In [50]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import chi2_contingency
from scipy.stats import ttest_ind
import statsmodels.api as sm
from sklearn.model_selection import train_test_splitp
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, roc_
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
import warnings
```

In [51]:
```
# Suppress warnings
warnings.filterwarnings("ignore")
```

In [52]:
```
# 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  dsthostdiffsrvrat
e  \
0              0       0    0  ...                0.17                0.0
3
1              0       0    0  ...                0.00                0.6
0
2              0       0    0  ...                0.10                0.0
5
3              0       0    0  ...                1.00                0.0
0
4              0       0    0  ...                1.00                0.0
0

   dsthostsamesrcportrate  dsthostsrvdiffhostrate  dsthostserrorrate  \
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 [53]:
```
# 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 [54]:
```
# Separate numerical and categorical columns
numerical_columns = df.select_dtypes(include=['float64', 'int64']).column
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 frequent va
for col in categorical_columns:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

In [55]:
```
# 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 [56]:
```
# 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().sum(
```

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

In [57]:
```
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 enumerate(le.

# 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())
```

```
Encoded Dataset:
   duration  protocoltype  service  flag  srcbytes  dstbytes  land  \
0         0             1       20     9       491         0     0
1         0             2       44     9       146         0     0
2         0             1       49     5         0         0     0
3         0             1       24     9       232      8153     0
4         0             1       24     9       199       420     0

   wrongfragment  urgent  hot  ...  dsthostsamesrvrate  dsthostdiffsrvrat
e  \
0              0       0    0  ...                0.17                0.0
3
1              0       0    0  ...                0.00                0.6
0
2              0       0    0  ...                0.10                0.0
5
3              0       0    0  ...                1.00                0.0
0
4              0       0    0  ...                1.00                0.0
0

   dsthostsamesrcportrate  dsthostsrvdiffhostrate  dsthostserrorrate  \
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]
```

In [58]:
```
# Identify numerical columns to scale/normalize
numerical_columns = df.select_dtypes(include=['float64', 'int64']).column

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

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

# 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      la
nd  \
0 -0.110249     -0.124706 -0.686785  0.751111 -0.007679 -0.004919 -0.0140
89
1 -0.110249      2.219312  0.781428  0.751111 -0.007737 -0.004919 -0.0140
89
2 -0.110249     -0.124706  1.087305 -0.736235 -0.007762 -0.004919 -0.0140
89
3 -0.110249     -0.124706 -0.442083  0.751111 -0.007723 -0.002891 -0.0140
89
4 -0.110249     -0.124706 -0.442083  0.751111 -0.007728 -0.004814 -0.0140
89

   wrongfragment    urgent       hot  ...  dsthostsamesrvrate  \
0      -0.089486 -0.007736 -0.095076  ...           -0.782367
1      -0.089486 -0.007736 -0.095076  ...           -1.161030
2      -0.089486 -0.007736 -0.095076  ...           -0.938287
3      -0.089486 -0.007736 -0.095076  ...            1.066401
4      -0.089486 -0.007736 -0.095076  ...            1.066401

   dsthostdiffsrvrate  dsthostsamesrcportrate  dsthostsrvdiffhostrate  \
0           -0.280282                0.069972               -0.289103
1            2.736852                2.367737               -0.289103
2           -0.174417               -0.480197               -0.289103
3           -0.439078               -0.383108                0.066252
4           -0.439078               -0.480197               -0.289103

   dsthostserrorrate  dsthostsrvserrorrate  dsthostrerrorrate  \
0          -0.639532             -0.624871          -0.224532
1          -0.639532             -0.624871          -0.387635
2           1.608759              1.618955          -0.387635
3          -0.572083             -0.602433          -0.387635
4          -0.639532             -0.624871          -0.387635

   dsthostsrvrerrorrate   attack  lastflag
```

```
0             -0.376387   normal  0.216426
1             -0.376387   normal -1.965556
2             -0.376387  neptune -0.219970
3             -0.345084   normal  0.652823
4             -0.376387   normal  0.652823

[5 rows x 43 columns]

Normalized Dataset (first 5 rows):
   duration  protocoltype   service  flag      srcbytes      dstbytes  la
nd  \
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  ...  dsthostsamesrvrate  dsthostdiffsrvrat
e  \
0            0.0     0.0  0.0  ...                0.17                0.0
3
1            0.0     0.0  0.0  ...                0.00                0.6
0
2            0.0     0.0  0.0  ...                0.10                0.0
5
3            0.0     0.0  0.0  ...                1.00                0.0
0
4            0.0     0.0  0.0  ...                1.00                0.0
0

   dsthostsamesrcportrate  dsthostsrvdiffhostrate  dsthostserrorrate  \
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  0.952381
1  0.714286
2  0.904762
3  1.000000
4  1.000000
```

```
[5 rows x 43 columns]
```

In [59]:
```python
# 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 if co
            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 second on
for pair in correlated_pairs:
    correlated_features.add(pair[0])  # Add only the first feature to the

# 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_features}
```

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

Dropped features due to high correlation: {'srvrerrorrate', 'numroot', 'd
sthostsrvserrorrate', 'dsthostserrorrate', 'dsthostrerrorrate', 'srvserro
rrate', 'dsthostsrvrerrorrate'}
```

In [60]:
```
df.head()
```

Out[60]:

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

5 rows × 36 columns

1. Network Traffic Volume and Anomalies:

Hypothesis: Network connections with unusually high or low traffic volume (bytes transferred) are more likely to be anomalous.

Tests: Use t-tests or ANOVA to compare the means of Src_bytes and Dst_bytes in normal versus anomalous connections.

In [61]:
```
# Define Normal and Anomalous categories
normal_connections = df[df["attack"] == "normal"]
anomalous_connections = df[df["attack"] != "normal"]

# Perform t-tests for Src_bytes and Dst_bytes
ttest_src = ttest_ind(normal_connections["srcbytes"], anomalous_connectio
ttest_dst = ttest_ind(normal_connections["dstbytes"], anomalous_connectio

# Print the hypotheses and results
print("Hypothesis for Src_bytes:")
print("Null Hypothesis (H0): The mean Src_bytes is the same for Normal an
print("Alternative Hypothesis (Ha): The mean Src_bytes is different for N
print(f"T-statistic: {ttest_src.statistic:.2f}, p-value: {ttest_src.pvalu
if ttest_src.pvalue < 0.05:
    print("Conclusion: Reject the null hypothesis. Significant difference
else:
    print("Conclusion: Fail to reject the null hypothesis. No significant

print("Hypothesis for Dst_bytes:")
print("Null Hypothesis (H0): The mean Dst_bytes is the same for Normal an
print("Alternative Hypothesis (Ha): The mean Dst_bytes is different for N
print(f"T-statistic: {ttest_dst.statistic:.2f}, p-value: {ttest_dst.pvalu
if ttest_dst.pvalue < 0.05:
    print("Conclusion: Reject the null hypothesis. Significant difference
else:
    print("Conclusion: Fail to reject the null hypothesis. No significant
```

```
Hypothesis for Src_bytes:
Null Hypothesis (H0): The mean Src_bytes is the same for Normal and Anoma
lous connections.
Alternative Hypothesis (Ha): The mean Src_bytes is different for Normal a
nd Anomalous connections.
T-statistic: -1.96, p-value: 0.0498
Conclusion: Reject the null hypothesis. Significant difference in Src_byt
es means between Normal and Anomalous connections.

Hypothesis for Dst_bytes:
Null Hypothesis (H0): The mean Dst_bytes is the same for Normal and Anoma
lous connections.
Alternative Hypothesis (Ha): The mean Dst_bytes is different for Normal a
nd Anomalous connections.
T-statistic: -1.36, p-value: 0.1727
Conclusion: Fail to reject the null hypothesis. No significant difference
in Dst_bytes means.
```

2. Impact of Protocol Type on Anomaly Detection:

Hypothesis: Certain protocols are more frequently associated with network anomalies.

Tests: Chi-square test to determine if the distribution of Protocol_type differs significantly in normal and anomalous connections.

In [62]:
```
# Create a contingency table for Protocol_type and Attacks
df["connectiontype"] = np.where(df["attack"] == "normal", "normal", "anom
contingency_table = pd.crosstab(df["protocoltype"], df["connectiontype"])

# Perform the chi-square test
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

# Print hypotheses and results
print("Hypothesis:")
print("Null Hypothesis (H0): The distribution of Protocol_type is indepen
print("Alternative Hypothesis (Ha): The distribution of Protocol_type is

print("Chi-square Test Results:")
print(f"Chi-square Statistic: {chi2_stat:.2f}")
print(f"Degrees of Freedom: {dof}")
print(f"P-value: {p_value:.4f}\n")

if p_value < 0.05:
    print("Conclusion: Reject the null hypothesis. The distribution of Pr
else:
    print("Conclusion: Fail to reject the null hypothesis. No significant

# Optional: Print the contingency table for reference
print("\nContingency Table:")
print(contingency_table)

print("\nExpected Frequencies Table:")
print(pd.DataFrame(expected, index=contingency_table.index, columns=conti
```

```
Hypothesis:
Null Hypothesis (H0): The distribution of Protocol_type is independent of
Normal and Anomalous connections.
Alternative Hypothesis (Ha): The distribution of Protocol_type is associa
ted with Normal and Anomalous connections.

Chi-square Test Results:
Chi-square Statistic: 10029.25
Degrees of Freedom: 2
P-value: 0.0000

Conclusion: Reject the null hypothesis. The distribution of Protocol_type
differs significantly between Normal and Anomalous connections.

Contingency Table:
connectiontype  anomalous  normal
protocoltype
0                    6982    1309
1                   49089   53600
2                    2559   12434

Expected Frequencies Table:
connectiontype     anomalous        normal
protocoltype
0                3858.773944   4432.226056
1               47793.226088  54895.773912
2                6977.999968   8015.000032
```

3. Role of Service in Network Security:

Hypothesis: Specific services are targets of network anomalies more often than others.

Tests: Chi-square test to compare the frequency of services in normal versus anomaly-flagged connections.

In [63]:
```
# Categorize connection type as 'Normal' or 'Anomalous'
df["Connection_Type"] = np.where(df["attack"] == "normal", "Normal", "Ano

# Create a contingency table for Service and Connection_Type
contingency_table = pd.crosstab(df["service"], df["Connection_Type"])

# Perform the chi-square test
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

# Print hypotheses and results
print("Hypothesis:")
print("Null Hypothesis (H0): The distribution of services is independent
print("Alternative Hypothesis (Ha): The distribution of services is assoc

print("Chi-square Test Results:")
print(f"Chi-square Statistic: {chi2_stat:.2f}")
print(f"Degrees of Freedom: {dof}")
print(f"P-value: {p_value:.4f}\n")

if p_value < 0.05:
    print("Conclusion: Reject the null hypothesis. The distribution of se
else:
    print("Conclusion: Fail to reject the null hypothesis. No significant

# Optional: Print the contingency table and expected frequencies for refe
print("\nContingency Table:")
print(contingency_table)

print("\nExpected Frequencies Table:")
print(pd.DataFrame(expected, index=contingency_table.index, columns=conti
```

```
Hypothesis:
Null Hypothesis (H0): The distribution of services is independent of Norm
al and Anomalous connections.
Alternative Hypothesis (Ha): The distribution of services is associated w
ith Normal and Anomalous connections.

Chi-square Test Results:
Chi-square Statistic: 93240.03
Degrees of Freedom: 69
P-value: 0.0000

Conclusion: Reject the null hypothesis. The distribution of services diff
ers significantly between Normal and Anomalous connections.

Contingency Table:
Connection_Type  Anomalous  Normal
service
0                        1     186
1                        6      67
2                      862       0
3                        2       0
4                      719     236
...                    ...     ...
65                       3     599
66                     780       0
67                     689       0
68                     617       0
69                     693       0

[70 rows x 2 columns]

Expected Frequencies Table:
Connection_Type   Anomalous      Normal
service
0                 87.033015   99.966985
1                 33.975455   39.024545
2                401.189620  460.810380
3                  0.930834    1.069166
4                444.473419  510.526581
...                     ...         ...
65               280.181150  321.818850
66               363.025410  416.974590
67               320.672446  368.327554
68               287.162408  329.837592
69               322.534114  370.465886

[70 rows x 2 columns]
```

Feature Engineering Steps

Interaction Features: Combine numerical features to create interaction terms.

Aggregated Features: Create summary statistics like the mean, sum, or count of certain groups of features.

Polynomial Features: Introduce non-linear relationships between features by applying polynomial transformation.

In [64]:
```
# Creating Interaction Features (combining numerical features)
df['src_dst_bytes_interaction'] = df['srcbytes'] * df['dstbytes']  # Inte
df['num_failed_logins_hot_interaction'] = df['numfailedlogins'] * df['hot
df['num_compromised_su_interaction'] = df['numcompromised'] * df['suattem

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



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

In [65]:
```
df.head()
```

Out[65]:

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

5 rows × 41 columns

4. Connection Status and Anomalies:

Hypothesis: Error flags in the Flag feature are significantly associated with anomalies.

Tests: Use logistic regression to assess the impact of connection status on the likelihood of an anomaly.

In [66]:
```
# Encode the 'attack' column as binary: 'normal' = 0, others = 1
df['attack_binary'] = df['attack'].apply(lambda x: 0 if x == 'normal' els

# Encode the 'flag' column using Label Encoding (or One-Hot Encoding if n
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=0.3,

# 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]]
```

Interpretation:

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

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).

5. Influence of Urgent Packets:

Hypothesis: Connections that include urgent packets are more likely to be anomalous. Tests: Logistic regression to evaluate whether the presence of Urgent packets increases the odds of an anomaly.

In [67]:
```
# Encode the 'attack' column as binary: 'normal' = 0, others = 1
df['attack_binary'] = df['attack'].apply(lambda x: 0 if x == 'normal' els

# Select the 'urgent' column as the feature and 'attack_binary' as the ta
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=0.3,

# 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]]
```

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).

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.

In [ ]: