In [2]:
```python
import sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```python
# Replace with your dataset path
df = pd.read_csv(r"C:\Users\AmlanC\OneDrive\Desktop\python\amlpy\CloudWatch_Traffic
```

In [9]:
```python
# df.info()
df.dtypes
# df.describe()
```

Out[9]:
```
bytes_in                 int64
bytes_out                int64
creation_time           object
end_time                object
src_ip                  object
src_ip_country_code     object
protocol                object
response.code            int64
dst_port                 int64
dst_ip                  object
rule_names              object
observation_name        object
source.meta             object
source.name             object
time                    object
detection_types         object
dtype: object
```

In [12]:
```python
print("Dataset shape:", df.shape)
print("Data types:\n", df.dtypes)
print("Missing values:\n", df.isnull().sum())
print("Unique values:\n", df.nunique())
```

```
Dataset shape: (282, 16)
Data types:
 bytes_in                              int64
bytes_out                              int64
creation_time          datetime64[ns, UTC]
end_time               datetime64[ns, UTC]
src_ip                                object
src_ip_country_code                   object
protocol                              object
response.code                          int64
dst_port                               int64
dst_ip                                object
rule_names                            object
observation_name                      object
source.meta                           object
source.name                           object
time                   datetime64[ns, UTC]
detection_types                       object
dtype: object
Missing values:
 bytes_in              0
bytes_out              0
creation_time          0
end_time               0
src_ip                 0
src_ip_country_code    0
protocol               0
response.code          0
dst_port               0
dst_ip                 0
rule_names             0
observation_name       0
source.meta            0
source.name            0
time                   0
detection_types        0
dtype: int64
Unique values:
 bytes_in              260
bytes_out              239
creation_time           30
end_time                30
src_ip                  28
src_ip_country_code      7
protocol                 1
response.code            1
dst_port                 1
dst_ip                   1
rule_names               1
observation_name         1
source.meta              1
source.name              1
time                    30
detection_types          1
dtype: int64
```

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

Out[21]:

	bytes_in	bytes_out	creation_time	end_time	src_ip	src_ip_country_code
0	5602	12990	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	147.161.161.82	AE
1	30912	18186	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	165.225.33.6	US
2	28506	13468	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	165.225.212.255	CA
3	30546	14278	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	136.226.64.114	US
4	6526	13892	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	165.225.240.79	NL

In [26]:
```
df['src_ip'].duplicated().values.any()
```

Out[26]:
```
np.int64(254)
```

In [27]:
```
df['src_ip'].value_counts()
```

Out[27]:
```
src_ip
165.225.209.4      29
165.225.26.101     28
155.91.45.242      28
136.226.67.101     28
147.161.131.1      21
165.225.240.79     18
136.226.77.103     17
147.161.161.82     16
165.225.212.255    15
94.188.248.74      14
136.226.64.114     13
165.225.33.6       12
165.225.213.7      11
136.226.80.97      11
165.225.8.79        6
192.241.230.19      2
65.49.1.69          2
198.235.24.81       1
65.49.1.72          1
65.49.1.94          1
65.49.1.104         1
65.49.1.97          1
65.49.1.99          1
65.49.1.76          1
65.49.1.96          1
65.49.1.95          1
65.49.1.74          1
192.241.205.18      1
Name: count, dtype: int64
```

Feature Engineering

In [11]:
```
# Convert time columns to datetime
df['creation_time'] = pd.to_datetime(df['creation_time'])
df['end_time'] = pd.to_datetime(df['end_time'])
df['time'] = pd.to_datetime(df['time'])
```

In [13]:
```
# Calculate duration in seconds
df['duration'] = (df['end_time'] - df['creation_time']).dt.total_seconds()
```

In [18]:
```
df.rename(columns={'duration': 'session_duration'}, inplace=True)
```

In [20]:
```
# Average packet size
df['avg_packet_size'] = (df['bytes_in'] + df['bytes_out']) / df['session_duration']
```

In [98]:
```
df
```

Out[98]:

creation_time	bytes_in	bytes_out	end_time	src_ip	src_ip_country_code
2024-04-25 23:00:00+00:00	5602	12990	2024-04-25 23:10:00+00:00	147.161.161.82	AE
2024-04-25 23:00:00+00:00	30912	18186	2024-04-25 23:10:00+00:00	165.225.33.6	US
2024-04-25 23:00:00+00:00	28506	13468	2024-04-25 23:10:00+00:00	165.225.212.255	CA
2024-04-25 23:00:00+00:00	30546	14278	2024-04-25 23:10:00+00:00	136.226.64.114	US
2024-04-25 23:00:00+00:00	6526	13892	2024-04-25 23:10:00+00:00	165.225.240.79	NL
...	...	...	...	...	...
2024-04-26 09:50:00+00:00	41336	13180	2024-04-26 10:00:00+00:00	136.226.77.103	CA
2024-04-26 09:50:00+00:00	3638	3190	2024-04-26 10:00:00+00:00	165.225.26.101	DE
2024-04-26 09:50:00+00:00	25207794	1561220	2024-04-26 10:00:00+00:00	155.91.45.242	US
2024-04-26 09:50:00+00:00	5736	12114	2024-04-26 10:00:00+00:00	165.225.209.4	CA
2024-04-26 09:50:00+00:00	9032	5862	2024-04-26 10:00:00+00:00	147.161.131.1	AT

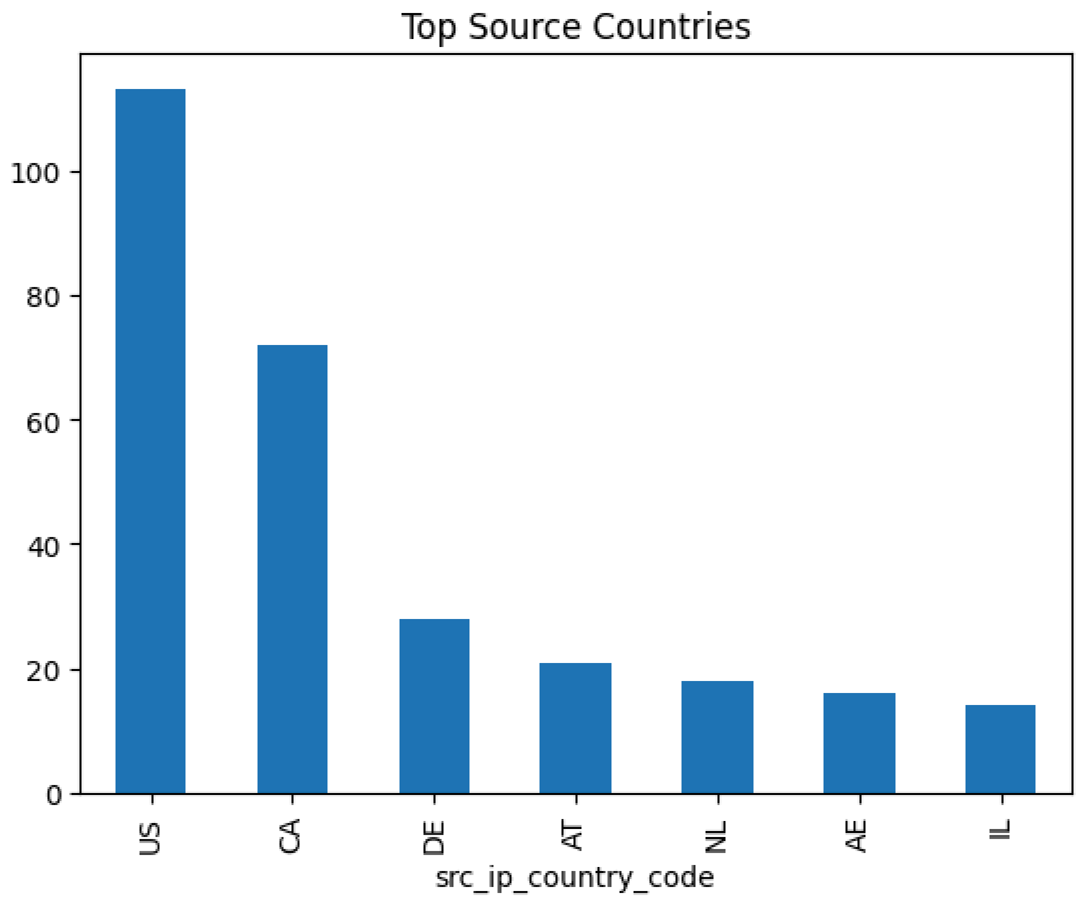
282 rows × 24 columns

5. EXPLORATORY DATA ANALYSIS (EDA)

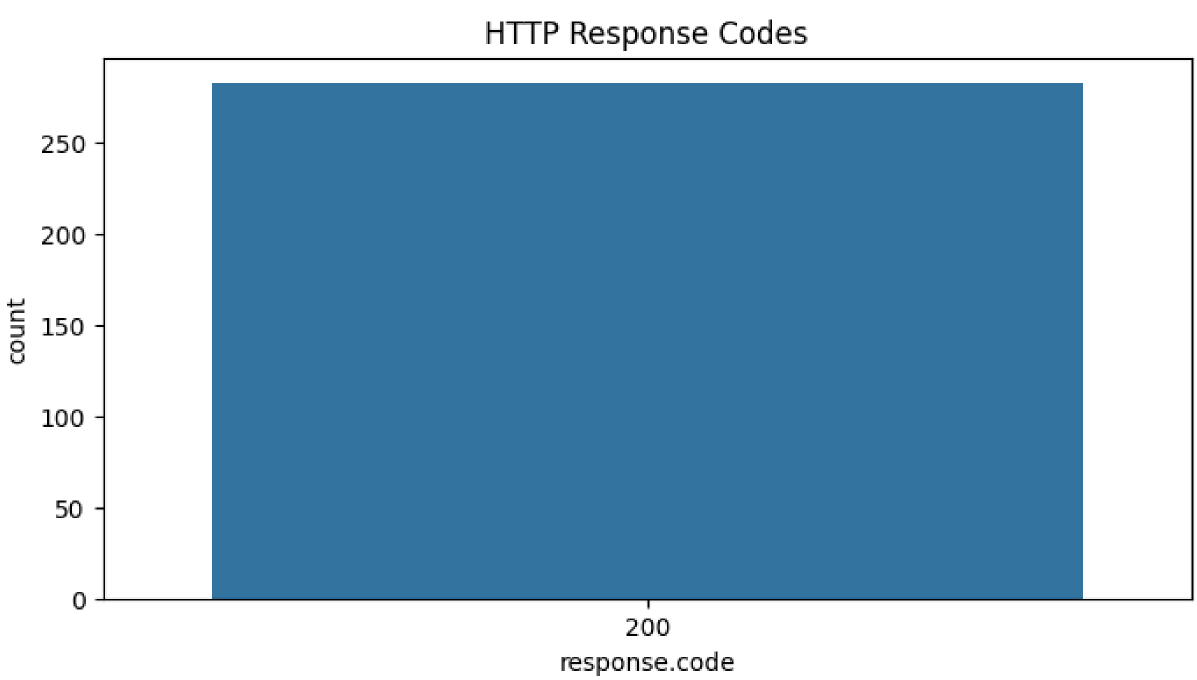
In [34]:

```
# Top source countries
df['src_ip_country_code'].value_counts().head(10).plot(kind='bar', title='Top Sourc
```

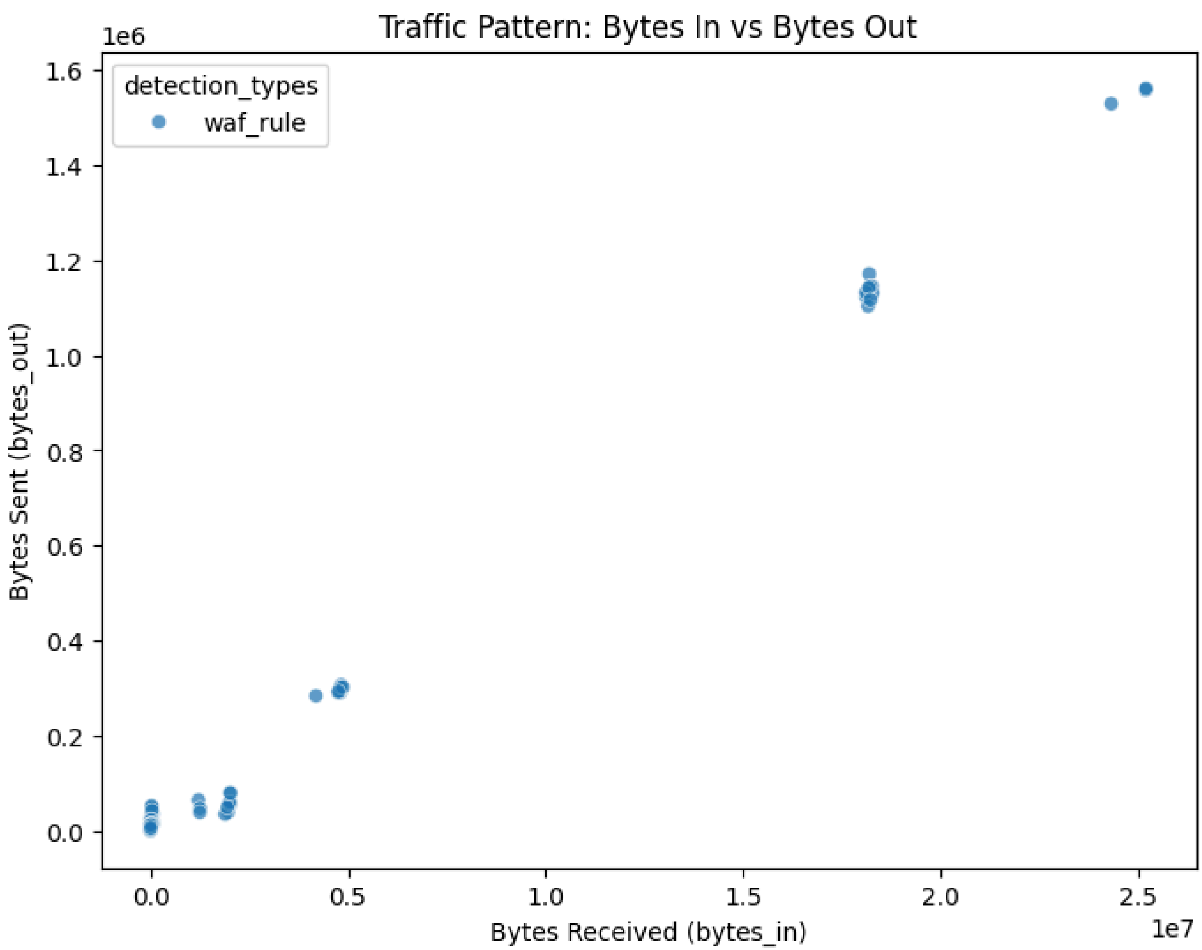
Out[34]: <Axes: title={'center': 'Top Source Countries'}, xlabel='src_ip_country_code'>

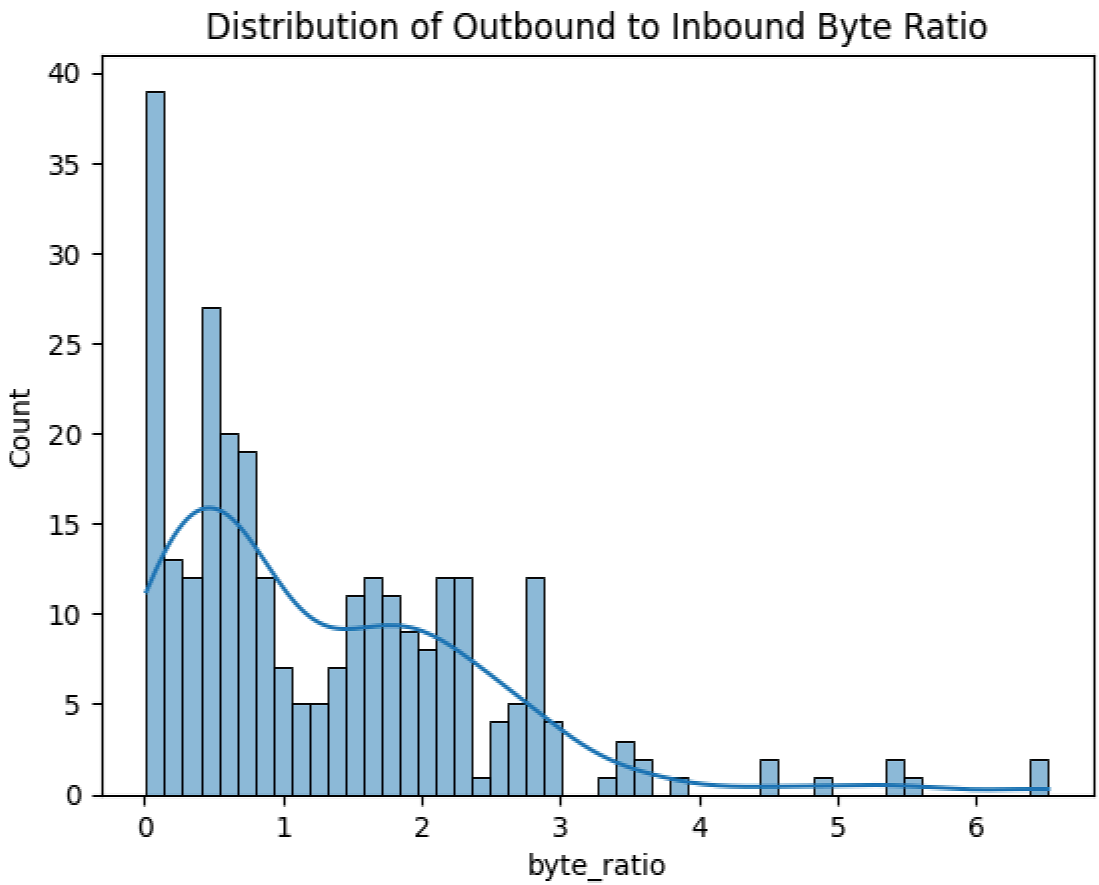


In [35]:
```
# Distribution of response codes
plt.figure(figsize=(8,4))
sns.countplot(data=df, x='response.code')
plt.title("HTTP Response Codes")
plt.show()
```

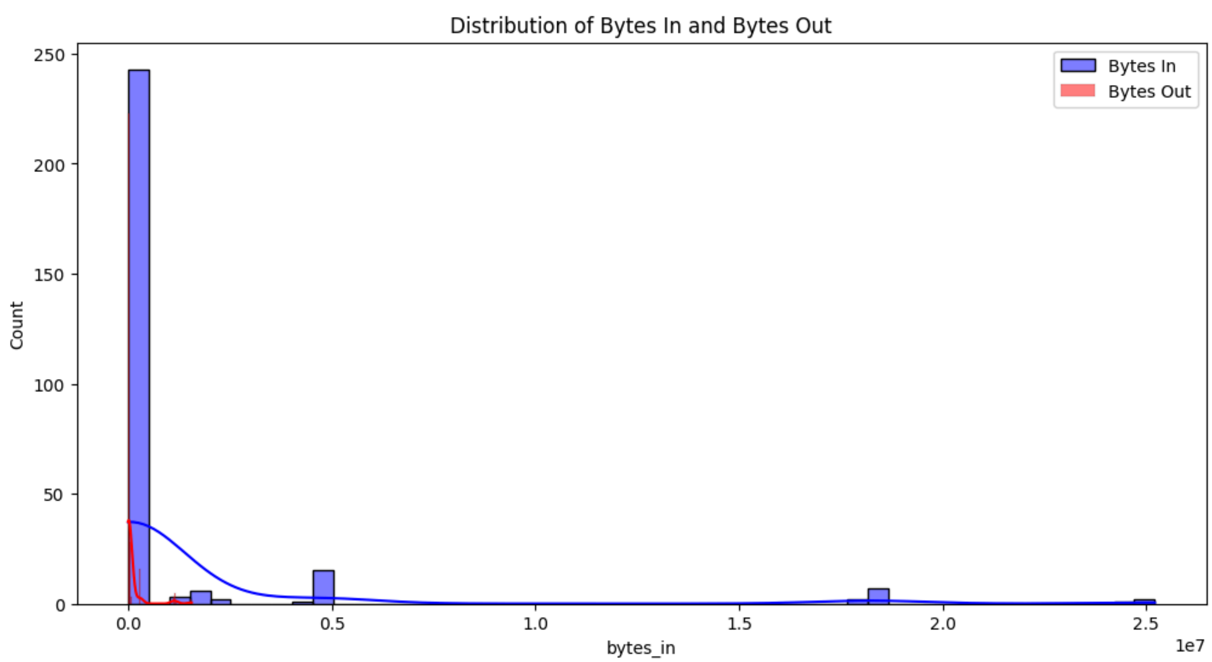


In [36]:
```python
# Scatter plot of bytes_in vs bytes_out
plt.figure(figsize=(8,6))
sns.scatterplot(data=df, x='bytes_in', y='bytes_out', hue='detection_types', alpha=
plt.title("Traffic Pattern: Bytes In vs Bytes Out")
plt.xlabel("Bytes Received (bytes_in)")
plt.ylabel("Bytes Sent (bytes_out)")
plt.show()

# Ratio: bytes_out / bytes_in (to detect potential exfiltration)
df['byte_ratio'] = df['bytes_out'] / (df['bytes_in'] + 1)
sns.histplot(df['byte_ratio'], bins=50, kde=True)
plt.title("Distribution of Outbound to Inbound Byte Ratio")
plt.show()
```



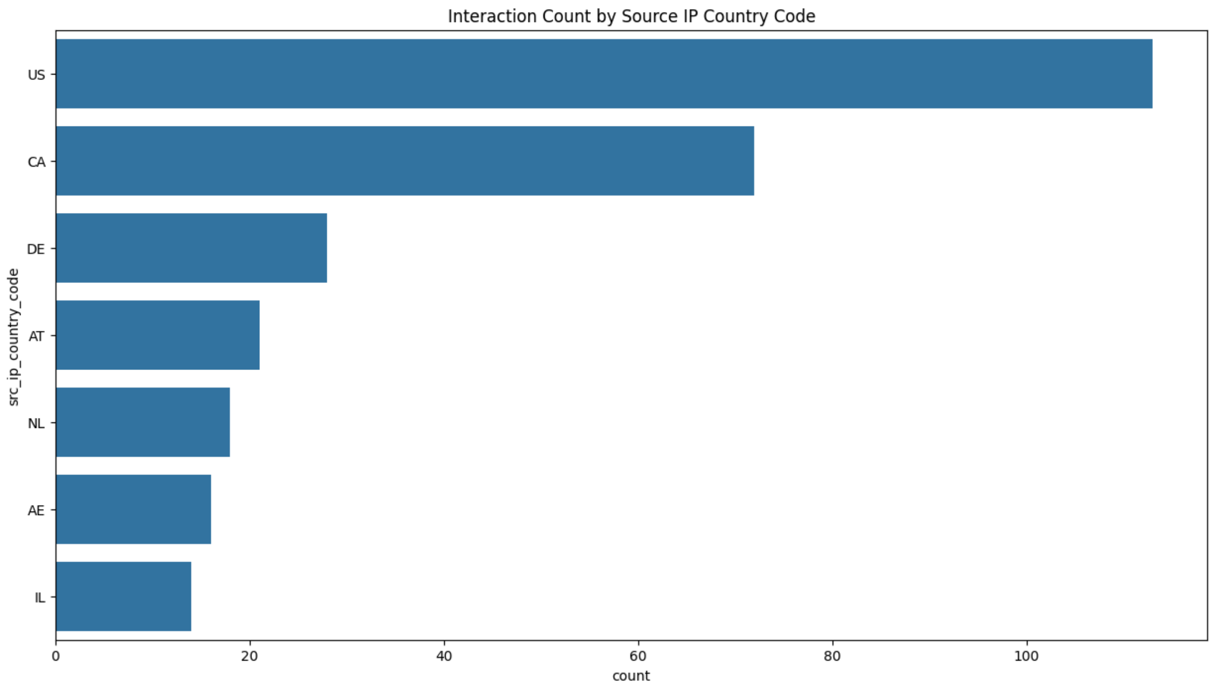


In [29]:
```
# Distribution of bytes in and bytes out
plt.figure(figsize=(12, 6))
sns.histplot(df['bytes_in'], bins=50, color='blue', kde=True, label='Bytes In')
sns.histplot(df['bytes_out'], bins=50, color='red', kde=True, label='Bytes Out')
plt.legend()
plt.title('Distribution of Bytes In and Bytes Out')
plt.show()
```



In [32]:

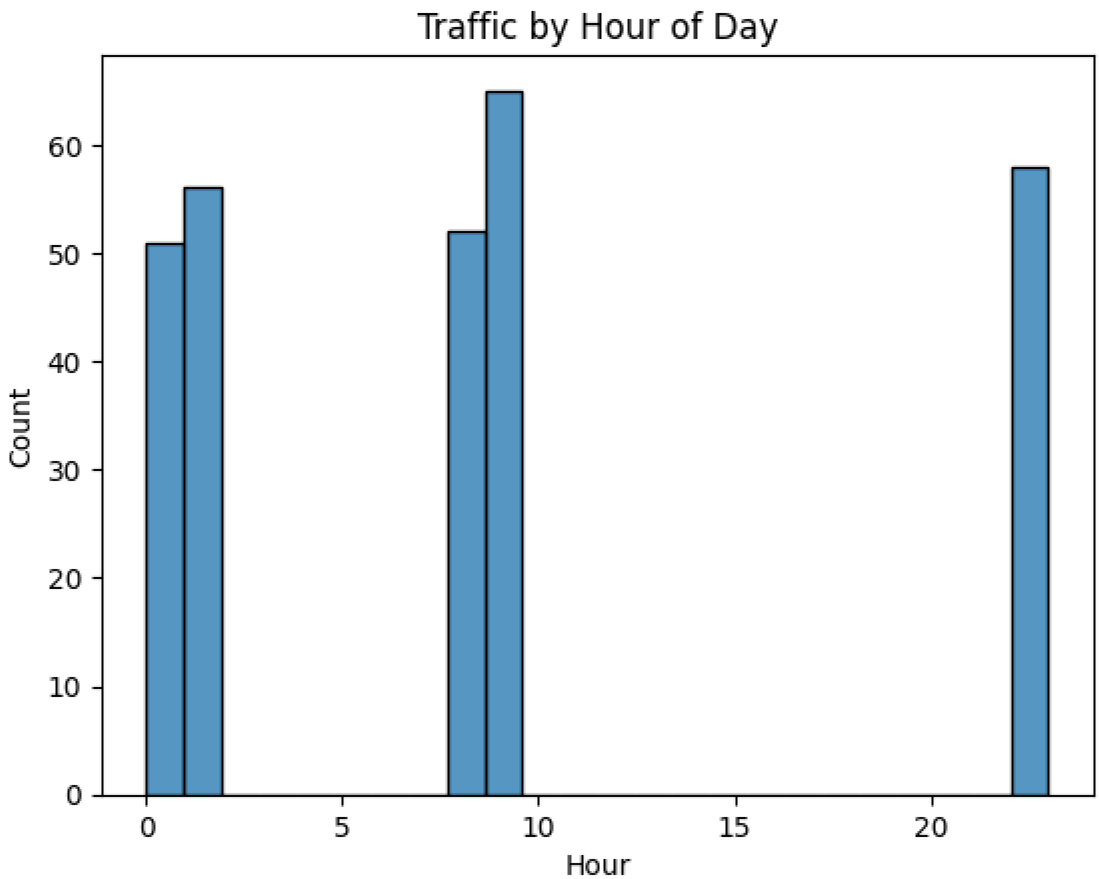
```python
#Country-based Interaction Analysis
plt.figure(figsize=(15, 8))
sns.countplot(y='src_ip_country_code', data=df, order=df['src_ip_country_code'].val
plt.title('Interaction Count by Source IP Country Code')
plt.show()
```

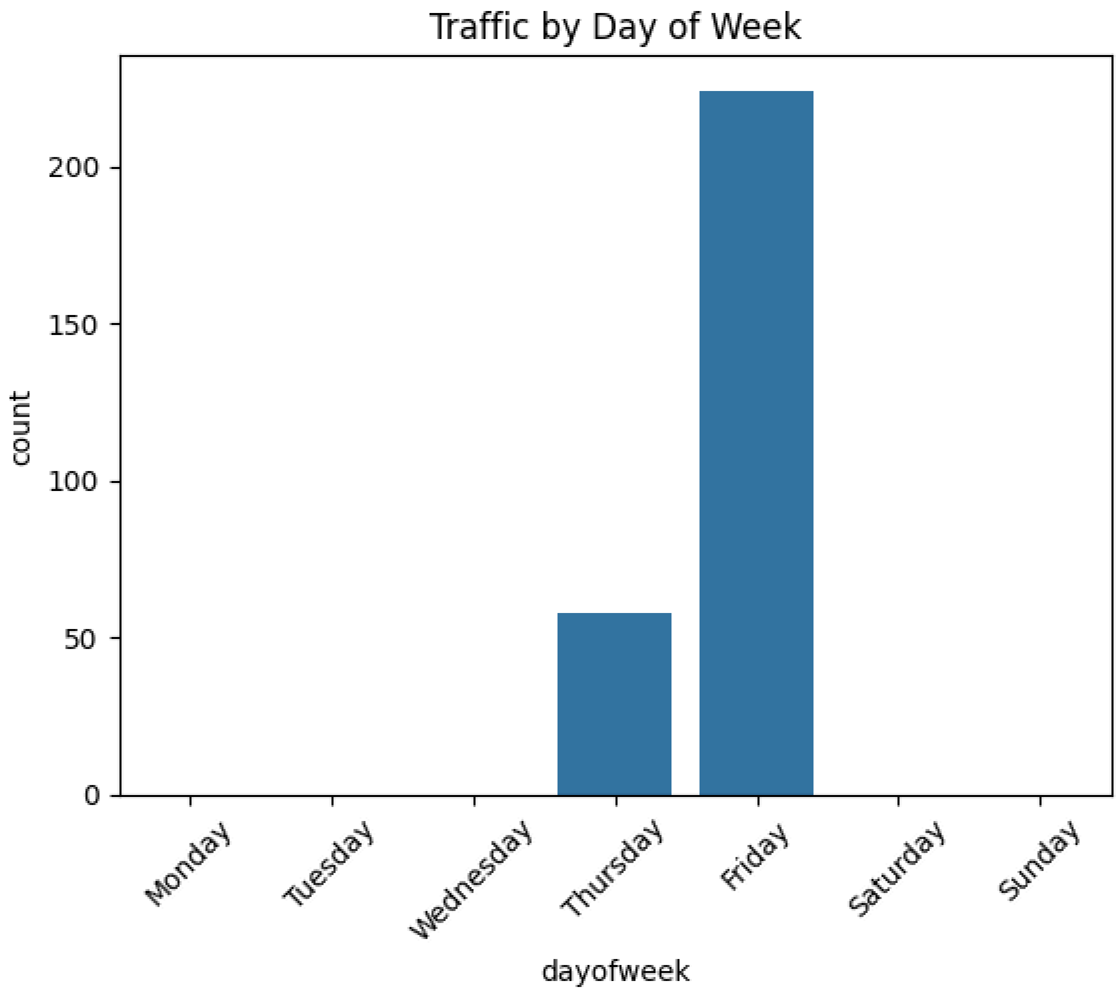


🕒 4. Temporal Pattern Analysis

In [41]:

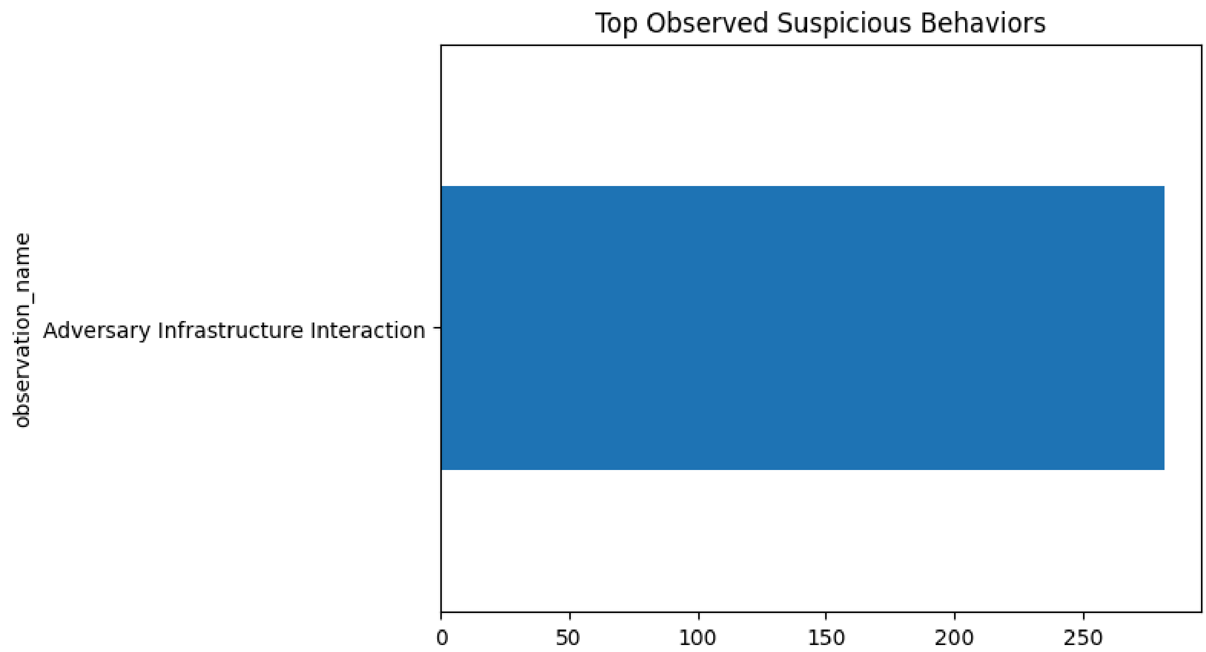
```python
df['hour'] = df['time'].dt.hour
df['dayofweek'] = df['time'].dt.day_name()

# Hourly activity
sns.histplot(df['hour'], bins=24, kde=False)
plt.title("Traffic by Hour of Day")
plt.xlabel("Hour")
plt.ylabel("Count")
plt.show()

# Weekly pattern
sns.countplot(data=df, x='dayofweek', order=['Monday', 'Tuesday', 'Wednesday', 'Thu
plt.title("Traffic by Day of Week")
plt.xticks(rotation=45)
plt.show()
```


Traffic by Hour of Day
Count
Hour
0
10
20
30
40
50
60
0
5
10
15
20

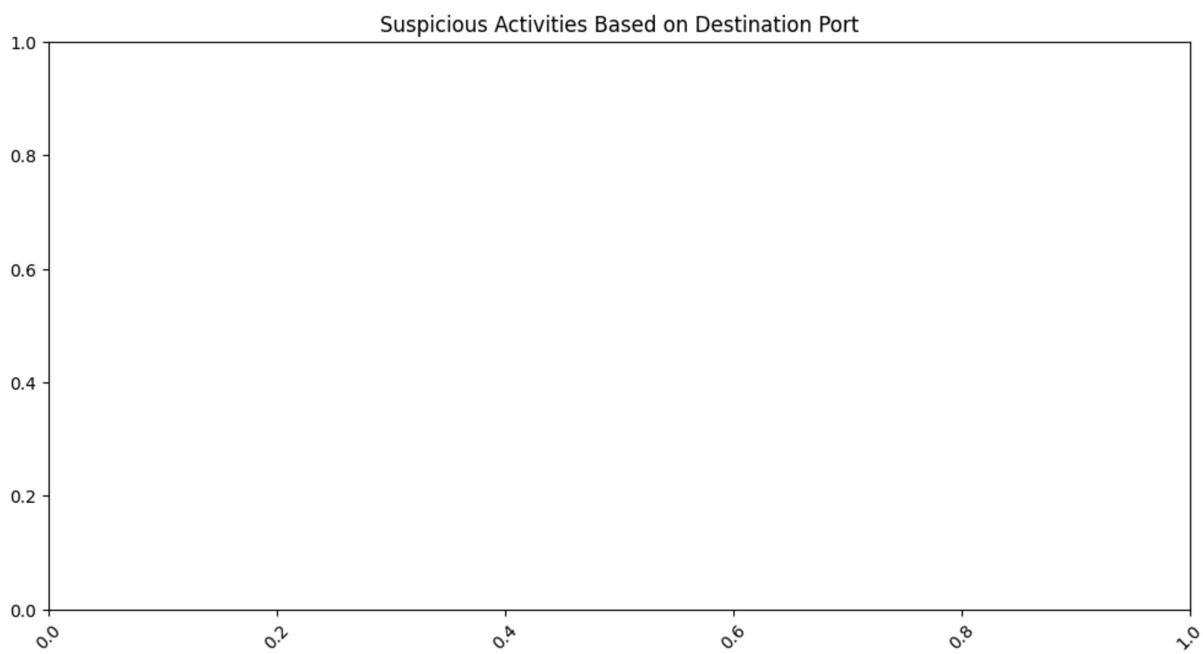


In [44]:
```
# Rule frequency
df['rule_names'].value_counts().head(10).plot(kind='barh', title="Most Triggered De

# Observations
df['observation_name'].value_counts().head(10).plot(kind='barh', title="Top Observe
```

Out[44]: <Axes: title={'center': 'Top Observed Suspicious Behaviors'}, ylabel='observation_name'>

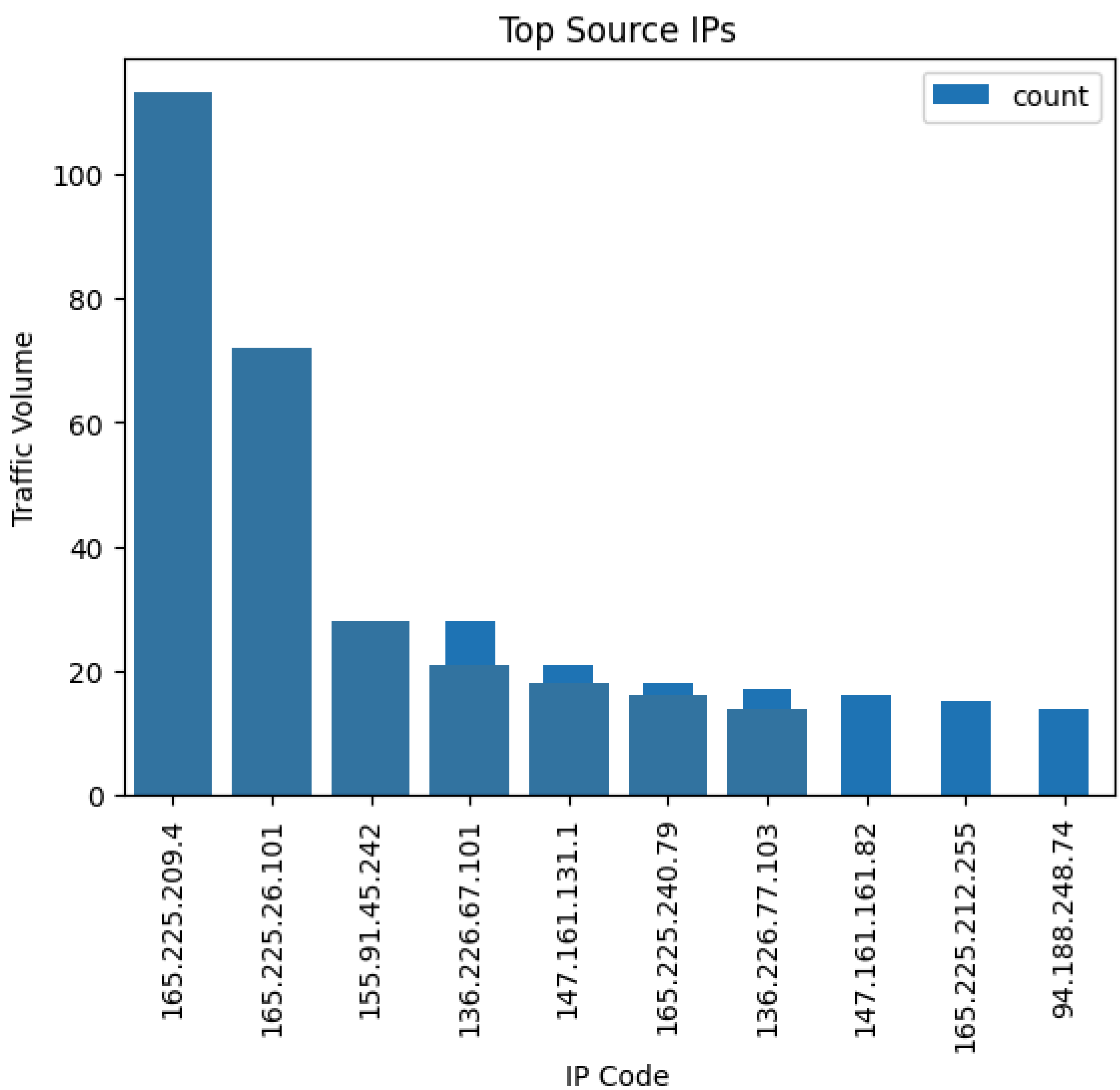


In [33]:
```
#Suspicious Activities Based on Ports
plt.figure(figsize=(12, 6))
sns.countplot(x='dst_port', data=df[df['detection_types'] =='Suspicious'], palette=
plt.title('Suspicious Activities Based on Destination Port')
plt.xticks(rotation=45)
plt.show()
```



In [40]:
```
# Top 10 Source IPs
df['src_ip'].value_counts().head(10).plot(kind='bar', title="Top 10 Source IPs")

# Source country activity
country_counts = df['src_ip_country_code'].value_counts().head(10)
sns.barplot(x=country_counts.index, y=country_counts.values)
plt.title("Top Source IPs")
plt.xlabel("IP Code")
```

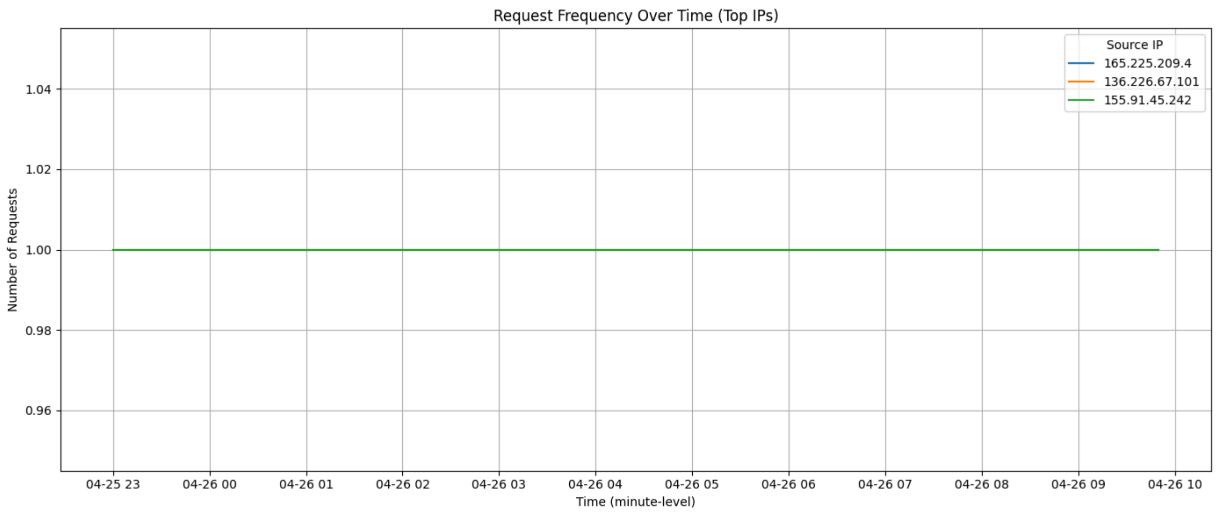
```python
plt.ylabel("Traffic Volume")
plt.show()
```



In [46]:

```python
# Convert 'time' to datetime if not already
# df['time'] = pd.to_datetime(df['time'])

# Set time index and group by IP and minute
df['minute'] = df['time'].dt.floor('min')
ip_minute_freq = df.groupby(['src_ip', 'minute']).size().reset_index(name='request_

# Plot request frequency for top suspicious IPs
top_ips = ip_minute_freq['src_ip'].value_counts().head(3).index

plt.figure(figsize=(14,6))
for ip in top_ips:
    subset = ip_minute_freq[ip_minute_freq['src_ip'] == ip]
    plt.plot(subset['minute'], subset['request_count'], label=ip)

plt.title("Request Frequency Over Time (Top IPs)")
plt.xlabel("Time (minute-level)")
plt.ylabel("Number of Requests")
plt.legend(title="Source IP")
```

```python
plt.grid(True)
plt.tight_layout()
plt.show()
```

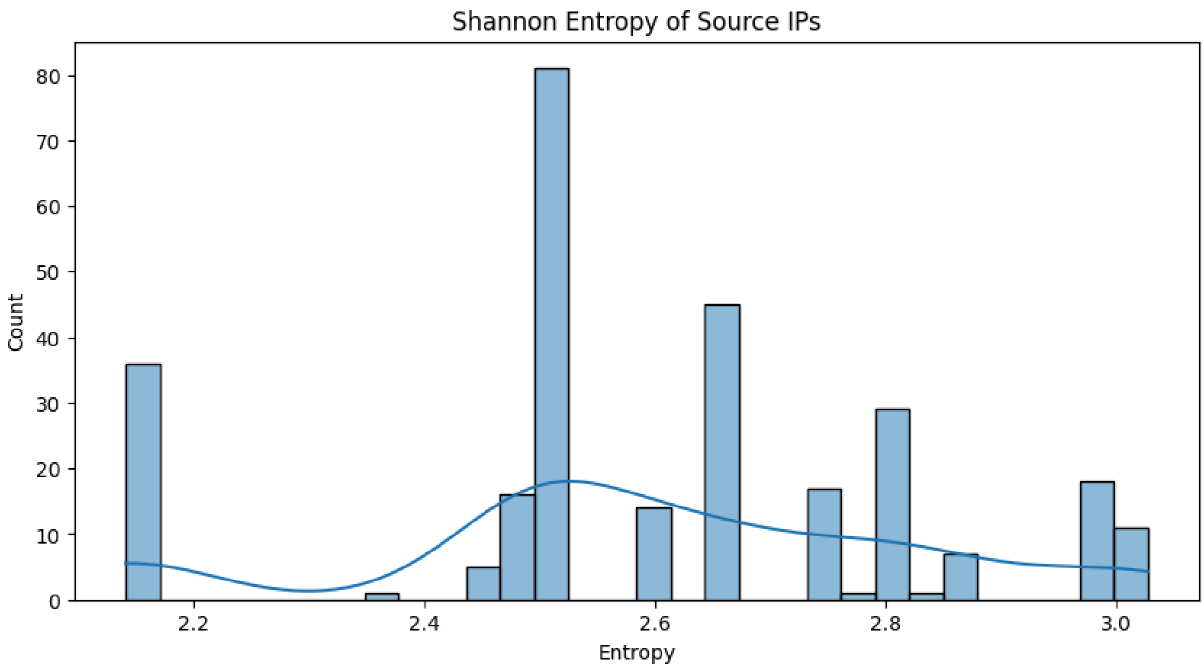


In [47]:

```python
import math

def shannon_entropy(s):
    """Calculate Shannon entropy of a string"""
    if not s:
        return 0
    prob = [float(s.count(c)) / len(s) for c in set(s)]
    entropy = -sum([p * math.log2(p) for p in prob])
    return entropy
```

In [48]:

```python
# Apply to source IPs (less common for entropy but demo-worthy)
df['ip_entropy'] = df['src_ip'].apply(shannon_entropy)

# OPTIONAL: if you have URLs or payloads, apply there
# df['url_entropy'] = df['url'].apply(shannon_entropy)

# Visualize
plt.figure(figsize=(10,5))
sns.histplot(df['ip_entropy'], bins=30, kde=True)
plt.title("Shannon Entropy of Source IPs")
plt.xlabel("Entropy")
plt.ylabel("Count")
plt.show()

# Top high-entropy IPs (possible obfuscation)
df[['src_ip', 'ip_entropy']].sort_values(by='ip_entropy', ascending=False).head(10)
```



Out[48]:

	src_ip	ip_entropy
238	136.226.80.97	3.026987
57	136.226.80.97	3.026987
216	136.226.80.97	3.026987
38	136.226.80.97	3.026987
30	136.226.80.97	3.026987
188	136.226.80.97	3.026987
203	136.226.80.97	3.026987
171	136.226.80.97	3.026987
164	136.226.80.97	3.026987
156	136.226.80.97	3.026987

In [50]:

```
df.head()
```

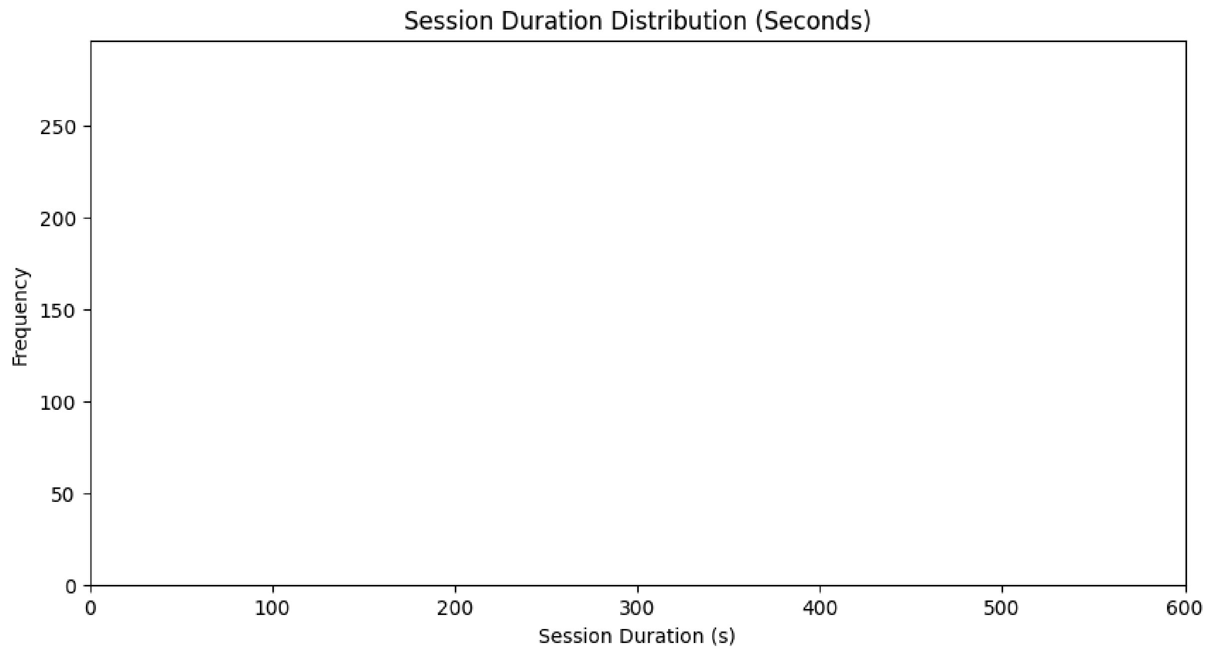
Out[50]:

	bytes_in	bytes_out	creation_time	end_time	src_ip	src_ip_country_code
0	5602	12990	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	147.161.161.82	AE
1	30912	18186	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	165.225.33.6	US
2	28506	13468	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	165.225.212.255	CA
3	30546	14278	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	136.226.64.114	US
4	6526	13892	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	165.225.240.79	NL

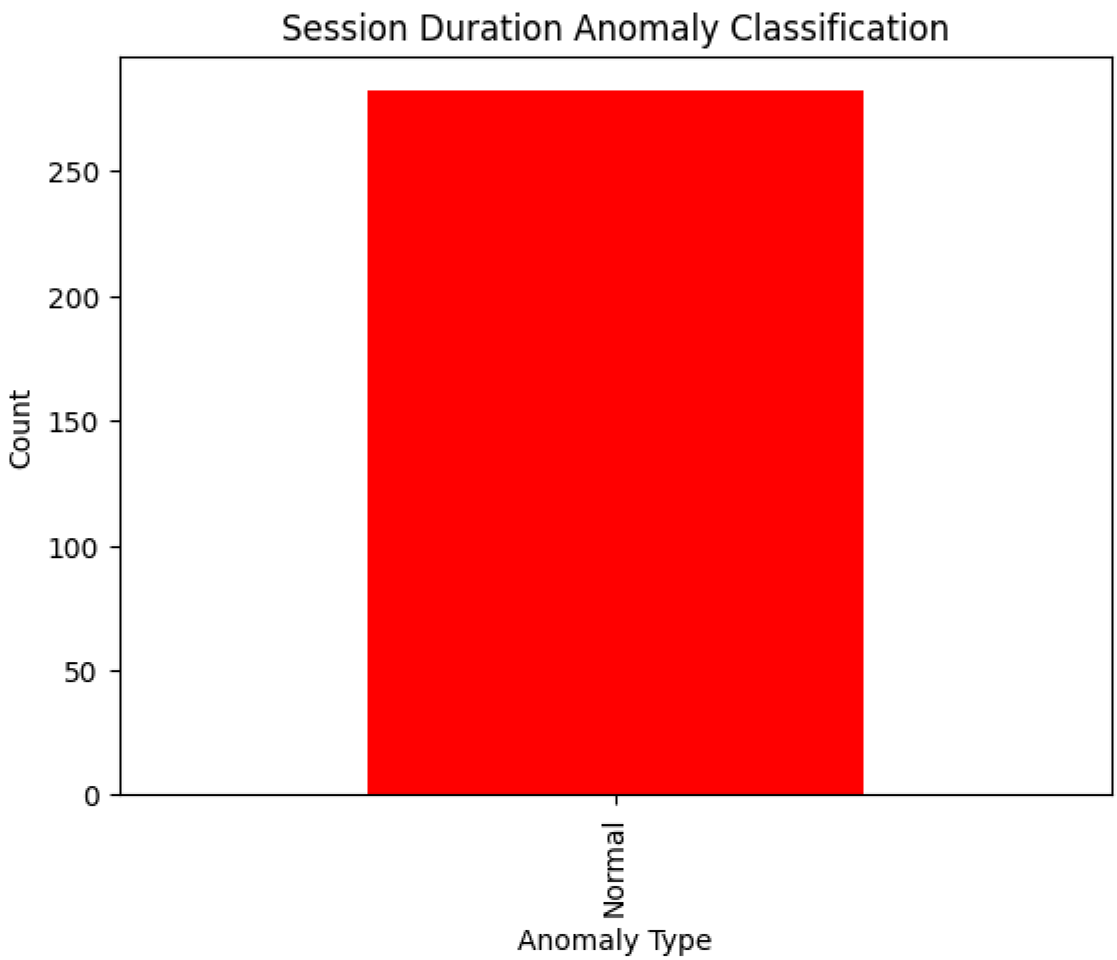
5 rows × 23 columns

In [52]:

```python
# Check distribution
plt.figure(figsize=(10, 5))
sns.histplot(df['session_duration'], bins=100, kde=True)
plt.title("Session Duration Distribution (Seconds)")
plt.xlabel("Session Duration (s)")
plt.ylabel("Frequency")
plt.xlim(0, df['session_duration'].quantile(0.99))  # Clip outliers
plt.show()

# Identify short and long session thresholds (e.g., 1st and 99th percentiles)
short_threshold = df['session_duration'].quantile(0.01)
long_threshold = df['session_duration'].quantile(0.99)

print(f"Short session threshold: < {short_threshold:.2f} seconds")
print(f"Long session threshold: > {long_threshold:.2f} seconds")

# Flag anomalies
df['duration_anomaly'] = df['session_duration'].apply(
    lambda x: 'Short' if x < short_threshold else ('Long' if x > long_threshold els
)

# Summary count
df['duration_anomaly'].value_counts().plot(kind='bar', color=['red', 'green', 'gray
plt.title("Session Duration Anomaly Classification")
plt.xlabel("Anomaly Type")
plt.ylabel("Count")
plt.show()

# Inspect a few records
df[df['duration_anomaly'] != 'Normal'][['src_ip', 'session_duration', 'detection_ty
```




```
Short session threshold: < 600.00 seconds
Long session threshold: > 600.00 seconds
```



Out[52]:

src_ip	session_duration	detection_types	dst_port

In [54]:

```
df.dtypes
```

Out[54]:
```
bytes_in                                    int64
bytes_out                                   int64
creation_time              datetime64[ns, UTC]
end_time                   datetime64[ns, UTC]
src_ip                                     object
src_ip_country_code                        object
protocol                                   object
response.code                               int64
dst_port                                    int64
dst_ip                                     object
rule_names                                 object
observation_name                           object
source.meta                                object
source.name                                object
time                       datetime64[ns, UTC]
detection_types                            object
session_duration                          float64
avg_packet_size                           float64
byte_ratio                                float64
hour                                        int32
dayofweek                                  object
minute                     datetime64[ns, UTC]
ip_entropy                                float64
duration_anomaly                           object
dtype: object
```

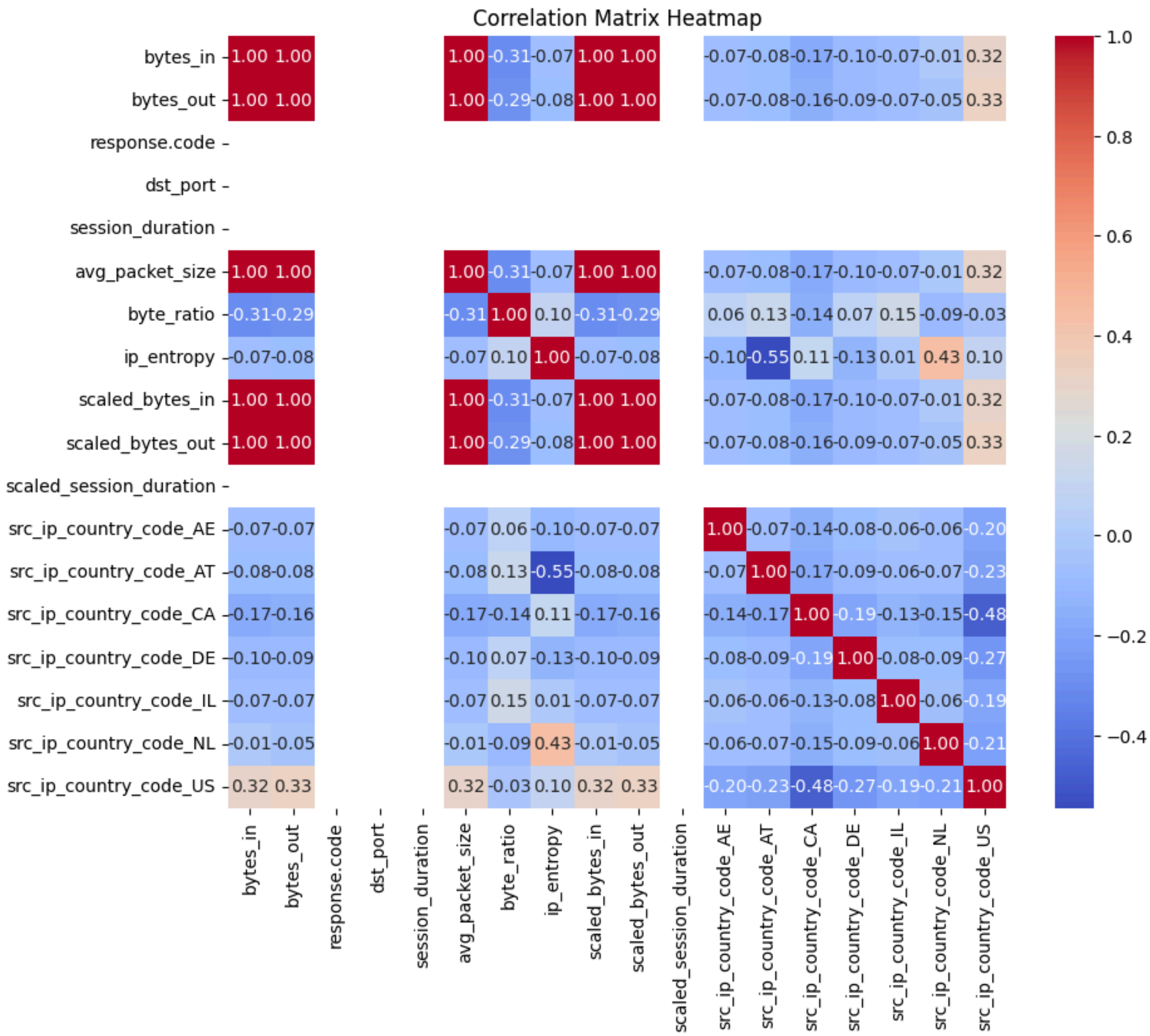
In [77]:
```
# Compute correlation matrix for numeric columns only
numeric_df = transformed_df.select_dtypes(include=['float64', 'int64'])
correlation_matrix_numeric = numeric_df.corr()
# Display the correlation matrix
correlation_matrix_numeric
```

Out[77]:

	bytes_in	bytes_out	response.code	dst_port	session_duration	av
bytes_in	1.000000	0.997705	NaN	NaN	NaN	
bytes_out	0.997705	1.000000	NaN	NaN	NaN	
response.code	NaN	NaN	NaN	NaN	NaN	
dst_port	NaN	NaN	NaN	NaN	NaN	
session_duration	NaN	NaN	NaN	NaN	NaN	
avg_packet_size	0.999992	0.997963	NaN	NaN	NaN	
byte_ratio	-0.306455	-0.290436	NaN	NaN	NaN	
ip_entropy	-0.072445	-0.084635	NaN	NaN	NaN	
scaled_bytes_in	1.000000	0.997705	NaN	NaN	NaN	
scaled_bytes_out	0.997705	1.000000	NaN	NaN	NaN	
scaled_session_duration	NaN	NaN	NaN	NaN	NaN	
src_ip_country_code_AE	-0.070559	-0.072452	NaN	NaN	NaN	
src_ip_country_code_AT	-0.081670	-0.081777	NaN	NaN	NaN	
src_ip_country_code_CA	-0.166488	-0.159587	NaN	NaN	NaN	
src_ip_country_code_DE	-0.095333	-0.090001	NaN	NaN	NaN	
src_ip_country_code_IL	-0.065939	-0.067630	NaN	NaN	NaN	
src_ip_country_code_NL	-0.006827	-0.045641	NaN	NaN	NaN	
src_ip_country_code_US	0.316015	0.327683	NaN	NaN	NaN	

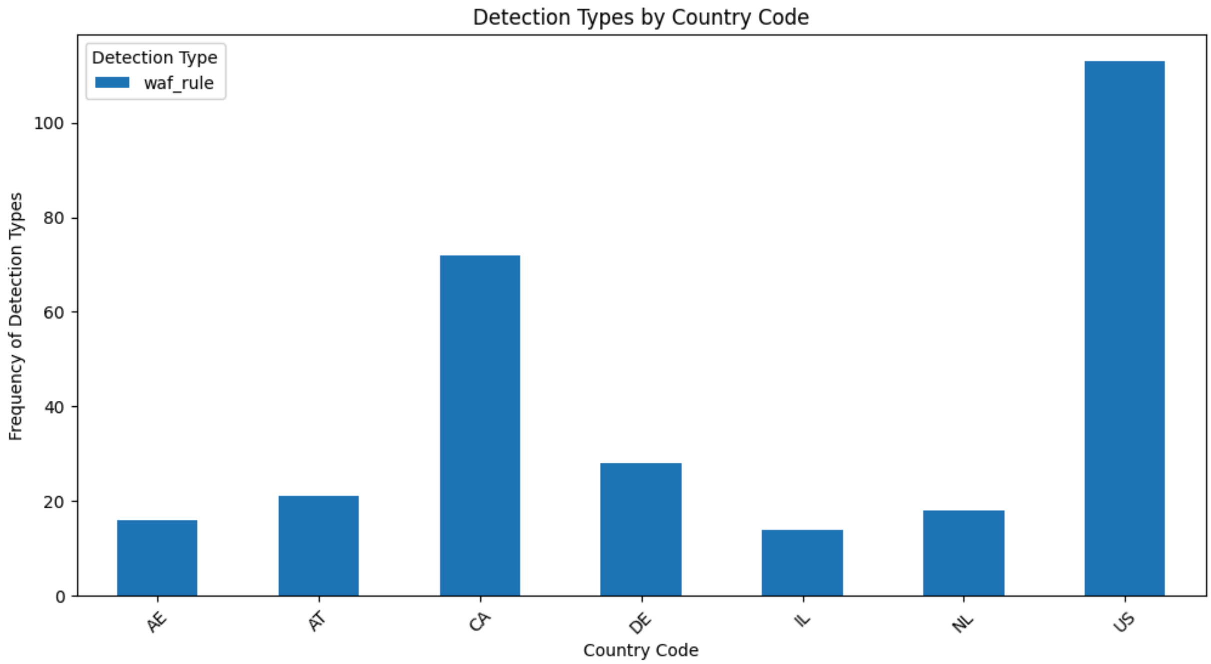
In [78]:

```
# Heatmap for the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix_numeric, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



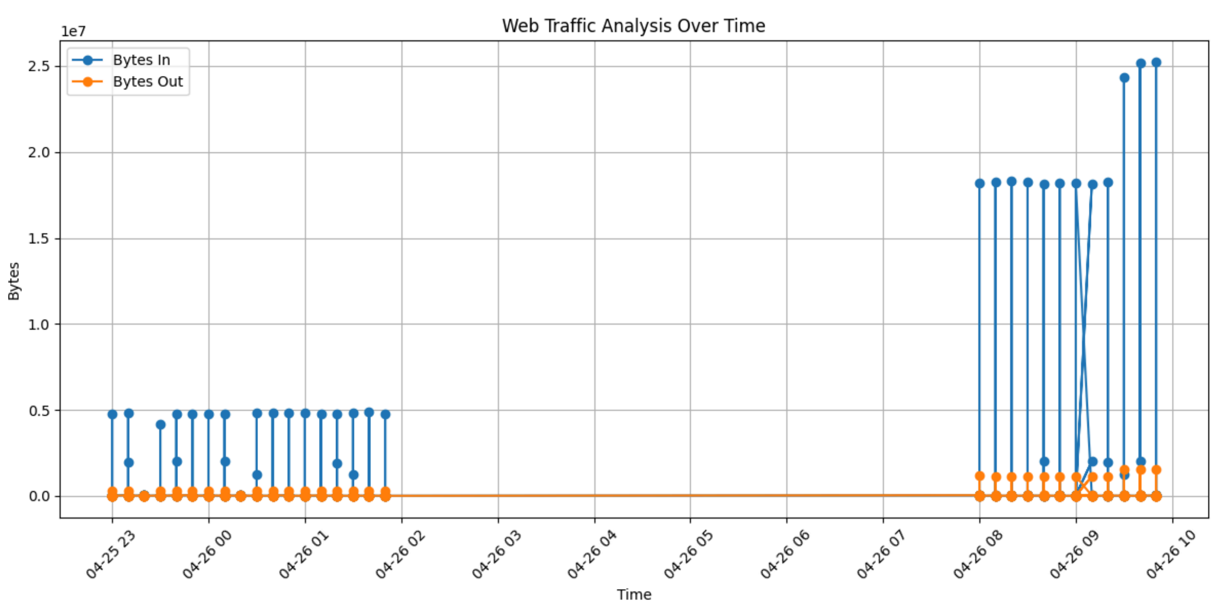
In [80]:

```
# Stacked Bar Chart for Detection Types by Country
# Preparing data for stacked bar chart
detection_types_by_country = pd.crosstab(transformed_df['src_ip_country_code'],tran
detection_types_by_country.plot(kind='bar', stacked=True, figsize=(12, 6))
plt.title('Detection Types by Country Code')
plt.xlabel('Country Code')
plt.ylabel('Frequency of Detection Types')
plt.xticks(rotation=45)
plt.legend(title='Detection Type')
plt.show()
```



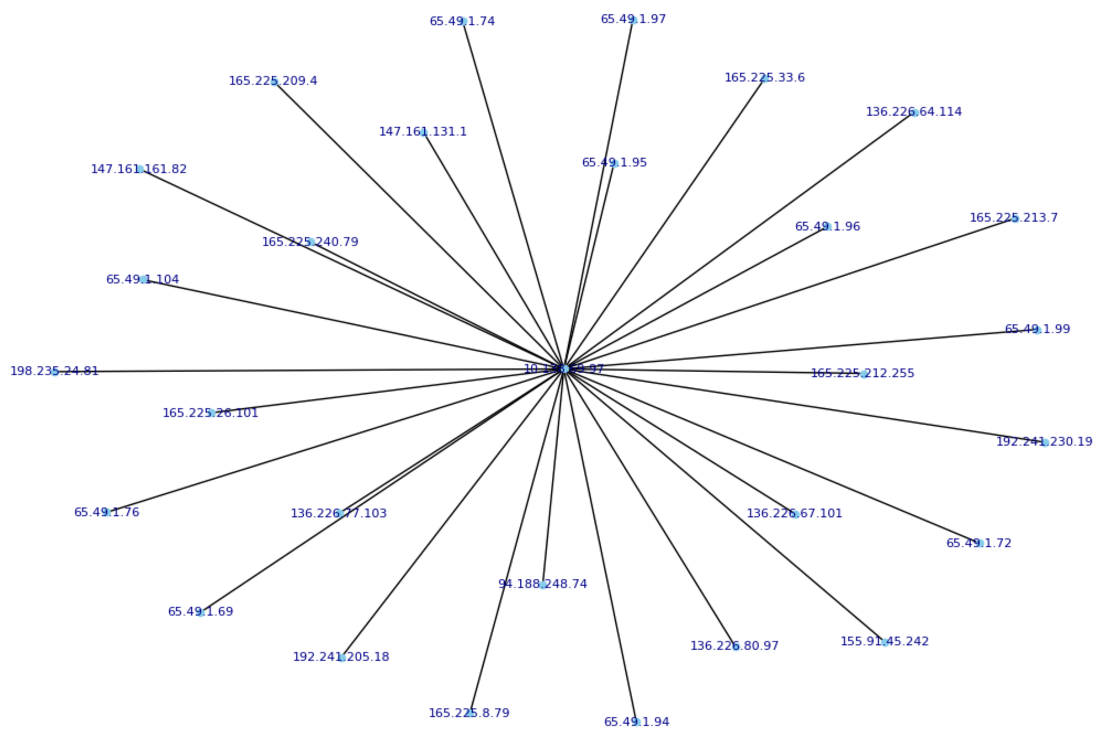
In [84]:

```
# Set 'creation_time' as the index
df.set_index('creation_time', inplace=True)

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['bytes_in'], label='Bytes In', marker='o')
plt.plot(df.index, df['bytes_out'], label='Bytes Out',marker='o')
plt.title('Web Traffic Analysis Over Time')
plt.xlabel('Time')
plt.ylabel('Bytes')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
# Show the plot
plt.show()
```



In [88]:
```
# Create a graph
G = nx.Graph()
# Add edges from source IP to destination IP
for idx, row in df.iterrows():
    G.add_edge(row['src_ip'], row['dst_ip'])
# Draw the network graph
plt.figure(figsize=(14, 10))
nx.draw_networkx(G, with_labels=True, node_size=20,font_size=8, node_color='skyblue
plt.title('Network Interaction between Source and Destination IPs')
plt.axis('off') # Turn off the axis
# Show the plot
plt.show()
```

Network Interaction between Source and Destination IPs



In [92]:
```
# RandomForestClassifier
# First, encode this column into binary labels
transformed_df['is_suspicious'] = (transformed_df['detection_types'] == 'waf_rule')

# Features and Labels
X = transformed_df[['bytes_in', 'bytes_out', 'scaled_session_duration']] # Numeric
y = transformed_df['is_suspicious'] # Binary labels
```

In [93]:
```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

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

```python
# Predict on the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification = classification_report(y_test, y_pred)

print("Model Accuracy: ",accuracy)
```

```
Model Accuracy:  1.0
```

In [94]:
```python
print("Classification Report: ",classification)
```

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

           1       1.00      1.00      1.00        85

    accuracy                           1.00        85
   macro avg       1.00      1.00      1.00        85
weighted avg       1.00      1.00      1.00        85
```

Feature Scaling

In [60]:
```python
df.columns
```

Out[60]:
```
Index(['bytes_in', 'bytes_out', 'creation_time', 'end_time', 'src_ip',
       'src_ip_country_code', 'protocol', 'response.code', 'dst_port',
       'dst_ip', 'rule_names', 'observation_name', 'source.meta',
       'source.name', 'time', 'detection_types', 'session_duration',
       'avg_packet_size', 'byte_ratio', 'hour', 'dayofweek', 'minute',
       'ip_entropy', 'duration_anomaly'],
      dtype='object')
```

In [69]:
```python
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
```

In [66]:
```python
import networkx as nx
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
import warnings
warnings.filterwarnings("ignore")
```

In [61]:
```
# Preparing column transformations
# StandardScaler for numerical features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[['bytes_in','bytes_out', 'session_duratio
```

In [73]:
```
# OneHotEncoder for categorical features
ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
encoded_features = ohe.fit_transform(df[['src_ip_country_code']])

# Combining transformed features back into the DataFrame
scaled_columns = ['scaled_bytes_in', 'scaled_bytes_out', 'scaled_session_duration']
encoded_columns = ohe.get_feature_names_out(['src_ip_country_code'])
```

In [75]:
```
# Convert numpy arrays back to DataFrame
scaled_df = pd.DataFrame(scaled_features, columns=scaled_columns, index=df.index)
encoded_df = pd.DataFrame(encoded_features, columns=encoded_columns, index=df.index
```

In [76]:
```
# Concatenate all the data back together
transformed_df = pd.concat([df, scaled_df, encoded_df], axis=1)

# Displaying the transformed data
transformed_df.head()
```

Out[76]:

	bytes_in	bytes_out	creation_time	end_time	src_ip	src_ip_country_code
0	5602	12990	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	147.161.161.82	AE
1	30912	18186	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	165.225.33.6	US
2	28506	13468	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	165.225.212.255	CA
3	30546	14278	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	136.226.64.114	US
4	6526	13892	2024-04-25 23:00:00+00:00	2024-04-25 23:10:00+00:00	165.225.240.79	NL

5 rows × 34 columns

In [ ]:
```
Neural Network
```

In [97]:
```
df['is_suspicious'] = (df['detection_types'] == 'waf_rule').astype(int)
# Features and labels
X = df[['bytes_in', 'bytes_out']].values # Using only numeric features
y = df['is_suspicious'].values
# 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, random_sta
# Normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Neural network model
model = Sequential([
Dense(8, activation='relu',
input_shape=(X_train_scaled.shape[1],)),
Dense(16, activation='relu'),
Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train_scaled, y_train, epochs=10,
batch_size=8, verbose=1)
# Evaluate the model
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

```
Epoch 1/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5148 - loss: 0.6897
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.6265
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.5690
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.4959
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.4139
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.3187
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.2286
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.1537
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.1048
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0761
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 1.0000 - loss: 0.0590
Test Accuracy: 100.00%
```

In [100…

```
from sklearn.ensemble import IsolationForest
# Selecting features for anomaly detection
features = df[['bytes_in', 'bytes_out', 'session_duration','avg_packet_size']]
# Initialize the model
model = IsolationForest(contamination=0.05, random_state=42)
# Fit and predict anomalies
df['anomaly'] = model.fit_predict(features)
df['anomaly'] = df['anomaly'].apply(lambda x: 'Suspicious' if x == -1 else 'Normal'
```

In [101…

```
# 7. Evaluation
# Evaluate the anomaly detection model by checking its accuracy in identifying susp
# Check the proportion of anomalies detected
print(df['anomaly'].value_counts())
# Display anomaly samples
suspicious_activities = df[df['anomaly'] == 'Suspicious']
print(suspicious_activities.head())
```

```
anomaly
Normal        267
Suspicious     15
Name: count, dtype: int64
                           bytes_in  bytes_out                  end_time  \
creation_time
2024-04-25 23:30:00+00:00   4190330     283456 2024-04-25 23:40:00+00:00
2024-04-26 00:30:00+00:00   1215594      64362 2024-04-26 00:40:00+00:00
2024-04-26 01:00:00+00:00   4827283     306181 2024-04-26 01:10:00+00:00
2024-04-26 01:20:00+00:00   1889834      34306 2024-04-26 01:30:00+00:00
2024-04-26 01:40:00+00:00   4869181     301752 2024-04-26 01:50:00+00:00

                                   src_ip src_ip_country_code protocol  \
creation_time
2024-04-25 23:30:00+00:00   155.91.45.242                  US    HTTPS
2024-04-26 00:30:00+00:00  165.225.240.79                  NL    HTTPS
2024-04-26 01:00:00+00:00   155.91.45.242                  US    HTTPS
2024-04-26 01:20:00+00:00  165.225.240.79                  NL    HTTPS
2024-04-26 01:40:00+00:00   155.91.45.242                  US    HTTPS

                           response.code  dst_port        dst_ip  \
creation_time
2024-04-25 23:30:00+00:00            200       443  10.138.69.97
2024-04-26 00:30:00+00:00            200       443  10.138.69.97
2024-04-26 01:00:00+00:00            200       443  10.138.69.97
2024-04-26 01:20:00+00:00            200       443  10.138.69.97
2024-04-26 01:40:00+00:00            200       443  10.138.69.97

                                       rule_names  ... session_duration  \
creation_time                                      ...
2024-04-25 23:30:00+00:00  Suspicious Web Traffic  ...            600.0
2024-04-26 00:30:00+00:00  Suspicious Web Traffic  ...            600.0
2024-04-26 01:00:00+00:00  Suspicious Web Traffic  ...            600.0
2024-04-26 01:20:00+00:00  Suspicious Web Traffic  ...            600.0
2024-04-26 01:40:00+00:00  Suspicious Web Traffic  ...            600.0

                          avg_packet_size byte_ratio hour dayofweek  \
creation_time
2024-04-25 23:30:00+00:00     7456.310000   0.067645   23  Thursday
2024-04-26 00:30:00+00:00     2133.260000   0.052947    0    Friday
2024-04-26 01:00:00+00:00     8555.773333   0.063427    1    Friday
2024-04-26 01:20:00+00:00     3206.900000   0.018153    1    Friday
2024-04-26 01:40:00+00:00     8618.221667   0.061972    1    Friday

                                             minute  ip_entropy  \
creation_time
2024-04-25 23:30:00+00:00 2024-04-25 23:30:00+00:00    2.507380
2024-04-26 00:30:00+00:00 2024-04-26 00:30:00+00:00    2.985228
2024-04-26 01:00:00+00:00 2024-04-26 01:00:00+00:00    2.507380
2024-04-26 01:20:00+00:00 2024-04-26 01:20:00+00:00    2.985228
2024-04-26 01:40:00+00:00 2024-04-26 01:40:00+00:00    2.507380

                           duration_anomaly  is_suspicious     anomaly
creation_time
2024-04-25 23:30:00+00:00            Normal              1  Suspicious
2024-04-26 00:30:00+00:00            Normal              1  Suspicious
```

```
2024-04-26 01:00:00+00:00           Normal              1  Suspicious
2024-04-26 01:20:00+00:00           Normal              1  Suspicious
2024-04-26 01:40:00+00:00           Normal              1  Suspicious

[5 rows x 25 columns]
```

In [103…

```python
# 8. Visualization of Anomalies
# Visualize bytes_in vs bytes_out with anomalies highlighted
plt.figure(figsize=(10, 6))
sns.scatterplot(x='bytes_in', y='bytes_out', hue='anomaly',
data=df, palette=['green', 'red'])
plt.title('Anomalies in Bytes In vs Bytes Out')
plt.show()
```