# Row-wise Computation

Data Wrangling and Husbandry

04/06/2020

# Rows versus columns

- Columns in R have a special structure
  - data frames are lists of columns
- This is great for data analysis, but
- working with rows is a bit more painful

The question came up on Twitter about how best to convert an R dataframe to a list of rows, since that's how the d3 Javascript graphing libraries like the data

*These notes quote extensively from Jenny Bryan's slides https://rstd.io/row-work*

You could use a loop

```
df <- SOME DATA FRAME
out <- vector(mode = "list", length = nrow(df))
for (i in seq_along(out)){
  out[[i]] <- as.list(df[i, , drop = FALSE])
}
out
```

However, it's

- generally faster
- generally easier to understand

if you take advantage of the for loops that someone else wrote

In this particular case, there's a specific special-purpose function

```
df <- SOME DATA FRAME
out <- purrr::transpose(df)
```

# Generally best option

- Use vectorized functions
- Many R functions are vectorized—don't use them on a row one-by-one

```r
new_df <- tribble(
    ~ name, ~ age,
    "Reed", 14,
    "Wesley", 12,
    "Eli", 12,
    "Toby", 1
)

df <- new_df
paste(df$name[1], "is", df$age[1], "years old")
```

```
## [1] "Reed is 14 years old"
```

```r
paste(df$name, "is", df$age, "years old")
```

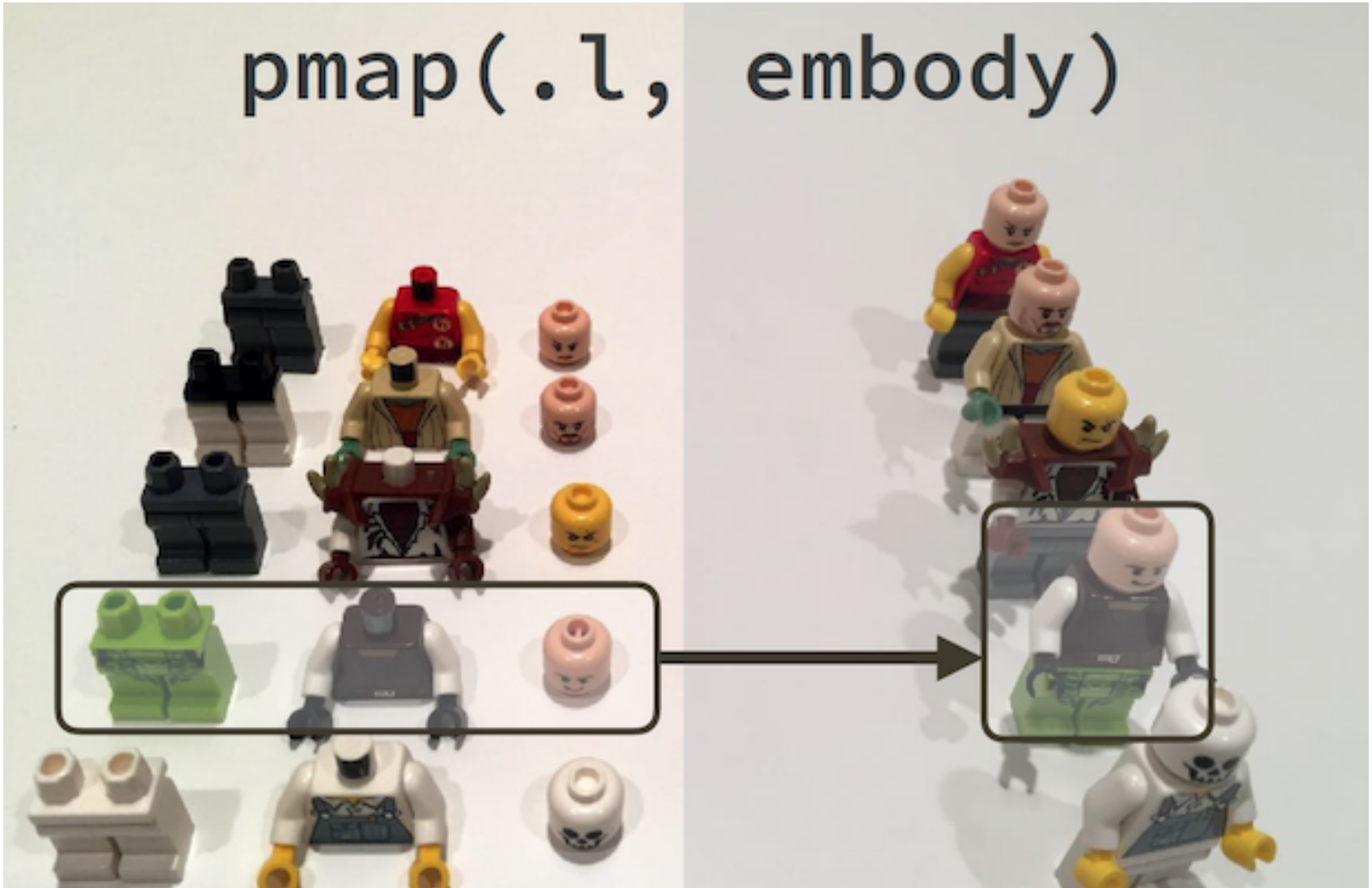
```
## [1] "Reed is 14 years old"   "Wesley is 12 years old" "E
## [4] "Toby is 1 years old"
```

For this particular sort of problem, there is a specialized function `glue()`:

```r
df %>%
```

# Next best option

- Use `map` functions
- `map(.x, .f)` itself applies a function to a list
- `pmap(.l, .f)` applies `.f` to every "tuple" in `.l`



`pmap(.l, .f)` is essentially a really efficient coding of

```
.l <- LIST OF LENGTH - N VECTORS
out <- vector(mode = "list", length = N)
for (i in seq_along(out)) {
  out[[i]] <- .f(.l[[1]][[i]], .l[[2]][[i]], ...)
}
out
```

There is a detailed set of examples here