Assignment 2: Data Integration & Preparation

Dimitrios Diamantidis - 7766408

Johan Hensman - 1118994

Nicky Schilperoort – 5755354

Task 1: Profiling relational data

We calculated the number of unique values in the accident_index column to see how many accidents occurred in 2022. We then checked the sum of the rows where the severity of the accidents was mild (where the value in the casualty_severity was 1). This can be a useful statistic depending on what kind of research you do.

It also can be useful to see at what age accidents happen, so we calculated the minimum, maximum and mean value of the age_of_casualty column. In addition, we calculated the standard deviation of the age_of_casualty column to see in what age range the most accidents occur. The median of the age_of_casualty column shows us an accurate representation of the center of the data, because it is not affected by for example outliers.

We then counted the number of null values in the age_of_casualty column. This will give an idea of how much values are missing and tells you how accurate your data actually is. The values '0', '1', '2' and '9' were present in the sex_of_casualty column, with '0' representing an unknown value. We took the mode of this column to find out what gender was present most in the column.

The correlation values of the dataframe were calculated to find what numeric columns are correlated with each other. This can be helpful in finding patterns in the data and see if there is reason why accidents happen more often under certain circumstances.

Task 2: Entity resolution

Part 1

a. We started by using pandas function to read the CSV files and store the data into dataframes in order to modify them further. Having these dataframes, we created new variables by copying the data excluding the ID columns using the .iloc[] function.
b. We used a nested 'for' loop to update every single record in every attribute and the .lower() function to change all the characters into lowercase.
c. Furthermore we used the .replace() function in every record to reduce the noise of multiple spaces. All multiple spaces were converted into one space.
d. We have used the 'py_stringmatching' library in order to calculate the similarity between the two strings, based on Levenshtein similarity theory. The function uses two strings as input and returns the distance (a float number) of them.
e. Similarly, we have used the same library to calculate the similarity between two strings, based on the Jaro similarity theory. We also created an new function 'mod_jaro_sim' to check the NaN cases, this function uses two strings as input and returns the similarity (a float number) of them.

f. Once more, we have developed a function 'aff_sim' that calculates the similarity between two strings, based on Gap similarity theory. This function takes two strings as an input and gives back the similarity of them, in a float number.
g. Another method of comparing two strings, is to check if the strings are exactly the same. This is achieved by using an if statement and the equal operator '=='. The result is a binary value, where 1 indicates that strings are equal, and 0 shows that they are different.
h. The rec_sim formula is defined to combine the scores of every previous method, while assigning different weight to each method. In this way, we produce a single similarity score for each comparison made using 4 distinct different similarity methods.
i. We store the ids of both records in a list. For every similarity score over 0.7, we save their IDs in a list 'similar_list'.
j. We need to compare our results with the number of correct duplicate records. To achieve that, we create a new dataframe variable 'dblp_acm_mapping' of 'DBLP-ACM_perfectMapping.csv' file to get access to the file. After that we need to use the len() function to find the number of the total rows of 'dblp_acm_mapping' and also the number of similarities of our way. Having this numbers we can easily compare the results and print the distance of these results.
k. The code's execution time currently stands at 55 minutes. Furthermore, we can reduce the running time in many ways, such as to remove the duplicates, to use Hashing methods, to create samples to find a representative proportion of similarities.

Part 2

1. After the required libraries were imported and assigned to variable, we first concatenated the values in all records into one single string.
2. All of the alphabetical characters were changed into lowercase.
3. In addition, we converted all the multiple spaces to one space. These steps made sure all data was in the same format.
4. The records from both tables were then turned into one big list.
5. In this step, we first computed the 3-shingles. With the shingle list we created a vocabulary and a one hot vector. After this, we computed the minhash signature with 100 different permutations. Then, the similarity was computed by using the LSH method and using 10 buckets.
6. The top 2224 candidates from the LSH algorithm were extracted and compared to the actual mappings. These actual mappings are in the DBLP-ACM_perfectMapping.csv file. We checked if our pairings are the same as the actual mappings using dictionaries as these are efficient for quick lookups. Following, we computed the precision by dividing the amount of correct pairs by the total amount of candidate pairs.
7. The running time was recorded and turned out to be 3.4 seconds.
8. We compared the running time of part 1 to the running time of part 2. The running time of part 1 was 55 minutes and the running time of part 2 was 3.4 seconds. The precision of part 1 was low (0.0005), but it can be adjusted to obtain a higher accuracy. The accuracy of part 2 stays around 0.50, but as it takes a significantly shorter time to get results this is the better method.

Task 3: Data preparation

1. First of all, the required libraries were imported. We stored the data in a variable and took a look at the table it produced. We then removed the 'Outcome' column and created a table which showed the correlation between all of the columns.
2. We replaced all the values that are 0 in the columns 'BloodPressure', 'SkinThickness' and 'BMI' with, in our case, NaN. This made sure the 0 values would not affect any future calculations.
3. Next, we filled all the cells that we just replaced with NaN, with the mean value of the column.
4. Then we computed the correlation again, just like we did in 3.1.
5. We then compared the correlation values we calculated in 3.1 with correlation values we calculated in 3.4. The full answer on the comparisons of the correlation tables are written in the Python notebook.