A faceting function with bare variable names in ggplot2 –

I’m working on a package of dplyr and ggplot2 functions that I can use on a day to day basis to plot data over time. I’ve done some basic experimenting with my preferred ggplot2 theme settings (highlandrthemes – it’s a thing), and the next stage is to get some commonly used plot structures sorted out so that I don’t have to go hunting around for code all the time.

As I’ve been getting to grips with tidy evaluation, I wondered if I could create a ggplot2 function that would allow me to input “bare” variable names instead of inputting them as a string in the function call. I’m not a huge fan of piping direct from dplyr straight into ggplot2 but I figured if I could suss out how to create these functions, I would have the ability to pipe or not as the case required. If I get my package working properly, there will be a master function that will use tidy eval for dplyr based data munging which will feed into ggplot2.

The main requirement is having a grouping / faceting variable defined so that I can have my plots facet\_wrap(ed) the way I want. As it turns out, the “~” in the facet\_wrap() call is a bit pesky, to say the least.

Here’s the (admittedly tortuous) set-up – this data should exist within the base R installation. In real life, the data I’m working with results in a plot with 90 facets. All the solutions here worked with my dataset, and the example data below:

library(dplyr)

library(ggplot2)

library(tidyr)

library(tibble)

data <- list(fdeaths,mdeaths,ldeaths) #time series data- needs prep

names(data)[1:3] <- c("fdeaths","mdeaths","ldeaths")

data <- as\_tibble(data)

startdate <- as.Date('1974-1-1')

data$date <- seq.Date(startdate,by = 'month',length.out = 72)

newdata <- tidyr::gather(data, key = key, value = value,-date)

newdata$value <- as.numeric(newdata$value)

manual\_facet\_plot <- function(df, datecol, y\_var,...){

p <- ggplot2::ggplot(df,ggplot2::aes\_(substitute(datecol),substitute(y\_var))) +

ggplot2::geom\_line(colour = "grey40", group = 1) +

ggplot2::geom\_point(colour = "grey40", na.rm = TRUE)

p

}

# works - but I want the facet\_wrap call inside the function

p <- manual\_facet\_plot(newdata,date,value) +

facet\_wrap( ~ key, ncol = 3)

p

The manual\_facet\_plot is a start, but for a really useful function, I don’t want to have to add the faceting line by hand every time I run it. So this was my second attempt – which didn’t work:

fail\_plot <- function(df, datecol, y\_var, group\_var,...){

p <- ggplot2::ggplot(df,ggplot2::aes\_(substitute(datecol),substitute(y\_var))) +

ggplot2::geom\_line(colour = "grey40", group = 1) +

ggplot2::geom\_point(colour = "grey40", na.rm = TRUE) +

ggplot2::facet\_wrap( ~ eval(group\_var), ncol = 3)

p

}

# fails miserably

#fail\_plot(newdata,date,value,key)

Normally my Google-Fu is pretty strong, but I couldn’t figure out how to do this (, if I quoted one variable,I kept getting “variable not found” style errors for the rest), and couldn’t find anything anywhere so I asked on the RStudio community site for some help. (Let’s not mention the reprex fiasco).

fail\_plot <- function(df, datecol, y\_var, group\_var = NULL, ...){

p <- ggplot2::ggplot(df,ggplot2::aes\_(substitute(datecol),substitute(y\_var))) +

ggplot2::geom\_line(colour = "grey40", group = 1) +

ggplot2::geom\_point(colour = "grey40", na.rm = TRUE)

if (!is.null(group\_var))

p <- p + ggplot2::facet\_wrap(as.formula(paste("~", group\_var)), ncol = 3)

print(p)

}

So, this now worked:

fail\_plot(newdata,date,value,"key")

And I was happy with that.  
But then it started to annoy me a bit that I had to enter a string / quote the grouping variable, so here are some other ways I found of achieving this.  
Big thanks to the boffins on Stack Overflow who posted these solutions.

**Base R – retro chic**

Here’s a nice one for starters (in this example I have coloured the lines and points, for no particular reason) :

plot\_lines <- function(df, x, y, group) eval.parent(substitute(

df %>%

ggplot +

geom\_point(aes(x = x,y = y,colour = group)) +

geom\_line(aes(x = x, y = y, group = group, color = group)) +

facet\_wrap( ~ group, ncol = 3 ) +

theme(legend.position = "none")

))

p <- plot\_lines(newdata,date,value,key)

p

Excellent! No need to quote the grouping variable.

Here’s another solution using a package that was totally new to me (formula tools)

**Using formula.tools**

library(ggplot2)

library(formula.tools)

tf\_wrap <- function(formula, faceting = NULL, data, print = TRUE) {

y <- rhs(formula)

x <- lhs(formula)

p <- ggplot(environment = parent.frame())

p <- p + geom\_point(aes\_string(x = x, y = y), data = data, colour = "grey40") +

geom\_line(aes\_string(x = x, y = y), data = data, colour = "grey40") +

theme\_minimal()

if (!is.null(faceting)) {

rhsfacet <- all.vars(rhs(faceting))

if (length(rhsfacet) == 1 & any(rhsfacet %in% '.')) {rhsfacet <- NULL}

p <- p + facet\_wrap(facet = (rhsfacet),ncol = 3)}

if (print) {print(p)}

}

tf\_wrap(date ~ value, faceting = ~ key, data = newdata, print = TRUE)

And finally, given that I’ve been tidily evaluating my dataframe and grouping variables in the run up to actually being able to produce my plot, here is an example that uses rlang:

**using rlang**

gtest <- function(df,x,y, group) {

x <- rlang::enquo(x)

y <- rlang::enquo(y)

group <- rlang::enquo(group)

p <- ggplot(df,aes\_(x = rlang::quo\_expr(x),

y = rlang::quo\_expr(y))) +

geom\_line(colour = "grey40", group = 1) +

geom\_point(colour = "grey40") +

facet\_wrap(group, ncol = 3)

#p <- p + highlandrthemes::theme\_highlandr() # custom theme that you won't have

p <- p + ggtitle(label = "thank you Stack Overflow",

subtitle = "ggplot with tidy eval and facet wrap - no strings")

p <- p + labs(x = NULL, y = NULL, caption = "")

p

ggsave("2018-03-04-rlang.png")

}

gtest(newdata,date,value,key)

Brilliant. The base R and rlang solutions are based on the answer here:

library(tidyverse)

set.seed(47)

my\_df <- data\_frame(month = sample(1:12, 1000, replace = TRUE),

category = sample(head(letters, 3), 1000, replace = TRUE),

approved = as.numeric(runif(1000) < 0.5),

converted = approved \* as.numeric(runif(1000) < 0.5))

plot\_lines <- function(df, x, y, group) {

x <- enquo(x)

y <- enquo(y)

group <- enquo(group)

df %>%

group\_by(!! x, !! group) %>%

summarise(conversion\_rate = sum(converted) / sum(approved)) %>%

ggplot(aes\_(x = rlang::quo\_expr(x),

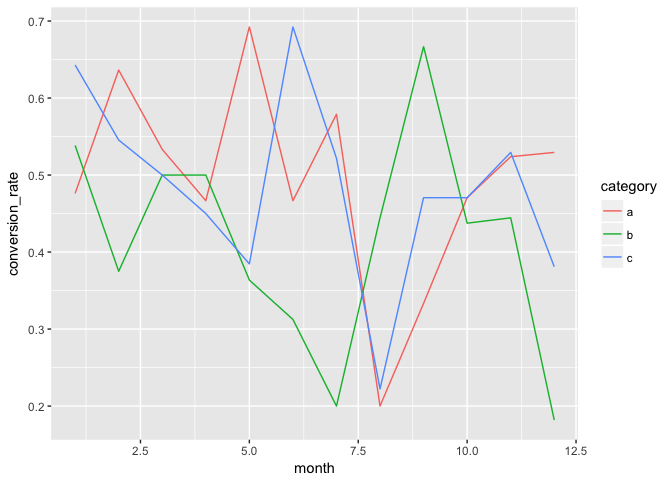
y = rlang::quo\_expr(y),

color = rlang::quo\_expr(group))) +

geom\_line()

}

my\_df %>% plot\_lines(month, conversion\_rate, category)



However, keep in mind that ggplot will almost inevitably be updated from lazyeval to rlang, so while this interface will probably keep working, a simpler, more consistent one will probably be possible shortly.

My specific use case is a function which takes a data frame and creates a ggplot object with simpler syntax. Here is an example of the code I want to automate with my function:

# create data frame

my\_df <- data.frame(month = sample(1:12, 1000, replace = T),

category = sample(head(letters, 3), 1000, replace = T),

approved = as.numeric(runif(1000) < 0.5))

my\_df$converted <- my\_df$approved \* as.numeric(runif(1000) < 0.5)

my\_df %>%

group\_by(month, category) %>%

summarize(conversion\_rate = sum(converted) / sum(approved)) %>%

ggplot + geom\_line(aes(x = month, y = conversion\_rate, group = category,

color = category))

I want to combine that group\_by, summarize, ggplot, and geom\_line into a simple function that I can feed an x, y, and group, and have it perform all the dirty work under the hood. Here's what I've gotten to work:

# create the function that does the grouping and plotting

plot\_lines <- function(df, x, y, group) {

x <- enquo(x)

group <- enquo(group)

group\_bys <- quos(!! x, !! group)

df %>%

group\_by(!!! group\_bys) %>%

my\_smry %>%

ggplot + geom\_line(aes\_(x = substitute(x), y = substitute(y),

group = substitute(group), color = substitute(group)))

}

# create a function to do the summarization

my\_smry <- function(x) {

x %>%

summarize(conversion\_rate = sum(converted) / sum(approved))

}

# use my function

my\_df %>%

plot\_lines(x = month, y = conversion\_rate, group = category)

I feel like the group\_by handling is pretty inelegant: quoting x and group with enquo, then unquoting them with !! inside of another quoting function quos, only to re-unquote them with !!! on the next line, but it's the only thing I've been able to get to work.

The formula.tools effort above was hacked from the second answer here :

I've also used environment = parent.frame() which works in my small testing

library(formula.tools)

tf <- function(formula, faceting = NULL, data, print = TRUE) {

y <- rhs(formula)

x <- lhs(formula)

p <- ggplot(environment = parent.frame())

p <- p + geom\_point(aes\_string(x = x, y = y), data = data)

if (! is.null(faceting)){

rhsfacet <- all.vars(rhs(faceting))

lhsfacet <- all.vars(lhs(faceting))

if(length(lhsfacet)==1 & any(lhsfacet %in% '.')) {lhsfacet <- NULL}

if(length(rhsfacet)==1 & any(rhsfacet %in% '.')) {rhsfacet <- NULL}

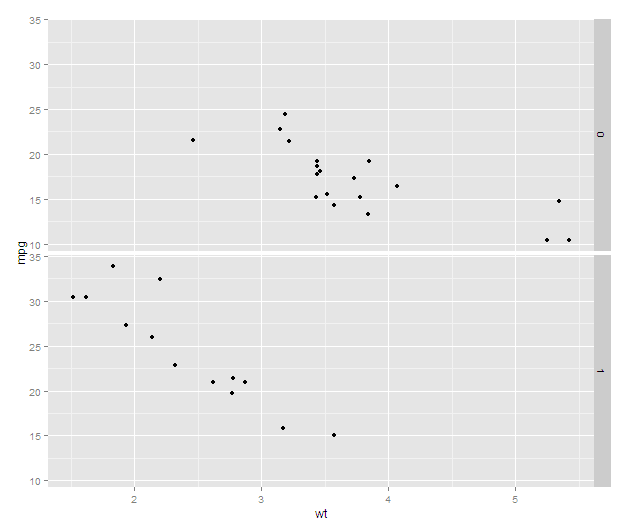
p <- p+ facet\_grid(facet = list( lhsfacet, rhsfacet))}

if(print) {print(p)}

p

}

tf(wt~mpg, faceting = ~am, data = mtcars, print = TRUE)



Interactively, this example works fine:

p <- ggplot(mtcars, aes(mpg, wt)) + geom\_point()

p + facet\_grid(. ~ vs)

Now, make a function with a formula interface and use aes\_string to do this same thing, and it doesn't work (error is: Error in layout\_base(data, cols, drop = drop) : At least one layer must contain all variables used for facetting):

tf <- function(formula, data) {

res <- as.character(formula[[2]])

fac2 <- as.character(formula[[3]][3])

fac1 <- as.character(formula[[3]][2])

# p <- ggplot(aes\_string(x = fac1, y = res), data = data)

# p <- p + geom\_point() # original attempt

p <- ggplot() # This is Joran's trick, but it doesn't work here

p <- p + geom\_point(aes\_string(x = fac1, y = res), data = data)

p <- p + facet\_grid(.~fac2) # comment this out, and it works but

# of course is not faceted

}

p <- tf(formula = wt ~ am\*vs, data = mtcars)