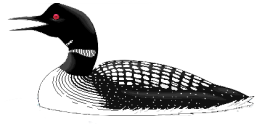
Introduction to loon

Rolling your own

. . . something old, something new, something borrowed, something blue . . .

Wayne Oldford
University of Waterloo

based on joint work with
Adrian Waddell
Roche

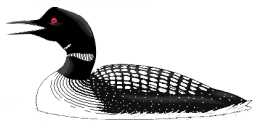


Pipes and loon - magrittr

One programming style that is sometimes convenient is that of pipes as developed in the package magrittr. These can also be used to great effect with loon.

For example,

```
# Get some data
library("ElemStatLearn")
data("SAheart")

# Pipes and data manipulation functions
library(magrittr)
library(tidyverse)

SAHealthy <- SAheart %>%
  filter(famhist == "Absent", chd == 0) %>%
  arrange(sbp)

SAHealthy %>%
  select(age, ldl) %>%
  l_plot(linkingGroup ="Healthy heart data") ->
  p
```



Pipes and loon - magrittr

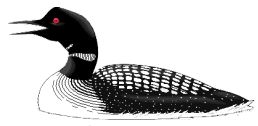
There is another pipe from magrittr, %T>% or "tee pipe", that can be handy.

Like %>%, the tee pipe %T>% pipes a value forward into a function or call expression but where %>% returns the result of the function (or call expression) %T>% instead returns the original value.

This can be handy in many instances, such as when piping through layers in loon.

```
SAHealthy %>%
  select(age, ldl) %>%
  l_plot(showGuides = TRUE, linkingGroup ="Healthy heart data")  %T>%
  l_layer_line(x=c(2,100), y=c(0,12),
               color="red", dash=c(20,10),
               linewidth = 2) %T>%
  l_scaleto_world() ->
  p
```

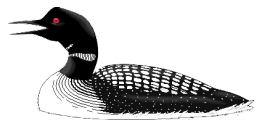
In the above, the data are passed along using %>% to l_plot() which produces the loon plot and %T>% passes this plot along throughout until it is finally assigned to p.



pipes and ggplot2

Many `tidyverse` users like to use `ggplot2` to define their graphics. For example,

```
SAheart %>%
    mutate(ltob = log(tobacco), lsbp = log(sbp)) %>%
    filter(age < 50) %>%
    ggplot(aes(x = ltob, y = lsbp)) +
    geom_point() +
    facet_wrap(~chd) -> p
```



pipes and ggplot2 and loon - `loon.ggplot` package‘

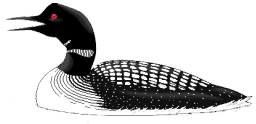
Wouldn’t this be cool?

```
library(loon.ggplot)

SAheart %>%
    mutate(ltob = log(tobacco), lsbp = log(sbp)) %>%
    filter(age < 50) %>%
    ggplot(aes(x = ltob, y = lsbp)) +
    geom_point() +
    facet_wrap(~chd) %>%
    loon.ggplot()
```

Coming soon!

Author: Zehao Xu

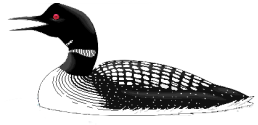
Warning: THIS FAILED LAST NIGHT . . . still under development



pipes and ggplot2 and loon - `loon.ggplot` package‘

For example, this does work

```
library(loon.ggplot)
p <- ggplot(quakes,
              aes(x = long, y = lat,
                  colour = factor(cut(quakes$depth, 100, label = FALSE)))) +
  geom_point(size = 4) +
  labs(x = "Longitude", y = "latitude")  +
  scale_colour_hue() +
  theme(legend.position="none") +
  facet_grid(~Depth)

loon.ggplot(p, linkingGroup = "quakes")
#
```



pipes and ggplot2 and loon - `loon.ggplot` package‘

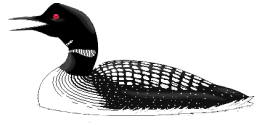
And even this

```
library(loon.ggplot)
library(lattice)
data(Oats, package = "MEMSS")
tp1.oats <- xyplot(yield ~ nitro | Variety + Block, data = Oats, type = "o")
tp1.oats

# The same in ggplot2
library(ggplot2)
pg.oats <- ggplot(Oats, aes(nitro, yield)) +
  geom_line() +
  geom_point() +
  facet_wrap(~Block + Variety, ncol = 3)
pg.oats

# from ggplot to loon
loon.ggplot(pg.oats)
```

We are getting close!

still under development . . . but looks good! Above bugs should be out in a week and the whole package on CRAN by summer’s end.

. . . stay tuned (also a `loon.micromaps` package coming from Alex Wang)

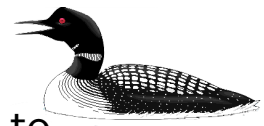


Demos

The `loon` package demos show how loon works and how you might create new functionality and displays with loon. Importantly, their source code is printed in the console!

```
library(loon)
demo(package = "loon")  # will produce
```

l_add_regressions	**interactively add regression lines of a particular order to selected points**
l_glyph_sizes	size mapping to various glyph types
l_glyphs	demonstrate glyph types
l_knn	*interactively highlight k nearest points in some subspace*
l_layers	demonstrate layer types
l_layout	**custom layout widgets**
l_linkPrimitiveGlyphs	*custom linking, link the primitive glyphs*
l_linking	*linking examples*
l_map	layer a map from the map R package
l_map_sp	layer a map with class sp as defined in the sp R package
l_ng_dimred	*compare results from various dimensionality reduction methods using navigation graphs*
l_ng_images_faces	navigation graph with olivetti faces data
l_ng_images_frey_LLE	navigation graph for the frey image data using LLE for dimensionality reduction
l_ng_images_frey_isomap	navigation graph for the frey image data using isomap for dimensionality reduction
l_power	**scatterplot and two scales that control the power transformation for each axis**
l_regression	layer fit, confidence, and prediction intervals
l_regression_influential	**move and recolor points to change the regression fit**
l_scagnostics	scatterplot maxtrix of scagnostic measures and a scatterplot that shows the scatterplot for the selected point in the scatterplot matrix
l_selectToActive	**two scatterplots; in one a regression is fit to the points that are selected in the other**
l_timeseries	seasonal trend decomposition stl
l_us_and_them	**gapminder data made famous by Hans Rosling; fertility, life expectancy, and income**
l_us_and_them_choropleth	**life expectancy on a world map and linked with a scatterplot**
l_us_and_them_slider	**show the life expectancy vs. fertility data for the year selected on a slider**
l_widgets	**custom layout**
loon	an introductory example



All demos ahow off loon functionality, *italic* demos show extensions or changes to functionality, and **bold** demos illustrate how you might create new functionality and

Build new things with tcltk

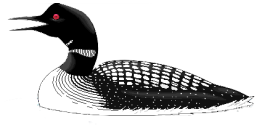
Because loon is built on the base R tcltk package (which ships with R), you can always build your own interactive displays using a mix of loon and tcltk functionality.

For example, a simple slider is built as

```
slider_window <- tktoplevel()
tktitle(slider_window) <- "slider"
slider_val <- tclVar('1') # default value
slider <- tkscale(parent = slider_window,
                  orient = "horizontal",
                  variable = slider_val,
                  from = -5,
                  to = 5,
                  resolution = 0.1,
                  command = function(...) print(paste0("Slider value = ", ...)))
(tkgrid(slider, row = 0, column = 0, sticky="we", padx = 50))
tkgrid.columnconfigure(slider_window, 0, weight=1)
```

Try it.

Try demo(l_power).



Binding events in loon

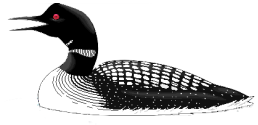
In a loon plot, it is easy to bind a function to fire whenever named states are changed.

The main function here is `l_bind_state(target, event, callback)`

For example

```
myPlot <- l_plot(iris, color = "Species")

makeItHappen <- function() {print ("It happened!")}

l_bind_state(myPlot, c("color", "xTemp", "yTemp"), makeItHappen)

WhoIsSpecial <- function() {
    cat("Selected points were: \n\t",
        myPlot["itemLabel"][myPlot["selected"]], "\n"
    )
    }

l_bind_state(myPlot, c("selected"), WhoIsSpecial)
```

Which opens up a whole range of possibilities.



Binding events in loon - a more realistic example

For example, we could fit a least-squares line to some data:

```
x <- 1:100
y <- 2 + 5 * x
data <- data.frame(x = x, y = y)
p <- l_plot(data$x, data$y)

lm_fit <- lm(y ~ x, data = data)
x_line <- extendrange(x)
y_line <- predict(lm_fit, newdata = data.frame(x = x_line))
fitted_line <- l_layer_line(p, x = x_line, y = y_line, color = "red")
```

And now have it change when points were moved.

Binding events in loon - a more realistic example

The updated least-squares line would be effected by a function like

```
updateLine <- function(myPlot, ls_line = NULL) {

    if (!is.null(ls_line) & ls_line %in% l_layer_ids(myPlot)) {
        # we'll update it.
        xnew <- myPlot["xTemp"]
        if (length(xnew) == 0) {
            xnew <- myPlot["x"]
        }

        # For y
        ynew <- myPlot["yTemp"]
        if (length(ynew) == 0) {
            ynew <- myPlot["y"]
        }

        # New fit
        new_fit <- lm(y ~ x, data = data.frame(x = xnew, y = ynew))
        x_line <- extendrange(xnew)
        y_line <- predict(new_fit, newdata = data.frame(x = x_line))

        # configure the least-squares line
        l_configure(ls_line, x = x_line, y = y_line)
    }

    # Update the tcl language's event handler
    tcl("update", "idletasks")
}
```

Binding events in loon - a more realistic example

The final step is simply

```
l_bind_state(p,
             c("xTemp", "yTemp"),
             function() {updateLine(p, fitted_line)})
```

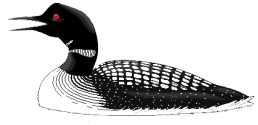
Try it!

A fuller teaching example might be as given in `demo(l_regression_influential)`.

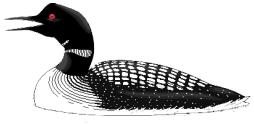
Try it!

Try to move points around in the plot titled "swiss data (least-squares)".

Finally, another complex example is given as a vignette file named "teaching-example-smoothing".



Thanks for your attention



loon - dive beneath the data surface to explore its depth

Loon is an open source project and we are always looking for collaborators and users.

Come on in . . . the water's fine!