In our [wrapr](https://winvector.github.io/wrapr/) [pipe RJournal article](https://journal.r-project.org/archive/2018/RJ-2018-042/index.html) we used piping into [ggplot2](https://cran.r-project.org/package=ggplot2) layers/geoms/items as an example.

Being able to use the same pipe operator for data processing steps and for ggplot2 layering is a question that comes up from time to time (for example: [Why can’t ggplot2 use %>%?](https://community.rstudio.com/t/why-cant-ggplot2-use/4372)). In fact the primary ggplot2 package author [wishes that magrittr piping was the composing notation for ggplot2](https://community.rstudio.com/t/why-cant-ggplot2-use/4372/7) (though it is obviously too late to change).

There are some fundamental difficulties in trying to use the [magrittr](https://cran.r-project.org/package=magrittr) pipe in such a way. In particular magrittr looks for its own pipe by name in un-evaluated code, and thus is difficult to engineer over (though it can be [hacked around](https://community.rstudio.com/t/why-cant-ggplot2-use/4372/3)). The general concept is: pipe stages are usually functions or function calls, and ggplot2 components are objects (verbs versus nouns); and at first these seem incompatible.

However, the wrapr [dot-arrow-pipe](https://winvector.github.io/wrapr/reference/dot_arrow.html) was designed to handle such distinctions.

Let’s work an example.

Suppose we want a single pipe notation to combine data processing and ggplot2 layer composition steps.

The wrapr dot-arrow-pipe performs data processing steps by explicit use of dot and sequencing of expressions. The dot-arrow semantics treats a %.>% b as being very much like {. <- a; b}. This means if the user writes an expression such as a %.>% b(.) the evaluation is very similar to b(a) (with some visible side-effects in the "." variable). [Explicit argument notation](http://www.win-vector.com/blog/2018/03/r-tip-make-arguments-explicit-in-magrittr-dplyr-pipelines/) is in fact a wrapr dot-arrow design principle, though wrapr dot arrow has some short-cuts and convenience methods (such as treating a %.>% f as a %.>% f(.) when f is bound to a function as in 5 %.>% sin). The wrapr dot-arrow-pipe concept is sequencing of expressions, which is related to (but not the same as) composition of functions.

With this in mind lets use wrapr dot-arrow-pipe for both data processing and ggplot2 layering.

First we load our packages.

library("ggplot2")

library("wrapr")

suppressPackageStartupMessages(library("dplyr"))

Now we tell wrapr what do do when the left-argument to a pipe is of class-gg. In this case we say treat a %.>% b as a + b (instead of the default {. <- a; b}). Notice this is an extension that any user or package can make, wrapr does not have special adaption for ggplot2. Instead it supplies sufficient control to be able to adapt to ggplot2.

apply\_left.gg <- function(pipe\_left\_arg,

pipe\_right\_arg,

pipe\_environment,

left\_arg\_name,

pipe\_string,

right\_arg\_name) {

pipe\_right\_arg <- eval(pipe\_right\_arg,

envir = pipe\_environment,

enclos = pipe\_environment)

pipe\_left\_arg + pipe\_right\_arg

}

Now we can run a single pipeline that combines data processing steps and ggplot plot construction.

data.frame(x = 1:20) %.>%

mutate(., y = cos(3\*x)) %.>%

ggplot(., aes(x = x, y = y)) %.>%

geom\_point() %.>%

geom\_line() %.>%

ggtitle("piped ggplot2")

Notice the data processing step mutate() and the initial ggplot() step *must* use ".", as that is how the dot-arrow-pipe normally moves data forward. The ggplot2 geom/layer/item pieces do not use ".". The evaluation rule is these items are evaluated as "pipe\_right\_arg" before seeing any of the pipeline to the left; this is roughly how ggplot2 handles composition through its override of "+".

And this is where we stopped the ggplot2 example in the [wrapr](https://winvector.github.io/wrapr/)‘s [pipe RJournal article](https://journal.r-project.org/archive/2018/RJ-2018-042/index.html). The article was about the wrap dot-arrow-pipe, and not about ggplot2 so it seemed important to move on from the example quickly.

A minor (undiscussed) technical difficulty is: ggplot2 users can store pieces of a plot in variables (or more correctly variable names my refer to pieces in environments). They may use this to write more modular plotting code. For example suppose the user stored the geom\_line() specification and the title in two variables as below.

line <- geom\_line(linetype = 2)

title <- ggtitle("piped ggplot2",

subtitle = "pre-stored title")

We can not, without a bit more adaption, pipe into these structures.

data.frame(x = 1:20) %.>%

mutate(., y = cos(3\*x)) %.>%

ggplot(., aes(x = x, y = y)) %.>%

geom\_point() %.>%

line %.>%

title

## Error: wrapr::apply\_right\_S4 default called with classes:

## gg, ggplot

## line gg, ggplot

## must have a more specific S4 method defined to dispatch

The above error message is intentional. When the right-hand side of a wrapr pipe is the name of an object (instead of an expression or function) wrapr uses a right-dispatch followed by an S4 dispatch to decide what to do. The error message here just means there is no registered right or S4 dispatch for the classes of objects seen. This is easy to fix by examining the classes of the objects and adding appropriate right dispatch methods. What we are trying to say: the original left dispatch was enough to deal with plots where all the layers are presented as un-evaluated function calls (the most typical way to user ggplot2), however layers/geoms/items already evaluated and stored in variables need additional adaption.

This detail is not fragility of the wrapr dot-arrow-pipe, but an important design distinction. Names and objects on the right side of the pipe can explicitly specify stand-in or surrogate functions or effects. The idea was already present in allowing 5 %.>% sin to be a shorthand for 5 %.>% sin(.). We have merely generalized the surrogate effect and added explicit control points.

To extend the ggplot2 effects to use stored values, we first examine the classes of the items we want rules for.

class(line)

## [1] "LayerInstance" "Layer" "ggproto" "gg"

class(title)

## [1] "labels"

Notice title is not of class gg, so it will need its own rule.

The new S3 right-dispatch rules are.

apply\_right.gg <- function(pipe\_left\_arg,

pipe\_right\_arg,

pipe\_environment,

left\_arg\_name,

pipe\_string,

right\_arg\_name) {

pipe\_left\_arg + pipe\_right\_arg

}

apply\_right.labels <- function(pipe\_left\_arg,

pipe\_right\_arg,

pipe\_environment,

left\_arg\_name,

pipe\_string,

right\_arg\_name) {

if(!("gg" %in% class(pipe\_left\_arg))) {

stop("apply\_right.labels expected left argument to be class-gg")

}

pipe\_left\_arg + pipe\_right\_arg

}

And, just for fun, let’s assign the wrapr-dot-arrow-pipe to the shorter pipe-name %.%. Unlike magrittr pipe the dot-arrow-pipe semantics do not depend on the name of the pipe.

`%.%` <- wrapr::`%.>%`

With these rules in place we can now build up a ggplot2 combining data processing, in-line steps, and stored layers/items.

data.frame(x = 1:20) %.%

mutate(., y = cos(3\*x)) %.%

ggplot(., aes(x = x, y = y)) %.%

geom\_point() %.%

line %.%

title