In [128…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import shap
import networkx as nx
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Input
```

In [129…

```
# Load dataset
df = pd.read_csv("CloudWatch_Traffic_Web_Attack.csv")

# Check structure
print(df.info())
print(df.head())
```

```
<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
None
   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   
```

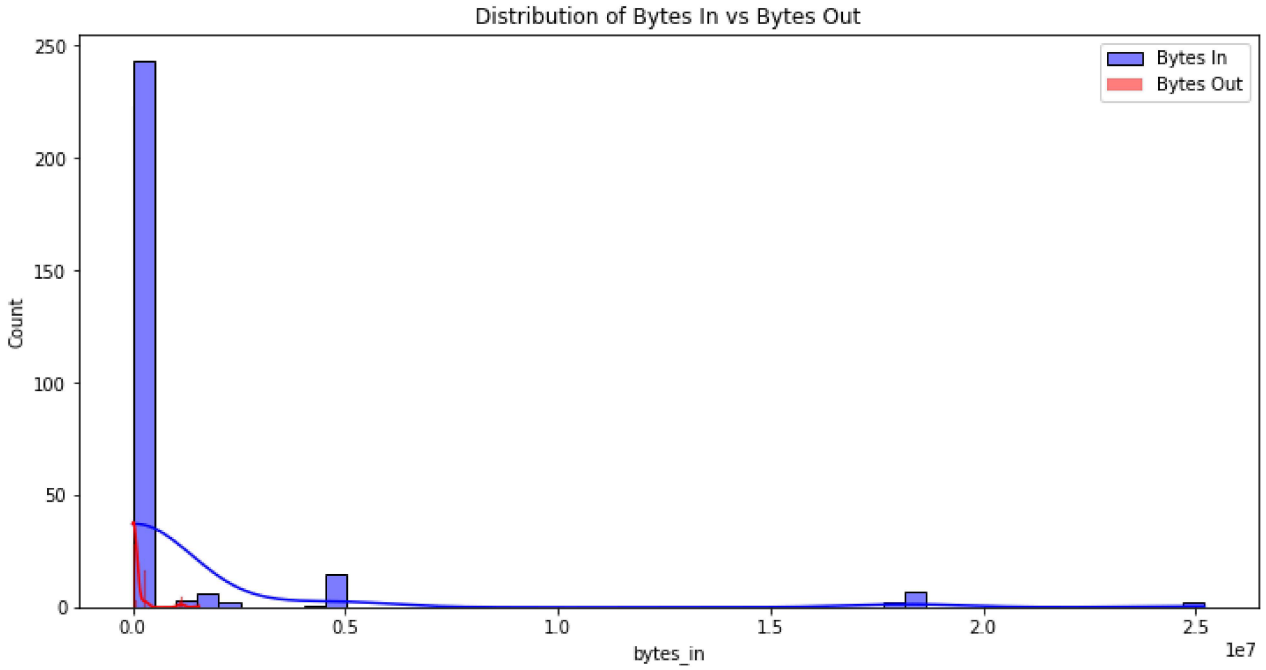
```
2  165.225.212.255                CA    HTTPS           200       443
3   136.226.64.114                US    HTTPS           200       443
4   165.225.240.79                NL    HTTPS           200       443

         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
```

In [130…

```
#Clean and Prepare Data

df = df.drop_duplicates()

# Convert timestamps
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'])

# Standardize country codes
df['src_ip_country_code'] = df['src_ip_country_code'].str.upper()
```

In [131…

```
#Traffic distribution

plt.figure(figsize=(12,6))
sns.histplot(df['bytes_in'], bins=50, kde=True, color='blue', label="Bytes In")
sns.histplot(df['bytes_out'], bins=50, kde=True, color='red', label="Bytes Out")
plt.legend(); plt.title("Distribution of Bytes In vs Bytes Out")
plt.show()
```

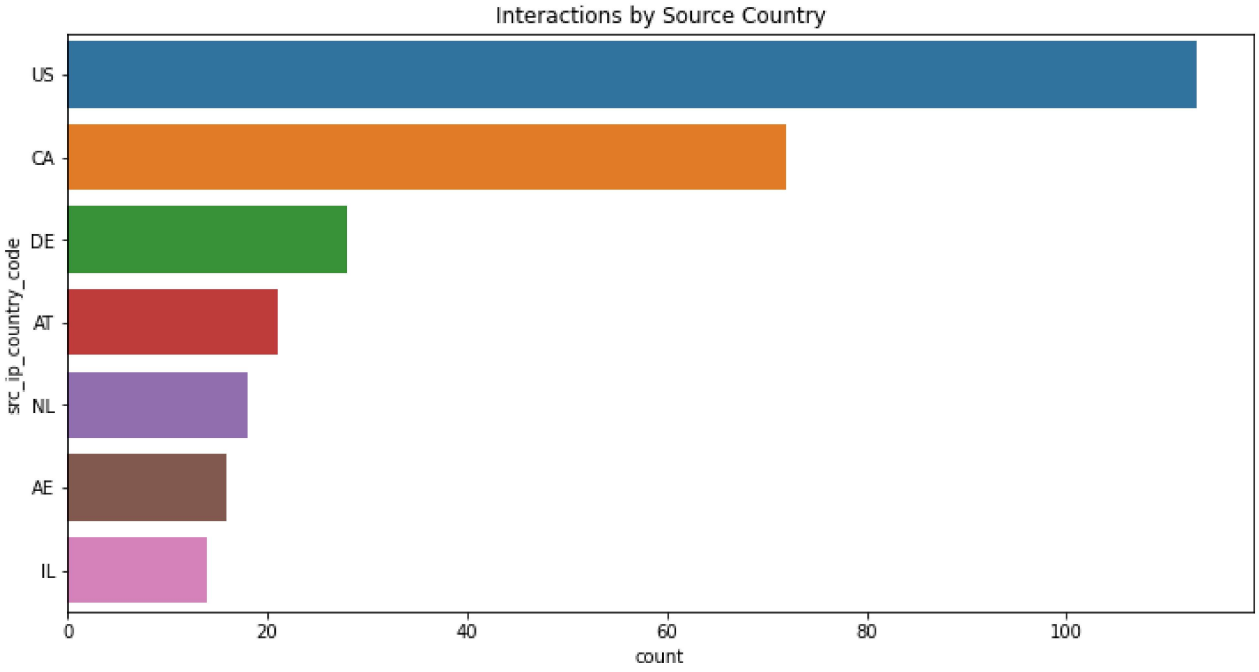


In [132…

```
#Feature Engineering

df['session_duration'] = (df['end_time'] - df['creation_time']).dt.total_seconds()
df['avg_packet_size'] = (df['bytes_in'] + df['bytes_out']) / df['session_duration']
```

In [145…

```
#Country & Port Analysis

suspicious_df = df[df['detection_types'] == "Suspicious"]
plt.figure(figsize=(12, 6))
sns.countplot(y="src_ip_country_code", data=df,
              order=df['src_ip_country_code'].value_counts().index)
plt.title("Interactions by Source Country")
plt.show()
```



In [134…

```
#Anomaly Detection (Isolation Forest)

features = df[['bytes_in', 'bytes_out', 'session_duration', 'avg_packet_size']]
model = IsolationForest(contamination=0.05, random_state=42)
df['anomaly'] = model.fit_predict(features)

# Label anomalies
df['anomaly'] = df['anomaly'].apply(lambda x: "Suspicious" if x==-1 else "Normal")
print(df['anomaly'].value_counts())
```

```
Normal        267
Suspicious     15
Name: anomaly, dtype: int64
```

In [135…

```
#Classification (Random Forest)

df['is_suspicious'] = (df['detection_types']=="waf_rule").astype(int)

X = df[['bytes_in','bytes_out','session_duration']]
y = df['is_suspicious']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
              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
```

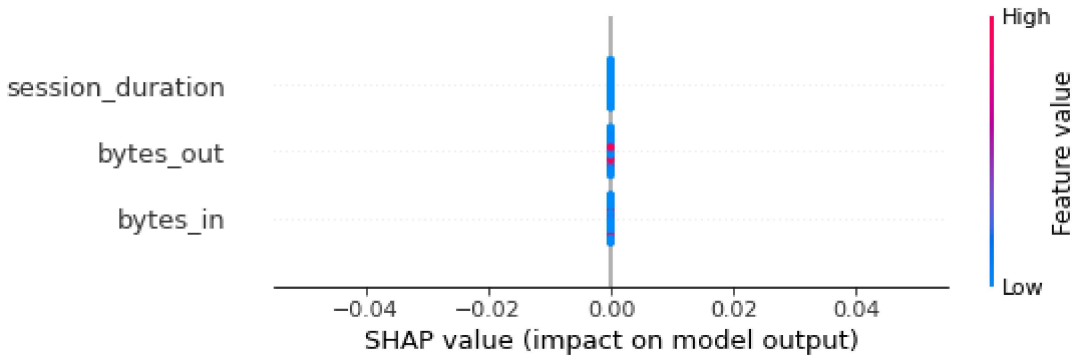
In [136…

```
#Neural Network Model

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_

model = Sequential([
    Dense(16, activation="relu", input_shape=(X_train.shape[1],)),
    Dense(8, activation="relu"),
    Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=10, batch_size=8, verbose=1)

loss, acc = model.evaluate(X_test, y_test)
print(f"Neural Net Test Accuracy: {acc*100:.2f}%")
```

```
Epoch 1/10
25/25 [==============================] - 0s 521us/step - loss: 0.6622 - accuracy: 0.7411
Epoch 2/10
25/25 [==============================] - 0s 574us/step - loss: 0.6274 - accuracy: 1.0000
Epoch 3/10
25/25 [==============================] - 0s 507us/step - loss: 0.5803 - accuracy: 1.0000
Epoch 4/10
25/25 [==============================] - 0s 565us/step - loss: 0.5218 - accuracy: 1.0000
Epoch 5/10
25/25 [==============================] - 0s 565us/step - loss: 0.4568 - accuracy: 1.0000
Epoch 6/10
25/25 [==============================] - 0s 578us/step - loss: 0.3884 - accuracy: 1.0000
Epoch 7/10
25/25 [==============================] - 0s 533us/step - loss: 0.3169 - accuracy: 1.0000
Epoch 8/10
25/25 [==============================] - 0s 542us/step - loss: 0.2463 - accuracy: 1.0000
Epoch 9/10
25/25 [==============================] - 0s 532us/step - loss: 0.1836 - accuracy: 1.0000
Epoch 10/10
25/25 [==============================] - 0s 578us/step - loss: 0.1289 - accuracy: 1.0000
3/3 [==============================] - 0s 1ms/step - loss: 0.1077 - accuracy: 1.0000
Neural Net Test Accuracy: 100.00%
```

In [137…

```
#Model Explainability (SHAP)

explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)

# Summary plot
shap.summary_plot(shap_values, X_test, feature_names=X.columns)
```



In [138…

```
#Cross-Validation & Hyperparameter Tuning

param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

grid = GridSearchCV(RandomForestClassifier(random_state=42),
                    param_grid, cv=5, scoring='accuracy', n_jobs=-1)

grid.fit(X, y)
print("Best Params:", grid.best_params_)
print("Best CV Accuracy:", grid.best_score_)
```

```
Best Params: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Best CV Accuracy: 1.0
```

In [139…

```
#Deep Learning with Regularization

model = Sequential([
    Dense(64, activation="relu", input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(32, activation="relu"),
    Dropout(0.3),
    Dense(1, activation="sigmoid")
])
```

In [140…

```
#For Better Visualization

G = nx.Graph()
for idx, row in df.iterrows():
    G.add_edge(row['src_ip'], row['dst_ip'])

plt.figure(figsize=(16, 10))
```

```
nx.draw_networkx(G, node_size=30, font_size=10, edge_color="gray")
plt.title("Network Graph of Source → Destination IPs")
plt.show()
```