Exploring data by interactive visualization: an introduction to the loon system

Adrian Waddell, Zehao Xu, and Wayne Oldford*
University of Waterloo

Article Type:

Primer, Overview, Software Focus

Abstract

Three different scientific problems (seismic events in the Tonga trench, risk factors for coronary heart disease in a South African study, and the patterns defining the periodic table of the elements) are investigated via interactive data visualization using the open source software system called loon. A relatively small set of interactive and linked displays, such as two- and three-dimensional scatterplots, and interactive histograms and bar charts, are shown to be powerful tools that enable the analyst to quickly raise, and even answer, scientific questions from the data with minimal to no programming in loon. Nearly arbitrary logical queries are shown to be able to be put to the data. Through these examples, the reader will see how to use loon in data analysis, understand the essential concepts of direct interaction visual exploration, some of the novel design features of loon, and, perhaps most importantly, come to appreciate whether, and how, their own scientific investigations might benefit from exploratory data analysis using an interactive visualization system like loon.

*Contact author: rwoldford@uwaterloo.ca

Introduction

Data visualization is essential to efficient exploratory data analysis. Our powerful visual system identifies patterns, whether or not they were anticipated in advance. It also easily detects departures from patterns observed. When interactive, through mouse and short keyboard gestures, visualizations can be queried and updated without the analyst changing focus from the displayed data. Analysis flows more quickly when using direct interaction visualization to explore data. This is best appreciated when applied to the analysis of real data, embedded within their scientific context.

Three separate scientific contexts and data sets (locations of Tonga trench earthquakes, coronary risk factors for Afrikaner men, and the periodic table of the elements) are used in the next section to demonstrate the direct interaction visualization system called loon (after the northern diver bird). Written mainly in TclTk (Flynt, 2012), loon adds interactive visualization to any data analysis scripting language having an interface to Tk (e.g., including TclTk, R, Python, and Julia). Presently, loon is most integrated R, working with several of its packages, effectively extending R, and will be the implementation (including R code snippets) used in the example analyses. The reader is encouraged to download and use the R loon package (Waddell & Oldford, 2021) while following the examples in the next section. Features of loon are incrementally introduced through the examples.

A more general discussion of direct interaction follows the example analyses. This includes a description of the display linking model used in loon and an illustration of software hooks designed for the analyst to adapt loon interactions to their own needs. The paper closes with some brief remarks.

Exploring data via interactive visualization

The examples begin with using linked two and three dimensional interactive scatterplots in loon to better understand the locations of seismic events near the Tonga trench. The basics of interactive visualization in loon are introduced here and illustrated by exploring data features of scientific interest.

Next, risk factors or coronary heart disease among Afrikaner males is investigated interactively via linked histograms in loon. The concept of layering is introduced and applied using statistical functionality available in R. The example also shows how direct interaction facilitates immediate examination of different conditional distributions. Odds-ratios are constructed and compared interactively. This invites fairly general queries involving odds-ratios of various risk factors under different conditions. How logical queries (of nearly arbitrary complexity) can, in general, be formed and immediately answered by direct interaction is demonstrated by examining the different odds-ratios.

Finally, the periodic table of the elements is constructed as an interactive scatterplot. This illustrates how analyses sometimes profit from interactive scatterplots where points are represented by arbitrary glyphs given by any drawing the analyst might choose, in this case each point being a standard presentation of an element in the periodic table. Interacting with the table, and other linked plots, uncovers its periodic nature. This, in turn, suggests possible value in splitting plots into separate pieces, or "facets", in exploratory analysis.

Sufficient scientific context is provided in each example so that the analysis proposed should seem natural. The code contained in the text is complete for the first two examples.

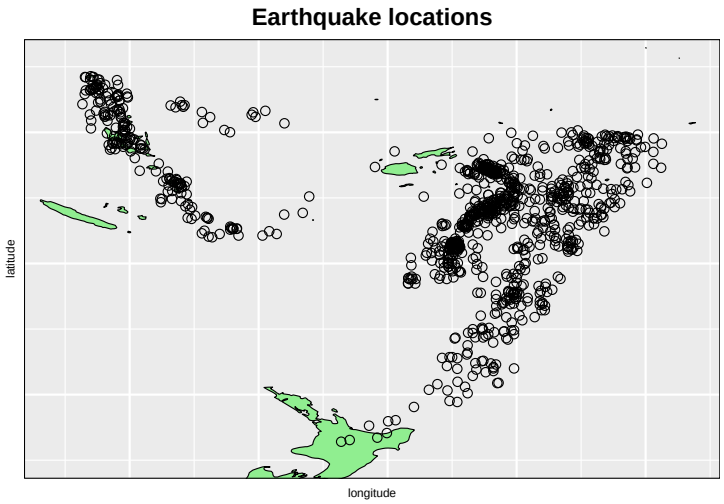
Example 1: Exploring earthquakes

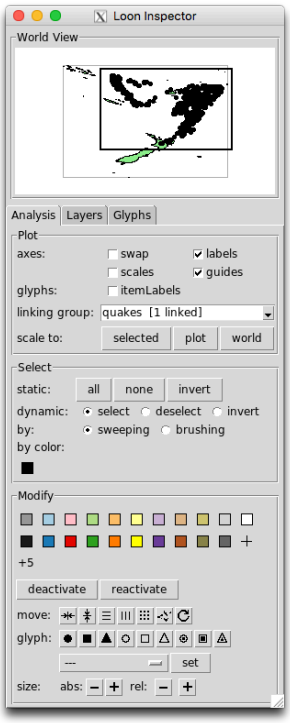
The `quakes` data from R record the `lat`, `long`, `depth` (in kilometres), `mag` (Richter scale magnitude), and number of reporting `stations`, for seismic events occurring near Fiji between 1964 and 1979 – judging the dates from Giardini and Woodhouse (1984, Figure 1). The events are a subsample of 1,000 from an original 5,000 having a "body-wave" or "Mb" magnitude greater than 4.0 as given by Woodhouse to the Harvard `PRIM-H` project (D. Donoho, Huber, & Thoma, 1981; D. L. Donoho, Huber, Ramos, & Thoma, 1982).

We consider here only the location variables: `lat`, `long`, and `depth`.

The interactive scatterplot

A loon scatterplot of longitude and latitude locations is shown in Figure 1 along with the code that produced it.






```
p <- with(quakes,
        l_plot(x = long, y = lat,
               xlabel = "longitude",  ylabel = "latitude",
               title = "Earthquake locations",
               glyph = "ocircle", size = 6, color = "black",
               linkingGroup = "quakes"))

l_layer(p, NZFijiMap,
        label = "Tonga trench islands", color = "lightgreen")
```

Figure 1: A two dimensional scatterplot in loon (with a map `layer`) and its `inspector`.

The function `l_plot()` *is designed to resemble* `plot()` from R's base graphics package, but with more evocative names for several arguments (e.g., `title`, `glyph`, and `size` in place of `main`, `pch`, and `cex`) and with additional arguments peculiar to interactive graphics (e.g., `linkingGroup` identifies a group of plots whose *linked display states* change in concert).

Unlike `plot()`, `l_plot()` returns a data structure, assignable to a variable as shown in Figure 1, whose features can be changed in place – either *programmatically* by changing their state using a variety of functions, or *interactively* via mouse (and other) events and gestures.

For example, R's square bracket extraction function `[]` is specialized for loon plots so that `p["color"]` returns a vector of its point colours; similarly `p["color"] [1:3] <- "red"` changes the first three points to red. All plot *states* accessible and settable in this way are given by `names(p)`. The same can be given as named arguments to `l_plot()` used to set these values at creation. Though technically a tcltk data structure, a loon plot mimics an R structure through specialized functions and S3 methods. A useful example of the latter is R's `plot()`

function, which when called on a loon plot,

More complex programmatic interactions are effected by functions specialized for loon plots. One such function is R's `plot()` which, when called on a loon plot object, will translate it into a grid graphics object (e.g., `plot(p)` produced the plot of Figure 1). This enables the analyst to take *a snapshot of the interactive display* at any time in the analysis.

Loon plots have *layers*, each one containing any number of geometric objects. Layers can be added, removed, hidden, shown, raised, lowered, and grouped within a loon plot; their position in a list determines their display order in the plot. The second command line in Figure 1 adds `NZFijiMap` to `p` as the *layer* labelled `"Tonga trench islands"` (`NZFijiMap` is a `map` object extracted from the `"world2"` database of the maps package of R). Only one layer, the *model layer* (here the points), is *interactive* – in that its contents are accessible for direct mouse manipulation on the plot.

The inspector

Whenever a plot is created its loon inspector appears with it (Figure 1 at right). This *inspector* is focused on, and tightly coupled with, the *currently active* plot (in this case `p`). If closed, the inspector will reappear whenever the plot again becomes the topmost window.

A miniature, nearly identical, version of the scatterplot appears as the "World View". Absent from the world view are details like title, labels, and background guidelines. Being a world view, however, means it shows all of the `NZFijiMap` layer, including parts of the map presently clipped from the loon plot `p`. The black thick bordered rectangle in the world view shows the location and extent of `p` as it appears in the scatterplot of Figure 1; the light grey rectangle shows the extent of the so-called "world", namely *all visible layers* of `p`. Whatever is contained within the black rectangle of the world view is displayed in the scatterplot window and vice versa – the black rectangle could be smaller or larger than the grey rectangle and even appear outside it, depending upon the current display of the scatterplot. Resizing or moving the black rectangle in the world view effects the same change on the scatterplot.

Three tabs appear below the world view: Analysis, Layers, and Glyphs. Each encapsulates a different type of interaction with `p`. The Analysis tab is present in the inspector for every type of loon plot and organizes common interactions into three distinct sections: "Plot",

"Select", and "Modify". These three sections are universal to all types of loon plots, though details may differ (e.g., scatterplots vs. histograms).

The Plot section of the Analysis tab has a variety of check boxes, pull down menus, and buttons that interact with the scatterplot. The text beside linking group is the value given by the argument `linkingGroup`. Three scale to buttons rescale the scatterplot in its window to the extent of: the "selected" points in the scatterplot, just the plot model layer, or the entire world of the plot. Check boxes provide Boolean values for different parameter *states* of the plot. For example, "labels" and "guides" are checked, respectively indicating that the `title` and axis labels appear around the plot and that visual guides of gridlines are shown in its background. Had the arguments `showLabels = FALSE` and `showGuides = FALSE` been given to `l_plot()`, these boxes would have been unchecked. The remaining check boxes in this section have `FALSE` as their default values.

The Select section is used to select points in the plot. Selection operations are distinguished as being:

- static (applies to the whole plot). Buttons cause all or none of the points to be selected, or simply invert the current selection in the plot (each point inverts its selection status).

- dynamic (how selection is done dynamically on the plot). In the plot, points may be selected one at time using the `<left>` (or primary) mouse button, one after another with a `<shift><left>` or "sticky" selection, or either of these for whole groups of points at a time via sweeping out a rectangle in the window, or via brushing a fixed rectangle over the points. The mode of dynamic selection is determined by the radio buttons select, deselect, or invert (interpreted as with the static selections).

- by color. A patch appears for every colour of point in the plot. Selecting any one (or more with the shift key) colour patch selects those coloured points in the scatterplot.

The Modify section organizes actions applicable to the selected points. Selected points can have their colours changed by clicking on a colour patch, be rendered invisible by the deactivate button (and returned to visibility by reactivate), be moved into different location configurations according to the buttons to the right of move (aligned vertically, horizontally,

in rows, in columns, on a grid, jittered, or returned to their original position), have the glyph shape changed to solid, open, or closed circles, squares, or triangles, and even to an arbitrary glyph shape constructed by the user (available from a pull down menu), or resized in unison either maintaining a common absolute (abs) size or changing together while preserving relative (rel) sizes.

Other sorts of organization and division of activities occur when either of the other two tabs are selected. The Layers tab brings up a list of the layers together with interactive buttons that can change the display order of layers, render selected layers visible or not, delete layers, and even organize them into nested hierarchies. The Glyphs tab lists any additional glyph choice added by the user. Some special glyphs, such as star-shaped radial axes glyphs or parallel coordinate glyphs, provide interactions peculiar to that glyph which would also appear on this tab if added to the plot.

An interactive 3D scatterplot

All three spatial dimensions (`long`, `lat`, and `depth`) of the seismic events can be plotted in loon using `l\_plot3D()` in Figure 2. The difference in measurement scales meant the data had to be scaled via `l_scale3D()` to ensure the points would appear within a box of equal width, height, and depth. The 3D scatterplot looks very much like that of Figure 1 except for some missing features and the appearance of a pair of coloured axes at the centre of the plot. The third axis, `depth`, does not appear because it is perpendicular to the plane of the display.

Two tiny changes also appear in the inspector. The linking group value quakes [2 linked] indicates that there are now two plots in the group and a new button appears in the move section as a tiny 3D set of positive axes. When pressed, this move button returns the 3D axes of the plot to their original orientation and the data points follow.

The illusion of three dimensions is effected through real time rotation controlled by the user. Pressing the keyboard letter `<r>` toggles the plot into Rotation Mode which is then displayed on the plot. The data points are rotated in 3D either freehand using the mouse or about the screen axes using arrow keys. The axes shown in the plot identify the centre of the rotation (e.g., initially the arithmetic average of the scaled data coordinates).

Four snapshots from pressing the `<down-arrow>` are shown in Figure 3. From top to

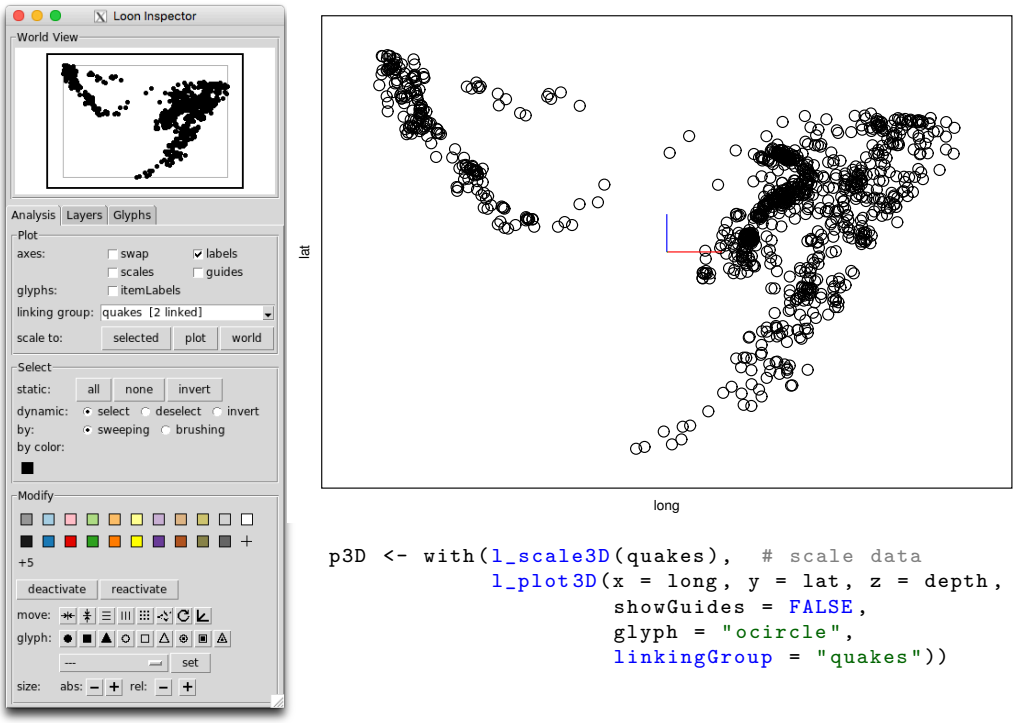


Figure 2: A three dimensional scatterplot in loon.

bottom, the point cloud rotates about the fixed horizontal longitude axis, its latitude axis rotating up and out of the plane of the screen as, simultaneously, from its initial position pointing out of the screen, the depth axis rotates down into the plane of the display. By the last plot in the column, the latitude axis has become invisible (being perpendicular to, and sticking out of, the screen) and the depth axis has become the downward pointing vertical axis in the plane of the screen. Throughout the rotation, the labels on the horizontal and vertical axes change continuously to reflect the linear combinations of the original variates which define the current screen axes. Once the rotation was complete, the point cloud was recentred by pressing the `<right-button>` (or secondary button) and moving the point cloud. Figure 3 shows the resulting scatterplot of negative `depth` versus the `long` of the data – the lower the points are vertically on the screen, the deeper the seismic event.

Analysis via linked interaction

Giardini and Woodhouse (1984) focused on the eastern group of quakes, those along the so-called Tonga trench. To isolate these, the other quakes are selected and deactivated,

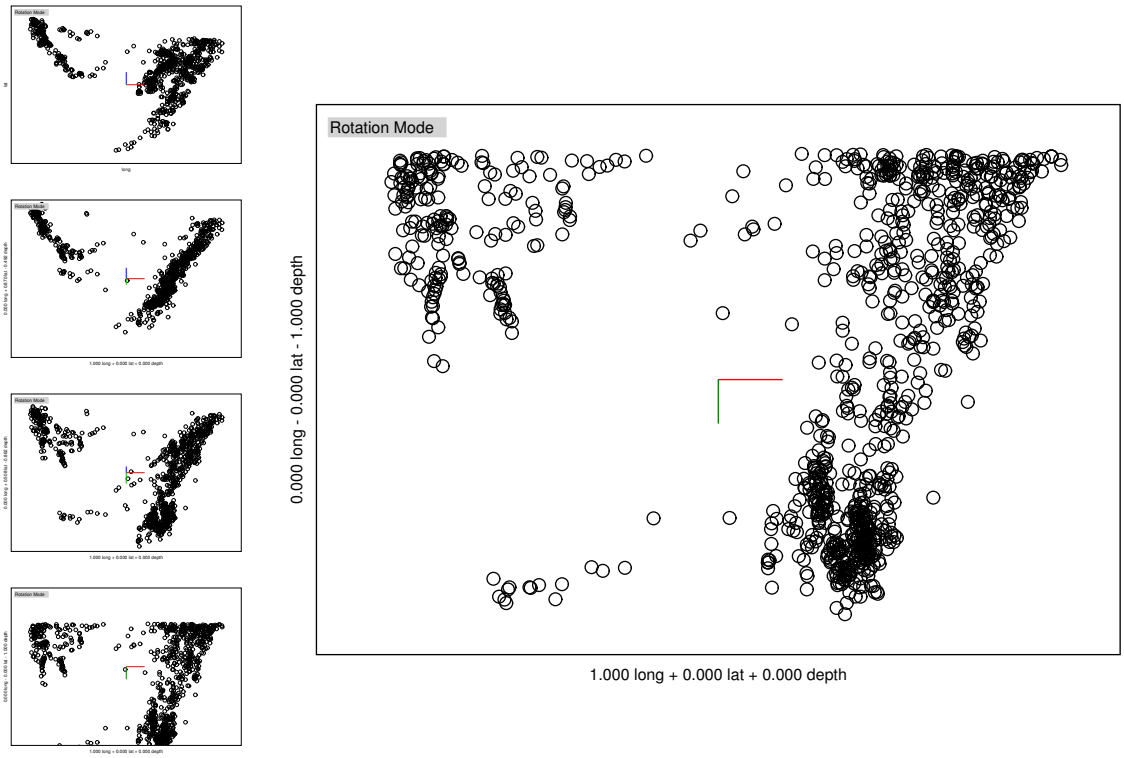
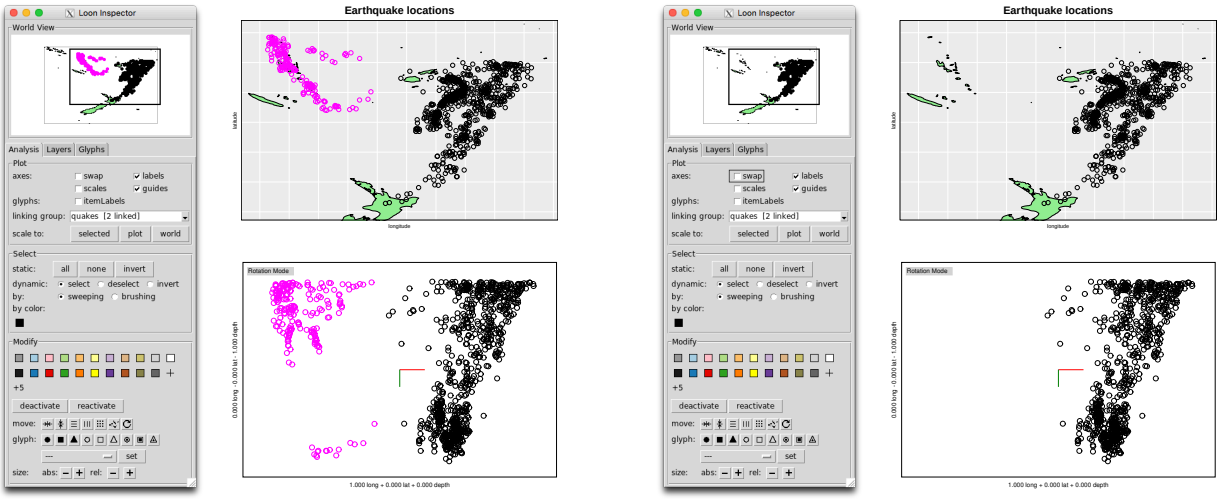


Figure 3: Four rotation positions (each 60° apart) using the `<down-arrow>` key (left). The rotated display after centring the data using the mouse `<right-button>` appears at right. Note linear combinations now describe the horizontal and vertical display axes.

leaving only those in the Tonga trench displayed in the plot. This is done by clicking on the background grid in the top left corner of the scatterplot of Figure 1 and sweeping out a rectangular region encompassing only the two groups in the northwest corner of the map. As the rectangle sweeps over them, the points are highlighted in all linked plots, and in the inspector, as shown in Figure 4a. Pressing the deactivate button in the inspector renders the selected points invisible and the plots appear as in Figure 4b. (N.B. the activate button will make all deactivated points visible again.)

Each plot has an associated set of *linked states* that will change whenever these states change in any linked plot. By default, linked states are selected, active, and color. Different types of loon plots may include other linked states (e.g., size for scatterplots) and individual plots may have their linked states changed at any time in the analysis (e.g., `p` could exclude size, or include glyph to have the point shape change with linking).



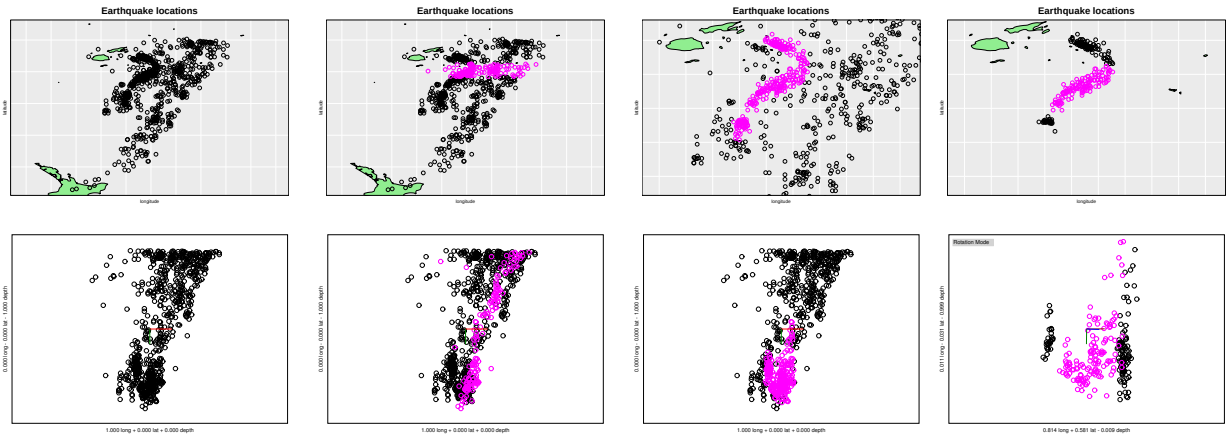
(a) Western quakes selected.

(b) Western quakes deactivated.

Figure 4: Removing events outside the Tonga Trench.

Each plot can now be rescaled and relocated interactively to focus on the remaining points in the display. The analyst *zooms* in a plot area by placing the mouse over a target position and scrolling. The mouse location is the centre of rescaling and the `<scrolling>` function does the rescaling. The plot region is then relocated with the `<right-button>` mouse movement. Each operation can be modified to constrain its direction: with the `<ctrl>` key for only horizontal transformation or with the `<alt>` (or `<cmd>`) key for only vertical. In this way, the analyst can interactively *stretch* and *pan* the plot in one direction at a time. For example, horizontal stretching and panning can be especially useful in the exploratory analysis of a long time series. Figure 5a shows the result of rescaling and relocating each plot to focus on the Tonga Trench events.

Giardini and Woodhouse (1984) then examine the depth of quakes within a few degrees of latitude of one another. Interactively, in the 2d scatterplot a low wide *brush* is created and used to examine the depth structure of quakes in narrow slices of latitude; by vertically *brushing* the map north to south the effect on depth is observed in the linked 3D plot of depth versus longitude. Statistically, the analyst is examining the joint distribution of longitude and depth conditional on (narrow bands of) latitude. Figure 5b shows one slice revealing structure that is typical of all such slices. As the brush moves from north to south in the



(a) Eastern quakes. (b) Latitude shear. (c) The hook. (d) Three zones.

Figure 5: Exploring the Tonga Trench quakes

map, a consistently vertically diagonal slash moves from right to left in the 3D plot. Each latitudinally narrow slice reveals a deep vertical shear of the quakes longitudinally. These are known as "Benioff zones" which, in the words of Giardini and Woodhouse (1984, p. 505), "...with the advent of plate tectonics, have been recognized as the trajectories of oceanic lithosphere subducted deep into the Earth's mantle."

Of particular interest to Giardini and Woodhouse (1984) are seismic events which are deep ($>$ 400 km.) within a Benioff zone. They turn their attention to the question mark shaped "hook" near Tonga. Zooming in on the map, and using a small "sticky" (multiple selection) brush, the hook is highlighted in the top plot of Figure 5c. As can be seen in the 3D plot at bottom, these are amongst the deepest earthquakes recorded in the zone – nearly all range from 500 to nearly 700 kilometres deep. The hook is isolated by inverting the selection of Figure 5c and deactivating the newly highlighted points.

Giardini and Woodhouse (1984) divide the hook into four regions, a northern region of quakes, a central region, a southern region (the dot of the question mark), and the gap between the central and southern regions. The map of Figure 5d shows the hook isolated with its central region selected. The 3D plot of Figure 5d has zoomed in on the hook events and rotated them. The horizontal axis now provides a few of the depths from an angle oblique to the lines of both latitude and longitude. The southern (left) and northern (right)

regions both show vertical shear like that seen in Figure 5b at much shallower depths.

The angled, or letter "J", shape of the earthquakes in the central (highlighted) region seen in the 3D plot of Figure 5d is of particular interest. The shape is indicative of deep shearing deformation in two different directions: one in the plane of the Benioff zone (vertical part of the "J") and a second "out-of-plane" shear (left part of the "J"). Giardini and Woodhouse (1984, p. 507) "...infer that in this central zone there are simultaneously active modes of shear deformation both in and out of the plane of the Benioff zone." They conclude that the patterns in this deepest part of the descending plate "indicate thickening and bottoming out of the descending material, and suggest that subduction experiences great resistance in penetrating the 670-km discontinuity."

Many more plots are studied in Giardini and Woodhouse (1984) to justify this inference. For our purposes, the important point is that all could be reproduced here using interactive data visualization as in loon – 2D and 3D scatterplots together with zooming, moving, linking, selection, and isolation, provide a powerful suite of exploratory tools for scientific data analysis. Indeed, this kind of interactive 3D exploration would have been the principal reason for sharing the data with the Harvard `PRIM-H` researchers (D. Donoho et al., 1981; D. L. Donoho et al., 1982).

Example 2: Coronary risk factors

In the late 1970s, an unusually high incidence of ischaemic heart disease had been observed to exist amongst white Afrikaans-speaking segments of South African society (Wyndham, 1982). To explore the prevalence and intensity of coronary risk factors (especially possibly reversible ones like hypercholestrolaemia, hypertension, and smoking), Rossouw et al. (1983) recruited about 82% of the known target population of three Afrikaner communities in the southwestern Cape Province (3,357 white males and 3,831 white females).

Hastie and Tibshirani (1987) select a subset of these data, namely all 162 cases with coronary heart disease (`chd = 1`) along with 303 controls sampled from the remaining survey subjects. These appear to be used again by Hastie, Tibshirani, and Friedman (2009a, 2009b), although with fewer observations (160 cases and 302 controls). The data were made available as `SAheart`, a data frame in a now archived ElemStatLearn companion R package (Halvorsen,

2015). It has since been reproduced in the loon.data package (Oldford & Waddell, 2021) where it appears in slightly different form. In ElemStatLearn, the variable `chd` representing the binary outcome of whether the subject had coronary heart disease or not is recorded as a `numeric` `1` or `0`, whereas in loon.data it is a two level `factor`. The `SAheart` as originally presented in ElemStatLearn will be used here.

Of interest is the relationship between the (numeric) binary response `chd` and any of the nine risk factors recorded in `SAheart`. Only two risk factors are considered here, one potentially reversible (systolic blood pressure as `sbp` in mm mercury), the other not (presence or absence of a family history of heart disease as binary factor variable `famhist`). Prevalence and intensity of risk factors will be explored using interactive histograms.

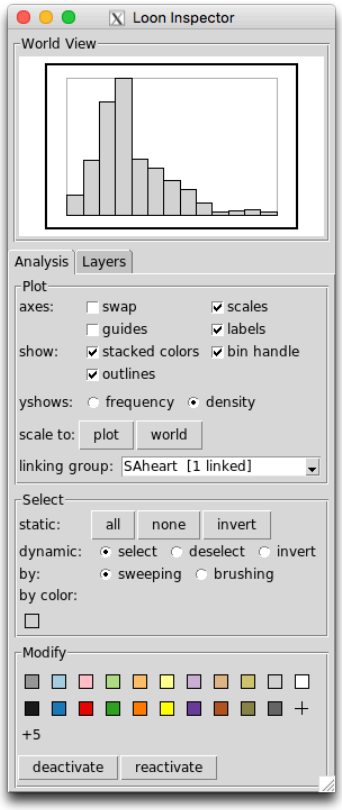
The interactive histogram: systolic blood pressure

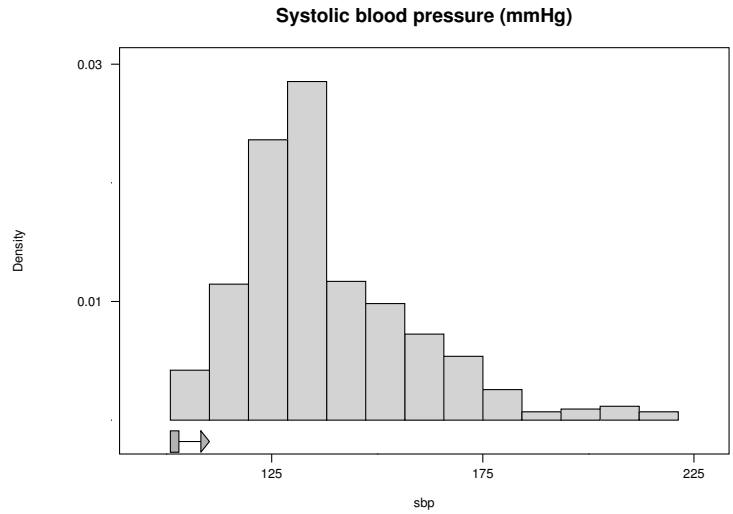
In loon a histogram is created using `l_hist()` as shown, for example, for the systolic blood pressure variable `sbp` from `SAheart` in Figure 6. Like `l_plot()`, the syntax of `l_hist()` intentionally resembles that of its base R counterpart `hist()`. A histogram appears in a new window and the inspector adapts to the histogram.

In loon a single inspector exists by default and switches its focus to whichever loon plot is active at the time. Much of the inspector's organization and layout remains the same, whichever plot is its focus. For a histogram, the organization is much like that for a scatterplot. Some of the features, such as the logical check boxes and radio buttons in the Plot section, will be only for histograms. For example, in Figure 6, yshows appears and is one of frequency or density depending on whether the histogram shows counts or density (the latter having unit area). Features peculiar to scatterplots, such as the shape, size, or movement of point glyphs, do not appear in the inspector of a histogram. Colour modifications remain and apply to selected (parts of) bars in the histogram.

Interaction and layering

Direct interactions (e.g. mouse selection, panning, zooming, etc.) follow the same logic as before. Similarly, programmatic interaction with the variable `h_sbp` allows querying of, or changes to, the states of the histogram object, and hence display.



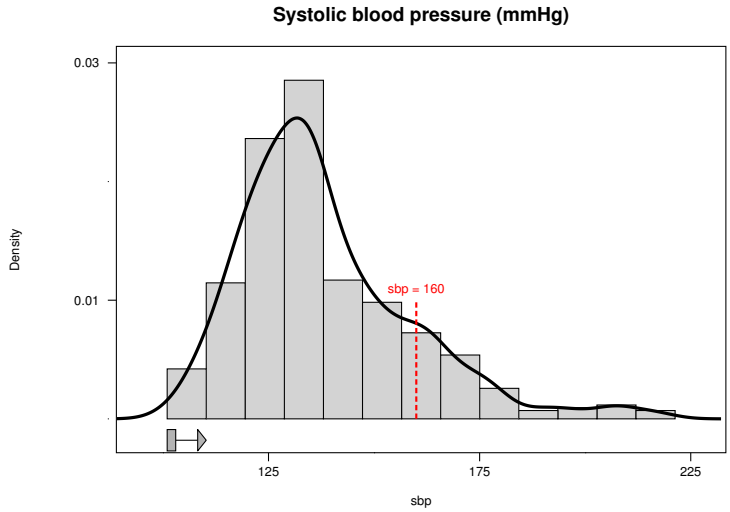



```
h_sbp <- with(SAheart,
              l_hist(sbp,
                     title = "Systolic blood pressure (mmHg)",
                     showScales = TRUE,
                     yshows = "density",
                     showBinHandle = TRUE,
                     linkingGroup = "SAheart"))
```

Figure 6: The histogram of `sbp` (systolic blood pressure) and its inspector.

It would be natural, for example, to layer a continuous density estimate on the histogram. This is easily done in R, where the function `density()` from the stats package will produce a density estimate. The density data structure is recognized by the generic loon function `l_layer()` so as to add the density curve to the histogram. This is shown by the first line of code in Figure 7. More generally, loon provides functions for layering nearly any kind of geometric shape on a loon histogram or scatterplot.

For example, one definition of hypertension (e.g. World Health Organization) is a blood pressure reading of systolic/diastolic $> 160/95$ mmHg. We might mark this boundary on the histogram by adding a line and text at `sbp = 160`. The remaining code in Figure 7 does this. There, `l_layer_line()` adds to `h_sbp` a red vertical line segment given by the coordinates `x` and `y` and dashing defined as 5 pixels solid followed by 3 pixels blank. The function `l_layer_text()` adds the text `"sbp = 160"` above the line segment. The result of these two calls would appear




```
sbp_density <- l_layer(h_sbp, density(SAheart[,"sbp"]), linewidth = 3, label = "density")
# Add a group of layers
high_sbp <- l_layer_group(h_sbp, label = "hypertension")
l_layer_line(h_sbp, x = c(160, 160), y = c(0, 0.01),
             color = "red", linewidth = 2, dash = c(5,3), label = "line at 160",
             parent = high_sbp)  # parent is group layer
l_layer_text(h_sbp, x = 160, y = 0.011, text = "sbp = 160",
             color = "red", size = 12, label = "text at 160",
             parent = high_sbp) # parent is group layer
```

Figure 7: Adding layers to the `sbp` histogram: a density estimate and marking hypertension.

as shown in the histogram on the right of Figure 7. The function call `l_layer_group()` which preceded these, though unnecessary, was added to help organize the layers. The arguments `parent = high_sbp` ensured that the line and text layers were group together as a single layer in the plot.

Selecting the Layers tab changes the histogram inspector as shown in Figure 8 where five layers appear as part of the plot. One, labelled "density" is greyed out since it is presently not visible in the plot; the eyeball icon stroked out beside it indicates it is hidden. Layers are drawn in the order in which they appear with lower layers drawn first and hence appearing below those drawn later. The topmost layer is a group layer containing a text and a line layer; it is drawn last and appears above the histogram. Layers can be reordered, grouped, nested, hidden, revealed, renamed, and deleted using the buttons at the bottom of the inspector. Alternatively, the analyst can interact with any layer programmatically (e.g. as shown by the code appearing in Figure 8).

Some direct manipulation features are peculiar to a histogram. The box and arrow icon

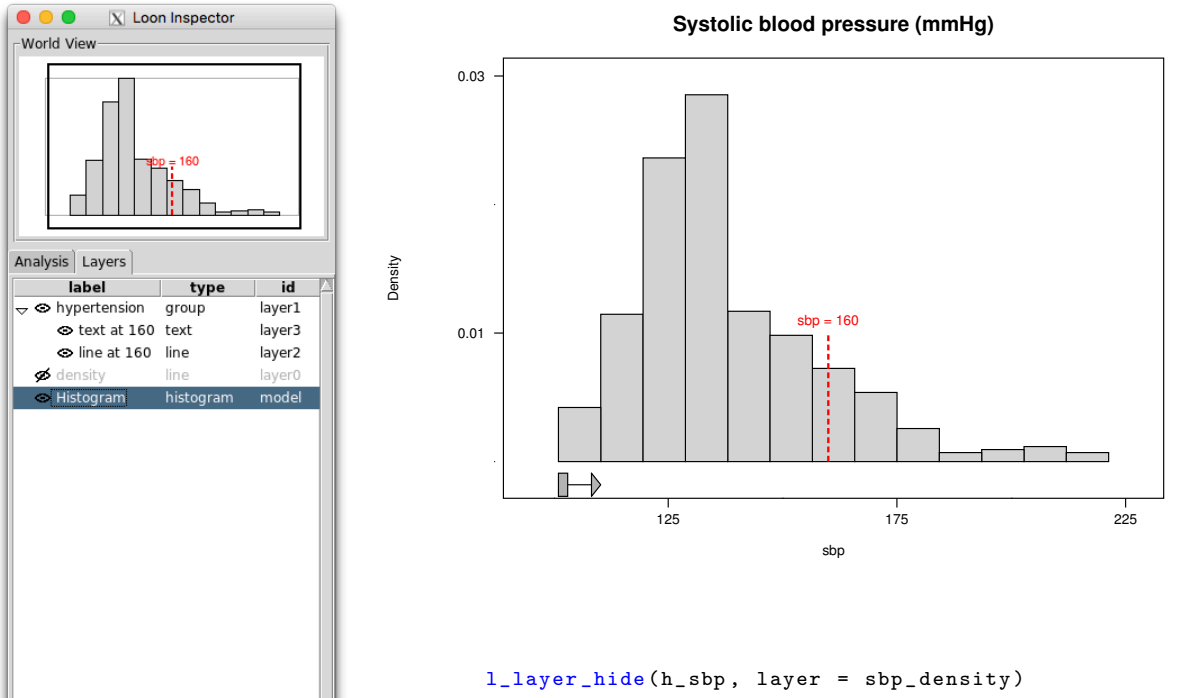
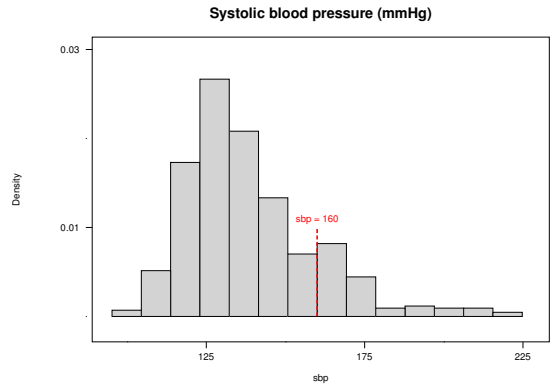
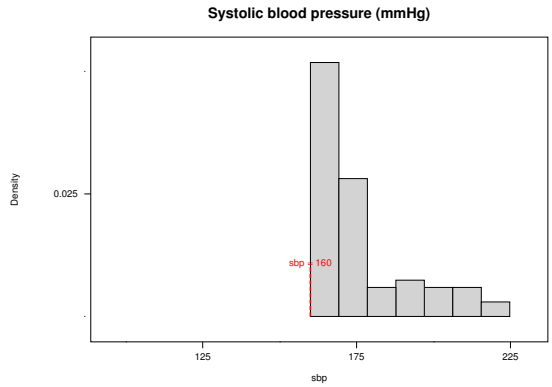


Figure 8: Editing layers from the Layers tab of the inspector or from the command line.

appearing in the histogram of Figures 6-8 is the bin handle (specified at creation in Figure 6). The box at its left end identifies the left boundary of a bin and the arrow tip the right boundary. Selecting and moving the box changes the location of the bins; selecting and moving the arrow tip changes their size (all equal in loon). For example, the histogram on the left of Figure 9 resulted from moving the box to create a boundary at the line `sbp = 160` and then unchecking the bin handle in the inspector. The same effect could be had programmatically, using the code shown immediately below the left histogram in Figure 9.

To focus on those who are hypertensive, all bars to the left on the vertical line can be selected and deactivated interactively, or programmatically, as shown in the right half of Figure 9. The vertical axis adjusted automatically so that the resulting unit area histogram provides and estimate of the *conditional* density of `sbp` given $\texttt{sbp} \geq 160$. Reactivating all deactivated subjects removes the condition and returns the histogram to that of the left






```
# Programmatically relocate bin boundary
h_sbp["origin"] <- 160
h_sbp["showBinHandle"] <- FALSE
```

```
hypertensive <- (SAheart[, "sbp"] >= 160)
h_sbp["active"]  <- hypertensive
l_scaleto_plot(h_sbp)
```

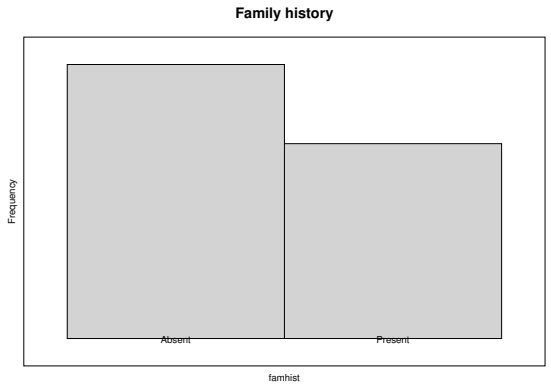
Figure 9: Programmatic (or direct manipulation) setting of the bin boundary origin (at left) followed by focusing only the hypertensive subjects (at right) through the `"active"` state.

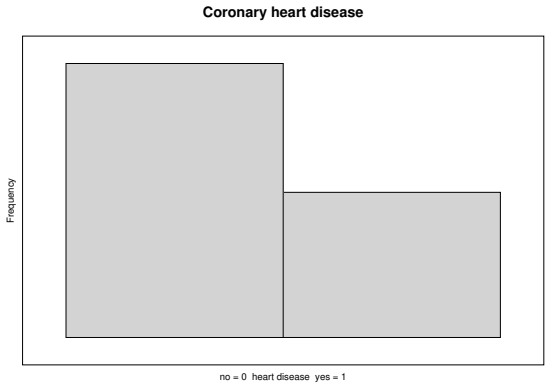
hand side of Figure 9. Again, this can be done directly by pressing the reactivate button on the inspector, or, programmatically via `h_sbp["active"] <- TRUE`.

Histograms or barplots?

Consider two binary variables: the `numeric` valued response `chd` and the family history risk factor `famhist` recorded as a `factor` with two levels(`"Absent"` and `"Present"`). In R, the counts on the former would be displayed as a histogram with `hist()` and the latter as a barplot with `barplot()` after the frequencies of each level were first found. In loon, `l_hist()` handles both `numeric` vectors and `factor` levels whose order determines the bar order. Considering both as "histograms" reflects, and expedites, the common use intended by the display in interactive analysis.

The two displays, one for the risk factor `famhist` and one for the numeric response `chd` are created as shown in Figure 10. Note the differences in the displays and the code used to create them. The leftmost display is the barplot of the `factor` `famhist`, whose levels are layered as text to identify the bars; the rightmost shows the histogram of the `numeric` binary variable `chd`. Both show counts rather than density values as heights (the default value for






```
with(SAheart,
        l_hist(famhist,
                  title = "Family history",
                  linkingGroup = "SAheart")
     ) -> h_famhist
```

```
with(SAheart,
  l_hist(chd, binwidth = 1,
         title = "Coronary heart disease",
         xlabel = paste("0 = no",
                        "heart disease",
                        "yes = 1"),
         linkingGroup = "SAheart")
    ) -> h_chd
```

Figure 10: Histograms of `famhist`, an `Rfactor` with levels `{"Absent", "Present"}` (i.e. a barplot), and of `chd`, an `SAheart` variable having only `numeric` values 0 or 1.

`yshows` being `"frequency"`) and are added to the linking group `"SAheart"`. Selection, colouring, or deactivating in any one of the three linked histograms will now force the same on the other two in the group.

Case-control considerations

Consider the histogram of the response `chd` appearing on the right hand side of Figure 10. The relative height of the left-hand to the right-hand bars shows there to be roughly twice (actual ratio is 302:160) as many without heart disease as there are with. Note that this ratio does not reflect the relative proportions in the population. The data appear to have been collected as a retrospective case-control study where all 160 available case subjects (`chd` = 1) were first collected and then supplemented by collecting the 302 control subjects (`chd` = 0) who, presumably, were similar to the case subjects in other respects. There is not enough information in the sources (Hastie & Tibshirani, 1987; Hastie et al., 2009a, 2009b) to say more on how the 302 controls were selected from the more than 3,000 men recruited.

Nevertheless, the data in hand may still be profitably investigated.

Selecting the right bar of Figure 10 selects those with a family history of heart disease

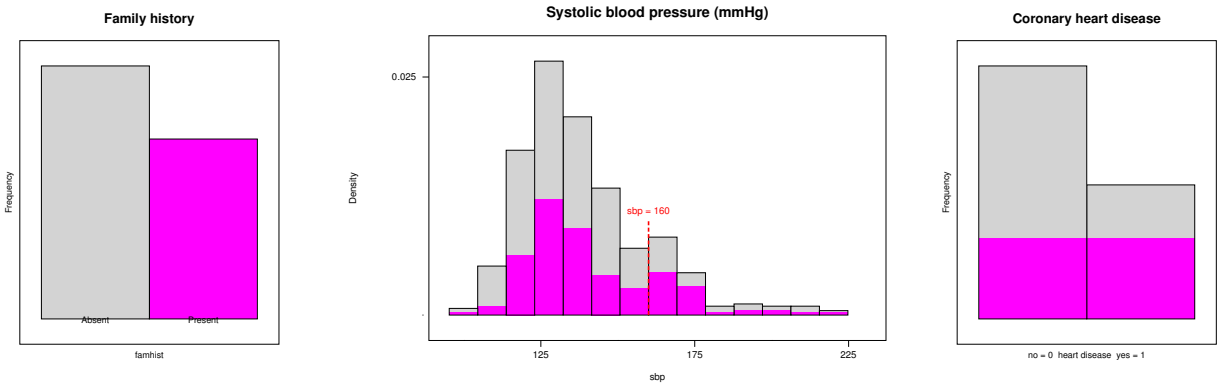


Figure 11: Select Present on Family history (left) to observe related Systolic blood pressure (middle) distribution and Coronary heart disease (right) proportions.

and propagates that selection to all linked plots as shown in Figure 11. Looking at the highlighted (darker or magenta coloured) part of the systolic blood pressure histogram (Figure 11; middle), the conditional histogram of `sbp` given a family history is not too different (in shape) from its unconditional histogram. A very different situation is seen in the histogram for the response coronary heart disease (Figure 11; right). There, conditional on having a family history, the number of men with heart disease is seem to be about the same as the number without (it is in fact identical). This 1:1 ratio is much larger than the unconditional 1:2 (approximately), which is still larger than the approximately 1:3 ratio of disease to no disease for those without a family history (i.e., the relative heights of the light grey bars). One suspects that controls may have been collected in some order until exactly as many had a family history as did the cases (viz., 96). Nevertheless, it would appear that the Afrikaner men were more likely to have heart disease if they had a family history than if they had not.

Odds ratios

A better analysis would examine the odds of having a family history compared to not having a family history, under each of the conditions: case or control. The ratio of these odds is equivalent to that of the odds of having coronary heart disease compared to not having coronary heart disease under the two conditions: having family history or not having family

history. The first odds ratio follows the logic of the data collection, providing a reliable measure of association, and is mathematically equivalent to the second odds ratio which admits a causal interpretation.

Selecting and deactivating the controls from all linked histograms conditions on the cases and produces the histograms in the top row of Figure 12. Of interest is the ratio of the bar

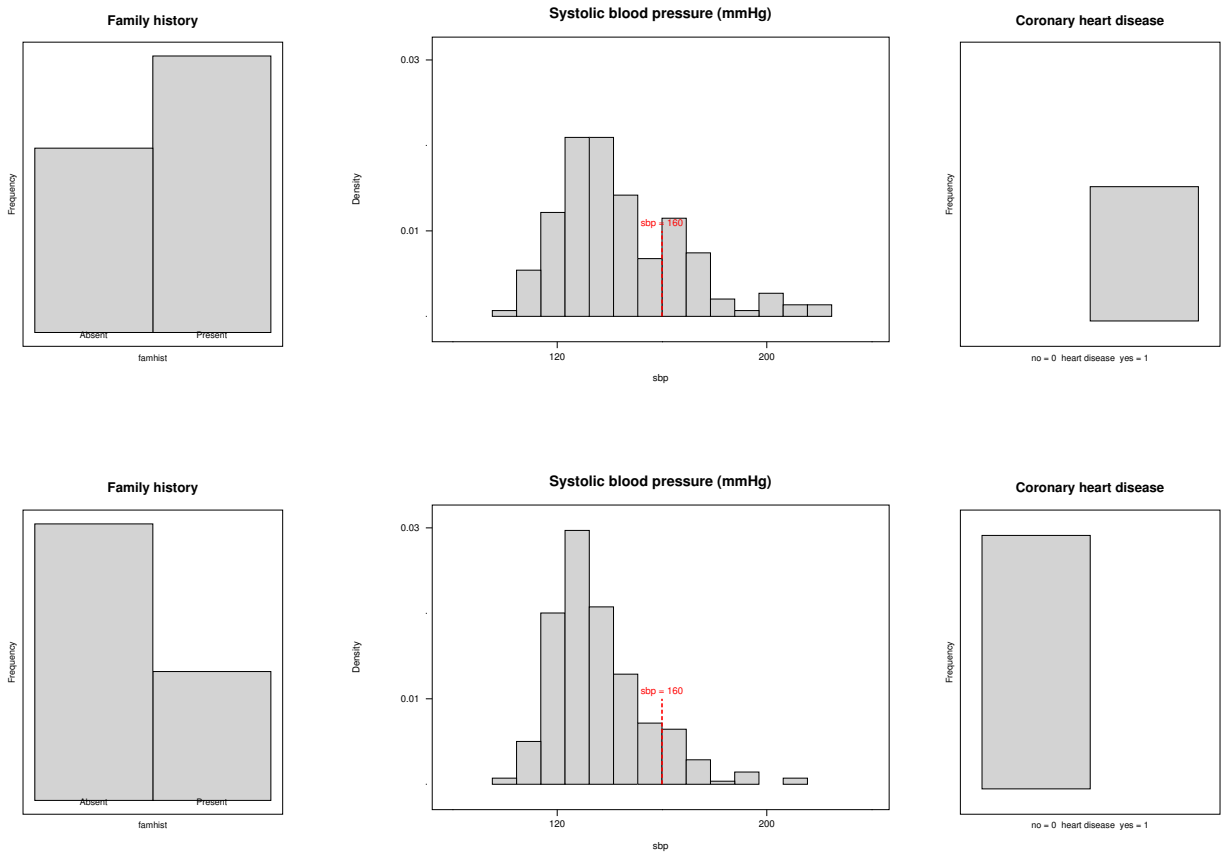


Figure 12: Case-Control Odds: Top row shows distributions of `famhist` and `sbp` for *cases* (`chd = 1`); bottom row the same for *controls* (`chd = 0`). Rightmost histograms are the unscaled numbers of cases and controls; leftmost can provide the appropriate odds ratios.

heights in the family history histogram at left where the ratio of `"Present"` to `"Absent"` is about 3 to 2. Similarly, the bottom row of Figure 12 conditions on the controls. The odds of interest are again the relative heights of the bars in the family history histogram where the ratio of `"Present"` to `"Absent"` is now about 1 to 2. The odds ratio of interest then is about 3:1. The same odds ratios arise in the coronary heart disease histogram of Figure 11 where conditioning is instead on the presence or absence of family history.

Conditioning on hypertensive status

A query might also be made of those who are hypertensive by selecting all systolic blood pressure histogram bars to the right of the mark at `sbp = 160` as in Figure 13. In the

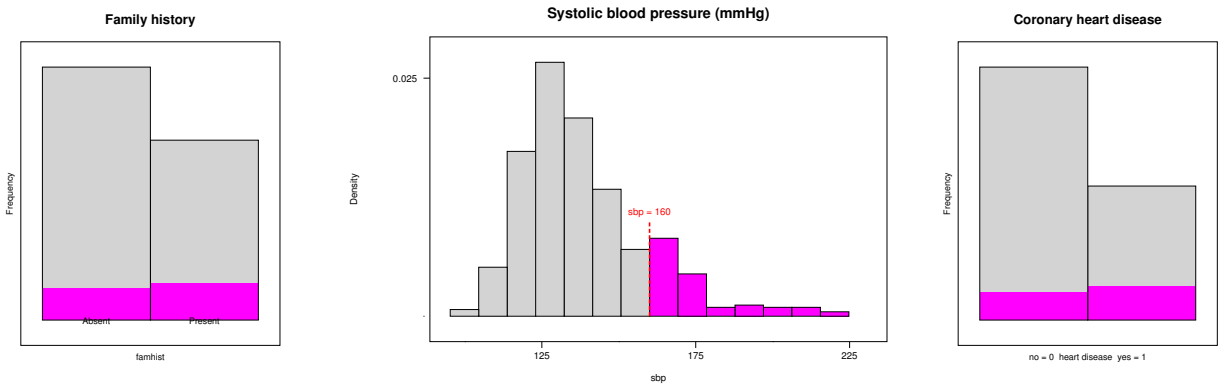


Figure 13: Select hypertensive subjects (`sbp > = 160`) on Systolic blood pressure (middle) to observe related Family history (left) and Coronary heart disease (right) change in proportions.

coronary heart disease histogram, the relative heights of the selected parts show that, among hypertensive men, more have heart disease than do not; similarly, the relative heights of the light grey parts show about half as many non-hypertensive men have heart disease as do not. Similar conclusions hold (with slightly differing ratios) from the family history histogram (Figure 13; left).

Logical queries via linked histograms

So far, conditioning has been on a single variate and had by either selection of the conditioning values or deactivation of their complement. The complement is a logical `NOT` operation effected by simply inverting the selection from the inspector. More complex logical queries can also be applied interactively.

For example, the logical inclusive `OR` is effected simply by multiple (or sticky) selection. In Figure 14, the logical query `(famhist == "Present") OR (sbp > 160)` done interactively by first selecting the `"Present"` bar in the family history histogram and then `<Shift>`-selecting all bars to the right of the mark `sbp = 160` in the systolic blood pressure histogram. The effect of this condition on the outcome variable can then be examined in the coronary heart

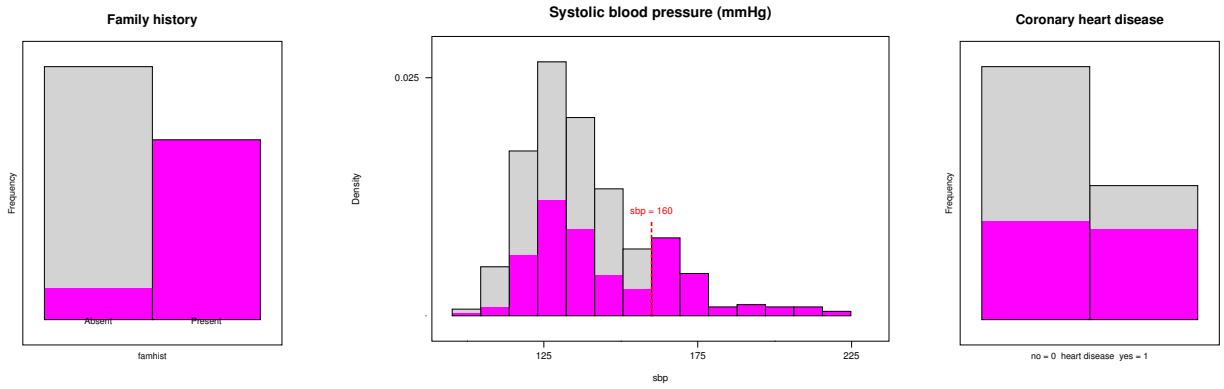


Figure 14: Inclusive OR by sticky selecting BOTH hypertensive subjects (`sbp > = 160`) on Systolic blood pressure (middle) AND Present on Family history (left) to observe any Coronary heart disease (right) change in proportions.

disease histogram.

Interactively constructing a logical `AND` query involves *both* selection and activation/reactivation. The query `(famhist == "Present") AND (sbp > 160)` can be constructed as follows. First *select* and *deactivate* all non-hypertensive subjects in the systolic blood pressure histogram. The same subjects will now be deactivated from all linked plots, including the family history histogram. Second, in the family history histogram *select* the bar identifying those with a family history. The selected portions of all linked plots of Figure 15 show those subjects which satisfy the query. The top row shows the construction of the query; the bottom row includes all subjects by reactivating the non-hypertensive via the inspector. Of interest is the relative heights of selected (or unselected) bar portions in the coronary heart disease histogram.

Of course much more complex queries can now be built up from the combined use of `NOT`, `OR`, and `AND`. These queries are constructed interactively using selection, inversion, deactivation, and activation which by their immediacy, facilitate the analyst quickly forming queries to probe different statistical features of the data in a highly exploratory way (e.g., as shown in the video Oldford, 2019). For a video showing more complex queries on this data set, see the video Oldford (2019); for a more general discussion on logical queries via interaction graphics Wills (2000). Again, queries may also be made programmatically using standard logical operations of R and plot states such as `"selected"`, `"active"`, and even

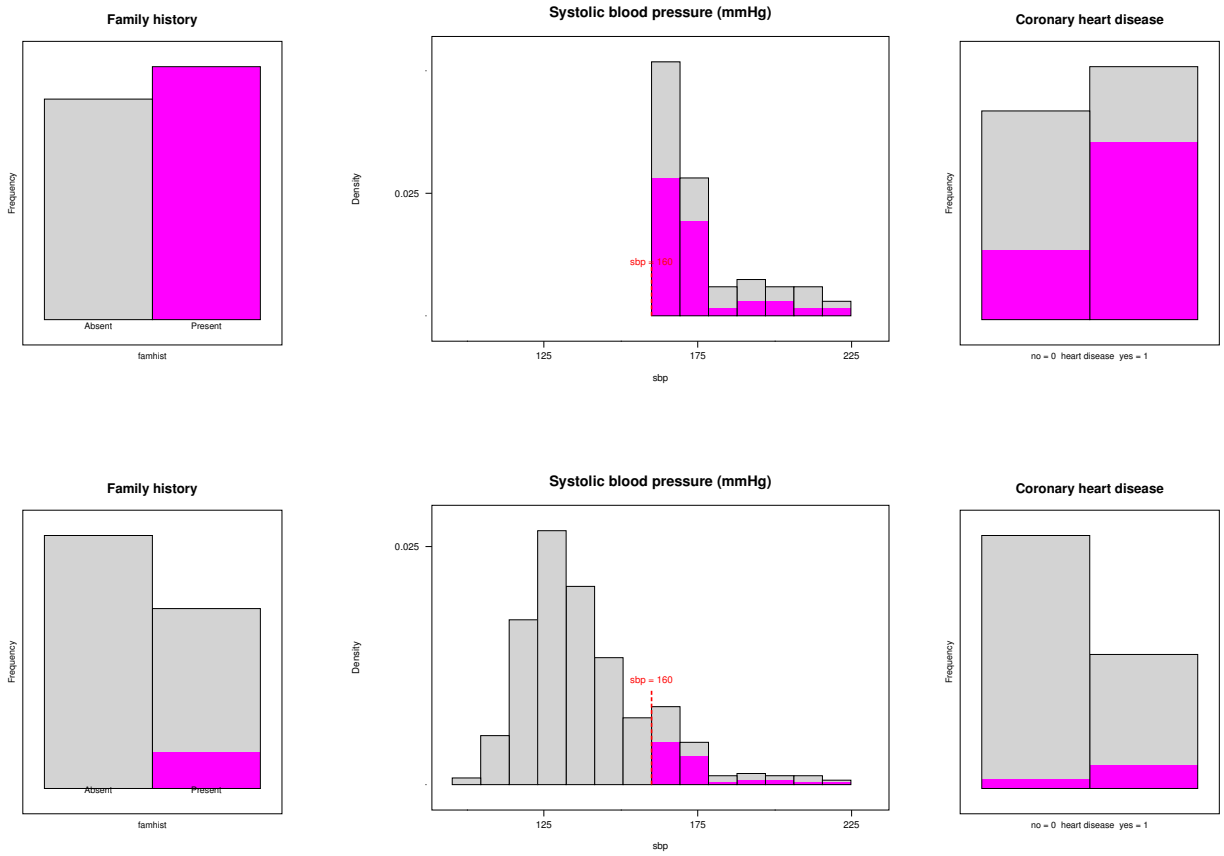


Figure 15: Selected subjects satisfy `(famhist == "Present") AND (sbp > 160)` . The top row emphasizes the construction. The bottom row is the same selection (now with all subjects) after reactivating the non-hypertensive. Conditional on this query, the proportions selected in the rightmost histogram could be examined to see the relationship with coronary heart disease.

`"color"`.

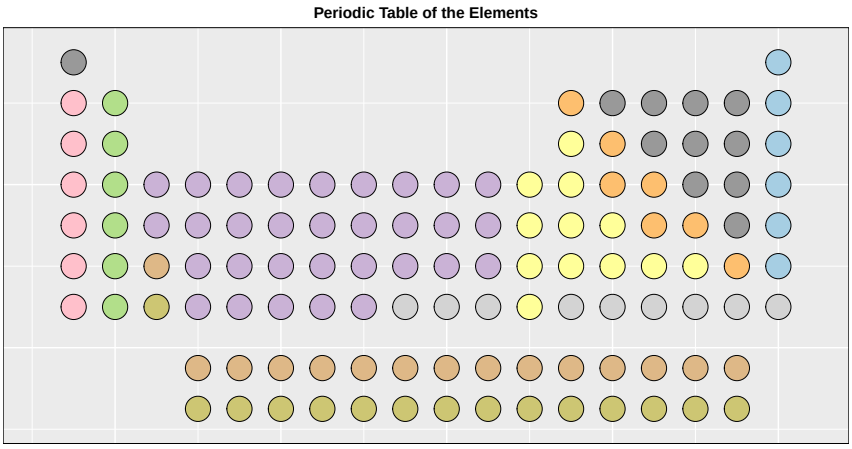
Example 3: The periodic table of the elements

Two other general plotting features of loon, namely *arbitrary user defined glyphs* for scatterplot points and *multiple displays or facets* of the data based on any logical condition, are now illustrated by creating an interactive periodic table of the elements.

The `elements` dataset from the R package loon.data contains the first 118 elements of the periodic table together with 17 separate measurements and descriptors of each element. These include standard information such as symbol, name, mass, atomic number, and so

on, as well as `x` and `y` coordinates to locate each element in the table. Other scientific measurements included are `Melting_point`, `Boiling_point`, `Specific_heat_capacity`, and others which may be missing for some elements.

With the `x` and `y` locations in `elements`, the table is easily created as a loon scatterplot with colours determined by distinct pairs of `Category` and `Subcategory` (see Figure 16). While



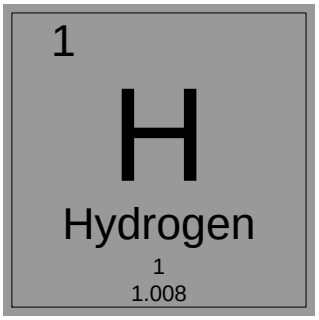

```
catSubcat <- paste(elements$Category, elements$Subcategory)

periodicTable <- l_plot(x = elements$x, y = elements$y,
                        size = 100, color = catSubcat,
                        xlabel = "", ylabel = "",
                        title = "Periodic Table of the Elements",
                        linkingGroup = "elements")
```

Figure 16: The periodic table as a scatterplot in loon.

the scatterplot visually suggests the periodic table, it is missing the standard information on each element as shown in Figure 17 for hydrogen, produced here using `draw_element_box()`. This function uses base R graphics functions to draw a standard periodic table glyph for any element.

Anything you can draw can be a point glyph in loon

To replace the glyphs in the scatterplot by whatever the analyst might wish to draw, loon provides two important functions: `l_make_glyphs()` and `l_glyph_add_image()`. The first of these takes a list of data objects and and a draw function as arguments. It returns a vector of images produced by applying the draw function to each data object. The second




```
draw_element_box <- function(symbol, name, number,
                             mass_number, mass, col) {
    oldPar <-  par(bg = col, mar = rep(1, 4))

    plot(NA, xlim = c(0,1), ylim = c(0, 1),
         axes=FALSE, ann = FALSE)
    text(0.5, 0.6, labels = symbol, cex = 18)
    text(0.15, 1, labels = number, cex = 6, adj= c(0.5,1))
    text(0.5, 0.25, labels = name, cex = 6)
    text(0.5, 0.11, labels = mass_number, cex = 3)
    text(0.5, 0.01, labels = mass, cex = 3)
    box()

    par(oldPar) }
```

Figure 17: The periodic table glyph for hydrogen and the function which draws it.

function, takes a vector of images and adds them to the menu of possible glyphs for the loon scatterplot. Once added to the scatterplot, the glyph of any selected point can be changed to its image glyph (via the inspector, or programmatically). With `draw_element_box()` as the draw function, proper element glyphs can assigned to each point with the resulting scatterplot `periodicTable` now appearing as in Figure 18.

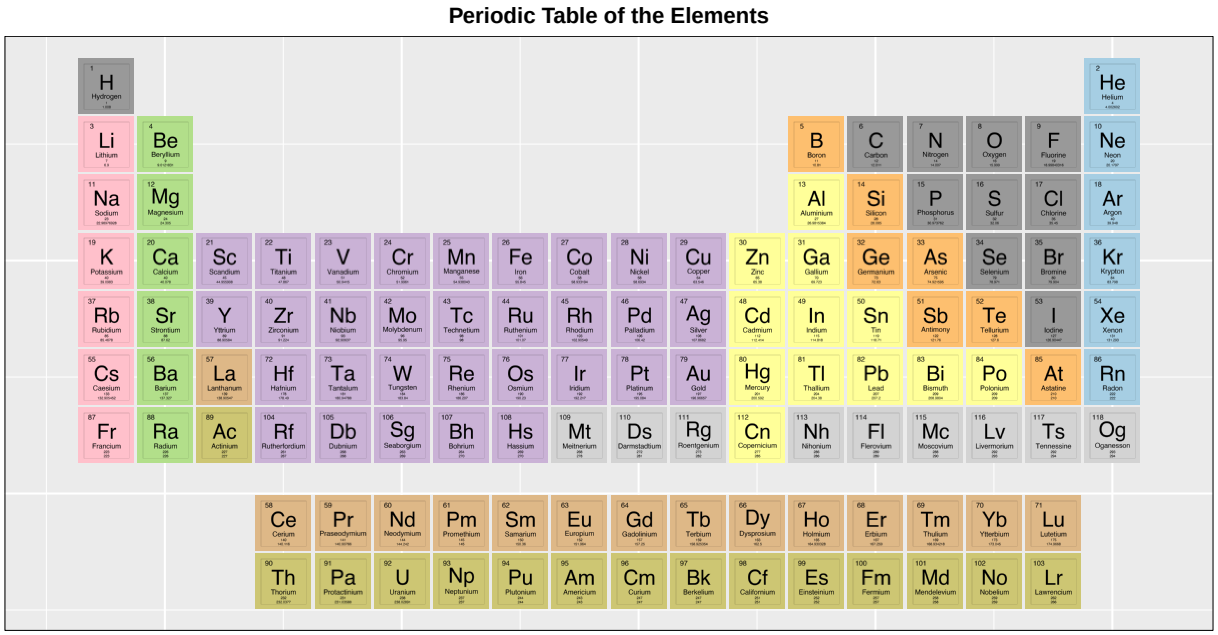
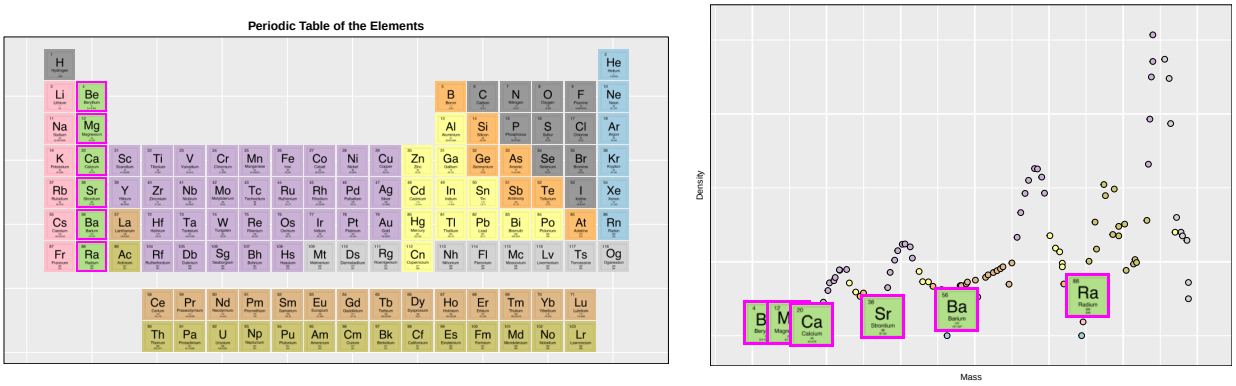


Figure 18: The periodic table as a loon scatterplot.

Again, the glyph image can be *anything that can be drawn* in R: stars, cartoon faces, maps, heatmaps, histograms, scatterplots, whatever.

Exploring the periodic table

Being an interactive scatterplot, the periodic table of Figure 18 can be linked to other interactive plots. The right hand scatterplot of Figure 19 shows the apparent periodicity



(a) Periodic table (`periodicTable`)

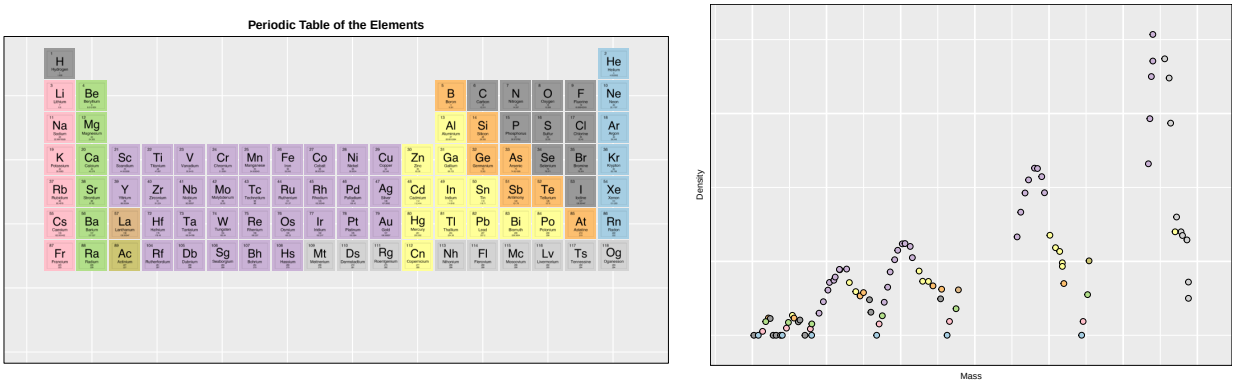
(b) Density versus mass (`cyclePlot`)

Figure 19: Linked scatterplots of the elements. The alkaline earth metals are selected in both (green points). These are replaced by element boxes in the right hand plot.

in the mass versus density of each element. As given by the default linked states, colours match points across the plots but glyph shapes do not. This permits choice between glyphs in the second plot without disrupting the periodic table; Figure 19 shows the element glyph only for the selected points.

Selecting or brushing points in one plot reveals structure in the other. Brushing rows in the table highlights the periodic cycles in the right hand plot. Selecting columns will identify consistent positions in the cycles. The second column in the periodic table is selected and appears with a magenta (highlighted) background in both plots.

Examining the right hand plot of Figure 19 reveals that cycle amplitude increases with mass and that there is some disruption in the regular form of the last two cycles (a relatively flat section and a relative noisy section, respectively). Brushing the disruptions reveals that these correspond to rare earth metals (i.e., the lanthanides and actinides). These appear as separate rows at the bottom of the periodic table. Figure 20 shows the effect of deactivating these two rows in the table. The rightmost scatterplot now reveals six smooth cycles (the last two slightly displaced) of increasing amplitude with mass.



(a) Periodic table (`periodicTable`)

(b) Density versus mass (`cyclePlot`)

Figure 20: The bottom two rows of the table (most lanthanides and actinides) are deactivated in (a) to reveal the last two (slightly displaced) cycles in (b).

Creating facets

Conditioning on the values of one or two variables has so far been effected by colour, selection, brushing, or active status. Alternatively, a separate plot called a facet could be shown for each value of the conditioning variable (e.g., as in Wilkinson, 1999; Wickham, 2009). In loon, facetted displays are created in two ways: from an existing loon display or *ab initio* at the time the loon plot is created.

Figure 21 shows the results of calling `l_facet()` with the argument `by = elements$Period` on each the two plots of Figure 19, 20 (n.b., after all points were first unselected and reactivated, and, in the second plot, element symbols were replaced by closed circles). The distinct values in the variable `Period` determine the facets and which points appear in each. Each facet is a separate loon plot of the same type and presentation as the original plot `periodicTable` (or `cyclePlot`). The name and values of the `by` argument produce labels to identify the facets.

Facet layout is controlled by the argument `layout` which can be one of `grid` (when the `by` argument consists of a list of variables), `wrap` (which wraps the plots in the display according to the number of columns given by the `ncol` argument), and `separate` (which places each facet in its own window). All facets are in the same linking group as the original plot and the scales (in terms of panning and zooming) may be constrained to work together in different ways depending on the argument `connectedScales`. For the facets of Figure 21, argument

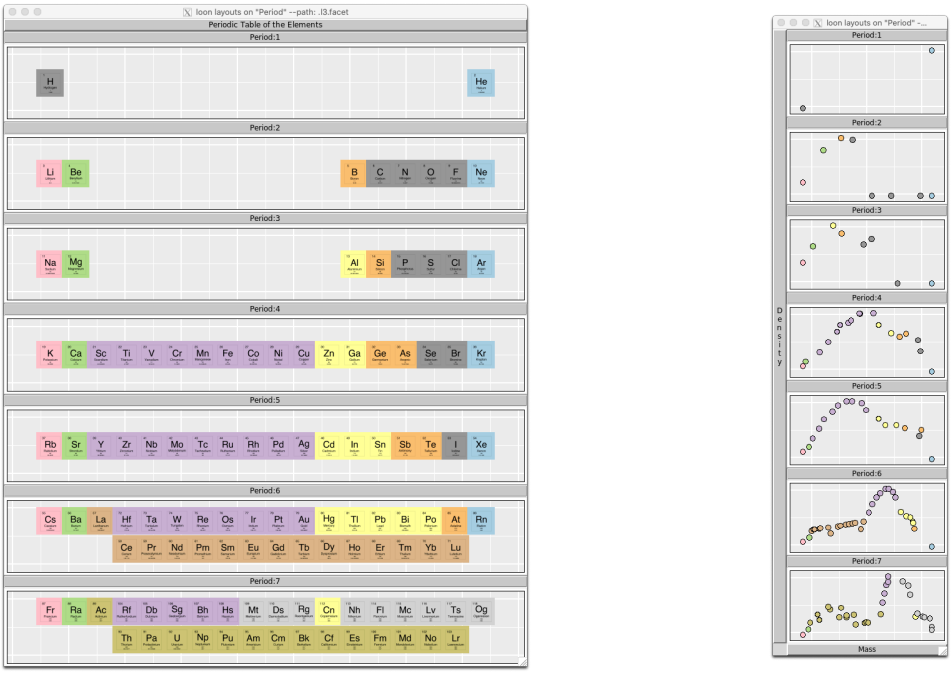


Figure 21: Facets by period: at left, of the periodic table; at right, of the cycle plot. In both cases, the facets within the plots are scaled to locations of the points they contain. Screenshots of windows are shown so that all labels appear.

values were `by = elements$Period`, `layout = "wrap"`, and `connectedScales = "none"`. The last of these means the facets of the cycle plot are each scaled to fill their plot, facilitating the comparison of the cycles by visually matching their amplitudes.

The same facetted plots could have been created directly using `l_plot()` (or any of the loon plot create functions) by supplying values for the arguments `by`, `layout`, and `connectedScales`. It would be a bit more work to construct them this way, since information the new element box glyphs would have to be added separately for each facet. In contrast, when `l_facet()` is called on an existing plot all of its components are available allowing `l_facet()` to migrate these from the existing plot to the facets automatically. This means, for example, that the `by` argument can also use the value of current states (e.g., `"color"` or `"glyph"`) for the facet variable.

On direct interaction in general

Direct interaction is a physical gesture such as a button press or a mouse movement. Programmatic interaction is writing and executing a script in the console. In any interactive statistical analysis, one wants both types of interaction (e.g., see Oldford, 1999).

As with writing code, getting the levels of abstraction right is equally important to providing methods of direct interaction. In gaming, for example, a multitude of accessible key interactions and mouse gestures lets the experienced gamer focus on the action, but might overwhelm a more occasional user. Conversely, having to consult different panels of options before every action reduces the mental overhead but at the cost of slowing the action, possibly to the point of losing the plot. Good game design depends on finding a good balance between the two extremes. Similar considerations need to be made for interactive data visualization.

In loon, the degree to which each interaction benefits from immediacy is balanced against increasing complexity by giving too many choices. For immediate direct interaction (the ideal), the physical gestures are few in number, simple enough to remember, and consistent across displays. Mouse button point-and-click and scrolling together with a few modifier keys provides an immediate and consistent interaction across different plots for selection, identification, panning, and zooming. The analyst's focus need never leave the plots during the interaction.

When a multiplicity of choices are required these are offered in the loon inspector focused on the plot. There, like choices appear near each other within titled boundaries (i.e., plot, select, and modify sections), tabbed panels organize distinct sets of plot interactions (e.g., analysis, layers, glyphs), and the general arrangements are consistent across inspectors and differ only in minor detail peculiar to each type of plot. Immediacy is slightly reduced as the analyst shifts focus between the inspector and its target plot. The omnipresence of the inspector together with its consistent arrangement should improve immediacy as the user becomes more experienced.

Between these two design features is the possibility of pop-up menus/option panels accessed via mouse interaction on the plot (e.g., see Oldford & Peters, 1988; Hurley & Oldford,

1991, for early examples) . The analyst's focus would only move slightly to make possibly complex changes on the plot. This seems a very good approach when changes are being made to a single plot. When there are multiple linked plots however, the objective is often to observe the effect of change on many plots simultaneously. In this case, having the choices be made on an inspector that is separate from all plots seems more appropriate. Pop-up menus or option panels are not currently provided in loon, mainly because to date they have not been needed.

The narrower the field of view containing its target, action, and effect, the more immediate the direct interaction. This is what makes panel displays, like pairs plots or layouts of facets, more effective than the same plots distributed over a screen – narrowing the field of view reduces visual travel, improving the immediacy of interaction.

Linking

Loon implements linking as a *group-key-state* model. Every plot in loon has its own *linking group*, its own *linking key*, and its own *linked states*. Any of these may be changed at any time, for any plot, and the plot values for the three components determine its linking behaviour.

Every plot having the same (string) value of their `linkingGroup` state will be linked. Each plot's `linkingKey` uniquely determines the identity of the plot elements to be linked, and the plot's `linkedStates` identify the visual features to be changed through linking.

Consider two plots, `p1` and `p2`, in the same linking group, say `"my group"`, and suppose that the plot state `"color"` is changed on three points in `p1`. If `"color"` is one of the linked states of `p1`, then `p2` (and every other plot in the linking group `"my group"`) will be informed of the change in `"color"` in `p1`. If `p2` also has `"color"` as one of its linked states, then the elements in `p2` corresponding to the three points of `p1` (whose `"color"` changed) will change their `"color"` to match. Which elements of `p2` correspond to the changed points of `p1` is determined by the value of the `linkingKey` of the three points of `p1`. If the `linkingKey` of a plot element of `p2` matches that of one of three points of `p1`, then the its `"color"` is changed to that of the matching point.

Functions `l_getLinkedStates()` and `l_setLinkedStates()` access and set a plot's linked

states. Each class of loon plot has a default set of linked states (e.g., `"color"`, `"selected"`, and `"active"` for a histogram; a scatterplot has the same plus a linkable `"size"` state as its default). Some classes of loon plot have additional optional linkable states (e.g., a scatterplot can also link the states `"glyph"`, `"itemLabel"`, and `"tag"`). The default linked states are intended to be all that most users/analyses will need. Optional linkable states, and the ability to changed which states are linked for each plot, provide more nuanced linking whenever the analyst chooses.

A plot's `linkingKey` is a character vector of the same length as its `x` state and is a *key* in the sense that its elements are distinct. The default value of the `linkingKey` is the character vector `c("0", "1", "2", ... , "n-1")`, with `n` being the length `x`. This key based design has two interesting consequences.

First, matching only on the linked key allows plots to be linked across different data sets. Indeed, the 1,000 observations in the `quakes` data, the 462 in `SAheart`, and the 118 in `elements` could all be linked by sharing linking keys and the same `linkingGroup`. Brushing points in the periodic table might then simultaneously highlight some of the first 118 points in a scatterplot of the `quakes` locations, and those parts of the histogram bars of `sbp` (i.e., systolic blood pressure) for the `SAheart` subjects. Wherever the `linkingKey` in each plot matched, the corresponding observations would be linked across the displays; neither the order, nor the number of observations, need match, as long as the same key is used in each plot. Of course, a sensible matching would assign the same linking key only to observations in each plot having the same identity (especially when from different data sets).

Second, missing values can be handled using each plot's `linkingKey`. Loon plots contain no missing values, so whenever a state argument has a row with a missing value, that row is removed from all state arguments including the `linkingKey` (e.g., at the time the plot is constructed or when it is facetted). Preserving the linked keys ensures all linked plots behave as expected, even when displaying intersecting sets of observations. It also means that, when programming, observations in a plot need to be identified by their linking key and not, say, by the row in which they appear in a data set.

The *group-key-state* model is a *one-to-one* linking model between plots; no value can be repeated within the `linkingKey` of a plot. However, provided the conceptual object is

identical across two plots but represented by many display elements in one and as a single display element in another, a *many-to-many* linking model might be more natural (e.g., Eick & Wills, 1995; Hurley & Oldford, 1991; Hurley, 2000). An obvious example would be when a country appears as many polygons in one display and as a single point in a linked scatterplot. The *group-key-state* model could accommodate this in particular instances by having a collection of display elements (e.g., polygons) represented by a single linking key.

A more general treatment, where, say, an additional level of indirection was introduced to provide a key hierarchy seems overly complicated for the benefit it might accrue to data exploration. For simplicity, then, the *group-key-state* model is the only linking model provided by loon. Instead, a number of other functions are offered in loon which can bind actions to standard loon plot events, such as state changes (e.g. `"selected"`), or to more general mouse and keyboard events over a loon plot. These event binding functions make it possible for the user to define their own interactions.

User defined interactions

There will always be analysis situations where the above suite of possible interactions fall short. An analyst might want some *ad hoc* interaction to expose patterns peculiar to the data in hand, or, perhaps a novel, as yet unimplemented, interactive display is conceived that could yield a powerful new data analysis tool if only it could be tried. Generic interactions, adaptable to a user's purpose, would enable such exploratory programming. In loon, a number of functions (each prefixed by `l_bind_`) can be used to *bind* an R function to an *event* on a plot or display. This *callback function* fires whenever there is a change in its bound event.

For example, in Figure 22 the `l_bind_state()` function *binds* the loon plot *states* of `cyclePlot` (from Figure 19, 20) to call the anonymous function given by `callback` whenever any of the states named by `event` change. Here `event = "all"` means *all* states of `cyclePlot` are so bound; a change to any one of them will fire the callback function. The `callback` function has (up to) three specific arguments `W`, `e`, and `b` (in that order), corresponding to the loon *widget* (`cyclePlot`'s unique string *path*, e.g., `".l0.plot"`), the *event* or state(s) that changed, and the state *binding* (a string, e.g.,`"stateBinding0"`) that uniquely identifies the

```
l_bind_state(cyclePlot,              # the plot
             event = "all",          # "all" states of cyclePlot
             callback =              # the callback function for event
                 function(W, e, b){ # widget W, event e, binding ID b

                                 # Message the user the argument values
                                 message("Binding ID ", b,
                                         " reports that states (", e, ")",
                                         " just changed on plot ", W)}
                     )
# Example state changes:
cyclePlot["showGuides"] <- FALSE
l_configure(cyclePlot, showScales = TRUE, showGuides = TRUE)

# Remove all existing state bindings of cyclePlot
l_bind_state_delete(cyclePlot, l_bind_state_ids(cyclePlot))
```

Figure 22: Binding a callback function to changes in plot states for `cyclePlot`

binding produced by the original `l_bind_state()` call.

The anonymous callback function in Figure 22 simply prints a message identifying the binding `b`, the changed states `e`, and the path string `W` of `cyclePlot` whenever any of the plot states of `cyclePlot` change. Examples shown first change only the plot state `"showGuides"` and then change both `"showGuides"` and `"showScales"` simultaneously using `l_configure()`. The last line in Figure 22 shows how to remove a state binding from a plot, in this case all state bindings of `cyclePlot`. The callback function in Figure 22 could have been anything function of the arguments `W`, `e`, and `b`, that the user desired.

A more interesting, and complex, illustration of state binding is shown in Figure 23. The intent here is to have the points in the `cyclePlot` identify themselves more clearly whenever they are selected by (optionally) magnifying their size and/or by changing their display glyph. Because the user would like to change the behaviour simply and at will, a named function will have to be used within the callback function; that way, the function value attached to the name can be changed without having to reset the binding of the callback function.

The code in Figure 23 begins with establishing the binding on `cyclePlot`. Whenever its points are selected or deselected (i.e., any change in the state `"selected"`), an anonymous function is called which in turn calls the named function, `changeGlyphs()` on the widget `W`.

```
# Bind a callback function (of only W) to selection
# Note that the function changeGlyphs() need not YET exist.
l_bind_state(cyclePlot,
             event = "selected",
             callback = function(W){changeGlyphs(W)})

# A function to create a function, to serve as changeGlyphs()
glyphChangeSelectedFn <- function(plot, glyphID, magnification = 1){
            # access features of the original plot
            originalGlyphs <- plot["glyph"]
            originalSizes <- plot["size"]
            # return the following function
            function(W){
                loonplot  <- l_getFromPath(W)
                sel <- loonplot["selected"]
                loonplot["size"][sel] <- magnification * originalSizes[sel]
                loonplot["size"][!sel] <- originalSizes[!sel]
                loonplot["glyph"][sel] <- glyphID
                loonplot["glyph"][!sel] <- originalGlyphs[!sel]
           }}

# Create a function and assign it to changeGlyphs
changeGlyphs <- glyphChangeSelectedFn(cyclePlot, "ctriangle", 5)
# When called on the plot, selected points
# become closed triangles magnified 5 times
changeGlyphs(cyclePlot)

# Simply change the definition of changeGlyphs
# to produce a different kind of change.
changeGlyphs <- glyphChangeSelectedFn(cyclePlot, "glyph0")
changeGlyphs(cyclePlot)
# now shows the periodic table glyph without magnification when selected
```

Figure 23: A more complicated (changeable) state binding that causes glyph types and/or sizes to change whenever selected.

Note that only the single argument `W` is used since that is all that `changeGlyphs()` appears to require; no need to include `e` and `b` because `W` is always the first argument passed to the callback. At this point, the function `changeGlyphs()` need not even have been created yet; the binding has access to the anonymous function as its callback.

The next function, `glyphChangeSelectedFn()`, creates and returns an anonymous function that could serve as the function `changeGlyphs()`. Looking at its definition, this anonymous function first gets the loon plot from the path identified by the widget `W`; this is necessary to be able to use such loon functionality as accessing and setting plot states with R's `[]`. The selected points are then identified as `sel` and used to set the magnification and glyph to be different for selected (`sel`) and unselected (`!sel`) on the plot. Note that, being defined within

`glyphChangeSelectedFn()`, this anonymous function will always have access to the original glyphs and their size as they were when it was created by `glyphChangeSelectedFn()` being called (i.e., within that closure).

Supposing the user wishes the points in `cyclePlot` to be magnified 5 times and appear as closed triangles whenever they are selected, they need only call `glyphChangeSelectedFn()` with the appropriate arguments and assign the result to `changeGlyphs` in order to achieve that effect. When points are not selected, they will return to whatever they were in `cyclePlot` at the time `glyphChangeSelectedFn(cyclePlot, ...)` was called. Calling `changeGlyphs(cyclePlot)` directly has the same effect as the binding. More interestingly, knowing that the periodic table symbols have been assigned to be `"glyph0"` in `cyclePlot`, the function `changeGlyphs` can be reassigned with another call to `glyphChangeSelectedFn()` to have the periodic table symbols appear as points in `cyclePlot` whenever they are selected (e.g., as in Figure 19).

Any plot state in loon can be bound to any callback function in R and all changes will propagate through the linked group where appropriate. This provides the user with a simple means to effect all sorts of new kinds of interaction.

Interactions need not be limited to loon plot states. Loon also offers the user the following types of binding:

- **item** (`l_bind_item()`) Used to evaluate callbacks at certain mouse and/or keyboard gestures (as events) on visual items on the canvas.

  Callback arguments: `W` (widget), `b` (binding id), `x` (x coordinate of mouse at event), and `y` (y coordinate of mouse at event).

  Example event: `"<ButtonPress-1>"` to indicate a left (or primary) mouse button press.

- **canvas** (`l_bind_canvas()`) Used to evaluate callbacks at certain events on the Tk canvas widget. Canvas bindings are Tk bindings for the canvas widget with one level of indirection to support loon's context-specific substitutions.

  Callback arguments: `W` (widget), `b` (binding id), `x` (x coordinate of mouse at event), `y` (y coordinate of mouse at event), `w` (plot width in pixels), and `h` (plot height in pixels).

  Example events: re-sizing of the canvas, entering the canvas with the mouse. Valid event specifiers as in Tk.

- **content** (`l_bind_layer()`, `l_bind_glyph()`, `l_bind_navigator()`,`l_bind_context()`) Used to evaluate callbacks whenever the collection of layers, glyphs, navigators, or navigator context, respectively, change for a display. Navigators are peculiar to a loon display for exploring high dimensional data interactively called a *navigation graph* and are not discussed here.

  Callback arguments: `W` (widget), `e` (event), and `b` (binding id).

  Example events: add, delete, relabel.

See Waddell (2016, Section 5.2) for details and examples.

In addition to the above bindings specialized to facilitate new interactions with loon plots, the user may use any of the event binding interactions of Tk, as accessed through the R package tcltk. In this way, looncan be extended to nearly any kind of new interactive functionality by the user.

Concluding Remarks

Except for facetting and the '`[]`' operator, all features of loon presented above are written in TclTk; with function names and syntax in TclTk similar to that appearing in the examples (e.g., see Waddell, 2016, for more details on the TclTk features). There are many other loon TclTk interactive displays that did not come up in the examples. These include a variety of interactive displays for multivariate data analysis, including parallel coordinate plots, radial axes (or radar or spider) plots, interactive graphs, and the kinematic exploration of high-dimensional spaces by navigating through low dimensional spaces (i.e., so-called "navigation graphs", see Hurley & Oldford, 2011; Oldford & Waddell, 2011). All this loon functionality would be available to any scripting language with an interface to TclTk.

The R implementation of loon is the most advanced version and the one best integrated with a complete data analysis system – it is available on CRAN (R Core Team, 2021) as the R package loon. The loon package has many more interactive displays, including pairs plots (or scatterplot matrices), interactive displays for time series (and their decompositions), and advanced navigation graph displays. As a part of the R ecosystem, loon provides added

interactive functionality for a number of other R packages including zenplots (for spatial layout of many displays, see Hofert & Oldford, 2020), loon.ggplot (providing interactive ggplots, see Xu & Oldford, 2021), and others. Many of these have been gathered together in the diveR ("direct interaction visual exploration in R", see Oldford, 2021) suite of loon and related packages.

Coming full circle, we recall ideas from Tukey and Wilk (1966), beginning with the quote:

> *Exposure, the effective laying open of the data to display the unanticipated, is to us a major portion of data analysis.* (Tukey & Wilk, 1966)

Exploratory data analysis engages the analyst with their data, encouraging a more intimate familiarity with the measurements and their relationships one to another. Patterns revealed may affirm that which was already known, or at least suspected, or, perhaps more importantly, even surprise the analyst and, so, provoke new lines of enquiry! Either way, time spent in exploration encourages, and gifts time for, thinking about the scientific problem at hand in the face of empirical evidence.

Acknowledgements

The authors thank Martin Gauch who contributed the code for `l_plot3D()` in loon, first as a student project, later as a contributor to loon, as well as the many students at the University of Waterloo who have used loon in courses over the many years of its development.

References

Donoho, D., Huber, P. J., & Thoma, H. M. (1981). The Use of Kinematic Displays to Represent High Dimensional Data. In W. Eddy (Ed.), *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface* (Vol. 13, p. pp. 274-278). Springer-Verlag.

Donoho, D. L., Huber, P. J., Ramos, E., & Thoma, H. M. (1982). Kinematic display of multivariate data. In *Proceedings of the 3rd Annual Conference NCGA* (Vol. 3, p. 393-400).

Eick, S. G., & Wills, G. J. (1995). High interaction graphics. *European Journal of Operational Research*, *81*(3), 445-459.

Flynt, C. (2012). *Tcl/Tk A Developer's Guide* (Third ed.). Waltham, MA: Morgan Kaufman.

Giardini, D., & Woodhouse, J. H. (1984). Deep seismicity and modes of deformation in Tonga subduction zone. *Nature*, *307*(5951), 505-509.

Halvorsen, K. B. (2015). R package ElemStatLearn: Data Sets, Functions and Examples from the Book: "The Elements of Statistical Learning, Data Mining, Inference, and Prediction" [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=ElemStatLearn (R package version 2015.6.26.2)

Hastie, T., & Tibshirani, R. (1987). Non-Parametric Logistic and Proportional Odds Regression. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, *36*(3), 260–276. Retrieved from http://www.jstor.org/stable/2347785

Hastie, T., Tibshirani, R., & Friedman, J. (2009a). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Second ed.). Springer New York. doi: 10.1007/978-0-387-84858-7

Hastie, T., Tibshirani, R., & Friedman, J. (2009b). *Webpage: Datasets for "The Elements of Statistical Learning"*. Retrieved December 15, 2019, from https://web.stanford.edu/~hastie/ElemStatLearn/

Hofert, M., & Oldford, W. (2020). Zigzag expanded navigation plots in r: The r package zenplots. *Journal of Statistical Software, Articles*, *95*(4), 1–44. Retrieved from https://www.jstatsoft.org/v095/i04 doi: 10.18637/jss.v095.i04

Hurley, C. B. (2000). Graphical Selection of Data Views. *Journal of Computational and Graphical Statistics*, *9*(3), 558–581. Retrieved from http://www.jstor.org/stable/1390946

Hurley, C. B., & Oldford, R. W. (1991). A software model for statistical graphics. In A. Buja & P. A. Tukey (Eds.), *Computing and Graphics in Statistics* (Vol. 36, pp. 77–94). Springer-Verlag.

Hurley, C. B., & Oldford, R. W. (2011). Graphs as navigational infrastructure for high dimensional data spaces. *Computational Statistics*, *26*(4), 585–612.

Oldford, R. W. (1999). Mental Models and Interactive Statistics: Design Principles. In *Computing Science and Statistics* (Vol. 31, p. 254-262). Interface Foundation of North America. Retrieved from http://www.math.uwaterloo.ca/~rwoldfor/papers/Interface1999/paper.pdf

Oldford, R. W. (2019). *The (not so) humble (interactive) histogram*. YouTube. Retrieved June 29, 2020, from https://youtu.be/JFJcg855HcQ

Oldford, R. W. (2021). diveR: Easily Install and Load Interactive Data Visualization Tools [Computer software manual]. Retrieved from https://great-northern-diver.github.io/diveR/ (R package version 0.1.1)

Oldford, R. W., & Peters, S. C. (1988). DINDE: Towards More Sophisticated Software Environments for Statistics. *SIAM Journal on Scientific and Statistical Computing*, *9*(1), 191-211.

Oldford, R. W., & Waddell, A. (2011). Visual clustering of high-dimensional data by navigating low-dimensional spaces. In *Proc. 58th world statistical congress* (p. 3294 - 3303). Dublin. Retrieved from http://2011.isiproceedings.org/papers/650370.pdf (Special Topics Session STS057)

Oldford, R. W., & Waddell, A. R. (2021). loon.data: Data Used to Illustrate Loon Functionality [Computer software manual]. Retrieved from http://great-northern-diver.github.io/loon.data/ (R package version 0.1.3)

R Core Team. (2021). The Comprehensive R Archive Network [Computer software manual]. Vienna, Austria.

Rossouw, J. E., Plessis, J. P. D., Benadé, A. J. S., Jordaan, P. C. J., Kotzé, J. P., Jooste, P. L., & Ferreira, J. J. (1983). Coronary risk factor screening in three rural communities: The CORIS baseline study. *South African Medical Journal*, *64*, 430-436.

Tukey, J. W., & Wilk, M. B. (1966). Data Analysis and Statistics: An Expository Overview. In *Proceedings of the Fall Joint Computer Conference* (p. 695-709).

Waddell, A. (2016). *Interactive Visualization and Exploration of High-Dimensional Data* (Unpublished doctoral dissertation). University of Waterloo.

Waddell, A., & Oldford, R. W. (2021). loon: Interactive Statistical Data Visualization [Computer software manual]. Retrieved from https://great-northern-diver

.github.io/loon/ (R package version 1.3.7)

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis* (First ed.). Springer New York.

Wilkinson, L. (1999). *The Grammar of Graphics* (First ed.). Springer New York.

Wills, G. J. (2000). Natural Selection: Interactive Subset Creation. *Journal of Computational and Graphical Statistics*, *9*(3), 544-.

Wyndham, C. (1982, June). Trends with time of cardiovascular mortality rates in the populations of the RSA for the period 1968 - 1977. *South African Medical Journal*, *61*, 987-993.

Xu, Z., & Oldford, R. W. (2021). loon.ggplot: Making 'ggplot2' Plots Interactive with 'loon' and Vice Versa [Computer software manual]. Retrieved from http://great-northern-diver.github.io/loon.ggplot/ (R package version 1.2.0)

FURTHER READING

The best way to become familiar with the concepts described above is to actually download the loon software and try it out. In the loon R packages there are numerous examples given of how to use loon, as well as several demonstrations available accessed by typing `demo(package = "loon")` in R. The entirety of the package is described online at https://great-northern-diver.github.io/loon/ where there also exist several articles and videos describing the use of loon. The initial base design is described in (Waddell, 2016).