School of Computer Science and Mathematics

Keele University

Group Number: 23

Student ID: 21025647

Module Name: Visualisation for Data Analytics

Module Number: CSC-40048

Lecturer: Prof Sangeeta

Group Members: Millicent Djabatey, Angel Ariyibi, Daniel Devis & Gabriel Blackstock

Word Count: 77690 (excluding, figure legends, appendix, references, and footnotes)

Dataset Description

The dataset used in this report was extracted from Kaggle, specifically the Kaggle's Cyber Security Attacks repository. The dataset consists of a total of 40000 rows, each representing a cyber security attack incident and 5 main columns : Timestamp, Source IP Address, Destination IP Address, Source Port, and Destination Port. All the other 25 columns are subsets of the five main ones listed above. Below is a summary of the statistical details including the data types of the dataset:

1. **Timestamp (Numerical):** 39,997 entries, representing the time when each attack occurred.
2. **IP Addresses (Text/Numerical):**
- Source IP Addresses: 40,000 entries.
- Destination IP Addresses: 40,000 entries
3. **Ports (Numerical):**
- Source Ports: 29,761 entries.
- Destination Ports: 29,895 entries
4. **Protocol (Categorical):** 3 types - TCP, UDP, ICMP
5. **Packet Details (Numerical/Categorical):**
- Packet Length: 1,437 types
- Packet Genre: 2 types
6. **Traffic and Payload Data (Categorical/Text):**
- Traffic Types: 3 types
- Payload Data: 40,000 entries
7. **Attack Indicators (Numerical/Categorical):**
- Malware Indices
- Anomaly Scores: 9,826 varieties
- Alerts
- Attack Modalities: 3 types.
- Unique Signatures
8. **Response and Impact (Categorical/Numerical):**
- Countermeasures
- Threat Severity
- Intrusion Detection/Alerts
9. **User & Infrastructure Data (Categorical/Numerical/Text):**
- User Profiles: 32,389 entries
- Device Metrics: 32,104 entries
- Network Sections
- Geographical Markers: 8,723 types
- Proxy Logs: 20,148 entries
- Firewall Narratives
- Log Sources

Rationale for Selecting this Dataset:

This dataset was selected because it covers a wide range of attributes related to cyber security attacks in detail. It spans almost four years, from January 1, 2020, to October 11, 2023, allowing us to analyse trends and patterns in cyber-attacks over a long period.

The dataset includes various attributes like IP addresses, ports, protocols, packet details, traffic and payload data, indicators of attacks, response and impact metrics, and information about users and infrastructure. This diverse set of attributes gives us a comprehensive view of cyber-attacks from multiple angles. With this information, we can identify recurring patterns, observe how cyber strategies evolve over time, pinpoint potential sources of threats, and gain insights to improve cyber defences.

Additionally, the dataset is quite large, containing 40,000 entries. This gives us a substantial amount of data for conducting robust statistical analysis and recognizing patterns effectively. Overall, the comprehensiveness of the dataset aligns well with our goal of developing visualisations that would help in studying cyber-attacks in depth, identifying patterns, and ultimately enhancing cyber security measures.

Data pre-processing steps applied and their benefits.

Several data pre-processing techniques were employed for to prepare the dataset for analysis. These techniques were essential as they ensured that the data was clean, of good quality and usable for the analysis stage. The key pre-processing techniques applied include:

Checking/Removing Duplicate Records:

Having duplicate values in a dataset can lead to bias in analysis , data redundancy, misinterpretations of patterns and pose data integrity issues. Hence, the first pre-processing technique applied to the dataset was checking for duplicate records using the '**duplicated()'** method from the Pandas library. The total number of duplicates were then counted using `**num_duplicates = duplicates.sum()**'.

Although the results from the analysis showed that there were no duplicates, the importance of checking for duplicates cannot be overlooked. Some of the benefits of this pre-processing technique include:

- Ensures each observation in the dataset is unique, preventing redundancy and inaccuracies in analysis.
- Maintains data integrity by removing duplicate entries that could skew analysis results.
- Improves the efficiency of downstream analyses by providing a clean dataset without redundant information.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from wordcloud import WordCloud

# Read the dataset (csv file) from Kaggle
df = pd.read_csv("C:/Users/djaba/Downloads/Cybersecurity Dataset/cybersecurity_attacks.csv")
# Get the statistics of all the columns and rows in the dataset
print("The total number of columns:", df.shape[1])
print("The total number of rows:", df.shape[0])
print(df.head(10))
print(df.describe(include=['object']))

# Data Cleaning
# Checking for duplicates
duplicates = df.duplicated()

# Counting the total number of duplicates
num_duplicates = duplicates.sum()
print("Number of duplicates:", num_duplicates)
print("Rows with duplicates:")
print(df[duplicates])
```

Figure 1. Snippet of code for checking and removing duplicate records.

Data Type Identification

The data type of each column in the dataset was examined using the **`df[column_name].dtype`** function, which returned the data type of each column. Subsequently, the script printed whether each column was categorical or numeric based on its data type.

Identifying the data types of columns is a crucial aspect of data preprocessing. By understanding the nature of the data in each column, it becomes easier to apply appropriate techniques for handling and transforming the data. For instance, categorical data often requires encoding into numerical format for machine learning algorithms to process effectively. On the other hand, numerical data may need scaling or normalization to ensure that the range of values does not bias the model's learning process. By accurately identifying data types, potential errors or inconsistencies in the preprocessing stage can be mitigated, leading to more reliable and robust analyses.

```python
# Checking the datatype of each column
for column_name in df.columns:
    data_type = df[column_name].dtype
    is_categorical = isinstance(data_type, pd.CategoricalDtype)
    is_numeric = pd.api.types.is_numeric_dtype(data_type)
    print(f"Column '{column_name}' - Data Type: {data_type}, Categorical: {is_categorical}, Numeric: {is_numeric}")
```

Figure 2. Snippet of code for the identification of data types of the various columns.

Handling Missing Data:

The dataset underwent thorough preprocessing steps to handle missing values effectively. Initially, the dataset was checked for missing values using the **`df.isnull().sum()`** function from Python's Pandas library. This provided a comprehensive count of missing values for each column. Following this initial assessment, the columns with missing values were identified, including 'Malware Indicators', 'Alerts/Warnings', 'Proxy Information', 'Firewall Logs', and 'IDS/IPS Alerts'.

To address the missing values, a common strategy involved filling them with the mode (most frequent value) of the respective columns using **`df['column_name'].fillna(df['column_name'].mode()[0])`**. This approach is particularly suitable for categorical or ordinal data, as it replaces missing values with the most frequently occurring value, providing a reasonable approximation.

Handling missing data is paramount as it helps maintain the integrity and accuracy of the dataset. By addressing missing values appropriately, the preprocessing step aims to prevent the introduction of biases or inaccuracies in subsequent analysis or modelling tasks. Ultimately, this enhances the overall quality and reliability of the dataset, facilitating more robust and meaningful insights.

```
#Check for Missing Values
missing_values = df.isnull().sum()
print("Missing values count for each column:", missing_values)

# Checking the datatype of each column
for column_name in df.columns:
    data_type = df[column_name].dtype
    is_categorical = isinstance(data_type, pd.CategoricalDtype)
    is_numeric = pd.api.types.is_numeric_dtype(data_type)
    print(f"Column '{column_name}' - Data Type: {data_type}, Categorical: {is_categorical}, Numeric: {is_numeric}")

# Inputting the missing values with mode
df['Malware Indicators'] = df['Malware Indicators'].fillna(df['Malware Indicators'].mode()[0])
df['Alerts/Warnings'] = df['Alerts/Warnings'].fillna(df['Alerts/Warnings'].mode()[0])
df['Proxy Information'] = df['Proxy Information'].fillna(df['Proxy Information'].mode()[0])
df['Firewall Logs'] = df['Firewall Logs'].fillna(df['Firewall Logs'].mode()[0])
df['IDS/IPS Alerts'] = df['IDS/IPS Alerts'].fillna(df['IDS/IPS Alerts'].mode()[0])
```

Figure 3. Snippet of code for handling the missing data.

Feature Selection

Feature selection is a process used to identify the most relevant columns or attributes in a dataset that an analyst presumes to relevant for their analysis (Raj, Kamel and Lafata, 2022). In this project, the 'Payload Data' column from the dataset was dropped using the function *`df = df.drop('Payload Data', axis=1)`*. This step was undertaken because the 'Payload Data' column was deemed irrelevant to the analysis process.

In data analysis and machine learning, feature selection provides enormous benefits:

- Removing irrelevant or redundant features can improve the computational efficiency of analyses or models, reduce overfitting, and enhance the interpretability of the results. (Raj, Kamel and Lafata, 2022).
- Feature selection can increase the accuracy of predictive models and training times for algorithm. This is because less data means algorithms can be trained quickly (Aretov Inc, 2020)

```
# Drop Payload column (irrelevant)
df = df.drop( labels: 'Payload Data', axis=1)
```

Figure 4. Snippet of code demonstrating how the Payload column was dropped as an example of feature selection.

Dropping Columns with Excessive Missing Values:

After identifying the columns with null or missing values, a decision was made to drop columns with more than 90% of their values missing. This step was taken to ensure the dataset's integrity and accuracy. Using the `df_cleaned = df.dropna(thresh=len(df) * 0.1, axis=1)` function, columns with 90% or more missing data were effectively removed from the dataset.

Dropping columns with excessive missing values can significantly benefit data analysis processes. Imputing large amounts of missing data may introduce biases or inaccuracies, potentially leading to unreliable results. By removing columns with a high percentage of missing values, the analysis can focus on the remaining features, which are completer and more reliable for further exploration and modelling.

```
# Drop column if it has more than 90% of its data missing
df_cleaned = df.dropna(thresh=len(df) * 0.1, axis=1)
```

Figure 5. Snippet of code for further handling of columns with excessive missing values.

Data Visualization

In today's digital world, cyber-attacks pose a constant threat, challenging security, and stability. As these threats rapidly evolve, our aim is to analyse a comprehensive dataset spanning nearly four years. This extensive dataset, obtained from Kaggle's Cyber Security Attacks repository, covers the

period from January 1st, 2020, to October 11th, 2023, and consists of 40,000 entries across five key attributes.

Through this analysis, we seek to uncover the core characteristics of cyber-attacks, identify recurring patterns, and understand how these threats have evolved over time. The dataset provides a panoramic view of cyber-attack dimensions, including timestamps, source and destination IP addresses, ports, protocols, packet details, traffic and payload data, attack indicators, and response and impact metrics.

This report aims to conduct an in-depth examination of cyber-attacks, dissecting patterns, tracking the evolution of cyber strategies, pinpointing recurrent sources of threats, and ultimately synthesising insights to strengthen future cyber defences. By creating a series of visualisations, we will illuminate the intricacies of these cyber threats, empowering stakeholders to make informed decisions and fortify their digital security measures.

1. **Word Cloud Visualization:**

Word cloud visualizations offer an engaging way to uncover prevalent themes and patterns within textual data. By visually representing the frequency of words or terms, they provide valuable insights into the underlying structure of the dataset [3].

To create this visualization, I extracted text data from key columns such as 'Attack Type', 'Malware Indicators', 'Firewall Logs', 'Log Source', and 'Geo-location Data'. These columns were concatenated into a single string, which served as the input for generating the word cloud. Utilizing the `Word Cloud` library, a visual representation was produced where the size of each word corresponds to its frequency in the combined text.

```
# Word Cloud Visualisation
# Obtain the columns from the DataFrame
columns = ['Attack Type', 'Malware Indicators','Firewall Logs','Log Source','Geo-location Data']

# Obtain the data from the selected columns
data = df[columns]

# Convert the data to a string format
text = " ".join(str(value) for column in data.columns for value in data[column])

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

# Display the word cloud
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Figure 6.Snippet of code that shows the implementation of the Word cloud visualisation.

The resulting word cloud offers a visual snapshot of the most prevalent words or phrases associated with cyber security attacks. It highlights specific attack types, malware indicators, geographical locations, and log sources that occur frequently within the dataset. This allows for a quick identification of dominant themes and focal points within the data.

While word clouds serve as effective tools for identifying common terms, they should be interpreted with caution. They may oversimplify complex relationships and lack the nuance required for in-depth analysis. Additionally, the selection of columns for text extraction may influence the composition of the word cloud, potentially biasing the visualization towards certain aspects of the dataset. Therefore, it is essential to complement word cloud analysis with other techniques to gain a comprehensive understanding of the dataset [3].

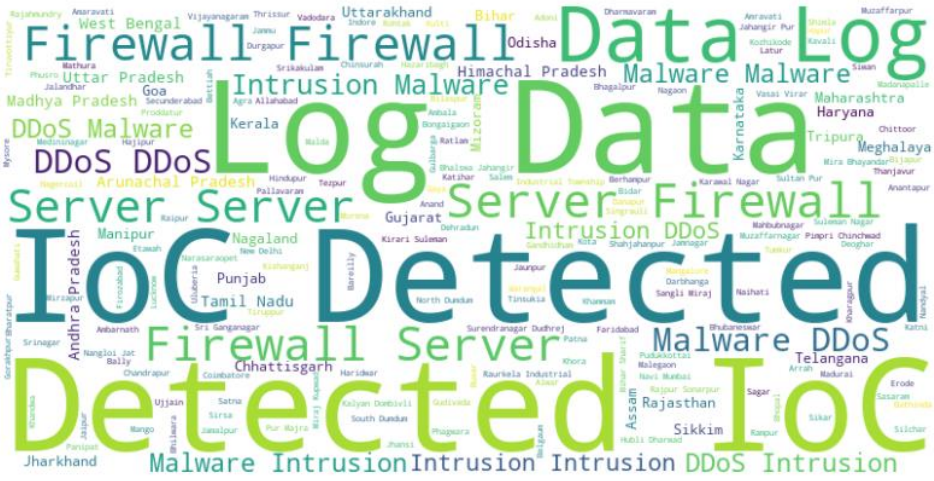

Figure 7. Image displaying the Word Cloud visualisation.

2. <u>Pie Chart: Distribution of Attack Types:</u>

Pie charts serve as effective visualizations for illustrating the distribution of categorical data, such as different types of cyber-attacks within a dataset [5].They offer a straightforward way to showcase the relative proportions of each attack type, enabling quick comparisons and insights into the prevalence of various attack modalities.

Visualizing the distribution of attack types is crucial for understanding the landscape of cyber threats present in the dataset. By highlighting the most prevalent attack types, stakeholders can prioritize mitigation strategies and allocate resources effectively to address the most significant threats.

To create the pie chart, the **`value_counts()`** function was utilized on the 'Attack Type' column of the dataframe **(`df['Attack Type'].value_counts()`).** This provided the frequency counts of each attack type. Subsequently, the pie chart was plotted using Matplotlib's **`plt.pie()`** function, customizing it with appropriate colours and an exploded slice for emphasis.

```
# Visualisation of Attack Analysis
import matplotlib.pyplot as plt
import seaborn as sns
attack_type_counts = df['Attack Type'].value_counts()
print("Distribution of Attack Types:", attack_type_counts)

# Obtain data for the pie chart
sizes = df['Attack Type'].value_counts()
# Setting custom colours and explode parameter
custom_colours = sns.color_palette('Accent')
explode = [0] * (len(attack_type_counts) - 1) + [0.1]
# Plotting the different distributions of the attack analysis in a pie chart
plt.figure(figsize=(8, 8))
plt.pie(attack_type_counts, labels=attack_type_counts.index, autopct='%1.1f%%', startangle=9,
        colors=custom_colours, explode=explode)
plt.title( label: 'Distribution of Attack Types', fontsize=20, fontweight='bold')
plt.axis('equal')
plt.show()
```

Figure 8. Snippet of code that shows the implementation of the Pie Chart visualisation.

The resulting pie chart offers a clear visualization of the distribution of attack types, with each slice representing a different attack modality. The pie is divided into three slices, each representing one of the attack types: DDoS (green, 33.6%), malware (purple, 33.3%), and intrusion (orange, 32.2%).

The pie chart shows that DDoS and malware attacks have nearly equal proportions, accounting for approximately one-third of the total recorded incidents each. Intrusion attacks follow closely behind, making up slightly less than one-third of the incidents and the lowest of incident occurrence.

By observing the relative sizes of the slices, viewers and stakeholders can easily discern the most common attack types as well as those that occur less frequently. This enables stakeholders to gain insights into the predominant threats within the dataset at a glance.

While pie charts are effective for comparing proportions of categorical data, they have limitations, particularly when the number of categories becomes large. As the number of attack types increases, the pie chart may become cluttered and difficult to interpret [16]. In such cases, alternative visualizations, such as bar charts or stacked bar charts, may be more suitable for presenting the distribution of attack types in a clearer and more concise manner. Therefore, it is essential to consider the complexity of the data and choose the appropriate visualization technique accordingly to ensure effective communication of insights.

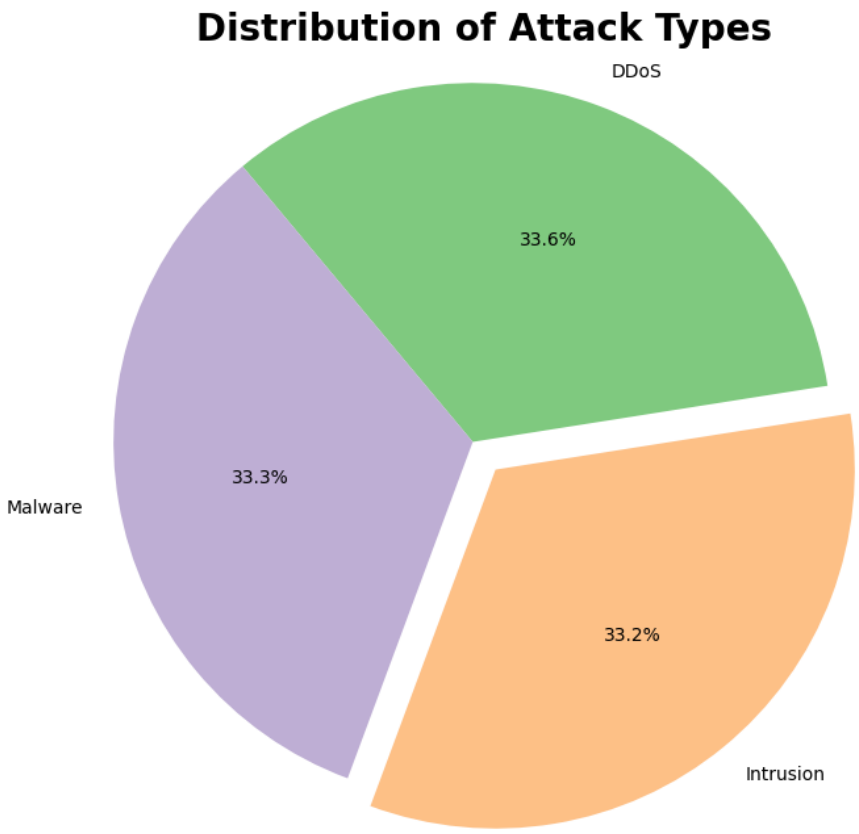

Figure 9. Image displaying the Pie Chart visualisation.

3. **Point Plot: Severity Levels of Attacks:**

The visualization of severity levels of attacks offers valuable insights into the potential impact or severity associated with different attack types, aiding in risk assessment and prioritization efforts. Understanding the severity levels of attacks is essential for assessing their potential impact on systems, networks, and organizations. By visualizing severity levels, stakeholders can identify which types of attacks pose the greatest risks and allocate resources accordingly to mitigate these threats effectively.

To create the point plot, the frequency counts of severity levels were first obtained using the value_counts() function applied to the 'Severity Level' column (df['Severity Level'].value_counts()). Next, a copy of the DataFrame was created to avoid modifying the original data **(df_copy = df.copy()).** The categorical variable 'Severity Level' was then encoded into numerical values using

factorization (df_copy["Severity Level"] = pd.factorize(df["Severity Level"])[0]), ensuring that it could be plotted effectively.

The sns.pointplot() function from the Seaborn library was utilized to generate the point plot, with attack types represented on the y-axis and severity levels on the x-axis. The resulting plot provides a visual representation of the distribution of severity levels for each attack type, with data points coloured by attack type. By examining the position and spread of the data points, viewers can discern which attack types tend to have higher or lower severity levels on average, as well as the variability within each attack type.

```
# Severity levels of the attack in a line chart
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Obtain the counts of the severity levels
severity_level_counts = df['Severity Level'].value_counts()
print("\nDistribution of Severity Levels:", severity_level_counts)

# Create a copy of the DataFrame to avoid modifying the original
df_copy = df.copy()
# encoding the categorical variables into numerical values
df_copy["Severity Level"] = pd.factorize(df["Severity Level"])[0]

# Creating point plots for the severity levels.
sns.pointplot(data=df_copy, x="Severity Level", y="Attack Type", hue="Attack Type", markers="o", linestyles="")
plt.title("Severity Level vs Attack Type")
plt.xlabel("Severity Level")
plt.ylabel("Attack Type")
plt.legend(title="Attack Type", loc="upper right")
plt.show()
```

Figure 10. Snippet of code showing the implementation of the point plot visualisation.

The image shows that malware attacks generally have the highest severity levels, followed by DDoS (Distributed Denial of Service) attacks, while intrusion attacks tend to have the lowest severity levels. This insight enables stakeholders to prioritize their response efforts, focusing on addressing the most severe threats first.

While point plots are suitable for visualizing the relationship between categorical and continuous variables, the current implementation may not be the most effective for presenting the distribution of severity levels across attack types. Since severity levels are categorical, a bar chart, stacked bar chart, or a box plot might be more appropriate for clearly depicting the distribution and allowing for easier comparison between attack types [16]

Additionally, the point plot could be enhanced by incorporating confidence intervals or error bars to provide a sense of the variability or uncertainty in the severity levels for each attack type. This would further strengthen the insights derived from the visualization and aid in more informed decision-making.

It is essential to carefully consider the nature of the data and the specific goals of the analysis when selecting the most suitable visualization technique. Appropriate chart types should be chosen to

effectively communicate insights to stakeholders and facilitate data-driven decision-making processes.

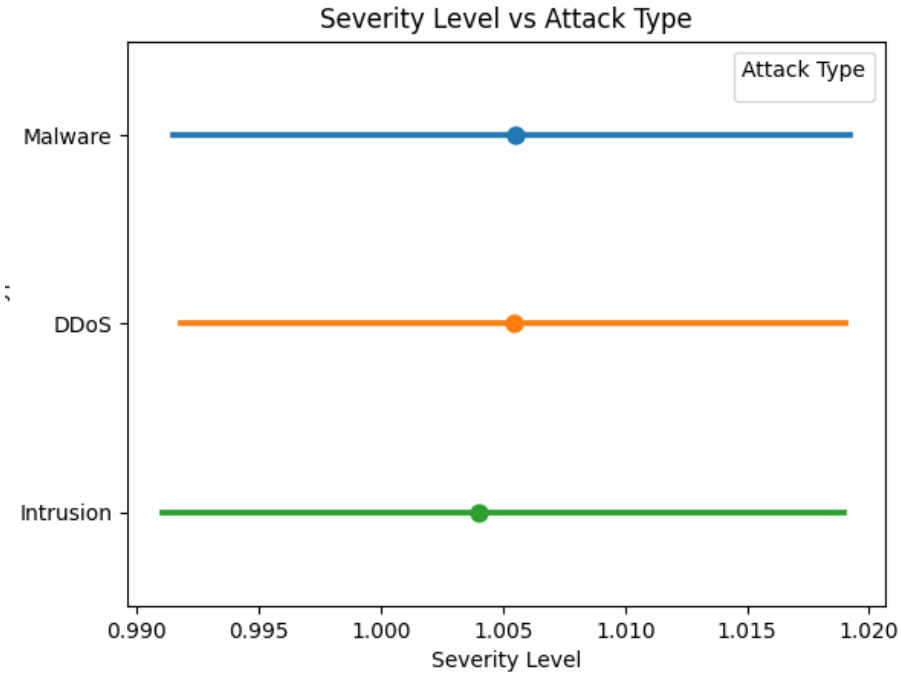

Figure 11. Point plot showing the severity level vs attack type.

4. Heatmap: Analysing the Relationship between Attack Type and Severity Level:

Understanding the relationship between the different attack types and severity levels is crucial for assessing the potential impact or risk associated with different attack modalities. By examining this relationship, stakeholders can gain insights into which types of attacks are more likely to result in severe consequences, enabling them to prioritize their security measures effectively.

To analyse this relationship, the **`pd.crosstab()`** function was used to create a cross-tabulation of attack types and severity levels. This allowed for the quantification of the frequency or counts of each combination of attack type and severity level. Subsequently, the results from the cross-tabulation results were visualized using **`sns.heatmap()`,** which generated a heatmap highlighting the distribution of attack types across different severity levels.

```
# Analysing the relationship between the Attack Type and Severity level
# Creating a cross-tabulation of Attack Type and Severity Level
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
cross_tab = pd.crosstab(df['Attack Type'], df['Severity Level'])

# Plotting a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(cross_tab, annot=True, cmap='coolwarm', fmt='d')
plt.title('Relationship between Attack Type and Severity Level')
plt.xlabel('Severity Level')
plt.ylabel('Attack Type')
plt.show()
```

Figure 12. Snippet of code showing the implementation of heatmap visualisation

The resulting heatmap provides a comprehensive overview of how attack types are distributed across various severity levels. Darker colours on the heatmap indicate higher frequencies of specific combinations of attack types and severity levels, whereas lighter colours represent lower frequencies. This visualization allows stakeholders to quickly identify which combinations are most common and which may require closer attention or mitigation efforts. In the resulting visualisation, the x-axis is divided into three sections representing the severity levels: High, Low, and Medium.

Within each severity level section, there are three bars representing the different attack types: DDoS (red), intrusion (orange), and malware (blue). The height of each bar corresponds to the count or number of incidents for that specific attack type and severity level combination.

For the High severity level, DDoS attacks have the highest count, with a bar height of 4523. This indicates that DDoS attacks are the most prevalent in the high severity category.

In the Low severity level section, intrusion attacks have the tallest bar, with a count of 4374, suggesting that intrusion attacks are more common among low-severity incidents in this data.

Lastly, for the medium severity level, malware attacks have the highest count, represented by a bar height of 4516, implying that malware incidents are predominant in the medium severity range.

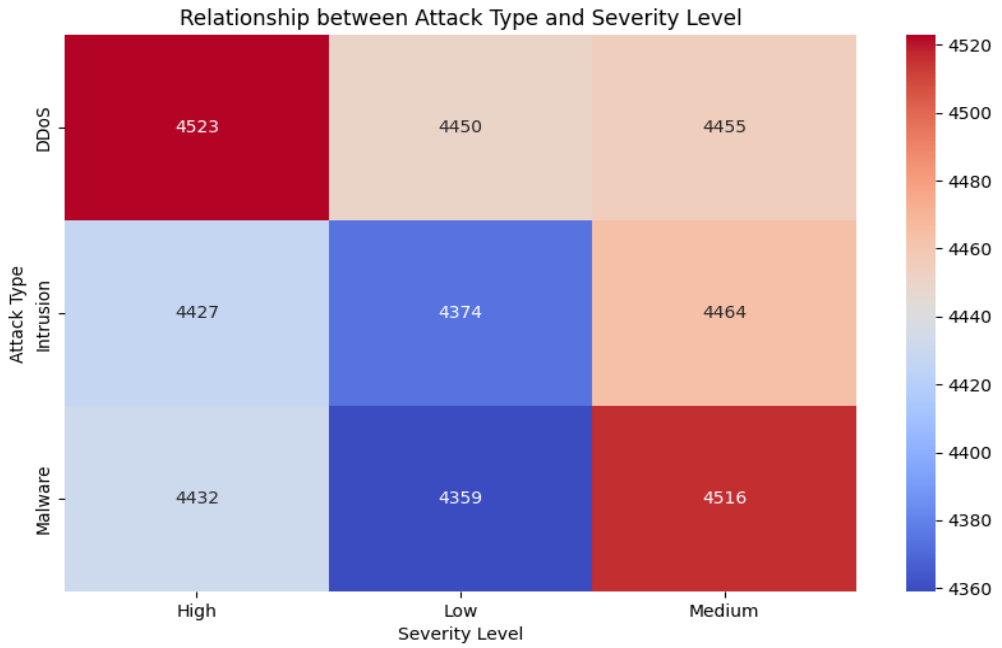

Figure 13. Heatmap showing the relationship between the different attack types and their severity levels.

Heatmaps are effective tools for visualizing the relationship between two categorical variables, such as attack types and severity levels. However, their interpretation may become challenging if there are too many categories or if the colour scale used is not intuitive. To mitigate these challenges, it is essential to carefully select the colour scale and ensure that the heatmap remains easy to interpret even with a large amount of data. Additionally, providing appropriate annotations or labels can enhance the clarity of the heatmap and aid in its interpretation by stakeholders.

5. **Line Chart: Number of Attacks per Month (Temporal Analysis):**

The exploration of temporal trends in cyber-attacks aims to uncover underlying patterns and variations in attack frequencies over time. This analysis provides valuable insights into the dynamics of cyber threats, aiding in the development of proactive defence strategies and implementing mitigation measures.

To conduct temporal analysis, the 'Timestamp' column was first converted into a datetime series, enabling precise time-based calculations. Subsequently, the dataset was resampled by using the monthly data monthly with the **`df.resample('M', on='Timestamp')`** function, effectively aggregating the attacks into monthly intervals. The resulting time series data, representing the count of attacks per month, was then visualized using a line chart generated with **`monthly_attacks.plot()`.**

```
# Performing Temporal Analysis to identify attack Patterns over time.
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Converting the Timestamp column to Pandas datetime series.
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
# Resample the data by month and count the number of attacks per month
monthly_attacks = df.resample( rule: 'ME', on='Timestamp')['Attack Type'].count()

# Plot the number of attacks by month
monthly_attacks.plot(color='orange', figsize=(10, 6))
plt.title('Number of Attacks per Month')
plt.ylabel('Number of Attacks')
plt.show()
```

Figure 14. Snippet of code showing the implementation of a line chart on the number of attacks in a month.

The resulting line chart illustrates the temporal distribution of cyber-attacks, highlighting fluctuations in attack frequencies across different months. By examining the plotted trend, analysts can discern patterns such as periodic spikes, seasonal variations, or long-term trends in attack activity. For this analysis, the results of the line chart depicts that most attacks were low in the month of January, this could be explored by stakeholders to find out why. Information from this investigation could be further used to make informed decisions.

This comprehensive view of temporal dynamics enhances the understanding of cyber threat landscapes and facilitates informed decision-making.

While line charts are well-suited in depicting temporal trends, their effectiveness is dependent on factors such as the time range covered by the dataset and the granularity of the timestamps. Monthly aggregation provides a broader perspective on attack frequencies but may obscure finer temporal nuances that may have occurred on a weekly basis or daily basis. Analysts should carefully consider the appropriate level of granularity based on the specific objectives of the analysis and the characteristics of the dataset.

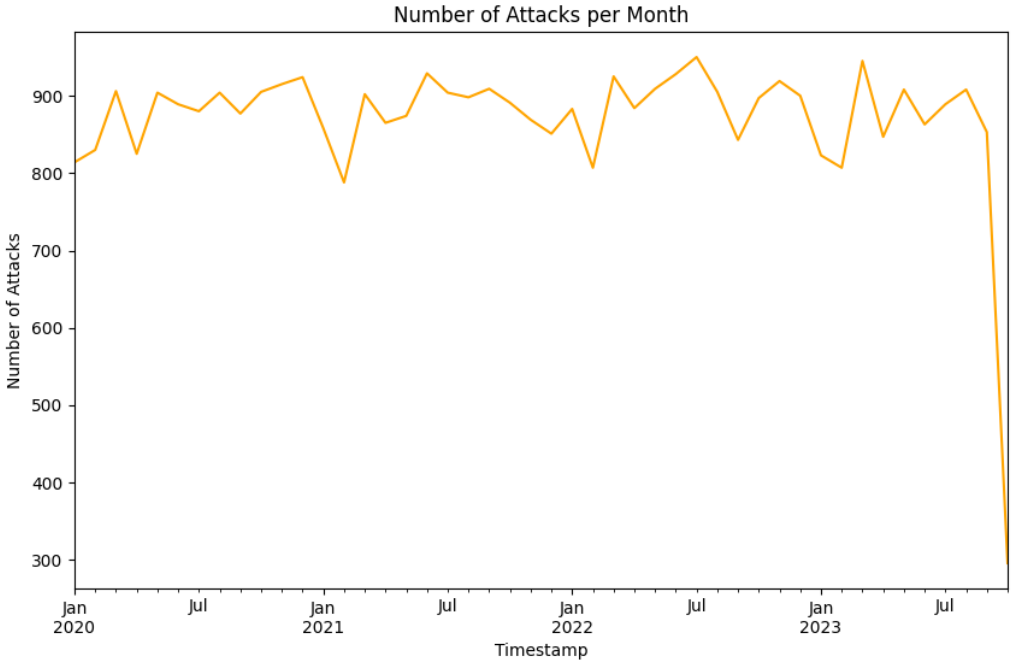

Figure 15. Line chart showing the monthly attacks over the period between 2020 – 2023.

6. **Box Plot: Anomaly Scores by Attack Type:**

Understanding the distribution of anomaly scores in relation to specific attack types is essential for uncovering potential patterns or anomalies indicative of cyber threats. Box plots, a graphical representation of the distribution of data, offer a visual means to explore the relationship between anomaly scores and attack modalities, facilitating anomaly detection and classification efforts.

Leveraging the **`sns.boxplot()`** function, a box plot was constructed with anomaly scores plotted on the x-axis and attack types categorized on the y-axis. Each attack type was colour-coded, enhancing visual differentiation and facilitating interpretation. Box plots succinctly summarize key statistical properties such as the median, quartiles, and potential outliers within each attack type category.

```
# Anomaly Scores on Attack Type
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='Anomaly Scores', y='Attack Type', hue='Attack Type', showfliers=False)
plt.title(f'Distribution of Anomaly Scores by Attack Type')
plt.xlabel( xlabel: 'Anomaly Scores', fontsize=14, fontweight='bold')
plt.ylabel( ylabel: 'Attack Type', fontsize=14, fontweight='bold')
plt.show()
```

Figure 16. Code snippet showing the implementation of the visualisation of Box plots on the anomaly scores by attack type.

The generated box plot provides a comprehensive overview of the distribution of anomaly scores across different attack types. The box plot visualisation shows the distribution of anomaly scores

grouped by attack type: malware, DDoS (Distributed Denial of Service), and intrusion. The x-axis represents the range of anomaly scores, and the length of each coloured bar indicates the frequency or number of anomalies within that score range for each attack type.

The malware attack type exhibits the highest anomaly scores, indicating that malware-related activities or events tend to be more anomalous or suspicious according to the system's detection mechanisms. DDoS attacks have a slightly lower range of anomaly scores, while intrusion attacks generally have the lowest anomaly scores among the three attack types.

This distribution suggests that the cybersecurity system or application is more effective at detecting and assigning higher anomaly scores to malware-related threats, followed by DDoS attacks, and then intrusion attempts.

This enables analysts to easily identify variations in central tendencies, spreads, and outlier presence across attack modalities, enabling targeted investigation into potential anomalous behaviour or trends associated with specific attack types.

Box plots serve as powerful tools for visualizing the spread and central tendency of numerical data across categorical variables [16]. However, the effectiveness of interpretation may be compromised in scenarios with excessive categories or when outliers dominate the visualization. To mitigate these challenges, careful consideration of data granularity and outlier handling strategies is paramount to ensure meaningful insights are derived from the visualization.

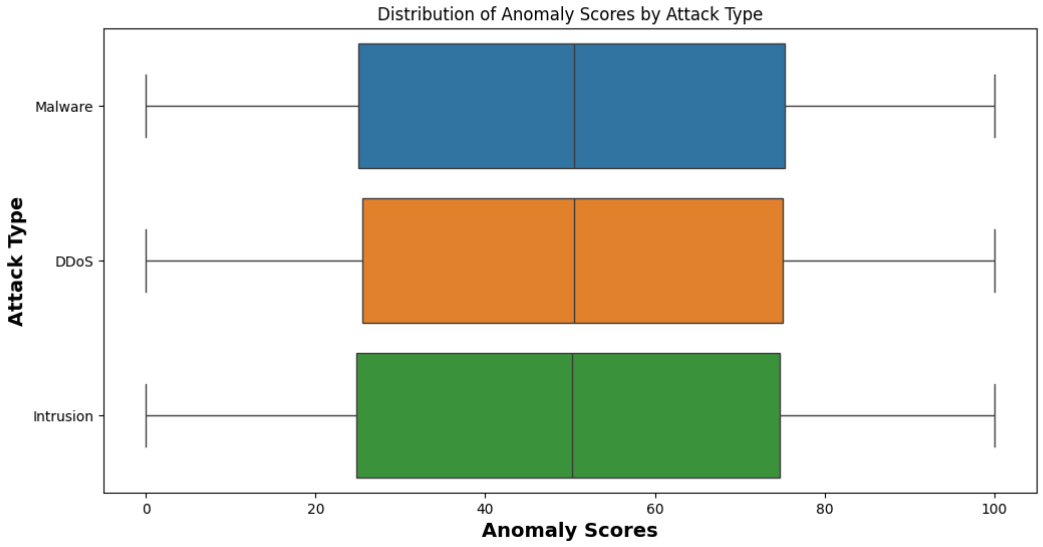

Figure 17. Box plot showing the distribution of anomaly scores across different attack types.

7. Count Plot: Attack Type & Action Taken

Understanding the interplay between attack types and the corresponding actions taken in response is crucial for evaluating the efficacy of incident response strategies and countermeasures. Count plots, a form of categorical plot, offer a straightforward method to visualize the distribution of

categorical variables and assess the frequency of different actions taken in response to various attack types.

Employing the **`sns.countplot()`** function, a count plot was generated with the 'Action Taken' column delineated on the x-axis. The count plot was further enhanced by color-coding each bar based on the corresponding 'Attack Type', facilitating easy comparison and interpretation. By aggregating the frequency of different actions taken across attack types, the count plot provides valuable insights into response patterns, potential gaps, and strategy effectiveness.

```
# Attack Type & Action Taken
import matplotlib.pyplot as plt
import seaborn as sns
sns.countplot(data=df, x='Action Taken', hue='Attack Type')
plt.show()
```

Figure 18. Code snippet showing the implementation of the count plot for the different actions taken to address the attack types.

The resulting count plot offers a comprehensive depiction of the frequency of various actions taken in response to different attack types**.** Analysts and stakeholders can readily view predominant response actions (e.g., blocking, logging, ignoring) for each attack type, enabling targeted evaluation of incident response protocols and potential areas for improvement.

Count plots serve as effective tools for visualizing categorical data and assessing frequency distributions. However, the utility of count plots may diminish in scenarios with many categories or imbalanced distributions, potentially obscuring nuanced insights. Careful consideration of data granularity and visualization design is essential to extract meaningful conclusions from count plot analyses.

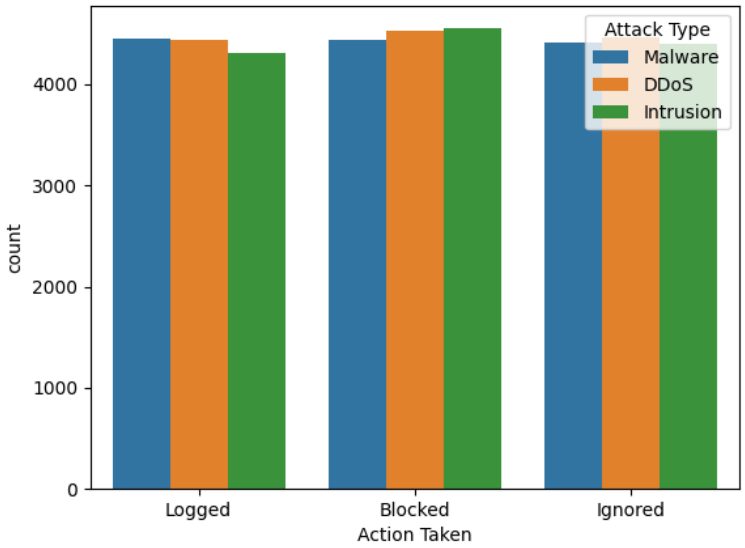

Figure 19. Count plot analysis showing the frequency of various actions taken in response to different attack types.

8. Scatter Plot: Anomaly Scores Over Time:

The scatter plot visualizes anomaly scores over time, offering insights into the occurrence of anomalous events and potential patterns indicative of cyber-attacks or security incidents.

To create the plot, a threshold value of 90 was defined for anomaly scores. Subsequently, a new binary column 'Anomaly' was generated based on whether the anomaly scores exceeded the threshold. The scatter plot was then constructed using **`sns.scatterplot()`,** with time on the x-axis and anomaly scores on the y-axis. The colour-coding was applied based on the 'Anomaly' column, distinguishing between scores that exceeded the threshold and those that did not.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
# Incident Response Analysis:
threshold = 90
df['Anomaly'] = df['Anomaly Scores'] > threshold
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Timestamp', y='Anomaly Scores', hue='Anomaly')
plt.title('Anomaly Scores Over Time')
plt.xlabel('Timestamp')
plt.ylabel('Anomaly Scores')
plt.show()
```

Figure 20.Code snippet showing the implementation of a scatter plot of anomaly scores over time.

The scatter plot effectively displays the distribution of anomaly scores over time. By differentiating points based on whether they surpass the defined threshold, the visualization highlights periods with elevated anomaly concentrations. This aids in identifying potential attack patterns or periods of heightened security risk. shows the anomaly scores over time for a cybersecurity system or application. In figure 21 below **t**he anomaly scores are represented by coloured dots, where orange dots indicate "False" anomalies (false positives), and blue dots represent "True" anomalies (actual anomalies or threats).

The y-axis represents the anomaly score, which is likely a metric or probability value indicating how anomalous or suspicious an event or activity is, with higher scores signifying more anomalous behaviour. The x-axis shows the timestamp over which these anomaly scores were recorded.

Throughout the observed time range from early 2020 to mid-2023, there is a dense concentration of blue dots, indicating a consistent presence of true anomalies or threats. However, there are also sporadic orange dots scattered among the blue ones, representing false positive anomalies or false alerts.

It is worth noting that the true anomalies (blue dots) appear to be more clustered and denser, suggesting periods or patterns of heightened threat activity or incidents. In contrast, the false positives (orange dots) seem more randomly distributed, which is expected for false alerts.

While scatter plots are useful for revealing relationships between two numerical variables, their efficacy can be influenced by factors such as the chosen threshold value and point density. High

point densities may obscure patterns, emphasizing the importance of choosing appropriate visualization parameters for optimal interpretation.

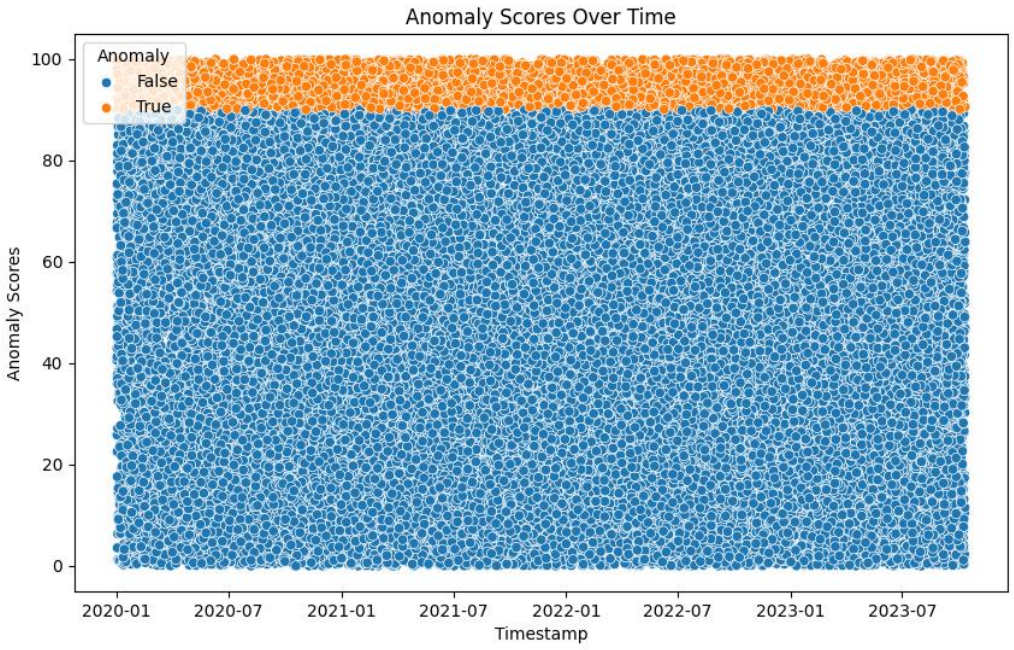

Figure 21. Scatter plot showing anomaly scores over time.

9. Horizontal Bar Chart: Top 10 most (security risk) alerted Users

The horizontal bar chart shows the top users ranked by how many security alerts they've triggered, specifically for "Blocked" or "Alerted" actions. The bars are ordered from highest to lowest, so the users at the top are the ones setting off the most of these types of alerts. This makes it easy to spot which users might need closer monitoring or investigation from the cybersecurity team, as they could potentially be insider threats or have risky behaviour patterns that need to be discussed. A colour gradient is used to differentiate the device bars, and the horizontal layout makes it easy to compare the alert counts side-by-side.

The code first checks what unique "Action Taken" values exist in the data, then filters it down to just the rows where the action was "Blocked" or "Alerted" since those indicate serious security concerns. It groups the filtered data by "User Information" and counts how many alerts each user has. Then it sorts those counts from highest to lowest to get the top 10 alert-prone users.
Some formatting is applied like setting the chart size to '**plt.figure(figsize=(10,7))',** using a colour gradient from '**plt.cm.viridis()',** and making the bars horizontal with '**kind='barh''** so the longest bars really stick out. The axes are also labelled accordingly.

```
# Check for unique actions + categorizing to identify suspicious ones
print('Columns from Action Taken:', df['Action Taken'].unique())
# Filtering out suspicious activities
suspicious_df = df[df['Action Taken'].isin(['Blocked', 'Alerted'])]

# Grouping the data by 'User Information' to count the number of alerts per user
user_alert_counts = df.groupby('User Information').size()

# Find the people with highest alert count
max_alert_count = user_alert_counts.max()

# Output the results of finding the people with highest alert count
print(f"The highest alert count for any user is: {max_alert_count}")

# finding the user(s) with this highest count
users_with_max_alerts = user_alert_counts[user_alert_counts == max_alert_count]
print("User(s) with the highest alert count:")
print(users_with_max_alerts)

# Sort the alert counts in descending order and getting the top 10
top_user_alert_counts = user_alert_counts.sort_values(ascending=False).head(10)

# Plotting the data
plt.figure(figsize=(10, 7))
colors = plt.cm.viridis(np.linspace( start: 0,  stop: 1, len(top_user_alert_counts)))  # Generate a color gradient for the bars
top_user_alert_counts.sort_values().plot(kind='barh', color=colors)  # Plot as a horizontal bar chart
plt.xlabel('Number of Alerts')
plt.ylabel('User Information- Names')
plt.title('Top 10 Users by Number of Alerts')
plt.grid( visible: True, linestyle='--', alpha=0.5)  # Add light grid lines for better readability
plt.show()
```

Figure 22. Code snippet showing the implementation of horizontal bar chart of the Top 10 most alerted Users.

While **Figure 23** this lets you quickly see which users are linked to the most security alerts and is very helpful for cybersecurity teams to identify high-risk users that may need extra training, policy changes, or other protective measures to reduce insider threats and damaging behaviour on the network. But it doesn't dig into what those alerts mean - like how severe they are or what type of threat they represent. Categorizing the alerts further could give more context on the level of risk each user poses. Investigating the device types with the most alerts is the best advised way to continue. This could help with discovering a pattern in faults.

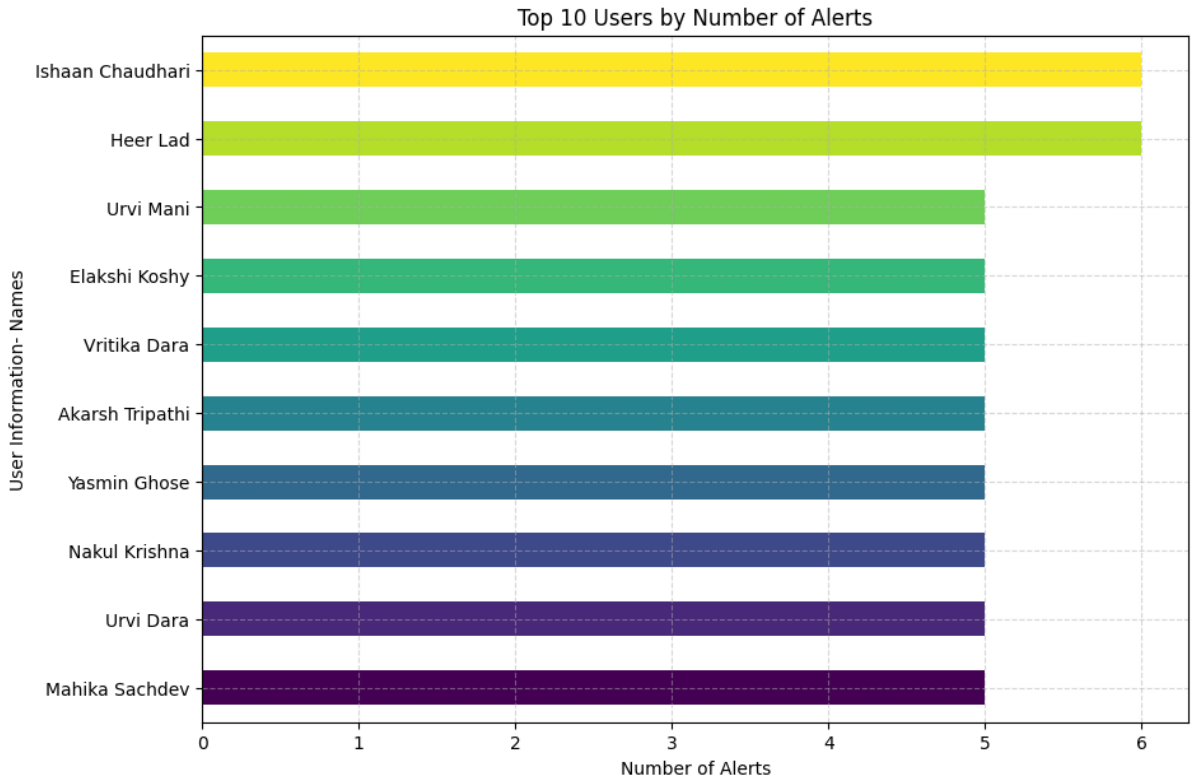

Figure 23. Horizontal bar chart showing the implementation of horizontal bar chart of the Top 10 most alerted Users.

10. Horizontal Bar Chart: Top 10 most (security risk) alerted Devices

The visualization presented in Figure 25 depicts the top devices triggering the most security alerts within a network. It highlights devices that have taken either the 'Blocked' or 'Alerted' action, providing a quick overview of potential security risks. The bar chart is ordered from highest to lowest, aiding cybersecurity teams in identifying the most critical devices that require attention.

The implementation involved grouping the data by 'Device Information' and counting the alerts associated with each device using **`df.groupby('Device Information').size()`.** The counts were then sorted in descending order to identify the top 10 device offenders. Visual styling techniques, such as setting the figure size **(`plt.figure(figsize=(10, 7))`)** and applying a colour gradient using **`plt.cm.viridis()`**, were employed. Additionally, **`kind='barh'`** was specified to create horizontal bars, and axis labels were added for clarity.

```
# Visualisation: Horizontal bar chart showing laptops with the highest number
of alerts count according to device model
# Count the occurrences of each device in the alerts
device_alert_counts = df.groupby('Device Information').size()

# Sort the alert counts in descending order and get the top devices
top_device_alert_counts =
device_alert_counts.sort_values(ascending=False).head(10)

# Plotting the data
plt.figure(figsize=(10, 7))
colors = plt.cm.viridis(np.linspace(0, 1, len(top_device_alert_counts)))
# Generate a color gradient for the bars
top_device_alert_counts.sort_values().plot(kind='barh', color=colors)
plt.xlabel('Number of Alerts')
plt.ylabel('Device Information')
plt.title('Top 10 Devices by Number of Alerts')
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```

Figure 24. Code snippet showing the implementation of horizontal bar chart of the Top 10 most alerted devices.

The resulting visualization illustrates that many devices generating the most alerts are using outdated Windows versions and Internet Explorer browsers, indicating potential vulnerabilities associated with legacy systems. While the visualization effectively identifies devices involved in alerts, it does not convey the severity of those alerts.

To enhance the analysis, future research could focus on identifying devices with the highest severity alerts to prioritize responsive action. Nevertheless, the chart serves as a valuable tool for informing strategic decisions in cybersecurity. By highlighting the need for an updated approach to address legacy systems, cybersecurity teams can consider upgrading systems, increasing monitoring of legacy devices, or implementing targeted security measures to mitigate risks associated with outdated software and hardware.

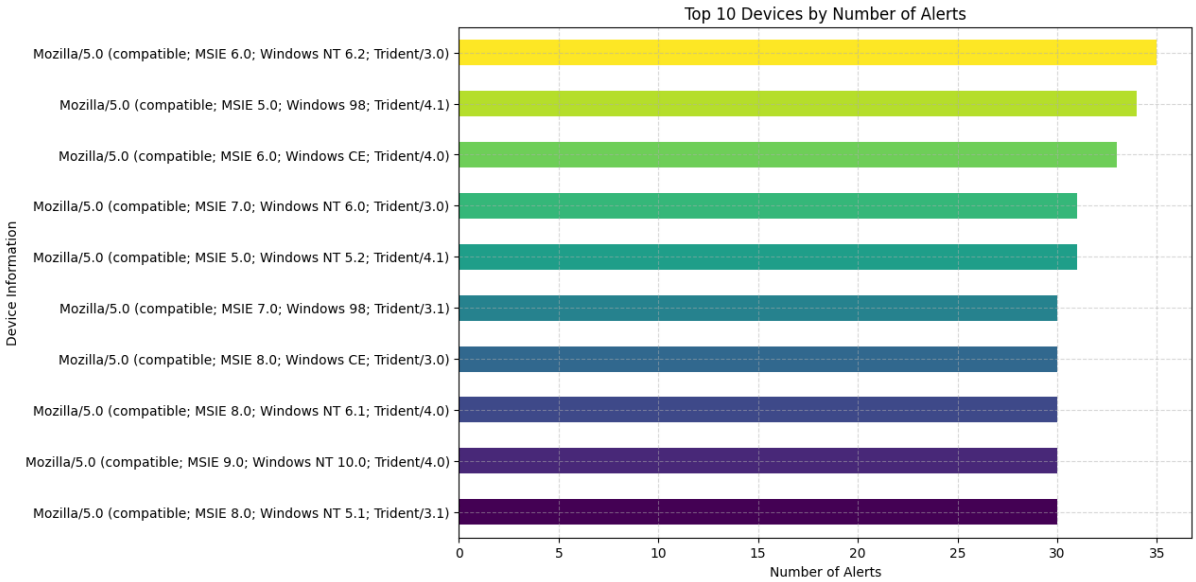

Figure 25. Horizontal bar chart showing the Top 10 most alerted devices in descending order.

11. Line Chart: Monthly alert counts according to Attack types from 2020 to 2023

This visualisation represents a timeline depicting the fluctuation of various types of security alerts from 2020 to 2023. The lines on the graph represent DDoS (blue), Intrusion (orange), Malware (green), and Unknown alerts (red).

Unknown alerts consistently exhibit a higher number of occurrences compared to the other types, typically ranging between 350 and 450 alerts per month. The remaining lines predominantly maintain a presence at a lower level, with alert counts ranging from 100 to 200 per month. However, a significant drop is observed around September 2023, indicating either a remarkably successful cybersecurity effort or a potential oversight in data recording during that period. Further investigation is necessary to discover the cause of this anomaly in the dataset.

Implementation involved ensuring accuracy of all timestamps and handling missing values in the 'Attack Type' column with 'Unknown' using **`data["Attack Type"].fillna("Unknown", inplace=True)`** was coded. Subsequently, frequency of each alert type on a monthly basis was obtained using **`alerts_over_time = data.resample('M', on="Timestamp")["Attack Type"].value_counts().unstack(fill_value=0)`.**

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

# Load the data
data = pd.read_excel("/Applications/Quant /Data processed.xlsx")

# Ensure the timestamp is in datetime format and drop rows with null timestamps
data["Timestamp"] = pd.to_datetime(data["Timestamp"], errors='coerce')
data.dropna(subset=["Timestamp"], inplace=True)

# Ensure the "Attack Type" column doesn't contain null values
data["Attack Type"].fillna("Unknown", inplace=True) # Replace null values with "Unknown"

# Resample data to get the monthly count of alerts/warnings for each attack type
alerts_over_time = data.resample('M', on="Timestamp")["Attack Type"].value_counts().unstack(fill_value=0)

# Create a line plot for each attack type with monthly resampling
alerts_over_time.plot(kind='line', figsize=(12, 6), lw=2)

plt.xlabel("Month")
plt.ylabel("Number of Alerts")
plt.title("Alert Types Over Time (Monthly)")
plt.show()
```

Figure 26. Code snippet showing the implementation of a line chart of the Monthly alert counts according to Attack types from 2020 to 2023.

While the visualization does not provide insight into the severity or specific nature of each alert type, it serves as a valuable tool for identifying periods of heightened alert activity. This allows cybersecurity analysts to focus their efforts on investigating and addressing potential security threats during those timeframes. A noticeable dip can be seen towards the end of Jul 2023, this suggests a significant decrease in the number of alerts recorded for that period. There are many reasons this could happen, it could be system outages or logging malfunctions that interrupt standard recording methods of incidents, or it could be caused by reporting issues or delays, with alerts not being logged in real-time. The last possibility is the successful implementation of security measures preventing cyber threats

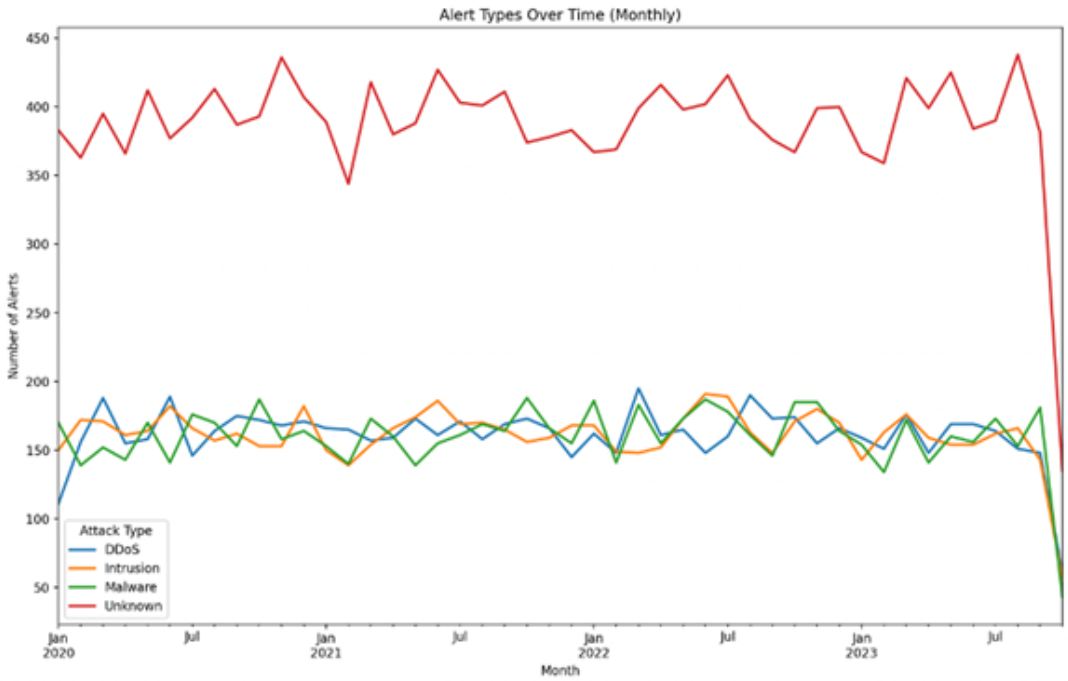

Figure 27. Line chart showing the monthly alert counts according to Attack types from 2020 to 2023, with DDoS (blue), Intrusion (orange), Malware (green), and Unknown alerts (red).

12. Stacked bar plot for Traffic Type and protocol:

The stacked bar plot for "Traffic Type and Protocol" offers significant insights into network behaviour by showing the relationship between traffic types and various protocols. The x-axis hosts the traffic types, categorised into DNS, FTP, and HTTP, while the y-axis shows the total count of protocols within each traffic type. The stacked bars are colour-coded to represent different protocols, such as ICMP, TCP, and UDP. The height of each segment within the bar correlates with the count of that protocol within its respective traffic type.

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


# Create a pivot table for traffic type and protocol
protocol_traffic_pivot = data.pivot_table(index="Traffic Type", columns="Protocol", aggfunc='count',
values="Timestamp")

# Create a stacked bar plot
protocol_traffic_pivot.plot(kind='bar', stacked=True, figsize=(12, 6))

plt.xlabel("Traffic Type")
plt.ylabel("Count")
plt.title("Stacked Bar Plot for Traffic Type and Protocol")
plt.show()
```

Figure 28. Code snippet showing the implementation of the visualisation for the Stacked bar plot for traffic type and protocol.

A notable observation is that UDP is the dominant protocol across all three traffic types, with TCP following closely behind, while ICMP has a smaller representation. This pattern is consistent across DNS, FTP, and HTTP traffic types. However, the proportional differences among traffic types reveal

unique characteristics; for instance, FTP traffic shows a higher proportion of TCP compared to DNS, which predominantly features UDP.

The overall distribution of protocols across these traffic types suggests a diverse range of protocol usage in the network. The consistent presence of all three protocols might indicate typical network behaviour or a broad range of applications that use these protocols. The dominance of UDP could reflect its prevalent use in DNS queries and real-time applications, while TCP's significant representation in FTP traffic aligns with its reliable data transmission requirements.

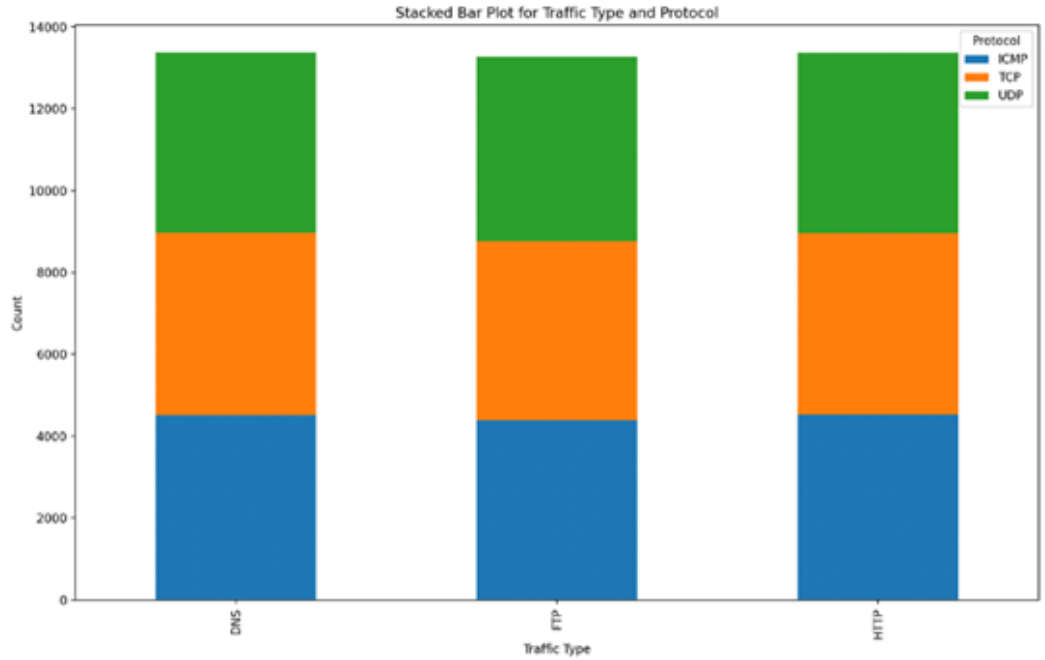

Figure 29. Stacked bar plot showing the different distribution of traffic types and protocols.

13. Packet types and actions taken:

The stacked bar plot for "Packet Types and Actions Taken (Cleaned)" offers crucial insights into how different packet types are processed and the various actions applied to them. The x-axis represents the two packet types, "Data" and "Control," while the y-axis indicates the number of packets within each type. The stacked bars are colour-coded to show the different actions taken on the packets, including "Unknown," "Ignored," "Blocked," and "Logged." The height of each segment in the bar correlates with the count of that action within its respective packet type.

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

# Load the data
data = pd.read_excel("/Applications/Quant /Data processed.xlsx")

# Remove rows with blank or NaN 'Packet Type'
data = data[data['Packet Type'].notna() & (data['Packet Type'] != '')]

# Ensure 'Action Taken' has no blanks or NaN
data['Action Taken'].fillna('Unknown', inplace=True)

# Stacked bar plot for packet types and actions taken (without blanks)
plt.figure(figsize=(12, 6))
sns.histplot(data, x='Packet Type', hue='Action Taken', multiple='stack', shrink=0.8, palette='pastel')
plt.title('Packet Types and Actions Taken (Cleaned)')
plt.xlabel('Packet Type')
plt.ylabel('Count')
plt.show()
```

Figure 30. Code snippet showing the implementation of the visualisation for the heat map for packet type and actions taken.

An intriguing observation is the prevalence of "Unknown" actions in both "Data" and "Control" packet types, suggesting that a significant number of packets are either unclassified or require further investigation. The distribution of "Ignored," "Blocked," and "Logged" is relatively even, indicating that these actions are applied consistently across both packet types. These patterns suggest a uniform security policy for handling network traffic.

The overall distribution of actions across these packet types indicates a consistent approach to processing network data. The high proportion of "Unknown" actions might indicate gaps in packet classification or point to areas that need further analysis to understand why these packets are marked as "Unknown." The even distribution of "Ignored," "Blocked," and "Logged" across both "Data" and "Control" packet types reflects a structured approach to security and packet management.

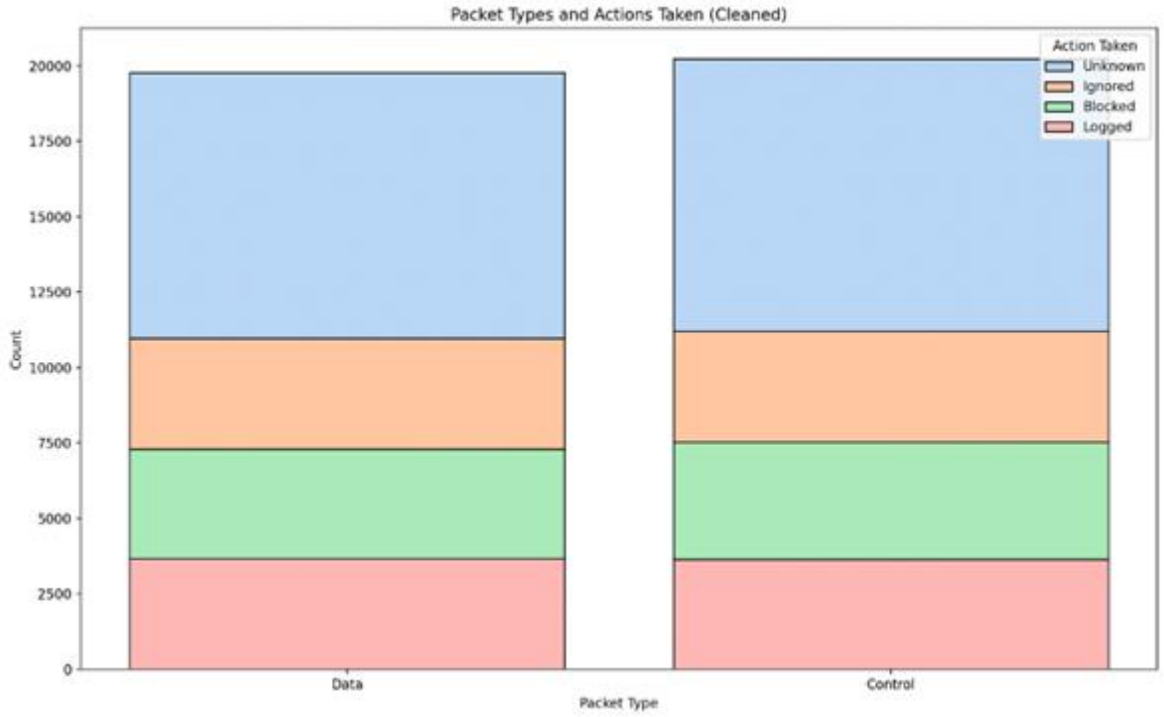

Figure 31. Heatmap showing the relationship between the different packet types and the actions taken to handle them.

14. Violin Plot

The plot_violin function generates a violin plot, an advanced visualization illustrating the distribution of a numerical dataset across different categories. Utilizing the seaborn library, known for its user-friendly interface for statistical graphics, this function accepts a DataFrame, a categorical column, and a numerical column. The plot offers insight into the numerical variable's distribution within the defined categories. Setting the figure size ensures readability, while the violinplot function creates the main plot, with palette set to "muted" for softer colours. Overlaying a swarmplot adds individual data points, aiding in outlier identification. The plot's title clarifies it showcases weekly averages of the numerical column by category, with outliers emphasized.

```
def plot_violin(df, category, avg_numeric):
    plt.figure(figsize=(12, 6))
    sns.violinplot(x=category, y=avg_numeric, data=df, palette="muted",
inner=None)
# Set inner=None to remove the bars inside the violins
    sns.swarmplot(x=category, y=avg_numeric, data=df, color='k', alpha=0.6)
# Add a swarmplot to show individual points
    avg_numeric = avg_numeric.replace('_', ' ')
    plt.title(f'Violin Plot of Weekly {avg_numeric} by {category} with
Outliers')
    plt.show()

# Ensure the 'Avg_Anomaly_Scores' column is of type float for the violin
plot
weekly_severity_avg_df['Avg_Anomaly_Scores'] =
weekly_severity_avg_df['Avg_Anomaly_Scores'].astype(float)
# Call the plotting function for the aggregated weekly average data
plot_violin(weekly_severity_avg_df, 'Severity Level', 'Avg_Anomaly_Scores')
```

Figure 32. Code snippet for the implementation of the violin plot.

This technique effectively illustrates anomaly score distributions across different severity levels of cyber threats. The violin plot provides density estimation, while the swarm plot highlights individual data points, particularly outliers. Clusters of scores within each severity level are evident, with outliers standing apart. However, dense datasets might compromise readability, necessitating techniques like jittering or summary statistics. Additionally, the static nature of the plot limits its ability to capture temporal changes. It's best used alongside dynamic visualizations to reflect evolving data distributions over time. This visualization method offers valuable insights but requires careful consideration of dataset characteristics to ensure effective communication of information.

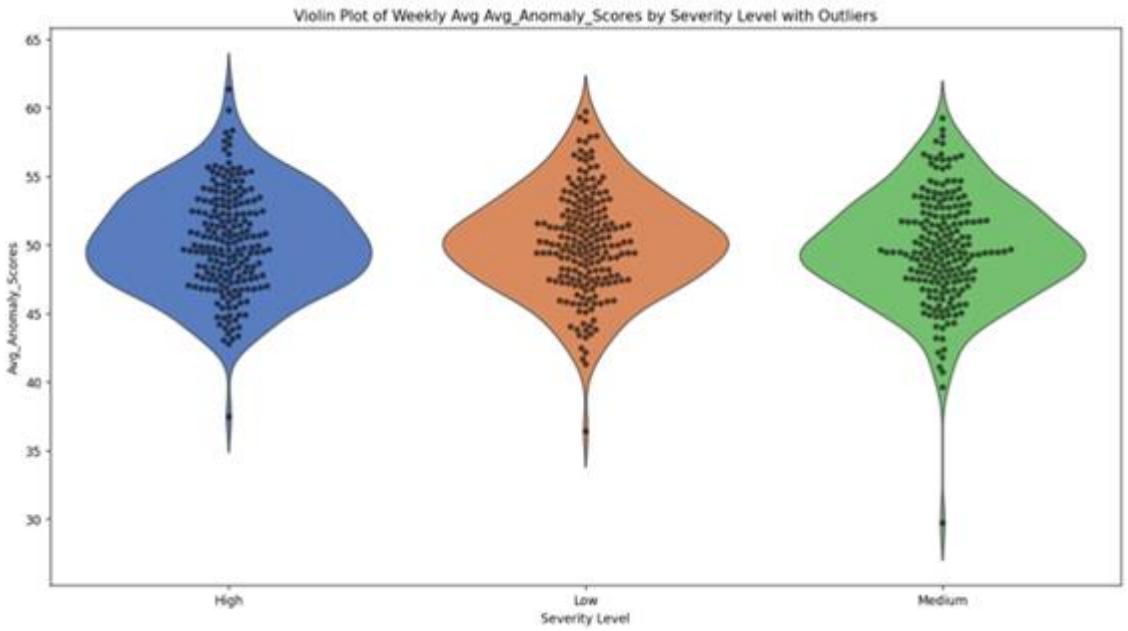

Figure 33. Violon plot illustrating anomaly score distributions across different severity levels of cyber threats.

15. Scatter Matrix Plot

The **'plot_scatter_matrix'** function generates a scatter matrix, or pair plot, showcasing pairwise comparisons of multiple numerical variables. In this code, it visualizes relationships and distributions within the **'weekly_avg_df'** DataFrame, excluding the 'Week' column. The function initializes a figure with specified size for clarity. Using 'pd.plotting.scatter_matrix', it generates the scatter matrix, with off-diagonal subplots representing scatter plots and diagonal subplots showing Kernel Density Estimates (KDE) for each variable against itself. Transparency of points is controlled with the alpha parameter. The scatter matrix's title is set to "Weekly Average Metrics Scatter Matrix" for clear identification.

```
# Plotting functions
def plot_scatter_matrix(df_subset, title):
    fig, ax = plt.subplots(figsize=(12, 12))
# Create a figure and a grid of subplots
    pd.plotting.scatter_matrix(df_subset, alpha=0.2, ax=ax, diagonal='kde')
    fig.suptitle(title, fontsize=16)
# Add a main title to the figure

plot_scatter_matrix(weekly_avg_df.drop('Week', axis=1), "Weekly Average
Metrics Scatter Matrix")
```

Figure 34. Code snippet for the implementation of the scatter matrix plot.

This visualization method offers a comprehensive view of relationships between weekly averaged cybersecurity metrics. Each subplot pairs two metrics, revealing potential patterns or correlations. Histograms along the diagonal provide univariate overviews, aiding in rapid assessment of each metric's distribution. While powerful for preliminary analysis, scatter matrices can suffer from dense matrices obscuring individual plots and scalability issues with large datasets. Careful selection of variables and techniques like transparency adjustments or point jittering can mitigate these challenges, ensuring effective exploration of data relationships.

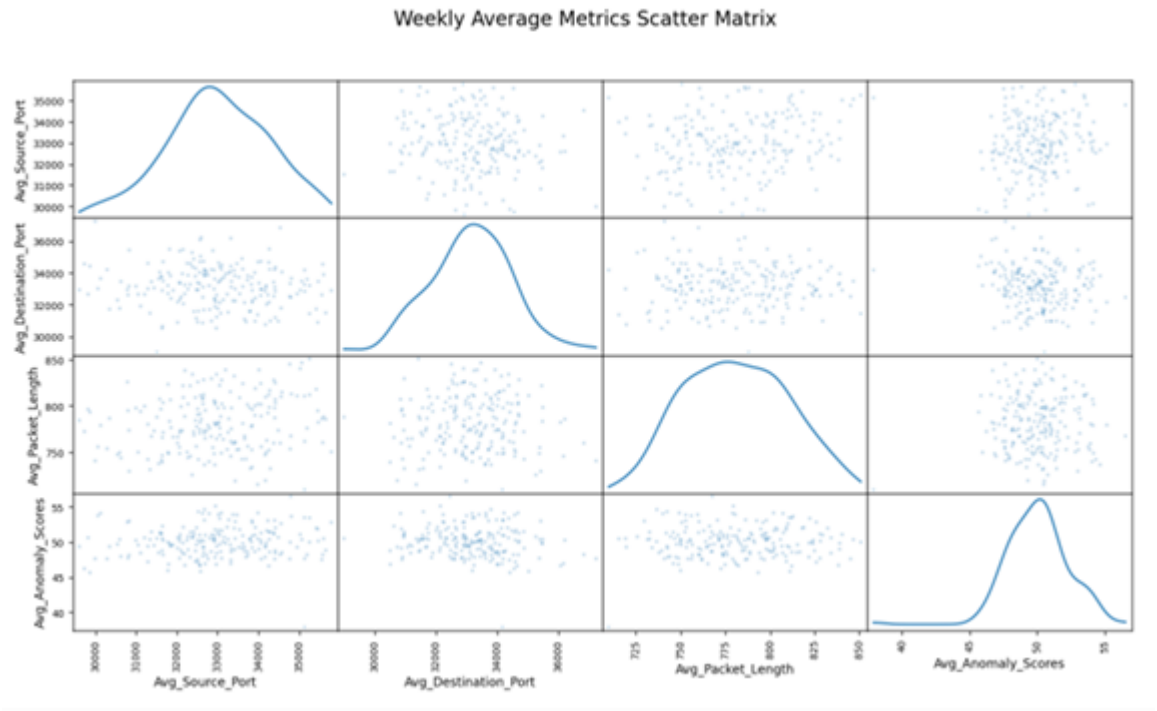

Figure 35. Scatter matrix plot illustrating the weekly average matrix of the different columns.

16. Heat Map

The heatmap visualization displays the distribution of alerts and warnings across varying severity levels within a dataset. Figure 35 shows the heatmaps respective code **' heatmap_data = pd.crosstab(data['Alerts/Warnings'], data['Severity Level'])'** is used to form the foundation of the heatmap, displaying integer values within each cell revealing the precise number of occurrences corresponding to each severity level, which can be seen on the x-axis. The alerts or warnings shown on the y-axis, with the colormap **'YlGnBu'** n the code **'ns.heatmap(heatmap_data, annot=True, fmt="d", cmap='YlGnBu') '** provides a visually gradient. The colour intensity directly correlates with the frequency of occurrences, with darker hues indicating higher numbers. A scale alongside the visualisation reveals lower to higher densities of events, this allows readers to interpret the visualisation clearly. The axes are labelled 'Severity Level' horizontally and 'Alerts/Warnings' vertically using the code '**plt.xlabel('Severity Level') plt.ylabel('Alerts/Warnings')',** creating more clarity and less information congestion.

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

# Load the data from Excel
data = pd.read_excel(" /Applications/Quant /Data processed.xlsx")

# Remove rows with 'Unknown' in 'Alerts/Warnings' or 'Severity Level'
data = data[(data['Alerts/Warnings'] != 'Unknown') & (data['Severity Level'] != 'Unknown')]

# Create a crosstab for the heatmap
heatmap_data = pd.crosstab(data['Alerts/Warnings'], data['Severity Level'])

# Heatmap showing the relationship between Alerts/Warnings and Severity Level
plt.figure(figsize=(12, 6))
sns.heatmap(heatmap_data, annot=True, fmt="d", cmap='YlGnBu')  # 'annot=True' to display values, 'fmt="d"' for
integer format
plt.title('Heatmap of Alerts/Warnings by Severity Level')
plt.xlabel('Severity Level')
plt.ylabel('Alerts/Warnings')

# Proper layout to avoid overlapping
plt.tight_layout()

plt.show()
```

Figure 36. Code snippet for the implementation of the heat map.

The heat map in Figure 37 reveals the medium severity level is marginally the most prevalent at 3708 instances, surpassing high severity at 3695 and low severity at 3616. This suggests a balanced operational monitoring across all spectrums of severity. The heatmap guides the cyber security expert to identify the most common threat levels and adjusting cybersecurity measures accordingly. Even though it is highly advantageous, the heatmap does have limitations. It doesn't show the specifics of individual alerts or warnings, which means additional visualisations would need to be carried to allow for a deeper investigation.

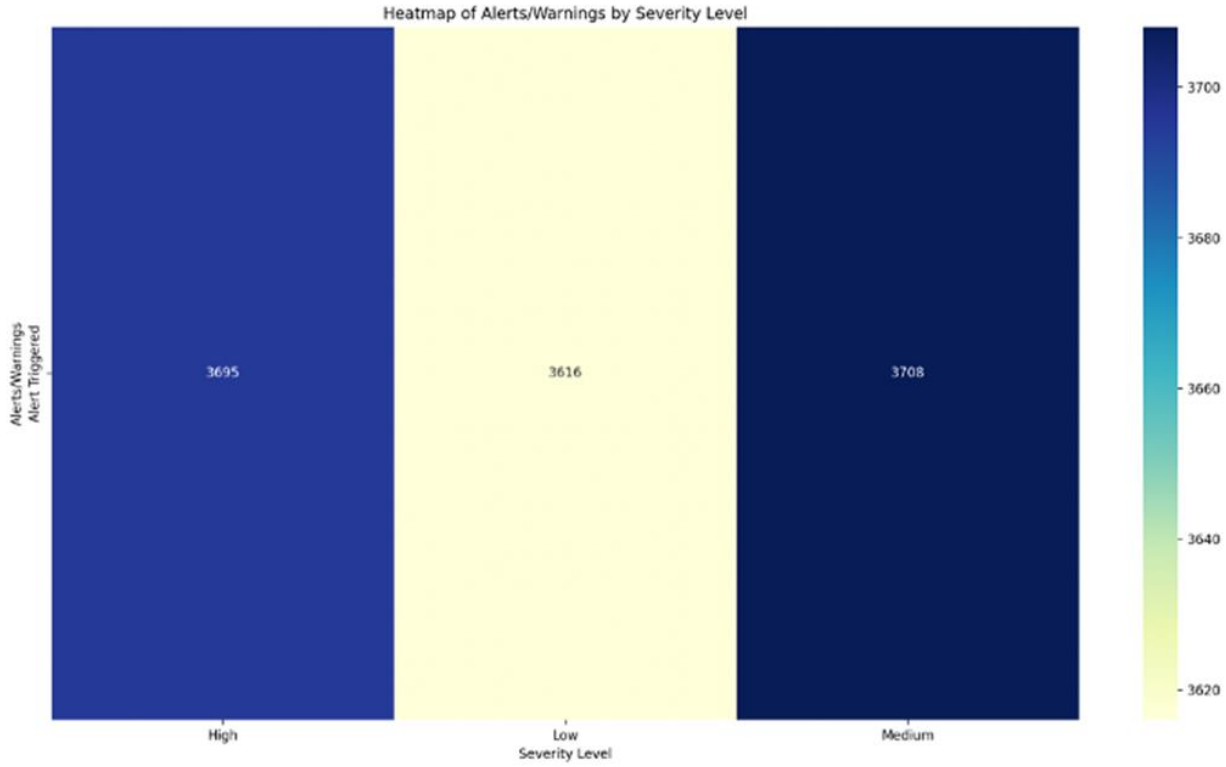

Figure 37. Heat Map illustrating the distribution of alerts and warnings across different severity levels.

17. Stacked Bar Chart

The stacked bar chart shows the actions taken in response to cyberattacks over a series of weeks. Demonstrated through the code in Figure 38, Pivot tables are used '**pivot_table.plot(kind='bar', stacked=True, figsize=(12, 6))**' to restructure the data. With the week_column set as the index, allowing the analysis to be temporal. The sub_category values, representative of the various actions taken in response to cyberattacks, the code ' **values='Count', aggfunc='sum', fill_value=0)**' transforms them into columns with their counts accumulated through the aggfunc='sum' operation. Any potential voids in the data are addressed by the **'fill_value=0'**. Visualisation aspects are determined using the .plot() method, with the readers visual readability being prioritised to allow for a clear analysis of the visualisation. As a result, a 45-degree rotation of the x-tick labels and the use of **plt.tight_layout()** reflect a conscientious effort to avoid text congestion and allows the visualisation to present neatly.

```
def plot_stacked_bar(df, week_column, category, sub_category):
    pivot_table = df.pivot_table(index=week_column, columns=sub_category,
values='Count', aggfunc='sum', fill_value=0)
    pivot_table.plot(kind='bar', stacked=True, figsize=(12, 6))
    plt.title(f'Stacked Bar Chart of {sub_category} by {category} over
Weeks')
    plt.ylabel('Count')
    plt.xlabel('Week')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

plot_stacked_bar(weekly_counts_df, 'Week', 'Attack Type', 'Action Taken')
```

Figure 38. Code snippet for the implementation of the stacked bar chart actions taken in response to cyberattacks over a series of weeks.

On observation of the chart in Figure 39, dynamic shifts in the types of actions taken are common and could be due to changes in the cyberattack strategy used or reveal the effectiveness of response protocols within the network. The chart also shows significant changes in patterns by making it easy to highlight differences in action. This shed light on the flow of cybersecurity measures, allowing cybersecurity analysts to understand the volume and type of reactions (whether an attack was blocked, ignored, or merely logged), this revelation offers a narrative on the effectiveness of attack responses and allows for the detection of trends. However, the stacked bar chart does have limitations in terms of readability, individual weeks can become challenging read due the density of the data present, this can affect the clarity an analyst may have on the individual data points. In an effort to assist with clarity, a descriptive key allows viewers to navigate the visualisation more clearly, enhancing the readers examination.

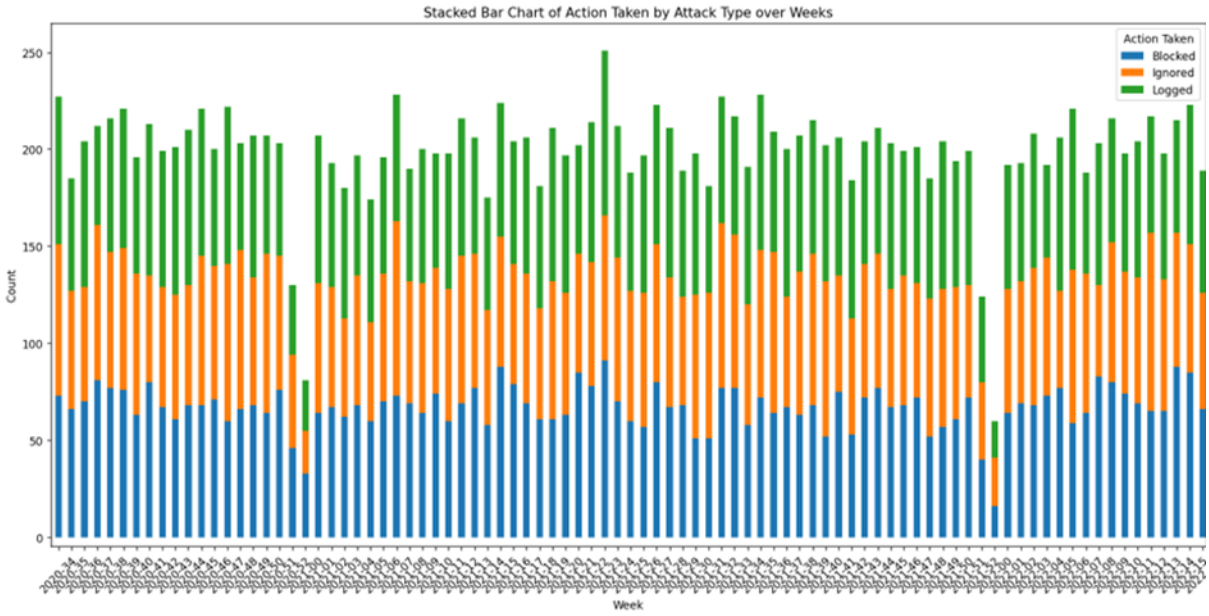

Figure 39. Stacked bar chart illustrating the actions taken in response to cyberattacks over a series of weeks.

18. Area Plot

The Area plot shows the average anomaly scores over time. The code in Figure 40 **'df[week_column] = pd.to_datetime(df[week_column] + '-0', format='%Y-%W-%w')'** first converts the week into a datetime format **%Y-.%W-%w**. This step is vital to preform accurate time-series visualization within pandas and allows the data to be placed in chronological order of data. This step is followed by visual customisations including the **'plt.tight_layout()'** function, to organise the layout of visualisation. The titles are placed, with the rotation of the x-ticks **'plt.xticks(rotation=45)'** to allow for a clean layout of the title along the x-axis. Overall, the code is structured to allow for the clear presentation of the area plot.

```
def plot_area_chart(df, week_column, avg_column):
    df[week_column] = pd.to_datetime(df[week_column] + '-0', format='%Y-%W-
%w')
    df.set_index(week_column, inplace=True)
    df[avg_column].plot.area(alpha=0.4)
    avg_column = avg_column.replace('_', ' ')
    plt.title(f'Area Chart of {avg_column} over Time')
    plt.ylabel(avg_column)
    plt.xlabel('Date')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

plot_area_chart(weekly_avg_df[['Week', 'Avg_Anomaly_Scores']], 'Week',
'Avg_Anomaly_Scores')
```

Figure 40. Code snippet for the implementation of the Area Plot showing the weekly behaviour of anomaly scores.

Figure 41 shows many peaks and drops in the area plot, with a specific anomaly appearing before 01-23. These ups and downs could be due to potential surges in attack attempts or a change in the detection methods being used. A pronounced peak, for example, might correspond to a specific attack campaign or a breach, while a sudden dip could indicate system downtime or a lapse in data collection. The chart areas beneath the plotted lines are filled allowing for a clearer interpretation of the intensity of cybersecurity issues present, this method allows for the easy identification of trends. Limitations of this visualisation can be found. For example, the averaging of the data across time can affect how specific details may be, like small spikes in anomaly scores that may be important for cybersecurity incident response. Another limitation can be found in the static nature of the chart, it does not communicate the precise timing or events that contribute to the fluctuations. Lastly, the area chart's simplicity could be misleading without proper context. Outliers are smoothed over in the weekly average, which could potentially lead to a downplaying of severe anomalies. More analytical techniques and visualisations would need to be carried out to flag significant deviations or

to correlate with known security incidents.

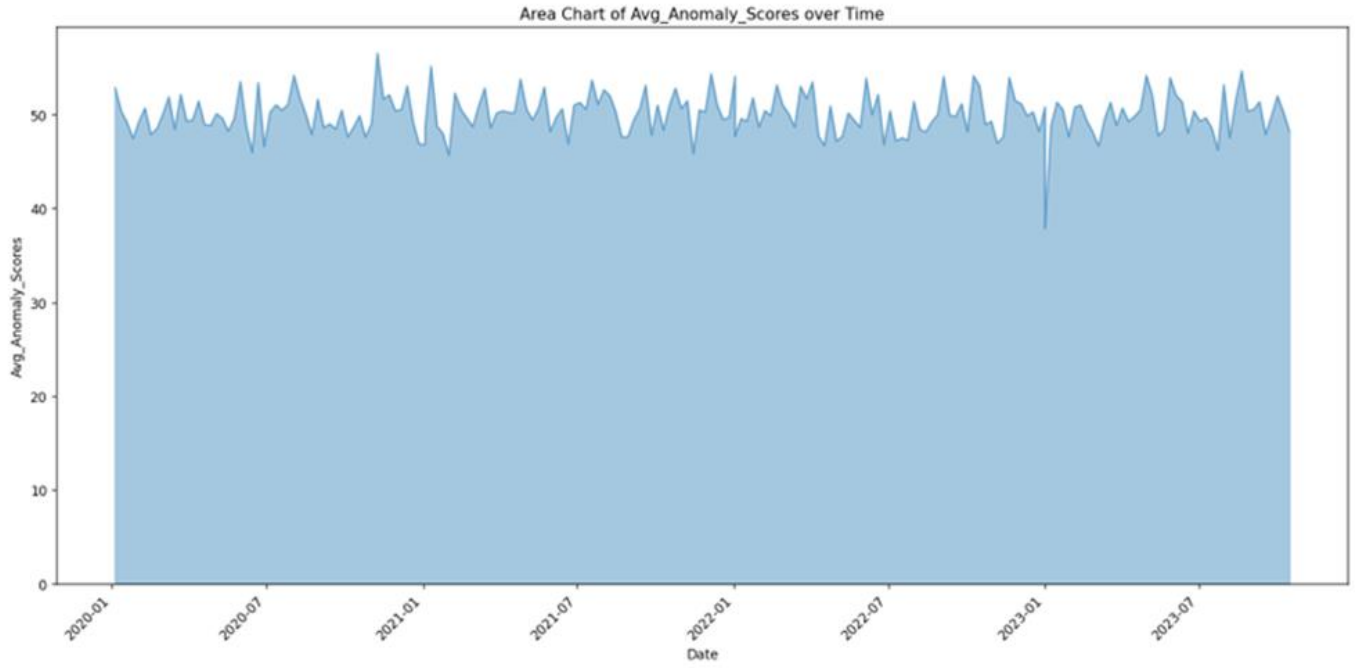

Figure 41. Area Plot showing the weekly behaviour of anomaly scores.

Big Data Techniques and Analysis

Big data technologies have revolutionized the way data is processed, analysed, and how valuable insights are extracted from massive and complex datasets. These technologies offer several benefits that enable organizations to exploit the power of data and drive informed decision-making. There are several advantages of big data technologies. An example is their ability to handle and process vast amounts of structured, unstructured, and semi-structured data from various sources, such as social media, sensor networks, financial transactions, and online interactions [6].

Another significant benefit is their scalability and distributed computing capabilities. Traditional data processing and analysis methods often struggle with large-scale datasets, leading to performance bottlenecks and limitations [7][8].

Big data technologies, such as Apache Hadoop and Apache Spark, leverage distributed computing frameworks that can scale horizontally across multiple nodes, allowing for parallel processing and efficient handling of massive datasets [6].

Furthermore, big data technologies offer advanced analytical capabilities, including machine learning, deep learning, and predictive modelling techniques. These techniques enable organizations to uncover hidden patterns, correlations, and trends within their data, leading to valuable insights and data-driven decision-making processes. For example, predictive modelling can be used in fraud detection, customer churn analysis, and supply chain optimization, while machine learning algorithms can be applied to image and speech recognition, natural language processing, and recommendation systems [9][10].

One of the latest techniques in big data analysis is the use of deep learning algorithms and neural networks. These techniques have shown remarkable success in areas such as computer vision, natural language processing, and speech recognition. Deep learning models can automatically learn and extract complex patterns and features from vast amounts of data, enabling more accurate predictions and decision-making processes [9].

Another emerging technique is graph analytics, which focuses on analysing and extracting insights from interconnected data structures, such as social networks, transportation networks, and financial transactions. Graph analytics can uncover relationships, patterns, and dependencies within complex datasets, leading to valuable insights in areas such as fraud detection, risk management, and recommendation systems [10] [6].

To implement big data technologies in this project, Apache Spark, a powerful open-source distributed computing framework designed for large-scale data processing and analysis was used. Known for its Python-based interface – PySpark, Apache Spark was installed, and its various components explored, including Spark Core, and Spark MLlib (Machine Learning library). Using the same dataset from the previous visualizations, data was loaded into amazon S3 bucket and then into a Spark DataFrame. Exploratory data analysis, data cleaning, feature engineering tasks and machine learning models were performed. PySpark's Resilient Distributed Datasets (RDDs) and DataFrame APIs allowed for the performance of these operations in a distributed and fault-tolerant manner, leveraging the computing resources of multiple nodes if available [13] [15].

RDDs are collections of data distributed across a cluster, designed to be fault-tolerant and to support parallel operations. They are inherently resilient, allowing them to recover from failures without data loss—a critical feature when processing massive datasets. With PySpark, users can perform complex transformations like **`map()`** (applying a function to each element), **`filter()`** (selecting elements that meet a condition), and **`groupByKey()`** (grouping elements by a key), with lazy evaluation [14]. This means these transformations aren't executed immediately but rather when an action, such as **`collect()`** (gathering results to the driver program), is triggered. Lazy evaluation adds efficiency and flexibility to big data workflows [10].

For example, in this project analysing a 40,000-row dataset with various attack types, PySpark's capabilities were tested. Using the **`groupByKey()`** transformation, the data was categorised by "Attack Type", followed by applying further actions to count the occurrences in each category. This demonstrated how PySpark could effectively process large datasets, delivering detailed insights.

One of the challenges experienced during this project was understanding Spark's programming model and adapting the existing data processing and analysis techniques to the distributed computing paradigm. However, Spark's well-documented API and extensive online resources helped in overcoming this challenge and helped with understanding the framework.

Using Spark's MLlib library, various machine learning models, such as anomaly detection and predictive modelling using random forest classifiers were used for predicting attack types based on the given features. Spark's distributed computing capabilities allowed training these models on the entire dataset efficiently, without the need for sampling or data partitioning.

Throughout this experience, Apache Spark's ability to handle large datasets seamlessly, its comprehensive set of libraries and APIs for data processing, analysis, and machine learning tasks, and its scalability and fault-tolerance capabilities were explored and appreciated. However, it’s worth noting the need for a strong understanding of distributed computing concepts and the potential challenges associated with data partitioning, data skew, and resource management in a distributed environment.

To conclude, big data technologies like Apache Spark play a crucial role in enabling organisations to explore the full potential of large datasets. By providing scalable, efficient, and flexible tools, these technologies open the door to deeper insights and more informed decision-making across a range of industries, from cybersecurity to healthcare to product development and many more.

Contribution of members

In this project, our team employed a collaborative strategy, utilizing our varied skills and expertise to thoroughly analyse and visualize data from the chosen dataset. Each team member played a crucial role in understanding the dataset, conducting data preprocessing, creating visualizations, and engaging in critical analysis.

We selected an appropriate dataset and each member focused on creating individual visualizations, coordinating closely to avoid duplication of efforts. This approach allowed us to employ a variety of visualization and preprocessing techniques, enriching our report, and enhancing our collective knowledge, thereby improving our skills.

Angel initiated the report, setting the framework for the subsequent sections. The main body of the report consisted of data preprocessing and visualization, with contributions from each team member: Daniel produced three visualizations, Millicent created eight, Gabriel developed four, and Angel added two. Additionally, Gabriel investigated contemporary big data analysis techniques while Millicent used one of the investigated approaches – Apache Spark to create sample visualisations and develop a predictive model that predicts possible cyber-attacks with an accuracy of 53% (see appendix). A few challenges were encountered some of which could not be fixed due to time constraints.

Millicent and Angel were responsible for the final assembly of the report, ensuring it was coherent, aligned with the assessment guidelines, and effectively communicated our findings. The entire team participated in the proofreading and editing process, contributing feedback and insights to ensure a streamlined and effective workflow.

References

[1]

J. S. Raj, K. Kamel, and P. Lafata, *Innovative Data Communication Technologies and Application*. Springer Nature, 2022.

[2]

Aretov Inc, “Importance of Feature Selection in Machine Learning | Aretove,” *Aretove Technologies*, Dec. 23, 2020. https://www.aretove.com/importance-of-feature-selection-in-machine-learning

[3]

A. Bhuyan, “Word Clouds: A Visual Representation of Language - DZone,” *dzone.com*, Dec. 11, 2023. https://dzone.com/articles/word-clouds-a-visual-representation-of-language (accessed Apr. 25, 2024).

[4]

J. Dougherty and Ilya Ilyankou, *Hands-On Data Visualization*. “O’Reilly Media, Inc.,” 2021.

[5]

IBM, “Data Visualization,” *Ibm.com*, 2019. https://www.ibm.com/topics/data-visualization

[6]

A. Katal, M. Wazid, and R. H. Goudar, “Big data: Issues, challenges, Tools and Good Practices,” *2013 Sixth International Conference on Contemporary Computing (IC3)*, Aug. 2013, doi: https://doi.org/10.1109/ic3.2013.6612229.

[7]

M. M. Alani, “Big data in cybersecurity: a survey of applications and future trends,” *Journal of Reliable Intelligent Environments*, vol. 7, no. 1, Jan. 2021, doi: https://doi.org/10.1007/s40860-020-00120-3.

[8]

I. Kotenko, Igor Saenko, and A. Branitskiy, “Machine Learning and Big Data Processing for Cybersecurity Data Analysis,” *Data Science in Cybersecurity and Cyberthreat Intelligence*, vol. 177, no. 1, pp. 61–85, Jan. 2020, doi: https://doi.org/10.1007/978-3-030-38788-4_4.

[9]

A. Oussous, F.-Z. Benjelloun, A. Ait Lahcen, and S. Belfkih, “Big Data technologies: A survey,” *Journal of King Saud University - Computer and Information Sciences*, vol. 30, no. 4, pp. 431–448, Oct. 2018, doi: https://doi.org/10.1016/j.jksuci.2017.06.001.

[10]

M. Chen, S. Mao, Y. Zhang, and V. C. M. Leung, *Big Data*. Cham: Springer International Publishing, 2014. doi: https://doi.org/10.1007/978-3-319-06245-7.

[11]

AmpCode, “Building Machine Learning Model using Apache Spark | PySpark MLlib Tutorial,” *www.youtube.com*, Jun. 05, 2023. https://www.youtube.com/watch?v=HK4YW9qvQE8&t=745s&ab_channel=AmpCodehttps://www.youtube.com/watch?v=HK4YW9qvQE8&t=745s&ab_channel=AmpCode (accessed Apr. 28, 2024).

[12]

D. Parmar, “IPL Data Analysis | Apache Spark End-To-End Data Engineering Project,” *www.youtube.com*, Apr. 21, 2021. https://www.youtube.com/watch?v=0iNJPKheQqM&ab_channel=DarshilParmar (accessed Apr. 28, 2024).

[13]

Databricks, “Tutorial: Load and transform data using Apache Spark DataFrames,” *docs.databricks.com*, Apr. 25, 2024. https://docs.databricks.com/en/getting-started/dataframes.html (accessed Apr. 26, 2024).

[14]

Apache Spark, “RDD Programming Guide - Spark 3.0.0 Documentation,” *spark.apache.org*. https://spark.apache.org/docs/latest/rdd-programming-guide.html

[15]

L. Yang, “Distributed Deep Learning Made Easy with Spark 3.4,” *NVIDIA Technical Blog*, Jun. 12, 2023. https://developer.nvidia.com/blog/distributed-deep-learning-made-easy-with-spark-3-4/ (accessed Apr. 28, 2024).

[16]

A. Kirk, *Data visualisation : a handbook for data driven design*. Los Angeles: Sage, 2019.

Appendix

Link for code and dataset on GitHub:
https://github.com/AngelAriyibi/CyberSecurityAnalysis/blob/main/Data%20Visualization.py

Link for application of Big Data Technique: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4062922020393116/2782086901302475/4058282943020947/latest.html