|  |
| --- |
| Global Data on Happiness |
|  |

|  |  |
| --- | --- |
| Sept 16, 2019 | ETL - PROJECT |

**George Bigham & Dawn Brenner**

Global Data on Happiness

ETL - PROJECT

## INTRODUCTION

The purpose of this project was to build a project from the ground up using a typical flow of data in the workplace: 1) pull real data from various sources (extract), 2) clean and restructure it as needed (transform), and 3) write it into a database for storage (load). Extract, transform, and load - ETL.

## E - EXTRACT

We used the following data sources to gather interesting information about countries and happiness:

* World Happiness Report, Sustainable Development Solutions Network:   
  This dataset includes happiness rankings and scores by country for 2015-2017. It also includes factors contributing to the happiness score. The data came in three separate CSV files, one per year.
* Suicide Rates Overview 1985 to 2016, compiled dataset by “Rusty” from various sources:   
  This dataset includes the number of suicides per year per country by gender and age group. We were planning to include the Human Development Index (HDI) as another dataset, but found HDI already included in this dataset. (While working with the data we realized all of the HDI values were null so we ultimately dropped it.) This dataset did include thorough population data which we were originally planning to extract from a World Bank dataset. This data came in a single CSV file. We thought this data interesting to include to see if happiness and suicide are as correlated as one would expect.

All CSV data was downloaded then converted to Pandas dataframes using pd.read\_csv.

*Lessons learned:*

* *There are a lot of interesting datasets available.*
* *Many datasets are a compilation of other datasets created by unknown persons. User beware.*
* *Prior to finalizing dataset selection, open datasets to see if data actually provided matches the description of data provided.*

## T - TRANSFORM

Python was used to clean and transform our data to achieve the data we wanted to upload to our database. The following was performed:

* happiness: This dataset required uploading multiple CSV files and appending them to the dataframe. Unfortunately, the CSV column names changed with each year as well as the reporting method of the happiness score. This made cleanup more time consuming than expected.
* suicide: This dataset required aggregation of the data to determine a total number of suicides per country rather than separated by age and gender. Many of the small countries have very low numbers of suicide (0 to 5) so it was impossible to know which countries actually had no suicides versus which countries were unreported. As stated in the previous section, the HDI columns were null in additional to many other “promised” values from the dataset’s description.
* population: This dataset was created by creating a new table from the suicide table with the country\_id, year, and population.
* countries: We created this dataframe to store the country\_id, country, and region. The table was created using unique country names from the happiness table. A country\_id was automatically generated. Each country’s region was added to the table using a function that looked up countries in a dictionary and returned the associated region.
* All datasets: Many of the countries of the world were written out differently in the different datasets. Because we were joining all of the tables by country, it was important all countries were represented consistently in the datasets. Originally, we had planned to include a country\_lookup dataframe and function in the python code to look up all variations of a country’s name in each dataset and return the country code. The user would be prompted if a country’s variant name was not found. The user would enter the country\_id and the dataframe would be appended with the new variation. Unfortunately, we ran out of time to create this functionality. Instead, we used the country’s name given in the happiness dataframe. For each row of the dataframes, we grabbed the given country name, looked for it in the ‘countries’ dataframe, and returned the country\_id to that country’s row of the data table. Any unmatched countries were dropped from the dataframe.

*Lessons learned:*

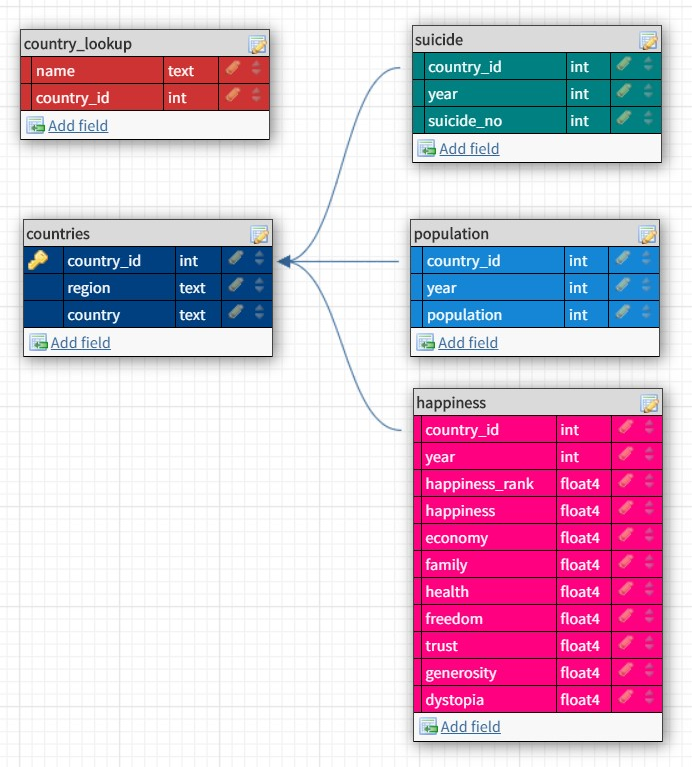
* *Inspect ALL datasets thoroughly prior to transforming any datasets. Problems found later with one dataset changed how we would have proceeded.*
* *There are more issues to consider when you know you have to get data into a relational database.*

## L - LOAD

Load: the final database, tables/collections, and why this was chosen.

A relational database made sense for our data because we had one-to-many relationships for all tables with multiple years. Also, the database structure would enforce the use of consistent country names. Finally, the relational database helped make sense of the multiple tables we were using. Our datasets were not overly large, so processing time did not have to be considered.

We created the Entity Relationship Diagram (ERD) shown in the figure below and created the SQL code (schema.sql) to create the tables and relationships in our etlproject\_db. The end of our python script included a postgres connection to the etlproject\_db database and loaded each table to the database using the .to\_sql function.



*Lessons learned:*

* *Add a ‘DELETE table’ row before all tables in SQL query while debugging and comment them out when the code works. Ultimately, this will save a lot of time.*
* *Be careful with datatypes, including decimal places.*