Journal of Data Science 0 (0), 1–10
??? 2024
DOI: 10.6339/24-JDS1153
Computing in Data Science

RCloud – Collaborative Visualization and Analysis Platform

Simon Urbanek[1]

[1]*Department of Statistics, University of Auckland, Auckland, New Zealand*

Abstract

The last decade has seen a vast increase of the abundance of data, fuelling the need for data analytic tools that can keep up with the data size and complexity. This has changed the way we analyze data: moving from away from single data analysts working on their individual computers, to large clusters and distributed systems leveraged by dozens of data scientists. Technological advances have been addressing the scalability aspects, however, the resulting complexity necessitates that more people are involved in a data analysis than before. Collaboration and leveraging of other's work becomes crucial in the modern, interconnected world of data science. In this article we propose and describe an open-source, web-based, collaborative visualization and data analysis platform RCloud. It de-couples the user from the location of the data analysis while preserving security, interactivity and visualization capabilities. Its collaborative features enable data scientists to explore, work together and share analyses in a seamless fashion. We describe the concepts and design decisions that enabled it to support large data science teams in the industry and academia.

Keywords *cloud computing; collaboration; data analysis; data science; distributed computing; reproducible research; visualization*

1 Introduction

Over the last decades the availability of data has been steadily increasing (Hilbert and López, 2011), opening more opportunities for analyses and solutions to real problems. Beside advances in methodology, this evolution has also changed the nature of the interaction with the data. Historically, it was more common for an individual statistician to perform analyses on a local computer in isolation. However, nowadays many datasets are stored in large data lakes and analyses are performed in collaboration with other statisticians, data scientists and domain experts. It is therefore important to provide tools that allow both the flexibility necessary for a data analysis, but also at the same time allow collaboration of possibly remote team members with varying levels of familiarity with the tools.

In addition to considering single projects, many larger organisations have entire teams of data scientists, possibly in different parts of the company that would benefit from leveraging each other's work (Gupta and George, 2016). Therefore it would be highly beneficial to have a tool that enables discovery and exploration of existing analyses such that similar analyses can be leveraged by different team members in different parts of the organization. Ideally, reproducibility should be guaranteed (Sandve et al., 2013) such that when a user discovers an interesting analysis, they can re-create it to understand it and modify it for their own purpose.

We are proposing a collaborative open-source platform RCloud which seeks to address such needs. In the following we will describe the important design decisions that set it apart from

Received July 31, 2023; Accepted September 6, 2024

other existing data analytic user platforms, describe some main implementation features and modular design which makes its application very versatile and scalable.

Several tools are well-known for enabling data analysis in various languages, probably the best known among R (R Core Team, 2022) users is RStudio (RStudio Team, 2020) while among Python (Van Rossum and Drake, 2009) users it is Project Jupyter (Kluyver et al., 2016). Both were designed with the single-user single-machine mindset. Both offer a layer on top of the original software which allow multi-user setup to a degree by running sessions for separate users, but sharing and collaboration was not the design goal. More detailed comparison is provided later in Section 3.4 in light of the methodology and design described in the following sections.

2 Methodology

RCloud evolves around the idea that all analyses are open and can be shared among all users such that anyone can see the code and run it to replicate the results, conditional on data access permissions. The platform aims to make it easy to perform data analyses in an environment that supports visualization and interactive graphics, while enabling collaboration, reproducibility and exploration of other user's work.

The smallest entity of work is a "notebook", a concept that can be traced back to Knuth (1984). A notebook consist mainly of "cells" which may be written in different languages or formats such as R, Python, shell, Scala (Odersky et al., 2008) or markdown. RCloud provides a web interface which starts a session for the user, who can create, modify and run a notebook. All operations (creating cells, editing, re-ordering, renaming) are recorded in a fully version controlled system which is modular, but in most cases backed by git (Chacon and Straub, 2014). Notebooks can be organized in a hierarchical tree structure, supporting arbitrarily complex projects. Figure 1 shows a screenshot of a typical RCloud session in a browser.

The collaborative nature of RCloud is that the notebook tree is global, so users can explore each other's notebook trees. All notebooks can be explored and run by any other user. A discovery feature also enables thumbnail-like exploration of most recent notebooks. For large organisations random exploration may be tedious, so we also include a search feature which ranges from simple word searches to complex queries based on the Lucene Query Syntax (The Apache Software Foundation, 2020a) which can involve constraints such as search based on code in a particular programming language.

Other user's notebooks are read-only initially, but can be executed by all users. Unlike Jupyter Notebook, results are not implicitly cached to avoid leakage of confidential data. The execution of a notebook guarantees that the user's access permissions will be checked as part of the process. If a user wants to modify a notebook, they can "fork" it, which makes a linked copy in their own tree which can be edited – similar to the concept of forking on GitHub (GitHub, 2020). This is a key feature that allows other users to quickly modify parameters of an analysis, re-use it for other datasets or expand it in alternative different directions. User can also fork their own notebooks in a similar fashion to explore different ideas. RCloud tracks changes between forks and allows merging of work back to the original notebook if desired, very analogous to the model of merges and pull-requests in GitHub.

Another major feature of RCloud is a modular access control as integral part of the platform. The user is authenticated in the system which allows access to the notebooks and determines the rights on the nodes running the back-end. In addition, this system also includes key management

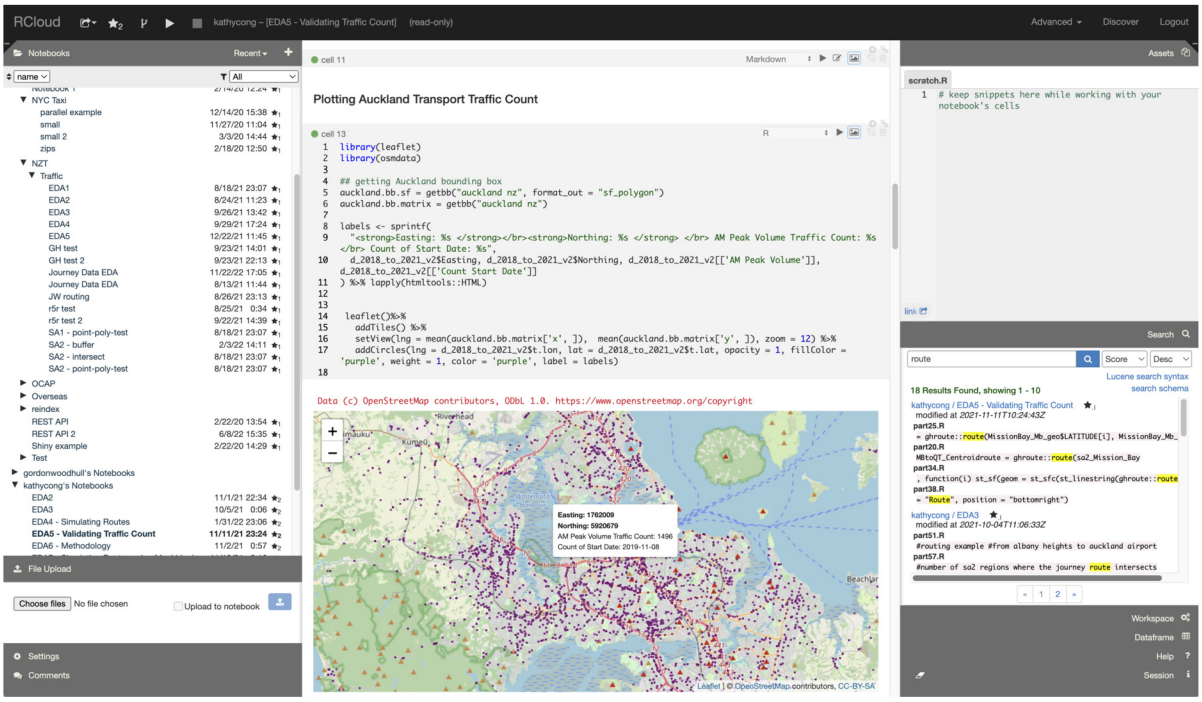

Figure 1: Screenshot of an active RCloud session in a browser tab. Notebook tree is on the left, notebook with its cells in the center and auxiliary panes such as the asset editor, search and data viewer on the right.

which enables seamless use of encryption. Although the notebook store is public, it is possible to encrypt notebooks such that only individual users or specific groups of users can use them. This can also be extended to data, enabling automatic encryption and decryption on the fly where data never rests on disk in unencrypted form, which is crucial for sensitive datasets.

Finally, due to its design, RCloud allows easy deployment of projects (sets of notebooks) as in a container as a service. This is particularly useful when creating data analytic visualizations and dashboards as it makes their deployment very simple.

3 Design

RCloud has been designed as a very modular platform which can be used in very versatile ways: as a single application on a laptop for one user or deployed across an entire cluster of machines or VMs in enterprise setting supporting hundreds of users. Key to this is the separation of the various services in the platform which have declared APIs and thus allow multiple implementations. The following are the key parts of the platform:

- *Notebook store* is the version-controlled storage of all content. For local use we offer an implementation where each notebook is a git repository, for multi-user applications GitHub can be used as back-end (including authentication), or we offer an open-source implementation compatible with the GitHub Gist API which in turn uses git repositories under the hood.
- *Metadata store* is a key/value database (typically using Redis (Redis, 2020) or files) which makes it fast and easy to locate notebooks, users and sessions.

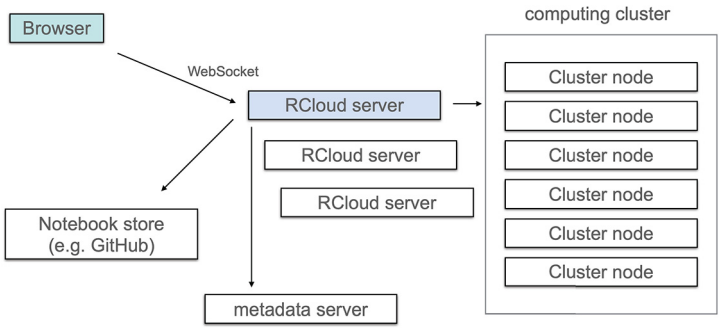

Figure 2: RCloud architecture illustrated on a multi-node enterprise deployment.

- *Search engine* is a Solr-based (The Apache Software Foundation, 2020b) server that facilitates search in the entire instance or even across multiple instances. It is optional if lightweight deployment is desired.
- *Authentication* service manages the login credentials, mapping to user accounts on the instances and keys for encryption and decryption.
- *Compute* instances, which can be bare-metal machines, VMs or even docker containers, provide the computing power for the code run in the notebook. They can be scaled dynamically and load-balanced. They can run R and any Jupyter kernel such as Python, Scala, shell and similar.

Figure 2 shows the architecture overview of the components. The user uses a web-browser to connect to an *RCloud server* instance using the standard secure HTTP protocol which is upgraded to a WebSocket connection to enable fully interactive communication between the compute node and the user. One tab in the browser then corresponds exactly to one live session. The RCloud server is based on Rserve (Urbanek, 2003) and itself manages communication to all other components including the notebook store, metadata server, search or auxiliary services such as computing cluster or databases. To provide scalability, there can be multiple RCloud server nodes: a suitable one (e.g., with least live connections) is automatically picked at connection time. In addition, RCloud nodes can be dynamically created and disposed of to adjust according to the need.

The *metadata server* provides global services such as authentication, search and RCloud Storage (RCS) – a global key/value store. The individual services can be run in separate nodes or containers. Upon connection first actions typically involve the metadata server: authenticate the user and (if successful) query RCS for initial information such as last opened notebooks and user's notebook tree structure. Then a session is created and either an existing notebook is loaded (typically the last one for returning users) or new notebook is created (for an entirely new user).

The *Notebook store* can be thought of as a repository holding all contents and history of a notebook. A notebook consist of *cells* and *assets*. The latter are simply arbitrary files that can be included in a notebook – they may be notes, images or even datasets. They don't have any special semantic meaning as far as the platform is concerned, but can be searched, displayed and edited. Cells, however, are ordered, each holding a piece of code or documentation. "Executing" a notebook means to run all its cells in their order, sequentially, and to collect the output – typically print out or graphics. Each cell can be written in another language. The most typically used languages are R, Python and shell, but RCloud provides an extension mechanism by which support for additional languages (or even groups of languages) can be added without any changes to the platform.

All interfaces within RCloud are managed as R packages which makes it possible to use arbitrary protocols or implementations for any of the services. For example, interactions with the Notebook store are encapsulated in the `gist` package which defines a common interface. The name "gist" is rooted in the history of the implementation: the first RCloud prototype used GitHub Gists as the notebook storage. Later re-implementation introduced the modular interface. Two different implementations of the interface are provided in the RCloud sources: `gitgist` uses a collection of bare Git repositories as the notebook storage. This is practical for single-machine environment or for container deployment of notebooks as a service. Alternative `githubgist` package uses the GitHub Gist REST API protocol to connect to GitHub or compatible services (such as open-source gist-service), suitable for multi-user deployments. Any other interface can be used without changes in the platform itself.

The benefit of the modular approach is that components can be shared or accessed across multiple RCloud instances. For example, a research organization may want to deploy a separate instance from a production organization to separate their run-time environments, yet both can share the search index and access each other's notebooks.

3.1 Interactivity

One core design aspect of RCloud was to fully leverage the direct connection to the web-browser, which opens enormous possibilities with respect to visualization and interaction. Most web-based user interfaces allow for web-aware content to be generated and served, but RCloud goes far beyond that by enabling a fully bi-directional communication between R and JavaScript. The underlying implementation allows us the execute any R function from JavaScript and any JavaScript function from R. Function arguments are automatically translated between the two languages. This allows the user to natively and interactively call JavaScript code from R, for example, to create interactive maps. Conversely, any interactions in the browser can call back into R to perform additional computations. Unlike serving static content, everything in the browser can be dynamically created and modified, including the loading of new JavaScript code and interfaces. We leverage this capability in the very implementation of RCloud: the entire user interface is driven by direct communication between R and the browser. All user interactions, such as renaming a notebook, result in R function calls in the RCloud session, which allows for highly modular implementation of the back-end.

Due to its web-based nature, the platform not only supports analysis, but inherently enables the creation of dashboards and sharing of results in reports. It natively supports interactive visualizations including htmlwidgets (Vaidyanathan et al., 2021) in R.

3.2 Security

The overall design has been focused on security. The entire platform opens only one single HTTP/HTTPS port to be connected to by the user which means it integrates easily with reverse proxies or Single Sign On platforms. The protocol used for communication with the session is based on the Object Capability model (Miller, 2006) as implemented in Rserve where only capabilities in accordance with the user permissions are granted successively. Only one capability is provided upon connection which is `authenticate`. It means that no other operations can be performed until the authentication is successful and the `authenticate` function alone determines which capabilities should be exposed to the user.

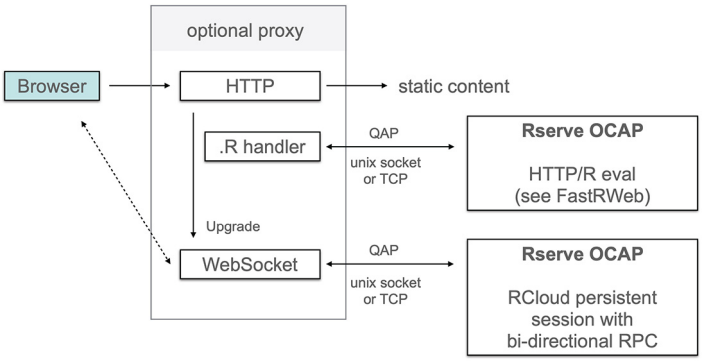

Figure 3: Architecture of the forwarding service securing the RCloud web interface.

Any fully functional data analytic platform has to allow developers to write and run arbitrary code, because the analyst should be able to fully leverage the data analytic language of choice. To guarantee provenance, we want to make sure that no code can be executed without being properly authorized and recorded. For that purpose all user code is first committed to the version controlled Notebook store irrevocably, which also verifies authorization and ownership. The back-end will not execute any code directly, cells can only be executed by a reference to the commit thus eliminating any unauthorized code execution. The need to commit any changes before execution provides security guarantees, but also introduces an overhead. This overhead generally does not affect the usability: using an Internet connection, separate compute and notebook service machines the round-trip is typically less than 100 ms from the browser initiation to receiving the commit identifier.

For additional security, RCloud allows for computing services to be separated such that they are not listening to a network port, but instead are bound only to a local, named socket, which prevents network attacks. Figure 3 illustrates the architecture of that setup. Any static content is served directly without the involvement of R code. R scripts are served by a separate process which handles any part of RCloud that does not require a persistent connection. Finally, a separate process is connected to the WebSocket connection which constitutes the live session with bi-directional interface between the browser and R. Both R processes communicate using a local socket and are thus isolated from potential network attacks.

3.3 Use

The main goal of RCloud was to facilitate collaboration and de-couple the location of the user from the machine running the analysis in a secure manner. Individual collaborators can be operating anywhere in the world, the only requirement is an Internet connection and a browser. In addition, the platform allows immediate access to analyses where anyone can pick up any existing work and continue to work in a different direction. In our daily work, this has proven to be extremely valuable as analyses are no longer restricted to static output, but instead it allows us to make quick changes interactively, explore ideas and new threads in a working meeting between collaborators. This worked surprisingly well in an earlier prototype (North et al., 2015) even before the recent shift to remote work where such capability became crucial. This also extends to the ability to simply send RCloud URLs to others – it is not necessary to generate reports or views, simply sharing the notebook link proved to be sufficient and very effective due

to the fact that the recipient is not limited to viewing, but can explore the results by writing additional code immediately.

Finally, the social aspect of the platform has connected teams across organizations. Once aware of the capabilities, data scientists started to use the search and discovery functions to see how certain technologies or models are used, expanding their horizon and fostering new collaborations. The visualizations capabilities enabled the creating of dashboards which can be used by non-technical users.

3.4 Related platforms

RCloud was conceived at a time when RStudio and IPython (Pérez and Granger, 2007) (later Jupyter) were becoming popular user interfaces in the R and Python community. RStudio was the first web-based integrated development environment (IDE) for R with focus on providing a unified experience across platforms. Similarly, IPython started as an interactive interface to Python and evolved into a web-based form with the introduction of the web-based notebooks. Both projects were fundamentally focused on a single user interaction with a computing engine. RStudio is mostly run locally on user's computer as an application. Although a server version exists, it is merely a frontend to a single-user application with the same interface. IPython has envisioned remote use to leverage computing power of remote machines, but it became most well-known for local use given the control the user requires over their Python environment.

Neither system provided way to share or organize the work beyond single documents. RStudio was extended to use R Markdown files to enable reproducibility, allowing the combination of code and documentation. IPython has introduced IPython Notebook format with similar purpose except that it is also capable of storing results. Unlike R Markdown it is an internal exchange format and never intended to be edited by the user directly. IPython has eventually spawned the Jupyter project which has enhanced some of the capabilities to support multiple languages. In either case it is left to the user to manage the storage, access and lifetime of a document. External tools or file systems are have to be used.

Due to a different focus and target audience neither RStudio nor Jupyter is offering support for user management or advanced security. Local instances are accessible to anyone and server instances are guarded by a single entry login without specific authorization levels. Jupyter includes signing in the wire protocol (Jupyter Development Team, 2015) to detect tampering as well as abilities beyond one-on-one communication, but neither system prevents the user from removing all evidence of a potential attack thus making provenance tracking difficult. RCloud manages several access levels from anonymous access to published services, authenticated read-only notebooks all the way to owner's read-write access. This also enables the platform to automatically manage encryption keys and sensitive content during execution.

The primary goal of RCloud is to foster collaboration and sharing. The fundamental concept was to abstract out the notion of a document in a way that is not limited to files, so that the platform can be used to manage analytic code, documents and sharing. This also requires that user management, authentication and security have much more central role in RCloud as they is part of document access management. The abstraction also enabled RCloud to de-centralize notebook storage and allow inter-connection of multiple instances (such as between different organizations or companies). This also means that RCloud had to provide services not available elsewhere, such as discovery (shared multi-user notebook tree, filtering, history) or cross-instance search.

All three projects are converging in a common space while they still keep their distinct focus, but useful features are assimilated. For example, RCloud defined its own modular interface to support various languages in notebook cells, but eventually it became clear that it is more efficient to implement an interface to Jupyter kernels. Although this limits the language capabilities in that case to a subset supported by Jupyter, it means that new languages modules don't have to be created specifically for RCloud if they already exist for Jupyter.

The difference in philosophy of RCloud as a data analytic platform in contrast to RStudio as a development environment makes RStudio unique in that it also supports tools for R package development. Since R packages rely heavily on file-based approach, is more intuitive to use locally and therefore better supported in RStudio. On the other hand the tighter integration of notebooks in RCloud allows it to automatically generate services such as Docker containers based on notebooks, something RStudio does not support.

Under the hood all platforms use HTTP and WebSocket protocols to exchange information between the browser and the computing session. Jupyter and RStudio use JSON text as message payload while RCloud leverages binary Rserve QAP protocol which is suitable for large data as it does not require parsing of the payload. All three systems support web-based output during the analysis. RStudio's htmlwidgets only support uni-directional communication from R to the browser. In addition to htmlwidgets, RCloud allows the user to leverage the same seamless bi-directional OCAP-based communication between JavaScript and R which is the foundation of its own implementation. Jupyter Widgets are an extension of Jupyter which allows callbacks into the kernel using the model-view-controller approach as opposed to direct interaction.

The modularization of RCloud components means they can live in separate containers or virtual machines, which allows it to distribute load and scale capacity elastically. It also makes it easier to replicate results as the environment is centrally managed by the platform. However, this is in contrast to more recent developments such as JupyterLite (Tuloup et al., 2021) or webR (Stagg and Henry, 2024) where the computing is performed in the browser as opposed to remote computing services. Such approaches are easier to deploy as no computing servers are required. RCloud currently does not support such approach, but given that RCloud's services such as notebook store and authentication are based on REST protocols it is conceivable to use such remote services from a webR session to enable both approaches to co-exist.

4 Discussion

In this article we have described a collaborative web-based data analytic platform RCloud, its design principles and modular implementation. The platform was designed by data scientists for data scientists. The platform is been using in daily work which has driven the feature set and allowed us to re-design aspects of the platform until it was most effective for collaborative data analysis. It has enabled increasing number of people with different backgrounds to collaborate and create tools for their daily work.

All components of RCloud have been released as open source to invite extensibility and integration. The modular nature of the platform makes it very flexible depending on the needs of the users. For a single analysts using a laptop the deployment would constitute of one RCloud server with local flat-file RCS and git notebook storage. On the other end of the spectrum, an enterprise deployment may consist of a swarm of dynamically scaled RCloud servers, gist notebook storage service, authentication server and Redis RCS. The central orchestration of the platform allows for easy bindings to Hadoop clusters and Database Management Systems

where credentials and configuration settings can be automatically managed in encrypted form, secured by the authentication service. For example, sessions can automatically authenticate with Kerberos (Garman, 2003) as part of the initialization.

The platform is released in its entirety as open source at https://github.com/att/rcloud. It has been proven in a large organisations supporting entire data science teams. It is also being used in academic teaching. We hope that its availability as free, open source enables others to contribute and expand the capabilities.

Supplementary Material

- Source code repository and documentation: https://github.com/att/rcloud
- Public instance and tutorials: https://rcloud.social

Acknowledgements

RCloud is a large team effort and many thanks go to everyone involved. The development has been initiated at AT&T Labs - Research, based on the works of Carlos Scheidegger and Simon Urbanek. The user interface, interactive graphics and language integrations have been created by Gordon Woodhull. Tutorials and demos were provided by Jo Frabetti. Parts of the platform have been developed by Mango Solutions. Many thanks to everyone at the Labs who supported, used and tested RCloud.

Funding

Current work and public instance is supported by the University of Auckland and the Centre for eResearch.

References

Chacon S, Straub B (2014). *Pro Git*. Apress.

Garman J (2003). *Kerberos: The Definitive Guide*. O'Reilly Media.

GitHub (2020). https://github.com/.

Gupta M, George JF (2016). Toward the development of a big data analytics capability. *Information & Management*, 53(8): 1049–1064.

Hilbert M, López P (2011). The world's technological capacity to store, communicate, and compute information. *Science*, 332: 60–65.

Jupyter Development Team (2015). Messaging in Jupyter.

Kluyver T, Ragan-Kelley B, Pérez F, Granger B, Bussonnier M, Frederic J, et al. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (F Loizides, B Schmidt, eds.), 87–90. IOS Press.

Knuth DE (1984). Literate programming. *The Computer Journal*, 27: 97–111.

Miller MS (2006). Robust composition: Towards a unified approach to access control and concurrency control, Ph.D. thesis, Johns Hopkins University, Baltimore, Maryland, USA.

North S, Scheidegger C, Urbanek S, Woodhull G (2015). Collaborative visual analysis with RCloud. In: *2015 IEEE Conference on Visual Analytics Science and Technology (VAST)*, 25–32.

Odersky M, Spoon L, Venners B (2008). *Programming in Scala*. Artima.

Pérez F, Granger BE (2007). IPython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3): 21–29.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Redis (2020). https://redis.io/.

RStudio Team (2020). *RStudio: Integrated Development Environment for R*. RStudio, PBC., Boston, MA.

Sandve GK, Nekrutenko A, Taylor J, Hovig E (2013). Ten simple rules for reproducible computational research. *PLoS Computational Biology*, 9(10): e1003285.

Stagg GW Henry L, (2024). webr: The statistical language R compiled to WebAssembly via Emscripten.

The Apache Software Foundation (2020a). Apache Lucene. https://lucene.apache.org/.

The Apache Software Foundation (2020b). Apache Solr. https://solr.apache.org/.

Tuloup J, Tandon M, Renou M, Beier T (2021). Jupyterlite: Wasm powered Jupyter running in the browser.

Urbanek S (2003). Rserve – a fast way to provide R functionality to applications. In: *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*.

Vaidyanathan R, Xie Y, Allaire J, Cheng J, Sievert C, Russell K (2021). *htmlwidgets: HTML Widgets for R*. R package version 1.5.4.

Van Rossum G, Drake FL (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.