



✓ Cyber Security Attack

Attacks against cybersecurity cover a wide spectrum of malevolent actions meant to take advantage of weaknesses in data, networks, and computer systems. These assaults can come in many different forms, such as ransomware, phishing schemes, malware infections, denial-of-service (DoS) attacks, and social engineering techniques. Attackers frequently target financial assets, sensitive data, or activities to further their activism, espionage, or financial gain. Cybersecurity threats are constantly evolving due to the growing interconnection of digital systems and the proliferation of devices with Internet access, hence posing serious hazards to individuals, organisations, and governments throughout the globe. Strong security measures must be put in place, risk assessments must be carried out on a regular basis, awareness must be raised, and new threats must be watched out for.

✓ Key Data Points


 **Timestamp (39,997 entries):** Serving as the temporal framework, this allows for the comprehension of patterns throughout the temporal range.

 **IP addresses:** We can map the geographic epicentres of cyber activity by using entries that provide the source (40,000) and destination (40,000).

 **Ports:** Examining the 29,761 source and 29,895 destination ports reveals the services that are being targeted.

 **Protocol (3 types):** The palette of protocols, which includes TCP, UDP, and ICMP, provides hints about possible attack methods.

 **Packet Details:** Specifics of data communication are clarified by the nuances of packet length (1,437 types) and packet genre (2 types).

 **Traffic and Payload Information:** By distinguishing between three different traffic kinds and utilising payload insights (40,000 entries), it is possible to determine the likely cyber purpose.

🚩 **Attack Indicators:** The cyber threat landscape is mapped using metrics such as malware indices, anomaly scores (9,826 types), alerts, attack modalities (3 types), and unique signatures.

💡 **Reaction and Effect:** The counter-reaction plan and its possible effects are outlined by indicators like the countermeasures taken, the intensity of the threat, and intrusion detection/alerts.

🖥️ **User & Infrastructure Data:** A comprehensive picture of the cyber ecosystem is assembled by the data mosaic that includes user profiles (32,389 entries), device metrics (32,104 entries), network sections, geographic markers (8,723 types), proxy logs (20,148 entries), firewall narratives, and log sources.

✓ Importing Libraries

Downloading sketch module

```
pip install sketch
```

```
Collecting sketch
```

```
  Downloading sketch-0.5.2-py3-none-any.whl (16 kB)
```

```
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from sketch) (1.5.3)
```

```
Collecting datasketch>=1.5.8 (from sketch)
```

```
  Downloading datasketch-1.6.4-py3-none-any.whl (88 kB)
```

```
      ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88.3/88.3 kB 2.7 MB/s eta 0:00:00
```

```
Collecting datasketches>=4.0.0 (from sketch)
```

```
  Downloading datasketches-5.0.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (678 kB)
```

```
      ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 678.8/678.8 kB 18.2 MB/s eta 0:00:00
```

```
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages (from sketch) (7.34.0)
```

```
Collecting lambdaprompt>=0.6.1 (from sketch)
```

```
  Downloading lambdaprompt-0.6.1-py3-none-any.whl (14 kB)
```

```
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from sketch) (23.2)
```

```
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.10/dist-packages (from datasketch>=1.5.8->sketch) (1.25.2)
```

```
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from datasketch>=1.5.8->sketch) (1.11.0)
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from lambdaprompt>=0.6.1->sketch) (2.31.0)
```

```
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from lambdaprompt>=0.6.1->sketch) (3.9.3)
```

```
Collecting python-dotenv (from lambdaprompt>=0.6.1->sketch)
```

```

Downloading python_dotenv-1.0.1-py3-none-any.whl (19 kB)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from lambdaprompt>=0.6.1->sketch) (3.1.3)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages (from lambdaprompt>=0.6.1->sketch) (1.
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from lambdaprompt>=0.6.1->sketch) (6.0.1)
Requirement already satisfied: tenacity in /usr/local/lib/python3.10/dist-packages (from lambdaprompt>=0.6.1->sketch) (8.2.3)
Requirement already satisfied: pydantic in /usr/local/lib/python3.10/dist-packages (from lambdaprompt>=0.6.1->sketch) (2.6.3)
Collecting pydantic-settings (from lambdaprompt>=0.6.1->sketch)
  Downloading pydantic_settings-2.2.1-py3-none-any.whl (13 kB)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.3.0->sketch)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.3.0->sketch) (2023.4)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython->sketch) (67.7.2)
Collecting jedi>=0.16 (from ipython->sketch)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
    1.6/1.6 MB 50.5 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython->sketch) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython->sketch) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipython->sketch) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!<3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython->sketch) (2.16.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython->sketch) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython->sketch) (0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython->sketch) (4.9.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython->sketch)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython->sketch)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!<3.0.1,<3.1.0,
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas>=1.3.
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->lambdaprompt>=0.6.1
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->lambdaprompt>=0.6.1->s
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->lambdaprompt>=0.6.
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->lambdaprompt>=0.
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->lambdaprompt>=0.6.1->
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->lambdaprompt
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->lambdaprompt>=0.6.1->
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic->lambdaprompt
Requirement already satisfied: pydantic-core==2.16.3 in /usr/local/lib/python3.10/dist-packages (from pydantic->lambdaprompt>
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from pydantic->lambdaprom
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->lambdaprom
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->lambdaprompt>=0.6.1->s
Requirement already satisfied: urllib3<2,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->lambdaprompt>=0

```

Downloading dataprep module

pip install dataprep

```
Collecting dataprep
  Downloading dataprep-0.4.5-py3-none-any.whl (9.9 MB)
    _____ 9.9/9.9 MB 36.4 MB/s eta 0:00:00
Requirement already satisfied: aiohttp<4.0,>=3.6 in /usr/local/lib/python3.10/dist-packages (from dataprep) (3.9.3)
Collecting bokeh<3,>=2 (from dataprep)
  Downloading bokeh-2.4.3-py3-none-any.whl (18.5 MB)
    _____ 18.5/18.5 MB 52.6 MB/s eta 0:00:00
Requirement already satisfied: dask[array,dataframe,delayed]>=2022.3.0 in /usr/local/lib/python3.10/dist-packages (from dataprep) (2022.12.0)
Requirement already satisfied: flask<3,>=2 in /usr/local/lib/python3.10/dist-packages (from dataprep) (2.2.5)
Collecting flask_cors<4.0.0,>=3.0.10 (from dataprep)
  Downloading Flask_Cors-3.0.10-py2.py3-none-any.whl (14 kB)
Requirement already satisfied: ipywidgets<8.0,>=7.5 in /usr/local/lib/python3.10/dist-packages (from dataprep) (7.7.1)
Collecting jinja2<3.1,>=3.0 (from dataprep)
  Downloading Jinja2-3.0.3-py3-none-any.whl (133 kB)
    _____ 133.6/133.6 kB 11.4 MB/s eta 0:00:00
Collecting jsonpath-ng<2.0,>=1.5 (from dataprep)
  Downloading jsonpath_ng-1.6.1-py3-none-any.whl (29 kB)
Collecting metaphone<0.7,>=0.6 (from dataprep)
  Downloading Metaphone-0.6.tar.gz (14 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: nltk<4.0.0,>=3.6.7 in /usr/local/lib/python3.10/dist-packages (from dataprep) (3.8.1)
Requirement already satisfied: numpy<2.0,>=1.21 in /usr/local/lib/python3.10/dist-packages (from dataprep) (1.25.2)
Requirement already satisfied: pandas<2.0,>=1.1 in /usr/local/lib/python3.10/dist-packages (from dataprep) (1.5.3)
Collecting pydantic<2.0,>=1.6 (from dataprep)
  Downloading pydantic-1.10.14-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1 MB)
    _____ 3.1/3.1 MB 72.6 MB/s eta 0:00:00
Requirement already satisfied: pydot<2.0.0,>=1.4.2 in /usr/local/lib/python3.10/dist-packages (from dataprep) (1.4.2)
Collecting python-crfsuite==0.9.8 (from dataprep)
  Downloading python_crfsuite-0.9.8-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.0 MB)
    _____ 1.0/1.0 MB 59.9 MB/s eta 0:00:00
Collecting python-stdnum<2.0,>=1.16 (from dataprep)
  Downloading python_stdnum-1.19-py2.py3-none-any.whl (1.0 MB)
    _____ 1.0/1.0 MB 59.8 MB/s eta 0:00:00
Collecting rapidfuzz<3.0.0,>=2.1.2 (from dataprep)
  Downloading rapidfuzz-2.15.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.0 MB)
```

```

3.0/3.0 MB 76.1 MB/s eta 0:00:00
Collecting regex<2022.0.0,>=2021.8.3 (from dataprep)
  Downloading regex-2021.11.10-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (764 kB)
764.0/764.0 kB 10.1 MB/s eta 0:00:00
Requirement already satisfied: scipy<2.0,>=1.8 in /usr/local/lib/python3.10/dist-packages (from dataprep) (1.11.4)
Collecting sqlalchemy==1.3.24 (from dataprep)
  Downloading SQLAlchemy-1.3.24.tar.gz (6.4 MB)
6.4/6.4 MB 80.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: tqdm<5.0,>=4.48 in /usr/local/lib/python3.10/dist-packages (from dataprep) (4.66.2)
Collecting varname<0.9.0,>=0.8.1 (from dataprep)
  Downloading varname-0.8.3-py3-none-any.whl (21 kB)
Requirement already satisfied: wordcloud<2.0,>=1.8 in /usr/local/lib/python3.10/dist-packages (from dataprep) (1.9.3)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.6->dataprep)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.6->dataprep) (2)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.6->dataprep)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.6->dataprep)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.6->dataprep) (
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.6->da
Requirement already satisfied: packaging>=16.8 in /usr/local/lib/python3.10/dist-packages (from bokeh<3,>=2->dataprep) (23.2)
Requirement already satisfied: pillow>=7.1.0 in /usr/local/lib/python3.10/dist-packages (from bokeh<3,>=2->dataprep) (9.4.0)
Requirement already satisfied: PyYAML>=3.10 in /usr/local/lib/python3.10/dist-packages (from bokeh<3,>=2->dataprep) (6.0.1)

```

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sketch
from dataprep.eda import plot, plot_missing, plot_correlation

```

- Pandas: Python data manipulation and analysis are performed using the pandas package, which provides high-level data structures and methods for effective management of structured data.
- NumPy: Python's NumPy library, which offers robust arrays and methods for mathematical operations, is used for numerical computation.
- Matplotlib: A Python package called Matplotlib may be used to create static, interactive, and publication-quality visuals.

- Seaborn: A high-level interface for making visually appealing statistical visualisations is offered by the Python data visualisation toolkit Seaborn, which is built on top of Matplotlib.
- Sketch: Sketch is a Python package for vector graphics creation and manipulation that offers tools for producing and adjusting scaled pictures.
- dataprep.eda: A Python package called dataprep.eda is used for automated exploratory data analysis. It provides informative visualisations and statistical summaries to help with interpreting the data.

✓ Loading Dataset

```
df = pd.read_csv('/content/cybersecurity_attacks.csv', parse_dates=['Timestamp'], dtype={'Protocol':'category', 'Packet Type':'category'})
```

- Reading a CSV file called "cybersecurity_attacks.csv" that is stored in the "/content" directory using Python's pandas module.
- pd.read_csv loads the CSV file into a DataFrame in pandas.
- The 'Timestamp' column should be interpreted as datetime objects, according to the parse_dates argument.
- To optimise efficiency and conserve memory, the dtype argument can be used to define the data types for certain columns.

✓ Exploratory Data Analysis (EDA)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40000 entries, 0 to 39999
Data columns (total 25 columns):
 #   Column              Non-Null Count  Dtype  
---  -
 0   Timestamp           40000 non-null  datetime64[ns]
```

```
1 Source IP Address      40000 non-null object
2 Destination IP Address 40000 non-null object
3 Source Port            40000 non-null int64
4 Destination Port       40000 non-null int64
5 Protocol               40000 non-null category
6 Packet Length          40000 non-null int64
7 Packet Type            40000 non-null category
8 Traffic Type           40000 non-null category
9 Payload Data           40000 non-null object
10 Malware Indicators     20000 non-null object
11 Anomaly Scores         40000 non-null float64
12 Alerts/Warnings       19933 non-null object
13 Attack Type           40000 non-null category
14 Attack Signature       40000 non-null category
15 Action Taken           40000 non-null category
16 Severity Level        40000 non-null category
17 User Information       40000 non-null object
18 Device Information     40000 non-null object
19 Network Segment       40000 non-null category
20 Geo-location Data      40000 non-null object
21 Proxy Information      20149 non-null object
22 Firewall Logs          20039 non-null object
23 IDS/IPS Alerts         19950 non-null object
24 Log Source             40000 non-null category
dtypes: category(9), datetime64[ns](1), float64(1), int64(3), object(11)
memory usage: 5.2+ MB
```

It contains details on the number of rows and columns, the data types in each column, the number of values in each column that are not null, and how much RAM the DataFrame is using.

```
df.head()
```

	Timestamp	Source IP Address	Destination IP Address	Source Port	Destination Port	Protocol	Packet Length	Packet Type	Traffic Type	Payload Data	...	Action Taken	Severity
0	2023-05-30 06:33:58	103.216.15.12	84.9.164.252	31225	17616	ICMP	503	Data	HTTP	Qui natus odio asperiores nam. Optio nobis ius...	...	Logged	
1	2020-08-26 07:08:30	78.199.217.198	66.191.137.154	17245	48166	ICMP	1174	Data	HTTP	Aperiam quos modi officiis veritatis rem. Omni...	...	Blocked	
2	2022-11-13 08:23:25	63.79.210.48	198.219.82.17	16811	53600	UDP	306	Control	HTTP	Perferendis sapiente vitae soluta. Hic delectu...	...	Ignored	
3	2023-07-02 10:38:46	163.42.196.10	101.228.192.255	20018	32534	UDP	385	Data	HTTP	Totam maxime beatae expedita explicabo porro l...	...	Blocked	Me
4	2023-07-16 13:11:07	71.166.185.76	189.243.174.238	6131	26646	TCP	1462	Data	DNS	Odit nesciunt dolorem nisi iste iusto. Animi v...	...	Blocked	

5 rows × 25 columns

Displays the first 5 rows of the DataFrame.

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Source Port	40000.0	32970.356450	18560.425604	1027.0	16850.75	32856.000	48928.25	65530.0
Destination Port	40000.0	33150.868650	18574.668842	1024.0	17094.75	33004.500	49287.00	65535.0
Packet Length	40000.0	781.452725	416.044192	64.0	420.00	782.000	1143.00	1500.0
Anomaly Scores	40000.0	50.113473	28.853598	0.0	25.15	50.345	75.03	100.0

Provides descriptive statistics for each numerical column in the DataFrame, transposed for easier readability.

✓ Data Visualization

1. WordCloud

Importing WordCloud Library

```
from wordcloud import WordCloud
```

```
df['Payload Data'].dtype
```


2. Pie Chart

2.1. Distribution of Network Traffic Protocols: TCP, UDP & ICMP

```
df['Protocol'].value_counts()
```

```
ICMP    13429
UDP     13299
TCP     13272
Name: Protocol, dtype: int64
```

Counts the occurrences of 'Internet Control Message Protocol (ICMP), User Datagram Protocol (UDP), & Transmission Control Protocol (TCP)' in the 'Protocol' column of the DataFrame.

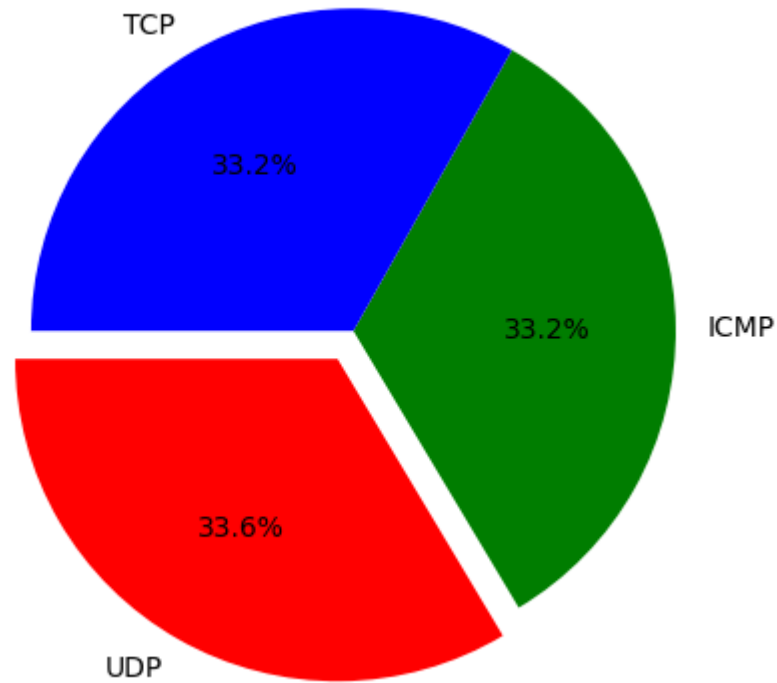
```
labels = ['UDP', 'ICMP', 'TCP']
sizes = df['Protocol'].value_counts() # Proportional sizes of each category
colors = ['red', 'green', 'blue'] # Color for each category segment
explode = (0.1, 0, 0) # Explode a slice if needed (0 means no explosion)

# Create a pie chart
plt.pie(sizes, labels=labels, colors=colors, explode=explode, autopct='%1.1f%%', startangle=180)

plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title('Distribution of Network Traffic Protocols')

# Display the pie chart
plt.show()
```

Distribution of Network Traffic Protocols



2.2. Distribution of Network Traffic Types: DNS, HTTP & FTP

```
df['Traffic Type'].value_counts()
```

```
DNS      13376
HTTP     13360
FTP      13264
Name: Traffic Type, dtype: int64
```

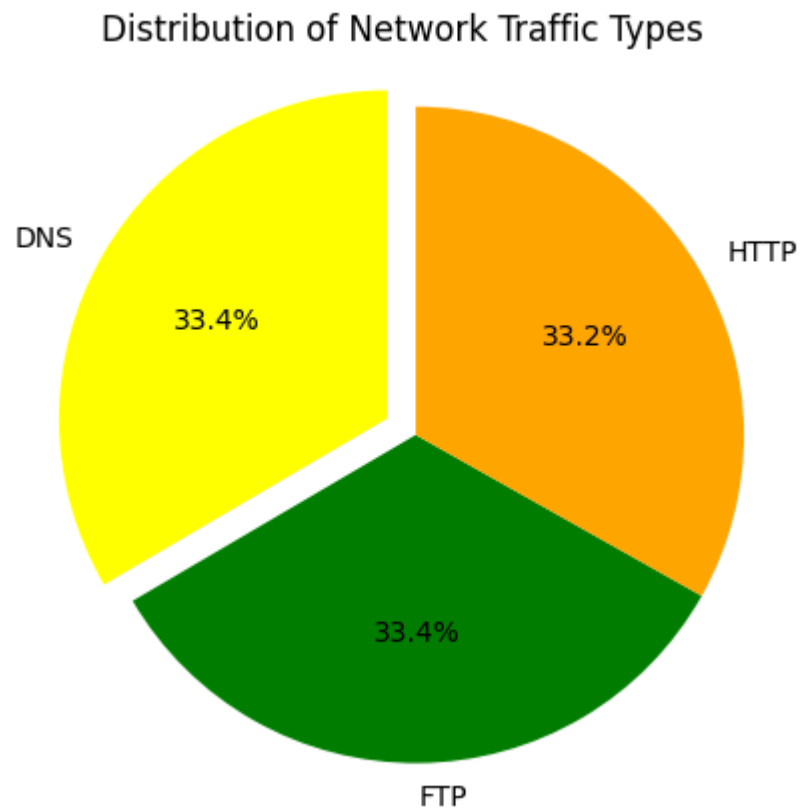
Counts the occurrences of 'Domain Name System (DNS), HyperText Transfer Protocol (HTTP), & File Transfer Protocol (FTP)' in the 'Traffic Type' column of the DataFrame.

```
# Data for the pie chart
labels = ['DNS', 'FTP', 'HTTP']
sizes = df['Traffic Type'].value_counts()
colors = ['yellow', 'green', 'orange']
explode = (0.1, 0, 0)

# Create a pie chart
plt.pie(sizes, labels=labels, colors=colors, explode=explode, autopct='%1.1f%%', startangle=90)

plt.axis('equal')
plt.title('Distribution of Network Traffic Types')

# Display the pie chart
plt.show()
```



2.3. Distribution of Action Taken (Logged, Ignored & Blocked)

```
df['Action Taken'].value_counts()
```

```
Blocked    13529
Ignored    13276
Logged     13195
Name: Action Taken, dtype: int64
```

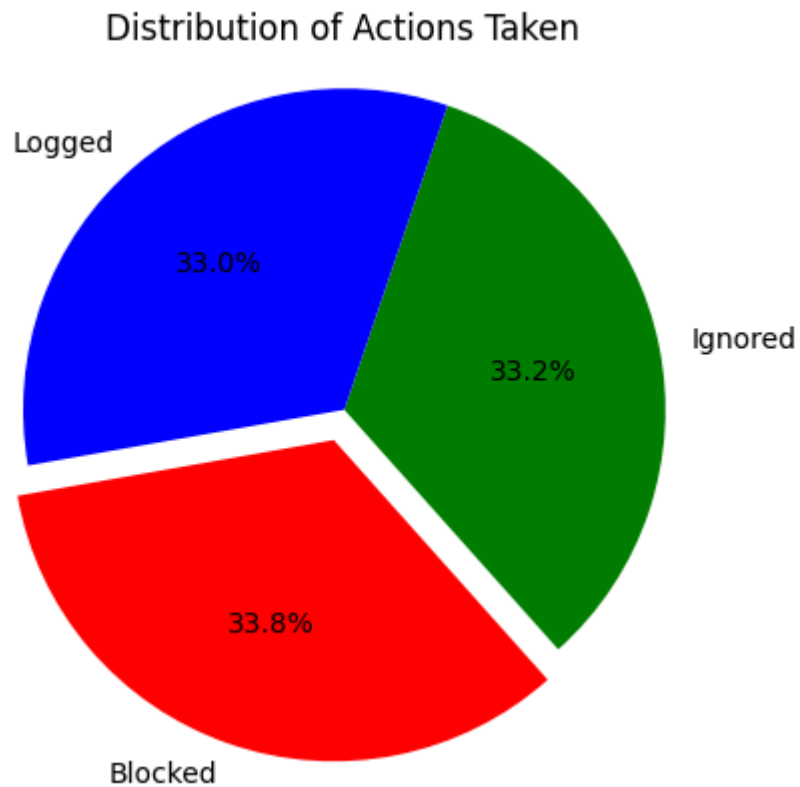
Counts the occurrences of 'Logged, Ignored & Blocked' in the 'Action Taken' column of the DataFrame.

```
labels = ['Blocked', 'Ignored', 'Logged']
sizes = df['Action Taken'].value_counts()
colors = ['Red', 'green', 'blue']
explode = (0.1, 0, 0)

# Create a pie chart
plt.pie(sizes, labels=labels, colors=colors, explode=explode, autopct='%1.1f%%', startangle=190)

plt.axis('equal')
plt.title('Distribution of Actions Taken')

# Display the pie chart
plt.show()
```



2.4. Distribution of Packet Types (Data & Control)

```
df['Packet Type'].value_counts()
```

```
Control    20237  
Data       19763  
Name: Packet Type, dtype: int64
```

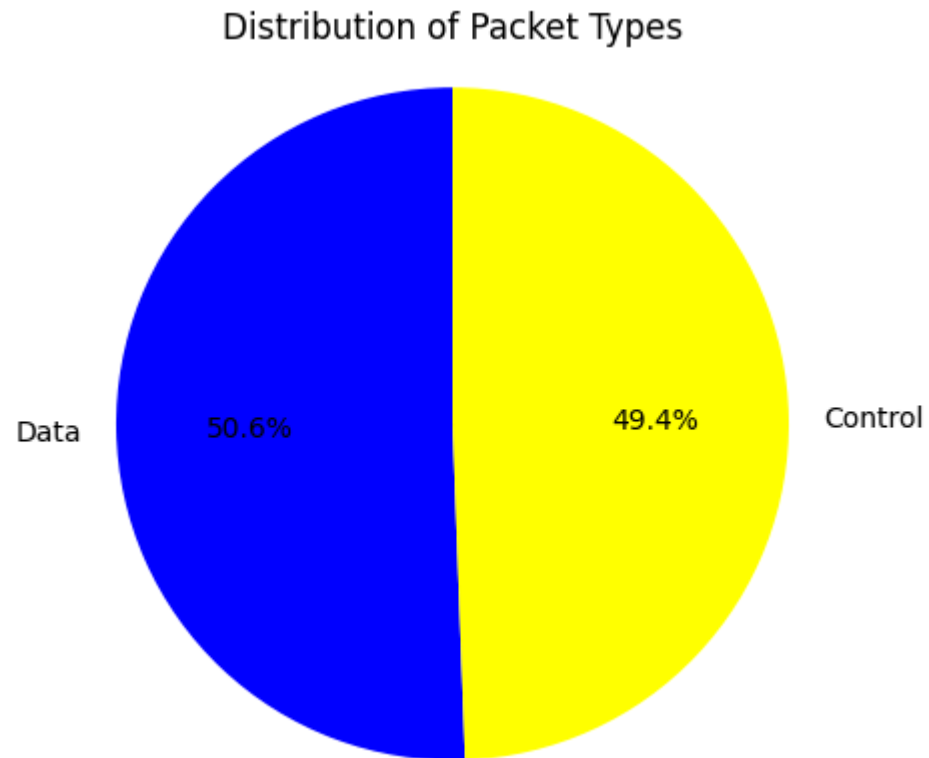
Counts the occurrences of 'Control & Data' in the 'Packet Type' column of the DataFrame.

```
labels = ['Data', 'Control']
sizes = df['Packet Type'].value_counts()
colors = ['blue', 'yellow']
explode = (0, 0)

# Create a pie chart
plt.pie(sizes, labels=labels, colors=colors, explode=explode, autopct='%1.1f%%', startangle=90)

plt.axis('equal')
plt.title('Distribution of Packet Types')

# Display the pie chart
plt.show()
```



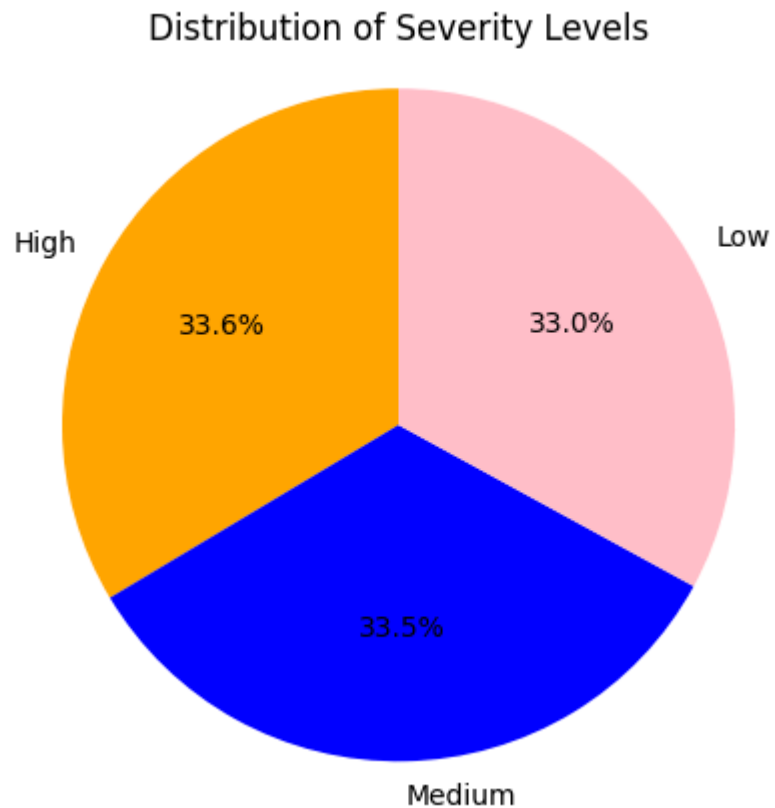
2.5. Distribution of Severity Levels (High, Medium & Low)

```
df['Severity Level'].value_counts()
```

```
Medium    13435  
High      13382  
Low       13183  
Name: Severity Level, dtype: int64
```

Counts the occurrences of 'High, Medium & Low' in the 'Severity Levels' column of the DataFrame.

```
labels = ['High', 'Medium', 'Low']  
sizes = df['Severity Level'].value_counts()  
colors = ['orange', 'blue', 'pink']  
explode = (0, 0, 0)  
  
# Create a pie chart  
plt.pie(sizes, labels=labels, colors=colors, explode=explode, autopct='%1.1f%%', startangle=90)  
  
plt.axis('equal')  
plt.title('Distribution of Severity Levels')  
  
# Display the pie chart  
plt.show()
```



2.6. Distribution of Log Sources (Firewall & Server)

```
df['Log Source'].value_counts()
```

```
Firewall    20116
Server      19884
Name: Log Source, dtype: int64
```

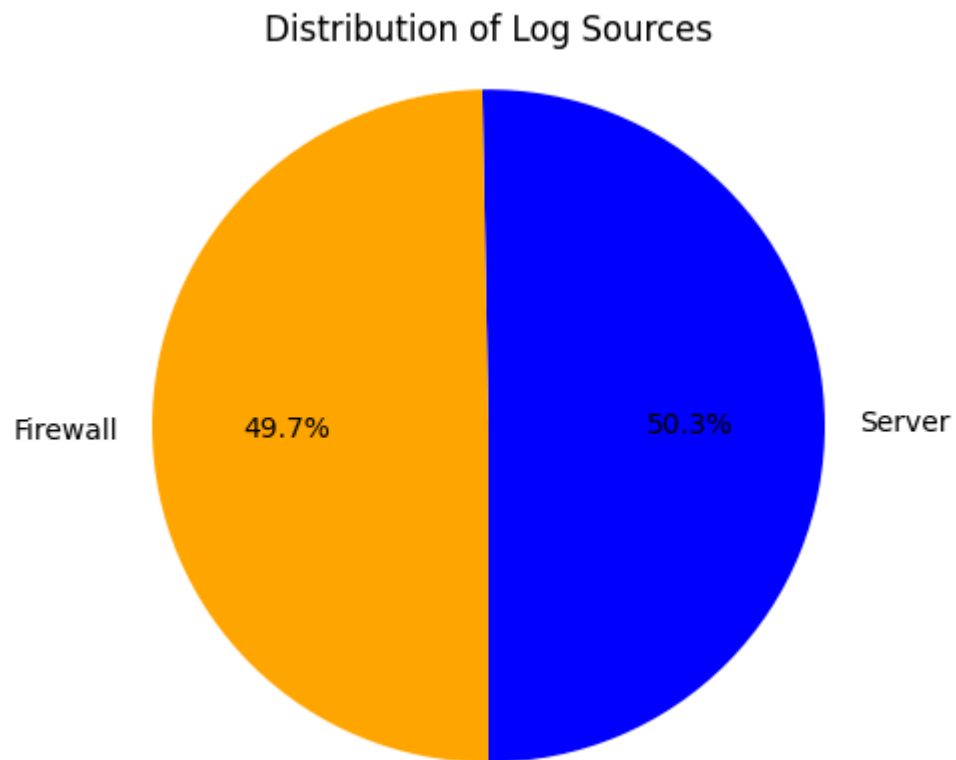
Counts the occurrences of 'Server & Firewall' in the 'Log Source' column of the DataFrame

```
labels = ['Server', 'Firewall']
sizes = df['Log Source'].value_counts()
colors = ['blue', 'orange']
explode = (0, 0)

plt.pie(sizes, labels=labels, colors=colors, explode=explode, autopct='%1.1f%%', startangle=270)

plt.axis('equal')
plt.title('Distribution of Log Sources')

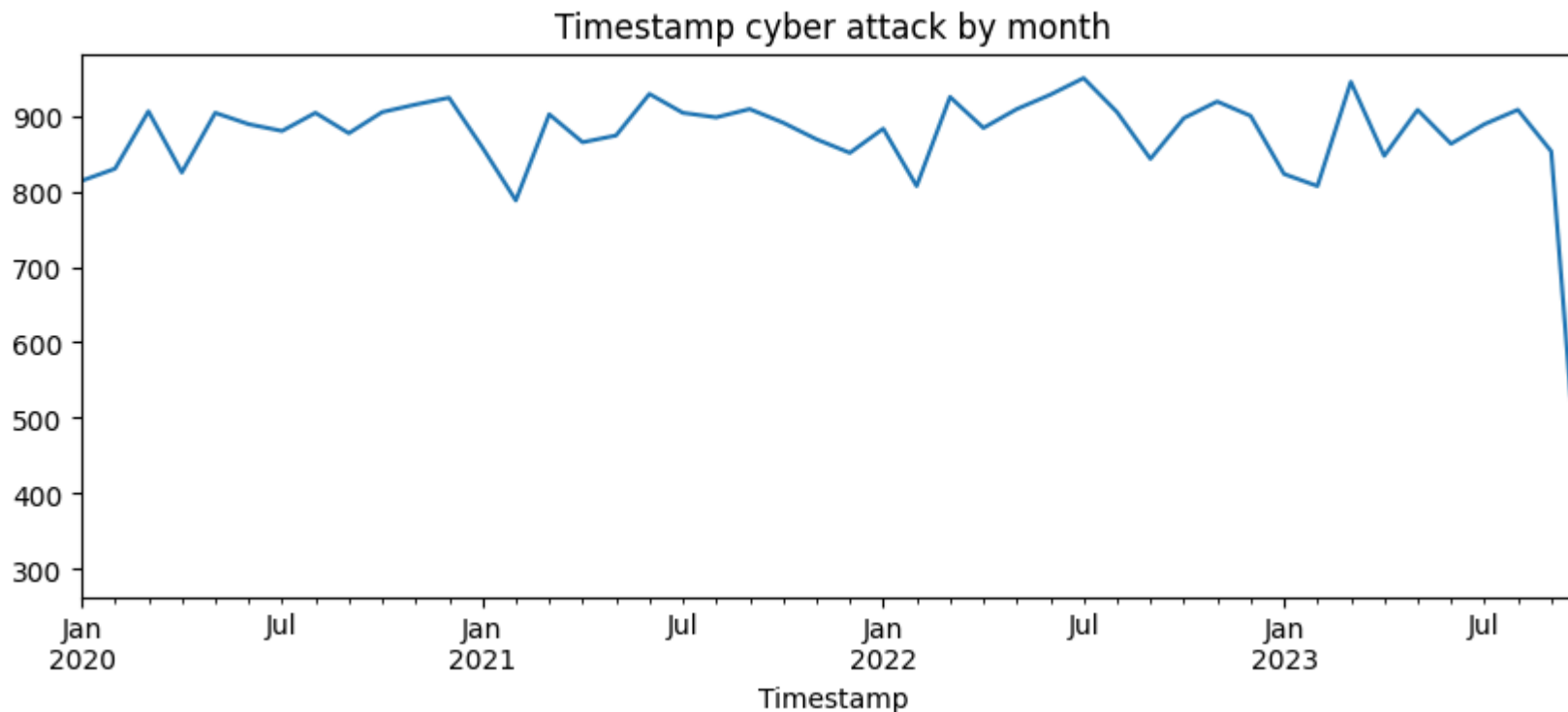
# Display the pie chart
plt.show()
```



3. Subplot of Timestamp Cyber Attack

3.1. Timestamp Cyber Attack by Month

```
fig = plt.figure(figsize=(10, 8))  
fig.add_subplot(211)  
df.resample('M', on='Timestamp')['Attack Type'].count().plot(title='Timestamp cyber attack by month')  
plt.show()
```

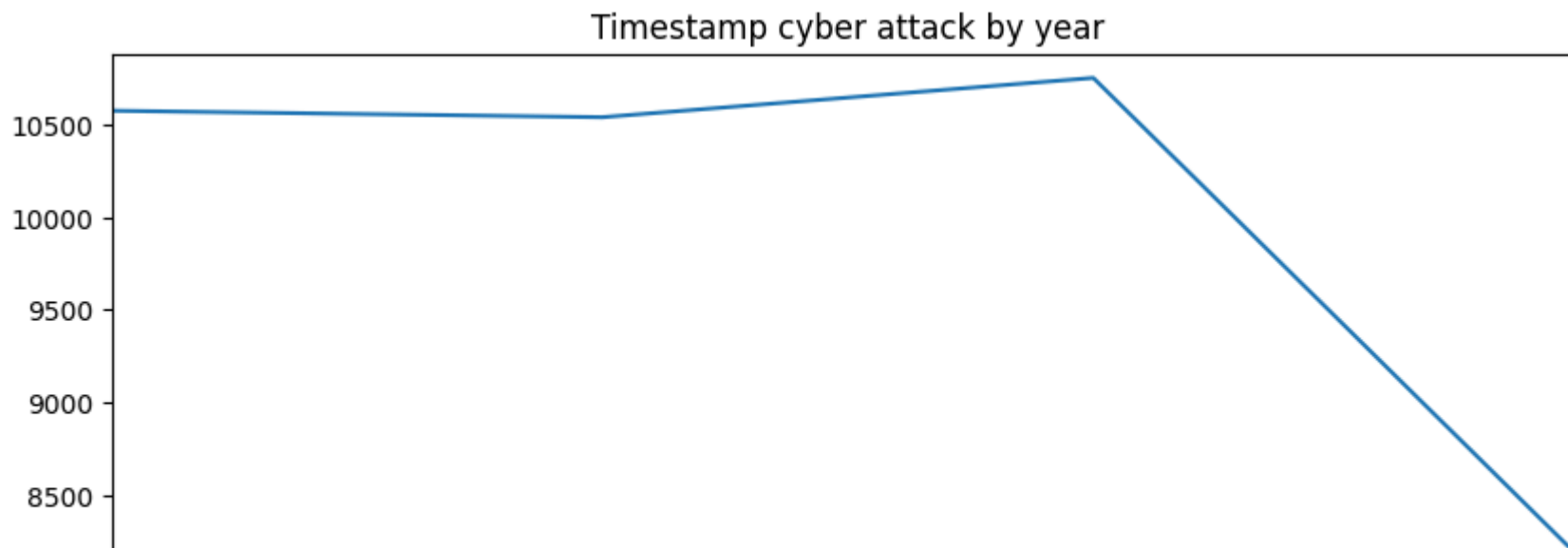


Develop a figure that is 10 by 8 inches and has a subplot using Matplotlib. After that, it resamples a DataFrame df using the 'Timestamp' column as a basis, groups the data by month ('M'), and counts how many times each 'Attack Type' occurs. Ultimately, the outcome is displayed

as a line graph labelled "Timestamp cyber attack by month." This graphic illustration aids in comprehending the monthly distribution of cyberattacks across time.

3.2. Timestamp Cyber Attack by Year

```
fig = plt.figure(figsize=(10, 8))  
fig.add_subplot(212)  
df.resample('Y', on='Timestamp')['Attack Type'].count().plot(title='Timestamp cyber attack by year')  
plt.show()
```

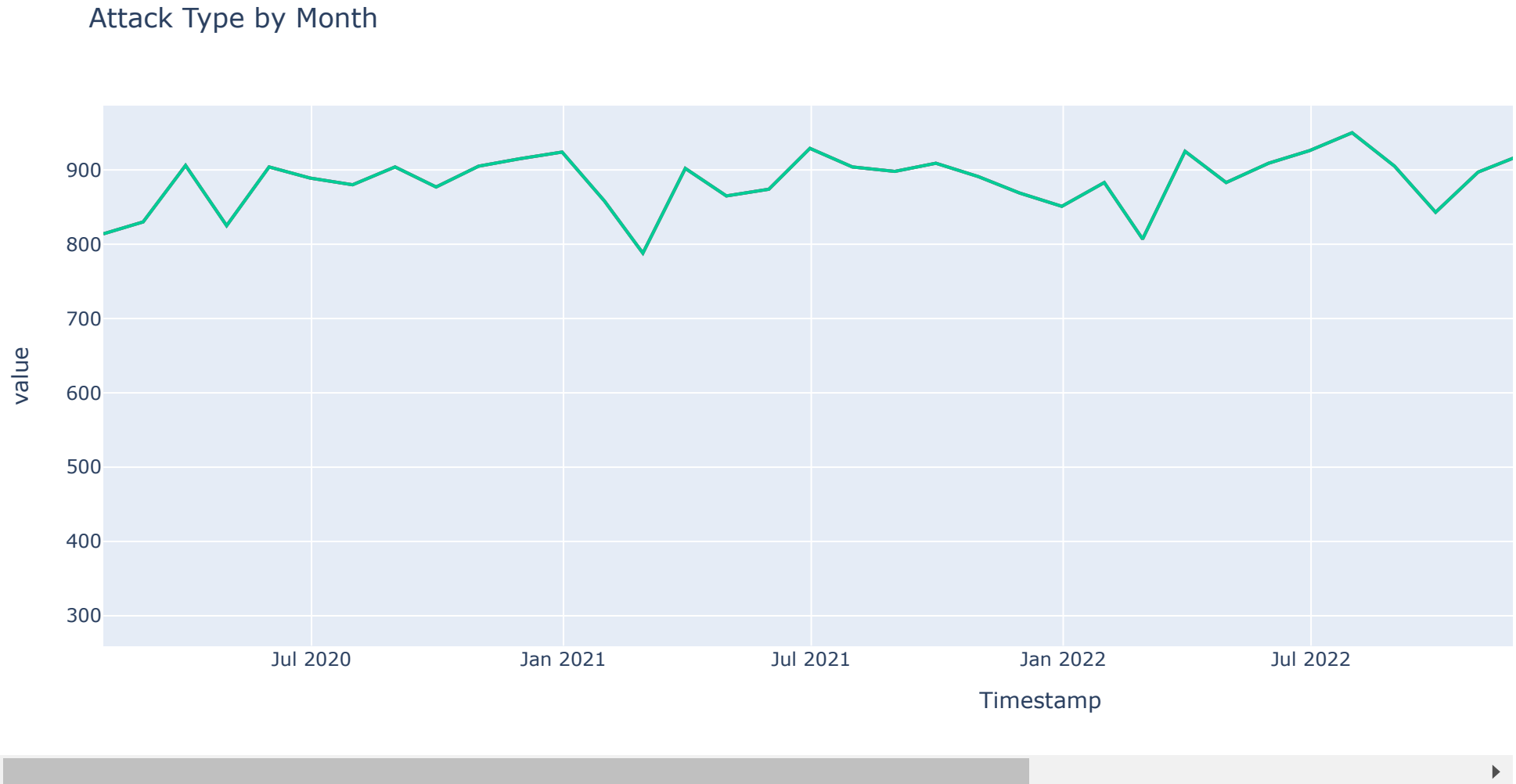


Develop a figure that is 10 by 8 inches and has a subplot using Matplotlib. After that, it resamples a DataFrame df using the 'Timestamp' column as a basis, groups the data by year ('Y'), and counts how many times each 'Attack Type' occurs. 'Timestamp cyber assault by year' is the title of the line graph that displays the outcome in the end. The yearly trend of cyberattacks is shown in this visualisation, which makes it possible to analyse long-term trends and variations in attack frequency over time.

4. Subplot of Attack Type

4.1. Attack Type by Month

```
import plotly.express as px
data = pd.crosstab(df['Timestamp'], df['Attack Type']).resample('M').count().melt(ignore_index=False)
px.line(data, x=data.index, y='value', color='Attack Type', title='Attack Type by Month').show()
```

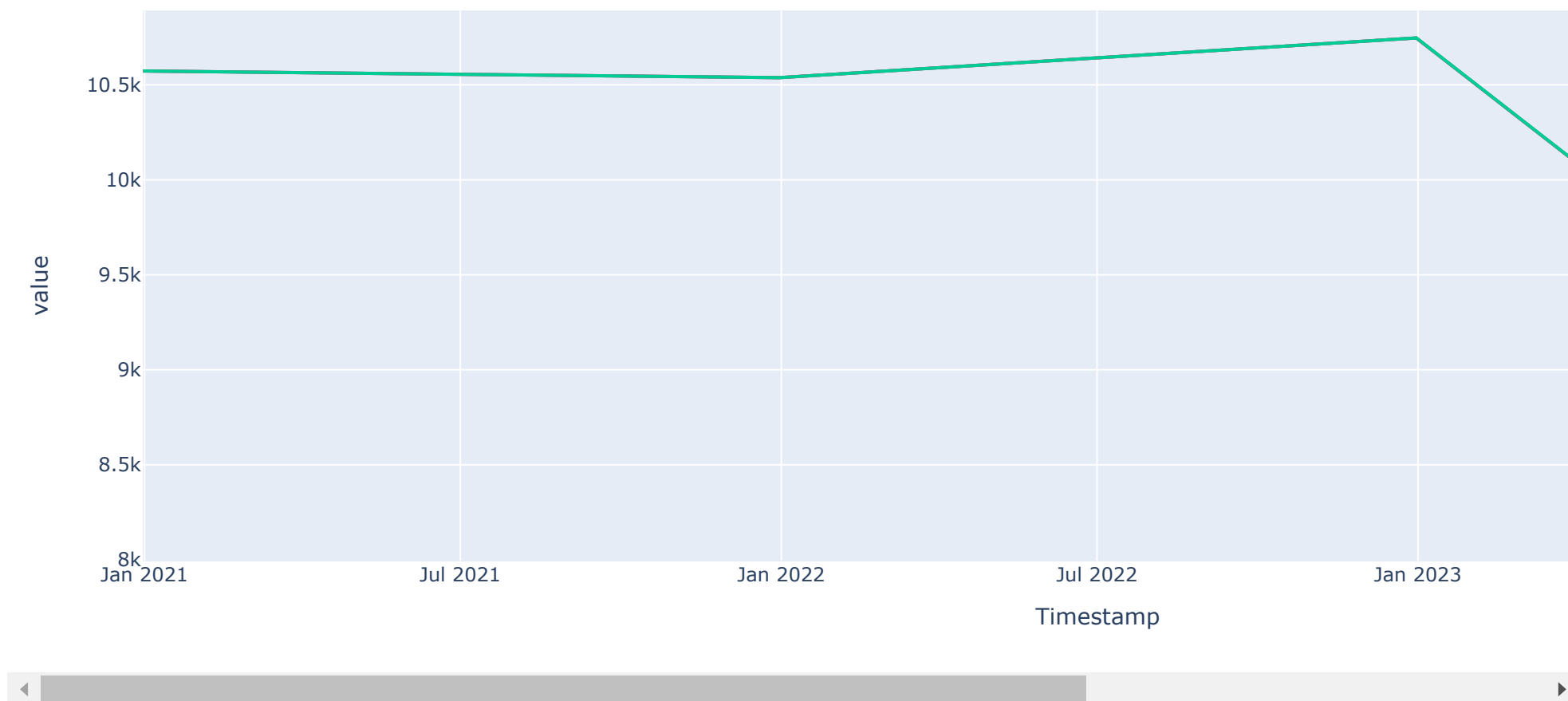


Utilizes plotly Express to generate an interactive line graph. 'Timestamp' and 'Attack Type' are first cross-tabulated, the occurrences are counted, and the data is then resampled monthly. To convert the resultant DataFrame into a plottable format, it is melted. Lastly, it creates a line plot that shows how different attack kinds change over the course of months. The x-axis indicates time, the y-axis the number of attacks, and each line refers to a particular attack type.

4.2. Attack Type by Year

```
import plotly.express as px
data = pd.crosstab(df['Timestamp'], df['Attack Type']).resample('Y').count().melt(ignore_index=False)
px.line(data, x=data.index, y='value', color='Attack Type', title='Attack Type by Year').show()
```

Attack Type by Year

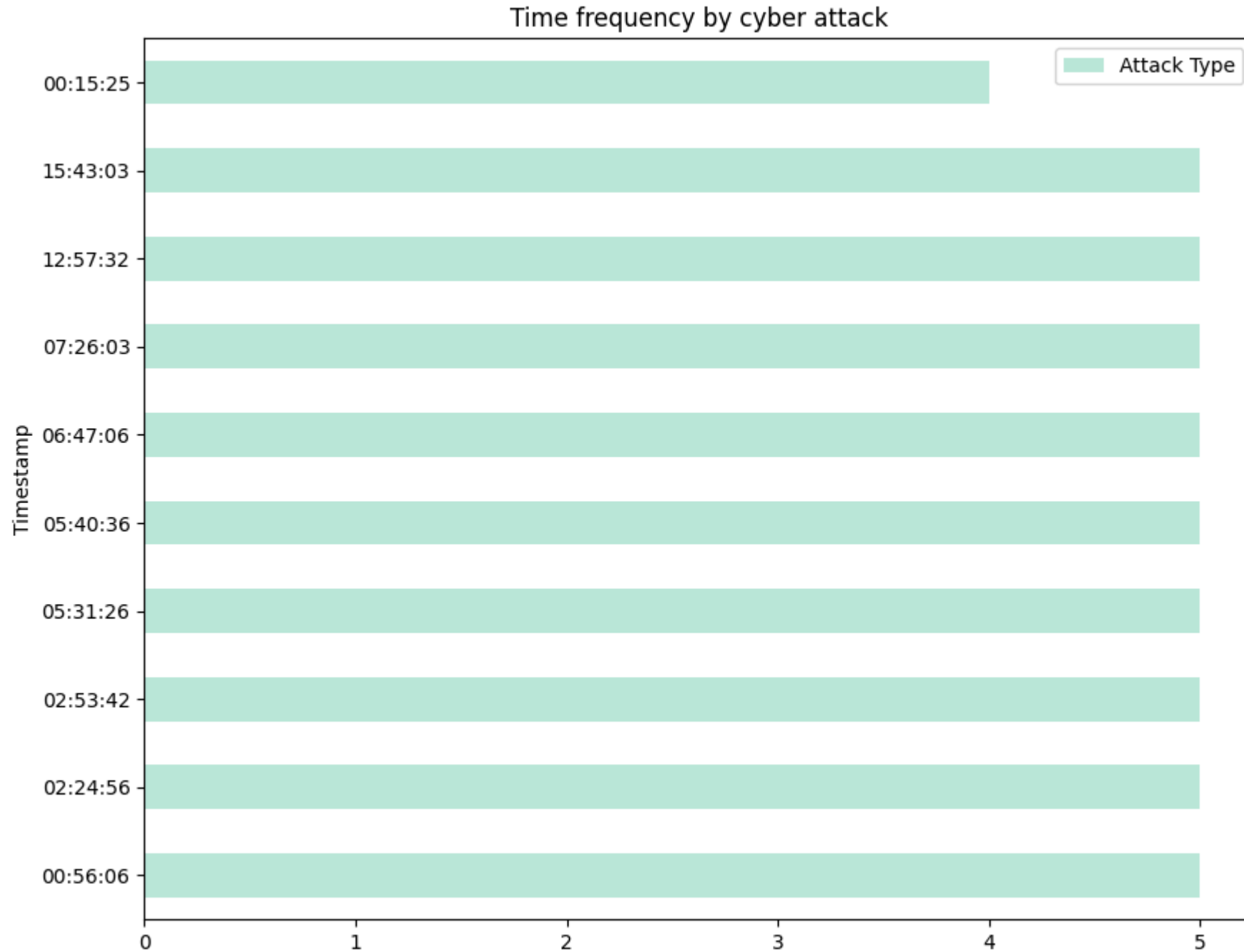


Utilizes plotly Express to generate an interactive line graph. 'Timestamp' and 'Attack Type' are first cross-tabulated, the occurrences are counted, and the data is then resampled yearly. To convert the resultant DataFrame into a plottable format, it is melted. Lastly, it creates a line plot that shows how different attack kinds change over the course of year. The x-axis indicates time, the y-axis the number of attacks, and each line refers to a particular attack type.

5. Time Frequency by Cyber Attack

```
df.groupby(df['Timestamp'].dt.time).agg({'Attack Type': 'count'}).nlargest(10, 'Attack Type').plot(kind='barh', figsize=(10,8), cmap=
```

<Axes: title={'center': 'Time frequency by cyber attack'}, ylabel='Timestamp'>



The count of 'Attack Type' occurrences within each time category is then aggregated after grouping the DataFrame `df` by the time component of the 'Timestamp' column. Plotting the top 10 periods with the highest attack frequency results in a horizontal bar chart. Darker hues on the colormap ('icefire') denote higher assault frequencies, which improves the visual depiction. This graphic helps to detect peak attack periods by clearly displaying the distribution of cyberattacks throughout the day.

6. Default Graph Plotting with Stats and Insights

```
plot(df)
```

Show Stats and Insights

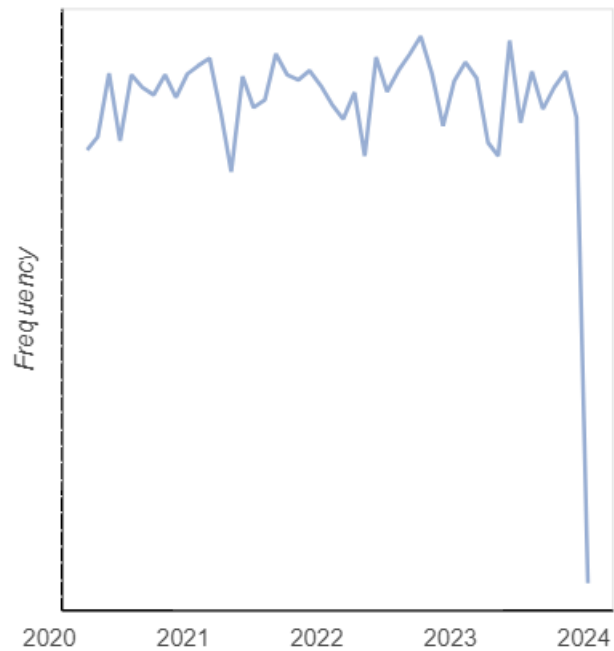
Number of plots per page:

-

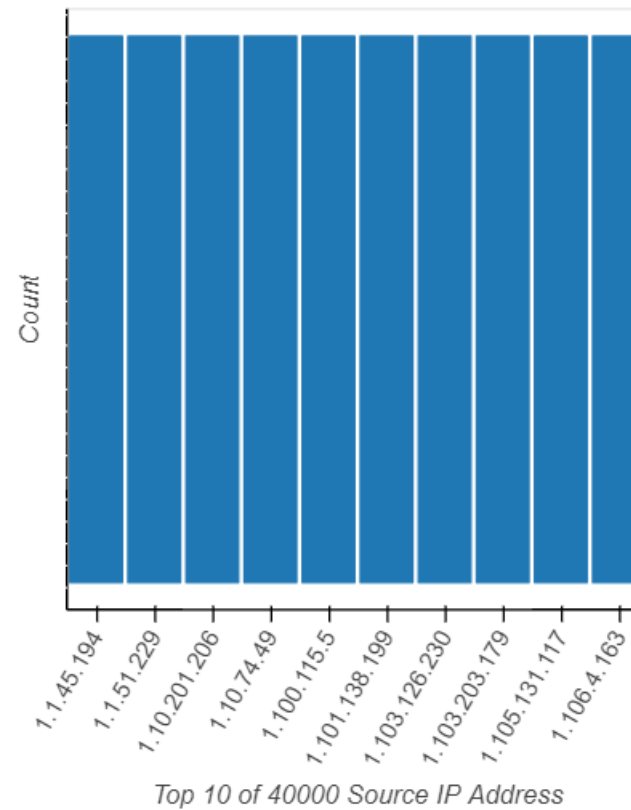
9

+

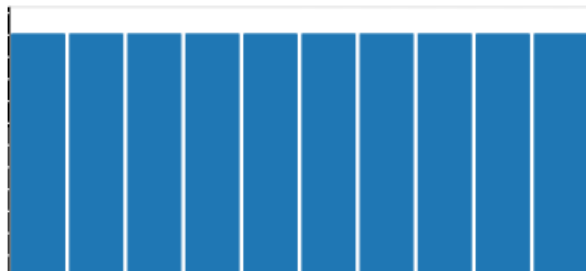
Timestamp



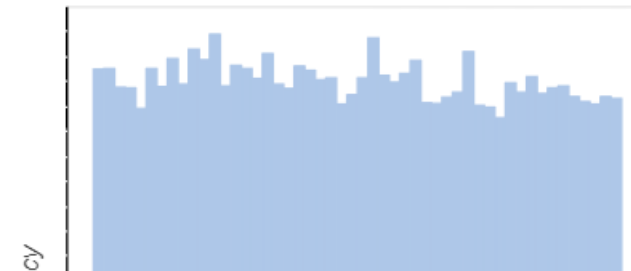
Source IP Address

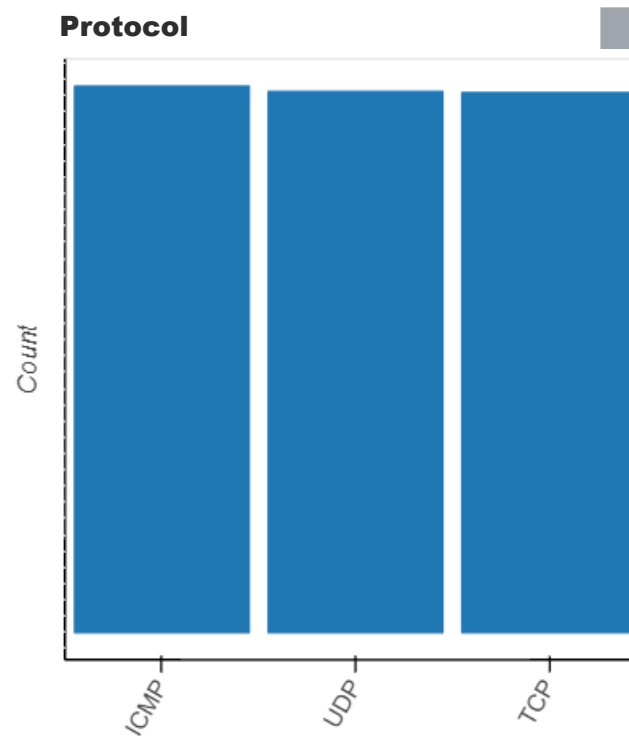
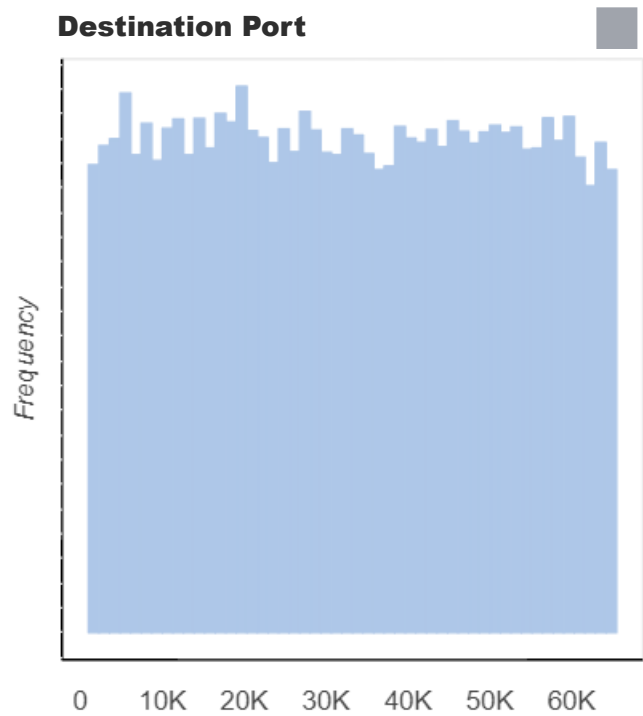
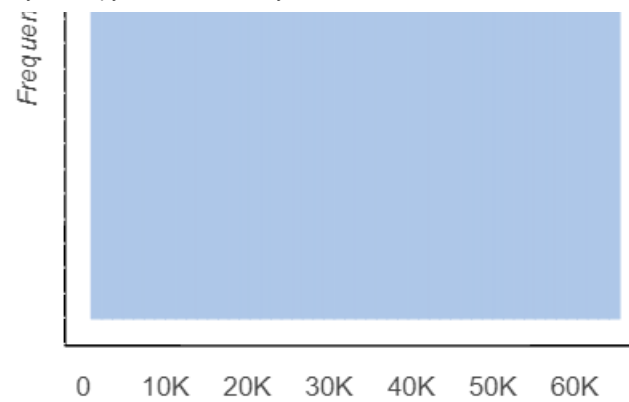


Destination IP Address



Source Port





Packet Length

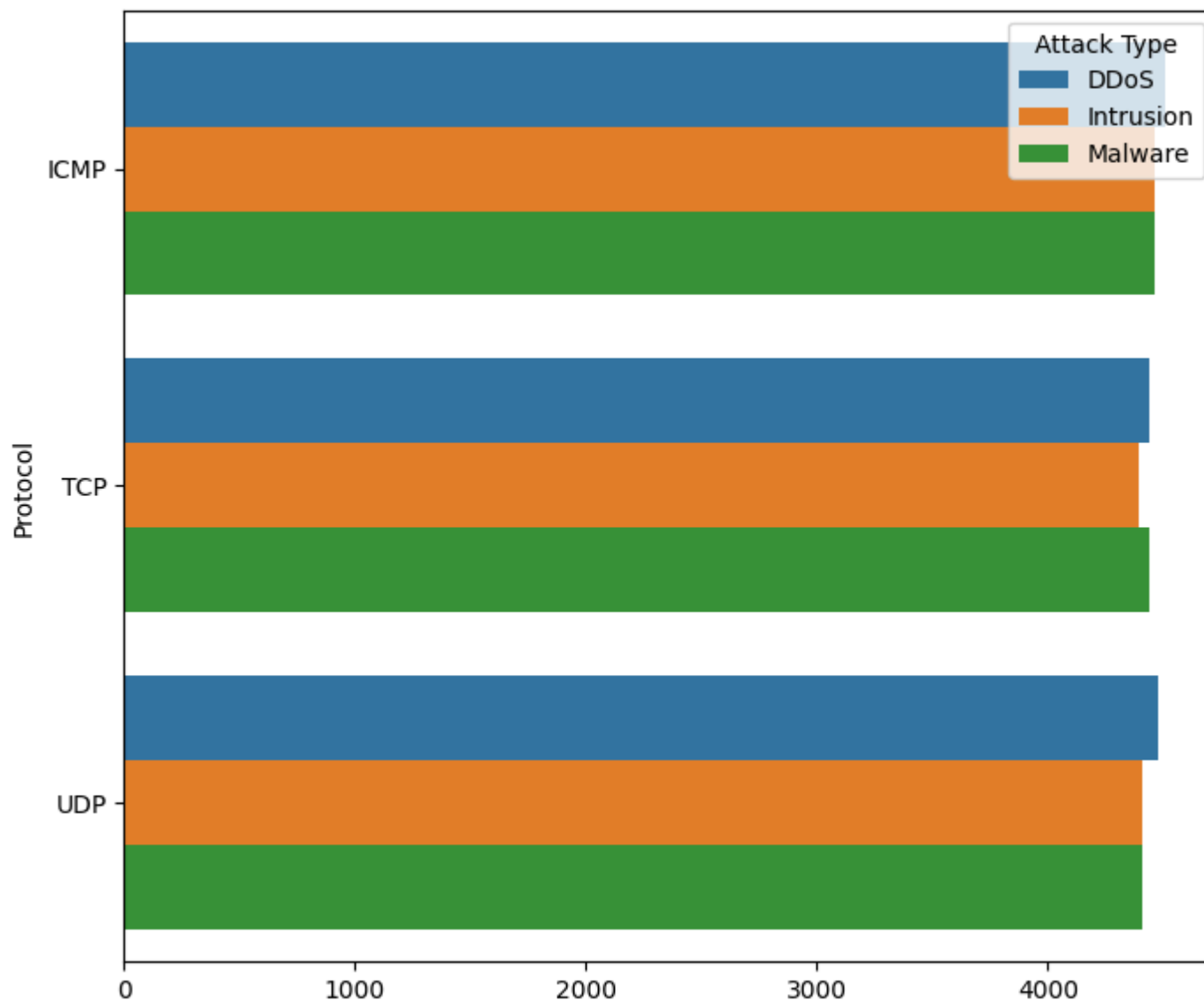
Packet Type

A general call to plot the DataFrame `df` using the default charting approach is made via the function `plot(df)`. It's challenging to give a clear description of what this function call will do without more context or particulars on the composition and contents of `df`. The DataFrame's structure and content, the default plotting parameters, and the libraries being used can all have a significant impact on the final result. Generally speaking, it would try to create a plot using the data that is included in `df`. Depending on the data types and dimensions in the DataFrame, this may be any kind of plot, including a line, scatter, or histogram plot.

7. Bar Chart for Respected Attack Type

7.1. Attack Type: Protocol

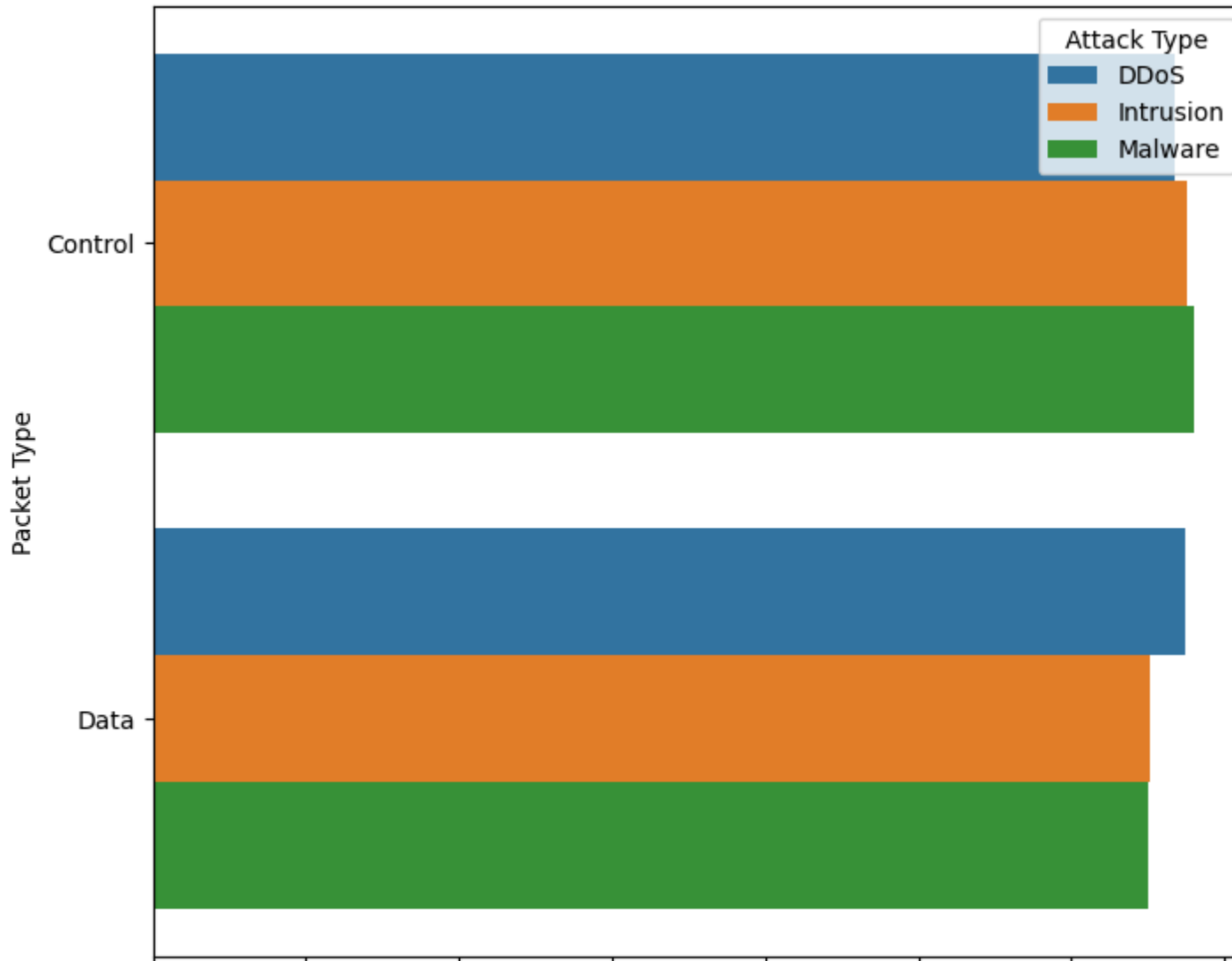
```
for col in ['Protocol']:
    plt.figure(figsize=(8,7))
    sns.countplot(data=df, y=df[col], hue='Attack Type')
    plt.show()
```



Iterates distinct figure for the count distribution of each column by iterating over each one included in the list ['Protocol']. It shows the frequency of occurrences for each distinct value of the "Protocol" column, broken down by various "Attack Types," using Seaborn's countplot function. Understanding the prevalence of assaults across each protocol category is made easier by the resultant plots, which provide a visual comparison of attack types across various protocols.

7.2. Attack Type: Packet Type

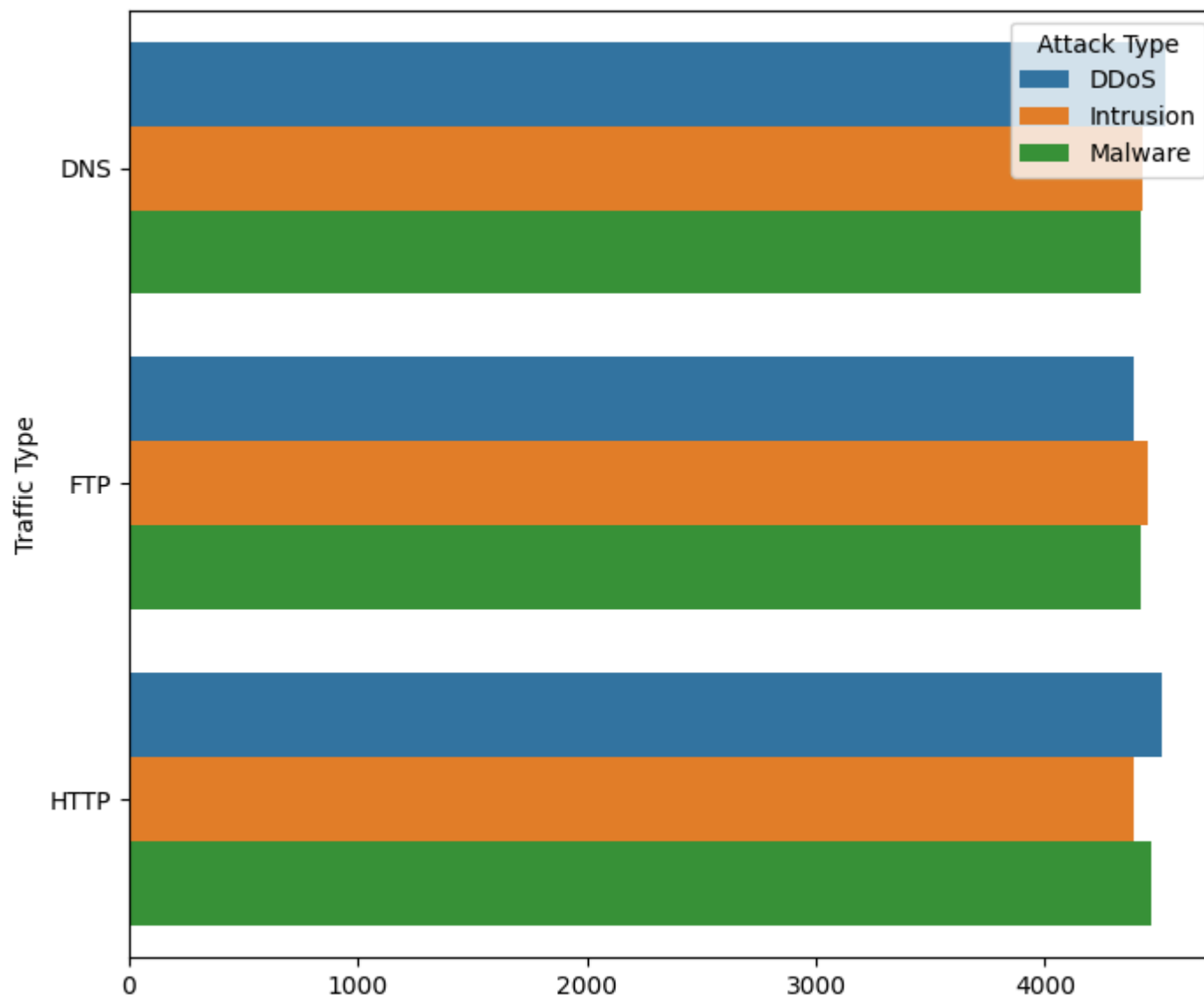
```
for col in ['Packet Type']:  
    plt.figure(figsize=(8,7))  
    sns.countplot(data=df, y=df[col], hue='Attack Type')  
    plt.show()
```



Iterates distinct figure for the count distribution of each column by iterating through each column listed in the list ['Packet Type']. It shows the frequency of occurrences for each distinct value of the "Packet Type" column, grouped by various "Attack Types," using Seaborn's countplot function. These charts make the distribution of attacks within each packet type category easier to comprehend by providing a visual comparison of attack types across various packet types.

7.3. Attack Type: Traffic Type

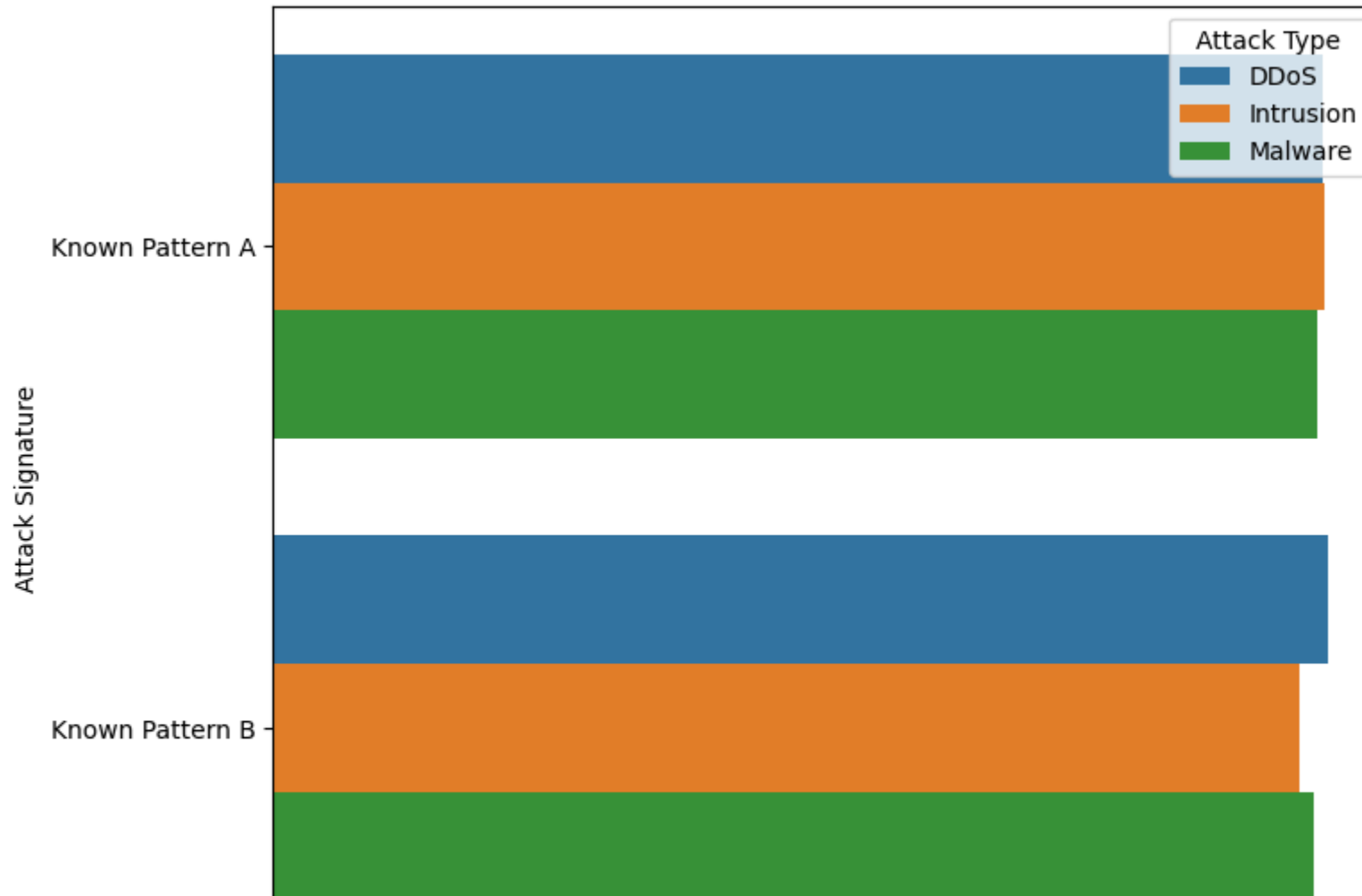
```
for col in ['Traffic Type']:
    plt.figure(figsize=(8,7))
    sns.countplot(data=df, y=df[col], hue='Attack Type')
    plt.show()
```



Iterates over each column listed in the list ['Traffic Type'], producing a unique figure for the count distribution of each column. It uses the countplot function from Seaborn to show the frequency of occurrences for each distinct value in the "Traffic Type" column, broken down into several "Attack Types." Understanding the prevalence of assaults across each traffic type category is made easier by these charts, which provide a visual comparison of attack types across various traffic kinds.

7.4. Attack Type: Attack Signature

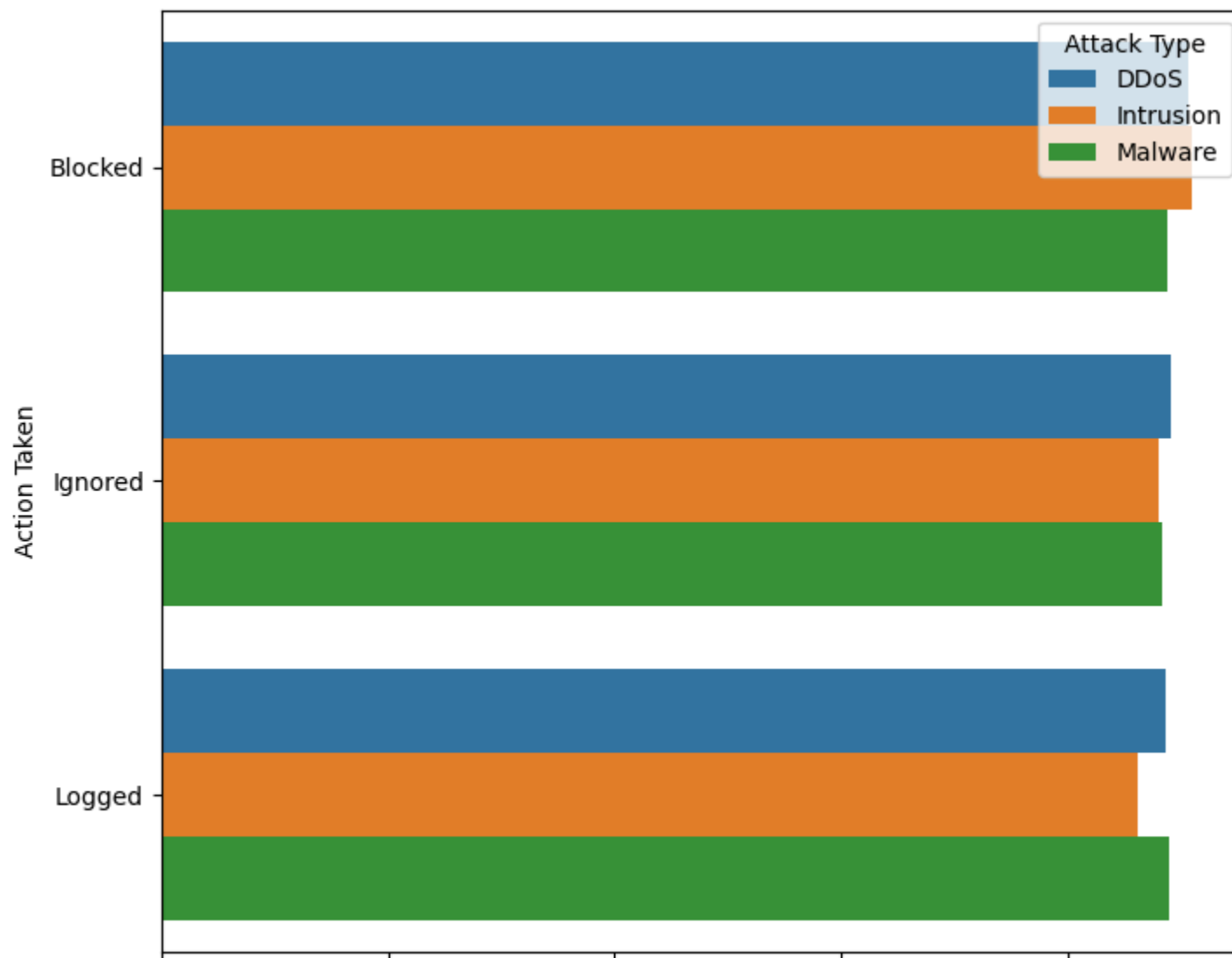
```
for col in ['Attack Signature']:  
    plt.figure(figsize=(8,7))  
    sns.countplot(data=df, y=df[col], hue='Attack Type')  
    plt.show()
```



Iterates over each of the columns included in the list ['Attack Signature'], producing a unique figure for the count distribution of each column. It shows the frequency of occurrences for each distinct value of the "Attack Signature" column, grouped by various "Attack Types," using Seaborn's countplot function. These charts make it easier to comprehend attack distributions based on particular attack features or patterns by giving a visual comparison of the different attack types linked to different attack signatures.

7.5. Attack Type: Action Taken

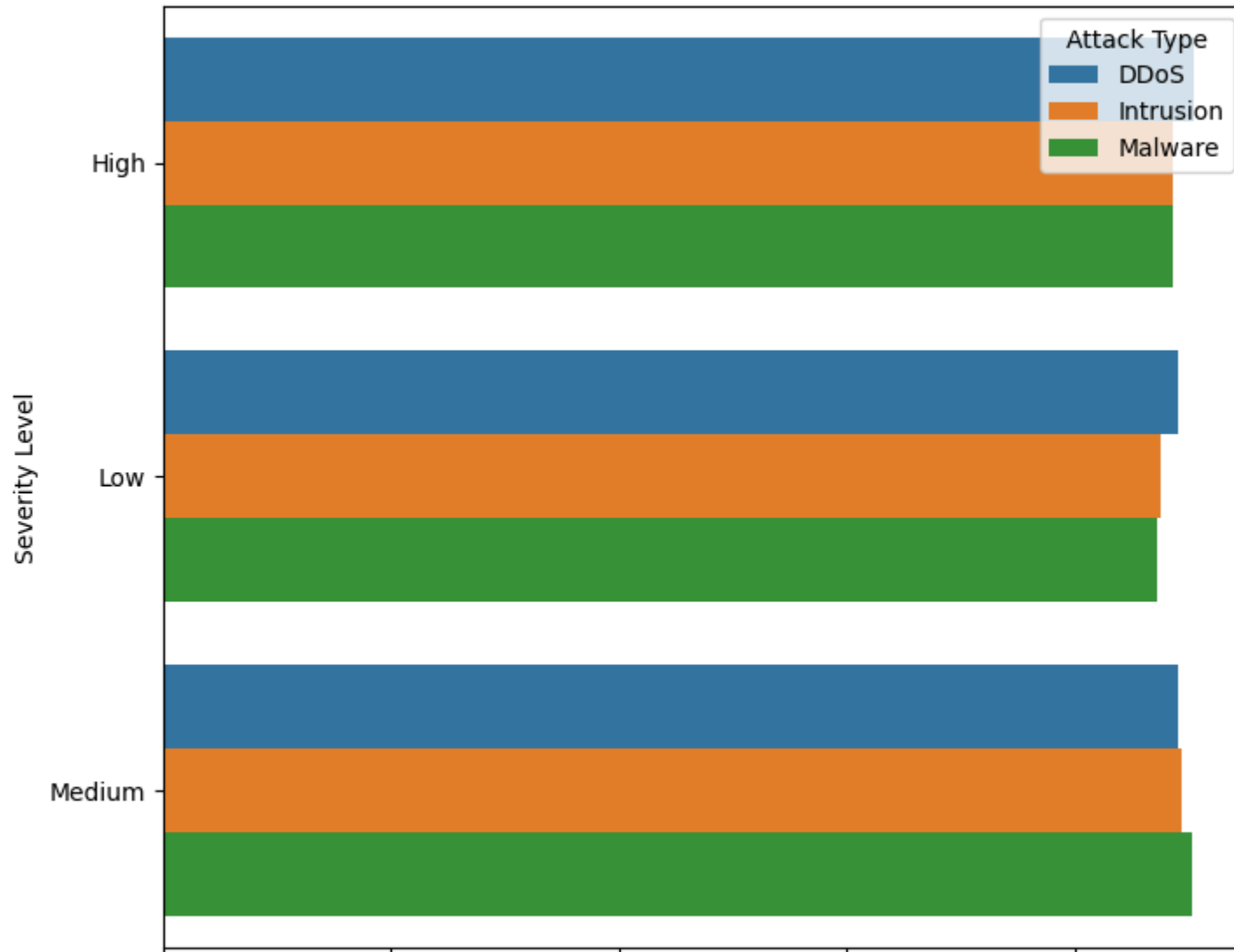
```
for col in ['Action Taken']:  
    plt.figure(figsize=(8,7))  
    sns.countplot(data=df, y=df[col], hue='Attack Type')  
    plt.show()
```



Iterates different figure for the count distribution of each column by iterating over each column included in the list ['Action Taken']. It displays the frequency of occurrences for each distinct value of the "Action Taken" column, broken down by various "Attack Types," using Seaborn's countplot function. This helps to understand the distribution of attacks based on the activities done after an attack occurrence. These plots provide a visual comparison of different attack types connected with different actions made in response.

7.6. Attack Type: Severity Level

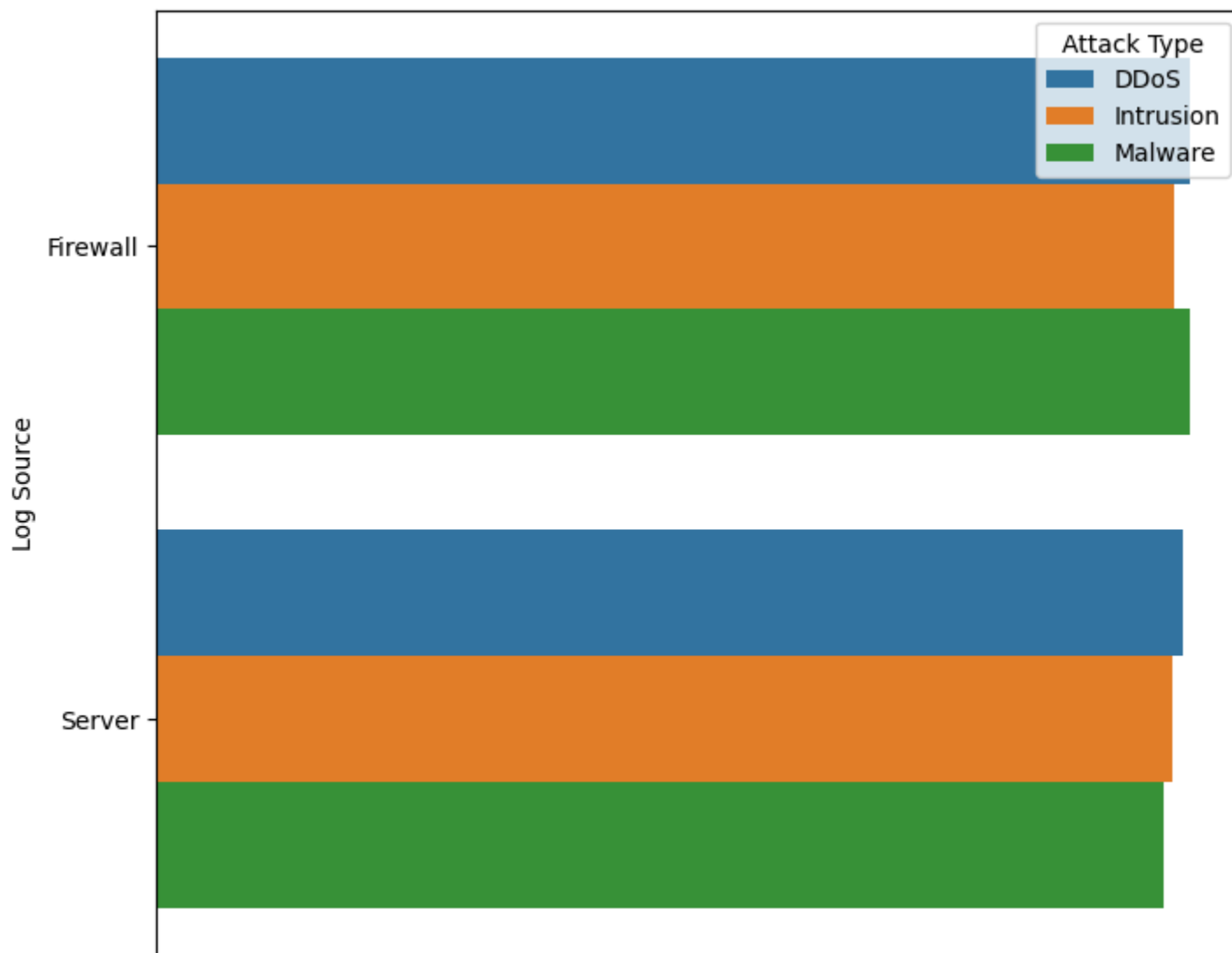
```
for col in ['Severity Level']:  
    plt.figure(figsize=(8,7))  
    sns.countplot(data=df, y=df[col], hue='Attack Type')  
    plt.show()
```



Iterates over each of the columns included in the list ['Severity Level'] and produces a unique figure for the count distribution of each column. It shows the frequency of occurrences for each distinct value of the "Severity Level" column, broken down by various "Attack Types," using Seaborn's countplot function. Understanding the distribution of assaults according to their severity is made easier by these plots, which offer a visual comparison of attack types across various severity levels.

7.7. Attack Type: Log Source

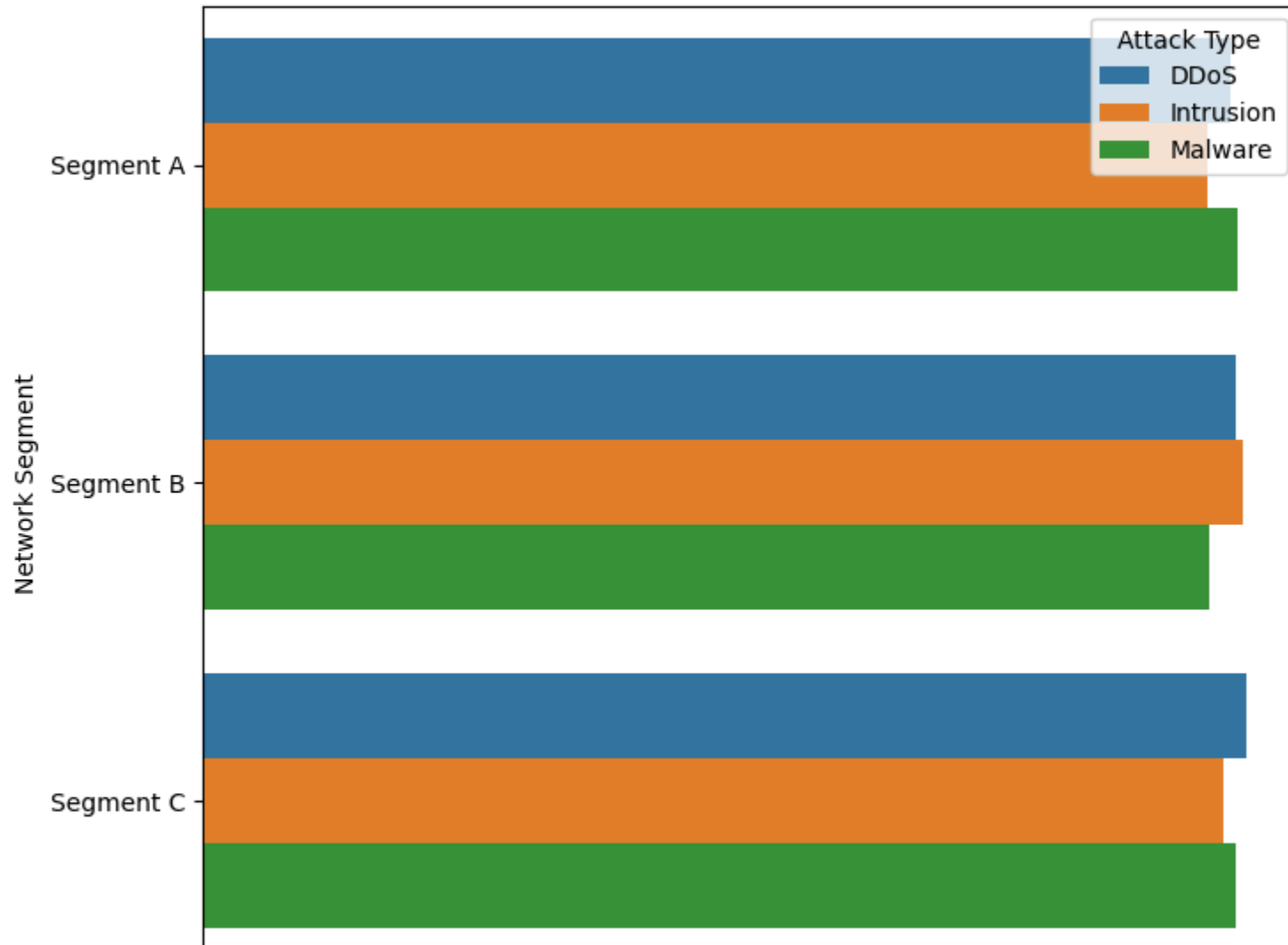
```
for col in ['Log Source']:
    plt.figure(figsize=(8,7))
    sns.countplot(data=df, y=df[col], hue='Attack Type')
    plt.show()
```



Iterates over each column listed in the list ['Log Source'] to produce a unique figure representing the count distribution for each column. It uses the countplot function from Seaborn to show the frequency of occurrences for each distinct value in the "Log Source" column, broken down by several "Attack Types." The distribution of assaults across different sources or systems within the network architecture may be better understood with the help of these plots, which provide a visual comparison of attack types originating from distinct log sources.

7.8. Attack Type: Network Segment

```
for col in ['Network Segment']:  
    plt.figure(figsize=(8,7))  
    sns.countplot(data=df, y=df[col], hue='Attack Type')  
    plt.show()
```



Iterates over every column listed in the list ['Network Segment'] and produces a unique figure for the count distribution of each column. It shows the frequency of occurrences for each distinct value of the "Network Segment" column, broken down by various "Attack Types," using Seaborn's countplot function. Understanding the prevalence of assaults across particular network infrastructure segments is made easier by these plots, which offer a visual comparison of attack types across various network segments.

8. Plots for Respected Columns

8.1. Plots for Device Information Column

```
plot(df, 'Device Information')
```

```
/usr/local/lib/python3.10/dist-packages/dataprep/eda/distribution/render.py:274: FutureWarning:
```

```
The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
```

[Stats](#)[Bar Chart](#)[Pie Chart](#)[Word Frequency](#)[Word Length](#)[Value Table](#)

Customised charting method with a DataFrame called df and a special column called "Device Information" Most likely, this function creates a visualisation based on the information in the 'Device Information' column. However, it is impossible to specify the precise kind of plot or how the data is visualised without more information on the implementation specifics or libraries that are accessible.

8.2. Plots for Payload Data Column

```
plot(df, 'Payload Data')
```

```
/usr/local/lib/python3.10/dist-packages/dataprep/eda/distribution/render.py:274: FutureWarning:
```

```
The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
```

[Stats](#)[Bar Chart](#)[Pie Chart](#)[Word Frequency](#)[Word Length](#)[Value Table](#)

Customised plotting function, where 'Payload Data' indicates a particular column within a DataFrame and df stands for a DataFrame. Most likely, this function creates a visualisation based on the information in the 'Payload Data' column. Nevertheless, it is impossible to identify the precise kind of plot or the manner in which the data is displayed without knowing more about the implementation or the libraries that are accessible.

8.3. Plots for Packet Length Column

```
plot(df, 'Packet Length')
```

Stats

Histogram

KDE Plot

Normal Q-Q Plot

Box Plot

Value Table

Utility designed to produce a visual representation of the data in the DataFrame df's "Packet Length" column. Most usually, a plot that depicts the distribution or properties of the packet lengths in the dataset is produced. Depending on the nature and distribution of the data in the

'Packet Length' column, a particular plot or visualisation approach may be used, such as a box plot, density plot, or histogram.

8.4. Plots for Anomaly Scores

```
plot(df, 'Anomaly Scores')
```

Stats

Histogram

KDE Plot

Normal Q-Q Plot

Box Plot

Value Table

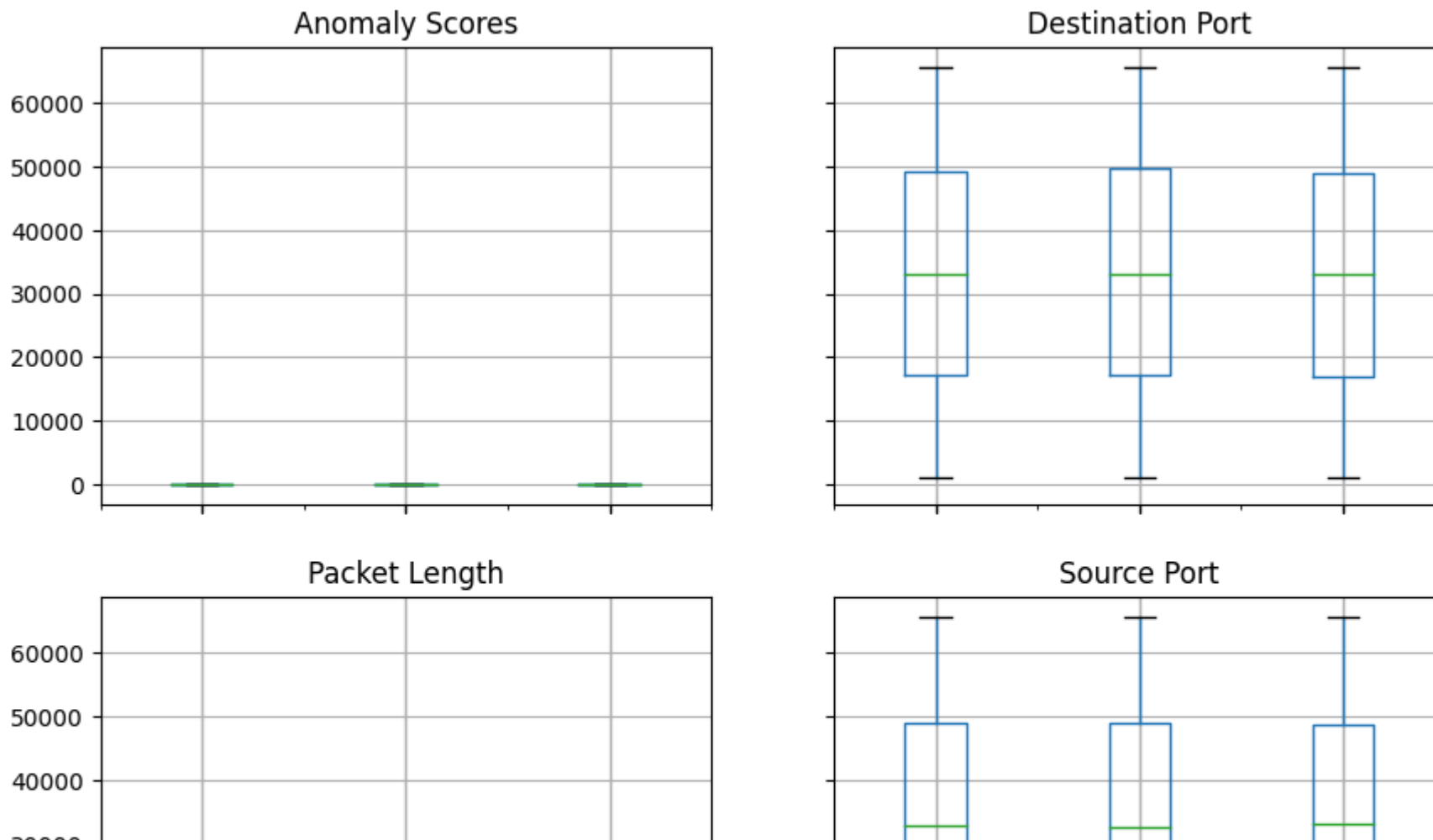
Customised charting function, where 'Anomaly Scores' is probably a column with numerical values that reflect anomaly scores and df is a DataFrame. A visualisation showing the distribution or pattern of anomaly scores within the dataset would probably be produced by this function. Depending on the features and range of the anomaly scores, line plots, box plots, and histograms are common visualisations for this type of data.

9. Box Plot

```
df.boxplot(figsize=(10,8), by='Attack Type')
```

```
array([[<Axes: title={ 'center': 'Anomaly Scores'}, xlabel='[Attack Type] '>,  
      <Axes: title={ 'center': 'Destination Port'}, xlabel='[Attack Type] '>],  
      [<Axes: title={ 'center': 'Packet Length'}, xlabel='[Attack Type] '>,  
      <Axes: title={ 'center': 'Source Port'}, xlabel='[Attack Type] '>]],  
      dtype=object)
```

Boxplot grouped by Attack Type



Utilizes boxplot function in Pandas to produce a grouped boxplot display. In the 'assault Type' column of the DataFrame df, it creates distinct boxplots for every distinct category, giving a comparative overview of the distribution and variability of numerical data across various assault

kinds. The size of the generated plot is modified by the `figsize` argument. This visual tool makes it easier to spot any differences in the distribution of data and outliers across various assault types.

DDoS**Intrusion****Malware****DDoS****Intrusion****Malware**

10. Scatter Plot

10.1. For Protocol

```
px.scatter_3d(df, x='Source Port', y='Destination Port', z='Packet Length', color='Protocol').show()
```

Visualizes information from the DataFrame `df` with values for the x, y, and z axes representing the "Source Port," "Destination Port," and "Packet Length" of the data. The colour of each data point is determined by the 'Protocol' column. In order to help find patterns or correlations in the data, this visualisation makes it possible to examine the relationships between source ports, destination ports, packet lengths, and protocols in a three-dimensional environment.