Terrorism Analysis with Insights using Python.

Under the guidance of

Dr. Sylvester Fernandes,
Yogendra Singh

By
T.Eswar

Table of Contents:

1. **Introduction ……..……..6**
2. **My work and implementation……....……7**
   2.1 About Data Set……....……...7
   2.2 About User Interface UI……...……8
      2.2.1 Map tool……...…...8
      2.2.2 Chart tool…….....8
   2.3 Explanation of Code…….. ……...9
      2.3.1 Libraries used……....9
      2.3.2 Main Function……..…….. 10
         2.3.2.1 Load data……..……11
         2.3.2.2 Open browser……..……..14
         2.3.2.3 Create app UI…….……..14
3. **Software requirements……..……31**
4. **Result…….....…….32**
5. **Conclusion and Future work…….……...61**
6. **Acknowledgement…….……62**
7. **References…….……..63**
8. **Biography…….……...64**

List of Figures:

Fig. 3.1: Execution of Program on Visual Studio Code
Fig. 3.2: Global Map Tool - Application depicting without any input in dropdowns and range slider
Fig. 3.3: Global Map Tool - Application depicting loading of graph
Fig. 3.4: Global Map Tool - The whole data of terrorism in the world is shown without any filters
Fig. 3.5: Global Map Tool - On hover of cursor onto an event shows the details of the event
Fig. 3.6: Global Map Tool - Few months are selected as per the user requirements through dropdown
Fig. 3.7: Global Map Tool - The graph shows filtered data as per selected months
Fig. 3.8: Global Map Tool - Few dates are selected as per selected months
Fig. 3.9: Global Map Tool - The graph shown filtered as per selected months and date
Fig.3.10: Global Map Tool - Few regions are selected
Fig. 3.11: Global Map Tool - The graph shows filtered data as per selected month, date and region
Fig. 3.12: Global Map Tool - Few countries are selected based on the selected region
Fig. 3.13: Global Map Tool - The graph shows filtered data as per selected month, date, region and country
Fig. 3.14: Global Map Tool - Few states are selected based on the selected regions and countries
Fig. 3.15: Global Map Tool - The graph shows filtered data as per selected month, date, region, country and state
Fig. 3.16: Global Map Tool - Few cities are selected based on selected regions, countries and states
Fig. 3.17: Global Map Tool - The graph shows filtered data as per selected month, date, region, country, state and city
Fig. 3.18: Global Map Tool - An attack type is selected
Fig. 3.19: Global Map Tool - The graph shows filtered data as per selected month, date, region, country, state, city and attack type
Fig. 3.20: Global Map Tool - range of years are selected
Fig. 3.21: Global Map Tool - The graph shows filtered data as per selected month, date, region, country, city, attack type and years

Fig. 3.22: India Map Tool - Application depicting without any inputs in dorpdown and range slider
Fig. 3.23: India Map Tool - The whole data of India is shown without any filters
Fig. 3.24: India Map Tool - Few months are selected as per the user requirements through dropdown
Fig. 3.25: India Map Tool - The graph shown filtered data as per selected month
Fig. 3.26: India Map Tool - Few dates are selected as per selected months
Fig. 3.27: India Map Tool - The graph shown filtered data as per selected month and date
Fig. 3.28: India Map Tool - Few states from india are selected
Fig. 3.29: India Map Tool - The graph shows filtered data as per selected month, date and state in india
Fig. 3.30: India Map Tool - Few cities are selected based on selected states and in india
Fig. 3.31: India Map Tool - The graph shows filtered data as per selected month, date, states and few cities in India
Fig. 3.32: India Map Tool - An attack type is selected
Fig. 3.33: India Map Tool - the graph shows filtered data as per selected month, date,state , city in india and attack type
Fig. 3.34: India Map Tool - A range of year is selected
Fig. 3.35: India Map Tool - The graph shows filtered data as per selected month, date, year, state and city in india and attack type
Fig. 3.36: Global chart Tool - Application depicting without any inputs (region selected as default) in dropdown and range slider
Fig. 3.37: Global chart Tool - the whole data without any filters
Fig. 3.38: Global chart Tool - Search filter is selected as per user requirements
Fig. 3.39: Global chart Tool - The chart shows filtered data as per search filter
Fig. 3.40: Global chart Tool - range of year slider is selected as per the user requirement
Fig. 3.41: Global chart Tool - The graph shows filtered data as per search filter and year slider
Fig. 3.42: Global chart Tool - Application depicting dropdown (part 1)
Fig. 3.43: Global chart Tool - Application depicting dropdown (part 2)
Fig. 3.44: Global chart Tool Type of attack is selected as per user requirements
Fig. 3.45: Global chart Tool - the graph shows filtered data as per type of attack
Fig. 3.46: Global chart Tool - search filter is selected as per user requirements (exp)
Fig. 3.47: Global chart Tool - the graph shows filtered data as per search filter and type of attack
Fig. 3.48: Global chart Tool - A range of year is selected using range slider
Fig. 3.49: Global chart Tool - The graph shows filtered data as per attack type, search filter and year slider
Fig. 3.50: India chart Tool - The whole data of terrorism in india is shown without any filters (regions is selected as default)
Fig. 3.51: India chart Tool - - The graph shows filtered data as per region in India

Fig. 3.52: India chart Tool - - Type of weapon is selected as per user requirements through dropdown
Fig. 3.53: India chart Tool - The graph shows filtered data as per type of weapon
Fig. 3.54: India chart Tool - The search filter is selected as per user requirement (exp)
Fig. 3.55: India chart Tool - The graph shows filtered data as per search filter and type of weapon
Fig. 3.56: India chart Tool - Range of years is selected using range slider
Fig. 3.57: India chart Tool - The graph shows filtered data as per type of weapon, search filter and year slider

1 Introduction:

As we have more information at our fingertips than ever before, the importance of data visualization has been greater than it is right now. Data visualization is the graphical representation of information and data with the help of charts, graphs, maps for getting insights of data.

In this project, Terrorism analysis with insights using python, Data set containing 1,91,465 records derived from Global Terrorism Database was used to analyze the information and project it in the form of maps and graphs. Initially the whole data set was visualized in the form of maps and charts. Depending on the user inputs, like date, month, year, region, country, state, city and attack type, only the preferential data had been filtered.

The User Interface (UI) contains multiple dropdowns, a range slider, map or graph tool and is user friendly.

2 My work and Implementation

2.1 ABOUT DATA SET:

The data set global_terror.csv contains the following columns:

- iyear → specifies the year of attack
- imonth → specifies the month of attack
- iday → specifies the day of attack
- country → specifies 205 unique countries
- country_txt → specifies the name of the country in accordance to country(unique code)
- Region → specifies 12 unique regions
- Region_txt → specifies the name of the region in accordance to region(unique code)
- Provstate → specifies 2580 unique province/ state
- City → specifies 39489 unique cities
- Latitude → specifies latitude of the location of attack
- Longitude → specifies longitude of the location of attack
- Attacktype1 → specifies 9 unique attack types
- Attacktype1_txt → specifies the name of attack type in accordance to attacktype1 (unique code)
- Targtype1 → specifies 22 unique target type
- Targtype1_txt → specifies the name of target type in accordance to targtype1 (unique code)
- Natlty1 → specifies unique nationalities
- Natlty1_txt → specifies the name of nationality in accordance to natlty1
- Gname → specifies the terrorist organization
- Weaptype1 → specifies the unique code of weapon
- Weaptype1_txt → specifies the name of the weapon in accordance to the weaptype1
- Nkill → specifies the number of people deceased in the attack
- Propextent → specifies the unique code of the property loss.
- Propextent_txt → specifies the amount of the damage incurred during the attack

2.2 About User Interface (UI)

Visualization of data is doone and represented in the below two ways

- Map tool
    - World map plot
    - India map plot
- Chart tool
    - World area chart - showing number of incident counts vs every year
    - India specific area chart - showing number of incident counts vs every year

2.2.1 MAP TOOL

Map User Interface

- There are seven dropdown and a year range slider in UI each for filtering the data as per user requirements based on the inputs
- The dropdown UI are
    - Months
    - Date
    - Region
    - Country
    - province/state
    - City
    - Attack type
- The range slider is for
    - Year
- The graph is also shown below them.(map).
- There are few restriction like
    - The user has to select month first and then day
    - The user should select region before selecting country and similarity down the path
    - Based on the data inputted, the data is filtered with the help of call backs
- The world map plot has a legend beside it, showing the attacktype based on the filters and they are clickable to select any
- India specific map plot: the region and country are fixed and set to south asia and india respectively. Rest everything works similarly to the world map tool.

2.2.2 Chart Tool:

The chart tool has a world chart tool and india specific chart tool

Chart tool User Interface:

- Incidents grouped by region initially. But with the help of drop down, grouping can be changed to any of the following
  - terrorist organization
  - target nationality
  - target type
  - type of attack
  - weapon type
  - region
  - country attacked.
- Search box filter
- Selecting range of years using range slider
- Area chart

The dropdown filter which groups incidents based on dropdown value, is also the legend of the area chart.

The search filters works as search for a specific requirement among all, like a country details or attack type or any.

India specific area chart: The region and country are fixed and set to South Asia and India respectively. Rest everything works similarly to the world chart tool.

2.3 Explanation of code

Github:https://github.com/eswar159/Terrorism-Analysis-with-Insights-using-python
Working:https://drive.google.com/file/d/1BI8_cjbp0HTZAeIbTx-Xtlh-sk4OgKbY/view

2.3.1 LIBRARIES USED:

```
import pandas as pd
import webbrowser
```

```
import dash
import dash_html_components as html
from dash.dependencies import Input, State, Output
import dash_core_components as dcc
import plotly.graph_objects as go
import plotly.express as px
from dash.exceptions import PreventUpdate
```

Pandas:

- It is used to read and operate on data easily and flexibly like reading data from csv and to get particular columns as per requirement

Webbrowser:

- It is used to open a tab in default browser you use

Dash:

- It is used to create an app and a dummy server to host the website

Dash_html_components:

- It consists of html components like headings, break, horizontal line etc.,

Dash.dependencies import input, state, output

- It is used in callbacks to take inputs and give outputs.

Dash_core_components:

- It has many useful components like graphs, maps, dropdowns, range slider etc.,

Plotly.graph_objects:

- It is used to generate figures

Plotly.express:

- It is used make charts and graphs

Dash.exception:

- It is used to prevent exceptions

Global variable declaration

```
app = dash.Dash()
```

App = dash.Dash():

- It is the first step to make an app in dash. The object named app is created.

2.3.2 MAIN FUNCTION:

- The program execution is started here.

```
if __name__ == '__main__':
  application()
```

Here application is a method or function which is invoked when the programs starts its execution.

```
def application():

 load_data()

 open_browser()

 global app
 app.layout = create_app_ui()

 app.title = "Terrorism Analysis with Insights"

 app.run_server()

 print("Stopped by the user")
 #df = None
 app = None
```

Method named application contains the following components:

1. Load data
2. Open browser
3. create_ app_ui

App.layout

- It stores the layout from create_app_ui

App.title

- It specifies the title of the app

App.run_server

- It starts the server

df=None and app=None:

- Once the program is terminated df and app are removed

2.3.2.1 Load data

```
def load_data():
 dataset_name = "global_terror.csv"

 pd.options.mode.chained_assignment = None

 global df
 df = pd.read_csv(dataset_name)

 global month_list
 month = {
       "January":1,
       "February": 2,
       "March": 3,
       "April":4,
       "May":5,
       "June":6,
       "July": 7,
       "August":8,
       "September":9,
       "October":10,
       "November":11,
       "December":12
       }
 month_list= [{"label":key, "value":values} for key,values in month.items()]

 global date_list
 date_list = [x for x in range(1, 32)]


 global region_list
 region_list = [{"label": str(i), "value": str(i)}  for i in sorted( df['region_txt'].unique().tolist() ) ]
```

```
global country_list
country_list = df.groupby("region_txt")["country_txt"].unique().apply(list).to_dict()


global state_list
state_list = df.groupby("country_txt")["provstate"].unique().apply(list).to_dict()


global city_list
city_list  = df.groupby("provstate")["city"].unique().apply(list).to_dict()


global attack_type_list
attack_type_list = [{"label": str(i), "value": str(i)}  for i in df['attacktype1_txt'].unique().tolist()]


global year_list
year_list = sorted ( df['iyear'].unique().tolist()  )

global year_dict
year_dict = {str(year): str(year) for year in year_list}

global chart_dropdown_values
chart_dropdown_values = {"Terrorist Organisation":'gname',
                         "Target Nationality":'natlty1_txt',
                         "Target Type":'targtype1_txt',
                         "Type of Attack":'attacktype1_txt',
                         "Weapon Type":'weaptype1_txt',
                         "Region":'region_txt',
                         "Country Attacked":'country_txt'
                        }

chart_dropdown_values = [{"label":keys, "value":value} for keys, value in
chart_dropdown_values.items()]
```

Load_data:

- Dataset_ name = “global_terror.csv”
    - The file name is stored in dataset_name.

- Df = pd.read_csv(dataset_name)
    - It is used to read the data from csv file
- Each column values of the data set is read, unique names are selected and they are converted into a dictionary format

  [{“label”: ”label name”, “value”: “value name”}, {“label”: ”label name1”, “value”: “value name1”}, ……..]

  As this format is used in the dropdown, the column values are converted into this format.

  All the columns values are collected and converted in the above format. The variables are made global, as they need to be available in the whole program.

  They are
    - region_list
    - Country_list
    - state_list
    - City_list
    - Attack_type_list
    - Year_dict
    - chart_dropdown_values

- Whereas months_list and date_list are hard coded, as they are fixed
- Few functions used are
    - unique() → to get the unique values among all
    - to_dict→ converts into dictionary
    - to_list→ converts into list
    - Basic for loops are used to iterate

2.3.2.2 Open_browser

```
def open_browser():
  webbrowser.open_new('http://127.0.0.1:8050/')
```

- webbrowser.open_new(‘http://127.0.0.1:8050/’)
    - It is used to open open the webpage (of the specified link) in default browser

2.3.2.3 create_app_ui

```
def create_app_ui():

 dropdown_style={
       'width': '99%',
       'margin': 'auto',
       "padding-top": "4px",
       "padding-bottom": "2px",
       "border-radius":"25px"
  }

 main_layout = html.Div(
     style={"background-color": "#FFE9DB"},
     className="Test",
     children=[
 html.H1('Terrorism Analysis with Insights', id='Main_title',style={"text-align":
"center","background-color": "#FFA38A","margin":"0","padding-top": "7px","padding-bottom": "7px"}),
   dcc.Tabs(id="Tabs", value="Map",children=[
     dcc.Tab(label="Map tool    "    u"\U0001F30D" ,selected_style={"fontWeight":
"bold"},style={"fontWeight": "bold"},id="Map tool",value="Map",children=[
         dcc.Tabs(id = "subtabs", value = "WorldMap",children = [
            dcc.Tab(label="World Map tool    " u"\U0001F30D",selected_style={"fontWeight":
"bold"},style={"fontWeight": "bold"}, id="World", value="WorldMap"),
            dcc.Tab(label="India Map tool    " u"\U0001F30D",selected_style={"fontWeight":
"bold"},style={"fontWeight": "bold"}, id="India", value="IndiaMap")
            ]),
         dcc.Dropdown(
            id='month',
              options=month_list,
              placeholder='Select Month',
              multi = True,
              style=dropdown_style
               ),
         dcc.Dropdown(
              id='date',
```

```
            placeholder='Select Day',
            multi = True,
            style=dropdown_style
              ),
        dcc.Dropdown(
            id='region_dropdown',
            options=region_list,
            placeholder='Select Region',
            multi = True,
            style=dropdown_style
              ),
        dcc.Dropdown(
            id='country_dropdown',
            options=[{'label': 'All', 'value': 'All'}],
            placeholder='Select Country',
            multi = True,
            style=dropdown_style
              ),
        dcc.Dropdown(
            id='state_dropdown',
            options=[{'label': 'All', 'value': 'All'}],
            placeholder='Select State or Province',
            multi = True,
            style=dropdown_style
              ),
        dcc.Dropdown(
            id='city_dropdown',
            options=[{'label': 'All', 'value': 'All'}],
            placeholder='Select City',
            multi = True,
            style=dropdown_style
              ),
        dcc.Dropdown(
            id='attacktype_dropdown',
            options=attack_type_list,
            placeholder='Select Attack Type',
            multi = True,
            style=dropdown_style
              ),
```

```
        html.H4('Select the Year', id='year_title',style={"padding-left": "20px"}),
        dcc.RangeSlider(
                id='year_slider',
                min=min(year_list),
                max=max(year_list),
                value=[min(year_list),max(year_list)],
                marks=year_dict,
                step=None
                  ),
    ]),
     dcc.Tab(label = "Chart Tool   " u"\U0001F4CA",selected_style={"fontWeight":
"bold"},style={"fontWeight": "bold"}, id="chart tool", value="Chart", children=[
        dcc.Tabs(id = "subtabs2", value = "WorldChart",children = [
           dcc.Tab(label="World Chart tool   "u"\U0001F4CA",selected_style={"fontWeight":
"bold"},style={"fontWeight": "bold"}, id="WorldC", value="WorldChart"),
           dcc.Tab(label="India Chart tool   "u"\U0001F4CA",selected_style={"fontWeight":
"bold"},style={"fontWeight": "bold"}, id="IndiaC", value="IndiaChart")]),
           dcc.Dropdown(id="Chart_Dropdown", options = chart_dropdown_values, placeholder="Select
option", value = "region_txt",style={'width': '99%','margin': 'auto',"padding-top": "2px","padding-bottom":
"2px","border-radius":"25px"}),
           html.Br(),
           html.Hr(),
           html.Center( dcc.Input(id="search", placeholder="Search
Filter",style={"width":"97%","padding-bottom": "10px","padding-top": "10px","border-radius":"25px"})
),
           html.Hr(),
           html.Br(),
           html.H4('Select the Year', id='year_titlee',style={"padding-left": "20px"}),
           dcc.RangeSlider(
                id='cyear_slider',
                min=min(year_list),
                max=max(year_list),
                value=[min(year_list),max(year_list)],
                marks=year_dict,
                step=None
                  ),
          html.Br()
            ]),
       ]),
```

```
  dcc.Loading(children=[dcc.Graph(id='graph-object')],type='graph',style={"backgroundColor":
"transparent","z-index":"1","position":"absolute"}),
  html.Footer(id= "footer",
     style={"padding-bottom": "1px"},
     children=[
        html.H3(style={"text-align": "center"},children=[
           html.H3(children="Made with ",style={"display": "inline"}),
           html.H3(children="\u2764",style={"color":"red","display": "inline"}),
           html.H3(children=" by ESWAR",style={"display": "inline"}),
           ])
        ])


 ])

 return main_layout
```

- dropdown_style={
  “Width” : “99%”,
  “Margin” : “auto”,
  “Padding-top” : “4px”,
  “Padding-bottom” : “2px”,
  “Border-radius” : ”25px”
  }
  - Dropdown_style contains the styling for the dropdowns. (css styling)
- Main_layout =
  - It contains the whole layout of the webapp
  - It contains the following components
    - html.H1 to html.H6 → specifies the headings with their sizes in decreasing order respectively.
      - Title
      - Id → specifies a unique identification for the component
      - Style = {....} → specifies styling for the particular components
    - dcc.Tab and tabs → used to create tab and subtabs
      - label→ specifying the name of the tab
      - selected _style = {....} → specifies the selected styling
      - style={...} → specifies style of tab
      - Id → specifies unique identification for the component

        - Value
        - Children = [....] → specifies the subcomponents inside that tab
    - dcc.Dropdown
        - Id → specifies unique identification for the component
        - Options
        - Placeholder → specifies the name of the dropdown when nothing is selected
        - Multi = true → specifies that multiple values can be selected in the dropdown
        - style={...} → specifies style of dropdown
    - dcc.Rangeslider → to select range (between any two values)
        - Id → specifies unique identification for the component
        - Min
        - Max
        - Value
        - Marks → specifies the numbers on the ring slider
    - html.Br → specifies a break.
    - html.Hr → specifies horizontal line

Return main_layout:

- Returns the main layout which has been created in the above steps.

```
@app.callback(dash.dependencies.Output('graph-object', 'figure'),
  [
   dash.dependencies.Input("Tabs", "value"),
  dash.dependencies.Input('month', 'value'),
  dash.dependencies.Input('date', 'value'),
  dash.dependencies.Input('region_dropdown', 'value'),
  dash.dependencies.Input('country_dropdown', 'value'),
  dash.dependencies.Input('state_dropdown', 'value'),
  dash.dependencies.Input('city_dropdown', 'value'),
  dash.dependencies.Input('attacktype_dropdown', 'value'),
  dash.dependencies.Input('year_slider', 'value'),
  dash.dependencies.Input('cyear_slider', 'value'),

  dash.dependencies.Input("Chart_Dropdown", "value"),
  dash.dependencies.Input("search", "value"),
  dash.dependencies.Input("subtabs2", "value")
  ]
  )
```

```
def update_app9_ui(Tabs, month_value,
date_value,region_value,country_value,state_value,city_value,attack_value,year_value,chart_year_selector
, chart_dp_value, search,
                   subtabs2):
    figure = None

    if Tabs == "Map":
        print("Data Type of month value = " , str(type(month_value)))
        print("Data of month value = " , month_value)

        print("Data Type of Day value = " , str(type(date_value)))
        print("Data of Day value = " , date_value)

        print("Data Type of region value = " , str(type(region_value)))
        print("Data of region value = " , region_value)

        print("Data Type of country value = " , str(type(country_value)))
        print("Data of country value = " , country_value)

        print("Data Type of state value = " , str(type(state_value)))
        print("Data of state value = " , state_value)

        print("Data Type of city value = " , str(type(city_value)))
        print("Data of city value = " , city_value)

        print("Data Type of Attack value = " , str(type(attack_value)))
        print("Data of Attack value = " , attack_value)

        print("Data Type of year value = " , str(type(year_value)))
        print("Data of year value = " , year_value)

        # year_filter
        year_range = range(year_value[0], year_value[1]+1)
        new_df = df[df["iyear"].isin(year_range)]

        # month_filter
        if month_value==[] or month_value is None:
            pass
        else:
```

```
    if date_value==[] or date_value is None:
        new_df = new_df[new_df["imonth"].isin(month_value)]
    else:
        new_df = new_df[new_df["imonth"].isin(month_value)
                        & (new_df["iday"].isin(date_value))]

# region, country, state, city filter
if region_value==[] or region_value is None:
    pass
else:
    if country_value==[] or country_value is None :
        new_df = new_df[new_df["region_txt"].isin(region_value)]
    else:
        if state_value == [] or state_value is None:
            new_df = new_df[(new_df["region_txt"].isin(region_value))&
                            (new_df["country_txt"].isin(country_value))]
        else:
            if city_value == [] or city_value is None:
                new_df = new_df[(new_df["region_txt"].isin(region_value))&
                (new_df["country_txt"].isin(country_value)) &
                (new_df["provstate"].isin(state_value))]
            else:
                new_df = new_df[(new_df["region_txt"].isin(region_value))&
                (new_df["country_txt"].isin(country_value)) &
                (new_df["provstate"].isin(state_value))&
                (new_df["city"].isin(city_value))]

if attack_value == [] or attack_value is None:
    pass
else:
    new_df = new_df[new_df["attacktype1_txt"].isin(attack_value)]

mapFigure = go.Figure()
if new_df.shape[0]:
    pass
else:
    new_df = pd.DataFrame(columns = ['iyear', 'imonth', 'iday', 'country_txt', 'region_txt', 'provstate',
      'city', 'latitude', 'longitude', 'attacktype1_txt', 'nkill'])

    new_df.loc[0] = [0, 0 ,0, None, None, None, None, None, None, None, None]
```

```
        mapFigure = px.scatter_mapbox(new_df,
          lat="latitude",
          lon="longitude",
          color="attacktype1_txt",
          hover_name="city",
          hover_data=["region_txt", "country_txt", "provstate","city",
"attacktype1_txt","nkill","iyear","imonth", "iday"],
          zoom=1
          )
        mapFigure.update_layout(mapbox_style="open-street-map",
          autosize=True,
          margin=dict(l=20, r=20, t=20, b=20),
          )

        figure = mapFigure

    elif Tabs=="Chart":
        figure = None


        year_range_c = range(chart_year_selector[0], chart_year_selector[1]+1)
        chart_df = df[df["iyear"].isin(year_range_c)]


        if subtabs2 == "WorldChart":
            pass
        elif subtabs2 == "IndiaChart":
            chart_df = chart_df[(chart_df["region_txt"]=="South Asia") &(chart_df["country_txt"]=="India")]
        if chart_dp_value is not None and chart_df.shape[0]:
            if search is not None:
                chart_df = chart_df.groupby("iyear")[chart_dp_value].value_counts().reset_index(name =
"count")
                chart_df  = chart_df[chart_df[chart_dp_value].str.contains(search, case=False)]
            else:
                chart_df =
chart_df.groupby("iyear")[chart_dp_value].value_counts().reset_index(name="count")
```

```
    if chart_df.shape[0]:
        pass
    else:
        chart_df = pd.DataFrame(columns = ['iyear', 'count', chart_dp_value])

        chart_df.loc[0] = [0, 0,"No data"]
    chartFigure = px.area(chart_df, x="iyear", y ="count", color = chart_dp_value)
    figure = chartFigure
return figure
```

```
@app.callback(dash.dependencies.output("graph-object", "figure"),
        [
        dash.dependencies.Input("tabs","value"),
        …
        …
        …
        …
        …
        dash.dependencies.Input("subtabs2","value")
        ]
        )
```

- It collects all the inputs and gives a single output
- Then the method or function below is called automatically with parameters which are input in the above call back.

```
Figure = None
```

- Initially figure is initialized to none, everytime when the method is called.

```
If tabs == "map":
```

- Condition checks whether the selected tab is mapped, if yes, executes the statement inside it.

Few print statements are used to check the type of the variables and the value inside them.

#year_filter

```
Year_range = range(year_value[0],year_value[1]+1)
New_df = df[df["iyear"].isin(year_range)]
```

- Year_range → stores the range of values in between the user input years from the range slider.
- Eg: 2000 - 2005 - selected by the user
- Year_value[0] = 2000, year_value[1] = 2005
- Range (x,y) → gives the values in between x and y including x but not y.

- So in order to get values from 2000 to 2005, range(year_value[0],year_value[1]+1) is used, as it specifies range of 2000 to 2006.
- Hence year_range contains values from 2000 to 2006
- Now dataset (df) is filtered based on the user input i.e year_range
- If input is not given, whole data years are selected.

 # month_filter

```
if month_value==[] or month_value is None:
    pass
else:
    if date_value==[] or date_value is None:
        new_df = new_df[new_df["imonth"].isin(month_value)]
    else:
        new_df = new_df[new_df["imonth"].isin(month_value)
                & (new_df["iday"].isin(date_value))]
```

- If month is not selected, no filtering is done based on month or date
- else
    - If date is not selected, the data is filtered based on month
    - Else, data is filtered based on both months and dates.

 # region, country, state, city filter

```
if region_value==[] or region_value is None:
    pass
else:
    if country_value==[] or country_value is None :
        new_df = new_df[new_df["region_txt"].isin(region_value)]
    else:
        if state_value == [] or state_value is None:
            new_df = new_df[(new_df["region_txt"].isin(region_value))&
                    (new_df["country_txt"].isin(country_value))]
        else:
            if city_value == [] or city_value is None:
                new_df = new_df[(new_df["region_txt"].isin(region_value))&
                (new_df["country_txt"].isin(country_value)) &
                (new_df["provstate"].isin(state_value))]
            else:
                new_df = new_df[(new_df["region_txt"].isin(region_value))&
                (new_df["country_txt"].isin(country_value)) &
                (new_df["provstate"].isin(state_value))&
                (new_df["city"].isin(city_value))]
```

- If region is not selected, no data is filtered
- Else,

- If country is not selected, data is filtered based on region
- Else,
    - If state is not selected, data is filtered based on region and country
    - Else
        - If city is not selected, data is filtered based on region, country and state
        - else , if city is selected, data is selected based on region, country, state and city

#attacktype

```
if attack_value == [] or attack_value is None:
        pass
    else:
        new_df = new_df[new_df["attacktype1_txt"].isin(attack_value)]
```

- If attack type is not selected, the data is not filtered.
- Else, data is filtered based on attack type selected.

Now the final data for plotting on the map is ready in new_df with all the filters applied by the users.

#figure

```
    mapFigure = go.Figure()
```

- An empty figure is created and stored in mapFigure.

```
    if new_df.shape[0]:
        pass
    else:
        new_df = pd.DataFrame(columns = ['iyear', 'imonth', 'iday', 'country_txt', 'region_txt',
'provstate',
          'city', 'latitude', 'longitude', 'attacktype1_txt', 'nkill'])

        new_df.loc[0] = [0, 0 ,0, None, None, None, None, None, None, None, None]
```

- If the shape of the new_df is greater than 0, there are atleast one record in new_df which are filtered based on user input
- Else, as there is no records in new_df, all column stored value are made to default i.e 0 or None

```
    mapFigure = px.scatter_mapbox(new_df,
      lat="latitude",
      lon="longitude",
      color="attacktype1_txt",
      hover_name="city",
```

```
        hover_data=["region_txt", "country_txt", "provstate","city",
"attacktype1_txt","nkill","iyear","imonth", "iday"],
        zoom=1
        )
```

- Scatter_mapbox → is used to scatter the points on the graph.
  - new_df→ the data set is passed
  - Lat → specifies latitude
  - lon→ specifies longitude
  - Color → specifies unique color based on attack type
  - Hover_name → displays heading name
  - Hover_data
  - Zoom → range of zoom value between 1 t o15

```
      mapFigure.update_layout(mapbox_style="open-street-map",
        autosize=True,
        margin=dict(l=20, r=20, t=20, b=20),
        )

      figure = mapFigure
```

- Update_layout → is used to update mapFigure
  - Mapbox_style → specifies the style of map
  - Autosize
  - Margin → in all directions (left, right, top and bottom)

Finally mapFigure is stored in figure

```
elif Tabs=="Chart":
```

- Elif → else if, selected tab is chart, execute the following statements

```
      year_range_c = range(chart_year_selector[0], chart_year_selector[1]+1)
      chart_df = df[df["iyear"].isin(year_range_c)]
```

- Data is filtered based on the years selected by the user
- If no input is given, whole data years are selected.

```
      if subtabs2 == "WorldChart":
         pass
      elif subtabs2 == "IndiaChart":
```

```
        chart_df = chart_df[(chart_df["region_txt"]=="South Asia")
&(chart_df["country_txt"]=="India")]
```

If subtabs2 is world chart, then no data is filtered

- Else, if subtabs2 is india chart, data is filtered as region is south asia and country is india.

```
if chart_dp_value is not None and chart_df.shape[0]:
        if search is not None:
            chart_df =
chart_df.groupby("iyear")[chart_dp_value].value_counts().reset_index(name = "count")
            chart_df  = chart_df[chart_df[chart_dp_value].str.contains(search, case=False)]
        else:
            chart_df =
chart_df.groupby("iyear")[chart_dp_value].value_counts().reset_index(name="count")
```

- If input is given in chart dropdown,
    - If input is also given for search, then data is filtered based on both chart dropdown and search
    - Else, the data is filtered based on only chart dropdown.

```
    if chart_df.shape[0]:
        pass
    else:
        chart_df = pd.DataFrame(columns = ['iyear', 'count', chart_dp_value])
```

- If the shape of the chart_df is greater than 0, there are atleast one record in chart_df which are filtered based on user input
- Else, as there are no records in chart_df, all column stored values are made to default.

```
        chart_df.loc[0] = [0, 0,"No data"]
    chartFigure = px.area(chart_df, x="iyear", y ="count", color = chart_dp_value)
    figure = chartFigure
  return figure
```

- Based on chart_df data, area chart is made using area method.
    - Chart_df as dataset
    - X → x axis value
    - Y → y axis value
    - Color → chart_dp_value (unique colors)

Finally chartFigure is stored in figure.

Return figure

Finally figure is returned as the output from the call back

```
@app.callback(
 Output("date", "options"),
 [Input("month", "value")])
def update_date(month):
   option = []
   if month:
      option= [{"label":m, "value":m} for m in date_list]
   return option
```

- The above callback is used to add days to the dropdown based on the month selected.
- To be simple, the date_list values ranging from 1 to 31 are given as multiple months can be selected.
- Finally the date_dropdown options are returned.

```
@app.callback([Output("region_dropdown", "value"),
               Output("region_dropdown", "disabled"),
               Output("country_dropdown", "value"),
               Output("country_dropdown", "disabled")],
              [Input("subtabs", "value")])
def update_r(tab):
   region = None
   disabled_r = False
   country = None
   disabled_c = False
   if tab == "WorldMap":
      pass
   elif tab=="IndiaMap":
      region = ["South Asia"]
```

```
        disabled_r = True
        country = ["India"]
        disabled_c = True
    return region, disabled_r, country, disabled_c
```

- The above callback is used to disable region and country dropdown based on the user's tab selection.
    - If selected tab is world map, region and country are not disable
    - Else, if selected tab is india map, region and country are enabled and set to south asia and india respectively
    - It returns region, disabled_r, country, disabled_c

```
@app.callback(
    Output('country_dropdown', 'options'),
    [Input('region_dropdown', 'value')])
def set_country_options(region_value):
    option = []
    if region_value is  None:
        raise PreventUpdate
    else:
        for var in region_value:
            if var in country_list.keys():
                option.extend(country_list[var])
    return [{'label':m , 'value':m} for m in option]
```

- The above callback is used to filter countries based on the selected regions.
    - If a region is not selected, then countries are not filtered.
    - Else, countries are filtered based on the regions and converted into the dictionary format which will be returned at the end.

```
@app.callback(
    Output('state_dropdown', 'options'),
    [Input('country_dropdown', 'value')])
```

```
def set_state_options(country_value):
    option = []
    if country_value is None :
        raise PreventUpdate
    else:
        for var in country_value:
            if var in state_list.keys():
                option.extend(state_list[var])
    return [{'label':m , 'value':m} for m in option]
```

- The above callback is used to filter states based on the selected country.
    - If a country is not selected, then states are not filtered.
    - Else, states are filtered based on the country and converted into the dictionary format which will be returned at the end.

```
@app.callback(
    Output('city_dropdown', 'options'),
    [Input('state_dropdown', 'value')])
def set_city_options(state_value):
    option = []
    if state_value is None:
        raise PreventUpdate
    else:
        for var in state_value:
            if var in city_list.keys():
                option.extend(city_list[var])
    return [{'label':m , 'value':m} for m in option]
```

- The above callback is used to filter cities based on the selected states.
    - If a state is not selected, then cities are not filtered.
    - Else, cities are filtered based on the state and converted into the dictionary format which will be returned at the end.

The project or application can be executed directly from the Command prompt (cmd) or from any integrated development environment (ide) like spyder, visual studio code, pycharm etc.,

3.System Requirements

System requirements:

- Processor Type: Intel i3, i5 or higher
- RAM: 4 GB or above
- Disc Space: 250 GB
- Language: Python 3.5 x or above

Software requirements:

- Operating system: Microsoft Windows 7, Windows 8, Windows 10
- Language: Python 3.8x
- Python Modules: Pandas, webbroser
- Python Framework: Plotly's Dash 1.0
- Python IDE or Command Prompt
- Internet Browser like Google Chrome, Mozilla Firefox, Microsoft Edge, Safari

4. Results

The application is executed on visual studio has shown below.

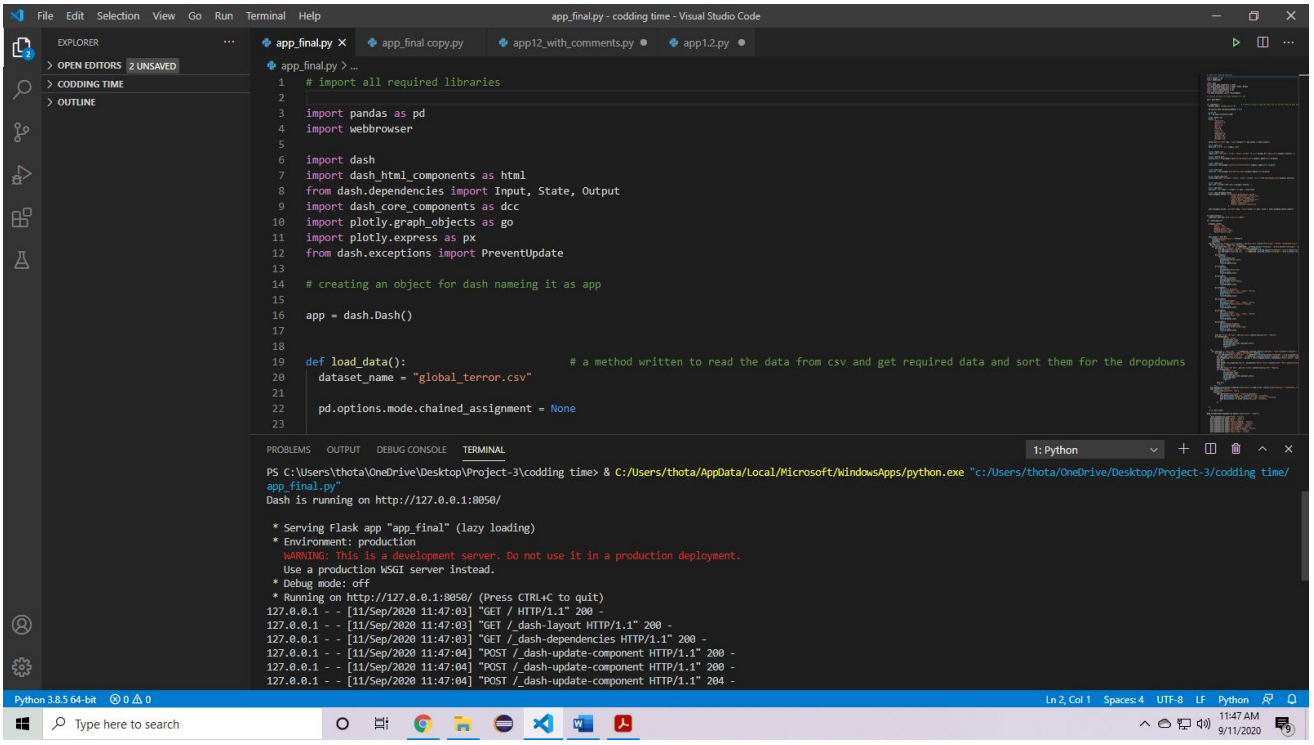

Fig 3.1 : execution of program on Visual Studio Code

Outputs:

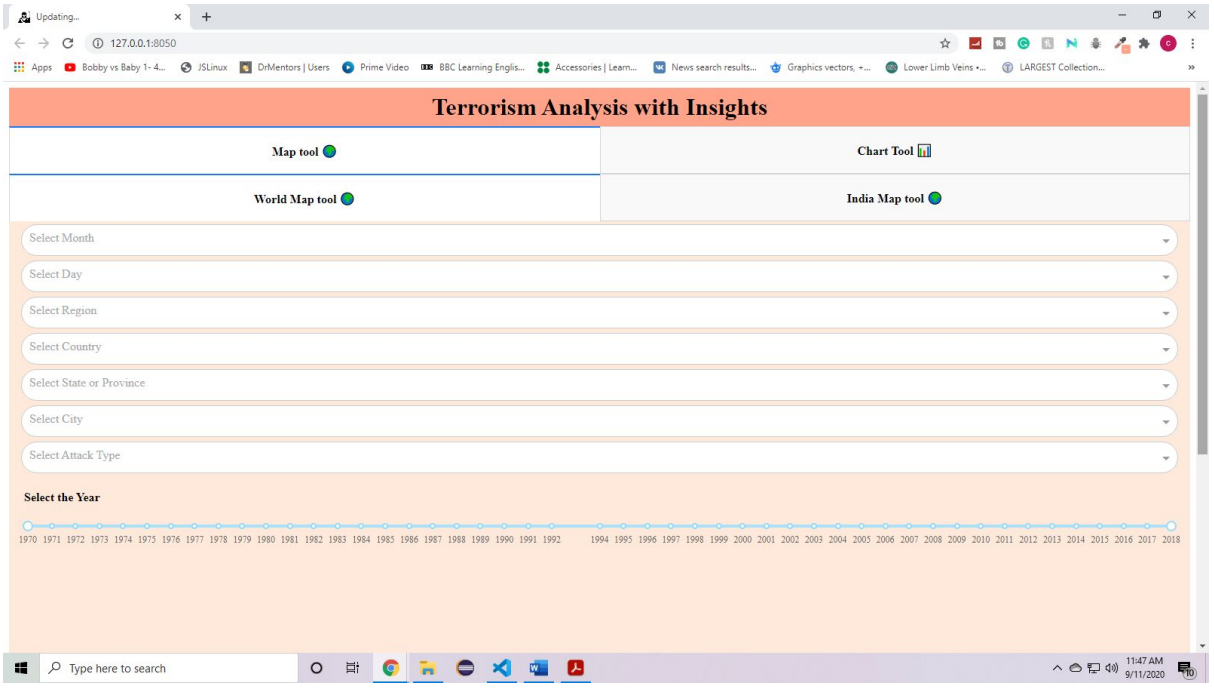

Fig 3.2: Global Map Tool - application depicting without any inputs in dropdowns and range slider

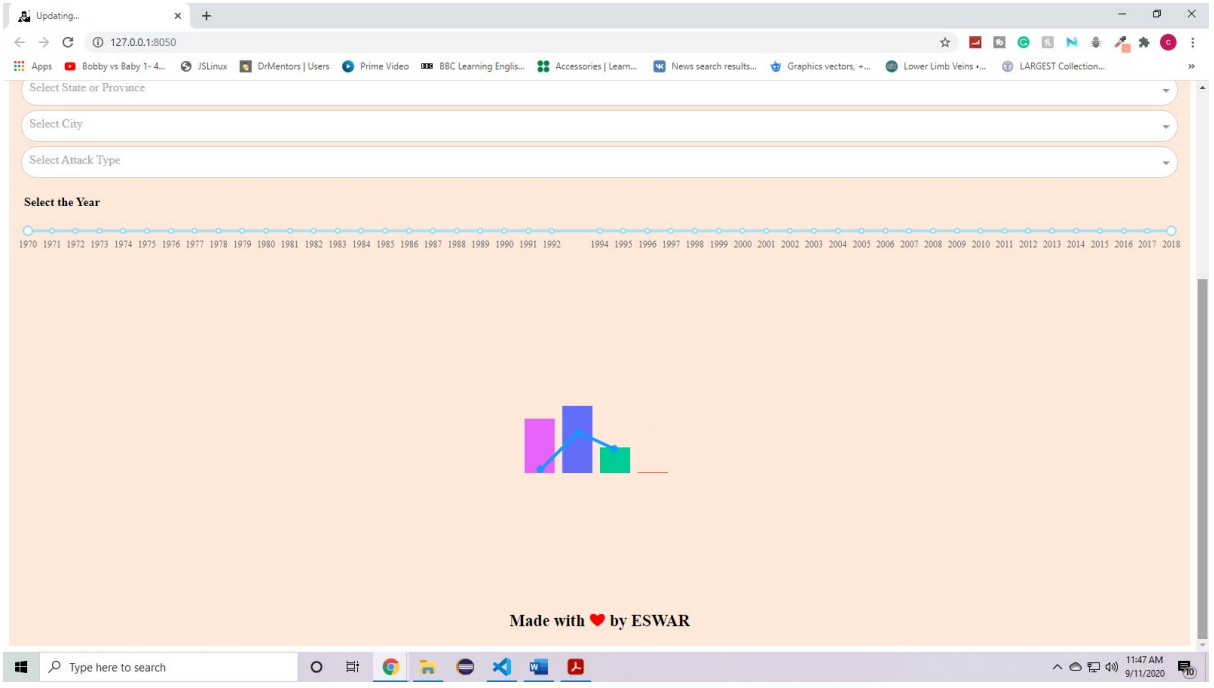

Fig 3.3 : Global Map Tool - application depicting loading of graph

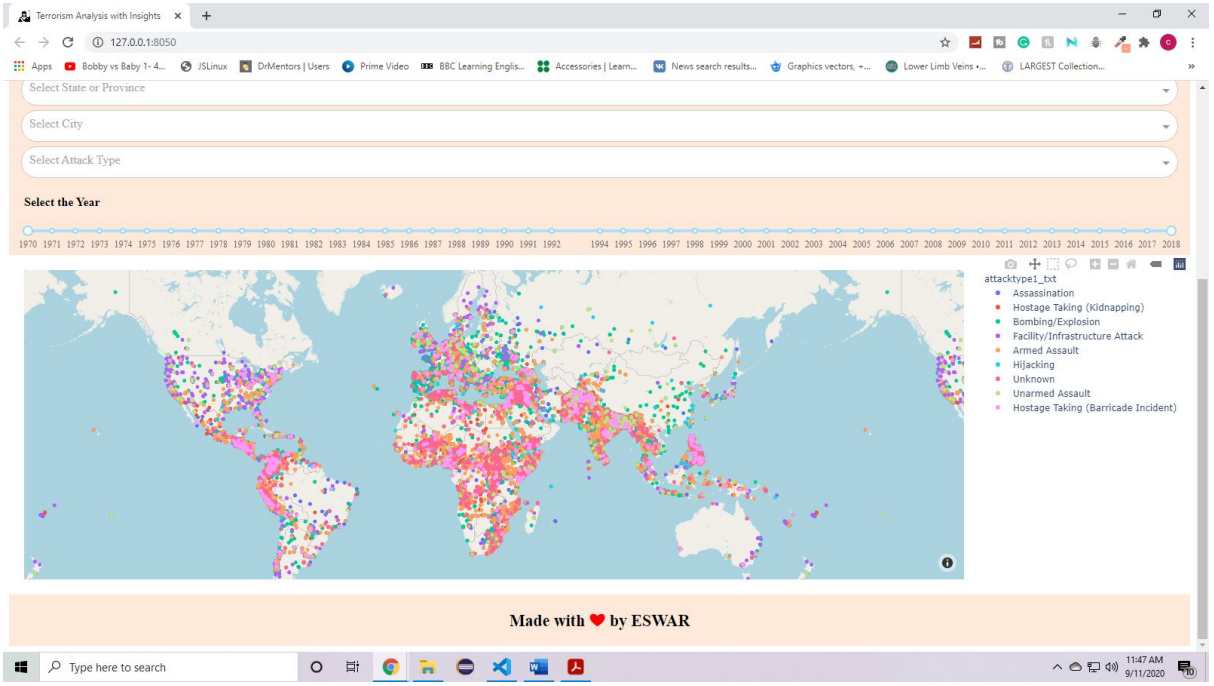

Fig 3.4 : Global Map Tool - The whole data of terrorism in world is shown without any filters

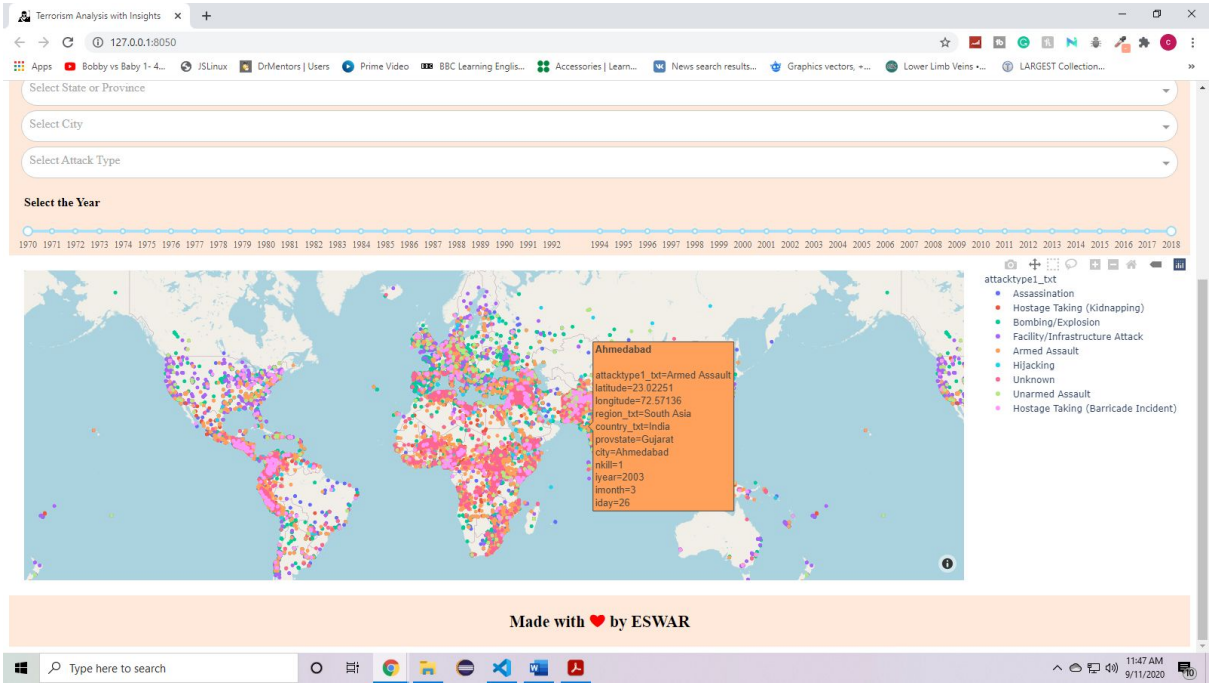

Fig 3.5 : Global Map Tool - On hover of cursor onto an event shows the details of the event

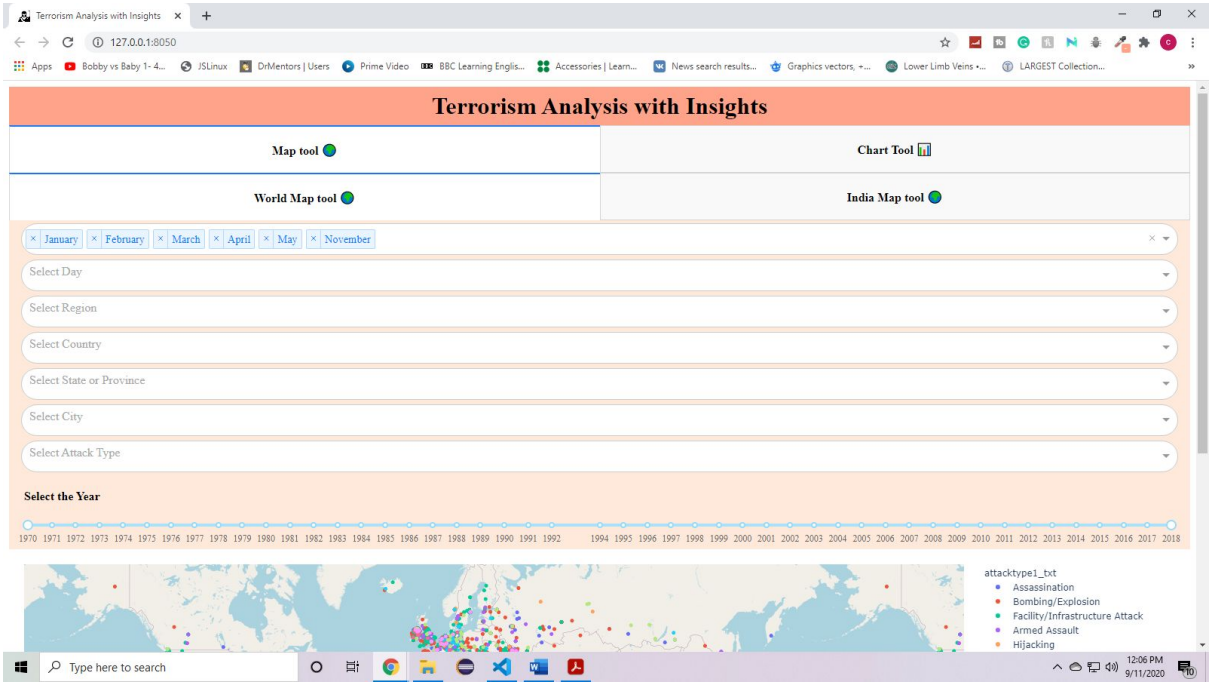

Fig 3.6 : Global Map Tool - Few months are selected as per the user requirements through dropdown

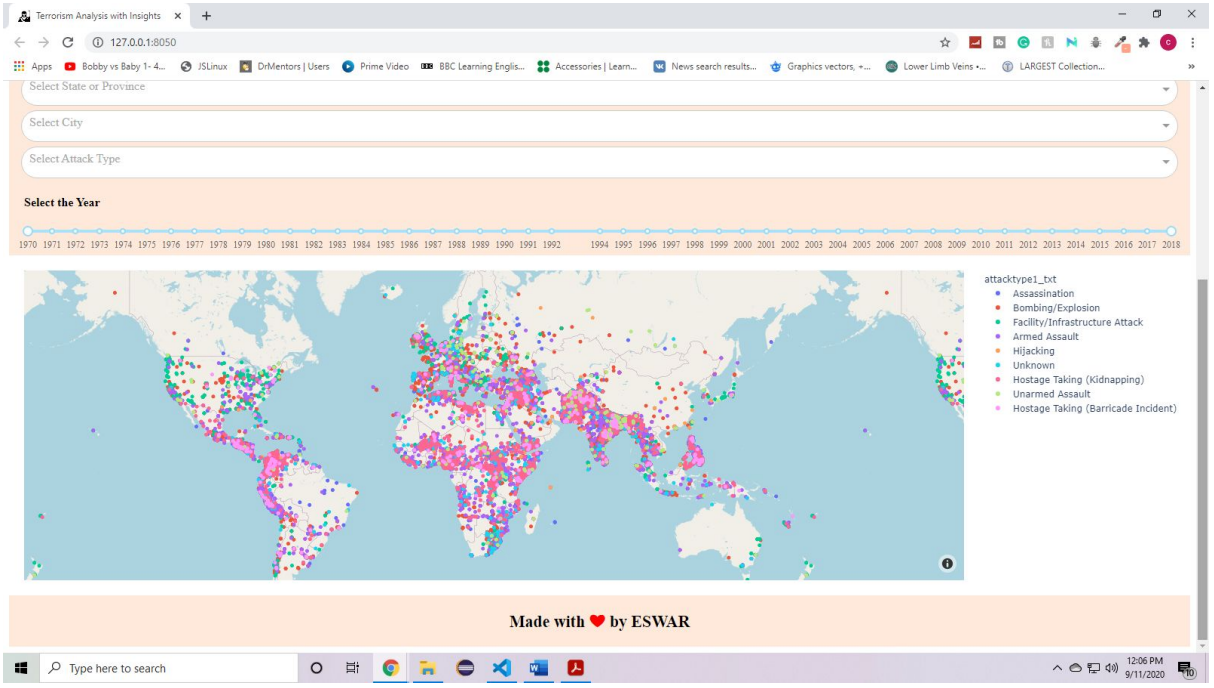

Fig 3.7 : Global Map Tool - The graph shows filtered data as per selected months

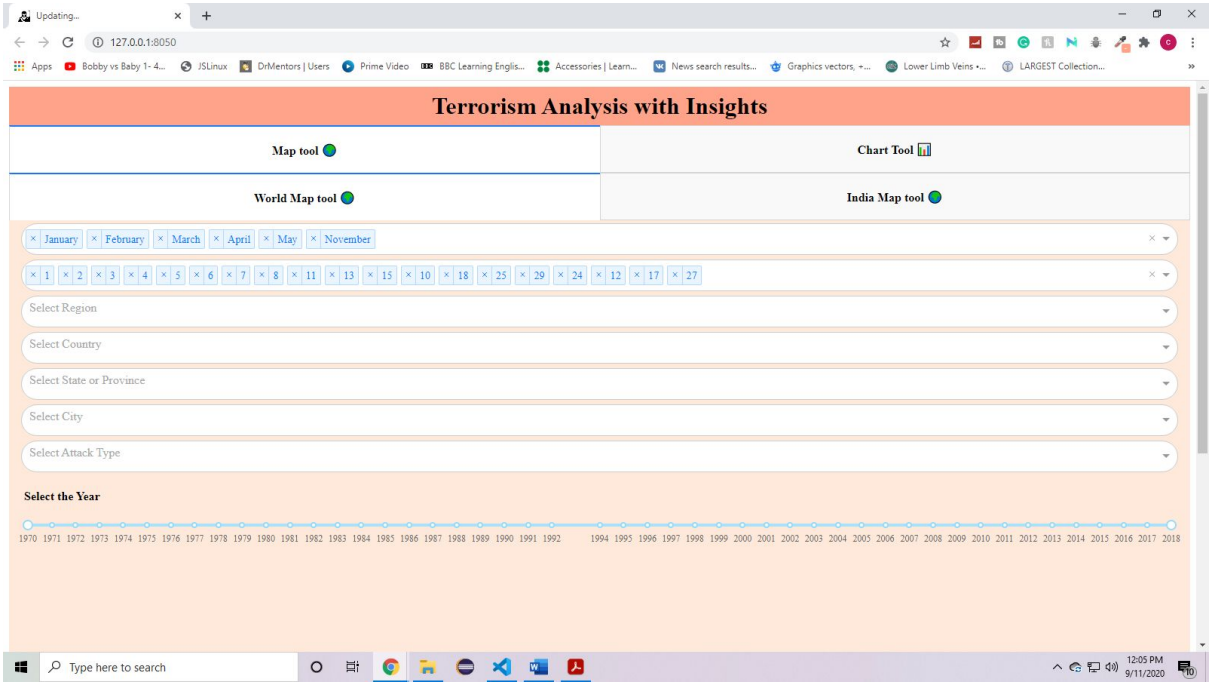

Fig 3.8 : Global Map Tool - Few dates are selected as per selected months.

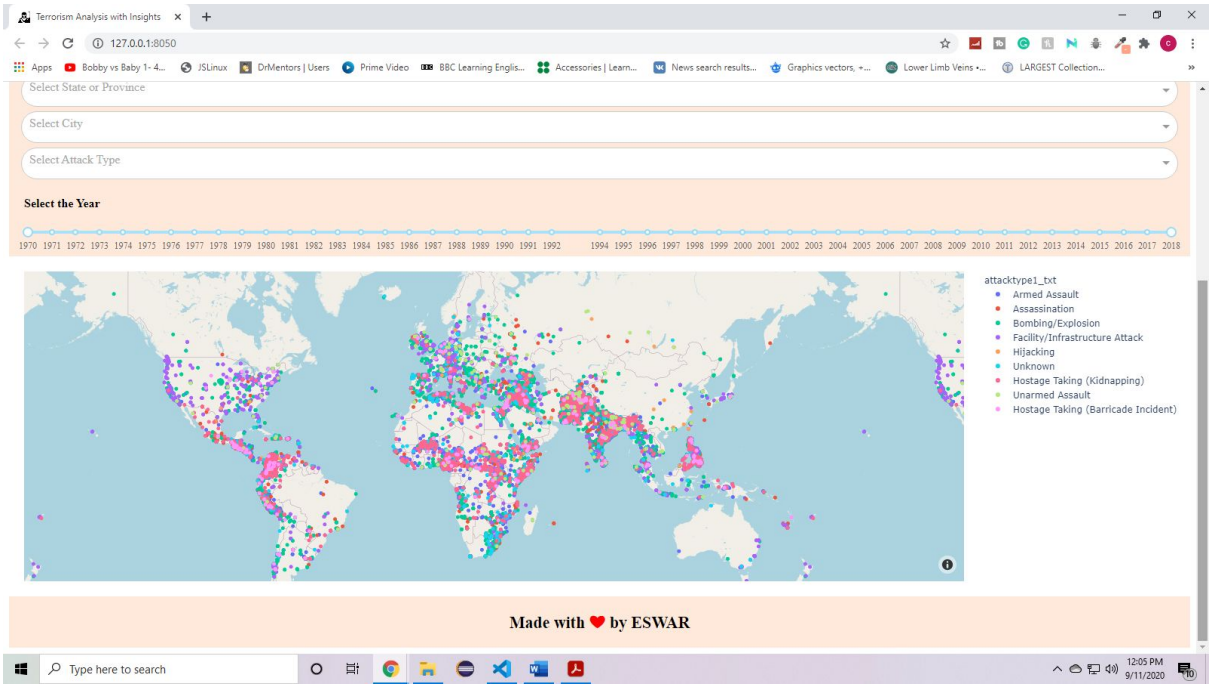

Fig 3.9 : Global Map Tool - The graph shown filtered data as per selected months and date.

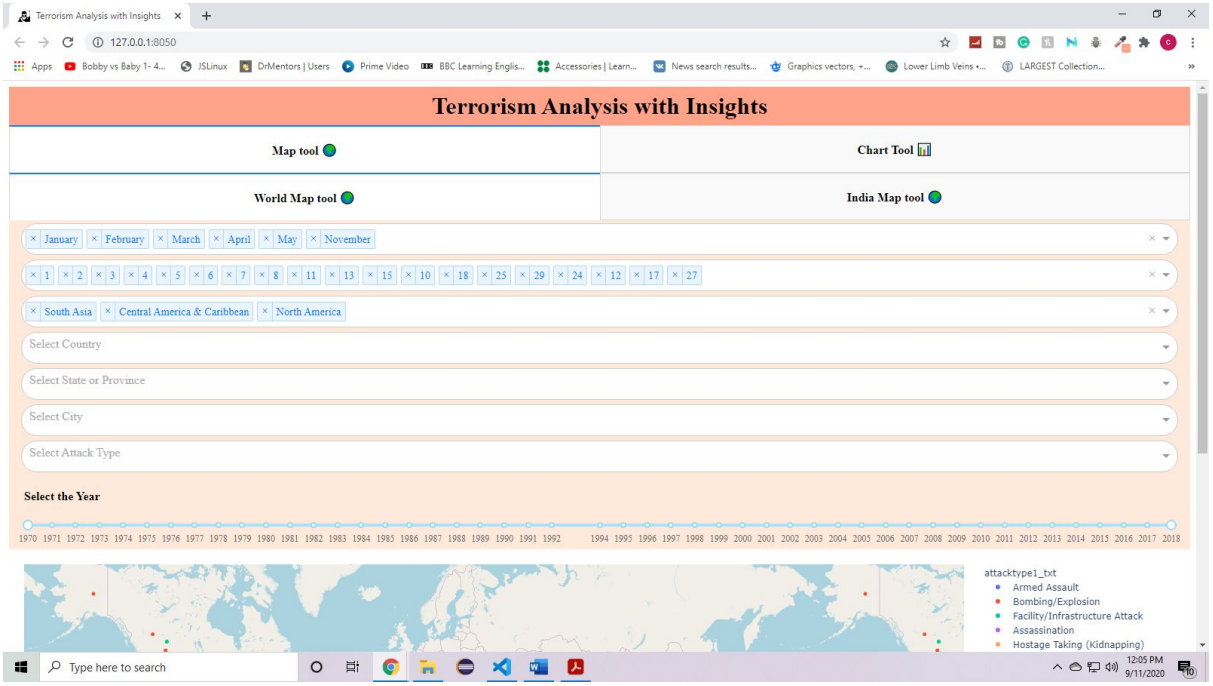

Fig 3.10 : Global Map Tool - Few regions are selected

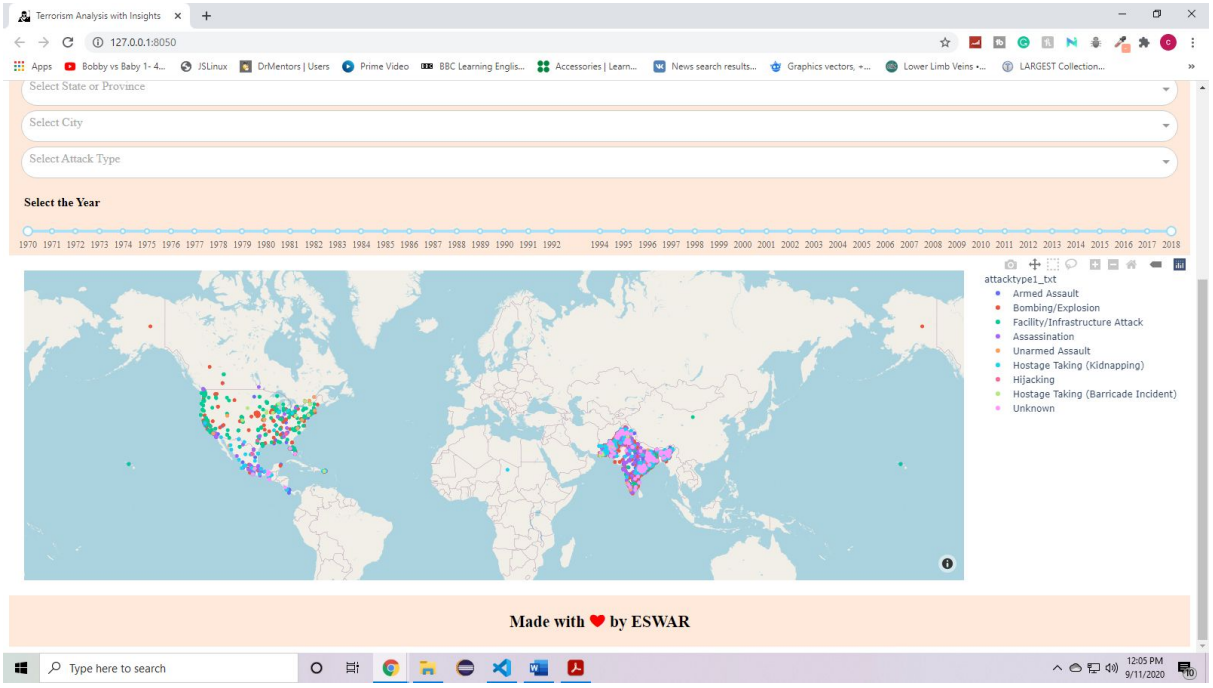

Fig 3.11 : Global Map Tool - The graph shows filtered data as per selected month, date and region.

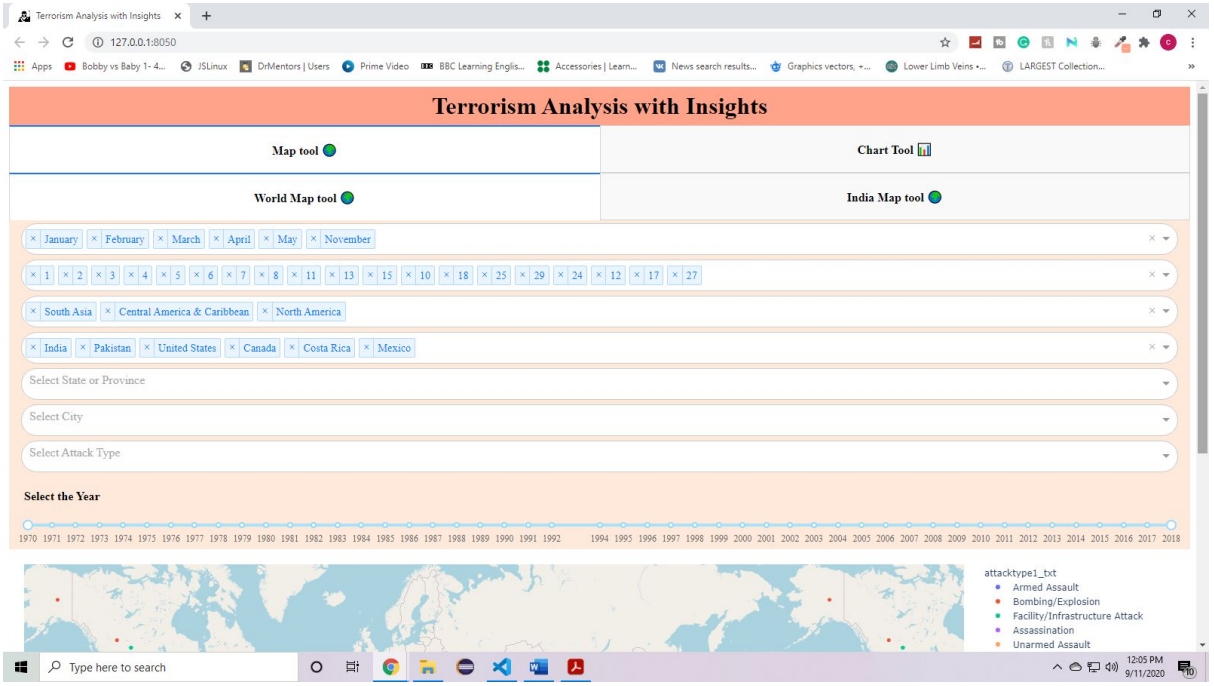

Fig 3.12 : Global Map Tool - Few countries are selected based on the selected regions

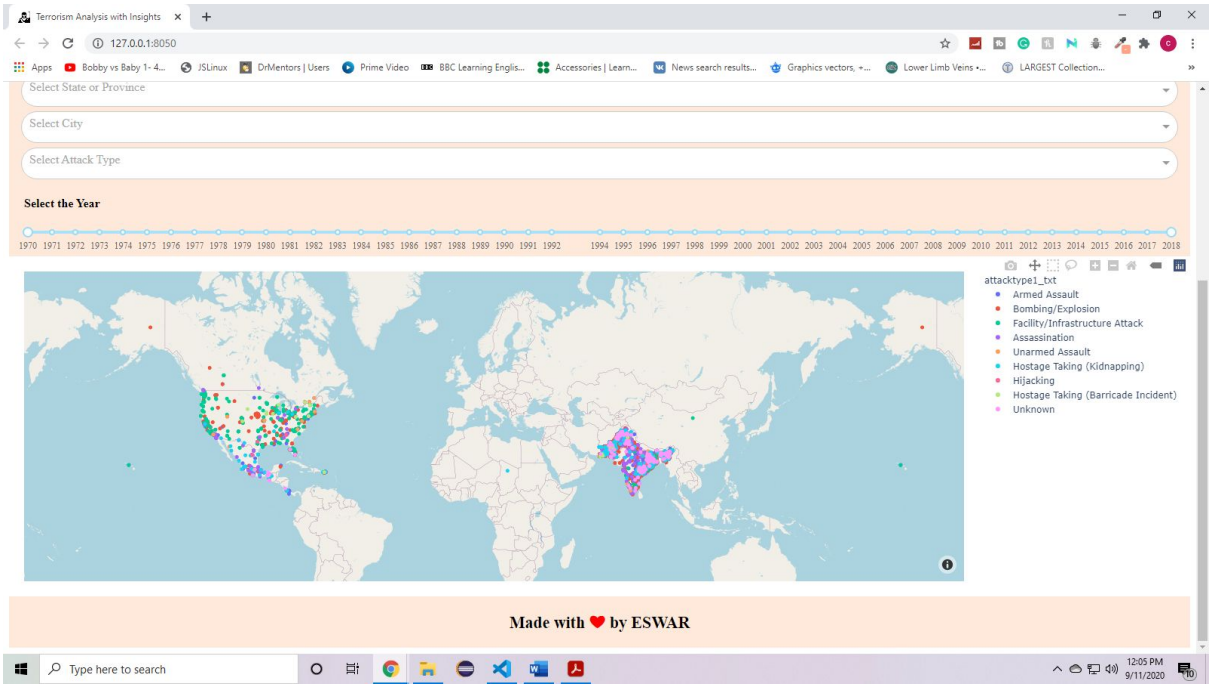

Fig 3.13 : Global Map Tool - the graph shows filtered data as per selected month, date, region and country

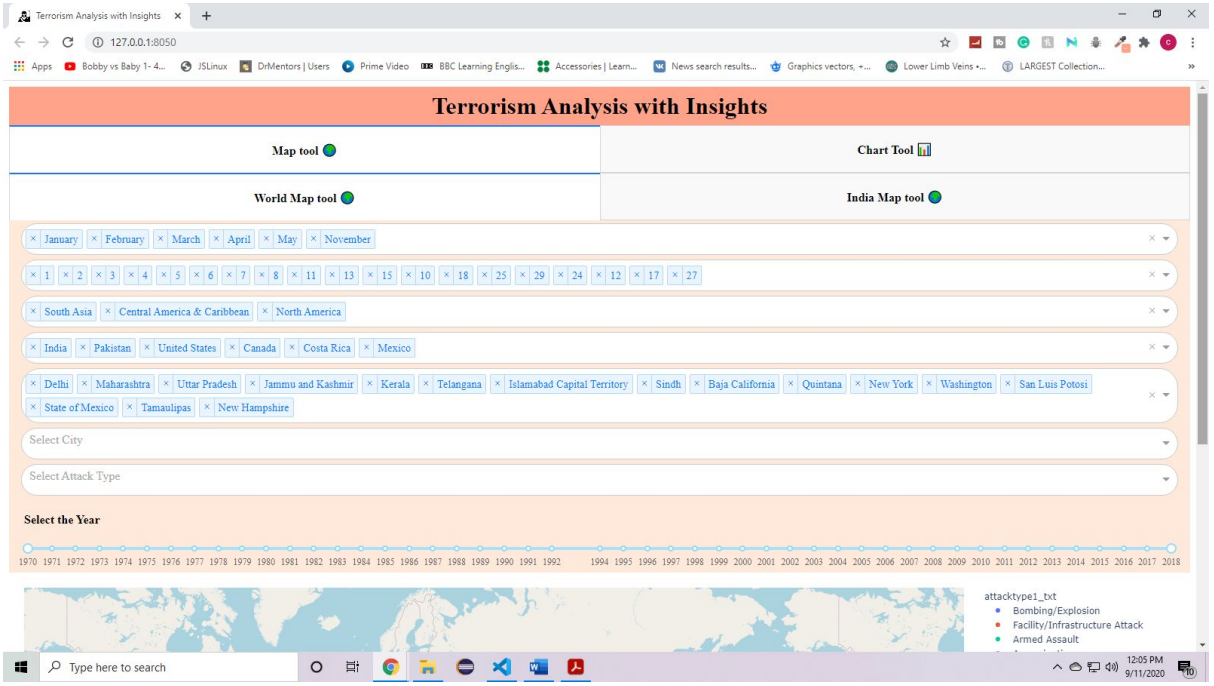

Fig 3.14 : Global Map Tool - Few states are selected based on the selected regions and countries

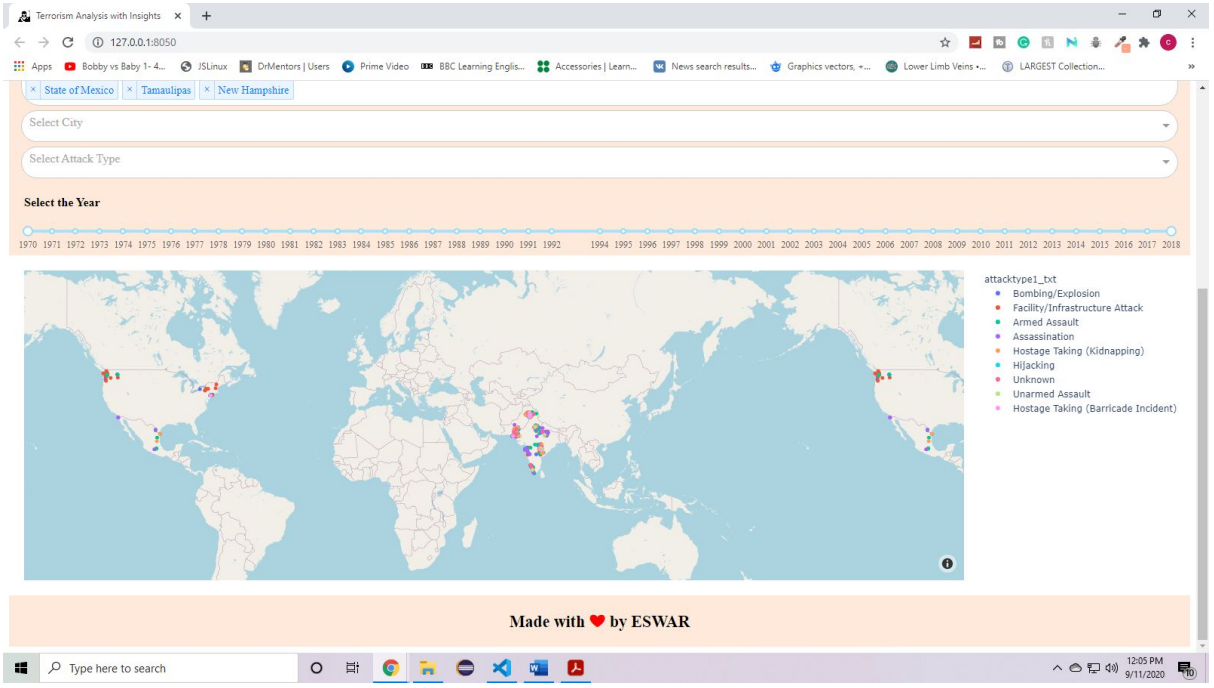

Fig 3.15 : Global Map Tool - The graph shows filtered data as per selected month, date, region, country and state.

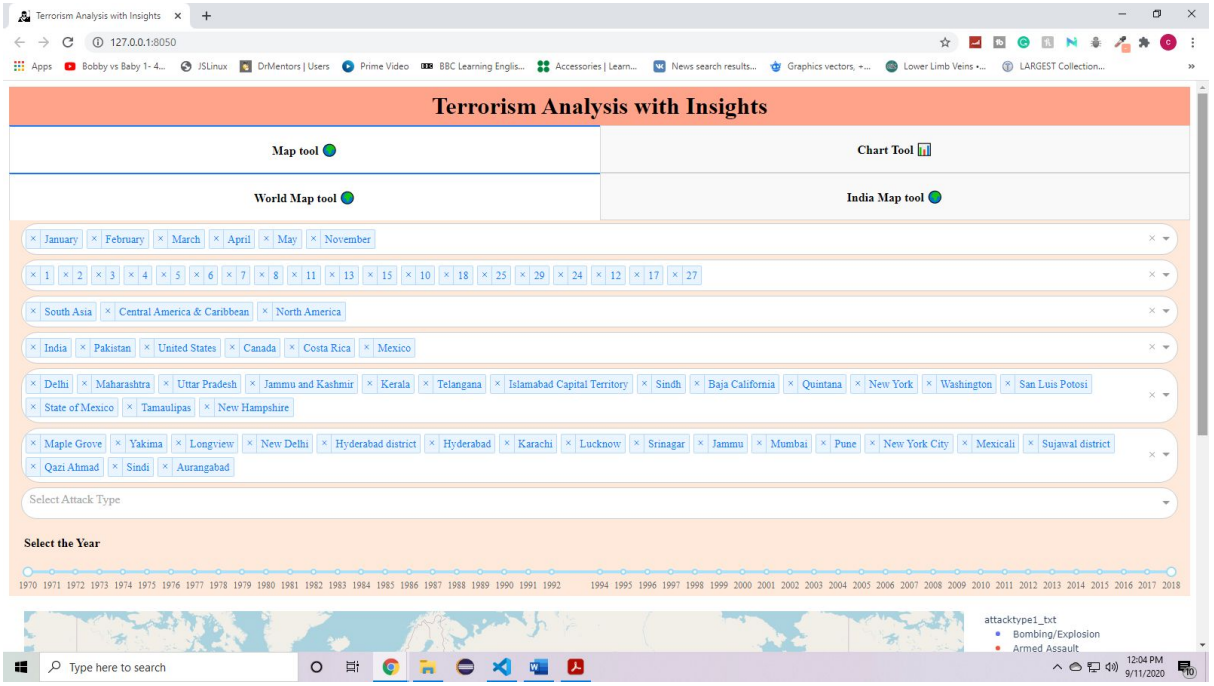

Fig 3.16 : Global Map Tool - Few cities are selected based on selected regions, countries and states

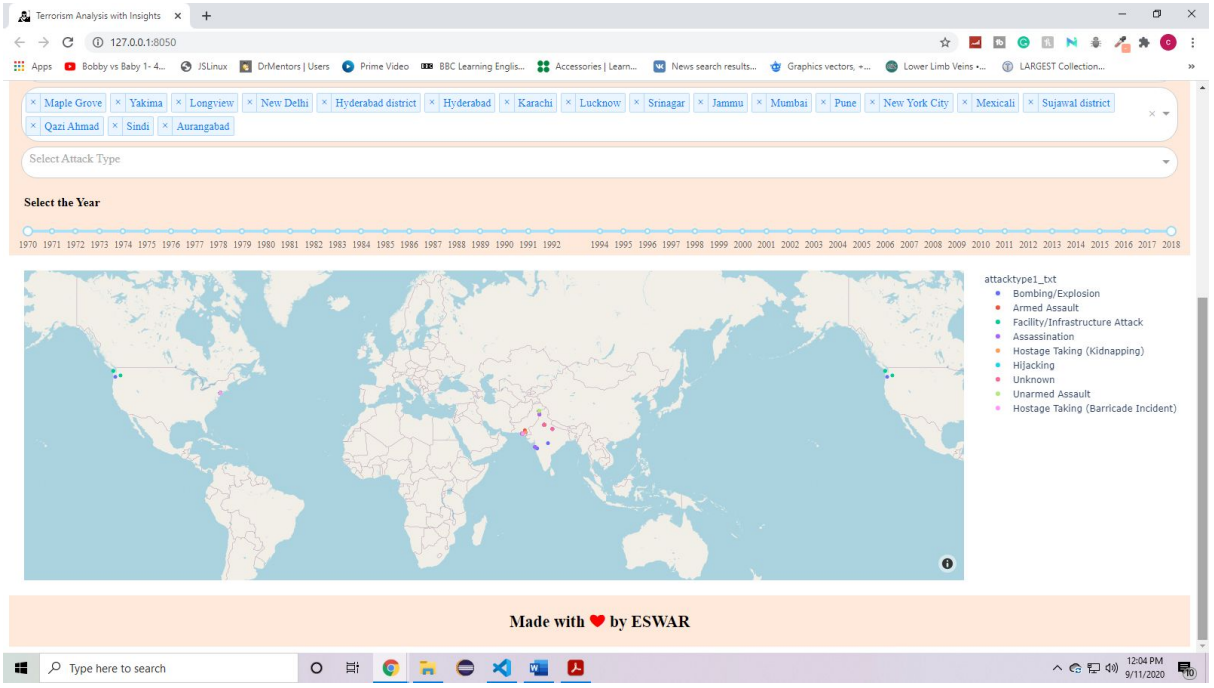

Fig 3.17 : Global Map Tool - The graph shows filtered data as per selected month, date, region, country, state and city

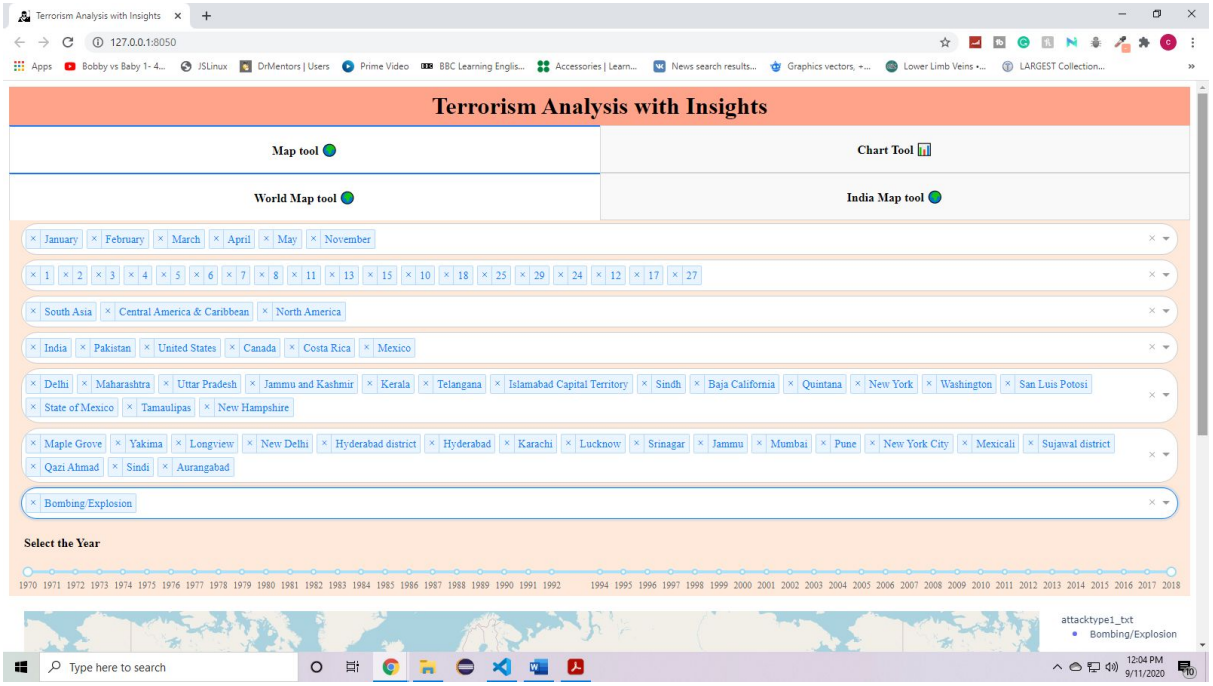

Fig 3.18 : Global Map Tool - An attack type is selected

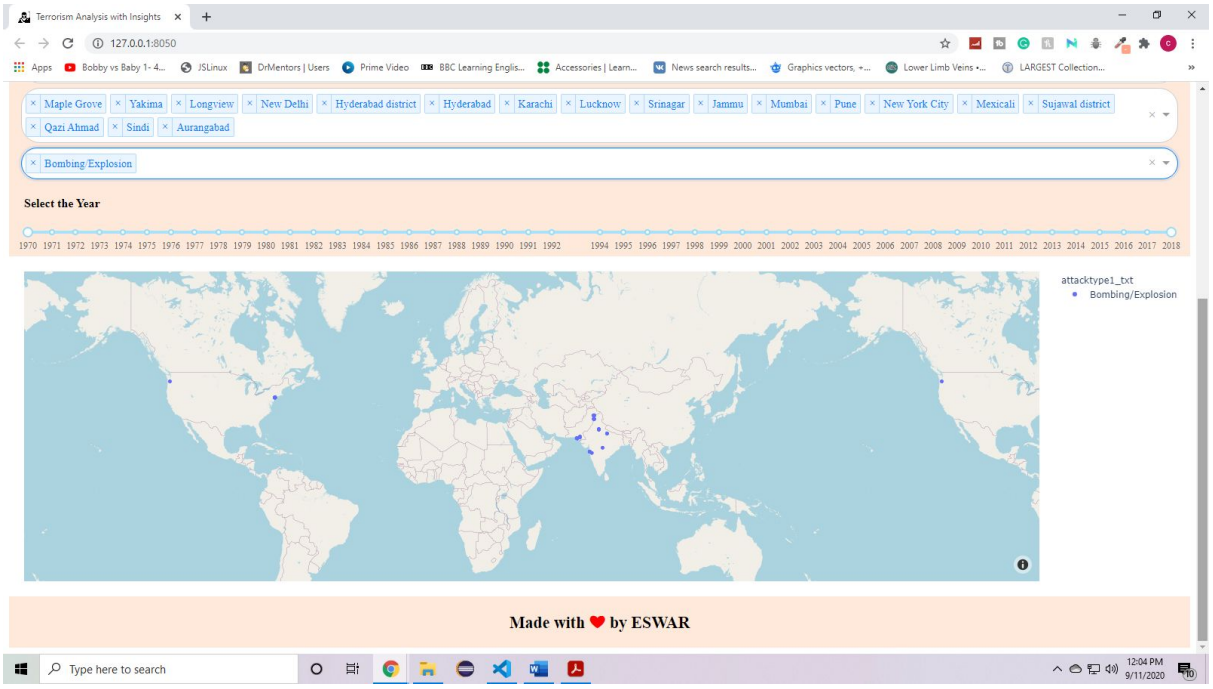

Fig 3.19 : Global Map Tool - The graph shows filtered data as per selected month, date, region, country, state, city and attack type.

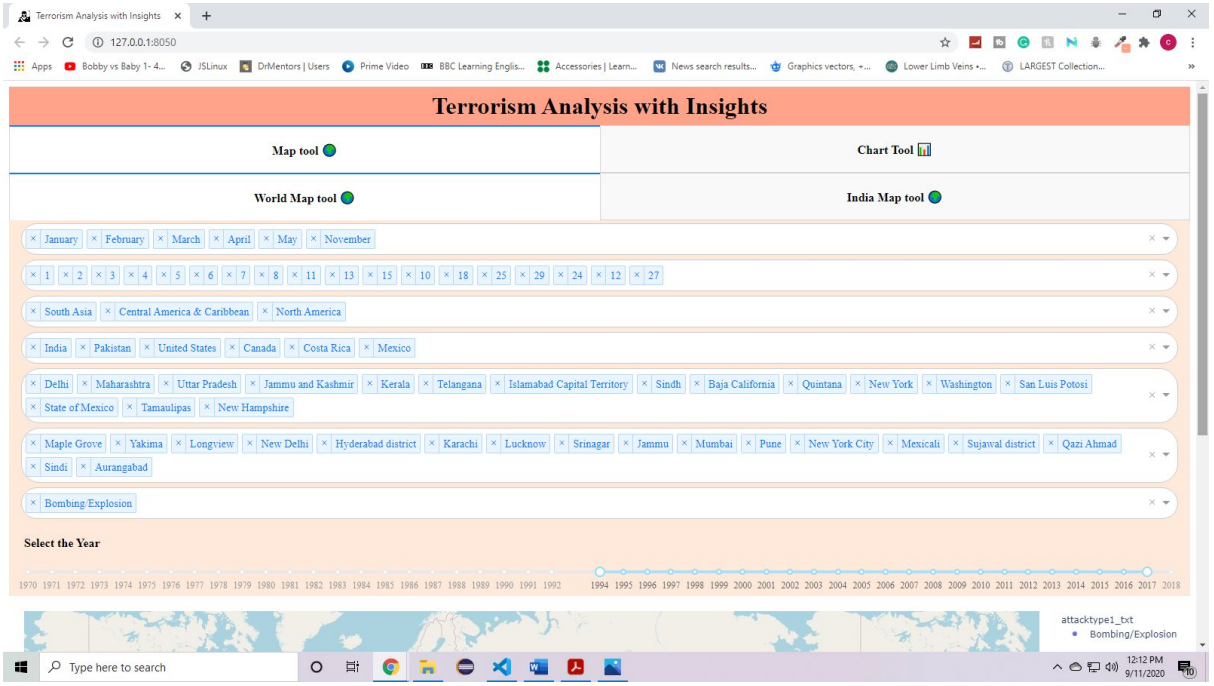

Fig 3.20 : Global Map Tool - Range of years are selected.

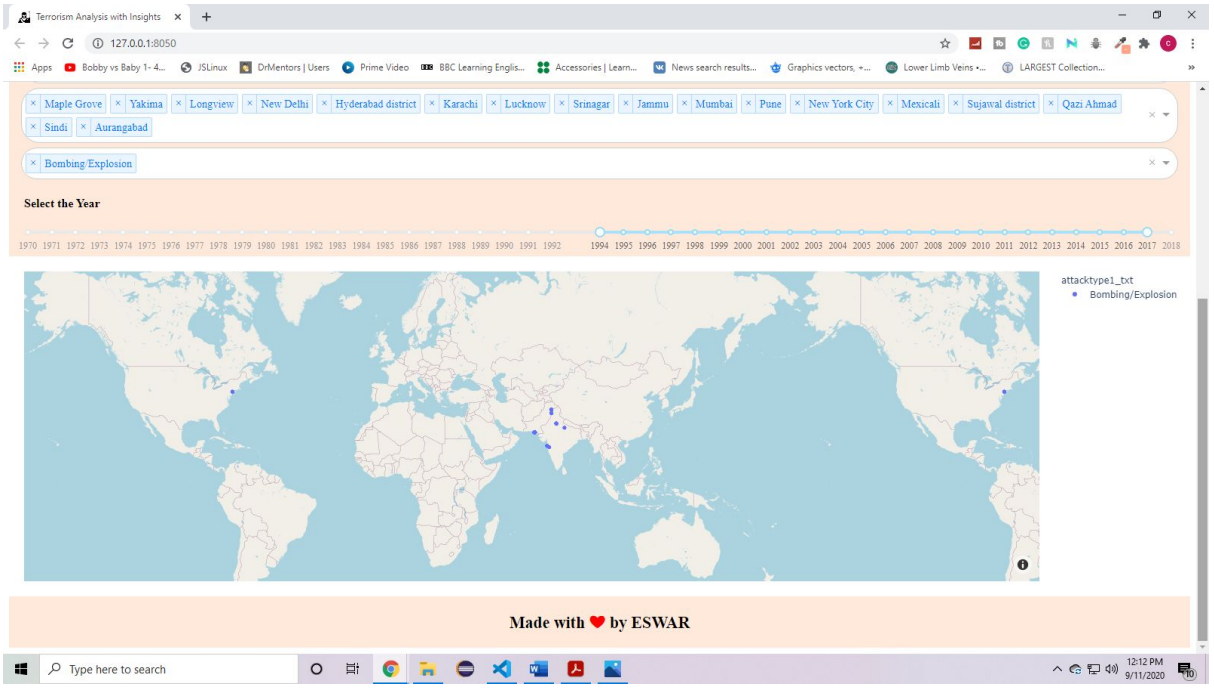

Fig 3.21 : Global Map Tool - The graph shows filtered data as per selected month, date, region, country, state, city, attack type and years.

Fig 3.22 .: India Map Tool - Application depicting without any inputs in dropdown and range slider

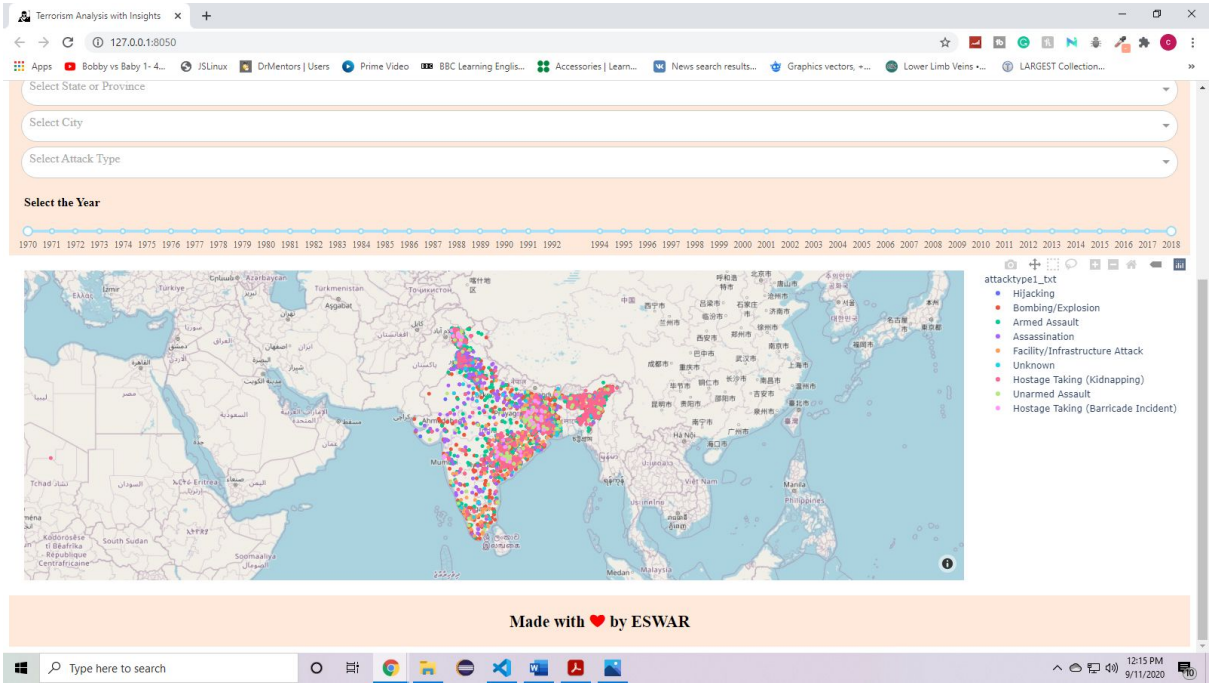

Fig 3.23 : India Map Tool - The whole data of india is shown without any filters

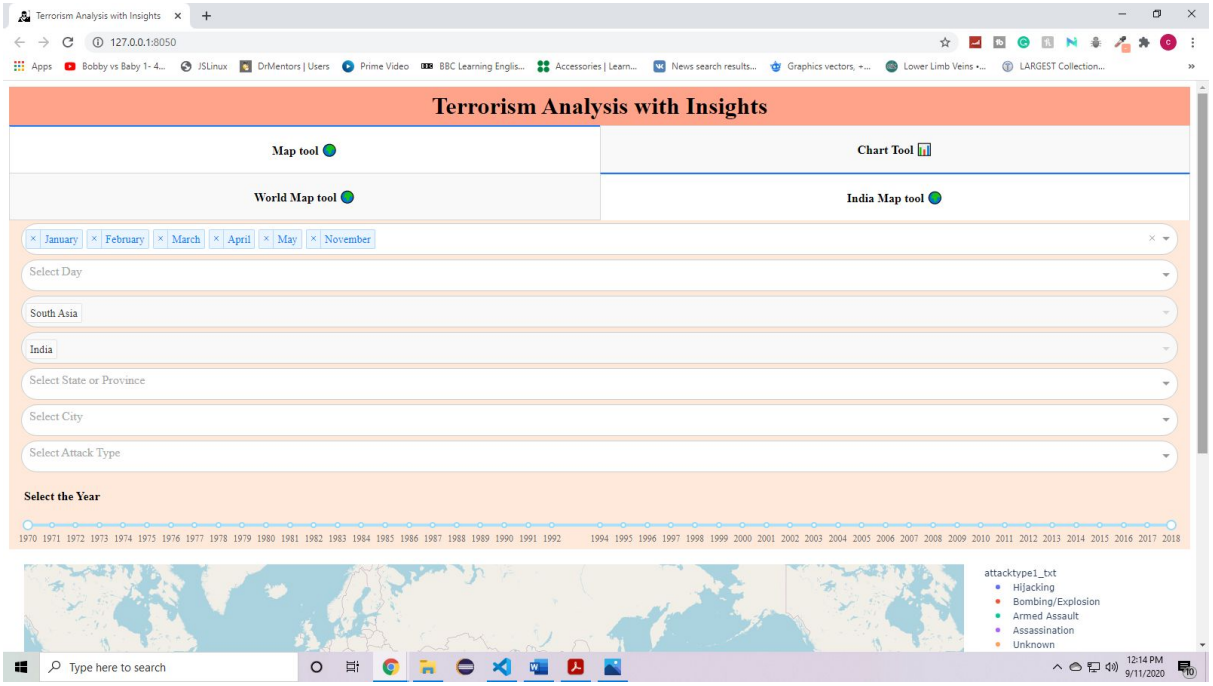

Fig 3.24 : India Map Tool - Few months are selected as per the user requirements through dropdown

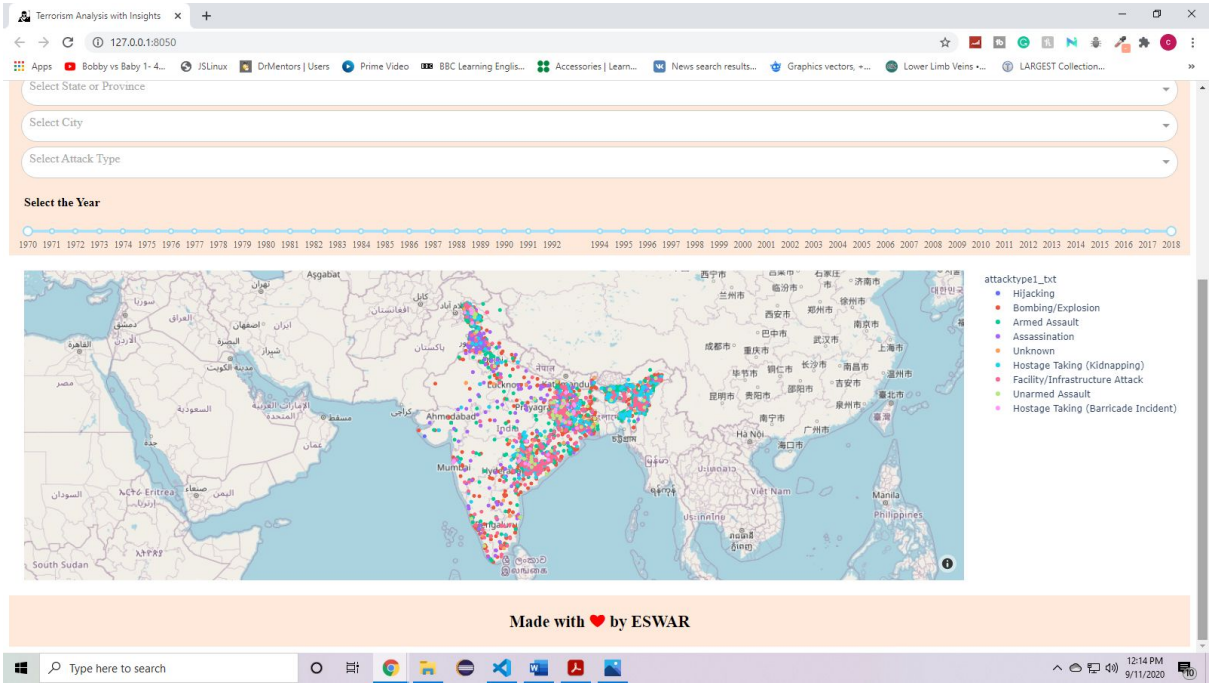

Fig 3.25 : India Map Tool - The graph shown filtered data as per selected months

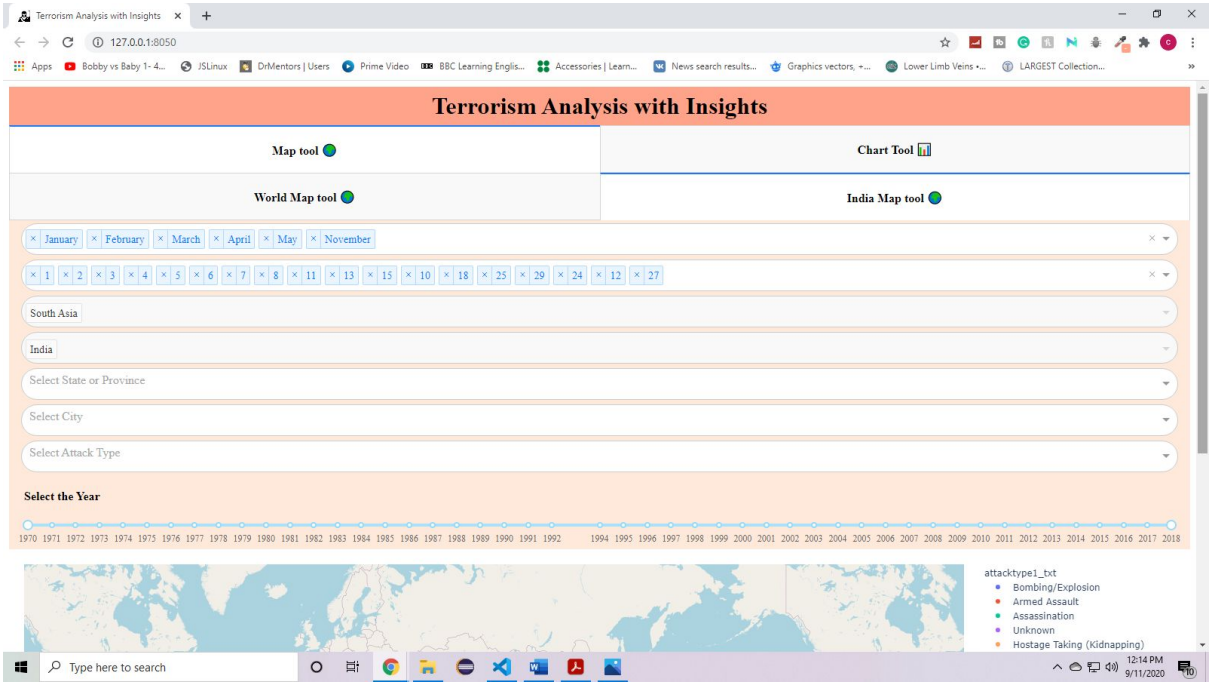

Fig 3.26 : India Map Tool - Few dates are selected as per selected months

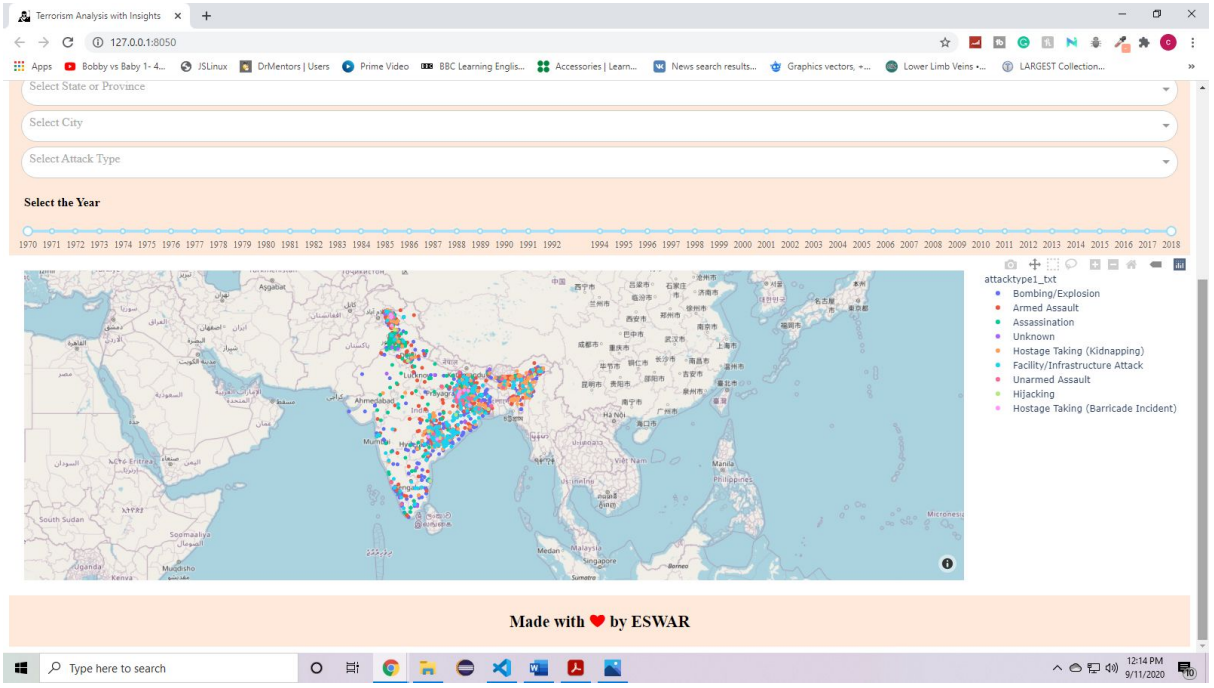

Fig 3.27 : India Map Tool - The graph shown filtered data as per selected month and date

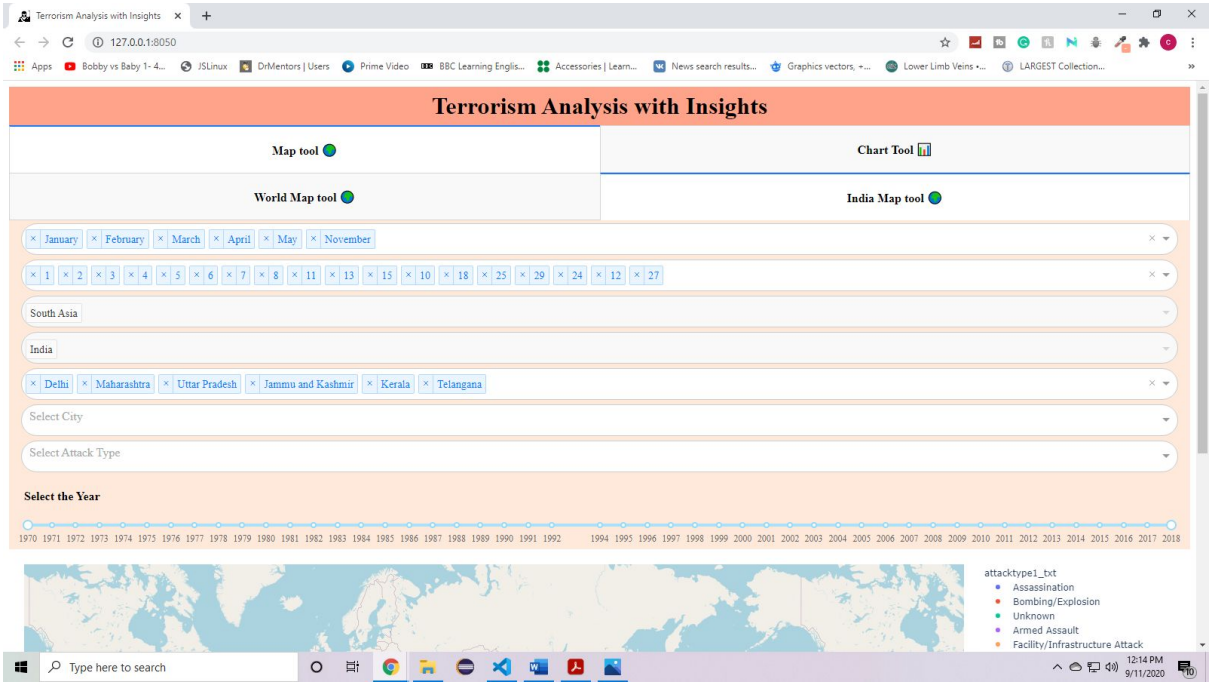

Fig 3.28 : India Map Tool - Few states from india are selected .

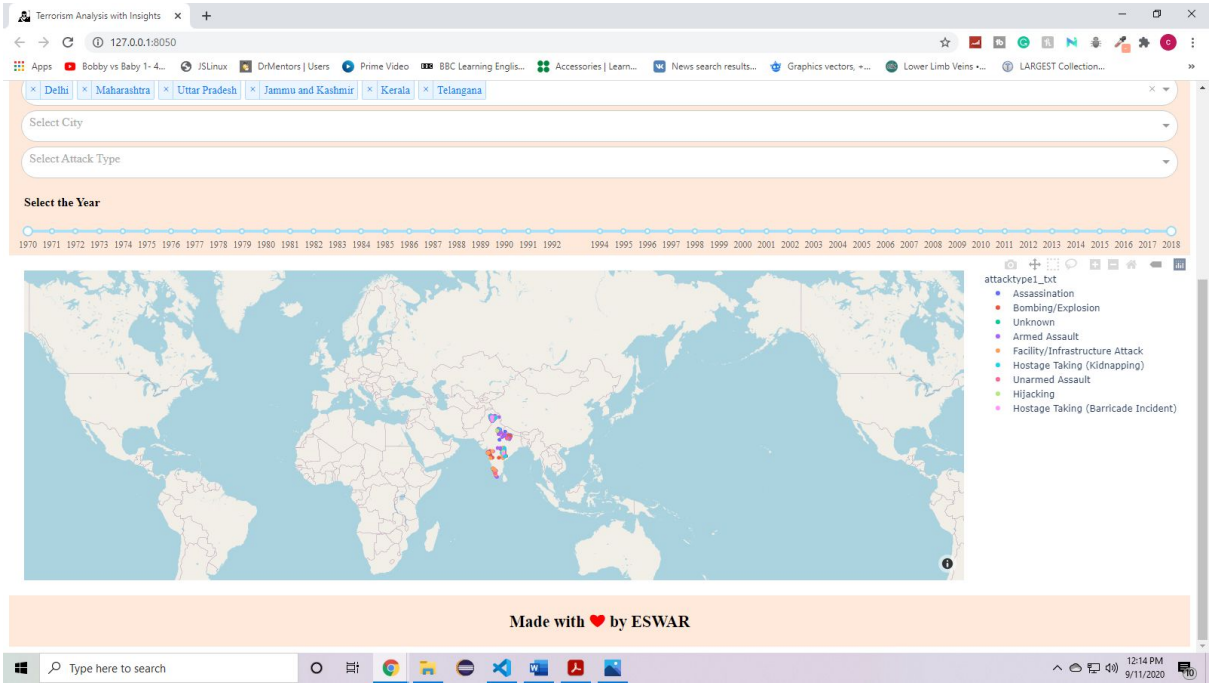

Fig 3.29: the graph shows filtered data as per selected month, date and state in india

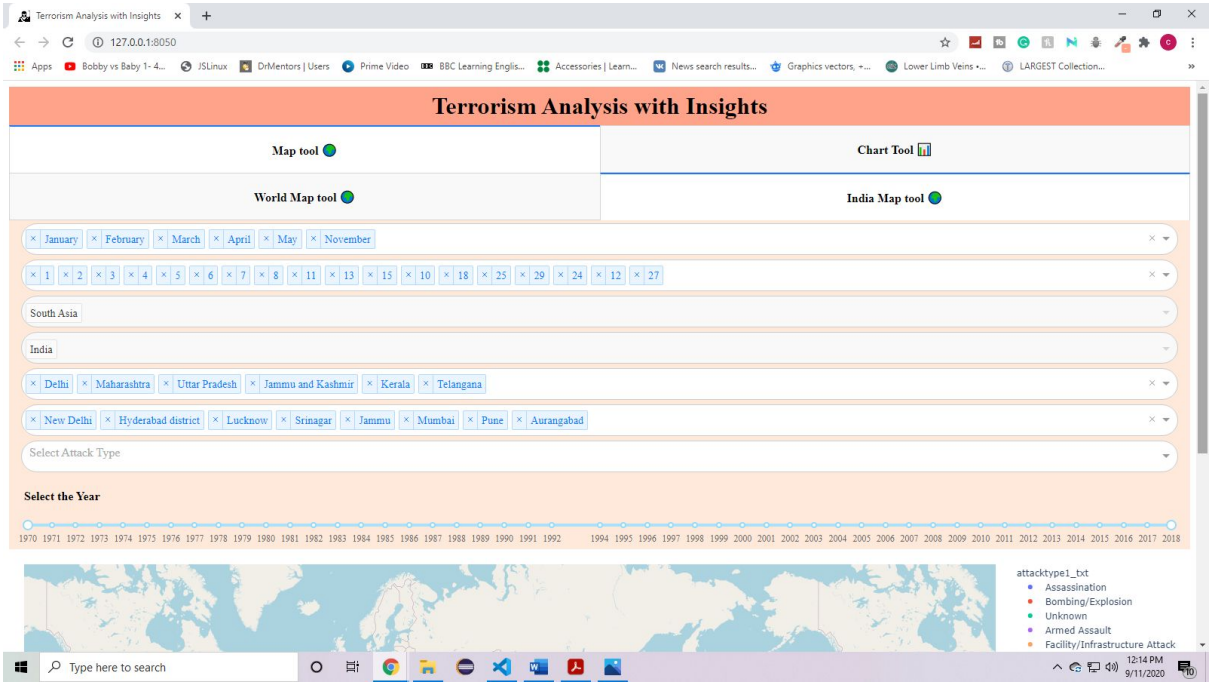

Fig 3.30: India Map Tool - Few cities are selected based on selected states and in india

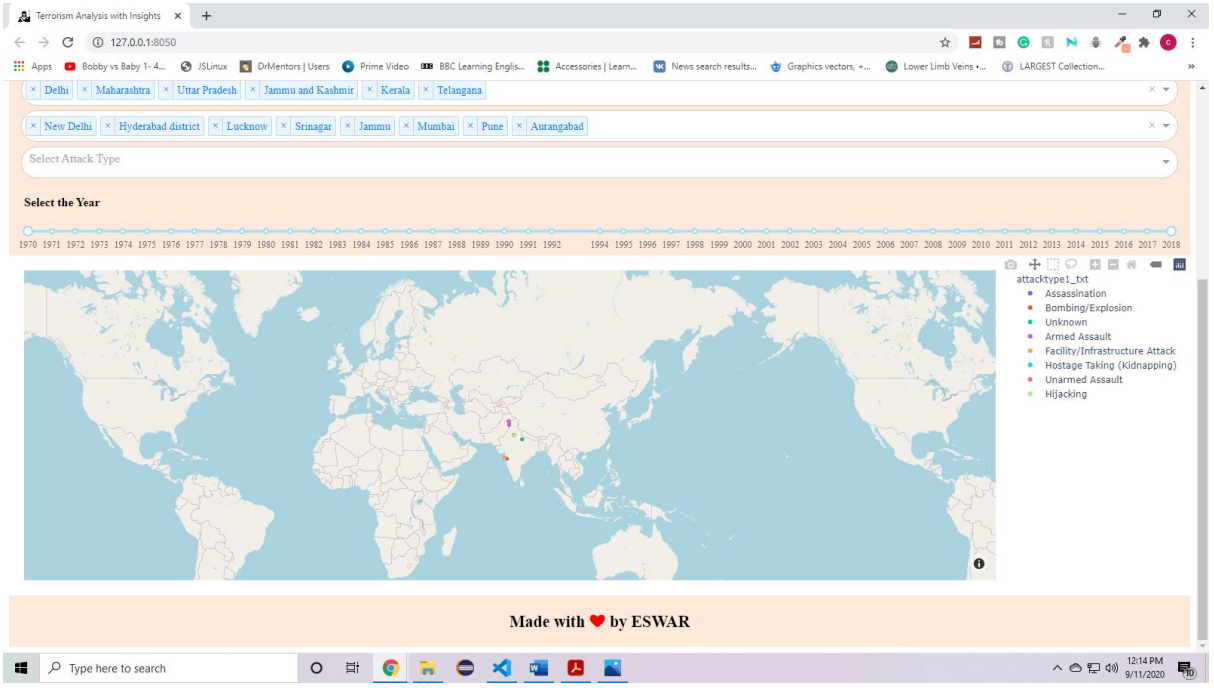

Fig 3.31 : India Map Tool - The graph shows filtered data as per selected month, date, states and few cities in india

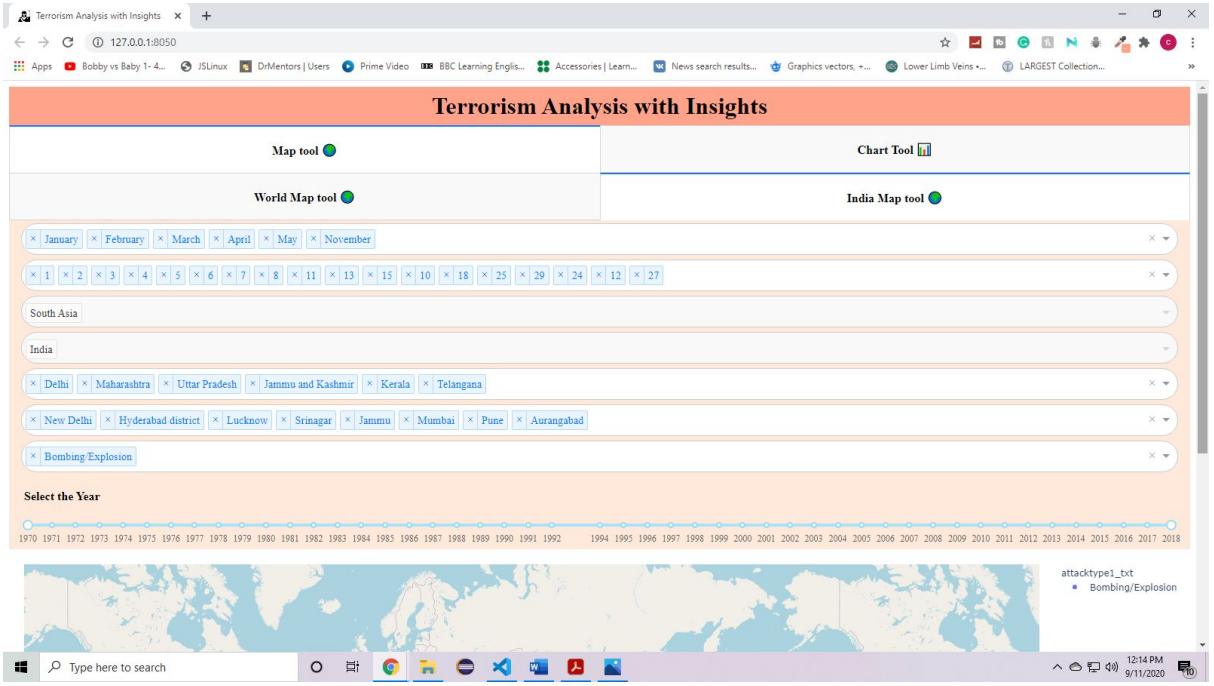

Fig 3.32 : India Map Tool - An attack type is selected.

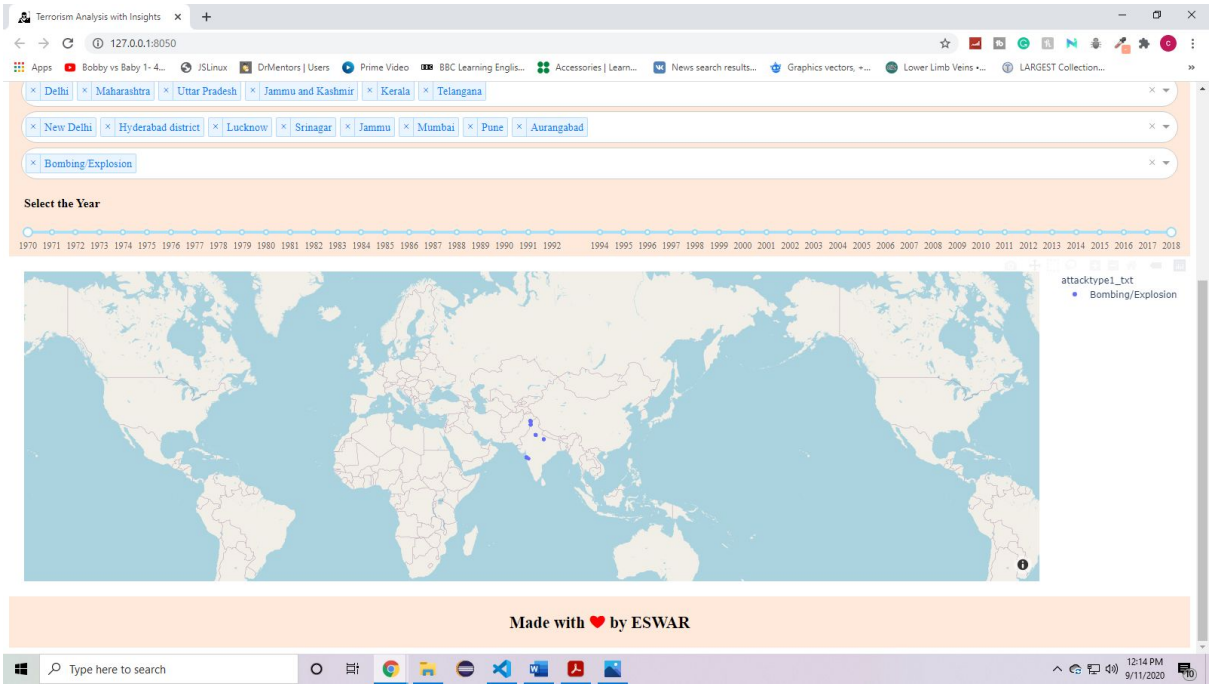

Fig 3.33: India Map Tool - The graph shows filtered data as per selected month, date, state, city in india and attack type

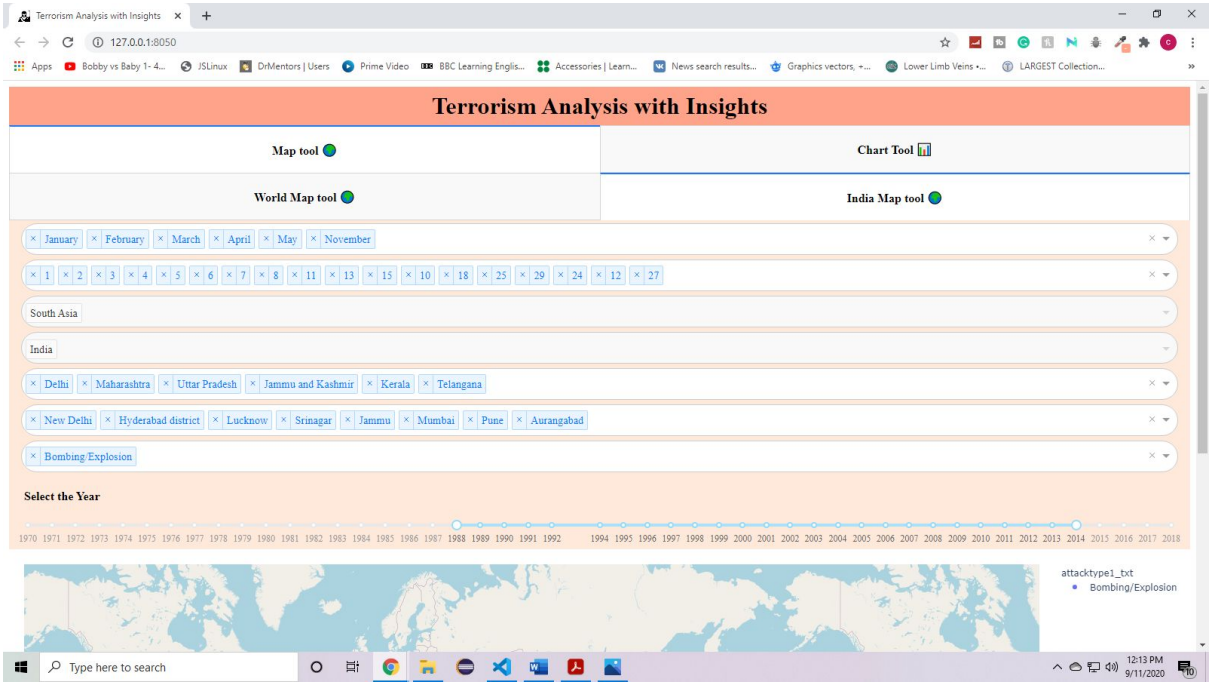

Fig 3.34 : India Map Tool - A range of year is selected

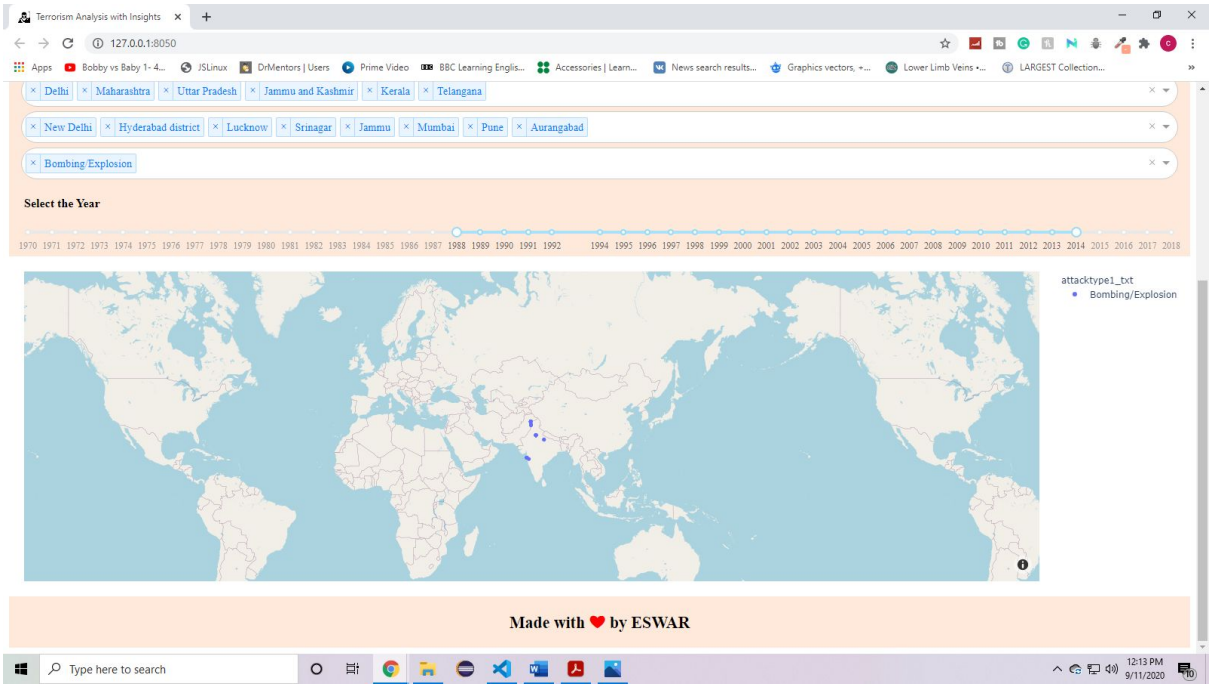

Fig 3.35 : India Map Tool - The graph shows filtered data as per selected month, date, year, state and city in india and attack type.

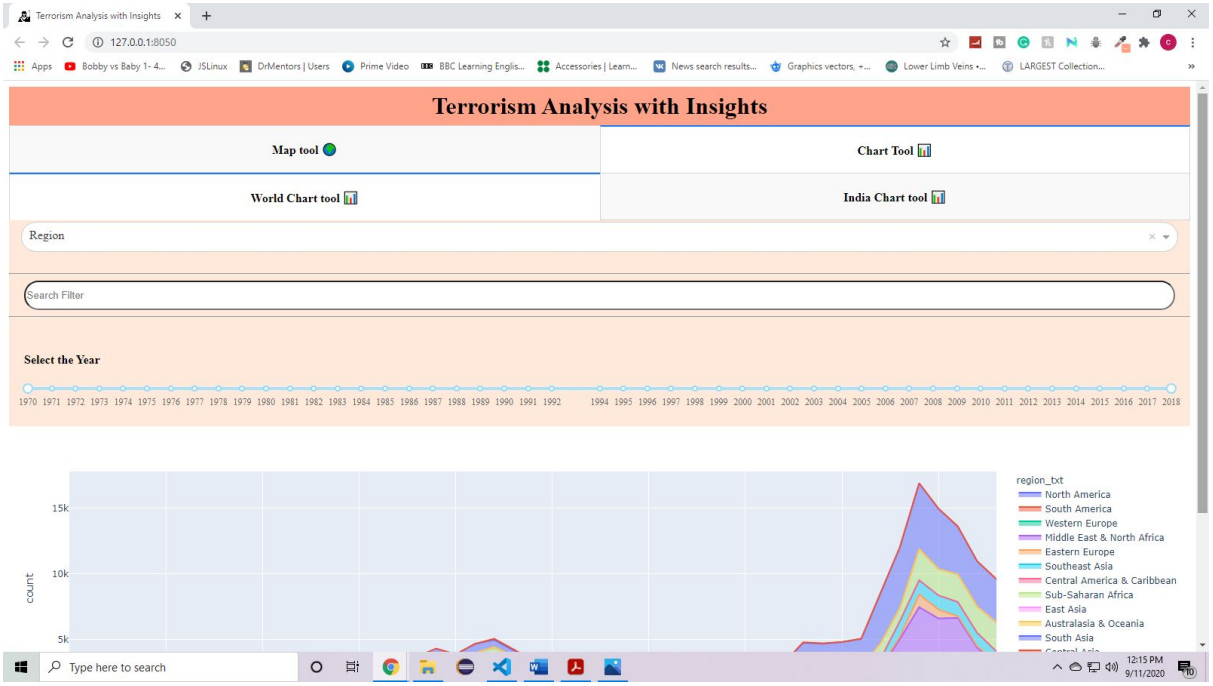

Fig 3.36 : Global Chart Tool - application - chart tool depicting without any inputs (region selected as default) in dropdown and range slider

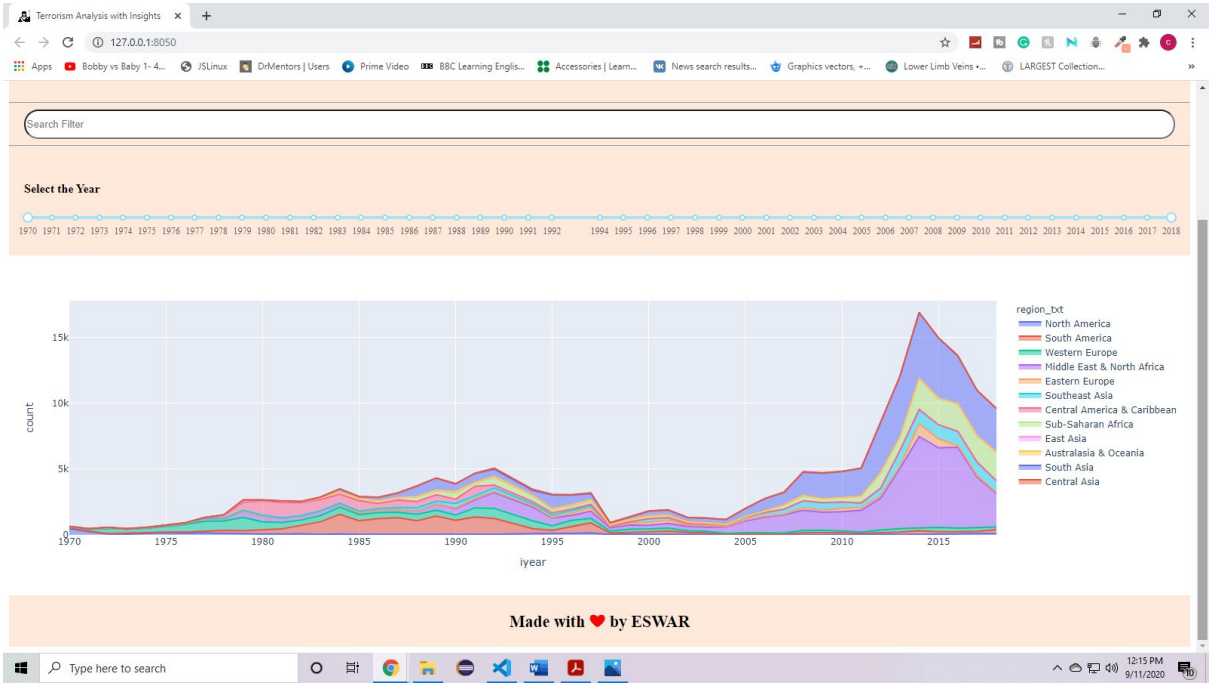

Fig 3.37 : Global Chart Tool - The whole data shows without any filters

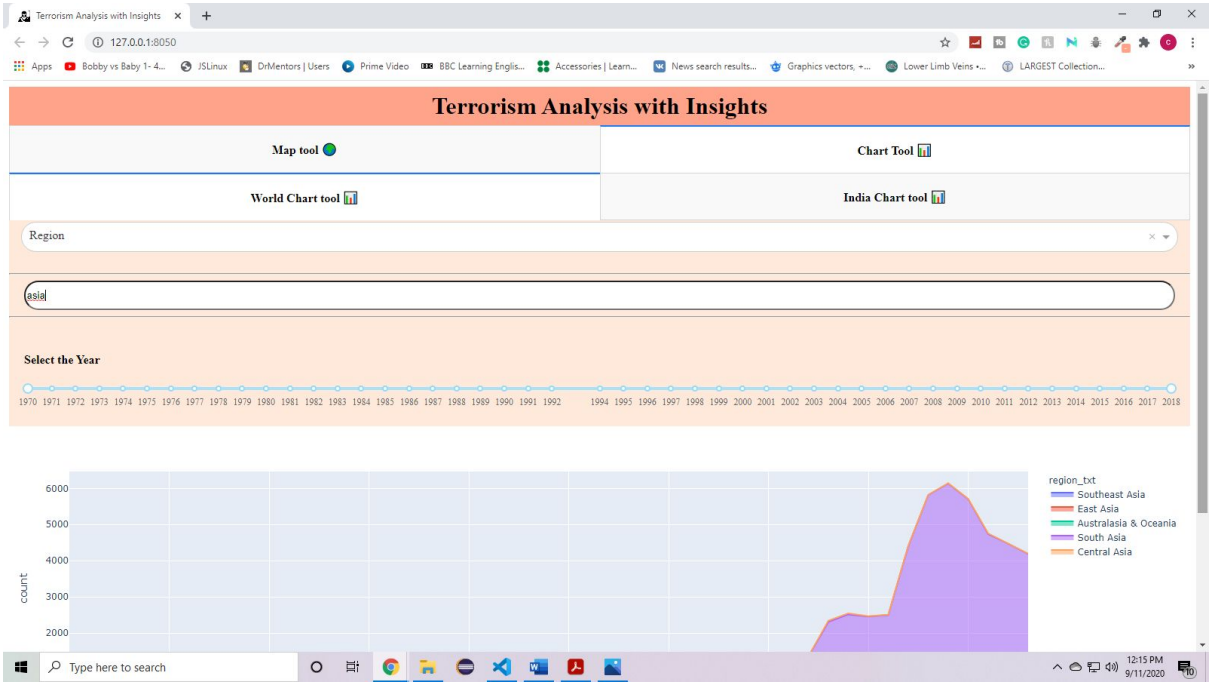

Fig 3.38 : Global Chart Tool - Search filter is selected as per user requirement

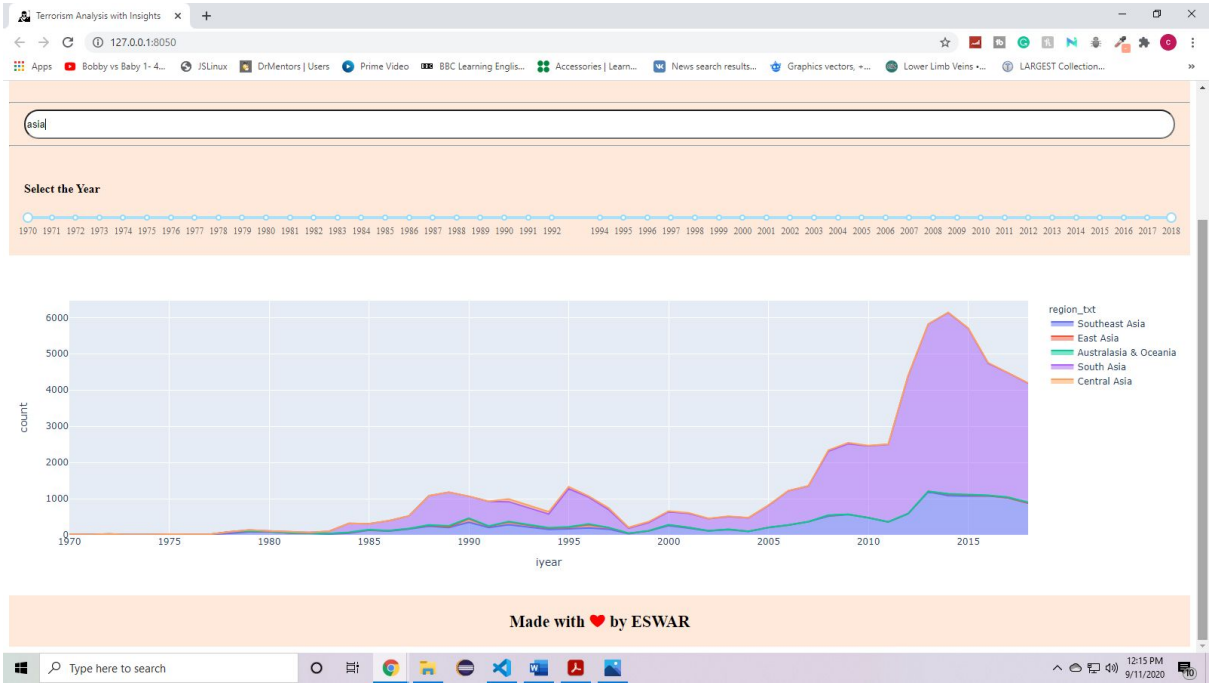

Fig 3.39 : Global Chart Tool - the chart shows filtered data as per search filter

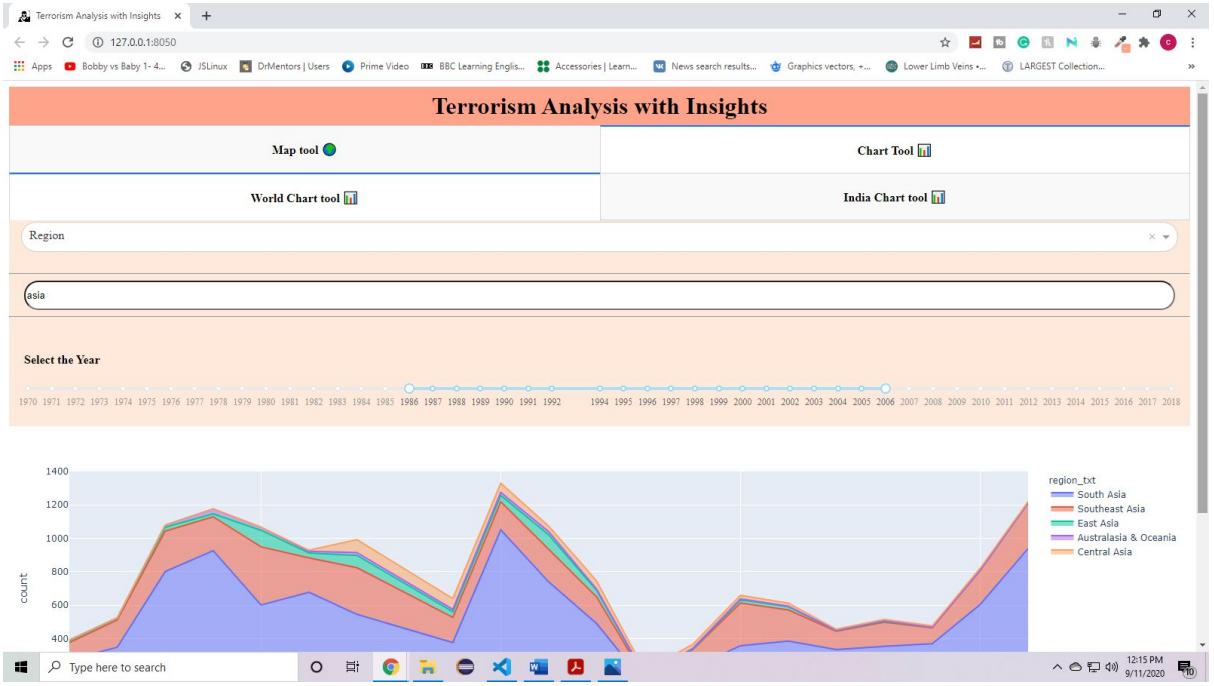

Fig 3.40: Global Chart Tool - Range of year slider is selected as per the user requirement

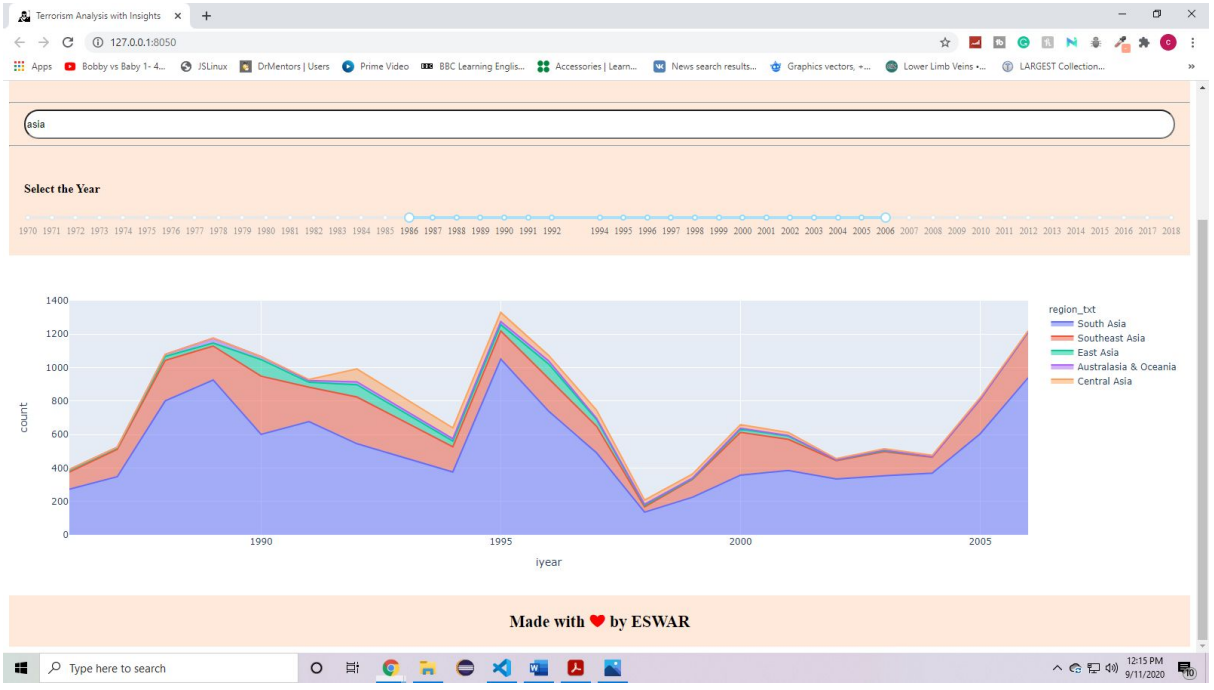

Fig 3.41: Global Chart Tool - the graph shows filtered data as per search filter and year slider

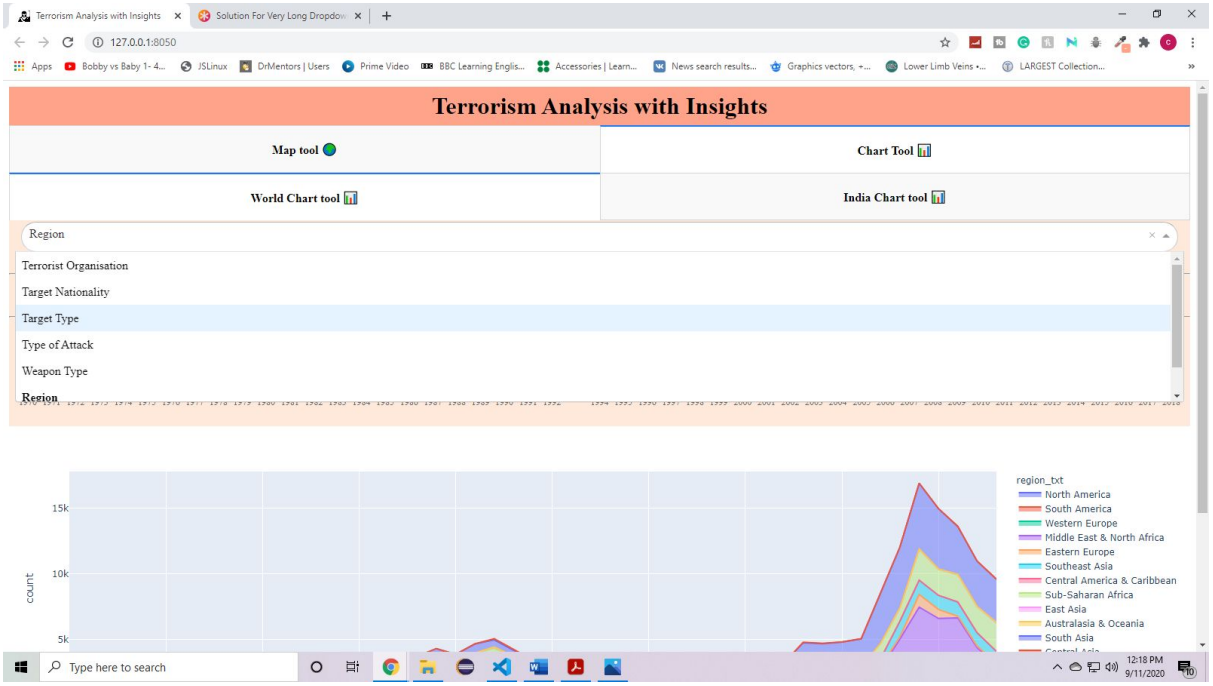

Fig 3.42 : Global Chart Tool - application depicting dropdown (part 1)

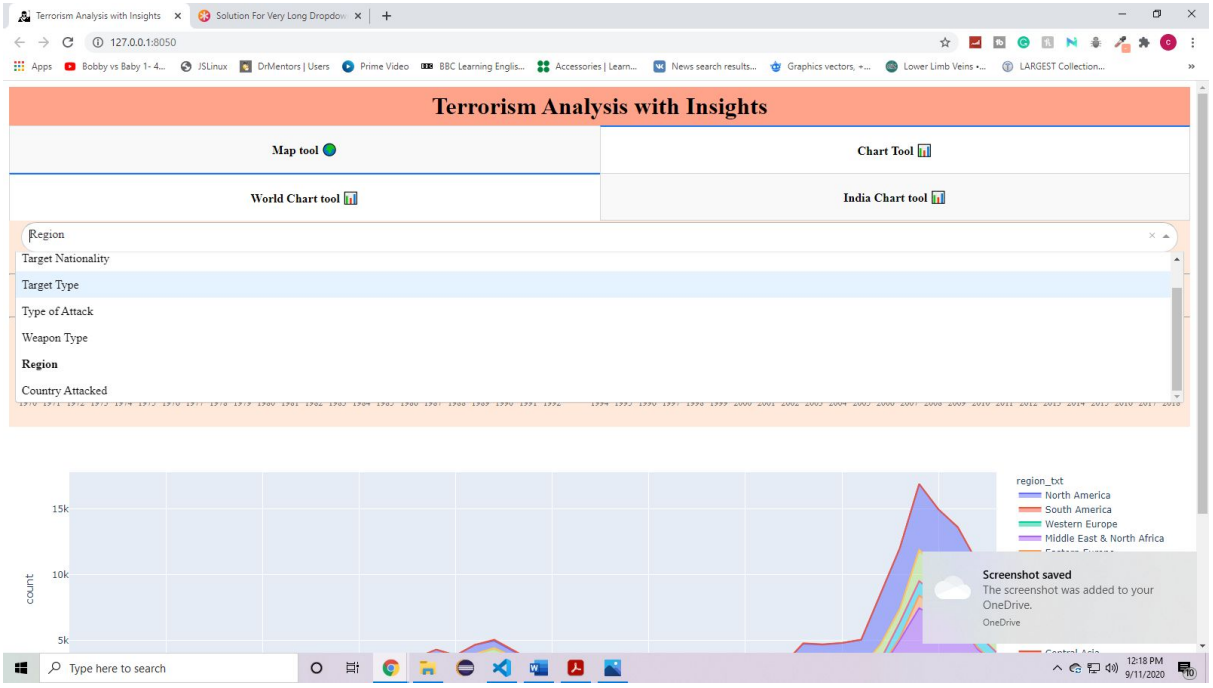

Fig 3.43 : Global Chart Tool - Application depicting dropdown (part 2)

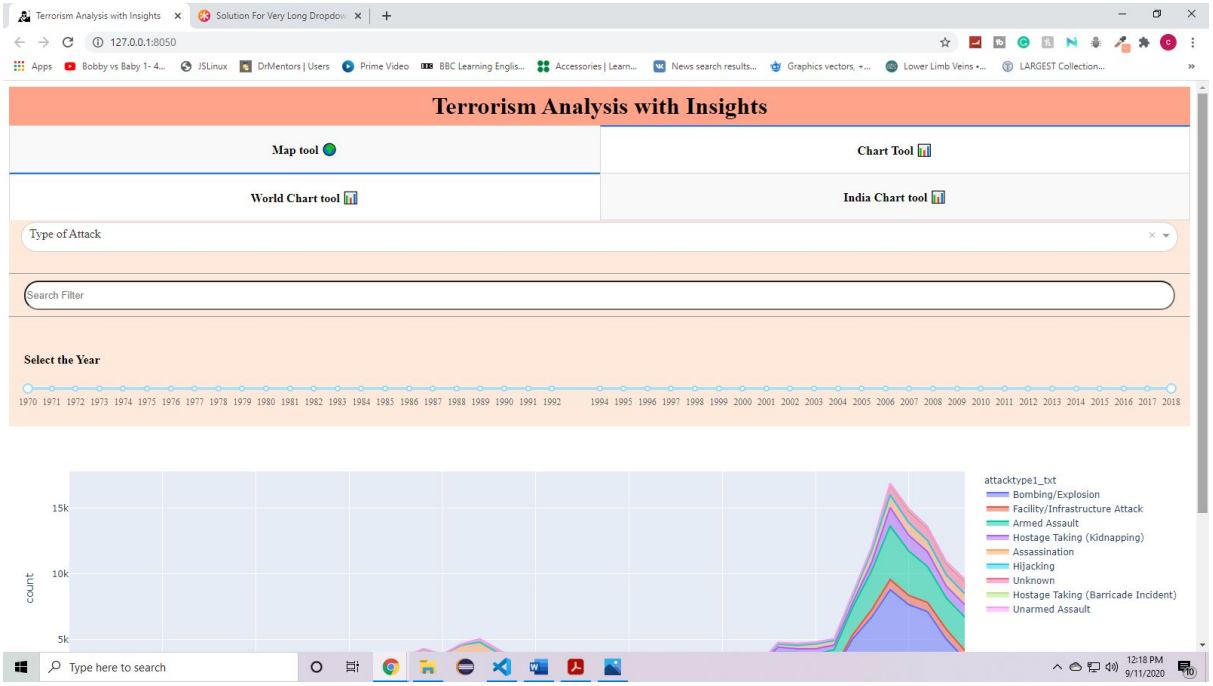

Fig 3.44 : Global Chart Tool - Type of attack is selected as per user requirements

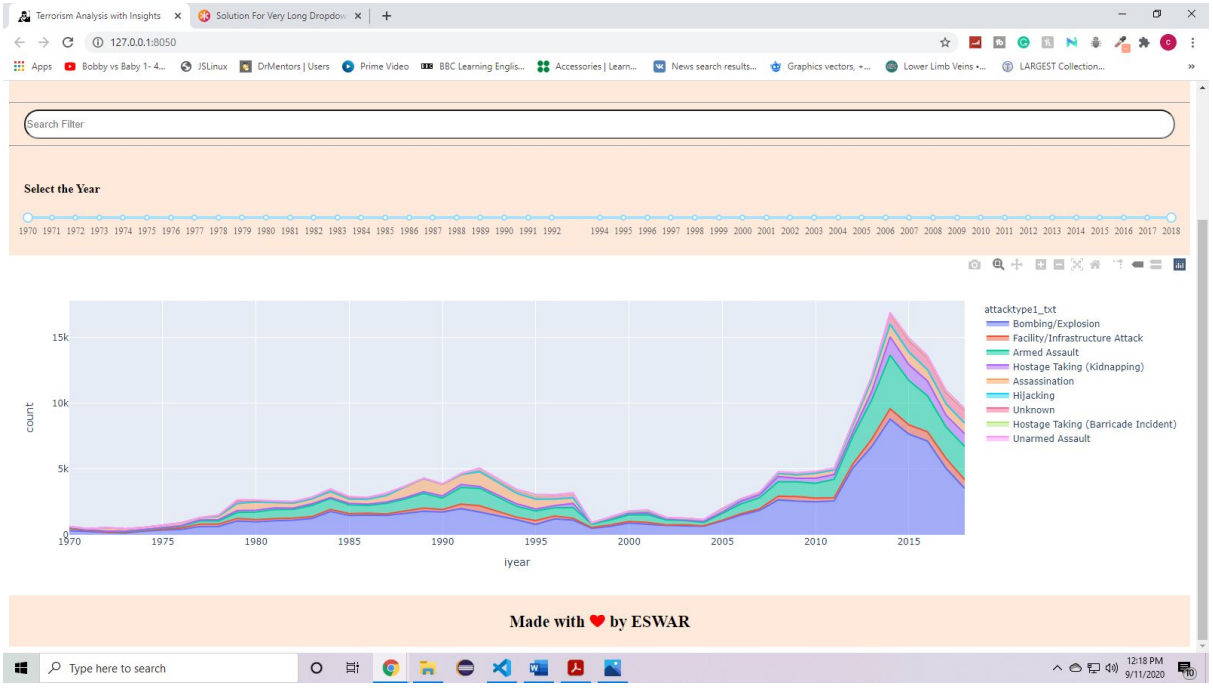

Fig 3.45: Global Chart Tool - The graph shows filtered data as per type of attack

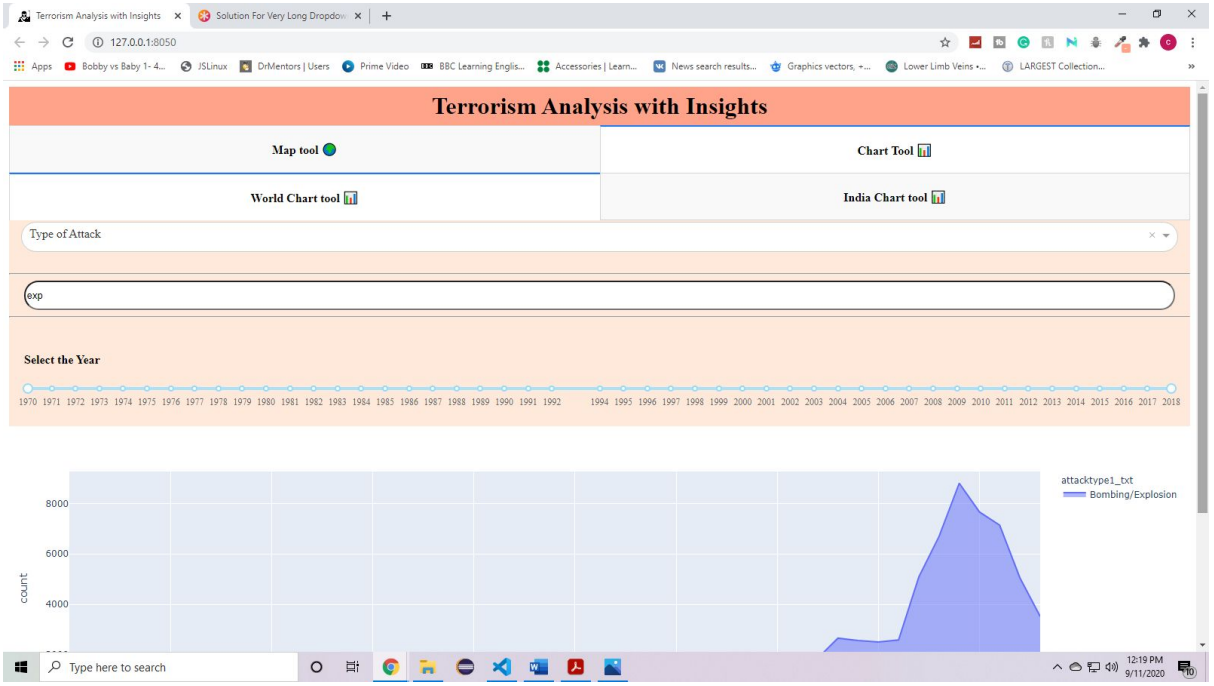

Fig 3.46 : Global Chart Tool -Search filter is selected as per user requirement (exp)

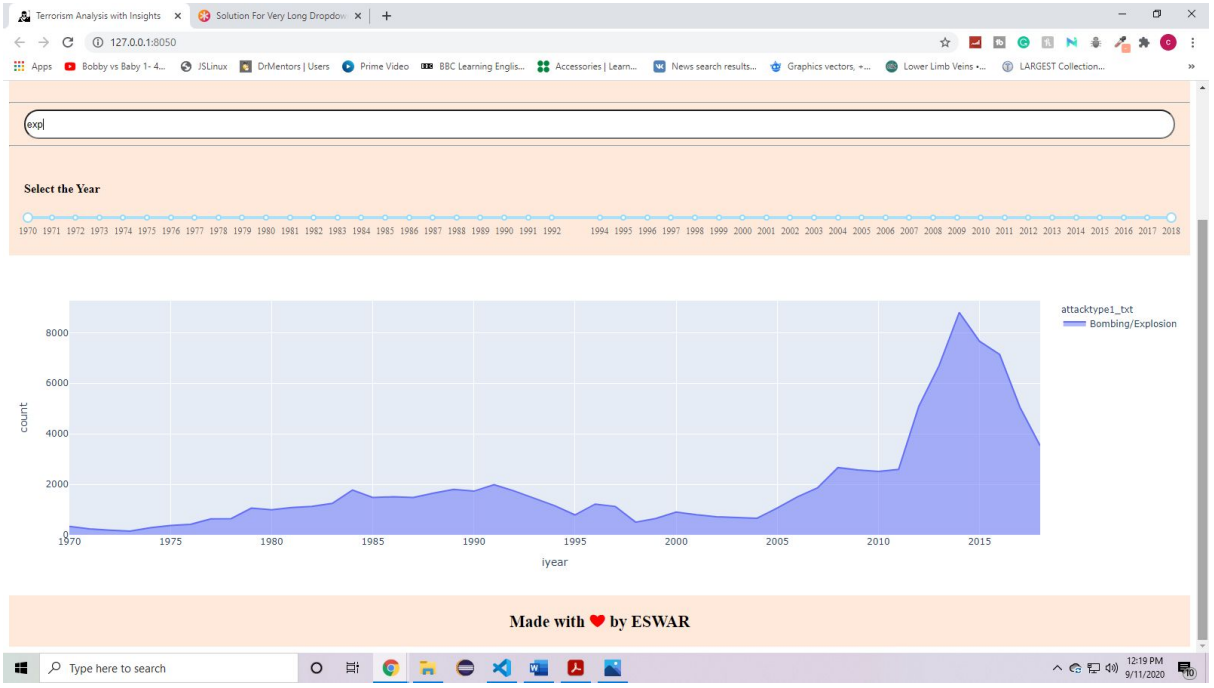

Fig 3.47 : Global Chart Tool - The graph shows filtered data as per search filter and type of attack

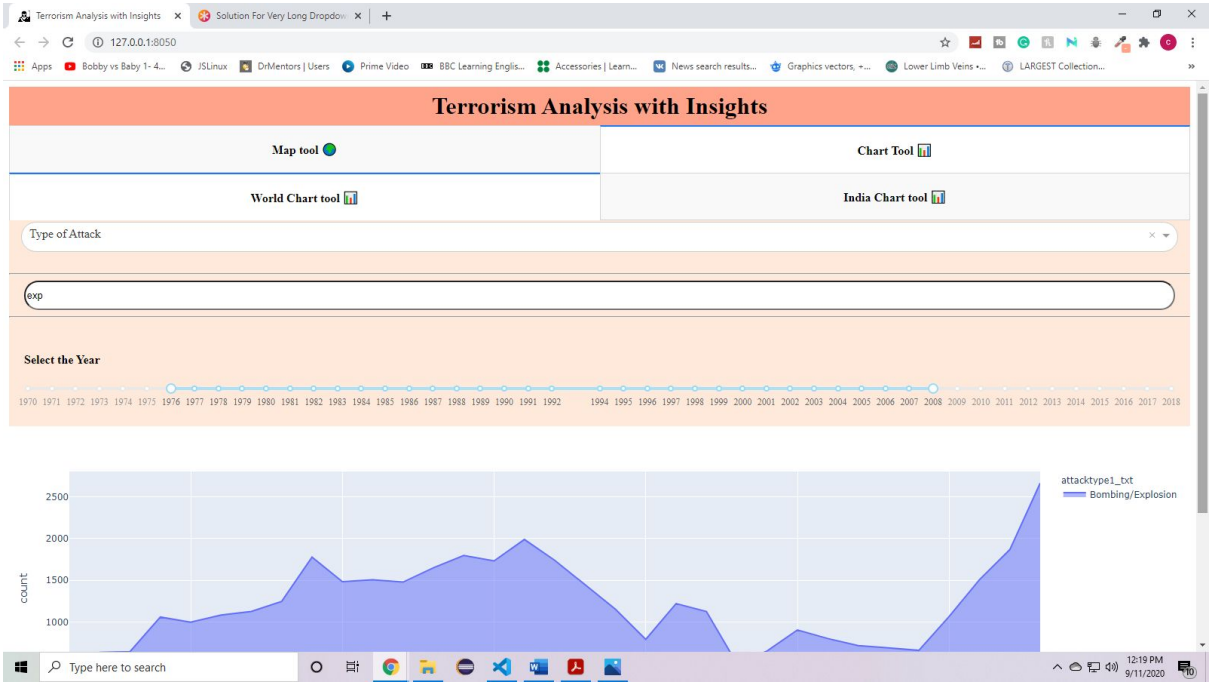

Fig 3.48: Global Chart Tool - A range of year is selected using range slider

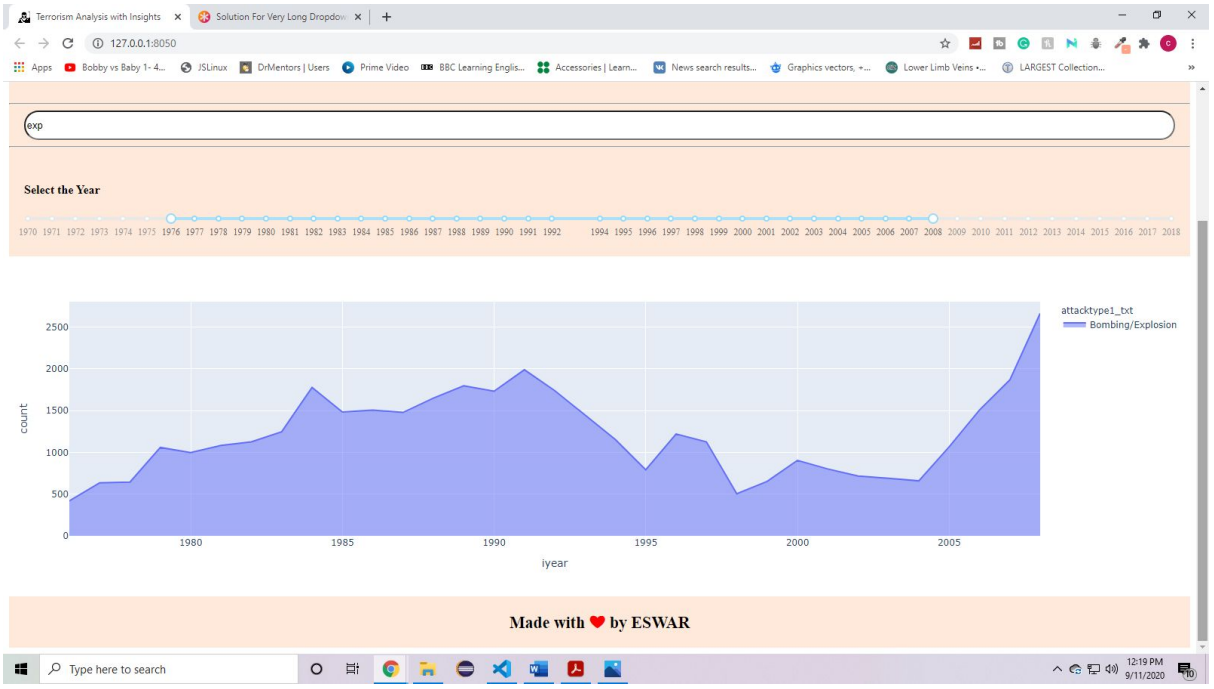

Fig 3.49 : Global Chart Tool - The graph shows filtered data as per attack type, search filter and year slider

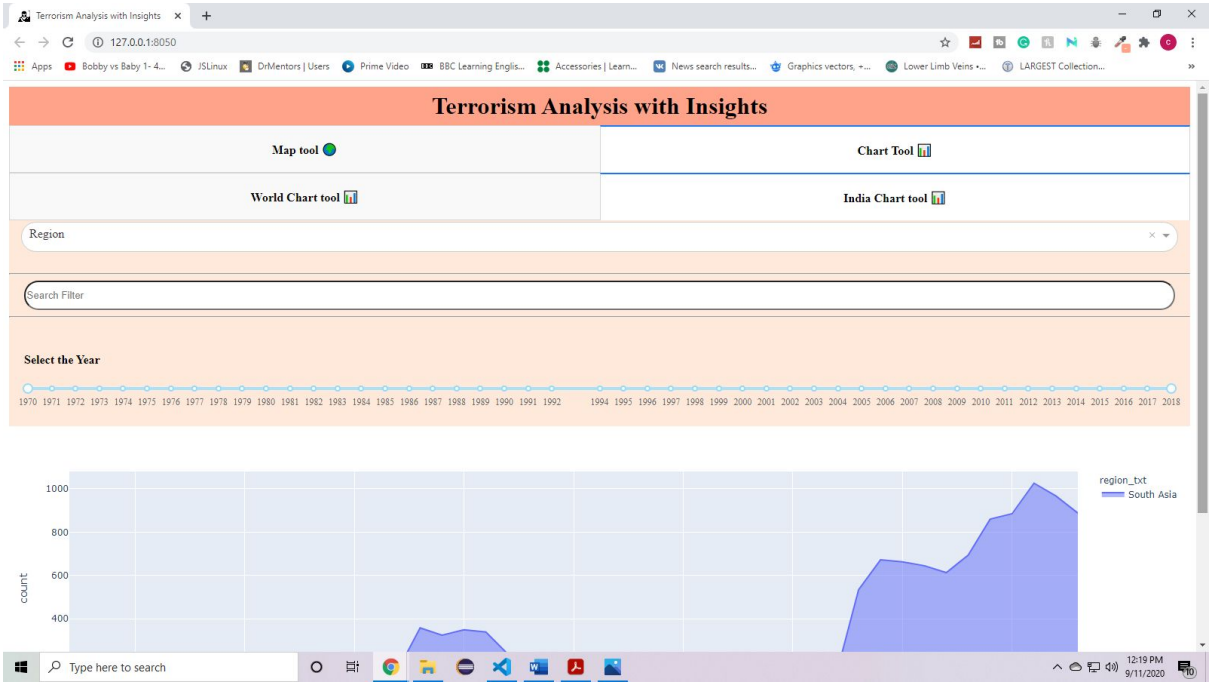

FIg 3.50 : India Chart Tool - The whole data of terrorism in india is shown without any filters (region is selected as default)

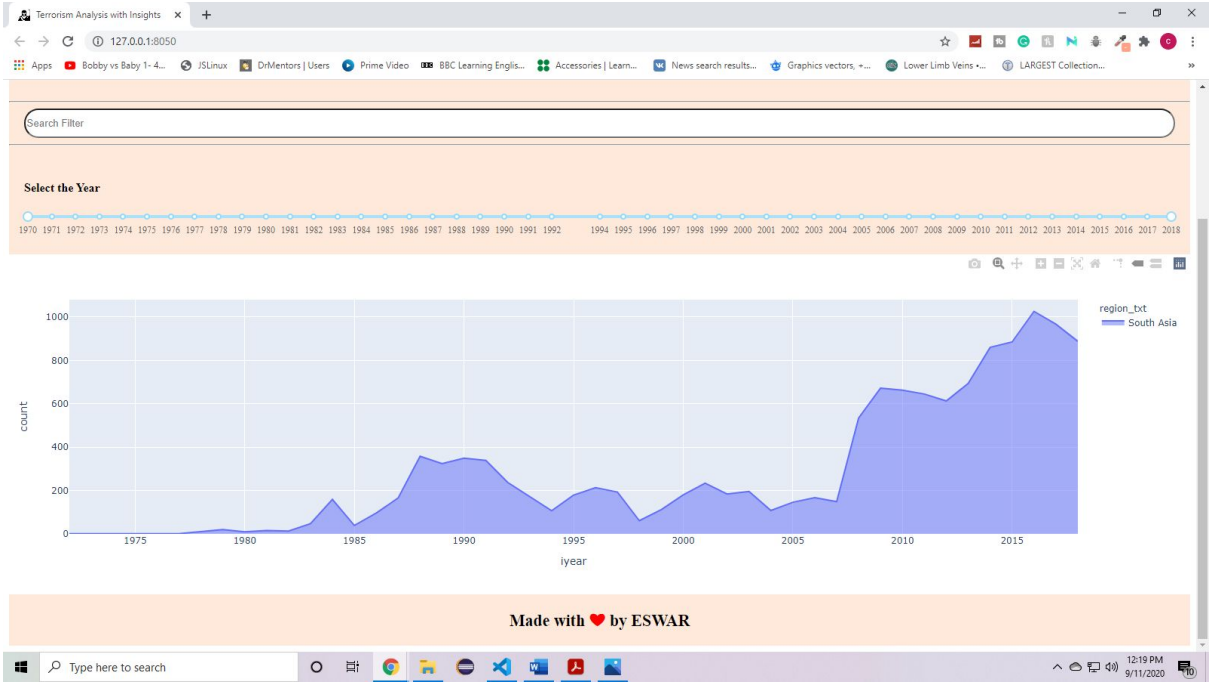

Fig 3.51 : India Chart Tool - The graph shows filtered data as per region in india

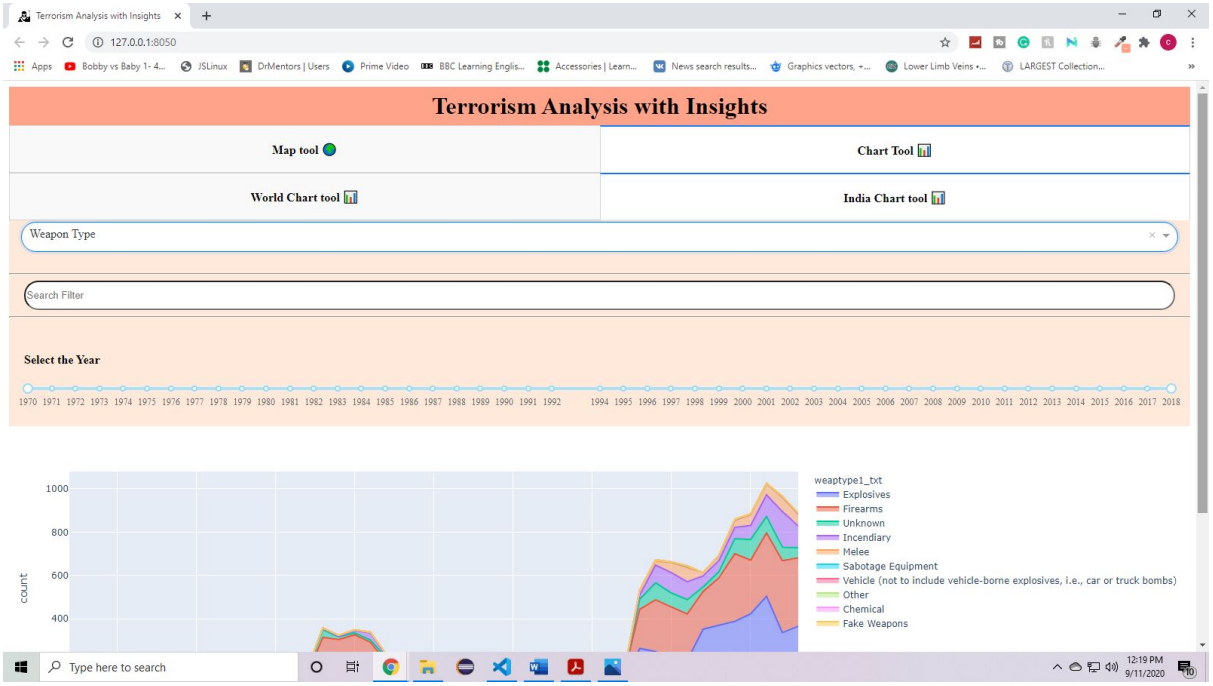

Fig 3.52 : India Chart Tool - Type of weapon is selected as per user requirements through dropdown

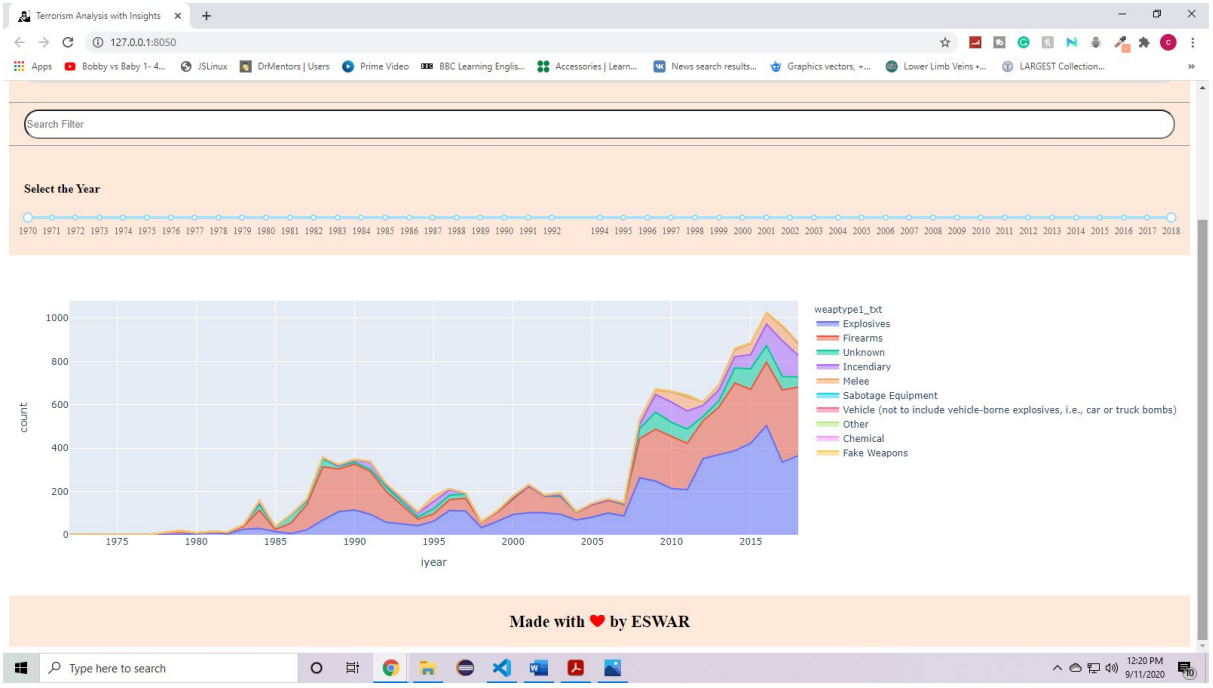

Fig 3.53 : India Chart Tool The graph shows filtered data as per type of weapon

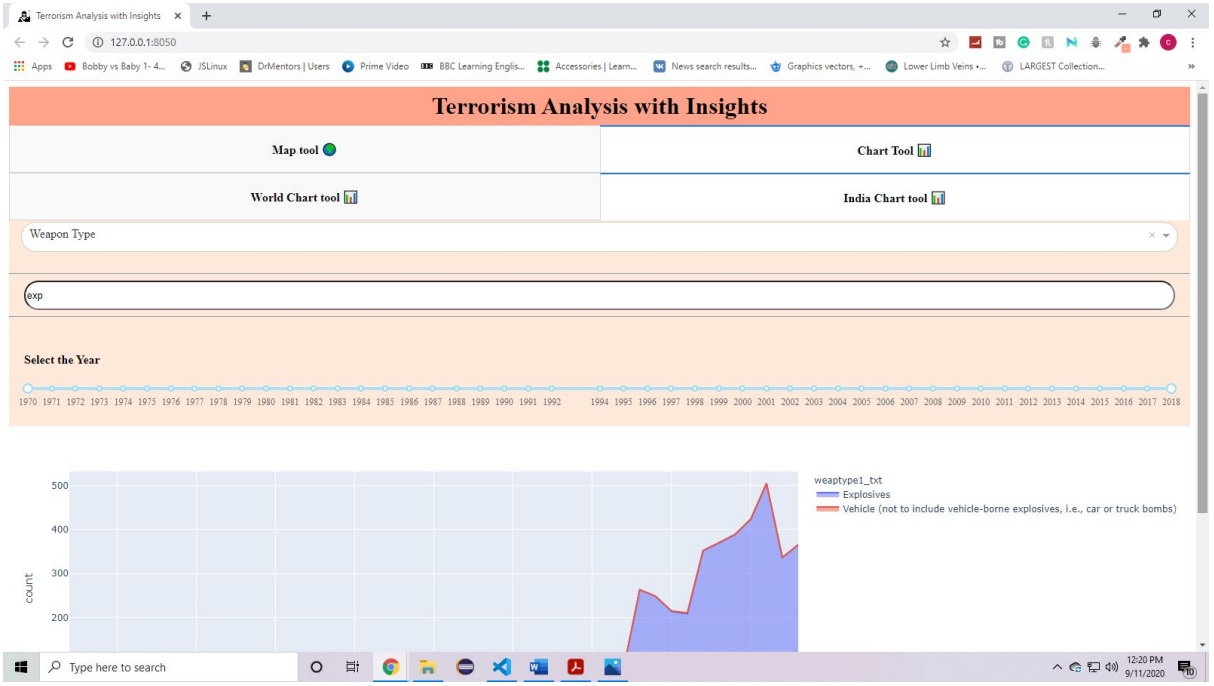

Fig 3.54 : India Chart Tool - The search filter is selected as per user requirement (exp)

Fig 3.55 : India Chart Tool - The graph shows filtered data as per search filter and type of weapon.

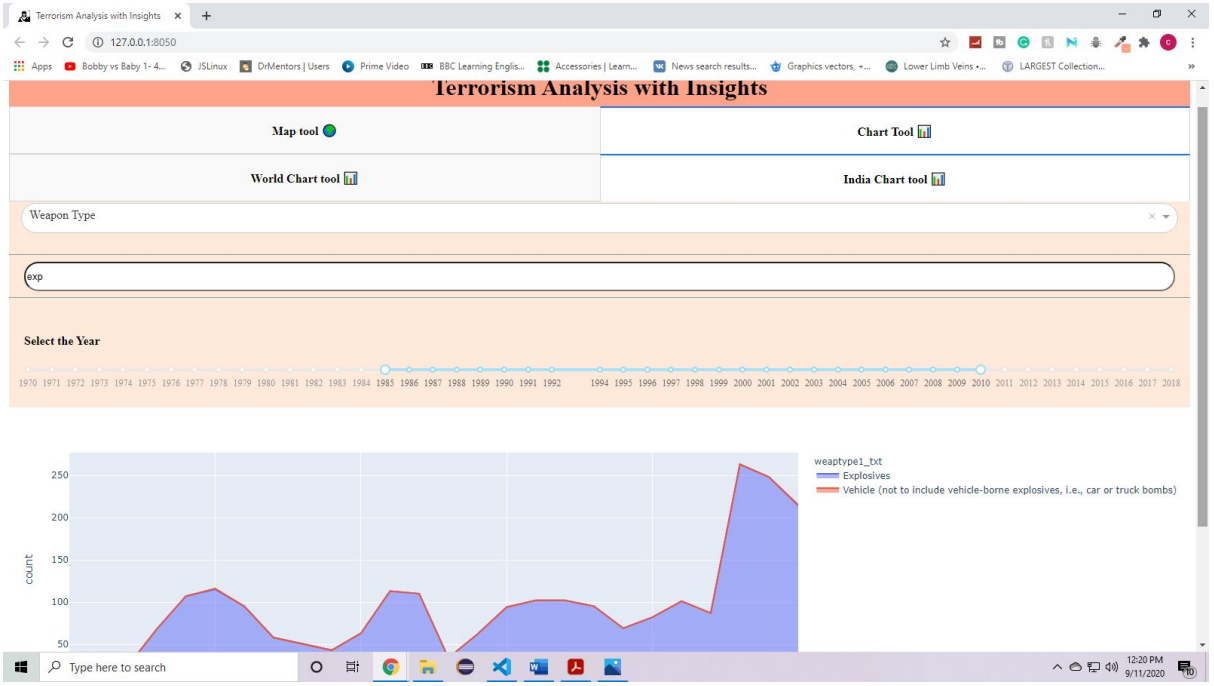

Fig 3.56 : India Chart Tool - Range of years is selected using range slider

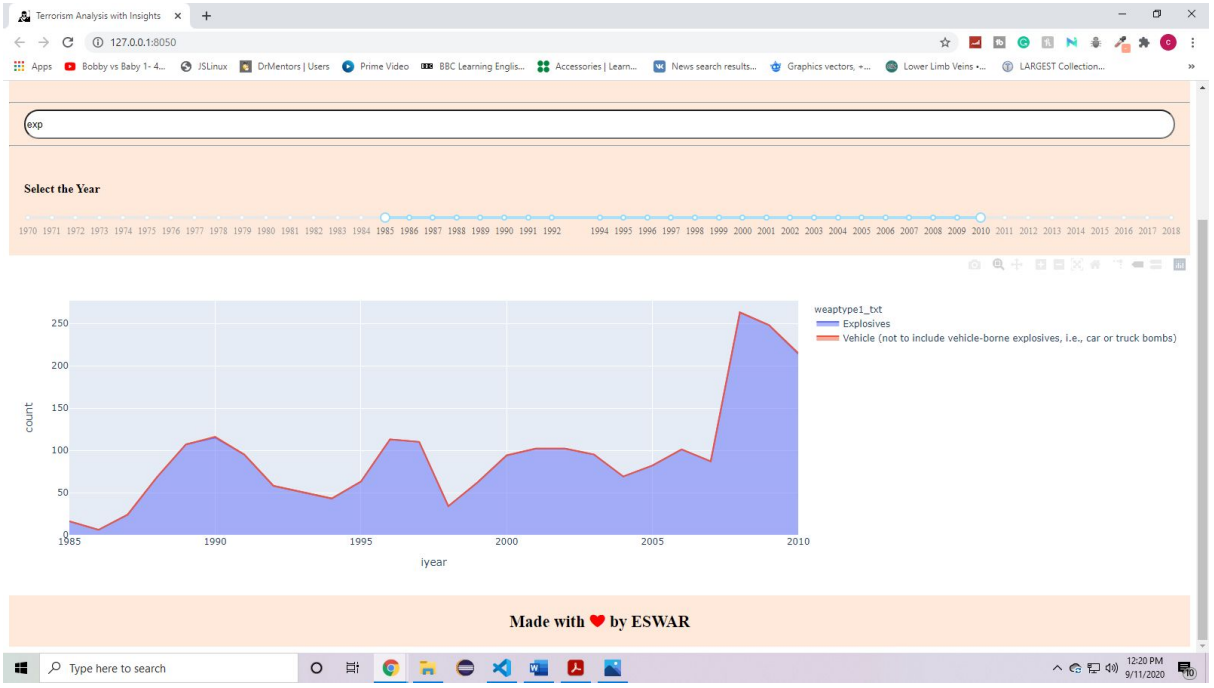

Fig 3.57 : India Chart Tool - The graph shows filtered data as per type of weapon, search filter and year slider

4. Conclusion and future work

The application helps the user to collect insights of the data.
Using multiple filters and dropdowns, the data is filtered as per user requirements and the data is shown in terms of maps and charts colorfully.
It helps in identifying different types of attacks in map tool and progression of attacks in chart tools with respect to year.
Each event data is shown on the mouse hover on a particular event

Future Work:

The project can be further improved by adding future predictions.

5. Acknowledgements:

My acknowledgements to Dr. Sylvester Fernandes, Forsk Coding School and Yogendra Singh for guiding me in completion of the project “terrorism analysis with insights using Python”.

6. References:

---

1. https://towardsdatascience.com/how-to-build-a-complex-reporting-dashboard-using-dash-and-plotl-4f4257c18a7f
2. https://www.dash.org/
3. https://plotly.com/
4. https://www.youtube.com/watch?v=Ma8tS4p27JI&list=PLH6mU1kedUy8fCzkTTJlwsf2EnV_UvOV-&index=1
5. https://stackoverflow.com/

7. Biography:

I am T.Eswar, pursuing my 4th year of Bachelor of Engineering in Computer Science and Engineering in Methodist College of Engineering and Technology in hyderabad.

Certified Courses:

Year of completion	Course	Institution
6th june 2020 to 20th july 2020	IOT and Robotics (internship)	APSIS Solutions
25th May 2020 to 25 July 2020	Java full stack development (internship)	Goal street
April 2020	Industrial IoT on Google Cloud Platform	Google Cloud (Coursera)
8th and 9th Feb 2020	IoT Challange 2020, hyderabad	I3indya Technologies
14th and 15th sep. 2019	International Space Apps Challenge, hyderabad	NASA
May 2018	Python Bootcamp: Go from zero to hero in Python 3	Pireian Data International by Jose Portilla (Udemy)

I have also done other projects, Grocery House, website from Goal Street in online mode.