BUS464: VISUALIZATION IN R

Roger Bivand

25 April 2018

INTRODUCTION

- Today, we'll look at how graphics work, touching on the internals
- First, graphics devices and base graphics
- Next, grid graphics and their use in **lattice**, **ggplot2**, **vcd**, **tmap** and other packages
- Finally, interactive graphics both simple and using external software, especially JavaScript rendering in web browsers

JUST FOR FUN (?)

- Visualization is closely linked to aesthetic preferences
- Consider visiting Lukasz Piwek's site for plenty of ideas for visualization comparing approaches
- Consider looking at **pinp** and **tint** as well as **tufte** for creating deliverables (HTML for conversion to PDF by printing to a PDF device, or PDF)
- Consider for example youngmetro for formatting presentations
- I use the LaTeX Beamer `m` theme (now `metropolis`) with Fira fonts (which sometimes fall out)

Base graphics

BASE GRAPHICS

- In early R, all the graphics functionality was in **base**; **graphics** was split out of **base** in 1.9.0 (2004-04-12), and **grDevices** in 2.0.0 (2004-10-04)
- When R starts now, the **graphics** and **grDevices** packages are loaded and attached, ready for use
- **graphics** provides the user-level graphical functions and methods, especially the most used `plot()` methods that many other packages extend
- **grDevices** provides lower-level interfaces to graphics devices, some of which create files and others display windows

BASE GRAPHICS PACKAGES

The `search()` function shows the packages present in the search path, so here we run an instance of R through `system()` to check the startup status. In RStudio, one will also see `”tools:rstudio”` in the search path.

```
> cat(system(’echo ”search()” | R --no-save --vanilla’, intern=TRUE)[20:23], sep=”\n”)
```

```
## > search()
## [1] ”.GlobalEnv”        ”package:stats”     ”package:graphics”
## [4] ”package:grDevices” ”package:utils”     ”package:datasets”
## [7] ”package:methods”   ”Autoloads”         ”package:base”
```

GRAPHICS DEVICES

- The PDF (`pdf()`) and PostScript (`postscript()`) devices write commonly used vector files
- Display formats and raster/pixel file devices must rasterize vector graphics input
- Display formats vary by platform: `X11()` on X11 systems, `quartz()` on macOS, `windows()` on Windows and are available if compiled
- `png()`, `jpeg()`, `tiff()` are generally available, `svg()` and `cairo_pdf()` and `cairo_ps()` may also be built

GRAPHICS DEVICES

The `capabilities()` function shows what R itself can offer, including non-graphics capabilities, and we can also check the versions of external software used

```
> capabilities()
```

```
##        jpeg         png        tiff       tcltk
##        TRUE        TRUE        TRUE        TRUE
##         X11        aqua    http/ftp     sockets
##        TRUE       FALSE        TRUE        TRUE
##      libxml        fifo      cledit       iconv
##        TRUE        TRUE       FALSE        TRUE
##         NLS     profmem       cairo         ICU
##        TRUE       FALSE        TRUE        TRUE
## long.double     libcurl
##        TRUE        TRUE
```

```
> grSoftVersion()
```

```
##                    cairo                   libpng
##                ”1.15.10”                 ”1.6.31”
##                     jpeg                  libtiff
##                    ”6.2” ”LIBTIFF, Version 4.0.9”
```

GRAPHICS DEVICES

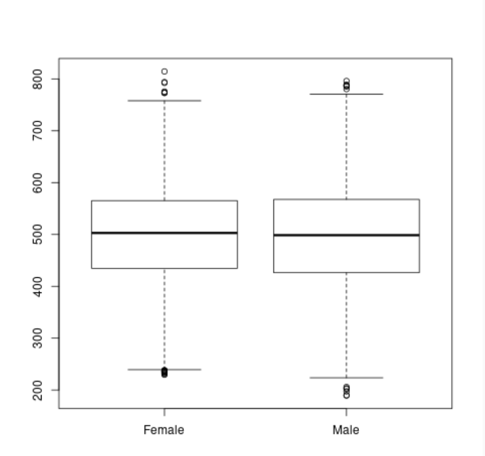
When there is no device active, the current device is the null device at position 1; one can open several and move between active devices. In RStudio, the devices used vary by context - if the console is used, RStudioGD will be used, backed by png; in a notebook, small embedded devices appear (apparently PNG inline images).In addition to providing a device, RStudio appears to work quite hard to embed fonts and crop white space on the edges of the graphics file

```
> a11 <- readRDS(”../mon/dicook/a11.rds”)
> png(”plot.png”)
> dev.cur()

## png
##   3

> boxplot(sci_mean ~ ST004D01T, a11)
> dev.off()

## pdf
##   2
```

GRAPHICS DEVICES

```
> png(tempfile())
> unlist(dev.capabilities())

##      semiTransparency transparentBackground
##                ”TRUE”                ”semi”
##           rasterImage               capture
##                 ”yes”               ”FALSE”
##               locator
##               ”FALSE”

> dev.size(”in”)

## [1] 6.666667 6.666667

> dev.interactive()

## [1] FALSE

> dev.off()

## pdf
##   2
```

Read in and display `plot.png`:



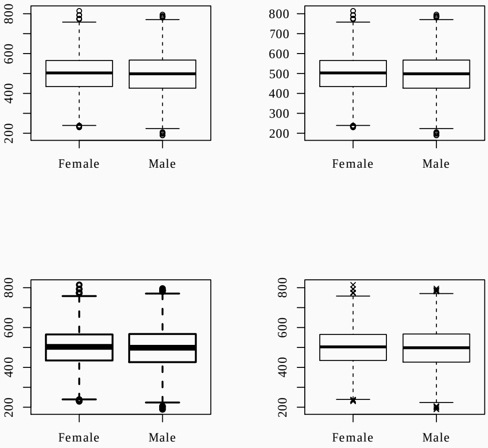
BASE GRAPHICS

- The **graphics** package provides the basic components for drawing on the devices, and user-level functions bundling the components
- A `boxplot()` will need to be placed in a plotting area, the boxes and whiskers drawn, axes and ticks drawn, and anotation added
- The graphical parameters are set and may be modified using the `par()` function; the `layout()` function can also be used to position multiple display elements.
- Inspecting `?par` shows how much can be modified within the limits of the base graphics system

BASE GRAPHICS

We can manipulate the number of plots `mfrow=`, axis label orientation `las=`, line width `lwd=`, point symbol `pch=` and so on

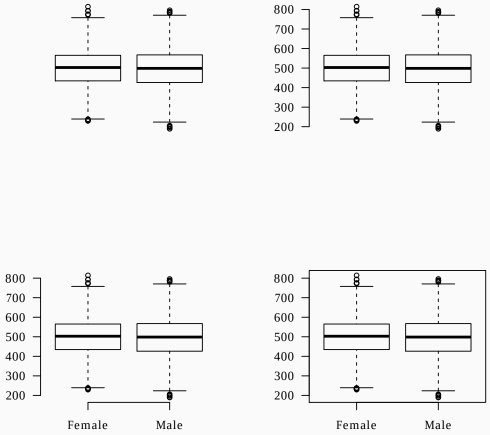
```
> opar <- par(mfrow=c(2,2))
> boxplot(sci_mean ~ ST004D01T, a11, las=0)
> boxplot(sci_mean ~ ST004D01T, a11, las=1)
> boxplot(sci_mean ~ ST004D01T, a11, lwd=2)
> boxplot(sci_mean ~ ST004D01T, a11, pch=4)
> par(opar)
```



BASE GRAPHICS

We can build up the plot bit-by-bit - here using the axes as an example

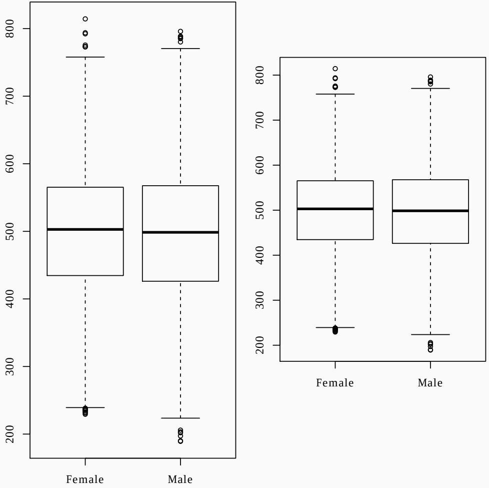
```
> opar <- par(mfrow=c(2,2))
> boxplot(sci_mean ~ ST004D01T, a11, axes=FALSE)
> axis(2, las=1)
> axis(1, at=1:2, labels=levels(a11$ST004D01T))
> box()
> par(opar)
```



BASE GRAPHICS

We can manipulate the margins of the plot region within the figure region; `layout()` helps us align the output

```
> layout(cbind(c(1, 1), c(2,2)))
> opar <- par(mar=c(3, 3, 0, 0)+0.1)
> boxplot(sci_mean ~ ST004D01T, a11)
> par(mar=c(10, 3, 4, 0)+0.1)
> boxplot(sci_mean ~ ST004D01T, a11)
> layout(1)
```



BASE GRAPHICS

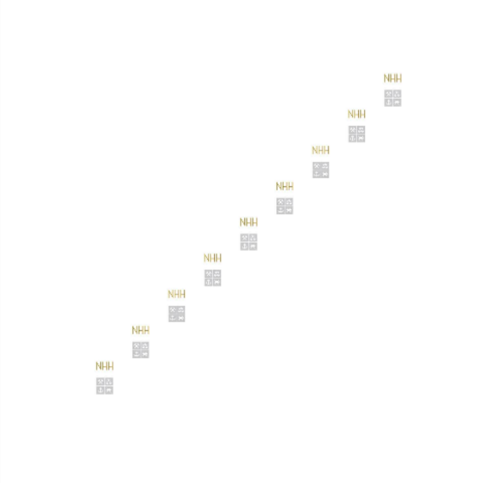
It is tricky to use plot region metrics in RStudioGD, so we’ll try the PNG device to look at device dimensions and to plot some logo images

```
> library(”jpeg”)
> im <- as.raster(readJPEG(”image001.jpg”))
> prop <- dim(im)[2]/dim(im)[1]
> png(”plot1.png”)
> par(”din”)
```

```
## [1] 6.666667 6.666667
```

```
> plot(1, type=”n”, axes=FALSE, xlim=c(1, 10),
+   ylim=c(1, 10), asp=1, ann=FALSE)
> rasterImage(im, 1:9, 1:9, (1:9) + prop, 2:10)
> dev.off()
```

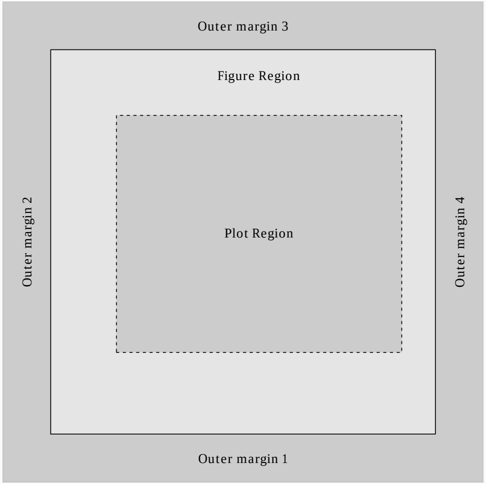
```
## pdf
##   2
```



BASE GRAPHICS

The plot region and figure region are shown using R Graphics code

```
> par(oma=rep(3, 4), bg="gray80")
> plot(c(0, 1), c(0, 1), type="n", ann=FALSE,
+   axes=FALSE); box("outer", col="gray")
> par(xpd=TRUE)
> rect(-1, -1, 2, 2, col="gray90")
> box("figure"); par(xpd=FALSE)
> rect(-1, -1, 2, 2, col="gray80")
> box("plot", lty="dashed")
> text(.5, .5, "Plot Region")
> mtext("Figure Region", side=3, line=2)
> for (i in 1:4) mtext(paste("Outer margin", i),
+   side=i, line=1, outer=TRUE)
```



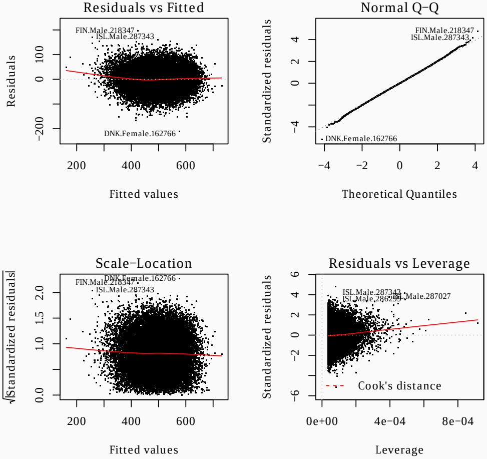
BASE GRAPHICS

- Most of base graphics is vector graphics, but some innovations apply both to base and grid
- These include Raster Images in R Graphics, and complex paths covered in It’s Not What You Draw,It’s What You Don’t Draw
- The **gridBase** package permits base graphics elements, often created as `plot()` methods in contributed packages, to be placed in grid graphics displays

BASE GRAPHICS

The `plot.lm()` method in base package **stats** shows diagnostic plots, here indicating potential outliers; outlier detection is also provided in **OutliersO3** and **HDoutliers** among others

```
> opar <- par(mfrow=c(2,2))
> plot(lm(math_mean ~ read_mean, data=a11), pch=".")
> par(opar)
```



Grid graphics

GRID AND BASE GRAPHICS

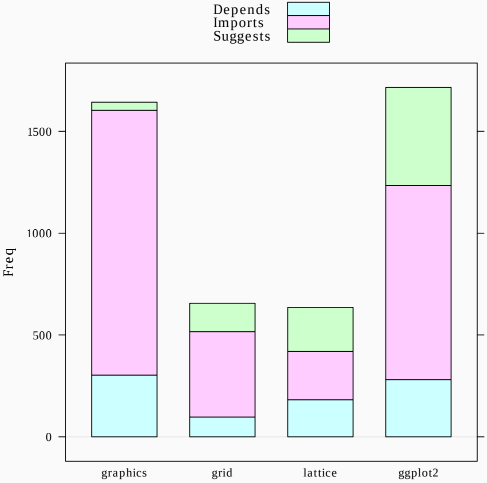
We can check the relative standing of **graphics** and **grid** from the CRAN package database, and add **lattice** and **ggplot2** (Suggests are typically in examples):

```
> db <- tools::CRAN_package_db()
> types <- c(”Depends”, ”Imports”, ”Suggests”)
> pkgs <- c(”graphics”, ”grid”, ”lattice”, ”ggplot2”)
> (tbl <- sapply(types, function(type) sapply(pkgs,
+   function(pkg) length(db[grep(pkg, db[, type]), 1]))))
```

```
##            Depends Imports Suggests
## graphics       303    1300       40
## grid            97     419      140
## lattice        182     238      216
## ggplot2        281     952      482
```

```
> class(tbl)
```

```
## [1] ”matrix”
```



GRID GRAPHICS

- The **grid** and **lattice** entered as Recommended in R 1.5.0 in April 2002, and **grid** became a base package in 1.8.0 in October 2003
- Some changes were made in **grid** for R 3, but its structure remains very stable
- The **gridBase** and **gridGraphics** packages provide functions for capturing the state of the current device drawn with base graphics tools (see The gridGraphics Package)
- One reason for this is the unsolved problem of testing graphics output for identity, to ensure that the same commands for the same data give the same output; for `grid` objects this is feasible, but not for base graphics on interactive devices

GRID GRAPHICS

- Over and above the use of **grid** directly, the general-purpose packages **lattice** and **ggplot2** build on **grDevices** and **grid**
- In addition, it is worth mentioning the **vcd** (visualizing categorical data) and **vcdExtra** packages and a recent book on Discrete Data Analysis with R
- Paul Murrell and co-authors have documented progress in some articles: Drawing Diagrams with R, What's in a Name?, Debugging grid Graphics, The gridSVG Package,

GRID AND BASE GRAPHICS

We can combine `grob` from different sources

```
> b <- barchart(tbl, auto.key=TRUE,
+   horizontal=FALSE)
> x11()
> barplot(t(tbl), legend.text=TRUE,
+   args.legend=list(x="top", bty="n",
+   cex=0.8, y.intersp=3))
```

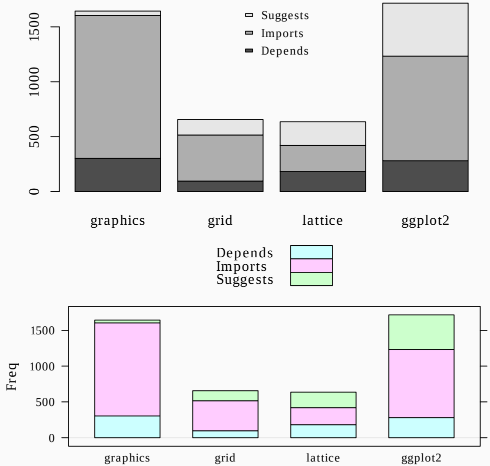
```
> gridGraphics::grid.echo()
```

```
> library(grid)
> g <- grid.grab()
```

```
> dev.off()
```

```
## pdf
##   2
```

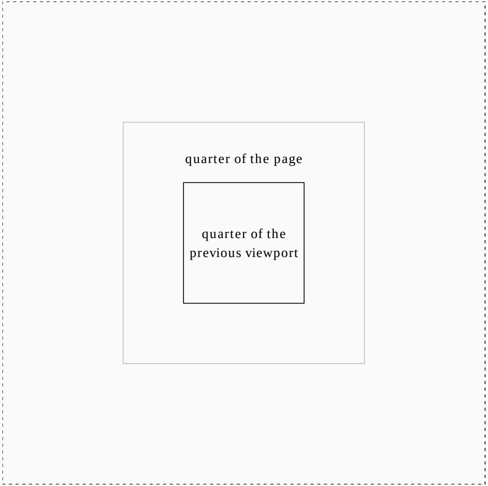
```
> #grid.newpage()
> #gridExtra::grid.arrange(g, b, ncol=1)
```



GRID GRAPHICS

grid pushes viewports onto a stack, then pops them, see R Graphics and vignettes

```
> grid.rect(gp = gpar(lty = "dashed"))
> vp <- viewport(width = 0.5, height = 0.5)
> pushViewport(vp)
> grid.rect(gp = gpar(col = "grey"))
> grid.text("quarter of the page", y = 0.85)
> pushViewport(vp)
> grid.rect()
> grid.text("quarter of the\nprevious viewport")
> popViewport(2)
```



GRID GRAPHICS

Just reading the `print` method for `ggplot` objects shows how close **grid** is under **ggplot2**.

```
> ggplot2:::print.ggplot
```

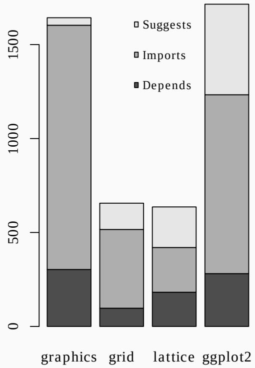
```
## function (x, newpage = is.null(vp), vp = NULL, ...)
## {
##     set_last_plot(x)
##     if (newpage)
##         grid.newpage()
##     grDevices::recordGraphics(requireNamespace(”ggplot2”,
##         list(), getNamespace(”ggplot2”))
##     data <- ggplot_build(x)
##     gtable <- ggplot_gtable(data)
##     if (is.null(vp)) {
##         grid.draw(gtable)
##     }
##     else {
##         if (is.character(vp))
##             seekViewport(vp)
##         else pushViewport(vp)
##         grid.draw(gtable)
##         upViewport()
##     }
##     invisible(data)
## }
## <environment: namespace:ggplot2>
```

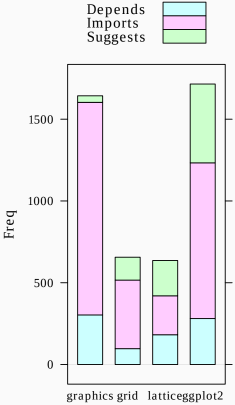
GRID GRAPHICS

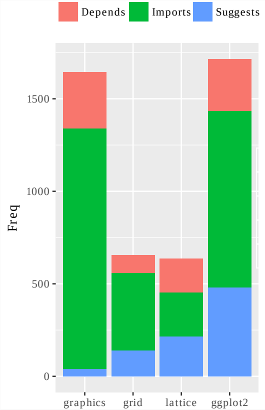
There are various lower and higher-level ways of combining graphical output: some are described in a vignette in the **egg** package

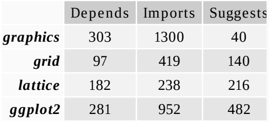
```
> gg <- ggplot(broom::tidy(as.table(tbl)),
+   aes(x=Var1, y=Freq, fill=Var2)) + geom_col() +
+   xlab(””) + guides(fill=guide_legend(title=””)) +
+   theme(legend.position=”top”)
> gg2 <- ggplotGrob(gg)
> t <- gridExtra::tableGrob(as.data.frame(tbl))
> #gridExtra::grid.arrange(g, b, gg2, t, ncol=4)
```

GRID GRAPHICS









	Depends	Imports	Suggests
graphics	303	1300	40
grid	97	419	140
lattice	182	238	216
ggplot2	281	952	482

Interactive graphics

INTERACTIVE GRAPHICS

- Using the standard graphics devices, even if interactive, for interactive graphics is clunky
- From early on, interactive graphics have used compiled (**rggobi**, **rgl**) external libraries or programs, or Java (**iplots**)
- Most recent interactive graphics packages bundle JavaScript libraries: **ggvis**, **plotly**, **googleVis**, **metricsgraphics**, **highcharter** among many; client-side interactivity is the main benefit using JavaScript in a browser
- Using external libraries may make functionalities vulnerable if not updated; see this example for JavaScript

rggobi

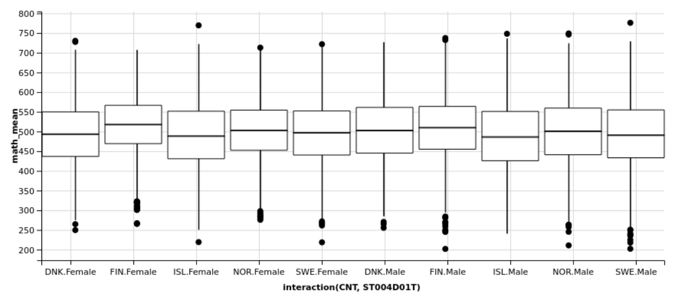
rggobi passes data out to Ggobi for manipulation; Ggobi needs the GTK2 GUI toolbox (now outdated) and an XML library. Typical uses are manipulation of point clouds, project pursuit on point clouds, and interactive parallel coordinate plots. (video on ggobi1-2018-04-22_16.05.30.mp4)

```
> library(rggobi)
> ggobi(a11[,c(1, 4, 39:44)])
```

ggvis

ggvis is an RSudio package not unlike **ggplot2** providing a range of data visualization functions mixing magrittr pipes and formulae. A key feature is integration with **shiny** (separate but related topic).

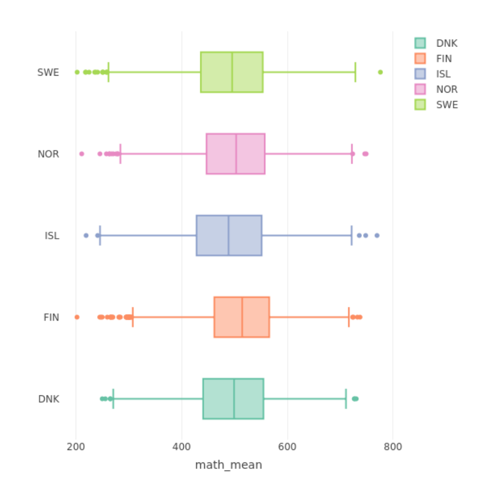
```
> library(ggvis)
> a11 %>% ggvis(~ interaction(CNT, ST004D01T), ~ math_mean) %>% layer_boxplots()
```



plotly

The **plotly** package provides an interface to plotly, a company that "creates leading open source tools for composing, editing, and sharing interactive data visualization via the Web", and lets you pay them if you don't have time or skills:

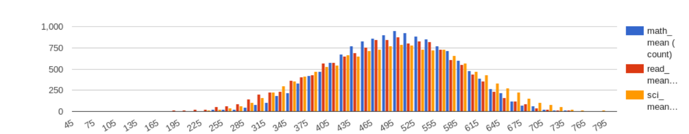
```
> library(plotly)
> plot_ly(a11, x = ~math_mean, color = ~CNT, type = "box")
```



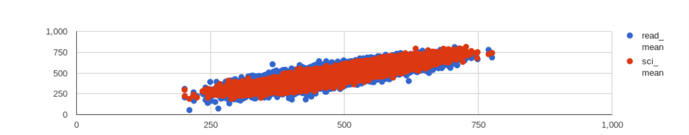
googlevis

googleVis provides an R interface to Google Charts, and is a mixture of R and JS code; JSON is the prime format for transferring data.

```
> library(googleVis)
> plot(gvisHistogram(a11[a11$ST004D01T == "Male", c("math_mean", "read_mean", "sci_mean")]))
> plot(gvisScatterChart(a11[, c("math_mean", "read_mean", "sci_mean")]))
```



Data: a11[a11$ST004D01T == "Male", c("math_mean", "read_mean", "sci_mean")] • Chart ID: HistogramID729239f5d26f • googleVis-0.6.2
R version 3.4.4 (2018-03-15) • Google Terms of Use • Documentation and Data Policy

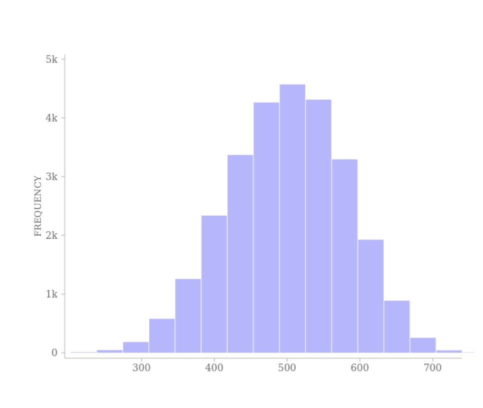


Data: a11[, c(39, 41, 43)] • Chart ID: ScatterChartID72925845a58 • googleVis-0.6.2
R version 3.4.4 (2018-03-15) • Google Terms of Use • Documentation and Data Policy

metricsgraphics

metricsgraphics is another JS library

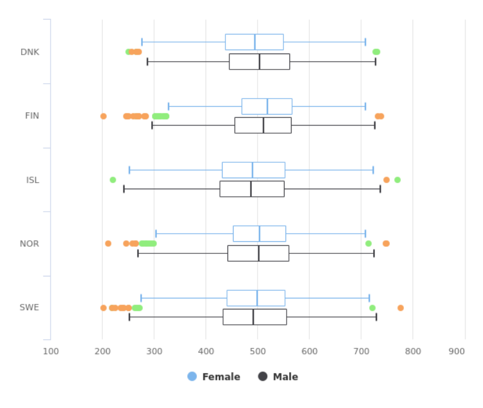
```
> library(metricsgraphics)
> a11$math_mean %>% mjs_hist()
```



highcharter

highcharter is yet another JS library

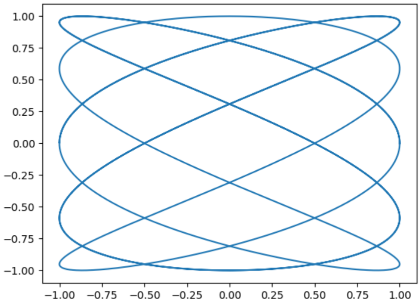
```
> library(highcharter)
> ## Highcharts (www.highcharts.com) is a Highsoft
> ## software product which is not free for
> ## commercial and Governmental use
> hcboxplot(x=a11$math_mean, var2=a11$ST004D01T,
+   var=a11$CNT)
```



reticulate AS AN INTERFACE TO MATPLOTLIB UNDER PYTHON

A recent blog points to the possibility of using the **reticulate** interface to Python to permit the use of `matplotlib`

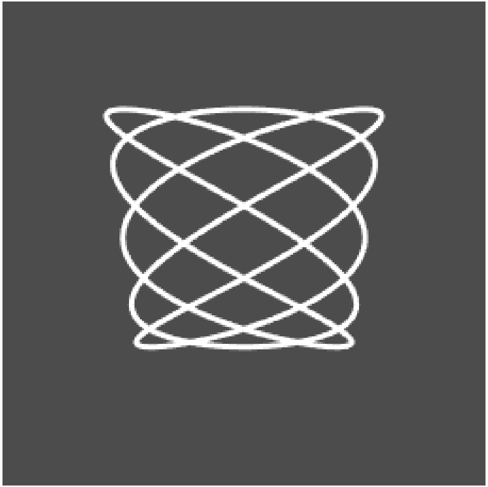
```
> library(reticulate)
> np <- import("numpy")
> plt <- import("matplotlib.pyplot")
> phi <- np$arange(0, 3*np$pi, 0.0025)
> x <- np$cos(5*phi)
> y <- np$sin(3*phi)
> plt$plot(x,y)

## [[1]]
## Line2D(_line0)

> plt$savefig('sampleFileName.png')
> #plt$show()
```



rgl INTERACTIVE OPENGL 3D VISUALIZATION

The **rgl** package has provided an interface for manipulating 3D point clouds and objects for many years

```
> library(rgl)
> z <- rep(1, nrow(x))
> rgl.open()
> rgl.points(x, y, z)
> rgl.snapshot("rgl_snapshot.png", top=TRUE)
> rgl.close()
```



POWERBI

Here are a number of links to blog posts and tutorials on using R graphics in Power BI: Interactive R visuals in Power BI, R Script Showcase, Create Power BI visuals using R

R’s sessioninfo( )

```
> sessionInfo()
```

```
## R version 3.4.4 (2018-03-15)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Fedora 27 (Workstation Edition)
##
## Matrix products: default
## BLAS: /home/rsb/topics/R/R344-share/lib64/R/lib/libRblas.so
## LAPACK: /home/rsb/topics/R/R344-share/lib64/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8
##  [2] LC_NUMERIC=C
##  [3] LC_TIME=en_GB.UTF-8
##  [4] LC_COLLATE=en_GB.UTF-8
##  [5] LC_MONETARY=en_GB.UTF-8
##  [6] LC_MESSAGES=en_GB.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8
##  [8] LC_NAME=C
##  [9] LC_ADDRESS=C
## [10] LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_GB.UTF-8
## [12] LC_IDENTIFICATION=C
##
## attached base packages:
```