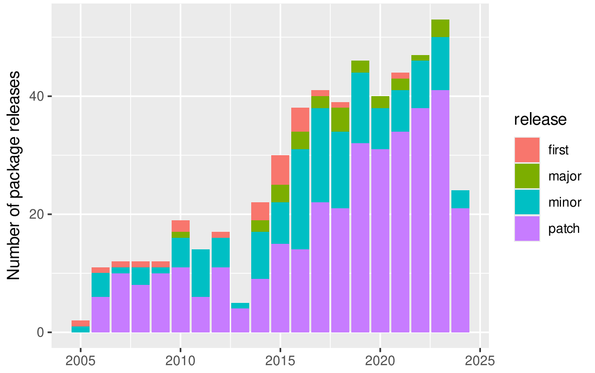
A brief history of the tidyverse

Hadley Wickham

2024-12-19

1 Introduction

Unlike our universe, the tidyverse did not start with a big bang. It start with a gradual accumulation of packages to make data science easy which eventually snowballed into the creation of the tidyverse. In this paper, I'll explore the process of its creation, starting from the influences that lead to the first packages of the proto-tidyverse (reshape and ggplot), leading into the early years of the tidyverse. I'll then discuss what exactly makes the tidyverse the tidyverse, discussing some of the ideas that make the packages work well together. I'll finish off with some thoughts about the current maturity of the tidyverse and the challenges of continuing to evolve a system that is now used by so many people.



2 Before the tidyverse

While the tidyverse itself was named in 2016, the work that lead to it started much earlier, and its inextricably connected to the course of my career. **? @fig-first-release** shows the releases of the tidyverse packages (and a couple of important precursors). From this diagram it's pretty clear that a lot of work happened before the tidyverse was named.

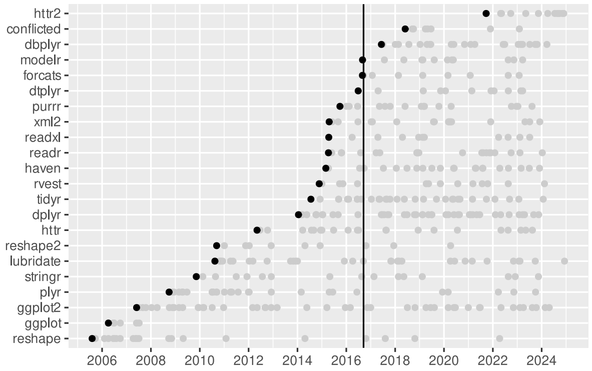


Figure 1: Initial releases of tidyverse packages and important precusors. A vertical line indicates the release of the tidyverse package.

2.1 Growing up

I was lucky that my dad, Brian Wickham, had access to a laptop for work, so always around computers. I gained an interest in MS Access and developed quite a few. That gave me an appreciation for databases.

From my mum I learned that you can choose to make a different in any area of life.

After high school, I immediately entered medical school. At the time in New Zealand, this consisted of two three year degrees and you were admitted directly from high school.

During my undergarduate degree I also got into web development and that was my part time job.

That ability to create websites has had a big impact on my ability to promote my work online.

I was an R user because I did my undergraduate at the University of Auckland, the birthplace of R, and so unsurprisingly I was an R user. Started using R in 2003, version 1.6.2.

2.2 PhD (2004-2008)

The inspiration for the tidyverse started during my PhD at Iowa State. I was lucky enough to get a consulting assistanceship with the Agricultural Experiment Station, where I helped PhD students in other departments do their analyses. This work lead me to two insights that remain with me today: the biggest challenge was almost never fitting the right statistical model; the biggest challenge was typically getting the collected data into a form that made sense to me for analysis. This challenge lead to the creation of the reshape pacakge which made it easier to work with a variety of input datasets by first converting to a "molten" form which you could then "cast" into the desired form.

(I took a brief detour through multi-dimensional arrays as a way of storing data, as they have some particularly elegant properties and can use much less memory than data frames. But they only work well for certain experimental designs and my experiences trying to teach the manipulation of even 4D arrays lead me to abandon this approach. You can see related work in https://juliasilge.github.io/widyr/ and https://github.com/hadley/cubelyr. Also a lot of work in plyr for converting between data frames, arrays, and lists. This work fell by the wayside once I decided that it was better to standardise on data frames, making it possible to focus tools around a single form. More on that below when we talk about tidy data.)

I also often found it hard to translate the plots that I could imagine in my head to code using either base or lattice (CITE) graphics. At the time I was reading the grammar of graphics (CITE) and really wanted to be able to use those tools. But the only implementation available at the time was very expensive, so I decided I'd have a go at creating my own in R. That lead to ggplot. I was very lucky to get the opportunity to meet Lee Wilkinson who was tremendously supportive on my work. (I also remember a throw away comment he made to me about data reshaping that lead to a vastly more performative implementation that become the heart of reshape2.)

This work wouldn't have been possible without the support of my PhD advisors Di Cook and Heike Hoffman, who let me work on what I thought was most important, regardless of whether or not it fit the mold of a traditional statistics PhD. They also provided aircover for me when I let my disdane for the utility of theoretical statistics shine a little too clearly.

(At this time I was also working with ggobi, a tool for interactively exploring high-dimensional data. Lead to a number of R packages including clusterfly and classify that used the rggobi package to get your data from R and into ggobi. This was really cool stuff, but unfortunately it's hard to use it today because of the challenges of maintaining GUI tools across multiple decades. You can see the continuation of this work in the work of Di Cook and students.)

This work culminated in my PhD thesis: "Practical tools for exploring data and models". http://had.co.nz/thesis/practical-tools-hadley-wickham.pdf.

2.3 Rice University (2008-2012)

The next major formative experience for the tidyverse was teaching Stat405 "Introduction to data analysis" at Rice. I taught this class four times (2009-2012) and you can still find my final class website at http://stat405.had.co.nz. Repeatedly teaching a mixed class of 60 grads and undergrads was an extremely useful experience as it allowed me to discover specific pain points and then in the following year attempt to solve them with better packages. This lead to the creation of the stringr (2009) and lubridate (2010) packages as I discovered that many students struggled to master the many intricacies of string and date-time manipulation provided by base R.

lubridate with Garrett, part of his PhD thesis. https://www.jstatsoft.org/article/view/v040i03

This also catalysed my work on tidy data. Although the paper wasn't published until 2014, I developed and taught the principles of tidy data in 2012. This was a big improvement over my previous attempts to teach reshape2 because reshape2 didn't declare any form of data more or less useful. But it was very clear to me that there was one form that was substantially easier to work with for most R packages, and that was what we now call tidy data.

My teaching also revealed the challenges of performing group-wise operations in base R. There were a number of interesting questions that seemed easy to formulate but hard to solve, like find the most popular male and female baby names in each year. http://plyr.had.co.nz (good example of a pre-pkgdown packge website. Looking back at this I know that this was more of a website than many packages provided, but is this really how I expectd people to learn a package?)

Very little success with grants. BD technologies funded ggplot2, plyr, and ggplot2. Funding from google to work on future of plyr.

> The generous support of bd.com has allowed me to implement many performance improvements to plyr, reshape and ggplot2, and begin work on the next generation of interactive graphics. Without such support, it is difficult for me to spend time on these projects as they do not directly contribute to my research portfolio. Your support not only gives me the financial backing to pursue these important optimisations, but also sends a strong signal to the statistics community that this work is important. The sections below describe what I have achieved this summer as a result of your funding.

(It was around this time that I kicked off another major parallel work stream: tooling for package development. I was developing enough packages that investing in tooling made sense, and this lead to devtools (2011), testthat (2009), roxygen2 (2011) taking over from Peter Danenberg (who created it as a Google Summer of Code project in 2008?), pkgdown (2012).)

In 2012, started working with Winston Chang. Garret and Winston were the first external contributors to packages that are now in the tidyverse and I'm still lucky enough to have them as my colleagues.

2.4 RStudio (2012-)

In 2012, I left Rice for RStudio, moving to a position where the practice of software engineering was valued and I no longer needed to produce papers (although I was still welcome to if I wanted). This gave me the time and freedom to learn C++, an important tool that I was missing for writing high performance code. Mentoring from JJ Allaire was tremendously useful at rapidly improving my C++ skills. Overall, my first few years at RStudio lead to a precambrian explosion of packages because I had both the time to work on what I thought was important and the ability to invest in core skills (like C++ programming) that were not valued in academia.

Particularly important was my work on dplyr. dplyr grew from my dissatisfation with the plyr package for solving grouped data frame problems-For example, to figure out the rank of each name for each year, for each sex, you had to write plyr code like this:

```r
library(babynames)

ranked <- plyr::ddply(babynames,
c("year", "sex"), function(df) {
  plyr::mutate(df, rank = rank(-n))
})
head(ranked, 10)
```

```
   year sex       name    n       prop
rank
1   1880    F       Mary 7065 0.07238359
1
2   1880    F       Anna 2604 0.02667896
2
3   1880    F       Emma 2003 0.02052149
3
4   1880    F Elizabeth 1939 0.01986579
4
5   1880    F     Minnie 1746 0.01788843
5
6   1880    F   Margaret 1578 0.01616720
6
7   1880    F        Ida 1472 0.01508119
7
8   1880    F      Alice 1414 0.01448696
8
9   1880    F     Bertha 1320 0.01352390
9
10 1880   F     Sarah 1288 0.01319605   10
```

Here we're using `ddply()` because the input is a data frame and we want the output also to be a data frame. We then describe how we want to split the input up (here by year and sex), and then what we want to do to each piece. (I've also avoided attached plyr as it has some unfortunate conflicts with dplyr that will break my code later in the article. If I was to work on this again today, I would take much greater pains to avoid such conflicts.)

This code is unappealing when teaching students new to R because I think the question is very accesssible but the code is not: you not only have to understand functions but also the basics of functional programming. This meant that it couldn't be taught until later in the semester, even though I think the basic ideas are relatively straight foreward.

The creation of dplyr meant that you could instead write code like this:

```
library(dplyr)

babynames |>
  group_by(year, sex) |>
  mutate(rank = rank(desc(n)))
```

```
# A tibble: 1,924,665 × 6
# Groups:   year, sex [276]
     year sex   name           n   prop
rank
    <dbl> <chr> <chr>      <int>  <dbl>
<dbl>
 1   1880 F     Mary        7065 0.0724
1
 2   1880 F     Anna        2604 0.0267
2
 3   1880 F     Emma        2003 0.0205
3
 4   1880 F     Elizabeth   1939 0.0199
4
 5   1880 F     Minnie      1746 0.0179
5
 6   1880 F     Margaret    1578 0.0162
6
 7   1880 F     Ida         1472 0.0151
7
 8   1880 F     Alice       1414 0.0145
8
 9   1880 F     Bertha      1320 0.0135
9
10   1880 F     Sarah       1288 0.0132
10
# i 1,924,655 more rows
```

This does require learning some new ideas, like the pipe and grouping, but in my experience students picked these up much faster than functional programming. And thanks to the hard work of Romain François dplyr ended up being much faster than plyr too: so much faster that one of my advertising claims was that learning dplyr would pay off immediately in computational time solved.

(It was unfortunate that I wasn't aware of data.table at this time, or I would have had much higher expectations for performance. I also inadvertently damaged my relationship with that commmunity by failing to understand what drove them and why dplyr was seen as such a threat. Fortunately in the years since I have worked to repair those relationships and I'm proud that dtplyr, the dplyr backend that uses data.table, recieved a data.table seal of approval.)

My work on dplyr also lead to the development of the purrr package (2015). It's hard for me to describe exaclty what purrr is, except that it provides a bunch of useful and consistent functional programming tools. purrr was an update to plyr. plyr supported four main data structures: lists, data frames, and arrays. recognising that the data frame side of plyr was handled by dplyr, and arrays to not be that useful, extract out the list handling code into purrr.

Over this time I expanded my scope to also consider getting data into R. In 2013, I took over maintenance of DBI and RSQLite from Seth Falcon in order to. RMySQL (2014? Jeroen?). I developed bigrquery in 2015 to make it easier to connect to Google's BigQuery database and also forked RPostgres from the mostly unmaintained RPostgresSQL and had several particularly annoying issues related to secure access. Funded by the R consortium. Much of this work was done in concert with Kiril Mueller, who now maintains this ecosystem of tools. I still remember having the strong sense of realising that a lot of R users had data in excel files and no easy and reliable way to get into R. This build on early work providing tools for web scraping (rvest, 2014), and lead to the development of readxl (2015). I also worked on packages for reading flat text file like csv and fixed width files (readr, 2015), SPSS, SAS, and Stata files (haven, 2015), and XML files (xml2, 2015). None of these packages would have been possible without my new found C++ skills, as they all relied on tight integration with existing C libraries for reading those data types.

I also created tidyr (2014), an update of reshape2, during this period. I'll come back to that later when I talk more about tidy data.

This also felt like the time where I started to become particularly well known in the R Community and did a couple of internet Q&A sessions on reddit (2015) and quara (2016).

3 Naming and defining the tidyverse

As the collection of packages I had developed grew, the community needed some name to collectively refer to them, and many people started calling them the "Hadleyverse" (e.g. https://github.com/imanuelcostigan/hadleyverse). I found this name unappealing because by this point the packages weren't just my work, and I found it overweeningly arrogant to refer to my own oeuvre as the "Hadleyverse" (a name I still can't say or write without making a face).

So I started the process of coming up with an official name that I could live with. Unfortunately I can no longer find the discussion, but alternatives included the sleekverse, the dapperverse, and the deftverse. Given the success of tidy data, I eventually decided that the tidyverse was the most natural name, and announced this to the world at useR 2016, June 29.

Shortly afterwards (in September), I released the tidyverse package. Two goals: make it easy to install all the dependencies and make it easy to load the most common packages. Core packgaes: ggplot2, dplyr, tidyr, readr, purrr, tibble. tidyerse 1.2.0 (September 2017) added forcats and stringr. tidyverse 2.0.0 (March 2023) added lubridate.

Giving the tidyverse a name created a bigger question. What exactly is the tidyverse? What are the unifying principles the underlie all packages in the tidyverse?

The goal was always to provide composable "lego blocks" that allowed you to build up the unique analysis toolkit that you needed. Learning one part of the tidyverse should help you to learn the next because of shared conventions.

In this section, I'll discuss some of our attempts to describe the unifying principles of the tidyverse. But also want to talk about five specific innovations strongly associated with the tidyverse: tidy data, tibbles, the pipe, tidy evaluation, and hex stickers.

3.1 tidyverse principles

In my useR talking naming the tidyverse, I cited three unifying principles:

- Uniform data structures
- Uniform APIs
- Support referential transparency

> No matter how complex and polished the individual operations are, it is often the quality of the glue that most directly determines the power of the system. — Hal Abelson

I think the first two are straightforward to understand. The goal of the tidyverse is to be easy to learn by sharing data structured and interface design across as many functions as possible. You learn a few big new ideas and then apply them in as many places as possible. But what does referential transparency refer to? That's a call to the principles of tidy evaluation, which we'll come back to shortly.

I repeated this talk a few times and by December the principles had been refined to these four:

- Share data structures (i.e. tidy tibbles).
- Compose simple pieces (i.e. use the pipe).
- Embrace functional programming (instead of for loops).
- Write for humans.

> Programs must be written for people to read, and only incidentally for machines to execute. —— Hal Abelson

And that has since been further refined to the four guiding principles elucidate at https://design.tidyverse.org/unifying.html:

- It is **human centered**, i.e. the tidyverse is designed specifically to support the activities of a human data analyst.
- It is **consistent**, so that what you learn about one function or package can be applied to another, and the number of special cases that you need to remember is as small as possible.
- It is **composable**, allowing you to solve complex problems by breaking them down into small pieces, supporting a rapid cycle of exploratory iteration to find the best solution.
- It is **inclusive**, because the tidyverse is not just the collection of packages, but it is also the community of people who use them.

https://design.tidyverse.org/unifying.html

3.2 Tidy data

One of the things I really struggled with when learning visualisation with base R was the variety of data structures that different plotting functions could use. For example, with `plot()` you can either provide it with vectors of data (e.g. `plot(x, y)`) or a formula and a data frame (e.g. `plot(y ~ x, data = df)`). If you want to plot multiple series at once you can use `matplot()`, `matpoints()`, or `matlines()`, all of which take a matrix. But what if you have a longitudinal dataset with columns `x`, `y`, and `series`, where each series has distinct values of `y`. This guided the interface for ggplot2: you must always put your data in a data frame before plotting.

But how do you organise that data, as there are often multiple ways to record the same data in a data frame? I always had a very clear view that there was a "right" way to organise your data and when looking at a dataset I could usually identify what form would be most productive. But it me took me a long time to figure out how to explain my process to other people. But eventually I stumbled on what now seems obvious, the principles of tidy data:

- Each variable goes in a column.
- Each observation goes in a row.
- Each value goes in a cell.

And this explanation seems to mostly work for people, even without a precise definition of variable and obseration. I explored this idea in detail in the “Tidy data” paper, https://www.jstatsoft.org/article/view/v059i10.

(This is certainly not the only data stucture you might ever want to use, but it’s extremely useful to have a organising structure that you can rely on for the majority of your work. It’s a great default, even if you might need other forms for specialised purposes.)

These principles made it much easier to identify the status of a data set and determine whether it was likely to be a good fit for the functions of the tidyverse. But it took some further evolution before I was happy with the tools to convert from one form to another. tidyr started by providing the `gather()` and `spread()` functions, but many people had a hard time remembering which was which, and their original design wasn’t quite flexible enough to handle all the problems which we later discovered (particularly when multiple variables were encoded in the columns). In tidyr 1.0.0 (released in 2019), these were superseded by `pivot_longer()` and `pivot_wider()`.

Performed a casual survey, publicised via twitter, to determine what names people use to refer to . Received 2,649 results. (This is one of the reasons that I have found social media so rewarding: it makes it very easy to rapidly validate research). https://github.com/hadley/table-shapes

tidyr 1.0.0 also expanded the scope of the package to encompass “rectangling”, the art of tidying the deeply nested data that you get back from JSON web APIs. And formed connenctions between various ways of nesting data frames inside of other data frames. My enthusiasm for these data structures has waxed and waned over time, but they’re certainly useful to be aware of and I really appreciate the beauty of an approach that unifies the ways of working with them.

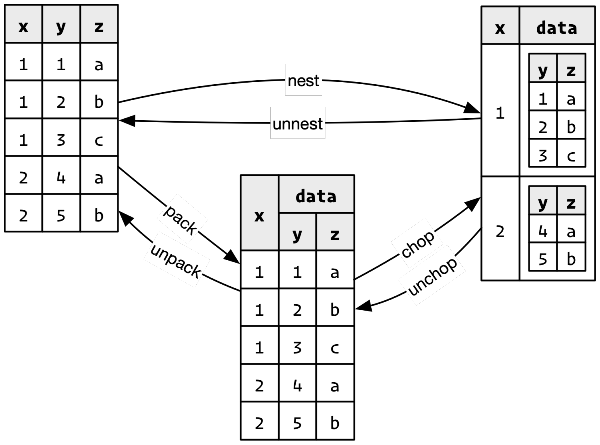


Figure 2: Three ways of representing the same data. On the right, we have a flat data frame. On the right, a nested data frame, where the data column is a list of data frames. On the bottom, we have a packed data frame, where the data column is itself a data frame.

3.3 tibbles

Somewhat related to tidy data is the “tibble”, an extension of the base R data.frame class. I created the tibble class to solve a few problems that I found annoying with data.frames, particularly around printing and subsetting.

tibbles fix a number of individually small printing issues that bothered me about data frames:

- Often flooded my console with output when working with large datasets. In older versions of R, this printing wasn’t interruptible, so you could easily lock yourself out of the console for quite a long time if you printed a big data frame. tibbles only print a small number of rows and only the columns that fit on one screen.
- When teaching, I found that students didn’t have a good sense of what type each variable was (and even I sometimes got confused whether a variable was a date or a string). tibbles include the variable types in the heading.
- In the unfortunate case you have this data frame `data.frame(x = c(NA, "<NA>"))`, there’s no way to figure out which value is actually missing. Tibbles use a side-channel, colour, to ensure

that there's no way to mix up a missing value with a string that has the same printed representation.

- My experiments with nested and packed columns required better printing tools. While they are technically supported by base data frames their printing is sufficiently confusing that their utility is much hampered.

There are a few gotchas with subsetting data frames that make it easier to make silent mistakes. `df[, cols]` will return a vector if `cols` selects one column and a data frame if it selects more than one column. `df$x` does partial matching. Subsetting a tibble with `[` always returns another tibble and `$` never does partial matching. It will also complain if `x` does not exist. For these reasons, I have often jokingly framed tibbles as a lazier and surlier version of data frames.

tibbles started life in dplyr 0.4.0 in early 2015, originally just called `tbl_df` with no suggestion as how to pronounce. tibbles started live in the dplyr package but were extracted into their own package in 2016.

tibbles were a much more contentious data structure than I had anticipated. They broke code in older packages, and my strong opinions about the formatting of significant digits also caused some consternation. That seems to have mostly died down now. Today I am much more cautious about introducing new data strucutres: they are fundamentally much more costly that introduction new functiosn and packages, and even programming paradigms.

3.4 The pipe

You can't talk about the tidyverse without talking about the pipe. I think the pipe has probably had more impact on R programming in and out of the tidyverse than any other idea. You can find an excellent and detailed history of the pipe by Adolfo Álvarez on his blog. I've summarised this history below, along with some personal colour.

First pipe operator introduced in dplyr in Oct 2013, `%.%`. Announced dplyr and this pipe operator in Jan 2014, where I learned that Stefan Milton Bache had been thinking along similar lines and had created the magrittr package. Its syntax was better `%>%` (since you can hold down the shift button the whole time) and it had more features comprehensive, so I quickly switched to using magrittr and deprecated the dplyr equivalent.

Called the pipe and pronounced "then".

Today the pipe is available in base R thanks to much collaborative effort. Lionel Henry proposed new syntax and wrote a patch to R in 2016 and Jim Hester presented it at the DSC in 2017. This had a positive impact but it took some time for the R Core Team to fully align on its benefits. But thanks to the work of Luke Tierney it was added to R in 4.1 (2020) along with lambda syntax and raw strings. Because R could modify the parser, the base pipe was written `|>`. Took a few iterations to get a placeholder syntax (_ in 4.2) and the ability to pipe into functions like `$` (in 4.3). Final major magrittr release (https://www.tidyverse.org/blog/2020/11/magrittr-2-0-is-here/) to bring consistency between magrittr and base pipe. Now gradually moving all tidyverse packages to the base pipe and beginning the long process of

As an interesting historial anecdote, there would have been no need for ggplot2 had I discovered the pipe earlier, as ggplot was based on function composition. I correctly identified that function composition was an unappealing user interface, but if I had discovered the pipe at that time, ggplot could have used it, rather than requiring a switch to `+` in ggplot2. You can try out ggplot if you want, or learn why ggplot2 can't switch to the pipe.

3.5 Tidy evaluation

One of the most contentious features of the tidyverse has been the idea of tidy evaluation. The goal of tidy evaluation is to solve a problem introduced by ggplot2 and dplyr's syntax that allows you to mix variables defined in the current environment with variables defined in some data frame. This makes interactive data analysis much more fluent at the cost of making programming much harder.

The need for tidy evaluation is fairly simple to explain. If you have repeated code, you typically want to extract out a function:

But the naive approach doesn't work:

It took multiple years of struggle to get to a place that we are happy with. Started with lazyeval [2014-2017]. While this approach worked initially, it abused R internals in a way that eventually stopped working.

This lead to the development of the tidy evaluation framework which had five big ideas:

- Code is tree
- You can capture the tree by "quoting"
- You can use unquoting to build new trees
- quoting + unquoting …
- Quosures capture the tree + the evaluation environment

I still believe that these big ideas are correct and useful. But they're primary useful for creating a strong underlying theory. During 2017 and 2018 I gave 12 talks on tidyeval to audiences around the world, trying to convince others that they could use tidyeval for their own code. But this was much less effective than I had hoped and there were general rumblings in the community that tidyeval was too hard. (e.g. commuity posts titled "Should tidyeval be abandoned?" and "Will tidyeval kill the tidyverse?")

This sent us back to the drawing board and eventually we came up with a new approach called embracing. We launched this in 2019 and this seems to have been successful.

Because we didn't pay enough attention to exactly what people were struggling with, we also missed a couple of common use cases that were surprisingly hard to solve. Now we have a cookbook of tidy eval recipes that cover the most common situations that you're likely to encounter, so that there's usually one example you can modify.

Tidy evaluation is still one of the hardest parts to grasp, but I think that's unavoidable. At least it's now no longer harder than it needs to be. I think one of the reasons it's hard is because understanding it requires distentangling two concepts that the tidyverse intentionally blurs: df-variables and env-variables. We use the same word for both, and one of the big features of the tidyverse is that it blurs their use. But to use tidyeval effectively you have to grasp the difference because you'll end up with a reference to a df-var stored in an env-var. On top of the additional complexities of writing a funtion and losing access to some of your usual debugging tools, this makes tidyeval necesasarily complex.

3.6 Hex logos

Not possible to discuss the tidyverse without also including a discussion of hex logos. It appears that Stefan and I again co-discovered https://hexb.in around the same time. I love having a shape that tiles the plane (and is bit more intersting than a square) and a spec that ensures every one's stickers are the same size and the same orientation (point down!). While the early history of hex logos is now a little murky, I'm pretty sure the first hex logo was magrittr's: https://github.com/max-mapper/hexbin/commits/gh-pages/hexagons/magrittr.png in Dec? 2014. Stefan also proposed an early version of the dplyr logo.

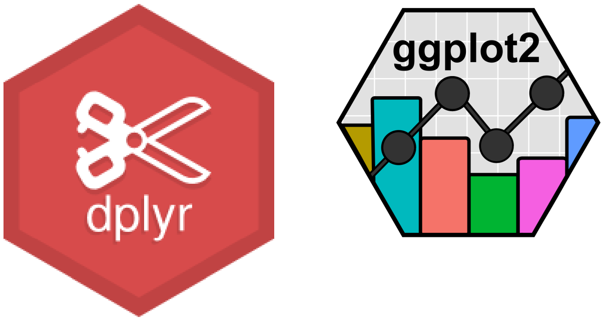


Figure 3: Early versions of the dplyr and ggplot2 logos. Shockingly the dplyr logo used scissors (not pliers!) and the ggplot2 logo was flat-down, not point-down.

By looking at emails, it looks like I fully I embraced the idea of hex stickers and started the creation for the major packages in early 2015. By mid-2016 we were ordering en masse for RStudio events, and the R community started to really coalesce around the idea. For my own packages, it now doesn't feel like a real package before it has a logo and sometime I have a name and logo before I even write any code.

I love them as a community building tool, as people looking at your laptop can immediately recognise you. I've heard many stories of people striking up a conversation with strangers just because they recognised the stickers.

Today, the vast majority of packages have a logo and I love the diversity of visual styles. Hex wall? My hex sticker board.

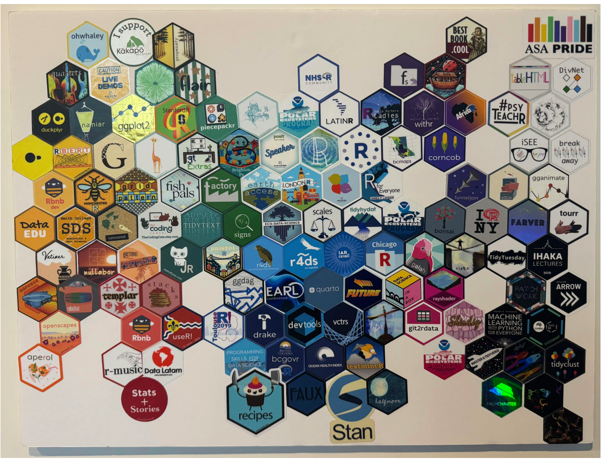


Figure 4: In my office I've made a display board with all the stickers that I've been gifted over multiple years.

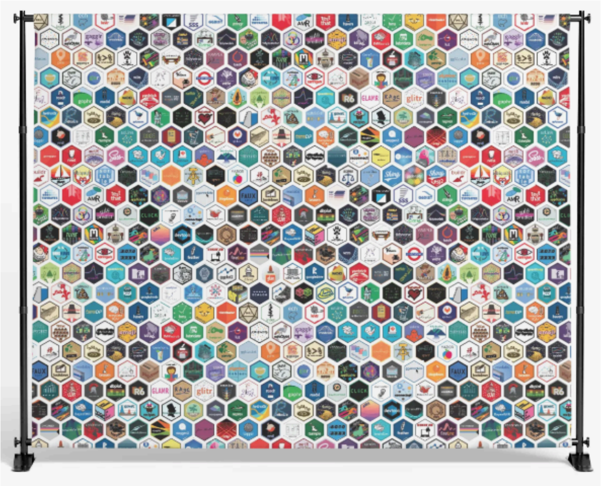
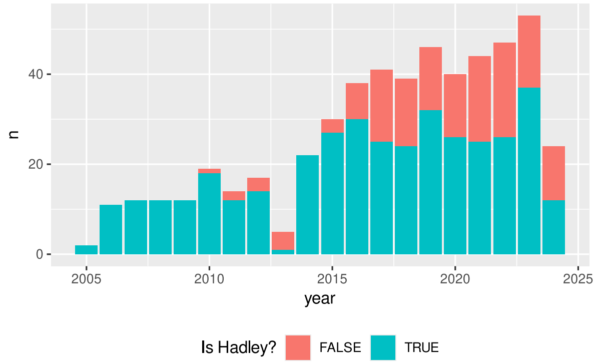
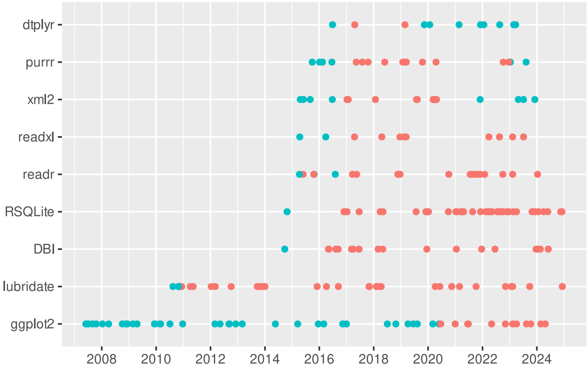


Figure 5: posit::conf() (and rstudio::conf() before) it features a hex wall that's a popular place to take photos.

(As an aside I think one of the weirdest parts of the python community is that they don't have cool and colourful logos for their packages!)

4 Growing the tidyverse





Bit about the team at RStudio (now posit). Need to think about how to frame

- Jim Hester (2016-2021)
- Mara Averick (2017-2023)
- Romain Francois (2018-2023)
- Tracy Teal (2021-2023)
- Andy Teucher (2023)
- Jenny Bryan (2017-)
- Gabor Csardi (2017-)
- Lionel Henry (2016-)
- Thomas Lin Pedersen (2018-)
- George Stagg (2022-)
- Davis Vaughan (2018-)

Paid consultants including Kirill Muller, Oliver Gjoneski, Jeroen Ooms, Charlie Gao,

ggplot2 interns:

- 2016: Thomas Lin Pedersen
- 2017: Kara Woo
- 2018: Dana Paige Seidel
- 2019: Dewey Dunnington

And in 2022, we hired Teun van Brand as a ggplot2 "fellow". This is a position similar to a summer internship but instead of lasting six months, it's lasted two years and counting.

reprex, gargle, googlesheets4, googledrive

rlang, tidyselect

cli, pillar

tidyverse.org (particularly the blog), June 2017. https://www.tidyverse.org/blog/2017/07/welcome/

4.1 tidymodels

From Max:

Hadley contacted Max Kuhn to discuss what RStudio might want to do regarding statistical and ML modeling. They and owner JJ Allaire met in New York City on 2016-09-14. One idea proposed by Max was the outline of what would become the recipes package. After more discussions with JJ in Boston, Max joined the company on 2016-11-29.

Max had already developed a popular R package called caret. That package provided functionality for basic predictive modeling tasks such as performance measures and resampling as well as a consistent user interface. Max's idea was to create a more extensible framework that would integrate more advanced tools, such as more complex pre- and post-processing, censored regression models, and others. The resulting collection of packages was called tidymodels and heavily relied on the tidyverse syntax and the underlying tools being developed (e.g., non-standard evaluation, etc.).

R has an extensive collection of user-created modeling packages and the tidymodels package has a much higher reliance on external dependencies than the tidyverse. Previous work on caret led to a very long list of formal dependencies, which made it difficult to maintain and not robust to sudden changes to these dependencies. To avoid this in tidymodels, many calls to external modeling packages are made by programmatically by constructing symbolic calls to those functions. In doing this, the tidymodels code base is smaller and depends on fewer external "hard dependencies."

tidymodels took a somewhat cheeky approach to naming. For one package that was close in spirit to caret, Max's initial idea was to give it the code name "carrot" to confuse outside users. Hadley proposed "parsnip," and this became the working name. However, eventually this name was codified by the community and was retained. The tidymodels team has been composed of between two and six people over the years, and the GitHub repository currently contains over 40 packages for modeling and predicting data.

4.2 R for data science

R for data science (2017). modelr package which is currently part of the tidyverse, but has been superseded and will be removed the next time. The modelling chapter of the book was also removed in the 2nd edition. I still believe in my vision of modelling but it was a bit quirky (making it hard to use in many courses) and generally underdeveloped. Better to use tidymodels.

Incredibly successful book.

Many translations. I'm particularly enamored of the community translations which now include Spanish, Portugese, Turkish, and Italian. But commercial translations also included Russian, Polish, Japanese, Chinese (traditional) and Chinese (simplified).

4.3 tidyverse dev day

4.4 tidyverse community

welcome to the tidyverse. Cited ~15,000 in Nov 2024. Makes it easy to cite the tidyverse (instead of citing individual packages or papers) and also gives academic credit to tidyverse maintainers who might benefit from it.

Other tidyverse maintainers.

ggplot2 governance.

ggplot2 contributors framework. ggplot2 interns. ggplot2 extension group.

https://www.tidyverse.org/blog/2019/11/tidyverse-1-3-0/

4.5 Maintaining the tidyverse

For the last few years, the tidyverse has felt pretty mature to me. It's certainly not perfect, but it feels like we have all of the main pieces in place, and much of the remaining work is grinding down the minor inconsistencies between them. Overall, the goal of the tidyverse is now consoliation and maintainence, not growth. There have been three major initiatives that have helped us create a more cohesive and streamlined experience for everyone using it.

- In 2019, we created a formal policy as to which versions of R we support: the current version, the devel version, and the previous four versions. Coupled with R's yearly release cycle, this means we support 5 years worth of R versions. This policy is important because many large enterprises use older versions of R, but still want to be able to use the latest and greatest package versions. Supporting 5 years worth of R versions only increases our maintenance burden slightly. The major downside is that we can rely on new R features only five years after they're implemented.
- In 2020 and early 2021, https://www.tidyverse.org/blog/2021/02/lifecycle-1-0-0/. 20-maintenace rstudio::global(2020). During the tidyverse's early life, there were a lot of changes as we iterated towards the right solutions in many different domains. We got the message from the community that the pace of change was too high, and so we we firmed up our policies around deprecating and removing tidyverse functions. We also introduced a new lifecycle phase called "superseded"; these are functions that we no longer recommend for new code but because of their widespread usage we have no plans to remove (but they will no longer be actively developed).
- In late 2021, thanks to the hard work on Mara Averick, we relicensed most tidyverse packages to MIT. This increased consistency across tidyverse packages, making it easier for legally conservative organisations to convince themselves their was little risk to using the tidyvrse.

What does the future hold? When I think about the tidyverse today, there's only one sweeping change that I'd like to make, and that's introducing *editions*. You would deliberately opt-in to an edition by running code like `tidyverse::edition(2025)`, stating that you want to adopt our recommended practices as of 2025. Editions would generally change defaults and disable superseded functions and arguments, ensuring that you're using our latest API recommendations. Editions makes it possible for us to change behaviour that we now believe is suboptimal without breaking existing code. You can continue to use the latest package versions (ensuring that you get new features and bug fixes) but you can increase the edition when its convenient for you to spend some time refactoring your code.

For example, we could use editions to change the default colour schemes in ggplot2, which we now know could be improved.

5 Conclusion

As the tidyverse becomes more mature, the places where the tidyverse team spends our innovation energy have started to change. Broadly, the mission of the team is to make R more awesome for doing data science, and we're willing to go whereever this takes us. Currently there are three new areas that we are exploring as a team:

- **Positron**. Positron is a new IDE for data science, produced by the same team that created RStudio. The tidyverse team has been deeply involved in the R tooling. This is existing because it gives us the skills for tighter integrations in the future. Code where coding makes sense, and use an graphical user interface where that is a better fit for the task.
- **R in production**. If you're working in industry, most tasks aren't completed by writing a one-off report. Instead you will typically produce an artifact that's run repeatedly on another machine. This is the challenge of putting your code in production, which in my opinion at least currently suffers from a thousand paper cuts. From getting your database authentication to work to ensuring that you're using exactly the same dependencies both in development and deployment, and over time, there are a lot of rough edges that you have to overcome that are not directly related to doing data science. I'm convinced that we can make this better.
- **LLMs for data science**. Pretty clear now that LLMs are going to have a transformative impact on how we do data science. We see them as invaluable assistants for the data scientist, not replacements. Allow you to get help where you need it and automate fiddly annoying tasks. Also provides a new tool kit creating tidy data frames from unstructured data, which seems likely to considerably expand the reach of the tidyverse to new types of data. Still very early days, but one initiative is the elmer package which lets you call LLMs from R.

Encompasses much of my output for the last 20 years, so it's hard to summarise it all. I hope you forgive me for anything I've forgotten, and please feel to reach out if there's something important that you think I'm missing.