HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT REPORT

GLOBAL TERRORISM DASHBOARD

Course: Data Visualization

Supervisor: Professor Tran Viet Trung

Authors:	Student ID:
Nguyen Duc Binh	20225475
Tran An Khanh	20225447
Doan Anh Vu	20225465
Luong Huu Thanh	20225458
Vu Trung Thanh	20220066

Hanoi, June 2024

Contents

1 Introduction

1.1 About the Dataset

The Global Terrorism Database™ (GTD) is an open-source database including information on terrorist events around the world from 1970 through 2020 (with additional annual updates planned for the future). Unlike many other event databases, the GTD includes systematic data on domestic as well as transnational and international terrorist incidents that have occurred during this time period and now includes more than 200,000 cases. For each GTD incident, information is available on the date and location of the incident, the weapons used and nature of the target, the number of casualties, and–when identifiable–the group or individual responsible.

Statistical information contained in the Global Terrorism Database is based on reports from a variety of open media sources. Information is not added to the GTD unless and until we have determined the sources are credible. Users should not infer any additional actions or results beyond what is presented in a GTD entry and specifically, users should not infer an individual associated with a particular incident was tried and convicted of terrorism or any other criminal offense. If new documentation about an event becomes available, an entry may be modified, as necessary and appropriate.

The National Consortium for the Study of Terrorism and Responses to Terrorism (START) makes the GTD available via this online interface in an effort to increase understanding of terrorist violence so that it can be more readily studied and defeated.

1.2 Characteristics of the GTD

- Contains information on over 200,000 terrorist attacks
- Currently the most comprehensive unclassified database on terrorist attacks in the world.
- Includes information on more than 88,000 bombings, 19,000 assassinations, and 11,000 kidnappings since 1970.
- Includes information on at least 45 variables for each case, with more recent incidents including information on more than 120 variables.
- More than 4,000,000 news articles and 25,000 news sources were reviewed to collect incident data from 1998 to 2017 alone.

Our project aims to leverage data visualization techniques to analyze and interpret the rich information contained within the GTD. By visually representing key trends, geographical distributions, and attack characteristics, we seek to uncover underlying patterns and dynamics of terrorism, shedding light on its evolving nature and identifying areas for further research and intervention.

Through interactive visualizations and insightful analyses, we endeavor to enhance understanding of terrorist violence, facilitate knowledge dissemination, and contribute to efforts aimed at countering the global threat of terrorism.

2 Data Preprocessing

In this section, we outline the steps taken to preprocess the Global Terrorism Database (GTD) for visualization purposes. The preprocessing steps are essential to streamline the dataset and prepare it for insightful data visualization.

- **Column Removal**: Initially, we remove columns that are considered unnecessary for our visualization objectives. These columns typically contain data that would not contribute significantly to the insights we aim to derive (subtypes of weapon types, target types, or columns that only serve to label another text column using ids). By eliminating such columns, we streamline the dataset for more efficient processing and visualization.

- **Handling Missing Values**: Handling missing data is crucial to ensure the integrity of our analysis. We address missing values (NA) by filling them with appropriate default values. This step ensures that our dataset is complete and ready for visualization without compromising the quality of insights due to missing information.

- **Feature Engineering**: To enhance the dataset for visualization, we perform feature engineering to create new columns that facilitate effective plotting and analysis. One notable feature includes the creation of a specific date column derived from the existing columns for date, month, and year. Additionally, we generate a country code column (country_code), which maps the country names from the original dataset to their corresponding ISO codes. This mapping is crucial for generating choropleth maps, providing geographical insights into terrorism incidents across different regions.

3 Dashboard Design

3.1 Tools

Our project utilizes the Dash framework in conjunction with the Plotly visualization library to develop an interactive web-based dashboard for visualizing global terrorism data. Dash provides a Python framework for building web applications, while Plotly offers a powerful set of tools for creating interactive plots and visualizations.

3.2 Dashboard Structure

The visual design of our dashboard is carefully crafted to enhance the user experience and ensure effective data presentation. Key design elements include:

- **Background:** The dashboard uses a black background with subtle drops of blood, creating a stark and impactful visual environment. This use of dark colors and the blood drop motif align with the serious and somber nature of the global terrorism data being presented. The background is made slightly dim to ensure that the plots and visualizations remain the focal point, standing out clearly against the backdrop.

- **Color Theme:** A consistent color theme of dark red is employed throughout the dashboard. This color is chosen for its strong visual impact and thematic relevance to the subject matter. The dark red color is used for various elements including charts, icons, and text highlights, providing a cohesive and striking visual experience.
- **Thematic Consistency:** The use of dark colors and the blood drop motif align with the serious and somber nature of the global terrorism data being presented. This thematic consistency helps to reinforce the gravity of the subject matter while ensuring that the dashboard is visually engaging.

Our web application is structured into distinct pages, each focusing on specific aspects of the global terrorism dataset. This is the code in our main application. Here, we initialize the Dash application, load our style sheets, and organize the other pages and the corresponding buttons to redirect to these different pages:

```
app = Dash(__name__, external_stylesheets=[dbc.themes.COSMO,
    'assets/gtd.css'], use_pages=True)
app.title = "Global Terrorism Dashboard"
# Define the desired order of the pages
page_order = ['Introduction', 'Home', 'Geo dashboard', 'Time dashboard']
page_dict = {page['name']: page for page in dash.page_registry.values()}
ordered_pages = [page_dict[name] for name in page_order if name in
    page_dict]
app.layout = html.Div([
    html.H1(
        children='Global Terrorism Dashboard',className='title',
        style={'textAlign':'center'}),
    html.Div([
        html.Div(
            dbc.Button(
                children=f"{page['name']}",
                id={'type': 'page-button', 'index': page['relative_path']},
                href=page["relative_path"],className='page-button',
                style={'margin-right':'10px', 'borderColor': 'white',
                    'borderRadius': '10px'}) # Padding between buttons
        ) for page in ordered_pages
    ], style={'display':'flex', 'justifyContent':'center',
        'flexWrap':'wrap', 'margin-bottom':'38px'}),
    dash.page_container
], style={
    'background-image':'url("assets/dark_bg1.PNG")',
    'background-size':'cover',
    'background-repeat':'no-repeat',
    'background-position':'center',
    'background-attachment': 'fixed',
    'height': '200vh',
    'top': 0, 'left': 0,
    'z-index': -1})
```

The following sections outline the purpose and content of each page:

3.2.1 Introduction Page

The Introduction Page is the gateway to our web application, offering users a summary of the dataset and the dashboard's features and capabilities. It presents a concise overview of the project and includes guidance on how to explore the different sections of the dashboard.

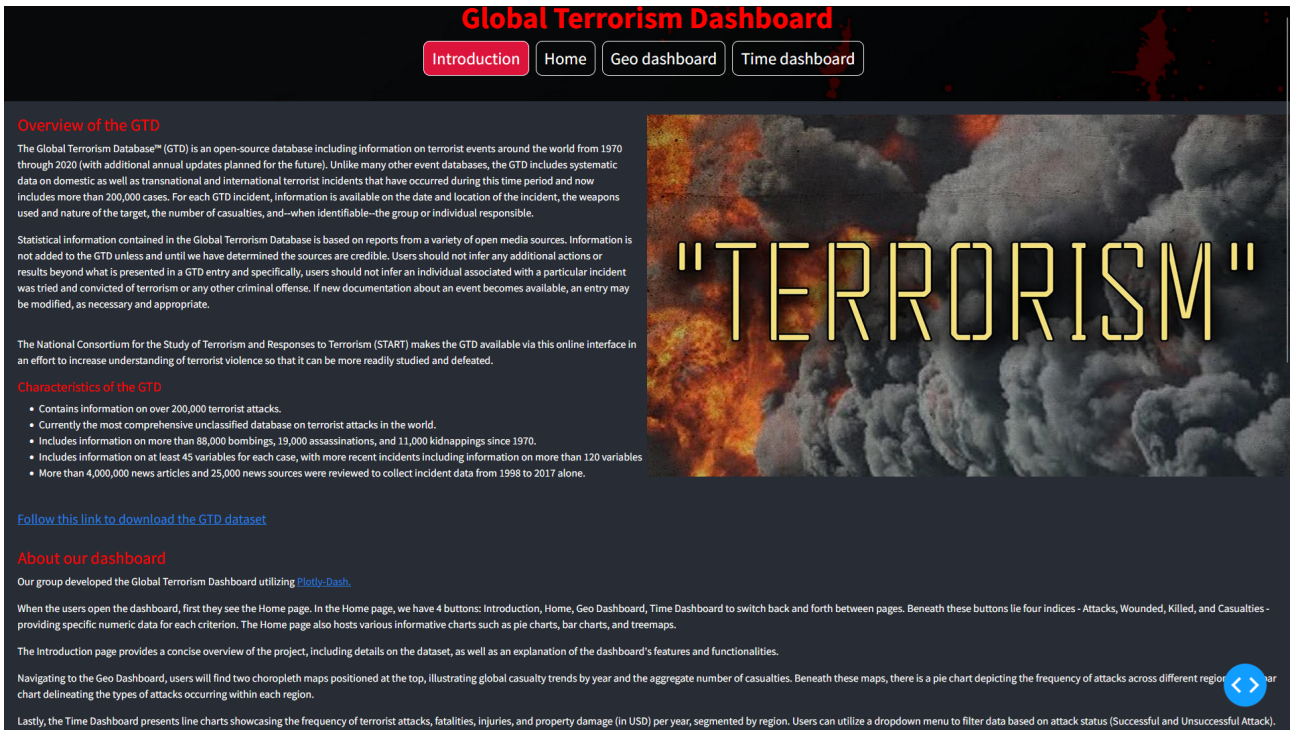

Figure 1: *The Introduction Page*

3.2.2 Home Page

The Home Page serves as the central hub of the dashboard, featuring general visualizations and key indices related to global terrorism. Here, users can explore high-level trends and statistics, such as the total number of terrorist incidents, casualty counts, and who the most notorious groups are.

- **Key Indices:**
  - **Total number of people killed**
  - **Total number of people wounded**
  - **Total casualties**
  - **Total number of attacks**

  These metrics provide users with an immediate understanding of the overall impact of terrorism. By presenting these indices prominently, users can quickly grasp the scale and severity of terrorist activities worldwide. In our project, they are organized into the same

Div row right at the top, so users can see them right away after navigating to the home page, and they will set the stage for more detailed visualizations below

- **Donut Charts:**
  - **Target Type Distribution:** This donut chart illustrates the proportion of different target types affected by terrorist attacks. It helps users understand which types of targets (e.g., civilians, military, government) are most frequently targeted, providing insights into the strategic focus of terrorist groups.

    Step-by-step Description

    Create new dataframe consist the only the target type's name and it's count.

```
target_type = df.groupby('target_type')['target_type'].count()
tar_type_df = pd.DataFrame({'target_type': target_type.index,
                'total': target_type.values})
tar_type_df = tar_type_df.sort_values(by='total', ascending=False)
```

    However, the number of target type is big, and we handle it by only keeping the first five rows, concatenate with one more rows, which is called `'Others'` with the total values is summary of remain target type's value.

```
top_5 = tar_type_df.head(5)
others = tar_type_df.iloc[5:]

others_total = others['total'].sum()
others_df = pd.DataFrame({'target_type': 'Others', 'total':
    others_total}, index=[0])

tar_type_df = pd.concat([top_5, others_df], ignore_index=True)
tar_type_df = tar_type_df.sort_values(by='total', ascending=False)
```

    Create a Plotly figure titles 'Target's type distribution' with the dataframe `'tar_type_df'`

```
fig_target_type = px.pie(
    data_frame = tar_type_df,
    names = 'target_type',
    values = 'total',
    hole = 0.5,
    color='target_type',
    color_discrete_sequence=px.colors.sequential.Oryel,
)
```

    * **Line 2:** Specify the data source for the pie chart.
    * **Line 3:** Set the column `'target_type'` from the dataframe as the labels for the pie chart slices.

* **Line 4:** Set the column `'total'` from the dataframe as the values for the pie chart slices.
* **Line 5:** Specify the size of the hole in the center.
* **Line 6:** Color the pie chart slices based on the `'target_type'` column values.
* **Line 7:** Set the color sequence for the pie chart slices.

And we add some update to get the better look in the home page

```
fig_target_type.update_layout(
    title_text = "Target type's distribution",
    title_x = 0.15,
    title_y = 0.975,
    paper_bgcolor='rgba(0,0,0,0)',
    plot_bgcolor='rgba(0,0,0,0)',
    font=dict(family='Arial',size=14, color='gray'),
    margin=dict(l=0, r=0, t=0, b=0),
    legend={
        "x":0.9,
        "y":0.5,
        "xref":"container",
        "yref":"container"
})
fig_target_type.update_traces(textinfo='percent', textfont_size=12,
                  marker=dict(line=dict(color='#000000', width=2)))
```

* **Line 2:** Set the title text of the figure.
* **Line 3, 4:** Set the position of the title.
* **Line 5:** Set the background color of the entire figure.
* **Line 6:** Set the background color of the plotting area.
* **Line 7:** Set the font properties for the figure.
* **Line 8:** Set the margins around the figure.
* **Line 9-13:** Customize the legend's position.
* **Line 15:** Set the text information displayed on the traces to show percentages. Set the font size of the text displayed.
* **Line 16:** Customize the markers for the traces.

Output:

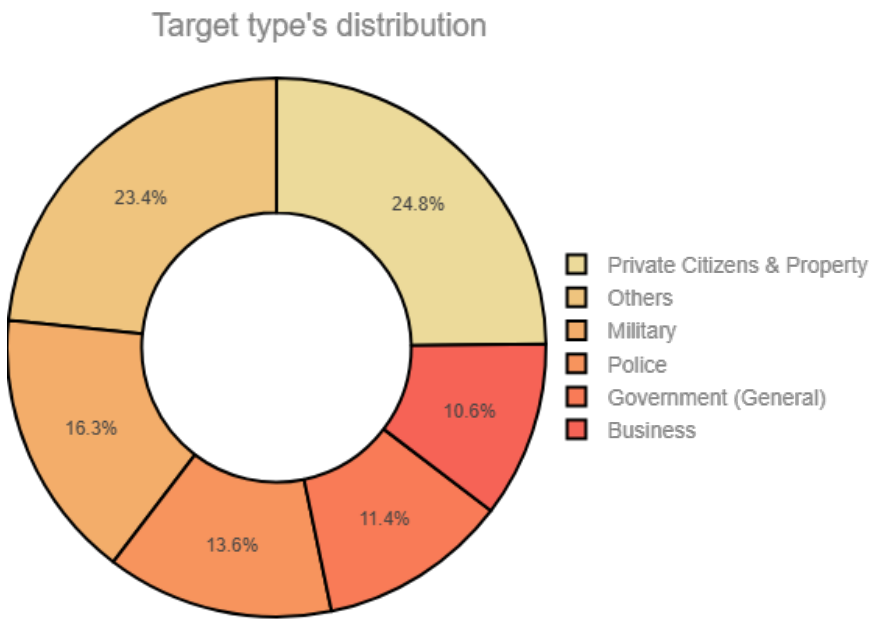

Figure 2: *'Target type distribution' Donut chart*

– **Attack Type Distribution:** This donut chart shows the distribution of various attack methods used by terrorists. By visualizing the prevalence of different attack types (e.g., bombings, armed assaults, hijackings), users can gain an understanding of the common tactics employed in terrorist activities.

The step to create this donut chart is similar to the `Target type chart` above.

Step-by-step Description

```
attack_type = df.groupby('attack_type')['attack_type'].count()
att_type_df = pd.DataFrame({'attack_type': attack_type.index,
                            'total': attack_type.values})
att_type_df = att_type_df.sort_values(by='total', ascending=False)

top_5 = att_type_df.head(5)
others = att_type_df.iloc[5:]
others_total = others['total'].sum()
others_df = pd.DataFrame({'attack_type': 'Others', 'total':
    others_total}, index=[0])

# Concatenate top_5 and others_df
att_type_df = pd.concat([top_5, others_df], ignore_index=True)
att_type_df = att_type_df.sort_values(by='total', ascending=False)
```

```
fig_attack_type = px.pie(
    data_frame = att_type_df,
    names = 'attack_type',
    values = 'total',
```

```
    hole = 0.5,
    color='attack_type',
    color_discrete_sequence=px.colors.sequential.Oryel
)
fig_attack_type.update_layout(
    title_text = "Attack type's distribution",
    title_x = 0.15,
    title_y = 0.975,
    paper_bgcolor='rgba(0,0,0,0)',
    plot_bgcolor='rgba(0,0,0,0)',
    font=dict(family='Arial',size=14, color='gray'),
    margin=dict(l=0, r=0, t=0, b=0),
    legend={
        "x":0.9,
        "y":0.5,
        "xref":"container",
        "yref":"container"
})
fig_attack_type.update_traces(textinfo='percent', textfont_size=12,
      marker=dict(line=dict(color='#000000', width=2)))
```

Output:

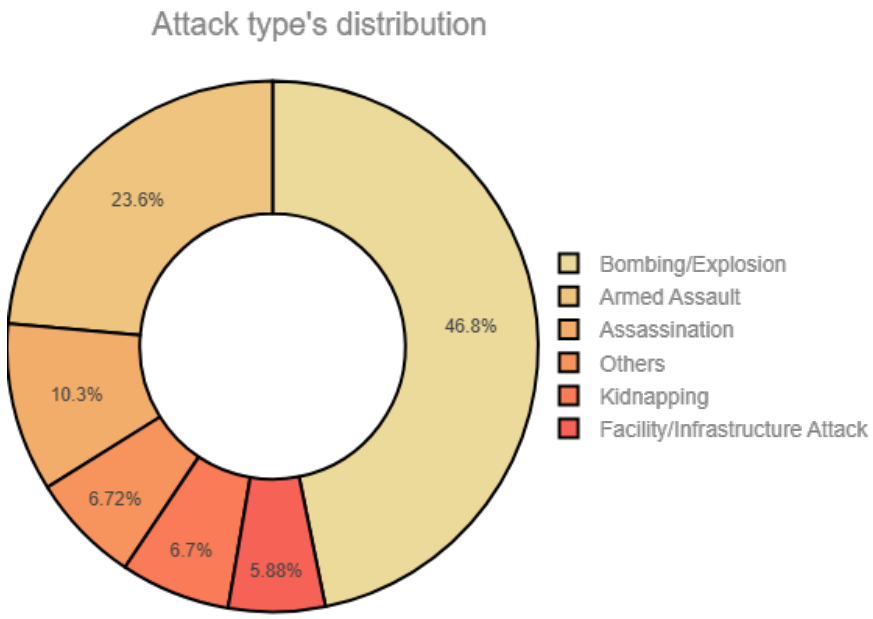

Figure 3: *'Attack type distribution' Donut chart*

- **Treemap:**
  - **Total Success and Failure by Region:** A treemap is employed here for understanding the distribution of successful and failed attacks among various regions. This

visualization effectively represents hierarchical data and allows users to compare the volume of successful, failed attack across different regions at a glance. The use of a treemap helps visualize the distribution of successful and failed attacks across different regions. By comparing the size and optionally the color of each region's rectangle, users can quickly identify areas with high attack volume and differentiate between successful and unsuccessful attempts

Step-by-step Description

From dataframe exist, create three new dataframe correspondings to success, failure and total attacks for each region.

```
success_attack = df[df['success'] ==
    1].groupby('region_txt')['success'].count()
suc_att_df = pd.DataFrame({'region_txt': success_attack.index,
    'success': 'success', 'total': success_attack.values})

failure_attack = df[df['success'] ==
    0].groupby('region_txt')['success'].count()
fal_att_df = pd.DataFrame({'region_txt': failure_attack.index,
    'success': 'failure', 'total': failure_attack.values})

tot_att_df =pd.concat([suc_att_df, fal_att_df])
```

The concatenated dataframe will contain region column, success column, which own binary value: 'success', 'failure', total number of attacks.

Create a Plotly figure titled 'Total of attacks by Region'. It takes the `tol_att_df` as the DataFrame, The `Path` show the hierarchical structure of the data (regions then success).

```
fig_success_by_region = px.treemap(
    data_frame=tot_att_df,
    path=["region_txt",'success'],  # Correct column names for
        hierarchy
    values="total",
    color='success',
    color_discrete_sequence=["rgb(245, 22, 22)", "rgb(46, 198,
        240)",  "rgb(235, 202, 96)"],
    hover_data=['total']
)
```

Sure! Here's a detailed explanation of the provided Python code using LaTeX:

latex Copy code

```
fig_success_by_region = px.treemap(
    data_frame=tot_att_df,
    path=["region_txt",'success'],
```

```
4       values="total",
5       color='success',
6       color_discrete_sequence=["rgb(245, 22, 22)", "rgb(46, 198,
            240)",  "rgb(235, 202, 96)"],
7       hover_data=['total']
8   )
```

* **Line 2:** Specify the source of the data for the treemap.
* **Line 3:** Defines the hierarchical structure of the treemap. The treemap will be grouped first by `region_txt` and then by `success`.
* **Line 4:** Sets the size of each rectangle in the treemap based on the values in the `total` column of the DataFrame.
* **Line 5:** Specifies the column `success` to determine the color of the rectangles in the treemap.
* **Line 6:** Provides a list of colors for the different categories of `success`
* **Line 7:** Adds hover data to the treemap.

Output:

Figure 4: *'Successful/Failed attack by region' Treemap chart*

- **Top 5 Rankings:**
  - **Top 5 Terrorist Groups by People Killed:** This chart ranks the most deadly terrorist groups based on the number of fatalities they have caused. It helps users identify the groups responsible for the highest casualties, providing a focused view on the most lethal actors in global terrorism.

```
fig_top_groups = px.bar(top_5_groups,
    x='civ_killed',
    y='te_group',
    title='Top 5 Terrorist Groups by Civilians Killed',
    labels={'te_group':'Terrorist Group', 'civ_killed':'Civilians
       Killed'},
    orientation='h',
)
```

- **Line 2:** Specify the data source for the bar chart.
- **Line 3, 4:** Set the values for the x-axis and the y-axis.
- **Line 5:** Set the title of the bar chart to `'Top 5 Terrorist Groups by Civilians Killed'`.
- **Line 6:** Provide custom labels for the axes.
- **Line 7:** Set the orientation of the bar chart to horizontal.

Change the layout and appearance of the ranked bar chart:

```
fig_top_groups.update_yaxes(visible=True, showticklabels=False)
fig_top_groups.update_layout(
    # margin=dict(l=0, r=0, t=0, b=0),
    xaxis=dict(showline=True, linecolor='white', linewidth=2),
    yaxis=dict(showline=True, linecolor='white', linewidth=2),
    paper_bgcolor='rgba(0,0,0,0)',
    plot_bgcolor='rgba(0,0,0,0)',
    font=dict(family='Arial',size=14, color='gray')
)
fig_top_groups.update_traces(
    marker_color='#FF0000'
)
```

- **Line 1:** Update the properties of the y-axis of the `fig_top_groups` bar chart. It makes the y-axis visible and hides the tick labels.
- **Line 3, 4:** Customize the x-axis and the y-axis properties.
- **Line 5:** Set the background color of the entire plot area to transparent.
- **Line 6:** Set the background color of the plotting area (excluding the axes and the legend) to transparent.
- **Line 7:** Specify the font properties for the text elements within the plot.
- **Line 9:** Set the color of the bars.

Adding icons corresponding to the terrorist groups:

```
c=-1
for x,y, png in zip(fig_top_groups.data[0].x,
   fig_top_groups.data[0].y, top_group_icons):
```

```
c+=1
fixed_size = (50, 50)  # Adjust these values as needed
img = Image.open(png)
img = img.resize(fixed_size, Image.BICUBIC)
fig_top_groups.add_layout_image(
    x=x,
    y=y,
    source=img,
    xref="x",
    yref="y",
    sizex=3000,
    sizey=3000,
    xanchor="center",
    yanchor="middle",
)
```

  - **Line 4:** Sets a fixed size for the images as a tuple `(50, 50)`.
  - **Line 6:** Resizes the opened image to the `fixed_size` dimensions using bicubic interpolation.
  - **Line 7-17:** Adds the resized image as a layout image to the `fig_top_groups` plot with specified properties.

– **Top 5 Weapons by Number of Attacks:** This chart shows the most commonly used weapons in terrorist attacks. By ranking the top 5 weapons, users can understand the preferred means of attack and assess the prevalence of different types of weapons in terrorist incidents. The code for this ranked bar chart is necessarily the same as the ranked bar chart of the top 5 terrorist groups by kills, mentioned above. This is the two ranked bar charts in our home page:

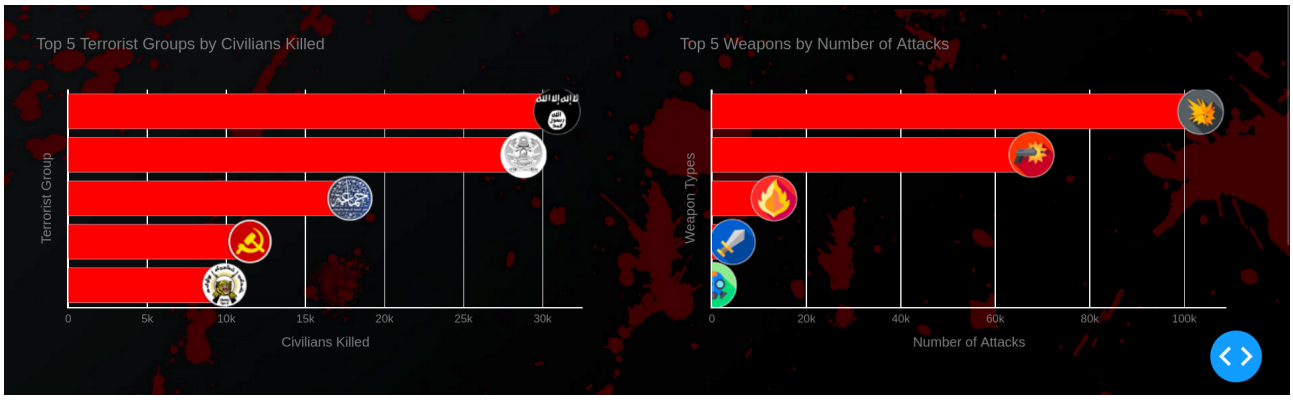

Figure 5: *'Top 5 Groups by Kills' and 'Top 5 Weapon types by Attacks' ranked bar charts*

During our project, we also add icons corresponding to terrorist groups and weapons in the two ranked bar charts above, which would improve visual recognition and reduce data interpretation time for users, at the same time make the dashboard more visually

appealing and engaging. Users can also hover over the bars to see specific information like the group's name along with the total number of people killed by that group.

Below is our layout for this page:

```
1  layout = html.Div([
2      html.Div(
3          [
4              html.Div('209.7K',className='index'),
5              html.Div('585.6K',className='index'),
6              html.Div('479.3K',className='index'),
7              html.Div('1064.9K',className='index'),
8              html.Div('Attacks',className='index'),
9              html.Div('Wounded',className='index'),
10             html.Div('Killed',className='index'),
11             html.Div('Casualties',className='index')
12         ],
13         style={'display': 'grid', 'gridTemplateColumns': '1fr 1fr 1fr
               1fr', 'gap': '10px'}
14     ),
15     html.Div(
16         [
17             dcc.Graph(figure=fig_target_type),
18             dcc.Graph(figure=fig_attack_type),
19             dcc.Graph(figure=fig_success_by_region)
20         ],
21         style={'display': 'grid', 'gridTemplateColumns': '1fr 1fr 1fr',
               'gap': '1px', 'margin-left': '50px','margin-right': '50px',}
22     ),
23     html.Div(
24         [
25             dcc.Graph(figure=fig_top_groups),
26             dcc.Graph(figure=fig_top_weapons)
27         ],
28         style={'display': 'grid', 'gridTemplateColumns': '1fr 1fr',
               'gap': '1px'}
29     )
30 ])
```

- **Lines 2-14:** Creates a nested Div containing numerical values and labels representing various statistics such as Attacks, Wounded, Killed, and Casualties.
- **Lines 15-22:** Defines another nested Div containing three graphs: `fig_target_type`, `fig_attack_type`, and `fig_success_by_region`).
- **Lines 23-29:** Creates another nested Div containing two graphs: (`fig_top_groups` and `fig_top_weapons`).

This is the resulting page:

Figure 6: *Visualizations for the home page*

These visualizations on the Home Page are designed to provide users with a holistic view of global terrorism, combining summary metrics with detailed breakdowns of key aspects. This approach ensures that users can quickly access both high-level trends and specific insights into the nature and impact of terrorist activities.

3.2.3 Geography Dashboard

The Geography Dashboard is dedicated to visualizing geographic patterns and trends in global terrorism. This page features interactive maps and charts that allow users to explore terrorist incidents based on their geographical location. Users can drill down into specific regions or countries to gain insights into the spatial distribution of terrorist activity.

The page features three visualizations: a choropleth map, a sunburst chart, and a horizontal stacked bar chart.

- Choropleth map

  The purpose of the first choropleth map is to analyze and display trends in global casualties over time.

  First, we need to handle missing data - missing data will make some of the countries not visible in specific years. The preprocessing steps ensure that all combinations of years and countries are represented, with missing values filled with zeros.

  – Aggregating the data

```
kill_data = data[['year', 'country_code', 'country_txt',
    'total_casualties']]
kill_data = kill_data.groupby(['year', 'country_code',
    'country_txt'])['total_casualties'].sum().reset_index()
kill_data = kill_data.rename(columns={'country_txt': 'Country'})
```

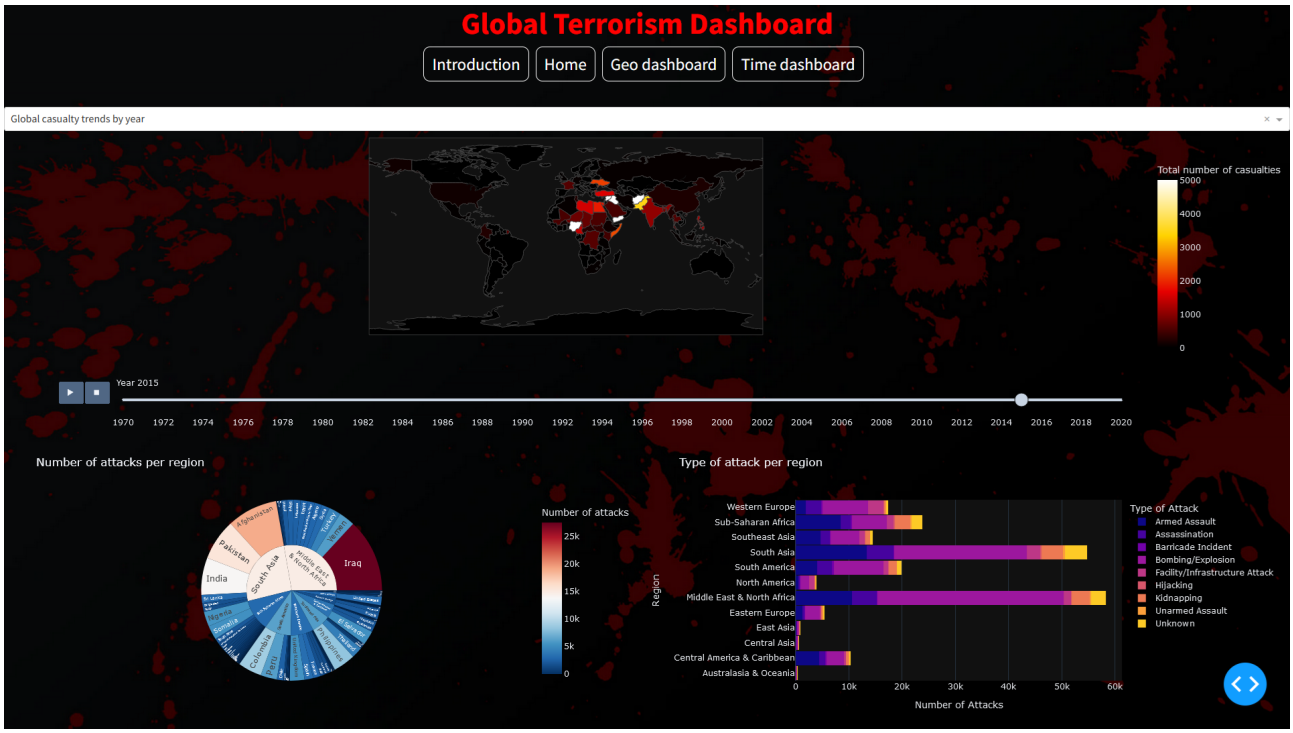

Figure 7: *The main screen of Geography dashboard*

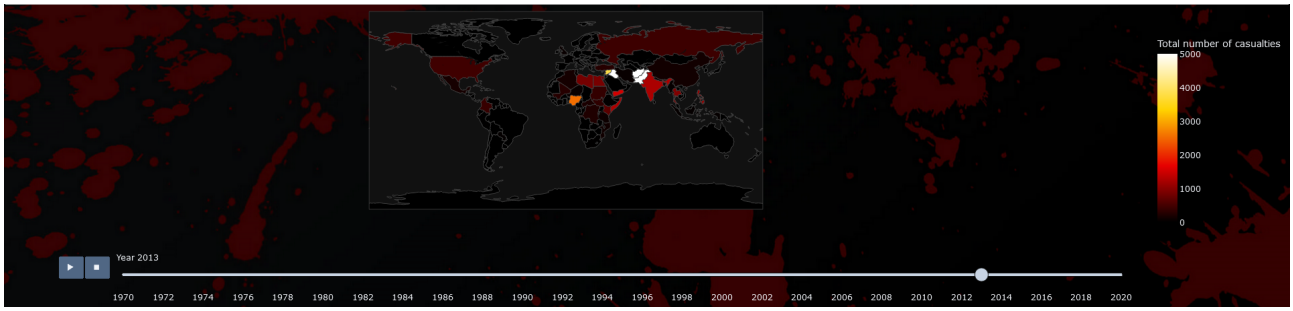

Figure 8: *Choropleth map - Global casualty trends by year*

– Fill in missing countries for every year

```
all_countries = kill_data['country_code'].unique()
all_years = all_years = np.arange(1970, 2021)
all_combinations = pd.MultiIndex.from_product([all_years,
    all_countries], names=['year',
    'country_code']).to_frame(index=False)
complete_data = pd.merge(all_combinations, kill_data, on=['year',
    'country_code'], how='left')
complete_data['total_casualties'] =
    complete_data['total_casualties'].fillna(0)
country_names = kill_data[['country_code',
    'Country']].drop_duplicates()
complete_data = pd.merge(complete_data, country_names,
    on='country_code', how='left')
complete_data = complete_data.sort_values(['year',
    'country_code']).reset_index(drop=True)
complete_data.drop_duplicates(inplace=True)
```

– Make the column name make more sense

```
complete_data.drop(['Country_x'], axis=1,  inplace=True)
complete_data.rename(columns={'Country_y': 'Country'}, inplace=True)
complete_data.sort_values(['year', 'total_casualties'],
    inplace=True)
complete_data.reset_index(drop=True, inplace=True)
```

The choropleth map color-codes countries based on the number of casualties, making it easy to identify geographic regions with high or low casualty rates.

```
fig1 = px.choropleth(complete_data,
    locations='country_code',
    color='total_casualties',
    hover_data={'Country': True, 'total_casualties': True,
        'country_code': False, 'year': False},
    range_color=(0, 5000),
    animation_frame='year',
    color_continuous_scale='hot',
    labels={'total_casualties':'Total number of casualties'},
    template='plotly_dark'
)
```

The map includes an animation that transitions through each year from 1970 to 2020. This allows users to see how the distribution of casualties has changed over time, highlighting trends and patterns in different periods.

```
sliders1=[{
    "active": 0,
    "currentvalue": {"prefix": "Year "},
    "steps": fig1.layout.sliders[0].steps,
}]
```

Layout adjustments are made for visual aesthetics, including background color, margins, and color bar positioning. Geographical properties are also updated to display the land in black.

```
fig1.update_layout(
    plot_bgcolor='black',
    margin={"r":0,"t":10,"l":0,"b":0},
    sliders=sliders1,
    coloraxis_colorbar=dict(
        yanchor="top",
        y=0.9,
    ),
    paper_bgcolor='rgba(0,0,0,0)',
)
fig1.update_geos(
    showland=True, landcolor="black",
)
```

The animation speed and transition duration are set to be quicker than the default speed.

```
fig1.layout.updatemenus[0].buttons[0].args[1]['frame']['duration'] = 120
fig1.layout.updatemenus[0].buttons[0].args[1]['transition']['duration']
    = 30
```

The second choropleth is used to visualize the total number of casualties across different countries, categorized into various bins for better interpretability.

First, we also need to prepare the data.

- Extract relevant columns and aggregating

```
all_kill_data = data[['year', 'country_code', 'country_txt',
    'total_casualties']]
all_kill_data = all_kill_data.groupby(['country_code',
    'country_txt'])['total_casualties'].sum().reset_index()
```

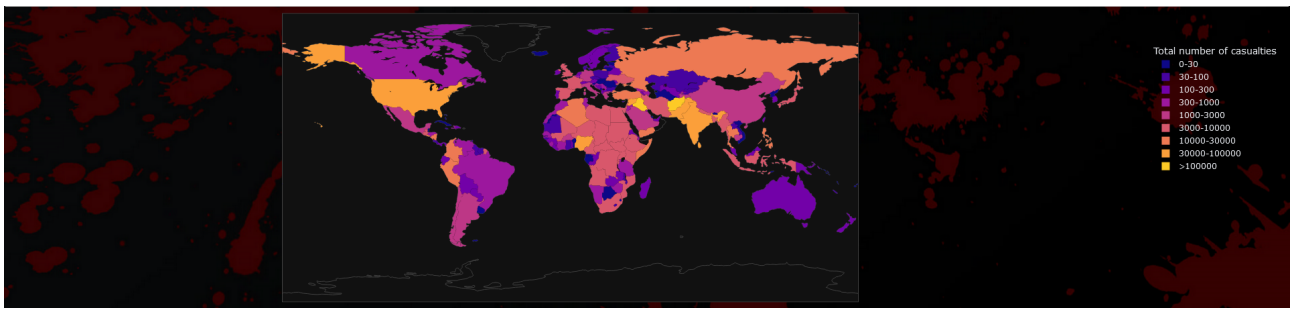

Figure 9: *Choropleth map - Total number of casualties*

– Define bins and labels for casualties; categorize and sort

```
log_bin_edges = [0, 30, 100, 300, 1000, 3000, 10000, 30000, 100000,
    300000000]
log_bin_labels = ['0-30', '30-100', '100-300', '300-1000',
    '1000-3000', '3000-10000', '10000-30000', '30000-100000',
    '>100000']
all_kill_data['total_casualties_cat'] =
    pd.cut(all_kill_data['total_casualties'], bins=log_bin_edges,
    labels=log_bin_labels, include_lowest=True)
all_kill_data.sort_values(['total_casualties'], inplace=True)
all_kill_data.reset_index(drop=True, inplace=True)
all_kill_data = all_kill_data.rename(columns={'country_txt':
    'Country'})
```

Next, we will create the choropleth map and changing the style.

– Create the map

```
fig2 = px.choropleth(all_kill_data[['country_code',
    'total_casualties_cat', 'Country']],
     locations='country_code',
     hover_data={'Country': True, 'country_code': False,
         'total_casualties_cat': True},
     color='total_casualties_cat',
     color_discrete_sequence=['#0d0887', '#46039f', '#7201a8',
         '#9c179e', '#bd3786', '#d8576b', '#ed7953', '#fb9f3a',
         '#fdca26', '#f0f921'],
     labels={'total_casualties_cat':'Total number of casualties'},
     template='plotly_dark',
)
```

– Updates the layout of the map to set the background colors, margins, and legend position

```
fig2.update_layout(
     plot_bgcolor='black',
```

```
    margin={"r":10,"t":10,"l":10,"b":10},
    legend=dict(
        yanchor="top",
        y=0.9,
    ),
    paper_bgcolor="rgba(0, 0, 0, 0)",
)
```

– Customizes the trace properties and the hover template to display the country name and the total number of casualties

```
fig2.update_traces(
    marker_line_width=0,
    hovertemplate='%{customdata[0]}<br>Total number of casualties:
       %{customdata[2]}<extra></extra>'
)
```

The user can switch between two maps using the dropdown menu, which also serves as the title of the current map:

Figure 10: *Dropdown menu*

- Sunburst chart

The primary purpose of our sunburst chart is to provide a detailed and hierarchical view of the number of attacks per region and country. The sunburst chart effectively visualizes how attacks are distributed geographically.

– Data preparation:

```
data_num_atk = data[['region_txt', 'country_txt', 'country_code']]
data_num_atk.loc[data_num_atk['country_txt'] == 'Germany',
   'region_txt'] = 'Western Europe' # damn
data_num_atk.loc[data_num_atk['region_txt'] == 'Middle East & North
   Africa', 'region_txt'] = 'Middle East<br>& North Africa'
data_num_atk.loc[data_num_atk['region_txt'] == 'Central America &
   Caribbean', 'region_txt'] = 'Central America<br>& Caribbean'
data_num_atk = data_num_atk.groupby(['country_txt',
   'region_txt']).size().reset_index(name='Number of attacks')
```

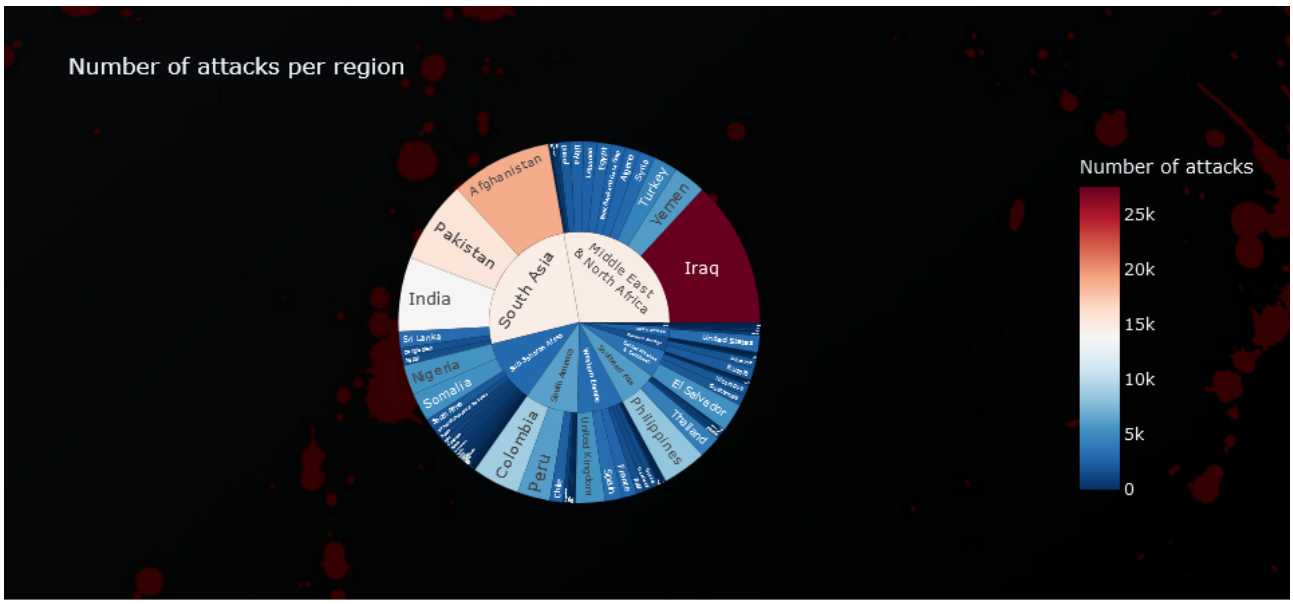

Figure 11: *Sunburst chart showing number of attacks per region and country. The color in the middle shows the mean number of attacks of each country for each region.*

– Making the chart:

```
fig3 = px.sunburst(data_num_atk,
    path=['region_txt', 'country_txt'], values='Number of attacks',
    color='Number of attacks',
    hover_data={'Number of attacks': True},
    color_continuous_scale='RdBu_r',
    title='Number of attacks per region',
    template='plotly_dark'
)
fig3.update_layout(
     paper_bgcolor='rgba(0,0,0,0)',
)
fig3.update_traces(
     hovertemplate='%{label}<br>Number of attacks:
        %{value}<extra></extra>'
)
```

The users can click on a region to filter out that specific region.

- Stacked bar chart

  The stacked bar chart provide a detailed view of the types of attacks in different regions. The chart effectively visualizes the distribution and composition of attack types within each region, allowing users to quickly grasp the variety and frequency of attack types in different regions, and to be able to easily compare regions to see which types of attacks are more prevalent in each region.

  – Data preparation:

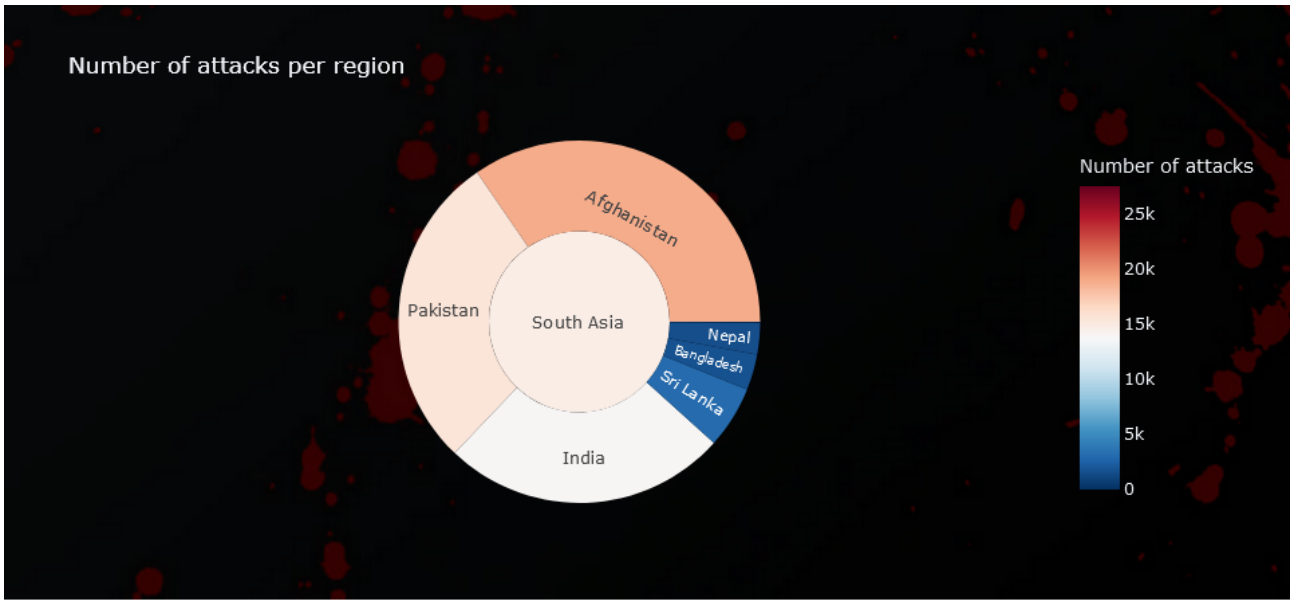

Figure 12: *Number of attacks for countries in South Asia.*

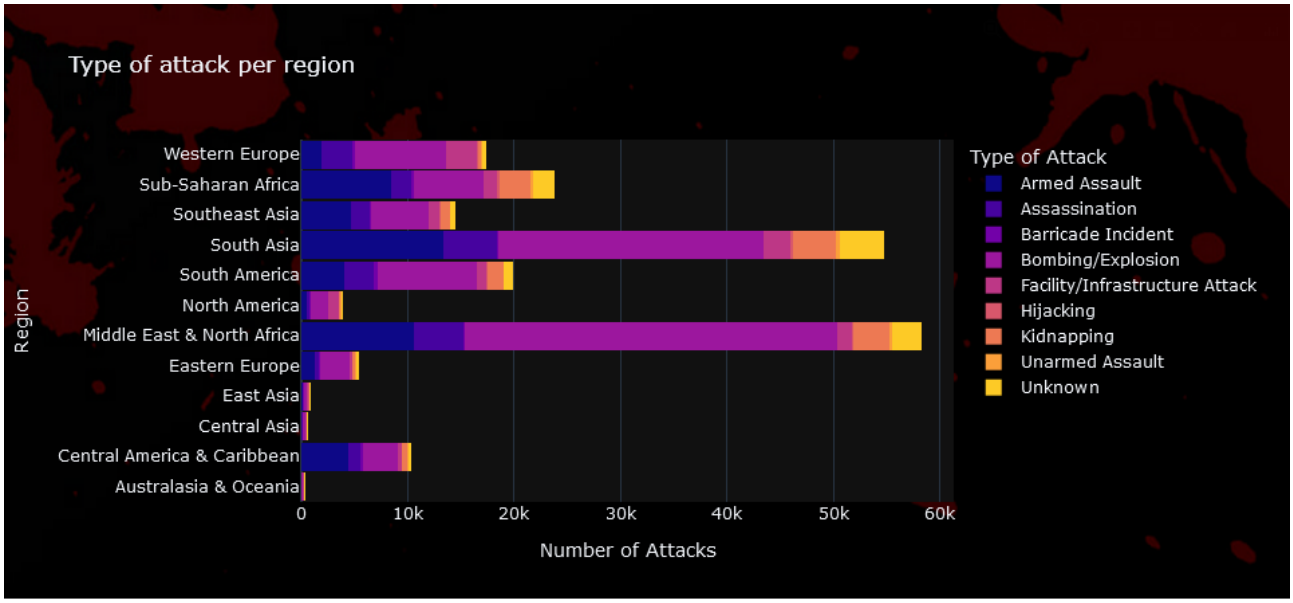

Figure 13: *Stacked bar chart*

```
count_atk_type_data = data[['region_txt', 'attack_type']]
count_atk_type_data = count_atk_type_data.groupby(['region_txt',
    'attack_type']).size().reset_index(name='count')
```

– Making the chart:

```
fig4 = px.bar(
    count_atk_type_data,
    x='count',
    y='region_txt',
    color='attack_type',
    color_discrete_sequence=px.colors.sequential.Plasma,
    template='plotly_dark',
    labels={'region_txt':'Region', 'attack_type':'Type of Attack',
        'count':'Number of Attacks'},
    orientation='h',
    title='Type of attack per region'
)
fig4.update_layout(
    bargap=0,
    bargroupgap=0.05,
    barmode="stack",
    paper_bgcolor='rgba(0,0,0,0)',
)
fig4.update_traces(marker_line_width=0)
```

To switch between chorepleth maps, we use a specific function with a Dash callback function wrapper. The callback listens for changes to the dropdown menu. When a user selects an option from the dropdown, it triggers the `update_figure` function, which then updates the figure displayed in the selected-figure component based on the selected value.

```
@callback(
    Output(component_id="selected-figure", component_property="figure"),
    Input(component_id="figure-dropdown", component_property="value")
)
def update_figure(selected_value):
    if selected_value == "fig1":
        return fig1
    else:
        return fig2
```

All of the charts and maps are then displayed using the Dash framework. It includes a dropdown menu to select between different choropleth maps and displays the selected map, along with two other figures side by side.

```
layout = html.Div(children=[
    html.Div(children=[
        # html.Br(),
        dcc.Dropdown(
            id="figure-dropdown",
            options=[
                {"label": "Global casualty trends by year", "value":
                    "fig1"},
                {"label": "Total number of casualties", "value": "fig2"},
            ],
            value="fig1"
        ),
        # dcc.Graph(id="selected-figure")
        dcc.Graph(id="selected-figure")
    ]),
    html.Div(children=[
        dcc.Graph(id='fig3', figure=fig3, style={'width': '50%', 'display':
            'inline-block'}),
        dcc.Graph(id='fig4', figure=fig4, style={'width': '50%', 'display':
            'inline-block'})
    ])
])
```

3.2.4 Time Dashboard

The Time Dashboard focuses on analyzing temporal patterns and trends in global terrorism over time. This page includes time series plots and other visualizations that illustrate how the frequency and severity of terrorist incidents have evolved over different time periods.

By dividing the dashboard into separate pages, we aim to provide users with a structured and intuitive navigation experience, allowing them to explore specific aspects of the global terrorism dataset in detail.

`df_pivot` is a Dataframe with years as rows and regions as column. It contains 4 categories corresponding to 4 metrics: `total_attacks, total_killed, total_wounded` and `property_damage`.

Stacked Area Plots

— **Number of Terrorist Attacks per Year by Region**:
This stacked area plot visualizes the annual count of terrorist attacks across different regions. By examining the changes over time, users can identify trends and patterns in regional terrorist activity, highlighting which areas experience the most attacks and how these patterns evolve.

Step-by-step Description

Create a Plotly figure titled 'Number of Terrorist Attacks per Year by Region'. It takes

the 'total_attacks' segment of df_pivot as the DataFrame, with x-axis representing the number of attacks and y-axis representing years. Assign a distinct color for each of the region area using color_map. Hide color legend.

```
fig_attacks = px.area(df_pivot['total_attacks'], x=df_pivot.index,
    y=df_pivot['total_attacks'].columns,
     title='Number of Terrorist Attacks per Year by Region',
        color_discrete_map=color_map)
fig_attacks.update_traces(showlegend=False)
```

Output:

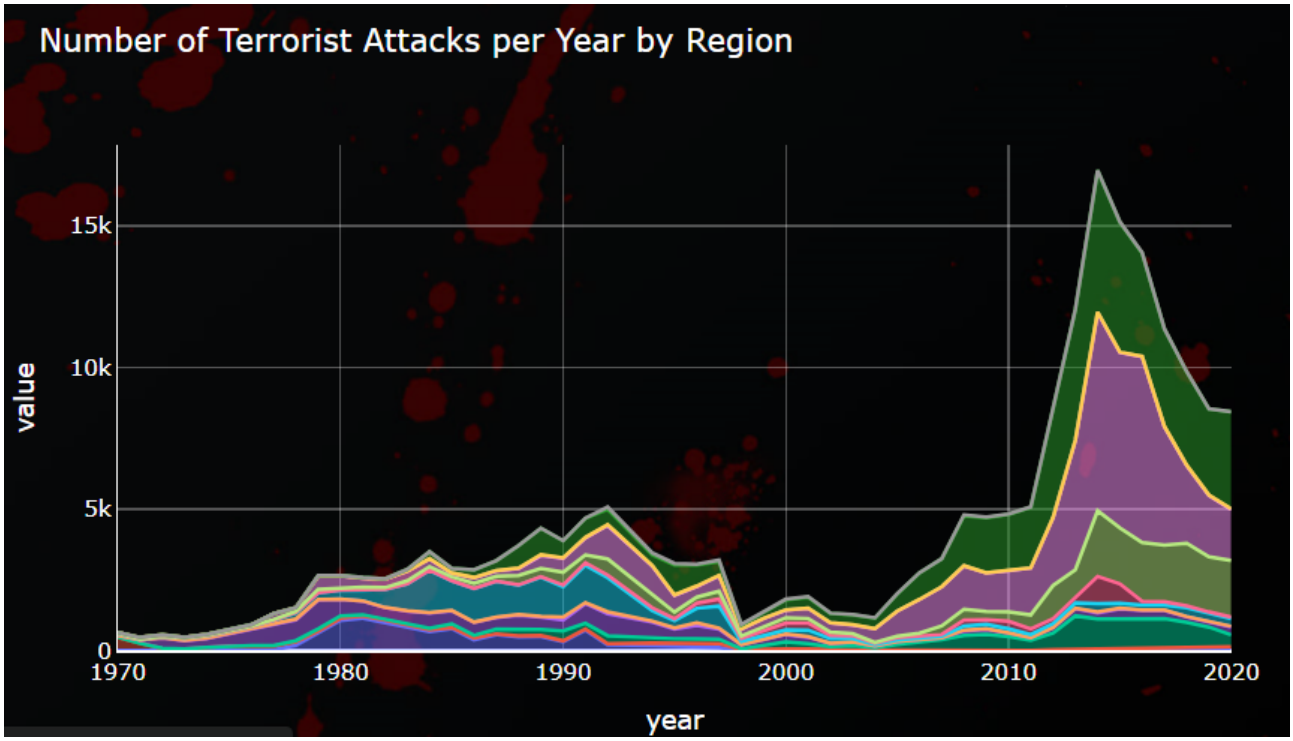

Figure 14: *'Number of Terrorist Attacks per Year by Region' stacked area graph*

— **Number of Fatalities per Year by Region**:

This stacked area plot depicts the number of fatalities resulting from terrorist attacks in various regions over time. It provides insights into the human cost of terrorism, showing which regions suffer the most in terms of loss of life and how these numbers fluctuate year by year.

Step-by-step Description

Create a Plotly figure titled 'Number of Fatalities per Year by Region'. It takes the 'total_killed' segment of df_pivot as the DataFrame, with x-axis representing the number of fatalities and y-axis representing years. Assign a distinct color for each of the region area using color_map. Hide color legend.

```
fig_fatalities = px.area(df_pivot['total_killed'], x=df_pivot.index,
    y=df_pivot['total_killed'].columns,
```

```
    title='Number of Fatalities per Year by Region',
        color_discrete_map=color_map)
fig_fatalities.update_traces(showlegend=False)
```

Output:

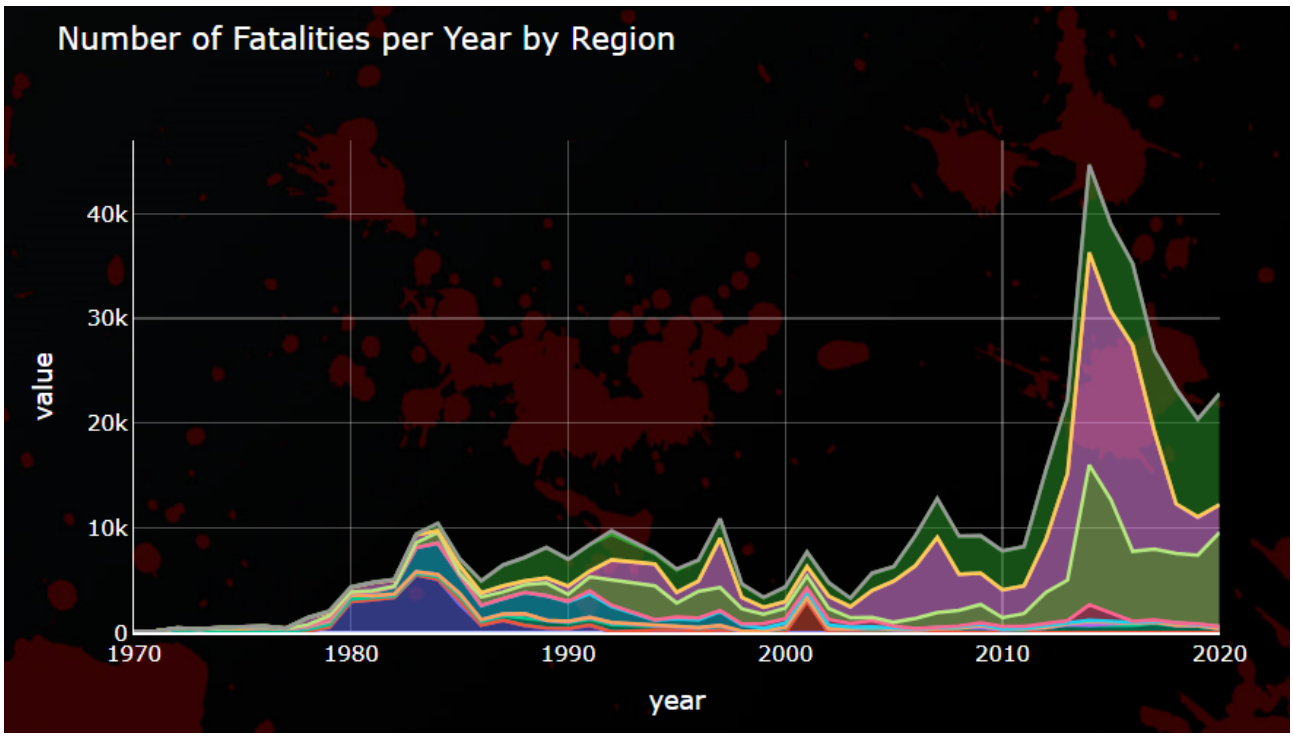

Figure 15: *'Number of Fatalities per Year by Region' stacked area graph*

— **Number of Injuries per Year by Region**:
This stacked area plot shows the annual number of injuries caused by terrorist attacks across different regions. It helps users understand the physical impact of terrorism on populations in various areas, revealing which regions see the highest number of injured individuals and how these figures change over the years.

Step-by-step Description

Create a Plotly figure titled 'Number of Injuries per Year by Region'. It takes the `'total_wounded'` segment of `df_pivot` as the DataFrame, with x-axis representing the number of injuries and y-axis representing years. Assign a distinct color for each of the region area using `color_map`. Hide color legend.

```
fig_injuries = px.area(df_pivot['total_wounded'], x=df_pivot.index,
    y=df_pivot['total_wounded'].columns,
     title='Number of Injuries per Year by Region',
        color_discrete_map=color_map)
fig_injuries.update_traces(showlegend=False)
```

Output:

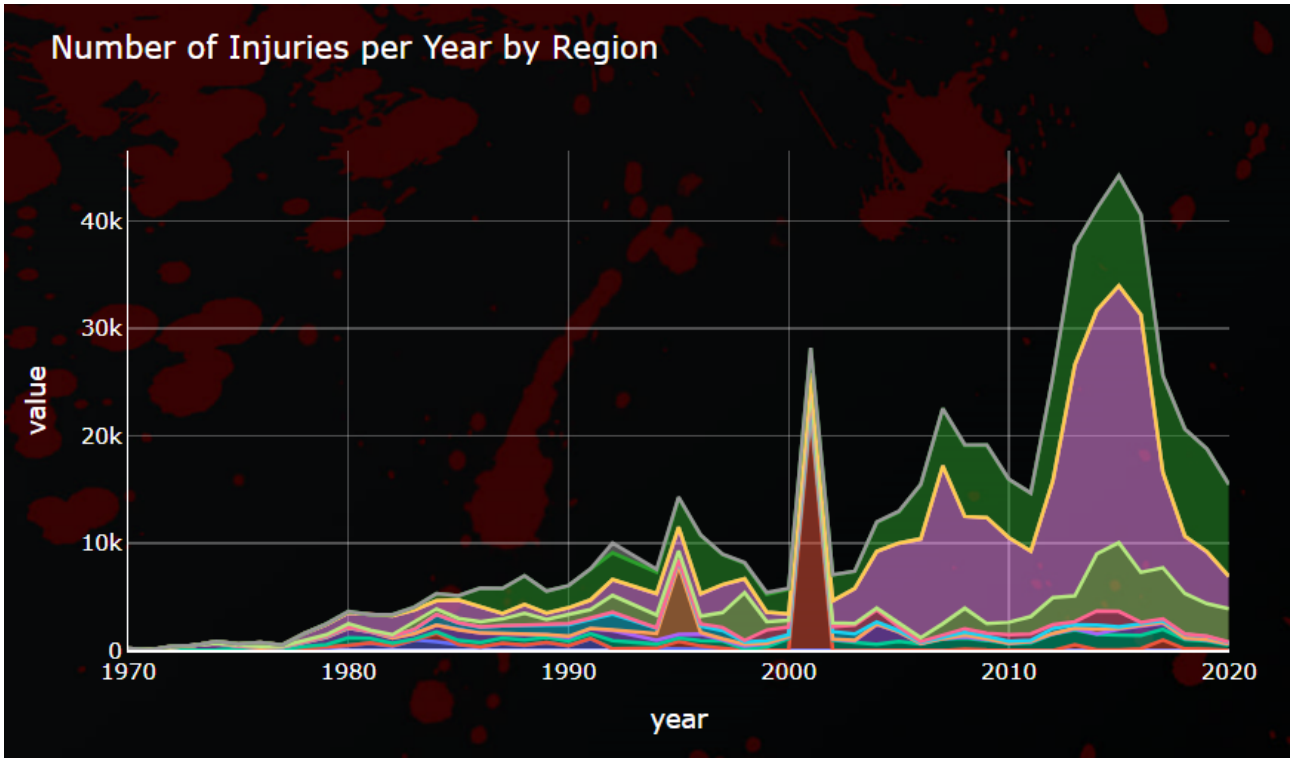

Figure 16: *'Number of Injuries per Year by Region' stacked area graph*

— **Property Damage in USD per Year by Region**:
This stacked area plot illustrates the financial impact of terrorism by displaying the estimated property damage (in USD) caused by terrorist attacks in different regions over time. Users can assess the economic consequences of terrorism, identifying regions that experience significant property losses and observing trends in financial damage year over year.

Step-by-step Description

Create a Plotly figure titled 'Property Damage in USD per Year by Region'. It takes the `'property_damage'` segment of `df_pivot` as the DataFrame, with x-axis representing the value of property damage and y-axis representing years. Assign a distinct color for each of the region area using `color_map`. Hide color legend.

```
fig_damage = px.area(df_pivot['property_damage'], x=df_pivot.index,
    y=df_pivot['property_damage'].columns,
     title='Property Damage in USD per Year by Region',
        color_discrete_map=color_map)
fig_damage.update_traces(showlegend=False)
```

Output:

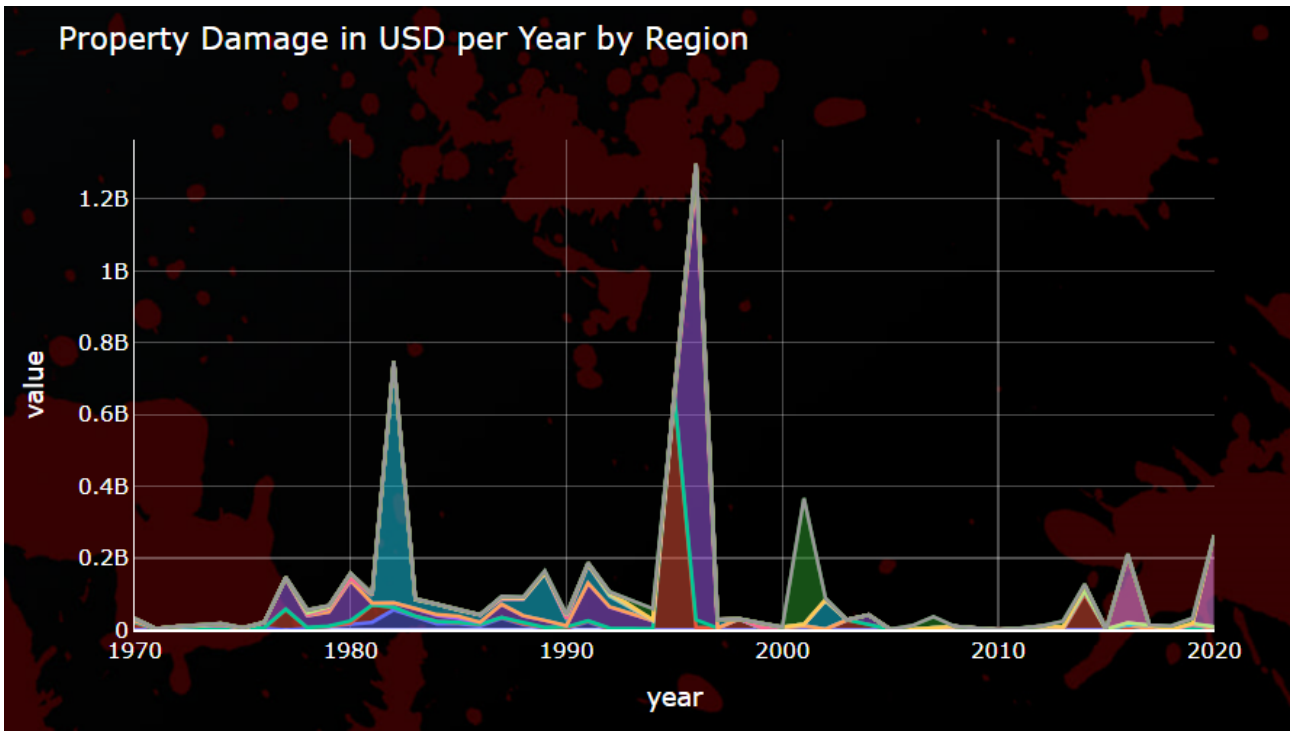

Figure 17: *'Property Damage in USD per Year by Region' stacked area graph*

To get the outputs above, we customize the layout of the figures using the following code:

```
1  for fig in [fig_attacks, fig_fatalities, fig_injuries, fig_damage]:
2      fig.update_layout(xaxis=dict(showline=True, linecolor='white',
           gridcolor='rgba(255, 255, 255, 0.3)'),
3          yaxis=dict(showline=True, linecolor='white', gridcolor='rgba(255,
               255, 255, 0.3)'),
4          paper_bgcolor='rgba(0,0,0,0)',
5          plot_bgcolor='rgba(0,0,0,0)',
6          font=dict(color='white'),
7          title=dict(font=dict(color='white')),
8          title_y=0.95,
9          xaxis_title=dict(font=dict(color='white')),
10         yaxis_title=dict(font=dict(color='white')))
```

* **Line 1:** Apply the customization to all 4 figures
* **Line 2, 3:** Specify the properties of the x-axis and the y-axis
* **Line 4, 5:** Set the background color of the entire plot area.
* **Line 6, 7:** Set the color of the text elements (like axis labels, title).
* **Line 8:** Set y-coordinate of the title.
* **Line 9, 10:** Specify the color of the x-axis and the y-axis label text.

Text Annotations

This visualization highlights and provides some contextual insight into major attacks that significantly impacted the number of terrorist attacks, fatalities, injuries, and property damage. Define the style and appearance of annotations for Plotly figures:

```
1  annotation_arg = {
2      'font': {'color': 'white', 'size': 14},
3      'bgcolor': 'rgba(255, 255, 255, 0)',
4      'showarrow': True,
5      'arrowhead': 1,
6      'axref': 'pixel',
7      'ayref': 'pixel',
8      'arrowwidth': 2,
9      'arrowcolor': 'white',
10     'ax': 0,
11     'ay': -150,
12     'borderpad': 3
13 }
```

* **Line 2**: Specify the font style of the annotation text.
* **Line 3**: Specify the background color of the annotation text box.
* **Line 4-11**: Show an arrow pointing to the annotation location. Specify the style of the arrowhead. Set the reference for the arrow's x-coordinate and y-coordinate to be in pixels (`'pixel'`). Sets the width and color of the arrow. Specify the x-coordinate and y-coordinate for the arrow's starting point.
* **Line 12**: Specify the padding around the text in the annotation text box.

Output:

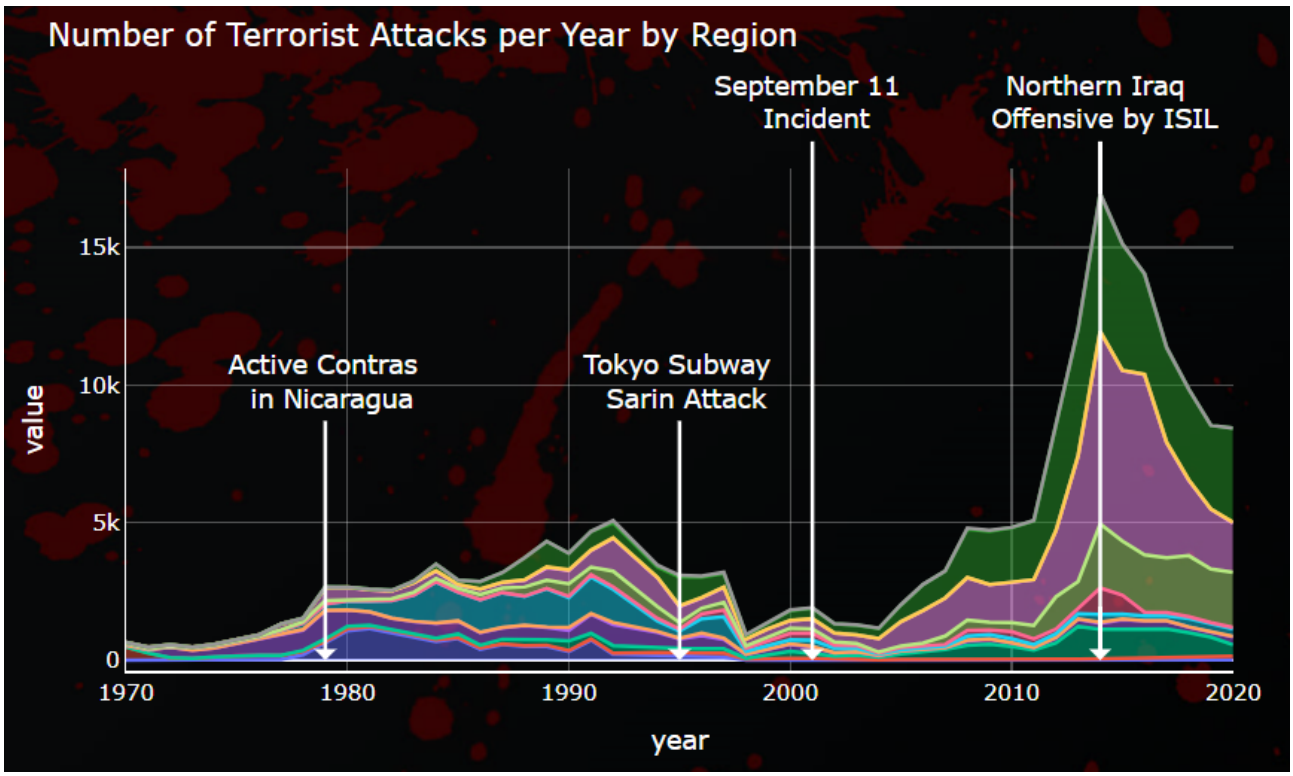

Figure 18: *'Number of Terrorist Attacks per Year by Region' stacked area graph with text annotations*

Define the layout of the Dash app, including the input controls and graphs.

```
layout = html.Div([
    html.Div([
        html.Div([
            html.Label('Select Status:', style={'font-size':'18px',
                'color': 'white'}),
            dcc.RadioItems(id='success-radioitems', options=[
                {'label': '(All)', 'value': '(All)'}, {'label':
                    'Successful', 'value': 'Successful'},
                {'label': 'Unsuccessful', 'value': 'Unsuccessful'}
            ], value='(All)', labelStyle={'color':'white'},
                style={'display': 'grid', 'gridTemplateColumns': '1fr',
                'gap': '10px'})
        ], style={'border': '1px solid white', 'border-radius': '10px',
            'padding': '10px', 'padding-left':'25px',
            'margin-right':'10px', 'background-color':'rgba(0, 0, 0, 1)'}),
        html.Div([
            html.Label('Filter by:', style={'font-size':'18px', 'color':
                'white'}),
            dcc.Checklist(id='region-checklist', options=[
                {'label': [html.Span('    ', style={'backgroundColor':
                    colors[i], 'display': 'inline-block', 'height': '10px',
                    'width': '30px', 'marginRight': '5px', 'marginLeft':
                        '5px'}), html.Span(regions[i])], 'value': regions[i]}
                for i in range(len(regions))], value=regions,
                    labelStyle={'display': 'inline-block', 'margin-right':
```

```
                    '10px', 'color':'white'},
17                 style={'display': 'grid', 'gridTemplateColumns': '1fr 1fr
                       1fr 1fr', 'gap': '10px'})
18         ], style={'border': '1px solid white', 'border-radius': '10px',
               'padding': '10px','padding-left':'25px',
19              'background-color':'rgba(0, 0, 0, 1)'})
20     ], style={'display': 'grid', 'gridTemplateColumns':'300px
           auto','margin-right': '10px'}),
21     html.Div([
22         dcc.Graph(id='graph-attacks', figure=fig_attacks),
23         dcc.Graph(id='graph-fatalities', figure=fig_fatalities),
24         dcc.Graph(id='graph-injuries', figure=fig_injuries),
25         dcc.Graph(id='graph-damage', figure=fig_damage),
26     ], style={'display': 'grid', 'gridTemplateColumns': '1fr 1fr', 'gap':
           '10px', 'backgroundColor': 'rgba(0,0,0,0)'})
27 ], className='content')
```

Output:

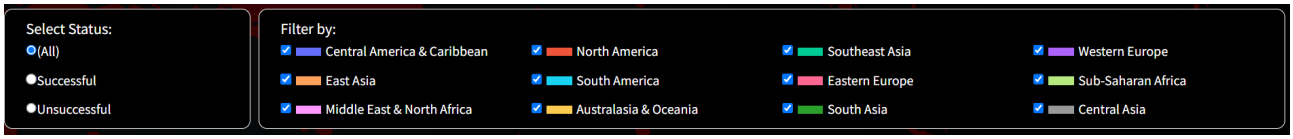

Figure 19: *Status selection section and region filtering section*

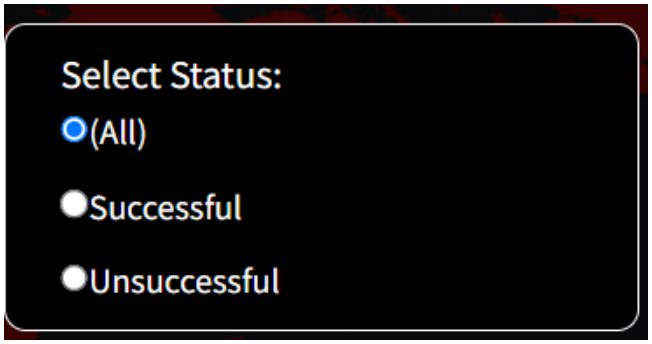

Figure 20: *Status selection section*

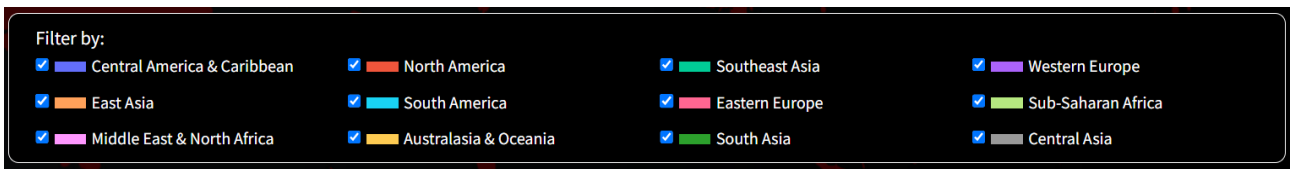

Figure 21: *Region filtering section*

* **Line 2-27:** Nested HTML structure consisting of two main divisions: one for the filter options and another for the graphs.

* **Line 3-20:** Inside the first division, there are two sub-divisions for the status selection and region filtering.

* **Line 3-10:** Status selection section with radio items for choosing between all, successful, and unsuccessful.

* **Line 12-17:** Region filtering section with checkboxes to select regions.

* **Line 21-26:** Inside the second division, there are four sub-divisions for displaying the graphs. Graphs are displayed using Dash components with unique IDs to use in callbacks and associated figures.

Define a callback function to update the graphs based on the selected regions and status.

```
@callback(
    Output('graph-attacks', 'figure'),
    Output('graph-fatalities', 'figure'),
    Output('graph-injuries', 'figure'),
    Output('graph-damage', 'figure'),
    Input('region-checklist', 'value'),
    Input('success-radioitems', 'value')
)
def update_graphs(selected_regions, selected_status):
    if selected_status == 'Successful':
        filtered_df = df[df['success'] == 1]
    elif selected_status == 'Unsuccessful':
        filtered_df = df[df['success'] == 0]
    else:
        filtered_df = df

    filtered_grouped_attacks = filtered_df.groupby([year_column,
       region_column]).size().reset_index(name='total_attacks')
    filtered_grouped = filtered_df.groupby([year_column,
       region_column]).agg({'total_killed': 'sum',
                            'total_wounded': 'sum',
                            'prorperty_damage':'sum'}).reset_index()
    filtered_merged_attacks = filtered_grouped_attacks.merge(all_years,
       on=year_column, how='right').fillna({region_column: 'Unknown',
       'total_attacks': 0})
    filtered_merged = filtered_grouped.merge(all_years, on=year_column,
       how='right').fillna({region_column: 'Unknown',
    'total_killed': 0,
    'total_wounded': 0,
    'prorperty_damage': 0})
    filtered_merged_df = pd.merge(filtered_merged_attacks, filtered_merged,
       on=[year_column, region_column],
                                  how='outer')
    filtered_merged_df =
       filtered_merged_df[filtered_merged_df[region_column] != 'Unknown']
    filtered_pivot = filtered_merged_df.pivot(index=year_column,
       columns=region_column,
```

```
    values=['total_attacks', 'total_killed', 'total_wounded',
    'prorperty_damage']).fillna(0)

    filtered_attacks = filtered_pivot['total_attacks'][selected_regions]
    filtered_fatalities = filtered_pivot['total_killed'][selected_regions]
    filtered_injuries = filtered_pivot['total_wounded'][selected_regions]
    filtered_damage = filtered_pivot['prorperty_damage'][selected_regions]

    fig_attacks = px.area(filtered_attacks,
                          x=filtered_attacks.index,
                          y=filtered_attacks.columns,
                          title='Number of Terrorist Attacks per Year by
                              Region',
                          color_discrete_map=color_map)

    fig_fatalities = px.area(filtered_fatalities,
                             x=filtered_fatalities.index,
                             y=filtered_fatalities.columns,
                             title='Number of Fatalities per Year by Region',
                             color_discrete_map=color_map)


    fig_injuries = px.area(filtered_injuries,
                             x=filtered_injuries.index,
                             y=filtered_injuries.columns,
                             title='Number of Injuries per Year by Region',
                             color_discrete_map=color_map)


    fig_damage = px.area(filtered_damage,
                          x=filtered_damage.index,
                          y=filtered_damage.columns,
                          title='Property Damage in USD per Year by Region',
                          color_discrete_map=color_map)

    fig_attacks.update_traces(showlegend=False)
    fig_fatalities.update_traces(showlegend=False)
    fig_injuries.update_traces(showlegend=False)
    fig_damage.update_traces(showlegend=False)

    for fig in [fig_attacks, fig_fatalities, fig_injuries, fig_damage]:
        fig.update_layout(
            xaxis=dict(showline=True, linecolor='white',
                gridcolor='rgba(255, 255, 255, 0.3)'),
            yaxis=dict(showline=True, linecolor='white',
                gridcolor='rgba(255, 255, 255, 0.3)'),
            paper_bgcolor='rgba(0,0,0,0)',
            plot_bgcolor='rgba(0,0,0,0)',
            font=dict(color='white'),
            title=dict(font=dict(color='white')),
```

```
77              xaxis_title=dict(font=dict(color='white')),
78              yaxis_title=dict(font=dict(color='white'))
79          )
80          if 'North America' in selected_regions and selected_status in
                ['Successful', '(All)']:
81              fig.add_annotation(
82                  arg=annotation_arg,
83                  x=2001,
84                  y=0,
85                  text="September 11 <br> Incident",
86                  ay=-300
87              )
88          if 'Middle East & North Africa' in selected_regions and
                selected_status in ['Successful', '(All)']:
89              fig.add_annotation(
90                  arg=annotation_arg,
91                  x=2014,
92                  y=0,
93                  text="Northern Iraq <br> Offensive by ISIL",
94                  ay=-300
95              )
96          if 'East Asia' in selected_regions and selected_status in
                ['Successful', '(All)']:
97              fig.add_annotation(
98                  arg=annotation_arg,
99                  x=1995,
100                 y=0,
101                 text="Tokyo Subway <br> Sarin Attack",
102             )
103         if 'Central America & Caribbean' in selected_regions and
                selected_status in ['Successful', '(All)']:
104             fig.add_annotation(
105                 arg=annotation_arg,
106                 x=1979,
107                 y=0,
108                 text="Active Contras <br> in Nicaragua",
109             )
110     return fig_attacks, fig_fatalities, fig_injuries, fig_damage
```

* **Line 1-9:** Define a callback function named `update_graphs` with outputs specified as the four figures and values selected in the 'region-checklist' and 'success-radioitems' components.

* **Line 10-15:** Based on the selected success status, filter the DataFrame (`df`) to include only successful attacks, unsuccessful attacks, or all attacks.

* **Line 17-31:** Preprocess the filtered data.

* **Line 33-36:** Extract the filtered data for attacks, fatalities, injuries, and property damage based on the selected regions.

* **Line 59-79:** Generate Plotly figures for each aspect (attacks, fatalities, injuries, and damage) using the filtered data. Keep the original customization.

* **Line 80-109:** Add annotations to figures according to the selected regions and status. Specify the style, coordinate and text of the annotations.

4 Conclusion

This comprehensive analysis of global terrorism, utilizing a rich dataset from the Global Terrorism Database (GTD), has provided a valuable insight into the complexity and diversity of the global threats of terrorism. Through the application of various data visualization techniques, we have gained valuable insights into the characteristics, geographical patterns, and temporal trends of terrorist attacks. In this report, we focus on several types of chart and graph:

- Basic *Donut charts* reveal the prevalence of specific target types and the predominance of certain attack methods.
- *Ranked Bar charts* and *Stacked Bar chart* identify the terrorist groups responsible for the highest death tolls, the most commonly used weapons in attacks, and the relation between the type of attack and region.
- *Treemap chart* shows how many attacks by region are successful or failed.
- *Sunburst chart* to get further details of the number of attacks in each country.
- *Choropleth map* shows the total number of casualties for each country, filtered by year.
- *Stacked Area charts* indicate the number of terrorist Attack, Fatalities, Injuries, Property damage in USD per year by region.

Prefer the Warm colors (red, orange, yellow, .etc) which are typically associated with things like emotions, danger, warning, and urgency. These connections cause them to the perfectly speaking tools for the type of high stakes and severe nature global terrorism involves. The visualizations above have significant implications. Understanding the characteristics, geographical distribution, and temporal trends of terrorist attacks is crucial for developing effective strategies to prevent, mitigate, and respond to these threats.

References

[1] START (National Consortium for the Study of Terrorism and Responses to Terrorism). (2022). Global Terrorism Database 1970 - 2020. https://www.start.umd.edu/gtd

[2] Dash-Plotly Docs. https://dash.plotly.com/

∎