Graficación en Aplicaciones Web Usando Sinatra con R

Rebeca Rodríguez Huesca[1], Salomón Rincón Torres[2], Argelia Berenice Urbina Nájera[1]
[1]*Universidad Politécnica de Puebla,* [2]*Universidad Popular Autónoma del Estado de Puebla*
rebeca.rodriguez@uppue.edu.mx, salomon.rincon@upaep.mx, argelia.urbina@uppue.edu.mx

Resumen

El objetivo de este documento es describir el proceso para poder generar gráficas de forma rápida y fácil, a partir de la información que se tenga disponible en una aplicación web, la cual se pudo haber obtenido desde un archivo de texto, una base de datos o haber solicitado al usuario que la introdujera directamente, entre otras. La aplicación principal se desarrolló en Ruby bajo el framework Sinatra, y posteriormente para la graficación se emplearon dos herramientas: R y ggplot2. Este artículo describe cómo lograr la interacción entre el sistema para computación estadística y graficación "R" y el framework para desarrollo de aplicaciones web "Sinatra" utilizando el servidor TCP/IP "Rserve". También muestra la forma de crear gráficas más complejas usando ggplot2, el cual es un paquete diferente al sistema de graficación estándar de "R".

1. Introduction

Actualmente la disponibilidad de la información y la cantidad que se genera de ésta es impresionante, por ello, es necesario que se ponga al alcance de todos para una mejor toma de decisiones. Sin embargo, si ésta no se organiza, sintetiza y presenta de manera adecuada puede no ser de ayuda para las personas que la requieren.

El uso de gráficos se remonta a la edad antigua, pero es a partir del siglo XVIII cuando surgió nuevamente esta herramienta. De acuerdo con Beniger y Robyin [1] esta forma de presentar resultados respondió a las necesidades de: i.) la organización dentro de un espacio, ii.) la comparación entre variables discretas y continuas, y iii.) la distribución de variables continuas y una comparación entre continuas y discretas. Actualmente, con la ayuda de la tecnología, los gráficos se han convertido en una herramienta que se emplea a diario para presentar la información de manera que sea útil para las personas que la requieren.

Utilizar gráficos para presentar resultados, hace posible que la información se pueda manipular fácilmente, mostrar información relevante, comparar resultados, entre otros.

Aunado a lo anterior, la oferta cada vez mayor de SaaS (*Software as a Service*) y el cómputo en la nube (*cloud computing*) están demandando que muchas aplicaciones de escritorio (*Desktop*) se migren a aplicaciones web, y que cuando se requiere de un nuevo sistema de información, casi siempre se considere como primera opción desarrollarlo como aplicación web.

2. Herramientas

Ruby es un lenguaje de programación que se ha hecho muy conocido, debido a que es uno de los lenguajes más utilizados en los repositorios de GitHub en estos momentos[1]. Es un lenguaje dinámico, orientado a objetos y de código abierto enfocado en la simplicidad.

Su creador, Yukihiro "Matz" Matsumoto, mezcló partes de sus lenguajes favoritos (Perl, Smalltalk, Eiffel, Ada, y Lisp) para formar un nuevo lenguaje que incorporara tanto la programación funcional como la programación imperativa. Además, posee diversos frameworks que lo convierten en un lenguaje potente para desarrollar aplicaciones web. Uno de estos frameworks toma el nombre del célebre artista Frank Sinatra.

Sinatra es un Framework y DSL (Domain Specific Language) para la creación rápida de aplicaciones web en Ruby con un mínimo esfuerzo. Fue diseñado y desarrollado por Blake Mizerany en Septiembre de 2007 en California; es Software Libre y se distribuye bajo licencia MIT (*Massachusetts Institute of Technology)*. Algunas de las compañías más destacadas que usan Sinatra son BBC, Engine Yard, Heroku, GitHub, y Songbird; siendo Heroku, la que provee la mayor parte del apoyo para su desarrollo.

R es un sistema para computación estadística y graficación, el cual está formado por un lenguaje de programación orientado a objetos, un ambiente de

[1] http://adambard.com/blog/top-github-languages-for-2013-so-far/

ejecución con capacidad para realizar gráficas, un depurador (debugger), acceso a ciertas funciones del sistema y la capacidad de ejecutar programas almacenados como scripts.

Es software libre y se distribuye bajo licencias GPL-2 (*General Public License*) y GPL-3; adicionalmente, puede decirse que es uno de los lenguajes de programación más utilizados en investigación estadística[2]. Su uso para el análisis de datos tanto en la academia como dentro del ambiente corporativo va en aumento debido a que la minería de datos (data mining) está en pleno apogeo. Fue escrito inicialmente por Ross Ihaka y Robert Gentleman en el departamento de estadística de la Universidad de Auckland, en Auckland, Nueva Zelanda. Adicionalmente, muchas personas han contribuido al desarrollo de R enviando código y reportes de errores.

El paquete base constituye el núcleo de R y contiene las funciones básicas para leer y manipular datos, algunas funciones gráficas y algunas funciones estadísticas [2]. Muchas de las técnicas estadísticas implementadas se encuentran en el entorno base de R y otras se acompañan en forma de bibliotecas (packages).

Los gráficos pueden verse de manera y guardarse en varios formatos (jpg, png, bmp, ps, pdf, emf, pictex, xfig; los formatos disponibles dependen del sistema operativo).

Existen diferentes opciones para la integración de R y Ruby: RinRuby, rsruby, rApache, y RServe. Las únicas dos opciones que son 100% Ruby son RinRuby y RServe, ya que rsruby es una extensión de C para Ruby, enlazada a la biblioteca compartida de R (R's shared library) y rApache tiene el inconveniente de que sólo funciona con el servidor web Apache.

Se seleccionó RServe debido a que es de 5 a 10 veces más rápido que RinRuby [3]. RServe es un servidor TCP/IP desarrollado por Simon Urbanek, que permite a otros programas utilizar servicios de R desde varios lenguajes sin necesidad de inicializar R o enlazar la biblioteca compartida R. Se han desarrollado implementaciones del lado del cliente (client-side) para lenguajes como C/C++, PHP y Java. Existe un cliente para Ruby (en forma de gema) desarrollado por Claudio Bustos llamado rserve-client, el cual es Software Libre y se distribuye bajo licencia MIT.

Típicamente RServe se usa para integrar R en el backend para modelos estadísticos o gráficas desde otras aplicaciones.

ggplot2 además de ser Software Libre, se distribuye bajo licencia GPL-2; fue desarrollado por Hadley Wickham, profesor asistente de estadística en la Universidad de Rice en Houston, Texas y Doctor en estadística por la Universidad Estatal de Iowa.

Wilkinson [4] creó la gramática de gráficos para describir las características profundas que subyacen en todos los gráficos estadísticos. El paquete ggplot2 es una implementación de las ideas plasmadas en el libro, *The Grammar of Graphics*, escrito por Leland Wilkinson, cuyo objetivo era establecer una serie de principios generales para la unificación de la visualización de datos. Razón por la cual ggplot2 es más poderoso al no limitarse a un conjunto de gráficos predefinidos, sino que es posible crear nuevos gráficos que se adapten a nuestras necesidades, además de ofrecer un enfoque más elegante y natural que lo hace un paquete gráfico básico de R.

Está diseñado para trabajar en capas, comenzando con una capa que muestra los datos en bruto y posteriormente se van agregando capas de anotaciones y resúmenes estadísticos.

Existen varios paquetes de graficación disponibles para R además de ggplot2, [5], dentro de las cuales se encuentran:

- Gráficas base
- Gráficas grid
- Gráficas trellis/lattice

3. Métodos

Se explica inicialmente el proceso a seguir para realizar el gráfico y posteriormente las posibilidades en cuanto a tipos de gráficos que se tienen.

3.1 Proceso general

Para los ejemplos presentados se utiliza la siguiente estructura de directorios:

En el directorio raíz se tiene el programa principal en Ruby llamado graficas_R.rb.

El directorio views contiene todas las vistas, la vista llamada menu.erb corresponde a la página principal desde la cual se llama a las vistas correspondientes a cada uno de los ejemplos.

El proceso general consta de 4 pasos:

1. Crear una conexión con R utilizando RServe
2. Pasar la información a R
3. Generar la gráfica
4. Cerrar la conexión

El sistema de graficación estándar de R permite crear además gráficas de pie, gráficas de barras apiladas y otros, pero hay ocasiones en las que se

[2] http://es.wikipedia.org/wiki/R_%28lenguaje_de_programaci%C3%B3n%29

requiere generar un tipo de gráficas más complejas, por ejemplo agrupando la información en diferentes niveles. En este caso es necesario emplear paquetes desarrollados por terceros, diseñados como una alternativa al sistema antes descrito; ggplot2 es uno de ellos.

3.2 Tipos de gráficos

Para seleccionar el tipo de gráfico que se va a realizar se deben considerar algunos aspectos:

- Finalidad (ver figura 1)
- Relación entre variables
- Composición de los datos
- Comparaciones
- La distribución de los datos
- Tipo y cantidad de variables que se va a graficar.

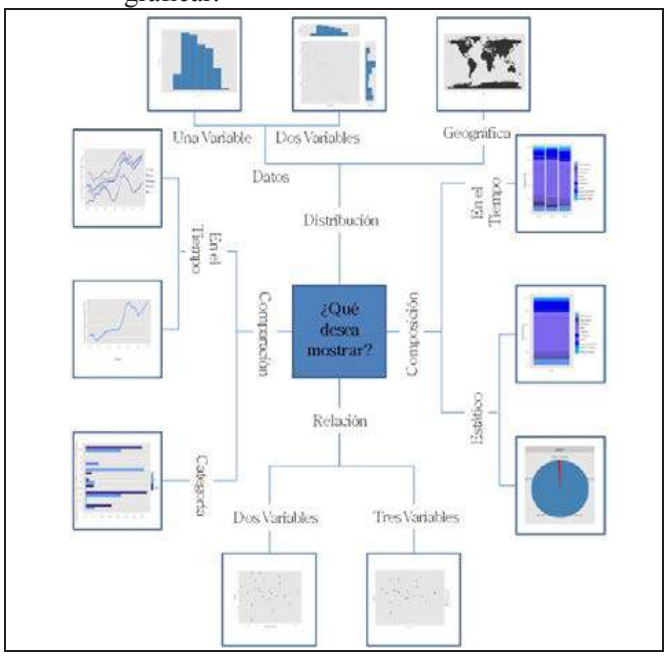

Figura 1. Tipos de gráficos para variables categóricas y numéricas [6]

4. Implementación y resultados

El contenido del programa principal **graficas_R.rb** puede observarse en la figura 2, donde se hace el llamado al menú principal y a las gráficas que se presentan.

En la figura 3 se muestra el contenido del archivo menu.erb, el cual muestra el menú principal.

4.1 Generación de gráficas con R.

A continuación se explica la forma de generar gráficas de barras y líneas utilizando el sistema R.

```
#encoding: utf-8
require 'sinatra'
require 'rserve'

get '/' do
  erb :menu
end

get '/barras1' do
  erb :grafica_barras1
end

get '/puntos_lineas' do
  erb :grafica_pl
end

get '/bapiladas/ggplot2' do
  erb :grafica_barrasapiladas4
end
```

Figura 2. Programa principal

```
<h1>Menú Principal</h1>
<p><a href="/barras1">Gráfica de barras</a></p>
<p><a href="/puntos_lineas">Gráfica de puntos y lineas</a></p>
<p><a href="/bapiladas/ggplot2">Gráfica de barras apiladas con ggplot2</a></p>
```

Figura 3. Menú principal

4.1.1 Gráfica de barras

Se utilizará la gráfica de barras para ir explicando cada uno de los pasos del proceso, En la figura 4 se muestra el contenido del archivo grafica_barras1.erb.

```
<h1>Gráfica de barras</h1>
<%
 datos_x = [1,3,6,4,9]

 c = Rserve::Connection.new
 if c.connected?
   c.assign("x", datos_x)
   c.assign("ruta","/public/images/grafica_barras1.png")
   c.void_eval <<-EOF
     # Start PNG device driver to save output to figure.png
     png(file=ruta, height=600, width=800)
     barplot(x)
     # Turn off device driver (to flush output to png)
     dev.off()
   EOF

 else %>
  <p>NO conectado</p>
<%
 end

 c.close if c.connected?
%>
<img src="../images/grafica_barras1.png" alt="grafica"/>
```

Paso 1 — Paso 2 — Paso 3 — Paso 4

Figura 4. Código de archivo grafica_barras1.erb

Paso 1. Crear una conexión con R utilizando Rserve. Esto es similar a abrir una sesión en R a la cual en este caso se hará referencia mediante la letra "c".

Paso 2. Pasar la información a R. Lo primero que se hace es crear un vector en R al que en este caso se llamó x, y se le asigna el contenido del arreglo datos_x. Esto equivale a que desde una sesión de R se hubiera tecleado x <- c(1,3,6,4,9).

Paso 3. Generar la gráfica. Para esto se debe:

- Crear una variable llamada ruta que contiene la cadena de caracteres "/public/images/ grafica_barras1.png".
- Después, se indica que la gráfica que se genera, se va a guardar como archivo PNG, el nombre del archivo incluyendo la ruta está indicado en la

variable ruta; asimismo que la gráfica será de 600 pixels de alto por 800 de ancho.

- Con la línea barplot(x) se está indicando a R que genere una gráfica de barras utilizando la información contenida en el vector llamado x.
- Una vez generada la gráfica, mediante la instrucción dev_off() se indica que la guarde en el archivo PNG especificado anteriormente.

Paso 4. Cerrar la conexión. Finalmente, se cierra la conexión, si es que se logró establecer en el paso 1.

La línea toma el archivo PNG que generó R y lo muestra en el navegador web, como se observa en la figura 5.

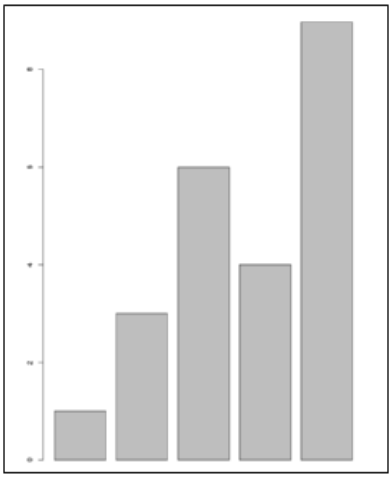

Figura 5. Gráfica de barras

4.1.2 Grafica de líneas

Se pueden realizar gráficas más vistosas agregando colores, entre otros elementos. En la figura 6 se muestra el código para generar una gráfica de líneas, la cual permite graficar valores correspondientes a 2 vectores y la figura 7 muestra la gráfica de líneas resultante.

4.2 Gráficas más complejas

Como se mencionó anteriormente, el sistema de graficación estándar de R no permite generar gráficas más complejas, donde se requiera presentar diferentes datos y gráficos en un mismo plano cartesiano.
Suponiendo que se tiene la información mostrada en la tabla 1, y es necesario realizar una gráfica de barras apiladas, donde además se tienen 2 series de valores que comparten información por categoría, esto no se podría lograr con el sistema R. La solución que se presenta en este artículo es utilizar el paquete ggplot2, donde si es posible realizarlo.

```
<h1>Gráfica de puntos y lineas</h1>
<%
 arreglo1 = [1, 3, 6, 4, 9]
 arreglo2 = [2, 5, 4, 5, 12]
 c = Rserve::Connection.new
 if c.connected?
   c.assign("autos", arreglo1)
   c.assign("camiones", arreglo2)
   c.assign("ruta","/public/images/grafica_pl.png")
   c.void_eval <<-EOF
     # Start PNG device driver to save output to figure.png
     png(file=ruta, height=800, width=600)
     # Calculate range from 0 to max value of autos and camiones
     g_range <- range(0, autos, camiones)
     # Turn off axes and annotations (axis labels) so we can
     # specify them ourself
     plot(autos, type="o", col = "blue", ylim=g_range, axes=FALSE, ann=FALSE)
     lines(camiones, type = "l", col = "red")
     # Create a title with a red, bold/italic font
     title(main="Autos", col.main="red", font.main=4)
     # Label the x and y axes with dark green text
     title(xlab="Dias", col.lab=rgb(0,0.5,0))
     title(ylab="Total", col.lab=rgb(0,0.5,0))
     # Make x axis using Mon-Fri labels
     axis(1, at=1:5, lab=c("Lunes","Martes","MiÃ©rcoles","Jueves","Viernes"))
     # Make y axis with horizontal labels that display ticks at
     # every 4 marks. 4*0:g_range[2] is equivalent to c(0,4,8,12).
     axis(2, las=1, at=4*0:g_range[2])
     # Create box around plot
     box()
     # Create a legend at (1, g_range[2]) that is slightly smaller
     # (cex) and uses the same line colors and points used by
     # the actual plots
     legend(1, g_range[2], c("autos","camiones"), cex=0.8, col=c("blue","red"), pch=21:22, lty=1:2)
     # Turn off device driver (to flush output to png)
     dev.off()
   EOF
 else %>
  <p>NO conectado</p>
<%
 end
   c.close if c.connected?
%>
<img src="../images/grafica_pl.png" alt="grafica"/>
```

Figura 6. Código de archivo grafica_pl.erb

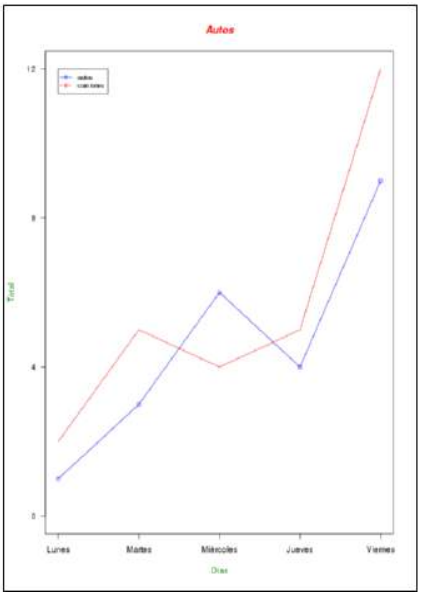

Figura 7. Gráfica de líneas

Tabla 1. Valores que se desean graficar

Periodo		Formación	En Consolidación	Consolidado
Ene-Abr 2009	Meta	40	30	30
	Realizado	35	45	20
May-Ago 2009	Meta	35	35	30
	Realizado	34	45	20
Sep-Dic 2009	Meta	30	30	40
	Realizado	20	40	20

En la figura 8 se muestra el código del archivo grafica_barrasapiladas4.erb, el cual permite generar una gráfica con ggplot2, donde se siguen también los 4 pasos del proceso antes mencionado, y en la figura 9 muestra la gráfica de barras apiladas resultante.

```
<h1>Gráfica de barras apiladas con ggplot2</h1>
<%
 periodos = ["Ene-Abr 2009", "May-Ago 2009", "Sep-Dic 2009"]
 categoria = ["Meta", "Realizado"]
 estatus = ["Formación", "En Consolidación", "Consolidado"]
 formacion = [40, 35, 35, 34, 30, 20]
 consolidacion = [30, 45, 35, 45, 30, 40]
 consolidado = [30, 20, 30, 20, 40, 20]

 c = Rserve::Connection.new
 if c.connected?
 c.assign("periodos", periodos)
    c.assign("categoria", categoria)
    c.assign("estatus", estatus)
    c.assign("formacion", formacion)
    c.assign("consolidacion", consolidacion)
    c.assign("consolidado", consolidado)
    c.assign("ruta","/public/images/grafica_barrasapiladas4.png")
    c.eval("library(reshape2)")
    c.eval("library(ggplot2)")
    c.void_eval <<-EOF
      df = structure(list(Periodo = structure(c(1L, 1L, 2L, 2L, 3L, 3L), .Label = periodos, class = "factor"),
                          Grupo = structure(c(1L, 2L, 1L, 2L, 1L, 2L), .Label = categoria, class = "factor"),
                          Formacion = formacion, En.consolidacion = consolidacion,
                          Consolidado = consolidado), .Names = c("Periodo", "Grupo", "Formación",
                                                                 "En consolidación", "Consolidado"),
                          class = "data.frame", row.names = c(NA, -6L))

      m=melt(df)
      ggplot(m,aes(x=factor(Grupo),y=value,fill=factor(variable))) + geom_bar(position="fill", stat="identity") +
                      scale_y_continuous(formatter = "percent",
                      breaks=c(0.2, 0.4, 0.6, 0.8, 1)) + facet_wrap(~ Periodo, ncol = 3)
      ggsave(file=ruta)
      # Turn off device driver (to flush output to png)
      dev.off()
    EOF
  else %>
  <p>NO conectado</p>
<%
  end

  c.close if c.connected?
%>
<img src="../images/grafica_barrasapiladas4.png" alt="grafica" height=600 width=800/>
```

Figura 8. Código de archivo grafica_barrasapiladas4.erb

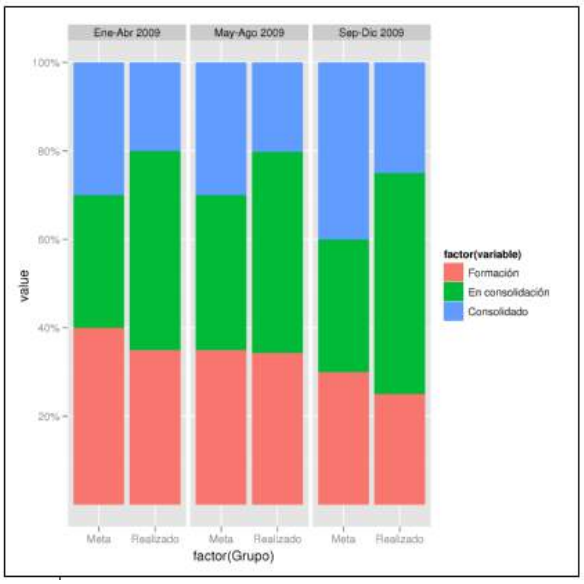

Figura 9. Gráfica de barras apiladas con ggplot2

4.3 Personalización del gráfico en ggplot2

La estructura básica de la función ggplot implica típicamente 4 argumentos como se puede observar en la siguiente línea [6]:

ggplot(data,aes(x)) + geom *(aes(y=\ ")) donde:

- ✓ data: corresponde al dataframe que se va a utilizar para el gráfico.
- ✓ x: Es el nombre de la columna (del dataframe especificado en el paso anterior) que contiene los datos que se van a graficar en el eje horizontal.
- ✓ y: entre las comillas se escribe el nombre de la columna que contiene los datos que se van a graficar en el eje horizontal (este argumento se emplea sólo en caso de ser necesario).
- ✓ *: en lugar del asterisco se determina el tipo de gráfico que se quiere realizar. Algunos tipos de gráficos son: histogramas, de densidad, líneas, barras, entre otros.

5. Conclusiones

La visualización de información cada vez más se ha convertido en una necesidad para entender las grandes cantidades de información disponibles. En este documento se presentaron sólo algunas aplicaciones sencillas que pueden crearse de manera rápida empleando el sistema de graficación R y ggplot2; sin embargo, es importante mencionar que ambos paquetes brindan muchas opciones para realizar gráficos bastante complejos.

6. Referencias

[1] J. R. Beniger, D. L. Robyn, "Quantitative graphics in statistics: A brief history", *The American Statistician*, Vol. 32, No. 1, Feb., 1978

[2] Andrés González y Silvia González (R Development Core Team), "Introducción a R", http://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf

[3] Chang Sau Sheong, "Ruby and R", http://www.slideshare.net/sausheong/rubyand-r, Abril 2011

[4] Leland Wilkinson, "The Grammar of graphics. Statistics and Computing", Springer, 2a edición, 2005.

[5] Hadley Wickham, "Ggplot2: Elegant Graphics for Data Analysis". Springer, Rice University, Department of Statistics. Houston, TX USA, Agosto 2009, ISBN: 978-0387981406

[6] Julio César Alonso, Alejandra González, "Ggplot: gráficos de alta calidad", Noviembre 2012, ISSN 1794-029X No. 33

[7] Emmanuel Paradis, "*R para Principiantes*", http://cran.r-project.org/doc/contrib/rdebuts_es.pdf

[8] Paul Teetor, "*R Cookbook*", O'Reilly Media, Marzo 2011, ISBN: 978-0-596-80915-7

[9] Ramon Saccilotto, "Tutorial: ggplot2", http://www.ceb-institute.org/bbs/wp-content/uploads/2011/09/handout_ggplot2.pdf

[10] Winston Chang, "R Graphics Cookbook", O'Reilly Vlg. Gmbh & Co., Enero 2013, ISBN: 978-0387981406