|  |  |
| --- | --- |
| Project Title | **Cybersecurity: Suspicious Web Threat Interactions** |
| language | Machine learning, python, SQL, Excel |
| Tools | VS code, Jupyter notebook |
| Domain | Data Analyst |
| Project Difficulties level | Advance |

Dataset : Dataset is available in the given link. You can download it at your convenience.

[Click](https://drive.google.com/file/d/1-OpnR9FK8EqGuLFB1k45ctPbl-vuZnC-/view?usp=sharing) [here](https://drive.google.com/file/d/1-OpnR9FK8EqGuLFB1k45ctPbl-vuZnC-/view?usp=sharing) [to](https://drive.google.com/file/d/1-OpnR9FK8EqGuLFB1k45ctPbl-vuZnC-/view?usp=sharing) [download](https://drive.google.com/file/d/1-OpnR9FK8EqGuLFB1k45ctPbl-vuZnC-/view?usp=sharing) [data](https://drive.google.com/file/d/1-OpnR9FK8EqGuLFB1k45ctPbl-vuZnC-/view?usp=sharing) [set](https://drive.google.com/file/d/1-OpnR9FK8EqGuLFB1k45ctPbl-vuZnC-/view?usp=sharing)

# 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.

# Dataset Content

Each entry in the dataset represents a stream of traffic to a web server, including the following columns: bytes\_in: Bytes received by the server. bytes\_out: Bytes sent from the server. creation\_time: Timestamp of when the record was created. end\_time: Timestamp of when the connection ended.

src\_ip: Source IP address. src\_ip\_country\_code: Country code of the source IP. protocol: Protocol used in the connection. response.code: HTTP response code. dst\_port: Destination port on the server. dst\_ip: Destination IP address. rule\_names: Name of the rule that identified the traffic as suspicious. observation\_name: Observations associated with the traffic. source.meta: Metadata related to the source. source.name: Name of the traffic source. time: Timestamp of the detected event. detection\_types: Type of detection applied.

# 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.

**Example : from here you can get idea that how you can create project**

|  |
| --- |
| **Project Overview**  **Objective:** To detect and analyze patterns in web interactions for identifying suspicious or potentially harmful activities.  **Steps** |

|  |
| --- |
| 1. **Data Import and Basic Overview**   import pandas as pd  # Load dataset df = pd.read\_csv('cybersecurity\_data.csv')  # View basic information df.info() df.head()   1. **Data Preprocessing**   Handle missing values, outliers, and data inconsistencies.  # Check for missing values missing\_values = df.isnull().sum()  # Fill or drop missing values as needed df['bytes\_in'].fillna(df['bytes\_in'].median(), inplace=True) df.dropna(subset=['src\_ip', 'dst\_ip'], inplace=True)  # Convert columns to appropriate datatypes df['creation\_time'] = pd.to\_datetime(df['creation\_time']) |

|  |
| --- |
| df['end\_time'] = pd.to\_datetime(df['end\_time'])  **3. Exploratory Data Analysis (EDA)**  **Analyze Traffic Patterns Based on bytes\_in and bytes\_out**  import matplotlib.pyplot as plt import seaborn as sns  # Distribution of bytes in and bytes out plt.figure(figsize=(12, 6)) sns.histplot(df['bytes\_in'], bins=50, color='blue', kde=True, label='Bytes In') sns.histplot(df['bytes\_out'], bins=50, color='red', kde=True, label='Bytes Out') plt.legend() plt.title('Distribution of Bytes In and Bytes Out') plt.show()  **Count of Protocols Used**  plt.figure(figsize=(10, 5)) sns.countplot(x='protocol', data=df, palette='viridis') plt.title('Protocol Count') plt.xticks(rotation=45) |

|  |
| --- |
| plt.show()   1. **Feature Engineering**   Extract useful features, like duration and average packet size, to aid in analysis.  # Duration of the session in seconds df['session\_duration'] = (df['end\_time'] df['creation\_time']).dt.total\_seconds()  # Average packet size df['avg\_packet\_size'] = (df['bytes\_in'] + df['bytes\_out']) / df['session\_duration']   1. **Data Visualization**   **Country-based Interaction Analysis**  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.show()  **Suspicious Activities Based on Ports** |

|  |
| --- |
| plt.figure(figsize=(12, 6)) sns.countplot(x='dst\_port', data=df[df['detection\_types'] ==  'Suspicious'], palette='coolwarm') plt.title('Suspicious Activities Based on Destination Port') plt.xticks(rotation=45) plt.show()  **6. Modeling: Anomaly Detection**  This step uses Isolation Forest, a common technique for detecting anomalies.  from sklearn.ensemble import IsolationForest  # Selecting features for anomaly detection features = df[['bytes\_in', 'bytes\_out', 'session\_duration', 'avg\_packet\_size']]  # Initialize the model model = IsolationForest(contamination=0.05, random\_state=42)  # Fit and predict anomalies df['anomaly'] = model.fit\_predict(features) df['anomaly'] = df['anomaly'].apply(lambda x: 'Suspicious' if x  == -1 else 'Normal') |

|  |
| --- |
| 1. **Evaluation**   Evaluate the anomaly detection model by checking its accuracy in identifying suspicious activities.  # Check the proportion of anomalies detected print(df['anomaly'].value\_counts())  # Display anomaly samples suspicious\_activities = df[df['anomaly'] == 'Suspicious'] print(suspicious\_activities.head())   1. **Visualization of Anomalies**   # Visualize bytes\_in vs bytes\_out with anomalies highlighted plt.figure(figsize=(10, 6)) sns.scatterplot(x='bytes\_in', y='bytes\_out', hue='anomaly', data=df, palette=['green', 'red']) plt.title('Anomalies in Bytes In vs Bytes Out') plt.show()   1. **Report Findings**   Based on the model output and visualizations, interpret the most frequent anomaly patterns, source IPs, and ports related to suspicious activities. |

|  |
| --- |
| **Example Insights:**   * **High bytes\_in and low bytes\_out sessions** could indicate possible infiltration attempts. * **Frequent interactions from specific country codes** may indicate targeted or bot-related attacks. * **High activity on non-standard ports** may signal unauthorized access attempts. |

**Example: You can get the basic idea how you can create a project from here**

**Sample code with output**

Module Importing

In [1]:

import pandas as pd import seaborn as sns import networkx as nx import matplotlib.pyplot as plt from sklearn.preprocessing import StandardScaler, OneHotEncoder from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import classification\_report, accuracy\_score from sklearn.compose import ColumnTransformer from sklearn.pipeline import Pipeline import pandas as pd 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 from tensorflow.keras.layers import Dense, Conv1D,

MaxPooling1D, Flatten, Dropout

|  |
| --- |
| from tensorflow.keras.optimizers import Adam import warnings warnings.filterwarnings("ignore")  2024-05-07 21:10:10.181949: E  external/local\_xla/xla/stream\_executor/cuda/cuda\_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered  2024-05-07 21:10:10.182342: E  external/local\_xla/xla/stream\_executor/cuda/cuda\_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered  2024-05-07 21:10:10.352062: E  external/local\_xla/xla/stream\_executor/cuda/cuda\_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered  In [2]:  *# Load the data into a DataFrame* data = pd.read\_csv("/kaggle/input/cybersecurity-suspicious-web-threatinteractions/CloudWatch\_Traffic\_Web\_Attack.csv")  *# Display the first few rows of the DataFrame to understand its* |

*structure*

data

.

head()

Out[2]:

b

y

t

e

s

\_

i

n

b

yt

e

s

\_

o

ut

cre

ati

on

\_ti

me

en

d\_t

im

e

src

\_ip

src\_i

p\_co

untry

\_cod

e

p

r

o

t

o

c

o

l

res

po

ns

e.c

od

e

d

s

t

\_

p

o

rt

dst

\_ip

rul

e\_

na

m

es

obs

erva

tion

\_na

me

sou

rce.

met

a

sou

rce.

na

me

tim

e

det

ecti

on\_

typ

es

0

5

6

0

2

1

2

9

9

0

20

24-

04-

25

T2

3:0

0:0

Z

0

20

24-

04-

25

T2

3:1

0:0

Z

0

147

.16

1.1

61.

82

AE

H

T

T

P

S

20

0

4

4

3

10.

13

8.6

9.9

7

S

us

pi

ci

ou

s

W

eb

Tr

aff

ic

Adv

ersa

ry

Infra

stru

ctur

e

Inter

actio

n

AW

S\_

VP

C\_

Flo

w

pro

d\_

we

bse

rver

20

24-

04-

25

T2

3:0

0:0

Z

0

waf

\_rul

e

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | |
|  | 1 | 3  0  9 1 2 | 1  8  1 8 6 | 20  24-  04-  25  T2  3:0  0:0  0Z | 20  24-  04-  25  T2  3:1  0:0  0Z | 165  .22  5.3  3.6 | US | H  T  T  P S | 20  0 | 4  4 3 | 10. 13  8.6  9.9  7 | S  us  pi ci  ou s  W  eb  Tr  aff ic | Adv ersa ry Infra stru ctur e  Inter actio n | AW  S\_  VP  C\_  Flo  w | pro d\_ we bse rver | 20  24-  04-  25  T2  3:0  0:0  0Z | waf  \_rul e |
| 2 | 2 8 5 0 6 | 1 3 4 6 8 | 20  24-  04-  25  T2  3:0  0:0  0Z | 20  24-  04-  25  T2  3:1  0:0  0Z | 165  .22  5.2 12.  255 | CA | H  T T  P S | 20  0 | 4 4 3 | 10. 13  8.6  9.9  7 | S  us  pi ci  ou s  W  eb  Tr  aff ic | Adv ersa ry Infra stru ctur e  Inter actio n | AW  S\_  VP  C\_  Flo  w | pro d\_ we bse rver | 20  24-  04-  25  T2  3:0  0:0  0Z | waf  \_rul e |
|  | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | |
|  | 3 | 3  0  5 4 6 | 1  4  2 7 8 | 20  24-  04-  25  T2  3:0  0:0  0Z | 20  24-  04-  25  T2  3:1  0:0  0Z | 136  .22  6.6  4.11  4 | US | H  T  T  P S | 20  0 | 4  4 3 | 10. 13  8.6  9.9  7 | S  us  pi ci  ou s  W  eb  Tr  aff ic | Adv ersa ry Infra stru ctur e  Inter actio n | AW  S\_  VP  C\_  Flo  w | pro d\_ we bse rver | 20  24-  04-  25  T2  3:0  0:0  0Z | waf  \_rul e |
| 4 | 6 5 2 6 | 1 3 8 9 2 | 20  24-  04-  25  T2  3:0  0:0  0Z | 20  24-  04-  25  T2  3:1  0:0  0Z | 165  .22  5.2 40.  79 | NL | H  T T  P S | 20  0 | 4 4 3 | 10. 13  8.6  9.9  7 | S  us  pi ci  ou s  W  eb  Tr  aff ic | Adv ersa ry Infra stru ctur e  Inter actio n | AW  S\_  VP  C\_  Flo  w | pro d\_ we bse rver | 20  24-  04-  25  T2  3:0  0:0  0Z | waf  \_rul e |
| Data Preparation | | | | | | | | | | | | | | | | |

|  |
| --- |
| 1. Data Cleaning  The dataset contains **282 entries** across **16 columns**. There are no **null values** in any of the columns, which is **good news for data integrity**. However, let's proceed with the following data cleaning tasks:   1. **Removing Duplicate Rows :** Even though all entries appear non-null, there may still be duplicate entries that should be removed to prevent skewing our analysis. 2. **Correcting Data Types :** Some columns such as creation\_time, end\_time, and time should ideally be in datetime format for any time series analysis or operations that involve time intervals. 3. **Standardize Text Data :** Ensuring consistency in how text data is formatted can be important, particularly if you're going to perform text-based operations or integrations.   The data has been cleaned with the following steps implemented:   1. **Duplicate Rows :** No duplicate rows were found, so the dataset remains with 282 entries. 2. **Data Types :** The creation\_time, end\_time, and time columns have been successfully converted to datetime format, which is more appropriate for any operations involving time. 3. **Text Data Standardization :** The src\_ip\_country\_code has been standardized to uppercase to ensure consistency across this field.   Handling Missing Data  In [3]: |

|  |
| --- |
| *# Remove duplicate rows* df\_unique = data.drop\_duplicates()  *# Convert time-related columns to datetime format* df\_unique['creation\_time'] = pd.to\_datetime(df\_unique['creation\_time']) df\_unique['end\_time'] = pd.to\_datetime(df\_unique['end\_time']) df\_unique['time'] = pd.to\_datetime(df\_unique['time']) *# Standardize text data (example: convert to lower case)* df\_unique['src\_ip\_country\_code'] = df\_unique['src\_ip\_country\_code'].str.upper() *# Ensuring country codes are all upper case*  *# Display changes and current state of the DataFrame* print("Unique Datasets Information:") df\_unique.info()  Unique Datasets Information:  <class 'pandas.core.frame.DataFrame'>  RangeIndex: 282 entries, 0 to 281 Data columns (total 16 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----   1. bytes\_in 282 non-null int64 2. bytes\_out 282 non-null int64 3. creation\_time 282 non-null datetime64[ns, UTC] |

|  |
| --- |
| 1. end\_time 282 non-null datetime64[ns, UTC] 2. src\_ip 282 non-null object 3. src\_ip\_country\_code 282 non-null object 4. protocol 282 non-null object 5. response.code 282 non-null int64 6. dst\_port 282 non-null int64 7. dst\_ip 282 non-null object 8. rule\_names 282 non-null object 9. observation\_name 282 non-null object 10. source.meta 282 non-null object 11. source.name 282 non-null object 12. time 282 non-null datetime64[ns, UTC] 13. detection\_types 282 non-null object   dtypes: datetime64[ns, UTC](3), int64(4), object(9) memory usage: 35.4+ KB  In [4]: print("Top 5 Unique Datasets Information:") df\_unique.head()  Top 5 Unique Datasets Information: |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[4]: | | | | | | | | | | | | | | | | | |
|  |  | b y t  e s \_ i  n | b  yt  e s \_ o  ut | cre atio n\_ti  me | en  d\_ti  me | src \_ip | src\_i p\_co untry \_cod e | p r  o  t  o c o  l | res po ns  e.c od e | d s  t  \_ p o rt | dst \_ip | rul e\_ na  m  es | obs erva tion  \_na me | sou rce. met a | sou rce. na me | tim e | det ecti  on  \_ty pes |
| 0 | 5 6 0 2 | 1 2 9 9 0 | 20  24-  04-  25  23:  00:  00  +0  0:0  0 | 20  24-  04-  25  23:  10:  00  +0  0:0  0 | 147  .16  1.1 61.  82 | AE | H  T T  P S | 20  0 | 4 4 3 | 10  .1  38  .6 9.  97 | S  us  pi ci  ou s  W  eb  Tr  aff ic | Adv ersa ry Infra stru ctur e  Inter acti  on | AW  S\_  VP  C\_  Flo  w | pro d\_ we bse rve  r | 20  24-  04-  25  23:  00:  00  +0  0:0  0 | waf \_ru le |
| 1 | 3  0 | 1  8 | 20 24- | 20 24- | 165  .22 | US | H  T | 20  0 | 4  4 | 10  .1 | S  us | Adv ersa | AW  S\_ | pro d\_ | 20 24- | waf  \_ru |
|  | | | | | | | | | | | | | | | | |

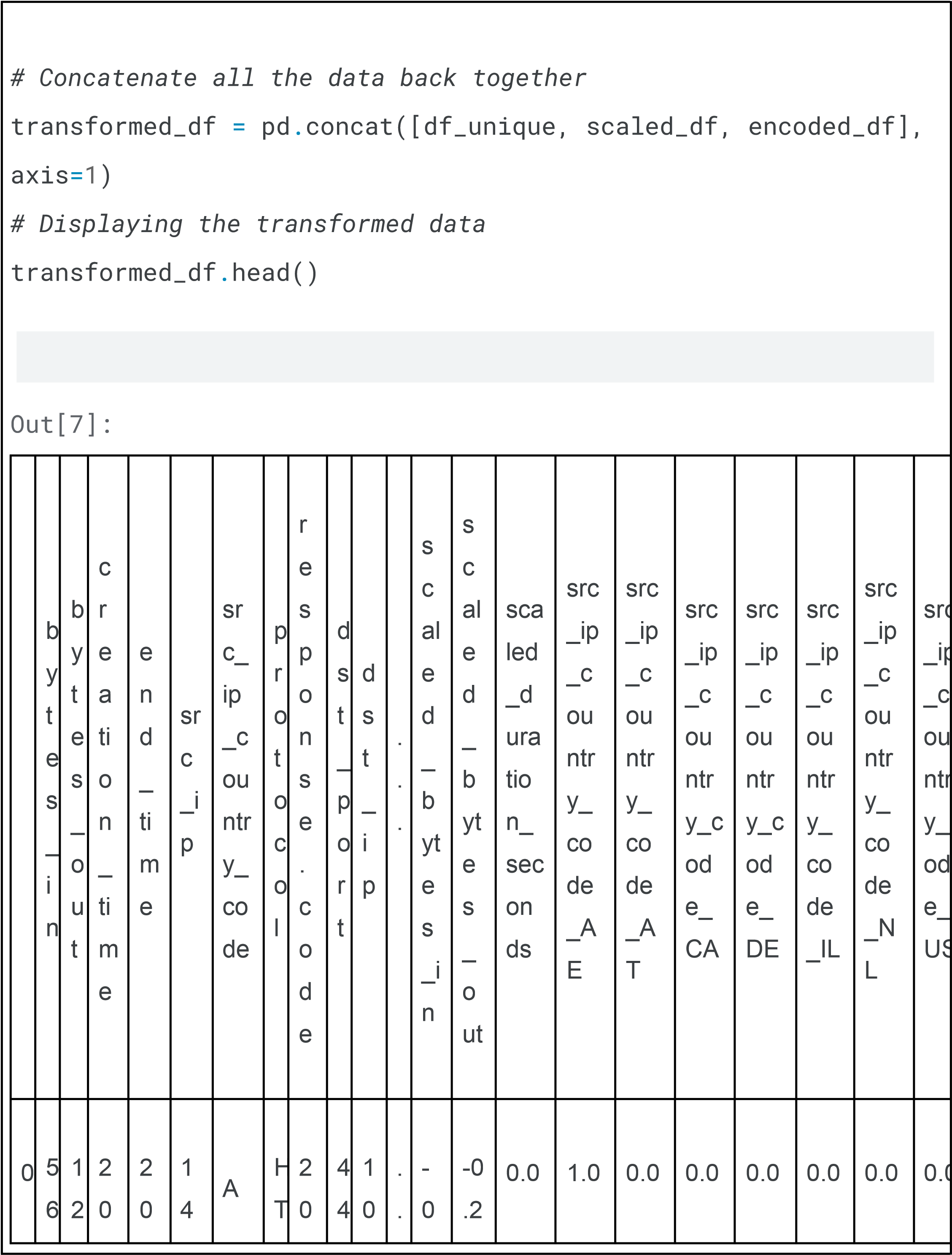
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | |
|  |  | 9  1 2 | 1  8 6 | 04-  25  23:  00:  00  +0  0:0  0 | 04-  25  23:  10:  00  +0  0:0  0 | 5.3  3.6 |  | T  P S |  | 3 | 38  .6 9.  97 | pi ci  ou s  W  eb  Tr  aff ic | ry Infra stru ctur e  Inter acti  on | VP  C\_  Flo  w | we bse rve  r | 04-  25  23:  00:  00  +0  0:0  0 | le |
| 2 | 2 8 5 0 6 | 1 3 4 6 8 | 20  24-  04-  25  23:  00:  00  +0  0:0  0 | 20  24-  04-  25  23:  10:  00  +0  0:0  0 | 165  .22  5.2 12.  255 | CA | H  T T  P S | 20  0 | 4 4 3 | 10  .1  38  .6 9.  97 | S  us  pi ci  ou s  W  eb  Tr  aff ic | Adv ersa ry Infra stru ctur e  Inter acti  on | AW  S\_  VP  C\_  Flo  w | pro d\_ we bse rve  r | 20  24-  04-  25  23:  00:  00  +0  0:0  0 | waf \_ru le |
| 3 | 3 0 5 4 | 1 4 2 7 | 20  24-  04-  25 | 20  24-  04-  25 | 136  .22 6.6  4.1 | US | H  T T  P | 20  0 | 4 4 3 | 10  .1  38  .6 | S  us  pi ci | Adv ersa ry  Infra | AW  S\_  VP  C\_ | pro d\_ we bse | 20  24-  04-  25 | waf \_ru le |
|  | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | |
|  |  | 6 | 8 | 23:  00:  00  +0  0:0  0 | 23:  10:  00  +0  0:0  0 | 14 |  | S |  |  | 9.  97 | ou s  W  eb  Tr  aff ic | stru ctur e  Inter acti  on | Flo  w | rve  r | 23:  00:  00  +0  0:0  0 |  |
| 4 | 6 5 2 6 | 1 3 8 9 2 | 20  24-  04-  25  23:  00:  00  +0  0:0  0 | 20  24-  04-  25  23:  10:  00  +0  0:0  0 | 165  .22  5.2 40.  79 | NL | H  T T  P S | 20  0 | 4 4 3 | 10  .1  38  .6 9.  97 | S  us pi ci  ou s  W  eb  Tr  aff ic | Adv ersa ry Infra stru ctur e  Inter acti  on | AW  S\_  VP  C\_  Flo  w | pro d\_ we bse rve  r | 20  24-  04-  25  23:  00:  00  +0  0:0  0 | waf \_ru le |
| Data Transformation  When it comes to preparing our dataset for machine learning models, one of the most important steps is data transformation. This phase helps to standardize or normalize the data, which in turn makes it simpler for the models to learn and generate correct predictions. Listed below are some of the more typical methods of data transformation that you could use: | | | | | | | | | | | | | | | | |

|  |
| --- |
| 1. 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.  1. 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.  1. 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.   Applying These Transformations |

|  |
| --- |
| Now will try to apply some of these transformations to our dataset:   1. Scale the bytes\_in and bytes\_out columns using Standardization. 2. One-hot encode the src\_ip\_country\_code column since it is a categorical   feature.   1. Feature engineering example : Create a new feature that measures the duration of the connection based on creation\_time and end\_time.   Now we will start with these transformations.   1. Scaling : The bytes\_in, bytes\_out, and the newly created duration\_seconds (which captures the duration of the connection) columns have been standardized using z-score scaling. This means their mean is now 0 and standard deviation is 1, which helps in normalizing the data for better performance of many machine learning algorithms. 2. One-Hot Encoding : The src\_ip\_country\_code column has been one-hot encoded. This has transformed each country code into its own feature, allowing categorical data to be used effectively in machine learning models. 3. Feature Engineering : A new feature duration\_seconds was added to measure the duration of each web session.   In [5]:  *# Feature engineering: Calculate duration of connection* df\_unique['duration\_seconds'] = (df\_unique['end\_time'] df\_unique['creation\_time']).dt.total\_seconds()  *# Preparing column transformations* |

|  |
| --- |
| *# StandardScaler for numerical features* scaler = StandardScaler() scaled\_features = scaler.fit\_transform(df\_unique[['bytes\_in', 'bytes\_out', 'duration\_seconds']])  In [6]:  *# OneHotEncoder for categorical features* encoder = OneHotEncoder(sparse=False) encoded\_features = encoder.fit\_transform(df\_unique[['src\_ip\_country\_code']])  *# Combining transformed features back into the DataFrame* scaled\_columns = ['scaled\_bytes\_in', 'scaled\_bytes\_out',  'scaled\_duration\_seconds'] encoded\_columns =  encoder.get\_feature\_names\_out(['src\_ip\_country\_code'])  In [7]:  *# Convert numpy arrays back to DataFrame* scaled\_df = pd.DataFrame(scaled\_features, columns=scaled\_columns, index=df\_unique.index) encoded\_df = pd.DataFrame(encoded\_features, columns=encoded\_columns, index=df\_unique.index) |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | | | | | | |
|  |  | 0  2 | 9  9 0 | 2  4  -  0  4  -  2 5  2  3  :  0  0  :  0 0 +  0  0  :  0 0 | 2  4  -  0  4  -  2 5  2  3  :  1  0  :  0 0 +  0  0  :  0 0 | 7.  1 6  1.  1 6  1.  8 2 | E | T  P S | 0 | 3 | .  1 3  8  .  6  9  .  9 7 | . | .  2 8 8  2 1 9 | 8  1 2 2 3 |  |  |  |  |  |  |  |  |
| 1 | 3 0 9 1 | 1 8 1 8 | 2 0 2 4 | 2 0 2 4 | 1 6  5.  2 | U  S | H  T T  P | 2 0 0 | 4 4 3 | 1  0  .  1 | .  .  . | -  0  .  2 | -0  .2  6 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |  |
|  | | | | | | | | | | | | | | | | | | | | | |

1.0

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | | | | | | |
|  |  | 2 | 6 | -  0  4  -  2 5  2  3  :  0  0  :  0 0 +  0  0  :  0 0 | -  0  4  -  2 5  2  3  :  1  0  :  0 0 +  0  0  :  0 0 | 2  5.  3  3.  6 |  | S |  |  | 3  8  .  6  9  .  9 7 |  | 8  2 1 0 8 | 8  0 4 |  |  |  |  |  |  |  |  |
| 2 | 2 8 5 0 6 | 1 3 4 6 8 | 2 0 2  4  -  0 | 2 0 2  4  -  0 | 1. 6   5.   1. 2   5. | C  A | H  T T  P S | 2 0 0 | 4 4 3 | 1  0  .  1 3 8 | .  .  . | -  0  .  2 8 2 | -0  .2  7 9 3 4 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |  |
|  | | | | | | | | | | | | | | | | | | | | | |

0.0

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | | | | | | |
|  |  |  |  | 4  -  2 5  2  3  :  0  0  :  0 0 +  0  0  :  0 0 | 4  -  2 5  2  3  :  1  0  :  0 0 +  0  0  :  0 0 | 2  1  2.  2  5 5 |  |  |  |  | .  6  9  .  9 7 |  | 6  8 9 | 4 |  |  |  |  |  |  |  |  |
| 3 | 3 0 5 4 6 | 1 4 2 7 8 | 2 0 2  4  -  0  4  - | 2 0 2  4  -  0  4  - | 1. 3   6.   1. 2   6.  6  4. | U  S | H  T T  P S | 2 0 0 | 4 4 3 | 1  0  .  1 3  8  .  6 | .  .  . | -  0  .  2 8 2 1 9 | -0  .2  7 6 1 6 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |  |
|  | | | | | | | | | | | | | | | | | | | | | |

1.0

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | | | | | | |
|  |  |  |  | 2  5 2  3  :  0  0  :  0  0 +  0  0  :  0 0 | 2  5 2  3  :  1  0  :  0  0 +  0  0  :  0 0 | 1  1 4 |  |  |  |  | 9  .  9 7 |  | 7 |  |  |  |  |  |  |  |  |  |
| 4 | 6 5 2 6 | 1 3 8 9 2 | 2 0 2  4  -  0  4  -  2 5 | 2 0 2  4  -  0  4  -  2 5 | 1. 6   5.   1. 2   5.  2 4  0.  7 | N  L | H  T T  P S | 2 0 0 | 4 4 3 | 1  0  .  1 3  8  .  6  9  . | .  .  . | -  0  .  2 8 7 9 9 6 | -0  .2  7 7 6 7 8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |  |
|  | | | | | | | | | | | | | | | | | | | | | |

0.0

2

3

:

0

0

:

0

0

+

0

0

:

0

0

2

3

:

1

0

:

0

0

+

0

0

:

0

0

9

9

7

5

rows

×

27

columns

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:

1.

**Descriptive**

**Statistics**

**:**

This

includes

mean,

median,

mode,

min,

max,

range,

quartiles,

and

standard

deviations.

2.

**Correlation**

**Analysis**

**:**

To

investigate

the

relationships

between

numerical

features

and

how

they

relate

to

each

other.

|  |
| --- |
| 3. **Distribution Analysis :** Examine the distribution of key features using histograms and box plots to identify the spread and presence of outliers.  **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.   In [8]:  *# Compute correlation matrix for numeric columns only* numeric\_df = transformed\_df.select\_dtypes(include=['float64',  'int64']) correlation\_matrix\_numeric = numeric\_df.corr()  *# Display the correlation matrix* correlation\_matrix\_numeric |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[8]:   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | b y  t  e s \_ i  n | b y t  e s \_ o u  t | re s p o n s  e. c o d e | d s  t  \_ p o r t | du rat io  n\_ se co nd s | sc  al  ed \_b yt  es  \_i  n | sc  al  ed \_b yt  es \_o  ut | scal ed\_ dura tion  \_se con ds | src\_ ip\_c ount ry\_c ode  \_AE | src\_ ip\_c ount ry\_c ode  \_AT | src\_ ip\_c ount ry\_c ode  \_CA | src\_ ip\_c ount ry\_c ode  \_DE | src \_ip \_co untr y\_c ode  \_IL | src\_ ip\_c ount ry\_c ode  \_NL | src\_ ip\_c ount ry\_c ode  \_US | | byte s\_in | 1  .  0 0 0 0 0 0 | 0  .  9 9 7 7 0 5 | N  a  N | N  a  N | Na  N | 1.  00 00 00 | 0.  99 77 05 | NaN | -0.0  705  59 | -0.0  816  70 | -0.1  664  88 | -0.0  953  33 | -0.0  659  39 | -0.0  068  27 | 0.31  601  5 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | byte s\_o  ut | 0  .  9 9 7  7  0 5 | 1  .  0 0 0  0  0 0 | N a  N | N a  N | Na  N | 0.  99 77 05 | 1.  00 00 00 | NaN | -0.0  724  52 | -0.0  817  77 | -0.1  595  87 | -0.0  900  01 | -0.0  676  30 | -0.0  456  41 | 0.32 768  3 | | resp ons  e.co de | N  a  N | N  a  N | N  a  N | N  a  N | Na  N | N  a  N | N  a  N | NaN | NaN | Na  N | NaN | NaN | Na  N | NaN | NaN | | dst\_ port | N  a  N | N  a  N | N  a  N | N  a  N | Na  N | N  a  N | N  a  N | NaN | NaN | Na  N | NaN | NaN | Na  N | NaN | NaN | | dura tion  \_se con ds | N  a  N | N  a  N | N  a  N | N  a  N | Na  N | N  a  N | N  a  N | NaN | NaN | Na  N | NaN | NaN | Na  N | NaN | NaN | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | scal ed\_ byte s\_in | 1  .  0 0 0  0  0 0 | 0  .  9 9 7  7  0 5 | N a  N | N a  N | Na  N | 1.  00 00 00 | 0.  99 77 05 | NaN | -0.0  705  59 | -0.0  816  70 | -0.1  664  88 | -0.0  953  33 | -0.0  659  39 | -0.0  068  27 | 0.31 601  5 | | scal ed\_ byte s\_o  ut | 0  .  9 9 7 7 0 5 | 1  .  0 0 0 0 0 0 | N  a  N | N  a  N | Na  N | 0.  99 77 05 | 1.  00 00 00 | NaN | -0.0  724  52 | -0.0  817  77 | -0.1  595  87 | -0.0  900  01 | -0.0  676  30 | -0.0  456  41 | 0.32  768  3 | | scal ed\_ dura tion  \_se con | N  a  N | N  a  N | N  a  N | N  a  N | Na  N | N  a  N | N  a  N | NaN | NaN | Na  N | NaN | NaN | Na  N | NaN | NaN | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ds |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | src\_ ip\_c ount ry\_c ode  \_AE | -  0  .  0  7  0 5 5 9 | -  0  .  0  7  2 4 5 2 | N a  N | N a  N | Na  N | -0  .0  70  55  9 | -0.  07 24 52 | NaN | 1.00 000  0 | -0.0  695  68 | -0.1  436  07 | -0.0  814  29 | -0.0  560  55 | -0.0  640  40 | -0.2  005  46 | | src\_ ip\_c ount ry\_c ode  \_AT | -  0  .  0 8 1 6 7 0 | -  0  .  0 8 1 7 7 7 | N  a  N | N  a  N | Na  N | -0  .0  81  67  0 | -0.  08 17 77 | NaN | -0.0  695  68 | 1.00  000  0 | -0.1  660  91 | -0.0  941  78 | -0.0  648  31 | -0.0  740  67 | -0.2  319  45 | | src\_ ip\_c ount | -  0  . | -  0  . | N  a | N  a | Na  N | -0  .1  66 | -0.  15 95 | NaN | -0.1  436 | -0.1  660 | 1.00  000 | -0.1  944 | -0.1  338 | -0.1  528 | -0.4  787 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ry\_c ode  \_CA | 1 6 6 4 8 8 | 1 5 9 5 8 7 | N | N |  | 48  8 | 87 |  | 07 | 91 | 0 | 10 | 30 | 94 | 98 | | src\_ ip\_c ount ry\_c ode  \_DE | -  0  .  0 9 5 3 3 3 | -  0  .  0 9 0 0 0 1 | N  a  N | N  a  N | Na  N | -0  .0  95  33  3 | -0.  09 00 01 | NaN | -0.0  814  29 | -0.0  941  78 | -0.1  944  10 | 1.00  000  0 | -0.0  758  85 | -0.0  866  95 | -0.2  714  93 | | src\_ ip\_c ount ry\_c ode  \_IL | -  0  .  0 6 5 9 3 9 | -  0  .  0 6 7 6 3 0 | N  a  N | N  a  N | Na  N | -0  .0  65  93  9 | -0.  06 76 30 | NaN | -0.0  560  55 | -0.0  648  31 | -0.1  338  30 | -0.0  758  85 | 1.0  000  00 | -0.0  596  80 | -0.1  868  93 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | src\_ ip\_c ount ry\_c ode  \_NL | -  0  .  0 0  6  8 2 7 | -  0  .  0 4  5  6 4 1 | N  a  N | N  a  N | Na  N | -0  .0  06  82  7 | -0.  04  56 41 | NaN | -0.0  640  40 | -0.0  740  67 | -0.1  528  94 | -0.0  866  95 | -0.0  596  80 | 1.00  000  0 | -0.2  135  16 | | src\_ ip\_c ount ry\_c ode  \_US | 0  .  3 1 6 0 1 5 | 0  .  3 2 7 6 8 3 | N  a  N | N  a  N | Na  N | 0.  31 60 15 | 0.  32 76 83 | NaN | -0.2  005  46 | -0.2  319  45 | -0.4  787  98 | -0.2  714  93 | -0.1  868  93 | -0.2  135  16 | 1.00  000  0 |   In [9]:  *# Heatmap for the correlation matrix* plt.figure(figsize=(10, 8)) sns.heatmap(correlation\_matrix\_numeric, annot=True, fmt=".2f", cmap='coolwarm') plt.title('Correlation Matrix Heatmap') |

plt

.

show()

In

[10]:

*#*

*Stacked*

*Bar*

*Chart*

*for*

*Detection*

*Types*

*by*

*Country*

*#*

*Preparing*

*data*

*for*

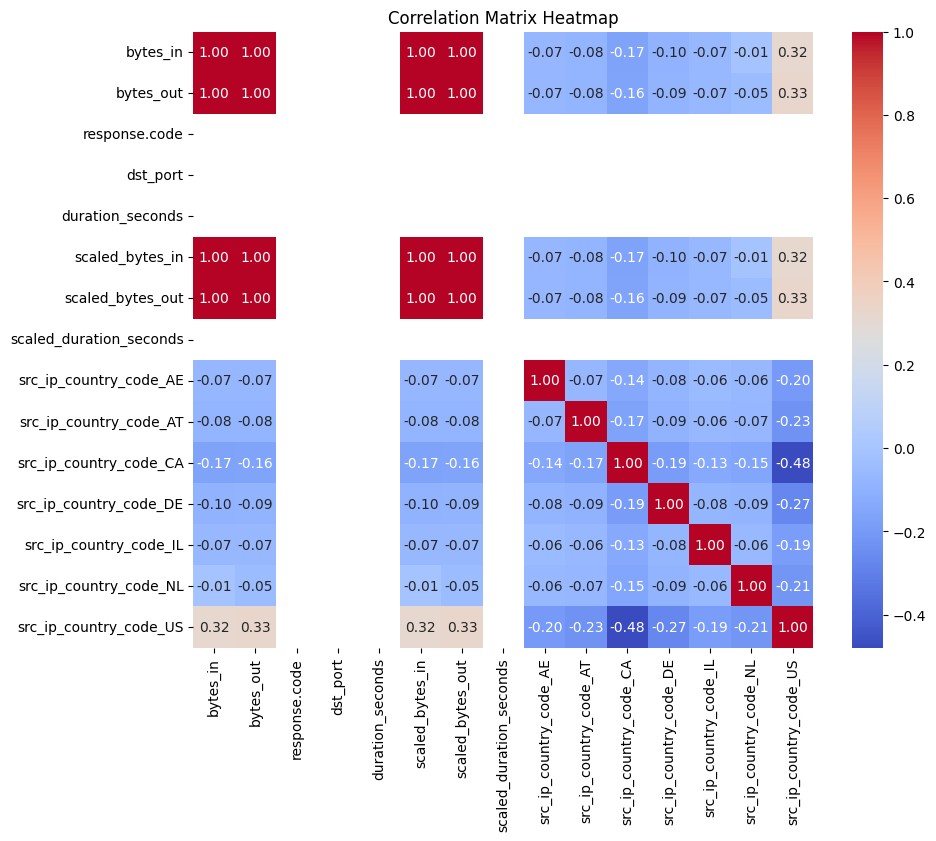
*stacked*

*bar*

*chart*

detection\_types\_by\_country

=



pd

.

crosstab(transformed\_df[

'src\_ip\_country\_code'

]

,

transformed\_df[

'detection\_types'

])

detection\_types\_by\_country

.

plot(kind

=

'bar'

,

stacked

=

True

,

figsize

=

(

12

,

6

))

plt

.

title(

'Detection

Types

by

Country

Code'

)

plt

.

xlabel(

'Country

Code'

)

plt

.

ylabel(

'Frequency

of

Detection

Types'

)

plt

.

xticks(rotation

=

45

)

plt

.

legend(title

=

'Detection

Type'

)

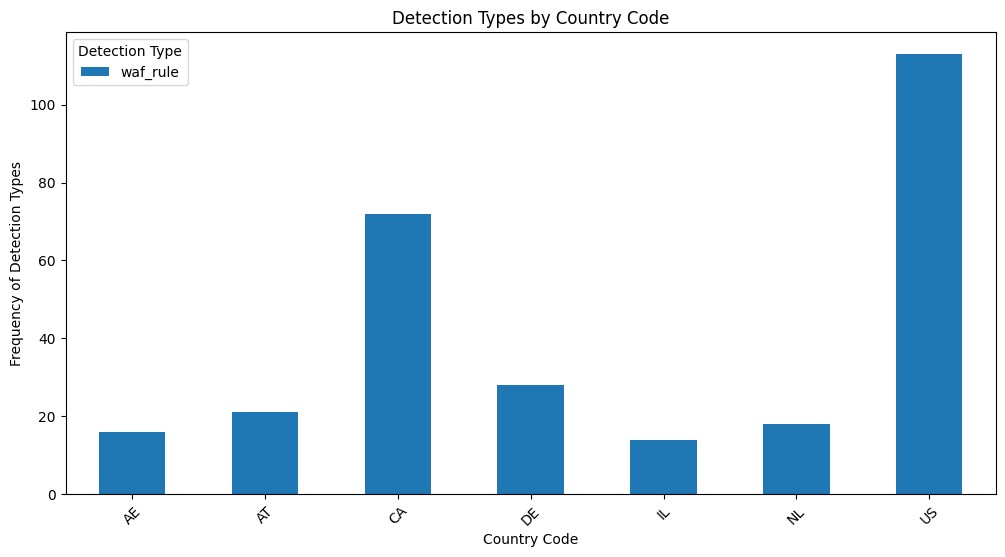
plt

.

show()

In

[11]:



|  |
| --- |
| *# Convert 'creation\_time' to datetime format* data['creation\_time'] = pd.to\_datetime(data['creation\_time'])  *# Set 'creation\_time' as the index* data.set\_index('creation\_time', inplace=True)  *# Plotting* plt.figure(figsize=(12, 6)) plt.plot(data.index, data['bytes\_in'], label='Bytes In', marker='o') plt.plot(data.index, data['bytes\_out'], label='Bytes Out', marker='o') plt.title('Web Traffic Analysis Over Time') plt.xlabel('Time') plt.ylabel('Bytes') plt.legend() plt.grid(True) plt.xticks(rotation=45) plt.tight\_layout()  *# Show the plot* plt.show() |

In

[12]:

*#*

*Create*

*a*

*graph*

G

=

nx

.

Graph()

*#*

*Add*

*edges*

*from*

*source*

*IP*

*to*

*destination*

*IP*

for

idx,

row

**in**

data

.

iterrows():

G

.

add\_edge(row[

'src\_ip'

]

,

row[

'dst\_ip'

])

*#*

*Draw*

*the*

*network*

*graph*

plt

.

figure(figsize

=

(

14

,

10

))

nx

.

draw\_networkx(G,

with\_labels

=

True

,

node\_size

=

20

,

font\_size

=

8

,

node\_color

=

'skyblue'

,

font\_color

=

'darkblue'

)

plt

.

title(

'Network

Interaction

between

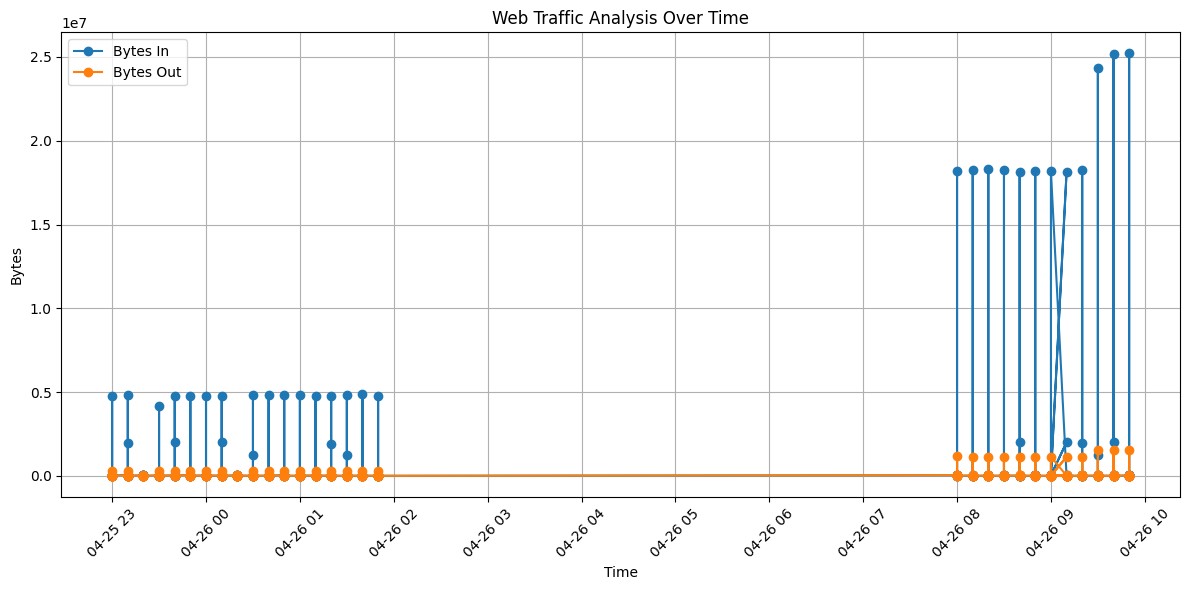
Source

and

Destination

IPs'

)



plt

.

axis(

'off'

)

*#*

*Turn*

*off*

*the*

*axis*

*#*

*Show*

*the*

*plot*

plt

.

show()

RandomForestClassifier

In

[13]:

*#*

*First,*

*encode*

*this*

*column*

*into*

*binary*

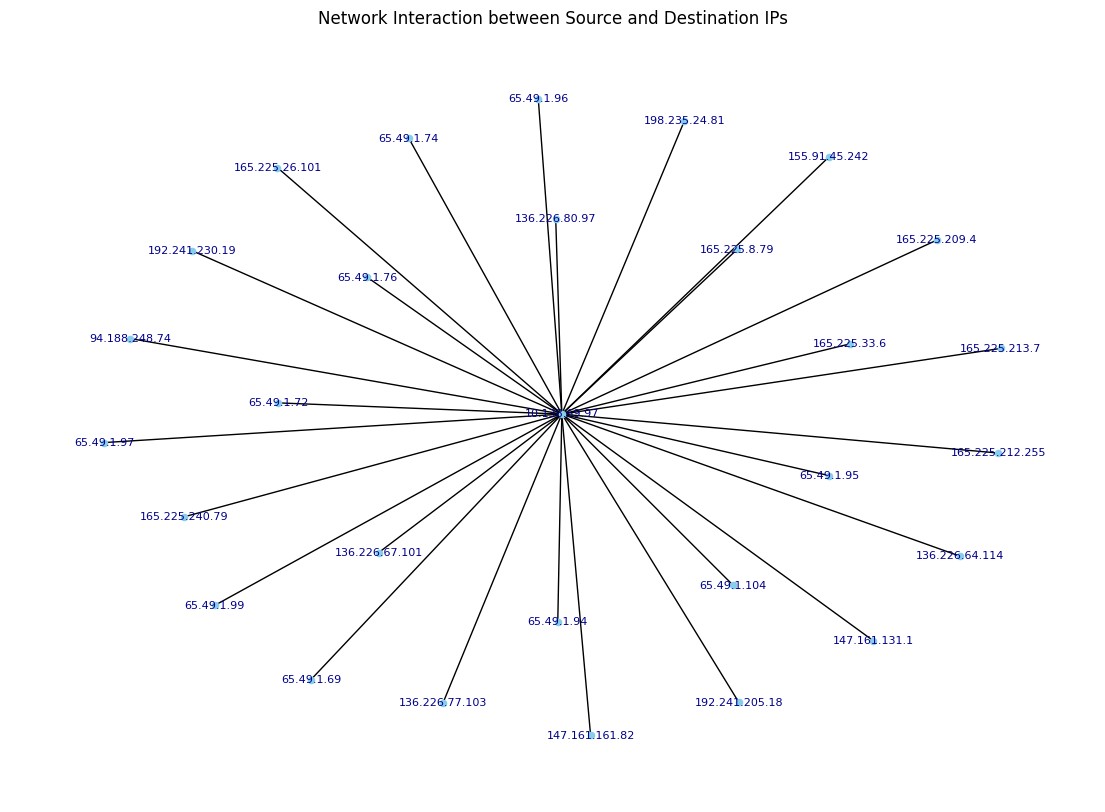
*labels*

transformed\_df[

'is\_suspicious'

]

=



|  |
| --- |
| (transformed\_df['detection\_types'] == 'waf\_rule').astype(int)  *# Features and Labels*  X = transformed\_df[['bytes\_in', 'bytes\_out', 'scaled\_duration\_seconds']] *# Numeric features* y = transformed\_df['is\_suspicious'] *# Binary labels*  In [14]:  *# Split the data into training and test sets*  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  *# Initialize the Random Forest Classifier* rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)  *# Train the model* rf\_classifier.fit(X\_train, y\_train)  *# Predict on the test set*  y\_pred = rf\_classifier.predict(X\_test)  In [15]: |

*# Evaluate the model* accuracy = accuracy\_score(y\_test, y\_pred) classification = classification\_report(y\_test, y\_pred)

In [16]:

print("Model Accuracy: ",accuracy)

Model Accuracy: 1.0

In [17]: print("Classification Report: ",classification)

Classification Report: precision recall

f1-score support

1 1.00 1.00 1.00 85

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

|  |
| --- |
| Neural Network  In [18]:  data['is\_suspicious'] = (data['detection\_types'] == 'waf\_rule').astype(int)  *# Features and labels*  X = data[['bytes\_in', 'bytes\_out']].values *# Using only numeric features* y = data['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* history = model.fit(X\_train\_scaled, y\_train, epochs=10, batch\_size=8, verbose=1)  *# Evaluate the model* loss, accuracy = model.evaluate(X\_test\_scaled, y\_test) print(f"Test Accuracy: **{**accuracy\*100**:**.2f**}**%")  Epoch 1/10  **25/25** ━━━━━━━━━━━━━━━━━━━━ **2s** 2ms/step -  accuracy: 1.0000 - loss: 0.5825  Epoch 2/10  **25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step -  accuracy: 1.0000 - loss: 0.5093  Epoch 3/10  **25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - |

|  |
| --- |
| accuracy: 1.0000 - loss: 0.4409  Epoch 4/10  **25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step -  accuracy: 1.0000 - loss: 0.3579  Epoch 5/10  **25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step -  accuracy: 1.0000 - loss: 0.2755  Epoch 6/10  **25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step -  accuracy: 1.0000 - loss: 0.2074  Epoch 7/10  **25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step -  accuracy: 1.0000 - loss: 0.1354  Epoch 8/10  **25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step -  accuracy: 1.0000 - loss: 0.0840  Epoch 9/10  **25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step -  accuracy: 1.0000 - loss: 0.0498  Epoch 10/10  **25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step -  accuracy: 1.0000 - loss: 0.0323  **3/3** ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy:  1.0000 - loss: 0.0237  Test Accuracy: 100.00% |

In [19]:

*# 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* history = model.fit(X\_train\_scaled, y\_train, epochs=10, batch\_size=32, verbose=1, validation\_split=0.2)

*# Evaluate the model* loss, accuracy = model.evaluate(X\_test\_scaled, y\_test) print(f"Test Accuracy: **{**accuracy\*100**:**.2f**}**%")

|  |
| --- |
| *# Plotting the training history* plt.figure(figsize=(12, 6)) 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()  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.show()  Epoch 1/10  **5/5** ━━━━━━━━━━━━━━━━━━━━ **2s** 59ms/step - accuracy: |

0.7806 - loss: 0.6534 - val\_accuracy: 1.0000 - val\_loss: 0.5717 Epoch 2/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 0.9870 - loss: 0.5804 - val\_accuracy: 1.0000 - val\_loss: 0.4919 Epoch 3/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 1.0000 - loss: 0.5095 - val\_accuracy: 1.0000 - val\_loss: 0.4191 Epoch 4/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 1.0000 - loss: 0.4369 - val\_accuracy: 1.0000 - val\_loss: 0.3445 Epoch 5/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 1.0000 - loss: 0.3474 - val\_accuracy: 1.0000 - val\_loss: 0.2689 Epoch 6/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 1.0000 - loss: 0.2784 - val\_accuracy: 1.0000 - val\_loss: 0.1975 Epoch 7/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - accuracy: 1.0000 - loss: 0.2130 - val\_accuracy: 1.0000 - val\_loss: 0.1360 Epoch 8/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 1.0000 - loss: 0.1526 - val\_accuracy: 1.0000 - val\_loss: 0.0882 Epoch 9/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - accuracy:

1.0000 - loss: 0.0989 - val\_accuracy: 1.0000 - val\_loss: 0.0550

Epoch

10

/

10

**5**

**/**

**5**

━━━━━━━━━━━━━━━━━━━━

**0**

**s**

11

ms/step

-

accuracy:

1.0000

-

loss:

0.0629

-

val\_accuracy:

1.0000

-

val\_loss:

0.0341

**3**

**/**

**3**

━━━━━━━━━━━━━━━━━━━━

**0**

**s**

4

ms/step

-

accuracy:

1.0000

-

loss:

0.0393

Test

Accuracy:

100.00

%

In

[20]:

scaler

=

StandardScaler()

X\_train\_scaled

=

scaler

.

fit\_transform(X\_train

.

reshape(

-

1

,

X\_train

.

shape[

-

1

]))

.

reshape(X\_train

.

shape)

X\_test\_scaled

=

scaler

.

transform(X\_test

.

reshape(

-

1

,

X\_test

.

shape[

-

1

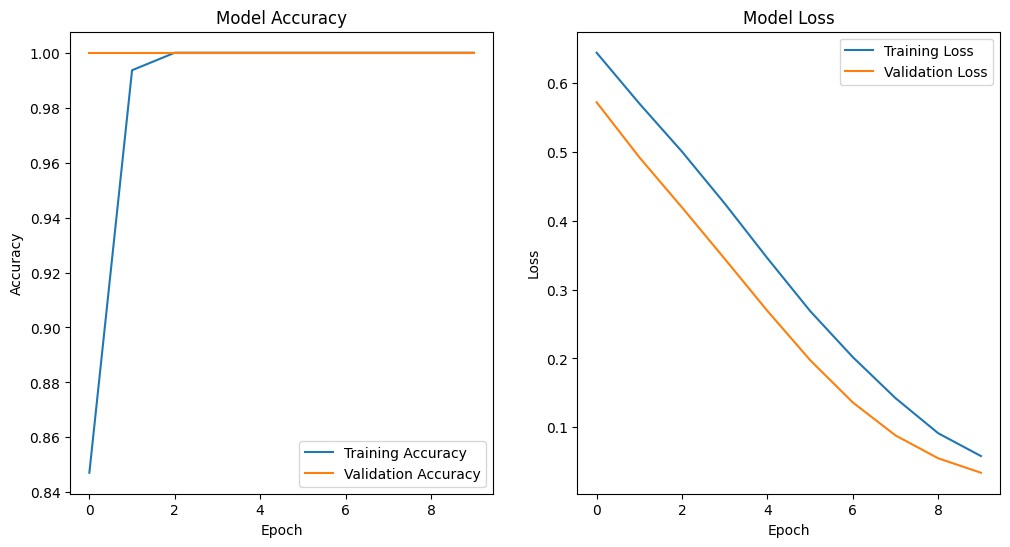
]))

.

reshape(X\_test

.

shape)



*# 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* history = model.fit(X\_train\_scaled, y\_train, epochs=10, batch\_size=32, verbose=1, validation\_split=0.2)

*# Evaluate the model* loss, accuracy = model.evaluate(X\_test\_scaled, y\_test) print(f"Test Accuracy: **{**accuracy\*100**:**.2f**}**%")

*# Plotting the training history* plt.figure(figsize=(12, 6))

|  |
| --- |
| 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()  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.show()  Epoch 1/10  **5/5** ━━━━━━━━━━━━━━━━━━━━ **2s** 64ms/step - accuracy: 0.7993 - loss: 0.6541 - val\_accuracy: 1.0000 - val\_loss: 0.5830  Epoch 2/10 |

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 1.0000 - loss: 0.6132 - val\_accuracy: 1.0000 - val\_loss: 0.5506 Epoch 3/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 1.0000 - loss: 0.5934 - val\_accuracy: 1.0000 - val\_loss: 0.5194 Epoch 4/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - accuracy: 1.0000 - loss: 0.5494 - val\_accuracy: 1.0000 - val\_loss: 0.4886 Epoch 5/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 1.0000 - loss: 0.5132 - val\_accuracy: 1.0000 - val\_loss: 0.4560 Epoch 6/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - accuracy: 1.0000 - loss: 0.4873 - val\_accuracy: 1.0000 - val\_loss: 0.4188 Epoch 7/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 1.0000 - loss: 0.4496 - val\_accuracy: 1.0000 - val\_loss: 0.3772 Epoch 8/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - accuracy: 1.0000 - loss: 0.4046 - val\_accuracy: 1.0000 - val\_loss: 0.3320 Epoch 9/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 1.0000 - loss: 0.3570 - val\_accuracy: 1.0000 - val\_loss: 0.2845 Epoch 10/10

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - accuracy:

1.0000

-

loss:

0.3042

-

val\_accuracy:

1.0000

-

val\_loss:

0.2370

**3**

**/**

**3**

━━━━━━━━━━━━━━━━━━━━

**s**

**0**

ms/step

4

-

accuracy:

1.0000

-

loss:

0.2563

Test

Accuracy:

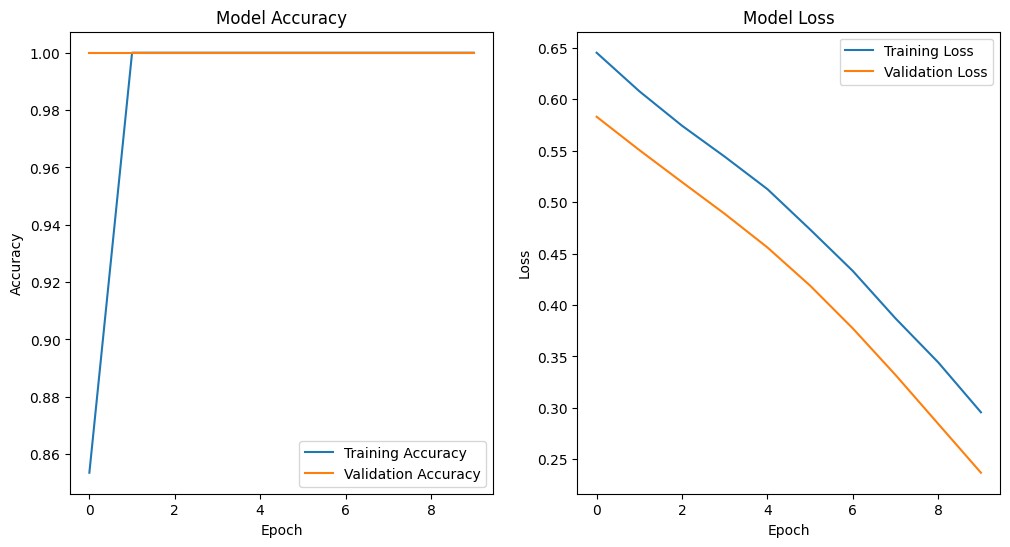
100.00

%

In

[

]:



[Reference](https://github.com/Tharunr0/Web-Threat-Analysis-Cyber-Security) [link](https://github.com/Tharunr0/Web-Threat-Analysis-Cyber-Security)