From data consumer to tool maker

Winston Chang

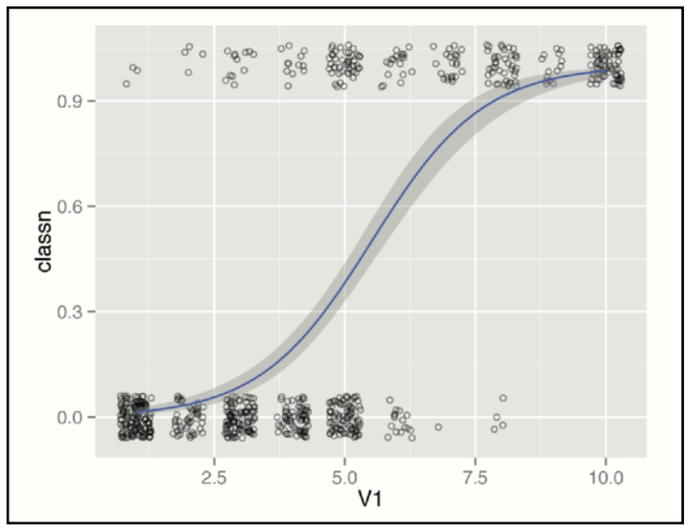
RStudio

Tool maker

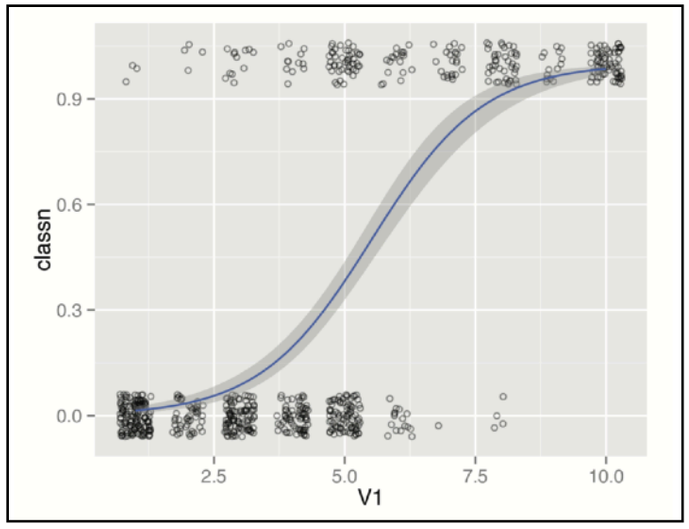
```r
# Track the indices to keep
keep_idx <- which(keep_rows)

# Order by distance
dists <- dists[keep_idx]
keep_idx <- keep_idx[order(dists)]

# Keep max number of rows
if (!is.null(maxpoints) && length(keep_idx) > maxpoints) {
  keep_idx <- keep_idx[seq_len(maxpoints)]
}

if (allRows) {
  # Add selected_ column if needed
  df$selected_ <- FALSE
  df$selected_[keep_idx] <- TRUE

} else {
  # If we don't keep all rows, return just the selected rows, sorted by
  # distance.
  df <- df[keep_idx, , drop = FALSE]
}
```

Data analyst

ID	V1	V2	V3	V4	class
1036172	2	1	1	1	benign
1041801	5	3	3	3	malignant
1043999	1	1	1	1	benign
1044572	8	7	5	10	malignant
1047630	7	4	6	4	malignant
1048672	4	1	1	1	benign
1049815	4	1	1	1	benign



Consumer



No surgery needed.

Decision/action

Tool maker

https://flic.kr/p/fXvJS6

Furniture maker

Consumer

Things you need

Tool maker

https://www.flickr.com/photos/mksfca/4083922894/

Furniture maker

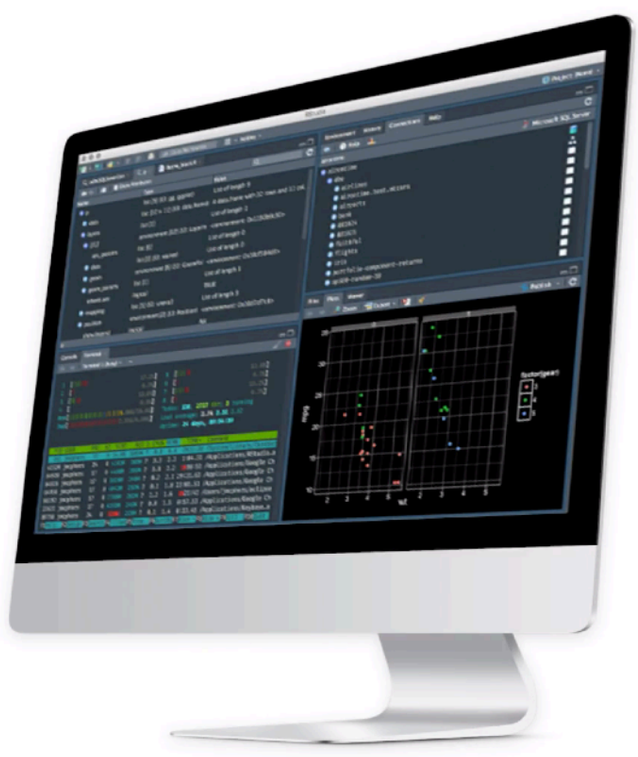
Consumer

Things you need

Tool maker

Data analyst

Consumer

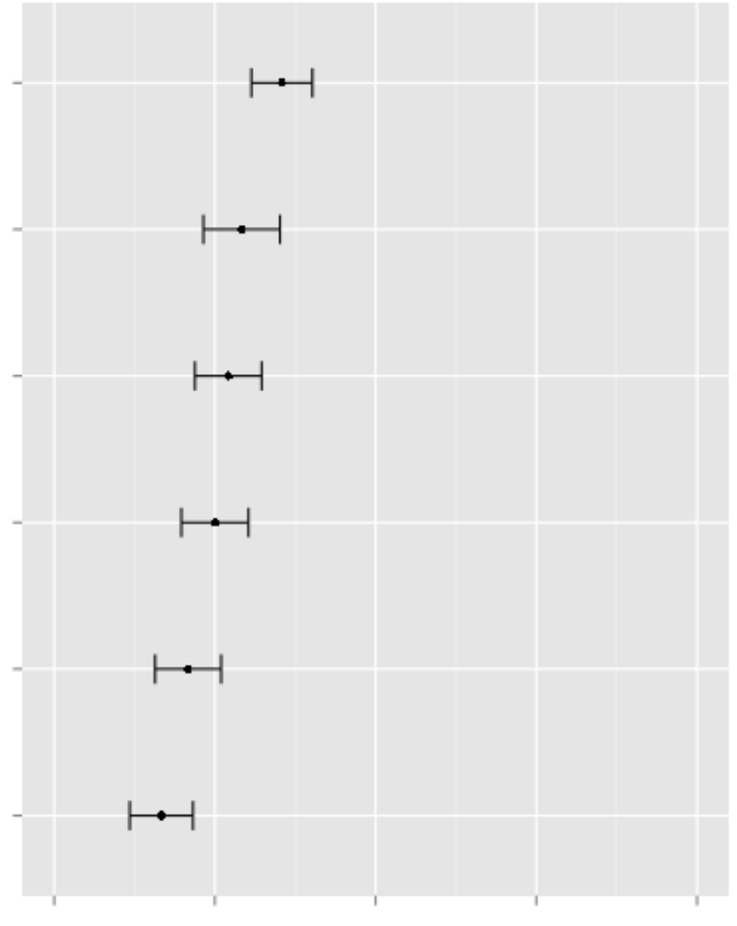


Using R to analyze data

subject	csqForm	c1	c2	c3	c4	c5	c6
490	B	1	1	2	1	2	2
529	B	2	2	1	1	2	1

form	new	original
A	1	1
A	2	2
A	3	3
A	4	4
A	5	5
A	6	6
B	1	5
B	2	6
B	3	4

subject	csqForm	qNum.new	qNum	rating
490	B	5	1	2
490	B	6	2	2
490	B	4	3	1
529	B	5	1	2
529	B	6	2	1
529	B	4	3	1



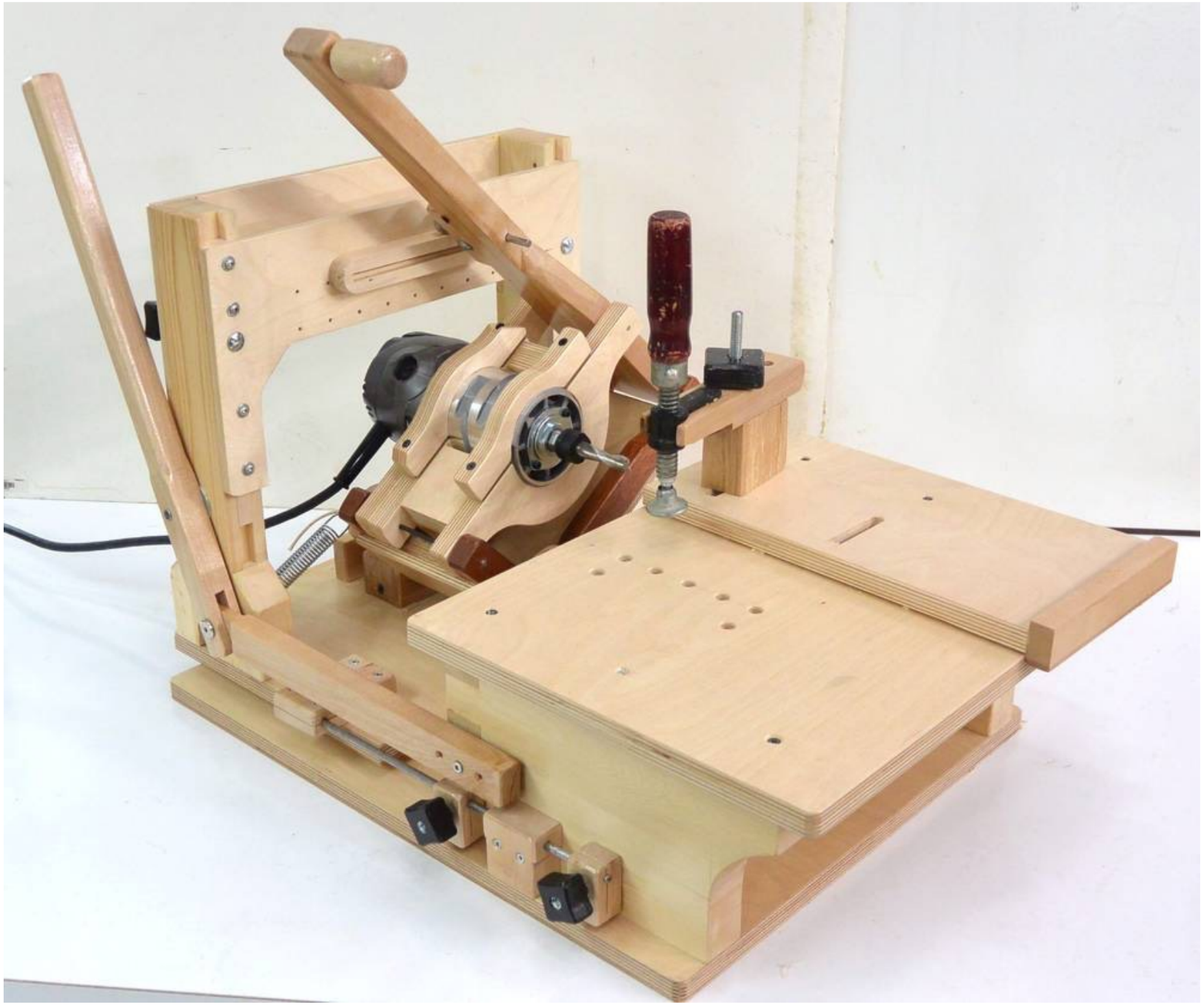
```r
library(reshape)
rawdata <- read.csv('data.csv', header=T)
csqOrder <- read.csv('csq_order.csv', header=T)

csqRawData <- melt(rawdata,
    id.vars = c("subject", "csqForm"),
    measure.vars = c("c1","c2","c3","c4","c5","c6"),
    variable_name = "qNum.new")

names(csqRawData)[names(csqRawData)=="value"] <- "rating"

levels(csqRawData$qNum.new)[levels(csqRawData$qNum.new)=="c1"] <- "1"
levels(csqRawData$qNum.new)[levels(csqRawData$qNum.new)=="c2"] <- "2"
levels(csqRawData$qNum.new)[levels(csqRawData$qNum.new)=="c3"] <- "3"
levels(csqRawData$qNum.new)[levels(csqRawData$qNum.new)=="c4"] <- "4"
levels(csqRawData$qNum.new)[levels(csqRawData$qNum.new)=="c5"] <- "5"
levels(csqRawData$qNum.new)[levels(csqRawData$qNum.new)=="c6"] <- "6"

# Merge the two data frames to get original numbers
csqData <- merge(csqRawData, csqOrder,
    by.x = c("csqForm","qNum.new"),
    by.y = c("form",   "new"),
    )

names(csqData)[names(csqData)=="original"] <- "qNum"

# Reorder the columns to something a little nicer
csqData <- csqData[, c(3,1,2,6,5,4)]

csqData <- csqData[order(csqData$subject, csqData$qNum), ]
```

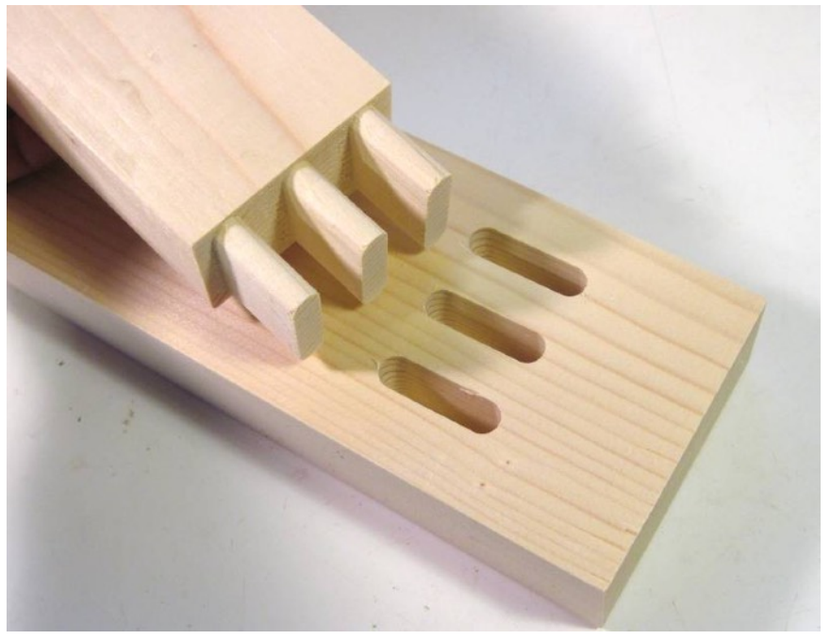
https://blog.mitchelloharawild.com/blog/hexwall/

```r
library(tidyverse)
rawdata <- read.csv('data.csv', header=T)
csqOrder <- read.csv('csq_order.csv', header=T)

csqRawData <- rawdata %>%
  pivot_longer(cols = c1:c6, names_to = "qNum.new",
               values_to = "rating") %>%
  select(subject, csqForm, qNum.new, rating) %>%
  mutate(qNum.new = recode(qNum.new,
                           c1=1, c2=2, c3=3, c4=4, c5=5, c6=6))

csqData <-
  inner_join(csqRawData, csqOrder,
             by = c("csqForm" = "form", "qNum.new" = "new")) %>%
  select(subject, csqForm, qNum.new, story, qNum = "original",
         rating) %>%
  arrange(subject, qNum)
```

Making your own tools


4.0Ah





https://woodgears.ca/pantorouter/

```r
do <- function(.data, ...) {
  UseMethod("do")
}
#' @export
do.default <- function(.data, ...) {
  do_(.data, .dots = compat_as_lazy_dots(...))
}
#' @export
#' @rdname se-deprecated
do_ <- function(.data, ..., .dots = list()) {
  UseMethod("do_")
}

#' @export
do.NULL <- function(.data, ...) {
  NULL
}
#' @export
do_.NULL <- function(.data, ..., .dots = list()) {
  NULL
}

# Helper functions ------------------------------------------------

label_output_dataframe <- function(labels, out, groups) {
  data_frame <- vapply(out[[1]], is.data.frame, logical(1))
  if (any(!data_frame)) {
    bad("Results {bad} must be data frames, not {first_bad_class}",
      bad = fmt_comma(which(!data_frame)),
      first_bad_class = fmt_classes(out[[1]][[which.min(data_frame)]])
    )
  }

  rows <- vapply(out[[1]], nrow, numeric(1))
  out <- bind_rows(out[[1]])

  if (!is.null(labels)) {
    # Remove any common columns from labels
    labels <- labels[setdiff(names(labels), names(out))]

    # Repeat each row to match data
    labels <- labels[rep(seq_len(nrow(labels)), rows), , drop = FALSE]
    rownames(labels) <- NULL
```

```r
label_output_list <- function(labels, out, groups) {
  if (!is.null(labels)) {
    labels[names(out)] <- out
    rowwise(labels)
  } else {
    class(out) <- "data.frame"
    attr(out, "row.names") <- .set_row_names(length(out[[1]]))
    rowwise(out)
  }
}

named_args <- function(args) {
  # Arguments must either be all named or all unnamed.
  named <- sum(names2(args) != "")
  if (!(named == 0 || named == length(args))) {
    abort("Arguments must either be all named or all unnamed")
  }
  if (named == 0 && length(args) > 1) {
    bad("Can only supply one unnamed argument, not {length(args)}")
  }

  # Check for old syntax
  if (named == 1 && names(args) == ".f") {
    abort("do syntax changed in dplyr 0.2. Please see documentation for details")
  }

  named != 0
}
```

```
install.packages("dplyr")
```

Functions

A collection of functions in a .R file

R package on GitHub

R package on CRAN

Making tools: some lessons learned

A good API is hard to design.

Renaming factor levels

```
levels(df$col)[levels(df$col)=='c1'] <- '1'
levels(df$col)[levels(df$col)=='c2'] <- '2'
levels(df$col)[levels(df$col)=='c3'] <- '3'
```

```
library(plyr)
revalue(df$col, c(c1='1', c2='2', c3='3'))
```

```
library(dplyr)
recode(df$col, c1='1', c2='2', c3='3'))
```

```
library(forcats)
fct_recode(df$col, '1'='c1', '2'='c2', '3'='c3'))
```

Shiny: renderImage

```
renderImage(expr, deleteFile = TRUE)
```

```
output$image <- renderImage(
  {
    # Code to generate a PNG
  },
  deleteFile = FALSE
)
```

Shiny: server.R and ui.R

```r
# server.R
shinyServer(function(input, output) {
  # ...
})

# ui.R
shinyUI(fluidPage(
  # ...
))
```

```r
# app.R
shinyApp(
  ui = fluidPage(
    #...
  ),
  server = function(input, output) {
    # ...
  }
)
```

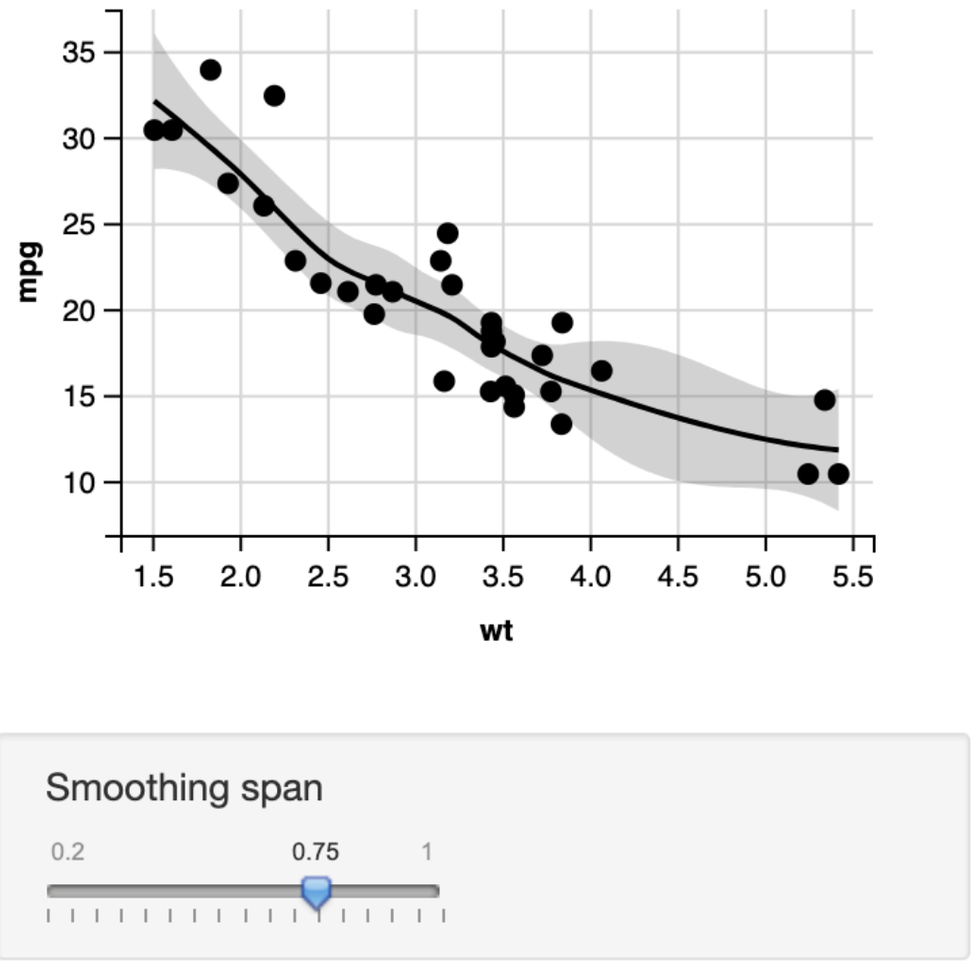
Why are good APIs hard to design?

- Don't have enough experience to decide on a consistent pattern.
- Don't fully understand how people will use a function.
- Too familiar with old ways of looking at things.
- Not familiar enough with old ways of looking at things.

If it takes more time to explain how something works than it does to simplify it, then simplify it.

Sometimes a project can be too ambitious.

- ggvis: *interactive* grammar of graphics
  - different syntax from ggplot2
  - rendered in the browser, using Vega (JavaScript) library



- Why didn't it work?
  - Vega made some things easy, other things impossible
  - ggvis wasn't better than ggplot2 for most users

If your code seems slower than it should be, don't guess about why; use a profiler.

(but it probably has something to do with memory churn)

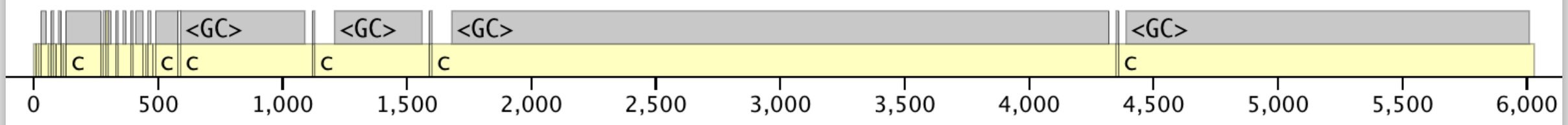
```r
v <- numeric(0)
for (i in 1:40000) {
  x <- rnorm(20)
  x <- mean(x)
  v <- c(v, x)
}
```

```
profvis::profvis({
  v <- numeric(0)
  for (i in 1:40000) {
    x <- rnorm(20)
    x <- mean(x)
    v <- c(v, x)
  }
})
```

Flame Graph | Data | Options

<expr>

	Memory		Time
1 `profvis::profvis({`			
2 `  v <- numeric(0)`			
3 `  for (i in 1:40000) {`			
4 `    x <- rnorm(20)`	-83.9	136.0	140
5 `    x <- mean(x)`	-45.5	12.7	30
6 `    v <- c(v, x)`	-4392.4	4369.5	5860
7 `  }`			
8 `})`			
9			

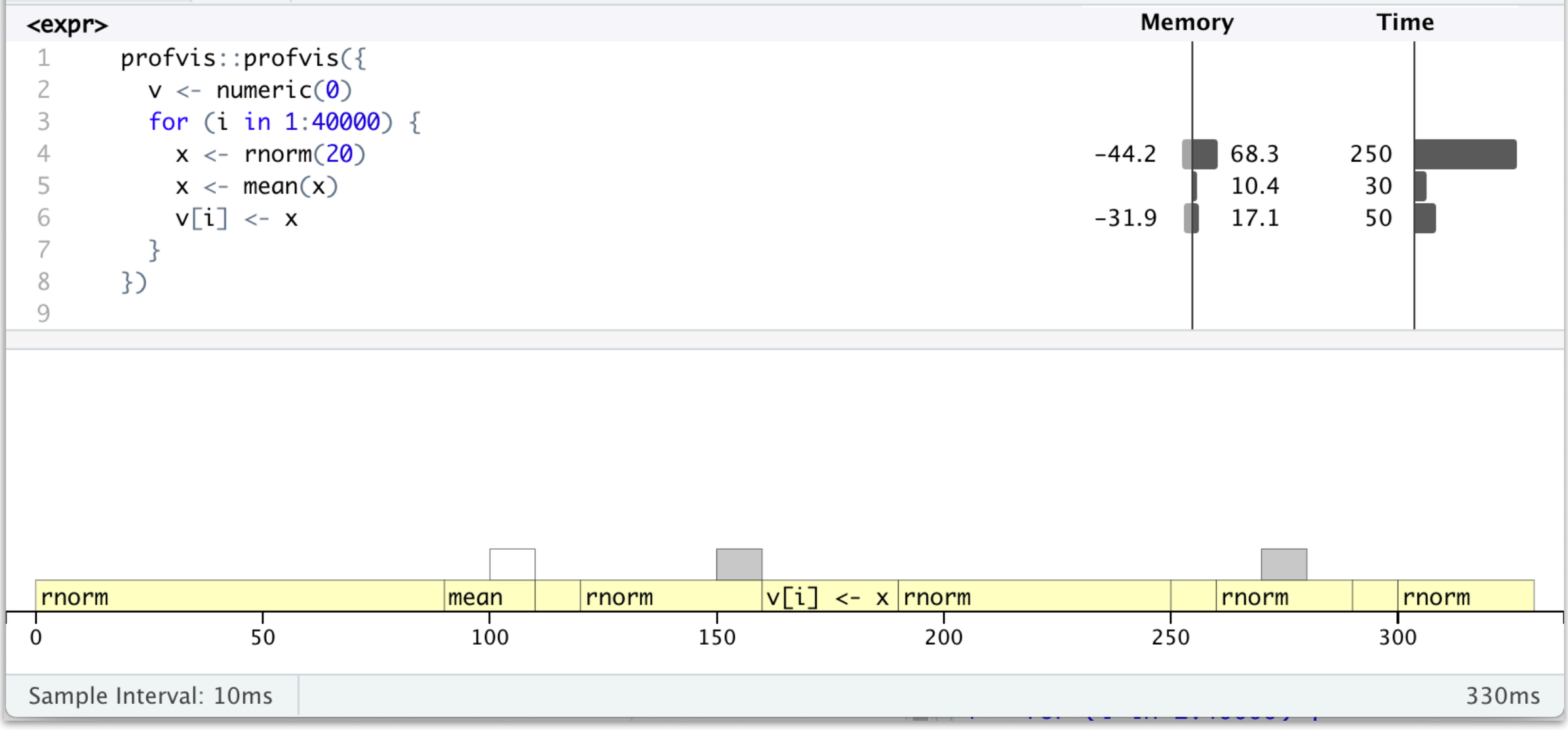


Sample Interval: 10ms

6030ms

```r
v <- numeric(0)
for (i in 1:40000) {
  x <- rnorm(20)
  x <- mean(x)
  v[i] <- x
}
```

Flame Graph | Data | Options



Sample Interval: 10ms

330ms

Think about scale when deciding what to work on.

Sometimes small improvements require deep digging.

It's not a complete waste of time if you learn something from it.

Thank you!