Lab 4 Instructions

PUBH 523/623

Nicky Wakim

Directions

Please turn in your **.qmd** and **.html** file on Brightspace. Please use the same naming convention as previous labs: `lastname_firstinit_lab_04.qmd` and `lastname_firstinit_lab_04.html`.

You can download the `.qmd` file for this lab here.

The above link will take you to your **editing** file. **Please do not remove anything from this editing file!!** You will only add your code and work to this file.e.

Purpose

In Lab 3, you built a categorical version of your composite score(s), a clean overall summary table, and a grouped summary comparing your composite score across the groups of your other main variable. This lab turns those summaries into the actual **visualizations** you'll use in your final project.

Your final project's one-page information sheet is built around three findings:

1. One main variable on its own
2. The relationship between your two main variables
3. One main variable alongside one additional variable

For Findings 2 and 3, you'll use the **categorical** version of your composite score (from Lab 3, Section 1.4) rather than the raw numeric score — category breakdowns like "33% described access as difficult" are easier for a general audience to take in at a glance than a mean and standard deviation.

This lab has you build one polished visualization for each of these. By the end of this lab, you should have 2-3 finished, publication-ready plots (or a table image) saved as image files — the same files you'll drag directly into your final project template. Some of the `ggplot2` code here is more involved than what you've used before, so this lab walks through each new piece in more detail than usual — take your time with it.

Lab activities

Restate your selected project

Recall the following table from Labs 2 and 3 with each project's population, main variables, and composite score(s).

#	Project Question	Population	Population variables	Main variables	Suggested additional variables	Composite score
1	For individuals under the federal poverty line, how do food support systems help individuals' food security?	People under 100% federal poverty line (FPL) and received free groceries or meals in the past 12 months	`FPL_LT50`, `FPL50_100`	`FOOD_INSEC`, `Q26D_1`, `Q26D_2`, `Q26D_3`, `Q26D_4`	`Q23`, `PPETHM`, `Q37`, `NUM_CHILD_HH`, `Q25`, `Q25A`, `Q26`, `Q26A`	Composite score of ease of access to free groceries or meals (combined score from `Q26D_`)
2	For individuals who identify as LGBTQ, how does trust and/or fair treatment in healthcare effect self-reported health?	People who identify as LGBTQ and have a healthcare provider	`LGBTQ`, `Q108`	`Q108B_2`, `Q108B_1`, `Q27`	`Q108`, `Q70F_4`, `Q70F_5`, `PPETHM`, `Q280`	Level of trust, courtesy, and respect around treatment (combined score from `Q108B_1` and `Q108B_2`)

#	Project Question	Population	Population variables	Main variables	Suggested additional variables	Composite score
3	For individuals with school-aged children and below the federal poverty line, how do benefits received effect their mental health?	People who have at least one child (aged 5-18 years old) in the household and are under the 100% FPL	`ANY_CHILDHH_5_18`, `FPL_LT50`, `FPL50_100`	`Q30_1-Q30_6`, `Q53_A-Q53_J`, `Q11`, `Q31_4`	`NUM_CHILD_HH`, `Q1`, `Q37`, `PPETHM`, `Q27`	Total mental health (combined scores from `Q30_`), Total benefits (combined scores from `Q53_`, `Q11`, and `Q31_4`)

! Task

Copy and paste the row from the above table that corresponds to your project into your `.qmd` file, same as you did in Lab 3.

Import your saved data from Lab 3

! Task

From the `.rda` file you exported at the end of Lab 3, import `wbns_06_cat` and `summary_table_overall` into R using `load()` or `import()` from the `rio` package. Print out the number of rows and columns of `wbns_06_cat` to verify the import.

Plan your three findings

In the final project, we will present 2-3 findings. These findings will include a table or plot to demonstrate the data/relationship. Before writing any plotting code, decide what each of your findings will show and which variable(s) it needs. This is a planning step, there is no code required here!

- **Finding 1 (one main variable on its own):** Which one of your main variables will you visualize by itself? This could be your composite score's categorical version (from `wbns_06_cat` from Lab 3) or your other main variable (e.g., `FOOD_INSEC`, `Q27`, or your `total_benefits` category).
- **Finding 2 (the relationship between your two main variables):** No decisions here! You will visualize the relationship between your composite score's categorical version compared across the groups of your other main variable.
- **Finding 3, optional (one main variable + one additional variable):** If you're building this one, which additional variable from your Lab 2 table will you use, and which main variable will you pair it with? (Feel free to reference the table above for suggested additional variables.)

> **!** Task
>
> For Finding 1 and 3, write down which variable(s) you will visualize.

Finding 1: create the dataframe

We want to visualize one of your main variables on its own. The variable you choose to visualize should be one of the main variables from the table above. This will be a categorical version of your composite score (from Lab 3) or the other main variable (e.g., `FOOD_INSEC`, `Q27`).

Before plotting, we want to summarize the data into percentages to make our display a little more polished for the audience. We will summarize our main variable into a small dataframe of counts and percentages using `count()` and `mutate()`:

```
finding1_data <- wbns_06_cat |>
  count(<your_main_variable>) |>                    ①
  mutate(pct = n / sum(n) * 100)                    ②
```

① Replace `<your_main_variable>` with the name of your main variable. `count()` counts how many respondents fall into each category of that variable, storing the result in a new column called `n`.

② `mutate(pct = n / sum(n) * 100)` divides each category's count by the total across *all* categories, turning `n` into a percentage. Since there's no grouping variable yet, `sum(n)` here is just the overall total.

> **!** Task
>
> Use `count()` and `mutate()` to build a small dataframe of counts and percentages for your Finding 1 variable. Save it as a new object called `finding1_data`.

Finding 1: Visualize one main variable

Using the `finding1_data` dataframe you just built, create a bar chart of your main variable using `ggplot2`. The code below gives a starting point for this plot. Note that we are using `geom_col()` instead of `geom_bar()` here because we already calculated the percentages ourselves in `finding1_data`. `geom_col()` plots the y-values you give it directly, while `geom_bar()` would try to do its own internal counting.

```
finding1_plot <- ggplot(finding1_data, aes(x = <your_main_variable>, y = pct)) +
  geom_col() +                                                                  ①
  geom_text(aes(label = paste0(round(pct, 1), "%")), vjust = -0.5,              ②
            size = 6, fontface = "bold") +
  scale_x_discrete(labels = ________) +                                         ③
  scale_y_continuous(limits = ______) +                                         ④
  labs(______) +                                                                ⑤
  theme_minimal(base_size = 12)                                                 ⑥

finding1_plot                                                                   ⑦
```

① Check out the extra arguments in `geom_col()`! You can change the fill or the width of the bars.

② `geom_text()` adds the percentage labels above each bar. `vjust = -0.5` nudges the labels slightly above the top of the bar, and `paste0(round(pct, 1), "%")` rounds the percentage to one decimal place and adds a percent sign.

③ Your data still has the raw category values (e.g., `"Yes"` and `"No"`) — `scale_x_discrete(labels = ...)` lets you swap in reader-friendly labels for the x-axis *without changing your underlying data*. The names on the left of each `=` (`"No"`, `"Yes"`) must exactly match the values in your data; the names on the right are what the reader will actually see.

④ We can add limits to the y-axis. When we plot percentages, it is often a good idea to plot the y-axis from 0 to 100. Check out the `limits` argument in `scale_y_continuous()` to set the y-axis limits.

⑤ `labs()` lets you add a title, x-axis label, and y-axis label. Use plain-language labels rather than variable names or question codes.

⑥ We can adjust the theme and font size!

⑦ Now we can print the plot!

> **!** Task
>
> Build your Finding 1 visualization using `ggplot2`, adding and editing the code above with your own variable and category labels. Save the plot as a new object (e.g., `finding1_plot`).

Finding 2: create the dataframe

This is the heart of your project — the finding most directly built from your Lab 3 work. Use the **categorical** version of your composite score here (e.g., `ease_free_food_cat`) rather than the raw numeric score — a reader can take in "33% described access as difficult" much faster than a mean and standard deviation.

Unlike Finding 1, you now need percentages calculated *within* each group of your other main variable (e.g., what percentage of the "food insecure" group falls into each access category?). `count()` and `mutate()` alone can't do that "within-group" calculation, so this time you'll add `group_by()` before you calculate `pct`.

```
finding2_data <- wbns_06_cat |>
  count(<your_grouping_variable>, <your_categorical_composite>) |>          ①
  group_by(<your_grouping_variable>) |>                                     ②
  mutate(pct = n / sum(n) * 100) |>
  ungroup()

finding2_data
```

① Replace `<your_grouping_variable>` and `<your_categorical_composite>` with your two main variables. `count()` counts how many respondents fall into every *combination* of the two variables (e.g., "food insecure AND difficult access," "food insecure AND moderate access," and so on), storing the result in a new column called `n`.

② `group_by(<your_grouping_variable>)` tells the following `mutate()` to calculate `sum(n)` *separately within each group*, rather than across the whole table — that's what makes `pct` a percentage *within* each group (summing to 100% within each group) instead of a percentage of the grand total. `ungroup()` at the end removes that grouping so it doesn't silently affect anything later.

> **!** Task
>
> Use `count()`, `group_by()`, and `mutate()` to build a dataframe of within-group percentages for your two main variables. Save it as a new object called `finding2_data`.

Finding 2: Visualize the relationship between your two main variables

Using the `finding2_data` dataframe you just built, create a 100%-stacked bar chart with `ggplot2`. As in Finding 1, we use `geom_col()` instead of `geom_bar()` since you already calculated the percentages yourself. The code below gives a starting point for this plot.

```
finding2_plot <- ggplot(finding2_data, aes(x = <your_grouping_variable>, y = pct, fill = <you
  geom_col() +                                                                          ①
  geom_text(aes(label = paste0(round(pct), "%")),                                       ②
            position = position_stack(vjust = 0.5),
            color = "white", fontface = "bold", size = 4.5) +
  scale_x_discrete(labels = ________) +                                                 ③
  scale_fill_manual(values = ________, name = NULL) +                                   ④
  labs(______) +                                                                        ⑤
  theme_minimal(base_size = 12) +                                                       ⑥
  theme(legend.position = "bottom")

finding2_plot                                                                           ⑦
```

① Mapping `fill = <your_categorical_composite>` (in the `ggplot()` line above) tells `geom_col()` to draw one segment per category, stacked on top of each other by default — this is what creates the 100%-stacked bar. Check out the extra arguments in `geom_col()` too, like `width`.

② `geom_text()` places its labels at the same x/y position as the data by default, which for stacked bars means the *top* of each segment. `position = position_stack(vjust = 0.5)` moves each label to the vertical *center* of its own segment instead, so it doesn't look like it belongs to the segment above it. `paste0(round(pct), "%")` builds the label text from your calculated percentage.

③ Same as Finding 1 — swap in reader-friendly labels for your x-axis categories without changing your underlying data. The names on the left must exactly match the raw category values in your data.

④ `scale_fill_manual()` lets you assign a specific, deliberate color to each level of your fill variable (rather than `ggplot2`'s default color choices), e.g. `values = c("Difficult access" = "#01414A", "Moderate access" = "#028090", "Easy access" = "#8ED1D6")`. The names on the left must exactly match the category labels in your data — pick colors that make sense together (e.g., darker = less favorable).

⑤ Same as Finding 1 — add a plain-language title, x-axis label, and y-axis label. Keep your title **descriptive, not causal**.

⑥ Same as Finding 1 — adjust the theme and font size if you'd like.

⑦ Now we can print the plot!

> **Tip**
>
> If your composite score doesn't split cleanly into a small number of meaningful categories, a boxplot of the raw numeric score (`geom_boxplot(aes(x = <your_grouping_variable>, y = <your_numeric_composite>))`) is a reasonable alternative — just be consistent about which form (categorical or continuous) you use across Findings 2 and 3.

> **!** Task
>
> Build your Finding 2 visualization using `ggplot2`, adding and editing the code above with your own variables and category labels. Save the plot as a new object (e.g., `finding2_plot`).

Finding 3: create the dataframe

If you have time and want a third finding, pick one of the suggested additional variables from your Lab 2 table and pair it with the categorical version of one of your main variables.

> **Caution**
>
> Your additional variables haven't been cleaned yet the way your main variables were in Lab 2 — they may still contain "Don't know," "Refused," or skip-pattern missingness. Take a quick look with `count()` or `skim()` before you build this dataframe, and handle any `NA`/"Don't know"/"Refused" values the same way you did for your main variables in Lab 2 (Section 1.6). Skipping this step can quietly distort your plot.

Once your additional variable is cleaned, build the dataframe the exact same way you did for Finding 2:

```
finding3_data <- wbns_06_cat |>
  count(<your_additional_variable>, <your_categorical_composite>) |>
  group_by(<your_additional_variable>) |>
  mutate(pct = n / sum(n) * 100) |>
  ungroup()

finding3_data
```

> **!** Task
>
> Clean your chosen additional variable, then use `count()`, `group_by()`, and `mutate()` to build a dataframe of within-group percentages. Save it as a new object called `finding3_data`.

Finding 3: Visualize one main variable + one additional variable

Same approach as Finding 2, again using the categorical version of your composite score so the story stays consistent across findings.

```
finding3_plot <- ggplot(finding3_data, aes(x = <your_additional_variable>, y = pct, fill = <y
  geom_col() +
  geom_text(aes(label = paste0(round(pct), "%")),
            position = position_stack(vjust = 0.5),
            color = "white", fontface = "bold", size = 4.5) +
  scale_x_discrete(labels = ________) +
  scale_fill_manual(values = ________, name = NULL) +
  labs(______) +
  theme_minimal(base_size = 12) +
  theme(legend.position = "bottom")

finding3_plot
```

If you're unsure what any piece of this code is doing, revisit the annotated explanations under Finding 2 above — `geom_col()`, `position_stack()`, `scale_fill_manual()`, and `labs()` all work exactly the same way here.

> **!** Task
>
> Build your Finding 3 visualization using `ggplot2`, adding and editing the code above with your own variables and category labels. Save the plot as a new object (e.g., `finding3_plot`).

Export your visualizations and save your environment

Save each of your finished plots as a `.png` file. These files will be used in later in the final project! If you need to adjust the size and look of your plots, you will need to come back to this lab, modify them, and re-export them.

Example:

```
ggsave("plots/finding1_ease_food_category.png", plot = finding1_plot,
       width = 5, height = 5, units = "in", dpi = 300)

ggsave("plots/finding2_ease_food_by_insecurity.png", plot = finding2_plot,
       width = 5, height = 6, units = "in", dpi = 300)
```

- Save at `dpi = 300` or higher so your plot looks crisp on a printed page.
- Use a descriptive file name for each plot (e.g., `finding1_...`, `finding2_...`) so they're easy to find later.
- If you built a table instead of a plot for any of your findings, use `gtsave()` the same way you'll see described in your final project instructions.

! Task

Export each of your polished visualizations as a `.png` file into your `plots` folder. Include a screenshot showing the exported files in your `plots` folder.

Looking ahead to your final project

The plots you polished and exported in this lab are exactly what will go into the three finding boxes on your final project template — you shouldn't need to rebuild or re-polish them later. What's left for the final project is mostly writing: a title, a headline takeaway, background, population, a short headline and caption for each finding, a "so what," and your data source and limitations. If you want a preview of how these pieces fit together, take a look at the final project instructions and template now.