It is thrilled to finally be able to announce that patchwork has  
been released on CRAN. Patchwork has, without a doubt, been my most popular  
unreleased package and it is great to finally make it available to everyone.

Patchwork is a package for composing plots, i.e. placing multiple plots together  
in the same figure. It is not the only package that tries to solve this. grid.arrange() from gridExtra, and plot\_grid() from cowplot are two  
popular choices while some will claim that all you need is base graphics and  
layout() (they would be wrong, though). Do we really need another package for  
this? I personally feel that patchwork brings enough innovation to the table to  
justify its existence, but if you are a happy user of cowplot::plot\_grid() I’m  
not here to force you away from that joy.

The claim to fame of patchwork is mainly two things: A very intuitive API, and  
a layout engine that promises to keep your plots aligned no matter how complex  
a layout you concoct.

library(ggplot2)

library(patchwork)

p1 <- ggplot(mpg) +

geom\_point(aes(hwy, displ))

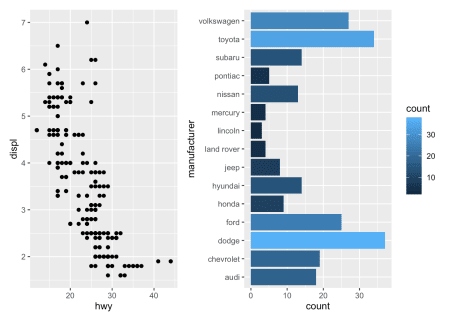
p2 <- ggplot(mpg) +

geom\_bar(aes(manufacturer, fill = stat(count))) +

coord\_flip()

# patchwork allows you to add plots together

p1 + p2



If you find this intriguing, you should at least give patchwork a passing  
glance. I’ve already written at length about all of its features at its  
[webpage](https://patchwork.data-imaginist.com/), so if you don’t want to  
entertain my ramblings more than necessary, make haste to the  
[Getting Started](https://patchwork.data-imaginist.com/articles/patchwork.html)  
guide, or one of the in-depth guides covering:

* [Assembling Plots](https://patchwork.data-imaginist.com/articles/guides/assembly.html)
* [Defining Layouts](https://patchwork.data-imaginist.com/articles/guides/layout.html)
* [Adding Annotation](https://patchwork.data-imaginist.com/articles/guides/annotation.html)
* [Aligning Across Pages](https://patchwork.data-imaginist.com/articles/guides/multipage.html)

**The Patch that Worked**

If you are still here, I’ll tell you a bit more about the package, and round up  
with some examples of my favorite features in patchwork. As I described in  
[my look back at 2017](https://www.data-imaginist.com/2017/looking-back-on-2017/)  
patchwork helped me out of burn-out fueled by increasing maintenance burdens of  
old packages. At that time I don’t think I expected two years to pass before it  
got its proper release, but here we are… What I don’t really go into is why I  
started on the package. The truth is that I was beginning to think about the new  
gganimate API, but was unsure whether it was possible to add completely foreign  
objects to ggplots, alter how it behaves, while still allowing normal ggplot2  
objects to be added afterwards. I was not prepared to create a POC of gganimate  
to test it out at this point, so I came up with the idea of trying to allow  
plots to be added together. The new behavior was that the two plots would be  
placed beside each other, and the last plot would still be able to receive new  
ggplot objects. It worked, obviously, and I began to explore this idea a bit  
more, adding more capabilities. I consciously didn’t advertise this package at  
all. I was still burned out and didn’t want to do anything for anyone but  
myself, but someone picked it up from my github and made a moderately viral  
tweet about it, so it quickly became popular despite my intentions. I often  
joke that patchwork is my most elaborate tech-demo to date.

All that being said, I was in search for a better way to compose plots (I think  
most R users have cursed about misaligned axes and butchered facet\_wrap() into  
a layout engine) and I now had a blurry vision of a solution, so I had to take  
it out of tech-demo land, and begin to treat it as a real package. But, along  
came gganimate and swallowed up all my development time. Further, I had hit a  
snag in how nested layouts worked that meant backgrounds and other elements were  
lost. This snag was due to a fundamental part of why patchwork otherwise worked  
so well, so I was honestly in no rush to get back to fixing it.

So patchwork lingered, unreleased…

At the start of 2019 I decided that the year should be dedicated to finishing of  
updates and unreleased packages, and by November only patchwork remained. I was  
still not feeling super exited about getting back to the aforementioned snag,  
but I saw no way out so I dived in. After having explored uncharted areas of  
grid in search of something that could align the layout engine implementation  
with not removing background etc. I was ready to throw it all out, but I decided  
to see how hard it would be to simply rewrite a subset of the layout engine. 1  
day later I had a solution… There is a morale in there somewhere, I’m sure —  
feel free to use it.

**The Golden Patches**

I don’t want to repeat what I’ve written about at length in the guides I linked  
to in the beginning of the post, so instead I’ll end with simply a few of my  
favorite parts of patchwork. There will be little explanation about the code  
(again, check out the guides), so consider this a blindfolded tasting menu.

# A few more plots to play with

p3 <- ggplot(mpg) +

geom\_smooth(aes(hwy, cty)) +

facet\_wrap(~year)

p4 <- ggplot(mpg) +

geom\_tile(aes(factor(cyl), drv, fill = stat(count)), stat = 'bin2d')

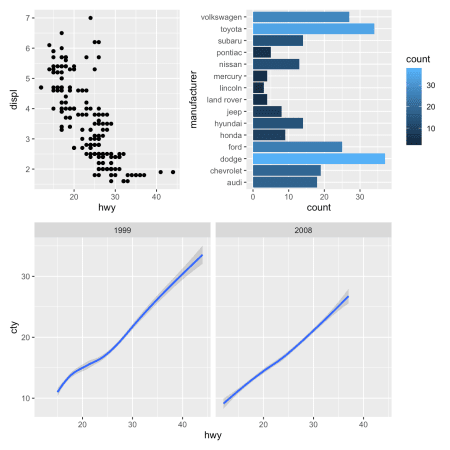
**Human-Centered API**

Patchwork implements a few API innovations to make plot composition both quick,  
but also readable: Consider this code

(p1 | p2) /

p3

It is not too difficult to envision what kind of composition comes out of this  
and, lo and behold, it does exactly what is expected:



As layout complexity increases, the use of operators get less and less readable.  
Patchwork allows you to provide a textual representation of the layout instead,  
which scales much better:

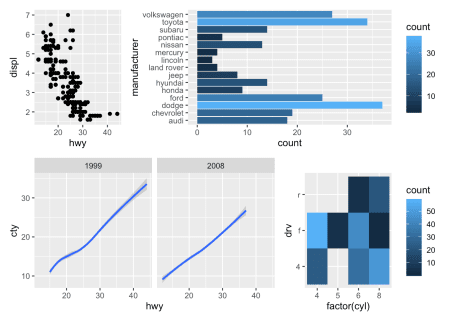
layout <- '

ABB

CCD

'

p1 + p2 + p3 + p4 + plot\_layout(design = layout)



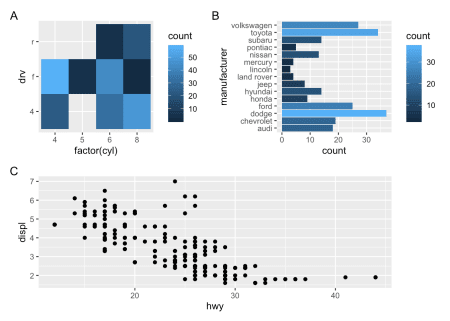
**Capable auto-tagging**

When plot compositions are used in scientific literature, the subplots are  
often enumerated so they can be referred to in the figure caption and text.  
While you could do that manually, it is much easier to let patchwork do it for  
you.

patchwork <- (p4 | p2) /

p1

patchwork + plot\_annotation(tag\_levels = 'A')

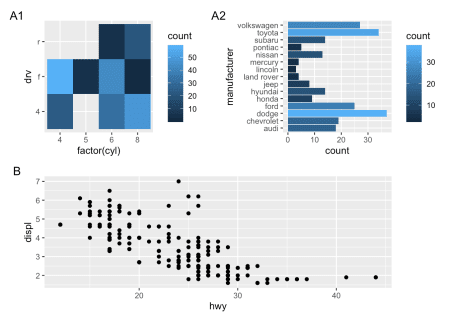


If you have a nested layout, as in the above, you can even tell patchwork to  
create a new tagging level for it:

patchwork <- ((p4 | p2) + plot\_layout(tag\_level = 'new')) /

p1

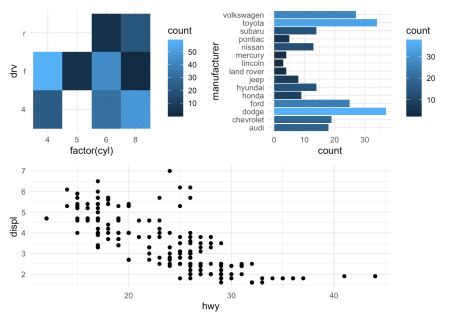
patchwork + plot\_annotation(tag\_levels = c('A', '1'))



**It allows you to modify subplots all at once**

What if want to play around with the theme? Do you begin to change the theme of  
all of your subplots? No, you use the & operator that allows you to add ggplot  
elements to all your subplots:

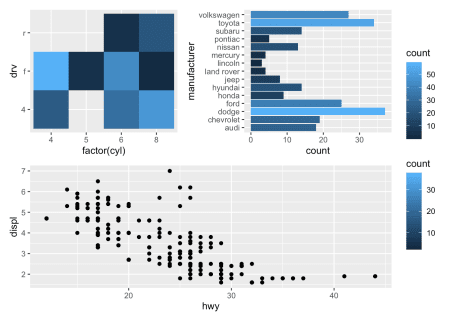
patchwork & theme\_minimal()



**It shepherds the guides**

Look at the plot above. The guides are annoying, right. Let’s put them together:

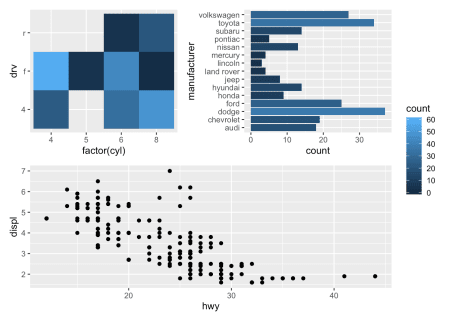
patchwork + plot\_layout(guides = 'collect')



That is, visually, better but really we only want a single guide for the fill.  
patchwork will remove duplicates, but only if they are alike. If we give them  
the same range, we get what we want:

patchwork <- patchwork & scale\_fill\_continuous(limits = c(0, 60))

patchwork + plot\_layout(guides = 'collect')



Pretty nice, right?

**This is not a grammar**

I’ll finish this post off with something that has been rummaging inside my head  
for a while, and this is as good a place as any to put it. It seems obvious to  
call patchwork a grammar of plot composition, after all it expands on ggplot2  
which has a grammar of graphics. I think that would be wrong. A grammar is not  
an API, but a theoretical construct that describes the structure of something in  
a consistent way. An API can be based on a grammar (as is the case for ggplot2  
and dplyr) which will guide its design, or a grammar can be developed in close  
concert with an API as I tried to do with gganimate. Not everything lends itself  
well to being described by a grammar, and an API is not necessarily bad if it is  
not based on one (conversely, it may be bad even if it is). Using operators to  
combine plots is hardly a reflection of an underlying coherent theory of plot  
composition, much less a reflection of a grammar. It is still a nice API though.

Why do I need to say this? It seems like the programming world has been taken  
over by grammars and you may feel bad about just solving a problem with a nice  
API. Don’t feel bad — “grammar” has just been conflated with “cohesive API”  
lately.

**Towards some new packages**

As mentioned in the beginning, It sets out to mainly finish off stuff in 2019.  
tidygraph, ggforce, and ggraph has seen some huge updates, and with patchwork  
finally released that reached my year goal with time to spare.