

Emacs User Survey Analysis

An exploration of the 2020 Emacs User Survey results

TEC

2020-12-10

Contents

1	Basic	2
1.1	Univariate breakdowns	2
1.1.1	Survey respondents	3
1.1.2	Languages	3
1.1.3	Packages	6
1.1.4	Emacs use cases	6
1.1.5	Disabled UI elements	9
1.2	Breakdowns by category	11
1.2.1	Observations	11
1.3	Breakdown by Emacs experience	16
1.3.1	Framework	16
1.3.2	Observations	16
1.4	Prior Editor/IDE	19
1.4.1	Observations	19
2	Text mining	19
2.1	Org mode purpose	20
2.1.1	Sentiments	20
2.2	Emacs improvements	21
2.2.1	Sentiments	21
2.3	Emacs strengths	22
2.3.1	Sentiments	22
2.4	Emacs learning difficulties	22
2.4.1	Sentiments	23
2.5	Emacs, one thing to do differently	23
2.5.1	Sentiments	24

3	Multivariate analysis	24
3.1	Pairwise correlation	25
3.1.1	Observations	25
3.2	PCA	28
4	Conclusions	30
4.1	The Current State of Affairs	30
4.2	Trends	30
4.3	Pain points (new users)	31
4.4	Desired improvements	32
4.5	Final comments	32

1 Basic

❖ Warning

Frequently throughout this document I will refer to “Emacers” or “Doomers” or “Vanilla users”. In every such case imagine a little extra “(that responded to this survey)” caveat — sampling bias is no joke.

The respondent pool looks fairly diverse though, so that’s rather nice ¿.

This is an analysis of the publically available data made available from the [2020 Emacs User Survey](#).

This analysis was done with the intent of helping the Emacs community understand itself better, and which aspects of Emacs could benefit the most from development effort.

❖ Information

Feeling lazy? Jump to Conclusions.

1.1 Univariate breakdowns

Pairwise incidence matrix adds to $[x, y]$ for every result containing *both* x and y .

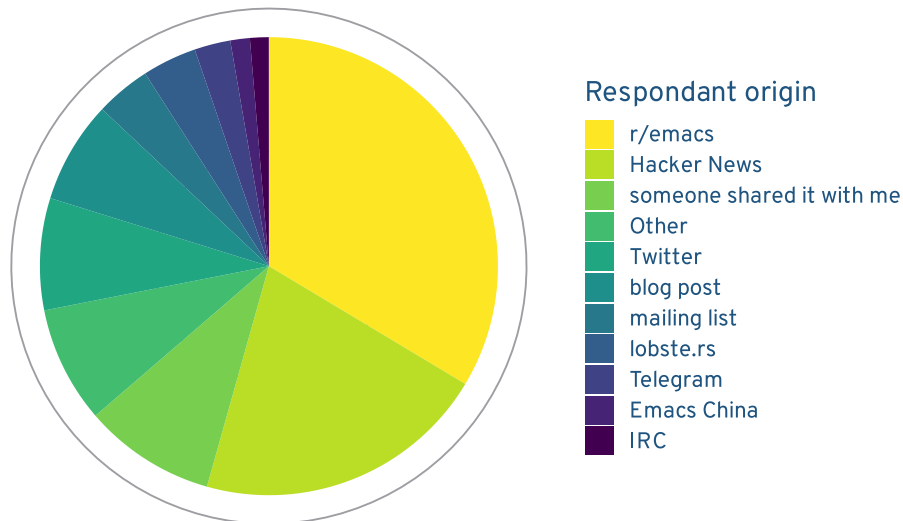


Figure 1: Breakdown of survey respondents

1.1.1 Survey respondents

1.1.2 Languages

Throughout this analysis, I will use a number of *pairwise incidence matrix* plots. This is useful when examining variables which may hold multiple values, seeing which pairs of values do and don't appear together.

Each square has 1 added to it, for each case where the variable of its row and column are both present. So, the (x, y) cell contains the number of instances where *both* x and y were present. The diagonal (x, x) simply gives the number of times x appears.

In the *proportional* pairwise incidence matrix, the values of each row are divided by the diagonal (x, x) . Now the (x, y) entry gives the proportion of the time that y is present when x already is.

This visualisation is also applied to correlation matrices further on.

The proportional pairwise incidence matrix simply divides each $[x, y]$ entry by $[x, x]$.

It could be interesting to consider how the Emacs community's use of languages differs from programmers in general. I considered using the Tirobe index, but the StackOverflow developer survey better mirrors the style of question used in this survey (list languages you use vs. primary language).

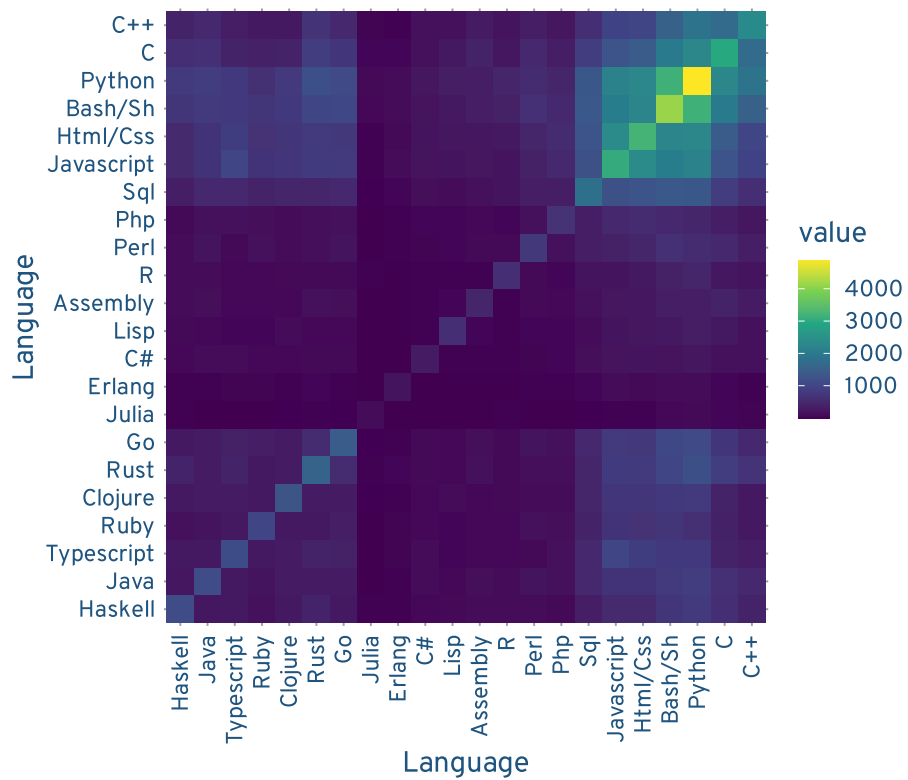


Figure 2: Pairwise incidence matrix of programming language use

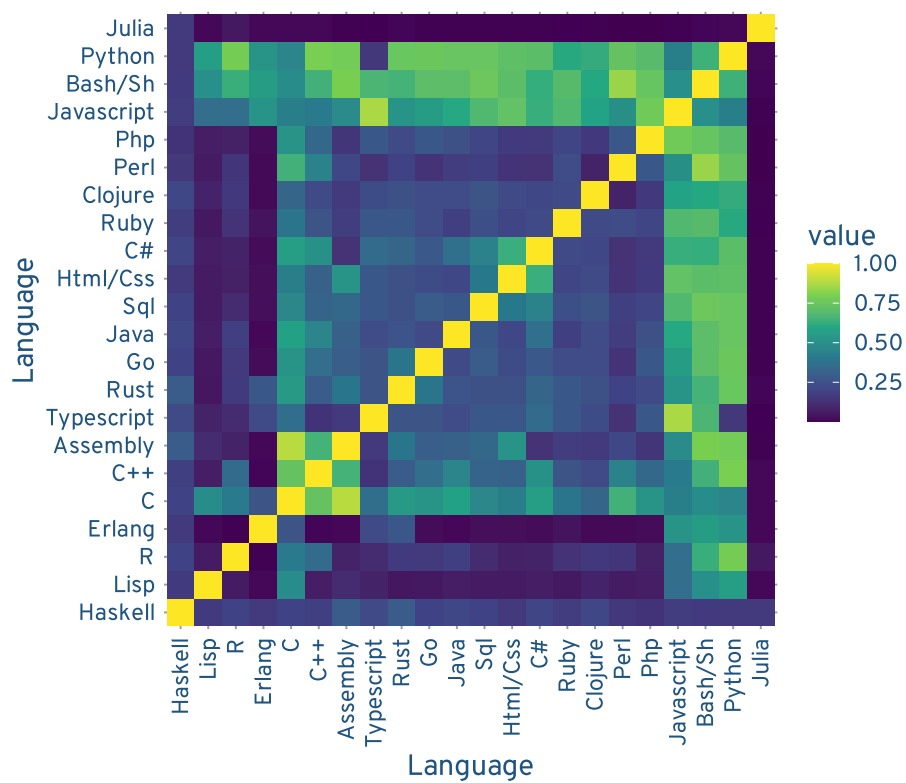


Figure 3: Proportional pairwise incidence matrix of programming language use

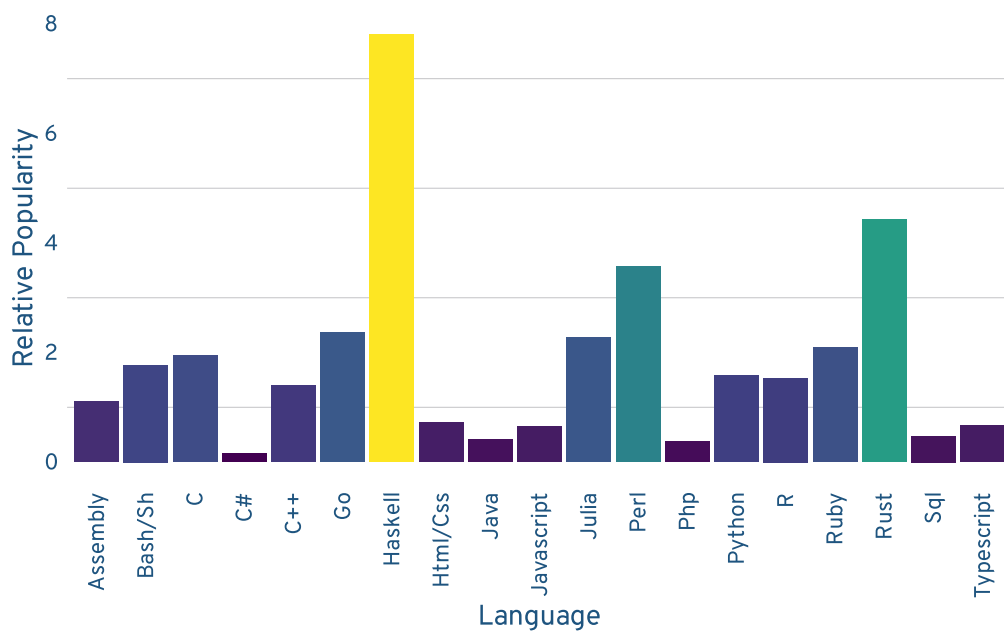


Figure 4: Language popularity relative to StackOverflow 2020 survey

Observations

- The 'usual suspects' seem popular (Python, Bash, HTML, Javascript, etc.)
 - Haskell seems more popular than one would expect (similar to Go)
 - Given that this is an *Emacs* survey, I would have expected more Lisp. Perhaps this relates to the low portion of responses that rate their elisp proficiency above "simple functions", or people are simply assuming elisp and only checking "lisp" when they use other lisps
- Python, Bash, and Javascript can be thought of as "the big three" — they're consistently used a lot in combination with other languages
- C, C++, and Assembly really like each other
- The distribution of people using Haskell with other languages is unusually flat / uniform.
- Julia users seem exceptionally monogamous, only dallying with a little Haskell and a pinch of R
- Python is nigh-universally popular, with the two exceptions of Haskell (but consistent with Haskell's general use of other languages), and oddly — Typescript.
- C users seem to tend to use a large number of other languages as well.

1.1.3 Packages

Observations

- The top 4 packages are clear: Magit, Org-mode, Projectile, and LSP-mode
- Magit seems to have uniquely broad appeal, no other package comes close in the 6

1.1.4 Emacs use cases

Observations

- Use cases are very mixed.

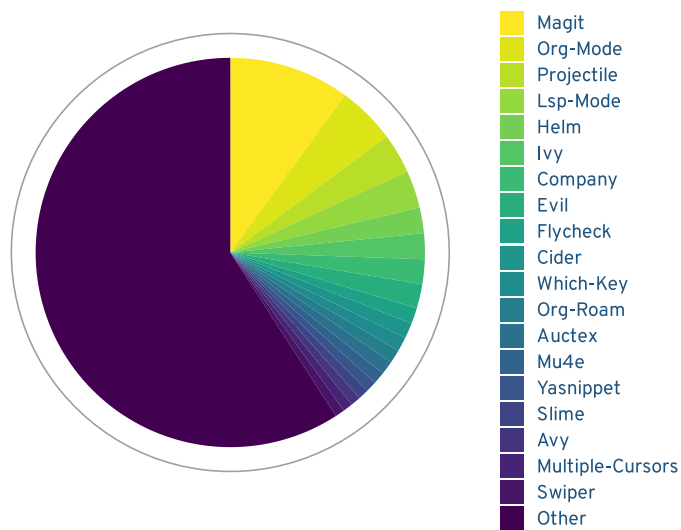


Figure 5: Most popular packages

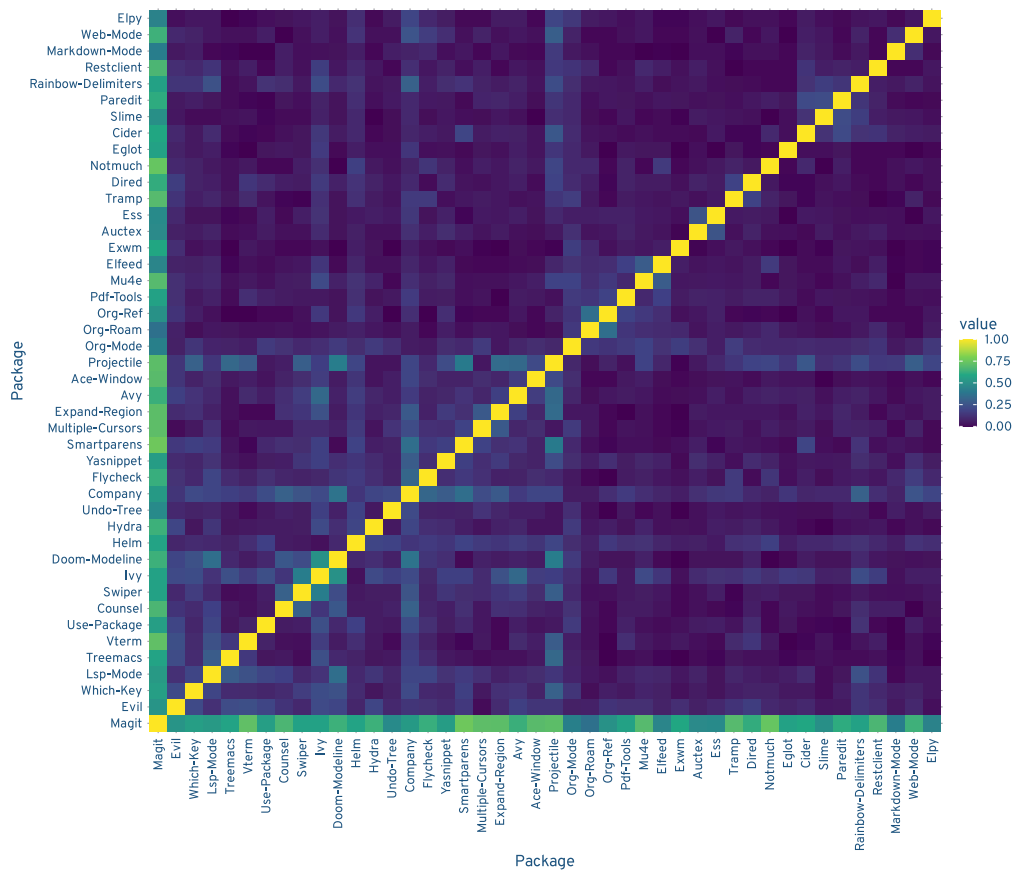


Figure 6: Proportional pairwise incidence matrix of package usage

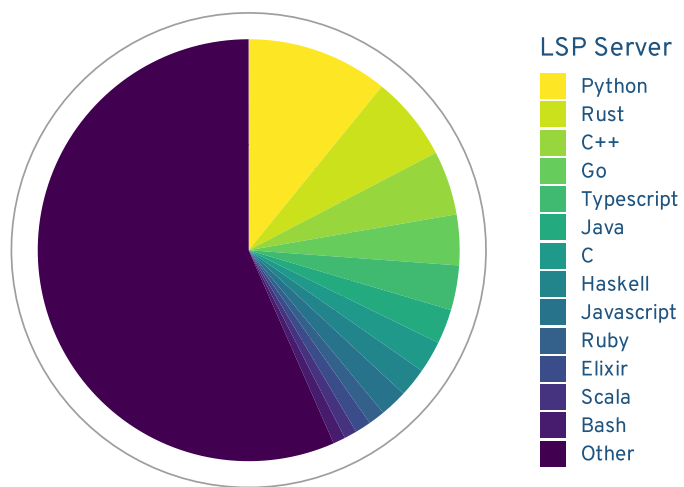


Figure 7: LSP server popularity



Figure 8: Pairwise incidence matrix of Emacs usecases

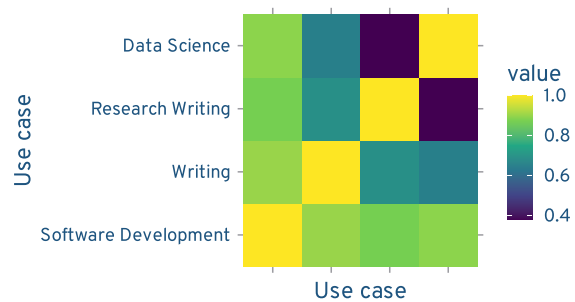


Figure 9: Proportional pairwise incidence matrix of Emacs usecases

- The vast majority of responses consisted of software development and writing
- Software developers mix uses the most, followed by writers

1.1.5 Disabled ui elements

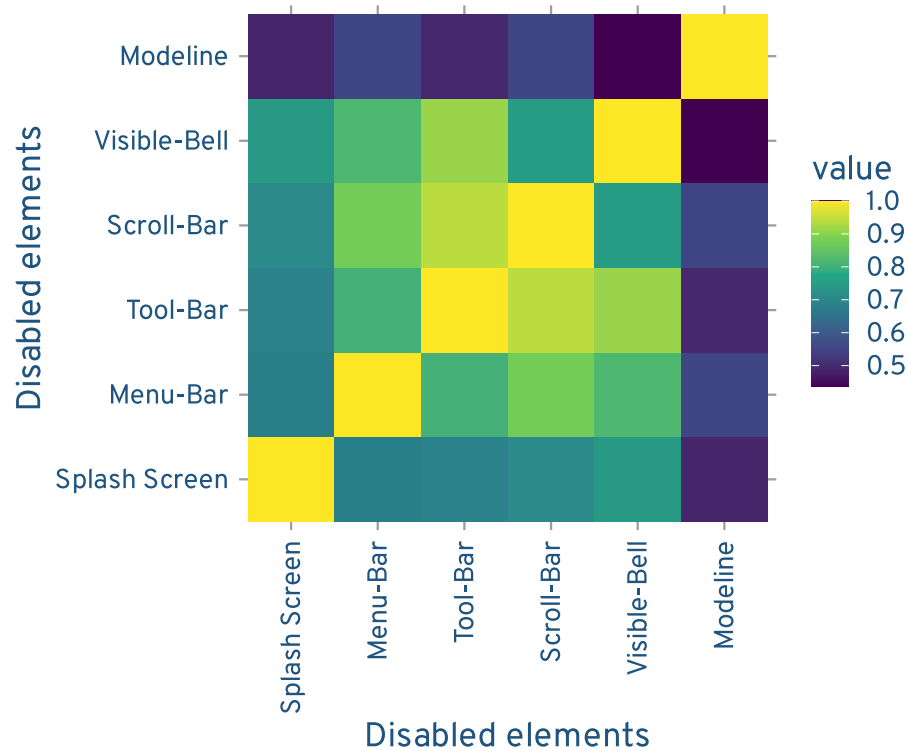


Figure 10: Pairwise incidence matrix of disabled ui elements

Observations

- Barely anyone likes the tool bar, almost everyone likes the modeline
 - What's with these modeline people? Disable the modeline but keep everything else. They're crazy.
- Tool and scroll bar dislike are most closely linked
- Splash screen haters seem to dislike everything else (except the modeline) pretty evenly

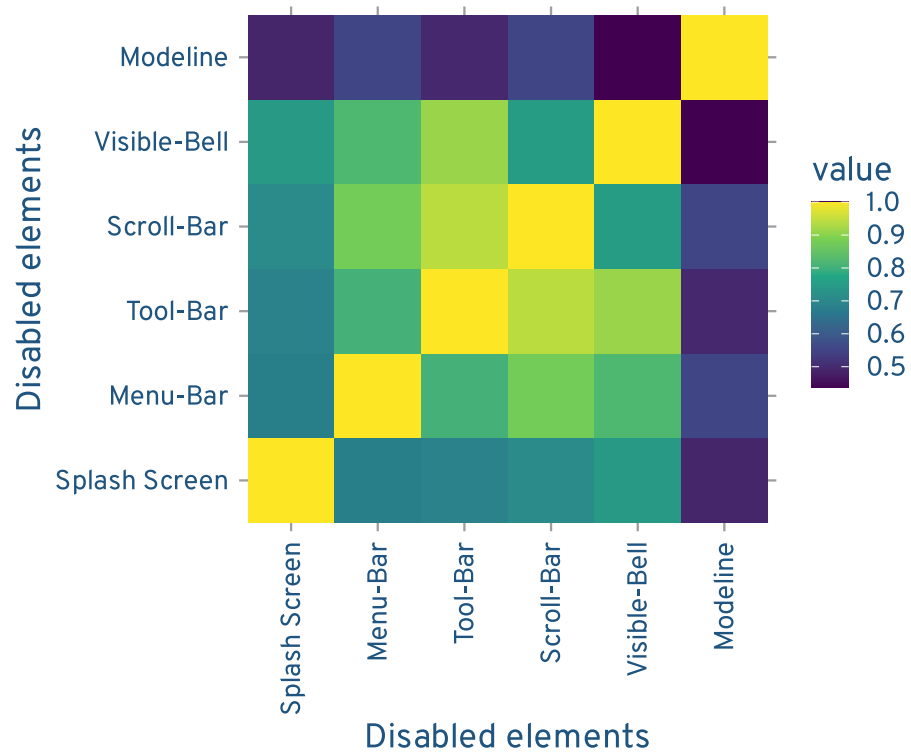


Figure 11: Proportional pairwise incidence matrix of disabled UI elements

1.2 Breakdowns by category

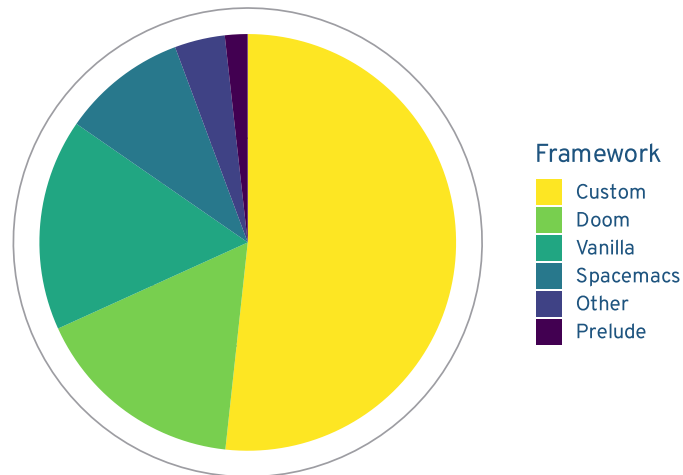


Figure 12: Emacs users by Framework

In the following graphics, to make it easier to compare the *proportion* of users who match a criteria within each framework, the user counts are normalised. To gain an intuition for the overall situation, mix the Custom column with a pinch of Doom, Vanilla, and Spacemacs.

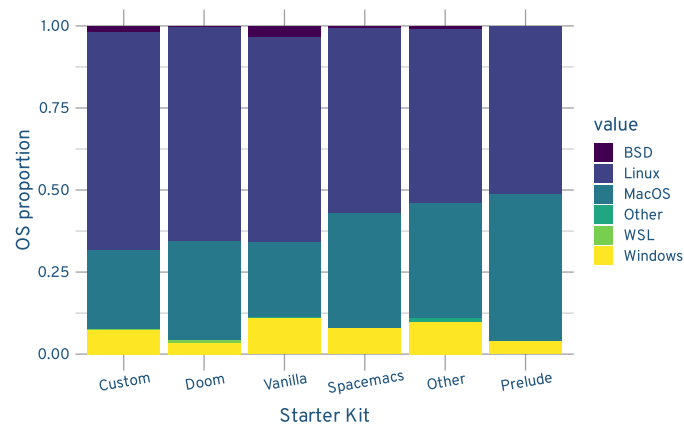


Figure 13: os proportion by starter kit, broken down by Emacs framework

1.2.1 Observations

Oh wow, a lot to unpack. Let's pick some highlights.

Purpose

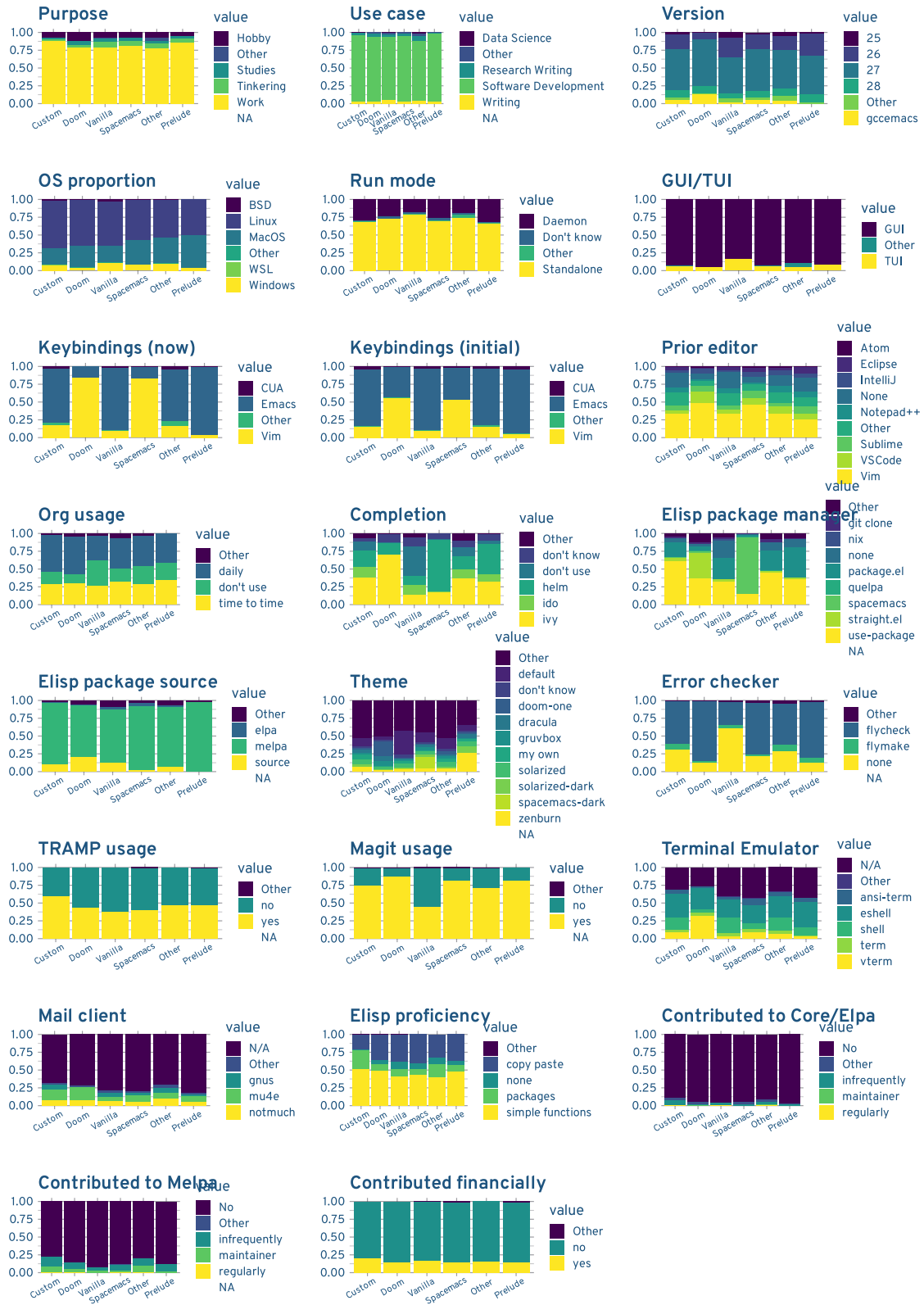


Figure 14: Categorical survey responses, broken down by Emacs framework

- Work usage high across frameworks, and all have a decent slice of hobbyists
 - Custom almost entirely work + hobby
- Doom less popular with tinkers, more with students and hobbyists

Use case

- Vanilla particularly popular for writing
- Doom is more popular for research writing, but outdone by “Other” which is highest (of the frameworks) in research writing and other.

Version

- Doom users are the most up-to-date
- Vanilla users use older versions the most

OS

- Doom and Prelude have the least Windows users
- Vanilla has the most BSD users
- Prelude has the most Mac users

Run mode

- Vanilla users use the daemon the least

GUI/TUI

- TUI is massively more popular with Vanilla users
- GUI very slightly more popular with Doom/Spacemacs than the rest

Keybindings, now + initial

- Doom and Spacemacs
 - Starts as half Vim half Emacs
 - another third converts from Emacs to Vim bindings from “initial” to “now”
 - * Vim keybindings are popular, and well-received by these users
 - CUA least popular
- Everything else
 - around 80% Emacs, but more like 90% for Prelude
 - Not much change between “initial” and “now”
 - * Custom users grab some other keybindings

Previous editor

- Doom slightly more popular than Spacemacs for ex-Vimmers
- Doom twice as popular than the next most (Spacemacs) for vscode users
- None of the others differ notably

Org usage

- Doom users use Org the most, but not by much
 - However rate of “not using org” is the lowest by a fair bit
- Across frameworks, around half use Org daily, and 80% use Org

Completion

- Doomers like Ivy, Spacers like Helm

- Half of Vanillans don't like completion it seems
 - but those that do, use ido as much as ivy/helm
- Other frameworks have a pretty consistent ~15% on ido

Elisp package management

- use-package rules, and a lot of other people like package.el
- spacemacs does it's own thing mostly

Elisp package source

- Melpa dominates
- Doom users grab packages from source much more than anyone else
 - Prelude and spacemacs seem to avoid source

Theme

- Prelude users like zenburn a fair bit
- Doomers like doom-one

Error checking

- Most vanilla users don't make mistakes ;
- Everybody else is fairly similar (mostly flycheck, some rather confident individuals, and a small slice of flymake)

TRAMP

- Consistently around 50/50 usage

Terminal emulator

- Doomers *love* vterm
- Eshell is generally pretty popular (quarter of users)

Mail client

- Quarter of people do mail in Emacs it seems
- Mu4e dominates in Doom and Custom, semi-even split between Mu4e/Notmuch elsewhere

Elisp proficiency

- Consistently, half of people feel confident with simple functions, and most of the remainder with copy and paste
- Custom users are the most confident about package writing by far

1.3 Breakdown by Emacs experience

1.3.1 Framework

Let's now look at the the distribution of years of Emacs experience, by framework, normalised by the total users of each framework.

Now normalising by total Emacs usage,

1.3.2 Observations

- Consistent preferences throughout the 10-30 year experience range
- Only one dip as users become more recent, which is from ~2000–2005
 - dot com bubble?

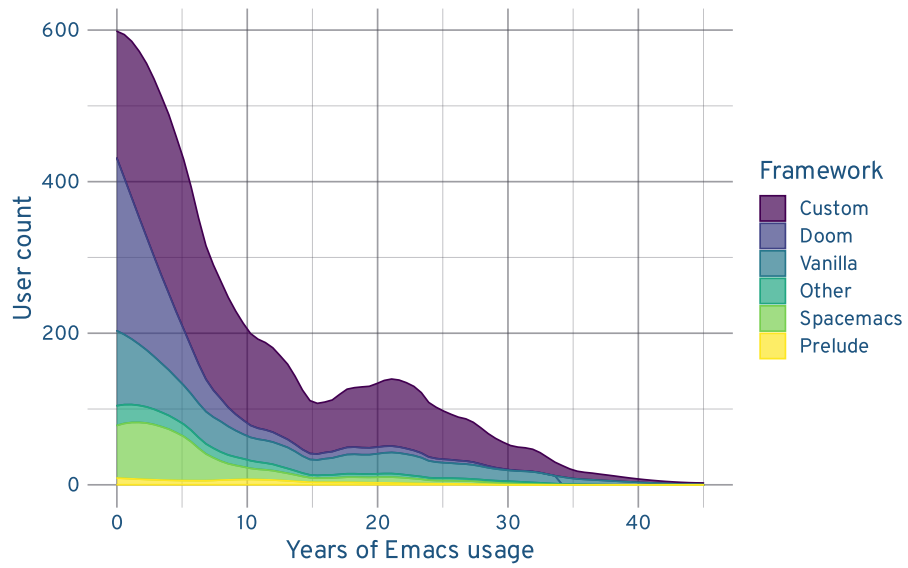


Figure 15: Emacs absolute users by year, broken down by framework

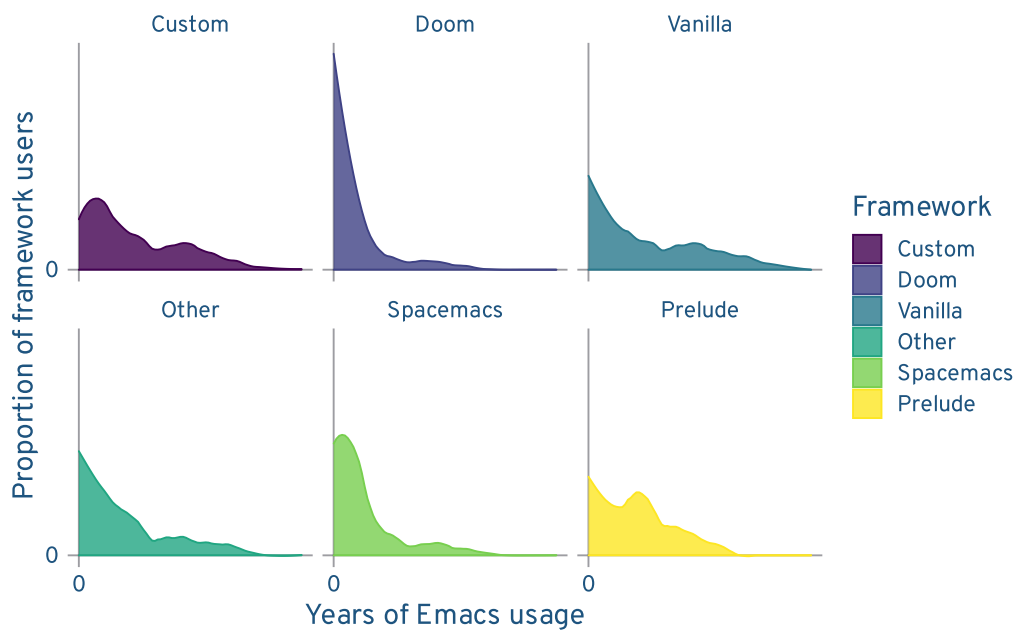


Figure 16: Emacs users by year, broken down by framework, as a proportion of the total users of each framework

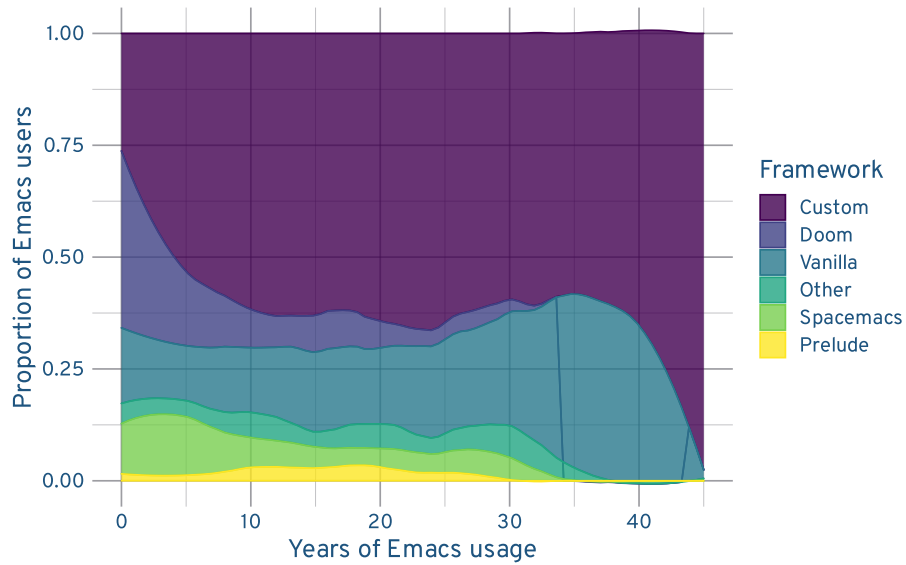


Figure 17: Emacs framework popularity

- The few 30+ year users are almost all on Custom + Vanilla
- Spacemacs has a ~5 year wide peak of ~15% centred on 3 year old users
- Prelude has a ~10 year wide to peak of ~3% users centred on 15 year old users
- Doom's popularity looks like a trumpet bell, almost half of new Emacs users (who are involved in the community) seem to be using Doom.

1.4 Prior Editor/IDE

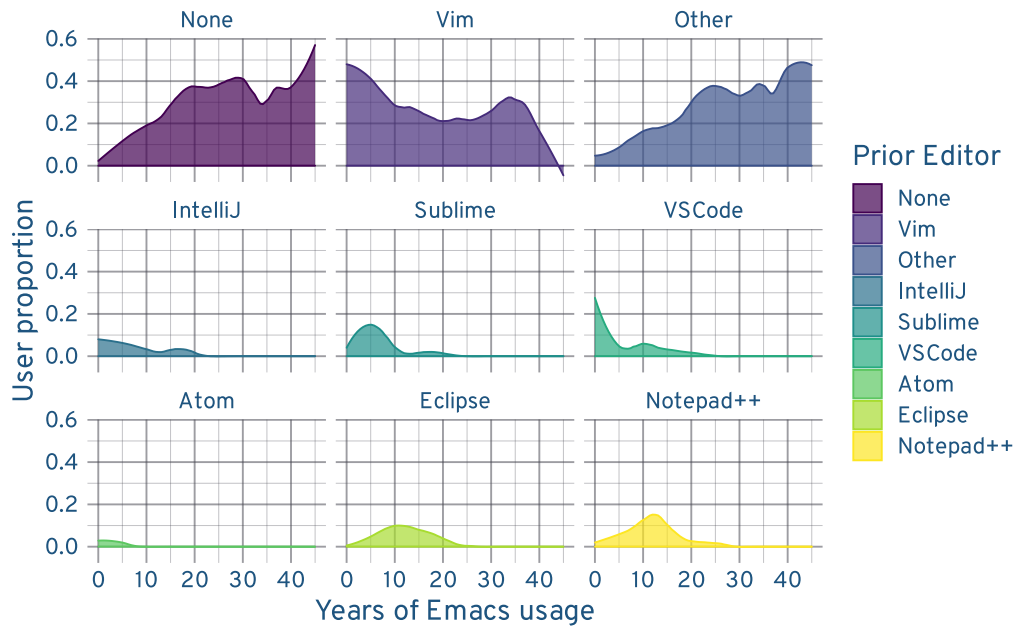


Figure 18: Prior editor as a proportion of users for each year

1.4.1 Observations

- Initially, the majority of users were 'fresh' to Emacs (no prior editor/IDE)
- Vim has semi-consistently been a source for around a quarter of new users, though that's been increasing to almost half as of late
- Eclipse, Notepad++, and Sublime have all 'peaked'
- The proportion of users coming from vscode has risen *rapidly*, from 5% to 30% over 5 years.

2 Text mining

Here, four techniques are applied:

- Word clouds, to for an indication of which words are most prevalent

- Association graphs, where links are made between words that appear together a lot
- Cluster dendogram, a hierachical tree of words
- Response 'representativeness'
 - We have word frequency data
 - Responses are given points equal to the number of times a word is seen in the corpus for each of the 100 most frequent words (the same words seen in the word clouds) to create a 'representativeness score'
 - We plot the distribution of response points, and provide the top responses to examine

2.1 Org mode purpose

Figure 19: Org purpose word cloud, association graph, and cluster dendogram.

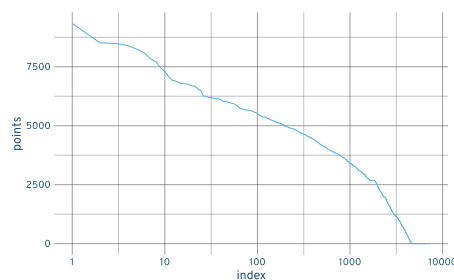


Figure 20: Representativeness points distribution across Org purpose responses

Most representative Org purpose responses

2.1.1 Sentiments

- Org is used for all kinds of writing, primaraly note taking
- A lot of people use it with task management, todo list management, . . . helping themselves get organised (see the L3 chunk of the dendogram)
- People who reference writing with Org tend to mention

- research
- literate programing
- The use of the task management facilities of Org is split between personal and work settings (see association graph)

2.2 Emacs improvements

Figure 21: Emacs improvements word cloud, association graph, and cluster dendrogram.

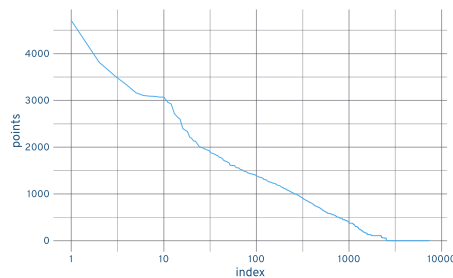


Figure 22: Representativeness points distribution across Emacs improvements responses

Most representative Emacs improvements responses

2.2.1 Sentiments

- Performance, speed improvements are popular
 - Seems to be some hope that gccemacs and multithreading may be good for this
- Talk about:
 - A more modern GUI
 - Better defaults
 - LSP support
- New users may struggle in getting Emacs to “just work” (emacs-new-easier-make-work)

2.3 Emacs strengths

Figure 23: Emacs strengths word cloud, association graph, and cluster dendrogram.

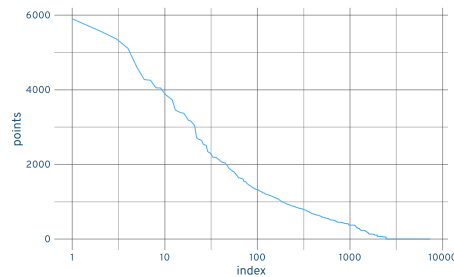


Figure 24: Representativeness points distribution across Emacs strengths responses

Most representative Emacs strengths responses

2.3.1 Sentiments

- Extensibility, Extensibility, Extensibility
 - Oh, and flexibility, configuration, customisation, . . .
- Emacs lisp can do anything I want
- It's free software
- Great community, who have created a good package ecosystem
 - Magit and Org being standout examples, which “just work”
- Use one editor for everything text
 - good programming language support

2.4 Emacs learning difficulties

Figure 25: Emacs learning difficulties word cloud, association graph, and cluster dendrogram.

Most representative Emacs learning difficulties responses

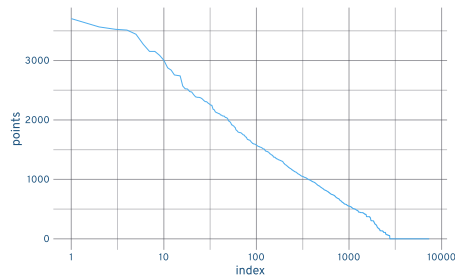


Figure 26: Representativeness points distribution across Emacs learning difficulties responses

2.4.1 Sentiments

- Keybindings are the main stumbling block
- Elisp is hard to get into, looks really strange at first
 - lots of people didn't understand it, some still don't
- It takes a lot of time to get comfortable with the 'basics'
- Not enough help getting started. Interested in a good tutorial.

2.5 Emacs, one thing to do differently

Figure 27: Emacs, do one thing differently word cloud, association graph, and cluster dendrogram.

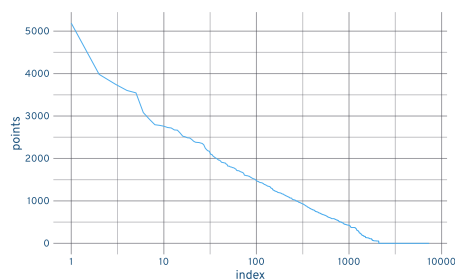


Figure 28: Representativeness points distribution across Emacs, do one thing differently responses

[Most representative Emacs, do one thing differently responses](#)

2.5.1 Sentiments

- Better defaults / language support
- Modern defaults
- Need to “just work” better

3 Multivariate analysis

To perform multivariate analysis, I'll examine the subset of questions and responses that I feel can be (sensibly) placed on a numeric scale.

```
os_score <- # unix-ness
  match_scorer(os_matcher,
    c("BSD"=0,
      "Linux"=1,
      "MacOS"=2,
      "WSL"=3,
      "Windows"=3,
      "Other"=NA))

usecase_score <- # how much coding
  match_scorer(usecase_matcher,
    c("Software Development"=0,
      "Data Science"=1,
      "Research Writing"=2,
      "Writing"=3,
      "Other"=NA))

version_score <-
  match_scorer(version_matcher,
    c("25"=25,
      "26"=26,
      "27"=27,
      "28"=28,
      "gcc"=28,
      "Other"=NA))

keybindings_score <- # how far from defaults
  match_scorer(keybindings_matcher,
    c("CUA"=0,
      "Emacs"=1,
      "Vim"=2,
      "Other"=3))
```



```

usage_score <- # how frequent
  match_scorer(usage_matcher,
    c("daily"=4,
      "weekly"=3,
      "monthly"=2,
      "time to time"=1,
      "don't use"=0,
      "no"=0,
      "Other"=NA))

package_repo_score <- # how walled-garden
  match_scorer(package_repo_matcher,
    c("elpa"=0,
      "melpa"=1,
      "source"=2,
      "Other"=NA))

elisp_skill_score <- # how proficient
  match_scorer(elisp_skill_matcher,
    c("packages"=3,
      "simple functions"=2,
      "copy paste"=1,
      "none"=0,
      "no"=0,
      "Other"=NA))

contribution_score <- # how much contributing
  match_scorer(contribution_matcher,
    c("maintainer"=3,
      "regularly"=2,
      "time to time"=1,
      "no"=0,
      "Other"=NA))

```

3.1 Pairwise correlation

How's a pairwise correlation matrix look?

3.1.1 Observations

These variables exhibit a high degree of independence, with few exceptions.

It is interesting that MELPA contribution is more strongly correlated with elisp proficiency than contribution to the Emacs core.

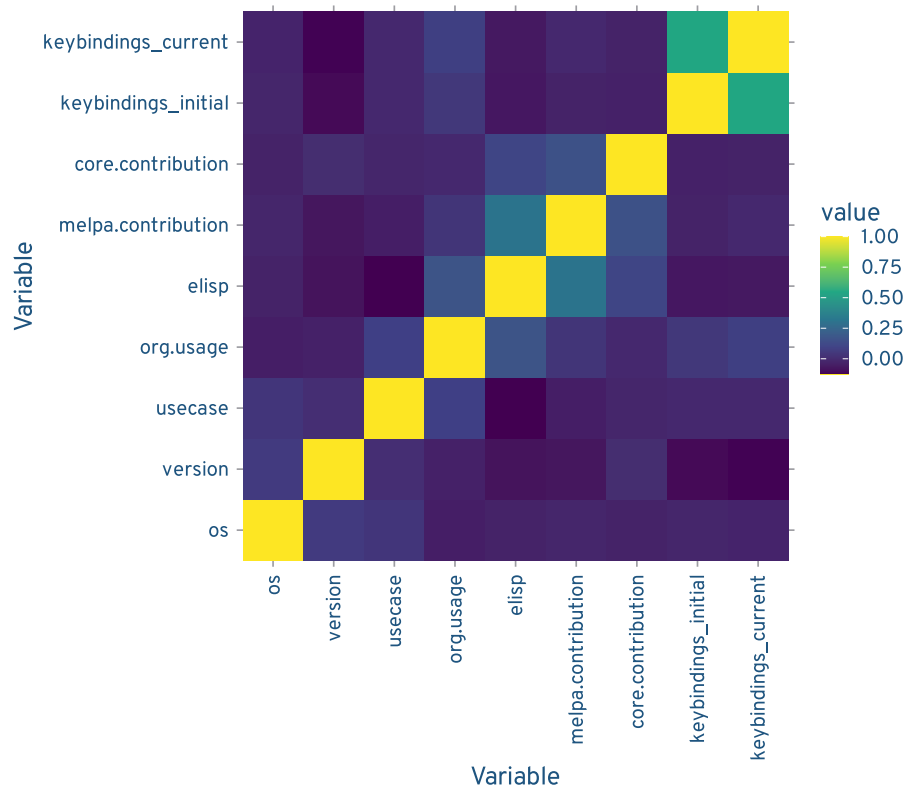


Figure 29: Pairwise correlation matrix of numeric form survey data

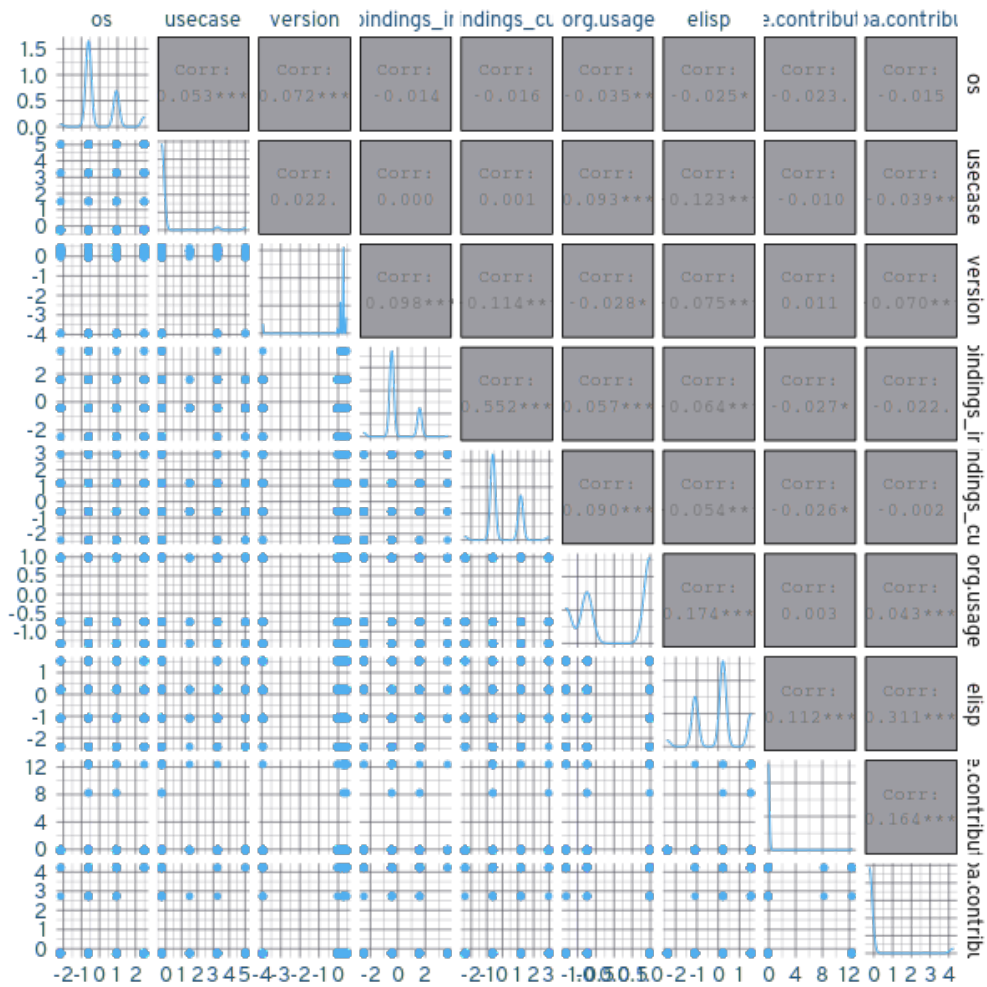


Figure 30: Pairwise value and correlation, and univariate distributions

3.2 PCA

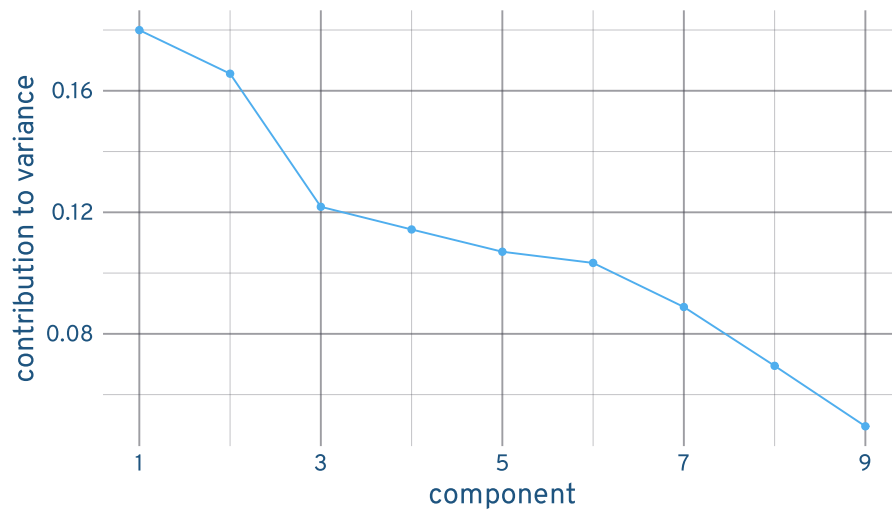


Figure 31: Scree plot

This decline in contribution to total variance is rather slow. Let's look at the first few PCs.

So far, this direction of analysis does not look very promising.

This has been good for establishing the independence between these factors, and it is interesting to see the scree plot and loadings.

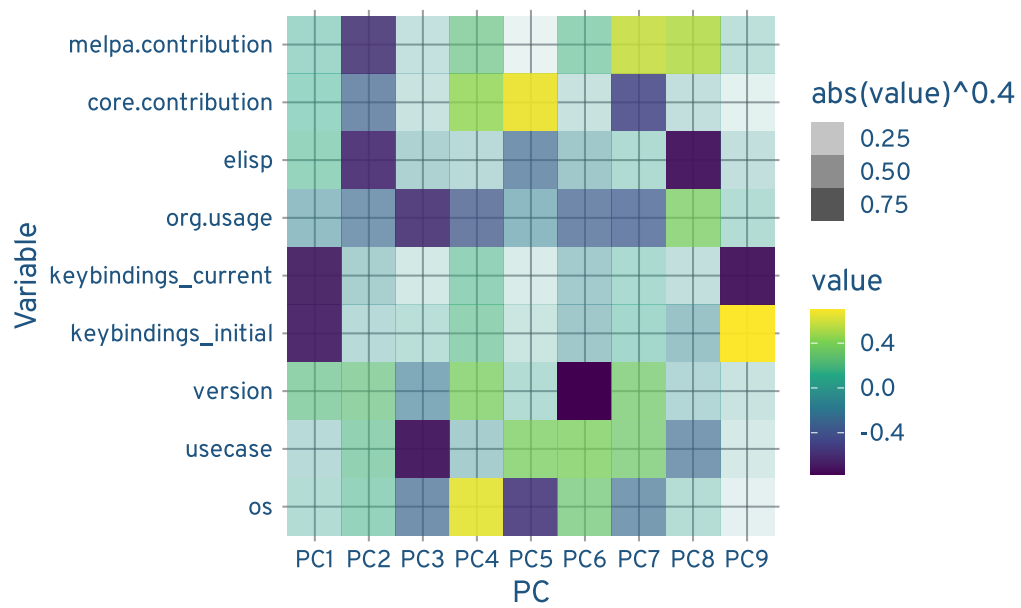


Figure 32: Variable loadings of each principal component

4 Conclusions

This survey was, in many respects very successful. It had 7344 respondents, from a 1.

Unsurprisingly, the respondents seem to be heavily biased towards more community-involved users. For instance, using the number of self-reported MELPA maintainers (394), the total respondents and total number of MELPA packages (~ 4,800) suggest a mere 90,000 Emacs users globally. The last StackOverflow survey that polled [Development Environments](#) indicated StackOverflow sees around 2 million Emacs users monthly.

4.1 The Current State of Affairs

The single most apparent result of this survey is the diversity. There is no good 'average' respondent. Emacs is used 8, however only 27% of respondents only listed Software Development as their use of Emacs. It's a similar story when it comes to languages, where there are half as many people using Emacs for Haskell as C++. It is impossible to make an accurate generalisation about the nature of Emacs' use.

However, it is possible to make generalisations about what Emacs users like. In a word: "Extensibility" (and to the surprise of no one). Related terms like "Versatility", "Flexibility", "Customisation", etc. come up frequently in the responses. I doubt the apparent diversity of use cases, and the headline strength of Emacs being "Extensibility" are a coincidence.

The respondents are 13 Linux (65%), with most of the rest on MacOS (25%), then a sliver on Windows (10%) / BSD (2%). This is a huge Compared to the [2020 StackOverflow Survey](#), BSD is 20x more prevalent, Linux 2.5x, MacOS 1x, and Windows 0.15x.

4.2 Trends

As the first *Emacs User Survey*, one can hope that future instances of this survey (this would be fantastic annually, or biannually) will provide the ability to examine trends within the community.

Without historical data to compare to, the best that can be done is to look to decisions that are rarely changed. We can the examine this with reported Emacs years of experience to get an idea of shifts in the community.

The first prominent choice is which starter-kit/framework people choose to build their configuration on. The most striking shift is the arrival of the Doom Emacs framework,

which appears 16. If one were to treat this as a growth rate, 40% of the 15 users would be from Doom users alone. While Spacemacs has also been quite popular, it appears to have peaked among individuals who have been using Emacs for two years.

The second prominent choice is choosing Emacs in the first place. Over the past four decades there has been a monumental shift in the 18. Early on around 60% of new Emacs users had *no* prior experience with text editors / IDEs. Now, that *only* applies to 3% of new users. As other editors have faded into the dust, the majority of new users stem from two sources:

- 45% from Vim, up from 30% a decade ago (moderate increase)
- 30% from vscode, up from 5% five years ago (*massive* increase)

One can suspect that the appearance of frameworks like Doom and Spacemacs may have played a role in both the increase of users, and the increase coming from Vim/vscode — however we have no way of investigating whether they're a cause or effect from this survey.

Concerningly, the apparent 15 indicated does not seem to be mirrored in development of Emacs itself. The commit frequency seems to have peaked in the late 2000s (but hasn't dropped much since), and the number of first-time contributors peaked in 2012.

4.3 Pain points (new users)

With this section it's worth keeping in mind there is likely a strong survivor bias at play — only those that persevered through any difficulties they faced would still be using Emacs and answering this survey.

Three topics consistently appeared as off-putting factors

- Keybindings
 - Four decades ago the keyboard / CUA landscape was very different
 - 12% of *all* respondents mentioned keybindings when discussing learning difficulties
- Lack of a good tutorial
 - Without anything, Emacs can be overwhelming

- Whe completely new to Emacs, the manual can also be overwhelming
- Elisp
 - Hard to work out where to start (see: Tutorial)
 - The non-elisp way of customising Emacs is not as obvious and smooth (to use) as it should be

4.4 Desired improvements

Bearing in mind the apparent bias towards Emacs-developers discussed earlier, the three most-mentioned topic seem to be:

- Improved performance
- Improved threading / async / coroutines
- Better defaults, OOTB language functionality
 - Oh, and inclusion some generally useful tools like company/magit

4.5 Final comments

All in all, I think this paints a rather positive picture for the state of Emacs and its community. Interest in Emacs seems on the rise, likely helped by the popularisation of Emacs starter kits / frameworks — which are exploring ways to make Emacs more accessible to certain segments of the population (ex-Vimmers for instance).

Some of the lesser pain-points, and a few major desired improvements are actively being addressed as I write this (thanks to gccemacs and pgtk), and LSP is unlocking a fantastic amount of work on language-specific functionality. I am optimistic that with time other prominent concerns/desires will also be addressed, and with luck future surveys will be able to interrogate the community about their involvement with Emacs development.

To everyone that participated in the survey, **thank you!** It is my hope that these results, and (with luck) those of future surveys will help us better understand the Emacs community, and inform development.