Mass shootings in the US

1. Web-scraping

The Python script scrapes mass shooting data from the Gun Violence Archive website "https://www.gunviolencearchive.org/mass-shooting?page={}" by sending requests to each page, extracting information such as Incident ID, Date, State, City, Address, number of people killed, and injured. It uses the `requests` library to fetch the webpage and `BeautifulSoup` to parse the HTML content and locate the relevant data in a table. The script processes data for a total of 80 pages, formats the date, and collects all the data in a list. Finally, the script saves the scraped data into a CSV file using `pandas` for further analysis. Error handling is implemented to skip incomplete rows or handle issues during the scraping process.

2. Geocoding(adding longitude and latitude)

The Python script processes the scraped mass shooting data by adding geographic coordinates (latitude and longitude) to the CSV file. It reads the 'City or County' and ' State' columns from the existing file and using the `geopy` library's `Nominatim` geocoder, the script converts city and state information into latitude and longitude. It also includes logging to track the process and handle errors. If geocoding fails for a particular location, it logs a warning. The script then adds the latitude and longitude values to the data and saves the updated information in a new CSV file. The script includes a delay between requests to avoid hitting geocoding service rate limits.

In this script, since it's only geocoding a relatively small number of locations, using `Nominatim` which allows you to convert place names into geographic coordinates without needing an API key is sufficient for this use case.

3.Visualizations/Dashboard

Using Python script and Jupyter Notebook, the script creates a Dash-based interactive web application to visualise the scraped data, with such visualizations as a map, time series plot and word cloud. The first steps include loading the CSV file, processing some variables for later use and creating additional column for the year variable. For creating dynamic elements visualisation functions are defined with `draw_map`, `draw_time_series`, and `generate_wordcloud`. The app layout with visualisations is updated with additional components like `Dropdown` and `RangeSlider`, providing user interactivity. Finally, Dash callbacks handle dynamic updates to the visualisations, such as filtering by year, selecting variables, or adjusting incident ranges.