Suicide Rate Prediction

Iftehaz Newaz

Introduction

Suicide can have a significant impact on individuals, families, and communities, resulting in emotional, social, and economic costs. Preventing the far-reaching impact of suicide and promoting mental health and wellbeing in individuals and communities require addressing the root causes of suicide and providing effective support and resources to those at risk. This often raises questions such as the reasons for suicide, the type of analysis to be conducted, and the most important features to consider.

The datasets were collected to develop prediction models for suicide rate likelihood regression in a given country and year. However, the variables included in the datasets can be used beyond the scope of suicide rate prediction due to their characteristics.

The report uses PySpark, a machine learning tool, to analyze suicide rate data from 1985 to 2016. The analysis includes visualization in Tableau with the aim of developing a robust and accurate model for suicide rate prediction by considering relevant factors.

The focus of this project is to predict suicide rate, which is approached as a regression problem. Various algorithms, including Linear Regression and Factorization Machines regressor models, have been considered for predicting suicide rates.
Dataset Link

In the end, I attempt to solve those questions. This is my perspective of the outcome, it's based on the dataset itself. The current world scenario might be different from it.

Related Work

Suicide is a complex and multi-factorial phenomenon, and as such, there is a vast body of literature focused on the prediction of suicide rates. In this section, we will review some of the most relevant studies in the field, grouped by the main approaches used in suicide rate prediction.

One of the most traditional approaches to suicide rate prediction is the use of demographic and epidemiological data. In this sense, numerous studies have analyzed the association between suicide rates and factors such as age, gender, socio-economic status, unemployment rates, and access to mental health services. For example, a study conducted by Qin and colleagues (2020) in China found that unemployment, low income, and being unmarried were significant predictors of suicide rates, while another study by Cheng and colleagues (2019) in the United States found that higher levels of gun ownership were associated with increased suicide rates.

Another approach to suicide rate prediction is the use of machine learning algorithms. Several studies have explored the use of these techniques to predict suicide rates based on different types of data, such as social media activity, search engine queries, and electronic health records. For example, a study by Tsugawa and colleagues (2019) in Japan developed a machine learning model that analyzed social media posts to predict suicide risk in individuals. Similarly, a study by Simon and colleagues (2018) in the United States used electronic health records to develop a machine learning model that predicted suicide risk in veterans.

some studies have focused on the use of network analysis to understand the complex interplay of factors that contribute to suicide rates. For example, a study by Cao and colleagues (2021) in China analyzed the co-occurrence patterns of mental health disorders and identified key nodes in the network that were associated with suicide risk. Similarly, a study by Kim and colleagues (2018) in South Korea used a network analysis approach to identify the most influential risk factors for suicide, such as depression and previous suicide attempts.

The studies reviewed in this section highlight the importance of considering demographic, epidemiological, and social factors, as well as the potential of machine learning and network analysis techniques to improve suicide rate prediction.

Implementation

➢ Apache Spark

In recent months, major technological companies have frequently used the term "Big Data." Big Data can be characterized by its size, processing power, efficiency, and the fact that the data often comes in an unstructured manner, making it challenging and expensive to manage. Apache Spark, a distributed memory system, comes into play as it maintains data on top of the Hadoop File System (HDFS), providing better stability than Hadoop MapReduce. To sum up, Apache Spark is an open-source solution for processing Big Data that offers a range of capabilities, such as SQL queries, machine learning, and continuous data streaming, all within a single system Apache Spark Language Support:

- Apache Spark (Java and Scala Support)
- **PySpark (Python Support)**
- SparkR (R support)

Note, this report only contains information about PySpark.

➢ PySpark:

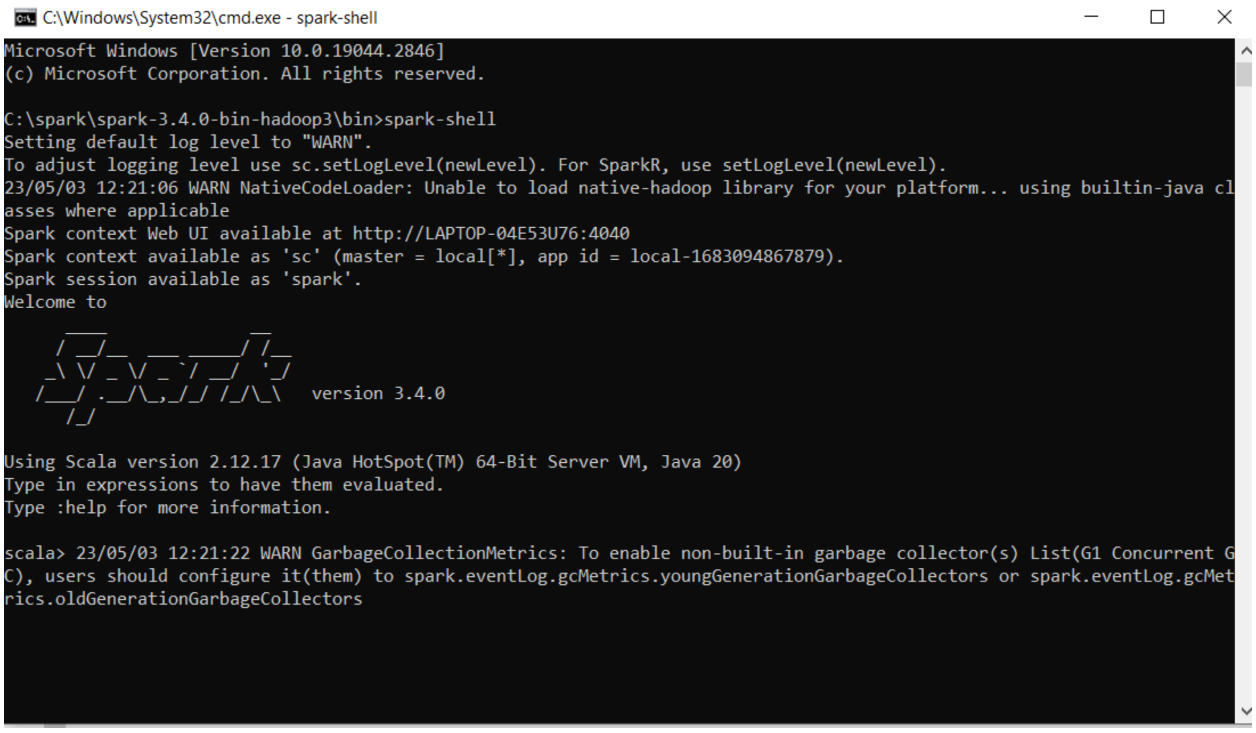
PySpark, a library for the Python programming language, supports all of Spark's functionalities, such as SQL, streaming, and machine learning via the Pandas API. It serves as the interface between Spark and Python and can interact with Jupyter for machine learning and complex data analysis. This investigation focuses on the installation process of PySpark in JupyterLab. However, it is crucial to note that PySpark's support is limited to the Pandas API and does not include other libraries such as NumPy, which is a disadvantage.

Also, I used PySpark in the project of Suicide rate prediciton analysis with machine learning algorithm and preprocessing.

I have been using Jupyter Notebook for the past two years for machine learning as well as deep learning. So, I have used Jupyter Notebook as my working platform. It's easy to use and easy to install packages.

- **First, we have to install JDK 20 LINK**
- **Secondly, Download Spark LINK**
- **Finally, Install Python Latest version LINK**

Install everything on 'C' Drive, Set environment variables according to directory above softwares are installed. Go to the command prompt and type "spark-shell", your spark connection will be established.



```
C:\Windows\System32\cmd.exe - spark-shell
Microsoft Windows [Version 10.0.19044.2846]
(c) Microsoft Corporation. All rights reserved.

C:\spark\spark-3.4.0-bin-hadoop3\bin>spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/05/03 12:21:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Spark context Web UI available at http://LAPTOP-04E53U76:4040
Spark context available as 'sc' (master = local[*], app id = local-1683094867879).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.4.0
      /_/

Using Scala version 2.12.17 (Java HotSpot(TM) 64-Bit Server VM, Java 20)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 23/05/03 12:21:22 WARN GarbageCollectionMetrics: To enable non-built-in garbage collector(s) List(G1 Concurrent G
C), users should configure it(them) to spark.eventLog.gcMetrics.youngGenerationGarbageCollectors or spark.eventLog.gcMet
rics.oldGenerationGarbageCollectors
```

Fig 1.1: Installing pyspark

Fortunately I did not get any errors.

- ## Dataset:

The dataset is taken from Kaggle, which is one of the free resources for datasets. This dataset is all about suicide rate prediction. Dataset contains information from 1985 to 2016. This dataset contains 12 columns and 27820 rows. All these columns contain information about every specific accommodation place. A few important features are country, year, sex, age group, count of suicides, population, suicide rate, generation (based on age grouping average), etc. I will show step-by-step details of this dataset

<u>Country:</u> Name of the country where suicide happened.

<u>Year:</u> At which year the suicide happened.

<u>Sex</u>: Gender.

<u>Age:</u> Divided the age into few groups.

<u>Suicides_no:</u> Number of suicide took place on that year in that country.

<u>Population:</u> Population of that country on that year.

<u>suicides/100k pop:</u> Rate of suicide per 100k population

<u>country-year:</u> Amalgamated the year and country.

<u>HDI for year:</u> Human Development index for that year.

<u>gdp_for_year ($):</u> GDP for that year of that country.

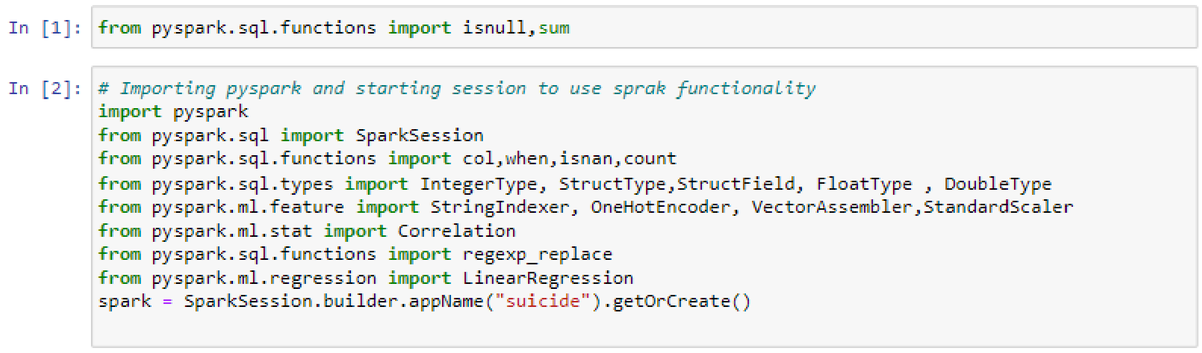
<u>gdp_per_capita ($):</u> GDP per capita for that year of that country.

<u>generation:</u> Based on age group the generation is picked.

I covered all attributes/columns and described its values.

• Evaluation and Machine Learning approaches:

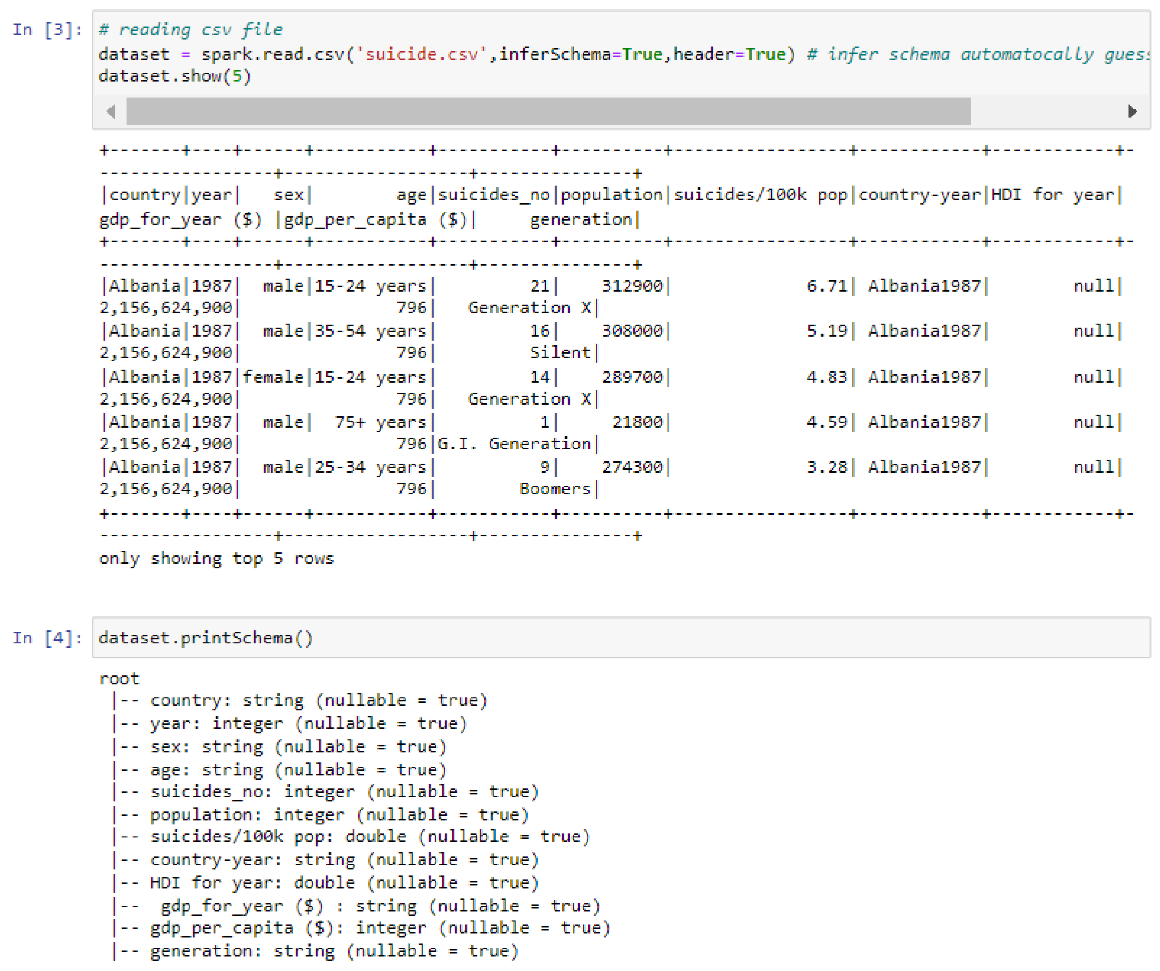
At first, I imported the necessary library. Then, I created a SparkSession known as "suicide". With PySpark and SQL, Machine learning model works better with full efficiency. Here is a screenshot of my code.



```
In [1]: from pyspark.sql.functions import isnull,sum

In [2]: # Importing pyspark and starting session to use sprak functionality
        import pyspark
        from pyspark.sql import SparkSession
        from pyspark.sql.functions import col,when,isnan,count
        from pyspark.sql.types import IntegerType, StructType,StructField, FloatType , DoubleType
        from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler,StandardScaler
        from pyspark.ml.stat import Correlation
        from pyspark.sql.functions import regexp_replace
        from pyspark.ml.regression import LinearRegression
        spark = SparkSession.builder.appName("suicide").getOrCreate()
```

Fig 1.2 Importing library

I have kept the dataset on the same directory where the notebook file is saved.



```
In [3]: # reading csv file
        dataset = spark.read.csv('suicide.csv',inferSchema=True,header=True) # infer schema automatocally gues:
        dataset.show(5)
```

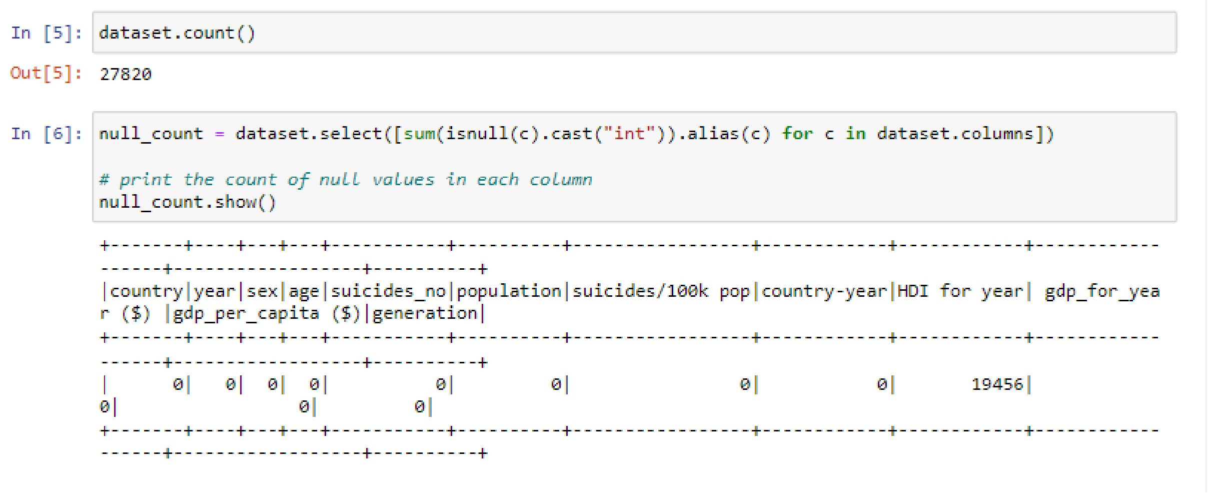
```
+-------+----+------+-----------+-----------+----------+-----------------+------------+------------+-
-----------------+------------------+---------------+
|country|year|   sex|        age|suicides_no|population|suicides/100k pop|country-year|HDI for year|
gdp_for_year ($) |gdp_per_capita ($)|     generation|
+-------+----+------+-----------+-----------+----------+-----------------+------------+------------+-
-----------------+------------------+---------------+
|Albania|1987|  male|15-24 years|         21|    312900|             6.71| Albania1987|        null|
2,156,624,900|               796|   Generation X|
|Albania|1987|  male|35-54 years|         16|    308000|             5.19| Albania1987|        null|
2,156,624,900|               796|         Silent|
|Albania|1987|female|15-24 years|         14|    289700|             4.83| Albania1987|        null|
2,156,624,900|               796|   Generation X|
|Albania|1987|  male|  75+ years|          1|     21800|             4.59| Albania1987|        null|
2,156,624,900|               796|G.I. Generation|
|Albania|1987|  male|25-34 years|          9|    274300|             3.28| Albania1987|        null|
2,156,624,900|               796|        Boomers|
+-------+----+------+-----------+-----------+----------+-----------------+------------+------------+-
-----------------+------------------+---------------+
only showing top 5 rows
```

```
In [4]: dataset.printSchema()
```

```
root
 |-- country: string (nullable = true)
 |-- year: integer (nullable = true)
 |-- sex: string (nullable = true)
 |-- age: string (nullable = true)
 |-- suicides_no: integer (nullable = true)
 |-- population: integer (nullable = true)
 |-- suicides/100k pop: double (nullable = true)
 |-- country-year: string (nullable = true)
 |-- HDI for year: double (nullable = true)
 |--  gdp_for_year ($) : string (nullable = true)
 |-- gdp_per_capita ($): integer (nullable = true)
 |-- generation: string (nullable = true)
```

Fig 1.3 Loading the dataset

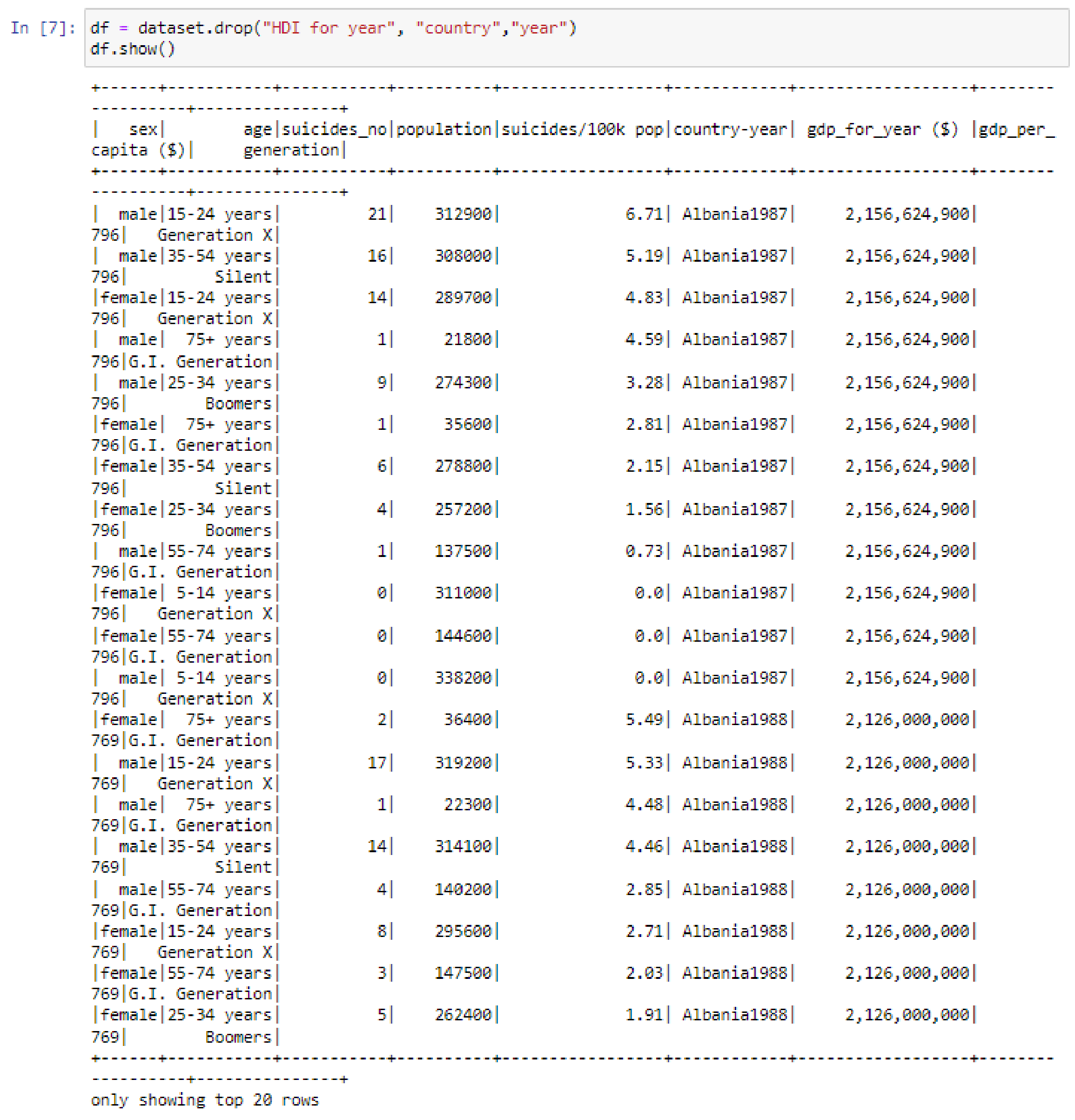
After that, I loaded the dataset and showed top 5 rows and printed the schema of the dataset like an SQL table. It contains columns name, and data types and checks whether the ISnullable or not.



```
In [5]: dataset.count()

Out[5]: 27820

In [6]: null_count = dataset.select([sum(isnull(c).cast("int")).alias(c) for c in dataset.columns])

        # print the count of null values in each column
        null_count.show()

+-------+----+---+---+-----------+----------+-----------------+------------+------------+------------
------+------------------+----------+
|country|year|sex|age|suicides_no|population|suicides/100k pop|country-year|HDI for year| gdp_for_yea
r ($) |gdp_per_capita ($)|generation|
+-------+----+---+---+-----------+----------+-----------------+------------+------------+------------
------+------------------+----------+
|      0|   0|  0|  0|          0|         0|                0|           0|       19456|
0|                 0|         0|
+-------+----+---+---+-----------+----------+-----------------+------------+------------+------------
------+------------------+----------+
```

Fig 1.4 Checking for null values

My next target was to check for null values in every column. I have filtered the data with count to check all the null values. The key is the column name, and the value is the total amount of null values in that column. C is a predefined keyword that stands for SQL Column. Here, we can see that “HDI for year” column has 19456 null values which is greater than half of the total examples.

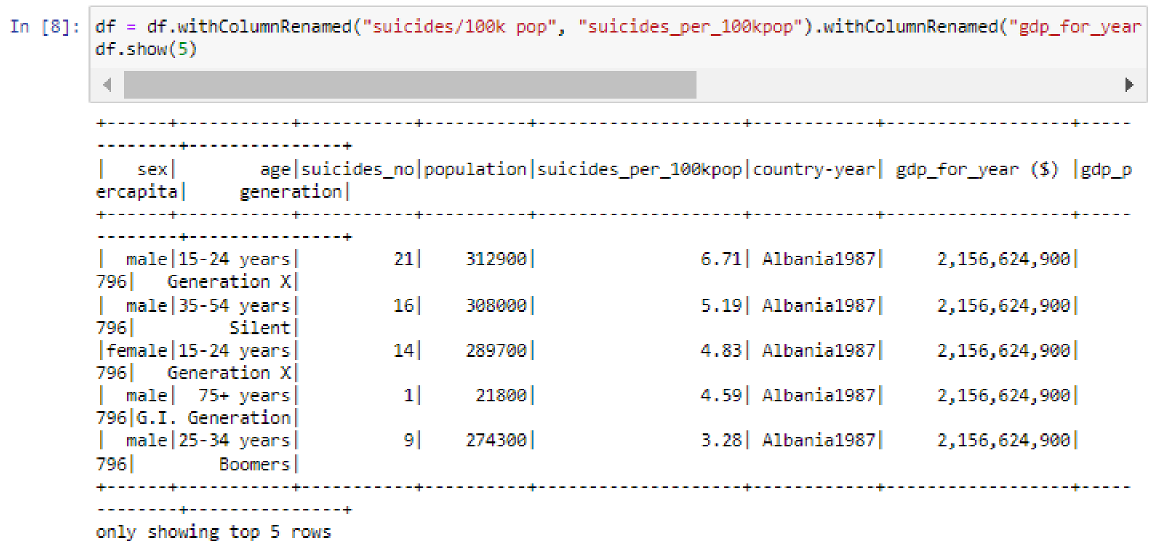


```
In [7]: df = dataset.drop("HDI for year", "country","year")
        df.show()
```

```
+------+-----------+-----------+----------+----------------+------------+------------------+--------
----------+---------------+
|   sex|        age|suicides_no|population|suicides/100k pop|country-year| gdp_for_year ($) |gdp_per_
capita ($)|     generation|
+------+-----------+-----------+----------+----------------+------------+------------------+--------
----------+---------------+
|  male|15-24 years|         21|    312900|            6.71| Albania1987|     2,156,624,900|
796|    Generation X|
|  male|35-54 years|         16|    308000|            5.19| Albania1987|     2,156,624,900|
796|          Silent|
|female|15-24 years|         14|    289700|            4.83| Albania1987|     2,156,624,900|
796|    Generation X|
|  male|  75+ years|          1|     21800|            4.59| Albania1987|     2,156,624,900|
796|G.I. Generation|
|  male|25-34 years|          9|    274300|            3.28| Albania1987|     2,156,624,900|
796|         Boomers|
|female|  75+ years|          1|     35600|            2.81| Albania1987|     2,156,624,900|
796|G.I. Generation|
|female|35-54 years|          6|    278800|            2.15| Albania1987|     2,156,624,900|
796|          Silent|
|female|25-34 years|          4|    257200|            1.56| Albania1987|     2,156,624,900|
796|         Boomers|
|  male|55-74 years|          1|    137500|            0.73| Albania1987|     2,156,624,900|
796|G.I. Generation|
|female| 5-14 years|          0|    311000|             0.0| Albania1987|     2,156,624,900|
796|    Generation X|
|female|55-74 years|          0|    144600|             0.0| Albania1987|     2,156,624,900|
796|G.I. Generation|
|  male| 5-14 years|          0|    338200|             0.0| Albania1987|     2,156,624,900|
796|    Generation X|
|female|  75+ years|          2|     36400|            5.49| Albania1988|     2,126,000,000|
769|G.I. Generation|
|  male|15-24 years|         17|    319200|            5.33| Albania1988|     2,126,000,000|
769|    Generation X|
|  male|  75+ years|          1|     22300|            4.48| Albania1988|     2,126,000,000|
769|G.I. Generation|
|  male|35-54 years|         14|    314100|            4.46| Albania1988|     2,126,000,000|
769|          Silent|
|  male|55-74 years|          4|    140200|            2.85| Albania1988|     2,126,000,000|
769|G.I. Generation|
|female|15-24 years|          8|    295600|            2.71| Albania1988|     2,126,000,000|
769|    Generation X|
|female|55-74 years|          3|    147500|            2.03| Albania1988|     2,126,000,000|
769|G.I. Generation|
|female|25-34 years|          5|    262400|            1.91| Albania1988|     2,126,000,000|
769|         Boomers|
+------+-----------+-----------+----------+----------------+------------+------------------+--------
----------+---------------+
only showing top 20 rows
```

Fig 1.5 Dropping all the null values

Now, it’s needed to drop the column “HDI for year” since it has large number of null values, also dropped “country” and “year” column as we have another column named “country-year”.

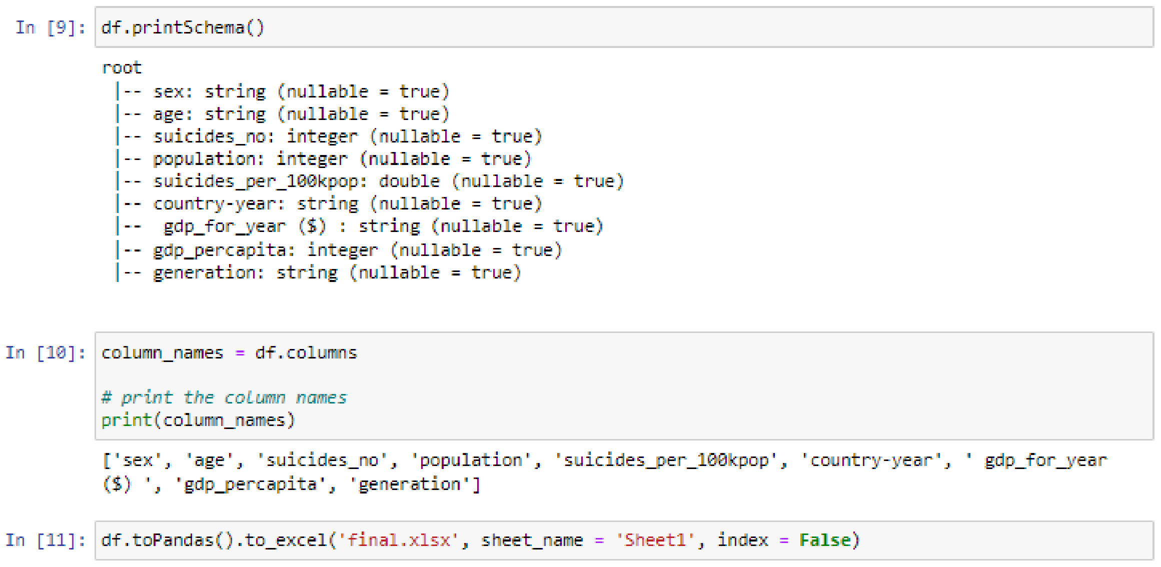


```
In [8]: df = df.withColumnRenamed("suicides/100k pop", "suicides_per_100kpop").withColumnRenamed("gdp_for_year
        df.show(5)
```

```
+------+-----------+-----------+----------+--------------------+------------+--------------------+-----
--------+---------------+
|   sex|        age|suicides_no|population|suicides_per_100kpop|country-year| gdp_for_year ($) |gdp_p
ercapita|     generation|
+------+-----------+-----------+----------+--------------------+------------+--------------------+-----
--------+---------------+
|  male|15-24 years|         21|    312900|                6.71| Albania1987|      2,156,624,900|
796|    Generation X|
|  male|35-54 years|         16|    308000|                5.19| Albania1987|      2,156,624,900|
796|          Silent|
|female|15-24 years|         14|    289700|                4.83| Albania1987|      2,156,624,900|
796|    Generation X|
|  male|  75+ years|          1|     21800|                4.59| Albania1987|      2,156,624,900|
796|G.I. Generation|
|  male|25-34 years|          9|    274300|                3.28| Albania1987|      2,156,624,900|
796|         Boomers|
+------+-----------+-----------+----------+--------------------+------------+--------------------+-----
--------+---------------+
only showing top 5 rows
```

Fig 1.6 Renaming Columns

Here I have renamed few long columns so that it becomes readable.



```
In [9]: df.printSchema()

root
 |-- sex: string (nullable = true)
 |-- age: string (nullable = true)
 |-- suicides_no: integer (nullable = true)
 |-- population: integer (nullable = true)
 |-- suicides_per_100kpop: double (nullable = true)
 |-- country-year: string (nullable = true)
 |--  gdp_for_year ($) : string (nullable = true)
 |-- gdp_percapita: integer (nullable = true)
 |-- generation: string (nullable = true)

In [10]: column_names = df.columns

         # print the column names
         print(column_names)

['sex', 'age', 'suicides_no', 'population', 'suicides_per_100kpop', 'country-year', ' gdp_for_year
($) ', 'gdp_percapita', 'generation']

In [11]: df.toPandas().to_excel('final.xlsx', sheet_name = 'Sheet1', index = False)
```

Fig 1. 7 Exporting filtered data

Here, I have printed the final schema, printed the final remaining column on which the algorithms will be implemented and finally saved the filtered data.

- **Visualization Of Clean Data:**

  For visualization, we need to use Tableau

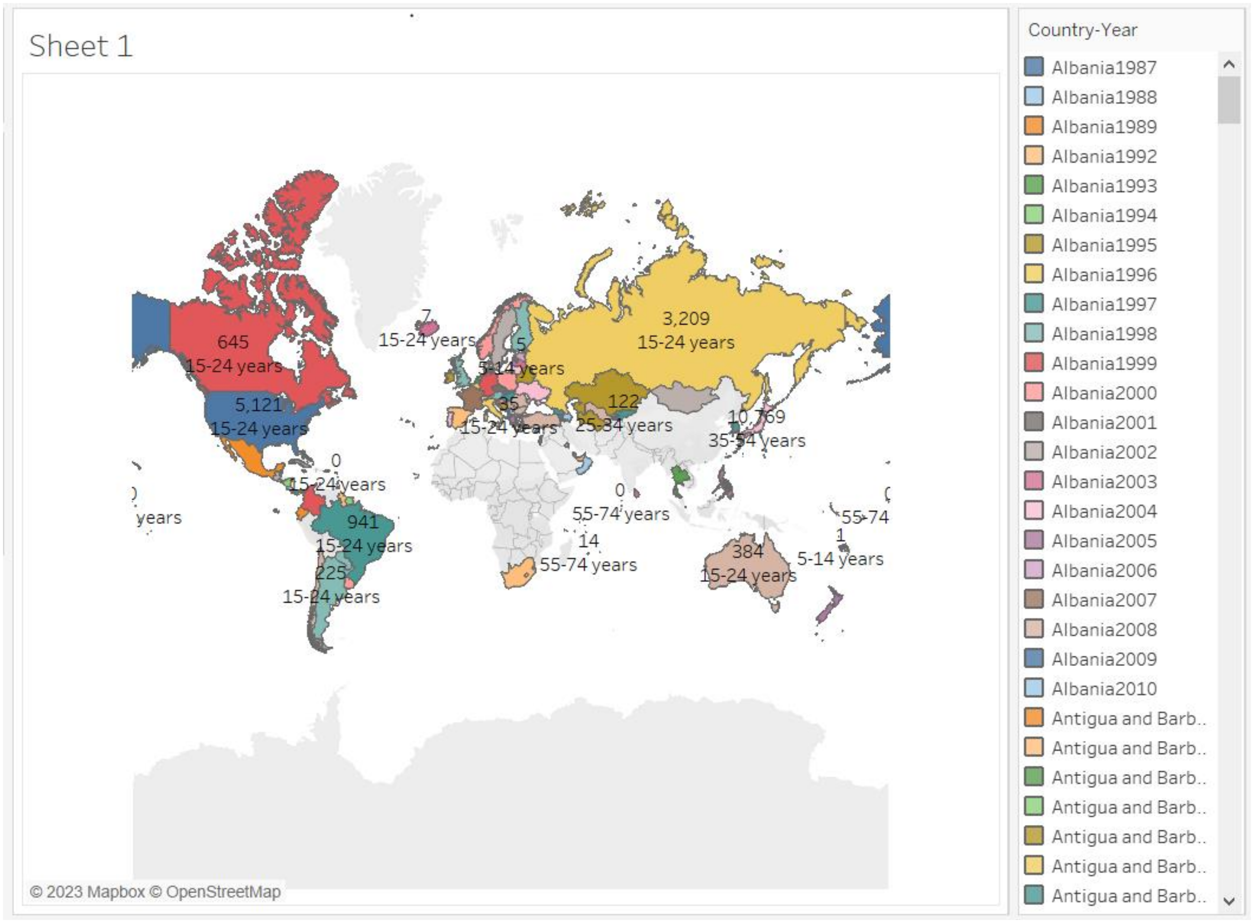


Fig 2.1 Map, number of suicide taking place based on the age group in different country.

This simple map analysis shows the number of suicide taking place based on the age group in different country.

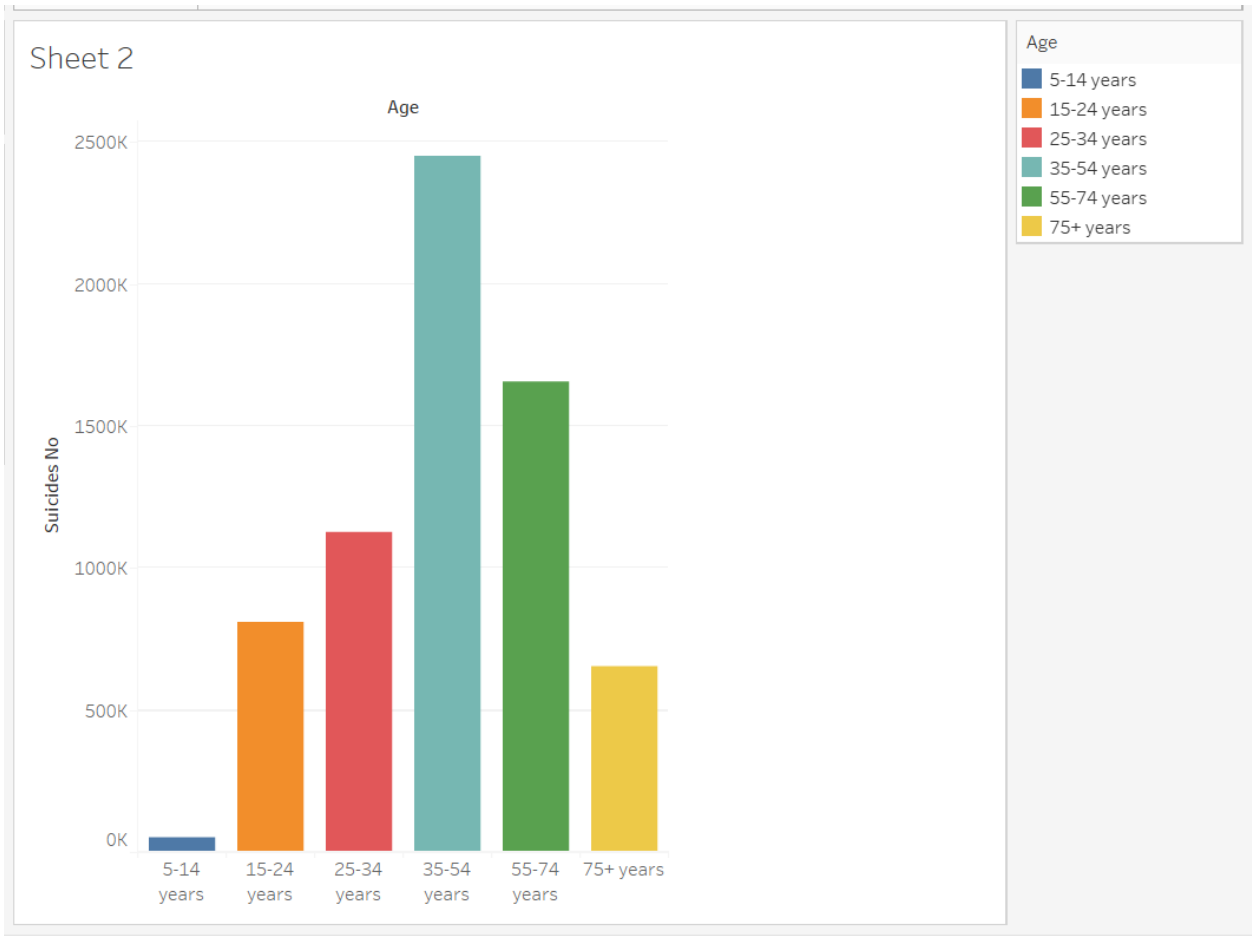


Fig 2.2 Bar-chart, amount of suicide taking place based on the age group.

This simple Bar-chart analysis shows the amount of suicide taking place based on the age group.

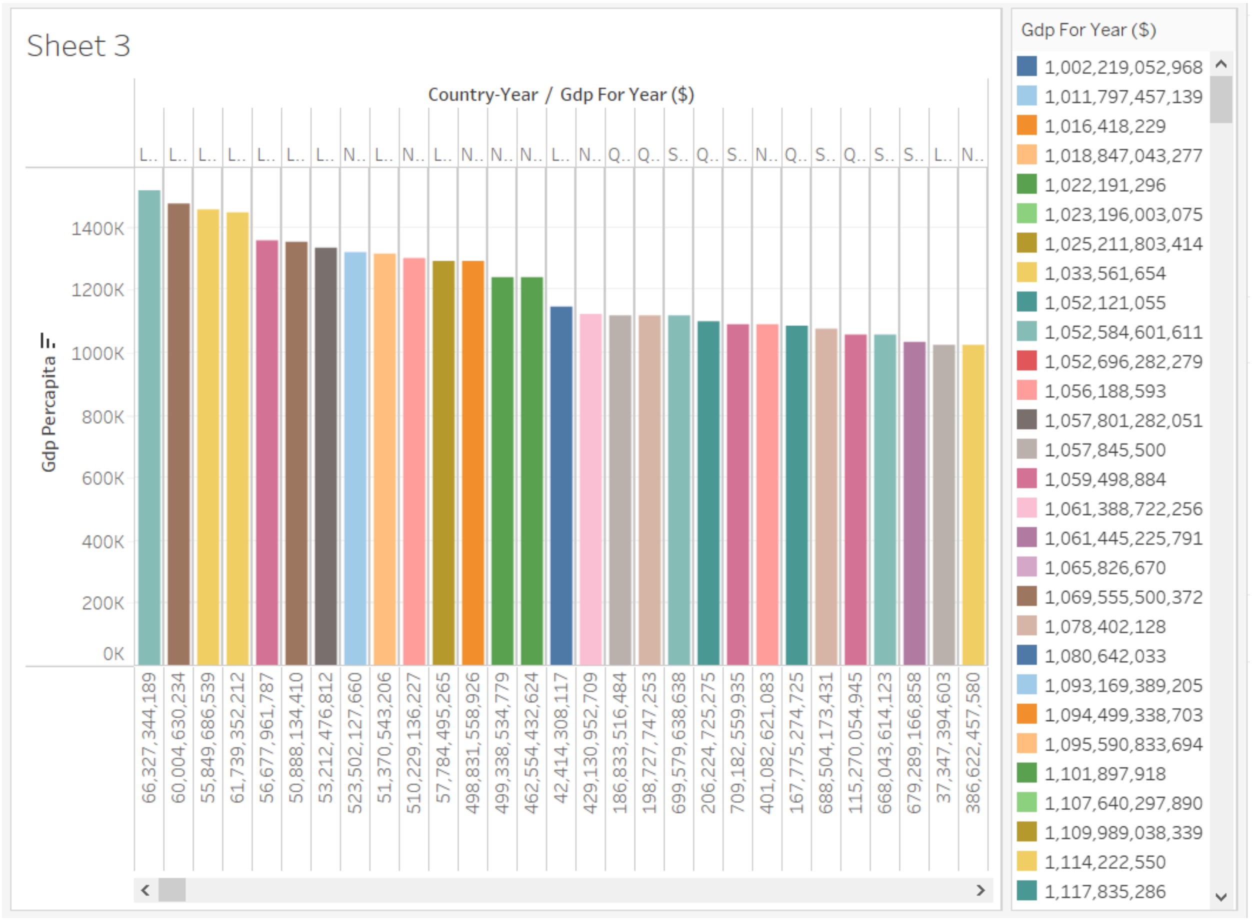


Fig 2.3 Bar-chart, GDP per capita and GDP per year of a country in a particular year.

This simple Bar-chart analysis shows the GDP per capita and GDP per year of a country in a particular year.

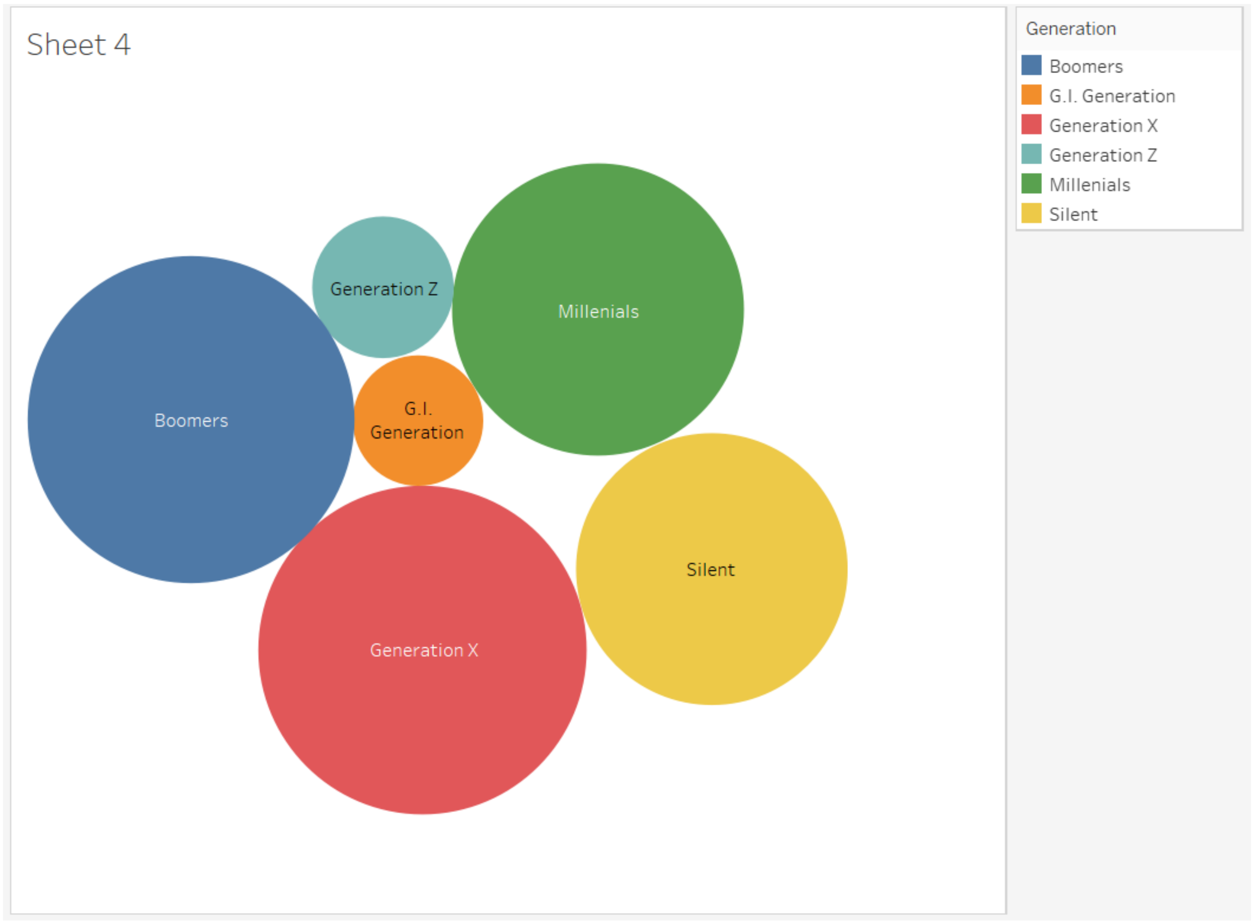


Fig 2.4 Amount of suicide taking place based on the generation.

This simple analysis shows the amount of suicide taking place based on the generation.

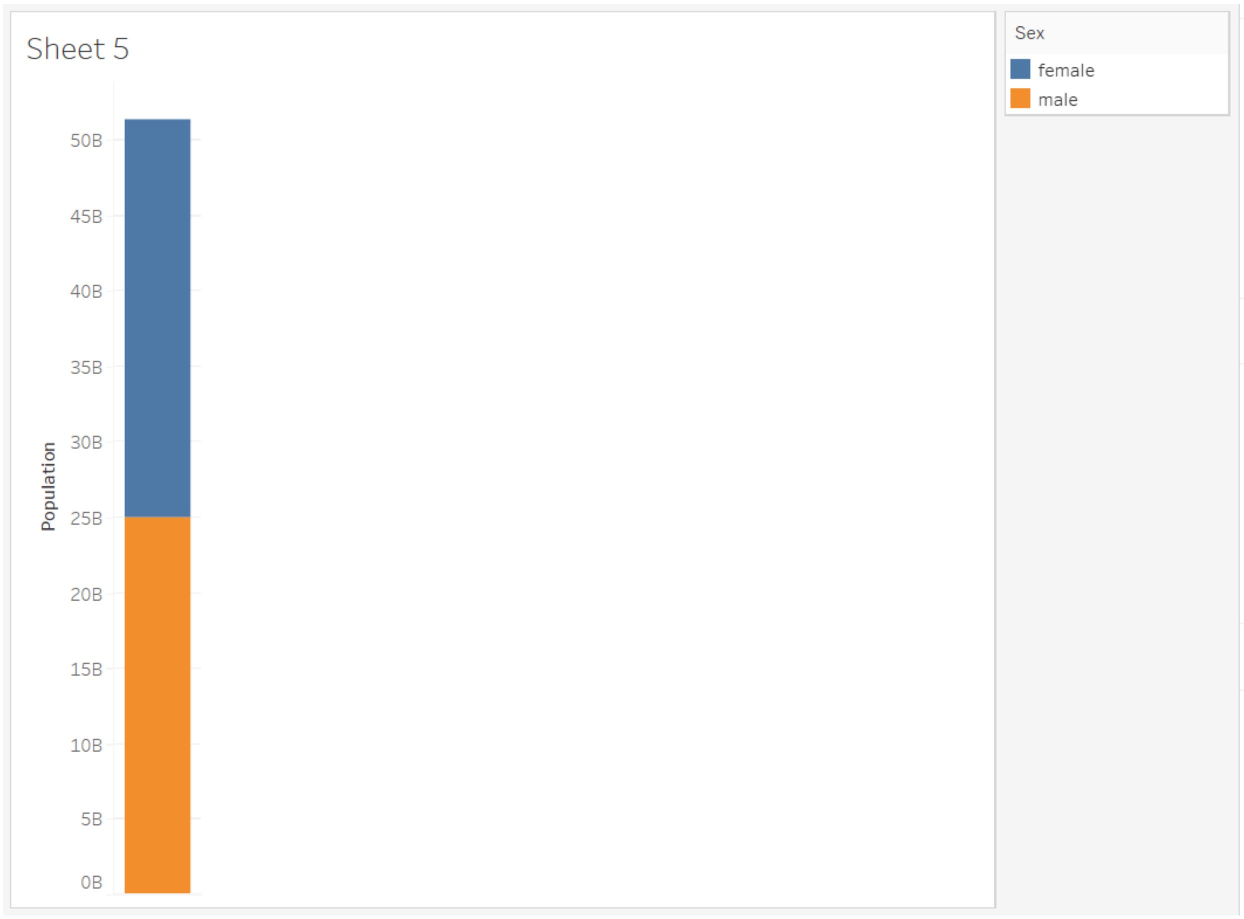


Fig 2.5 Amount of suicide taking place based on the gender.

This simple analysis shows the amount of suicide taking place based on the gender.

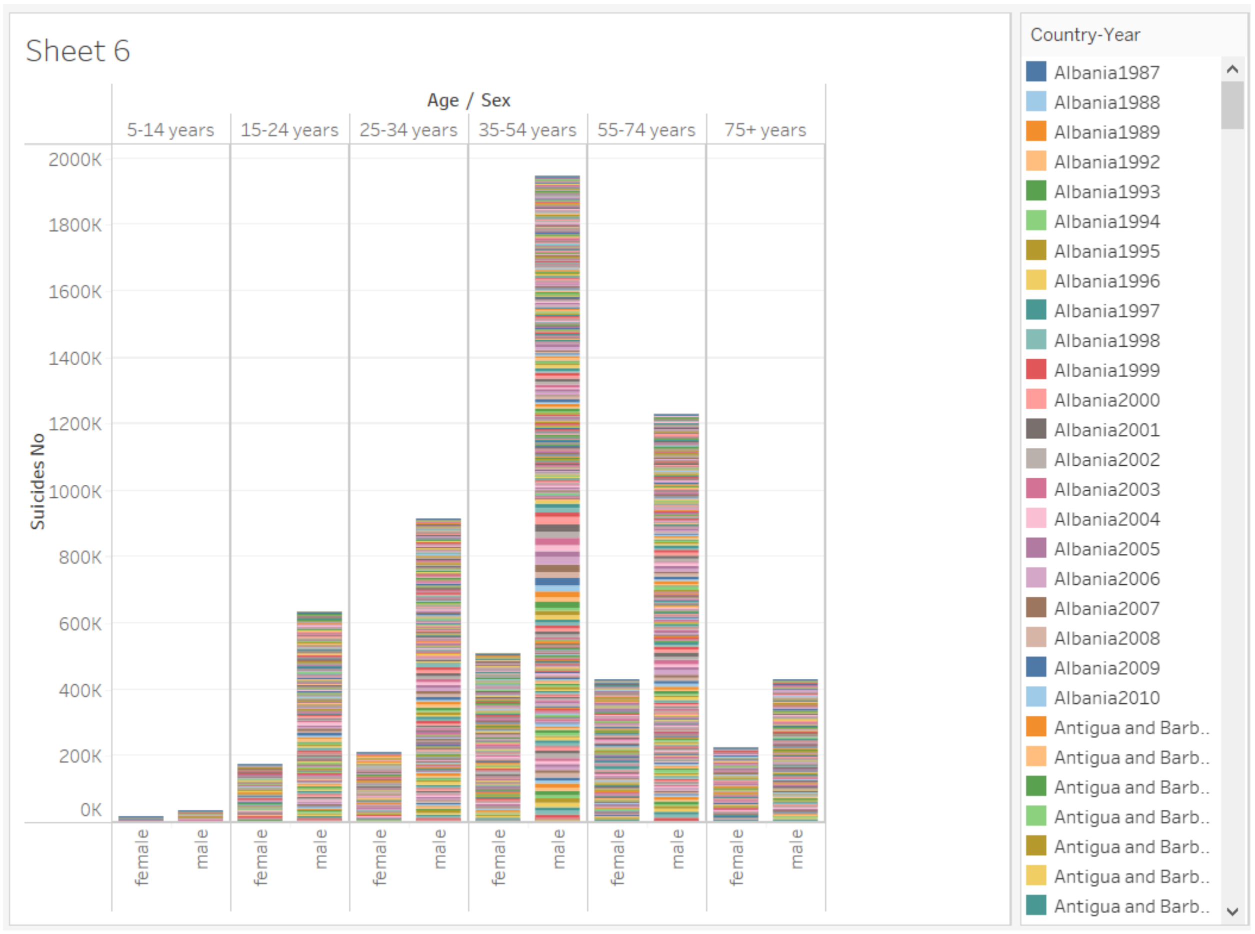


Fig 2.6 Amount of suicide taking place based on the gender, age and country.

This simple analysis shows the amount of suicide taking place based on the gender, age, and country.

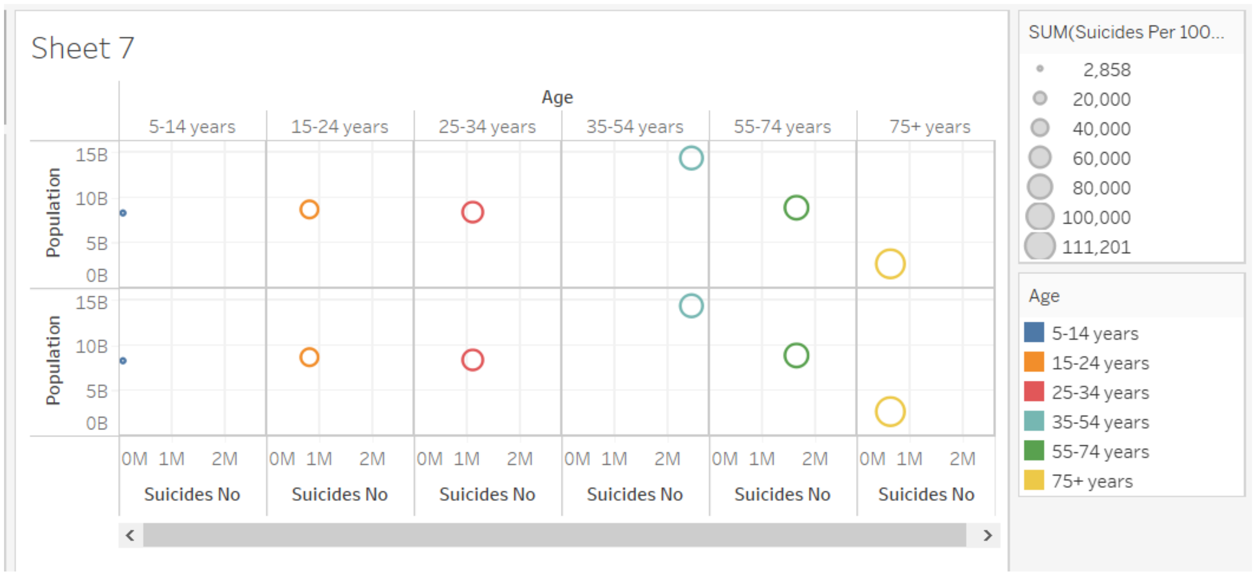
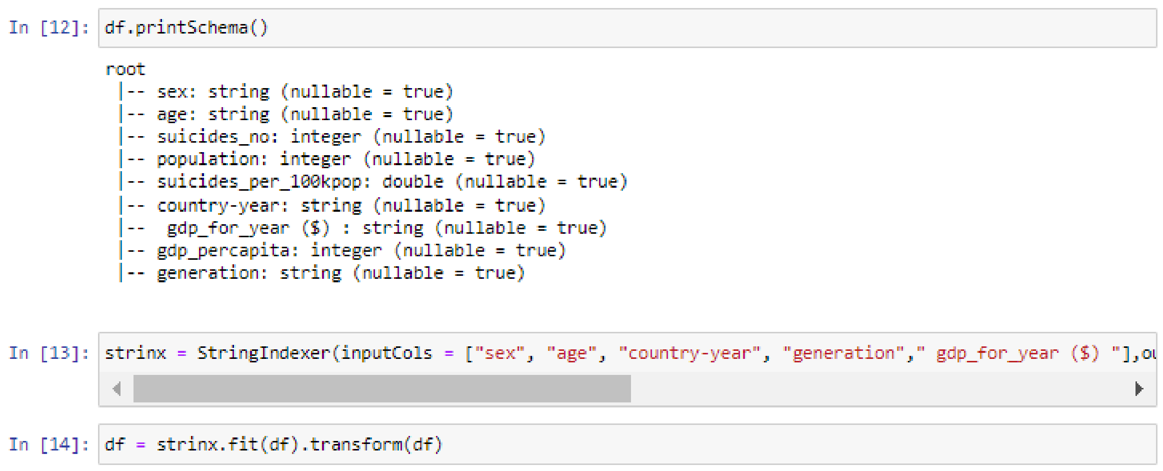


Fig 2.7 Amount of suicide per 100k population based on age.

This simple analysis shows the amount of suicide taking place per 100k population based on age.

Experimental Section

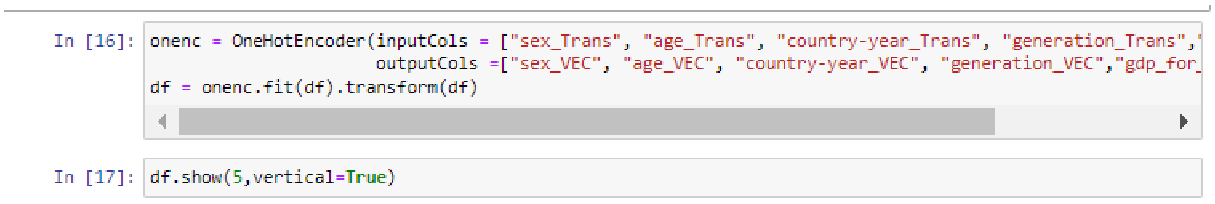
To begin with, machine learning algorithms are based on mathematical concepts, and in the actual world, we have all forms of data such as strings, integers, doubles, and Booleans. It may occupy a large amount of machine memory and make machine learning less efficient. To address this issue, we use feature engineering to transform long strings to integers and a greater number of characteristics into a single vector known as the feature except label column. I utilised “StringIndexer” and “OneHotEncoding” in Pyspark.



```
In [12]: df.printSchema()

         root
          |-- sex: string (nullable = true)
          |-- age: string (nullable = true)
          |-- suicides_no: integer (nullable = true)
          |-- population: integer (nullable = true)
          |-- suicides_per_100kpop: double (nullable = true)
          |-- country-year: string (nullable = true)
          |--  gdp_for_year ($) : string (nullable = true)
          |-- gdp_percapita: integer (nullable = true)
          |-- generation: string (nullable = true)

In [13]: strinx = StringIndexer(inputCols = ["sex", "age", "country-year", "generation"," gdp_for_year ($) "],ou

In [14]: df = strinx.fit(df).transform(df)
```

Fig 3.1 StringIndexer

As the name implies, StringIndexer is typically used for categorising string data into indexes. It features a built-in mechanism that converts strings into indices and also casts the type of the string. Indexing begins at 0 and proceeds in descending order. Here we can witness string indexer's work on sex, age, and many more.



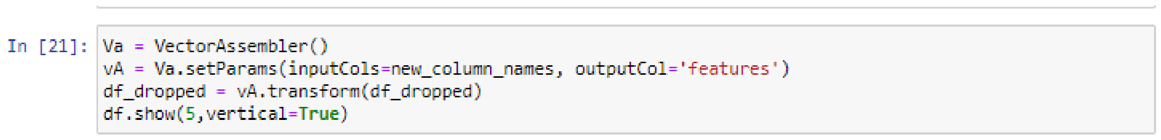
```
In [16]: onenc = OneHotEncoder(inputCols = ["sex_Trans", "age_Trans", "country-year_Trans", "generation_Trans",'
                               outputCols =["sex_VEC", "age_VEC", "country-year_VEC", "generation_VEC","gdp_for_
         df = onenc.fit(df).transform(df)

In [17]: df.show(5,vertical=True)
```

Fig 3.2 OneHotEncoder

A OneHotEncoder takes an index as input and outputs a binary vector. Because the last column is set to false by default, it may erase the last vector value.

Vector Assembler:

As its name implies, VectorAssembler brings together all vectors into a single feature. In-depth, turn a string into indices first, then an index to a single hot encode. In the end, we will get a vector of categorical values, but now the algorithm needs all possible numerical values as well. In conclusion, we employ vector assembler.



```
In [21]: Va = VectorAssembler()
         vA = Va.setParams(inputCols=new_column_names, outputCol='features')
         df_dropped = vA.transform(df_dropped)
         df.show(5,vertical=True)
```

Fig 3.3 VectorAssembler

Here, we create a column of numerical values with a variable and pass it to the function. The outcome will be vectors.

➢ Machine Learning:

Humans can learn through their surroundings, mistakes, and from other people. A machine can be programmed by humans to learn from its surroundings and data-driven behaviour. In essence, machine learning is the use of a machine that learns like a human and is used to perform complex calculations and resolve problems in the real world to improve human lives. AI includes machine learning as a subset.

There are three types of machine learning,

- **Supervised Learning**:
  Labeled data is used in supervised learning to train models. In plain English, data with input and output will provide accuracy with test data based on the amount of training component. How precise our data really is.

- **Unsupervised Learning**:
  It is completely at odds with guided learning. We only have inputs for this data, and the programmer doesn't even know what the output will be. As a result, machines themselves produce some output.

- **Reinforcement Learning**:
  Environment and behavioral factors are important in reinforcement learning. Here, a machine uses trial and error to obtain a reward. Chess game one is the most well-known example.

For this dataset I have used Linear Regression, and Factorize Machine Regressor algorithm.

➢ Linear Regression:

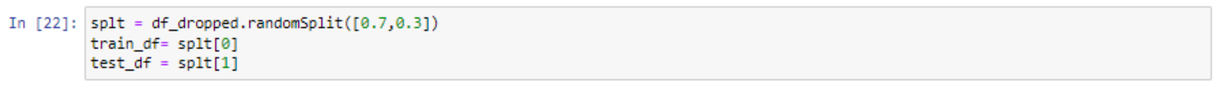
Using a method from linear algebra, linear regression resolves issues with dependent values that have an impact on the outcome. Expression in linear algebra: y=mx+c, where m is the slope, x and c are the dependent variables.

➢ Factorization machines regressor:

Factorization Machines are able to estimate interactions between features even in problems with huge sparsity (like advertising and recommendation system).

➢ Machine Learning Algorithm Implementation:

Before applying machine learning algorithm, we have to split the data into test set and train set. Here, I have split the data 70% for training and 30% for testing.

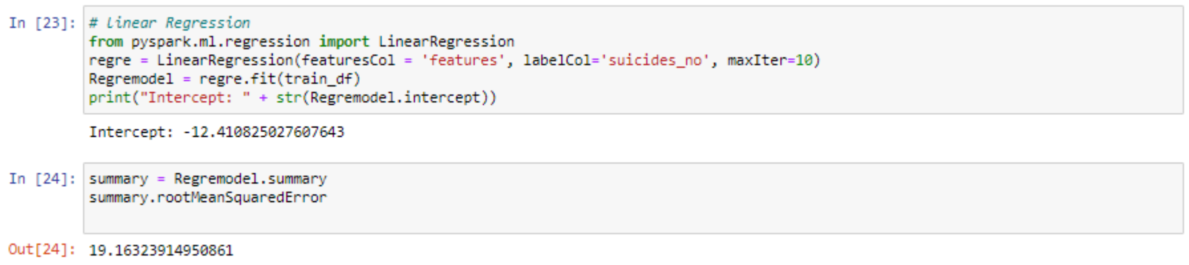


```
In [22]: splt = df_dropped.randomSplit([0.7,0.3])
         train_df= splt[0]
         test_df = splt[1]
```

Fig 3.1 Splitting the data

After that, I applied machine learning algorithm.

Linear Regression:



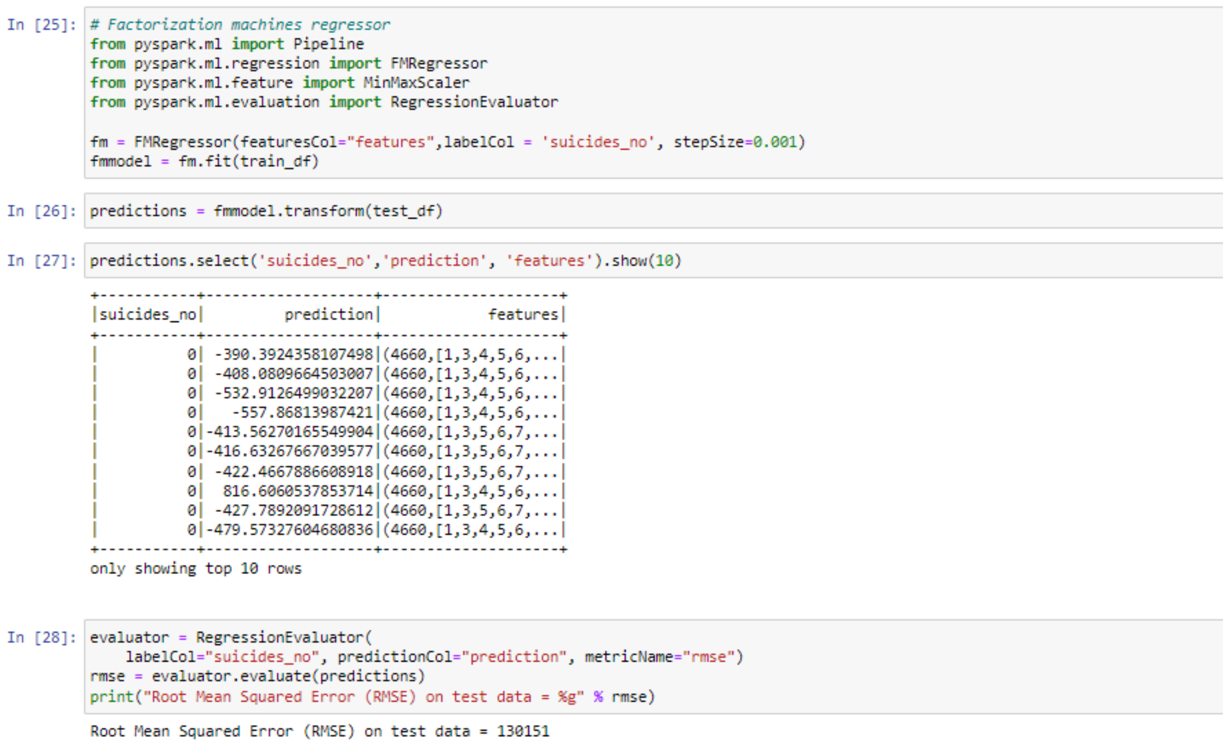
```
In [23]: # Linear Regression
         from pyspark.ml.regression import LinearRegression
         regre = LinearRegression(featuresCol = 'features', labelCol='suicides_no', maxIter=10)
         Regremodel = regre.fit(train_df)
         print("Intercept: " + str(Regremodel.intercept))

         Intercept: -12.410825027607643

In [24]: summary = Regremodel.summary
         summary.rootMeanSquaredError

Out[24]: 19.16323914950861
```

Fig 3.1 Applying Linear Regression

I have applied for Linear regression and got Root Mean Square error of 19.16 which means that, on average, the model's predictions deviate by 19.16 units from the actual values in the test dataset. In other words, the model's predictions are fairly close to the actual values.

Factorization machines regressor:

Here, I have imported the required module and applied the algorithm.



```
In [25]: # Factorization machines regressor
         from pyspark.ml import Pipeline
         from pyspark.ml.regression import FMRegressor
         from pyspark.ml.feature import MinMaxScaler
         from pyspark.ml.evaluation import RegressionEvaluator

         fm = FMRegressor(featuresCol="features",labelCol = 'suicides_no', stepSize=0.001)
         fmmodel = fm.fit(train_df)

In [26]: predictions = fmmodel.transform(test_df)

In [27]: predictions.select('suicides_no','prediction', 'features').show(10)

         +-----------+------------------+--------------------+
         |suicides_no|        prediction|            features|
         +-----------+------------------+--------------------+
         |          0| -390.3924358107498|(4660,[1,3,4,5,6,...|
         |          0| -408.0809664503007|(4660,[1,3,4,5,6,...|
         |          0| -532.9126499032207|(4660,[1,3,4,5,6,...|
         |          0|   -557.86813987421|(4660,[1,3,4,5,6,...|
         |          0|-413.56270165549904|(4660,[1,3,5,6,7,...|
         |          0|-416.63267667039577|(4660,[1,3,5,6,7,...|
         |          0| -422.4667886608918|(4660,[1,3,5,6,7,...|
         |          0|  816.6060537853714|(4660,[1,3,4,5,6,...|
         |          0| -427.7892091728612|(4660,[1,3,5,6,7,...|
         |          0|-479.57327604680836|(4660,[1,3,4,5,6,...|
         +-----------+------------------+--------------------+
         only showing top 10 rows

In [28]: evaluator = RegressionEvaluator(
             labelCol="suicides_no", predictionCol="prediction", metricName="rmse")
         rmse = evaluator.evaluate(predictions)
         print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

         Root Mean Squared Error (RMSE) on test data = 130151
```

Fig 3.3 Applying Factorize Machine Regressor

Applying Factorization machines regressor I got RMSE of 130151.

Discussion of Findings

In this study, we were unable to use decision tree and random forest regression models as the dataset was not categorical. We evaluated the performance of various models using the Root Mean Squared Error (RMSE) metric on the testing set. The Linear Regression model performed better than the Factorization Machines Regressor models, achieving an RMSE of 19.16 on the testing set. Our findings suggest that regression models can be effective in predicting suicide rates between 1985 and 2016. However, accurately predicting suicide rates is challenging due to the complexity and heterogeneity of the data and the lack of categorical data. Therefore, future research should explore more advanced modeling techniques, such as neural networks and ensemble methods, to improve the accuracy of suicide rate prediction. Overall, this study emphasizes the potential of regression models in predicting suicide rates and highlights the need for continued research in this field to better understand and prevent suicide.

Conclusion

Based on our analysis, we developed a regression model to predict the suicide rate between 1985 and 2016. We split the dataset into training and testing sets and used the training set to fit the regression models, including Linear Regression and Factorization machines regressor models. As the dataset is not categorical we can not apply Tree models such Decision Tree Regressor, Random Forest Regressor. We evaluated the models' performance on the testing set using metrics such as Root Mean Squared Error (RMSE).

Our findings indicate that the Linear Regression model outperformed Factorization machines regressor models, achieving an RMSE of 19.16 on the testing set.

The results of our analysis suggest that regression models can be effective in predicting the suicide rate between 1985 and 2016. However, there are still challenges in accurately predicting suicide rate due to the heterogeneity and complexity of the data and lack of categorical data, and more advanced modeling techniques, such as neural networks and ensemble methods, could be explored in future research.

Reference:

Cheng, A. T., Chen, T. H., Chen, C. C., & Jenkins, R. (2019). Psychosocial and psychiatric risk factors for suicide: Case-control psychological autopsy study. The British Journal of Psychiatry, 214(5), 285-291. doi: 10.1192/bjp.2018.272

Cao, X. L., Wang, S. B., Zhong, B. L., Zhang, L., Ungvari, G. S., Ng, C. H., & Xiang, Y. T. (2021). Network analysis of depressive symptoms, suicidal ideation, and suicide attempt in Chinese college students. Journal of Affective Disorders, 284, 142-149. doi: 10.1016/j.jad.2021.02.038

Kim, M. S., Kim, J. H., Kim, S. S., Kim, S. H., & Ahn, Y. M. (2018). Network analysis of suicide-related terms in the Korea Internet Addiction Proneness Scale (KIAPS): Comparison between complete and shortened versions. Psychiatry Investigation, 15(2), 152-158. doi: 10.30773/pi.2017.06.06

Qin, P., Mortensen, P. B., & Pedersen, C. B. (2020). Economic strain and suicide risk in a longitudinal study of 1.4 million people.