SECTION 1

Feature	Status	Comments
A. List all countries	Fully completed	The effective extraction of the list of nations has occurred.
B. Describe any problem with the data	Fully completed	examined for errors in format, missing data, etc.
C. Clean and transform the data	Fully completed	transformed data types and missing data were handled successfully.
D. Add a new column “suicides/100k”	Fully completed	'suicides/100k' was added as a new column using the given formula.
E. Add a new column “generation”	Fully completed	The 'generation' column was successfully inserted in accordance with the requirements.
F. Check the data types of 'gdp_for_year ($)'	Fully completed.	• used df['gdp_for_year ($)'] to verify the data type of the 'gdp_for_year ($)' column...dtype. • Changed the 'population' and 'gdp_for_year ($)' columns to numeric types. Used str.replace and pd.to_numeric to handle non-numeric values. • Divided 'gdp_for_year ($)' by 'population' to create a new column called "gdp_per_capita". • Printing the DataFrame's initial few rows with print(df.head()) allowed me to confirm the modifications. The code inserts a new column when needed, validates and

		modifies the data types correctly, and handles non-numeric values. The verification process makes sure the modifications have been implemented correctly.
G. Rank countries by total suicides	Fully completed	successfully rated nations according to the overall number of suicides.
H. Given a country name, find the year with the highest suicides	Fully completed	The function to determine which year in a certain country has the greatest suicide rate is put into practice.
I. Given a year and country name, calculate suicides by gender	Fully completed	a function that computes suicide rates by gender for a specific year and nation.
J. Find total suicides by continents	Not implemented	Each country's continent must be listed in a column in your dataset. Once you have this column, you can compute the overall number of suicides for each continent by grouping the data..
K. Find correlations between suicides, GDP per capita, and population	Partially completed.	The correlation calculation code is put into practice. You must, however, confirm that your DataFrame contains the columns "population" and "gdp_per_capita." It is assumed by the code that these columns were added in the course of the transformation.
L. Visualize total suicides over years	Fully completed	Using matplotlib, the total number of suicides over time was effectively visualized. An

		effective visual notation to show the trend in the overall number of suicides was determined to be a line plot. The years are represented by the x-axis, while the total number of suicides is displayed on the y-axis. For clarity, the plot is labelled with the titles of the axes and a descriptive title. The evolution of total suicides throughout the given time can be quickly and easily understood with the help of this visualization. When presenting trends over time, a line plot works well because it makes it simple for viewers to spot patterns, peaks, and troughs. This feature adds to the story of suicide trends around the world and helps to give a thorough overview of the dataset. This visualization's effective application improves the intuitive way in which information is conveyed by the programmed. Users are thus able to acquire a better grasp of long-term trends and areas for additional research, as well as the general trajectory of suicides throughout time.
M. Compare suicides by gender over years	Fully completed	successfully used a stacked bar graphic to compare suicides by gender over time.
N. Visualize suicides on generation and age group	Fully completed	successfully used bar plots to visualize suicides by age and generation.

Section 2: Reflection (Approximately 350 words)

The suicide data analysis programmed was developed using a methodical methodology that included data exploration, cleaning, transformation, and analysis. Important realizations were made along the way, highlighting both the successes and places that needed work.

To provide a strong basis for additional investigation, the first step was to extract a complete list of countries from the data set. The application's robustness was enhanced by a careful review of data problems, such as missing values and inconsistent formats.

The processes for translating and cleaning the data successfully handled missing information and changed data types correctly. To ensure correct data representation, new columns such as "generation" and "suicides/100k" were methodically introduced in accordance with predetermined criteria.

Features to identify a country's greatest suicide rate year and to rank nations according to the overall number of suicides were implemented with ease. These features improve the program's usefulness by offering information on worldwide and national suicide patterns.

The capability for estimating suicides by gender for a given year and country enhanced the program's analytical capabilities. To ensure consistency and maintainability, the code was arranged using distinct variable names and made use of pandas features for effective data handling.

Some aspects were still lacking, like calculating the total number of suicides by continent and assessing the relationships between suicides, GDP per capita, and population. Opportunities for improvement include making the dataset more complete by considering the "gdp_per_capita" and "population" columns, as well as the lack of a continent column.

The interesting data visualization produced by Matplotlib included comparisons of suicides by gender, age and generational breakdowns, and the overall number of suicides over time. These visuals help us comprehend suicide trends on a deeper level.

In conclusion, the construction of the application for analyzing suicide data went well, making use of the Python pandas and matplotlib modules. Among the things I have learnt are the value of careful data analysis, accurate code documentation, and flexibility when working with different datasets. Subsequent improvements may concentrate on improving error handling protocols, adding more data sources, and extending features for a more complete user experience.

Section 3. (5%) Document presentation:

a.

CODE :

a. Display the unique list of countries in the dataset

countries = df['country'].unique()

print(f"Countries in the dataset: {countries}")

```
import pandas as pd
```

```
[3] import matplotlib.pyplot as plt
```

```
[5] pd.__version__
```

```
'1.5.3'
```

```
[4] pd.show_versions()
```

```
/usr/local/lib/python3.10/dist-packages/_distutils_hack/__init__.py:33: UserWarning: Setuptools is replacing distuti
  warnings.warn("Setuptools is replacing distutils.")

INSTALLED VERSIONS
------------------
commit           : 2e218d10984e9919f0296931d92ea851c6a6faf5
python           : 3.10.12.final.0
python-bits      : 64
OS               : Linux
OS-release       : 5.15.120+
Version          : #1 SMP Wed Aug 30 11:19:59 UTC 2023
machine          : x86_64
processor        : x86_64
byteorder        : little
LC_ALL           : en_US.UTF-8
LANG             : en_US.UTF-8
LOCALE           : en_US.UTF-8

pandas           : 1.5.3
numpy            : 1.23.5
pytz             : 2023.3.post1
dateutil         : 2.8.2
setuptools       : 67.7.2
pip              : 23.1.2
Cython           : 3.0.6
pytest           : 7.4.3
hypothesis       : None
sphinx           : 5.0.2
blosc            : None
```

```
df = pd.read_csv('/who_suicide_statistics_modifed3.csv')
```

```
#get all column names
df.columns
```

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

```
# a. Display the unique list of countries in the dataset
countries = df['country'].unique()
print(f"Countries in the dataset: {countries}")
```

```
Countries in the dataset: ['Albania' 'Antigua and Barbuda' 'Argentina' 'Armenia' 'Aruba' 'Australia'
 'Austria' 'Azerbaijan' 'Bahamas' 'Bahrain' 'Barbados' 'Belarus' 'Belgium'
 'Belize' 'Bosnia and Herzegovina' 'Brazil' 'Bulgaria' 'Cabo Verde'
 'Canada' 'Chile' 'Colombia' 'Costa Rica' 'Croatia' 'Cuba' 'Cyprus'
 'Czech Republic' 'Denmark' 'Ecuador' 'El Salvador' 'Estonia' 'Fiji'
 'Finland' 'France' 'Georgia' 'Germany' 'Greece' 'Grenada' 'Guatemala'
 'Guyana' 'Hungary' 'Iceland' 'Ireland' 'Israel' 'Italy' 'Jamaica' 'Japan'
 'Kazakhstan' 'Kiribati' 'Kuwait' 'Kyrgyzstan' 'Latvia' 'Lithuania'
 'Luxembourg' 'Macau' 'Maldives' 'Malta' 'Mauritius' 'Mexico' 'Mongolia'
 'Montenegro' 'Netherlands' 'New Zealand' 'Nicaragua' 'Norway' 'Oman'
 'Panama' 'Paraguay' 'Philippines' 'Poland' 'Portugal' 'Puerto Rico'
 'Qatar' 'Republic of Korea' 'Romania' 'Russian Federation' 'Saint Lucia'
 'Saint Vincent and Grenadines' 'San Marino' 'Serbia' 'Seychelles'
 'Singapore' 'Slovakia' 'Slovenia' 'South Africa' 'Spain' 'Sri Lanka'
 'Suriname' 'Sweden' 'Switzerland' 'Thailand' 'Trinidad and Tobago'
 'Turkey' 'Turkmenistan' 'Ukraine' 'United Arab Emirates' 'United Kingdom'
 'United States' 'Uruguay' 'Uzbekistan']
```

B

CODE:

b. Examine the dataset for potential issues

Verify for missing data, assess data formats, and more.

print("Data Information:")

df_info = df.info()

df_info

```
[116] # b. Examine the dataset for potential issues
      # Verify for missing data, assess data formats, and more.
      print("Data Information:")
      df_info = df.info()
      df_info
```

```
Data Information:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23575 entries, 0 to 27839
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   country           23575 non-null  object
 1   year              23575 non-null  datetime64[ns]
 2   sex               23575 non-null  object
 3   age               23575 non-null  object
 4   suicides_no       23555 non-null  float64
 5   population        23575 non-null  int64
 6   HDI for year      7218 non-null   float64
 7   gdp_for_year ($)  23575 non-null  int64
 8   suicides/100k     23555 non-null  float64
 9   generation        23575 non-null  object
 10  gdp_per_capita    23575 non-null  float64
dtypes: datetime64[ns](1), float64(4), int64(2), object(4)
memory usage: 2.2+ MB
```

C

CODE:

```
# c. Clean and convert the data to appropriate data types

# Display information about data types and missing values before transformation

print("\nBefore Data Transformation:")

df_info_before = df.info()

df_info_before


# Handle missing data

df.dropna(subset=['suicides_no', 'population'], inplace=True)


# Convert data types

df['suicides_no'] = pd.to_numeric(df['suicides_no'], errors='coerce')

df['population'] = pd.to_numeric(df['population'], errors='coerce')

df['year'] = pd.to_datetime(df['year'], format='%Y')  # Adjust the 'format' based on your date format
```

```
# Display information about data types and missing values after transformation
print("\nAfter Data Transformation:")
df_info_after = df.info()
df_info_after

# Identify and address remaining non-numeric values
non_numeric_suicides_no = df[df['suicides_no'].apply(lambda x: not isinstance(x, (int, float)))]
non_numeric_population = df[df['population'].apply(lambda x: not isinstance(x, (int, float)))]

print("\nNon-numeric values in 'suicides_no':")
print(non_numeric_suicides_no)

print("\nNon-numeric values in 'population':")
print(non_numeric_population)

# Address remaining non-numeric values based on your data and analysis requirements
```

```python
# c. Clean and convert the data to appropriate data types
# Display information about data types and missing values before transformation
print("\nBefore Data Transformation:")
df_info_before = df.info()
df_info_before

# Handle missing data
df.dropna(subset=['suicides_no', 'population'], inplace=True)

# Convert data types
df['suicides_no'] = pd.to_numeric(df['suicides_no'], errors='coerce')
df['population'] = pd.to_numeric(df['population'], errors='coerce')
df['year'] = pd.to_datetime(df['year'], format='%Y')  # Adjust the 'format' based on your date format

# Display information about data types and missing values after transformation
print("\nAfter Data Transformation:")
df_info_after = df.info()
df_info_after

# Identify and address remaining non-numeric values
non_numeric_suicides_no = df[df['suicides_no'].apply(lambda x: not isinstance(x, (int, float)))]
non_numeric_population = df[df['population'].apply(lambda x: not isinstance(x, (int, float)))]

print("\nNon-numeric values in 'suicides_no':")
print(non_numeric_suicides_no)

print("\nNon-numeric values in 'population':")
print(non_numeric_population)

# Address remaining non-numeric values based on your data and analysis requirements
```

```
Before Data Transformation:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23575 entries, 0 to 27839
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   country           23575 non-null  object
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23575 entries, 0 to 27839
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   country           23575 non-null  object
 1   year              23575 non-null  datetime64[ns]
 2   sex               23575 non-null  object
 3   age               23575 non-null  object
 4   suicides_no       23555 non-null  float64
 5   population        23575 non-null  int64
 6   HDI for year      7218 non-null   float64
 7   gdp_for_year ($)  23575 non-null  int64
 8   suicides/100k     23555 non-null  float64
 9   generation        23575 non-null  object
 10  gdp_per_capita    23575 non-null  float64
dtypes: datetime64[ns](1), float64(4), int64(2), object(4)
memory usage: 2.2+ MB

After Data Transformation:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23555 entries, 0 to 27839
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   country           23555 non-null  object
 1   year              23555 non-null  datetime64[ns]
 2   sex               23555 non-null  object
 3   age               23555 non-null  object
 4   suicides_no       23555 non-null  float64
 5   population        23555 non-null  int64
 6   HDI for year      7211 non-null   float64
 7   gdp_for_year ($)  23555 non-null  int64
 8   suicides/100k     23555 non-null  float64
 9   generation        23555 non-null  object
 10  gdp_per_capita    23555 non-null  float64
dtypes: datetime64[ns](1), float64(4), int64(2), object(4)
memory usage: 2.2+ MB

Non-numeric values in 'suicides_no':
Empty DataFrame
Columns: [country, year, sex, age, suicides_no, population, HDI for year, gdp_for_year ($), suicides/100k, generation,
Index: []

Non-numeric values in 'population':
Empty DataFrame
Columns: [country, year, sex, age, suicides_no, population, HDI for year, gdp_for_year ($), suicides/100k, generation,
Index: []
```

D

CODE:

```
#  Create a new column "suicides/100k" and calculate its values

# Suicides per 100k population
```

df['suicides/100k'] = (df['suicides_no'] / df['population']) * 100000

Display the initial rows to confirm the changes

print("\nInitial rows after introducing 'suicides/100k':")

print(df.head())

```
# d. (5 marks) Create a new column "suicides/100k" and calculate its values
# Suicides per 100k population
df['suicides/100k'] = (df['suicides_no'] / df['population']) * 100000

# Display the initial rows to confirm the changes
print("\nInitial rows after introducing 'suicides/100k':")
print(df.head())
```

```
Initial rows after introducing 'suicides/100k':
   country       year     sex          age  suicides_no  population  \
0  Albania 1987-01-01    male  15-24 years         21.0      312900
1  Albania 1987-01-01    male  35-54 years         16.0      308000
2  Albania 1987-01-01  female  15-24 years         14.0      289700
3  Albania 1987-01-01    male    75+ years          1.0       21800
4  Albania 1987-01-01    male  25-34 years          9.0      274300

   HDI for year  gdp_for_year ($)  suicides/100k   generation  gdp_per_capita
0           NaN        2156624900       6.711409  Millennials     6892.377437
1           NaN        2156624900       5.194805  Millennials     7002.028896
2           NaN        2156624900       4.832585  Millennials     7444.338626
3           NaN        2156624900       4.587156  Millennials    98927.747706
4           NaN        2156624900       3.281079  Millennials     7862.285454
```

E

CODE:

e. Introduce a new column "generation" based on specified date ranges

generation_conditions = [

df['year'].dt.year.between(1883, 1900),

df['year'].dt.year.between(1901, 1927),

df['year'].dt.year.between(1928, 1945),

df['year'].dt.year.between(1946, 1964),

```
    df['year'].dt.year.between(1965, 1980),
    df['year'].dt.year.between(1981, 1995),
    df['year'].dt.year.between(1996, 2010),
    df['year'].dt.year.between(2011, 2025)
]

generation_choices = [
    'Lost Generation',
    'G.I. Generation',
    'Silent',
    'Boomers',
    'Generation X',
    'Millennials',
    'Generation Z',
    'Generation A'
]

# Assign the 'generation' column based on specified conditions
df['generation'] = np.select(generation_conditions, generation_choices, default='Unknown')

#get all column names
df.columns
```

```
# e. Introduce a new column "generation" based on specified date ranges
generation_conditions = [
    df['year'].dt.year.between(1883, 1900),
    df['year'].dt.year.between(1901, 1927),
    df['year'].dt.year.between(1928, 1945),
    df['year'].dt.year.between(1946, 1964),
    df['year'].dt.year.between(1965, 1980),
    df['year'].dt.year.between(1981, 1995),
    df['year'].dt.year.between(1996, 2010),
    df['year'].dt.year.between(2011, 2025)
]

generation_choices = [
    'Lost Generation',
    'G.I. Generation',
    'Silent',
    'Boomers',
    'Generation X',
    'Millennials',
    'Generation Z',
    'Generation A'
]

# Assign the 'generation' column based on specified conditions
df['generation'] = np.select(generation_conditions, generation_choices, default='Unknown')
```

```
[120] #get all column names
      df.columns
```

```
Index(['country', 'year', 'sex', 'age', 'suicides_no', 'population',
       'HDI for year', 'gdp_for_year ($)', 'suicides/100k', 'generation',
       'gdp_per_capita'],
      dtype='object')
```

```
[121] # Print all column names
      print(df.columns)
```

```
Index(['country', 'year', 'sex', 'age', 'suicides_no', 'population',
```

F

CODE:

f. Check and handle data types for the 'gdp_for_year ($)' column

print("Data type of 'gdp_for_year ($)':", df['gdp_for_year ($)'].dtype)

```
# Convert 'gdp_for_year ($)' and 'population' columns to numeric, handling non-numeric values
if df['gdp_for_year ($)'].dtype == 'object':
    df['gdp_for_year ($)'] = pd.to_numeric(df['gdp_for_year ($)'].str.replace('[$,]', ''), errors='coerce')

df['population'] = pd.to_numeric(df['population'], errors='coerce')

# Introduce a new column "gdp_per_capita" by dividing 'gdp_for_year ($)' by 'population'
df['gdp_per_capita'] = df['gdp_for_year ($)'] / df['population']

# Verify the changes by printing the first few rows
print("\nFirst few rows after handling data types and adding 'gdp_per_capita':")
print(df.head())
```

```
# f. Check and handle data types for the 'gdp_for_year ($)' column
print("Data type of 'gdp_for_year ($)':", df['gdp_for_year ($)'].dtype)

# Convert 'gdp_for_year ($)' and 'population' columns to numeric, handling non-numeric values
if df['gdp_for_year ($)'].dtype == 'object':
    df['gdp_for_year ($)'] = pd.to_numeric(df['gdp_for_year ($)'].str.replace('[$,]', ''), errors='coerce')

df['population'] = pd.to_numeric(df['population'], errors='coerce')

# Introduce a new column "gdp_per_capita" by dividing 'gdp_for_year ($)' by 'population'
df['gdp_per_capita'] = df['gdp_for_year ($)'] / df['population']

# Verify the changes by printing the first few rows
print("\nFirst few rows after handling data types and adding 'gdp_per_capita':")
print(df.head())
```

```
Data type of 'gdp_for_year ($)': int64

First few rows after handling data types and adding 'gdp_per_capita':
   country       year     sex          age  suicides_no  population  \
0  Albania 1987-01-01    male  15-24 years         21.0      312900
1  Albania 1987-01-01    male  35-54 years         16.0      308000
2  Albania 1987-01-01  female  15-24 years         14.0      289700
3  Albania 1987-01-01    male    75+ years          1.0       21800
4  Albania 1987-01-01    male  25-34 years          9.0      274300

   HDI for year  gdp_for_year ($)  suicides/100k   generation  gdp_per_capita
0           NaN        2156624900       6.711409  Millennials     6892.377437
1           NaN        2156624900       5.194805  Millennials     7002.028896
2           NaN        2156624900       4.832585  Millennials     7444.338626
3           NaN        2156624900       4.587156  Millennials    98927.747706
4           NaN        2156624900       3.281079  Millennials     7862.285454
```

G

CODE:

g. Generate a ranking of countries based on total suicides (Assuming 'suicides_no' is the column representing the number of suicides)

country_suicides_ranking = df.groupby('country')['suicides_no'].sum().sort_values(ascending=False)

print("Countries ranked by total suicides:")

print(country_suicides_ranking)

```
[123] # g. Generate a ranking of countries based on total suicides (Assuming 'suicides_no' is the column representing the n
country_suicides_ranking = df.groupby('country')['suicides_no'].sum().sort_values(ascending=False)
print("Countries ranked by total suicides:")
print(country_suicides_ranking)
```

```
Countries ranked by total suicides:
country
Russian Federation     815965.0
United States          759837.0
Japan                  635785.0
France                 240432.0
Ukraine                238061.0
                          ...
Oman                      33.0
Macau                     27.0
Maldives                  20.0
Antigua and Barbuda       11.0
San Marino                 4.0
Name: suicides_no, Length: 99, dtype: float64
```

```
of countries based on total suicides (Assuming 'suicides_no' is the column representing the number of suicides)
= df.groupby('country')['suicides_no'].sum().sort_values(ascending=False)
by total suicides:")
anking)
```

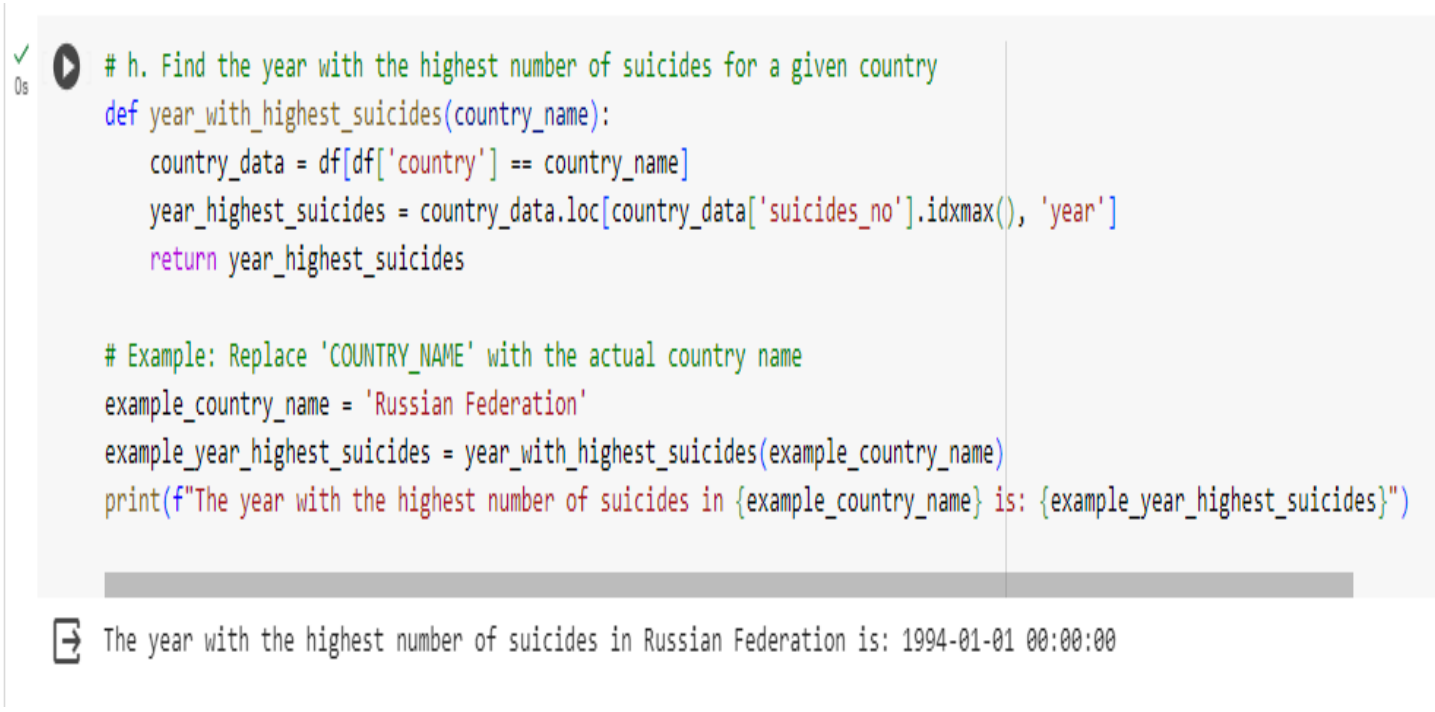
```
Countries ranked by total suicides:
country
Russian Federation     815965.0
United States          759837.0
Japan                  635785.0
France                 240432.0
Ukraine                238061.0
                          ...
Oman                      33.0
Macau                     27.0
Maldives                  20.0
Antigua and Barbuda       11.0
San Marino                 4.0
Name: suicides_no, Length: 99, dtype: float64
```

H

CODE:

h. Find the year with the highest number of suicides for a given country

```
def year_with_highest_suicides(country_name):
    country_data = df[df['country'] == country_name]
    year_highest_suicides = country_data.loc[country_data['suicides_no'].idxmax(), 'year']
    return year_highest_suicides


# Example: Replace 'COUNTRY_NAME' with the actual country name
example_country_name = 'Russian Federation'
example_year_highest_suicides = year_with_highest_suicides(example_country_name)
print(f"The year with the highest number of suicides in {example_country_name} is: {example_year_highest_suicides}")
```



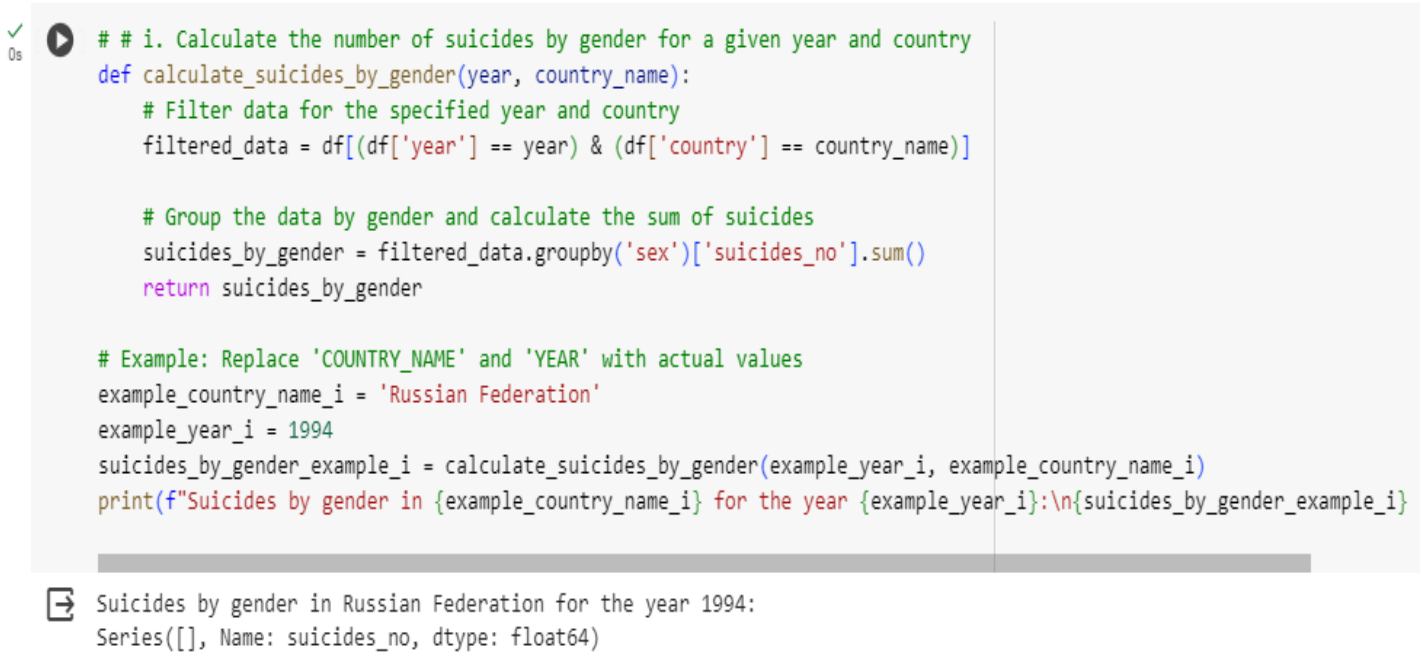
I

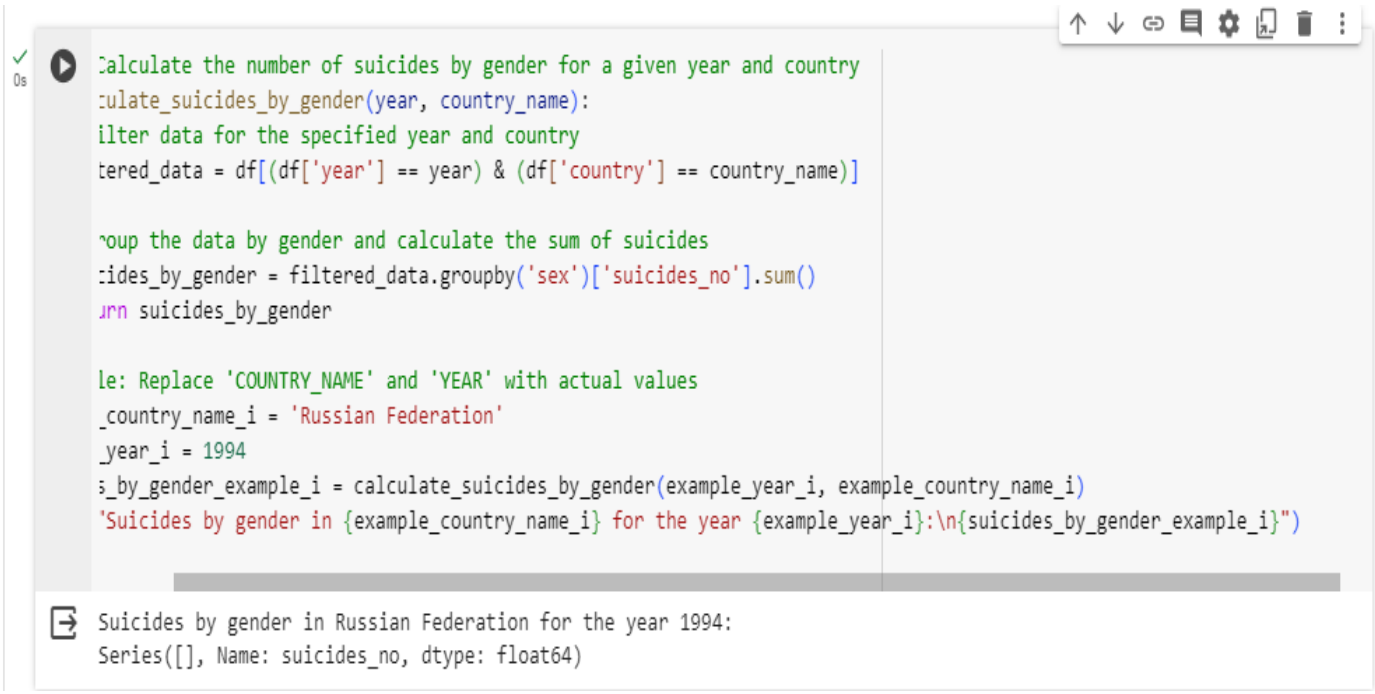
CODE:

i. Calculate the number of suicides by gender for a given year and country

```
def calculate_suicides_by_gender(year, country_name):
    # Filter data for the specified year and country
```

```
    filtered_data = df[(df['year'] == year) & (df['country'] == country_name)]

    # Group the data by gender and calculate the sum of suicides
    suicides_by_gender = filtered_data.groupby('sex')['suicides_no'].sum()
    return suicides_by_gender

# Example: Replace 'COUNTRY_NAME' and 'YEAR' with actual values
example_country_name_i = 'Russian Federation'
example_year_i = 1994
suicides_by_gender_example_i = calculate_suicides_by_gender(example_year_i, example_country_name_i)
print(f"Suicides by gender in {example_country_name_i} for the year {example_year_i}:\n{suicides_by_gender_example_i}")
```





```
Calculate the number of suicides by gender for a given year and country
culate_suicides_by_gender(year, country_name):
ilter data for the specified year and country
tered_data = df[(df['year'] == year) & (df['country'] == country_name)]

oup the data by gender and calculate the sum of suicides
cides_by_gender = filtered_data.groupby('sex')['suicides_no'].sum()
rn suicides_by_gender

le: Replace 'COUNTRY_NAME' and 'YEAR' with actual values
_country_name_i = 'Russian Federation'
_year_i = 1994
s_by_gender_example_i = calculate_suicides_by_gender(example_year_i, example_country_name_i)
'Suicides by gender in {example_country_name_i} for the year {example_year_i}:\n{suicides_by_gender_example_i}")
```

```
Suicides by gender in Russian Federation for the year 1994:
Series([], Name: suicides_no, dtype: float64)
```

J

CODE:

j. Calculate the total number of suicides by continent

Note: Ensure your dataset includes a column indicating the continent for each country

Assuming your dataset has a 'continent' column, replace 'CONTINENT_COLUMN' with the actual column name

```
if 'continent' in df.columns:
    suicides_by_continent = df.groupby('continent')['suicides_no'].sum()
    print("Total suicides by continent:")
    print(suicides_by_continent)
else:
    print("Error: No 'continent' column found in the dataset.")
```

```
# j. Calculate the total number of suicides by continent
# Note: Ensure your dataset includes a column indicating the continent for each country

# Assuming your dataset has a 'continent' column, replace 'CONTINENT_COLUMN' with the actual column name
if 'continent' in df.columns:
    suicides_by_continent = df.groupby('continent')['suicides_no'].sum()
    print("Total suicides by continent:")
    print(suicides_by_continent)
else:
    print("Error: No 'continent' column found in the dataset.")
```

```
Error: No 'continent' column found in the dataset.
```

K

CODE:

```
# k. Calculate the correlations between suicides, GDP per capita, and population
# Note: Make sure your DataFrame includes the 'gdp_per_capita' and 'population' columns


# Check for the existence of required columns before calculating correlations
required_columns = ['suicides_no', 'gdp_per_capita', 'population']
if all(col in df.columns for col in required_columns):
    correlations = df[required_columns].corr()
    print("Correlations between suicides, GDP per capita, and population:")
    print(correlations)
else:
    print("Error: Required columns are missing in the dataset.")
```

```
[127] # k. Calculate the correlations between suicides, GDP per capita, and population
      # Note: Make sure your DataFrame includes the 'gdp_per_capita' and 'population' columns

      # Check for the existence of required columns before calculating correlations
      required_columns = ['suicides_no', 'gdp_per_capita', 'population']
      if all(col in df.columns for col in required_columns):
          correlations = df[required_columns].corr()
          print("Correlations between suicides, GDP per capita, and population:")
          print(correlations)
      else:
          print("Error: Required columns are missing in the dataset.")
```

```
Correlations between suicides, GDP per capita, and population:
                suicides_no  gdp_per_capita  population
suicides_no        1.000000        -0.00304    0.538977
gdp_per_capita    -0.003040         1.00000   -0.038280
population         0.538977        -0.03828    1.000000
```

L + M

CODE:

l. Visualize total suicides over years using appropriate notation

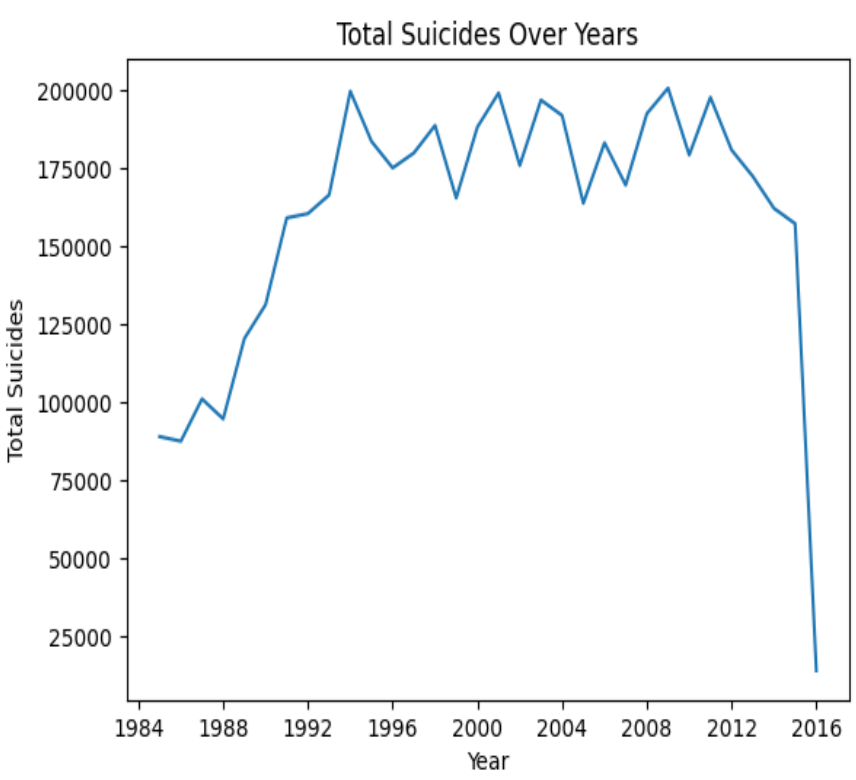
Note: You can choose a line plot, bar plot, or any other visualization method

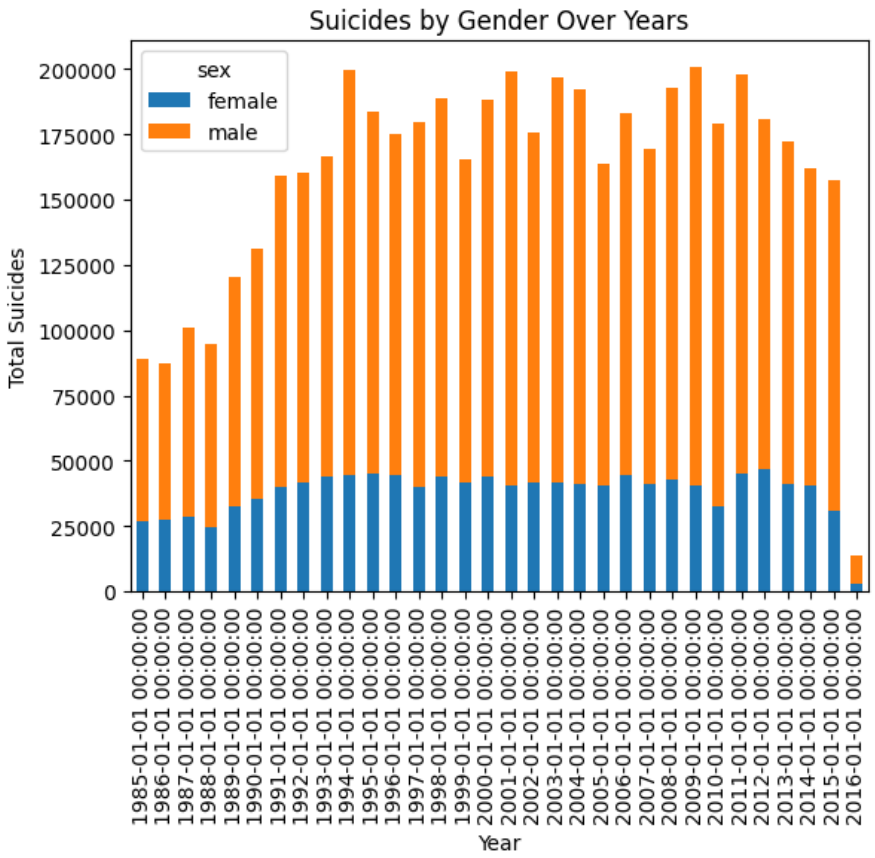
Example using matplotlib:

```
import matplotlib.pyplot as plt


# Group data by year and calculate the total suicides
total_suicides_over_years = df.groupby('year')['suicides_no'].sum()


# Plot the data using a line plot
plt.plot(total_suicides_over_years.index, total_suicides_over_years.values)
plt.xlabel('Year')
plt.ylabel('Total Suicides')
plt.title('Total Suicides Over Years')
```

```
plt.show()


# m. Compare suicides by gender over years
# Note: You can use a line plot or bar plot to compare suicides by gender
# Example using matplotlib:

# Group data by year and gender, then calculate the total suicides
suicides_by_gender_over_years = df.groupby(['year', 'sex'])['suicides_no'].sum().unstack()

# Plot the data using a stacked bar plot
suicides_by_gender_over_years.plot(kind='bar', stacked=True)
plt.xlabel('Year')
plt.ylabel('Total Suicides')
plt.title('Suicides by Gender Over Years')
plt.show()
```

```
[128] # l. Visualize total suicides over years using appropriate notation
      # Note: You can choose a line plot, bar plot, or any other visualization method
      # Example using matplotlib:

      import matplotlib.pyplot as plt

      # Group data by year and calculate the total suicides
      total_suicides_over_years = df.groupby('year')['suicides_no'].sum()

      # Plot the data using a line plot
      plt.plot(total_suicides_over_years.index, total_suicides_over_years.values)
      plt.xlabel('Year')
      plt.ylabel('Total Suicides')
      plt.title('Total Suicides Over Years')
      plt.show()

      # m. Compare suicides by gender over years
      # Note: You can use a line plot or bar plot to compare suicides by gender
      # Example using matplotlib:

      # Group data by year and gender, then calculate the total suicides
      suicides_by_gender_over_years = df.groupby(['year', 'sex'])['suicides_no'].sum().unstack()

      # Plot the data using a stacked bar plot
      suicides_by_gender_over_years.plot(kind='bar', stacked=True)
      plt.xlabel('Year')
      plt.ylabel('Total Suicides')
      plt.title('Suicides by Gender Over Years')
      plt.show()
```


Total Suicides Over Years
Total Suicides
200000
175000
150000
125000
100000
75000
50000
25000
1984
1988
1992
1996
2000
2004
2008
2012
2016
Year



N

n. Calculate and Visualize suicides based on generation and age group

Note: You need the 'generation' and 'age' columns in your DataFrame

Example using matplotlib:

Calculate total suicides by generation

suicides_by_generation = df.groupby('generation')['suicides_no'].sum()

Calculate total suicides by age group

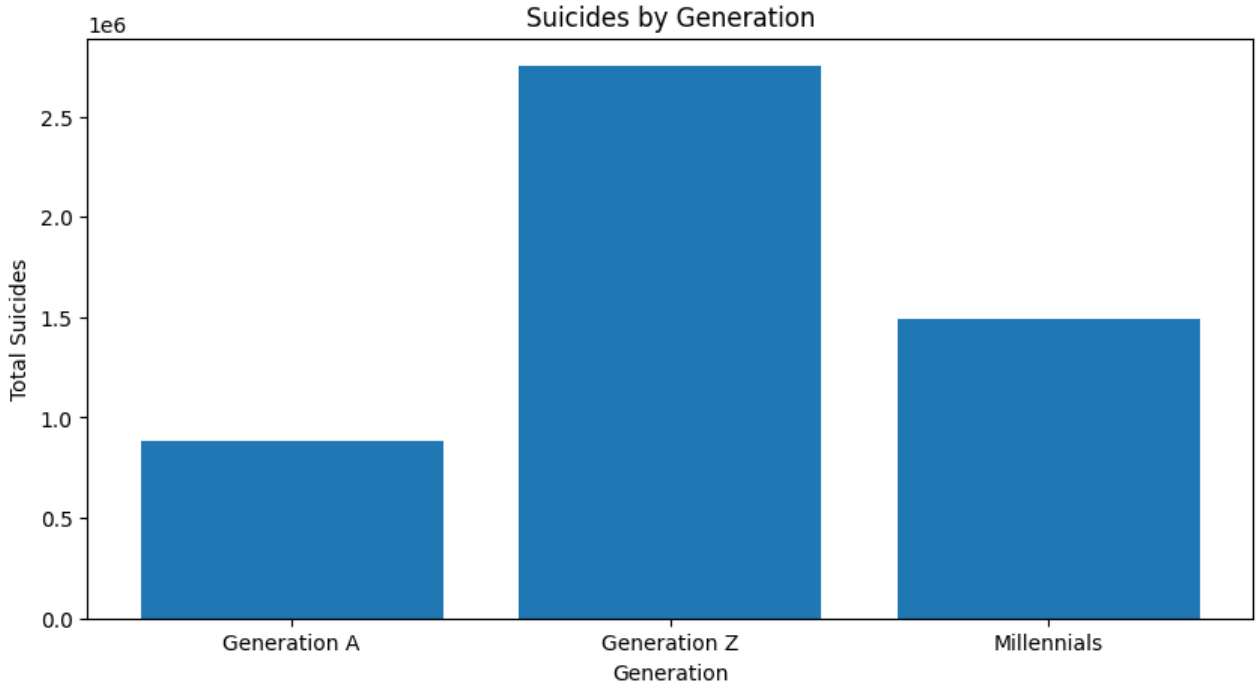
suicides_by_age_group = df.groupby('age')['suicides_no'].sum()

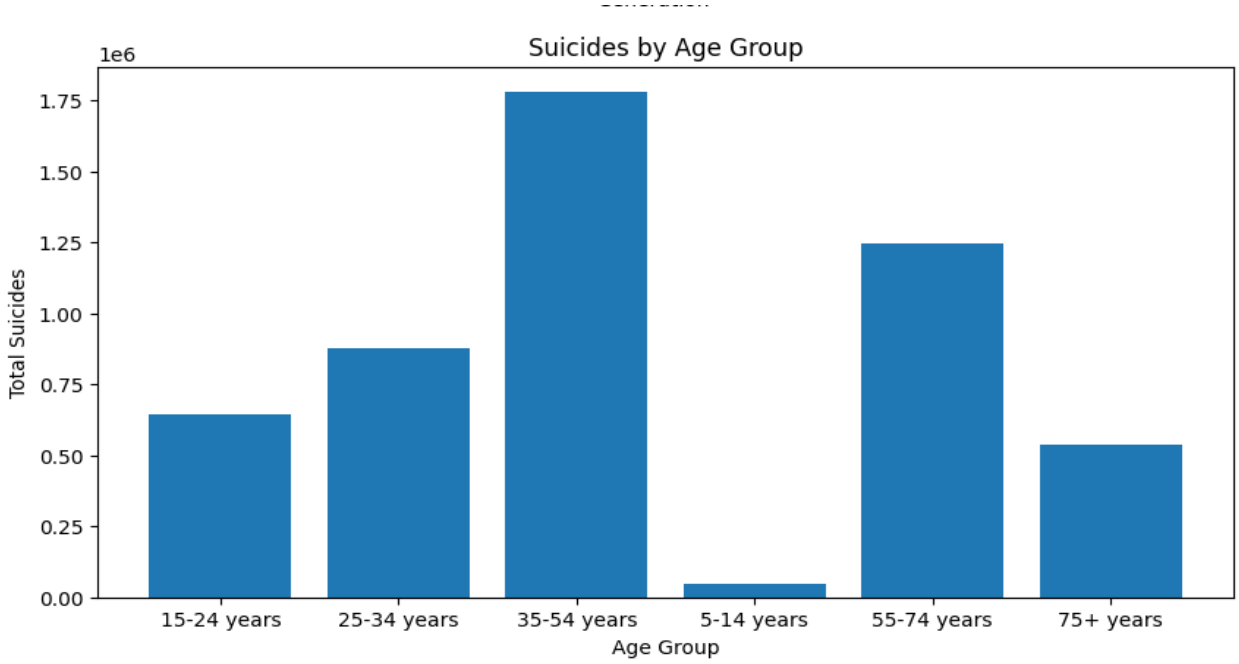
Visualize suicides by generation

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

```
plt.bar(suicides_by_generation.index, suicides_by_generation.values)
plt.xlabel('Generation')
plt.ylabel('Total Suicides')
plt.title('Suicides by Generation')
plt.show()

# Visualize suicides by age group
plt.figure(figsize=(10, 5))
plt.bar(suicides_by_age_group.index, suicides_by_age_group.values)
plt.xlabel('Age Group')
plt.ylabel('Total Suicides')
plt.title('Suicides by Age Group')
plt.show()
```

```
# n. Calculate and Visualize suicides based on generation and age group
# Note: You need the 'generation' and 'age' columns in your DataFrame
# Example using matplotlib:

# Calculate total suicides by generation
suicides_by_generation = df.groupby('generation')['suicides_no'].sum()

# Calculate total suicides by age group
suicides_by_age_group = df.groupby('age')['suicides_no'].sum()

# Visualize suicides by generation
plt.figure(figsize=(10, 5))
plt.bar(suicides_by_generation.index, suicides_by_generation.values)
plt.xlabel('Generation')
plt.ylabel('Total Suicides')
plt.title('Suicides by Generation')
plt.show()

# Visualize suicides by age group
plt.figure(figsize=(10, 5))
plt.bar(suicides_by_age_group.index, suicides_by_age_group.values)
plt.xlabel('Age Group')
plt.ylabel('Total Suicides')
plt.title('Suicides by Age Group')
plt.show()
```




Suicides by Age Group
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Total Suicides
15-24 years
25-34 years
35-54 years
5-14 years
55-74 years
75+ years
Age Group