A model-driven approach for the definition of reproducible and replicable data analysis projects

Fran Melchor,[0000−0002−9383−3583], Roberto Rodriguez-Echeverria[0000−0002−6545−0913], and José M. Conejero[0000−0003−2640−679X]

INTIA, Universidad de Extremadura, Cáceres, Spain
{fmelchor,rre,chemacm}@unex.es

Abstract. It is becoming increasingly common to exploit the data collected by Information Systems in order to carry out an analysis of them and obtain conclusions that give rise to a series of decisions in the different research fields. The fact that in most cases these conclusions cannot be properly backed up has given rise to a reproducibility crisis in Data Science, the discipline that makes it possible to convert such data into knowledge, and in research fields that apply it. In this paper we envision a Model-Driven framework to foster reproducible and replicable Data Science projects. The framework proposes the definition of systematic pipelines that may be (semi)automatically executed in terms of concrete implementation platforms. Proprietary or third party tools are also considered so that flexibility may be ensured without hindering.

Keywords: reproducibility · replicability · process · data science · model-driven engineering.

1 Introduction

The rapid emergence and democratization of Big Data platforms, together with the increase of data generated in our daily lives, have fostered the appearance of data analysis studies that explore data to extract knowledge from them. These analyses often require tasks such as data acquisition activities, data integration, and visualizations, pattern identifications, or the identification of evidence to prove patterns reliability. However, less attention has been paid to make all these tasks reproducible and replicable arising a well-known problem that has been termed in many fields as the reproducibility crisis [9]. The reasons why reproducibility and replicability (R&R) is still a problem in many data analyses are several. As explained in [17], there is no agreement among authors about the origin of the reproducibility problem in data science. On the one hand, some authors argue that this is an infrastructure problem [14]. On the other hand, other authors refer to failed communication as being the seed for this situation [10]. The infrastructure problem would include accessing data or code and computer platforms dependence as some of the most critical barriers for reproducibility.

However, the current development of open access repositories or cloud computing solutions makes these problems affordable. Concerning communication, the need for improving analysis documentation, tracing information, or process explanation arrives as the main issues for achieving real reproducible studies.

This quick increase of Data Science usage for the development of projects in many fields (such as education, medicine, or psychology), together with the aforementioned lack of R&R for many of these projects [6] has encouraged us to define the framework that is presented here.

The framework relies on the separation of the pipeline definition into two different modelling layers: conceptual, where the data scientist may specify all the data and models operations to be carried out by the pipeline; operational, where the data engineer may describe the execution environment details where the operations (defined in the conceptual part) will be implemented. Based on this abstract definition and layers separation, the approach allows: the usage of different tools improving, thus, process replicability; the automation of the process execution, enhancing process reproducibility.

The rest of the paper is organized as follows Section 2 describes the main concepts of our approach and main issues of reproducibility and replicability. Section 3 describes our conceptual framework. Section 3 introduces an illustrative example. In Section 5 related works are presented. Section 6 draws some preliminary conclusions and outlines future work.

2 Background

2.1 Main concepts

In order to make the paper self-content, we now proceed to describe the main concepts related to the framework presented, namely reproducibility and replicability. These terms have been defined in different ways in the literature. Examples of these definitions, sometimes contradictory, may be found in [11]. However, this section provides the definitions according to the way they were used in our framework. In particular in [8]:

- **Reproducibility** in an experiment refers to the action of obtaining consistent results using the same input data; computational steps, methods, and code; and conditions of analysis.
- **Replicability** refers to the action of obtaining consistent results across studies aimed at answering the same scientific question, each of which has obtained its own data.

2.2 Main issuees for R&R

Manual ad hoc pipelines (MAP) defined on top of a variety of tools and technologies. Pipelines' stages are not always the same, so flexibility is mandatory. Each stage (or partial stage) could be implemented by a different tool increasing the complexity of its execution and monitoring. Flexibility and heterogeneity are

relevant properties of this kind of projects, but we need to manage them in a systematic way in order to make the process measurable and traceable to satisfy R&R.

Proof-and-error processes (PEP) are commonly followed by exploring different alternatives along the pipeline. Versioning code and data is possible, but doing it systematically is a different challenge. Moreover, versioning decisions and tool usage also remains a challenge. The final pipeline becomes then a particular path (along particular versions of stages) inside the exploration tree organically generated in the process of its definition.

Traceability (TR) can easily become complex. How can we trace a column of the modeling stage back to the original data? How can we navigate back the sequence of decisions made to obtain a particular column? How do we trace the different operations applied over data when using different tools in the pipeline?

In this context, several rules [12], [14] have been defined in the literature in order to specify basic guidelines to follow in the development of reproducible and replicable data analysis projects. Taken those works as reference, we elaborated a distilled collection of those rules, shown in Table 1. Three final columns also show whether the guideline contributes to mitigate the problem represented in the corresponding column.

Table 1. Ten rules and our framework comparison

Rule	Desc.	MAP	PEP	TR
01	For every result, keep track of how it was produced			✓
02	Avoid manual data manipulation steps	✓		
03	Archive the exact versions of all external programs used	✓		
04	Version control all custom scripts		✓	
05	Record all intermediate results, when possible in standardized formats		✓	✓
06	For analyses that include randomness, note underlying random seeds			
07	Always store raw data behind plots		✓	
08	Generate hierarchical analysis output, allowing layers of increasing detail to be inspected			✓
09	Connect textual statements to underlying results			✓
10	Provide public access to scripts, runs, and results		✓	
11	Data Version Control		✓	

However, although these rules are a good starting point for defining systematic pipelines, they are too high level without the necessary detail to specify how to satisfy them. The next section introduces our Model-Driven framework where we describe a potential solution for the definition of these pipelines.

3 Model-Driven Data Science Projects R&R

As shown in Figure1, the core of our Model-Driven Data Science Projects R&R (MD4DSPRR) approach is composed of two different levels: the conceptual and operational.

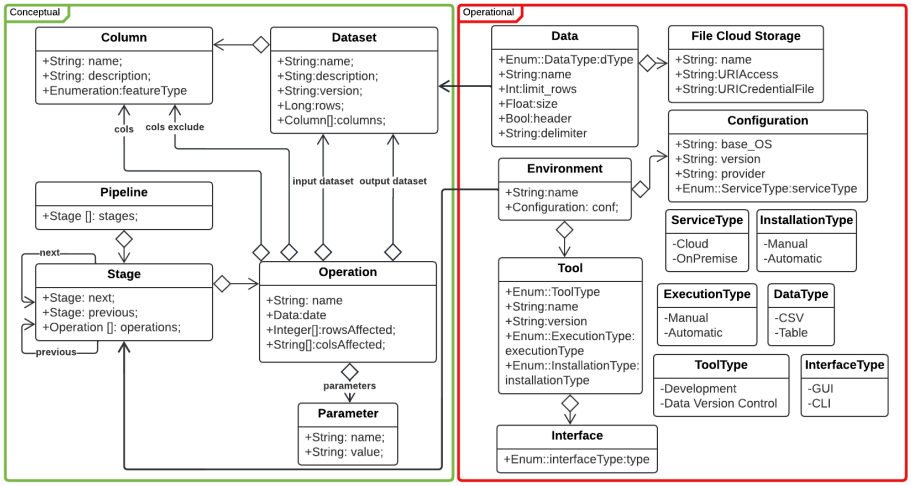

Fig. 1. MD4DSPRR approach overview

The conceptual level allows the description of data science projects in a platform independent way. This conceptual part would be ideally used by data scientists who describe the pipeline without implementation details. The conceptual part relies on a metamodel that introduces the different concepts needed to model any pipeline (pipeline's steps and connection between them). Based on this high level description, the data scientists could, among others, i) check the pipeline validity, e.g. based on a source dataset and a particular objective, are the proper columns and machine learning (ML) algorithms selected?; ii) compare different pipelines; iii) reuse pipelines, totally or partially (iv) version control pipeline alternatives and the involved data, (v) better collaborate in pipeline definitions, among others.

Regarding the operational part of the framework, it is conceived as an extension of the conceptual one where the concrete implementation details are described. Among the aspects specified in this part, we include the environment characteristics and its configuration details. In other words, the operational part is responsible for specifying the service type (cloud or on premise), the system infrastructure (operating system, version, provider, etc.), tool for develop and execute the pipeline specified on conceptual level. Note that the right description and proper configuration of the operational part of the pipeline would be a data

engineer responsibility. The operational part extends the conceptual one providing the implementation details that would be needed for the (semi)automatic pipeline execution. In other words, this part does not describe the system, conversely it specifies how its execution will be. Due to the flexibility required by this kind of projects, the framework should allow the specification of different types of tools to be used: i) programming languages and their libraries (e.g. Python or R scripts); or ii) third party tools that are executed locally (e.g. KNIME). Thus, while in some cases the automation of the pipeline execution (including the logging and monitoring features) will be based on the generation of the local specification to be used in a particular programming language; in other cases, the logging and monitoring specifications could be only partially defined, since some third party tools could not allow logging all the intermediate results for an operations sequence. In this second case, the generated infrastructure will deal with these tools as black-box components that will be handled by the data engineer that provides the inputs and registers the outputs. Additionally, it is useful to be able to specify an uncertainty degree for these tool due to the lack of automation in these cases (i.e the execution and installation type and whether the interface is CLI or GUI).

This conceptual and operational separation, together with the unidirectional dependency from the operational to the conceptual part, will allow the definition of pipelines where the execution environment could be easily modified (e.g. for comparing the results obtained by different tool combinations or pipeline verification purpose). Moreover, the definition of platform-agnostic pipelines improves separation of concerns since the different roles in teams involved in data projects (e.g. data scientist and engineer) could focus on their specific tasks, as MLOps approaches claim [1,5,15]. Additionally, based on this two levels description and by using Model Transformation techniques, a software engineer could generate all the infrastructure code needed for the (semi)automatic pipeline execution (including, again, logging and monitoring features).

3.1 Conceptual components

The main components of the conceptual layer are: pipeline, stage, operation, dataset and column. In essence a pipeline could be seen as a sequence of data operations included in each stage distilling the original datasets to extract the most significant features for the project goal (e.g. stock price prediction), which is eventually used to train a ML/DL model to derive the final model for execution. Note that the pipeline sets an established order between the stages, so that the output of each stage will be used as input of the subsequent one.

Stages allow the definition of the path to be followed in order to complete the data science project. A stage defines a particular mission (data fetching, ETL, data integration, etc.). Each stage consumes an input dataset and produces an output one. Stages are also defined as a sequence of data operations (as a workflow) and they must specify previous and subsequent operations.

Operations are defined on an input dataset and generate an output (temporal or permanent) dataset. In addition to the corresponding input and out-

put dataset, through the cols and cols exclude relationship we define on which columns the operation is applied (cols) or on which columns the operation is not applied (cols exclude). Those data operations can accomplish different missions, e.g. loading, cleansing, etc., and can deal with values (rows) or schemas (columns). Nevertheless, we think we can arrange them into three fundamental categories: filter, mapping and transformation (not represented in the diagram of Figure 1 for the sake of simplicity). A filter allows defining criteria (conditions on values) to get subsets of a particular dataset, e.g. removing all the rows with blank values in a particular column. A mapping defines a relationship between a set of input columns from the input dataset and a set of output columns.

We can further categorize them according to its cardinality into: one-to-one, one-to-many, many-to-one, and many-to-many. A transformation defines a real mutant operation, so the original data is transformed somehow to get the desired output data. They can be defined at row level, e.g. delete all the rows satisfying a criteria (filter), or at column level, e.g. round real values to their nearest integer. Obviously, they can be applied together, e.g. a mapping might entail a transformation (the output value is the average of several input values).

Regarding data operations scope, different stages may require different operations, e.g. it might be not useful to have a statistical operation in an ETL stage. Therefore data operation scope needs to be specified. For example, data operations can be labelled to constraint their application to particular types of stages, so editing tools can assist the engineer in the definition process. Moreover, operation scoping could allow checking the pipeline before its final implementation by applying model checking techniques and tools.

Stages (and consequently operations) may be later implemented by using different tools. However, to this purpose, the mission, inputs and outputs for each stage should be clearly specified. In particular, the metadata that will be stored for each operation will include at least: stage, input data, output data, date, row/cols affected, execution tool metadata, etc. Moreover, it needs to contain previous and next operations.

3.2 Operational components

The main components of the operational layer are: configuration, environment and tool.

Environments allow specifying concrete execution machines and their baseline configuration: operative system and its version, provider (e.g. Vagrant, Docker). Two main different execution environments could be considered through its service type: local (on premises) and cloud.

Tools allow specifying the different types of tool on a data science project (this could be extended in the future e.g. data visualization tools), name of tool, version, execution and installation type and the interface needed to be able to run the tools.

Tools, as aforementioned, could be programming languages of third-party tools specifically designed to accomplish a particular task in a data pipeline. In this context, a task would be a complete stage or a particular sequence of

data operations. In order to provide flexibility but keeping the maximum level of automation and traceability, we consider different types of tools according to their possible automation and external configuration. The main objective is to maintain the necessary information to be able to generate the execution and monitoring infrastructure as complete and automatic as possible.

3.3 Libraries

Both models (conceptual and operational) can be created from scratch, thus requiring a huge modeling effort. Moreover, a great number of elements in those models can become redundant, e.g. each data operation only needs to be defined once, and a similar situation is faced by tools and environments in the operational model. For this reason, we expect to collect those elements in reusable libraries.

As a first step, several known data mining tools (RapidMiner[1], KNIME[2], Orange Data Mining[3]) were thoroughly analyzed to collect their main data operations and map them into the operations defined in our framework[4]. A small excerpt of the results obtained is shown in table 2. Note that the existence of each operation in the PMML[5] standard version 4.2 is also indicated.

Table 2. Data operations mapping.

RapidMiner	KNIME	Orange	PMML	MD4DSPRR
Nominal to Numerical	One to Many	Continuize	Yes	Get Dummy Variables
Remove Attribute Range	Column Filter	Select Columns	Yes	Remove Columns
SMOTE Upsampling	SMOTE	–	No	SMOTE
Split Data	Partitioning	Data Sampler	No	Split Data
Gradient Boosted	GB Trees Learner	Gradient Boosting	Yes	Gradient Boosting (Train)
Apply Model	GB Trees Predictor	Predictions	Yes	Gradient Boosting (Predict)
Performance	Scorer	Test & Score	No	Performance Measures

Following, we provide a brief description of those operations. Get Dummy Variables transforms categorical columns into numerical ones (i.e. one-hot encoding). Remove Columns deletes a set of columns. The SMOTE algorithm oversamples the dataset (i.e. adds artifical rows), adding instances of the minority class. Split Data splits the input dataset into two partitions (i.e. row-wise). Gradient Boosting (Train) represents the training of a ML model using the Gradient Boosting algorithm. Gradient Boosting (Predict) represents the execution of a ML model. Performance Measures calculates the most relevant metrics for a classification model (e.g. confusion matrix, accuracy, recall, etc.) by comparing the prediction made with the real value.

[1] https://rapidminer.com/
[2] https://www.knime.com/
[3] https://orangedatamining.com/
[4] `https://bit.ly/3FXDbp5`
[5] http://dmg.org/pmml/pmml-v4-2-1.html

4 Illustrative Example

Next, an illustrative example is presented to better convey how the previously defined metamodels can be used to specify data science projects. This example was excerpted from a previous project on academic dropout prediction developed by our team [3]. The primary objective of this project was to predict whether an engineering student is going to drop out of her degree at the end of each semester, so correction actions may be triggered to reduce the existing high-rate dropout in technical careers. For this purpose, a model was trained from academic and personal data. Concretely, an ensemble model was applied integrating the results of three different algorithms: Gradient Boosting, Random Forest and Support Vector Machines.

For the sake of brevity, here we present the pipeline fragment for the creation of the Gradient Boosting learning model as an illustrative example, because its reduced size and representative operations flow. This flow contains (i) operations involved in the learning model generation (training, testing and getting the performance metrics), (ii) common data processing operations, and (iii) feature engineering operations. More information and materials are available at the project repository[6].

4.1 Conceptual Model

The input dataset structure is presented in Table 3. Each row shows the data needed to define each column of the dataset with our conceptual DSML. Two additional columns present a brief description and some example values for each column respectively.

Table 3. Dataset Definition

Columns	Data Types	ID	Target	Description	Examples
degree_name	categorical	0	0	Name of degree	Computer Science
call_year	categorical	0	0	Year of the call for Access	2007-08, 2008-09
call	categorical	0	0	Call for Access	June, September
access_description	categorical	0	0	Access Type Description	Exam, Transferred
gender	categorical	0	0	Gender of student	H, M
interval_year_birth	categorical	0	0	Year of Birth divided into 5-year intervals	(1990-1995], (200-2005]
scholarship	categorical	0	0	Indicate if the student has a scholarship	N, S
cum_pass_ratio	float	0	0	Ratio of subjects passed per taken	0,5; 0,7
marks	float	0	0	Mark obtained in University Entrance Exam (0 - 14)	10, 11, 9
cum_absent_ratio	float	0	0	Ratio of subjects not presented to the exam per taken	0,8; 0,9
record_id	categorical	1	0	Student ID by degree	1, 5, 105
degree_id	categorical	1	0	Unique Degree Identifier	233, 1623, 1627
dropout	categorical	0	1	Indicates if the student has dropped out	0, 1

Once the input dataset has been specified according to our DSML (not shown here), we proceed to define the data pipeline, presented in Fig. 2. Note that an ad-hoc concrete syntax (described in the legend) is used to better illustrate this example. Here we just explain the parameters and cols of the most relevant operations previously described in section 3.3.

[6] `https://github.com/i3uex/education_drop`

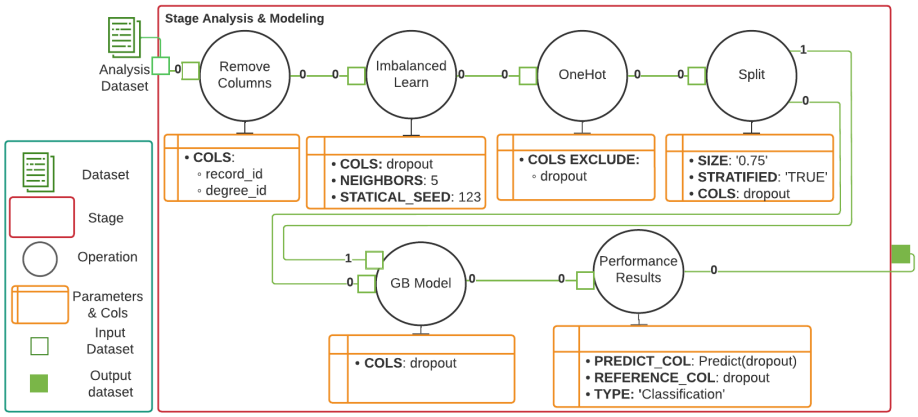

Fig. 2. Conceptual Model

In IMBALANCED LEARN, COLS refers to the categorical column to generate new cases (dropout column in our dataset), while NEIGHBORS and STATISTICAL_SEED are internal parameters of the algorithm.

In ONEHOT, COLS EXCLUDE indicates that the OneHot operation, will be applied on all columns of the input dataset except to dropout column.

In SPLIT, the SIZE parameter indicates the percentage of rows for the first partition. In this case it is 0.25 because we want to use the 75% of the input rows for training and the remaining 25% for testing. The RANDOM STATE parameter indicates a fixed random seed for the split. The STRATIFIED parameter is set to true to indicate that the distribution of the values in the COL column should be maintained in the partitions. Finally, this operation returns the first partition on output dataset 0 (training) and the second partition on output dataset 1 (testing).

In GB (GRADIENT BOOSTING), the TARGET_COL parameter indicates that dropout values will be later predicted by the ML model. This operation returns the results predicted by the trained ML model of testing input dataset, to use them as input of the PERFORMANCE RESULTS operation, that allow us know the performance metrics of trained ML model.

4.2 Operational Model

For illustrative purposes, two different execution environments were defined in the operational model. Fig. 3 presents the commonalities of both execution environments, while Fig. 4 shows their specificities. Fig. 3 also illustrates the binding between both environments and the Analysis & Modeling stage.

The information system used for data storage was a Cloud File System, so its URI (uriAccess) and credentials file (credentialFileUri) are specified. CSV data files were used for both input and output with the same configuration: header is True and delimiter is '|'.

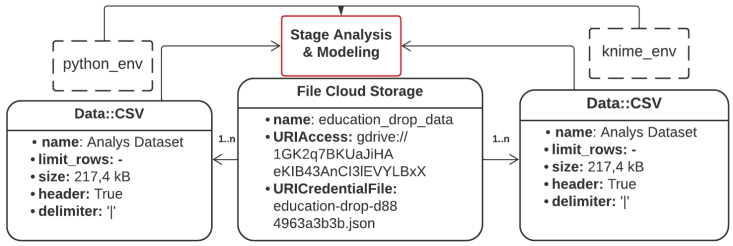

Fig. 3. Operational Model

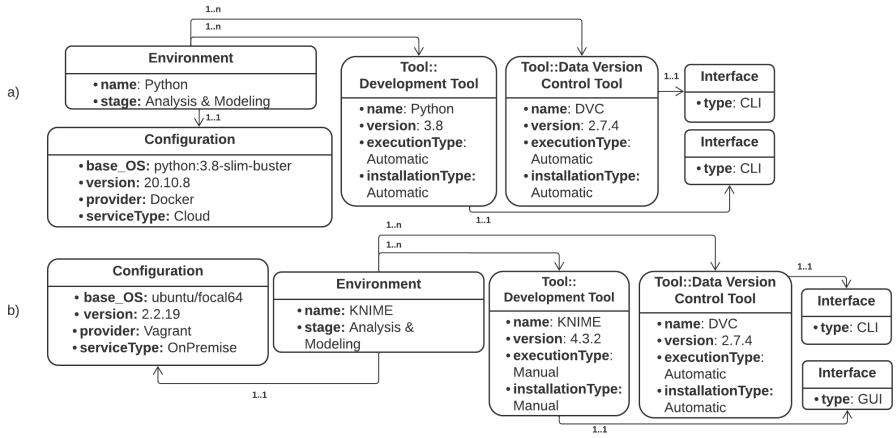

Fig. 4. Execution Environment Details

The Python environment uses a Docker provider with a Linux OS image. Python[7] is selected as development tool with automatic installation and execution (preinstalled in the OS image) and CLI as its interface. DVC[8] is the data versioning control tool selected, with automatic installation and execution also (in both environments). Its DataStore configuration is shown in Fig. 3. The KNIME environment uses a Vagrant provider preinstalled with a particular distribution of Linux OS. KNIME[9] is the development tool, which requires manual installation, because no package is available, and also manual execution through its GUI.

With the definition of the conceptual model, defined in the previous section, and this definition of the operational model, all the necessary infrastructure could be generated to execute the defined pipeline through M2T transforma-

[7] https://www.python.org/
[8] https://dvc.org/
[9] https://www.knime.com/

tions. Although this still remains an open issue for our approach, we are already working on providing these transformations.

5 Related Works

First of all, it is worth mentioning that the work in [7] reviews eleven applications focused on assisting researchers in publishing reproducible research, including executable code and data. Among them, the REANA platform [19] aims at facilitating reproducible science practices using container technologies for preserving and reinstantiating runtime environments. Basically, a cluster composed of a set of micro-services permit to instantiate, launch and monitor container-based computational workflows on remote compute clouds. On the other hand, ReproZip [13] not only provides reproducibility but also basic replicability by allowing the substitution of datasets if researchers have previously ensured compatibility. Basically, it allows encapsulating data, code, and the computational environment. In this case, researchers can choose either execute the result on a ReproZip server or locally without tracking down and installing the dependencies.

Specifically focused on replicability, MORF [4] is an open-source platform which provides a Python API for predictive modeling or production rule analysis in education. It also uses Docker for reproducibility. Moreover, DataDeps.jl [16] is a julia package that facilitates to automate the data setup step in data driven research. It allows joining code and data, so that different projects can be reproduced and replicated avoiding human errors in this part of a Data Science process. It is similar to one part of our approach, but in a simplest way.

Finally, focusing on reproducibility and reusability, ML-flow[18] allows ensuring both aspects by offering itself as an API available for Python, Java and R languages. Researchers can define a pipeline using a YAML specification that can be reproduced managing dependencies through Conda or Docker. Subsequently ml-experiment[2] integrates it into a Python Framework to create reproducible Data Science projects. This framework shares with our approach features such as the use of containers (Docker) or the use of Stages to encapsulate code.

In general, we have some commonalities with some of the above works. However, first, our approach is technology independent, second, it makes an explicit separation between data scientist and data engineering roles and responsibilites, and third, it raises the level of abstraction for verification, comparison and reusability of data science artifacts.

6 Conclusions

This paper has presented a conceptual framework for the definition of systematic Data Science processes that may be (semi)automatically executed, enhancing, thus, R&R in these projects. The framework is based on modelling these projects at two different layers or abstraction levels: logical and physical. While the former allows data scientist to focus just on the conceptual pipeline definition for

the project, the latter may be used by data engineers to provide the execution environment and platform concrete details to generate the final implementation for the project. The tool-agnostic pipeline definition ensures that a particular project could later have different implementations by using different platforms. This would not only ensure pipeline adaptability but also the possibility of comparing results obtained by different tools.

As further work, we envision the next research lines to follow. R&R verification at model level. Logical and physical models provide enough information to define specific (and reusable) R&R constraints. For instance, a concrete execution environment (e.g. a Linux PC with KNIME) may be adequate to execute its associated pipeline for a small volume of data, however it turns out unresponsive when the expected volume of data grows significantly. Thorough analysis of related notations and languages. We have defined our own metamodels as a mean to explore the domain and elicitate the main concepts and their relationships. However, existing languages and notations from different domains may be used for logical and physical models. Furthermore, the formal specification of data operations could be useful to verify, compare or simulate pipeline execution.

References

1. Byrne, C.: Development Workflows for Data Scientists. O'Reilly Media, Inc. (2017)
2. Domenech, A.M., Guillén, A.: ml-experiment: A Python framework for reproducible data science. Journal of Physics: Conference Series **1603**(1), 012025 (sep 2020). https://doi.org/10.1088/1742-6596/1603/1/012025
3. Fernández-García, A.J., Preciado, J.C., Melchor, F., Rodriguez-Echeverria, R., Conejero, J.M., Sánchez-Figueroa, F.: A real-life machine learning experience for predicting university dropout at different stages using academic data. IEEE Access **9**, 133076–133090 (2021)
4. Gardner, J., Brooks, C., Andres, J.M., Baker, R.S.: Morf: A framework for predictive modeling and replication at scale with privacy-restricted mooc data. Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018 pp. 3235–3244 (1 2019). https://doi.org/10.1109/BIGDATA.2018.8621874
5. van den Heuvel, W.J., Tamburri, D.A.: Model-Driven ML-Ops for Intelligent Enterprise Applications: Vision, Approaches and Challenges. Lecture Notes in Business Information Processing **391 LNBIP**, 169–181 (jul 2020). https://doi.org/10.1007/978-3-030-52306-0_11
6. Ioannidis, J.P.A.: Why Most Published Research Findings Are False. PLoS Medicine **2**(8), e124 (aug 2005). https://doi.org/10.1371/journal.pmed.0020124
7. Konkol, M., Nüst, D., Goulier, L.: Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication. Research Integrity and Peer Review **5**, 1–8 (12 2020). https://doi.org/10.1186/S41073-020-00095-Y/TABLES/2
8. National Academies of Sciences, Engineering, and Medicine: Reproducibility and Replicability in Science. The National Academies Press, Washington, DC (2019). https://doi.org/10.17226/25303
9. Peng, R.: The reproducibility crisis in science: A statistical counterattack. Significance **12**(3), 30–32 (jun 2015). https://doi.org/10.1111/j.1740-9713.2015.00827.x

10. Peng, R.: Disseminating reproducible research is fundamentally a language and communication problem (2016), https://simplystatistics.org/2016/05/13/reproducible-research-language/
11. Plesser, H.E.: Reproducibility vs. replicability: a brief history of a confused terminology. Frontiers in neuroinformatics **11**, 76 (2018)
12. Sandve, G.K., Nekrutenko, A., Taylor, J., Hovig, E.: Ten Simple Rules for Reproducible Computational Research. PLoS Computational Biology **9**(10), e1003285 (oct 2013). https://doi.org/10.1371/journal.pcbi.1003285
13. Steeves, V., Rampin, R., Chirigati, F.: Using reprozip for reproducibility and library services. IASSIST Quarterly **42**, 14–14 (12 2018). https://doi.org/10.29173/IQ18
14. Stodden, V., Miguez, S.: Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. Journal of Open Research Software **2**(1) (2014)
15. Treveil, M., Omont, N., Stenac, C., Lefevre, K., Phan, D., Zentici, J., Lavoillotte, A., Miyazaki, M., Heidmann, L.: Introducing MLOps: How to Scale Machine Learning in the Enterprise. O'Reilly Media, Inc. (2021), https://www.oreilly.com/library/view/introducing-mlops/9781492083283/
16. White, L., Togneri, R., Liu, W., Bennamoun, M.: DataDeps.jl: Repeatable Data Setup for Reproducible Data Science. Journal of Open Research Software **7**(1), 33 (oct 2019). https://doi.org/10.5334/jors.244
17. Yu, B., Hu, X.: Toward Training and Assessing Reproducible Data Analysis in Data Science Education. Data Intelligence **1**(4), 381–392 (2019). https://doi.org/https://doi.org/10.1162/dint_a_00053
18. Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S.A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., Zumar, C.: Accelerating the machine learning lifecycle with mlflow. IEEE Data Engineering Bulletin **41**, 39–45 (2018), https://www-cs.stanford.edu/people/matei/papers/2018/ieee_mlflow.pdf
19. Šimko, T., Heinrich, L., Hirvonsalo, H., Kousidis, D., Rodríguez, D.: Reana: A system for reusable research data analyses. EPJ Web of Conferences **214**, 06034 (9 2019). https://doi.org/10.1051/epjconf/201921406034