Clean the dataset to ensure accuracy in analysis.

```
import pandas as pd


file_path = '/content/CloudWatch_Traffic_Web_Attack.csv'
data = pd.read_csv(file_path)


print("Initial Data:")
print(data.head())
```

```
Initial Data:
   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
```

```
# 1. Handle Missing Values
# Check for missing values
missing_values = data.isnull().sum()
print("\nMissing Values:")
print(missing_values[missing_values > 0])
```

```

Missing Values:
Series([], dtype: int64)
```

```
# Fill missing values or drop rows/columns with missing values
# Example: Dropping rows with any missing values
data_cleaned = data.dropna()
print(data_cleaned.head())
```

```
   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
```

```
# 2. Convert Categorical Variables to Numerical
# Convert 'src_ip_country_code' and 'protocol' to categorical codes
data_cleaned['src_ip_country_code'] = data_cleaned['src_ip_country_code'].astype('category').cat.codes
data_cleaned['protocol'] = data_cleaned['protocol'].astype('category').cat.codes
```

```
print(data_cleaned['protocol'])
```

```
0      HTTPS
1      HTTPS
2      HTTPS
3      HTTPS
4      HTTPS
       ...
277    HTTPS
278    HTTPS
279    HTTPS
280    HTTPS
281    HTTPS
Name: protocol, Length: 282, dtype: object
```

```
# 3. Normalize Numerical Features
# Example: Normalize 'bytes_in' and 'bytes_out' using Min-Max Scaling
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
data_cleaned[['bytes_in', 'bytes_out']] = scaler.fit_transform(data_cleaned[['bytes_in', 'bytes_out']])
```

```
print(data_cleaned[['bytes_in', 'bytes_out']])
```

```
      bytes_in  bytes_out
0    0.000221   0.008292
1    0.001225   0.011621
2    0.001129   0.008599
3    0.001210   0.009117
4    0.000257   0.008870
..        ...        ...
277  0.001638   0.008414
278  0.000143   0.002015
279  1.000000   1.000000
280  0.000226   0.007731
281  0.000357   0.003727

[282 rows x 2 columns]
```

```
# 4. Check Data Types
print("\nData Types After Preprocessing:")
print(data_cleaned.dtypes)
```

```

Data Types After Preprocessing:
bytes_in               float64
bytes_out              float64
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
```

```
# 5. Display the cleaned dataset
print("\nCleaned Data:")
```

```
print(data_cleaned.head())
```

```
Cleaned Data:
   bytes_in  bytes_out         creation_time             end_time  \
0  0.000221   0.008292  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
1  0.001225   0.011621  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
2  0.001129   0.008599  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
3  0.001210   0.009117  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
4  0.000257   0.008870  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
```

```
# Save the cleaned dataset to a new CSV file
data_cleaned.to_csv('/content/Cleaned_CloudWatch_Traffic_Web_Attack.csv', index=False)
```

Exploratory Data Analysis (EDA)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# Load the dataset
file_path = '/content/CloudWatch_Traffic_Web_Attack.csv'
data = pd.read_csv(file_path)


# Display the first few rows of the dataset
print("Initial Data:")
print(data.head())
```

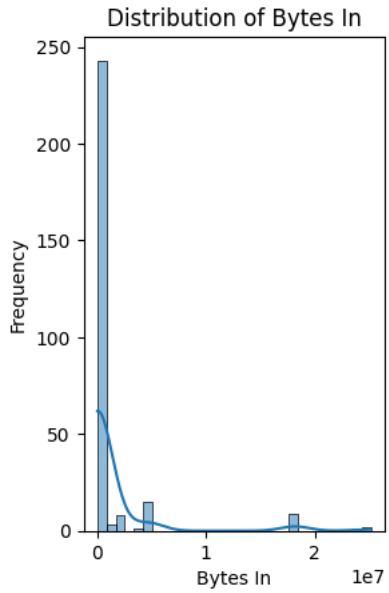
```
Initial Data:
   bytes_in  bytes_out         creation_time             end_time  \
0      5602      12990  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
1     30912      18186  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
2     28506      13468  2024-04-25T23:00:00Z  2024-04-25T23:10:00Z
3     30546      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
```

```
# 1. Visualize distributions of key variables using histograms
plt.figure(figsize=(12, 6))
```

```
<Figure size 1200x600 with 0 Axes>
<Figure size 1200x600 with 0 Axes>
```

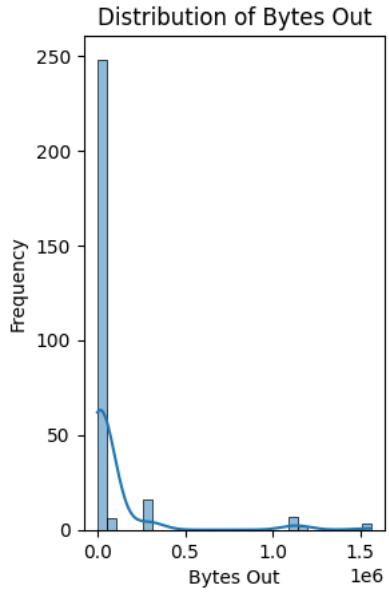
```
# Histogram for 'bytes_in'
plt.subplot(1, 2, 1)
sns.histplot(data['bytes_in'], bins=30, kde=True)
plt.title('Distribution of Bytes In')
plt.xlabel('Bytes In')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```




```
# Histogram for 'bytes_out'
plt.subplot(1, 2, 2)
sns.histplot(data['bytes_out'], bins=30, kde=True)
plt.title('Distribution of Bytes Out')
plt.xlabel('Bytes Out')
plt.ylabel('Frequency')
```

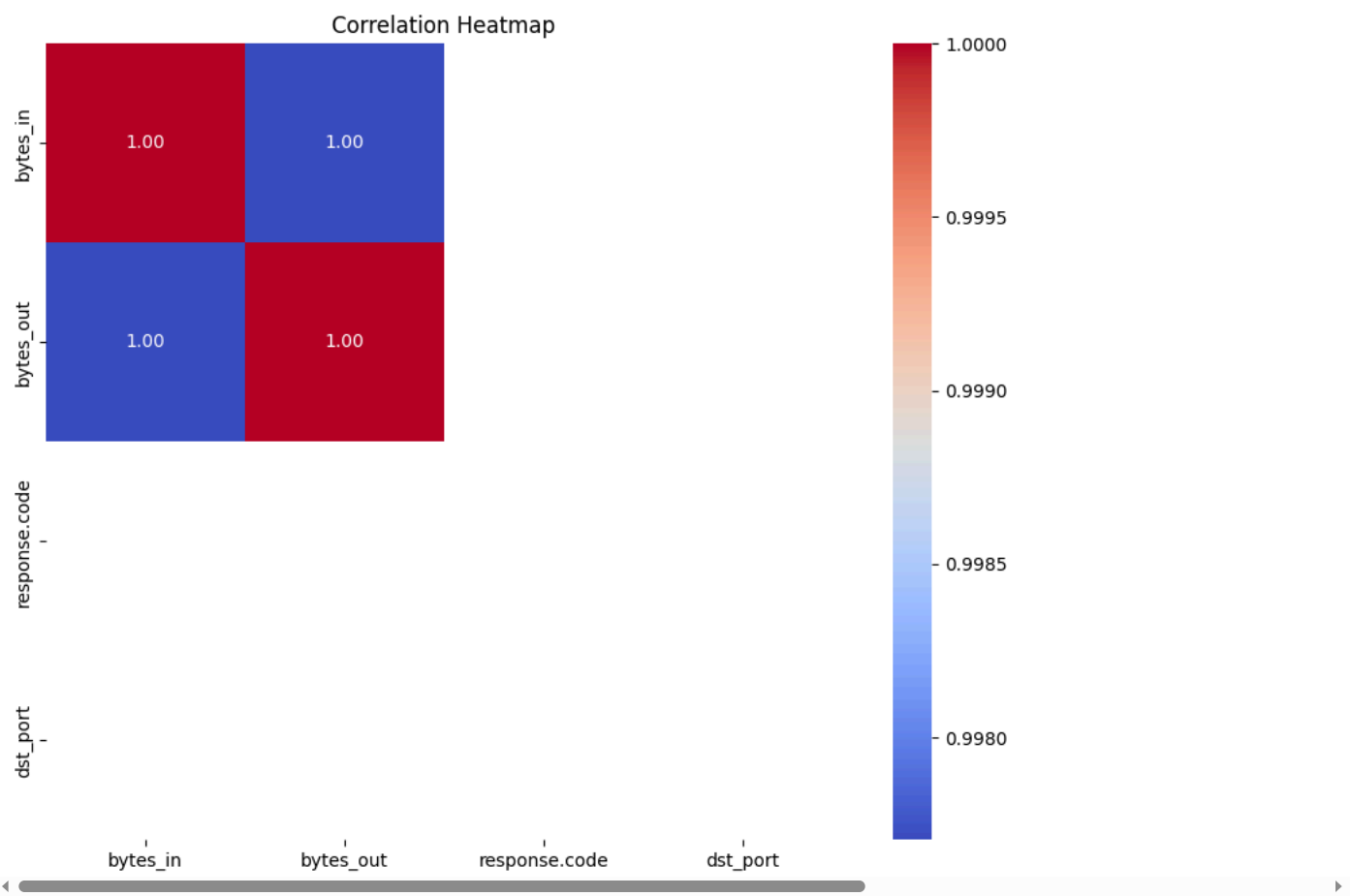
```
Text(0, 0.5, 'Frequency')
```



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

```
<Figure size 640x480 with 0 Axes>
```

```
# 2. Analyze correlations between features and the target variable using heatmaps
# Select only numeric columns for correlation analysis
numeric_data = data.select_dtypes(include=['float64', 'int64'])


# Calculate the correlation matrix
correlation_matrix = numeric_data.corr()


plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Heatmap')
plt.show()
```

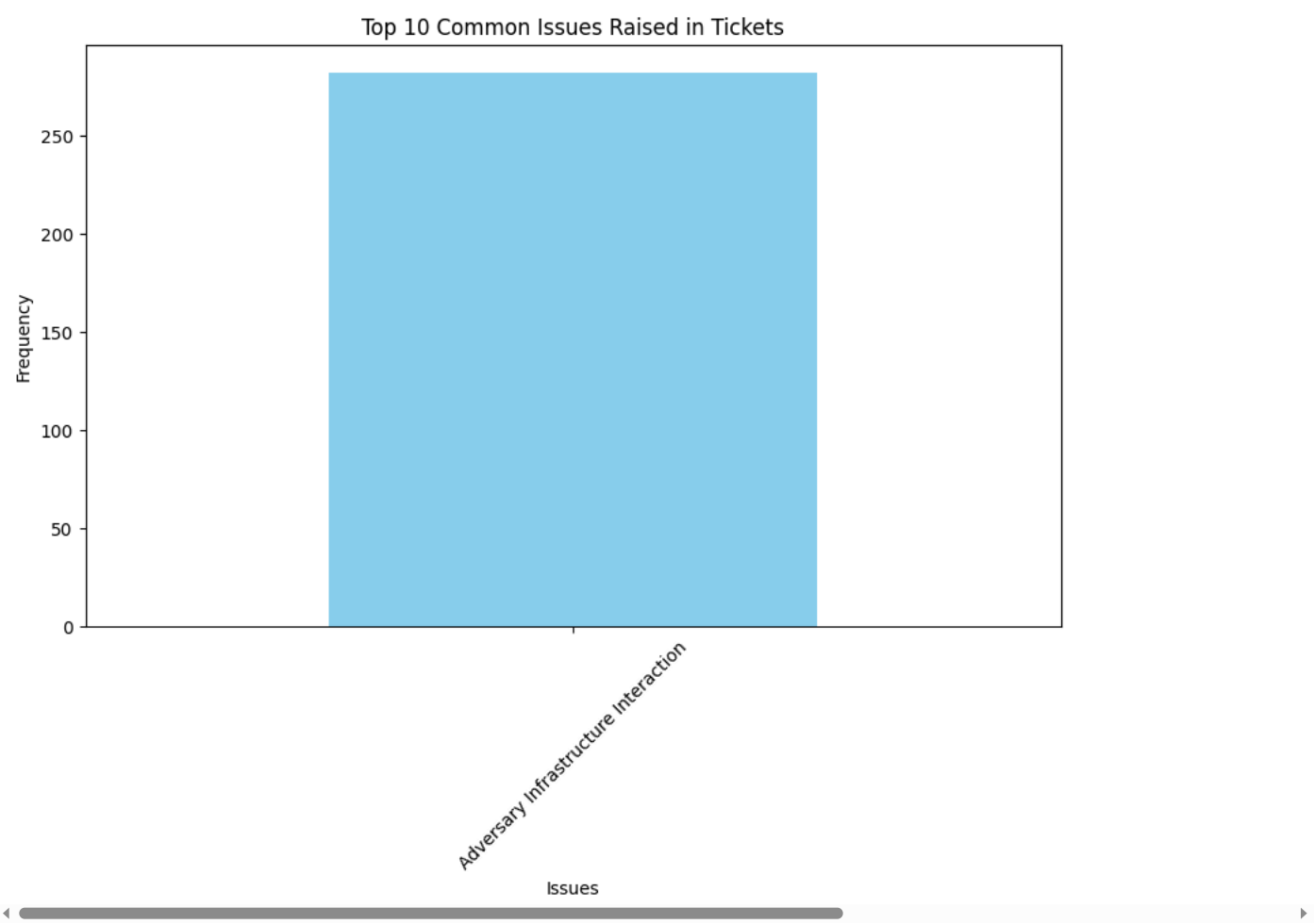



```
# 3. Identify common issues raised in tickets by analyzing the Ticket Subject and Ticket Description
# Assuming 'observation_name' is the equivalent of Ticket Subject
# Count occurrences of each unique observation name
common_issues = data['observation_name'].value_counts()


# Display the top 10 common issues
print("\nTop 10 Common Issues Raised in Tickets:")
print(common_issues.head(10))
```

```
Top 10 Common Issues Raised in Tickets:
observation_name
Adversary Infrastructure Interaction    282
Name: count, dtype: int64
```

```
# Visualize the top 10 common issues
plt.figure(figsize=(10, 6))
common_issues.head(10).plot(kind='bar', color='skyblue')
plt.title('Top 10 Common Issues Raised in Tickets')
```

```
plt.xlabel('Issues')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()
```



Feature Engineering

```
import pandas as pd


# Load the dataset
file_path = '/content/CloudWatch_Traffic_Web_Attack.csv'
data = pd.read_csv(file_path)


# Display the first few rows of the dataset
print("Initial Data:")
print(data.head())
```

```
Initial Data:
   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
```

```
# 1. Derive features such as the time since the last interaction
# Convert 'creation_time' and 'end_time' to datetime
data['creation_time'] = pd.to_datetime(data['creation_time'])
data['end_time'] = pd.to_datetime(data['end_time'])


# Calculate the duration of each interaction in seconds
data['interaction_duration'] = (data['end_time'] - data['creation_time']).dt.total_seconds()


# Assuming we want to calculate the time since the last interaction
# For this example, we will use the maximum end_time as the reference point
latest_interaction_time = data['end_time'].max()
data['time_since_last_interaction'] = (latest_interaction_time - data['end_time']).dt.total_seconds()


# 2. Categorize customers into segments based on response code
# Create a new feature for response code categories
data['response_category'] = pd.cut(data['response.code'], bins=[0, 199, 299, 399, 499, 599],
                                   labels=['Success', 'Redirection', 'Client Error', 'Server Error', 'Unknown'])


# Display the new features
print("\nData with New Features:")
print(data[['interaction_duration', 'time_since_last_interaction', 'response_category']].head())
```

```
Data with New Features:
   interaction_duration  time_since_last_interaction response_category
0                 600.0                      39000.0       Redirection
1                 600.0                      39000.0       Redirection
2                 600.0                      39000.0       Redirection
3                 600.0                      39000.0       Redirection
4                 600.0                      39000.0       Redirection
```

```
# 3. Categorize based on bytes_in and bytes_out for segmentation
# Create segments based on bytes_in
data['bytes_in_category'] = pd.cut(data['bytes_in'], bins=[0, 10000, 50000, 100000, 500000, float('inf')],
                                   labels=['Very Low', 'Low', 'Medium', 'High', 'Very High'])


# Create segments based on bytes_out
data['bytes_out_category'] = pd.cut(data['bytes_out'], bins=[0, 10000, 50000, 100000, 500000, float('inf')],
                                    labels=['Very Low', 'Low', 'Medium', 'High', 'Very High'])


# Display the updated dataset with new categorical features
print("\nUpdated Data with Categorical Features:")
print(data[['bytes_in_category', 'bytes_out_category']].head())
```

```
Updated Data with Categorical Features:
  bytes_in_category bytes_out_category
0          Very Low               Low
1               Low               Low
2               Low               Low
3               Low               Low
4          Very Low               Low
```

```
# Save the updated dataset to a new CSV file
data.to_csv('/content/Updated_CloudWatch_Traffic_Web_Attack.csv', index=False)
```

Model Building

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns


# Load the dataset
file_path = '/content/CloudWatch_Traffic_Web_Attack.csv'
data = pd.read_csv(file_path)


# Display the first few rows of the dataset
print("Initial Data:")
print(data.head())
```

```
Initial Data:
   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                    0         0            200       443
1     165.225.33.6                    6         0            200       443
2  165.225.212.255                    2         0            200       443
3   136.226.64.114                    6         0            200       443
4   165.225.240.79                    5         0            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
```

```
# Preprocessing: Convert 'creation_time' and 'end_time' to datetime
data['creation_time'] = pd.to_datetime(data['creation_time'])
data['end_time'] = pd.to_datetime(data['end_time'])


# Create response category based on response code
data['response_category'] = pd.cut(data['response.code'], bins=[0, 199, 299, 399, 499, 599],
                                   labels=['Success', 'Redirection', 'Client Error', 'Server Error', 'Unknown'])


# Convert categorical variables to numerical
data['src_ip_country_code'] = data['src_ip_country_code'].astype('category').cat.codes
data['protocol'] = data['protocol'].astype('category').cat.codes
data['response_category'] = data['response_category'].astype('category').cat.codes


# Define features and target variable
# Assuming 'response.code' is the target variable (you can change this based on your analysis)
X = data.drop(columns=['response.code', 'creation_time', 'end_time', 'observation_name'])  # Features
y = data['response.code']  # Target variable


X = X.select_dtypes(include=['int64', 'float64'])


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Train a Random Forest Classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

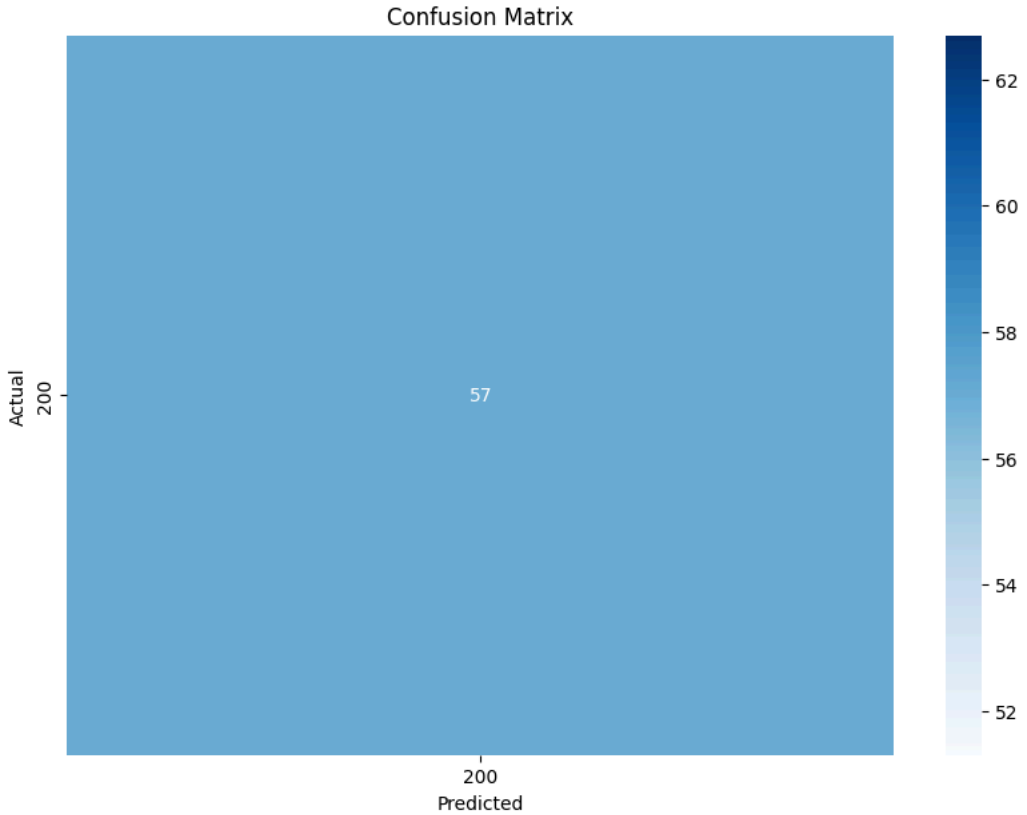
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Make predictions
y_pred = model.predict(X_test)


# 1. Evaluate model performance using metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)


print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

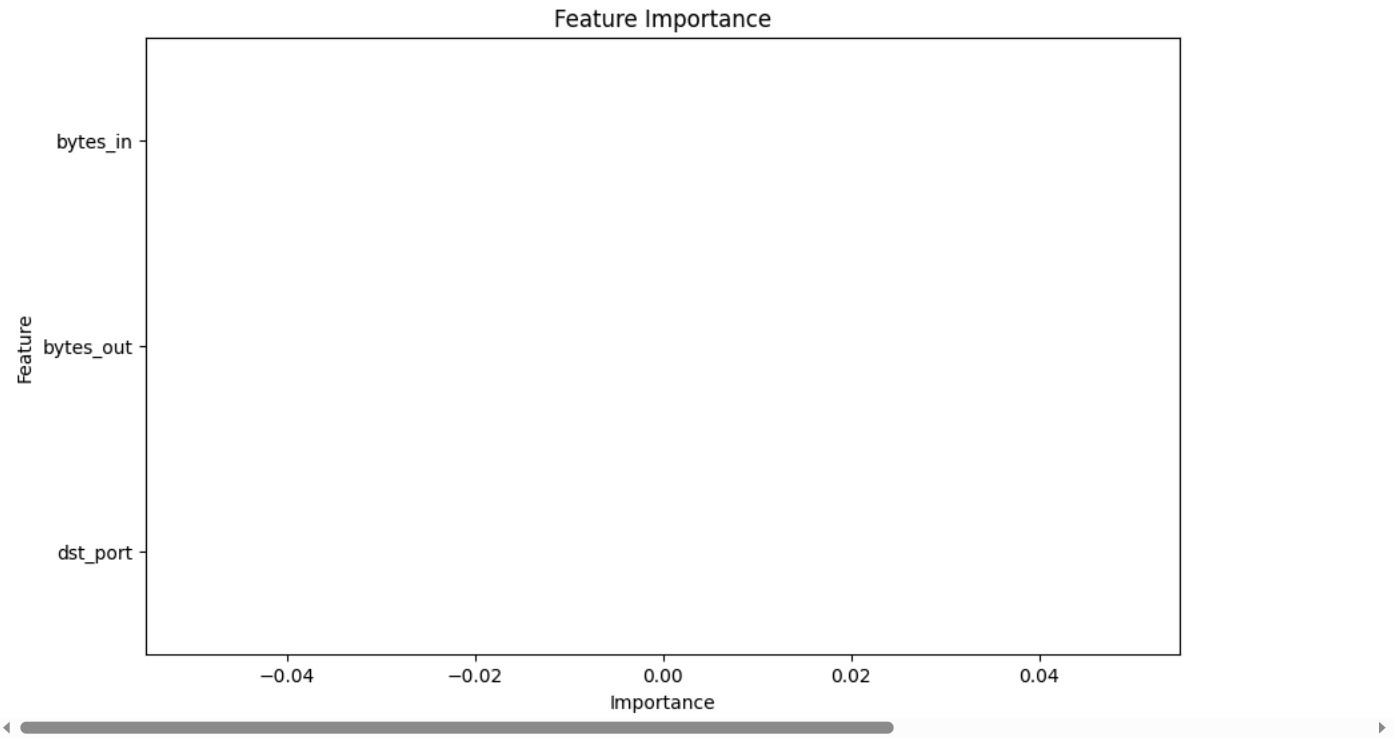
```
# 2. Analyze confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_, yticklabels=model.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:407: UserWarning: A single label was found in 'y_true' and 'y
  warnings.warn(
```




```
# 3. Perform feature importance analysis
feature_importances = model.feature_importances_
features = X.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)


# Display feature importance
plt.figure(figsize=(10, 6))
```

```
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importance')
plt.show()
```



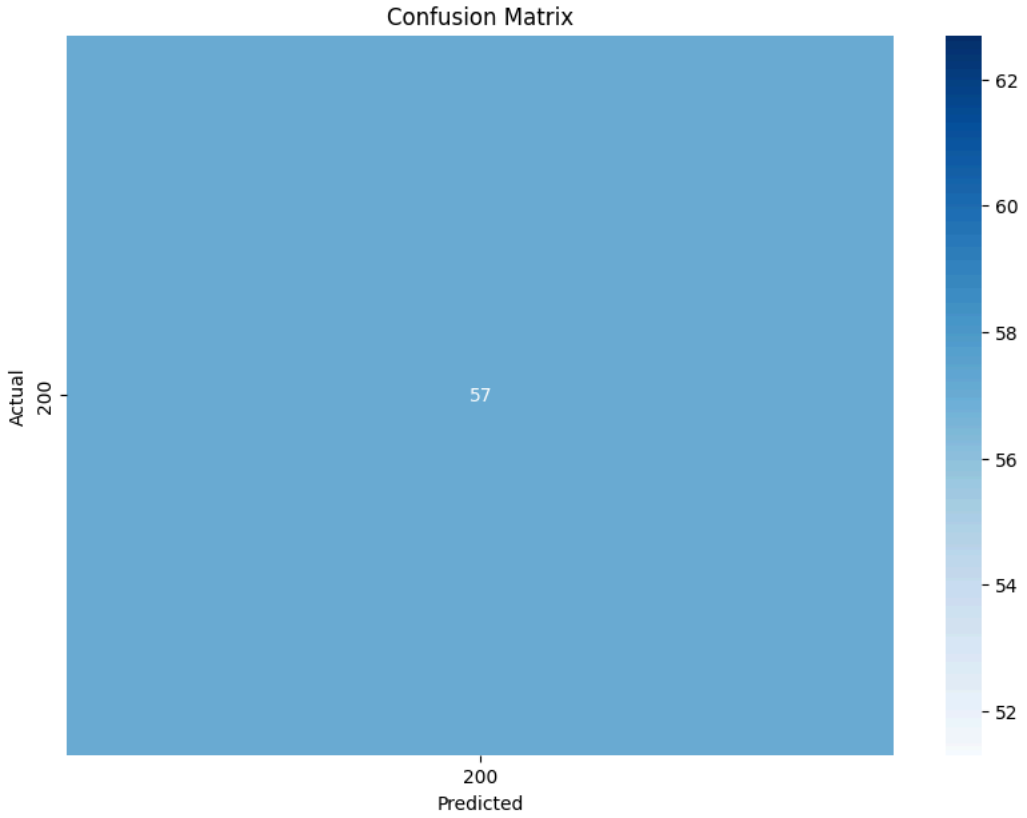
Model Evaluation

```
# 1. Evaluate model performance using metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)


print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

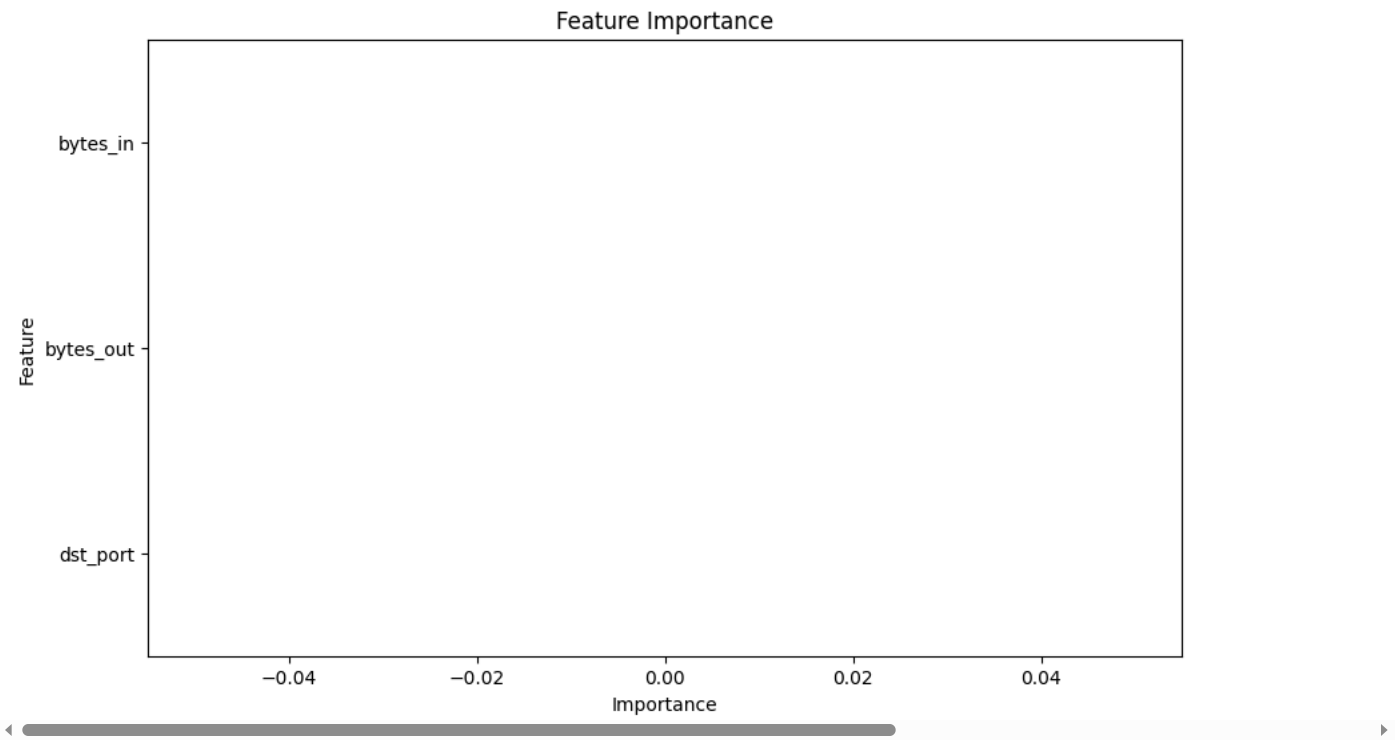
```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

```
# 2. Analyze confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_, yticklabels=model.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:407: UserWarning: A single label was found in 'y_true' and 'y
  warnings.warn(
```




```
# 3. Perform feature importance analysis
feature_importances = model.feature_importances_
features = X.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)


# Display feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importance')
plt.show()
```



Visualization of Results

```
pip install dash
```

```
Collecting dash
  Downloading dash-3.0.1-py3-none-any.whl.metadata (10 kB)
Collecting Flask<3.1,>=1.0.4 (from dash)
  Downloading flask-3.0.3-py3-none-any.whl.metadata (3.2 kB)
Collecting Werkzeug<3.1 (from dash)
  Downloading werkzeug-3.0.6-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.11/dist-packages (from dash) (5.24.1)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.11/dist-packages (from dash) (8.6.1)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.11/dist-packages (from dash) (4.12.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from dash) (2.32.3)
Collecting retrying (from dash)
  Downloading retrying-1.3.4-py3-none-any.whl.metadata (6.9 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.11/dist-packages (from dash) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from dash) (75.1.0)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (3.1.6)
Requirement already satisfied: itsdangerous>=2.1.2 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (8.1.8)
Requirement already satisfied: blinker>=1.6.2 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (1.9.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly>=5.0.0->dash) (9.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from plotly>=5.0.0->dash) (24.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from Werkzeug<3.1->dash) (3.0.2)
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.11/dist-packages (from importlib-metadata->dash) (3.21.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (2025.1.31)
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from retrying->dash) (1.17.0)
Downloading dash-3.0.1-py3-none-any.whl (8.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.0/8.0 MB 34.7 MB/s eta 0:00:00
Downloading flask-3.0.3-py3-none-any.whl (101 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.7/101.7 kB 5.5 MB/s eta 0:00:00
Downloading werkzeug-3.0.6-py3-none-any.whl (227 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 228.0/228.0 kB 12.3 MB/s eta 0:00:00
Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Installing collected packages: Werkzeug, retrying, Flask, dash
  Attempting uninstall: Werkzeug
    Found existing installation: Werkzeug 3.1.3
    Uninstalling Werkzeug-3.1.3:
      Successfully uninstalled Werkzeug-3.1.3
  Attempting uninstall: Flask
    Found existing installation: Flask 3.1.0
    Uninstalling Flask-3.1.0:
      Successfully uninstalled Flask-3.1.0
Successfully installed Flask-3.0.3 Werkzeug-3.0.6 dash-3.0.1 retrying-1.3.4
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from dash import Dash, dcc, html
import plotly.express as px
```

```
pip install pandas scikit-learn matplotlib seaborn dash plotly
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: dash in /usr/local/lib/python3.11/dist-packages (3.0.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (5.24.1)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.11/dist-packages (from dash) (3.0.3)
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.11/dist-packages (from dash) (3.0.6)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.11/dist-packages (from dash) (8.6.1)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.11/dist-packages (from dash) (4.12.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from dash) (2.32.3)
Requirement already satisfied: retrying in /usr/local/lib/python3.11/dist-packages (from dash) (1.3.4)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.11/dist-packages (from dash) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from dash) (75.1.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly) (9.0.0)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (3.1.6)
Requirement already satisfied: itsdangerous>=2.1.2 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (8.1.8)
Requirement already satisfied: blinker>=1.6.2 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (1.9.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from Werkzeug<3.1->dash) (3.0.2)
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.11/dist-packages (from importlib-metadata->dash) (3.21.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (2025.1.31)
```

```
# Load the dataset
file_path = '/content/CloudWatch_Traffic_Web_Attack.csv'
data = pd.read_csv(file_path)
```

```
# Preprocessing
data['creation_time'] = pd.to_datetime(data['creation_time'])
data['end_time'] = pd.to_datetime(data['end_time'])
data['response_category'] = pd.cut(data['response.code'], bins=[0, 199, 299, 399, 499, 599],
                                   labels=['Success', 'Redirection', 'Client Error', 'Server Error', 'Unknown'])
data['src_ip_country_code'] = data['src_ip_country_code'].astype('category').cat.codes
data['protocol'] = data['protocol'].astype('category').cat.codes
data['response_category'] = data['response_category'].astype('category').cat.codes
```

```
# Define features and target variable
X = data.drop(columns=['response.code', 'creation_time', 'end_time', 'observation_name'])
y = data['response.code']
X = X.select_dtypes(include=['int64', 'float64'])
```

```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Train a Random Forest Classifier
model = RandomForestClassifier(random_state=42)
```

```
model.fit(X_train, y_train)
```

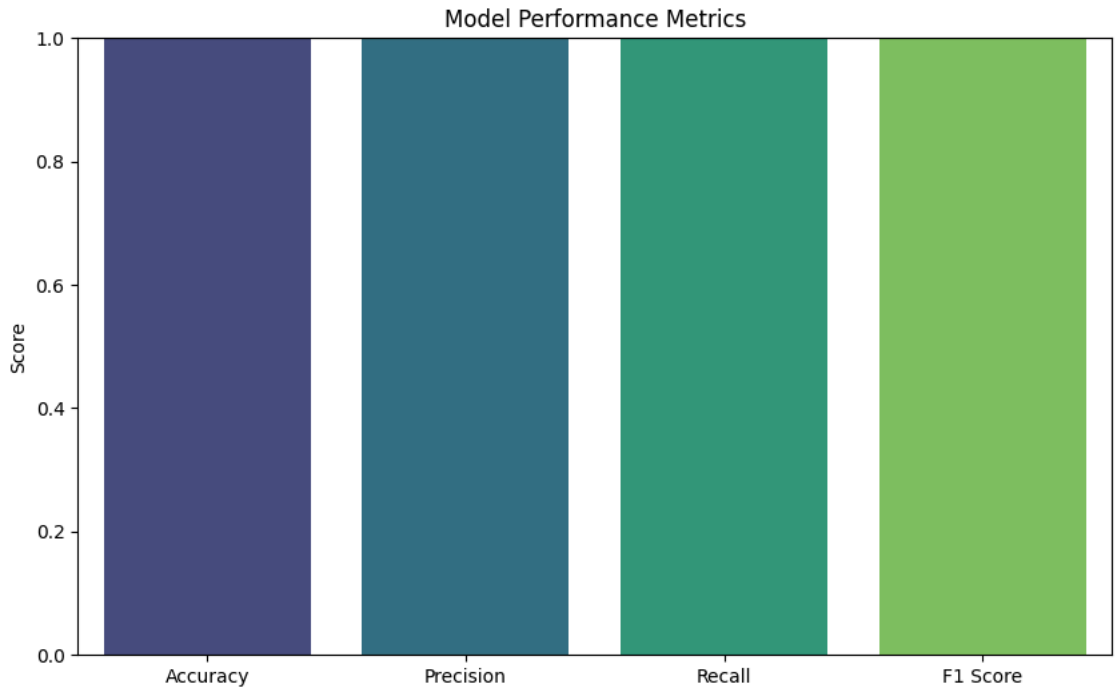
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Make predictions
y_pred = model.predict(X_test)


# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)


# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:407: UserWarning: A single label was found in 'y_true' and 'y
  warnings.warn(
```

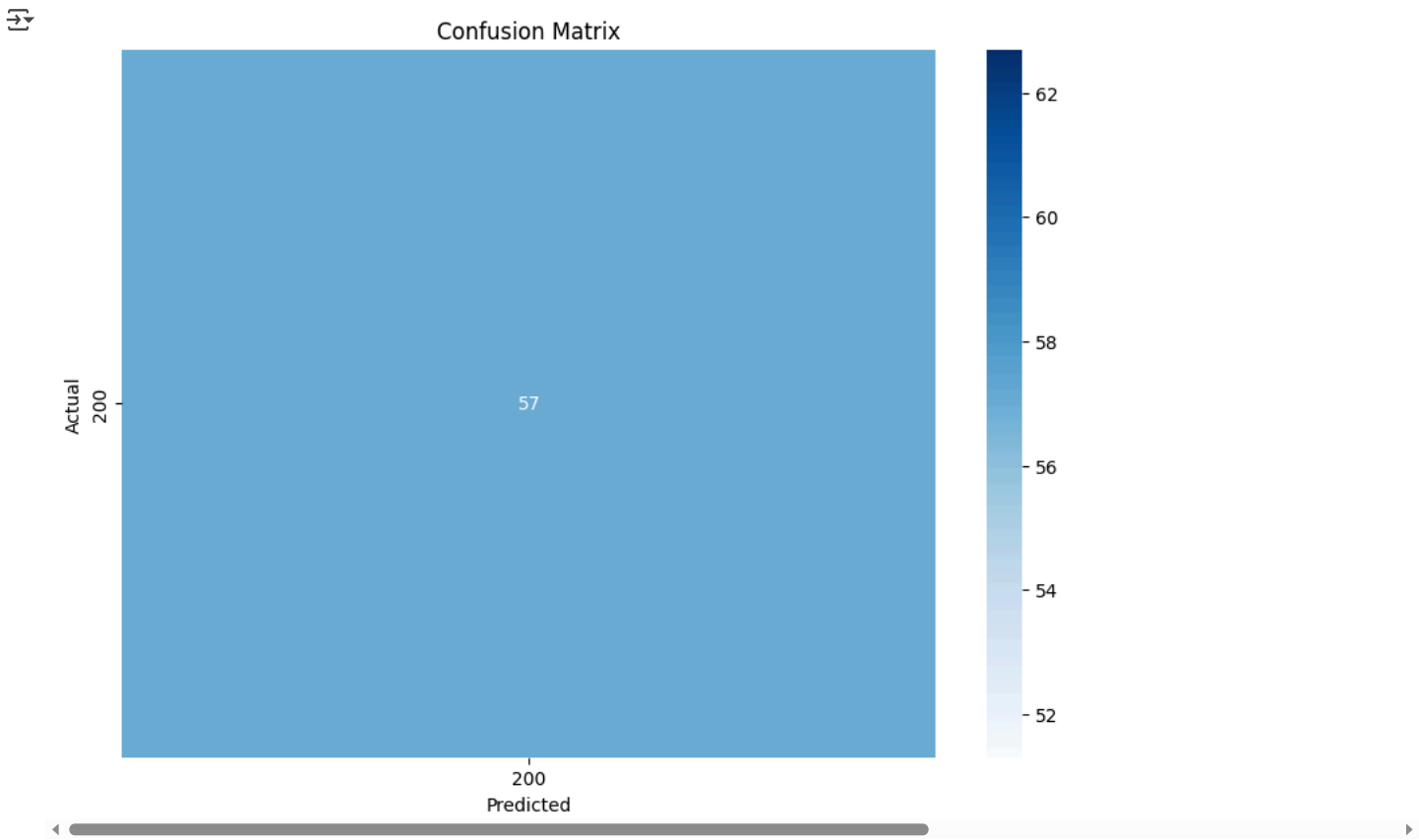
```
# Feature importance
feature_importances = model.feature_importances_
features = X.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importances}).sort_values(by='Importance', ascending=False)


# Visualization of model performance
plt.figure(figsize=(10, 6))
sns.barplot(x=['Accuracy', 'Precision', 'Recall', 'F1 Score'], y=[accuracy, precision, recall, f1], palette='viridis')
plt.title('Model Performance Metrics')
plt.ylabel('Score')
plt.ylim(0, 1)
plt.show()
```

```
<ipython-input-95-52c6b4f8ef72>:3: FutureWarning:

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

  sns.barplot(x=['Accuracy', 'Precision', 'Recall', 'F1 Score'], y=[accuracy, precision, recall, f1], palette='viridis')
```

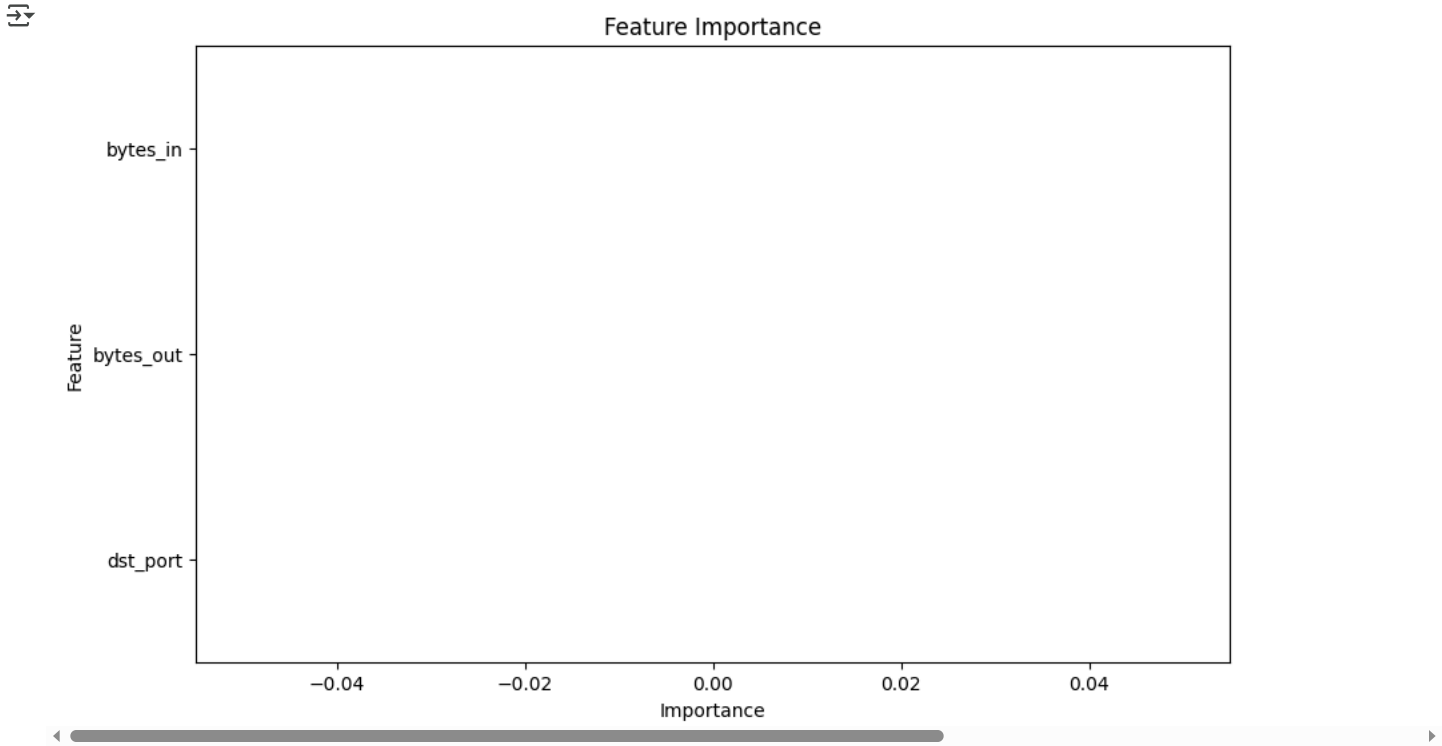



```
# Visualization of confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_, yticklabels=model.classes_)
plt.title('Confusion Matrix')
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

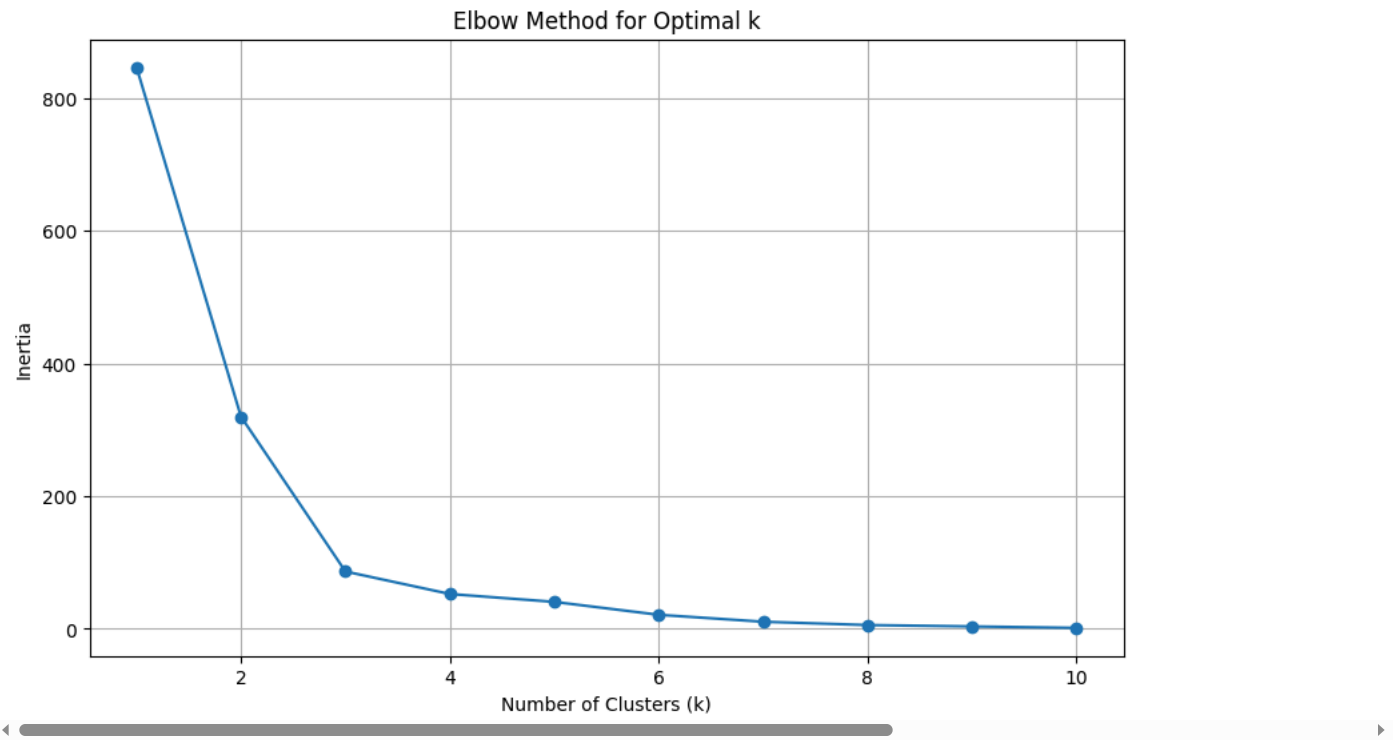



```
# Visualization of feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importance')
plt.show()
```



Customer Segmentation Analysis

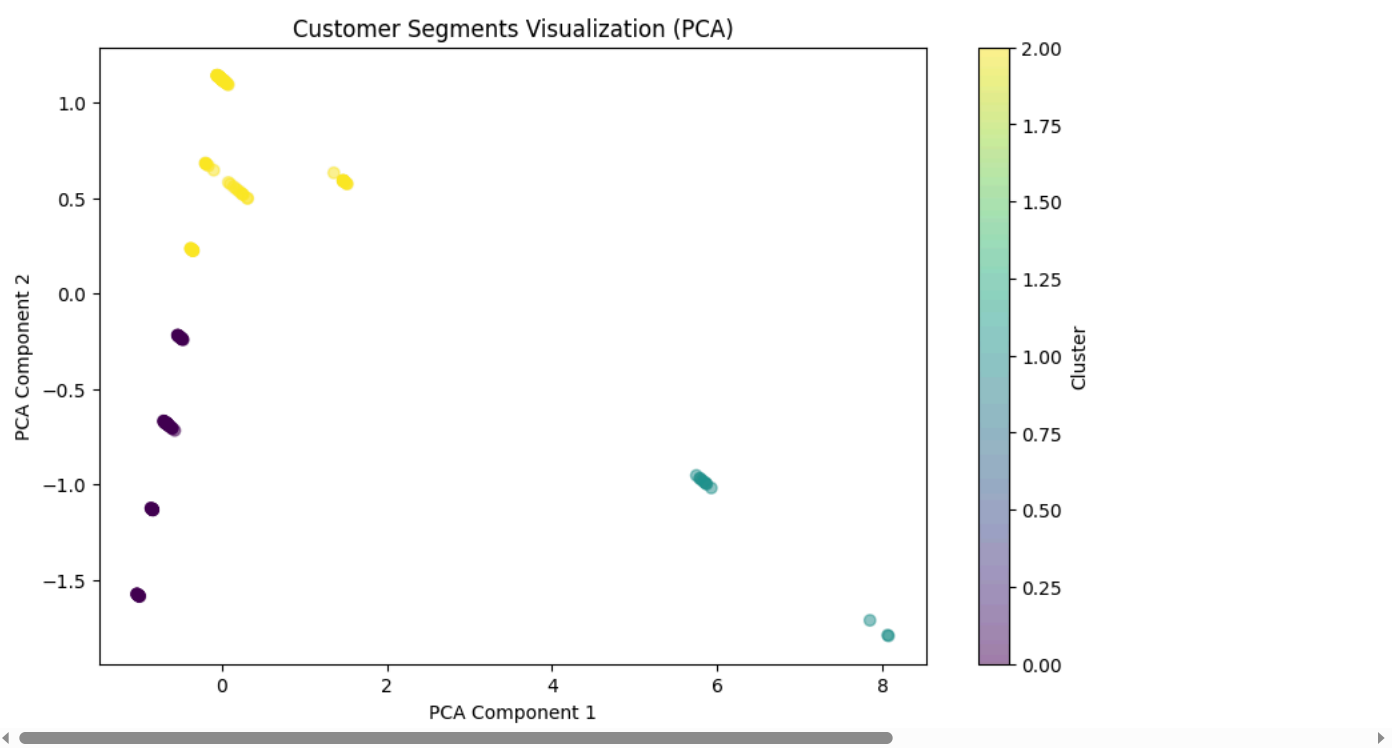
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA


# Load the dataset
file_path = '/content/CloudWatch_Traffic_Web_Attack.csv'
data = pd.read_csv(file_path)


# Preprocessing
data['creation_time'] = pd.to_datetime(data['creation_time'])
data['end_time'] = pd.to_datetime(data['end_time'])
data['response_category'] = pd.cut(data['response.code'], bins=[0, 199, 299, 399, 499, 599],
                                   labels=['Success', 'Redirection', 'Client Error', 'Server Error', 'Unknown'])
data['src_ip_country_code'] = data['src_ip_country_code'].astype('category').cat.codes
data['protocol'] = data['protocol'].astype('category').cat.codes
data['response_category'] = data['response_category'].astype('category').cat.codes


# Define features for clustering
# Selecting relevant features for clustering
features = data[['src_ip_country_code', 'protocol', 'response.code', 'bytes_in', 'bytes_out']]
X = features.dropna()  # Drop any rows with missing values


# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


# 1. Use K-Means clustering to segment customers
# Determine the optimal number of clusters using the Elbow method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)


# Plot the Elbow curve
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.grid()
plt.show()
```




```
# From the Elbow method, choose an optimal k (e.g., 3)
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
data['cluster'] = kmeans.fit_predict(X_scaled)


# 2. Analyze the characteristics of each segment
# Calculate the mean values of each numeric feature for each cluster
cluster_analysis = data.groupby('cluster').mean(numeric_only=True)  # Use numeric_only=True to avoid non-numeric columns
print("\nCluster Characteristics:")
print(cluster_analysis)
```

```
Cluster Characteristics:
              bytes_in     bytes_out  src_ip_country_code  protocol  \
cluster
0         1.542768e+04  1.398054e+04             1.817518       0.0
1         1.988073e+07  1.238034e+06             6.000000       0.0
2         7.334265e+05  5.317725e+04             5.654135       0.0

          response.code  dst_port  response_category
cluster
0                 200.0     443.0               1.0
1                 200.0     443.0               1.0
2                 200.0     443.0               1.0
```

```
# Visualize the clusters using PCA for dimensionality reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)


plt.figure(figsize=(10, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=data['cluster'], cmap='viridis', alpha=0.5)
plt.title('Customer Segments Visualization (PCA)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Cluster')
plt.show()
```



Sentiment Analysis

```
pip install pandas textblob matplotlib seaborn
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: textblob in /usr/local/lib/python3.11/dist-packages (0.19.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: nltk>=3.9 in /usr/local/lib/python3.11/dist-packages (from textblob) (3.9.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob) (4.67.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
import pandas as pd
from textblob import TextBlob
import matplotlib.pyplot as plt
import seaborn as sns


# Load the dataset
file_path = '/content/CloudWatch_Traffic_Web_Attack.csv'
data = pd.read_csv(file_path)


# Display the first few rows of the dataset
print("Initial Data:")
print(data.head())
```

```
Initial Data:
   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
```