**Extract:**

We were able to obtain 3 csv files from Kaggle.com. In addition were able to pull 5 csv files and 1 excel file from the website Data.World. The excel file was resaved as a csv file to keep the sources in a consistent format and for easy import.

1. “Total economic damage from natural disasters (US$)”

<https://www.kaggle.com/dataenergy/natural-disaster-data>

1. “Global Temperatures”

<https://www.kaggle.com/schedutron/global-temperatures>

1. “Number of reported natural disasters (reported disasters)”

<https://www.kaggle.com/dataenergy/natural-disaster-data>

1. “People killed in natural disasters”

Dataset in Humanitarian Data Exchange

<https://data.world/hdx/73fcf87e-c8d7-4310-a3ed-8d201ae12246>

1. “Number of people made homeless by natural disasters”

Dataset in Humanitarian Data Exchange

<https://data.world/hdx/d2ec211d-faf6-4fb5-a46c-2094dc5830af>

1. “Number of people injured in natural disasters”

Dataset in Humanitarian Data Exchange

<https://data.world/hdx/0255cf49-2c13-430d-b738-aebdc9733bdd>

1. “Total cost of damage done by natural disasters”

Dataset in Humanitarian Data Exchange

<https://data.world/hdx/f83e5a9a-67d0-4861-aff4-09fc38eb78da>

1. “Total number of people affected by natural disasters”

Dataset in Humanitarian Data Exchange

<https://data.world/hdx/97e007af-4733-4b60-a472-a733f10dedd5>

1. “Fossil-Fuel CO2 Emissions”

Dataset By Adam Helsinger:

<https://data.world/adamhelsinger/fossil-fuel-co-2-emissions>

Original Source:

<http://cdiac.ornl.gov/ftp/ndp030/global.1751_2013.ems>

**Transform:**

In order to have a consistent format for all of our sources we had to transform the data sets, ultimately having the “Year” as the index for all datasets.

The “Total economic damage” and “Number of reported natural disasters” csv files were close, but needed to have some rows removed (before 1900 and after 2014 – these were the years we had data on across all sources), also they had information arranged sub-optimally; each year was split per disaster type. To stay with the year as the index, these had to be re-arranged to suit.

The “Global Temperatures” csv was split by month, so the most reasonable approach seemed to be to average the data by year. We also removed columns that were not relevant to our project.

The five datasets from the Humanitarian Data Exchange were formatted the same and therefore we chose to merge these into one DataFrame. When trying to pull the csv datasets into our jupyter notebook, we quickly realized that they had a different delimiter than the more common comma, in order to load the csv files we had to account for the semi-colon delimiter. The years were originally columns, we had to create a list from the column names (since there were 115 individual columns). Utilizing the column list, we were able to use the Pandas Functions including : Melt, Groupby, Sum, Column rename that transforms the 115 columns into two columns consisting of Years and the Total Values.

The last DataSet from Data.World needed very little transforming, as it was already in the desired format. In order to condense the information we dropped the columns keeping only the Year and Total columns, we then Filtered the “Year” column to keep the years in the range of 1900-2014 to match the other DataSets.

**Load:**

After extracting and transforming the Data into four desired DataFrames, we uploaded these three tables into a single PostregSQL DataBase. We chose a relational database because we had an obvious primary key to use across tables, a clear way to define the tables, and after cleanup, a consistent data set.

One issue we encountered was that the int data type could not hold numbers large enough for the economic damage costs, so we had to use bigint instead.

**Flask:**

We created a Flask application (04\_DISASTER\_FLASK.py), that once the database is loaded can be used to retrieve summary or detailed information for one year or for all years. The number of columns of data presented a challenge in that it made the SQL query unwieldy, but it does work.

We also found it more efficient and precise to create an SQL query and pass it to sqlalchemy with .execute than to use sqlalchemy’s internal querying methods.

The output from the flask application prioritizes ease of parsing over other concerns.

With this flask application, future users can query the data to find correlations between the various contributing factors and the disaster events from the years 1900-2013.