

Technical Project Report 1

Title: Health Nutrition & Population Statistics
Author: Balaji Shanmugasundaram
Student ID: L00171109
Supervisor: Shagufta Henna
Degree: MSc Data Science

Summary:

This report seeks to understand the population in each country in relation to the birth rate by analysing the Health Nutrition & Population Statistics dataset. The output population of this dataset consists of 74843 records. This dataset includes various points, including the name of the nation, the country code, the name and the code of the indicator, as well as the years 1960 to 2015. To assure the accuracy of the data, data cleaning was performed. To find the important features that may be used to forecast the output, feature extraction was carried out. The data was fed to two machine learning models, and each model was assessed using a separate assessment score. The project's findings revealed that both models had an accuracy score of 0. The benefit of applying data analysis and machine learning methods to the Health Nutrition & Population Statistics dataset has been illustrated by this research.

Introduction:

The Health Nutrition & Population Statistics dataset can be utilized to generate a Decision Tree model & Support Vector model to forecast the population growth in each nation based on the birth rate. Using the birth rate as a basis, these two models can produce a precise estimate of any nation's population. This dataset offers important health, nutrition, and population statistics that have been compiled from a variety of global and country sources.

Data Explanation:

The Health Nutrition and Population Statistics dataset compiles information from 1960 to 2015 for the entire globe. It contains details regarding the country name, country code, indicator name, indicator code and the years from 1960 to 2015. This dataset gives information on each nation's population as well as its birth rate.

Methodology:

Cleaning:

The act of modifying or removing useless, inaccurate, duplication, damaged, or unfinished data from a dataset is known as data cleaning. Cleaning is seen as a fundamental component of the foundation and plays a significant role in the analytical method as well as in creating trustworthy responses. The goal of data cleaning solutions is to provide consistent and standard datasets that give business intelligence and data analytical tools better access to and the ability to see the exact data for every issue.

The below figure defines that the dataset must be imported into a pandas data frame in order to start the cleaning process with the help of `pd.read_csv()`.

```
import pandas as pd
import io

from google.colab import files
uploaded = files.upload()

##Importing the dataset##


data = pd.read_csv(io.BytesIO(uploaded['data.csv']), header = 0)
```

Fig 1: Importing the dataset and cleaning.

Following dataset import, the following step displays the total number of rows and columns in the below figure.

(% gross)				
74840	Somalia	SOM	Sex ratio at birth (male births per female bir...	SP.POP.BRTH.MF
74841	Somalia	SOM	Share of women employed in the nonagricultural...	NaN

74842 rows x 60 columns



##We have 74842 rows and 60 columns##

Fig 2: Display the total number of rows and columns.

One of the most important tools for cleaning up data is drop(), dropping the missing elements and unwanted data which given in the below figure.

```
##Dropping the rows where all elements are missing##
data.dropna(how='all')
```

	Country Name	Country Code	Indicator Name	Indicator Code
0	Arab World	ARB	% of females ages 15-49 having comprehensive C...	SH.HIV.KNOW.FE.ZS
			% of males ages	

Fig 3: Dropping the missing elements.

The following stage is to import the file containing country populations from 1995 to 2020, as seen in the figure below.

```
from google.colab import files
uploaded = files.upload()

##Importing files where countries population from 1995 to 2020##

data2 = pd.read_csv(io.BytesIO(uploaded['Countries Population from 1995 to 2020.csv']), header = 0)
```

Fig 4: Importing the file containing country population from 1995 to 2020.

Following a few cleaning steps, the data's total number of rows and columns looks like the figure below.

4191	1970	Holy See	644	-5.49
4192	1965	Holy See	854	-1.18
4193	1960	Holy See	906	-0.04
4194	1955	Holy See	908	0.00

4195 rows × 14 columns




Fig 5: The total number of rows and columns after the cleaning stage.

Using `sqlite3.connect()`, as shown in the accompanying figure, you can create a new database and connect the local database.

```
##Opening a database connection and building a new database to operate with sqlite3##
import sqlite3

##Creating an access to the database local.db by using the sqlite3.connect() method##
conn = sqlite3.connect('local.db')

data.to_sql("countries", conn, if_exists="replace", index=False)

pd.read_sql_query('select * from countries', conn)
```

Fig 6: Creating new database using sqlite3.

The process of extracting a SQL query or database table into a data frame is shown in the figure below.

```
##The act of reading a SQL query or database table into a DataFrame##

data = pd.read_sql_query('select * from countries where "Indicator Name" = "Birth rate, crude (per 1,000 people)')
```

Fig 7: Showing the query into a data frame.

By using `data.columns`, the column labels of the provided data frame are displayed in the figure below.

```
##Returning the column labels of the given Dataframe##
data.columns

Index(['Country Name', 'Country Code', 'Indicator Name', 'Indicator Code',
      '1960', '1961', '1962', '1963', '1964', '1965', '1966', '1967', '1968',
      '1969', '1970', '1971', '1972', '1973', '1974', '1975', '1976', '1977',
      '1978', '1979', '1980', '1981', '1982', '1983', '1984', '1985', '1986',
      '1987', '1988', '1989', '1990', '1991', '1992', '1993', '1994', '1995',
      '1996', '1997', '1998', '1999', '2000', '2001', '2002', '2003', '2004',
      '2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013',
      '2014', '2015'],
      dtype='object')
```

Fig 8: Column labels of the provided data frame.

The data from a data frame is then added to a SQL database and then extracting the SQL database table into data frame, as seen in the below figure.

```
##Adding data from a DataFrame to a SQL database##

data2.to_sql("population", conn, if_exists="replace", index=False)

pd.read_sql_query('select * from population', conn)
```

Fig 9: Adding the data from data frame to SQL database.

The columns for the year 2000 are shown in the figure below, which is group by country.

```
##Delivering a DataFrame that corresponds to the query's return set##

data2 = pd.read_sql_query('select * from population where "Year" = 2000 group by "Country" ', conn)

##Displaying columns##

data2.columns

Index(['Year', 'Country', 'Population', 'Yearly % Change', 'Yearly Change',
      'Migrants (net)', 'Median Age', 'Fertility Rate', 'Density (P/km²)',
      'Urban Pop %', 'Urban Population', 'Country's Share of World Pop %',
      'World Population', 'Country Global Rank'],
      dtype='object')
```

Fig 10: Displaying columns for the year 2000 and group by country.

The aggregation of related records is then combined in the following figure.

```
##Merging the process of combining similar records##

new_data=data[['Country Name', 'Indicator Name', '2000']].merge(data2[['Country', 'Population']], left_on=['Country Name', 'Indicator Name'], right_on=['Country'])
```

Fig 11: Merging the similar records.

By importing matplotlib.pyplot, the functions of the figure, the plotting area, the plot lines, and the addition of plot labels are collected in the figure below.

```
##Collecting the functions in visualization elements of figure format includes figure, plotting area,
#plotting lines, and adding plot labels##

%matplotlib inline
import matplotlib.pyplot as plt
```

Fig 12: Showing in the figure format by importing matplotlib.pyplot

In the below figure, displaying the information with the country name and the year 2000 using a plot bar.

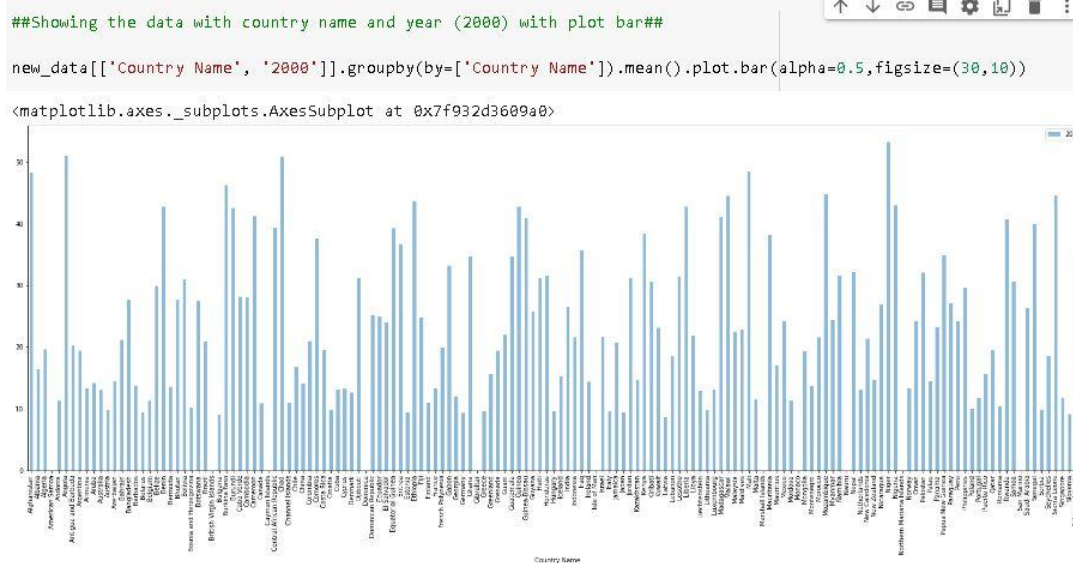


Fig 13: Showing the data with plot bars.

In the below figure, displaying the information with the country name and the year 2000 using a plot bar in different figure size.

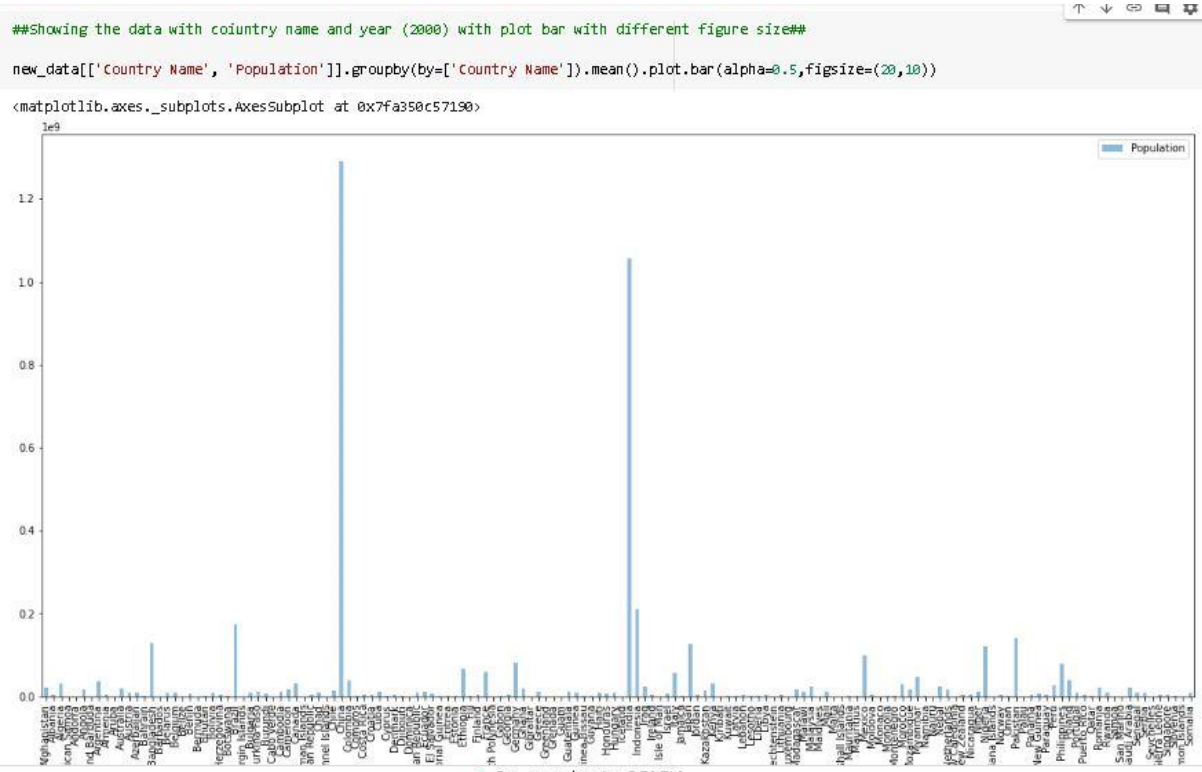


Fig 14: Displaying the data using plot bar in different size.

After a number of steps, the data is finally shown in the below figure along with the country name, indicator name, year, country, and population with different rows and columns.

```
##Here is the data with country name, indicator name, year, country and population##
new_data
```

	Country Name	Indicator Name	2000	Country	Population
0	Afghanistan	Birth rate, crude (per 1,000 people)	48.332	Afghanistan	20779953
1	Albania	Birth rate, crude (per 1,000 people)	16.401	Albania	3129243
2	Algeria	Birth rate, crude (per 1,000 people)	19.570	Algeria	31042235
3	American Samoa	Birth rate, crude (per 1,000 people)	NaN	American Samoa	57821
4	Andorra	Birth rate, crude (per 1,000 people)	11.300	Andorra	65390
...
147	Sierra Leone	Birth rate, crude (per 1,000 people)	44.634	Sierra Leone	4584571
148	Singapore	Birth rate, crude (per 1,000 people)	11.800	Singapore	4028871
149	Slovenia	Birth rate, crude (per 1,000 people)	9.100	Slovenia	1987717
150	Solomon Islands	Birth rate, crude (per 1,000 people)	35.615	Solomon Islands	412660
151	Somalia	Birth rate, crude (per 1,000 people)	48.668	Somalia	8872254

152 rows × 5 columns

Fig 15: Showing new data with different rows and columns.

The below figure shows after removing the unwanted column 'country' with 152 rows and 4 columns.

```
##Displaying the new columns after eliminating unwanted columns##
```

```
new_data.columns
```

```
Index(['Country Name', 'Indicator Name', '2000', 'Country', 'Population'], dtype='object')
```

```
new_data = new_data[['Country Name', 'Indicator Name', '2000', 'Population']]
```

```
new_data
```

	Country Name	Indicator Name	2000	Population
0	Afghanistan	Birth rate, crude (per 1,000 people)	48.332	20779953
1	Albania	Birth rate, crude (per 1,000 people)	16.401	3129243
2	Algeria	Birth rate, crude (per 1,000 people)	19.570	31042235
3	American Samoa	Birth rate, crude (per 1,000 people)	NaN	57821
4	Andorra	Birth rate, crude (per 1,000 people)	11.300	65390
...
147	Sierra Leone	Birth rate, crude (per 1,000 people)	44.634	4584571
148	Singapore	Birth rate, crude (per 1,000 people)	11.800	4028871
149	Slovenia	Birth rate, crude (per 1,000 people)	9.100	1987717
150	Solomon Islands	Birth rate, crude (per 1,000 people)	35.615	412660
151	Somalia	Birth rate, crude (per 1,000 people)	48.668	8872254

152 rows × 4 columns

Fig 16: New data with 152 rows and 4 columns after rejecting the 'country' column.

Importing the Decision Tree classifier to do the model in upcoming steps as shown in the below figure.

```
##Importing DecisionTree classifier##
```

```
from sklearn.tree import DecisionTreeClassifier
```

Fig 17: Importing Decision Tree classifier.

In the next step, imported the LabelEncoder to transform non-numerical labels to numerical labels.

```
##Using Labelencoder to transform non-numerical labels to numerical labels##
```

```
from sklearn.preprocessing import LabelEncoder
```

Fig 18: Importing LabelEncoder to transform non-numerical to numerical labels.

By applying .iloc, displaying all the columns till last where the integer position based from 0 to 1 with Boolean array as shown in the below figures.

```
##Displaying all the columns till last##  
new_data = new_data.iloc[:,:].values
```

Fig 19: Applying .iloc to display all columns

```
new_data  
array([[ 'Afghanistan', 'Birth rate, crude (per 1,000 people)', 48.332,  
        20779953],  
       [ 'Albania', 'Birth rate, crude (per 1,000 people)', 16.401,  
        3129243],  
       [ 'Algeria', 'Birth rate, crude (per 1,000 people)', 19.57,  
        31042235],  
       [ 'American Samoa', 'Birth rate, crude (per 1,000 people)', nan,  
        57821],  
       [ 'Andorra', 'Birth rate, crude (per 1,000 people)', 11.3, 65390],  
       [ 'Angola', 'Birth rate, crude (per 1,000 people)', 51.009,  
        16395473],  
       [ 'Antigua and Barbuda', 'Birth rate, crude (per 1,000 people)',  
        20.24, 76016],  
       [ 'Argentina', 'Birth rate, crude (per 1,000 people)', 19.413,  
        36870787],  
       [ 'Armenia', 'Birth rate, crude (per 1,000 people)', 13.203,  
        3069591],  
       [ 'Aruba', 'Birth rate, crude (per 1,000 people)', 14.187, 90853],  
       [ 'Australia', 'Birth rate, crude (per 1,000 people)', 13.0,  
        18991431],  
       [ 'Austria', 'Birth rate, crude (per 1,000 people)', 9.8, 8069276],  
       [ 'Azerbaijan', 'Birth rate, crude (per 1,000 people)', 14.5,  
        8122741],  
       [ 'Bahrain', 'Birth rate, crude (per 1,000 people)', 21.165,  
        664611],
```

Fig 20: All columns from 0 to 1 in Boolean array

In the next step, I faced challenge that got the data without columns index which as shown below.

```
##Displaying data without columns##
```

```
new_data = pd.DataFrame(new_data)
```

```
new_data
```

	0	1	2	3
0	0	Birth rate, crude (per 1,000 people)	48.332	20779953
1	1	Birth rate, crude (per 1,000 people)	16.401	3129243
2	2	Birth rate, crude (per 1,000 people)	19.57	31042235
3	3	Birth rate, crude (per 1,000 people)	NaN	57821
4	4	Birth rate, crude (per 1,000 people)	11.3	65390
...
147	147	Birth rate, crude (per 1,000 people)	44.634	4584571
148	148	Birth rate, crude (per 1,000 people)	11.8	4028871
149	149	Birth rate, crude (per 1,000 people)	9.1	1987717
150	150	Birth rate, crude (per 1,000 people)	35.615	412660
151	151	Birth rate, crude (per 1,000 people)	48.668	8872254

152 rows × 4 columns

Fig 21: Faced challenge which data showed without columns index.

In the below figure, adding columns index to the final data with 152 rows and 4 columns.

```
##Adding columns to the final data##
```

```
new_data.columns = ['Country Name', 'Indicator Name', '2000', 'Population']
```

```
new_data
```

	Country Name	Indicator Name	2000	Population
0	0	Birth rate, crude (per 1,000 people)	48.332	20779953
1	1	Birth rate, crude (per 1,000 people)	16.401	3129243
2	2	Birth rate, crude (per 1,000 people)	19.57	31042235
3	3	Birth rate, crude (per 1,000 people)	NaN	57821
4	4	Birth rate, crude (per 1,000 people)	11.3	65390
...
147	147	Birth rate, crude (per 1,000 people)	44.634	4584571
148	148	Birth rate, crude (per 1,000 people)	11.8	4028871
149	149	Birth rate, crude (per 1,000 people)	9.1	1987717
150	150	Birth rate, crude (per 1,000 people)	35.615	412660
151	151	Birth rate, crude (per 1,000 people)	48.668	8872254

152 rows × 4 columns

Fig 22: Adding columns to the final data.

After adding columns, dropping the rows with a null by checking all the columns in every rows in the below figure.

```
##Dropping the rows with a null by checking all the columns in each row##
new_data = new_data.dropna(axis=0)
```

new_data

	Country Name	Indicator Name	2000	Population
0	0	Birth rate, crude (per 1,000 people)	48.332	20779953
1	1	Birth rate, crude (per 1,000 people)	16.401	3129243
2	2	Birth rate, crude (per 1,000 people)	19.57	31042235
4	4	Birth rate, crude (per 1,000 people)	11.3	65390
5	5	Birth rate, crude (per 1,000 people)	51.009	16395473
...
147	147	Birth rate, crude (per 1,000 people)	44.634	4584571
148	148	Birth rate, crude (per 1,000 people)	11.8	4028871
149	149	Birth rate, crude (per 1,000 people)	9.1	1987717
150	150	Birth rate, crude (per 1,000 people)	35.615	412660
151	151	Birth rate, crude (per 1,000 people)	48.668	8872254

141 rows × 4 columns

Fig 23: Dropping the rows with a null value.

In the below figure, finally showing the data with country name, year and population.

```
##Showing the final data with country name, year and population##
new_data = new_data[['Country Name', '2000', 'Population']]
```

Fig 24: Final data with country name, year and population.

Importing train_test_split to split the data arrays into two subsets in the below figure.

```
##Splitting the data arrays into two subsets by importing train_test_split##
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Fig 25: Splitting data arrays into two subsets by importing train_test_split.

In the below figure, predicting the data with the another model which is Support Vector classifier.

```
##Importing Support Vector Classifier##  
from sklearn.svm import SVC
```

Fig 26: Importing Support Vector classifier.

The below figure is to find the accuracy score.

```
X = new_data.drop(columns=['Population'])  
Y = new_data['Population'].astype('int')  
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2)  
  
our_model= DecisionTreeClassifier()  
our_model.fit(X_train,Y_train)  
predictions = our_model.predict(X_test)  
  
score = accuracy_score(Y_test, predictions)  
score  
0.0
```

Fig 27: Accuracy score.

Decision Tree Model:

A supervised machine learning algorithm called Decision Tree applies a set of principles to make judgments, much like how people do. In decision trees, we begin at the tree's root when anticipating a record's class label. We contrast the base attribute's values with that of the attribute on the register. It is used to predict the outcomes of a dataset by learning decision rules generated from the features of the data. By using a set of criteria, the decision tree algorithm can be utilized to categorize the target variable.

Here showing the predictions after using Decision Tree classifier model as show in the below figure.

```
##Showing the predictions after using DecisionTree Classifier model##  
  
X = new_data.drop(columns=['Population'])  
Y = new_data['Population'].astype('int')  
  
our_model= DecisionTreeClassifier()  
our_model.fit(X,Y)  
predictions = our_model.predict([[27,9],[46,12.6]])  
predictions  
  
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but Decision  
warnings.warn(  
array([[7997957, 5341194]])
```

Fig 28: Showing the predictions after using Decision Tree classifier.

Support Vector Classifier Model:

It is an algorithm for supervised machine learning that is frequently applied to classification jobs. In order to split the data into two classes, the best classifier must be found after mapping the datasets to a high dimensional region. The greatest margin classifier's efficiency issue is solved to produce the support vector classifier, albeit with added constraints.

Next, showing the predictions after using Support Vector classifier model as show in the below figure.

```
##Showing the predictions after using Support Vector Classifier model##

X = new_data.drop(columns=['Population'])
Y = new_data['Population'].astype('int')
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2)

second_model = SVC(kernel='rbf', random_state=1)
second_model.fit(X,Y)

new_predictions = second_model.predict([[27,9],[46,12.6]])
new_predictions

/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but SVC
  warnings.warn(
array([7997957, 5341194])
```

Fig 29: Showing the predictions after using Support Vector classifier.

Finally, the below figure shows that the predictions are same in both the models as what I showed above with 141 rows and 3 columns.

new_data			
	Country Name	2000	Population
0	0	48.332	20779953
1	1	16.401	3129243
2	2	19.57	31042235
4	4	11.3	65390
5	5	51.009	16395473
...
147	147	44.634	4584571
148	148	11.8	4028871
149	149	9.1	1987717
150	150	35.615	412660
151	151	48.668	8872254

141 rows × 3 columns

Fig 30: As per the figure, predictions are same in both the models.

Conclusion:

Python is a great resource for comprehending complicated data sets and getting insightful knowledge about visualization and analysis of data. It is a fantastic technique to comprehend the material more fully. Python's model has the advantage of being a rapid and efficient data analysis tool. Based on the above steps, and portrayal in images as shown above that has been carried out effectively and obtained as expected.

Github Link:

https://github.com/adenlad555/Technical_Project

References:

1. <https://datacatalog.worldbank.org/>
2. www.analyticsvidhya.com/blog/2021/06/
3. Carolina Bento/<https://towardsdatascience.com/decision-tree-classifier>
4. Nagesh Singh Chauhan/<https://www.kdnuggets.com/2020/>

