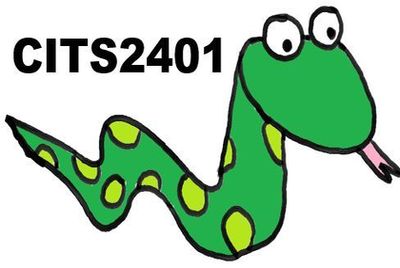
Project 2 2023

CITS 2401 Computer Analysis and Visualisation



University of Western Australia

Professor: Jin Hong

Student: Edward Le

Student ID: 2302568

Due date: 19/05/2023

Motivation:

The primary motivation for this data analysis is to evaluate the performance of the intrusion detection system (IDS) by analyzing the given smart car sensor readings and network traffic datasets. We want to explore and understand the datasets, extract meaningful insights, and recommend improvements to enhance the IDS's capabilities. After that, we will also employ data visualization techniques using Matplotlib to enhance our understanding and effectively communicate our findings.

Methodology:

We used the NumPy library to perform various data analysis tasks on the given datasets. We implemented functions in the Task1 class to extract descriptive statistics, count unique values, select specific rows and columns, find the difference between two columns, stack data, and conditionally select values based on class. We used these functions to gain insights into the datasets and identify patterns and anomalies.

For example, we first check which file number is being used to analysis using method check\_file\_number(), the first file is used by default. Similarly, we can analysis more datasets instead of 2 using the same method.

Methods such as overall() and get\_descriptive\_statistics() are for first sight look at the data, through that we know numbers of lines, which columns are numeric, and some statistics like count, min, std, etc.

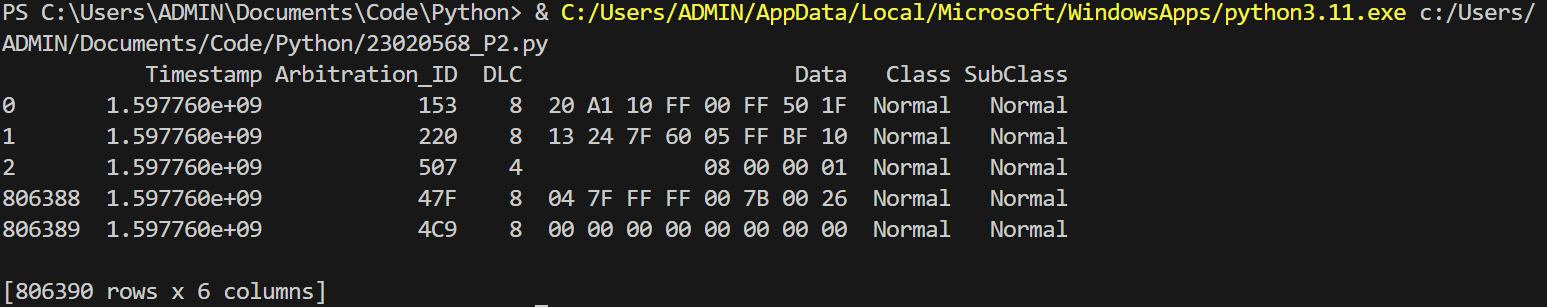


Figure 1: overall()

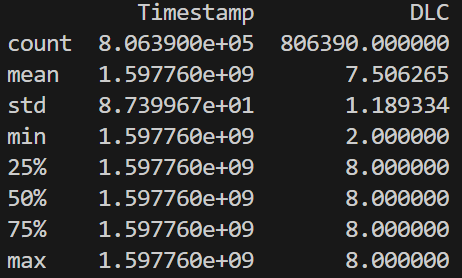


Figure 2: get\_descriptive\_statistics()

Then method get\_unique\_values\_count() can be used with "Arbitration\_ID” to remove duplicate ID and get each ID’s count. Or Numpy where function can be useful to select values that are classified as “Attack”.

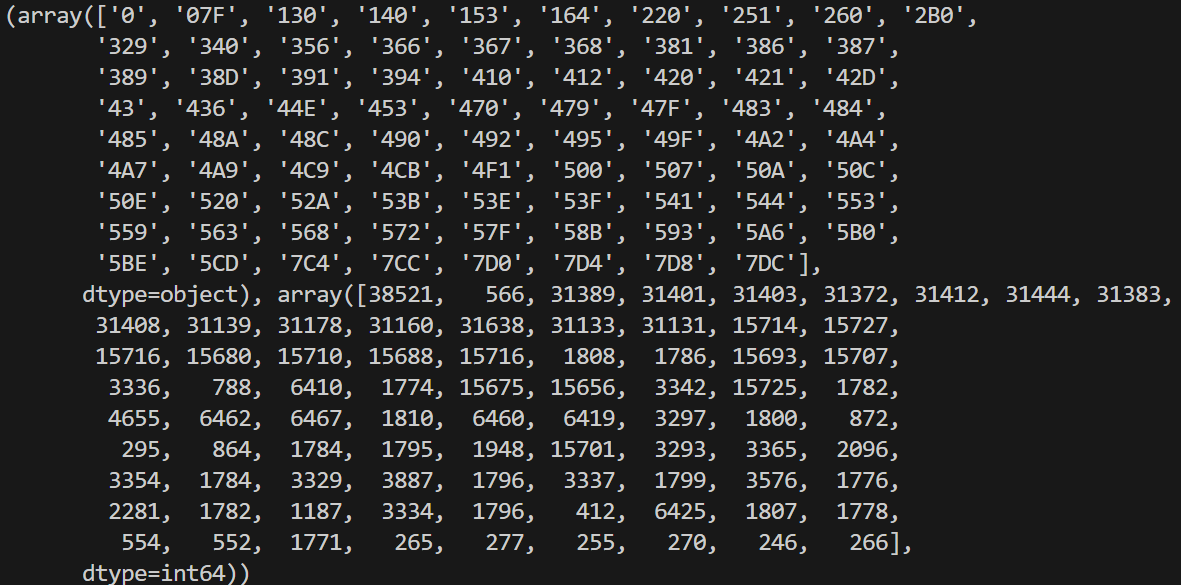


Figure 3: get\_unique\_values\_count("Arbitration\_ID")

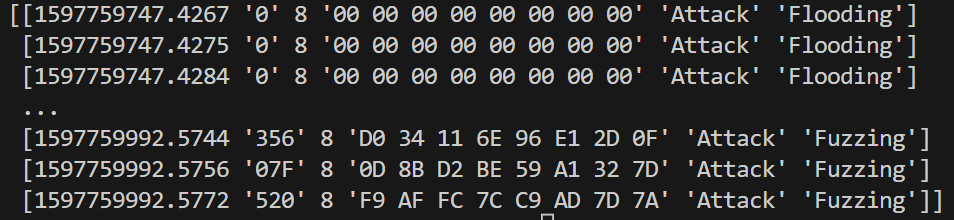


Figure 4: condition("Class", "Attack")

And stack\_data() will merge values according to same column between 2 files, we can use it to create bigger data file and maybe write into new csv file.

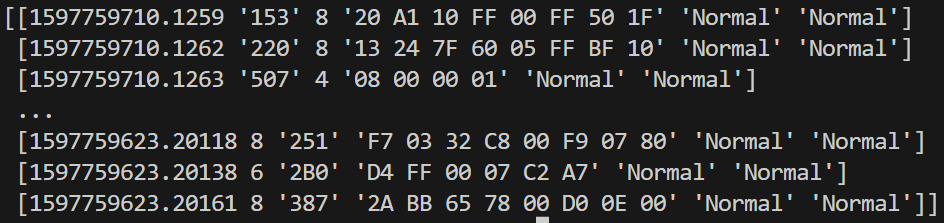


Figure 5: stack\_data()



Figure 6: len(stack\_data())

We have also created the Task2 class, which utilizes the Matplotlib library to visualize the data. The Task2 class includes functions for plotting line charts, line plots, bar charts, grouped bar charts, pie charts, and histograms. These visualization methods help us explore the datasets from different angles and provide a visual representation of the data.

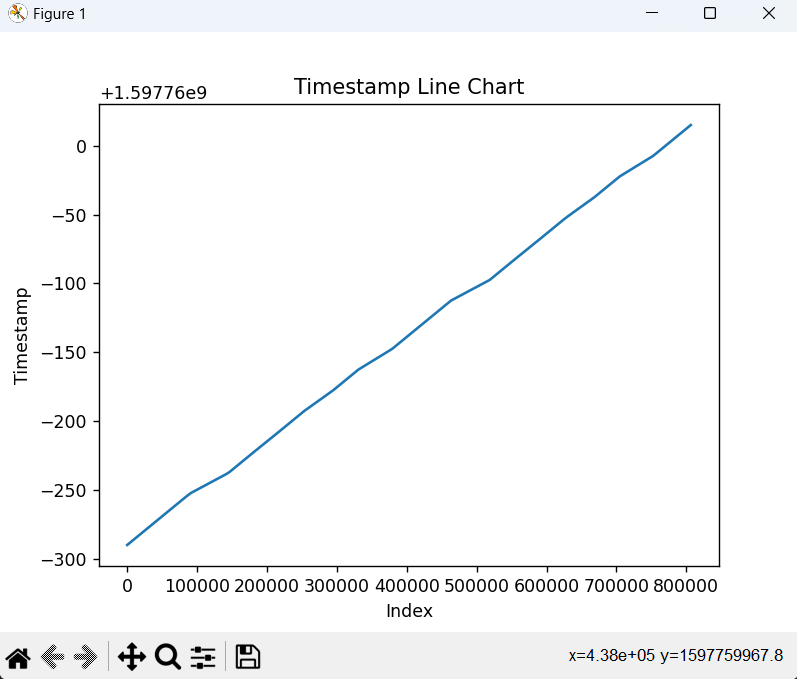


Figure 7: plot\_line\_chart()

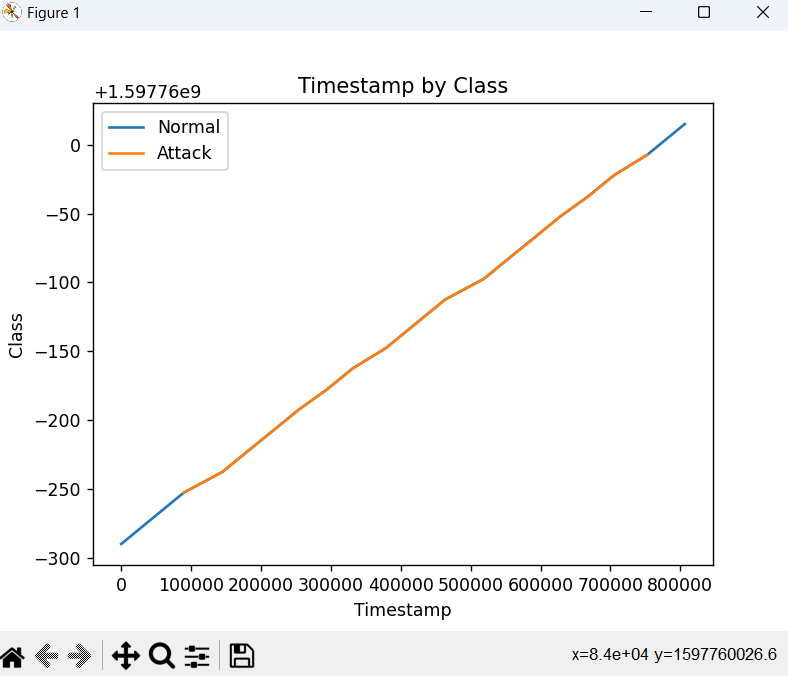


Figure 8: plot\_line\_plot()

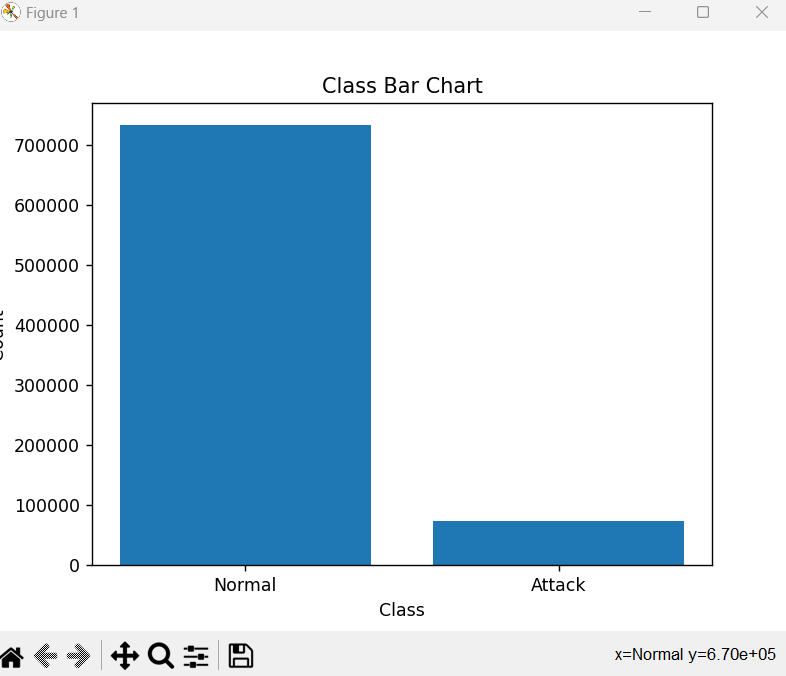


Figure 9: plot\_bar\_chart()

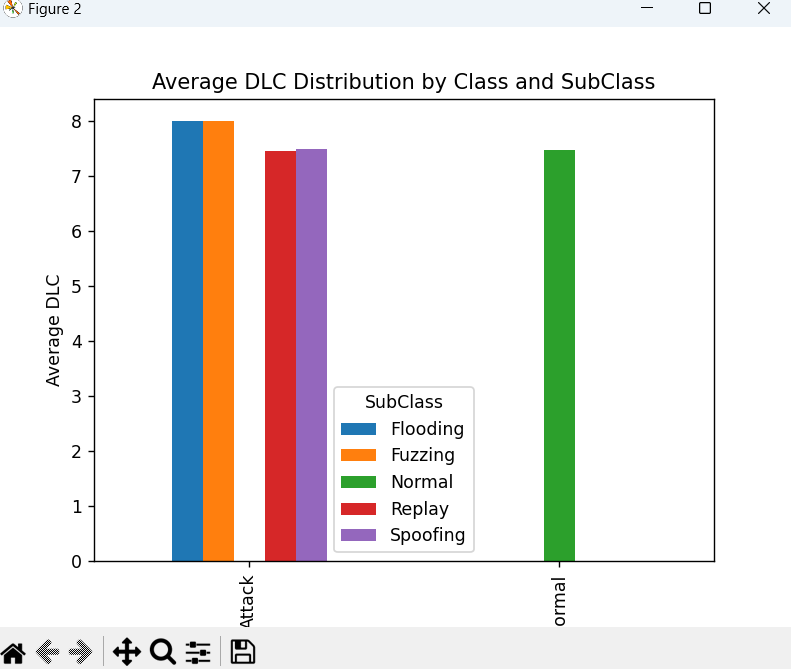


Figure 10: plot\_grouped\_bar\_chart()

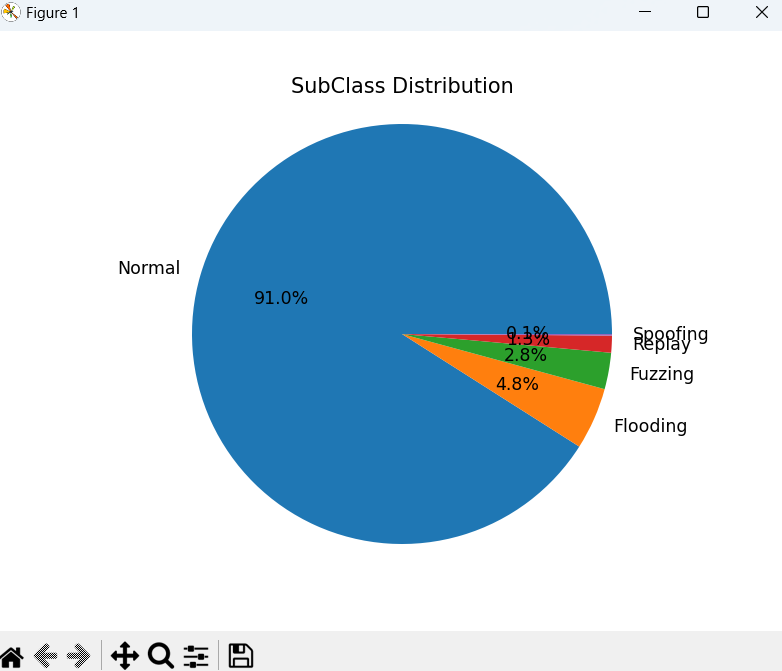


Figure 11: plot\_pie\_chart()

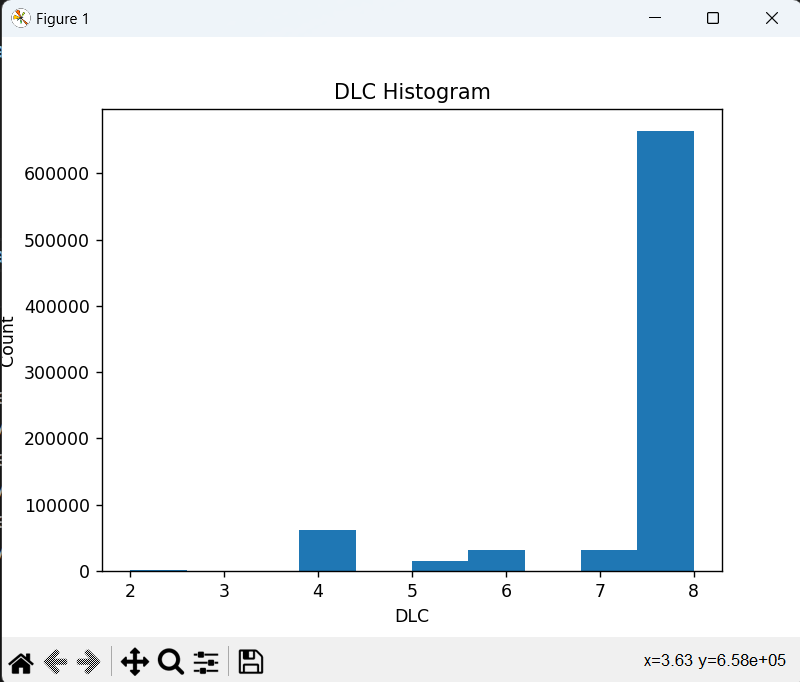


Figure 12: plot\_histogram()

Results:

By performing descriptive statistics on the datasets, we found that the mean and standard deviation of the data columns in the smart car sensor readings dataset are generally consistent across the normal and attack events. However, we observed that the maximum and minimum values of some columns are significantly higher in the attack events than in the normal events. In the network traffic dataset, we found that the length column's mean and standard deviation are higher in the attack events than in the normal events.

We used the different function to find the time difference between corresponding timestamps in the two datasets. We found that the time difference between the two datasets is consistently within a few milliseconds, indicating that the events in the two datasets occur almost simultaneously.

We also stacked the two datasets using the stack\_data function to create a combined dataset. We found that the combined dataset has a total of 1605682 events, with the majority of events labeled as normal.

Line Chart: We plotted a line chart of the timestamps in the dataset using the plot\_line\_chart() function. A line chart is a good choice for visualizing how a single continuous variable changes over time. In this case, we observe that timestamp do not vary much, with value of just around 300. It makes the action of detecting intrusions through time difference becomes less accurate.

Line Plot: We used the plot\_line\_plot() function to create a line plot of timestamps categorized by class. The line plot distinguishes between normal and attack events by plotting the timestamps for each class separately. This visualization helps us compare the timestamp distributions between normal and attack events. Through this, we observe that timestamp of attacks are in range of the middle, lower and higher timestamp value than that are always not attack.

Bar Chart: We employed the plot\_bar\_chart() function to generate a bar chart that displays the count of events for each class. The bar chart provides a clear visual representation of the distribution of attack and normal instances in the dataset. This is an important first step in understanding the dataset and determining if it is balanced or imbalanced, which shows that normal events occur much more than attack ones, with 9 times higher. However, attack events occur around 100000 times, it means the security system is currently not working well, as there are too many intrusions happen.

Grouped Bar Chart: We analyze the distribution of DLC values within each combination of Class and SubClass labels. By creating a grouped bar chart, we can compare the DLC distributions across different Class and SubClass combinations, identifying any variations or similarities in the DLC values within each group.

Pie Chart: We utilized the plot\_pie\_chart() function to plot a pie chart representing the distribution of subclasses in the dataset. The pie chart displays the relative proportions of each subclass, making it easier to identify the subclasses with higher or lower representation. Via this chart, we observe that 91% of time the system is not attacked. Moreover, among the attacks, more than half of them make the system in ‘flooding’ status, which is the highest level of attacks, this proves that if the system is attacked, it is likely to have serious damages.

Histogram: By using the plot\_histogram() function, we generated a histogram of the DLC values in the dataset. The histogram provides a visual representation of the distribution of DLC values and helps us identify any notable patterns or outliers, which shows that most DLC is 8.

Recommendations:

Based on our analysis, we recommend that the IDS should focus on detecting events with unusual maximum and minimum values in the smart car sensor readings dataset and higher length values in the network traffic dataset. Additionally, we recommend that the IDS should pay particular attention to other factors that are more likely to be associated with attack events as our IDS just focused on timestamp, and factor such as DLC is not considered using effectively. Furthermore, address the class imbalance issue, as the dataset contains a larger proportion of normal events compared to attack events. Finally, the system should be improved to be protected from order of attack seriousness, with level of seriousness decreasing: flooding, fuzzing, spoofing, replay. This can help improve the IDS's ability to detect attacks effectively.

Capabilities of the IDS:

The data analysis and visualizations provided insights into the IDS's capabilities and limitations. The IDS demonstrates the ability to effectively detect normal events based on the majority class. However, it may have limitations in detecting attacks that have similar statistical characteristics to normal events. The recommendations provided can enhance the IDS's performance by addressing these limitations and focusing on the identified areas of improvement.

Appendix:

Appendix A: Task 1 - Data Analysis using NumPy

The Task1 class in the provided source code contains functions for data analysis using the NumPy library. The following methods are available:

1. check\_file\_number(): This method checks which file of the dataset is being used and returns the corresponding data.
2. overall(): Returns an overall look at the data.
3. get\_descriptive\_statistics(): Provides descriptive statistics of the data, including count, mean, standard deviation, and more.
4. get\_unique\_values\_count(column\_name): Returns unique values in a specified column and their respective counts.
5. select\_specific\_rows\_and\_columns(rows, columns): Selects specific values from rows and columns in the data.
6. difference(column\_name): Calculates the difference between values in two columns.
7. stack\_data(): Stacks data from two files vertically.
8. condition(column\_name, compare\_value): Returns a new 2D array with values that match a specified condition.

Appendix B: Task 2 - Data Visualization using Matplotlib

The Task2 class in the provided source code focuses on data visualization using the Matplotlib library. The following methods are available:

1. check\_file\_number(): Checks which file of the dataset is being used and returns the corresponding data.
2. plot\_line\_chart(): Plots a line chart using the timestamps in the data.
3. plot\_line\_plot(): Creates a line plot of timestamps categorized by class (normal and attack events).
4. plot\_bar\_chart(): Generates a bar chart displaying the count of events for each class.
5. plot\_grouped\_bar\_chart(): Plots a grouped bar chart with the distribution of DLC values within each combination of class and subclass labels.
6. plot\_pie\_chart(): Creates a pie chart representing the distribution of subclasses in the dataset.
7. plot\_histogram(): Generates a histogram of the DLC values in the dataset.

Appendix C: Example Test Cases

C1: Test cases for class Task1:

task1 = Task1("proj1\_data1.csv", "proj1\_data2.csv")

1. print(task1.overall())
2. print(task1.get\_descriptive\_statistics())
3. print(task1.get\_unique\_values\_count("Arbitration\_ID"))
4. print(task1.select\_specific\_rows\_and\_columns(1, 3))
5. print(task1.difference("Timestamp"))
6. print(task1.condition("Class", "Attack"))
7. new\_data = task1.stack\_data()
8. print(new\_data)
9. print(len(new\_data))

C2: Test cases for class Task2:

task2 = Task2("proj1\_data1.csv", "proj1\_data2.csv")

1. task2.plot\_line\_chart()
2. task2.plot\_line\_plot()
3. task2.plot\_bar\_chart()
4. task2.plot\_grouped\_bar\_chart()
5. task2.plot\_pie\_chart()
6. task2.plot\_histogram()

References:

Mode. (n.d). *Pandas .values\_count() & .plot().*

<https://mode.com/python-tutorial/counting-and-plotting-in-python/>

Mulani, S. (2020). *Python iloc() function – All you need to know!* AskPython.

<https://www.askpython.com/python/built-in-methods/python-iloc-function>

Numpy. (n.d). *numpy.unique.*

<https://numpy.org/doc/stable/reference/generated/numpy.unique.html>

Numpy. (n.d). *numpy.vstack.*

<https://numpy.org/doc/stable/reference/generated/numpy.vstack.html>

Riyaaggarwaldtu. (n.d). *Bin Size in Matplotlib Histogram.* GeeksforGeeks.

<https://www.geeksforgeeks.org/bin-size-in-matplotlib-histogram/>

Stackoverflow. (2011). *How do I use matplotlib autopct?*

<https://stackoverflow.com/questions/6170246/how-do-i-use-matplotlib-autopct>

SparkBy{Example}. (2023). *How to use Pandas unstack() Function.*

<https://sparkbyexamples.com/pandas/how-to-use-pandas-unstack-function/#:~:text=unstack()%20is%20used%20to,technique%20for%20reshaping%20the%20DataFrame>.

Zach. (2021). *Pandas: How to Use Groupby and Plot (With Examples).* Statology.

<https://www.statology.org/pandas-groupby-plot/>