



# Detailed Implementation of a Reproducible Machine Learning-Enabled Workflow

PRACTICE PAPER

KENNETH E. SCHACKART III

HEIDI J. IMKER

CHARLES E. COOK

\*Author affiliations can be found in the back matter of this article

ubiquity press

## ABSTRACT

Machine learning (ML) and advanced computational methods are powerful tools for processing and deriving value from large data volumes. These methods are being developed and deployed rapidly, but best practices are still evolving regarding code and data standards, leading to irreproducibility of ML-enabled research. In this Practice Paper, we describe our efforts to make a ML-enabled research project to create a global inventory of biodata resources open and reproducible. To contribute to community conversations on evolving norms and expectations, we present our experiences as a practical, real-world case study that includes the implementation details as well as our overall approach and subsequent decisions. Our goal in openly sharing this experience is to provide a concrete example that others may consider as they look to vet, adapt, and adopt similar strategies to make their own work open and reproducible.

## CORRESPONDING AUTHOR:

**Kenneth E. Schackart III**

Global Biodata Coalition,  
12 quai Saint-Jean 67080,  
Strasbourg, France

[schackartk1@gmail.com](mailto:schackartk1@gmail.com)

## KEYWORDS:

computational reproducibility;  
open science; research  
software; FAIR data; machine  
learning workflow; biodata  
resource inventory

## TO CITE THIS ARTICLE:

Schackart III, K E, Imker,  
H J and Cook, C E 2024  
Detailed Implementation  
of a Reproducible Machine  
Learning-Enabled Workflow.  
*Data Science Journal*, 23: 23,  
pp. 1–14. DOI: [https://doi.  
org/10.5334/dsj-2024-023](https://doi.org/10.5334/dsj-2024-023)

## 1. INTRODUCTION

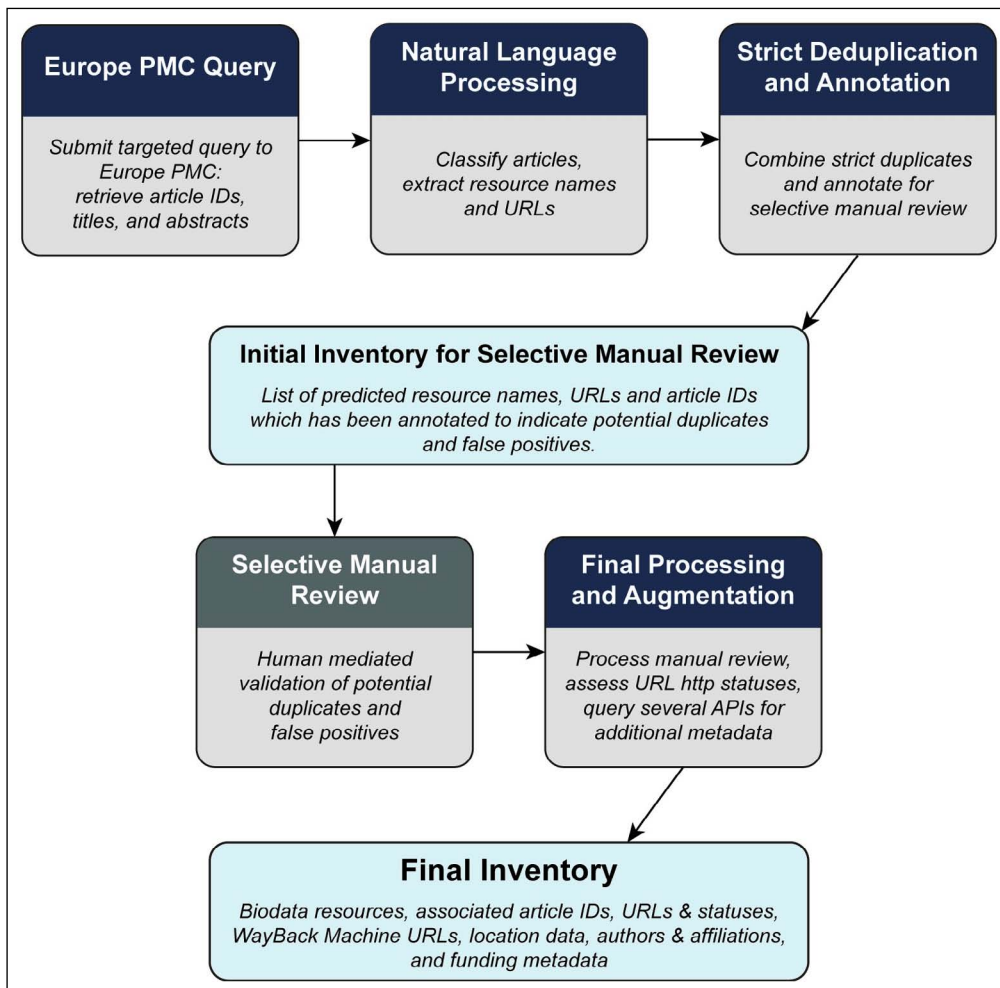
There is broad concern over the lack of reproducibility in science (Baggerly and Coombes 2009; Peng and Hicks 2021), with many believing there is a crisis (Baker 2016). While the extent is contested (Fanelli 2018; Leek and Jager 2017), concerns about scientific reproducibility are ongoing, and flawed study designs and irreproducible analyses play a role. There have been efforts to encourage better practices, such as pre-publication of study protocols, analysis plans, and all code (Haring and Bell 2018). However, as argued in Haring (2018), while the different biases in production and reporting of research are largely identifiable and modifiable, continued methodological training for early career researchers is also crucial.

Use of machine learning (ML) in biosciences has proliferated so rapidly that it is difficult for adoption of good practices and proper training to keep pace. Open Science practices, such as public release of code and data, aim to remedy this (Walters 2020). While access to code and data are necessary for reproduction of computational results, such access does not guarantee that results can be reproduced. Indeed, the recent Ten Years Reproducibility Challenge investigated the ability to rerun code and reproduce results from projects ten years or older, and the issues involved resulted in a useful ‘reproducibility checklist’ (Perkel 2020). Additionally, efforts have been made to set standards for reproducible code, including for ML, and they serve as rubrics for assessing reproducibility (Heil et al. 2021). What seems lacking, however, are detailed examples of practical implementations. This work provides such an example by explaining how a ML-enabled study was planned and executed with reproducibility as an explicit goal from the onset of the project.

In our example, the study is a ML-enabled inventory of biodata resources identified from the scientific literature. Biodata resources are biological, life sciences, and biomedical databases that archive research data generated by scientists, serving as the repositories of record for particular data types; as well as knowledge bases that add value by aggregation, processing, and expert curation. These resources are connected through extensive exchanges of data and form a distributed global infrastructure. They are crucial for the entire life science research endeavor and are used ubiquitously.

However, the infrastructure is not well-described. A number of existing resource registries, such as re3data and FAIRsharing, have done a commendable job of cataloging resources either through self-registration by the resource owner or through addition by a curator. However, neither the number of resources nor their location has been systematically explored. A better understanding of the scale of the infrastructure, as provided by this inventory, will aid funders and other stakeholders in addressing challenges to sustainability faced by the infrastructure. The methods and results of creating this inventory are fully described elsewhere (Imker et al. 2023). However, during preparation of that manuscript we realized that there were many additional details to share about how we attempted to design and implement a reproducible workflow—details we wish we had found in the literature ourselves.

As context for this reproducibility case study, the following provides an outline of the research project (Figure 1), and we invite readers to access the openly available article referenced above for additional details. Briefly, the study first utilized the API of Europe PMC (europepmc.org) (The Europe PMC Consortium 2015), which is a data resource that archives a large corpus of medical and life sciences publications (Ferguson et al. 2021). Europe PMC provides both individual (browser-based) and automated (API-based) queries. Our workflow started with a targeted query to the Europe PMC API to retrieve the titles and abstracts of publications for which both a URL and the word ‘data,’ ‘database,’ or ‘resource’ are present in the title and/or abstract. The results of the query represented publications that might describe a biological (biodata) resource. A 10% random subset of publications from this initial result was manually classified as describing or not describing a biodata resource (see Imker et al. 2023 and additional documentation in Imker and Schackart 2023). Those that did describe a biodata resource were curated to label the resource’s common name (e.g., PDB) and full name (e.g., Protein Data Bank) (Berman et al. 2000). Recently, BERT (Bidirectional Encoder Representations from Transformers) performed well on NLP tasks (Wolf et al. 2020). Several BERT models pre-trained on biomedical corpora (e.g., SciBERT, PubMedBERT, BioMed-RoBERTa-RCT, etc.) were selected from huggingface.co and fine-tuned for the classification (predicting if the article describes a biodata resource) and named-entity recognition (predicting common and full name) tasks. Further downstream processing was performed, including URL extraction and HTTP status checking, before finalizing the inventory.



**Figure 1** Flowchart of overall study design to identify biodata resources from the scientific literature. The fine-tuning procedure is not shown. Reproduced unmodified from (Imker et al. 2023) under Creative Commons Attribution License.



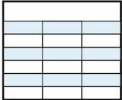
During the study, a strong emphasis was placed on Open Science, reproducibility, and robustness of the codebase and documentation for both philosophical reasons (in support of Open Science) and practical reasons (enabling future updating of the inventory). The entire process, from data splitting, model training and selection, to all downstream processing, is encapsulated in a Snakemake workflow (Köster and Rahmann 2012). This allows reproduction of the entire analysis with a single command. Strong standards of code quality were developed and are enforced through the use of static code checking and automated testing. Additionally, significant efforts were made to make all data products findable, accessible, interoperable, and reusable (FAIR) (Wilkinson et al. 2016).

When we began the project, we turned to the literature for robust examples of reproducibility that implemented both open data practices and code standards. Several articles contain excellent conceptual overviews (e.g., Wilson et al. 2017; Gruning et al. 2018; and a recent synthesis in Ziemann et al. 2023), and examples of efforts to implement Open Science practices, including open data and/or computational reproducibility, have been reported from many domains (e.g. Bush et al. 2022 in neuroimaging; Figueiredo et al. 2022 in ecology; and Kim et al. 2018 in bioinformatics). These examples show how reports often focus on a few critical aspects of implementing Open Science practices; for example, although Bush et al.’s work didn’t provide the explicit code details we were interested in, it provides excellent administrative considerations like accounting for trade-offs. Figueiredo et al. provides a clear and detailed ‘kit’ for using computational notebooks in order to both show the value of reproducible workflows as well as enable their adoption. In Kim et al.’s article, they first describe their efforts to reproduce a study in which the original authors had taken steps towards reproducibility, the challenges faced despite those steps, and then their own iteration towards greater reproducibility. While there is similarity between these efforts and our goals, when it comes to implementation, there are many details which are inherently different, if described at all, because of variation in the nature of the work and relevant packages and tools. Not surprisingly, we were unable to locate implementation details that mapped exactly to our project and goals, so we adapted to fit our scenario. As a ML project, we found Heil et al.’s rubrics especially helpful in providing a

framework for us to consider and specific goals to aim towards. We recognize that there are other ways of attaining these goals, and projects that have subsequently cited Heil et al.'s standards show this diversity (e.g., [Wanner et al. 2023](#); [Kaczmarzyk et al. 2023](#); and [Heil et al. 2023](#)). We offer our experience as just one example of how to make a computationally heavy study reproducible and open. We provide the reasoning behind the various considerations, which may be applicable to other research projects. We also provide specific examples of how those were realized in this study.

## 2. HAVE A PLAN

'A goal without a plan is just a wish,' wrote Antoine de Saint-Exupéry in *The Little Prince* ([de Saint-Exupéry 1943](#)). As with any other part of a research project, planning ahead makes the path to achieving reproducibility as smooth as possible. To this end, early in the project we developed an Open Science Implementation Plan ([Imker and Schackart 2022](#)). In this document, we outlined the goals for reproducibility and how we planned to achieve them. These goals were organized into four groups: reproducibility of methods, code standards, data standards, and external review/validation ([Figure 2](#)).

	Objectives	Tools & Methods
<b>Reproducibility</b> 	Data, model, and code availability	GitHub, Zenodo
	Single-command dependency installation	Pip, conda, renv, Make
	Ability to rerun entire analysis	Snakemake
<b>Code Quality</b> 	In-code documentation	docstrings, type hints
	Readability and community compliance	pylint, flake8, lintr
	Bug prevention	Unit tests, mypy
<b>Data</b> 	Findability	DataCite metadata
	Accessibility	GitHub, Zenodo
	Interoperability	PMIDs, ISO 3166
	Reusability	CSV formatting

**Figure 2** Graphical overview of the objectives of the study and the tools and methods used to address them regarding reproducibility, code quality, and data standards. The execution of these objectives was assessed by external review and validation.

By considering these topics early in the project, we explicitly defined what expectations we had for our Open Science goals. Keeping these goals in mind helped ensure that the effort and resources required to obtain them was anticipated and considered a core aspect of the project. This minimized the accumulation of technical debt that would have been time-consuming and difficult to address near the end of the project.

## 3. REPRODUCIBILITY OF METHODS

We found the reproducibility standards (bronze, silver, gold) defined by Heil et al. ([2021](#)) useful for ranking reproducibility levels. In our case, bronze alone was not acceptable (data published and downloadable, models published and downloadable, source code published and downloadable). Obtaining silver was acceptable (bronze + dependencies set up in a single command, key analysis details recorded, analysis components set to deterministic), but the gold standard was our goal (silver + entire analysis reproducible with a single command).

### 3.1. MEETING THE BRONZE STANDARD

The bronze standard of reproducibility is characterized by having the following published and downloadable: all data necessary for reproduction, trained models, and source code.

Data availability and, more broadly, FAIRness (findability, accessibility, interoperability, and reusability) will be further discussed in a later section. To address the minimum requirements of the bronze standard, all data are available for download from the project's Github and Zenodo repositories.

Model availability is addressed in a few ways. All of the models used in this project were pre-trained by other groups and made available on HuggingFaceHub (HFHub, <https://huggingface.co/>). As part of model training, these pretrained models were fine-tuned to various tasks (sequence classification and token classification). These fine-tuned models are made available on HFHub.

All source code is stored in two places. First, GitHub serves as a ‘living’ repository. An important aspect of Open Science is providing a place for open discussion (and criticism) of methods. The GitHub Issues system permits and encourages free and open commentary of computational methods. However, GitHub repositories are not immutable. It is important to have the methods, as described in the original publication, preserved and available, so the source code used to obtain the results in the associated full publication mentioned above has been archived as a code release on GitHub and also deposited into the Zenodo archive unmodified.

### 3.2. MEETING THE SILVER STANDARD

The silver standard requires, in addition to those aspects listed in the bronze standard, that all dependencies can be installed and set up with a single command, key analysis details are recorded, and all analysis components are deterministic (not random).

A common challenge for reproducibility is having simple installation procedures. To reach the silver standard in this regard we wanted it to be possible to install all dependencies with a single command. For Python-based projects that is often possible with the command ‘pip install -r requirements.txt’ ([pypi, n.d.](#)). However, sometimes other dependencies not covered by pip need to be installed. To simplify this step, we utilized Make (GNU Make v4.2.1) ([GNU Make 1988](#)). While Make is a powerful tool intended for the control of executable files, we use it only for effectively creating aliases for shell commands. In the case of installation, we provide a Make target called ‘setup’. By doing so, the user can simply type ‘make setup’ and shell commands are executed to install all dependencies, including running pip (v21.1.2) for installing Python dependencies ([pypi, n.d.](#)) and renv (v0.14.0) for installing R dependencies ([Ushey 2022](#)).

In addition to providing a simple pip install procedure we created a conda installation procedure ([Conda 2017](#)). While using pip to install dependencies at the user level is sufficient in isolated environments, such as Google Colab (<https://colab.research.google.com/>), it can lead to conflicts on other systems if a virtual environment is not used. Conda (v22.9.0) provides an isolated environment in which the project-specific dependencies are installed. By providing a conda environment description (yaml) file, it is possible to recreate the conda environment in a single command.

Beyond virtual environments, containers such as Docker ([Merkel 2014](#)) are often used for documenting and sharing computational environments directly. However, containers can be challenging to use in certain environments. We wanted this project to be reusable for people with a wide range of technical skills, including those who may not have ready access to a robust computational infrastructure. This is especially important when thinking of potential users on a global scale, whose access to resources will be highly variable. This dependence on access to computational resources has been noted as an important part of data democratization ([Hook and Porter 2021](#)). Here, we designed this project to be run on Google Colab for its low barrier to entry and its provision of graphics processing units (GPUs) for free use. Unfortunately, Colab does not natively support common container services such as Docker. However, by providing several options for dependency installation we hope that future users can find one to suit their needs.

Sufficient documentation of ‘key analysis details’ is subjective. To satisfy this requirement, in addition to an overview README that describes the entire repository, we provide README files in every directory within the repository. These explain what the various files/scripts are and how they relate to each other. Since 2021 GitHub supports the use of Mermaid, a JavaScript-based diagramming and charting tool ([Sveidqvist 2014](#)), in markdown files, which we leverage to create informative flowcharts illustrating workflow logic.

An often overlooked key to reproducibility in computational methods, particularly ML methods, is seeding pseudo-random processes such that they are deterministic ([Ahmed and Lofstead 2022](#); [Heil et al. 2021](#)). The random numbers generated by pseudo-random number generators can have significant effects on the trained model and model performance ([Ahmed and Lofstead 2022](#)). So, to make the process reproducible, we added options to use seeding to make the processes deterministic.

### 3.3. MEETING THE GOLD STANDARD

The gold standard implies that the entire analysis can be run with a single command (Heil et al. 2021). Such single-command analyses require the use of a workflow manager, of which there are several options. We utilize Snakemake (v7.1.1), which facilitates automation through the definition of ‘rules’ or steps that take inputs and generate outputs. By stating what outputs are desired Snakemake creates a directed acyclic graph of which rules must be executed to create the specified output. For instance, in this project we specify that we would like the final output file to contain the classified articles along with extracted metadata. If the final output is not present, Snakemake executes all necessary steps in the pipeline including data splitting, model training and comparison, classification and Named Entity Recognition (NER), and all downstream processing. With the help of a Make alias, the Snakemake workflow for reproducing all results can be run with the single command ‘make train\_and\_predict’.

It is important to be able to reproduce all results from the raw data to final results, including model training. However, model training is resource intensive, and may require the use of specialized hardware such as a GPU for training to be performed in a reasonable amount of time. Requiring that all models be trained to reproduce results may be a practical challenge to reproducibility. To minimize the computational resources necessary for reproduction all fine-tuned models are available in HFHub. If the fine-tuned models are downloaded and present when Snakemake is run then Snakemake will not execute model training.

## 4. BEYOND REPRODUCIBILITY

The goal of reproducibility is to allow anyone to reproduce the results of published research. We have provided, as described above, a system that allows the results of the inventory of global biodata resources to be reproduced. However, this project was also designed to allow the entire analysis to be rerun periodically. Strictly speaking, this goes beyond reproduction since the underlying data is expected to change as more publications are added to the corpus of literature archived in Europe PMC, so the methods developed need to be generalizable. Generalizability benefits from the same considerations as reproducibility but tends to include additional challenges.

We approached generalizability with the same standards as reproducibility and wanted to make updating the inventory possible with a single command. To this end we designed a second Snakemake workflow for periodically updating the inventory. For this process the trained models can be automatically obtained from Zenodo using the setup command. The previously best performing models for each task are used, which eliminates the need for retraining and evaluation.

## 5. CODE STANDARDS

We’ve taken the philosophy that the results of a computational research project are no more trustworthy than the code used to produce them. Trustworthiness of code is dependent on code quality, including considerations such as readability and robustness. In this section we will describe the measures taken to ensure code quality such as code formatting, static code checking, and automated testing.

### 5.1. CODE FORMATTING

To accomplish Open Science, accessibility of code should not be limited to code being publicly available. True accessibility requires that code also be readable and well documented. A good first step is to utilize a code formatter, which all modern programming languages have. We used yapf v0.31.0 to format all of the Python code in this project (Google Inc.). Similarly, Snakemake files were formatted with snakefmt v0.6.0, and R files were formatted with styler v1.7.0 (Hall and Letcher 2020; Müller et al. 2021). These steps are meant to ensure that all components of the project are readably formatted and documented to maximize their ease of use for others.

### 5.2. STATIC CODE CHECKING

Another measure taken to increase code robustness is static code checking. Again, the code checking tools available will depend on the language. We utilize the linters pylint v2.8.2 and flake8 v3.9.2 to check all Python code to ensure that community code standards are upheld and to detect code smell (patterns indicative of potential problems) (Thénault 2001; Ziade and

[Cordasco 2011, p. 8](#)). Many of the items that these linters consider can greatly improve code quality and readability. Some examples of considerations of the linters are: line lengths must be limited to predefined thresholds, within any context (e.g., a function) there should not be too many variables, and all functions should have docstrings. These, and many other requirements, encourage developers to write cleaner, more readable code.

Additionally, while type annotations are not required in the Python community, we implemented them as they provide a number of benefits. Type annotations provide built-in documentation by defining the data types of all inputs and outputs of functions. A lesser discussed benefit of type annotations is that they provide an enhanced integrated development environment (IDE) experience since the IDE has more knowledge of the variables and can give better help messages, syntax highlighting, and autocompletion. The final benefit of type annotations is prevention of unforeseen bugs when they are used in conjunction with a static type checker. We used mypy v0.812 to check type compatibility within all our Python code ([Lehtosalo 2012](#)). This can significantly reduce the chances of encountering bugs that occur not at compile time (since Python is interpreted and dynamically typed), but instead at runtime, which can be more difficult to resolve and may not show up until running the code at a later time.

While static code checking has many benefits, programmers need not strictly adhere to all suggestions made by the code checkers. Luckily, most tools are configurable. Importantly, the user can disable certain warnings. To ensure portability of these configurations, most code checkers allow for configurations to be defined in a resource configuration (rc) file rather than in global or user settings. Accordingly, we have included our rc files in the GitHub repository so that when someone else runs the code checkers on our published code they yield the same results.

### 5.3. TESTING

A crucial software engineering practice that is often absent from research code is testing. Testing in all of its forms: unit, integration, and end-to-end, defines the specifications of a piece of software and ensures that the software meets those specifications when the tests pass. This has numerous benefits that cannot be understated.

One of the primary benefits is that tests serve as a contract, which is a form of documentation. A unit test of a function explicitly states what kinds of input are expected and what kinds of outputs will be produced. For documentation, the only thing better than telling what a function does (through comments and docstrings) is showing through tests (asserting that when certain inputs are provided, the expected output is returned). While the descriptions provided in docstrings and comments are what the developer intends the software to do, a passing test demonstrates that it indeed does what was intended. Conversely, anything not covered in the test cases is where the contract ends. Tests ensure that the code can do what it says.

From an Open Science perspective testing is particularly valuable. Not only does testing provide more detailed documentation than could ever be provided in an article's methods section, but it facilitates community feedback and contributions. Making changes to software always poses the risk of disrupting previous functionality. When considering applying community feedback or contributions this is problematic. However, with strong test coverage, developers can have more confidence that updates do not introduce breaking changes, as long as all previously passed tests still pass. Indeed, they provide a clear avenue for addressing bugs which may be caught by the community. Developers can add another test case that exposes the bug, then modify the code such that the new test and all previous ones pass. This is effectively amending the contract provided by the tests so that it is more comprehensive. Without tests in place developers would have to check that the code still behaves as described manually. Such checking is so error prone that many researchers may be hesitant to implement changes suggested by others.

Of course, adding strong test coverage does require more work than, for instance, implementing static code checks or formatting. Without tests, though, code must be manually assessed to ensure that a given piece of software is able to perform its intended task, and there is a barrier to implementing community feedback. Further, a lack of tests is a form of technical debt, and the price is paid when trying to refactor or fix bugs.

Pytest v6.2.4 was used as a testing framework for all Python code in this project ([Krekel 2004](#)). Pytest plugins for flake8, pylint, and mypy are used to include static code checks of each file as part of the test suite (pytest-flake8 v1.0.7, pytest-pylint v0.18.0, pytest-mypy v0.8.1) ([Bader](#)

2016; Gee 2015; Lockhart, 2015). This makes it such that the test suite cannot pass without all static checks passing. Additionally, most functions have associated tests, and most scripts also have end-to-end tests that ensure that they properly reject bad inputs and produce correct output when given good input. While we aim to have good test coverage, some functions and scripts are not comprehensively tested. This is generally the case for functions/scripts that take a very long time to run, such as the actual process of model training. Additionally, the Snakemake workflows developed are not formally tested using an automated testing framework, although it would be best to do so and we may implement this at a later time.

## 5.4. CONFIGURABILITY

Our aim was that the users of code, whether for reproducibility, generalization, or separate implementation, would not need to edit source code to change its behavior within the intended use cases. Parameters that may change could be supplied as inputs/arguments instead. Often, this means that paths to input files should not be hard-coded but rather passed in when calling a script. In terms of ML projects, this also often applies to hyperparameters.

One solution to this is to use parameterization extensively and, in order to make the analyses reproducible, to store the parameters used in configuration (config) files. By doing so, others can see what parameters were used to generate the results. This process additionally gives future users a clear indication of what parameters are likely okay to change, all without them having to edit any source code.

We store a large number of parameters in config files such as input/output directories, training parameters, and locations of fine-tuned models. To train a new model and compare its performance to existing models, a new row need simply be added to a tab-separated config file. The README file in the config/ directory describes the acceptable ranges of values allowed in the config files, such as a description of what kind of models are compatible with the existing workflow.

Snakemake also makes extensive use of config files, and the config files described here are formatted such that Snakemake can utilize them when executing the workflow. So, to change the behavior of the workflow (again, within the expected range of uses), only config files need to be edited.

## 6. DATA STANDARDS

### 6.1. SOURCE SELECTION

Both code and data were integral components of this project and both required consideration for reproducible outcomes. To create an open inventory as a product we aimed to reuse and create data that aligned with the FAIR guiding principles (Wilkinson et al. 2016). The primary data source needed was bibliographic metadata. There are several commercial sources of bibliographic metadata such as Dimensions (Digital Science), Scopus (Elsevier), and Web of Science (Clarivate Analytics). However, these resources require a subscription which would limit others' ability to reproduce and reuse our workflow and neither are they openly licensed. Therefore, we opted to use the open metadata available from Europe PMC as the data source for creating the inventory. Although not as exhaustive as the commercial options mentioned, Europe PMC covers a large swath of the life sciences; as of October 2023, high quality, interoperable metadata, including titles and abstracts, was available for over 40 million articles. Additionally, Europe PMC offers robust and well-documented APIs that facilitate access and are especially useful for a reproducible pipeline. Although we know that some biodata resources will be missed due to articles being published outside of the ~4000 journals available in Europe PMC, we felt that this tradeoff was justified in order to optimize openness and reproducibility.

### 6.2. ADDRESSING DATA FINDABILITY AND ACCESSIBILITY

Depending on context, anyone interested in reusing the data from this project might wish to start at different points. We therefore offer multiple options. The exact query string we used can be rerun to obtain results from Europe PMC. Additionally, since bibliographic databases may change slightly over time (e.g. records added, removed, or corrected), query results themselves (PMID, title, abstract) may be of use to reproduce our results using the exact same data. There is also the labeled training data that was used to train the various models, a preliminary inventory



that is subjected to selective review by a curator, and, finally, the primary data product for this project is the final inventory itself. The query string, query results, training data, preliminary inventory, and the final inventory are all available within the project's GitHub repository and were archived for long-term preservation and persistent reference in an associated Zenodo deposition once the article was accepted for publication. Zenodo provides a DOI and relies on the DataCite metadata schema, which allows the dataset to be found within Zenodo's search interface, DataCite's central metadata store, and via internet search engines such as Google.

### 6.3. ADDRESSING DATA INTEROPERABILITY

For the final inventory, we retained unique article identifiers (PMIDs) to allow easy extraction of additional metadata or for access to the full text, when available, from either Europe PMC or PubMed Central. Additionally, we logged URL status codes per specification RFC 9110 (Fielding et al. 2022), extracted countries from author affiliations following ISO 3166 (ISO 3166 n.d.), and retained geo coordinates for IP address look-ups, when available. While it would have been ideal to include a persistent identifier for the biodata resources located (e.g., ROR ID or DOI), most resources do not have an identifier, which perfectly illustrates the challenge of trying to locate these resources in the first place.

### 6.4. ADDRESSING DATA REUSABILITY

In addition to the efforts towards interoperability described above, we also maintained a structured format throughout and used the CSV format for preservability and to ensure ease of reuse. These files are accompanied by a plaintext README file that includes a description of each variable as well as data collection details and licensing. By using open data from Europe PMC, we were able to release the data with CCO licensing, thus allowing the broadest reuse possible. Together, this documentation, the repository's Github history, and Zenodo's commitment to long-term archiving all provide provenance.

Finally, to further extend the potential for reuse, we plan to provide identified biodata resources to Europe PMC as community annotations. This will allow easy bulk access to the identified resources as well as their associated articles. The annotations can be used for several purposes, for example, mining articles with full text available or analysis of the intersection between these annotations and the many other annotation types available within Europe PMC.

## 7. EXTERNAL REVIEW/VALIDATION

In the Open Science Implementation Plan that we drafted (see Section 2 above), we also included a desire to have a party external to the team review the products of the study. Working within a team inherently provides a mechanism for internal feedback, but review by another person outside of the project helps reveal implicit knowledge developed during the project that would otherwise remain hidden to potential reusers. For example, team members may, without realizing it, adopt terms or abbreviations that are not well-known outside of the project.

This section of the Open Science Implementation Plan was not particularly well-developed beyond acknowledging that such a review would be ideal, as noted by others (Coburn and Johnston 2020; Heil et al. 2021), and that this role is included in the CRediT taxonomy (Allen et al. 2019). As we moved closer to having products finalized, we had a better sense of what sorts of reviews will be most valuable. We recruited an individual who reviewed the code and documentation in detail and ran nearly all the code available in the open archive. We budgeted 40 hours for this work, which was easily consumed given review effort required. Others may wish to allocate even more resources to this activity, which we found extremely helpful for identifying errors and pointing out gaps in our documentation. We formally acknowledge this effort here as well as in the associated article.

## 8. DISCUSSION

Here we have described the efforts that were taken to develop a methodology for obtaining and updating a biodata resource inventory with Heil et al.'s gold standard of reproducibility, a robust codebase, and complying with FAIR data standards.

## 8.1 FROM PRINCIPLES TO PRACTICE

We, and many others, are committed to Open Science and see the imperative of reproducibility. Putting these principles into practice on a complex project presented an opportunity for us to work through philosophical, organizational, and technical details. We were successful in meeting the goals outlined in the Open Science Implementation Plan established at the beginning of the project. Installation of dependencies and reproduction of the entire analysis can each be performed with a single command, and analysis steps are fully documented. All code passes static code checks for formatting, linting, and type compatibility. Much of the code was formally tested with unit and integration tests. The core data products, such as the labeled training data and preliminary inventory, are present in GitHub and in Zenodo, with accompanying documentation.

The methodologies used in this work are not novel on their own. Wherever possible, we looked to existing tools and practices. The automation employed to make reproduction simple relies on the widely used Snakemake workflow manager. It is also common practice in software engineering disciplines to leverage static code checking and testing as we have done. Regarding data standards, we looked to the FAIR principles. The purpose of this report is to provide an example of how a research project that utilizes computational methods, particularly ML, can be implemented to maintain robustness and strive for a high level of reproducibility. However, we recognize that there are numerous ways to accomplish this and do not mean to claim our implementation is failproof.

## 8.2 FROM DETAILS TO DECISIONS

When we began the project, we were especially interested in finding implementation details. How *exactly* does one make it possible to re-run an entire analysis with a single command? How *exactly* does one make data ‘interoperable’? Although we knew these details would be different in our case, concrete examples can provide clarity and inspiration. As the project progressed and we learned by doing, our questions evolved to focus on the choices that must be made. One example is the tradeoff of using only open data versus a more extensive commercial data source, which would likely have yielded a larger, but in our estimation less useful, inventory. Many of the trickiest decisions involved accounting for the diverse interests of, and the resources available to, potential reusers, now and into the future.

There were also ambitions that we had at the start of the study that are now future directions because we chose to devote time developing a robust workflow instead. This required principled project management and caused, even as we write this, some amount of wistfulness. In the end, we could not do it ‘all’, and we fully appreciate that others must decide for themselves where to place their efforts. Such decisions required us, and will require others, to devote a substantial amount of time to think through and implement. We were able to do this only because of our team’s collective belief that these efforts were worth the resources invested.

## 8.3 LIMITATIONS

Certain improvements could be made, such as using a more robust package manager like poetry and using git hooks to automatically run tests upon committing to git. Importantly, test coverage is lacking in some areas, especially for portions that involve heavy computation such as model training. Still, the current test coverage is enough to increase confidence in the code’s behavior. As Peng (2011) noted, ‘Given the barriers to reproducible research, it is tempting to wait for a comprehensive solution to arrive.’ Thus we thought our experiences may be helpful to share.

Possibly the greatest limitation, or threat to long-term reproducibility, was the decision to not use containers as a trade-off to be compatible with Google Colaboratory. In the current configuration, all dependencies are listed in a requirements.txt file and must be installed to run the code. However, it is possible that dependencies become unavailable or incompatible in future. Containers mitigate this problem by packaging all dependencies with the code, eliminating this concern.

A key consideration is how generalizable the efforts and methods toward reproducibility presented here are to other research projects, methods, and domains. Fortunately, most of the methods and tools here are not specific to natural language processing pipelines, and therefore

generalize well to most computational research tasks. For example, workflow managers such as Snakemake can be applied to data analysis pipelines in general. Additionally, the more conceptual steps, like creating the Open Science Implementation Plan at the start of a project, could be broadly applied.

## 9. CONCLUSION

Through articulating our goals early on and dedicating time and resources, we were able to accomplish our Open Science and reproducibility goals. Throughout this case study, we provided details on the steps we took to make the code clean and robust and the data FAIR. We invested considerable effort into ensuring reproducibility, with the intent that both the methods and outputs would be of use to us and others. Our first update of the inventory, initiated approximately one year after project completion, only required modification to the Colab notebooks to account for Google Colaboratory changes, but otherwise functioned as expected. With this promising, albeit early, success, we remain cautiously optimistic that the work is durable. By presenting our experiences, we hope this Practice Paper provides a helpful example for others to consider as they work to build greater reproducibility in their research.

## DATA ACCESSIBILITY STATEMENT

Code and data generated during the course of the project are archived in Zenodo along with associated documentation (<https://zenodo.org/doi/10.5281/zenodo.10105161>). The final inventory and associated data dictionary are available as a separate Zenodo deposit (<https://zenodo.org/doi/10.5281/zenodo.10105947>). Readers may visit HuggingFaceHub ([https://huggingface.co/globalbiodata/inventory\\_2022\\_all\\_models/tree/main](https://huggingface.co/globalbiodata/inventory_2022_all_models/tree/main)) to access the fine-tuned models. Additionally, all materials are available on GitHub, which may be updated after this publication ([https://github.com/globalbiodata/inventory\\_2022/](https://github.com/globalbiodata/inventory_2022/)). All other software used is openly available and shown Table 1.

NAME	DESCRIPTION	REFERENCE
conda	Package and environment management system	(Conda 2017)
flake8	Python linter (static code checking)	(Ziade and Cordasco 2011)
Make	Build automation tool, used here for creating shell command aliases	(GNU Make 1988)
Mermaid	Diagram generator for Markdown	(Sveidqvist 2014)
mypy	Static type checker for Python	(Lehtosalo 2012)
pip	Package manager for Python	(pypi n.d.)
pylint	Python linter (static code checking)	(Thénault 2001)
pytest	Python testing framework	(Krekel 2004)
pytest-flake8	Pytest plugin to run flake8	(Lockhert 2015)
pytest-mypy	Pytest plugin to run mypy	(Bader 2016)
pytest-pylint	Pytest plugin to run pylint	(Gee 2015)
renv	Dependency manager for R	(Ushey 2022)
snakefmt	Code formatter for Snakemake	(Hall and Letcher 2020)
Snakemake	General-purpose workflow manager	(Köster and Rahmann 2012)
styler	Code formatter for R	(Müller et al 2021)
yapf	Code formatter for Python	(Google Inc.)

Table 1 Glossary of Software.

## ACKNOWLEDGEMENTS

The authors would like to thank Ana-Maria Istrate with the Chan Zuckerberg Initiative for her contributions to developing the machine learning methods used in the project as well as CZI colleagues Dario Taraborelli, Donghui Li, and Gully Burns for their support and feedback on early

versions of the study. We also thank Ken Youens-Clark formerly at The University of Arizona, Alise Ponsoero at The University of Helsinki, and Bonnie Hurwitz at The University of Arizona for their mentorship of Kenneth Schackart. Additionally, we thank the Europe PMC team, especially Aravind Venkatesan, Mohamed Selim, and Melissa Harrison, for their guidance and expertise. Finally, we would like to acknowledge Jodie Forbes for detailed review of the associated code and documentation.

## FUNDING INFORMATION

This work was funded by the Global Biodata Coalition ([globalbiodata.org](https://globalbiodata.org)), a coalition of research funding organizations working towards sustainability of biodata resources worldwide.

## COMPETING INTERESTS

The authors have no competing interests to declare.

## AUTHOR CONTRIBUTIONS

KES – Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing

HJI – Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Validation, Writing – original draft, Writing – review & editing

CEC – Conceptualization, Data curation, Funding acquisition, Supervision, Writing – original draft, Writing – review & editing

## AUTHOR AFFILIATIONS

**Kenneth E. Schackart III**  [orcid.org/0000-0002-1658-3699](https://orcid.org/0000-0002-1658-3699)

Global Biodata Coalition, 12 quai Saint-Jean 67080, Strasbourg, France; Department of Biosystems Engineering, The University of Arizona, Tucson, Arizona 85721, USA

**Heidi J. Imker**  [orcid.org/0000-0003-4748-7453](https://orcid.org/0000-0003-4748-7453)

Global Biodata Coalition, 12 quai Saint-Jean 67080, Strasbourg, France; University Library, University of Illinois at Urbana-Champaign, Urbana, Illinois 31821, USA

**Charles E. Cook**  [orcid.org/0000-0002-4145-8048](https://orcid.org/0000-0002-4145-8048)

Global Biodata Coalition, 12 quai Saint-Jean 67080, Strasbourg, France

## REFERENCES

- Ahmed, H** and **Lofstead, J** 2022 Managing Randomness to Enable Reproducible Machine Learning. In: *Proceedings of the 5th International Workshop on Practical Reproducible Evaluation of Computer Systems*. New York, NY, USA: Association for Computing Machinery. pp. 15–20. DOI: <https://doi.org/10.1145/3526062.3536353>
- Allen, L, O’Connell, A** and **Kiermer, V** 2019 How can we ensure visibility and diversity in research contributions? How the Contributor Role Taxonomy (CRediT) is helping the shift from authorship to contributorship. *Learned Publishing*, 32(1): 71–74. DOI: <https://doi.org/10.1002/leap.1210>
- Bader, D** 2016 *pytest-mypy: Mypy static type checker plugin for Pytest*. Available at <https://github.com/realpython/pytest-mypy> [Last accessed 22 November 2022].
- Baggerly, K A** and **Coombes, K R** 2009 Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. *The Annals of Applied Statistics*, 3(4): 1309–1334. DOI: <https://doi.org/10.1214/09-AOAS291>
- Baker, M** 2016 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604): 452–454. DOI: <https://doi.org/10.1038/533452a>
- Berman, H M, Westbrook, J, Feng, Z, Gilliland, G, Bhat, T N, Weissig, H, Shindyalov, I N** and **Bourne, P E** 2000 The Protein Data Bank. *Nucleic Acids Research*, 28(1): 235–242. DOI: <https://doi.org/10.1093/nar/28.1.235>
- Bush, K A, Calvert, M L** and **Kilts, C D** 2022 Lessons learned: A neuroimaging research center’s transition to open and reproducible science. *Frontiers in Big Data*, 5. DOI: <https://doi.org/10.3389/fdata.2022.988084>
- Coburn, E** and **Johnston, L** 2020 Testing our assumptions: Preliminary results from the Data Curation Network. *Journal of eScience Librarianship*, 9(1). DOI: <https://doi.org/10.7191/jeslib.2020.1186>

- Conda** 2017 Available at <https://www.anaconda.com> [Last accessed 22 November 2022].
- Country Codes – ISO 3166** n.d. Available at <https://www.iso.org/iso-3166-country-codes.html>.
- de Saint-Exupéry, A** 1943 *Le petit prince* [*The little prince*]. Verenigde State van Amerika: Reynal & Hitchcock (US), Gallimard (FR).
- Fanelli, D** 2018 Is science really facing a reproducibility crisis, and do we need it to? *Proceedings of the National Academy of Sciences*, 115(11): 2628–2631. DOI: <https://doi.org/10.1073/pnas.1708272114>
- Ferguson, C, Araújo, D, Faulk, L, Gou, Y, Hamelers, A, Huang, Z, Ide-Smith, M, Levchenko, M, Marinou, N, Nambiar, R, Nassar, M, Parkin, M, Pi, X, Rahman, F, Rogers, F, Roochun, Y, Saha, S, Selim, M, Shafique, Z, Sharma, S, Stephenson, D, Talo', F, Thouvenin, A, Tirunagari, S, Vartak, V, Venkatesan, A, Yang, X and McEntyre, J** 2021 Europe PMC in 2020. *Nucleic Acids Research*, 49(D1): D1507–D1514. DOI: <https://doi.org/10.1093/nar/gkaa994>
- Fielding, R, Nottingham, M and Reschke, J** 2022 RFC 9910 HTTP Semantics. Internet Engineering Task Force. Available at <https://www.rfc-editor.org/rfc/rfc9910>.
- Figueiredo, L, Scherer, C and Sarmento Cabral, J** 2022 A simple kit to use computational notebooks for more openness, reproducibility, and productivity in research. *PLOS Computational Biology*, 18(9): e1010356. DOI: <https://doi.org/10.1371/journal.pcbi.1010356>
- Gee, C** 2015 *pytest-pylint: pytest plugin for running pylint against your codebase*. Available at <https://github.com/carsongee/pytest-pylint> [Last accessed 22 November 2022].
- GNU Make** 1988. Available at <https://www.gnu.org/software/make/> [Last accessed 22 November 2022].
- Google Inc.** *yapf: A formatter for Python files* 2004. Available at <https://github.com/google/yapf> [Last accessed 22 November 2022].
- Grüning, B, Chilton, J, Köster, J, Dale, R, Soranzo, N, van den Beek, M, Goecks, J, Backofen, R, Nekrutenko, A and Taylor, J** 2018 Practical computational reproducibility in the life sciences. *Cell Systems*, 6(6): 631–35. DOI: <https://doi.org/10.1016/j.cels.2018.03.014>
- Hall, M and Letcher, B** 2020 *Snakefmt: The uncompromising Snakemake code formatter*. Available at <https://github.com/snakemake/snakefmt> [Last accessed 22 November 2022].
- Haring, R and Bell, R J** 2018 Lack of research reproducibility, the rise of open science and the need for continuing education in research methods. *Climacteric*, 21(5): 413–414. DOI: <https://doi.org/10.1080/13697137.2018.1476968>
- Heil, B J, Crawford J and Greene, C S** 2023 The effect of non-linear signal in classification problems using gene expression. *PLoS Computational Biology*, 19(3): e1010984. DOI: <https://doi.org/10.1371/journal.pcbi.1010984>
- Heil, B J, Hoffman, M M, Markowitz, F, Lee, S-I, Greene, C S and Hicks, S C** 2021 Reproducibility standards for machine learning in the life sciences. *Nature Methods*, 18(10): 1132–1135. DOI: <https://doi.org/10.1038/s41592-021-01256-7>
- Hook, D W and Porter, S J** 2021 Scaling scientometrics: Dimensions on Google BigQuery as an infrastructure for large-scale analysis. *Frontiers in Research Metrics and Analytics*, 6. Available at <https://www.frontiersin.org/articles/10.3389/frma.2021.656233> [Last accessed 3 February 2023].
- Imker, H J and Schackart, K E** 2022 Open Science implementation plan for the biodata resource inventory. *Zenodo*. DOI: <https://doi.org/10.5281/zenodo.7392518>
- Imker, H J and Schackart, K E** 2023 Manual review process for the biodata resource inventory. *Zenodo*. DOI: <https://doi.org/10.5281/zenodo.7768363>
- Imker, H J, Schackart, K E, III, Istrate, A-M and Cook, C E** 2023 A machine learning-enabled open biodata resource inventory from the scientific literature. *PLOS ONE*, 18(11): 1–28. DOI: <https://doi.org/10.1371/journal.pone.0294812>
- Kaczmarzyk, J R, Gupta, R, Kurc, T M, Abousamra, S, Saltz, J H and Koo, P K** 2023 ChampKit: A framework for rapid evaluation of deep neural networks for patch-based histopathology classification. *Computer Methods and Programs in Biomedicine*, 239. DOI: <https://doi.org/10.1016/j.cmpb.2023.107631>
- Kim, Y-M, Poline, J-B and Dumas, G** 2018 Experimenting with reproducibility: A case study of robustness in bioinformatics. *GigaScience*, 7(7). DOI: <https://doi.org/10.1093/gigascience/giy077>
- Köster, J and Rahmann, S** 2012 Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19): 2520–2522. DOI: <https://doi.org/10.1093/bioinformatics/bts480>
- Krekel, H** 2004 *pytest: The pytest framework makes it easy to write small tests, yet scales to support complex functional testing*. Available at <https://github.com/pytest-dev/pytest> [Last accessed 22 November 2022].
- Leek, J T and Jager, L R** 2017 Is Most Published Research Really False? *Annual Review of Statistics and Its Application*, 4(1): 109–122. DOI: <https://doi.org/10.1146/annurev-statistics-060116-054104>
- Lehtosalo, J** 2012 *mypy: Optional static typing for Python*. Available at <https://github.com/python/mypy> [Last accessed 22 November 2022].
- Lockhart, T** 2015 *pytest-flake8: Pytest plugin to run flake8*. Available at <https://github.com/tholo/pytest-flake8> [Last accessed 22 November 2022].

- Merkel, D** 2014 Docker: Lightweight linux containers for consistent development and deployment. *Linux j*, 239(2): 2.
- Müller, K, Walther, L and Patil, I** 2021 *styler: Non-invasive pretty printing of R code*. Available at <https://github.com/r-lib/styler> [Last accessed 22 November 2022].
- Peng, R D** 2011 Reproducible Research in Computational Science. *Science*, 334(6060): 1226–1227. DOI: <https://doi.org/10.1126/science.1213847>
- Peng, R D and Hicks, S C** 2021 Reproducible Research: A Retrospective. *Annual Review of Public Health*, 42(1): 79–93. DOI: <https://doi.org/10.1146/annurev-publhealth-012420-105110>
- Perkel, J M** 2020 Challenge to scientists: does your ten-year-old code still run? *Nature*, 584(7822): 656–658. DOI: <https://doi.org/10.1038/d41586-020-02462-7>
- pypi** n.d. *Python Package Index – PyPI*. Available at <https://pypi.org/> [Last accessed 22 November 2022].
- Sveidqvist, K** 2014 *Mermaid: Generation of diagrams like flowcharts or sequence diagrams from text in a similar manner as markdown*. Available at <https://github.com/mermaid-js/mermaid/> [Last accessed 22 November 2022].
- The Europe PMC Consortium** 2015 Europe PMC: a full-text literature database for the life sciences and platform for innovation. *Nucleic Acids Research*, 43(D1): D1042–D1048. DOI: <https://doi.org/10.1093/nar/gku1061>
- Thénault, S** 2001 *Pylint: It's not just a linter that annoys you!* Available at <https://github.com/PyCQA/pylint> [Last accessed 22 November 2022].
- Ushey, K** 2022 *renv: Project Environments*. Available at <https://rstudio.github.io/renv/> [Last accessed 6 January 2023].
- Walters, W P** 2020 Code sharing in the Open Science era. *Journal of Chemical Information and Modeling*, 60(10): 4417–4420. DOI: <https://doi.org/10.1021/acs.jcim.0c01000>
- Wanner, J, Cuellar, L K, Rausch, L, Berendts, K W, Wanke, F, Gabernet, G, Harter, K and Nahsen, S** 2023 *nf-root: A best-practice pipeline for deep learning-based analysis of apoplastic pH in microscopy images of developmental zones in plant root tissue*. *bioRxiv*, 2023.01.16.524272. DOI: <https://doi.org/10.1101/2023.01.16.524272>
- Wilkinson, M D, Dumontier, M, Aalbersberg, Ij J, Appleton, G, Axton, M, Baak, A, Blomberg, N, Boiten, J-W, da Silva Santos, L B, Bourne, P E, Bouwman, J, Brookes, A J, Clark, T, Crosas, M, Dillo, I, Dumon, O, Edmunds, S, Evelo, C T, Finkers, R, Gonzalez-Beltran, A, Gray, A J G, Groth, P, Goble, C, Grethe, J S, Heringa, J, 't Hoen, P A C, Hooft, R, Kuhn, T, Kok, R, Kok, J, Lusher, S J, Martone, M E, Mons, A, Packer, A L, Persson, B, Rocca-Serra, P, Roos, M, van Schaik, R, Sansone, S-A, Schultes, E, Sengstag, T, Slater, T, Strawn, G, Swertz, M A, Thompson, M, van der Lei, J, van Mulligen, E, Velterop, J, Waagmeester, A, Wittenburg, P, Wolstencroft, K, Zhao, J and Mons, B** 2016 The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1): 160018. DOI: <https://doi.org/10.1038/sdata.2016.18>
- Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L and Teal, T K** 2017 Good enough practices in scientific computing. *PLoS Computational Biology*, 13(6): e1005510. DOI: <https://doi.org/10.1371/journal.pcbi.1005510>
- Wolf, T, Debut, L, Sanh, V, Chaumond, J, Delangue, C, Moi, A, Cistac, P, Rault, T, Louf, R, Funtowicz, M, Davison, J, Shleifer, S, von Platen, P, Ma, C, Jernite, Y, Plu, J, Xu, C, Le Scao, T, Gugger, S, Drame, M, Lhoest, Q and Rush, A** 2020 Transformers: state-of-the-art Natural Language Processing. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics. pp. 38–45. DOI: <https://doi.org/10.18653/v1/2020.emnlp-demos.6>
- Ziade, T and Cordasco, I** 2011 *Flake8: Your tool for style guide enforcement*. Available at <https://github.com/PyCQA/flake8> [Last accessed 22 November 2022].
- Ziemann, M, Poulain, P and Bora, A** 2023 The five pillars of computational reproducibility: bioinformatics and beyond. *Briefings in Bioinformatics*, 24(6). DOI: <https://doi.org/10.1093/bib/bbad375>

#### TO CITE THIS ARTICLE:

Schackart III, K E, Imker, H J and Cook, C E 2024 Detailed Implementation of a Reproducible Machine Learning-Enabled Workflow. *Data Science Journal*, 23: 23, pp. 1–14. DOI: <https://doi.org/10.5334/dsj-2024-023>

**Submitted:** 09 December 2023

**Accepted:** 08 April 2024

**Published:** 29 April 2024

#### COPYRIGHT:

© 2024 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See <http://creativecommons.org/licenses/by/4.0/>.

*Data Science Journal* is a peer-reviewed open access journal published by Ubiquity Press.