In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
df = pd.read_csv("C:\\Users\\amit9\\Downloads\\py.csv\\Israel-Palestine-conflict.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	Year	Month	Palestinians Injuries	Israelis Injuries	Palestinians Killed	Israelis Killed
0	2000.0	DECEMBER	781	NaN	51	8
1	2000.0	NOVEMBER	3838	NaN	112	22
2	2000.0	OCTOBER	5984	NaN	104	10
3	2000.0	SEPTEMBER	NaN	NaN	16	1
4	2001.0	DECEMBER	304	NaN	67	36

In [4]:
```
df.shape
```

Out[4]:
```
(251, 6)
```

Data Cleansing: Will work on Nan value and data type

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Year                    2
Month                   2
Palestinians Injuries    55
Israelis Injuries       118
Palestinians Killed      1
Israelis Killed          1
dtype: int64
```

In [6]:
```
# Here, we have all columns having NAN values.
#We can remove the whole row with lesser NAN value.
#But columns with higher number of NAN value are the work around area. I used mean function to fillna.
```

In [7]:
```
# to replace the NAN value with the mean of same column, I have converted these columns to numeric data using pandas.
df['Palestinians Injuries'] = pd.to_numeric(df['Palestinians Injuries'], errors='coerce')
df['Israelis Injuries'] = pd.to_numeric(df['Israelis Injuries'], errors='coerce')

df['Palestinians Injuries'].fillna(df['Palestinians Injuries'].mean(), inplace=True)
df['Israelis Injuries'].fillna(df['Israelis Injuries'].mean(), inplace=True)
```

In [8]:
```
# coding Month from character to numeric data type
month_to_numeric = {
    'JANUARY': 1, 'FEBRUARY': 2, 'MARCH': 3, 'APRIL': 4,
    'MAY': 5, 'JUNE': 6, 'JULY': 7, 'AUGUST': 8,
    'SEPTEMBER': 9, 'OCTOBER': 10, 'NOVEMBER': 11, 'DECEMBER': 12
}

df['Month'] = df['Month'].map(month_to_numeric)
```

In [9]:
```
# wanted to look on remaing rows with NAN values.
nan_rows = df[df.isna().any(axis=1)]
nan_rows
```

Out[9]:

	Year	Month	Palestinians Injuries	Israelis Injuries	Palestinians Killed	Israelis Killed
203	2017.0	NaN	577.590674	39.692308	6	0
227	2019.0	NaN	577.590674	39.692308	33	4
239	2020.0	NaN	577.590674	39.692308	4	1
249	NaN	NaN	577.590674	39.692308	NaN	NaN
250	NaN	NaN	577.590674	39.692308	10,000	1,275

In [10]:
```
df = df.dropna()
```

In [11]:
```
# We are good with NAN values.
df.isnull().sum()
```

Out[11]:
```
Year                   0
Month                  0
Palestinians Injuries  0
Israelis Injuries      0
Palestinians Killed    0
Israelis Killed        0
dtype: int64
```

In [12]:
```
df.head()
```

Out[12]:

	Year	Month	Palestinians Injuries	Israelis Injuries	Palestinians Killed	Israelis Killed
0	2000.0	12.0	781.000000	39.692308	51	8
1	2000.0	11.0	3838.000000	39.692308	112	22
2	2000.0	10.0	5984.000000	39.692308	104	10
3	2000.0	9.0	577.590674	39.692308	16	1
4	2001.0	12.0	304.000000	39.692308	67	36

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 246 entries, 0 to 248
Data columns (total 6 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Year                   246 non-null    float64
 1   Month                  246 non-null    float64
 2   Palestinians Injuries  246 non-null    float64
 3   Israelis Injuries      246 non-null    float64
 4   Palestinians Killed    246 non-null    object
 5   Israelis Killed        246 non-null    object
dtypes: float64(4), object(2)
memory usage: 13.5+ KB
```

In [14]:
```
# here, I have prepared whole dataset to numeric.
df = df.apply(pd.to_numeric, errors='coerce')
df.fillna(0, inplace=True)

df = df.astype(int)
```

In [15]:
```
df.head()
```

Out[15]:

	Year	Month	Palestinians Injuries	Israelis Injuries	Palestinians Killed	Israelis Killed
0	2000	12	781	39	51	8
1	2000	11	3838	39	112	22
2	2000	10	5984	39	104	10
3	2000	9	577	39	16	1
4	2001	12	304	39	67	36

Data Visualization:

In [16]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [17]:
```
# Data belongs to Year
print("Max :", df["Year"].max(), "\nMin :",df["Year"].min())
```

```
Max : 2021
Min : 2000
```

In [18]:
```
total_injuries = df['Palestinians Injuries'].sum() + df['Israelis Injuries'].sum()
total_death = df['Palestinians Killed'].sum() + df['Israelis Killed'].sum()
total_incident = total_injuries+total_death
print("General Statistics:",
      "\nTotal injuries :", total_injuries,
      "\nTotal Death :", total_death,
     "\nTotal Incident :", total_incident)
```

```
General Statistics:
Total injuries : 151740
Total Death : 11227
Total Incident : 162967
```

In [19]:
```
print("Palestinian Statistics:", "\n")
```

```
max_injuries_row = df.loc[df['Palestinians Injuries'].idxmax()]
min_injuries_row = df.loc[df['Palestinians Injuries'].idxmin()]


# year having max/min injuries-Palestinians
print("*****Injuries*****")

print("Year with Maximum injuries", int(max_injuries_row['Year']),
      "& with number of", int(max_injuries_row['Palestinians Injuries']),
     "\nYear with Minimum injuries",int(min_injuries_row['Year']),
      "& with number of", int(min_injuries_row['Palestinians Injuries']),
     "\nInjured Rate of Palestinians :", df['Palestinians Injuries'].sum()/total_incident*100)

# year having max/min death-Palestinians
print("*****Death*****")

max_death_row = df.loc[df['Palestinians Killed'].idxmax()]
min_death_row = df.loc[df['Palestinians Killed'].idxmin()]

print("Year with Maximum Death", int(max_death_row['Year']),
      "& with number of", int(max_death_row['Palestinians Killed']),
     "\nYear with Minimum Death",int(min_death_row['Year']),
      "& with number of", int(min_death_row['Palestinians Killed']),
          "\nDeath Rate of Palestinians :", df['Palestinians Killed'].sum()/total_incident*100)

print("\n")

print("Israelis Statistics:", "\n")

max_injuries_row = df.loc[df['Israelis Injuries'].idxmax()]
min_injuries_row = df.loc[df['Israelis Injuries'].idxmin()]

print("*****Injuries*****")
# year having max/min injuries-Israelis

print("Year with Maximum injuries", int(max_injuries_row['Year']),
      "& with number of", int(max_injuries_row['Israelis Injuries']),
     "\nYear with Minimum injuries",int(min_injuries_row['Year']),
      "& with number of", int(min_injuries_row['Israelis Injuries']),
          "\nInjured Rate of Israelis :", df['Israelis Injuries'].sum()/total_incident*100)

print("*****Death*****")
# year having max/min death-Israelis
```

```
max_death_row = df.loc[df['Israelis Killed'].idxmax()]
min_death_row = df.loc[df['Israelis Killed'].idxmin()]

print("Year with Maximum Death", int(max_death_row['Year']),
      "& with number of", int(max_death_row['Israelis Killed']),
     "\nYear with Minimum Death",int(min_death_row['Year']),
      "& with number of", int(min_death_row['Israelis Killed']),
          "\nDeath Rate of Israelis :", df['Israelis Killed'].sum()/total_incident*100)
```

```
Palestinian Statistics:

*****Injuries*****
Year with Maximum injuries 2014 & with number of 13735
Year with Minimum injuries 2009 & with number of 26
Injured Rate of Palestinians : 87.16856786956868
*****Death*****
Year with Maximum Death 2014 & with number of 1590
Year with Minimum Death 2004 & with number of 0
Death Rate of Palestinians : 6.10982591567618


Israelis Statistics:

*****Injuries*****
Year with Maximum injuries 2014 & with number of 2347
Year with Minimum injuries 2010 & with number of 0
Injured Rate of Israelis : 5.942307338295484
*****Death*****
Year with Maximum Death 2002 & with number of 122
Year with Minimum Death 2004 & with number of 0
Death Rate of Israelis : 0.7792988764596513
```

Insights:

Injury Trends: Both Palestinians and Israelis experienced the highest number of injuries in 2014. Palestinians had a slightly higher injury rate compared to Israelis.

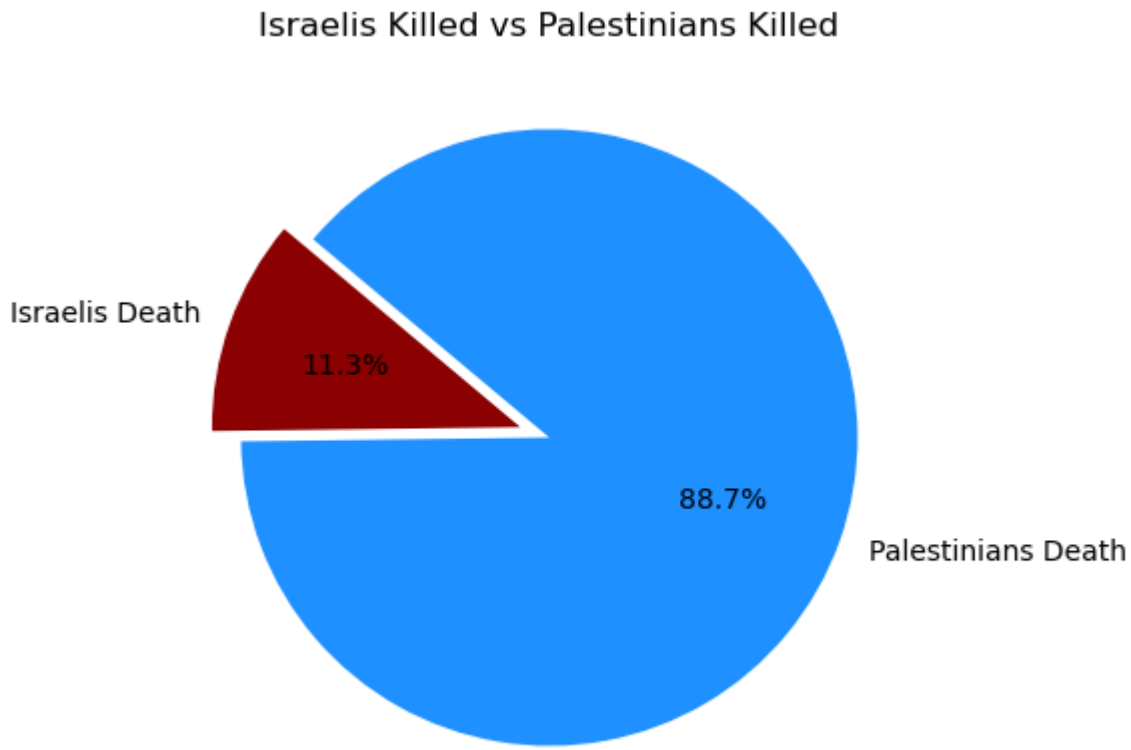
Death Trends: Palestinians experienced the highest number of deaths in 2014, whereas Israelis had the highest number in 2002. The death rate for Palestinians is considerably higher compared to Israelis.

Temporal Patterns: 2004 stands out as a year with no reported deaths for both Palestinians and Israelis. 2014 appears to be a year with significant casualties for both groups.

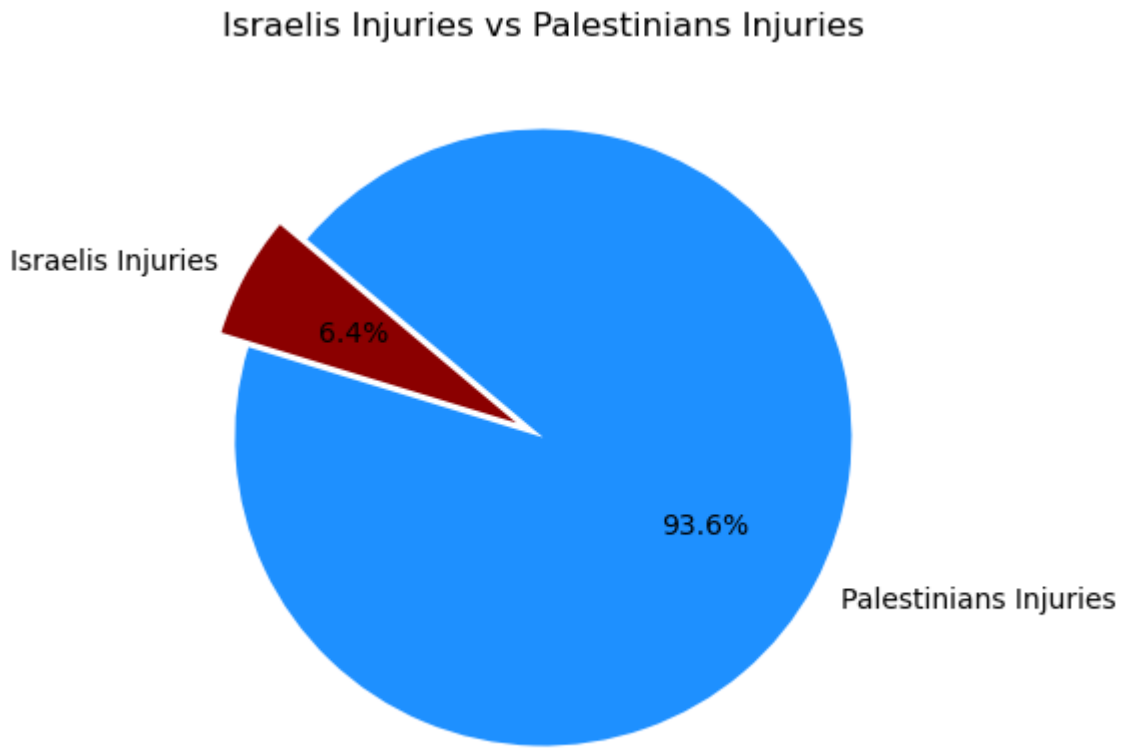
Comparative Risk: Palestinians have a higher death rate and injury rate compared to Israelis, indicating a higher risk faced by Palestinians in conflict situations.

Death Rate per Incident: The death rate per incident is significantly higher for Palestinians compared to Israelis, further highlighting the severity of incidents involving Palestinians.

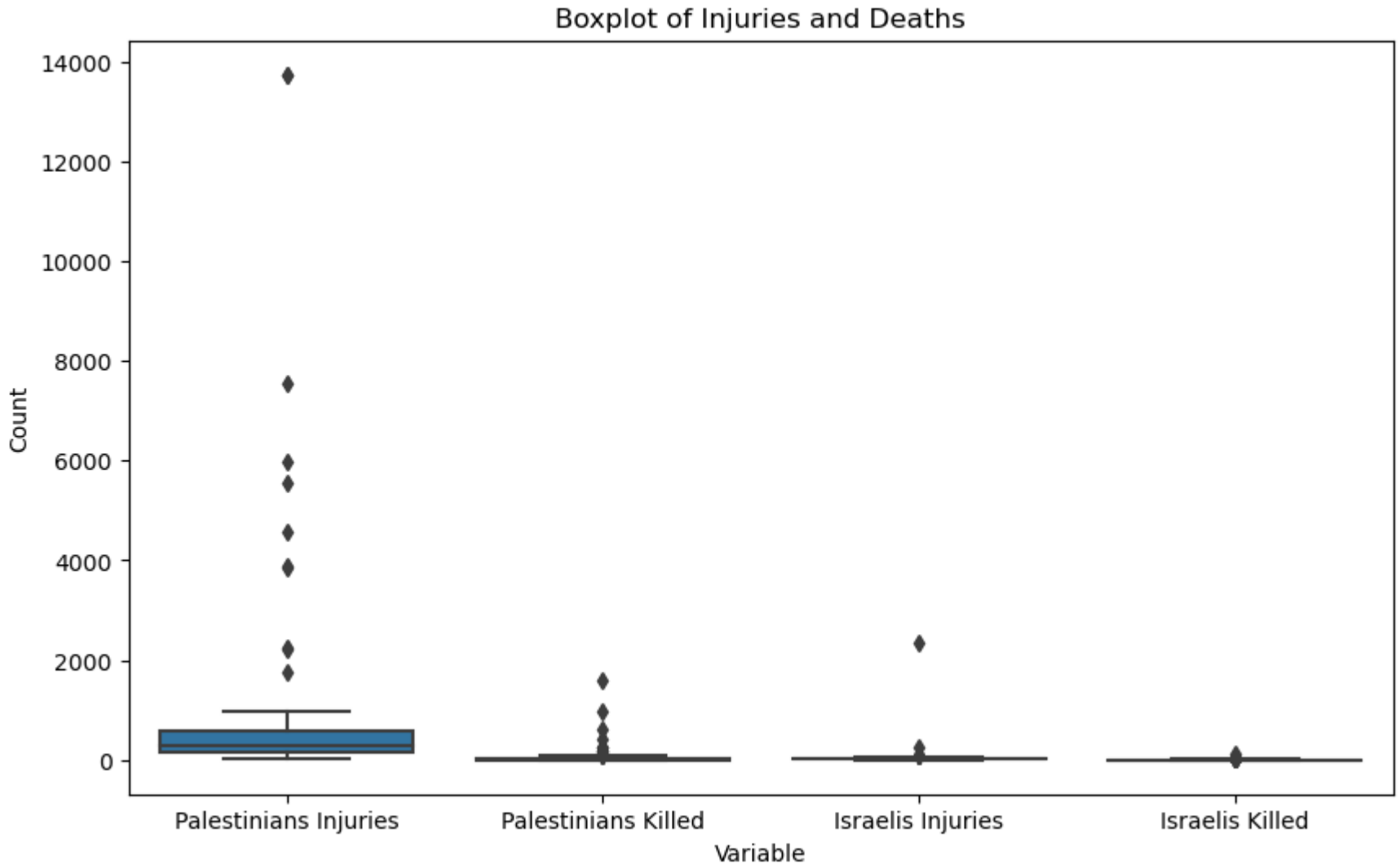
In [20]:
```
# Pie representation of death toll:
labels = ['Israelis Death', 'Palestinians Death']
sizes = [df['Israelis Killed'].sum(), df['Palestinians Killed'].sum()]

plt.figure(figsize=(10,5))
plt.pie(sizes, explode=(0.1,0), labels=labels, colors= ['#8b0000', '#1e90ff'],
       autopct='%1.1f%%', startangle=140)
plt.title("Israelis Killed vs Palestinians Killed")
plt.show()
```



In [21]:
```
# Pie representation of Injuries toll:
labels = ['Israelis Injuries', 'Palestinians Injuries']
sizes = [df['Israelis Injuries'].sum(), df['Palestinians Injuries'].sum()]

plt.figure(figsize=(10,5))
plt.pie(sizes, explode=(0.1,0), labels=labels, colors= ['#8b0000', '#1e90ff'],
       autopct='%1.1f%%', startangle=140)
plt.title("Israelis Injuries vs Palestinians Injuries")
plt.show()
```

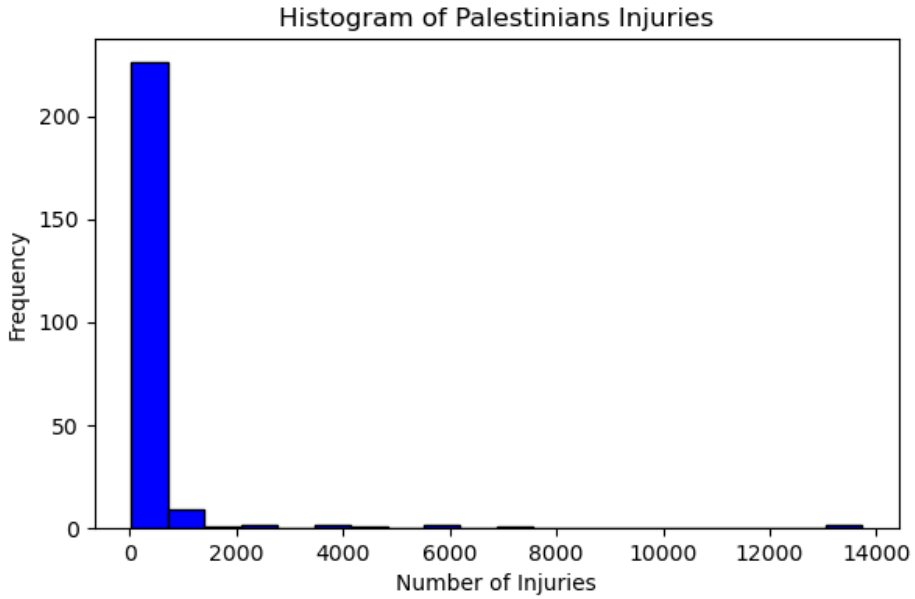


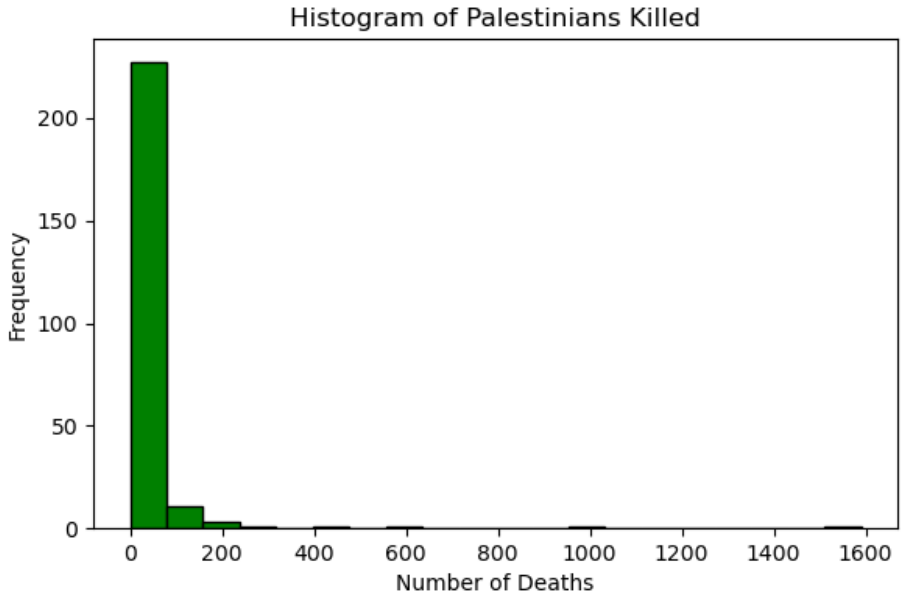
In [22]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df[['Palestinians Injuries', 'Palestinians Killed', 'Israelis Injuries', 'Israelis Killed']])
plt.title('Boxplot of Injuries and Deaths')
plt.xlabel('Variable')
plt.ylabel('Count')
plt.show()
```

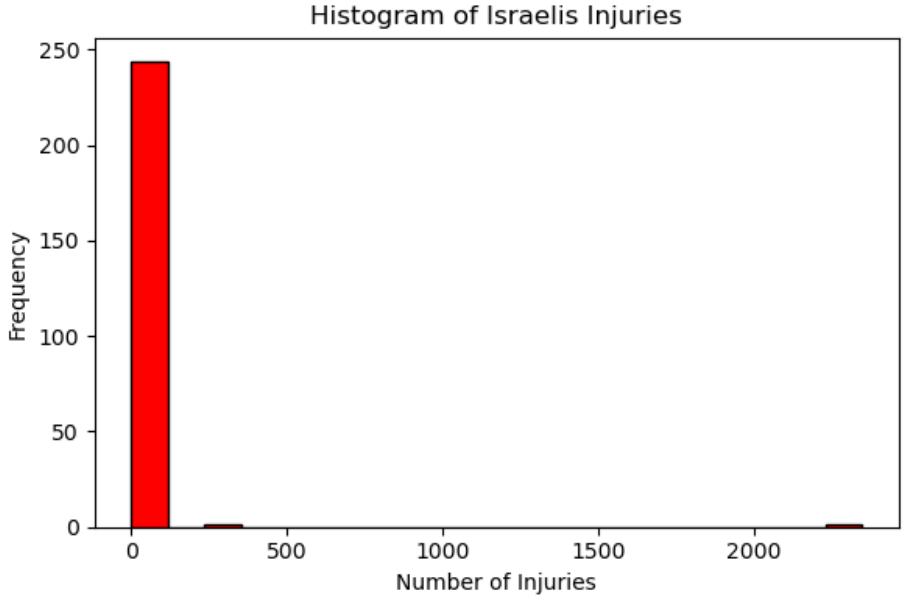


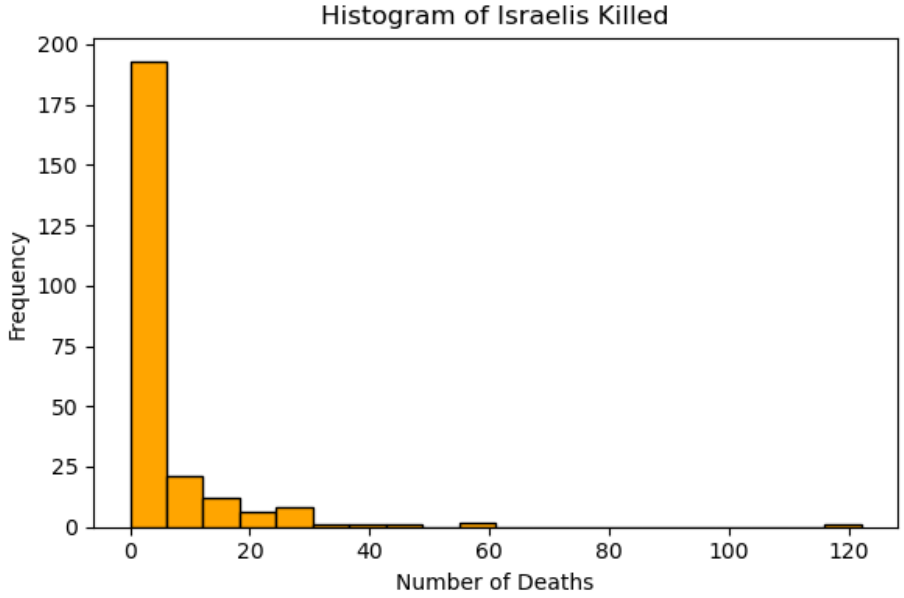
In [23]:
```
# Histograms representation:
plt.figure(figsize=(12, 8))

# Palestinians Injuries
plt.subplot(2, 2, 1)
plt.hist(df['Palestinians Injuries'], bins=20, color='blue', edgecolor='black')
plt.title('Histogram of Palestinians Injuries')
plt.xlabel('Number of Injuries')
plt.ylabel('Frequency')
```

```
# Palestinians Killed
plt.subplot(2, 2, 2)
plt.hist(df['Palestinians Killed'], bins=20, color='green', edgecolor='black')
plt.title('Histogram of Palestinians Killed')
plt.xlabel('Number of Deaths')
plt.ylabel('Frequency')

# Israelis Injuries
plt.subplot(2, 2, 3)
plt.hist(df['Israelis Injuries'], bins=20, color='red', edgecolor='black')
plt.title('Histogram of Israelis Injuries')
plt.xlabel('Number of Injuries')
plt.ylabel('Frequency')

# Israelis Killed
plt.subplot(2, 2, 4)
plt.hist(df['Israelis Killed'], bins=20, color='orange', edgecolor='black')
plt.title('Histogram of Israelis Killed')
plt.xlabel('Number of Deaths')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```









In [24]:
```
# Using IQR method:  Here I've looked in for outliers. My intension is not to remove outliers.
# To ensure transparency in my analysis by documenting the rationale behind the decision.

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
```

```
IQR = Q3-Q1
threshold = 1.5 #commonly used

outliers = ((df < (Q1-threshold*IQR)) | (df > (Q3+threshold*IQR))).any(axis=1)
```

In [25]:
```
df[outliers].head()
```

Out[25]:

	Year	Month	Palestinians Injuries	Israelis Injuries	Palestinians Killed	Israelis Killed
1	2000	11	3838	39	112	22
2	2000	10	5984	39	104	10
4	2001	12	304	39	67	36
5	2001	11	160	39	39	14
6	2001	10	407	39	89	14

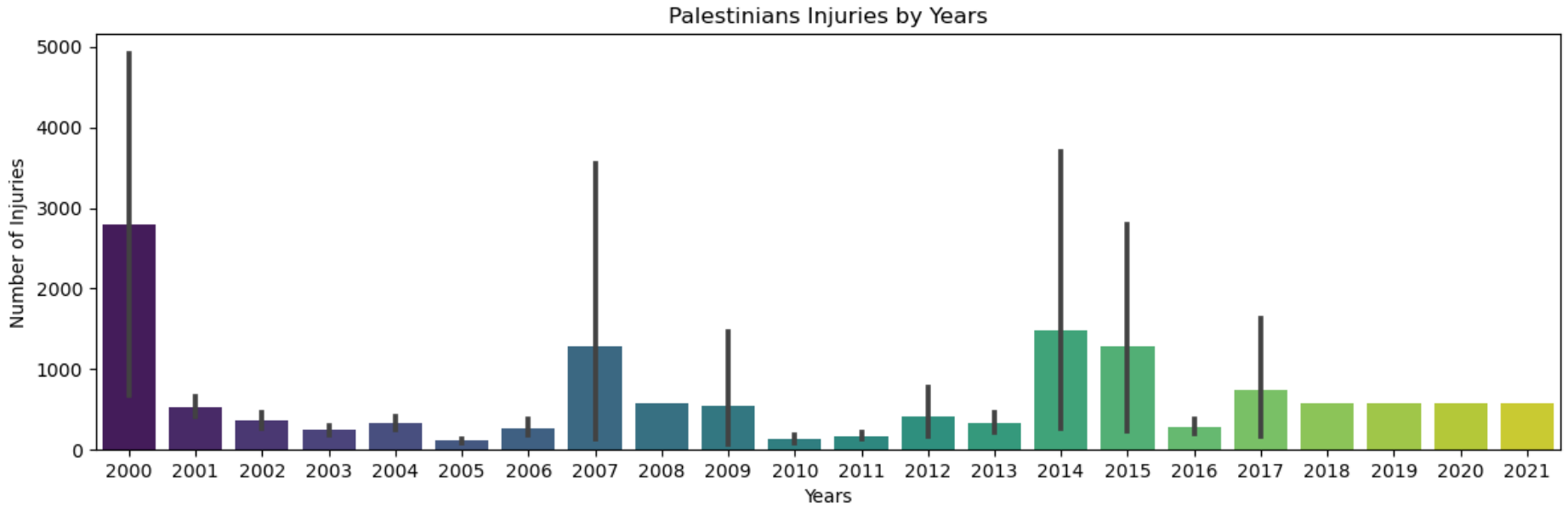
In [26]:
```
df[outliers].shape
```

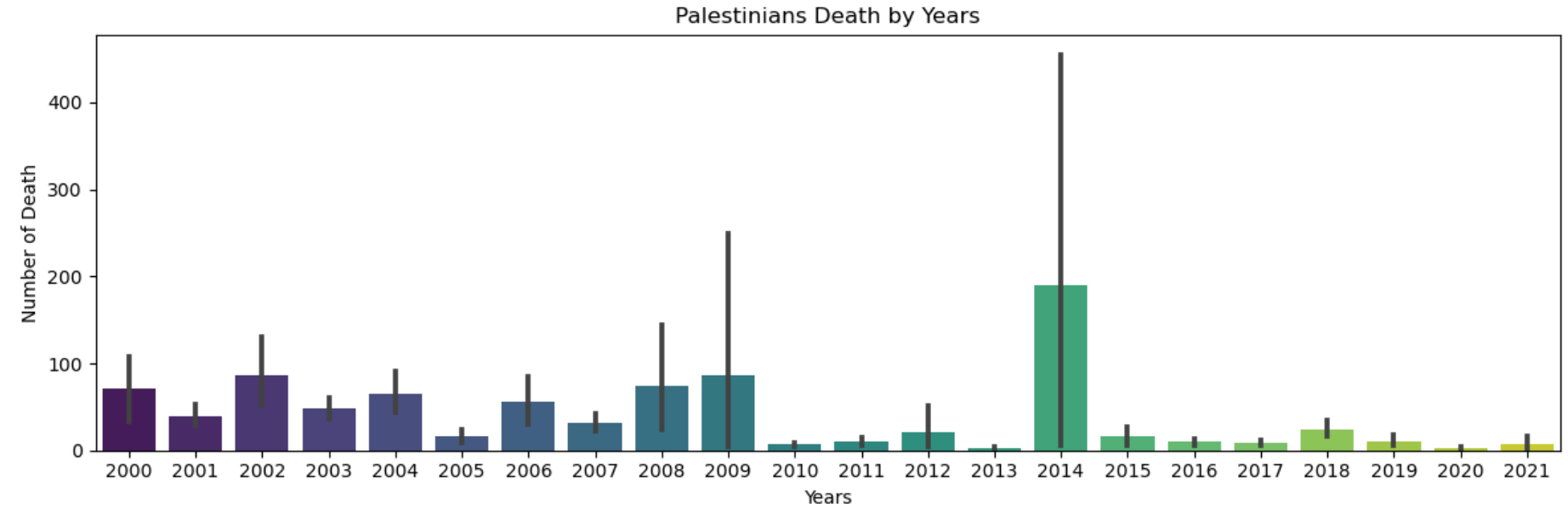
Out[26]:
```
(53, 6)
```

Palestinians:

In [27]:
```
plt.figure(figsize=(12, 8))

plt.subplot(2,1,1)
sns.barplot(data = df, x = df['Year'], y = df['Palestinians Injuries'], palette='viridis')
plt.title('Palestinians Injuries by Years')
plt.xlabel('Years')
plt.ylabel('Number of Injuries')


plt.subplot(2,1,2)
sns.barplot(data = df, x = df['Year'], y = df['Palestinians Killed'], palette='viridis')
plt.title('Palestinians Death by Years')
plt.xlabel('Years')
plt.ylabel('Number of Death')

plt.tight_layout()
```

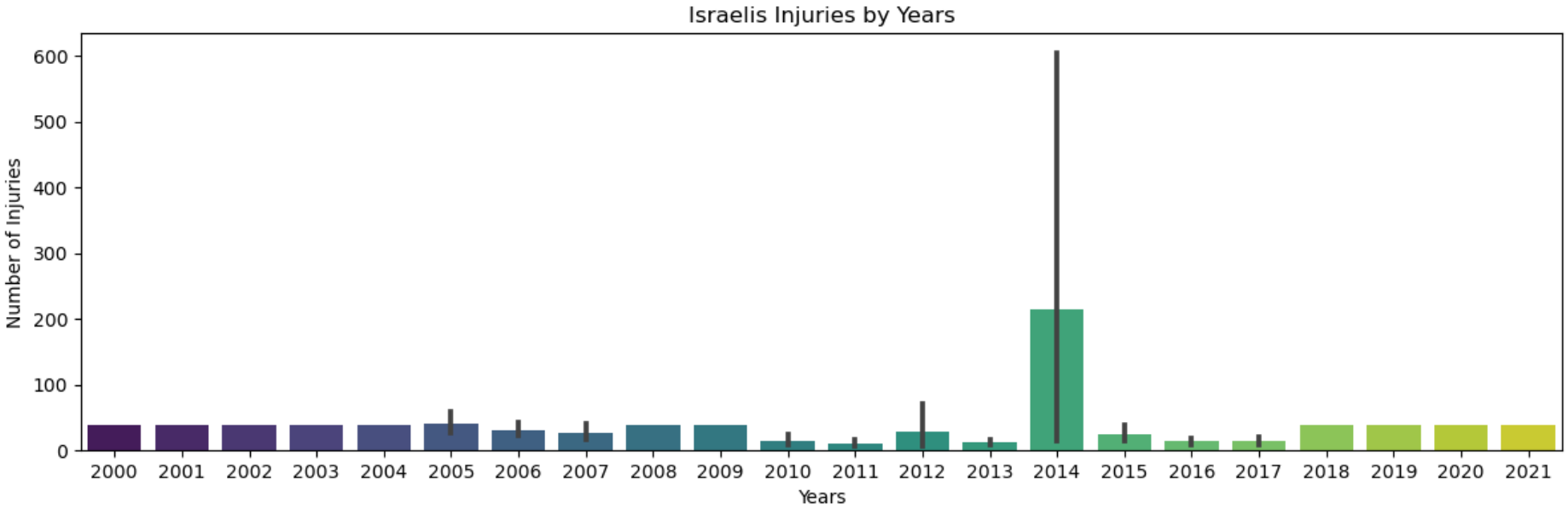
```
plt.show()
```

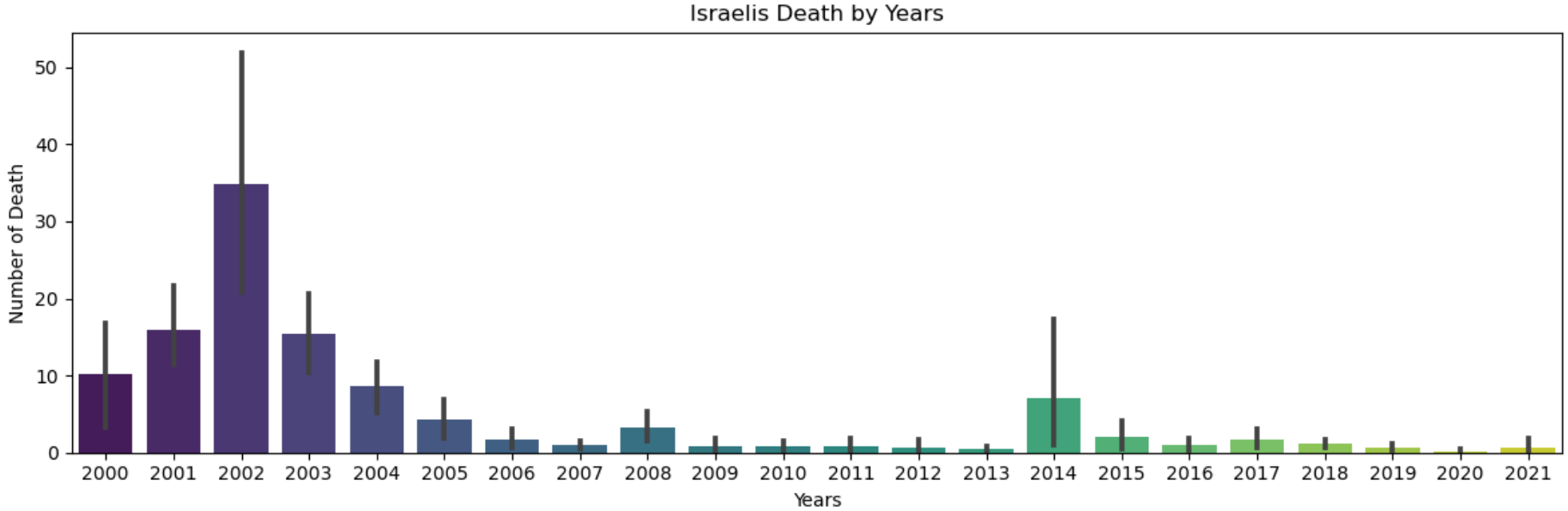

Palestinians Injuries by Years
Number of Injuries
0
1000
2000
3000
4000
5000
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021
Years


Palestinians Death by Years
Number of Death
0
100
200
300
400
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021
Years

Israelis:

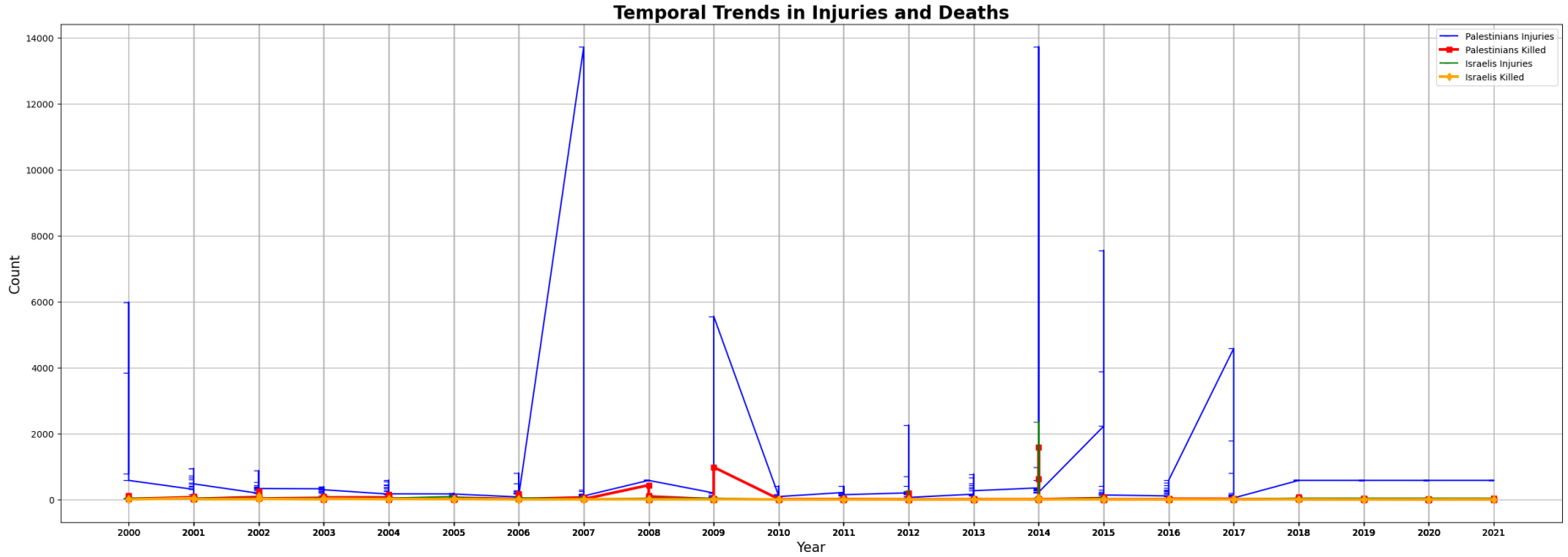
In [28]:
```
plt.figure(figsize=(12, 8))

plt.subplot(2,1,1)
sns.barplot(data = df, x = df['Year'], y = df['Israelis Injuries'], palette='viridis')
plt.title('Israelis Injuries by Years')
plt.xlabel('Years')
plt.ylabel('Number of Injuries')


plt.subplot(2,1,2)
sns.barplot(data = df, x = df['Year'], y = df['Israelis Killed'], palette='viridis')
plt.title('Israelis Death by Years')
plt.xlabel('Years')
plt.ylabel('Number of Death')

plt.tight_layout()

plt.show()
```





In [29]:
```
# Temporal Trends in Injuries and Deaths
plt.figure(figsize=(30,10))
plt.plot(df['Year'], df['Palestinians Injuries'],color='blue', marker=0, label='Palestinians Injuries')
plt.plot(df['Year'], df['Palestinians Killed'], color='red', marker='s', label='Palestinians Killed', linewidth=3)
plt.plot(df['Year'], df['Israelis Injuries'], color='green', marker=0, label='Israelis Injuries')
```

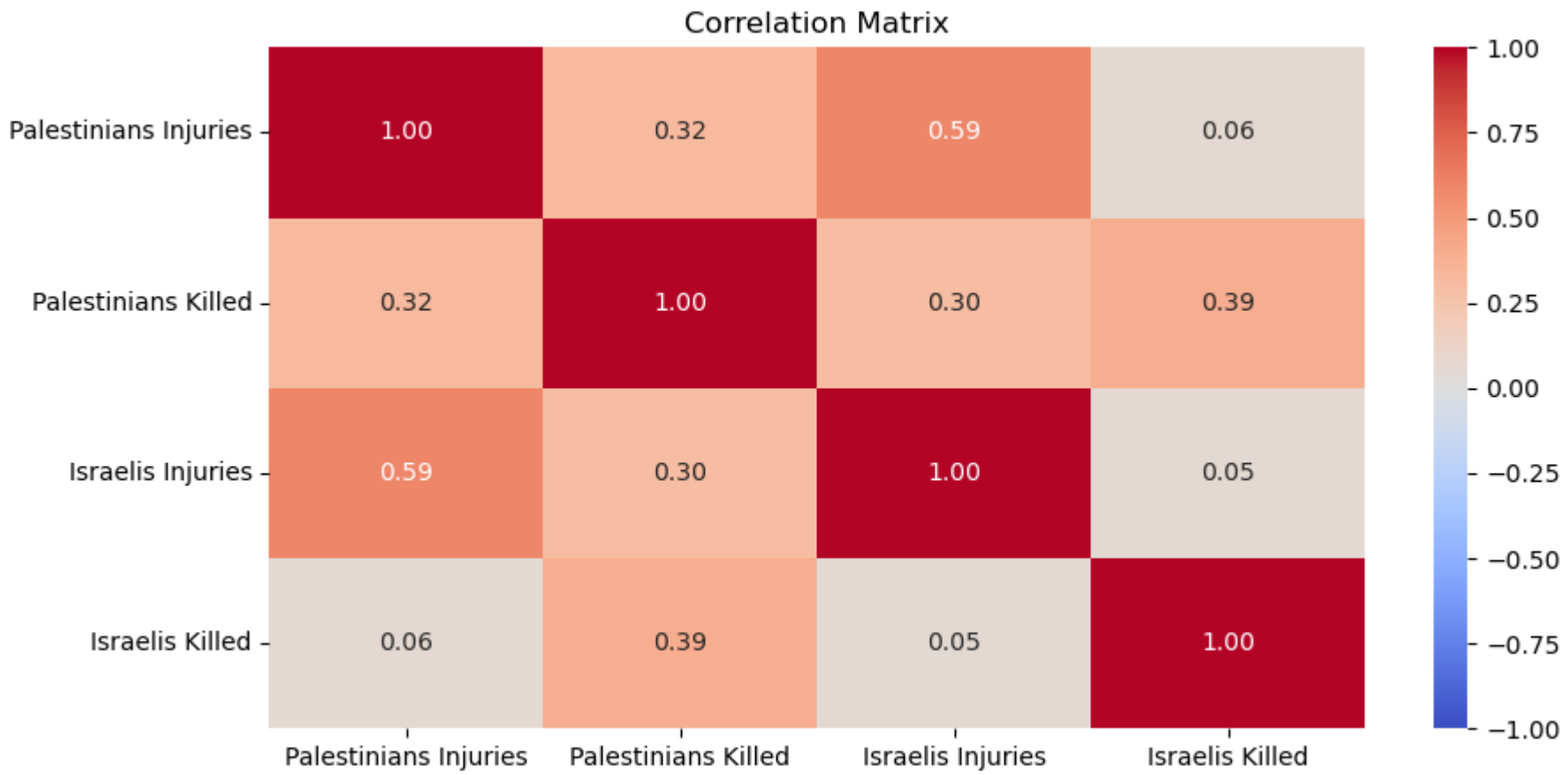
```
plt.plot(df['Year'], df['Israelis Killed'], color='orange', marker='d', label='Israelis Killed', linewidth=3)

plt.title("Temporal Trends in Injuries and Deaths", fontweight='bold', fontsize=20)
plt.xlabel('Year', fontsize=15)
plt.ylabel('Count', fontsize=15)
plt.xticks(df['Year'])
plt.legend()
plt.grid(True)
plt.show()
```



In [30]:

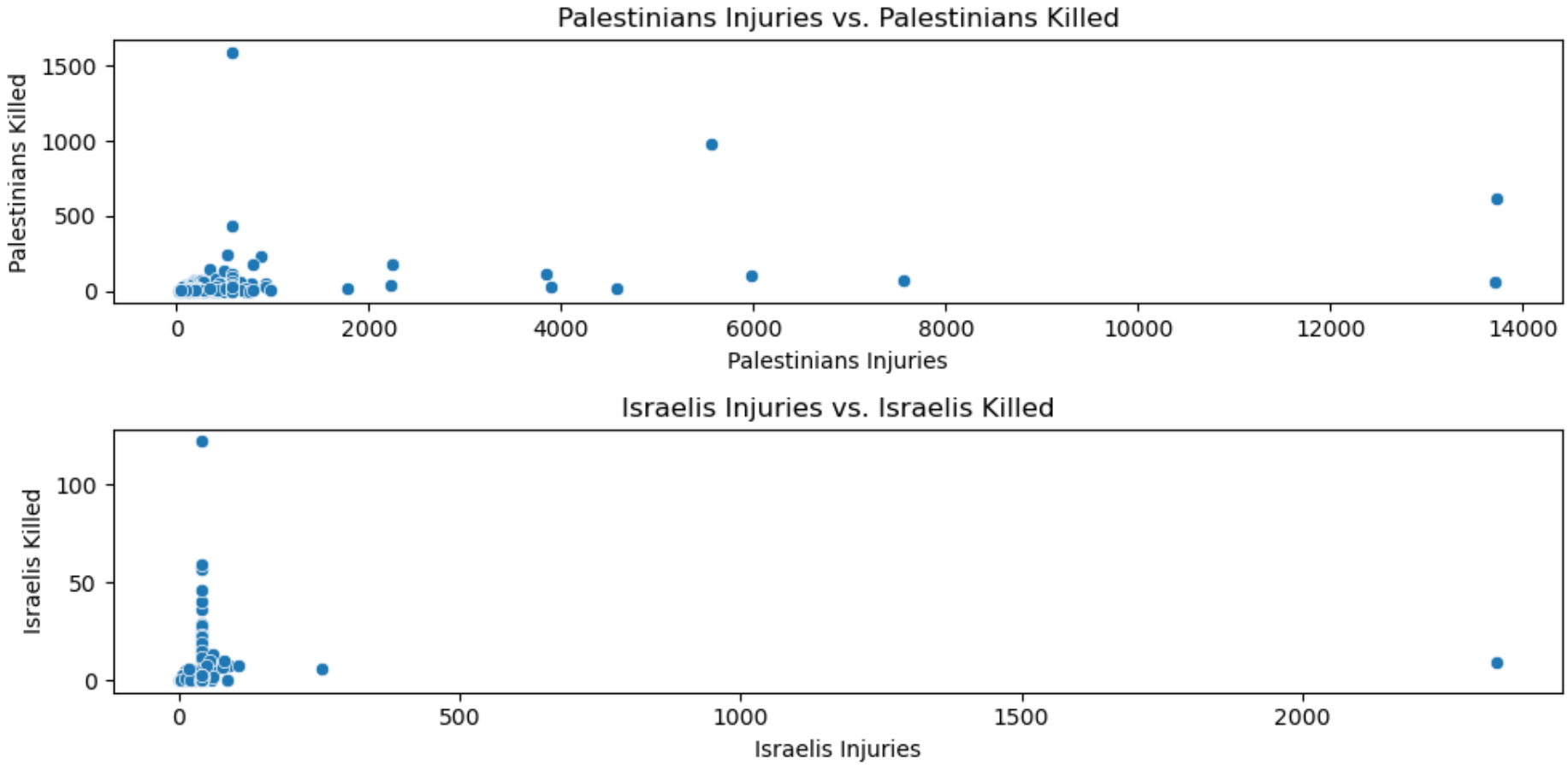
```
#Correlation Analysis:
#The correlation between different variables in the dataset (e.g., injuries and deaths for Palestinians vs. Israelis).
#Using correlation matrices or scatter plots.

correlation_matrix = df[['Palestinians Injuries', 'Palestinians Killed', 'Israelis Injuries', 'Israelis Killed']].corr()

#heatmap:

plt.figure(figsize=(10,5))
sns.heatmap(correlation_matrix, annot=True,
            cmap='coolwarm', fmt=".2f", vmin=-1, vmax=1)
```

```
plt.title('Correlation Matrix')
plt.show()
```



In [31]:
```
#scatterplot:

plt.figure(figsize=(10,5))

#Palestinians Injuries vs. Palestinians Killed
plt.subplot(2,1,1)
sns.scatterplot(data=df, x = df['Palestinians Injuries'], y = df['Palestinians Killed'])
plt.title('Palestinians Injuries vs. Palestinians Killed')

#Israelis Injuries vs. Israelis Killed
plt.subplot(2,1,2)
sns.scatterplot(data=df, x = df['Israelis Injuries'], y = df['Israelis Killed'])
```

```
plt.title('Israelis Injuries vs. Israelis Killed')

plt.tight_layout()
plt.show()
```



Linear Regression:

In [32]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [33]:
```
x = df['Palestinians Injuries']
y = df['Palestinians Killed']
```

In [34]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

x_train = x_train.to_numpy().reshape(-1, 1)
x_test = x_test.to_numpy().reshape(-1, 1)
```

In [35]:
```
model = LinearRegression()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)
```

In [36]:
```
y_pred
```

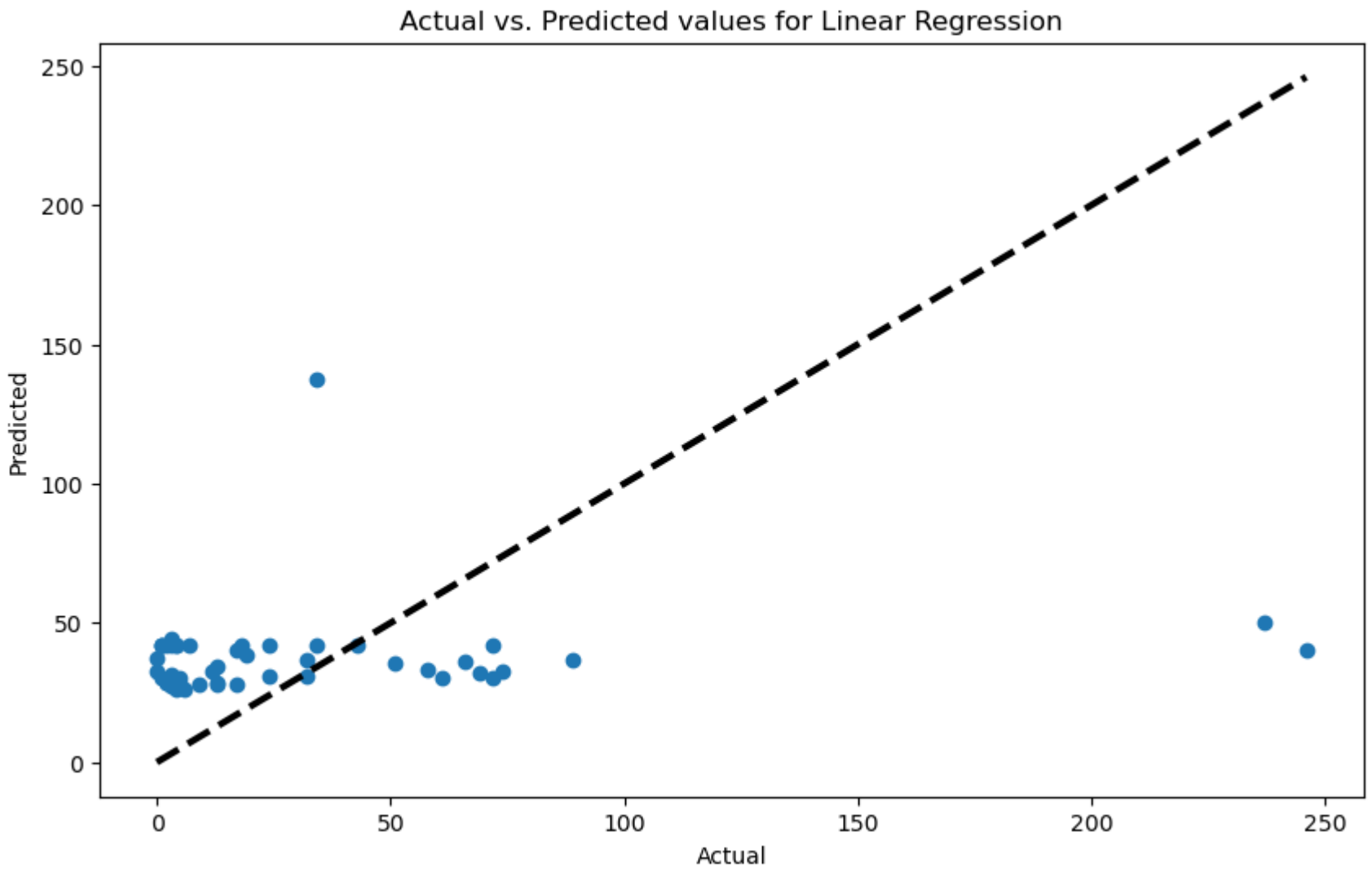
Out[36]:
```
array([ 26.15458058,  36.91913508,  30.25261474,  32.56136637,
        37.55404178,  27.42439398,  40.0070904 ,  26.21229937,
        36.54396294,  41.82523231,  41.82523231,  41.82523231,
        41.82523231, 137.37869063,  30.36805232,  41.82523231,
        41.82523231,  38.76613639,  29.99288018,  40.26682496,
        28.6076292 ,  35.36072773,  28.20359766,  30.82980265,
        41.82523231,  44.10512455,  28.5787698 ,  34.37950829,
        33.51372642,  41.82523231,  41.82523231,  41.82523231,
        27.62640975,  50.28103518,  41.82523231,  36.37080657,
        30.74322446,  28.63648859,  27.77070673,  32.4170694 ,
        41.82523231,  27.68412854,  30.4546305 ,  29.04052013,
        30.5123493 ,  28.5787698 ,  32.84996033,  31.40699055,
        32.21505363,  41.82523231])
```

In [37]:
```
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error :", mse,
     "\nIntercept :", model.intercept_,
     "\nCoefficient :", model.coef_)
```

```
Mean Squared Error : 2531.886582882586
Intercept : 25.173361138252137
Coefficient : [0.0288594]
```

In [38]:
```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=3)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

```
plt.title('Actual vs. Predicted values for Linear Regression')
plt.show()
```



Insights:

Mean Squared Error : 17321.37, on average, the squared difference between the actual and predicted values is around 17321.37.

Intercept= 23.76, the number of Palestinians injuries is zero, the predicted number of Palestinians killed is approximately 23.76.

Coefficient = 0.02, For each additional injury to Palestinians, the number of Palestinians killed increases by approximately 0.02, on average

In [39]:
```
df.head()
```

Out[39]:

	Year	Month	Palestinians Injuries	Israelis Injuries	Palestinians Killed	Israelis Killed
0	2000	12	781	39	51	8
1	2000	11	3838	39	112	22
2	2000	10	5984	39	104	10
3	2000	9	577	39	16	1
4	2001	12	304	39	67	36

In [40]:
```
data = df.copy()
```

In [41]:
```
data.head()
```

Out[41]:

	Year	Month	Palestinians Injuries	Israelis Injuries	Palestinians Killed	Israelis Killed
0	2000	12	781	39	51	8
1	2000	11	3838	39	112	22
2	2000	10	5984	39	104	10
3	2000	9	577	39	16	1
4	2001	12	304	39	67	36

Time Series Analysis:

In [42]:
```
data['Date'] = pd.to_datetime(data[['Year', 'Month']].assign(day=1))
data.set_index('Date', inplace=True)
data.drop(columns=['Year', 'Month'], inplace=True)
```

In [43]:
```
data.head()
```

Out[43]:

Date	Palestinians Injuries	Israelis Injuries	Palestinians Killed	Israelis Killed
2000-12-01	781	39	51	8
2000-11-01	3838	39	112	22
2000-10-01	5984	39	104	10
2000-09-01	577	39	16	1
2001-12-01	304	39	67	36

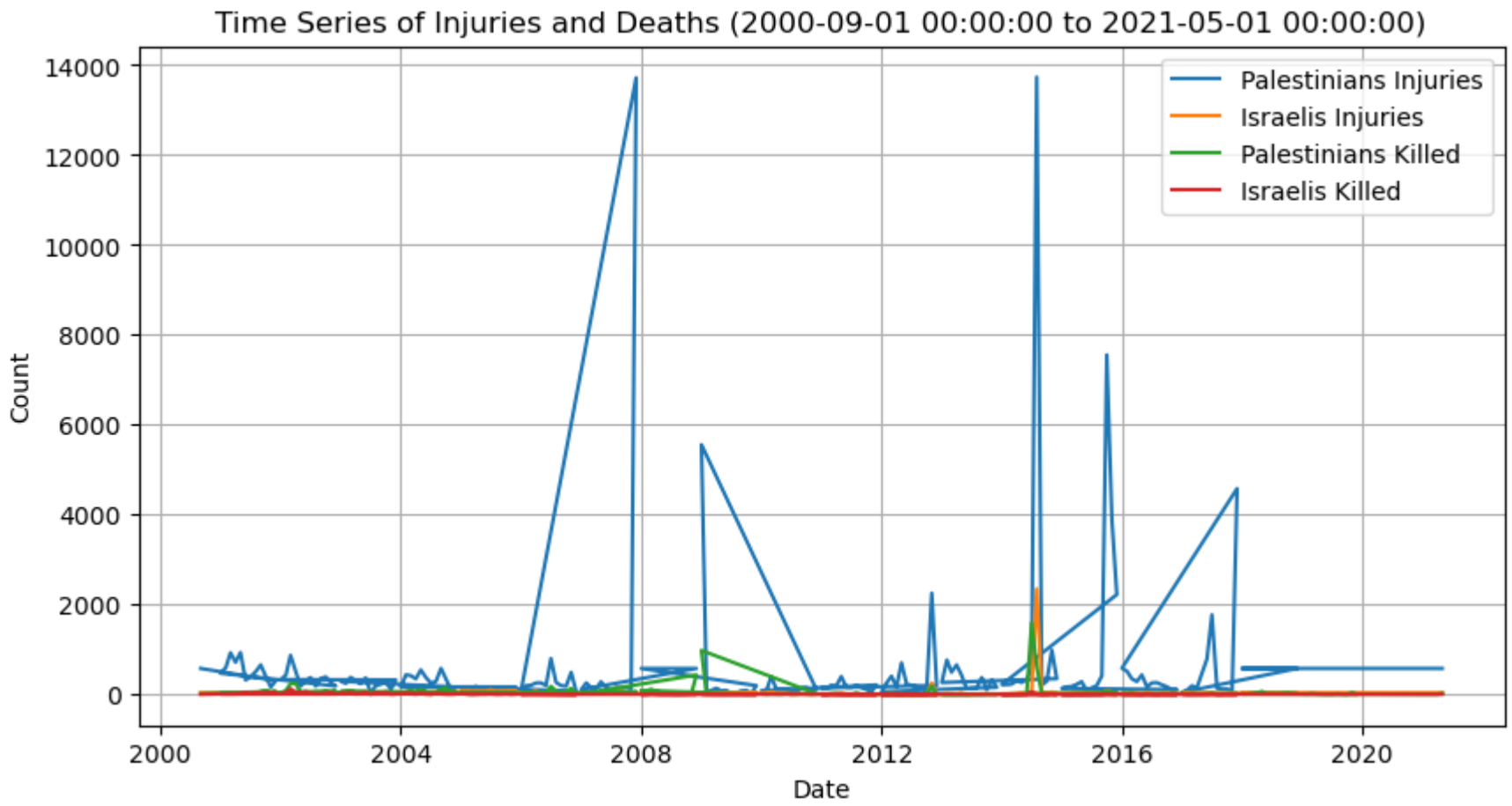
In [44]:
```
data.shape
```

Out[44]:
```
(246, 4)
```

In [45]:
```
start_date = data.index.min
end_date = data.index.max
```

In [46]:
```
start_date = data.index.min()
end_date = data.index.max()

plt.figure(figsize=(10,5))



subset_data = data.loc[start_date:end_date]

plt.plot(subset_data.index, subset_data['Palestinians Injuries'], label='Palestinians Injuries')
plt.plot(subset_data.index, subset_data['Israelis Injuries'], label='Israelis Injuries')
plt.plot(subset_data.index, subset_data['Palestinians Killed'], label='Palestinians Killed')
plt.plot(subset_data.index, subset_data['Israelis Killed'], label='Israelis Killed')


plt.title('Time Series of Injuries and Deaths ({} to {})'.format(start_date, end_date))
plt.xlabel('Date')
plt.ylabel('Count')
plt.legend()
```

```
plt.grid(True)
plt.show()
```

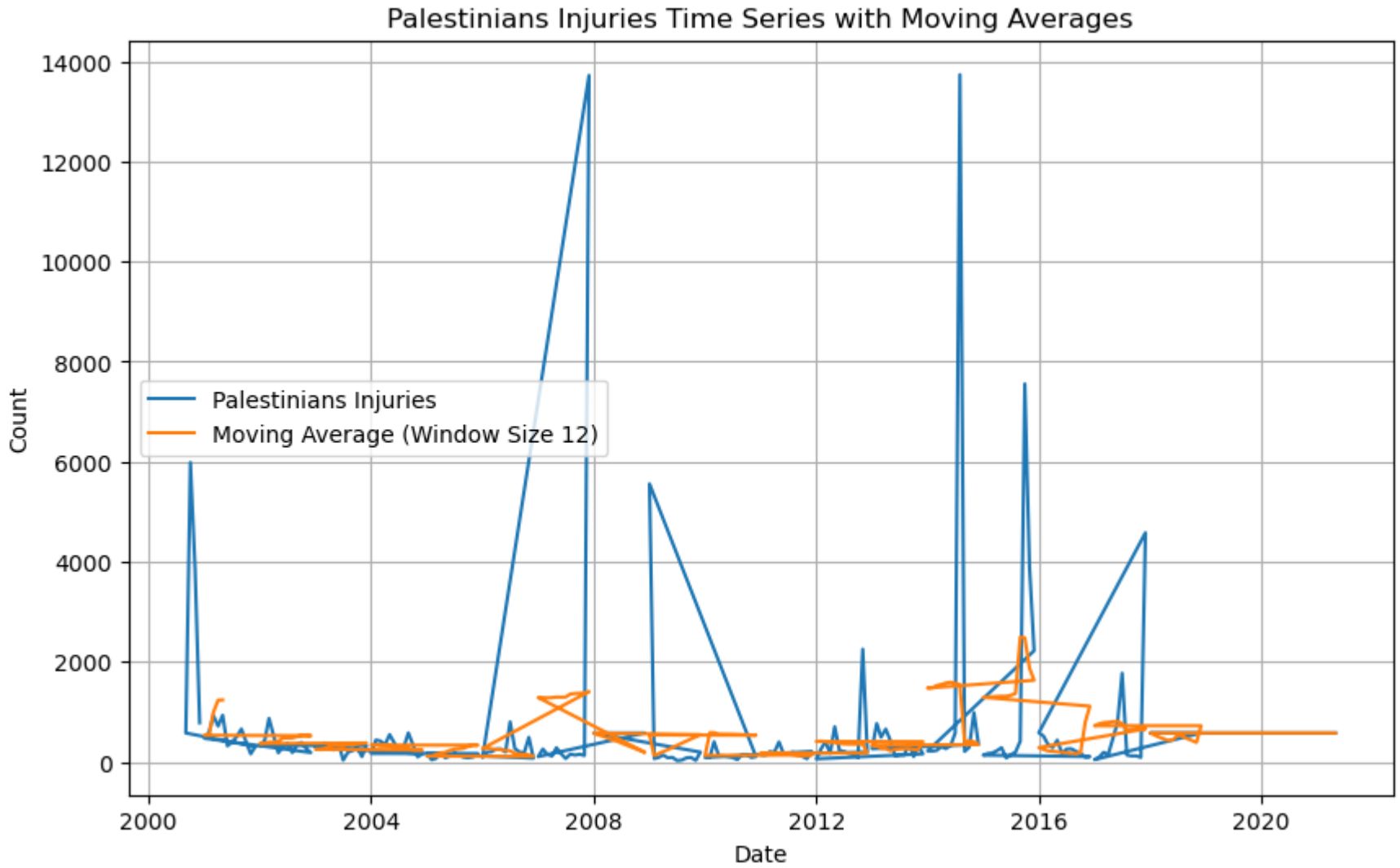


Trend Analysis:moving averages, regression analysis & seasonal decomposition of time series.

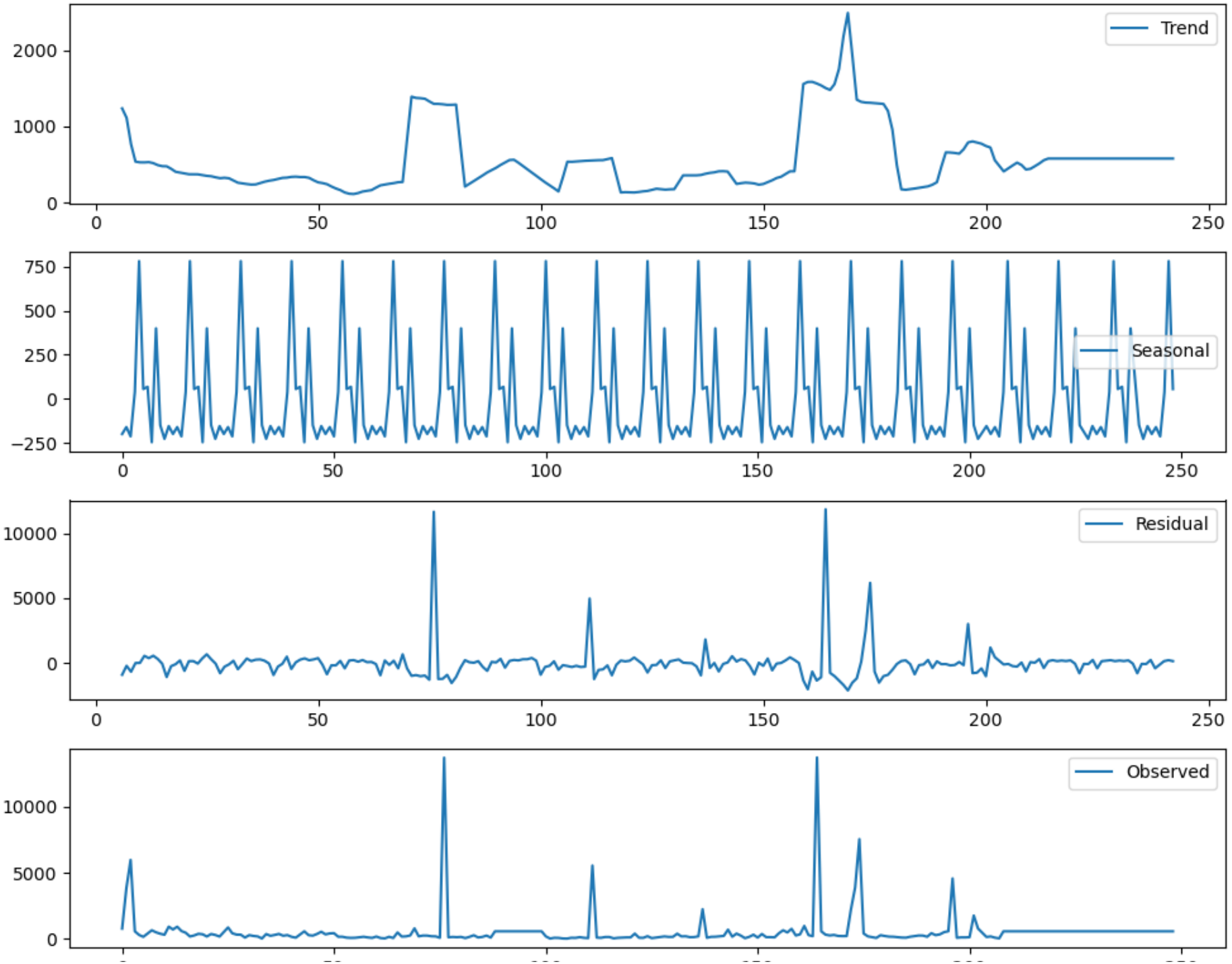
In [47]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [48]:
```
window_size = 12 # window size for Moving average.
data['Palestinians Injuries Moving Average'] = data['Palestinians Injuries'].rolling(window=window_size).mean()

result = seasonal_decompose(data['Palestinians Injuries'], model='additive', period=12)

plt.figure(figsize=(10, 6))
```

```
plt.plot(data.index, data['Palestinians Injuries'], label='Palestinians Injuries')
plt.plot(data.index, data['Palestinians Injuries Moving Average'],
         label='Moving Average (Window Size {})'.format(window_size))
plt.title('Palestinians Injuries Time Series with Moving Averages')
plt.xlabel('Date')
plt.ylabel('Count')
plt.legend()
plt.grid(True)
plt.show()
```



In [49]:
```
plt.figure(figsize=(10, 8))
plt.subplot(411)
plt.plot(df.index, result.trend, label='Trend')
plt.legend()
plt.subplot(412)
plt.plot(df.index, result.seasonal, label='Seasonal')
plt.legend()
plt.subplot(413)
plt.plot(df.index, result.resid, label='Residual')
plt.legend()
plt.subplot(414)
plt.plot(df.index, result.observed, label='Observed')
plt.legend()
plt.tight_layout()
plt.show()
```


Trend
2000
1000
0
0
50
100
150
200
250
Seasonal
750
500
250
0
−250
0
50
100
150
200
250
Residual
10000
5000
0
0
50
100
150
200
250
Observed
10000
5000
0

Fitting ARIMA model using pmdarima

In [50]:
```
from pmdarima import auto_arima
```

In [51]:
```
model = auto_arima(data['Palestinians Killed'], seasonal=True, m=12)
forecast_palestinians_killed = model.predict(n_periods=12)
```

```
C:\Users\amit9\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:836: ValueWarning: No supported index is available.
Prediction results will be given with an integer index beginning at `start`.
  return get_prediction_index(
C:\Users\amit9\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:836: FutureWarning: No supported index is availabl
e. In the next version, calling this method in a model without a supported index will result in an exception.
  return get_prediction_index(
```

In [52]:
```
forecast_palestinians_killed
```

Out[52]:
```
246    38.875004
247    40.467987
248    40.467987
249    40.467987
250    40.467987
251    40.467987
252    40.467987
253    40.467987
254    40.467987
255    40.467987
256    40.467987
257    40.467987
dtype: float64
```

Random Forest Regressor

In [53]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
WARNING:tensorflow:From C:\Users\amit9\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [54]:
```
X = data[['Palestinians Injuries', 'Israelis Injuries', 'Palestinians Killed', 'Israelis Killed']]
Y = data['Palestinians Injuries']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=32)
```

In [55]:
```
n_steps = 1
n_features = X.shape[1] # with one variable: Palestinians Killed

X_train_lstm = X_train.values.reshape(-1, n_steps, n_features)
X_test_lstm = X_test.values.reshape(-1, n_steps, n_features)

print(X_train_lstm.shape)
print(X_test_lstm.shape)
```

```
(172, 1, 4)
(74, 1, 4)
```

In [56]:
```
lstm_model = Sequential([
    LSTM(units=100, activation='relu', input_shape=(n_steps, n_features)),
    Dense(1)
])

lstm_model.compile(optimizer='adam', loss='mse')
lstm_model.fit(X_train_lstm, Y_train, epochs=50, batch_size=32)

rf_model = RandomForestRegressor()
rf_model.fit(X_train, Y_train)


y_pred_rf = rf_model.predict(X_test)
```

```
WARNING:tensorflow:From C:\Users\amit9\anaconda3\Lib\site-packages\keras\src\layers\rnn\lstm.py:148: The name tf.executing_eager
ly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\amit9\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optim
izer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/50
WARNING:tensorflow:From C:\Users\amit9\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTen
sorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

6/6 [==============================] - 1s 7ms/step - loss: 1634430.7500
Epoch 2/50
6/6 [==============================] - 0s 2ms/step - loss: 1564910.7500
Epoch 3/50
6/6 [==============================] - 0s 2ms/step - loss: 1536222.5000
Epoch 4/50
6/6 [==============================] - 0s 2ms/step - loss: 1512393.0000
Epoch 5/50
6/6 [==============================] - 0s 2ms/step - loss: 1479173.2500
Epoch 6/50
6/6 [==============================] - 0s 2ms/step - loss: 1456745.3750
Epoch 7/50
6/6 [==============================] - 0s 2ms/step - loss: 1389773.6250
Epoch 8/50
6/6 [==============================] - 0s 2ms/step - loss: 1358043.3750
Epoch 9/50
6/6 [==============================] - 0s 2ms/step - loss: 1303147.3750
Epoch 10/50
6/6 [==============================] - 0s 2ms/step - loss: 1189532.1250
Epoch 11/50
6/6 [==============================] - 0s 2ms/step - loss: 1027094.8750
Epoch 12/50
6/6 [==============================] - 0s 2ms/step - loss: 934228.0000
Epoch 13/50
6/6 [==============================] - 0s 2ms/step - loss: 681521.1250
Epoch 14/50
6/6 [==============================] - 0s 2ms/step - loss: 531047.6250
Epoch 15/50
6/6 [==============================] - 0s 2ms/step - loss: 378076.8750
Epoch 16/50
6/6 [==============================] - 0s 2ms/step - loss: 251747.3281
Epoch 17/50
6/6 [==============================] - 0s 2ms/step - loss: 207852.4219
Epoch 18/50
```

```
6/6 [==============================] - 0s 2ms/step - loss: 126968.7656
Epoch 19/50
6/6 [==============================] - 0s 2ms/step - loss: 97099.6016
Epoch 20/50
6/6 [==============================] - 0s 2ms/step - loss: 70847.2578
Epoch 21/50
6/6 [==============================] - 0s 3ms/step - loss: 56793.3672
Epoch 22/50
6/6 [==============================] - 0s 2ms/step - loss: 38079.5547
Epoch 23/50
6/6 [==============================] - 0s 2ms/step - loss: 28028.8340
Epoch 24/50
6/6 [==============================] - 0s 2ms/step - loss: 22614.5078
Epoch 25/50
6/6 [==============================] - 0s 2ms/step - loss: 15026.1670
Epoch 26/50
6/6 [==============================] - 0s 2ms/step - loss: 11419.2705
Epoch 27/50
6/6 [==============================] - 0s 2ms/step - loss: 10169.7100
Epoch 28/50
6/6 [==============================] - 0s 2ms/step - loss: 6700.4468
Epoch 29/50
6/6 [==============================] - 0s 2ms/step - loss: 4670.5073
Epoch 30/50
6/6 [==============================] - 0s 2ms/step - loss: 2948.1211
Epoch 31/50
6/6 [==============================] - 0s 2ms/step - loss: 2432.6958
Epoch 32/50
6/6 [==============================] - 0s 2ms/step - loss: 1842.5250
Epoch 33/50
6/6 [==============================] - 0s 2ms/step - loss: 1533.1860
Epoch 34/50
6/6 [==============================] - 0s 2ms/step - loss: 944.4199
Epoch 35/50
6/6 [==============================] - 0s 2ms/step - loss: 527.3589
Epoch 36/50
6/6 [==============================] - 0s 2ms/step - loss: 404.5840
Epoch 37/50
6/6 [==============================] - 0s 2ms/step - loss: 378.5113
Epoch 38/50
6/6 [==============================] - 0s 2ms/step - loss: 295.5035
Epoch 39/50
6/6 [==============================] - 0s 2ms/step - loss: 1050.2238
Epoch 40/50
```

```
6/6 [==============================] - 0s 2ms/step - loss: 901.7965
Epoch 41/50
6/6 [==============================] - 0s 2ms/step - loss: 782.2011
Epoch 42/50
6/6 [==============================] - 0s 2ms/step - loss: 673.5119
Epoch 43/50
6/6 [==============================] - 0s 2ms/step - loss: 587.1608
Epoch 44/50
6/6 [==============================] - 0s 2ms/step - loss: 560.4431
Epoch 45/50
6/6 [==============================] - 0s 2ms/step - loss: 1062.9493
Epoch 46/50
6/6 [==============================] - 0s 2ms/step - loss: 943.4404
Epoch 47/50
6/6 [==============================] - 0s 2ms/step - loss: 820.4557
Epoch 48/50
6/6 [==============================] - 0s 2ms/step - loss: 309.6065
Epoch 49/50
6/6 [==============================] - 0s 2ms/step - loss: 493.2812
Epoch 50/50
6/6 [==============================] - 0s 2ms/step - loss: 207.8605
```

In [57]:
```
mae = mean_absolute_error(Y_test, y_pred_rf)
mse = mean_squared_error(Y_test, y_pred_rf)
rmse = np.sqrt(mse)

print("Mean Absolute Error:", mae,
     "\nMean Squared Error:", mse,
     "\nRoot Mean Squared Error:", rmse)
```

```
Mean Absolute Error: 117.05432432432436
Mean Squared Error: 335256.7009567568
Root Mean Squared Error: 579.0135585258404
```

In [58]:
```
start_date_future = data.index[-1] + pd.DateOffset(days=1)
end_date_future = start_date_future + pd.DateOffset(years=6)
print("Start Date for Future forecast :", start_date_future,
      "\nEnd date for Future forecast :", end_date_future)
```

```
Start Date for Future forecast : 2021-05-02 00:00:00
End date for Future forecast : 2027-05-02 00:00:00
```

In [59]:
```
future_dates = pd.date_range(start=start_date_future, end=end_date_future, freq='Y')
X_future = pd.DataFrame(columns=X_train.columns, index=future_dates)
```

```
X_future.fillna(0, inplace=True)
future_forecast = rf_model.predict(X_future)
future_forecast
```

Out[59]:
```
array([35.78, 35.78, 35.78, 35.78, 35.78, 35.78])
```