SUICIDE RATE PREDICTION

Predictive Modelling Case Study

Abstract

This project aims to predict the suicide rate in a country based on features such as population, generation, year, age group, gender, GDP etc. and find the most prominent of those features

Ojasvi Jain(J022)
Sidh Satam(J042)

Dataset Selection

The dataset chosen by us is based on the number of Suicides in different countries, from 1985 through 2016. It also consists of various factors like GDP, population, etc. which may impact the suicide rate. The link to the dataset is mentioned below.
https://www.kaggle.com/russellyates88/suicide-rates-overview-1985-to-2016#master.csv
Since the number of suicides is such a pressing issue, it is imperative to understand the factors that may be relevant in cause of the same. Through this case study, we hope to identify significant factors and also, predict the number of suicides using regression algorithms.

	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

Exploring the Data

Before we perform a regression analysis, we must perform Exploratory Data Analysis to understand what the data is trying to tell us.

- ## Descriptive Statistics

  Getting a general idea of the mean and standard deviation of the columns gives us an idea of the distribution of the data.

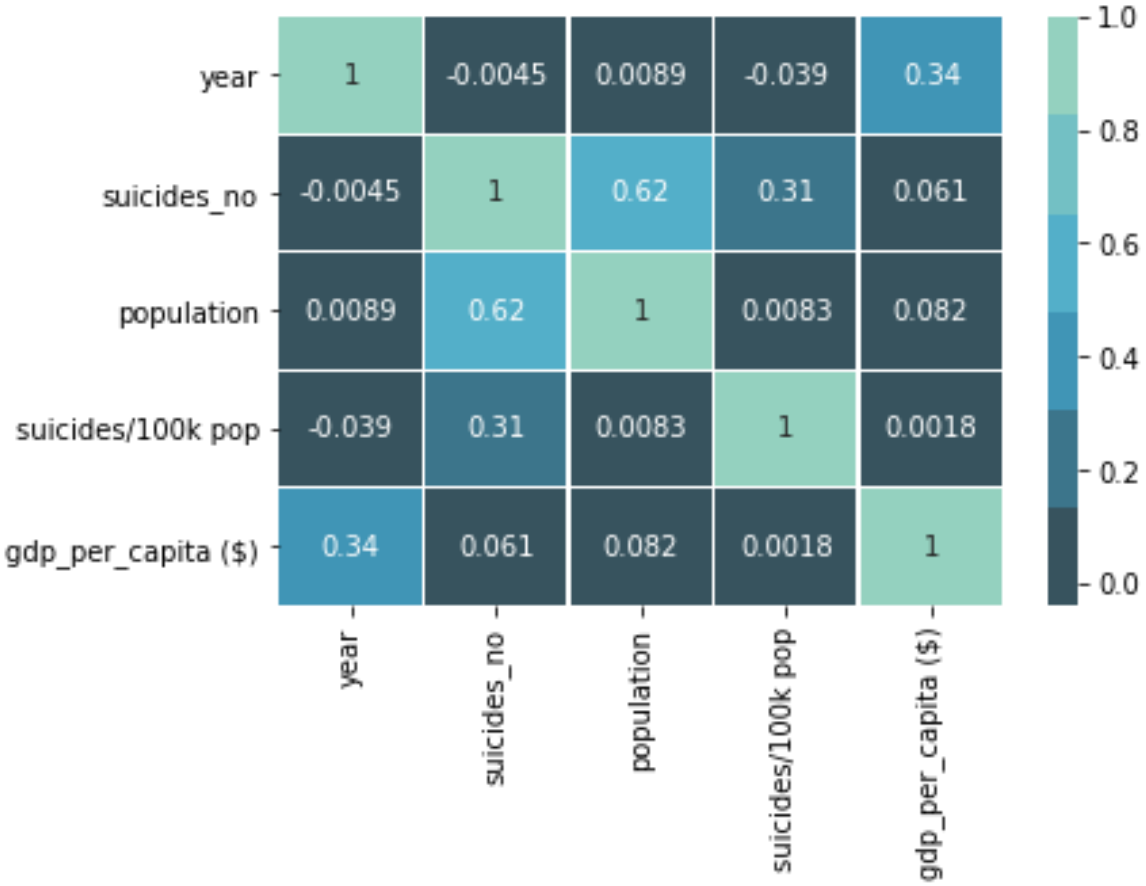
```
df.describe()
```

	year	suicides_no	population	suicides/100k pop	gdp_per_capita ($)
count	27820.000000	27820.000000	2.782000e+04	27820.000000	27820.000000
mean	2001.258375	242.574407	1.844794e+06	12.816097	16866.464414
std	8.469055	902.047917	3.911779e+06	18.961511	18887.576472
min	1985.000000	0.000000	2.780000e+02	0.000000	251.000000
25%	1995.000000	3.000000	9.749850e+04	0.920000	3447.000000
50%	2002.000000	25.000000	4.301500e+05	5.990000	9372.000000
75%	2008.000000	131.000000	1.486143e+06	16.620000	24874.000000
max	2016.000000	22338.000000	4.380521e+07	224.970000	126352.000000

- Pearson Correlation Heatmap

Pearson Correlation tells us how closely the variables are related to each other.
The heatmap allows for an easy visualization.

```
sns.heatmap(df.corr(),cmap=sns.color_palette("GnBu_d"),annot=True,
      linewidths=0.25)
plt.show()
```

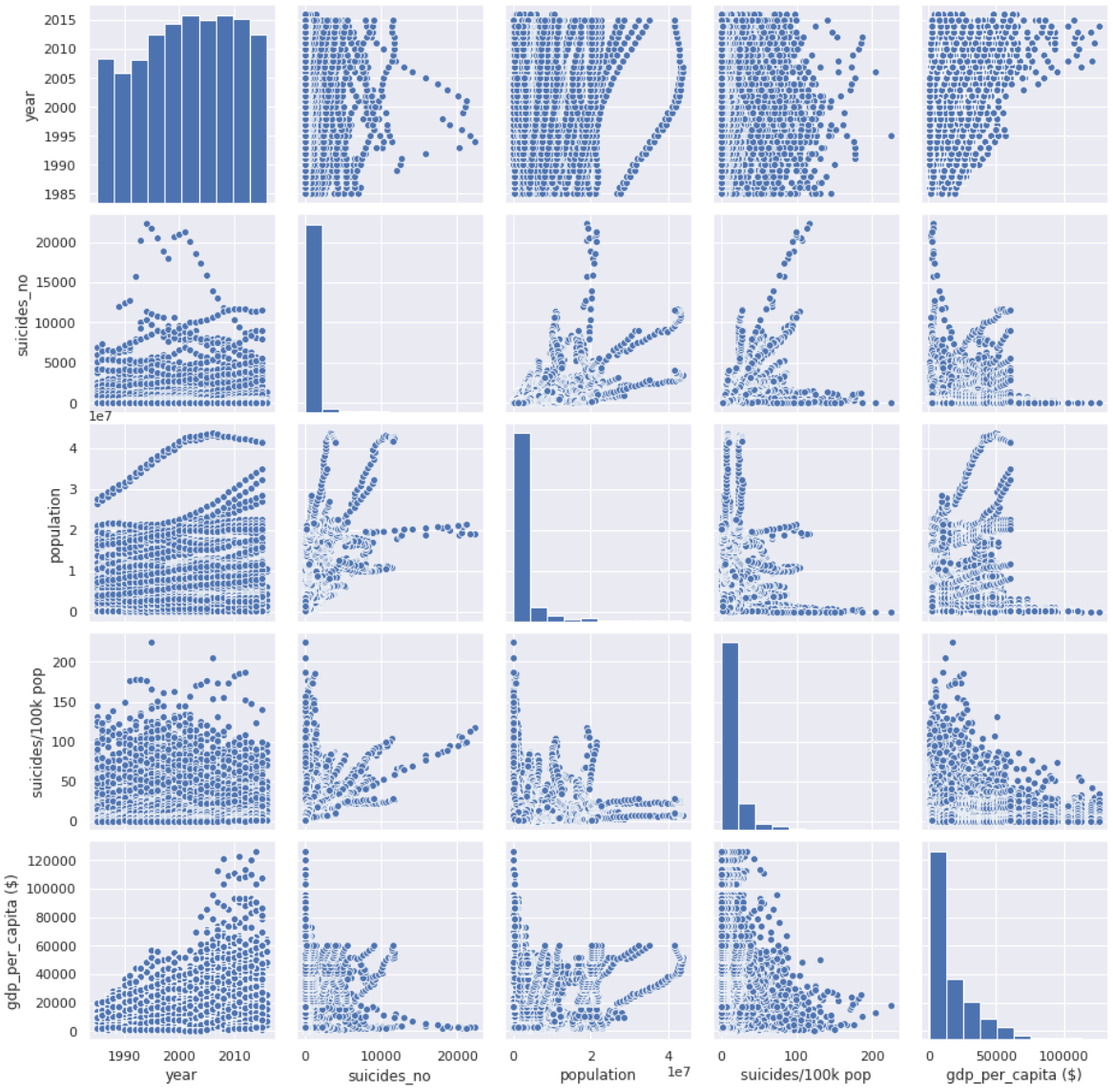


As can be seen, there is a relatively strong positive correlation between population and the number of suicides per year. Hence more the population of a country, generally more is the number of suicides.

- Pairs plot

Pairs plots are used to plot every pair of variables in the dataset as well as show the distribution of the individual variables.
A pairs plot allows us to see both the distribution of single variables and relationships between two variables.

```
sns.set()
sns.pairplot(df)
plt.show()
```

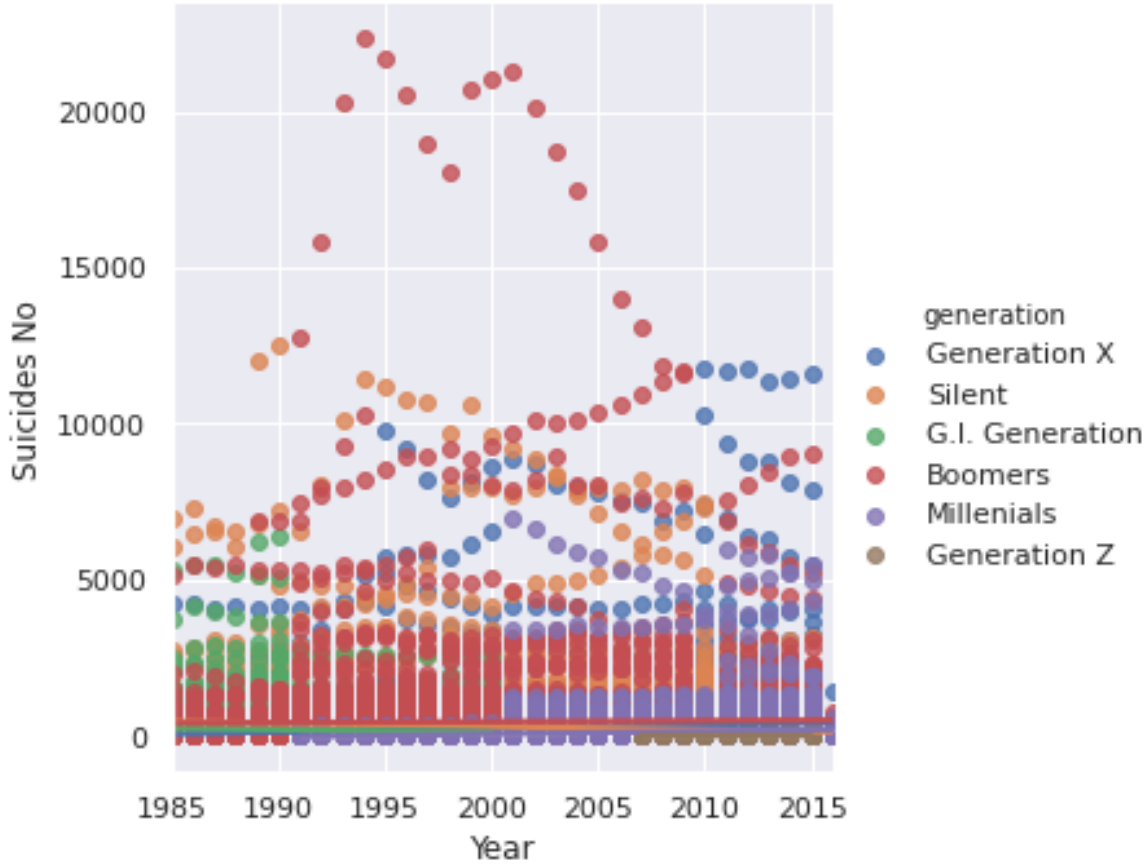


Similar to the correlation heatmap we observe a general rise in the number of suicides in a year as the population rises. No other distinct valuable insights are found.

- Number of Suicides vs Year Segregated by Generation

Lmplot is used to visualize a linear relationship as determined through regression.
It draws a scatterplot of the variables(no of suicides and year), and then fits the regression model Suicides ~ year and plot the resulting regression line and a 95% confidence interval for that regression.
The data is segregated by Generation to see different results for the different generations.

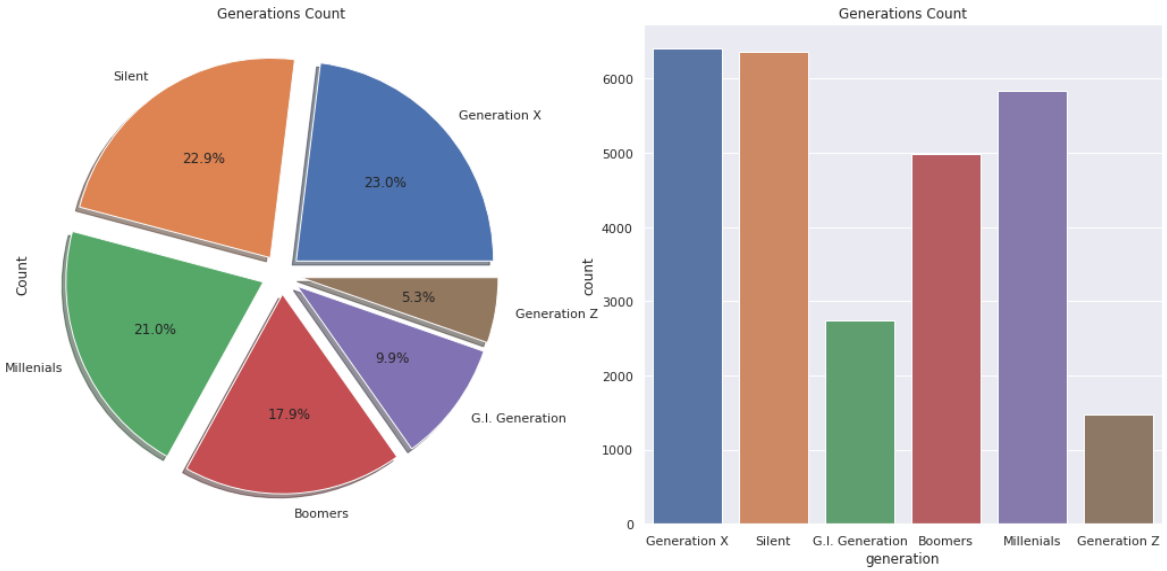
```
g = sns.lmplot(x="year", y="suicides_no", hue="generation",
               truncate=True, height=5, data=df)
g.set_axis_labels("Year", "Suicides No")
plt.show()
```



As seen by the output, even though the number of suicides has some variance, especially for the Boomer Generation and somewhat for Silent Generation and Gen X, the regression line plotted is nearly horizontal and similar for all generations, we can say that Generation does not have a significant impact in the prediction of suicides vs years.

- Descriptive Statistics for Generation

```
f, ax=plt.subplots(1, 2, figsize=(18, 8))
df['generation'].value_counts().plot.pie(explode=
                                    [0.1,0.1,0.1,0.1,0.1,0.1],
                                     autopct='%1.1f%%',
                                  ax=ax[0], shadow=True)
ax[0].set_title('Generations Count')
ax[0].set_ylabel('Count')
sns.countplot('generation', data=df, ax=ax[1])
ax[1].set_title('Generations Count')
plt.show()
```

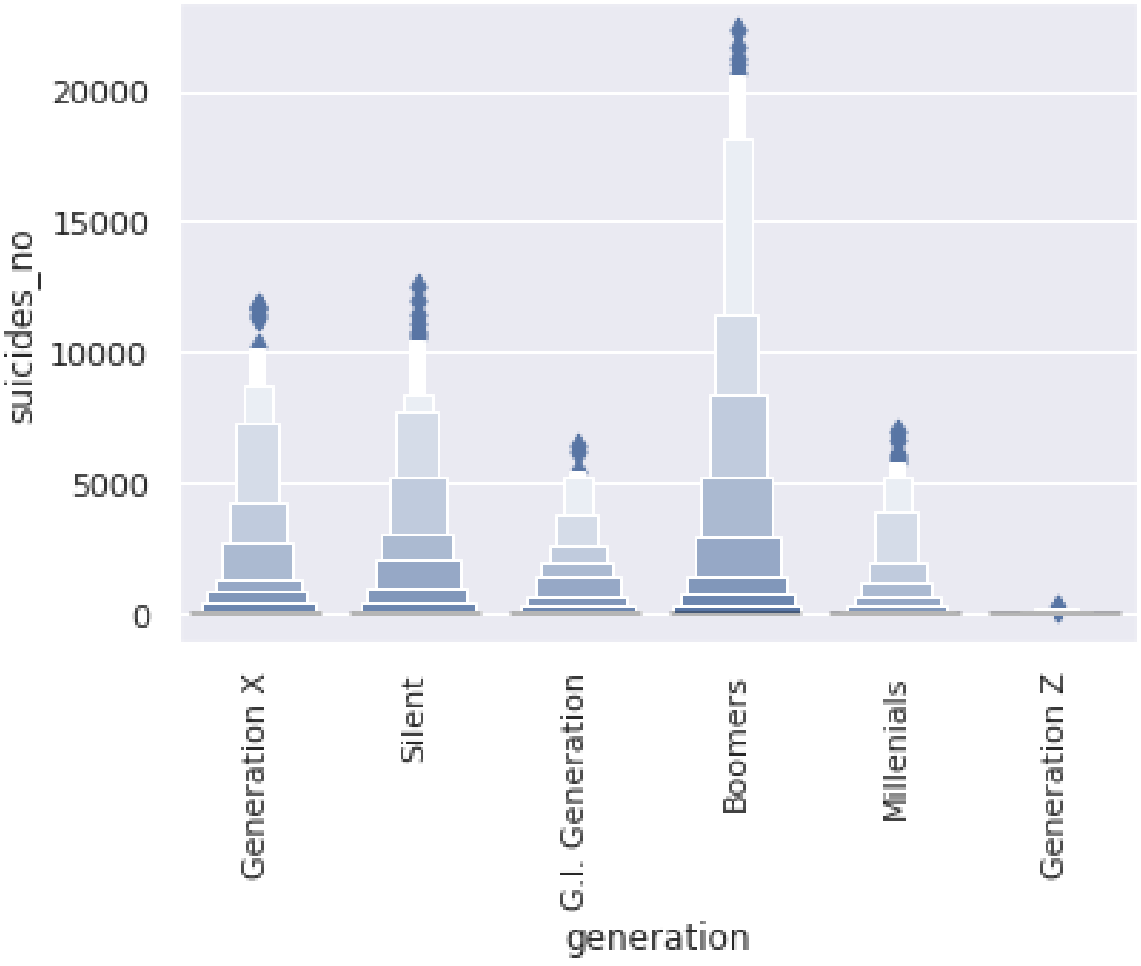


We can see the distribution of data across the various Generations.
As can be seen, the majority of the data is taken up by Gen X, Silent and Millenials with equal proportions and Gen Z has the least number of observations.

- Boxen plot for Number of Suicides vs Generation

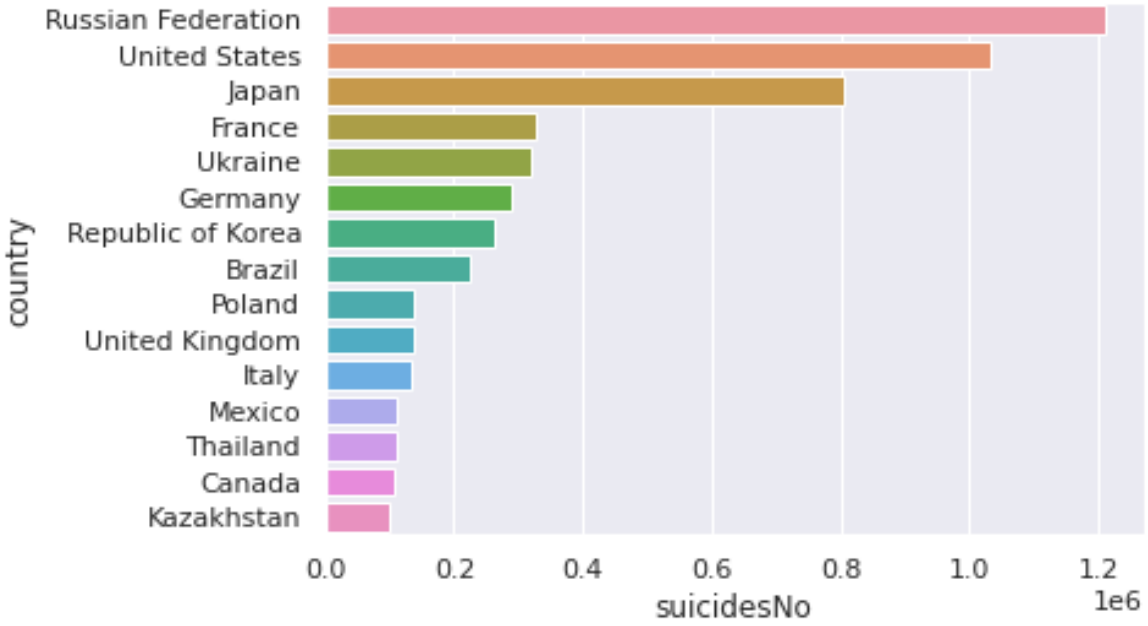
  Boxen plot provides a better representation of the distribution of the data than boxplot, but without distorting the appearance of the distribution as would be the case with an improperly configured violin plot.

```
sns.boxenplot(x="generation", y="suicides_no", color="b",
    scale="linear", data=df)
plt.tight_layout()
plt.xticks(rotation=90)
plt.show()
```



- Number of Suicides vs Country

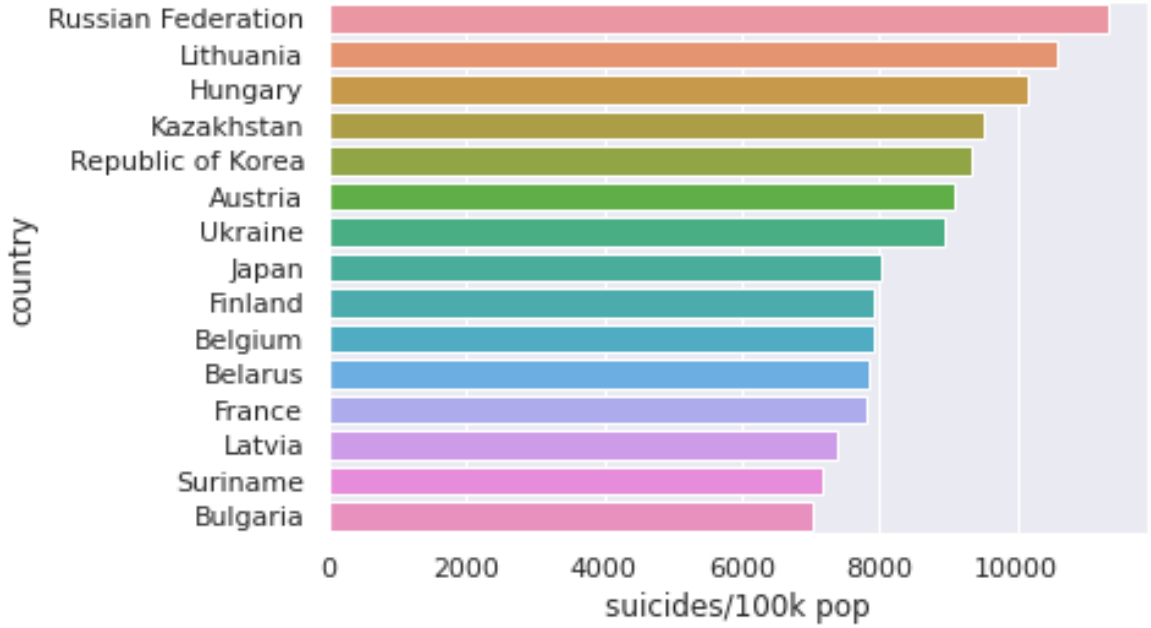
```
suicidesNo = []
for country in df['country'].unique():
    suicidesNo.append(sum(df[df['country']==country].suicides_no))
suicidesNo = pd.DataFrame(suicidesNo, columns=['suicidesNo'])
country=pd.DataFrame(df.country.unique(), columns=['country'])
Data_suicide_countr = pd.concat([suicidesNo, country], axis=1)
data_suicide_countr=data_suicide_countr.sort_values(by='suicidesNo',
                                                    ascending=False)
sns.barplot(y=data_suicide_countr.country[:15],
  x=data_suicide_countr.suicidesNo[:15])
plt.show()
```



This graph shows the top 15 countries with the most number of suicides
As can be seen from the barplot that the most number of suicides occurs in the Russian Federation followed by the United States and Japan.
This analysis, however, is not normalized based on the population of the countries.

- Number of Suicides per 100k population vs Country

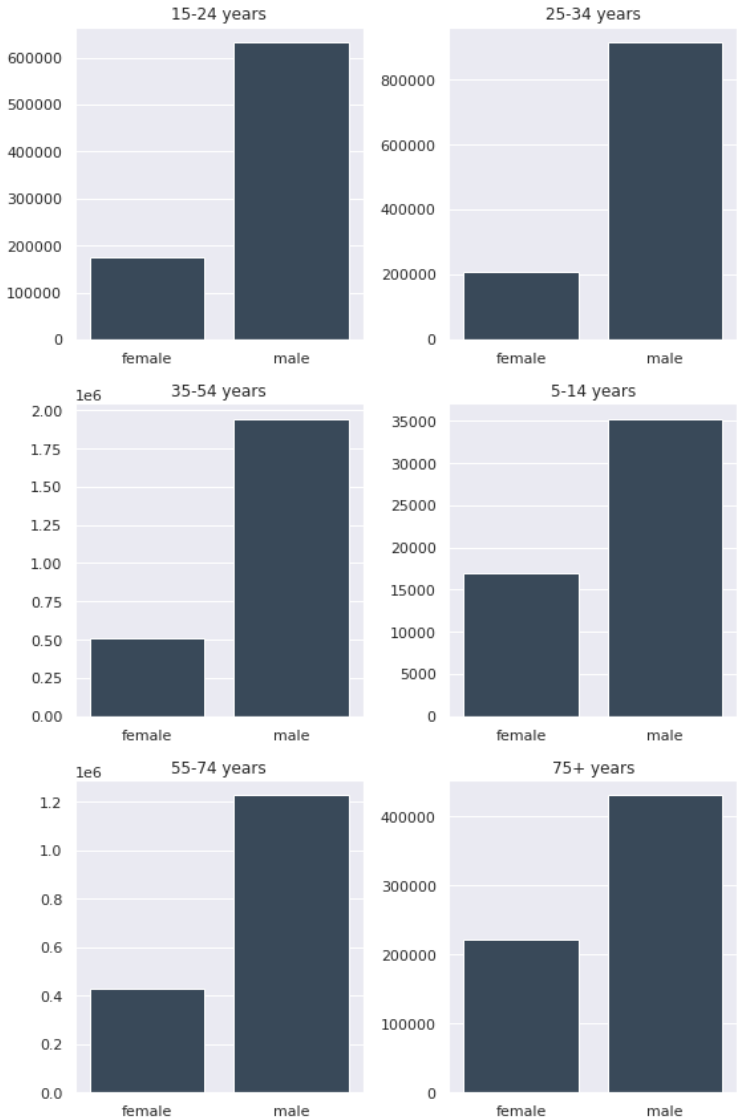
```
Suicidesperpop = []
for country in df['country'].unique():
    suicidesperpop.append(sum(
        df[df['country']==country]['suicides/100k pop']))
Suicidesperpop = pd.DataFrame(suicidesperpop,
                      columns=['suicides/100k pop'])
country=pd.DataFrame(df.country.unique(), columns=['country'])
Data_suicide_countr = pd.concat([suicidesperpop,country],axis=1)
Data_suicide_countr = data_suicide_countr.sort_values(
                    by='suicides/100k pop', ascending=False)
sns.barplot(y=data_suicide_countr.country[:15],
  x=data_suicide_countr['suicides/100k pop'][:15])
plt.show()
```



The same graph, now divided by the population, we can see that Russian Federation is still at the top however Japan has now moved down and the United States is no longer in the top 15.

- Number of readings segregated on Gender and Age Groups

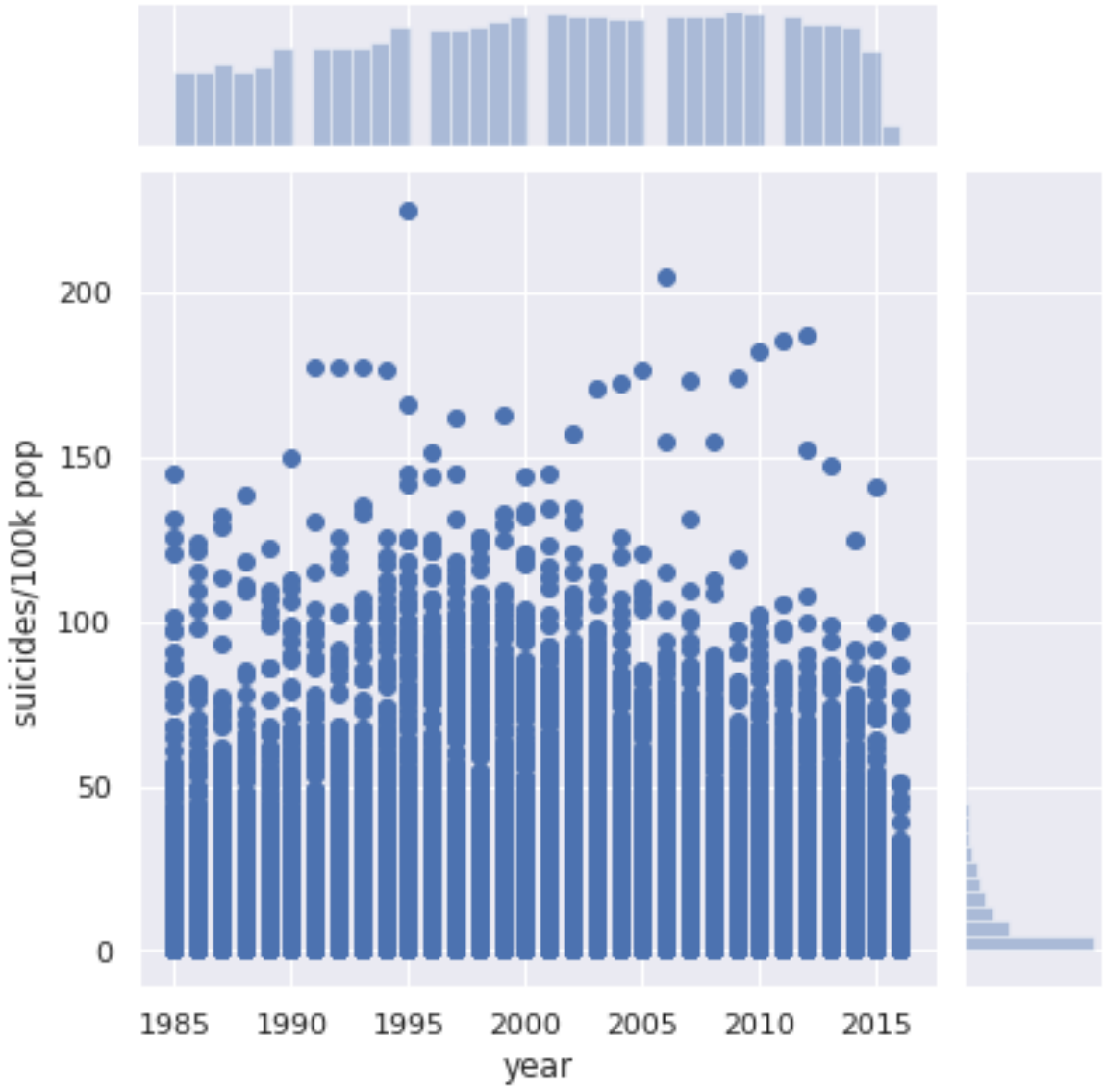
```
female_ = [175437,208823,506233,16997,430036,221984]
male_ = [633105,915089,1945908,35267,1228407,431134]
for i, age in enumerate(['15-24 years','25-34 years','35-54 years','5-
14 years','55-74 years','75+ years']):
    plt.subplot(3,2,i+1)
    plt.title(age)
    sns.barplot(x=['female','male'], y=[female_[i], male_[i]],
color="#34495e")
plt.show()
```



- Joint plot for Suicides/Population vs Year

Jointplot is used for displaying bivariate distribution along with the frequency distribution of each variable.

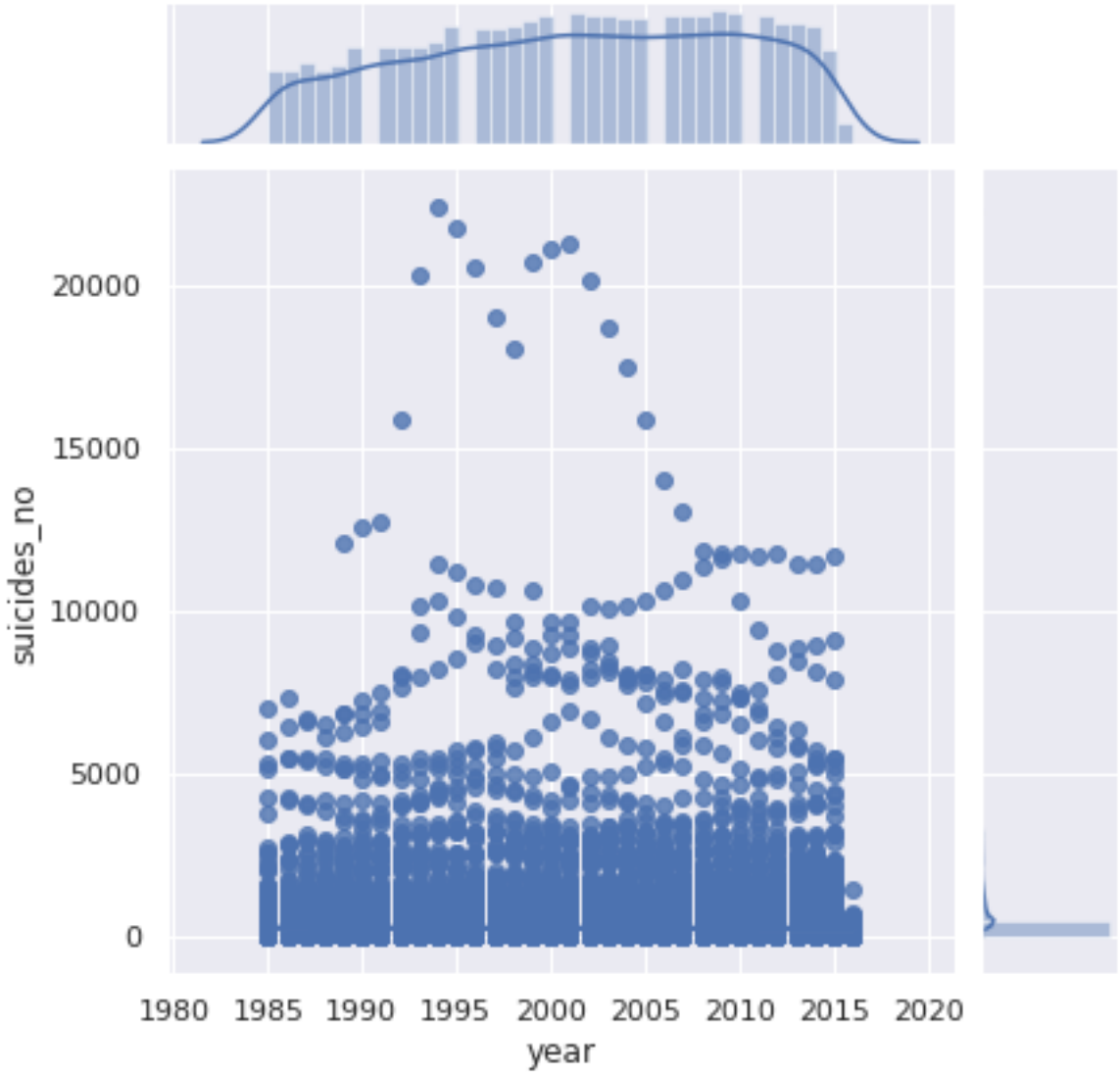
```
fig=sns.jointplot(y='suicides/100k pop',x='year',data=df)
plt.show()
```



- Joint plot for Suicides vs Year

Jointplot along with the estimated linear regression fit on the joint axes.

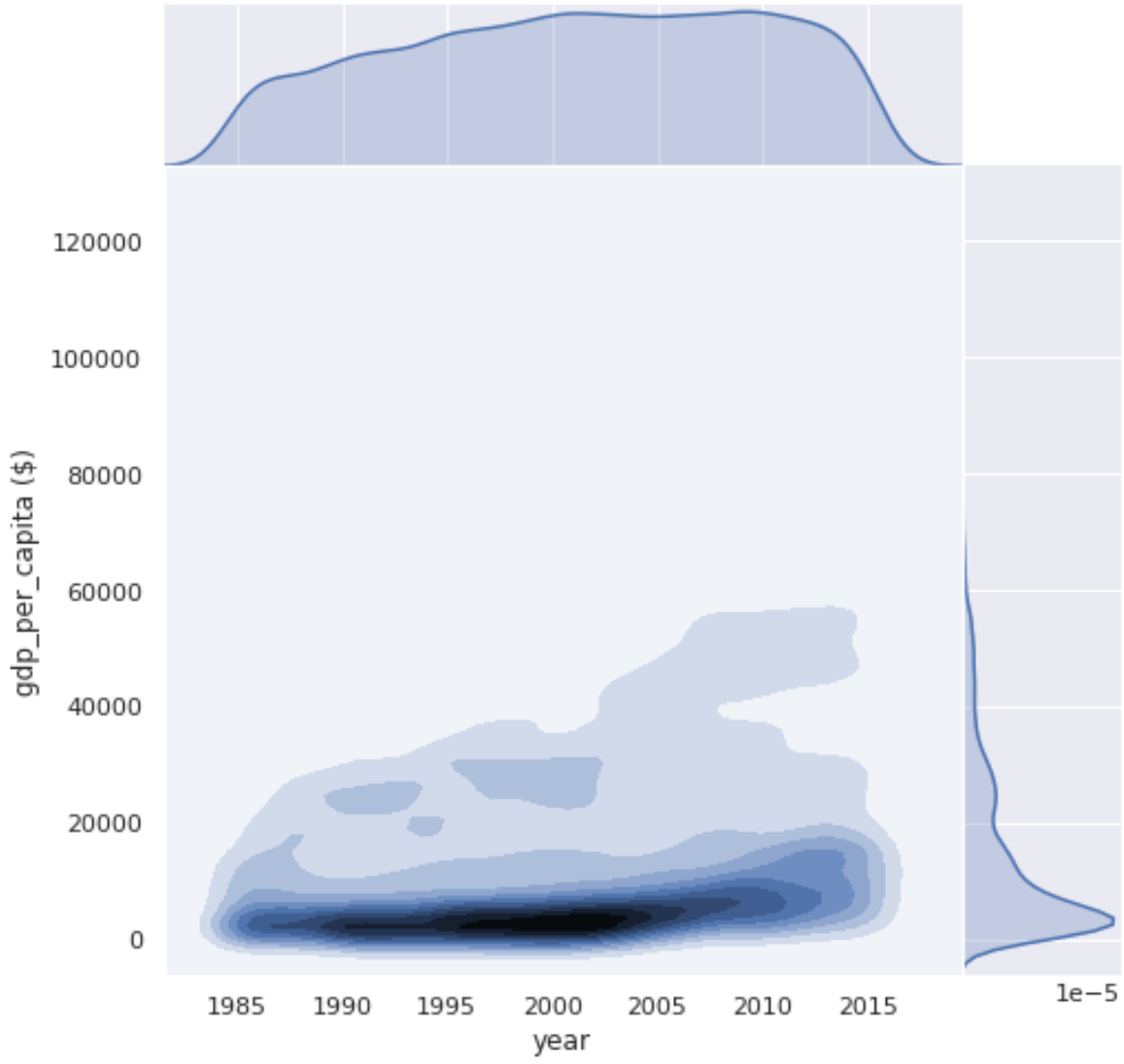
```
sns.jointplot("year", "suicides_no", data=df, kind="reg")
plt.show()
```



- Joint plot for GDP per Capita vs Year

Jointplot with Kernel density estimation that shows the distribution of GDP per Capita vs Year as a contour plot

```
g = sns.jointplot(df['year'],df['gdp_per_capita ($)'], kind="kde",
height=7, space=0)
plt.show()
```



Since we have explored our data, and now understand what factors to consider, we proceed to preprocess the data and get our dataset ready for prediction.

Preprocessing the Data

Our preprocessing involves getting rid of NA values, encoding and scaling our dataset in order to reduce computational time.

- Checking shape of our dataset.

```
df.shape
```

```
(27820, 12)
```

- Checking NA values

```
df.isna().sum()
```

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

- Dropping ‘HDI for year’ column since 70% of the data is missing.

```
df.drop(['HDI for year'], axis=1, inplace=True)
```

- Checking the datatypes of the columns in our dataset.

```
df.dtypes
```

```
country                 object
year                     int64
sex                     object
age                     object
suicides_no              int64
population               int64
suicides/100k pop      float64
country-year            object
 gdp_for_year ($)       object
gdp_per_capita ($)       int64
generation              object
dtype: object
```

- Defining x (independent variables) and y (dependent variable)

```
X, y = df.drop(['suicides_no', 'suicides/100k pop', 'country-year', '
gdp_for_year ($) '], axis=1), df['suicides_no']
```

x contains:

```
X['year'] = X['year'].astype('object')
X.head()
```

	country	year	sex	age	population	gdp_per_capita ($)	generation
0	Albania	1987	male	15-24 years	312900	796	Generation X
1	Albania	1987	male	35-54 years	308000	796	Silent
2	Albania	1987	female	15-24 years	289700	796	Generation X
3	Albania	1987	male	75+ years	21800	796	G.I. Generation
4	Albania	1987	male	25-34 years	274300	796	Boomers

y contains:

```
y.head()
```

```
0    21
1    16
2    14
3     1
4     9
Name: suicides_no, dtype: int64
```

- ## Encoding the independent variables.

  Encoding is essential since the algorithms do not accept dtypes ‘objects’ and, also reduces computation time.
  We one-hot-encode the independent variables since they are nominal in nature and not ordinal.

```
X = pd.get_dummies(X, drop_first=True)
```

- ## Scaling the variables.

  We use MinMax scaling.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X['population'] = scaler.fit_transform(X[['population']])
X['gdp_per_capita ($)'] = scaler.fit_transform(X[['gdp_per_capita ($)']])
X.head()
```

	population	gdp_per_capita ($)	country_Antigua and Barbuda	country_Argentina	country_Armenia	country_Aruba	country_Australia	country_Austria	country_Azerbai
0	0.007137	0.004322	0	0	0	0	0	0	
1	0.007025	0.004322	0	0	0	0	0	0	
2	0.006607	0.004322	0	0	0	0	0	0	
3	0.000491	0.004322	0	0	0	0	0	0	
4	0.006256	0.004322	0	0	0	0	0	0	

5 rows × 144 columns

- ## Splitting the dataset into training and testing

```
from sklearn.model_selection import train_test_split as tts

X_train, X_test, y_train, y_test = tts(X, y, test_size=0.3,
random_state=42)
```

Fitting the Model

Since we need to predict the number of suicides, we use regression algorithms. We tried the following algorithms:

1. Linear Regression:

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_test)
print(r2_score(y_test, y_pred))
```

```
0.5838130107197023
```

```
print(np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
546.249224708846
```

Since the accuracy is less(58.38%), under the assumption that the model is overfit to training data, we try regularisation to prevent overfitting.

2. Lasso (L1 Regularisation):

```
from sklearn.linear_model import Lasso
lasso = Lasso()
lasso.fit(X_train, y_train)
y_pred = lasso.predict(X_test)
print(r2_score(y_test, y_pred))
```

```
0.5834036085173047
```

```
print(np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
546.5178307332759
```

Since the accuracy has not improved(58.34%), our assumption of overfitting was incorrect. We now attempt to fit a robust regressor in order to treat outliers if that is the problem. Out of RANSAC, Huber and ThielSen, we choose Huber Regressor.

3. Huber (Robust Regressor):

```
from sklearn.linear_model import HuberRegressor
huber = HuberRegressor(max_iter=1000)
huber.fit(X_train, y_train)
```

```
y_pred = huber.predict(X_test)
print(r2_score(y_test, y_pred))
```

```
0.4380866462577737
```

```
print(np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
634.7186515803572
```

As the accuracy has reduced(43.81%) it is possible that in the process of avoiding outliers our model has ignored some high leverage points causing a drop in the accuracy.
We suspect that this is because we are trying to fit a linear model on a non-linear dataset. To overcome this, we use a non-euclidean model, in this case, a Random Forest Regressor Model.

4. Random Forest Regressor:

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print(r2_score(y_test, y_pred))
```

```
0.98381641984908
```

```
print(np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
546.249224708846
```

Thus, Random Forest Regressor has the best accuracy(98.38%).