A brief history of the tidyverse

Hadley Wickham

2025-01-09

1 Introduction

Unlike our universe, the tidyverse did not start with a big bang. It started with a gradual accumulation of packages that eventually snowballed into the identification and naming 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, and leading in to the early years of the tidyverse. I'll then discuss what makes the tidyverse the tidyverse, and some of the contributions of the tidyverse that I'm particularly proud off. I'll finish off with some thoughts about the current maturity of the tidyverse and where we might be heading next.

This article summarises almost 20 years of package development encompassing over 500 releases of 26 packages, as summarised by Figure 1. That means this write up is necessarily abbreviated and coupled with my fallible memory, I've almost certainly forgotton some important details. So if you're reading an early version of this paper, please let me know so I can fix it/

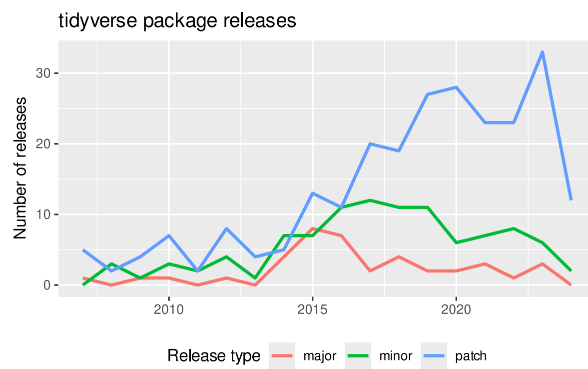

Figure 1: A timeline of tidyverse (and adjacent) package releases.

2 Before the tidyverse

While the tidyverse was named in 2016, most of the packages inside of it were created earlier, as summarised by Figure 2. This section explores how the tidyverse came to be, a journey that's inextricably tied to to the course of my career. I'll begin the story with a couple of formative experiences growing up and continue on to my PhD where I created reshape and ggplot. Then we'll move onto my professional career, first at Rice University where teaching forced my ideas to become more concrete and acessible, and then to RStudio (now Posit) where I was given the freedom and resources to dive deep in package development.

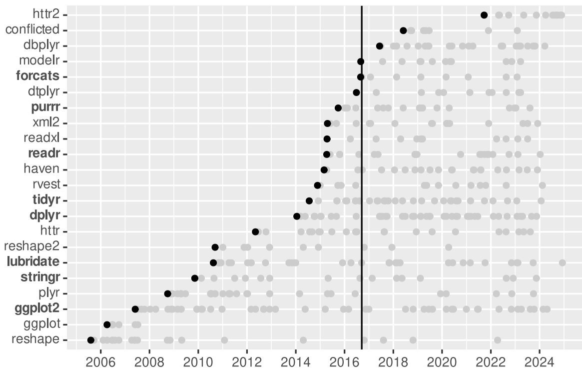

Figure 2: Initial releases of tidyverse packages and important precusors. The vertical line indicates the release of the tidyverse package. Core tidyverse packages appear in bold in the y-axis labels.

2.1 Growing up

Growing up, I was very lucky to have access to computers from a very early age, thanks to my dad[1]. This lead to an general interest in computers and programming. Thanks to my dad's work with databases, I had many conversations about relational database design and Codd's third normal form much in earlier in life than usual. That in turn lead to a lot playing around in MS Access, and an eventual part-time job developing databases. This work was invaluable when I later came to wrestle with the data needed to fit statistical models.

From my mum I learned that you can choose to make a difference in any area of life. And she instilled in me that belief that if you have the ability to make a positive impact in someone's life, you should. (She also taught me to bake, which I continue to get great enjoyment from, but I don't think that has influenced the development of the tidyverse, except for some fun purrr examples.)

The final most obvious formative experience prior to my PhD was my undergraduate at the University of Auckland, the birthplace of R. Unsurprisingly many of my courses were taught using R, and so I started using R in 2003, at version 1.6.2. Looking back at my early code is fun: the files use a `.txt` extension, mix `=` and `<-` for assignment, and use very inconsistent spacing.

[1]You can read more about him and how he shaped my life at https://tidydesign.substack.com/p/my-dad-brian-wickham.

2.2 PhD (2004-2008)

My undergraduate left me with a desire to learn more about statistics, and since my Dad had done his PhD at Cornell University, it seemed quite obvious that I should do mine in the US too. This lead me to Iowa State University (ISU) and my major advisors Di Cook and Heike Hofmann.

At ISU, 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 face two challenges that remain with me today. The first challenge was not fitting the right statistical model, but getting the data from my collaborator into that a form that I could actually work with. This challenge lead to the creation of the reshape packge 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[2].

The second challenge was translating the plots that I could picture in my head into 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[3].

[2]Compared to today's equivalent, tidyr, reshape includes a lot of tools for working with high-dimensional arrays. I was initially interested in arrays because they are rather elegant and can be much more memory efficient than data frames. But they only work well for highly crossed experimental designs and I found them very hard to explain to others. My work on arrays fell by the wayside once I decided that it was better to standardise on data frames.

[3]Lee also made a throw away comment about reshaping that lead to a vastly more performative implementation that become the heart of reshape2.

When I learned visualisation with base R, I struggled with the variety of data structures that different plotting functions could use. For example, `plot()` can either take vectors of data (e.g. `plot(x, y)`) or a formula and a data frame (e.g. `plot(y ~ x, data = df)`). But if you want to plot multiple series at once you need to use `matplot()`, `matpoints()`, or `matlines()`, all of which take a matrix.

This work wouldn't have been possible without the Di and Heike, who let me work on what I thought was most important[4], regardless of whether or not it fit the mold of a traditional statistics PhD. They also provided aircover for me when I let my disdain for the utility of theoretical statistics shine a little too clearly. My PhD culminated in my thesis "Practical tools for exploring data and models", a write-up of the collection of R packages I had begun to amass and the ideas that underpinned them.

2.3 Rice University (2008-2012)

After graduating from ISU, I got a job at Rice University. Here the most formative experience was teaching was teaching Stat405, "Introduction to data analysis" (which I'd now call introduction to data science). I taught this class four times (2009-2012) and found the experience of repeatedly teaching the class to be extremely useful. It helped me to discover both topics that students found hard to understand and tools that they found hard to use, and I could see the impact of changes from year-to-year in the maturity of students' analyses. On the small scale, 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 base R string and date-time manipulation. My teaching also catalysed my work on tidy data and group-wise manipulation that lead to the tidyr and dplyr packages that were both created after I left Rice.

At Rice, I was lucky enough to work with Garrett Grolemund as a PhD student. He developed the lubridate package, as part of his PhD thesis. https://www.jstatsoft.org/article/view/v040i03.

Throughout this time, the popularity of ggplot2 continued to rise, and I manage to carve out time to work on it, despite it not being research or valued by my department. But my interactions with the community kept me motivated and continued to reinforce my believe that this sort of work was valuable. And in 2012 I 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.

During my time at Rice I had very little success with grants. I had a hard time packaging my work in a way that the people reviewing statistics grants at the NSF could make sense of, despite my absolute conviction that this work with important. I was fortunate get some small grants from BD and Google to work on plyr, reshape, and ggplot2, but these were not inline with the amounts that I was expected to get. A paragraph from a final report to BD

> The generous support of BD 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.

Despite not having research value, I also worked on a parallel stream of work: tooling for package development. I was developing enough packages

[4] I was supposed to be working on ggobi, a tool for interactively exploring high-dimensional data. This 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. I still think this work is incredibly useful and empowering but someone interactive graphics has failed to have the impact on statistical practice that it really should.

that investing in tooling made sense, which lead to the creation of the testthat (2009) and devtools (2011) packages, and taking over maintenance of the roxygen2 (2011) package from Peter Danenberg who created it as a Google Summer of Code project in 2008 (mentored by Manuel J. A. Eugster).

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:

```
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 attaching 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. Particularly thanks to help + advice from Tareef Kawaf in 2019. 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 Quora (2016).

3 Naming and defining the tidyverse

As the collection of packages I had developed grew, the community needed some name to refer to them collectively, and many people started calling them the "Hadleyverse" (e.g. https://github.com/im anuelcostigan/hadleyverse). I found this name unappealing because the packages weren't just my work and I felt uncomfortable tying the work so closely to my name. So I started brainstorming an "official" name that I could live with. Unfortunately I can no longer find the discussion, but possibl names included the sleekverse, the dapperverse, and the deftverse. It seems inevitable now that it should be called the tidyverse, given the success of tidy data, and announced this name to the world at my keynote at useR on June 29, 2016.

A few months later in September, I released the tidyverse package. This package had two main goals:

- To make it easy to install all packages in the tidyverse with a single line of code, `install.packages("tidyverse")` so that folks could easily get a "batteries included" data science environment.

- To make it easy to load the most common packages, so that you could type `library(tidyverse)` instead of having to load packages one by one. We called these packages the **core** packgaes and in the first release these were ggplot2, dplyr, tidyr, readr, purrr, tibble. tidyerse 1.2.0 (September 2017) added forcats and stringr to the core packages, and tidyverse 2.0.0 (March 2023) added lubridate.

Naming tidyverse created a bigger question. What exactly is the tidyverse? What are the unifying principles the underlie all packages in the tidyverse? In my useR talk I cited three unifying principles:

- Uniform data structures
- Uniform APIs
- Support referential transparency

I think the first two are straightforward to understand: it's easy to learn new tools if they share data structure and interface design with existing tools. That way you can learn a few big new ideas and then apply in as many places. 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 think the first two are straightforward to understand. The goal of the tidyverse is to make it easy to learn by sharing data structures 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

Since that time I've continued to refine my description of what it means to be part of the tidyverse. You can see my latest iteration at https://design.tidyverse.org/unifying.html. At the time of writing this paper, the four principles were:

- 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.

4 Key innovations

In this section, I'll discuss five specific innovations strongly associated with the tidyverse: tidy data, tibbles, the pipe, tidy evaluation, and hex stickers.

4.1 Tidy data

I've always had a very strong sense that there's a "right" way to organise your data, a way that would make the rest of your analysis much easier. When looking at a dataset I could usually identify what form would be most productive, but I had a time explaining what I was doing to others, and experience numerous failures when trying to teach my approach. But eventually I stumbled on what now seems obvious, the principles of tidy data[5] explained in H. Wickham [1] :

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

This definition seems to mostly work for people, even without a precise definition of variable and observation, and make it much easier for folks to identify the structure of a data set and how it needed be transformed.

The tools needs to effect that transformation quite some iteration and evolution before I was happy with them. The tidy data paper was paired with the tidyr package, which initially provided the `gather()` and `spread()` functions. Unfortunately many people (including me!) had a hard time remembering which was which and what their arguments did. Additionally, their design wasn't quite flexible enough to handle all the problems which we later discovered in practice. In tidyr 1.0.0 (2019) resolved many of these problems by introducing the new`pivot_longer()` and `pivot_wider()` functions[6].

tidyr 1.0.0 also expanded the scope of the package to encompass "rectangling", what we called turning hierarchical data into tidy rectangles. This was becoming increasingly important as more data was coming from JSON web API, which tended to produce highly nested data structures. You can learn more about the problem and the solutions that tidyr provides in the tidyr rectangling vignette.

4.2 tibbles

Tidy data provides the big picture theoretical framework to help you understand productive ways of organising your data. But you also need convenient data structures to work with your tidy data and I created the tibble (an extension of the base R data.frame class) to solve a few problems that I found annoying. The problems are individually rather small, but I believed that they added needless friction and I had the confidence to believe I could do better.

- Data frames often flood your console with output when working with datasets containing many rows or many columns. In older versions of R, you couldn't even abort this printing, 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 the type of each column, and even I sometimes found myself confused about whether a column was a date, a string, or factor. To avoid this problem, tibbles show (abbreviated) variable types underneath the column names.

- In the unfortunate case you have `data.frame(x = c(NA, "<NA>"))`, there's no way to figure to

[5] 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.

[6] The names of these functions were informed functions was informed by some casual user research performed by asking my twitter followers a couple of questions. I received ~2,600 responses which you can learn read more about on https://github.com/hadley/table-shapes.

distinguish the two values from the printed output. Tibbles use a side-channel, colour, to ensure that there's no way to confuse 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. Tibbles strive to represent these more complex data types in an readable way.

There are also a few gotchas with subsetting data frames that make it easy to make silent errors in package code. For example:

- `df[, cols]` will return a vector if `cols` selects one column and a data frame if it selects more than one column.
- `df$x` does partial matching, so that if column `x`doesn't exist but column `xyz` does, it will silently return `df$xyz`.

For these reasons, subsetting a tibble with `[` always returns another tibble and `$` never does partial matching. It will also complain if `x` does not exist. This why I jokingly say that tibbles are a lazier and surlier version of data frames.

tibbles started life in dplyr, originally just called `tbl_df` with no suggestion as how to pronounce this pure consonant class name. Kevin Ushey proposed pronouncing them as tibble-diff in 2014, and the tibble bit stuck. As the utility and complexity of tibbles expanded, they need more space to grow and 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.

4.3 The pipe

> 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

The pipe has grown to be one of the defining features of the tidyverse. It allows you to rewrite function composition (e.g. `f(g(x))`) as a linear sequence of transformations (e.g. `x |> f() |> g()`). This makes certain programming patterns that are common in data science much easier to read, and has gained rapid community adoption.

My first attempt at the pipe was implemented in dplyr in Oct 2013 and called `%.%`. When I announced dplyr in Jan 2014, I learned that Stefan Milton Bache had been thinking along similar lines and had created the magrittr package. It used `%>%` instead of `%.%`, which was easier to type since you can hold down the shift button the whole time[7], and had more comprehensive features. So I quickly switched to magrittr and deprecated `%.%`. If you'd like to learn more about the history, I'd highly recommend "Plumbers, chains, and famous painters: The (updated) history of the pipe operator in R" by Adolfo Álvarez.

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

[7] The requirement for infix function names to start and end with `%` comes from R, and there's no avoiding it without patching R itself.

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). Today, we are gradually moving all tidyverse packages to use base pipe.

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 the switch to + in ggplot2. You can try out ggplot if you want, or learn why ggplot2 can't switch to the pipe.

4.4 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.

https://posit.co/blog/dplyr-0-3-2/ — _ sufffixes

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.

4.5 Hex logos

You can't talk about the history of the tidyverse without also talking about hex logos. While the early history of hex logos is now murky, from what I can tell, it appears that Stefan and I co-discovered hex stickers around the same time through https://

hexb.in. I personally found the it super appealing to have a shape that can tile the back of a laptop, along with a spec that ensures everyone's stickers are the same size and the same orientation (point down!).

I'm pretty sure the first hex logo was magrittr's[8], designed in December 2014. Soon afterwards, in early 2015, I fully I embraced the idea of hex stickers and started creating them for key packages with the help of designer Greg Swinhart; you can see two early versions of the ggplot2 logo in Figure 3. By mid-2016 we were ordering them en masse for RStudio events, and we were beginning to see logos for other packages in the community. Today, a huge number of packages have a logo, and I love seeing the creativity and diversity package authors bring to their design.

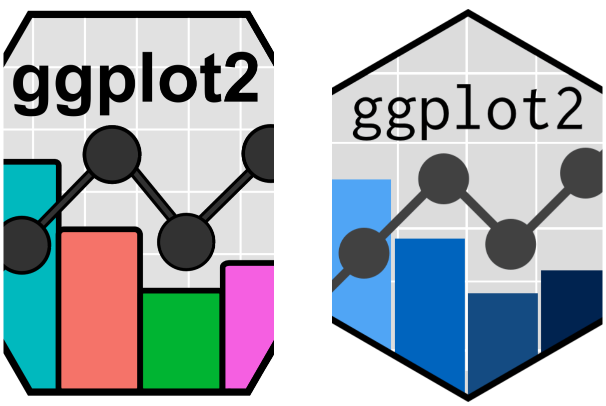

Figure 3: Two early versions of the ggplot2 hex logo. The second version was created in concert with Garrett Grolemund. I can't find any record of who created the first, but I presume it was me given that I messed up the direction of the hexagon.

I love hex stickers as a community building tool: people can see your laptop, immediately recognise you as a member of the R community, and get a sense of what you use R for. I've heard many stories of people striking up a conversation with strangers just because they recognised the stickers. They also play a role akin to trading cards and Figure 5 shows a collection of stickers that I've been given over the years. Figure 5 shows the Posit conf's hex wall, which is a popular place to take selfies.

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) features a hex wall that's a popular place to take photos.

5 Growing the tidyverse

While even the precursors to the tidyverse were greatly aided by discussions on mailing lists, the rise of GitHub and social media (particularly #rstats twitter) lead to substantial contributions from the community.

[8] https://github.com/max-mapper/hexbin/commits/gh-pages/hexagons/magrittr.png

As those contributions grew, so did RStudio, and the opportunity to hire members of the community. Forming the tidyverse, and then tidymodels.

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

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/

5.1 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.

5.2 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).

5.3 tidyverse dev day

5.4 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/

6 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.

7 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 exciting 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.

Bibliography

[1] H. Wickham, "Tidy Data," *Journal of Statistical Software*, vol. 59, no. 10, pp. 1–23, 2014, doi: 10.18637/jss.v059.i10.