Data Science End-to-End Project

(Part 2: the Data Analysis Backbone)

1. Motivation

We completed the data management backbone with the implementation of the exploitation zone, where we defined the data that we offer the analysts to perform further downstream tasks with. In this second part of the project we aim at developing the data analysis backbone. That is, we will utilise the data processed in the management backbone to carry out specific analytical tasks.

The data management backbone is common for all the organization and, hence, there is just one implementation of it. In contrast, the data analysis backbone is typically tailored to individual projects or specific types of analytical tasks. Consequently, an organization may develop multiple data analysis backbones to meet diverse analytical needs. You have to implement **at least one analytical backbone**.

Disclaimer: ADSDB is not meant to cover data analysis content. Hence, it is fine if you implement simple processes in the tasks we will define. This is especially so for the development of the model, as other, more specialised courses will emphasise these aspects. What we demand is that you develop all the tasks needed for the pipeline (even if they are straightforward implementations) and that you are able to justify your decisions throughout the project, including results. Logically, going beyond simple baselines, or even implementing several analytical backbones, will count positively towards the evaluation.

2. The Data Analysis Backbone

In this second part, aligned with your project context, you first need to explicitly state your analysis question, which you might have already done in the first part. Regardless, be sure to specify the goal of your backbone, as you cannot start without a clear analytical objective. Example of analysis questions for an insurance company follow:

- What are the most relevant features from the customer-related data available in order to identify customer churn?
- Predict, via the contact channels used by customers and their last actions in the last months, the degree of satisfaction of a customer.
- Classify the type of customers we have according to our type of insurance and the engagement of their relatives (i.e., how many relatives also have insurances with us).

Pay attention to the data you have and devise a clear analysis question that fits your project. If you have any doubts, don't hesitate to consult your supervisor. Once the analysis question is clear, let us proceed with DataOps-related aspects.

The analytical backbone will follow a similar structure to that of the management backbone, in that we are going to separate the pipeline into zones, each being responsible for applying certain transformations/processes to the data. Here, you are also recommended to implement the pipeline at your UPC Drive, creating different folders for each zone and employing Google

Colab notebooks (refer to the statement of the first part for more details). Nonetheless, you are free to choose different tools as long as your pipeline meets the project's restrictions.

The **analytical sandbox** is the first zone of the analytical backbone, and its purpose is to store the data that you deem relevant for the analytical task. That is, you have to select a subset of tables (and, potentially, a subset of columns from those tables) which will serve as the input for your analytical process. As you will see, we will further process the selected data, so this is just a first, hand-made filter to reduce the amount of data rather than a final data selection to be ingested by the model.

This data will be stored in yet another different database, as you want to have a copy of the *exact* data you used to perform the analysis, in case you want to go back to it. Once again, we recommend the use of an analytical database, such as DuckDB (refer to the statement of the first part for more details), but you can opt for other tools.

Next is the **feature engineering** zone. As you might guess, here we will perform transformations over the columns that we have selected to further refine them for the analytical task. This can be divided into three main tasks.

First, we will perform **feature generation**. A feature is a measurable attribute or property of data used as input for a machine learning model or analysis. It represents a characteristic of the data that helps describe, differentiate, or predict a target outcome. Hence, the set of columns/attributes of your datasets are not necessarily features, as they might not be helpful. The goal is, then, to adapt them. This can be done by (i) applying transformations to columns:

- Mapping from very specific attributes to broader, more useful features. For instance: birthdate to age, address to zip code, coordinates to region, specific colors (crimson, teal, etc) to "basic" colors (red, blue, etc.)
- Discretization. For instance: income (continuous, €) to income bracket (e.g. low, medium and high), from temperature (continuous, °C) to "perceptions" (e.g. hot and cold), from weight (continuous, in Kg) to weight class (e.g. lightweight, middleweight, heavyweight).

  It is easy to think that by discretizing data you are losing information. This can certainly be the case, but there are scenarios where discretizing can be helpful. For instance, if there is a lot of noise or outliers in the data, discretizing reduces their impact. Discretizing might also facilitate encountering and interpreting relationships with other variables.

- Domain-specific mappings. For instance: text to sentiment (via sentiment analysis), purchase history to recency (time since last purchase), login frequency to active user (define how active an user is, e.g. daily, weekly, infrequent), location to distance (from a given point), description to keyword count.

Another option (ii) is to generate features from the combination of several other features. For instance, weight and height can be combined into BMI, profit and revenue can be combined into profit margin, distance and time can be combined into speed. Similarly as with discretizing, this might seem unnecessary, as modern ML models are able to capture nonlinear relationships between attributes. Nonetheless, expliciting these relationships via variables can help in the explainability of the models, as the interactions are displayed in the data and not hidden in model computations.

Both of these tasks can be made easier by joining tables, as the amount of attributes available increases. This might require joining tables previously separated in the exploitation zone.

Related to feature generation is the task of **labelling**. Labelling is the process of assigning correct and meaningful labels to data points in a dataset (which we define as the target variable), often for supervised learning tasks. This is common in, for instance, image processing, where we want to develop a model to predict the content of an image. To train the model, we first need to define the content of each image of the training set; that is, to label every image. In the case of tabular data, one of the attributes already present in the data is often defined as the target variable. However, we might want to generate a new target variable that is more appropriate for our task.

For instance, you might have a dataset with information about products sold at a given store, with one of the fields being how many people bought a product in the last year. You can use this as your target variable. However, there might be some products that were released only some months ago. Hence, comparing the absolute amount of sales of those products with the rest of products (those that have been available for a year) is not significant. In that case, you could generate a new target variable: *average daily sales*, which is more representative of your analytical goal: measuring the popularity of a product.

In this same token, you might want to discretize the label, following the previously stated rationale.

Next step is **data preparation**. In the trusted zone of the analytical backbone we implemented *generic* data quality (i.e. "fixing mistakes"). Here we will prepare data to be ingested by a model (i.e. *specific* data quality). Some processes that can be included here are as follows:

- Outlier Handling: Outliers can considerably bias the predictions of a model, as having to account for an exceedingly abnormal value worsens the average prediction. To minimize this impact, we can handle them by either removing the outliers entirely or imputing them with more reasonable values based on domain knowledge, statistical methods, or other observations in the dataset.
- Missing Value Handling: Most machine learning models cannot natively process missing values, making it a common prerequisite to address them beforehand. This can be done by either removing rows or columns with excessive missing values or imputing them via techniques like mean, median, or mode substitution or advanced imputation algorithms (e.g., K-Nearest Neighbours, multivariate imputation)
- Normalization: Normalizing numerical values is a common preprocessing step that ensures features have a consistent scale, which is especially important for distance-based models (e.g. K-Nearest Neighbors). Another case in which normalization is widely used is with neural networks, as the gradient descent algorithm (used to define the values of the nodes in the network) is very sensitive to magnitude. Normalization methods include min-max scaling, standardization (z-scores), and robust scaling (for datasets with outliers).
- Encoding Categorical Variables: Many machine learning models require numerical input, so categorical variables must be encoded into a format suitable for processing. Techniques for encoding include one-hot encoding, label encoding, ordinal encoding, or target encoding, chosen based on the model and the nature of the categories.

To close off the feature engineering zone, and once the data has been prepared, we create the test and train splits that will be employed to train and evaluate the model. These train and test datasets should also be stored for the sake of traceability.

We recommend placing these processes (feature generation, data preparation, labelling (if required) and generating the training and test sets) in separate notebooks. Realise that the order matters between them! So properly orchestrate them to facilitate maintenance.

At this point in the pipeline we have defined the data to perform analysis with. Hence, the final zone is that of **model generation**, used to train and validate models, as well as to visualize and analyze the results from the analytical task. Your knowledge of ML models might be very limited at this point in the master. This is perfectly fine, as there are dedicated courses where you will explore ML in depth. Moreover, ADSDB is not meant to cover the analytical part, thus we will not focus on the correctness of the algorithms chosen or the complexity of the ML pipeline (yet, we expect a minimum level of rigor as of learnt at this stage of the master).

Python and R are recommended for this project, specifically the former due to the native integration with notebooks and the abundant amount of libraries that facilitate the use of models (mainly, scikit- learn), but it is up to you to decide what tools to use.

The minimum set of steps to fulfil are:

- Define a model to be used and train it. If you are not experienced in ML, you can use the concepts learnt in SIM (Statistical Inference and Modelling), such as linear regression or logistic regression.
- Store the trained model. Use the format that you think is most suitable.
- Choose a metric to evaluate the model and validate it. Once again, you can use the metrics defined in SIM, such as the coefficient of determination (i.e. R^2).
- Analyze (and visualize, if possible) the results obtained from the model with regards to your analytical task.
- Store the final model (potentially deploy it).

If you want to go beyond linear models, we suggest you employ ensemble models. These models are more complex and able to capture non-linearities in the data, making them obtain better results. In particular we recommend *Random Forests*, which is a type of ensemble model that is easy to deploy, robust and interpretable. We encourage you to seek further information to learn more about it.

Another interesting option is the (simplified) employment of generative AI (e.g. querying a model downloaded from HuggingFace to generate *something*). Logically, you can implement whichever other process you find interesting or relevant. Just make sure to agree with your supervisor the correctness of the proposal with respect to the course criteria.

As for the list of tasks to perform, you can enrich (i.e. this is **optional**) the pipeline with additional processes. For instance:

- Model selection process: you might want to try several models with preliminary (i.e. simple) tests before deciding to stick with one for the rest of the pipeline.
- Choosing several evaluation criteria to generate a more complex and nuanced evaluation.
- Hyperparameter tuning (for models that require so).
- Cross-validation.

- Feature selection.

The notebook structure of the model generation zone highly depends on the tasks that you have implemented. In a simple implementation one notebook for model training and another for validation and analysis might suffice (also keep in mind the storage of the generated models!). For more complex pipelines, more notebooks with more specific tasks might be required. Regardless, structure your notebooks to keep the desirable properties of DataOps (reusability, dynamicity, etc.).

Important Note: Nowadays there is a plethora of tools that cover the whole data analysis backbone (including some governance). For example: MLOps, Weight & Biases, Hugging Face, Hopsworks among many others. For this project, we advise you to avoid tools that automate this part. For sure, they will be interesting in your future professional career, but for the sake of learning, it is better you develop them on your own during the project and understand pros and cons by yourself. If you plan to include some tool of this kind, please, discuss it first with the project advisor.

Constraint 8. The scripts to generate the analytical sandbox, feature engineering and model generation zones must be implemented in notebooks and organized in subfolders (e.g., featureGeneration / dataPreparation). The particular structure of the notebooks will be correct as long as it is sensible (add any relevant explanation in the document if you deem it necessary).

Constraint 9. Following the idea of the first part of the project, create notebooks to scrutinize each zone. In these notebooks you must include the schema and profiles of the analytical sandboxes created, and the training and validation datasets. Of course, depending on your project you may need to go beyond profiling, which can include PCA or other techniques seen in MVA (Multivariate Analysis). Importantly, pay special attention when profiling the training and validation datasets to showcase your posterior data analysis will not be biased/useless (this is a starting point for *explainability*, a key aspect in DataOps). For this reason, in the document, you must include a discussion about the training/validation sets profiling and their validity for the analysis at hand.

Constraint 10. In the document, discuss the decisions taken in the model generation zone (type of algorithm chosen for the model, evaluation metric chosen, etc) as well as the results obtained. If other tasks were included, such as hyperparameter tuning or cross validation, its addition should also be justified. Recall that in this project it is not important to obtain good results in the evaluation metrics (we are not evaluating the value of the model).

3. Operations

You should now include the data analysis backbone in your operations. To do so, follow the same instructions as per the first part of the project and include the new artefacts created for the data analysis backbone in your orchestration.

Additionally, you must create a run-time app in operations. Typically, you need to create a simple app that reads new data, prepares it to be input into the model created and prints the output of the model.

4. Advanced Topic

The data management and analysis backbones generate the skeleton of a data science project. However, further iterations and needs (that for sure will come) will complicate, bit by bit, your DataOps solution. To exemplify it, in this second part of the project, we want you to dive into some relevant yet, typically overlooked aspects specific to Data Science projects and include it in your project.

For this part of the project you must choose one of the following topics (these are further detailed at the end of the document):

- Data Discovery
- Feature Selection
- Data Augmentation
- Data Quality
- Entity Resolution

Note: you may propose a topic to the lecturer additionally to these ones. But be sure to get a green light from the lecturer before going for it.

For the chosen topic, you must learn the basics about it and implement a basic solution tackling it in your project. More precisely, you have to define an extension on top of your project to include the studied topic. The objective is to practice how to incrementally add complexity to the baseline we created while learning more about hot topics in Data Science.

First, identify the zone either in the data management or analysis backbone where your solution must be implemented. After reading the topic, you must decide in which zone you should include new scripting / data repositories to include it in the project. You must justify in the document your decision. Importantly, we want you to realise that the zones facilitate the maintenance (adding, removing or modifying modules to tackle new needs) of a data science project.

Once identified the zone, follow the same pattern as for the data management and analysis backbones: use notebooks in the prototyping phase to generate a solution. Once you are satisfied with it, you should move it to operations and generate the required operations artefacts and orchestrate them with the other modules of your project.

The goal of this part of the project is for you to experience the process of exploring scientific literature (with the complexity it involves), distilling the relevant information and implementing an advanced process. Logically, we are not expecting you to redefine the state-of-the-art of the chosen field nor to mimic very complex processes implemented by experienced researchers over months. We ask that you are able to understand the chosen domain, the challenges it involves, the solutions proposed by the community and that you are able to include a part of it (which requires extensive thinking over how to do it) in the pipeline.

Constraint 11. Identify the zone where to add your advanced topic prototype. Justify your choice. Next, propose a new notebook (or set of notebooks) to add in this zone and discuss how they relate to the existing ones. Maintainability and separation of concerns must be guaranteed. To decide if one or many notebooks are needed, we advise you to follow the same principles as in the rest of the project; i.e., be sure to separate different concerns in different notebooks.

Constraint 12. Your operations layer must be able to execute the complete set of artefacts needed to cover the solution devised for your advanced topic. It is important you properly orchestrate your solution for the advanced topic with the other elements of the zone where you included it.

5. Deliverables

The outcome of the second part of the project is twofold:

- An explanatory document (max. 10 pages long),
- The project repository (it must be the same as in the first part, but now also including the data analysis backbone and the advanced topic)

The document must contain the following sections:

1. Your project supervisor should have access to your **development** (e.g., Drive) and **operations** (e.g., Github) **platforms** where you developed your project. Instructions on how to access these platforms must be in a mandatory section to be included in the document (first section of the document, in a separate page, without numeration) providing the link to these platforms. This section does not count for the maximum number of pages to be used in the document.
2. **Context**. If there is any change you want to include in the context delivered for the first part, do it here. Further, precisely and concisely state your analysis question.
3. **The data analysis backbone**. Using the figure from the slides, instantiate it (i.e., add on top) the tools you used for each element from the architecture.
4. **Advanced topic**. Identify the advanced topic chosen. Explain the solution you implemented and the different notebooks used in the prototyping phase.
5. **Operations**. Add an explanation of how you have organized your operations for the data analysis backbone and the artefacts needed to implement the advanced topic chosen. Specifically, pay special attention to justify the new code added to orchestrate the pieces added.

For items 3 and 4, it is important you discuss the pros and cons of your chosen solution. We will positively assess if you identify the limitations of your current approach (do not worry to state them, it is a baseline, simple solution, it will have plenty). Do not repeat here the information in the notebooks / code. The document should be an overall description to understand your development and operations platform and facilitate the supervisor to explore it.

6. Evaluation

The evaluation of this part of the project is follows the same criteria as for the first part:

Dynamicity. How easy is it to add a new feature to be considered in the analytical backbone? And add a complete set of new features? How easy is it to change the transformations executed between two zones?

Reusability. Code and data are not duplicated and they are well organized.

Openness. Your system could be easily extended and evolved with new / advanced aspects. This is evaluated via the solution provided for the advanced topic.

Single source of truth. Analysts can access a single source of truth from where to start the analysis.

Reproducibility. The executables in operations can be easily executed and are properly documented to do so. Similarly, for the notebooks in development.

Soundness. The choice of tools / solutions is adequate for each zone.

Completeness. All constraints in the statement are met.

Rigorous thinking. In the document, there is a detailed discussion about pros and cons of each solution and the students identify room for improvement for advanced solutions.

Annex: Advanced Topics

Find below a description of the topics proposed.

Data Discovery

A key aspect of Data Science is to learn powerful mathematical models that can predict or classify all sorts of events of interest. This is done by leveraging a set of features (roughly speaking, variables) from which the models can learn the necessary patterns and apply the collected knowledge on subsequent predictions.

Thus, the characteristics of the available data are a crucial conditioner of the correctness of the model. First, in terms of quality, as better data (that is, how well it represents a given domain) leads to more accurate results. Secondly, in terms of quantity, as the more data we have, the more we avoid overfitting the model to, potentially, only a subset of entities that are not representative of the entire domain.

This implies that an essential aspect of data scientists' work is to gather relevant datasets for the problem at hand. Indeed, the more datasets and variables you gather, the more informative data features you will likely identify to learn robust, generic models not falling into overfitting. Practically, what this means is that you have a dataset and you want to find data that is complementary to improve its quality. By complementary we usually refer to being able to perform a join or union with.

To accomplish this idea, in data science we rely on *data lakes*. A data lake is a huge repository of data where different companies or organizations place their data assets (their own or obtained from third parties) so that data scientists have access to a wealth of data to facilitate building better models. Hence, we have an abundance of data, coming from different sources, and being highly heterogeneous.

Manually navigating through these massive repositories to find relevant data is unfeasible. Moreover, datasets coming from different data sources have syntactic (e.g., different formats such as JSON, CSV, relational, key-value, etc.) and semantic (e.g., customer vs. client) heterogeneities that sometimes make it hard to realize these datasets have complementary data. Thus a new problem is created: how can I create a system that can autonomously analyze a large amount of datasets to detect which ones are relevant to my problem? And also, how do I define whether a dataset is relevant in the first place? Data Discovery is the field that aims at answering these questions.

Thus, we may summarize this discussion with the following statement: "*Data Discovery is the automated set of tasks that facilitate identifying relevant and complementary data sets for a given analytical task from a huge repository of data sets*".

Seminal bibliography:

- Paton, Norman W., Jiaoyan Chen, and Zhenyu Wu. *Dataset discovery and exploration: A survey*. ACM Computing Surveys 2023.
- Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, Nikolaos Konstantinou: *Dataset Discovery in Data Lakes*. 2020 IEEE 36th International Conference on Data Engineering (ICDE 2020).
- Javier Flores, Sergi Nadal, Oscar Romero: *Towards Scalable Data Discovery*. 24th International Conference on Extending Database Technology (EDBT 2021).

Feature Selection

Big Data plays a significant role for building efficient models improving decision making, yet it comes with the problems of storage (and large processing times), challenging data understanding and visualization, poor generalization by models due to overfitting, and lengthy model building times. As the dimensionality of a dataset increases, the required number of samples needed to build a model that generalizes it properly grows more than exponentially. The state of having insufficient samples compared to the dimensionality of a dataset is referred to as sparseness and it generally increases as the dimensionality increases. Sparseness negatively affects the quality of a model's performance metrics, for example, its accuracy.

Besides the problem of sparseness, high-dimensional datasets are difficult to understand and visualize. Models built with such datasets are thus difficult to explain, especially in terms of the many features; the time required to build such models could also be very long. Associated with high dimensional data is also the need for relatively more storage space which signifies increased expenses whether on-site or in the cloud. These resultant challenges from the high dimensionality of datasets have been termed the curse of dimensionality. To fight the curse of dimensionality, effective techniques to reduce the dimensionality of datasets by selecting the most informative features of a dataset are necessary.

Many of such techniques have been long proposed and are called Feature Selection (FS) methods. Most FS methods were designed for small-medium sized datasets with polynomial theoretical runtime and so in the face of large-sized datasets, their effectiveness becomes jeopardized and they fall into the curse of dimensionality.

The aim of this topic is thus to study the state-of-the-art on "*feature selection methods, focusing especially on the ones that scale well over Big datasets; highlighting the current issues of feature selection in Big Data*".

Seminal bibliography:

- Kewei Cheng, Jundong Li, and Huan Liu. 2016. FeatureMiner: a tool for interactive feature selection. In CIKM. 2445–2448
- C. Reggiani, Y.-A. Le Borgne, and G. Bontempi, "Feature selection in high-dimensional dataset using mapreduce," in Benelux Conference on Artificial Intelligence, pp. 101–115, Springer, 2017.
- D. Peteiro-Barral, V. Bolon-Canedo, A. Alonso-Betanzos, B. Guijarro-Berdinas, and N. Sanchez-Maroono, "Scalability analysis of filter-based methods for feature selection," Advances in Smart Systems Research, vol. 2, no. 1, p. 21, 2012.

Data Augmentation

The abundance of data originated from Big Data implies that our predictive models can be trained with increasingly more information. This leads to better predictions, as we have access to an improved representation of the domain we are working with, which also reduces overfitting. Nonetheless, this is only true if the new data is relevant to the model. That is, that provides new and relevant interactions with regards to the target variable.

Hence, the increase in the availability of data also creates a new problem: how can I assess whether the new data is relevant for my model? That is, how can I know if adding new information is going to improve the predictions or make them worse (due to poor quality)? This

is a crucial problem to address, as having more data to work with does not provide any benefit unless we are certain that the new data information is helpful. As a result, it is essential to establish methods for evaluating the relevance and quality of new data. This involves assessing whether the data provides meaningful information related to the target variable and ensuring it aligns with the existing dataset in terms of quality and context.

Data augmentation encapsulates all the processes to perform such a task, that is, to find relevant data to *augment* a given dataset with, considering as the primary objective the improvement of the predictive capabilities of the model (measure in metrics like precision, recall or MSE). With the growth of Big Data and the increasing complexity of datasets, the importance of systematic and reliable data augmentation has grown exponentially. In earlier days, practitioners relied on simple, ad hoc methods to enhance their datasets. Today, automation and intelligence in augmentation processes are critical. Libraries and frameworks now offer advanced tools for implementing augmentation strategies, allowing practitioners to fine-tune their pipelines based on the task at hand. However, augmentation is still a process that often requires deep domain knowledge and careful tuning to ensure its success.

Interestingly, data augmentation techniques make use of “real” data (extracted from data lakes, capturing real-world occurrences and phenomena), but they can also use synthetic data. That is, some methodologies fabricate their own datasets following a set of latent properties extracted from the original domain. By capturing underlying characteristics of the data, they are able to create new data that is able to represent interactions that are hidden. This is especially helpful when collecting new data is expensive or impractical.

Thus, we may summarize this discussion with the following statement: “*Data Augmentation involves a series of automated tasks to discover new data to improve the predictive capabilities of previously-defined machine learning models*”.

Important note: *Data discovery* is often the first part of the data augmentation pipeline. Moreover, certain data augmentation techniques are akin to performing *feature selection*, as datasets are joined and then the most relevant features (including the new ones) are selected (although more modern techniques avoid using traditional feature selection processes that require training a model several times). Hence, data augmentation, for the purposes of the course, can be seen as a holistic process that encompasses all the tasks from the detection of new data to the creation of a new model with improved performance. As a result, the focus is placed on combining all the steps rather than on individual parts of it

Seminal bibliography:

- Mumuni, Alhassan, and Fuseini Mumuni. "Data augmentation: A comprehensive survey of modern approaches." Array 16 (2022): 100258.
- Ionescu, Andra, et al. "AutoFeat: Transitive Feature Discovery over Join Paths." 2024 IEEE 40th International Conference on Data Engineering (ICDE). IEEE, 2024.
- Lin, Yin, et al. "SMARTFEAT: Efficient Feature Construction through Feature-Level Foundation Model Interactions." arXiv preprint arXiv:2309.07856 (2023).

Data Quality

Data is the main asset of a Data Science project. However, different data sets may attain different data quality levels. For example, data from a source generating too many errors (e.g., a faulty sensor, manually introduced data by humans with default values, etc.) might be problematic if not filtered and cleaned before being used. In databases, a well-known motto is that of Garbage In-Garbage Out (GIGO), which means that if you do not care about the quality of your data, then your models and analysis drawn from them will indeed be garbage too.

With the arrival of Big Data and the availability of a deluge of digital information, data quality has become even more crucial and more difficult to deal with.

In the past, we talked about data curators, data wranglers or the like, which were people manually taking care of the data quality or creating semi-automatic processes that helped to identify errors and process them. To help these people, different data quality perspectives were introduced, which provided hints of what to look at to assess data quality (e.g., accuracy, freshness, completeness, uniqueness, etc.). But now the focus is on automatic data quality processes, or at least, as automatic as possible.

A huge effort was recently invested in the area of supporting the data curator. Indeed, some of the companies raising more funds in the last year in Silicon Valley are related to this topic. But unfortunately, data quality has some artisanal aspects, very ad-hoc to the project at hand, that make this problem huge and far away from being solved.

Nevertheless, checking the bright side of this history, there have been recent and interesting advances that are allowing us to move forward in advanced data quality processes for Data Science. The most crucial of them all is the introduction of the concept of denial constraint (DC). A denial constraint is a formalism able to express most of the integrity constraints we are used to (e.g., key constraints, functional dependencies, etc.) but in a compact yet very expressive manner. Denial constraints are grounded in logic, which in addition allow to automatically infer interesting consequences (such as clashing DCs, subsumed DCs, etc.). Out of this formalism, a wealth of research appeared as well as very popular tools already in the market. In short, DCs have allowed to automate most of the Data Quality lifecycle: elicit DCs from data sets, propose automatic reparation rules and apply them (always supervised by humans in what is called human-in-the-loop tools).

This topic can be reformulated in different manners, but essentially address the same problem. For example, "*how to assess the quality of a data source or how to repair dirty data"*.

Seminal bibliography:

- Redyuk, S., Kaoudi, Z., Markl, V., Schelter, S.: *Automating data quality validation for dynamic data ingestion*. 24th International Conference on Extending Database Technology (EDBT 2021).
- X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. 39th International Conference on Very Large Databases (VLDB 2013).
- El Kindi Rezig*: Data Cleaning in the Era of Data Science: Challenges and Opportunities.* 11th Conference on Innovative Data Systems Research, (CIDR 2021).
- El Kindi Rezig, Lei Cao, Michael Stonebraker, Giovanni Simonini, Wenbo Tao, Samuel Madden, Mourad Ouzzani, Nan Tang, Ahmed K. Elmagarmid: *Data Civilizer 2.0: A Holistic Framework for Data Preparation and Analytics*. 45th International Conference on Very Large Databases (VLDB 2019). Note: Data Civilizer is the backbone of the popular tool *Trifacta (https://www.trifacta.com/)*.

Entity Resolution

Once data is integrated and crossed from different sources, we have the problem of identifying if two instances from two different data sets (or even the same data set) might be the same. For example, suppose two sessions browsing an e-commerce platform. It might be interesting to identify all sessions of the same person even if this person is not logged in and, depending on the information logged, it might not be straightforward to identify such sessions. Entity resolution (aka entity matching) indeed aims at identifying different descriptions (e.g., two instances) that refer to the same real-world entity appearing either within or across data sources, when unique entity identifiers are not available.

The relevance of entity resolution for Data Science is huge. We require processes that are able to provide hints whether two different instances from two different data sets refer to the same real-world entity. Realize the impact of this when performing data analysis: a data model might be biased if a certain event appears many times as input or you may lose the big picture related to a real-world event if you fail to detect all data pieces related to it. For this reason, entity resolution nowadays transcends its relevance beyond the analysis and it is also essential for organizing data in Data Lakes into categories.

In the past, entity resolution was tackled as duplicate detection within a database. But with the arrival of Big Data and Data Science, current approaches were challenged and felt obsolete. Similar to the previous cases, the current wealth of data available forces us to look for automatable approaches reducing the impact of this problem in Data Science projects.

By definition, entity resolution has been tackled as a probabilistic problem. Further, given the inherent computational complexity (comparing all instances against all other instances is quadratic but in front of very large databases this cost is not attainable) many techniques trying to reduce the search space were introduced. The most popular, most probably, is blocking, which blocks similar instances (e.g., according to a function) and performs a deep comparison only among similar instances.

Besides blocking, the other usual assumption that entity resolution techniques make is the definition of a similarity score. This is basically the core of the approach and it is a function (or even a complex learned model) that tells us if two descriptions resemble each other. Usually, the output is a confidence score.

Of course, entity resolution might be even more difficult with the presence of dirty data and, in a way, finding duplicates is a cleaning task. For that reason, this is typically a process carried out after (or iteratively intertwined with) cleaning data and, at the same time, considered to be part of Data Quality. Nevertheless, this specific problem is huge enough to be researched on its own.

Last, but not least, a usual question you may raise is the difference between entity resolution and data discovery. Intuitively, you can think of entity resolution as looking for duplicates, while data discovery aims at identifying *unionable* and *joinable* datasets.

All in all, entity resolution is "*the problem to identify, from one or several data sets, the instances referring to the same real-world entity*".

Seminal bibliography:

- G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas. *Comparative analysis of approximate blocking techniques for entity resolution*. 42th International Conference on Very Large Databases (VLDB 2016).
- T. Papenbrock, A. Heise and F. Naumann, *Progressive Duplicate Detection.* IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 5 (2015).
- V. Rastogi, N. N. Dalvi, and M. N. Garofalakis. *Large-scale collective entity matching*. 37th International Conference on Very Large Databases (VLDB 2011).