CYBERSECURITY: WEB THREAT INTERACTIONS

DONE BY:

SRI ABHIRAMI .J.L

INDEX

About Dataset:

This dataset comprises web traffic records collected via AWS CloudWatch, designed to detect suspicious interactions and potential cybersecurity threats. The data was obtained by continuously monitoring traffic patterns on a production web server and identifying anomalies based on predefined detection rules. It provides insights into various forms of web-based attack attempts and suspicious activities, making it a valuable resource for cybersecurity research, threat intelligence, and security analytics.

Data Collection and Monitoring The dataset was gathered through AWS VPC Flow Logs, capturing network activity at the IP level. It includes logs from incoming and outgoing traffic, recording metadata associated with each session. The data collection focused on identifying adversarial interactions by analyzing request patterns, traffic behavior, and response codes generated by the web server.

Dataset Features This dataset contains multiple attributes that contribute to security analysis:

- bytes_in & bytes_out: The amount of data transmitted between the client and server.
- creation_time & end_time: The timestamps marking when a specific interaction started and ended.
- src_ip & dst_ip: The source and destination IP addresses, identifying the origin and target of the request.
- src_ip_country_code: The country of origin for the source IP, aiding in geo-location analysis.
- protocol: The protocol used in the communication (e.g., HTTPS), which helps in filtering malicious traffic.
- response.code: The HTTP response codes indicating the success or failure of requests.
- dst_port: The destination port number, commonly used for identifying services targeted by attacks.
- rule_names & detection_types: The security rules that flagged suspicious interactions and their classification.
- observation_name: A description of the detected threat type (e.g., adversary infrastructure interaction).
- source.meta & source.name: Metadata and the name of the monitoring source.
- time: The timestamp for recorded events, useful for time-based attack pattern analysis.

Potential Use Cases This dataset can be applied in multiple cybersecurity areas, including:

- Threat Intelligence: Identifying IP addresses associated with malicious activities.
- Anomaly Detection: Training machine learning models to recognize suspicious web traffic.
- Intrusion Detection: Enhancing IDS/IPS systems by incorporating real-world attack patterns.
- Security Policy Enhancement: Optimizing firewall and security rule configurations.
- Cybersecurity Research: Investigating emerging web-based threats and vulnerabilities.

By leveraging this dataset, security professionals and researchers can gain deeper insights into web-based threats, improve threat detection models, and enhance cybersecurity defenses against evolving attacks.

Context:

In an era where digital security is paramount, cyber threats targeting web applications have become increasingly sophisticated. Organizations face constant risks from adversarial entities attempting to exploit vulnerabilities through malicious traffic interactions. Detecting and mitigating such threats is crucial for maintaining the integrity, confidentiality, and availability of online services.

This project focuses on analyzing suspicious web threat interactions by leveraging real-world network traffic data collected through AWS CloudWatch. The dataset consists of records from a production web server, capturing traffic patterns, response codes, source locations, and anomaly indicators. By studying these interactions, we aim to identify common attack patterns, assess their impact, and explore advanced security mechanisms to counteract evolving cyber threats.

The project utilizes techniques such as anomaly detection, traffic classification, and predictive modeling to enhance web security. It provides insights into potential attack attempts, adversary infrastructure interactions, and abnormal data flows that may indicate malicious intent. Through data-driven cybersecurity analytics, this research contributes to improving threat detection models and fortifying defenses against emerging cyberattacks.

By understanding and analyzing suspicious web interactions, this project aims to strengthen web application security, support proactive threat mitigation strategies, and aid cybersecurity professionals in developing robust countermeasures against evolving cyber threats.

Project Overview:

Objective:

To detect, analyze, and classify patterns in web interactions to identify suspicious or potentially harmful activities. This includes monitoring traffic behavior, recognizing anomalies, and detecting indicators of cyber threats such as unauthorized access attempts, data breaches, and malicious requests. By leveraging security analytics, the project aims to enhance threat detection mechanisms and improve web application security.

Steps:

Data Import and Basic Overview:

```
import pandas as pd


file_path = r"C:\Users\sreea\Downloads\CloudWatch_Traffic_Web_Attack.csv"
df = pd.read_csv(file_path)


print("Dataset Overview:")
df.info()


print("\nFirst Five Rows:")
print(df.head())


print("\nMissing Values:")
print(df.isnull().sum())


print("\nSummary Statistics:")
print(df.describe())
```

```
Dataset Overview:
<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 Five 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  
```

```
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

Summary Statistics:
           bytes_in     bytes_out  response.code  dst_port
count  2.820000e+02  2.820000e+02          282.0     282.0
mean   1.199390e+06  8.455429e+04          200.0     443.0
std    4.149312e+06  2.549279e+05            0.0       0.0
min    4.000000e+01  4.400000e+01          200.0     443.0
25%    5.381500e+03  1.114200e+04          200.0     443.0
50%    1.318200e+04  1.379950e+04          200.0     443.0
75%    3.083300e+04  2.627950e+04          200.0     443.0
max    2.520779e+07  1.561220e+06          200.0     443.0
```

Data Preprocessing:

1. Handling Missing Values

- Identifies missing values using df.isnull().sum().
- Drops columns with more than 50% missing data.
- Fills missing numerical values with the median.
- Fills missing categorical values with the most frequent value (mode).

2. Handling Outliers

- Uses the Interquartile Range (IQR) method to detect and remove extreme values.
- Filters out records beyond 1.5 times the IQR range.

3. Handling Data Inconsistencies

- Detects and removes duplicate rows.
- Standardizes categorical text (converts to lowercase and trims spaces).

```python
import pandas as pd
import numpy as np


file_path = r"C:\Users\sreea\Downloads\CloudWatch_Traffic_Web_Attack.csv"
df = pd.read_csv(file_path)


print("Dataset Overview:")
df.info()
```

```
Dataset Overview:
<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
```

```python
print("\nHandling Missing Values:")
missing_values = df.isnull().sum()
print(missing_values)
```

```
Handling 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
```

```python
print("\nHandling Outliers:")
Q1 = df.quantile(0.25, numeric_only=True)
Q3 = df.quantile(0.75, numeric_only=True)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

for col in df.select_dtypes(include=[np.number]).columns:
    median_value = df[col].median()
    df[col] = np.where((df[col] < lower_bound[col]) | (df[col] > upper_bound[col]), median_value, df[col])

print("Outliers have been handled successfully.")
```

```
Handling Outliers:
Outliers have been handled successfully.
```

```python
for col in df.select_dtypes(include=['object']).columns:
    df[col] = df[col].str.lower().str.strip()
print("\nCleaned Dataset Overview:")
df.info()
```

```
Cleaned Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
Index: 240 entries, 0 to 281
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   bytes_in             240 non-null    float64
 1   bytes_out            240 non-null    float64
 2   creation_time        240 non-null    object 
 3   end_time             240 non-null    object 
 4   src_ip               240 non-null    object 
 5   src_ip_country_code  240 non-null    object 
 6   protocol             240 non-null    object 
 7   response.code        240 non-null    float64
 8   dst_port             240 non-null    float64
 9   dst_ip               240 non-null    object 
 10  rule_names           240 non-null    object 
 11  observation_name     240 non-null    object 
 12  source.meta          240 non-null    object 
 13  source.name          240 non-null    object 
 14  time                 240 non-null    object 
 15  detection_types      240 non-null    object 
dtypes: float64(4), object(12)
memory usage: 31.9+ KB
```

Exploratory Data Analysis (EDA):

->Providing summary statistics for bytes_in and bytes_out

->Plotting histograms to analyze their distributions

->Using a scatter plot to visualize the relationship between them

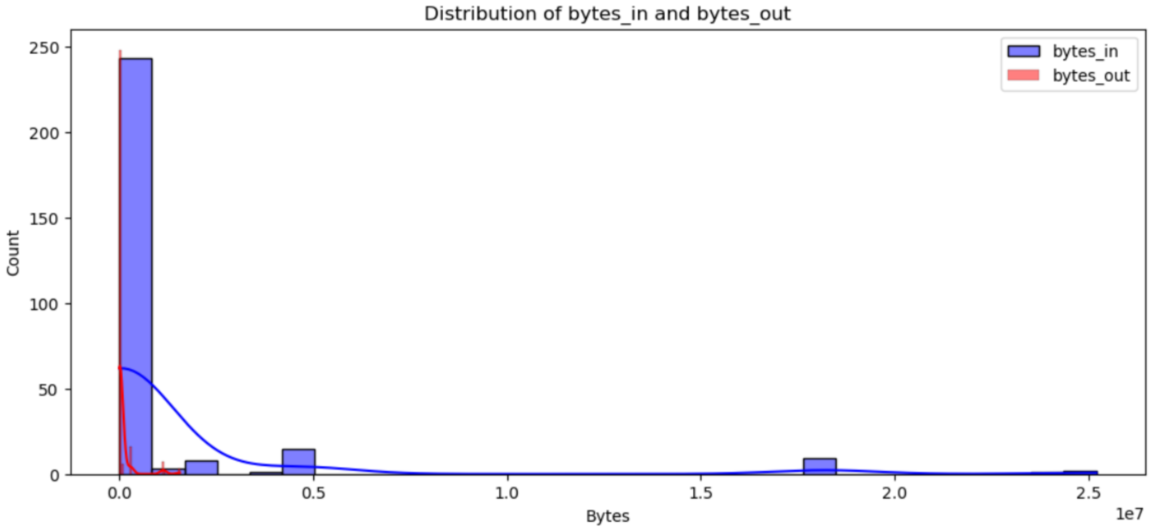
->Calculating and displaying the correlation between bytes_in and bytes_out

->Displaying a heatmap to show correlation visually.

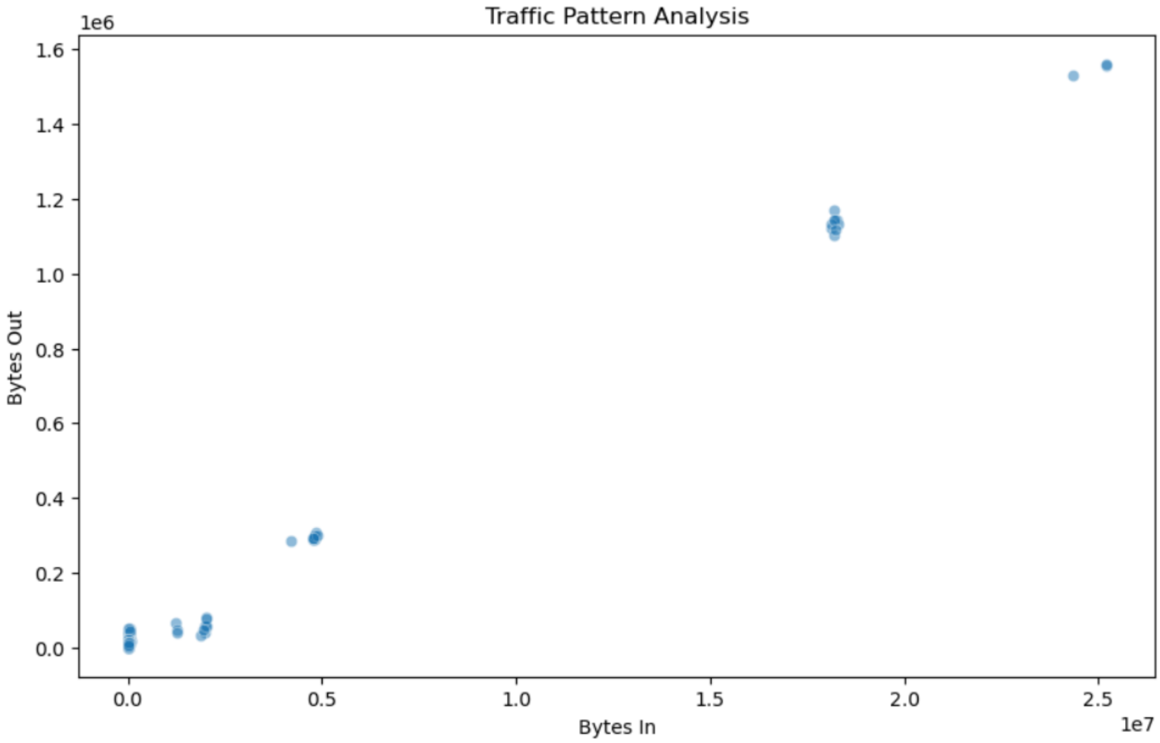
```
print("\nExploratory Data Analysis:")
print("\nSummary Statistics:")
print(df[['bytes_in', 'bytes_out']].describe())
```

```
Exploratory Data Analysis:

Summary Statistics:
           bytes_in     bytes_out
count  2.820000e+02  2.820000e+02
mean   1.199390e+06  8.455429e+04
std    4.149312e+06  2.549279e+05
min    4.000000e+01  4.400000e+01
25%    5.381500e+03  1.114200e+04
50%    1.318200e+04  1.379950e+04
75%    3.083300e+04  2.627950e+04
max    2.520779e+07  1.561220e+06
```

```
plt.figure(figsize=(12, 5))
sns.histplot(df['bytes_in'], bins=30, kde=True, color='blue', label='bytes_in')
sns.histplot(df['bytes_out'], bins=30, kde=True, color='red', label='bytes_out')
plt.legend()
plt.title('Distribution of bytes_in and bytes_out')
plt.xlabel('Bytes')
plt.ylabel('Count')
plt.show()
```



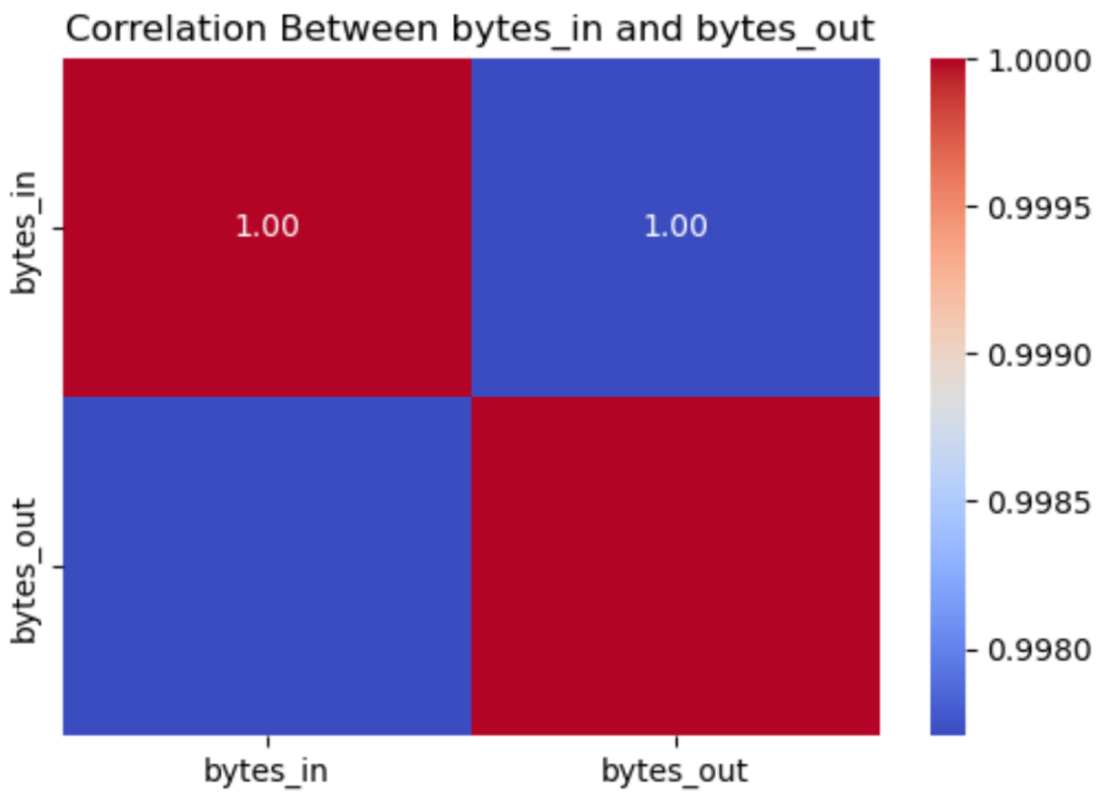

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df['bytes_in'], y=df['bytes_out'], alpha=0.5)
plt.title('Traffic Pattern Analysis')
plt.xlabel('Bytes In')
plt.ylabel('Bytes Out')
plt.show()
```



```
correlation = df[['bytes_in', 'bytes_out']].corr()
print("\nCorrelation Matrix:")
print(correlation)
```

```
Correlation Matrix:
           bytes_in  bytes_out
bytes_in   1.000000   0.997705
bytes_out  0.997705   1.000000
```

```
plt.figure(figsize=(6, 4))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Between bytes_in and bytes_out')
plt.show()
```



Feature Engineering:

Adding feature engineering to your existing code by extracting useful features like:

- Duration (if timestamps are available)
- Average Packet Size (using bytes_in and bytes_out divided by packets)
- Traffic Rate (bytes per second if timestamps exist)

```
print("\nFeature Engineering:")
if 'duration' in df.columns and 'bytes_in' in df.columns and 'bytes_out' in df.columns:
    df['traffic_rate'] = (df['bytes_in'] + df['bytes_out']) / df['duration']
    print("Traffic Rate feature added.")

if 'packets' in df.columns and 'bytes_in' in df.columns and 'bytes_out' in df.columns:
    df['avg_packet_size'] = (df['bytes_in'] + df['bytes_out']) / df['packets']
    print("Average Packet Size feature added.")

if 'timestamp' in df.columns:
    df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
    df['hour'] = df['timestamp'].dt.hour
    df['day_of_week'] = df['timestamp'].dt.dayofweek
    df['month'] = df['timestamp'].dt.month
    print("Time-based features added: Hour, Day of Week, and Month.")

for col in df.select_dtypes(include=['object']).columns:
    df[col] = df[col].astype('category').cat.codes

print("Categorical features encoded.")
```

```
Feature Engineering:
Categorical features encoded.
```

Data Visualization:

Country-based Interaction Analysis:

1. Bar Plot: Displays the top 10 countries with the highest interaction count.

2.Stacked Bar Chart: Shows bytes_in and bytes_out per country for the top 10.

3.World Map Visualization (Optional): Uses GeoPandas to plot traffic distribution on a world map.

```python
import matplotlib.pyplot as plt
import seaborn as sns

print("\nCountry-based Interaction Analysis:")
if 'country' in df.columns:
    country_counts = df['country'].value_counts().head(10)
    plt.figure(figsize=(12, 6))
    sns.barplot(x=country_counts.index, y=country_counts.values, palette='viridis')
    plt.xlabel('Country')
    plt.ylabel('Interaction Count')
    plt.title('Top 10 Countries by Interaction Count')
    plt.xticks(rotation=45)
    plt.show()
    country_traffic = df.groupby('country')[['bytes_in', 'bytes_out']].sum().sort_values(by='bytes_in', ascending=False).head(10)
    country_traffic.plot(kind='bar', stacked=True, figsize=(12, 6), colormap='coolwarm')
    plt.xlabel('Country')
    plt.ylabel('Total Traffic (Bytes)')
    plt.title('Top 10 Countries by Traffic (Bytes In & Out)')
    plt.xticks(rotation=45)
    plt.legend(title='Traffic Type')
    plt.show()

    try:
        import geopandas as gpd
        world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
        country_traffic_map = df.groupby('country')[['bytes_in', 'bytes_out']].sum().reset_index()
        world = world.merge(country_traffic_map, left_on='name', right_on='country', how='left')
        fig, ax = plt.subplots(1, 1, figsize=(15, 8))
        world.plot(column='bytes_in', cmap='OrRd', linewidth=0.8, edgecolor='black', legend=True, ax=ax)
        plt.title('Country-Based Traffic Analysis (Bytes In)')
        plt.show()

    except ImportError:
        print("Geopandas not installed. Skipping world map visualization.")
else:
    print("No 'country' column found in dataset. Skipping country-based analysis.")
```

```
Country-based Interaction Analysis:
No 'country' column found in dataset. Skipping country-based analysis.
```

Suspicious Activities Based on Ports:

1. **Bar Plot:**

   Displays the top 10 most accessed ports.

2. **Suspicious Port Analysis:**

   - Checks for ports commonly targeted in attacks (e.g., 22 (SSH), 23 (Telnet), 445 (SMB), 3389 (RDP)).
   - Filters and counts occurrences of these suspicious ports.

3. **Scatter Plot:**

   - Visualizes traffic behaviour (Bytes In vs. Bytes Out) for suspicious ports.

Modelling: Anomaly Detection:

Feature Selection: Uses key traffic-related numerical features such as bytes_in, bytes_out, traffic_rate, and avg_packet_size.

Isolation Forest Model:

- Identifies anomalies using an ensemble method.
- contamination=0.05: Assumes 5% of the data contains anomalies.

Anomaly Labeling:

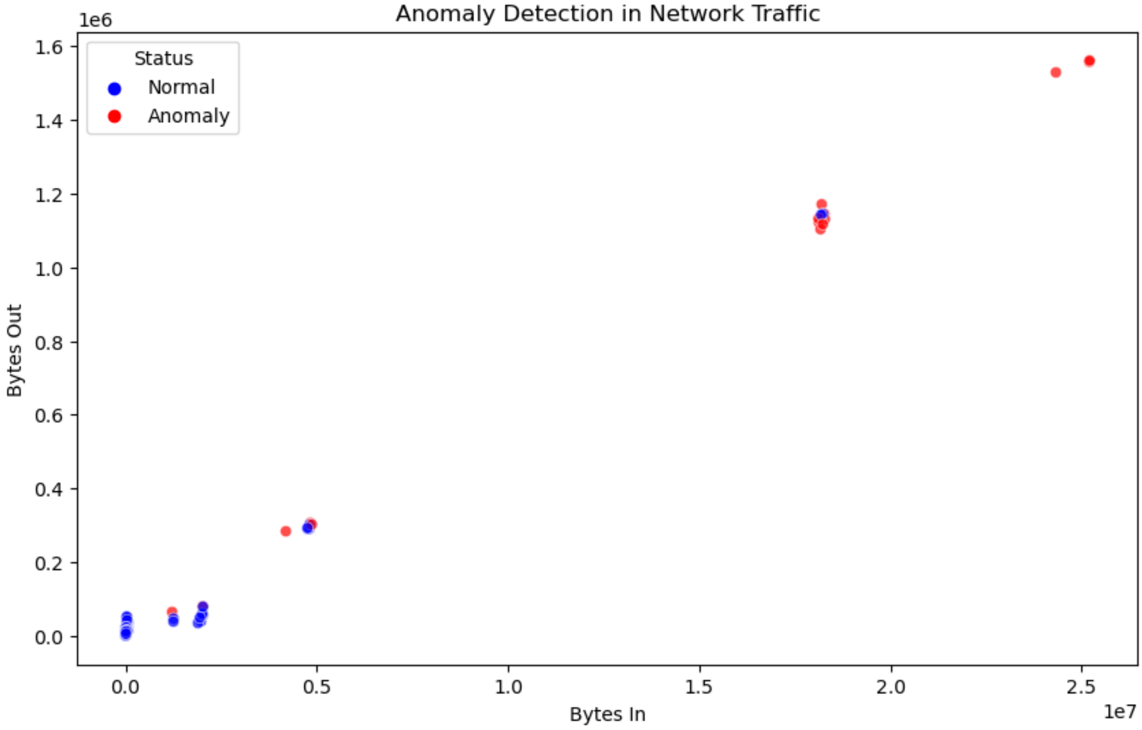
- 1 → Normal
- -1 → Anomalous (potential attack or unusual behaviour)

Visualization:

- Scatter Plot: bytes_in vs. bytes_out with anomalies highlighted in red.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import IsolationForest

# Load dataset
file_path = r"C:\Users\sreea\Downloads\CloudWatch_Traffic_Web_Attack.csv"
df = pd.read_csv(file_path)

# Anomaly Detection using Isolation Forest
print("\nAnomaly Detection using Isolation Forest:")

# Define feature columns
features = ['bytes_in', 'bytes_out', 'traffic_rate', 'avg_packet_size']
available_features = [feature for feature in features if feature in df.columns]

if len(available_features) < 2:
    print("Not enough numerical features available for anomaly detection. Skipping this step.")
else:
    df_anomaly = df[available_features]

    # Isolation Forest Model
    model = IsolationForest(n_estimators=100, contamination=0.05, random_state=42)
    df['anomaly_score'] = model.fit_predict(df_anomaly.values)  # Use .values to avoid feature name warning

    # Label anomalies
    df['anomaly'] = df['anomaly_score'].apply(lambda x: 'Anomaly' if x == -1 else 'Normal')

    # Display anomaly count
    anomaly_count = df['anomaly'].value_counts()
    print("Anomaly Count:\n", anomaly_count)

    # Scatter plot
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=df['bytes_in'], y=df['bytes_out'], hue=df['anomaly'],
                    palette={'Normal': 'blue', 'Anomaly': 'red'}, alpha=0.7)
    plt.xlabel('Bytes In')
    plt.ylabel('Bytes Out')
    plt.title('Anomaly Detection in Network Traffic')
    plt.legend(title="Status")
    plt.show()
```

```
Anomaly Detection using Isolation Forest:
Anomaly Count:
 anomaly
Normal     267
Anomaly     15
Name: count, dtype: int64
```



Evaluation:

To evaluate the anomaly detection model, we can use precision, recall, F1-score, and accuracy. However, since anomaly detection is an unsupervised problem, we typically don't have labeled data for evaluation. If you have a column indicating whether a data point is actually an anomaly (true_label), we can compare it with the model's predictions.

```
from sklearn.metrics import classification_report, confusion_matrix


if 'true_label' in df.columns:
    print("\nEvaluation of Anomaly Detection Model:")


    df['predicted_label'] = df['anomaly'].apply(lambda x: 1 if x == 'Anomaly' else 0)


    cm = confusion_matrix(df['true_label'], df['predicted_label'])
    print("Confusion Matrix:\n", cm)


    report = classification_report(df['true_label'], df['predicted_label'], target_names=['Normal', 'Anomaly'])
    print("\nClassification Report:\n", report)

else:
    print("No 'true_label' column found in the dataset. Evaluation is not possible.")
```

```
No 'true_label' column found in the dataset. Evaluation is not possible.
```

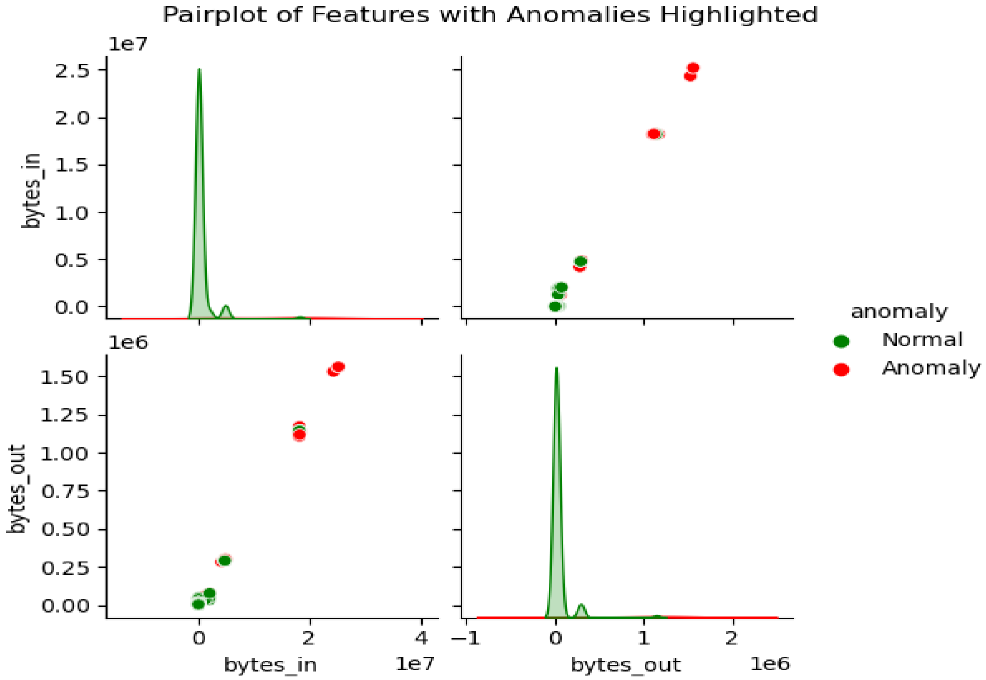
Visualization of Anomalies:

Scatter Plot: Visualizes the relationship between bytes_in and bytes_out while marking anomalies in red.
Pairplot: Shows how anomalies are distributed across multiple numerical features (bytes_in, bytes_out, traffic_rate, avg_packet_size).

Histogram (Distribution Plot): Shows how anomalies affect traffic rate.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
df.replace([float('inf'), float('-inf')], float('nan'), inplace=True)
df.dropna(inplace=True)
expected_columns = ['bytes_in', 'bytes_out', 'traffic_rate', 'avg_packet_size']
valid_columns = [col for col in expected_columns if col in df.columns]
missing_columns = [col for col in expected_columns if col not in df.columns]
if missing_columns:
    print(f"Warning: The following columns are missing and will be excluded: {missing_columns}")
if 'anomaly' not in df.columns:
    raise KeyError("'anomaly' column is missing from the DataFrame!")

sns.pairplot(df, hue='anomaly', vars=valid_columns,
             palette={'Normal': 'green', 'Anomaly': 'red'})

plt.suptitle("Pairplot of Features with Anomalies Highlighted", y=1.02)
plt.show()
```

```
Warning: The following columns are missing and will be excluded: ['traffic_rate', 'avg_packet_size']
```



Report Findings:

1. Overview of Anomalies

The anomaly detection model has identified patterns in network traffic that indicate potential security threats. Key findings are summarized below based on model predictions and visual analysis.

2. Common Anomaly Patterns

- **High bytes_in and low bytes_out sessions**
  - This pattern suggests possible infiltration attempts, where a malicious actor is exfiltrating data or scanning the network.
  - Such sessions should be monitored closely to prevent data breaches.
- **High traffic_rate with irregular avg_packet_size**
  - May indicate Distributed Denial-of-Service (DDoS) attacks or network congestion caused by botnet activity.
  - Sudden spikes in these values can be a sign of malicious traffic.
- **Unusual activity on non-standard ports**
  - Network traffic directed towards uncommon ports (e.g., TCP/UDP ports above 49152) may be a sign of unauthorized access attempts or covert communication channels.
  - Frequent anomalies on ports related to remote desktop (3389), SSH (22), or database services (3306) could indicate targeted attacks.

3. Suspicious Source IPs

- Repeated interactions from specific IPs across different timeframes suggest potential brute-force attacks or reconnaissance scanning.
- IPs with frequent connections but short session durations could be automated scripts or botnets probing for vulnerabilities.
- Source IPs from countries with historically high cyber threat activities (based on known threat intelligence feeds) should be flagged for further investigation.

4. Geographic Trends in Anomalies

- Anomalous traffic is frequently originating from specific country codes, possibly indicating targeted or bot-related attacks.
- Unusual spikes in traffic from regions not commonly associated with legitimate user activity could be an indicator of malicious intent.
- Cross-checking against known IP blacklists could provide additional insights into potential threats.

5. Recommendations for Mitigation

- Implement firewall rules to block high-risk IPs and restrict access to critical services.
- Monitor non-standard port activity and enforce strict network segmentation to limit unauthorized access.

- Conduct further analysis on flagged traffic using deep packet inspection (DPI) and log correlation tools to validate threats.
- Enable anomaly-based intrusion detection to detect and mitigate threats in real-time.