The RED R Journal

A peer-reviewed, open-access publication

The Red-R Journal No.1-08/08/2011 ISSN 2161-2498

The Red-R Framework for Integrated Discovery

by Kyle R. Covington and Anup Parikh

Abstract

The Red-R Framework is a visual programming environment for data analysis. The framework focuses on data interactivity, readability, and share-ability. Interaction is provided in a canvas in which users visually wire functional modules together. A logging system and saving modules ensure that data are reproducible, shareable, and well documented. The underlying philosophy of Red-R is that users with all levels of expertise should use the same software to analyse data. While Red-R provides an environment in which users can easily interact with data using graphical user interfaces, it also provides the underlying power of R and Python for data analysis. Herein, we describe the Red-R Framework.

Introduction

R [1] is a high-level interpreted languages which has gained wide spread usage in the data mining community. **R** provides numerous well tested and maintained algorithms for data modelling, machine learning as well as data visualization. Thanks to a well-established system for extending functionality and its active community of users and developers, **R** maintains an extensive collection of packages in repositories such as Rforge [2], Bioconductor [3] and CRAN.

While **R** is a powerful tool for data analysis, one major limitation is it's interface. The command line interface (CLI) is the primary method of interacting with **R**. While technical users, those with a high level understanding of programming or statistics, require the power and flexibility of CLIs, this interface represents a major barrier to use by non-technical users, those with-out a high level understanding of programming. Even for those that have expertise in other data mining tools, **R** can be difficult to use due to the differences in syntax resulting from the distributed development model that made **R** a success in the first place. For example, to access the row names of a table, functions use the syntax rownames, row.names, Rownames, or row_names depending on the package and function. Many graphical user interfaces exist that can make **R** functionality accessible to non-technical users without extensive training, but often limits functionality to well defined and implemented tasks. GUIs such as R Commander [4], Deducer[5] and Rattle provide frameworks with point and click functionality through menus and tool-bars. However, these packages often perform only a few analyses on one data object and lack the ability to perform sequential data analysis or analyses in parallel. Thus, a more abstract framework would go far to allow users to interact with high level statistical and data models. It is also essential that a programming environment can be extended to provide new tools for users developed by the community.

Collaboration between experts in different domains is essential for productive investigations. However, as the volume of data and technical complexity of the analysis increases, non-technical users cannot participate in the data exploration and analysis. Furthermore, due to the complexity, data analysis scripts are difficult to understand and share, even with other computational users. An unfortunate example of these difficulties is clearly evident in biomedical research. There is little interaction among biologist, statisticians, and computational biologist to conduct meaningfully collaborations on data intensive projects. The lack of powerful yet user friendly tools greatly limits analysis interpret-ability and the ability of non-computational users to participate in the data analysis process. A common framework would greatly contribute to collaborations between technical and non-technical users.

A further limitation is that data and analyses are often separated. Both **Python** and **R** store the working session (the "image" in **R** and pickled data or text files in **Python**) separately from any input script. This practice allows data and analyses to be separated and can lead to confusion in the application of functions to the data.

Thus, two major problems with collaborative data analysis are access to programmatic functions to novice users, and grouping analysis pipelines with analysis data. To address these challenges we have developed Red-R, a user friendly visual programming and data analysis framework for **R**. Red-R makes the advanced functionality of **R** available to non **R** experts by hiding the CLI complexity behind a visual programming interface. Analyses are performed by visually linking a series of widgets together that perform simple tasks such as read, manipulate, and interactively display data. These pipelines, representing both the data and analysis, can be

easily shared with others as native Red-R sessions or printable reports. Along with the visual representation, all the **R** code is readily available for review by the computational collaborators.

Additionally, this framework leaves open the ability to write programs and execute functions in Python, without the need to interact with **R** at all. Red-R tools can be easily made that make use of the **Python** language to execute programs written in a variety of languages. Red-R can also be easily extended by other developers by simply installing the packages through the Red-R interface (Package Manager).

Red-R Core Functionality and Architecture

We designed Red-R to achieve four main goals: First, to provide an easy to use interface for non-technical users. Second, to make analysis pipelines more readable and standardized through visual representation. Third, to allow users to easily share analysis logic, parameters, and data with others. Forth, to provide a channel for technical users to share knowledge with non-technical users.

Red-R consists of the Core software and a repository of packages that provide computing functionality. The Core software provides a point and click interface for accessing the repository of widgets to build graphical analysis pipelines. In addition the core provides functionality for saving/loading pipelines and generating printable reports.

Red-R packages consist of a set of widgets and documentation for performing specific analysis. Although Red-R Core is Open Source, packages are independently developed and licensed by the author. For those packages available under and Open license, we provide a repository for packages at http://www.red-r.org.

Graphical Interface

The main component of Red-R core is the canvas on which analysis pipelines are built (Figure 1). The canvas provides the interface to add, delete and make connections between widgets (Figure 1A). Widgets can be added to the canvas by using the widget tree menu or through the search bar (Figure 1B,C). While building a pipeline, the canvas recommends new widgets to add based on connection history and suggestions from the widget developer (Figure 1D).

The canvas provides numerous forms of visual feedback to inform the user of the current state of widgets and the flow of data between widgets. While building pipelines, the canvas provides visual cues to which widgets have compatible signals and can be connected by turning the color of compatible sockets green. The canvas also provides feedback on the flow of data between widgets by altering the color of the connections, with active signals colored in green and inactive ones colored in grey (Figure 2). For example, Widget specific messages and errors are presented as small icons above widgets on the canvas.

One barrier to any programming environment is knowing which functions to use when. A key solution to this barrier is the ability to search and screen potential functions. As mentioned above, Red-R provides a search bar which dynamically loads and refreshes search terms while the user is typing (Figure 1B, Figure 3). In addition to widgets and templates (discussed below) the search bar always shows options for accessing help and further documentation on the Red-R website or on the users local help documentation. By integrating the default search abilities users can directly access help from within the Red-R application.

Though Red-R implements GUIs which can be used by both technical and non-technical users, technical users are allowed access to the underlying **R** session through the R Executor widget. R Executor allows the user to execute functions in the **R** session. This widget can be used to inspect elements in the **R** session or to perform other operations for which either widgets do not exist or which are easier to accomplish in the **R** session directly. Objects can be emitted from the R Executor widget as a signal using the Send functions of the widget.

Widgets

Widgets are the building block of Red-R analysis pipelines and can be thought of as functions in R. They take input, perform some data transformation, and output the data as a Red-R signal. We followed the Unix tools philosophy [6] when developing widgets. That is, rather than developing widgets to perform multiple functions, each widget performs a specific task and multiple widgets are linked together to complete the analysis.

There are three main types of widgets: input widgets, transformation widgets and output widgets. The data input widgets are generally the start of a pipeline and import data from external sources, such as files or databases. A series of data transformation widgets perform analysis and finally pass the results to a data output widget. The output widgets can write data to files or visual the data as tables or graphs.

Signals

Signals in Red-R are data container objects passed between widgets. The core determines the compatibility between widgets based on the input and output signals types. Signals also contain methods to convert data to other compatible signal types. As an example, the vector signal can convert to a single column matrix signal. This framework allows widgets to specify exactly the type of data they expect, but provides tremendous freedom to users to make connections between widgets the package developers may never have imagined.

Packages

Packages are collections of widgets and signals. Like **R**, Red-R is a package based system whereby developers extend the functionality by creating new packages. Users install new packages in Red-R using the Package Manager interface (Figure 4). The package manager allows installation, deletion, and updating of packages compatible with their version of Red-R. Red-R, by default, checks for updates to packages pe-

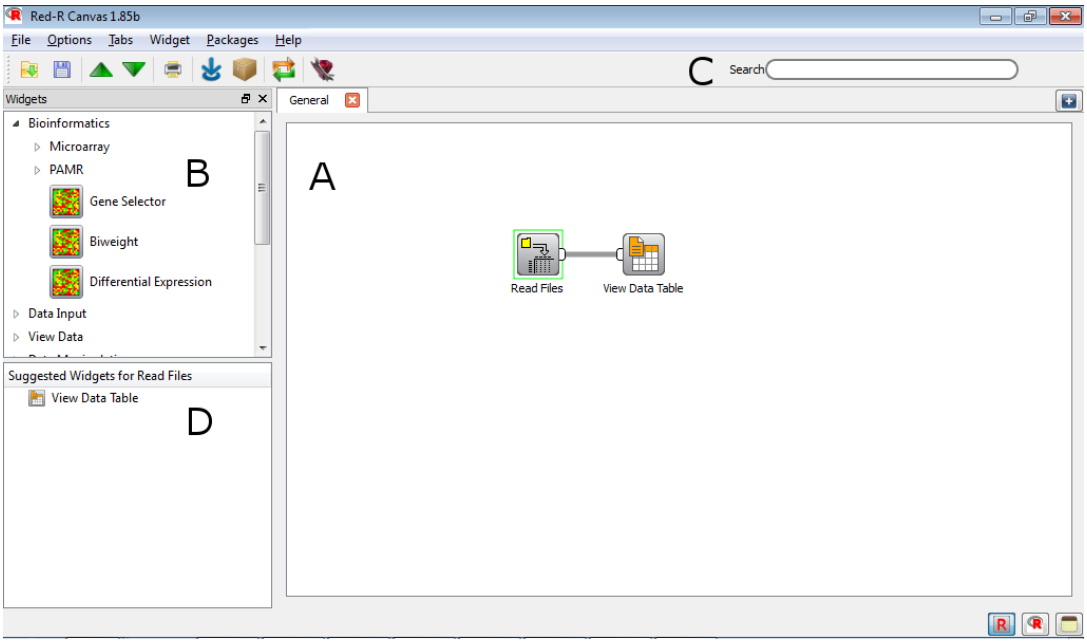

Figure 1: Screen capture of the Red-R canvas. A. The canvas area where widgets are connected. B. The widget tree view where widgets are organized by package or function. C. The search bar where widgets are searched by text. D. Widget suggestion area where widgets recommended for connection with the selected widget are displayed.

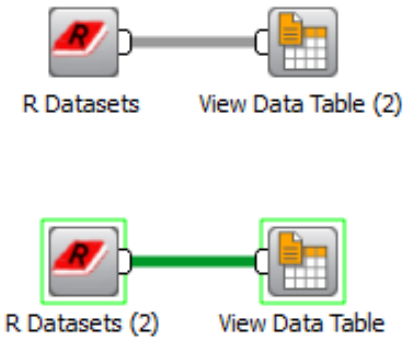

Figure 2: Demonstration of Red-R signal prompts. Active signals are shown in green and inactive ones are shown in grey.

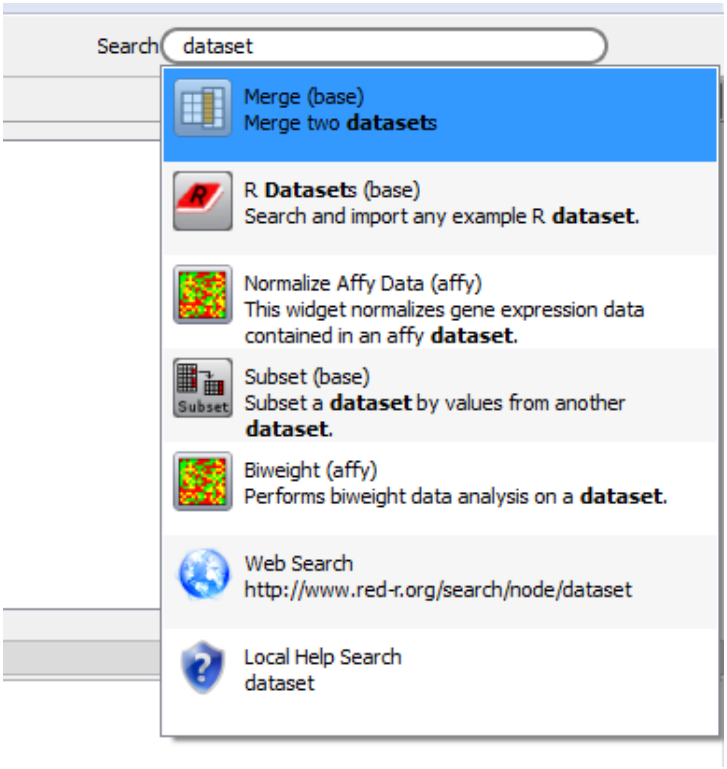

Figure 3: Demonstration of the Red-R searchbar. The top 10 widgets and templates are listed first then default tabs for online searching and local help are available.

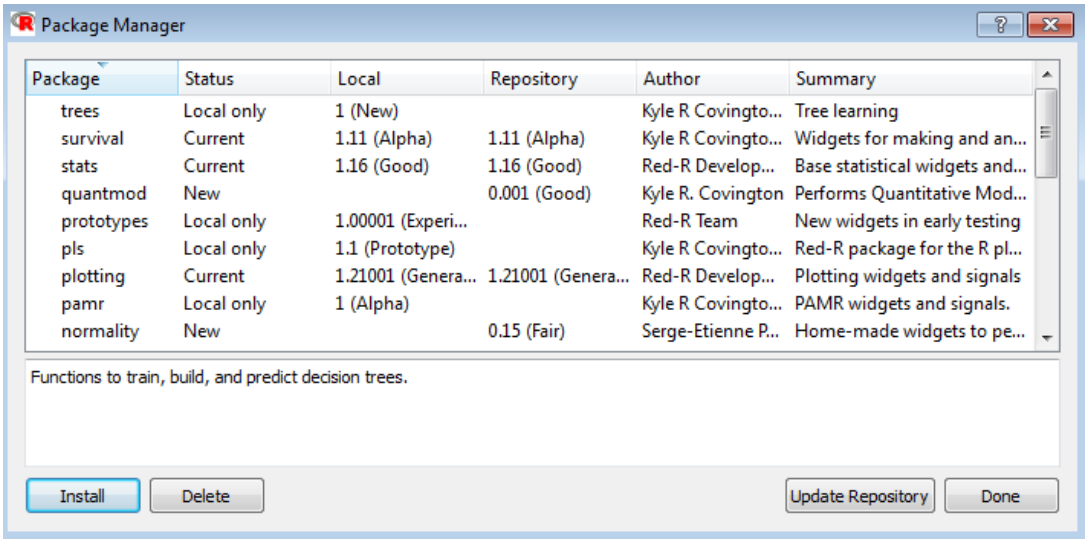

Figure 4: The Package Manager in Red-R. Users can use this interface to install, update, and remove packages from Red-R to extend functionality.

riodically and will alert the user that new package updates are available.

Packages can also be installed directly. Package developers can share packages stored in zip archives. This allows developers to distribute packages outside of the Red-R archive. While this method allows freedom to package developers, packages distributed in this way will not be available for automated download when sharing a schema saved using those packages.

Package development is beyond the scope of this manuscript. However, more information on package development is available on the Red-R website (`www.red-r.org`).

Share Red-R Pipelines

Communication of analysis results is vital for scientific discovery. There are three components to an analysis: logic, parameters, and data. The logic outlines the series of steps used to transform the raw data into the final results. In Red-R, logic is encoded by the user in the form of the pipeline itself. The specific parameters are represented by the the parameters input into the widgets.

Data are represented by the signals that pass between widgets. Output signals of widgets contain new data objects that the next widget can further manipulate. This insures that no widget can overwrite the work of another widget and therefore maintains analysis traceability. Analyses are thus constructed of both a series of functions and intermediate data objects. All of this information must be passed to other users for complete reproducibility. Many other tools separate the data from the analyses. Analyses without data and data without detailed instructions on how that data is derived are significantly inferior to the two combined.

Generally, programmers will save a "script" in one place and some of the intervening data as tables or perhaps other objects in another. When analyses are saved often much of the intervening models are lost and all that remains are the beginning and ending tables, lists, or other select output. Red-R strives to save these two elements, the analysis and the data, together. As such, the data are never separated from the analysis. This makes sharing the data with later users more complete. Using Red-R, there is no confusion as to which table in a directory is intervening data, which is the input data, and which is the final data that is to be consumed.

In addition to sharing saved sessions of Red-R, users can generate printable reports of sessions. While these reports do not hold all of the data of a Red-R session, they may be useful for sharing analyses with users that do not have Red-R installed, or for printing and making further notes of analyses. The automated report generation of Red-R makes a Open Document style of document. Red-R documents should not represent the final presentation of data but rather a starting point for further documentation.

Auto-Processing Pipelines

A critical limitation of most analysis environments is that functions are executed, often, without a "pipeline" model for data processing. The user will often execute one function at a time either by interacting with a CLI or GUI. In Red-R analyses are performed in the context of a pipeline. In pipelines data is passed from one widget to another within a model. In Red-R, the pipeline is defined by the connections of widgets and signals. Because there is a pipeline in place, as users update and change analyses, downstream widgets can be updated to reflect the fact that new data are available.

Many widgets in Red-R will automatically perform their processing and transmit that data to downstream widgets. These auto-processing widgets allow users to quickly put together analysis pipelines that can then be used to screen parameters for data analysis. Other widgets require some input from the user before processing. The decision to make a widget that processes on data connection or to force the user to initiate processing is made by the widget developer. The choice is usually based on the speed by which processing can take place for the function of interest. If the function can be performed quickly then the developer will likely implement the auto-processing functions. Most of the widgets in the core **R** packages will process data as soon as it arrives.

Widgets that are designed for auto-processing execute the "commit" function when new data is connected. For example, the widget named Transpose in the base package transposes table data, making the rows into columns in the new data and columns into the rows. If a user places the Transpose widget in a pipeline and changes the data in the widget upstream of the Transpose widget, the Transpose widget will automatically transpose the new data and send it from the output socket. The auto-process feature of Red-R allows users to quickly build complex analysis pipelines to process data quickly, and ensures that all data are up to date as new data are selected.

Expert Templates

One critical limitation to using many analysis frameworks and programs is demonstrating to users how to perform analyses. While technical or expert users may have experiential knowledge about how an analysis should be performed, non-technical or novice users may not know which functions to connect to achieve their goals. The problem essentially reduces to changing an idea, i.e. "I want to know which diet makes chicks grow largest", into an analysis pipeline, i.e. "Read in data, check data integrity, model effect of diet on growth over time, check significance of effects, generate plots of data".

To accomplish this, Red-R supports the idea of templates. Templates are collections of widgets and connections with no data pre-set. For example, there might be a template named "Read Merge" which is composed of two widgets that read data from files and a widget that merges those files together. Users could build this themselves by loading each widget separately, but all three could be placed on the canvas by loading the "Read Merge" template. Furthermore, templates can be pre-set with different settings in each widget, so templates might have several of the same type of widget, but each with different settings. Users can develop templates that have pre-set settings, but these can't be dependent on data as data may change from run to run. Templates are indexed in the

Red-R drop-down menu and can be loaded as though they were widgets themselves. Expert users and package developers can save templates and annotate them with a description, i.e. "Compare and plot effects over time", which users can then search for. Users can then share and search for these so that they are not limited to only calling function names but can instead call task names.

Users can also develop and save their own templates to use later. If a user finds that she is constantly connecting the same widgets together or is changing settings from the default often, the user can save the widgets as a template and reload them on command. Thus, users can extend and create new templates to solve specific analysis problems. These templates can then be shared with other users, generating families of analysis templates. By sharing these templates users can better learn from other users and build upon their analysis experience and extend those experiences for specific tasks.

A Community of Analysis

Red-R's philosophy of data sharing and analysis extends to the community at large. Red-R is integrated with the Red-R.org website (Figure 5) and search terms can be submitted to the Red-R.org website using the search bar menu. Users are encouraged to search for, develop, share, publish, and comment on analysis pipelines, templates, and packages. Integrating the software with the community allows users to query the community of discussions and thought about analysis with users as they are performing analyses.

Discussion

Creating a visual representation of the analysis greatly increases interpretability and gives non-computational users the ability to explore the data and further analysis it without knowing **R** code syntax. Along with the visual representation, all the **R** code is readily available for review. This functionality in Red-R can create more effective collaborations by allowing users to perform complex analyses and share them so that others can easily understand the analysis and further explore the data.

By combining the power of **R** and **Python** with a simple point and click user environment we are able to markedly enhance the collaboration of technical and non-technical users. Technical users can have access to all of the power of **R** that they are accustomed to, through utilities such as RExecutor, while still generating objects that can be used in other interactive widgets. Red-R provides an input widget that allows existing **R** sessions to be imported into the Red-R environment. This allows **R** experts to conduct analysis with all the power of a CLI and then expose key data and parameters to users in Red-R. Non-technical users then have the ability to explore the analysis. For example subsetting the data to examine a specific group or generating a new graphic based on altered parameters. The altered pipelines can then be validated by technical users or experts. This iterative process of collaboration is very powerful and only possible with tools, such as Red-R, that allow data and analysis to be easily shared.

The ability to share one analysis environment between technical and non-technical users represents a leap forward in collaborative data analysis. This new ability for users to interact with powerful analysis tools in an easy to understand environment will revolutionize data sharing and communication.

The Red-R Framework is not only analysis software but a community of technical and non-technical users centred around data analysis and discovery. Users can search this community from within the Red-R program to access available packages, templates, help, and discussions. Red-R moves users from needing to memorize sets of functions to simply asking questions and querying the community of knowledge. Interaction with the community of researchers is one of the key components that sets Red-R apart from other data analysis programs which do not integrate analysis with the community.

Red-R is written in **Python** and can be configured on any platform that supports **R**, **Python**, and Qt [7]. Installation instructions are available on the Downloads page of the Red-R website. Red-R supports the Windows7 and Mac OS X operating systems, but users may be able to configure the system from source by downloading the source files via the repository at http://www.code.google.com/p/r-orange.

Figure 5: Websearch screen capture. The term View Data was entered into the Red-R search bar and the web search option was selected. A web page opens in the default web browser as shown here.

Acknowledgements

KRC would like to acknowledge support from U.S. Army Medical Research and Materiel Command under W81XWH-08-1-0222 (BC073237-3370).

Associated Files and Packages

- Red-R core - http://www.red-r.org/Downloads

Author Affiliation, Contribution, and Contact

- Kyle R Covington
  - Affiliation; Red-R.org, Translational Biology and Molecular Medicine Baylor College of Medicine. Contribution; Contributed to the document, core developer for Red-R. Contact; kyle@red-r.org, covingto@bcm.edu
- Anup Parikh
  - Affiliation; Red-R.org. Contribution; Wrote the document, core developer for Red-R. Contact; anup@red-r.org

References

[1] R Development Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria; 2010. ISBN 3-900051-07-0. Available from: http://www.R-project.org.

[2] Theußl S, Zeileis A. Collaborative software development using R-Forge. The R Journal. 2009;1(1):9–14.

[3] Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, et al. Bioconductor: open software development for computational biology and bioinformatics. Genome biology. 2004;5(10):R80.

[4] Fox J. The Comprehensive R Archive Network. Journal of Statistical Software. 2005;14(9):1–42.

[5] Fellows I, Fellows MI. Package Deducer. 2011;.

[6] Raymond ES. The art of unix programming. Pearson Education; 2003.

[7] Nokia Corporation. Qt Reference Documentation; 2010. Online. Available from: http://doc.qt.nokia.com/4.7/index.html.

Copyright Notice