# Towards a conceptual framework for the specification of reproducible and replicable data analysis projects

***Roberto Rodriguez-Echeverria***
*University of Extremadura*
*Cáceres, Spain* *rre@unex.es*

***José M. Conejero***
*University of Extremadura*
*Cáceres, Spain* *chemacm@unex.es*

***Fran Melchor***
*University of Extremadura*
*Cáceres, Spain* *frmelchorg@unex.es*

***Juan D. Gutiérrez***
*University of Extremadura*
*Cáceres, Spain* *andy@unex.es*

***Álvaro E. Prieto***
*University of Extremadura*
*Cáceres, Spain* *aeprieto@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 it research fields that apply it. In this paper we envision a conceptual 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 reproducibility and replicability.

**Keywords:** reproducibility, replicability, process, data science, framework

## 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 projects exploring data to extract knowledge from them. However, little attention has been paid to make all these tasks reproducible and replicable, hence arising a well-known problem termed in many fields as the reproducibility crisis [7].

According to [6], **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 whilst **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. Although there could be found different reasons explaining why many data analysis projects are continuously struggling with reproducibility and replicability (R&R) [1], [4], the infrastructure problem is considered one major factor[10]. The infrastructure problem would include barriers as accessing data or code and computer platforms dependence, among others.

In this work we envision the necessity of a conceptual framework and its associated execution and monitoring infrastructure. The main goal of the conceptual framework would be to free data scientist from the incidental complexity introduced by the technological stack selected for the project, while providing data engineers with the proper methods and tools to automatically derive an execution infrastructure ensuring R&R.

The rest of the paper is organized as follows. Section 2 illustrates the main motivation problems. Section 3 describes our conceptual framework. Section 4 summarizes other approaches. Finally, Section 5 draws some preliminary conclusions and outlines future work.

## 2. Main issues 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 feature of the modeling stage back to the original data? How can we navigate back the sequence of decisions made to obtain a particular feature? How do we trace the different operations applied over data when using different tools in the pipeline?

**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 | | ✓ | |

In this context, several rules [8], [10] 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. 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 conceptual framework where we describe a potential solution for the definition of these pipelines.

## 3. Conceptual Framework

As shown in Figure 1, our framework is compound by two different levels: logical and physical.

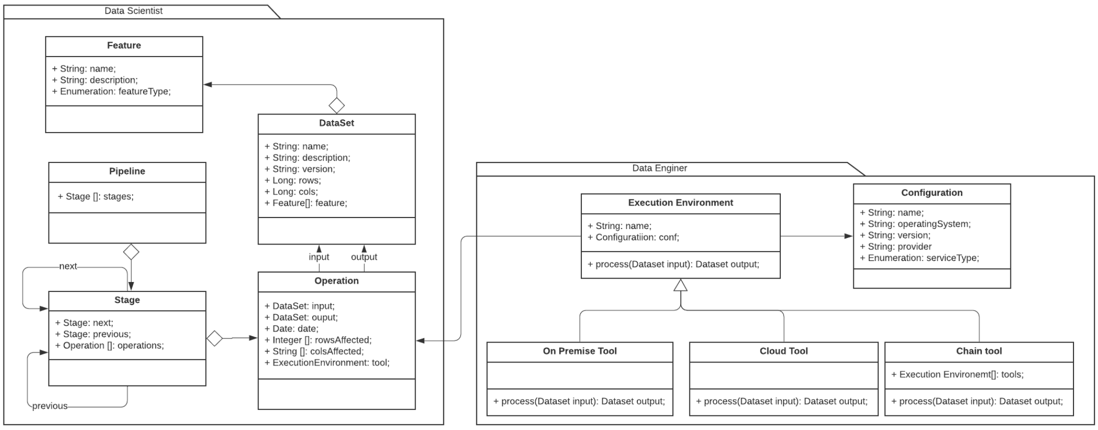

**Fig. 1.** Conceptual Framework

The logical level allows the description of data science projects in a platform independent way. This logical part would be ideally used by data scientists who describe the pipeline without implementation details. The logical 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 features and ML algorithms selected?; ii) compare different pipelines; iii) reuse pipelines, totally or partially.

Regarding the physical part of the framework, it is conceived as an extension of the logical 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 physical part is responsible for specifying the machine type (local, cloud or container), the system infrastructure (operating system, version, base configuration, etc.), the code and data version control systems configuration, the library versions, tools information, logging system, and so on. Note that the right description and proper configuration of the physical part of the pipeline would be a data engineer responsibility. The physical part extends the logical 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) or in the cloud (e.g. Azure). 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 would be useful to be able to specify an uncertainty degree for these tool due to the lack of automation in these cases (e.g. could the tool import an external configuration or run from CLI, etc.).

This logical and physical separation, together with the unidirectional dependency from the physical to the logical 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). Moreover, 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). As an example, the code for registering all the changes performed to a dataset (by one or more data operations) in a particular stage. This code could even automatically upload the resulting dataset to the data versions system. This systematic specification of the pipeline and automatic generation of the environment configuration and execution will contribute to a significant improvement of R&R in data science projects.

### 3.1. Logical components

The main components of the logic layer are: pipeline, stage, operation, dataset and feature. 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. 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 attributes (columns) from the input dataset and a set of output attributes. 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. Physical components

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

Execution environments allow specifying concrete execution machines and their baseline configuration. Two main different execution environments could be considered: local (on premises) and cloud. In the local case, the configuration should mainly indicate the operating system, its specific version and the initial setup (users, permissions, network access, etc.). Meanwhile, in the cloud case, it could identify the provider and the kind of service provided. An execution environment provides a specific technological stack for a collection of tools that will be useful for the execution of a pipeline. And the execution of a pipeline could entail one or more execution environments.

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.

## 4. Related work

In the related literature, we can find some proposals for tools that try to contribute to the realization of reproducible data science projects, such as [5], [9], and [11]. [5] defines a framework for interactive record linking used in data science projects. [11] defines a package that enables data management in Data Science projects in a reproducible and replicable way. Finally, [9] is a platform that allows the reproduction of Data Science projects through a pipeline that collects everything necessary for the project to be and its dependencies, making use of tools such as Docker or Kubernetes to carry out the reproduction of the different projects. However, these tools do not define a systematic way to carry out a Data Science project to comply with reproducible and replicable characteristics as we do in our proposal. Nevertheless, instead, they define tools that contribute to the development of the same. It is challenging to find frameworks that define a way of carrying out a systematic data science project to be reproducible and replicable from start to finish (even more, regardless of the concrete tools used).

The following book [3] talks about defining a reproducible workflow for the development of Data Science projects and gives us the only example pending improvement, the following process proposed by BinaryEdge [2]. This process defines how to carry out a Data Science project, but it does not provide the separation between the two abstraction layers that we presented here.

As main conclusion, although some tools contribute partially to the reproducibility of data science projects, the definition of systematic automatized processes is still under development.

## 5. 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 be 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 plan to provide a complete pipeline definition and its concrete implementation based on Ansible execution environment and Python libraries, which will keep record of all the data operations by using a logging strategy where logs may be later parsed and queried.

## Acknowledgements

This work was carried out with the support of (i) Ministerio de Ciencia, Innovación y Universidades (MCIU), Agencia Estatal de Investigación (AEI), and European Regional Development Fund (ERDF): RTI2018-098652-B-I00 project, and (ii) European Regional Development Fund (ERDF) and Junta de Extremadura: IB18034, and GR18112 projects.

## References

1. Baker, M.: Reproducibility crisis. Nature. 533 (26), 353–366 (2016)
2. BinaryEdge: The Data Science Workflow, https://blog.binaryedge.io/2015/09/08/the-data-science-workflow. Accessed: April 14, 2021
3. Byrne, C.: Development Workflows for Data Scientists. O'Reilly Media, Inc. (2017)
4. Ioannidis, J.P.A.: Why Most Published Research Findings Are False. PLoS Med. 2 (8), e124 (2005)
5. Karim, M., Ramezani, M., Sunbury, T., Ohsfeldt, R., Kum, H.-C.: VIEW: a framework for organization level interactive record linkage to support reproducible data science. arXiv. (2021)
6. National Academy of Sciences: Reproducibility and Replicability in Science. The National Academies Press, Washington, DC (2019)
7. Peng, R.: The reproducibility crisis in science: A statistical counterattack. Significance. 12 (3), 30–32 (2015)
8. Sandve, G.K., Nekrutenko, A., Taylor, J., Hovig, E.: Ten Simple Rules for Reproducible Computational Research. PLoS Comput. Biol. 9 (10), e1003285 (2013)
9. 1. Šimko, T., Heinrich, L., Hirvonsalo, H., Kousidis, D., Rodríguez, D.: REANA: A System for Reusable Research Data Analyses. EPJ Web Conf. 214 6034 (2019)
10. Stodden, V., Miguez, S.: Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research. J. Open Res. Softw. 2 (1), (2014)
11. White, L., Togneri, R., Liu, W., Bennamoun, M.: DataDeps.jl: Repeatable Data Setup for Reproducible Data Science. J. Open Res. Softw. 7 (1), 33 (2019)