Data Collection

In the data collection I chose the UNSW-NB15 dataset a network traffic dataset this dataset include a blend of modern normal activities and synthetic contemporary attack behaviors including nine attack categories such as Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms it provides a rich set of feature suitable for training and evaluate network intrusion detection systems.

```python
import pandas as pd

training_file_path = 'UNSW_NB15_training-set.parquet'
testing_file_path = 'UNSW_NB15_testing-set.parquet'

# Read the parquet file
training_df = pd.read_parquet(training_file_path)
testing_df = pd.read_parquet(testing_file_path)

# Print the first five rows of the training dataset
print("Training Dataset:")
display(training_df.head())

# Print information about the training dataset
print("\nTraining Dataset Information:")
display(training_df.info())

# Print the first five rows of the testing dataset
print("\nTesting Dataset:")
display(testing_df.head())

# Print information about the testing dataset
print("\nTesting Dataset Information:")
display(testing_df.info())
```

```
Training Dataset:

        dur proto service state  spkts  dpkts  sbytes  dbytes
rate  \
0  0.121478   tcp       -   FIN      6      4     258     172
74.087486
1  0.649902   tcp       -   FIN     14     38     734   42014
78.473373
2  1.623129   tcp       -   FIN      8     16     364   13186
14.170161
3  1.681642   tcp     ftp   FIN     12     12     628     770
13.677108
4  0.449454   tcp       -   FIN     10      6     534     268
33.373825
```

```
          sload  ...  trans_depth  response_body_len  ct_src_dport_ltm  \
0  14158.942383  ...            0                  0                 1   
1   8395.112305  ...            0                  0                 1   
2   1572.271851  ...            0                  0                 1   
3   2740.178955  ...            0                  0                 1   
4   8561.499023  ...            0                  0                 2   

   ct_dst_sport_ltm  is_ftp_login  ct_ftp_cmd  ct_flw_http_mthd  \
0                 1             0           0                 0   
1                 1             0           0                 0   
2                 1             0           0                 0   
3                 1             1           1                 0   
4                 1             0           0                 0   

   is_sm_ips_ports  attack_cat  label  
0                0      Normal      0  
1                0      Normal      0  
2                0      Normal      0  
3                0      Normal      0  
4                0      Normal      0  

[5 rows x 36 columns]


Training Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175341 entries, 0 to 175340
Data columns (total 36 columns):
 #   Column             Non-Null Count   Dtype   
---  ------             --------------   -----   
 0   dur                175341 non-null  float32 
 1   proto              175341 non-null  category
 2   service            175341 non-null  category
 3   state              175341 non-null  category
 4   spkts              175341 non-null  int16   
 5   dpkts              175341 non-null  int16   
 6   sbytes             175341 non-null  int32   
 7   dbytes             175341 non-null  int32   
 8   rate               175341 non-null  float32 
 9   sload              175341 non-null  float32 
 10  dload              175341 non-null  float32 
 11  sloss              175341 non-null  int16   
 12  dloss              175341 non-null  int16   
 13  sinpkt             175341 non-null  float32 
```

```
 14  dinpkt             175341 non-null  float32
 15  sjit               175341 non-null  float32
 16  djit               175341 non-null  float32
 17  swin               175341 non-null  int16
 18  stcpb              175341 non-null  int64
 19  dtcpb              175341 non-null  int64
 20  dwin               175341 non-null  int16
 21  tcprtt             175341 non-null  float32
 22  synack             175341 non-null  float32
 23  ackdat             175341 non-null  float32
 24  smean              175341 non-null  int16
 25  dmean              175341 non-null  int16
 26  trans_depth        175341 non-null  int16
 27  response_body_len  175341 non-null  int32
 28  ct_src_dport_ltm   175341 non-null  int8
 29  ct_dst_sport_ltm   175341 non-null  int8
 30  is_ftp_login       175341 non-null  int8
 31  ct_ftp_cmd         175341 non-null  int8
 32  ct_flw_http_mthd   175341 non-null  int8
 33  is_sm_ips_ports    175341 non-null  int8
 34  attack_cat         175341 non-null  category
 35  label              175341 non-null  int8
dtypes: category(4), float32(11), int16(9), int32(3), int64(2),
int8(7)
memory usage: 17.1 MB

None


Testing Dataset:

        dur proto service state  spkts  dpkts  sbytes  dbytes
rate  \
0  0.000011   udp       -   INT      2      0     496       0
90909.09375
1  0.000008   udp       -   INT      2      0    1762       0
125000.00000
2  0.000005   udp       -   INT      2      0    1068       0
200000.00000
3  0.000006   udp       -   INT      2      0     900       0
166666.65625
4  0.000010   udp       -   INT      2      0    2126       0
100000.00000

         sload  ...  trans_depth  response_body_len  ct_src_dport_ltm
\
0  180363632.0  ...            0                  0                 1

1  881000000.0  ...            0                  0                 1
```

```
2  854400000.0  ...            0                  0                 1

3  600000000.0  ...            0                  0                 2

4  850400000.0  ...            0                  0                 2


   ct_dst_sport_ltm  is_ftp_login  ct_ftp_cmd  ct_flw_http_mthd  \
0                 1             0           0                 0
1                 1             0           0                 0
2                 1             0           0                 0
3                 1             0           0                 0
4                 1             0           0                 0

   is_sm_ips_ports  attack_cat  label
0                0      Normal      0
1                0      Normal      0
2                0      Normal      0
3                0      Normal      0
4                0      Normal      0

[5 rows x 36 columns]


Testing Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82332 entries, 0 to 82331
Data columns (total 36 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   dur                82332 non-null  float32
 1   proto              82332 non-null  category
 2   service            82332 non-null  category
 3   state              82332 non-null  category
 4   spkts              82332 non-null  int16
 5   dpkts              82332 non-null  int16
 6   sbytes             82332 non-null  int32
 7   dbytes             82332 non-null  int32
 8   rate               82332 non-null  float32
 9   sload              82332 non-null  float32
 10  dload              82332 non-null  float32
 11  sloss              82332 non-null  int16
 12  dloss              82332 non-null  int16
 13  sinpkt             82332 non-null  float32
 14  dinpkt             82332 non-null  float32
 15  sjit               82332 non-null  float32
 16  djit               82332 non-null  float32
 17  swin               82332 non-null  int16
 18  stcpb              82332 non-null  int64
 19  dtcpb              82332 non-null  int64
```

```
 20  dwin               82332 non-null  int16
 21  tcprtt             82332 non-null  float32
 22  synack             82332 non-null  float32
 23  ackdat             82332 non-null  float32
 24  smean              82332 non-null  int16
 25  dmean              82332 non-null  int16
 26  trans_depth        82332 non-null  int16
 27  response_body_len  82332 non-null  int32
 28  ct_src_dport_ltm   82332 non-null  int8
 29  ct_dst_sport_ltm   82332 non-null  int8
 30  is_ftp_login       82332 non-null  int8
 31  ct_ftp_cmd         82332 non-null  int8
 32  ct_flw_http_mthd   82332 non-null  int8
 33  is_sm_ips_ports    82332 non-null  int8
 34  attack_cat         82332 non-null  category
 35  label              82332 non-null  int8
dtypes: category(4), float32(11), int16(9), int32(3), int64(2),
int8(7)
memory usage: 8.0 MB

None
```

Data Preprocessing:

- Clean the dataset by handling missing values, removing duplicates and normalizing the data.
- Perform feature engineering to create new features that may improve model performance.

Handle Missing Values

```
# Handle missing values for numeric and non numeric columns separately

# Numeric columns
numeric_cols = training_df.select_dtypes(include=['float64',
'int64']).columns

# Fill numeric columns with mean
training_df[numeric_cols] =
training_df[numeric_cols].fillna(training_df[numeric_cols].mean())
testing_df[numeric_cols] =
testing_df[numeric_cols].fillna(testing_df[numeric_cols].mean())

# Non numeric columns
non_numeric_cols = training_df.select_dtypes(exclude=['float64',
'int64']).columns

# Fill non numeric columns with mode most frequent value
```

```
for col in non_numeric_cols:
    training_df[col] = training_df[col].fillna(training_df[col].mode()
[0])
    testing_df[col] = testing_df[col].fillna(testing_df[col].mode()
[0])

# Print there are no missing values
print("Missing values in Training Data after handling:")
print(training_df.isnull().sum())

print("\nMissing values in Testing Data after handling:")
print(testing_df.isnull().sum())
```

```
Missing values in Training Data after handling:
dur                  0
proto                0
service              0
state                0
spkts                0
dpkts                0
sbytes               0
dbytes               0
rate                 0
sload                0
dload                0
sloss                0
dloss                0
sinpkt               0
dinpkt               0
sjit                 0
djit                 0
swin                 0
stcpb                0
dtcpb                0
dwin                 0
tcprtt               0
synack               0
ackdat               0
smean                0
dmean                0
trans_depth          0
response_body_len    0
ct_src_dport_ltm     0
ct_dst_sport_ltm     0
is_ftp_login         0
ct_ftp_cmd           0
ct_flw_http_mthd     0
is_sm_ips_ports      0
attack_cat           0
label                0
```

```
dtype: int64

Missing values in Testing Data after handling:
dur                  0
proto                0
service              0
state                0
spkts                0
dpkts                0
sbytes               0
dbytes               0
rate                 0
sload                0
dload                0
sloss                0
dloss                0
sinpkt               0
dinpkt               0
sjit                 0
djit                 0
swin                 0
stcpb                0
dtcpb                0
dwin                 0
tcprtt               0
synack               0
ackdat               0
smean                0
dmean                0
trans_depth          0
response_body_len    0
ct_src_dport_ltm     0
ct_dst_sport_ltm     0
is_ftp_login         0
ct_ftp_cmd           0
ct_flw_http_mthd     0
is_sm_ips_ports      0
attack_cat           0
label                0
dtype: int64
```

Remove Duplicates

```
# Remove duplicates in the datasets
training_df = training_df.drop_duplicates()
testing_df = testing_df.drop_duplicates()
```

Normalize the Data

```
from sklearn.preprocessing import MinMaxScaler

# Select numerical columns for normalization
numerical_columns = training_df.select_dtypes(include=['float64',
'int64']).columns

# Apply MinMaxScaler
scaler = MinMaxScaler()

# Scale the numerical column only
scaled_training_data = pd.DataFrame(
    scaler.fit_transform(training_df[numerical_columns]),
    columns=numerical_columns,
    index=training_df.index
)

scaled_testing_data = pd.DataFrame(
    scaler.transform(testing_df[numerical_columns]),
    columns=numerical_columns,
    index=testing_df.index
)

# Replace numerical columns in the original DataFrame with scaled data
training_df[numerical_columns] = scaled_training_data
testing_df[numerical_columns] = scaled_testing_data
```

Feature Engineering

Ratio of Source Packets to Destination Packets

```
# Adding a new feature Source to Destination Packet Ratio
training_df['pkt_ratio'] = training_df['spkts'] /
(training_df['dpkts'] + 1)
testing_df['pkt_ratio'] = testing_df['spkts'] / (testing_df['dpkts'] +
1)
```

Total Bytes

```
# Adding a new feature Total Bytes
training_df['total_bytes'] = training_df['sbytes'] +
training_df['dbytes']
testing_df['total_bytes'] = testing_df['sbytes'] +
testing_df['dbytes']
```

Data After Preprocessing

```
# Check the dataset after preprocessing
print("\nTraining Data after preprocessing:")
training_df.head()
```

```
Training Data after preprocessing:

        dur proto service state  spkts  dpkts  sbytes  dbytes
rate  \
0  0.121478   tcp       -   FIN      6      4     258     172
74.087486
1  0.649902   tcp       -   FIN     14     38     734   42014
78.473373
2  1.623129   tcp       -   FIN      8     16     364   13186
14.170161
3  1.681642   tcp     ftp   FIN     12     12     628     770
13.677108
4  0.449454   tcp       -   FIN     10      6     534     268
33.373825

          sload  ...  ct_src_dport_ltm  ct_dst_sport_ltm  is_ftp_login
\
0  14158.942383  ...                 1                 1             0

1   8395.112305  ...                 1                 1             0

2   1572.271851  ...                 1                 1             0

3   2740.178955  ...                 1                 1             1

4   8561.499023  ...                 2                 1             0


   ct_ftp_cmd  ct_flw_http_mthd  is_sm_ips_ports  attack_cat  label  \
0           0                 0                0      Normal      0
1           0                 0                0      Normal      0
2           0                 0                0      Normal      0
3           1                 0                0      Normal      0
4           0                 0                0      Normal      0

   pkt_ratio  total_bytes
0   1.200000          430
1   0.358974        42748
2   0.470588        13550
3   0.923077         1398
4   1.428571          802

[5 rows x 38 columns]
```

```python
print("\nTesting Data after preprocessing:")
testing_df.head()
```

```
Testing Data after preprocessing:

        dur proto service state  spkts  dpkts  sbytes  dbytes
rate  \
0  0.000011   udp       -   INT      2      0     496       0
90909.09375
1  0.000008   udp       -   INT      2      0    1762       0
125000.00000
2  0.000005   udp       -   INT      2      0    1068       0
200000.00000
3  0.000006   udp       -   INT      2      0     900       0
166666.65625
4  0.000010   udp       -   INT      2      0    2126       0
100000.00000

         sload  ...  ct_src_dport_ltm  ct_dst_sport_ltm  is_ftp_login
\
0  180363632.0  ...                 1                 1             0

1  881000000.0  ...                 1                 1             0

2  854400000.0  ...                 1                 1             0

3  600000000.0  ...                 2                 1             0

4  850400000.0  ...                 2                 1             0


   ct_ftp_cmd  ct_flw_http_mthd  is_sm_ips_ports  attack_cat  label  \
0           0                 0                0      Normal      0
1           0                 0                0      Normal      0
2           0                 0                0      Normal      0
3           0                 0                0      Normal      0
4           0                 0                0      Normal      0

   pkt_ratio  total_bytes
0        2.0          496
1        2.0         1762
2        2.0         1068
3        2.0          900
4        2.0         2126

[5 rows x 38 columns]
```

Session 3 EDA

- Conduct EDA to understand the dataset identify patterns and visualize the data.
- Used advanced visualization techniques such as heatmaps pair plots and correlation matrices

Import Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# To display plots inline
%matplotlib inline
```

Dataset Overview

```
# Display basic info about the dataset
print("Dataset Overview:")
print(training_df.info())
print("\nBasic Statistics:")
print(training_df.describe())
```

```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
Index: 96822 entries, 0 to 175337
Data columns (total 38 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   dur                96822 non-null  float32
 1   proto              96822 non-null  category
 2   service            96822 non-null  category
 3   state              96822 non-null  category
 4   spkts              96822 non-null  int16
 5   dpkts              96822 non-null  int16
 6   sbytes             96822 non-null  int32
 7   dbytes             96822 non-null  int32
 8   rate               96822 non-null  float32
 9   sload              96822 non-null  float32
 10  dload              96822 non-null  float32
 11  sloss              96822 non-null  int16
 12  dloss              96822 non-null  int16
 13  sinpkt             96822 non-null  float32
 14  dinpkt             96822 non-null  float32
 15  sjit               96822 non-null  float32
 16  djit               96822 non-null  float32
 17  swin               96822 non-null  int16
 18  stcpb              96822 non-null  float64
```

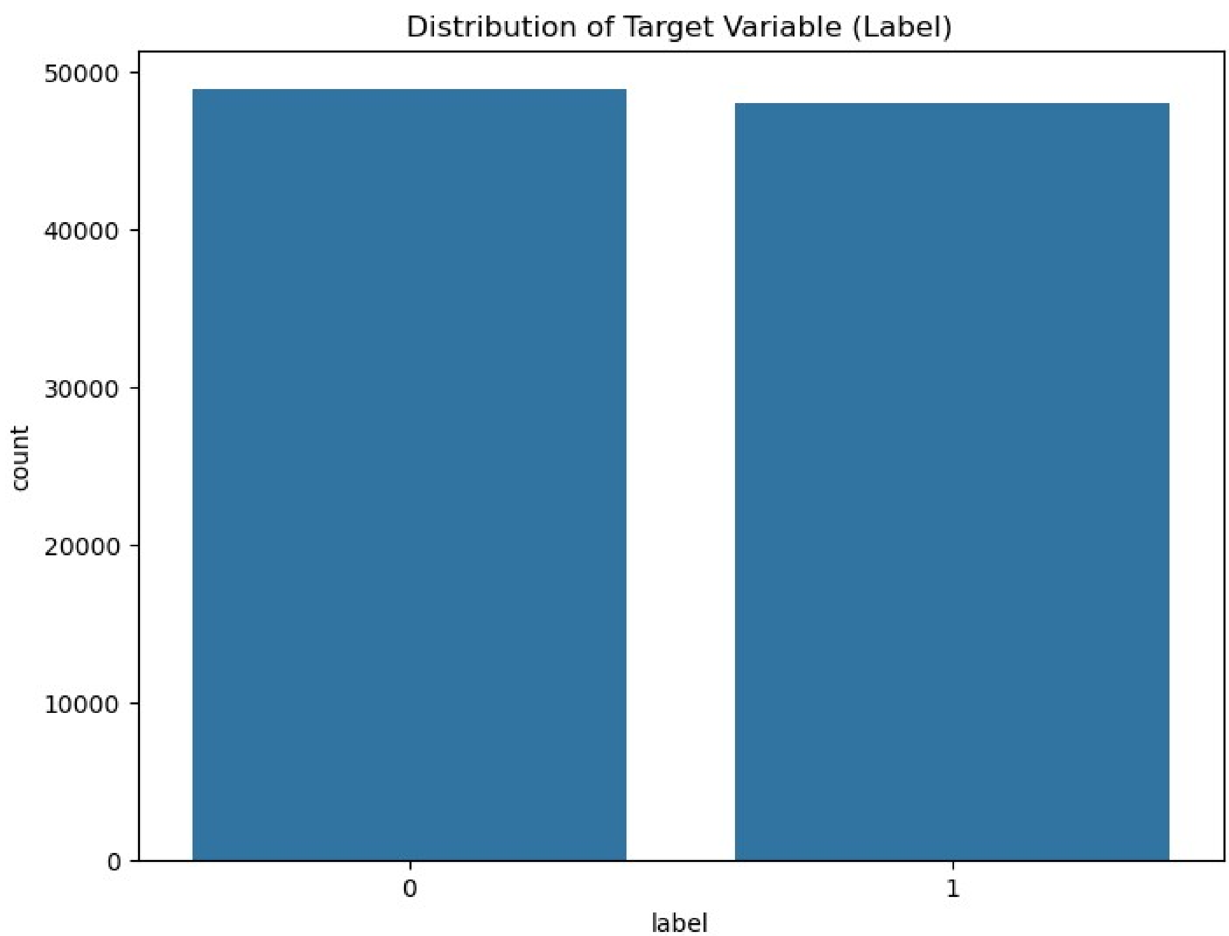
```
 19  dtcpb              96822 non-null  float64
 20  dwin               96822 non-null  int16
 21  tcprtt             96822 non-null  float32
 22  synack             96822 non-null  float32
 23  ackdat             96822 non-null  float32
 24  smean              96822 non-null  int16
 25  dmean              96822 non-null  int16
 26  trans_depth        96822 non-null  int16
 27  response_body_len  96822 non-null  int32
 28  ct_src_dport_ltm   96822 non-null  int8
 29  ct_dst_sport_ltm   96822 non-null  int8
 30  is_ftp_login       96822 non-null  int8
 31  ct_ftp_cmd         96822 non-null  int8
 32  ct_flw_http_mthd   96822 non-null  int8
 33  is_sm_ips_ports    96822 non-null  int8
 34  attack_cat         96822 non-null  category
 35  label              96822 non-null  int8
 36  pkt_ratio          96822 non-null  float64
 37  total_bytes        96822 non-null  int32
dtypes: category(4), float32(11), float64(3), int16(9), int32(4),
int8(7)
memory usage: 11.3 MB
None

Basic Statistics:
                dur         spkts         dpkts        sbytes
dbytes  \
count  96822.000000  96822.000000  96822.000000  9.682200e+04
9.682200e+04
mean       1.492981     33.343155     33.400498  1.534994e+04
2.640205e+04
std        5.765602    182.663081    143.654676  2.348021e+05
1.882358e+05
min        0.000000      1.000000      0.000000  2.800000e+01
0.000000e+00
25%        0.017723      8.000000      6.000000  5.640000e+02
2.680000e+02
50%        0.444509     10.000000      8.000000  1.048000e+03
6.980000e+02
75%        1.019829     24.000000     22.000000  2.958000e+03
3.674000e+03
max       59.999989   9616.000000  10974.000000  1.296523e+07
1.465555e+07

                 rate         sload         dload         sloss
dloss  \
count    96822.000000  9.682200e+04  9.682200e+04  96822.000000
96822.000000
mean     21616.699219  3.673790e+07  1.201664e+06      8.779327
```

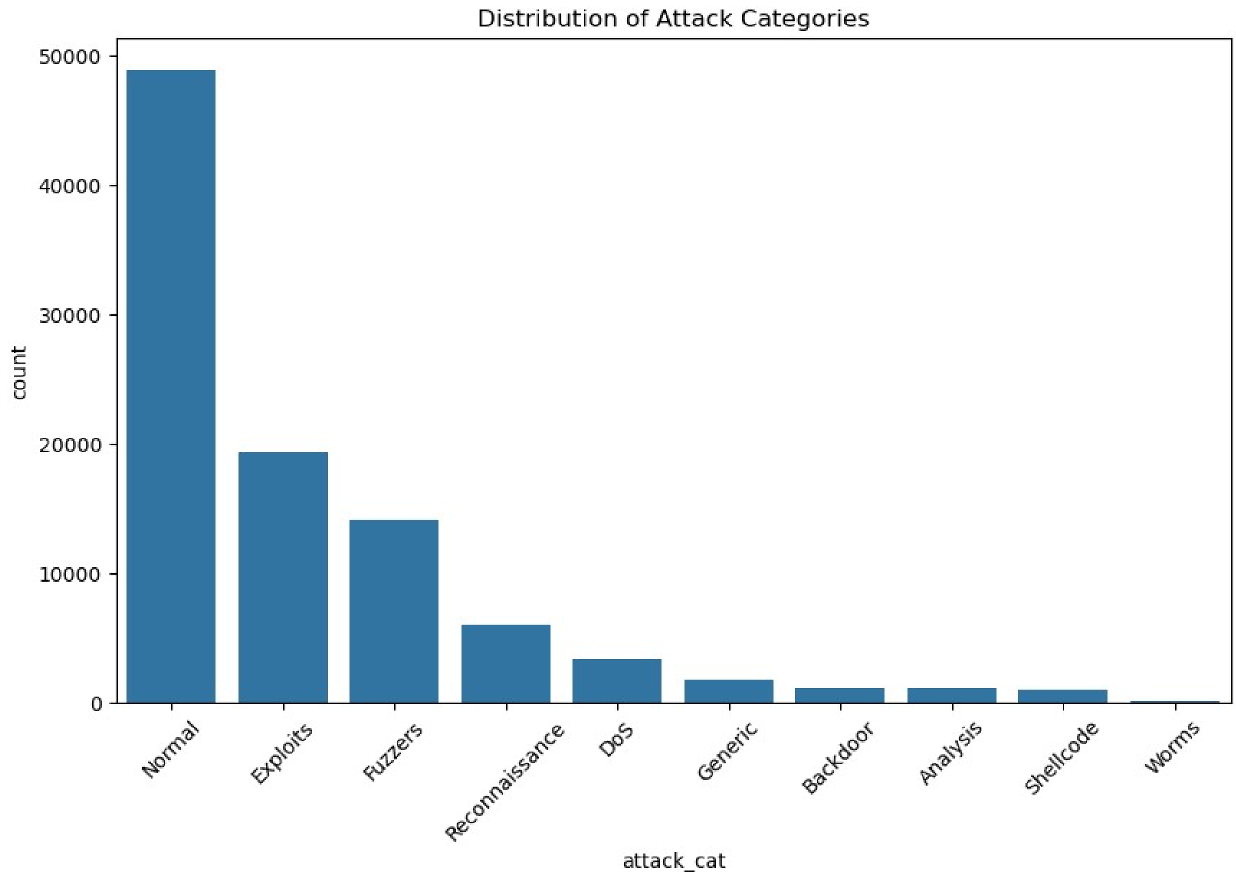
```
12.243819
std      86625.593750  1.964549e+08  3.159119e+06     88.603529
68.913487
min          0.000000  0.000000e+00  0.000000e+00      0.000000
0.000000
25%         24.589354  8.312729e+03  2.972233e+03      2.000000
1.000000
50%         63.841385  3.635026e+04  9.971823e+03      2.000000
2.000000
75%       2832.861084  6.434555e+05  6.136576e+05      7.000000
7.000000
max    1000000.000000  5.988000e+09  2.242273e+07   4803.000000
5484.000000

       ...  response_body_len  ct_src_dport_ltm  ct_dst_sport_ltm  \
count  ...       9.682200e+04      96822.000000      96822.000000
mean   ...       3.824908e+03          1.718721          1.205573
std    ...       7.271763e+04          2.656315          1.282107
min    ...       0.000000e+00          1.000000          1.000000
25%    ...       0.000000e+00          1.000000          1.000000
50%    ...       0.000000e+00          1.000000          1.000000
75%    ...       0.000000e+00          1.000000          1.000000
max    ...       6.558056e+06         51.000000         46.000000

       is_ftp_login    ct_ftp_cmd  ct_flw_http_mthd
is_sm_ips_ports  \
count  96822.000000  96822.000000      96822.000000     96822.000000

mean       0.019200      0.019200          0.200709         0.005205

std        0.139246      0.139246          0.601968         0.071961

min        0.000000      0.000000          0.000000         0.000000

25%        0.000000      0.000000          0.000000         0.000000

50%        0.000000      0.000000          0.000000         0.000000

75%        0.000000      0.000000          0.000000         0.000000

max        4.000000      4.000000         30.000000         1.000000


              label     pkt_ratio   total_bytes
count  96822.000000  96822.000000  9.682200e+04
mean       0.495011      1.668271  4.175199e+04
std        0.499978      6.333095  3.018464e+05
min        0.000000      0.002497  2.800000e+01
25%        0.000000      0.869565  9.180000e+02
50%        0.000000      1.111111  2.176000e+03
```

```
75%        1.000000      1.428571  1.185200e+04
max        1.000000    512.000000  1.472678e+07

[8 rows x 34 columns]
```

Target Variable Distribution

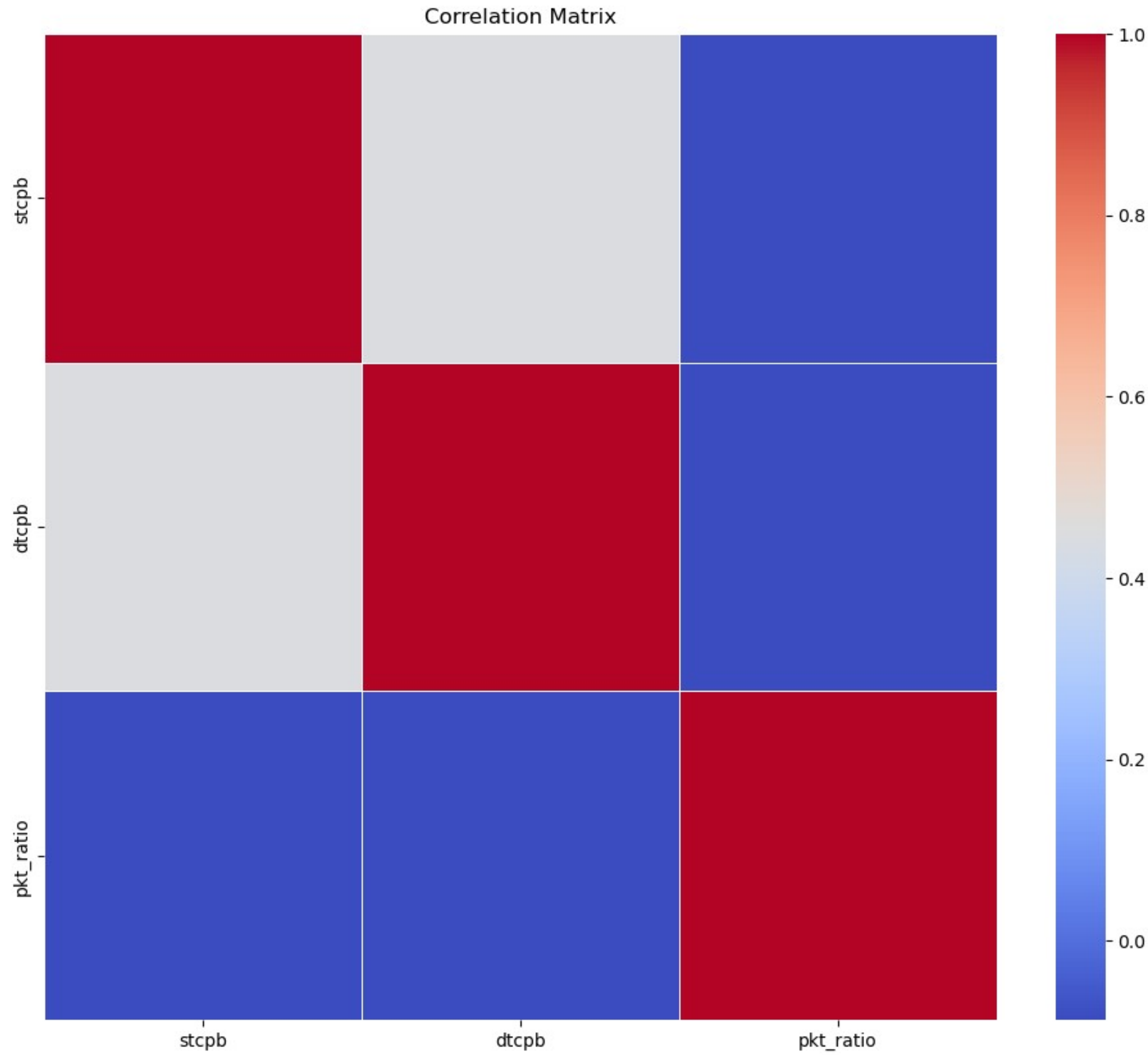
```
# Distribution of the Label Binary Target
plt.figure(figsize=(8, 6))
sns.countplot(x='label', data=training_df)
plt.title('Distribution of Target Variable (Label)')
plt.show()

# Distribution of Attack Categories
plt.figure(figsize=(10, 6))
sns.countplot(x='attack_cat', data=training_df,
order=training_df['attack_cat'].value_counts().index)
plt.xticks(rotation=45)
plt.title('Distribution of Attack Categories')
plt.show()
```





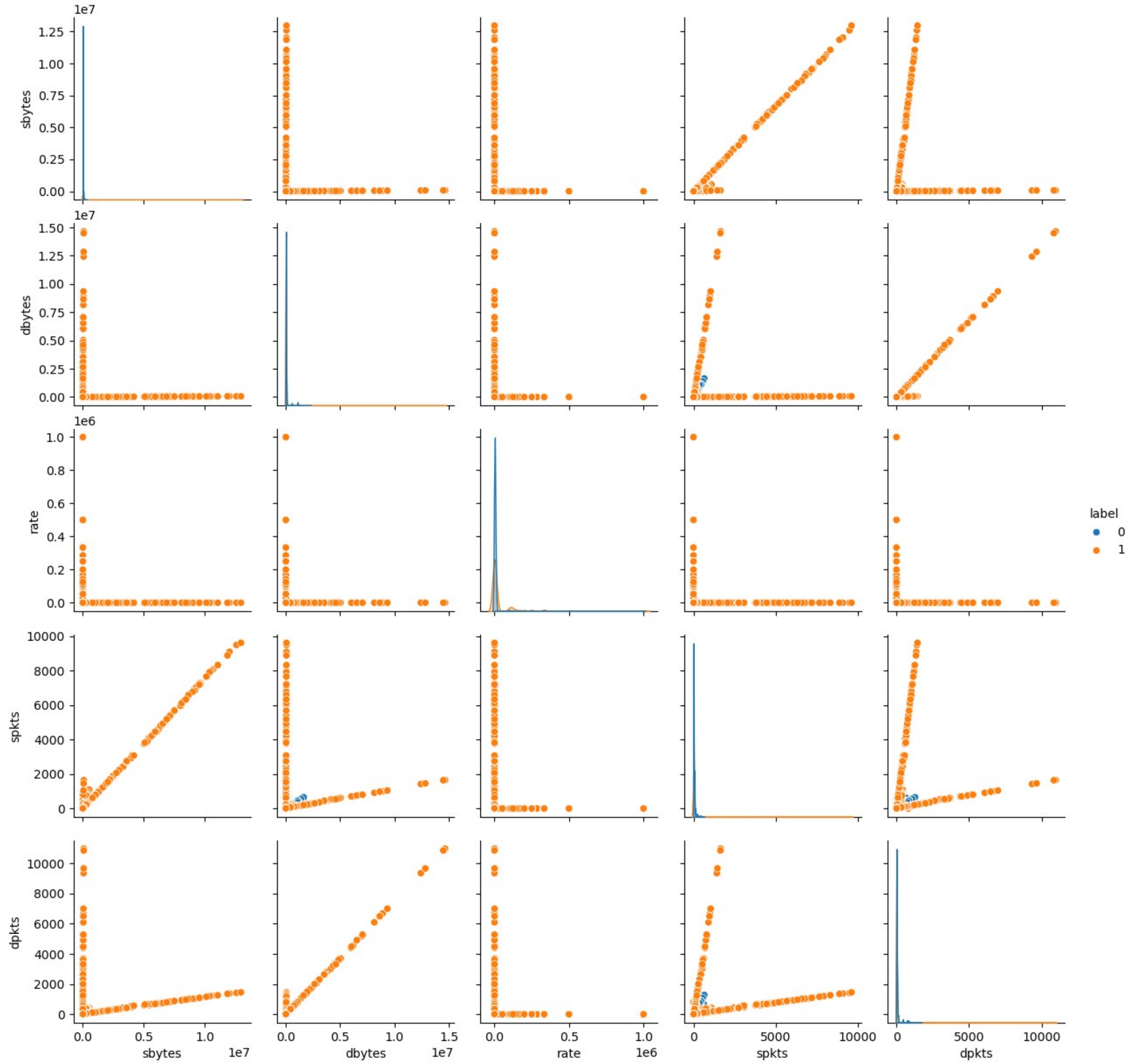
Correlation Heatmap

```
# Select only numerical column
numerical_columns = training_df.select_dtypes(include=['float64',
'int64']).columns
numerical_df = training_df[numerical_columns]

# Calculate the correlation matrix
correlation = numerical_df.corr()

# Plot the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation, annot=False, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



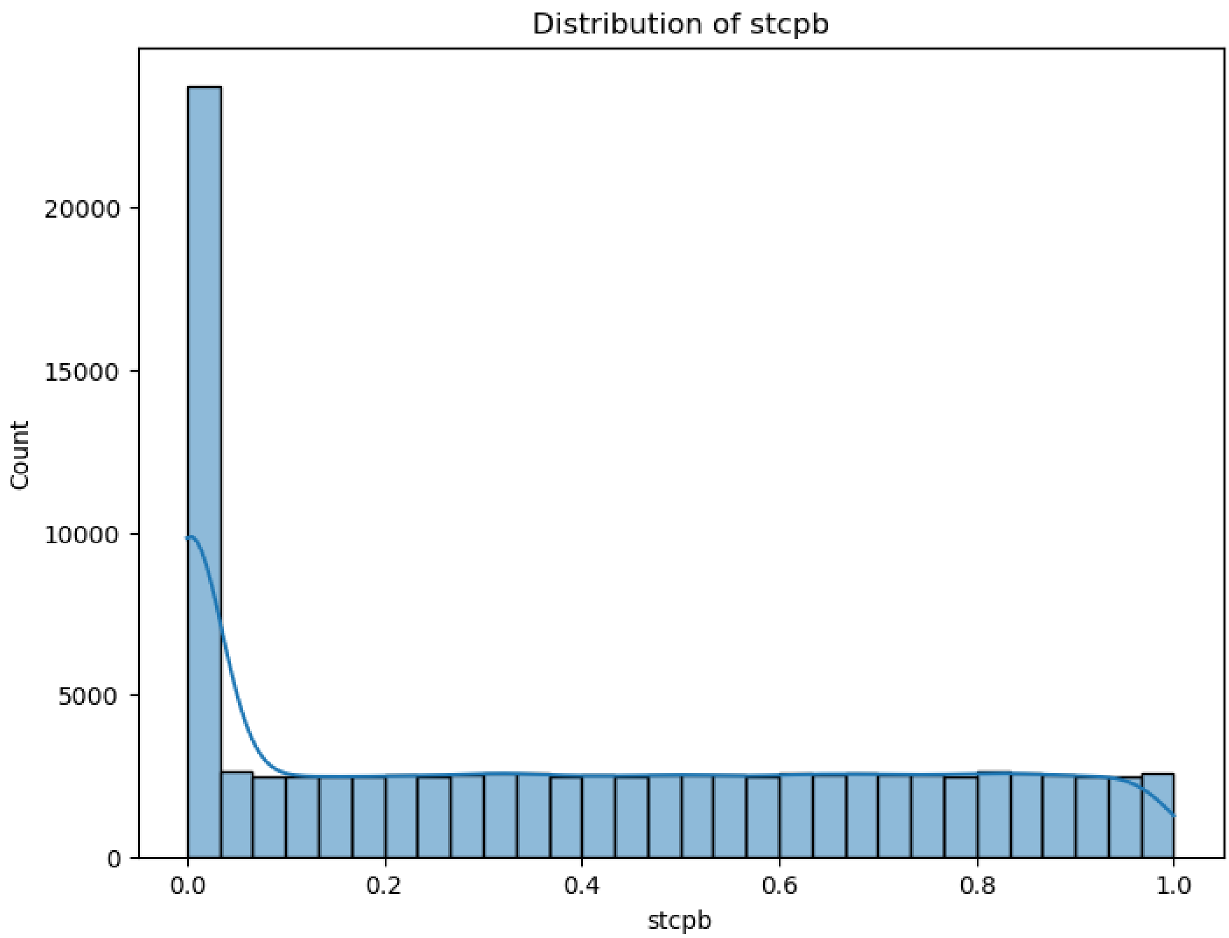
Pair Plot for Selected Features

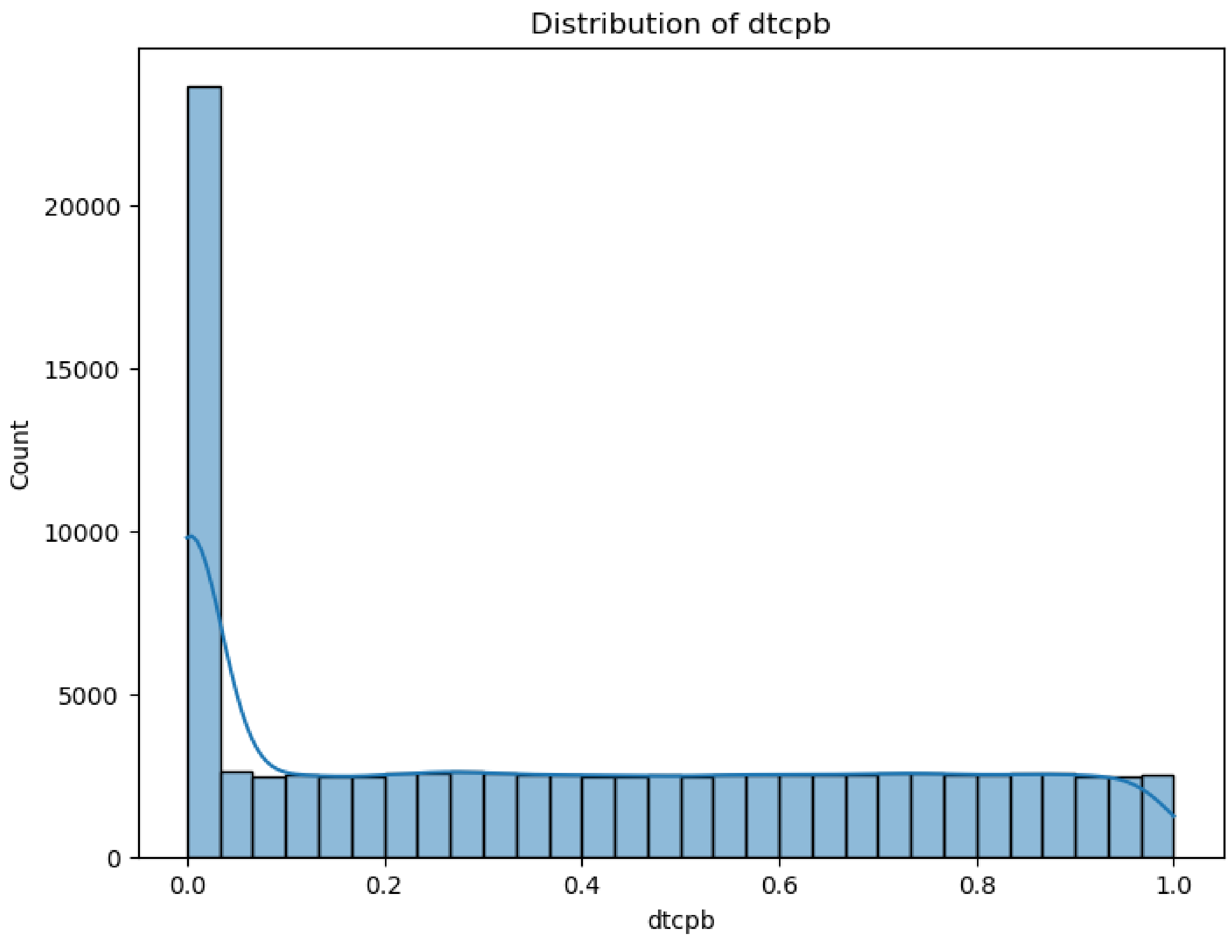
```
# Pair plot for selected numerical columns
sample_columns = ['sbytes', 'dbytes', 'rate', 'spkts', 'dpkts', 
'label']
sns.pairplot(training_df[sample_columns], hue='label', 
diag_kind='kde')
plt.show()
```



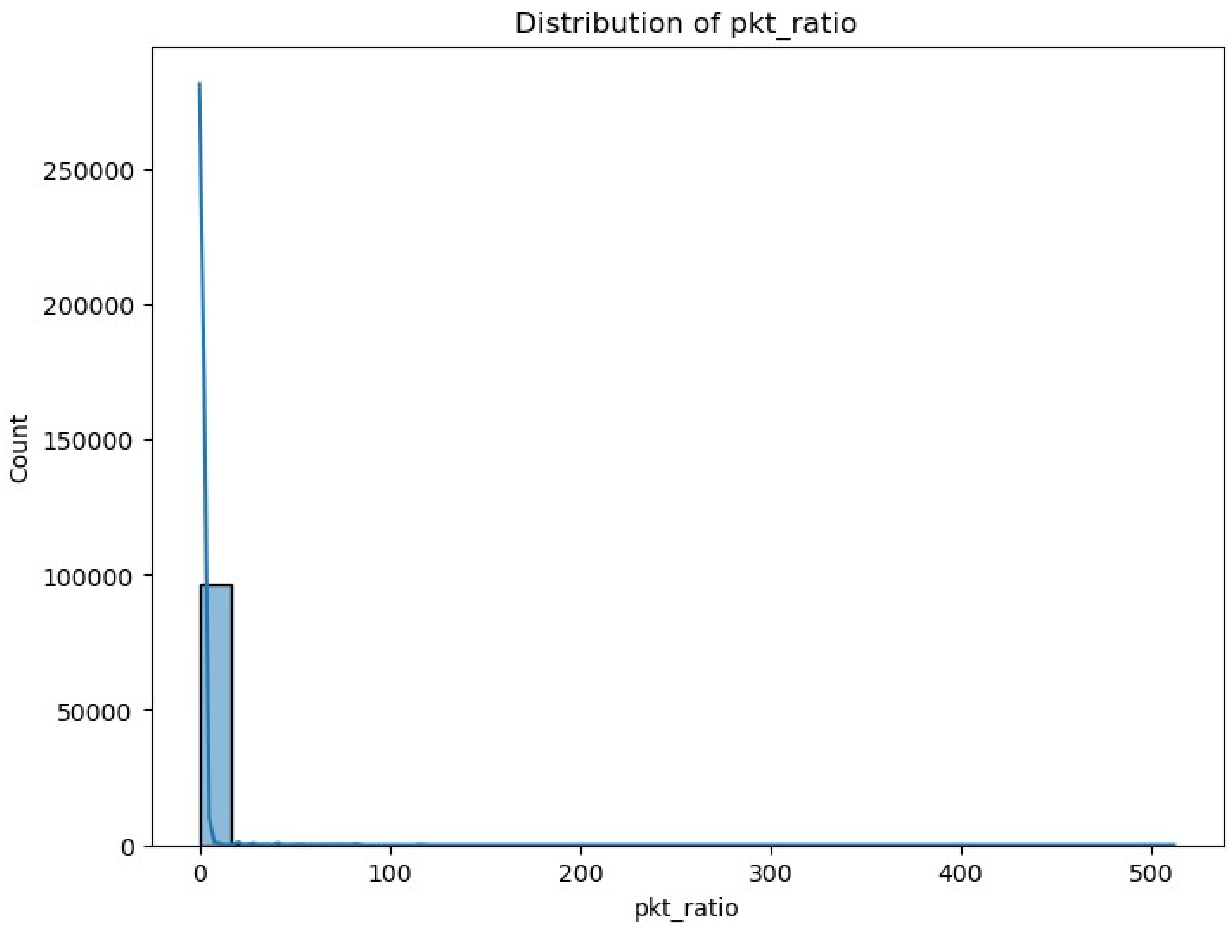
Feature Distributions

```
# Distribution of Numerical Features
numerical_columns = training_df.select_dtypes(include=['float64',
'int64']).columns

# Plot first 6 features
for col in numerical_columns[:6]:
    plt.figure(figsize=(8, 6))
    sns.histplot(training_df[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')
    plt.show()
```


Distribution of stcpb
Count
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
stcpb

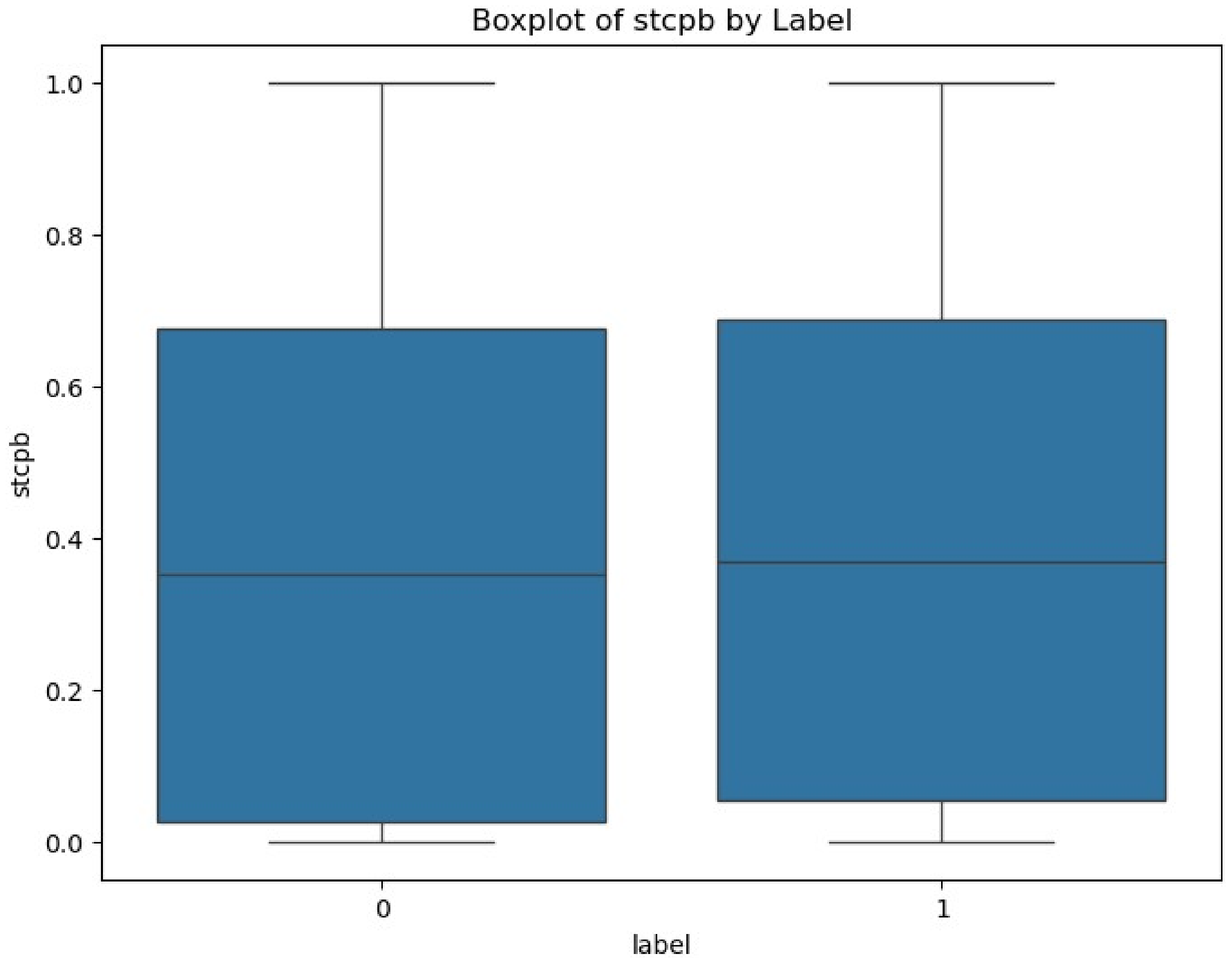


Distribution of dtcpb
Count
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
dtcpb

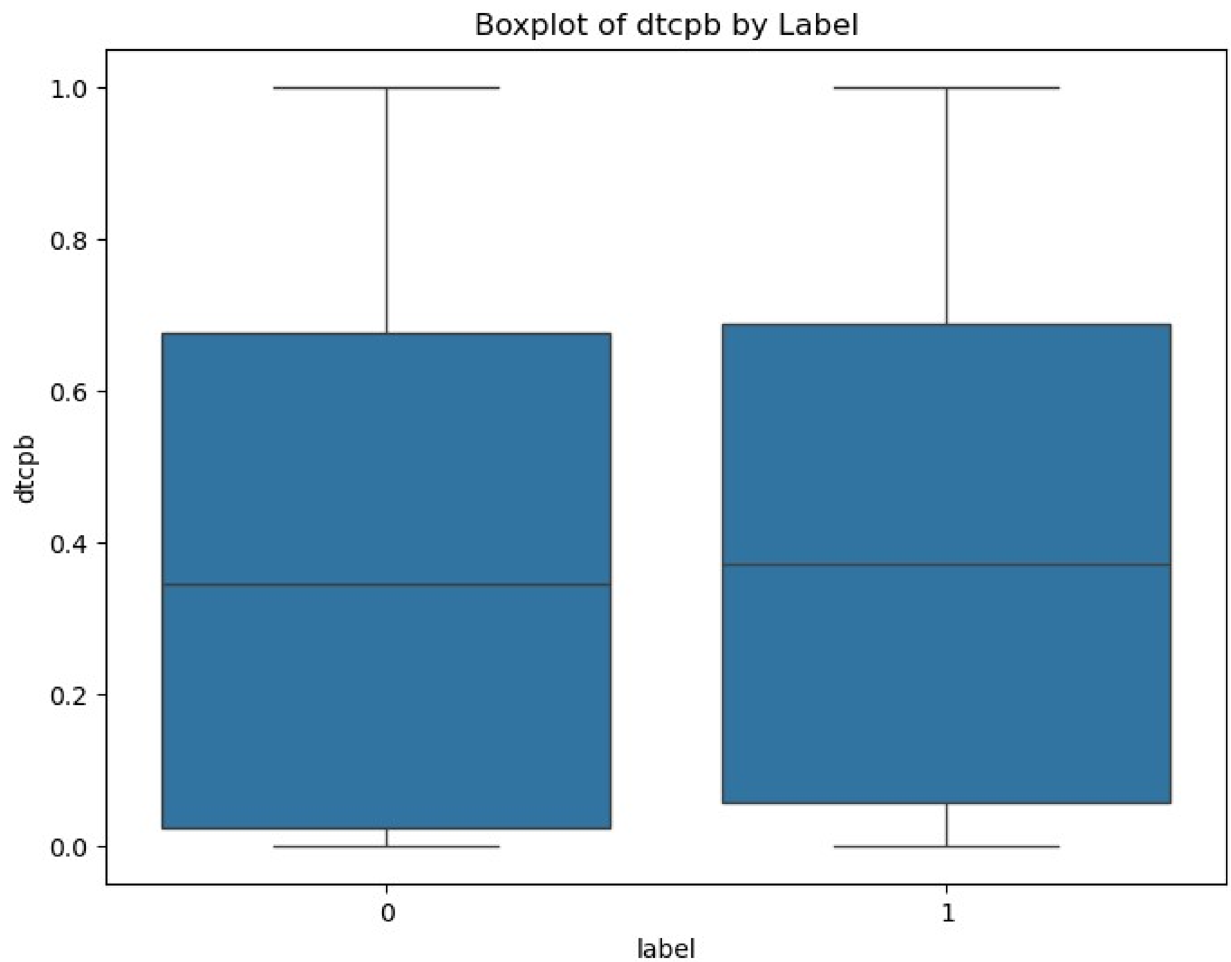


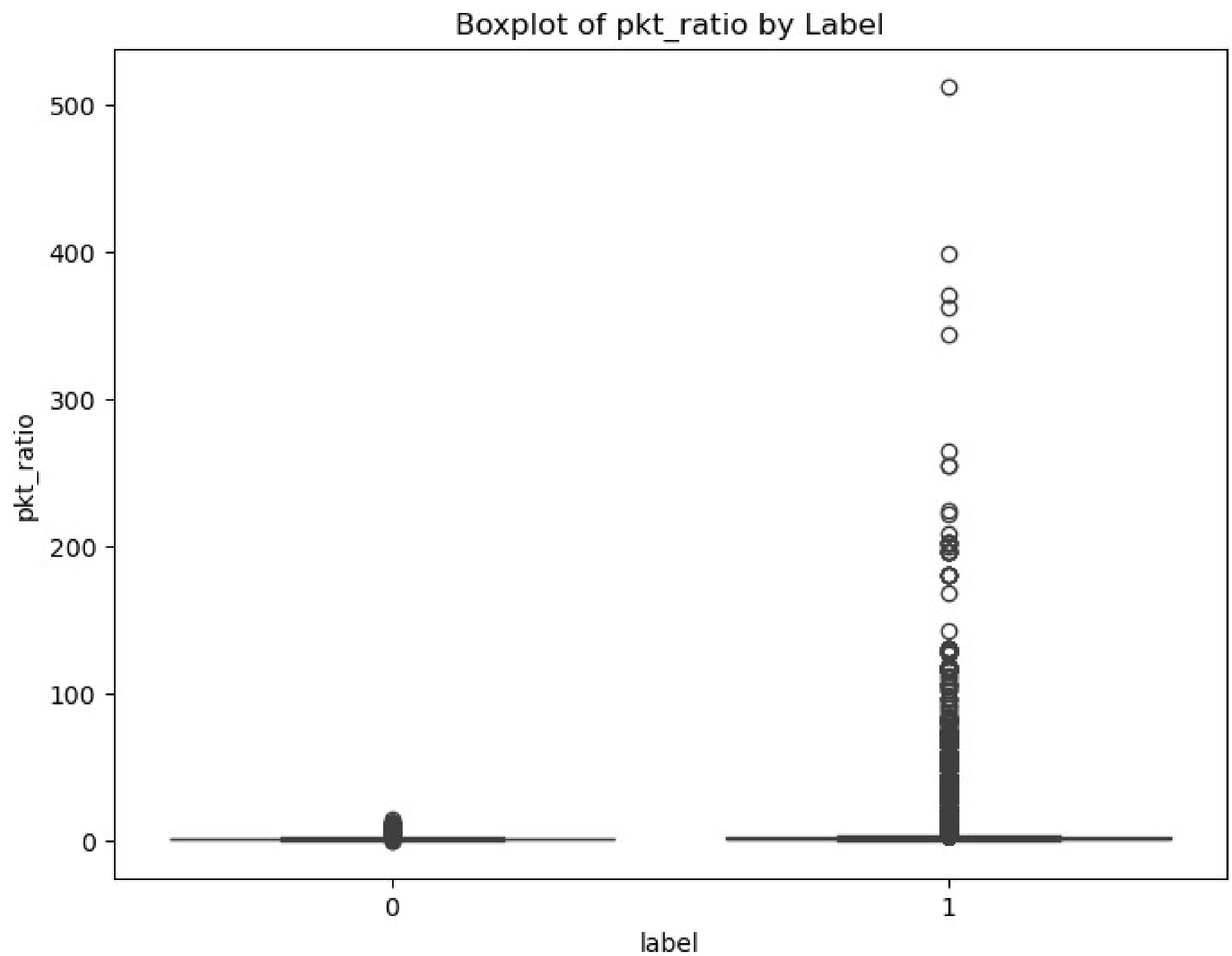
Boxplots for Numeric Features by Label

```
# Boxplots of numerical features grouped by label
# Plot first 6 features
for col in numerical_columns[:6]:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x='label', y=col, data=training_df)
    plt.title(f'Boxplot of {col} by Label')
    plt.show()
```



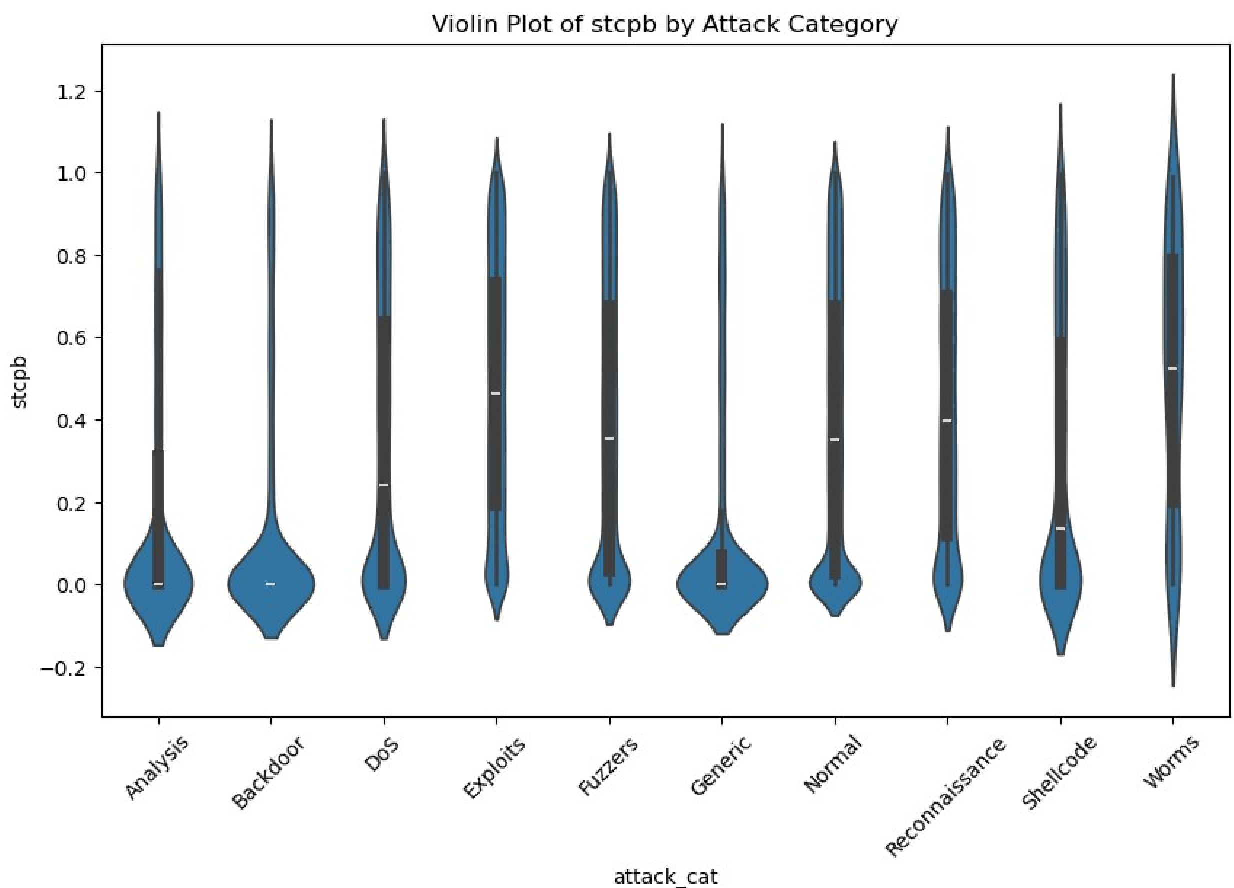
Boxplot of stcpb by Label
stcpb
1.0
0.8
0.6
0.4
0.2
0.0
0
1
label


Boxplot of dtcpb by Label
dtcpb
label
0
1
0.0
0.2
0.4
0.6
0.8
1.0

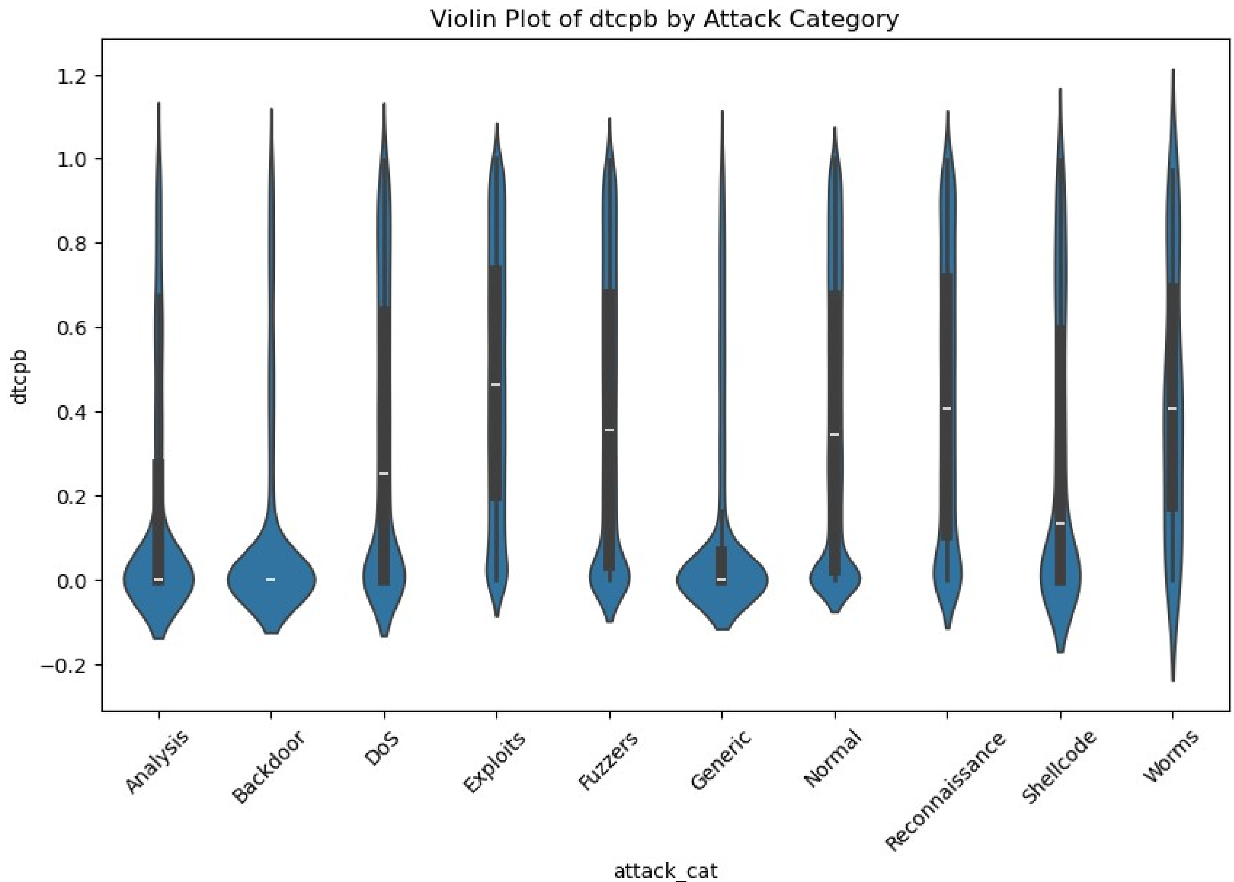


Violin Plots for Features by Attack Category

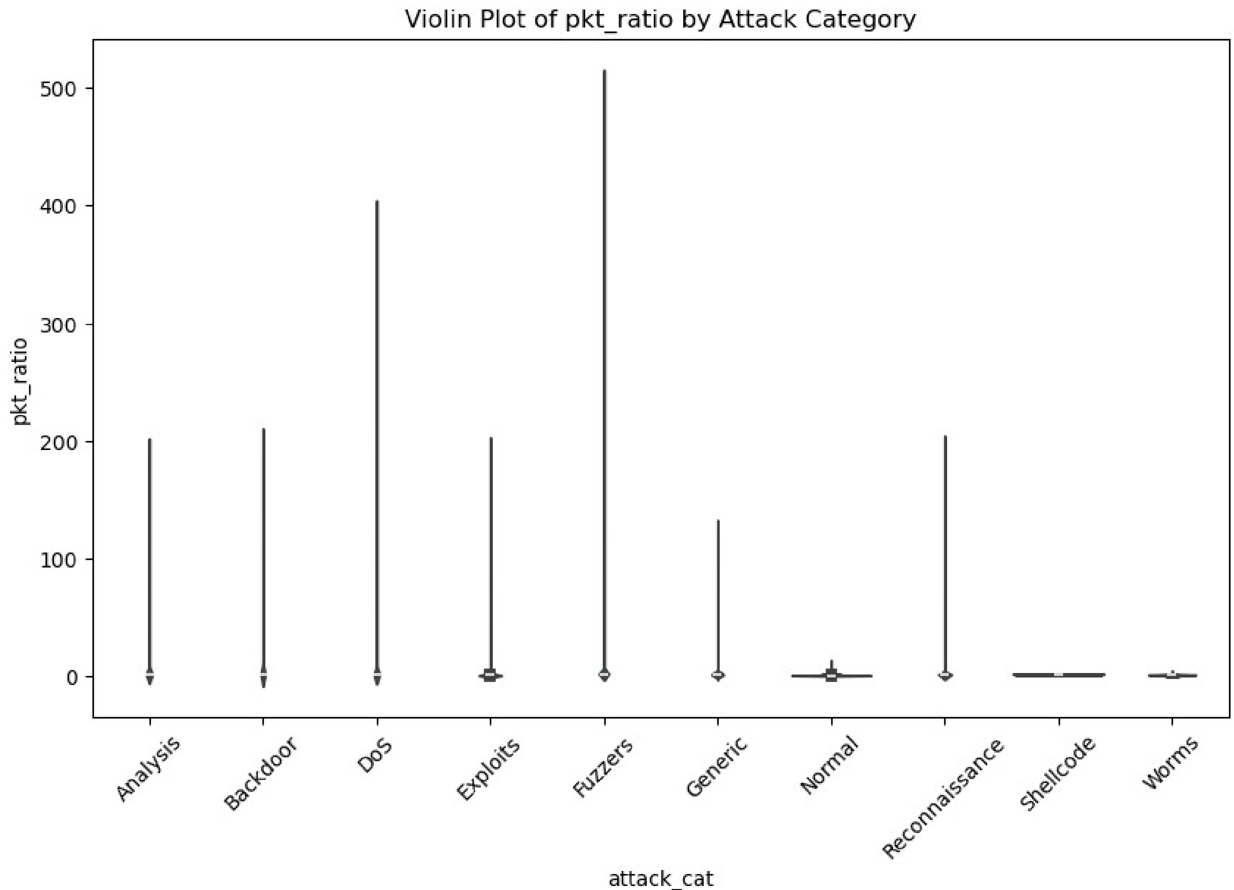
```
# Violin plots for numerical features grouped by attack_cat
 # Plot first 10 features
for col in numerical_columns[:10]:
    plt.figure(figsize=(10, 6))
    sns.violinplot(x='attack_cat', y=col, data=training_df)
    plt.title(f'Violin Plot of {col} by Attack Category')
    plt.xticks(rotation=45)
    plt.show()
```



Violin Plot of stcpb by Attack Category
stcpb
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
Analysis
Backdoor
DoS
Exploits
Fuzzers
Generic
Normal
Reconnaissance
Shellcode
Worms
attack_cat

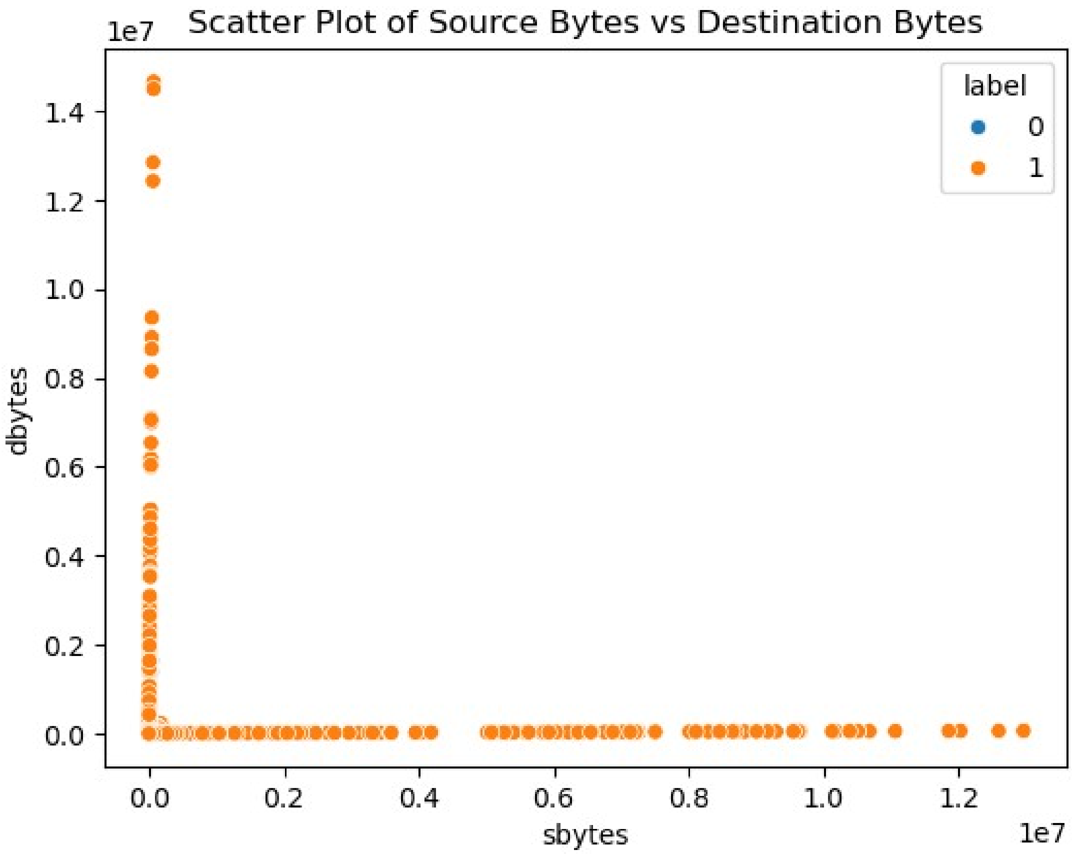


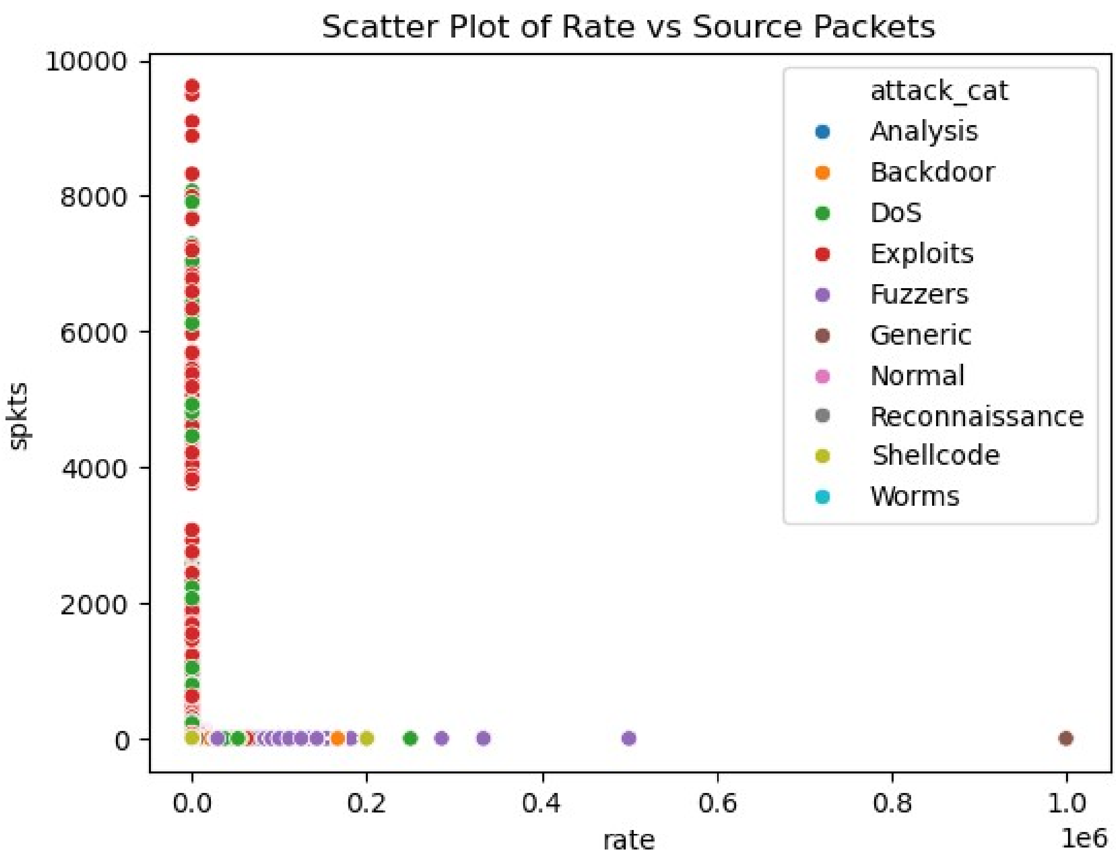
Violin Plot of dtcpb by Attack Category
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
dtcpb
Analysis
Backdoor
DoS
Exploits
Fuzzers
Generic
Normal
Reconnaissance
Shellcode
Worms
attack_cat



Scatter Plots

```
# Scatter plot for relationship between important features
sns.scatterplot(x='sbytes', y='dbytes', hue='label', data=training_df)
plt.title('Scatter Plot of Source Bytes vs Destination Bytes')
plt.show()

sns.scatterplot(x='rate', y='spkts', hue='attack_cat',
data=training_df)
plt.title('Scatter Plot of Rate vs Source Packets')
plt.show()
```

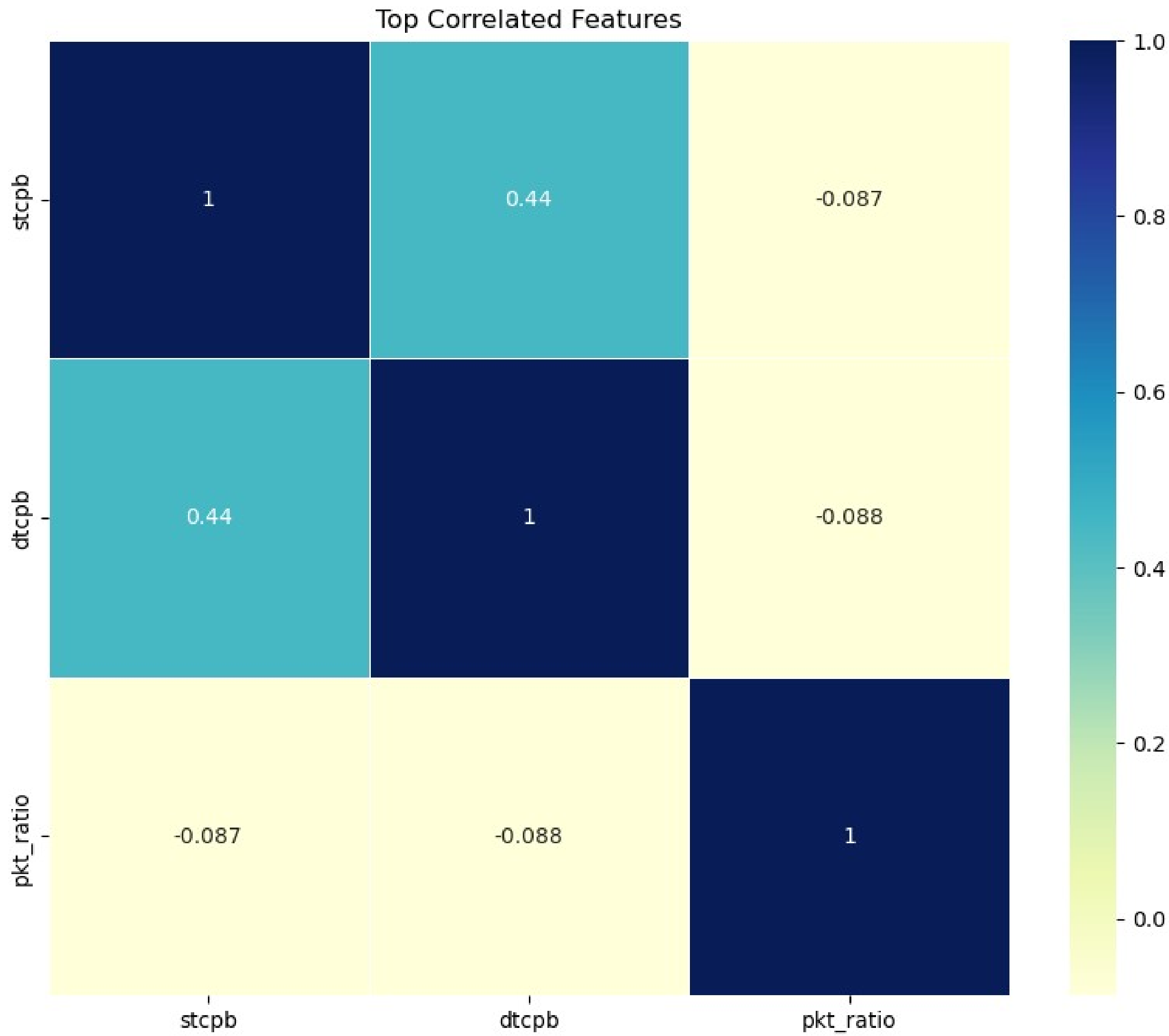

Scatter Plot of Source Bytes vs Destination Bytes
1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
dbytes
label
0
1
0.0
0.2
0.4
0.6
0.8
1.0
1.2
sbytes
1e7



Advanced Correlation Analysis

```
# Compute the correlation matrix for numerical column
numerical_columns = training_df.select_dtypes(include=['float64',
'int64']).columns
numerical_df = training_df[numerical_columns]

correlation = numerical_df.corr()

# Pairwise Correlation for top 10 correlated features
top_corr_pairs = (
    correlation.abs()
    .unstack()
    .sort_values(kind="quicksort", ascending=False)
    .drop_duplicates()
    .head(10)
    .index
)

# Extract unique feature name from the pairs
unique_features = set([feature for pair in top_corr_pairs for feature
in pair])
```

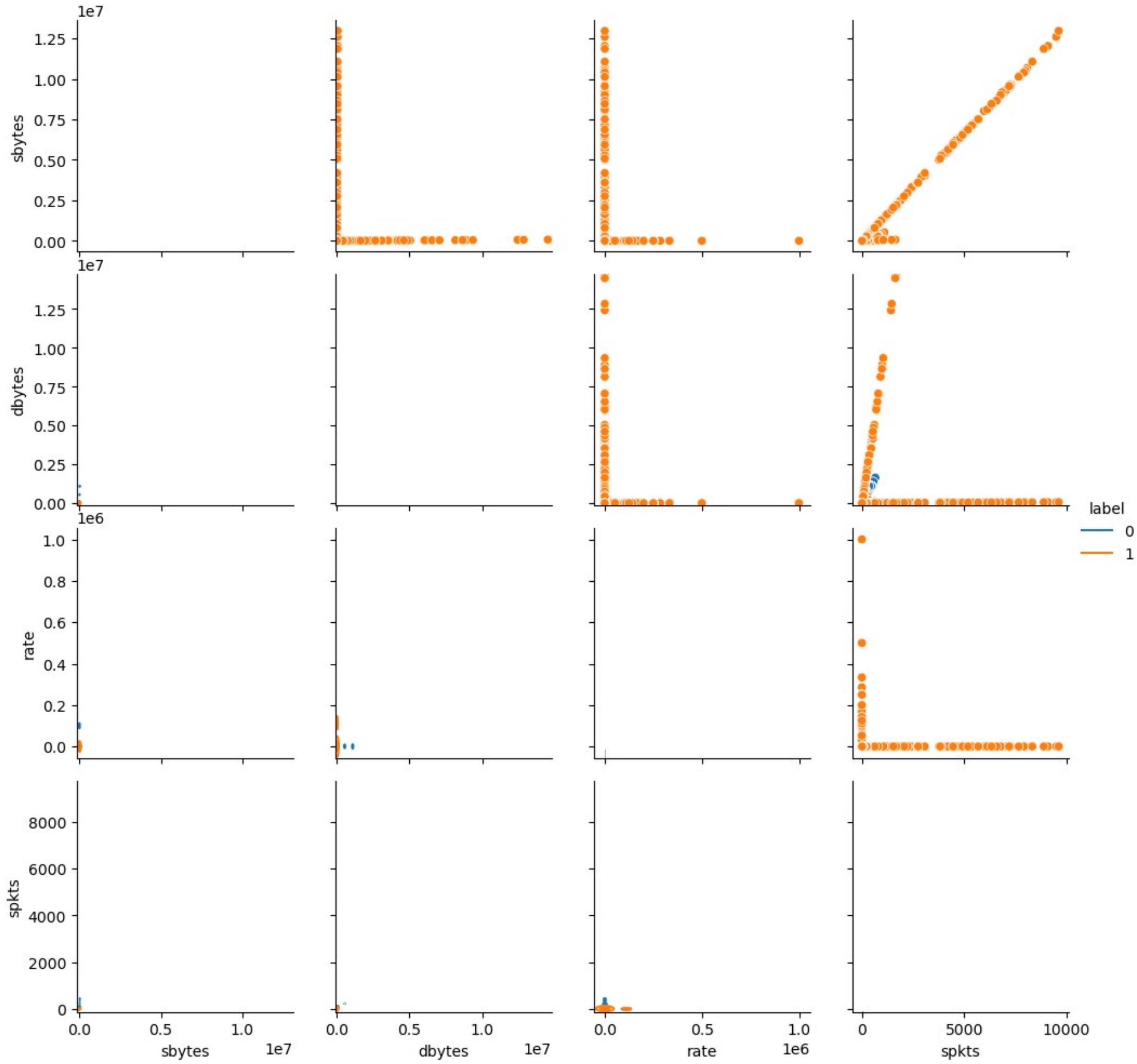
```
# Create a heatmap for the top correlated feature
plt.figure(figsize=(10, 8))
sns.heatmap(
    numerical_df[list(unique_features)].corr(),
    annot=True,
    cmap="YlGnBu",
    linewidths=0.5,
)
plt.title("Top Correlated Features")
plt.show()
```



Pair Grid Plot for Selected Features

```
import warnings
warnings.filterwarnings("ignore")
```

```python
# Pair grid plot for selected features
selected_features = ['sbytes', 'dbytes', 'rate', 'spkts', 'label']

# Creating the pair grid with only scatterplot on upper and kdeplot on lower
g = sns.PairGrid(training_df[selected_features], hue="label")
g.map_upper(sns.scatterplot)  # Upper triangle scatter plots
g.map_lower(sns.kdeplot)  # Lower triangle density plot
g.map_diag(sns.histplot, kde=False)  # Diagonal histograms without KDE
g.add_legend()

plt.show()
```



Model Development:

- Split the dataset into training and testing sets.
- Develop multiple machine learning models for intrusion detection, such as:
  - Decision Trees
  - Random Forests
  - Support Vector Machines (SVM)
  - Neural Networks
- Use Python libraries such as Scikit-learn, TensorFlow, or PyTorch.

Import Required Libraries

```
# Import multiple library
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Prepare Data for Model Development

```
# Features X and target y are separated
X = training_df.drop(columns=['label', 'attack_cat'])  # Drop target
columns
y = training_df['label']  # Binary target: 0 (normal), 1 (attack)

# Identify categorical columns
categorical_columns = X.select_dtypes(include=['category',
'object']).columns

# Apply Label Encoding to all categorical columns
label_encoders = {}  # To store label encoders for future use
for col in categorical_columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])  # Transform categorical to
numeric
    label_encoders[col] = le  # Save the encoder for inverse
transformation if needed

# Ensure all data types are numeric
X = X.apply(pd.to_numeric)
```

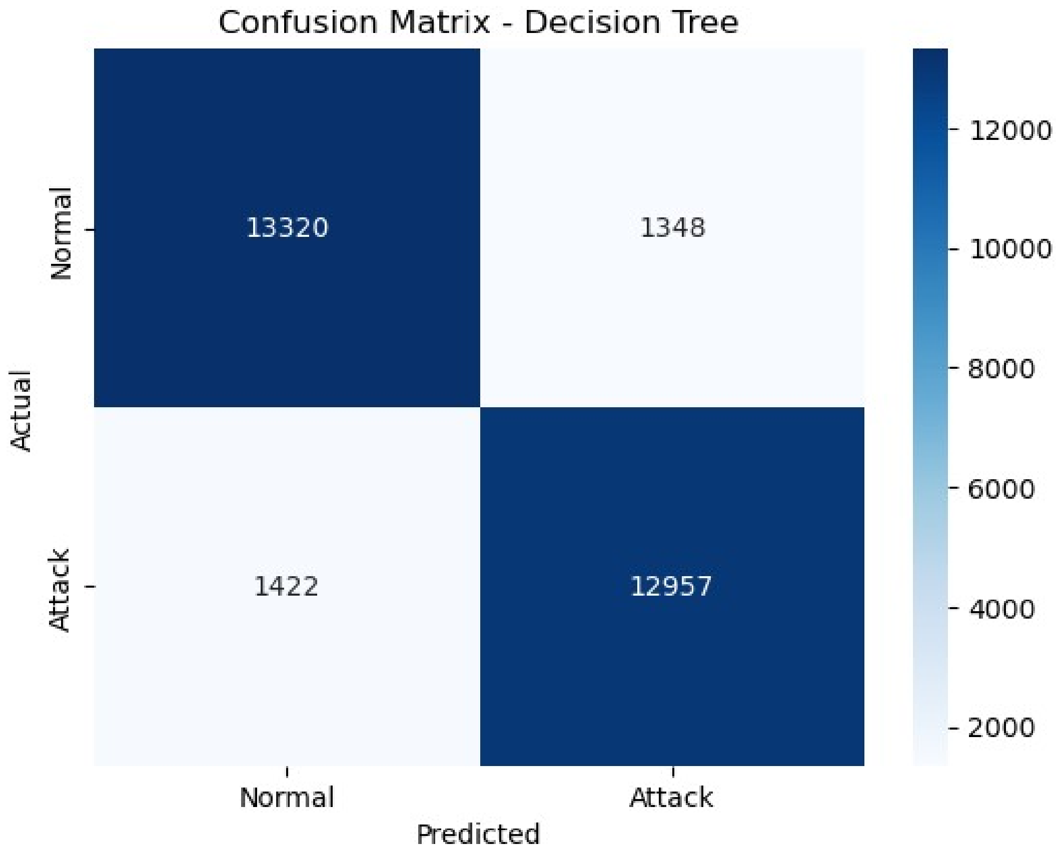
```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42, stratify=y)

# Print shapes of training and testing sets
print(f"Training set shape: {X_train.shape}")
print(f"Testing set shape: {X_test.shape}")
```

```
Training set shape: (67775, 36)
Testing set shape: (29047, 36)
```

Decision Tree Classifier

```python
# Train the model Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Prediction on test set for prediction
y_pred_dt = dt_model.predict(X_test)

# Confusion Matrix Decision Tree Model
print("Decision Tree Model Report:")
print(classification_report(y_test, y_pred_dt))

# Confusion Matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues',
xticklabels=['Normal', 'Attack'], yticklabels=['Normal', 'Attack'])
plt.title("Confusion Matrix - Decision Tree")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

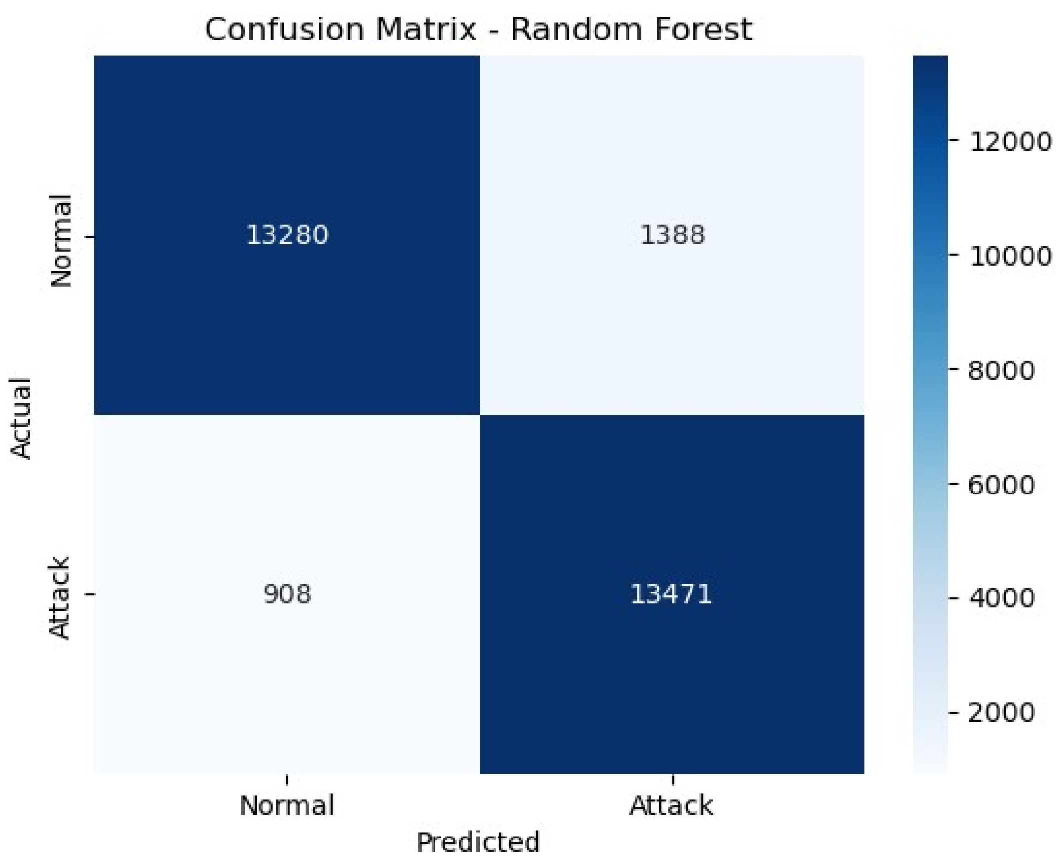
```
Decision Tree Model Report:
              precision    recall  f1-score   support

           0       0.90      0.91      0.91     14668
           1       0.91      0.90      0.90     14379

    accuracy                           0.90     29047
   macro avg       0.90      0.90      0.90     29047
weighted avg       0.90      0.90      0.90     29047
```



Random Forest Model

```
# Train the model Random Forest classifier
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
rf_model.fit(X_train, y_train)

# Prediction on the test set
y_pred_rf = rf_model.predict(X_test)

# Confusion Matrix Report for Random Forest Model
print("Random Forest Model Report:")
print(classification_report(y_test, y_pred_rf))

# Confusion Matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues',
xticklabels=['Normal', 'Attack'], yticklabels=['Normal', 'Attack'])
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

Random Forest Model Report:
              precision    recall  f1-score   support
```

```
           0       0.94      0.91      0.92     14668
           1       0.91      0.94      0.92     14379

    accuracy                           0.92     29047
   macro avg       0.92      0.92      0.92     29047
weighted avg       0.92      0.92      0.92     29047
```

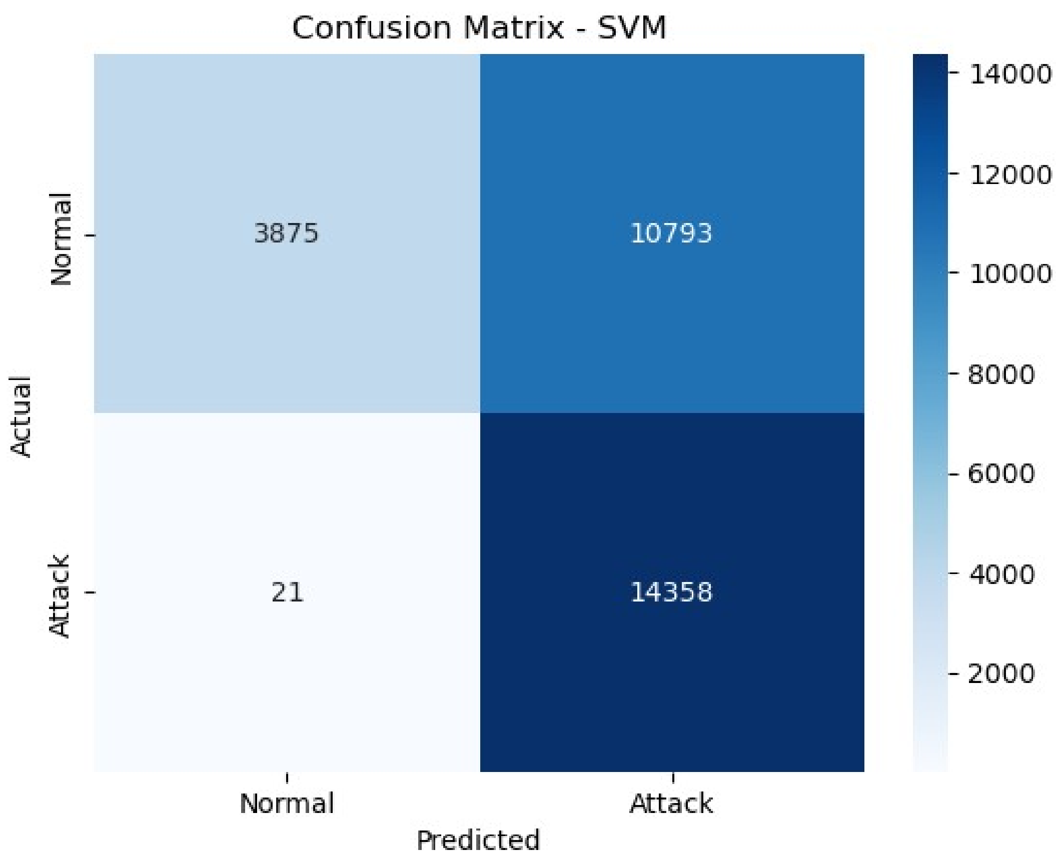


Support Vector Machine (SVM)

```
# Train SVM Classifier
svm_model = SVC(kernel='rbf', random_state=42)
svm_model.fit(X_train, y_train)

# Prediction on test set
y_pred_svm = svm_model.predict(X_test)

# Classification Report SVM Model
print("SVM Model Report:")
print(classification_report(y_test, y_pred_svm))

# Confusion Matrix
```

```
cm_svm = confusion_matrix(y_test, y_pred_svm)
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Normal', 'Attack'], yticklabels=['Normal', 'Attack'])
plt.title("Confusion Matrix - SVM")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
SVM Model Report:
              precision    recall  f1-score   support

           0       0.99      0.26      0.42     14668
           1       0.57      1.00      0.73     14379

    accuracy                           0.63     29047
   macro avg       0.78      0.63      0.57     29047
weighted avg       0.78      0.63      0.57     29047
```

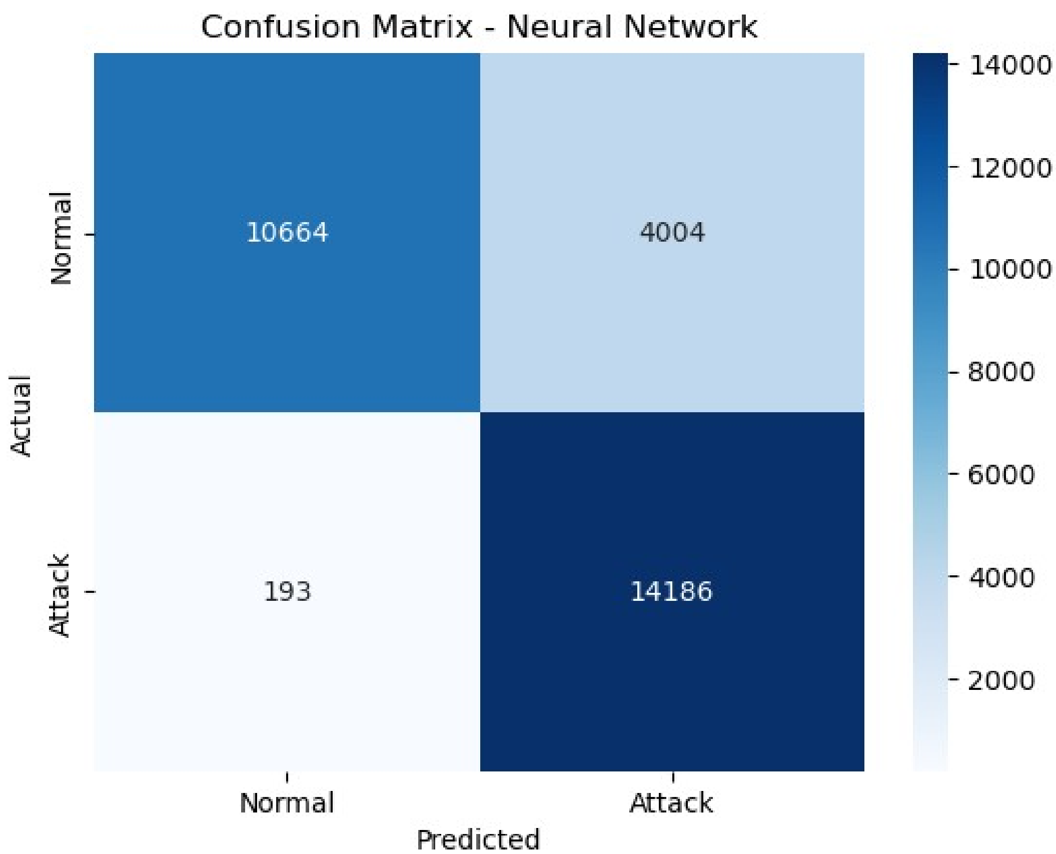


Neural Network MLP Classifier

```
# Train Neural Network on the MLP Classifier
mlp_model = MLPClassifier(random_state=42, hidden_layer_sizes=(50,
```

```
25), max_iter=300)
mlp_model.fit(X_train, y_train)

# Prediction on test set
y_pred_mlp = mlp_model.predict(X_test)

# Classification Report Neural Network Model
print("Neural Network (MLP) Model Report:")
print(classification_report(y_test, y_pred_mlp))

# Confusion Matrix with Normal and Attack in y axis
cm_mlp = confusion_matrix(y_test, y_pred_mlp)
sns.heatmap(cm_mlp, annot=True, fmt='d', cmap='Blues',
xticklabels=['Normal', 'Attack'], yticklabels=['Normal', 'Attack'])
plt.title("Confusion Matrix - Neural Network")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Neural Network (MLP) Model Report:
              precision    recall  f1-score   support

           0       0.98      0.73      0.84     14668
           1       0.78      0.99      0.87     14379

    accuracy                           0.86     29047
   macro avg       0.88      0.86      0.85     29047
weighted avg       0.88      0.86      0.85     29047
```



Model Evaluation

- Evaluate the models using metrics such as accuracy precision recall f1 score and ROC-AUC.
- Perform cross-validation to ensure the robustness of the models

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, roc_curve
from sklearn.model_selection import cross_val_score
```

Function for all model evaluation

```
# Function to calculate the evaluation of a model
def evaluate_model(model, X_test, y_test, model_name):
    # Prediction on test data
    y_pred = model.predict(X_test)

    # Compute the all metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
```

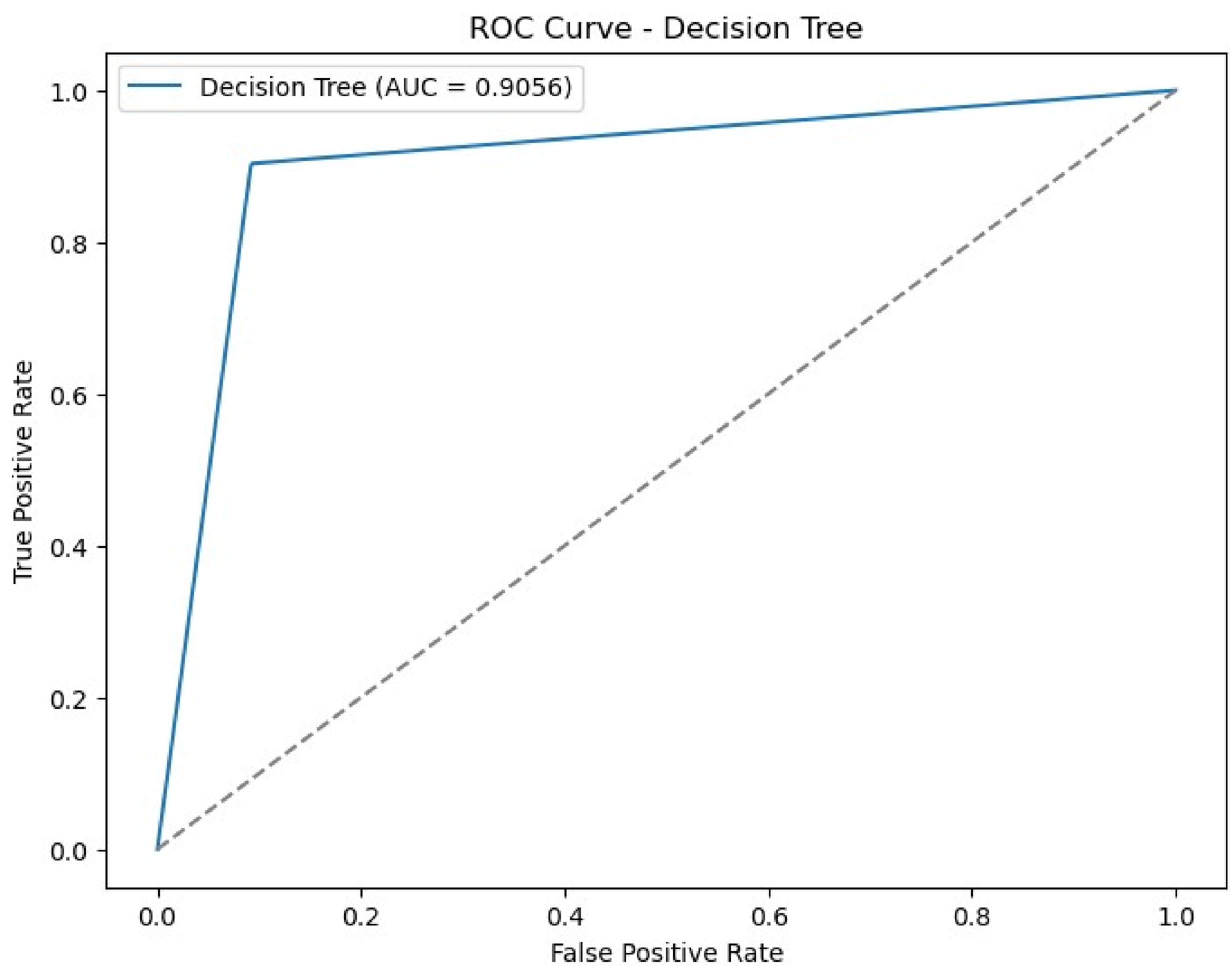
```
    roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
if hasattr(model, "predict_proba") else None

    # Print the all evaluation result
    print(model_name + " Evaluation Metrics:")
    print("Accuracy: " + str(round(accuracy, 4)))
    print("Precision: " + str(round(precision, 4)))
    print("Recall: " + str(round(recall, 4)))
    print("F1 Score: " + str(round(f1, 4)))
    if roc_auc is not None:
        print("ROC-AUC: " + str(round(roc_auc, 4)))

    # Plot ROC Curve
    if roc_auc is not None:
        fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test)[:,
1])
        plt.figure(figsize=(8, 6))
        plt.plot(fpr, tpr, label=model_name + " (AUC = " +
str(round(roc_auc, 4)) + ")")
        plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
        # ROC Curve Main Title
        plt.title("ROC Curve - " + model_name)
        # X label title
        plt.xlabel("False Positive Rate")
        # Y label title
        plt.ylabel("True Positive Rate")
        plt.legend()
        plt.show()
```

Evaluate the all models by using above function

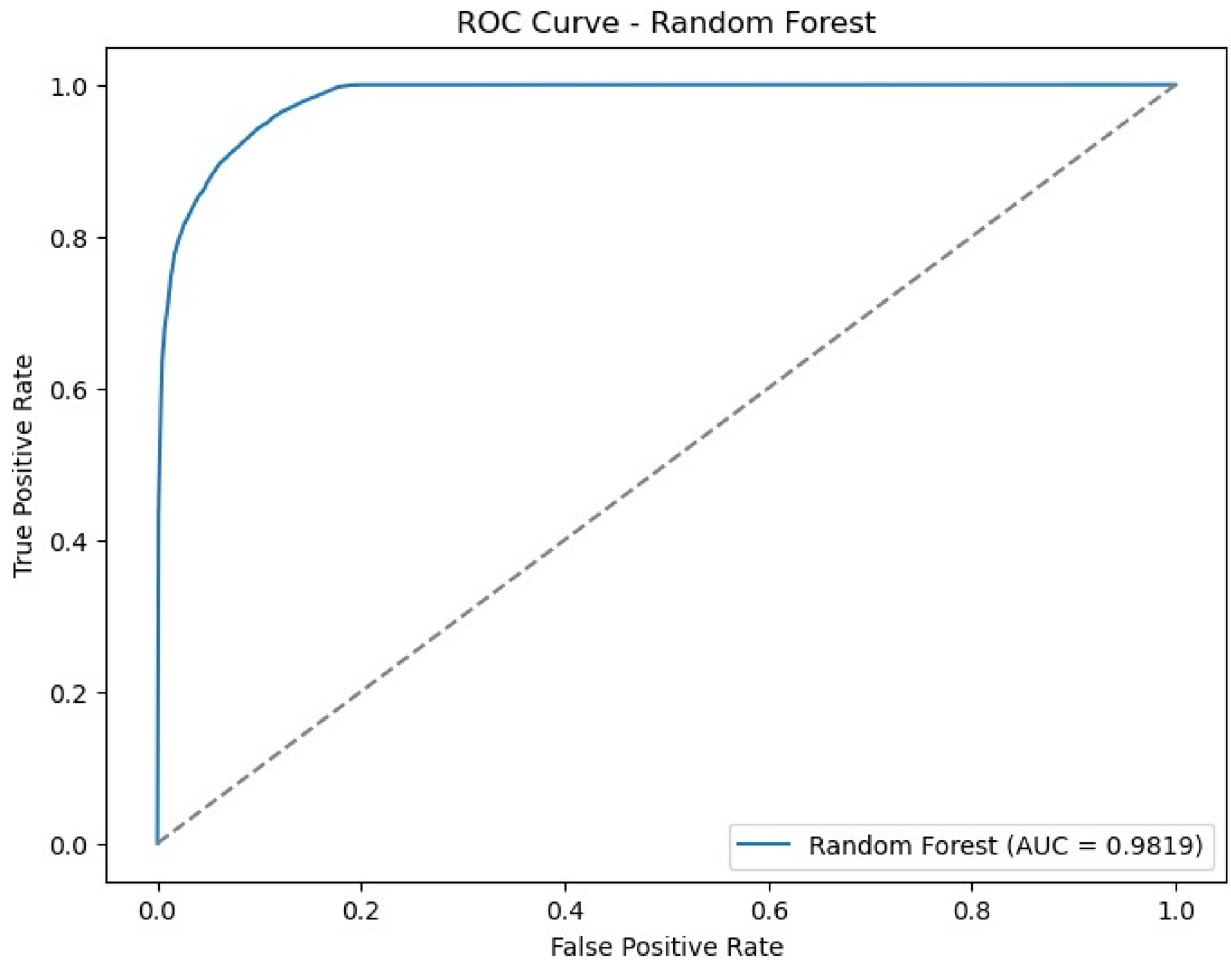
```
# Evaluate the Decision Tree model
evaluate_model(dt_model, X_test, y_test, "Decision Tree")

# Evaluate the Random Forest model
evaluate_model(rf_model, X_test, y_test, "Random Forest")

# Evaluate SVM model SVM does not provide predict_proba by the default
if hasattr(svm_model, "decision_function"):
    y_pred_svm = svm_model.decision_function(X_test)
    roc_auc_svm = roc_auc_score(y_test, y_pred_svm)
    print("SVM ROC-AUC: " + str(round(roc_auc_svm, 4)))
    fpr, tpr, _ = roc_curve(y_test, y_pred_svm)
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label="SVM (AUC = " + str(round(roc_auc_svm,
4)) + ")")
    plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
    # Print the main title
    plt.title("ROC Curve - " + "SVM")
```

```
    # print the x label
    plt.xlabel("False Positive Rate")
    # print the y label
    plt.ylabel("True Positive Rate")
    # print the legend
    plt.legend()
    plt.show()
else:
    print("SVM does not support predict_proba. Skipping ROC-AUC
evaluation.")

# Evaluate the last model neural network
evaluate_model(mlp_model, X_test, y_test, "Neural Network")
```

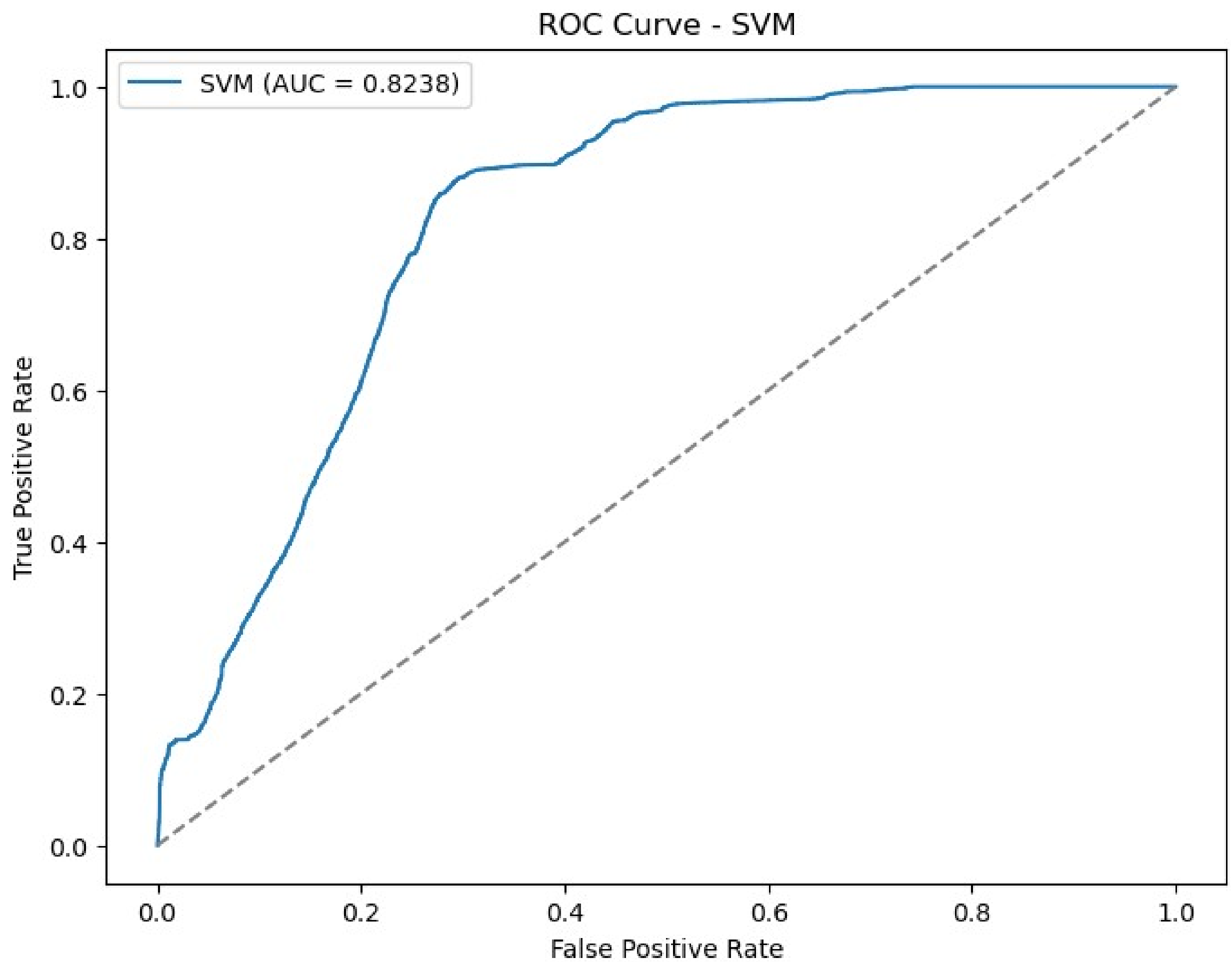
```
Decision Tree Evaluation Metrics:
Accuracy: 0.9046
Precision: 0.9058
Recall: 0.9011
F1 Score: 0.9034
ROC-AUC: 0.9056
```



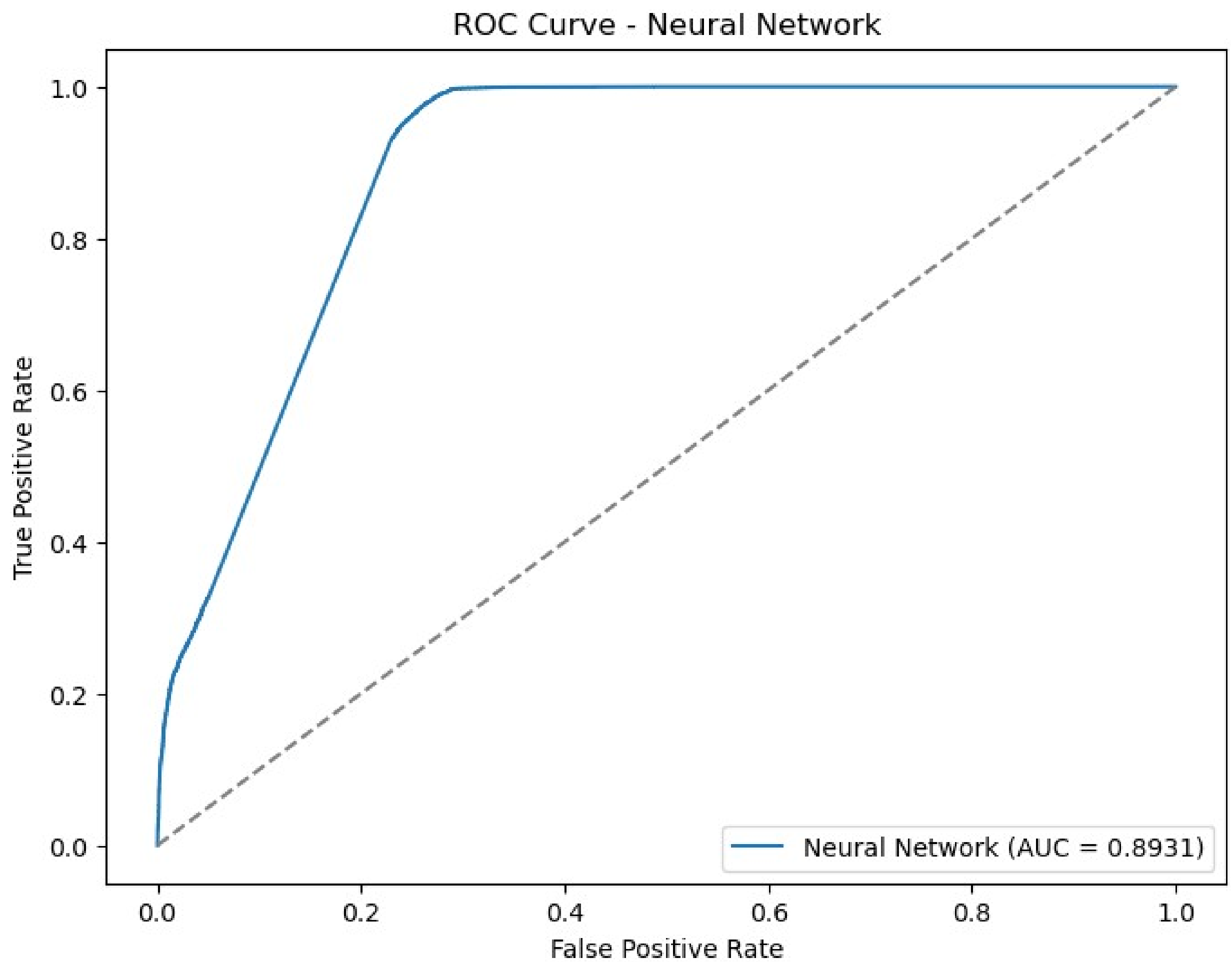
```
Random Forest Evaluation Metrics:
Accuracy: 0.921
Precision: 0.9066
Recall: 0.9369
F1 Score: 0.9215
ROC-AUC: 0.9819
```




```
SVM ROC-AUC: 0.8238
```




```
Neural Network Evaluation Metrics:
Accuracy: 0.8555
Precision: 0.7799
Recall: 0.9866
F1 Score: 0.8711
ROC-AUC: 0.8931
```



Perform The Cross Validation

```
# Function for the cross validation
def cross_validate_model(model, X, y, model_name, cv=5):
    print("Cross-Validation Scores for " + model_name + ":")
    cv_scores = cross_val_score(model, X, y, cv=cv,
scoring="accuracy")
    print("Scores: " + str(cv_scores))
    # print the mean accuracy
    print("Mean Accuracy: " + str(round(cv_scores.mean(), 4)))
    # print the standard deviation
    print("Standard Deviation: " + str(round(cv_scores.std(), 4)))
    print("\n")

# Cross Validation for Decision Tree
cross_validate_model(dt_model, X, y, "Decision Tree")

# Cross Validation for Random Forest
cross_validate_model(rf_model, X, y, "Random Forest")
```

```python
# Cross Validation for SVM
cross_validate_model(svm_model, X, y, "SVM")

# Cross Validation for Neural Network
cross_validate_model(mlp_model, X, y, "Neural Network")
```

```
Cross-Validation Scores for Decision Tree:
Scores: [0.897134   0.93787761 0.93756455 0.93673828 0.56930386]
Mean Accuracy: 0.8557
Standard Deviation: 0.1441


Cross-Validation Scores for Random Forest:
Scores: [0.91959721 0.95522851 0.95600083 0.9531605  0.53088205]
Mean Accuracy: 0.863
Standard Deviation: 0.1666


Cross-Validation Scores for SVM:
Scores: [0.62912471 0.67900852 0.68265854 0.66174344 0.49375129]
Mean Accuracy: 0.6293
Standard Deviation: 0.0703


Cross-Validation Scores for Neural Network:
Scores: [0.73684482 0.81611154 0.80417269 0.78227639 0.4508366 ]
Mean Accuracy: 0.718
Standard Deviation: 0.1363
```

Hyperparameter tuning

- Use techniques such as Grid Search or Random Search to optimize the Hyperparameters of the models.

```python
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

Decision Tree Hyperparameter Tuning with Grid Search

```python
# Define parameter grid for Decision Tree
dt_param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform the Grid Search on the model
```

```
dt_grid_search =
GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                              param_grid=dt_param_grid,
                              scoring='accuracy',
                              cv=5,
                              verbose=1,
                              n_jobs=-1)
dt_grid_search.fit(X_train, y_train)

# Print the best parameters and accuracy for the decision tree after
hyperparameter tuning
print("Best Parameters for Decision Tree:")
print(dt_grid_search.best_params_)
print("Best Cross-Validation Accuracy: " +
str(round(dt_grid_search.best_score_, 4)))
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
Best Parameters for Decision Tree:
{'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': 4,
'min_samples_split': 10}
Best Cross-Validation Accuracy: 0.9088
```

Random Forest Hyperparameter Tuning with Random Search

```
# Parameter grid for Random Forest
rf_param_grid_optimized = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'bootstrap': [True]
}

# Perform Random Search with fewer iteration and reduced parameter
grid than
rf_random_search_optimized = RandomizedSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_distributions=rf_param_grid_optimized,
    # Reduced number of parameter combination
    n_iter=10,
    scoring='accuracy',
    # Reduced cross validation folds
    cv=3,
    verbose=1,
    random_state=42,
    n_jobs=-1
)
```

```python
# fitting the data in it
rf_random_search_optimized.fit(X_train, y_train)

# Print the best parameter and accuracy after random search
print("Best Parameters for Random Forest (Optimized):")
print(rf_random_search_optimized.best_params_)
print("Best Cross-Validation Accuracy (Optimized): " +
str(round(rf_random_search_optimized.best_score_, 4)))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Parameters for Random Forest (Optimized):
{'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 1,
'max_depth': 20, 'bootstrap': True}
Best Cross-Validation Accuracy (Optimized): 0.9212
```

Support Vector Machine Hyperparameter Tuning with Grid Search

```python
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Pipeline with scaling
svm_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(random_state=42))
])

# Parameter grid
svm_param_grid = {
    'svc__C': [1, 10],
    # Linear Kernel
    'svc__kernel': ['linear']
}

# Perform Randomize Search with limit fold
svm_random_search = RandomizedSearchCV(
    estimator=svm_pipeline,
    param_distributions=svm_param_grid,
    scoring='accuracy',
    cv=2,
    n_iter=2,
    verbose=1,
    # Use single core to prevent freezing
    n_jobs=1
)

# Use a subset of data for quick testing
X_train_subset = X_train.sample(25000) if len(X_train) > 25000 else
X_train
```

```
y_train_subset = y_train.loc[X_train_subset.index]

# Fit the model
svm_random_search.fit(X_train_subset, y_train_subset)

# Print result
print("Best Parameters for SVM:")
print(svm_random_search.best_params_)
print("Best Cross-Validation Accuracy: " +
str(round(svm_random_search.best_score_, 4)))
```

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
Best Parameters for SVM:
{'svc__kernel': 'linear', 'svc__C': 1}
Best Cross-Validation Accuracy: 0.8667
```

Neural Network Hyperparameter Tuning with Grid Search

```
# Parameter grid for Neural Network
mlp_param_grid = {
    'hidden_layer_sizes': [(50,), (100,)],
    # Most commonly use activation
    'activation': ['relu'],
    # Effective solver
    'solver': ['adam'],
    'alpha': [0.0001, 0.001],
    # Single learning rate
    'learning_rate': ['adaptive']
}

# Perform Grid Search on Neural network
# iteration is 300 and state is random
mlp_grid_search = GridSearchCV(estimator=MLPClassifier(max_iter=300,
random_state=42),
                               param_grid=mlp_param_grid,
                               scoring='accuracy',
                               cv=3,
                               verbose=1,
                               n_jobs=-1)
mlp_grid_search.fit(X_train, y_train)

# Print the best parameter and accuracy
print("Best Parameters for Neural Network:")
print(mlp_grid_search.best_params_)
print("Best Cross-Validation Accuracy: " +
str(round(mlp_grid_search.best_score_, 4)))
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
Best Parameters for Neural Network:
{'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50,),
```

```
'learning_rate': 'adaptive', 'solver': 'adam'}
Best Cross-Validation Accuracy: 0.8363
```

Model Performance Visualization (Accuracy, Precision, Recall, F1-Score)

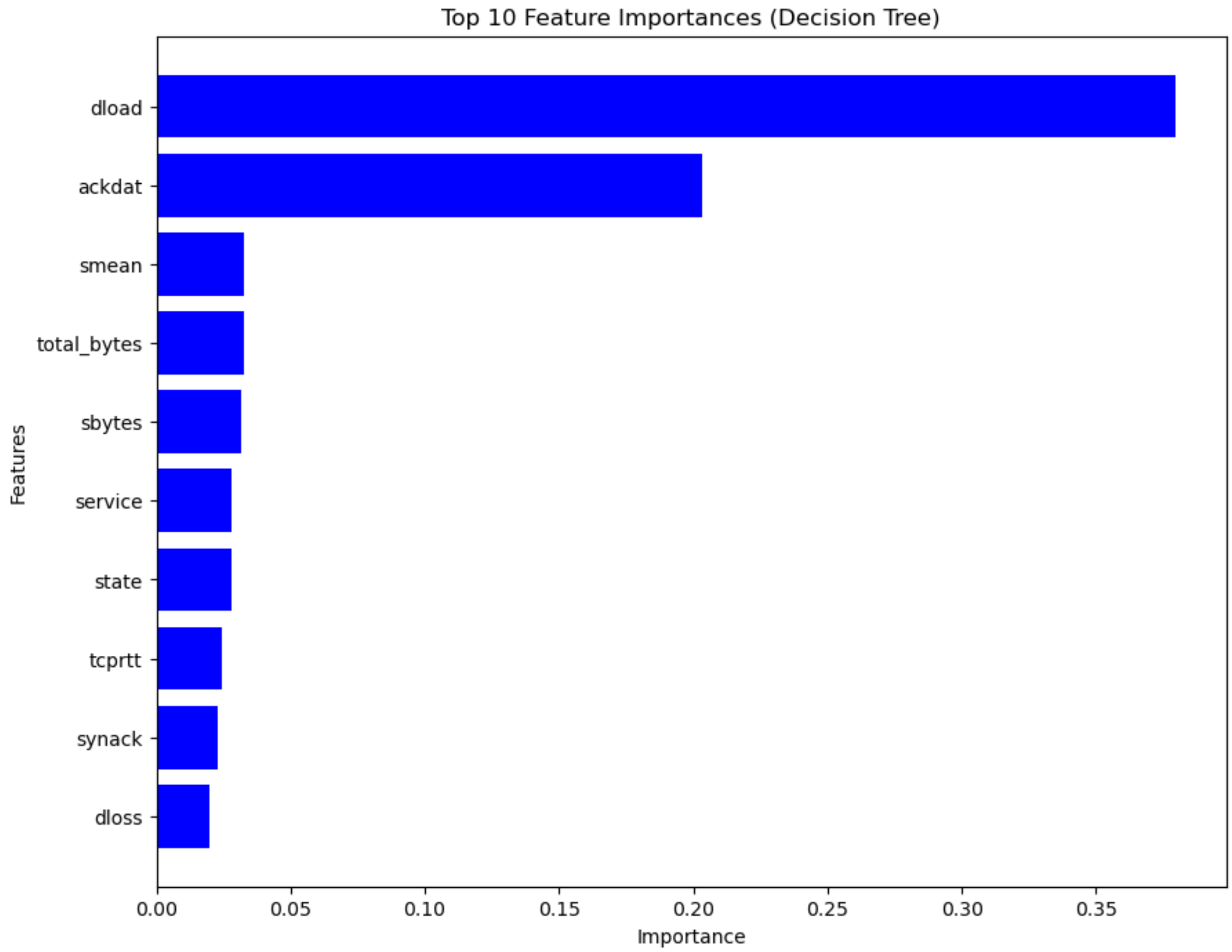
```
# Model names
models = ['Decision Tree', 'Random Forest', 'SVM', 'Neural Network']

# Model performance metrics
accuracy = [0.9046, 0.921, 0.6300, 0.8555]
precision = [0.9058, 0.9066, 0.7800, 0.7799]
recall = [0.9011, 0.9369, 0.9900, 0.9866]
f1_score = [0.9034, 0.9215, 0.5700, 0.8711]

# Metrics and corresponding values
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
values = [accuracy, precision, recall, f1_score]

# Save all figures
fig_paths = []
for i, metric in enumerate(metrics):
    plt.figure(figsize=(8, 6))
    plt.bar(models, values[i], color=['pink', 'red', 'orange',
'yellow'])
    plt.title('Model Comparison: ' + metric)
    plt.ylabel(metric)
    plt.xlabel('Models')
    plt.ylim(0, 1)
    file_name = f"model_comparison_{metric.lower()}.png"
    plt.savefig(file_name)
    fig_paths.append(file_name)
     # Close the figure after saving
    plt.close()

fig_paths
```

```
['model_comparison_accuracy.png',
 'model_comparison_precision.png',
 'model_comparison_recall.png',
 'model_comparison_f1 score.png']
```

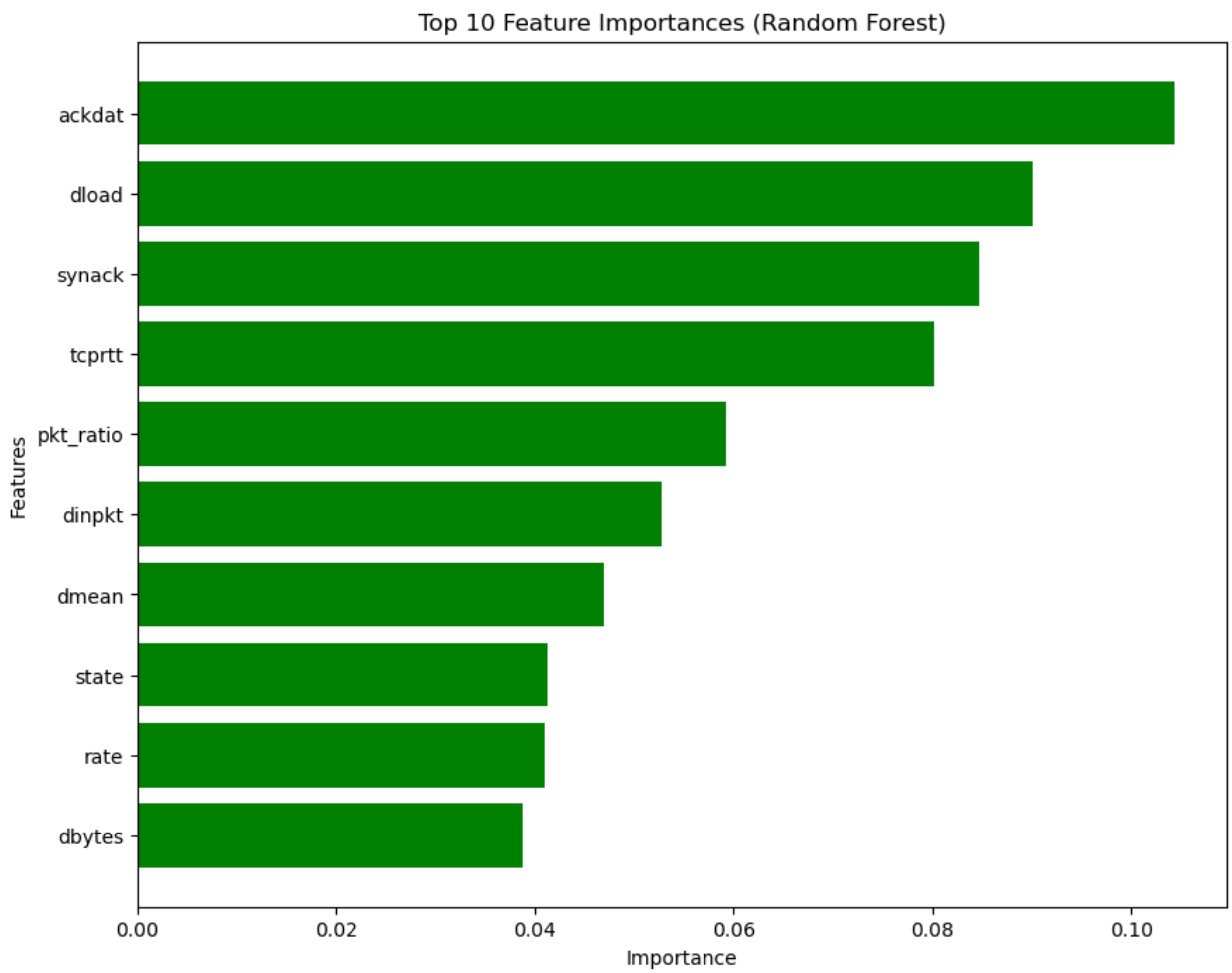
Feature Importance for Decision Tree and Random Forest

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

# Feature Importance for Decision Tree
dt_feature_importances = pd.DataFrame({
```

```
    'Feature': X_train.columns,
    'Importance': dt_model.feature_importances_
}).sort_values(by='Importance', ascending=False)

# Plot Decision Tree Feature Importance
plt.figure(figsize=(10, 8))
plt.barh(dt_feature_importances['Feature'][:10],
dt_feature_importances['Importance'][:10], color='blue')
plt.gca().invert_yaxis()
plt.title('Top 10 Feature Importances (Decision Tree)')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.savefig("feature_importance_decision_tree.png")
plt.show()

# Feature Importance for Random Forest
rf_feature_importances = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': rf_model.feature_importances_
}).sort_values(by='Importance', ascending=False)

# Plot Random Forest Feature Importance
plt.figure(figsize=(10, 8))
plt.barh(rf_feature_importances['Feature'][:10],
rf_feature_importances['Importance'][:10], color='green')
plt.gca().invert_yaxis()
plt.title('Top 10 Feature Importances (Random Forest)')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.savefig("feature_importance_random_forest.png")
plt.show()
```



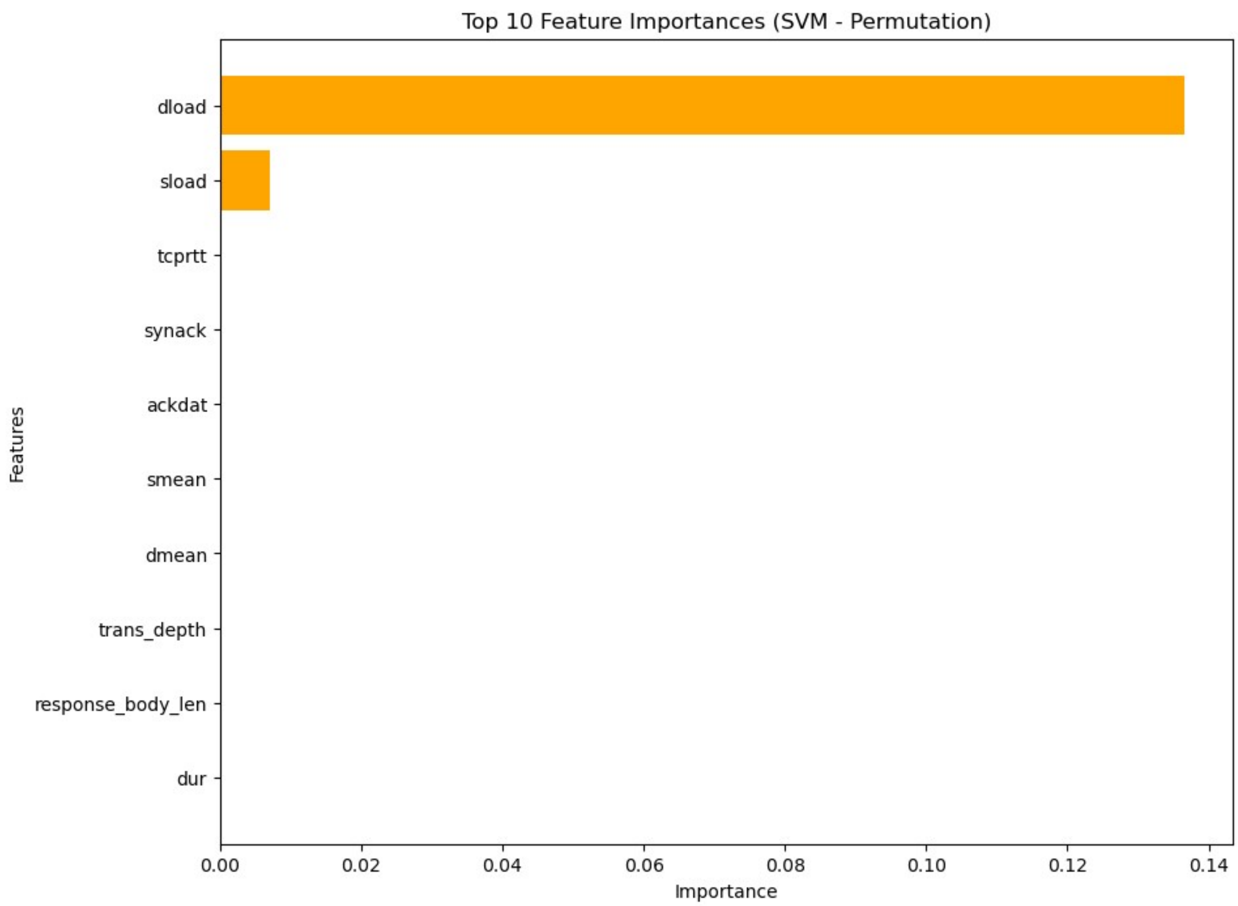
Top 10 Feature Importances (Decision Tree)
Features
dload
ackdat
smean
total_bytes
sbytes
service
state
tcprtt
synack
dloss
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Importance

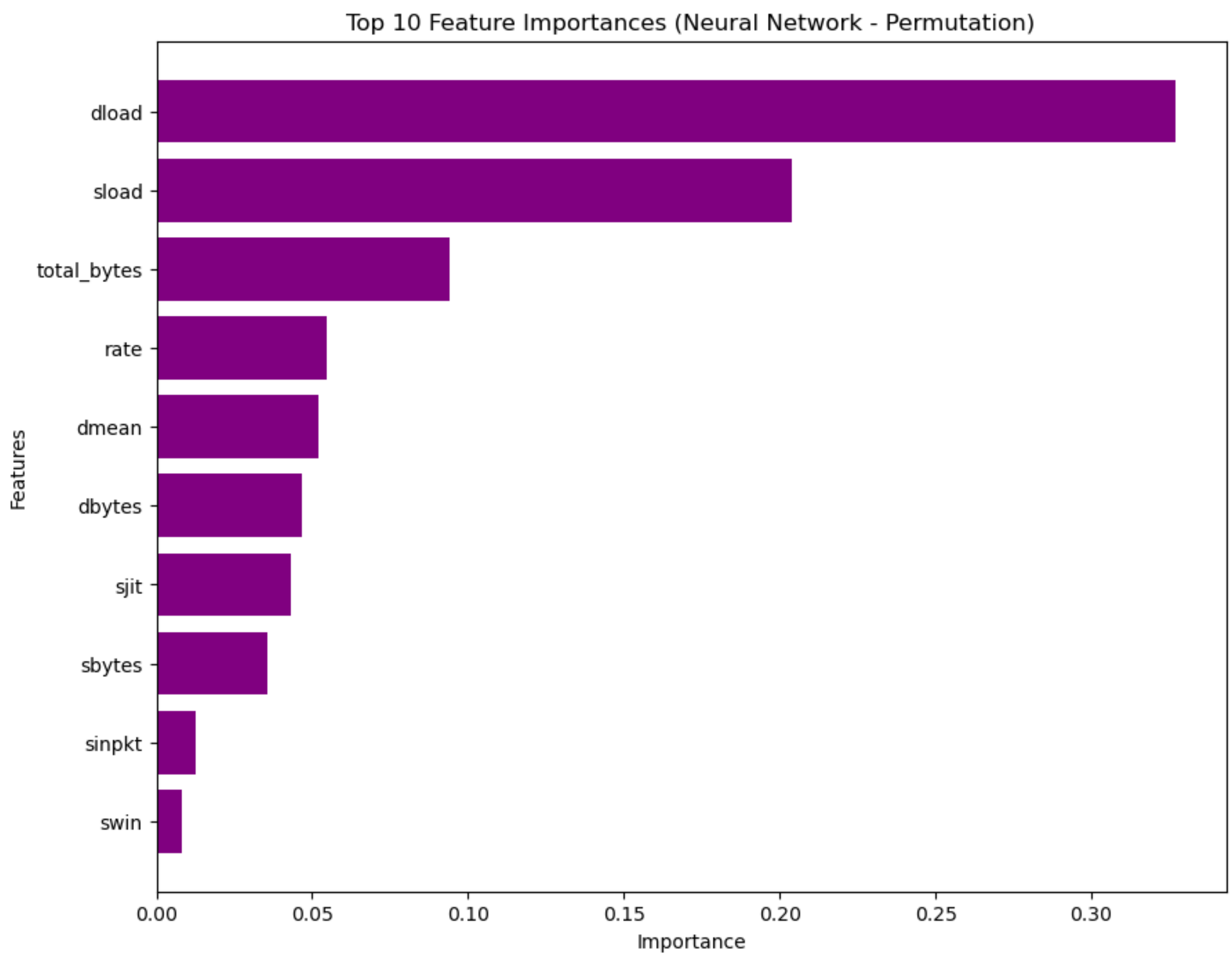


Feature Importance for SVM and Neural Network Using Permutation Importance 2500 samples

```
from sklearn.inspection import permutation_importance

# Sample test data to 2500 rows
X_test_sample = X_test.sample(n=2500, random_state=42)
y_test_sample = y_test.loc[X_test_sample.index]

# n_repeats 3
n_repeats = 3

# Permutation Importance for SVM
svm_importance = permutation_importance(svm_model, X_test_sample,
y_test_sample, scoring='accuracy', n_repeats=n_repeats,
random_state=42)

# DataFrame for SVM
svm_importances_df = pd.DataFrame({
    'Feature': X_test.columns,
```

```
    'Importance': svm_importance.importances_mean
}).sort_values(by='Importance', ascending=False)

# Plot Permutation Importance for SVM
plt.figure(figsize=(10, 8))
plt.barh(svm_importances_df['Feature'][:10],
svm_importances_df['Importance'][:10], color='orange')
plt.gca().invert_yaxis()
plt.title('Top 10 Feature Importances (SVM - Permutation)')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.savefig("feature_importance_svm.png")
plt.show()

# Permutation Importance for Neural Network
mlp_importance = permutation_importance(mlp_model, X_test_sample,
y_test_sample, scoring='accuracy', n_repeats=n_repeats,
random_state=42)

# DataFrame for Neural Network
mlp_importances_df = pd.DataFrame({
    'Feature': X_test.columns,
    'Importance': mlp_importance.importances_mean
}).sort_values(by='Importance', ascending=False)

# Plot Permutation Importance for Neural Network
plt.figure(figsize=(10, 8))
plt.barh(mlp_importances_df['Feature'][:10],
mlp_importances_df['Importance'][:10], color='purple')
plt.gca().invert_yaxis()
plt.title('Top 10 Feature Importances (Neural Network - Permutation)')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.savefig("feature_importance_mlp.png")
plt.show()
```


Top 10 Feature Importances (SVM - Permutation)
dload
sload
tcprtt
synack
ackdat
smean
dmean
trans_depth
response_body_len
dur
Features
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Importance


Top 10 Feature Importances (Neural Network - Permutation)
dload
sload
total_bytes
rate
dmean
dbytes
sjit
sbytes
sinpkt
swin
Features
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Importance