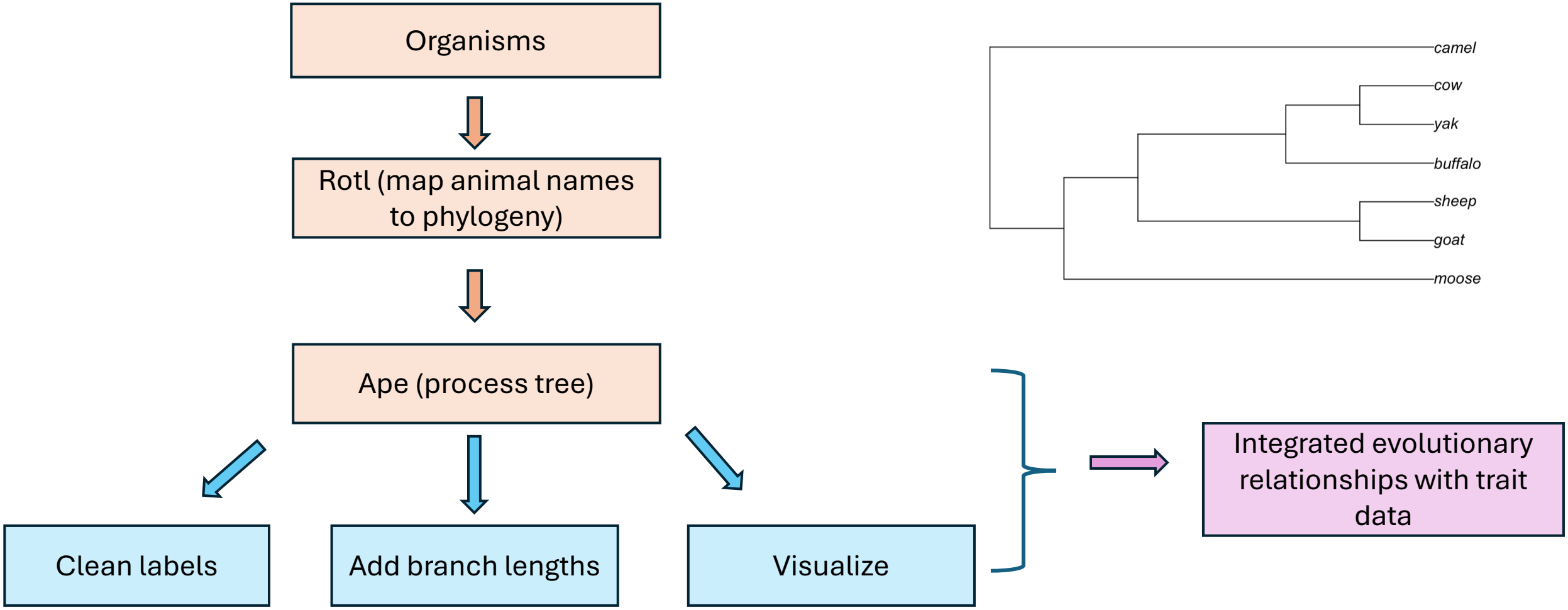

Exploring cheeses around the world!

Check Out Our Dashboard!

- https://github.com/alinagalyon/Cheese-database/tree/main

Ape (Analyses of Phylogenetics and Evolution)& ROTL- Interface to the 'Open Tree of Life'

Phylogenetic relationships among milk-producing animals were retrieved using Open Tree of Life (`rotl`) and processed with ape to align flavor and nutritional traits in a heatmap.



We used the **ggimage** package to display animal icons within the graph

- CRAN package, depends on ggplot2
- Need folder of images as .png files
  - Joined with cheese dataset

```
geom_image(
aes(image = image, y= -0.4),
size = 0.09, by = "width", asp = 1.5,
nudge_y = 0) +
```

```
#Create animal image data frame from BioRender (saved png files in Animalpngs folder)
Animal_Images <- data.frame(
  animal = c("yak", "water buffalo", "sheep", "plant-based", "moose", "goat", "donkey", "cow",
"camel", "buffalo"),
  value = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
  image = c("Animalpngs/Yak.png", "Animalpngs/Waterbuffalo.png",
"Animalpngs/Sheep.png","Animalpngs/Plant.png", "Animalpngs/Moose.png", "Animalpngs/Goat.png",
"Animalpngs/Donkey.png", "Animalpngs/Cow.png","Animalpngs/Camel.png", "Animalpngs/Buffalo.png"))

Animal_Cheese <- full_join(cheeses_indv, Animal_Images, by = "animal", keep = FALSE) %>%
  arrange(desc(cheese))
```

```
Animal_Cheese %>%
  ggplot(aes(x = reorder(animal, cheese), y = log10(cheese))) +
  geom_col(color = "black", fill = "white") +
  geom_text(aes(label = cheese), vjust = 0, nudge_y = 0.11) +
  coord_flip() +
  scale_y_continuous(limits = c(-0.5, 3.5)) +
 geom_image(aes(image = image, y= -0.4), size = 0.09, by = "width", asp = 1.5, nudge_y = 0) +
  labs(x = "Animal Milk", y = "Number of Cheese Types (log10(cheese counts))", title = "Amount of
cheese made from different animals' milk") +
  theme_classic() +
    theme(axis.text.x = element_text(size = 10),
        axis.text.y = element_text(size = 12),
        axis.text.x.top = element_text(size = 15))
```

Leaflet for Interactive Mapping

Cleaning Borders

```
#load the country borders
borders <- ne_countries(scale = "small",
                        returnclass = "sf")
#clean country borders
sf_use_s2(FALSE)  #turn off spherical lat/long- use Cartesian

## Spherical geometry (s2) switched off

borders_clean <- borders %>%  #make valid borders
  st_make_valid() %>%
  st_buffer(dist = 0)

## dist is assumed to be in decimal degrees (arc_degrees).

cat("Cleaned borders valid:", all(st_is_valid(borders_clean)), "\n") #check
to see worked, TRUE

## Cleaned borders valid: TRUE

sf_use_s2(TRUE) #back to default setting for coordinates

## Spherical geometry (s2) switched on
```

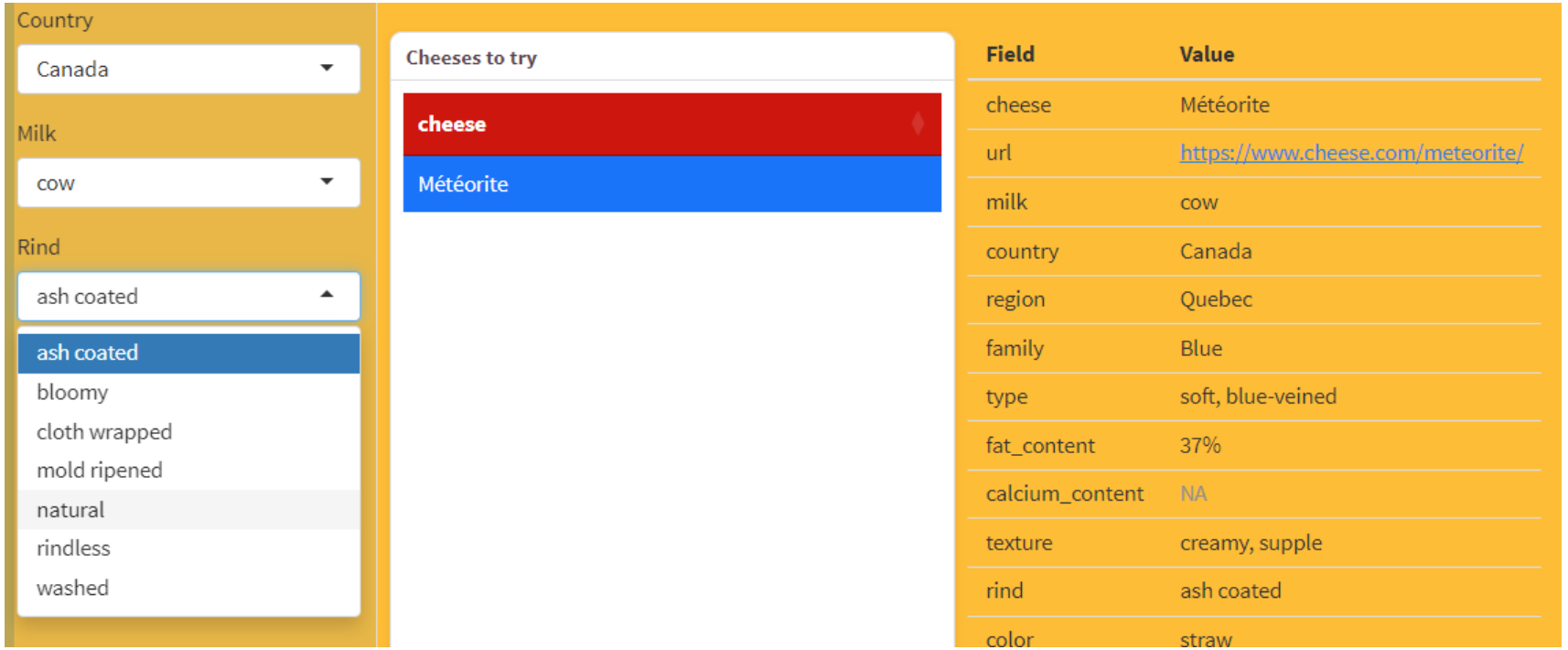
- The sf package defaults to spherical coordinates
- Turn this off to fix overlapping/misaligned borders

HTMLTools

```
#print the plot
cheese_map <- leaflet(data = cheese_borders,
                      options = leafletOptions(minZoom = 1.45)) %>%
  addPolygons(
    fillColor = ~pal(cheese_types),
    fillOpacity = 0.7,
    color = "black",
    weight = 0.5,
    label = ~lapply(
      paste0(
        sovereignt, "<br/>Cheese Types Reported: ", cheese_types,
"<br/>Favored Milk Source: ", common_animal$milk, "<br/>Largest Producer: ",
top_company$producers), HTML)) %>%
  addLegend(
    colors = c("white", cheese_pal),
    position = "bottomright",
    values = cheese_borders$cheese_types,
    title = "World Cheese Production<br/> # Varieties",
    labels = c(
      "No Data Collected", "1-3", "4-6", "7-10", "11-15", "16-25", "26-40",
"41-60", "61-90", "91-150",
      "151-200", "201-250", "251-300", "<301"
    ),
    opacity = 0.7
  ) %>%
  print()
```

- Allows you to work with HTML content inside R

Cheese Finder

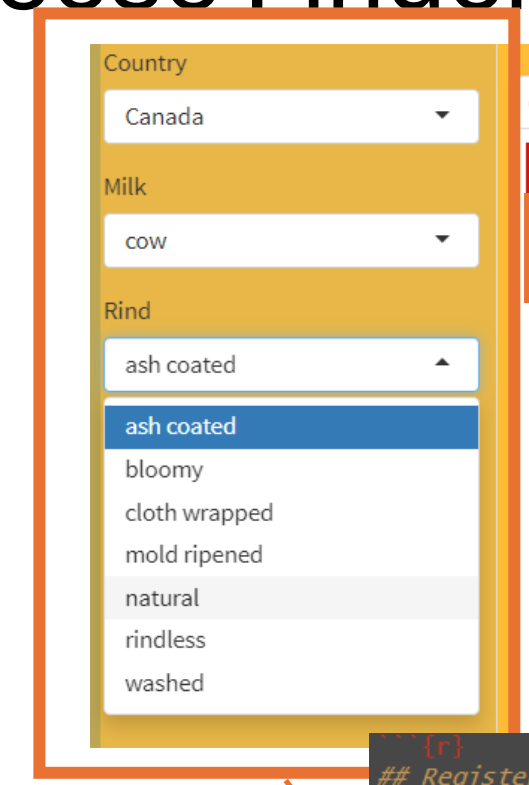


The cheese finder is a way for users to discover new cheeses or find out more on cheese they know.

The finder is interactable and adapt to a user's inputs.

Cheese Finder

Cheeses are selectable to show more information about them



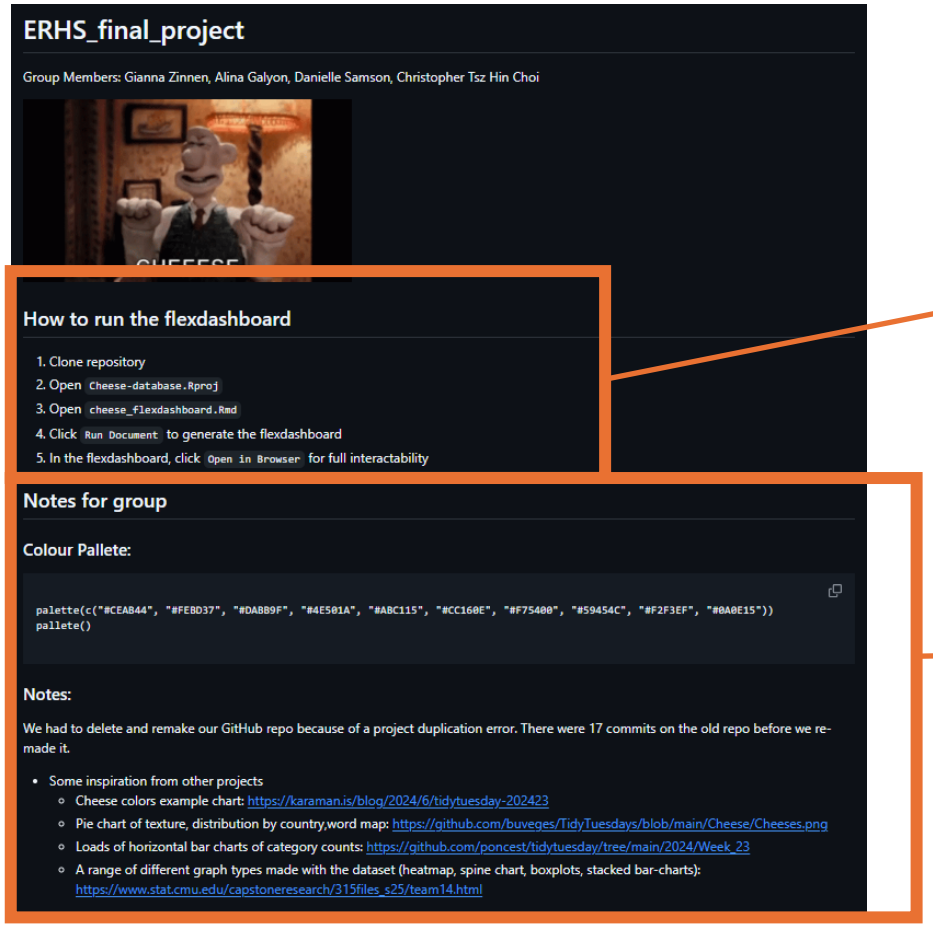
Field	Value
cheese	Météorite
url	https://www.cheese.com/meteorite/
milk	cow
country	Canada
region	Quebec
family	Blue
type	soft, blue-veined
fat_content	37%
calcium_content	NA
texture	creamy, supple
rind	ash coated
color	straw

```r
## Registers what the columns record
filtered_cheeses <- reactive({
  req(
    input$selected_country,
    input$selected_milk,
    input$selected_rind
  )
  cheeses_finder |>
    dplyr::filter(
      country == input$selected_country,
      milk == input$selected_milk,
      rind == input$selected_rind
      )
})
```

Based on selections, we can filter the data table to show cheeses that fit the criteria

Each drop down also filters to available options based on what is selected in the other drop downs

Flexdashboard Overview and github Process



ERHS_final_project

Group Members: Gianna Zinnen, Alina Galyon, Danielle Samson, Christopher Tsz Hin Choi

How to run the flexdashboard

1. Clone repository
2. Open `Cheese-database.Rproj`
3. Open `cheese_flexdashboard.Rmd`
4. Click `Run Document` to generate the flexdashboard
5. In the flexdashboard, click `Open in Browser` for full interactability

Notes for group

Colour Pallete:

```
palette(c("#CEAB44", "#FEBD37", "#DABB9F", "#4E501A", "#ABC115", "#CC160E", "#F75400", "#59454C", "#F2F3EF", "#0A0E15"))
pallete()
```

Notes:

We had to delete and remake our GitHub repo because of a project duplication error. There were 17 commits on the old repo before we re-made it.

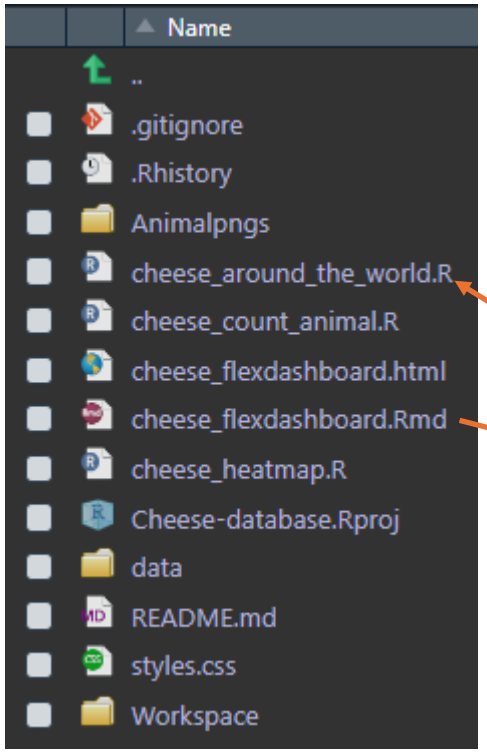
- Some inspiration from other projects
  - Cheese colors example chart: https://karaman.is/blog/2024/6/tidytuesday-202423
  - Pie chart of texture, distribution by country,word map: https://github.com/buveges/TidyTuesdays/blob/main/Cheese/Cheeses.png
  - Loads of horizontal bar charts of category counts: https://github.com/poncest/tidytuesday/tree/main/2024/Week_23
  - A range of different graph types made with the dataset (heatmap, spine chart, boxplots, stacked bar-charts): https://www.stat.cmu.edu/capstoneresearch/315files_s25/team14.html

Using the README for coordination

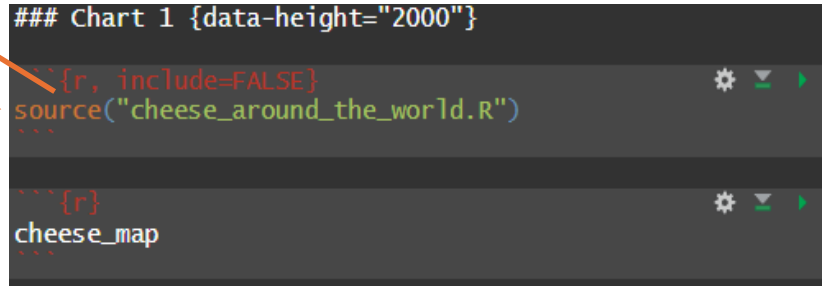
Instructions on how to run flexdashboard

The README was also used for communicating with the group on shared resources

Flexdashboard Overview and github Process

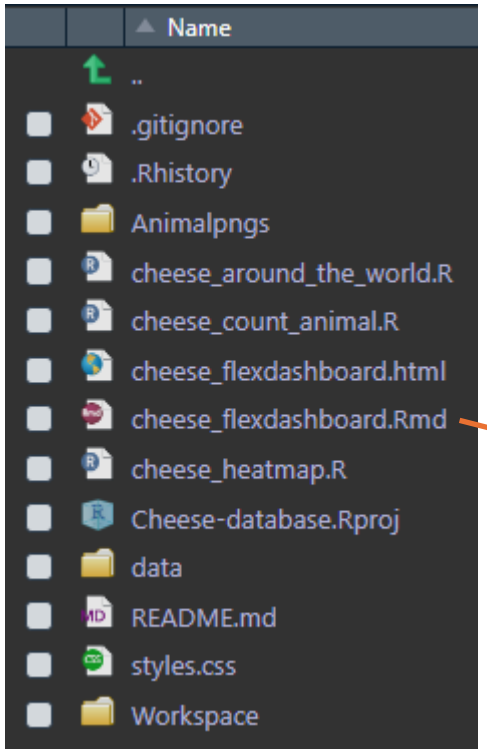


To keep our flexdashboard minimal, we used source() functions to run other scripts. This way, the flexdashboard primarily comprises of code for formatting and allows for easier legibility and editability.



This also means that from a user only needs to open and run a single script making operation simple.

Flexdashboard Overview and github Process



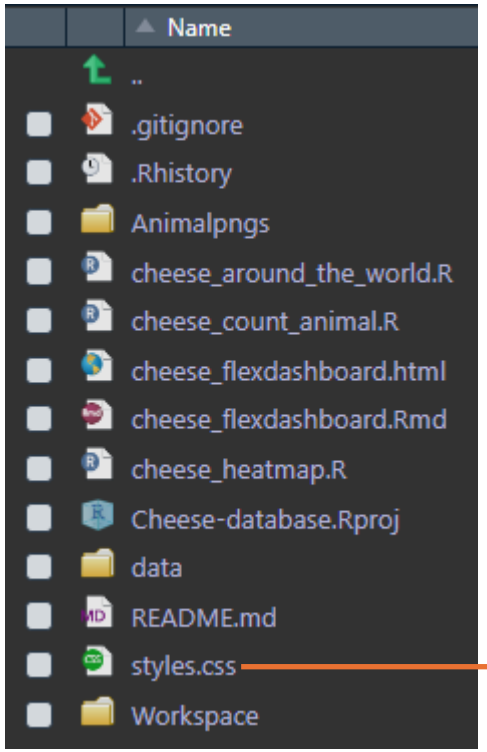

```r
## Install/load packages required
load_or_install <- function(pkg) {
  if (!requireNamespace(pkg, quietly = TRUE)) {
    install.packages(pkg, dependencies = TRUE, repos = "https://cloud.r-project.org")
  }
  library(pkg, character.only = TRUE)
}

pkgs <- c("dplyr",
          "tidyr",
          "DT",
          "shiny",
          "flexdashboard",
          "tidyverse",
          "stringr",
          "tidyr",
          "ape",
          "rotl",
          "ComplexHeatmap",
          "circlize",
          "tibble",
          "tidyverse",
          "readr",
          "leaflet",
          "leaflet.extras",
          "sf",
          "knitr",
          "rnaturalearth",
          "tigris",
          "ggthemes",
          "cartogram",
          "scales",
          "viridis",
          "htmltools",
          "ggplot2",
          "ggimage",
          "plotly",
          "scales",
          "grid",
          "tidyverse",
          "dplyr")

invisible(lapply(pkgs, load_or_install))
```

Packages used in the project are checked on a user's PC, installed if it not available, and loaded in.

Flexdashboard Overview and github Process




```css
/* ============================
   Global background
   ============================ */

body {
  background-color: #CEAB44;   /* dark cheddar */
}


/* ============================
   Section (tab) background
   ============================ */

.section.level1 {
  background-color: #FEBD37;  /* young cheddar */
  padding-top: 20px;
}


/* ============================
   Navbar styling
   ============================ */

.navbar {
  background-color: #CEAB44;   /* dark cheddar */
  border-color: #2c7a7b;
}

/* Navbar title + links */
.navbar a,
.navbar-brand {
  color: #ffffff !important;
  font-weight: 600;
}

/* Hover state */
.navbar a:hover {
  color: #d1fae5 !important;
```

To have better control over the styling of the flexdashboard, we used a CSS

Lessons Learned…

- Positives:
  - Flexdashboards are quite powerful – it was a great way to deliver information and adaptable to the complexity required
    - Allows for flexible ways to integrate markdown, R, html, and css together

- Errors:
  - Having 2 Rproj files can lead to big problems as they can share dependencies (we had to purge our original repo and create a new one)
    - This could have been avoided early on with more push/pulls so that we could have caught and resolved it early on


Image generated with Flux 1.1 Pro