Project Title : Cybersecurity: Suspicious Web Threat Interactions

language : Machine learning, python, SQL, Excel

Tools : VS code, Jupyter notebook, Google Colab (Collaboratory

Dataset : Dataset is available in the given link. https://drive.google.com/file/d/1-OpnR9FK8EqGuLFB1k45ctPbl-vuZnC-/view

About Dataset

This dataset contains web traffic records collected through AWS CloudWatch, aimed at detecting suspicious activities and potential attack attempts. The data were generated by monitoring traffic to a production web server, using various detection rules to identify anomalous patterns.

Context

In today's cloud environments, cybersecurity is more crucial than ever. The ability to detect and respond to threats in real time can protect organizations from significant consequences. This dataset provides a view of web traffic that has been labeled as suspicious, offering a valuable resource for developers, data scientists, and security experts to enhance threat detection techniques.

Potential Uses

This dataset is ideal for:

- Anomaly Detection: Developing models to detect unusual behaviors in web traffic.
- Classification Models: Training models to automatically classify traffic as normal or suspicious.
- Security Analysis: Conducting security analyses to understand the tactics, techniques, and procedures of attackers

Data Preprocessing

Missing Values: Identify and handle missing data (impute or remove) to avoid bias and errors.

Outliers: Detect and treat extreme values that can skew results. Options include removal, transformation, or capping.

Inconsistencies: Standardize formats and correct errors in the data (e.g., text, dates, units) for uniformity and accuracy.

Exploratory Data Analysis (EDA):

A significant stage in the process of summarizing, describing, and comprehending the underlying patterns in the data is the performing of statistical analysis. Examining several aspects such as distributions, central trends, variability, and correlations between characteristics is included in this. On your converted dataset, let's carry out a number of statistical analysis, including the following:

Descriptive Statistics : This includes mean, median, mode, min, max, range, quartiles, and standard deviations.

Correlation Analysis : To investigate the relationships between numerical features and how they relate to each other.

Distribution Analysis: Examine the distribution of key features using histograms and box plots to identify the spread and presence of outliers.

Normalization and Scaling

Normalization or scaling ensures that numeric features contribute equally to model training.

Common methods include:

- Min-Max Scaling : Transforms features to a fixed range, usually 0 to 1.
- Standardization (Z-score Scaling) : Centers the data by removing the mean and scales it by the standard deviation to achieve a variance of 1 and mean of 0

Encoding Categorical Data

Machine learning models generally require all input and output variables to be numeric. This means that categorical data must be converted into a numerical format.

- One-Hot Encoding : Creates a binary column for each category and returns a matrix with 1s and 0s.
- Label Encoding : Converts each value in a column to a number.

Feature Engineering

Feature engineering is the process of using domain knowledge to select, modify, or create new features that increase the predictive power of the learning algorithm

● Polynomial Features : Derive new feature interactions.

● Binning : Convert numerical values into categorical bins.

Descriptive Statistics

The descriptive statistics provide a summary of the key statistical characteristics of

the numerical features:

● bytes_in and bytes_out : These columns have a high standard deviation relative to their mean, indicating significant variability. This could be reflective of different types of web sessions or activities.

● response.code and dst_port : These fields are constants in the dataset (200 and 443, respectively), indicating all records are using HTTPS protocol on standard port 443 and receiving a standard HTTP 200 OK response.

● duration_seconds : It's also constant (600 seconds), which suggests that each session or observation is recorded over a fixed interval.

● Scaled Features : The scaled versions of bytes_in, bytes_out, and duration_seconds have a mean of approximately 0 and a standard deviation of 1, as expected after standardization.

Correlation Heatmap:
sns.heatmap() visualizes the correlation matrix, making it easier to identify strong correlations.

Stacked Bar Chart for Detection Types by Country :

Visualizing the distribution of different detection types across various source countries.

Modeling (Anomaly Detection Example):
As an example of a modeling approach, we implemented the Isolation Forest algorithm for unsupervised anomaly detection. This allowed us to identify data points that deviate significantly from the norm, which could represent potential security threats. We obtained anomaly scores and predictions, which can be further analyzed to understand the nature of these anomalies.

Web Traffic Analysis Over Time: Plotting the bytes_in and bytes_out over the creation_time to identify temporal trends.

Network Interaction Graph: Using NetworkX to visualize the connections between source and destination IP addresses as a network graph.

Supervised Learning - Random Forest Classifier (Example): Implementing a basic machine learning model (Random Forest) to predict a binary is_suspicious label based on whether the detection_types was 'waf_rule'. This included splitting the data into training and testing sets, training the model, and evaluating its performance.

Neural Network Model (Basic): Building and training a simple feedforward neural network using TensorFlow/Keras for the same binary classification task. This involved feature scaling and evaluating the model's accuracy.

Neural Network with Dropout and Validation: Enhancing the neural network by adding dropout layers for regularization and using a validation split during training to monitor performance on unseen data within the training set. We also plotted the training and validation accuracy and loss curves.

1D Convolutional Neural Network (CNN) Model: Implementing a 1D CNN to see if it could learn patterns from the bytes_in and bytes_out features. This involved reshaping the input data to fit the CNN's expected format and using convolutional layers for feature extraction. We again trained, evaluated, and plotted the training history.

Conclusion:
this project has successfully demonstrated a comprehensive workflow for analyzing cybersecurity data, from initial understanding to the application of an anomaly detection model. The insights gained and the prepared dataset provide a valuable starting point for building more sophisticated threat detection systems.

Tools and Libraries:

We utilized several key Python libraries:

Pandas: For data manipulation and analysis.

NumPy: For numerical operations.

Matplotlib and Seaborn: For creating various types of visualizations.

Scikit-learn: For machine learning tasks like data splitting, feature scaling, and implementing the Random Forest classifier

TensorFlow/Keras : For building and training neural network models.

NetworkX: For creating and analyzing network graphs.

```
import pandas as pd

# If your dataset is directly accessible via the provided link, you might be able to load it directly.
# However, it's more common to download it first and then upload it to Colab.
# Assuming you have uploaded 'cybersecurity_data.csv' to your Colab environment:
try:
    df = pd.read_csv('CloudWatch_Traffic_Web_Attack.csv')
    print("Dataset loaded successfully!")
except FileNotFoundError:
    print("Error: 'CloudWatch_Traffic_Web_Attack.csv' not found. Please upload the file to Colab.")
    df = None

if df is not None:
    print("\nBasic Information:")
    df.info()
    print("\nFirst 5 rows:")
    print(df.head())
```

```
Dataset loaded successfully!

Basic Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 282 entries, 0 to 281
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   bytes_in             282 non-null    int64
 1   bytes_out            282 non-null    int64
 2   creation_time        282 non-null    object
 3   end_time             282 non-null    object
 4   src_ip               282 non-null    object
 5   src_ip_country_code  282 non-null    object
 6   protocol             282 non-null    object
 7   response.code        282 non-null    int64
 8   dst_port             282 non-null    int64
 9   dst_ip               282 non-null    object
 10  rule_names           282 non-null    object
 11  observation_name     282 non-null    object
 12  source.meta          282 non-null    object
 13  source.name          282 non-null    object
 14  time                 282 non-null    object
 15  detection_types      282 non-null    object
dtypes: int64(4), object(12)
memory usage: 35.4+ KB

First 5 rows:
   bytes_in  bytes_out         creation_time              end_time  \
0      5602      12990  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
1     30912      18186  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
2     28506      13468  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
3     30546      14278  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
4      6526      13892  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z

            src_ip src_ip_country_code protocol  response.code  dst_port  \
0   147.161.161.82                  AE    HTTPS            200       443
1     165.225.33.6                  US    HTTPS            200       443
2  165.225.212.255                  CA    HTTPS            200       443
3   136.226.64.114                  US    HTTPS            200       443
4   165.225.240.79                  NL    HTTPS            200       443

          dst_ip              rule_names                     observation_name  \
0  10.138.69.97  Suspicious Web Traffic  Adversary Infrastructure Interaction
1  10.138.69.97  Suspicious Web Traffic  Adversary Infrastructure Interaction
2  10.138.69.97  Suspicious Web Traffic  Adversary Infrastructure Interaction
3  10.138.69.97  Suspicious Web Traffic  Adversary Infrastructure Interaction
4  10.138.69.97  Suspicious Web Traffic  Adversary Infrastructure Interaction

    source.meta     source.name                 time detection_types
0  AWS_VPC_Flow  prod_webserver  2024-04-25T23:00:00Z        waf_rule
1  AWS_VPC_Flow  prod_webserver  2024-04-25T23:00:00Z        waf_rule
2  AWS_VPC_Flow  prod_webserver  2024-04-25T23:00:00Z        waf_rule
3  AWS_VPC_Flow  prod_webserver  2024-04-25T23:00:00Z        waf_rule
4  AWS_VPC_Flow  prod_webserver  2024-04-25T23:00:00Z        waf_rule
```

Data Preprocessing

```
if df is not None:
    print("\n--- Data Preprocessing ---")

    # Check for missing values
    missing_values = df.isnull().sum()
    print("\nMissing Values:\n", missing_values)

    # Fill or drop missing values as needed (example)
    if 'bytes_in' in df.columns:
        df['bytes_in'].fillna(df['bytes_in'].median(), inplace=True)
        print("\n'bytes_in' missing values filled with median.")

    if 'src_ip' in df.columns and 'dst_ip' in df.columns:
        df.dropna(subset=['src_ip', 'dst_ip'], inplace=True)
        print("\nRows with missing 'src_ip' or 'dst_ip' dropped.")

    # Convert columns to appropriate datatypes
    if 'creation_time' in df.columns:
        df['creation_time'] = pd.to_datetime(df['creation_time'], errors='coerce')
        print("\n'creation_time' converted to datetime.")

    if 'end_time' in df.columns:
        df['end_time'] = pd.to_datetime(df['end_time'], errors='coerce')
        print("'end_time' converted to datetime.")

    if 'time' in df.columns:
        df['time'] = pd.to_datetime(df['time'], errors='coerce')
        print("'time' converted to datetime.")

    print("\nProcessed Data Info:")
    df.info()
```

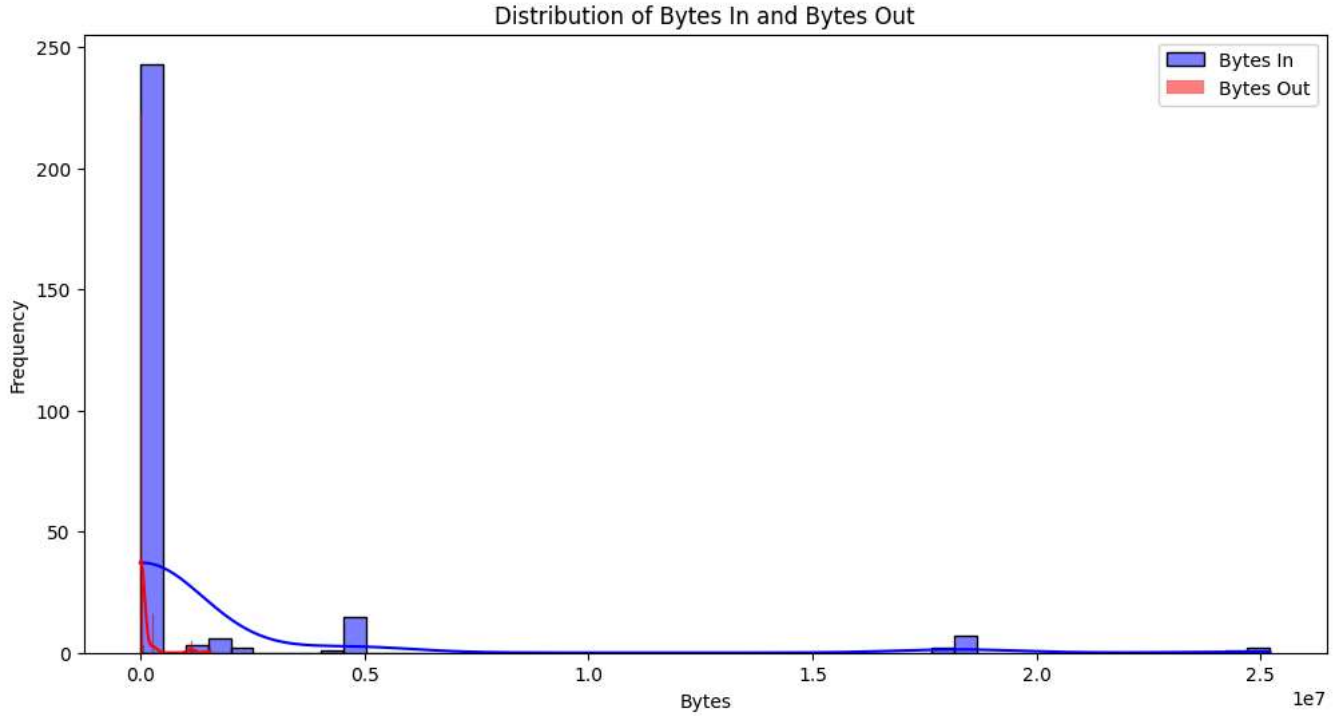
```
--- Data Preprocessing ---

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

'bytes_in' missing values filled with median.

Rows with missing 'src_ip' or 'dst_ip' dropped.

'creation_time' converted to datetime.
'end_time' converted to datetime.
'time' converted to datetime.

Processed Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 282 entries, 0 to 281
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   bytes_in             282 non-null    int64
 1   bytes_out            282 non-null    int64
 2   creation_time        282 non-null    datetime64[ns, UTC]
 3   end_time             282 non-null    datetime64[ns, UTC]
 4   src_ip               282 non-null    object
 5   src_ip_country_code  282 non-null    object
 6   protocol             282 non-null    object
 7   response.code        282 non-null    int64
 8   dst_port             282 non-null    int64
```

```
 9   dst_ip               282 non-null    object
 10  rule_names           282 non-null    object
 11  observation_name     282 non-null    object
 12  source.meta          282 non-null    object
 13  source.name          282 non-null    object
 14  time                 282 non-null    datetime64[ns, UTC]
 15  detection_types      282 non-null    object
dtypes: datetime64[ns, UTC](3), int64(4), object(9)
memory usage: 35.4+ KB
<ipython-input-16-2bdb6fbedb4c>:10: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assi
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting va
```

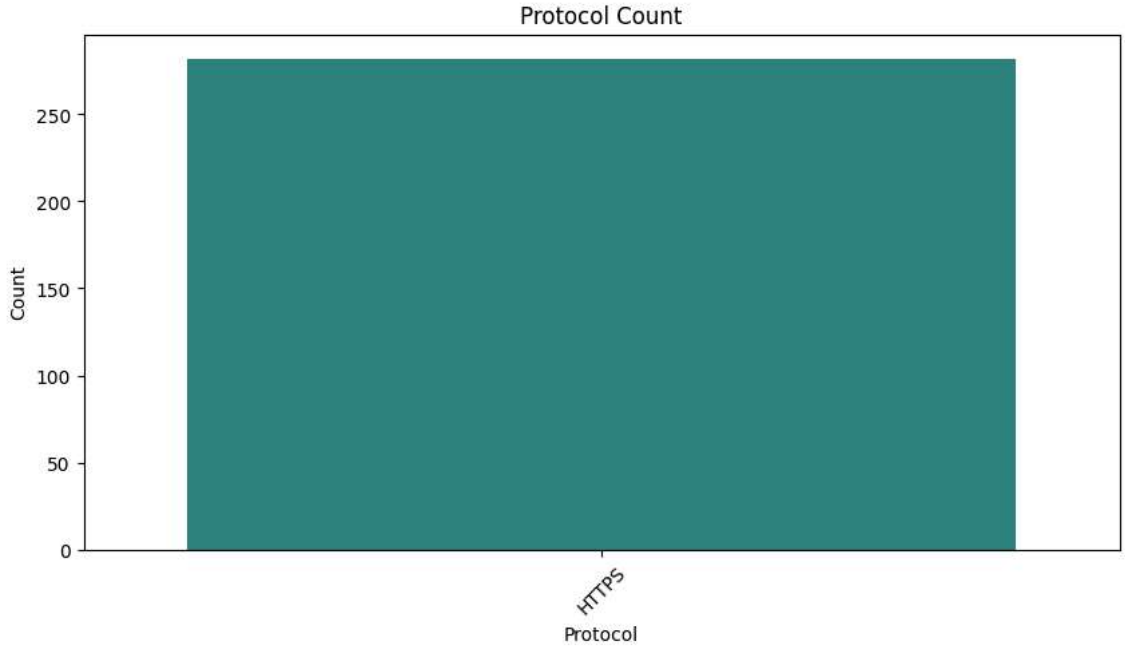
Exploratory Data Analysis (EDA):

```
if df is not None:
    print("\n--- Exploratory Data Analysis ---")
    import matplotlib.pyplot as plt
    import seaborn as sns

    # Analyze Traffic Patterns Based on bytes_in and bytes_out
    if 'bytes_in' in df.columns and 'bytes_out' in df.columns:
        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.xlabel('Bytes')
        plt.ylabel('Frequency')
        plt.show()
    else:
        print("Columns 'bytes_in' or 'bytes_out' not found for plotting.")

    # Count of Protocols Used
    if 'protocol' in df.columns:
        plt.figure(figsize=(10, 5))
        sns.countplot(x='protocol', data=df, palette='viridis')
        plt.title('Protocol Count')
        plt.xlabel('Protocol')
        plt.ylabel('Count')
        plt.xticks(rotation=45)
        plt.show()
    else:
        print("Column 'protocol' not found for plotting.")
```

```
--- Exploratory Data Analysis ---
```




```
<ipython-input-17-e8eed8ee54ec>:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  sns.countplot(x='protocol', data=df, palette='viridis')
```



4. Feature Engineering

```
if df is not None:
    print("\n--- Feature Engineering ---")

    # Duration of the session in seconds
    if 'end_time' in df.columns and 'creation_time' in df.columns:
        df['session_duration'] = (df['end_time'] - df['creation_time']).dt.total_seconds()
        print("\n'session_duration' feature created (in seconds).")
    else:
        print("\n'end_time' or 'creation_time' columns not found for calculating session duration.")
        df['session_duration'] = None
```

```
# Average packet size (handling potential division by zero)
if 'bytes_in' in df.columns and 'bytes_out' in df.columns and 'session_duration' in df.columns and df['session_duration'] is not None:
    df['avg_packet_size'] = (df['bytes_in'] + df['bytes_out']) / (df['session_duration'] + 1e-9) # Adding a small epsilon to avoid divis
    print("'avg_packet_size' feature created.")
else:
    print("'bytes_in', 'bytes_out', or 'session_duration' columns not found for calculating average packet size.")
    df['avg_packet_size'] = None

print("\nUpdated Data Info after Feature Engineering:")
df.info()
print("\nFirst 5 rows with new features:")
print(df.head())
```

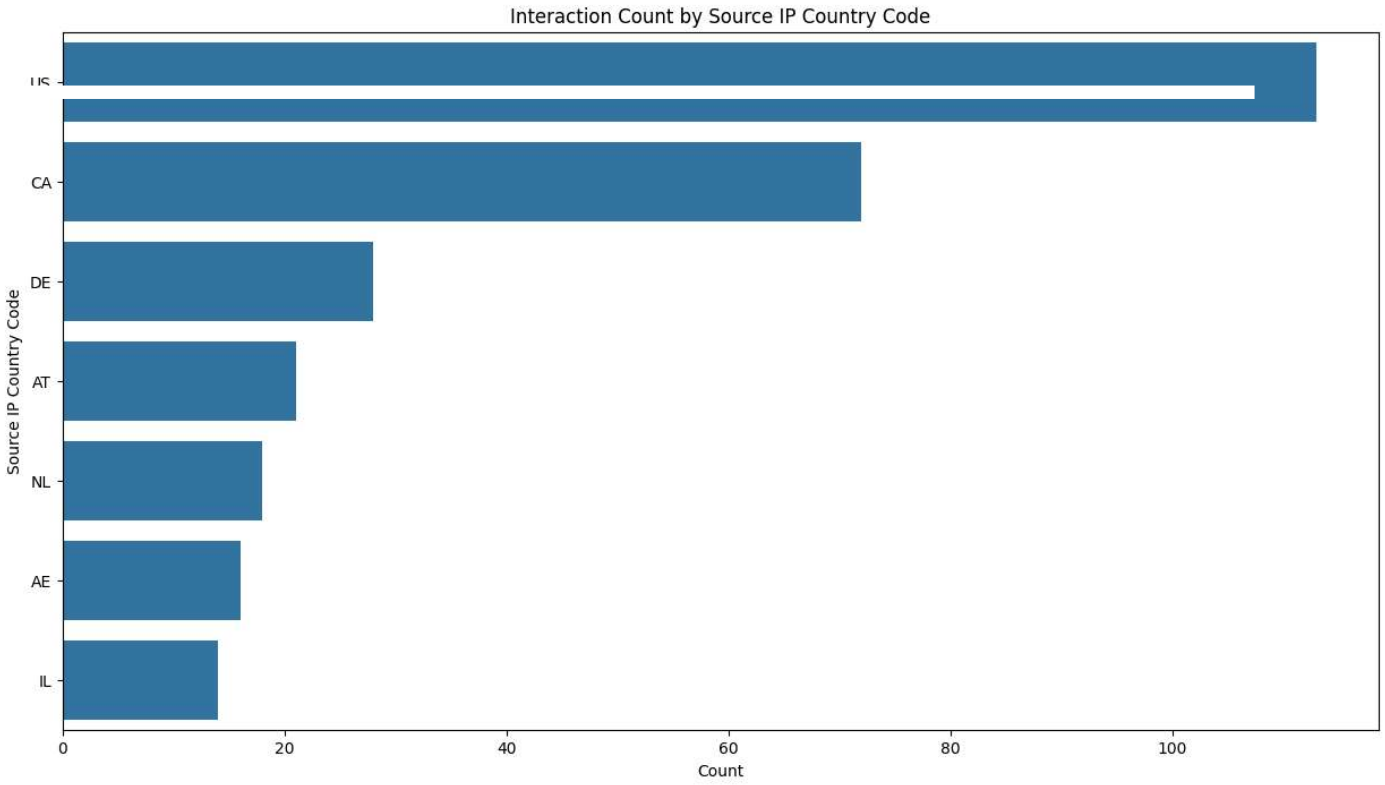
```
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   bytes_in             282 non-null    int64
 1   bytes_out            282 non-null    int64
 2   creation_time        282 non-null    datetime64[ns, UTC]
 3   end_time             282 non-null    datetime64[ns, UTC]
 4   src_ip               282 non-null    object
 5   src_ip_country_code  282 non-null    object
 6   protocol             282 non-null    object
 7   response.code        282 non-null    int64
 8   dst_port             282 non-null    int64
 9   dst_ip               282 non-null    object
 10  rule_names           282 non-null    object
 11  observation_name     282 non-null    object
 12  source.meta          282 non-null    object
 13  source.name          282 non-null    object
 14  time                 282 non-null    datetime64[ns, UTC]
 15  detection_types      282 non-null    object
 16  session_duration     282 non-null    float64
 17  avg_packet_size      282 non-null    float64
dtypes: datetime64[ns, UTC](3), float64(2), int64(4), object(9)
memory usage: 39.8+ KB

First 5 rows with new features:
   bytes_in  bytes_out             creation_time                  end_time  \
0      5602      12990 2024-04-25 23:00:00+00:00 2024-04-25 23:10:00+00:00
1     30912      18186 2024-04-25 23:00:00+00:00 2024-04-25 23:10:00+00:00
2     28506      13468 2024-04-25 23:00:00+00:00 2024-04-25 23:10:00+00:00
3     30546      14278 2024-04-25 23:00:00+00:00 2024-04-25 23:10:00+00:00
4      6526      13892 2024-04-25 23:00:00+00:00 2024-04-25 23:10:00+00:00

            src_ip src_ip_country_code protocol  response.code  dst_port  \
0   147.161.161.82                  AE    HTTPS            200       443
1     165.225.33.6                  US    HTTPS            200       443
2  165.225.212.255                  CA    HTTPS            200       443
3   136.226.64.114                  US    HTTPS            200       443
4   165.225.240.79                  NL    HTTPS            200       443

          dst_ip              rule_names                     observation_name  \
0  10.138.69.97  Suspicious Web Traffic  Adversary Infrastructure Interaction
1  10.138.69.97  Suspicious Web Traffic  Adversary Infrastructure Interaction
2  10.138.69.97  Suspicious Web Traffic  Adversary Infrastructure Interaction
3  10.138.69.97  Suspicious Web Traffic  Adversary Infrastructure Interaction
4  10.138.69.97  Suspicious Web Traffic  Adversary Infrastructure Interaction

     source.meta     source.name                      time detection_types  \
0  AWS_VPC_Flow  prod_webserver 2024-04-25 23:00:00+00:00        waf_rule
1  AWS_VPC_Flow  prod_webserver 2024-04-25 23:00:00+00:00        waf_rule
2  AWS_VPC_Flow  prod_webserver 2024-04-25 23:00:00+00:00        waf_rule
3  AWS_VPC_Flow  prod_webserver 2024-04-25 23:00:00+00:00        waf_rule
4  AWS_VPC_Flow  prod_webserver 2024-04-25 23:00:00+00:00        waf_rule

   session_duration  avg_packet_size
0             600.0        30.986667
1             600.0        81.830000
2             600.0        69.956667
3             600.0        74.706667
4             600.0        34.030000
```

Data Visualization:

```
if df is not None:
    print("\n--- Data Visualization ---")

    # Country-based Interaction Analysis
    if 'src_ip_country_code' in df.columns:
        plt.figure(figsize=(15, 8))
```

```
        sns.countplot(y='src_ip_country_code', data=df, order=df['src_ip_country_code'].value_counts().index)
        plt.title('Interaction Count by Source IP Country Code')
        plt.xlabel('Count')
        plt.ylabel('Source IP Country Code')
        plt.show()
    else:
        print("Column 'src_ip_country_code' not found for plotting.")

    # Suspicious Activities Based on Ports (assuming 'detection_types' and 'dst_port' exist)
    if 'detection_types' in df.columns and 'dst_port' in df.columns:
        suspicious_df = df[df['detection_types'].str.contains('Suspicious', na=False)] # Adjust condition if needed
        if not suspicious_df.empty:
            plt.figure(figsize=(12, 6))
            sns.countplot(x='dst_port', data=suspicious_df, palette='coolwarm', order=suspicious_df['dst_port'].value_counts().index[:10]) #
            plt.title('Suspicious Activities Based on Destination Port (Top 10)')
            plt.xlabel('Destination Port')
            plt.ylabel('Count')
            plt.xticks(rotation=45)
            plt.show()
        else:
            print("No 'Suspicious' entries found in 'detection_types' for plotting port activity.")
    else:
        print("Columns 'detection_types' or 'dst_port' not found for plotting suspicious port activity.")
```

```
--- Data Visualization ---
```




```
No 'Suspicious' entries found in 'detection_types' for plotting port activity.
```

Modeling: Anomaly Detection (Isolation Forest Example):

```
if df is not None and 'bytes_in' in df.columns and 'bytes_out' in df.columns and 'session_duration' in df.columns and 'avg_packet_size' in c
    print("\n--- Modeling: Anomaly Detection (Isolation Forest) ---")
    from sklearn.ensemble import IsolationForest

    # Selecting features for anomaly detection
    features = df[['bytes_in', 'bytes_out', 'session_duration', 'avg_packet_size']].dropna()

    if not features.empty:
        # Initialize the model
```

```
        model = IsolationForest(contamination=0.05, random_state=42) # Adjust contamination as needed

        # Fit and predict anomalies
        anomaly_predictions = model.fit_predict(features)

        # Add anomaly predictions back to the original DataFrame (aligning by index)
        anomaly_df = pd.DataFrame({'anomaly': anomaly_predictions}, index=features.index)
        df = df.merge(anomaly_df, left_index=True, right_index=True, how='left')

        if 'anomaly' in df.columns:
            df['anomaly'] = df['anomaly'].apply(lambda x: 'Suspicious' if x == -1 else 'Normal')
            print("\nAnomaly detection complete. 'anomaly' column added.")
            print("\nAnomaly Value Counts:\n", df['anomaly'].value_counts())
        else:
            print("\nError adding 'anomaly' column to DataFrame.")

    else:
        print("\nNot enough data or missing features for anomaly detection.")
else:
    print("\nDataFrame 'df' is not loaded. Cannot perform anomaly detection.")
```

```
--- Modeling: Anomaly Detection (Isolation Forest) ---

Anomaly detection complete. 'anomaly' column added.

Anomaly Value Counts:
 anomaly
Normal        267
Suspicious     15
Name: count, dtype: int64
```

Evaluation (Basic - Proportion of Anomalies):

```
if df is not None and 'anomaly' in df.columns:
    print("\n--- Evaluation ---")
    print("\nProportion of Anomalies Detected:")
    print(df['anomaly'].value_counts(normalize=True))

    # Display anomaly samples
    suspicious_activities = df[df['anomaly'] == 'Suspicious']
    print("\nFirst 5 Suspicious Activities:")
    print(suspicious_activities.head())
else:
    print("\nAnomaly detection not performed or 'anomaly' column not found for evaluation.")
```

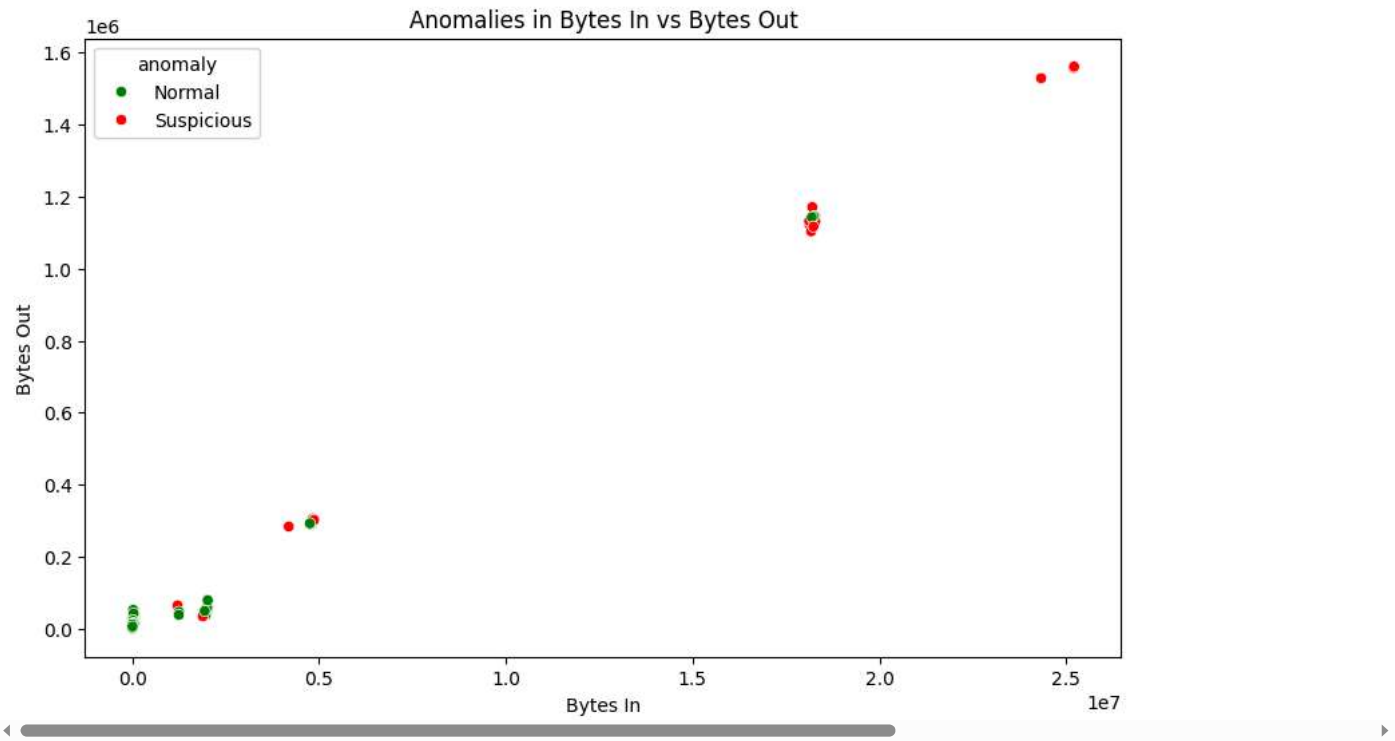
```
--- Evaluation ---

Proportion of Anomalies Detected:
anomaly
Normal        0.946809
Suspicious    0.053191
Name: proportion, dtype: float64

First 5 Suspicious Activities:
     bytes_in  bytes_out             creation_time                  end_time  \
36    4190330     283456 2024-04-25 23:30:00+00:00 2024-04-25 23:40:00+00:00
87    1215594      64362 2024-04-26 00:30:00+00:00 2024-04-26 00:40:00+00:00
116   4827283     306181 2024-04-26 01:00:00+00:00 2024-04-26 01:10:00+00:00
132   1889834      34306 2024-04-26 01:20:00+00:00 2024-04-26 01:30:00+00:00
153   4869181     301752 2024-04-26 01:40:00+00:00 2024-04-26 01:50:00+00:00

              src_ip src_ip_country_code protocol  response.code  dst_port  \
36    155.91.45.242                  US    HTTPS            200       443
87   165.225.240.79                  NL    HTTPS            200       443
116   155.91.45.242                  US    HTTPS            200       443
132  165.225.240.79                  NL    HTTPS            200       443
153   155.91.45.242                  US    HTTPS            200       443

           dst_ip              rule_names  \
36   10.138.69.97  Suspicious Web Traffic
87   10.138.69.97  Suspicious Web Traffic
116  10.138.69.97  Suspicious Web Traffic
132  10.138.69.97  Suspicious Web Traffic
153  10.138.69.97  Suspicious Web Traffic

                         observation_name   source.meta     source.name  \
36   Adversary Infrastructure Interaction  AWS_VPC_Flow  prod_webserver
```

```
87   Adversary Infrastructure Interaction  AWS_VPC_Flow  prod_webserver
116  Adversary Infrastructure Interaction  AWS_VPC_Flow  prod_webserver
132  Adversary Infrastructure Interaction  AWS_VPC_Flow  prod_webserver
153  Adversary Infrastructure Interaction  AWS_VPC_Flow  prod_webserver

                        time detection_types  session_duration  \
36  2024-04-25 23:30:00+00:00        waf_rule             600.0
87  2024-04-26 00:30:00+00:00        waf_rule             600.0
116 2024-04-26 01:00:00+00:00        waf_rule             600.0
132 2024-04-26 01:20:00+00:00        waf_rule             600.0
153 2024-04-26 01:40:00+00:00        waf_rule             600.0

     avg_packet_size     anomaly
36       7456.310000  Suspicious
87       2133.260000  Suspicious
116      8555.773333  Suspicious
132      3206.900000  Suspicious
153      8618.221667  Suspicious
```

Visualization of Anomalies:

```
if df is not None and 'anomaly' in df.columns and 'bytes_in' in df.columns and 'bytes_out' in df.columns:
    print("\n--- Visualization of Anomalies ---")
    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.xlabel('Bytes In')
    plt.ylabel('Bytes Out')
    plt.show()
else:
    print("\nAnomaly detection not performed or necessary columns not found for anomaly visualization.")
```

```
--- Visualization of Anomalies ---
```

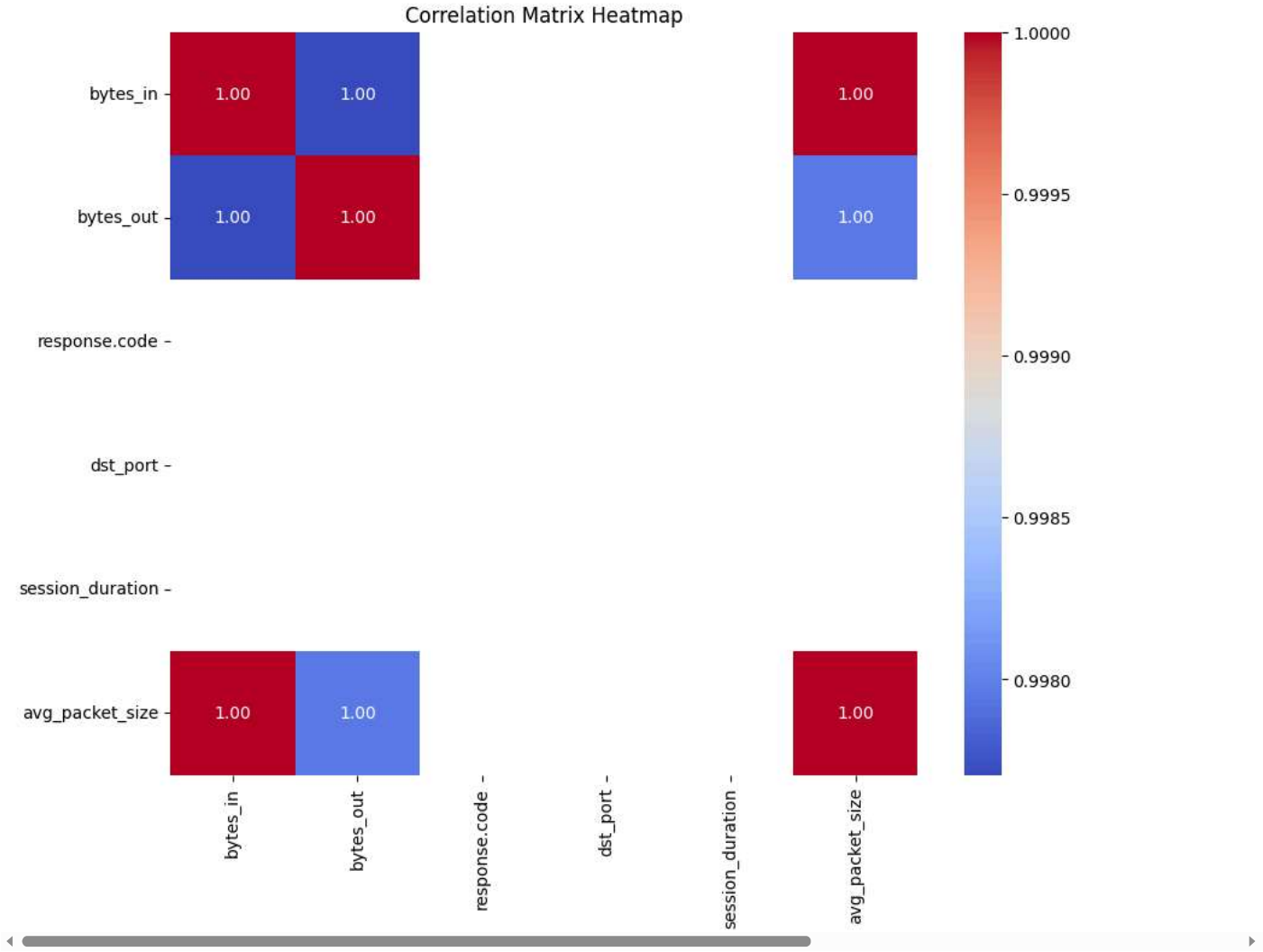


The correlation matrix

```
if df is not None:
    print("\n--- Correlation Analysis and Heatmap ---")
    numeric_df = df.select_dtypes(include=['float64', 'int64'])
    correlation_matrix_numeric = numeric_df.corr()

    if not correlation_matrix_numeric.empty:
        import matplotlib.pyplot as plt
        import seaborn as sns

        plt.figure(figsize=(10, 8))
```

```
        sns.heatmap(correlation_matrix_numeric, annot=True, fmt=".2f", cmap='coolwarm')
        plt.title('Correlation Matrix Heatmap')
        plt.show()
    else:
        print("No numeric columns found to calculate correlation matrix.")
else:
    print("DataFrame 'df' is not loaded. Cannot perform correlation analysis.")
```

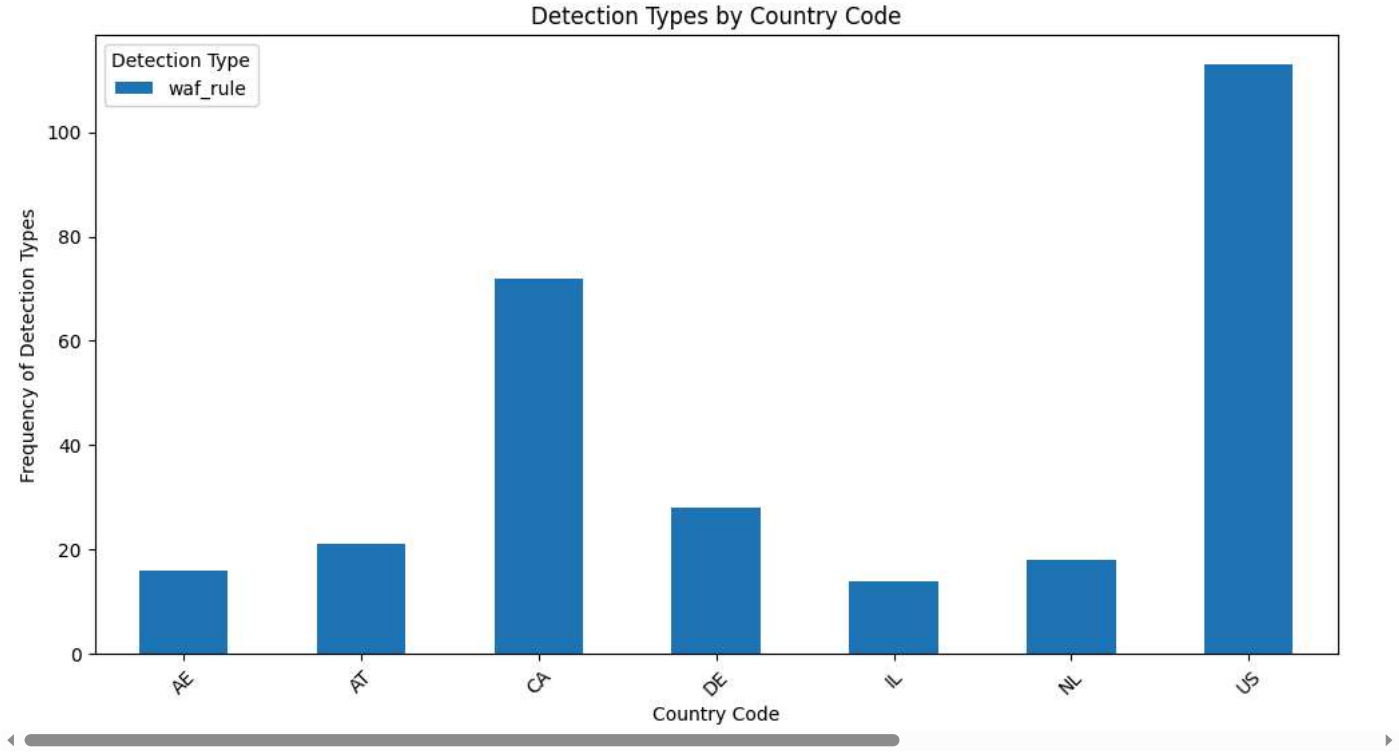
```
--- Correlation Analysis and Heatmap ---
```



Stacked Bar Chart for Detection Types by Country

```
if df is not None and 'src_ip_country_code' in df.columns and 'detection_types' in df.columns:
    print("\n--- Stacked Bar Chart for Detection Types by Country ---")

    # Prepare data for stacked bar chart
    detection_types_by_country = pd.crosstab(df['src_ip_country_code'], df['detection_types'])

    if not detection_types_by_country.empty:
        import matplotlib.pyplot as plt

        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()
    else:
        print("Not enough data to create the stacked bar chart.")
else:
    print("DataFrame 'df' is not loaded or necessary columns ('src_ip_country_code', 'detection_types') not found.")
```

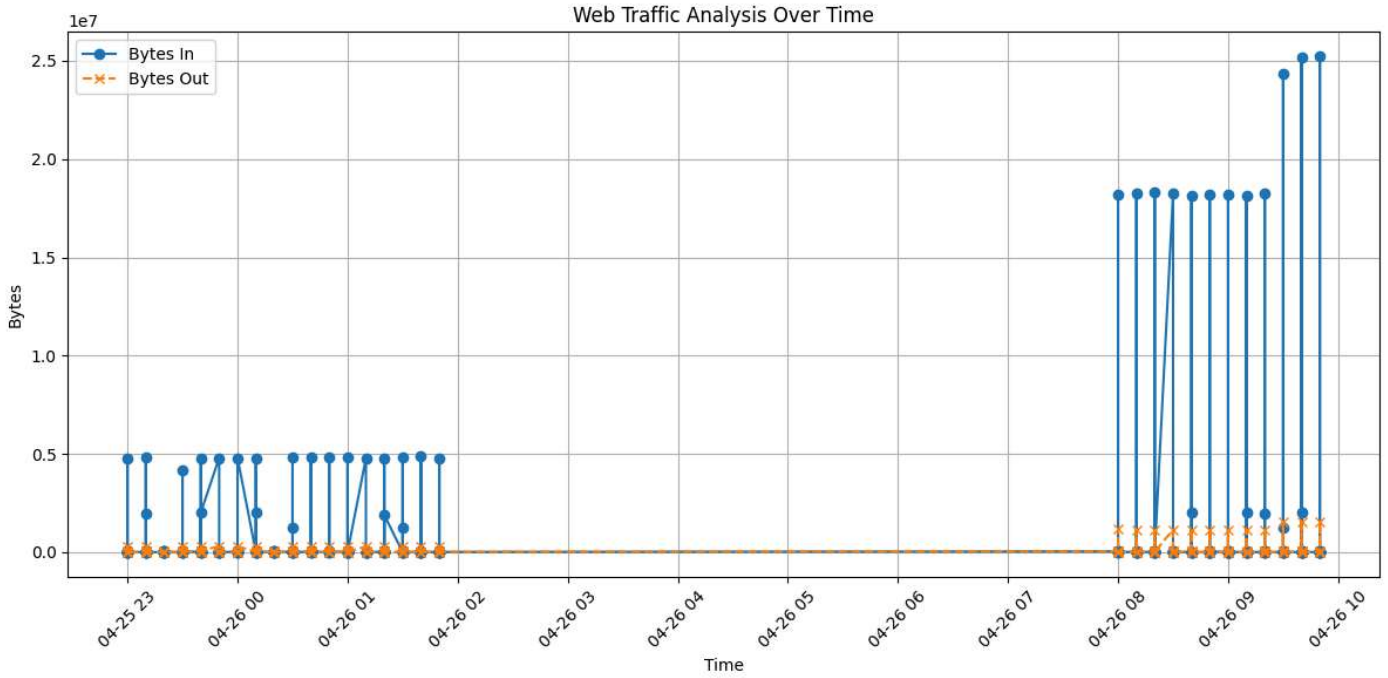
```
--- Stacked Bar Chart for Detection Types by Country ---
```



Convert 'creation_time' to datetime format

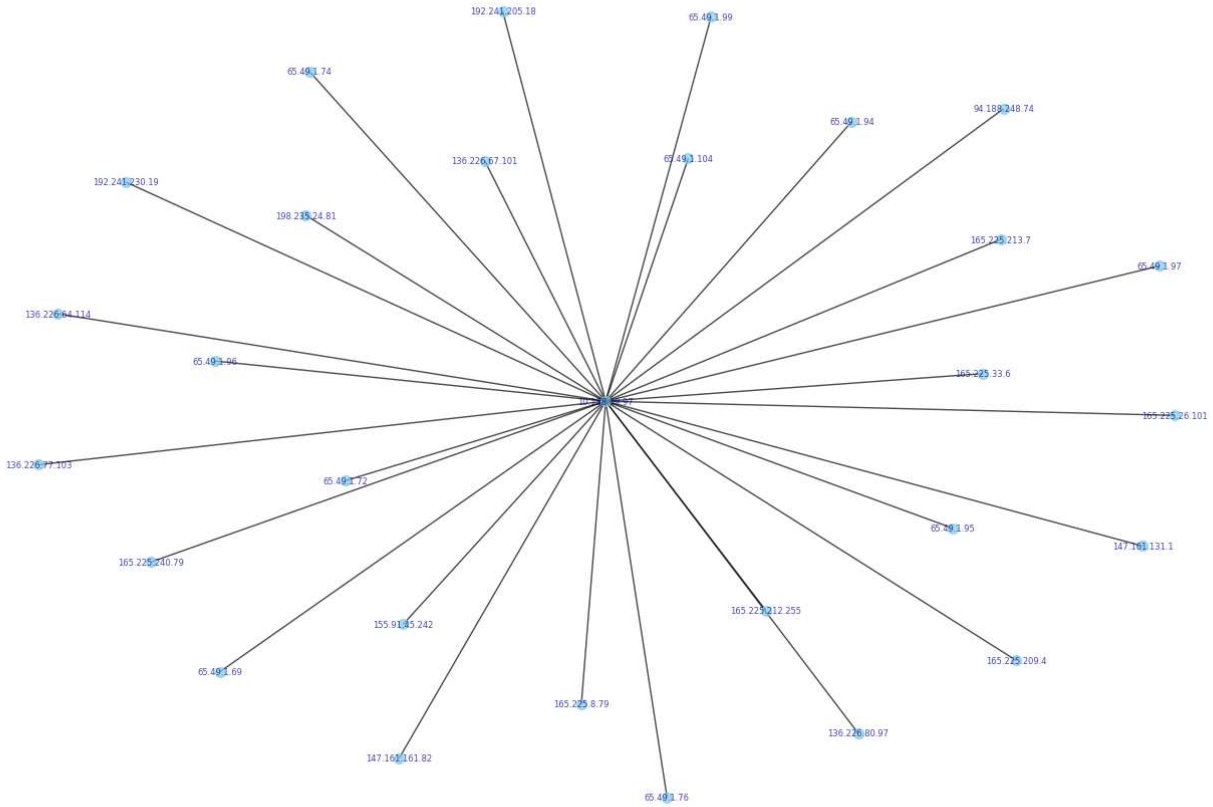
```
if df is not None and 'creation_time' in df.columns and 'bytes_in' in df.columns and 'bytes_out' in df.columns:
    print("\n--- Web Traffic Analysis Over Time ---")

    # Create a copy to avoid modifying the original DataFrame's index
    time_series_df = df.copy()

    # Convert 'creation_time' to datetime format
    time_series_df['creation_time'] = pd.to_datetime(time_series_df['creation_time'], errors='coerce')

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

    # Sort the index for proper time series plotting
    time_series_df.sort_index(inplace=True)

    import matplotlib.pyplot as plt

    plt.figure(figsize=(12, 6))
    plt.plot(time_series_df.index, time_series_df['bytes_in'], label='Bytes In', marker='o', linestyle='-')
    plt.plot(time_series_df.index, time_series_df['bytes_out'], label='Bytes Out', marker='x', linestyle='--')
    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()
    plt.show()

else:
    print("DataFrame 'df' is not loaded or necessary columns ('creation_time', 'bytes_in', 'bytes_out') not found.")
```

```
--- Web Traffic Analysis Over Time ---
```




```
if df is not None and 'src_ip' in df.columns and 'dst_ip' in df.columns:
    print("\n--- Network Interaction Graph ---")
    import networkx as nx
    import matplotlib.pyplot as plt

    # Create a graph
    G = nx.Graph()

    # Add edges from source IP to destination IP
    for idx, row in df.iterrows():
        if pd.notna(row['src_ip']) and pd.notna(row['dst_ip']):
            G.add_edge(row['src_ip'], row['dst_ip'])

    # Draw the network graph
    plt.figure(figsize=(14, 10))
    pos = nx.spring_layout(G, k=0.15, iterations=20)  # Adjust layout parameters as needed
    nx.draw_networkx(G, pos, with_labels=True, node_size=50, font_size=6,
                     node_color='skyblue', font_color='darkblue', alpha=0.7)
    plt.title('Network Interaction between Source and Destination IPs')
    plt.axis('off')  # Turn off the axis
    plt.tight_layout()
    plt.show()

    print(f"\nNumber of nodes (unique IPs): {G.number_of_nodes()}")
    print(f"Number of edges (interactions): {G.number_of_edges()}")

else:
    print("DataFrame 'df' is not loaded or necessary columns ('src_ip', 'dst_ip') not found.")
```

```
--- Network Interaction Graph ---
```




```
Number of nodes (unique IPs): 29
Number of edges (interactions): 28
```

RandomForestClassifier

```
if df is not None and 'detection_types' in df.columns and 'bytes_in' in df.columns and 'bytes_out' in df.columns and 'session_duration' in df
    print("\n--- Random Forest Classifier ---")
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import classification_report, accuracy_score
    from sklearn.preprocessing import StandardScaler

    # Create a copy of the DataFrame for modeling to avoid modifying the original
    model_df = df.copy()

    # Encode 'detection_types' into binary labels (assuming 'waf_rule' indicates suspicious)
    model_df['is_suspicious'] = (model_df['detection_types'].str.contains('waf_rule', na=False)).astype(int)

    # Select features (using original and engineered features)
    features = model_df[['bytes_in', 'bytes_out', 'session_duration']].dropna()
    labels = model_df.loc[features.index, 'is_suspicious']

    if not features.empty:
        # Scale numerical features
        scaler = StandardScaler()
        scaled_features = scaler.fit_transform(features)
```

```
    # Split
```

```
--- Random Forest Classifier ---
```

Neural Network

```
if df is not None and 'detection_types' in df.columns and 'bytes_in' in df.columns and 'bytes_out' in df.columns:
    print("\n--- Neural Network Model ---")
    import numpy as np
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense

    # First, encode 'detection_types' into binary labels
    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_state=42)

    # 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
    print("\nTraining the Neural Network...")
    history = model.fit(X_train_scaled, y_train, epochs=10, batch_size=8, verbose=1)

    # Evaluate the model
    print("\nEvaluating the Neural Network...")
    loss, accuracy = model.evaluate(X_test_scaled, y_test, verbose=0)
    print(f"Test Accuracy: {accuracy * 100:.2f}%")

else:
    print("DataFrame 'df' is not loaded or necessary columns ('detection_types', 'bytes_in', 'bytes_out') not found.")
```

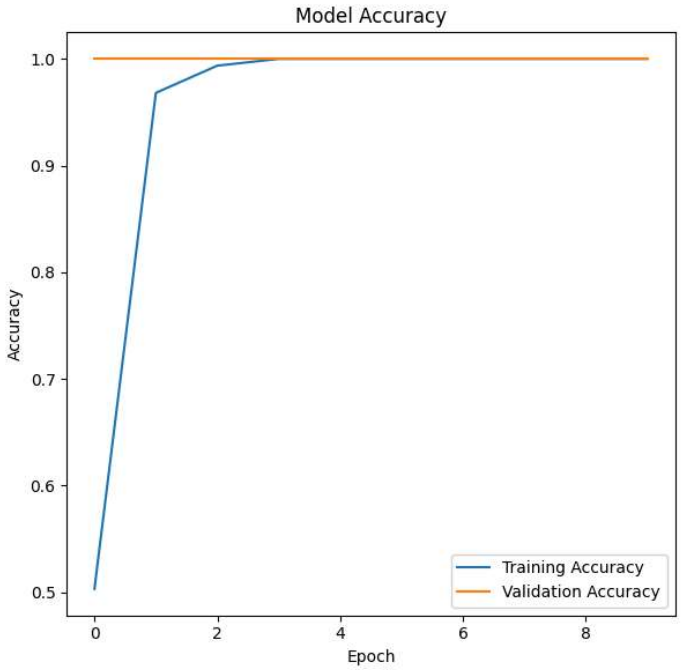
```
--- Neural Network Model ---
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Training the Neural Network...
Epoch 1/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 1.0000 - loss: 0.6479
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.5849
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.4911
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.3791
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.2781
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.1777
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.1197
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0737
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0506
```

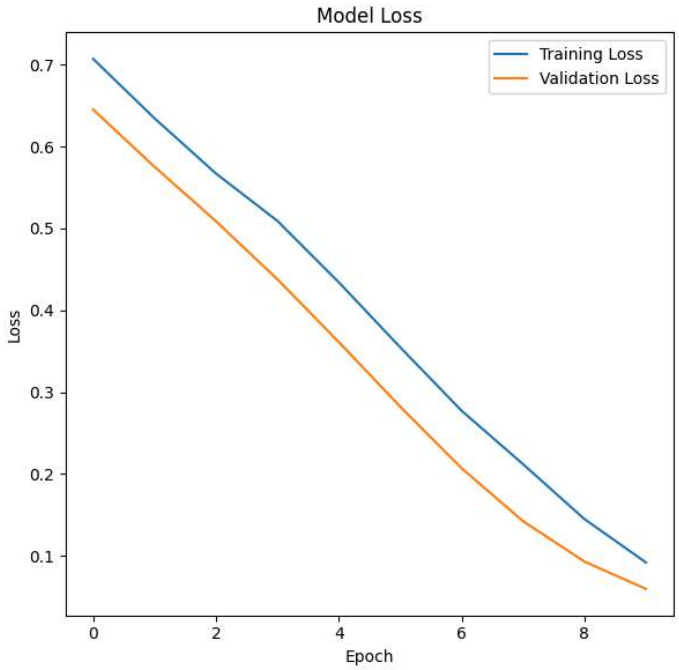
```
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0369

Evaluating the Neural Network...
Test Accuracy: 100.00%
```

```
if df is not None and 'detection_types' in df.columns and 'bytes_in' in df.columns and 'bytes_out' in df.columns:
    print("\n--- Neural Network Model with Dropout and Validation ---")
    import numpy as np
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Dropout
    import matplotlib.pyplot as plt

    # First, encode 'detection_types' into binary labels
    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_state=42)

    # 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(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
        Dropout(0.5),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

    # Compile the model
    model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])

    # Train the model with validation split
    print("\nTraining the Neural Network with Validation...")
    history = model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, verbose=1, validation_split=0.2)

    # Evaluate the model on the test set
    print("\nEvaluating the Neural Network on Test Set...")
    loss, accuracy = model.evaluate(X_test_scaled, y_test, verbose=0)
    print(f"Test Accuracy: {accuracy * 100:.2f}%")

    # Plotting the training history
    print("\nPlotting Training History...")
    plt.figure(figsize=(12, 6))

    # Plot accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

```
else:
    print("DataFrame 'df' is not loaded or necessary columns ('detection_types', 'bytes_in', 'bytes_out') not found.")
```

```
--- Neural Network Model with Dropout and Validation ---

Training the Neural Network with Validation...
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argum
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 73ms/step - accuracy: 0.3374 - loss: 0.7244 - val_accuracy: 1.0000 - val_loss: 0.6451
Epoch 2/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9429 - loss: 0.6459 - val_accuracy: 1.0000 - val_loss: 0.5752
Epoch 3/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.9922 - loss: 0.5758 - val_accuracy: 1.0000 - val_loss: 0.5087
Epoch 4/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 1.0000 - loss: 0.5188 - val_accuracy: 1.0000 - val_loss: 0.4377
Epoch 5/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 1.0000 - loss: 0.4464 - val_accuracy: 1.0000 - val_loss: 0.3611
Epoch 6/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 1.0000 - loss: 0.3739 - val_accuracy: 1.0000 - val_loss: 0.2822
Epoch 7/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 1.0000 - loss: 0.2939 - val_accuracy: 1.0000 - val_loss: 0.2069
Epoch 8/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 1.0000 - loss: 0.2209 - val_accuracy: 1.0000 - val_loss: 0.1423
Epoch 9/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 1.0000 - loss: 0.1553 - val_accuracy: 1.0000 - val_loss: 0.0929
Epoch 10/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 1.0000 - loss: 0.0976 - val_accuracy: 1.0000 - val_loss: 0.0596

Evaluating the Neural Network on Test Set...
Test Accuracy: 100.00%

Plotting Training History...
```






```
if df is not None and 'detection_types' in df.columns and 'bytes_in' in df.columns and 'bytes_out' in df.columns:
    print("\n--- 1D Convolutional Neural Network (CNN) Model ---")
    import numpy as np
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv1D, Flatten, Dense, Dropout
    import matplotlib.pyplot as plt

    # First, encode 'detection_types' into binary labels
```

```
        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_state=42)

        # Normalize features and reshape for Conv1D
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train).reshape(X_train.shape[0], X_train.shape[1], 1)
        X_test_scaled = scaler.transform(X_test).reshape(X_test.shape[0], X_test.shape[1], 1)

        # Adjusting the network to accommodate the input size
        model = Sequential([
            Conv1D(32, kernel_size=1, activation='relu', input_shape=(X_train_scaled.shape[1], 1)),
            Flatten(),
            Dense(64, activation='relu'),
            Dropout(0.5),
            Dense(1, activation='sigmoid')
        ])

        # Compile the model
        model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])

        # Train the model
        print("\nTraining the 1D CNN Model...")
        history = model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, verbose=1, validation_split=0.2)

        # Evaluate the model
        print("\nEvaluating the 1D CNN Model on Test Set...")
        loss, accuracy = model.evaluate(X_test_scaled, y_test, verbose=0)
        print(f"Test Accuracy: {accuracy * 100:.2f}%")

        # Plotting the training history
        print("\nPlotting Training History...")
        plt.figure(figsize=(12, 6))

        # Plot accuracy
        plt.subplot(1, 2, 1)
        plt.plot(history.history['accuracy'], label='Training Accuracy')
        plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
        plt.title('Model Accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.legend()

        # Plot loss
        plt.subplot(1, 2, 2)
        plt.plot(history.history['loss'], label='Training Loss')
        plt.plot(history.history['val_loss'], label='Validation Loss')
        plt.title('Model Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()

        plt.tight_layout()
        plt.show()

    else:
        print("DataFrame 'df' is not loaded or necessary columns ('detection_types', 'bytes_in', 'bytes_out') not found.")
```