Suicide: Are we dealing with a pandemic that cannot be contained?

An Exploratory and Predictive Analysis on the Global suicide rate (1985-2016)

GIFT E. IDAMA
Department of Computer and Information sciences.
Towson University.
Towson, MD, USA.
gidama1@students.towson.edu

1. INTRODUCTION:

Suicide is death caused by injuring oneself with the intent to die. A suicide attempt is when someone harms themselves with any intent to end their life, but they do not die as a result of their actions. Many factors can increase the risk for suicide or protect against it. Suicide is connected to other forms of injury and violence. For example, people who have experienced violence, including child abuse, bullying, or sexual violence have a higher suicide risk. Being connected to family and community support and having easy access to health care can decrease suicidal thoughts and behaviors.

Suicide and suicide attempts cause serious emotional, physical, and economic impacts. People who attempt suicide and survive may experience serious injuries that can have long-term effects on their health. They may also experience depression and other mental health concerns. The good news is that more than 90% of people who attempt suicide and survive never go on to die by suicide.

Suicide and suicide attempts affect the health and well-being of friends, loved ones, co-workers, and the community. When people die by suicide, their surviving family and friends may experience shock, anger, guilt, symptoms of depression or anxiety, and may even experience thoughts of suicide themselves.

The financial toll of suicide on society is also costly. In 2019, suicide and nonfatal self-harm cost the nation nearly $490 billion in medical costs, work loss costs, value of statistical life, and quality of life costs.

In addition to that, the pandemic during the past two years caused much stress on people. Therefore, I decided to research suicide rates and try to find out which group and portion of the population has a higher risk when facing suicidal thoughts.

2. PROJECT OBJECTIVES:

The project seeks to study the global suicide rate among various countries. This project analyzes the dataset that covers the numbers of suicide cases happening in different countries from the 80s. In the first part of this project, I will try to see if there are patterns involved in suicide rates among countries, gender, age group etc. For the purpose of this study, some research questions have been proposed and with the help of our exploratory data analysis, conclusions would be made based off the visualization of the trends/patterns in our data.

Some of the proposed research questions that will be answered during data preprocessing are.

- Which countries have the highest and least suicide rates and suicide counts.
- Is suicide rate depreciating or increasing over the years?
- Are there signs that certain age groups are inclined towards suicide?
- Is gender an important factor?
- What variables correlate to suicide rate? (Variable importance).

Also, during the exploratory data analysis, I will be picking a country of interest (United states) to analyze the specific rates in this country. Some questions would be answered during this analysis, such as,

- How does the number of suicides vary over time in the United States?
- In the United states, do adults also have higher suicide rates?
- Which gender is more inclined towards suicide in the US?
- Which generation has the highest suicide number in the US?

The second part of this project involves building several regression models that will be trained to understand the correlation between the features/predictor of the dataset and the target variable(suicides_no). We will be predicting the suicide number/count given a specific year, pertaining to a certain age group and gender.

The Final part of my project involves comparison of the performance of each model. Each model would be compared based on accuracy to see which one performed best. The performance evaluation would be done using the R-squared score.

3. DATASET DESCRIPTION:

The masters.csv data file contains the global suicide rates of various countries during the 80s. It also contains the suicide counts of each country pertaining to the sex of the individual. The Global suicide rate dataset contains about 29821 instances and observations along with 12 variables/columns.

This data set was gotten from kaggle, it contains the suicide rates of various countries from the year 1985 to 2016. The 12 features in the dataset include.

1 Country
2 Year
3 Sex
4 Age
5 suicide_no (target variable)
6 population
7 suicides/100k population
8 country-year
9 HDI for year
10 Gdp_for_year
11 Gdp_per_capital
12 Generation

```
In [48]: import pandas as pd
         data = pd.read_csv('master.csv')
         data.head(5)
```

Out[48]:

	country	year	sex	age	suicides_no	population	suicides/100k pop	country-year	HDI for year	gdp_for_year ($)	gdp_per_capita ($)	generation
0	Albania	1987	male	15-24 years	21	312900	6.71	Albania1987	NaN	2,156,624,900	796	Generation X
1	Albania	1987	male	35-54 years	16	308000	5.19	Albania1987	NaN	2,156,624,900	796	Silent
2	Albania	1987	female	15-24 years	14	289700	4.83	Albania1987	NaN	2,156,624,900	796	Generation X
3	Albania	1987	male	75+ years	1	21800	4.59	Albania1987	NaN	2,156,624,900	796	G.I. Generation
4	Albania	1987	male	25-34 years	9	274300	3.28	Albania1987	NaN	2,156,624,900	796	Boomers

Fig1: The view of the dataset

4. DATA WRANGLING

After importing the dataset successfully, I must clean up the dataset and make it even more organized. Studying the dataset, we find that there are some problems that exists in the dataset.

1. Nomenclature of the variables/columns.
2. Some columns are redundant for our analysis.
3. Duplicate rows exist.
4. Rows and columns with zero value.
5. Age column violation.

4.1. Dealing with Nomenclature of the variables:

To do that, I first check the property of the dataset, and then I see that some names for the column could potentially be an issue due to the syntax. Some columns exist with symbols and white spaces in the variable name that might not be recognized which can lead to difficulty when trying to make use of them. Dealing with those names is very imperative. We must rename them and remove the symbols and white spaces.

- gdp_for_year and gdp_per_capital ($) have white spaces and some symbols that are not recognized.
- Suicide/100k pop is the same as suicide rate, so it was changed to suicide_rate.
- Sex was renamed to gender
- The hyphen in country-year was replaced.
- The white spaces in HDI for year were replaced.

```
# Let's examine the column nomenclature.
data.columns
```

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

```
# Dealing with nomenclature of the variables
data = data.rename(columns = {'sex' : 'gender'})
data = data.rename(columns = {'suicides/100k pop' : 'suicides_per_100k_pop'})
data = data.rename(columns = {'country-year' : 'country_year'})
data = data.rename(columns = {'HDI for year' : 'HDI_for_year'})
data = data.rename(columns = {' gdp_for_year ($) ' : 'gdp_for_year'})
data = data.rename(columns = {'gdp_per_capita ($)' : 'gdp_per_capital'})
```

```
data.head(1)
```

	country	year	gender	age	suicides_no	population	suicides_per_100k_pop	country_year	HDI_for_year	gdp_for_year	gdp_per_capital	generation
0	Albania	1987	male	15-24 years	21	312900	6.71	Albania1987	NaN	2,156,624,900	796	Generation X

Fig2: Dealing with the nomenclature

4.2. Removing redundant columns:

After fixing the naming of the columns, I dropped the columns that I will not use in this EDA. Several redundant and unnecessary columns exist that we won't be needed during the analysis. So, these columns need to be removed. HDI for year, country year, GDP for year are all redundant for our analysis, therefore there were all removed.

```
# Removing redundant columns
data = data.drop(['country_year', 'HDI_for_year', 'gdp_for_year'], axis = 1)
data.head(3)
```

	country	year	gender	age	suicides_no	population	suicides_per_100k_pop	gdp_per_capital	generation
0	Albania	1987	male	15-24 years	21	312900	6.71	796	Generation X
1	Albania	1987	male	35-54 years	16	308000	5.19	796	Silent
2	Albania	1987	female	15-24 years	14	289700	4.83	796	Generation X

Fig3: Eliminating redundant rows

4.3. Dealing with Missing values:

After examining the dataset, there were no missing values present.

```
# checking for missing values
data.isnull()
```

	country	year	gender	age	suicides_no	population	suicides_per_100k_pop	gdp_per_capital	generation
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
27815	False	False	False	False	False	False	False	False	False
27816	False	False	False	False	False	False	False	False	False
27817	False	False	False	False	False	False	False	False	False
27818	False	False	False	False	False	False	False	False	False
27819	False	False	False	False	False	False	False	False	False

Fig4: Checking for missing values

4.4. Dealing with Duplicate row:

Duplicate rows were dropped using the drop_duplicates function in pandas.

data = data.drop_duplicates (keep = 'first')

```
# checking if Duplicate rows exist
data.duplicated()
```

```
0         False
1         False
2         False
3         False
4         False
          ...
27815     False
27816     False
27817     False
27818     False
27819     False
Length: 27820, dtype: bool
```

Fig5: After duplicate rows have been removed.

4.5. Removing rows with zero values:

Rows or columns with zero value just makes our dataset look so messy and there are not important for our analysis. Removing them does not affect the structure of our dataset.

```
# Removing rows will zero value in certain columns

data = data[data.suicides_no != 0]
```

```
data = data[data.suicide_rate != 0]
```

Fig6: Removing rows with zero values

After dealing with the few problems in our dataset, the dataset is now organized and ready for analysis.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27820 entries, 0 to 27819
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   country          27820 non-null  object
 1   year             27820 non-null  int64
 2   gender           27820 non-null  object
 3   age              27820 non-null  object
 4   suicides_no      27820 non-null  int64
 5   population       27820 non-null  int64
 6   suicide_rate     27820 non-null  float64
 7   gdp_per_capital  27820 non-null  int64
 8   generation       27820 non-null  object
dtypes: float64(1), int64(4), object(4)
memory usage: 1.9+ MB
```

Fig7: Dataset information

4.6. Dealing with the Age column violation:

There are 6 unique age groups in our dataset. Using feature engineering, I will be creating two new columns, one column will contain the minimum value of each age range. While the other will contain the categories of each age group. Using if else statements, each age group would be categorized into an age category.

The steps required for the feature engineering is shown below.

Step 1: identify the unique age group in the dataset

```
data['age'].unique()
```

```
array(['15-24 years', '35-54 years', '75+ years', '25-34 years',
       '55-74 years', '5-14 years'], dtype=object)
```

Step 2: Extract the first two characters to represent that age group.

```
# Feature Engineering - creating new column for age group
# grabbing the minimum value in the age range in order to arrive at a single int value
data['MinAgeRange'] = data['age'].str[:2]
data.head(50)
```

Step 3: Remove the hyphen present in some values.

```
data['MinAgeRange'].tail(3)
```

```
27817    5-
27818    5-
27819    55
Name: MinAgeRange, dtype: object
```

```
data['MinAgeRange'] = data['MinAgeRange'].map(lambda x: x.replace('-',''))
```

```
data['MinAgeRange'] = data['MinAgeRange'].astype(int)
data['MinAgeRange'].tail(3)
```

```
27817     5
27818     5
27819    55
Name: MinAgeRange, dtype: int32
```

Step 4: Categorize the age group into various categories.

```
# Feature Engineering
# Categorizing the age range into age category
def Group(x):
    if(x >= 15 and x < 25):
        return "Teenagers"
    elif(x >= 25 and x < 35):
        return "Adults"
    elif(x >= 35 and x < 55):
        return "Middle_Aged"
    elif(x >= 55):
        return "Elderly"
    else:
        return "Adolescent"
# Match each column to the new category
data['AgeCategory'] = data['MinAgeRange'].map(lambda x: Group(x))
```

```
# convert new column to type String
data['AgeCategory'] = data['AgeCategory'].astype(str)
```

```
data['AgeCategory'].tail(3)
```

```
27817    Adolescent
27818    Adolescent
27819       Elderly
Name: AgeCategory, dtype: object
```

We arrive at two new columns titled 'MinAgeRange' and 'AgeCategory' respectively.

5. EXPLORATORY DATA ANALYSIS:

Exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. Primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

Under this section, I will be answering some of the questions proposed earlier in this article.

5.1. Comparing the suicide count in each country:

Question 1: Countries with the highest and least suicide count?

```
# Countries with the highest total suicide count

data.groupby('country')['suicides_no'].sum().nlargest(10)
```

```
country
Russian Federation    1209742
United States         1034013
Japan                  806902
France                 329127
Ukraine                319950
Name: suicides_no, dtype: int64
```

The country with the highest total suicide count is the Russian federation, the next country with the highest total suicide count is United states of America. While the country with the least total number of suicide count is San Marino.

```
# Countries with the least total suicide count

data.groupby('country')['suicides_no'].sum().nsmallest(10)
```

```
country
San Marino              4
Antigua and Barbuda    11
Maldives               20
Macau                  27
Oman                   33
Grenada                38
Cabo Verde             42
Kiribati               53
Bahamas                93
Seychelles             98
Name: suicides_no, dtype: int64
```

Bar chart of the top 10 countries with the highest total suicide count is shown below. And a second bar chart of the top 10 countries with the least total suicide count is also shown below.

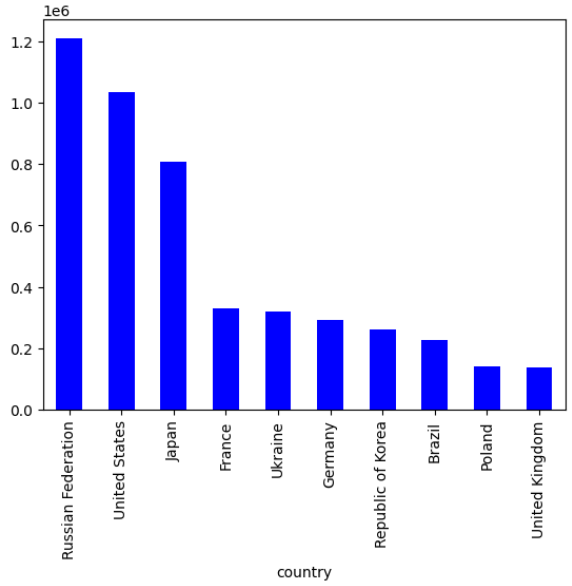


Fig8: Top 10 countries with the highest total suicide count.

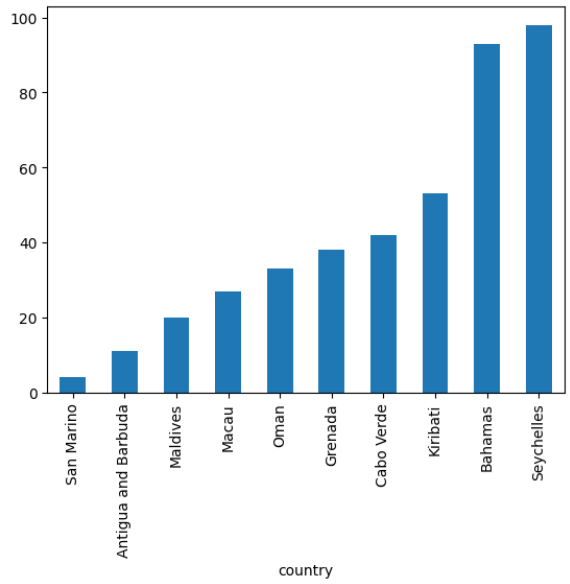


Fig9: Top 10 countries with the least total suicide count.

5.2. Comparing Suicide count over the years.

Question 2: Is suicide rate depreciating or increasing over the years?

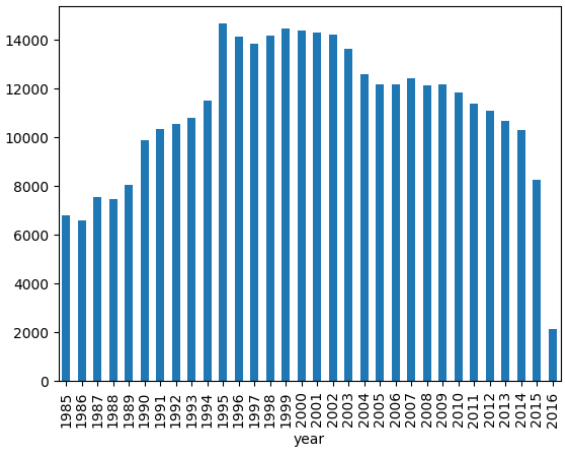


Fig10: Bar chart showing the suicide rate over the years.

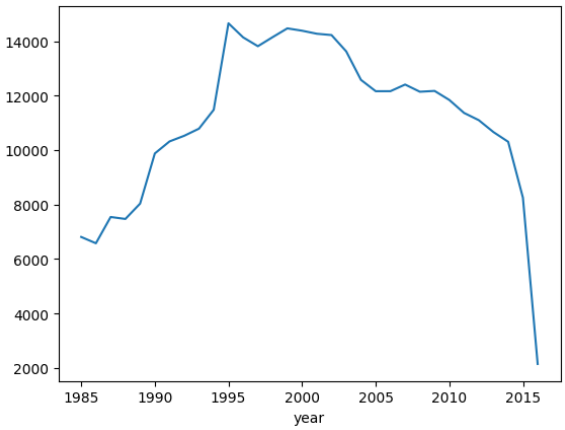


Fig11: Line plot showing the suicide rate over the years.

From observing our Line Plot, we can see a sharp drop in suicides from 1985 to 1986. This decrease could be due to awareness of suicide & mental health in the 80s, as well as improved recognition of those at risk.

After that we can se that the suicide rate continues to rise gradually until we get a sharp spike from the year 1994 to 1995 before dropping and rising gradually again. There's a fluctuation in the suicide rate over the years.

The year with the highest suicide rate was 1995, followed by 1999 and them 2000. We can conclude from our charts and plot that suicide rate was on the high from 2000 to 2004. Noticeably, the suicide rate started dropping from the year 2009.

```
# Comparing Year and Suicide rate

data.groupby(['year'])['suicide_rate'].sum().nlargest(10)
```

```
year
1995    14660.26
1999    14473.91
2000    14387.45
2001    14276.21
2002    14227.72
1998    14150.72
1996    14142.21
1997    13817.83
2003    13627.58
2004    12581.80
Name: suicide_rate, dtype: float64
```

Fig12: Top 10 years with the highest suicide rate.

Taking into consideration the suicide count, 1999 was the year with the highest total suicide deaths.

```
# Suicide count vs Year
# Total suicide count for each year

data.groupby(['year'])['suicides_no'].sum().nlargest(10)
```

```
year
1999    256119
2002    256095
2003    256079
2000    255832
2001    250652
1998    249591
1996    246725
1995    243544
2009    243487
2004    240861
Name: suicides_no, dtype: int64
```

Fig13: Top 10 year with the highest total suicide count.

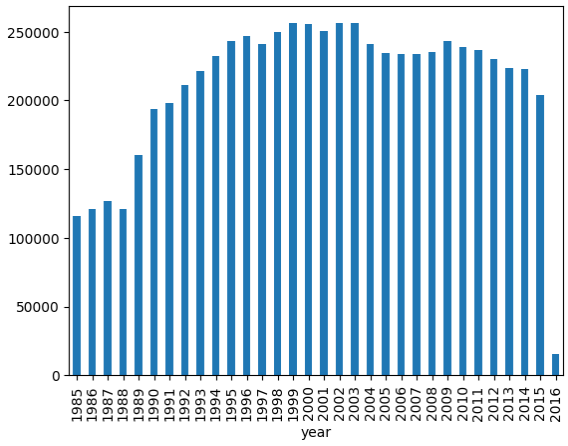


Fig14: Histogram showing the distribution of the suicide count over the years.

Taking into consideration the maximum suicide deaths that happen in a year, 1994 has the max suicide death

out of all the years. 22338 suicide deaths occurred in the year 1994.

```
# Suicide count vs Year
# The year with  the highest suicide count

data.groupby(['year'])['suicides_no'].max().nlargest(5)
```

```
year
1994    22338
1995    21706
2001    21262
2000    21063
1999    20705
Name: suicides_no, dtype: int64
```

5.3. Comparing Suicide count among each Age group.

Question 3: Are certain age groups more inclined towards suicide?

Earlier we performed feature engineering in order to categorize the age group. Let me give a breakdown of each category.

5-14 years – Adolescent

15–24 years – Teenagers

25-34 years – Adults

35-54 years – Middle Aged adults

55-74 years – Elderly

75+ years - Elderly

```
# Comparing suicide count among each age group
data.groupby('AgeCategory')['suicides_no'].sum()
```

```
AgeCategory
Adolescent       52264
Adults         1123912
Elderly        2311561
Middle_Aged    2452141
Teenagers       808542
Name: suicides_no, dtype: int64
```

Fig15: Total suicide count of each age group.

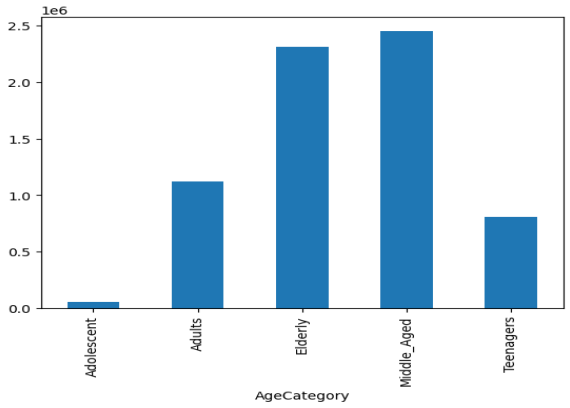


Fig16: Distribution of the suicide count over each age group.

The data clearly illustrates that Middle-aged adults (35-55) have the highest suicide count. Elderly has the next highest, followed by adults while adolescent has the least suicide count.

Next section, we will do a comparison between age, gender and suicide count.

5.4. Comparing suicide count among gender.

Question 4: Which gender is more inclined towards suicide?

Let's first get the count of female and male in our dataset.

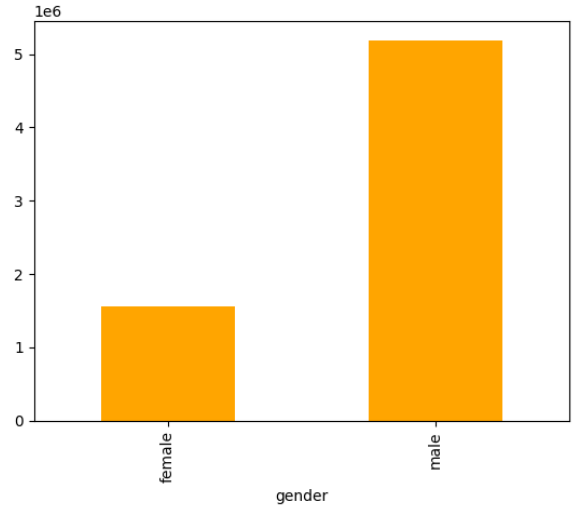
```
# Number of male and female in the data set
data['gender'].value_counts()
```

```
male      12286
female    11253
Name: gender, dtype: int64
```

Next, we visualize and analyze the suicide count in each gender, and we can compare the age group in each gender as well.

```
data.groupby('gender')['suicides_no'].sum()
```

```
gender
female    1559510
male      5188910
Name: suicides_no, dtype: int64
```



From the illustration, it is evident that males are more inclined to suicide compared to females. Now let's compare the age group in each gender that is more inclined towards suicide. Let's see if it correlates with the age group vs suicide count analysis.

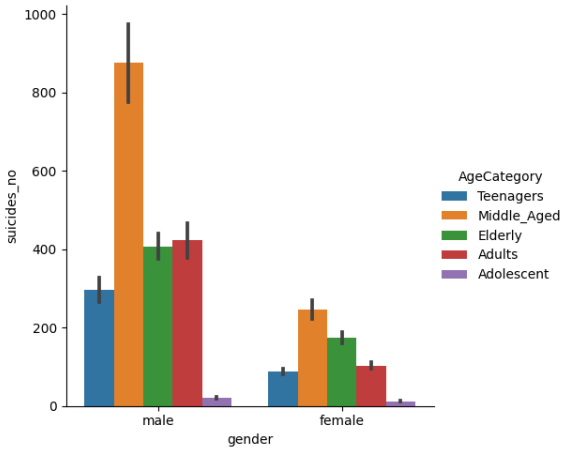


Fig17: Analysis of the age group in each gender.

From the data, in both gender, middle-aged male and middle-aged female has the highest suicide count. That is, middle-aged male and females are more inclined towards suicide. This makes sense as it correlates with the age group vs suicide count analysis from earlier.

Both Gender show that middle aged adults are the leading group in suicide.

5.5. Comparing Suicide count among generations.

We have 6 unique generations in our dataset.

```
data['generation'].unique()
```

```
array(['Generation X', 'Silent', 'G.I. Generation', 'Boomers',
       'Millenials', 'Generation Z'], dtype=object)
```

Fig18: Generations in our dataset.

Taking into consideration the total suicide count. We can see that boomers had the highest total suicide count. While the least was generation z (This could be due to inadequate data to support our analysis).

```
# Suicide count vs Generation
# Taking into consideration, the total suicide count
# Boomers had the highest total suicide count

data.groupby(['generation'])['suicides_no'].sum()
```

```
generation
Boomers            2284498
G.I. Generation     510009
Generation X       1532804
Generation Z         15906
Millenials          623459
Silent             1781744
Name: suicides_no, dtype: int64
```

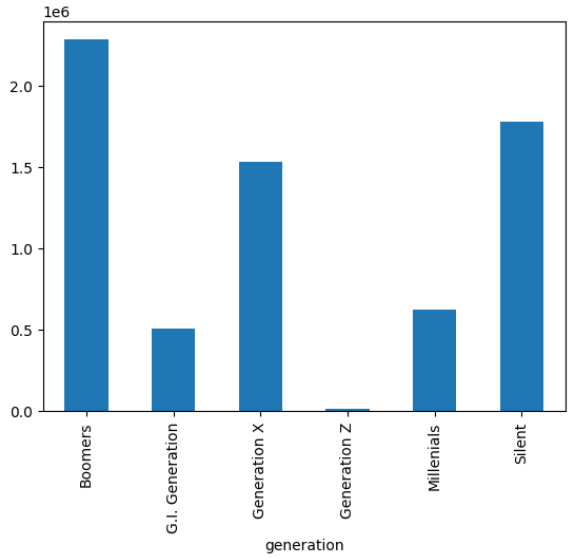


Fig19: Bar chart showing the distribution of suicide count among generations

The chart illustrates that over the 31 years, boomers have been involved in more suicide deaths than any other generations.

5.6. Suicide in the United states

Question 1: How does the number of suicides vary over time?

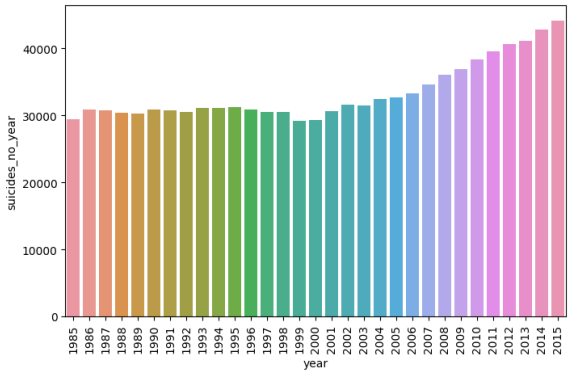
```
suicides_no_year = []

for y in df_us['year'].unique():
    suicides_no_year.append(sum(df_us[df_us['year'] == y]['suicides_no']))

n_suicides_year = pd.DataFrame(suicides_no_year, columns=['suicides_no_year'])
n_suicides_year['year'] = df_us['year'].unique()

top_year = n_suicides_year.sort_values('suicides_no_year', ascending=False)['year']
top_suicides = n_suicides_year.sort_values('suicides_no_year', ascending=False)['suicides_no_year']

plt.figure(figsize=(8,5))
plt.xticks(rotation=90)
sns.barplot(x = top_year, y = top_suicides)
```



The data illustrates that the number of suicides has been increasing over the years in the United states. There was a slight constant decrease in the year 1999 and 2000 but other than that it has been on the rise gradually until 2015.

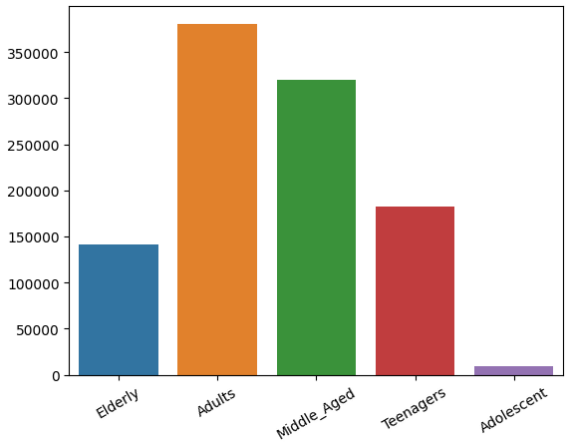
Question 2: In the United states, do adults also have higher suicide rates?

```
# Age group with the highest suicide count in the United States

suicides_no_age = []

for a in data['AgeCategory'].unique():
    suicides_no_age.append(sum(df_us[df_us['AgeCategory'] == a]['suicides_no']))

plt.xticks(rotation=30)
sns.barplot(x = df_us['AgeCategory'].unique(), y = suicides_no_age)
```



From the chart, the answer is YES. Adults in the united states have the highest suicide count and are more inclined towards suicide.

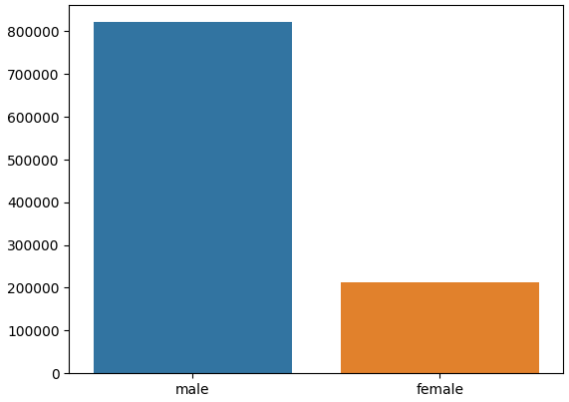
Question 3: Which gender is more inclined towards suicide in the US?

```
# Gender more inclined towards suicide in the United states

suicides_no_sex = []

for s in data['gender'].unique():
    suicides_no_sex.append(sum(df_us[df_us['gender'] == s]['suicides_no']))

sns.barplot(x = df_us['gender'].unique(), y = suicides_no_sex)
```

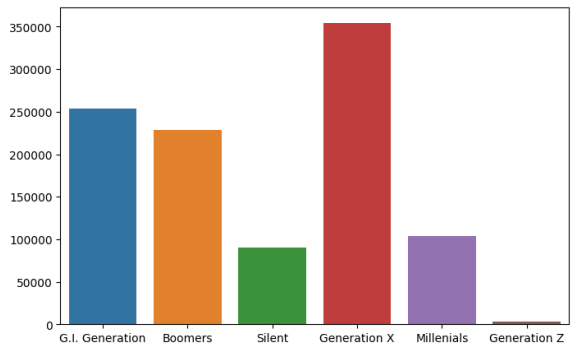


Males in the united states have the highest suicide count among both genders.

Question 4: Which generation has the highest suicide number in the US?

```
suicides_no_gen = []
for g in data['generation'].unique():
    suicides_no_gen.append(sum(df_us[df_us['generation'] == g]['suicides_no']))

plt.figure(figsize=(8,5))
sns.barplot(x = df_us['generation'].unique(), y = suicides_no_gen)
```



In the United states, Generation X had more suicide deaths than any other generation.

5.7. Correlation Analysis.

Next, I made a heat map that demonstrates the correlations between columns that have numeral values. Blue means that there is a positive correlation whereas red means that there is a negative correlation. For example, Suicide number and population have a strong positive correlation. This heat map helped me determine which columns I should analyze. Because the dark blue and dark red colors indicate that the two variables are possibly related.

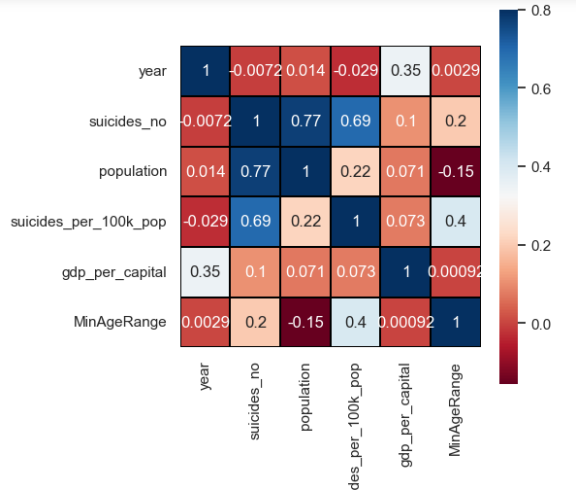


Fig20: Correlation analysis heatmap.

From the heatmap, the GDP per capital and the suicide rate have a strong negative correlation. This means that when the suicide rate increases, the GDP per capital decreases. It is proved by the scatter plot below. We can see that there most of the values are placed on the spot where GDP is high, and the suicide rate is low.

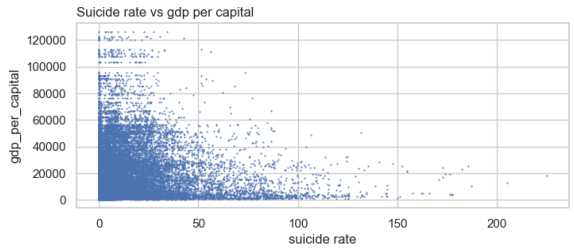


Fig21: Scatterplot of Suicide rate vs GDP per capital.

Besides using raw values, I also used the p-value means for the suicide rate and try to double-check the correlation. The p-value of the graph is about 3.02, which means that it is statistically significant. Therefore, there is a strong negative correlation between the Suicide rate and GDP.

```
# P-value for the graph
from scipy.stats import spearmanr
sns.regplot(x=data.gdp_per_capital, y=data.suicides_per_100k_pop, color='g')
print(spearmanr(data.gdp_per_capital, data.suicides_per_100k_pop))
```

```
SpearmanrResult(correlation=-0.07306331576449537, pvalue=3.0241518188246404e-34)
```

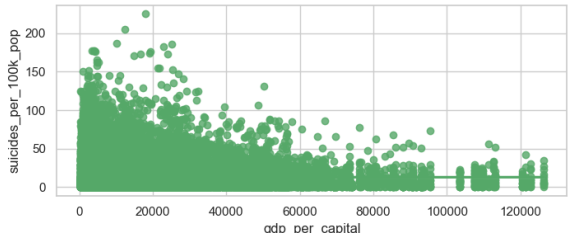


Fig22: Scatter chart showing the p value of the graph.

Also, from the heap map, there is a strong positive correlation between suicide count and population. That is, as population increase, the suicide count increase.

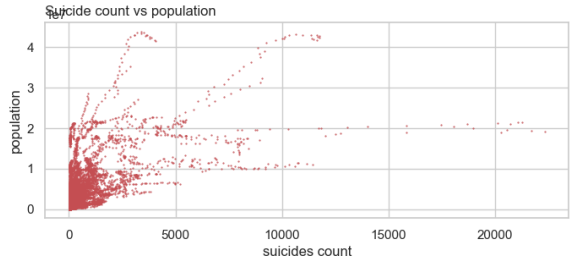


Fig23: Scatter chart showing the relationship between population and suicide count.

It seems like that there is a weak negative correlation between the two columns. In the scatter plot below, there is a negative correlation indicated.

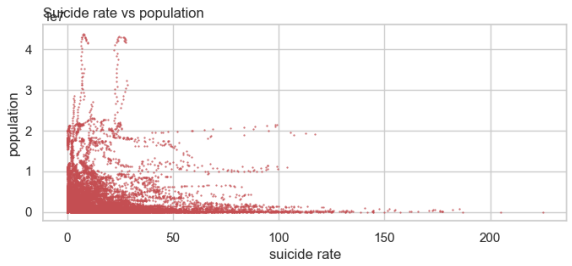


Fig24: Scatter chart showing the relationship between suicide rate and population.

6. MODELLING:

Our goal in this section is to build four different regression model all using different algorithms that will be trained to understand correlation between our features and our target variable. We want to predict Y (suicides count), given a specific year, pertaining to a specific age group & gender and possibly population.

Our model follows Supervised Learning, which consists in learning the link between two datasets: the observed data, X and an external variable y that we are trying to predict, usually called "target" or "labels". Most often, y is a 1D array of length n samples.

I will be using Scikit-learn to implement our models. All supervised estimators in scikit-learn implement a fit (X, y) method to fit the model and a predict (X) method that, given unlabeled observations X, returns the predicted labels y.

While assigning values to X, we drop some columns which we do not require, or which are less relevant to our model while predicting the output.

The models built will be compared based on accuracy of the predictions using the R-squared mean estimator. The four-regression model to be considered under this section are, linear regressor, random forest, decision tree and SVM.

The steps involved include.

Preparing the data for modelling, that is we are going to define X and y, then split the data by assigning the features to the X variable and assigning the target variable to the y variable.

Next, I am going to encode the gender column into binary output. Gender is a categorical data and we need to encode into a binary output of 1s and 0s for our model.

After defining X and y, we are going to split our new dataset into two sections. One for training and one for testing.

After splitting, we build out model and then train the model using sklearn package.

After training, we then make predictions using our model and we calculate the accuracy of each model to see how they perform.

It is good to note, that we are going to be considering 3 set of features for our models.

SET 1: age, gender, year

SET 2: age, gender, year, population

SET 3: age, gender, year, population, suicide rate.

6.1. LINEAR REGRESSION MODEL (Using SET 1 features)

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, *this regression technique finds out a linear relationship between x (input) and y(output).*

Using SET 1 as the features. Let's build our first model.

Preparing the data for modelling

```
# Next step is to prepare the data for modelling
# I will be excluding some data as there are not fit for my model, country, age, gdp_per_capital, generation and AgeCategory

model_data= data.loc[:,['year','gender','MinAgeRange', 'suicides_no']]
model_data.head(3)
```

	year	gender	MinAgeRange	suicides_no
0	1987	male	15	21
1	1987	male	35	16
2	1987	female	15	14

Next, we Define X and y.

```
# Next, i will define x (features/predictor) and y (target value)
# features = year, gender, MinAgeRange
# Target = suicides_no

X = model_data.iloc[:, :-1].values
y = model_data.iloc[:, -1].values
```

```
X.shape
```

```
(23539, 3)
```

```
y.shape
```

```
(23539,)
```

Encode the gender column to binary output.

```
# Next, i will fix the gender constraint. Gender is a categorical data. so i will encode it to binary output

import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
gen = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
X = np.array(gen.fit_transform(X))
```

```
X
```

```
array([[0.0, 1.0, 1987, 15],
       [0.0, 1.0, 1987, 35],
       [1.0, 0.0, 1987, 15],
       ...,
       [0.0, 1.0, 2014, 5],
       [1.0, 0.0, 2014, 5],
       [1.0, 0.0, 2014, 55]], dtype=object)
```

```
y
```

```
array([21, 16, 14, ..., 60, 44, 21], dtype=int64)
```

Split the training data into training and test sets

```
# Splitting the Data into training and testing sets

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state = 1)
```

```
y_train.shape
```

```
(16477,)
```

Build our linear regression model and train it.

```
# Next step is building out linear regression model and training it

from sklearn.linear_model import LinearRegression
mod = LinearRegression()
mod.fit(x_train, y_train)
```

```
LinearRegression
LinearRegression()
```

Make prediction with our model

```
# Making predictions for the suicide count based of the test data
pred = mod.predict(x_test)
print(pred)
```

```
[121.6171659  470.74018403 394.83601382 ... 394.00473902 413.48720554
 126.60481467]
```

Next, we are going to compare the actual values with the predicted values.

```
# Comparing the top 10 Actual values with the predicted values

preddt = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred, 'Difference':y_test - pred})
preddt[0:10]
```

	Actual Value	Predicted value	Difference
0	43	121.617166	-78.617166
1	8	470.740184	-462.740184
2	437	394.836014	42.163986
3	55	182.558390	-127.558390
4	2	235.654994	-233.654994
5	1074	414.318480	659.681520
6	158	180.064565	-22.064565
7	289	422.631228	-133.631228
8	7	141.099632	-134.099632
9	25	87.171753	-62.171753

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005.

```
# Making predictions for certain demographics
# for instance, predicting the suicide count for males in the age group of 55-74 in the year 2005

print(mod.predict([[1,0,2005,55]]))
```

```
[180.89584025]
```

This yields a suicide count of approximately 181.

6.2. RANDOM FOREST REGRESSOR (Using SET 1 features)

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Using SET 1 as the features. Let's build our second model.

Earlier, we have implemented all the necessary steps of defining X and y, encoding our categorical data, splitting into training and testing sets. Here we just start off by building the model and training it.

Build Random forest regression model.

```
# Let's try a different model and see how it performs - RANDOM FOREST

from sklearn.ensemble import RandomForestRegressor
mod1 = RandomForestRegressor(n_estimators=30)
mod1.fit(x_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(n_estimators=30)
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data

pred1 = mod1.predict(x_test)
print(pred1)
```

```
[ 83.65711647 538.29061782 448.52704771 ... 345.36763684 666.41139515
 111.68394745]
```

Compare actual value with predicted values

```
# Comparing the top 10 Actual values with the predicted values

predt1 = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred1, 'Difference':y_test - pred1})
predt1[0:10]
```

	Actual Value	Predicted value	Difference
0	43	83.657116	-40.657116
1	8	538.290618	-530.290618
2	437	448.527048	-11.527048
3	55	171.329828	-116.329828
4	2	135.534862	-133.534862
5	1074	808.363827	265.636173
6	158	208.085602	-50.085602
7	289	1144.550981	-855.550981
8	7	187.190042	-180.190042
9	25	78.536319	-53.536319

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005.

```
# Making predictions for certain demographics
# for instance, predicting the suicide count for males in the age group of 55-74 in the year 2005

print(mod1.predict([[1,0,2005,55]]))
```

```
[260.59858738]
```

This yields a suicide count of approximately 261.

6.3. DECISION TREE REGRESSION MODEL (Using SET1 features).

Decision Tree is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all their possible results, including outcomes, input costs and utility.

Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables. We can see that if the maximum depth of the tree (controlled by the max depth parameter) is set too high, the decision trees learn too fine details of the training data and learn from the noise, i.e. they overfit.

Build decision tree regression model.

```
# Let's try a different model DECISION TREE

from sklearn.tree import DecisionTreeRegressor
mod2 = DecisionTreeRegressor()
mod2.fit(x_train, y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor()
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data

pred2 = mod2.predict(x_test)
print(pred2)
```

```
[ 91.72340426 517.6097561  452.98275862 ... 339.14516129 658.03508772
 104.12195122]
```

Compare actual values with predicted values

```
# Comparing the top 10 Actual values with the predicted values

predt2 = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred2, 'Difference':y_test - pred2})
predt2[0:10]
```

	Actual Value	Predicted value	Difference
0	43	91.723404	-48.723404
1	8	517.609756	-509.609756
2	437	452.982759	-15.982759
3	55	177.088889	-122.088889
4	2	145.205128	-143.205128
5	1074	843.211538	230.788462
6	158	208.686275	-50.686275
7	289	1184.461538	-895.461538
8	7	183.735849	-176.735849
9	25	87.448980	-62.448980

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005.

```
# Making predictions for certain demographics
# for instance, predicting the suicide count for a males in the age group of 55-74 in the year 2005

print(mod2.predict([[1,0,2005,55]]))
```

```
[260.23913043]
```

This yields a suicide count of approximately 260.

6.4. SUPPORT VECTOR REGRESSION MODEL (Using SET1 features).

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

Build Support vector regression model.

```
# Let's try a different model SVR

from sklearn.svm import SVR
mod3 = SVR(kernel='rbf')
mod3.fit(x_train, y_train)
```

```
SVR()
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data

pred3 = mod3.predict(x_test)
print(pred3)
```

```
[42.02261208 43.10199272 42.02347598 ... 42.02337783 42.38231173
 42.02319169]
```

Compare actual values with predicted values

```
# Comparing the top 10 Actual values with the predicted values

predt3 = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred3, 'Difference':y_test - pred3})
predt3[0:10]
```

	Actual Value	Predicted value	Difference
0	43	42.022612	0.977388
1	8	43.101993	-35.101993
2	437	42.023476	394.976524
3	55	43.099384	11.900616
4	2	43.817520	-41.817520
5	1074	42.382413	1031.617587
6	158	43.099086	114.900914
7	289	42.383404	246.616596
8	7	42.381564	-35.381564
9	25	41.661888	-16.661888

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005.

```
# Making predictions for certain demographics
# for instance, predicting the suicide count for males in the age group of 55-74 in the year

print(mod3.predict([[1,0,2005,55]]))
```

```
[43.0991864]
```

This yields a suicide count of approximately 43.

6.5. LINEAR REGRESSION MODEL (Using SET 2 features)

Using SET 2 as the features. Let's build our first model.

Prepare data for modelling

```
# Next step is to prepare the data for modelling
# I will be excluding some data as there are not fit for my model, country, age, gdp_per_capital, generation and AgeCategory

model_data= data.loc[:,['year','gender','MinAgeRange', 'population', 'suicides_no']]
model_data.head(3)
```

	year	gender	MinAgeRange	population	suicides_no
0	1987	male	15	312900	21
1	1987	male	35	308000	16
2	1987	female	15	289700	14

Next, we Define X and y.

```
# Next, i will define x (features/predictor) and y (target value)
# features = year, gender, MinAgeRange, population
# Target = suicides_no

X = model_data.iloc[:, :-1].values
y = model_data.iloc[:, -1].values
```

Encode the gender column to binary output.

```
# Next, i will fix the gender constraint. Gender is a categorical data. so i will encode it to binary output

import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
gen = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
X = np.array(gen.fit_transform(X))
```

```
X
```

```
array([[0.0, 1.0, 1987, 15, 312900],
       [0.0, 1.0, 1987, 35, 308000],
       [1.0, 0.0, 1987, 15, 289700],
       ...,
       [0.0, 1.0, 2014, 5, 2762158],
       [1.0, 0.0, 2014, 5, 2631600],
       [1.0, 0.0, 2014, 55, 1438935]], dtype=object)
```

Split the data into training and test sets

```
# Splitting the Data into training and testing sets

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state = 1)
```

```
y_train.shape
```

```
(16477,)
```

Build Linear regression model.

```
# Next step is building out linear regression model and training it

from sklearn.linear_model import LinearRegression
mod = LinearRegression()
mod.fit(x_train, y_train)
```

```
LinearRegression()
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data
pred = mod.predict(x_test)
print(pred)
```

```
[-198.74579303  236.25907219  404.82471437 ...  839.26586665  251.06256375
 -154.4931808 ]
```

Compare actual values with predicted values

```
# Comparing the top 10 Actual values with the predicted values

preddt = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred, 'Difference':y_test - pred})
preddt[0:10]
```

	Actual Value	Predicted value	Difference
0	43	-198.745793	241.745793
1	8	236.259072	-228.259072
2	437	404.824714	32.175286
3	55	553.015270	-498.015270
4	2	19.421699	-17.421699
5	1074	415.262791	658.737209
6	158	22.388874	135.611126
7	289	237.741402	51.258598
8	7	-179.229126	186.229126
9	25	-213.110400	238.110400

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005 in a population of 321900.

```
# Making predictions for certain demographics
# for instance, predicting the suicide count for males in the age group of 55-74 in the year 2005
# with a population of 312900

print(mod.predict([[1,0,2005,55, 312900]]))
```

```
[-63.5132821]
```

This yields a suicide count of approximately -64.

6.6. RANDOM FOREST REGRESSOR (Using SET 2 features)

Build Random forest regression model.

```
# Let's try a different model and see how it performs - RANDOM FOREST

from sklearn.ensemble import RandomForestRegressor
mod1 = RandomForestRegressor(n_estimators=30)
mod1.fit(x_train, y_train)
```

```
RandomForestRegressor(n_estimators=30)
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data

pred1 = mod1.predict(x_test)
print(pred1)
```

```
[ 20.6          3.56666667 416.73333333 ... 637.53333333 199.96666667
  55.1        ]
```

Compare actual values with predicted values

```
# Comparing the top 10 Actual values with the predicted values

predt1 = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred1, 'Difference':y_test - pred1})
predt1[0:10]
```

	Actual Value	Predicted value	Difference
0	43	20.600000	22.400000
1	8	3.566667	4.433333
2	437	416.733333	20.266667
3	55	168.366667	-113.366667
4	2	3.900000	-1.900000
5	1074	947.333333	126.666667
6	158	155.300000	2.700000
7	289	222.233333	66.766667
8	7	11.566667	-4.566667
9	25	27.133333	-2.133333

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005 in a population of 321900.

```
# Making predictions for certain demographics
# for instance, predicting the suicide count for males in the age group of 55-74 in the year 2005
# in a population of 312900

print(mod1.predict([[1,0,2005,55, 312900]]))
```

```
[20.1]
```

This yields a suicide count of approximately 20.

6.7. DECISION TREE REGRESSION MODEL (Using SET 2 features).

Build Decision tree regression model.

```
# Let's try a different model DECISION TREE

from sklearn.tree import DecisionTreeRegressor
mod2 = DecisionTreeRegressor()
mod2.fit(x_train, y_train)
```

```
DecisionTreeRegressor()
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data

pred2 = mod2.predict(x_test)
print(pred2)
```

```
[ 29.   4. 410. ... 707. 219.  63.]
```

Compare actual values with predicted values

```
# Comparing the top 10 Actual values with the predicted values

predt2 = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred2, 'Difference':y_test - pred2})
predt2[0:10]
```

	Actual Value	Predicted value	Difference
0	43	29.0	14.0
1	8	4.0	4.0
2	437	410.0	27.0
3	55	175.0	-120.0
4	2	1.0	1.0
5	1074	1062.0	12.0
6	158	91.0	67.0
7	289	47.0	242.0
8	7	9.0	-2.0
9	25	19.0	6.0

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005 in a population of 321900.

```
# Making predictions for ceratin demographics
# for instance, predicting the suicide count for a males in the age group of 55-74 in the year 2005
# in a population of 312900

print(mod2.predict([[1,0,2005,55,312900]]))
```

```
[7.]
```

This yields a suicide count of approximately 7.

6.8. SUPPORT VECTOR REGRESSION MODEL (Using SET 2 features).

Build Support vector regression model.

```
# Let's try a different model SVR

from sklearn.svm import SVR
mod3 = SVR(kernel='rbf')
mod3.fit(x_train, y_train)
```

```
▾ SVR
SVR()
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data

pred3 = mod3.predict(x_test)
print(pred3)
```

```
[ 22.32560831   7.36591672 158.46942574 ... 361.81404083  59.2454816
  35.24240108]
```

Compare actual values with predicted values

```
# Comparing the top 10 Actual values with the predicted values

predt3 = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred3, 'Difference':y_test - pred3})
predt3[0:10]
```

	Actual Value	Predicted value	Difference
0	43	22.325608	20.674392
1	8	7.365917	0.634083
2	437	158.469426	278.530574
3	55	313.609164	-258.609164
4	2	8.807030	-6.807030
5	1074	139.748742	934.251258
6	158	54.938223	103.061777
7	289	46.277282	242.722718
8	7	13.613733	-6.613733
9	25	44.834070	-19.834070

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005 in a population of 321900.

```
# Making predictions for certain demographics
# for instance, predicting the suicide count for males in the age group of 55-74 in the year 2005
# in a population of 312900

print(mod3.predict([[1,0,2005,55, 312900]]))
```

```
[20.72727195]
```

This yields a suicide count of approximately 21.

6.9. LINEAR REGRESSION MODEL (Using SET 3 features)

Using SET 3 as the features. Let's build our first model.

Prepare data for modelling

```
# Next step is to prepare the data for modelling
# I will be excluding some data as there are not fit for my model, country, age, gdp_per_capital, generation and A

model_data= data.loc[:,['year','gender','MinAgeRange', 'population', 'suicide_rate', 'suicides_no']]
model_data.head(3)
```

	year	gender	MinAgeRange	population	suicide_rate	suicides_no
0	1987	male	15	312900	6.71	21
1	1987	male	35	308000	5.19	16
2	1987	female	15	289700	4.83	14

Next, we Define X and y

```
# Next, i will define x (features/predictor) and y (target value)
# features = year, gender, MinAgeRange, population
# Target = suicides_no

X = model_data.iloc[:, :-1].values
y = model_data.iloc[:, -1].values
```

Encode the gender column to binary output.

```
# Next, i will fix the gender constraint. Gender is a categorical data. so i will encode it to bin

import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
gen = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
X = np.array(gen.fit_transform(X))
```

```
X
```

```
array([[0.0, 1.0, 1987, 15, 312900, 6.71],
       [0.0, 1.0, 1987, 35, 308000, 5.19],
       [1.0, 0.0, 1987, 15, 289700, 4.83],
       ...,
       [0.0, 1.0, 2014, 5, 2762158, 2.17],
       [1.0, 0.0, 2014, 5, 2631600, 1.67],
       [1.0, 0.0, 2014, 55, 1438935, 1.46]], dtype=object)
```

```
y
```

```
array([21, 16, 14, ..., 60, 44, 21], dtype=int64)
```

Split the data into training and test sets

```
# Splitting the Data into training and testing sets

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state = 1)
```

```
y_train.shape
```

```
(16477,)
```

Build Linear regression model.

```
# Next step is building out linear regression model and training it

from sklearn.linear_model import LinearRegression
mod = LinearRegression()
mod.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data
pred = mod.predict(x_test)
print(pred)
```

```
[-62.00237875 434.62754203 409.70610443 ... 755.93358921 355.37429892
 -56.97956094]
```

Compare actual values with predicted values

```
# Comparing the top 10 Actual values with the predicted values

preddt = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred, 'Difference':y_test - pred})
preddt[0:10]
```

	Actual Value	Predicted value	Difference
0	43	-62.002379	105.002379
1	8	434.627542	-426.627542
2	437	409.706104	27.293896
3	55	371.763300	-316.763300
4	2	-243.442325	245.442325
5	1074	870.834005	203.165995
6	158	94.487522	63.512478
7	289	463.199771	-174.199771
8	7	-212.733281	219.733281
9	25	-135.964766	160.964766

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005 in a population of 321900 with a suicide rate of 5.40

```
# Making predictions for certain demographics
# for instance, predicting the suicide count for males in the age group of 55-74 in the year 2005
# with a population of 312900 and a suicide rate of 5.40

print(mod.predict([[1,0,2005,55, 312900, 5.40]]))
```

```
[-174.57650626]
```

This yields a suicide count of approximately -175.

6.10.RANDOM FOREST REGRESSOR (Using SET 3 features)

Build Random forest regression model.

```
# Let's try a different model and see how it performs - RANDOM FOREST

from sklearn.ensemble import RandomForestRegressor
mod1 = RandomForestRegressor(n_estimators=30)
mod1.fit(x_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(n_estimators=30)
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data

pred1 = mod1.predict(x_test)
print(pred1)
```

```
[ 42.76666667   8.63333333 426.03333333 ... 697.4        286.9
  61.9        ]
```

Compare actual values with predicted values

```
# Comparing the top 10 Actual values with the predicted values

predt1 = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred1, 'Difference':y_test - pred1})
predt1[0:10]
```

	Actual Value	Predicted value	Difference
0	43	42.766667	0.233333
1	8	8.633333	-0.633333
2	437	426.033333	10.966667
3	55	55.733333	-0.733333
4	2	2.066667	-0.066667
5	1074	1111.366667	-37.366667
6	158	160.633333	-2.633333
7	289	290.200000	-1.200000
8	7	6.800000	0.200000
9	25	24.500000	0.500000

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005 in a population of 321900 with a suicide rate of 5.40

```
# Making predictions for certain demographics
# for instance, predicting the suicide count for males in the age group of 55-74 in the year 2005
# in a population of 312900 and a suicide rate of 5.40

print(mod1.predict([[1,0,2005,55, 312900, 5.40]]))
```

```
[16.7]
```

This yields a suicide count of approximately 17.

6.11.DECISION TREE REGRESSION MODEL (Using SET 3 features).

Build Decision tree regression model.

```
# Let's try a different model DECISION TREE

from sklearn.tree import DecisionTreeRegressor
mod2 = DecisionTreeRegressor()
mod2.fit(x_train, y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor()
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data

pred2 = mod2.predict(x_test)
print(pred2)
```

```
[ 42.   9. 427. ... 707. 265.  61.]
```

Compare actual values with predicted values

```
# Comparing the top 10 Actual values with the predicted values

predt2 = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred2, 'Difference':y_test - pred2})
predt2[0:10]
```

	Actual Value	Predicted value	Difference
0	43	42.0	1.0
1	8	9.0	-1.0
2	437	427.0	10.0
3	55	58.0	-3.0
4	2	2.0	0.0
5	1074	1062.0	12.0
6	158	169.0	-11.0
7	289	306.0	-17.0
8	7	7.0	0.0
9	25	26.0	-1.0

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005 in a population of 321900 with a suicide rate of 5.40

```
# Making predictions for ceratin demographics
# for instance, predicting the suicide count for a males in the age group of 55-74 in the year 2005
# in a population of 312900 and a suicide rate of 5.40

print(mod2.predict([[1,0,2005,55,312900, 5.40]]))
```

```
[17.]
```

This yields a suicide count of approximately 17.

6.12.SUPPORT VECTOR REGRESSION MODEL (Using SET 3 features).

Build Support vector regression model.

```
# Let's try a different model SVR

from sklearn.svm import SVR
mod3 = SVR(kernel='rbf')
mod3.fit(x_train, y_train)
```

```
▾ SVR
SVR()
```

Make predictions with our model

```
# Making predictions for the suicide count based of the test data

pred3 = mod3.predict(x_test)
print(pred3)
```

```
[ 22.3029167    7.38116302 158.23553509 ... 361.93381459  59.12857443
  35.18544094]
```

Compare actual values with predicted values

```
# Comparing the top 10 Actual values with the predicted values

predt3 = pd.DataFrame({'Actual Value':y_test, 'Predicted value':pred3, 'Difference':y_test - pred3})
predt3[0:10]
```

	Actual Value	Predicted value	Difference
0	43	22.302917	20.697083
1	8	7.381163	0.618837
2	437	158.235535	278.764465
3	55	313.643051	-258.643051
4	2	8.818871	-6.818871
5	1074	139.517467	934.482533
6	158	54.831473	103.168527
7	289	46.191835	242.808165
8	7	13.613711	-6.613711
9	25	44.752274	-19.752274

Test the model by making predictions based on certain demographics, for instance predicting the suicide count of males of age group 55 in the year 2005 in a population of 321900 with a suicide rate of 5.40

```
# Making predictions for certain demographics
# for instance, predicting the suicide count for males in the age group of 55-74 in the year 2005
# in a population of 312900 and a suicide rate of 5.40

print(mod3.predict([[1,0,2005,55, 312900, 5.40]]))
```

```
[20.70880376]
```

This yields a suicide count of approximately 21.

In the next section, I am going to check how each model perform in each set. I will be using the R-squared score as the measure of performance and I will compare each model using graphs and charts.

7. RESULTS.

7.1. Performance Evaluation.

There are various metrics that can be used to evaluate the performance of a Regression model. In this section, we will use the R-Squared score value. R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

The definition of R-squared is straight-forward; it is the percentage of the response variable variation that is explained by a linear model. 0% indicates that the model explains none of the variability of the response data around its mean. 100% indicates that the model explains all the variability of the response data around its mean.

7.2. **Results when using the features from SET 1**.

The Linear regression model had an r2 score of 2.4%

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred)
```

```
0.02424167488701512
```

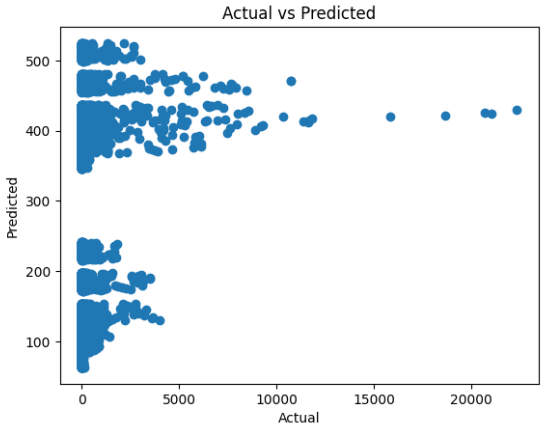


Fig25: Plot for the linear regression model using set 1 features

The Random forest regression model had an r2 score of 3.3%.

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred1)
```

```
0.0328237896215563
```

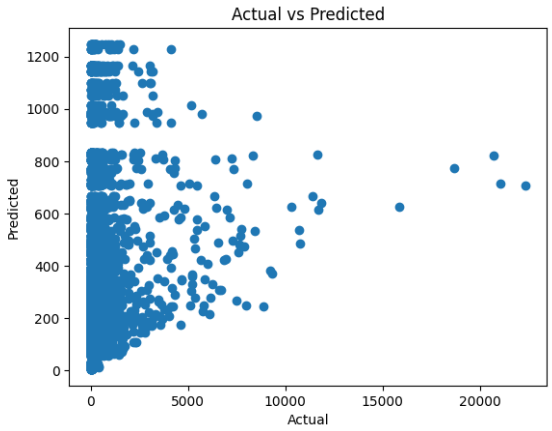


Fig26: Plot for the Random forest regression model using set 1 features

The Decision tree regression model had an r score value of 3.4%

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred2)
```

```
0.033586924000516216
```

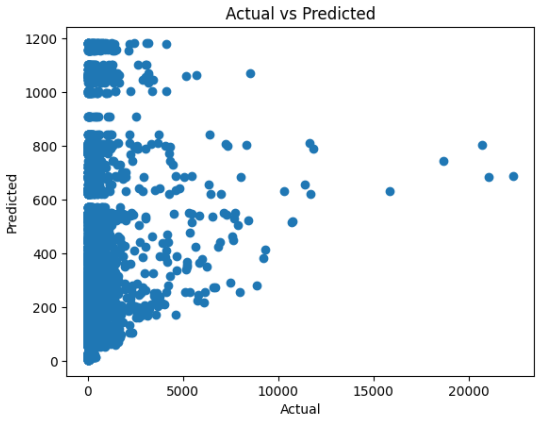


Fig27: Plot for the Decision tree regression model using set 1 features.

The Support vector regression model had an r2 score of -6.1%

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred3)
```

```
-0.061052464329653944
```

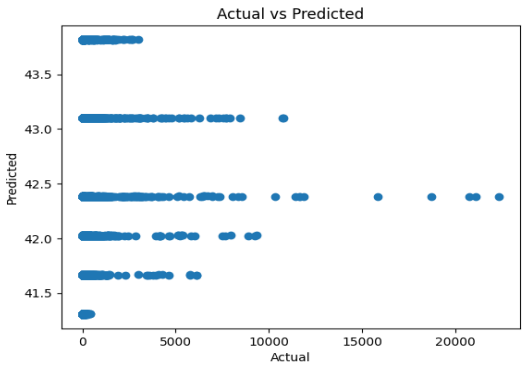


Fig28: Plot for the Support vector regression model using set 1 features.

models	R-squared score
Linear regression	**0.024**
Random forest	**0.033**
Decision tree	**0.034**
Support vector	**-0.061**

7.3. Results when using the features from SET 2 features

The Linear regression model had an r2 score of 41%

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred)
```

```
0.40776942045947917
```

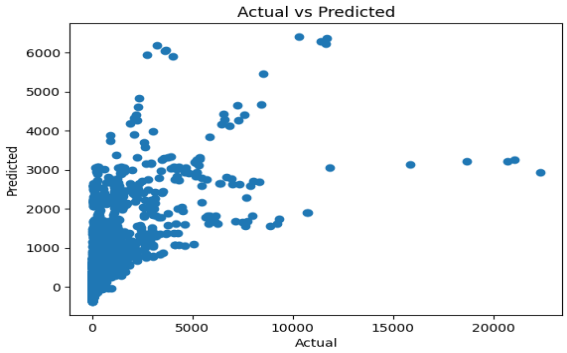


Fig29: Plot for the Linear regression model using set 2 features.

The Random forest regression model had an r2 score of 87%.

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred1)
```

```
0.871810784720337
```

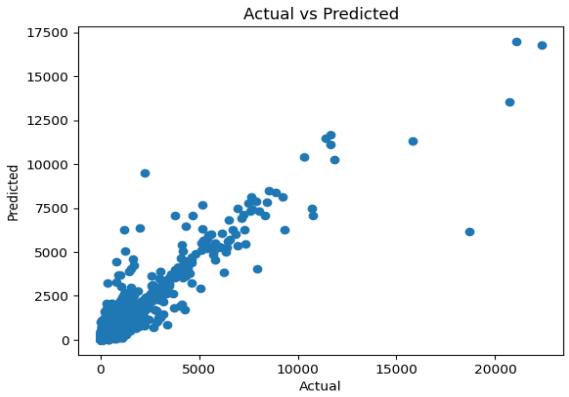


Fig30: Plot for the Random forest regression model using set 2 features.

The Decision tree regression model had an r2 score of 81%

```
# Using R Squared to check the accuracy of the mode

from sklearn.metrics import r2_score
r2_score(y_test, pred2)
```

```
0.8067654422836746
```

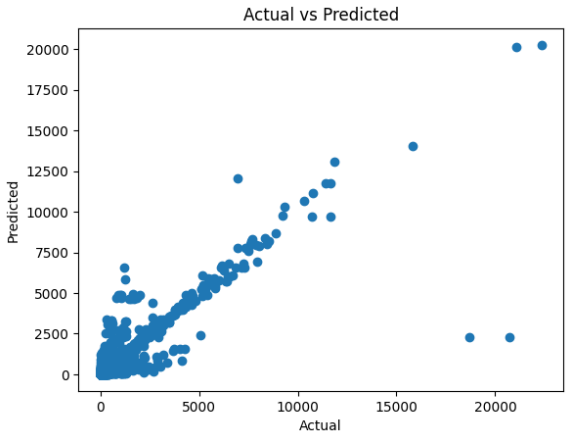


Fig31: Plot for the Decision tree regression model using set 2 features

The Support vector model had an r2 score of 8.9%

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred3)
```

```
0.0899039145528352
```

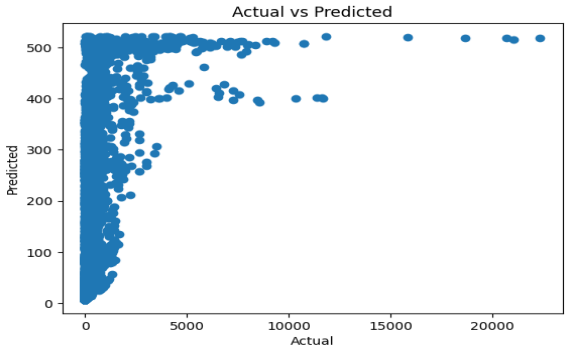


Fig32: Plot for the Support vector regression model using set 2 features.

models	R-squared score
Linear regression	**0.41**
Random forest	**0.87**
Decision tree	**0.81**
Support vector	**0.089**

7.4. **Results when using the features from SET 3**.

The Linear regression model had an r2 score of 47%

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred)
```

```
0.4672362146431194
```

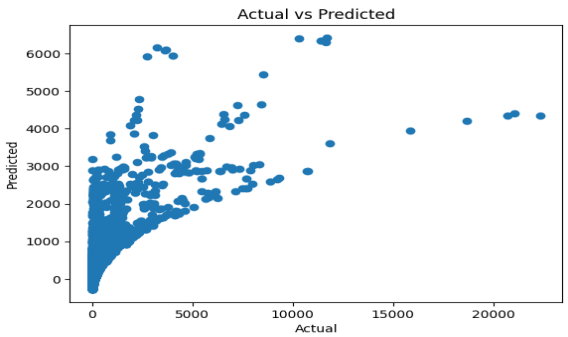


Fig33: Plot for the Linear regression model using set 3 features.

The Random forest regression model had an r2 score of 99%

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred1)
```

```
0.9978501703801682
```

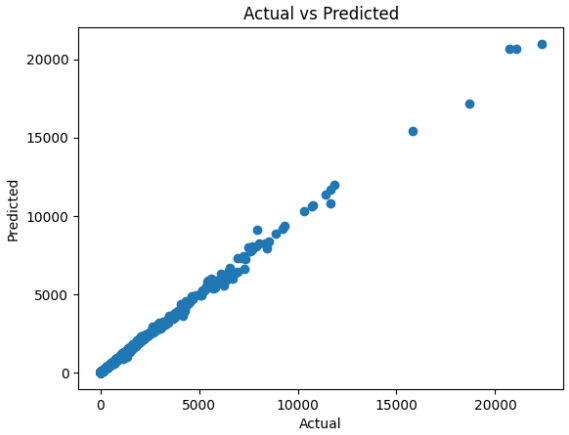


Fig34: Plot for the Random forest regression model using set 3 features.

The Decision tree regression model had an r2 score of 98%

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred2)
```

```
0.9882948416170059
```

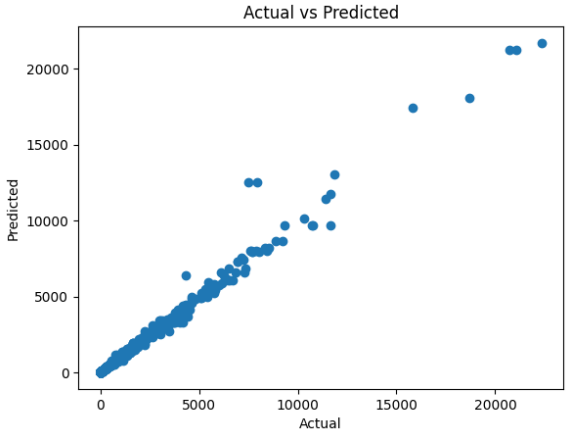


Fig35: Plot for the Decision tree regression model using set 3 features

The Support vector model had an r2 score of 8.9%

```
# Using R Squared to check the accuracy of the model

from sklearn.metrics import r2_score
r2_score(y_test, pred3)
```

```
0.08998332455163738
```

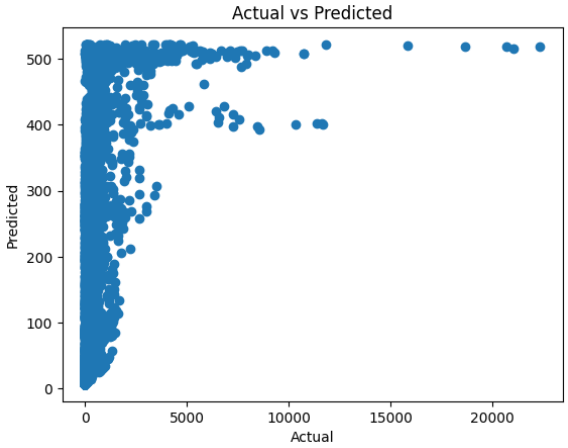


Fig36: Plot for the Support vector regression model using set 3 features.

models	**R-squared score**
Linear regression	**0.47**
Random forest	**0.99**
Decision tree	**0.98**
Support vector	**0.089**

8. CONCLUSION:

From the exploratory data analysis, I was able to draw the following conclusions.

- There was a decrease in suicides toward the 80's. This could be due to awareness of suicide & mental health in the 80s, as well as improved recognition of those at risk. But shortly after that there is a rise in suicides that we are seeing.
- The data illustrates that middle-aged adults, between the ages of 34 through 60, have the highest suicide count.
- It's evident that males are more inclined to suicide, than females.
- The data further illustrates that older generations are more inclined towards suicide.

- In the United states, according to the data, there was a steady rise in the suicide numbers over the years.
- Unlike the general data that says middle-aged adults has the highest suicide count, in the United states it is quite different, adults (25-34 years) tend to commit more suicide than middle aged adults.
- Males in the United states are more inclined towards suicide than females.
- Generation X had more suicide deaths in the United states than any other generations.

From the modelling and predictive analysis, taking into consideration the features in set 2 and set 3, it is very evident that the Random forest regression model performed the best. The r2 scores were compared and the r2 score was about 87% accurate for set 2 features and about 99% accurate for set 3 features. The accuracy of the model can be further improved by Backward Elimination. It is a stepwise regression approach, that begins with a full (saturated) model and at each step gradually eliminates variables from the regression model to find a reduced model that best explains the data.

In General, Machine learning has strong potential for improving estimation of future suicide risk and for monitoring changes in this risk over time; however, important challenges remain before this benefit can be realized clinically. Further research must address persistent methodological issues by incorporating novel methods for addressing data imbalance and overfitting and understanding factors that affect generalizability across samples and settings. Expanding the richness of the input data, leveraging newer analytic approaches, and developing automatic learning systems offer strong promise for both improving predictive ability and adjusting risk estimations over time. As important as pure predictive ability, we need to explore the best ways to represent risk to the clinician, so it is easily interpretable, actionable, and minimizes alert fatigue.