Reproducible analysis pipelines using containers and data exploration using R/Shiny

Universitat Oberta de Catalunya

UNIVERSITAT DE BARCELONA

Nombre Estudiante

Luis Morís Fernández

MU Bioinf. i Bioest.

Omics

Nombre Tutor/a de TF

Mireia Ferrer Almirall

Profesor/a responsable de la asignatura

David Merino Arranz

Fecha Entrega

15/01/202

FICHA DEL TRABAJO FINAL

Título del trabajo:	Reproducible analysis pipelines using containers and data exploration using R/Shiny
Nombre del autor:	Luis Morís Fernández
Nombre del consultor/a:	Mireia Ferrer Almirall
Nombre del PRA:	David Merino Arranz
Fecha de entrega (mm/aaaa):	01/2023
Titulación o programa:	MU Bioinf. i Bioest.
Área del Trabajo Final:	Omics
Idioma del trabajo:	Inglés
Palabras clave	reproducibility, pipeline, bioinformatics

Resumen del Trabajo

Although reproducibility is one of the cornerstones of scientific research, it is not necessarily guaranteed and in some cases it fails even at its most basic level. After publication, many studies are not able to reproduce the published results when reanalyzing the same datasets used in the publication. This problem is worsened as time elapses, researchers may forget the exact analysis steps and the published description of the analysis may not be as exhaustive as required to reproduce the analysis or the software used to create the results may change, be discontinued or disappear.

The aim of this work is to improve this most basic level of reproducibility by creating a reproducible microarray analysis pipeline using a workflow manager, *targets,* and the use of software containers, *Docker*. Additionally, an interactive web application is developed to explore the results of this microarray analysis pipeline.

Abstract

Aunque la reproducibilidad es una de las características fundamentales de la investigación científica, esta no está necesariamente garantizada y en algunas ocasiones esta reproducibilidad falla incluso a los niveles más básicos. Después de publicarse, muchos estudios no pueden reproducir los resultados publicados cuando se reanalizan los mismos datasets utilizados en la publicación. Este problema empeora con el paso del tiempo, los investigadores

pueden olvidarse de los pasos exactos del análisis y la descripción publicada no es tan detallada como sería necesario para reproducir los análisis o el software utilizado para calcular estos resultados puede cambiar, discontinuarse o no estar disponible.

El objetivo de este trabajo es mejorar la reproducibilidad al nivel más básico. Para ello se ha creado un *pipeline* de análisis para *microarrays* utilizando un *workflow manager*, *targets*, y contenedores de software, *Docker.* Además se ha desarrollado una aplicación interactiva para la exploración de los resultados de este *pipeline* de análisis.

Table of Contents

Table of Figures

Index of Code Blocks

Introduction

Context and justification of this work

Reproducibility is one of the cornerstones of the scientific method, nonetheless is one of the most commonly forgotten, sometimes shadowed by the novelty or alleged impact of the results (Ioannidis 2005; Errington, Denis, et al. 2021). With the advent of informatics more powerful and flexible analyses are available to researchers in all fields everyday (Gauthier et al. 2018). Unfortunately, this power and flexibility comes usually at the cost of complexity and length. Bioinformatics pipelines are paramount examples of the benefit of these powerful and flexible analyses and also of the cost of complexity (Brito et al. 2020; Ioannidis et al. 2009; Mangul et al. 2019; Markowetz 2015). This issue is not solved exclusively by properly storing the scripts of analysis and the data, additional considerations must be taken, such as the pathing, the version of the language used, the version of the libraries, the operating system and its version, etc (Wallach, Boyack, and Ioannidis 2018; Wratten, Wilm, and Göke 2021). All these problem hinders reproducibility and utmost care must be taken when preparing and storing these pipelines, so they can be run not only at the moment of the initial analysis but at any moment in the future when another researcher may need to verify the analyses, extend them or run them on new data.

This work will explore two possible ways of improving reproducibility when developing bioinformatic pipelines. The first one is making the analysis steps traceable from the start to end. This regards the actual scripts that will be executed during the analysis. To address this issue this work will take advantage of workflow creation tools that allows creating long reproducible analysis pipelines. The second is creating a reproducible light-weight stable environment were the previous pipeline will run by using containers.

The other objective of this work has to do with the interactivity of the reports obtained during the analysis process. It is rarely the case, particularly in clinical environments, that the same person has the ability to program and run analyses (e.g.: a bioinfomatics expert) and also to make decisions based on those data (e.g.: a medical doctor) (Ludt et al. 2022). This usually creates a communication overhead between the data expert and the decision-maker until the exact data needed to make the decision is found. This work will also create an interactive tool that will try to ameliorate this problem by providing the decision-maker with a small set of tools that would decrease this communication overhead.

General description

The intention of this work is to create a proof of concept that would address two problems commonly found when developing bioinformatics pipelines. The first one is how to improve its reproducibility through the use of workflow tools and containers. And the second one is to explore interactive tools as a mean to

reduce the overhead between bioinformatics and medical personnel improving the decision making loop.

General Objectives

Create a proof of concept that includes a traceable and reproducible pipeline through the use of workflow tools. This pipeline must run inside a container and produce an interactive report as a result[1].

Specific objectives

1. Describe the reproducibility problem in bioinformatics
2. Explore containers and workflow tools as a mean to improve the reproducibility of bioinformatics pipelines
3. Explore interactive tools as a mean to improve the decision making loop in clinical settings
4. Create a microarray analysis pipeline using containers that produce an interactive report as a result in a real setting

Approach and method

Firstly, this work will address the state of the art in reproducibility with particular emphasis in the bioinformatics field. Secondly, this work will explore the creation of a proof of concept microarrays analysis pipeline created in a workflow tool that will run inside a container that will produce, among other products, an interactive report. In this second part, this work will be supported by *Plataforma de Bioinformática de la Unidad de Estadística y Bioinformática (UEB)* for having access to data and creating a useful pipeline that could be used in a real clinical setting.

From a technical point of view, albeit other tools may be explored during this work:

- The `R (R Core Team 2022)` language will be used as the foundation for all the software produced in it.
- The `maUEB` package functions, currently under development by the *UEB*, will be used to implement the microarray analysis pipeline.
- The `targets (Landau 2021)` package will be used as the workflow tool to create the pipeline itself in combination with the `maUEB` functions.
- `R/Shiny (Chang et al. 2022)` framework will be used to create an interactive tool.
- `Docker (Merkel 2014)` container technology will be used for containerizing the pipeline.

While selecting the tools for this work the following considerations were kept:

1 As discussed later in the results section, there was a deviation from the original planification and the interactive report was replaced by an static report and an R/Shiny application.

- Minimize friction between components when developing the proof of concept. That is the reason why all tools are `R`-based, with the exception of `docker`.
- Use components easily installed and run inside a container.
- Favor analysis scripts over graphical user interfaces.
- Favor simplicity and closeness to the `R` language (Ousterhout 2018).

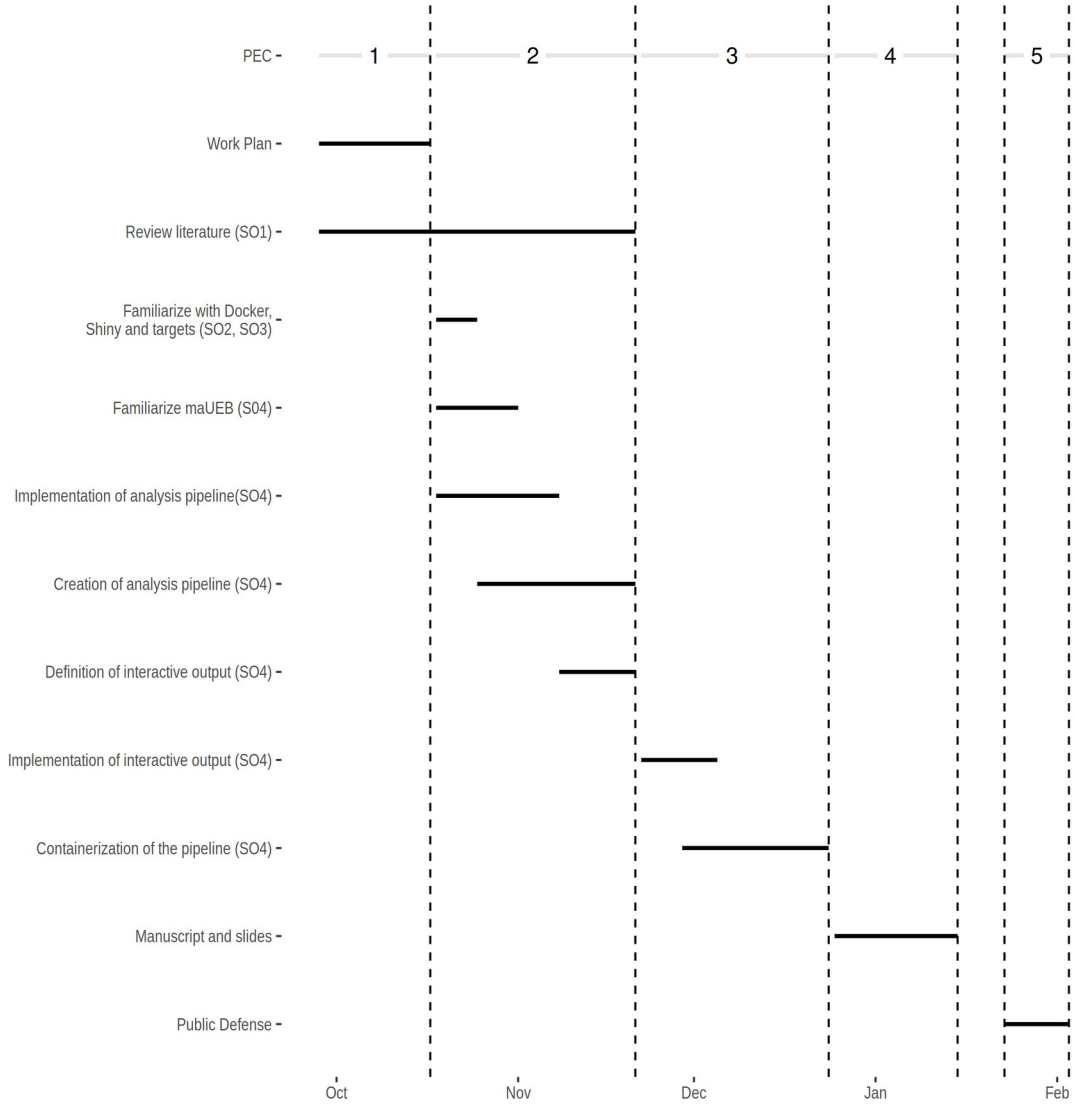
Planning

Tasks

Each task will be related to a specific objective using the acronym *SO*

1. Review literature for:
    1. **(SO1)** General problems of reproducibility in science and in bioinformatics
    1. **(SO1)** Current approaches to solve this problem
1. Software and technical knowledge and familiarization with:
    1. **(SO2, SO3)** Docker, `R/Shiny` and `targets` R package
    1. **(SO4)** `maUEB` package
2. **(SO2, SO4)** Definition and implementation of the analysis pipeline
3. **(SO3, SO4)** Definition and implementation of the interactive tool
4. **(SO2, SO4)** Containerization of the pipeline

Schedule



Milestones

1. Workplan **(17/07/2022)**
1. Literature review and state of the art
2. Creating a pipeline using `targets`
3. Defining the interactive output
4. PEC2 **(21/11/2022)**
5. Implementing interactive output
6. Containerize the pipeline
7. PEC3 **(24/12/2022)**
8. Manuscript and slides **(27/12/2022 - 15/01/2023)**
9. Public defense **(23/01/2023 - 03/02/2023)**

Risk analysis

Below are described present possible risks and proposed solutions.

1. Oversized scope. This work tries to address two well-differentiated problems, the reproducibility and the report interactivity. The interactivity scope can be reduced or removed from the work if the amount of time is not enough to cover both. The solution proposed is to reevaluate the scope when PEC2 is finalized and the scope is better defined.
2. Package availability. The initial intention is to use a recently developed package `maUEB`. If the package is not available other alternative packages covered during the master classes will be used.
3. Data availability. If data is unavailable, for example confidentiality for confidentiality reasons, open data sources will be used.
4. Pipeline definition. If the the pipeline is not possible to define in a real clinical setting, steps similar to those used in the other pipelines during the master classes will be used.

Expected results

1. Work plan
    1. A detailed workplan for the thesis work.
2. Thesis document
    1. A document containing an state of the art, methodology, results and conclusions obtained during the master's thesis.
3. A working proof of concept
    1. A containerized pipeline[2]
4. Slides and virtual presentation
    1. Slides and a video presentation for the defense.

CCEG

Although the work has a strong technical nature, its nature can be related to at least one *Sustainable Development Goals* (SDG) directly and another one which is transversal to many research works.

In first place, this work is directly related to *SDG 9 Build resilient infrastructure, promote sustainable industrialization and foster innovation* in particular to point *9.5 Enhance scientific research, upgrade the technological capabilities of industrial sectors in all countries(...)*. One of the core aspects of this work is to improve the reproducibility of research in the field of Bioinformatics. This reproducibility is valuable in itself as it improves the quality of the evidence produced, but its impact goes beyond that. Low quality evidence may impact the allocation of research resources, as researchers may pursue lines that may have been discarded in the light of more solid and reproducible evidence. In fact research indicates that low quality evidence, although not restricted to reproducibility, wastes billions of dollars each in year in the biomedical field (Macleod et al. 2014).

2 Due to the deviation mentioned in Footnote 1 and in the results section, and R package with an R/Shiny application is also delivered.

In second place, almost all research works can impact into *SDG 5: Achieve gender equality and empower all women and girls*. In particular, a common issue is that cites are not equally distributed between genders and men tends to cite other men more often (Chatterjee and Werner 2021). Therefore in this work particular caution will be placed on considering all works that may be referenced regardless of the gender of the author/s and citing using full names. Additionally, inclusive language will be used during the writing and the third person plural will be used to avoid making assumptions about gender, for example, of the possible users of the work produced (“Singular "They"” 2022). Although initially none is detected, this work will also consider possible biases or issues with this product beyond gender, regarding race, social position or level of technology access among others. One key consideration will be the open source nature of the work and the licensing of the software used to maximize its availability in institutions in developing countries (also related to *SDG 4 Quality education*).

State of the art

«To boldly go where no *hu*man has gone before!». This quote[3] from one of the most famous science fiction universes, Star Trek, could be used to describe scientific research as the strive to learn and advance human knowledge. Unfortunately, as appealing and exciting as this quote may be, a non-trivial amount of the scientific effort should be devoted to check and reproduce previous results. Therefore, the previous quote should be complemented with «and also to revisit and thoroughly check where other humans affirm they have gone before».

Reproducibility is one of the hallmarks of science, as described by Karl Popper in his book *The Logic of Scientific Discovery* (Popper 1959): *«Only when certain events recur in accordance with rules or regularities, as is the case with repeatable experiments, can our observations be tested — in principle — by anyone. We do not take even our own observations quite seriously, or accept them as scientific observations, until we have repeated and tested them. Only by such repetitions can we convince ourselves that we are not dealing with a mere isolated 'coincidence', but with events which, on account of their regularity and reproducibility, are in principle inter-subjectively testable.»* Nevertheless, despite its mention in fundamental works describing the scientific research method, reproducibility of scientific research is under discussion (Errington, Denis, et al. 2021; Ioannidis 2005; Munafò et al. 2017; Nelson, Simmons, and Simonsohn 2018).

As a particular example in the biomedical field the *Reproducibility Project: Cancer Biology (Errington, Denis, et al. 2021; Errington, Mathur, et al. 2021; Rodgers and Collings 2021)* was an initiative that tried to replicate high-profile papers in the field of cancer biology. This project found many hurdles that prevented the replication of many of the studies included in the project: failure to replicate the effects reported in the original studies, lack of descriptive and inferential statistics or lack of detail in the methodology.

Albeit, usually (as in Errington, Mathur, et al., 2021) the focus of reproducibility is the the replication of the results of one study by running an experiment equivalent to the one run in another laboratory, the reproducibility problem expands beyond that (e.g: re-creating the results by rerunning the same analysis on the same data set; replicating the results by using a similar experiment setup but a different team; replicating the results by using a different experiment setup and a different team). In fact, the reproducibility problem is so multifaceted that the terminology can be confusing. Several terms (reproducibility, replicability, repeatability,...) have different meanings depending on the context where they are used (Barba 2018). In this work we will focus on

3 The quote is slightly modified, as indicated by the italics.

what Barba (2018) defined as reproducible research according to the nomenclature used by Claerbout & Karrenbach (1992) Donoho et al.(2009) and Peng (2011): «*Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results».*

Recreating the results of a study given its data and its methods can be considered the most basic level of reproducibility, as it only assures that the results of a study were correctly calculated, but it does not delve in them being correct at an epistemological level. One could be tempted to think that this level of reproducibility is so trivial that could be considered granted but it is not (Hardwicke et al. 2018). In 1995, Buckheit & Donoho paraphrased Jon Claerbout (one of the pioneers in computational reproducibility Claerbout & Karrenbach, 1992) stating that an article should be considered an «*advertising*» and that the full research work is «*the complete software development environment and the complete set of instructions that generated the figures*». The initial solution proposed by Claerbout & Karrenbach (1992) involved using CD-ROMs and magnetic tapes as well as the ubiquitous *makefile* tool, luckily a lot have changed in those 30 years. In the present work it is illustrated how to use the *targets (Landau 2021)* R package and Docker (Merkel 2014) containers to ameliorate the reproducibility problem in the bioinformatics field.

Workflow manager systems: targets Package

Workflow managers[4] tools can now be found in many different flavors, from the most general like Apache Airflow, to those focused on scientific analysis like Scipipe and *targets*, to those specific to the bioinformatics domain such as Galaxy and KNIME (Amstutz et al. 2022). Wratten et al. (2021) reviewed the state of the art of workflow managers and compared key features of the most common ones in bioinformatics as depicted in Figure 1. *targets* package was not included in this review, but in this section the rationale followed for choosing it is described.

The author of *targets (Landau 2021)* describes it the following way:

«The `targets` package is a Make-like pipeline tool for Statistics and data science in R. With `targets`, you can maintain a reproducible workflow without repeating yourself. `targets` learns how your pipeline fits together, skips costly runtime for tasks that are already up to date, runs only the necessary computation, supports implicit parallel computing, abstracts files as R objects,

4 At the moment of writing this document the terms «pipelines» and «workflow managers» is difficult to disambiguate. For example, targets is described as a «pipeline-tool» while Scipipe is used for «Scientific workflows, sometimes also called "pipelines"». The definition used in Wratten et al. (2021) will be used in this document: «Workflow managers [...] simplify pipeline development [...]». Therefore, targets will be considered a Workflow manager and the product an analysis pipeline.

and shows tangible evidence that the results match the underlying code and data.»

Tool	Class	Ease of use[a]	Expressiveness[b]	Portability[c]	Scalability[d]	Learning resources[e]	Pipeline initiatives[f]
Galaxy	Graphical	●●●	●○○	●●●	●●●	●●●	●●○
KNIME	Graphical	●●●	●○○	○○○	●●◐	●●●	●●○
Nextflow	DSL	●●○	●●●	●●●	●●●	●●●	●●●
Snakemake	DSL	●●○	●●●	●●◐	●●●	●●○	●●●
GenPipes	DSL	●●○	●●●	●●○	●●○	●●○	●●○
bPipe	DSL	●●○	●●●	●●○	●●◐	●●○	●○○
Pachyderm	DSL	●●○	●●●	●○○	●●○	●●●	○○○
SciPipe	Library	●●○	●●●	○○○	○○○	●●○	○○○
Luigi	Library	●●○	●●●	●○○	●●◐	●●○	○○○
Cromwell + WDL	Execution + workflow specification	●○○	●●○	●●●	●●◐	●●○	●●○
cwltool + CWL	Execution + workflow specification	●○○	●●○	●●◐	○○○	●●●	●●○
Toil + CWL/WDL/Python	Execution + workflow specification	●○○	●●●	●◐○	●●●	●●○	●●○

Figure 1: Overview of workflow managers for bioinformatics. Extracted from Wratten et al. (2021)

The first valuable feature is that this workflow manager is written in R. R (R Core Team 2022) is one of the most common languages used in bioinformatics and data analysis in general (Giorgi, Ceraolo, and Mercatelli 2022). Bioconductor, one of the main repositories for the analysis of biological data, contains tools programmed in R. Therefore using *targets* allows using all the tools available in Bioconductor directly in our pipeline[5]. Also, given the prevalence of R in bioinformatics, and other data science fields, many users will already be familiar with the language itself. For users already familiar with R, targets itself brings a small learning overhead, as most of the programming usually correspond to the task themselves and not so much to the pipeline preparation.

targets package by design also automatically keeps track of the dependencies in the different pipeline steps, therefore, when the pipeline is modified and rerun many time consuming calculations are skipped if the modifications do not affect them. This is particularly beneficial in the case of bioinformatics given that many of the data analysis steps include long computations. This dependency tracking also makes the pipeline easily reproducible as it guarantees that no invalid information is used and the results will be consistent when running the pipeline. It is also easy to track which information is used as an input in each of the tasks.

Regarding scalability, targets is prepared to run single-threaded, locally in parallel or in high-performance computing environments with just simple changes in the declaration of the pipeline.

5 Thanks to reticulate even tools programmed in Python could be used in our pipeline (https://rstudio.github.io/reticulate/)

As all the content in *targets* pipelines consist of text files (with the exceptions of inputs and outputs) this makes the pipelines defined using targets ideal for versioning control systems (e.g.: git).

targets is a general purpose solution, offering a very concrete functionality and is highly decoupled, from a software perspective, from the packages offering the analysis functionality, this means that changes in those packages can be incorporated without waiting for any update in *targets*.

Nonetheless, *targets* does have some shortcomings. The fact that it is a code-based solution, is not ideal for users with no, or few, programming experience. In those cases other solutions such as Galaxy or KNIME maybe more attractive and offer a more smooth learning curve, in fact both of them offer interfaces to the R language so they have access to R packages functionality at the cost of making them available in the web interface a bit more cumbersome. Albeit, R has interfaces with Python and, as mentioned before, can run Python code *reticulate* and can call any other language using system calls, other packages have more straightforward approaches to solve this problem. For example, Nextflow allows calling any scripting language in the same pipeline with almost no overhead.

Finally, one substantial difference with other more complex workflow managers is that *targets* does not keep track or controls the version of software or the environment where the pipeline is executed. This can be easily overcome by using complimentary R packages that control the versions in the R library such as *renv* (Ushey, 2022), or, as covered in a following section of this work, running the pipeline inside a Docker container.

Reproducible analysis using containers: Docker

Controlling the executing environment of a software has been a challenging task for many years. Software developers and system administrators coined the term *«Dependency Hell» (2023)* to describe the common situation in which a «Software A» depended on a given version «Library Z» while «Software B» depended on a different version of «Library Z» creating a complicated situation for installing «Software A» and «Software B» in the same environment. This situation was particularly troubling when installing «Software B» automatically updated «Library Z» making «Software A» in the best case unusable or in the worst case silently providing erroneous outputs. A similar kind of problem appeared when a piece of software was discontinued for the new version of a given operating system forcing the user to find an alternative software that provided the same or similar functionality. These problems were inherited and expanded as soon as researchers started using computational methods and writing analysis scripts.

Controlling the environment, from the OS to the software versions and libraries, are fundamental to be able to recreate the results of a given study. Albeit,

arguably, it could be easier to recreate the results days just after the initial analysis was done, recreating it months or even years after the analysis happened may be more complicated. R versions, versions of R packages, and Bioconductor packages change very often. Just in 2022 four different versions of R (Index of /src/base/R-4, 2022) and two Bioconductor versions were released (Bioconductor Releases, 2022) as well as five versions of the Ubuntu operating system. Therefore, controlling the exact versions used for creating them in first place is fundamental. On top of that, it is important that other researchers can recreate this results in different machines so they can build a new research on top, or maybe run the same analysis on their own set of data. Containers, and particularly Docker containers, are a common solution for these problems nowadays.

A Docker container is a piece of software that packs up code and dependencies and allows to an application to be run in different computing environments or at different times. Because it packs both code and dependencies it recreates a full environment, in our case: OS, system libraries, R, R libraries, Bioconductor libraries, etc. Allowing us to recreate the full set of results in any machine and in any time in the future, even if R, or any of the libraries used, would no longer be available.

Interactive data exploration tools: R/Shiny

Finally, in this work an interactive tool was created to reduce the loop between the data analysts and the final consumers of the results. It is commonly the case that the same person is not in charge of analyzing data and making decisions on the results of this analysis. Sometimes this creates a loop between the data analyst and the decision maker until the correct set of results required by the decision maker is achieved and a decision can be made. A way of improving this situation is empowering the decision maker to modify some of the setting in the final analysis, without the intervention of the data-analyst, through interactive applications.

In this work the R/Shiny (Chang et al. 2022) framework will be used that allows creating web applications using R.

Methodology & Materials

To implement this microarray analysis the maUEB package (Ferrer Almirall and Camacho Cano 2022) was used. This package, under development, is prepared to run a full microarray analysis.

For this work, a microarray analysis was implemented in *targets* (Landau 2021) and later run inside a Docker (Merkel 2014) container[6].

6 The Results section contain links to the full source code of this work if readers are interested in checking the code while reading the manuscript.

To implement the interactive application the R/Shiny framework was used (Chang et al. 2022).

The example dataset included in the *maUEB* package is used in this work. This datasets is extracted from a study based 12 samples hybridized in Clariom S Mouse Arrays from _Thermofisher_ with the following treatments.

Treatment:

- 4 samples from untreated mice (*CTL*)
- 4 samples from mice treated with PD1 (*PD1*)
- 4 samples from mice treated with CMP (*CMP*)

For illustration purposes, an imaginary Batch factor will be added to the sample data.

The following group of comparisons will be performed to determine the effects of each treatment on gene expression:

Effect of Treatment:

- PD1vsCTL = PD1 – CTL
- CMPvsCTL = CMP – CTL
- PD1vsCMP = PD1 – CMP

Results

This section will describe the main results of this work. All the code developed during this work can be found in the repositories and URLs listed below. Each of the repositories contain a short Readme with some instructions to run the pipeline and the package has a brief documentation site.

1. *targets* pipeline: https://github.com/zsigmas/UOC_TFM
2. Shiny applications:
    1. Package: https://github.com/zsigmas/brightPlots
    2. Package Documentation: https://zsigmas.github.io/brightPlots/
    3. Applications:
        1. https://zsigmas.shinyapps.io/brightplots_demo/
        2. https://zsigmas.shinyapps.io/brightplots_demo_upload/

All the repositories contain a Readme with a brief explanation on the usage of the packages and scripts.

A targets containerized microarray pipeline

Steps of the microarray analysis pipeline

In this work a common microrray analysis was implemented, the steps followed are listed below. The functions used[7], main inputs and outputs of the the pipeline are described in the list, but as the main object of this work is not the pipeline analysis itself but its implementation inside the *targets* package, the next sections will focus on the strategies used for its implementation. The resulting outputs of this pipeline can be found in the *targets* pipeline repository although the results themselves are not discussed in this work. A more extensive discussion of the results, the steps followed in the analysis, and documentation of the functions used can be found in the vignettes and documentation of the *maUEB* package.

1. **Data loading**
    1. Load file with sample information[8]
        - Function: `maUEB::read_targets`
        - Input: A targets file in csv format
        - Outputs: A dataframe with the loaded targets file
    2. Load CEL files[9]
        - Function: `maUEB::read_celfiles`
        - Input: A set of cel files
        - Output: An ExpressionSet
2. **Quality control**
    1. Raw expression data
        1. Create a boxplot figure
            - Function: `maUEB::qc_boxplot`
            - Input: An expression dataset
            - Output: A boxplot in PDF format
        2. Perform a Principal Component Analysis (PCA)
            - Function: `maUEB::qc_pca1`
            - Input: An expression matrix from the ExpressionSet

7 The usual R nomenclature, *package_name*::*function_name* is used to indicate which functions from which packages were used in each of the steps.

8 Notice that this file is commonly referred as «targets» file in microarray analysis, which unfortunately collides with the name of the *targets* package.

9 These files contain the raw expression information

- Output: A PCA figure in PDF or HTML format

3. Evaluate sample distance
   - Function: `maUEB::qc_hc`
   - Input: An expression matrix from the ExpressionSet
   - Output: An heatmap of sample distances with a dendogram of the hierarchical clustering in PDF format.
4. Create an arrayQualityMetrics package reported
   - Function: arrayQualityMetrics::arrayQualityMetrics
   - Input: An ExpressionSet
   - A quality control report in HTML format.

2. Data normalization
   - Function: `maUEB::normalization`
   - Input: A non normalized ExpressionSet
   - Output: A normalized ExpressionSet
3. Annotate normalized data
   - Function: `maUEB::save_annotations`
   - Input: An ExpressionSet and annotation Package
   - Output: an aafTable
4. The same quality control steps were repeated for the normalized data
   1. A PCA figure to control for batch effects was included
      - Function: `maUEB::qc_pca1`
      - Input: An expression matrix from the ExpressionSet
      - Output: A PCA figure in PDF or HTML format with a Batch Factor
5. Artifactual or outlier samples are removed from the normalized data
6. Create a figure of the standard deviations of the probesets
   - Function: `maUEB::sdplot`
   - Input: An expression matrix from the ExpressionSet
   - Output: A plot with the standard deviation of all probsets saved as a PDF file

7. Control probesets, probesets with missing or duplicated EntrezIDs and probesets with low variance are removed from the dataset
    - Function: `maUEB::filtering`
    - Input: An ExpressionSet
    - Output: A filtered ExpressionSet
8. Annotate filtered data
    1. Function, Input and Output as in the previous annotation step

3. Differential Expression Analysis (DEA) using a linear model

1. Create a design matrix for the analysis
    - Function: `maUEB::dea_lmdesign`
    - Input: An ExpressionSet and the DEA settings: grouping, covariates, etc.
    - Output: The design matrix of the model
2. Fit the linear model to the data
    - Function: `maUEB::dea_lmfit`
    - Input: An ExpressionSet and the design matrix
    - Output: An MArrayLM object containing the result of the fits
3. Calculate the relevant contrasts for the model
    - Function: `maUEB::dea_compare`
    - Input: The fitting and a set of contrasts
    - Output: An MArrayLM object containing the result of the contrasts
4. Gather results of the linear model comparisons:
    1. Create toptables with the results of the contrasts
        - Function: `maUEB::dea_toptab`
        - Input: A linear model fit, the ExpressionSet used for fitting and the list of coefficient and the table settings: p.value adjust method, maximum number of entries, etc.
        - Output: A dataframe with the toptable listing
    2. Create a summary table with the number of significant genes in each of the contrasts
        - Function: `maUEB::dea_summary_ngc1`

- Input: A list of toptable
- Output: A summary of the number of significant genes per threshold and comparison

3. Create a volcano plot of the contrasts
   - Function: `maUEB::dea_volcanoplot`
   - Input: A list of toptable
   - Output: A volcano plot in PDF format
4. Create a Venn diagram of the contrasts
   - Function: `maUEB::mc_venn_upset`
   - Input: A list of toptable
   - Output: A Venn Diagram in PDF format
5. Create a heatmap figure of the results
   - Function: `maUEB::mc_hm`
   - Input: A list of toptable and a toptable to plot on the heatmap
   - Output: A set of heatmap plots between the selected contrasts in PDF format

4. Gene Set Enrichment analysis

1. Over Reactome database
   - Function: `maUEB::abs_gsea_ReactomePAunfilt`
     - Input: A list of toptables and a list of contrasts
     - Output: A list of unfiltered Reactome results and a table with the number of terms at different pvalue thresholds for the different categories analyzed
   - Function: `maUEB::abs_gsea_ReactomePAfiltplot`
     - Input: An unfiltered Reactome results object
     - Output: A list of filtered results according to a p value threshold and a set of plots created with *clusterprofiler* in PDF format
2. Over Gene Ontology
   - Function: `maUEB::abs_gsea_GOunfilt`
     - Input: A list of toptables and a list of contrasts

        - Output: A list of unfiltered Gene Ontology results and a table with the number of terms at different pvalue thresholds for the different categories analyzed
    - Function: `maUEB::abs_gsea_GOfiltplot`
        - Input: An unfiltered Gene Ontology results object
        - Output: A list of filtered results according to a p value threshold and a set of plots created with *clusterprofiler* in PDF format

5. Over-representation analysis
    1. Over Reactome database
        - Function: `maUEB::abs_ora_ReactomePAunfilt`
            - Input: A list of toptables and a list of contrasts
            - Output: A list of unfiltered Reactome results and a table with the number of terms at different pvalue thresholds for the different categories analyzed
        - Function: `maUEB::abs_ora_ReactomePAfiltplot`
            - Input: An unfiltered Reactome results list
            - Output: A list of filtered results according to a p value threshold and a set of plots with the overrepresented term pathways
    2. Over Gene Ontology
        - Function: `maUEB::abs_ora_GOunfilt`
            - Input: A list of toptables and a list of contrasts
            - Output: A list of unfiltered Gene Ontology results and a table with the number of terms at different pvalue thresholds for the different categories analyzed
        - Function: `maUEB::abs_ora_GOfiltplot`
            - Input: An unfiltered Gene Ontology results list
            - Output: A list of filtered results according to a p value threshold and a set of plots with the overrepresented term pathways
6. Proof of concept Rmarkdown report:
    - Input: The quality control results calculated in the previous sections
    - Output: An HTML report of the quality control results

Parameters of the analysis pipeline

The parameters of the script are divided in four different sections:

1. Data loading and Quality Control
2. Linear model
3. Multiple comparisons
4. Gene Set Enrichment and Over-Representation analysis

The parameters for each of the sections are contained in four different nested lists with separated entries for each of the steps in the section. Each of these parameter lists are declared inside a function to avoid cluttering the global environment with the auxiliary variables.

In Code 1 an example of the function used to create the first parameter list for «Data loading and Quality Control» is included.

Code 1: Example of parameter list declaration

```
get_load_qc_parameters <- function() {
  data_dir <- "data"
  cel_dir <- "celfiles"
  targets_file_name <- "targets.RSRCHR.STUDY.csv"
  targets_file <- file.path(data_dir, targets_file_name)
  results_dir <- "results"
  targets_fact <- c("Group", "Batch")
  colorlist <- c(
    "firebrick1", "blue", "darkgoldenrod1",
    "darkorchid1", "lightblue", "orange4",
    "seagreen", "aquamarine3", "red4",
    "chartreuse2", "plum", "darkcyan",
    "darkolivegreen3", "forestgreen", "gold",
    "khaki1", "lightgreen", "gray67",
    "deeppink"
  )
  summary_fn <- "ResultsSummary_Load-QC-Norm-Filt.txt"

  raw <- list()
  norm <- list()
  norm_f <- list()
  norm_filt <- list()

 # boxplot ----

  raw[["boxplot"]] <- list(
    group = "Group",
    group.color = "Colors",
    samplenames = "ShortName",
    cex.lab = .6,
    outputDir = create_dir(results_dir, "raw", "boxplot"),
    label = "RawData"
  )

  norm[["boxplot"]] <- purrr::list_modify(
```

```
36      raw[["boxplot"]],
37      label = "NormData",
38      outputDir = create_dir(results_dir, "norm", "boxplot"),
39    )
40  # [...Part of the function is removed for space reasons...]
41
42    list(
43      targets_file = targets_file,
44      cel_dir = cel_dir,
45      targets_fact = targets_fact,
46      results_dir = results_dir,
47      color_list = colorlist,
48      raw = raw,
49      norm = norm,
50      norm_f = norm_f,
51      norm_filt = norm_filt,
52      outlier_samples = outlier_samples,
53      sdplot = sdplot,
54      filtering = filtering
55      )
56 }
```

For ergonomic purposes all the parameters in the pipeline are defined at the beginning of the analysis script. This allows the user to concentrate the changes on the first part of the script, while keeping the target definition, explained in the next section, intact.

This is only one of the multiple options to create this lists of parameters. For example, the same parameter declaration could be moved outside of the targets script, in a separate R file, separate R files could be declared for each of the parameters or they could be declared in other configuration markup languages, such as yaml. For this work it was decided that it was more ergonomic to have a single file containing the whole pipeline, but this is a highly opinionated decision.

Small multiples

A *targets'* pipeline is implemented as a list of targets each of them depending on previous targets. Below, there is a graph of an example *targets'* pipeline in Figure 2. Each of the elements in the graph is a target inside the pipeline that can, as depicted in the figure, depend on previous targets of the pipeline. When defined, each target will declare its name followed by the command used to create the target. In Code 2, the target *data* is declared and is created by running the function *get_data* over a previously calculated target *file*.

Code 2: Example of a target declaration

```
1  tar_target(data, get_data(file))
```

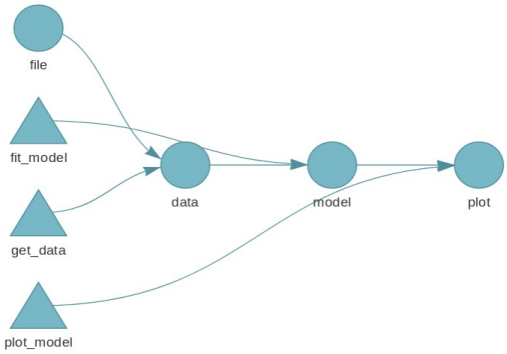


Figure 2: Pipeline graph extracted from targets manual.

The shape of the elements in the graph indicate its type: up-pointing triangles are functions, down-pointing triangles are R objects and circles are targets. While its color indicate its state, light blue are are outdated and dark green are up to date.

The depiction of the graph for the implemented in the pipeline of this work appears below. As seen in Figure 3[10], the pipeline implemented is more complicated than the one in the example.

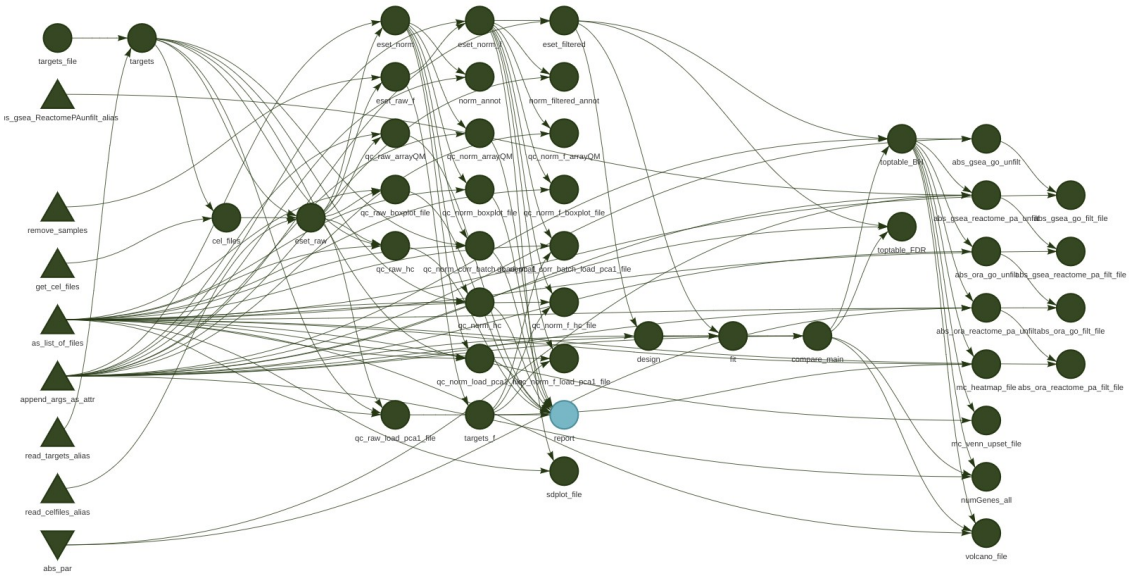


Figure 3: Graph of the pipeline implemented in this work

10 An interactive version of this figure can be found in *dag.html* file in the repository containing the pipeline.

Given the complexity of the pipeline, we followed a small multiples approach, described below, that in combination with the parameter approach in the previous section, will minimize the cost of modifying and maintaining the pipeline.

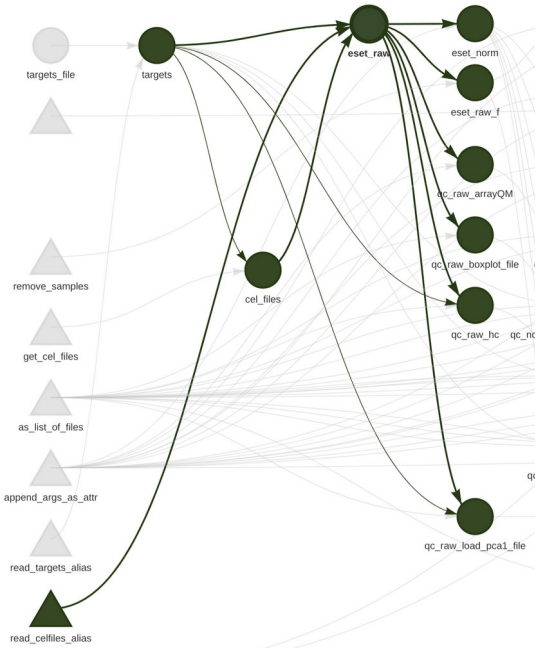


Figure 4: Extract of Figure 3 highlighting all the targets that are dependencies of eset_raw and those depending on it.

All the targets in the pipeline, with only few exceptions, are declared following the same pattern following a small multiples approach and follow a simple naming convention for easy tracking, see an example of a target declaration in Code 3.

Code 3: Example of a target declaration using small multiples

```
tar_target(
  name = qc_raw_boxplot_file,
  command = do.call(
    what = maUEB::qc_boxplot,
    args = purrr::list_modify(
      !!qcl_par$raw$boxplot,
      data = eset_raw
     )
   )
)
```

This target is called *qc_raw_boxplot_file*, this name will be used identify this target, for example, when loading it or when represented in the graph in Figure 3 and Figure 4. All the steps described in *Steps of the microarray analysis pipeline* section can be identified in Figure 3, for example, all targets starting

with the prefix *qc_raw* correspond to the targets of the quality control for raw data, all those starting with *qc_norm* to the quality control of the normalized data and the rest continue in a similar manner with easily identifiable target names.

Instead of including all the parameters in each of the target declarations we use the *do.call* function from R that allows calling a function with a set of parameters contained in a list. Those lists of parameters are declared at the top of the file. In this case, it is the list from quality control that corresponds to the list of parameters for the raw boxplot. The only parameters that are appended during the target declaration are those that make reference to previous targets in the pipeline and could not be declared in the parameter section.

This small multiples approach helps the user focusing on a single part of the script the parameter list, without having to navigate the whole target list changing the values of the parameters. Even more, if several targets use the same parameter list they can be reused very easily.

Note that in line 6 in Code 3, the reference to the parameter list is preceded by two exclamation marks, !!, also called the *bang bang* operator. This operator is part of the helpers in tidy evaluation[11] defined in Tidyverse. Very briefly, if that operator was not present, the whole declaration would depend on the whole qcl_par list, and not just on `qcl_par$raw$boxplot`. That would imply that any change in `qcl_par` would invalidate the target, when in fact only a change in `qcl_par$raw$boxplot` should invalidate it. By using the *bang bang* operator we do an in place replacement of `qcl_par$raw$boxplot` by its value therefore, when executing, it would be equivalent to Code 4. There would be no dependency on the other elements of the list, just on that specific entry·

Code 4: bang bang operator replacement

```
1    tar_target(
2      name = qc_raw_boxplot_file,
3      command = do.call(
4        what = maUEB::qc_boxplot,
5        args = purrr::list_modify(
6          list(
7            group = "Group",
8            group.color = "Colors",
9            samplenames = "ShortName",
10           cex.lab = .6,
11           outputDir = create_dir(results_dir, "raw", "boxplot"),
12           label = "RawData"
13       ),
14       data = eset_raw
15      )
16    )
```

11 The following link offers and indepth view of the operator: https://adv-r.hadley.nz/quasiquotation.html#quasi-motivation

```
17   )
```

This allows the pipeline user to modify the parameter of other targets without invalidating other targets in the same list.

Running specific targets

targets is prepared to calculate the minimal sets of targets needed for the required target. As seen in Code 5, it is possible to calculate specific targets during the analysis, for example it is interesting to calculate 'qc_raw_arrayQM' in an initial stage to check the arrayQualityMetrics reports for the raw data, before modifying other parameters and proceeding to calculate the normalized or the filtered data.

Code 5: running an specific target

```
1  targets::tar_make(qc_raw_arrayQM)
```

Recovering targets

Recovering targets back into the workspace can be done with a simple command as in Code 6.

Code 6: loading targets in the workspace

```
1  targets::tar_load(qc_raw_arrayQM)
2  arrQM <- targets::tar_read(qc_raw_arrayQM)
```

File outputs in target

Some of the functions of the maUEB package, do not return an R object (plot, dataframe, etc.), but it writes a series of files, usually PDFs. Therefore, a helper was required to create an Rmarkdown report based on these static files. Albeit, *targets* package covers targets in the form of files (see Code 7) by specifying the format parameter, it requires that the returned value of the command is the list of files created by the target. This is not the case in the maUEB functions, therefore a helper function *as_list_of_files* that lists all the files in a given directory was created. This functions decorates the maUEB functions by listing the files inside the output directory of the folder. In the example in Code 7 the decorated function returns a list of all the files inside the directory specified in the *outputDir* parameter passed to in the parameter list. Unfortunately, this strategy forces to place the outputs of each of the targets in separate directories.

Code 7: file outputs in targets

```
1  tar_target(
2    name = qc_raw_boxplot_file,
3    command = do.call(
4      what = maUEB::qc_boxplot %>% as_list_of_files("outputDir"),
```

```
5       args = purrr::list_modify(
6         !!qcl_par$raw$boxplot,
7         data = eset_raw
8         )
9       ),
10    format = "file"
11  )
12
```

Static report

The final target of the pipeline is an static[12] Rmarkdown report, only a proof of concept is presented for time reasons (see Figure 5). *targets* has a complimentary package *tarchetypes* that allow additional target types among which is the *tar_render* target as shown in Code 8.

Code 8: tar_render example

```
1  tarchetypes::tar_render(report, "report.Rmd", output_dir = "results")
```

This type allows rendering Rmarkdown that make reference to other targets as in Code 9.

Code 9: example Rmarkdown chunk using a target

```
1  ```{r, out.width=pdf_width, out.height=out_height, fig.align="center", echo =
FALSE}
2    file <- targets::tar_read(qc_raw_boxplot_file)
3    knitr::include_graphics(file)
4  ```
```

Results structure

As mentioned in section *File outputs in target*, the strategy used forces all results to be stored in different folders as shown in Code 10. This structure imitates the steps and have a simple naming convention so navigating through the result is as simple as possible.

Code 10: Folder structure of the pipeline results. For brevity only the main folders are shown in this structure

```
results/
├── raw
│   ├── boxplot
│   ├── hc
│   ├── pca
│   └── QCDir.Raw
├── norm
```

12 Note that, initially, this report was planned to be interactive. In the discussion section the reasons for this modification are explained.

```
|   ├── annotate
|   ├── batch_pca
|   ├── boxplot
|   ├── hc
|   ├── normalize
|   ├── pca
|   ├── pvca
|   └── QCDir.Norm
├── norm_f
|   ├── annotate
|   ├── batch_pca
|   ├── boxplot
|   ├── hc
|   ├── normalize
|   ├── pca
|   ├── pvca
|   └── QCDir.NormF
├── norm_filt
|   ├── annotate
|   └── filtering
├── norm_filtered
|   └── sdplot
├── lm
|   ├── lmcompare
|   ├── lmdesign
|   ├── lmfit
|   ├── sum_ngc1
|   ├── toptab
|   ├── toptab_BH
|   ├── toptab_FDR
|   └── volcano
├── mc
|   ├── heatmap
|   └── venn_upset
├── abs
|   ├── gsea
|   |   ├── go
|   |   |   ├── filt
|   |   |   └── unfilt
|   |   └── reactome
|   |       ├── filt
```

```
│   │       └── unfilt
│   └── ora
│       ├── go
│       │   ├── filt
│       │   └── unfilt
│       └── reactome
│           ├── filt
│           └── unfilt
└── report.html
```

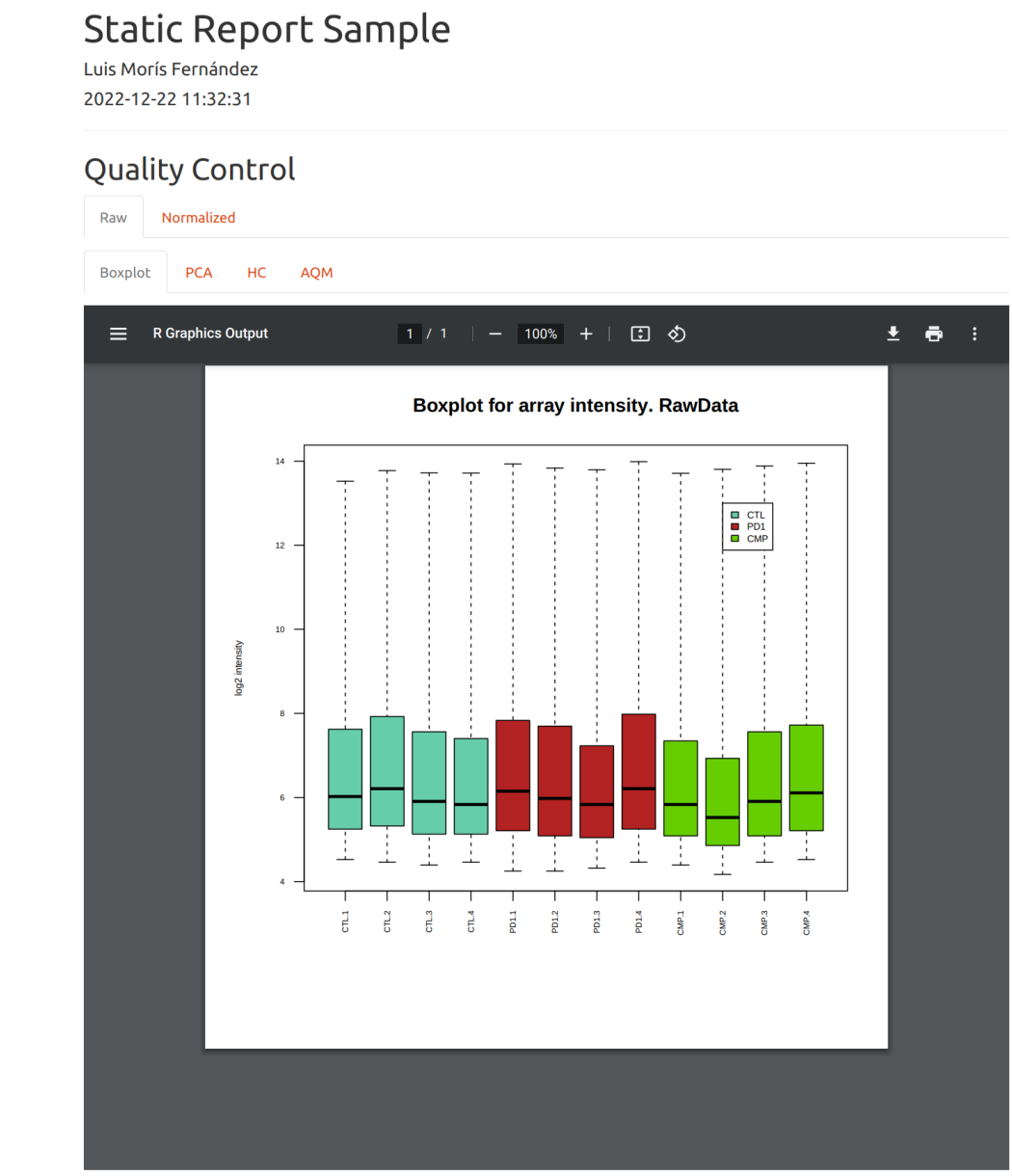


Figure 5: Static report output

Modifying the pipeline

Here an example of a modification in the pipeline is presented. Lets imagine that after the reviewing the quality control data of the raw and normalized data an additional anomalous sample is detected. The only change needed is adding a new sample in the qcl_par$outlier_samples entry, as shown in Code 11. Notice in Figure 6 that, after the change in Code 11, all targets depending on `eset_raw_f` are automatically outdated and when the whole pipelines is rerun using `targets::tar_make()`, or just part of it if a particular target is specified in the call, only the outdated targets will be recalculated, while all other targets will be skipped.

Code 11: Example adding an extra outlier sample in the qcl_par *list.*

```
1  # Before
2   outlier_samples <- "CMP.2"
3  # After
4  outlier_samples <- c("CMP.2", "PD1.1")
```

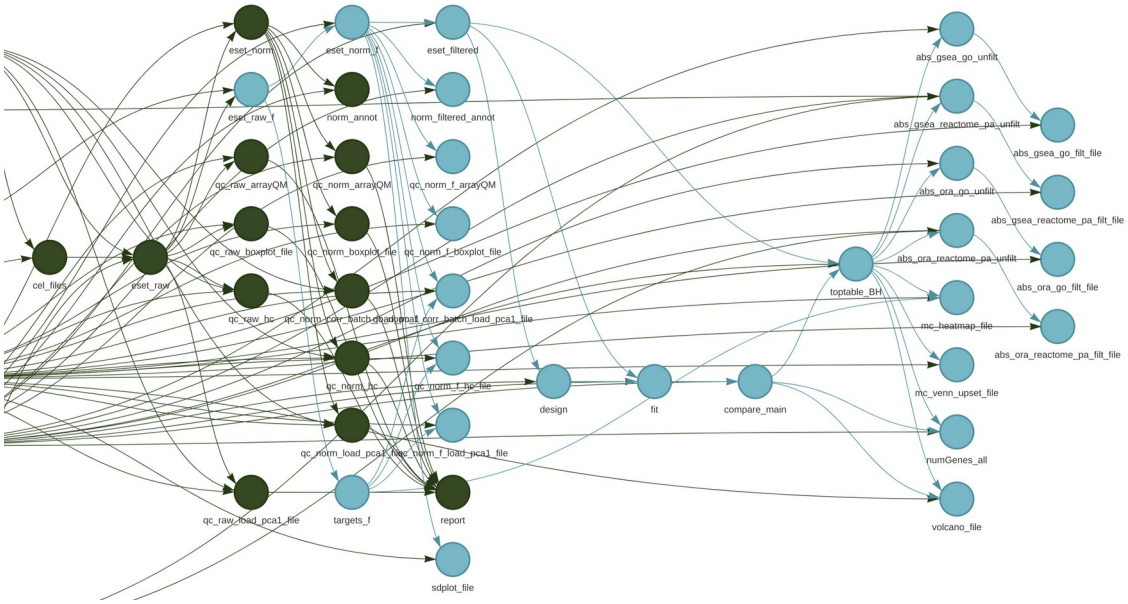


Figure 6: Partial view of the pipeline graph after adding an outlier sample. Notice that the report target is not outdated as, in the proof of concept in this work, it does only present quality control information for the Raw and Normalized data, and these quality control information is calculated before any outlier samples are removed

Containerizing in a Docker

A custom Docker image was created to make the previous pipeline fully reproducible. The image created was based on a rocker image (Boettiger and Eddelbuettel 2017) created on top of a Ubuntu Focal image and with a full Tidyverse installation (“Rocker-Org/Rocker-Versioned2” 2022). One of the

advantages of this image is that the R libraries version is frozen in an specified date, therefore rebuilding the image does not impact the R libraries version[13].

All the dependencies needed to run the previous pipeline were installed. The automatic installation of the Bioconductor dependencies for maUEB failed, therefore some Bioconductor packages are installed independently. Additionally a bug, required that the *preprocessCore* dependency was installed from source. Code 12 shows the code for the Dockerfile and all the additional scripts for installation are included in the folder *docker_scripts*.

Code 12: Dockerfile

```
# 4.2.1
FROM ghcr.io/rocker-org/tidyverse:4.2.1

ADD docker_scripts/install_system_dep.sh docker_scripts/install_system_dep.sh
RUN chmod +x docker_scripts/install_system_dep.sh
RUN docker_scripts/install_system_dep.sh
ADD docker_scripts/install_script.R docker_scripts/install_script.R
RUN chmod +x docker_scripts/install_script.R
RUN docker_scripts/install_script.R
ADD docker_scripts/install_bioc_dep.R docker_scripts/install_bioc_dep.R
RUN chmod +x docker_scripts/install_bioc_dep.R
RUN docker_scripts/install_bioc_dep.R

# Fixes bug
ADD docker_scripts/install_fix.sh docker_scripts/install_fix.sh
RUN chmod +x docker_scripts/install_fix.sh
RUN ls docker_scripts/*
RUN docker_scripts/install_fix.sh

CMD ["sh"]
```

A shell script, Code 13, that runs the whole pipeline inside the container is included:

Code 13: run_in_docker.sh contents

```
docker run --name target_trial -v $(pwd)/:/workspace -w /workspace --rm uoc:final Rscript -e "targets_::tar_make()"
echo "Remember the property of _targets is transferred to root due to running the pipeline in a docker!"
echo "Return property to the current user running: sudo chown -R ${USER} ./_targets"
```

This shell script makes several assumptions, easily changed for another use case:

1. The Docker image name is tag is *uoc:final*

13 This freezing date is specified as explained in: https://packagemanager.rstudio.com/client/#/repos/2/overview

2. The Docker command will be invoked from the base folder of the pipeline.

The Docker call will mount the pipeline folder inside the running Docker and execute the command at the end of the call. As the results are saved in the same pipeline folder they will be available once the command is run inside the Docker. As indicated in the messages of the shell script, one particularity in Unix systems is that the results folder will be owned by the *root* or *docker* user. This are superusers and usually those folders cannot be accessed or modified by a regular user, therefore, at the end of the execution the ownership of those folders must be modified.

An interactive application for data exploration using R/Shiny

An R/Shiny application to visualize and explore the results of the contrasts was implemented.

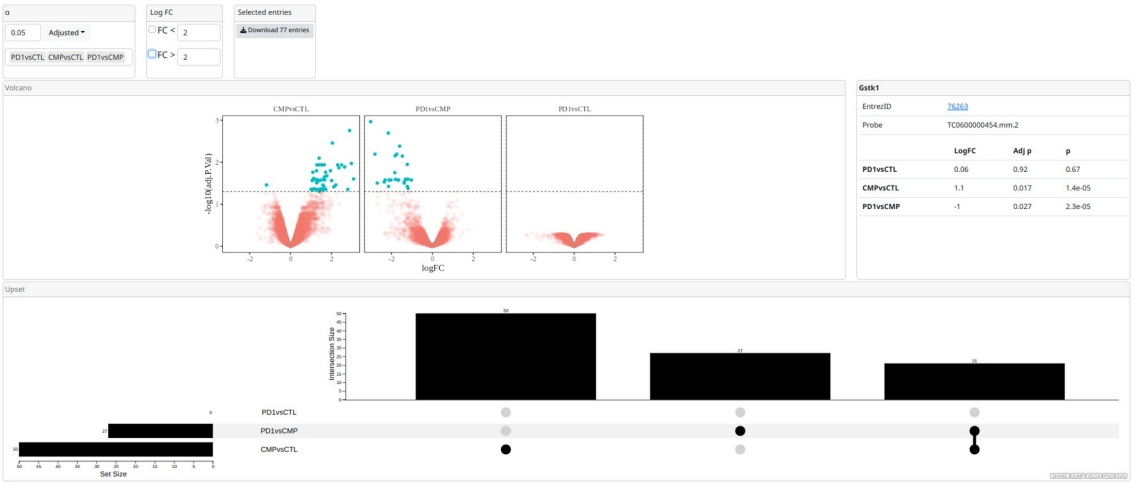


Figure 7: Application screenshot

As seen in Figure 7, the applications includes:

1. A control for selecting the p value threshold and if it should be applied over the adjusted or unadjusted p-value to select genes
2. A control for including limits depending on the log Fold Change, so only values above or below a given log Fold Change are selected.
3. A download button to download a csv file with all the selected genes
4. A volcano plot that shows the selected genes in another color
5. An information box that shows the information of a gene when it is hovered
6. An alternative graph to the classical Venn Diagram, that shows which of the selected genes are significative en each of the contrasts implemented using *upset.js* (Gratzl 2022)

The application can be loaded with an specific dataset, or an alternative application can be loaded with the option of uploading an RDS[14] file. The example applications can be run locally by installing the *brightPlots* package, available in https://zsigmas.github.io/brightPlots/, or accessed in their deployed versions at:

- https://zsigmas.shinyapps.io/brightplots_demo/ (With a demo dataset)
- https://zsigmas.shinyapps.io/brightplots_demo_upload/ (With the option of uploading a local RDS dataset)

The developed applications are composed by several Shiny modules (Chang 2020). Therefore the plots and menus presented in the application can be quickly recomposed or reused in other Shiny applications.

Discussion

In this section the results of this work, following the same section structure is discussed. The results of the CCEG objectives are also discussed.

A targets containerized microarray pipeline

The main goal of implementing the pipeline was fully achieved. An structure that allow pipeline users to concentrate in smaller portions of the script, parameter definition, and minimize the complexity of the target declaration, small multiples, was achieved.

Regarding difficulties, *maUEB* package is focused at this stage of development on static files that prevented the creation of more interactive reports. Nonetheless, this difficulty is easily overcome when further developments of the *maUEB* will also cover R objects and plots, so they could be embedded in the report and made interactive. Unfortunately, these changes escaped the time frame and scope of this work.

Containerizing in a Docker

The goal of running the pipeline inside a Docker container was achieved. A simple strategy to run this pipeline. This solution can easily be generalized to other pipelines. The main difficulty in this regard was founding some, already described but not solved yet, bugs when installing Bioconductor packages. A more smooth package installation procedure was expected, but the difficulties found did not prevent achieving this objective.

The solution found will help improving the reproducibility of Bioinformatics pipelines.

14 RDS is one of the R native formats for storing objects in files

An interactive application for data exploration using R/Shiny

This goal was fully achieved with an easy running application, which parts can be reused and recomposed. Albeit it is simple it may speed up some data-analyst vs data-decision-makers loops.

CCEG

This goals has been achieved as described in the initial planning. The goal of creating a pipeline that could improve reproducibility in Bioinformatics research was achieved and this may have an impact by reducing the amount of wasted resources in this field. Additionally, all software used to develop this work is free and mostly open source (with the exception Docker that is free but not open source), this allow researchers and institutions with low resources to benefit from the results of this work.

Regarding gender equality as mentioned during the initial planning the bibliography includes full names, citations were done regardless of the gender of the authors and, albeit the impact was minor in this work as no manual was written no assumptions about the gender of the application user or reader were made.

Conclusions

Reviewing the initial objectives of this work:

1. Describe the reproducibility problem in bioinformatics
2. Explore containers and workflow tools as a mean to improve the reproducibility of bioinformatics pipelines
3. Explore interactive tools as a mean to improve the decision making loop in clinical settings
4. Create a microarray analysis pipeline using containers that produce a report

All of these goals were achieved and new future lines of work expanding this work are discussed below.

Future lines of work

In the following section a short summary of possible improvements and future lines of work are discussed for each of the results of this work.

A targets containerized microarray pipeline

- Modifications to the *maUEB* package could be done to allow implementing an interactive report
- A more extensive manual on the pipeline could be written
- Additional versions of this pipeline could be written for other analyses

- Another strategies for creating the parameters list could be explored

Containerizing in a Docker

- Instead of mounting a volume, the data needed for the analysis could be included in the Docker image making the image self-contained for running the analysis
- Docker image could be uploaded to a Docker image repository for direct use by researchers

An interactive application for data exploration using R/Shiny

- Run refinement sessions with users to prioritize new functionalities, for example:
    - Download only genes that are significant in several comparisons
    - Provide more links to external databases with information about the genes
- Add more interactive graphs to the Shiny applications
- Include the interactive graphs inside an Rmarkdown report instead of a Shiny application[15]
- Explore alternatives to Shiny to add interactivity, for example, including javascript widgets that could be seen without the need of a Shiny runtime environment and an R installation and stored and shared as standalone files

15 https://rmarkdown.rstudio.com/lesson-14.html

References

1. Amstutz, Peter, Maxim Mikheev, Michael R. Crusoe, Nebojša Tijanić, Samuel Lampa, and et al. 2022. “Existing Workflow Systems.” Common Workflow Language Wiki, GitHub. August 30, 2022. https://s.apache.org/existing-workflow-systems.
2. Barba, Lorena A. 2018. “Terminologies for Reproducible Research.” arXiv. http://arxiv.org/abs/1802.03311.
3. “Bioconductor Releases.” 2022. 2022. https://www.bioconductor.org/about/release-announcements/.
4. Boettiger, Carl, and Dirk Eddelbuettel. 2017. “An Introduction to Rocker: Docker Containers for R.” https://doi.org/10.48550/ARXIV.1710.03675.
5. Brito, Jaqueline J, Jun Li, Jason H Moore, Casey S Greene, Nicole A Nogoy, Lana X Garmire, and Serghei Mangul. 2020. “Recommendations to Enhance Rigor and Reproducibility in Biomedical Research.” *GigaScience* 9 (6). https://doi.org/10.1093/gigascience/giaa056.
6. Buckheit, Jonathan B., and David L. Donoho. 1995. “WaveLab and Reproducible Research.” In *Wavelets and Statistics*, edited by Anestis Antoniadis and Georges Oppenheim, 103:55–81. Lecture Notes in Statistics. New York, NY: Springer New York. https://doi.org/10.1007/978-1-4612-2544-7_5.
7. Chang, Winston. 2020. “Modularizing Shiny App Code.” 2020. https://shiny.rstudio.com/articles/modules.html.
8. Chang, Winston, Joe Cheng, J. J. Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges. 2022. *Shiny: Web Application Framework for R*. https://shiny.rstudio.com/.
9. Chatterjee, Paula, and Rachel M. Werner. 2021. “Gender Disparity in Citations in High-Impact Journal Articles.” *JAMA Network Open* 4 (7): e2114509–e2114509. https://doi.org/10.1001/jamanetworkopen.2021.14509.
10. Claerbout, Jon F., and Martin Karrenbach. 1992. “Electronic Documents Give Reproducible Research a New Meaning.” In *SEG Technical Program Expanded Abstracts 1992*, 601–4. Society of Exploration Geophysicists. https://doi.org/10.1190/1.1822162.
11. “Dependency Hell.” 2023. Wikipedia. January 2, 2023. https://en.wikipedia.org/wiki/Dependency_hell.
12. Donoho, David L., Arian Maleki, Inam Ur Rahman, Morteza Shahram, and Victoria Stodden. 2009. “Reproducible Research in Computational Harmonic Analysis.” *Computing in Science & Engineering* 11 (1): 8–18. https://doi.org/10.1109/MCSE.2009.15.
13. Errington, Timothy M, Alexandria Denis, Nicole Perfito, Elizabeth Iorns, and Brian A Nosek. 2021. “Challenges for Assessing Replicability in Preclinical Cancer Biology.” *ELife* 10 (December). https://doi.org/10.7554/elife.67995.
14. Errington, Timothy M, Maya Mathur, Courtney K Soderberg, Alexandria Denis, Nicole Perfito, Elizabeth Iorns, and Brian A Nosek. 2021. “Investigating the Replicability of Preclinical Cancer Biology.” *ELife* 10 (December): e71601. https://doi.org/10.7554/eLife.71601.

15. Ferrer Almirall, Mireia, and Esther Camacho Cano. 2022. “MaUEB.” 2022. https://github.com/uebvhir/maUEB.
16. Gauthier, Jeff, Antony T Vincent, Steve J Charette, and Nicolas Derome. 2018. “A Brief History of Bioinformatics.” *Briefings in Bioinformatics* 20 (6): 1981–96. https://doi.org/10.1093/bib/bby063.
17. Giorgi, Federico M., Carmine Ceraolo, and Daniele Mercatelli. 2022. “The R Language: An Engine for Bioinformatics and Data Science.” *Life* 12 (5): 648. https://doi.org/10.3390/life12050648.
18. Gratzl, Samuel. 2022. “Upsetjs: ‘HTMLWidget’ Wrapper of ‘UpSet.Js’ for Exploring Large Set Intersections.” https://github.com/upsetjs/upsetjs_r/.
19. Hardwicke, Tom E., Maya B. Mathur, Kyle MacDonald, Gustav Nilsonne, George C. Banks, Mallory C. Kidwell, Alicia Hofelich Mohr, et al. 2018. “Data Availability, Reusability, and Analytic Reproducibility: Evaluating the Impact of a Mandatory Open Data Policy at the Journal *Cognition*.” *Royal Society Open Science* 5 (8): 180448. https://doi.org/10.1098/rsos.180448.
20. “Index of /Src/Base/R-4.” 2022. 2022. https://cran.r-project.org/src/base/R-4/.
21. Ioannidis, John P. A. 2005. “Why Most Published Research Findings Are False.” *PLoS Medicine* 2 (8): e124. https://doi.org/10.1371/journal.pmed.0020124.
22. Ioannidis, John P A, David B Allison, Catherine A Ball, Issa Coulibaly, Xiangqin Cui, Aedín C Culhane, Mario Falchi, et al. 2009. “Repeatability of Published Microarray Gene Expression Analyses.” *Nature Genetics* 41 (2): 149–55. https://doi.org/10.1038/ng.295.
23. Landau, William Michael. 2021. “The Targets R Package: A Dynamic Make-like Function-Oriented Pipeline Toolkit for Reproducibility and High-Performance Computing.” *Journal of Open Source Software* 6 (57): 2959.
24. Ludt, Annekathrin, Arsenij Ustjanzew, Harald Binder, Konstantin Strauch, and Federico Marini. 2022. “Interactive and Reproducible Workflows for Exploring and Modeling RNA-seq Data with PcaExplorer, Ideal, and GeneTonic.” *Current Protocols* 2 (4). https://doi.org/10.1002/cpz1.411.
25. Macleod, Malcolm R., Susan Michie, Ian Roberts, Ulrich Dirnagl, Iain Chalmers, John P. A. Ioannidis, Rustam Al-Shahi Salman, An-Wen Chan, and Paul Glasziou. 2014. “Biomedical Research: Increasing Value, Reducing Waste.” *The Lancet* 383 (9912): 101–4. https://doi.org/10.1016/S0140-6736(13)62329-6.
26. Mangul, Serghei, Lana S. Martin, Eleazar Eskin, and Ran Blekhman. 2019. “Improving the Usability and Archival Stability of Bioinformatics Software.” *Genome Biology* 20 (1). https://doi.org/10.1186/s13059-019-1649-8.
27. Markowetz, Florian. 2015. “Five Selfish Reasons to Work Reproducibly.” *Genome Biology* 16 (1). https://doi.org/10.1186/s13059-015-0850-7.
28. Merkel, Dirk. 2014. “Docker: Lightweight Linux Containers for Consistent Development and Deployment.” *Linux Journal* 2014 (239): 2.
29. Munafò, Marcus R., Brian A. Nosek, Dorothy V. M. Bishop, Katherine S. Button, Christopher D. Chambers, Nathalie Percie du Sert, Uri Simonsohn, Eric-Jan Wagenmakers, Jennifer J. Ware, and John P. A.

Ioannidis. 2017. “A Manifesto for Reproducible Science.” *Nature Human Behaviour* 1 (1): 0021. https://doi.org/10.1038/s41562-016-0021.
30. Nelson, Leif D., Joseph Simmons, and Uri Simonsohn. 2018. “Psychology’s Renaissance.” *Annual Review of Psychology* 69 (1): 511–34. https://doi.org/10.1146/annurev-psych-122216-011836.
31. Ousterhout, John. 2018. *A Philosophy of Software Design*. 1st ed.
32. Peng, Roger D. 2011. “Reproducible Research in Computational Science.” *Science* 334 (6060): 1226–27. https://doi.org/10.1126/science.1213847.
33. Popper, Karl R. 1959. “*The Logic of Scientific Discovery*.” *Physics Today* 12 (11): 53–54. https://doi.org/10.1063/1.3060577.
34. R Core Team. 2022. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.
35. “Rocker-Org/Rocker-Versioned2.” 2022. 2022. https://github.com/rocker-org/rocker-versioned2.
36. Rodgers, Peter, and Andy Collings. 2021. “What Have We Learned?” *ELife* 10 (December): e75830. https://doi.org/10.7554/eLife.75830.
37. Ushey, Kevin. 2022. “Renv: Project Environments.” https://rstudio.github.io/renv/.
38. Wallach, Joshua D., Kevin W. Boyack, and John P. A. Ioannidis. 2018. “Reproducible Research Practices, Transparency, and Open Access Data in the Biomedical Literature, 2015–2017.” Edited by Ulrich Dirnagl. *PLOS Biology* 16 (11): e2006930. https://doi.org/10.1371/journal.pbio.2006930.
39. Wratten, Laura, Andreas Wilm, and Jonathan Göke. 2021. “Reproducible, Scalable, and Shareable Analysis Pipelines with Bioinformatics Workflow Managers.” *Nature Methods* 18 (10): 1161–68. https://doi.org/10.1038/s41592-021-01254-9.