Organizing Public Health Data with gt Tables in R - Basics

GRAPH Network & WHO, supported by the Global Fund to fight HIV, TB & Malaria

October 2023

This document is a draft of a lesson made by the GRAPH Network, a non-profit headquartered at the University of Geneva Global Health Institute, in collaboration with the World Health Organization, under a Global Fund 2023 grant to create e-learning modules to build in-country data capacity for epidemiological and impact analysis for National HIV, TB and malaria programs

Introduction

Tables are a powerful tool for visualizing data in a clear and concise format. With R and the gt package, we can leverage the visual appeal of tables to efficiently communicate key information. In this lesson, we will learn how to build aesthetically pleasing, customizable tables that support data analysis goals.

Learning objectives

- Use the `gt()` function to create basic tables
- Group columns under spanner headings
- Relabel column names
- Add summary rows for groups

By the end, you will be able to generate polished, reproducible tables like this:

Sum of HIV cases in Malawi

from Q1 2019 to Q2 2019

		New cases		Previous cases	
	period	Positive	Negative	Positive	Negative
Central Region					
	2019 Q1	2004	123018	3682	2562
	2019 Q2	1913	116443	3603	1839
	2019 Q3	1916	127799	4002	2645
	2019 Q4	1691	124728	3754	1052
sum	—	7524.00	491988.00	15041.00	8098.00
mean	—	1881.00	122997.00	3760.25	2024.50
Northern Region					
	2019 Q1	664	36196	1197	675
	2019 Q2	582	35315	1084	590
	2019 Q3	570	36850	1191	542
	2019 Q4	519	34322	1132	346
sum	—	2335.00	142683.00	4604.00	2153.00
mean	—	583.75	35670.75	1151.00	538.25
Southern Region					
	2019 Q1	3531	125480	9937	3358
	2019 Q2	3637	130491	10414	3176

Example summary table

Packages

We will use these packages:

- {gt} for creating tables
- {tidyverse} for data wrangling
- {here} for file paths

```
# Load packages
pacman::p_load(tidyverse, gt, here)
```

Introducing the dataset

Our data comes from the **Malawi HIV Program** and covers antenatal care and HIV treatment during 2019. We will focus on quarterly regional and facility-level aggregates (available here).

```
# Import data
hiv_malawi <- read_csv(here::here("data/clean/hiv_malawi.csv"))
```

Let's explore the variables:

```
# First 6 rows
head(hiv_malawi)
```

```
# Variable names and types
glimpse(hiv_malawi)
```

```
## Rows: 17,235
## Columns: 29
## $ region                          <chr> "Northern R…
## $ zone                            <chr> "Northern Z…
## $ district                        <chr> "Chitipa", …
## $ traditional_authority           <chr> "Senior TA …
## $ facility_name                   <chr> "Kapenda He…
## $ datim_code                      <chr> "K9u9BIAaJJ…
## $ system                          <chr> "e-masterca…
## $ hsector                         <chr> "Public", "…
## $ period                          <chr> "2019 Q1", …
## $ reporting_period                <chr> "1st month …
## $ sub_groups                      <chr> "All patien…
## $ new_women_registered            <dbl> 45, NA, 40,…
## $ total_women_in_booking_cohort   <dbl> NA, 55, NA,…
```

```
## $ not_tested_for_syphilis                 <dbl> NA, 45, NA,…
## $ syphilis_negative                       <dbl> NA, 10, NA,…
## $ syphilis_positive                       <dbl> NA, 0, NA, …
## $ hiv_status_not_ascertained              <dbl> 4, 7, 9, 4,…
## $ previous_negative                       <dbl> 0, 0, 0, 0,…
## $ previous_positive                       <dbl> 0, 0, 0, 1,…
## $ new_negative                            <dbl> 40, 47, 30,…
## $ new_positive                            <dbl> 1, 1, 1, 1,…
## $ not_on_cpt                              <dbl> NA, 0, NA, …
## $ on_cpt                                  <dbl> NA, 1, NA, …
## $ no_ar_vs                                <dbl> 0, 0, 0, 0,…
## $ already_on_art_when_starting_anc        <dbl> 0, 1, 0, 1,…
## $ started_art_at_0_27_weeks_of_pregnancy <dbl> 1, 0, 1, 1,…
## $ started_art_at_28_weeks_of_preg         <dbl> 0, 0, 0, 0,…
## $ no_ar_vs_dispensed_for_infant           <dbl> NA, 0, NA, …
## $ ar_vs_dispensed_for_infant              <dbl> NA, 1, NA, …
```

The data covers geographic regions, healthcare facilities, time periods, patient demographics, test results, preventive therapies, antiretroviral drugs, and more. More information about the dataset is in the appendix section.

The key variables we will be considering are:

1. previous_negative: The number of patients who visited the healthcare facility in that quarter that had prior negative HIV tests.
2. previous_positive: The number of patients (as above) with prior positive HIV tests.
3. new_negative: The number of patients newly testing negative for HIV.
4. new_positive: The number of patients newly testing positive for HIV.

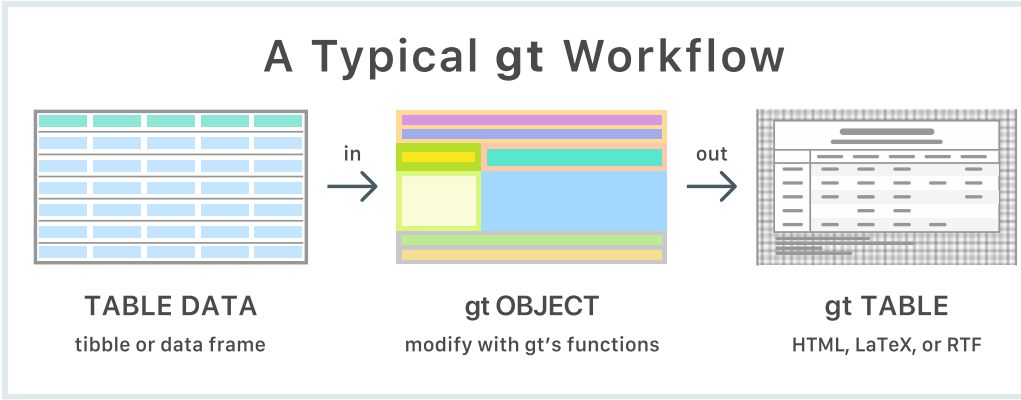
In this lesson, we will aggregate the data by quarter and summarize changes in HIV test results.

Creating simple tables with `{gt}`

`{gt}`'s flexibility, efficiency, and power make it a formidable package for creating tables in R. We'll explore some of it's core features in this lesson.

KEY POINT

KEY POINT



The `{gt}` package contains a set of functions that take raw data as input and output a nicely formatted table for further analysis and reporting.

To effectively leverage the `{gt}` package, we first need to wrangle our data into an appropriate summarized form.

In the code chunk below, we use {dplyr} functions to summarize HIV testing in select Malawi testing centers by quarter. We first group the data by time period, then sum case counts across multiple variables using `across()`:

```
# Variables to summarize
cols <- c("new_positive", "previous_positive", "new_negative",
          "previous_negative")

# Create summary by quarter
hiv_malawi_summary <- hiv_malawi %>%
  group_by(period) %>%
  summarize(
    across(all_of(cols), sum) # Summarize all columns
  )

hiv_malawi_summary
```

```
## # A tibble: 4 × 5
##   period  new_positive previous_positive new_negative
##   <chr>          <dbl>             <dbl>        <dbl>
## 1 2019 Q1         6199             14816       284694
## 2 2019 Q2         6132             15101       282249
## 3 2019 Q3         5907             15799       300529
## 4 2019 Q4         5646             15700       291622
## # ℹ 1 more variable: previous_negative <dbl>
```

This aggregates the data nicely for passing to `{gt}` to generate a clean summary table.

To create a simple table from the aggregated data, we can then call the `gt()` function:

```
hiv_malawi_summary %>%
  gt()
```

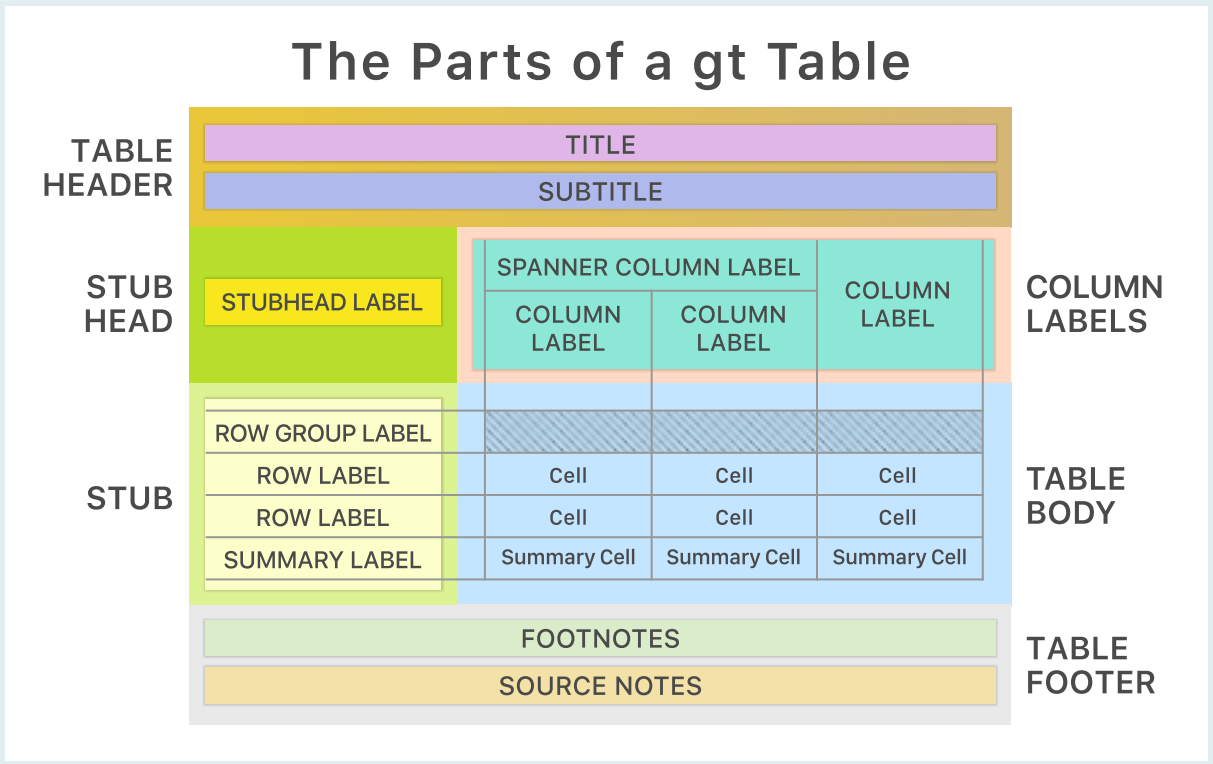
period	new_positive	previous_positive	new_negative	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

As you can see, the default table formatting is quite plain and unrefined. However, `{gt}` provides many options to customize and beautify the table output. We'll delve into these in the next section.

Customizing `{gt}` tables

The {gt} package allows full customization of tables through its "grammar of tables" framework. This is similar to how {ggplot2}'s grammar of graphics works for plotting.

To take full advantage of {gt}, it helps to understand some key components of its grammar.



As seen in the figure from the package website, The main components of a {gt} table are:

-**Table Header**: Contains an optional title and subtitle

-**Stub**: Row labels that identify each row

-**Stub Head**: Optional grouping and labels for stub rows

-**Column Labels**: Headers for each column

-**Table Body**: The main data cells of the table

-**Table Footer**: Optional footnotes and source notes

Understanding this anatomy allows us to systematically construct {gt} tables using its grammar.

Table Header and Footer

The basic table we had can now be updated with more components.

Tables become more informative and professional-looking with the addition of headers, source notes, and footnotes. We can easily enhance the basic table from before by adding these elements using {gt} functions.

To create a header, we use `tab_header()` and specify a `title` and `subtitle`. This gives the reader context about what the table shows.

```
hiv_malawi_summary %>%
  gt() %>%
  tab_header(
    title = "HIV Testing in Malawi",
    subtitle = "Q1 to Q4 2019"
  )
```

HIV Testing in Malawi

Q1 to Q4 2019

period	new_positive	previous_positive	new_negative	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

We can add a footer with the function `tab_source_note()` to cite where the data came from:

```
hiv_malawi_summary %>%
  gt() %>%
  tab_header(
    title = "HIV Testing in Malawi",
    subtitle = "Q1 to Q4 2019"
  ) %>%
  tab_source_note("Source: Malawi HIV Program")
```

HIV Testing in Malawi

Q1 to Q4 2019

period	new_positive	previous_positive	new_negative	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

Source: Malawi HIV Program

Footnotes are useful for providing further details about certain data points or labels The `tab_footnote()` function attaches footnotes to indicated table cells. For example, we can footnote the diagnosis columns:

```
hiv_malawi_summary %>%
  gt() %>%
  tab_header(
    title = "HIV Testing in Malawi",
    subtitle = "Q1 to Q2 2019"
  ) %>%
  tab_source_note("Source: Malawi HIV Program") %>%
  tab_footnote(
    footnote = "New diagnosis",
    locations = cells_column_labels(
      columns = c(new_positive, new_negative)
    )
  )
```

HIV Testing in Malawi

Q1 to Q2 2019

period	new_positive[1]	previous_positive	new_negative[1]	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

[1] New diagnosis

Source: Malawi HIV Program

These small additions greatly improve the professional appearance and informativeness of tables.

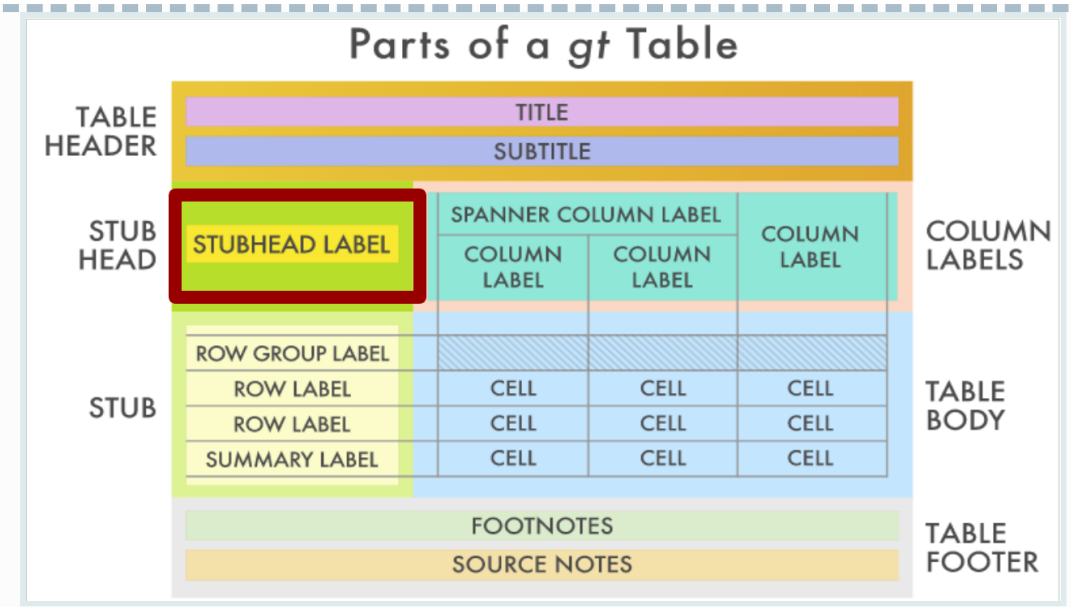
Stub

The stub is the left section of a table containing the row labels. These provide context for each row's data.

REMINDER

This image displays the stub component of a `{gt}` table, marked with a red square.

REMINDER



In our HIV case table, the `period` column holds the row labels we want to use. To generate a stub, we specify this column in `gt()` using the `rowname_col` argument:

```
hiv_malawi_summary %>%
  gt(rowname_col = "period") %>%
  tab_header(
    title = "HIV Testing in Malawi",
    subtitle = "Q1 to Q2 2019"
  ) %>%
  tab_source_note("Source: Malawi HIV Program")
```

HIV Testing in Malawi

Q1 to Q2 2019

	new_positive	previous_positive	new_negative	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

Source: Malawi HIV Program

Note that the column name passed to `rowname_col` should be in quotes.

For convenience, let's save the table to a variable `t1`:

```
t1 <- hiv_malawi_summary %>%
  gt(rowname_col = "period") %>%
  tab_header(
    title = "HIV Testing in Malawi",
    subtitle = "Q1 to Q2 2019"
  ) %>%
  tab_source_note("Source: Malawi HIV Program")

t1
```

<table>
<tr><th colspan="5">HIV Testing in Malawi</th></tr>
<tr><td colspan="5">Q1 to Q2 2019</td></tr>
<tr><td></td><td>new_positive</td><td>previous_positive</td><td>new_negative</td><td>previous_negative</td></tr>
<tr><td>2019 Q1</td><td>6199</td><td>14816</td><td>284694</td><td>6595</td></tr>
<tr><td>2019 Q2</td><td>6132</td><td>15101</td><td>282249</td><td>5605</td></tr>
<tr><td>2019 Q3</td><td>5907</td><td>15799</td><td>300529</td><td>6491</td></tr>
<tr><td>2019 Q4</td><td>5646</td><td>15700</td><td>291622</td><td>6293</td></tr>
<tr><td colspan="5">Source: Malawi HIV Program</td></tr>
</table>

Spanner columns & sub columns

To better structure our table, we can group related columns under “spanners”. Spanners are headings that span multiple columns, providing a higher-level categorical organization. We can do this with the `tab_spanner()` function.

Let’s create two spanner columns for new and Previous tests. We’ll start with the “New tests” spanner so you can observe the syntax:

```
t1 %>%
  tab_spanner(
    label = "New tests",
    columns = starts_with("new") # selects columns starting with "new"
  )
```

HIV Testing in Malawi Q1 to Q2 2019				
	New tests			
	new_positive	new_negative	previous_positive	previous_negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293
Source: Malawi HIV Program				

The `columns` argument lets us select the relevant columns, and the `label` argument takes in the span label.

Let's now add both spanners:

```
# Save table to t2 for easy access
t2 <- t1 %>%
  # First spanner for "New tests"
  tab_spanner(
    label = "New tests",
    columns = starts_with("new")
  ) %>%
  # Second spanner for "Previous tests"
  tab_spanner(
    label = "Previous tests",
    columns = starts_with("prev")
  )

t2
```

HIV Testing in Malawi Q1 to Q2 2019				
	New tests		Previous tests	
	new_positive	new_negative	previous_positive	previous_negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293
Source: Malawi HIV Program				

Note that the `tab_spanner` function automatically rearranged the columns in an appropriate way.

Question 1: The Purpose of Spanners

What is the purpose of using "spanner columns" in a `gt` table?

A. To apply custom CSS styles to specific columns.

B. To create group columns and increase readability.

C. To format the font size of all columns uniformly.

D. To sort the data in ascending order.

Question 2: Spanners Creation

Using the `hiv_malawi` data frame, create a `gt` table that displays a summary of the `sum` of "new_positive" and "previous_positive" cases for each region. Create spanner headers to label these two summary columns. To achieve this, fill in the missing parts of the code below:


```
region_summary <- hiv_malawi %>%
  group_by(region) %>%
  summarize(
    _________(
      c(new_positive, previous_positive),
      _______
    )
  )

# Create a gt table with spanner headers
summary_table_spanners <- region_summary %>%
  _______________ %>%
  ____________(
    label = "Positive cases",
    _________ = c(new_positive, previous_positive)
  )
```

Renaming Table Columns

The column names currently contain unneeded prefixes like “new_” and “previous_”. For better readability, we can rename these using `cols_label()`.

`cols_label()` takes a set of old names to match (on the left side of a tilde, `~`) and new names to replace them with (on the right side of the tilde). We can use `contains()` to select columns with “positive” or “negative”:

```
t3 <- t2 %>%
  cols_label(
    contains("positive") ~ "Positive",
    contains("negative") ~ "Negative"
  )

t3
```

HIV Testing in Malawi				
Q1 to Q2 2019				
	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293
Source: Malawi HIV Program				

This relabels the columns in a cleaner way.

`cols_label()` accepts several column selection helpers like `contains()`, `starts_with()`, `ends_with()` etc. These come from the tidyselect package and provide flexibility in renaming.

`cols_label()` has more identification function like `contains()` that work in a similar manner that are identical to the `tidyselect` helpers, these also include :

- `starts_with()`: Starts with an exact prefix.
- `ends_with()`: Ends with an exact suffix.
- `contains()`: Contains a literal string.
- `matches()`: Matches a regular expression.
- `num_range()`: Matches a numerical range like x01, x02, x03.

These helpers are useful especially in the case of multiple columns selection.

More on the `cols_label()` function can be found here: https://gt.rstudio.com/reference/cols_label.html

Question 3: column labels

Which function is used to change the labels or names of columns in a `gt` table?

A. `tab_header()`

B. `tab_style()`

C. `tab_options()`

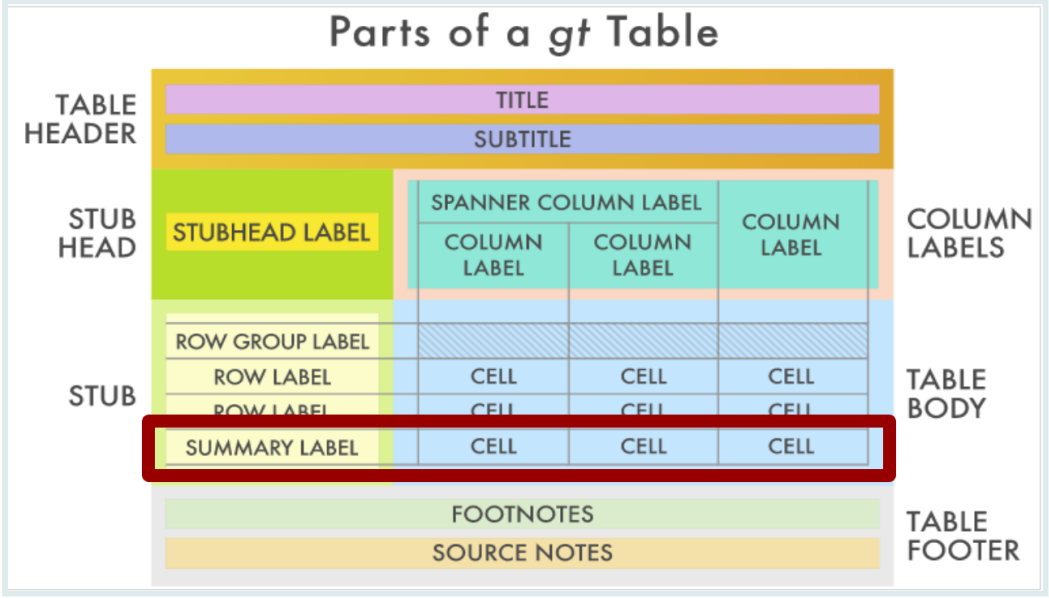
D. `tab_relabel()`

Summary rows

Let's take the same data we started with in the beginning of this lesson and instead of only grouping by period(quarters), let's group by both period and region. We will do this to illustrate the power of summarization features in `gt`: summary tables.

{gt} reminder - Summary Rows This image shows the summary rows component of a `{gt}` table, clearly indicated within a red square. Summary rows, provide aggregated data or statistical summaries of the data contained in the corresponding columns.

REMINDER



First let's recreate the data:

```
summary_data_2 <- hiv_malawi %>%
  group_by(
    # Note the order of the variables we group by.
    region,
    period
  ) %>%
  summarise(
    across(all_of(cols), sum)
    ) %>%
  gt()
```

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

```
summary_data_2
```

period	new_positive	previous_positive	new_negative	previous_negative
Central Region				
2019 Q1	2004	3682	123018	2562
2019 Q2	1913	3603	116443	1839
2019 Q3	1916	4002	127799	2645
2019 Q4	1691	3754	124728	1052
Northern Region				
2019 Q1	664	1197	36196	675
2019 Q2	582	1084	35315	590
2019 Q3	570	1191	36850	542
2019 Q4	519	1132	34322	346
Southern Region				
2019 Q1	3531	9937	125480	3358
2019 Q2	3637	10414	130491	3176
2019 Q3	3421	10606	135880	3304
2019 Q4	3436	10814	132572	4895

The order in the `group_by()` function affects the row groups in the `gt` table.

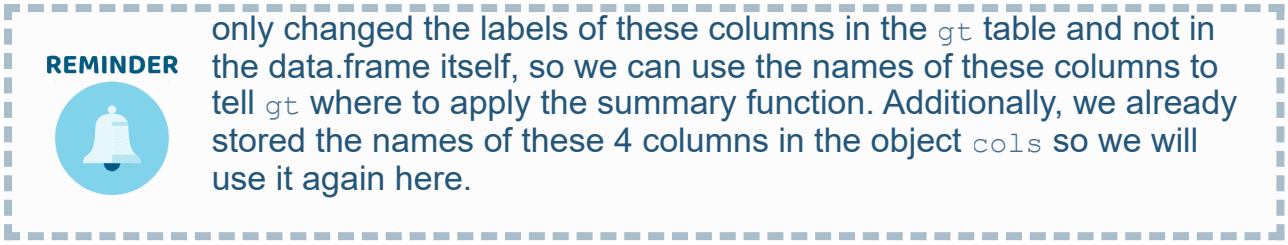
Second, let's re-incorporate all the changes we've done previously into this table:

```r
# saving the progress to the t4 object

t4 <- summary_data_2 %>%
  tab_header(
    title = "Sum of HIV Tests in Malawi",
    subtitle = "from Q1 2019 to Q4 2019"
  ) %>%
  tab_source_note("Data source: Malawi HIV Program") %>% tab_spanner(
    label = "New tests",
    columns = starts_with("new") # selects columns starting with "new"
  ) %>%
   # creating the first spanner for the Previous tests
  tab_spanner(
    label = "Previous tests",
    columns = starts_with("prev") # selects columns starting with "prev"
  ) %>%
  cols_label(
    # locate ### assign
    contains("positive") ~ "Positive",
    contains("negative") ~ "Negative"
  )

t4
```

Sum of HIV Tests in Malawi from Q1 2019 to Q4 2019				
	New tests		Previous tests	
period	Positive	Negative	Positive	Negative
Central Region				
2019 Q1	2004	123018	3682	2562
2019 Q2	1913	116443	3603	1839
2019 Q3	1916	127799	4002	2645
2019 Q4	1691	124728	3754	1052
Northern Region				
2019 Q1	664	36196	1197	675
2019 Q2	582	35315	1084	590
2019 Q3	570	36850	1191	542
2019 Q4	519	34322	1132	346
Southern Region				
2019 Q1	3531	125480	9937	3358
2019 Q2	3637	130491	10414	3176
2019 Q3	3421	135880	10606	3304
2019 Q4	3436	132572	10814	4895
Data source: Malawi HIV Program				

Now, what if we want to visualize on the table a summary of each variable for every region group? More precisely we want to see the sum and the mean for the 4 columns we have for each region.

REMINDER



In order to achieve this we will use the handy function `summary_rows` where we explicitly provide the columns that we want summarized, and the functions we want to summarize with, in our case it's `sum` and `mean`. Note that we assign the name of the new row(unquoted) a function name ("quoted").

```
t5 <- t4 %>%
  summary_rows(
    columns = cols, #using columns = 3:6 also works
    fns = list(
      TOTAL = "sum",
      AVERAGE = "mean"
    )
  )

t5
```

Sum of HIV Tests in Malawi

from Q1 2019 to Q4 2019

		New tests		Previous tests	
	period	Positive	Negative	Positive	Negative
Central Region					
	2019 Q1	2004	123018	3682	2562
	2019 Q2	1913	116443	3603	1839
	2019 Q3	1916	127799	4002	2645
	2019 Q4	1691	124728	3754	1052
sum	—	7524.00	491988.00	15041.00	8098.00
mean	—	1881.00	122997.00	3760.25	2024.50
Northern Region					
	2019 Q1	664	36196	1197	675
	2019 Q2	582	35315	1084	590
	2019 Q3	570	36850	1191	542
	2019 Q4	519	34322	1132	346
sum	—	2335.00	142683.00	4604.00	2153.00
mean	—	583.75	35670.75	1151.00	538.25
Southern Region					
	2019 Q1	3531	125480	9937	3358
	2019 Q2	3637	130491	10414	3176
	2019 Q3	3421	135880	10606	3304
	2019 Q4	3436	132572	10814	4895
sum	—	14025.00	524423.00	41771.00	14733.00

Sum of HIV Tests in Malawi				
from Q1 2019 to Q4 2019				
period	**New tests**		**Previous tests**	
	Positive	**Negative**	**Positive**	**Negative**
Data source: Malawi HIV Program				

Question 4 : summary rows

What is the correct answer (or answers) if you had to summarize the standard deviation of the rows of columns “new_positive” and “previous_negative” only?

A. Use `summary_rows()` with the `columns` argument set to “new_positive” and “previous_negative” and `fns` argument set to “sd”.

```
# Option A
your_data %>%
  summary_rows(
    columns = c("new_positive", "previous_negative"),
    fns = "sd"
  )
```

B. Use `summary_rows()` with the `columns` argument set to “new_positive” and “previous_negative” and `fns` argument set to “summarize(sd)”.

```
# Option B

your_data %>%
  summary_rows(
    columns = c("new_positive", "previous_negative"),
    fns = summarize(sd)
  )
```

C. Use `summary_rows()` with the `columns` argument set to “new_positive” and “previous_negative” and `fns` argument set to `list(SD = "sd")`.

```
# Option C
your_data %>%
  summary_rows(
    columns = c("new_positive", "previous_negative"),
    fns = list(SD = "sd")
  )
```

D. Use `summary_rows()` with the `columns` argument set to "new_positive" and "previous_negative" and `fns` argument set to "standard_deviation".

```
# Option D
your_data %>%
  summary_rows(
    columns = c("new_positive", "previous_negative"),
    fns = "standard_deviation"
  )
```

Wrap-up

In today's lesson, we got down to business with data tables in R using `gt`. We started by setting some clear goals, introduced the packages we'll be using, and met our dataset. Then, we got our hands dirty by creating straightforward tables. We learned how to organize our data neatly using spanner columns and tweaking column labels to make things crystal clear and coherent. Then wrapped up with some nifty table summaries. These are the nuts and bolts of table-making in R and `gt`, and they'll be super handy as we continue our journey on creating engaging and informative tables in R.

Answer Key

1. B
2.

```
# Solutions are where the numbered lines are

# summarize data first
district_summary <- hiv_malawi %>%
  group_by(region) %>%
  summarize(
    across(  #1
      c(new_positive, previous_positive),
      sum #2
    )
  )

# Create a gt table with spanner headers
summary_table_spanners <- district_summary %>%
  gt() %>% #3
  tab_spanner( #4
    label = "Positive cases",
    columns = c(new_positive, previous_positive) #5
  )
```

3. D
4. C

Contributors

The following team members contributed to this lesson:

BENNOUR HSIN

Data Science Education Officer
Data Visualization enthusiast

JOY VAZ

R Developer and Instructor, the GRAPH Network
Loves doing science and teaching science

References

1. Tom Mock, “The Definite Cookbook of {gt}” (2021), The Mockup Blog, https://themockup.blog/static/resources/gt-cookbook.html#introduction.

2. Tom Mock, “The Grammar of Tables” (May 16, 2020), The Mockup Blog, https://themockup.blog/posts/2020-05-16-gt-a-grammar-of-tables/#add-titles.

3. RStudio, “Introduction to Creating gt Tables,” Official {gt} Documentation, https://gt.rstudio.com/articles/intro-creating-gt-tables.html.

4. Fleming, Jessica A., Alister Munthali, Bagrey Ngwira, John Kadzandira, Monica Jamili-Phiri, Justin R. Ortiz, Philipp Lambach, et al. 2019. “Maternal Immunization in Malawi: A Mixed Methods Study of Community Perceptions, Programmatic Considerations, and Recommendations for Future Planning.” *Vaccine* 37 (32): 4568–75. https://doi.org/10.1016/j.vaccine.2019.06.020.