In [1]:
```
# Importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.impute import SimpleImputer
```

Das übergeordnete Ziel des Projekts scheint darin zu bestehen, die Suizidraten zu verstehen und zu analysieren, Trends im Laufe der Zeit zu erkunden und maschinelles Lernen anzuwenden, um Suizidzahlen vorherzusagen und zu bewerten. Die Analyse kann Einblicke in die Faktoren liefern, die Suizidraten beeinflussen, und zur Sensibilisierung und Prävention im Bereich der öffentlichen Gesundheit beitragen.

(The overall goal of the project seems to be understanding and analyzing suicide rates, exploring trends over time, and applying machine learning techniques to predict and evaluate suicide numbers. The analysis may provide insights into factors influencing suicide rates and contribute to public health awareness and prevention strategies.)

Projektüberlauf: Lesen --------> is NaN -----> Duplicate --------> Korrektur ------> Ausreißer Entfernen -------> Normalvertailung ------> Traines Modul ------ >Visualisierung -----> Zeitreihenanalyse (Time Series Analysis)

plot:

Boxplot: A boxplot, also known as a box-and-whisker plot, provides a visual summary of the distribution of a dataset.

Countplot: A countplot is a type of bar plot that shows the count of occurrences of categorical variables.

Histplot:A histogram is a graphical representation of the distribution of a dataset. It divides the data into bins and counts the number of observations that fall into each bin. The result is a series of contiguous bars, where the height of each bar represents the frequency of data within that bin. Histograms are commonly used to illustrate the underlying frequency distribution of a continuous variable.

In [2]:
```
# Loading the dataset
df = pd.read_csv('master.csv')
df
```

Out[2]:

	country	year	sex	age	suicides_no	population	suicides/100k pop	country-ye
0	Albania	1987	male	15-24 years	21	312900.0	6.71	Albania19
1	Albania	1987	male	35-54 years	16	308000.0	5.19	Albania19
2	Albania	1987	female	15-24 years	14	289700.0	4.83	Albania19
3	Albania	1987	male	75+ years	1	21800.0	4.59	Albania19
4	Albania	1987	male	25-34 years	9	274300.0	3.28	Albania19
...	...	...	...	...	...	...	...	
27815	Uzbekistan	2014	female	35-54 years	107	3620833.0	2.96	Uzbekistan20
27816	Uzbekistan	2014	female	75+ years	9	348465.0	2.58	Uzbekistan20
27817	Uzbekistan	2014	male	5-14 years	60	2762158.0	2.17	Uzbekistan20
27818	Uzbekistan	2014	female	5-14 years	44	2631600.0	1.67	Uzbekistan20
27819	Uzbekistan	2014	female	55-74 years	21	1438935.0	1.46	Uzbekistan20

27820 rows × 14 columns

In [3]:
```
# Checking null values in the dataframe
print(df.isnull().sum())
```

```
country                   0
year                      0
sex                       0
age                       0
suicides_no               0
population                1
suicides/100k pop         0
country-year              0
HDI for year          19456
 gdp_for_year ($)         0
gdp_per_capita ($)        0
generation                0
Unnamed: 12           27820
Gender                27812
dtype: int64
```

In [4]:
```
# Summary of the dataset
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27820 entries, 0 to 27819
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   country             27820 non-null  object 
 1   year                27820 non-null  int64  
 2   sex                 27820 non-null  object 
 3   age                 27820 non-null  object 
 4   suicides_no         27820 non-null  int64  
 5   population          27819 non-null  float64
 6   suicides/100k pop   27820 non-null  float64
 7   country-year        27820 non-null  object 
 8   HDI for year        8364 non-null   float64
 9    gdp_for_year ($)   27820 non-null  object 
 10  gdp_per_capita ($)  27820 non-null  int64  
 11  generation          27820 non-null  object 
 12  Unnamed: 12         0 non-null      float64
 13  Gender              8 non-null      object 
dtypes: float64(4), int64(3), object(7)
memory usage: 3.0+ MB
None
```

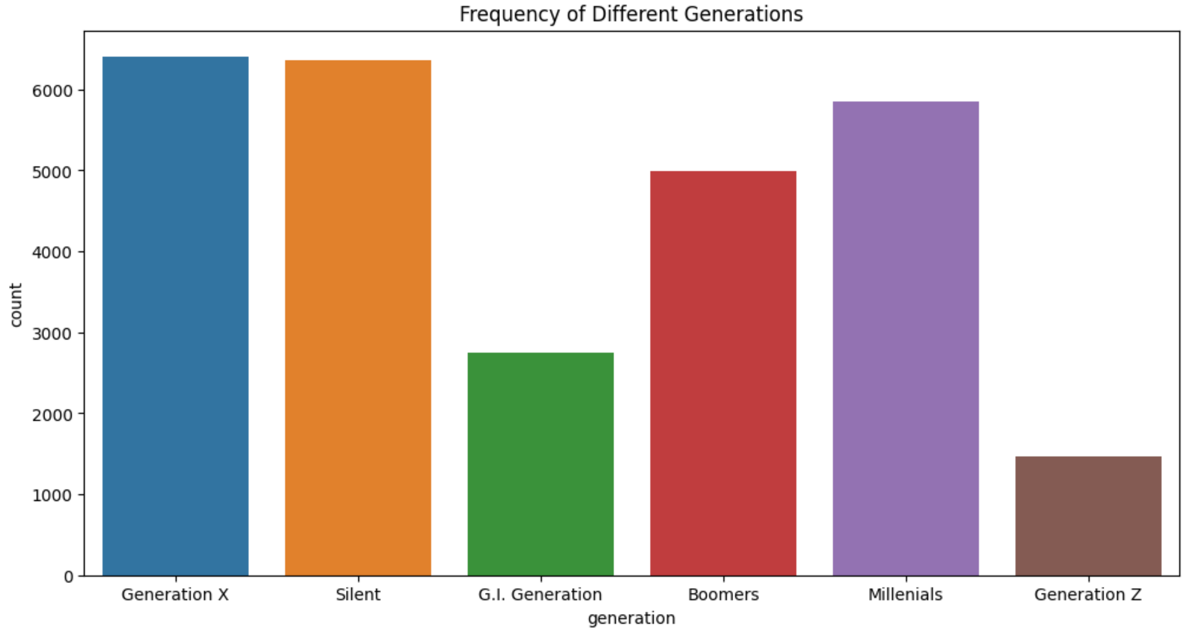
In [5]:

```
# Population of different countries
population_by_country = df.groupby('country')['population'].max().sort_value
print(population_by_country)
```

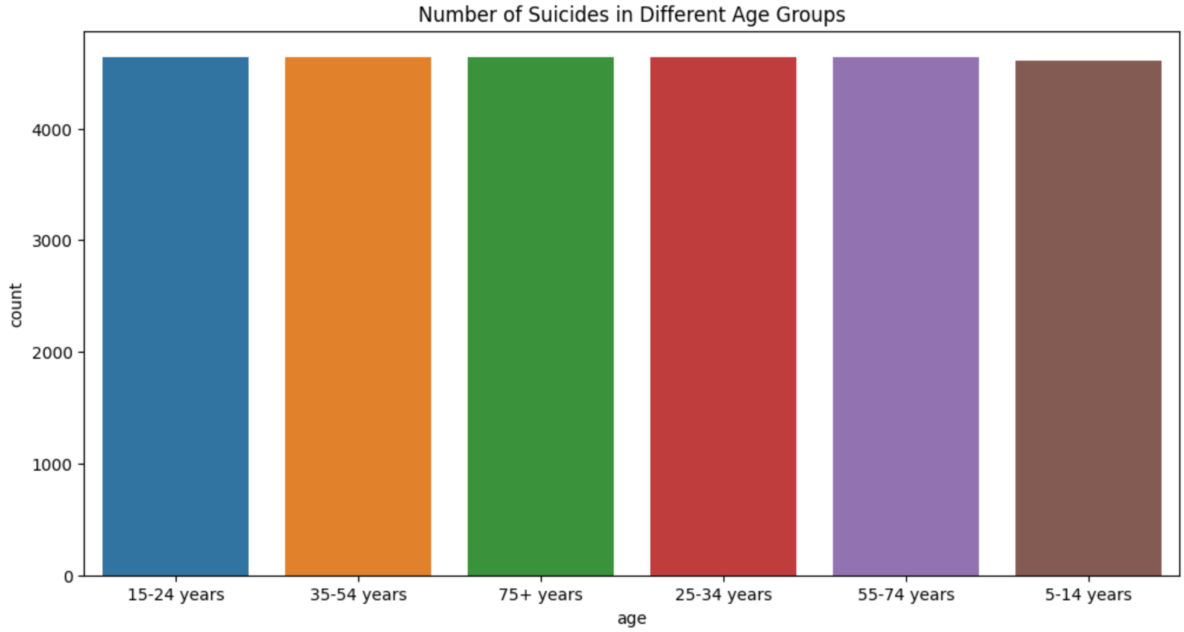
```
country
United States            43805214.0
Brazil                   28461855.0
Russian Federation       23046634.0
Japan                    18362000.0
Mexico                   15940497.0
                            ...    
Grenada                     13546.0
Kiribati                    11403.0
Dominica                     9500.0
Saint Kitts and Nevis        6000.0
San Marino                   4856.0
Name: population, Length: 101, dtype: float64
```

In [6]:

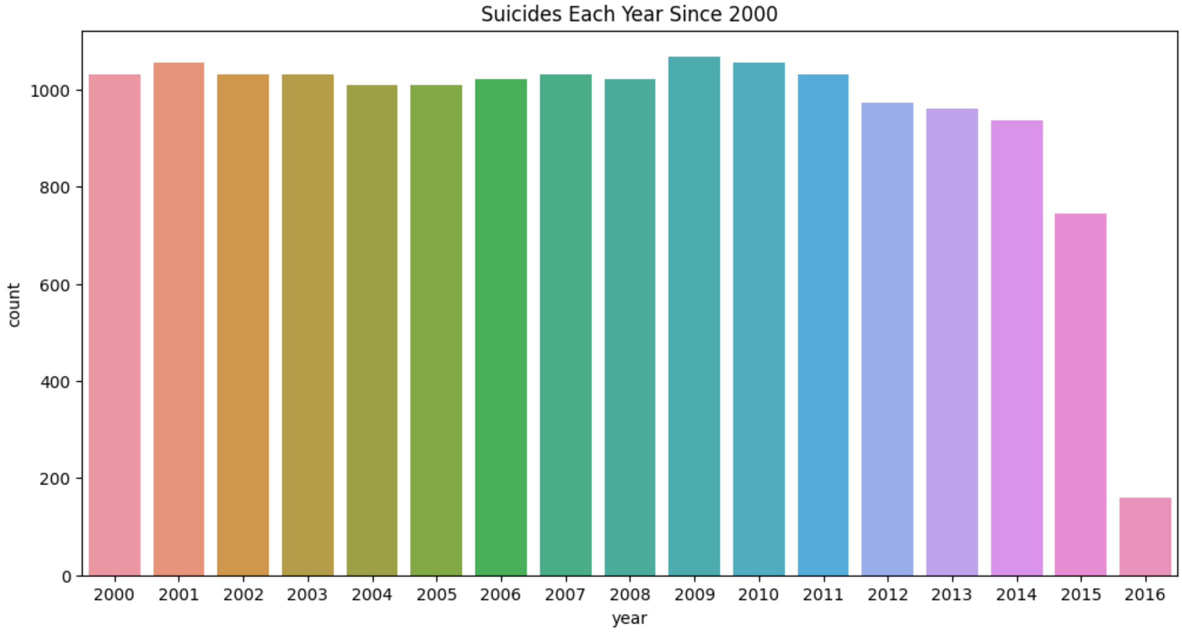
```
# Plot Frequency of different generations with countplot
plt.figure(figsize=(12, 6))
sns.countplot(x='generation', data=df)
plt.title('Frequency of Different Generations')
plt.show()
```



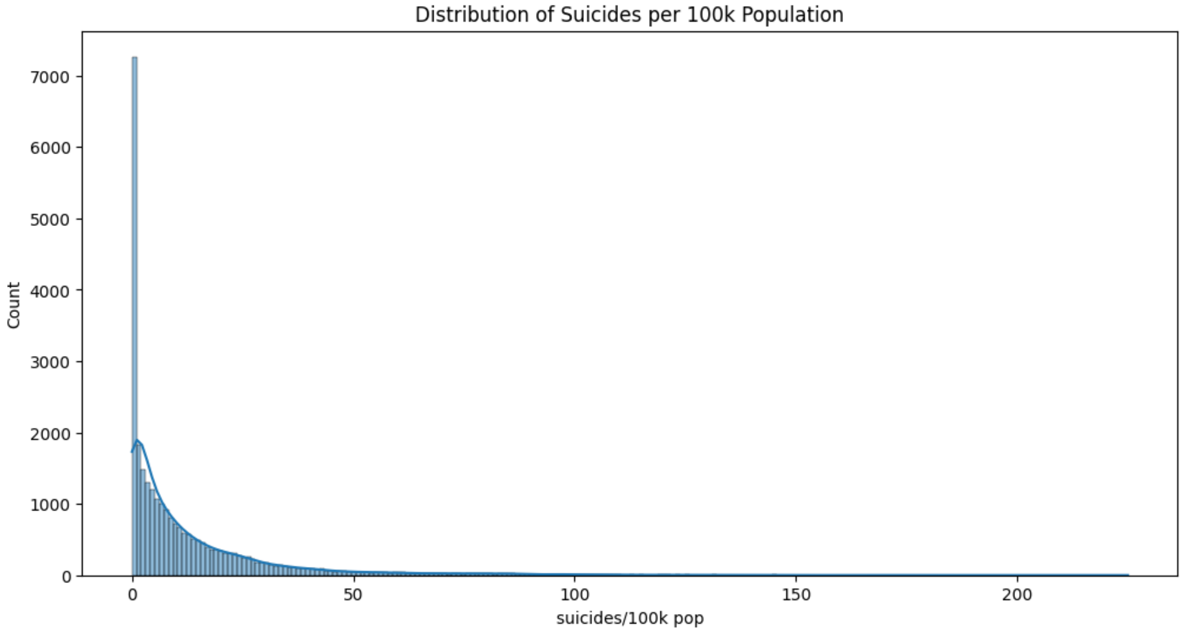
In [7]:
```
# Plot Number of suicides in different age groups
plt.figure(figsize=(12, 6))
sns.countplot(x='age', data=df)
plt.title('Number of Suicides in Different Age Groups')
plt.show()
```



In [8]:
```
# Plot Suicides each year since 2000
plt.figure(figsize=(12, 6))
df_year_2000_onwards = df[df['year'] >= 2000]
sns.countplot(x='year', data=df_year_2000_onwards)
plt.title('Suicides Each Year Since 2000')
plt.show()
```



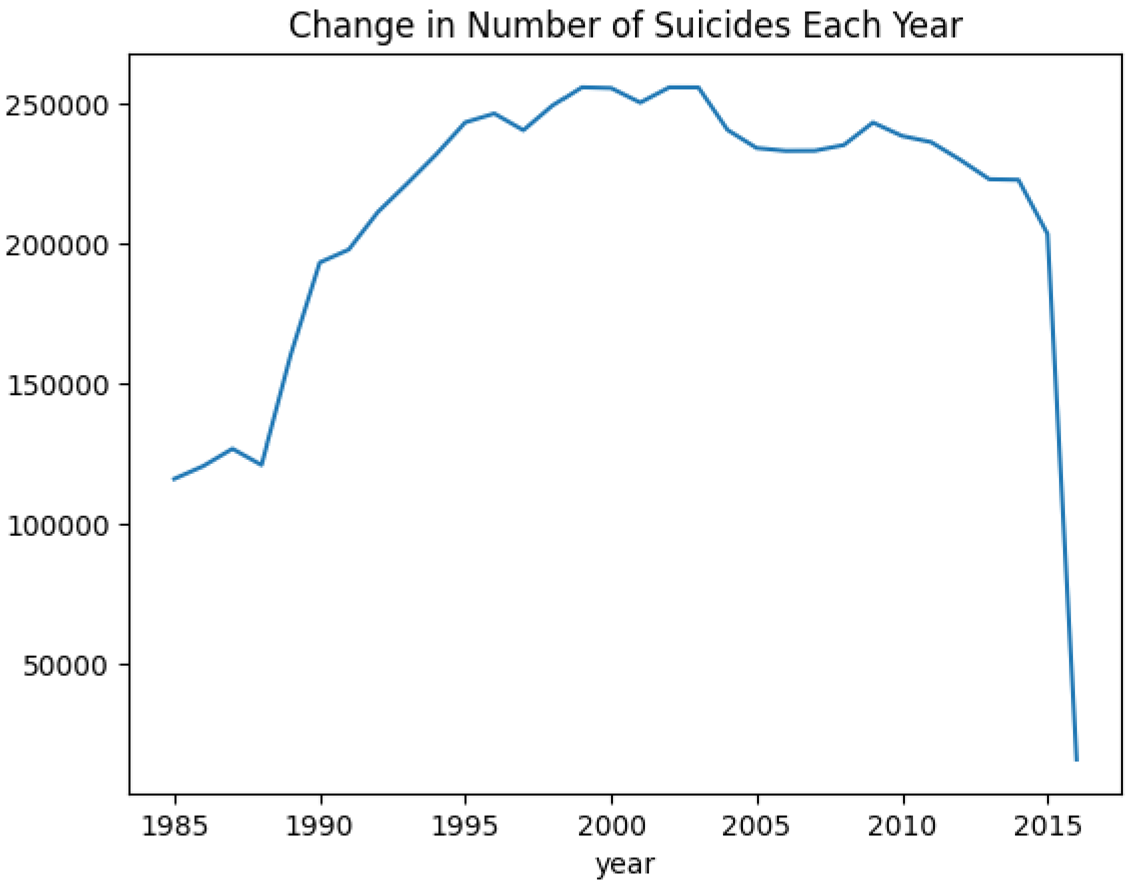
In [9]:
```
# Plot Distribution of suicides/100k pop
plt.figure(figsize=(12, 6))
sns.histplot(df['suicides/100k pop'], kde=True)
plt.title('Distribution of Suicides per 100k Population')
plt.show()
```



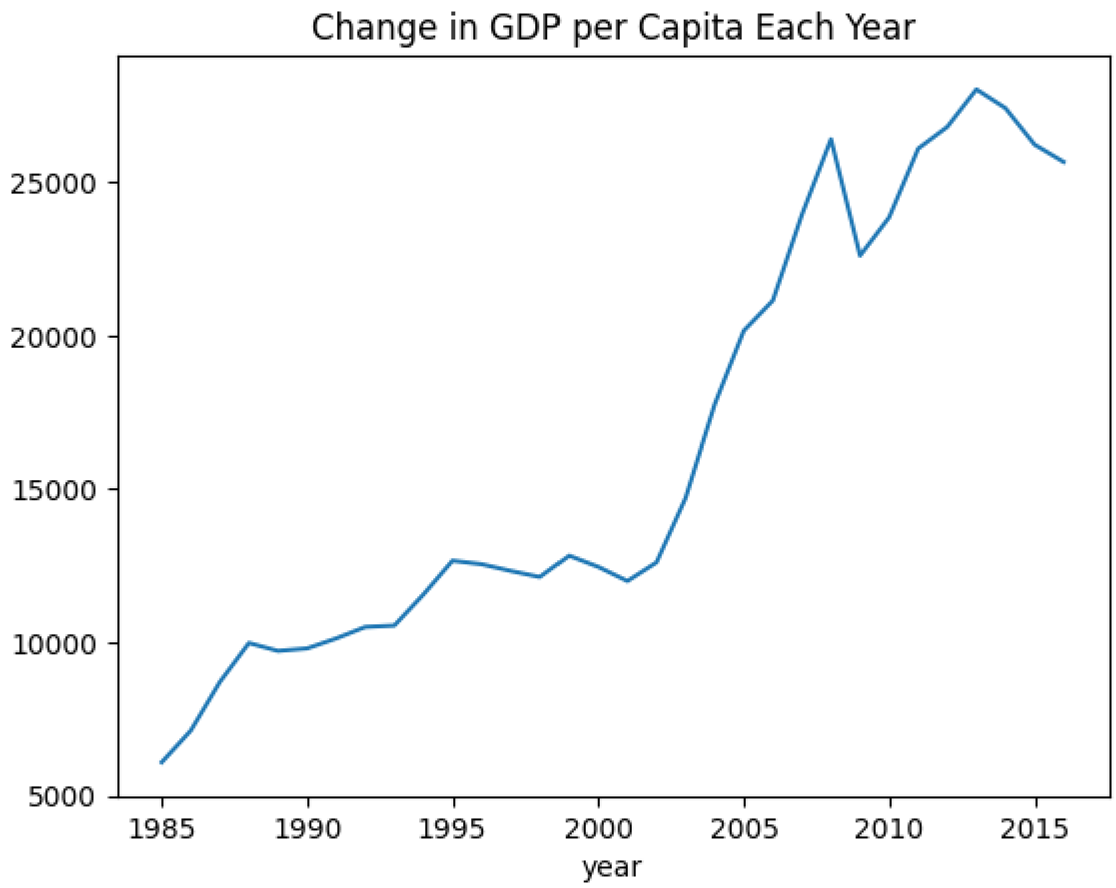
In [10]:
```
# Top 10 Countries with maximum number of suicides
top_10_countries_suicides = df.groupby('country')['suicides_no'].sum().nlarg
print(top_10_countries_suicides)
```

```
country
Russian Federation    1209742
United States         1034013
Japan                  806902
France                 329127
Ukraine                319950
Germany                291262
Republic of Korea      261730
Brazil                 226613
Poland                 139098
United Kingdom         136805
Name: suicides_no, dtype: int64
```

In [11]:
```
# Change in number of suicides each year
suicides_each_year = df.groupby('year')['suicides_no'].sum()
suicides_each_year.plot(kind='line')
plt.title('Change in Number of Suicides Each Year')
plt.show()
```



In [12]:
```
# Change in gdp_per_capita per year
gdp_per_capita_each_year = df.groupby('year')['gdp_per_capita ($)'].mean()
gdp_per_capita_each_year.plot(kind='line')
plt.title('Change in GDP per Capita Each Year')
plt.show()
```



In [13]:
```
# Top 10 countries with maximum number of suicides since 1985
top_10_countries_suicides_since_1985 = df[df['year'] >= 1985].groupby('coun
print(top_10_countries_suicides_since_1985)
```

```
country
Russian Federation    1209742
United States         1034013
Japan                  806902
France                 329127
Ukraine                319950
Germany                291262
Republic of Korea      261730
Brazil                 226613
Poland                 139098
United Kingdom         136805
Name: suicides_no, dtype: int64
```

In [14]:
```
# Top 10 countries with least number of suicides since 1985
top_10_countries_least_suicides_since_1985 = df[df['year'] >= 1985].groupby
print(top_10_countries_least_suicides_since_1985)
```

```
country
Dominica                  0
Saint Kitts and Nevis     0
San Marino                4
Antigua and Barbuda      11
Maldives                 20
Macau                    27
Oman                     33
Grenada                  38
Cabo Verde               42
Kiribati                 53
Name: suicides_no, dtype: int64
```

In [15]:
```
# Remove duplicate values
df.drop_duplicates(inplace=True)
df
```

Out[15]:

	country	year	sex	age	suicides_no	population	suicides/100k pop	country-ye
0	Albania	1987	male	15-24 years	21	312900.0	6.71	Albania19
1	Albania	1987	male	35-54 years	16	308000.0	5.19	Albania19
2	Albania	1987	female	15-24 years	14	289700.0	4.83	Albania19
3	Albania	1987	male	75+ years	1	21800.0	4.59	Albania19
4	Albania	1987	male	25-34 years	9	274300.0	3.28	Albania19
...	...	...	...	...	...	...	...	
27815	Uzbekistan	2014	female	35-54 years	107	3620833.0	2.96	Uzbekistan20
27816	Uzbekistan	2014	female	75+ years	9	348465.0	2.58	Uzbekistan20
27817	Uzbekistan	2014	male	5-14 years	60	2762158.0	2.17	Uzbekistan20
27818	Uzbekistan	2014	female	5-14 years	44	2631600.0	1.67	Uzbekistan20
27819	Uzbekistan	2014	female	55-74 years	21	1438935.0	1.46	Uzbekistan20

27820 rows × 14 columns

In [16]:
```
# Welche Variablen sind numerisch?
numerical_variables = df.select_dtypes(include=np.number).columns
print(numerical_variables)
```

```
Index(['year', 'suicides_no', 'population', 'suicides/100k pop',
       'HDI for year', 'gdp_per_capita ($)', 'Unnamed: 12'],
      dtype='object')
```

In [17]:
```
# Welche Variablen sind kategorisch?
categorical_variables = df.select_dtypes(include='object').columns
print(categorical_variables)
```

```
Index(['country', 'sex', 'age', 'country-year', ' gdp_for_year ($) ',
       'generation', 'Gender'],
      dtype='object')
```

In [18]:
```
# ÜBERPRÜFEN Sie die value_counts für jede Variable
for column in df.columns:
    print(df[column].value_counts())
```

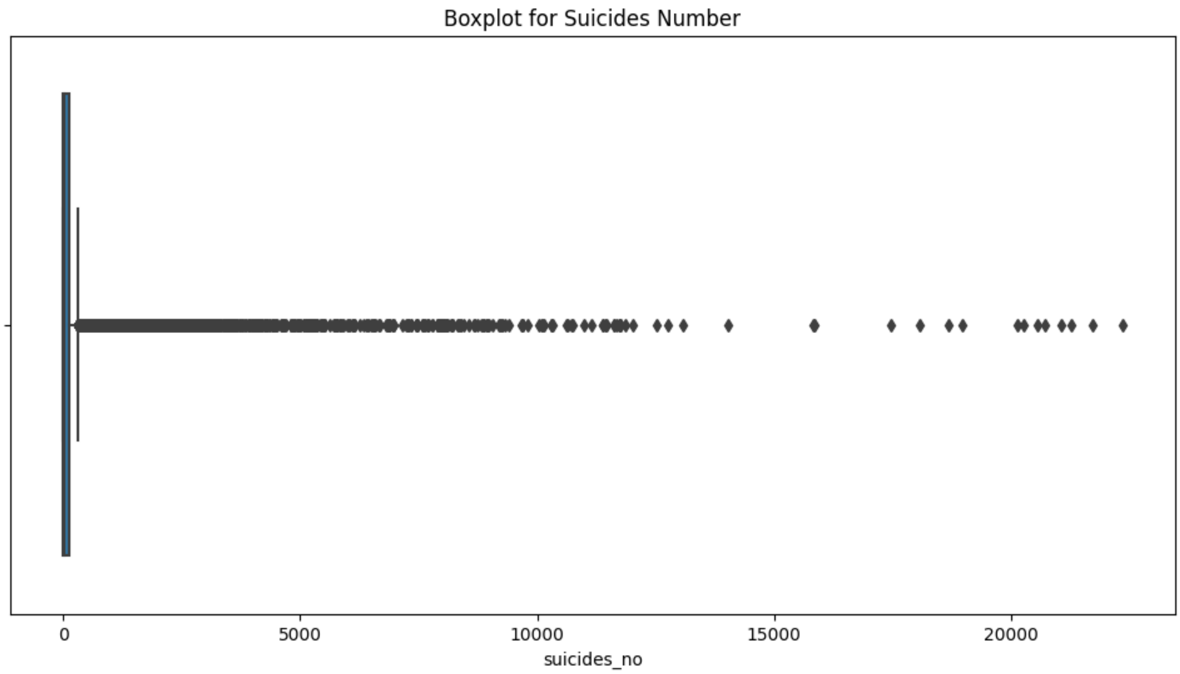
```
country
Mauritius                 382
Austria                   382
Netherlands               382
Iceland                   382
Brazil                    372
                         ...
Bosnia and Herzegovina     24
Cabo Verde                 12
Dominica                   12
Macau                      12
Mongolia                   10
Name: count, Length: 101, dtype: int64
year
2009    1068
2010    1056
2001    1056
2002    1032
2000    1032
2011    1032
2007    1032
2003    1032
2008    1020
2006    1020
2004    1008
2005    1008
1999     996
2012     972
2013     960
1998     948
1995     936
2014     936
1997     924
1996     924
1994     816
1993     780
1992     780
1990     768
1991     768
2015     744
1987     648
1989     624
1988     588
1986     576
1985     576
2016     160
Name: count, dtype: int64
sex
male      13910
female    13910
Name: count, dtype: int64
age
15-24 years    4642
35-54 years    4642
75+ years      4642
25-34 years    4642
55-74 years    4642
5-14 years     4610
Name: count, dtype: int64
suicides_no
0       4281
1       1539
2       1102
3        867
```

```
4        696
        ...
2158       1
525        1
2297       1
5241       1
2872       1
Name: count, Length: 2084, dtype: int64
population
24000.0      20
26900.0      13
22000.0      12
20700.0      12
4900.0       11
             ..
6363131.0     1
3282478.0     1
3953119.0     1
5745824.0     1
1438935.0     1
Name: count, Length: 25563, dtype: int64
suicides/100k pop
0.00     4281
0.29       72
0.32       69
0.34       55
0.37       52
         ...
46.73       1
41.47       1
61.03       1
28.25       1
26.61       1
Name: count, Length: 5298, dtype: int64
country-year
Albania1987     12
Poland1993      12
Panama2009      12
Panama2010      12
Panama2011      12
                ..
Austria2016     10
Croatia2016     10
Hungary2016     10
Armenia2016     10
Mongolia2016    10
Name: count, Length: 2321, dtype: int64
HDI for year
0.772    84
0.713    84
0.888    84
0.830    72
0.761    72
         ..
0.696    12
0.894    12
0.893    12
0.770    12
0.675    12
Name: count, Length: 305, dtype: int64
 gdp_for_year ($)
2,156,624,900      12
96,045,645,026     12
27,116,635,600     12
```

```
29,440,287,600     12
34,686,224,300     12
                   ..
390,799,991,147    10
51,338,524,831     10
125,816,640,421    10
10,546,135,160     10
11,183,458,131     10
Name: count, Length: 2321, dtype: int64
gdp_per_capita ($)
2303     36
1299     36
4104     36
1698     24
939      24
         ..
62484    10
46976    10
15742    10
12905    10
48108    10
Name: count, Length: 2233, dtype: int64
generation
Generation X       6408
Silent             6364
Millenials         5844
Boomers            4990
G.I. Generation    2744
Generation Z       1470
Name: count, dtype: int64
Series([], Name: count, dtype: int64)
Gender
MALE      4
FEMALE    3
Female    1
Name: count, dtype: int64
```

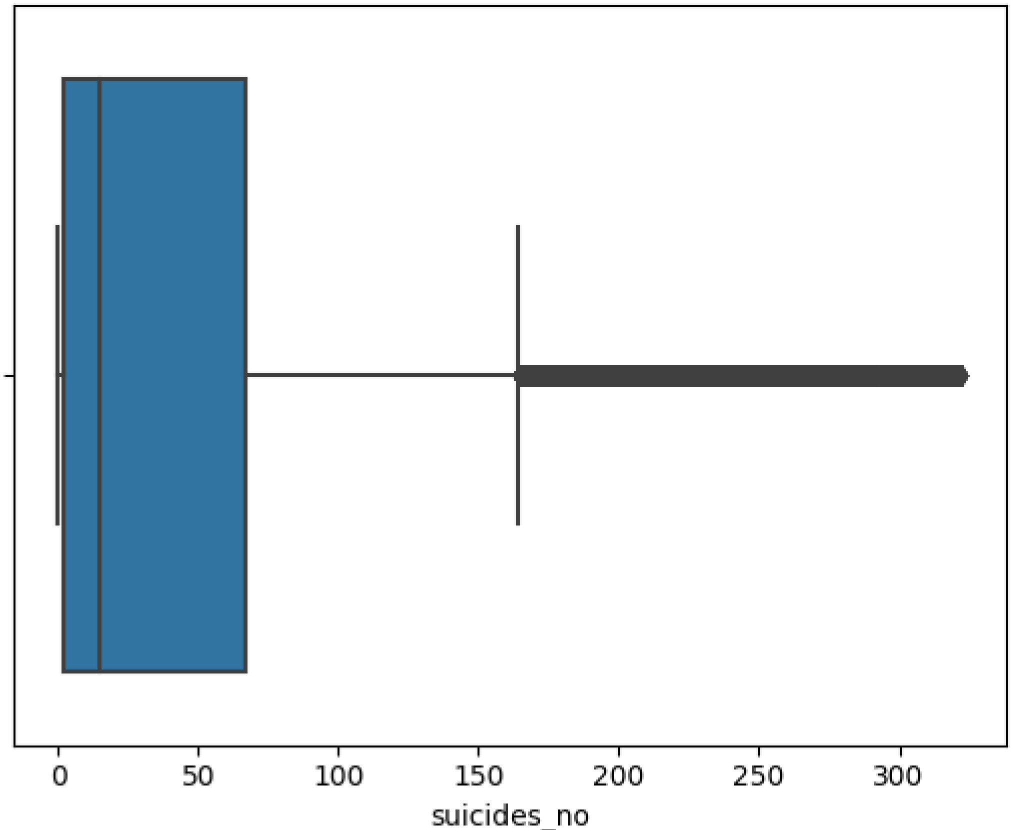
In [19]:

```
# Erkennung und Entfernung von Ausreißern (Outliers)
# Assuming 'suicides_no' as a target variable for outlier detection
plt.figure(figsize=(12, 6))
sns.boxplot(x=df['suicides_no'])
plt.title('Boxplot for Suicides Number')
plt.show()
```



In [37]:
```
iqr = np.percentile(df['suicides_no'],75) - np.percentile(df['suicides_no'],
upper_limit = np.percentile(df['suicides_no'],75) + 1.5*iqr
lower_limit =np.percentile(df['suicides_no'],25) - 1.5*iqr
```

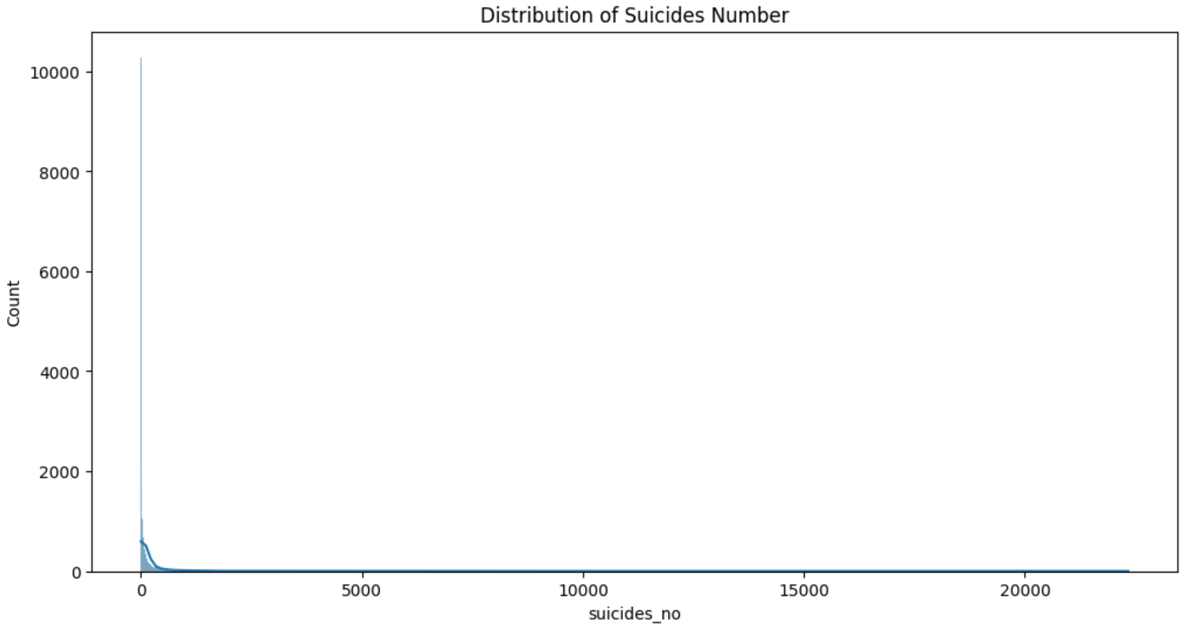
In [38]:
```
df_suicides_no = df[(df['suicides_no'] > lower_limit) & (df['suicides_no'] <
sns.boxplot(x=df_suicides_no['suicides_no'])
plt.show()
```



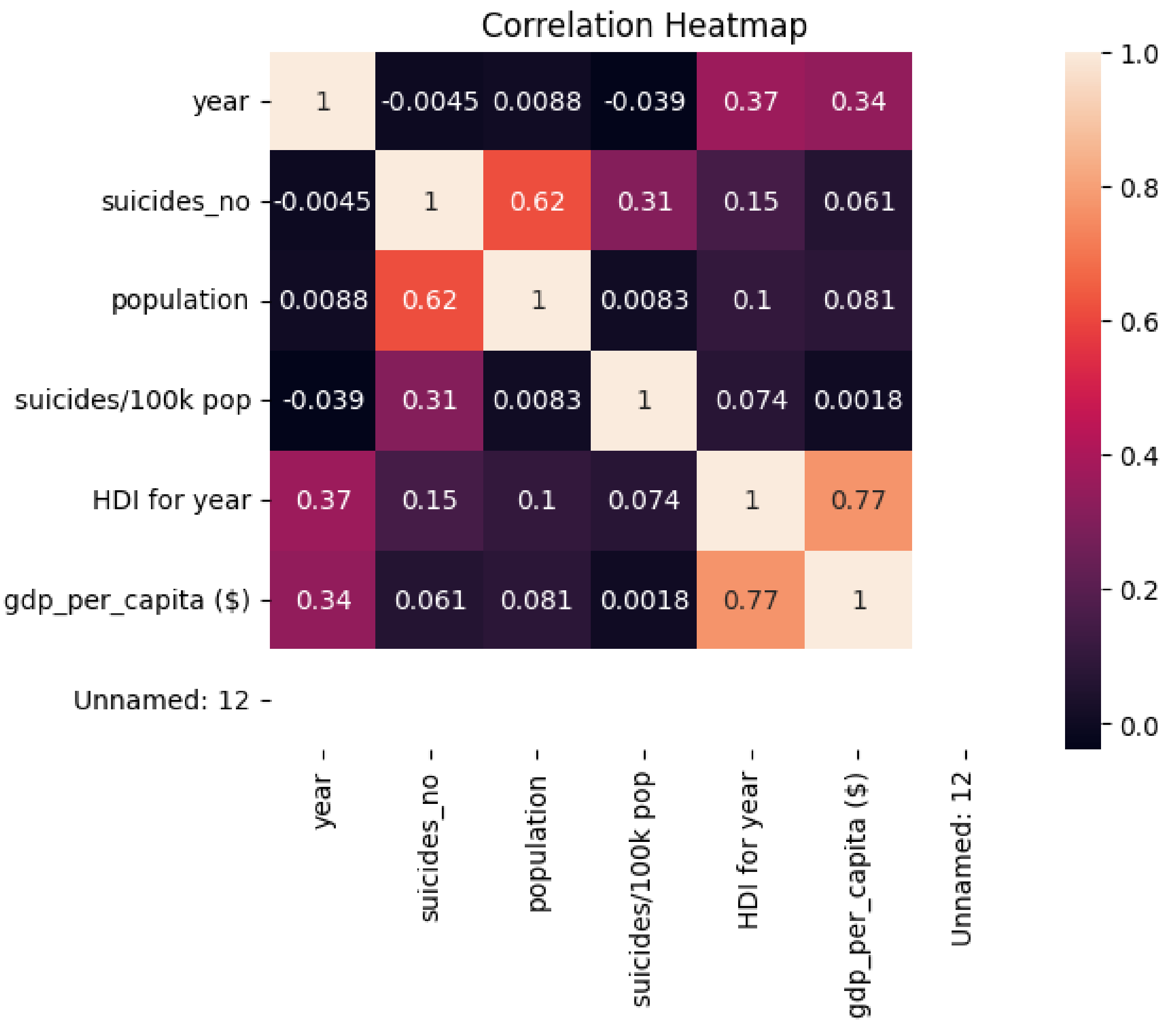
In [20]:
```
# Wer ist der ältere Kontakt?
oldest_contact = df.loc[df['age'] == '75+ years', 'country'].iloc[0]
print(f'The oldest contact is in {oldest_contact}')
```

```
The oldest contact is in Albania
```

In [21]:
```
# Überprüfen Sie die Normalitätsverteilung
plt.figure(figsize=(12, 6))
sns.histplot(df['suicides_no'], kde=True)
plt.title('Distribution of Suicides Number')
plt.show()
```



In [22]:
```
# Überprüfen Sie die Korrelation mit der Bibliothek Seaborn und mit der Bibl
# Selecting only numerical columns for correlation calculation
numerical_columns = df.select_dtypes(include=np.number)
correlation_matrix = numerical_columns.corr()

sns.heatmap(correlation_matrix, annot=True)
plt.title('Correlation Heatmap')
plt.show()
```



In [23]:
```
# Splitting des Modells
X = df[['gdp_per_capita ($)', 'population']]
y = df['suicides_no']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran
```

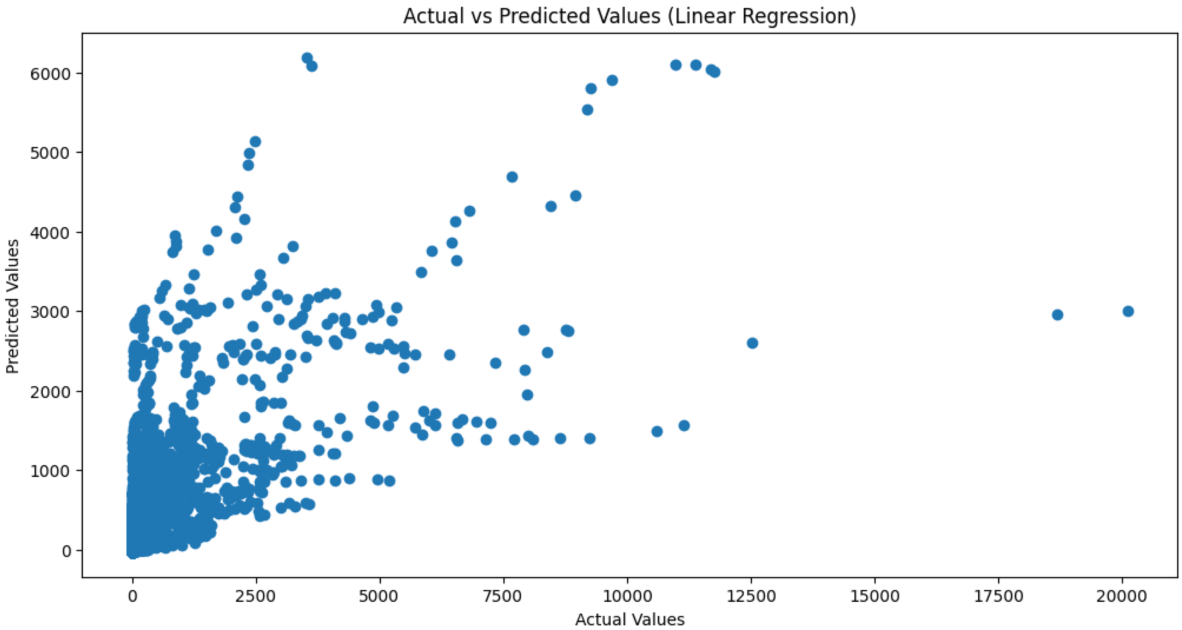
In [24]:
```
# Handling missing values in X_train
imputer_X = SimpleImputer(strategy='mean')  # You can use other strategies 
X_train_imputed = imputer_X.fit_transform(X_train)

# Handling missing values in y_train (if any)
imputer_y = SimpleImputer(strategy='mean')
y_train_imputed = imputer_y.fit_transform(y_train.values.reshape(-1, 1)).rav

# Linear Regression
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train_imputed, y_train_imputed)
```

Out[24]:
```
▾ LinearRegression
LinearRegression()
```

In [25]:
```
# Führen Sie das Modell mit sklearn aus: lineare Regression
y_pred_linear_reg = linear_reg_model.predict(X_test)
```

```
/opt/homebrew/lib/python3.11/site-packages/sklearn/base.py:457: UserWarnin
g: X has feature names, but LinearRegression was fitted without feature nam
es
  warnings.warn(
```

In [26]:
```
# Plot the actual value und the predicted value
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred_linear_reg)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values (Linear Regression)')
plt.show()
```



In [27]:
```
# Determinate the mean square errors and r square for the model
mse_linear_reg = mean_squared_error(y_test, y_pred_linear_reg)
r2_linear_reg = r2_score(y_test, y_pred_linear_reg)
print(f'Mean Squared Error (Linear Regression): {mse_linear_reg}')
print(f'R-Squared (Linear Regression): {r2_linear_reg}')
```

```
Mean Squared Error (Linear Regression): 413572.2202425951
R-Squared (Linear Regression): 0.4231562499638689
```

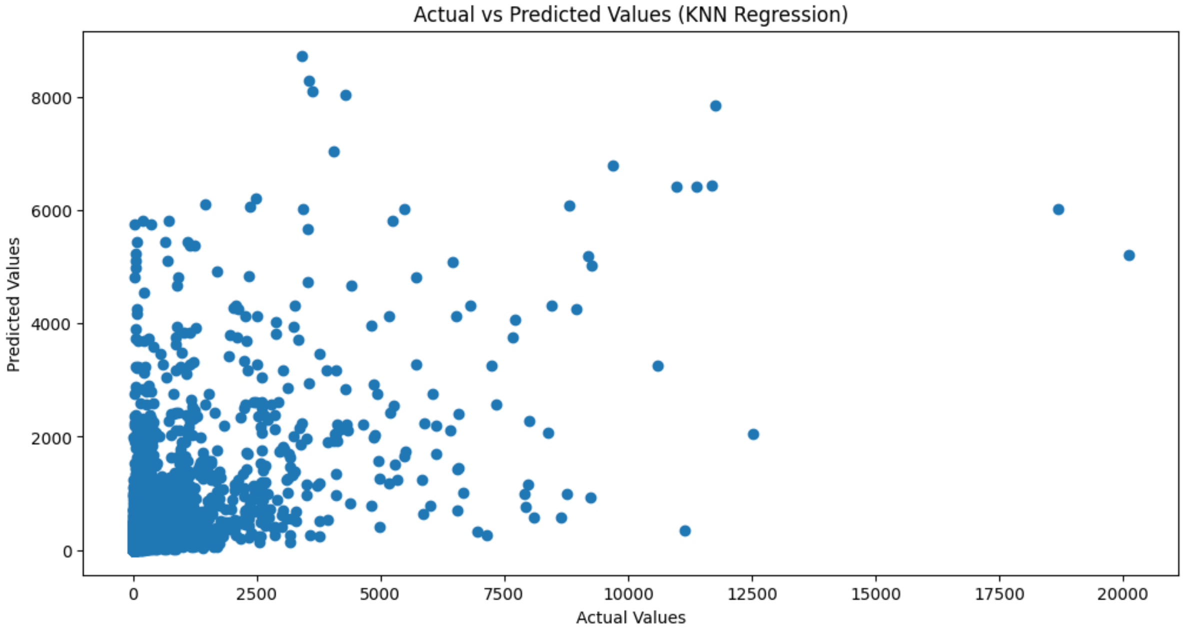
In [28]:
```
# Handling missing values in X_train
imputer_X = SimpleImputer(strategy='mean')  # You can use other strategies
X_train_imputed = imputer_X.fit_transform(X_train)

# Handling missing values in y_train (if any)
imputer_y = SimpleImputer(strategy='mean')
y_train_imputed = imputer_y.fit_transform(y_train.values.reshape(-1, 1)).ra

# KNN Regression
knn_reg_model = KNeighborsRegressor(n_neighbors=5)
knn_reg_model.fit(X_train_imputed, y_train_imputed)
```

Out[28]:
```
▾ KNeighborsRegressor
KNeighborsRegressor()
```

In [29]:
```
# Predictions
y_pred_knn_reg = knn_reg_model.predict(X_test)
```

```
/opt/homebrew/lib/python3.11/site-packages/sklearn/base.py:457: UserWarnin
g: X has feature names, but KNeighborsRegressor was fitted without feature
names
  warnings.warn(
```

In [30]:
```
# Plot the actual value und the predicted value
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred_knn_reg)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values (KNN Regression)')
plt.show()
```



In [31]:
```
# Determinate the mean square errors and r square for the model
mse_knn_reg = mean_squared_error(y_test, y_pred_knn_reg)
r2_knn_reg = r2_score(y_test, y_pred_knn_reg)
print(f'Mean Squared Error (KNN Regression): {mse_knn_reg}')
print(f'R-Squared (KNN Regression): {r2_knn_reg}')
```

```
Mean Squared Error (KNN Regression): 472312.0804744788
R-Squared (KNN Regression): 0.3412268562708344
```

In [32]:
```
# überprüfen Sie die Stationarität für jede Variable mit ADF und KPSS
# Assuming 'suicides_no' as a target variable for stationarity check
from statsmodels.tsa.stattools import adfuller, kpss
```

Time Series Analysis:

Check the stationarity of the 'suicides_no' variable using ADF and KPSS tests. Train a linear regression model on time-series data using the 'year' as a feature and plot actual vs. predicted values.

In [33]:
```
adf_result = adfuller(df['suicides_no'])
kpss_result = kpss(df['suicides_no'])

print(f'ADF Statistic: {adf_result[0]}, p-value: {adf_result[1]}, Critical V
print(f'KPSS Statistic: {kpss_result[0]}, p-value: {kpss_result[1]}, Critica
```

```
ADF Statistic: -6.6612626905704495, p-value: 4.840541023497506e-09, Critica
l Values: {'1%': -3.4305855109867127, '5%': -2.86164408918305, '10%': -2.56
68254035491365}
KPSS Statistic: 1.2040718149045126, p-value: 0.01, Critical Values: {'10%':
0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
```

```
/var/folders/n_/ykzc4f9d07d1vpwy6td0g3fc0000gn/T/ipykernel_37105/131991390
5.py:2: InterpolationWarning: The test statistic is outside of the range of
p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_result = kpss(df['suicides_no'])
```

In [34]:

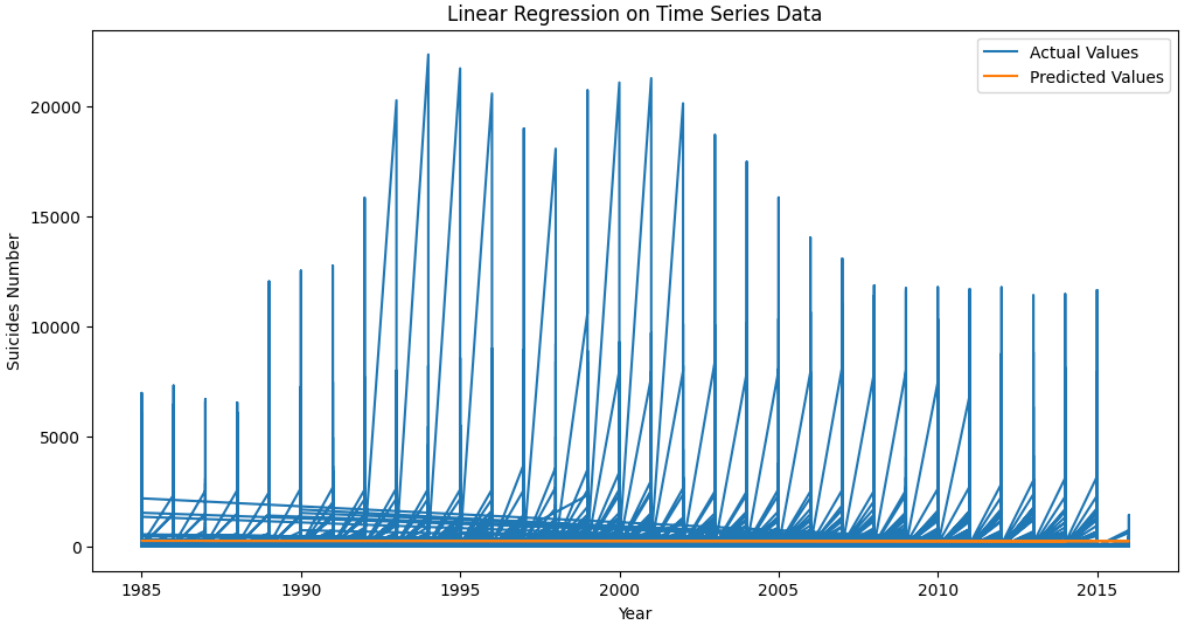
```
# Führen Sie das lineare Regressionsmodell aus (beispielhaft)
# You can use linear regression on time-series data by considering time as a
# Assuming 'suicides_no' as a target variable
time_feature = df['year'].values.reshape(-1, 1)
linear_reg_model_time = LinearRegression()
linear_reg_model_time.fit(time_feature, df['suicides_no'])
```

Out[34]:

```
▾ LinearRegression
LinearRegression()
```

In [35]:

```
# Predictions
y_pred_time = linear_reg_model_time.predict(time_feature)

# Plot the actual value und the predicted value
plt.figure(figsize=(12, 6))
plt.plot(df['year'], df['suicides_no'], label='Actual Values')
plt.plot(df['year'], y_pred_time, label='Predicted Values')
plt.xlabel('Year')
plt.ylabel('Suicides Number')
plt.title('Linear Regression on Time Series Data')
plt.legend()
plt.show()
```



In [ ]: