From data consumers to tool makers

Winston Chang

RStudio

Tool maker

https://flic.kr/p/fXvJS6

Furniture maker

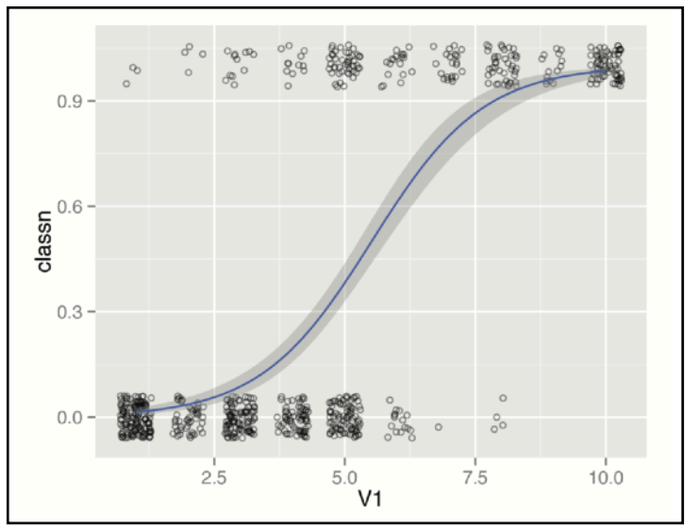
Consumer

Different roles for working with data

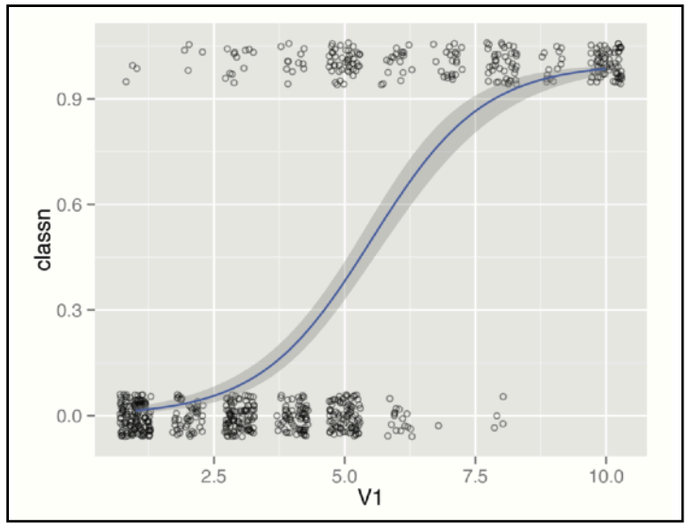
Tool maker

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.

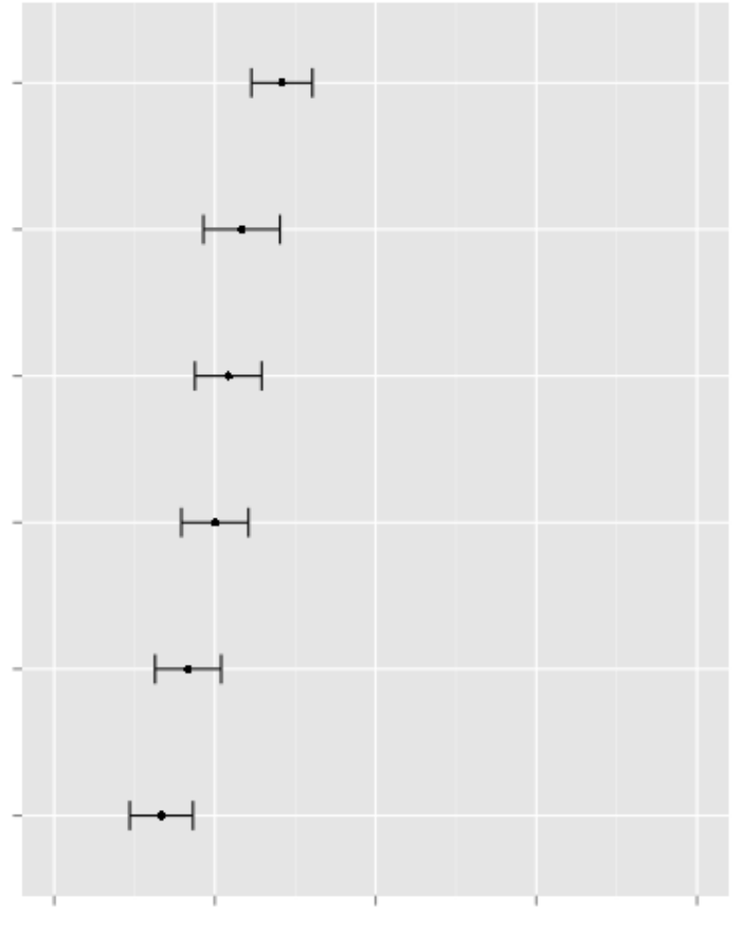
Decision/action

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

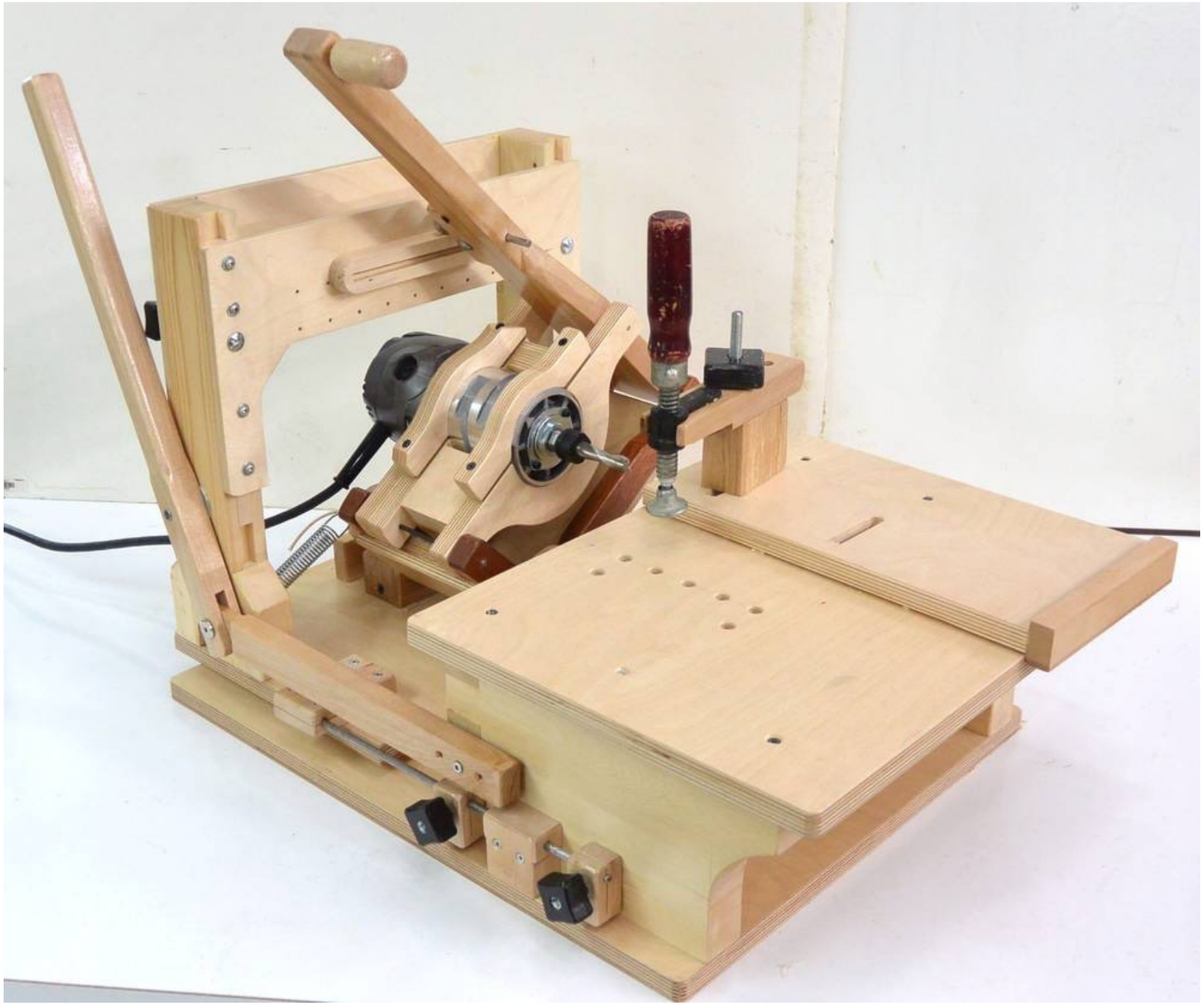
Functions

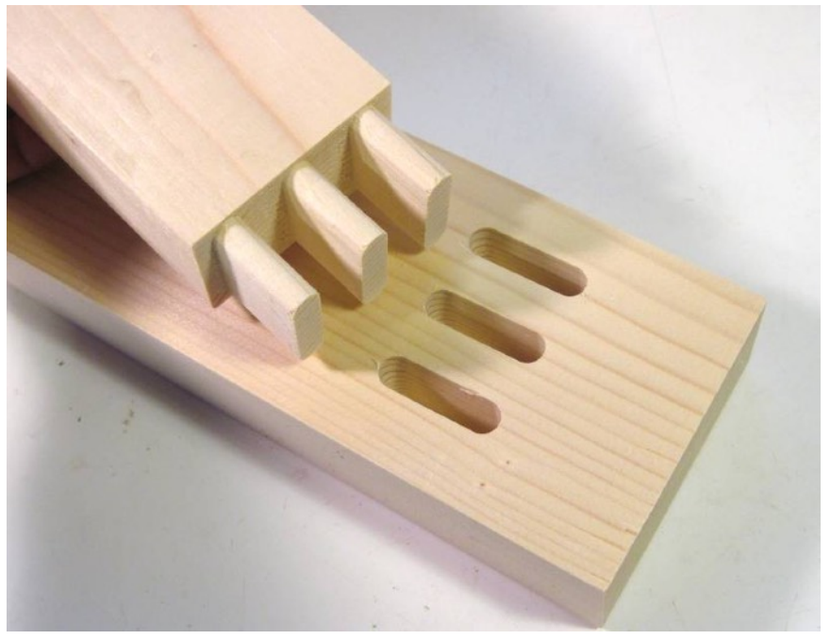
A collection of functions in a .R file

R package for internal use

R package on CRAN


4.0Ah





https://woodgears.ca/pantorouter/

Making tools: lessons learned

A good API is hard to design

Renaming factor levels

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

```r
# plyr
revalue(df$col, c(c1='1', c2='2', c3='3', c4='4'))
```

```r
# dplyr
recode(df$col, c1='1', c2='2', c3='3', c4='4'))
```

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

Reshaping data

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

```
library(reshape2)
melt(table4a,
     measure.vars = c('1999', '2000'),
     variable.name = 'year',
     value.name = 'cases')
```

```
library(tidyr)
gather(table4a, '1999', '2000', key = 'year',
       value = 'cases')


# In upcoming version of tidyr
pivot_longer(table4a, '1999':'2000',
             names_to = 'year', values_to = 'cases')
```

Shiny: update*Input functions

```
updateTextInput <-
  function (session, inputId, value = NULL) { ... }

updateTextInput(session, "myslider", value = 123)
```

```
showModal <-
  function(ui, session = getDefaultReactiveDomain()) {
    ...
  }

showModal(modalDialog("Hello"))
```

Shiny: renderImage

```
renderImage <- function (expr, deleteFile = TRUE)
```

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

Why are good APIs hard to design?

- Don't have enough experience to decide on a consistent pattern.
- Don't have enough technical skill to make desired API work.
- Don't fully understand how people will use a function.

If something is slower than it should be, it probably has something to do with memory allocation

Growing a vector

```
x <- numeric(0)
for (i in 1:50000) {
  x <- c(x, i*2)
}
```

```
x <- numeric(0)
for (i in 1:50000) {
  x[i] <- i*2
}
```

Growing a data frame

```r
df <- data.frame(x = numeric(0), y = numeric(0))
```

```r
for (i in 1:10000) {
  df[i, c("x", "y")] <- c(i, i)
}
```

Sometimes you need to dig deep to make a small improvement

A real-world problem: concatenating strings

Shiny UI code generates HTML

```
fluidPage(
  sidebarPanel(
    sliderInput("n", "Observations", 1, 100, 50)
  ),
  mainPanel(
    plotOutput("plot")
  )
)
```

```
<div class="container-fluid">
  <div class="col-sm-4">
    <form class="well">
      <div class="form-group shiny-input-container">
        <label class="control-label" for="n">Number of observations</label>
        <input class="js-range-slider" id="n" data-min="1" data-max="100"
data-from="50" data-step="1" data-grid="true" data-grid-num="9.9" data-grid-
snap="false" data-prettify-separator="," data-prettify-enabled="true" data-
keyboard="true" data-data-type="number"/>
      </div>
    </form>
  </div>
  <div class="col-sm-8">
    <div id="plot" class="shiny-plot-output" style="width: 100% ; height:
400px"></div>
  </div>
</div>
```

Concatenating strings

```
str <- ''
str <- paste(str, '<div class="container-fluid">')
str <- paste(str, '<div class="col-sm-4">')
str <- paste(str, '<form class="well">')
str <- paste(str, '<div class="form-group shiny-input-container">')
```

How slow?

```
strs <- as.character(rnorm(10000))
#> [1] "-0.194947715309794"   "0.298308969181886"    "-1.26964914219051"
#> [4] "0.887132612093076"    "0.0788425606480748"   "-0.187354117656301"
#> [7] "-0.0819054749614721"  "-1.21418664082276"    "2.2179901491329"
#> ...
```

```
s <- ''
for (str in strs) {
  s <- paste(s, str)
}
```

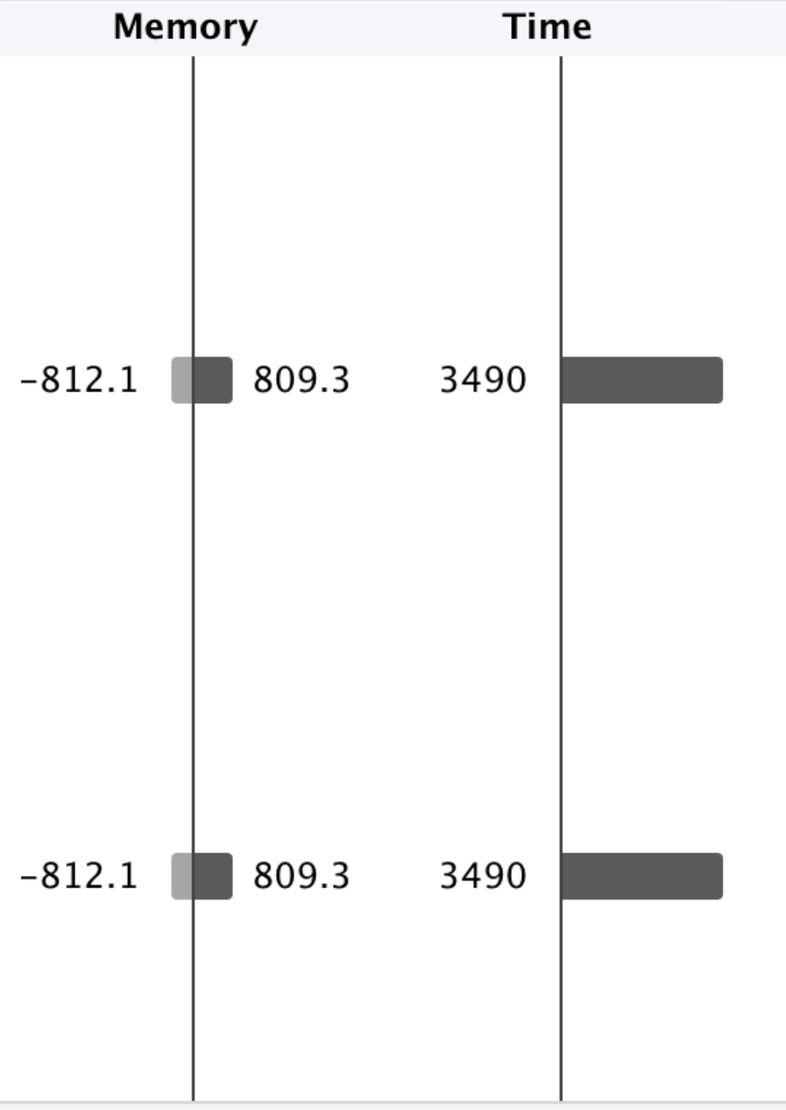
```
system.time({
  s <- ''
  for (str in strs) {
    s <- paste(s, str)
  }
})
```

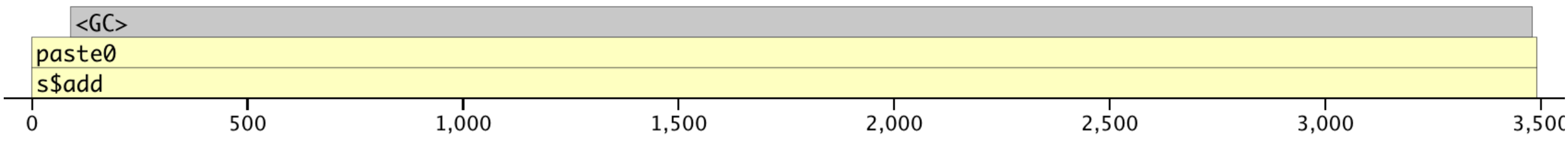
Profiling with profvis

Flame Graph | Data | Options

<expr>

```
1     profvis({
2       string_builder <- function() {
3         s <- ''
4
5         list(
6           add = function(x) {
7             s <<- paste0(s, x)
8           },
9           get = function() {
10            s
11          }
12        )
13      }
14
15      s <- string_builder()
16      for (str in strs) {
17        s$add(str)
18      }
19      s$get()
20    })
21
```





Sample Interval: 10ms | 3490ms

```
string_builder2 <- function() {
  strings <- character(0)

  list(
    add = function(x) {
      strings[length(strings)+1] <<- x
    },
    get = function() {
      paste(strings, collapse = "")
    }
  )
}

s <- string_builder2()
for (str in strs) {
  s$add(str)
}
s$get()
```

- Roles: Consumer, data analyst, tool maker
- With software, you have everything you need to move between roles
- As a tool maker:
  - Having people use your tools is gratifying
  - Designing a "perfect" API may be unrealistic, but you can strive for designing a good one
  - Keep an open mind for learning how things work at a deeper level