While it is possible to mangage typographic needs with the foundry tools provided via the font-rendering package-triad, one would be hard-pressed to say that the following is “fun”, or even truly manageable coding:

```
library(systemfonts)

register_variant(
  name = "some-unique-prefix Inter some-style-01",
  weight = "normal",
  features = font_feature(
    poss = 1, ibly = 1, many = 1,
    four = 1, char = 1, open = 1,
    type = 1, code = 1, spec = 1
  )
)

# remember that name

register_variant(
  name = "some-unique-prefix Inter some-style-02",
  weight = "normal",
  features = font_feature(
    poss = 1, ibly = 1, many = 1,
    four = 1, char = 1, open = 1,
    type = 1, code = 1, spec = 1
  )
)

# remember that name

# add a dozen more lines ...

ggplot() +
   geom_text(family = "oops-i-just-misspelled-the-family-name-*again*",
...)
```

We’ve been given the power to level up our chart typography, but it’s sort of where literal typesetters (the ones who put blocks of type into a press) were and we can totally make our lives easier and charts prettier with the help of a new package — {hrbragg} https://git.rud.is/hrbrmstr/hrbragg — which is somewhat of a bridge between {ragg}, {systemfonts}, {textshaping} and a surprisingly popular package of mine: {hrbrthemes}. {hrbragg} is separate from {hrbrthemes} since this new typographic ecosystem is fairly restricted to {ragg} graphics devices (*for the moment*, as Thomas alluded the other day), and the new themes provided in {hrbragg} are a bit of a level-up from those in its sibling package.

Feature Management

At the heart of {systemfonts} lies the ability to tweak font features and bend them to your will. This somewhat old post shows why these tweaks exist and delves (but not *too* deeply) into the details of them, down to the four-letter codes that are used to represent and work with a given feature. But, what does `calt` mean? And, what is this `tnum` fellow you’ll be seeing a great deal

of in R-land over the coming months? While one *could* leave the comfort of RStudio, VS Code, or vim to visit one of the reference links in Thomas' package or {hrbragg}, I've included the most recent copy of tag-code<->full-tag-name<->short-tab-description in {hrbragg} as a usable data frame so you can treat it like the data it is!

```
library(systemfonts) # access to and tweaking OTFs!
library(textshaping) # lets us treat type as data
library(ragg)        # because it'll be lonely w/o the other two
library(hrbragg)     # remotes::install_git("https://git.rud.is/hrbrmstr/hrbragg.
git")
library(tidyverse)   # nice printing, {ggplot2}, and b/c we'll do some
font data wrangling

data("feature_dict")

feature_dict
## # A tibble: 122 x 3
##    tag   long_name                  description
##
##  1 aalt  Access All Alternates      Special feature: used to
present user with choice all alternate forms of the character
##  2 abvf  Above-base Forms           Replaces the above-base part of
a vowel sign. For Khmer and similar scripts.
##  3 abvm  Above-base Mark Positioning Positions a mark glyph above a
base glyph.
##  4 abvs  Above-base Substitutions   Ligates a consonant with an
above-mark.
##  5 afrc  Alternative Fractions      Converts figures separated by
slash with alternative stacked fraction form
##  6 akhn  Akhand                     Hindi for unbreakable.  Ligates
consonant+halant+consonant, usually only for k-ss and j-ny
combinations.
##  7 blwf  Below-base Forms           Replaces halant+consonant
combination with a subscript form.
##  8 blwm  Below-base Mark Positioning Positions a mark glyph below a
base glyph
##  9 blws  Below-base Substitutions   Ligates a consonant with a
below-mark.
## 10 c2pc  Capitals to Petite Caps    Substitutes capital letters
with petite caps
## # … with 112 more rows
```

You can also use `help("opentype_typographic_features")` to see an R help page with the same information. That page also has links external resource, one of which is a detailed manual of each feature with use-cases (in the event even the short-description is not as helpful as it could be).

Before one can think about using the bare-metal `register_variant(..., font_feature(...))` duo, one has to know what features a particular type family supports. We can retrieve the feature codes with `textshaping::get_font_features()` and look them up in this data frame to get an at-a-glance view:

```
# old school subsetting ftw!
feature_dict[feature_dict$tag %in% textshaping::get_font_
features("Inter")[[1]],]
## # A tibble: 19 x 3
##    tag   long_name                   description
##
##  1 aalt  Access All Alternates       Special feature: used to
present user with choice all alternate forms of the character
##  2 calt  Contextual Alternates       Applies a second substitution
feature based on a match of a character pattern within a context of
surrounding patterns
##  3 case  Case Sensitive Forms        Replace characters, especially
punctuation, with forms better suited for all-capital text, cf. titl
##  4 ccmp  Glyph Composition/Decompos… Either calls a ligature
replacement on a sequence of characters or replaces a character with a
sequence of glyphs. Provides…
##  5 cpsp  Capital Spacing             Adjusts spacing between letters
in all-capitals text
##  6 dlig  Discretionary Ligatures     Ligatures to be applied at the
user's discretion
##  7 dnom  Denominator                 Converts to appropriate
fraction denominator form, invoked by frac
##  8 frac  Fractions                   Converts figures separated by
slash with diagonal fraction
##  9 kern  Kerning                     Fine horizontal positioning of
one glyph to the next, based on the shapes of the glyphs
## 10 locl  Localized Forms             Substitutes character with the
preferred form based on script language
## 11 mark  Mark Positioning            Fine positioning of a mark
glyph to a base character
## 12 numr  Numerator                   Converts to appropriate
fraction numerator form, invoked by frac
## 13 ordn  Ordinals                    Replaces characters with
ordinal forms for use after numbers
## 14 pnum  Proportional Figures        Replaces numerals with glyphs
of proportional width, often also onum
## 15 salt  Stylistic Alternates        Either replaces with, or
displays list of, stylistic alternatives for a character
## 16 subs  Subscript                   Replaces character with
subscript version, cf. numr
## 17 sups  Superscript                 Replaces character with
superscript version, cf. dnom
## 18 tnum  Tabular Figures             Replaces numerals with glyphs
of uniform width, often also lnum
## 19 zero  Slashed Zero                Replaces 0 figure with slashed
0
```

Most of those will not be super-useful (yet) but there are three key features that I believe one needs when picking a font for a chart:

- One of the `*lig`s (because ligatures.) are so gosh darn cool, pretty, and useful)
- `tnum` for tabular numbers (essential in axis value display, and more)

- `kern` for sweet, sweet letterspacing, or *kerning*

Since I've just made up a rule, let's see how many fonts I have that support said rule:

```
(fam <- unique(system_fonts()[["family"]])) %>%
  get_font_features() %>%
  set_names(fam) %>%
  keep(~sum(c(
    any(grepl("kern", .)),
    any(grepl("tnum", .)),
    any(grepl(".lig|liga", .))
    )) == 3
  ) %>%
  names() %>%
  sort()
##  [1] "Barlow"                 "Goldman Sans"           "Goldman Sans
Condensed" "Grantha Sangam MN"
##  [5] "Inter"                  "Kohinoor Devanagari"    "Mukta Mahee"
"Museo Slab"
##  [9] "Neufile Grotesk"        "Roboto"                 "Roboto
Black"           "Roboto Condensed"
## [13] "Roboto Light"           "Roboto Medium"          "Roboto Thin"
"Tamil Sangam MN"
## [17] "Trattatello"
```

I do have more, but they're on a different Mac 😎.

{hrbragg} comes with Inter, Goldman Sans, and Roboto Condensed, so let's explore one of them — Inter — and see how we might be able to make it useful but not tedious. The supported features of Inter are above and here are the family members:

```
system_fonts() %>%
  filter(family == "Inter") %>%
  select(name, family, style, weight, width, italic, monospace)
##  A tibble: 18 x 7
##    name                   family style              weight     width
italic monospace
##
##  1 Inter-ExtraLight       Inter  Extra Light        light
normal FALSE  FALSE
##  2 Inter-MediumItalic     Inter  Medium Italic      medium
normal TRUE   FALSE
##  3 Inter-ExtraLightItalic Inter  Extra Light Italic light
normal TRUE   FALSE
##  4 Inter-Bold             Inter  Bold               bold
normal FALSE  FALSE
##  5 Inter-ThinItalic       Inter  Thin Italic        ultralight
normal TRUE   FALSE
##  6 Inter-SemiBold         Inter  Semi Bold          semibold
normal FALSE  FALSE
##  7 Inter-BoldItalic       Inter  Bold Italic        bold
normal TRUE   FALSE
##  8 Inter-Italic           Inter  Italic             normal
```

```
normal TRUE   FALSE
##  9 Inter-Medium           Inter  Medium             medium
normal FALSE  FALSE
## 10 Inter-BlackItalic      Inter  Black Italic       heavy
normal TRUE   FALSE
## 11 Inter-Light           Inter  Light              normal
normal FALSE  FALSE
## 12 Inter-SemiBoldItalic   Inter  Semi Bold Italic   semibold
normal TRUE   FALSE
## 13 Inter-Regular          Inter  Regular            normal
normal FALSE  FALSE
## 14 Inter-ExtraBoldItalic  Inter  Extra Bold Italic  ultrabold
normal TRUE   FALSE
## 15 Inter-LightItalic      Inter  Light Italic       normal
normal TRUE   FALSE
## 16 Inter-Thin             Inter  Thin               ultralight
normal FALSE  FALSE
## 17 Inter-ExtraBold        Inter  Extra Bold         ultrabold
normal FALSE  FALSE
## 18 Inter-Black            Inter  Black              heavy
normal FALSE  FALSE
```

Nobody. I mean, *nobody* wants to type eighteen+ font variant registration statements, which is why {hrbragg} comes with `reconfigure_font()`. Just give it the family name, the features you want supported, and it will take care of the tedium for you:

```
reconfigure_font(
  prefix = "hrbragg-pkg",
  family = "Inter",
  width = "normal",
  ligatures = "discretionary",
  calt = 1, tnum = 1, case = 1,
  dlig = 1, ss01 = 1, kern = 1,
  zero = 0, salt = 0
) -> customized_inter

# I'll have a proper print method for this soon

str(customized_inter, 1)
## List of 17
##  $ ultralight_italic: chr "clever-prefix Inter Thin Italic"
##  $ ultralight       : chr "clever-prefix Inter Thin"
##  $ light            : chr "clever-prefix Inter Extra Light"
##  $ light_italic     : chr "clever-prefix Inter Extra Light Italic"
##  $ normal_italic    : chr "clever-prefix Inter Light Italic"
##  $ normal_light     : chr "clever-prefix Inter Light"
##  $ normal           : chr "clever-prefix Inter Regular"
##  $ medium_italic    : chr "clever-prefix Inter Medium Italic"
##  $ medium           : chr "clever-prefix Inter Medium"
##  $ semibold         : chr "clever-prefix Inter Semi Bold"
##  $ semibold_italic  : chr "clever-prefix Inter Semi Bold Italic"
##  $ bold             : chr "clever-prefix Inter Bold"
```

```
##  $ bold_italic      : chr "clever-prefix Inter Bold Italic"
##  $ ultrabold_italic : chr "clever-prefix Inter Extra Bold Italic"
##  $ ultrabold        : chr "clever-prefix Inter Extra Bold"
##  $ heavy_italic     : chr "clever-prefix Inter Black Italic"
##  $ heavy            : chr "clever-prefix Inter Black"
##  - attr(*, "family")= chr "Inter"
```

The `reconfigure_font()` function applies the feature settings to all the family members, gives each a name with the stated prefix and provides a return value that supports *autocompletion of the name* in smart IDEs and practically negates the need to type out long, unique font names, like this:

```
ggplot() +
  geom_text(
    aes(1, 2, label = "Welcome to a <- customized -> Inter!"),
    size = 6, family = customized_inter$ultrabold
  ) +
  theme_void()
```

Note that we have a lovely emboldened font with clean ligatures without much work at all! (I should mention that if a prefix is not specified, a UUID is chosen instead since we don't really care about the elongated names anymore).

While we've streamlined things a bit already, we can do even better.

Font-centric Themes

Just like {hrbrthemes}, {hrbragg} comes with some font/typographic-centric themes. We'll focus on the one with Inter for the blog post. For the moment, you'll need to `install_inter()` (you likely got prompted to do that if you already installed the package). This requirement will go away soon, but you'll want to use Inter everywhere anyway, so I'd keep it installed.

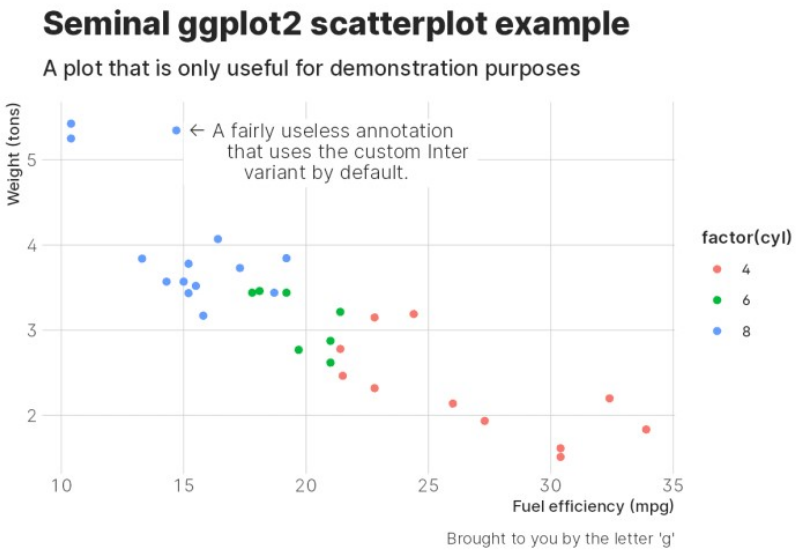
Once that's done, you're ready to use `theme_inter()`.

What's that you say? Don't we need to create a font variant first?

Would *I* do that to you? Never! {hrbragg} comes with a preconfigured `inter_pkg` font variant (which I'll be tweaking a bit over the weekend for some edge cases) that pairs nicely with `theme_inter()`. Here it is in action with an old friend of ours:

```
ggplot() +
  geom_point(
    data = mtcars,
    aes(mpg, wt, color = factor(cyl))
  ) +
  geom_label(
```

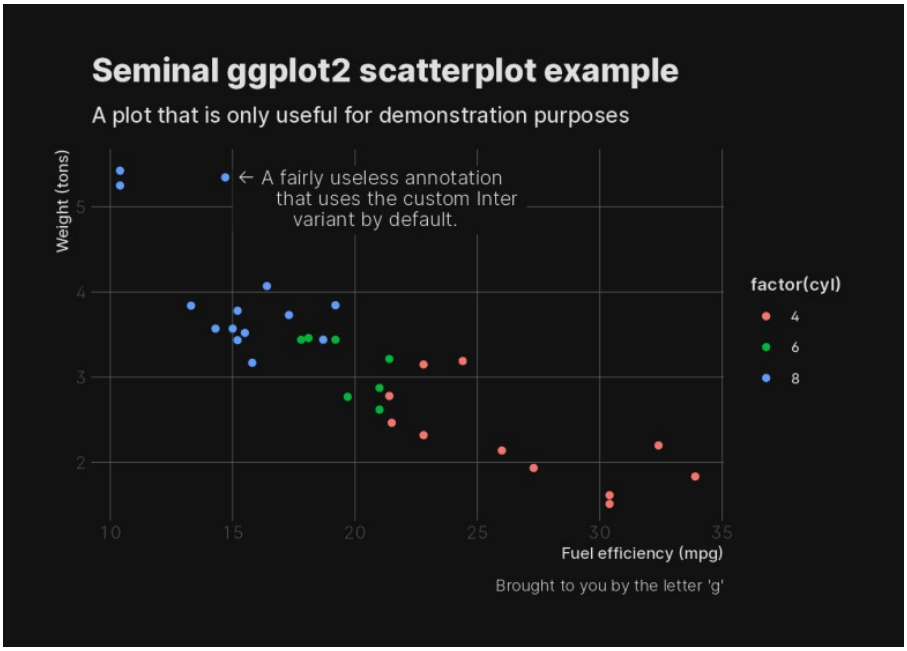
```
    aes(
      x = 15, y = 5.48,
      label = "<- A fairly useless annotation\n       that uses the
custom Inter\n          variant by default."
    ),
    label.size = 0, hjust = 0, vjust = 1
  ) +
  labs(
    x = "Fuel efficiency (mpg)", y = "Weight (tons)",
    title = "Seminal ggplot2 scatterplot example",
    subtitle = "A plot that is only useful for demonstration purposes",
    caption = "Brought to you by the letter 'g'"
  ) -> gg1

gg1 + theme_inter(grid = "XY", mode = "light")
```



Wonderful kerning, a custom-built arrow due to fantastic, built-in ligatures, and spiffy tabular numbers. Gorgeous!

What was that you just asked? *What's up with that* `mode = "light"`*?*. Did I forget to mention that all the {hrbragg} themes come with dark-mode support built in? My sincerest apologies. Choosing `mode = "dark"` will use a (configuratble) dark theme and using `mode = "rstudio"` (if you're an RStudio user) will have the charts take on the IDE theme setting automagically. Here's dark mode:

```
gg1 + theme_inter(grid = "XY", mode = "dark")
```



The font+theme pairs automatically work and reconfigure all the ggplot2 aesthetic defaults accordingly. Since this makes heavy use of `update_geom_defaults()` I've included a (very necessary) `reset_ggplot2_defaults()` to get things back to normal when you need to.

Note that you can use `adaptive_color()` to help enable dark/light-mode color switching for your own pairings, and `theme_background_color()` or `theme_foreground_color` to utilize the (reconfigurable) default fore- and background theme colors.

Try before you buy…into using a given font

One can't know ahead of time whether a font is going to work well, and you might want go get a feel for how a given set of family variants work for you. To that end, I've made it possible to preview any font you've reconfigured with `reconfigure_font()` via `preview_variant()`. It uses some pre-set text that exercises the key features I've outlined, but you can sub your own for them if you want to look at something in particular. Let's give `inter_pkg` a complete look:

```
preview_variant(inter_pkg)
```

Inter

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

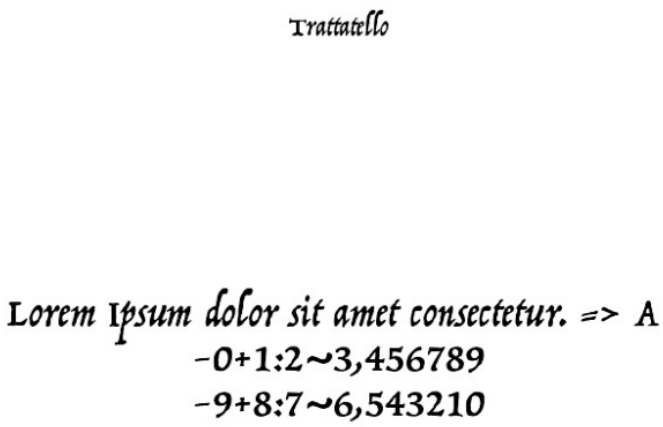
Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

Lorem Ipsum dolor sit amet consectetur. ⇒ Ⓐ
−0+1:2~3,456789
−9+8:7~6,543210

We can look at another one that we'll create now (I did not realize this font had tabular numbers

until Thomas built all these wonderful toys to play with!):

```
reconfigure_font(
  family = "Trattatello",
  width = "normal",
  ligatures = "discretionary",
  calt = 1, tnum = 1, case = 1,
  dlig = 1, kern = 1,
  zero = 0, salt = 0
) -> trat

preview_variant(trat)
```



FIN

The {hrbragg} package is not even 24 hours old yet, so there are breaking changes and many new, heh, *features* still to come, but please — as usual — kick the tyres and post questions, feedback, contributions, or suggestions wherever you're most comfortable (the package is on most of the popular social coding sites).