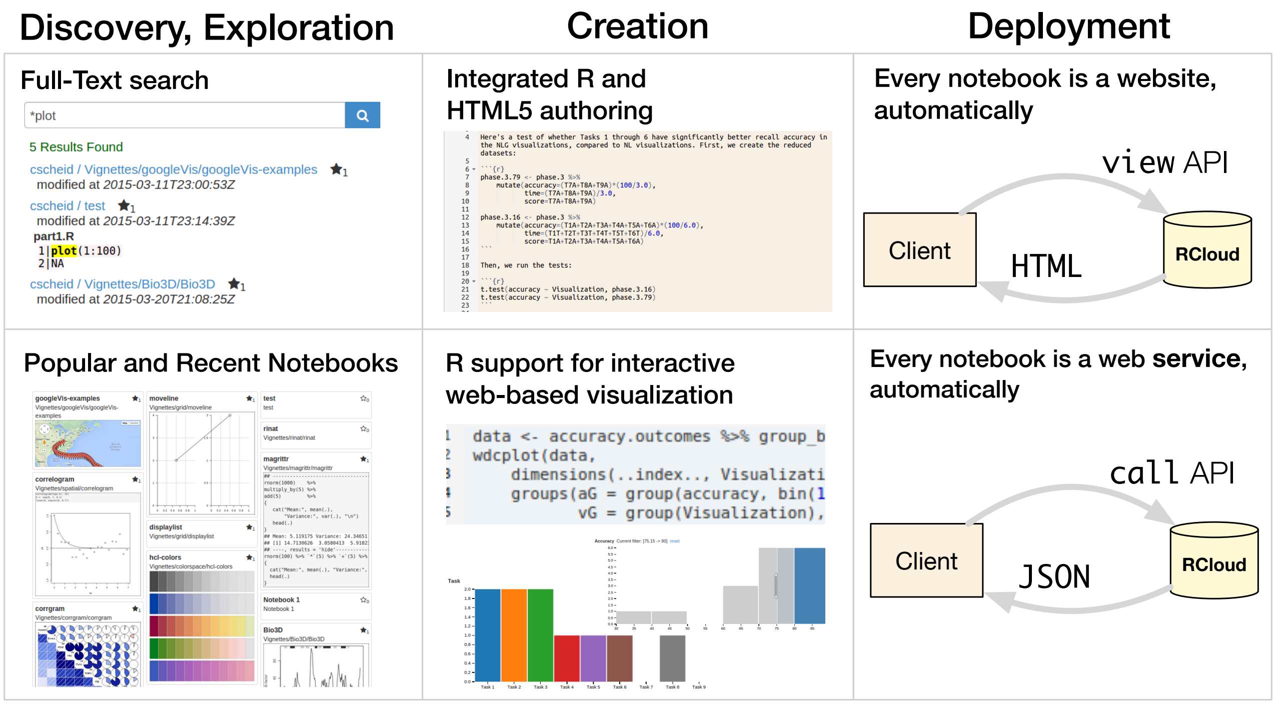
Collaborative Visual Analysis on the Web with RCloud

Category: Research


Discovery, Exploration
Creation
Deployment
Full-Text search
Integrated R and HTML5 authoring
Every notebook is a website, automatically
view API
Client
HTML
RCloud
Popular and Recent Notebooks
R support for interactive web-based visualization
Every notebook is a web service, automatically
call API
Client
JSON
RCloud
Accuracy Current filter: [75.15 -> 90] reset
Task

*plot

5 Results Found

cscheid / Vignettes/googleVis/googleVis-examples ★1
modified at *2015-03-11T23:00:53Z*

cscheid / test ★1
modified at *2015-03-11T23:14:39Z*

part1.R

```
1|plot(1:100)
2|NA
```

cscheid / Vignettes/Bio3D/Bio3D ★1
modified at *2015-03-20T21:08:25Z*

~~~
Here's a test of whether Tasks 1 through 6 have significantly better recall accuracy in
the NLG visualizations, compared to NL visualizations. First, we create the reduced
datasets:

```{r}
phase.3.79 <- phase.3 %>%
    mutate(accuracy=(T7A+T8A+T9A)*(100/3.0),
           time=(T7A+T8A+T9A)/3.0,
           score=T7A+T8A+T9A)

phase.3.16 <- phase.3 %>%
    mutate(accuracy=(T1A+T2A+T3A+T4A+T5A+T6A)*(100/6.0),
           time=(T1T+T2T+T3T+T4T+T5T+T6T)/6.0,
           score=T1A+T2A+T3A+T4A+T5A+T6A)
~~~

Then, we run the tests:

```{r}
t.test(accuracy ~ Visualization, phase.3.16)
t.test(accuracy ~ Visualization, phase.3.79)
```
```

```
data <- accuracy.outcomes %>% group_b
wdcplot(data,
     dimensions(..index.., Visualizati
     groups(aG = group(accuracy, bin(1
                vG = group(Visualization),
```

Fig. 1. An overview of features in RCloud. RCloud supports 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 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 work to production web services.

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: *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 two years, 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 [37]. 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 [38].

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, in wikis, project documents and by other means. Results of experiments are often shared first by copying static output, usually a screen shot or image output. By the time decisions are made based on the results of that screenshot, data and processes may have changed so much that the decision can be wrong.

This situation can affect how data science teams work; as analysts and tool builders, 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) "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."

Currently, these concerns are being addressed *alongside* the visual analytics environment. Some symptoms of the 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 involvement 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 identified 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 [21]). *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

Broadly, the entire field of information visualization and visual analytics revolves around improving how users solve problems involving data. In this section, we focus specifically on system work relating to problem-solving environments and multiple users.

**Social Data exploration and Analysis** ManyEyes [40] 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

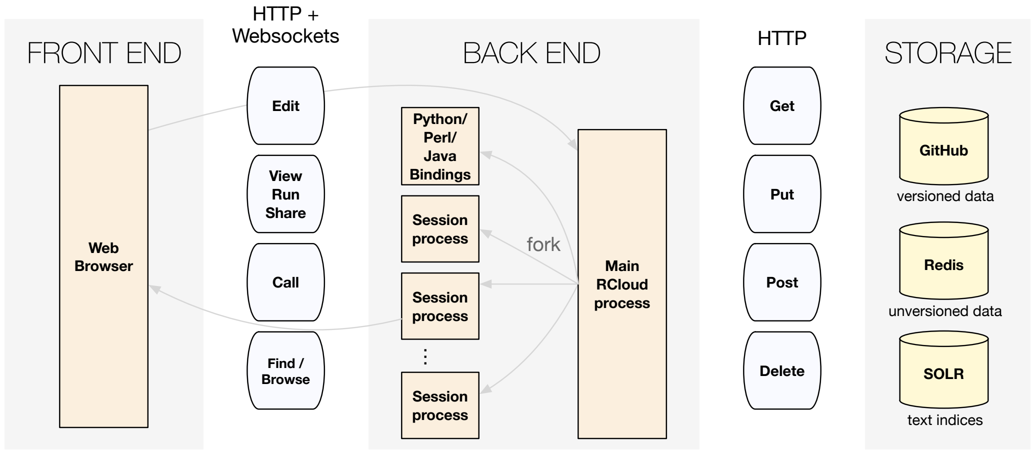


Fig. 2. A diagram of RCloud's architecture.

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 [22]. 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 [42]. Project Jupyter [29] (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[32]. 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 mentioned, one central issue 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 analysis 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 [33] defines a schema and semantics for automatically deriving user interfaces from workflows, while Crowdlabs exposes these capabilities on a website feature [24] 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 quite a few tools recently developed for quickly sharing small programs on the web, including bl.ocks [6], 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. If Javascript eventually becomes the lingua franca of exploratory data analysis, we can foresee building a simpler version of RCloud where all execution happens either on the client side or via web services. Unfortunately that is not a realistic assumption at present, making these stand-alone solutions unsuitable for our purpose. 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 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.

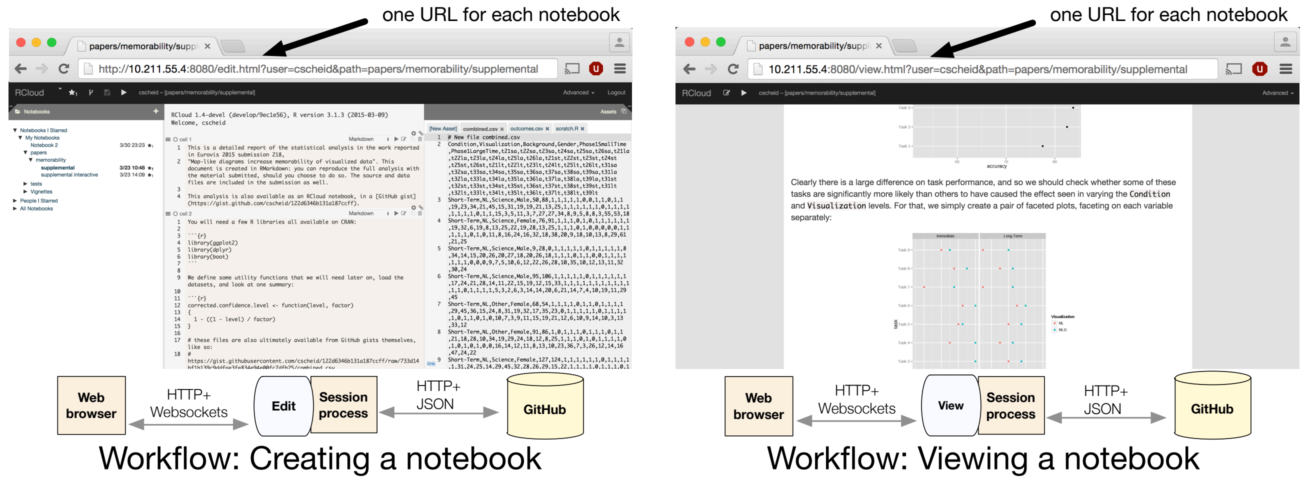
**Needs of data analysts** Kandel et al.'s interview study points out the typical "explore", "model", "report" cycle in enterprise data analysis [21]. There are many discontinuities in this cycle that cost time and effort to overcome. RCloud seeks to reduce this mismatch. Kandel et al also point out 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 [20]. 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 is not recorded or available in a reusable form even within the same organization.

Heer and Agarwala identify many design considerations for collaborative visual analytics [14] 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.

The need for integrating statistics and visualization has been highlighted in previous studies and is widely understood by various technical communities [28] Lucas and Roth were early advocates of combining data exploration with presentation and publication [23].

There has been noteworthy work on specific techniques to support collaborative or social code development and data analysis, such as


one URL for each notebook
http://10.211.55.4:8080/edit.html?user=cscheid&path=papers/memorability/supplemental
cscheid – [papers/memorability/supplemental]
Markdown
Web browser
HTTP+ Websockets
Edit
Session process
HTTP+ JSON
GitHub
Workflow: Creating a notebook
one URL for each notebook
10.211.55.4:8080/view.html?user=cscheid&path=papers/memorability/supplemental
cscheid – [papers/memorability/supplemental]
accuracy
Immediate
Long.Term
task
Visualization
NL
NLG
Web browser
HTTP+ Websockets
View
Session process
HTTP+ JSON
GitHub
Workflow: Viewing a notebook

```
RCloud 1.4-devel (develop/9ec1e56), R version 3.1.3 (2015-03-09)
Welcome, cscheid
```

```
This is a detailed report of the statistical analysis in the work reported in Eurovis 2015 submission 218,
"Map-like diagrams increase memorability of visualized data". This document is created in RMarkdown: you can reproduce the full analysis with the material submitted, should you choose to do so. The source and data files are included in the submission as well.

This analysis is also available as an RCloud notebook, in a [GitHub gist](https://gist.github.com/cscheid/122d6346b131a187ccff).
```

~~~
You will need a few R libraries all available on CRAN:

```{r}
library(ggplot2)
library(dplyr)
library(boot)
~~~

We define some utility functions that we will need later on, load the datasets, and look at one summary:

```{r}
corrected.confidence.level <- function(level, factor)
{
  1 - ((1 - level) / factor)
}

# these files are also ultimately available from GitHub gists themselves, like so:
# https://gist.githubusercontent.com/cscheid/122d6346b131a187ccff/raw/733d14bf1b139c9ddfae3fe834e94e00fc7dfb75/combined.csv
```

Clearly there is a large difference on task performance, and so we should check whether some of these tasks are significantly more likely than others to have caused the effect seen in varying the `Condition` and `Visualization` levels. For that, we simply create a pair of faceted plots, faceting on each variable separately:

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 provides a lightweight, low-friction mechanism for sharing results which we discuss in detail in Section 5.

social bookmarking [25] [15] and crowdsourcing [9]. 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 similar fields. By making it convenient for data scientists to expend just a little additional 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 next give a description of the system architecture, and how it enables capabilities that satisfy the requirements as described.

3 The System

The starting point in our design was to take advantage of the current infrastructure of web software as much as possible. HTTP, despite its deficiencies, has become the lingua franca of distributed interprocess communication. This meant that if our system could speak HTTP natively, then it would be possible to provide a low-friction path from development of an interactive data analysis script to an automated *web service*, essentially a remote function call over the web, which can be invoked by higher-level tools. An example of this sort of progression is described in Section 4.

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, this is not novel; executable documents like this are a feature of many other systems, including Mathematica, IPython and Sage.

Notebooks can be executed one cell at a time during an interactive editing session, providing a similar experience to traditional read-eval-print loops, or can also be executed all at once, providing an experience 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 as the notebook is being developed. 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), we believe that the alternative of losing automatic sharing is worse (and see Section 5 again for users reporting their satisfaction over the low-friction shareability).

When we combine immediate deployment with the ability to refer to the latest version of a notebook by name, 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 [18] and VisTrails's version tree [8], 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* [11], 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. Advanced users then have access to features like command-line checkout, history editing, and so on.

3.2 Reputation and Interest: starring

One side advantage of centralizing the execution and storage of notebooks is that it becomes feasible to collect usage information that would otherwise be lost. In the case of social data visualization platforms, we would like to exploit usage data to help analysts find content of interest, whether the content is actual source code or accompanying data. The standard way to achieve this is 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, then you should like this one too"). To compute 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, to encourage its 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 [17].

Implicit signaling of interest is supported by keeping click-through counts [19] 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, in the style of Fast et al's Codex system. [9].

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 [41], 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* [26]. Specifically, 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. This number is interpreted as an opaque identifier to a function. Because these identifiers are cryptographically safe, all that the Javascript layer can do is return them to the R side, in a message requesting that this function be called on its behalf. In that case, the results of this function call might include *new* opaque identifiers, exposing new "capabilities" to the web browser.

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 respond to user input in the browser, giving them access to features ranging from password prompts, to the currently selected set of points in an interactive linked brushing 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 back to analysis functions in R without having to define an additional protocol between the processes. From the Javascript side, an RPC call into the R process is just another HTTP request. From the R process, a call into Javascript looks like just another function call.

4 Case Study: visual exploration of topic modeling in RCloud

As an example, we present a simple real-world application deployed under RCloud.

The application is an implementation of LDAVis, a recent method for visualizing large text corpora. It was designed to help 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 some manual intervention and interpretation [35].

LDAVis was developed by two technical staff members at AT&T Labs, and originally was written in RStudio's Shiny [31], a framework for developing R applications for the web. While Shiny provides outstanding ease of development, it turns out that discoverability and deployment are equally important aspects in the lifecycle of an internal application (see Section 5). In these aspects, RCloud provides a simple solution: *all developed notebooks are automatically deployed*.

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 R code written by the developers. The analysis module is, ultimately, 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 the document. To expose patterns in the relationships between topics, LDAVis employs a combination of 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 the 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 [7]. One of the most popular web-based visualization libraries for R is ggvis [30], 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, by extension, Shiny) are a subset of those in Vega [36]. Custom interactions in LDAVis (such as hovering over a topic, topic cluster, or word) do not appear to be available yet in Vega [36], although the required components for reactive interaction were recently described by Satyanarayan et al. [34]. Although custom Javascript was required, the flexibility is welcomed by many web developers.

The LDAVis application is simple, but highlights some 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 with 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 (since analysts simply write R in the style they are used to), and the front end visualization side is simpler for the web 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 all RCloud notebooks.

5 Interview Study

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

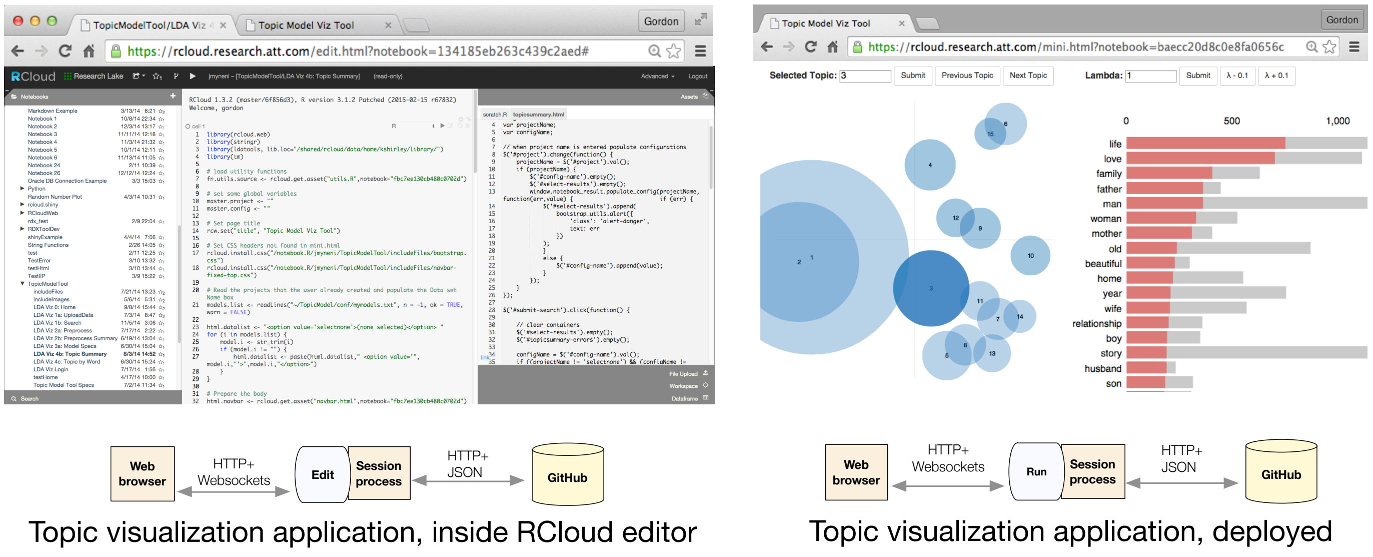
The subjects differed in their evaluation of its benefits and disadvantages, as described below.

5.1 Sharing of results

Sharing of notebooks 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. Lilo 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, notebook sharing can provide a starting point for coding. Lilo says, "The best part is how



Markdown Example	3/13/14 6:21	☆2
Notebook 1	10/8/14 22:34	☆1
Notebook 2	12/3/14 13:17	☆1
Notebook 3	11/11/14 12:18	☆1
Notebook 4	11/3/14 21:32	☆1
Notebook 5	10/1/14 12:11	☆1
Notebook 6	11/13/14 11:05	☆1
Notebook 24	2/11 10:39	☆1
Notebook 26	12/12/14 12:24	☆1
Oracle DB Connection Example	3/3 15:03	☆1
▶ Python		
Random Number Plot	4/3/14 10:31	☆1
▶ rcloud.shiny		
▶ RCloudWeb		
rdx_test	2/9 22:04	☆1
▶ RDXToolDev		
shinyExample	4/4/14 7:06	☆1
String Functions	2/26 14:05	☆1
test	2/11 12:25	☆1
TestError	3/10 13:32	☆1
testHtml	3/10 13:44	☆1
TestIIP	3/9 15:22	☆1
▼ TopicModelTool		
includeFiles	7/21/14 13:23	☆2
includeImages	5/6/14 5:31	☆2
LDA Viz 0: Home	9/8/14 15:44	☆2
LDA Viz 1a: UploadData	7/3/14 8:47	☆2
LDA Viz 1b: Search	11/5/14 3:06	☆1
LDA Viz 2a: Preprocess	7/17/14 2:22	☆1
LDA Viz 2b: Preprocess Summary	6/19/14 13:04	☆1
LDA Viz 3a: Model Specs	6/30/14 15:04	☆1
LDA Viz 4b: Topic Summary	**8/3/14 14:52**	☆1
LDA Viz 4c: Topic by Word	6/30/14 15:24	☆1
LDA Viz Login	7/17/14 1:56	☆1
testHome	4/17/14 10:00	☆1
Topic Model Tool Specs	7/2/14 11:34	☆1

```
RCloud 1.3.2 (master/6f856d3), R version 3.1.2 Patched (2015-02-15 r67832)
Welcome, gordon
```

```r
library(rcloud.web)
library(stringr)
library(ldatools, lib.loc="/shared/rcloud/data/home/kshirley/library/")
library(tm)

# load utility functions
fn.utils.source <- rcloud.get.asset("utils.R",notebook="fbc7ee130cb480c0702d")

# set some global variables
master.project <- ""
master.config <- ""

# Set page title
rcw.set("title", "Topic Model Viz Tool")

# Set CSS headers not found in mini.html
rcloud.install.css("/notebook.R/jmyneni/TopicModelTool/includeFiles/bootstrap.
css")
rcloud.install.css("/notebook.R/jmyneni/TopicModelTool/includeFiles/navbar-
fixed-top.css")

# Read the projects that the user already created and populate the Data set
Name box
models.list <- readLines("~/TopicModel/conf/mymodels.txt", n = -1, ok = TRUE,
warn = FALSE)

html.datalist <- "<option value='selectnone'>(none selected)</option> "
for (i in models.list) {
    model.i <- str_trim(i)
    if (model.i != "") {
        html.datalist <- paste(html.datalist," <option value='",
model.i,"'>",model.i,"</option>")
    }
}

# Prepare the body
html.navbar <- rcloud.get.asset("navbar.html",notebook="fbc7ee130cb480c0702d")
```

```javascript
var projectName;
var configName;

// when project name is entered populate configurations
$('#project').change(function() {
    projectName = $('#project').val();
    if (projectName) {
        $('#config-name').empty();
        $('#select-results').empty();
        window.notebook_result.populate_config(projectName,
function(err,value) {                              if (err) {
            $('#select-results').append(
                bootstrap_utils.alert({
                    'class': 'alert-danger',
                    text: err
                })
            );
            }
            else {
                $('#config-name').append(value);
            }
        });
    }
});

$('#submit-search').click(function() {

    // clear containers
    $('#select-results').empty();
    $('#topicsummary-errors').empty();

    configName = $('#config-name').val();
    if ((projectName != 'selectnone') && (configName !=
```

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

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 time for me."

Evelyn 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 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 problems with data sources.

Some users, who tried RCloud and were not able to continue for organizational reasons, miss certain capabilities. Leith "[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." She said, "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."

5.2 Forking

The ability to start work where someone else let off, by forking, 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). 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 initially didn't anticipate support forking one's 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 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."

A common but somewhat problematic use of forking is to change parameters. Wendy says, "I've been forking other people's notebooks [because] I want to run them 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." Evelyn 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 requires expertise. Tool builders see a need for facilities to make it easier. Allison 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 they can [still] fork it if they want to."

5.3 Automatic source control

Automatic source control is also a popular feature. 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 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 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."

5.4 Discovering others' work

Many users find the search function valuable. 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 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 number of notebooks grows further. We explore some ideas in Section 6.

5.5 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 also helps Allison's application: "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."

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 prefers to do analysis in another tool, and paste code into RCloud for sharing.

This is mainly out of familiarity. Evelyn 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 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 command line, rearranging a file so the good code is at the top of the file. When he's done, all the right code is at the top. RCloud does not readily support this workflow, but Coby often shares 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. Joy 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. Lilo says,

> 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.

Simply the difference between a cell and the command prompt can confuse some people. Kenyon 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 I was editing one at the bottom, I could type it but then it would automatically execute [by pressing enter].

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." Kenyon 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."

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: they either hold too much code or too little. 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 Evelyn 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." Another complaint related to the human interface was that cells can take up substantial vertical space, requiring scrolling.

Some users would like a way 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. In Coby's work, there is often a "long tail". The first cell may take just seconds to run, the next cell minutes, and the last cell 5 hours. 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 avert this pitfall, some users write code within cells to explicitly cache results. Coby ends up "littering the notebook with little switches that comment out the slow parts" and either load 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 a calculation only if it's needed.

5.8 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, 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. She 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.

Evelyn, who writes packages and example notebooks for them, mentioned the accumulation of dead notebooks. When Evelyn 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.

The problem of dead notebooks is compounded by changing packages and example notebooks at the same time. Evelyn 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 they complain."

Evelyn tried to keep his notebook tree well organized, but this didn't help, because some people kept old links in their email, or forked notebooks which had become obsolete. By design, notebooks that are deleted in the user interface are not purged from the repository. Now he says he is scared to share notebooks: “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 in 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 to 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 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 [26] recently added to the Rserve protocol [39] 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. More sophisticated approaches, such as information flow analysis, and

	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.

query languages that ensure differential privacy, are active research topics, and many problems remain [27]. 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 higher latency when commands run, because RCloud saves a cell to a 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 switching between the keyboard and mouse, which can be cumbersome. Shortcut keys can defined as a workaround, but it would be better to reduce the need for them.

Other complaints mostly focus on layout, and can probably be addressed by offering better customization, and designs such as tabbed views.

Discovery is hard While we 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 unwieldy with hundreds of users and many hundreds of notebooks. In addition, our choice to retain notebooks even when they have been deleted exacerbates the situation, since obsolete and broken notebooks stay around forever.

One thing to try is tagging and filtering 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, which is not always reliable. It seems better if possible to lean more on passive metrics, such as how often a notebook has been run.

The system also needs to help users to curate notebooks. If a notebook has been deleted or superseded 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 proliferate.

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.

A more general solutions would be a serious engineering endeavor. Even GitHub's admirable web interface does not support merging where there are conflicts. Unrestricted 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, presenting 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 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] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

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

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

[32] 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.

[33] 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.

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

[35] 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.

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

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

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

[39] 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.

[40] 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.

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

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