GUN VIOLENCE DATASET - EXPLORATORY DATA ANALYSIS

EDA PROJECT - 1

SOURCE CODE:

(Created By Ritwik Chandra Pandey - October 2022)

---

1. LIBRARIES USED:

pandas, opendatasets, plotly, re, wordcloud, math, numpy, matplotlib, tqdm, warnings, plotly, collections, PIL, folium.

In [1]:

```
import pandas as pd
pd.options.mode.chained_assignment = None  # default='warn'

import opendatasets as od

import plotly.express as px
import plotly.graph_objs as go
from plotly.offline import iplot
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
from plotly.subplots import make_subplots

import re

from wordcloud import WordCloud

from math import pi

import numpy as np

import matplotlib.pyplot as plt
from matplotlib.patches import Patch
from matplotlib.lines import Line2D

from tqdm import tqdm

import warnings
warnings.filterwarnings("ignore") #For decpracation warnings, if any
```

```
from collections import Counter

from PIL import Image

import folium
```

2. DOWNLOADING THE DATA

In [2]:
```
download_url = 'https://www.kaggle.com/jameslko/gun-violence-data'
od.download(download_url)
```

```
Skipping, found downloaded files in "./gun-violence-data" (use force=True to force download)
```

3. DATA CLEANING AND PREPARATION

3.1 DATA CLEANING

We load the dataset and inlcude the shooting that took place on 10-01-2017 in Las Vegas, Nevada as it is not inlcuded in the dataset.

In [3]:
```
data_filename = './gun-violence-data/gun-violence-data_01-2013_03-2018.csv'
df = pd.read_csv(data_filename)

missing_row = ['sban_1', '2017-10-01', 'Nevada', 'Las Vegas', 'Mandalay Bay 3950 Blvd S', 59, 489,
               'https://en.wikipedia.org/wiki/2017_Las_Vegas_shooting',
               'https://en.wikipedia.org/wiki/2017_Las_Vegas_shooting',
               False, 4, '-', '0::223 Rem [AR-15]', 'Shot - Dead (murder, accidental, suicide)', '36.095', 'Hotel',
               '-115.171667', 47, 'Route 91 Harvest Festiva; concert, open fire from 32nd floor. \
               47 guns seized; TOTAL:59 kill, 489 inj, number shot TBD,girlfriend Marilou Danley POI'
               ,'-', '548::Adult 18+', '0::Male', '-', '-', '-', '-', '-', '-', '-']
df.loc[len(df)] = missing_row
```

Let us look at this dataframe *df* and see the different columns that it has.

In [4]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 239678 entries, 0 to 239677
Data columns (total 29 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   incident_id                  239678 non-null  object
 1   date                         239678 non-null  object
 2   state                        239678 non-null  object
 3   city_or_county               239678 non-null  object
 4   address                      223181 non-null  object
 5   n_killed                     239678 non-null  int64
 6   n_injured                    239678 non-null  int64
 7   incident_url                 239678 non-null  object
```

```
 8   source_url                  239210 non-null  object 
 9   incident_url_fields_missing  239678 non-null  bool   
 10  congressional_district      227734 non-null  float64
 11  gun_stolen                  140180 non-null  object 
 12  gun_type                    140227 non-null  object 
 13  incident_characteristics    239352 non-null  object 
 14  latitude                    231755 non-null  object 
 15  location_description        42090 non-null   object 
 16  longitude                   231755 non-null  object 
 17  n_guns_involved             140227 non-null  float64
 18  notes                       158661 non-null  object 
 19  participant_age             147380 non-null  object 
 20  participant_age_group       197559 non-null  object 
 21  participant_gender          203316 non-null  object 
 22  participant_name            117425 non-null  object 
 23  participant_relationship    15775 non-null   object 
 24  participant_status          212052 non-null  object 
 25  participant_type            214815 non-null  object 
 26  sources                     239069 non-null  object 
 27  state_house_district        200906 non-null  object 
 28  state_senate_district       207343 non-null  object 
dtypes: bool(1), float64(2), int64(2), object(24)
memory usage: 53.3+ MB
None
```

Let us now look the number of missing values that we have in each column of *df*.

In [5]:

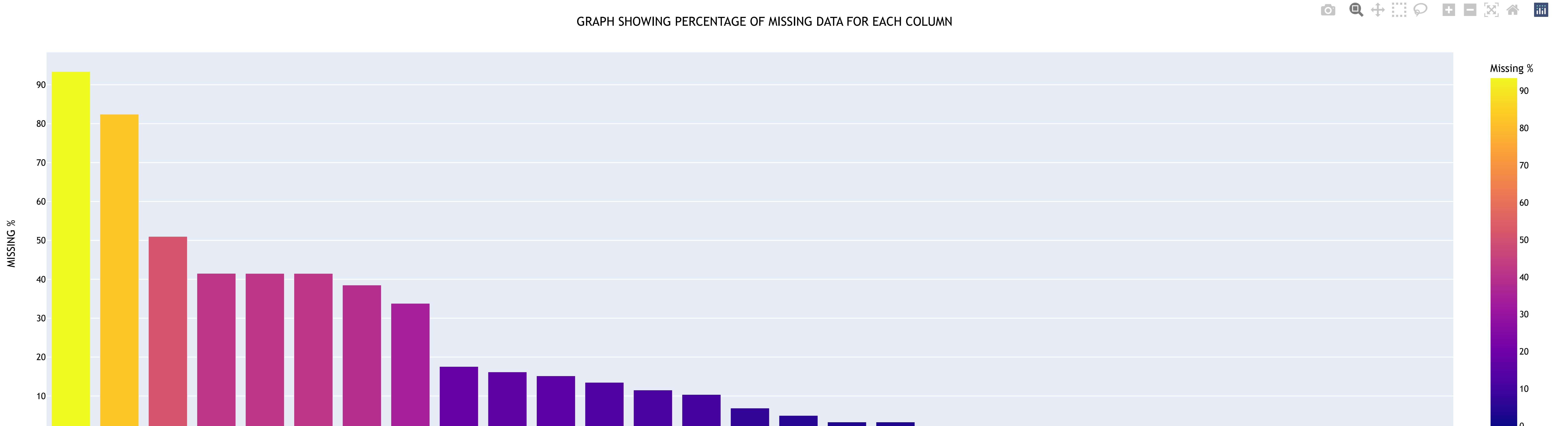
```
print(df.isna().sum())
```

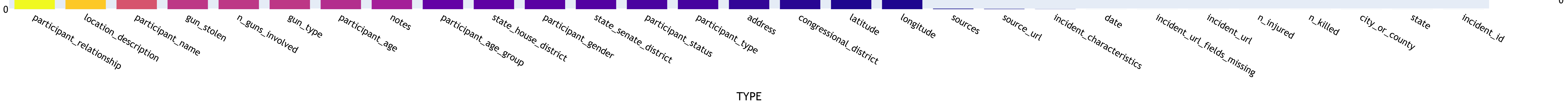
```
incident_id                         0
date                                0
state                               0
city_or_county                      0
address                         16497
n_killed                            0
n_injured                           0
incident_url                        0
source_url                        468
incident_url_fields_missing         0
congressional_district          11944
gun_stolen                      99498
gun_type                        99451
incident_characteristics          326
latitude                         7923
location_description           197588
longitude                        7923
n_guns_involved                 99451
notes                           81017
participant_age                 92298
participant_age_group           42119
participant_gender              36362
participant_name               122253
participant_relationship       223903
participant_status              27626
participant_type                24863
sources                           609
state_house_district            38772
```

```
state_senate_district          32335
dtype: int64
```

We create a new dataframe called *missing* that shows us the percentage of missing data in each column of *df*. We visualise *missing* dataframe using a bar plot.

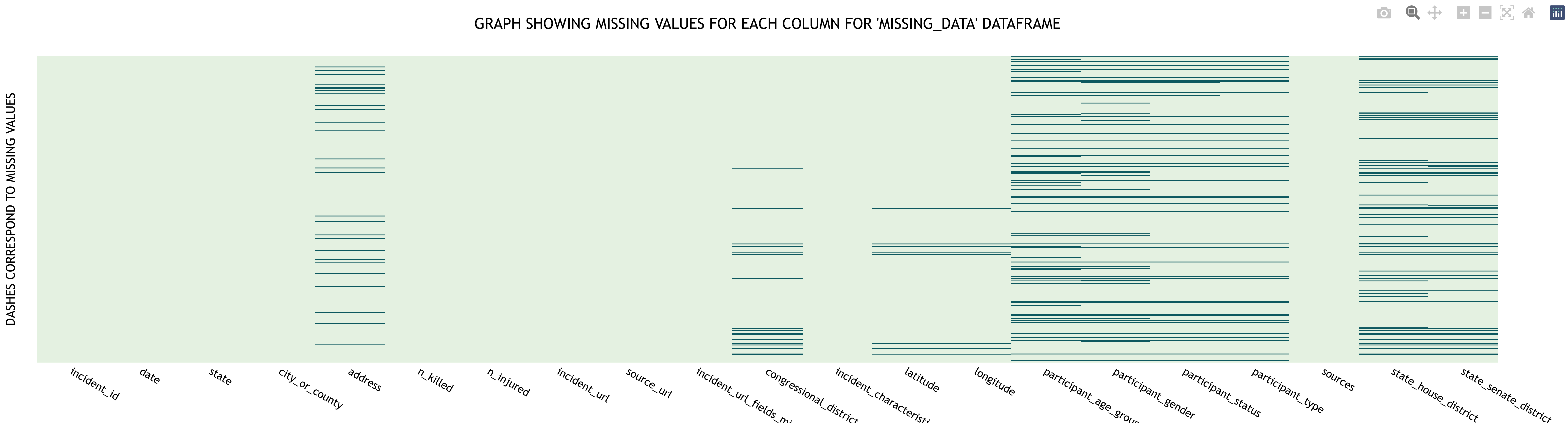
In [6]:
```
missing = (df.isna().sum().sort_values(ascending = False) / len(df))*100
missing = missing.to_frame().reset_index()
missing = missing.rename(columns= {'index':'Type', 0:'Missing %'})

fig = px.bar(missing,x='Type',y='Missing %', color = 'Missing %',height = 600)



fig.update_layout(
    title="GRAPH SHOWING PERCENTAGE OF MISSING DATA FOR EACH COLUMN",
    title_x=0.5,
    xaxis_title="TYPE",
    yaxis_title="MISSING %",
    font=dict(
        family="Trebuchet MS",
        size=10,
        color="Black"
    )
)
fig.show()
missing.set_index('Type',inplace = True)
```





We drop the first eight columns from *df* as they have more than 30% of data missing and give this dataframe a new name called *missing_data*. Moreover, we visualise *missing_data* by observing the columns that have empty values in them.

In [7]:
```
missing_data = df.drop((missing[missing['Missing %']>30]).index, axis = 1)
fig = px.imshow(missing_data.isnull(),color_continuous_scale='mint')
fig.update_yaxes(showticklabels=False)

fig.update_layout(
    title="GRAPH SHOWING MISSING VALUES FOR EACH COLUMN FOR 'MISSING_DATA' DATAFRAME",
    title_x=0.5,
    xaxis_title="TYPE",
    yaxis_title="DASHES CORRESPOND TO MISSING VALUES",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
    coloraxis_showscale=False
)
fig.show()
```



issing

TYPE

We take some specific columns from *missing_data* and collectively call them *picked_data_master*. This new dataframe will help us later on.

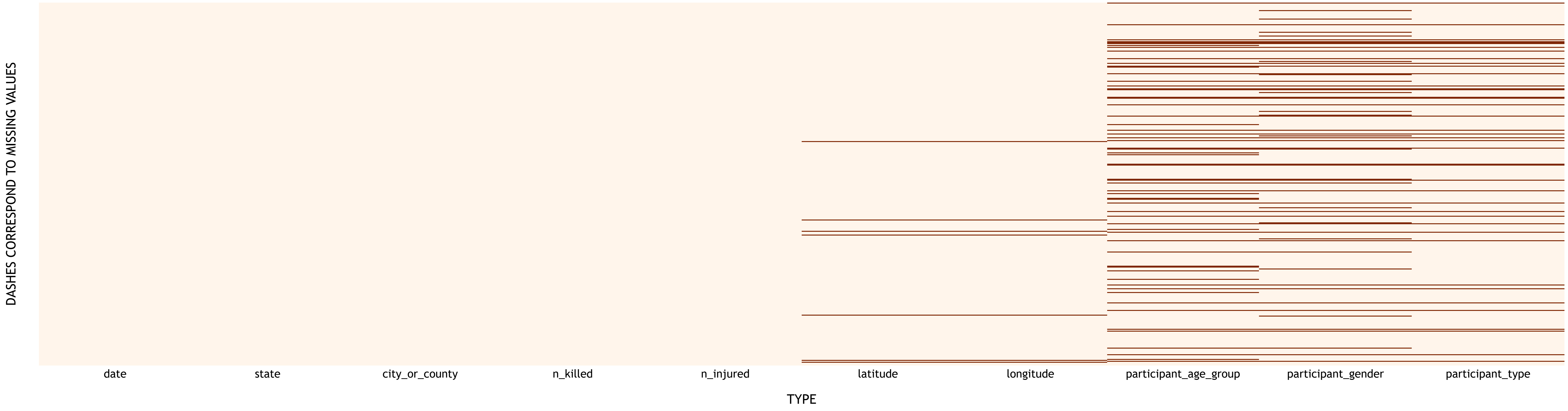
In [9]:
```
picked_data_master = missing_data[['date','state','city_or_county','n_killed','n_injured','latitude',\
                                   'longitude','participant_age_group','participant_gender','participant_type']].copy()
picked_data_master.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 239678 entries, 0 to 239677
Data columns (total 10 columns):
 #   Column                 Non-Null Count   Dtype 
---  ------                 --------------   ----- 
 0   date                   239678 non-null  object
 1   state                  239678 non-null  object
 2   city_or_county         239678 non-null  object
 3   n_killed               239678 non-null  int64 
 4   n_injured              239678 non-null  int64 
 5   latitude               231755 non-null  object
 6   longitude              231755 non-null  object
 7   participant_age_group  197559 non-null  object
 8   participant_gender     203316 non-null  object
 9   participant_type       214815 non-null  object
dtypes: int64(2), object(8)
memory usage: 20.1+ MB
```

We visualise *picked_data_master* now.

In [10]:
```
#picked_data_master = missing_data[['date','state','city_or_county','n_killed','n_injured','latitude',\
                                   #'longitude','participant_age_group','participant_gender','participant_type']].copy()
fig = px.imshow(picked_data_master.isnull(),color_continuous_scale='oranges')
fig.update_yaxes(showticklabels=False)

fig.update_layout(
    title="GRAPH SHOWING MISSING VALUES FOR EACH COLUMN FOR 'PICKED_DATA_MASTER' DATAFRAME",
    title_x=0.5,
    xaxis_title="TYPE",
    yaxis_title="DASHES CORRESPOND TO MISSING VALUES",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
    coloraxis_showscale=False
)
fig.show()
```

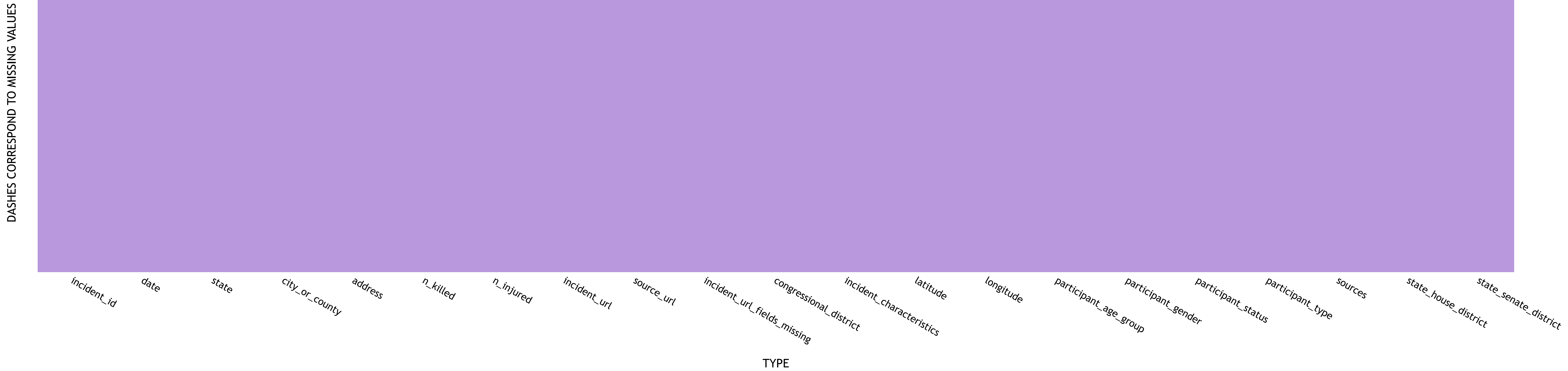
GRAPH SHOWING MISSING VALUES FOR EACH COLUMN FOR 'PICKED_DATA_MASTER' DATAFRAME



Now we delete all those rows in *missing_data* that have any null value in it and call the resulting dataframe as *picked_data*. Moreover, we visualise it.

In [11]:
```
picked_data = missing_data.drop(missing_data[missing_data.isnull().any(axis = 1)].index, axis = 0)
fig = px.imshow(picked_data.isnull(),color_continuous_scale='Purp')
fig.update_yaxes(showticklabels=False)
fig.update_layout(coloraxis_showscale=False)
fig.update_layout(
    title="GRAPH SHOWING MISSING VALUES AFTER DELETING THE ROWS HAVING nan VALUES (PICKED_DATA)",
    title_x=0.5,
    xaxis_title="TYPE",
    yaxis_title="DASHES CORRESPOND TO MISSING VALUES",
    font=dict(
        family="Trebuchet MS",
        size=11,
        color="Black"
    ),
    coloraxis_showscale=False
)
fig.show()
```

GRAPH SHOWING MISSING VALUES AFTER DELETING THE ROWS HAVING nan VALUES (PICKED_DATA)



Now, we add some more columns to *picked_data* which will help us later on to do good amount of analysis.

So till now we have the following data frames:

- df
- missing_data
- picked_data_master
- picked_data

We shall use them whenever required.

3.2 Addition of Columns: Data Preparation

- ### 3.2.1 Total no. of people and total no. of unharmed people

  By iterating over the entire *picked_data* dataframe, we add up no. of people killed and people injured and compare this value with numbers in participant age group. Whichever is greater, gets appended to a list. Number of unharmed people is calculated by carrying out a subtraction between sum of killed and injured people and no. of people obtained from *participant_age_group*. This number obtained is also appended to a list.

  Both the lists are then assigned to appropriate columns as shown below.

In [12]:

```
picked_data = picked_data.reset_index().drop(['index'],axis = 1)

tpeople = []
tunharmed =  []
for x in range(len(picked_data)):
    sum = int(picked_data['participant_age_group'][x].split('||')[-1][0]) + 1
    n_value = picked_data['n_killed'][x] + picked_data['n_injured'][x]


    if(sum < n_value):
```

```
        tpeople.append(n_value)
    else:
        tpeople.append(sum)
    tunharmed.append( tpeople[x] - n_value )
picked_data['TPeople'] = tpeople
picked_data['TUnharmed'] = tunharmed
```

- ### 3.2.2 Total no. adults

  By iterating over the entire *picked_data* dataframe, we calculate no. of adults using the *participant_age_group* column. Each number calculated gets appended to a list and that list is assigned to appropriate column as shown below.

In [13]:

```
tadults = []
for value in range(len(picked_data)):
    x = 0
    lst = re.split('::| ',picked_data['participant_age_group'][value])
    for i in lst:
        if i == 'Adult':
            x = x+1
    tadults.append(x)


picked_data['TAdults'] = tadults
```

- ### 3.2.3 Total no. of teens

  By iterating over the entire *picked_data* dataframe, we calculate no. of teens using the *participant_age_group* column. Each number calculated gets appended to a list and that list is assigned to appropriate column as shown below.

In [14]:

```
 tteens = []
for value in range(len(picked_data)):
    x = 0
    lst = re.split('::| ',picked_data['participant_age_group'][value])
    for i in lst:
        if i == 'Teen':
            x = x+1
    tteens.append(x)

picked_data['TTeens'] = tteens
```

- ### 3.2.4 Total no. of children

  By iterating over the entire *picked_data* dataframe, we calculate no. of children using the *participant_age_group* column. Each number calculated gets appended to a list and that list is assigned to appropriate column as shown below.

In [15]:

```
tchild = []
for value in range(len(picked_data)):
```

```
    x = 0
    lst = re.split('::| ',picked_data['participant_age_group'][value])
    for i in lst:
        if i == 'Child':
            x = x+1
    tchild.append(x)

picked_data['TChild'] = tchild
```

- ### 3.2.5 Total no. of males

  By iterating over the entire *picked_data* dataframe, we calculate no. of males using the *participant_gender* column. Each number calculated gets appended to a list and that list is assigned to appropriate column as shown below.

In [16]:

```
tmales = []
for value in range(len(picked_data)):
    x = 0;
    lst = re.split('',picked_data['participant_gender'][value])
    for i in lst:
        if i == 'M':
            x = x+1
    tmales.append(x)

picked_data['TMales'] = tmales
```

- ### 3.2.6 Total no. of females

  By iterating over the entire *picked_data* dataframe, we calculate no. of females using the *participant_gender* column. Each number calculated gets appended to a list and that list is assigned to appropriate column as shown below.

In [17]:

```
tfemales = []
for value in range(len(picked_data)):
    x = 0;
    lst = re.split('',picked_data['participant_gender'][value])
    for i in lst:
        if i == 'F':
            x = x+1
    tfemales.append(x)

picked_data['TFemales'] = tfemales
```

- ### 3.2.7 Total no. of people whose gender is unknown

  By iterating over the entire *picked_data* dataframe, we calculate no. of people whose gender is unknown using three columns, *TMales*, *TFemales* and *TPeople* as shown below. We use a list here too which is appended at every step.

  Finally, the list is assigned to appropriate column as shown below.

In [18]:
```
tunknown_gender = []
for value in range(len(picked_data)):
    x = 0
    if (picked_data['TMales'][value] + picked_data['TFemales'][value]) !=  picked_data['TPeople'][value]:
        x = picked_data['TPeople'][value] - (picked_data['TMales'][value] + picked_data['TFemales'][value])
        if( x<0 ):
            x = 0
    tunknown_gender.append(x)
picked_data['TUnknown_gender'] = tunknown_gender
```

- ### 3.2.8 Total no. of people who were victims

  By iterating over the entire *picked_data* dataframe, we calculate no. of people who were victims using *participant_type* column as shown below. We use a list here too which is appended at every step.

  Finally, the list is assigned to appropriate column as shown below.

In [19]:
```
tvictim = []
for value in range(len(picked_data)):
    x = 0;
    lst = re.split('',picked_data['participant_type'][value])
    for i in lst:
        if i == 'V':
            x = x+1
    tvictim.append(x)

picked_data['TVictim'] = tvictim
```

- ### 3.2.9 Total no. of people who were subjects

  By iterating over the entire *picked_data* dataframe, we calculate no. of people who were subjects using *participant_type* column as shown below. We use a list here too which is appended at every step.

  Finally, the list is assigned to appropriate column as shown below.

In [20]:
```
tsubject = []
for value in range(len(picked_data)):
    x = 0;
    lst = re.split('',picked_data['participant_type'][value])
    for i in lst:
        if i == 'j':
            x = x+1
    tsubject.append(x)

picked_data['TSubject'] = tsubject
```

- ### 3.2.10 Total no. of people who were arrested

  By iterating over the entire *picked_data* dataframe, we calculate no. of people who were arrested using *participant_status* column as shown below. We use a list here too which is appended at every step.

  Finally, the list is assigned to appropriate column as shown below.

In [21]:
```
tarrested = []
for value in range(len(picked_data)):
    x = 0;
    lst = re.split('',picked_data['participant_status'][value])
    for i in lst:
        if i == 'A':
            x = x+1
    tarrested.append(x)

picked_data['TArrested'] = tarrested
```

- ### 3.2.11 Total no. of people who were not arrested

  By iterating over the entire *picked_data* dataframe, we calculate no. of people who were not arrested using *TSubject* and *TArrested* column as shown below. We use a list here too which is appended at every step.

  Finally, the list is assigned to appropriate column as shown below.

In [22]:
```
tnarrested = []
for value in range(len(picked_data)):
    x = picked_data['TSubject'][value] - picked_data['TArrested'][value]
    if x>0:
        tnarrested.append(x)
    else:  tnarrested.append(0)

picked_data['TNarrested'] = tnarrested
```

We add one more dataframe to our collection now. It shall be used for analysis later on. The new dataframe is *numeric_data*.

In [23]:
```
numeric_data = picked_data[['TPeople','n_killed', 'n_injured', 'TUnharmed', 'TAdults', 'TTeens', 'TChild', 'TMales', 'TFemales', \
                            'TUnknown_gender', 'TSubject', 'TVictim', 'TArrested', 'TNarrested']].copy()
numeric_data = numeric_data.rename(columns={'n_killed':'TKilled', 'n_injured':'TInjured','TUnknown_gender':'TUnknownGender'})
```

4. DATA ANALYSIS - GENERAL

We shall use *.describe()* to get an estimate of the meaningful numeric columns of our dataframe *numeric_data*.

In [24]:
```
numeric_data.describe()
```

Out[24]:

	TPeople	TKilled	TInjured	TUnharmed	TAdults	TTeens	TChild	TMales	TFemales	TUnknownGender	TSubject	TVictim	TArrested	TNarrested
count	148737.000000	148737.000000	148737.000000	148737.000000	148737.000000	148737.000000	148737.000000	148737.000000	148737.000000	148737.000000	148737.000000	148737.000000	148737.000000	148737.000000

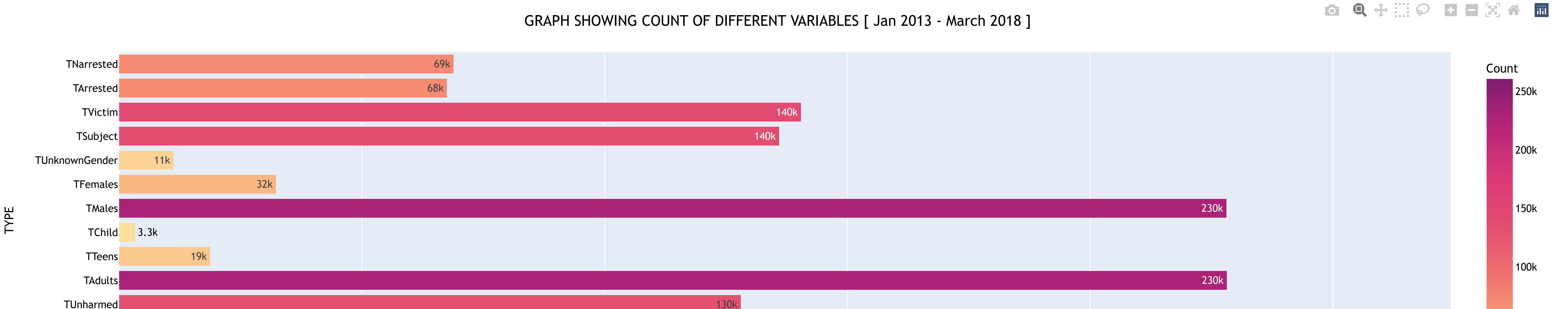
mean	1.751703	0.317682	0.572816	0.861205	1.534124	0.126626	0.022348	1.533526	0.217787	0.075825	0.914271	0.944513	0.454346	0.463543
std	1.796217	0.593857	1.476776	1.038078	1.084341	0.447739	0.185385	0.993129	0.498925	1.464343	0.935184	0.855100	0.802070	0.766363
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000
75%	2.000000	1.000000	1.000000	1.000000	2.000000	0.000000	0.000000	2.000000	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000
max	548.000000	59.000000	489.000000	10.000000	103.000000	27.000000	11.000000	61.000000	23.000000	547.000000	32.000000	102.000000	31.000000	10.000000

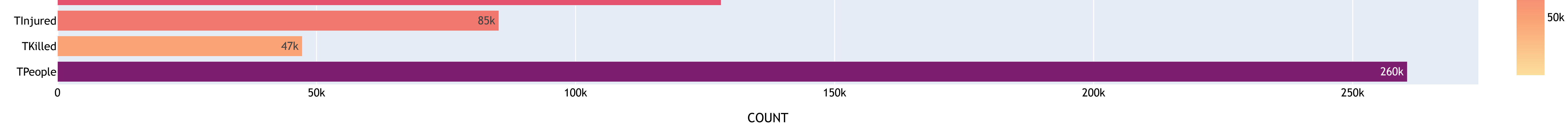
Lets us plot the various columns of *numeric_dataframe* using a bar graph to draw some insights.

For this, we use a dictionary called *bar_plot_data1*, and employ *px.bar()*. The necessary details and adjustments to the figure are made using px.update_layout() as shown below.

In [25]:

```
bar_plot_data1 = {'Type' : list(numeric_data), 'Count' : []}
for i in bar_plot_data1['Type']:
    bar_plot_data1['Count'].append( numeric_data[i].sum())
fig = px.bar(bar_plot_data1,text_auto='.2s',orientation='h', x = 'Count', y = 'Type',color='Count',color_continuous_scale = 'Sunsetdark')
fig.update_layout(
    title="GRAPH SHOWING COUNT OF DIFFERENT VARIABLES [ Jan 2013 - March 2018 ]",
    title_x=0.5,
    xaxis_title="COUNT",
    yaxis_title="TYPE",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
    coloraxis_showscale=True
)
fig.show()
```





KEY TAKEAWAYS:

1. It is observed here that total no. of people who are supposed to be arrested (*TNarrested*) is slightly MORE THAN people (subjects) who have been arrested.
2. The subjects are almost same in number to victims.
3. The number of males involved in these accidents is almost 7x the number of females.
4. Adults (18+) are way more in number (≈ 10x) than no. of childeren (0-11) and teens (12-17) combined.
5. No. of people who were killed is half the number of people who were injured.
6. 18% of people involved in these accidents died.

5. DATA ANALYSIS - TIME

We shall do time related analysis of our data and see what insights we can get from it.

Before that, we change the datatype of the *date* column in both *picked_data_master* and *picked_data*.

In [26]:

```
picked_data['date'] = picked_data['date'].astype('datetime64[ns]')
picked_data_master['date'] = picked_data_master['date'].astype('datetime64[ns]')
```

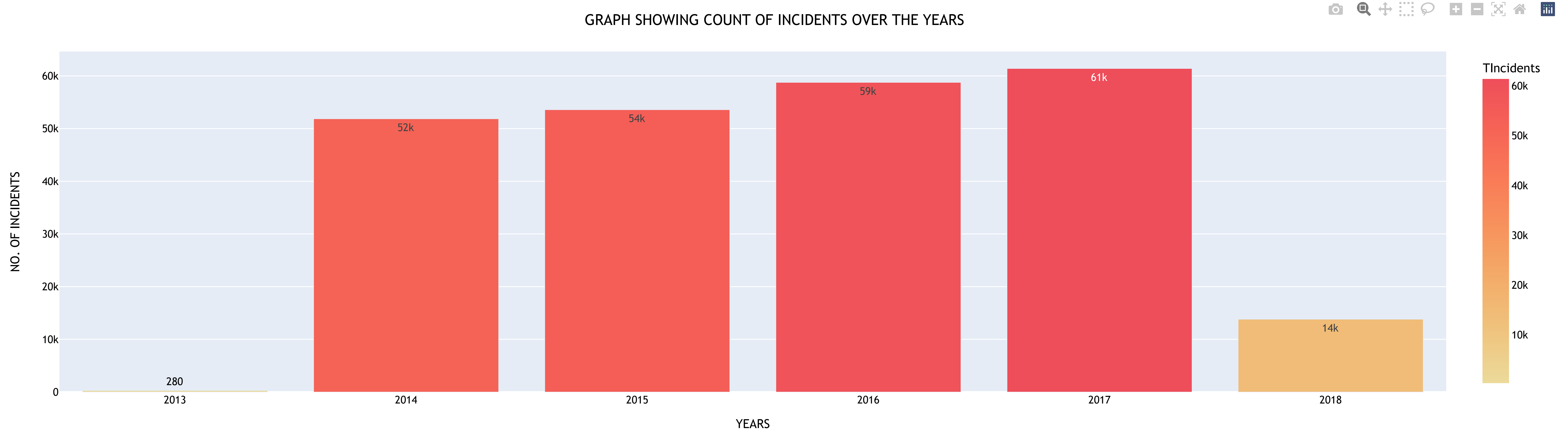
5.1 Incidents over the years

A dictionary *years* is used as one of the arguments to *px.bar()*. The total number of incidents is calculated using a for loop and *count* is progressively appended to the list associated with '*TIncidents*' as shown below.

In [27]:

```
years = {'Years':['2013','2014','2015','2016','2017','2018'], 'TIncidents':[]}
for x in years['Years']:
    count = 0
    for value in range(len(picked_data_master)):
        if(str(picked_data_master['date'][value].year) == x):
            count = count + 1
    years['TIncidents'].append(count)

fig = px.bar(years,text_auto='.2s',x = 'Years', y = 'TIncidents',color='TIncidents',color_continuous_scale = 'Oryel')
fig.update_layout(
    title="GRAPH SHOWING COUNT OF INCIDENTS OVER THE YEARS",
    title_x=0.5,
    xaxis_title="YEARS",
    yaxis_title="NO. OF INCIDENTS",
    font=dict(
        family="Trebuchet MS",
        size=12,
```

```
        color="Black"
    ),

)
fig.show()
```



KEY TAKEAWAYS:

1. The no. of incidents reported in the year 2013 is very less, this shows that the dataset is not complete when it comes to having all incidents from 2013.
2. Interestingly, the number of incidents are increasing year after year (when considered from 2014 onwards).
3. 2018 has data only till March and it seems to follow the trend.

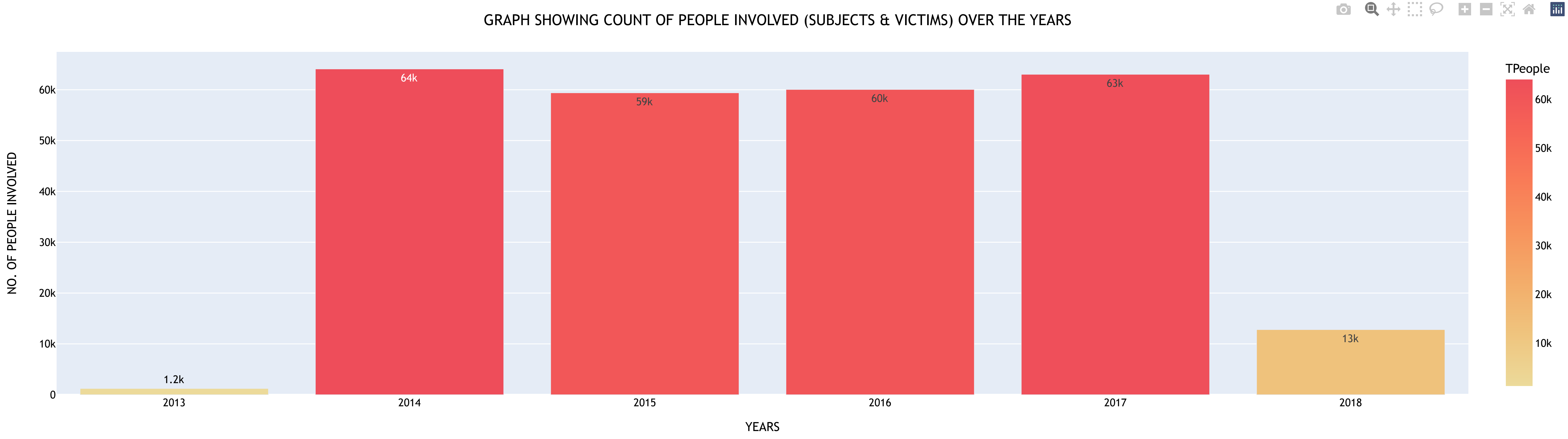
5.2 No. of people involved over the years

The methodology of calculating no. of people involved and presenting the data is almost same as above.

In [28]:

```
years = {'Years':['2013','2014','2015','2016','2017','2018'], 'TPeople':[]}
for x in years['Years']:
    count = 0
    for value in range(len(picked_data)):
        if(str(picked_data['date'][value].year) == x):
            count = count + numeric_data['TPeople'][value]
    years['TPeople'].append(count)
```

```
fig = px.bar(years,text_auto='.2s', x = 'Years', y = 'TPeople',color='TPeople',color_continuous_scale = 'Oryel')
fig.update_layout(
    title="GRAPH SHOWING COUNT OF PEOPLE INVOLVED (SUBJECTS & VICTIMS) OVER THE YEARS ",
    title_x=0.5,
    xaxis_title="YEARS",
    yaxis_title="NO. OF PEOPLE INVOLVED",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),

)
```



KEY TAKEAWAYS:

1. As expected, the no. of people involved in incidents form 2013 is very less.
2. One surprising thing is that, though 2014 had the least amount of incidents when compared with the next 3 successive years, the number of people involved is maximum.
3. From 2015 onwards, the trend is followed and number increasing progressively.

NOTE : AS THE NO. OF INCIDENTS IN THE YEAR 2013 IS VERY LESS, WE SHALL NOT CONSIDER THAT FOR THE REST OF OUR ANALYSIS. WE WILL NOT DELETE THOSE DATA POINTS BUT SIMPLY REFUSE TO ACKNOWLEDGE IT. THEIR INCLUSION WILL NOT AFFECT OUR ANALYSIS IN ANY MANNER.

5.3 Incidents by months (Usual and Average)

USUAL - We take the combined count of incidents over all the years [Jan 2014 - Mar 2015] throughout different months.

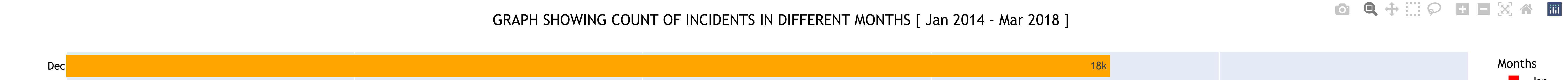
AVERAGE - We take the combined count of incidents over all the years [Jan 2014 - Mar 2015] throughout different months but divide each count by the numnber of occurences of each month over the years.

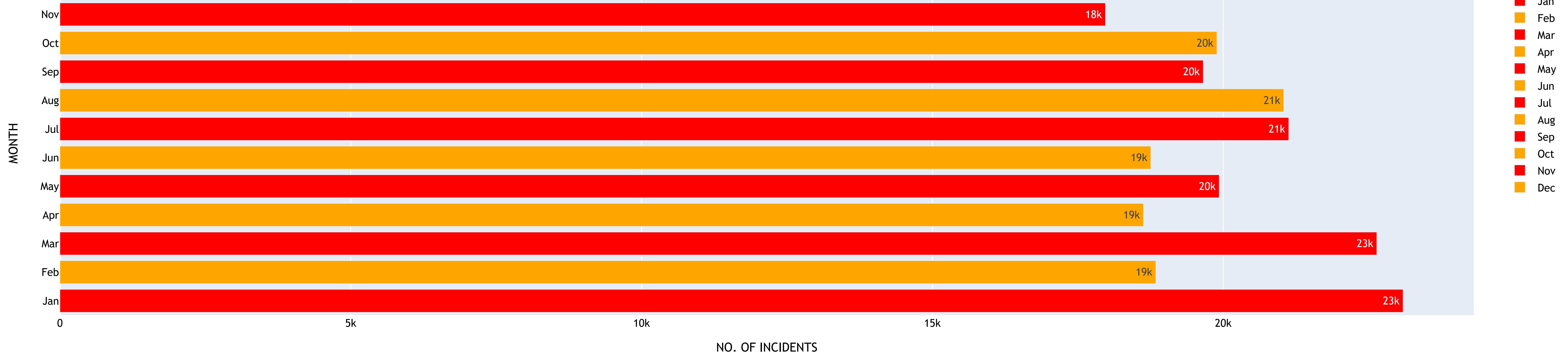
- ### 5.3.1 Using bar graph

  For 'usual' graph, *months* dictionary is used. It is provided values using the *date* column of *picked_data_master* dataframe and the dictionary is then passed on to *px.bar()* for plotting. In the case of 'average' graph, the *count* obtained for each month is divided by the number of occurences of those months from Jan 2014 to Mar 2018. Here also, a dictionary is passed on to *px.bar()* for plotting.

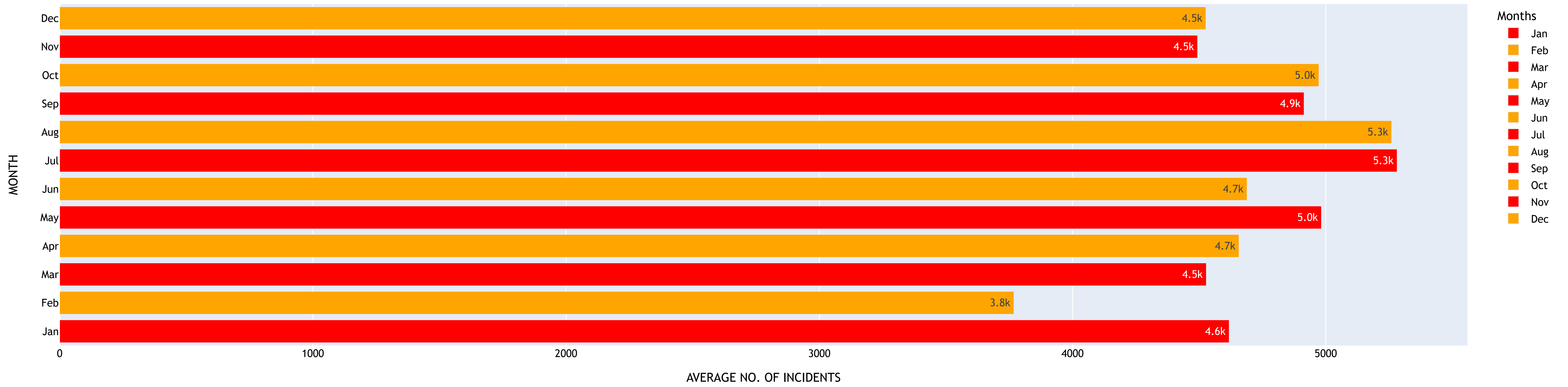
In [29]:
```
months = {'month' : [], 'count' : []}

for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].month) not in months['month']):
        months['month'].append(str(picked_data_master['date'][value].month))
        months['count'].append(0)
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].month) in months['month']):
        index = months['month'].index(str(picked_data_master['date'][value].month))
        months['count'][index] += 1;

months['month'] = ['Jan','Feb','Mar','Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct','Nov','Dec']

fig1 = px.bar(months,orientation = 'h',text_auto='.2s',labels={'y':'Month', 'x':'No. of incidents'},x = months['count'],color_discrete_sequence = ['red','orange'], y=months['month'],color =
months['month'])
fig1.update_layout(
    title="GRAPH SHOWING COUNT OF INCIDENTS IN DIFFERENT MONTHS [ Jan 2014 - Mar 2018 ]",
    title_x=0.5,
    yaxis_title="MONTH",
    xaxis_title="NO. OF INCIDENTS",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
    legend_title_text = "Months"

)

months = {'month' : [], 'count' : []}

for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].month) not in months['month']):
        months['month'].append(str(picked_data_master['date'][value].month))
        months['count'].append(0)
for value in range(len(picked_data_master)):
```

```
    if (str(picked_data_master['date'][value].month) in months['month']):
        index = months['month'].index(str(picked_data_master['date'][value].month))
        months['count'][index] += 1;




quotient = []

for number in months['count']:
    x = months['month'][months['count'].index(number)]
    if x == '1':
        divisor = 5
    elif x == '2' :
        divisor = 5
    elif x == '3':
        divisor = 5
    else:
        divisor = 4

    quotient.append(number/divisor)
months['month'] = ['Jan','Feb','Mar','Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct','Nov','Dec']
hovertemplate = "Count : " + x
fig = px.bar(months,orientation = 'h',text_auto='.2s',labels={'y':'Month', 'x':'no. of incidents'},x = quotient, y=months['month'],color_discrete_sequence = ['red','orange'],color = months['month'])

fig.update_layout(
    title="GRAPH SHOWING AVERAGE COUNT OF INCIDENTS BY MONTHS [ Jan 2014 - Mar 2018 ]",
    title_x=0.5,
    yaxis_title="MONTH",
    xaxis_title="AVERAGE NO. OF INCIDENTS",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
    legend_title_text = "Months"

)
fig1.show()
fig.show()
```





GRAPH SHOWING AVERAGE COUNT OF INCIDENTS BY MONTHS [Jan 2014 - Mar 2018]



KEY TAKEAWAYS:

1. Note that in the 'usual' plot, the number of accidents is way more in the months of January and March (over 23k) than any other month. One reason can be the fact that Jan and March also have inputs from the year 2018, unlike other months. Addressing the issue with February, it is beacuse February has 3 days less (28 days) than most of the other months.

2. It is in the 'average' plot that we get the real sense of distribution of incidents over the months. Here also, Feb has least number of incidents. Explanation: we have 5 Februaries in total, we lose close to 9-10 days worth of incidents as it has only 28 days (which is atleat two days less than other months). We note that July and August have most incidents throughout the years. These are summer months and usually people go out out of their homes most in these months. One thing interesting that I found was that July is most dangerous month when it comes to preventable deaths. Also, road accidents happen maximum in July and August. It also has to do with major holidays that occur during these months. Further research is required on these lines.

- ### 5.3.2 Using radial bar graph

  Here also we use a dictionary called *months* and utilise the power of matplotlib library to create radial bar charts. All the counts associated with each month is divided by the maximum *count* value that occurs(known from the previous graph) and thats why there is one full cicle in radial bar charts below. The others are visully estimated in relation to this full circle. It gives a different way of visulaising data. That is all for 'usual' radial bar graph, but for 'average' one, we divide each *count* value with the number of times the months occur from Jan 2014 to Mar 2018 and then all the resulting values are divided by the largest *count* value to get atleast one full circle.

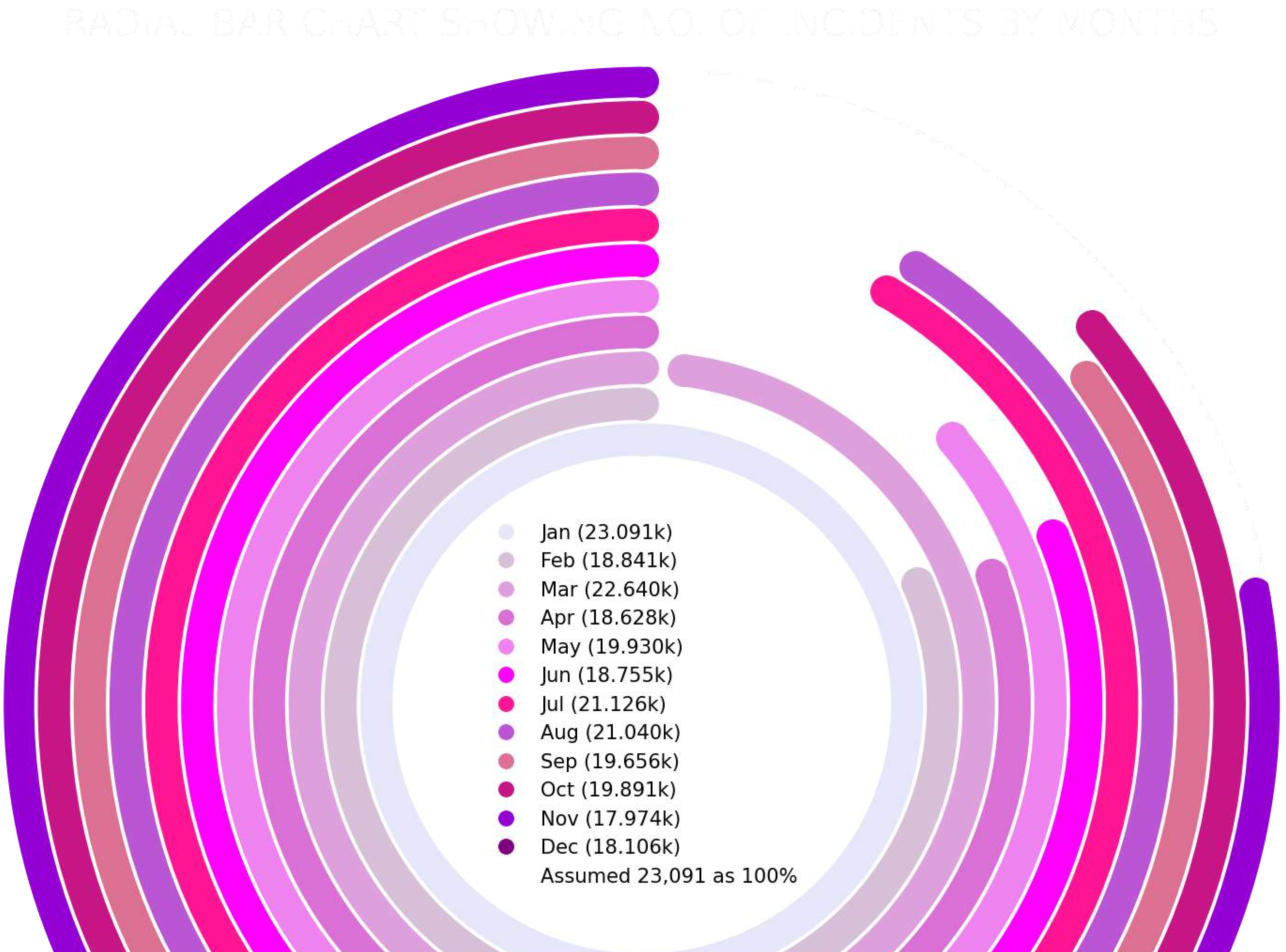
In [30]:

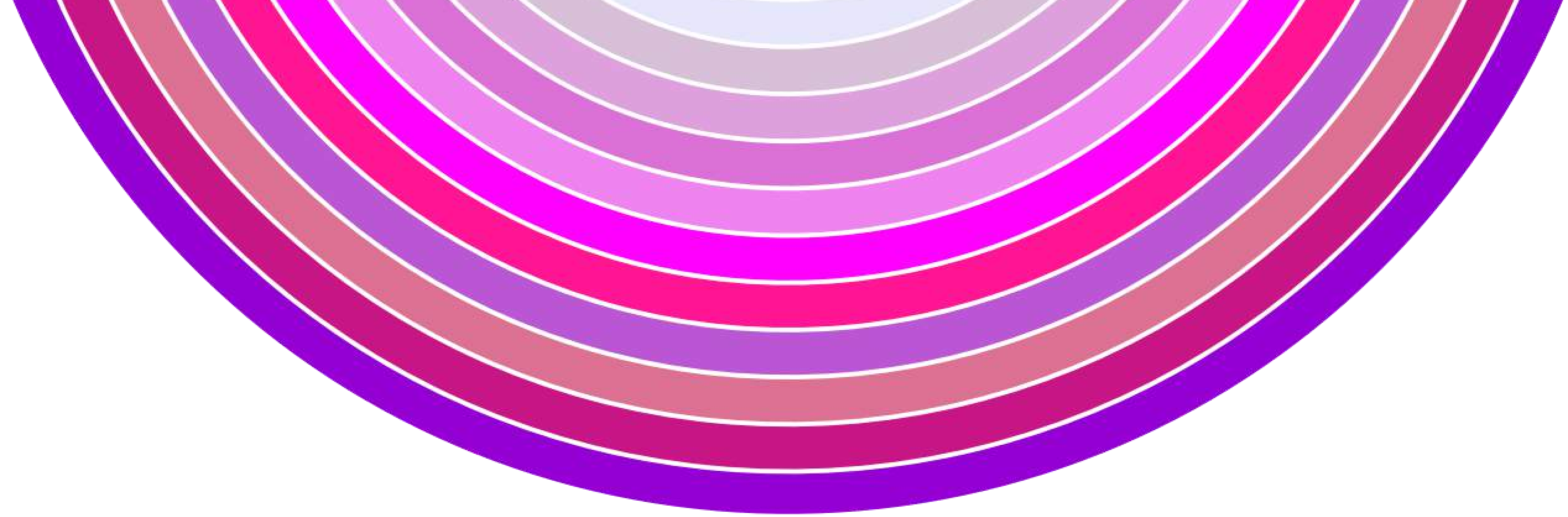
```
months = {'month' : [], 'count' : []}

for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].month) not in months['month']):
        months['month'].append(str(picked_data_master['date'][value].month))
        months['count'].append(0)
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].month) in months['month']):
        index = months['month'].index(str(picked_data_master['date'][value].month))
        months['count'][index] += 1;
for value in range(len(months['count'])):
    months['count'][value] = (months['count'][value]/(23091)) * 100



fig, ax = plt.subplots(figsize=(25, 25))
ax = plt.subplot(projection='polar')
data = months['count']
startangle = 90
colors = ['Lavender','Thistle','Plum','Orchid','Violet', 'Fuchsia','DeepPink','MediumOrchid','PaleVioletRed','MediumVioletRed','DarkViolet','Purple']

xs = [(i * pi *2)/ 100 for i in data]
ys = [-3.0, -1.0, 1.0, 3.0, 5.0, 7.0, 9.0, 11.0, 13.0, 15.0, 17.0, 19.0]
left = (startangle * pi *2)/ 360 #this is to control where the bar starts
# plot bars and points at the end to make them round
for i, x in enumerate(xs):
    ax.barh(ys[i], x, left=left, height=1.8, color=colors[i])
    ax.scatter(x+left, ys[i], s = 1150, color=colors[i], zorder=2)
    ax.scatter(left, ys[i], s=1150, color=colors[i], zorder=2)

plt.ylim(-17.8,17.8)
# legend
legend_elements =[Line2D([0], [0], marker='o', color='w', label='Jan (23.091k)', markerfacecolor='Lavender', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Feb (18.841k)', markerfacecolor='Thistle', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Mar (22.640k)', markerfacecolor='Plum', markersize=18),
```

```python
                  Line2D([0], [0], marker='o', color='w', label='Apr (18.628k)', markerfacecolor='Orchid', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='May (19.930k)', markerfacecolor='Violet', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Jun (18.755k)', markerfacecolor='Fuchsia', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Jul (21.126k)', markerfacecolor='DeepPink', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Aug (21.040k)', markerfacecolor='MediumOrchid', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Sep (19.656k)', markerfacecolor='PaleVioletRed', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Oct (19.891k)', markerfacecolor='MediumVioletRed', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Nov (17.974k)', markerfacecolor='DarkViolet', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Dec (18.106k)', markerfacecolor='Purple', markersize=18),

                  Line2D([0], [0], marker='', color='w', label='Assumed 23,091 as 100%', markersize=1),]
ax.legend(handles=legend_elements, loc='center', frameon=False,fontsize = 21)
# clear ticks, grids, spines
plt.xticks([])
plt.yticks([])
plt.title("RADIAL BAR CHART SHOWING NO. OF INCIDENTS BY MONTHS",y=1.02,fontdict = {'color':'white','fontsize':40})
ax.spines.clear()
plt.show()





for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].month) not in months['month']):
        months['month'].append(str(picked_data_master['date'][value].month))
        months['count'].append(0)
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].month) in months['month']):
        index = months['month'].index(str(picked_data_master['date'][value].month))
        months['count'][index] += 1;
quotient = []

for number in months['count']:
    x = months['month'][months['count'].index(number)]
    if x == '1':
        divisor = 5
    elif x == '2' :
        divisor = 5
    elif x == '3':
        divisor = 5
    else:
        divisor = 4
```

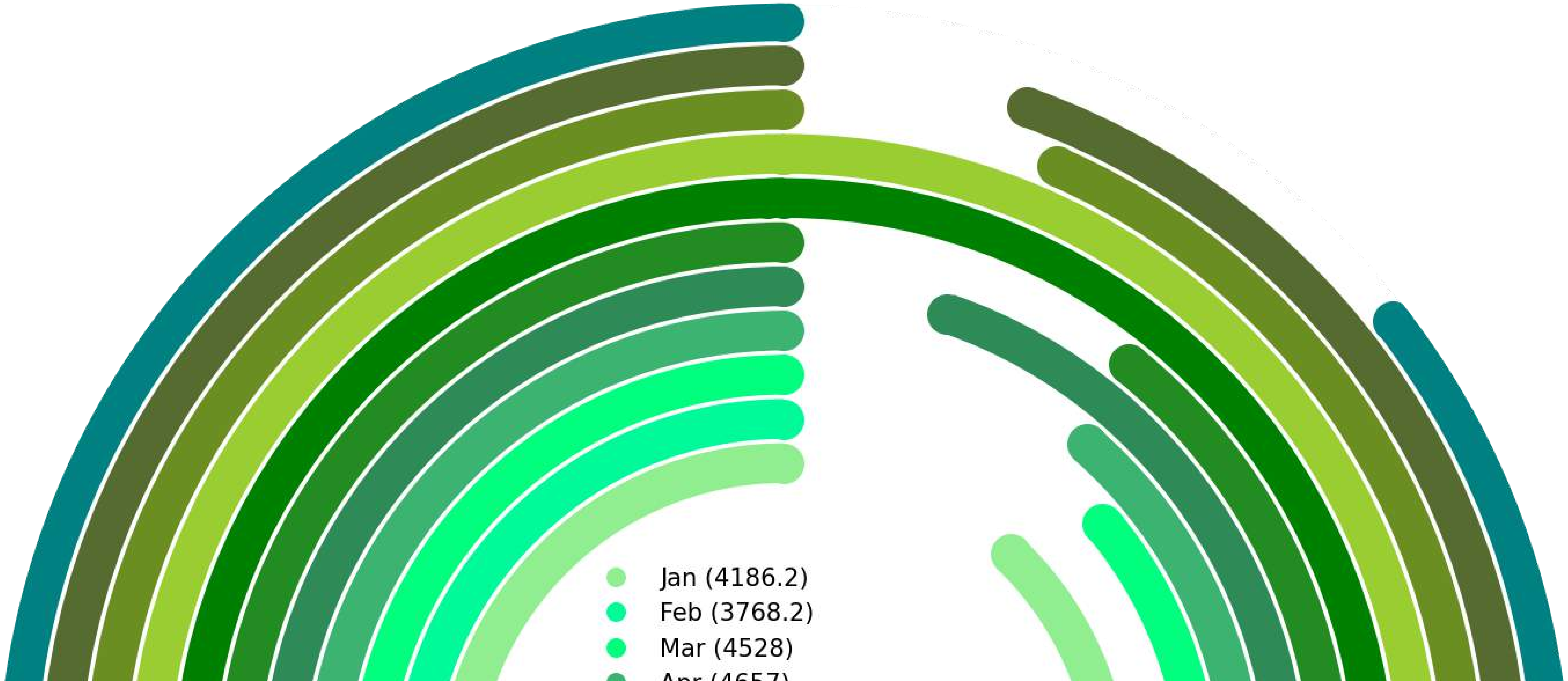
```
    quotient.append(number/divisor)

for value in range(len(quotient)):
    quotient[value] = (quotient[value]/(5281)) * 100




fig, ax = plt.subplots(figsize=(25, 25))
ax = plt.subplot(projection='polar')
data = quotient
startangle = 90
colors = ['LightGreen','MediumSpringGreen','SpringGreen','MediumSeaGreen','SeaGreen', 'ForestGreen','Green','YellowGreen','OliveDrab','DarkOliveGreen','Teal','DarkGreen']

xs = [(i * pi *2)/ 100 for i in data]
ys = [-3.0, -1.0, 1.0, 3.0, 5.0, 7.0, 9.0, 11.0, 13.0, 15.0, 17.0, 19.0]
left = (startangle * pi *2)/ 360 #this is to control where the bar starts
# plot bars and points at the end to make them round
for i, x in enumerate(xs):
    ax.barh(ys[i], x, left=left, height=1.8, color=colors[i])
    ax.scatter(x+left, ys[i], s = 1150, color=colors[i], zorder=2)
    ax.scatter(left, ys[i], s=1150, color=colors[i], zorder=2)

plt.ylim(-17.8,17.8)
# legend
legend_elements =[Line2D([0], [0], marker='o', color='w', label='Jan (4186.2)', markerfacecolor='LightGreen', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Feb (3768.2)', markerfacecolor='MediumSpringGreen', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Mar (4528)', markerfacecolor='SpringGreen', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Apr (4657)', markerfacecolor='MediumSeaGreen', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='May (4982.5)', markerfacecolor='SeaGreen', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Jun (4688.75)', markerfacecolor='ForestGreen', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Jul (5281.5)', markerfacecolor='Green', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Aug (5260)', markerfacecolor='YellowGreen', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Sep (4914)', markerfacecolor='OliveDrab', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Oct (4972.75)', markerfacecolor='DarkOliveGreen', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Nov (4493.5)', markerfacecolor='Teal', markersize=18),
                  Line2D([0], [0], marker='o', color='w', label='Dec (4526.5)', markerfacecolor='DarkGreen', markersize=18),

                  Line2D([0], [0], marker='', color='w', label='Assumed 5281.5 as 100%', markersize=1),]
ax.legend(handles=legend_elements, loc='center', frameon=False,fontsize = 21)
# clear ticks, grids, spines
plt.xticks([])
plt.yticks([])
plt.title("RADIAL BAR CHART SHOWING AVERAGE NO. OF INCIDENTS BY MONTHS",y=1.02,fontdict = {'color':'white','fontsize':38})
```

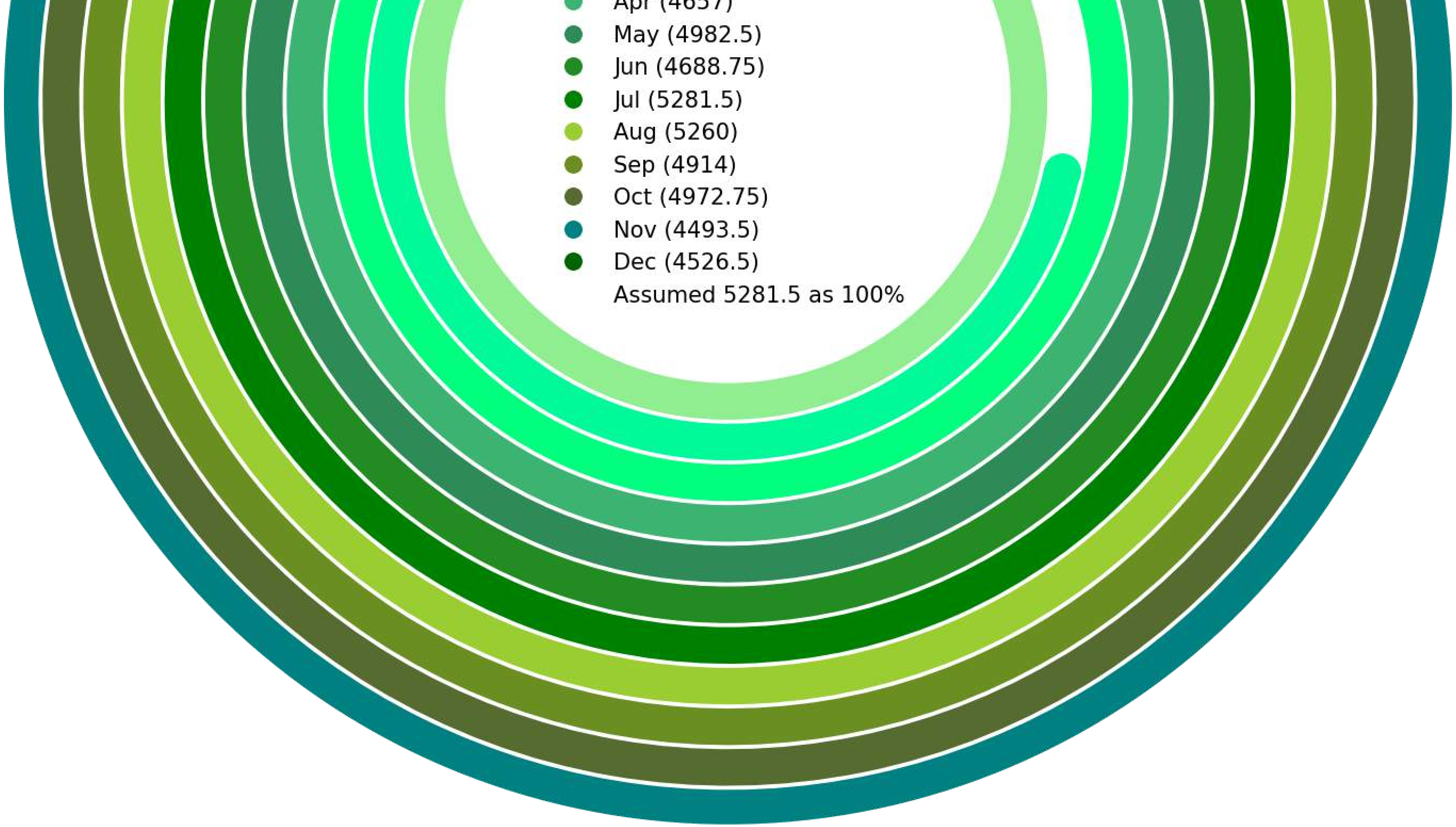
```
ax.spines.clear()
plt.show()
```





RADIAL BAR CHART SHOWING AVERAGE NO. OF INCIDENTS BY MONTHS





NOTE: Takeaways remain the same as above.

5.4 Incidents by weekdays (Usual and Average)

USUAL - We take the combined count of incidents over all the years [Jan 2014 - Mar 2015] throughout different weekdays.

AVERAGE - We take the combined count of incidents over all the years [Jan 2014 - Mar 2015] throughout different weekdays but divide each count by the number of occurences of each weekday over the years.

- ### 5.4.1 Using bar graph

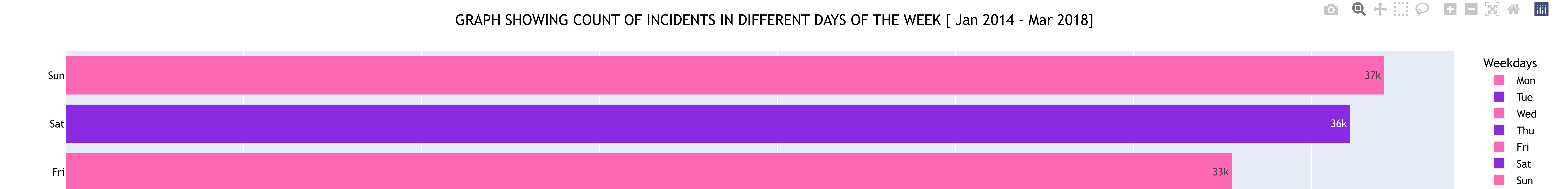
  Here we shall make use of a dictionary called *weekday*, but here we have to make an effort to sort the weekdays as the weekdays are appended to the dictionary in unsorted manner. Not only weekdays but the count associated with each weekday also has to be collectively sorted. That is why we create two functions *swapPositions()* and *bubbleSort_modified()*. These functions together help us to sort weekdays and their counts. The modified list resulting from the sorting procedure is then given names in english and the weekdays and thier counts are collectively passed onto *px.bar()* for plotting. In case of 'average' graph, the process remains same as mentioned above, the only thing is that we change is that we divide the *count* associated with each weekday with the number of times the day occurs form Jan 2014 to Mar 2018. (One might think why using sorting methodswa not required for deaaling with momnths, the answer is that the dataframe *picked_data_master* has incindents arranged datewise. So the incidents start from Jan 2013 and end in Mar 2018 in the order of dates.

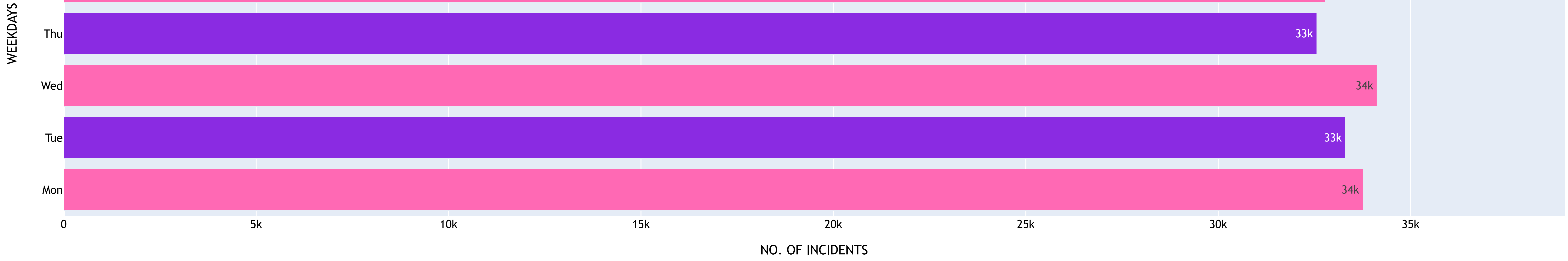
In [31]:

```
weekday = {'weekday' : [], 'count' : []}
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].weekday()) not in weekday['weekday']):
        weekday['weekday'].append(str(picked_data_master['date'][value].weekday()))
        weekday['count'].append(0)
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].weekday()) in weekday['weekday']):
        index = weekday['weekday'].index(str(picked_data_master['date'][value].weekday()))
        weekday['count'][index] += 1;

def swapPositions(lst, pos1, pos2):
    lst[pos1], lst[pos2] = lst[pos2], lst[pos1]
    return lst

def bubbleSort_modified(arr1,arr2):
    n = len(arr1)
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr1[j] > arr1[j + 1] :
                swapPositions(arr1,j,j+1)
                swapPositions(arr2,j,j+1)
integer_map = map(int, weekday['weekday'])
integer_list =  list(integer_map)
bubbleSort_modified(integer_list,weekday['count'])
string_map = map(str, integer_list)
string_list = list(string_map)

weekday['weekday'] = ['Mon','Tue','Wed','Thu','Fri','Sat','Sun']
fig = px.bar(weekday,text_auto='.2s',orientation = 'h',labels={'y':'Weekday', 'x':'no. of incidents'},y = weekday['weekday'],color_discrete_sequence = ['Hotpink','blueviolet'],
x=weekday['count'],color = weekday['weekday'])
fig.update_layout(
    title="GRAPH SHOWING COUNT OF INCIDENTS IN DIFFERENT DAYS OF THE WEEK [ Jan 2014 - Mar 2018]",
    title_x=0.5,
    yaxis_title="WEEKDAYS",
    xaxis_title="NO. OF INCIDENTS",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
```

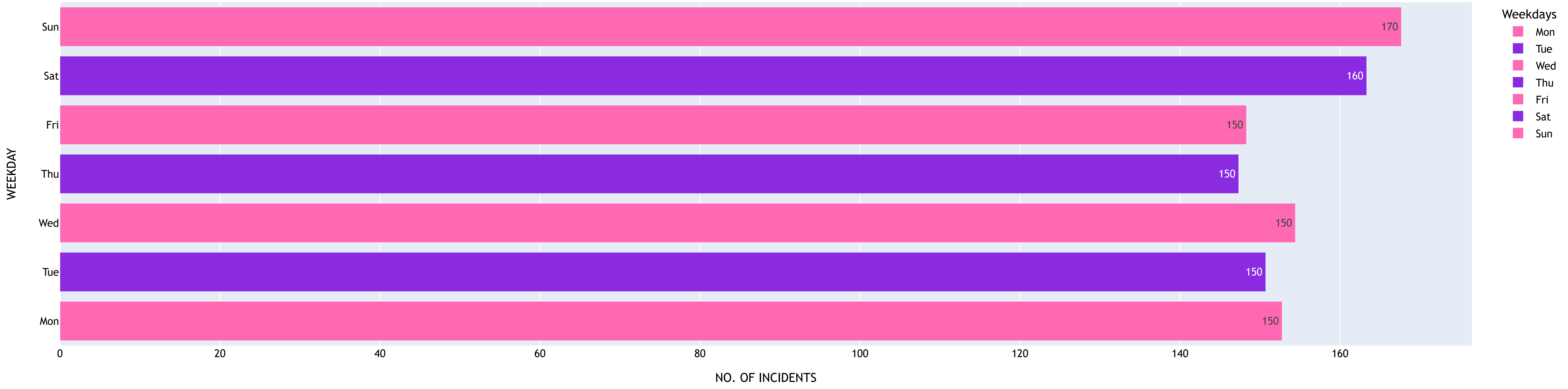
```python
    legend_title = "Weekdays"

)
weekday = {'weekday' : [], 'count' : []}
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].weekday()) not in weekday['weekday']):
        weekday['weekday'].append(str(picked_data_master['date'][value].weekday()))
        weekday['count'].append(0)
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].weekday()) in weekday['weekday']):
        index = weekday['weekday'].index(str(picked_data_master['date'][value].weekday()))
        weekday['count'][index] += 1;

#Mondays - 221
#Tuesdays - 221
#Wednesdays - 222
#Thursdays - 222
#Fridays - 222
#Saturdays - 222
#Sundays - 221
quotient = []

for number in weekday['count']:
    x = weekday['weekday'][weekday['count'].index(number)]
    if x == 0:
        divisor = 221
    elif x == 1 :
        divisor = 221
    elif x == 2:
        divisor = 222
    elif x == 3:
        divisor = 222
    elif x == 4:
        divisor = 222
    elif x == 5:
        divisor = 222
    else:
        divisor = 221

    quotient.append(number/divisor)
def swapPositions(lst, pos1, pos2):
    lst[pos1], lst[pos2] = lst[pos2], lst[pos1]
    return lst
```

```
def bubbleSort_modified(arr1,arr2):
    n = len(arr1)
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr1[j] > arr1[j + 1] :
                swapPositions(arr1,j,j+1)
                swapPositions(arr2,j,j+1)
integer_map = map(int, weekday['weekday'])
integer_list =  list(integer_map)
bubbleSort_modified(integer_list,quotient)
string_map = map(str, integer_list)
string_list = list(string_map)

weekday['weekday'] = ['Mon','Tue','Wed','Thu','Fri','Sat','Sun']
fig1 = px.bar(weekday,text_auto='.2s',orientation = 'h',labels={'y':'Weekday', 'x':'no. of incidents'},color_discrete_sequence = ['Hotpink','blueviolet'],y = weekday['weekday'], x=quotient,color =
weekday['weekday'])
fig1.update_layout(
    title="GRAPH SHOWING AVERAGE COUNT OF INCIDENTS BY DAYS OF THE WEEK [ Jan 2014 - Mar 2018]",
    title_x=0.5,
    yaxis_title="WEEKDAY",
    xaxis_title="NO. OF INCIDENTS",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
    legend_title = 'Weekdays'
)

fig.show()
fig1.show()
```





GRAPH SHOWING AVERAGE COUNT OF INCIDENTS BY DAYS OF THE WEEK [Jan 2014 - Mar 2018]



KEY TAKEAWAYS:

1. First of all notice that the 'usual' and 'average' graphs look almost the same, it is because, in the case of 'average' graph, the number with which each *count* value is divided is almost the same in all cases.
2. It is observed that saturdays and sundays have the maaximum no. of incidents and it is reasonable, considering that weekends witness people going out of their homes and spending time in public places more often than other days. Most of the subjects are also supposed to be more free during weekends as their jobs are off on that day.

- ### 5.4.2 Using radial graph

  We make an effort to sort the *weekday['weekday']* list using the same two functions as above and use matplotlib similarly as used in constructing radial graph in 'Incidents by months' case to obtain the figures below.

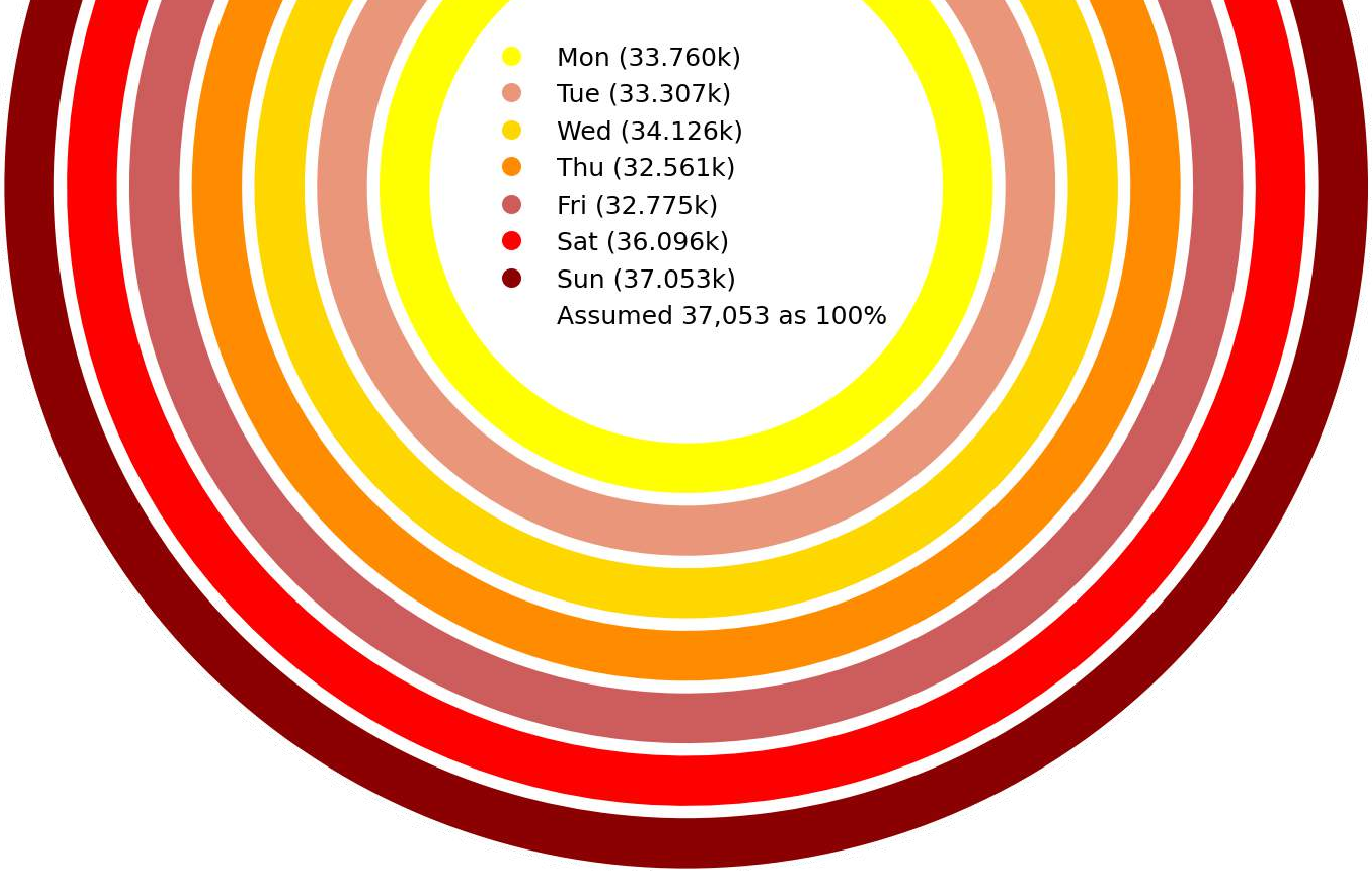
In [32]:

```
weekday = {'weekday' : [], 'count' : []}

for value in range(len(picked_data)):
    if (str(picked_data_master['date'][value].weekday()) not in weekday['weekday']):
        weekday['weekday'].append(str(picked_data_master['date'][value].weekday()))
        weekday['count'].append(0)
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].weekday()) in weekday['weekday']):
        index = weekday['weekday'].index(str(picked_data_master['date'][value].weekday()))
        weekday['count'][index] += 1;
for value in range(len(weekday['count'])):
    weekday['count'][value] = (weekday['count'][value]/(37053)) * 100
def swapPositions(lst, pos1, pos2):
    lst[pos1], lst[pos2] = lst[pos2], lst[pos1]
    return lst

def bubbleSort_modified(arr1,arr2):
    n = len(arr1)
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr1[j] > arr1[j + 1] :
                swapPositions(arr1,j,j+1)
                swapPositions(arr2,j,j+1)
integer_map = map(int, weekday['weekday'])
integer_list =  list(integer_map)
bubbleSort_modified(integer_list,weekday['count'])
string_map = map(str, integer_list)
string_list = list(string_map)

fig, ax = plt.subplots(figsize=(25, 25))
ax = plt.subplot(projection='polar')
data = weekday['count']
startangle = 90
colors = ['yellow','darksalmon','gold','darkorange','indianred', 'red','darkred']

xs = [(i * pi *2)/ 100 for i in data]
ys = [-1.0, 0.0, 1.0, 2.0, 3.0,4.0, 5.0]
left = (startangle * pi *2)/ 360 #this is to control where the bar starts
# plot bars and points at the end to make them round
for i, x in enumerate(xs):
    ax.barh(ys[i], x, left=left, height=0.8, color=colors[i])
```

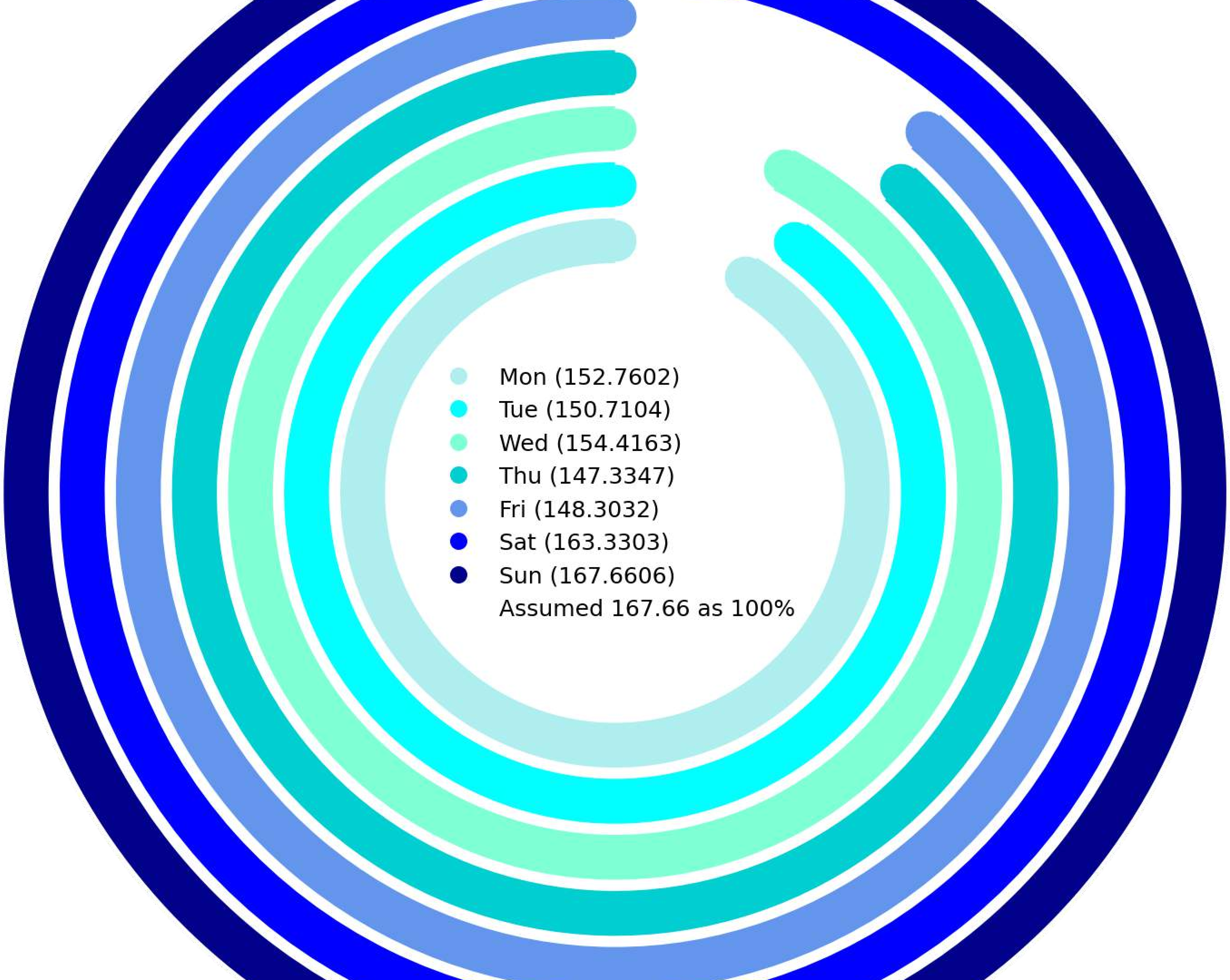
```
    ax.scatter(x+left, ys[i], s = 2100, color=colors[i], zorder=2)
    ax.scatter(left, ys[i], s=2100, color=colors[i], zorder=2)

plt.ylim(-5.5,5.5)
# legend
legend_elements = [Line2D([0], [0], marker='o', color='w', label='Mon (33.760k)', markerfacecolor='yellow', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Tue (33.307k)', markerfacecolor='darksalmon', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Wed (34.126k)', markerfacecolor='gold', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Thu (32.561k)', markerfacecolor='darkorange', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Fri (32.775k)', markerfacecolor='indianred', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Sat (36.096k)', markerfacecolor='red', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Sun (37.053k)', markerfacecolor='darkred', markersize=20),
                  Line2D([0], [0], marker='', color='w', label='Assumed 37,053 as 100%', markersize=1),]
ax.legend(handles=legend_elements, loc='center', frameon=False,fontsize = 25)
# clear ticks, grids, spines
plt.xticks([])
plt.yticks([])
plt.title("RADIAL BAR CHART SHOWING NO. OF INCIDENTS ON WEEKDAYS",y=1.02,fontdict = {'color':'white','fontsize':40})
ax.spines.clear()
plt.show()



for value in range(len(picked_data)):
    if (str(picked_data_master['date'][value].weekday()) not in weekday['weekday']):
        weekday['weekday'].append(str(picked_data_master['date'][value].weekday()))
        weekday['count'].append(0)
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].weekday()) in weekday['weekday']):
        index = weekday['weekday'].index(str(picked_data_master['date'][value].weekday()))
        weekday['count'][index] += 1;

quotient = []

for number in weekday['count']:
    x = weekday['weekday'][weekday['count'].index(number)]
    if x == 0:
        divisor = 221
    elif x == 1 :
        divisor = 221
    elif x == 2:
        divisor = 222
    elif x == 3:
```

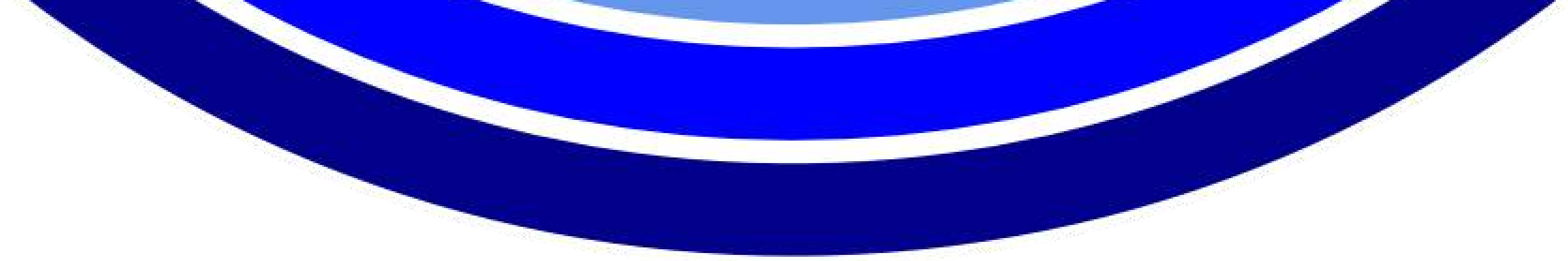
```
        divisor = 222
    elif x == 4:
        divisor = 222
    elif x == 5:
        divisor = 222
    else:
        divisor = 221

    quotient.append(number/divisor)
for value in range(len(weekday['count'])):
    quotient[value] = (quotient[value]/(167.66)) * 100

def swapPositions(lst, pos1, pos2):
    lst[pos1], lst[pos2] = lst[pos2], lst[pos1]
    return lst

def bubbleSort_modified(arr1,arr2):
    n = len(arr1)
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr1[j] > arr1[j + 1] :
                swapPositions(arr1,j,j+1)
                swapPositions(arr2,j,j+1)
integer_map = map(int, weekday['weekday'])
integer_list =  list(integer_map)
bubbleSort_modified(integer_list,quotient)
string_map = map(str, integer_list)
string_list = list(string_map)

fig, ax = plt.subplots(figsize=(25, 25))
ax = plt.subplot(projection='polar')
data = quotient
startangle = 90
colors = ['PaleTurquoise','aqua','Aquamarine','DarkTurquoise','CornflowerBlue','Blue','DarkBlue']

xs = [(i * pi *2)/ 100 for i in data]
ys = [-1.0, 0.0, 1.0, 2.0, 3.0,4.0, 5.0]
left = (startangle * pi *2)/ 360 #this is to control where the bar starts
# plot bars and points at the end to make them round
for i, x in enumerate(xs):
    ax.barh(ys[i], x, left=left, height=0.8, color=colors[i])
```

```python
    ax.scatter(x+left, ys[i], s = 2000, color=colors[i], zorder=2)
    ax.scatter(left, ys[i], s=2000, color=colors[i], zorder=2)

plt.ylim(-5.5,5.5)
# legend
legend_elements = [Line2D([0], [0], marker='o', color='w', label='Mon (152.7602)', markerfacecolor='PaleTurquoise', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Tue (150.7104)', markerfacecolor='aqua', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Wed (154.4163)', markerfacecolor='Aquamarine', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Thu (147.3347)', markerfacecolor='DarkTurquoise', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Fri (148.3032)', markerfacecolor='CornflowerBlue', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Sat (163.3303)', markerfacecolor='Blue', markersize=20),
                  Line2D([0], [0], marker='o', color='w', label='Sun (167.6606)', markerfacecolor='DarkBlue', markersize=20),
                  Line2D([0], [0], marker='', color='w', label='Assumed 167.66 as 100%', markerfacecolor='Black', markersize=1),]
ax.legend(handles=legend_elements, loc='center', frameon=False,fontsize = 25)
# clear ticks, grids, spines
plt.xticks([])
plt.yticks([])
plt.title("RADIAL BAR CHART SHOWING AVERAGE NO. OF INCIDENTS ON WEEKDAYS",y=1.02,fontdict = {'color':'white','fontsize':37})
ax.spines.clear()
plt.show()
```



RADIAL BAR CHART SHOWING AVERAGE NO. OF ACCIDENTS ON WEEKDAYS


Mon (152.7602)
Tue (150.7104)
Wed (154.4163)
Thu (147.3347)
Fri (148.3032)
Sat (163.3303)
Sun (167.6606)
Assumed 167.66 as 100%



NOTE: Takeaways remain the same as above.

5.5 Incidents by days of the months (Usual and Average)

USUAL - We take the combined count of incidents over all the years [Jan 2014 - Mar 2015] throughout different days of the months.

AVERAGE - We take the combined count of incidents over all the years [Jan 2014 - Mar 2015] throughout different months but divide each count by the numnber of occurences of each day of the month over the years.

- ### 5.5.1 Using bar graph

We use dictionary *day* and run a loop over *picked_data_master* to gather key and value pairs as shown. We use *swapPositions()* and *bubbleSort_modified()* to sort our values as before and pass the dictionary to *px.bar()*. In case of 'average', we repaeat the process with the only difference that we divide the *count* values with the number of occurences of days of the month.

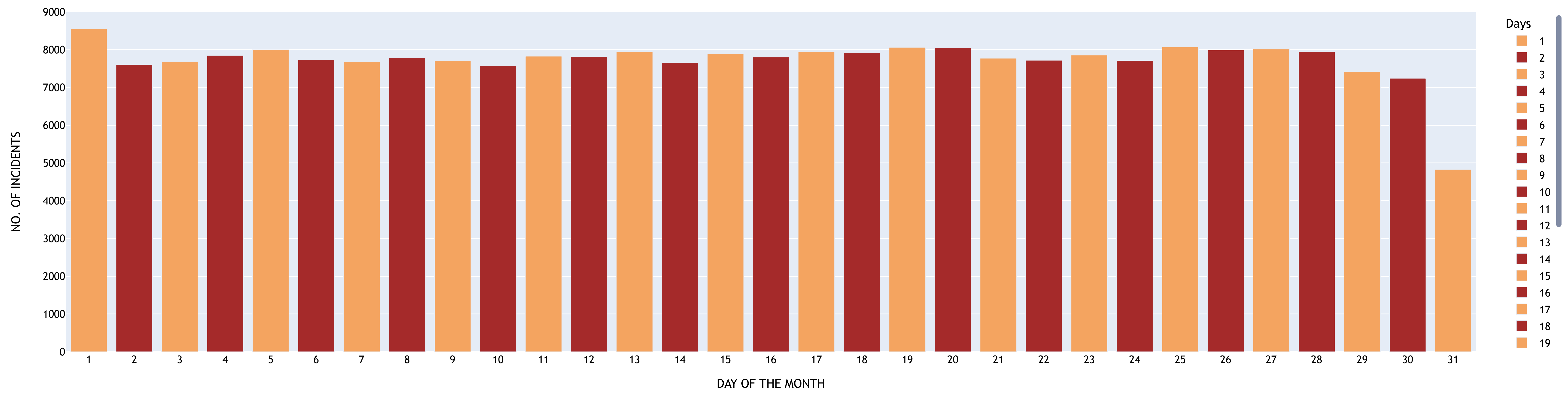
In [33]:

```
day = {'day' : [], 'count' : []}

for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].day) not in day['day']):
        day['day'].append(str(picked_data_master['date'][value].day))
        day['count'].append(0)
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].day) in day['day']):
        index = day['day'].index(str(picked_data_master['date'][value].day))
        day['count'][index] += 1;

def swapPositions(lst, pos1, pos2):
    lst[pos1], lst[pos2] = lst[pos2], lst[pos1]
    return lst

def bubbleSort_modified(arr1,arr2):
    n = len(arr1)
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr1[j] > arr1[j + 1] :
                swapPositions(arr1,j,j+1)
                swapPositions(arr2,j,j+1)

integer_map = map(int, day['day'])
```

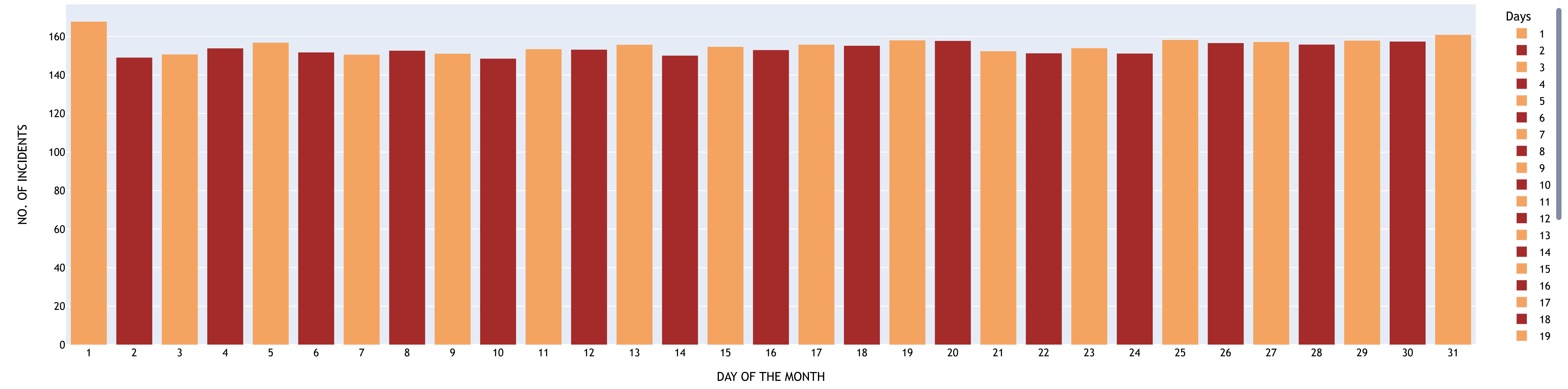
```
integer_list =  list(integer_map)
bubbleSort_modified(integer_list,day['count'])
string_map = map(str, integer_list)
string_list = list(string_map)



fig = px.bar(day,labels={'x':'Day', 'y':'no. of incidents'},x = string_list, y=day['count'],color_discrete_sequence = ['sandybrown','brown'], color = string_list)
fig.update_layout(
    title="GRAPH SHOWING COUNT OF INCIDENTS IN DIFFERENT DAYS OF THE MONTH [ Jan 2014 - Mar 2018 ]",
    title_x=0.5,
    xaxis_title="DAY OF THE MONTH",
    yaxis_title="NO. OF INCIDENTS",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
    legend_title = 'Days'

)
fig.show()

day = {'day' : [], 'count' : []}

for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].day) not in day['day']):
        day['day'].append(str(picked_data_master['date'][value].day))
        day['count'].append(0)
for value in range(len(picked_data_master)):
    if (str(picked_data_master['date'][value].day) in day['day']):
        index = day['day'].index(str(picked_data_master['date'][value].day))
        day['count'][index] += 1;

x = [str(num) for num in range(1,29)]
quotient = []
for d in day['day']:
    if(d in x):
        divisor = 51
    elif(d == '29'):
        divisor = 47
    elif(d == '30'):
        divisor = 46
```

```
    else:
        divisor = 30
    quotient.append(day['count'][day['day'].index(d)]/divisor)


def swapPositions(lst, pos1, pos2):
    lst[pos1], lst[pos2] = lst[pos2], lst[pos1]
    return lst

def bubbleSort_modified(arr1,arr2):
    n = len(arr1)
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr1[j] > arr1[j + 1] :
                swapPositions(arr1,j,j+1)
                swapPositions(arr2,j,j+1)

integer_map = map(int, day['day'])
integer_list =  list(integer_map)
bubbleSort_modified(integer_list,quotient)
string_map = map(str, integer_list)
string_list = list(string_map)



fig = px.bar(day,labels={'x':'Day', 'y':'no. of incidents'},x = string_list, y=quotient,color_discrete_sequence = ['sandybrown','brown'],color = string_list)
fig.update_layout(
    title="GRAPH SHOWING AVERAGE COUNT OF INCIDENTS BY DAYS OF THE MONTH [ Jan 2014 - Mar 2018 ]",
    title_x=0.5,
    xaxis_title="DAY OF THE MONTH",
    yaxis_title="NO. OF INCIDENTS",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
    legend_title = 'Days'

)
fig.show()
```

GRAPH SHOWING COUNT OF INCIDENTS IN DIFFERENT DAYS OF THE MONTH [Jan 2014 - Mar 2018]



GRAPH SHOWING AVERAGE COUNT OF INCIDENTS BY DAYS OF THE MONTH [Jan 2014 - Mar 2018]



KEY TAKEAWAYS:

1. The only significant difference between 'usual' and 'average' graphs is that 31st in 'usual' graph has considerably lower values than other days. It is because the number of times 31 occurs is almost half the times other days occur. When average is taken into account, we see that all the days have similar number of incidents. So nothing unusual here.

5.6 Analysis using Time Series

We use a Time Series to understand how the distribution of incidents, people killed and people injured is throughout any given year is, be it 2014, 2015, 2016 or 2017.

First we create 4 dictionaries, one for each year. Then, we create a new dataframe called *TimeSeries* from *picked_data_master*. We fill each of dictionaries with values using a for loop over *TimeSeries* and then create new dataframes (4 in number, one for each year) using dictionaries. We group each of these dataframes by 'Date' so as to sum up incidents, no. of people killed and no. of people injured using *.agg()* function. The newly created resulting dataframes are then used to plot the time series for each year as shown below.

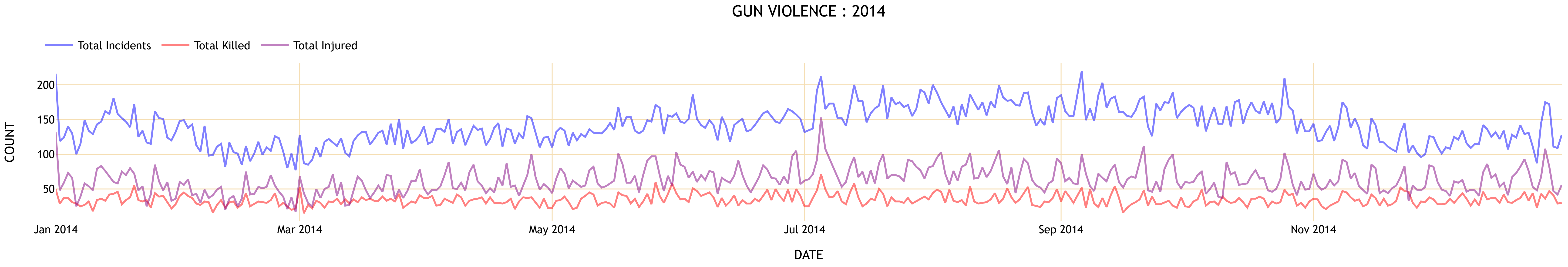
In [34]:

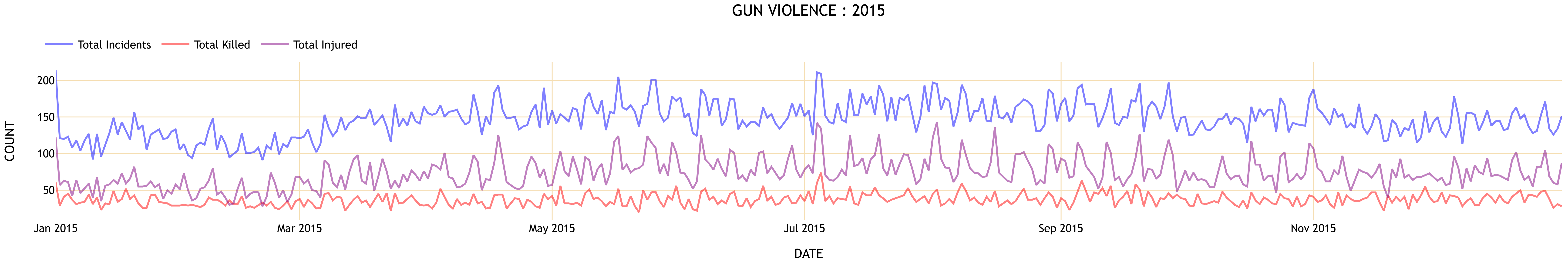
```
TimeSeries = pd.DataFrame()
t2014  = {}
t2015 = {}
t2016 = {}
t2017 = {}
i = 0

TimeSeries = picked_data_master[['date','n_killed','n_injured']].copy()

for entry in tqdm(range(len(TimeSeries))):
    if(TimeSeries['date'][i].year == 2014):
        t2014[i] = {"Date": TimeSeries['date'][i], "Killed": TimeSeries['n_killed'][i], "Injured":TimeSeries['n_injured'][i], "Incidents":1}
    elif(TimeSeries['date'][i].year == 2015):
        t2015[i] = {"Date": TimeSeries['date'][i], "Killed": TimeSeries['n_killed'][i], "Injured":TimeSeries['n_injured'][i], "Incidents":1}

    elif(TimeSeries['date'][i].year == 2016):
        t2016[i] = {"Date": TimeSeries['date'][i], "Killed": TimeSeries['n_killed'][i], "Injured":TimeSeries['n_injured'][i], "Incidents":1}

    elif(TimeSeries['date'][i].year == 2017):
        t2017[i] = {"Date": TimeSeries['date'][i], "Killed": TimeSeries['n_killed'][i], "Injured":TimeSeries['n_injured'][i], "Incidents":1}



    i = i+1

TimeSeries_2014 = pd.DataFrame.from_dict(t2014,'index')
TimeSeries_2015 = pd.DataFrame.from_dict(t2015,'index')
TimeSeries_2016 = pd.DataFrame.from_dict(t2016,'index')
TimeSeries_2017 = pd.DataFrame.from_dict(t2017,'index')

TimeSeries_2014 = TimeSeries_2014.groupby(['Date']).agg('sum')
TimeSeries_2014.reset_index(level = 0,inplace=True)
TimeSeries_2015 = TimeSeries_2015.groupby(['Date']).agg('sum')
TimeSeries_2015.reset_index(level=0, inplace=True)
TimeSeries_2016 = TimeSeries_2016.groupby(['Date']).agg('sum')
```

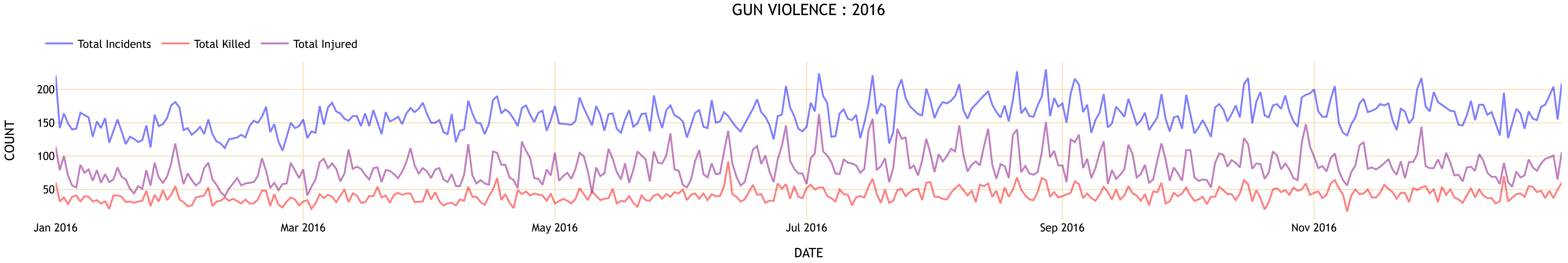
```
TimeSeries_2016.reset_index(level=0, inplace=True)
TimeSeries_2017 = TimeSeries_2017.groupby(['Date']).agg('sum')
TimeSeries_2017.reset_index(level=0, inplace=True)
########################################2014
trace1 = go.Scatter(opacity = 0.50,x = TimeSeries_2014['Date'], y =  TimeSeries_2014['Incidents'], name='Total Incidents', mode = "lines", marker = dict(color = 'Blue'))
trace2 = go.Scatter(opacity = 0.50,x = TimeSeries_2014['Date'], y = TimeSeries_2014['Killed'], name="Total Killed", mode = "lines", marker = dict(color = 'Red'))
trace3 = go.Scatter(opacity = 0.50,x = TimeSeries_2014['Date'], y = TimeSeries_2014['Injured'], name="Total Injured", mode = "lines", marker = dict(color = 'Purple'))

data = [trace1, trace2, trace3]
layout = dict(height=350,hovermode = 'x', title = 'GUN VIOLENCE : 2014',title_x=0.5,\
legend=dict(orientation="h", x=-.01, y=1.2),plot_bgcolor='rgba(0,0,0,0)',yaxis = dict(showgrid=True,gridcolor='Wheat'),xaxis= dict(title='Date',\
       ticklen= 1,showgrid=True,gridcolor='Wheat'),\
       yaxis_title="COUNT",
       xaxis_title="DATE",
     font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ))
fig = dict(data = data, layout = layout)

iplot(fig)
########################################2015
trace1 = go.Scatter(opacity = 0.50,x = TimeSeries_2015['Date'], y =  TimeSeries_2015['Incidents'], name='Total Incidents', mode = "lines", marker = dict(color = 'Blue'))
trace2 = go.Scatter(opacity = 0.50,x = TimeSeries_2015['Date'], y = TimeSeries_2015['Killed'], name="Total Killed", mode = "lines", marker = dict(color = 'Red'))
trace3 = go.Scatter(opacity = 0.50,x = TimeSeries_2015['Date'], y = TimeSeries_2015['Injured'], name="Total Injured", mode = "lines", marker = dict(color = 'Purple'))

data = [trace1, trace2, trace3]
layout = dict(height=350,hovermode = 'x',title = 'GUN VIOLENCE : 2015',title_x=0.5,plot_bgcolor='rgba(0,0,0,0)', legend=dict(orientation="h", \
        x=-.01, y=1.2), yaxis = dict(showgrid=True,gridcolor='Wheat'),xaxis= dict(title='Date', ticklen= 1,\
        showgrid=True,gridcolor='Wheat'),
       yaxis_title="COUNT",
       xaxis_title="DATE",
     font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ))
fig = dict(data = data, layout = layout)

iplot(fig)

########################################2016
```

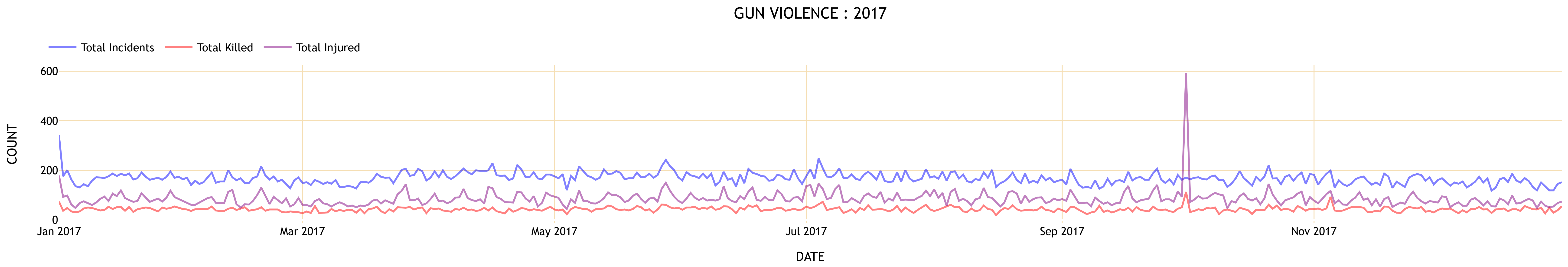
```
trace1 = go.Scatter(opacity = 0.50,x = TimeSeries_2016['Date'], y =  TimeSeries_2016['Incidents'], name='Total Incidents', mode = "lines", marker = dict(color = 'Blue'))
trace2 = go.Scatter(opacity = 0.50,x = TimeSeries_2016['Date'], y = TimeSeries_2016['Killed'], name="Total Killed", mode = "lines", marker = dict(color = 'Red'))
trace3 = go.Scatter(opacity = 0.50,x = TimeSeries_2016['Date'], y = TimeSeries_2016['Injured'], name="Total Injured", mode = "lines", marker = dict(color = 'Purple'))

data = [trace1, trace2, trace3]
layout = dict(height=350,hovermode = 'x',title = 'GUN VIOLENCE : 2016',\
        title_x=0.5, legend=dict(orientation="h", x=-.01, y=1.2), \
        yaxis = dict(showgrid=True,gridcolor='Wheat'),plot_bgcolor='rgba(0,0,0,0)',\
        xaxis= dict(title='Date', ticklen= 1,showgrid=True,gridcolor='Wheat'),\
     yaxis_title="COUNT",
       xaxis_title="DATE",
     font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ))
fig = dict(data = data, layout = layout)

iplot(fig)

########################################2017
trace1 = go.Scatter(opacity = 0.50,x = TimeSeries_2017['Date'], y =  TimeSeries_2017['Incidents'], name='Total Incidents', mode = "lines", marker = dict(color = 'Blue'))
trace2 = go.Scatter(opacity = 0.50,x = TimeSeries_2017['Date'], y = TimeSeries_2017['Killed'], name="Total Killed", mode = "lines", marker = dict(color = 'Red'))
trace3 = go.Scatter(opacity = 0.50,x = TimeSeries_2017['Date'], y = TimeSeries_2017['Injured'], name="Total Injured", mode = "lines", marker = dict(color = 'Purple'))

data = [trace1, trace2, trace3]
layout = dict(height=350,hovermode = 'x',\
              title = 'GUN VIOLENCE : 2017',title_x=0.5, \
              legend=dict(orientation="h", x=-.01, y=1.2), \
              yaxis = dict(showgrid=True,gridcolor='Wheat'),\
              plot_bgcolor='rgba(0,0,0,0)',\
              xaxis= dict(title='Date', ticklen= 1,\
            showgrid=True,gridcolor='Wheat'),
              yaxis_title="COUNT",
           xaxis_title="DATE",
      font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ))
fig = dict(data = data, layout = layout)

iplot(fig)
```

```
100%|██████████| 239678/239678 [00:33<00:00, 7260.65it/s]
```









KEY TAKEAWAYS:

1. 2014: July 5th (212 incidents), September 6th (230 incidents) and October 25th (210 incidents) mark the maximum number of gun violence incidents.
2. 2015: May 17th (205 incidents), May 25th (201 incidents), May 26th (201 incidents), July 4th (211 incidents) and July 5th (209 incidents) mark the maximum number of gun violence incidents.
3. 2016: July 4th (224 incidents), July 17th (221 incidents), August 21st (227 incidents) and August 28th (230 incidents) mark the maximum number of gun violence incidents.
4. 2017: April 16th (229 incidents), May 28th (242 incidents), July 4th (248 incidents) and October 21 (220 incidents) mark the maximum number of gun violence incidents (One is tempted to add the Oct 1st Las Vegas incident in this list but note that the number of incidents on that day is 171 which is considerably lower than other dates mentioned).
5. Note that 4th July is Independence Day, September 5th is Labour Day and Last Monday of May every year is Memorial Day. This explains the occurence of certain dates in the dates mentioned above.
6. July 4th is the most dangerous of all dates as it makes it in every single point above. Not only for gun violence incidents but for other incidemts also, 4th of July is very dangerous. Here is why.

5.7 Plotting different variables over the years

Here we plot a bar graph that covers different aspects of gun violence incidents over the years. It includes no. of incidents, people involved, killed, injured, unharmed, no. of people who were subjects, victims, no. of people having gender as males, females, no. of people whose gender could not be identified, no. of people who were arrested and no. of people who were supposed to be arrested but were not arrested. We shall use a dictionary named *years* here and iterate over *years['Years']* to append values to various lists associated with different keys of *years* as shown below. For each of these aspects, we separately call *go.Bar()* and assign them to *trace* (*trace1*, *trace2*,..). After chooseing *data* and *layout* as required we plot the bar graph which consists of several bars grouped togther based on the year.

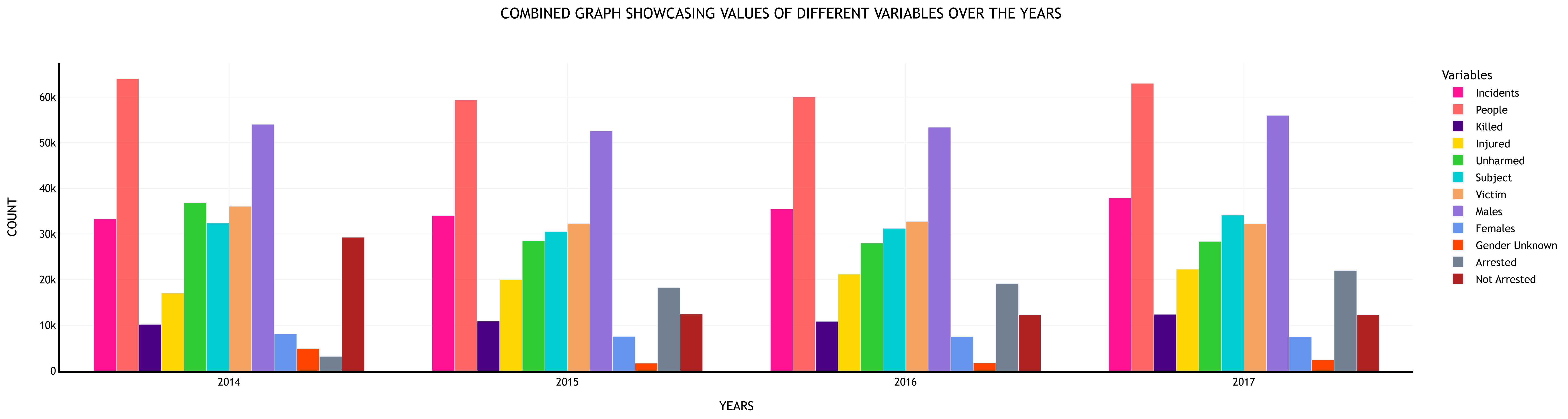
In [35]:
```
years = {'Years':['2014','2015','2016','2017'], 'Count1':[], 'Count2': [], 'Count3':[], 'Count4':[], 'Count5':[],'Count6':[],'Count7':[],'Count8':[],'Count9':[],
         'Count10':[],'Count11':[],'Count12':[]}

for x in years['Years']:
    count1 = 0
    count2 = 0
    count3 = 0
    count4 = 0
    count5 = 0
    count6 = 0
    count7 = 0
    count8 = 0
    count9 = 0
    count10 = 0
```

```
    count11 = 0
    count12 = 0
    for value in range(len(picked_data)):
        if(str(picked_data['date'][value].year) == x):
            count1 = count1 + 1
    years['Count1'].append(count1)
    for value in range(len(picked_data)):
        if(str(picked_data['date'][value].year) == x):
            count2 = count2 + picked_data['TPeople'][value]
            count3 = count3 + picked_data['n_killed'][value]
            count4 = count4 + picked_data['n_injured'][value]
            count5 = count5 + picked_data['TUnharmed'][value]
            count6 = count6 + picked_data['TSubject'][value]
            count7 = count7 + picked_data['TVictim'][value]
            count8 = count8 + picked_data['TMales'][value]
            count9 = count9 + picked_data['TFemales'][value]
            count10 = count10 + picked_data['TUnknown_gender'][value]
            count11 = count11 + picked_data['TArrested'][value]
            count12 = count12 + picked_data['TNarrested'][value]

    years['Count2'].append(count2)
    years['Count3'].append(count3)
    years['Count4'].append(count4)
    years['Count5'].append(count5)
    years['Count6'].append(count6)
    years['Count7'].append(count7)
    years['Count8'].append(count8)
    years['Count9'].append(count9)
    years['Count10'].append(count10)
    years['Count11'].append(count11)
    years['Count12'].append(count12)



trace1 = go.Bar(
   x = years['Years'],
   y = years['Count1'],
   name = 'Incidents',
   marker_color='deeppink',

)
trace2 = go.Bar(
   x = years['Years'],
```

```
   y = years['Count2'],
   name = 'People',
   marker_color='red',
    opacity=0.60
)
trace3 = go.Bar(
   x = years['Years'],
   y = years['Count3'],
   name = 'Killed',
    marker_color='indigo',

)
trace4 = go.Bar(
   x = years['Years'],
   y = years['Count4'],
   name = 'Injured',
    marker_color='gold',

)
trace5 = go.Bar(
   x = years['Years'],
   y = years['Count5'],
   name = 'Unharmed',
    marker_color='limegreen',

)
trace6 = go.Bar(
   x = years['Years'],
   y = years['Count6'],
   name = 'Subject',
    marker_color='darkturquoise',

)

trace7 = go.Bar(
   x = years['Years'],
   y = years['Count7'],
   name = 'Victim',
    marker_color='sandybrown',

)
trace8 = go.Bar(
   x = years['Years'],
```

```
   y = years['Count8'],
   name = 'Males',
    marker_color='mediumpurple',

)
trace9 = go.Bar(
   x = years['Years'],
   y = years['Count9'],
   name = 'Females',
    marker_color='cornflowerblue',

)
trace10= go.Bar(
   x = years['Years'],
   y = years['Count10'],
   name = 'Gender Unknown',
    marker_color='orangered',

)
trace11 = go.Bar(
   x = years['Years'],
   y = years['Count11'],
   name = 'Arrested',
    marker_color='slategray',

)
trace12 = go.Bar(
   x = years['Years'],
   y = years['Count12'],
   name = 'Not Arrested',
    marker_color='firebrick',

)

data = [trace1, trace2, trace3, trace4, trace5,trace6,trace7, trace8, trace9, trace10, trace11,trace12 ]
layout = go.Layout(barmode = 'group')
fig = go.Figure(data = data, layout = layout)
fig.update_layout(
    title="COMBINED GRAPH SHOWCASING VALUES OF DIFFERENT VARIABLES OVER THE YEARS ",
    title_x=0.5,
    xaxis_title="YEARS",
    yaxis_title="COUNT",
    font=dict(
```

```
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
    legend_title_text = "Variables",
    plot_bgcolor='white',
    )
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', gridcolor ='WhiteSmoke')
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', gridcolor ='WhiteSmoke')
iplot(fig)
```



KEY TAKEAWAYS:

1. The number of incidents is increasing every year.
2. The year 2014 saw a staggering amount of people involved in incidents, an amount which is much more than next 3 years. From 2015 onwards, there is a steady increase in the number of people involved.
3. The number of people being killed or injured has been increasing every year (with the exception of 2015->2016 which saw a marginal decrease in the number of people who were killed).
4. The number of unharmed people was maximum in 2014 and more or less the same in the following years.
5. Subjects were max in 2017 and more or less same for other three years.
6. Victims were max in 2014 and almost same every other year.
7. The number of males and females is almost same every year; people whose gender was not known remains highest in 2014 (almost double than every other year).
8. The number of people who were arrested is very less in the year 2014 compared to other years, probably because the data for that year is incomplete. It is for this reason that the people who are subjects but not arrested is quite high for 2014 (almost 2.5x than other years). For 2015, 2016 and 2017, number of people who were arrested has been increasing year after year but the same for unarrested people remains constant throughout.

6.DATA ANALYSIS - LOCATION

We shall now analyse our data and look out for location trends.

6.1 Incidents in different states (Usual and Population Adjusted)

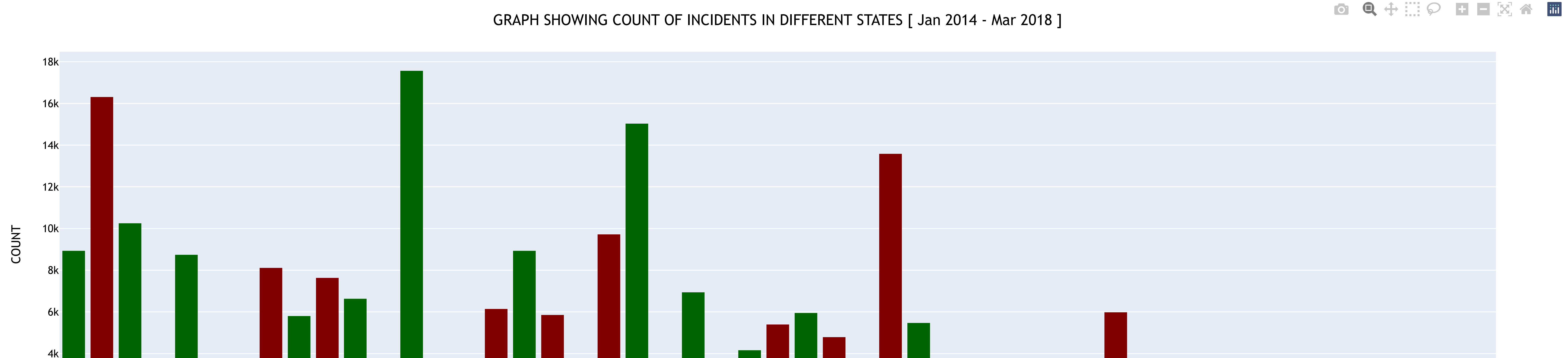
USUAL - We take the combined count of incidents over all the states. [Jan 2014 - Mar 2018]

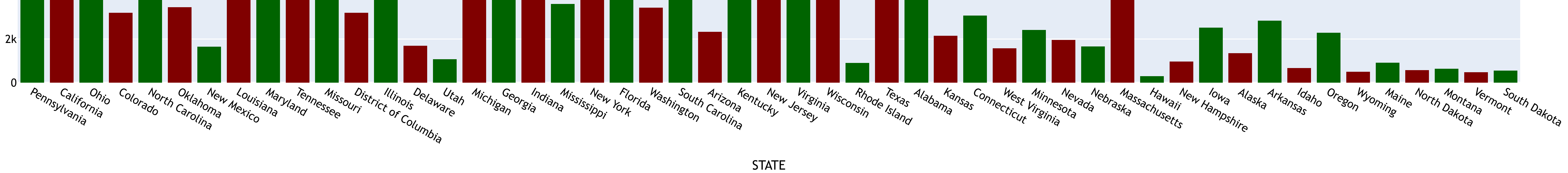
POPULATION ADJUSTED - We take the combined count of incidents over all the states but multiply each count by the number obtained by dividing the population of each state by 100,000. [Jan 2014 - Mar 2018]

- ### 6.1.1 Usual : Using histogram

  A very simple plot wherein we straightaway pass *picked_data_master['state']* to px.histogram() and just let it do all the work.

In [36]:
```
fig = px.histogram(picked_data_master['state'],x = 'state',color_discrete_sequence = ['darkgreen','maroon'], color = 'state',height = 600)
fig.update_layout(
    title="GRAPH SHOWING COUNT OF INCIDENTS IN DIFFERENT STATES [ Jan 2014 - Mar 2018 ]",
    title_x=0.5,
    xaxis_title="STATE",
    yaxis_title="COUNT",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
    showlegend = False,

)
fig.show()
```





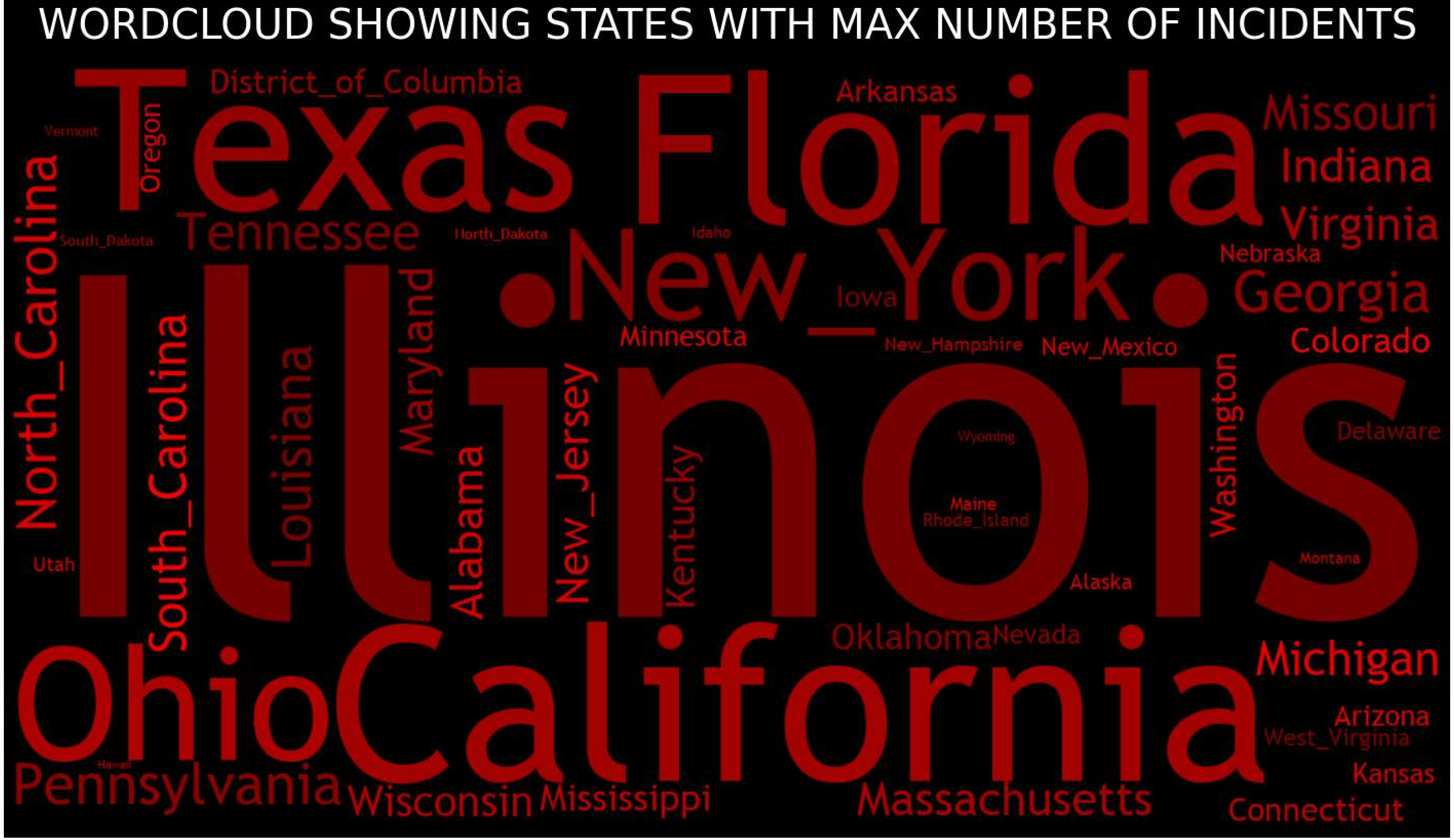
KEY TAKEAWAYS

1. Illionis, California, Florida and Texas stand out be the top 4 states (in the same order) when it comes to the number of gun violence incidents from Jan 2014 to Mar 2018.
2. This kind of analysis is misleading as it is not taking into account the population of each state. A population adjusted research is required in order to determine the most dangerous state.

- ### 6.1.2 Usual : Using wordcloud

  The most important function here to use is *WordCloud()*, this comes from *wordcloud* library. The function WordCloud(), most importantly, takes in a string and then presents it as a word cloud by splitting that one single string by spaces. We cannot use it effectively unless we bind the names of states together that have more than one single word in their name, for eaxmple, North Carolina. Thus we take each and every state name, find if there is any "space" in their name and replace it with an underscore (_). One more issue is that the default settings for *WordCloud()* is to present words with random colors (depending on the background color) and in my case I wanted to have red on black and thus I have employed *random_color_func()* as shown below. The *random_color_func()* is creating colours of the same hue but with different luminosities. You just have to decide which hue you want. Use color picker to get hsl values.

In [37]:
```
word_list = picked_data_master['state'].to_list()
state_list = []
for word in word_list:
    state_list.append(word.replace(" ", "_"))
state_list = " ".join(state_list)
def random_color_func(word=None, font_size=None, position=None,  orientation=None, font_path=None, random_state=None):
    h = int(360.0 * 0.0 / 255.0)
    s = int(100.0 * 255.0 / 255.0)
    l = int(100.0 * float(random_state.randint(60, 120)) / 255.0)

    return "hsl({}, {}%, {}%)".format(h, s, l)
fig = WordCloud(font_path = 'Trebuchet MS',collocations = False, background_color = 'Black',width=1500,color_func=random_color_func,max_font_size = 500, height=800).generate(state_list)
plt.figure(figsize=[25,25],facecolor='k')
plt.axis("off");
plt.title("WORDCLOUD SHOWING STATES WITH MAX NUMBER OF INCIDENTS",y=1.03,fontdict = {'color':'white','fontsize':40})
plt.imshow(fig, interpolation='bilinear')
plt.show()
```



KEY TAKEAWAYS

1. The size of Illionis is the greatest and thus this is the state with maximum number of incidents, followed by California, Florida, Texas and others.
2. As before, we conclude that this kind of analysis is misleading as it is not taking into account the population of each state. A population adjusted research is required in order to determine the most dangerous state.

- ### 6.1.3 Population Adjusted: Using Choropleth map

We use choropleth map in order to showcase those states that have very high number of incidents...but this time it is different as we will make a 'population adjusted' analysis. For that, we need to get the population data for each state and also obtain S tate code for each state so as to plot Choropleth map. The population data was obtained from here and State codes from here. We extract data from 'state-population.csv' and create a dataframe called *PD* whic has both population data as well as state codes.

Next, we create another dataframe called *PA_States* and have the following columns in it:

- 'pop' for population (from *PD*)
- 'Code' for State Code (from *PD*)
- 'state' for State name (from *picked_data_master*)
- 'n_killed' fro no. of people killed (from *picked_data_master*)
- 'n_injured' for no. of people injured (from *picked_data_master*)
- 'Count' for no. of incidents in each state
- 'n_killed_adj' for number of people killed (but population adjusted)
- 'n_injured_adj' for number of people injured (but population adjusted)
- 'count_adj' for count of incidents in each state (but count adjusted)

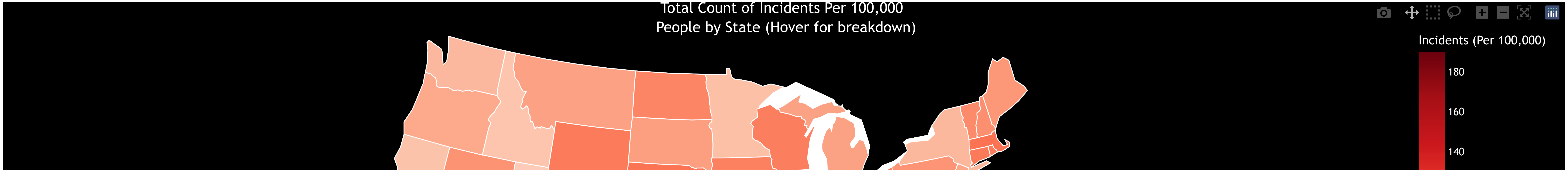
After calculating the values of *'n_killed_adj'*, *'n_injured_adj'* and *'count_adj'*, we convert the datatypes of all the columns of *PA_States* into string and create our hover-text. Finally, we plot our map using *go.Choropleth()* and visualise our data. We then also create a similar map but the determining factor is the no. of people who were killed and injured per 100,000 people (collectively known as 'total loss').

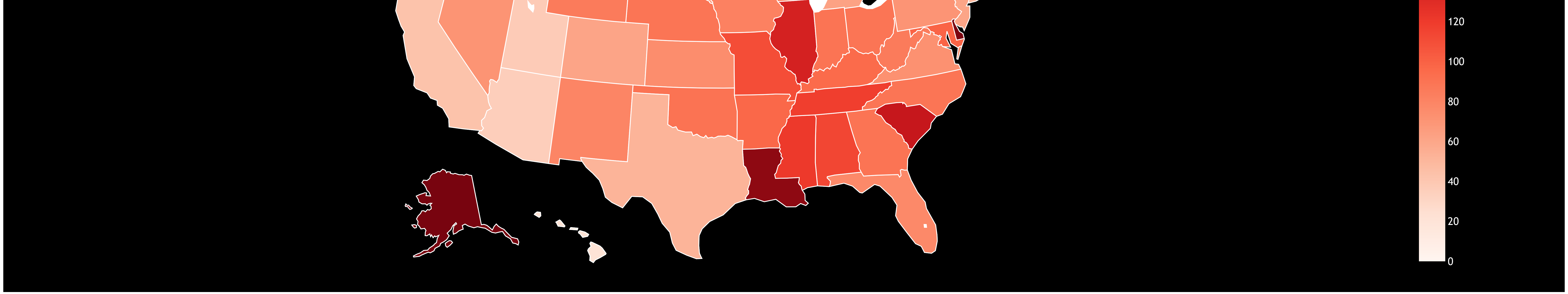
In [38]:

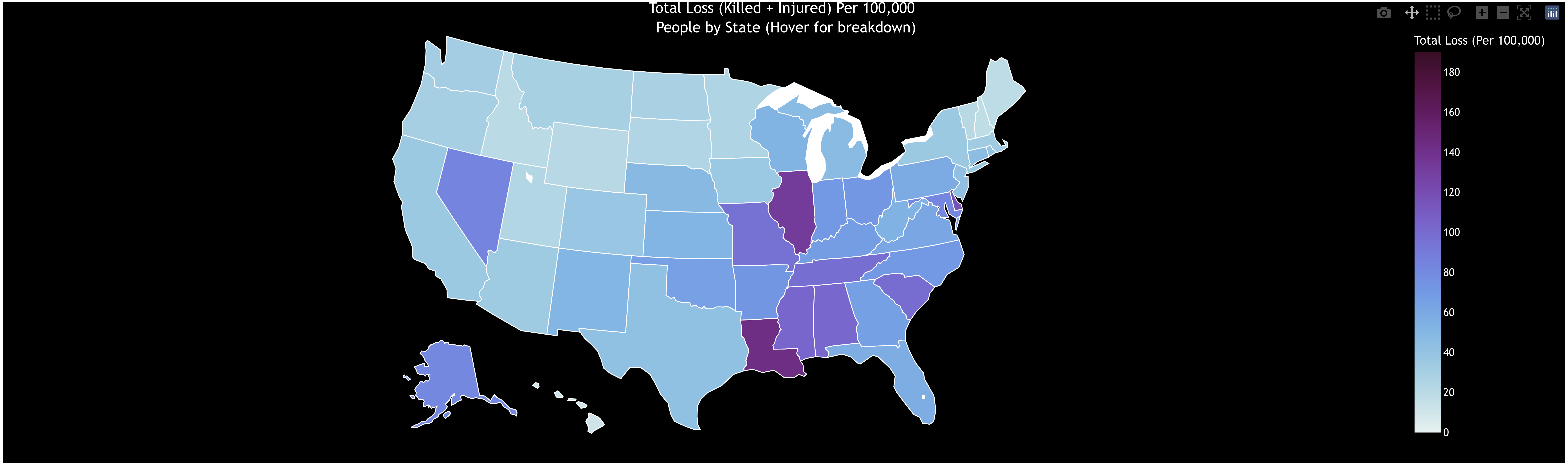
```
pop = {}
pop_filename = './gun-violence-data/state-population.csv'
PopData = pd.read_csv(pop_filename)
code_filename = './gun-violence-data/Codes.csv'
CodesData = pd.read_csv(code_filename)
PD = pd.DataFrame(columns = ['Code','Pop'])
x = 0
for i in range(len(PopData)):
    if PopData['state/region'][i] in CodesData['Code'].to_list():
        if PopData['ages'][i] == 'total':
            if PopData['year'][i] == 2013:
                pop[x] = {'Pop': PopData['population'][i],'Code': PopData['state/region'][i]}
                x = x+1
PD = pd.DataFrame.from_dict(pop,'index')



PA_States = picked_data_master[['state','n_killed','n_injured']].copy()
PA_States['Count'] = 1
PA_States = PA_States.groupby(['state']).agg('sum')
PA_States.reset_index(level = 0,inplace=True)


PA_States[['n_killed_adj','n_injured_adj','Count_adj']] = 0
PA_States.insert(1,'Pop',PD['Pop'])
PA_States.insert(2,'Code',PD['Code'])

for i in range(len(PA_States['Pop'])):
    PA_States['n_killed_adj'][i] = (PA_States['n_killed'][i]/PA_States['Pop'][i])*100000
    PA_States['n_injured_adj'][i] = (PA_States['n_injured'][i]/PA_States ['Pop'][i])*100000
    PA_States['Count_adj'][i] = (PA_States['Count'][i]/PA_States['Pop'][i])*100000
```

```
for col in PA_States.columns:
    PA_States[col] = PA_States[col].astype(str)

PA_States['text'] = 'State: ' + PA_States['state'] + '<br>' + 'Per 100,000 Stats:'+'<br>'\
    'Total Killed: ' + PA_States['n_killed_adj'] + '<br>'+'Total Injured: ' + PA_States['n_injured_adj'] + '<br>' + \
    'Total Incidents: ' + PA_States['Count_adj']

fig = go.Figure(data=go.Choropleth(
    locations=PA_States['Code'],
    z=PA_States['Count_adj'].astype(float),
    zmin = 0,
    zmax = 190,
    locationmode='USA-states',
    colorscale='reds',
    autocolorscale=False,
    text=PA_States['text'], # hover text
    marker_line_color='white', # line markers between states
    colorbar_title="Incidents (Per 100,000)",

))

fig.update_layout(
    title_text='Total Count of Incidents Per 100,000 <br> People by State (Hover for breakdown)',
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="White"
    ),

    geo = dict(
        scope='usa',
        projection=go.layout.geo.Projection(type = 'albers usa'),
        showlakes=True, # lakes
        lakecolor='rgb(255, 255, 255)',
        bgcolor ='black'),
    title_x=0.5,
    paper_bgcolor='black',
)
fig.update_layout(margin=dict(l=40, r=40, t=25, b=25))

fig.show()
```

```python
fig = go.Figure(data=go.Choropleth(
    locations=PA_States['Code'],
    z= PA_States['n_killed_adj'].astype('float') + PA_States['n_injured_adj'].astype('float'),
    zmin = 0,
    zmax = 190,
    locationmode='USA-states',
    colorscale='dense',
    autocolorscale=False,
    text=PA_States['text'], # hover text
    marker_line_color='white', # line markers between states
    colorbar_title="Total Loss (Per 100,000)",

))

fig.update_layout(
    title_text='Total Loss (Killed + Injured) Per 100,000 <br> People by State (Hover for breakdown)',
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="White"
    ),

    geo = dict(
        scope='usa',
        projection=go.layout.geo.Projection(type = 'albers usa'),
        showlakes=True, # lakes
        lakecolor='rgb(255, 255, 255)',
        bgcolor ='black'),
    title_x=0.5,
    paper_bgcolor='black',
)
fig.update_layout(margin=dict(l=40, r=40, t=25, b=25))


fig.show()
```







KEY TAKEAWAYS

1. The state with the most number of incidents per 100,000 people is Alaska, followed by Delaware, Louisiana, South Carolina and others. This is really different from our usual analysis and represents the actual scenario.
2. The state with maximum no. of loss (killed + injured) is Louisiana, followed by Illionis, Delaware, Mississippi and others. Note that Alaska is not in top of the list this time.
3. This analysis shows us that Alaska is most dangerous when it comes to incidents but Louisiana is most dangerous when it comes to actuall loss done by these incidents.

6.2 Incidents in different cities (Usual and Population Adjusted)

USUAL - We take the combined count of incidents over all the cities. [Jan 2014 - Mar 2018]

POPULATION ADJUSTED - We take the combined count of incidents over all the cities but multiply each count by the number obtained by dividing the population of each city by 100,000. [Jan 2014 - Mar 2018]

- ### 6.2.1 Usual : Using bar graph

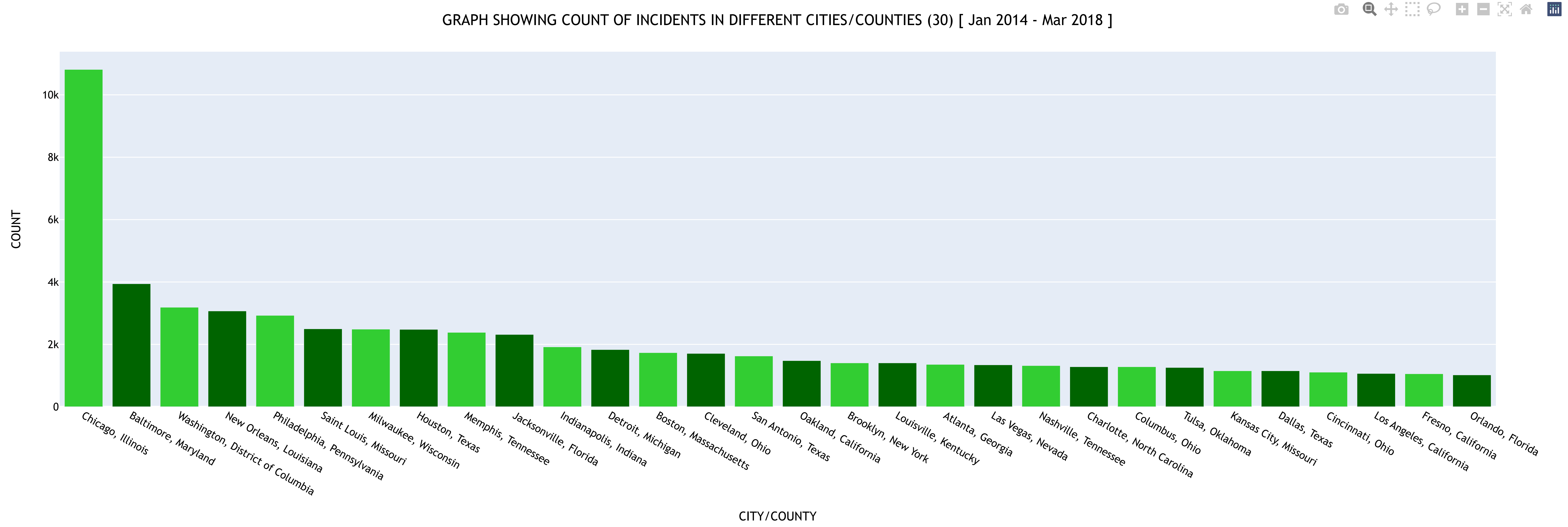
  We shall make use of *picked_data_master* dataframe and create a new dataframe called *cities_state*, add a *count* column to it and then take aggregate so as to join city and state names with a coma as one single column. This allows us to know which city or we are talking about (In the US there are 31 cities named Franklin, 29 named Clinton, 29 named Washington, 28 named Arlington, etc.). Using *cities* dictionary, we have 2 lists as dictionary values, one for name and one for count. After filling in the data in dictionary, we sort both the dictionary values together using *bubbleSort_modified* and *swapPositions* and then display the data using a bar graph.

In [39]:

```
cities = {'Name' : [], 'Count' : []}
cities_state = picked_data_master[['city_or_county','state','n_killed','n_injured']].copy()
cities_state['Count'] = 1
cities_state['city_or_county'] = cities_state[['city_or_county', 'state']].agg(', '.join, axis=1)


for value in tqdm(range(len(cities_state))):
    if (cities_state['city_or_county'][value] not in cities['Name']):
        cities['Name'].append(cities_state['city_or_county'][value])
        cities['Count'].append(0)

for value in tqdm(range(len(cities_state))):
    if (cities_state['city_or_county'][value] in cities['Name']):
        index = cities['Name'].index(cities_state['city_or_county'][value])
        cities['Count'][index] += 1;

def swapPositions(lst, pos1, pos2):
    lst[pos1], lst[pos2] = lst[pos2], lst[pos1]
    return lst

def bubbleSort_modified(arr1,arr2):
    n = len(arr1)
    for i in tqdm(range(n-1)):

        for j in range(0, n-i-1):

            if arr1[j] < arr1[j + 1] :
                swapPositions(arr1,j,j+1)
                swapPositions(arr2,j,j+1)

bubbleSort_modified(cities['Count'],cities['Name'])
fig = px.bar(cities,labels={'x':'City', 'y':'no. of incidents'},height = 600, x = cities['Name'][:30], y=cities['Count'][:30],color_discrete_sequence = ['limegreen','darkgreen'], color = cities['Name'][:30])
fig.update_layout(
    title="GRAPH SHOWING COUNT OF INCIDENTS IN DIFFERENT CITIES/COUNTIES (30) [ Jan 2014 - Mar 2018 ]",
    title_x=0.5,
```

```
    xaxis_title="CITY/COUNTY",
    yaxis_title="COUNT",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="Black"
    ),
      showlegend = False

)
fig.show()
```

```
100%|██████████| 239678/239678 [01:06<00:00, 3589.65it/s]
100%|██████████| 239678/239678 [01:45<00:00, 2282.32it/s]
100%|██████████| 17540/17540 [02:21<00:00, 124.00it/s]
```



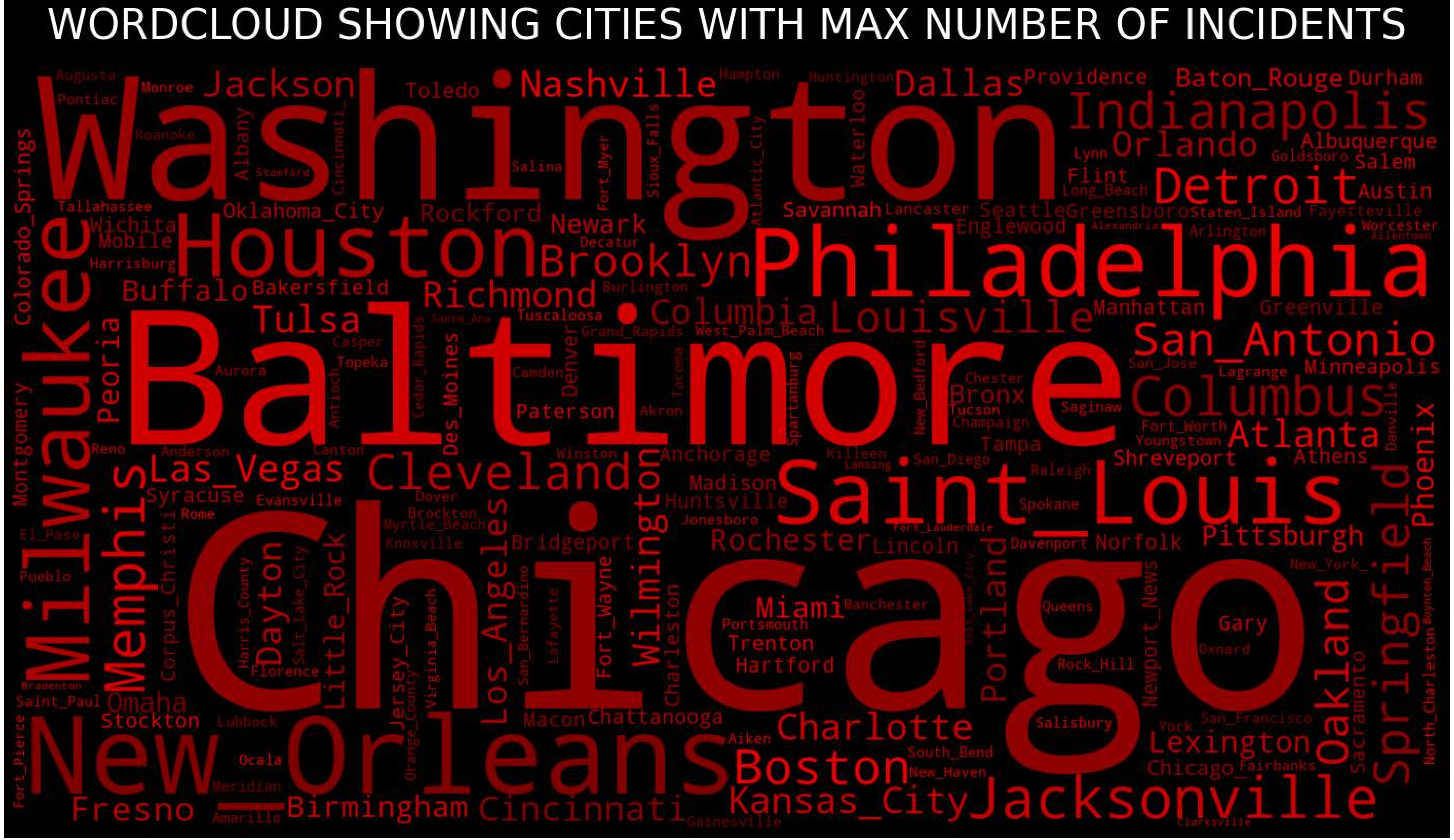
KEY TAKEAWAYS

1. The city with most number of incidents is Chicago, Illinois. Followed by Baltimore, Maryland and Washington, District of Columbia.
2. Chicago has 2.7x the no. of incidents compared to the next city, showing that Chicago is way more dangerous than other cities.
3. A population adjusted dataset must be used for a more precise analysis.

- ### 6.2.2 Usual : Using wordcloud

  The WordCloud() function is used here exactly like the one used for states and the desired figure is obtained.

In [40]:
```
word_list = cities_state['city_or_county']

word_list = [x.split(',')[0] for x in word_list]



city_list = []
for word in word_list:
    word = word.replace(" ",'_')
    city_list.append(word.replace("(county)", "County"))
city_list = " ".join(city_list)

def random_color_func(word=None, font_size=None, position=None,  orientation=None, font_path=None, random_state=None):
    h = int(360.0 * 0.0 / 255.0)
    s = int(100.0 * 255.0 / 255.0)
    l = int(100.0 * float(random_state.randint(60, 120)) / 255.0)

    return "hsl({}, {}%, {}%)".format(h, s, l)
fig = WordCloud(max_words = 200, collocations = False,color_func=random_color_func, background_color = 'black',width=1500, height=800).generate(city_list)
plt.figure(figsize=[25,25],facecolor='k')
plt.axis("off")
plt.imshow(fig, interpolation='bilinear')
plt.title("WORDCLOUD SHOWING CITIES WITH MAX NUMBER OF INCIDENTS",y=1.03,fontdict = {'color':'white','fontsize':40})
plt.show()
```



KEY TAKEAWAYS

1. Unlike 30 cities above, this wordcloud shows and gives us an idea about the range of data based on individual sizes. Chicago stll stands at no. 1 but even though Chicago has 2.7x the number of incidents when compared to Baltimore but still the wordcloud does not potray the same. Thus, wordcloud is not a good idea when things are supposed to be drawn to scale.
2. A population adjusted analysis is required.

- ### 6.2.3 Population Adjusted : Using bar graph

  We use a grouped bar graph to represent the data. First of all, we obtain city population data from here. We create dataframes *PopData* and *Citate*, along with dictionary *PA_cities_dict*. In *PopData*, we combine the city and statenames together in one column (from the data obtained from document 'uscities.csv'). *Citate* is made from the contribution of *picked_data_master* and *PopData* and an added column 'count' for no. of incidents. After this, we run a for loop over the length of *PopData* and fill out the following details in the dictionary using *Citate* and *PopData*:

- values for 'city' key as city names

- values for 'state' key as state names
- values for 'population' key as count of population
- values for 'Count' key as count of incidents
- values for 'Killed' key as count of people killed
- values for 'Injured' key as count of people injured

  We create anoter dataframe called *PA_cities_df* and add columns:

  - 'Killed_adj' for no. of people killed per 100,000 people
  - 'Injured_adj' for no. of people injured per 100,00 people
  - 'Count_adj' for no. of incidents per 100,000 people
  - 'Loss_adj' for total no. of people killed and injured per 100,000 people.

  We create a copy of *PA_cities_df* as *PA_cities_df_loss*. From here, we go on to create two bar graphs using *go.Figure()*, the first one is made using *PA_cities_df* with the deciding factor being 'Count_adj' and the second figure is made using *PA_cities_df_loss* with the deciding factor being 'Loss_adj'. Population is displayed as hovertext for both the figures along with the usual details and we have selected only those cities that have population 0.5 million or above, because there are some cities with vey low population but as they have some 1 or 2 incidents, their adjusted figures are very high. They are more of outliers than reasonable piece of data, lying in an abnormal distance from other values. Check the code below for implementation of each of the figures.

In [41]:

```
pop_filename = './gun-violence-data/uscities.csv'
PopData = pd.read_csv(pop_filename)
PopData.sort_values(by = 'city',inplace = True)
PopData.reset_index(drop = 'index',inplace = True)

PopData['city'] = PopData[['city', 'state_name']].agg(', '.join, axis=1)







Citate = pd.DataFrame()
Citate = picked_data_master[['city_or_county','state','n_killed','n_injured']].copy()

Citate['Count'] = 1

Citate['city_or_county'] = Citate[['city_or_county', 'state']].agg(', '.join, axis=1)

Citate = Citate.groupby(['city_or_county']).agg('sum')
Citate.reset_index(inplace = True)

Citate['state'] = '_'




PA_cities_dict = {}
for i in tqdm(range(len(Citate))):
    Citate['state'][i] = Citate['city_or_county'][i].split(', ')[1]
```

```python
x = 0
for i in tqdm(range(len(PopData))):
    if PopData['city'][i] in Citate['city_or_county'].to_list():
        index = Citate['city_or_county'].to_list().index(PopData['city'][i])
        lst = Citate['city_or_county'][index].split(', ')
        str1 = lst[0]
        str2 = lst[1]
        if (Citate['state'][index] == PopData['state_name'][i]) and (Citate['n_killed'][index]>0 or Citate['n_injured'][index]>0):

            PA_cities_dict[x] = {'city':Citate['city_or_county'][index], 'state':str2,\
                                  'population':PopData['population'][i],'Count':Citate['Count'][index],'Killed':Citate['n_killed'][index],\
                                 'Injured':Citate['n_injured'][index]}
        else:
            PA_cities_dict[x] = {'city':Citate['city_or_county'][index], 'state':str2,\
                                  'population':PopData['population'][i],'Count':0,'Killed':0,\
                                 'Injured':0}
        x = x+1
    else:
        PA_cities_dict[x] = {'city':PopData['city'][i], 'state':PopData['state_name'][i],\
                              'population':PopData['population'][i],'Count':0,'Killed':0,\
                              'Injured':0}
        x = x+1

PA_cities_df = pd.DataFrame.from_dict(PA_cities_dict,'index')


PA_cities_df[['Killed_adj','Injured_adj','Count_adj', 'Loss_adj']] = 0


for i in tqdm(range(len(PA_cities_df['population']))):
    PA_cities_df['Loss_adj'][i] = ((PA_cities_df['Killed'][i] + PA_cities_df['Injured'][i])/PA_cities_df['population'][i])*100000
    PA_cities_df['Count_adj'][i] = (PA_cities_df['Count'][i]/PA_cities_df['population'][i])*100000


PA_cities_df_loss =  PA_cities_df


##########################################################################################################FIGURE 1

PA_cities_df.sort_values(by = 'Count_adj',inplace = True,ascending=False)
```

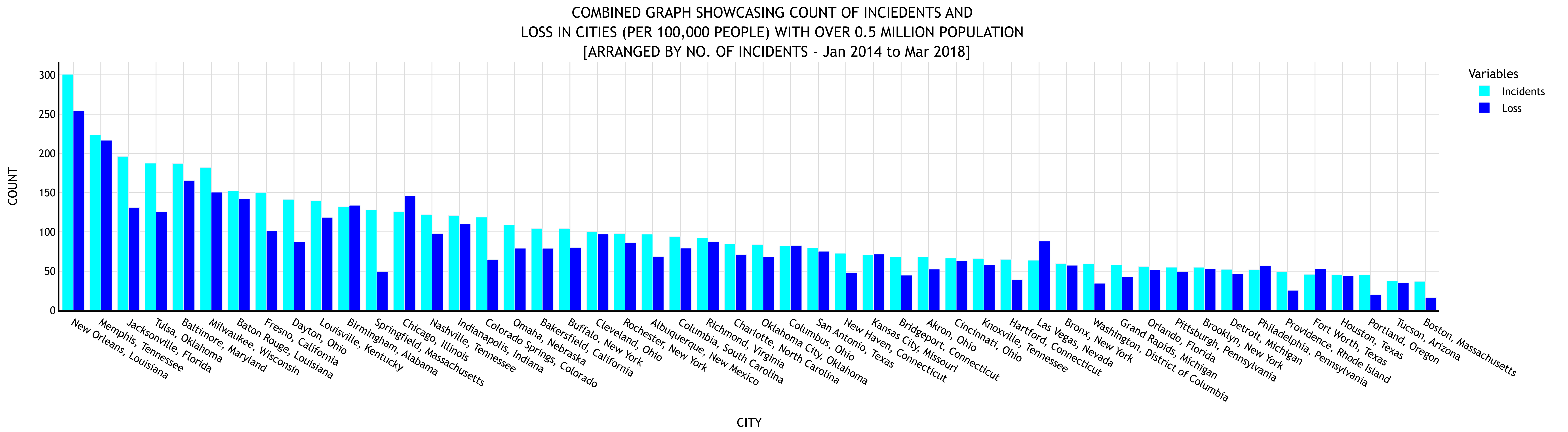
```
PA_cities_df.reset_index(drop = True,inplace = True)



City_adj = {'City':[], 'Count1':[], 'Count2': [], 'Population':[]}


Reasonable_cities = PA_cities_df.drop((PA_cities_df[PA_cities_df['population']<500000]).index, axis = 0)
Reasonable_cities.reset_index(drop = True,inplace = True)


for x in Reasonable_cities['city'][:50]:
    City_adj['City'].append(x)


for x in Reasonable_cities['Count_adj'][:50]:
    City_adj['Count1'].append(x)

for i in range(len(Reasonable_cities['Killed_adj'][:50])):
    City_adj['Count2'].append(Reasonable_cities['Loss_adj'][i])

for i in range(len(Reasonable_cities['population'][:50])):
    City_adj['Population'].append("Population : " + str(Reasonable_cities['population'][i]))

trace1 = go.Bar(
   x = City_adj['City'],
   y = City_adj['Count1'],
   name = 'Incidents',
   marker_color='aqua',

   hovertext = City_adj['Population'],
   opacity=1
)
trace2 = go.Bar(
   x = City_adj['City'],
   y = City_adj['Count2'],
   name = 'Loss',
   marker_color='blue',

   hovertext = City_adj['Population'],
   opacity=1
)
```

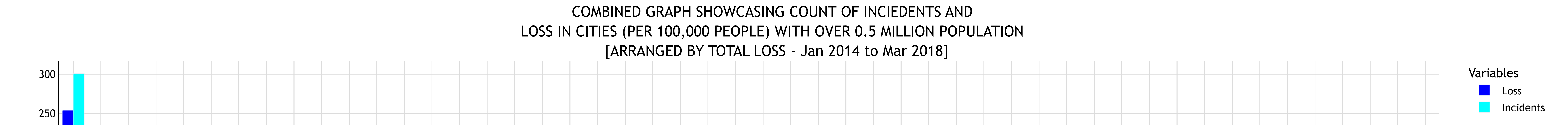
```
data = [trace1, trace2]
layout = go.Layout(barmode = 'group')
fig = go.Figure(data = data, layout = layout)
fig.update_layout(
    title="COMBINED GRAPH SHOWCASING COUNT OF INCIEDENTS AND "+'<br>'+ "LOSS IN CITIES (PER 100,000 PEOPLE) WITH OVER 0.5 MILLION POPULATION <br> [ARRANGED BY NO. OF INCIDENTS - Jan 2014 to Mar 2018]",
    title_x=0.5,
    xaxis_title="CITY",
    yaxis_title="COUNT",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="black"
    ),

    legend_title_text = "Variables",
    plot_bgcolor='white',
    paper_bgcolor='white',
    )
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', gridcolor ='Gainsboro')
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', gridcolor ='Gainsboro')
iplot(fig)

#########################################################################################################FIGURE 2

PA_cities_df_loss.sort_values(by = 'Loss_adj' ,inplace = True,ascending=False)


PA_cities_df_loss.reset_index(drop = True,inplace = True)




City_adj = {'City':[], 'Count1':[], 'Count2': [], 'Population':[]}



Reasonable_cities = PA_cities_df_loss.drop((PA_cities_df_loss[PA_cities_df_loss['population']<500000]).index, axis = 0)
Reasonable_cities.reset_index(drop = True,inplace = True)

for x in Reasonable_cities['city'][:50]:
    City_adj['City'].append(x)
```

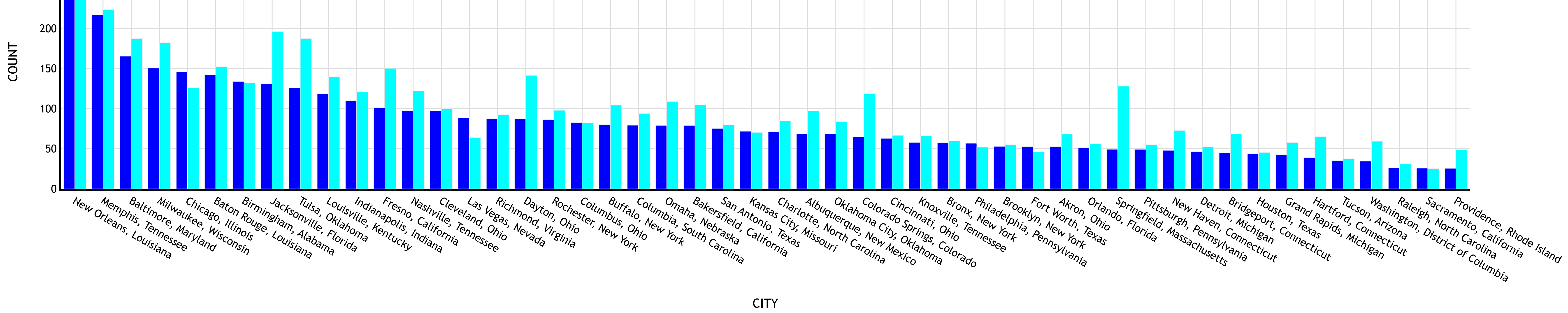
```
for x in Reasonable_cities['Count_adj'][:50]:
    City_adj['Count1'].append(x)

for i in range(len(Reasonable_cities['Killed_adj'][:50])):
    City_adj['Count2'].append(Reasonable_cities['Loss_adj'][i])

for i in range(len(Reasonable_cities['population'][:50])):
    City_adj['Population'].append("Population : " + str(Reasonable_cities['population'][i]))



trace1 = go.Bar(
   x = City_adj['City'],
   y = City_adj['Count1'],
   name = 'Incidents',
   marker_color='aqua',

   hovertext = City_adj['Population'],
   opacity=1
)
trace2 = go.Bar(
   x = City_adj['City'],
   y = City_adj['Count2'],
   name = 'Loss',
   marker_color='blue',

   hovertext = City_adj['Population'],
   opacity=1
)



data = [trace2, trace1]
layout = go.Layout(barmode = 'group')
fig = go.Figure(data = data, layout = layout)
fig.update_layout(
    title="COMBINED GRAPH SHOWCASING COUNT OF INCIEDENTS AND "+'<br>'+ "LOSS IN CITIES (PER 100,000 PEOPLE) WITH OVER 0.5 MILLION POPULATION <br> [ARRANGED BY TOTAL LOSS - Jan 2014 to Mar 2018]",
    title_x=0.5,
    xaxis_title="CITY",
    yaxis_title="COUNT",
    font=dict(
```

```
        family="Trebuchet MS",
        size=12,
        color="black"
    ),

    legend_title_text = "Variables",
    plot_bgcolor='white',
    paper_bgcolor='white',
    )
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', gridcolor ='Gainsboro')
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', gridcolor ='Gainsboro')
iplot(fig)
```

```
100%|██████████| 17541/17541 [00:15<00:00, 1107.08it/s]
100%|██████████| 28338/28338 [03:34<00:00, 131.97it/s]
100%|██████████| 28338/28338 [00:04<00:00, 6344.65it/s]
```







KEY TAKEAWAYS

1. For figure 1, New Orleans, Memphis and Jacksonville stand at top 3 positions for most number of incidents per 100,000 people. This marks these are the most dangerous cities as per count of incidents. Notice how Chicago is nowhere in top 3 (it is in 12). This shows how true analysis is actually carried out when we do a population adjusted analysis.
2. For figure 2, New Orleans, Memphis and Baltimore grap the first 3 spots, now note that Jacksonville is not in top 3 as was in the first figure, It is beacuse, though the number of incidents per 100,000 people is quite high in Jacksonville but the number of people who are getting killed or injured pwr 100,000 people is comparitively not that high. Chigogo has climbed to the 5th place now, showing that total loss per 100,000 people is pretty high though no. of incidents per 100,000 people is not.

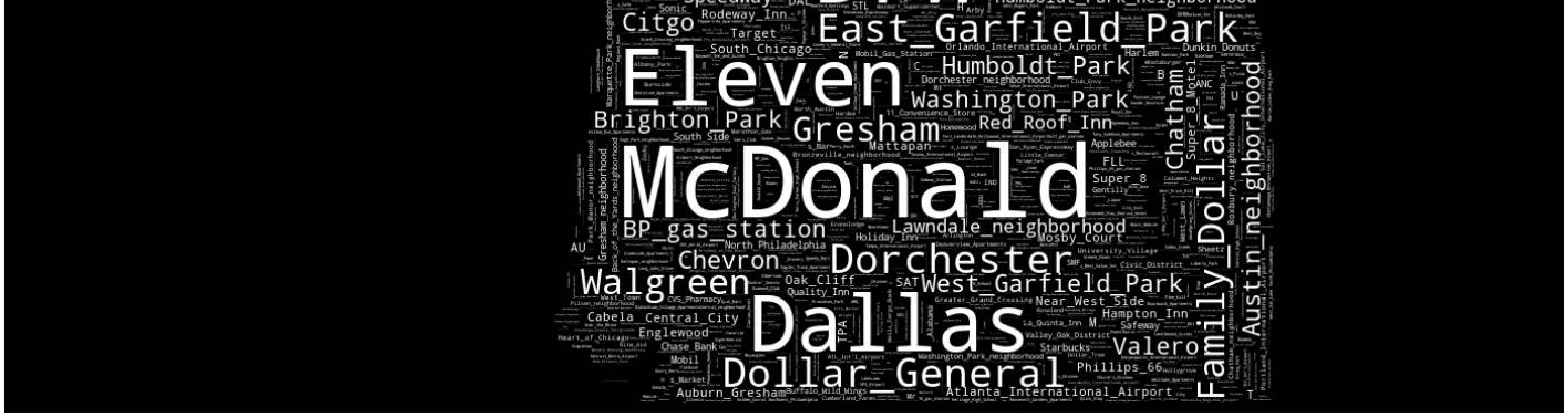
6.3 Most common locations of incidents : Using wordcloud

Our original dataframe *df* has a column as '*location_description*' which contains the location of incident, we pick up each of those locations and replace ' ' by '_' for each string. After that we use *Image* from PIL to create mask, *random_color_func* for color_func argument and pass all of this into the *WordCloud()* function. The result is shown below.

In [42]:

```
#Locations



location = df['location_description'].dropna().reset_index(drop = True)
location = location.to_list()
loc = []
for word in location:
    word = word.replace(" ","_")
    loc.append(word)
loc =  " ".join(loc)
mask = np.array(Image.open("./gun-violence-data/james.JPG"))

def random_color_func(word=None, font_size=None, position=None,  orientation=None, font_path=None, random_state=None):
    h = int(360.0 * 0.0 / 255.0)
    s = int(100.0 * 0.0 / 255.0)
    l = int(100.0 * float(random_state.randint(60, 120)) / 255.0)
```

```
wordcloud = WordCloud(width = 1200, height = 800,min_font_size =1,max_font_size = 100,max_words = len(loc),color_func = random_color_func, background_color='black', collocations=False,mask =
mask).generate(loc)
# create coloring from image



plt.figure(figsize=(25,25),facecolor = 'k')
plt.title("WORDCLOUD SHOWING MOST COMMON LOCATIONS OF INCIDENTS",y=1.05,fontdict = {'color':'cornsilk','fontsize':40})
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```



KEY TAKEAWAYS

1. Walmart is the most dangerous public location when it comes to gun violence.
2. McDonalds along with Fort Worth International Airport, East and West Garfield Park are some other noteworthy mentions.
3. Motel 6, being the largest owned and operated hotel chain in North America is, no doubt, is also in the list.

6.DATA ANALYSIS - GENDER

Our main aim here is to know no. of people killed, injured and unharmed in gun violence based on gender. From *picked_data* dataframe, we extract *'participant_gender'* as *pgender* and *'participant_status'* as *pstatus*. Then, we run a for loop over *pgender*, get gender and their status using *re.split()*, and get no. of males injured, killed, unharmed, no. of females injured, killed and unharmed, as shown below. We plot 3 figures as part of figure 1, one for each characteristic : Killed, Injured, Unharmed, divided by gender.

For second figure, we again plot 3 figures, one for each characteristic : Killed, Injured, Unharmed, divided by gender but this time we each of them are adjusted numbers, in the sense that each characteristic, be it Killed, Injured or Unharmed is per 10 females/males (whatever be the gender). See below for implementation.

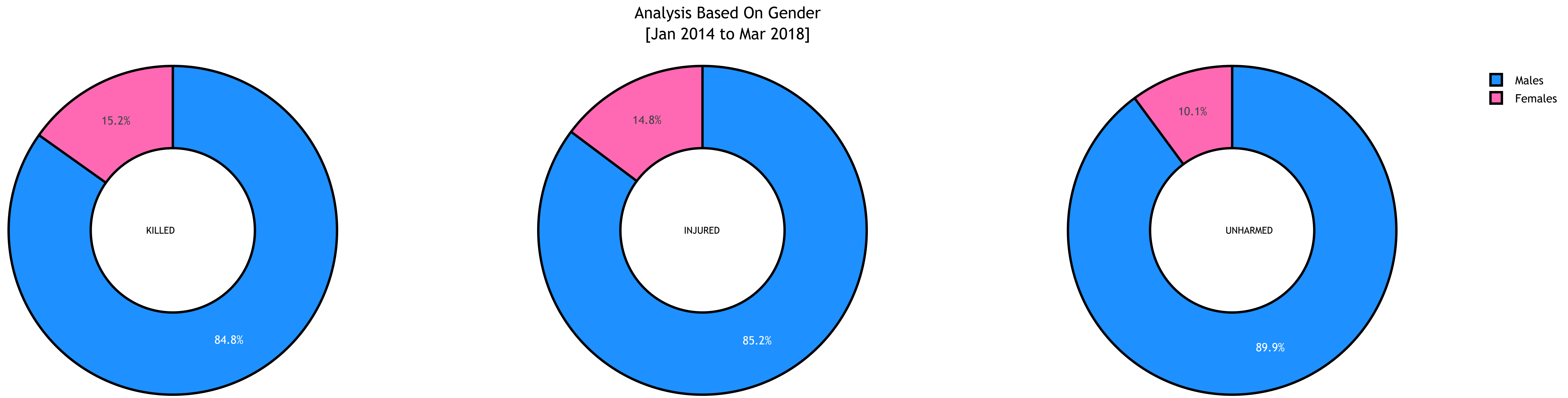
In [43]:

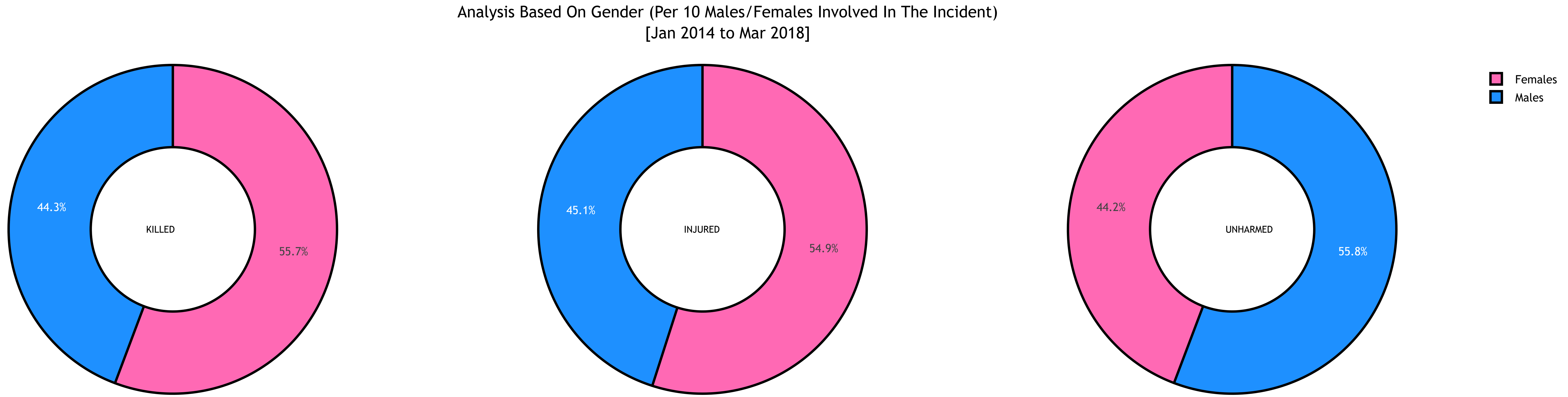
```
Males_Injured = 0
Males_Killed = 0
Females_Injured = 0
Females_Killed = 0
pgender = picked_data['participant_gender']
pstatus = picked_data['participant_status']
for i in range(len(pgender)):
    ginfo = re.split('::| ',pgender[i])
    sinfo = re.split('::| ',pstatus[i])
    length_ginfo = len(ginfo)
    length_sinfo = len(sinfo)
    if length_ginfo<=length_sinfo:
        for y in range(1,length_ginfo):
            ginfo[y] = ginfo[y].strip('|1234567890,')
            sinfo[y] = sinfo[y].strip('|1234567890,')
```

```
            if(ginfo[y] == 'Male'):
                if(sinfo[y] == 'Killed'):
                    Males_Killed += 1
                elif(sinfo[y] == 'Injured'):
                     Males_Injured +=1
            elif(ginfo[y] == 'Female'):
                if(sinfo[y] == 'Killed'):
                    Females_Killed += 1
                elif(sinfo[y] == 'Injured'):
                     Females_Injured +=1

Females_Involved = numeric_data['TFemales'].sum()
Males_Involved = numeric_data['TMales'].sum()
Males_Unharmed = Males_Involved - (Males_Injured + Males_Killed)
Females_Unharmed = Females_Involved - (Females_Injured + Females_Killed)


labels = ["Males", "Females"]

colors = ['DodgerBlue','hotpink']

fig = make_subplots(rows=1, cols=3, specs=[[{'type':'domain'}, {'type':'domain'},{'type':'domain'}]])
fig.add_trace(go.Pie(labels=labels, values=[Males_Killed,Females_Killed], name="Killed"),
              1, 1)
fig.add_trace(go.Pie(labels=labels, values=[Males_Injured,Females_Injured], name="Injured"),
              1, 2)
fig.add_trace(go.Pie(labels=labels, values=[Males_Unharmed,Females_Unharmed], name="Unharmed"),
              1, 3)
# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.5, hoverinfo="label+percent+name+value",marker=dict(colors=colors, line=dict(color='black', width=2.5)))

fig.update_layout(
    title_text="Analysis Based On Gender<br>[Jan 2014 to Mar 2018]",
    title_x=0.5,
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="black"
    ),
    paper_bgcolor='white',
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='KILLED', x=0.125, y=0.5, font_size=10, showarrow=False),
                 dict(text='INJURED', x=0.50, y=0.5, font_size=10, showarrow=False),
```

```
                    dict(text='UNHARMED', x=0.885, y=0.5, font_size=10, showarrow=False)])



fig.show()








Males_Killed_adj = (Males_Killed/Males_Involved)*10
Females_Killed_adj = (Females_Killed/Females_Involved)*10
Males_Unharmed_adj = (Males_Unharmed/Males_Involved)*10
Females_Unharmed_adj = (Females_Unharmed/Females_Involved)*10
Males_Injured_adj = (Males_Injured/Males_Involved)*10
Females_Injured_adj = (Females_Injured/Females_Involved)*10

# Create subplots: use 'domain' type for Pie subplot
fig = make_subplots(rows=1, cols=3, specs=[[{'type':'domain'}, {'type':'domain'},{'type':'domain'}]])
fig.add_trace(go.Pie(labels=labels, values=[Males_Killed_adj,Females_Killed_adj ], name="Killed"),
              1, 1)
fig.add_trace(go.Pie(labels=labels, values=[Males_Injured_adj,Females_Injured_adj], name="Injured"),
              1, 2)
fig.add_trace(go.Pie(labels=labels, values=[Males_Unharmed_adj,Females_Unharmed_adj], name="Unharmed"),
              1, 3)
# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.5, hoverinfo="label+percent+name+value",marker=dict(colors=colors, line=dict(color='black', width=2.5)))

fig.update_layout(
    title_text="Analysis Based On Gender (Per 10 Males/Females Involved In The Incident)<br>[Jan 2014 to Mar 2018]",
    title_x=0.5,
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="black"
    ),paper_bgcolor='white',
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='KILLED', x=0.125, y=0.5, font_size=10, showarrow=False),
                 dict(text='INJURED', x=0.50, y=0.5, font_size=10, showarrow=False),
                 dict(text='UNHARMED', x=0.885, y=0.5, font_size=10, showarrow=False)])
```

```
fig.show()
```


Analysis Based On Gender
[Jan 2014 to Mar 2018]
15.2%
84.8%
KILLED
14.8%
85.2%
INJURED
10.1%
89.9%
UNHARMED
Males
Females


Analysis Based On Gender (Per 10 Males/Females Involved In The Incident)
[Jan 2014 to Mar 2018]
44.3%
55.7%
KILLED
45.1%
54.9%
INJURED
44.2%
55.8%
UNHARMED
Females
Males

KEY TAKEAWAYS

1. In Figure 1, we note that 84.8% of people killed are males, 85.2% of people injured are males and 89.9% of people unharmed are males. This paints an image that in any scenario males are always more in number. This is because males ARE more in number. What we must do in do an adjusted analysis and this is exactly what we have done in Figure 2.
2. Now the scenario changes completely, note that when we carry out **'Killed per 10 males/females'**, females turn out to be more in percentage. This means that for every 10 people who are killed in gun violence incidents, 2.1 of them (55.7%) are females and 1.6 of them (44.3%) are males.
3. For every 10 people who are injured in gun violence incidents, 3.6 of them (54.9%) are females and 2.9 of them (45.1%) are males.
4. For every 10 people who remain unharmed in gun violence incidents, 5.3 of them (55.8%) are males and 4.2 of them (44.2%) are females. This shows that females are more likely to be injured or worst, killed, than males in gun violence. This shows us the real picture.

7.DATA ANALYSIS - GUNS

- ## 7.1 Number of guns involved

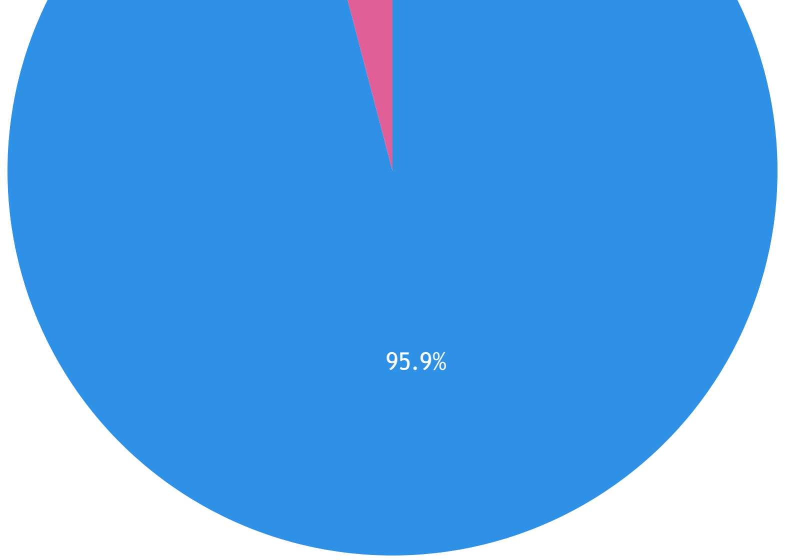
  In this, we shall figure out the number of guns involved in incidents.

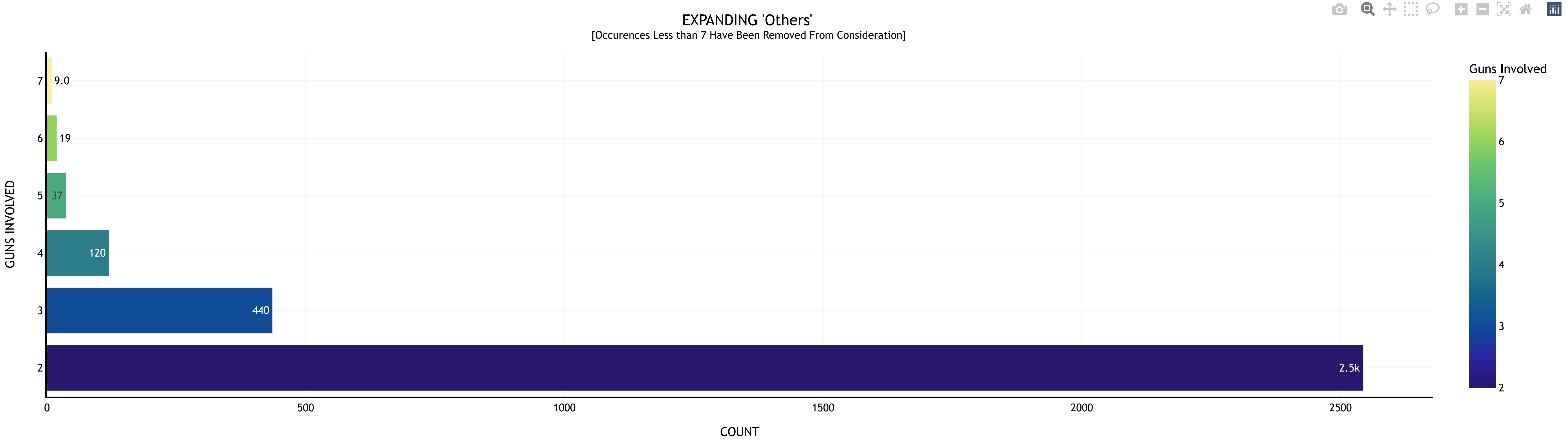
  First of all, we create a dataframe called *guns1* which is sorted by '*n_guns_involved*'. Then we create another dataframe called '*guns2*' which is basically same as *guns1* but all the rows having '*n_killed*' and '*n_injured*' values as zero has been dropped. Now we create a dictionary from *guns2* by using *Counter()* from *collections* library. Counter() takes in input a list, tuple, dictionary, string, which are all iterable objects, and gives output that will have the count of each element. We create a dictionary '*guns*' and use 8Guns_data.keys() *and* Gunds_data.values() *as values for this dictionary, which is later converted to a dataframe called* Guns_df. *Then, we create another* temp_df *dataframe which has 1st row of* Guns_df. *We then delete all rows from* Guns_df *having rows in which* 'Guns_involved' *is 1 (there is only one row like that).* temp_df *is then reassigned to itself after appending one more row to it in the form of a dictionary having 2nd key value pair as* `'Count':Guns_df['Count'].sum()`. *This* .sum() *adds up all the values in* 'Count' *column of* Guns_df *dataframe. So ultimately we have a dataframe called* temp_df *having only 2 rows. The first row has info about number of incidents that had only 1 gun. And the second row has number of incidents that had guns greater than 1. The pie chart has been plotted using* px.pie()*.

  The bar graph below is an attempt to get an idea of how many incidents were there with 2, 3 and more guns. The rows of 'Guns_df' having less than 7 guns has been deleted as they are outliers (some incidents had 47, 80+ guns, etc. but it happend only once from 2013 (atleast those that are recorded here), 2014 to Mar 2018.

In [44]:
```python
guns1 = df.sort_values(by = ['n_guns_involved'])[['n_guns_involved','n_killed','n_injured']].dropna().reset_index(drop = True)


guns2 = guns1.drop(guns1[(guns1['n_killed']==0) & (guns1['n_injured']==0)].index).reset_index(drop = True)



Guns_data = Counter(guns2['n_guns_involved'])


Guns_No = Guns_data.keys()
Guns_Values = Guns_data.values()
Guns = {'Guns Involved':Guns_No,'Count':Guns_Values}
Guns_df = pd.DataFrame(Guns)
temp_df = Guns_df.head(1)
Guns_df = Guns_df.drop(Guns_df[(Guns_df['Guns Involved']==1.0)].index).reset_index(drop = True)
temp_df = temp_df.append({'Guns Involved':'Others','Count':Guns_df['Count'].sum()},ignore_index = True)
temp_df['Guns Involved'] = temp_df['Guns Involved'].astype(str)
fig = px.pie(temp_df, values='Count', names='Guns Involved',color_discrete_sequence = px.colors.qualitative.Dark24)
```

```
fig.update_layout(
    title_text="NUMBER OF GUNS INVOLVED [Jan 2014 - Mar 2018]",
    title_x=0.485,
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="black"
    ),paper_bgcolor='white',



)



fig.show()



####################################################################################################BAR GRAPH
Guns_df = Guns_df.drop(Guns_df[(Guns_df['Count']<=6)].index).reset_index(drop = True)
fig = px.bar(Guns_df,orientation = 'h',text_auto='.2s',x = Guns_df['Count'], y=Guns_df['Guns Involved'],color_continuous_scale = 'Haline' ,color = Guns_df['Guns Involved'])
fig.update_layout(
    title_text="EXPANDING 'Others'<br><sup> [Occurences Less than 7 Have Been Removed From Consideration]",
    title_x=0.485,
    xaxis_title="COUNT",
    yaxis_title="GUNS INVOLVED",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="black"
    ),paper_bgcolor='white',
    plot_bgcolor = 'white'


)
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', gridcolor ='WhiteSmoke')
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', gridcolor ='WhiteSmoke')
fig.show()
```





KEY TAKEAWAYS

1. From figure 1, we conclude that, 1 gun is involved in 95.9% of the incidents with other numbers being quite rare (4.08%).
2. From figure 2, we conclude that, in more than 81% incidents out of 3208 incidents, the number of guns involved is 2. And other numbers (like 3,4,...) are even lesser as shown.

- ## 7.2 Types of guns involved

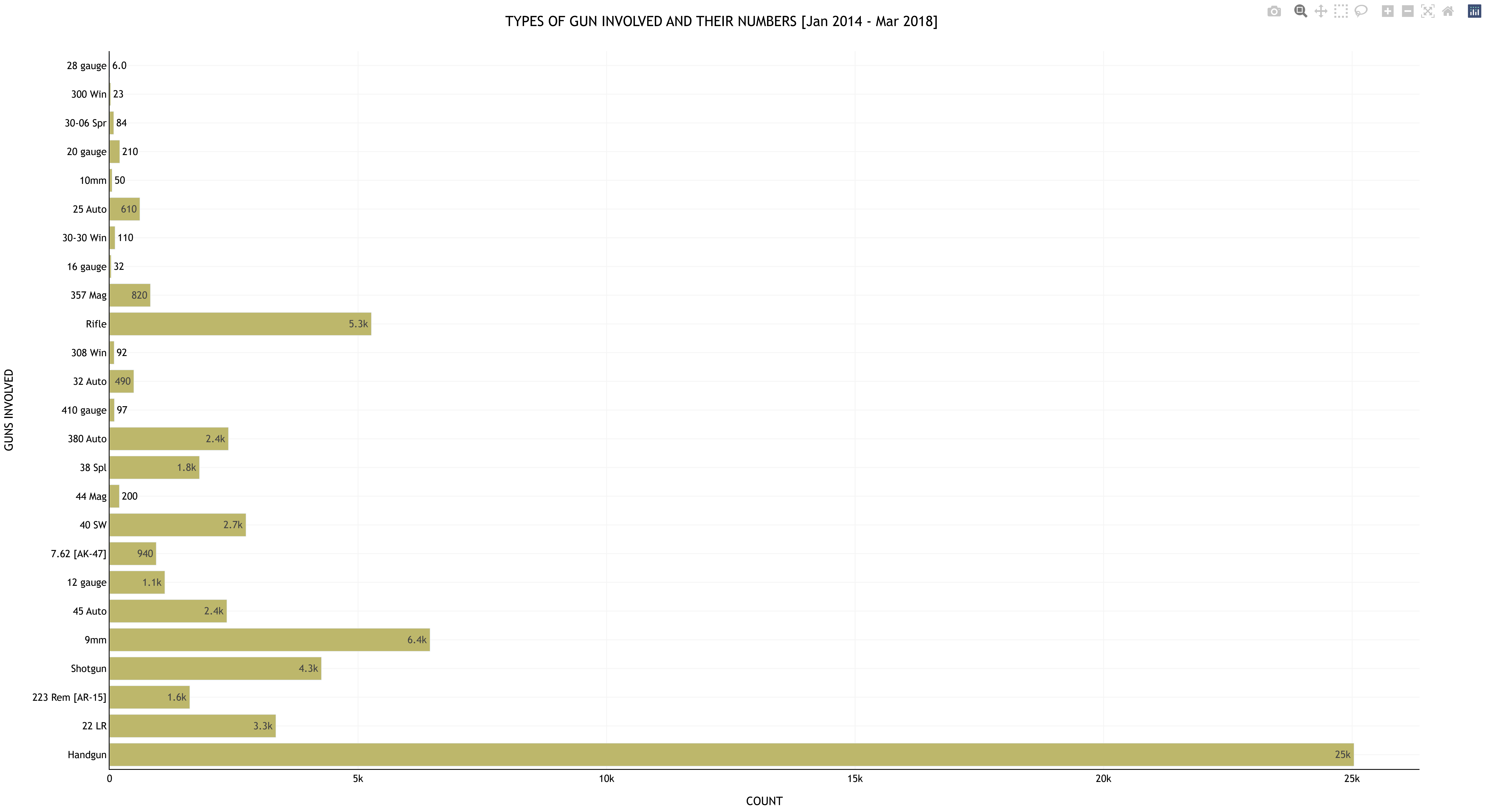
  Moving on, we try to find the types of guns that are involved in these incidents.

For this, we create a dataframe called *guntype* using *df['gun_type']* and drop all rows that have *'gun_type'* as '0::Unknown' or '-'. We carefully extract gun types using a for loop over *guntype* and utilising *re.split()* at every iterative step as shown below, obtain a dictionary *Guns*, fill it up with keys and values (values using *Counter()*) and pass it to *px.bar()* to plot the number of guns of each type involved in gun violence incidents.

Check the code below for implementation.

In [45]:

```
guntype = df['gun_type'].dropna().reset_index(drop = True)
guntype = pd.DataFrame(guntype)

guntype= guntype.drop(guntype[(guntype['gun_type'] == '0::Unknown') | (guntype['gun_type']=='-')].index).reset_index(drop = True)



gtype = []
for i in range(len(guntype)):
    lst = re.split('::',guntype['gun_type'][i])
    for z in range(1,len(lst)):
        if 'Unknown' not in lst[z]:
            if 'Other' not in lst[z]:
                ele = lst[z].split('||')[0]
                gtype.append(ele)



Guns = {'Gun Type' : Counter(gtype).keys(), 'Count' : Counter(gtype).values()}

fig = px.bar(Guns,height = 1000,orientation = 'h',text_auto='.2s',labels={'y':'Gun Involved ', 'x':'Count '},x = Guns['Count'], y=Guns['Gun Type'],color_discrete_sequence = ['darkkhaki'])
fig.update_layout(
    title_text="TYPES OF GUN INVOLVED AND THEIR NUMBERS [Jan 2014 - Mar 2018]",
    title_x=0.485,
    xaxis_title="COUNT",
    yaxis_title="GUNS INVOLVED",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="black"
    ),paper_bgcolor='white',
     showlegend = False,
    plot_bgcolor = 'white',






)
fig.update_xaxes(showline=True, linewidth=1, linecolor='black', gridcolor ='WhiteSmoke')
fig.update_yaxes(showline=True, linewidth=1, linecolor='black', gridcolor ='WhiteSmoke')
fig.show()
```



KEY TAKEAWAYS

1. Handgun, 9mm and Rifle take the first 3 spots. They are the most common guns used in these incidents.
2. The handgun is the most common, more than 25,000+ guns involved in these incidents were handguns.

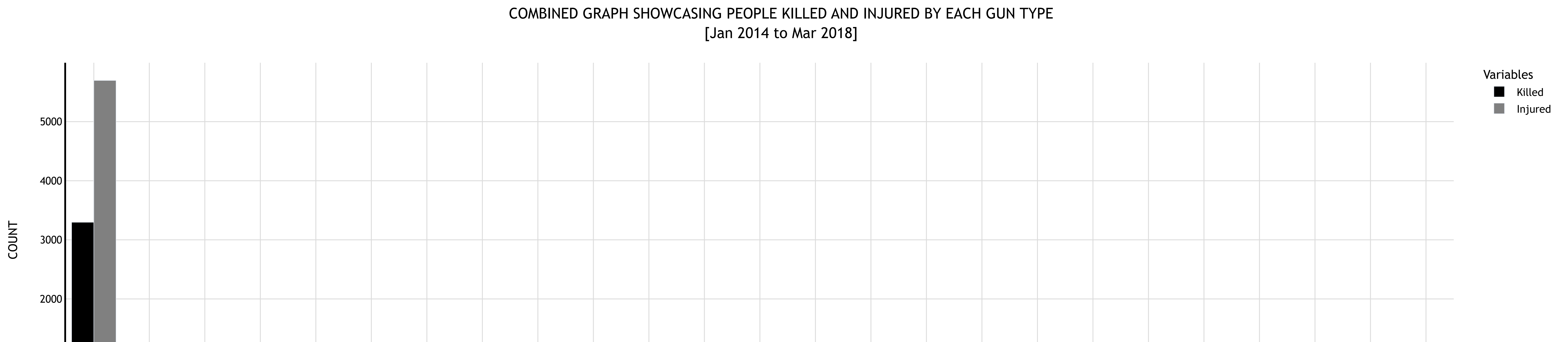
- ## 7.3 Injured and Killed (based on type of the gun involved)

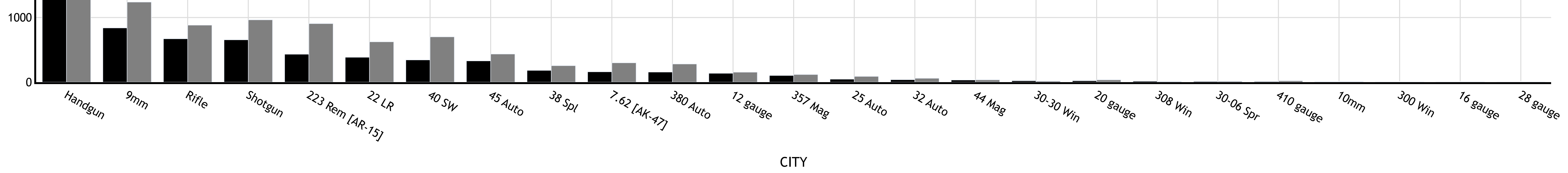
  Actually, based on the dataset, we cannot pinpoint and say which injury or killing is caused by which gun as in some incidents there are multiple guns involved and we have no data on which gun killed/injured how many people. But, we can have an estimate by finding correlation between gun involved and killings/injuries associated with that incident. This can simply be achieved by analysing which incident had which gun and the killings/injuries of that incident.

In [46]:

```
guntype = df[['gun_type','n_killed','n_injured']].dropna().reset_index(drop = True)

guntype = pd.DataFrame(guntype)

guntype= guntype.drop(guntype[(guntype['gun_type'] == '0::Unknown') | (guntype['gun_type']=='-')].index).reset_index(drop = True)



gtype = []
killed = []
injured = []

for i in range(len(guntype)):
    lst = re.split('::',guntype['gun_type'][i])
    for z in range(1,len(lst)):
        if 'Unknown' not in lst[z]:
            if 'Other' not in lst[z]:
                ele = lst[z].split('||')[0]
                gtype.append(ele)
                killed.append(guntype['n_killed'][i])
                injured.append(guntype['n_injured'][i])



gun_KI = pd.DataFrame({'Guntype':gtype ,'Killed':killed ,'Injured':injured })
gun_KI = gun_KI.groupby(['Guntype']).agg('sum').reset_index().reset_index(drop = True).sort_values(by = 'Killed',ascending = False)



trace1 = go.Bar(
   x = gun_KI['Guntype'],
   y = gun_KI['Killed'],
   name = 'Killed',
   marker_color='black',

)
trace2 = go.Bar(
   x = gun_KI['Guntype'],
   y = gun_KI['Injured'],
```

```
    name = 'Injured',
    marker_color='gray',

)

data = [trace1, trace2]
layout = go.Layout(height = 600,barmode = 'group')
fig = go.Figure(data = data, layout = layout)
fig.update_layout(
    title="COMBINED GRAPH SHOWCASING PEOPLE KILLED AND INJURED BY EACH GUN TYPE"+'<br>'+ "[Jan 2014 to Mar 2018]",
    title_x=0.5,
    xaxis_title="CITY",
    yaxis_title="COUNT",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="black"
    ),

    legend_title_text = "Variables",
    plot_bgcolor='white',
    paper_bgcolor='white',
    )
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', gridcolor ='Gainsboro')
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', gridcolor ='Gainsboro')
iplot(fig)
```





8. DATA ANALYIS : AGE

We plot the number of people belonging to each age group (0-10, 11-20,etc) and also age distribution of people involved in gun violence.

We create two lists, one is a list with age groups (*L1*) and other is a list (*L2*) with integer variables (which have already been initialised to 0). A series *p_age* containing non empty rows of *df['participant_age']* is looped over and if-elif is used to increment each variable associated with a specific age group as shown below. *age_dict* dictionary is made using list *L1* and *L2* as values of the keys in dictionary. This *age_dict* dictionary is then passed on to *px.bar()* to create the following bar graph.

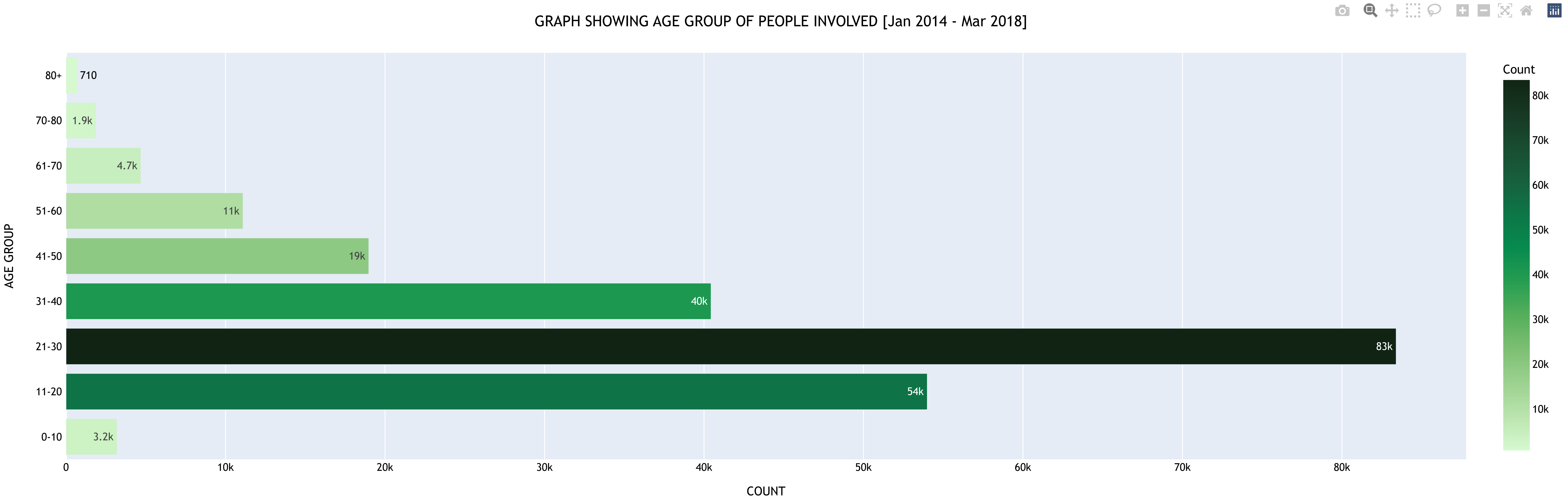
Following that we create a bar graph showing us the number of people of each specific age. We simple create two lists *age_specific* and *count_age* and use for loops to fill in values for them. We then create a dictionary with these lists as its values and pass this dictionary as an argument to *px.bar()* for obtaining the plot.

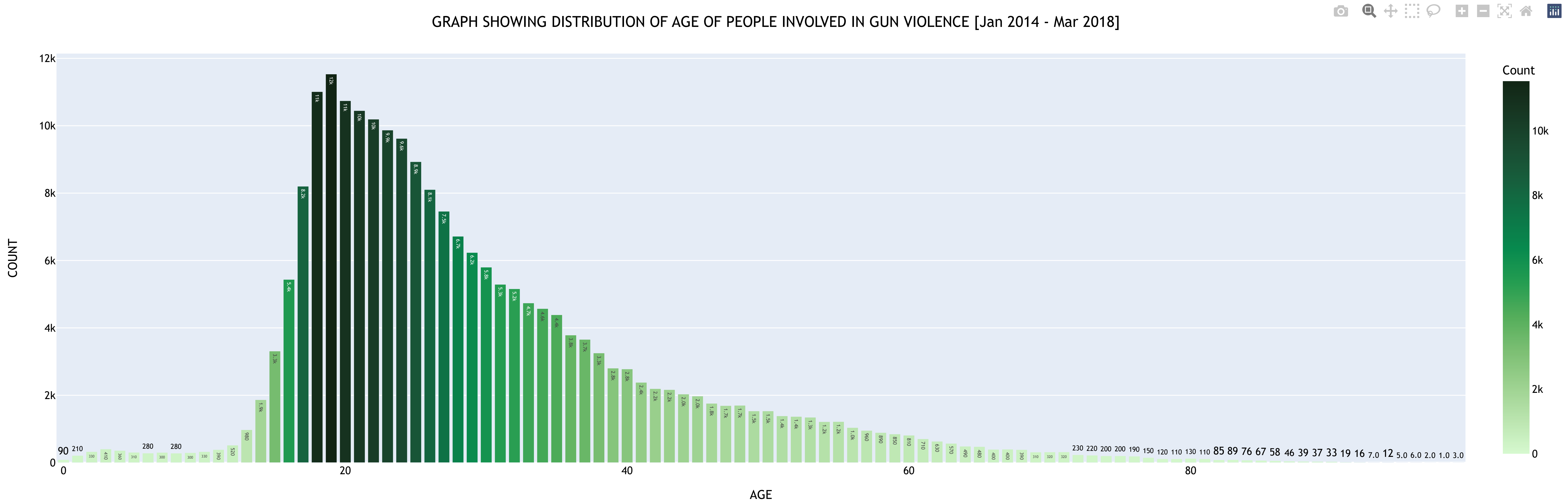
In [47]:

```
a10 = a20 = a30 = a40 = a50 = a60 = a70 = a80 = a90 = 0
L1 = ["0-10 ", "11-20 ", "21-30 ", "31-40 ", "41-50 ","51-60 ","61-70 ","70-80 ","80+ "]
L2 = [a10,a20,a30,a40,a50,a60,a70,a80,a90]

p_age = df['participant_age'].dropna().reset_index(drop = True)




for i in range(len(p_age)-1):        #Last row in. p_age series has '-' as its value.
    age_info = re.split("::| ",p_age[i])[1:]
    for y in age_info:

        age = int(y[:2].strip('|'))
        if(age>=0 and age<=10):
            L2[0] += 1
        elif(age>=11 and age<=20):
            L2[1] += 1
        elif(age>=21 and age<=30):
            L2[2] += 1
        elif(age>=31 and age<=40):
            L2[3] += 1
        elif(age>=41 and age<=50):
            L2[4] += 1
        elif(age>=51 and age<=60):
            L2[5] += 1
```

```
        elif(age>=61 and age<=70):
            L2[6] += 1
        elif(age>=70 and age<=80):
            L2[7] += 1
        elif(age>=80):
            L2[8] += 1

age_dict = {'Age Group ':L1,'Count ':L2}

fig = px.bar(age_dict,text_auto='.2s',orientation='h', height = 600, x = 'Count ', y = 'Age Group ',color='Count ',color_continuous_scale = 'algae')
fig.update_layout(
    title="GRAPH SHOWING AGE GROUP OF PEOPLE INVOLVED [Jan 2014 - Mar 2018]",
    title_x=0.5,
    xaxis_title="COUNT",
    yaxis_title="AGE GROUP",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="black"
    ),
    coloraxis_showscale=True
)
fig.show()

#####################################################################################################################################
age_specific = []
count_age = []

for i in range(len(p_age)-1):
    age_info = re.split("::| ",p_age[i])[1:]
    for y in age_info:

        age = int(y[:2].strip('|'))
        if age not in age_specific:
            age_specific.append(age)
            count_age.append(0)

for i in range(len(p_age)-1):
    age_info = re.split("::| ",p_age[i])[1:]
    for y in age_info:

        age = int(y[:2].strip('|'))
        index = age_specific.index(age)
```

```
        count_age[index] += 1

age_dict = {'Age' : age_specific, 'Count': count_age}

fig = px.bar(age_dict,text_auto='.2s',height = 600, orientation='v', x = 'Age', y = 'Count',color='Count',color_continuous_scale = 'algae')

fig.update_layout(
    title="GRAPH SHOWING DISTRIBUTION OF AGE OF PEOPLE INVOLVED IN GUN VIOLENCE [Jan 2014 - Mar 2018]",
    title_x=0.5,
    xaxis_title="AGE",
    yaxis_title="COUNT",
    font=dict(
        family="Trebuchet MS",
        size=12,
        color="black"
    ),
    coloraxis_showscale=True
)
fig.show()
```





KEY TAKEAWAYS

1. 21-30 age-group is the dominant age-group when it comes to involvement in gun violence.
2. The age-group that follows the top one is 11-20 age-group, next to this is 41-50 age-group.
3. Sadly, there are about 3.2k people involved in these incidents who are less than 11 years old.
4. Also, 710 80+ people are involved.
5. Coming to the second figure, we get an even clearer understanding. 19 year olds are the one with maximum involvement in these incidents.
6. 18, 20 and 21 closely follow the lead.
7. 90 childeren, who not even completed a single year of their existence on this planet are also involved in these violent incidents.

9. DATA VISUALISATION : LOCATION OF TOP 5000 INCIDENTS (Based on no. of people killed and injured)

For this analysis, we shall plot the location top 5000 incidents on a map using *folium*.

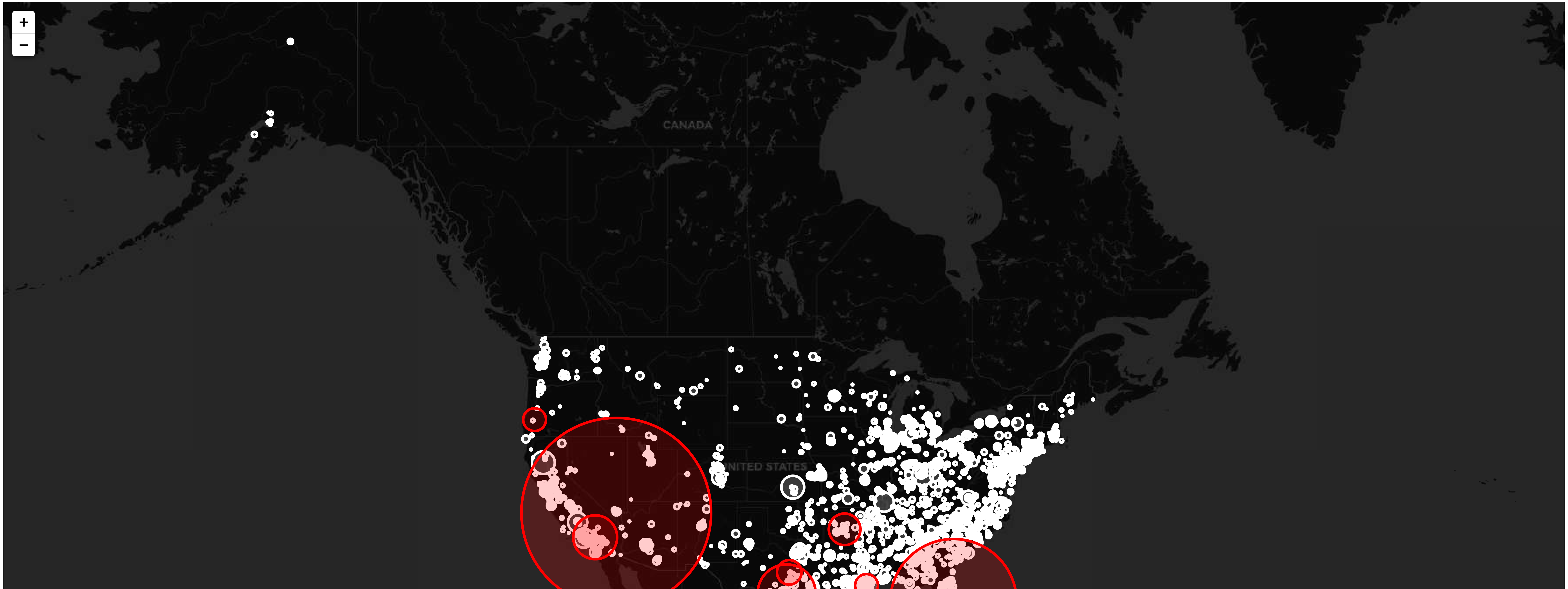
We start by creating a new column in *df* which is known as *Total Loss*. It contains the sum of '*n_killed*' and '*n_injured*' of every row. *folium_map_data* is created using *df* by using `df.sort_values(by = 'Total Loss',ascending = False)` to have values sorted by '*Total Loss*' in *folium_map_data*.Now we use *folium.Map()* and create our map named *map1*. We create title of the map using *.get_root().html.add_child()*, popup using *folium.Popup()* add add red circle markers to the map for top 10 incidents and white circle markers for 11 to top 5000 incidents using *folium.CircleMarker()*. After adding all these details we display the map using `display(map1)`.

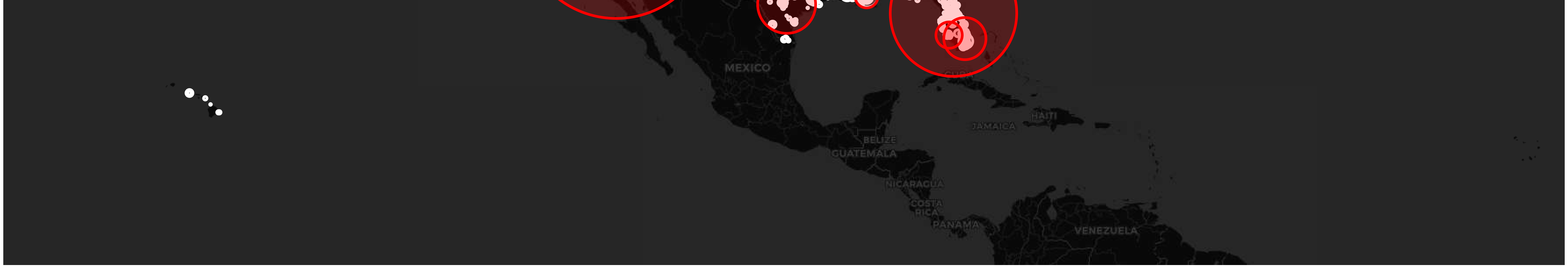
In [48]:
```
df['Total Loss'] = df['n_killed'] + df['n_injured']
```

```python
folium_map_data = df.sort_values(by = 'Total Loss',ascending = False)

folium_map_data = folium_map_data[['date','latitude','longitude','Total Loss','n_killed','n_injured','n_guns_involved','state','address','city_or_county']]

folium_map_data = folium_map_data.drop(folium_map_data[folium_map_data.isnull().any(axis = 1)].index, axis = 0)


map1 = folium.Map(location=[39.50, -98.35], tiles='CartoDB dark_matter', zoom_start=3.5)
markers = []

tooltip = 'Click For Details'
title_html = '''
             <h3 align="center" style="font-size:20px"><b>DETAILS ABOUT TOP 5000 INCIDENTS<br>(TOP 10 IN RED)<br> [Jan 2014 - Mar 2018]</b></h3>
             '''
map1.get_root().html.add_child(folium.Element(title_html))

for i, row in folium_map_data[10:5000].iterrows():
    html = "Date: " + str(row['date']) + "<br>" +"Total Loss: " + str(row['Total Loss']) + \
       "<br>" + "Total Killed: " + str(row['n_killed']) + "<br>" +\
      "Total Injured: " + str(row['n_injured']) + "<br>" +\
      "Total Guns Involved: " + str(int(row['n_guns_involved'])) + "<br>" +\
      "Address: " + str(row['address']) + "<br>" +\
      "City: " + str(row['city_or_county']) + "<br>" +\
      "State: " + str(row['state']) + "<br>"
    iframe = folium.IFrame(html,width=300,height=170)


    popup = folium.Popup(iframe, max_width='100%')
    loss = row['Total Loss']*0.7
    folium.CircleMarker([float(row['latitude']),float(row['longitude'])], popup = popup,tooltip = tooltip,radius=float(loss), color='WHITE', fill=True).add_to(map1)

for i, row in folium_map_data[:10].iterrows():
    html = "Date: " + str(row['date']) + "<br>" +"Total Loss: " + str(row['Total Loss']) + \
       "<br>" + "Total Killed: " + str(row['n_killed']) + "<br>" +\
      "Total Injured: " + str(row['n_injured']) + "<br>" +\
      "Total Guns Involved: " + str(int(row['n_guns_involved'])) + "<br>" +\
      "Address: " + str(row['address']) + "<br>" +\
      "City: " + str(row['city_or_county']) + "<br>" +\
      "State: " + str(row['state']) + "<br>"

    iframe = folium.IFrame(html,width=300,height=170)
```

```
    popup = folium.Popup(iframe, max_width='100%')

    loss = row['Total Loss']*0.7
    if loss > 270:
        loss = 270*0.4
    folium.CircleMarker([float(row['latitude']),float(row['longitude'])],tooltip = tooltip, popup = popup, radius=float(loss), color='RED', fill=True).add_to(map1)




display(map1)
```

DETAILS ABOUT TOP 5000 INCIDENTS
(TOP 10 IN RED)
[Jan 2014 - Mar 2018]





CONCLUSION:

With the analysis of the gun violence dataset, it can be concluded that gun violence is progressively increasing yearly and will continue to do so unless strict laws in the US prevent these incidents from occurring. It turns out that no. of people involved in incidents is more or less increasing every year as well. July and August seem to be the months with the most incidents every year, with July 4th and 5th specifically being days with alarming no. of incidents. Furthermore, The weekends happen to be the unsafest days of the week. Coming to individual states, Illinois and California top the chart for most incidents. However, the moment population-adjusted analysis is carried out, one cannot fail to notice that Alaska (for most incidents per 100,000 people) and Louisiana (for the most killings and injuries per 100,000 people) stand at the top. For cities, Chicago has the maximum incidents, but New Orleans, Louisiana tops in population-adjusted analysis. According to the analysis, Walmart and McDonald's have proven to be quite vicious because the number of incidents is very high in these public spaces. Gender-wise, males have more involvement in these, but females are more likely to be killed or injured in gun violence, as evidenced by the dataset. 20-30 year age group people are the most involved in shooting incidents. Analyzing each age individually, it can be said that 18 and 19-year-olds have the most extensive involvement. Coming to guns, Handguns, 9mm ones, and Rifles are the most popular choices for the subjects, and most of the incidents have been witnessed with only one gun. Overall, these are the most important facts from this exploratory data analysis.

In [ ]:
```

```