- [Data Preprocessing](#)
- [Cleaning the Dataset](#)
- [IP address to int](#)**
- [Label Encoding](#)
- [Correlation](#)
- [Model built and prediction](#)
- [Conclusion](#)

- Label is the main feature in our dataset which is used for prediction. this feature says weather the network traffic is malicious or benign.
- Below is the pi chart representation of label feature.

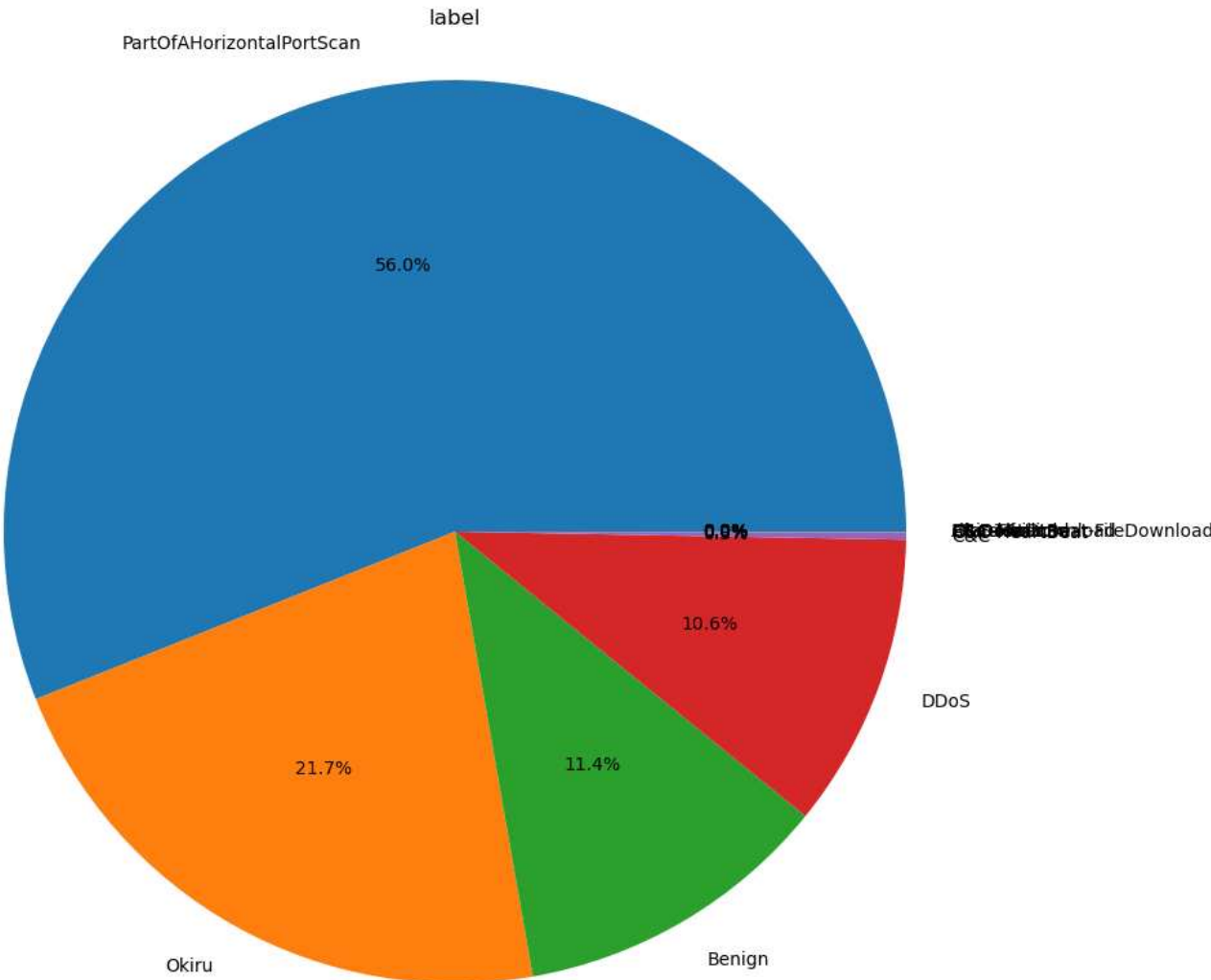


Figure 13: Pi chart representation of label





# Label Visualization

- Problem Definition
- Data Preprocessing and visualization
  - Data Preprocessing
  - Cleaning the Dataset
  - IP address to int
  - Label Encoding
  - Correlation
- Model built and prediction
- Conclusion

- Label feature have 13 values where the majority is occupied by 4 values as shown above. Removing those four values in below image we represent the rest of the values in bar chart.

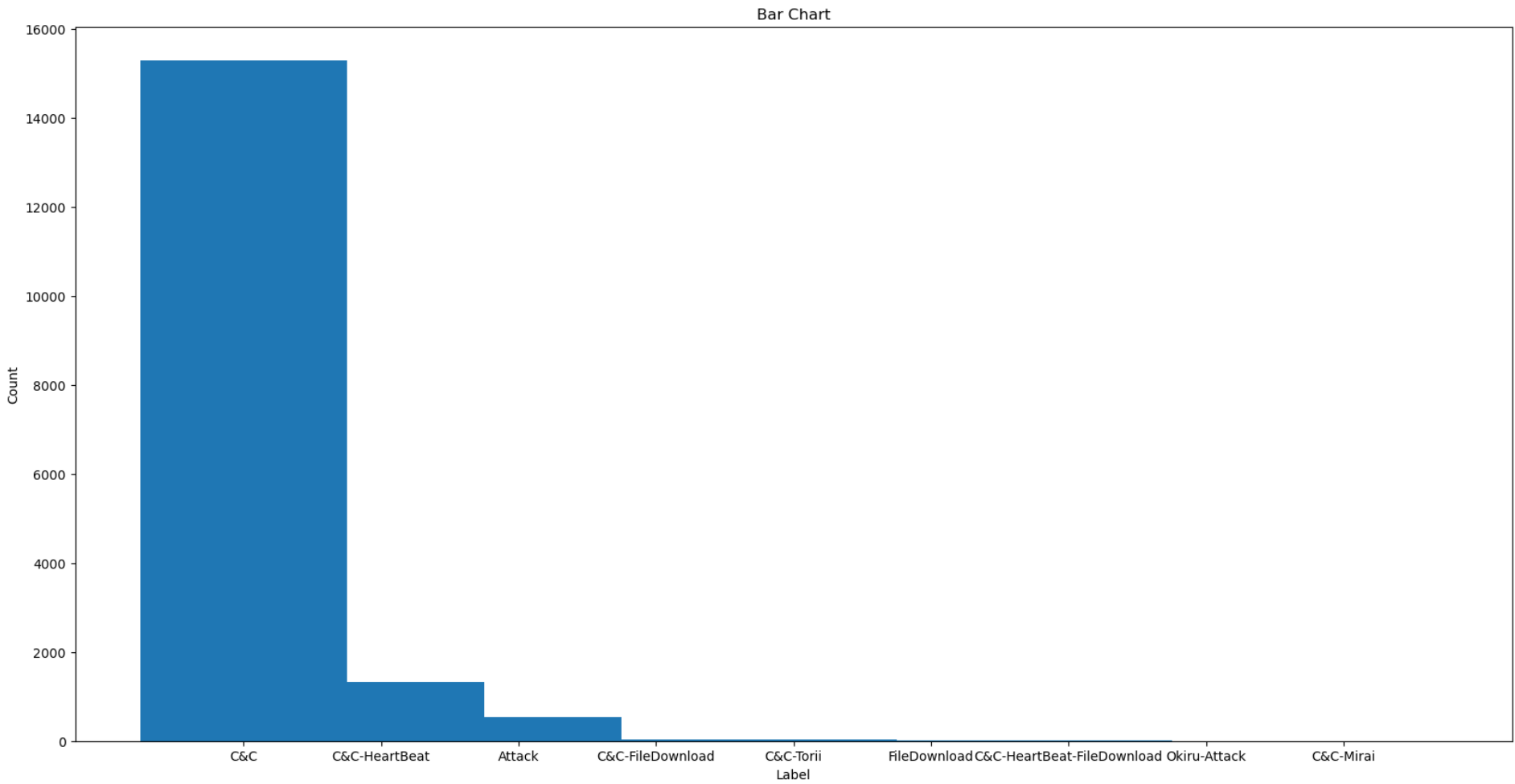


Figure 14: Filtered label representation



# Label Encoding

- Problem Definition
- Data Preprocessing and visualization
- Data Preprocessing
  - Cleaning the Dataset
  - IP address to int
  - Label Encoding**
  - Correlation
- Model built and prediction
- Conclusion

- Categorical data are those that are represented by labels or categories, such as gender, color, or type of product. Machine learning algorithms require numerical input, so numerical encoding is necessary to transform categorical data into numerical values.
- We use label encoding to convert 'proto', 'service', 'conn\_state', 'label' variables to numerical values.

Label Encoding

```
In [49]: pmap = {'tcp':0, 'udp':1, 'icmp':2}
         dd['proto']=dd['proto'].map(pmap)

In [50]: pmap = {'nil':0, 'dns':1, 'irc':2, 'http':3, 'dhcp':4, 'ssl':5, 'ssh':6}
         dd['service']=dd['service'].map(pmap)

In [51]: pmap = {'S0':0, 'OTH':1, 'SF':2, 'REJ':3, 'S3':4, 'RSTR':5, 'SH':6, 'RSTO':7, 'RSTOS0':8, 'S1':9, 'SHR':10, 'S2':11, 'RSTRH':12}
         dd['conn_state']=dd['conn_state'].map(pmap)

In [52]: pmap = {'PartOfAHorizontalPortScan':0, 'Okiru':1, 'Benign':2, 'DDoS':3, 'C&C':4, 'C&C-HeartBeat':5, 'Attack':6, 'C&C-FileDownload':7, 'C&C-HeartBeat':8, 'C&C-FileDownload':9, 'C&C-HeartBeat':10, 'C&C-FileDownload':11, 'C&C-HeartBeat':12, 'C&C-FileDownload':13}
         dd['label']=dd['label'].map(pmap)
```

Figure 15: Label encoding



# Correlation

- Problem Definition
- Data Preprocessing and visualization
  - Data Preprocessing
    - Cleaning the Dataset
    - IP address to int
    - Label Encoding
  - Correlation
- Model built and prediction
- Conclusion

- Correlation is a statistical measure that describes the relationship between two variables. It is used to determine how strongly and in what way two variables are related to each other. Correlation coefficients range between -1 and +1.
- We are representing correlation using heat map.

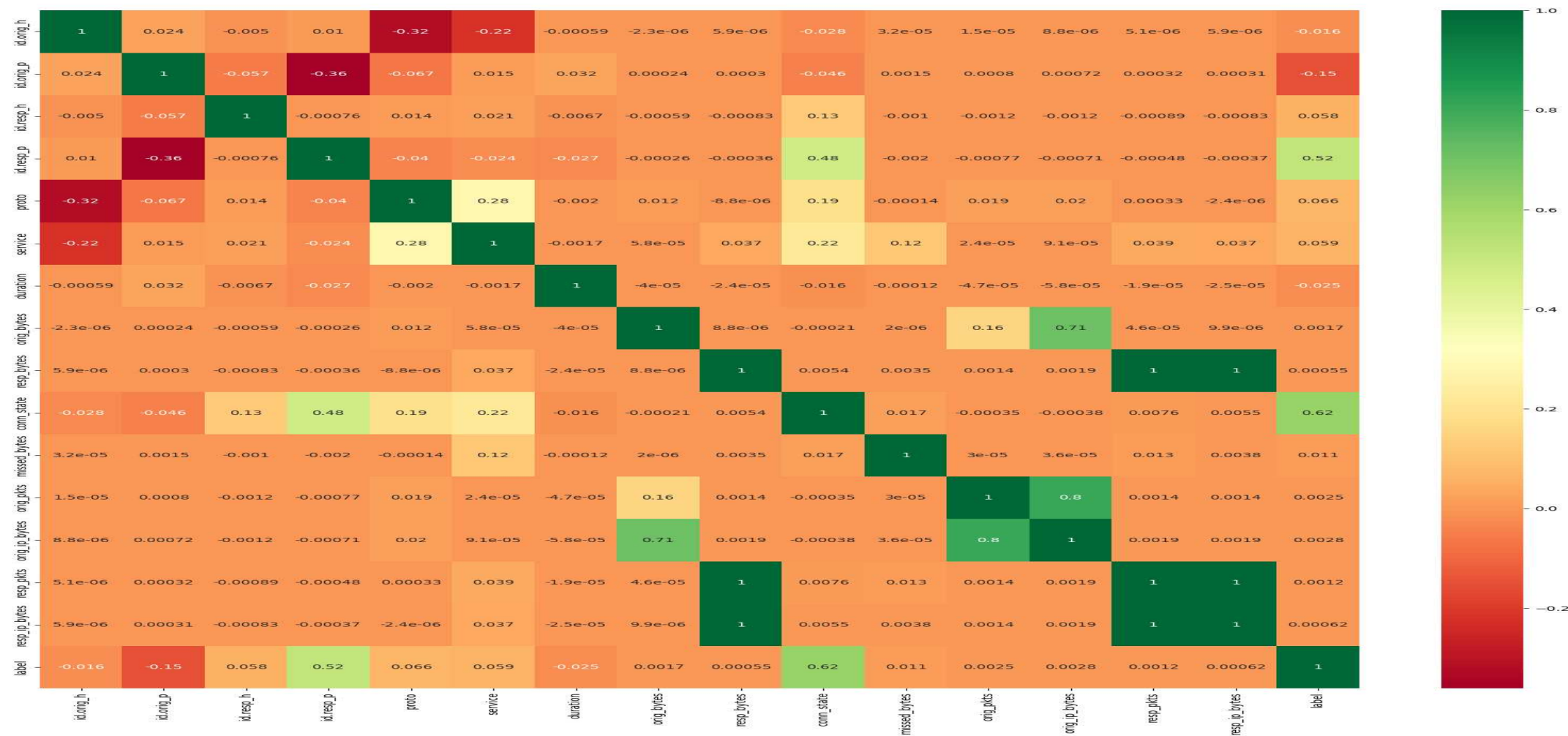


Figure 16: Correlation



# High correlation

- [Problem Definition](#)
- [Data Preprocessing and visualization](#)
- [Data Preprocessing](#)
- [Cleaning the Dataset](#)
- [IP address to int](#)
- [Label Encoding](#)
- [Correlation](#)**
- [Model built and prediction](#)
- [Conclusion](#)

- If variables are highly correlated, it may be a good idea to remove one of the variables to avoid multicollinearity, which can lead to unstable and inaccurate models.
- When we refer the above diagram we can see that some variables are highly correlated. To avoid multicollinearity we are removing 'resp\_bytes', 'resp\_pkts', 'orig\_pkts'.



- [Problem Definition](#)
- [Data Preprocessing and visualization](#)
- [Model built and prediction](#)**
- [Splitting the data](#)
- [Decision Tree Classifier](#)
- [Evaluating the model](#)
- [Conclusion](#)

# Model built and prediction





# Splitting the data

- [Problem Definition](#)
- [Data Preprocessing and visualization](#)
- [Model built and prediction](#)
- [Splitting the data](#)**
- [Decision Tree Classifier](#)
- [Evaluating the model](#)
- [Conclusion](#)

- The most common way to split a dataset is to divide it into two subsets: a training set and a testing set. The training set is used to train the model, and the testing set is used to evaluate its performance.
- Training set is of 80% and test is of 20%.

```
In [56]: X = dd.drop(['label'], axis=1)
          y = dd[['label']]
```

Figure 17: Splitting the value to train and test set



# Decision Tree Classifier

[Problem Definition](#)

[Data Preprocessing and visualization](#)

[Model built and prediction](#)

[Splitting the data](#)

[Decision Tree Classifier](#)

[Evaluating the model](#)

[Conclusion](#)

- We used Decision Tree Classifier which is the supervised learning algorithm.
- This prints out important feature with its values

```
In [93]: # Build the decision tree
df = DecisionTreeClassifier()
df.fit(X_train, y_train)

# Evaluate the decision tree
y_pred = df.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

importances = df.feature_importances_

feature_names = list(X.columns)

feature_importances = sorted(zip(feature_names, importances), key=lambda x: x[1], reverse=True)

for feature, importance in feature_importances:
    print(f"{feature}: {importance}")
```

Figure 18: Decision tree classifier



**TULIP**

Team for Universal Learning and Intelligent Processing





# Important Features used

- [Problem Definition](#)
- [Data Preprocessing and visualization](#)
- [Model built and prediction](#)
- [Splitting the data](#)
- [Decision Tree Classifier](#)**
- [Evaluating the model](#)
- [Conclusion](#)

- Calculates the feature importances of the model
- Sorts the features by importance in descending order
- Prints each feature and its importance

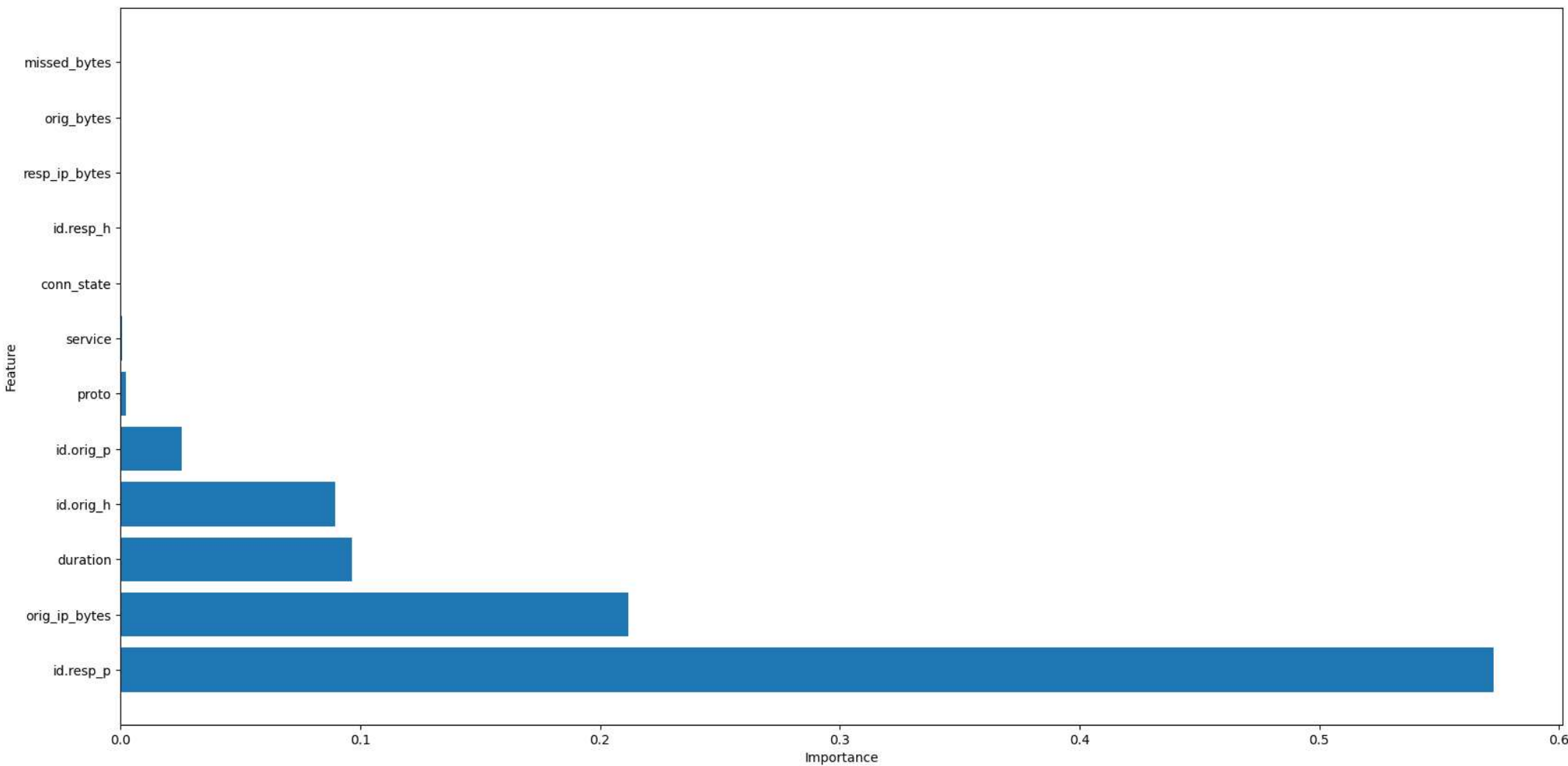


Figure 19: Important features used

# Evaluating the model

Problem Definition
Data Preprocessing and visualization
Model built and prediction
Splitting the data
Decision Tree Classifier
Evaluating the model
Conclusion

```
In [99]: mse = mean_squared_error(y_test, y_pred)
print("MSE:", mse)
```

MSE: 8.434457238542162e-05

```
In [100]: rmse = mean_squared_error(y_test, y_pred, squared=False)
print("RMSE:", rmse)
```

RMSE: 0.009183930116536254

```
In [101]: mae = mean_absolute_error(y_test, y_pred)
print("MAE:", mae)
```

MAE: 2.6461042316995017e-05

```
In [103]: accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Accuracy: 0.9999875963864139

Figure 20: Accuracy obtained





# Evaluating the model

- Problem Definition
- Data Preprocessing and visualization
- Model built and prediction
- Splitting the data
- Decision Tree Classifier
- Evaluating the model
- Conclusion

- Confusion matrix is used to evaluate the performance of a classification model. It compares the actual and predicted target variables and counts the number of correct and incorrect predictions

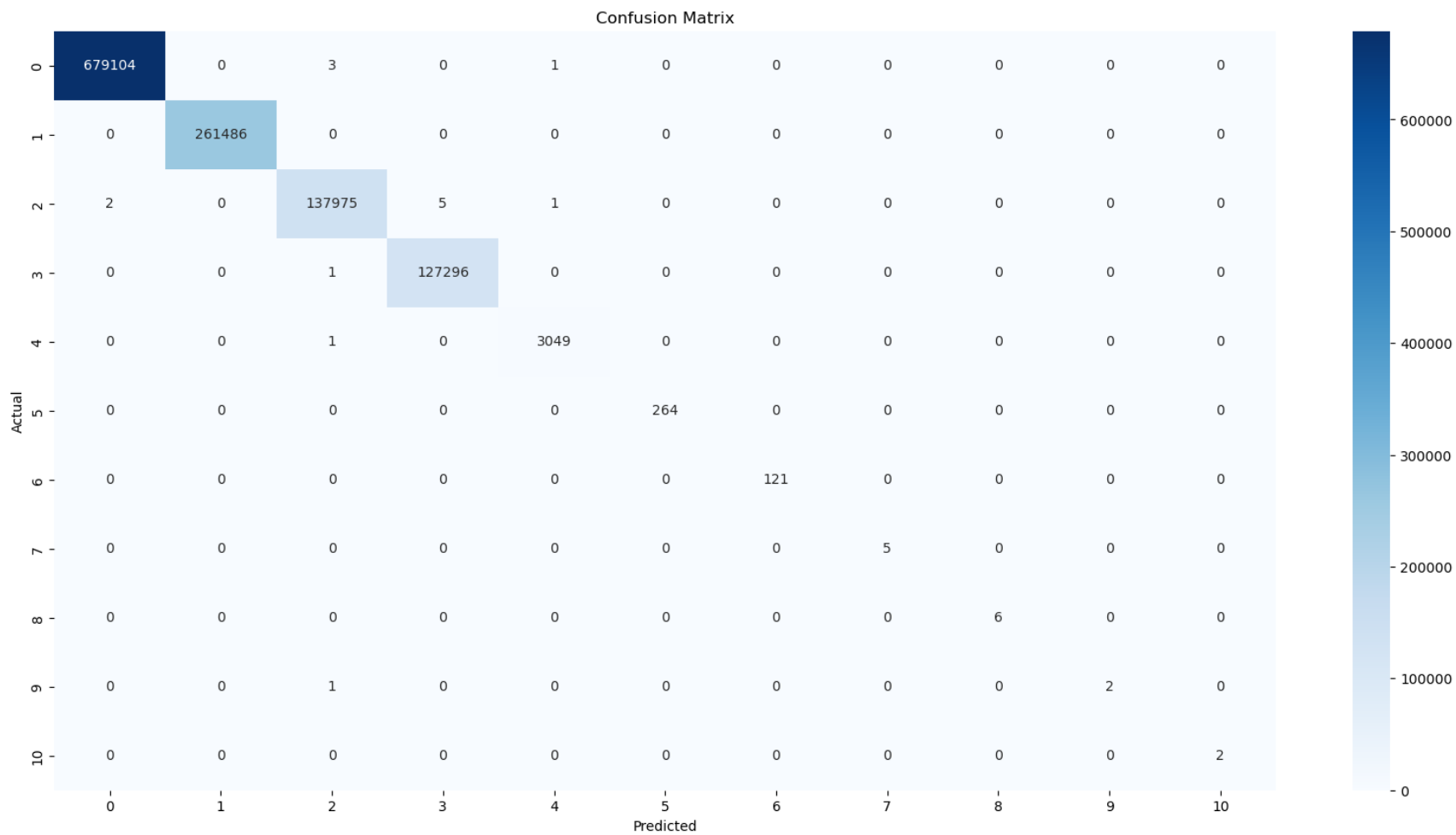


Figure 21: Evaluation result



- [Problem Definition](#)
- [Data Preprocessing and visualization](#)
- [Model built and prediction](#)
- [Conclusion](#)

# Conclusion



# Conclusion

[Problem Definition](#)

[Data Preprocessing and visualization](#)

[Model built and prediction](#)

[Conclusion](#)

- For this Flip00 task I have taken dataset from Kaggle. For which data pre-processing, data cleaning, splitted the data for train and test, finally applied Decision Tree Classifier algorithm.
- Additionally used originator IP address (id.orig\_h) and respondent IP address (id.resp\_h) by converting them to integer form.
- This prediction model helps to effectively predict the Labels of traffic.



**TULIP**

*Team for Universal Learning and Intelligent Processing*



# Questions?

- [Problem Definition](#)
- [Data Preprocessing and visualization](#)
- [Model built and prediction](#)
- [Conclusion](#)



## Contact Information

Spoorthy Reddy Jarugu  
Vellore Institute of Technology  
Vellore, India

