Anomaly Detection in Network Security

Problem Statement:

> The problem is to develop an effective and efficient anomaly detection system for network security. Anomalies in network traffic can be indicative of security breaches, cyber-attacks, or other malicious activities. Detecting these anomalies is crucial for maintaining a secure network environment and ensuring the confidentiality, integrity, and availability of the data.

Objectives:

- Anomaly Detection: Implement a machine learning model that can accurately detect anomalies in network traffic data.
- False Positive Minimization: Optimize the model to reduce the number of false positive alerts, ensuring that genuine network activity is not flagged incorrectly as an anomaly.
- Real-time Monitoring: Design the system to operate in real-time, continuously monitoring network traffic for any suspicious patterns or behaviors.

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
df = pd.read_csv('NSL-KDD/KDDTest+.txt', delimiter=',')
```

In [3]:
```
df.sample(5)
```

Out[3]:

	0	tcp	private	REJ	0.1	0.2	0.3	0.4	0.5	0.6	...	0.04.1	0.06.1	0.00.3	0.00.4	0.00
1322	0	tcp	other	REJ	0	0	0	0	0	0	...	0.00	1.00	0.00	0.00	0
14661	0	udp	domain_u	SF	36	0	0	0	0	0	...	1.00	0.00	1.00	0.04	0
21344	0	udp	domain_u	SF	44	115	0	0	0	0	...	0.89	0.01	0.01	0.00	0
8427	0	udp	other	SF	1	1	0	0	0	0	...	0.00	1.00	1.00	0.00	0
21490	0	tcp	private	S0	0	0	0	0	0	0	...	0.05	0.08	0.00	0.00	1

5 rows × 43 columns

-> we are using a dataset called 'NSL-KDD', it is a widely used benchmark dataset for evaluating intrusion detection systems, particularly in the domain of network security.

In [4]:
```
df.shape
```

Out[4]:
```
(22543, 43)
```

| Pre-Processing & Data Cleaning

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22543 entries, 0 to 22542
Data columns (total 43 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   0        22543 non-null  int64  
 1   tcp      22543 non-null  object 
 2   private  22543 non-null  object 
 3   REJ      22543 non-null  object 
 4   0.1      22543 non-null  int64  
 5   0.2      22543 non-null  int64  
 6   0.3      22543 non-null  int64  
 7   0.4      22543 non-null  int64  
 8   0.5      22543 non-null  int64  
 9   0.6      22543 non-null  int64  
 10  0.7      22543 non-null  int64  
 11  0.8      22543 non-null  int64  
 12  0.9      22543 non-null  int64  
 13  0.10     22543 non-null  int64  
 14  0.11     22543 non-null  int64  
 15  0.12     22543 non-null  int64  
 16  0.13     22543 non-null  int64  
 17  0.14     22543 non-null  int64  
 18  0.15     22543 non-null  int64  
 19  0.16     22543 non-null  int64  
 20  0.17     22543 non-null  int64  
 21  0.18     22543 non-null  int64  
 22  229      22543 non-null  int64  
 23  10       22543 non-null  int64  
 24  0.00     22543 non-null  float64
 25  0.00.1   22543 non-null  float64
 26  1.00     22543 non-null  float64
 27  1.00.1   22543 non-null  float64
 28  0.04     22543 non-null  float64
 29  0.06     22543 non-null  float64
 30  0.00.2   22543 non-null  float64
 31  255      22543 non-null  int64  
 32  10.1     22543 non-null  int64  
 33  0.04.1   22543 non-null  float64
 34  0.06.1   22543 non-null  float64
 35  0.00.3   22543 non-null  float64
 36  0.00.4   22543 non-null  float64
 37  0.00.5   22543 non-null  float64
 38  0.00.6   22543 non-null  float64
 39  1.00.2   22543 non-null  float64
 40  1.00.3   22543 non-null  float64
 41  neptune  22543 non-null  object 
 42  21       22543 non-null  int64  
dtypes: float64(15), int64(24), object(4)
memory usage: 7.4+ MB
```

-> we have 43 columns

- 4 categorized columns: tcp, private, REJ, neptune
  - these are probably network types
- 39 numeric or continuous features: 0, 0.1, 0.2, ..., 21
- column 21 is probably our target column
  - it has distinct values 21, 11, 15

- Let's assign column names for this dataset

In [6]:
```
column_names = ['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'd
                'wrong_fragment', 'urgent', 'hot', 'num_failed_logins', 'logged
                'root_shell', 'su_attempted', 'num_root', 'num_file_creations',
                'num_outbound_cmds', 'is_host_login', 'is_guest_login', 'unknow
                'count', 'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror
                'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate', 'dst_ho
                'dst_host_same_srv_rate', 'dst_host_diff_srv_rate', 'dst_host_s
                'dst_host_srv_diff_host_rate', 'dst_host_serror_rate', 'dst_hos
                'dst_host_rerror_rate', 'neptune', 'target']
```

In [7]:
```
df.columns = column_names
```

In [8]:
```
df.sample(5)
```

Out[8]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urge
8786	0	tcp	private	S0	0	0	0	0	
10341	2067	tcp	http	RSTR	56504	0	0	0	
4786	0	tcp	private	S0	0	0	0	0	
14979	0	tcp	imap4	RSTO	0	44	0	0	
13611	0	tcp	smtp	SF	1227	390	0	0	

5 rows × 43 columns

In the NSL-KDD dataset, the target column contains discrete numerical values that correspond to different types of network attacks or classes. Here's a breakdown of some common labels found in the target column based on the provided information:

21: This label often represents the "normal" or benign network traffic, indicating that the network activity is considered normal and not malicious.

15: This label could represent a specific type of network attack or intrusion, such as a denial-of-service (DoS) attack.

11: This label could represent another type of network attack or intrusion, possibly related to unauthorized access or probing.

- Let's scale our data using MinMaxScalar

In [9]:
```
categorical_cols = ['protocol_type', 'service', 'flag', 'neptune']
df2 = pd.get_dummies(df, columns=categorical_cols)
```

In [10]:
```
numeric_cols = ['duration', 'src_bytes', 'dst_bytes', 'count', 'srv_count', 'se
                'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv
                'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count',
                'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_
                'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_
```

In [11]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [12]:
```
scaler = MinMaxScaler()
df2[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

In [13]:
```
# Assume 21 represents normal (benign) traffic, and other values (e.g., 15, 11)

df2['target'] = df['target'].apply(lambda x: 1 if x != 21 else 0)
# 1 for attacks, 0 for normal
```

In [14]:
```
df.head
```

Out[14]:

```
<bound method NDFrame.head of        duration protocol_type   service  flag
src_bytes  dst_bytes  land  \
0             0           tcp   private   REJ          0          0     0
1             2           tcp  ftp_data    SF      12983          0     0
2             0          icmp     eco_i    SF         20          0     0
3             1           tcp    telnet  RSTO          0         15     0
4             0           tcp      http    SF        267      14515     0
...         ...           ...       ...   ...        ...        ...   ...
22538         0           tcp      smtp    SF        794        333     0
22539         0           tcp      http    SF        317        938     0
22540         0           tcp      http    SF      54540       8314     0
22541         0           udp  domain_u    SF         42         42     0
22542         0           tcp    sunrpc   REJ          0          0     0

       wrong_fragment  urgent  hot  ...  dst_host_srv_count  \
0                   0       0    0  ...                0.00
1                   0       0    0  ...                0.61
2                   0       0    0  ...                1.00
3                   0       0    0  ...                0.31
4                   0       0    0  ...                1.00
...               ...     ...  ...  ...                 ...
22538               0       0    0  ...                0.72
22539               0       0    0  ...                1.00
22540               0       0    2  ...                1.00
22541               0       0    0  ...                0.99
22542               0       0    0  ...                0.08

       dst_host_same_srv_rate  dst_host_diff_srv_rate  \
0                        0.06                    0.00
1                        0.04                    0.61
2                        0.00                    1.00
3                        0.17                    0.03
4                        0.00                    0.01
...                       ...                     ...
22538                    0.06                    0.01
22539                    0.00                    0.01
22540                    0.00                    0.00
22541                    0.01                    0.00
22542                    0.03                    0.00

       dst_host_same_src_port_rate  dst_host_srv_diff_host_rate  \
0                             0.00                         0.00
1                             0.02                         0.00
2                             0.28                         0.00
3                             0.02                         0.00
4                             0.03                         0.01
...                            ...                          ...
22538                         0.01                         0.01
22539                         0.01                         0.01
22540                         0.00                         0.00
22541                         0.00                         0.00
22542                         0.00                         0.00

       dst_host_serror_rate  dst_host_srv_serror_rate  dst_host_rerror_rate
\
0                       0.0                      1.00                  1.00
1                       0.0                      0.00                  0.00
```

```
2                    0.0                    0.00                   0.00
3                    0.0                    0.83                   0.71
4                    0.0                    0.00                   0.00
...                  ...                     ...                    ...
22538                0.0                    0.00                   0.00
22539                0.0                    0.00                   0.00
22540                0.0                    0.07                   0.07
22541                0.0                    0.00                   0.00
22542                0.0                    0.44                   1.00

       neptune  target
0      neptune      21
1       normal      21
2        saint      15
3        mscan      11
4       normal      21
...        ...     ...
22538   normal      21
22539   normal      21
22540     back      15
22541   normal      21
22542    mscan      14

[22543 rows x 43 columns]>
```

In [15]:
```
df2.head
```

Out[15]:
```
<bound method NDFrame.head of        duration     src_bytes  dst_bytes  land
wrong_fragment  urgent  hot  \
0      0.000000  0.000000e+00   0.000000     0              0       0    0
1      0.000035  2.066513e-04   0.000000     0              0       0    0
2      0.000000  3.183413e-07   0.000000     0              0       0    0
3      0.000017  0.000000e+00   0.000011     0              0       0    0
4      0.000000  4.249857e-06   0.010784     0              0       0    0
...         ...           ...        ...   ...            ...     ...  ...
22538  0.000000  1.263815e-05   0.000247     0              0       0    0
22539  0.000000  5.045710e-06   0.000697     0              0       0    0
22540  0.000000  8.681168e-04   0.006177     0              0       0    2
22541  0.000000  6.685168e-07   0.000031     0              0       0    0
22542  0.000000  0.000000e+00   0.000000     0              0       0    0

       num_failed_logins  logged_in  num_compromised  ...  \
0                      0          0                0  ...
1                      0          0                0  ...
2                      0          0                0  ...
3                      0          0                0  ...
4                      0          1                0  ...
...                  ...        ...              ...  ...
22538                  0          1                0  ...
22539                  0          1                0  ...
22540                  0          1                1  ...
22541                  0          0                0  ...
22542                  0          0                0  ...

       neptune_snmpgetattack  neptune_snmpguess  neptune_sqlattack  \
0                          0                  0                  0
1                          0                  0                  0
2                          0                  0                  0
3                          0                  0                  0
4                          0                  0                  0
...                      ...                ...                ...
22538                      0                  0                  0
22539                      0                  0                  0
22540                      0                  0                  0
22541                      0                  0                  0
22542                      0                  0                  0

       neptune_teardrop  neptune_udpstorm  neptune_warezmaster  neptune_worm
\
0                     0                 0                    0             0
1                     0                 0                    0             0
2                     0                 0                    0             0
3                     0                 0                    0             0
4                     0                 0                    0             0
...                 ...               ...                  ...           ...
22538                 0                 0                    0             0
22539                 0                 0                    0             0
22540                 0                 0                    0             0
22541                 0                 0                    0             0
22542                 0                 0                    0             0

       neptune_xlock  neptune_xsnoop  neptune_xterm
0                  0               0              0
1                  0               0              0
```

```
2                 0               0               0
3                 0               0               0
4                 0               0               0
...             ...             ...             ...
22538             0               0               0
22539             0               0               0
22540             0               0               0
22541             0               0               0
22542             0               0               0

[22543 rows x 155 columns]>
```

-> we have no missing or duplicate values in out dataset so we dont't have to process those aspects of the data

---

| Exploratory Data Analysis

In [16]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [17]:
```
df2.describe()
```

Out[17]:

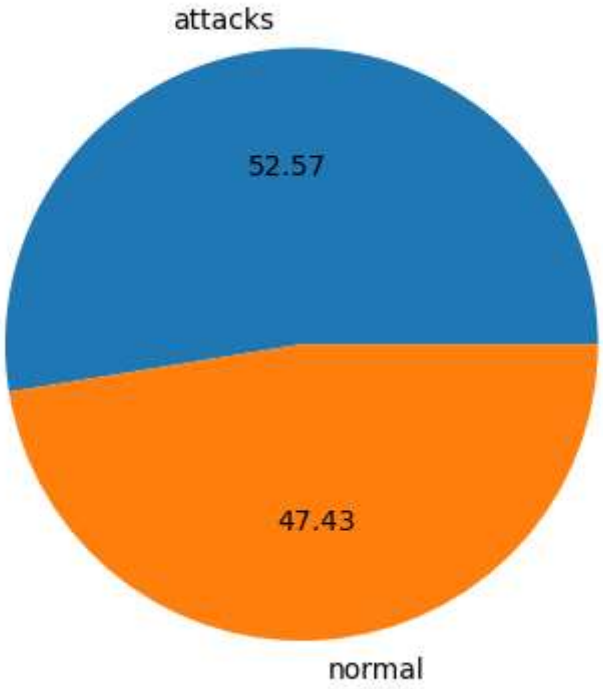
	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	2
count	22543.000000	2.254300e+04	22543.000000	22543.000000	22543.000000	22543.000000	
mean	0.003792	1.654724e-04	0.001528	0.000311	0.008428	0.000710	
std	0.024382	7.525540e-03	0.015766	0.017619	0.142602	0.036474	
min	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	8.595216e-07	0.000034	0.000000	0.000000	0.000000	
75%	0.000000	4.568198e-06	0.000447	0.000000	0.000000	0.000000	
max	1.000000	1.000000e+00	1.000000	1.000000	3.000000	3.000000	

8 rows × 155 columns

In [18]:
```
df2['target'].value_counts()
```

Out[18]:
```
1    11850
0    10693
Name: target, dtype: int64
```

In [19]:
```
plt.pie(df2['target'].value_counts(), labels=['attacks','normal'],autopct="%0.2
plt.show()
```

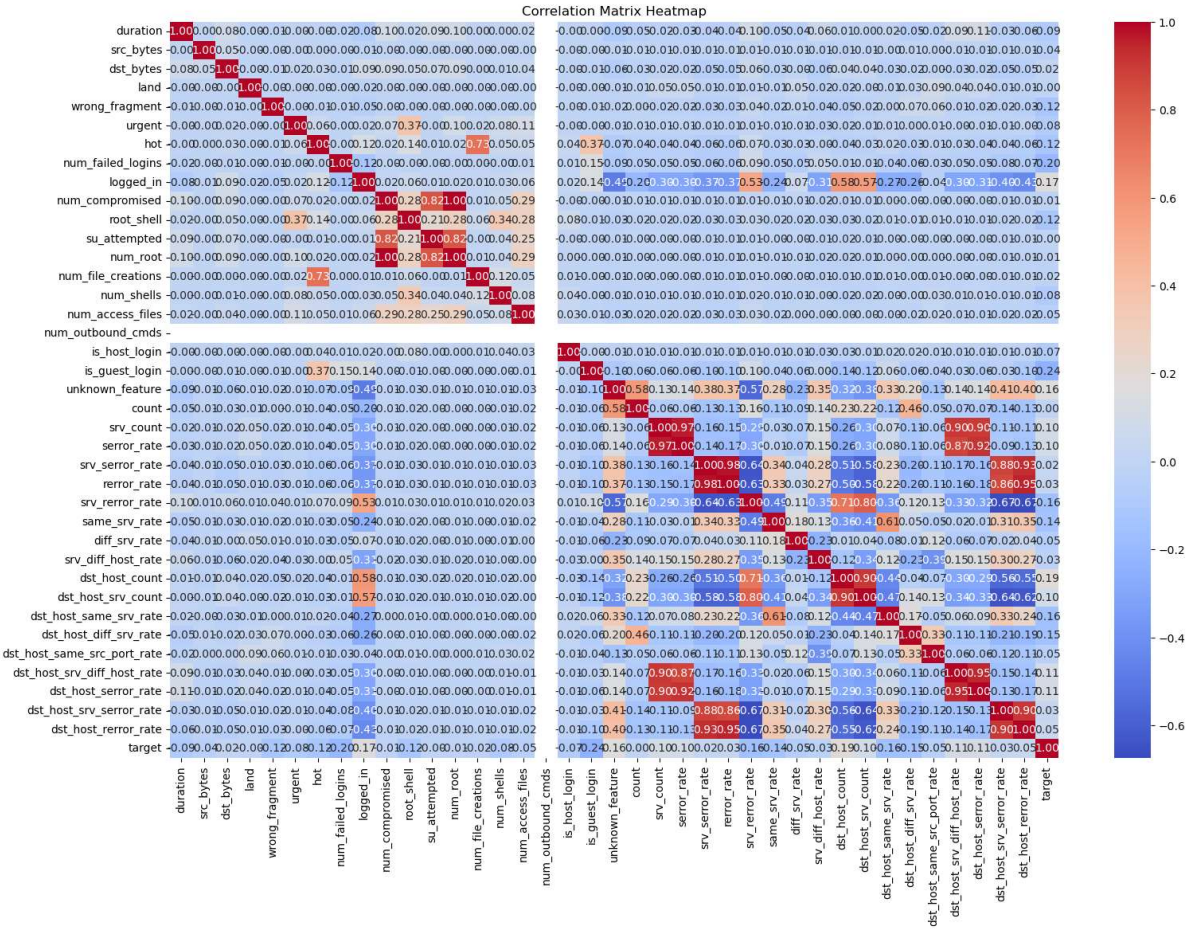


-> NSL-KDD is a data set suggested to solve some of the inherent problems of the KDD'99 data set like imabalanced data. KDD is more balanced maintaining a 52.57% of 'attack' or '1' data.

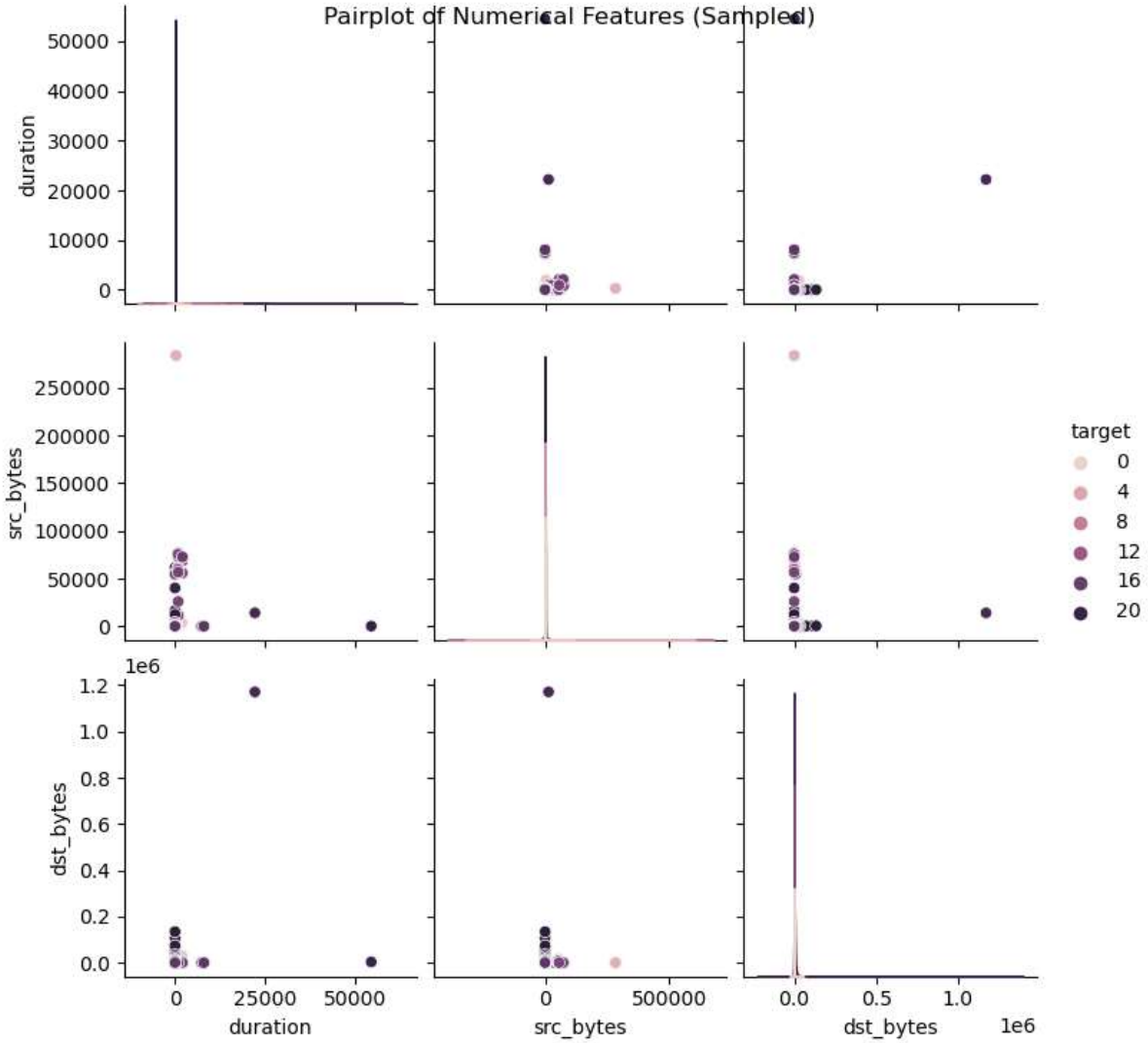
In [20]:
```
plt.figure(figsize=(18, 12))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
C:\Users\Arindal Char\AppData\Local\Temp\ipykernel_16188\3922245334.py:3: Fut
ureWarning: The default value of numeric_only in DataFrame.corr is deprecate
d. In a future version, it will default to False. Select only valid columns o
r specify the value of numeric_only to silence this warning.
  sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
```

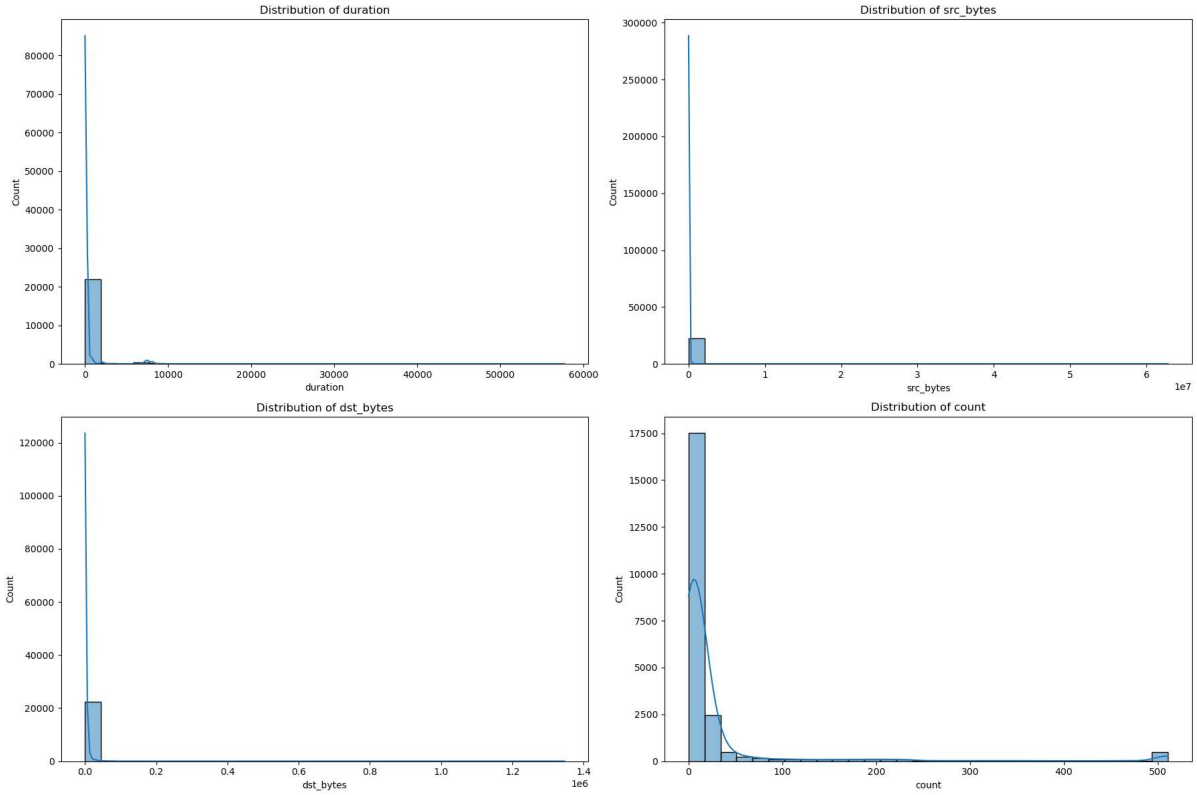


In [21]:
```
sns.pairplot(df.sample(1000), vars=numeric_cols[:3], hue='target')
plt.suptitle('Pairplot of Numerical Features (Sampled)')
plt.show()
```



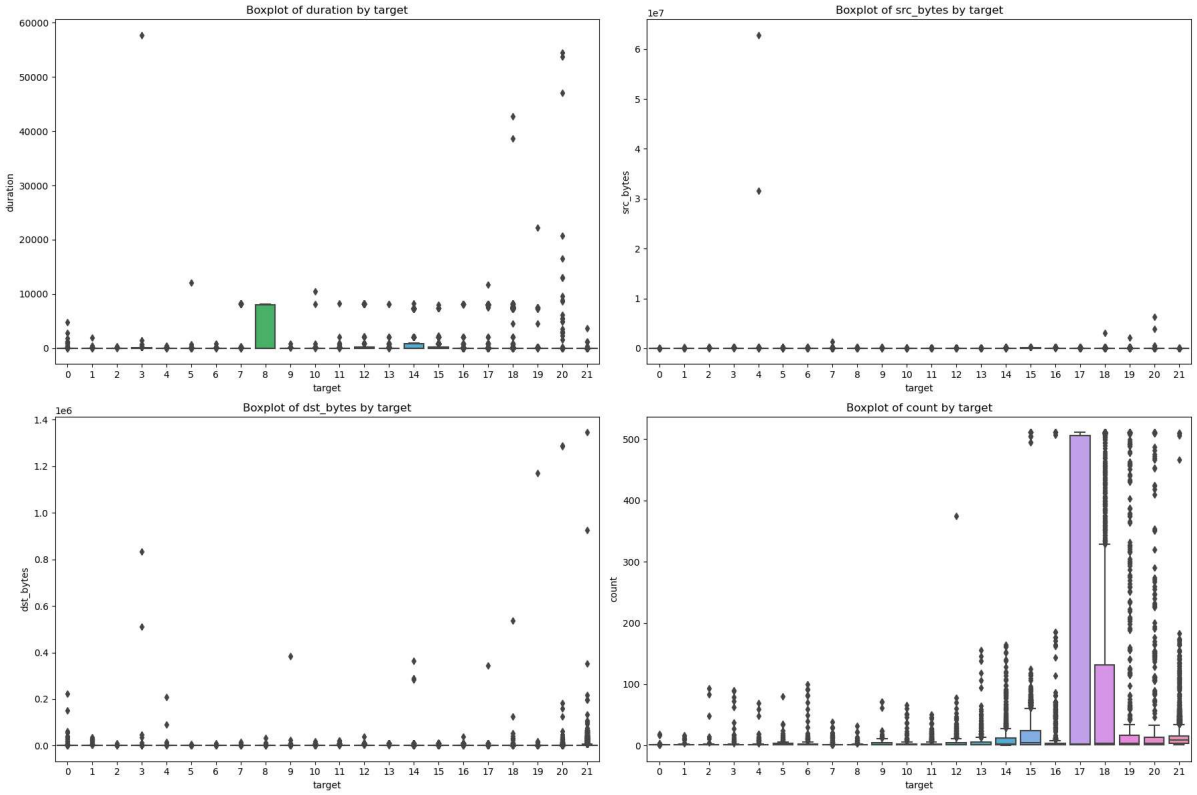
In [22]:
```
# Distribution of numerical features

plt.figure(figsize=(18, 12))
for i, col in enumerate(numeric_cols[:4]):  # Displaying the first 4 numerical
    plt.subplot(2, 2, i+1)
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



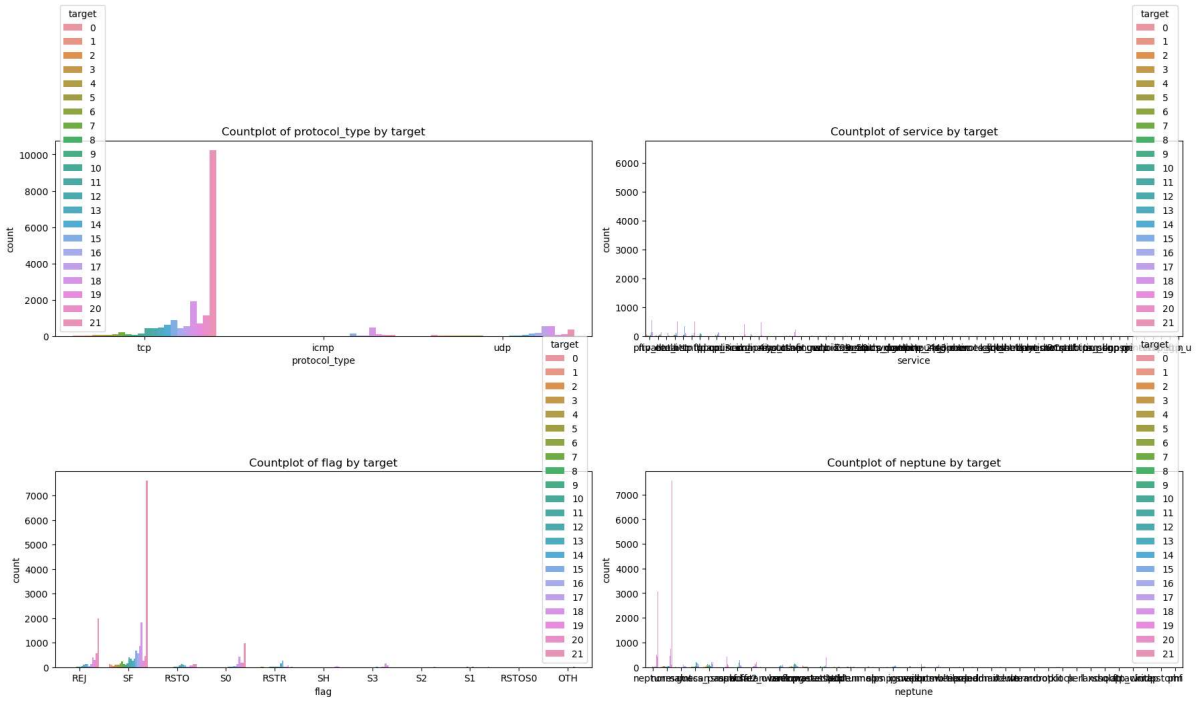
In [23]:
```
# Boxplots for numerical features by target

plt.figure(figsize=(18, 12))
for i, col in enumerate(numeric_cols[:4]):
    plt.subplot(2, 2, i+1)
    sns.boxplot(x='target', y=col, data=df)
    plt.title(f'Boxplot of {col} by target')
plt.tight_layout()
plt.show()
```



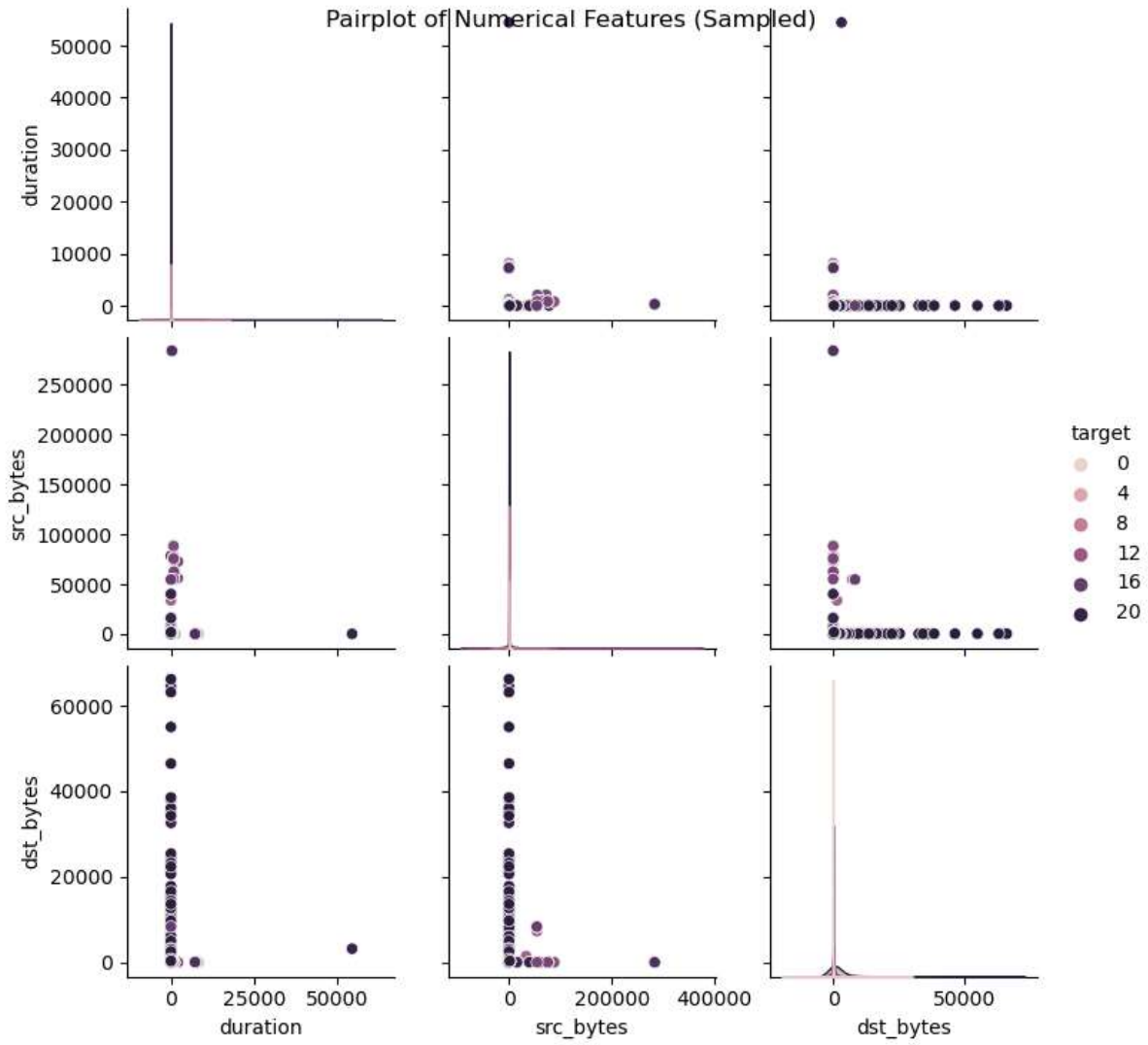
In [24]:
```
# Count plot of categorical features

plt.figure(figsize=(18, 10))
for i, col in enumerate(categorical_cols):
    plt.subplot(2, 2, i+1)  # Use 2 rows and 2 subplots in each row
    sns.countplot(x=col, data=df, hue='target')
    plt.title(f'Countplot of {col} by target')
plt.tight_layout()
plt.show()
```



In [25]:
```
# Pairplot of a few numerical features

sns.pairplot(df.sample(1000), vars=numeric_cols[:3], hue='target')
plt.suptitle('Pairplot of Numerical Features (Sampled)')
plt.show()
```



| Feature Engineering

In [26]:
```
# Create binary features for each protocol type

df2['is_protocol_tcp'] = (df2['protocol_type_tcp'] == 1).astype(int)
df2['is_protocol_udp'] = (df2['protocol_type_udp'] == 1).astype(int)
df2['is_protocol_icmp'] = (df2['protocol_type_icmp'] == 1).astype(int)
```

In [27]:
```
# Create binary features for each flag

df2['is_flag_SF'] = (df2['flag_SF'] == 1).astype(int)
df2['is_flag_S0'] = (df2['flag_S0'] == 1).astype(int)
df2['is_flag_REJ'] = (df2['flag_REJ'] == 1).astype(int)
```

In [28]:
```
# Aggregate services into categories

web_services = ['http', 'https', 'smtp', 'ftp', 'ssh', 'ssl']
df2['service_category'] = df['service'].apply(lambda x: 'Web' if x in web_servi
```

In [29]:
```
df2.head
```

Out[29]:

```
<bound method NDFrame.head of        duration     src_bytes  dst_bytes  land
wrong_fragment  urgent  hot  \
0      0.000000  0.000000e+00   0.000000     0              0       0    0
1      0.000035  2.066513e-04   0.000000     0              0       0    0
2      0.000000  3.183413e-07   0.000000     0              0       0    0
3      0.000017  0.000000e+00   0.000011     0              0       0    0
4      0.000000  4.249857e-06   0.010784     0              0       0    0
...         ...           ...        ...   ...            ...     ...  ...
22538  0.000000  1.263815e-05   0.000247     0              0       0    0
22539  0.000000  5.045710e-06   0.000697     0              0       0    0
22540  0.000000  8.681168e-04   0.006177     0              0       0    2
22541  0.000000  6.685168e-07   0.000031     0              0       0    0
22542  0.000000  0.000000e+00   0.000000     0              0       0    0

       num_failed_logins  logged_in  num_compromised  ...  neptune_xlock  \
0                      0          0                0  ...              0
1                      0          0                0  ...              0
2                      0          0                0  ...              0
3                      0          0                0  ...              0
4                      0          1                0  ...              0
...                  ...        ...              ...  ...            ...
22538                  0          1                0  ...              0
22539                  0          1                0  ...              0
22540                  0          1                1  ...              0
22541                  0          0                0  ...              0
22542                  0          0                0  ...              0

       neptune_xsnoop  neptune_xterm  is_protocol_tcp  is_protocol_udp  \
0                   0              0                1                0
1                   0              0                1                0
2                   0              0                0                0
3                   0              0                1                0
4                   0              0                1                0
...               ...            ...              ...              ...
22538               0              0                1                0
22539               0              0                1                0
22540               0              0                1                0
22541               0              0                0                1
22542               0              0                1                0

       is_protocol_icmp  is_flag_SF  is_flag_S0  is_flag_REJ  service_categor
y
0                     0           0           0            1            Other
s
1                     0           1           0            0            Other
s
2                     1           1           0            0            Other
s
3                     0           0           0            0            Other
s
4                     0           1           0            0               We
b
...                 ...         ...         ...          ...
...
22538                 0           1           0            0               We
b
22539                 0           1           0            0               We
```

```
b
22540                0          1          0          0                We
b
22541                0          1          0          0             Other
s
22542                0          0          0          1             Other
s

[22543 rows x 162 columns]>
```

We currently have a lot of features, so lets select the top 10 features using 'RFE'

In [110]:
```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
```

In [111]:
```
random_model = RandomForestClassifier(random_state=42)
```

In [114]:
```
n_features_to_select = 10  # Adjust this based on your preference
rfe = RFE(estimator=rf_model, n_features_to_select=n_features_to_select)
fit = rfe.fit(X_train_encoded, y_train)
```

In [115]:
```
selected_features = [X_train_encoded.columns[i] for i in range(len(rfe.support_
print("Selected Features:", selected_features)
```

```
Selected Features: ['dst_bytes', 'unknown_feature', 'dst_host_count', 'dst_ho
st_srv_count', 'dst_host_same_srv_rate', 'dst_host_srv_serror_rate', 'service
_http', 'service_private', 'neptune_neptune', 'neptune_normal']
```

We'll later use these features to train the best performing model

---

| Model Building

In [30]:
```
from sklearn.model_selection import train_test_split
```

In [31]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_ma
from sklearn.neural_network import MLPClassifier
```

In [32]:
```
X = df2.drop('target', axis=1)  # Features
y = df2['target']  # Target variable
```

In [33]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
```

In [34]:
```
# Perform one-hot encoding on categorical features

X_train_encoded = pd.get_dummies(X_train, columns=['service_category'])
X_test_encoded = pd.get_dummies(X_test, columns=['service_category'])
```

Random Forest:

In [35]:
```
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_encoded, y_train)
```

Out[35]:
```
RandomForestClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Support Vector Machine:

In [36]:
```
svm_model = SVC(random_state=42)
svm_model.fit(X_train_encoded, y_train)
```

Out[36]:
```
SVC(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Logistic Regression:

In [37]:
```
log_reg_model = LogisticRegression(random_state=42, max_iter=10000)  # Increase
log_reg_model.fit(X_train_encoded, y_train)
```

Out[37]:
```
LogisticRegression(max_iter=10000, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Neural Network (Multi-layer Perceptron):

In [38]:
```
nn_model = MLPClassifier(random_state=42)
nn_model.fit(X_train_encoded, y_train)
```

Out[38]:
```
MLPClassifier(random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [39]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_
```

In [40]:
```
rf_predictions = rf_model.predict(X_test_encoded)
svm_predictions = svm_model.predict(X_test_encoded)
log_reg_predictions = log_reg_model.predict(X_test_encoded)
nn_predictions = nn_model.predict(X_test_encoded)
```

In [41]:
```
rf_accuracy = accuracy_score(y_test, rf_predictions)
svm_accuracy = accuracy_score(y_test, svm_predictions)
log_reg_accuracy = accuracy_score(y_test, log_reg_predictions)
nn_accuracy = accuracy_score(y_test, nn_predictions)
```

In [42]:
```
print("Accuracy Scores:")
print("Random Forest:", rf_accuracy)
print("SVM:", svm_accuracy)
print("Logistic Regression:", log_reg_accuracy)
print("Neural Network:", nn_accuracy)
print("\n")
```

```
Accuracy Scores:
Random Forest: 0.9831448214681747
SVM: 0.8148148148148148
Logistic Regression: 0.9634065202927479
Neural Network: 0.9733865602129075
```

In [43]:
```
models = ['Random Forest', 'SVM', 'Logistic Regression', 'Neural Network']
predictions = [rf_predictions, svm_predictions, log_reg_predictions, nn_predict
```

In [44]:
```
for model_name, preds in zip(models, predictions):
    precision = precision_score(y_test, preds, average='weighted')
    recall = recall_score(y_test, preds, average='weighted')
    confusion_mat = confusion_matrix(y_test, preds)
    fp = confusion_mat[0, 1] / (confusion_mat[0, 0] + confusion_mat[0, 1])  # F
    roc_auc = roc_auc_score(y_test, preds)

    print(f"Metrics for {model_name}:")
    print("Precision:", precision)
    print("Recall:", recall)
    print("False Positive Rate:", fp)
    print("Area under ROC curve:", roc_auc)
    print("Confusion Matrix:")
    print(confusion_mat)
    print("\n")
```

```
Metrics for Random Forest:
Precision: 0.9831959203923114
Recall: 0.9831448214681747
False Positive Rate: 0.02430886558627264
Area under ROC curve: 0.982660996489319
Confusion Matrix:
[[2047   51]
 [  25 2386]]


Metrics for SVM:
Precision: 0.8167135584564832
Recall: 0.8148148148148148
False Positive Rate: 0.16682554814108674
Area under ROC curve: 0.8160065540090916
Confusion Matrix:
[[1748  350]
 [ 485 1926]]


Metrics for Logistic Regression:
Precision: 0.9634039477930217
Recall: 0.9634065202927479
False Positive Rate: 0.04003813155386082
Area under ROC curve: 0.963182925098225
Confusion Matrix:
[[2014   84]
 [  81 2330]]


Metrics for Neural Network:
Precision: 0.9734284892628444
Recall: 0.9733865602129075
False Positive Rate: 0.03479504289799809
Area under ROC curve: 0.9728554856020172
Confusion Matrix:
[[2025   73]
 [  47 2364]]
```

Comparison and Analysis:

- **Precision and Recall**:
  - Random Forest has the highest precision and recall, indicating that it makes fewer false positive and false negative predictions compared to the other models.
  - Logistic Regression and Neural Network also have high precision and recall.
  - SVM has relatively lower precision and recall compared to the other models.
- **False Positive Rate**:
  - Random Forest has the lowest false positive rate, indicating that it has a good ability to avoid false alarms.
  - Logistic Regression and Neural Network also have low false positive rates.
  - SVM has the highest false positive rate, indicating it has a higher tendency to produce false alarms.
- **Area under ROC curve (AUC-ROC)**:
  - Random Forest and Logistic Regression have the highest AUC-ROC, suggesting better overall classification performance.
  - Neural Network also has a high AUC-ROC.
  - SVM has a slightly lower AUC-ROC compared to the other models.
- **Confusion Matrix**:
  - Random Forest has the lowest count of misclassifications (FP and FN) in the confusion matrix, indicating superior classification performance.
  - SVM has a higher count of misclassifications compared to Random Forest.
  - Logistic Regression and Neural Network also show good performance with relatively low misclassification counts.

> In summary, based on these metrics, Random Forest appears to be the best-performing model among the ones evaluated, followed by Logistic Regression and Neural Network. SVM, while performing reasonably, has slightly lower metrics compared to the other models in this evaluation.

Let's train a Random Forest model on our previously selected features

In [116]:
```
X_train_selected = X_train_encoded[selected_features]
X_test_selected = X_test_encoded[selected_features]
```

In [117]:
```
rf_model_selected = RandomForestClassifier(random_state=42)
rf_model_selected.fit(X_train_selected, y_train)
```

Out[117]:
```
RandomForestClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [118]:
```
rf_predictions_selected = rf_model_selected.predict(X_test_selected)
```

In [119]:
```
rf_accuracy_selected = accuracy_score(y_test, rf_predictions_selected)
print("Accuracy Score with Selected Features:", rf_accuracy_selected)
```

```
Accuracy Score with Selected Features: 0.9691727655799512
```

In [121]:
```
print("Classification Report:")
print(classification_report(y_test, rf_predictions_selected), "\n")
print("Confusion Matrix:")
print(confusion_matrix(y_test, rf_predictions_selected))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      2098
           1       0.97      0.97      0.97      2411

    accuracy                           0.97      4509
   macro avg       0.97      0.97      0.97      4509
weighted avg       0.97      0.97      0.97      4509


Confusion Matrix:
[[2029   69]
 [  70 2341]]
```

The accuracy of this model is a bit lower than the rf_model, but we want to deploy a model, we need to keep less features

| Fine Tuning

In [45]:
```
from sklearn.model_selection import GridSearchCV
```

In [46]:
```
rf_params = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30]
}
```

In [59]:
```
svm_params = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid']
}
```

In [60]:
```
log_reg_params = {
    'C': [0.1, 1, 10],
    'max_iter': [5000, 7000, 10000, 12500, 15000]
}
```

In [61]:
```
nn_params = {
    'hidden_layer_sizes': [(64, 64), (128, 128)],
    'activation': ['relu', 'tanh'],
    'max_iter': [5000, 7000, 10000, 12500, 15000]
}
```

In [62]:
```
rf_grid = GridSearchCV(RandomForestClassifier(random_state=42), param_grid=rf_p
svm_grid = GridSearchCV(SVC(random_state=42), param_grid=svm_params, cv=3)
log_reg_grid = GridSearchCV(LogisticRegression(random_state=42), param_grid=log
nn_grid = GridSearchCV(MLPClassifier(random_state=42), param_grid=nn_params, cv
```

In [56]:
```
rf_grid.fit(X_train_encoded, y_train)
```

Out[56]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [None, 10, 20, 30],
                         'n_estimators': [50, 100, 150]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [65]:
```
svm_grid.fit(X_train_encoded, y_train)
```

Out[65]:
```
GridSearchCV(cv=3, estimator=SVC(random_state=42),
             param_grid={'C': [0.1, 1, 10],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [63]:
```
log_reg_grid.fit(X_train_encoded, y_train)
```

Out[63]:
```
GridSearchCV(cv=3, estimator=LogisticRegression(random_state=42),
             param_grid={'C': [0.1, 1, 10],
                         'max_iter': [5000, 7000, 10000, 12500, 15000]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [64]:
```
nn_grid.fit(X_train_encoded, y_train)
```

Out[64]:
```
GridSearchCV(cv=3, estimator=MLPClassifier(random_state=42),
             param_grid={'activation': ['relu', 'tanh'],
                         'hidden_layer_sizes': [(64, 64), (128, 128)],
                         'max_iter': [5000, 7000, 10000, 12500, 15000]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Let's make a Voting Classifier for the four models

In [71]:
```
from sklearn.ensemble import VotingClassifier
```

In [72]:
```
models = [('Random Forest', rf_model), ('SVM', svm_model), ('Logistic Regressic
```

In [73]:
```
voting_clf = VotingClassifier(estimators=models, voting='hard')
```

In [74]:
```
voting_clf.fit(X_train_encoded, y_train)
```

Out[74]:
```
VotingClassifier(estimators=[('Random Forest',
                              RandomForestClassifier(random_state=42)),
                             ('SVM', SVC(random_state=42)),
                             ('Logistic Regression',
                              LogisticRegression(max_iter=10000,
                                                 random_state=42)),
                             ('Neural Network',
                              MLPClassifier(random_state=42))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [75]:
```
ensemble_predictions = voting_clf.predict(X_test_encoded)
```

In [76]:
```
ensemble_accuracy = accuracy_score(y_test, ensemble_predictions)
```

In [77]:
```
print("Ensemble (Voting Classifier) Accuracy:", ensemble_accuracy)
print("Classification Report for Ensemble:")
print(classification_report(y_test, ensemble_predictions))
print("Confusion Matrix for Ensemble:")
print(confusion_matrix(y_test, ensemble_predictions))
```

```
Ensemble (Voting Classifier) Accuracy: 0.9711687735639831
Classification Report for Ensemble:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      2098
           1       0.97      0.97      0.97      2411

    accuracy                           0.97      4509
   macro avg       0.97      0.97      0.97      4509
weighted avg       0.97      0.97      0.97      4509

Confusion Matrix for Ensemble:
[[2036   62]
 [  68 2343]]
```

- **Accuracy:** The accuracy of the model is approximately 97.12%.
- **Precision:** Precision is a measure of the accuracy of the classifier in predicting the positive class. For both classes (0 and 1), the precision is around 97%.
- **Sensivity:** Recall measures the proportion of actual positive instances that were correctly predicted by the model. It's also around 97% for both classes.
- **The F1-score** is the weighted average of precision and recall. It's a good way to show that a classifer has a good value for both recall and precision. It's also around 97% for both classes.

Confusion Matrix:

- The confusion matrix shows the actual vs. predicted labels.
- The diagonal elements represent the number of correctly classified instances for each class.
- Off-diagonal elements are the misclassifications. For example, in the top right element, it shows that 62 instances of class 0 were misclassified as class 1.
- Bottom left element: True negatives (TN) - Instances correctly classified as class 0.
- Bottom right element: True positives (TP) - Instances correctly classified as class 1.

| Predictive Model

In [99]:
```
from sklearn.preprocessing import LabelEncoder
```

In [100]:
```
print("Sample predictions using Random Forest:")
print(rf_predictions[:10])
```

```
Sample predictions using Random Forest:
[1 1 1 0 1 1 1 1 0 1]
```

In [101]:
```
predicted_probabilities_rf = rf_model.predict_proba(X_test_encoded)
print("Sample predicted probabilities using Random Forest:")
print(predicted_probabilities_rf[:10])
```

```
Sample predicted probabilities using Random Forest:
[[0.00000000e+00 1.00000000e+00]
 [1.50000000e-01 8.50000000e-01]
 [0.00000000e+00 1.00000000e+00]
 [9.99950134e-01 4.98655517e-05]
 [0.00000000e+00 1.00000000e+00]
 [0.00000000e+00 1.00000000e+00]
 [0.00000000e+00 1.00000000e+00]
 [0.00000000e+00 1.00000000e+00]
 [1.00000000e+00 0.00000000e+00]
 [0.00000000e+00 1.00000000e+00]]
```

| Model Serialization

In [105]:
```
import pickle
```

In [106]:
```
model_filename = 'random_forest_model.pkl'
```

In [107]:
```
with open(model_filename, 'wb') as file:
    pickle.dump(rf_model, file)
```

In [108]:
```
print(f"Random Forest model saved to {model_filename}")
```

```
Random Forest model saved to random_forest_model.pkl
```

In [109]:
```
pickle.dump(scaler, open('normalization.pkl','wb'))
```

In [122]:
```
pickle.dump(rf_model_selected, open('selected_randomforest.pkl','wb'))
```

In [ ]:
```

```