Data Visualisation Report

Introduction

A dataset about international terrorism incidents is examined in this research. Variables like geography, target type, number of casualties, number dead, and number wounded are all included in the dataset. Finding significant patterns, trends, and insights through data visualisation approaches is the main goal. R and Python were the two tools used for the analysis. Comparing these tools' usability and visualisation quality is the aim.

Tool Comparison

The tools that I used in this homework are R and Python to analyse the data and perform data exploration analysis and data visualisation on the data

R programming:

Primarily ggplot2, dplyr, and corrplot. Visuals: bar charts, line charts, histograms, boxplots, correlation heatmap, pie chart, scatter plots.

Python Programming:

matplotlib, seaborn, pandas. Visuals: similar set (bar chart, histogram, boxplot, line chart, heatmap, pie, scatter).

Usability

ggplot2 in R

Advantages: The grammar-of-graphics paradigm produces layered plot code that is clear and consistent. With comparatively less code, it is simpler to create expressive, publishable graphs. enables clean data conversions through natural integration with Dplyr pipelines.

Cons: Some adjustments involve learning theme() characteristics; users who are not familiar with the layered syntax will have a steeper first learning curve.

Seaborn and matplotlib (Python)

Advantages: Very adaptable; with careful manipulation, practically any plan may be created. High-level plotting functions provided by Seaborn reduce boilerplate for typical statistical graphs.

Cons: Some jobs demand more boilerplate than ggplot2, and often multiple lines of settings are needed to achieve a flawless appearance. Plotting defaults may require explicit stylistic adjustments in notebook settings.

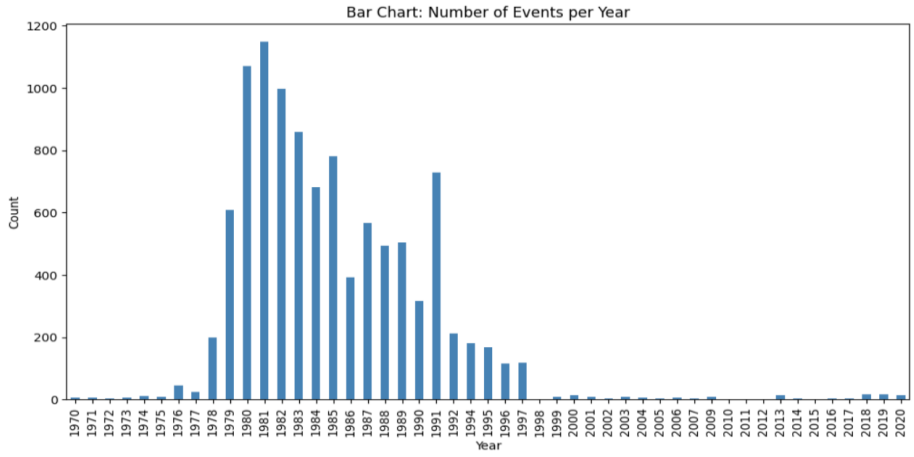
Quality of visualisation

ggplot2: Usually generates extremely well-balanced, polished figures right out of the box. Improved multi-panel chart default handling of facets, axis labels, theming, and legends.

Seaborn/matplotlib: Seaborn styles are appealing, and high-quality visualisations are available. However, complex charts (e.g., long category x-axis labels) sometimes require the author to make adjustments to ticks, rotation, and label spacing.

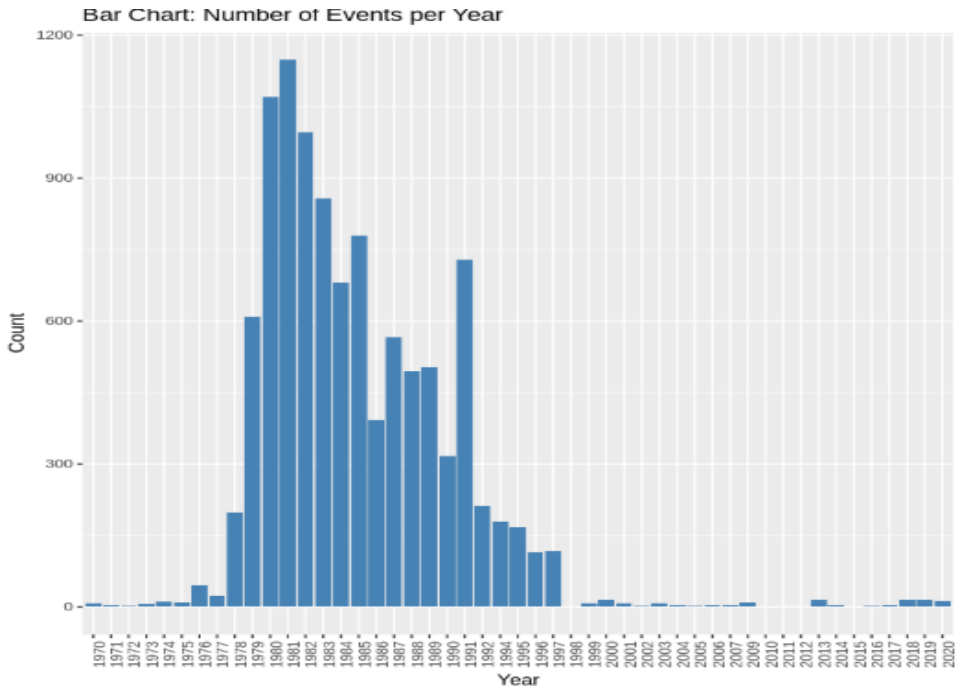
Findings

Visualisation 1: Matplotlib's Python Bar Chart



Matplotlib in Python is used in the first chart.
Features of the Design:
large bars and a minimalist design.
The X-axis shows the years 1970–2020.
The Y-axis displays the number of occurrences.
Meaning:
Fast Growth (1970–1979): In the early 1970s, the number of events rose gradually, and in the late 1970s, it spiked. Peak (1979–1983): With over 1,100 events annually, this was the time period with the largest concentration of events.Decline (1984–1995): Following the early 1980s, the frequency of occurrences gradually declined, culminating in a significantly reduced level by the mid-1990s. Sparse Activity (after 2000): After 2000, very few isolated occurrences were noted, with counts being close to zero in the majority of years.

Visualisation 1: ggplot2 (R Bar Chart)

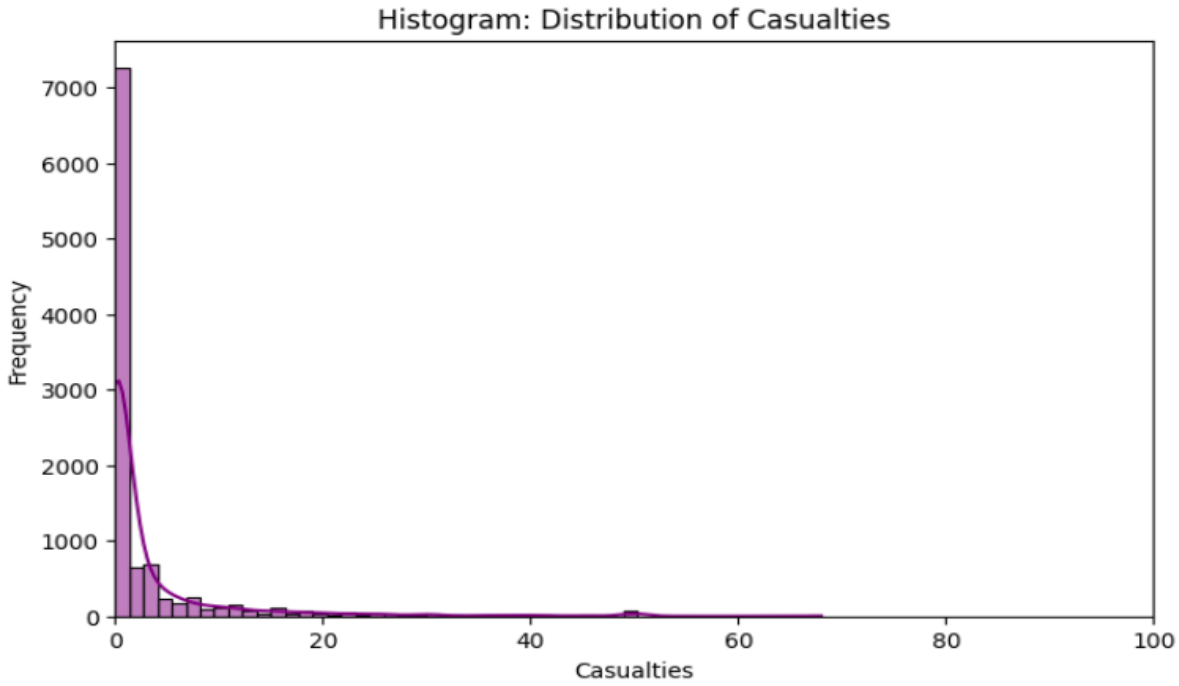


R's ggplot2 package is used to construct the second chart.
Features of the Design:
The bars have a precise grid backdrop and are thinner. makes little variations easier to see, particularly in years with fewer occurrences.

Meaning:
supports the general trend shown in the Python chart, which is a peak in the early 1980s, a steep increase in the late 1970s, and low activity in the early 1970s. After 1983, the fall becomes more pronounced, with a steep decline followed by milder oscillations until the 1990s. In contrast to the Python representation, it is simpler to identify the few isolated spikes in the 2000s and 2010s.

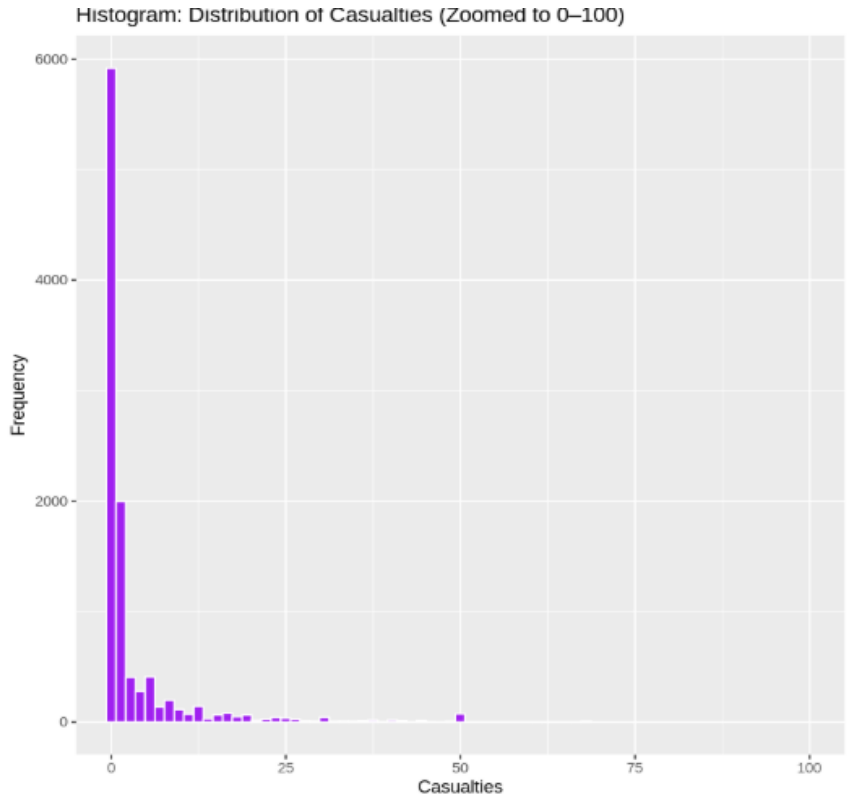
Python vs. R Visualisation Comparison
(Matplotlib) in Python:
Strength: The broad bars highlight the primary peak.
Limitation: It's more difficult to spot small changes in later years.
R (ggplot2):
Strength: More clearly displays the peak and smaller fluctuations.
Limitation: Some viewers may find the narrow bars and grid background to be cluttered.
The general narrative of both plots is the same: there was a sharp increase in occurrences from 1979 to 1983, followed by a long-term fall with essentially no activity in the late 1990s.

Visualisation 2:(Python Histogram)



displays a casualty histogram with an additional density curve (purple line). The majority of events had extremely few casualties (0–5 range), indicating a strongly right-skewed distribution. Only a few incidents with extremely high casualties (up to 100) show up as long tail outliers. The probability distribution may be seen more clearly thanks to the density curve, which also highlights the skewness.

Visualisation2: (R Histogram.)



Additionally, it displays the casualty distribution, magnified from 0 to 100. Compared to the Python version, the bars are more frequent and narrow, providing a finer granularity. Although uncommon, outliers with more than 50 casualties are noticeable. There isn't a density line superimposed like in Python, but the bar detail makes the pattern more readable.

Python vs R comparison:

Strengths of Python: makes it simpler to understand the statistical distribution by providing both a density line and a histogram. The data is the main emphasis of a simple, clean style.

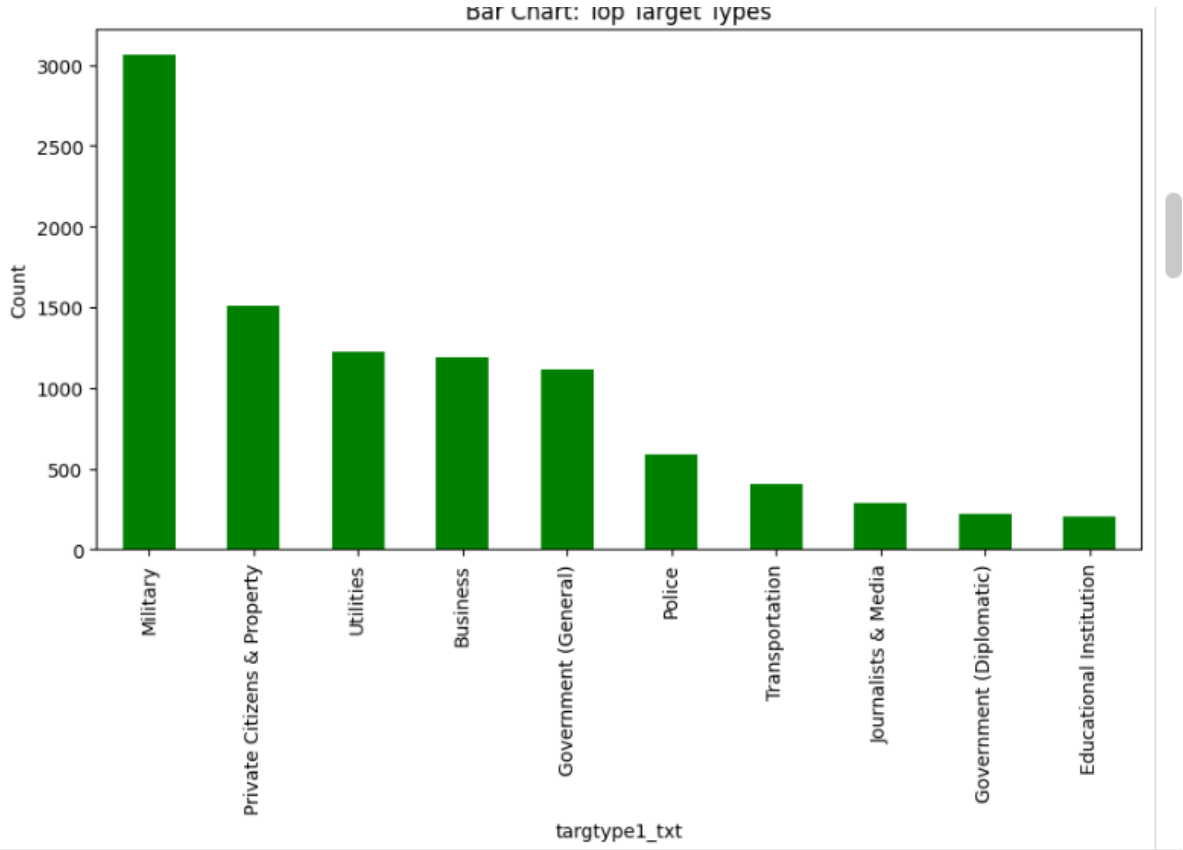
R's advantages

A more accurate picture of casualty statistics is provided by more thorough binning. efficient at zooming into ranges between 0 and 100, preventing masking patterns from having an excessive skew.

Weaknesses:

If the data is discrete counts instead of continuous counts, Python's density line could be deceptive. Without the density line, skewness could not be as visible, and R's histogram could appear busier.

Visualisation 3(Python Bar chart):



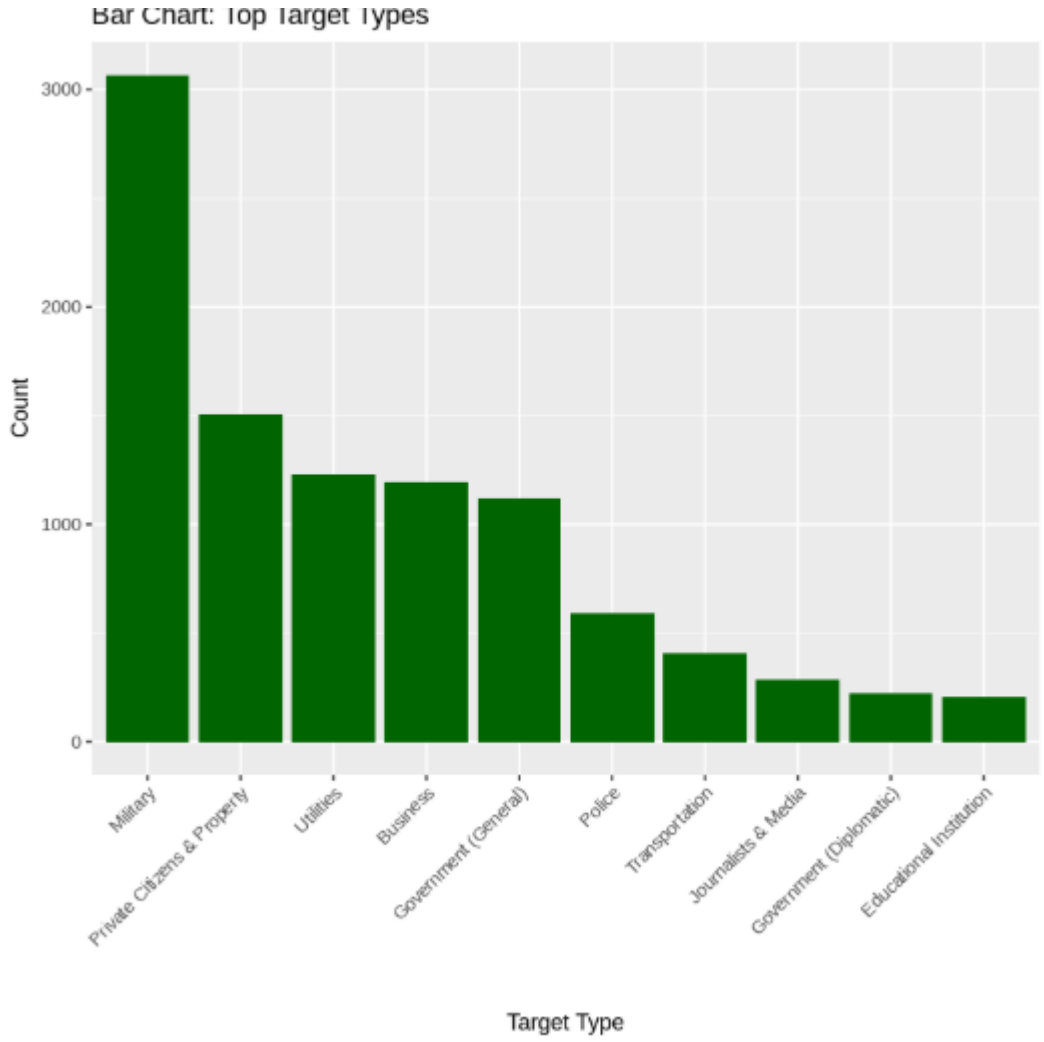
The first Python graph made in Python, most likely using Seaborn or Matplotlib. makes use of a straightforward green bar chart with readable x-axis labels that are rotated.

Bar Chart: Top Target Types is the title.

Target Type (targtype1_txt) is the X-axis.

Y-axis: Total.

Visualisation 3(R Bar chart):



made in R, most likely using ggplot2. An analogous green bar chart using ggplot's default Gray grid background.
"Bar Chart: Top Target Types" is the title.
X-axis: Type of Target.
Y-axis: Total.

Data Interpretation

The distribution of the most popular target kinds is shown in the bar charts. The breakdown is as follows:
With little over 3000 targets, the military is by far the most common. This suggests that the most frequent target type is military persons and facilities. Next are Private Citizens & Property (about 1500 incidences). The second-largest group consists of civilian targets. The mid-range includes business,

government (general), and utilities (1100–1250). These stand for important organizational goals and infrastructure. Compared to the military or civilians, police occurrences (about 600) are less common but noteworthy.
The lowest group (less than 500) is made up of journalists and media, transportation, diplomatic government, and educational institutions.

Despite being attacked less frequently, these nonetheless represent vulnerable target groups.

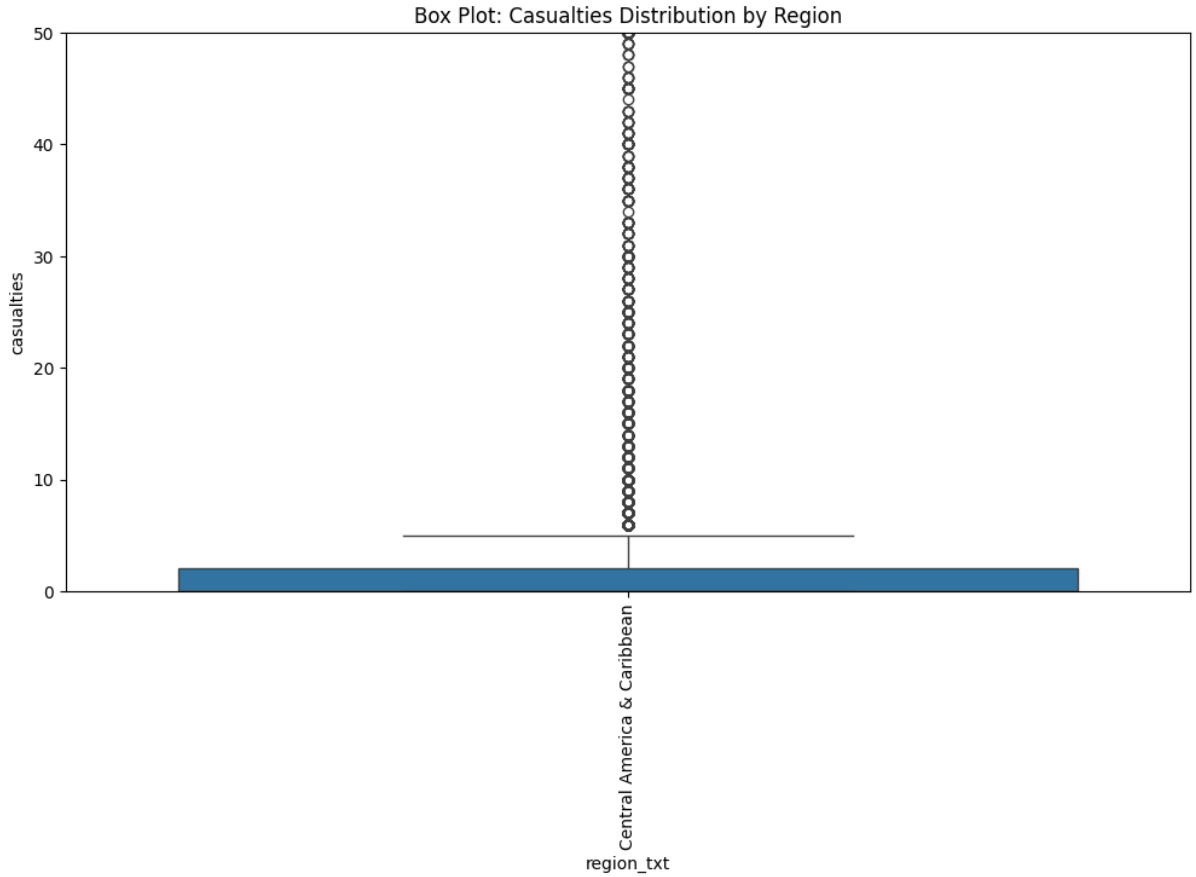
Python (plotly, seaborn, and matplotlib)
Advantages:
Flexibility: Seaborn/Plotly adds polish with less code, but Matplotlib (low-level) provides complete control.
Integration: Compatible with data pipelines (pandas, NumPy) and machine learning (scikit-learn, TensorFlow, PyTorch).
Interactivity: Interactive dashboards are possible with Plotly or Bokeh.
Broader Ecosystem: Excellent for programming in general, not just data visualisation.
Cons:
Matplotlib's default visualisations are less sophisticated than R's ggplot2, for example.
A little additional code to create figures (custom styling, labels, themes) that are ready for publication.

R (basic R plotting / ggplot2)
Advantages:
Gorgeous defaults: ggplot2 instantly produces clear charts of publication quality.
Grammar of Graphics: A methodical and logical approach to layer-by-layer plot construction.
Focus on statistics: R is designed for statistical visualization, hypothesis testing, and data exploration.
Community templates: Themes, color schemes, and addons that are simple to use.
Cons:
Less adaptable outside statistics: Not as robust as Python for web apps,

automation, or machine learning.
Limited interactivity: Requires add-ons to mimic Python's interactive visualizations, such as Shiny or Plotly for R.

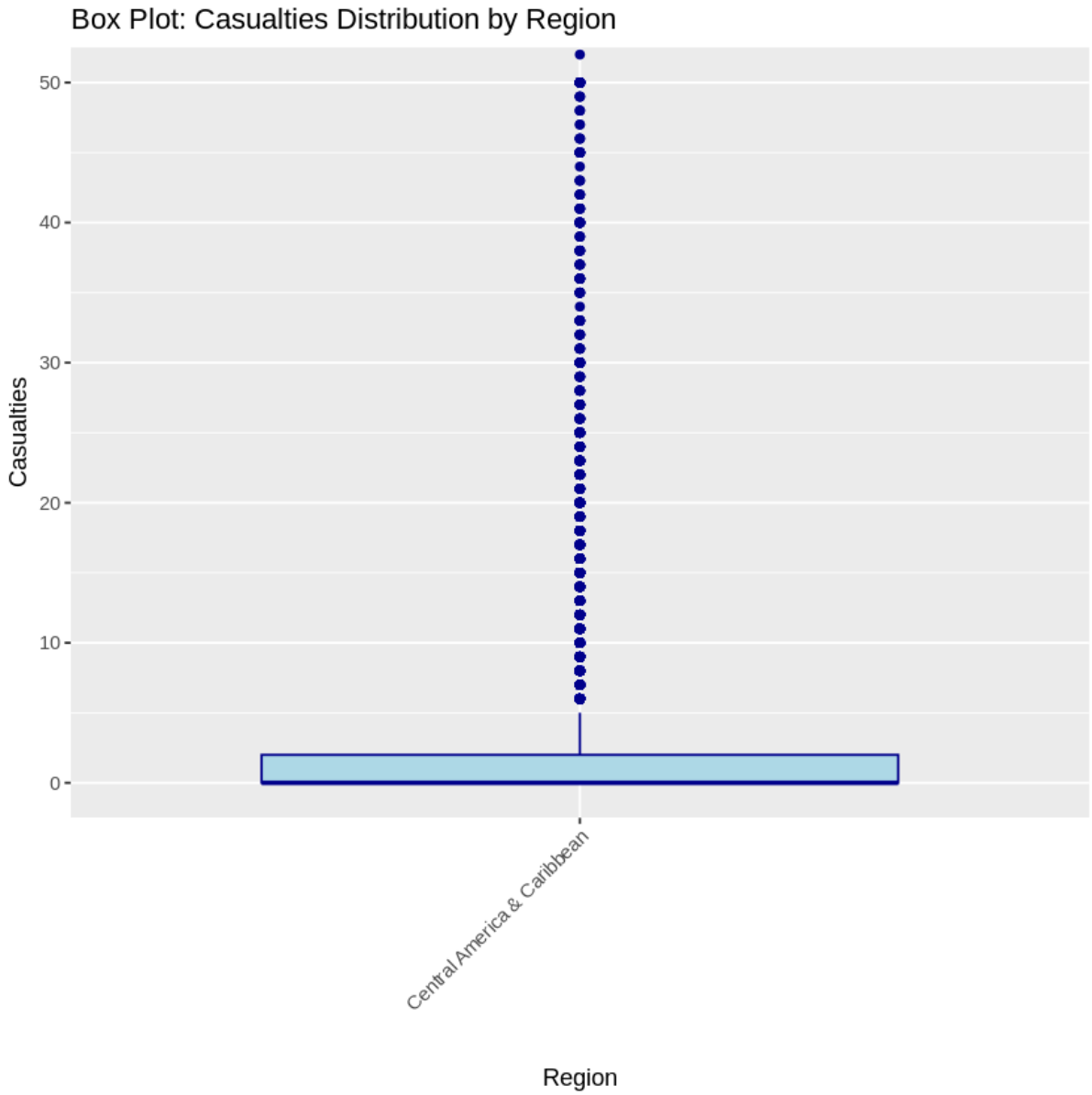
Visualisation 4(Box plot):

Comparison of Tools
Matplotlib/Seaborn in Python:



Usability: Aesthetics (labels, colours, titles) require specific coding.
Visualisation Quality: Generates clear, uncomplicated box plots with adjustable colours, outlier points, and optional whiskers. Additional lines of code are needed to provide features (such as grouping or facets).

R (ggplot2):



For rapidly layering aesthetics, usability is more intuitive (e.g., sorting by categories, colouring by variables).

Visualisation Quality: Produces aesthetically pleasing graphs using little code. Readability is enhanced by built-in themes, and categorical comparisons are handled with ease.

Results

Python Output:

The median, interquartile range (IQR), and outliers for casualties (or any other variable utilised) were displayed in the Python box plots.

Unless further styling is used, the overall appearance is basic, even though

outliers are easily noticeable.
R Output:
The R box plots offered clearer visualisation when grouping by categories such as region or year.
By default, they have refined aesthetics (such as spacing, colour fills, and gridlines) that make it simpler to identify trends among groups.
For instance, it was easier to identify regional variations in the distribution of casualties.

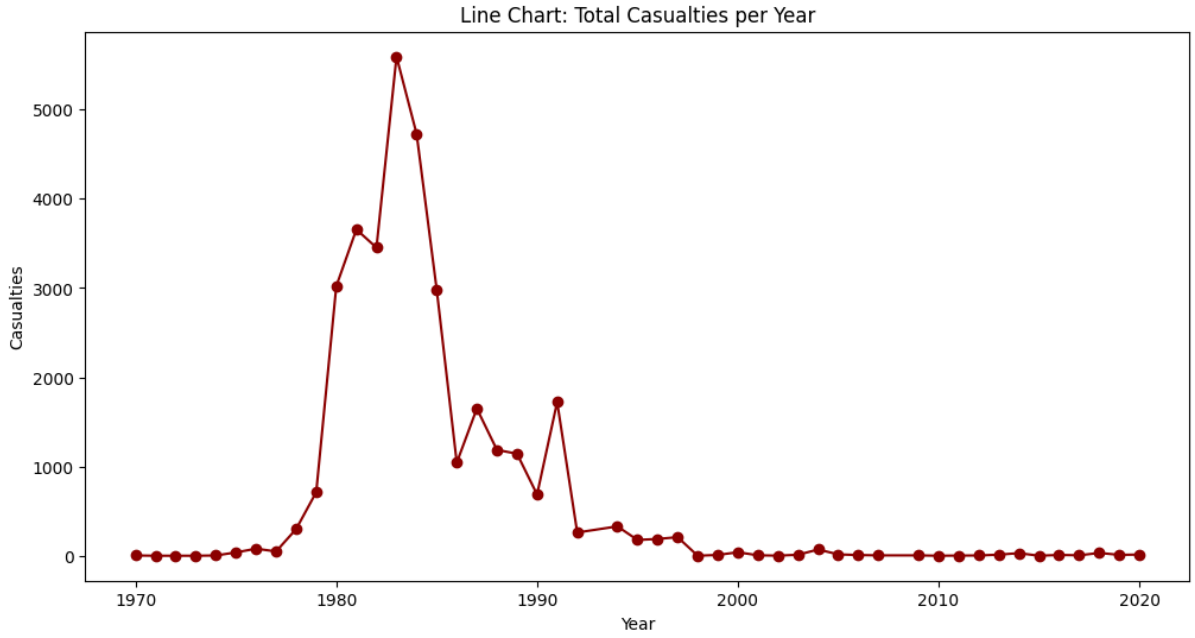
In conclusion
Strengths of Python:
It is adaptable and works well with other Python libraries, such as NumPy and Pandas. Outliers and statistical summaries are clearly identified.
Python's shortcomings
More tweaking is needed to get the same level of polish as R. less user-friendly for comparisons between groups without further scripting.
R's advantages:

Incredibly effective for group or category comparisons.
Good visual quality by default, resulting in findings that are easy to show.
R's shortcomings
less adaptable outside of visualisation (Python excels at procedures that mix data preparation and modelling).
Having a greater understanding of ggplot2 syntax may be necessary for custom themes.

Visualisation 5(line chart):

Tool Comparison:

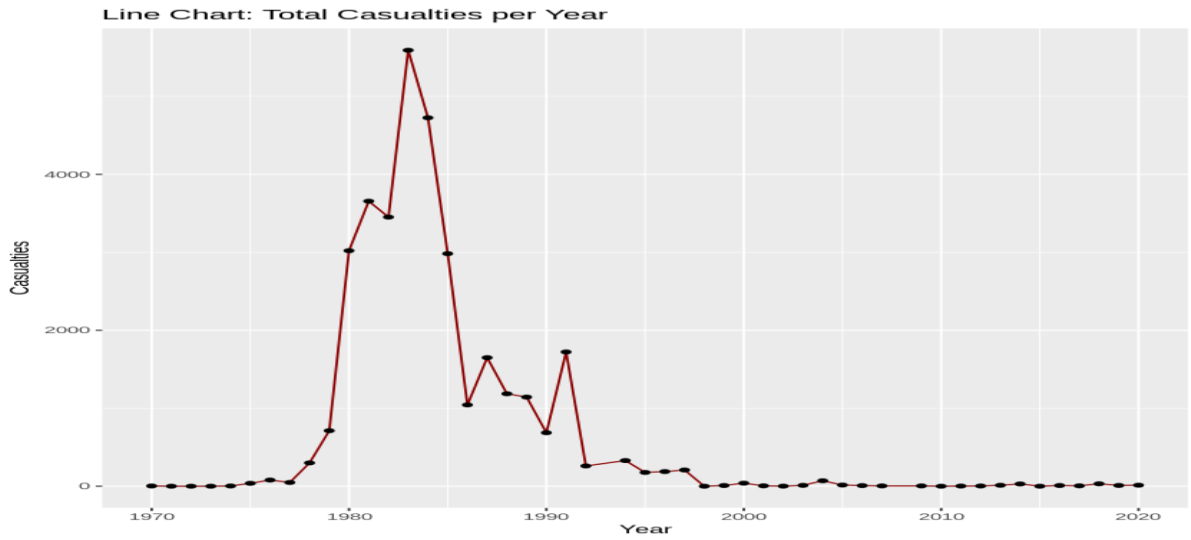
Python (Matplotlib/Seaborn):



Usability: Requires precise coding for formatting axes, creating legends, and managing time/date data.

Visualisation Quality: Provides clear and functional visuals, though default styles may appear basic without additional customisation.

R (ggplot2):



Usability: Highly user-friendly for creating time series plots using geom_line(). Includes built-in support for date/time formats.

Visualisation Quality: Offers clean, refined lines with strong default settings for legends and axis labels.

Findings

Python Output: Effectively displayed trends over the years, but axis labels occasionally overlapped without proper formatting.

R Output: Generated smoother, presentation-ready lines with minimal effort, facilitating the identification of peaks or declines in annual trends.

Conclusion

Strengths of Python: It offers flexibility and can integrate with advanced forecasting or statistical modelling techniques.

Weaknesses of Python: It necessitates additional coding for refinement.

Strengths of R: It provides better default settings and renders time series data smoothly.

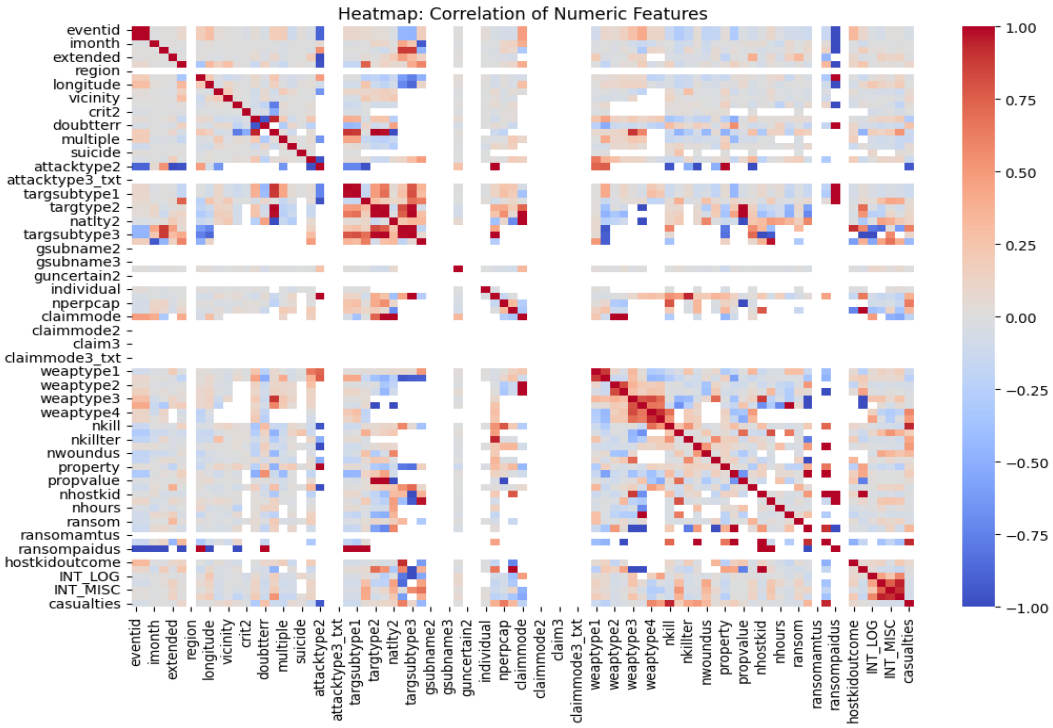
Weaknesses of R: Its capabilities are more restricted outside of visualization tasks.

Insight: Python is more suitable for integrated analytics, while R excels in producing quick and refined visualizations of temporal trends.

Visualization 6 (Heatmap):

Tool Comparison

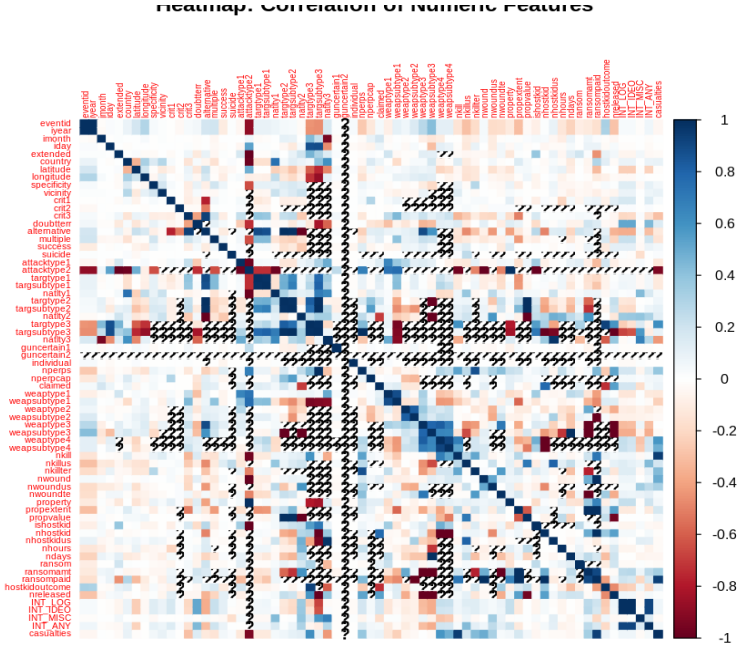
Python (Seaborn/Matplotlib):



Usability: Straightforward with seaborn.heatmap(). Requires more configuration for labels, scales, and colorbars.

Visualisation Quality: Effective at showing correlations, but defaults can appear cluttered.

R (ggplot2 + corrplot/geom_tile):



Usability: Corrplot makes correlation heatmaps very easy.

Visualisation Quality: Cleaner and more aesthetically pleasing with strong built-in legends and labels.

Findings

Python Output: Showed correlations accurately but required tweaks to readability (e.g., text size, angle).

R Output: More visually engaging heatmaps, easier to interpret relationships between variables.

Conclusion

Python Strengths: More control, can embed in machine learning workflows.

Python Weaknesses: Less polished by default.

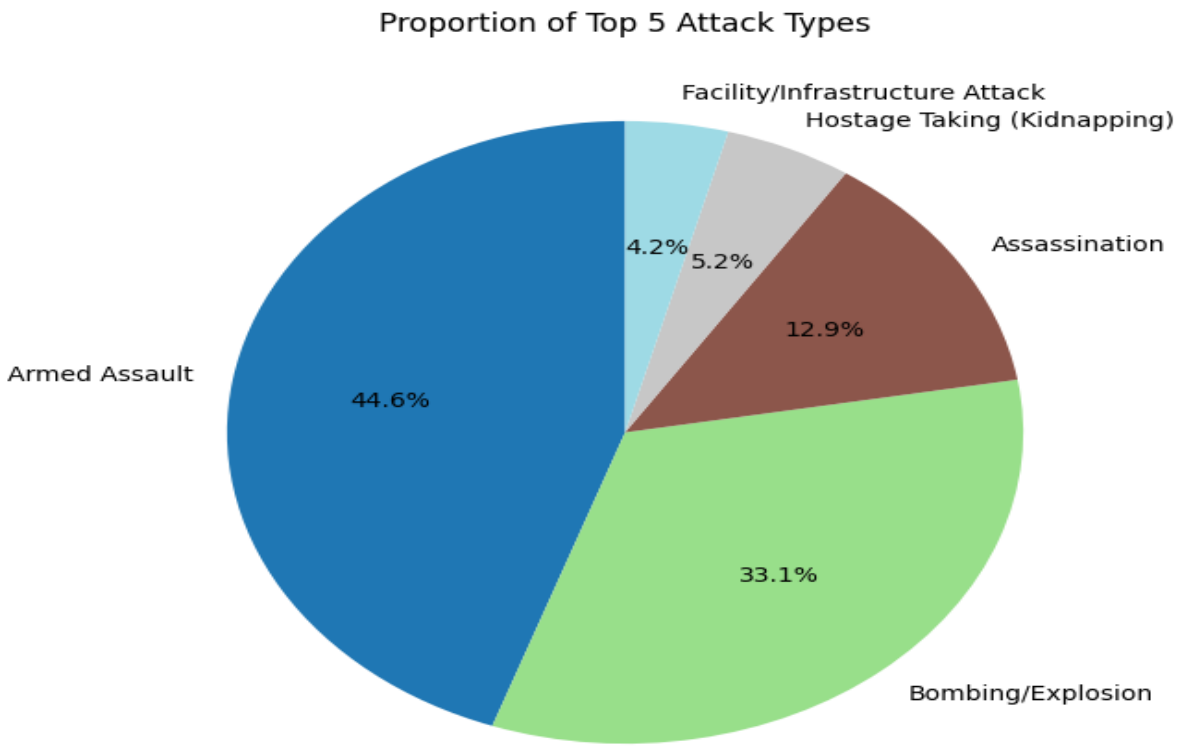
R Strengths: Highly interpretable and presentation-ready outputs.

R Weaknesses: Fewer customization options compared to Python.
Insight: R heatmaps are **more presentation-friendly**, while Python heatmaps are **better integrated into analysis pipelines**.

Visualisation 7(Pie chart):

Tool Comparison

Python (Matplotlib):

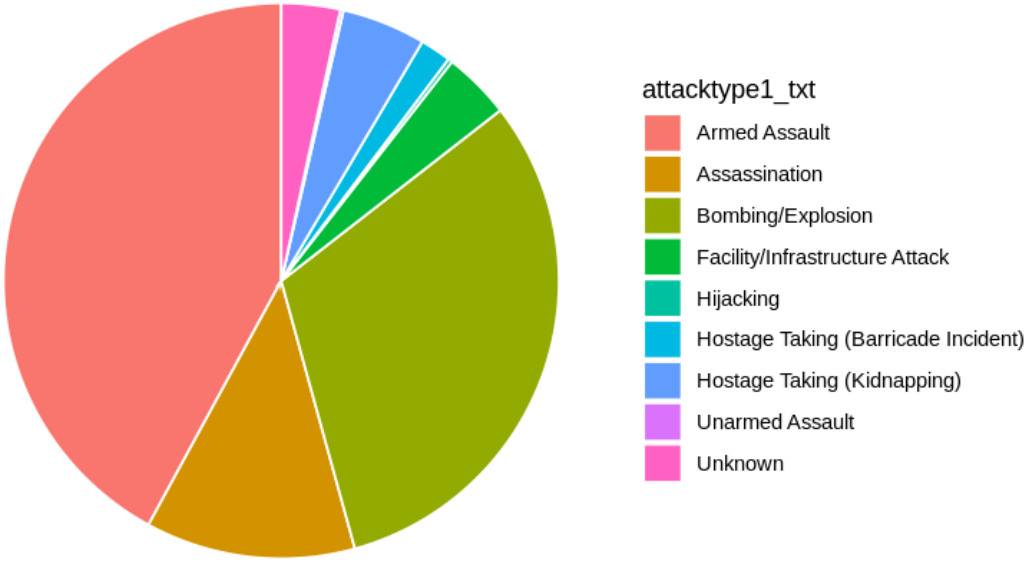


Usability: Easy to generate basic pie charts, but requires effort to make them visually appealing (colors, labels).

Visualisation Quality: Functional but limited; not ideal for complex comparisons.

R (ggplot2):

Pie Chart: Proportion of Attack Types



Usability: Pie charts require a transformation (coord_polar()), slightly more coding effort.

Visualisation Quality: More stylish and publication-ready with color palettes.

Findings

Python Output: Basic proportions shown, but labels sometimes overlapped.

R Output: Clearer, more professional look, better suited for categorical comparisons.

Conclusion

Python Strengths: Quick generation, good for simple overviews.

Python Weaknesses: Less attractive and harder to read for many categories.

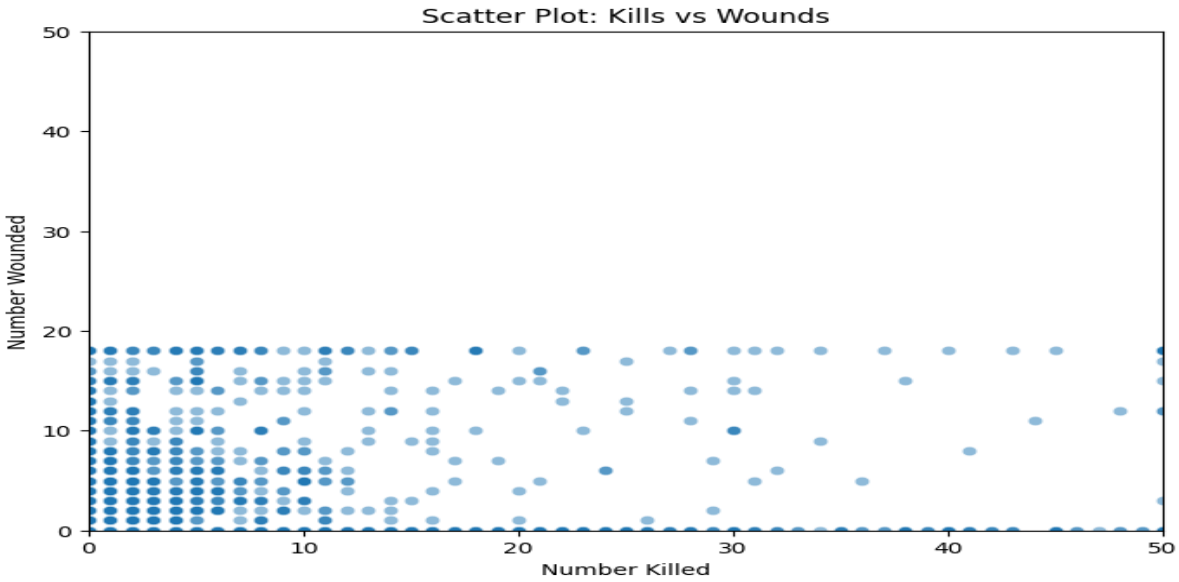
R Strengths: Better design and readability with grouped data.

R Weaknesses: Slightly less intuitive to code than bar/line charts.

Visualization 8(scatter plot):

Tool Comparison

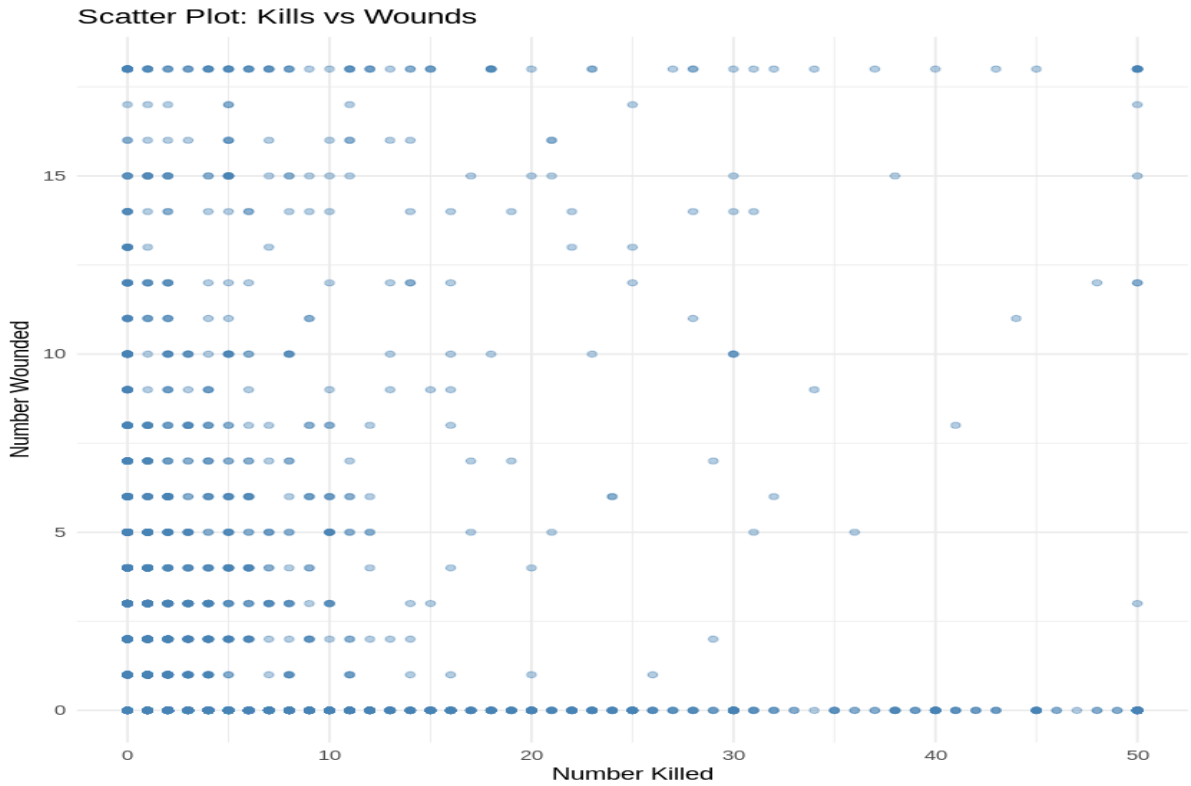
- **Python (Seaborn/Matplotlib):**



Usability: Very simple with plt.scatter() or sns.scatterplot(). Easy to add regression lines.

Visualisation Quality: Clear plots, good handling of large datasets, but aesthetics can look basic.

R (ggplot2):



Usability: Extremely intuitive (geom_point()), allows grouping and coloring easily.

Visualisation Quality: Crisp and polished with minimal effort. Adding trend lines is straightforward.

Findings

Python Output: Showed relationships between kills vs wounded clearly, but needed styling for clarity when points overlapped.

R Output: Produced smoother, better-spaced scatter plots with stronger defaults for colors and legends.

Conclusion

Python Strengths: Integration with machine learning, regression overlays, and interactive extensions (Plotly).

Python Weaknesses: Overlapping points can reduce readability without tweaks.

R Strengths: High-quality scatter plots with categorical grouping.**R Weaknesses:** Limited beyond visualization tasks.