Homework 4

2025-04-01

Homework 4: STRUCTURED DATA + MODELS

```
library(tidyverse)
```

```
## Warning: package 'purrr' was built under R version 4.3.3
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.5.1     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.1
## ✔ purrr     1.0.4
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
to become errors
```

```
library(dplyr)
library(lubridate)
# library(ggHoriPlot)
# library(ggthemes)
library(ggraph)
library(tidygraph)
```

```
##
## Attaching package: 'tidygraph'
##
## The following object is masked from 'package:stats':
##
##     filter
```

```
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 4.3.3
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(tsibble)
```

```
## Warning: package 'tsibble' was built under R version 4.3.3
```

```
## Registered S3 method overwritten by 'tsibble':
##   method               from
##   as_tibble.grouped_df dplyr
##
## Attaching package: 'tsibble'
##
## The following object is masked from 'package:lubridate':
##
##     interval
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, union
```

```
library(feasts)
```

```
## Warning: package 'feasts' was built under R version 4.3.3
```

```
## Loading required package: fabletools
```

```
## Warning: package 'fabletools' was built under R version 4.3.3
```

```
library(sf)
```

```
## Warning: package 'sf' was built under R version 4.3.3
```

```
## Linking to GEOS 3.13.0, GDAL 3.8.5, PROJ 9.5.1; sf_use_s2() is TRUE
```

```
library(terra)
```

```
## Warning: package 'terra' was built under R version 4.3.3
```

```
## terra 1.8.21
##
## Attaching package: 'terra'
##
## The following object is masked from 'package:fabletools':
##
##     interpolate
##
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
library(tidyr)
library(patchwork)
```

```
## Warning: package 'patchwork' was built under R version 4.3.3
```

```
##
## Attaching package: 'patchwork'
##
## The following object is masked from 'package:terra':
##
##     area
```

```
library(ggmap)
```

```
## ℹ Google's Terms of Service: <https://mapsplatform.google.com>
##   Stadia Maps' Terms of Service: <https://stadiamaps.com/terms-of-service/>
##   OpenStreetMap's Tile Usage Policy: <https://operations.osmfoundation.org/policies/t
iles/>
## ℹ Please cite ggmap if you use it! Use `citation("ggmap")` for details.
##
## Attaching package: 'ggmap'
##
##
## The following object is masked from 'package:terra':
##
##     inset
```

```
library(ggplot2)
library(ggspatial)
library(tmap)
```

```
## Warning: package 'tmap' was built under R version 4.3.3
```

```
library(tidyverse)
library(tidymodels)
```

```
## Warning: package 'tidymodels' was built under R version 4.3.3
```

```
## ── Attaching packages ─────────────────────────────────────── tidymodels 1.3.0 ──
## ✔ broom        1.0.8     ✔ rsample      1.3.0
## ✔ dials        1.4.0     ✔ tune         1.3.0
## ✔ infer        1.0.7     ✔ workflows    1.2.0
## ✔ modeldata    1.4.0     ✔ workflowsets 1.1.0
## ✔ parsnip      1.3.1     ✔ yardstick    1.3.2
## ✔ recipes      1.2.1
```

```
## Warning: package 'broom' was built under R version 4.3.3
```

```
## Warning: package 'dials' was built under R version 4.3.3
```

```
## Warning: package 'scales' was built under R version 4.3.3
```

```
## Warning: package 'modeldata' was built under R version 4.3.3
```

```
## Warning: package 'parsnip' was built under R version 4.3.3
```

```
## Warning: package 'recipes' was built under R version 4.3.3
```

```
## Warning: package 'rsample' was built under R version 4.3.3
```

```
## Warning: package 'tune' was built under R version 4.3.3
```

```
## Warning: package 'workflows' was built under R version 4.3.3
```

```
## Warning: package 'yardstick' was built under R version 4.3.3
```

```
## ── Conflicts ──────────────────────────────────────── tidymodels_conflicts() ──
## ✖ yardstick::accuracy() masks fabletools::accuracy()
## ✖ dials::buffer()       masks terra::buffer()
## ✖ gridExtra::combine()  masks dplyr::combine()
## ✖ scales::discard()     masks purrr::discard()
## ✖ terra::extract()      masks tidyr::extract()
## ✖ tidygraph::filter()   masks dplyr::filter(), stats::filter()
## ✖ recipes::fixed()      masks stringr::fixed()
## ✖ infer::generate()     masks fabletools::generate()
## ✖ infer::hypothesize()  masks fabletools::hypothesize()
## ✖ dplyr::lag()          masks stats::lag()
## ✖ parsnip::null_model() masks fabletools::null_model()
## ✖ yardstick::spec()     masks readr::spec()
## ✖ recipes::step()       masks stats::step()
## ✖ recipes::update()     masks terra::update(), stats::update()
```

```
library(recipes)
library(tidygraph)
library(ggraph)
library(dplyr)
library(readr)
library(countrycode)
```

```
## Warning: package 'countrycode' was built under R version 4.3.3
```

```
library(tidyverse)
library(cluster)
library(dendextend)
```

```
## Warning: package 'dendextend' was built under R version 4.3.3
```

```
##
## ----------------------
## Welcome to dendextend version 1.19.0
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendext
end/issues
## You may ask questions at stackoverflow, use the r and dendextend tags:
##   https://stackoverflow.com/questions/tagged/dendextend
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ----------------------
##
##
## Attaching package: 'dendextend'
##
## The following object is masked from 'package:dials':
##
##     prune
##
## The following object is masked from 'package:terra':
##
##     rotate
##
## The following object is masked from 'package:stats':
##
##     cutree
```

```
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WB
a
```

```
library(patchwork)
```

[Coding]

[Geospatial Datasets]

- NYC Building Footprints

Format: Vector

Geometry Type: Polygon

Explanation: The NYC Building Footprints dataset represents the outlines of buildings throughout New York City. Since it captures the shape and size of each building using defined boundaries, the data is stored as polygons, which are a type of vector geometry. Vector formats are appropriate here because they can accurately represent the spatial extent and complex shapes of individual buildings.

- Africa Population 2020

Format: Raster

Explanation: This dataset represents population density across the African continent. The values vary continuously over space and are visualized as a grid of varying brightness, with darker regions indicating higher population concentrations. This is a classic example of raster data, which is best suited for modeling continuous variables such as population, elevation, or temperature.

- Himalayan Glacial Lakes

Format: Vector

Geometry Type: Polygon

Explanation: The Himalayan Glacial Lakes dataset represents the physical outlines of glacial lakes in the Himalayan region. Each lake is mapped with its geographical boundaries, making polygons the appropriate geometry type. Because the dataset records the shape and spatial extent of each lake, it is stored as a vector dataset. Polygons are ideal for modeling areas with defined edges, such as lakes, making them suitable for analysis like measuring surface area or studying changes in lake size over time.

- Wisconsin EV Charging

Format: Vector

Geometry Type: Point

Explanation: This dataset stores the locations of EV charging stations in Wisconsin, with each entry represented as a point geometry in GeoJSON format. Vector data is appropriate here because it captures discrete features, and point geometry is ideal for representing specific locations like charging stations.

- Zion Elevation

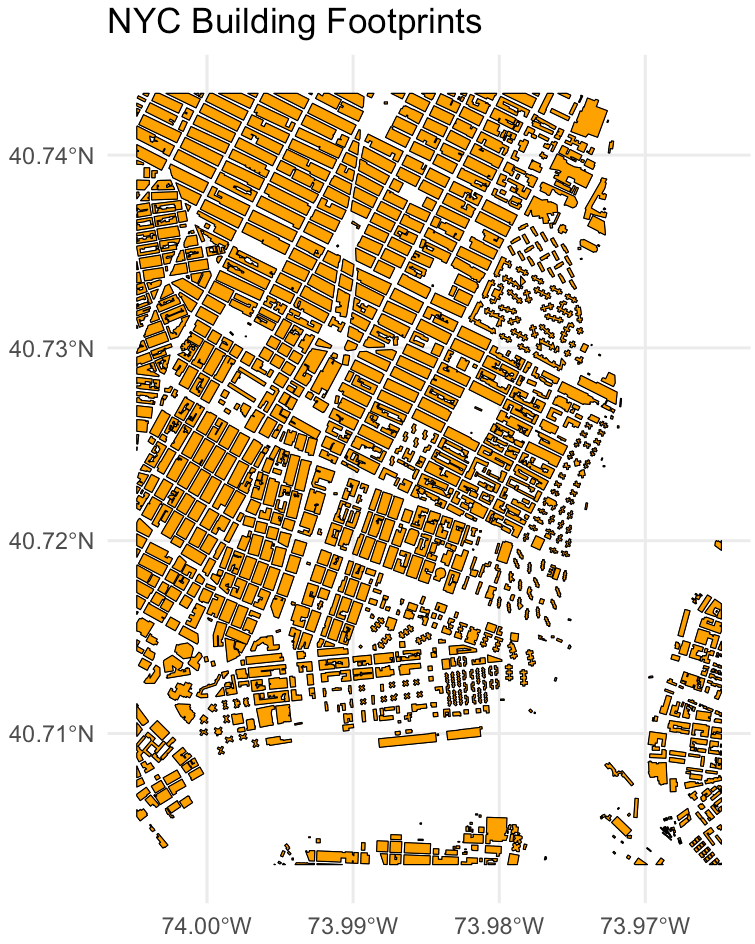
Format: Raster

Explanation: This dataset represents elevation in Zion National Park using a continuous raster grid. Each cell holds an elevation value, and variations in shading reflect changes in terrain height. Raster format is ideal for representing continuous spatial phenomena such as elevation.

```
nyc <- st_read("https://uwmadison.box.com/shared/static/qfmrp9srsoq0a7oj0e7xmgu5spojr33
e.geojson")
```

```
## Reading layer `nyc-buildings' from data source
##   `https://uwmadison.box.com/shared/static/qfmrp9srsoq0a7oj0e7xmgu5spojr33e.geojson'
##   using driver `GeoJSON'
## Simple feature collection with 2726 features and 0 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -74.0048 ymin: 40.7032 xmax: -73.9648 ymax: 40.7432
## Geodetic CRS:  WGS 84
```

```
#f. Visualize one of these datasets

ggplot(nyc) +
  geom_sf(fill = "#FFA500", color = "black", size = 0.2) +
  labs(title = "NYC Building Footprints") +
  theme_minimal()
```



[CalFresh Comparison]

a. In your own words, define the term, “graphical encoding” and give an example

Graphical encoding is the process of transforming data values into visual elements in a chart or graph. These elements include position, color, size, shape, and orientation, and they allow viewers to interpret patterns, trends, and differences in the data without reading exact numbers.

Example: In a bar chart, each bar’s height represents the magnitude of a numeric value this is a graphical encoding using position. Similarly, in a choropleth map, color intensity can encode population density across regions.

b. The code below reads in a dataset of monthly unemployment rates in California from 2014 through 2020. These data are visualized using three different techniques below. Describe the relative strengths and weaknesses of two of these views with respect to the tasks that they support.

Among the three approaches to visualizing the CalFresh unemployment dataset, Approach 1 and Approach 3 offer useful yet distinct perspectives. Approach 1, which uses small multiples of line plots for selected counties, is particularly effective for showing monthly trends and changes over time. It allows for easy detection of seasonal

patterns or sudden spikes, such as the noticeable increase in unemployment during 2020. However, it does not scale well to all counties, and because it lacks spatial context, it is difficult to identify regional disparities across the state.

In contrast, Approach 3 presents a series of choropleth maps, one for each year, highlighting the maximum unemployment rate per county. This approach excels at revealing geographic patterns, such as clusters of high or low unemployment. It supports spatial reasoning and makes it easy to compare county-level differences year by year. On the downside, it abstracts away the monthly variation, showing only annual extremes, and small differences in unemployment rates can be hard to notice depending on the color scale. Together, these approaches illustrate the trade-off between temporal detail and spatial insight.

```
library(tsibble)
library(sf)
library(dplyr)
library(tsibble)

unemployment <- read_sf("https://raw.githubusercontent.com/krisrs1128/stat436_s25/main/data/unemployment.geojson") |>
  mutate(date = yearmonth(date))

focus <- c("Colusa", "Monterey", "San Francisco", "Imperial", "Los Angeles", "San Joaquin")

head(unemployment, 3)
```

```
## Simple feature collection with 3 features and 4 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -122.3316 ymin: 37.45444 xmax: -121.4693 ymax: 37.90502
## Geodetic CRS:  WGS 84
## # A tibble: 3 × 5
##   county      date  year unemployment                                geometry
##   <chr>      <mth> <dbl>        <dbl>                     <MULTIPOLYGON [°]>
## 1 Alameda 2014 Jan  2014          6.5 (((-122.3129 37.89733, -122.2885 37.89793…
## 2 Alameda 2014 Feb  2014          6.4 (((-122.3129 37.89733, -122.2885 37.89793…
## 3 Alameda 2014 Mar  2014          6.4 (((-122.3129 37.89733, -122.2885 37.89793…
```

```
#c.Provide code to answer: For each county and year, what was the maximum unemployment rate across all months?

#This code groups the unemployment dataset by county and year, then summarizes it by taking the maximum unemployment rate for each group. The result shows the highest monthly unemployment rate per county in each year.

max_unemployment <- unemployment %>%
  group_by(county, year) %>%
  summarise(max_rate = max(unemployment, na.rm = TRUE)) %>%
  ungroup()
```

```
## `summarise()` has grouped output by 'county'. You can override using the
## `.groups` argument.
```

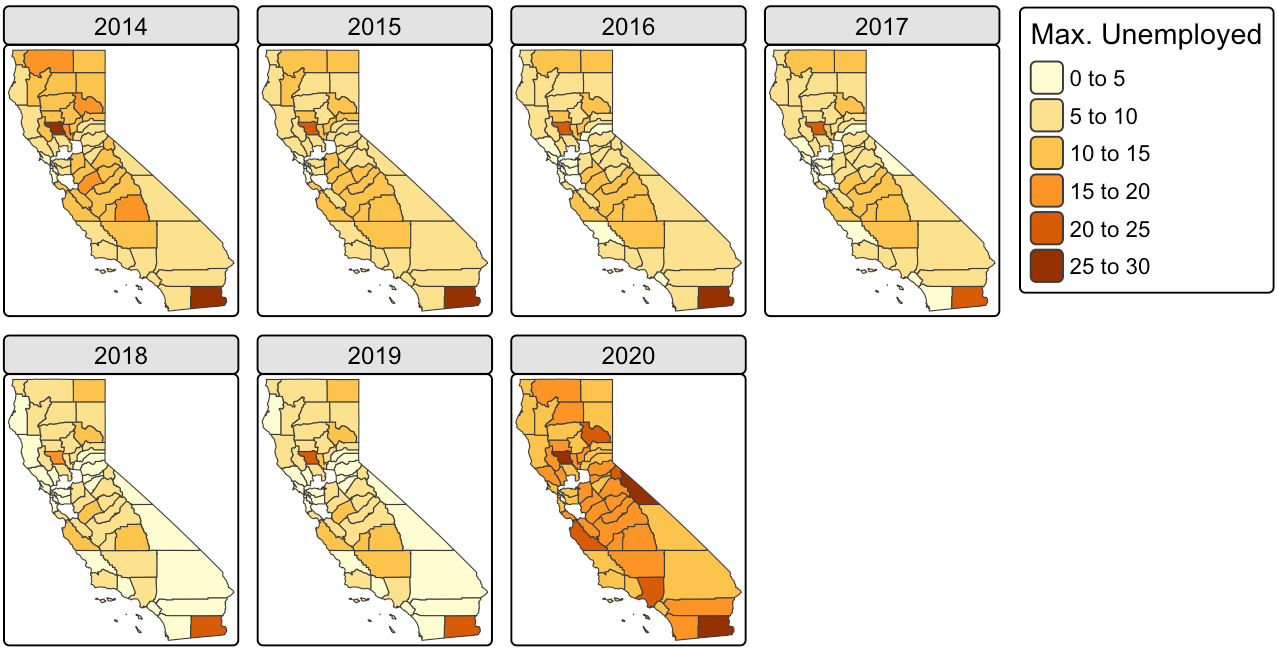
```
geom_data <- unemployment %>%
  group_by(county, year) %>%
  slice(1) %>%
  ungroup() %>%
  select(county, year, geometry)


unemp_max_map <- geom_data %>%
  left_join(as.data.frame(max_unemployment), by = c("county", "year"))
```

```
#d. Provide code to implement one of the figures above. You may assume the result from p
art (c) and do not need to match color or text themes.

tmap_mode("plot")
```

```
## ℹ tmap mode set to "plot".
```

```
tm_shape(unemp_max_map) +
  tm_polygons(
    fill = "max_rate",
    fill.scale = tm_scale_intervals(
      breaks = c(0, 5, 10, 15, 20, 25, 30),
      values = "brewer.yl_or_br"
    ),
    fill.legend = tm_legend(title = "Max. Unemployed")
  ) +
  tm_facets(by = "year", ncol = 4) +
  tm_title("Approach 3") +
  tm_layout(
    legend.outside = TRUE,
    panel.label.bg.color = "gray90"
  )
```

Approach 3

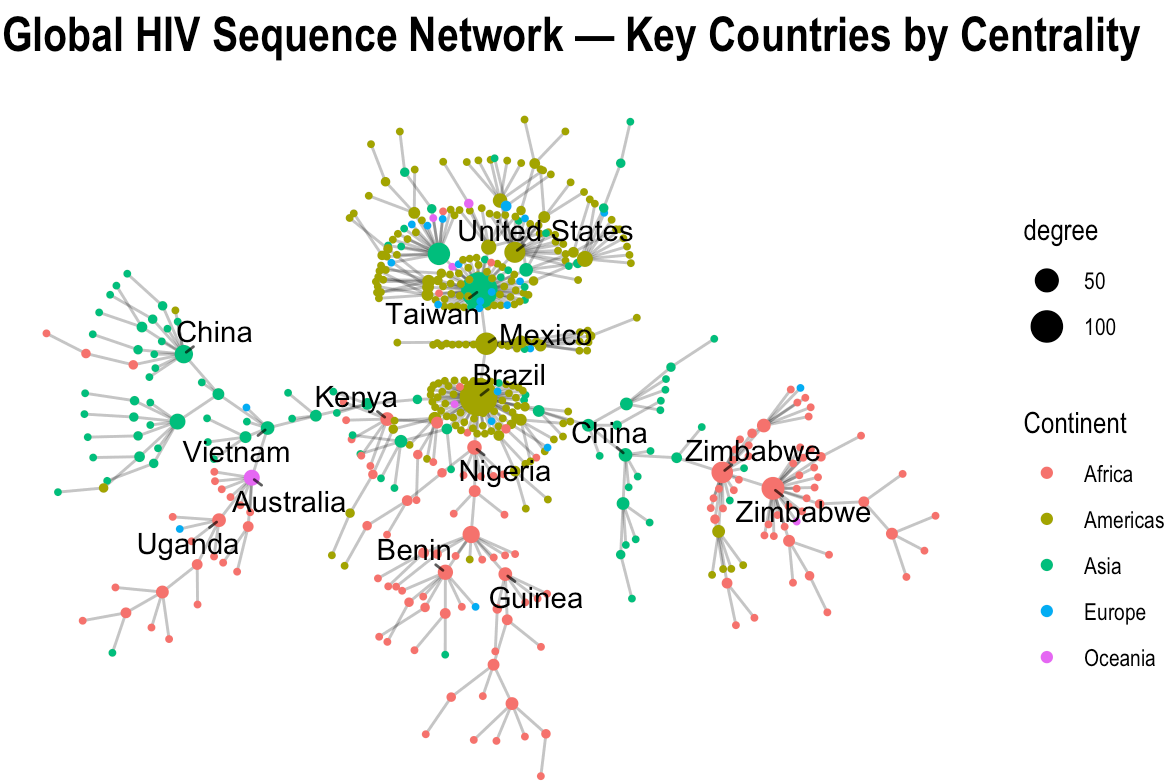

2014
2015
2016
2017
2018
2019
2020
Max. Unemployed
0 to 5
5 to 10
10 to 15
15 to 20
20 to 25
25 to 30

[HIV Network]

```
#a. Provide code to generate a node-link visualization similar to that in Figure 14. Mak
e at least one style customization that improves the visualization. Justify your choice.

#The following code creates a node-link visualization of the HIV similarity network. Nod
es are colored by continent and sized by degree centrality, while labels are only shown
for representative nodes in each country with high centrality. This improves the clarity
of the visualization by avoiding overplotting and highlighting key countries. Edges are
drawn with transparency to reduce visual clutter.


hiv_network <- tbl_graph(
  read_csv("https://github.com/krisrs1128/stat436_s25/raw/refs/heads/main/data/hiv_node
s.csv", show_col_types = FALSE),
  read_csv("https://github.com/krisrs1128/stat436_s25/raw/refs/heads/main/data/hiv_edge
s.csv", show_col_types = FALSE)
) %>%
  to_undirected() %>%
  mutate(community = as.factor(group_louvain())) %>%
  activate(nodes) %>%
  mutate(
    degree = centrality_degree(),
    betweenness = centrality_betweenness(),
    continent = countrycode(name, "country.name", "continent",
                            custom_match = c("Taiwan" = "Asia")),
    continent = case_when(
      continent %in% c("North America", "South America") ~ "Americas",
      continent %in% c("Australia", "Oceania") ~ "Oceania",
      is.na(continent) ~ "Other",
      TRUE ~ continent
    )
  )

hiv_network <- hiv_network %>%
  activate(nodes) %>%
  group_by(name) %>%
  mutate(
    influence_score = (rank(degree) + rank(betweenness)) / 2,
    is_top_country = influence_score == max(influence_score)
  ) %>%
  ungroup() %>%
  mutate(
    label_threshold = quantile(degree, 1 - (30 / n())),
    show_label = is_top_country & degree > label_threshold
  )

# Plot the network
ggraph(hiv_network) +
  geom_edge_link(alpha = 0.15) +
  geom_node_point(aes(color = continent, size = degree)) +
  scale_size_area() +
```

```
  geom_node_text(
    data = function(x) filter(x, show_label),
    aes(label = name),
    repel = TRUE,
    segment.color = "black",
    segment.alpha = 0.7,
    segment.size = 0.5,
    min.segment.length = 0
  ) +
  theme_graph() +
  labs(
    title = "Global HIV Sequence Network — Key Countries by Centrality",
    color = "Continent"
  )
```

```
## Using "stress" as default layout
```



```
#b. The figure below shows the node-link visualization for a simple graph. Sketch its ex
act adjacency matrix.

edges <- matrix(c(
  1, 2,
  1, 3,
  1, 5,
  2, 4
), byrow = TRUE, ncol = 2)


adj_matrix <- matrix(0, nrow = 5, ncol = 5)


for (i in 1:nrow(edges)) {
  from <- edges[i, 1]
  to <- edges[i, 2]
  adj_matrix[from, to] <- 1
  adj_matrix[to, from] <- 1
}

rownames(adj_matrix) <- colnames(adj_matrix) <- as.character(1:5)

adj_matrix
```

```
##   1 2 3 4 5
## 1 0 1 1 0 1
## 2 1 0 0 1 0
## 3 1 0 0 0 0
## 4 0 1 0 0 0
## 5 1 0 0 0 0
```

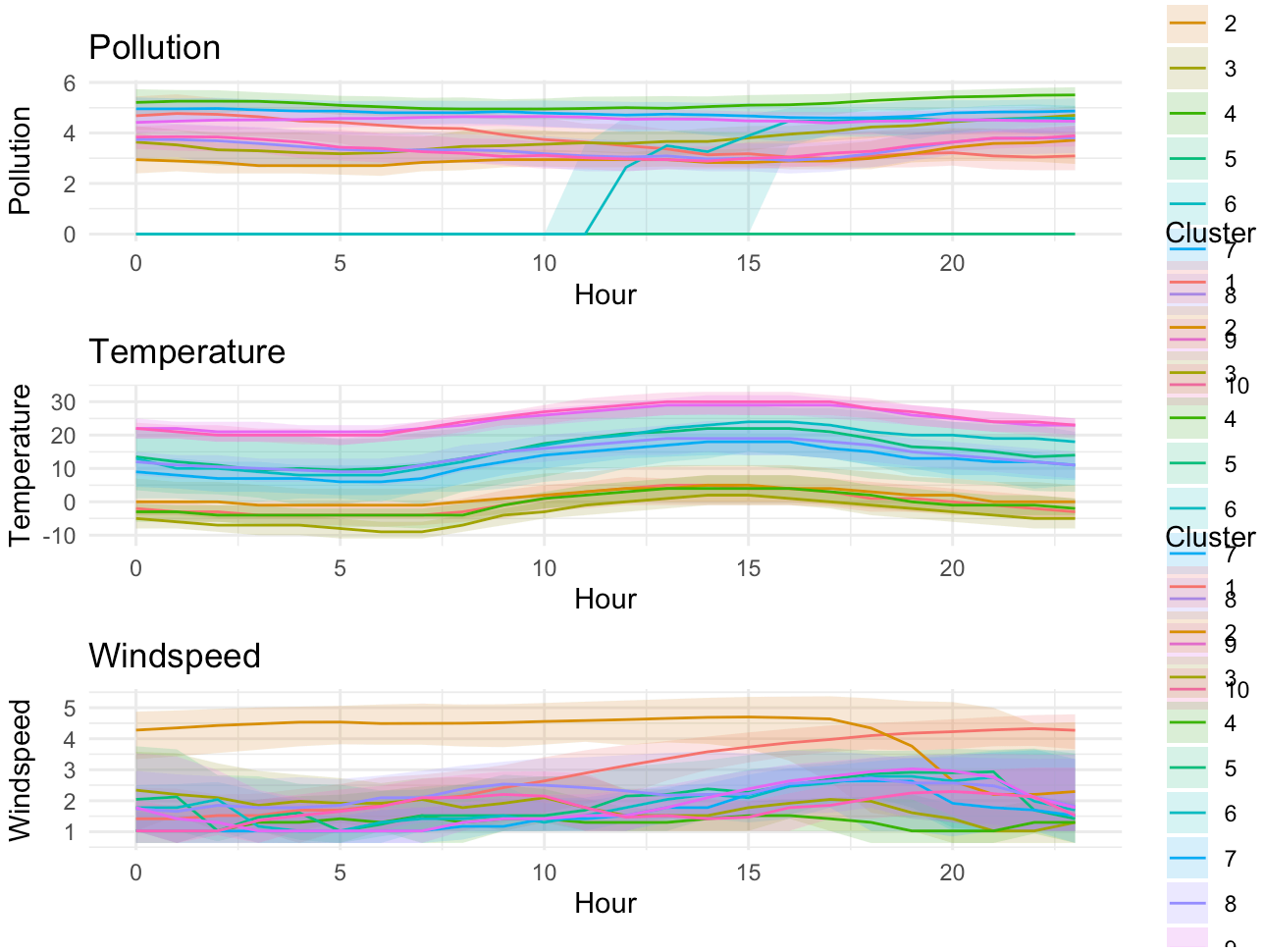
```
#c. The figure below shows the node-link visualization for a more complex graph. How do you expect its corresponding adjacency matrix visualization to look? Provide a rough sketch and a one-sentence justification.

#The adjacency matrix will show a block-diagonal structure, with dense blocks along the diagonal representing tightly connected clusters and sparse off-diagonal areas due to few connections between the clusters.

#Since the graph shows multiple dense clusters with very few inter-cluster edges, its adjacency matrix would exhibit a block-diagonal structure, where each block corresponds to a tightly connected community.


size_A <- 5
size_B <- 4
size_C <- 6


block_A <- matrix(1, nrow = size_A, ncol = size_A)
block_B <- matrix(1, nrow = size_B, ncol = size_B)
block_C <- matrix(1, nrow = size_C, ncol = size_C)


diag(block_A) <- 0
diag(block_B) <- 0
diag(block_C) <- 0


adj_matrix <- Matrix::bdiag(block_A, block_B, block_C)

adj_matrix <- as.matrix(adj_matrix)
rownames(adj_matrix) <- colnames(adj_matrix) <- paste0("N", 1:nrow(adj_matrix))


adj_matrix
```

```
##     N1 N2 N3 N4 N5 N6 N7 N8 N9 N10 N11 N12 N13 N14 N15
## N1   0  1  1  1  1  0  0  0  0   0   0   0   0   0   0
## N2   1  0  1  1  1  0  0  0  0   0   0   0   0   0   0
## N3   1  1  0  1  1  0  0  0  0   0   0   0   0   0   0
## N4   1  1  1  0  1  0  0  0  0   0   0   0   0   0   0
## N5   1  1  1  1  0  0  0  0  0   0   0   0   0   0   0
## N6   0  0  0  0  0  0  1  1  1   0   0   0   0   0   0
## N7   0  0  0  0  0  1  0  1  1   0   0   0   0   0   0
## N8   0  0  0  0  0  1  1  0  1   0   0   0   0   0   0
## N9   0  0  0  0  0  1  1  1  0   0   0   0   0   0   0
## N10  0  0  0  0  0  0  0  0  0   0   1   1   1   1   1
## N11  0  0  0  0  0  0  0  0  0   1   0   1   1   1   1
## N12  0  0  0  0  0  0  0  0  0   1   1   0   1   1   1
## N13  0  0  0  0  0  0  0  0  0   1   1   1   0   1   1
## N14  0  0  0  0  0  0  0  0  0   1   1   1   1   0   1
## N15  0  0  0  0  0  0  0  0  0   1   1   1   1   1   0
```

[Beijing Air Pollution (part a only)]

```
beijing_data <- read.csv("/Users/hyoungwookim/Desktop/stat436_s25/data/pollution_wide.csv")
```

```
#a. We performed hierarchical clustering on daily pollution, temperature, and windspeed
trajectories in Beijing from 2010 to 2014, after scaling the variables to ensure equal i
nfluence. The dendrogram was cut to yield 10 clusters. For each cluster, we visualized t
he 25th, 50th (median), and 75th percentiles of pollution, temperature, and windspeed ov
er the course of a day using ribbon plots. Several distinct patterns emerged. Clusters 4
and 6 show consistently high pollution levels and low windspeed, suggesting stagnant and
heavily polluted days. In contrast, Cluster 1 exhibits lower pollution and higher windsp
eed, indicating cleaner and windier conditions, likely in colder months. Cluster 10 show
s high temperatures and moderate pollution, consistent with hot summer days. These patte
rns highlight meaningful groupings of daily weather-pollution profiles in Beijing.

date_col <- beijing_data$date
beijing_numeric <- beijing_data %>% select(-date)

scaled_data <- scale(beijing_numeric)

hc <- hclust(dist(scaled_data), method = "ward.D2")


clusters <- cutree(hc, k = 10)
beijing_data$cluster <- clusters


extract_quantiles <- function(df, prefix) {
  df %>%
    select(cluster, starts_with(prefix)) %>%
    pivot_longer(-cluster, names_to = "hour", values_to = "value") %>%
    mutate(hour = as.integer(str_extract(hour, "\\d+"))) %>%
    group_by(cluster, hour) %>%
    summarise(q25 = quantile(value, 0.25),
              q50 = quantile(value, 0.50),
              q75 = quantile(value, 0.75), .groups = "drop")
}

pollution_q <- extract_quantiles(beijing_data, "pollution")
temp_q <- extract_quantiles(beijing_data, "temp")
wndspd_q <- extract_quantiles(beijing_data, "wndspd")

ribbon_plot <- function(df, title) {
  ggplot(df, aes(x = hour)) +
    geom_ribbon(aes(ymin = q25, ymax = q75, fill = factor(cluster)), alpha = 0.2) +
    geom_line(aes(y = q50, color = factor(cluster))) +
    labs(title = title, x = "Hour", y = title, fill = "Cluster", color = "Cluster") +
    theme_minimal()
}

p1 <- ribbon_plot(pollution_q, "Pollution")
p2 <- ribbon_plot(temp_q, "Temperature")
p3 <- ribbon_plot(wndspd_q, "Windspeed")
p1 / p2 / p3
```




```
set.seed(1)
sample_dates <- beijing_data %>%
  select(date, cluster) %>%
  group_by(cluster) %>%
  slice_sample(n = 3)
sample_dates
```

```
## # A tibble: 30 × 2
## # Groups:   cluster [10]
##    date       cluster
##    <chr>        <int>
##  1 2012-12-01       1
##  2 2011-01-31       1
##  3 2010-01-02       1
##  4 2014-11-02       2
##  5 2010-12-30       2
##  6 2010-04-01       2
##  7 2013-01-15       3
##  8 2012-02-04       3
##  9 2012-01-21       3
## 10 2012-01-09       4
## # ℹ 20 more rows
```

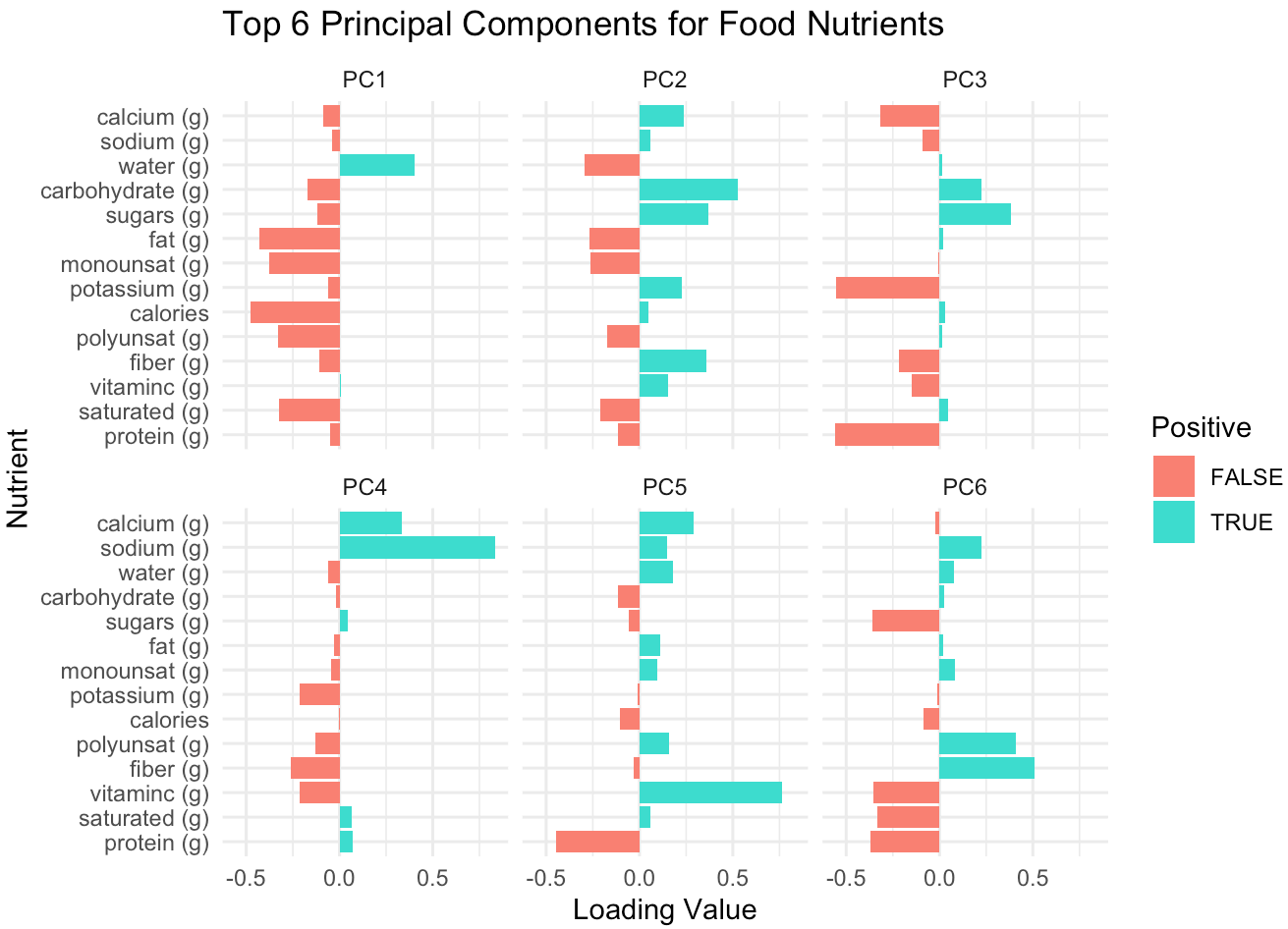
[Food Nutrients]

```
nutrients <- read_csv("https://uwmadison.box.com/shared/static/nmgouzobq5367aex45pnbzgkhm7sur63.csv")
```

```
## Rows: 7637 Columns: 18
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (3): name, group, group_lumped
## dbl (15): id, protein (g), calcium (g), sodium (g), fiber (g), vitaminc (g),...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#a.  Define a tidymodels recipe that normalizes all nutrient features and specifies that PCA should be performed.

pca_recipe <- recipe(~ `protein (g)` + `calcium (g)` + `sodium (g)` + `fiber (g)` +
                       `vitaminc (g)` + `potassium (g)` + `carbohydrate (g)` + `sugars (g)` +
                       `fat (g)` + `water (g)` + calories + `saturated (g)` +
                       `monounsat (g)` + `polyunsat (g)`,
                     data = nutrients) %>%
  step_normalize(all_predictors()) %>%
  step_pca(all_predictors(), num_comp = 6)


pca_prep <- prep(pca_recipe)


pca_data <- bake(pca_prep, new_data = NULL)


pca_data <- bind_cols(
  nutrients %>% select(name, group),
  pca_data
)
```

```
#b. Visualize the top 6 principal components. What types of food do you expect to have l
ow or high values for PC1 or PC2?

# Based on the PCA loadings shown in the graph, PC1 appears to represent overall energy
density and fat content. It shows strong positive loadings for calories, fat, saturated
fat, and monounsaturated fat, and strong negative loadings for water and vitamin C. Ther
efore, foods with high PC1 scores are likely to be energy-dense and high in fat—such as
oils, butter, and fried foods—while low PC1 foods are water rich and lower in calories,
such as fruits, vegetables, and beverages. PC2 contrasts carbohydrates, sugars, and pota
ssium (positive loadings) with sodium, fat, and saturated fat (negative loadings). This
suggests that high PC2 foods are likely to be rich in natural sugars and plant-based nut
rients, like fruits or cereals, while low PC2 foods may be more salty, fatty, or protein
-rich, such as meats, cheeses, or processed savory snacks.


pca_obj <- tidy(pca_prep, number = 2)


top_pcs <- pca_obj %>%
  filter(component %in% paste0("PC", 1:6)) %>%
  mutate(
    Positive = value > 0,
    terms = fct_reorder(terms, value)
  )

ggplot(top_pcs, aes(x = value, y = terms, fill = Positive)) +
  geom_col() +
  facet_wrap(~ component, nrow = 2) +
  labs(
    title = "Top 6 Principal Components for Food Nutrients",
    x = "Loading Value",
    y = "Nutrient"
  ) +
  scale_fill_manual(values = c("TRUE" = "turquoise", "FALSE" = "salmon")) +
  theme_minimal() +
  theme(legend.position = "right")
```



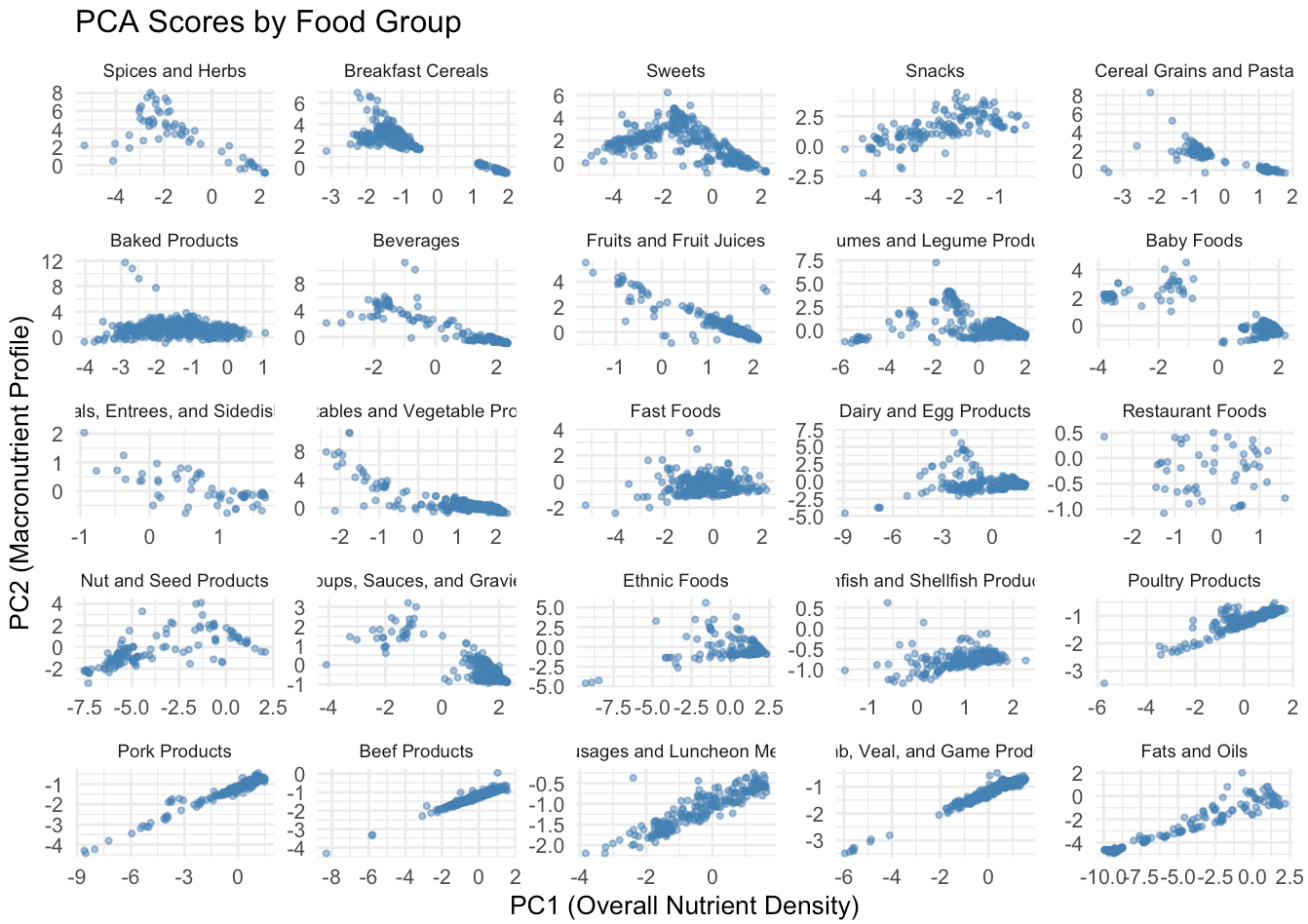

```
#c. Compute the average value of PC2 within each category of the group column. Give the
names of the groups sorted by this average.

pc2_group_avg <- pca_data %>%
  group_by(group) %>%
  summarize(avg_PC2 = mean(PC2, na.rm = TRUE)) %>%
  arrange(desc(avg_PC2))


pc2_group_avg$group
```

```
##  [1] "Spices and Herbs"                  "Breakfast Cereals"
##  [3] "Sweets"                            "Snacks"
##  [5] "Cereal Grains and Pasta"           "Baked Products"
##  [7] "Beverages"                         "Fruits and Fruit Juices"
##  [9] "Legumes and Legume Products"       "Baby Foods"
## [11] "Meals, Entrees, and Sidedishes"    "Vegetables and Vegetable Products"
## [13] "Fast Foods"                        "Dairy and Egg Products"
## [15] "Restaurant Foods"                  "Nut and Seed Products"
## [17] "Soups, Sauces, and Gravies"        "Ethnic Foods"
## [19] "Finfish and Shellfish Products"    "Poultry Products"
## [21] "Pork Products"                     "Beef Products"
## [23] "Sausages and Luncheon Meats"       "Lamb, Veal, and Game Products"
## [25] "Fats and Oils"
```

#d. Visualize the scores of each food item with respect to the first two principal components. Facet the visualization according to the group column, and sort the facets according to the results of part (c). How does the result compare with your guess from part (b)?

 # The faceted PCA visualization shows clear distinctions in how food groups distribute along the first two principal components. Groups like Fats and Oils, Pork Products, and Beef Products are positioned far to the right on the PC1 axis, indicating high values in fat and calorie-related nutrients, which matches our earlier interpretation from part (b). In contrast, groups such as Spices and Herbs, Fruits and Fruit Juices, and Vegetables and Vegetable Products tend to lie toward the left on PC1, reflecting low energy density and high water or vitamin content. Regarding PC2, food groups like Breakfast Cereals, Sweets, and Fruits appear higher, indicating higher sugar and carbohydrate content. On the other hand, groups such as Poultry, Beef, and Restaurant Foods are positioned lower, consistent with their high protein, fat, and sodium content. Overall, the results of the visualization are consistent with the expectations from the loading plots in part (b), validating our interpretation of what PC1 and PC2 represent nutritionally.

```
pca_data <- pca_data %>%
  mutate(group = factor(group, levels = pc2_group_avg$group))

ggplot(pca_data, aes(x = PC1, y = PC2)) +
  geom_point(alpha = 0.5, size = 0.8, color = "steelblue") +
  facet_wrap(~ group, scales = "free") +
  labs(
    title = "PCA Scores by Food Group",
    x = "PC1 (Overall Nutrient Density)",
    y = "PC2 (Macronutrient Profile)"
  ) +
  theme_minimal(base_size = 10) +
  theme(
    strip.text = element_text(size = 7),
    legend.position = "none"
  )
```


PCA Scores by Food Group
Spices and Herbs
Breakfast Cereals
Sweets
Snacks
Cereal Grains and Pasta
Baked Products
Beverages
Fruits and Fruit Juices
umes and Legume Produ
Baby Foods
als, Entrees, and Sidedis
ables and Vegetable Pro
Fast Foods
Dairy and Egg Products
Restaurant Foods
Nut and Seed Products
oups, Sauces, and Gravi
Ethnic Foods
nfish and Shellfish Produ
Poultry Products
Pork Products
Beef Products
sages and Luncheon Me
b, Veal, and Game Prod
Fats and Oils
PC2 (Macronutrient Profile)
PC1 (Overall Nutrient Density)

[Code Diary]

#a. What was your programming environment/workflow? I used RStudio as my primary program ming environment, mainly working in an R Markdown notebook. This setup allowed me to wri te, test, and document my code in the same file, which helped me keep my analysis organi zed and reproducible. Occasionally, I also used the console to quickly check variable ou tputs or test single lines of code.

#b. What techniques did you use to plan your code? To plan my code, I broke the task int o smaller steps before starting. I listed out what each step needed to accomplish (e.g., data cleaning, performing PCA, visualizing results) and worked through them one at a tim e. I also looked back at lecture examples to understand how to structure my recipe and t idy evaluation. When I wasn't sure how to do something, I would write comments in my cod e to guide myself or check online documentation to plan the next step.

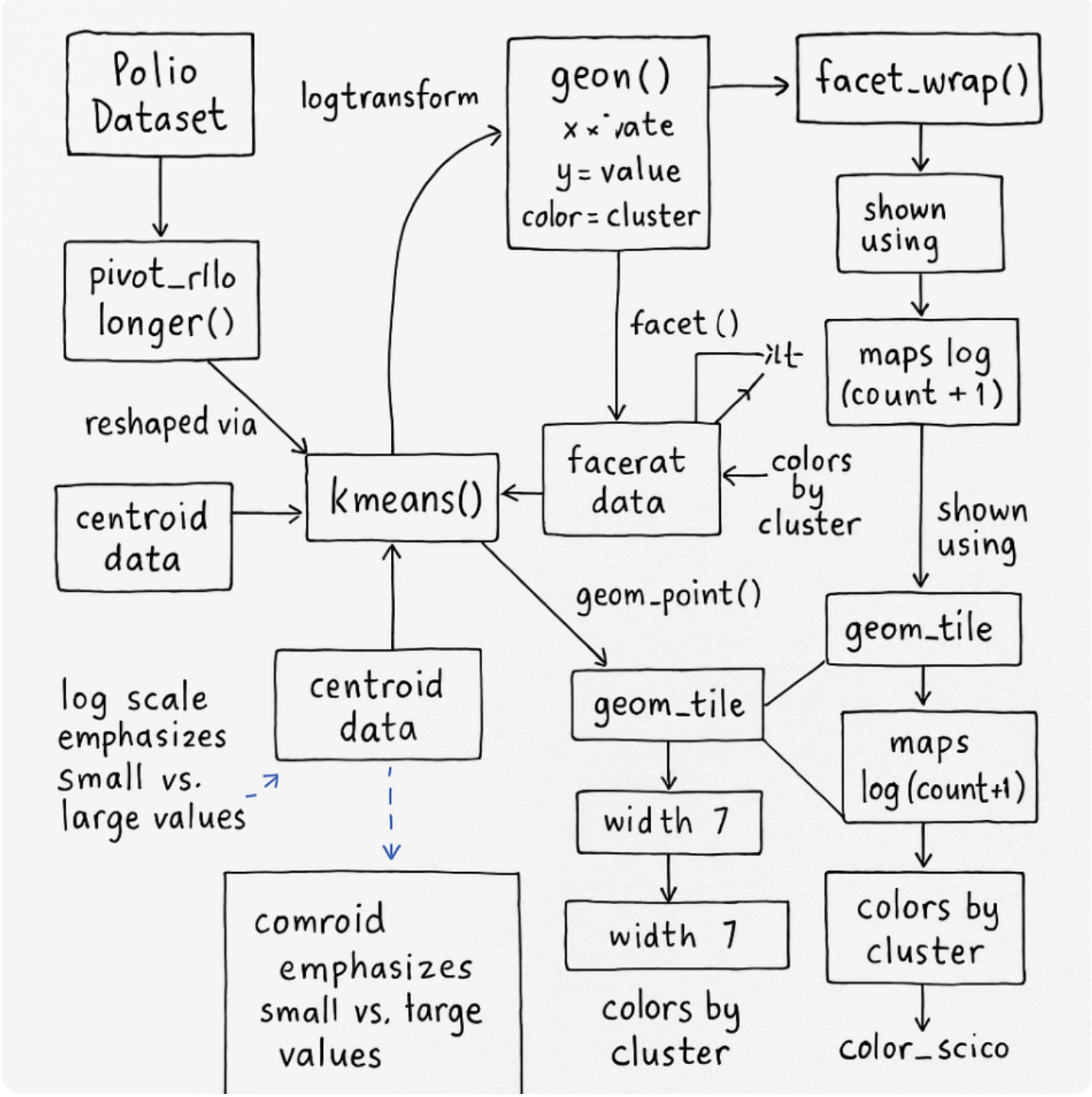
#c. What kinds of bugs did you encounter? What steps did you take to resolve them?

#While using geom_ribbon() to create ribbon plots by cluster, I ran into several bugs re lated to the required aesthetics. Specifically, the variables for ymin and ymax (i.e., t he 25th and 75th percentiles) were either missing or incorrectly formatted after using p ivot_longer() and summarise(). At times, column names were misaligned or contained non-n umeric values such as function names (closures), which caused coercion errors in ggplot 2. To fix these issues, I inspected the structure of the data with glimpse() and head(), made sure that column names were correct, and used as.numeric() where necessary to enfor ce proper types. I also referred to the geom_ribbon() documentation and tested small chu nks of code step-by-step to isolate and resolve each issue.

#d. If you were starting the programming task again, would you do anything differently?

#Yes. If I were starting the task again, I would make sure that the data structure was f ully compatible with geom_ribbon() before proceeding to plotting. That includes verifyin g that ymin, ymax, and x values exist and are in the correct numeric format. I would als o use glimpse() and intermediate visual checks more frequently throughout the transforma tion process, especially after reshaping and summarizing the data. Breaking the task int o smaller testable units and validating each step would have made debugging easier. Fina lly, I would take some time upfront to review the geom_ribbon() function documentation t o clearly understand its input requirements before implementing the plot.

[Concept Map]



K-means clustering concept map

```
#Summary
#This concept map illustrates the full pipeline of clustering time-series data using the
polio dataset. The process begins by reshaping the dataset with pivot_wider() to prepare
it for clustering. The transformed data is passed into the kmeans() function after optio
nal scaling and transposition to compute clusters across states based on temporal polio
case patterns. After clustering, two main types of visualization are used. First, tidy(f
it) extracts centroid data, which is then plotted using geom_point() and facet_wrap() to
display the temporal signature of each cluster. Here, aesthetics like x = date, y = valu
e, and color = cluster are mapped using aes().Second, augment(fit, x) attaches cluster a
ssignments to the full dataset, allowing us to create heatmap-style visualizations with
geom_tile(). These tiles are faceted by cluster using facet_grid() and use log(1 + valu
e) for color scaling to emphasize variation in case intensity. Aesthetic encodings like
fill and color are controlled via scale_fill_scico() and scale_color_scico().
```