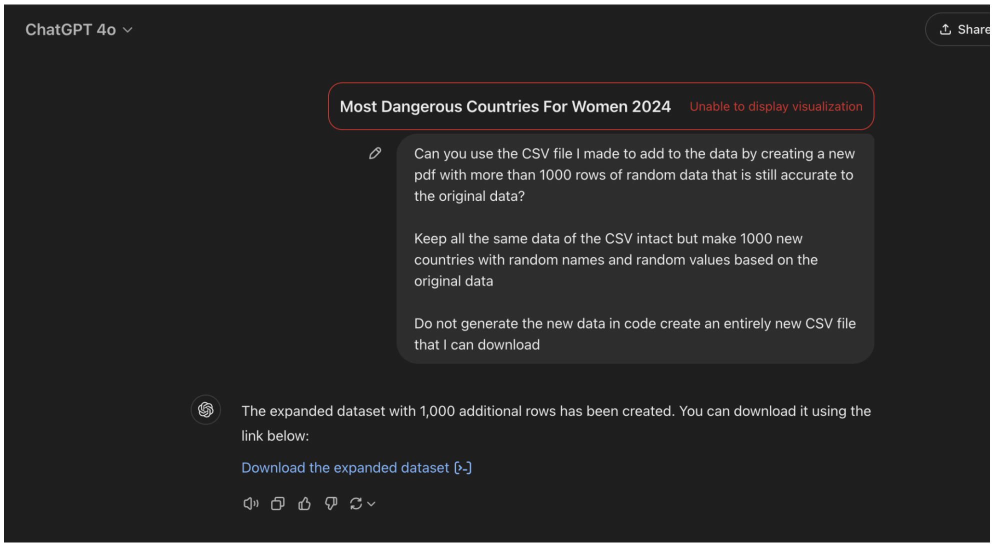
Eduardo Torres

DS210

12/8/24

The dataset I chose for this project is *"Most Dangerous Countries for Women 2024"*. This data contains 176 rows containing most of the countries in the world and 11 columns. The first column is the countries name and the following columns were Women, Peace and Security Index score for 2023, Women's Danger Index total score for 2019, Women's Danger Index score for street safety in 2019, Women's Danger Index score for intentional homicide in 2019, Women's Danger Index score for non-partner violence in 2019, Women's Danger Index score for intimate partner violence in 2019, Women's Danger Index score for legal discrimination in 2019, Women's Danger Index score for global gender gap in 2019, Women's Danger Index score for gender inequality in 2019, Women's Danger Index score for attitudes toward violence in 2019. With this dataset, I wanted to know what countries for women were the most dangerous, moderately dangerous, and lastly safest. Due to my dataset being too small but still interested in this topic one of the TAs Zachary Gentile suggested I use *ChatGpt* to make it larger by adding more non-existent countries and basing it on the original dataset. This way I could keep my code and satisfy the requirements of this project.


ChatGPT 4o
Share
Most Dangerous Countries For Women 2024
Unable to display visualization

Can you use the CSV file I made to add to the data by creating a new pdf with more than 1000 rows of random data that is still accurate to the original data?

Keep all the same data of the CSV intact but make 1000 new countries with random names and random values based on the original data

Do not generate the new data in code create an entirely new CSV file that I can download

The expanded dataset with 1,000 additional rows has been created. You can download it using the link below:

Download the expanded dataset

With this larger dataset, I used graph-based clustering. I represented my graph with a similarity matrix, where each node is a country, and the edges that different countries formed together would represent the similarity between countries based on their distance in their scores. I would group these nodes (countries) based on their similarities (edge weights). Countries with higher similarity scores are placed together, these clusters are then outputted so I can analyze.

My first module is the DataFrame module, the purpose of this module is to read the data and turn the data into usable values for Rust to read and use. As a result of remaking the panda's data frame in Rust, I was able to salvage some code from homework 8 and adjust it for this project. I create an enum first called ColumnVal which takes in Country as a string and Score as a 64-bit float point integer. On my ColumnVal enum, I made an impl statement implementing Display for ColumnVal. This code formats CSV files into a table when printed, I got this code from homework 8. Next, I created a struct called DataFrame which takes in labels as a vector of strings, types as a vector of unsigned 32-bit integers, and rows which is a vector of vectors of ColumnVal values. I created an implementation to use the new data type I made, new() was the first function I made which returned Self. This function initializes all the 3 types inside DataFrame as new empty vectors. read_csv was the following function which took in a referenced mutable version of self, path which is a referenced string, and types which is a reference of a vector containing unsigned 32-bit sized integers. This function returns Result<(), Box<dyn Error>> which is used to catch errors just in case the file isn't found without crashing rust entirely. This function first reads the given file it finds from path, separates all values by commas, and assumes the first row is headers. I define first_row as true to iterate through all the labels(using .iter().map()) in the first-row dataset converting them to strings (using .to_string()). I make sure that the number of columns matches the length of types, prevent any errors later, and make sure every column gets printed. Lastly, in a for loop, I create a match statement to separate the first row of strings and other rows of integers. I push each value into vectors by matching their corresponding type and put it all together in the variable self.types. The final function in this module is get_scores which takes in a reference of self and returns a vector of Array1s containing 64-bit float point integers. Get_scores extracts and returns the numeric values from the rows field in DataFrame. The function iterates through each row in self.rows which contains the vector of ColumnVals. I use filter_map to process each ColumnVal, and apply a filter, and a transformation in a single step. For each row, if ColumnVal is a Score(val) the f64 value is extracted and wrapped in Some(val). However, if ColumnVal is not a Score, None is returned. This helps my code separate numbers from country names. Once all the filtering is done I am left with a vector of f64 values for the row. Following this, the filtered vector gets collected and converted into a one-dimensional array from the ndarry crate (Array1) of f64 integers. This is used to represent the list of scores as an ndarry object. Lastly, I finish the function with .collect() to collect all the results for each row and create one large vector of Array1s containing 64-bit float point integers. Each item in this new vector represents an entire row in self.rows, where the array holds the data's scores for that row.

The following module I created was Country_Score_Creator, where all of the computational functions for the clustering are created. The first function I created was euclidean_distance which calculates the euclidean_distance between two one-dimensional arrays containing 64-bit floating point integers and returns a 64-bit floating point integer. This function iterates through the first array given, and using .zip() also iterates through the second given array and with map() applies the Euclidean distance function on every item in both arrays. Lastly, the sum of all the values is taken and squared.

Euclidean Distance Formula →

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

The next function is create_similarity_matrix() which creates a matrix representing the pairwise Euclidean distances between vectors provided. This function takes in data, a referenced vector containing one-dimensional arrays containing 64-bit floating point integers. It returns a 2-dimensional array of 64-bit floating point integers. To initialize the matrix, the variable n represents the number of vectors in the input dataset, and the mutable variable similarity_matrix is the initialized empty matrix with n x n dimensions that will store all the pair distances between countries. Following the creation of these variables, I set up a nested for loop that iterates through all pairs of vectors and applies the Euclidean_distance function on them. Lastly, I fill in the matrix with the calculated values and return the Matrix. The last function in this module is called simple_clustering which clusters countries into three groups based on their similarity. This function takes in similarity_matrix which is a referenced two-dimensional array containing 64-bit floating point integers, scores which is a vector of 64-bit floating point integers, and it returns a vector of usize. I begin by creating the variable n which gets the dimensions of the given matrix and stores the number of data points in n. The mutable variable clusters is a vector of 0s n times. I created an avg_score variable that stores the average score of each country based on its similarity to other countries. I iterate through each row and calculate the mean using row.mean().unwrap_or(0.0) (if the current option observed has some value it is returned if not 0.0 is returned). In a for loop each country is placed in a cluster based on their score. If their score ranges from 0.0 - 0.6 they are in the first cluster (most dangerous), if the score ranges from 0.6 - 0.8 they are in the second cluster (moderately dangerous), and lastly 0.8 - 1.0 these countries are placed in the third cluster (safest)

In my main file, I import various libraries as well as both modules I created. In my main function, I define the file path with the name of the CSV, open the file, and parse through all of the country's scores. This all sets up the data for Rust to be able to read and use. I extract the scores from each country and create the similarity matrix with these scores. Then I use simple_clustering on the similarity matrix, once my algorithm is applied the countries are classified the countries based on their score. Then I print out the top 5 of each category.

The three tests in my code test that my euclidean_distance function is correct. I did this by running the function on two vectors and comparing the result to the actual answer. Next, I tested that the scores I calculated in my main function were correct by checking the most dangerous countries and the safest countries, I compared these to the first ranked that were printed in my main function. My final test checked that my create_similarity matrix function works correctly which I did by comparing an example with my function being used and the expected outcome.

Some issues I ran into while creating this project revolved around how I would cluster my data points into groups. I tried to use k-means at first but I ran into multiple stability issues. After researching more options I tried to use the k-means algorithm again but through another dependency called linfa which is a Rust version of the k-means clustering algorithm. However, the linfa-clustering algorithm also ran me into more clustering problems, and in the end, I wasn't able to output useful results with the data I had. This is what led me to create scores using similarity matrices and place similar countries into the same groups. This provided me with more accurate results which answered my original question the top 5 most dangerous countries for women in the world are Afghanistan, Yemen, Central African Republic, DR Congo, and South Sudan. The top 5 moderately safe countries for women in the world are - Romania, Seychelles, North Macedonia, Albania, and Mongolia. Lastly, the top 5 most dangerous countries in the world for women are Denmark, Switzerland, Sweden, Finland, and Luxembourg. Despite the data being biased towards women the top 5 most dangerous countries for women were generally dangerous due to having high crime rates, terrorism, lack of justice due to corrupt/weak government, and active military groups in war. These countries are generally unsafe for anyone, however in moderately safe these countries were more biased toward women. They had some of the issues above but crimes against women were higher. Lastly, the safest countries all shared the common trait of being economically well stabled with low rates of the issues mentioned above. Denmark specifically has a high social trust rate and high police activity level. Denmark along with the other countries has built a sense of safety and security among citizens.