In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.decomposition import PCA
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from scipy import stats
from sklearn.decomposition import PCA
from scipy.stats import chi2_contingency
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostir
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import classification_report, roc_auc_score
import warnings
```

In [2]:
```
# Suppress warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
# Load the cleaned dataset
df = pd.read_csv("Network_anomaly_data.csv")

# Check the first few rows of the data
print(df.head())

# Get general information about the dataset
print(df.info())
```

```
   duration protocoltype   service flag  srcbytes  dstbytes  land
\
0         0          tcp  ftp_data   SF       491         0     0
1         0          udp     other   SF       146         0     0
2         0          tcp   private   S0         0         0     0
3         0          tcp      http   SF       232      8153     0
4         0          tcp      http   SF       199       420     0
```

```
   wrongfragment  urgent  hot  ...  dsthostsamesrvrate  dsthostdif
fsrvrate  \
0              0       0    0  ...                0.17
0.03
1              0       0    0  ...                0.00
0.60
2              0       0    0  ...                0.10
0.05
3              0       0    0  ...                1.00
0.00
4              0       0    0  ...                1.00
0.00

   dsthostsamesrcportrate  dsthostsrvdiffhostrate  dsthostserrorra
te  \
0                    0.17                    0.00                0.
00
1                    0.88                    0.00                0.
00
2                    0.00                    0.00                1.
00
3                    0.03                    0.04                0.
03
4                    0.00                    0.00                0.
00

   dsthostsrvserrorrate  dsthostrerrorrate  dsthostsrvrerrorrate
attack  \
0                  0.00               0.05                  0.00
normal
1                  0.00               0.00                  0.00
normal
2                  1.00               0.00                  0.00
neptune
3                  0.01               0.00                  0.01
normal
4                  0.00               0.00                  0.00
normal

   lastflag
0        20
1        15
2        19
3        21
4        21

[5 rows x 43 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125973 entries, 0 to 125972
Data columns (total 43 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   duration                125973 non-null  int64
 1   protocoltype            125973 non-null  object
```

```
 2   service               125973 non-null  object
 3   flag                  125973 non-null  object
 4   srcbytes              125973 non-null  int64
 5   dstbytes              125973 non-null  int64
 6   land                  125973 non-null  int64
 7   wrongfragment         125973 non-null  int64
 8   urgent                125973 non-null  int64
 9   hot                   125973 non-null  int64

 10  numfailedlogins       125973 non-null  int64
 11  loggedin              125973 non-null  int64
 12  numcompromised        125973 non-null  int64
 13  rootshell             125973 non-null  int64
 14  suattempted           125973 non-null  int64
 15  numroot               125973 non-null  int64
 16  numfilecreations      125973 non-null  int64
 17  numshells             125973 non-null  int64
 18  numaccessfiles        125973 non-null  int64
 19  numoutboundcmds       125973 non-null  int64
 20  ishostlogin           125973 non-null  int64
 21  isguestlogin          125973 non-null  int64
 22  count                 125973 non-null  int64
 23  srvcount              125973 non-null  int64
 24  serrorrate            125973 non-null  float64
 25  srvserrorrate         125973 non-null  float64
 26  rerrorrate            125973 non-null  float64
 27  srvrerrorrate         125973 non-null  float64
 28  samesrvrate           125973 non-null  float64
 29  diffsrvrate           125973 non-null  float64
 30  srvdiffhostrate       125973 non-null  float64
 31  dsthostcount          125973 non-null  int64
 32  dsthostsrvcount       125973 non-null  int64
 33  dsthostsamesrvrate    125973 non-null  float64
 34  dsthostdiffsrvrate    125973 non-null  float64
 35  dsthostsamesrcportrate  125973 non-null  float64
 36  dsthostsrvdiffhostrate  125973 non-null  float64
 37  dsthostserrorrate     125973 non-null  float64
 38  dsthostsrvserrorrate  125973 non-null  float64
 39  dsthostrerrorrate     125973 non-null  float64
 40  dsthostsrvrerrorrate  125973 non-null  float64
 41  attack                125973 non-null  object
 42  lastflag              125973 non-null  int64
dtypes: float64(15), int64(24), object(4)
memory usage: 41.3+ MB
None
```

In [4]:
```
# Display missing values before handling
print("Missing values before handling:")
print(df.isnull().sum())
```

```
Missing values before handling:
duration                 0
protocoltype             0
service                  0
flag                     0
srcbytes                 0
dstbytes                 0
land                     0
wrongfragment            0
urgent                   0
hot                      0
numfailedlogins          0
loggedin                 0
numcompromised           0
rootshell                0
suattempted              0
numroot                  0
numfilecreations         0
numshells                0
numaccessfiles           0
numoutboundcmds          0
ishostlogin              0
isguestlogin             0
count                    0
srvcount                 0
serrorrate               0
srvserrorrate            0
rerrorrate               0
srvrerrorrate            0
samesrvrate              0
diffsrvrate              0
srvdiffhostrate          0
dsthostcount             0
dsthostsrvcount          0
dsthostsamesrvrate       0
dsthostdiffsrvrate       0
dsthostsamesrcportrate   0
dsthostsrvdiffhostrate   0
dsthostserrorrate        0
dsthostsrvserrorrate     0
dsthostrerrorrate        0
dsthostsrvrerrorrate     0
attack                   0
lastflag                 0
dtype: int64
```

In [5]:
```
# Separate numerical and categorical columns
numerical_columns = df.select_dtypes(include=['float64', 'int64']).
categorical_columns = df.select_dtypes(include=['object']).columns

# Replace missing values in numerical columns with the median
for col in numerical_columns:
    df[col].fillna(df[col].median(), inplace=True)

# Replace missing values in categorical columns with the most frequ
for col in categorical_columns:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

In [6]:
```
# Display missing values after handling
print("\nMissing values after handling:")
print(df.isnull().sum())
```

```
Missing values after handling:
duration                  0
protocoltype              0
service                   0
flag                      0
srcbytes                  0
dstbytes                  0
land                      0
wrongfragment             0
urgent                    0
hot                       0
numfailedlogins           0
loggedin                  0
numcompromised            0
rootshell                 0
suattempted               0
numroot                   0
numfilecreations          0
numshells                 0
numaccessfiles            0
numoutboundcmds           0
ishostlogin               0
isguestlogin              0
count                     0
srvcount                  0
serrorrate                0
srvserrorrate             0
rerrorrate                0
srvrerrorrate             0
samesrvrate               0
diffsrvrate               0
srvdiffhostrate           0
dsthostcount              0
dsthostsrvcount           0
dsthostsamesrvrate        0
dsthostdiffsrvrate        0
dsthostsamesrcportrate    0
dsthostsrvdiffhostrate    0
dsthostserrorrate         0
dsthostsrvserrorrate      0
dsthostrerrorrate         0
dsthostsrvrerrorrate      0
attack                    0
lastflag                  0
dtype: int64
```

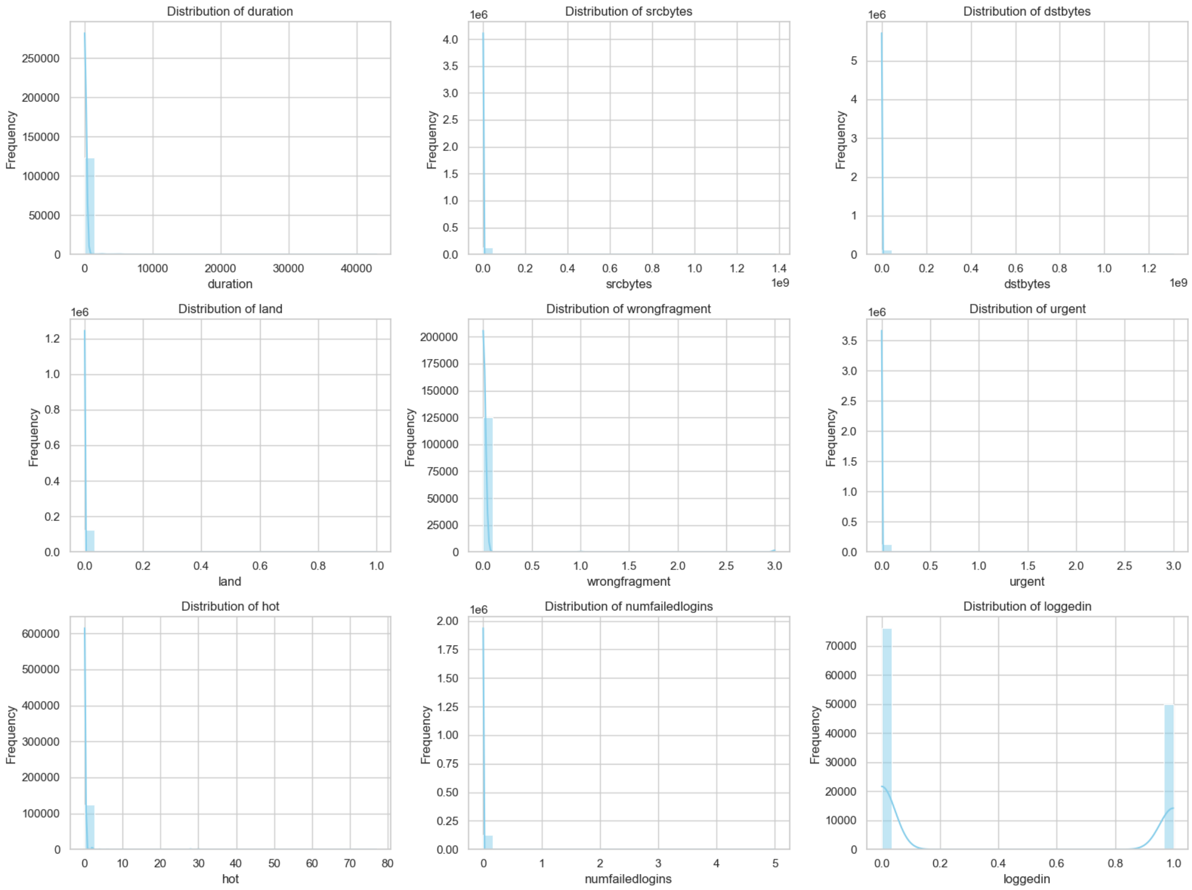
In [7]:
```
# Set plot style
sns.set(style="whitegrid")

# Function to plot the distribution of numeric features
def plot_feature_distributions(data, feature_columns):
    n_cols = 3
    n_rows = (len(feature_columns) + n_cols - 1) // n_cols
    plt.figure(figsize=(16, n_rows * 4))

    for i, feature in enumerate(feature_columns, 1):
        plt.subplot(n_rows, n_cols, i)
        sns.histplot(data[feature], kde=True, bins=30, color="skybl
        plt.title(f"Distribution of {feature}")
        plt.xlabel(feature)
        plt.ylabel("Frequency")

    plt.tight_layout()
    plt.show()

# Select numeric columns for distribution analysis
numeric_columns = df.select_dtypes(include=['int64', 'float64']).co
plot_feature_distributions(df, numeric_columns[:9])  # Plot for the
```



Observations:

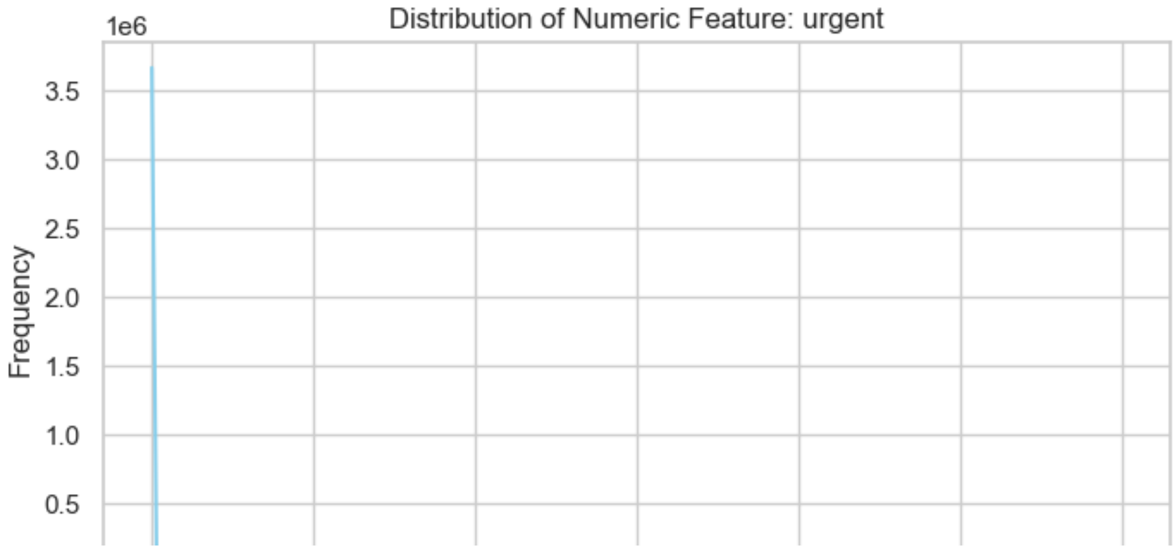
Features like duration, srcbytes, and dstbytes have highly skewed distributions, likely influenced by extreme outliers or infrequent high values. Binary features such as land and urgent show a discrete distribution. Some features, like wrongfragment, have a significant number of zero entries, indicating sparsity.

In [8]:
```
def plot_distributions(data):
    # Separate numeric and categorical columns
    numeric_columns = data.select_dtypes(include=['int64', 'float64
    categorical_columns = data.select_dtypes(include=['object', 'ca

    # Plot distributions for numeric features
    for column in numeric_columns:
        plt.figure(figsize=(8, 4))
        sns.histplot(data[column], kde=True, bins=30, color="skyblu
        plt.title(f"Distribution of Numeric Feature: {column}")
        plt.xlabel(column)
        plt.ylabel("Frequency")
        plt.show()

    # Plot distributions for categorical features
    for column in categorical_columns:
        plt.figure(figsize=(8, 4))
        sns.countplot(data=data, x=column, palette="viridis")
        plt.title(f"Distribution of Categorical Feature: {column}")
        plt.xlabel(column)
        plt.ylabel("Count")
        plt.xticks(rotation=45)
        plt.show()
```

In [9]:
```
# Call the function to visualize all feature distributions
plot_distributions(df)
```



Corelation

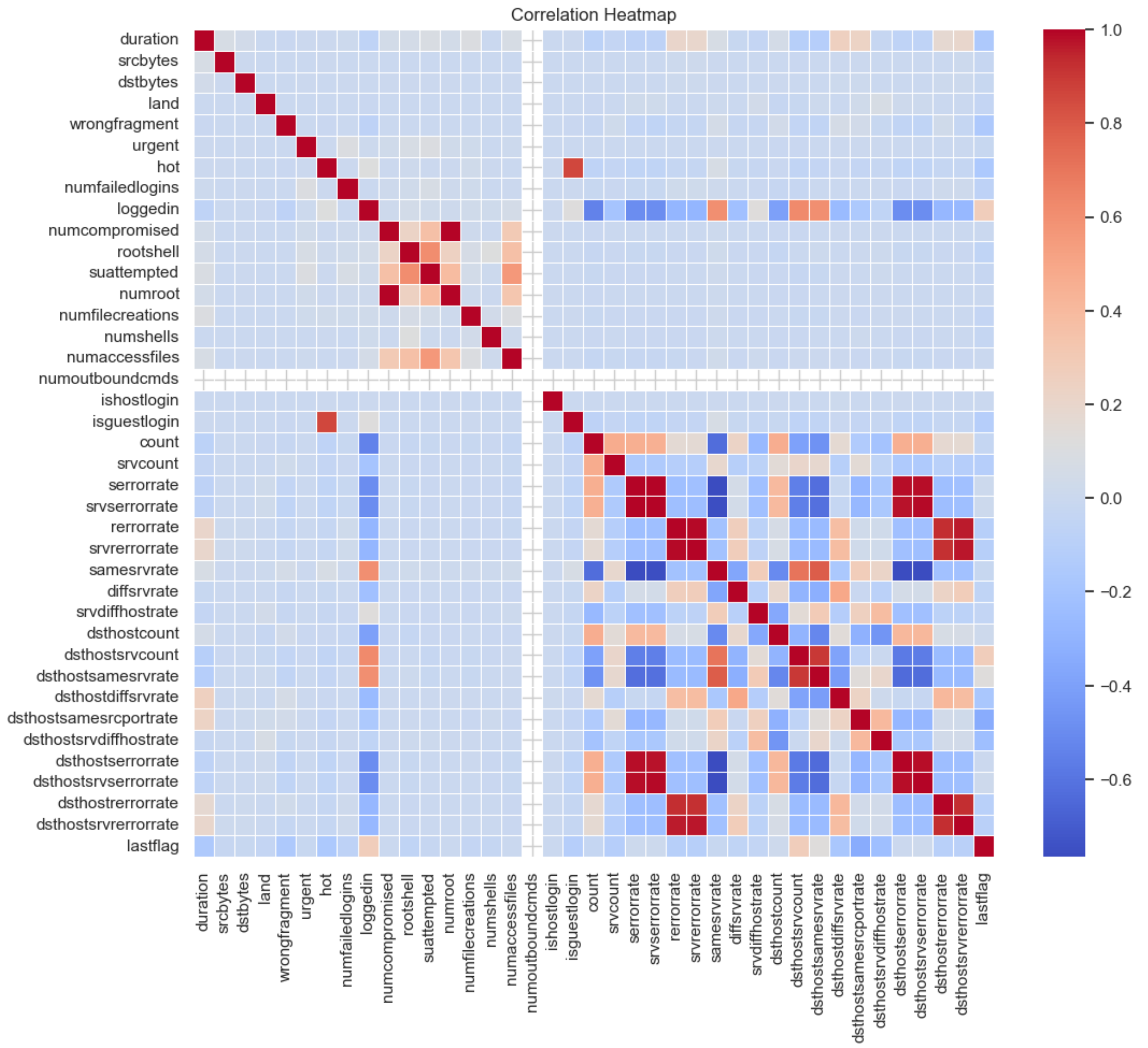
To identify highly correlated features in your dataset and drop the ones that are redundant, we can calculate the correlation matrix and use a threshold to decide which features to drop.

In [10]:
```
def correlation_analysis(data):
    # Compute the correlation matrix
    # Identify numerical columns to scale/normalize
    numerical_columns = data.select_dtypes(include=['float64', 'int
    corr_matrix = data[numerical_columns].corr()

    # Plot the heatmap
    plt.figure(figsize=(12, 10))
    sns.heatmap(corr_matrix, annot=False, cmap="coolwarm", fmt='.2f
    plt.title("Correlation Heatmap")
    plt.show()

    # Return the correlation matrix for further analysis
    return corr_matrix
```

In [11]:
```
# Call the function for correlation analysis
correlation_matrix = correlation_analysis(df)
```



Network Traffic Volume and Anomalies:

In [12]:
```
import pandas as pd
from scipy import stats

# load the dataset
#df = pd.read_csv('network_anomaly_data.csv')

# check unique values in the 'attack' column to ensure it has 'norm
print(df['attack'].value_counts())

# filter the data based on the 'attack' column
normal_connections = df[df['attack'] == 'normal']
neptune_connections = df[df['attack'] != 'normal']

# ensure there are no missing values in src_bytes and dst_bytes
normal_connections = normal_connections.dropna(subset=['srcbytes',
```

```
neptune_connections = neptune_connections.dropna(subset=['srcbytes'

# perform t-tests for src_bytes
t_stat_src, p_value_src = stats.ttest_ind(normal_connections['srcby
print(f"t-test for src_bytes: t-statistic = {t_stat_src}, p-value =

# perform t-tests for dst_bytes
t_stat_dst, p_value_dst = stats.ttest_ind(normal_connections['dstby
print(f"t-test for dst_bytes: t-statistic = {t_stat_dst}, p-value =
```

```
attack
normal             67343
neptune            41214
satan               3633
ipsweep             3599
portsweep           2931
smurf               2646
nmap                1493
back                 956
teardrop             892
warezclient          890
pod                  201
guess_passwd          53
buffer_overflow       30
warezmaster           20
land                  18
imap                  11
rootkit               10
loadmodule             9
ftp_write              8
multihop               7
phf                    4
perl                   3
spy                    2
Name: count, dtype: int64
t-test for src_bytes: t-statistic = -2.101656020563486, p-value =
0.03558539933331456
t-test for dst_bytes: t-statistic = -1.4614241258205836, p-value =
0.14390157812640425
```

Impact of Protocol Type on Anomaly Detection:

To test the hypothesis that certain protocols are more frequently associated with network anomalies, you can use a Chi-square test to determine if the distribution of protocol_type differs significantly between normal and anomalous connections.

Here's the step-by-step approach:

Steps:

Group Data: You will divide the data into normal connections and anomalous connections (using the attack column).

Create a Contingency Table: The table will show the frequency distribution of protocol_type for both normal and anomalous connections.

Perform the Chi-Square Test: You can then apply the Chi-square test of independence to see if there is a significant difference in protocol usage between normal and anomalous connections.

In [13]:

```
# load the dataset
#df = pd.read_csv('network_anomaly_data.csv')

# filter the data based on the 'attack' column
normal_connections = df[df['attack'] == 'normal']
anomalous_connections = df[df['attack'] != 'normal']

# create a contingency table for protocol_type and attack
contingency_table = pd.crosstab(df['protocoltype'], df['attack'])

# print the contingency table
print(contingency_table)

# perform chi-square test
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_ta

# print the results
print(f"Chi-square test statistic = {chi2_stat}")
print(f"P-value = {p_value}")
print(f"Degrees of freedom = {dof}")
print(f"Expected frequencies table: \n{expected}")

# Check if the p-value is less than 0.05 to determine statistical s
if p_value < 0.05:
    print("There is a significant difference in the distribution of
else:
    print("There is no significant difference in the distribution o
```

```
attack          back   buffer_overflow   ftp_write   guess_passwd   imap
ipsweep   \
```

```
protocoltype
icmp             0               0          0             0     0
3117
tcp            956              30          8            53    11
482
udp              0               0          0             0     0
0

attack        land  loadmodule  multihop  neptune  ...  phf  pod
portsweep  \
protocoltype                                       ...
icmp             0           0         0        0  ...    0  201
5
tcp             18           9         7    41214  ...    4    0
2926
udp              0           0         0        0  ...    0    0
0

attack        rootkit  satan  smurf  spy  teardrop  warezclient  w
arezmaster
protocoltype
icmp                0     32   2646    0         0            0
0
tcp                 7   2184      0    2         0          890
20
udp                 3   1417      0    0       892            0
0

[3 rows x 23 columns]
Chi-square test statistic = 110962.04754876824
P-value = 0.0
Degrees of freedom = 44
Expected frequencies table:
[[6.29198003e+01 1.97447072e+00 5.26525525e-01 3.48823161e+00
  7.23972597e-01 2.36870671e+02 1.18468243e+00 5.92341216e-01
  4.60709835e-01 2.71252788e+03 9.82628262e+01 4.43222606e+03
  1.97447072e-01 2.63262763e-01 1.32289538e+01 1.92905789e+02
  6.58156907e-01 2.39108404e+02 1.74148317e+02 1.31631381e-01
  5.87075961e+01 5.85759647e+01 1.31631381e+00]
 [7.79299405e+02 2.44550023e+01 6.52133394e+00 4.32038373e+01
  8.96683416e+00 2.93378510e+03 1.46730014e+01 7.33650068e+00
  5.70616719e+00 3.35962821e+04 1.21704395e+03 5.48957739e+04
  2.44550023e+00 3.26066697e+00 1.63848515e+02 2.38925372e+03
  8.15166742e+00 2.96150077e+03 2.15693120e+03 1.63033348e+00
  7.27128734e+02 7.25498400e+02 1.63033348e+01]
 [1.13780794e+02 3.57052702e+00 9.52140538e-01 6.30793106e+00
  1.30919324e+00 4.28344225e+02 2.14231621e+00 1.07115811e+00
  8.33122971e-01 4.90519002e+03 1.77693228e+02 8.01500003e+03
  3.57052702e-01 4.76070269e-01 2.39225310e+01 3.48840490e+02
  1.19017567e+00 4.32390822e+02 3.14920483e+02 2.38035135e-01
  1.06163670e+02 1.05925635e+02 2.38035135e+00]]
There is a significant difference in the distribution of protocol_
type between normal and anomalous connections.
```

Explanation:

Contingency Table: We create a contingency table (pd.crosstab) that shows the frequency of each protocol_type for normal and anomalous connections. Rows represent the different protocol_type values.

Columns represent attack categories (normal or anomalous).

Chi-Square Test: The chi2_contingency function is used to perform the test. This test checks whether the observed frequency distribution differs significantly from the expected distribution under the null hypothesis (that protocol usage is independent of whether a connection is normal or anomalous).

Results:

chi2_stat: Chi-square statistic.

p_value: P-value to determine significance.

dof: Degrees of freedom.

expected: Expected frequencies under the null hypothesis.

If the p-value is less than 0.05, you can conclude that there is a statistically significant difference in the distribution of protocol_type between normal and anomalous connections. If the p-value is greater than 0.05, you fail to reject the null hypothesis and conclude that there is no significant difference

Role of Service in Network Security:

To test the hypothesis that specific services are targeted by network anomalies more frequently than others, you can use the Chi-square test to compare the frequency of service in normal versus anomaly-flagged connections.

Steps:

Group Data: Divide the data into normal connections and anomaly-flagged connections (using the attack column).

Create a Contingency Table: This table will show the frequency distribution of service for both normal and anomalous connections.

Perform the Chi-Square Test: Apply the Chi-square test of independence to determine if the distribution of service differs significantly between normal and anomalous connections.

In [14]:
```
load the dataset
df = pd.read_csv('network_anomaly_data.csv')

 filter the data based on the 'attack' column
ormal_connections = df[df['attack'] == 'normal']
nomalous_connections = df[df['attack'] != 'normal']

 create a contingency table for service and attack
ontingency_table = pd.crosstab(df['service'], df['attack'])

 print the contingency table
rint(contingency_table)

 perform chi-square test
hi2_stat, p_value, dof, expected = chi2_contingency(contingency_tabl

 print the results
rint(f"Chi-square test statistic = {chi2_stat}")
rint(f"P-value = {p_value}")
rint(f"Degrees of freedom = {dof}")
rint(f"Expected frequencies table: \n{expected}")

 Check if the p-value is less than 0.05 to determine statistical sig
f p_value < 0.05:
   print("There is a significant difference in the distribution of s
lse:
   print("There is no significant difference in the distribution of
```

```
attack     back  buffer_overflow  ftp_write  guess_passwd  imap  i
psweep  \
service
IRC           0                0          0             0     0
```

```
0
X11            0                 0          0             0     0
0
Z39_50         0                 0          0             0     0
0
aol            0                 0          0             0     0
0
auth           0                 0          0             0     0
0
...          ...               ...        ...           ...   ...
...
urp_i          0                 0          0             0     0
0
uucp           0                 0          0             0     0
0
uucp_path      0                 0          0             0     0
0
vmnet          0                 0          0             0     0
0
whois          0                 0          0             0     0
5

attack     land  loadmodule  multihop  neptune  ...  phf  pod  por
tsweep  \
service                                         ...
IRC           0           0         0        0  ...    0    0
0
X11           0           0         0        0  ...    0    0

0
Z39_50        0           0         0      851  ...    0    0
8
aol           0           0         0        0  ...    0    0
0
auth          0           0         0      703  ...    0    0
8
...         ...         ...       ...      ...  ...  ...  ...
...
urp_i         0           0         0        0  ...    0    0
0
uucp          0           0         0      769  ...    0    0
5
uucp_path     0           0         0      676  ...    0    0
10
vmnet         0           0         0      606  ...    0    0
8
whois         0           0         0      670  ...    0    0
14

attack     rootkit  satan  smurf  spy  teardrop  warezclient  ware
zmaster
service
IRC              0      1      0    0         0            0
0
```

```
X11             0     6     0    0         0           0
0
Z39_50          0     2     0    0         0           0
0
aol             0     2     0    0         0           0
0
auth            0     7     0    0         0           0
0
...           ...   ...   ...  ...       ...         ...
...
urp_i           0     3     0    0         0           0
0
uucp            0     5     0    0         0           0
0
uucp_path       0     2     0    0         0           0
0
vmnet           0     2     0    0         0           0
0
whois           0     3     0    0         0           0
0

[70 rows x 23 columns]
Chi-square test statistic = 350657.88534601394
P-value = 0.0
Degrees of freedom = 1518
Expected frequencies table:
[[1.41912950e+00 4.45333524e-02 1.18755606e-02 ... 1.32412501e+00
  1.32115612e+00 2.96889016e-02]

 [5.53991728e-01 1.73846777e-02 4.63591405e-03 ... 5.16904416e-01
  5.15745438e-01 1.15897851e-02]
 [6.54165575e+00 2.05282084e-01 5.47418891e-02 ... 6.10372064e+00
  6.09003517e+00 1.36854723e-01]
 ...
 [5.22877124e+00 1.64082780e-01 4.37554079e-02 ... 4.87872798e+00
  4.86778913e+00 1.09388520e-01]
 [4.68236844e+00 1.46936248e-01 3.91829995e-02 ... 4.36890445e+00
  4.35910870e+00 9.79574988e-02]
 [5.25912696e+00 1.65035365e-01 4.40094306e-02 ... 4.90705151e+00
  4.89604915e+00 1.10023576e-01]]
There is a significant difference in the distribution of services
between normal and anomalous connections.
```

Explanation:

Contingency Table: The pd.crosstab function is used to create a contingency table that shows the frequency of each service for normal and anomalous connections. Rows represent different services.

Columns represent the connection status (attack column values: 'normal' and 'anomalous').

Chi-Square Test: The chi2_contingency function tests the independence between service and attack. It checks whether the frequency distribution of services is the same across normal and anomalous connections or if there is a significant difference.

Results:

chi2_stat: Chi-square statistic.

p_value: P-value to determine significance.

dof: Degrees of freedom.

expected: Expected frequency counts under the null hypothesis.

p-value: If the p-value is less than 0.05, the distribution of service is significantly different between normal and anomalous connections, indicating that certain services may be targeted by anomalies more frequently. If the p-value is greater than 0.05, it suggests there is no significant difference, meaning no service is more likely to be targeted by anomalies.

Connection Status and Anomalies

To assess the impact of connection status (represented by the flag column) on the likelihood of an anomaly (based on the attack column), you can use logistic regression. This model will help you predict the likelihood of an anomaly occurring based on the connection status (flag feature) and other potential features.

Steps:

Binary Encoding for Target Variable: The attack column will be converted into a binary target variable where "normal" is 0 and any anomaly (e.g., "neptune", "satan", etc.) is 1.

Encode the flag Feature: The flag column needs to be encoded into numerical values (e.g., using one-hot encoding or label encoding).

Logistic Regression Model: Build a logistic regression model with the flag feature as the predictor variable and the anomaly status as the target variable.

In [15]:
```
# Encode the 'attack' column as binary: 'normal' = 0, others = 1
df['attack_binary'] = df['attack'].apply(lambda x: 0 if x == 'norma

# Encode the 'flag' column using Label Encoding (or One-Hot Encodin
label_encoder = LabelEncoder()
df['flag_encoded'] = label_encoder.fit_transform(df['flag'])

# Create the feature matrix (X) and target vector (y)
X = df[['flag_encoded']]  # Using only 'flag' feature for now
y = df['attack_binary']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

# Initialize and train the logistic regression model
log_reg_model = LogisticRegression()
log_reg_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = log_reg_model.predict(X_test)

# Evaluate the model
print(f"Accuracy Score: {accuracy_score(y_test, y_pred)}")
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Coefficients of the logistic regression model
print(f"Logistic Regression Coefficients: {log_reg_model.coef_}")
```

```
Accuracy Score: 0.8734917442845047
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.94      0.89     20083
           1       0.92      0.80      0.86     17709

    accuracy                           0.87     37792
   macro avg       0.88      0.87      0.87     37792
weighted avg       0.88      0.87      0.87     37792

Logistic Regression Coefficients: [[-0.7502049]]
```

Explanation:

Encoding the attack Column: We create a binary attack_binary column where 0 indicates normal connections, and 1 indicates anomalies.

Encoding the flag Column: We apply label encoding to convert the categorical values in the flag column into numerical values. Each unique value in flag will be converted to a unique integer. You could also use one-hot encoding if the flag feature has many unique values.

Modeling: A logistic regression model is built with flag_encoded as the predictor variable and attack_binary as the target variable. We use train_test_split to split the data into training and testing sets.

Evaluation: The model is evaluated using accuracy and a classification report, which includes precision, recall, and F1-score.

Coefficients: The coefficients of the logistic regression model indicate the strength and direction of the relationship between the flag feature and the likelihood of an anomaly.

The accuracy score indicates how well the model is performing.

The classification report shows how the model's predictions compare to the actual values, with precision, recall, and F1-score values for both normal and anomalous connections.

The logistic regression coefficient tells you the impact of the flag feature on the probability of anomaly occurrence (higher values indicate a greater likelihood of anomalies).

Interpretation:

If the coefficient is significantly different from 0, it suggests that the flag feature has an impact on predicting network anomalies.

The p-value can also be used to assess whether this coefficient is statistically significant. You can check this by using statsmodels if you want more detailed statistical analysis.

Influence of Urgent Packets:

To evaluate whether the presence of urgent packets (represented by the urgent column) increases the odds of a connection being anomalous, you can use logistic regression. The goal is to predict the likelihood of an anomaly (based on the attack column) based on the presence of urgent packets.

Steps: Binary Encoding for Target Variable: Convert the attack column into a binary target variable, where "normal" is 0 and any anomaly (e.g., "neptune", "satan", etc.) is 1.

Prepare the urgent Feature: The urgent feature is already numerical, so you can use it directly. If it's a binary feature (1 for presence, 0 for absence), it will be easy to include.

Logistic Regression Model: Build a logistic regression model with urgent as the predictor variable and attack_binary as the target variable.

Interpret the Results: Analyze the logistic regression coefficients to determine if the presence of urgent packets is significantly associated with anomalies.

In [16]:
```
# Encode the 'attack' column as binary: 'normal' = 0, others = 1
df['attack_binary'] = df['attack'].apply(lambda x: 0 if x == 'norma

# Select the 'urgent' column as the feature and 'attack_binary' as
X = df[['urgent']]  # Using only 'urgent' feature for now
y = df['attack_binary']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

# Initialize and train the logistic regression model
log_reg_model = LogisticRegression()
log_reg_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = log_reg_model.predict(X_test)

# Evaluate the model
print(f"Accuracy Score: {accuracy_score(y_test, y_pred)}")
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Coefficients of the logistic regression model
print(f"Logistic Regression Coefficients: {log_reg_model.coef_}")
```

```
Accuracy Score: 0.5314087637595258
Classification Report:
              precision    recall  f1-score   support

           0       0.53      1.00      0.69     20083
           1       0.00      0.00      0.00     17709

    accuracy                           0.53     37792
   macro avg       0.27      0.50      0.35     37792
weighted avg       0.28      0.53      0.37     37792

Logistic Regression Coefficients: [[-1.06406261]]
```

Explanation:

Encoding the attack Column: The attack column is encoded as binary: 0 for normal connections and 1 for anomalies (neptune, satan, etc.).

Using the urgent Feature: We use the urgent feature, which indicates the presence of urgent packets, as a predictor for the logistic regression model. This feature should already be binary (1 for urgent packets and 0 for non-urgent packets), making it suitable for this analysis.

Logistic Regression Model: We use logistic regression with urgent as the independent variable and attack_binary as the dependent variable. The logistic regression model will estimate the odds of an anomaly based on the presence of urgent packets.

Model Evaluation: We use accuracy and a classification report to evaluate the model. Additionally, we look at the coefficients of the model to understand the influence of the urgent feature on the likelihood of an anomaly.

Interpretation:

Accuracy Score: The accuracy score shows how well the model is performing, indicating how well the presence of urgent packets can predict anomalies.

Classification Report: This report includes precision, recall, and F1-score for both normal and anomalous connections. It shows the model's ability to correctly identify anomalies and normal connections.

Logistic Regression Coefficients: The coefficient for urgent indicates the relationship between the presence of urgent packets and the likelihood of an anomaly. If the coefficient is positive and significantly different from 0, it suggests that the presence of urgent packets increases the likelihood of an anomaly. A negative coefficient would suggest the opposite (that urgent packets decrease the likelihood of anomalies).

In [17]:
```
# Check for duplicates
print(f"Number of duplicates before removal: {df.duplicated().sum()

# Remove duplicates
df_cleaned = df.drop_duplicates()

# Verify if duplicates are removed
print(f"Number of duplicates after removal: {df_cleaned.duplicated(
```

```
Number of duplicates before removal: 0
Number of duplicates after removal: 0
```

In [18]:
```
categorical_columns = ['protocoltype', 'service', 'flag']

# Dictionary to store mappings
label_encoders = {}
label_mappings = {}

# Apply Label Encoding and store mappings
for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
    label_mappings[col] = {index: label for index, label in enumera

# Print the mappings for each column
for col, mapping in label_mappings.items():
    print(f"Mapping for {col}:")
    for encoded, original in mapping.items():
        print(f"  {encoded} -> {original}")
    print()

# Display the first few rows of the dataset
print("\nEncoded Dataset:")
print(df.head())
```

```
  10 -> discard
  11 -> domain
  12 -> domain_u
  13 -> echo
  14 -> eco_i
  15 -> ecr_i
  16 -> efs
  17 -> exec
  18 -> finger
  19 -> ftp
  20 -> ftp_data
  21 -> gopher
  22 -> harvest
  23 -> hostnames
  24 -> http
  25 -> http_2784
  26 -> http_443
  27 -> http_8001
  28 -> imap4
  29 -> iso_tsap
```

In [19]:
```
# Identify numerical columns to scale/normalize
numerical_columns = df.select_dtypes(include=['float64', 'int64']).

# Standardization: Mean = 0, Std Dev = 1
standard_scaler = StandardScaler()
df_standardized = df.copy()
df_standardized[numerical_columns] = standard_scaler.fit_transform(
```

```
# Normalization: Scale to range [0, 1]
minmax_scaler = MinMaxScaler()
df_normalized = df.copy()
df_normalized[numerical_columns] = minmax_scaler.fit_transform(df[n

# Display the transformed datasets
print("Standardized Dataset (first 5 rows):")
print(df_standardized.head())

print("\nNormalized Dataset (first 5 rows):")
print(df_normalized.head())
```

```
Standardized Dataset (first 5 rows):
   duration  protocoltype   service      flag  srcbytes  dstbytes
land  \
0 -0.110249     -0.124706 -0.686785  0.751111 -0.007679 -0.004919
-0.014089
1 -0.110249      2.219312  0.781428  0.751111 -0.007737 -0.004919
-0.014089
2 -0.110249     -0.124706  1.087305 -0.736235 -0.007762 -0.004919
-0.014089
3 -0.110249     -0.124706 -0.442083  0.751111 -0.007723 -0.002891
-0.014089
4 -0.110249     -0.124706 -0.442083  0.751111 -0.007728 -0.004814
-0.014089

   wrongfragment    urgent       hot  ...  dsthostsamesrcportrate
\
0      -0.089486 -0.007736 -0.095076  ...                0.069972
1      -0.089486 -0.007736 -0.095076  ...                2.367737
2      -0.089486 -0.007736 -0.095076  ...               -0.480197
3      -0.089486 -0.007736 -0.095076  ...               -0.383108
4      -0.089486 -0.007736 -0.095076  ...               -0.480197

   dsthostsrvdiffhostrate  dsthostserrorrate  dsthostsrvserrorrate
\
0               -0.289103          -0.639532             -0.624871
1               -0.289103          -0.639532             -0.624871
2               -0.289103           1.608759              1.618955
3                0.066252          -0.572083             -0.602433
4               -0.289103          -0.639532             -0.624871

   dsthostrerrorrate  dsthostsrvrerrorrate   attack  lastflag  att
ack_binary  \
0          -0.224532             -0.376387   normal  0.216426
-0.933069
1          -0.387635             -0.376387   normal -1.965556
-0.933069
2          -0.387635             -0.376387  neptune -0.219970
1.071732
3          -0.387635             -0.345084   normal  0.652823
-0.933069
4          -0.387635             -0.376387   normal  0.652823
-0.933069
```

```
   flag_encoded
0      0.751111
1      0.751111
2     -0.736235
3      0.751111
4      0.751111

[5 rows x 45 columns]

Normalized Dataset (first 5 rows):
   duration  protocoltype   service  flag      srcbytes      dstby
tes  land  \
0       0.0           0.5  0.289855   0.9  3.558064e-07  0.000000e
+00   0.0
1       0.0           1.0  0.637681   0.9  1.057999e-07  0.000000e
+00   0.0
2       0.0           0.5  0.710145   0.5  0.000000e+00  0.000000e
+00   0.0
3       0.0           0.5  0.347826   0.9  1.681203e-07  6.223962e

-06   0.0
4       0.0           0.5  0.347826   0.9  1.442067e-07  3.206260e
-07   0.0

   wrongfragment  urgent  hot  ...  dsthostsamesrcportrate  \
0            0.0     0.0  0.0  ...                    0.17
1            0.0     0.0  0.0  ...                    0.88
2            0.0     0.0  0.0  ...                    0.00
3            0.0     0.0  0.0  ...                    0.03
4            0.0     0.0  0.0  ...                    0.00

   dsthostsrvdiffhostrate  dsthostserrorrate  dsthostsrvserrorrate
\
0                    0.00               0.00                  0.00
1                    0.00               0.00                  0.00
2                    0.00               1.00                  1.00
3                    0.04               0.03                  0.01
4                    0.00               0.00                  0.00

   dsthostrerrorrate  dsthostsrvrerrorrate   attack  lastflag  att
ack_binary  \
0               0.05                  0.00   normal  0.952381
0.0
1               0.00                  0.00   normal  0.714286
0.0
2               0.00                  0.00  neptune  0.904762
1.0
3               0.00                  0.01   normal  1.000000
0.0
4               0.00                  0.00   normal  1.000000
0.0

   flag encoded
```

```
0         0.9
1         0.9
2         0.5
3         0.9
4         0.9

[5 rows x 45 columns]
```

In [20]:
```
# Select only numeric fields
numeric_df = df.select_dtypes(include=[np.number])

# Calculate the correlation matrix
correlation_matrix = numeric_df.corr()

# Set a threshold for correlation (e.g., 0.9)
threshold = 0.9

# Initialize a list to store correlated column pairs
correlated_pairs = []

# Find highly correlated features
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:  # Check
            colname1 = correlation_matrix.columns[i]
            colname2 = correlation_matrix.columns[j]
            correlated_pairs.append((colname1, colname2))

# Print correlated column pairs
if correlated_pairs:
    print("Highly correlated column pairs (correlation > 0.9):")
    for pair in correlated_pairs:
        print(f"{pair[0]} and {pair[1]}")
else:
    print("No highly correlated column pairs found.")

# Initialize a set to keep track of features to drop
correlated_features = set()

# Keep only the first feature of each correlated pair (drop the sec
for pair in correlated_pairs:
    correlated_features.add(pair[0])  # Add only the first feature

# Drop the selected features from the original dataframe
df = df.drop(columns=correlated_features)

# Output the dropped features
print(f"\nDropped features due to high correlation: {correlated_fea
```

```
Highly correlated column pairs (correlation > 0.9):
numroot and numcompromised
srvserrorrate and serrorrate
srvrerrorrate and rerrorrate
```

```
srvrerrorrate and rerrorrate
dsthostserrorrate and serrorrate
dsthostserrorrate and srvserrorrate
dsthostsrvserrorrate and serrorrate
dsthostsrvserrorrate and srvserrorrate
dsthostsrvserrorrate and dsthostserrorrate
dsthostrerrorrate and rerrorrate
dsthostrerrorrate and srvrerrorrate
dsthostsrvrerrorrate and rerrorrate
dsthostsrvrerrorrate and srvrerrorrate
dsthostsrvrerrorrate and dsthostrerrorrate
flag_encoded and flag

Dropped features due to high correlation: {'dsthostsrvrerrorrate',
'srvserrorrate', 'dsthostsrvserrorrate', 'flag_encoded', 'dsthosts
errorrate', 'srvrerrorrate', 'numroot', 'dsthostrerrorrate'}
```

In [21]:
```python
df.head()
```

Out[21]:

	duration	protocoltype	service	flag	srcbytes	dstbytes	land	wrongfragment	urgent	I
0	0	1	20	9	491	0	0	0	0	
1	0	2	44	9	146	0	0	0	0	
2	0	1	49	5	0	0	0	0	0	
3	0	1	24	9	232	8153	0	0	0	
4	0	1	24	9	199	420	0	0	0	

5 rows × 37 columns

In [22]:
```python
 Creating Interaction Features (combining numerical features)
f['src_dst_bytes_interaction'] = df['srcbytes'] * df['dstbytes']  #
f['num_failed_logins_hot_interaction'] = df['numfailedlogins'] * df[
f['num_compromised_su_interaction'] = df['numcompromised'] * df['sua

 Aggregated Features: Summary statistics over groups of features
f['total_data_transfer'] = df['srcbytes'] + df['dstbytes']  # Total
f['total_access_operations'] = df['numfilecreations'] + df['numshell



 Drop any features that you may not need
f = df.drop(columns=['srcbytes', 'dstbytes', 'attack'])  # Dropping
```

In [23]:
```
df.head()
```

Out[23]:

	duration	protocoltype	service	flag	land	wrongfragment	urgent	hot	numfailedlogins
0	0	1	20	9	0	0	0	0	0
1	0	2	44	9	0	0	0	0	0
2	0	1	49	5	0	0	0	0	0
3	0	1	24	9	0	0	0	0	0
4	0	1	24	9	0	0	0	0	0

5 rows × 39 columns

In [ ]: