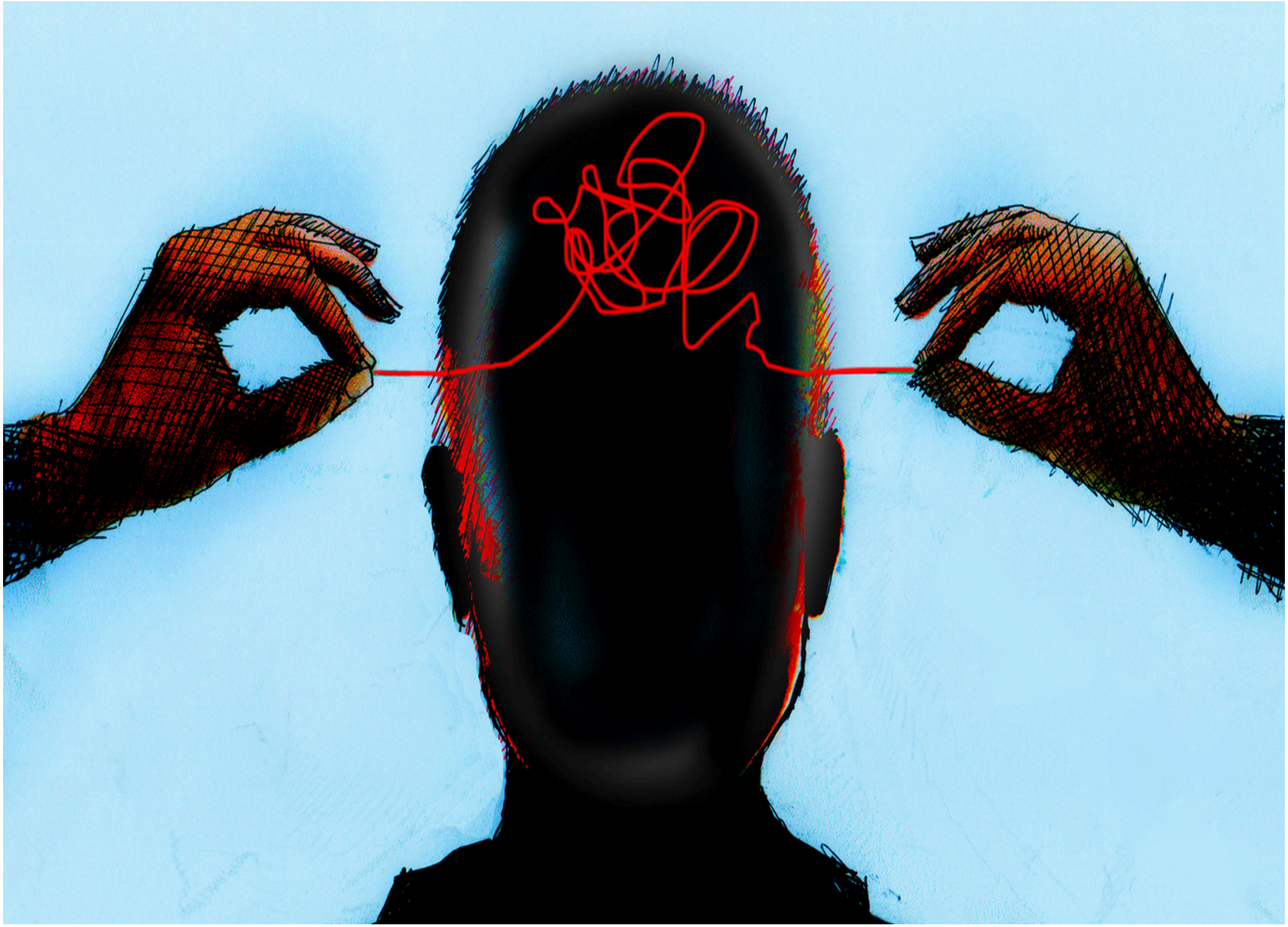
The project is from university Buckingham from their Business Analyst department, it was undertaking for practices purpose and development of my skill set

The task is focus on Python and R scripting for data analysis. I opted to used Python to enhance my skills and delve deeper into data analysis to the best of my ability.

The project comprised two main sections. The first section, labeled as Section A, centered on analyzing on suicide rates. Specifically, the objective was to identify vulnerable age groups. To achieve this,I examined the suicide rates across various countrie, with the year ranging from 1985 to 2016, this dataset was gather from WHO and named as WHO_Suicide_Data.csv.

Section B

This is to demostrate data acquiotion (Scraping, cleaning and exploration).



In [10]:
```
# import the libraries

import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
```

```
%matplotlib inline
import missingno as msno
```

Section A

Importaing the data set

In [13]:
```
who=pd.read_csv('WHO_Suicide_Data_3.csv')
```

In [14]:
```
#checking the head of the data
who.head()
```

Out[14]:

	country	year	sex	age	suicides_no	population	gdp_for_year ($)	Unnamed: 7	Unnamed: 8
0	Albania	1987	male	15-24 years	21	312900	2,156,624,900	NaN	NaN
1	Albania	1987	male	35-54 years	16	308000	2,156,624,900	NaN	NaN
2	Albania	1987	female	15-24 years	14	289700	2,156,624,900	NaN	NaN
3	Albania	1987	male	75+ years	1	21800	2,156,624,900	NaN	NaN
4	Albania	1987	male	25-34 years	9	274300	2,156,624,900	NaN	NaN

In [15]:
```
who.isnull().any()
```

Out[15]:
```
country               False
year                  False
sex                   False
age                   False
suicides_no            True
population            False
 gdp_for_year ($)     False
Unnamed: 7             True
Unnamed: 8             True
dtype: bool
```

In [16]:
```
# Drop the 'Unnamed: 8' column

who.drop(columns=['Unnamed: 7','Unnamed: 8'], inplace=True)
```

In [17]:
```
who.head()
```

Out[17]:

	country	year	sex	age	suicides_no	population	gdp_for_year ($)
0	Albania	1987	male	15-24 years	21	312900	2,156,624,900
1	Albania	1987	male	35-54 years	16	308000	2,156,624,900
2	Albania	1987	female	15-24 years	14	289700	2,156,624,900
3	Albania	1987	male	75+ years	1	21800	2,156,624,900
4	Albania	1987	male	25-34 years	9	274300	2,156,624,900

Checking for types

In [18]:
```
who.dtypes
```

Out[18]:
```
country                object
year                    int64
sex                    object
age                    object
suicides_no            object
population              int64
 gdp_for_year ($)      object
dtype: object
```

Checking the colum of the data frame

In [22]:
```
print(who.columns)
```

```
Index(['country', 'year', 'sex', 'age', 'suicides_no', 'population',
       ' gdp_for_year ($) '],
      dtype='object')
```

Converting the the data types

In [58]:
```
# Clean and convert 'gdp_for_year ($)' column to numeric
def clean_and_convert(value):
    cleaned_value = ''.join(c for c in str(value) if c.isnumeric() or c == '.')
    return float(cleaned_value)
```

```
who[' gdp_for_year '] = who[' gdp_for_year '].apply(clean_and_convert)

# Set display option to avoid scientific notation
pd.options.display.float_format = '{:.2f}'.format

# Print the first few rows to verify the changes
print(who.head())
```

```
   country  year     sex          age  suicides_no  population  \
0  Albania  1987    male  15-24 years        21.00      312900
1  Albania  1987    male  35-54 years        16.00      308000
2  Albania  1987  female  15-24 years        14.00      289700
3  Albania  1987    male    75+ years         1.00       21800
4  Albania  1987    male  25-34 years         9.00      274300

    gdp_for_year   suicides/100k   generation
0   2156624900.00           6.71  Millennials
1   2156624900.00           5.19  Millennials
2   2156624900.00           4.83  Millennials
3   2156624900.00           4.59  Millennials
4   2156624900.00           3.28  Millennials
```

Checking the suicide_no

In [24]:

```
non_numeric_mask = pd.to_numeric(who['suicides_no'], errors='coerce').isna()
problematic_values = who.loc[non_numeric_mask, 'suicides_no'].unique()

print(problematic_values)
```

```
[nan 'Null' 'Unknown']
```

Converting the suicides_no column and gdp_for_year from Object

In [59]:

```
# Convert non-numeric values to NaN
who['suicides_no'] = pd.to_numeric(who['suicides_no'], errors='coerce')


# Convert 'gdp_for_year' column to float
who[' gdp_for_year '] = who[' gdp_for_year '].astype(float)

# Print data types to confirm the changes
print(who.dtypes)
```

```
country             object
year                 int64
sex                 object
age                 object
suicides_no        float64
population           int64
 gdp_for_year      float64
suicides/100k      float64
generation          object
dtype: object
```

Confirming the dtypes

In [26]:
```python
who.dtypes
```

Out[26]:
```
country                 object
year                     int64
sex                     object
age                     object
suicides_no            float64
population               int64
 gdp_for_year ($)       object
dtype: object
```

In [27]:
```python
non_numeric_mask = pd.to_numeric(who['suicides_no'], errors='coerce').isna()
remaining_non_numeric_values = who.loc[non_numeric_mask, 'suicides_no'].unique()
print(remaining_non_numeric_values)
```

```
[nan]
```

In [28]:
```python
unique_suicides_values = who['suicides_no'].unique()
print(unique_suicides_values)
```

```
[   21.    16.    14. ... 11634.  4359.  2872.]
```

In [ ]:
```python

```

In [29]:
```python
who['suicides_no'] = who['suicides_no'].replace(['Null', 'Unknown'], np.nan)
```

In [30]:
```python
who.head()
```

Out[30]:

	country	year	sex	age	suicides_no	population	gdp_for_year ($)
0	Albania	1987	male	15-24 years	21.0	312900	2,156,624,900
1	Albania	1987	male	35-54 years	16.0	308000	2,156,624,900
2	Albania	1987	female	15-24 years	14.0	289700	2,156,624,900
3	Albania	1987	male	75+ years	1.0	21800	2,156,624,900
4	Albania	1987	male	25-34 years	9.0	274300	2,156,624,900

Replacing the dollar sign from the head

In [31]:

```
# Replace the dollar sign in the column name
who.columns = who.columns.str.replace('gdp_for_year \(\$\)','gdp_for_year')
```

```
C:\Users\Autoke Pro\AppData\Local\Temp\ipykernel_2532\1785478471.py:2: FutureWarning: The default value of regex will c
hange from True to False in a future version.
  who.columns = who.columns.str.replace('gdp_for_year \(\$\)','gdp_for_year')
```

In [32]:

```
who.head()
```

Out[32]:

	country	year	sex	age	suicides_no	population	gdp_for_year
0	Albania	1987	male	15-24 years	21.0	312900	2,156,624,900
1	Albania	1987	male	35-54 years	16.0	308000	2,156,624,900
2	Albania	1987	female	15-24 years	14.0	289700	2,156,624,900
3	Albania	1987	male	75+ years	1.0	21800	2,156,624,900
4	Albania	1987	male	25-34 years	9.0	274300	2,156,624,900

Dropping any duplicate in the data set

In [33]:

```
# Drop duplicate rows and update the DataFrame
who = who.drop_duplicates()
```

Checking for Duplicate

In [34]:

```
# Check for duplicate rows
duplicate_rows = who[who.duplicated()]

num_duplicate_rows = who.duplicated().sum()
print(f"Number of duplicate rows: {num_duplicate_rows}")
```

```
Number of duplicate rows: 0
```

Checking for missing value

In [35]:

```
who.isnull().sum()
```

Out[35]:

```
country            0
year               0
sex                0
age                0
suicides_no     4281
population         0
 gdp_for_year      0
dtype: int64
```

Used another library to visualise the missing data section

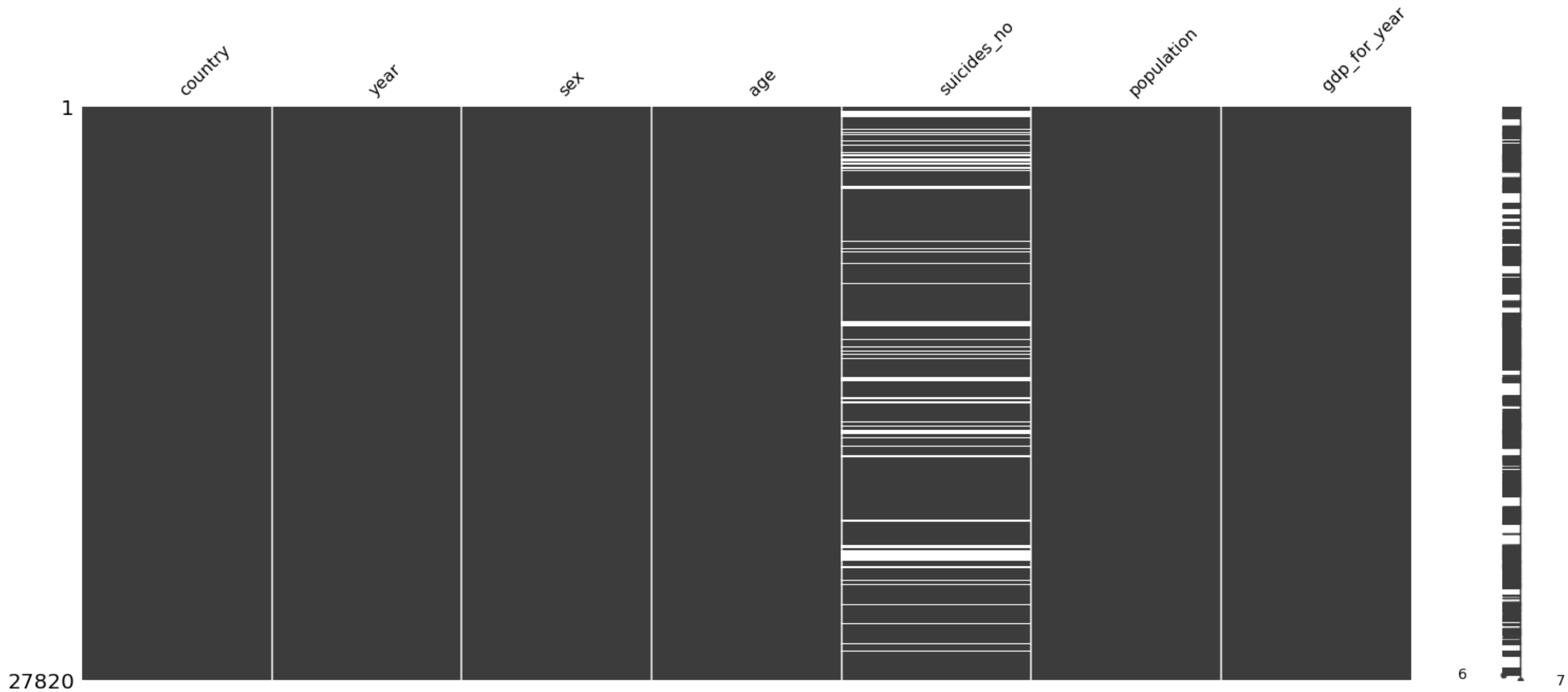
The grey and white area is the section where value are missing in the data set

In [36]:

```
msno.matrix (who)
```

Out[36]:

```
<AxesSubplot:>
```



Finding the mean of misssing value and going to use it to fill the NaN value. Note the missing value are in series, so fill forward or backward would not apply.

In [37]:
```
mean_missing_value_percent = who.isnull().mean()*100
print (mean_missing_value_percent)
```

```
country           0.00000
year              0.00000
sex               0.00000
age               0.00000
suicides_no      15.38821
population        0.00000
 gdp_for_year     0.00000
dtype: float64
```

Filling NaN with the Calculated mean of the suicides_no column

In [38]:
```
# Replace NaN values in 'suicides_no' column with mean value
mean_missing_value_percent = 15.32
```

```
who['suicides_no'] = who['suicides_no'].fillna(mean_missing_value_percent)


#who.interpolate()
new_who=who
```

```
C:\Users\Autoke Pro\AppData\Local\Temp\ipykernel_2532\732802329.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  who['suicides_no'] = who['suicides_no'].fillna(mean_missing_value_percent)
```

In [39]:

```
new_who.head(12)
```

Out[39]:

	country	year	sex	age	suicides_no	population	gdp_for_year
0	Albania	1987	male	15-24 years	21.00	312900	2,156,624,900
1	Albania	1987	male	35-54 years	16.00	308000	2,156,624,900
2	Albania	1987	female	15-24 years	14.00	289700	2,156,624,900
3	Albania	1987	male	75+ years	1.00	21800	2,156,624,900
4	Albania	1987	male	25-34 years	9.00	274300	2,156,624,900
5	Albania	1987	female	75+ years	1.00	35600	2,156,624,900
6	Albania	1987	female	35-54 years	6.00	278800	2,156,624,900
7	Albania	1987	female	25-34 years	4.00	257200	2,156,624,900
8	Albania	1987	male	55-74 years	1.00	137500	2,156,624,900
9	Albania	1987	female	5-14 years	15.32	311000	2,156,624,900
10	Albania	1987	female	55-74 years	15.32	144600	2,156,624,900
11	Albania	1987	male	5-14 years	15.32	338200	2,156,624,900

Comfirming the suicides_no column

In [40]:

```
nan_mask = who['suicides_no'].isna()
nan_values = who.loc[nan_mask, 'suicides_no']
```

```
print(nan_values)
```

```
Series([], Name: suicides_no, dtype: float64)
```

Confirming for missing value

In [41]:
```
new_who.isnull().sum()
```

Out[41]:
```
country         0
year            0
sex             0
age             0
suicides_no     0
population      0
 gdp_for_year   0
dtype: int64
```

Adding a new column and it will be called 'suicides/100k'

In [42]:
```
new_who['suicides/100k']=new_who['suicides_no']/ (new_who['population'] /100000)
```

```
C:\Users\Autoke Pro\AppData\Local\Temp\ipykernel_2532\3575281647.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  new_who['suicides/100k']=new_who['suicides_no']/ (new_who['population'] /100000)
```

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

Out[43]:

	country	year	sex	age	suicides_no	population	gdp_for_year	suicides/100k
0	Albania	1987	male	15-24 years	21.0	312900	2,156,624,900	6.711409
1	Albania	1987	male	35-54 years	16.0	308000	2,156,624,900	5.194805
2	Albania	1987	female	15-24 years	14.0	289700	2,156,624,900	4.832585
3	Albania	1987	male	75+ years	1.0	21800	2,156,624,900	4.587156
4	Albania	1987	male	25-34 years	9.0	274300	2,156,624,900	3.281079

Creating another column called 'generation'

In [60]:
```
def assign_generation(year):
    if 1883 <= year <= 1900:
        return 'Lost Generation'
    elif 1901 <= year <= 1927:
        return 'G.I. Generation'
    elif 1928 <= year <= 1945:
        return 'Silent'
    elif 1946 <= year <= 1964:
        return 'Boomers'
    elif 1965 <= year <= 1980:
        return 'Generation X'
    elif 1981 <= year <= 1996:
        return 'Millennials'
    elif 1997 <= year <= 2012:
        return 'Generation Z'
    elif 2013 <= year <= 2025:
        return 'Generation A'
    else:
        return 'Unknown'

new_who['generation'] = new_who['year'].apply(assign_generation)
```

In [45]:
```
new_who.head()
```

Out[45]:

	country	year	sex	age	suicides_no	population	gdp_for_year	suicides/100k	generation
0	Albania	1987	male	15-24 years	21.0	312900	2,156,624,900	6.711409	Millennials
1	Albania	1987	male	35-54 years	16.0	308000	2,156,624,900	5.194805	Millennials
2	Albania	1987	female	15-24 years	14.0	289700	2,156,624,900	4.832585	Millennials
3	Albania	1987	male	75+ years	1.0	21800	2,156,624,900	4.587156	Millennials
4	Albania	1987	male	25-34 years	9.0	274300	2,156,624,900	3.281079	Millennials

Crteating another column called 'gdp_per_capita'

In [61]:
```
# Add the 'gdp_per_capita' column
new_who['gdp_per_capita'] = new_who[' gdp_for_year '] / (new_who['population'] / 100000)
```

```
# Perform calculations on the 'gdp_per_capita' column
new_who['gdp_per_capita'] = new_who['gdp_per_capita'] / new_who['population'] * 100000
```

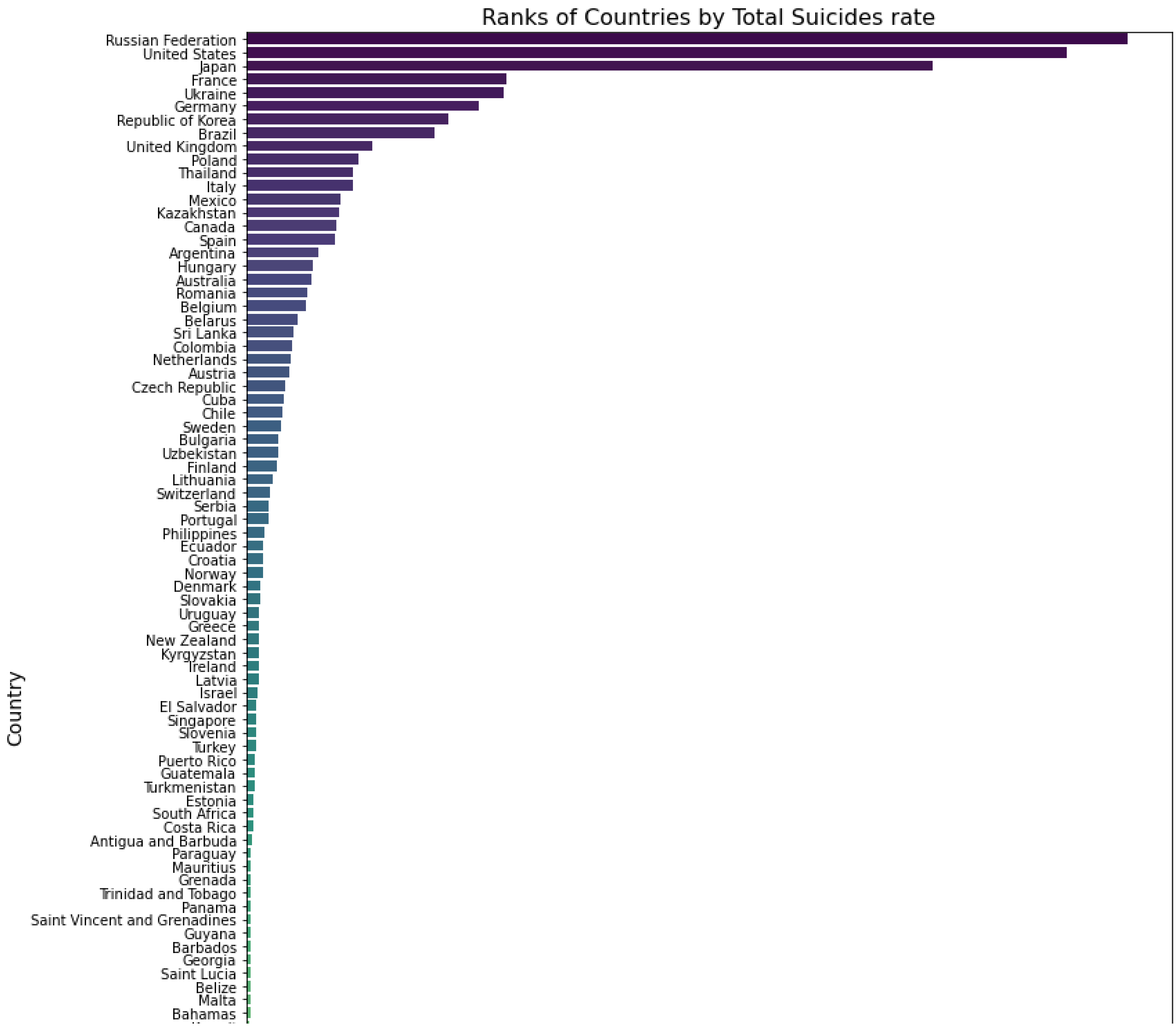
In [47]:
```
new_who.head()
```

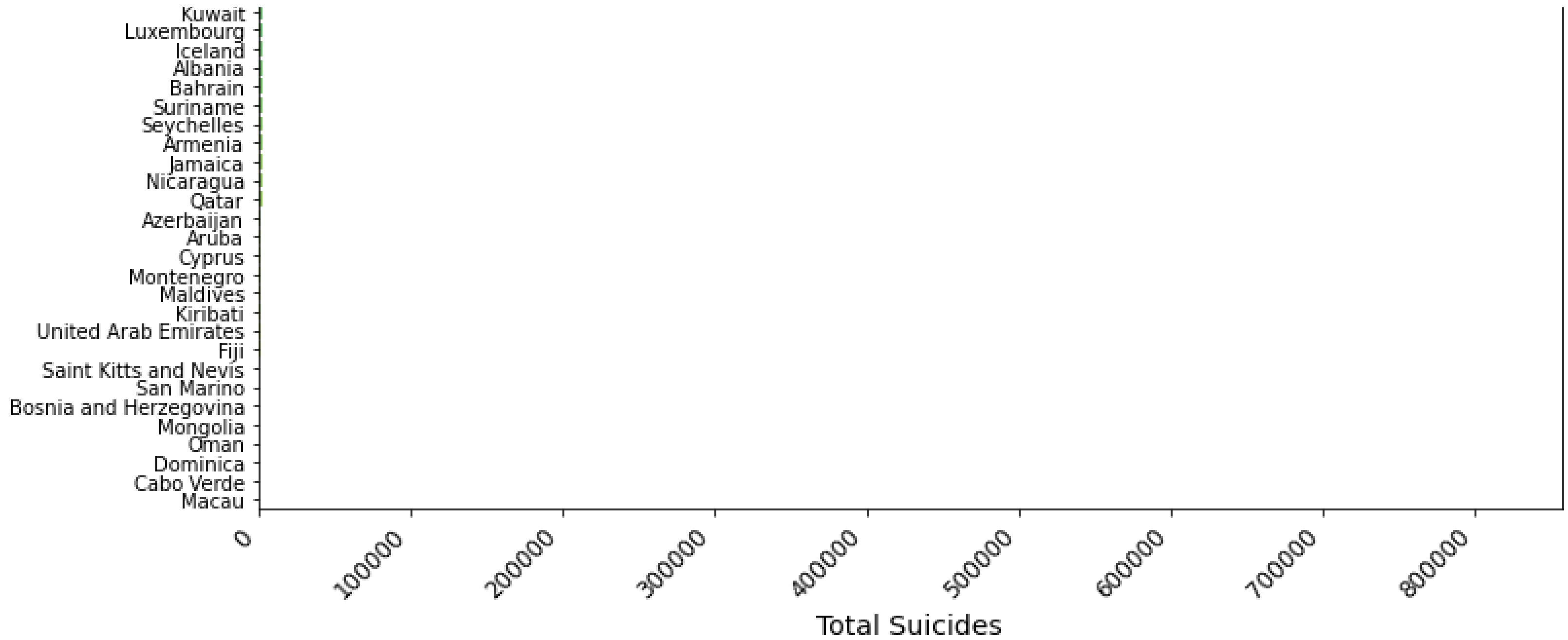
Out[47]:

	country	year	sex	age	suicides_no	population	gdp_for_year	suicides/100k	generation
0	Albania	1987	male	15-24 years	21.0	312900	2,156,624,900	6.711409	Millennials
1	Albania	1987	male	35-54 years	16.0	308000	2,156,624,900	5.194805	Millennials
2	Albania	1987	female	15-24 years	14.0	289700	2,156,624,900	4.832585	Millennials
3	Albania	1987	male	75+ years	1.0	21800	2,156,624,900	4.587156	Millennials
4	Albania	1987	male	25-34 years	9.0	274300	2,156,624,900	3.281079	Millennials

Ranking the countries by total suicides rate

In [49]:
```
# Assuming 'who' is the name of your DataFrame
# Group data by country and calculate total suicides
country_suicides = who.groupby('country')['suicides_no'].sum().reset_index()

# Sort the data by total suicides in descending order
country_suicides = country_suicides.sort_values(by='suicides_no', ascending=False)

# Create a larger bar plot using Seaborn with adjusted font size
plt.figure(figsize=(12, 15))  # Increase both width and height
sns.barplot(x='suicides_no', y='country', data=country_suicides, palette='viridis')
plt.xlabel('Total Suicides', fontsize=14)  # Increase font size
plt.ylabel('Country', fontsize=14)  # Increase font size
plt.title('Ranks of Countries by Total Suicides rate', fontsize=16)  # Increase font size

# Rotate country labels for better readability
plt.xticks(rotation=45, ha='right', fontsize=12)  # Rotate labels and set alignment

# Adjust spacing to avoid overlapping labels
plt.tight_layout()

plt.show()
```


Ranks of Countries by Total Suicides rate
Country
Russian Federation
United States
Japan
France
Ukraine
Germany
Republic of Korea
Brazil
United Kingdom
Poland
Thailand
Italy
Mexico
Kazakhstan
Canada
Spain
Argentina
Hungary
Australia
Romania
Belgium
Belarus
Sri Lanka
Colombia
Netherlands
Austria
Czech Republic
Cuba
Chile
Sweden
Bulgaria
Uzbekistan
Finland
Lithuania
Switzerland
Serbia
Portugal
Philippines
Ecuador
Croatia
Norway
Denmark
Slovakia
Uruguay
Greece
New Zealand
Kyrgyzstan
Ireland
Latvia
Israel
El Salvador
Singapore
Slovenia
Turkey
Puerto Rico
Guatemala
Turkmenistan
Estonia
South Africa
Costa Rica
Antigua and Barbuda
Paraguay
Mauritius
Grenada
Trinidad and Tobago
Panama
Saint Vincent and Grenadines
Guyana
Barbados
Georgia
Saint Lucia
Belize
Malta
Bahamas



Finding the correlations between suicides, GDP per capita and population

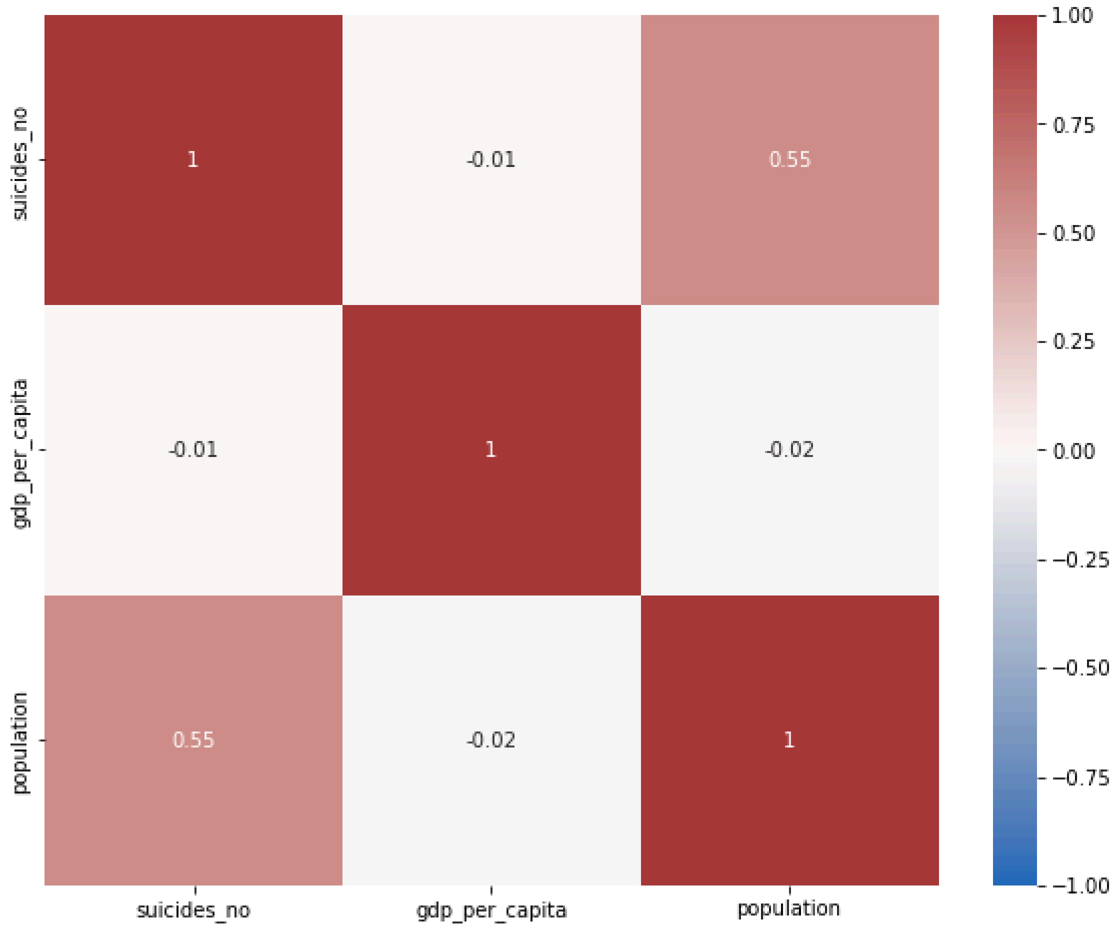
In [63]:

```
correlations =new_who [['suicides_no', 'gdp_per_capita', 'population']].corr()

print(correlations)
```

```
                suicides_no  gdp_per_capita  population
suicides_no            1.00           -0.01        0.55
gdp_per_capita        -0.01            1.00       -0.02
population             0.55           -0.02        1.00
```

In [64]:

```
hep = new_who[['suicides_no', 'gdp_per_capita', 'population']]

plt.figure(figsize=(10, 8))
sns.heatmap(hep.corr().round(2), annot=True, vmin=-1, vmax=1, cmap='vlag')
plt.show()
```

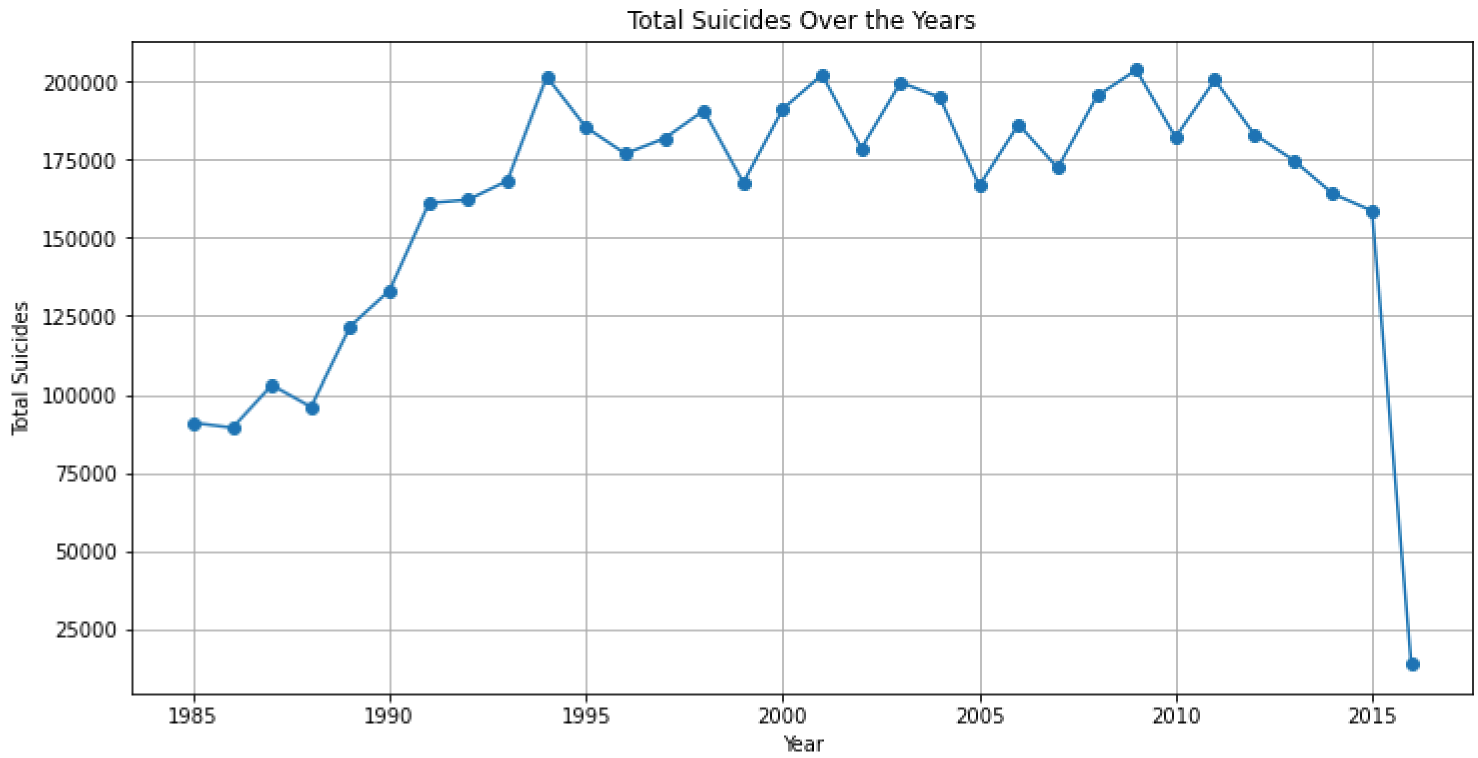


My conclusion from the correlation graph,From my point of view, there is a very high rate in suides_no

Use appropriate visual notation to visualise total suicides over years. Describe your findings

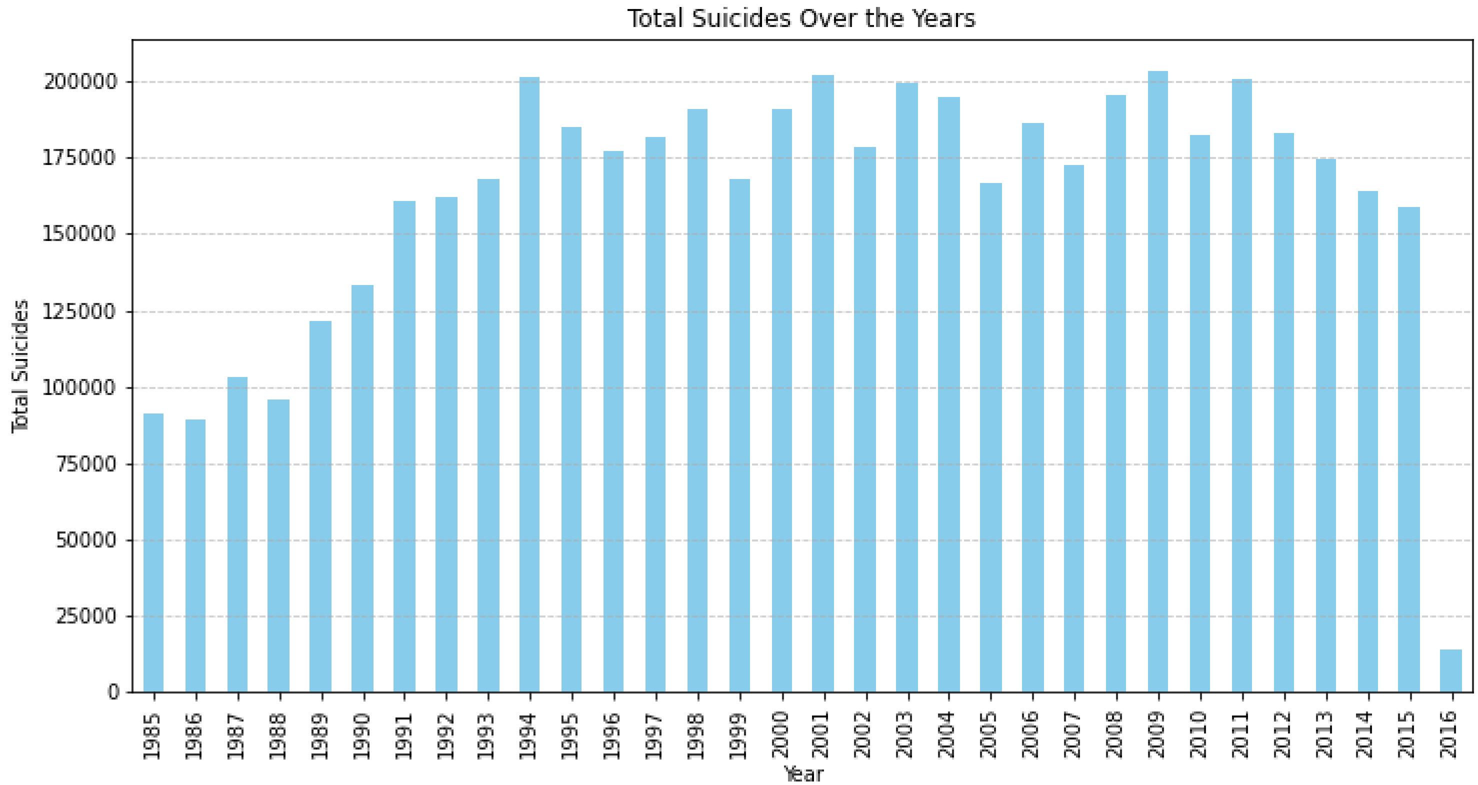
In [53]:
```
# Group the data by year and calculate the total suicides for each year
total_suicides_by_year = new_who.groupby('year')['suicides_no'].sum()

# Create a line plot to visualize total suicides over the years
plt.figure(figsize=(12, 6))
plt.plot(total_suicides_by_year.index, total_suicides_by_year.values, marker='o', linestyle='-')
plt.title('Total Suicides Over the Years')
plt.xlabel('Year')
plt.ylabel('Total Suicides')
```

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



In [54]:

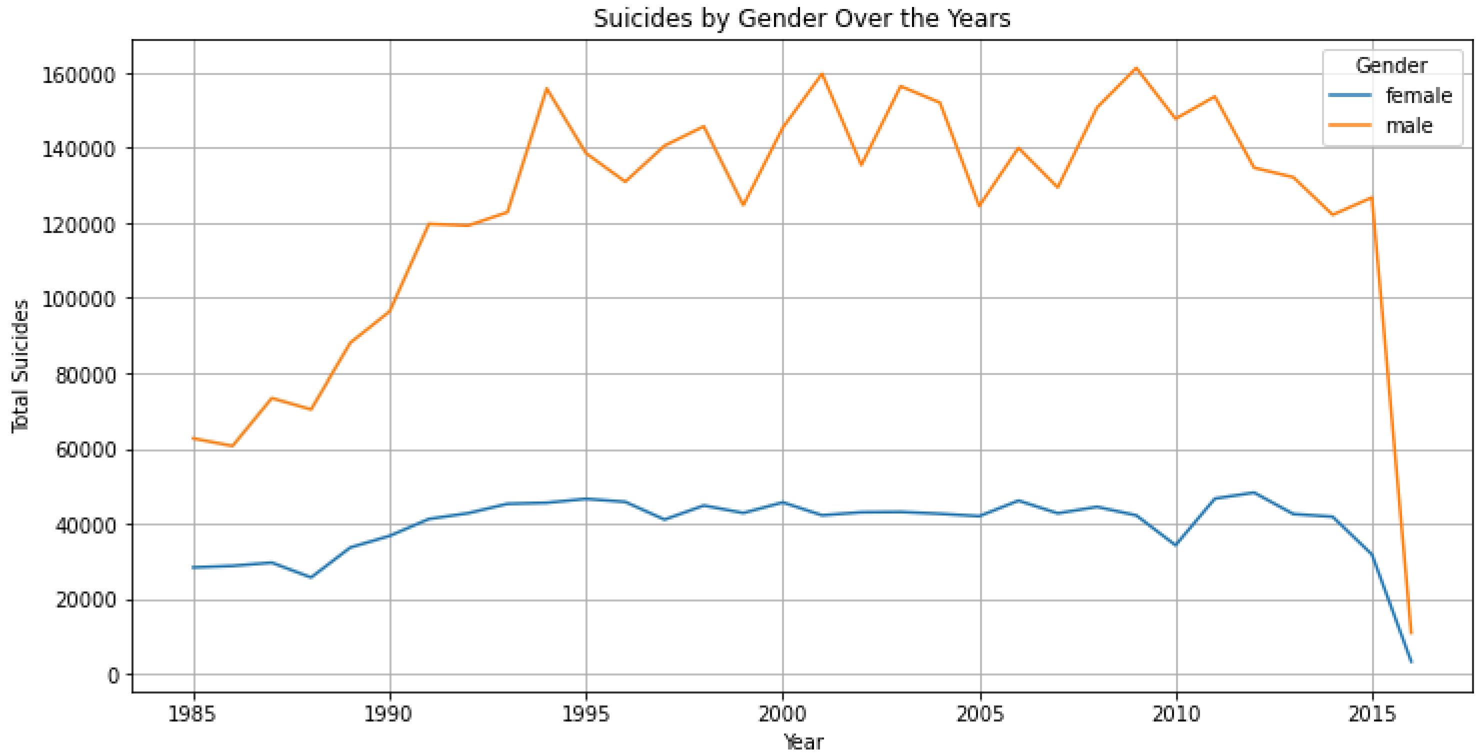
```
# Group the data by year and calculate the total suicides for each year
total_suicides_by_year = new_who.groupby('year')['suicides_no'].sum()

# Create a bar plot to visualize total suicides over the years
plt.figure(figsize=(12, 6))
total_suicides_by_year.plot(kind='bar', color='skyblue')
plt.title('Total Suicides Over the Years')
plt.xlabel('Year')
plt.ylabel('Total Suicides')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



My Finding 2009 and 1994 are the years that wentabove 200,000 mark, while 2003 just fall on the mark.1998,200,2004,2008,2003 fall in between 185,000 and 195,000. 2016 is the year with the lowest dead rate

Compare suicides by gender over years

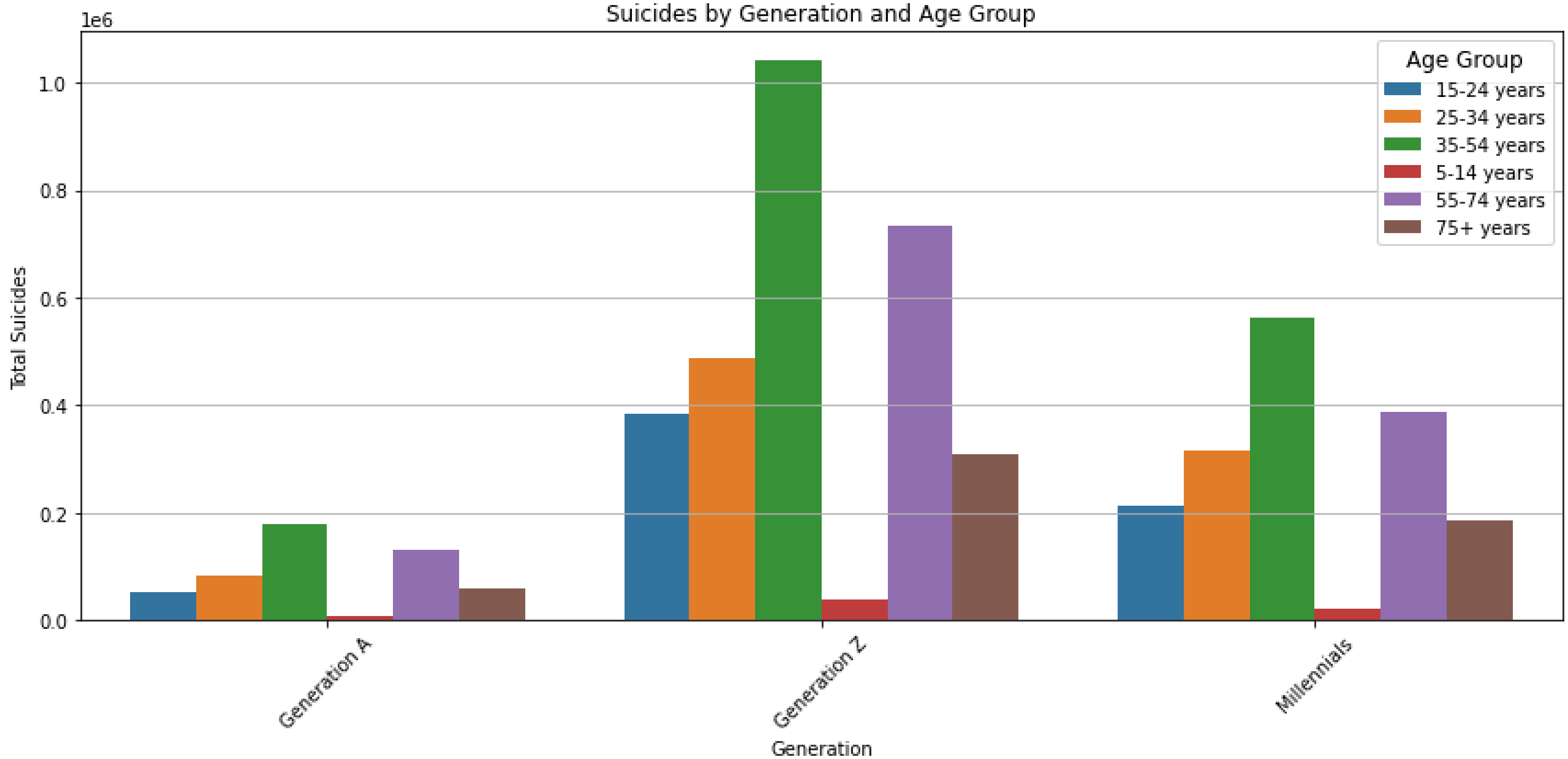
In [55]:
```
# Group the data by year and gender, and calculate the total suicides for each group
suicides_by_gender = new_who.groupby(['year', 'sex'])['suicides_no'].sum().reset_index()

# Create a line plot to compare suicides by gender over the years
plt.figure(figsize=(12, 6))
sns.lineplot(data=suicides_by_gender, x='year', y='suicides_no', hue='sex')
plt.title('Suicides by Gender Over the Years')
plt.xlabel('Year')
plt.ylabel('Total Suicides')
plt.grid(True)
plt.legend(title='Gender')
plt.show()
```



From the graph above it shows that male gener have a high rate in suicides and a sharp decline from 2015

Calculate and Visualise suicides on generation and on age group.

In [57]:
```
# Group the data by generation and age group and calculate the total suicides for each group
suicides_by_generation_age = new_who.groupby(['generation', 'age'])['suicides_no'].sum().reset_index()

# Create a bar plot to visualize suicides by generation and age group
plt.figure(figsize=(12, 6))
sns.barplot(data=suicides_by_generation_age, x='generation', y='suicides_no', hue='age')
plt.title('Suicides by Generation and Age Group')
plt.xlabel('Generation')
plt.ylabel('Total Suicides')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.legend(title='Age Group', title_fontsize='12')
plt.tight_layout()
plt.show()
```



From all the generation A, Z and Millenials, we can see age 35-54,55-74 and 25-34 years, they are the genraation with the hight suicides rate.

Section B

Task Two, Car Sale on AA

This is to demostrate data acquiotion (Scraping, cleaning and exploration, However scraping could noot be done due to security for the website

In [65]:
```
# importaing the data set
cars=pd.read_csv('used_cars.csv')
```

In [66]:
```
cars.head()
```

Out[66]:

	Sale title	Year	Body type	Number of seats	Number of reviews	Location	Distance	Mileage	Co2 emissions	Colour	Rating	price	Fuel type	Transmission	Engir siz
0	Toyota Yaris	2020	Hatchback	5	155	Steven Eagell Toyota Bedford	0	18,887	92 g/km	White	4.50	18890.00	Hybrid	Automatic	1.5
1	Toyota Yaris	2020	Hatchback	5	155	Steven Eagell Toyota Bedford	0	13,350	92 g/km	Red	4.50	19190.00	Hybrid	Automatic	1.5
2	Toyota Yaris	2020	Hatchback	5	155	Steven Eagell Toyota Bedford	0	13,552	92 g/km	White	4.50	19990.00	Hybrid	Automatic	1.5
3	Toyota C-HR	2020	Suv	5	1	Steven Eagell Toyota Bedford	0	55,220	86 g/km	Grey	2.50	20290.00	Hybrid	Automatic	1.8
4	Toyota Yaris	2021	Hatchback	5	155	Steven Eagell Toyota Bedford	0	14,156	92 g/km	Grey	4.50	20290.00	Hybrid	Automatic	1.5

Checking the data types

In [67]:

```
cars.dtypes
```

```
Out[67]: Sale title            object
         Year                   int64
         Body type             object
         Number of seats        int64
         Number of reviews      int64
         Location              object
         Distance               int64
         Mileage               object
         Co2 emissions         object
         Colour                object
         Rating               float64
         price                float64
         Fuel type             object
         Transmission          object
         Engine size           object
         dtype: object
```

Checking for NaN

```
In [68]: cars.isnull().sum()
```

```
Out[68]: Sale title           0
         Year                 0
         Body type            0
         Number of seats      0
         Number of reviews    0
         Location             0
         Distance             0
         Mileage              0
         Co2 emissions        0
         Colour               0
         Rating               0
         price                0
         Fuel type            0
         Transmission         0
         Engine size          0
         dtype: int64
```

```
In [69]: # Read the dataset into a DataFrame (replace 'your_dataset.csv' with your file)
         cars = pd.read_csv('used_cars.csv')

         # Check for missing values in each column
         missing_values = cars.isnull().sum()
```

```
# Check for unique values in each column
unique_values = cars.nunique()

# Check for data types of each column
data_types = cars.dtypes

# Combine the information into a summary DataFrame
summary_cars = pd.DataFrame({
    'Missing Values': missing_values,
    'Unique Values': unique_values,
    'Data Types': data_types
})

# Display columns with missing values
problematic_columns = summary_cars[summary_cars['Missing Values'] > 0]
print("Columns with missing values:")
print(problematic_columns)

# Display columns with too many unique values (potential categorical columns)
categorical_columns = summary_cars[summary_cars['Unique Values'] > len(cars) * 0.9]
print("\nPotential categorical columns:")
print(categorical_columns)

# Display columns with data type 'object' (usually text or categorical data)
text_columns = summary_cars[summary_cars['Data Types'] == 'object']
print("\nText (object) columns:")
print(text_columns)
```

```
Columns with missing values:
Empty DataFrame
Columns: [Missing Values, Unique Values, Data Types]
Index: []

Potential categorical columns:
Empty DataFrame
Columns: [Missing Values, Unique Values, Data Types]
Index: []

Text (object) columns:
               Missing Values  Unique Values Data Types
Sale title                  0            338     object
Body type                   0             16     object
Location                    0             30     object
Mileage                     0            605     object
Co2 emissions               0            197     object
Colour                      0             47     object
Fuel type                   0              9     object
Transmission                0              5     object
Engine size                 0             46     object
```

In [ ]: