INFOMDWR – Assignment 2: Data Integration & Preparation

Ilse van Deventer (9996974)

Majdouline Hamdi (9767738)

Zexuan Li (8069182)

Zoé Ricardie (9107096)

Menno Zoetbrood (1084720)

Task 1: Profiling relational data

For this task, download and read the paper about profiling relational data, select a set of summary statistics about the data (minimum of 10 different values) and write Python code to compute these quantities for a dataset of your choice. Preferably, you can use one of the csv files from the road safety dataset. Explain the importance of each summary statistic that you selected in understanding the characteristics of the dataset.

Note: Computing the same statistical quantity on multiple columns of the dataset will be counted only once.

Importance of each summary statistic

Profiling data is a crucial step in evaluating the quality, structure, and content of a database. By generating useful summary statistics, we create metadata that helps with further analysis.

There are numerous methods for gaining a deeper understanding of the data in a database. In their article, Abedjan et al. (2015) give an extensive overview of methods for profiling data.

The most basic profiling data steps include checking the type of data and the number of values (p.560). Data type helps to understand the nature of the data (float, integer, string), which is an essential step to determine what type of analysis is possible on the data. It also helps to prevent computational errors and guide the process of data visualisation. The number of values is fundamental to understanding the scale and scope of the database. It also facilitates the execution of other data profiling statistics.

The authors also dive into null values (p. 561). Checking for null values is essential for a relevant analysis. It prevents calculation errors, reveals the quality of the data and the possible collection problems. Knowing their existence enables the data scientist to either exclude them from the analysis or perform data transformation techniques to reduce their impact on data analysis.

The number of distinct values (p.558) is a relevant statistic to compute as it reveals fundamental metadata information. Similar to the previous methods, it helps understand the distribution and the quality of the data. It also helps with tasks such as normalisation, detection of foreign keys and preventing redundancy. The number of unique values(p.563) is slightly different to the number of distinct values. While the former is a count of the number of values per column, the latter is a relative measure that determines how unique the column is compared to its size. The main goal of the uniqueness ratio is to determine the candidate key. For example, a ratio of 1 means that every value in the column is distinct, which often indicates that it is a candidate key. This statistical measure also helps to check the quality of the data and avoid redundancy.

While identifying distinct and unique values is important, it is also relevant to understand how influential the most frequently occurring data is. That’s constant (p. 564). It measures how the most common value dominates within a column. It helps detect columns that do not have significant insight and determine if and how useful that column is for further analysis.

Mean and median are two other data profiling methods (p. 561). Mean gives a sense of central tendency, which is useful for comparison. It is a good summary for normal data, but the result might be skewed by outliers. On the other hand, the median is robust to outliers and shows the true ‘middle’ of the data. Those two methods can also be looked at together, which provides a good understanding of the distribution shape of the data and its quality. Standard deviation (p. 561) also helps to measure variability and help detect anomalies, but while the mean and median measure central tendency, the standard deviation qualifies the degree of variability. Max-min values (p. 561) complement those methods to provide a full understanding of the data distribution. Min-max shows the boundary of the dataset, including possible errors or outliers. Combining those methods enables data scientists to see the centre, the limits and how spread out the data is.

The authors also expand on Benford’s law and first-digit distribution analysis (p.564). This method looks at the frequency of the leading digit in numerical data. The expectation is that number 1 is naturally the most leading digit in a dataset. If not, it helps to determine anomalies and check the quality of the data. It also helps in understanding data characteristics.

Finally, a histogram is one of the most useful tools in data profiling (p. 558). It creates a visualisation of the distribution described by the other methods (mean, median, etc.). It is important because it graphs and reveals the true shape of the data and exposes hidden structure while clearly exposing anomalies and outliers.

Task 2: Entity resolution

Part 1

When reading in the data into a Pandas DataFrame, the 'DBLP2' is encoded using 'Latin1', which is not Python's standard encoding type (UTF-8). Therefore, we have to specify the type of encoding the csv file is using.

a. Ignore the id column

We can ignore the id column by simply deleting it from our dataset. However, since we will need to answer a future question, we can simply create a new variable (_noID).

b. Change all alphabetical characters into lowercase

c. Convert multiple spaces to one

We combine steps b and c by defining a function (data_prep). The function loops over the columns of a DataFrame after it has verified that the column has the 'object' data type, meaning it contains string-based data.

The 's\+' expression finds all white space characters in each row, and replaces them with a single space.

d. Compute Levenshtein similarity on the title attribute

e. Compute Jaro similarity on the authors attribute

f. Compute affine similarity on the venue attribute (normalised to the scale [0,1])

g. Use Match (1) / Mismatch (1) for the year

To answer steps d through g, we first merge the ACM and DBLP2 DataFrames, creating a dataset with 600.000 columns.

We then define the function for step g (year_match). The reason why we first define step g is because we need to use this function in another function, which will compute the match / mismatch in addition to the three similarity scores (sim_measures). For Levenshtein and Jaro similarity, we compute the similarity score directly for each row. However, we need to compute the raw score for affine similarity, meaning it is not normalised yet. To normalise it, we transform the affine similarity column by min-max scaling the results to the interval [0, 1]. Lastly, for match / mismatch for the year, we can simply use our pre-defined function.

h. Computing the similarity score using the formula rec_sim

For steps h through j, we need to reintroduce the 'id' column of both datasets into our merged DataFrame. To do this we cross join the 'id' columns of the two DataFrames, containing every possible pair of 'id' values. After that, we combine the existing similarity score DataFrame with the merged ID pairs side-by-side (axis = 1). We use 'drop = True' to ensure both DataFrames have aligned indices.

In order to define what weight to give each attribute, we use Pandas' .nunique() function to see the number of unique entries. Results of this function show that the 'title' and 'authors' attributes take up a significant amount of the unique entries compared to the 'venue' and 'year' attributes. Therefore, we opted to go with 45 percent (0.45) for 'title' and 'authors' each, and 5 percent (0.05) for 'venue' and 'year' each, totalling 100 percent (1.0).

i. Reporting the records with rec_sim > 0.7

To find all the duplicate records with rec_sim > 0.7, we filter the records with rec_sim > 0.7, storing the ids of both records. We then store the ids of both records to a list.

j. Computing the precision of this method compared to the perfect mapping

We start by mapping the predicted 'id' column values. To do this, we need to convert the list we made in point i into a tuple to make them usable inside a set. We then do the same thing for the true mappings between the two datasets. Lastly, to compute the precision, we create a simple formula which divides the true positives over all predicted positives, the result being the precision between our results and the true mappings.

k. Record the running time of the method. What can you do to reduce the running time?

To record the running time of steps d through g, we use the time library in Python. We start the timer right before we run the sim_measures() function (start) and end it right after this function has finished (end). We then compute the time it took for the function to run (end - start). For our method, it took roughly 40:57 minutes for the function to run.

The function took quite a while to execute. Therefore, it is important to look for ways to reduce the running time. For example, we can pre-filter candidate pairs based on a shared attribute or token (blocking). Furthermore, we could also use a locality-sensitive hashing (LSH) method, creating similar
buckets based on shingle signatures (which is what we will do in part 2).

Part 2:

For this part, we will use the code about LSH from the tutorial.

1. *Concatenate the values in each record into one single string.*
2. *Change all alphabetical characters into lowercase.*
3. *Convert multiple spaces to one.*
4. *Combine the records from both tables into one big list as we did during the lab.*
5. *Use the functions in the tutorials from lab 5 to compute the shingles, the minhash signature and the similarity.*
6. *Extract the top 2224 candidates from the LSH algorithm, compare them to the actual mappings in the file DBLP-ACM_perfectMapping.csv and compute the precision of the method.*
7. *Record the running time of the method.*
8. *Compare the precision and the running time in Parts 1 and 2.*

For task 2 part 2, we used locality-sensitive hashing (LSH) to find similar or duplicate records between the DBLP and ACM datasets. The main goal was to make the matching process faster compared to the approach in part 1.

We started by cleaning the data: all text was converted to lowercase, extra spaces were removed, and the fields title, authors, venue and year were combined into one single string per record. This gave us one unified text representation per publication. Then, both datasets were merged into a single dataframe so that LSH could be applied to all records together.

Next we used shingling to represent each record as a set of small character sequences (shingles). Each shingle acts like a fragment of the text and helps detect similarity even when the exact wording differs slightly. We tested multiple values of k (the shingle size) and found that k = 2 gave a slightly higher precision because shorter shingles create more overlap between similar texts. However smaller k values can also cause more false positives.

After shingling we created minhash signatures for each record using 100 random hash permutations. These signatures compress the text but still preserve its similarity structure. The signatures were then split into bands, which are smaller groups of rows that are hashed together. We set bands to 10. If two signatures share a hash value within the same band, they are considered potential duplicates. This drastically reduces the number of comparisons since only similar signatures end up in the same buckets.

Once we obtained the candidate pairs from LSH, we computed their Jaccard similarity using the minhash signatures. Pairs with a similarity higher than 0.7 were kept as likely duplicates. Finally, we compared those pairs to the ground-truth file DBLP-ACM_perfectMapping.csv to calculate the precision and evaluate the quality of our matching process.

Discussion of the results

<u>From the new output:</u>

- **Number of candidate pairs**: 2620
- **Number of candidate pairs after selecting top 2224**: 2224
- **Number of similar pairs (Jaccard > 0.7)**: 1701
- **True positives**: 1210
- **Precision (with Jaccard)**: 0.711
- **Runtime (seconds)**: 17.62

This shows that out of all 4910 records, LSH identified 2620 potential candidate pairs. After selecting the top 2224 pairs and filtering them using a Jaccard threshold of 0.7, 1701 pairs were considered highly similar. Among those, 1210 pairs were true matches according to the perfect mapping file.

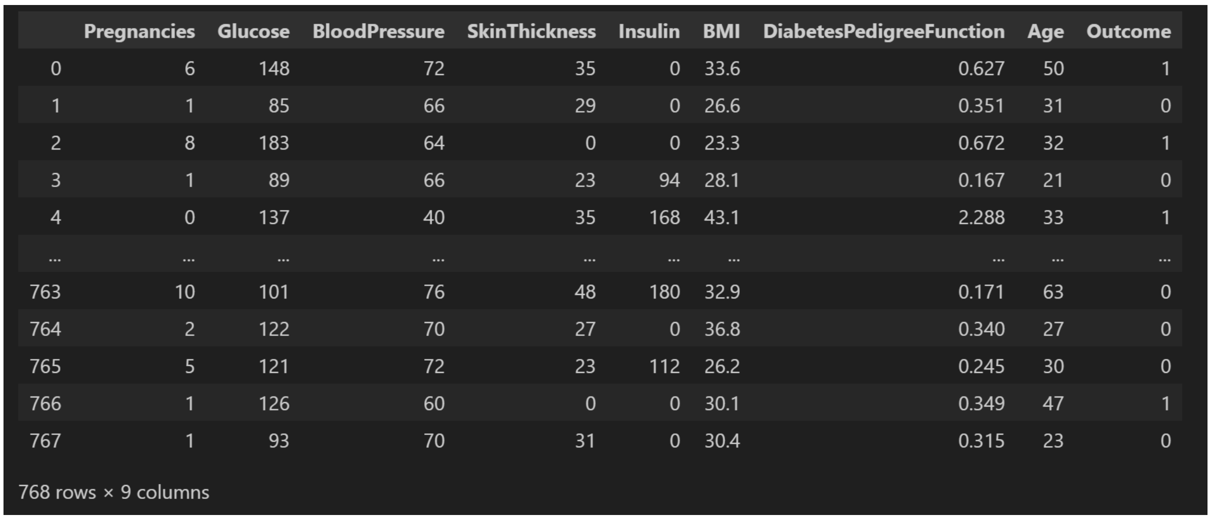
The resulting precision of 0.71 (71%) indicates that the LSH method performs quite well, but it is lower than the precision of part 1. The total runtime of 17.62 seconds of LSH is significantly faster than the record comparison from part 1. So this might be a beneficial trade-off, especially when you do more comparisons. The trade-off is that LSH can still miss a few matches (lower recall) or include some false positives depending on parameters like k, the number of bands, and the similarity threshold. When we increased the number of hashing functions and the number of bands the precision increased a bit, however the run time increased a lot so we decided to set the bands to 10 and 100 hashing functions.

To sum it up; the LSH-based entity resolution has a faster running time, however it was less accurate. Depending on the type and length of the documents you are comparing, as well as the purpose of the comparison and the required level of precision, one can determine which method is most appropriate for the specific task.

Task 3: Data preparation

For this task, use the Pima Indians Diabetes Database.

In this task, we are looking into Pima Indians Diabetes Data and observing correlation between different variables and how it changes after removing categorical variables.



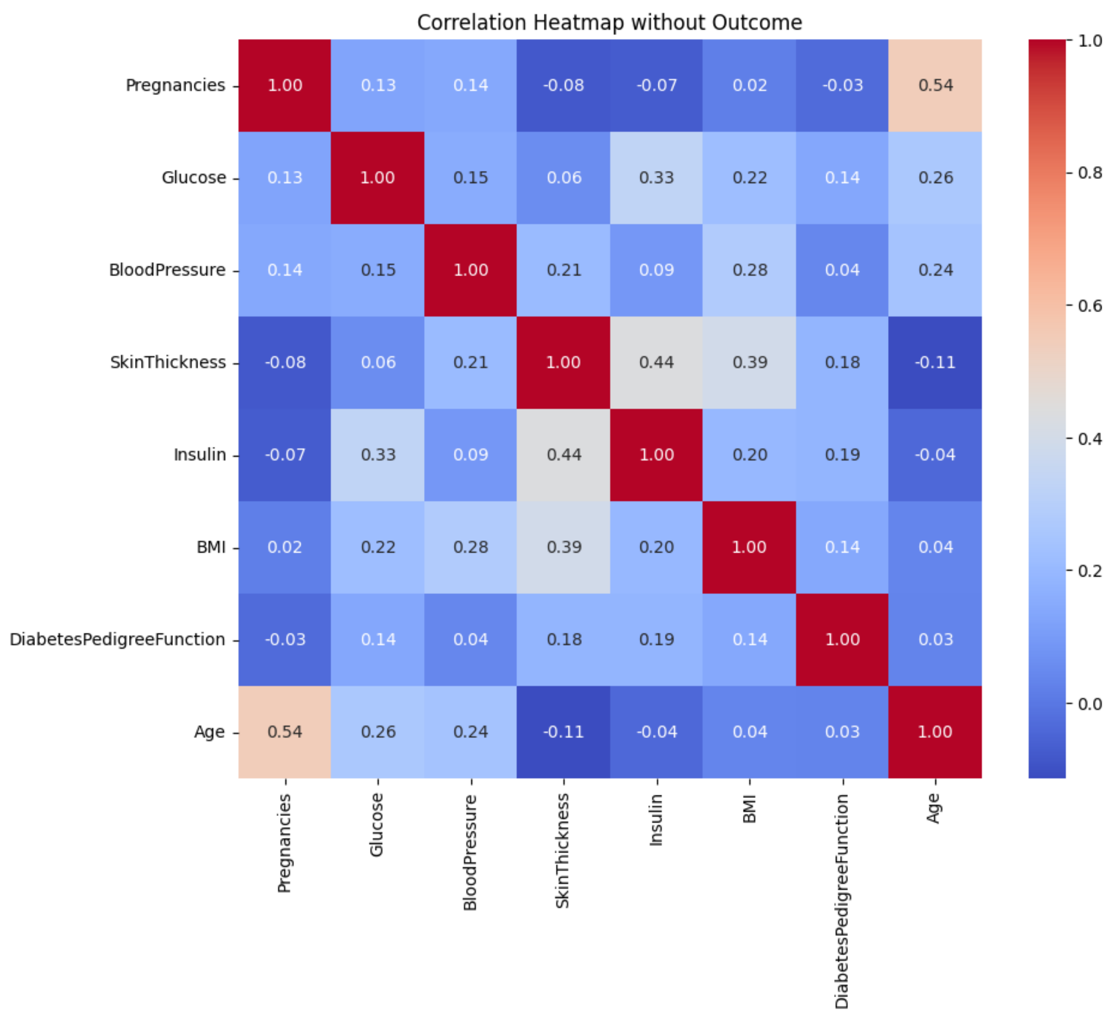
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Figure 1: Diabetes Dataset

1. *Compute the correlation between the different columns after removing the outcome column.*

After importing the data, we explore the data with .info() and .nunique() to check datatype and how many distinct values are there for each variable. Then we remove the 'Outcome' column from the data and compute the correlation and visualise.



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Pregnancies	1.00	0.13	0.14	-0.08	-0.07	0.02	-0.03	0.54
Glucose	0.13	1.00	0.15	0.06	0.33	0.22	0.14	0.26
BloodPressure	0.14	0.15	1.00	0.21	0.09	0.28	0.04	0.24
SkinThickness	-0.08	0.06	0.21	1.00	0.44	0.39	0.18	-0.11
Insulin	-0.07	0.33	0.09	0.44	1.00	0.20	0.19	-0.04
BMI	0.02	0.22	0.28	0.39	0.20	1.00	0.14	0.04
DiabetesPedigreeFunction	-0.03	0.14	0.04	0.18	0.19	0.14	1.00	0.03
Age	0.54	0.26	0.24	-0.11	-0.04	0.04	0.03	1.00

Figure 2: Correlation Heatmap without Outcome

2. ***Remove the disguised values from the table. We need to remove the values that equal to 0 from columns BloodPressure, SkinThickness and BMI as these are missing values but they have been replaced by the value 0. Remove the value but keep the record (i.e.) change the value to null.***

Here, we replace 0 with null using np.nan to represent missing values.

3. ***Fill the cells with null using the mean values of the records that have the same class label.(meaning of same class label?) with outcome?***

For all the null records, we calculate mean for class 0 and class 1 and replace null records with mean for each class.

4. ***Compute the correlation between the different columns.(with outcome?)***

After handling missing values, we compute the correlation with data that includes the ‘Outcome’ column and visualize.

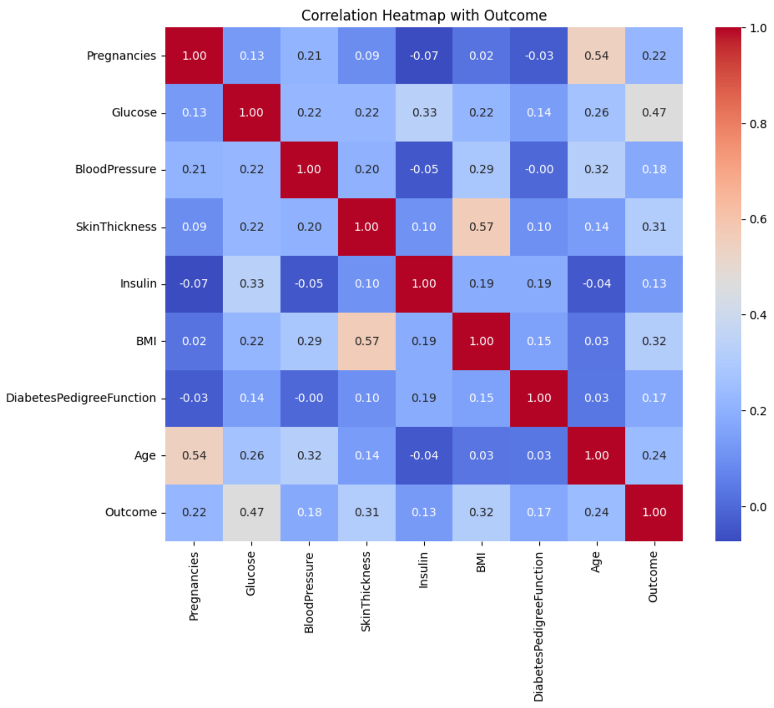


Figure 3: Correlation Heatmap with Outcome

5. ***Compare the values from this step with the values in the first step (just mention the most important changes (if any)) and comment on your findings.***

Here, we combined both correlation matrices into one and found that only the rows or columns involving **BloodPressure**, **SkinThickness**, and **BMI** show differences between the two matrices. Therefore, we focus on the rows corresponding to *BloodPressure*, *SkinThickness*, *BMI*, and *Outcome*.

BloodPressure

We observe that the correlations of BloodPressure with **Pregnancies**, **Glucose**, **BMI**, and **Age** all decrease after removing *Outcome*, while the correlations with **SkinThickness**, **Insulin**, and **DiabetesPedigreeFunction** increase.

SkinThickness

We observe that the correlations of SkinThickness with **Pregnancies**, **Glucose**, **BMI**, and **Age** decrease after removing *Outcome*, while the correlations with **BloodPressure**, **Insulin**, and **DiabetesPedigreeFunction** increase. Notably, the correlation with **Insulin** was weak before removing *Outcome*, but becomes moderate afterward, suggesting a stronger relationship between the two variables once the *Outcome* is excluded.

BMI

We observe that the correlations of BMI with **Pregnancies**, **BloodPressure**, **SkinThickness**, and **DiabetesPedigreeFunction** decrease after removing *Outcome*, while the correlations with **Glucose**, **Insulin**, and **Age** increase. The correlation between **SkinThickness** and **BMI** was strong before removing *Outcome*, but becomes weaker afterward.

Conclusion

In conclusion, there is no consistent pattern of increase or decrease in correlation values. However, the correlations clearly differ when *Outcome* is included as types (1, 0). This happens because the mean and sample size distributions change, which affects the correlation calculation. Here, *Outcome* is actually a **binary categorical variable (1, 0)** and not continuous, whereas `.corr()` assumes that all variables are continuous. For example, *SkinThickness* has a correlation of **0.436** (moderate) with *Insulin* when *Outcome* is excluded, but only **0.104** (very weak) when *Outcome* is included. Medically, individuals with higher skin thickness tend to have **higher insulin levels**, reflecting **greater insulin resistance**, which aligns with the observed positive correlation. Therefore, when analyzing correlations among continuous features, it is more appropriate to remove the *Outcome* column before performing the correlation analysis.

Submission

You need to submit a zip file that includes:

1. A pdf file that reports and explains how you answered the questions.
2. A Python notebook that includes the code with markdown boxes to specify the question under consideration. Use comments to explain the important steps in your code. The explanation of the methods in general with discussion about the most important steps in the code should be included in the report (the pdf file).

Submission should be done on blackboard and only one submission per group is needed. You can submit as many versions as you like but only the last submission before the deadline will be graded.

Deadline: as specified in the course website (2025-10-07 9:00AM).

BIBLIOGRAPHY

Abedjan, Z., Golab, L., & Naumann, F. (2015). Profiling relational data: A survey. *The VLDB Journal, 24*(4), 557–581. https://doi.org/10.1007/s00778-015-0389-y