Collaborative Visual Analysis with RCloud

Stephen North, Carlos Scheidegger, Simon Urbanek and Gordon Woodhull

Fig. 1. An overview of features in RCloud. RCloud is an environment in which *hackers* and *scripters* solve problems in data and visual analytics and create web applications and services. It is a shared environment, in which *discoverability* is a concern. RCloud supports search, annotation, recommendation, and commenting for all notebooks, and provides an overview where users can *browse* popular and recent analyses. When problems and data sources change frequently, deployment can become very costly; RCloud supports *transparent and automatic* deployment of analyses as web pages and web services, allowing a seamless transition from data exploration to production.

Abstract— Consider the emerging role of data science teams embedded in larger organizations. Individual analysts work on loosely related problems, and must share their findings with each other and the organization at large, moving results from exploratory data analyses (EDA) into automated visualizations, diagnostics and reports deployed for wider consumption. There are two problems with the current practice. First, there are gaps in this workflow: EDA is performed with one set of tools, and automated reports and deployments with another. Second, these environments often assume a single-developer perspective, while data scientist teams could get much benefit from easier sharing of scripts and data feeds, experiments, annotations, and automated recommendations, which are well beyond what traditional version control systems provide. We contribute and justify the following three requirements for systems built to support current data science teams and users: *discoverability*, *technology transfer*, and *coexistence*. In addition, we contribute the design and implementation of RCloud, a system that supports the requirements of collaborative data analysis, visualization and web deployment. About 100 people used RCloud for two years. We report on interviews with some of these users, and discuss design decisions, tradeoffs and limitations in comparison to other approaches.

Index Terms—visual analytics process, provenance, collaboration, visualization, computer-supported cooperative work

1 Introduction

More than a half-century ago, Tukey foresaw much of what is now commonplace in data analysis [34]. Powerful, interactive environments for analysis and programming were the goal, with unswerving insistence on visualization as a central part of the discovery process. His now-famous quip that "the picture-examining eye is the best finder we have of the wholly unanticipated" has come to define much of visual analytics and exploratory visualization [35].

Computing has moved far beyond what Tukey imagined. Processing and networking capabilities today far exceed what was barely imaginable then. There has been a lot of work on distributed computing frameworks, and social computing technologies for publishing, searching and sharing information are common on the internet. But how we *develop* analytic solutions has not changed much. Although environments such as RStudio include many features to graphically interact with a local workspace, they still follow the metaphor of terminal, text editor, and source files stored in file systems.

The work of data scientists in teams is also changing. Project teams vary in size, from just a few people to dozens or more. Assignments are broad and include tasks such as problem identification, data wrangling, modeling, analysis, visualization, summarization, presentation and interpretation of results, and recommending actions to help clients to realize the benefits of the analysis. Knowledge or software prototypes created by these teams are transferred to other organizations to employ them in production. This work depends extensively on communication and collaboration. At almost every step, collaborators share detailed

- *Stephen North is with Infovisible LLC, Oldwick, USA, and graphviz.org. Email: s.c.n@ieee.org*
- *Carlos Scheidegger is with the University of Arizona, USA. Email: cscheid@email.arizona.edu*
- *Simon Urbanek is with AT&T Labs in Bedminster, USA. Email: urbanek@research.att.com*
- *Gordon Woodhull is with AT&T Labs in Bedminster, USA. Email: gordon@research.att.com*

Manuscript received 31 March 2013; accepted 1 August 2013; posted online 13 October 2013; mailed on 4 October 2013.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

knowledge about tasks, data and code. Further, data scientists are increasingly asked to work closely with less technically-oriented business or domain specialists.

Previous work has identified important areas for improvement in visual analytics systems to address this situation.

1. Discoverability. A chronic complaint of data analysis teams is inefficiency due to a lack of transparency in accessing the work of other team members. The processes and technologies supporting data analysis and visual analytics today are fragmented. Knowledge is shared imperfectly through informal conversations, emails, meetings, phone calls, source code, wikis and project documents. This causes work to be repeated unnecessarily ("How can I connect to database X?", "How do you get clean data from column Y from feed Z?") due to lack of communication about previously solved problems. Our goal is support and promote transparency of work between team members.

2. Technology transfer. In most organizations, development of analyses and visualizations is done by *hackers* and *scripters* (adopting Kandel et al.'s terminology [20]). *Application users* benefit from tools developed by hackers and scripters. Results of experiments are often first shared by copying screen shots or static output. By the time decisions are made based on those screenshots, the data and processes may have changed so much that the decision can be wrong. Moreover, the connection between EDA and deployment are made by porting or rewriting code so it runs as a stable service. In an environment where business needs can change rapidly, this process does not scale. It is also difficult to trace problems from production back to EDA. Our goal is to merge the worlds of EDA and deployment, to close the gap.

3. Coexistence. In current ecosystems, the isolation of exploratory visualization and data analysis environments hinders wider adoption of modern techniques from each. The richness of interactive visualization tools is still somewhat separated from the power of statistical programming environments. There is an opportunity to provide a framework so that developments on each side are more easily adopted by the other, and made available in production services.

Fortunately, there is evidence that appropriate technology might solve some of these problems. Gutierrez's interviews with data scientists [12] include the following quotes:

- (upon being given access to other code and data analysis, in order to learn about Hadoop) "I could look at other peoples code and play with their code and data sets as well"
- (discussing the value of sharing prototypes rather than static data) "Prototyping our products so that internal customers can use them early on has been crucial for our success. Now we can shoot off a URL to internal customers and it allows them to provide feedback way before we're talking about getting it into production."
- (on sharing more than just a finished product) "We also share exploratory analyses and reports so that we can still exchange knowledge even if the work didn't make it into a larger project."
- (on changing analyses and processes) "It is important to have testing frameworks so you can go back and test all of your data."

In order to tackle these issues, we created RCloud, a software environment supporting the end-to-end visual analytics process for individuals and teams and their work within larger organizations. In this paper, we contribute the design of RCloud, and an interview study based on the course of its development and deployment at AT&T Labs for about three years.

RCloud knits together some familiar tools, and provides new features to find data and code; create experimental notebooks; run experiments; annotate and deploy experiments as end-user websites or as reusable, callable services; and to share, search and recommend these artifacts. The artifacts are stored in a version control system that provides a common workspace, as well as needed control and isolation between stable and experimental versions of code and other resources.

One of the key design criteria for RCloud was *transparency*: for the most part, the novel features are available and usable by default, without explicit involvement on the part of data scientists and other customers. Visualizations are shared and turned into production websites without moving or porting code; all artifacts present in the system can be searched at all times; recommendation is as easy as clicking a star on a useful workbook.

The high level architecture of RCloud is shown in figure 2. We chose R as the foundation because it is already the dominant statistical computing language in our lab. R also has many useful packages for data analysis and visualization, and the core system and its packages are open source that can be modified to support research.

Over the past three years, we prototyped RCloud and deployed it to a community of data scientists, business analysts and other colleagues. The platform today has over 300 active accounts; about 20 people use it regularly, another 30 people use it more than once a week, and another 50 use it more than once a month. Most of the active users are members of AT&T Labs, but some are data scientists and collaborators in other business units.

2 Related Work

Previous work has identified key requirements for supporting the visual analytics process, including by multiple users.

Social Data exploration and Analysis ManyEyes [37] was a landmark system for the crowdsourced creation and publication of data visualizations. Although ManyEyes supported only a fixed set of visual encodings, the system's success was an early indication of the potential to combine social media with visual analytics.

Notebooks as a medium for data analysis dissemination The concept of a "notebook" as we use it can be traced to Knuth's literate programming [21]. In literate programming, a prose description of the behavior of a program is "weaved" with its source code, yielding both an executable program and a human-readable document. A notebook represented as a collection of small executable cells originated with Mathematica, and in R, literate programming is supported by packages such as knitr and RMarkdown [39]. Project Jupyter [26] (originally part of IPython) offers some notebook features, though it lacks transparent multi-user sharing and deployment. RStudio is highly acclaimed for its polished and powerful human interface. Although RStudio offers publication of literate R programs as a free service on its website, the workflow is somewhat disconnected from the development of those programs. Once they're uploaded, it's hard for others to build off the work published, or even for the original author to update new versions. In other words, RStudio handles *publication*, but not *deployment* and sharing. Further afield, Electronic Lab Notebooks organize and share data from scientific lab experiments[29].

Provenance and versioning As mentioned, one central issue in exploratory analysis is that problems may change quickly, often while in the course of developing a solution. As a result, systems need to provide adequate support for tracking *changes* of data analysis scripts. VisTrails was an early system for managing *process provenance*, that demonstrated the value of capturing aspects of the processes that surround data analysis experiments and tools, including detailed history, collaboration, and deployment [7].

VisMashup [30] defines a schema and semantics for automatically deriving user interfaces from workflows, while Crowdlabs exposes these capabilities on a website featuring workflow upload and remote execution [22]. In our view, the impedance mismatch between dataflow pipeline specifications and powerful general-purpose languages is too great for exploratory work by data science teams. At the expense of ease of use for non-programmers, RCloud provides a closer match for analysts familiar with R and Python, while retaining attractive properties like transparent provenance tracking and interactive data visualization on the web.

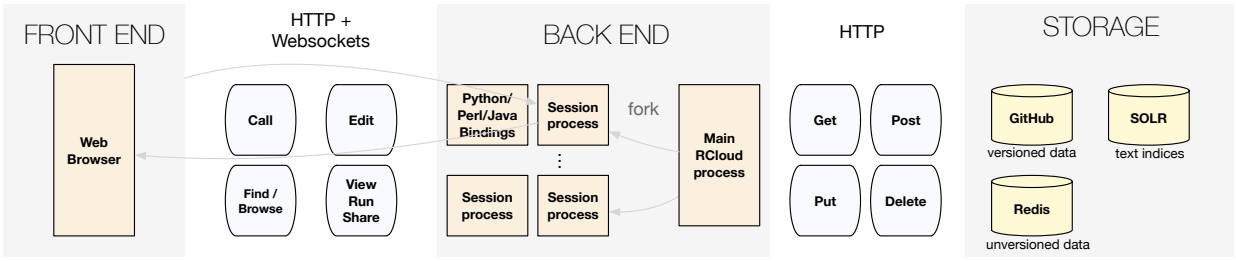


Fig. 2. A diagram of RCloud's architecture. RCloud was specifically designed to leverage existing systems and standards, and so communication between most parts of the system happens through HTTP. As a result, some desirable features of RCloud become natural, such as native support for RCloud notebooks as web services.

Web-based tools for sharing code snippets Quite a few tools were recently developed for quickly sharing small programs on the web, including bl.ocks [5], jsfiddle [1], and plot.ly [2]. bl.ocks and jsfiddle are designed to share Javascript programs, which means that deployment happens automatically through the web browser. This provides a seamless way to share visualization techniques, but it does not help with analysis. Plot.ly is notable in that it provides API support for publishing *from* scripts: it is possible to generate a plot.ly visualization within another program. Although this is an intriguing idea, it nevertheless creates a disconnection between the analysis and the resulting visualization. In RCloud, we wanted to ensure that every visualization is transparently linked to the source code that generates it.

Needs of data analysts Kandel et al.'s interview study points out the typical "explore", "model", "report" cycle in enterprise data analysis [20]. There are many discontinuities in this cycle that cost time and effort to overcome. Kandel et al also point out that larger teams are becoming more common in data analysis, that supporting collaboration is difficult and important, and that sharing and versioning of data sources and artifacts is hindered by current technology. "We found that analysts typically did not share scripts with each other. Scripts that were shared were disseminated similarly to intermediate data: either through shared drives or email. Analysts rarely stored their analytic code in source control."

An earlier study by Kandel et al argues that data wrangling (cleaning, parsing and transformation)is a major part of exploratory analysis and visualization [19]. Although attacking a different semantic level from ours, it shows the need for an environment that enables better sharing of knowledge, tools and processes for wrangling. Anecdotally we find much frustration among practitioners that this knowledge is difficult to find and often is not recorded or available in a reusable form even within the same organization.

Heer and Agrawala identify many design considerations for collaborative visual analytics [13] that influenced our work. RCloud notebooks, and the integrated version control system for them, described in Section 3.1, address *modularity and granularity*, and *artifact histories*. *Starring*, the means for signaling interest in notebooks, described in Section 3.2, addresses social-psychological incentives, recommendation, and voting and ranking. RCloud's integrated deployment mechanism, described in Section 3.3, addresses the cost of integration, content export, presentation and view sharing.

There has been noteworthy work on specific techniques to support collaborative or social code development and data analysis, such as social bookmarking [23] [14] and crowdsourcing [8]. Similarly, there are computational methods to support high performance execution in incremental code development environments [11]. The goal of RCloud is to define an environment in which many such techniques may be integrated and made available to a broad community.

One high-level goal is to reduce the gap between implementers and deployers in visual analytics. The fusion of development with production operations in software release management ("DevOps" [15] or "continuous integration" [9]) is a trend in web services and similar fields. When programming teams recreate the work of data scientists to deploy it in production, there is a high cost in time, expense and accuracy. By making it convenient for data scientists to take experiments into production, we may be able to eliminate the need for this costly step.

3 The System

Our design takes advantage of existing web software as much as possible. HTTP, despite its deficiencies, is the lingua franca of distributed interprocess communication. By speaking HTTP natively, our system provides a low-friction path from experimental data analysis scripts to automated *web services*: essentially a remote function call over the web which can be invoked by higher-level tools.

In consequence, the entire high-level infrastructure of RCloud, shown in Figure 3, uses web standards. The communication between a web browser and an active R session as a user edits a notebook is performed by a combination of HTTP and Websockets, while all other IPC is done through HTTP, from notebook versioning in GitHub to building and maintaining full-text indices through SOLR. The most novel aspect of RCloud's runtime system is its tight integration between the web client and the backend R process, described in Section 3.4.

3.1 Notebooks

The main unit of computation in RCloud is a *notebook*. A notebook holds a sequence of *cells*, each of which is a snippet of code or hypertext in Markdown. As mentioned in Section 2, executable documents like this are a feature of previous systems, including Mathematica, IPython and Sage.

Notebooks can be executed one cell at a time in an interactive session, similar to traditional read-eval-print loops, or can all be executed concurrently, similar to running a shell script.

One of the main contributions of RCloud is the notion that notebooks are "always deployed". In other words, the most recent version of a notebook is immediately available to all other users of the system. Another way to describe this is that RCloud lacks a "save" button: any notebook cell that runs is always associated to a notebook version serialized to disk. Although one of our interview subjects reported that this sometimes leads to an excessively fine-grained sequence of versions (see Section 5), losing information that could be shared is worse (see Section 5 user comments on satisfaction with low-friction shareability).

In this design, we get notebooks that are always live, but sometimes broken. Because stability is important, we also allow any previous version of a notebook to be tagged and referenced. Our notebook scheme is similar in many ways to models like Jankun-Kelly et al.'s p-set calculus [17] and VisTrails's version tree [7], where every change in the state of the system is tracked.

RCloud's implementation of the versioning mechanism is built on top of GitHub's *gists* [10], which are an HTTP interface for simplified repositories. The GitHub web-service API provides most of the semantics we need for the versioning portion of the storage back end: access to previous versions, comments, starring, and forking. This provides the additional benefit that every RCloud notebook can also be manipulated like a git repository, granting advanced users access to features like command-line checkout and history editing.

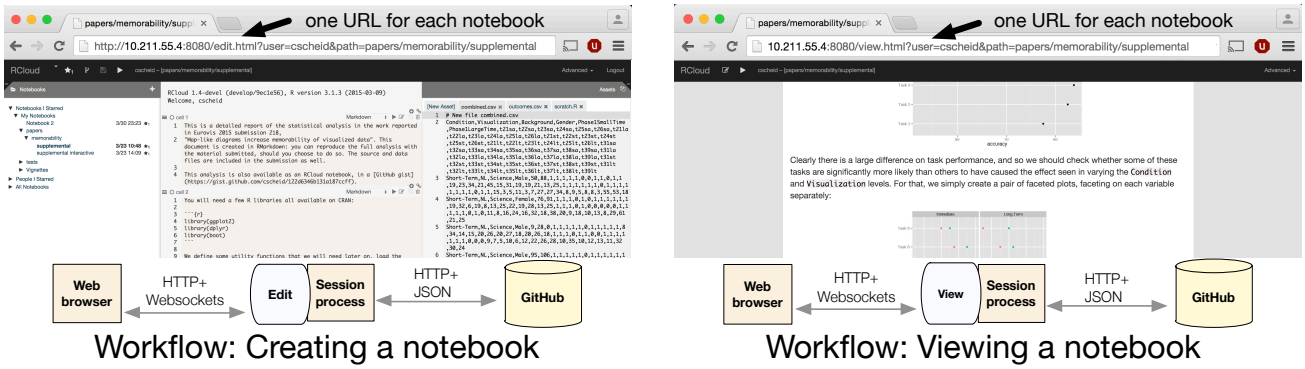


Fig. 3. An RCloud notebook is a sequence of *cells*, each a snippet of source in one of the supported languages (typically R, but Python and others are also supported) or Markdown. The main creation workflow involves editing notebooks, which are transparently stored as git repositories in GitHub, providing us with easy access to primitives for version tracking. Notebooks can be executed as they're edited (left), or in a standalone viewer (right), via a slightly different URL. This provides a lightweight, low-friction mechanism for sharing results which we discuss in detail in Section 5.

3.2 Reputation and Interest: starring

A side benefit of centralizing the execution and storage of notebooks is that it becomes feasible to collect usage information that is lost in conventional environments. In the case of social data visualization platforms, we would like to exploit usage data to help analysts find content of interest, whether that content is source code or data. Standard ways to achieve this are through *user-generated curation* and *automatic recommendations*.

Automatic recommendations have become famous in the user experience provided by companies such as Amazon and Netflix ("If you liked that film, you'll like this one too"). To make such recommendations, we need users to *curate* the collection, by providing explicit reviews or some other method of indicating interest. We incorporate both explicit and implicit indications of interest in notebooks. Explicit interest is indicated by "starring," or clicking on a button that marks a notebook as interesting. This makes explicit indication of interest a nearly trivial operation, always available and easy to use. RCloud today uses only simple counts of stars to measure overall notebook interest, mostly because standard recommender system algorithms require reasonably large training sets to pay off. Nevertheless, the starring mechanism is sufficient to create personalized recommendations [16].

Implicit signaling of interest is supported by keeping click-through counts [18] and execution counts. In addition to these standard techniques, we hope to apply static and dynamic code analysis to infer fine-grained information about relationships, for example, which packages and data sets often appear together, in the style of Codex [8].

3.3 Deployment of notebooks

Every notebook in RCloud is named by a URL, and notebooks by default are visible in the entire organization. This is deliberate. As pointed out by Wattenberg and Kriss [38], broad access to analysis outputs (in their case, for NameVoyager) increases long-term engagement in part through cross-references on the web. Although our prototype RCloud deployment is only visible inside a corporate intranet, we nevertheless found support for this notion by discovering links to RCloud notebooks in internal discussion fora and mailing lists. In addition, as we describe in Section 5, users have almost unanimously adopted "share-by-URL" as their default communication mechanism, as opposed to "share-by-screenshot", which we consider to be an encouraging validation of the system.

3.4 Executing R through a web browser, and Javascript through an R process

As mentioned, the other main goal in RCloud was to provide full access to the R statistical programming language during the *development* of a data analysis notebook. At the same time, when notebooks are *deployed* (and potentially accessed by anyone with a web browser), we'd like to allow the browser to invoke only a very limited subset of R, namely those notebooks that have been published.

The solution we developed is simple and general, and was directly inspired by Miller's *object capabilities* [24]. The R layer that communicates with Javascript does not expose unprotected evaluation of arbitrary functions. Instead, every function that the R layer intends to expose is associated with a large, cryptographically-safe random number (a "hash"). This random number is then sent across the wire and interpreted as an opaque function identifier. Because these identifiers are cryptographically safe, all the Javascript layer can do is send them to the R side, in a message requesting a function call. The result of this function call might include *new* opaque identifiers, exposing new "capabilities" to the client.

The same idea of exposing functionality via hashes can be used to give the server-side of RCloud access to Javascript functions. This allows R libraries to request user input in the browser, giving them access to features ranging from password prompts, to the currently selected set of points in an interactive visualization.

As a result, the features required to provide safe access to R from the client, and Javascript access from the R side, also enable full two-way communication between the languages. This provides considerable flexibility, so that for example, a chart built with dc.js or leaflet.js can call analysis functions in R without having to define an additional protocol between the processes. Calls in either direction look like ordinary function calls.

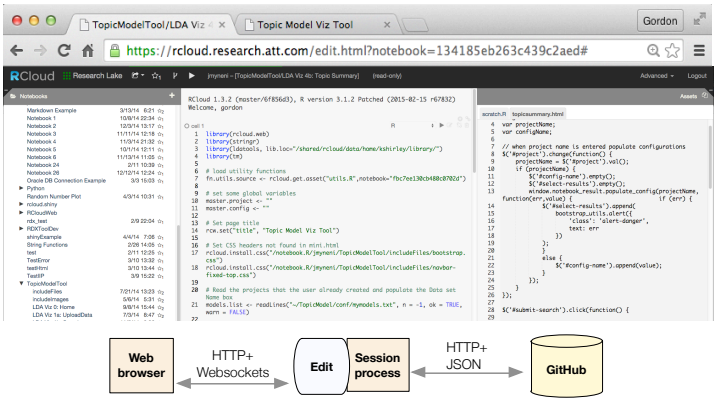
4 Case Study: Visual Exploration of Topic Modeling

LDAVis helps non-experts to explore collections of short text documents, using topic modeling and visualization. Topic modeling [4], although powerful, often requires human guidance and interpretation [32].

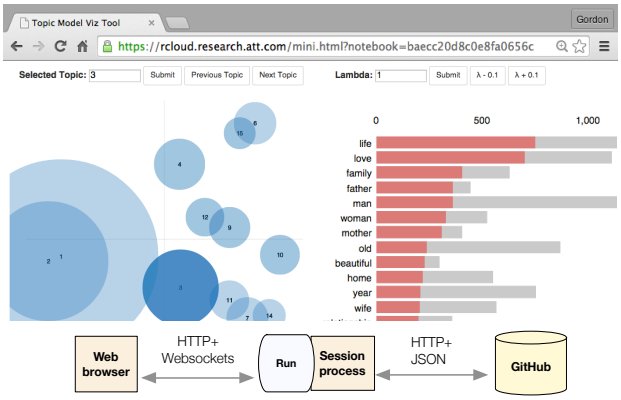
LDAVis was developed by two technical staff members at AT&T Labs, originally in RStudio Shiny [28], a framework for developing web applications in R. While Shiny offers outstanding ease of development, discoverability and deployment are also very important aspects in an application's lifecycle (see Section 5). RCloud's assumption that *all developed notebooks are automatically deployed* simplifies this for LDAVis.

LDAVis combines text analysis, dimensionality reduction and interactive visualization. Text analysis is performed by combining a standard R library to fit LDA models using Gibbs sampling with custom developed R code. The analysis module is a single R function exposed to the web application via the Javascript-R RPC mechanism described in Section 3; thus analysis is performed remotely on an RCloud server.

Each text topic is a probability distribution over the all the words of a document. To expose patterns in the relationships between topics,



Topic visualization application, inside RCloud editor



Topic visualization application, deployed

Fig. 4. LDAVis, an example application developed and deployed entirely in RCloud. LDAVis allows analysts to visually explore *topic models*, statistical representations of different topics in a document collection.

LDAVis combines interactive visualization and dimensionality reduction, allowing users to adjust measures for topic distances and the choice of dimensionality reduction technique. The dimensionality reduction algorithms and distance measures are implemented in R, which means they can be executed on RCloud servers as well.

The result of the dimensionality reduction process is a two-dimensional plot of the topic space, an example of which is shown in Figure 4. The interactive view is implemented in SVG and Javascript through D3 [6]. One of the most popular web-based visualization libraries for R is ggvis [27], so it is natural to ask if ggvis could have been used instead of custom Javascript. In this case, the interactive features of ggvis and Shiny are a subset of Vega's [33]. Custom interactions in LDAVis (like hovering over a topic, topic cluster, or word) are not available yet in Vega [33], although the required components were recently described by Satyanarayan et al. [31]. Although LDAVis required custom Javascript, that flexibility is welcomed by many web developers.

LDAVis demonstrates some unique features of RCloud. While RCloud notebooks allow deployment of analyses over the web with no extra effort, RCloud *applications* are more powerful, and are developed by combining Javascript and HTML for the front end. This requires more expertise than Shiny, but the RCloud model makes analysis simpler (since analysts simply write R in the style they already know) and the visualization front end is simpler for web developers (since they simply write Javascript in the style *they* know). LDAVis inherits the automatic deployment and discoverability of all RCloud applications.

5 Evaluation

To evaluate the effectiveness of RCloud, we informally interviewed 13 current or recent users of RCloud. 9 are data analysts, and 4 build tools for data analysts and business needs. Users felt that sharing works well, but the notebook interface slows down exploration, and the browsing interface is not scaling well.

Sharing of results is the core feature of RCloud, and a popular one. All the subjects praised this feature.

By default, all RCloud notebooks are publicly visible, and notebooks can be found by navigating the notebook tree, or by searching. However, users most often mentioned sharing by sending links through email.

Besides providing a way to present work, notebook sharing become a starting point in coding. Lilo says, "The best part is how easily you can share code. You can find a working example, rather than wearing out Google and finding questionable examples that may or may not work." Wendy notes, "[If] some person has done something similar, then you're able to just edit that, and that's saved a lot of work for me."

Eric develops packages for analysts, and uses RCloud "for sharing code with other people, and for doing tutorials for *iotools* or *hmr*", his packages. "I want people to see how the package works, so I clearly want them to see the code... I write it just like I would write GitHub Markdown, where you have little code snippets and text, but RCloud lets me actually run the snippets [and display the results]." He also uses RCloud for describing and debugging data sources.

Some users, who tried RCloud and were not able to continue for organizational reasons, miss certain capabilities. Leith "[likes] being able to create notebooks and share them... A wiki is not the best way for communicating results - it's like writing a blog post with very limited functionality. I have to save every picture and post it as an image." He adds, "I can't share all of the code because it would just get crowded and wouldn't look right on a wiki." Iris explains, "If I could make a folder on RCloud and have Python notebooks and also Pig notebooks there, and execute them from RCloud, that would be much better than my current [environment], because that would free me from manual documentation and version control and also telling people where my code was. It would be just, hey, go look on RCloud, here's the stuff."

Forking The ability to fork someone's notebook and continue their work proves to be a useful feature, but it is a lousy way to change simple parameters. Almost twice as many (131) users have forked someone else's notebook as have starred one (75). Lilo says, "It's one of the handiest functions, because instead of having to find it, copy, paste it, you just hit Fork, rename it, and it's done. It's pretty amazing."

Although we intended forking to be available to improve others' code, we didn't anticipate users forking their own notebooks, which proved very useful. Kenyon says, "I fork my own notebooks because I'm going off and doing some other analogous project, so I've got interesting content that I've already done in a previous analysis, that I want to start from and then tweak to match a new set of data."

Forking also provides a way for others to troubleshoot when something goes wrong. When Hugh collaborates with users of his notebooks, "I'll teach people to intercept the result in the middle, to insert print statements here and there and check values."

A common but ill-advised use of forking is to change parameters. Eric complains that a notebook might say "'This is a report of the volume of all of our feeds for this month', and someone would want to look at it for the next month or the previous month, so they'd fork it to change the month."

Parameters can be added to a notebook URL, but adding user interface elements to do the same thing takes more expertise. Tool builders told us it should be easier. Allison explains: "[Users] can always fork the notebooks and make changes, but I feel that if the owner of the notebook [adds a user interface], it's easier to run. Instead of forking it, if they can set options, it's probably more efficient, and they can [still] fork it if they want to."

Automatic source control is appreciated for its safety and ease, but the large number of commits can be hard to manage. Lilo says, "Instead of looking back and saying I've got a billion files here in this subdirectory and I hope I've got them backed up, if they're on RCloud I know they are."

Iris points out that automatic versioning works well for coping with the details of web development:

> I like the fact that it has a built-in editor, so if you need to fix a typo in a link, or an extra line break or other nonsense, you can just switch to the edit view, pull up your asset, type something, it's automatically saved, committed, everything. You don't have to go back to your source code, change it, commit it to the repo, pull the repo to your distribution version.

On the other hand, saving every change leads to many fine-grained versions. Kenyon says that for this reason, the history feature is not that helpful: "I don't need something that keeps track of every mistake I've made or every direction I've tried."

Discovering others' work Users reported that the search function is more helpful for learning about particular functions than techniques. Wendy says, "The fact that we have all the notebooks there, searchable, saves me from replicating what other people have done."

But other users prefer to browse just the notebooks of experts they know. Kenyon says, "Usually I know somebody's notebooks that I want to search through, because the kind of thing I'm looking for is something more obscure than I'm likely to find in some random person's." More selective ways to search will be needed as the number of notebooks grows further. We mention possible approaches in Section 6.

Integrated analysis Integrating an analysis language into a web development environment is something tool developers really appreciate. The structure of Hugh's visualization notebook means

> the other guys who want to do analytics on the data can first pull the data, do the analytics on it, and then feed the viewer the data. Anyone from the stats group can insert something in between. Once you get the data into RCloud, then you have a dataframe to work with, and then they can produce another dataframe.

Integrating R makes Allison's application development easier: "Having an open session where I can run R commands or functions without having to invoke an API or send a request and then wait for the response is extremely helpful in writing the application."

Web is limiting Although most analysts appreciated RCloud's sharing features, it was not a popular tool for exploratory data analysis. All analysts with R experience preferred to do analysis in another tool, then paste code into RCloud for sharing, because the web interface got in the way of their work.

The web interface of RCloud isn't satisfactory to Kenyon: "It's nice to have it saved, but there's this trade-off between it making it easier for me to present something or to save something, and my ease of typing and correcting and things in a plain editor window."

Coby also works with text files and commands, rearranging source files so active code is at the top. RCloud does not readily support this workflow, so Coby often shares work by pasting code into an RCloud notebook after he's done.

Working in a shared environment also entails compromises about what you can install. Joy says installing alternative or nonstandard packages is intrinsic to exploratory data analysis. This is a weakness in shared environments like RCloud.

Impermeable cells RCloud's notebook interface combines editable Markdown with a command line interface. Many users felt that this format, so effective for presentation, is incompatible with exploratory data analysis.

Much of the time, commands typed at the R command line serve only a transitory purpose, so having RCloud persist commands in cells can be annoying. Coby notes, "It saves everything I do like everything is gold, but most of it is junk not meant to be saved."

Material that is not appropriate to save includes "expressions that allow me to check that I'm the right track" (Kenyon), "checking out what your data is, or you make a plot of the data. Things that should not really become part of a notebook, but things that help you understand your data better" (Wendy).

Users felt that cells do not capture the right level of granularity. Kenyon says that RCloud's cell structure "tempts me to type a big long thing and then run the whole thing, as opposed to typing a few little pieces and then put them together" as he would on the command line. When Eric uses the R command line, he "copies and pastes 5-10 lines of code, so when something breaks, I get an error message on that one line, and I can up-arrow and change it and fix it, whereas in RCloud I have to run a whole cell, so the only way to get that functionality is if every line's in one cell."

Too vertical The vertical orientation of a notebook also poses problems not encountered on the command line, where typically only the last few page of output matters.

Scrolling between code and results is time consuming, and some users would prefer to keep results separate from code. Gerrard thinks RStudio's layout is more helpful because charts are shown in another pane that stays in place while doing analysis. Wendy says "Cells are really useful, [but] you want to see your output in a different window or on a separate part of the window".

The cell structure also can be problematic if some cells take a long time to execute. High variance is common: cells may take anywhere from seconds to hours to run. In this situation, the Run Whole Notebook button is dangerous. Coby reports that when converting his work to notebooks, he ends up with a lot of comments saying "This cell takes a long time to run." To avoid this pitfall, some users write code to explicitly cache results.

A sea of notebooks RCloud is becoming a victim of its own success, as it is becoming difficult to navigate all the notebooks. There are over 5200 notebooks in the research instance, of which more than 500 have been starred, and more than 350 have been forked.

For this reason, Kenyon doesn't find the notebook tree satisfactory:

> I don't necessarily need to see everybody's notebook that uses RCloud. Every time I do something new, I get a new notebook, so now I have 50 or 60 notebooks. That's enough to think about just on my own, but if everybody has 50 or 60 sitting on my display, it's more than I want to know about.

Although RCloud promises an environment where notebooks should keep working, not all our users have learned the habits that make this a reality. As Joy puts it, "bitrot" is still a big problem – notebooks often stop working because the user changed the structure of their data, or changed a filename or a database. Even if one forks someone else's notebook and corrects it, the original notebook still exists with the error. She says we need "organizational protocols" to catch up with the technology.

Eric, who writes packages and example notebooks for them, mentioned the accumulation of dead notebooks. When Eric and Hugh work together on a notebook:

> There's no way for both of us to have ownership of a notebook, so the only way is to fork it back and forth, and so we have dozens of old copies. We end up deleting all the old ones, but people still have links to them, because they don't actually disappear.

Eric tried to keep his notebook tree well organized, but this didn't help, because people kept old links in email, or forked notebooks which had become obsolete. He says he became scared to share notebooks: "Do I want to support this forever?"

	V/F	C	D	ML	R	A
RCloud	x	x	x	x	x	x
RStudio					x	x
JSFiddle	x	x				
bl.ocks	x	x			x	
shiny			x		x	x
Jupyter				x	x	x
Tableau		x	x		x	
ManyEyes		x	x			

Table 1. Summary of features for similar systems. The columns stand for, respectively, Versioning/Forking, Collaboration, Deployment, Multi-language support, Integrated Reports, and Integrated Analysis.

6 Discussion, Lessons and Limitations

6.1 Reflection on Design Considerations

Experience with deploying RCloud in a community of data analysts ("hackers") gave us some insight into whether the proposed requirements were appropriate, and were met. Our experience underscored the relevance of Heer and Agrawala's design considerations, and indicated areas for further exploration [13]. We adopt their taxonomy in the following discussion.

Shared artifacts and artifact histories are a central feature of RCloud, through shared workbooks. We observed that hackers readily share experiments, demonstrate techniques, publish results to peers and managers, and transfer algorithms to other groups. Artifact histories can be accessed through the notebook tree or through GitHub's web interface. But it is easy to create a large number of artifacts and the system does not help enough to organize and navigate them.

Modularity and granularity Modularity, or the ability to partition work into independent units, is a key to working productively in teams. If the units can be kept small or granular, team members can realize benefits at least proportional to their work on the units. RCloud's notebook and versioning allow work to be divided into units as fine as the underlying language allows, and encourage making incremental changes at low cost and without disrupting the work of others.

View sharing, bookmarking Most resources in RCloud are named and accessed as URLs. This proved to be effective for sharing analyses and integrating them with external processes. It is particularly advantageous for work to be shared as URLs that provide access to live notebooks, instead of by pasting screenshots into reports.

Discussion Annotation and commenting was another central goal. Commenting is supported through GitHub, but our hackers found it awkward and did not exercise it as much as we expected. An interesting question is, to what extent should application users be able to make annotations in published notebooks without coding and being exposed to the hackers's view? The design of more elaborate, integrated annotation remains as future work.

Content export is not a capability we aimed at supporting, and R already has many packages for this. Recently, due to popular demand, we added user interface support for exporting plot images. RCloud notebooks should play well in the R ecosystem, but sometimes it is difficult to know whether adding a feature for compatibility will offload complexity, or increase it.

Social-psychological incentives and **voting and ranking** are supported through starring and forking. These mechanisms are employed often on the platform. An obvious next step would be to enhance recommendations using relationships discovered by static and dynamic code analysis. This may be considered both within and across collections of scripts (the latter being similar to VisTrails' enhanced recommendations by clustering multiple workflows). It seems helpful to help users find which packages are frequently used together, or when using record types or data feeds. Overall, simple, passive mechanisms to collect data for recommendations are preferred.

Group management, size and diversity This area needs better support, and is clearly important to working in teams. We rely on external administrative processes and social conventions to manage accounts and groups. RCloud could benefit greatly from integrating social media to track identities and groups and to maintain communication channels, instead of having its own isolated solution.

Curation Even without formal group management, users often organize groups of related notebooks using tree folders. We found this particularly effective in collecting and distributing training materials.

6.2 Limitations

Because we developed our ideas while simultaneously creating a prototype, we did not foresee some of the requirements that emerged after people started working with RCloud.

Versioning and Upgrading Some aspects of the environment are more difficult to manage in RCloud than in conventional systems. RCloud does not separate its development and deployment environments, and every version of every notebook is shared by default. Although this encourages collaboration, it also quickly exposes misconfigurations, mismatches between package versions, and programming errors that can unintentionally affect production websites. Versioning of software components must be managed in several levels: in the notebooks themselves, in the installed libraries and packages of the R environment, and in the external environment such as the operating system, libraries, and protocols spoken by remote services.

Security Because RCloud allows arbitrary R code to be run *by design*, it's not straightforward to protect deployments from running unauthorized arbitrary code. RCloud uses an object capability model [24] recently added to the Rserve protocol [36]. Object capabilities are opaque handles to remote procedure calls: web browsers never directly instruct the RCloud backend to execute arbitrary code, preventing unauthenticated clients from making unauthorized calls to the RCloud runtime environment. Our back-end environment, on the other hand, assumes a high degree of trust between users: access control is delegated to the host operating system and web server. Most operating systems rely either on coarse permission models, which tend to be ineffective, or on detailed access control lists, which tend to be cumbersome. Information security in RCloud today is an unsolved matter. More sophisticated approaches are open research topics [25].

Nonforgetful, effortless exploration is hard Users reported that RCloud's interface clashes with the typical use of R for exploration. In current R practice, the command prompt is used to try things out. It is very fast and there is often no commitment: nothing gets saved by default, and clean up is automatic. In contrast, in RCloud, cells generated during exploration are saved in notebooks. We arrived at this from an insistence on reproducibility: no command should be run without being saved. In our experience, some history steps are only found to be important in hindsight, and deleting history by default makes it too easy to lose those steps by accident.

We found that when users realize that *scratch work is getting saved*, its sheer volume becomes burdensome. Clearly, RCloud is an extreme, and the right solution is probably in the middle. An intriguing possibility, suggested by one of our interviewees, is to implement auditing of data analyses as it existed in S [3], tying data artifacts to the processes that generated them. Bookkeeping would be reduced to selecting the *data* to be kept, and the code that generated it would be derived and saved transparently.

Discovery is hard With hundreds of users and thousands of notebooks, a global notebook tree becomes unwieldy. The search function does not protect searchers from obsolete and erroneous notebooks, which by default are retained forever. In addition, important metadata (has this notebook been recently run, or edited?) is not easily searchable. In other words, *textual search* is easy, but *relevance search* is what we need. Allowing users to *tag* notebooks is natural, but relies on users making a conscious effort, which goes against our ideal of not asking the user to do work for the system. Automatic *curation* of

software artifacts generated by teams seems to be an important avenue for future work.

Collaboration is hard While forking provides a simple way to continue someone else's work, teams working together generate workflows for which forking is not ideal. Although popular, forking poses a problem when dozens of versions of a notebook proliferate. A fully general solution seems intractable. Nevertheless, we expect future versions of the notebook infrastructure to move toward git's ideal of decentralized code repositories, instead of the exclusive ownership mode RCloud has now.

7 Conclusions and Future Work

We designed, implemented and deployed a prototype visual analytics environment for data exploration, sharing, presentation, and publishing. It is a step toward practical "DevOps for data science" and reproducible, publishable data science experiments.

RCloud was readily accepted as a platform for deployment of visualizations and interactive exploration tools inside our organization. Notebook sharing and publishing were eagerly adopted. On the other hand, features for single-user data exploration that compete with existing mature tools, were not readily accepted. In the future we hope to study and understand the degree to which this reluctance is caused by bad design decisions on our part, and how much comes from "mere" change aversion. It is also clear we need need better ways to manage the persistence of notebooks and to cope with potential information overload.

We find much evidence that the biggest barrier to the adoption of visual analytics is inadequate software infrastructure. Just as structuring visualization software around simple, composable parts is highly successful (exemplified by Grammar of Graphics approaches like Vega and ggplot) it is intriguing to consider what systems will be possible when this philosophy is extended to the broader requirements of interactive analysis.

RCloud is available at `github.com/att/rcloud/`.

References

[1] jsfiddle, 2015. https://jsfiddle.net.

[2] plot.ly, 2015. https://plot.ly.

[3] R. A. Becker and J. M. Chambers. Auditing of data analyses. *SIAM Journal on Scientific and Statistical Computing*, 9(4):747–760, 1988.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[5] M. Bostock. bl.ocks, 2015. https://bl.ocks.org.

[6] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[7] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 745–747. ACM, 2006.

[8] E. Fast, D. Steffee, L. Wang, J. Brandt, M. S. Bernstein, and A. Stanford University. Emergent, crowd-scale programming practice in the IDE. In *CHI 2014*, 2014.

[9] M. Fowler and M. Foemmel. Continuous integration. *Thought-Works http://www. thoughtworks. com/Continuous Integration. pdf*, 2006.

[10] Github Gist. `https://gist.github.com`. Accessed: 2014-03-30.

[11] P. J. Guo and D. Engler. Towards practical incremental recomputation for scientists: An implementation for the python language. In *Proceedings of the 2Nd Conference on Theory and Practice of Provenance*, TAPP'10, pages 6–6, Berkeley, CA, USA, 2010. USENIX Association.

[12] S. Gutierrez. *Data Scientists at Work*. Apress, 2014.

[13] J. Heer and M. Agrawala. Design considerations for collaborative visual analytics. *Information Visualization*, 7(1):49–62, 2008.

[14] J. Heer, F. Vigas, and M. Wattenberg. Voyagers and voyeurs: Supporting asynchronous collaborative information visualization. In *ACM Human Factors in Computing Systems (CHI)*, pages 1029–1038, 2007.

[15] M. Httermann. *DevOps for Developers*. Apress, Berkely, CA, USA, 1st edition, 2012.

[16] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. IEEE, 2008.

[17] T. J. Jankun-Kelly, K.-L. Ma, and M. Gertz. A model and framework for visualization exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):357–369, Mar. 2007.

[18] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 154–161, New York, NY, USA, 2005. ACM.

[19] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, 2011.

[20] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Enterprise data analysis and visualization: An interview study. In *IEEE Visual Analytics Science & Technology (VAST)*, 2012.

[21] D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[22] P. Mates, E. Santos, J. Freire, and C. T. Silva. Crowdlabs: Social analysis and visualization for the sciences. In *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management*, SSDBM'11, pages 555–564, Berlin, Heidelberg, 2011. Springer-Verlag.

[23] D. R. Millen, J. Feinberg, and B. Kerr. Dogear: Social bookmarking in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 111–120, New York, NY, USA, 2006. ACM.

[24] M. S. Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, May 2006.

[25] S. Moore and S. Chong. Static analysis for efficient hybrid information-flow control. In *Computer Security Foundations Symposium (CSF), 2011 IEEE 24th*, pages 146–160, June 2011.

[26] F. Perez. Project jupyter, 2015. https://github.com/jupyter.

[27] RStudio. ggvis: Interactive grammar of graphics for R, 2015. `https://github.com/rstudio/ggvis`.

[28] RStudio Inc. *shiny: Web Application Framework for R*, 2013. R package version 0.8.0.

[29] M. Rubacha, A. K. Rattan, and S. C. Hosselet. A review of electronic laboratory notebooks available in the market today. *Journal of Laboratory Automation*, 16(90), 2011.

[30] E. Santos, L. Lins, J. Ahrens, J. Freire, and C. Silva. Vismashup: Streamlining the creation of custom visualization applications. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1539–1546, Nov 2009.

[31] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In *ACM User Interface Software & Technology (UIST)*, 2014.

[32] C. Sievert and K. E. Shirley. LDAvis: A method for visualizing and interpreting topics. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 63–70, 2014.

[33] Trifacta. vega: a visualization grammar, 2015. `https://github.com/trifacta/vega`.

[34] J. W. Tukey. The future of data analysis. *The Annals of Mathematical Statistics*, pages 1–67, 1962.

[35] J. W. Tukey. *Exploratory data analysis*, volume 231. 1977.

[36] S. Urbanek. A fast way to provide R functionality to applications. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, pages 2–11, 2003.

[37] F. B. Viegas, M. Wattenberg, F. Van Ham, J. Kriss, and M. McKeon. Manyeyes: a site for visualization at internet scale. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1121–1128, 2007.

[38] M. Wattenberg and J. Kriss. Designing for social data analysis. *Visualization and Computer Graphics, IEEE Transactions on*, 12(4):549–557, 2006.

[39] Y. Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, 2013. ISBN 978-1482203530.