In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import plotly.express as px
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#Reading the csv file:
df = pd.read_csv(r'C:\Users\Public\gun-violence-data.csv')
```

In [3]:
```
#Data set columns and records count:
print("Number of Columns :",len(df.columns))
print("Number of Records :",len(df))
```

```
Number of Columns : 29
Number of Records : 239677
```

In [4]:
```
#Displaying first five records:
df.head()
```

Out[4]:

	incident_id	date	state	city_or_county	address	n_killed	n_injured	
0	461105	2013-01-01	Pennsylvania	Mckeesport	1506 Versailles Avenue and Coursin Street	0	4	http://www.g
1	460726	2013-01-01	California	Hawthorne	13500 block of Cerise Avenue	1	3	http://www.gu
2	478855	2013-01-01	Ohio	Lorain	1776 East 28th Street	1	3	http://www.gu
3	478925	2013-01-05	Colorado	Aurora	16000 block of East Ithaca Place	4	0	http://www.gu
4	478959	2013-01-07	North Carolina	Greensboro	307 Mourning Dove Terrace	2	2	http://www.gu

5 rows × 29 columns

In [5]:
```
# Data preprocessing
df = df.dropna()
df = df.drop_duplicates(keep=False)
```

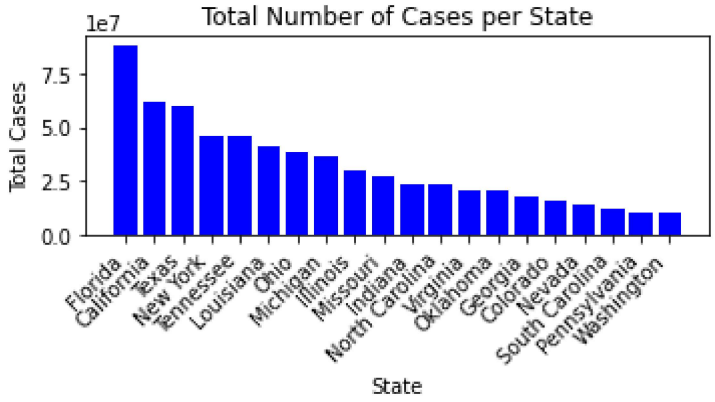
In [6]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1059 entries, 36 to 239511
Data columns (total 29 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   incident_id                  1059 non-null   int64  
 1   date                         1059 non-null   object 
 2   state                        1059 non-null   object 
 3   city_or_county               1059 non-null   object 
 4   address                      1059 non-null   object 
 5   n_killed                     1059 non-null   int64  
 6   n_injured                    1059 non-null   int64  
 7   incident_url                 1059 non-null   object 
 8   source_url                   1059 non-null   object 
 9   incident_url_fields_missing  1059 non-null   bool   
 10  congressional_district       1059 non-null   float64
 11  gun_stolen                   1059 non-null   object 
 12  gun_type                     1059 non-null   object 
 13  incident_characteristics     1059 non-null   object 
 14  latitude                     1059 non-null   float64
 15  location_description         1059 non-null   object 
 16  longitude                    1059 non-null   float64
 17  n_guns_involved              1059 non-null   float64
 18  notes                        1059 non-null   object 
 19  participant_age              1059 non-null   object 
 20  participant_age_group        1059 non-null   object 
 21  participant_gender           1059 non-null   object 
 22  participant_name             1059 non-null   object 
 23  participant_relationship     1059 non-null   object 
 24  participant_status           1059 non-null   object 
 25  participant_type             1059 non-null   object 
 26  sources                      1059 non-null   object 
 27  state_house_district         1059 non-null   float64
 28  state_senate_district        1059 non-null   float64
dtypes: bool(1), float64(6), int64(3), object(19)
memory usage: 241.0+ KB
```

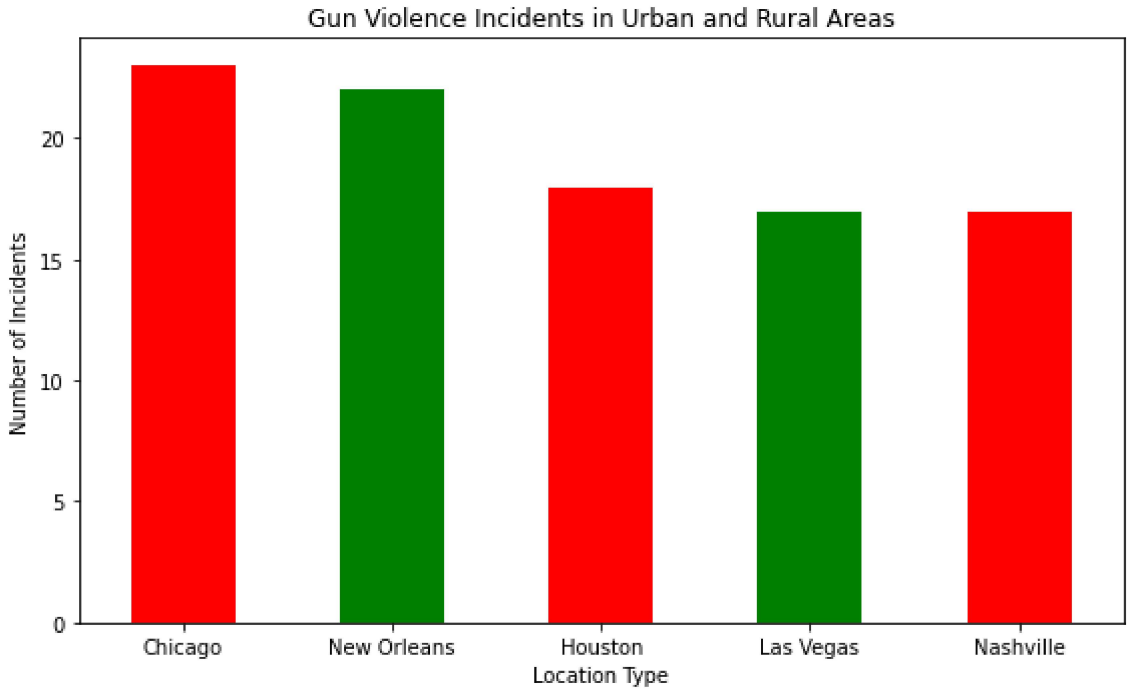
In [37]:

```
# How many cases have been registered in each state?
state_cases = df.groupby('state')['incident_id'].sum().sort_values(asce

# Plotting the bar chart
plt.figure(figsize=(5, 3))
plt.bar(state_cases['state'], state_cases['incident_id'], color='blue')
plt.xlabel('State')
plt.ylabel('Total Cases')
plt.title('Total Number of Cases per State')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better
plt.tight_layout()

# Show the plot
plt.show()
```



In [8]:

```
# Can you compare and contrast gun violence incidents between urban (ci
urban_vs_rural = df['city_or_county'].value_counts().head(5)

# Plotting the bar chart
plt.figure(figsize=(8, 5))
urban_vs_rural.plot(kind='bar', color=['red', 'green'])
plt.xlabel('Location Type')
plt.ylabel('Number of Incidents')
plt.title('Gun Violence Incidents in Urban and Rural Areas')
plt.xticks(rotation=0)
plt.tight_layout()

# Show the plot
plt.show()
```



In [9]:

```python
#Determine if there's any correlation between the presence of specific
participant_statuses  = {}

def participant_type_count(age_type):
    if str(age_type) != 'nan':
        ages = age_type.split("||")
        if len(ages) == 1:
            ages = ages[0].split("|")
        for age in ages:
            try:
                if age.split('::')[1] in participant_statuses.keys():
                    participant_statuses[age.split('::')[1]] += 1
                else:
                    participant_statuses[age.split('::')[1]] = 0
            except:
                if age.split(':')[1] in participant_statuses.keys():
                    participant_statuses[age.split(':')[1]] += 1
                else:
                    participant_statuses[age.split(':')[1]] = 0


df['participant_status'].apply(participant_type_count);

participant_statuses = {k: v for k, v in sorted(participant_statuses.it
age_df = pd.DataFrame(participant_statuses.items(), columns = ['partici
fig = px.bar(age_df, x='participant status', y='Count')
#fig.update_layout(xaxis_tickangle=45)
fig.show()
```

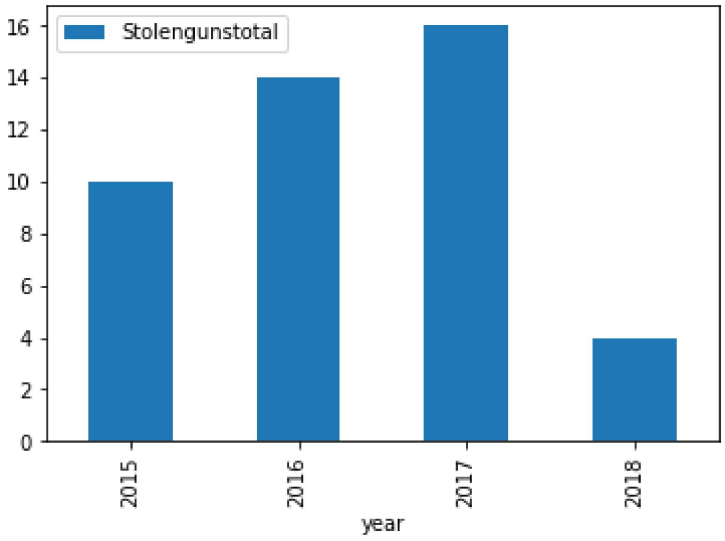
In [12]:

```
#Determine whether particular demographics are particularly susceptible
from collections import Counter
import re
from plotly.offline import download_plotlyjs, init_notebook_mode, plot,


total_incidents = []
for i, each_inc in enumerate(df['incident_characteristics'].fillna('Not
    split_vals = [x for x in re.split('\|', each_inc) if len(x)>0]
    total_incidents.append(split_vals)
    if i == 0:
        unique_incidents = Counter(split_vals)
    else:
        for x in split_vals:
            unique_incidents[x] +=1

unique_incidents = pd.DataFrame.from_dict(unique_incidents, orient='inc
colvals = unique_incidents[0].sort_values(ascending=False).index.values
find_val = lambda searchList, elem: [[i for i, x in enumerate(searchLis

a = np.zeros((df.shape[0], len(colvals)))
for i, incident in enumerate(total_incidents):
    aval = find_val(colvals, incident)
    a[i, np.array(aval)] = 1
incident = pd.DataFrame(a, index=df.index, columns=colvals)

prominent_incidents = incident.sum()[[4, 5, 6, 9, 10, 11, 13, 14, 15, 1
                                      21, 23,22,24,45,51]]
fig = {
    'data': [
        {
            'labels': prominent_incidents.index,
            'values': prominent_incidents,
            'type': 'pie',
            'hoverinfo':'label+percent+name',
            "domain": {"x": [0, .45]},
        }
    ],
    'layout': {'title': 'Prominent Incidents of Gun Violence',
               'showlegend': False, 'height':800, 'width':800}
}

iplot(fig)
```

In [13]:

```
# Determine whether cases using stolen guns differ from those involving
df['date']=pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year
df['month']=df['date'].dt.month
df['gun_stolen'] = df['gun_stolen'].fillna('Null')

df['gun_stolen'] = df['gun_stolen'].str.replace('::',',')
df['gun_stolen'] = df['gun_stolen'].str.replace('|',' ')
df['gun_stolen'] = df['gun_stolen'].str.replace(',',' ')
df['gun_stolen']= df['gun_stolen'].str.replace('\d+', '')


df['Stolenguns']=df['gun_stolen'].apply(lambda x: x.count('Stolen'))
df['stolenguns']=df['gun_stolen'].apply(lambda x: x.count('Stolen'))
df['Stolengunstotal'] = df['Stolenguns'] + df['stolenguns']

df_year_stolenguns = df[['year','Stolengunstotal']].groupby(['year'], a

df_year_stolenguns = df_year_stolenguns[df_year_stolenguns['Stolengunst
df_year_stolenguns[['year','Stolengunstotal']].set_index('year').plot(k
```

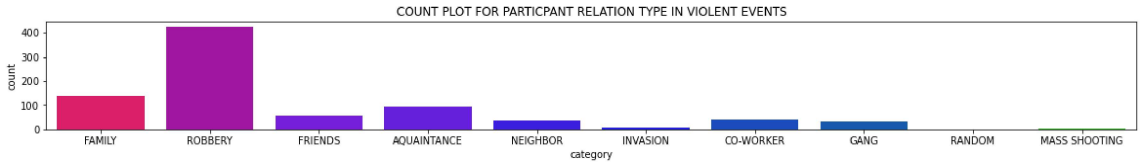
Out[13]: <AxesSubplot:xlabel='year'>



In [44]:

```
# To identify particpant relation type in violent events
relation = df['participant_relationship']
relation = relation[relation.notnull()]
relation = relation.str.replace("[:|0-9]"," ").str.upper()
relation1 = pd.DataFrame({"count":[len(relation[relation.str.contains('
                len(relation[relation.str.contains("ROBBERY")]),
                len(relation[relation.str.contains("FRIENDS")]),
                len(relation[relation.str.contains("AQUAINTANCE")]),
                len(relation[relation.str.contains("NEIGHBOR")]),
                len(relation[relation.str.contains("INVASION")]),
                len(relation[relation.str.contains("CO-WORKER")]),
                len(relation[relation.str.contains("GANG")]),
                len(relation[relation.str.contains("RANDOM")]),
                len(relation[relation.str.contains("MASS SHOOTING")])],
               "category":["FAMILY","ROBBERY","FRIENDS","AQUAINTANCE","N
relation1
plt.figure(figsize=(20,2))
sns.barplot("category","count",data=relation1,palette="prism")
plt.title("COUNT PLOT FOR PARTICPANT RELATION TYPE IN VIOLENT EVENTS")
```

Out[44]: Text(0.5, 1.0, 'COUNT PLOT FOR PARTICPANT RELATION TYPE IN VIOLENT EVENTS')



In [15]:

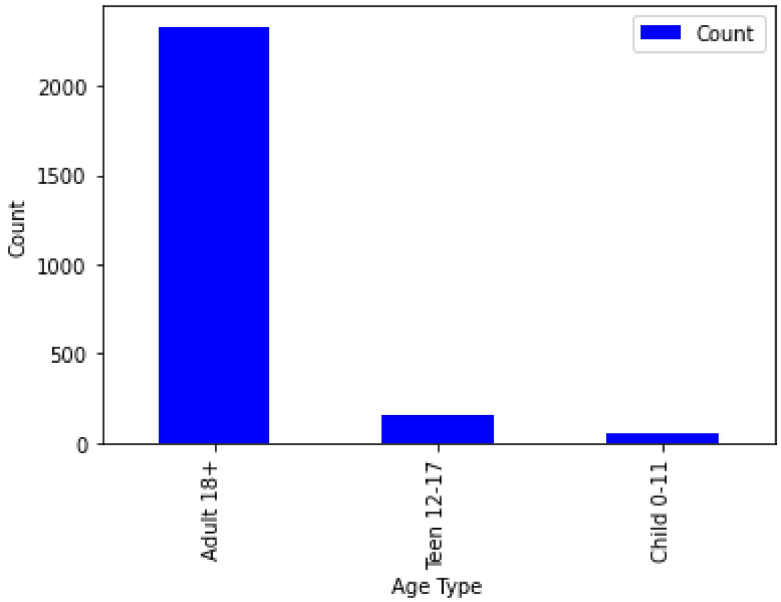
```
age_types = {}

def age_count(age_type):
    if str(age_type) != 'nan':
        ages = age_type.split("||")
        if len(ages) == 1:
            ages = ages[0].split("|")
        for age in ages:
            try:
                if age.split('::')[1] in age_types.keys():
                    age_types[age.split('::')[1]] += 1
                else:
                    age_types[age.split('::')[1]] = 0
            except:
                if age.split(':')[1] in age_types.keys():
                    age_types[age.split(':')[1]] += 1
                else:
                    age_types[age.split(':')[1]] = 0


df['participant_age_group'].apply(age_count)
```

Out[15]:

```
36        None
83        None
101       None
181       None
200       None
          ...
239279    None
239297    None
239335    None
239506    None
239511    None
Name: participant_age_group, Length: 1059, dtype: object
```

In [16]:

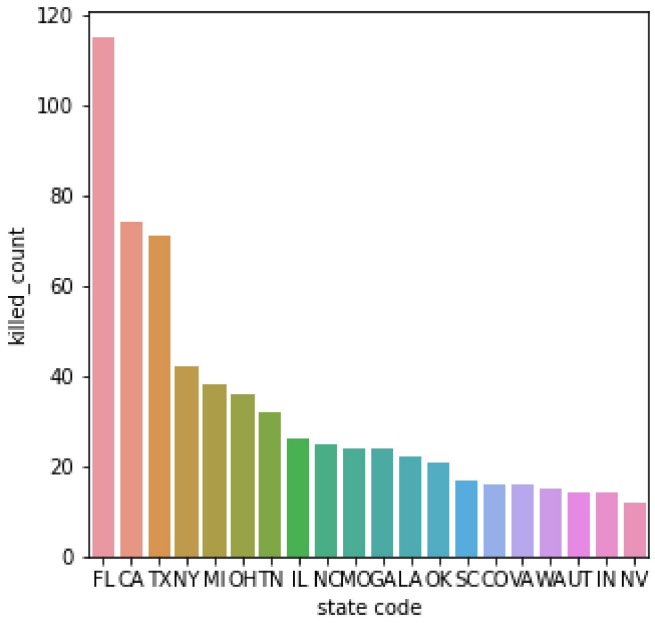
```
# Identify what particular ages are particularly engaged in these types
age_types = {k: v for k, v in sorted(age_types.items(), key=lambda iter
age_df = pd.DataFrame(age_types.items(), columns = ['Age Type', 'Count'
fig = px.bar(age_df, x='Age Type', y='Count')

age_df.plot(kind='bar', x='Age Type', y='Count', color='Blue')

# Add labels and title
plt.xlabel('Age Type')
plt.ylabel('Count')
plt.show()
#fig.update_layout(xaxis_tickangle=45)
#fig.show()
```



In [31]:

```
# Find the overall amount of people murdered or killed in each state.
states = {'Alabama': 'AL','Alaska': 'AK','American Samoa': 'AS','Arizor

df['state code'] = df['state'].apply(lambda x : states[x])
df_out = df.groupby('state code')['n_killed'].sum().sort_values(ascendi
df_out.rename(columns={'n_killed': 'killed_count'}, inplace=True)
top_10_states = df_out.head(20)
plt.figure(figsize=(5,5))
sns.barplot(x='state code', y='killed_count', data=top_10_states)
```

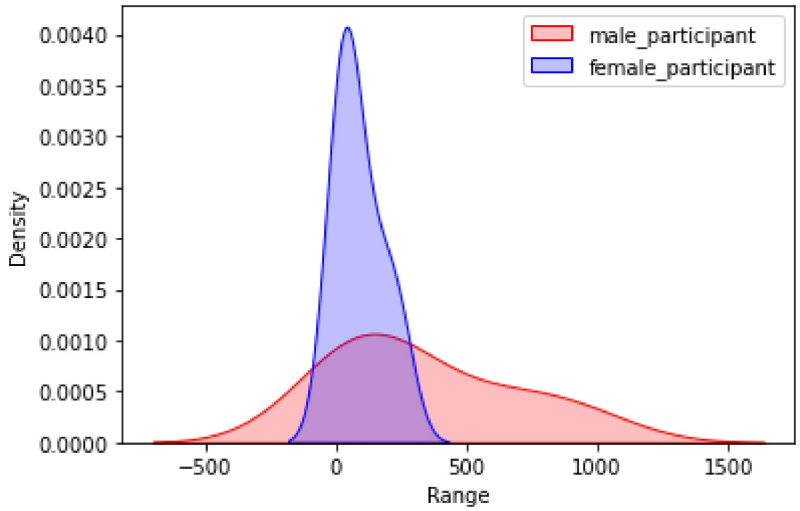
Out[31]: <AxesSubplot:xlabel='state code', ylabel='killed_count'>



In [18]:

```
# Identify which age group plays the biggest part in this violence.
df["participant_gender"] = df["participant_gender"].fillna("0::Unknown"

def gen(n) :
    gen_rows = []
    gen_row = str(n).split("||")
    for i in gen_row :
        g_row = str(i).split("::")
        if len(g_row) > 1 :
            gen_rows.append(g_row[1])

    return gen_rows

gen_series = df.participant_gender.apply(gen)
df["total_participant"] = gen_series.apply(lambda x: len(x))
df["male_participant"] = gen_series.apply(lambda i: i.count("Male"))
df["female_participant"] = gen_series.apply(lambda i: i.count("Female")
df["unknown_participant"] = gen_series.apply(lambda i: i.count("Unknown

genderwise_total = df[["total_participant", "male_participant", "female
dp_gen_plot=sns.kdeplot(genderwise_total['male_participant'], shade=Tru
dp_gen_plot=sns.kdeplot(genderwise_total['female_participant'], shade=T
plt.legend()
plt.xlabel('Range')
#del(genderwise_total)
```

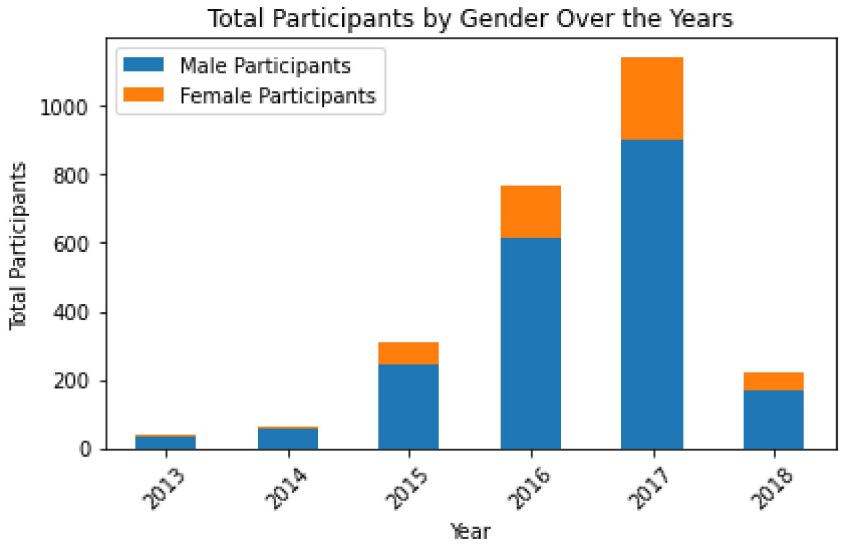
Out[18]: Text(0.5, 0, 'Range')



In [19]:

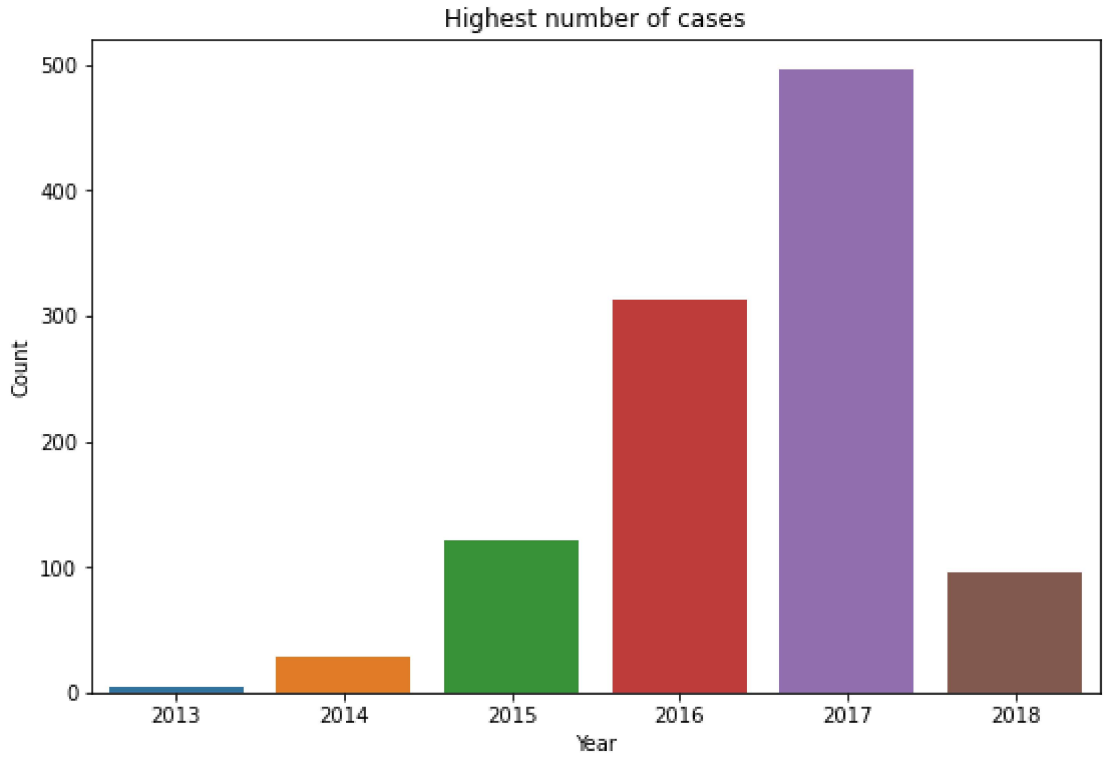
```
plt.figure(figsize=(10, 6))
genderwise_total[['male_participant', 'female_participant']].plot(kind=
plt.xlabel('Year')
plt.ylabel('Total Participants')
plt.title('Total Participants by Gender Over the Years')
plt.xticks(rotation=45)
plt.legend(['Male Participants', 'Female Participants'])
plt.tight_layout()
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



In [20]:

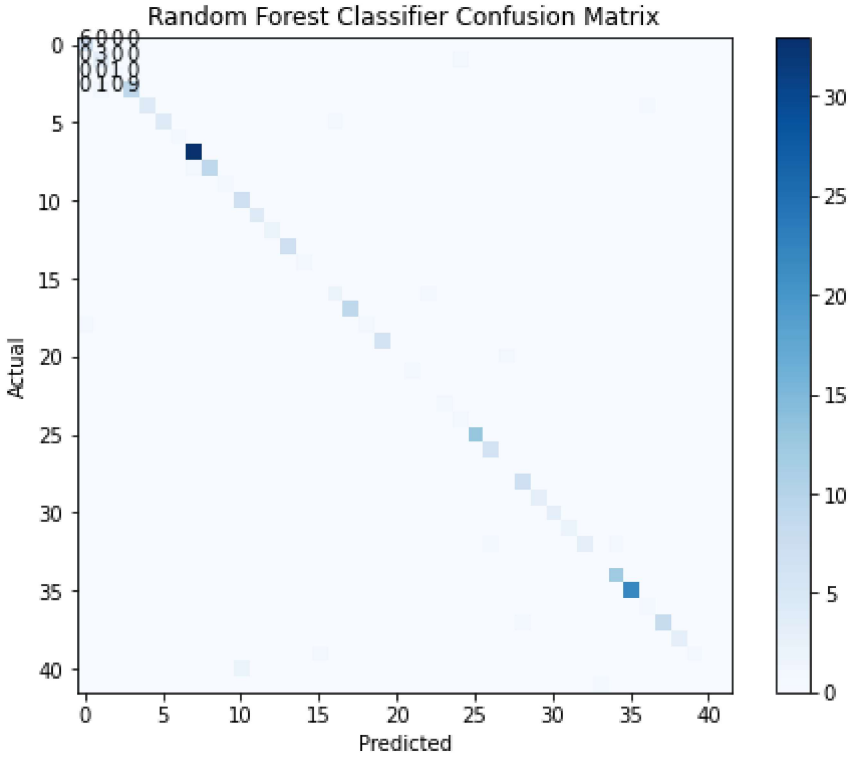
```
# In which year did the most cases occur?
df['date']=pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year
df['month']=df['date'].dt.month


plt.figure(figsize=(9,6))
sns.countplot(x='year', data=df)
plt.xlabel('Year')
plt.ylabel('Count')
plt.title('Highest number of cases')

# Show the plot
plt.show()
```



In [21]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split

# Define features and target variable
X = df[['n_killed', 'n_injured', 'congressional_district', 'latitude',
y = df['state']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Initialize and train the Random Forest model
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Random Forest Classifier Accuracy: {accuracy:.2f}')

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.title('Random Forest Classifier Confusion Matrix')
plt.colorbar()
#plt.xticks(ticks=range(len(df['state'].unique())), labels=df['state'].
#plt.yticks(ticks=range(len(df['state'].unique())), labels=df['state'].
plt.xlabel('Predicted')
plt.ylabel('Actual')
for i in range(len(df['state'].head(5).unique())):
    for j in range(len(df['state'].head(5).unique())):
        plt.text(j, i, format(cm[i, j], 'd'), horizontalalignment="cent
plt.show()

```

```
Random Forest Classifier Accuracy: 0.93
```

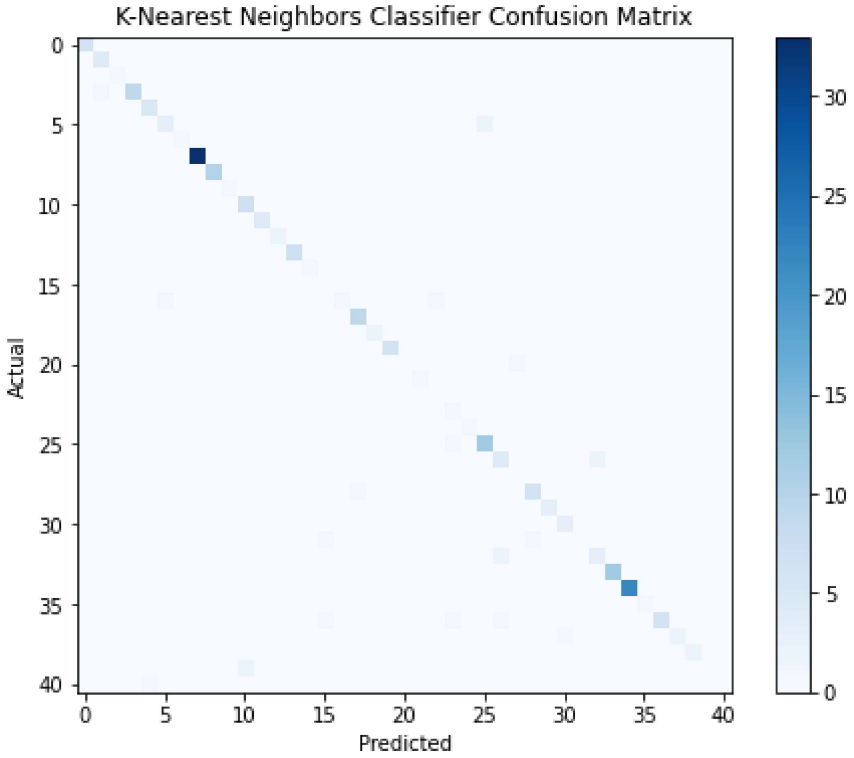

Random Forest Classifier Confusion Matrix
6000
0300
0010
0109
Actual
Predicted
0
5
10
15
20
25
30
35
40
0
5
10
15
20
25
30

In [22]:

```
from sklearn.neighbors import KNeighborsClassifier

# Initialize and train the K-Nearest Neighbors model
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)

# Make predictions
y_pred = knn_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'K-Nearest Neighbors Classifier Accuracy: {accuracy:.2f}')

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.title('K-Nearest Neighbors Classifier Confusion Matrix')
plt.colorbar()
#plt.xticks(ticks=range(len(data['state'].unique())), labels=data['stat
#plt.yticks(ticks=range(len(data['state'].unique())), labels=data['stat
plt.xlabel('Predicted')
plt.ylabel('Actual')
for i in range(len(data['state'].head(5).unique())):
    for j in range(len(data['state'].head(5).unique())):
        plt.text(j, i, format(cm[i, j], 'd'), horizontalalignment="cent
plt.show()

```

```
K-Nearest Neighbors Classifier Accuracy: 0.90
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp/ipykernel_24236/4187023220.py in <module>
     22 plt.xlabel('Predicted')
     23 plt.ylabel('Actual')
---> 24 for i in range(len(data['state'].head(5).unique())):
     25     for j in range(len(data['state'].head(5).unique())):
     26         plt.text(j, i, format(cm[i, j], 'd'), horizontalalignment
="center", color="white" if cm[i, j] > cm.max() / 2 else "black")

NameError: name 'data' is not defined
```


K-Nearest Neighbors Classifier Confusion Matrix
Actual
Predicted
0
5
10
15
20
25
30
35
40
0
5
10
15
20
25
30

In [31]:

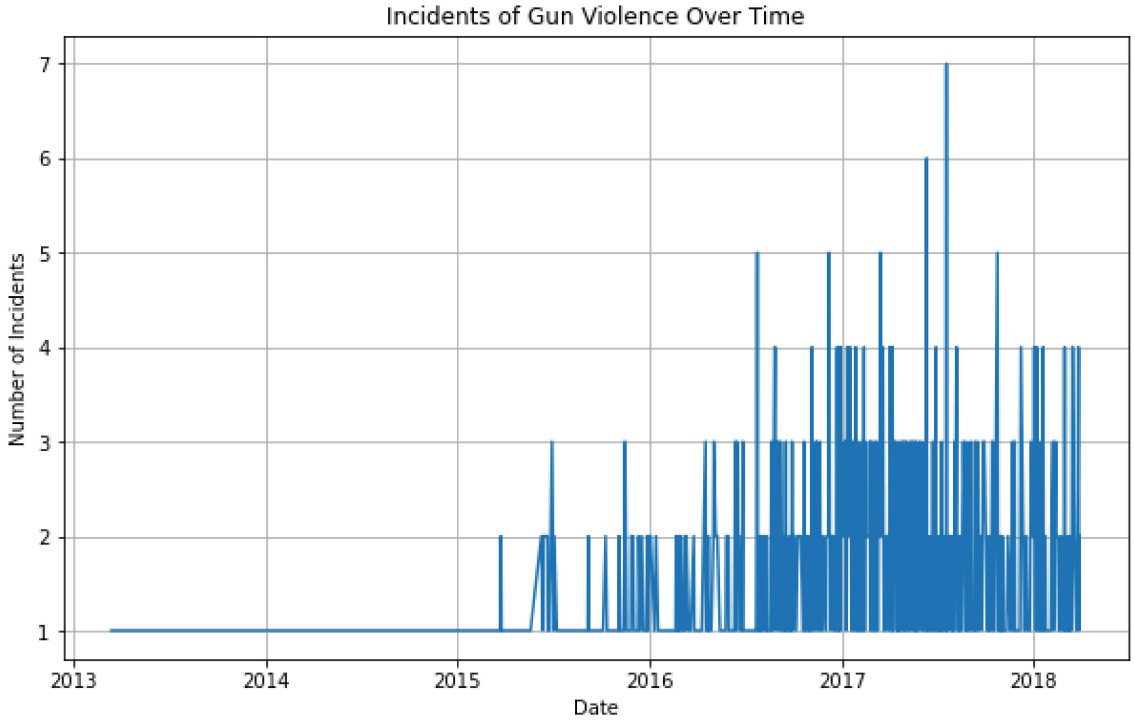
```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2

# Select the relevant columns
data = df[['state', 'n_killed']]

# Group the data by state and calculate the total number of killings fo
state_killed = data.groupby('state')['n_killed'].sum().reset_index()

# Convert the 'state' column into dummy variables (one-hot encoding)
state_dummies = pd.get_dummies(state_killed['state'], drop_first=True)

# Concatenate the dummy variables with the 'n_killed' column
X = pd.concat([state_dummies, state_killed['n_killed']], axis=1)

# Split the data into features (X) and target variable (y)
y = state_killed['n_killed']

# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Print the coefficients
print('Intercept:', model.intercept_)
print('Coefficients:', model.coef_)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
model = LinearRegression()
model.fit(X_train, y_train)
train_predictions = model.predict(X_train)

# Evaluate the model
#accuracy = accuracy_score(y_test, y_pred)
#print(f'Linear Regression Accuracy: {accuracy:.2f}')

# Evaluate the model on the training set
train_mae = mean_absolute_error(y_train, train_predictions)
train_mse = mean_squared_error(y_train, train_predictions)
train_r2 = r2_score(y_train, train_predictions)
print('Training Set Metrics:')
print('MAE:', train_mae)
print('MSE:', train_mse)
print('R-squared:', train_r2)

# Make predictions on the testing set
test_predictions = model.predict(X_test)

# Evaluate the model on the testing set
test_mae = mean_absolute_error(y_test, test_predictions)
test_mse = mean_squared_error(y_test, test_predictions)
test_r2 = r2_score(y_test, test_predictions)
print('\nTesting Set Metrics:')
print('MAE:', test_mae)
print('MSE:', test_mse)
```

```
55 print('R-squared:', test_r2)
```

```
Intercept: 0.00041373603639627277
Coefficients: [-2.89615225e-04 -4.13736036e-05 -2.89615225e-04  2.64791063
e-03
  2.48241622e-04 -1.24120811e-04 -3.30988829e-04  4.34422838e-03
  5.79230451e-04 -2.89615225e-04  1.59726988e-14  6.61977658e-04
  1.65494415e-04 -2.06868018e-04 -3.30988829e-04 -3.30988829e-04
  4.96483244e-04 -3.72362433e-04 -2.89615225e-04 -2.06868018e-04
  1.15846090e-03  8.27472073e-05  5.79230451e-04 -3.72362433e-04
  8.27472073e-05 -2.89615225e-04  4.13736037e-05 -8.27472073e-05
  1.32395532e-03  6.20604055e-04 -2.89615225e-04  1.07571369e-03
  4.55109640e-04 -8.27472073e-05 -1.24120811e-04 -3.30988829e-04
  2.89615225e-04 -4.13736036e-04  9.10219280e-04  2.52378982e-03
  1.65494415e-04  2.48241622e-04  2.06868018e-04 -2.48241622e-04
 -1.24120811e-04 -4.13736036e-04  9.99958626e-01]
Training Set Metrics:
MAE: 3.19776639946174e-15
MSE: 1.5254969233101717e-29
R-squared: 1.0

Testing Set Metrics:
MAE: 0.0007397303394362176
MSE: 1.4061917316493295e-06
R-squared: 0.9999999968646085
```

In [24]:

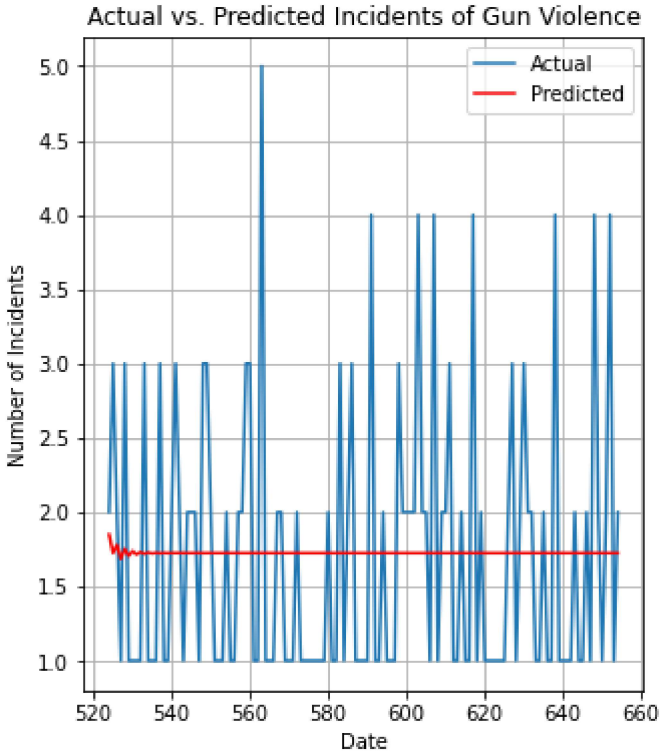
```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error

df['date'] = pd.to_datetime(df['date'])

# Group the data by date and count the number of incidents for each dat
time_series_data = df.groupby('date').size().reset_index(name='incident

# Set 'date' column as the index
time_series_data.set_index('date', inplace=True)

# Plot the time series data
plt.figure(figsize=(10, 6))
plt.plot(time_series_data.index, time_series_data['incident_count'])
plt.title('Incidents of Gun Violence Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Incidents')
plt.grid(True)
plt.show()
```



In [25]:

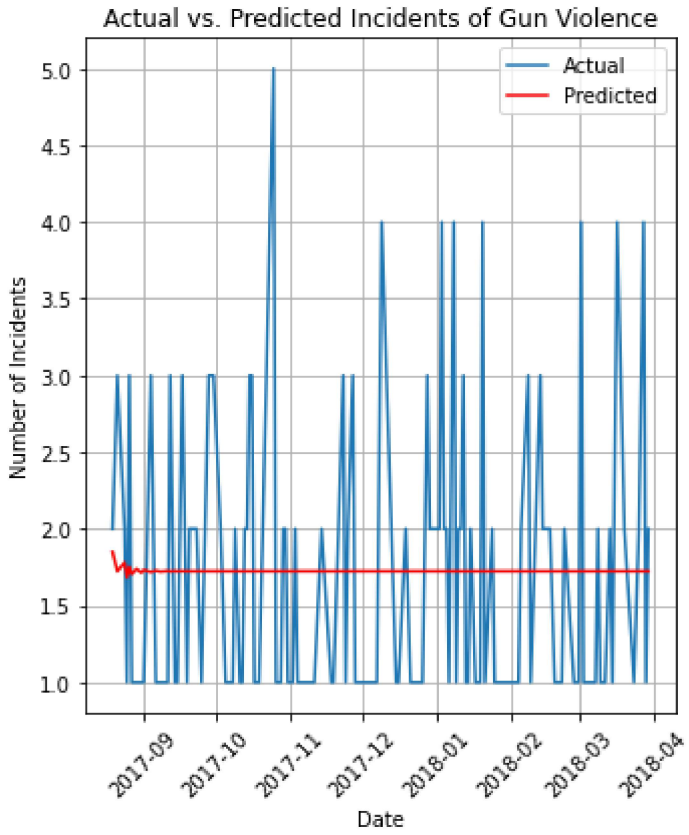
```
# Perform Time-Series Analysis
# Reset the index to ensure sequential indexing
time_series_data.reset_index(inplace=True)

# Split the data into training and testing sets
train_size = int(0.8 * len(time_series_data))
train_data = time_series_data.iloc[:train_size]
test_data = time_series_data.iloc[train_size:]

# Define and train the ARIMA model
order = (1, 1, 4)  # Example order for ARIMA model (p, d, q)
model = ARIMA(train_data['incident_count'], order=order)
arima_model = model.fit()

# Make predictions for the testing set
predictions = arima_model.forecast(steps=len(test_data))

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(test_data['incident_count'], predictions)
print('Mean Squared Error (MSE):', mse)

# Plot the actual vs. predicted values
plt.figure(figsize=(5, 6))
plt.plot(test_data.index, test_data['incident_count'], label='Actual')
plt.plot(test_data.index, predictions, label='Predicted', color='red')
plt.title('Actual vs. Predicted Incidents of Gun Violence')
plt.xlabel('Date')
plt.ylabel('Number of Incidents')
plt.legend()
plt.grid(True)
plt.show()
```

```
Mean Squared Error (MSE): 0.84925056608664
```



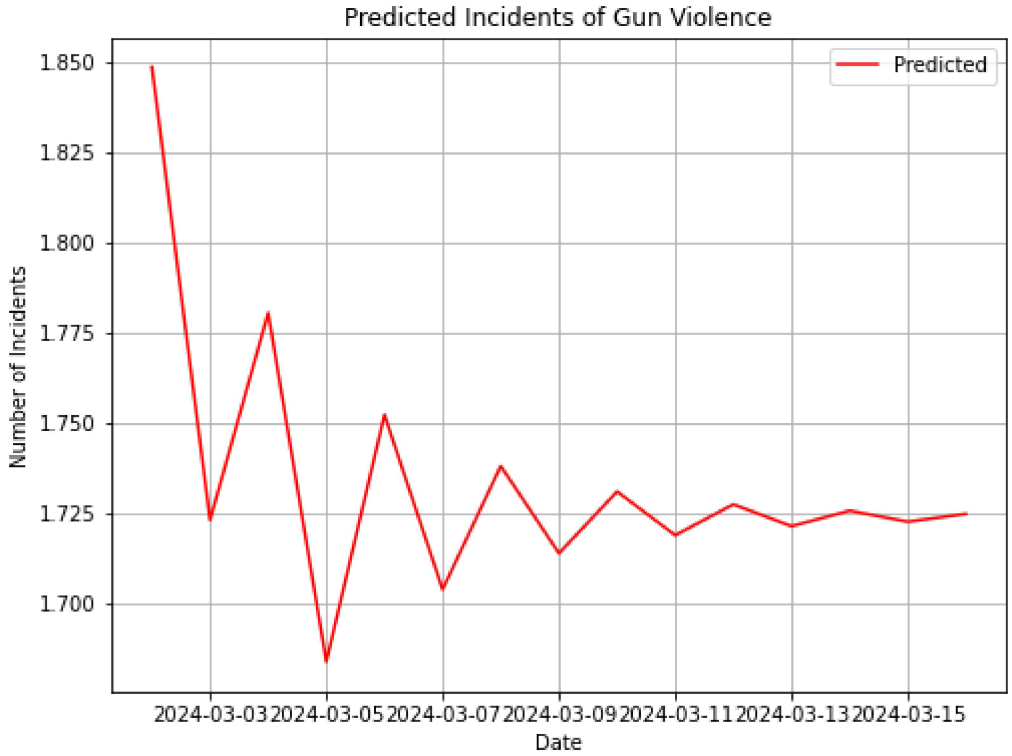
In [30]:

```
plt.figure(figsize=(5, 6))
plt.plot(test_data['date'], test_data['incident_count'], label='Actual'
plt.plot(test_data['date'], predictions, label='Predicted', color='red'
plt.title('Actual vs. Predicted Incidents of Gun Violence')
plt.xlabel('Date')
plt.ylabel('Number of Incidents')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
```



In [97]:

```
## Total Crime Incidents Predictions For Future Days:
today = pd.Timestamp.now().normalize()

# Define Future Dates
future_dates = pd.date_range(start=today, periods=15, freq='D')

# Generate Forecast
forecast = arima_model.forecast(steps=len(future_dates))

# Create DataFrame for Predictions
predicted_df = pd.DataFrame({
    'Date': future_dates,
    'Predicted Incidents': forecast
})

# Plot Predictions
plt.figure(figsize=(8, 6))
#plt.plot(time_series_data.index, time_series_data['incident_count'], l
plt.plot(predicted_df['Date'], predicted_df['Predicted Incidents'], lab
plt.title('Predicted Incidents of Gun Violence')
plt.xlabel('Date')
plt.ylabel('Number of Incidents')
plt.legend()
plt.grid(True)
plt.show()

# Display Predicted DataFrame
print(predicted_df)

```



	Date	Predicted Incidents
524	2024-03-02	1.848574
525	2024-03-03	1.723093
526	2024-03-04	1.780517
527	2024-03-05	1.683842
528	2024-03-06	1.752223
529	2024-03-07	1.703855
530	2024-03-08	1.738068
531	2024-03-09	1.713868
532	2024-03-10	1.730985
533	2024-03-11	1.718877
534	2024-03-12	1.727442
535	2024-03-13	1.721384
536	2024-03-14	1.725669
537	2024-03-15	1.722638
538	2024-03-16	1.724782