# MACHINE LEARNING

(Suicide Rate Prediction)

*Summer Internship Report Submitted in partial fulfilment of the requirement for*

*undergraduate degree of*

**Bachelor of Technology**

In

**Computer Science Engineering**

By

**Ettom. Srinija**

**221710310015**


Department of Computer Science and Engineering
GITAM School of Technology
GITAM (Deemed to be University)
Hyderabad-502329

June 2020

# DECLARATION

I submit this industrial training work entitled “SUICIDE RATES PREDICTION” to GITAM (Deemed to Be University), Hyderabad in partial fulfilment of the requirements for the award of the degree of “Bachelor of Technology ” in “Computer Science Engineering”. I declare that it was carried out independently by me under the guidance.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

**Place:**
**Hyderabad**
**Date: 12.07.2020**

**Ettom. Srinija**
**221710310015**

# CERTIFICATE

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated: **12.07.2020**

# CERTIFICATE

This is to certify that the Industrial Training Report entitled "**Suicide Rates Prediction**" is being submitted by Ettom. Srinija (221710310015) in partial fulfilment of the requirement for the award of Bachelor of Technology in **Computer Science and Engineering** at GITAM (Deemed to Be University), Hyderabad during the academic year 2020-21.

It is faithful record work carried out by her at the **Computer Science and Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

Assistant Professor
Department of CSE

**Dr. Phani Kumar**
Professor and HOD
Department of CSE

# CERTIFICATE

# ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad**, Pro Vice Chancellor, GITAM Hyderabad and **Dr. N. Seetharamaiah**, Principal, GITAM Hyderabad.

I would like to thank respected **Dr. Phani Kumar**, Head of the Department of Electronics and Communication Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Mr. _____** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Ettom. Srinija
221710310015

# ABSTRACT

Suicidal thoughts and behaviours are an international public health problem contributing to 800,000 annual deaths and up to 25 million nonfatal suicide attempts. In the United States, suicide rates have increased steadily for two decades, reaching 47,000 per year and surpassing annual motor vehicle deaths. This trend has prompted government agencies, healthcare systems, and multinational corporations to invest in artificial intelligence-based suicide prediction algorithms.

AI-based suicide prediction is developing along two separate tracks. In "medical suicide prediction," AI analyzes data from patient medical records. In "social suicide prediction," AI analyzes consumer behaviour derived from social media, smartphone apps, and the Internet of Things (IoT). Because medical suicide prediction occurs within the context of healthcare, it is governed by the Health Information Portability and Accountability Act (HIPAA), which protects patient privacy; the Federal Common Rule, which protects the safety of human research subjects; and general principles of medical ethics. Medical suicide prediction tools are developed methodically in compliance with these regulations, and the methods of its developers are published in peer-reviewed academic journals. In contrast, social suicide prediction typically occurs outside the healthcare system where it is almost completely unregulated. Corporations maintain their suicide prediction methods as proprietary trade secrets. Despite this lack of transparency, social suicide predictions are deployed globally to affect people's lives every day. Yet little is known about their safety or effectiveness.

Though AI-based suicide prediction has the potential to improve our understanding of suicide while saving lives, it raises many risks that have been underexplored. The risks include stigmatization of people with mental illness, the transfer of sensitive personal data to third-parties such as advertisers and data brokers, unnecessary involuntary confinement, violent confrontations with police, exacerbation of mental health conditions, and paradoxical increases in suicide risk.

# INDEX

# FIGURE INDEX

# CHAPTER 1

# MACHINE LEARNING

## 1.1 INTRODUCTION:

Machine Learning (ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence (AI).

## 1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and "more items to consider" and "get yourself a little something" on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today's data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

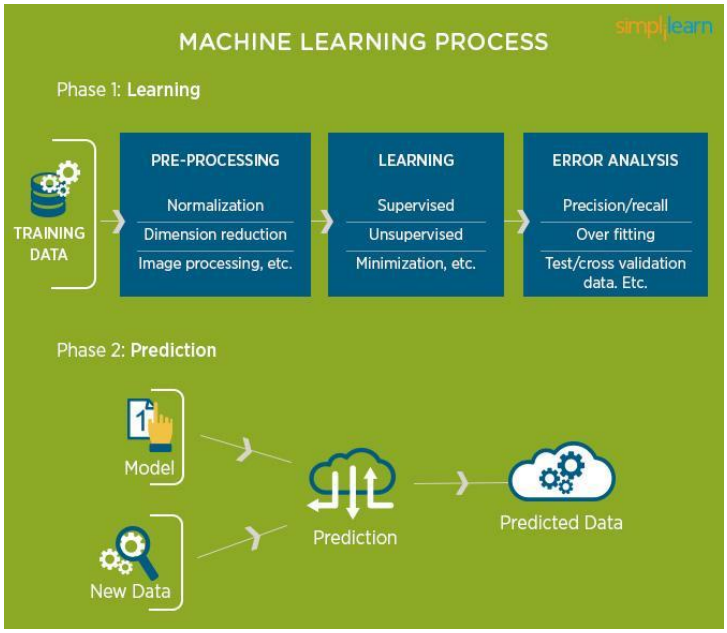


Figure 1.2.1: The Process Flow

## 1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

## 1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

### 1.4.1 **Supervised Learning:**

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

### 1.4.2 **Unsupervised Learning:**

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

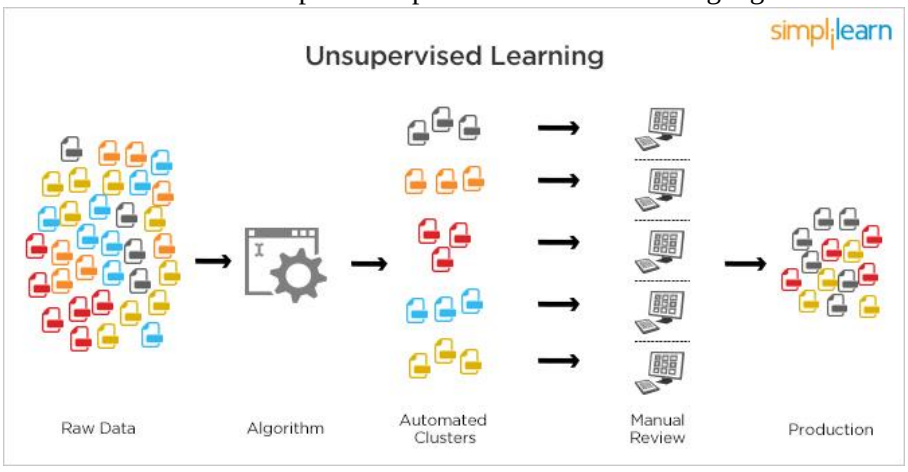


Figure 1.4.2.1: Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbour mapping, singular value decomposition, and k-

means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

### 1.4.3 **Semi Supervised Learning**:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labelled and un-labelled data for training. In a typical scenario, the algorithm would use a small amount of labelled data with a large amount of un-labelled data.

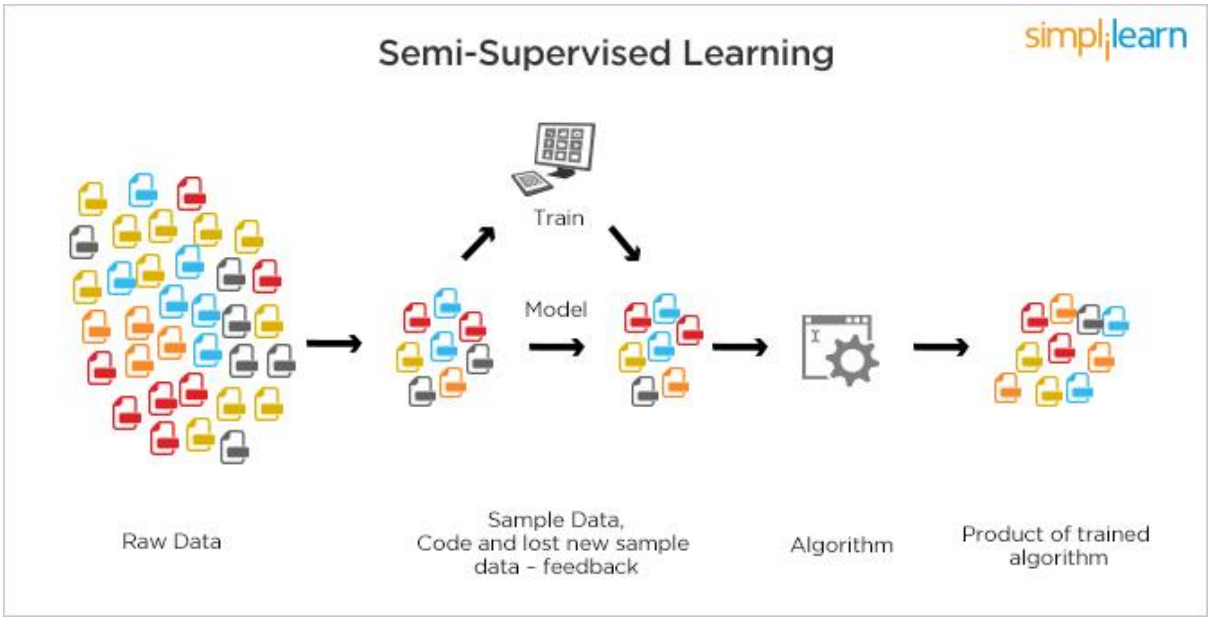


Figure 1.4.3.1: Semi Supervised Learning

## 1.5 RELATION BETWEEN DATA MINING, MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special 5 types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

# CHAPTER 2

# DEEP LEARNING

## 2.1 INTRODUCTION TO DEEP LEARNING

Deep learning is a class of ML algorithms that uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

## 2.2 IMPORTANCE OF DEEP LEARNING

The ability to process large numbers of features makes deep learning very powerful when dealing with unstructured data. However, deep learning algorithms can be overkill for less complex problems because they require access to a vast amount of data to be effective. For instance, ImageNet, the common benchmark for training deep learning models for comprehensive image recognition, has access to over 14 million images.

The data is too simple or incomplete, it is very easy for a deep learning model to become overfitted and fail to generalize well to new data. As a result, deep learning models are not as effective as other techniques (such as boosted decision trees or linear models) for most practical business problems such as understanding customer churn, detecting fraudulent transactions and other cases with smaller datasets and fewer features. In certain cases like multiclass classification, deep learning can work for smaller, structured datasets.

## 2.3 USES OF DEEP LEARNING:

### 2.3.1 Self Driving Cars:

Deep Learning is the force that is bringing autonomous driving to life. A million sets of data are fed to a system to build a model, to train the machines to learn, and then test the results in a safe environment. The Uber Artificial Intelligence Labs at Pittsburg is not only working on making driverless cars humdrum but also integrating several smart features such as food delivery options with the use of driverless cars. The major concern for autonomous car developers is handling unprecedented scenarios.

A regular cycle of testing and implementation typical to deep learning algorithms is ensuring safe driving with more and more exposure to millions of scenarios. Data from cameras, sensors, geo-mapping is helping create succinct and sophisticated models to navigate through traffic, identify paths, signage, pedestrian-only routes, and real-time elements like traffic volume and road blockages. According to Forbes, MIT is developing a new system that will allow autonomous cars to navigate without a map as 3-D mapping is still limited to prime areas in the world and not as effective in avoiding mishaps.

CSAIL graduate student Teddy Ort said, "The reason this kind of 'map-less' approach hasn't really been done before is because it is generally much harder to reach the same accuracy and reliability as with detailed maps. A system like this that can navigate just with on-board sensors shows the potential of self-driving cars being able to actually handle roads beyond the small number that tech companies have mapped."

### 2.3.2 Virtual Assistants:

The most popular application of deep learning is virtual assistants ranging from Alexa to Siri to Google Assistant. Each interaction with these assistants provides them with an opportunity to learn more about your voice and accent, thereby providing you a secondary human interaction experience. Virtual assistants use deep learning to know more about their subjects ranging from your dine-out preferences to your most visited spots or your favourite songs. They learn to understand your commands by evaluating natural human language to execute them.

Another capability virtual assistants are endowed with is to translate your speech to text, make notes for you, and book appointments. Virtual assistants are literally at your beck-and-call as they can do everything from running errands to auto-responding to your specific calls to coordinating tasks between you and your team members. With deep learning applications such as text generation and document summarizations, virtual assistants can assist you in creating or sending appropriate email copy as well.

## 2.4 RELATION BETWEEN DATA MINING, MACHINE LEARNING AND DEEP LEARNING:

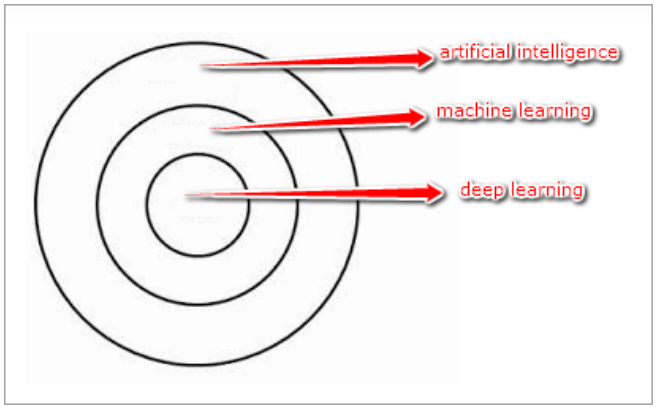


Figure 2.4.1: Relation

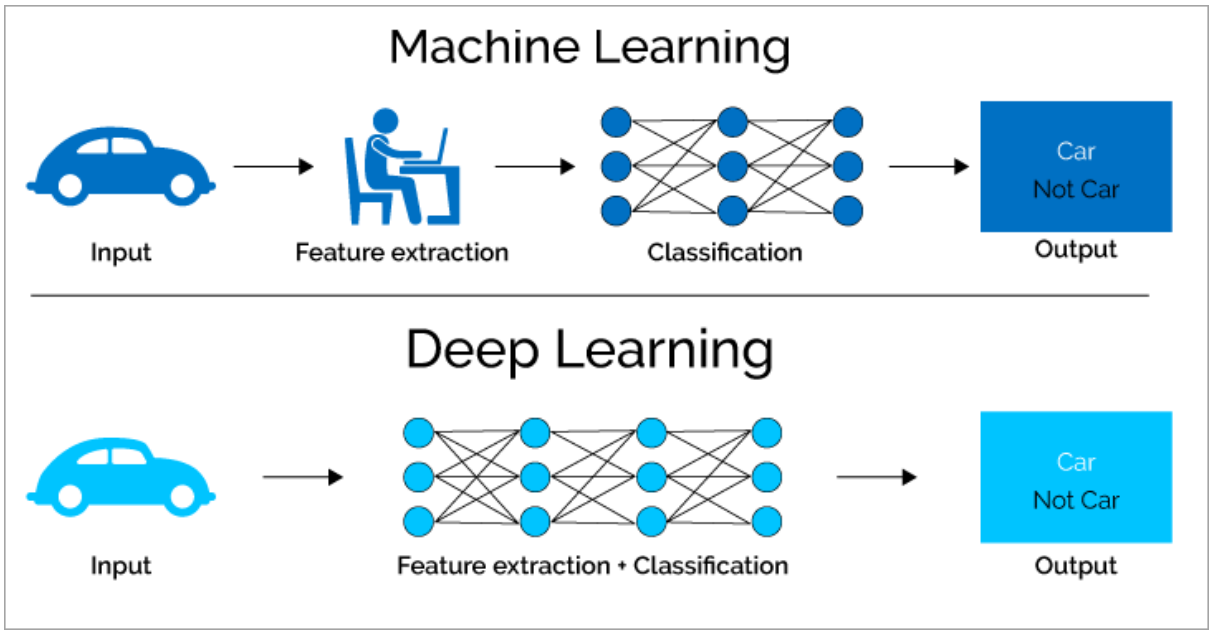


Figure 2.4.2: Machine Learning vs Deep Learning

Machine Learning comprises of the ability of the machine to learn from trained data set and predict the outcome automatically. It is a subset of artificial intelligence.

Deep Learning is a subset of machine learning. It works in the same way on the machine just like how the human brain processes information. Like a brain can identify the patterns by comparing it with previously memorized patterns, deep learning also uses this concept.

Deep learning can automatically find out the attributes from raw data while machine learning selects these features manually which further needs processing. It also employs artificial neural networks with many hidden layers, big data, and high computer resources.

Data Mining is a process of discovering hidden patterns and rules from the existing data. It uses relatively simple rules such as association, correlation rules for the decision-making process, etc. Deep Learning is used for complex problem processing such as voice recognition etc. It uses Artificial Neural Networks with many hidden layers for processing.

At times data mining also uses deep learning algorithms for processing the data

# CHAPTER 3

# PYTHON

Basic programming language used for machine learning is: PYTHON

## 3.1 INTRODUCTION TO PYHTON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

## 3.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7, it is generally called as python3

## 3.3 FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

- Extendable: You can add low-level modules to the Python interpreter. These enable programmers to add to or customize their tools to be more efficient.

- Databases: Python provides interfaces to all major commercial databases.

- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

## 3.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

### 3.4.1 Installation (using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.

- Download python from www.python.org

- When the download is completed, double click the file and follow the instructions to install it.

- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.


Figure 3.4.1.1: Python download

### 3.4.2 Installation (using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.
- In WINDOWS:
- In windows
  - Step 1: Open Anaconda.com/downloads in web browser.
  - Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
  - Step 3: select installation type(all users)
  - Step 4: Select path (i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
  - Step 5: Open jupyter notebook (it opens in default browser)

Figure 5 : Anaconda download
Figure 6 : Jupyter notebook

## 3.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

● Python has five standard data types –

o Numbers

o Strings

o Lists

o Tuples

o Dictionary

### 3.5.1 Python Numbers:

● Number data types store numeric values. Number objects are created when you assign a value to them.

● Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

### 3.5.2 Python Strings:

● Strings in Python are identified as a contiguous set of characters represented in the quotation marks.

● Python allows for either pairs of single or double quotes.

● Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

● The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

### 3.5.3 Python Lists:

● Lists are the most versatile of Python's compound data types.

● A list contains items separated by commas and enclosed within square brackets 11 ([]).

● To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

● The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.

● The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

### 3.5.4 Python Tuples:

● A tuple is another sequence data type that is similar to the list.

● A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

● The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated.

● Tuples can be thought of as read-only lists.

● For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

### 3.5.5 Python Dictionary:

● Python's dictionaries are kind of hash table type. They work like associative arrays 12 or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

● Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

● You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.

● What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

## 3.6 PYTHON FUNCTION:

### 3.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

### 3.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

## 3.7 PYTHON USING OOP's CONCEPTS:

### 3.7.1 Class:

● Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

● Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.

● Data member: A class variable or instance variable that holds data associated with a class and its objects.

● Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

● Defining a Class:

o We define a class in a very similar way how we define a function.

o Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is indented like a functions body is.

```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

Figure 3.7.1.1: Defining a Class

### 3.7.2 __init__ method in Class:

● The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.

● The init method has a special name that starts and ends with two underscores: __init__().

# CHAPTER 4

# SUICIDE RATES PREDICTION

## 4.1 PROJECT REQUIREMENTS:

### 4.1.1 Packages Used:

Packages imported for the project are:

**Importing Packages:**

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import missingno as msno
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
from sklearn.ensemble import RandomForestRegressor
```

Figure 4.1.1.1: Importing Packages

### 4.1.2 Versions of the packages used:

**Checking Versions of the Packages:**

```
print(np.__version__)  #numpy
print(pd.__version__)  #pandas
print(sns.__version__)  #seaborn
print(sklearn.__version__) #sklearn
print(matplotlib.__version__)  #matplotlib
print(msno.__version__) #missingno
```

```
1.17.2
0.25.1
0.9.0
0.21.3
3.2.2
0.4.2
```

Figure 4.1.2.1: Versions of the packages

### 4.1.3 Algorithms used:

#### 4.1.3.1. Simple Linear Regression:

Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. It's used to predict values within a continuous range, (e.g. sales, price) rather than trying to classify them into categories (e.g. cat, dog).

Simple linear regression uses traditional slope-intercept form, where mm and bb are the variables our algorithm will try to "learn" to produce the most accurate predictions. xx represents our input data and y represents our prediction.

$$y=mx+b$$

#### 4.1.3.2. K-Nearest Neighbors Regressor:

A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighbors. Another approach uses an inverse distance weighted average of the K nearest neighbors. KNN regression uses the same distance functions as KNN classification.

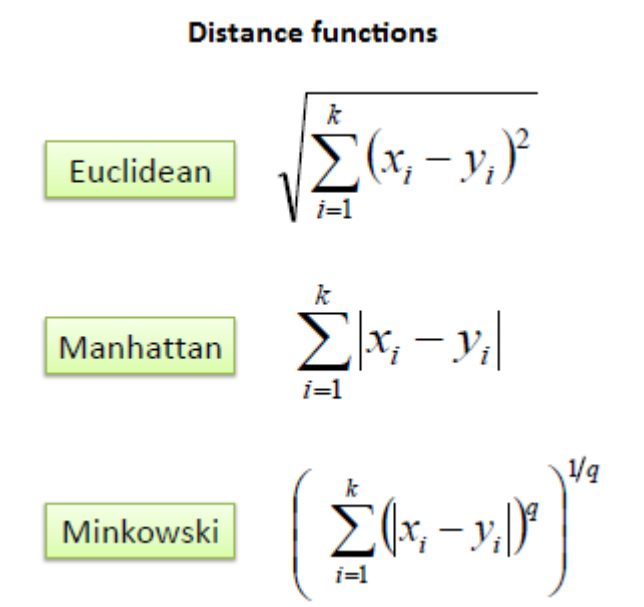


Figure 4.1.3.2.1 Distance

#### 4.1.3.3 Random Forest Regressor:

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as **bagging**.

The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

## 4.2 PROBLEM STATEMENT:

To Predict the Rate of suicides using various Machine Learning Algorithms Using various factors such as a person's age, sex, Generation also based on the country's population.

## 4.3 DATA SET DESCRIPTION:

The given data set consists of the following parameters:

1. Country
2. Year
3. Sex: Gender
4. Age: Range (Ex:15-24 years)
5. Suicides_no: Number of Suicides committed
6. Population
7. Suicides_hk: Sucicides per 100k population
8. Generation: (Ex: Silent, Boomers)

## 4.4 OBJECTIVE OF THE CASE STUDY:

To get a better understanding about what conditions and situations come under suicide situations, Example: If the person is a male or female, and his/her age range, Population in the country where the person lives, Generation the person belongs to.

# CHAPTER 5

# DATA PRE-PROCESSING

## 5.1 STATISTICAL ANALYSIS:

```
1 df.describe()
```

| | year | suicides_no | population | suicides_hk |
|---|---|---|---|---|
| count | 27820.000000 | 27820.000000 | 2.782000e+04 | 27820.000000 |
| mean | 2001.258375 | 242.574407 | 1.844794e+06 | 12.816097 |
| std | 8.469055 | 902.047917 | 3.911779e+06 | 18.961511 |
| min | 1985.000000 | 0.000000 | 2.780000e+02 | 0.000000 |
| 25% | 1995.000000 | 3.000000 | 9.749850e+04 | 0.920000 |
| 50% | 2002.000000 | 25.000000 | 4.301500e+05 | 5.990000 |
| 75% | 2008.000000 | 131.000000 | 1.486143e+06 | 16.620000 |
| max | 2016.000000 | 22338.000000 | 4.380521e+07 | 224.970000 |

Figure 5.1.1 Statistical Analysis.

## 5.2 GENERATING PLOTS:

### 5.2.1: VISUALIZE THE DATA BETWEEN TARGET AND FEATURES:

Here the target column is ‘suicides_no’ and the other columns are feature columns.

```
1 sns.barplot(x="sex",y='suicides_no',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x264faf4f978>
```

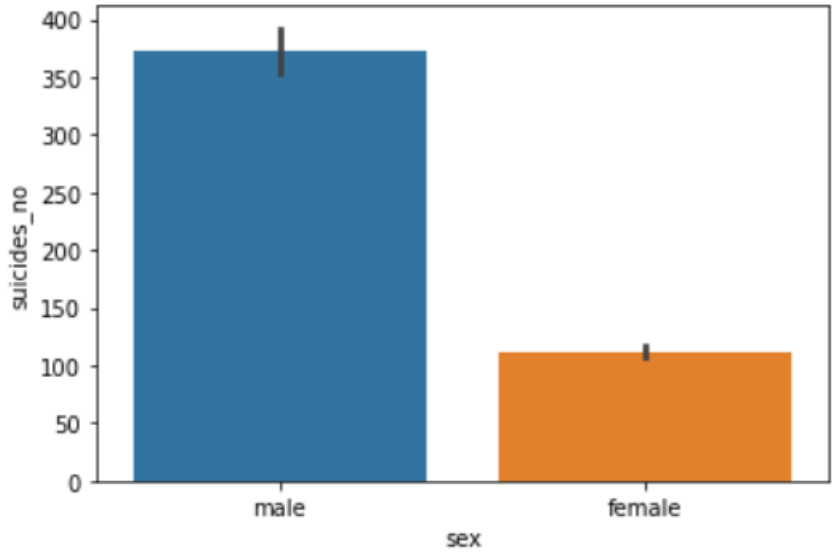


Figure 5.2.1.1 Barplot between sex, suicides_no

From the Barplot, we can observe that more male have committed suicide than compared to the female. There are almost 375 male members whereas as there are more than 100 female members who committed suicide.

```
1 sns.catplot(x='suicides_no', y='population', data =df)
```

```
<seaborn.axisgrid.FacetGrid at 0x264fa8cef98>
```

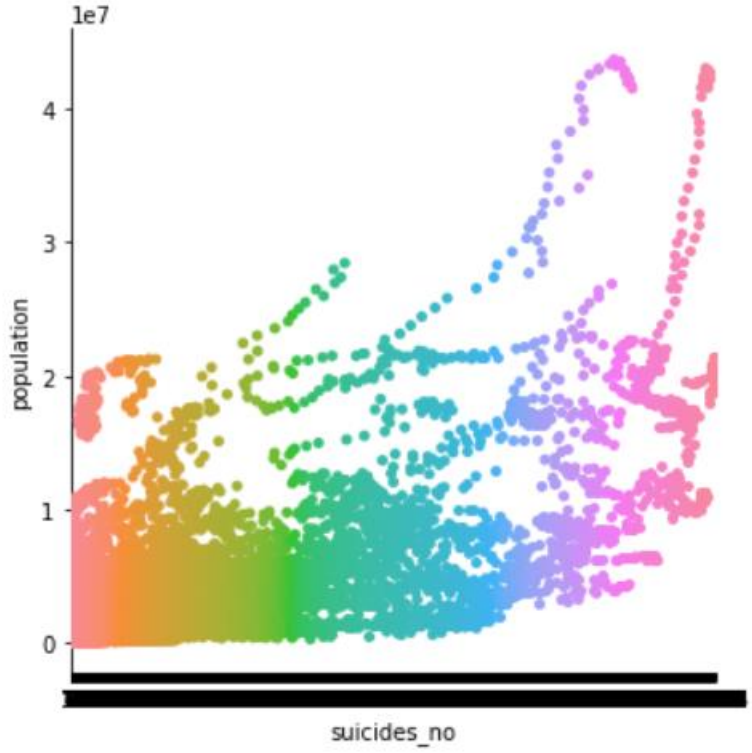


Figure 5.2.1.2 Catplot between population and suicides_no

From the Catplot, We can observe that as population gradually increases, the suicide number is increasing.

```
1 plt.figure(figsize=(10,10))
2 sns.barplot(x='age',y='suicides_no',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x264d76f1400>
```

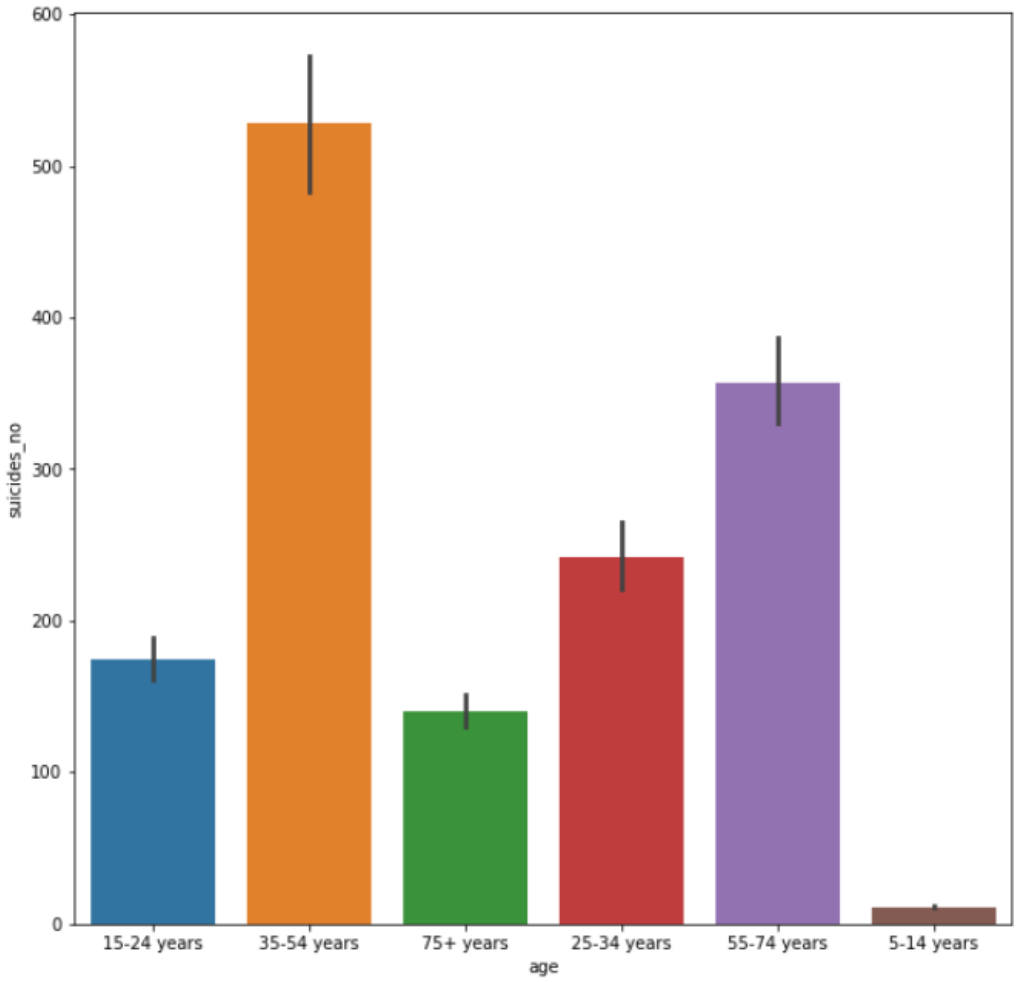


Figure 5.2.1.3: Barplot between age range and suicides_no

From the Barplot we can observe that citizen in the age range of 35-54 years are most prone to committing suicide. Then, next comes the age range of 55-74 years. Then it is continued by the age range of 25-34 years. Then 15-24 years. Then comes the age range of 5-14 years which is the least.

```
1 plt.figure(figsize=(50,50))
2 sns.barplot(x='year',y='suicides_no',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x26486e64160>
```

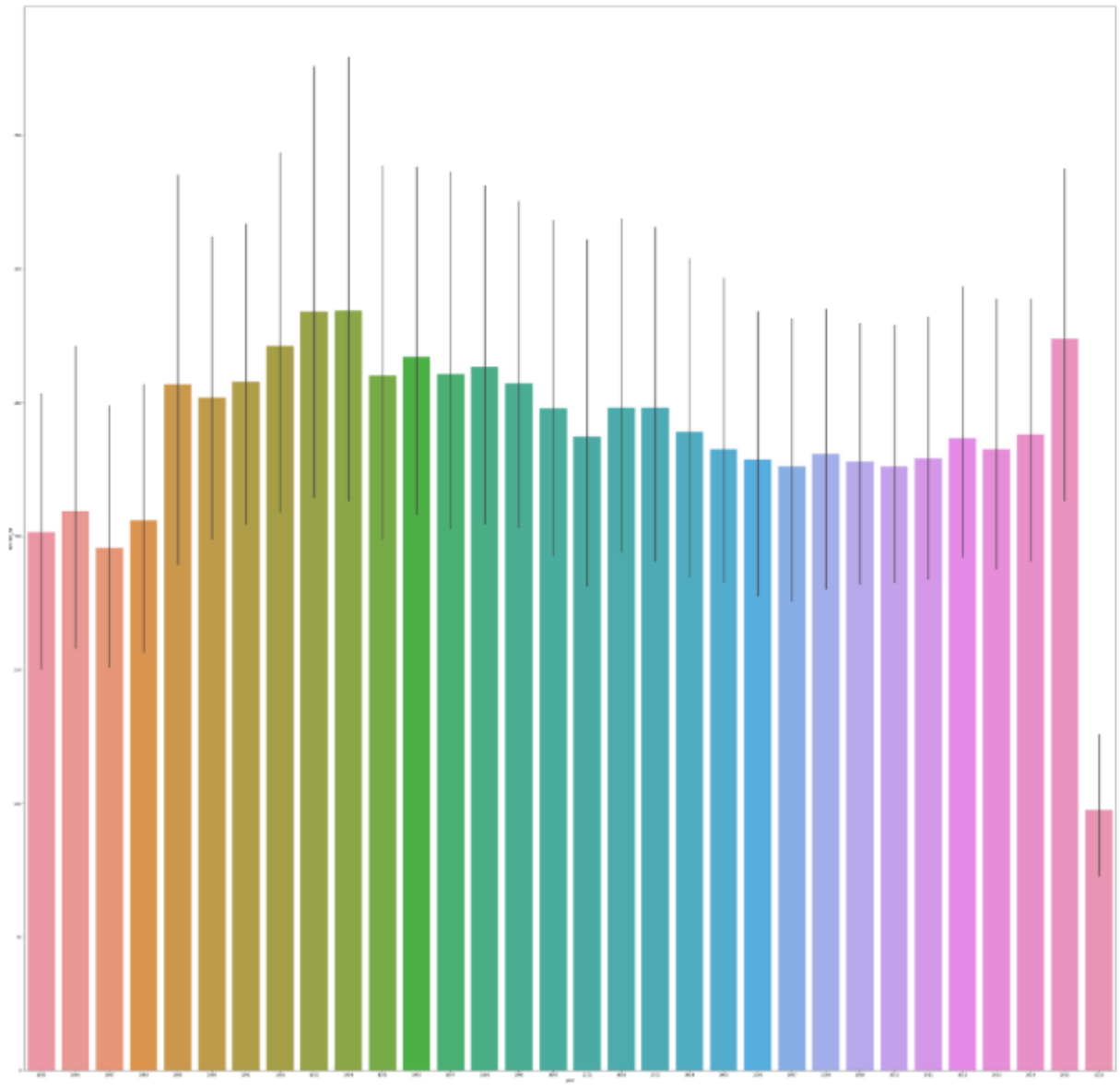

Figure 5.2.1.4: Barplot between year and suicides_no

From the Barplot, We can see that the suicide rates vary from year to year

## 5.3 DATA CONVERSIONS:

```
1  df.dtypes
```

```
country          object
year              int64
sex              object
age              object
suicides_no       int64
population        int64
suicides_hk     float64
generation       object
dtype: object
```

```
1  #Converting the datatypesof some of the columns to convinient datatypes
2  df.year = df.year.astype(int)
3  df.suicides_no = df.suicides_no.astype(int)
4  df.population = df.population.astype(int)
5  df.suicides_hk = df.suicides_hk.astype(float)
```

```
1  df.dtypes
```

```
country          object
year              int32
sex              object
age              object
suicides_no       int32
population        int32
suicides_hk     float64
generation       object
dtype: object
```

Figure 5.3.1 Datatype conversions

## 5.4 HANDLING MISSING VALUES:

My Dataset Doesn't Contain any null values.

```
1 df.isnull().sum()
```

```
country         0
year            0
sex             0
age             0
suicides_no     0
population      0
suicides_hk     0
generation      0
dtype: int64
```

Figure 5.4.1 Missing values

```
1 sns.heatmap(df.isna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x21836818a90>
```

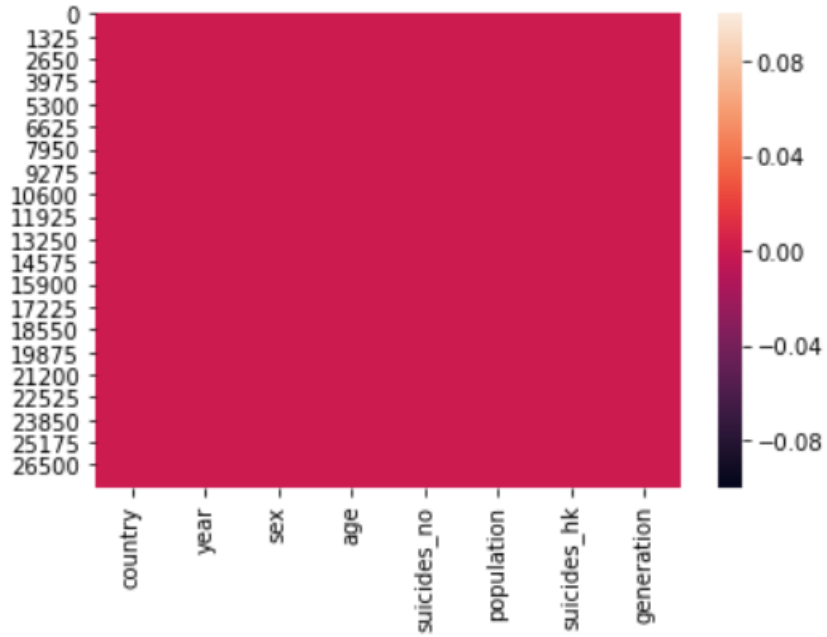


Figure 5.4.2 Heatmap showing that there are no missing values

However, Missing values can be handled in many ways using some inbuilt methods:

**(a)dropna():**

dropna() is a function which drops all the rows and columns which are having the missing values(i.e. NaN)

● dropna() function has a parameter called how which works as follows

● if how = 'all' is passed then it drops the rows where all the columns of the particular row are missing

● if how = 'any' is passed then it drops the rows where all the columns of the particular row are missing

**(b)fillna():**

fillna() is a function which replaces all the missing values using different ways.

● fillna() also have parameters called method and axis

● if we use method = 'ffill' where ffill is a method called forward fill, which carry forwards the previous row's value

● if we use method = 'bfill' where bfill is a method called backward fill, which carry backward the next row's value

● if we use method = 'ffill' , axis = 'columns' then it carry forwards the previous column's value

● if we use method = 'bfill' , axis = 'columns' then it carry backward the next column's value 22

**(c)interpolate():**

● interpolate() is a function which comes up with a guess value based on the other values in the dataset and fills those guess values in the place of missing values

**(d)mean and median imputation**:

● mean and median imputation can be performed by using fillna().

● mean imputation calculates the mean for the entire column and replaces the missing values in that column with the calculated mean.

● median imputation calculates the median for the entire column and replaces the missing values in that column with the calculated median.

## 5.5 ENCODING CATEGORICAL DATA:

● Machine Learning models are based on equations, we need to replace the text by numbers. So that we can include the numbers in the equations.

● Categorical Variables are of two types: Nominal and Ordinal

● **Nominal**: The categories do not have any numeric ordering in between them. They don't have any ordered relationship between each of them. Examples: Male or Female, any colour

● **Ordinal:** The categories have a numerical ordering in between them. Example: Graduate is less than Post Graduate, Post Graduate is less than Ph.D. customer satisfaction survey, high low medium

● Categorical data can be handled by using dummy variables, which are also called as indicator variables.

● **Handling categorical data using dummies:**

In pandas library we have a method called **get_dummies()** which creates dummy variables for those categorical data in the form of 0's and 1's.

Once these dummies got created we have to concat this dummy set to our dataframe or we can add that dummy set to the dataframe.

| age | generation |
|---|---|
| 15-24 years | Generation X |
| 35-54 years | Silent |
| 15-24 years | Generation X |
| 75+ years | G.I. Generation |
| 25-34 years | Boomers |
| ... | ... |
| 35-54 years | Generation X |
| 75+ years | Silent |
| 5-14 years | Generation Z |
| 5-14 years | Generation Z |
| 55-74 years | Boomers |

Figure 5.5.1: Categorical data

**Concatenating the dataframe with dummies of 'age' column**

```
df = pd.concat([df,pd.get_dummies(df.age,prefix="Age",drop_first=True)] ,axis=1)
df.head()
```

| | sex | age | suicides_no | population | suicides_hk | generation | Age_25-34 years | Age_35-54 years | Age_5-14 years | Age_55-74 years | Age_75+ years |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | male | 15-24 years | 21 | 312900 | 6.71 | Generation X | 0 | 0 | 0 | 0 | 0 |
| 1 | male | 35-54 years | 16 | 308000 | 5.19 | Silent | 0 | 1 | 0 | 0 | 0 |
| 2 | female | 15-24 years | 14 | 289700 | 4.83 | Generation X | 0 | 0 | 0 | 0 | 0 |
| 3 | male | 75+ years | 1 | 21800 | 4.59 | G.I. Generation | 0 | 0 | 0 | 0 | 1 |
| 4 | male | 25-34 years | 9 | 274300 | 3.28 | Boomers | 1 | 0 | 0 | 0 | 0 |

Figure 5.5.2: Getting dummies for ‘age’ column

```
df = pd.concat([df,pd.get_dummies(df.generation,prefix="Gen",drop_first=True)]   ,axis=1)
df.head()
```

| no | population | suicides_hk | generation | Age_25-34 years | Age_35-54 years | Age_5-14 years | Age_55-74 years | Age_75+ years | Gen_G.I. Generation | Gen_Generation X | Gen_Generation Z | Gen_Millenials | Gen_Silent |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21 | 312900 | 6.71 | Generation X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 16 | 308000 | 5.19 | Silent | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 14 | 289700 | 4.83 | Generation X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 21800 | 4.59 | G.I. Generation | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 9 | 274300 | 3.28 | Boomers | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 5.5.3: Getting Dummies for ‘generation’ column

- **Getting dummies using label encoder from scikit learn package**

We have a method called label encoder in scikit learn package .we need to import the label encoder method from scikitlearn package and after that we have to fit and transform the data frame to make the categorical data into dummies.

If we use this method to get dummies then in place of categorical data we get the numerical values (0,1,2….)

**Label Encoding 'sex' column** ¶

```
1  df["sex"].value_counts()
```

```
female     13910
male       13910
Name: sex, dtype: int64
```

```
1  df['sex']=LabelEncoder().fit_transform(df.sex)
2  df.head()
```

| | sex | age | suicides_no | population | suicides_hk | generation |
|---|---|---|---|---|---|---|
| 0 | 1 | 15-24 years | 21 | 312900 | 6.71 | Generation X |
| 1 | 1 | 35-54 years | 16 | 308000 | 5.19 | Silent |
| 2 | 0 | 15-24 years | 14 | 289700 | 4.83 | Generation X |
| 3 | 1 | 75+ years | 1 | 21800 | 4.59 | G.I. Generation |
| 4 | 1 | 25-34 years | 9 | 274300 | 3.28 | Boomers |

Figure 5.5.4: Label Encoding ‘sex’ column

# CHAPTER 6

# FEATURE SELECTION

## 6.1 SELECT RELEVANT FEATURS FOR THE ANALYSIS:

The columns in the dataset are perfectly the features we need to analyse the data.

```
2 df.head(10)
```

| | country | year | sex | age | suicides_no | population | suicides_hk | generation |
|---|---|---|---|---|---|---|---|---|
| 0 | Albania | 1987 | male | 15-24 years | 21 | 312900 | 6.71 | Generation X |
| 1 | Albania | 1987 | male | 35-54 years | 16 | 308000 | 5.19 | Silent |
| 2 | Albania | 1987 | female | 15-24 years | 14 | 289700 | 4.83 | Generation X |
| 3 | Albania | 1987 | male | 75+ years | 1 | 21800 | 4.59 | G.I. Generation |
| 4 | Albania | 1987 | male | 25-34 years | 9 | 274300 | 3.28 | Boomers |
| 5 | Albania | 1987 | female | 75+ years | 1 | 35600 | 2.81 | G.I. Generation |
| 6 | Albania | 1987 | female | 35-54 years | 6 | 278800 | 2.15 | Silent |
| 7 | Albania | 1987 | female | 25-34 years | 4 | 257200 | 1.56 | Boomers |
| 8 | Albania | 1987 | male | 55-74 years | 1 | 137500 | 0.73 | G.I. Generation |
| 9 | Albania | 1987 | female | 5-14 years | 0 | 311000 | 0.00 | Generation X |

Figure 6.1.1 All Relevant Features

## 6.2 DROP IRRELEVANT FEATURES.

As all the features are necessary, the dataset doesn't contain any Irrelevant features. Therefore, there's no need of dropping any features.

## 6.3 TRAIN – TEST SPLIT:

Splitting the data into target set and feature set.

```
X=df.drop(["suicides_no"],axis=1)
X
```

| | sex | population | suicides_hk | Age_25-34 years | Age_35-54 years | Age_5-14 years | Age_55-74 years | Age_75+ years | Gen_G.I. Generation | Gen_Generation X | Gen_Generation Z | Gen_Millenials | Gen_Silent |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 312900 | 6.71 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 308000 | 5.19 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 289700 | 4.83 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 3 | 1 | 21800 | 4.59 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 4 | 1 | 274300 | 3.28 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 27815 | 0 | 3620833 | 2.96 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 27816 | 0 | 348465 | 2.58 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 27817 | 1 | 2762158 | 2.17 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 27818 | 0 | 2631600 | 1.67 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 27819 | 0 | 1438935 | 1.46 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

27820 rows × 13 columns

Figure 6.3.1 Feature column = X

```
y=df.suicides_no
y
```

```
0          21
1          16
2          14
3           1
4           9
         ...
27815     107
27816       9
27817      60
27818      44
27819      21
Name: suicides_no, Length: 27820, dtype: int32
```

Figure 6.3.2 Target column = y

Splitting the data into train and test

We are using the package sklearn.model_selection and importing train_test_split. This is used to split the feature and target data into training and testing data.

The test_size defines the percentage of data that goes into the testing data.Here, I've divided the train-test sets into 80:20 ratio. Which means 80% of the data is used for training the model. Whereas the other 20% is used to test the model.

Random_state is used to keep the once divided sets the same. Even if the command is run multiple times. Assigning any value to it makes it happen.

**Splitting the Input and Output 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 = 0.2, random_state = 0 )
```

```
X_train
```

| | sex | population | suicides_hk | Age_25-34 years | Age_35-54 years | Age_5-14 years | Age_55-74 years | Age_75+ years | Gen_G.I. Generation | Gen_Generation X | Gen_Generation Z | Gen_Millenials | Gen_Silent |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 26864 | 1 | 19715000 | 21.69 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 26120 | 0 | 1920400 | 24.63 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 7876 | 1 | 1241310 | 8.14 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2011 | 0 | 567665 | 5.64 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 21416 | 1 | 11467 | 17.44 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 13123 | 1 | 2873682 | 0.14 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 19648 | 1 | 665893 | 8.86 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 9845 | 0 | 4192492 | 18.22 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 10799 | 0 | 55825 | 0.00 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2732 | 1 | 97288 | 7.20 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

22256 rows × 13 columns

Figure 6.3.3: Train-Test-Split

## 6.4 FEATURE SCALING:

### Feature Scaling is used in KNN Regressor in my project.

1. We have to scale(FEATURE SCALING) the training and testing data inorder to fit the data into the model.

    a. **SCALING:** Scaling is done to the dataset when there are huge differences between the values of each cell.
    b. **Scaling** is generally done to datasets which have values in the terms of distance, etc
    c. There are different types of scalers used for Scaling:
        i. StandardScaler()
        ii. MinMaxScaler()
    d. StandardScaler is Generally used to Scale the data.
    e. StandardScaler is imported using sklearn.preprocessing package
    f. The output of the scaling is in the form of an array, which is further converted into a DataFrame
    g. Scaling is done in the following way:

```
1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3
4 scaled_X_train=scaler.fit_transform(X_train)
5 scaled_X_train
```

```
array([[ 0.99363992,  4.55345772,  0.46122516, ..., -0.23704073,
        -0.51475887, -0.54696001],
       [-1.00640079,  0.01998544,  0.61549062, ..., -0.23704073,
        -0.51475887, -0.54696001],
       [ 0.99363992, -0.15302411, -0.2497602 , ..., -0.23704073,
        -0.51475887, -0.54696001],
       ...,
       [-1.00640079,  0.59883891,  0.27914994, ..., -0.23704073,
        -0.51475887,  1.82828722],
       [-1.00640079, -0.45504626, -0.67687613, ..., -0.23704073,
        -0.51475887, -0.54696001],
       [ 0.99363992, -0.44448287, -0.29908317, ..., -0.23704073,
        -0.51475887, -0.54696001]])
```

Figure 6.4.1: Scaling Training Data

```
# scaling for train data we use fit_transform
# cuz in fit when it comes to the train it finds mean and variance
# based on mean and variance it will apply it to the train data

scaled_X_train=pd.DataFrame(scaler.fit_transform(X_train),
                            columns=X_train.columns)
scaled_X_train

# scaling for test data and for test we should only use transform
# cuz in fit when it comes to the train it finds mean and variance
# based on mean and variance it will apply it to the train data

scaled_X_test=pd.DataFrame(scaler.transform(X_test),
                            columns=X_train.columns)
scaled_X_test
```

| | sex | population | suicides_hk | Age_25-34 years | Age_35-54 years | Age_5-14 years | Age_55-74 years | Age_75+ years | Gen_G.I. Generation | Gen_Generation X | Gen_Generation Z | Gen_Millenials | G |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.993640 | 0.649616 | -0.135898 | -0.448563 | 2.251503 | -0.444293 | -0.446755 | -0.453109 | -0.333699 | -0.546059 | -0.237041 | -0.514759 | - |
| 1 | 0.993640 | -0.389451 | 1.567319 | -0.448563 | 2.251503 | -0.444293 | -0.446755 | -0.453109 | -0.333699 | 1.831305 | -0.237041 | -0.514759 | - |
| 2 | -1.006401 | 0.626535 | -0.630701 | -0.448563 | -0.444148 | -0.444293 | 2.238361 | -0.453109 | -0.333699 | -0.546059 | -0.237041 | -0.514759 | |
| 3 | 0.993640 | -0.090103 | 1.614543 | 2.229340 | -0.444148 | -0.444293 | -0.446755 | -0.453109 | -0.333699 | 1.831305 | -0.237041 | -0.514759 | - |
| 4 | 0.993640 | -0.466398 | -0.211456 | -0.448563 | -0.444148 | -0.444293 | -0.446755 | -0.453109 | -0.333699 | 1.831305 | -0.237041 | -0.514759 | - |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 5559 | -1.006401 | 1.622363 | 0.281774 | -0.448563 | 2.251503 | -0.444293 | -0.446755 | -0.453109 | -0.333699 | -0.546059 | -0.237041 | -0.514759 | - |
| 5560 | 0.993640 | -0.463622 | 0.270230 | -0.448563 | -0.444148 | -0.444293 | 2.238361 | -0.453109 | -0.333699 | -0.546059 | -0.237041 | -0.514759 | |

Figure 6.4.2: Converting The output array into a DataFrame(Both training and testing data)

# CHAPTER 7

# MODEL BUILDING AND EVALUATION

## 7.1 BRIEF ABOUT THE MODEL USED:

### 7.1.1. SIMPLE LINEAR REGRSSION:

**Linear Regression** is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.

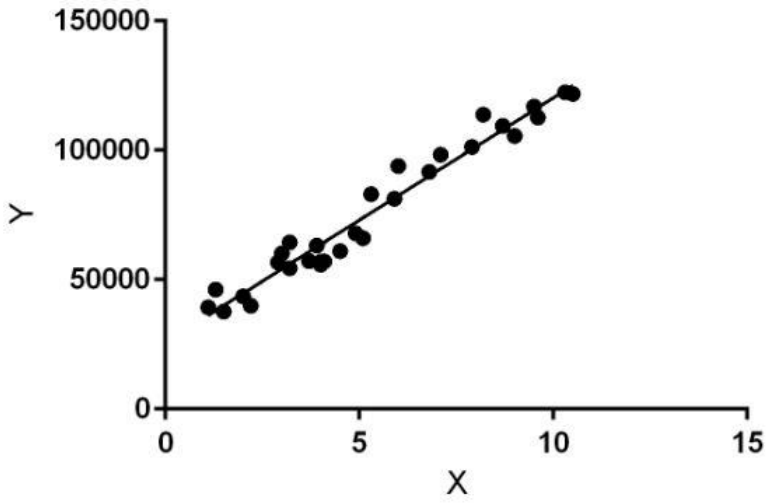


Figure 7.1.1.1: Simple linear plot

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). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

### 7.1.2. KNN REGRESSOR:

A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighbors. Another approach uses an inverse distance weighted average of the K nearest neighbors. KNN regression uses the same distance functions as KNN classification.

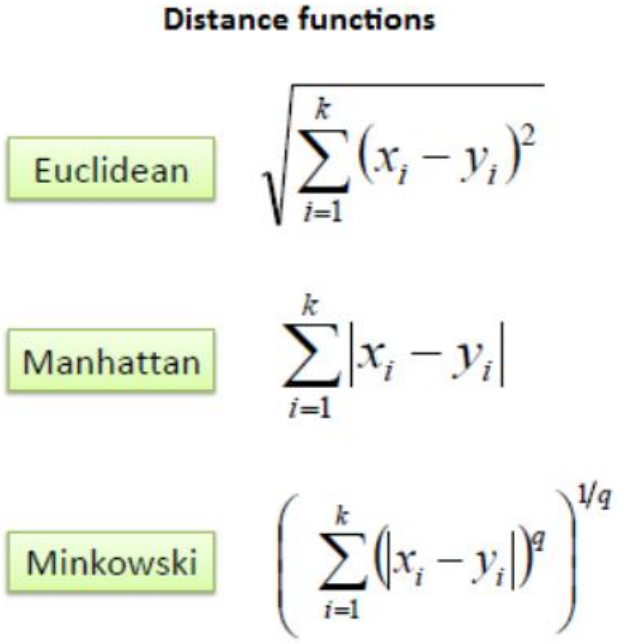


Figure 7.1.2.1 Distance functions

### 7.1.3 RANDOM FOREST REGRESSOR:

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap = True (default), otherwise the whole dataset is used to build each tree.

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

## 7.2 TRAIN THE MODLELS:

### 7.2.1 SIMPLE LINEAR REGRESSON:

We create an object for the model. Here we name it lm, then **Fit()** is used to fit the data into the selected model. Here, we are using X_train, y_train to fit it into the model.

And then next using predict() along with the object created(lm) to predict the training data's ouput using X_train. And feed it into a new variable y_pred

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression() # to consider it as a class give paranthesis
lm.fit(X_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
y_pred=lm.predict(X_train)
y_pred
```

```
array([2912.80483131,  289.98847866,  224.36274346, ...,  548.53240337,
       -319.27029888,  -45.06494951])
```

Figure 7.2.1.1 Training the data with Simple Linear Regressor

### 7.2.2 KNN REGRESSOR:

This method is available in sklearn.neighbors package.

So to use this method we have to import KneighborsRegressor from sklearn.neighbors.

Now, we train the Scaled data into model. And predict too.

We are taking n_neighbors as 3. Later we calculate what's the best fit to maximize the metrics. So the prediction comes out accurate.

**'N_neighbors' –** The count of neighbors , an unknown value takes when it takes the neighbors into consideration to find it's own value.

```
from sklearn.neighbors import KNeighborsRegressor
knn=KNeighborsRegressor(n_neighbors=3,metric="euclidean")
# applying on the the data set
knn.fit(scaled_X_train,y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='euclidean',
                    metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                    weights='uniform')
```

```
# predictions on the data
# predict function
# syntax objectname.predict(input)
y_train_predict=knn.predict(scaled_X_train)
y_train_predict
```

```
array([4234.33333333,  466.66666667,  102.66666667, ...,  761.        ,
          0.        ,    5.33333333])
```

Figure 7.2.2.1 Training the scaled training data

### 7.2.3 RANDOM FOREST REGRESSOR:

```
#import,initialize,fit

#importing
from sklearn.ensemble import RandomForestRegressor

#create an object for this
rfr = RandomForestRegressor(n_estimators=40)
#n_estimators-> Builds 40 dtrees

#Fit the random forest classifier to the training data
rfr.fit(X_train,y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=40,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

```
y_pred_train = rfr.predict(X_train)
```

Figure 7.2.3.1: Training data with RFR

## 7.3 VALIDATE THE MODELS:

Validating the models based on r2 and RMSE scores.

### 7.3.1 SIMPLE LINEAR REGRESSION:

**Checking accuracy using r2_score:**

```
from sklearn.metrics import r2_score
r2_score(y_train,y_pred)
```

```
0.4682965663756704
```

Figure 7.3.1.1: r2_score for linear regression.

The r2_score for training data came out to be 46%, so there is no point in calculating r2_score for testing data.

### 7.3.2 KNN REGRESSOR:

```
print('R^2:', r2_score(y_train,y_train_predict))
print('Adjusted R^2:', 1- (1-r2_score(y_train, y_train_predict))*(len(scaled_X_train)-1)/
                          (len(scaled_X_train)-scaled_X_train.shape[1]-1))
print("MAE",mean_absolute_error(y_train,y_train_predict))

print("MSE",mean_squared_error(y_train,y_train_predict))

print("RMSE",np.sqrt(mean_squared_error(y_train,y_train_predict)))
```

```
R^2: 0.997424078922969
Adjusted R^2: 0.9974225733490997
MAE 10.607461658279416
MSE 2239.6723829778743
RMSE 47.325177051732986
```

Figure 7.3.2.1 Metrics for KNN training data

As the r2_score and RMSE score came out to be 99.7% percent and 47.3I've done prediction on testing data too.

The following are the outputs:

```
## Test the model on testing data
y_test_predict =knn.predict(scaled_X_test) # test data-- unseen data
y_test_predict
```

```
array([450.66666667, 118.66666667,  45.33333333, ..., 211.66666667,
         5.        ,   8.33333333])
```

**Calculating the metrics for Testing data:**

```
## we need to compare the actual values(y_test) and predicted values(y_test_pred)
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error

print('R^2:', r2_score(y_test,y_test_predict))

print('Adjusted R^2:', 1- (1-r2_score(y_test, y_test_predict))*(len(scaled_X_test)-1)/
                                    (len(scaled_X_test)-scaled_X_test.shape[1]-1))

print("MAE",mean_absolute_error(y_test,y_test_predict))

print("MSE",mean_squared_error(y_test,y_test_predict))

print("RMSE",np.sqrt(mean_squared_error(y_test,y_test_predict)))
```

```
R^2: 0.9901550415238197
Adjusted R^2: 0.9901319812607223
MAE 17.510603882099208
MSE 5811.543553798228
RMSE 76.23348053052692
```

Figure 7.3.2.2 Metric for KNN testing data

Now after calculating the values and metrics. We find the best fit n_neighbors for the model and data. Which is popularly known as K-FOLD VALUE.

We build a program to find out the accuracy scores(r2_score) for the range of values we assign to the program. Int the following way:

```
scores=[]
# checking for optimum k value
# building the model with multiple k values
for k in range(1,20):
    knn_model=KNeighborsRegressor(n_neighbors=k)
    knn_model.fit(scaled_X_train,y_train)
    pred_test=knn_model.predict(scaled_X_test)
    scores.append(r2_score(y_test,pred_test))
scores
```

```
[0.9952600229474291,
 0.9949236985329581,
 0.9901550415238197,
 0.9869422611280094,
 0.9832721545031687,
 0.9786178978078205,
 0.9719673331113022,
 0.9694582993086969,
 0.9622053533832136,
 0.9565847475681516,
 0.9560373574604122,
 0.9528062213082739,
 0.9497748707712066,
 0.9440359997611577,
 0.9382935614107898,
 0.9329280190402246,
 0.9272256470134019,
 0.9214829377799609,
 0.914574051345229]
```

Figure 7.3.2.3: K-FOLD

```
# plot of k-values and scores
plt.plot(range(1,20),scores,marker='o',markerfacecolor="r",linestyle="--")
```

```
[<matplotlib.lines.Line2D at 0x244f3213c88>]
```

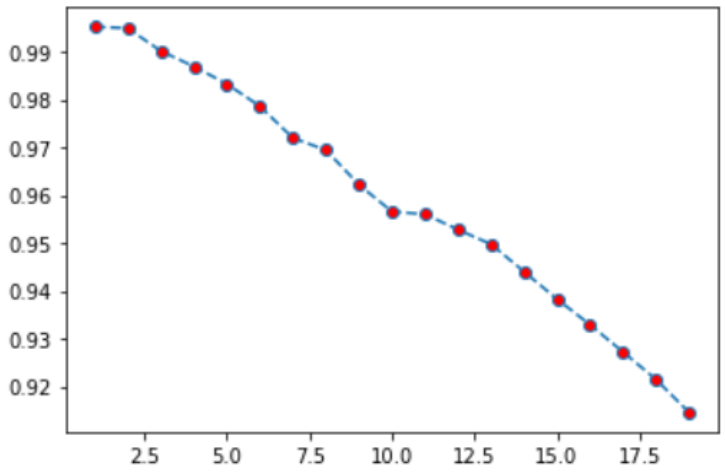


Figure 7.3.2.4: Plot for analysing the best K value

We found out that k=2 gives more accurate value.

Now we have to fit, predict and find the Metrics for train, test sets.

```
# optimun k value is 2
final_model=KNeighborsRegressor(n_neighbors=2,metric="euclidean")
final_model.fit(scaled_X_train,y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='euclidean',
                    metric_params=None, n_jobs=None, n_neighbors=2, p=2,
                    weights='uniform')
```

```
# predicting on the training data
final_train_predict=final_model.predict(scaled_X_train)
final_train_predict
```

```
array([4271.5,  474.5,  104.5, ...,  768.5,    0. ,    5.5])
```

Figure: 7.3.2.5: Final model fitting, predicting

```
print('R^2:', r2_score(y_train,final_train_predict))
print('Adjusted R^2:', 1- (1-r2_score(y_train, final_train_predict))*(len(scaled_X_train)-1)/
                                    (len(scaled_X_train)-scaled_X_train.shape[1]-1))
```

```
R^2: 0.9987914456393199
Adjusted R^2: 0.9987907392636932
```

```
## regplot(y_train --> actual values adn y_train_pred )
sns.regplot(y_train,final_train_predict)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x244f833da20>
```

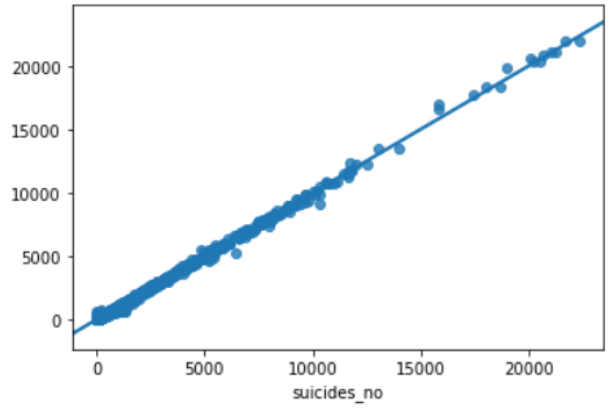


Figure 7.3.2.6: Metrics, analysing the final model training data

```
## Test the model on testing data
final_test_predict =final_model.predict(scaled_X_test) # test data-- unseen data
final_test_predict
```

```
array([489. , 117.5,  45. , ..., 239. ,   5. ,   8.5])
```

### Calculating the metrics for testing data of Final Model :

```
## we need to compare the actual values(y_test) and predicted values(y_test_pred)
print('R^2:', r2_score(y_test,final_test_predict))

print('Adjusted R^2:', 1- (1-r2_score(y_test, final_test_predict))*(len(scaled_X_test)-1)/
                                    (len(scaled_X_test)-scaled_X_test.shape[1]-1))

print("MAE",mean_absolute_error(y_test,final_test_predict))

print("MSE",mean_squared_error(y_test,final_test_predict))

print("RMSE",np.sqrt(mean_squared_error(y_test,final_test_predict)))
```

```
R^2: 0.9949236985329581
Adjusted R^2: 0.9949118080970893
MAE 14.634615384615385
MSE 2996.5740474478794
RMSE 54.74097229176588
```

Figure 7.3.2.7: Predict, metrics for final model testing data

### 7.3.3 RANDOM FOREST REGRESSOR:

Calculating the metrics of train data, And predicting using test data, also calculating the r2_score for the test data

**Calculating r2_score for training data:**

```
[ ]: 1 r2_score(y_train,y_pred_train)

[ ]: 0.9995592194921784

[ ]: 1 y_pred_test = rfr.predict(X_test)
```

**Calculating r2_score for testing data:**

```
[ ]: 1 r2_score(y_test,y_pred_test)

[ ]: 0.9985412936800853
```

Figure 7.3.3.1: metrics for train and predict, metrics for testing data

Visualizing the train and test r2 scores for all the models.

## Visualizing the every model's r2_scores.

### For Train r2_score

```
model=['LR','KNNR','RFR']
r2 = [0.4682965663756704,0.9987914456393199,0.9996278524822058]
plt.bar(model,r2,color='green')
plt.title("Visualizing r2 scores for model's training data")
plt.xlabel('models used')
plt.ylabel('r2 score')
plt.show()
```

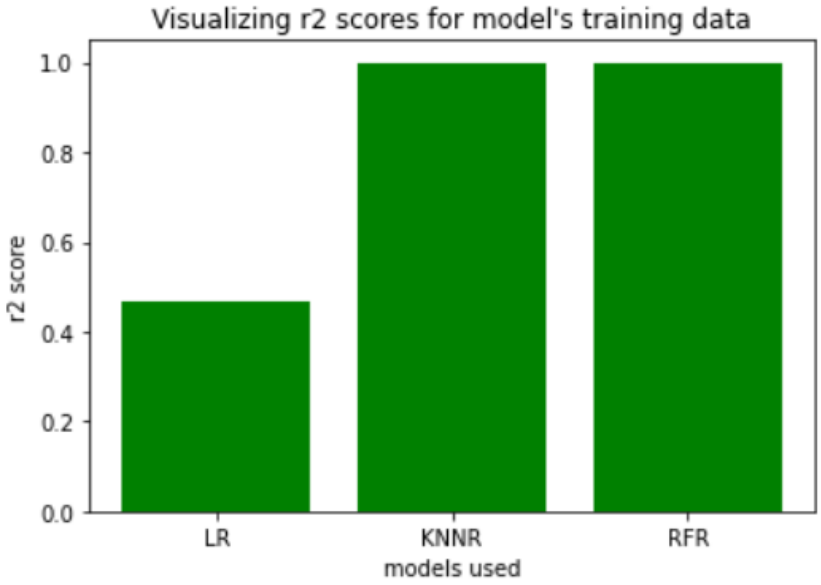


Figure 7.3.3.2 Visualizing train data scores for all models

## For Testing r2_score:

```
model=['LR','KNNR','RFR']
r2 = [0.5210149348104174,0.9949236985329581,0.9986249859145168]
plt.bar(model,r2,color='green')
plt.title("Visualizing r2 scores for model's training data")
plt.xlabel('models used')
plt.ylabel('r2 score')
plt.show()
```

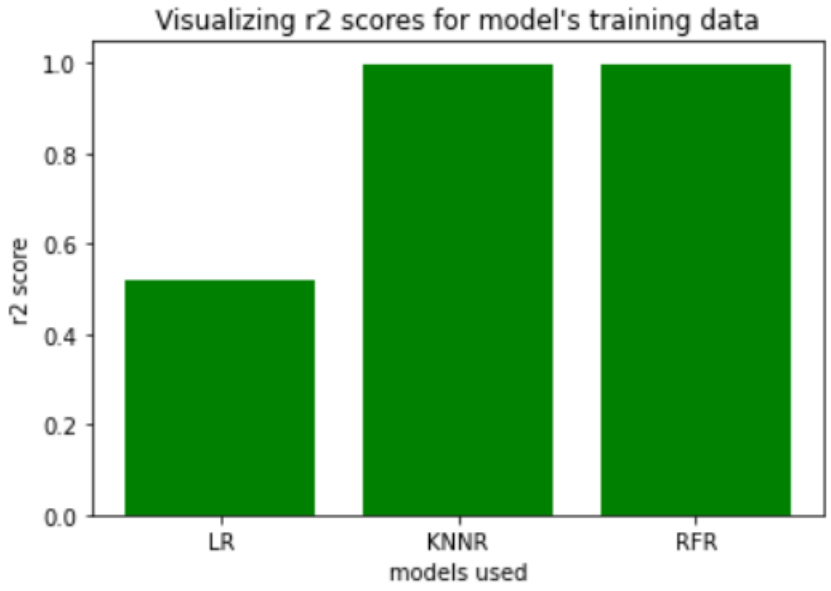


Figure 7.3.3.3 Visualizing test data scores for all models

SO, AS WE GOT THE HIGHEST R2_SCORE FOR RANDOM FOREST REGRESSOR. THAT IS

0.9995592194921784: For training data
0.9985412936800853: For testing data

WE PREDICT USING RANDOM FOREST REGRESSOR ONLY

## 7.4 MAKE PREDICTIONS:

Making Predictions using Random Forest Regressor

**Predicting using Random Forest Because it gave the highest r2_score:**

**Using dataframe values:**

```
1 print(rfr.predict([[0,278800,2.15,0,1,0,0,0,0,0,0,0,1]]))
```

```
[6.]
```

```
1 print(rfr.predict([[1,21800,4.59,0,0,0,0,1,1,0,0,0,0]]))
```

```
[1.]
```

```
1 print(rfr.predict([[0,289700,4.83,0,0,0,0,0,0,1,0,0,0]]))
```

```
[14.05]
```

Figure 7.4.1 Predictions using DataFrame Data

The Predictions came out to be 99% accurate

## 7.5 PREDICTIONS FROM RAW DATA:

Predicting using raw data:

**Using Random values which are not in the dataframe:**

```
1 print(rfr.predict([[1,20645,11.0,0,1,0,0,0,0,0,0,0,1]]))
```

```
[2.225]
```

```
1 print(rfr.predict([[0,21945,19.0,0,0,1,0,0,0,1,0,0,0]]))
```

```
[4.25]
```

```
1 print(rfr.predict([[1,22095,9.0,0,0,1,0,0,0,0,1,0,0]]))
```

```
[2.025]
```

Figure 7.5.1 Prediction using raw data

# CONCLUSION

It is concluded after performing thorough Exploratory Data analysis which include Stats models which are computed to get accuracy and also Heat maps which are computed to get a clear understanding of the data set (which parameter has most abundant effect on the study case) and its come to point of getting the solution for the problem statement being , that suicide rates are increasing as the population is increasing. The Male citizen tend to consider committing suicide more that the female citizen.

Thus, we can conclude that based on the parameters such as age, generation, population, the suicide rates are being affected. So as a solution, citizen need to think and take decisions. Rather than taking decisions at the stop.

# REFERENCES


1. https://expertsystem.com/machine-learning-definition/#:~:text=Machine%20learning%20is%20an%20application,use%20it%20learn%20for%20themselves.

2. https://en.wikipedia.org/wiki/Machine_learning

3. https://en.wikipedia.org/wiki/Deep_learning

4. geeksforgeeks.org/ml-linear-regression/

5. https://www.geeksforgeeks.org/random-forest-regression-in-python/

6. https://www.geeksforgeeks.org/k-nearest-neighbors-with-python-ml/