UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

DATA DISCOVERY THROUGH PROFILE-BASED SIMILARITY METRICS

MARC MAYNOU YELAMOS

Thesis supervisor: SERGI NADAL FRANCESCH (Department of Service and Information System Engineering)

Degree: Master's degree in data science

Thesis report

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

29/06/2023

Acknowledgements

First and foremost I would like to thank my supervisor Sergi Nadal for giving me the opportunity to develop this Thesis as well as his guidance throughout the past months. His continued interest and enthusiasm in the project have been pivotal in the completion of this work, and his advice has made of this piece something I can be proud of. I would also like to thank two of my former professors: Ernest Teniente and Xavier Oriol, who encouraged me to apply for this degree and allowed me to make a humble contribution to the research world. My appreciation goes to all the people that I have encountered throughout my six years in university, both colleagues and professors, that have allowed me to learn, grow and made of this journey something enjoyable.

Lastly, I would like to thank the family and friends that have supported me and accompanied me during all these years. My heart goes out for my parents, by far the two most important people in my life and the sole reason I have been able to get this far in all aspects of my life.

Abstract

The most essential step in a data integration process is to find the datasets whose combined information provides relevant insights. This task, defined as data discovery, is highly dependent on the definition of the similarity between the candidate attributes to join, which commonly involves assessing the closeness of the semantic concepts that the two attributes represent. Most of the state-of-the-art approaches to this issue rely on syntactic methodologies, that is, procedures in which the instances of the two columns are compared to determine whether they are similar or not. These approaches suffice when the two sets of instances share the same syntactic representation but fail to detect cases in which the same semantic idea is represented by different sets of values. This latter case is ever-increasing in proportion, given the characteristics of big-data environments and the lack of standardization of the data. The aim of this project is to develop a system that can solve this issue and facilitate the establishment of relationships between related data that do not share a syntactic relationship. The approach presented in this work leverages the extensively studied syntactic methodologies to data discovery in conjunction with a new formulation for semantic similarity: the resemblance of probability distributions. Additionally, this system will be made scalable and able to handle vast quantities of data.

Table of Contents

List of Figures

List of Tables

Introduction

1.1 Data Discovery

Data discovery (or dataset discovery) is defined as the process of navigating through numerous data sources in order to find relevant datasets for a given task or gain insights from massive data collections [1, 2]. It focuses on understanding the content of the data, uncovering all sorts of trends and, particularly, unveiling relationships among them. It is a crucial task in the early stages of any data science pipeline [3], given that it increases the number of sources and information available to perform posterior analyses. Data discovery is the first step in data integration, a broader process centered around bringing different data sources together into a single, unified view, allowing data analysts to work with these different sources seamlessly.

A correct and efficient implementation of data discovery tasks is essential in today's world due to the ever-increasing number of data sources, which both raises the amount of potentially useful data as well as complicating its detection amidst vast repositories of irrelevant data. This is more so given the popularization of data lakes and other large-scale data storage systems. These repositories can contain exabytes of heterogeneous datasets, making it infeasible for analysts to manually find relevant data. Despite data discovery being a pivotal step for any data science pipeline, it is still a task performed largely in a manual manner, which some statistics quantify as up to 80% of a data scientists' time [4]. These user-driven processes suffice when the data is limited and accurately characterized, two conditions that current large-scale scenarios cannot guarantee. As such, data discovery is a necessity [5], and the automatization of these processes is mandatory for the enhancement of data treatment and the opening of a plethora of new research paths to explore.

1.2 Joinable columns and similarity

Data discovery encompasses several tasks: from finding related datasets to discovering unionable columns. Of particular interest for the data management community, and the focus of this thesis, is the task of *automatically discovering joinable attributes among different datasets*. By *joinable* we adhere to the traditional definition of a *relational* join, where any pair of columns that refer to the same semantic entity can have their values joined into a single column, facilitating the posterior merging of the additional data in both datasets onto a single, unified view. Based on this definition, there is a crucial issue that needs to be addressed: *how do we define the joinability of two columns?* Alternatively, we can ask: *given two columns, how joinable are they?*

The problem boils down to the definition of the *similarity* between two columns. Similarity can be described in many ways, but in a data integration context a common characterization is by how close the semantic concepts that the two columns represent are (e.g. people, countries, movies) [6]. That is, ideally we want to join those columns that represent the

exact same semantics, ensuring that the combination of the two original datasets provides coherent and useful information. The following two examples present the relevance of accurately characterizing this definition.

Nation	Population (Millions)
Spain	47.5
United States	334.0
France	64.7
Japan	125.7
China	1,412.6

Table 1: Population per country

Country name	GDP (Trillions $)
Spain	1.50
United States	26.85
France	3.09
Japan	4.97
China	21.87

Table 2: GDP per country (country names)

Country code	GDP (Trillions $)
ESP	1.50
USA	26.85
FRA	3.09
JPN	4.97
CHN	21.87

Table 3: GDP per country (ISO codes)

Example 1: Columns with the same values

Table 1 and Table 28 present, respectively, the population and GDP of a series of countries. These countries are identified in the same manner for both sets, through its full name, which implies that the sets of values are identical. These two columns (*Nation* and *Country name*) are, as a result, highly similar, due to the fact that they represent the same concept (the idea of a country) and do so in the same manner. Executing a join would generate sensible results, as the information of each country present in both datasets would be combined into a single one, facilitating further analysis, such as the obtainment of the GDP *per capita*.

Example 2: Columns with different values

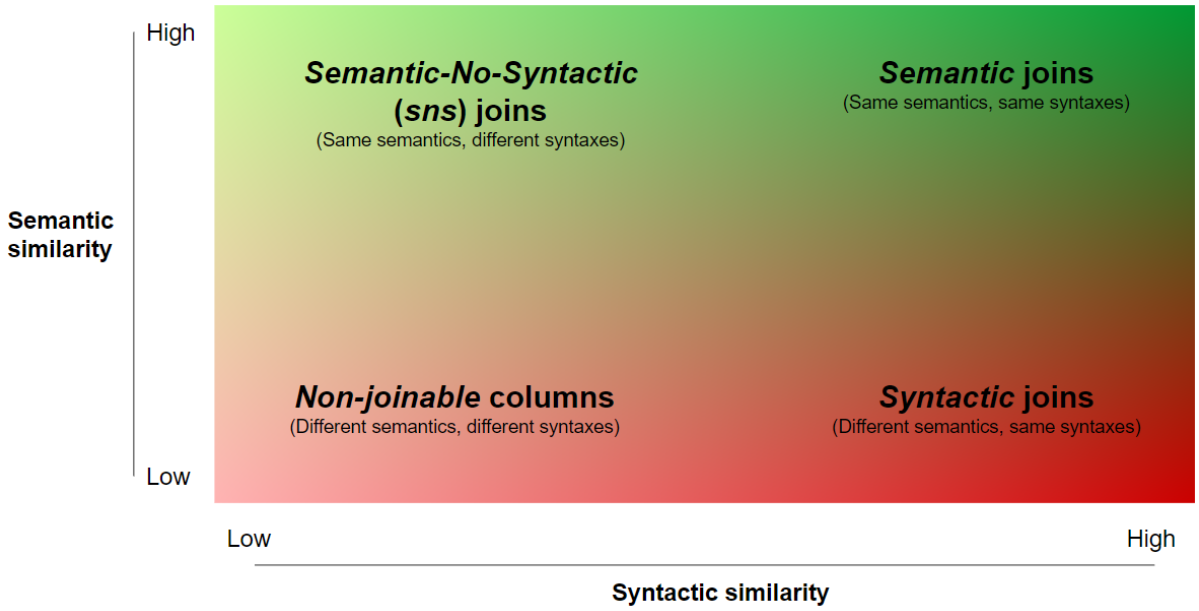
In example 1 the similarity between the columns is made evident due to the sharing of values between the two. However, a more interesting problem emerges when the two columns that represent the country do so with a different format. This is the case for Table 1 and Table 3: one table uses full names whereas the other uses ISO codes. Both sets of data still represent the same semantic concept, so their similarity should be high despite the difference in the used syntax. However, this differentiation between the representation of the values complicates the assessment of the similarity, as it can not be based purely on the intersection of the sets. For instance, a traditional equi-join would fail to find any matching, as it is purely based on the equality of the values.

Following the previous discussion, we are interested in detecting all cases in which the semantics of two columns are the same, regardless of whether they share the same values. As such, the definition of similarity has to be able to indicate how close the semantics

of two columns are (ideally with a numerical quantification) and, hence, whether a join should be executed or not. Defining a *good* similarity metric is essential, as it should give a precise orientation on whether a join is meaningful or not. It is difficult to define what good entails, as establishing how semantically close two columns are is a complex task. Additionally, this issue can be made more troublesome if this metric had to rank the joins. This would imply assigning a similarity score to each candidate, in which the pairing with score *s1* would be better suited for a join than a pairing with score *s2* as long as $s1 > s2$.

1.3 Different notions of joinability

The complexity of defining a similarity measure can be observed in the previous examples, as two columns might present the same semantic concept but their representations might not align. More precisely, Examples 1 and 2 present two different configurations of joins: semantically and syntactically related, and semantically but non-syntactically related. Having these two cases implies that there are two more typologies of joins that can be defined, representing the opposite scenarios to the ones described. The following figure summarizes this classification:



The color indicates the semantic similarity, with green being those joins that share semantic concepts (and, therefore, those that are interesting from a data discovery perspective) and red for those that do not. The intensity of the color highlights the degree of syntactic similarity.

Figure 1: Join typology classification

For the sake of simplicity, the names indicated in each sector will be used to define the type of join from this point onward. Thus, joins with the same semantics and a different syntactic representation (example 2), will be labeled as *sns* joins ("semantic-no-syntactic"), and joins with similar semantics and syntax (example 1) will be referred as *semantic* joins. As for the new typologies, candidate pairings that are *non-joinable* should be discarded by any data discovery process, as they share neither semantics nor syntax. However, *syntactic* joins pose a more interesting challenge. An example is presented next:

Title	Original language	Author
Robinson Crusoe	ENG	Daniel Defoe
One Hundred Years of Solitude	SPA	Gabriel García Márquez
The Art of War	CHN	Sun Tzu
The Book of Five Rings	JPN	Miyamoto Musashi

Table 4: Book characteristics

Example 3: *Syntactic* joins
Table 4 contains information about books, and one of the columns represents the language each book is written in, doing so through an abbreviation (SPA for Spanish, ENG for English, etc.). If Table 4 is compared with Table 3 a particular interaction can be noticed when assessing the potential join between *Country code* and *Original language*. Although the language and the country are related, these are completely different domains, which happen to be described by, in some cases, the same values. This is the case for *CHN* and *JPN*, which can be use to reference China/Chinese and Japan/Japanese. Executing an equi-join would produce some matches, as the syntax do align. However, the semantics are totally different, so a join between the two columns is unlikely to bring relevant information. That is, it would combine book titles and authors with the GDP of a country. As such, it is crucial to realize that non-sophisticated, purely syntactic-based systems might be fooled and label this scenario as highly similar, which is not the desired outcome.

An extensively studied approach to analyze the similarity of two columns has been to leverage the comparison of the two sets of values, developing several syntactic-based methodologies to determine the joinability of two columns. That is, if the sets of values resemble each other, the columns are defined as similar, such as in example 1. However, examples 2 and 3 showcase that such a notion of similarity does not suffice, as it misses *sns* joins and generates false positives for *syntactic* joins. Figure 1 identifies the issue with syntactic-based methods: they assume that a syntactic similarity implies a semantic similarity, which, based on the presented examples, is not the case. This claim can be proved empirically.

Throughout this work a ground truth that presents the four aforementioned types of joins will be used. This dataset is a modified version of the data generated by [7], in which a similar categorization of the presented join typologies is indicated. The data collects a series of pairings between columns of several datasets, indicating, for each pairing, the type of join. It contains 2703 *syntactic* relationships, 1701 *semantic* relationships, 2547 *non-joinable* columns and 532 *sns* relationships. The semantic similarity of *sns* and *semantic* joins was assessed manually, and the criteria to separate the two categories was the following: if a pairing of columns could increase the amount of values shared by executing some kind of transformation (i.e. mapping function), it was labeled as an *sns*. For instance, the columns described in example 2 would form an *sns* join, as transforming each country code to its full name would improve the amount of values shared.

The experimentation was as follows: for each of the joins in the ground truth two of the most common syntactic-comparison metrics were calculated: the containment of values and the Jaccard similarity. These metrics were then leveraged by a simple Machine

Learning model to try to predict the category of a join. That is, the model would try to correctly classify each join based purely on a syntactic contrast of the values. The results are presented in the following figure:

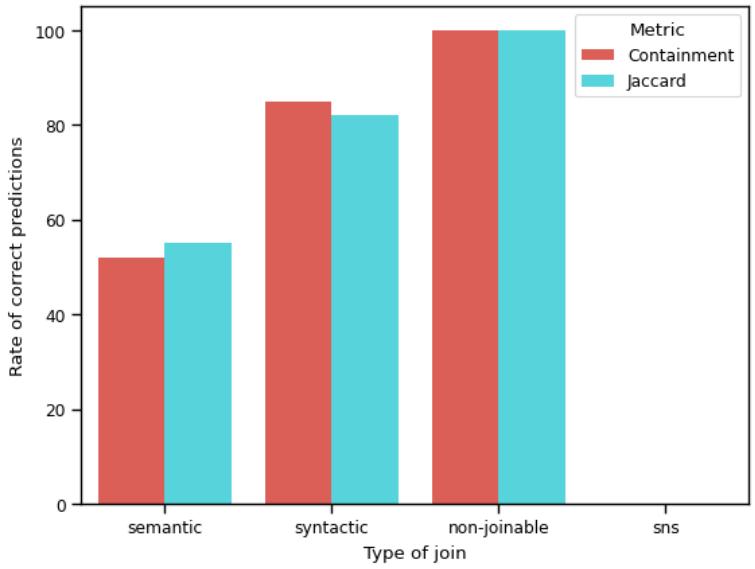


Figure 2: Rate of correct join classification with Jaccard similarity and containment

Firstly, we can observe that the model struggles in classifying *semantic* joins. This is so because *semantic* and *syntactic* joins are the two typologies that present a high degree of value intersection, and without a more nuanced definition of similarity the model can not leverage more information to set them apart. As a result, almost half of the *semantic* joins are classified as *syntactic*, as the patterns that differentiate the two are not clear due to the limitations of the approach. On the other hand, all *non-joinable* columns are correctly classified whereas all *sns* joins are missclassified. Contrary to the *semantic-syntactic* comparison in which the amount of shared values varies within a relatively high range, for this two types of joins the intersection of values is almost always 0. Therefore, from a syntactical perspective, it is impossible to differentiate *sns* from *non-joinable* columns. Given that there are more *non-joinable* columns than *sns* joins, the model develops a bias towards the more populated class, completely missing the other. This experiment highlights the need for a more nuanced definition of similarity.

1.4 The *sns* detection problem

This work is going to focus on the detection of *sns* joins, given that current state-of-the-art methods do not fulfill this task. The *sns* detection problem is similar to what the Entity Resolution field aims to achieve, albeit with some differentiations. Entity Resolution aims at identifying, out of collections of data, entities that represent the same real-world object [8], its first step being the candidate selection, in which the systems determine the entities worth comparing [9]. This first task resembles the described problem, although Entity Resolution techniques are not suited to work with large quantities of data, which is an imperative condition in data discovery mechanisms.

The detection of *sns* joins in large-scale repositories is a problem yet to be thoroughly addressed by the community. The lack of a dedicated system able to execute this process poses a considerable issue in the current big data environment. The generalized lack of standardization when representing a given domain is exacerbated by an ever-increasing amount of data available online, with no conventions over its format. This results in a vast number of datasets with columns that identify the same conceptual idea but through widely different formats, its detection being unachieved by syntactically-focused state-of-the-art data discovery processes.

Nonetheless, correctly assessing the similarity of these cases opens up the possibility of combining substantially more sources, which ultimately allows for the extraction of more nuanced information. As such, developing a method that can detect similar semantics when there is no similarity of the values is pivotal for furthering the analysis capabilities of the data science field. Creating such a system is the main purpose of this project.

Figure 1 establishes the distinctions between the four possible typologies that a join can be classified as, based on two concepts: syntactic similarity and semantic similarity. This differentiation is crucial, as it identifies that to fully isolate *sns* joins it is required to have, on one hand, a measure of the syntactic similarity (*horizontal* axis), and, on the other, a notion that can represent the semantic similarity (*vertical* axis). This, essentially, identifies a more specific definition of similarity between two columns, leveraging the combined results of two different measurements. Only by encompassing both factors *sns* joins can be thoroughly separated from the other types, fulfilling the purpose of this project.

1.5 Contributions

This Master Thesis has as its main objective the exploration of a pivotal issue in the data discovery field: developing and implementing a method that can accurately and efficiently discover semantically-joinable pairs of attributes over data lakes, with especial emphasis on the cases where columns have shared semantics but present different syntax. More precisely, the goal of this Master Thesis is three-fold:

- Firstly, it will encompass the development of such a method to identify semantic non-syntactic relationships. This will be achieved by leveraging, besides the commonly used intersection of data values (a purely syntactic approach), statistical properties of the data, aiming at developing a more complex notion of similarity.
- Secondly, the detection of these cases will be made scalable for a large number of datasets. This is crucial, as if the developed system is not able to deal with current large-scale repositories of data, its usage becomes very limited. This will be done by reducing the information of the datasets to profiles: brief and precomputable characterizations of the columns.
- Finally, this process will be integrated into a data science environment, where its usage can be easily accessed and incorporated into data processing pipelines. This requires the implementation of a process that is able to identify all possible semantically-related candidates (regardless of their syntactic or non-syntactic nature).

1.6 Outline

The remainder of the thesis is as follows. Chapter 2 presents the related work on data discovery tasks. Chapter 3 includes a high level explanation of the implemented solution for the *sns* detection problem. Chapters 4 and 5 develop systems to tests the stated hypothesis and empirically prove their validity for the task at hand. Chapter 6 ensures the scalability of the system in large-scale data repositories by applying the required modifications. Chapter 7 explores the usage of external datasets to prove the generalization of the approach. Chapter 8 provides a closing to the project, highlighting a final set of conclusions.

Related work

2.1 Syntactic-based approaches

A vast number of state-of-the-art systems rely on value comparison to assess the similarity of two columns (e.g. containment, Jaccard similarity, cosine similarity, etc.), commonly leveraging auxiliary data structures or approximate algorithms to optimize its calculation. In order to illustrate this idea, the next paragraphs detail the current main approaches to syntactic-based similarity detection.

Comparison by value: direct comparison of the values of both columns to calculate the percentage of intersection. These systems are, essentially, solving the (overlapping) set similarity search problem but optimizing its computation through the usage of auxiliary data structures (e.g. inverted indices, dictionaries) to minimize the look-up cost and make these computations feasible for large-scale data. An example of it is **JOSIE** [10], which creates an inverted dictionary of the data that allows it to quickly retrieve tables that contain specific data elements. This is followed by techniques to reduce the amount of dictionary entries that are read, mainly the estimation of the intersection size between two columns without having to actually compute the overlap.

Comparison by hash: usage of hash functions to map data into small, fixed-size representations of this same data, generating signatures that can be easily accessed and compared to find joins. More precisely, the most common implementation of this approach uses locality-sensitive hashing (LSH) schemes, in which the hashes (that is, signatures that represent the data) are stored in buckets, with a high probability of assigning similar hashes to the same buckets. Examples of this method are **D3L** [2], which uses metadata of the data to define the LSH index, or **Aurum** [11], which, on top of developing an LSH index, defines a graph that tracks the similarity of the signatures.

Comparison by profiles: profiles are defined as collections of metadata, presenting relevant information to understand the underlying properties of the data, doing so in a summarized manner. Reducing data to a brief representation makes this method adequate for large-scale data repositories, whilst still presenting a way to tackle the similarity assessment problem, as well-crafted profiles are sufficiently representative of the initial set of data to perform such a task. **SIMUBC** [12] selects 28 binary metrics, which can be numeric, string/token-based or phonetic. **NextiaJD** [7] employs metrics to measure cardinalities, value distributions and syntactic properties. Data Lake navigator (**DLN**) [13] uses software-specific capabilities to include data queries in their profiles.

As introduced beforehand, these methods are unable to detect those cases in which semantically similar columns present different instances of values, as a purely syntactic approach can not abstract the semantics of a column beyond the intersection of values. As such, they completely miss *sns* joins and the insight they could bring to further analyses.

2.2 Non-syntactic-based approaches

The systems described in this section present notions of similarity that do not leverage the intersection of values, at least not as a unique factor to determine the joinability of two columns. As the body of work is not that extensive as with syntactic-based methods, it harder to elaborate a distinction by typologies. Nonetheless, some common proposals are described next:

Embedding-based discovery: embeddings are high-dimensional representations of values, which present an implicit method to capture the underlying semantics of the data. Using embeddings and ontologies to relate semantic concepts through standardized nomenclature has been further studied (**SEMPROP** [14]). SEMPROP uses embeddings on word names to find attributes in a data lake that respond to a given semantic type. A more complex implementation of this idea involves the computation of embeddings for every value of the attribute. In this second scenario, each column is converted to a set of high-dimensional vectors that can be compared with other sets to truly assess its similarity. **PEXESO** [15] directly defines the similarity of two columns as the proximity of the embeddings of the instances of both columns. **WarpGate** [16] incorporates several optimizations to the comparison of embeddings, such as the use of LSH indexes.

Comparison of statistics: statistical procedures and measurements are leveraged to assess the similarity of two columns. The reasoning behind this approach is that statistical properties of the data highlight the relationships that are hidden underneath the values that can not be detected by a pure syntactical comparison. Some examples are the comparison of distributions to create clusters of columns followed by the execution of syntactic-based filtering [17], leveraging big table corpora to "look-up" and detect correlations between attributes (**SEMA-JOIN**, [18]) or executing post-statistical-analysis data transformations (i.e. operations to modify or enhance the data) to produce the joins (**Auto-Join**, [19]).

Entity Resolution methods: the first step in Entity Resolution tasks is to identify the entities worth comparing. Without this step, eEntity Resolution would suffer from a quadratic time complexity, as every entity needs to be compared with all others entities. The Entity Resolutionn field uses techniques to prune the amount of comparisons, the two most dominant frameworks being blocking and filtering [9]. The former focuses on the identification of entities that are likely to represent the same object, restricting comparisons to only these pairs, while the latter attempts to discard pairs that are guaranteed to not match, executing comparisons only on the rest of entities.

All of these approaches presents measures of semantic similarity that could address the *sns* detection problem. However, they all present issues that difficult its application for the task, specially so in large-scale environments. Embedding-based systems still require pairwise comparisons among the sets of embeddings, the usage of statistics is mostly designed to operate at a table-scale and eEntity Resolution techniques present efficiency issues when handling vast quantities of data. These problems will be further developed in the following section.

2.3 Research gap

SEMPROP finds related columns based on the semantics, but it does not consider whether joining these attributes produces a meaningful result. SEMA-JOIN and Auto-Join are designed to operate on a table-scale, finding the best way to generate an automatic join for a user-defined pairing of columns. Entity resolution is differentiated from data discovery in that the techniques they develop are not meant to be employed in an early exploratory phase over vast amounts of heterogeneous data sources, which require optimized algorithms to operate properly due to the vast amount of comparisons. As highlighted by [20], the processing costs of Entity Resolution techniques are excessively expensive, and more lightweight algorithms are desired.

DL3, PEXESO, WarpGate and [17] all propose systems that could handle the detection of joins with the same semantics but different syntax in moderate-scale repositories of data. However, by closely analyzing their research, it is clear to see that none of them have executed specific experimentations for the described task. That is, they do not draw an explicit separation between *sns* and *semantic* joins, grouping in the same category joins with similar semantics regardless of their syntactic similarity. This makes it impossible to discern the applicability of these processes for the described problem, as it is unknown which percentage of their accuracy rate belongs to each type.

Another issue with these approaches is that they rely in some manner of pairwise comparison. [17] executes a set intersection after the distribution-aided clustering and the embedding-based methods generate and compare embeddings for each value in the columns. Although the employment of filtering techniques or of optimized/approximate algorithms reduces the number of comparisons, the computational complexity of these processes is still concerned with the size of the datasets, as more values imply more comparisons of embeddings. Moreover, the embeddings need to be loaded into memory to be used, which might be an additional hurdle for datasets with millions of instances.

As a result, we can identify two separated issues to address:

1. It is necessary to develop a thorough study on how to detect *sns* relationships, given the relevance that they have in data integration tasks and how little research has been conducted for its specific isolation.

2. It is important that whichever system aims to detect these kinds of joins can do so in an efficient manner, reducing the amount of comparisons to the minimum possible and allowing it to scale effectively.

Measuring *sns* joins

3.1 Characteristics of semantic relations

Syntactic similarity

As presented in Section 2.1, the data science community has designed a plethora of systems that employ some variant of the syntactic similarity as a measure of the semantic similarity. However, these systems use the intersection of sets as an overall measurement of the similarity between two columns, whereas in this work this notion of similarity is only part of the defined solution. That is, the syntactic similarity is not a substitute of the semantic similarity, but rather a complementary metric to assess the overall joinability.

To back up this claim, we conducted a similar experiment to that of section 1.3 and figure 2, where we used a syntactic-based metric (in this case only the containment) to distinguish the four type of joins. However, here we will go more in depth, as, besides a model to classify the four typologies of joins at once, three smaller, binary models will be built, each separating *sns* joins from one other type of join only by leveraging the containment. Both the models and the evaluation metrics presented in the following table will serve as the foundation for the experiments of this entire work, and the details and reasoning behind its employment are described in sections 4.2.1 and 4.2.2, respectively. The results are as follows:

Model	**F1-score (macro)**	**Accuracy**	**F1-score (*sns*)**	**Recall (*sns*)**	**Precision (*sns*)**
Multilabel	57.24%	76.74%	0.00%	0.00%	0.00%
sns vs. *semantic*	-	-	100.00%	100.00%	100.00%
sns vs. *syntactic*	-	-	100.00%	100.00%	100.00%
sns vs. *non-joinable*	-	-	0.00%	0.00%	0.00%

Table 5: *sns* detection based on the containment of columns

The general metrics indicate a poor overall performance of the multilabel model, which correlates to the experimentation conducted in section 1.3. The *sns*-specific metrics highlight that the multilabel model is not able to detect a single *sns* join correctly, which is, once again, the behavior observed at the beginning of this document. These results are also coherent with figure 1, as a purely syntactic approach should not suffice in the detection of *sns* joins. However, such drastic results demand a more careful exploration, which can be achieved with the binary models. Once again, the results are consistent with Figure 1, as the purely syntactic-based approach is able to perfectly differentiate *semantic* and *syntactic* joins from *sns* joins (vertical axis), but it is incapable of separating *sns* joins from *non-joinable* columns (horizontal axis).

This explains why the *sns* joins can not be detected. From a syntactical perspective, the two groups can not be differentiated, and the model develops a bias towards labeling *sns* joins as *non-joinable* columns, given that there are more of the latter type. A bigger proportion of *non-joinable* columns is the expected behavior in real-life scenarios, as most columns in a data repository do not share either semantics or syntax. Therefore, a data discovery system that includes only syntactic-based methods for similarity assessment will always fail to detect *sns* joins.

Non-syntactic measures for semantic similarity

The presented results are the empirical demonstration that a syntactic measurement of the joins is necessary, but not sufficient, to solve the *sns* detection problem. That is, we still need to leverage it to fully isolate *sns* joins, but it needs to act in conjunction with some other measurement to be completely effective. More precisely, we need a mechanism to quantify the degree of semantic similarity between the columns which, additionally, can not leverage the syntactical similarity. That is, we need a non-syntactic system to assess semantic similarity. The starting point of this exploration will be the following: *the semantic similarity of two columns can be defined by comparing their probability distributions* [17]. Two examples to present the intuition behind this claim are presented next:

> **Example 4: Distribution comparison for *sns* and *semantic* joins**
> Going back to examples 1 and 2, if we closely analyze the two columns that define the countries in both scenarios we can detect a common characteristic: the probability distribution of these values. That is, every country is going to appear a single time in both datasets, regardless of whether they do so in the same way (complete name for both columns) or in a different manner (complete name for one column and ISO code for the other). Therefore, the probability distribution of these columns is the same: uniform, with the probability of every value being one divided by the number of countries. We could develop a preliminary conclusion from this scenario: columns with the same semantics have the same distribution of values.

> **Example 5: Distribution comparison for *syntactic* and *non-joinable* columns**
> The assumption made in example 4 can be applied inversely: columns that do not share the same semantics present different distributions, regardless, once again, of whether they share the same values or not. This can be easily seen in example 3, in which the columns present the same instances but not the same semantics. The distribution of languages of books is not uniform, as some languages (such as English) will appear considerably more times than others. This will produce a Pareto distribution, which is considerably different than a uniform distribution. Therefore, we can develop a second preliminary conclusion: columns with different semantics will have different distributions of values.

Logically, this reasoning is meant to define a general trend in the behavior of the pairings of columns. As such, some cases might not perfectly align with the presented claim. Moreover, it is hardly the case that two columns that share the same semantics are going to present *exactly* the same probability distributions (besides some specific cases, such as the one presented above). Therefore, a more general hypothesis needs to be stated: *two columns represent a similar semantic concept if their distributions resemble each other*.

The opposite statement might be more intuitive: *two columns that do not present any kind of semantic relationship will likely have different distributions of values*. In order to assess whether these claims are valid or not, an experimentation needs to be conducted, as, oppositely to the set-intersection problems, the comparison of distributions has not been thoroughly explored as a method to assess the similarity of two columns, and less so for the detection of *sns* joins. Before, however, it is important to define how the comparison of distributions will be performed.

3.2 Statistics and tests to compare distributions

In [17] the selected algorithm to compare distributions is an optimized version of the Earth Mover's Distance algorithm (also known as the Wasserstein distance). The EMD is a measure of dissimilarity between frequency distributions, informally defined as the cost of "transforming" one distribution of values into the other. The employed version of the algorithm is a complex process that includes several stages, one of them being the computation of the containment of values for the columns. As such, the system is difficult to employ and time-consuming to execute. A more direct approach, that still follows the same comparison-distribution principle, could be defined by employing the **usage of statistical tests to determine if the distributions are significantly different**.

The first important aspect to highlight is that this work focuses on non-numerical columns, given that assessing the semantic resemblance of two sets of numbers is significantly harder. However, probability distributions can not be constructed with strings, so a conversion needs to be made. This transformation will involve changing the string values by *numerical probabilities*. More precisely, the frequency of every string will be divided by the cardinality of the column, which will indicate the probability of a given string.

> **Example 6: String columns transformation**
> Column *Original language* of Table 4 presents the following values: [*ENG*, *SPA*, *CHN*, *JPN*]. There are four different values and each appears a single time, so the probability of each string is 0.25. The set of probabilities that defines this columns is the following: [0.25, 0.25, 0.25, 0.25]. If we change the values of *Original language* to be [*ENG*, *ENG*, *CHN*, *CHN*, *CHN*], we observe that some instances present the same string, so the probability of these strings will increase. For this latter case the probability set would be the following: [0.4, 0.6]. Notice that the "size" of the set has been reduced from 5 strings to 2 probabilities, as there are repeated values that "merge" into a single probability. By applying this transformation, a set of string-based values is transformed into a numerical representation of their probability distribution, which can be used to draw comparisons.

Statistical tests are tools to mathematically assess whether two sets of data are significantly different from each other, leveraging certain statistical measures to do so, such as the mean, median or the standard deviation. As the string-values will be converted to sets of probabilities, the tests will determine if these same sets of probabilities are different, thus indicating whether the underlying distributions of the values align. If several of the tests are used and they all determine that the groups of probabilities are not different, then we can assume that the distribution is the same. This approach, however, poses a major issue.

Statistical tests are designed neither to prove nor disprove a hypothesis, but rather to showcase that an idea is untenable when its evaluation leads to an unsatisfactorily small probability [21]. As such, an arbitrary threshold is selected to indicate the point in which there is enough evidence to reject the initial hypothesis in a practical manner, this boundary commonly being 5%. That is, if the probability of a given hypothesis in the defined scenario (p-value) is below 5%, we assume the hypothesis to not hold. This, however, is just a convention to establish significance. As a result, employing only statistical tests might be too restrictive of an approach, and can fail to capture more nuanced patterns in the comparison of the distributions. Moreover, it is unlikely that the distributions of two columns are going to match perfectly, even if they have an *sns* relationship.

Using the p-values instead of a binary decision over the result of the tests might allow for some leniency when assessing the similarities of two sets of values (as p-values can present any number in the interval [0,1]). However, the methodology is still likely to miss more detailed insight, so the presented problem is still unresolved. In order to palliate this issue the p-values from the statistical tests can be combined with the **comparison of metrics that describe general properties of the distributions**. For instance, the F-test uses the mean of the two sets of groups to determine whether they are different or not. However, we can also include a *direct comparison* of the means of the two groups to obtain a higher-level intuition of the closeness of the sets of probabilities. Including an entire set of several descriptive statistics about the distributions can present a less constraining approach that can generalize better in real-life scenarios.

Given the reasons stated, the comparison of distributions to assess semantic similarity will be performed by combining two types of evaluation metrics. One one hand, statistical tests will determine if there is a significant difference between the two sets of probabilities of the columns. On the other hand, descriptive statistics will provide a more general comparison of how related two columns are leveraging properties of the data. Using both tests and descriptive statistics to compare distributions is the desired compromise between correctly assessing the closeness of the sets of data while allowing some leniency to develop a more generalized approach. Figure 3 illustrates the process of generating the metrics.

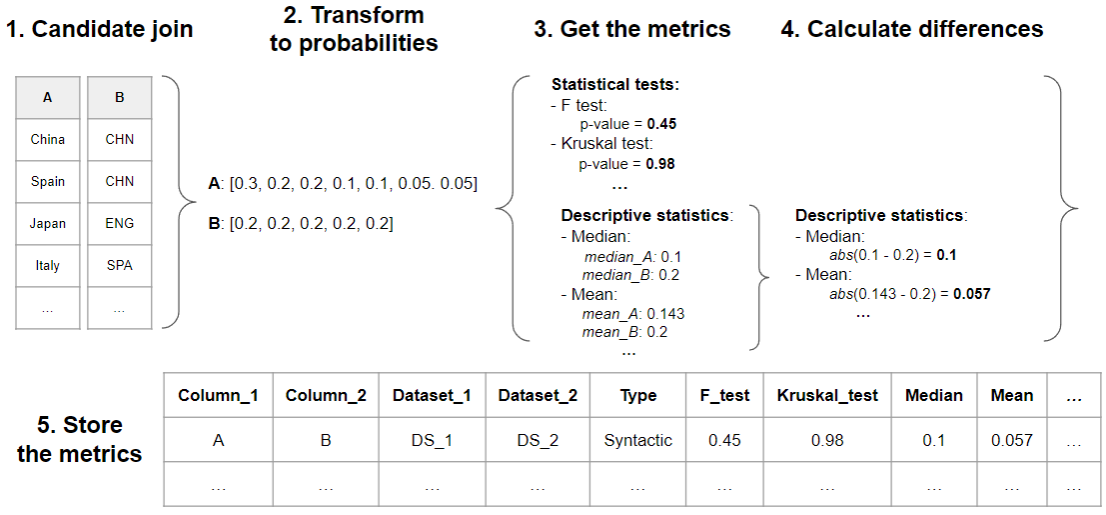


Figure 3: Generation of the metrics

3.3 Preliminary validity assessment of the approach

Studying the validity of the described methodology needs to be done through a nuanced exploration of the capabilities of the selected measures in the detection of the *sns* joins, which will be done throughout section 4 via the development of a Machine Learning model. Nonetheless, to provide a preliminary justification to employ this approach, a statistical analysis was performed over the defined set of metrics, whose goal was to answer the following question: *do the values of some metrics present variations depending on the type of join?* That is, the values of the metrics *depend* on the type of relationship, or follow the same trend regardless of the typology of the join? If the answer is affirmative, the given metrics will be useful to separate among two or more of the types of joins.

The analysis was as follows. A series of statistical tests and descriptive statistics were selected to perform the *sns* detection problem (the entire set of metrics is presented in section 4.1.2). Each metric was calculated for every join and separated into four groups, one per type of join. Then, a statistical analysis was employed over the groups, comparing whether the sets of values were significantly different. For instance, we can compare the set of *mean probability* of the *sns* joins and the set of *mean probability* of the *semantic* joins. If the sets of values do present statistical difference it can be assumed that this metric is suitable for separating the two types of joins, as there is a pattern present. More precisely, the values of the *mean probability* present differences depending on the type of join, which implies that a Machine Learning model can leverage the knowledge stored in the values to produce correct classifications.

To develop these meta-comparisons, two tests were selected: the F test and the Kruskal test. They are, respectively, parametric and non-parametric variants of the same process: the F test uses the mean to assess the differences between two groups of data whereas the Kruskal test uses the median. Given that we can not predict the distribution of the values for the metrics, the Kruskal test and its more generalized approach seems to be more appropriate, although observing the results for the F test can complement the analysis.

These tests can only compare two groups of data at a time and, given that our more precise goal is to identify *sns* joins from the rest, we decided to only evaluate *sns* joins with the other categories in one-to-one comparisons. That is, *sns* joins and *semantic* joins as one comparison, *sns* joins and *syntactic* joins as another comparison and *sns* joins and *non-joinable* as a third comparison; for every metric and for both tests. The resulting tables illustrate the percentage of metrics that presented statistical differences between *sns* joins and other typologies of joins:

# of sig. differences	% of metrics
0	0.00%
1	17.14%
2	42.86%
3	40.00%

(a) F test

# of sig. differences	% of metrics
0	0.00%
1	2.86%
2	45.71%
3	51.43%

(b) Kruskal test

Table 6: Number of statistically different groups defined by the metrics

As it can be observed for both tests, all metrics were significant in separating *sns* joins from, at least, one other kind of join, with the vast majority of them being significant in separating *sns* joins from two or three joins. These results attest to the selected group of metrics being relevant for the problem that we are trying to solve. Nonetheless, this is only a statistical (and, therefore, theoretical) exploration of this relevance. Moreover, these comparisons are binary, which means that they might not be that useful when trying to separate all four types of join at once. In order to fully ascertain whether they are useful or not, a complete predictive model needs to be developed, which will put to the test the capabilities of these metrics for assessing the type of join.

Distribution comparison as a semantic similarity assessment

This section provides an in-depth description of the methodology used to assess if the comparison of distributions (done through a series of statistical tests and descriptive statistics) is suitable for the *sns* detection task. This analysis will be achieved by the development of Machine Learning models to predict this typology of joins based on the indicated predictors, encompassing all the relevant stages in a Machine Learning project. Despite the data discovery field employing a wide variety of techniques to define joinable columns, Machine Learning is the methodology that the community is featuring the most in recent years [4, 22], so it is an appropriate approach to explore this task.

4.1 Preparation

4.1.1 Gathering the candidate joins

The first step to develop the models is to find the data to be ingested. The described predictors include both statistical tests as well as descriptive statistics, which can be obtained by developing a program to generate them for every candidate join. The main hurdle to overcome is the gathering of the candidate joins in the first place, that is, the pairings of columns to use in the training process.

The lack of labeled data is a prevalent issue in both data science and data discovery [4]. This is due to the fact that data can present complex relationships that need to be defined by a domain expert, which might be expensive and time consuming to produce. This issue is made more troublesome by the fact that the *sns* detection problem has not been thoroughly addressed prior to this project, which implies that there is no set of data readily available that contains *sns* relationships, at least not openly accessible. The only resembling approximation is that of [7], which, as described at the beginning of this work, has been modified to serve as the ground truth throughout this development.

As we are interested in the detection of *sns* joins, we could approach this problem as a binary classification task. Joins that are *sns* would be the positive class, and any other kind of join would be part of the negative class. However, and given that we have examples of the four types of joins, we could extend the model to try to accurately classify all typologies of joins at the same time, while keeping our focus on *sns* relationships. This could give potential insight on how the different kinds of joins relate to each other. Moreover, if the metrics succeeded at accurately separating the four groups, it would be an indicator that these metrics would not be useful only to characterize *sns* joins, but to classify and assess the typology of a join as a whole.

4.1.2 Selecting the metrics

There are a lot of different metrics and tests that relate two distributions, as it is an extensively studied process in the field of statistics. Nonetheless, as no previous exploration has been conducted to assess whether this distribution-comparison-approach is useful for the defined problem, there is no predefined set of metrics to begin the process from. As such, a set of arbitrarily selected metrics has been defined, belonging to the two specified groups.

There are a plethora of different **statistical tests** that employ different measures and procedures, so we deemed it relevant to incorporate several of these to get a more complete picture, both parametric and nonparametric. We included tests to compare the means (F test, Kruskal test, etc.), variances (Bartlett test, Levene test, etc.), directly the underlying distribution (Kolmogorov-Smirnov test, Cramer-Von Mises test, etc.) or complementary aspects such as the scale of the distributions (Ansari test, Mood test). We also include the aforementioned Wasserstein distance as a close approximation of the EMD algorithm.

As for the **descriptive statistics**, the selected set is mainly composed of arbitrarily-chosen metrics that define statistical properties. These metrics are specified for one column only, allowing, therefore, the possibility of precomputing them and doing so only once. These include, among others, means, standard deviations, confidence intervals and entropy measures. As described in figure 3, these metrics represent the *difference* in the values of two metrics.

The following tables gather all the used metrics:

Category	**Sub-category**	**Metrics**
Descriptive statistics	"Basic" descriptive statistics	Mean, mode, median, standard deviation, minimum value, maximum value
	Distribution statistics	Skewness, entropy, kurtosis
	"Advanced" descriptive statistics	Geometric mean, harmonic mean, variation, mean absolute deviation, inter-quantile range, standard error of the mean
	Confidence intervals	Mean of CI, Minimum value of CI, Maximum value of CI, standard deviation of CI
Statistical tests	Compare means	F test, Alexander-Govern test, T test, Kruskal test
	Compare distributions	Mann-Whitney test, Kolmogorov-Smirnov test, Cramer-Von Mises test, Anderson-Darling test
	Compare scale parameters	Ansari test, Mood test
	Compare variances	Bartlett test, Levene test, Fligner test
	Others	Wasserstein distance

Table 7: List of metrics

Taking a quick look, it is easy to observe metrics whose likelihood of being correlated is high, such as the confidence interval of the mean and its maximum and minimum values. It is important to remember the exploratory nature of this project: the goal is to test statistical measures in the *sns* detection problem. Although correlated, some metrics might still bring nuance to the detection process, so keeping them might be beneficial. Nonetheless, a feature selection process will deal with this issue.

4.1.3 Preprocessing tasks

Generally, the preprocessing phase is the most time consuming part of a Machine learning project, given the numerous processes that need to be conducted in order to prepare the data for the predictive model. However, due to the unique nature of the data that we have, most of the main preprocessing tasks will not be necessary.

The p-values from statistical tests will always be probabilities in the range [0,1], so normalization is redundant and outlier detection does not identify any case, as the results display values for the entire range [0,1]. A similar principle can be applied to the descriptive statistics, as these are obtained from the set of probabilities of the strings of the columns, which are, once again, values in the range [0,1]. As a result, the mean, the standard deviation and other such metrics will also fall in the [0,1] range, deeming normalization and outlier detection unnecessary.

The only exceptions to these conditions are a few statistics, such as the entropy, that do not have defined boundaries. Nonetheless, the results of these metrics always present values in the same order of magnitude, making normalization tasks dispensable. As for outlier detection, only very aggressive thresholds indicated abnormal values, so no pruning was conducted to not negatively condition the values. These conclusions were further proved by an empirical analysis, in which applying normalization and outlier detection tasks before training the models resulted in marginally worse results.

The last relevant task in the preprocessing stage encompasses the execution of a feature selection job. In a typical Machine learning scenario, before developing the model a feature selection process (or several of them) should be conducted to remove unnecessary or bias-inducing variables. However, the goal of this model is to study the predictive capabilities of a series of metrics with regards to the quality of a join. Therefore, we considered it more appropriate to first develop a model with all the initial metrics, measure the utility of the approach, take its performance as a baseline, and execute a feature pruning afterwards. This will prevent losing metrics that, although correlated or almost irrelevant, might still be useful for the task at hand and provide useful insight on the *sns* detection problem.

4.2 Defining the process

4.2.1 Structure of the experiments

The ground truth used in these experiments contains joins of the four categories described previously. As a result, we can use the models to predict not only *sns* joins but also the other three categories, always keeping the *sns* isolation as the priority. As a result, we will mainly work on a multilabel model to assess the viability of this approach. However, using only the multilabel model could make it difficult to understand some of the prediction patterns that are developed, as we can not analyze them further enough. In order to fulfill this role, three complementary models will be developed along with the first, each comparing *sns* joins with one of the other three typologies. The complete list of models is the following:

- *Multilabel*: the target variable will contain the four types of joins.
- *sns* vs *semantic*: binary classification model for *sns* and *semantic* joins.
- *sns* vs *syntactic*: binary classification model for *sns* and *syntactic* joins.
- *sns* vs *non-joinable*: binary classification model for *sns* and *non-joinable* joins.

Developing these separate models will bring nuance to the detection process. By generating alternate models that focus on the binary comparisons we can distinguish details that can make certain reasonings easier to conduct, which might be lost in the multilabel model. These alternative models will be analyzed at certain parts of the development to highlight additional conclusions.

The implementation of these models will use different estimators and techniques, each of these presented and justified accordingly. Our initial goal will be to find the models with the best predictive capabilities, so they will be compared through a series of evaluation metrics that define their behavior. Given how integral these criteria are for the development of the experiments, they will be discussed separately in Section 4.2.2. Additionally, the evaluation process will encompass cross-validation tasks to prevent the development of biases in the data and guarantee the generalization of the model as much as possible.

An important point to consider is regarding the splits used in the cross validation tasks. In most projects dividing the data into 5 or 10 folds suffices to ensure the generalization capabilities of the model. However, we are dealing with an unbalanced dataset, as *sns* joins are considerably less in number when compared to the other types. This implies that randomly splitting the data might generate unbalanced folds, some of them containing very few instances of the less represented class in the training data. This might considerably hinder the capacity of the model for predicting this class. Undersampling and/or oversampling techniques can be used to correct this unbalance of the data. However, introducing these techniques produces, respectively, loss of information and generation of noise. Therefore, and given that in our particular case the proportions are not insurmountably out of balance, it is preferable to use a less intrusive technique. More precisely, we decided to use a *stratified shuffle split* generation method.

This splitting methodology randomly divides the data into a given number of folds whilst maintaining the proportion of labels in each. That is, there will be no folds with no instances of the less represented class. With this, we can palliate the unbalanced data issue without actually altering the data. Moreover, with this technique we would be generating folds that more accurately represent the real scenarios, as it is likely to find a similar distribution of labels when generalizing the model. It is important to note that the 10-fold stratified split (with a fixed random state) will be used as the cross-validation methodology for evaluating and comparing *all* models throughout this development.

4.2.2 Evaluation metrics

The usage of the correct metrics to assess the capabilities of the model is a crucial point in any Machine Learning process, as depending on the purpose of the systems to develop some metrics might be more suitable than others. In this experimentation several metrics were employed, although only one of these was taken as the deciding factor on whether a model was superior than others. The mathematical representation of these metrics is defined in figure 4.

$$Accuracy = \frac{TP + TN}{FP + FN + TP + TN} \qquad Precision = \frac{TP}{FP + TP}$$

$$Recall = \frac{TP}{FN + TP} \qquad F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

Where TP = true positives, TN = true negatives, FP = false positives, FN = false negatives

Figure 4: Mathematical definition of the metrics used

Firstly, we have the **accuracy** of the model, which is the most simple evaluation metric, given that it just represents the rate of correct predictions. That is, the amount of true positives/negatives divided by the total amount of predictions for all classes. Generally, the higher the accuracy, the better the predictor, but relevant information might be missing behind this simplicity. In unbalanced datasets, the predictors might develop a bias towards the more represented labels, achieving good accuracy due to correctly classifying the most represented label, but being unable to predict the other labels. This resembles our scenario, in which the *sns* joins are the least populated class. In order to account for this, we will also take the F1-score into consideration.

The F1-score leverages precision and recall to get a more robust estimation of how good a predictor is. Precision indicates the proportion of correct predictions (true positives) among all predictions for a given class (true and false positives). That is, how many of the instances predicted as class A do belong to class A. Recall measures the rate of correct predictions (true positives) among all true examples (false negatives and true positives). That is, how many instances of A were classified as belonging to A. By combining the two measures, the F1-score presents a more nuanced metric than accuracy. However, precision and recall can only be calculated for binary classifiers, so for multilabel classifiers (such

as our case, in which we have four labels), we need to compute the ***macro* F1-score** (i.e. the unweighted mean of the F1-scores for all labels).

The *macro* F1-score is a more robust indicator of the overall quality of the predictor. However, it is important to remember that the focus of the project is in detecting *sns* relationships. So even though assessing whether we can predict the four different labels is an appealing complementary experiment, our priority is predicting *sns* joins. Therefore, we will also analyze the **recall, the precision and the F1-score for the *sns* label only**. Doing so will bring relevant insight related to how the predictors fare in the *sns*-detection problem.

More specifically, we will use the *sns* F1-score as the sole indicator to compare models. That is, if the *sns* F1-score of model A is higher than the F1-score of model B, then model A will be preferred over model B. The reasoning behind this is that the F1-score is the harmonic mean between the precision and recall, and, ideally, we desire for both of these metrics to be high. A high recall indicates that the model detects a lot of true *sns* joins, whereas a high precision indicates that most of what the model predicts to be *sns*, are *sns*, and not other types of joins. Focusing only one one of the two would imply either not being able to detect *sns* joins, or labeling a lot of *non-sns* joins are so, which is not useful.

4.3 Empirical testing

4.3.1 Selecting the estimator

There are a plethora of different estimators (for this experimentation, classifiers, as the target is binary) to choose from, each performing better than others depending on the scenario. Fully developing each predictor (cross-validation, parameter tuning and feature selection) to fairly compare their results is unfeasible, given the computational time required to thoroughly conduct these processes. Therefore, a more naive approach was selected: some of the most relevant classifiers would be fitted to the data in its "base" form (i.e. default parameters) and the best-performing predictors would be selected for further tuning. Doing so might leave out potentially good-performing classifiers that need its parameters tuned. Nonetheless, this was deemed as the better approach given the time constrictions and the prohibitive cost of thoroughly testing all classifiers.

This first estimator exploration was conducted in the following manner: 10 classifiers were selected and invoked with its default parameters, aside from a common value for its randomized state (for those that use it). These predictors would be evaluated on the stratified 10-fold cross validation presented in section 4.2.1, and used for the four typologies of models defined. However, only the results for the multilabel model will be compared:

Predictor	**F1-score (macro)**	**Accuracy**	**F1-score (*sns*)**	**Recall (*sns*)**	**Precision (*sns*)**
Random Forest	**77.03%**	**78.93%**	**70.37%**	71.96%	**68.95%**
Decision Tree	72.28%	73.45%	67.67%	76.27%	60.86%
Gradient Boosting	69.02%	71.79%	59.04%	57.02%	61.25%
K-Nearest Neighbors	60.49%	63.63%	47.50%	53.10%	43.01%
Multilayer Perceptron	61.88%	67.02%	42.09%	38.42%	48.39%
AdaBoost	61.88%	67.02%	36.27%	30.63%	45.01%
QDA	52.83%	57.15%	25.97%	**93.54%**	15.08%
Naive Bayes	32.93%	34.78%	15.00%	92.41%	8.17%
Logistic Regression	40.81%	53.00%	0.00%	0.00%	0.00%
SVM (RBF Kernel)	37.45%	50.17%	0.00%	0.00%	0.00%

Table 8: Results of implementing the *Multilabel* model with several classifiers

These results are the first empirical evidence of the validity of the initial hypothesis: distribution-comparison metrics are useful in the detection of *sns* joins. The Random Forest estimator scores a 70.37% in the *sns* F1-score, which is a considerably high result for such a naive model. This indicates that with further processing an even higher score could be achieved, which would reinforce the hypothesis even further. Nonetheless, Table 8 already reinforces the theorization presented in the first parts of the document, highlighting that comparing distributions is a useful technique to detect *sns* joins.

The *macro* F1-score and accuracy indicate that the Random Forest estimator is clearly superior, followed by Decision Trees and Gradient Boosting, although with a noticeable difference. This showcases that, if the goal was to select the model that can best classify the four types of joins, the Random Forest would be the best option. However, and as mentioned beforehand, we also need to take into account if the predictors are able to detect *sns* joins. Focusing on the F1-score, precision and recall for *sns* joins, we can observe some interesting differences with regards to the first conclusion. Firstly, the general trend is for the models to perform worse when compared to metrics for the four labels at once. What this means is that the predictors have trouble detecting *sns* joins as opposed to the other types of joins. This might be because of the data imbalance, which causes a bias towards more populated classes, despite the efforts to minimize its impact.

Surprisingly, the QDA and Naive Bayes have the highest recall of all the predictors. This means that they are able to correctly detect more than 90% of the *sns* joins. However, their precision are among the worst, which implies that a considerable number of *non-sns* joins are classified as *sns*. More precisely, only 15% and 8% of all the joins predicted to be *sns* are actually *sns*. We can conclude that the QDA and Naive Bayes are not good predictors, as the vast majority of *sns* joins that they predict are actually *not-sns*. Therefore, even if they detect a high percentage of true *sns*, these will need to be manually tracked among a vast number of *non-sns* joins, so there is little gain made. This is evidenced by their F1-score, which are the fourth and third lowest, respectively.

The other noticeable difference is in the top two predictors, which are substantially better than all others: Random Forests and Decision Trees. The latter has one of the highest recalls, only bested by QDA and Naive Bayes, whereas the former has the best precision. On the other hand, the recall of the Random Forest is relatively close to the recall of the Decision Trees (4.3% difference), whereas the precision of the Decision Tree is considerably lower than the precision of the Random Forest (8.1% difference). Following the argument presented before, this implies that Decision Trees detect more true *sns* joins, but Random Forests have less misclassified *sns* joins, with Random Forests presenting an overall more balanced behavior as evidenced by the higher F1-score.

It is clear that the best predictor is either the Random Forest or the Decision Tree. Selecting one or the other is, however, a more complex issue. As mentioned before, we are basing the comparison on the *sns* F1-score, so the Random Forest is clearly superior. Moreover, Random Forests have more parameters given that they are a more complex model, so there is more potential for fine tuning its behavior. As a result, **Random Forests will be the classifier used in the rest of the experimentation**. However, observing the differences in the recall and precision metrics opens up a more profound debate.

If our goal was uniquely to detect all *sns*, then Decision Trees are better, given that the recall is higher. However, we also want to keep misclassifications at a minimum, as we do not want *non-sns* joins labeled as so. Having true *sns* among a very high number of *non-sns* does not help in solving the *sns* detection problem. Therefore, even though Random Forests have a worse recall, the fact that they have a more balanced performance makes them more suitable for the task at hand, which is to separate *sns* joins from the rest. Nonetheless, in a different scenario we might not care that much about false positives, as we might intend to run a manual checking afterwards. Therefore, our priority would be to have as many true positives as possible. In these circumstances the model with the highest recall would be better, which would imply that the Decision Trees predictor is better suited for the task. Even though this is not the scenario presented in the project (in which we prioritize a more balanced performance), it is relevant to consider alternative methodologies for different circumstances.

4.3.2 An alternative prediction scheme - 3 binary classifiers

During the development of the analysis described in the previous section, an alternative proposal to the multilabel classification model was made: instead of using a single model to differentiate between the four different types of join (the *Multilabel* model), could the other three binary models be used in conjunction, each trying to separate *sns* joins from one other type and combine the predictions? More precisely, the prediction of this alternative scheme would be the result of the predictions of the three smaller, binary models (*sns vs semantic*, *sns vs syntactic* and *sns vs non-joinable*).

In order for this second approach to ensure correct *sns* detection, a join can only be labeled as an *sns* join if the predictions of the three different models are *sns*. For instance, if the predicted labels of the three models are [*semantic*, *sns*, *sns*] we assume that the predicted label for the join is not *sns*. On the other hand, if the predictions of the three models are [*sns*, *sns*, *sns*] we assume that the join is *sns*. The main logic behind this idea would be to not force a model to differentiate between four types of joins, which might cause the

patterns that separate each type from the rest to become more blurry. Instead, by doing binary comparisons, the patterns that separate the joins can, potentially, be more easily detected and lead to a better prediction.

The first step to ascertain if this scheme is appropriate is to study the performance of the binary models in the previous experimentation (of which the only showcased results were those of the *Multilabel* model). We will retain only two metrics (the general accuracy of the models and the *sns* F1-score) and present only the top five predictors.

Predictor	**Type of join and metric**					
	sns vs non-joinable		***sns vs syntactic***		***sns vs semantic***	
	F1-score (*sns*)	**Accuracy**	**F1-score (*sns*)**	**Accuracy**	**F1-score (*sns*)**	**Accuracy**
Random F.	**92.74%**	**97.50%**	**89.02%**	**96.33%**	**77.13%**	**88.62%**
Decision T.	86.24%	94.96%	85.67%	95.05%	75.84%	87.34%
Gradient B.	87.20%	95.74%	83.57%	94.65%	68.07%	86.06%
MLP	78.30%	92.66%	76.61%	91.89%	54.27%	81.89%
KNN	72.27%	90.36%	73.06%	91.16%	61.09%	81.02%

Table 9: Results of implementing the binary models with several classifiers

It is clear to see that the best performing predictor in the three cases is the Random Forest, so in order to build the three separate binary predictors, the best approach would be to employ a Random Forest in all of them. Table 9 also provides insight on how the metrics fare when used to separate between the types of joins. We can observe that the F1-score for the *sns* vs *non-joinable* and *sns* vs *syntactic* models are considerably high for the Random Forest (92.74% and 89.02%). This contrasts with the F1-score for the third model (77.13%), in which the predictive capabilities are substantially reduced. We can conclude that the set of metrics performs noticeably well when separating *sns* joins from *non-joinable* columns and *syntactic* joins, whilst having more trouble with *semantic* joins.

These results align with the analysis developed in section 3.1: distribution-comparison metrics are useful to, mainly, differentiate *sns* joins from *syntactic* and *non-joinable* columns (i.e. horizontal axis), and struggle more in the separation of *sns* joins and *semantic* joins (i.e. vertical axis). This is so because distribution-comparison metrics try to capture the semantic similarity but are not able to do the same for the syntactic similarity, hence the results. This reinforces the idea that these types of metrics and, as such, the overall comparison of distributions, are arguably a good indicator of the *semantic* similarity of two columns.

Once the individual analyses have been performed, we will build the 3-binary classifiers models. In order to do so a custom cross-validation is needed, which will take into account the three predictions. We will still use the *stratified shuffle split*, but each split will be used to train and test the three models. It is important to remember that each split has instances of the four kinds of joins. However, the predictors are binary, so we will only

fit a model to the subset of data which encompasses *sns* joins and one of the other three kinds of joins. Then, each model will be given the test set of the split to generate a series of predictions.

The goal is to calculate "global" metrics. For instance, in order to calculate the recall, we need to calculate the true positives and the false negatives. In our system, the true positives are those true *sns* joins that are correctly classified as *sns* by the three models. On the other hand, false negatives are those true *sns* joins that at least one of the models does not predict as *sns*. Essentially, we take into consideration the three models to assess the correctness of a prediction. The results are presented in the following table, alongside the scores for the best multilabel classifier:

Model	**F1-score (macro)**	**Accuracy**	**F1-score (*sns*)**	**Recall (*sns*)**	**Precision (*sns*)**
Multilabel R.F	**77.03%**	**78.93%**	**70.37%**	**71.96%**	68.95%
3-Binary models	-	-	70.15%	69.94%	**70.37%**

Table 10: Comparison of models (I)

As evidenced by the results, the models are very close in terms of performance. However, the combination of the binary classifiers seems slightly worse, given the values for the F1-scores. More precisely, we can observe that, although the 3-binary classifiers system has a higher precision, the difference in recall is enough for the multilabel model to present a modest advantage. Nonetheless, this difference is likely negligible, and does not indicate solid evidence to prefer one system over the other, as both would perform almost equally. As a result, selecting which system to continue the project with can be left to aspects other than the sheer performance. More precisely, the multilabel model has the added functionality of potentially classifying the four labels, which can allow us to study the patterns that differentiate the four types of joins. Additionally, working with one model instead of three facilitates the management of parameter-tuning and feature selection, while also being a more comprehensive approach. Therefore, the multilabel classifier will be the selected model.

4.3.3 Improving the model

In order to extract the best capabilities out of a model, it is crucial to develop a thorough fine-tuning process to find the best hyperparameters to use. To do so, we will employ the *grid search* methodology, which is a brute force approach that exhaustively tests different parameters combinations to find the best performing configuration.

The implementation of *sklearn* for the Random Forest predictor has 15 potential parameters to tune, which implies that it is unfeasible to thoroughly test all potential combinations. The average fitting time for the base version of the Random Forests takes slightly longer than 2 seconds. Assuming only 2 possible configurations for each parameter and 3-fold cross validation instead of 10-fold, it would take around 60 hours, without taking into account that changing some of these parameters can substantially increase the execution time. A less detailed approach was selected instead, with only the most relevant

parameters (those that govern the predictor's behavior more directly) being explored, with a very limited number of values and a 3-folds cross-validation instead of 10-fold. To compensate for this, a second grid search was conducted, with the goal of narrowing down these parameters even further, but with a starting point already defined.

After the grid search, it could be seen that most of the default values were already optimal, the most notorious difference being the parameter *max_features*, which indicates the amount of trees in the forest. This is, arguably, the most important parameter of the estimator, as the more trees in the predictor, the more nuance can be extracted from the data. However an overly large number of trees will generate a bias on the training set that impedes the generalization of the model. This last aspect is taken into consideration through the usage of cross validation to prevent overfitting. The optimal value for *max_features* was 450, contrary to the 100 default trees. Other relevant changes are:

- Shannon's entropy as the function to measure the quality of a split instead of the default Gini impurity.
- Not using bootstrap samples to generate the trees.
- Including a warm start, that is, reusing the solution of the previous call to fit and add more estimators to the ensemble, instead of just fitting a whole new forest.

Model	**F1-score (macro)**	**Accuracy**	**F1-score (*sns*)**	**Recall (*sns*)**	**Precision (*sns*)**
Multilabel R.F. (tuned)	**78.71%**	**80.20%**	**73.70%**	**77.72%**	70.18%
Multilabel R.F.	77.03%	78.93%	70.37%	71.96%	68.95%
3-Binary models	-	-	70.15%	69.94%	**70.37%**

Table 11: Comparison of models (II)

The improvement is noticeable, with almost a 3% increase in the F1-score. This is mainly due to a considerably better capacity for detecting true positives (recall), with close to a 6% increase. The precision also improves, although in a lesser degree. Overall, the parameter fine tuning was successful in generating a better model.

4.3.4 Chain model

The previously presented 3-binary model's implementation was quite naive, as we took the predictions of the three classifiers for every candidate join and determined that only if those three predictions were *sns*, the join could be defined as *sns*. Despite the simplicity of the approach, the final results signaled that this method was, in terms of performance, very close to that of the initial multilabel model. This opened the possibility of thinking that, potentially, systems that took several binary predictions and combined them in more complex ways could generate even better results. One of such systems is the *classifier chain* scheme.

The *classifier chain model* is a widely recommended approach for multilabel classification [23]. Essentially, instead of using a single predictor instance to classify several labels "at the same time", we will implement a chain of these predictors, each of them applying a "one-versus-the-rest" methodology, in which the multi-label problem will become a series of binary problems. Each member of the chain will define all the data points of one of the labels as the positive class and combine all the other data points of the rest of the labels as the negative class. The predictions that this model obtains will be *passed to the next classifiers* in the chain, which will take them into account to generate its own predictions. These additional features allow the chain to exploit correlations among the classes.

For instance, our first classifier in the chain will differentiate between *sns* joins (positive class) and the rest of joins (*semantic*, *syntactic* and *non-joinable* as the negative class). The model will generate a set of predictions, differentiating between *sns* joins and the rest. Then, the predictions will be passed to the second classifier, which will differentiate between *semantic* joins (positive class) and the rest of joins (*sns*, *syntactic* and *non-joinable* as the negative class). This process will repeat itself for the other two classifiers in the chain, and a final set of predictions will be obtained.

The main attractiveness of this system is that it separates the multilabel classification problem into smaller pieces that are easier to deal with, combining the output to generate a, potentially, better decision. What each classifier is trying to do is understand the patterns that separate the corresponding individual class from the rest, propagating this "knowledge" to the other classifiers and the final set of predictions. The parallelism with the 3-binary models approach used earlier is clear, but with the added benefit of taking into account the predictions of previous models for the latter models.

However, the fact that the classifiers are arranged in a chain implies that the order in which the differentiations are made plays a considerable role in the results. It is unlikely to obtain the same results from a chain that starts classifying *sns* joins against the rest and having the last predictor classify *semantic* joins versus the rest, than from a chain with the opposite order. Therefore, to thoroughly explore this technique, all possible order permutations needed to be taken into account, with one of them being the optimal configuration. To maintain consistency with the previous model assessment methods, we will apply the same stratified 10-fold cross validation to each.

A final question that needs to be answered is which predictor is going to be used for each classifier in the chain. In the previous experiments we have observed that the predictor that performs the best in the binary classifications models is the Random Forest, for the three types of comparisons. Therefore, it makes sense to use it as the base classifier. As for the parameters of the models, given that we can not perform a grid search for every classifier of every chain, we will reuse the best parameters obtained in the previous section. The following table represents the best performing chains for each of the metrics:

F1-score (macro)	**Accuracy**	**F1-score (*sns*)**	**Recall (*sns*)**	**Precision (*sns*)**
[1,0,3,2] 78.33%	[1,0,3,2] 79.66%	[1,0,3,2] 74.13%	[0,2,1,3] 80.51%	[3,2,1,0] 72.34%
[0,1,3,2] 78.25%	[1,0,2,3] 79.63%	[0,1,3,2] 73.95%	[0,1,2,3] 80.06%	[3,2,0,1] 72.34%
[1,0,2,3] 78.24%	[0,1,3,2] 79.59%	[0,2,3,1] 73.82%	[1,0,2,3] 80.00%	[3,1,2,0] 72.34%

0: *semantic vs the rest*, 1: *syntactic vs the rest*, 2: *non-joinable vs the rest*, 3: *sns vs the rest*

Table 12: Chain models comparison

The metric that we have been using throughout this experiment to compare models has been the F1-score for the *sns* label. If we apply the same logic, we have a chain that clearly performs better than the rest ([1,0,3,2]). Moreover, the F1-score obtained is higher than the F1-score for the tuned multilabel Random Forest, so it seems like the better option. As for the rest of the metrics, they can give useful insight over how the ordering of the chains conditions the performance.

If we focus on the precision results, we can observe that the three chains with the highest value differentiate *sns* joins (i.e. *3* in the first position) in the first classifier. In doing so, they reduce the amount of false positives, as the other classifiers of the chain are "aware" of the patterns of an *sns* joins and reduce the misclassifications of the other labels. On the other hand, these chains score the lowest in the recall metric (less than 71.50% for the three of them, a 9% below the best recall value), as the number of false negatives increases due to the fact that it is the first prediction in the chain, so no complementary information is provided. The opposite pattern can be seen in the recall metric, where the top three chains leave the prediction of *sns* for last. This allows them to incorporate the predictions of the other classifiers of the chains as features, improving the predictive capabilities. This implies, however, that the number of false positives increases, as the previous predictors misclassify some of the other labels as they do not have these added features. As a result, they have some of the lowest scores for precision (69.98% for [1,0,3,2], 68.25% for [1,0,2,3] and 67.30% for [0,2,3,1]).

If we go back to the *sns* F1-score, we can observe that the top scoring chains have the *sns* prediction neither at the end nor the beginning of the chain, but rather in the third spot. This seems to indicate that this configuration of the chain is the most balanced, as it is able to best compromise the detection of a high amount of true *sns* joins whilst preventing too many misclassifications. To better compare the results, the following table presents the scores of all the models developed so far.

Model	**F1-score (macro)**	**Accuracy**	**F1-score (*sns*)**	**Recall (*sns*)**	**Precision (*sns*)**
Chain model	**78.33%**	79.66%	**74.13%**	**79.05%**	69.98%
Multilabel R.F. (tuned)	78.71%	**80.20%**	73.70%	77.72%	70.18%
Multilabel R.F.	77.03%	78.93%	70.37%	71.96%	68.95%
3-Binary models	-	-	70.15%	69.94%	**70.37%**

Table 13: Comparison of models (III)

Comparing the chain model to the rest, we can observe that it is the best performing in terms of *sns* detection, although the 3-binary Random Forests still present a higher precision value. On the other hand, the tuned multilabel Random Forest does perform better when assessing the predictions for the four types of joins, which indicates that it would be better suited if we were interested in correctly classifying all four labels. As our deciding metric is the *sns* F1-score, the chain model with the specified configuration is deemed as better than the tuned multilabel.

4.3.5 Feature selection

At this point in the project, it is clear to see that the selected set of metrics is able to solve the *sns* detection problem, validating the initial hypothesis. Now that the best performing model has been found and we have a baseline accuracy to compare to, we can initiate the feature selection process to prune the less relevant metrics, which will reduce the computation time and, potentially, improve the predictions.

A first idea to execute the feature selection process could be to remove metrics based on the significance of the tests presented in section 3.3. It is tempting to simply remove those features that present significant differences only among *sns* and one other type of join, or to do so based on the p-value (keep the metrics that have the lower p-value for the tests). However, these approaches are unadvised. A metric that differentiates only among *sns* and one other type of join might be relevant if it does so in a way that no other metrics can do, so removing it would imply the loss of important information. The Machine Learning field presents more complex systems to properly prune the set of metrics. A first technique that we have at our disposal can be found in the Random Forest estimator itself, as it has a built-in relevance assignment functionality. With it, we can take a first look at which variables are most important to differentiate the four types of join:

Metric	Relevance
Entropy difference	6.86%
Variation difference	5.72%
Ansari test	5.06%
Maximum value difference	4.93%
Kurtosis difference	4.45%

(a) Most relevant features

Metric	Relevance
Anderson-Darling test	0.28%
Bartlett test	0.80%
Mann-Whitney test	0.86%
Kruskal difference	0.89%
Kolmogorov-Smirnov difference	0.92%

(b) Less relevant features

Table 14: Most and less relevant features of the chain model

It might seem surprising to observe that four out of the five most important variables are descriptive statistics and all of the five less relevant variables are statistical tests. It would seem that statistical tests, given their mathematical-based reasoning, should better highlight the differences in the distributions and, following our hypothesis, better separate the different types of joins. However, and drawing an analysis beyond table 14, the 11 less relevant features are all statistical tests, all with a relevance below 1.7%. The only ones overcoming this threshold are the Cramer-Von Mises test (2.5%) and the Ansari test (5.06%).

This brings us a first conclusion, which is that descriptive statistics of the distributions (and their differences) seem to better isolate *sns* joins than specific statistical tests. A preliminary explanation for this finding could be that statistical tests are too restrictive, and it is more beneficial to get an approximate estimation of the resemblance of the distributions through descriptive statistics, which are more lenient in the comparisons. This previous idea is supported by the best performing of the statistical tests: the Ansari test, which compares the scale parameters of two distributions, rather than comparing more common factors such as means, variances or even the distributions themselves. We can draw a final conclusion from focusing on the best performing descriptive statistics metrics. Out of the four shown in table 14, three of them (entropy, variation and kurtosis) describe more "advanced" concepts of the distribution of values, at least in comparison to more standard measures such as the mean or the median. This might indicate that this "kind" of statistics provide the best balance in terms of complexity, presenting enough details to detect *sns* but without being as restrictive as statistical tests.

Despite the differences in the relevance of the variables being quite significant, it is unadvised to remove all those variables that are below an arbitrary threshold, as, although their contribution might be small, the results may considerably diminish. Therefore, a more nuanced technique should be used to detect which features are irrelevant. There are a variety of different processes to perform an automatic feature selection, but the one used in this project is the RFECV (Recursive Feature Elimination with Cross Validation). This technique fits a model with all the variables, gets the importance of each of them and removes the less relevant feature. Then, it repeats the steps until only one feature remains.

Alongside this process, it keeps track of the performance of the model, detecting at which points the indicated evaluation metric (in our case, the *sns* F1-score) decreases. In order to prevent biases, the RFECV executes this process in a cross validation fashion, which, as we have done throughout this project, will consist of a 10-fold stratified split. Moreover, we will use the chain model as the base classifier to draw the relevance from, given that it is the best model as of now. The results of the RFECV indicate that only 10 of the initial 35 features are truly relevant for the detection of *sns* joins. These are:

Metric	Relevance	Metric	Relevance
Entropy difference	6.86%	Skewness difference	4.35%
Variation difference	5.72%	Wasserstein distance	4.13%
Ansari Test	5.06%	Mean difference	4.09%
Maximum value difference	4.93%	CI standard deviation difference	3.93%
Kurtosis difference	4.45%	CI mean difference	3.69%

Table 15: Metric selected by the feature selection process

It is remarkable that none of the selected features is a statistical test. This is congruent with the previous conclusions, in which the descriptive statistics were found far more relevant than the statistical tests. It is worth noting that not even the Ansari test, the third most relevant feature in the isolated analysis has been selected. On the other hand, the other four most relevant features presented before, are picked in the final selection. As for the remaining metrics, the mean and the median have been preferred to the mode

and the other types of means and medians, the maximum value was selected but not the minimum, and most of the confidence-interval-related metrics have been removed. Additionally, the standard deviation, the skewness and the IQR have also been left out, whereas the Wasserstein's distance (the closest approximation to a statistical test) was deemed relevant enough.

The most likely explanation for the removal of these descriptive statistics is the high correlation among most of the variables, which made them redundant and likely to cause some bias. This is, with a high probability, the case among most of the confidence-interval-related metrics, the different types of means and even the maximum and minimum values. However, knowing a priori which of the correlated metrics to use might be difficult, as we also need to factor in how these metrics relate to the rest of the features. Through the RFECV process we guarantee that the features selected are the best possible. In order to test this last statement, we can execute a final set of models and observe the results:

Model	**F1-score (macro)**	**Accuracy**	**F1-score (*sns*)**	**Recall (*sns*)**	**Precision (*sns*)**
Chain model (F.S.)	76.93%	77.66%	**74.54%**	**81.33%**	69.93%
Chain model	**78.33%**	79.66%	74.13%	79.05%	69.98%
Multilabel R.F. (tuned)	78.71%	**80.20%**	73.70%	77.72%	70.18%
Multilabel R.F.	77.03%	78.93%	70.37%	71.96%	68.95%
3-Binary models	-	-	70.15%	69.94%	**70.37%**

Table 16: Comparison of models (IV)

The new chain model performs better than the same model without feature selection. It is important to note that it does so in the *sns* F1-score metric, but performs worse in the global score. This is to be expected, as focusing on *sns* metrics implies that the rest of the labels might suffer more misclassifications. However, given that our goal is to detect *sns*, this is a non-relevant issue.

Combining the two approaches

5.1 Defining the new model

In the previous sections we have explored how the intersection of sets (i.e. containment) and the comparison of distributions are two aspects that do contribute to the *sns* detection problem, playing different but complementary roles. The logical continuation of this analysis involves combining the two measures into a single model, merging the capabilities of both methods into a single system. In the experimentation of section 3.1 the true containment value was used to assess its viability. However, it is our intent to avoid calculating it directly for every candidate join due to its high computational requirements, which would make such an operation infeasible in large-scale data scenarios. Therefore, it is imperative to approximate or predict the containment using an alternative system.

In [7] they use a profile-based system with a selection of metrics that follows a similar philosophy to the one developed in this project. However, its focus was to use the approximation of the containment of values as the notion of similarity, not the detection of *sns* joins. The results highlighted in [7] indicate that the set of metrics employed does generate a good approximation for the containment of the candidate joins, which implies that the metrics are useful in the assessment of the syntactic similarity. This is precisely the problem that needs to be addressed.

A combination of the two sets of metrics (the one presented in this project to assess semantic similarity and that of [7] to assess syntactic similarity) would define the concept of similarity presented in the introduction of this work. Before developing a model to prove so, a preliminary experimentation was conducted, in which the metrics of [7] were employed as the predictors of the previously defined models. This would give insight on whether an approximation of the containment could suffice for the *sns* detection task. The set of metrics used in [7] do contain some statistical measurements similar to those used in this project. In order to achieve a purely syntactic similarity search, these metrics will be removed. The final set of metrics used will be labeled as *syntactic* metrics.

In order not to repeat all the previous development for a preliminary test, we will directly apply a Random Forest predictor to generate the predictions, with no tuning nor feature selection. The results are as follows:

Model	F1-score (macro)	Accuracy	F1-score (*sns*)	Recall (*sns*)	Precision (*sns*)
Chain model (F.S.)	76.93%	77.66%	**74.54%**	**81.33%**	69.93%
Chain model	78.33%	79.66%	74.13%	79.05%	69.98%
Multilabel R.F. (*syntactic* metrics)	**81.61%**	**83.98%**	73.95%	68.50%	**81.44%**
Multilabel R.F. (tuned)	78.71%	80.20%	73.70%	77.72%	70.18%
Multilabel R.F.	77.03%	78.93%	70.37%	71.96%	68.95%
3-Binary models	-	-	70.15%	69.94%	70.37%

Table 17: Comparison of models (V)

We can observe that a default Random Forest with the new set of metrics performs just marginally worse than the best model obtained so far. This is remarkable given the extensive process to obtain the previous models, with thorough parameter tuning as well as feature selection tasks. These results are likely due to the fact that there are more *semantic* joins than *non-joinable* columns, and the containment (and, therefore, the metrics that approximate its value) is better at separating *sns* joins from the former rather than the latter, so it has an inherent advantage.

It is also worth noting the substantial difference between the *sns* precision and recall metrics of the *syntactic* metrics model in comparison with the rest. The best model so far has almost a 13% advantage in the recall metric, a distance that is observed in the opposite fashion for the precision metric. This indicates that the chain model has a higher rate of non-*sns* misclassifications, whereas the new model is more accurate in this regard. On the other hand, the *syntactic* metrics model presents a higher percentage of false negatives. By combining the two types of metrics (distribution comparison and syntactic metrics), we aim at preserving both of these high scores. We can complement the previous analysis observing the binary classification models results:

Predictor	Type of join and metric					
	sns vs non-joinable		***sns vs syntactic***		***sns vs semantic***	
	F1-score (*sns*)	**Accuracy**	**F1-score (*sns*)**	**Accuracy**	**F1-score (*sns*)**	**Accuracy**
Distribution-comparison metrics	**92.41%**	**97.39%**	**88.79%**	**96.25%**	76.28%	88.21%
Syntactic metrics	72.08%	94.12%	87.04%	96.06%	**89.52%**	**95.04%**

Table 18: Comparison of binary models (I)

We can clearly see the opposite trend in the two groups of metrics: the distribution-comparison metrics perform the best in separating *sns* and *non-joinable* columns and the worst in the *sns* and semantic comparison. On the other hand, the syntactic metrics

perform the best in the *sns* and *semantic* join differentiation, and the worst when dealing with *sns* and *non-joinable* columns. Additionally, we can observe how the model that classifies *sns* and *syntactic* joins presents similarly high results regardless of the set of metrics used. Once again, these results are coherent with the behavior defined in figure 1. After isolating only the *syntactic* metrics from [7], we will now combine all the distribution-comparison metrics defined in this project with all the metrics from [7], which included some statistical measurements not used previously. The results are as follows:

Model	**F1-score (macro)**	**Accuracy**	**F1-score (*sns*)**	**Recall (*sns*)**	**Precision (*sns*)**
Multilabel R.F. (*all* metrics)	**90.28%**	**91.80%**	**84.48%**	78.94%	**90.94%**
Chain model (F.S.)	76.93%	77.66%	74.54%	**81.33%**	69.93%
Chain model	78.33%	79.66%	74.13%	79.05%	69.98%
Multilabel R.F. (*syntactic* metrics)	81.61%	83.98%	73.95%	68.50%	81.44%
Multilabel R.F. (tuned)	78.71%	80.20%	73.70%	77.72%	70.18%
Multilabel R.F.	77.03%	78.93%	70.37%	71.96%	68.95%
3-Binary models	-	-	70.15%	69.94%	70.37%

Table 19: Comparison of models (VI)

This is the first model to combine both distribution-comparison and set-related metrics (i.e. *syntactic*), and the results align with the theorization proposed in the first half of this work. Without any additional tuning or feature selection, the model presents an increase of almost 10% in the *sns* F1-score, with regards to the two models analyzed previously. This seems to confirm that combining both types of metrics does provide the best environment for *sns* join detection, given the reasons previously presented. It is worth noting that the main improvement comes from the increase in the *sns* precision score. Combining the two types of metrics has resulted in the rate of false positives plummeting, whilst keeping a good recall value. Once again, we can complement the analysis with the binary models:

Predictor	**Type of join and metric**					
	sns vs non-joinable		***sns vs syntactic***		***sns vs semantic***	
	F1-score (*sns*)	**Accuracy**	**F1-score (*sns*)**	**Accuracy**	**F1-score (*sns*)**	**Accuracy**
All metrics	90.03%	96.73%	**93.43%**	**97.89%**	**93.87%**	**97.16%**
Distribution-comparison metrics	**92.41%**	**97.39%**	88.79%	96.25%	76.28%	88.21%
Syntactic metrics	72.08%	94.12%	87.04%	96.06%	89.52%	95.04%

Table 20: Comparison of binary models (II)

Combining the two types of metrics (distribution-comparison and *syntactic*) makes the models better overall. The only exception is with the *sns* and *non-joinable* comparison, which performs worse than the model with only distribution-comparison metrics. This could be because the union of the two groups includes too many statistical measures, which creates some contradicting patterns that hinders the prediction capabilities. This can be fixed through a process of feature selection, which will be included in the following section.

5.2 Improving the new model

Once the improvement of combining the two types of metrics has been tested, we can truly obtain the best model by repeating the process presented in section 4.3 with the new set of metrics. This includes the selection of the best predictor, a grid search to find the best parameters and feature selection to get rid of all the unhelpful features. Additionally, both the 3-binary model scheme as well as the chain model approach will be tested to ensure the best possible results. Given that the entire processing pipeline has already been explained, only the main results and conclusions will be presented here.

The only noticeable difference is that the feature selection will be conducted as the first step, as it is done in the majority of *sns* detection task. This analysis will be achieved by the development of Machine Learning projects. This is so because we have already tested that distribution-comparison metrics are useful in the *sns* detection problem, so our goal now is to find the best possible model. In order to achieve this, the best approach is to reduce the amount of features from the start, preventing redundant variables from biasing the models. After the feature selection, the amount of metrics shrink from 119 to merely 19 features, which are presented in the following table. Alongside them, it is indicated the percentage of importance during the feature selection process (which included all 119 metrics) as well as the original set of metrics they belong to.

Metric	Relevance	Metric	Relevance
Column names distance [1]	5.51%	Maximum value difference	1.46%
Frequent words (soundex) [2]	4.43%	CI mean difference	1.43%
First elements distance [3]	4.26%	Wasserstein distance	1.36%
Last elements distance [4]	3.93%	5th octile	1.36%
Frequent words [5]	3.65%	CI std. difference	1.35%
K [6]	2.24%	7th octile	1.33%
Cardinality of column 1	2.18%	6th octile	1.30%
Cardinality of column 2	2.09%	CI maximum value difference	1.25%
Entropy difference	2.07%	Percentage of Username [7]	1.24%
Variation difference	1.68%	-	-

[1] Levenshtein distance of the names of the two columns
[2] Containment of the 10 most frequent elements coded in Soundex format
[3] Levenshtein distance of the first element in both columns (ordered alphabetically)
[4] Levenshtein distance of the last element in both columns (ordered alphabetically)
[5] Containment of the 10 most frequent elements
[6] Cardinality proportion (i.e. cardinality of column 1 divided by cardinality of column 2)
[7] Percentage of instance that fit the username regex.
Syntactic metrics, distribution-comparison metrics

Table 21: Metric selected by the feature selection process over all possible metrics

Some notes about this selection:

- From the set of 10 metrics obtained at the end of the first model generation, only 7 remain (the differences of means, medians and kurtosis values were removed), which implies that 12 of the most relevant features come from [7]
- The most relevant feature by quite a margin is the distance of the names of the columns. This is sensible, as similar names do likely indicate a similar semantic concept. However, it might be a feature that is not always available, as large-scale data repositories often allow for unstructured data to be kept. If such a scenario becomes common, a new model without this particular metric will need to be developed.
- Besides the distance of the names of the columns, the other four most important features are directly linked with the containment of values, which is a testament to how important the inclusion of metrics to approximate it has been to further improve the model.
- The inclusion of K (the cardinality proportion) as well as both of the raw cardinalities of the tables is somewhat counter intuitive, as they seem to present the same type of information to the model. Its appearance might indicate that the model is generating some kind of bias towards these variables, which might not translate well when applied to external data. More tests should be conducted to confirm this. The same can be said for the percentage of instances that are a "username", as it seems a very arbitrary feature that should not be that relevant.

- From all the octiles, only the fifth, sixth and seventh octiles were chosen. This might indicate that the differences at the "end" of a distribution (i.e. the probabilities of the most common values) are more revealing than the differences at the "start" of a distribution (i.e. the probabilities of the less common values).

With this reduced set of features we repeated the entire pipeline described in chapter 4, of which the first step was obtaining the best base predictor. Once again, the best performing classifier was the Random Forest, so we simply had to execute the rest of the processes with the updated metrics, obtaining the following results:

Model	**F1-score (macro)**	**Accuracy**	**F1-score (*sns*)**	**Recall (*sns*)**	**Precision (*sns*)**
Chain model (*all* metrics, F.S., tuned)	**90.39%**	91.03%	**88.08%**	**86.65%**	90.04%
Multilabel R.F. (*all* metrics)	90.28%	**91.80%**	84.48%	78.94%	**90.94%**
Chain model (F.S.)	76.93%	77.66%	74.54%	81.33%	69.93%
Chain model	78.33%	79.66%	74.13%	79.05%	69.98%
Multilabel R.F. (*syntactic* metrics)	81.61%	83.98%	73.95%	68.50%	81.44%
Multilabel R.F. (tuned)	78.71%	80.20%	73.70%	77.72%	70.18%
Multilabel R.F.	77.03%	78.93%	70.37%	71.96%	68.95%
3-Binary models	-	-	70.15%	69.94%	70.37%

Table 22: Comparison of models (VII)

The best performing model follows, once again, the chain approach presented previously. In this case, however, it is substantially more balanced, as it scores the highest in the *sns* recall whilst being very close to the highest value of the *sns* precision. A *sns* F1-score of over 88% is considerably high, especially taking into account the stratified 10-fold strategy used in the cross validation. This guarantees that the model is able to detect the vast majority of *sns* joins while also misclassifying very few non-*sns* joins, which is the desired behavior.

Making the system scalable

Up to this point, this work has only been concerned with obtaining the model with the highest *sns* detection rate, which was represented by the comparison of the *sns* F1-scores. However, one of the goals of this project is to obtain a system that is able to perform the *sns* detection task, but that can also do so in a efficient manner for large quantities of data. This section will explore which are the required changes to make the system scalable and, additionally, to improve its speed.

6.1 Profile-based approach

In section 3.2 we concluded that using both tests and descriptive statistics to compare distributions was the desired compromise between correctly assessing the closeness of the sets of data while allowing some leniency to develop a more generalized approach. However, these two categories of metrics have different computational requirements to obtain them:

- Descriptive statistics are direct metadata obtained from the values. This set of metrics closely resembles the definition of a profile presented in section 2.1: a group of metrics that describe a set of data. As specified at the beginning of the document, the main advantage of this approach is that, from a computational complexity perspective, this method is much more efficient than other proposals. This is due to the fact that it is working with summaries of data as opposed to the data itself, which reduces the amount of comparisons required. Moreover, the profiles can be obtained prior to the data discovery process, precomputing and storing their values to be later used.

- Oppositely, statistical tests hardly fit the definition of a profile metric, as they need to be calculated for every candidate join. That is, for every pair of columns that we want to assess the similarity of, all tests need to be computed, as their results depend on how the two sets of values relate to each other. This poses an issue to the scalability and efficiency of the system, as statistical tests can not be precomputed and its overall calculation costs scale with the amount of comparisons to make, contrary to descriptive statistics that require to be computed only once per column.

To illustrate the relevance of not needing to execute specific computations for every candidate join, the time to generate both groups of metrics for the joins in the ground truth (over 7.400 pairings) was extracted. The results are presented next:

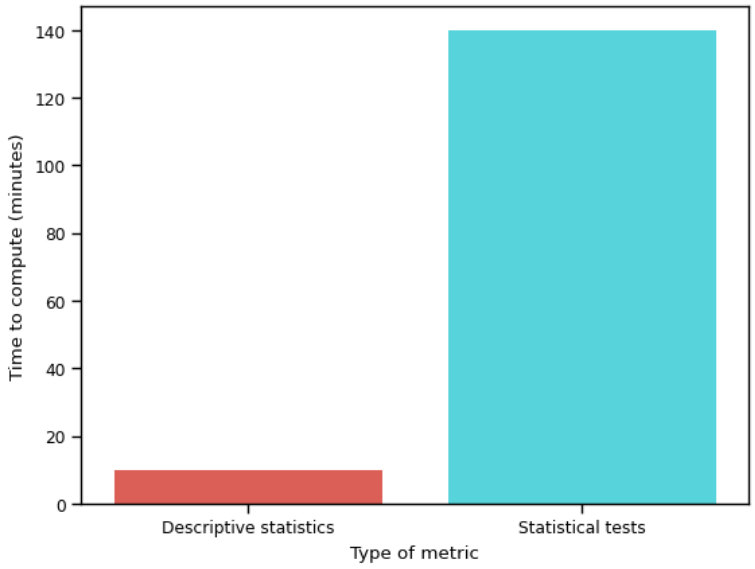


Figure 5: Time to compute the two typologies of metrics

The initial set of descriptive statistics required slightly less than 10 minutes to be generated, whereas the statistical tests required over 140 minutes to complete its calculations. This difference in the execution time signals that in order to make the system scalable it is imperative to remove the computation of statistical tests and, therefore, adopt a profile-based approach. More precisely, the profiles (i.e. the set of descriptive statistics) will be ingested by the models in order to make predictions over the type of join. By adopting this approach, the system will benefit from all the advantages that profiles bring and that have been extensively evaluated by the data discovery community.

6.2 Removing statistical tests

The employment of statistical tests served the purpose of providing an extensively studied form of comparing two sets of values and, given that these set of values were the probabilities of the columns, two distributions. However, these tests pose an issue to the scalability of the system and, as such, they will be removed from the pool of metrics. We can do so with minimal impact over the models because these tests were found not relevant for the task, as evidenced by the final set of selected metrics not containing any tests. It is important to note that its exclusion from the models does not imply that the can not differentiate *sns* from the other types of joins, only that the descriptive statistics do a better job in this same regard and thus should be prioritized.

The only exception to this is the Wasserstein distance, which was found relevant in the feature selection process of section 5.2. Despite the fact that it has a relatively low contribution, removing it might considerably affect the predictions of the models. Nonetheless, and given the reasons stated, its removal is necessary to produce a scalable system. We can measure the severity of its exclusion by developing a new model:

Model	F1-score (macro)	Accuracy	F1-score (*sns*)	Recall (*sns*)	Precision (*sns*)
Chain model (*all* metrics, F.S., tuned)	**90.39%**	91.03%	**88.08%**	**86.65%**	90.04%
Chain model (no Wasserstein distance)	90.30%	90.90%	88.01%	86.39%	89.77%
Multilabel R.F. (*all* metrics)	90.28%	**91.80%**	84.48%	78.94%	**90.94%**
Chain model (F.S.)	76.93%	77.66%	74.54%	81.33%	69.93%
Chain model	78.33%	79.66%	74.13%	79.05%	69.98%
Multilabel R.F. (*syntactic* metrics)	81.61%	83.98%	73.95%	68.50%	81.44%
Multilabel R.F. (tuned)	78.71%	80.20%	73.70%	77.72%	70.18%
Multilabel R.F.	77.03%	78.93%	70.37%	71.96%	68.95%
3-Binary models	-	-	70.15%	69.94%	70.37%

Table 23: Comparison of models (VIII)

The model worsens only marginally, which implies that removing the Wasserstein distance does not pose a big concern. Moreover, such a small decrease in the performance does not require the development of a complimentary method to approximate its value.

6.3 The final system

Among the final set of selected metrics there is another that requires a special explanation, the distance between the column names. If two columns have similar names, they are likely to represent the same conceptual idea, which is supported by this metric being the single-most relevant for the assessment of *sns* joins. As a result, removing it would considerably impact the accuracy of the models. In order to keep this metrics there are two factors to consider:

- Assuming that the names of the columns are available. In small-scale scenarios data tends to be accurately defined, making this assumptions a non-issue. However, large repositories of data might hold schemaless datasets that do not contain the names of the columns. In order not to develop an entirely new model without this feature, the final model will *assume* that the data it will encounter has the names of the columns present.

- The need for a computation for every candidate join. The distance of the names needs to be obtained for every candidate join. However, and differently from the statistical tests, this computation has a cost that does not scale with the amount of instances. This is because to calculate this distance it is only required to have the names of the columns, which are two strings of small size. In practice, this operation is not different from the distances that need to be computed from the other metrics, so despite requiring a specific computation for each join, its cost does not present an inconvenience.

Given these considerations, the distance of the names will not be removed from the pool of metrics. This implies that only with the removal of the Wasserstein distance and the explanations given, the model can be considered to be scalable, as adding more data will not make the system exponentially slower. Nonetheless, to fully confirm this assumption, the system should be tested for a large-scale repository and measure whether scalability is achieved. Given the difficulty of developing such a scenario and the time constraints, it was not possible to perform this experimentation. A diagram of how such a system would operate is defined next:

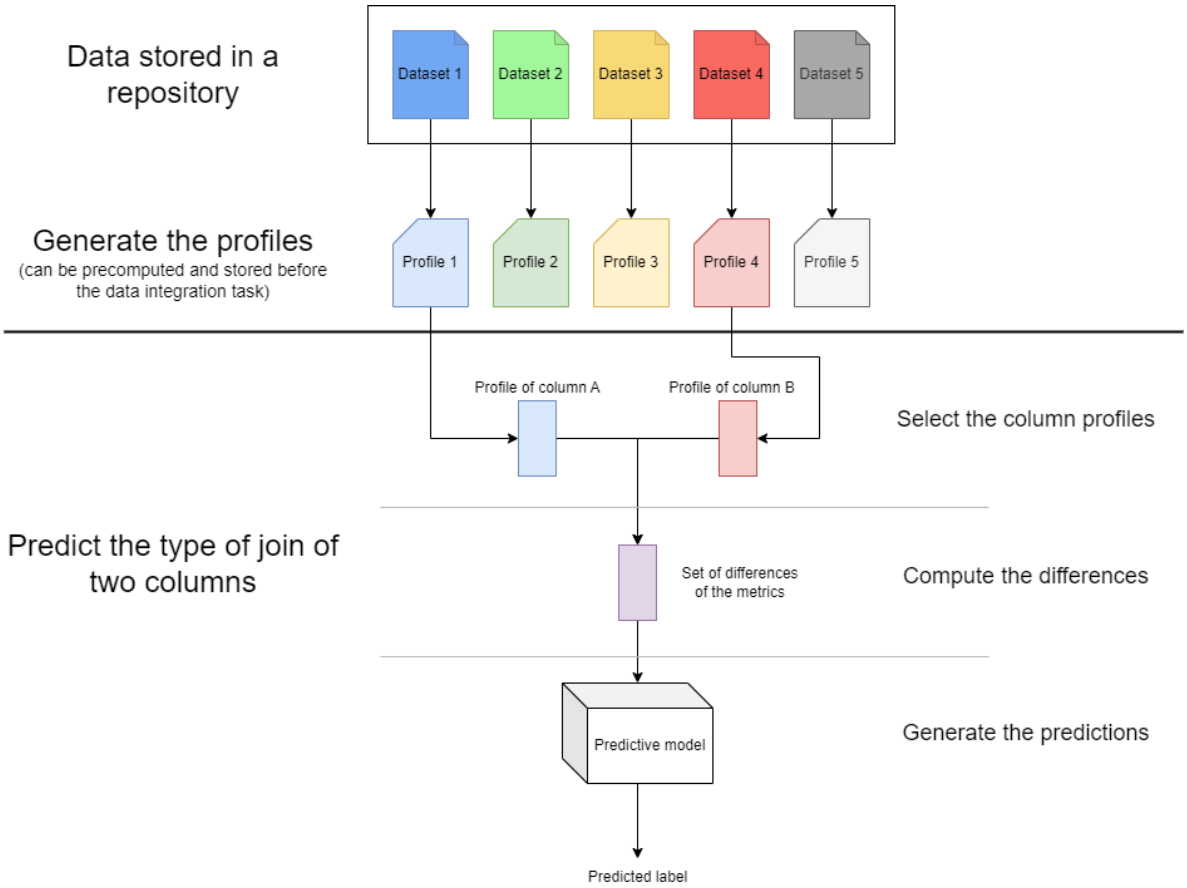


Figure 6: High-level diagram of a *sns* detection system

The first step is to obtain the profiles of the columns of all datasets in a given repository, or of those columns that we want to apply the data discovery process to. This can be done previously to the data discovery task. The actual discovery will take the profiles of the desired columns and compare it with all the others, which implies calculating the difference between the values of each metric and, given the considerations presented above, the distance of the names. The result will be introduced in the model and the category of the join will be predicted. It is important to note that in the experimentations done the cost of selecting the profiles, computing the differences and obtaining the prediction always took below 0.5 seconds. This is a very small overhead that is considerably better that most of the systems presented in section 2, as they required pairwise value comparison as opposed to computing the differences of 19 metrics. The process that, although it can be executed separately, can slow down the system is the generation of the profiles. As such, the following section will address the potential issues with this task.

6.4 Improving the computation of profiles

The candidate joins of the ground truth were moderate in size, with an average column length of 2.200 instances, but ranging anywhere from 100 to 800.000 instances. As indicated before, the time needed to obtain the initial set of metrics was approximately 10 minutes, but we need to acknowledge that datasets with much bigger sizes might be fed to the system more regularly. As such we can not know a priori if the generation of the profiles will supposed an obstacle to the overall efficiency of the system, as even though the profiles can be precomputed and used afterwards, an excessive amount of computation might be crippling. As such, this section will be devoted to studying the time required to generate the profiles and evaluate if some change is needed.

The first important aspect to consider is which is the true conditioner of the time to generate a profile. It is important to remember that we are working with string-based columns, which we can not use directly to obtain the vast majority of the metrics described. Therefore, the system translates each string to a probability, and applies the corresponding numerical processes to obtain the metrics. This implies that each unique string is transformed into the probability of that string appearing in the column. As such, the real factor that governs the time to compute each metric is not the overall amount of instances of the columns, but rather **the amount of *unique* instances**.

> **Example 7: Strings to probabilities**
> One column of a candidate join has 1.000.000 instances, but only 10 of them are unique. This means that the final set of probabilities will contain 10 probabilities, one per unique string. The actual values might change depending on the distribution of values, but the number will always be 10:
>
> - Uniform distribution: [0.1, 0.1, 0.1, ... , 0.1]
> - Non-uniform distributions (example): [0.3, 0.02, 0.05, ... , 0.23]
>
> The other column of the candidate join also presents 1.000.000 instances, but all the strings are unique. This translates into a set of 1.000.000 probabilities:
>
> - Uniform distribution: [0.000001, 0.000001, ... , 0.000001]
> - Non-uniform distributions (example): [0.000001, 0.003, ... , 0.00456]

To test this hypothesis we developed a set of 12 synthetic columns, each containing strings of 15 characters. These columns varied in size and also in the distribution of their values. These characteristics are summarized in the following table:

	1% of unique instances, uniform distribution	10% of unique instances, normal distributions	All values are different
10.000 instances			
100.000 instances			
1.000.000 instances			
10.000.000 instances			

Table 24: Sizes of the experiments to test the profile-generation speed (I)

The vertical axis represents the sizes of the sets, which range from 10 thousand instances to 10 million instances, an extreme enough case to explore how the system behaves for large datasets. The horizontal axis represents the distribution of values. Its main goal is to alter the amount of unique instances of the datasets, which, as discussed before, directly conditions the amount of probabilities and, as a result, the time to generate the metrics. More precisely, the presented configurations yield the following amount of unique instances:

	1% of unique instances, uniform distribution	10% of unique instances, normal distributions	All values are different
10.000 instances	100 unique instances	1.000 unique instances	10.000 unique instances
100.000 instances	1.000 unique instances	10.000 unique instances	100.000 unique instances
1.000.000 instances	10.000 unique instances	100.000 unique instances	1.000.000 unique instances
10.000.000 instances	100.000 unique instances	1.000.000 unique instances	10.000.000 unique instances

Table 25: Sizes of the experiments to test the profile-generation speed (II)

The datasets will have the indicated amount of unique instances, but the amount of times each of those instances will appear (which determined the probability) varies depending on the selected distribution:

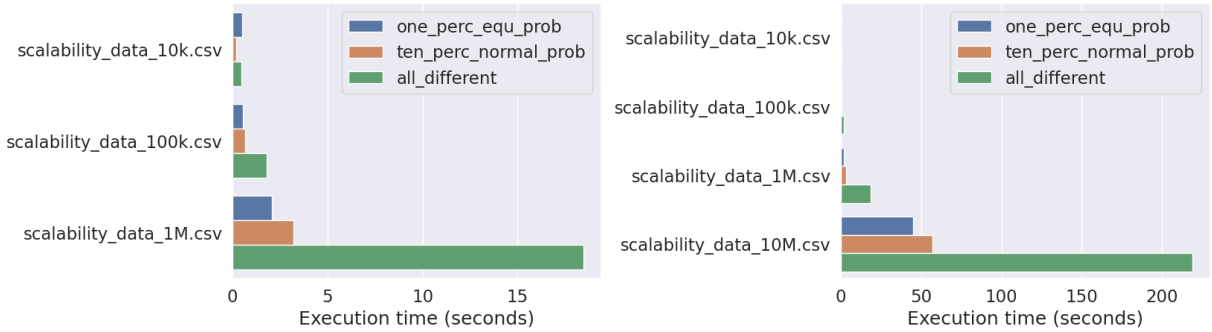
- First column: each instance will appear 100 times
- Second column: each instance will appear as indicated by a normal distribution, implying that the probabilities of the values will vary.
- Third column: each instance will appear only once.

These behaviors translate into the following sizes for the set of probabilities:

	1% of unique instances, uniform distribution	10% of unique instances, normal distributions	All values are different
10.000 instances	100 probabilities	865 probabilities	10.000 probabilities
100.000 instances	1.000 probabilities	8.519 probabilities	100.000 probabilities
1.000.000 instances	10.000 probabilities	85.220 probabilities	1.000.000 probabilities
10.000.000 instances	100.000 probabilities	852.374 probabilities	10.000.000 probabilities

Table 26: Sizes of the experiments to test the profile-generation speed (III)

These are the true volumes of data that the system will work with to generate the metrics. The selection of these specific distributions tries to define three differentiated scenarios. These are, respectively, a column with a lot of repetitions (lower probabilities with respect to the overall size), a column with a standard behavior, defined by a normal distribution, and the worst case scenario, in which every instance translates to a probability. To test the stated hypothesis, we obtained the time required to compute the metrics for each of the datasets.



(a) Datasets of 10k, 100k and 1M instances

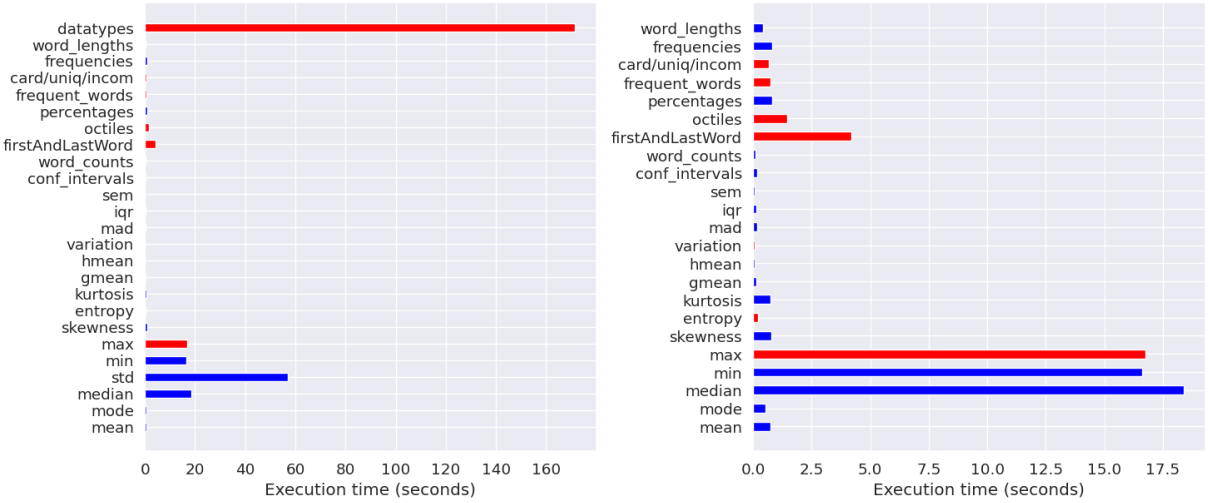
(b) All datasets

Figure 7: Required execution time to obtain the profiles for the tests sets

As we can observe for the larger datasets, specially for the 10 million instances cases, there is a clear pattern. The datasets in which all instances are different take considerably more time to compute than the other two, and the sets that are defined following a normal distribution take more time than the sets with only one percent of unique instances. If we compare the results with the prior tables, it is clear to see that the required time aligns with the number of probabilities of each set. The conclusion that we extract from this experiment is the confirmation that the time required to compute the metrics is defined, not by the total number of instances of a column, but by the number of unique instances of a column. This allows us to draw an upper bound to the time necessary to compute the profiles, as the worst case scenario is defined by all instances being unique. Additionally, we also have to consider that if the columns present repeated values, the time to generate the profiles will be lowered. There are two more noteworthy aspects:

- The first process to execute when generating the profiles is to transform the data from strings to the probabilities. This is a fixed cost that the system will need to incur into, as we can not work directly with the strings.
- Generally, the time to compute each metric scales with the size of the probability set. However, the specific manner in which each metric scales is different. For instance, the calculation of the mean and the calculation of entropy will be conditioned differently by an increase of an order of magnitude to the size of the set of probabilities. The specific pattern of each metric depends on the process to obtain each metric and how it is implemented in the code.

As we have observed, the worst possible scenario is that in which all values are unique. For this case, a dataset of 10 million instances takes over 200 seconds, which is a considerable high time. The last experimentation of this work will encompass the detailed analysis of the time needed to calculate all metrics for the most extreme case presented, with the goal of reducing the time complexity of generating the profiles. The results are as follows:



(a) All metrics

(b) All metrics except *datatypes* and *std*

Figure 8: Required execution time for each metric

The bars highlighted in red represent metrics that were selected in the final feature selection process, which mean that its computation is required to obtain the best performing model. We can observe (4.a) how there is one group of metrics that clearly require more time than the rest: those belonging to the datatypes of the values. One of the metrics selected for the final model was *PctUsername*, which indicates the percentage of instances of the column that present a value that could be categorized as a username. This metric seems particularly arbitrary, and its selection is likely due to a bias in the type of data used as the ground truth. Nonetheless, the cost of obtaining it is prohibitive, so it should be removed.

If we remake the previous plot without the *datatypes* and *std* metrics, the two computations that require the most time to complete, we will be able to see more clearly how the rest of the metrics fare (4.b). Taking into account the size of the dataset used to make the tests, the execution times to obtain the metrics are considerably low. The only exceptions are the *max* probability, *min* probability and *median* probability metrics, which require

over 15 seconds, whereas all the rest are below five seconds and the majority below two seconds. Of the former cases, the only one that is worrying is the *max* probability, as it was selected in the final set of metrics. Its cost is not as prohibitive as the *datatypes* but its removal would definitely speed up the process in a noticeable way. A final model will be developed, removing the *datatypes* and *max* metrics, the two most time consuming parameters that were also part of the best-performing model:

Model	**F1-score (macro)**	**Accuracy**	**F1-score (*sns*)**	**Recall (*sns*)**	**Precision (*sns*)**
Chain model (*all* metrics, F.S., tuned)	**90.39%**	91.03%	**88.08%**	**86.65%**	90.04%
Chain model (no Wasserstein distance)	90.30%	90.90%	88.01%	86.39%	89.77%
Chain model (no W.D., D.T. and max value)	89.30%	90.22%	85.49%	84.87%	86.20%
Multilabel R.F. (*all* metrics)	90.28%	**91.80%**	84.48%	78.94%	**90.94%**
Chain model (F.S.)	76.93%	77.66%	74.54%	81.33%	69.93%
Chain model	78.33%	79.66%	74.13%	79.05%	69.98%
Multilabel R.F. (*syntactic* metrics)	81.61%	83.98%	73.95%	68.50%	81.44%
Multilabel R.F. (tuned)	78.71%	80.20%	73.70%	77.72%	70.18%
Multilabel R.F.	77.03%	78.93%	70.37%	71.96%	68.95%
3-Binary models	-	-	70.15%	69.94%	70.37%

Table 27: Comparison of models (IX)

As we had previously explored, removing the wasserstein distance did not worsen the model in a significant way. However, removing the *PctUsername* and, specially, the *max* probability metrics did considerably reduce the predictive capabilities of the model, with a clear decrease in the *sns* precision. The *sns* F1-score is still high, which implies that this optimized model is still useful for the task at hand, and by removing these features we have substantially reduced the cost of calculating the profiles.

All in all, keeping or removing the highlighted metrics is an issue that needs to be addressed by testing the system in a real life scenario and assessing its viability. If the time to compute the profiles is not a problem, the indicated metrics should be kept to maximize the predictive capabilities. If, on the other hand, generating the profiles becomes an issue, more metrics could be removed. Moreover, we could develop a model that included only the less expensive metrics, and assess if it was still viable. However, we have considered these experiments to be an unnecessary endeavor without having tested the model with real cases.

Testing with external data

7.1 The lack of training data in data discovery

During the development of the models, these have always been tested using cross-validation functions. The purpose of this is to prevent the models from overfitting to the set of data available and to prevent a given test-training split from biasing the models. Nonetheless, in order to thoroughly evaluate a model it is recommended to use a completely external dataset, which will truly evaluate the generalization of the predictive capabilities. In doing so, we prevent biases in the data itself from conditioning the predictions, as the selected instances in the initial ground truth might not be fully representative of the entire population. This will ensure that the model can generalize well and be applied to different sets of data without substantially worsening the results. This was the last intended process to complete the development of the best *sns* detection model. However, a major issue was encountered, as there was not a readily available external dataset to assess the specific task at hand.

As mentioned in the initial sections of this document, the specific detection of *sns* joins is a novel problem that has not been tackled in state-of-the-art systems, at least not with the depth done in this work. Therefore, there are no papers that present data structured and divided in the manner that was required (that is, by type of join) other than the ground truth used. This problem is exacerbated by the generalized lack of data in the data integration field [4]. Nonetheless, there have been some papers that presented similar concepts to the ones used in this project, so there existed the possibility of processing some of the data used in their work to match our criteria. The most promising was Valentine [24], an open-source experiment suite to execute and organize large-scale automated matching experiments on tabular data.

In Valentine, the authors define four schema matching scenarios. Two of those scenarios align with two of the join typologies presented throughout this work: joinable (akin to *semantic*) and semantically-joinable (akin to *sns*). It could be possible to take their openly available data and develop an external testing test to evaluate the models. There were, however, problems with both the data itself as well as the theoretical definitions used.

Most of Valentine's data was synthetic, meaning that the candidate joins were, in most cases, manufactured. More precisely, the joins were formed by the same column in both extremes of the join, with one of the sides having been modified through a character-changing script. As a result, the pairings were always formed by, on one hand, a column with a certain distribution of instances and, on the other, a column with no repeated values. These types of joins are not representative of the task to perform, so they were discarded. However, amongst the synthetic pairings, there were some "natural" *sns* joins that could be used to test the model. Nonetheless, after a close inspection of the data more problems arise, which mainly stem from the definition of semantically-joinable.

7.2 The *sns* definition problem

In the ground truth used, *sns* joins were defined in the following manner: if a pairing of columns shared a (manually-assessed) semantic relationship and could increase the amount of values shared by executing some kind of transformation (i.e. mapping function), it was labeled as an *sns*. The Valentine definition of semantically-joinable states that as long as *one* instance can be mapped from one column to the other (through a given mapping function) the join can be identified as semantically-joinable. These definition are almost identical, so it could be expected for the sets of data that follow both definitions to showcase similar properties. However, it was not the case.

The labelling for the used ground truth was performed manually and, in a practical setting, this implied that most of the categorized *sns* joins presented **a considerable amount of potential mappings**, that is, most of the values of one of the columns could be related to a value of the other column. However, the semantically-joinable columns strictly obeyed to the presented definition, and most of the pairings contained **very few mappings**. This definition of semantically-joinable is problematic for the described solution to the *sns* detection problem, because as long as one instance can be mapped, the rest of the values and their probabilities do not contribute to the type of join. That is, the definition of *sns* is reduced to having one potential mapping, regardless of what the other instances in the columns represent. This resulted in columns with highly dissimilar distributions of values being categorized as semantically-joinable given that just a handful of values could be mapped. The following example illustrates this case:

Country name	GDP (Trillions $)
Spain	1.50
United States	26.85
France	3.09
Japan	4.97
China	21.87

Table 28: GDP per country

Country	State	# votes
USA	AL	2323282
USA	GA	4999960
USA	MA	3658005
USA	NV	1405376
USA	NY	8661735

Table 29: USA 2020 elections' votes count

Example 8: Highly dissimilar distributions
Tables 28 and 29 both present columns that define the concept of a country. However, in table 28 each country appears only once, whereas in table 29 the same country appears in *all* rows. The two columns that define the countries do not have any shared values. However, *USA* can be mapped to *United States*, as they represent the same entity. Therefore, following both the definition of *semantically-joinable* and *sns* joins, these columns should be labelled as so, because there exists a mapping, and by applying this same transformation the quantity of shared values has increased. However, the distributions are highly dissimilar, as table 28 has a uniform distribution with the probability of each country being one divided by the cardinality of the column whereas table 29 only presents one country. As a result, the model trained to compare distributions will not be able to categorize this pairing as an *sns* join given the differences in the distributions.

The definition of semantically-joinable highlights a problem with the original definition of *sns* joins, at least if we are trying to detect *sns* joins through the comparison of distributions. This definition goes against the initial hypothesis established at the beginning of this project, in which we theorized that *sns* joins presented two columns with similar distributions. As we have seen with the exploration of the models, such a statement seems to hold a certain validity, given the accuracy scores obtained when detecting the *sns* joins of the ground truth. However, this is due to the ground truth being obtained through a manual process of join categorization, in which *sns* joins selected by the aforementioned criteria tended to present a high level of potential one-to-one mappings and, therefore, a similar distribution of values. If the definition of *sns* is opened to account for pairings of columns with, potentially, highly different distributions, using metrics that take these distributions into account loses its purpose. If an *sns* join can present very similar as well as highly dissimilar distributions of values, no amount of statistical properties will aid in its detection, as there is not a pattern to learn.

The question that the data discovery field needs to answer is whether the more lenient definition of *sns* joins should prevail. Although in this work we have not been concerned with the posterior integration process (i.e. developing the mappings for the columns in the *sns* joins), the final goal is to develop such a task. Joins in which the number of mappings is small (with a single mapping being the extreme case), the actual joining of the columns is hardly to provide beneficial results, as most of the instances of the columns would not be joined, or a high number of instances of one column would be mapped to a single instance of the other (such as in example 7). By comparing the distributions we can guarantee a relatively high level of mappings, which is likely to bring more relevant information.

Given the results obtained in this work, alongside the aforementioned reasons, it seems like the best approach is to *narrow* the definition of *sns* to contain a relatively high number of mappings and, therefore, relatively similar distributions of values. Allowing for candidate joins with one single mapping to be included into the *sns* category seems far too permissive, making its separation even more difficult and needing to leverage another characteristic of the data to execute the isolation, which, as of now, has not been proposed. Moreover, defining an *sns* through one single pairing can lead to the generation of a considerable amount of false positives, that is, non-*sns* joins defined as so. As for the actual degree of potential mappings to define a join as *sns*, it is a task left for a future analysis.

As a result, the Valentine dataset was completely discarded, as those "natural" *sns* joins presented highly different distributions in comparison to the ones used in the ground truth. This was evidenced by analyzing the metrics obtained from these candidate joins as opposed to the metrics for the training set. In the case of the latter, most of the distribution-related metrics presented means and medians very close to 0, which indicated that the distributions were considerably close. As for the former, these same means and medians deviated substantially from 0, which indicated distinct distribution of values.

7.3 Other tests

Several other sets of data were probed to be the external validation set needed to fully test the model. However, none had any kind of join separation similar to the one defined

in the ground truth, so in order to use them the *sns* joins would need to be fabricated. Some sets of data, such as [25] and [26], had a metadata value with the semantic type of the column. These types belonged to the DBpedia knowledge base, so they followed a structured format. In order to generate *sns* joins, it could be possible to take two columns with the same type that had a low intersection of values. Then, we could assume that the instances of the columns could be mapped (as they present the same semantic concept), hence obtaining a *sns* join.

In practice, this *sns* generation methodology did not work. Most of the sets of data tested presented a very low number of instances (a few dozens per table, compared to the average of 2.200 instances per table in the ground truth), which implied that small deviations on the distributions caused great changes in the values of the metrics, hence further difficulting the prediction process. This is exemplified by most of these pairings being labeled as *non-joinable* rather than *sns* in the attempted experiments.

Moreover, a theoretical issue arises with this *sns* generation method. Despite the columns of the joins belonging to the same semantic type, this does not mean that they are joinable. For instance, two datasets of the semantic type *writer* might present widely different values. One of the datasets might include writers from a literary period such as Romanticism, whereas the other might include writers of sport articles. The semantics are relatively close, but executing a joining would yield no matching instances, resulting in the simple union of values. Once again, the long term goal is to execute the join (with some mapping function), so the training data used should allow for this behavior to occur.

Two additional conclusions can be drawn from the impossibility of executing these latter experiments. On one hand, there needs to be enough instances in the tables for the distributions to sufficiently relevant to draw conclusions. That is, with a small quantity of values it is difficult to assess if the similarity of the distributions of the columns is representative of the underlying joinability. On the other hand, it is important to acknowledge that data discovery (and specially the *sns* detection problem) is a complex task, and relatively high number of false positives is to be expected, such as in the example presented before. A further study should be conducted on which is the percentage of non-semantically related columns that do present a similar distributions of values, at least to a high enough degree for the developed system to categorize them as *sns*.

Closing

8.1 Integration in a data science pipeline

The third and last goal of this Thesis was to integrate the *sns* detection system into a data science environment, where its usage can be easily accessed and incorporated into data processing pipelines. This required the implementation of a mechanism that was able to identify all possible semantically-related candidates (regardless of their syntactic or non-syntactic nature). That is, it was necessary to develop a system that could detect *sns* as well as *semantic* joins.

This work was framed inside the ODIN project [27], an ambitious system that aims at completely automatizing the data science pipeline through the usage of knowledge graphs. ODIN attempts to overcome a series of issues with state-of-the-art methodologies, as they are not suitable for settings with varied data, non-relational structures and evolving requirements. The following figure presents a high-level overview of its functionalities:

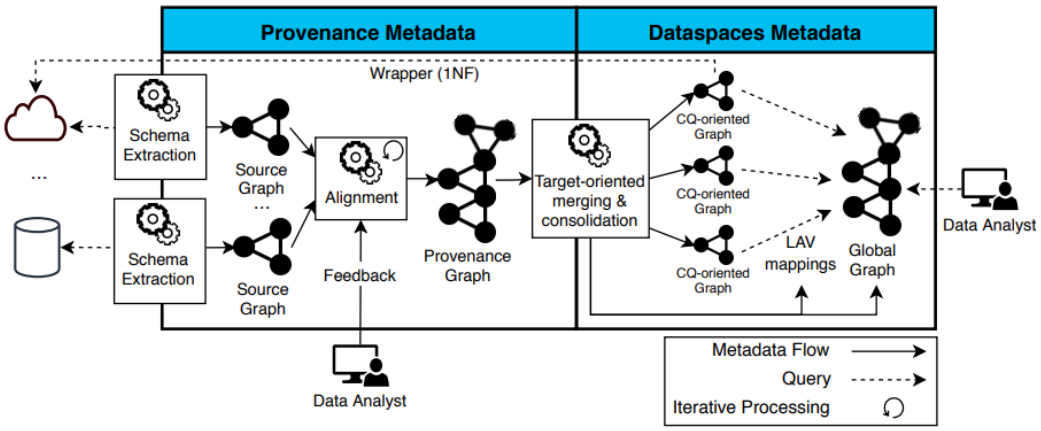


Figure 9: High-level overview of the functionalities offered by Odin

Logically, the first module encompasses a data discovery process, as it is an imperative task to increase the amount of sources available to execute posterior integration tasks with. The implementation of this procedure can be found in NextiaJD [7]. They define a novel notion of join *quality* that relies on, besides the containment of values, the cardinality proportion between the attributes. This is a more precise intuition of the quality of a join that allows to both produce good results while still being able to scale-up to large quantities of data, based on their profile-based approach. Nonetheless, the focus of NextiaJD is on the detection of *semantic* joins, so they do not present an approach to isolate *sns* joins, leaving out a considerable amount of potentially useful pairings. This is the issue that this project aims at solving: developing a complimentary system to detect *sns* joins.

The bulk of this document has been devoted to the exploration and analysis of the *sns* detection problem. However, the early stages of the project also involved the re-implementation of the NextiaJD processes in a more efficient environment. This tasks mainly encompassed the usage of Java to develop a more robust software architecture as well as DuckDB to redefine and optimize the generation of profiles. The remake of the NextiaJD code (provisionally named NextiaJD2) can be found in the following ***link***, alongside all the experiments conducted over the detection of *sns* joins.

The envisioned final system would execute a first exploration to detect *sns* joins using the model developed throughout this project. Its focus is to determine if a given candidate join is a *sns* or it is not a *sns*, leaving further specifications on the type of join to the NextiaJD system. This is so because, although the model used outputs one of the four possible labels of join types, it has been specifically trained to optimize the *sns* detection, which implies that non-*sns* categories might not be optimally predicted. Contrary to this, NextiaJD has been trained in the detection of *semantic* joins and, specifically, their differentiation from *syntactic* joins, which also present a high level of containment. A final diagram of the system would be the following:

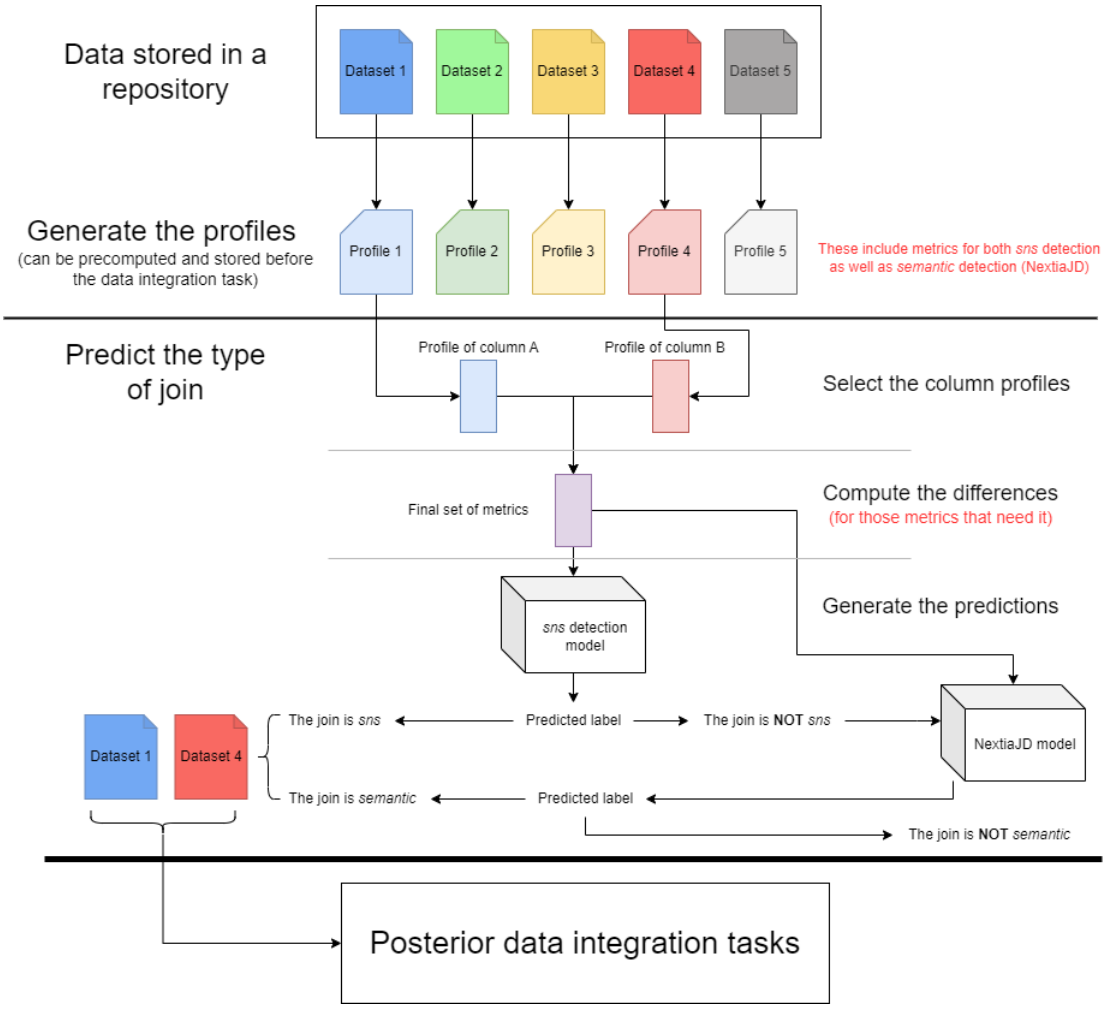


Figure 10: Complete data discovery pipeline

8.2 Future work

As presented in Section 7, the main task left to be performed is testing the system with external data, in order to guarantee the generalization of the properties presented throughout this work. The lack of training data in the data science community and the novelty of such a deep exploration of the *sns* join detection problem imply that finding a set of data that showcased the desired qualities was a complex task. As presented beforehand, none of the probed sets of data had the needed characteristics, so the model could not be applied properly. To that we have to add the need to redefine what an *sns* entails, in order to narrow its definition and make its detection easier to conduct. However, this is an issue that should be discussed by the data discovery field, in order to specify the goal to be targeted.

A first idea to generate additional data could be to use record reconciliation services, such as that of Wikidata, to discover alternatives forms of presenting a same concept. This is possible given that Wikidata has a unique identification for every entity and a set of "representations" for the entity. For instance, Spain's unique identification is *Q29*, and by accessing its page in Wikidata we can observe that in English Spain can also be represented as *Kingdom of Spain*, *ES* and *ESP*. Establishing these alternative representations for an entire set of countries would generate a *sns* joins.

The main problem with the previous approach is that all the generated *sns* joins share *exactly* the same distributions of values, as all the instances are transformed to alternative representations. Unless a posterior modification is applied to alter the distributions, the ending result might be too rigid to represent real repositories of data, in which finding perfectly matching distributions might be an impossible task. Another approach to the *sns* generation to produce more "organic" joins could be to use a lightweight semantic model (e.g. FastText or universal sentence encoder) to determine if a syntactically different value pair from two columns with the same semantic type refers to the same entity. A high semantic similarity (but no perfect syntactic similarity) could mean that they refer to the same entity. If there is a significant "semantic overlap" of value pairs in 2 columns, these columns would be good candidates for *sns* joins.

On a secondary note, it is also left for further work the incorporation of other metrics as predictors of the models. These new metrics can be additional statistical properties not yet leveraged by the models or completely new approaches to the task, as long as their calculation can be stored in a profile to be used afterwards. For instance, as presented in section 2.2, embeddings are a common non-syntactic approach to assess semantic similarity, with a major problem of efficiency due to the pairwise comparison. Developing a system to summarize the set of embeddings of a column could provide relevant information to the task.

As a final remark, the developed model has not yet been tested in real scenarios. Section 6 deals with scalability concerns and even reduces the amount of time required to compute the profiles to optimize the execution of the system as much as possible. Nonetheless, until the system is actually tested with vast quantities of data its effectiveness for dealing with large data repositories can not be ensured.

8.3 Conclusions

This Thesis had as its main objective to study, analyze and solve the *sns* detection problem. More precisely, three separated goals were presented at the beginning of the project, which have been satisfactorily accomplished throughout the execution of the work.

Firstly, we wanted to develop a system to identify *sns* joins. This required a preliminary examination of the characteristics of the problem as well as inspecting the state-of-the-art proposals for data discovery processes. The conclusion to these tasks highlighted that current systems were unable to detect *sns* joins, given their pure syntactic-based approach. This methodology did not suffice in capturing the semantic similarity of joins without intersecting values, which was precisely the problem to solve. As a result, a new approach to data discovery was presented, leveraging both the commonly used syntactic comparison as well as the comparison of distributions, developing a more nuanced definition of similarity that could accurately characterize the semantic closeness of two sets of values.

With this new method to assess the joinability of columns, a thorough analysis of the validity of such an approach was made. This encompassed the evaluation of the comparison of distributions as suitable solution for the task as well as exploring if its combination with syntactical methodologies produced good results. This required the gathering of a series of metrics to evaluate the difference in the distributions of the columns, developing a statistical analysis of the relevance of this metrics and a *sns* detection task. This analysis will be achieved by the development of Machine Learning model to empirically validate the capacity of the metrics to detect *sns* joins. These processes produced satisfactory results and were, thus, combined with syntactic approaches to data discovery. Merging both methodologies into a single model considerably increased the capacity to accurately predict *sns* joins, corroborating the claims stated in the initial hypothesis.

The second goal of the project consisted on modifying the system to be suitable for large-scale data repositories. This implied the removal of a series of metrics that endangered the capacity of the system to scale to large quantities of data, adopting a profile approach to the *sns* detection problem. The usage of profiles allowed the system to benefit from an extensively researched approach to data discovery at scale. Additionally, an study over the execution time proposed alternatives to further improve the efficiency of the system.

Finally, the *sns* detection system was envisioned to be a part of a bigger mechanism, integrating its functionality inside an overarching structure to solve data science tasks. This would allow its use to be accessed seamlessly and be part of an established pipeline that would facilitate the interaction with its functionalities. This was accomplished by the development of a complete data discovery process based on NextiaJD, whose final goal is to be the initial module of ODIN, a system to fully automatize the data science pipeline.

As a final remark, we would like to highlight that the findings of this work should not be taken as the definitive solution to the *sns* detection problem. This is a complex task that requires extensive analysis and testing, which could not be done to the desired degree due to the lack of available data. As such, the insights obtained with this project should be primarily understood as a first approach to detect *sns* joins, a majorly unexplored issue in the data discovery community, presenting a initial solution to help in further research.

References

[1] Grace Fan, Jin Wang, Yuliang Li, and Renée J. Miller. 2023. Table Discovery in Data Lakes: State-of-the-art and Future Directions. In Companion of the 2023 International Conference on Management of Data (SIGMOD-/PODS '23). Association for Computing Machinery, New York, NY, USA, 69–75. https://doi.org/10.1145/3555041.3589409.

[2] A. Bogatu, A. A. A. Fernandes, N. W. Paton and N. Konstantinou, "Dataset Discovery in Data Lakes," 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 2020, pp. 709-720, doi: 10.1109/ICDE48307.2020.00067.

[3] Michael Muller, Ingrid Lange, Dakuo Wang, David Piorkowski, Jason Tsay, Q. Vera Liao, Casey Dugan, and Thomas Erickson. 2019. How Data Science Workers Work with Data: Discovery, Capture, Curation, Design, Creation. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19). Association for Computing Machinery, New York, NY, USA, Paper 126, 1–15. https://doi.org/10.1145/3290605.3300356

[4] Stonebraker, Michael and Ihab F. Ilyas. "Data Integration: The Current Status and the Way Forward." IEEE Data Eng. Bull. 41 (2018): 3-9.

[5] Alon Halevy, Anand Rajaraman, and Joann Ordille. 2006. Data integration: the teenage years. In Proceedings of the 32nd international conference on Very large data bases (VLDB '06). VLDB Endowment, 9–16.

[6] Natalya F. Noy, AnHai Doan, and Alon Y. Halevy. 2005. Semantic integration. AI Mag. 26, 1 (March 2005), 7–9.

[7] Nadal, S., Panadero, R., Flores, J., Romero, O. 2023. Measuring and Predicting the Quality of a Join for Data Discovery. arXiv e-prints. doi:10.48550/arXiv.2305.19629

[8] Papadakis, George Svirsky, Jonathan Gal, Avigdor Palpanas, Themis. (2016). Comparative Analysis of Approximate Blocking Techniques for Entity Resolution. Proceedings of the VLDB Endowment. 9. 684-695. 10.14778/2947618.2947624.

[9] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and Filtering Techniques for Entity Resolution: A Survey. ACM Comput. Surv. 53, 2, Article 31 (March 2021), 42 pages. https://doi.org/10.1145/3377455

[10] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In SIGMOD Conference. ACM, 847–864.

[11] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In ICDE. IEEE Computer Society, 1001–1012.

[12] Mayank Kejriwal and Daniel P. Miranker. 2015. Semi-supervised Instance Matching Using Boosted Classifiers. In ESWC (Lecture Notes in Computer Science), Vol. 9088. Springer, 388–402.

[13] Sagar Bharadwaj, Praveen Gupta, Ranjita Bhagwan, and Saikat Guha. 2021. Discovering Related Data At Scale. Proc. VLDB Endow. 14, 8 (2021), 1392–1400

[14] R. Castro Fernandez et al., "Seeping Semantics: Linking Datasets Using Word Embeddings for Data Discovery," 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 2018, pp. 989-1000, doi: 10.1109/ICDE.2018.00093.

[15] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity Based Approach. In ICDE. IEEE, 456–467

[16] Cong, Tianji Gale, James Frantz, Jason Jagadish, H. Demiralp, Çağatay. (2022). WarpGate: A Semantic Join Discovery System for Cloud Data Warehouse. 10.48550/arXiv.2212.14155.

[17] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M. Procopiuc, and Divesh Srivastava. 2011. Automatic discovery of attributes in relational databases. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD '11). Association for Computing Machinery, New York, NY, USA, 109–120. https://doi.org/10.1145/1989323.1989336

[18] Yeye He, Kris Ganjam, and Xu Chu. 2015. SEMA-JOIN: joining semantically-related tables using big table corpora. Proc. VLDB Endow. 8, 12 (August 2015), 1358–1369. https://doi.org/10.14778/2824032.2824036

[19] E. Zhu, Y. He, and S. Chaudhuri. Auto-join: Joining tables by leveraging transformations. PVLDB, 10(10):1034–1045, 2017.

[20] Zeakis, Alexandros, George Papadakis, Dimitrios Skoutas, and Manolis Koubarakis. 2023. "Pre-Trained Embeddings for Entity Resolution: An Experimental Analysis [Experiment, Analysis and Benchmark]," April.

[21] Kanji, G. K. (1995). 100 statistical tests. London [u.a.]: Sage. ISBN: 0803987056

[22] Daniel Abadi, Anastasia Ailamaki, David Andersen, Peter Bailis, Magdalena Balazinska, Philip A. Bernstein, Peter A. Boncz, Surajit Chaudhuri, Alvin Cheung, AnHai Doan, Luna Dong, Michael J. Franklin, Juliana Freire, Alon Y. Halevy, Joseph M. Hellerstein, Stratos Idreos, Donald Kossmann, Tim Kraska, Sailesh Krishnamurthy, Volker Markl, Sergey Melnik, Tova Milo, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Jignesh Patel, Andrew Pavlo, Raluca A. Popa, Raghu Ramakrishnan, Christopher Ré, Michael Stonebraker, and Dan Suciu. 2019. The Seattle Report on Database Research. SIGMOD Rec. 48, 4 (2019), 44–53.

[23] Read, Jesse Pfahringer, Bernhard Holmes, Geoffrey Frank, Eibe. (2009). Classifier Chains for Multi-label Classification. Machine Learning. 85. 254-269. 10.1007/978-3-642-04174-7_17.

[24] C. Koutras et al., "Valentine: Evaluating Matching Techniques for Dataset Discovery," 2021 IEEE 37th International Conference on Data Engineering (ICDE), Chania, Greece, 2021, pp. 468-479

[25] Ritze, D., Bizer, C. (2017). Matching Web Tables To DBpedia - A Feature Utility Study. International Conference on Extending Database Technology.

[26] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. Proc. ACM Manag. Data 1, 1, Article 30 (May 2023), 17 pages. https://doi.org/10.1145/3588710

[27] Nadal, S., Rabbani, K., Romero, O., Tadesse, S. (2019). ODIN: A Dataspace Management System. International Workshop on the Semantic Web.