

# xstatR: an Environment for Running R and XLISP-STAT in Docker Containers UseR! 2019

E. James Harner and Jun Tan

West Virginia University  
Rc<sup>2</sup>ai

July, 2019

# Outline

Introduction

xstatR Architecture

Data mapping

Main functions

xstatR

RXLisp

# Why XLisp-Stat?



# Integrating R and XLisp-Stat

Two dockerized environment have been created for integrating R and XLisp-Stat.

**xstatR** for embedding R into XLisp-Stat using the R dynamic library

**RXLisp** for embedding XLisp-Stat into R using the XLisp-Stat dynamic library

xstatR was originally presented by Jim Harner with programming support by Jun Tan in the 2009 **Directions in Statistical Computing (DSC) Workshop** held in Copenhagen.

**RXLisp** was developed by Duncan Temple Lang in 2002.

## xstatR: Linking via the R dynamic library

xstatR is designed for the user who wants the full XLisp-Stat environment with extensions for advanced dynamic graphics and a full R environment both accessible by a command-line interface with X11 support.

- Allows for a natural interface for software written in C
- Calls R directly
- Does not depend on other software, except for R
- Needs no setup or configuration
- Compile R with the `–enable-R-shlib` option
- Recompile xlipstat after upgrading R (if necessary)

# RXLisp: Linking via the XLisp-Stat dynamic library

RXLisp is designed for an R user who wants access to the dynamic graphics of XLisp-Stat using the syntax of R.

- Loads XLisp into R as an R package
- Implements a subset of XLisp-Stat, e.g., callbacks not currently supported
- Allows an R interface to XLisp-Stat (with an optional XLisp interface)
- Requires little knowledge of XLisp-Stat except the functions of interest

# What is Docker?

**Docker containers** provides a platform for running xstatR and RXLisp.

The two principal Docker entities are:

**Image:** an executable package that includes everything needed to run an application

**Container:** a runtime instance of an image

The image contains the code, configuration files, environmental variables, libraries, and the runtime. A container is an image with state, i.e., a user process.

The xstatR and RXLisp images provide complete, immutable solutions for embedding R in XLisp-Stat or embedding XLisp-Stat in R, respectively.





## C/R Interface

- The C/R interface is a thin wrapper of the embedded R API.
- The C/R interface avoids name conflicts between xlipstat and R
- R functions are mapped to C-based R interface macros/functions in xlipstat
- The C/R layer is invisible to users of xlipstat

## Low Level Lisp/R API

- Developed in C for communicating with the external C interface in R
- Provides basic functions for allowing the Lisp user to access R
- Requires the user to take care of details, e.g., data synchronization between Lisp and R
- Designed for developers and advanced users
- Used as a platform for xlipstat packages, e.g., xstatR, that require the flexible use of R

## High Level Lisp/R API

- Developed in Lisp
- Communicates with the low-level C macros/functions
- Provides convenient functions based on the embedded R environment
- Hides the embedded R environment from the user
- Is customizable and extendible, e.g., the user can customize the mapping between an R object and a Lisp object

# Data type mapping

- Data type mapping needs to be dealt with in bridging/interface software
- One-to-one mapping is the ideal situation
- Due to the rich data structures in Lisp and R and their flexibility, no such mapping exists

## A possible solution

- Convert simple data types, e.g., vectors of scalars.
- Returns a reference to R objects rather than converting complex data types
- Depends on the user to retrieve information from the R object references
- Used by R interface packages such as JRI and RServe

## Problems of this method

- R objects need to be locked to avoid garbage collection
- User is responsible for releasing R objects
- Proper access methods to the object references are needed

## Our strategy

- Use a generic structure to represent R objects
- Copy the data into this structure, ignoring unrecognized components
- Unlock the R object by default after all information is copied
- Assign R objects to variable names to retain them in memory, if persistence is needed

# The generic structure representing R objects

- A list of length one or three
- The first element is the data part
- The second and third elements, for R objects with attributes, are the attribute names and the attribute values
- Attribute name list is a list of strings
- Attribute value list provides the corresponding values of the named attributes
- Data part is either an array of scalars/strings, etc., or (recursively) a list of generic structures



## Current Implementation

- Many complicated structures are (currently) copied without special handling, e.g., `lm` objects.
- This structure is not convenient for direct use.
- The generic structure is a good foundation for building highly flexible conversions to meet users' needs, e.g., as has been implemented for data frames.

## Low Level C Functions

- `(callR "R statement")`  
Parse and evaluate an R statement. The evaluation result will be copied into a generic structure and returned to the user.
- `(saveToR "rName", lispObj)`  
Save the value of *lispObj* into the embedded R environment (the value has to be encoded into the generic structure).

# Examples

```
(setf y-list (callR "rnorm(50)"))  
(setf y (first y-list))  
(histogram y)  
(saveToR "y" y-list)
```

## High level functions: converting R objects

- A prototype, Rengine-proto, was built to ease the process of calling R.
- Rengine-proto directly supports the xstatR package, but can be used by any xlipstat package.
- The user can determine how to convert an R object based on its class name, dimension, etc.
- Rengine-proto currently includes loading data from R, saving an xstatR dataset to an R data frame, etc.

## List of major methods

- (send R :call *"statement"* &key *asis*)  
Parse and evaluate a R statement.  
*asis* is a boolean value indicating whether to bypass the conversion process or not
- (send R :save *"rName"* *lispObj* &key *attr attrNames*)  
Save a lisp object to R. Attributes and names can be attached
- (send R :save-dataset *"rName"* *lispObj* &key *cols rows*)  
Save a dataset or part of a dataset as data frame in R

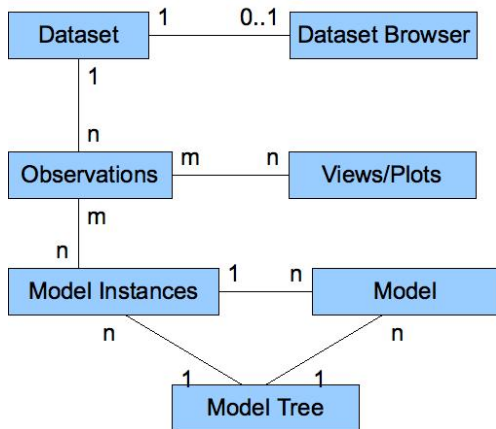
## Examples

```
(setf iris (send R :data "iris"))  
(send R :save-dataset "iris.data" iris  
:cols '(SEPAL.LENGTH SEPAL.WIDTH)  
:rows (iseq 1 10))  
(send R :call "ls()")  
(setf x (send R :call "iris.data"))
```

# The xstatR package

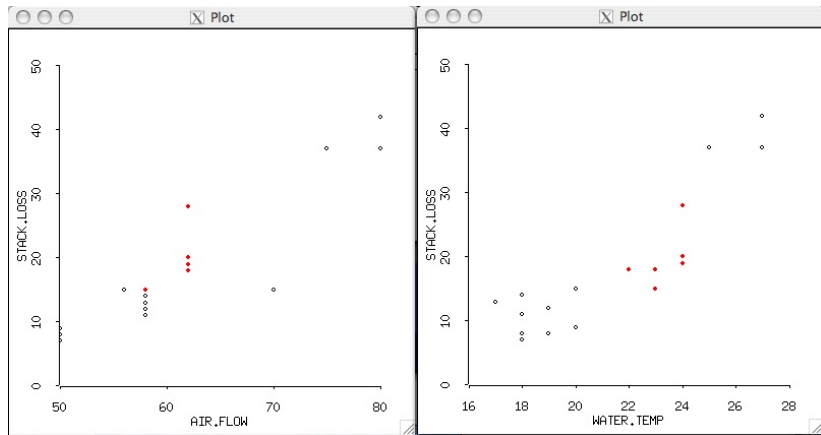
- A lisp package used for dynamic graphics, data modeling, and multivariate analysis
- Dataset objects are defined in terms of observation objects
- Virtual datasets are constructed to encapsulate derived variables from model or other statistical objects
- Observations are objects viewable in different graphs
- Point state, color, and symbol are properties of the observation
- Changes in an observation value or property is immediately updated in its linked views
- R acts as a computational engine using the Lisp/R bridge

## Partial Overview

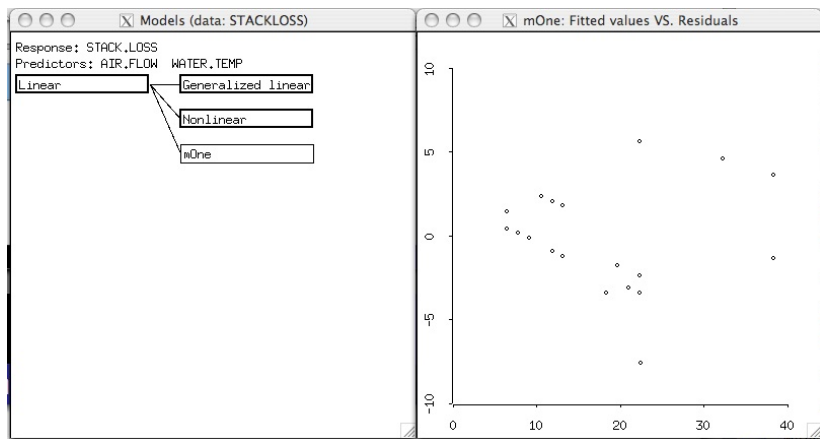




## Linked plots



# Modeling Tree



## More about virtual datasets for models

- Models generate derived variables from the original model variables.
- The newly generated data often need to be combined with the original dataset, e.g., to plot residuals against a predictor.
- SAS/JMP add (optionally) derived variables from models to the same dataset and the user soon loses track of which derived variables go with which model.
- R generates (optionally) named model and summary objects and the user soon loses track of the underlying models/summaries.
- Virtual datasets combined with tree-based model visualizations provide a solution.

## Currently working on...

- Virtual datasets combine the “variables” in a dataset with the “derived variables” from a model object
- The dataset browser shows the virtual dataset by clicking on the statistical object view, e.g., a model view
- Virtual datasets support linking, e.g., for model diagnostic plots
- Virtual model datasets combined with a model tree recursively allow model comparisons

# RXLisp

The RXLisp package provides an interface for calling XLisp-Stat functions from within R.