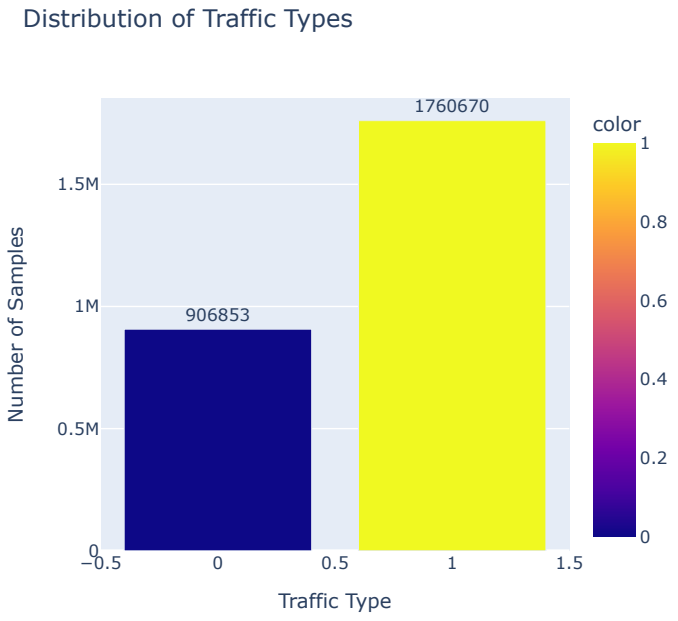
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=T
```

```
import pandas as pd


df = pd.read_csv('/content/drive/MyDrive/Major Project/dataset.csv')


df.columns
```

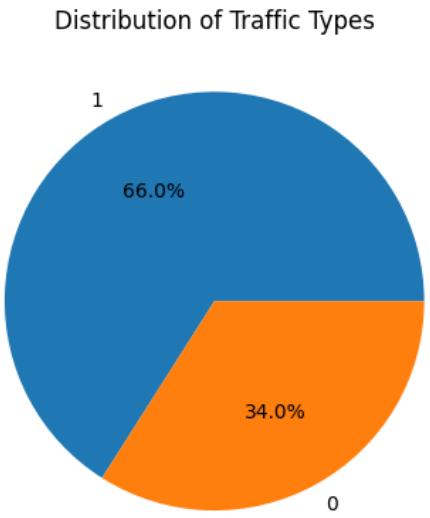
```
Index(['timestamp', 'datapath_id', 'flow_id', 'ip_src', 'tp_src', 'ip_dst',
       'tp_dst', 'ip_proto', 'icmp_code', 'icmp_type', 'flow_duration_sec',
       'flow_duration_nsec', 'idle_timeout', 'hard_timeout', 'flags',
       'packet_count', 'byte_count', 'packet_count_per_second',
       'packet_count_per_nsecond', 'byte_count_per_second',
       'byte_count_per_nsecond', 'label'],
      dtype='object')
```

```
import pandas as pd
import plotly.express as px


# Count number of normal and DDOS samples
count = df['label'].value_counts()

# Create bar chart
fig = px.bar(x=count.index, y=count.values, color=count.index, text=count.values,
             labels={'x': 'Traffic Type', 'y': 'Number of Samples'},
             title='Distribution of Traffic Types',
             )
fig.update_traces(textposition='outside')
fig.update_layout(width=500, height=500)
fig.show()
```

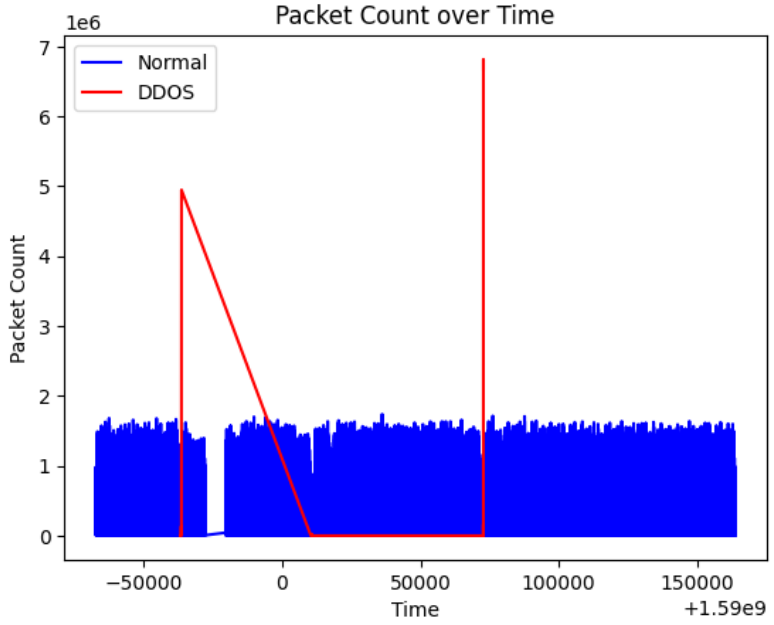



```
import pandas as pd
import matplotlib.pyplot as plt


# Count number of normal and DDOS samples
count = df['label'].value_counts()
```

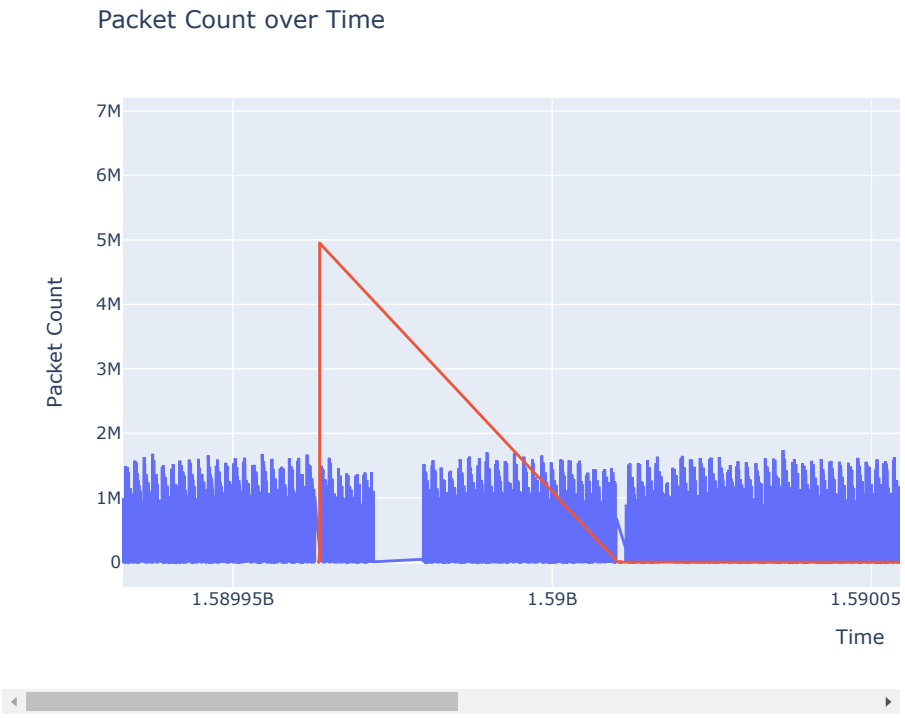
```
# Plot pie chart
plt.pie(count.values, labels=count.index, autopct='%1.1f%%')
plt.title('Distribution of Traffic Types')
plt.show()
```




```
import pandas as pd
import matplotlib.pyplot as plt


# Create subsets for normal and DDOS traffic
normal = df[df['label'] == 0]
ddos = df[df['label'] == 1]

# Group packet count by timestamp for each traffic type
normal_count = normal.groupby('timestamp')['packet_count'].sum()
ddos_count = ddos.groupby('timestamp')['packet_count'].sum()

# Plot line chart
plt.plot(normal_count.index, normal_count.values, c='blue', label='Normal')
plt.plot(ddos_count.index, ddos_count.values, c='red', label='DDOS')
plt.xlabel('Time')
plt.ylabel('Packet Count')
plt.title('Packet Count over Time')
plt.legend(loc='upper left')
plt.show()
```



```
import pandas as pd
import plotly.graph_objects as go


# Create subsets for normal and DDOS traffic
normal = df[df['label'] == 0]
ddos = df[df['label'] == 1]

# Group packet count by timestamp for each traffic type
normal_count = normal.groupby('timestamp')['packet_count'].sum()
ddos_count = ddos.groupby('timestamp')['packet_count'].sum()

# Create line chart
fig = go.Figure()
fig.add_trace(go.Scatter(x=normal_count.index, y=normal_count.values, mode='lines', name='Normal'))
fig.add_trace(go.Scatter(x=ddos_count.index, y=ddos_count.values, mode='lines', name='DDOS'))
fig.update_layout(title='Packet Count over Time', xaxis_title='Time', yaxis_title='Packet Count')
fig.show()
```

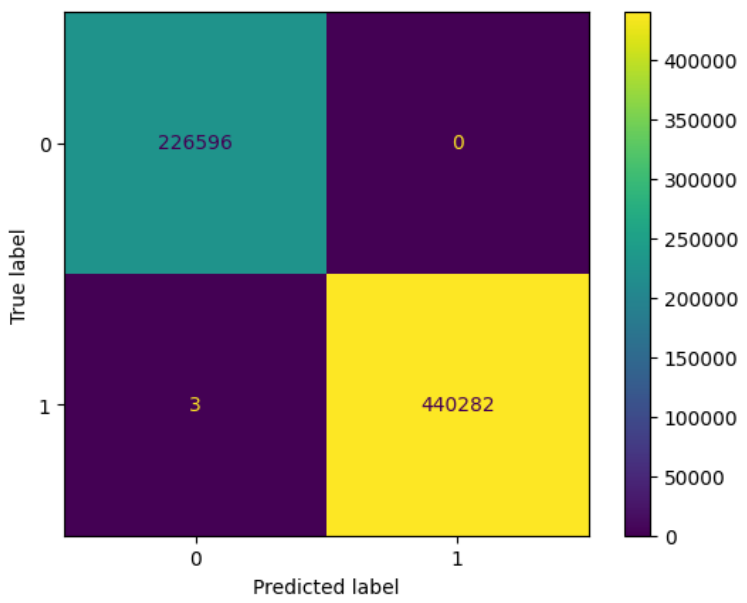



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression


flow_dataset = pd.read_csv('/content/drive/MyDrive/Major Project/dataset.csv')
```

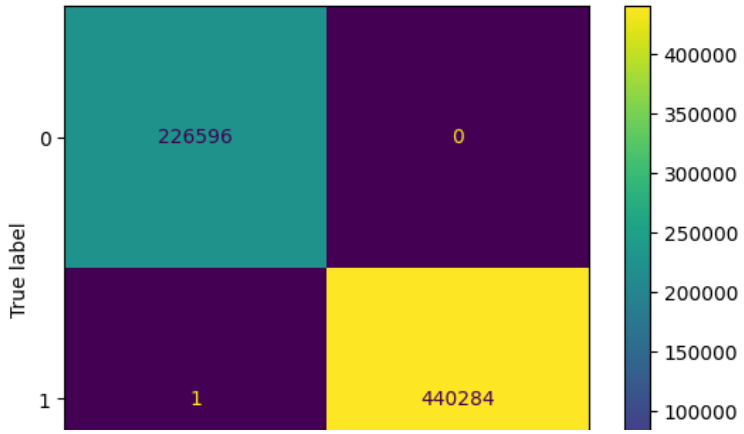
```
flow_dataset.iloc[:, 2] = flow_dataset.iloc[:, 2].str.replace('.', '')
flow_dataset.iloc[:, 3] = flow_dataset.iloc[:, 3].str.replace('.', '')
flow_dataset.iloc[:, 5] = flow_dataset.iloc[:, 5].str.replace('.', '')

X_flow = flow_dataset.iloc[:, :-1].values
X_flow = X_flow.astype('float64')
y_flow = flow_dataset.iloc[:, -1].values
X_flow_train, X_flow_test, y_flow_train, y_flow_test = train_test_split(X_flow, y_flow, test_size=0.25, random_state=0)
```

```
<ipython-input-36-42e8f3af7747>:1: FutureWarning:

The default value of regex will change from True to False in a future version. In addition, single character regular expres

<ipython-input-36-42e8f3af7747>:2: FutureWarning:

The default value of regex will change from True to False in a future version. In addition, single character regular expres

<ipython-input-36-42e8f3af7747>:3: FutureWarning:

The default value of regex will change from True to False in a future version. In addition, single character regular expres
```

```
DT_classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
DT_flow_model = DT_classifier.fit(X_flow_train, y_flow_train)


y_flow_pred = DT_flow_model.predict(X_flow_test)
cm = confusion_matrix(y_flow_test, y_flow_pred)


disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=DT_classifier.classes_)
disp.plot()
plt.show()
```




```
RF_classifier = RandomForestClassifier(n_estimators=10, criterion="entropy", random_state=42)
RF_flow_model = RF_classifier.fit(X_flow_train, y_flow_train)


y_flow_pred = RF_flow_model.predict(X_flow_test)
cm = confusion_matrix(y_flow_test, y_flow_pred)


disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=RF_classifier.classes_)
disp.plot()
plt.show()
```




```
LR_classifier = LogisticRegression(solver='liblinear', random_state=0)
LR_flow_model=LR_classifier.fit(X_flow_train, y_flow_train)
```

```
y_flow_pred = LR_flow_model.predict(X_flow_test)
cm = confusion_matrix(y_flow_test, y_flow_pred)
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=LR_classifier.classes_)
disp.plot()
plt.show()
```