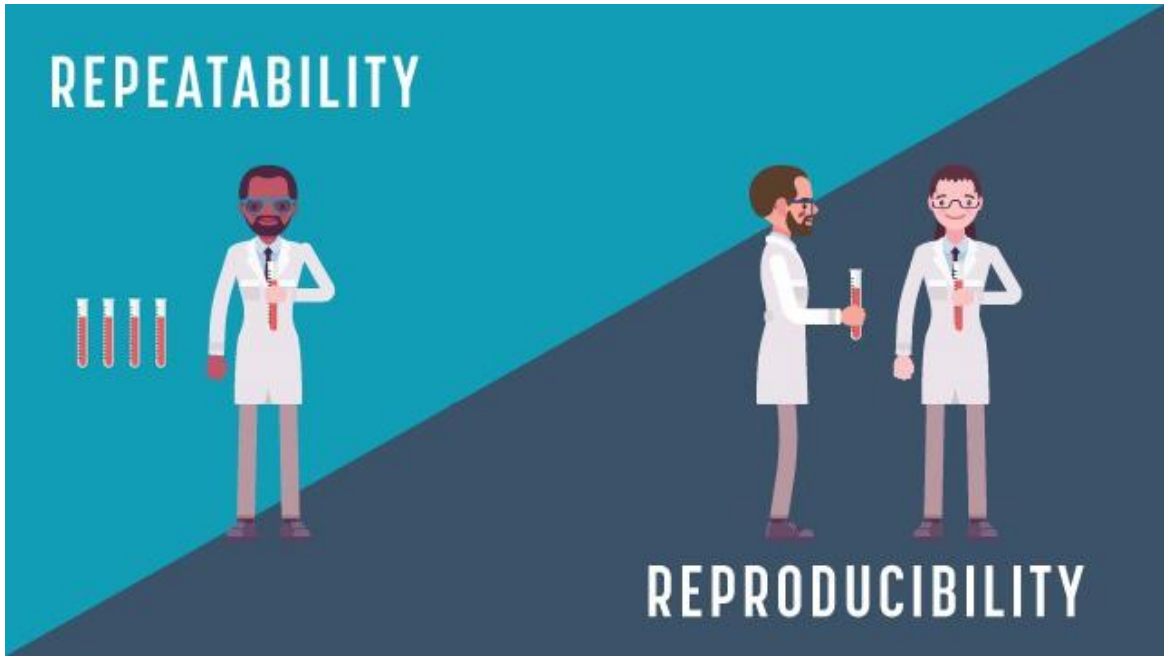
Reproducible Code & Visualization Best Practices

Lara N. Southard, PhD

Senior Research Scientist at Pearson

Reproducible Code

General tips and `ProjectTemplate`

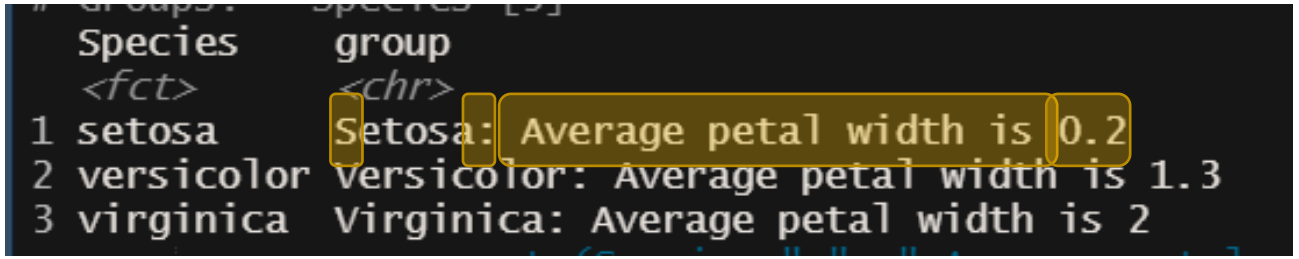


Medium blog post: [Quick tips for making reproducible code in R](lara-southard.medium/com)

paste() & paste0()

Concatenate vectors after converting to character.

```
iris %>%
  group_by(Species) %>%
  mutate(group = paste(str_to_title(Species),":",
                       " Average petal width is ",
                       paste0(format(
                         round(
                           mean(Petal.Width),
                           digits = 1),
                         big.mark = ",",
                         trim = T)),
                       sep= "")) %>%
  distinct(group)
```



Species	group
<fct>	<chr>
1 setosa	Setosa: Average petal width is 0.2
2 versicolor	Versicolor: Average petal width is 1.3
3 virginica	Virginica: Average petal width is 2

pull()

Extract a single column

```
iris %>%
  select(Species, Petal.Width) %>%
  group_by(Species) %>%
  summarise(avg.width = mean(Petal.Width)) %>%
  ungroup() %>%
  filter(Species == "setosa") %>%
  pull(avg.width)
```

Grab only the columns you need

Perform the calculation

Filter to one value or column

Pull that individual value or column

pull()

Extra handy in RMarkdown for in-line coding

In R Markdown:

```
` r format(round(iris %>%
                 select(Species, Petal.Width) %>%
                 group_by(Species) %>%
                 summarise(avg.width = mean(Petal.Width)) %>%
                 ungroup() %>%
                 filter(Species == "setosa") %>%
                 pull(avg.width), digits =3),
        big.mark = ",")`
```

Reducing & including dataframes

```
iris %>%
  select(Species, Petal.Width) %>% #reduce to necessary columns only
  filter(Petal.Width > 1) %>% #some filtering critera
  group_by(Species, Petal.Width) %>%
  summarise(count = n()) %>%  #one way of many to get a count
  ungroup() %>%
  #Now plot!
  ggplot(aes(x = Petal.Width, y = count, fill = Species))+
  geom_col(position = position_dodge())
```

Instead of saving every dataframe, just pipe it into your plots. This will make future changes will be much easier.

Model and Analysis

Leverage %>% in the `data =` argument of various model functions in R (e.g. `glm()`, `lmer()`, etc.).

Here's a crude example:

```
glm(Sepal.Length ~ Species,
    data = iris %>%
      filter(Petal.Width > 0))
```

!Another tip!
Use `set.seed()` any time your code has a random process to repeat the same results in the future.

Level-Up Reproducible Code with `library(ProjectTemplate)`

http://projecttemplate.net/getting_started.html

What is `library(ProjectTemplate)`?

- Automatically create a directory for a project
- Easy use of organization
- Easy to sharing code and track versioning
- Cached items

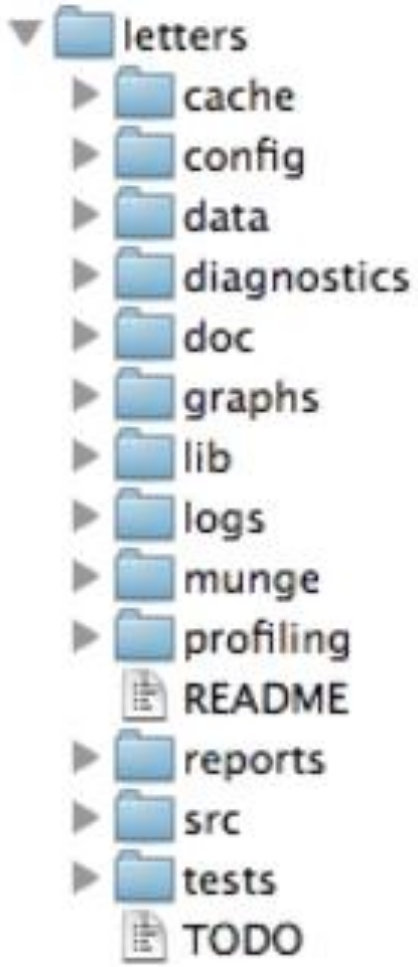
Getting started in `library(ProjectTemplate)`

```
> library('ProjectTemplate')
> create.project('letters')
```

Exit R and change into the directory for letters that you just set up

`$ cd letters` or just navigate in file explorer/finder

Then you can open up your new project:



Open R project or setwd() to project location (e.g. /letters/)

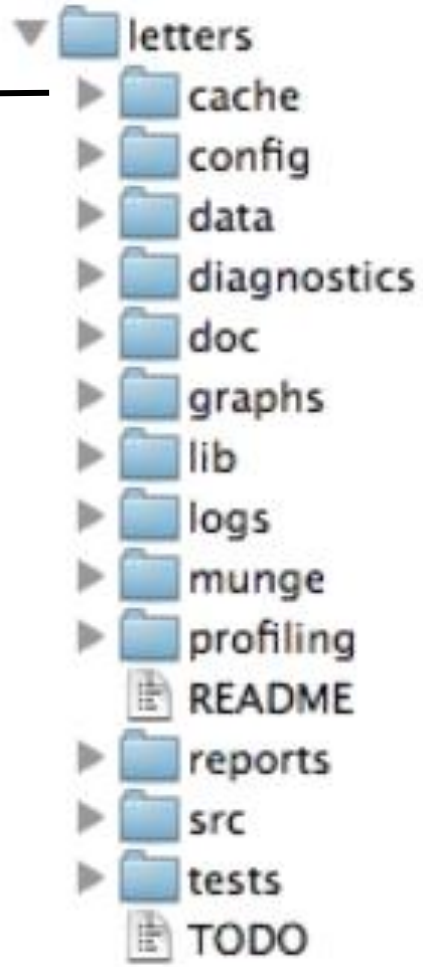
```
> library('ProjectTemplate')
> load.project()
```

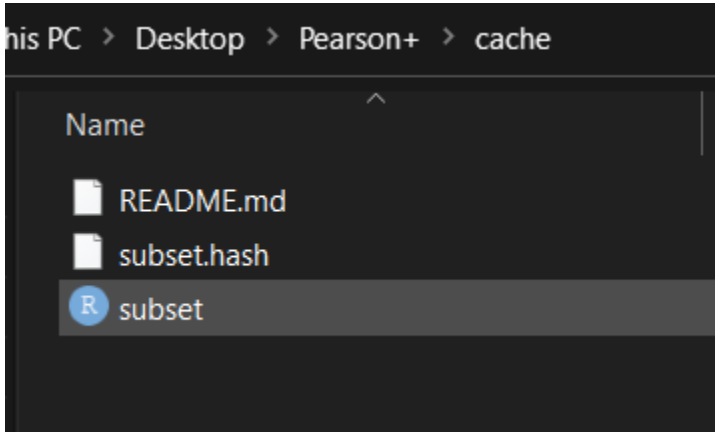
All cached data, functions, and libraries will be loaded. All data will be easily accessible and old scripts.

Folders of `library(ProjectTemplate)`

Stores cached dataframes

```
> cache("df")
```

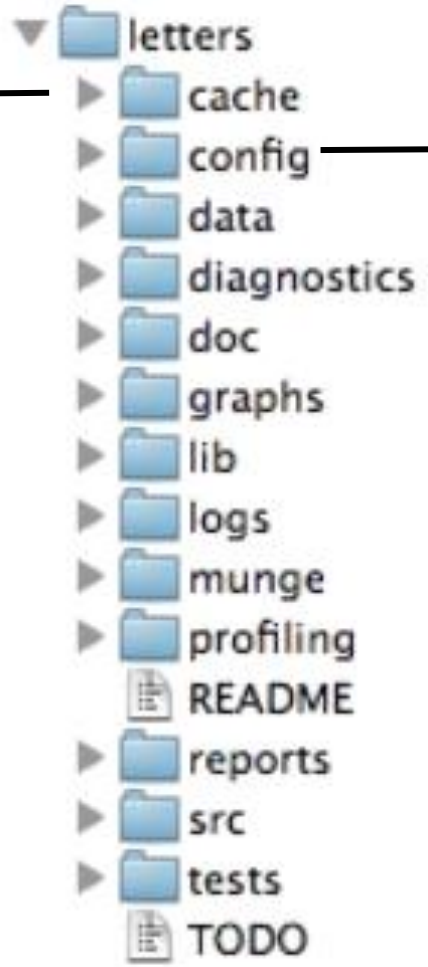




Folders of `library(ProjectTemplate)`

Stores cached dataframes

```
> cache("df")
```



Specify how you want your project to load

global - Notepad

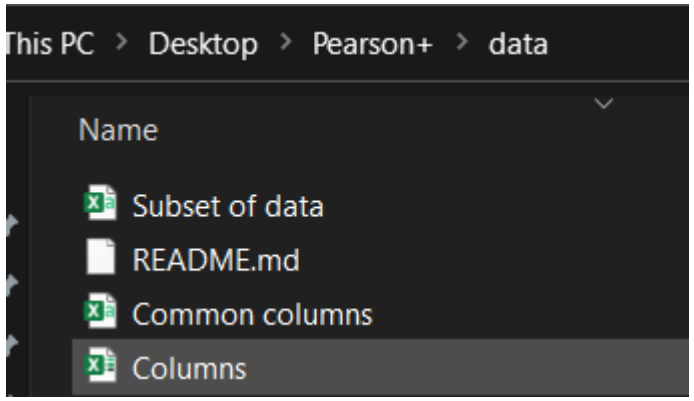
File Edit Format View Help

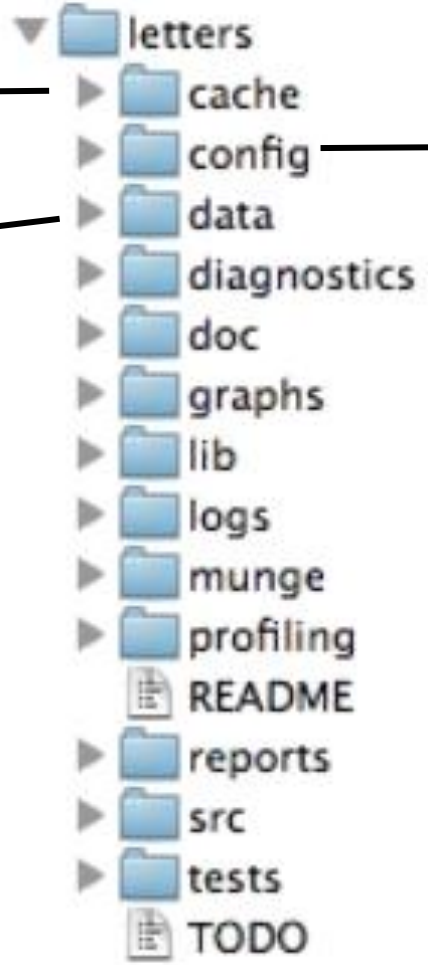
```
version: 0.10.1
data_loading: FALSE
data_loading_header: TRUE
data_ignore:
cache_loading: FALSE
recursive_loading: FALSE
munging: FALSE
logging: FALSE
logging_level: INFO
load_libraries: TRUE
libraries: reshape2, plyr, tidyverse, stringr,
as_factors: FALSE
tables_type: tibble
attach_internal_libraries: FALSE
cache_loaded_data:  TRUE
sticky_variables: NONE
underscore_variables: TRUE
cache_file_format: RData
```

Folders of `library(ProjectTemplate)`

Stores cached dataframes
`> cache("df")`

Data files (ex. .xlsx, .csv, JSON, etc.)





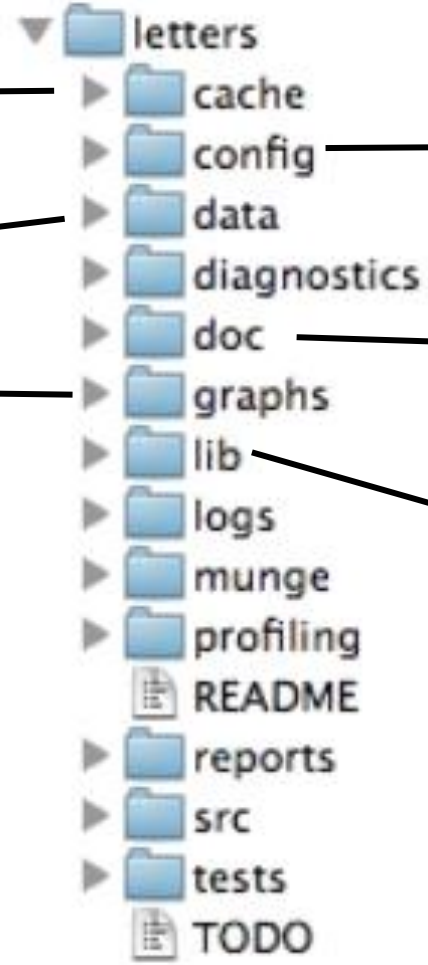
Specify how you want your project to load

Folders of `library(ProjectTemplate)`

Stores cached dataframes
`> cache("df")`

Data files (ex. .xlsx, .csv, JSON, etc.)

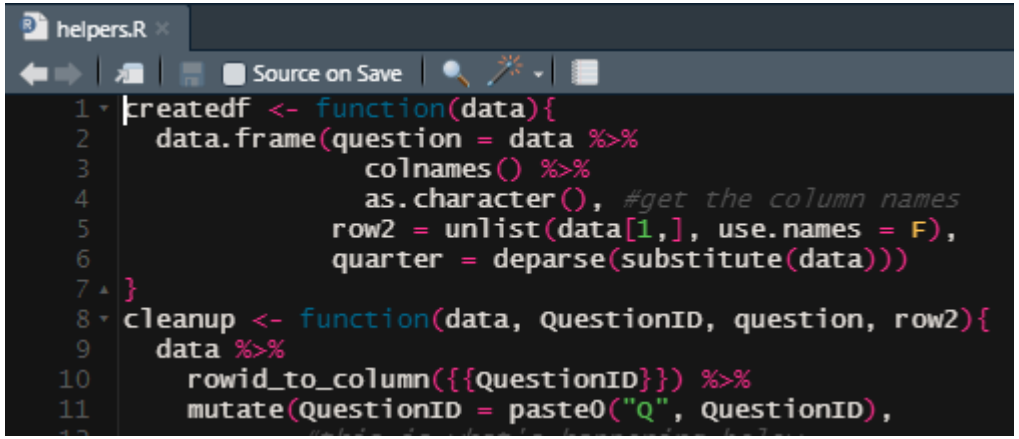
Optional location to store visualizations



Specify how you want your project to load

Project notes

/lib/helpers.R place to store functions to be automatically loaded into your environment



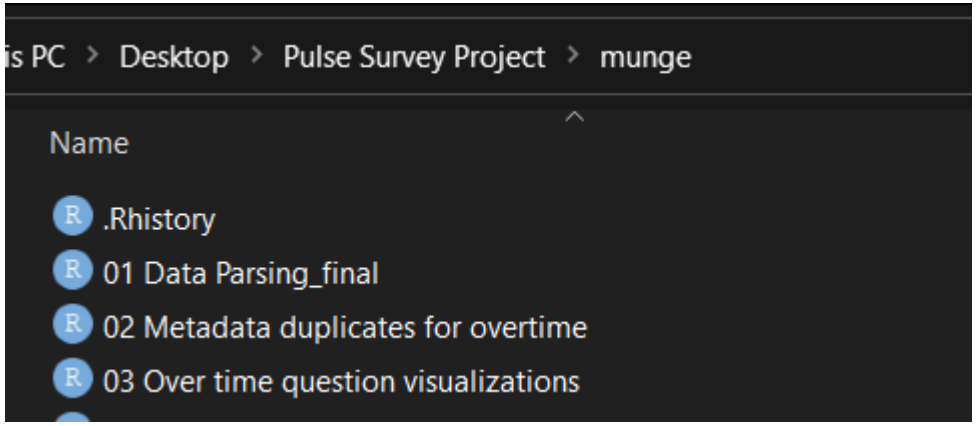
Folders of `library(ProjectTemplate)`

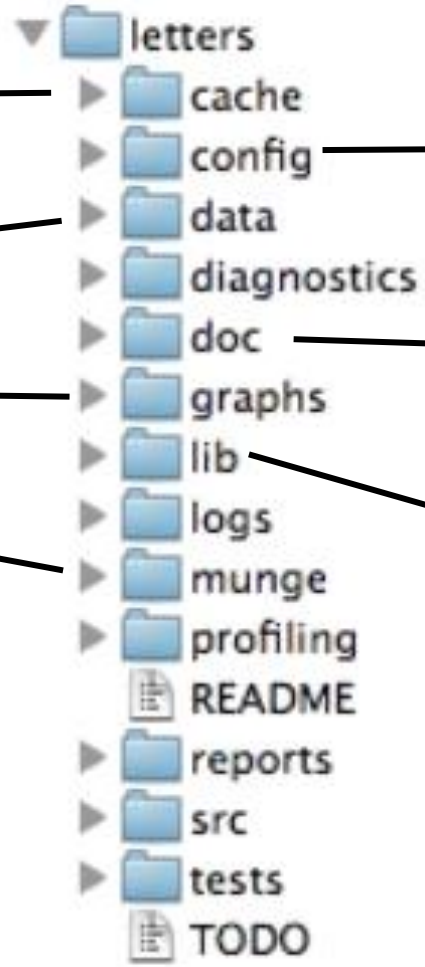
Stores cached dataframes
`> cache("df")`

Data files (ex. .xlsx, .csv, JSON, etc.)

Optional location to store visualizations

Place to store prep files for analysis (data cleaning, wrangling, etc.)





Specify how you want your project to load

Project notes

/lib/helpers.R place to store functions to be automatically loaded into your environment

Folders of `library(ProjectTemplate)`

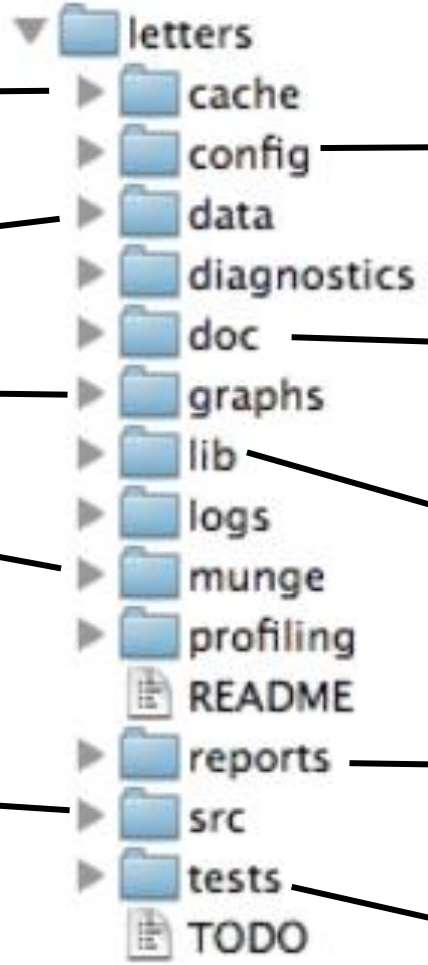
Stores cached dataframes
`> cache("df")`

Data files (ex. .xlsx, .csv, JSON, etc.)

Optional location to store visualizations

Place to store prep files for analysis (data cleaning, wrangling, etc.)

Where I personally like to store modeling scripts



Specify how you want your project to load

Project notes

/lib/helpers.R place to store functions to be automatically loaded into your environment

Store R Markdown file to write up reports. Can refer to any cached item or code chunks

Failed attempts that may be useful for other things

Put it all together: A beefy example

```
```{r, setup, include = F}
# If you're using ProjectTemplate, these lines will make sure all your packages
# and saved objects are loaded
library(ProjectTemplate)
load.project()
# Other useful defaults; adjust as desired
knitr::opts_chunk$set(echo = F, message = F, wa
options(knitr.kable.NA = "")
```

Brief Report: Fall 2020 Enrollment Analysis

Background & Methodology
Each term, Data Science & Analytics evaluates if the enrollment of current term (fall 2020) is systematically different from the year prior (fall 2019).
Third-week headcount and FTE were used to compare students enrolled in fall 2019 (total headcount = `r format(res1453 %>% distinct(group, PIDM) %>%  filter(group == "Fall 2019") %>% nrow(), big.mark = ",")`; total FTE = `r format(round(res1453 %>% group_by(group) %>% summarise(sum = sum(fte)) %>% ungroup() %>% filter(group == "Fall 2019") %>% pull(sum), digits = 0), big.mark = ",")`) to the fall 2020 term (total headcount = `r format(res1453 %>% distinct(PIDM, group) %>% filter(group == "Fall 2020") %>% nrow(), big.mark = ",")`; total FTE = `r format(round(res1453 %>% group_by(group) %>% summarise(sum = sum(fte)) %>% ungroup() %>% filter(group == "Fall 2020") %>% pull(sum), digits = 0), big.mark = ",")`).
```

Load in project in R Markdown

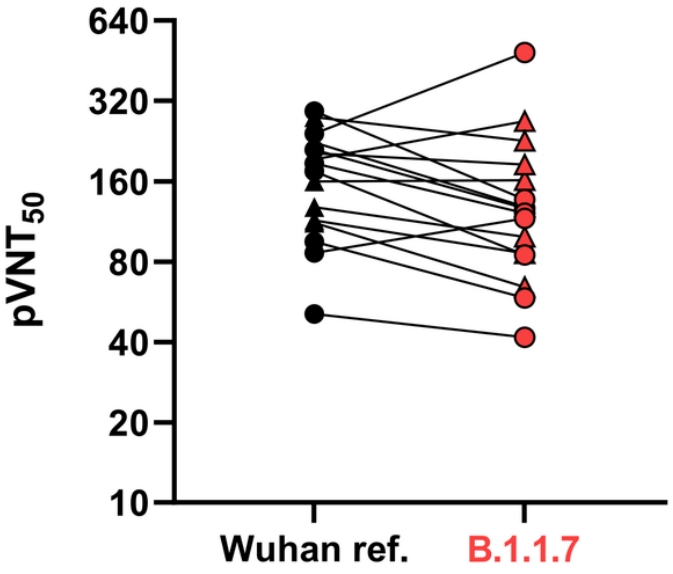
Example of `pull()` function to reduce human error

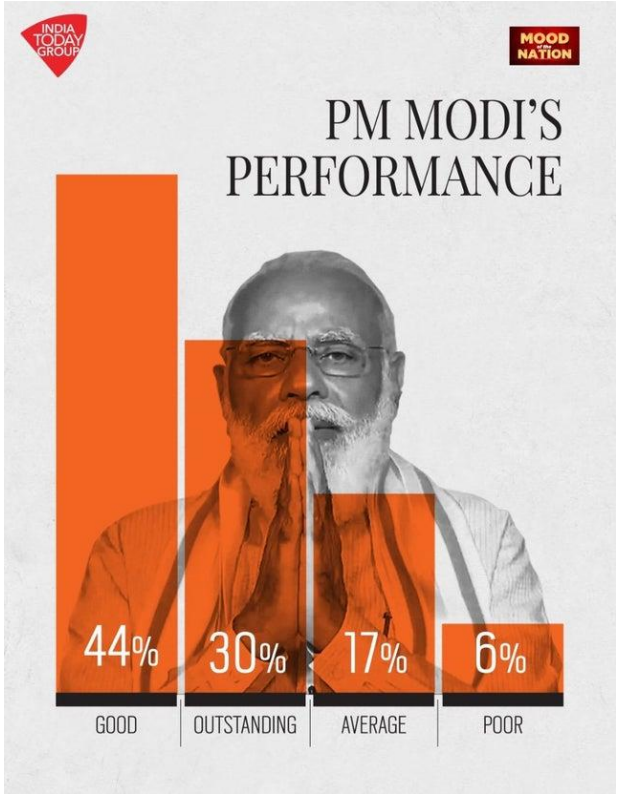
Immediately and directly use cached dataframes using functions from loaded libraries

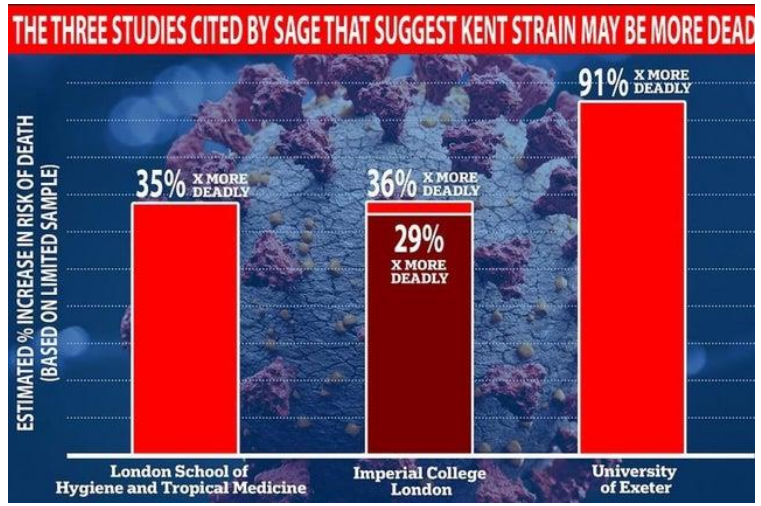
Visualizations: Best Practices

Medium blog post: [Level up your data visualizations with Human Factors Principles of Display]

lara-southard.medium/com

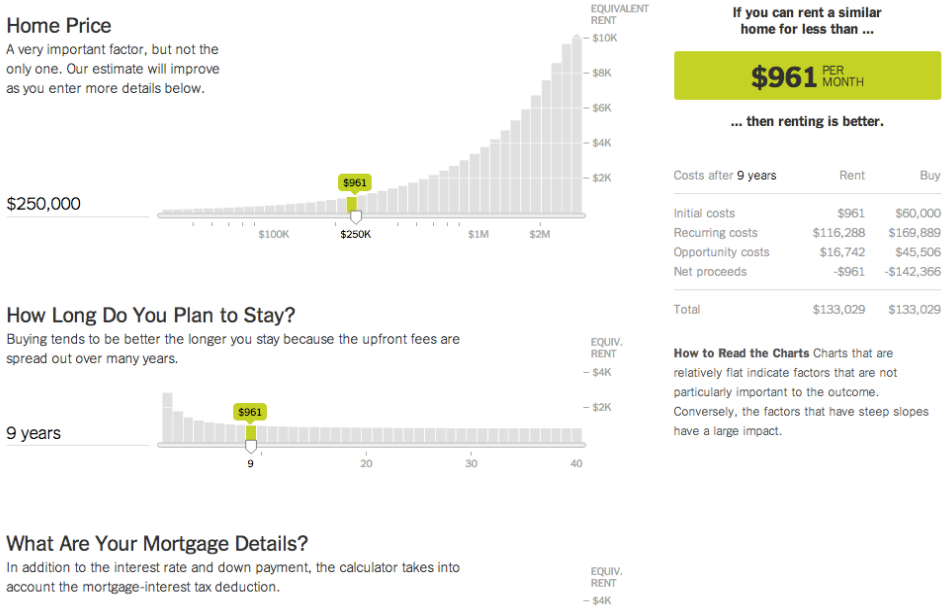


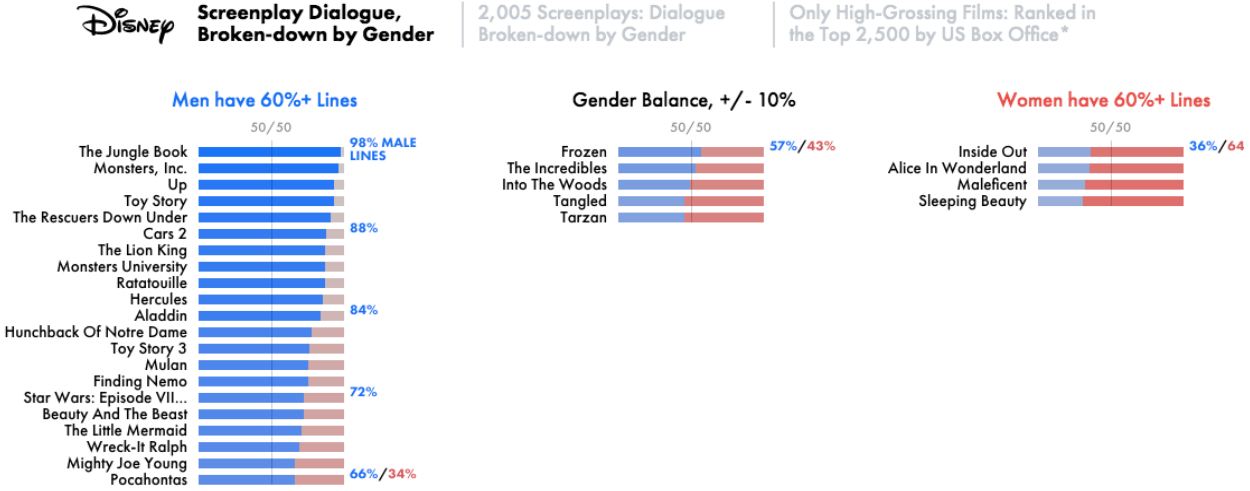




We know a bad visualization when we see one…

Great visualizations minimize cognitive load





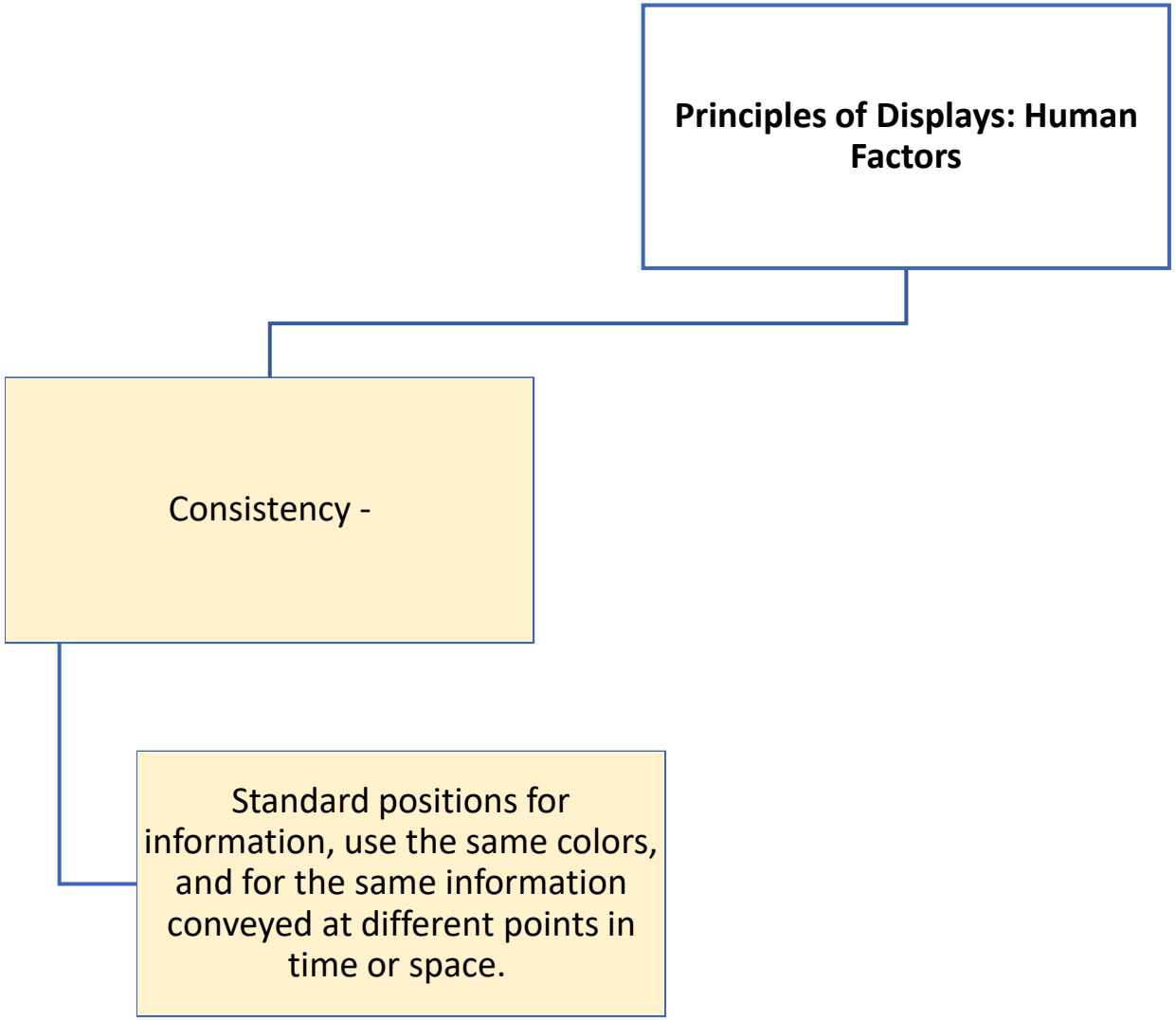
What makes some visualizations better than others?

Principles of Displays: Human Factors

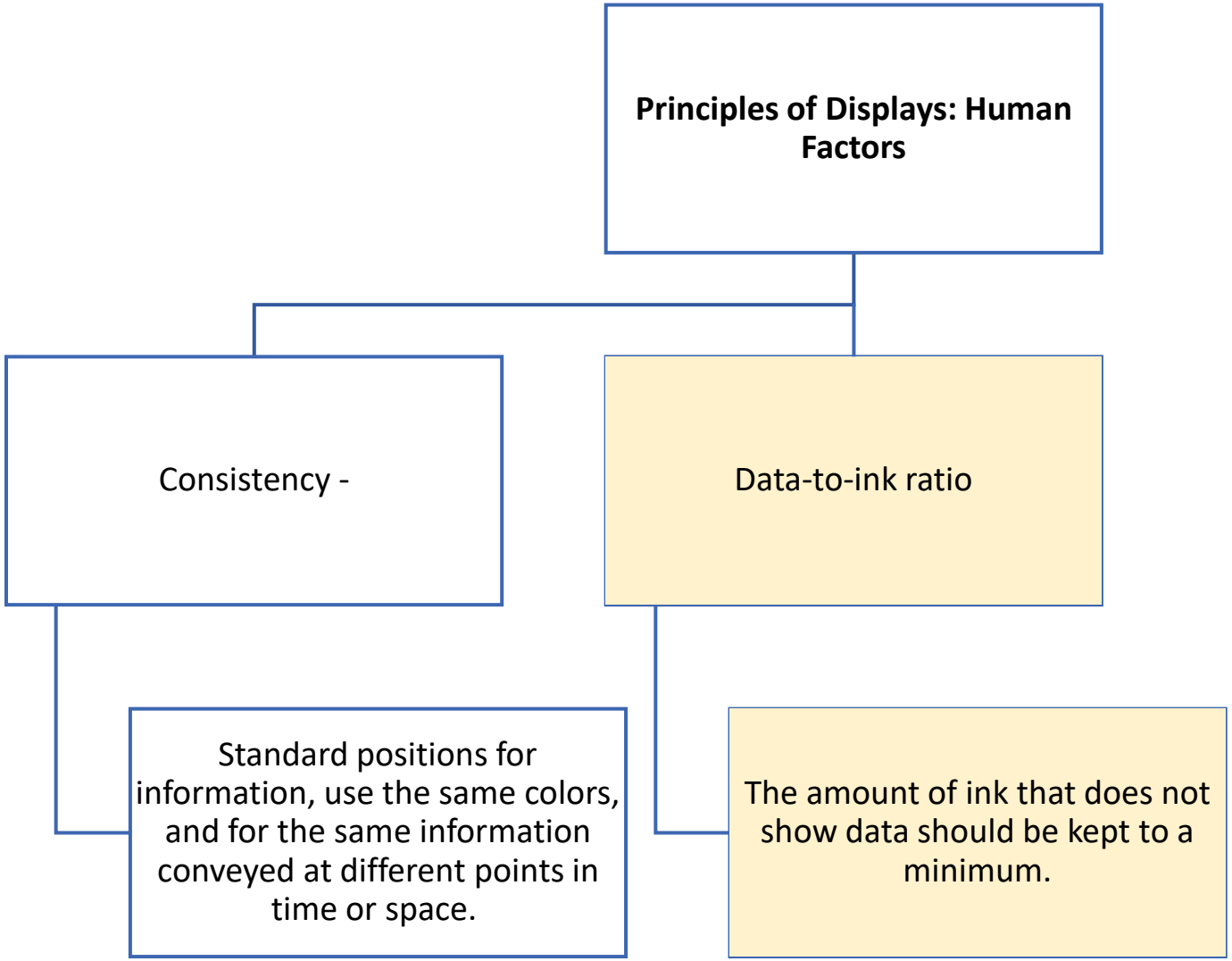
- Consistency - have a standard position for information, use the same colors, and features (line symbols, dashed lines, etc.) for the same information conveyed at different points in time or space.
- Data-to-ink ratio - The amount of ink that does not show data should be kept to a minimum.
- Leverage cultural short-cuts – minimize information access cost

Reduce noise in your visualization

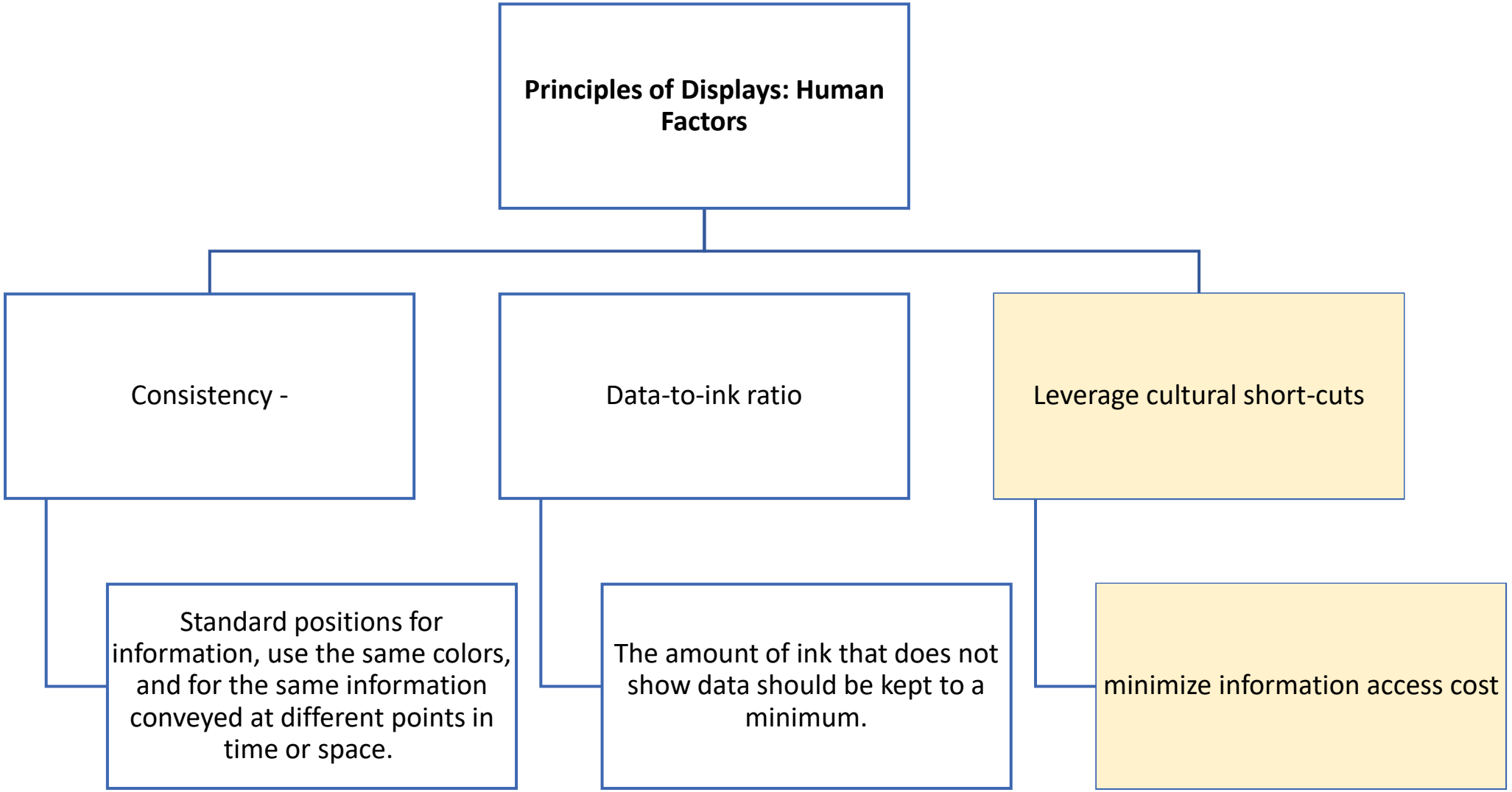
What makes some visualizations better than others?



What makes some visualizations better than others?

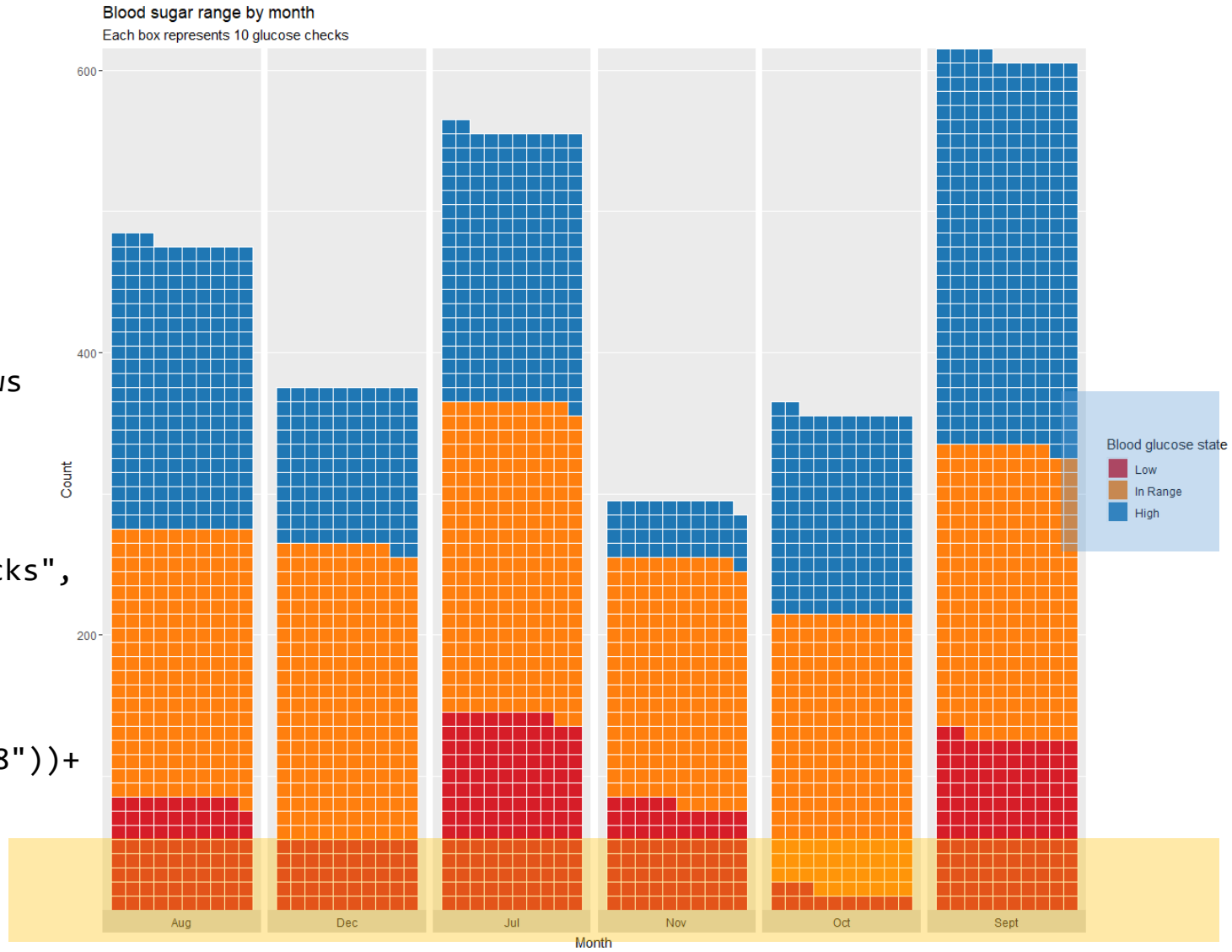


What makes some visualizations better than others?



Leveraging human factors to make a mediocre visualization better

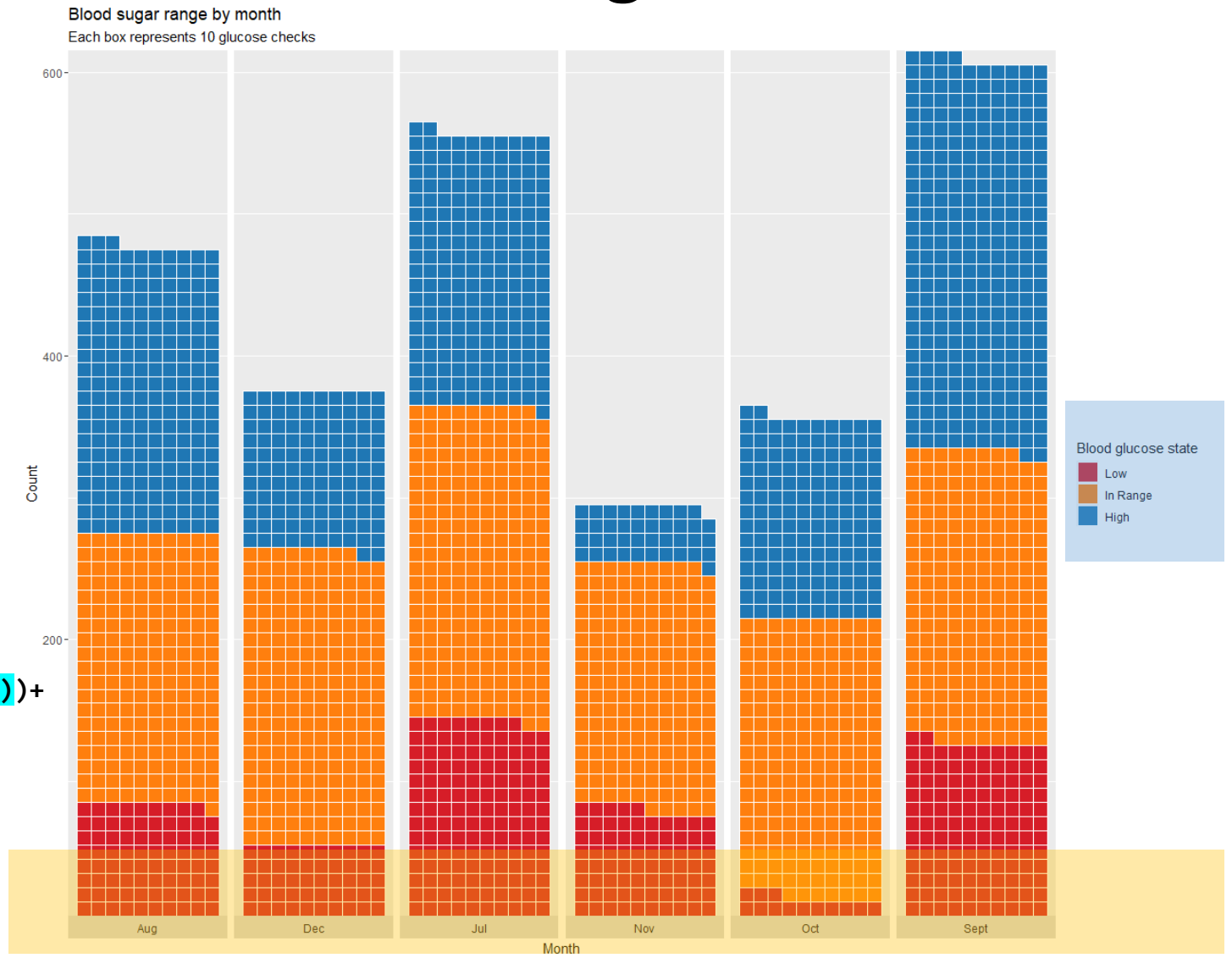
```
Example code made from fake diabetes dataset:
ggplot(diabetes,
  aes(fill = blood_sugar_state, values = n)) +
  geom_waffle(color = "white", size = .25,
              n_rows = 10, flip = TRUE) +
  facet_wrap(~month, nrow = 1,
             strip.position = "bottom") +
  scale_x_discrete() +
  scale_y_continuous(labels = function(x) x * 10,
# make last argument a multiplyer the same as n_rows
                     expand = c(0,0)) +
coord_equal() +
  labs(
    title = "Blood sugar range by month",
    subtitle = "Each box represents 10 glucose checks",
    x = "Month",
    y = "Count",
    fill = "Blood glucose state") +
  scale_fill_manual(values =
                      c("#1f77b4", "#ff7f0e", "#d61d28"))+
 theme(panel.grid = element_line(),
       axis.ticks.y = element_line())+
  guides(fill = guide_legend(reverse = T))
```



Download library for `geom_waffle()` here: https://github.com/hrbrmstr/waffle#waffle

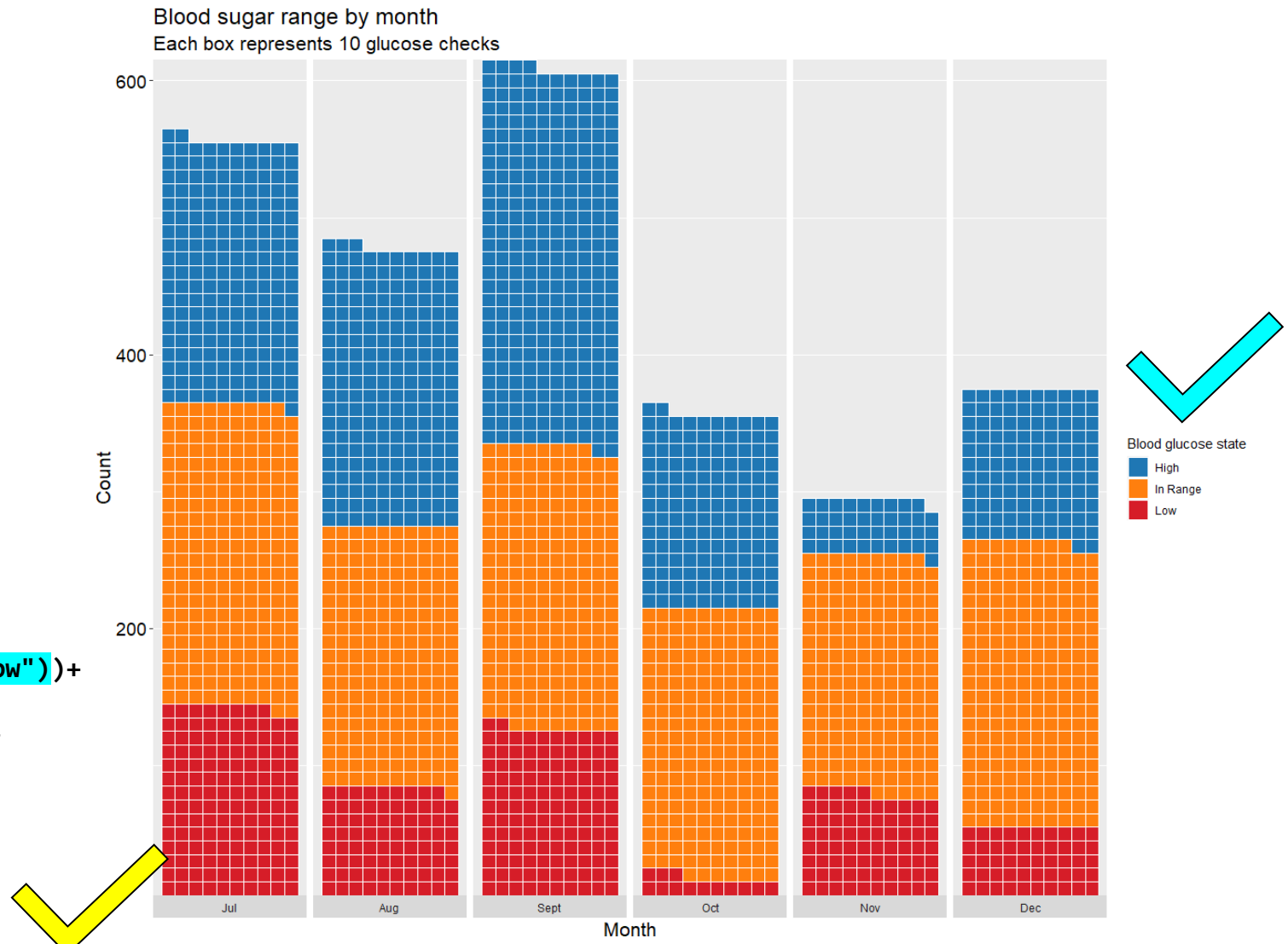
Update variable order to reduce cognitive load

```
ggplot(diabetes,
           aes(fill = blood_sugar_state, values = n)) +
    geom_waffle(color = "white", size = .25,
                n_rows = 10, flip = TRUE) +
    facet_wrap(~fct_reorder(month, year), nrow = 1,
                      strip.position = "bottom") +
    scale_x_discrete() +
    scale_y_continuous(labels = function(x) x * 10,
                       expand = c(0,0)) +
    coord_equal() +
    labs(
      title = "Blood sugar range by month",
      subtitle = "Each box represents 10 glucose checks",
      x = "Month",
      y = "Count",
      fill = "Blood glucose state") +
    scale_fill_discrete(breaks = c("High", "In Range", "Low"))+
    scale_fill_manual(values =
                      c("#1f77b4", "#ff7f0e", "#d61d28"))+
theme(... +
    guides(fill = guide_legend(reverse = F))
```

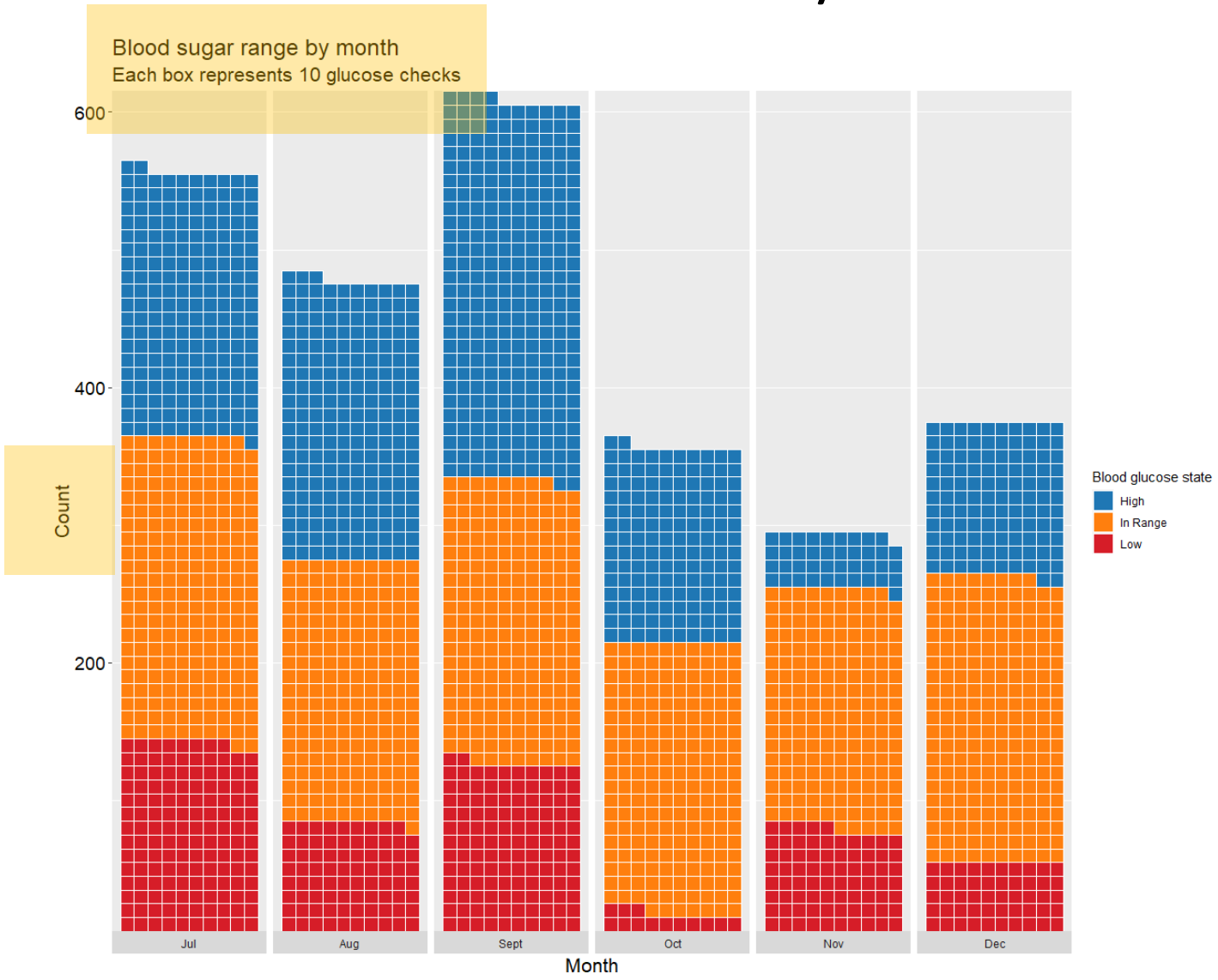


Update variable order to reduce cognitive load

```
ggplot(diabetes,
           aes(fill = blood_sugar_state, values = n)) +
    geom_waffle(color = "white", size = .25,
                n_rows = 10, flip = TRUE) +
    facet_wrap(~fct_reorder(month, year), nrow = 1,
                      strip.position = "bottom") +
    scale_x_discrete() +
    scale_y_continuous(labels = function(x) x * 10,
                       expand = c(0,0)) +
    coord_equal() +
    labs(
      title = "Blood sugar range by month",
      subtitle = "Each box represents 10 glucose checks",
      x = "Month",
      y = "Count",
      fill = "Blood glucose state") +
    scale_fill_discrete(breaks = c("High", "In Range", "Low"))+
    scale_fill_manual(values =
                      c("#1f77b4", "#ff7f0e", "#d61d28"))+
theme(... +
    guides(fill = guide_legend(reverse = F))
```

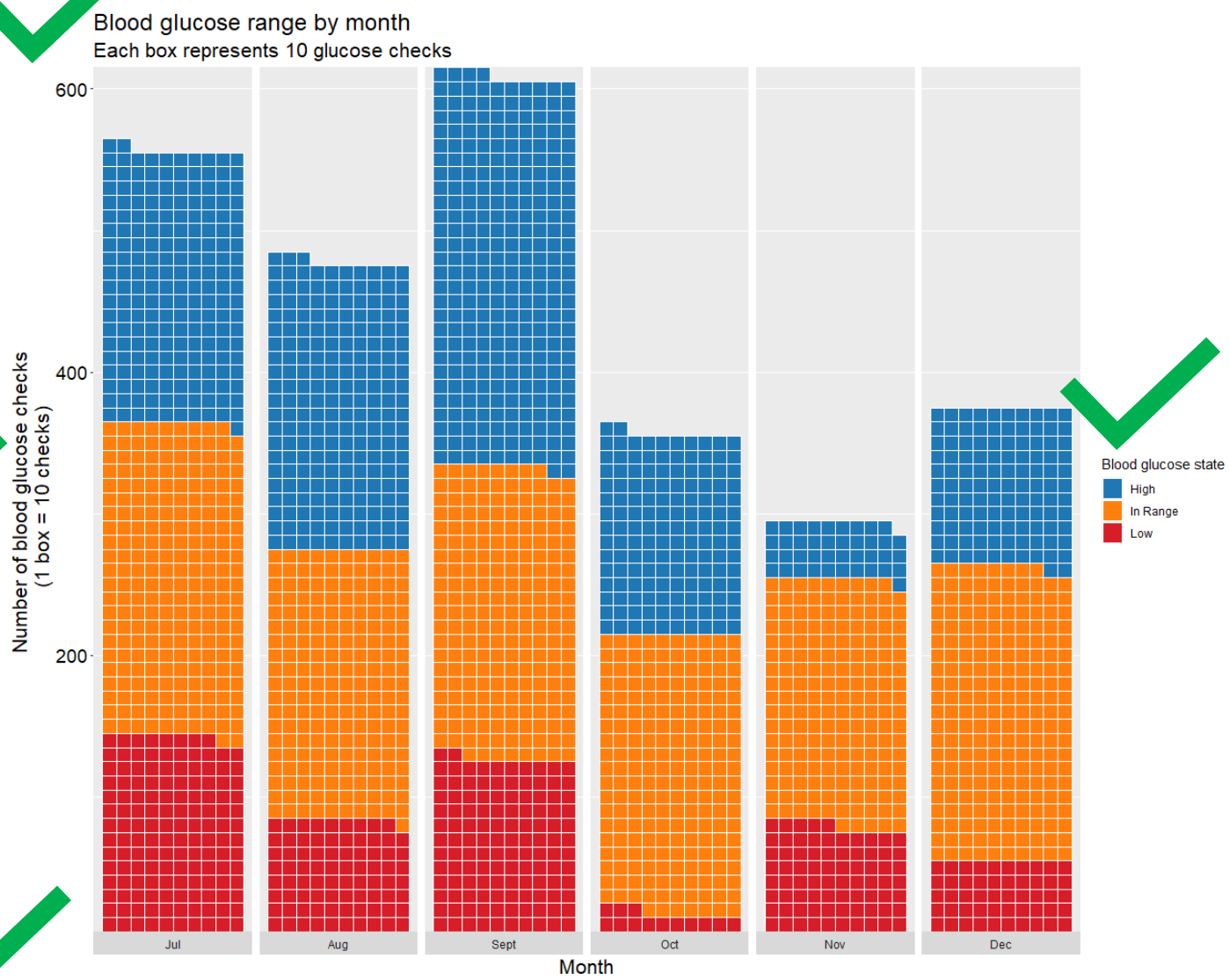


Consistency reduces distractions



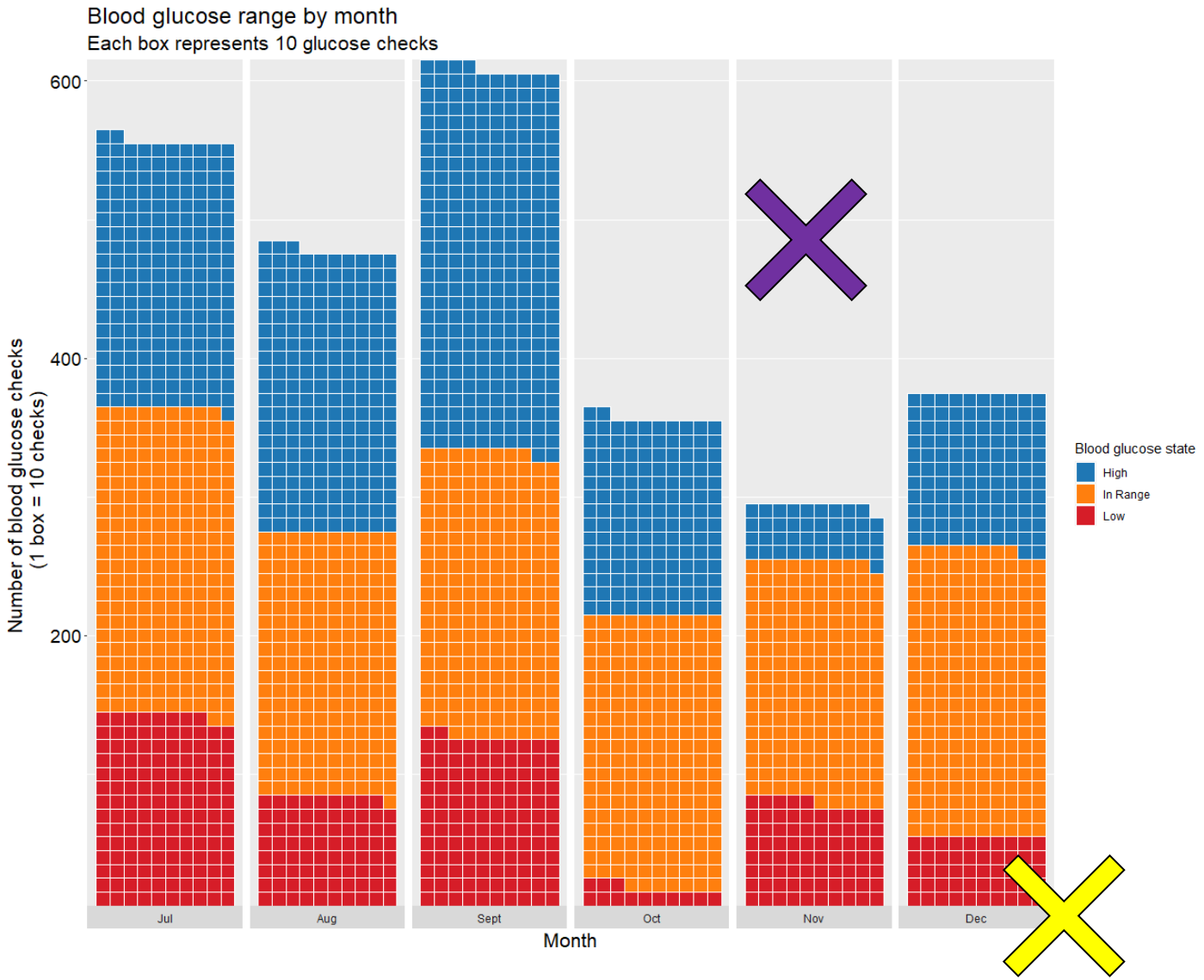

```
labs(
    title = "Blood glucose range by month",
    subtitle = "Each box represents 10 glucose checks",
    x = "Month",
    y = "Number of blood glucose checks \n (1 box = 10
checks)",
    fill = "Blood glucose state") +
```

Consistency achieved!



Have a standard position for information; use the same colors and features (line symbols, dashed lines, etc.) for the same information conveyed at different points in time or space. If there are differences in the types of information conveyed in the symbol, highlight the differences by using different representations.

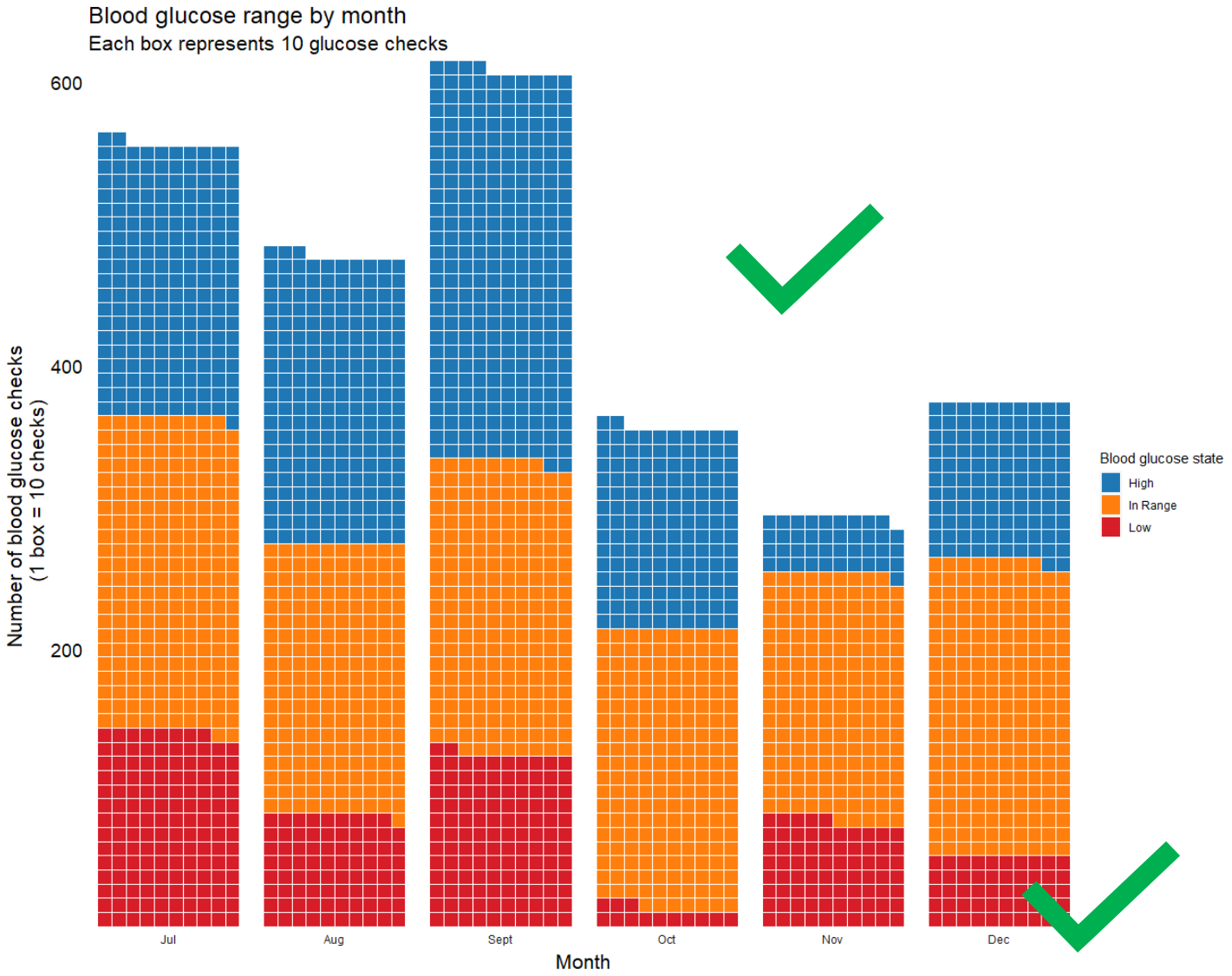
Data-to-ink ratio: remove useless gray area



The amount of ink that does not show data should be kept to a minimum.

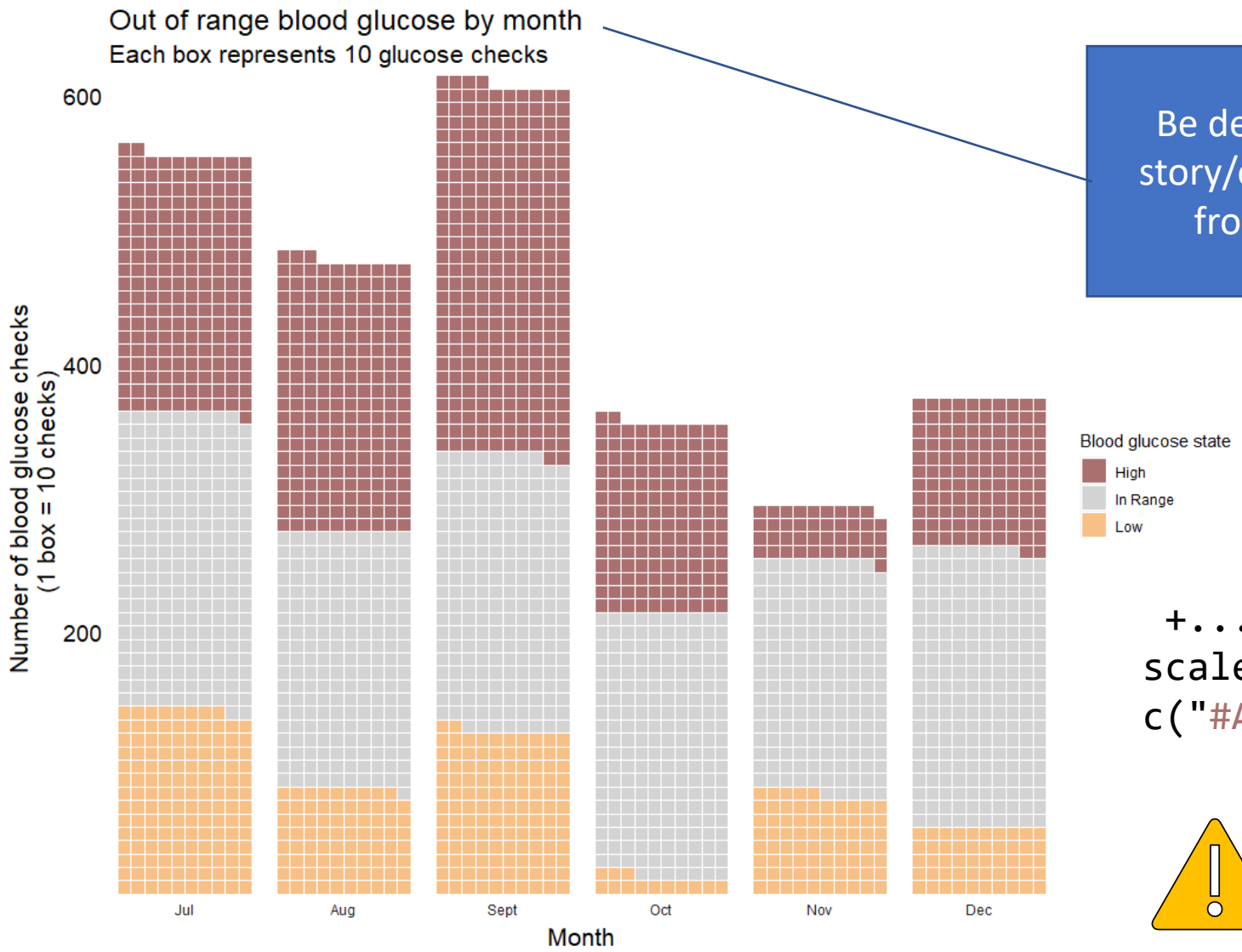
Use `element_blank()` in `theme` for any places you want to remove black lines or backgrounds. Specifically, `panel.background` to remove the plot background and `strip.background` to remove the background in the `facet_wrap` labels

Data-to-ink ratio achieved – resulting in a cleaner look




Level-up by leveraging
cultural short-cuts

Leverage cultural shortcuts – use of red



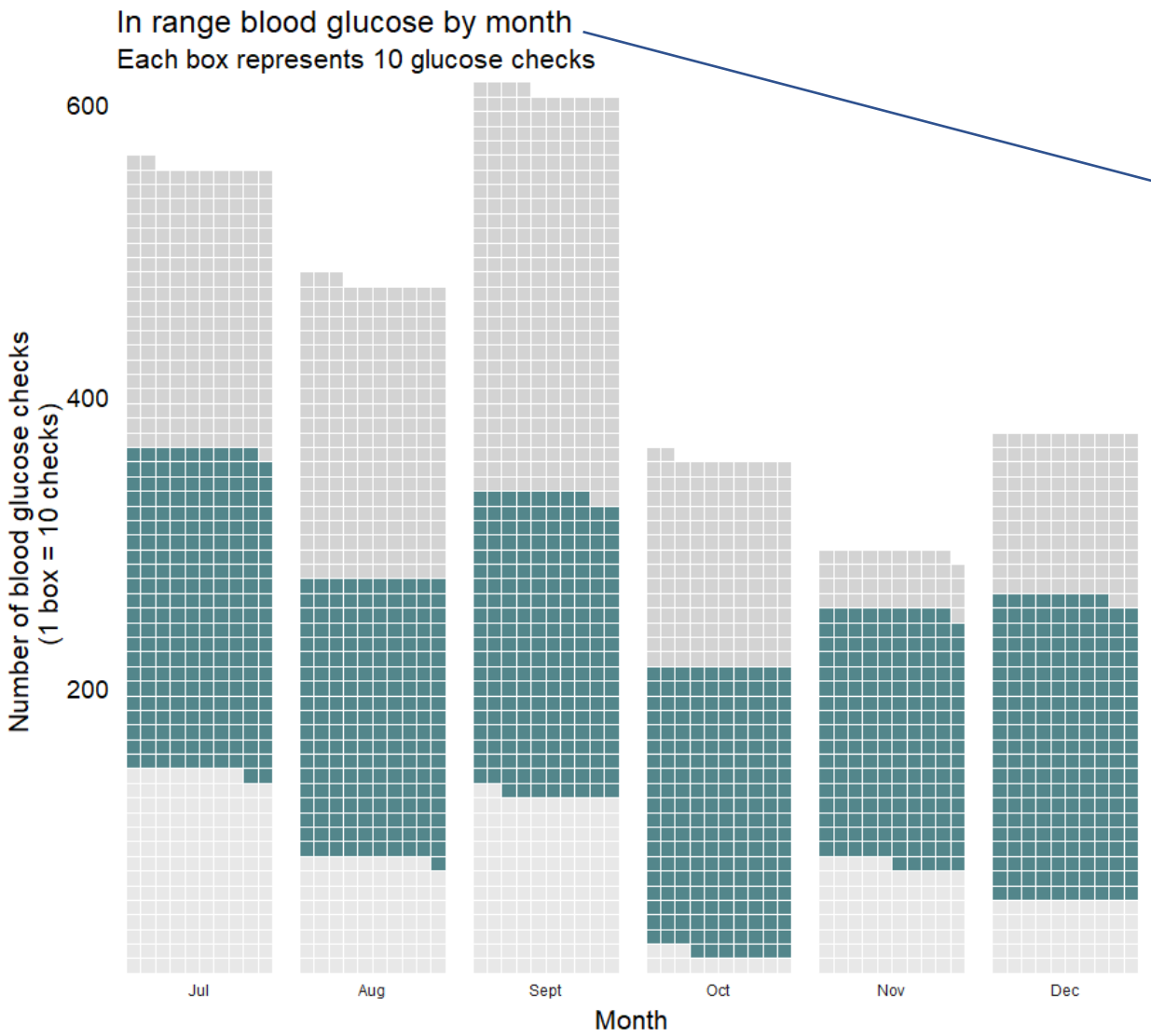
Be deliberate and put your findings/the story/or what you want the reader to pull from your visualization in your title

```
 +...
scale_fill_manual(values =
c("#AA7070","lightgray", "#F7C086"))+
```

Consider that typically red is not accessible.

I've chosen a specific red that I know works well with these colors. Check out https://coolors.co to check your own colors!

Leverage cultural shortcuts – use of neutral tones



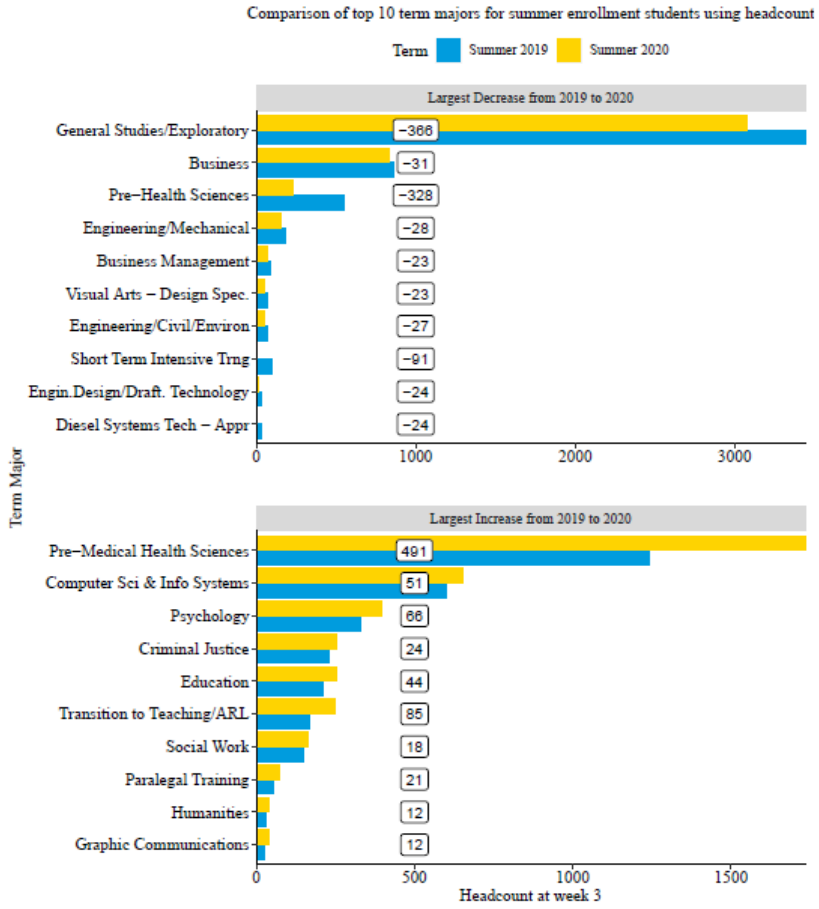
Be deliberate and put your findings/the story/or what you want the reader to pull from your visualization in your title

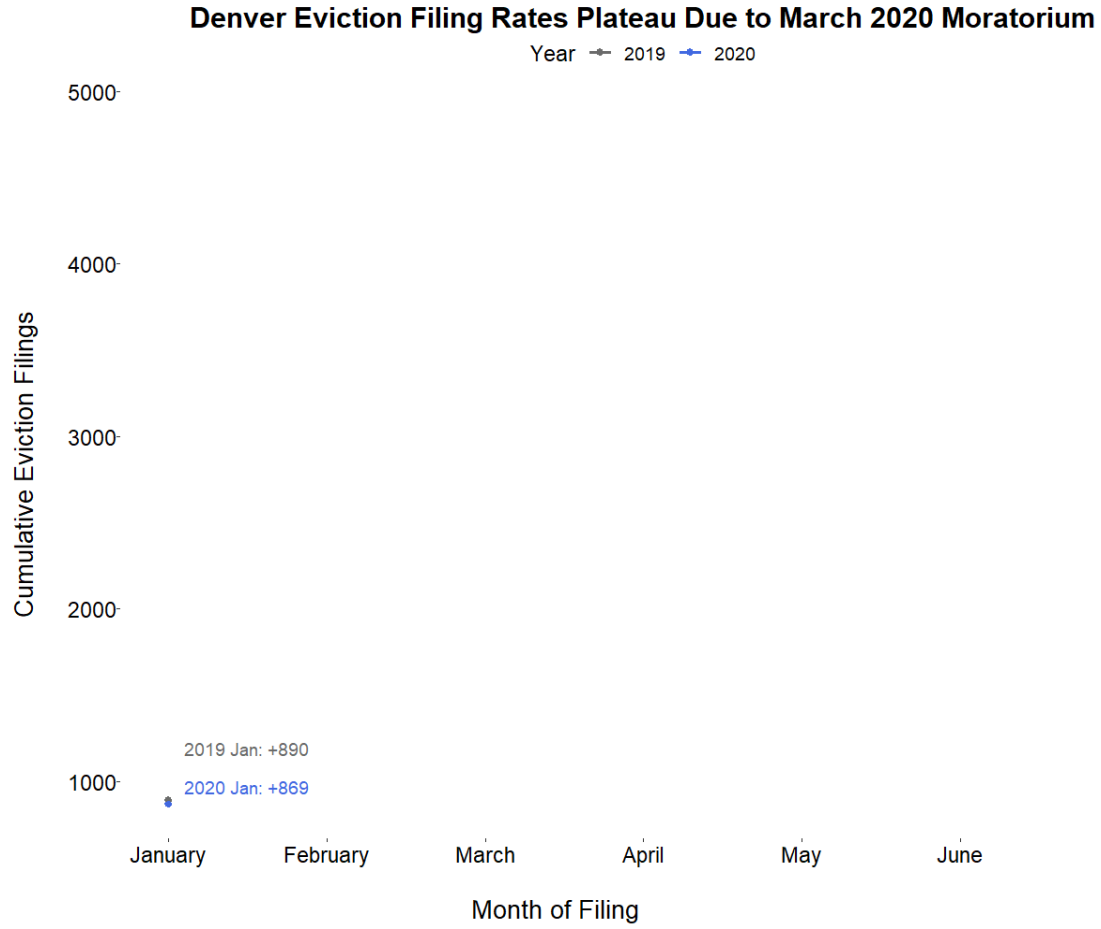
```
 +...
scale_fill_manual(values =
c("#D3D3D3","cadetblue4", "#E8E8E8"))+
```

Consider that the reader will not be able to draw conclusions about the gray areas.

This should be used to drive home a point where the other 2 groups are occurring more/less often than the group in focus.

Reducing noise allows you to present **more** information





Final Tips

- Imagine your visualization will be lifted with no context
- Reserve captions for data nuances
- Leverage associations

Reproducible Code & Visualization Best Practices

Lara N. Southard, PhD

Senior Research Scientist at Pearson

lara-southard.medium/com

lsouthard

@larasouthard

@lsouth14