Ruby Plotting with Galaaz

An example of tightly coupling Ruby and R in GraalVM

Rodrigo Botafogo

16 October 2018

Contents

1 Introduction **1**
1.1 What does Galaaz mean 2

2 Galaaz Demo **2**
2.1 Prerequisites 2
2.2 Preparation 3
2.3 Running the demo 3
2.4 Running other demos 3

3 The demo code **3**

4 An extension to the example **5**

5 Conclusion **8**

1 Introduction

Galaaz is a system for tightly coupling Ruby and R. Ruby is a powerful language, with a large community, a very large set of libraries and great for web development. However, it lacks libraries for data science, statistics, scientific plotting and machine learning. On the other hand, R is considered one of the most powerful languages for solving all of the above problems. Maybe the strongest competitor to R is Python with libraries such as NumPy, Panda, SciPy, SciKit-Learn and a couple more.

With Galaaz we do not intend to re-implement any of the scientific libraries in R, we allow for very tight coupling between the two languages to the point that the Ruby developer does not need to know that there is an R engine running. For this to happen we use new technologies provided by Oracle: GraalVM, TruffleRuby and FastR:

```
GraalVM is a universal virtual machine for running applications
written in JavaScript, Python 3, Ruby, R, JVM-based languages like Java,
Scala, Kotlin, and LLVM-based languages such as C and C++.

GraalVM removes the isolation between programming languages and enables
interoperability in a shared runtime. It can run either standalone or in
the context of OpenJDK, Node.js, Oracle Database, or MySQL.

GraalVM allows you to write polyglot applications with a seamless way to
pass values from one language to another. With GraalVM there is no copying
or marshaling necessary as it is with other polyglot systems. This lets
you achieve high performance when language boundaries are crossed. Most
of the time there is no additional cost for crossing a language boundary
```

```
at all.

Often developers have to make uncomfortable compromises that require them
to rewrite their software in other languages. For example:

 * "That library is not available in my language. I need to rewrite it."
 * "That language would be the perfect fit for my problem, but we cannot
   run it in our environment."
 * "That problem is already solved in my language, but the language is
   too slow."

With GraalVM we aim to allow developers to freely choose the right language
for the task at hand without making compromises.
```

Interested readers should also check out the following sites:

- GraalVM Home
- TruffleRuby
- FastR
- Faster R with FastR

1.1 What does Galaaz mean

Galaaz is the Portuguese name for "Galahad". From Wikipedia:

```
Sir Galahad (sometimes referred to as Galeas or Galath),
in Arthurian legend, is a knight of King Arthur's Round Table and one
of the three achievers of the Holy Grail. He is the illegitimate son
of Sir Lancelot and Elaine of Corbenic, and is renowned for his
gallantry and purity as the most perfect of all knights. Emerging quite
late in the medieval Arthurian tradition, Sir Galahad first appears in the
Lancelot-Grail cycle, and his story is taken up in later works such as
the Post-Vulgate Cycle and Sir Thomas Malory's Le Morte d'Arthur.
His name should not be mistaken with Galehaut, a different knight from
Arthurian legend.
```

2 Galaaz Demo

2.1 Prerequisites

- GraalVM (>= rc7)
- TruffleRuby
- FastR

The following R packages will be automatically installed when necessary, but could be installed prior to the demo if desired:

- ggplot2
- gridExtra

Installation of R packages requires a development environment. In Linux, the gnu compiler and tools should be enough. I am not sure what is needed on the Mac.

In order to run the 'specs' the following Ruby package is necessary:

- gem install rspec

2.2 Preparation

- gem install galaaz

2.3 Running the demo

The ggplot for this demos was extracted from: http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-html.

On the console do

```
> galaaz master_list:scatter_plot
```

2.4 Running other demos

Doing on the console

```
> galaaz -T
```

will show a list with all available demos. To run any of the demos in the list, substitute the call to 'rake' to 'galaaz'. For instance, one of the examples in the list is 'rake sthda:bar'. In order to run this example just do 'galaaz sthda:bar'. Doing 'galaaz sthda:all' will run all demos in the sthda cathegory. Some of the examples require 'rspec' do be available. To install 'rspec' just do 'gem install rspec'.

3 The demo code

The following is the Ruby code and plot for the above example. There is a small difference between the code in the example and the code bellow. If the example is ran, the plot will appear on the screen, bellow, we generate an 'svg' image and then include it in this document. In order to generate and image, the R.svg device is used. To generate the plot on the screen, use the R.awt device, as commented on the code.

```
require 'galaaz'
require 'ggplot'

# load package and data
R.options(scipen: 999)  # turn-off scientific notation like 1e+48
R.theme_set(R.theme_bw)  # pre-set the bw theme.

midwest = ~:midwest
# midwest <- read.csv("http://goo.gl/G1K41K")  # bkup data source

# R.awt  # run the awt device if the plot should show on the screen
R.svg    # run the svg device if an image should be generated

# Scatterplot
```

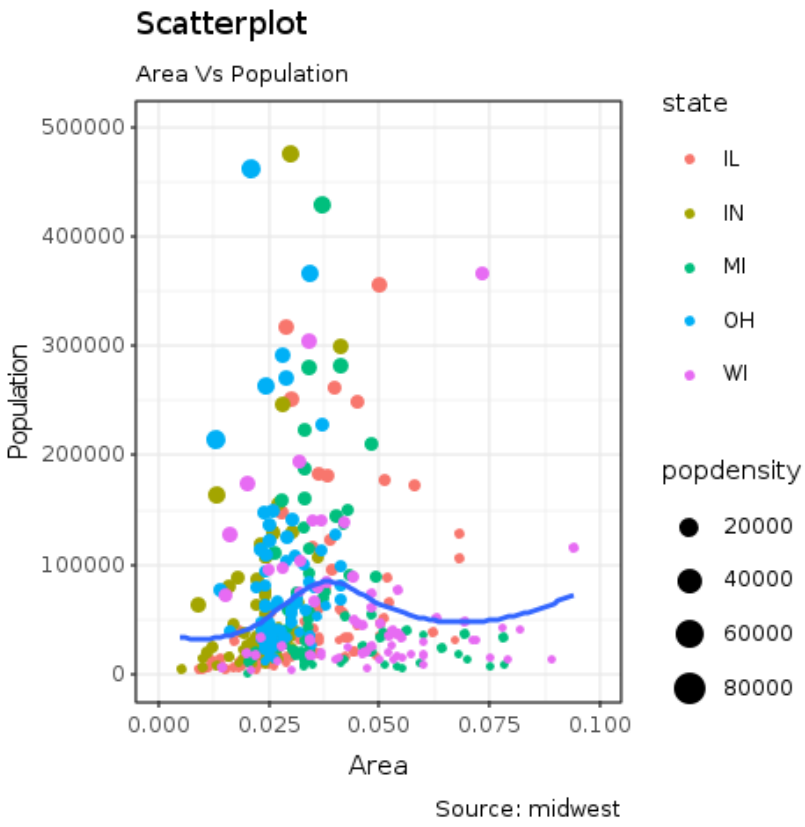

Figure 1: Midwest Plot

```
gg = midwest.ggplot(E.aes(x: :area, y: :poptotal)) +
     R.geom_point(E.aes(col: :state, size: :popdensity)) +
     R.geom_smooth(method: "loess", se: false) +
     R.xlim(R.c(0, 0.1)) +
     R.ylim(R.c(0, 500000)) +
     R.labs(subtitle: "Area Vs Population",
            y: "Population",
            x: "Area",
            title: "Scatterplot",
            caption: "Source: midwest")

R.png('midwest.png')      # this line is not necessary with the awt device
puts gg

R.dev__off                # R.dev__off turns off the device.  If using awt, the plot
                          # window will be closed
```

In R, the code to generate this plot is the following

```
# install.packages("ggplot2")
# load package and data
options(scipen=999)  # turn-off scientific notation like 1e+48
library(ggplot2)
theme_set(theme_bw())  # pre-set the bw theme.
data("midwest", package = "ggplot2")
```

```
# midwest <- read.csv("http://goo.gl/G1K41K")  # bkup data source

# Scatterplot
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
      geom_point(aes(col=state, size=popdensity)) +
      geom_smooth(method="loess", se=F) +
      xlim(c(0, 0.1)) +
      ylim(c(0, 500000)) +
      labs(subtitle="Area Vs Population",
           y="Population",
           x="Area",
           title="Scatterplot",
           caption = "Source: midwest")

plot(gg)
```

Note that both codes are very similar. The Ruby code requires the use of "R." before calling any functions, for instance R function 'geom_point' becomes 'R.geom_point' in Ruby. R named parameters such as (col = state, size = popdensity), become in Ruby (col: :state, size: :popdensity).

One last point that needs to be observed is the call to the 'aes' function. In Ruby instead of doing 'R.aes', we use 'E.aes'. The explanation of why E.aes is needed is an advanced topic in R and depends on what is know as Non-standard Evaluation (NSE) in R. In short, function 'aes' is lazily evaluated in R, i.e., in R when calling geom_point(aes(col=state, size=popdensity)), function geom_point receives as argument something similar to a string containing 'aes(col=state, size=popdensity)', and the aes function will be evaluated inside the geom_point function. In Ruby, there is no Lazy evaluation and doing R.aes would try to evaluate aes immediately. In order to delay the evaluation of function aes we need to use E.aes. The interested reader on NSE in R is directed to http://adv-r.had.co.nz/Computing-on-the-language.html.

4 An extension to the example

If both codes are so similar, then why would one use Ruby instead of R and what good is galaaz after all?

Ruby is a modern OO language with numerous very useful constructs such as classes, modules, blocks, procs, etc. The example above focus on the coupling of both languages, and does not show the use of other Ruby constructs. In the following example, we will show a more complex example using other Ruby constructs. This is certainly not a very well written and robust Ruby code, but it give the idea of how Ruby and R are strongly coupled.

Let's imagine that we work in a corporation that has its plot themes. So, it has defined a 'CorpTheme' module. Plots in this corporation should not have grids, numbers in labels should not use scientific notation and the preferred color is blue.

```
# corp_theme.rb
# defines the corporate theme for all plots

module CorpTheme

  #-------------------------------------------------------------------------------
```

```
  # Defines the plot theme (visualization).  In this theme we remove major and minor
  # grids, borders and background.  We also turn-off scientific notation.
  #-------------------------------------------------------------------------------

  def self.global_theme

    R.options(scipen: 999)  # turn-off scientific notation like 1e+48

    # remove major grids
    global_theme = R.theme(panel__grid__major: E.element_blank())
    # remove minor grids
    global_theme = global_theme + R.theme(panel__grid__minor: E.element_blank)
    # remove border
    global_theme = global_theme + R.theme(panel__border: E.element_blank)
    # remove background
    global_theme = global_theme + R.theme(panel__background: E.element_blank)
    # Change axis font
    global_theme = global_theme +
                   R.theme(axis__text: E.element_text(size: 8, color: "#000080"))
    # change color of axis titles
    global_theme = global_theme +
                   R.theme(axis__title: E.element_text(
                             color: "#000080",
                             face: "bold",
                             size: 8,
                             hjust: 1))
  end

end
```

We now define a ScatterPlot class:

```
# ScatterPlot.rb
# creates a scatter plot and allow some configuration

class ScatterPlot

  attr_accessor :title
  attr_accessor :subtitle
  attr_accessor :caption
  attr_accessor :x_label
  attr_accessor :y_label

  #-------------------------------------------------------------------------------
  # Initialize the plot with the data and the x and y variables
  #-------------------------------------------------------------------------------

  def initialize(data, x:, y:)
    @data = data
    @x = x
    @y = y
  end
```

```
#-------------------------------------------------------------------------------
# Define groupings by color and size
#-------------------------------------------------------------------------------

def group_by(color:, size:)
  @color_by = color
  @size_by = size
end

#-------------------------------------------------------------------------------
# Add a smoothing line, and if confidence is true the add a confidence interval, if
# false does not add the confidence interval
#-------------------------------------------------------------------------------

def add_smoothing_line(method:, confidence: true)
  @method = method
  @confidence = confidence
end

#-------------------------------------------------------------------------------
# Creates the graph title, properly formated for this theme
# @param title [String] The title to add to the graph
# @return textGrob that can be included in a graph
#-------------------------------------------------------------------------------

def graph_params(title: "", subtitle: "", caption: "", x_label: "", y_label: "")
  R.labs(
    title: title,
    subtitle: subtitle,
    caption: caption,
    y_label: y_label,
    x_label: x_label,
  )
end

#-------------------------------------------------------------------------------
# Prepare the plot's points
#-------------------------------------------------------------------------------

def points
  params = {}
  params[:col] = @color_by if @color_by
  params[:size] = @size_by if @size_by
  R.geom_point(E.aes(params))
end

#-------------------------------------------------------------------------------
# Plots the scatterplot
#-------------------------------------------------------------------------------
```

```
  def plot(device = 'awt')
    device == 'awt' ? R.awt : R.svg

    gg = @data.ggplot(E.aes(x: @x, y: @y)) +
      points +
      R.geom_smooth(method: @method, se: @confidence) +
      R.xlim(R.c(0, 0.1)) +
      R.ylim(R.c(0, 500000)) +
      graph_params(title: @title,
                   subtitle: @subtitle,
                   y_label: @y_label,
                   x_label: @x_label,
                   caption: @caption) +
      CorpTheme.global_theme

    R.png('scatter_plot.png') if !(device == 'awt')
    puts gg
    R.dev__off

  end

end
```

And this is the final code for making the scatter plot with the midwest data

```
require 'galaaz'
require 'ggplot'

sp = ScatterPlot.new(~:midwest, x: :area, y: :poptotal)
sp.title = "Midwest Dataset - Scatterplot"
sp.subtitle = "Area Vs Population"
sp.caption = "Source: midwest"
sp.x_label = "Area"
sp.y_label = "Population"
sp.group_by(color: :state, size: :popdensity)    # try sp.group_by(color: :state)
# available methods: "lm", "glm", "loess", "gam"
sp.add_smoothing_line(method: "glm")
sp.plot('svg')

# require input from the user so that the script does not end removing the plot from
# the screen
```

5 Conclusion

R is a very powerful language for statistical analysis, data analytics, machine learning, plotting and many other scientific applications with a very large package ecosystem. However R is often considered hard to learn and lacking modern computer languages constructs such as object oriented classes, modules, lambdas, etc. For this reason, many developers have started or switched from R to Python.

With Galaaz, R programmers can almost transparently migrate from R to Ruby, since syntax is

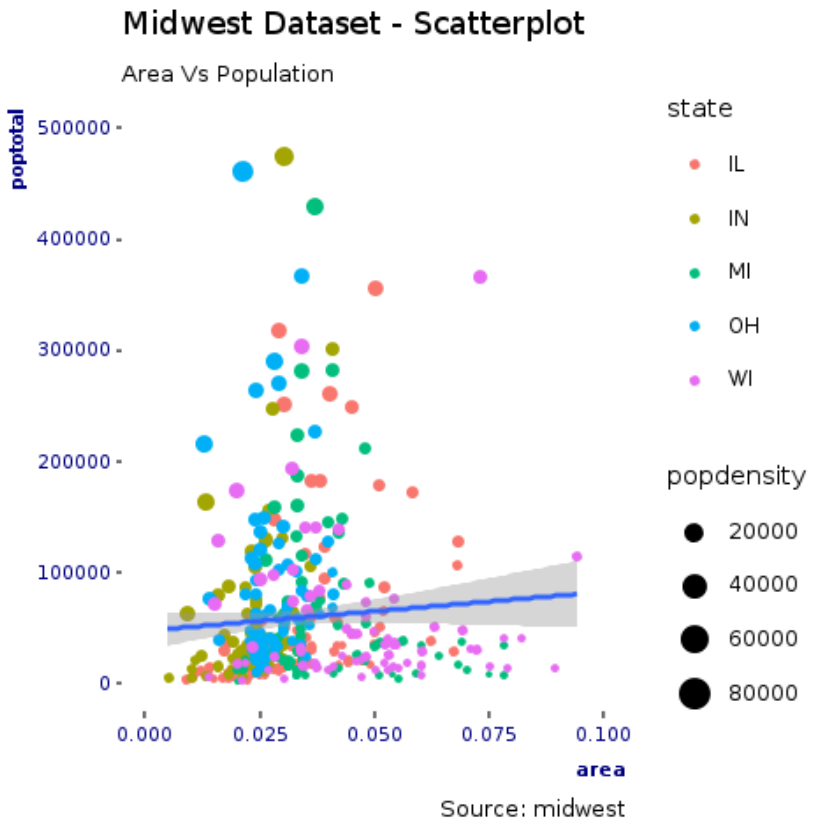

Figure 2: Midwest Plot with 'glm' function and modified theme

almost identical and they have fastR as the R engine. FastR, by most benchmarks, can be orders of magnitude faster than Gnu R. Further, by using Galaaz the R developer can start (slowly if needed) using all of Ruby's constructs and libraries that nicely complement R packages.

For the Ruby developer, Galaaz allows the immediate use of R functions completely transparently. As shown in the second example above, class ScatterPlot completely hides all the details an R calls from the Ruby developer, furthermore Galaaz is powered by TruffleRuby that can also be orders of magnitude faster than MRI Ruby.