Collaborative Visual Analysis on the Web with RCloud

Category: Research

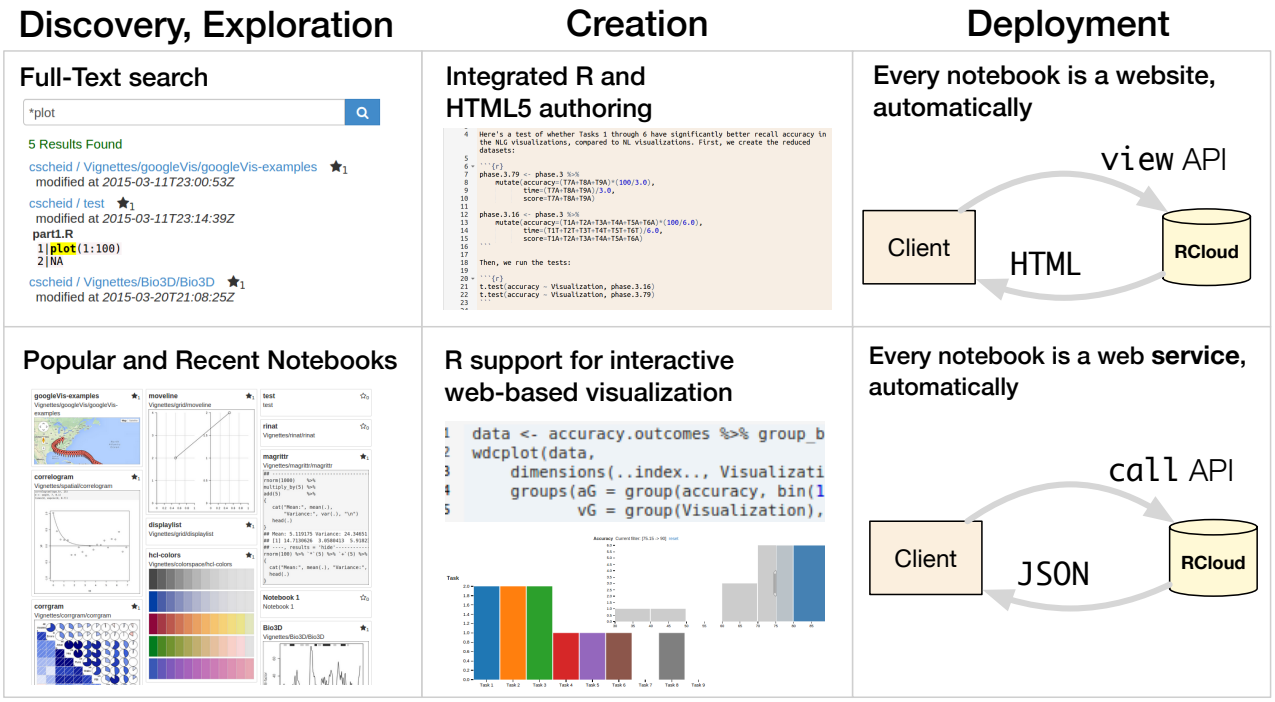

Fig. 1. An overview of features in RCloud. RCloud was built to support an environment in which a large number of loosely-related problems in data and visual analytics are solved by a small number of *hackers* and *scripters*. In such a shared environment, *discoverability* is also a problem. RCloud supports search, annotation, recommendation, and commenting for all notebooks, enabling an overview mode where users can *browse* popular and recent analyses. When problems and data sources change constantly, deployment becomes a time sink; RCloud supports *transparent and automatic* deployment of analyses as web pages and web services, allowing a seamless transition from exploratory work to production code.

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 would 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: *technology transfer*, *coexistence*, and *discoverability*. In addition, we contribute the design and implementation of RCloud, a system that supports the requirements of collaborative data analysis, visualization and web deployment. The biggest deployment of RCloud has been in active use for more than a year, and has about fifty active users. We report on interviews with some of these users, and discuss the design decisions, tradeoffs and limitations, comparing RCloud to other current proposals.

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 [36]. Powerful, interactive environments for analysis and programming were the goal, together with an unflinching (and, at the time, somewhat heretical) insistence in keeping humans 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 [37].

In some way, we have moved far beyond what Tukey imagined: computing and networking capabilities today far exceed what was barely imaginable then. We argue, on the other hand, that how we *develop* our data analysis solutions has not changed as much: the S language was developed essentially alongside Unix [3] and, although environments such as RStudio include a variety of modern features, they still follow the basic metaphor of terminal, text editor, and source files stored in file systems. We turn our attention to this opportunity to use computation to support, not only individuals, but entire teams and their work within larger organizations. In this paper, we contribute the design of RCloud, together with an interview study conducted over the course of its development and deployment at AT&T Labs, where RCloud has been in use for about two years.

Consider the role of a data science team within a business or tech-

nical organization today. Project teams vary in size, from just a few people to dozens or more, even within one project's duration. Assignments are often 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. Eventually, knowledge or working prototypes created by these teams are transferred to other organizations to employ them in production. There are many details to these tasks. For example, data wrangling can involve finding data, understanding its syntax and semantics, assessing data quality, performing normalization and data quality remediation, and making the data available to other tasks that will follow.

Visual analytics depends greatly on communication and collaboration. At almost every step, detailed knowledge about data, code and tasks is shared by collaborators. Further, data scientists are increasingly asked to work more closely with business or domain specialists who may be less technically oriented. Thus, data scientists and developers are being asked to become very broad, and integrate work across the spectrum.

Unfortunately, the processes and technologies supporting data analysis visual analytics are fragmented. Knowledge is shared through informal conversations, emails, meetings, phone calls, source code itself, in wikis, in project documents and by other means. Results of experiments are often shared first by copying static output (usually a simple screen shot or image output). By the time decisions are made based on the results of that screenshot, the data and the processes might have changed enough to bring to question the original decision.

This situation can affect how data science teams; as analysts and toolbuilders, we have experienced this first-hand in our environment, and others report similar findings. Gutierrez collected interviews with data scientists [13], which include the following:

- (upon being given access to other code and data analysis, in order to learn about Hadoop) "[...] Not only that, I could also 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) "[...] Finally, 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 for all of your data, so [...] you can go back and test all of your data.

Currently, these concerns need to be addressed *alongside* the visual analytics environment. Some symptoms of the current situation are:

- It is hard to find data, metadata, and knowledge about them.
- Most coordination is done through meetings, whose content is not linked to other artifacts and may not even be stored.
- Production deployment requires porting code to a different environment or even completely rewriting it, thwarting continuous release.
- Many tools adopt a standalone or single developer perspective, not suited to collaboration and web deployment.
- Analysts find insufficient support for tracing events or issues from production back to the EDA process.

In view of the opportunity to improve this situation, we created a software environment that supports the end-to-end visual analytics process for individuals and teams. This environment knits together some familiar tools, and provides new features to find data and code; create experimental workbooks; 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.

A key point is that, for the most part, the improved capabilities are available by default, without much explicit intervention on the part of data scientists and other customers. Visual analytics experiments are conveniently shared and turned into production websites, without moving or porting code. All the published artifacts in the system can be searched immediately. Recommendation is as easy as clicking a star on a useful workbook.

The high level architecture of the prototype system is shown in figure 2. We chose R as the foundation for our prototype 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.

From a more principled or theoretical perspective, certain visual analytics goals or objectives for emerging systems, described in previous work, closely match ours. We noted the following central themes.

1. Technology transfer. In most organizations, development of analyses and visualizations s done by *hackers* and *scripters* (adopting Kandel et al.'s terminology [20]). *Application users* gain the benefits of the tools developed by hackers and scripters. Generally, the connection between these two worlds is made by IT staff, who port or even rewrite code so it can run as a stable production service. In an environment where business needs can change rapidly, this process does not scale. Our objective is to merge the worlds of EDA and production.

2. Coexistence. In the current data science ecosystem, 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.

3. Discoverability. A chronic complaint of data analysis teams is repeated work ("How can I connect to database X?", "How do you get clean data from column Y from feed Z?") This work arises from lack of communication about previously solved subproblems. Our goal is encourage and support transparency of work between team members.

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 other collaborators in other business units.

.

2 Related Work

In some sense or another, the entire field of information visualization and visual analytics revolve around improving how users solve problems using data. In this section, we focus specifically on system work relating to problem-solving enviroments and multiple users.

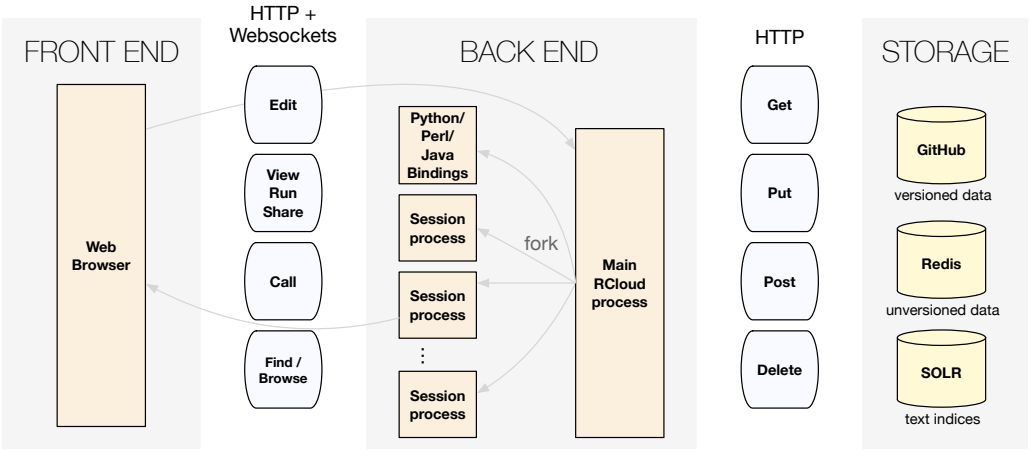

Fig. 2. A diagram of RCloud's architecture.

Social Data exploration and Analysis **ManyEyes** [39] was a landmark system designed for the crowdsourced creation and publication of data visualizations. Although ManyEyes only supported a limited number of visual encodings, the system's success was both a precursor to more sophisticated solutions and an early indication that the world-wide web was a suitable platform for data visualization. One of the main challenges we faced in designing RCloud was providing an experience for *consuming* visualizations as seamless as ManyEyes's, while sacrificing as little as possible on the generality of the analyses themselves.

Notebooks as a medium for data analysis dissemination The concept of a "notebook" as we use it here can be traced all the way back to Knuth's literate programming [21]. In literate programming, a comprehensive description in prose of the behavior of the program is "weaved" together with the source code, yielding both an executable program and a human-readable document. A notebook represented as a collection of short, executable cells, originated with Mathematica, and in R, literate programming is supported by packages such as knitr and RMarkdown [41]. Project Jupyter [28] (originally implemented as part of IPython) offer some notebook features, but lack a transparent mechanism for sharing and deployment in multiple-user settings. Further afield, **Electronic Lab Notebooks** are applications for organizing and sharing data from scientific lab experiments[31]. In a sense, we hope to adapt and extend this concept to the work of visual analytics teams. Although RStudio offers publication of literate R programs as a free service on their website, the workflow is somewhat disconnected from the development of those programs. Once they're uploaded, it's hard for other users to build off of the work published (or even for the original author to update new versions). In other words, RStudio handles *publication*, but not *deployment* and sharing.

Provenance and versioning As we mentioned, one central problem in exploratory analysis is that problems change quickly over time, often in the course of developing a solution. As a result, systems should provide adequate support for tracking *changes* of the data analyses scripts. VisTrails was one of the original systems for managing *process provenance* , and demonstrated the value of capturing aspects of the processes that surround data analysis experiments and tools, including detailed history, collaboration, and deployment [8].

VisMashup [32] defines a schema and semantics for automatically deriving user interfaces from workflows, while Crowdlabs exposes these capabilities on a website feature [23] workflow upload and remote execution. In our view, the impedance mismatch between a dataflow pipeline specification and the power of a general-purpose language is too great for the type of general exploratory work in data science teams. At the expense of ease of use for non-programmers, RCloud tries to provide a closer match for analysts accustomed to creating and executing R and Python code, while retaining attractive properties like transparent provenance tracking and interactive data visualization on the web.

Web-based tools for sharing code snippets There have been a variety of tools recently developed for quickly sharing small programs on the Web, including but not limited to: bl.ocks [6], jsfiddle [1], and plot.ly [2]. bl.ocks and jsfiddle are both designed to share Javascript programs, which means that their deployment happens automatically through the web browser. If and when Javascript becomes the lingua franca of exploratory data analysis, then we can foresee building a much simpler version of RCloud where all execution happens either on the client side or via web services. Unfortunately that is not a realistic assumption for the present, making these solutions by themselves unsuitable to our use case. Plot.ly is notable in that it provides API support for publishing *from* scripts: in other words, it is possible to generate a plot.ly visualization from inside a separate program. Although this is an intriguing idea, it nevertheless creates a disconnect between the analysis and the resulting visualization. In RCloud, we wanted to make sure that every visualization was transparently linked to the source code that generated 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. RCloud seeks to reduce this mismatch. 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." Their study highlights the opportunity for better ways of supporting collaboration and sharing in data analysis teams.

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]. We view this as attacking a different semantic level than ours, but also showing the need for an environment that enables better sharing of the knowledge, tools and processes to do this. Anecdotally we find much frustration among practitioners that this knowledge is difficult to find and often isnot recorded or available in a reusable form even within the same organization.

Heer and Agarwala identify many design considerations for collaborative visual analytics [14]. RCloud notebooks, and the integrated version control system for them, described in Section 3.3, address modularity and granularity, and artifact histories. *Starring*, the means for signaling interest in notebooks, described in Section 3.4, addresses social-psychological incentives, recommendation, and voting and ranking. RCloud's integrated deployment mechanism, described in Sec-

tion 3.7, addresses the cost of integration, content export, presentation and view sharing.

The need for integrating statistics and visualization has been highlighted in previous studies and is widely understood by various technical communities [27] Lucas and Roth were early advocates of combining data exploration with presentation and publication [22].

There has been noteworthy work on specific techniques such as social bookmarking [24] [15] and crowdsourcing [9] to support collaborative or social development or analysis processes. Similarly, there are computational methods to support high performance execution in incremental code development environments [12]. 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 overall 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" [16] or "continuous integration" [10]) is a trend in web services and related fields. By making it convenient for data scientists to expend just a little more effort when creating experiments, we may be able to eliminate the need for programming teams to recreate their work to deploy it in production, which has a high cost in time, expense and accuracy. We now turn to a description of the system architecture itself, and how it enables the features we have been highlighting.

3 The System

3.1 High-level Architecture

The internal computing infrastructure of organizations has changed radically in the last fifteen years. The shift from large servers toward scalable, lower-cost, distributed systems ("the cloud") led to a software ecosystem of processes distributed over a network, usually communicating via HTTP. HTTP is dominant because web browsers and servers are ubiquitous and available in almost all hardware devices, from tiny sensors, to handheld devices and laptops, to rack-mounted servers.

As a result, HTTP is the lingua franca of interprocess communication (IPC). One of the design goals for RCloud was to provide a productive environment for creators of data-analysis scripts, that also behaves as a first-class citizen in the pre-existing ecosystem of computer services and networks in an organization.

3.2 Human Interface

Figure 3 shows the RCloud developer interface. The center panel is a notebook for code and visualizations. Code is edited in this panel, and visualizations are rendered. An inventory of supplementary "asset" files, that are part of the notebook, though not in its executable flow, are edited on the right. Controls at the top of the screen allow the user to run, view, and share the notebook. The result of code execution is the final cell of the notebook. On the left is a browsing area for searching and viewing other users' notebooks, and a help system.

The call mechanism is the URL itself; arguments are read from the URL and the final cell is rendered as the result.

3.3 Notebooks

The main unit of computation in RCloud is a *notebook*. A notebook holds a sequence of *cells*, each of which contains a snippet of code or hypertext in Markdown. This is not a novel idea; executable documents structured this way are a feature of many other systems, including Mathematica, IPython and Sage.

One of the main contributions of RCloud is the idea that notebooks are "always deployed". In other words, the most recent version of a notebook is immediately available. This makes it convenient to share and modify experiments and compose results without binding to a specific version of a notebook. On the other hand, there are situations where it might be important to name specific versions. We do not expect designers to decide which versions need to be preserved, but we embrace *transparent* versioning. This is similar to models like Jankun-Kelly et al.'s p-set calculus [17] and VisTrails's version tree [8], where every change in the state of the system is tracked.

To implement this, we built RCloud on top of GitHub's *gists* [11]. GitHub offers a HTTP interface for creating simplified git repositories, the main limitation being a restriction to text-only files in a single directory. 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.

Using GitHub for storage and versioning also exposes other capabilities that can be invoked with JSON and HTTP. Particularly, we can provide full text search using Apache SOLR. SOLR is scalable, can index in near-realtime, supports multiple character sets, indexes several common types of documents, and has schemas and faceted search. Integrating existing services and software components, especially open source, instead of implementing custom code is a trend in the software industry, and is very positive for systems research and prototyping in visual analytics. By adopting existing technology, small teams with limited resoures can explore novel ideas toward the improvement of large, complex software ecosystems. In our case, even though this approach involved learning standards not directly related to visual analytics, our early decision to use the GitHub Gist API turned out to be successful. Future projects could gain many of the same benefits by adopting this strategy. In fact, it will be essential for technology adoption and transfer, because it is almost impossible for any one tool or platform to "own" or support the entire visual analytics process.

3.4 Reputation and Interest: starring

Information retrieval based on collecting usage and recommendations is a cornerstone of modern web services. We would like to help data scientists to find workbooks (and therefore items in their contents such as code, data, and colleagues with specific expertise) with the benefit of such information.

In RCloud, reputation and interest are a relationship between *notebooks* and *users*, rather than a relationship between user pairs. We chose this approach because we expect initial RCloud deployments to have relatively few users, but some users to create many notebooks. Under that assumption, assigning interest to users would not provide sufficiently "high-resolution" data.

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 readily available, to encourage its use.

Implicit signaling of interest is supported by keeping click-through counts [18] and execution counts. (In addition to these standard techniques of collecting feedback from web search, we anticipate applying static and dynamic code analysis to infer fine-grained information about relationships, for example, which packages and data sets often appear together.)

3.5 Executing R in a web browser

One our main goals is to provide ubiquitous access to the statistical tools of the R programming language in the wider environment in which data science teams participate. Given the recent developments of interactive visualization and visual analytics in HTML5 and the web, we decided to create an *interconnect* between the two environments. Every RCloud session spawns a new R execution process in a remote server.

HTTP and web services are convenient but much of the protocol is stateless. (For example, GET requests are required to not change the remote state, and results can be cached in transparent proxies along the network). In the case of close communication between a running R process and a web browser, we need something that is more flexible than distinct URLs for every R process, and which embraces statefulness of both calculations and visualization parameters.

3.6 Interactive notebooks

A key engineering decision in RCloud was to rely on full two-way communication between the web client human interface, and data and computing resources in the cloud. It is easier to engineer a design in which the human interface is pushed one-way from the server and thereafter all interaction takes place within the client. Two-way communica-

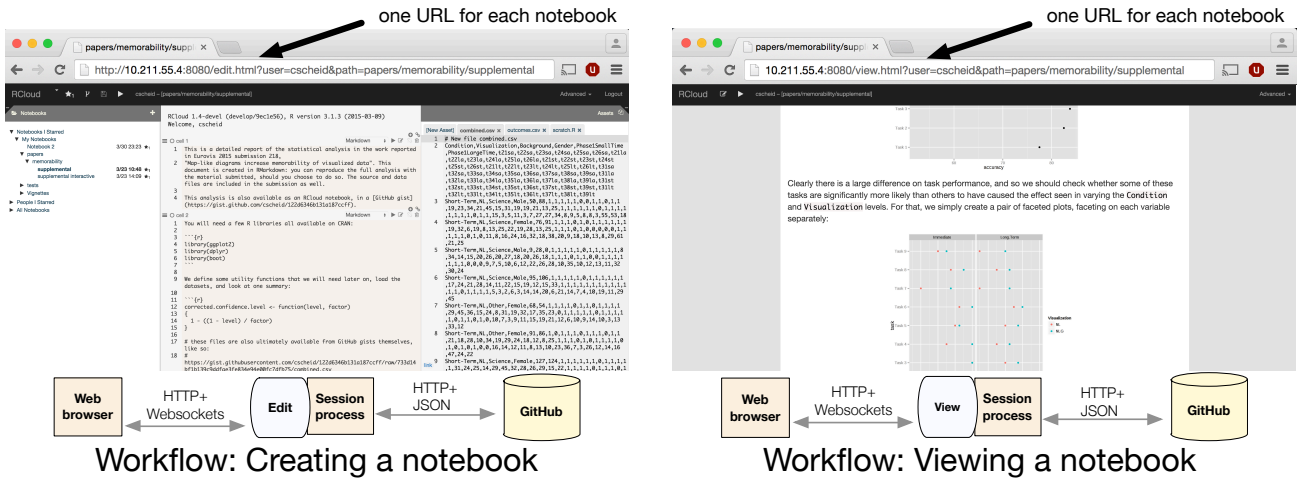

Fig. 3. An RCloud notebook is a sequence of *cells*, each a snippet of source in one of the supported languages (typically R, but Python and others are also supported) or Markdown. The main creation workflow involves editing notebooks, which are transparently stored as git repositories in GitHub, providing us with easy access 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 lightweight provides a very low-friction mechanism for sharing results which we discuss in full in Section 5.

tion is necessary, though, where the size of the data or computational performance makes it impractical to move all computation to the client.

In addition, R is well suited as a language for analysis, and JavaScript for interactive visualization. To draw on the strengths of both languages and both environments, the connection between the languages is not just procedural: it makes *closures* and *first-class functions* available across the network. This provides considerable flexibility, so that for example, a chart built with dc.js or leaflet.js can call back to analysis functions in R without having to formalize the protocol between the processes.

3.7 Deployment of notebooks

Every notebook in RCloud is named by a URL, and notebooks by default are visible by the entire organization. This is deliberate. As pointed out by Wattenberg and Kriss [40], 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 anecdotal support for this notion by discovering links to RCloud notebooks in internal discussion fora and mailing lists.

Because notebooks are continually published, any work is immediately available both as a subroutine and a visual component. Close colleagues can start on the next stage of analysis, or delve into the data, even while the original author is polishing an algorithm or its presentation. The code is the page, and can either be shaped into a function of inputs and outputs, or the linear cells of the notebook can be reworked into a full-fledged HTML layout.

4 Case Study: Visual Exploration of Topic Modeling in RCloud

In this section, as an example we present a simple real-world application that was deployed under RCloud.

The application is an implementation of LDAVis, a recently-published method for visualizing large text corpora. It was specifically designed to assist non-experts to explore collections of short text documents using topic modeling and visualization. Topic modeling [5] is a standard technique for text summarization which, although powerful, requires a certain amount of manual intervention and interpretation [34].

LDAVis was developed by two technical staff members at AT&T Labs, and originally targeted RStudio's Shiny [30], a framework for developing R applications for the web. While Shiny provides outstanding ease of development, discoverability and deployment turn out to be equally important aspects in the lifecycle of an internal application (see Section 5). On these aspects, RCloud provides a very simple model: *all developed notebooks are automatically deployed*.

LDAVis combines text analysis, dimensionality reduction and interactive visualization. Text analysis is performed via a combination of an existing R library that fits an LDA model using Gibbs sampling [] and some R code written specifically by the developers. The analysis module is, ultimately, a single R function that is exposed to the web application via the Javascript-R RPC mechanism described in Section 3; this way, the analysis is executed remotely on the RCloud servers.

Each textual topic is a probability distribution over the all the words of the document. In order to expose patterns in the relationships between the topics, LDAVis employs a combination of interactive visualization and dimensionality reduction (letting users choose different measures for the topic distances and different dimensionality reduction techniques). The dimensionality reduction algorithms and distance measures are again implemented in R, which means they are executed on the RCloud servers as well.

The result of the dimensionality reduction process is a two-dimensional plot of the topic space, which can be seen in Figure 4. The interactive view is implemented with SVG and Javascript through d3 [7]. The most popular web-based visualization libraries for R is ggvis [29], and so a natural question becomes: could ggvis have been used instead of custom Javascript? In this case, the interactive features of ggvis (and, by extension, Shiny) are a subset of those of Vega. The custom interactions in LDAVis (hovering over a single topic, hovering over a topic cluster, hovering over a word in the distribution, etc) do not appear to be available in the current version of Vega [35], although the required components for reactive interaction have been recently described by Satyanarayan et al. [33]. As a result, custom Javascript was required in this case, making RCloud uniquely positioned among systems available today.

This application is simple, but highlights some of the unique features of RCloud. While RCloud notebooks allow deployment of R analyses over the web with no additional effort, RCloud *applications* are more powerful, and are developed in a combination of Javascript and HTML for the front end. This requires additional knowledge over Shiny, but we argue that the RCloud model makes the analysis side simpler for the analysts (since they simply write the R code in the style they are used to), and the front-end visualization side simpler for the front-end developers (since they simply write the Javascript code in the style *they* are used to). In addition, RCloud applications inherit the automatic deployment and discoverability features of regular RCloud notebooks.

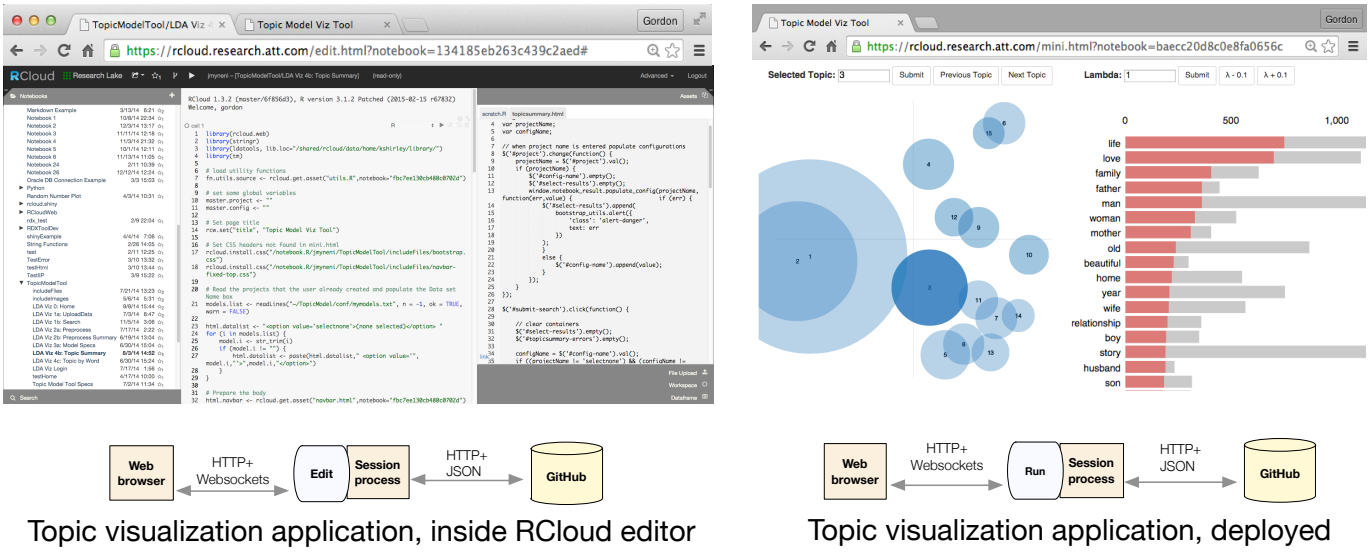

Fig. 4. An example application developed and deployed in RCloud.

5 Interview Study

To evaluate the effectiveness of RCloud, we interviewed 13 current and past users of RCloud. Of these, 9 are data analysts and 4 build tools for data analysts and business needs.

Generally, analysts were more annoyed by the cell interface and the perceived slowness of hte interface than web/tool developers. Analysts switching to R from another language were more likely to use RCloud for exploratory data analysis. Web/tool developers were more likely to see the need for parameterization.

5.1 Sharing of results

Sharing of notebooks is the core feature of RCloud, and a popular one. All active users we spoke to praised this feature.

By default, all notebooks are publicly visible in RCloud, and notebooks can be found through the notebook tree or search. However, the method for sharing that users mentioned most is sending links through email. Clint says, "If my supervisor wants to see what I've done or QA it, I can just send her a link."

Besides providing a way to present work to others, notebook sharing can also provide a starting point. Clint says, "I think the best part about it 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." Raif notes, if "some person has done something similar, then you're able to just edit that, and that's saved a lot of work time for me"

Taylor develops packages for analysts, and uses RCloud "more 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 kind of 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 explaining problems with data sources.

Some users who tried out RCloud but are not able to use it for organizational reasons, already miss the functionality. Shelly likes "the concept of being able to create notebooks and share them." A "wiki is not the best way for communicating results - it's kind of like writing a blog post with very limited functionality [...] I have to save every picture and post it as an image" and "I can't share all of the code because it would just get crowded and wouldn't look right on a wiki." Bart says, "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 thing, 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."

5.2 Forking

The ability to start work where someone else let off, by forking, has proved to be a popular feature. In fact, almost twice as many (131) users have forked someone else's notebook as have starred one (75). Clint says, "it's one of the handiest functions, because instead of having to find it, copy, paste it, you just hit Fork, you rename it, and it's done. It's pretty amazing."

Although we always intended forking to be used to improve others' code, we we initially didn't support forking one's own notebooks, and this has proved very useful. Rick 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 Horace works with the 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."

An extremely common but more problematic use of forking is to change parameters. Raif: "I've been forking other people's notebooks [because] I want to run it on a different part of data, or I want to change some parts, I don't want to see this column, I want a different column, things like that." Taylor 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."

Currently parameters can be added to the URL, but adding user interface elements to do the same thing requires expertise. The tool-builders see a need for facilities to make it easier. Jyothi calls it "web-enabling" the notebook: users "can always fork the notebooks and make changes, but I feel that if the owner of the notebook web-enables it, [...] it's easier to run. Instead of forking it, if they can set options, it's probably more efficient, and also they can [still] fork it if they want to."

5.3 Automatic source control

Automatic source control is also a popular feature. Clint 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."

Bart points out that the automatic versioning works well for dealing with the minutiae 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 any of this 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. Rick says that for this reason, the history feature is not all that helpful: "I don't need something that keeps track of every mistake I've made or every direction I've tried."

5.4 Discovering others' work

Many users find the search function useful. Raif 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 the experts they know. Rick says, "usually I know somebody's notebooks that I want to search through [...] because I know that 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 volume of notebooks continues to grow. We explore some ideas in Section 6.

5.5 Integrated analysis

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

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

Integrating R also helps Jyothi's topic modeling tool, described in the last section: "Having an open session where I can run R commands or R functions without having to invoke an API or send out a request and then wait for the request to come back [...] is extremely helpful in writing the application."

5.6 Exploring elsewhere

Although most of the analysts use and appreciate the sharing features, RCloud is less popular as a tool for exploratory data analysis. Every analyst with prior R experience still does analysis in another tool and pastes their code into RCloud for sharing.

This is partly out of familiarity. Taylor asks, "why would I want to use RCloud over my current setup? If it's just me, I like my text editor and terminal. There's nothing that I want that those two don't give me."

The web interface of RCloud doesn't work for Rick: "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 old editor window."

Kenny also works with text file and command line, rearranging a file so the good code is at the top of the file. When he's done, all the the right is code at the top. RCloud does not readily support this workflow, but Kenny often shares the work by pasting it into an RCloud notebook when he's done.

Working in a shared environment also entails compromises about what you can install. Byers says installing alternative or nonstandard packages is intrinsic to exploratory data analysis.

5.7 Cells versus the command line

RCloud's notebook interface combines editable Markdown with a command line interface.

The interface takes some learning. Clint says at first,

> I saw the cells and it just threw me for a total loop. I mean, it's a good idea because [...] I can run it all at once if I want or I can break these into sections for either debugging or staging purposes. I really like it now, but when I first saw it, [it was] very confusing.

Just the difference between a cell and the command prompt can confuse and slow down users. Rick explains:

> If I was typing into one of the cells near that top, I had to think "I'm editing this cell and then I have to execute it," and if was doing the one at the bottom I could type it but then it would automatically execute [by pressing enter], but then it became a standard cell and I have to edit it.

Much of the time, commands typed at the R command line just serve a transitory purpose, so having RCloud save commands into cells can be annoying. Kenny notes, "It saves everything I do like everything is gold, but most of it is junk not meant to be saved." Rick says "I just want to type in a couple of quick commands and get some results that are going to tell me what to do next, and they're not necessarily archival in any sense."

The material that is not appropriate to save includes "expressions that allow me to check that I'm the right track" (Rick), "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" (Raif).

Users also felt that cells do not capture the right level of granularity: they either hold too much code or too little. Rick 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 [...] on the inner parts and then put them together to make the big thing" as he would do on the command line. When Taylor uses the R command line, he copies and pastes "like 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 same functionality is if every line's in one cell."

Cells can also take up substantial vertical space, requiring scrolling. Rick complains that "every time I type more stuff, the notebook gets longer and longer and it's harder to deal with." Ivan says that cells' controls and blank space take extra screen space compared to a straight command line, and he and KC both complained that when there are long results or plots, it causes the code to scroll off the screen. Raif explains, "Imagine I run a cell whose output is huge. [...] To find the next cell, I have to scroll down and find where that cell starts. So I'm losing the continuity of my code."

Some users would like the option to keep results separate from code. Ivan thinks that RStudio's layout is more helpful because charts are kept in another pane and stay in one place while doing analysis. Raif 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 run. In Kenny's work, there is often a "long tail". The first cell may take seconds to run, the next cell minutes, and the last cell 5 hours. In this situation, the Run Whole Notebook button is "dangerous". When converting his work to notebooks, Kenny ends up with a lot of comments that say "this cell takes a long time to run."

To avert this danger, some users write code within the cells to cache results. Kenny ends up "littering the notebook with little switches that comment out the slow parts" and either "load the object instead" or "save the object to disk". One of the authors of this paper manually writes cells that check if a result file exists, and perform the calculation only if it's not there.

5.8 A sea of notebooks

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

For this reason, Rick doesn't find the notebook tree very useful:

> I don't necessarily need to see everybody's notebook that uses RCloud [...] Every time I do something new, I get a

> new notebook and so now my notebooks are maybe 50 or 60. That's enough to think about just on my own, but if I've got everybody's 50 or 60 sitting on my display, I find that it's more than I want to know about.

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

Taylor, who writes packages and example notebooks for them, seems to be haunted the most by dead notebooks. Taylor and Horace 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.

The problem of dead notebooks is compounded by changing packages and their example notebooks at the same time. Taylor will "go back and change it and update the page, but then what happens is people have in the meantime forked it. [...] I'll have to change a package as well, so their old fork stops working, and then they complain."

Taylor tried to keep his notebook tree tidy, but this didn't help, because people either kept old links in their email, or forked a notebook which is now obsolete. By design, notebooks that are deleted in the user interface are not purged from the repository. "Now I'm really gun-shy about sharing notebooks because it's like, do I want to support this forever?"

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 are appropriate, and how well they were met.

Our experience underscored the relevance of Heer and Agrawala's design considerations, and indicated areas for further exploration. We adopt their taxonomy for the following discussion.

Shared artifacts and artifact histories are a central feature of RCloud, through its shared workbooks of experiments and analyses. We observed that hackers readily share work through RCloud. They use it to demonstrate techniques, share results with peers and managers, and transfer algorithms to other groups. Artifact histories can be accessed through the notebook tree or through GitHub's web interface.

Modularity and granularity. The ability to partition work into independent units (modularity) is a key to working productively in teams. It is good if the units can be kept small, so team members can realize benefits at least proportional to the work on the units (granularity). RCloud's notebook and versioning capabilities allow work to be divided into units as fine as the underlying language allows, and versioning encourages 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 an effective mechanism for sharing and for integrating analyses with external processes and systems. It is particularly advantageous for work to be shared as URLs that provide access to code and experiments, instead of by pasting static screenshots into documents and presentations.

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. We assume that RCloud notebooks should play well in the ecosystem of other tools, but sometimes it is difficult to know whether adding a compatibility feature will offload complexity or bring more in.

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 valuable to know which packages are frequently used together, or appear in proximity to a certain record types or data feeds. In general, trivial or passive mechanisms to collect data for recommendations are essential.

Group management, size and diversity. This area needs better support in RCloud, but 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 may curate groups of related notebooks using notebook 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. Some important aspects of the environment are more difficult to manage in RCloud than in conventional systems. RCloud does not explicitly separate the development and deployment environments. More than that, every version of every prototype is shared by default. Although this encourages collaboration on work in progress it also quickly exposes errors such as misconfiguration, mismatches between versions of packages, and programming errors that can unintentionally affect production websites. Such errors are difficult to completely avoid, but the power of convenient sharing is often worth it (as proven true with distributed systems in general). Still, effective control over versioning when sharing problem needs more attention if RCloud would be scaled up to a wider number of users lacking informal channels to coordinate work.

Versioning needs to be managed in several places: in scripts; in the R environment such as the installed libraries and packages; and in the external environment such as the operating system and protocols spoken by remote services. At one end of the scale, we have full control over the versioning of scripts via git, along with conventions to name stable or working versions of scripts, and versions known to work together. The R environment itself is under the control of its package manager that has rules to ensure reasonable consistency. It is at least possible for RCloud to access information about package versions and configurations, but support for checking compatibility and maintaining multiple versions in the same environment is not strong. At the other end of the scale, there is not much reason to expect version control for the external environment. Most programmers rely on clean living and a careful approach to system upgrades.

On top of this, RCloud allows or even encourages hackers to fork experiments to try new ideas quickly. So far we have not done much more to address the problem of having a lot of bits and pieces of code and data lying around, though we have provided a framework in which it should be easier to find them and to apply algorithms and metadata to organize them.

Caching. An important consideration is how and where to introduce caching in RCloud's distributed computation model. Caching can dramatically improve performance in a way that is otherwise transparent to application program semantics [8, 12]. Though caching is usually implicit, in some environments, such as VisTrails, programmers may

	Versioning/Forking	Collaboration	Deployment	Multilanguage	Integrated Reports	Integrated Analysis
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	

Table 1. Comparison of system features.

also have some explicit control over cache management. This may be desirable to ensure the repeatability of computations that rely on volatile or unreliable data sources. For example, the stock ticker use case is one for which the data might temporarily become unavailable, so caching could improve reliability and consistency of results. Alternatively, in situations involving relatively expensive computation (for example, analysis of large text corpora, clustering multivariate time series or deep learning algorithms) caching may be essential to adequate performance. Currently we would program this in RCloud by explicitly saving an analysis in a persistent database, but it seems better to implement this capability in a general purpose associative cache, instead of application-specific libraries. We envision cache management being implemented in a new middle layer to be added in the future.

Security. It is essential to provide security in an environment for collaboration and data publishing. To provide some protection for RCloud workbooks in an organization's intranet, RCloud uses an object capability model [25] recently added to the Rserve protocol [38] that prevents unauthenticated clients from making unauthorized calls to the RCloud runtime environment.

Our back-end environment assumes a high degree of trust between users. Access control for information security is delegated to the host operating system and web server. In practice, most operating systems and web servers 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 should be improved, but more sophisticated approaches (such as information flow analysis, and query languages that ensure differential privacy) are active research topics, and many difficult problems remain [26]. We are hopeful that having a richer environment for collaboration and information sharing will encourage new approaches to information security in visual analytics.

Exploration is hard. R is a popular language for exploratory data analysis, and we built RCloud as a way for data scientists to seamlessly transition from EDA to sharing and presentation. But the combined notebook and command line interface we designed interferes with the use of R for exploration.

In existing R tools, the command prompt is used to try things out. It is very fast and there is no commitment: nothing gets saved and there is nothing to later clean up. In contrast, in RCloud's notebook interface, the extra cells need to get deleted later, and there seems to be a mental burden that *bad stuff is getting saved*.

One thing to try here is making deletion easier, for example with a shortcut key to remove the last cell, or a way to keep a few cells and cull the rest. A far more ambitious solution, suggested by one of our interviewees, is to implement auditing of data analyses as it existed in S [4], so that a result could be selected and code which did not lead to it could be discarded automatically.

We got here from an insistence on reproducibility: it is a core principle of RCloud that no command should be run without being saved, even if it gets deleted later. This is why we did not implement a simple command line as many users request. We could imagine making an exception for commands that have no side-effects, but in the general case this is no easier than auditing because R is not a purely functional language.

The interface is also perceived to be slower, both for computer and for human. First, there is a slower reaction to commands being run, because RCloud necessarily saves a cell to its repository before executing it. It may be possible to safely write to the repository behind execution of the same code. Second, and perhaps more detrimental, the interface requires mouse interaction and switching between the keyboard and mouse, which are known to be slower. Shortcut keys can be devised if people are willing to learn them, but it would be better to reduce the number of such operations needed.

Other complaints mostly focus on layout and can probably be addressed with some iteration and customization features.

Discovery is hard. While we have provided tools for searching and curating notebooks, the search function does not help for finding methodology (and does not protect searchers from erroneous notebooks), and the notebook tree starts to get unwieldly with hundreds of users with hundreds of notebooks. In addition, our choice to retain notebooks even when they have been deleted has exacerbated the situation, since obsolete and broken notebooks stay around forever.

One thing to try is tagging and filtering the tree on tags, so that users can mark their own notebooks or those of others to make them easier to find. However, this relies on users making a conscious effort, and it may be better to rely more on passive metrics such as how often a notebook has been run.

The system also needs to help users curate their notebooks. If a notebook has been deleted or superceded by one of its forks, a user who opens it should be warned. And operations on multiple notebooks and folders of notebooks should be supported to make curation easier.

Collaboration is hard. While forking provides a simple way to continue someone else's work, working in teams is a lot more complicated. The forking feature is popular but it poses a problem, too, as dozens of versions of a notebook start proliferating.

One solution we are considering is to allow overwriting the current state of a notebook with the current state of another fork of the notebook.

More general solutions would be serious engineering endeavors. Even GitHub's generous web interface does not support merging where there are conflicts. Actual sharing of notebooks would probably require moving toward git's notion of decentralized code repositories, because GitHub gists do not support shared ownership.

7 Conclusions and Future Work

We presented a case for an environment that supports the visual analytics process for work by teams. The process includes data acquisition, exploratory data analysis, code development and deployment.

Building on previous work to define requirements for visual analytics, we designed an environment for exploration, sharing, presentation and publishing. We implemented it in the RCloud prototype.

We deployed RCloud in a community of working data scientists. Experience with the prototype provides evidence that data science teams and the organizations in which they work benefit from having capabilities to support collaboration and to integrate the entire visual analytics process. We found that features for sharing and publishing were eagerly adopted. Features for single-user data exploration, that compete with existing mature tools, were not accepted as readily. Some experienced users fashioned their own workflow so they could keep using familiar EDA tools.

This study has provided a step toward practical "DevOps for data science" and reproducible, publishable experiments. Possible next steps are to incorporate richer recommendation techniques, to provide fine-grained information security, and to improve the usability of the human interface.

RCloud code is available at `github.com/att/rcloud/` under an MIT open source license.

REFERENCES

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

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

[3] R. A. Becker. A Brief History of S, 1984. http://cm.bell-labs.com/cm/ms/departments/sia/doc/94.11.ps.

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

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

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

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

[8] 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.

[9] 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.

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

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

[12] 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.

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

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

[15] 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.

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

[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. Lucas and S. F. Roth. Exploring information with Visage. In *Conference Companion on Human Factors in Computing Systems*, CHI '96, pages 396–397, New York, NY, USA, 1996. ACM.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] A. Perer and B. Shneiderman. Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 265–274. ACM, 2008.

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

[29] RStudio. ggvis: Interactive grammar of graphics for r, 2015. https://github.com/rstudio/ggvis.

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

[31] 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.

[32] 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.

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

[34] 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.

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

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

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

[38] 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.

[39] 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.

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

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