EPIDEMIOLOGICAL REPORTING WITH R

BEST PRACTICE FROM TABLES TO TIME SERIES

2023

This book is a compilation of training materials created by the GRAPH Network under a grant from the Global Fund to Fight AIDS, Tuberculosis and Malaria. The materials aim to build global capacity in epidemiological data analysis and decision-making for public health.

Demographic Pyramids for Epidemiological Analysis

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

Today you will learn about the importance of using demographic pyramids to help visualize the distribution of a disease by age and sex and how to create one using `ggplot2`.

Let's get into it!

Learning Objectives

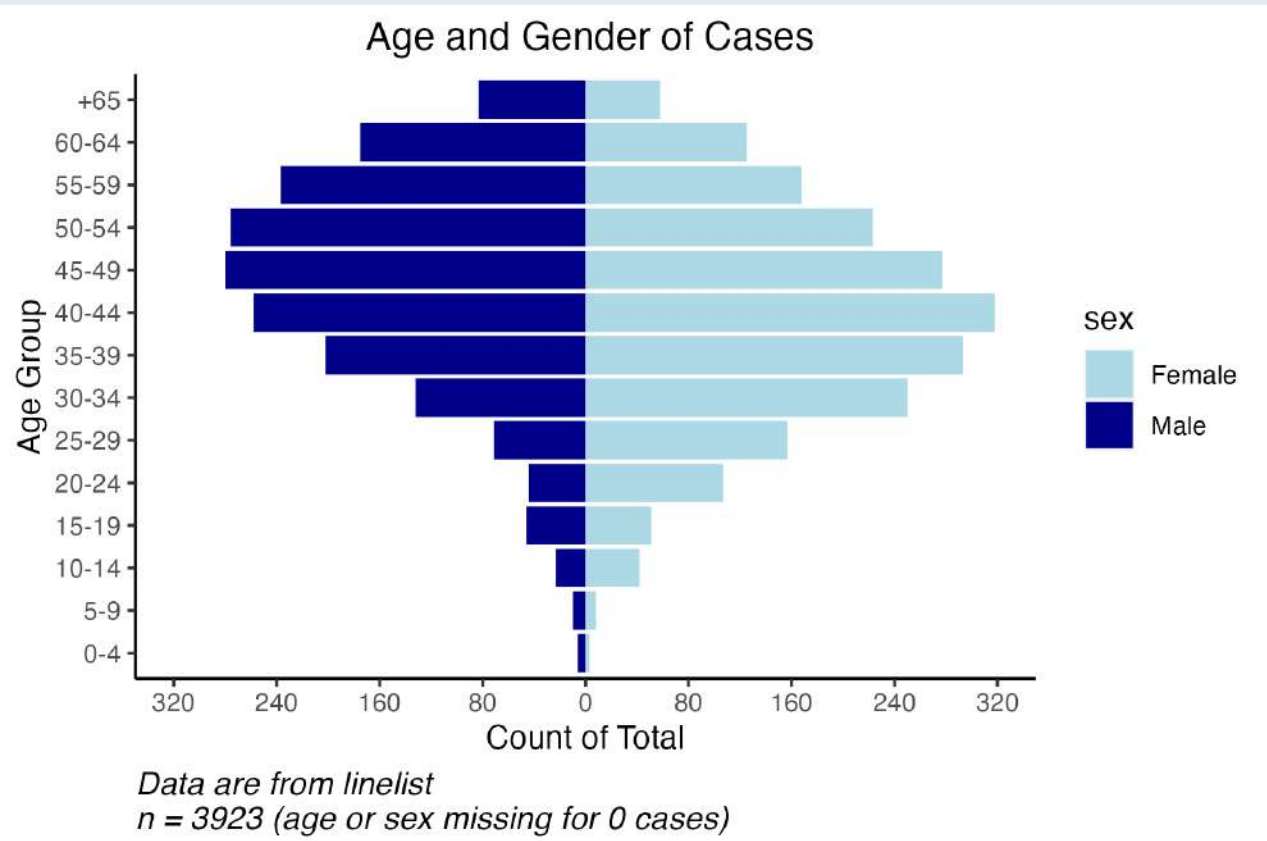
- You know the importance of using demographic pyramids to show the distribution of age groups and sex for communicable diseases.
- You can use `geom_col` from `ggplot2` to create a demographic pyramid showing the number or percent total of cases, deaths, and more.
- You can customize the plot by changing the color scheme, labels, and axis.

Introducing Demographic Pyramids

A demographic pyramid, also known as a population or age-sex pyramid, helps visualize the distribution of a population by two important demographic variables: **age** and **sex**. The overall population takes the shape of a pyramid, therefore taking its name.

Population pyramids are graphs that show the distribution of ages across a population divided down the center between male and female members of the population, where the y-axis shows the age groups and the x-axis the sex.

Through the use of `ggplot2` we are able to create pyramids while customizing it to our specific needs such as the plot below:



Multiple packages are available to facilitate data analysis and data visualization. In the case of demographic pyramids, the `apyramid` package can be a useful tool. Nonetheless, by using this package we are limited in the customization of our plot, making `ggplot2` a much versatile approach.

SIDE NOTE

The package `apyramid` is a product of the **R4Epis** project that allows for the rapid creation of population pyramids any many other useful

functions used in epidemiological reports.

Detailed information about the package can be read **here** or by entering `?age_pyramid` in your R console.

Using Population Pyramids in Epidemiology

To describe and understand the epidemiology of various communicable diseases, population pyramids provide useful information while facilitating the visualization of the distribution of disease by age and sex.

We know that the incidence of certain communicable diseases can vary with age. For the case of tuberculosis (TB), adolescents and young adults are primarily affected in the region of Africa. However, in countries where TB has gone from high to low incidence, such as the United States, TB is mainly a disease of older people, or the immunocompromised. Another disease that demonstrates age variation is malaria, where children under the age of 5 years old account for a high majority of deaths in the region of Africa.

Therefore, when describing the epidemiology of communicable diseases such as HIV, Malaria, and TB, it is important to observe the distribution of cases or deaths by age group and sex. This information helps inform national surveillance programs which age group is experiencing the highest burden and who to target for intervention.

Using Demographic Distribution for Data Quality Assessment

Demographic pyramids can also play a crucial role in assessing the data quality of routine surveillance systems by helping assess the internal and external consistency.

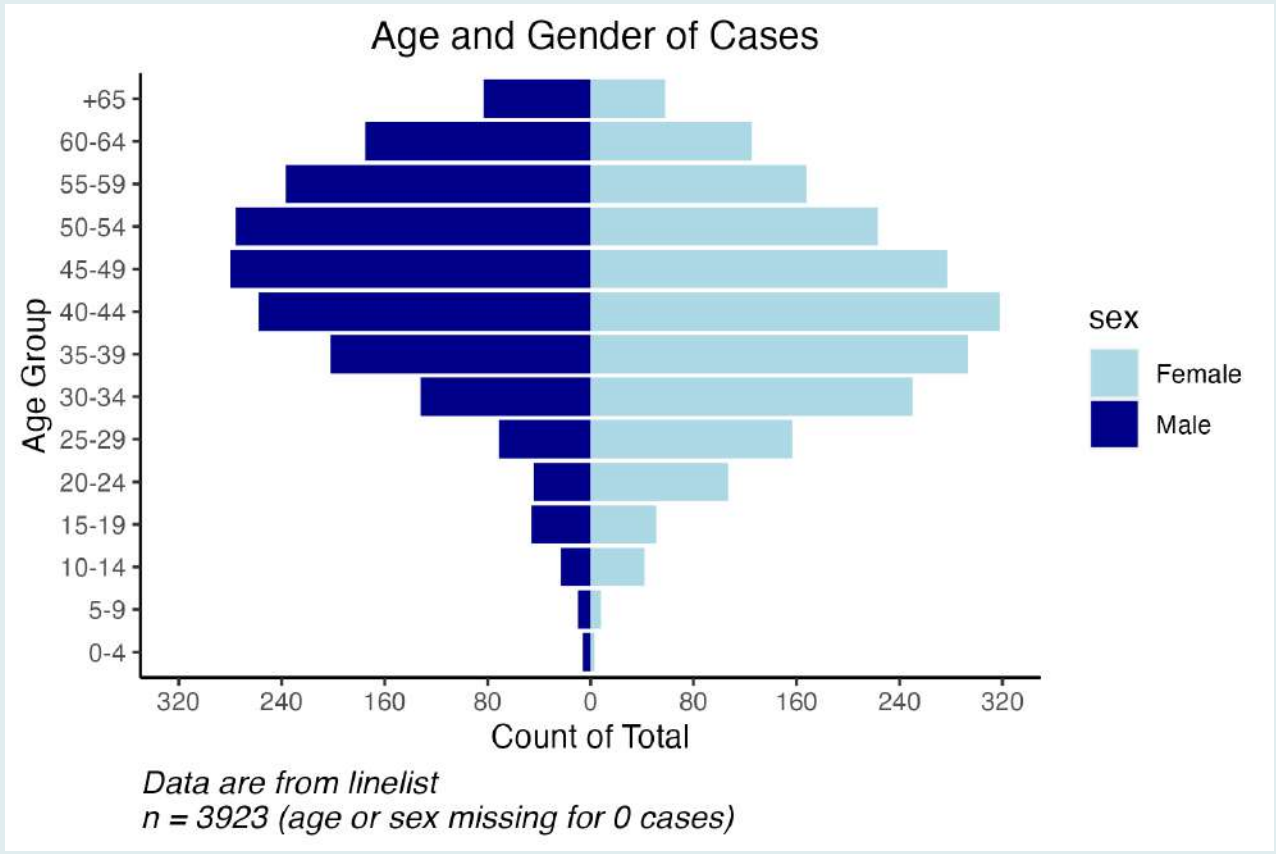
When trying to assess surveillance data quality standards of certain diseases, external consistency can be evaluated by comparing the national surveillance data with the global epidemiology of that disease. Data calculations based on demographic variables such as age group are sometimes used.

For the instance of TB surveillance data, external consistency can be evaluated by calculating the percentage of children diagnosed with TB within the program and comparing it with the global average cases.

Conceptualizing Demographic Pyramids

Let's take a closer look at a population pyramid and try to understand how it can be graphed using `geom_col` from `ggplot2`.

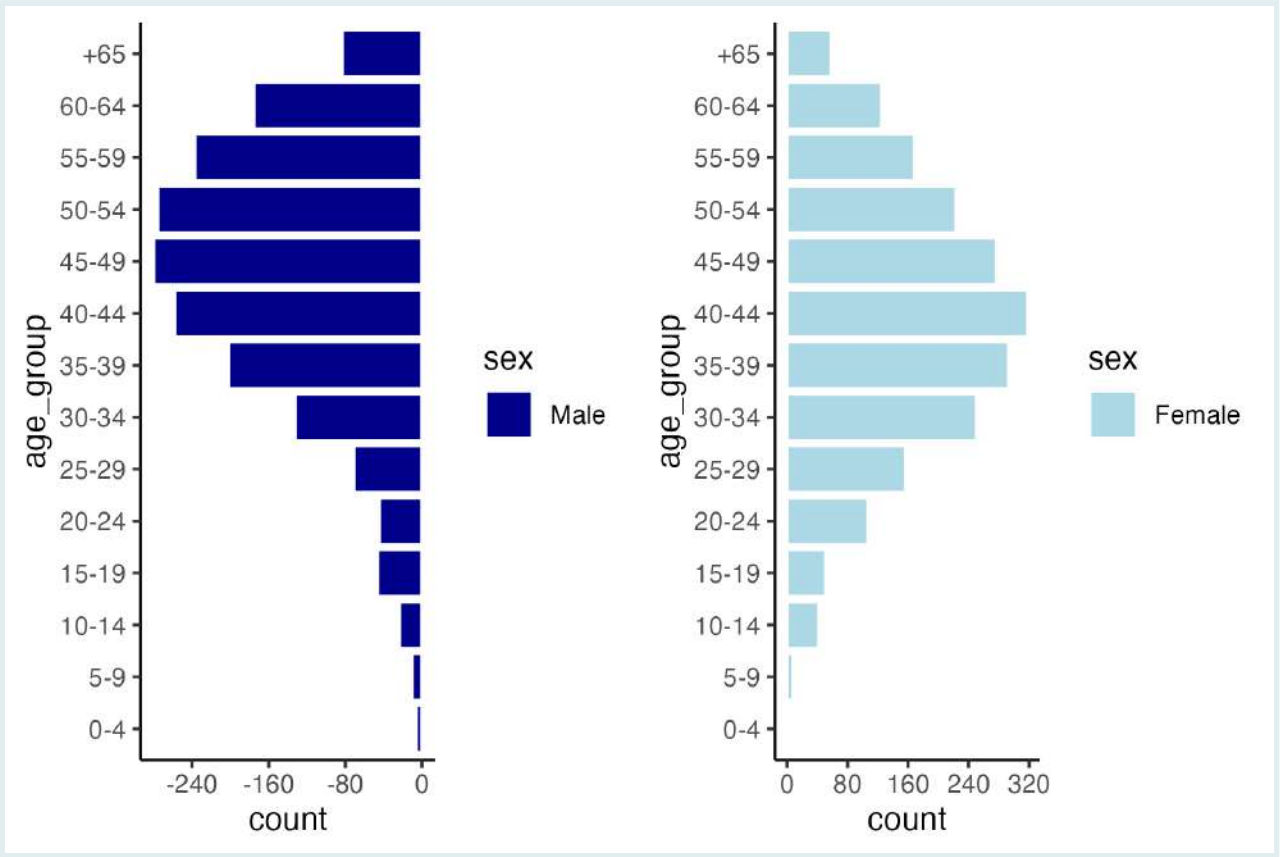
In order to conceptualize this, we will be using the dataset introduced later in the lesson. Let's take a closer look at the demographic pyramid we showed above:



As you can see from the image above, the x-axis is divided in two halves (males and females) where the units of the x-axis are symmetrical across the axis at point 0 and the age groups are labelled along the y-axis.

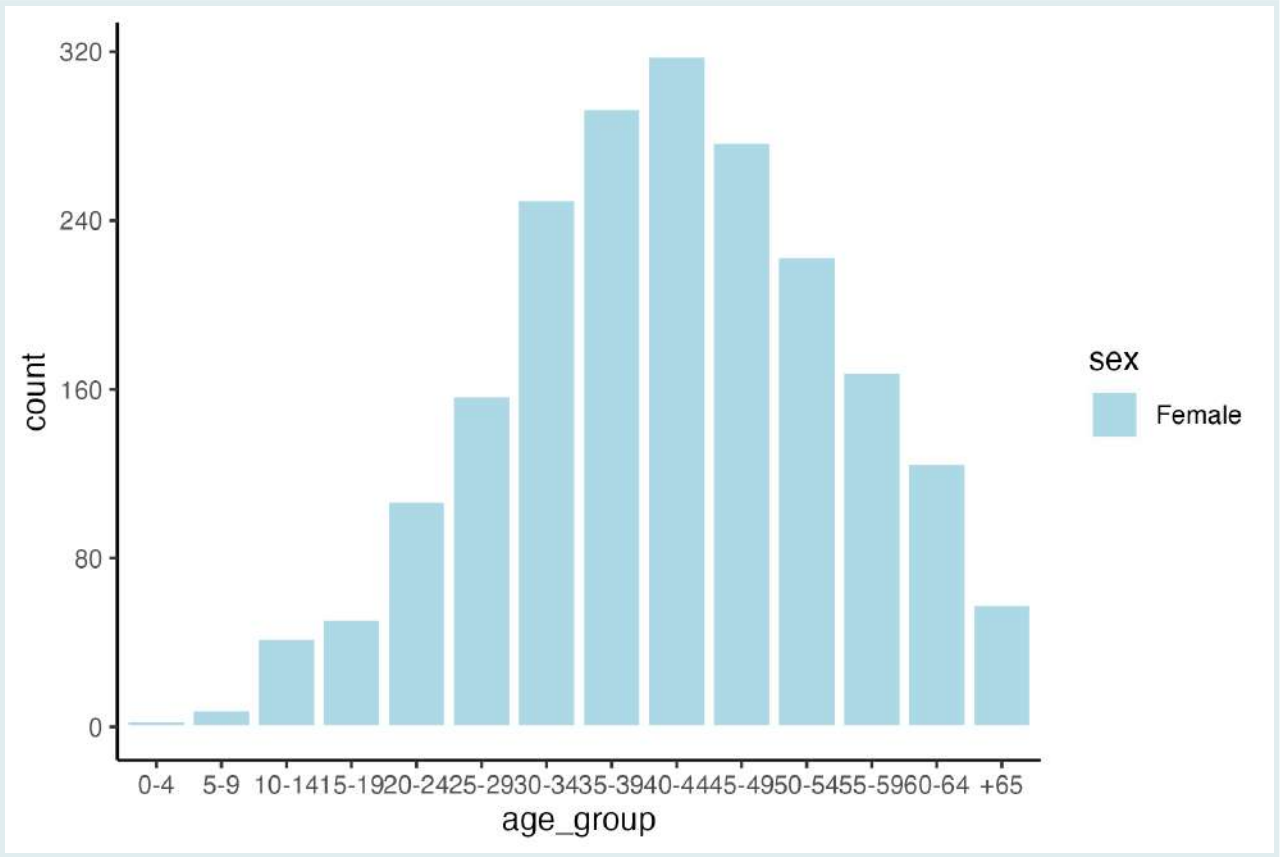
While closely looking at this graph, you probably noticed that the population pyramid consists of two plots merged together through the y-axis.

In other words, we can divide the pyramid into two sections (males or females) and graph them independently such as below:



Does this type of graph look familiar? What if we flip our x and y axes and turn the graph 90 degrees?

Let’s take the female half and flip our axes and see what we get:



As you can probably tell by now, in order to create a population pyramid we created two **bar graphs** that show the distribution of sex by age groups. Once we have created a plot for the females and the males, we then proceed to merged them together through their y-axis.

KEY POINT

In order for the male half to be graphed on the left of the y-axis, we will need to negate the **count** (total count of people per age group).

In other words, population pyramids are **bar graphs** whose axes are **flipped** (x and y axis flipped) and where females are graphed on the **positive side** of the y-axis and males are graphed on the **negative side** of the y-axis.

Packages

This lesson will require the following packages to be installed and loaded:

```r
# Load packages
if(!require(pacman)) install.packages("pacman")
pacman::p_load(here,        # to locate file
               tidyverse,   # to clean, handle, and plot the data (includes
        ggplot2 package)
               janitor,     # tables and cleaning data
               apyramid)    # package dedicated to creating age pyramids
```

```
## package 'apyramid' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\joych\AppData\Local\Temp\RtmpCKc7fE\downloaded_packages
```

Data Preparation

Intro to the Dataset

For this lesson, we will use a fictional HIV dataset imitating a linelist of HIV cases in Zimbabwe during 2016.

SIDE NOTE

You can access the source used to simulate the HIV data **here**

Each line (row) corresponds to one patient, while each column represents different variables of interest. The linelist only contains demographic and HIV-related variables (HIV status) .

For this specific lesson, we will focus on the **age-related** and **sex** variables to create our demographic pyramid.

Importing Data

Let's start by importing our data into our RStudio environment and taking a closer look at it to better understand the variables we will be using for the creation of our demographic pyramid.

SIDE NOTE

In order to focus our attention to the creation of demographic pyramids, we went ahead and created the data subset containing the variables of interest (`age_group` and `sex`).

```
hiv_data <- read_csv(here::here("data/clean/hiv_zimbabwe_2016.csv"))

hiv_data
```

```
## # A tibble: 10 × 3
##    age_group sex    hiv_status
##    <chr>     <chr>  <chr>
##  1 20-24     female positive
##  2 35-39     male   positive
##  3 15-19     female negative
##  4 40-44     male   negative
##  5 45-49     male   positive
##  6 35-39     female negative
##  7 50-54     male   positive
##  8 20-24     female positive
##  9 5-9       male   negative
## 10 60-64     male   negative
```

Our imported dataset contains **28000** rows and **3** columns containing the `age_group` and `sex` variables we will be using for the creation of our demographic pyramid. In addition, the `hiv_status` variable provides us with information on the status of patients (*positive* or *negative*).

Since we are interested on creating a demographic pyramid on HIV prevalence, we first need to filter for HIV positive patients and make sure the `age_group` and `hiv_status` variables are factorized.

Let's load an already cleaned subset of our data!

```
hiv_prevalence <-
        readRDS(here::here("data/clean/hiv_zimbabwe_prevalence.rds"))

hiv_prevalence
```

```
## # A tibble: 10 × 3
##    age_group sex    hiv_status
##    <fct>     <fct>  <fct>
##  1 20-24     female positive
##  2 35-39     male   positive
##  3 45-49     male   positive
##  4 50-54     male   positive
##  5 20-24     female positive
##  6 45-49     female positive
##  7 25-29     male   positive
##  8 35-39     female positive
##  9 60-64     male   positive
## 10 45-49     male   positive
```

SIDE NOTE

Notice that we now have data subset of **3923** rows and **3** columns where all of the patients are **HIV positive**!

Now, before moving to the creation of our demographic pyramid, let's inspect the data by creating a table summarizing the `age_group` and `sex` columns!

For this step, we will use `tabyl()` from **`janitor`**.

```
hiv_prevalence %>%
  tabyl(age_group, sex)
```

```
##  age_group female male
##        0-4      3    6
##        5-9      8   10
##      10-14     42   23
##      15-19     51   46
##      20-24    107   44
##      25-29    157   71
##      30-34    250  132
##      35-39    293  202
##      40-44    318  258
##      45-49    277  280
```

Based on the table, the data is clean while the `age_group` column is correctly organized in ascending order (youngest to oldest).

Before creating your demographic pyramid, make sure to check that your data is clean and correctly organized in **ascending order**! This is important when using categorical variables as the order of your `age_group` will affect the order it will be plotted in your pyramid.

In the case of demographic pyramids, we want the youngest age group to be located at the bottom of the y-axis and the oldest age group to be at the top of the y-axis.

Creating Aggregated Data Subset

Before we get started, we will need to create an aggregated data subset that aggregates the total occurrences per age group and sex as below:

```
# A tibble: 28 x 3
   age_group sex    count
   <fct>     <fct>  <dbl>
 1 0-4       female     3
 2 0-4       male      -6
 3 5-9       female     8
 4 5-9       male     -10
 5 10-14     female    42
 6 10-14     male     -23
 7 15-19     female    51
 8 15-19     male     -46
 9 20-24     female   107
10 20-24     male     -44
# i 18 more rows
```

Notice that the *male* values are negative in order to obtain the male bar plot on the *left side* of the graph!

It is important to understand how to use `geom_col`.

KEY POINT

When using the `geom_col` function, the count for each group **needs to be specified in the `aes` as the x or y variable**. In other words, you will need to use a dataset with the aggregated number of occurrences for each categorical level.

In the case of demographic pyramids, we will need to use a dataset with aggregated counts or percentages per age group and sex.

Let's start by calculating the total count and percentages per age group and sex and create a subset with this information where the female values are *positive* and the male values are *negative*!

REMINDER

Don't forget to negate the male **y value** in order to obtain the male bar plot on the *left side* of the graph!

```r
# Create new subset
pyramid_data <-
  hiv_prevalence %>%

# Count total number by age group and gender
  count(age_group,
        sex,
        name = "total") %>%
  ungroup() %>%

# Create new column for count of age_group by gender
  mutate(
    counts = case_when(
      sex == "female" ~ total,
      sex == "male" ~ -total, #convert male to negative
      TRUE ~ NA_real_),

# Create new column for percentage of age_group by gender
    percent = round(100*(total / sum(total, na.rm = T)), digits = 1),
         percent = case_when(
           sex == "female" ~ percent,
           sex == "male" ~ -percent, #convert male to negative
           TRUE ~ NA_real_)) #Make NA values numeric as well

pyramid_data
```

```
## # A tibble: 5 × 5
##   age_group sex    total counts percent
##   <fct>     <fct>  <int>  <dbl>   <dbl>
## 1 0-4       female     3      3     0.1
## 2 0-4       male       6     -6    -0.2
## 3 5-9       female     8      8     0.2
## 4 5-9       male      10    -10    -0.3
## 5 10-14     female    42     42     1.1
```

SIDE NOTE

Notice that the male values in the `counts` and `percent` columns are not *negative*!

Let's test your understanding with the following multiple-choice questions (Answer Key is located at the end):

1. **When creating the male bar plot, what modification is made to the occurrence values?**

a. They are added to the x-axis values.
b. They are multiplied by 2.

c. They are divided by 2.
d. They are negated (multiplied by -1).

Plot Creation

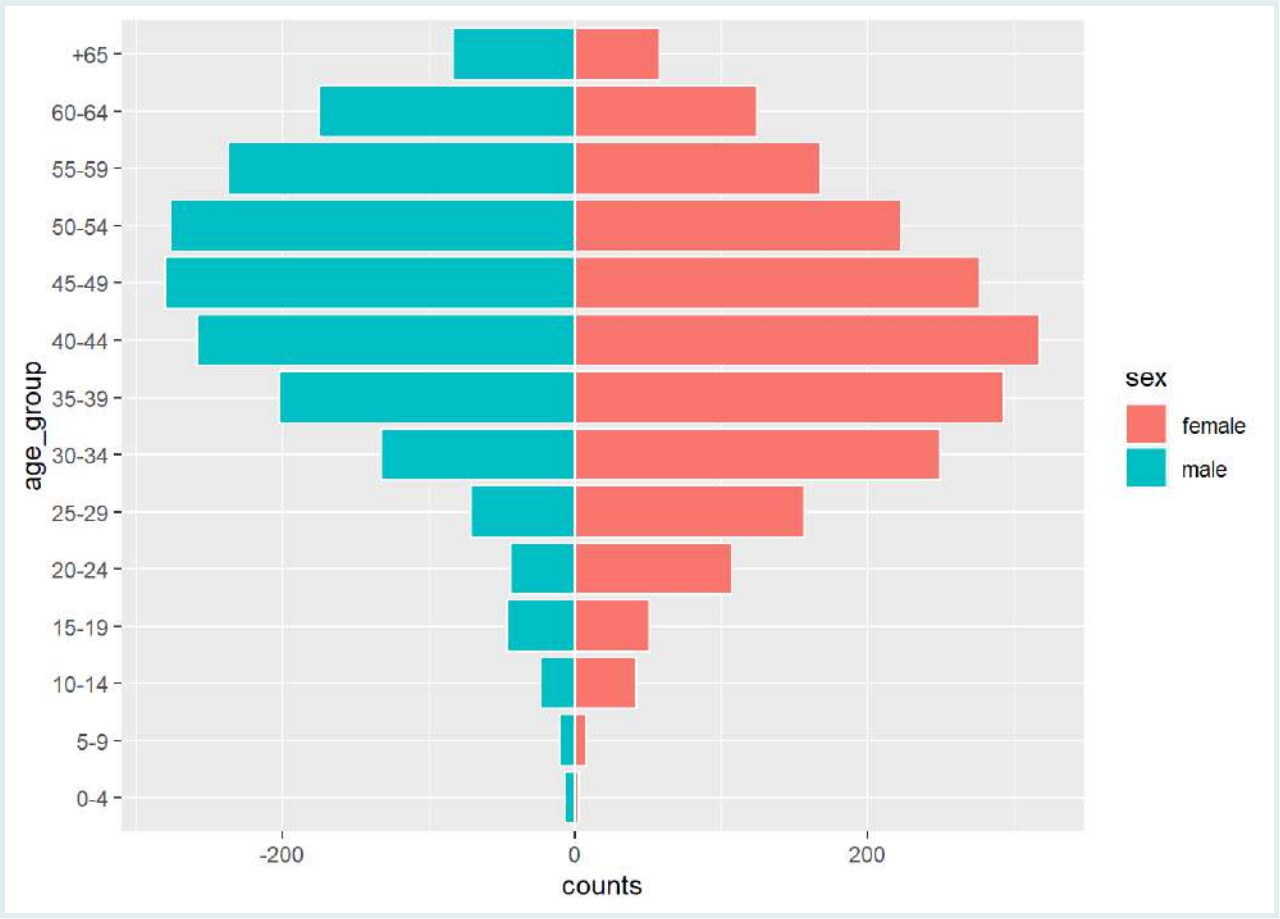
Using `geom_col`

Since we are going to create the demographic pyramid by plotting of a **bar graph**, we will need to use a **categorical variable**. Therefore, we will be using the categorical variable `age_group` to graph our bar plot and and 'fill' by `sex`.

In order to use the `geom_col` function, your dataset needs to include the aggregated count or percentage for each group (**age group** and **sex**)!

Let's create our demographic pyramid using the `geom_col` functions from `ggplot2`.

```
demo_pyramid <-

# Begin ggplot
  ggplot() +

# Create bar graph using geom_bar
  geom_col(data = pyramid_data, #specify data to graph
           mapping = aes(
             x = age_group,     #indicate x variable
             y = counts,        #indicate y variable
             fill = sex),       #fill by sex
           colour = "white") +

# Flip X and Y axes
  coord_flip()

demo_pyramid
```

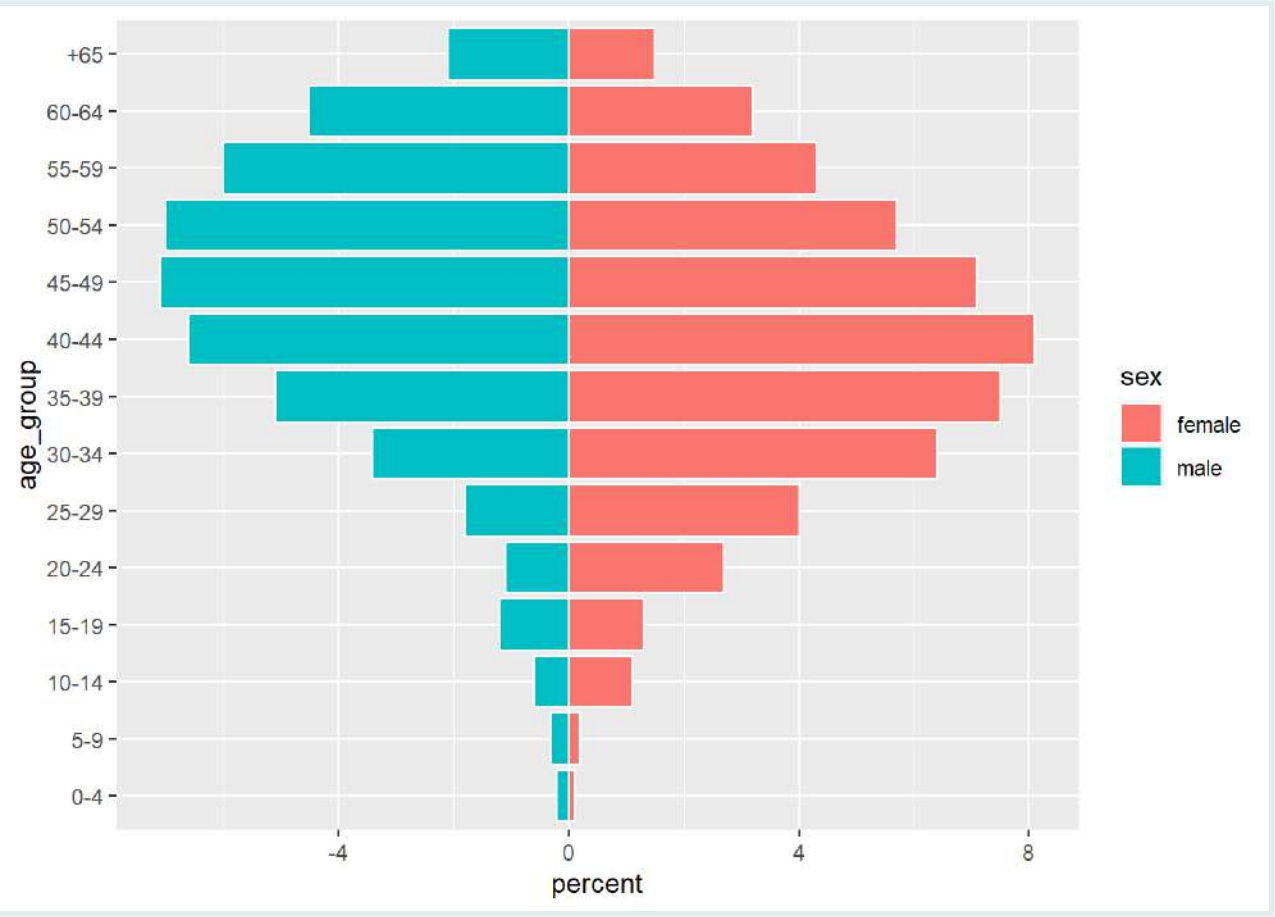


Did you notice that we plotted the data and mapping information of our graph in `geom_col()` function instead of `ggplot()` function?

By plotting it directly in the `geom_col()` function, we are able to add different layers to our graph (such additional bar plots, lines, points, and more) without affecting the outcome of our already created and individual bar plot.

We can also create the same population pyramid using the percent total as our y-axis value.

```r
demo_pyramid_percent <-

# Begin ggplot
  ggplot() +

# Create bar graph using geom_bar
  geom_col(data = pyramid_data,
           mapping = aes(
             x = age_group,
             y = percent,
             fill = sex),
           colour = "white") +

# Flip x and y axes
  coord_flip()

demo_pyramid_percent
```

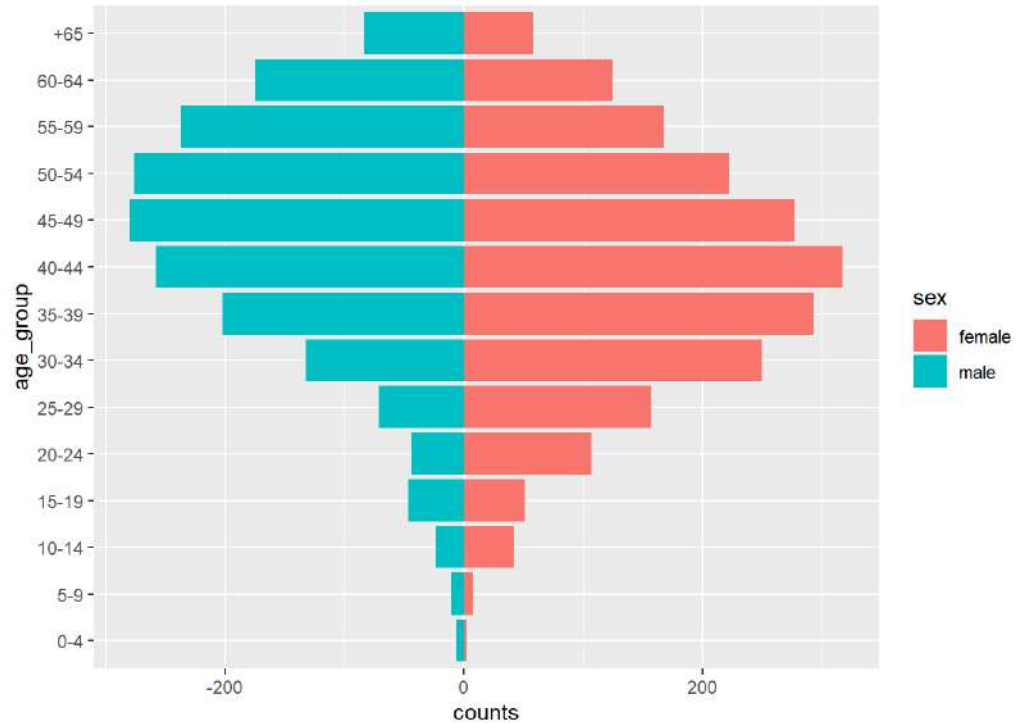


SIDE NOTE

If your dataset contains the count/percentage for each group, such as dataset `pyramid_data`, you can also use `geom_bar`. However, you will need to pass `stat = "identity"` inside `geom_bar` as following:

```
# Begin ggplot
ggplot(data = pyramid_data,
       mapping = aes(
         x = age_group,
         y = counts,     # Specify y-axis
         fill = sex),
       colour = "white") +

# Include stat = "identity" in gemo_bar
  geom_bar(stat = "identity") +

# Flip x and y axes
  coord_flip()
```

SIDE NOTE



Let's test your understanding with the following multiple-choice questions (Answer Key is located at the end):

2. **When using `geom_col`, what type of x variable should your dataset include?**
   a. Continuous variables
   b. Categorical variables
   c. Binary variables
   d. Ordinal variables
3. **Which `ggplot2` function can you use to flip the x and y axes?**
   a. `coord_flip()`
   b. `x_y_flip()`
   c. `geom_histogram_flip()`

d. All of the above

4. **Inside which `ggplot2` function should you pass `stat = "identity"` when using already aggregated data and trying to create a bar plot?**
   a. `geom_col`
   b. `geom_histogram`
   c. `geom_bar`
   d. All of the above

Now let's test your understanding with the following coding practice question (Answer Key is located at the end):

We will be using a cleaned and prepared dataset containing the total population of Zimbabwe in 2016 grouped by age group and sex.

Start by loading the dataset as following:

```
zw_2016 <- readRDS(here::here("data/clean/population_zw_2016.rds"))

zw_2016
```

```
## # A tibble: 28 × 3
## # Groups:   age_group [14]
##    age_group sex    total_count
##    <fct>     <fct>        <int>
##  1 0-4       Female     1091129
##  2 0-4       Male      -1114324
##  3 10-14     Female      815362
##  4 10-14     Male       -803657
##  5 15-19     Female      803754
##  6 15-19     Male       -792474
##  7 20-24     Female      690940
##  8 20-24     Male       -632342
##  9 25-29     Female      638192
## 10 25-29     Male       -535444
## # ℹ 18 more rows
```

*Note that the male total count is already **negated**!*

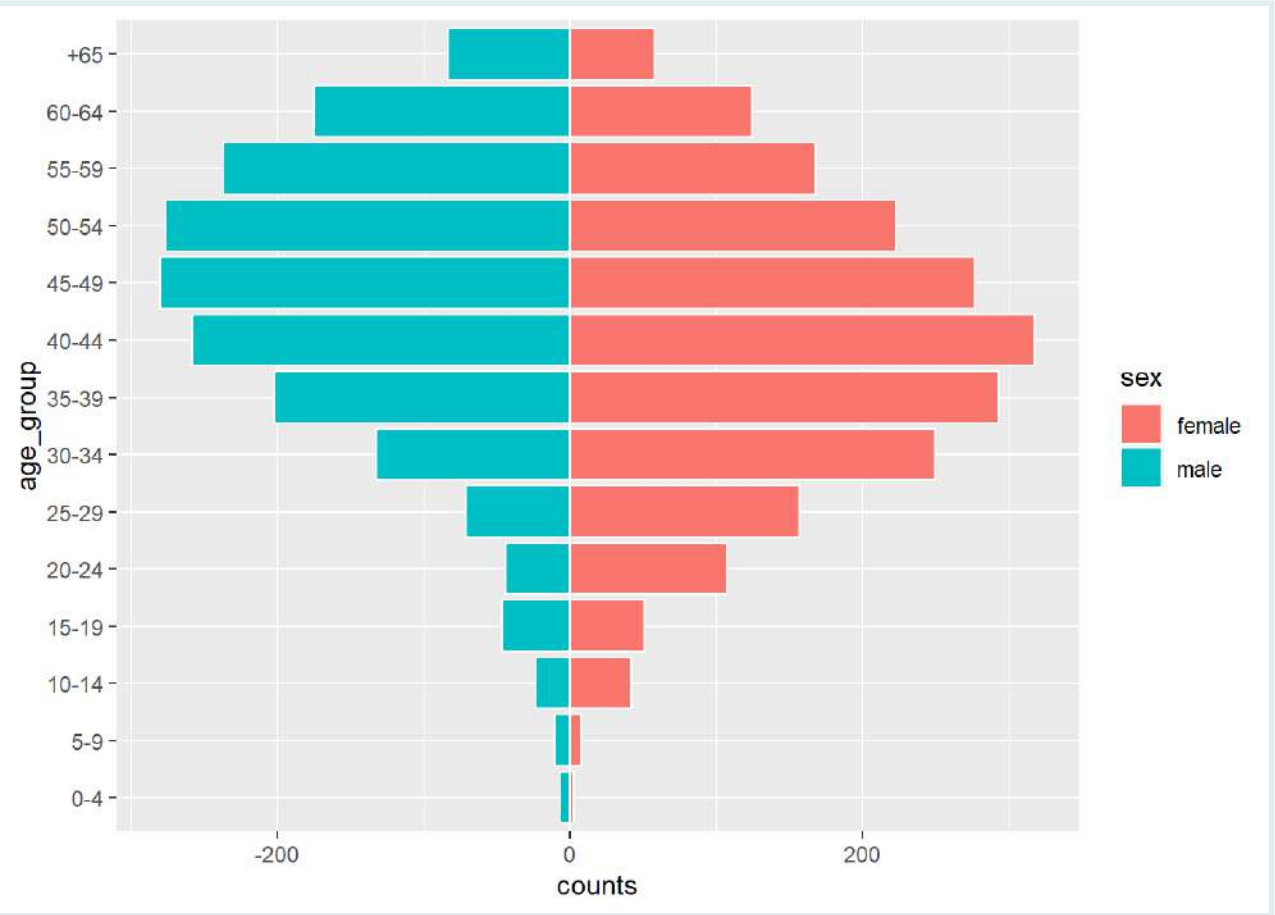
5. **Create a demographic pyramid for the total population of Zimbabwe in 2016 using the `geom_col` function from the `ggplot2` package. Make sure to add a white border around each bar!**

```
Q4_pyramid_zw_2016 <-

# Begin ggplot
  ggplot() +

# Create bar graph using geom_bar
  geom_col(data = ____,
           aes(x = ____,
               y = ____,
               fill = ____),
           color = ____) +

# Flip x and y axes
  ____
```

Plot Customization

So far, you have learned how to create a demographic pyramid using `ggplot2` as shown below:

```
demo_pyramid
```



However, in order to create an informative graph, a certain level of plot customization is needed. For instance, it is important to include informative labels and to re-scale the x and y axis for better visualization.

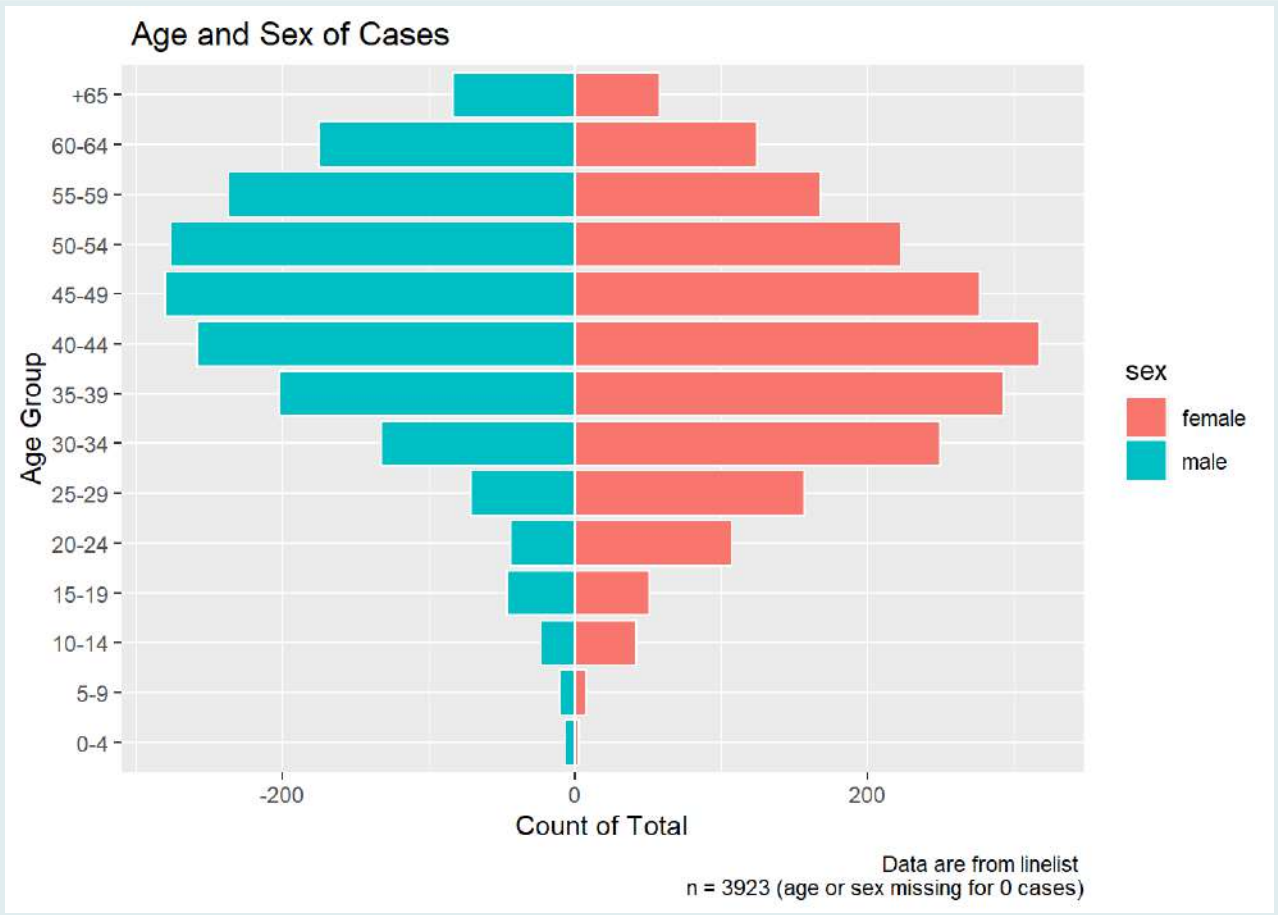
Let's learn some useful `ggplot2` customization!

Labels

Let's use the population pyramid we previously created using the `geom_col()` function and build upon it.

We can start by adding an informative title, axes, and caption to our graph:

```
adjusted_labels <-

# Use previous demographic pyramid
  demo_pyramid +

# Adjust the labels
  labs(
    title = " Age and Sex of Cases",
    x = "Age Group",
    y = "Count of Total",
    caption = stringr::str_glue("Data are from linelist \nn =
         {nrow(hiv_prevalence)} (age or sex missing for
         {sum(is.na(hiv_prevalence$sex) | is.na(hiv_prevalence$age_years))}
         cases)"))

adjusted_labels
```



Axis

Let's re-scale our axes to make sure the data is properly visualized and understood.

We will start by re-scaling the *total count* axis, or in the case of our plot, the **y-axis**. For this, we will start by identifying the maximum and minimum values and saving them as objects.

```
max_count <- max(pyramid_data$counts, na.rm = T)
min_count <- min(pyramid_data$counts, na.rm = T)

max_count
```

```
## [1] 318
```

```
min_count
```

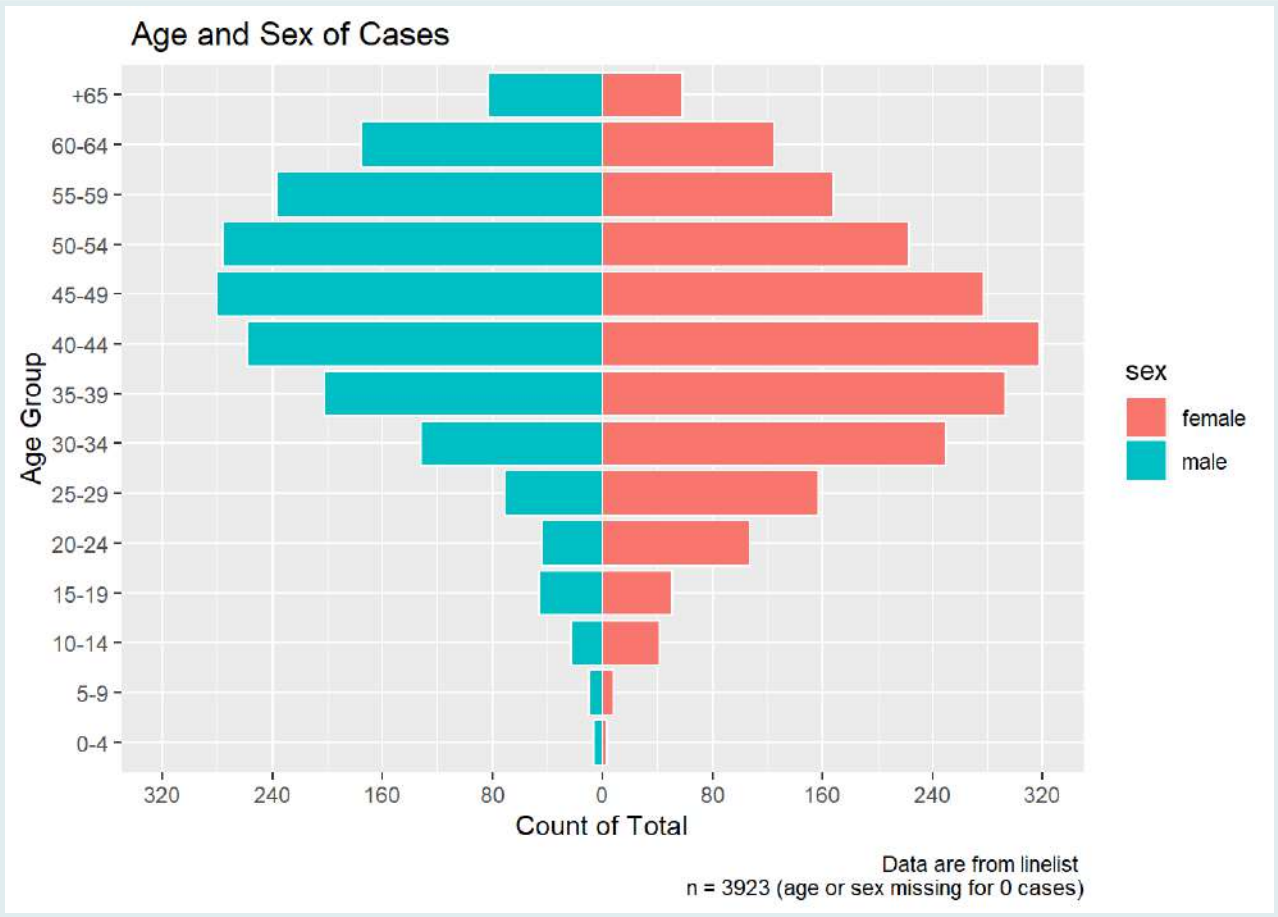
```
## [1] -280
```

Now that we have identified that the minimum value for the *total count* is **-280** and the maximum value is **318**, we can use it to re-scale our y-axis accordingly.

In this particular case, we want to rescale our y axis to be symmetrical. Therefore we will take the biggest absolute value and use it as our limit for both the *positive* and *negative* sides.

In this case, we will use our maximum value.

```
adjusted_axes <-

# Use previous graph
  adjusted_labels +

# Adjust y-axis (total count)
  scale_y_continuous(

    # Specify limit of y-axis using max value and making positive and negative
    limits = c(max_count * c(-1,1)),

    # Specify how to break the y-axis (based on max limit)
    breaks = seq(-400, 400, 400/5),

    # Make axis labels absolute so male labels appear positive
    labels = abs)


adjusted_axes
```

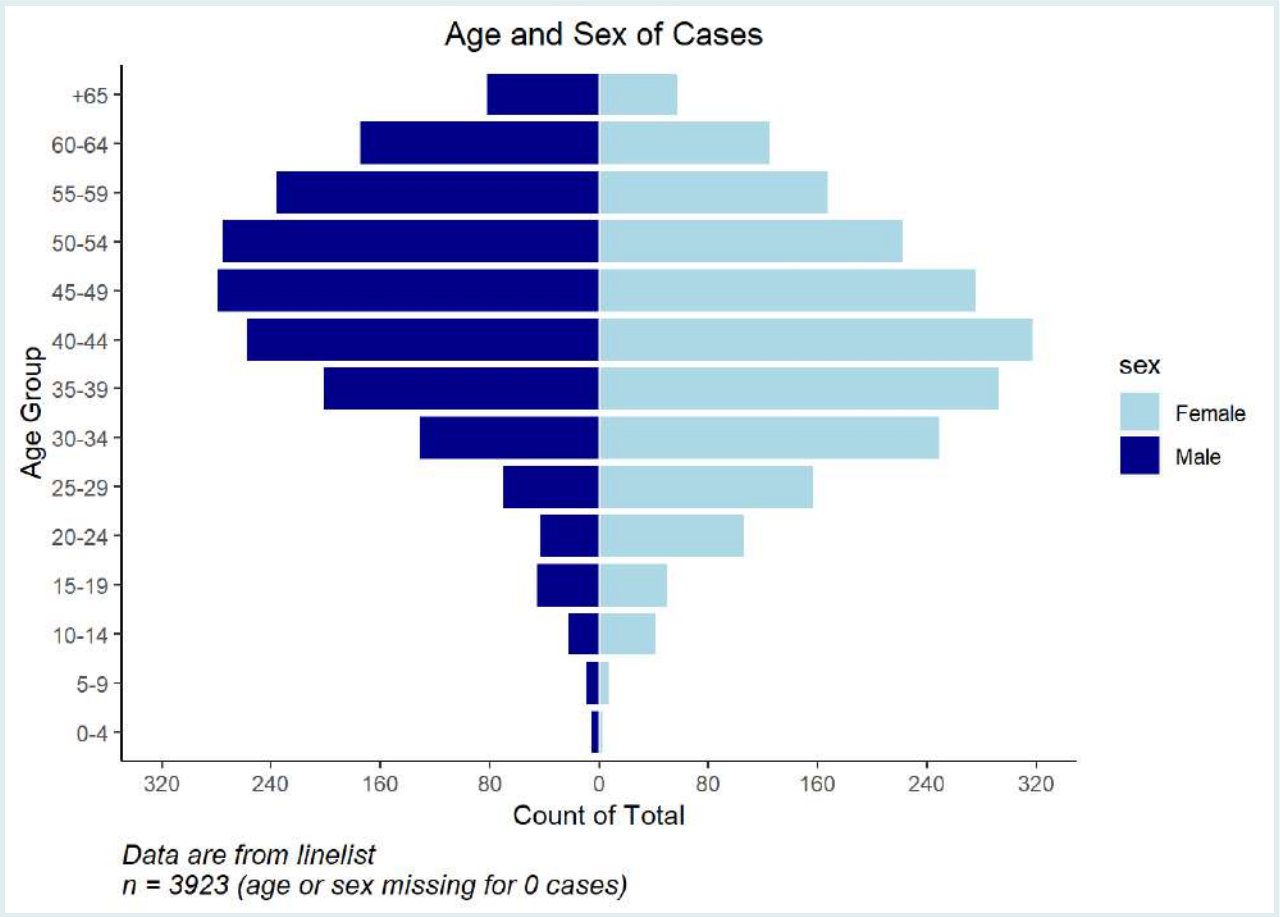


Color Scheme and Themes

We can also make necessary adjustments to the color scheme and theme of the graph.

Below is an example of some changes, we can perform:

```r
adjusted_color_theme <-

# Use previous graph
  adjusted_axes +

# Designate colors and legend labels manually
  scale_fill_manual(

    # Select color of sex fill
    values = c("female" = "lightblue",
               "male" = "darkblue"),

    # Rename legend labels
    labels = c("Female",
               "Male")) +

# Adjust theme settings
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank(),
    axis.line = element_line(colour = "black"), #make axis line black
    plot.title = element_text(hjust = 0.5),      #set high of title
    plot.caption = element_text(hjust = 0, size = 11, face = "italic"))
         #format caption

adjusted_color_theme
```



WRAP UP!

As you can see, demographic pyramids are an essential visualization tool to understand the distribution of specific diseases across age groups and sex.

The concepts learned in this lesson can also be applied to create other types of graphs that require both negative and positive outputs such as percentage change in case notification rates and more.

Now that you have learned the concept behind the creation of demographic pyramids, the possibilities are endless! From plotting the **cases** per age group and sex over the **baseline/true** population to graphing the change (positive and negative) of interventions in a population, you should be able to apply these concepts to create informative epidemiological graphs.

Congratulations on finishing this lesson. We hope you can now apply the knowledge learned in today's lesson during the analysis and creation of epidemiological review reports.

Answer Key

1. d
2. b
3. a
4. c
5.

```
Q4_pyramid_zw_2016 <-
  ggplot() +
  geom_col(data = zw_2016,
           aes(x = age_group,
               y = total_count,
               fill = sex),
           color = "white") +
  coord_flip()
```

Contributors

The following team members contributed to this lesson:

SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network
Infectiously enthusiastic about microbes and Global Health

JOY VAZ

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

References

1. Adjusted lesson content from: Batra, Neale, et al. The Epidemiologist R Handbook. 2021. https://doi.org/10.5281/zenodo.4752646
2. Adjusted lesson content from: WHO. Understanding and Using Tuberculosis Data. 2014. https://apps.who.int/iris/bitstream/handle/10665/129942/9789241548786_eng.pdf
3. Referenced package from: https://r4epis.netlify.app/

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:

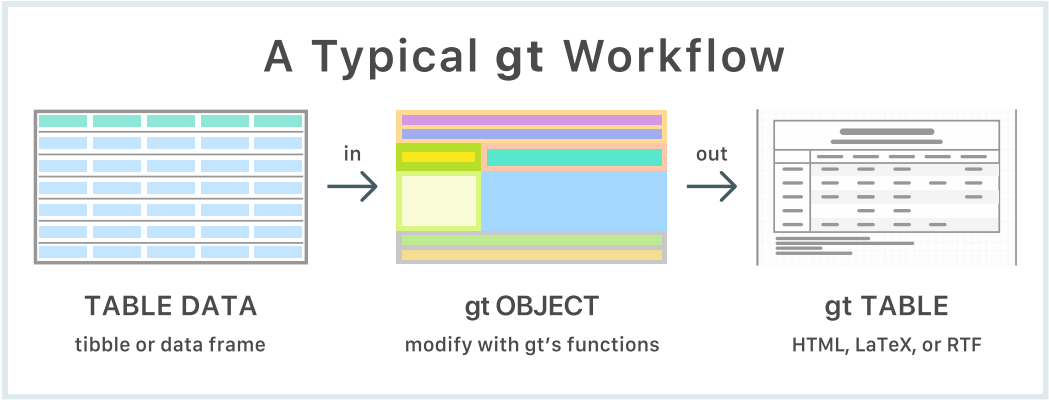
1. previous_negative: The number of patients who visited the healthcare facility in that quarter that had prior negative HIV tests.
2. previous_positive: The number of patients (as above) with prior positive HIV tests.
3. new_negative: The number of patients newly testing negative for HIV.
4. new_positive: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



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()
```

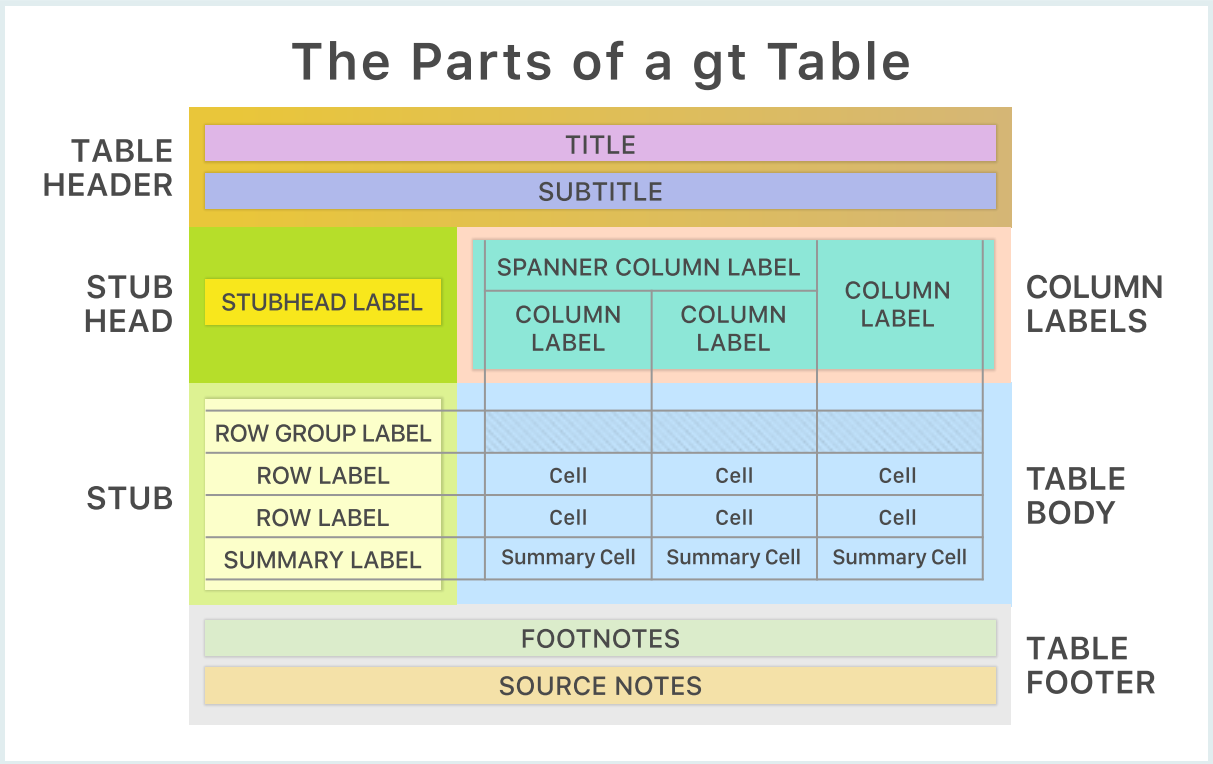
period	new_positive	previous_positive	new_negative	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

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.

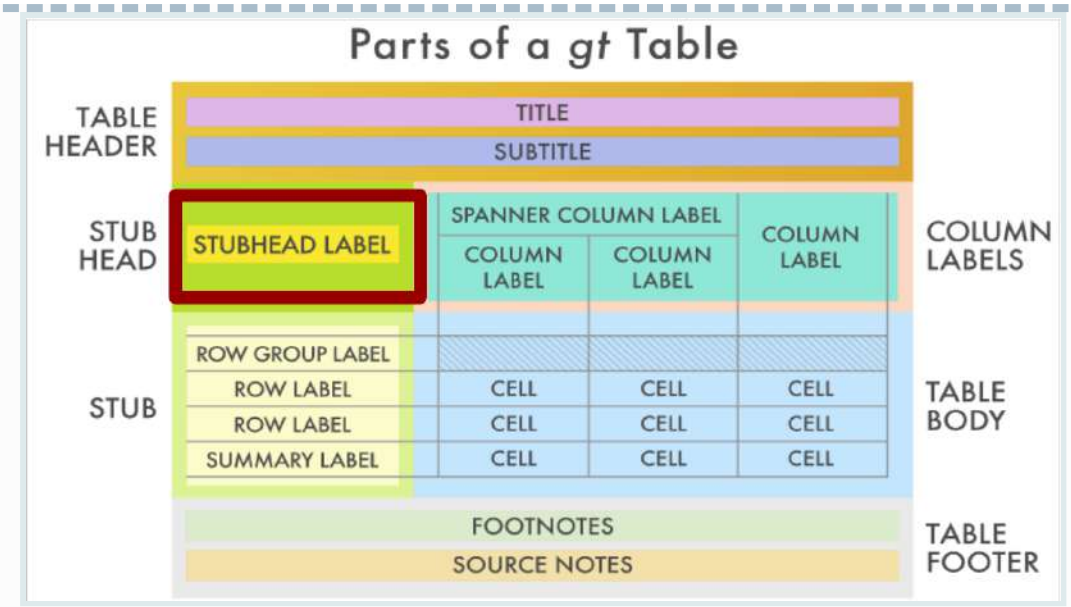
Stub

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
```

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				

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()`

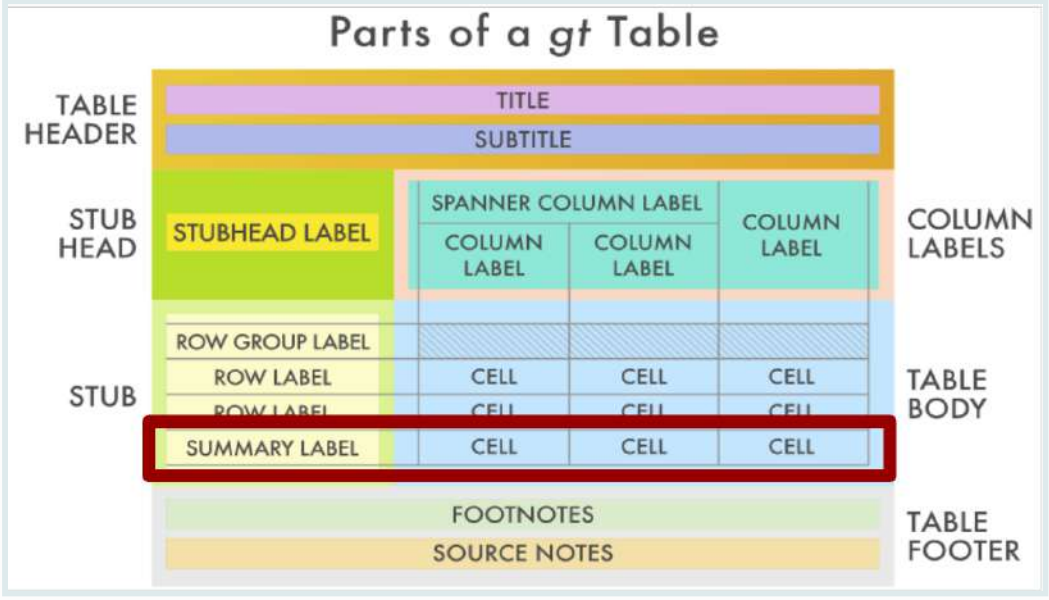
D. `tab_relabel()`

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

only changed the labels of these columns in the `gt` table and not in the data.frame itself, so we can use the names of these columns to tell `gt` where to apply the summary function. Additionally, we already stored the names of these 4 columns in the object `cols` so we will use it again here.

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.

Optimizing gt Tables for Enhanced Visualization

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

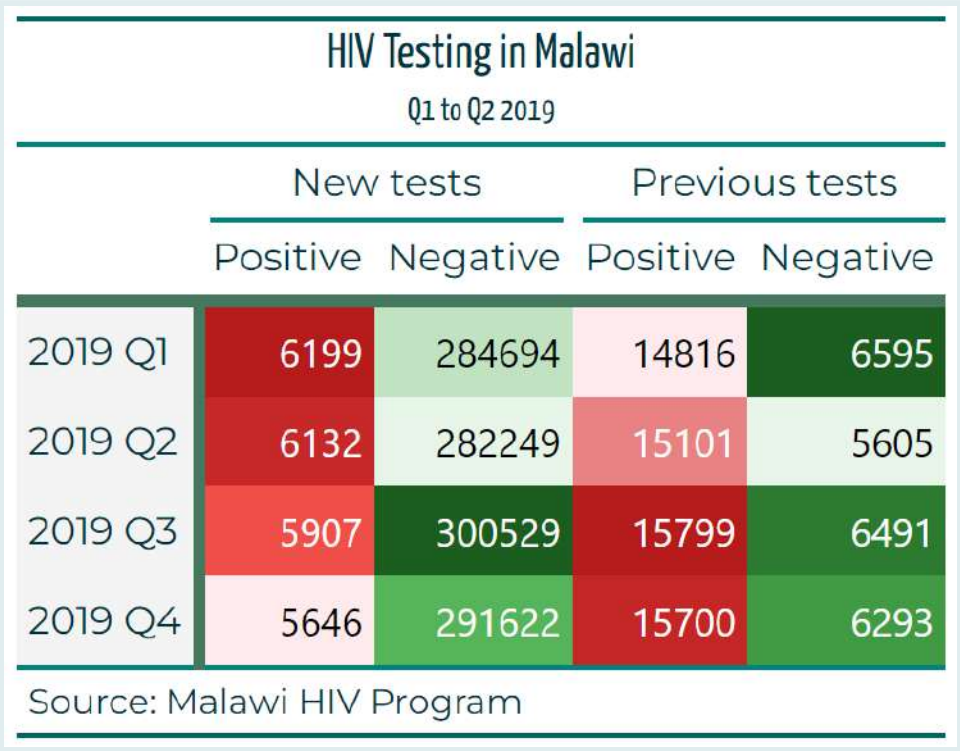
The previous `gt` lesson focused mainly on the components of the table its structure and how to manipulate it properly. This lesson, presenting the second part of the `gt` series will focus on using the package to polish, style, and customize the visual effects of tables in a way that elevate the quality and efficiency of your reports.

Let's dig in.

Learning objectives

- Cells Formatting
- Conditional coloring
- Format text(font color, bold,etc.)
- Add borders to text

By the conclusion of this lesson, you will have the skills to artfully style your **`gt`** tables to meet your specific preferences achieving a level of detail similar to this:



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

Packages

In this lesson, we will use the following packages:

- `gt`
- `dplyr` , `tidyr` , and `purrr`.
- `janitor`
- `KableExtra`
- `Paletteer` , `ggsci`

```
pacman::p_load(tidyverse, janitor, gt, here, paletteer, webshot2)
```

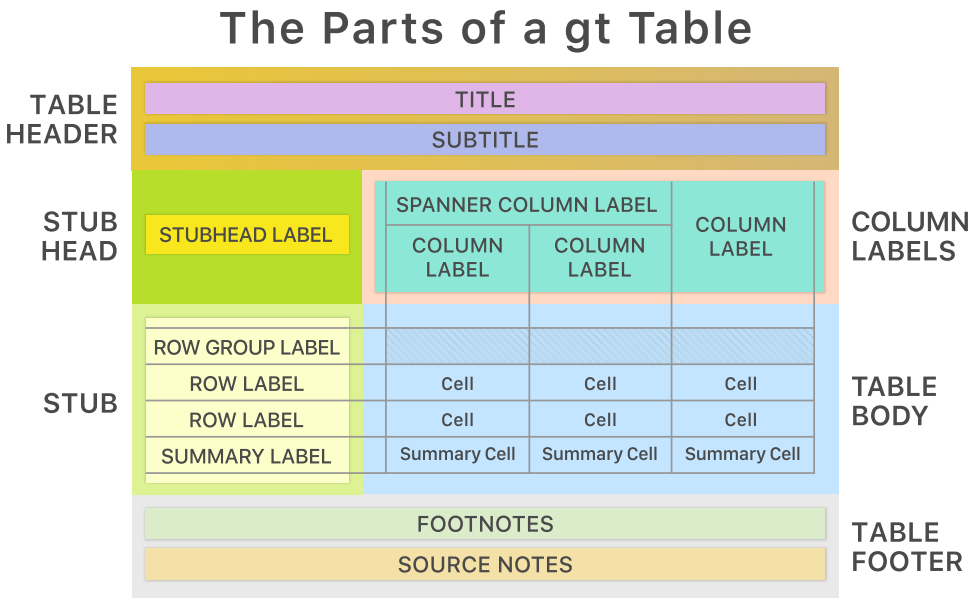
Previously in pt1

RECAP

In the previous `gt` lesson we had the opportunity to :

- Discover the HIV prevalence data of Malawi.
- Discover the grammar of tables and the `gt` package.
- create simple table.
- Add details like title and footnote to the table.
- Group columns into spanners.
- Create Summary rows.

RECAP



Dataset

In this lesson, we will use the same data from the previous lesson, you can go back for a detailed description of the data and the preparation process we made.

RECAP

Here's the full details of the columns we will use:

RECAP

- region: The geographical region or area where the data was collected or is being analyzed.
- period: A specific time period associated with the data, often used for temporal analysis.
- previous_negative: The count or number of individuals with a previous negative test result.
- previous_positive: The count or number of individuals with a previous positive test result.
- new_negative: The count or number of newly diagnosed cases with a negative result.
- new_positive: The count or number of newly diagnosed cases with a positive result.

But for the purposes of this lesson we will use the tables directly, this this the table that we created with the right spanners and columns labels, we will base the rest of our lesson on this particular one.

```
hiv_malawi_summary <- read_rds(here::here("data", "clean",
        "malawi_hiv_summary_t3.rds"))

hiv_malawi_summary
```

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

Themes

Since the objective of this lesson is mainly styling, let's start with using a pre-defined theme to add more visuals and colors to the table and its components. To do so we use the `opt_stylize` function. The function contains multiple pre-defined styles and can accept a color as well. In our case we chose to go with style No.6 and the color 'gray', you can set these arguments to your liking.

```
t1 <- hiv_malawi_summary %>%
  opt_stylize(
    style = 1,
    color = 'cyan'
  ) %>%
  tab_options(
    stub.background.color = '#F4F4F4',
  )

t1
```

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

For more sophisticated themes and styling, you can refer to the function `tab_options` (documentations here) which is basically the equivalent to the `theme` function in `ggplot2`. This function contains arguments and options on every single layer and component of the table. For the purposes of this lesson we won't dive into it.

Formatting the values in the table

Wouldn't it be useful to visualize in colors the difference between values in a specific column? In many reports, these kind of tables are quite useful especially if the number of rows is quite large. Let's do this for our table such that we have the `new_positive` column is formatted red.

We can do this by means of the `data_color` function for which we need two specify tow arguments, `columns` (as in at what column this styling will take place?) and `palette` as the color palette we intend to use.

```
t2 <- t1 %>%
  data_color(
    columns = new_positive, # the column or columns as we will see later
    palette = "ggsci::red_material" # the palette form the ggsci package.
  )

t2
```

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

`ggsci::red_material` is not the only palette we can use, in fact there are hundereds of palettes that are designed to be used in R. You can find a lot more in the `paletteer` package documentations in here, or in the official `data_color` documentation here.

We can do this for the `previous_negative` column as well. We can use a different kind of palette, I'm using for this case the green palette from the same package:

`ggsci::green_material` , the palette you choose is a matter of convenience and personal taste, you can explore more about this if you refer to the side note above.

```
t2 %>%
  data_color(
    columns = previous_negative,
    palette = "ggsci::green_material"
  )
```

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

Similarly, we can also color multiple columns at once, for example we can style the columns with positive cases in red, and those with negative cases in green. To do this we need to write *two* `data_color` statements one for each color style:

```
t4 <- t1 %>%

  data_color(
    columns = ends_with("positive"), # selecting columns ending with the wor
        positive
    palette = "ggsci::red_material" # red palette
  ) %>%
  data_color(
    columns = ends_with("negative"), # selecting columns ending with the wor
        negative
    palette = "ggsci::green_material" # green palette
  )


t4
```

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

Remember in the previous lesson we used the `tidyselect` functions to select columns, in the code above we used the function `ends_with` to select the columns ending either with the word 'negative' or 'positive' which is perfect for the purpose of our table.

Again, the column labels in the `gt` table and the actual column names in the `data.frame` can be different, in our case we refer to the names in the data.

Conditional formatting

We can also set up the table to conditionally change the style of a cell given its value. In our case we want to highlight values in the column `previous_positive` according to a threshold (the value 15700). Greater or equal values than the threshold should be in green.

To achieve this we use the `tab_style` function where we specify two arguments:

- `stye` : where we specify the color in the `cell_text` function since we intend to manipulate the text within the cells.
- `location` : where we specify the columns and the rows of our manipulation in the `cells_body` since these cells are in the main body of the table.

Let's use the t2 table as an example:

```
t5 <- t2 %>%
  tab_style(
    style = cell_text(
      color = "red",
    ),
    locations = cells_body(
      columns = previous_positive,
      rows = previous_positive >= 15700
    )
  )
t5
```

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

In the code above, the condition over which the styling will occur is stated in :

locations = cells_body(columns = previous_positive, `rows = previous_positive >= 15700 )`

Also, note that we can pass more arguments to the `cell_text` function, such as the size and the font of the cells we intend to style.

What if we want to have a two sided condition over the same threshold? Can we have cells with values greater or equal to the threshold styled in green, and simultaneously other cells with values less than the threshold styled in…. cyan?

We absolutely can, we've already done the first part (in the previous code chunk), we just need to add a second condition in a similar manner but in a different `tab_style` statement:

```
t6 <- t5 %>%
  tab_style(
    style = cell_text(
      color = 'cyan'
    ),
    location = cells_body(
      columns = 'previous_positive',
      rows = previous_positive < 15700
    )
  )

t6
```

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

Question 1: Conditional Formatting To highlight (in yellow) rows in a `gt` table where the "hiv_positive" column exceeds 1,000, which R code snippet should you use?

A.

```
data %>%
  gt() %>%
  tab_style(
    style = cells_body(),
    columns = "Sales",
    conditions = style_number(Sales > 1000, background =
        "yellow")
  )
```

B.

```
data %>%
  gt() %>%
  tab_style(
    style = cells_data(columns = "Sales"),
    conditions = style_number(Sales > 1000, background =
        "yellow")
  )
```

C.


```
data %>%
  gt() %>%
  tab_style(
    style = cell_fill(
      color = "yellow"
    ),
    locations = cells_body(
      columns = "hiv_positive",
      rows = hiv_positive > 1000
    )
  )
```

D.

```
data %>%
  gt() %>%
  tab_style(
    style = cells_data(columns = "Sales"),
    conditions = style_text(Sales > 1000, background =
        "yellow")
  )
```

Question 2: Cell Coloration Fill

Using the `hiv_malawi` data frame, create a `gt` table that displays the total number (**sum**) of "*new_positive*" cases for each "*region*". Highlight cells with values more than 50 cases in *red* and cells with

less or equal to 50 in *green*. Complete the missing parts (_____) of this code to achieve this.

```
# Calculate the total_new_pos summary
total_summary <- hiv_malawi %>%
  group_by(_________) %>%
  summarize(total_new_positive = __________)

# Create a gt table and apply cell coloration
summary_table <- total_summary %>%
  gt() %>%
  tab_style(
    style = cell_fill(color = "red"),
    locations = _________(
      columns = "new_positive",
      rows = ________________
    )
  ) %>%
  tab_style(
    style = ____________________,
    locations = cells_body(
      columns = "new_positive",
      _______ new_positive <= 50
    )
  )
```

Fonts and text

Now, we'll enhance the visual appeal of our table's text. For this, we'll use the `gt::tab_style()` function once again.

Let's modify the font and color of the title and the subtitle. We'll select the `Yanone Kaffeesatz` font from Google Fonts, a resource offering a vast array of fonts that can add a unique touch to your table, beyond the standard options in Excel.

To apply these changes, we'll configure the `gt::tab_style()` function as follows:

- The `style` argument is assigned the `cell_text()` function, which houses two other arguments:
  - `font` is assigned the `google_font()` function with our chosen font name.
  - `color` is set to a hexadecimal color code that corresponds to our desired text color.
- The `locations` argument is assigned the `cells_title()` function:

- We specify `title` and `subtitle` within the `groups` argument using vector notation `c(…)`.

To specifically modify the title or subtitle, you can use `locations = cells_title(groups = "title")` or `locations = cells_title(groups = "subtitle")`, respectively, without the need for `c(…)`.

Using lists to pass arguments in gt: Lists in R are an integral part of the language and are extremely versatile. A list can contain elements of different types (numbers, strings, vectors, and even other lists) and each element can be accessed by its index. In the context of our {gt} table, we use lists to group together style properties (with the style argument) and to specify multiple locations in the table where these styles should be applied (with the locations argument).

Using Hexadecimal Color Codes: Colors in many programming languages, including R, can be specified using hexadecimal color codes. These codes start with a hash symbol (#) and are followed by six hexadecimal digits. The first two digits represent the red component, the next two represent the green component, and the last two represent the blue component. So, when we set color = “#00353f”, we’re specifying a color that has no red, a bit of green, and a good amount of blue, which results in a deep blue color. This allows us to have precise control over the colors we use in our tables.

```
t7 <- t4 %>%
  tab_style(
      style = cell_text(
          font = google_font(name = 'Yanone Kaffeesatz'),
          color = "#00353f"
        ),
      locations = cells_title(groups = c("title", "subtitle"))
  )
t7
```

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				

We can extend our customization to include the labels for columns, spanners, and stubs, as well as the source note. Within the `locations` argument, we'll supply a list indicating the specific locations for these changes. For a comprehensive understanding of the locations, please refer to Appendix (List 1).

```
t8 <- t7 %>%
  tab_style(
    style = list(
      cell_text(
        font = google_font(name = "Montserrat"),
        color = "#00353f"
      )
    ),
    locations = list(
      cells_column_labels(columns = everything()), # select every column
      cells_column_spanners(spanners = everything()), # select all spanners
      cells_source_notes(),
      cells_stub()
    )
  )
t8
```

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

If you want to change the fill background of the title, you can do so by adjusting the `locations` argument to point at `cells_title(groups = "title")`. Here's how you could do it:

```
t9 <- t7 %>%
  tab_style(
    style = cell_fill(background = "#ffffff"),
    locations = cells_title(groups = "title")
  )
t9
```

In this code, `cell_fill(background = "#ffffff")` changes the background color to white, and `locations = cells_title(groups = "title")` applies this change specifically to the title of the table.

Question 2: Fonts and Text Which R code snippet allows you to change the font size of the footnote text in a `gt` table?

A.

```
data %>%
  gt() %>%
  tab_header(font.size = px(16))
```

B.

```
data %>%
  gt() %>%
  tab_style(
    style = cell_text(
      size = 16
    ),
    locations = cells_footnotes()
  )
```

C.


```
data %>%
  gt() %>%
  tab_style(
    style = cells_header(),
    css = "font-size: 16px;"
  )
```

D.

```
data %>%
  gt() %>%
  tab_style(
    style = cells_header(),
    css = "font-size: 16;"
  )
```

Borders

In `gt` it's also possible to draw borders in the tables to help the end user focus on specific area in the table. In order to add borders to a `gt` table we will use, again the, `tab_style` function and, again, specify the style and locations argument. The only difference now is that we will use the `cell_borders` helper function and assign it to the style argument. Here's how:

Let's first add a vertical line:

```
t10 <- t8 %>%
  tab_style(
    style = cell_borders( # we are adding a border
      sides = "left",     # to the left of the selected location
      color = "#45785e",     # with a dark green color
      weight = px(5)      # and five pixels of thickness
    ),
    locations = cells_body(columns = 2) # add this border line to the left of
        column 2
  )
t10
```

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

Now let's add another pink horizontal border line:

```
t11 <- t10 %>%
    tab_style(
    style = cell_borders( # we are adding a border line
      sides = "bottom",   # to the bottom of the selected location
      color = "#45785e",     # with a pink color
      weight = px(5)      # and five pixels of thickness
    ),
    locations = list(
      cells_column_labels(columns = everything()), # add this border line to
        the bottom of the column labels
      cells_stubhead()                             # and to the stubhead
    )
  )

t11
```

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				

Question 4: Borders To add a solid border around the entire `gt` table, which R code snippet should you use?

Hint : we can use a function that sets options for the entirety of the table, just like the `theme` function for the `ggplot` package.

A.

```
data %>%
  gt() %>%
  tab_options(table.border.top.style = "solid")
```

B.

```
data %>%
  gt() %>%
  tab_options(table.border.style = "solid")
```

C.

```
data %>%
  gt() %>%
  tab_style(
    style = cells_table(),
    css = "border: 1px solid black;"
  )
```

D.


```
data %>%
  gt() %>%
  tab_style(
    style = cells_body(),
    css = "border: 1px solid black;"
  )
```

Answer Key

1.C

2.

```
# Solutions are where the numbered lines are

# Calculate the total_new_pos summary
total_summary <- hiv_malawi %>%
  group_by(region) %>% ##1
  summarize(total_new_positive = new_positive) ##2

# Create a gt table and apply cell coloration
summary_table <- total_summary %>%
  gt() %>% ##3
  tab_style(
    style = cell_fill(color = "red"),
    locations = cells_body( ##4
      columns = "new_positive",
      rows = new_positive >= 50 ##5
    )
  ) %>%
  tab_style(
    style = cell_fill(color = "green"), ##6
    locations = cells_body(
      columns = "new_positive",
      rows = new_positive < 50  ##7
    )
  )
```

3.B

4.B

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

External resources and packages

- The definite cookbook of `gt` by Tom Mock : https://themockup.blog/static/resources/gt-cookbook.html#introduction
- the Grammar of Table article : https://themockup.blog/posts/2020-05-16-gt-a-grammar-of-tables/#add-titles
- official `gt` documentation page : https://gt.rstudio.com/articles/intro-creating-gt-tables.html
- Create Awesome HTML Table with knitr::kable and kableExtra book by Hao Zhu : https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html#Overview

Appendix

The `{gt}` package in R provides a variety of functions to specify locations within a table where certain styles or formatting should be applied. Here are some of them:

- `cells_body()`: This function targets cells within the body of the table. You can further specify rows and columns to target a subset of the body.
- `cells_column_labels()`: This function targets the cells that contain the column labels.

- `cells_column_spanners()`: This function targets cells that span multiple columns.
- `cells_footnotes()`: This function targets cells that contain footnotes.
- `cells_grand_summary()`: This function targets cells that contain grand summary rows.
- `cells_group()`: This function targets cells that contain group label rows.
- `cells_row_groups()`: This function targets cells that contain row group label rows.
- `cells_source_notes()`: This function targets cells that contain source notes.
- `cells_stub()`: This function targets cells in the table stub (the labels in the first column of the table).
- `cells_stubhead()`: This function targets the cell that contains the stubhead.
- `cells_stub_summary()`: This function targets cells that contain stub summary rows.
- `cells_title()` : This function targets cells that contain the table title and subtitle.
- `cells_summary()`: This function targets cells that contain summary rows.

These functions can be used in the `locations` argument of the `tab_style()` function to apply specific styles to different parts of the table.

Creating Choropleth maps with {ggplot2}

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

Creating Choropleth maps with {ggplot2}

Learning objectives

1. **Choropleth Maps with {ggplot2}:**
    - Master `ggplot()` and `geom_sf()` functions for map visualization.
2. **Data Matching with Polygons:**
    - Obtain boundaries and disease-related data.
    - Combine data based on administrative levels.
3. **Color Scaling Techniques:**
    - Implement color scales for both continuous and discrete data types.
4. **Faceting for Map Visualization:**
    - Use `facet_wrap()` and `facet_grid()` to create small multiple maps.

Introduction

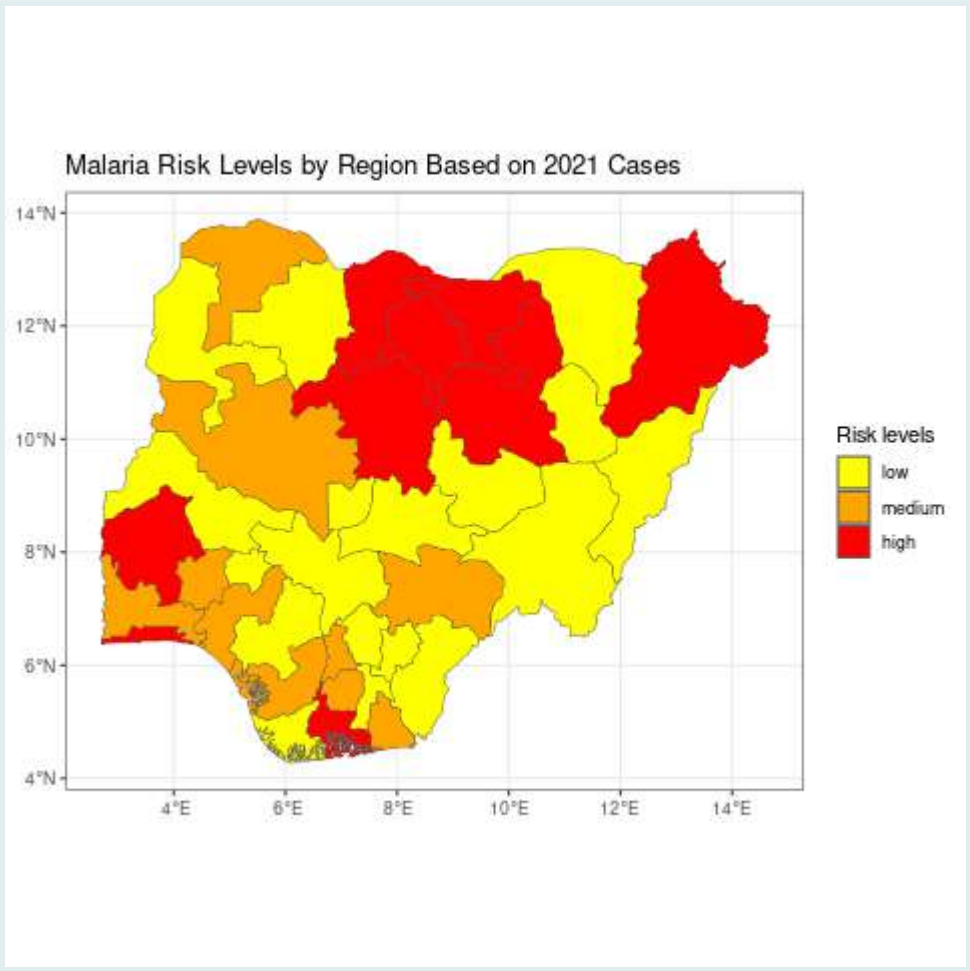
A choropleth map is a thematic map in which geographic regions are shaded or patterned in proportion to the value of a variable being represented. This variable can be an epidemiological indicator such as disease prevalence or mortality rate. Choropleth maps are particularly useful for visualizing spatial patterns and variations across different regions.

The essential components of a choropleth map include:

Geographic Regions: These are the areas that will be represented on the map, such as countries, states, districts, or any other geographic divisions.

Data Values: These are the values associated with each geographic region that will be represented on the map, including population density, prevalence, incidence, mortality rate, etc.

Color Scale: This is the range of colors used to represent the different data values. Typically, a gradient of colors is used, with lighter colors representing lower values and darker colors representing higher values.



SIDE NOTE

Choropleth maps offer:

- Clear visuals highlighting spatial data trends.
- Intuitive designs comprehensible without expertise.

However, they come with challenges:

- Data classification and color choices can skew appearances.
- Larger regions might dominate, causing visual bias.

In the following section, you will learn how to create a choropleth map using the {ggplot2} package in R.

Packages

This code snippet bellow shows how to load necessary packages using `p_load()` from the {pacman} package. If not installed, it installs the package before loading.

```
# Load necessary packages

# Use pacman to load multiple packages; it will install them if they're not
        already installed
pacman::p_load(tidyverse,   # for data wrangling and visualization
               here,        # for project-relative file paths
               sf)          # for spatial data

# Disable scientific notation for clearer numeric displays
options(scipen=10000)
```

Data Preparation

Before creating a choropleth map, it is essential to prepare the data. The data should contain the geographic regions and the values you want to visualize.

In this section, you will go through the data preparation process, which includes the following steps:

1. **Import polygon data**: This is the geographical data that contains the boundaries of each region that you want to include in your map.
2. **Import attribute data**: This is the data that contains the values you want to visualize on the map, such as disease prevalence, population density, etc.
3. **Join the polygon and attribute data**: This step involves merging the polygon data with the attribute data based on a common identifier, such as the administrative level or region name. This will create a single dataset that contains both the geographic boundaries and the corresponding data values.

Now, let's go through each step in detail!

Step 1: Import polygon data

Polygons are closed shapes with three or more sides. In spatial data, polygons are used to represent areas such as the boundary of a city, lake, or any land use type. They are essential in geographical information systems (GIS) for tasks like mapping, spatial analysis, and land cover classification.

SIDE NOTE

Shapefiles are a common format for storing spatial data. They consist of at least three files with extensions .shp (shape), .shx (index), and

SIDE NOTE

.dbf (attribute data).

In R, you can load shapefiles using the `sf` package. In our lesson today, we will be working with malaria data sourced from the Nigeria Epi Review (2022).

```
# Reading the shapefile
nga_adm1 <-
  sf::st_read(here::here("data/raw/NGA_adm_shapefile/NGA_adm1.shp"))
```

```
## Reading layer `NGA_adm1' from data source
##
`C:\Users\joych\epi_reports_staging\data\raw\NGA_adm_shapefile\NGA_adm1.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 38 features and 9 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 2.668431 ymin: 4.270418 xmax: 14.67642 ymax: 13.89201
## Geodetic CRS:  WGS 84
```

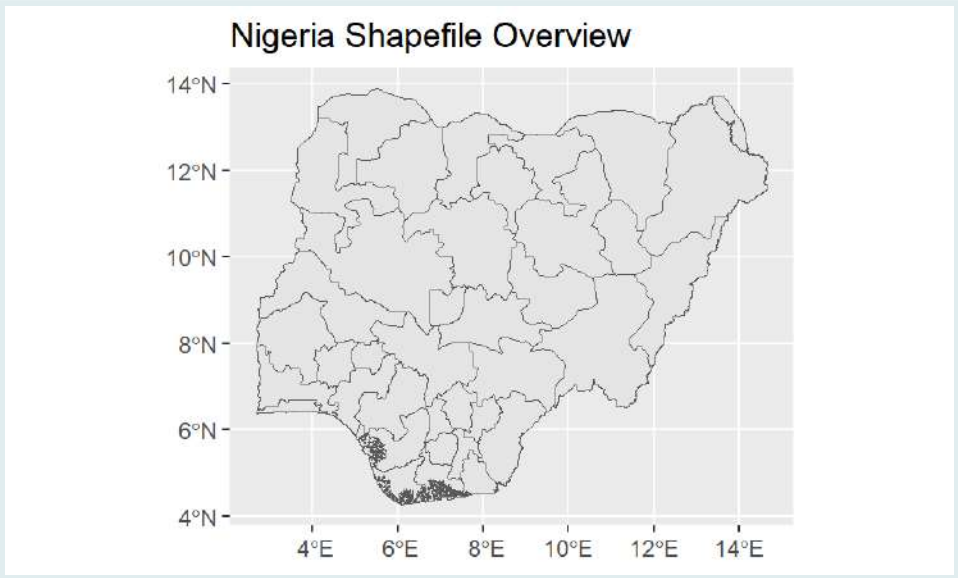
REMINDER

The important elements of any ggplot layer are the aesthetic mappings `aes(x,y, ...)` which tell ggplot where to place the plot objects.

We can imagine a map just like a graph with all features mapping to an x and y axis. All geometry (`geom_`) types in ggplot feature some kind of aesthetic mapping, and these can either be declaindianred at the plot level, e.g., `ggplot(data, aes(x = variable1, y = variable2))`, or at the individual layer level, e.g., `geom_point(aes(color = variable3))`.

Bellow you can see that `geom_sf()` is used to trace the boundaries of the different states of Nigeria. Similarly, different layers can be added on top.

```
ggplot()+
  geom_sf(data = nga_adm1)+
  labs(title = "Nigeria Shapefile Overview")
```



Here, we first read the Nigeria shapefile using `sf::st_read()` and then plot it using `ggplot`. The `geom_sf()` function is used to render the spatial data, and `labs()` is used to add a title to the plot.

PRO TIP

Since the shapefile was loaded using `sf::st_read()`, we don't need to specify the axis names. In fact, the coordinates are stoindianred as a multipolygon object in the geometry variable, and `geom_sf()` automatically recognizes them.

SIDE NOTE

For users who are interested in accessing subnational boundary data in R, there are a couple of packages available on GitHub which make this process straightforward:

- The {rgeoboundaries} (https://github.com/wmgeolab/rgeoboundaries) fetches data from the Geoboundaries, an open database of political administrative boundaries.
- The {rnaturalearth} (https://github.com/ropensci/rnaturalearth) fetches data from Natural Earth, a public domain map dataset.

Step 2: Import Attribute Data

VOCAB

In the context of choropleth maps, the “attribute data” refers to the quantitative or qualitative information that will be used to shade or color the various geographical areas on the map. For instance, if you have a map of a country’s regions and wish to shade each based on its population, the population data would be consideindianred the attribute data.

As highlighted above, we will be using the number of malaria cases reported in Nigeria for the years 2000, 2006, 2010, 2015, and 2021.

```
# Reading the attribute data
malaria_cases <- read_csv(here::here("data/malaria.csv"))
malaria_cases
```

Step 3: Checking the Joined Data

It is essential to check and validate the joined data to ensure that the merge will be successful and the data will be accurate.

```
# Validate the joined data
all.equal(unique(nga_adm1$NAME_1), unique(malaria_cases$state_name))
```

```
## [1] "Lengths (38, 37) differ (string compare on first 37)"
## [2] "2 string mismatches"
```

```
# Identify the mismatches
setdiff(unique(nga_adm1$NAME_1), unique(malaria_cases$state_name))
```

```
## [1] "Water body"
```

In the code snippet provided, we’re comparing the unique region names between the `nga_adm1` and `malaria_cases` datasets using the `all.equal()` function. This is to ensure that all the regions in the shapefile align with their counterparts in the attribute data.

If the region names are identical, `all.equal()` will return `TRUE`. However, if there are discrepancies, it will detail the differences between the two sets of region names.

It’s noted that the “Water body” is present in the shapefile but not in the `malaria_cases`. Since “Water body” isn’t a proper region, it should be removed prior to joining the datasets. We can do this with `filter()`.

```
nga_adm1 <- filter(nga_adm1, NAME_1 != "Water body")
```

Step 4: Join Data by Administrative Levels

Now we'll get the data we want to plot on the map. The important thing is that the region names are the same in the shapefile as in your data you want to plot, because that will be necessary for them to merge properly.

Before joining the two datasets, we need to define the join key (`by =`).

REMINDER

Remember to review the lesson on joining tables for more details on how to merge data.frames in R if you are not familiar with joining.

```
# Add population data and calculate cases per 10K population
malaria <- malaria_cases %>%
     left_join(nga_adm1,
               by = c("state_name" = "NAME_1")) %>%
     st_as_sf() # convert to shapefile

# Select the most important columns
malaria2 <- malaria%>%
     select(state_name,  cases_2000, cases_2006, cases_2010, cases_2015,
        cases_2021, geometry)
```

In this step, we merge the shapefile data `nga_adm1` with `malaria_cases` data using `left_join()` from the {dplyr} package. The `by` argument is used to specify the common column on which to merge the datasets.

Finally, we convert the merged data to a shapefile using `st_as_sf()`.

We can keep only the important variables that will be used in the construction of the plots by using `select()`.

RECAP

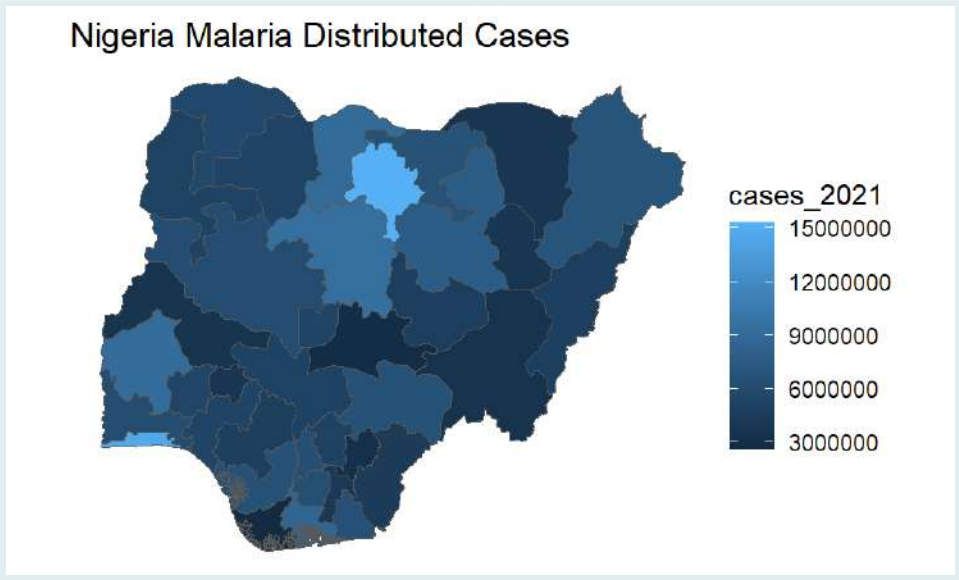
In this part of the lesson, we learned about the importance of administrative levels in spatial data and how to join spatial and attribute data by administrative levels using the {dplyr} package in R and how to check and validate the joined data.

Creating a Beautiful Choropleth Map with {ggplot2}

Using the Fill Variable (i.e., Attribute Variable)

To display the number of cases, for 2021 for example, we need to fill the polygons drawn with `geom_sf()` using the `fill` variable. This is very straightforward as it follows the same logic as the syntax of {ggplot2} package.

```
ggplot(data=malaria2) +
  geom_sf(aes(fill=cases_2021)) +   # set fill to vary by case count variable
  labs(title = "Nigeria Malaria Distributed Cases") +
  theme_void()
```

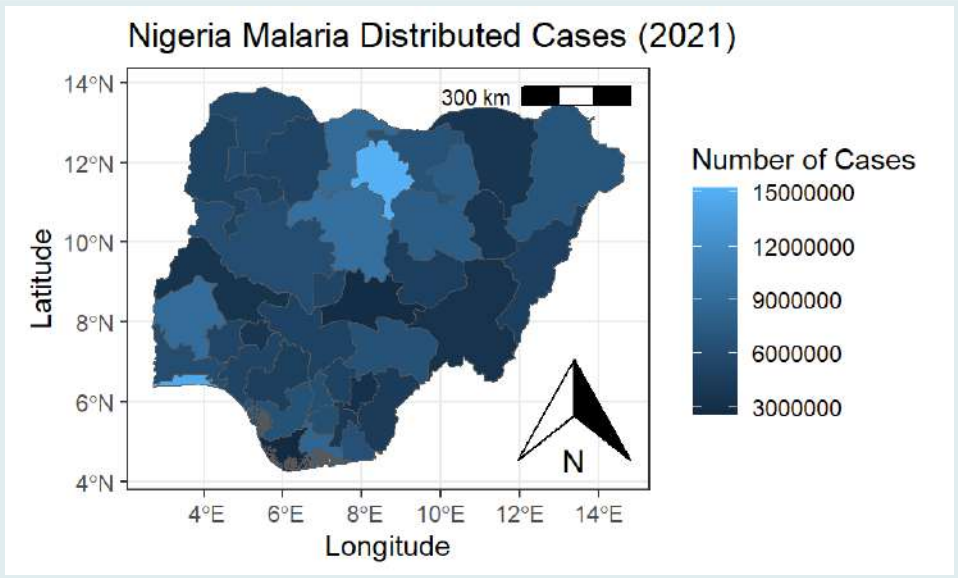


We create a basic choropleth map using {ggplot2}. The `fill` aesthetic is set to vary by the `cases` variable, which colors the regions based on the number of Malaria cases.

Customizing the Map

We can customize the map by adding titles to the axes and the legend, adding a north arrow and scale bar using `ggspatial::annotation_north_arrow()` and `ggspatial::annotation_scale()`, and changing the theme to `theme_bw()`.

```
ggplot(data=malaria2) +
  geom_sf(aes(fill=cases_2021)) +   # set fill to vary by case count variable
  labs(title = "Nigeria Malaria Distributed Cases (2021)",
       fill = "Number of Cases") +
  xlab("Longitude")+
  ylab("Latitude")+
  ggspatial::annotation_north_arrow(location = "br")+
  ggspatial::annotation_scale(location = "tr")+
  theme_bw()
```



RECAP

In this section, we first crafted a basic choropleth map using {ggplot2} to visualize regional malaria cases in Nigeria for 2021. Then, we enhanced the map by adding **titles**, **axis-labels**, a **north arrow**, a **scale bar**, and applying a monochrome **theme**.

1. Construct your beautiful choropleth map

Construct a choropleth map to display the distribution of Malaria cases in 2019, using the `cases_2019` column from the `malaria2` dataset. You can elevate your map's design and clarity by incorporating titles, axis labels, and any other pertinent labels.

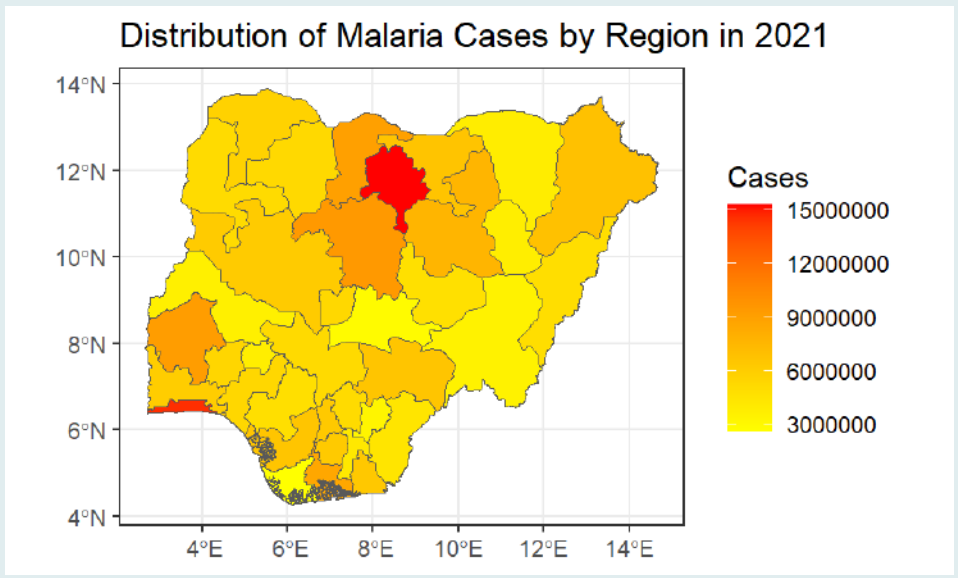
Color Scaling

Color Scaling for Continuous Attributes

The `scale_fill_continuous()` function from the {ggplot2} package in R is used to apply continuous color scaling to a choropleth map.

We can customize the color palette used for continuous color scaling by specifying the `low` and `high` parameters in the `scale_fill_continuous()` function.

```
# Create a ggplot object
ggplot(data = malaria2) +
  geom_sf(aes(fill = cases_2021)) +
  scale_fill_continuous(low = "yellow", high = "red", name = "Cases") + #
        apply continuous color scaling
  ggtitle("Distribution of Malaria Cases by Region in 2021") + # Add the
        corrected title here
  theme_bw()
```



RECAP

In this section, we learned how to apply continuous color scaling to a choropleth map using the `scale_fill_continuous()` function from the {ggplot2} package in R and how to customize the color palette.

Color Scaling for Discrete Attributes

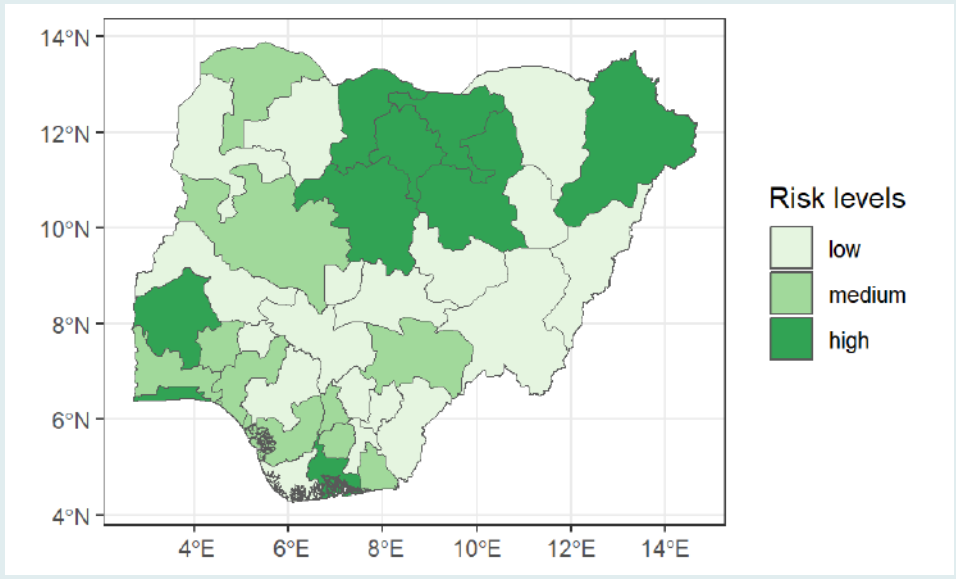
REMINDER

Discrete data is a type of quantitative data that can only take specific, separate values. It is often the result of counting objects or events. Discrete data is important in choropleth maps as it allows us to represent the count or quantity of objects or events in different regions.

The `scale_fill_brewer()` function from the {ggplot2} package in R is used to apply discrete color scaling to a choropleth map.

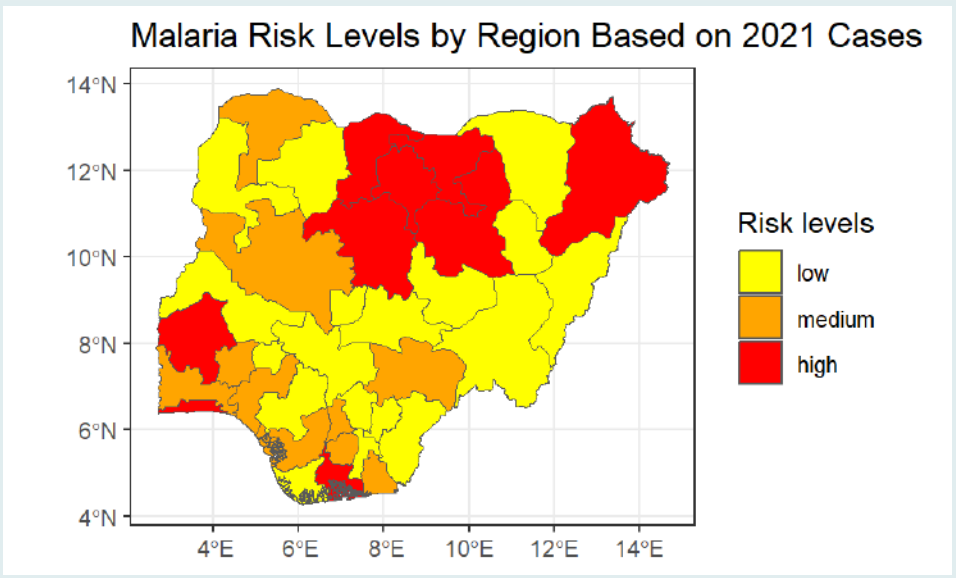
Before applying the discrete color scaling, we will need to create a new discrete column, which could be a risk level based on the number of cases. For this, we can use `mutate()` combined with `case_when()`.

```
# Transforming the data: Creating a new 'risk' column based on the number of
        cases in 2021
malaria2 %>%
  mutate(risk = case_when(cases_2021 < quantile(cases_2021, 0.5) ~ 'low',
                          cases_2021 > quantile(cases_2021, 0.75) ~ 'high',
                          TRUE ~ 'medium')) -> malaria3

# Visualizing the data
ggplot(data = malaria3) +
  geom_sf(aes(fill =  fct_reorder(risk, cases_2021)))+ # The risk levels are
        reordeindianred based on the number of cases
  scale_fill_brewer(palette = "Set4", "Risk levels")+
  theme_bw()
```



You can also create a custom color palette for discrete variables.

```
custom_palette <- c("yellow", "orange", "red") # create a custom color palette
        manually

# Apply custom color palette
ggplot(data = malaria3) + # Create a ggplot object
   geom_sf(aes(fill =  fct_reorder(risk, cases_2021))) + # reorder the risk
        labels according to the number of cases
   scale_fill_manual(values = custom_palette, name = "Risk levels") +
   coord_sf(expand = TRUE) +
   ggtitle("Malaria Risk Levels by Region Based on 2021 Cases") + # Add title
        here
   theme_bw()
```




```
png("malaria_risk_levels_map_1.png")
print(p)
```

```
## function (..., .noWS = NULL, .renderHook = NULL)
## {
##     validateNoWS(.noWS)
##     contents <- dots_list(...)
##     tag("p", contents, .noWS = .noWS, .renderHook = .renderHook)
## }
## <bytecode: 0x000001c6be15dbf8>
## <environment: namespace:htmltools>
```

```
dev.off()
```

```
## png
##   2
```

RECAP

In this section, we learned how to apply discrete color scaling to a choropleth map using the `scale_fill_brewer()` function from the {ggplot2} package and how to customize the color palette.

PRACTICE (in RMD)

2. Create a your own color palette Create a your own color palette distinct from the initial one provided bellow, and display the Malaria

cases across Nigeria for 2000 using this custom color palette. Don't foreget to incorporate additional aesthetic enhancements.

Facet Wrap vs. Grid

- `facet_wrap()`: This function wraps a 1D sequence of panels into 2D. This is useful when you have a single variable with many levels and want to arrange the plots in a more space-efficient manner.
- `facet_grid()`: This function creates a matrix of panels defined by row and column faceting variables. It is most useful when you have two discrete variables, and all combinations of the variables exist in the data.

- `facet_wrap()` is used for single variable faceting, while `facet_grid()` is used for two-variable faceting.
- `facet_wrap()` arranges the panels in a 2D grid, while `facet_grid()` arranges them in a matrix.

Creating Small Multiples using facet_wrap()

We can use the `facet_wrap()` function to create small multiples based on a given variable. In the following example, we can use the `year` variable.

Remember that color scaling can be adjusted by `scale_fill_continuous()` if the variable is continuous or by `scale_fill_brewer()` if the variable is discrete.

Here, we need to make a slight adjustment to the structure of our dataset. Since the cases are presented by year and each year is represented in a separate column, we'll use the `pivot_longer()` function to consolidate them into a single column, along with an identifier key.

```
malaria3_longer <- malaria3 %>%
  pivot_longer(cols = `cases_2000`:`cases_2021`, names_to = "year", values_to
        = "cases")
```

It seems that the variable `year` in our dataset has a prefix `cases_` that we would like to remove for aesthetic reasons before plotting.

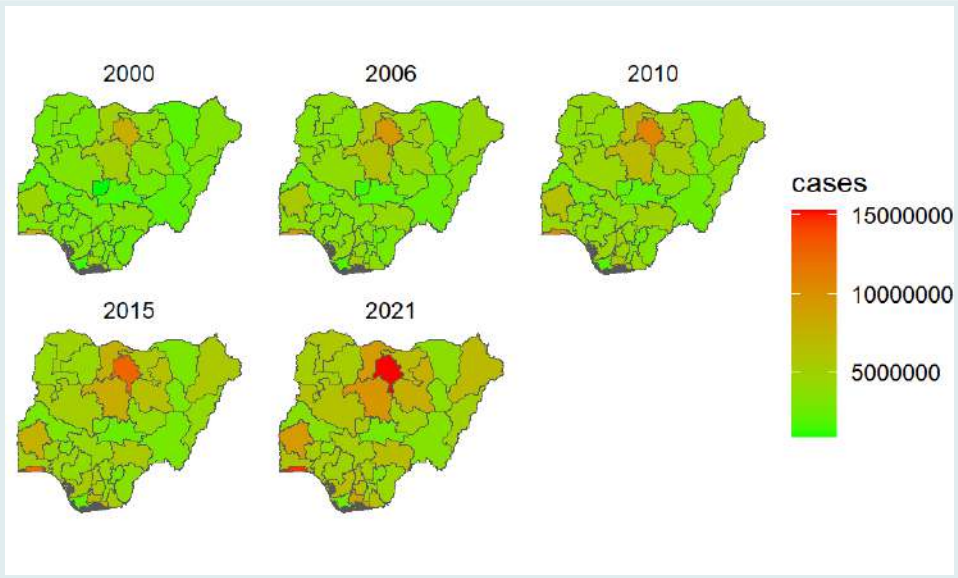
Here's how we can do it using the `str_replace()` function from {stringr} package, which is part of the tidyverse:

```
# Recode the 'year' variable by removing the 'cases_' prefix
malaria3_longer$year <- str_replace(malaria3_longer$year, "cases_", "")
```

After executing the code above, the `year` variable in our `malaria3_longer` dataset will no longer have the `cases_` prefix, making it cleaner for visualization purposes.

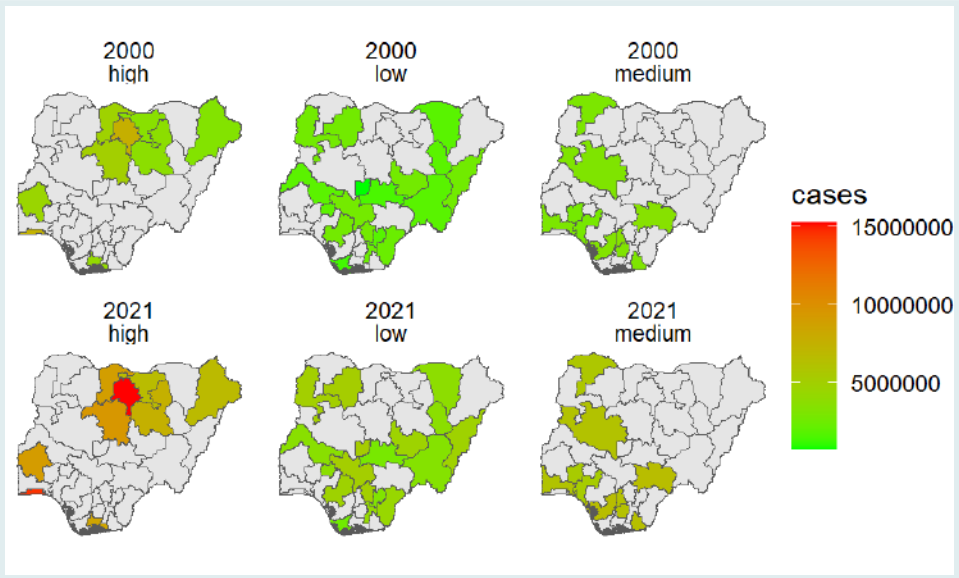
Now we want to create Small Multiples using `facet_wrap()`.

```
# Create ggplot object using facet_wrap
ggplot(data = malaria3_longer) +
  geom_sf(aes(fill = cases)) +
  facet_wrap(~ year)+
  scale_fill_continuous(low = "green", high = "red") + # apply continuous
        color scaling
  theme_void()
```



We can add another variable to the `facet_wrap()`. Here we add `risk`, but we will use only 2000 and 2021 cases for demonstration purpose:

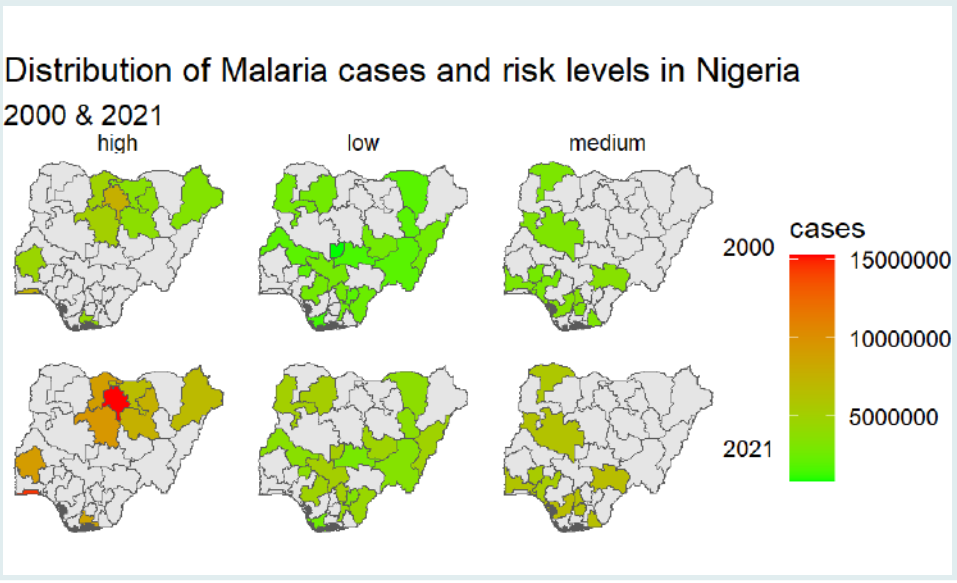
```
ggplot() +
  geom_sf(data = nga_adm1) +
  geom_sf(aes(fill = cases), data = filter(malaria3_longer, year %in%
        c("2000", "2021"))) +
  facet_wrap(year ~ risk)+
  coord_sf(expand = TRUE)+
  scale_fill_continuous(low = "green", high = "red")+
  theme_void()
```



In the last plot bellow, you will notice that we used a continuous fill variable, but we also used a `facet_wrap` on two other discrete variables.

Now, we will use `facet_grid()` to create a grid of plots. For each year (2000 and 2021), separate plots are made for each risk level, giving an easy visual comparison across both years and risks.

```
ggplot() +
  geom_sf(data = nga_adm1) + # need the
  geom_sf(aes(fill = cases), data = filter(malaria3_longer, year %in%
        c("2000", "2021"))) +
  facet_grid(year ~ risk)+
  coord_sf(expand = TRUE)+
  scale_fill_continuous(low = "green", high = "red")+
  labs(title = "Distribution of Malaria cases and risk levels in Nigeria",
       subtitle = "2000 & 2021")+
  xlab("Longitude")+
  ylab("Latitude")+
  theme_void()
```



PRO TIP

`facet_grid()` is ideal when you have two factors and you want to explore every combination. `facet_wrap()` is better when you have just one factor or when you have multiple factors but want a simpler layout.

Choosing between them often depends on the specific needs of your data and how you want to visualize the relationships.

CHALLENGE

3. Analyze the distribution of malaria cases

Your goal now is to analyze the distribution of malaria cases in Nigeria for the years 2000 and 2021. But you will need first to categorize the data into risk levels using the median (high/low), and then visualize this information on a map.

RECAP

In this section, we learned the differences between `facet_wrap()` and `facet_grid()`, and how to create small multiples using both of these functions.

WRAP UP!

Today, we went deep into the world of choropleth maps, unlocking the power of visual geographical representation.

You've gained insights into:

- The essence and components of a choropleth map.
- The pros and cons of these maps.
- Data preparation essentials for choropleth visualization.
- Crafting maps with {ggplot2} and applying varied color palettes.
- Harnessing `facet_wrap()` and `facet_grid()` for intricate map designs.

With these skills in hand, you're now equipped to innovate with different datasets and visualize them in impactful ways. Dive in, experiment, and enhance your map-making journey using {ggplot2} in R. Happy mapping!

Learning outcomes

Congratulations on completing this lesson! Let's recap the cognitive journey you've taken:

1. **Knowledge & Understanding:**
    - Recognized the definition and components of a choropleth map.
    - Comprehended the advantages and limitations of choropleth maps.
2. **Application:**
    - Prepaindianred data specifically for the creation of a choropleth map.
    - Employed {ggplot2} in R to design a choropleth map.
    - Implemented continuous and discrete color scaling techniques.
3. **Analysis:**
    - Differentiated between continuous and discrete color scaling.
4. **Synthesis:**
    - Integrated various components to create small multiples using `facet_wrap()` and `facet_grid()`.

With these skills developed, you are well-prepaindianred to not only replicate but also innovate and create your own choropleth map visualizations using {ggplot2} in

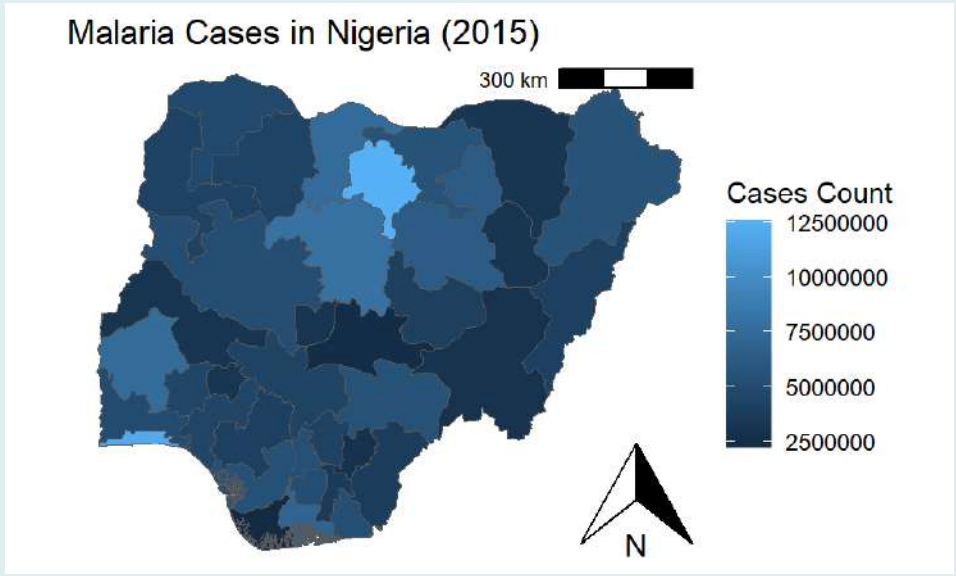
R.

Solutions

1. Construct your beautiful choropleth map

Construct a choropleth map to display the distribution of Malaria cases in 2019, using the `cases_2019` column from the `malaria2` dataset. You can elevate your map's design and clarity by incorporating titles, axis labels, and any other pertinent labels.

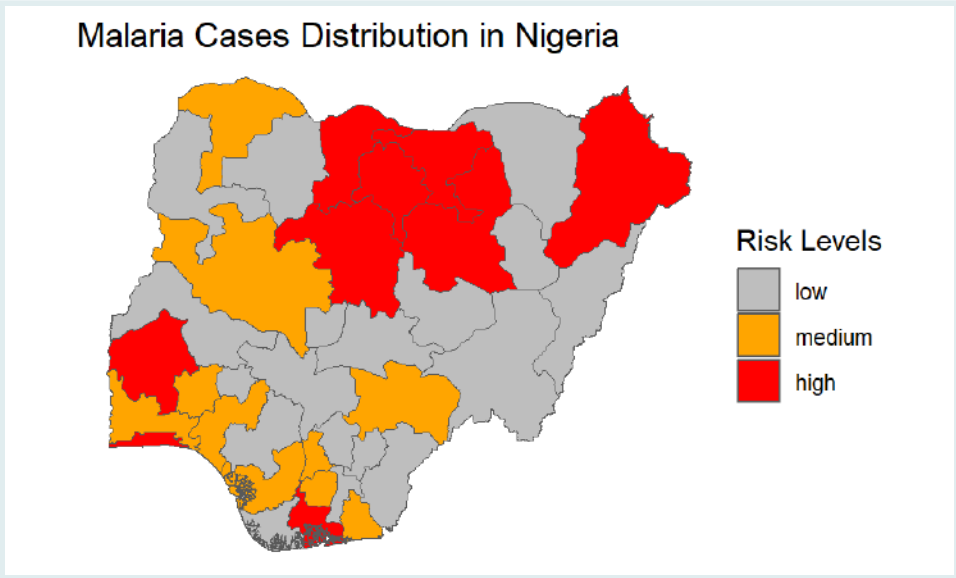
```
ggplot(data=malaria2) +
  geom_sf(aes(fill=cases_2015)) +
  labs(title = "Malaria Cases in Nigeria (2015)",
       fill = "Cases Count",
       x = "Longitude",
       y = "Latitude") +
  ggspatial::annotation_north_arrow(location = "br")+
  ggspatial::annotation_scale(location = "tr")+
  theme_void()
```



2. Create your own color palette

Create a your own color palette distinct from the initial one provided bellow, and display the Malaria cases across Nigeria for 2000 using this custom color palette. Don't foreget to incorporate additional aesthetic enhancements.

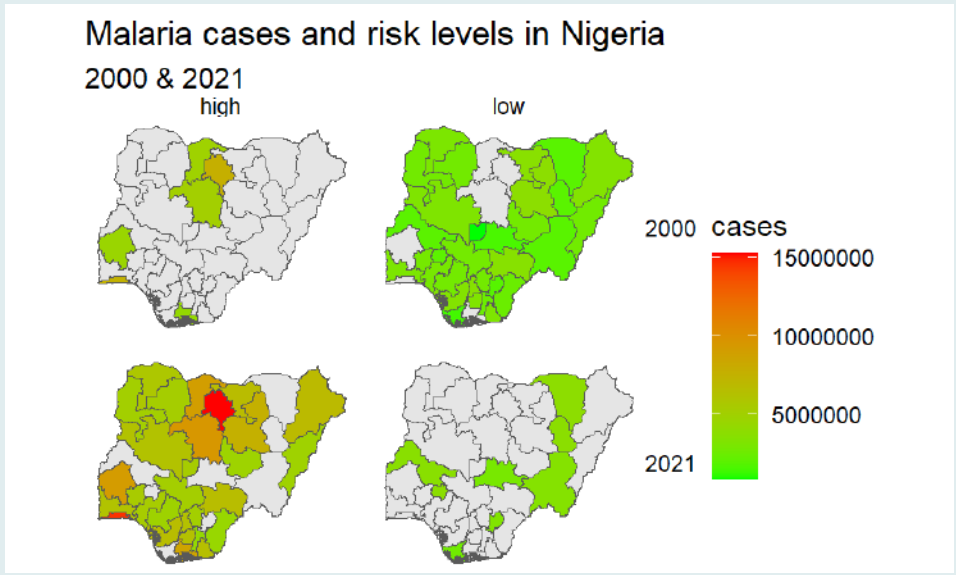
```
new_palette <- c("gray", "orange", "red") # Define a different set of colors
ggplot(data=malaria3) +
   geom_sf(aes(fill =  fct_reorder(risk, cases_2000))) + # Reorder risk labels
        according to year 2000 cases
   scale_fill_manual(values = new_palette, "Risk Levels") +
   labs(title = "Malaria Cases Distribution in Nigeria",
        fill = "Cases Count",
        x = "Longitude",
        y = "Latitude") +
   coord_sf(expand = TRUE)+
   theme_void()
```



3. Analyze the distribution of malaria cases

Your goal now is to analyze the distribution of malaria cases in Nigeria for the years 2000 and 2021. But you will need first to categorize the data into risk levels using the median (high/low), and then visualize this information on a map.

```r
# Pivoting the data
malaria3_longer <- malaria2 %>%
  pivot_longer(cols = `cases_2000`:`cases_2021`, names_to = "year", values_to
        = "cases")

# Categorizing risk based on median
malaria3_longer %>%
  mutate(risk = case_when(
    cases <= median(cases) ~ 'low',
    cases > median(cases) ~ 'high'
  )) -> malaria3_longer2

# Cleaning up the year values
malaria3_longer2$year <- str_replace(malaria3_longer2$year, "cases_", "")

# Plotting the data
ggplot() +
  geom_sf(data = nga_adm1) +
  geom_sf(aes(fill = cases), data = filter(malaria3_longer2, year %in%
        c("2000", "2021"))) +
  facet_grid(year ~ risk) +
  coord_sf(expand = TRUE) +
  scale_fill_continuous(low = "green", high = "red") +
  labs(title = "Malaria cases and risk levels in Nigeria",
       subtitle = "2000 & 2021") +
  xlab("Longitude") +
  ylab("Latitude") +
  theme_void()
```



Contributors

The following team members contributed to this lesson:

IMAD EL BADISY

Data Science Education Officer
Deeply interested in health data

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

JOY VAZ

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

References

- Wickham, Hadley, Winston Chang, and Maintainer Hadley Wickham. “Package 'ggplot2'.” *Create elegant data visualisations using the grammar of graphics. Version* 2, no. 1 (2016): 1-189.
- Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund. *R for data science*. ” O’Reilly Media, Inc.”, 2023.
- Lovelace, Robin, Jakub Nowosad, and Jannes Muenchow. *Geocomputation with R*. CRC Press, 2019.

Enhancing Disease Maps with Labels in R

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

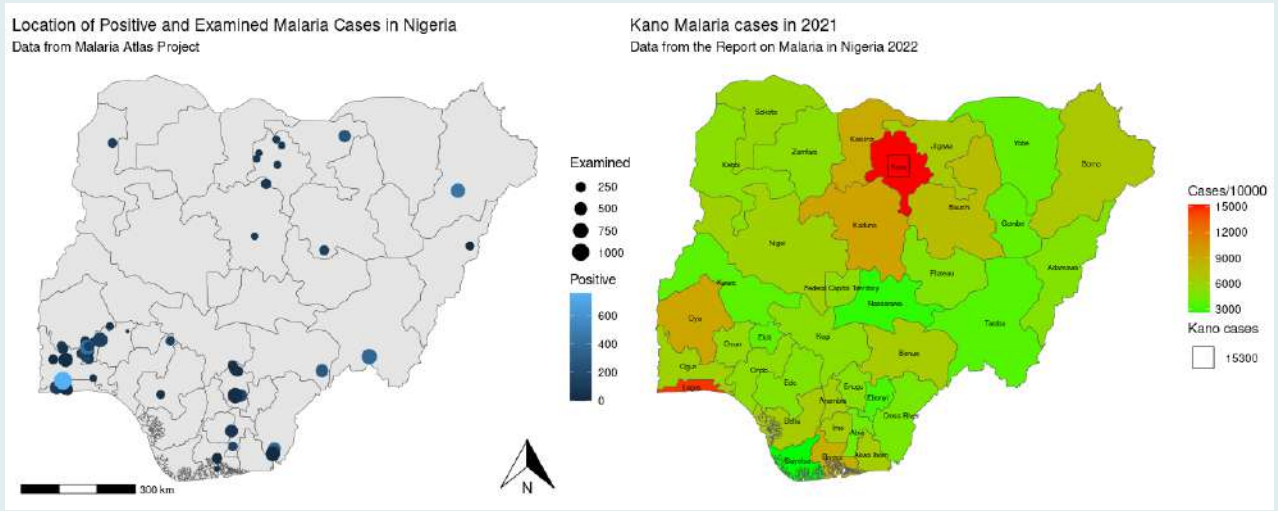
October 2023

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

In the geospatial data visualization, maps are powerful storytelling tools. However, a map without clear annotations and labels is like a book without titles or chapter headings. While the story might still be there, it becomes significantly harder to understand, interpret, and appreciate.

In this lesson, we place a special emphasis on the importance of annotating and labeling maps. Proper annotation transforms a simple visualization into an informative guide, allowing viewers to quickly grasp complex spatial data. With precise labeling, areas of interest can be immediately recognized, facilitating a clearer comprehension of the data's narrative.



Learning objectives

Learning Objectives: Advanced Geospatial Visualization Techniques

By the end of this section, you should be able to:

1. Incorporate continuous data indicators into choropleth maps for enhanced granularity.
2. Calculate and visualize the increase rates of malaria cases using choropleth maps.
3. Effectively overlay state names onto choropleth maps, ensuring clarity and readability.
4. Seamlessly integrate state names with increase rates on maps without compromising legibility.
5. Apply techniques to accentuate specific regions on a map while retaining the overall context.
6. Determine optimal point placement strategies and integrate them effectively into geospatial visualizations.

Upon mastering these objectives, you will have the tools and knowledge to create rich, detailed, and informative geospatial visualizations.

Packages

```
# Load packages
if(!require(pacman)) install.packages("pacman")
pacman::p_load(malariaAtlas,
               ggplot2,
               geodata,
               dplyr,
               here,
               readr,
               sf,
               patchwork)

# Unable scientific notation
options(scipen=100000)
```

Data Preparation

Before diving into any visualization or analysis, it's essential to load and preprocess our data. This includes reading in datasets, merging related information, and filtering out unnecessary or irrelevant entries.

```
# Reading the geographical shapefile data
nga_adm1 <- sf::st_read(here::here("data/raw/NGA_adm_shapefile/NGA_adm1.shp"))
```

```
## Reading layer `NGA_adm1' from data source
##
`C:\Users\joych\epi_reports_staging\data\raw\NGA_adm_shapefile\NGA_adm1.shp'
```

```
##   using driver `ESRI Shapefile'
## Simple feature collection with 38 features and 9 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 2.668431 ymin: 4.270418 xmax: 14.67642 ymax: 13.89201
## Geodetic CRS:  WGS 84
```

The geographical shapefile data for Nigeria's administrative boundaries (like states or provinces) is read and stored in the `nga_adm1` object.

```
# Reading the attribute data related to malaria cases
malaria_cases <- read_csv(here::here("data/malaria.csv"))
```

This step loads data related to malaria cases in different states of Nigeria.

```
# Filtering out the 'Water body' entries from the geographical data
nga_adm1 <- filter(nga_adm1, NAME_1 != "Water body")

# Merging the geographical data with the malaria cases data
malaria <- malaria_cases %>%
     left_join(nga_adm1, by = c("state_name" = "NAME_1")) %>%
     st_as_sf()
```

Here, we're combining the geographical boundaries data with the malaria cases data using state names as a reference. This merged data is then converted into a format suitable for geospatial visualizations.

```
# Filtering down to essential columns for our analysis
malaria2 <- malaria %>%
  select(state_name,  cases_2000, cases_2006, cases_2010, cases_2015,
         cases_2021, geometry)
```

We're narrowing down our dataset to specific columns, mainly the state names, the malaria cases from various years, and the geographical boundaries of these states (i.e. `geometry`).

```
# Reading in population data for different regions of Nigeria
population_nigeria <- read_csv(here::here("data/population_nigeria.csv"))
```

This step loads data indicating the population of different states or regions in Nigeria.

```
# Combining the population data with our malaria data
malaria3 <- malaria2 %>%
     left_join(population_nigeria, by = c("state_name")) %>%
     st_as_sf()
```

By merging the population data with our malaria cases data, we enrich our dataset. This combined data allows for more comprehensive visualizations and analyses, such as calculating incidence rates or assessing trends relative to population size.

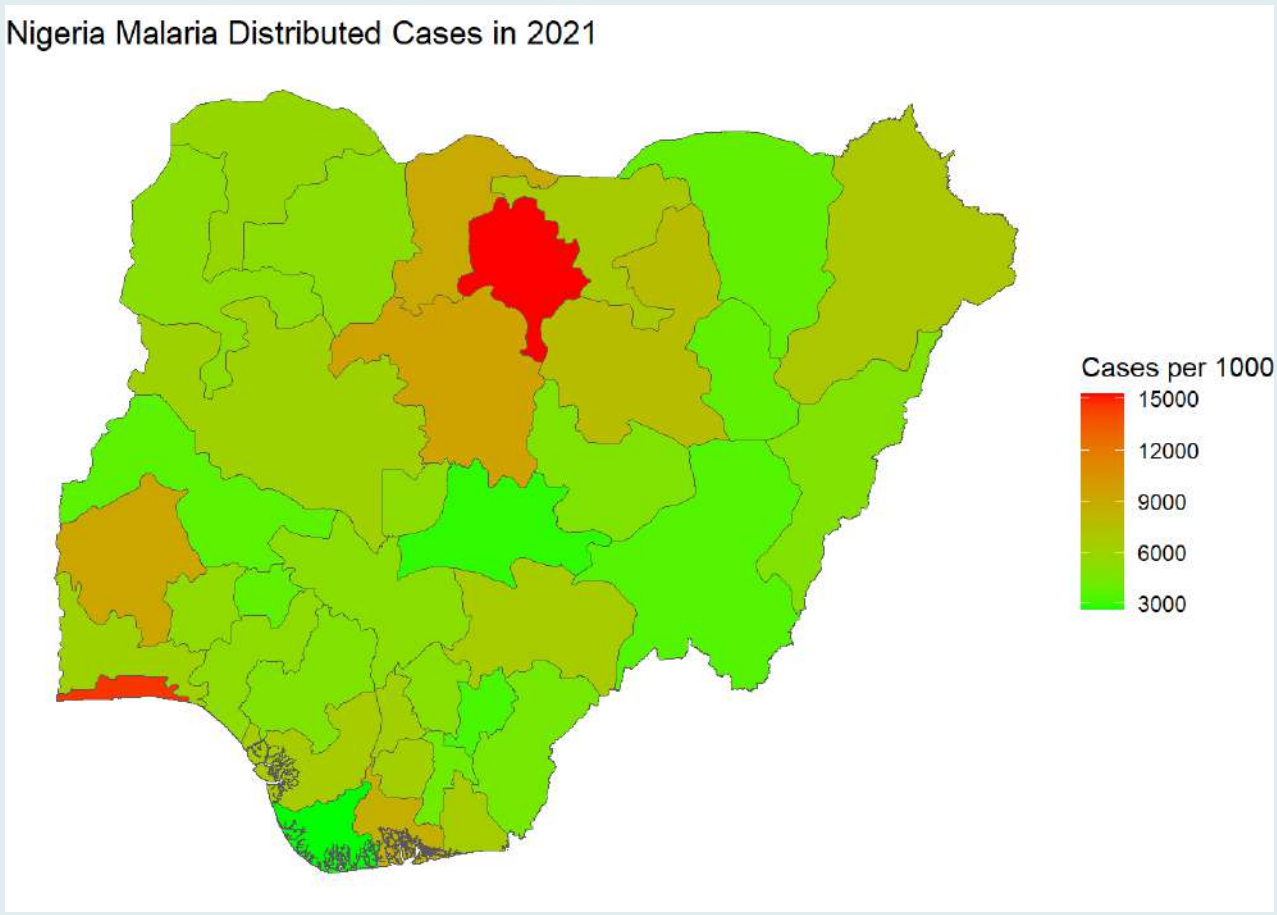
Building a Simple Choropleth Map

REMINDER

Choropleth maps are powerful visualization tools that display divided geographical areas shaded or patterned in proportion to the value of a variable.

In this example, we'll be using a choropleth map to visualize the distribution of malaria cases across different regions of Nigeria for the year 2021.

```
# Constructing the choropleth map using ggplot2
ggplot(data=malaria3) +
  geom_sf(aes(fill=cases_2021/1000)) + # The fill color is determined by the
        number of malaria cases in 2021, scaled per 1000
  labs(title = "Nigeria Malaria Distributed Cases in 2021", fill = "Cases per
        1000")+ # Adding labels and title to the plot
  scale_fill_continuous(low = "green", high = "red")+ # Using a continuous
        color scale transitioning from green to red
  theme_void() # Using a minimal theme for better visualization of the map
```



Here's a detailed explanation of the code:

- `ggplot(data=malaria3)`: Initiates a ggplot object using the `malaria3` dataset.
- `geom_sf(aes(fill=cases_2021/1000))`: Adds the geographical data from `malaria3` and fills each region based on the number of malaria cases in 2021, scaled down by a factor of 1000. This effectively represents the number of cases per thousand people.
- `labs(title = "Nigeria Malaria Distributed Cases in 2021", fill = "Cases per 1000")`: Specifies the title of the plot and the label for the color scale.
- `scale_fill_continuous(low = "green", high = "red")`: Applies a continuous color scale where regions with fewer cases are colored green and regions with more cases are colored red.
- `theme_void()`: Removes axis text, ticks, and other non-data ink to emphasize the map.

The resulting plot provides a clear view of how malaria cases are distributed across Nigeria in 2021, with the color intensity indicating the magnitude of cases in each region.

Practice 1: Modify the provided choropleth map to visualize the distribution of malaria cases in Nigeria for the year 2015.

Instructions

1. Update the data mapping in the `geom_sf()` function to reflect malaria cases for the year 2015.
2. Adjust the title in the `labs()` function to indicate that the visualization pertains to 2015.
3. Change the color gradient in `scale_fill_continuous()` to transition from blue (`low` cases) to yellow (`high` cases).

Bellow a starter code:


```
# Constructing the choropleth map for 2015 using ggplot2
ggplot(data=malaria3) +
  geom_sf(aes(fill=___________)) +  # Fill in the correct data
        column for 2015
  labs(title = "_____________________", fill = "Cases per
        1000")+  # Update the title appropriately
  scale_fill_continuous(___________)+  # Modify the color scale
  theme_void()
```

Adding Continuous Data Indicators to the Choropleth Map

When analyzing disease data, it's often useful to look beyond raw case numbers and focus on rates, particularly incidence rates. The incidence rate provides a normalized measure that can account for differing population sizes across regions, making comparisons more meaningful.

Understanding Incidence

The incidence of a disease is a cornerstone of epidemiological research. It quantifies the number of new cases of a disease that occur within a specific time frame, typically a year, relative to a population at risk.

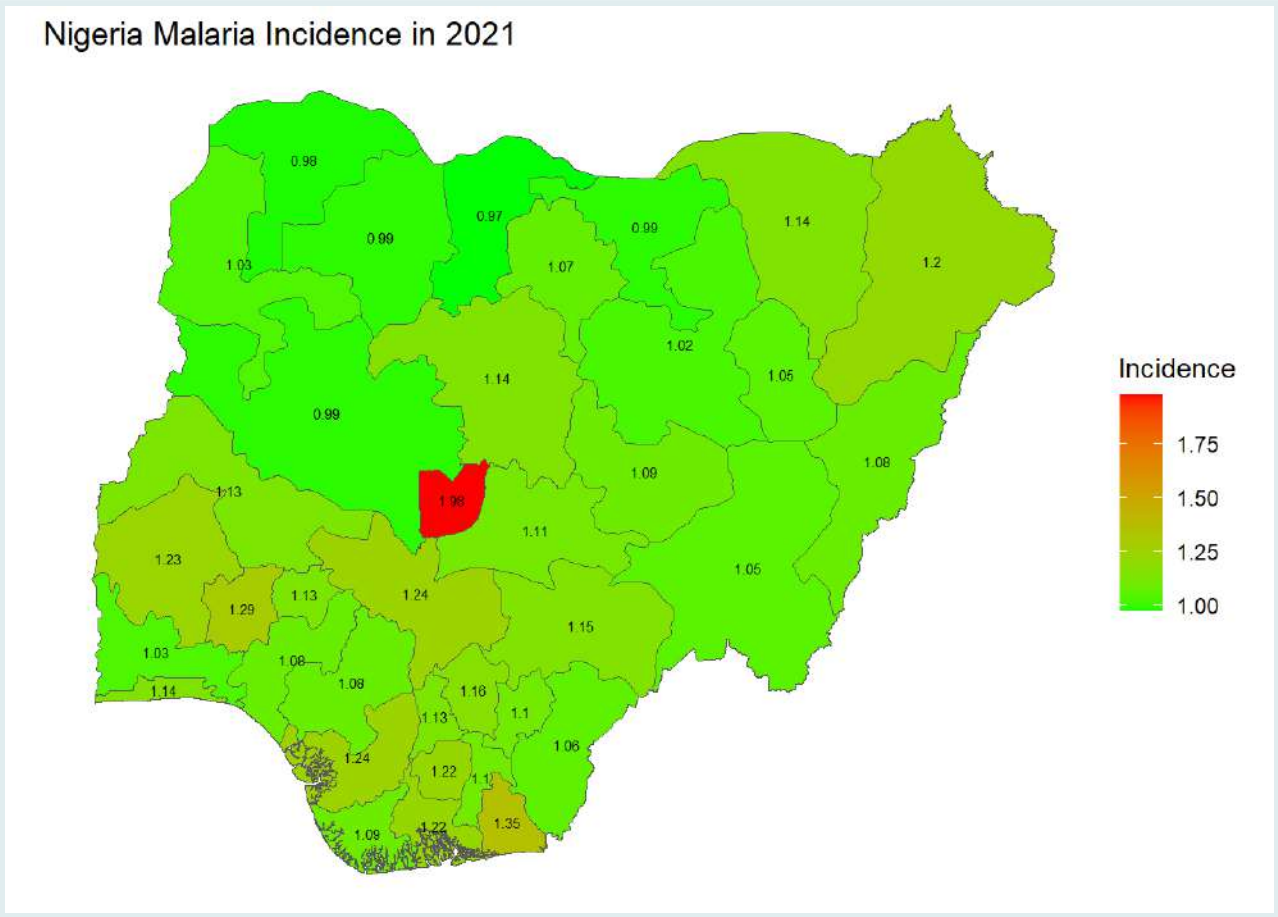
Mathematically, it's given by:

$$\text{Incidence} = \frac{\text{Number of new cases during the time period}}{\text{Population at risk at the start of the period}}$$

In this context, we're looking at the incidence of malaria in different states of Nigeria for the year 2021. Specifically, we'll compute the incidence rate by dividing the number of new malaria cases in 2021 by the population of each state from the last available census in 2019.

The following R code accomplishes this and visualizes the data:

```r
# Find the centroid coordinates of each state to position labels
centroid_coords <- st_coordinates(st_centroid(malaria2$geometry))

# Visualizing Malaria Incidence in 2021 using a Choropleth Map
ggplot(data = malaria3) +
  # Filling each state with a color representing the incidence rate in 2021.
  geom_sf(aes(fill = round(cases_2021/population_2019, 2))) +
  # Labeling each state with its incidence rate.
  geom_text(aes(x = centroid_coords[, 1], y = centroid_coords[, 2], label =
        round(cases_2021/population_2019, 2)), size = 2)+
  # Adding title and legend title.
  labs(title = "Nigeria Malaria Incidence in 2021", fill = "Incidence")+
  # Using a continuous color scale from green (low incidence) to red (high
        incidence).
  scale_fill_continuous(low = "green", high = "red")+
  # Using a minimalistic theme for clearer visualization.
  theme_void()
```



Here's a brief explanation of the visualization:

- The `geom_sf()` function creates a choropleth map where each state's color represents its malaria incidence rate in 2021.
- The `geom_text()` function labels each state with its specific incidence rate, positioning each label at the state's centroid.
- The color scale, transitioning from green to red, visually emphasizes regions with higher incidence rates.

This visualization offers an immediate grasp of the malaria situation in Nigeria, revealing areas of high incidence that might need more focused public health interventions.

Exploring Malaria Case Increase Rates Using Choropleth Maps

Understanding the change in the number of disease cases over time can provide insights into the effectiveness of interventions, the progression of the disease, and areas where increased resources might be needed. In this context, we're looking to visualize the percentage increase in malaria cases from 2015 to 2021 across different states in Nigeria.

Computing the Increase Rate

The increase rate for each state is computed as:

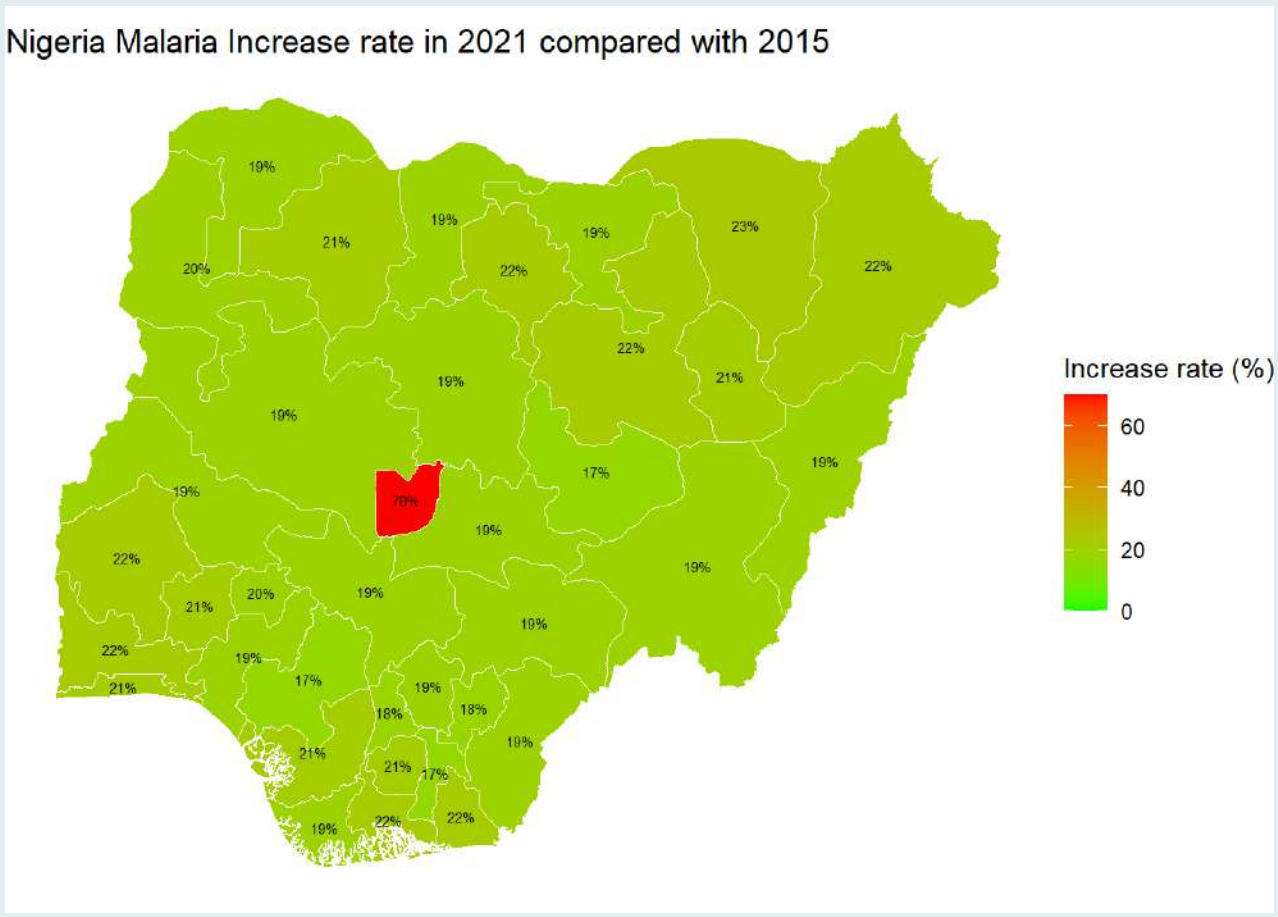
$$\text{Increase Rate} = \left(\frac{\text{Cases in 2021} - \text{Cases in 2015}}{\text{Cases in 2015}} \right) \times 100\%$$

This formula provides the percentage growth (or decrease) in malaria cases from 2015 to 2021.

Visualization of Increase Rates

Let's delve into the code that accomplishes this visualization:

```
# Calculate the increase rate for each state
malaria3 %>%
  mutate(increase_rate = round(((cases_2021 - cases_2015) / cases_2015) *
         100)) -> malaria3

# Visualizing the increase rates using a choropleth map
ggplot(data = malaria3) +
  # Coloring each state based on its increase rate
  geom_sf(aes(fill = increase_rate), color="white", size = 0.2) +
  # Using a continuous color scale to represent increase rates, transitioning
       from green to red
  scale_fill_continuous(name="Increase rate (%)", limits=c(0,70), low =
         "green", high = "red", breaks=c(0, 20, 40, 60))+
  # Labeling each state with its increase rate percentage
  geom_text(aes(x = centroid_coords[, 1], y = centroid_coords[, 2], label =
         paste0(increase_rate, "%")), size = 2)+
  # Adding a title to the plot
  labs(title = "Nigeria Malaria Increase rate in 2021 compared with 2015")+
  theme_void()
```



In this visualization:

- The `geom_sf()` function creates the choropleth map where each state's color intensity represents its malaria increase rate.
- `scale_fill_continuous()` sets the color scale for the increase rates, making areas of higher increase more prominent.
- `geom_text()` adds labels to each state, providing exact percentage values.
- The resulting map allows us to quickly identify regions with significant growth in malaria cases over the selected period.

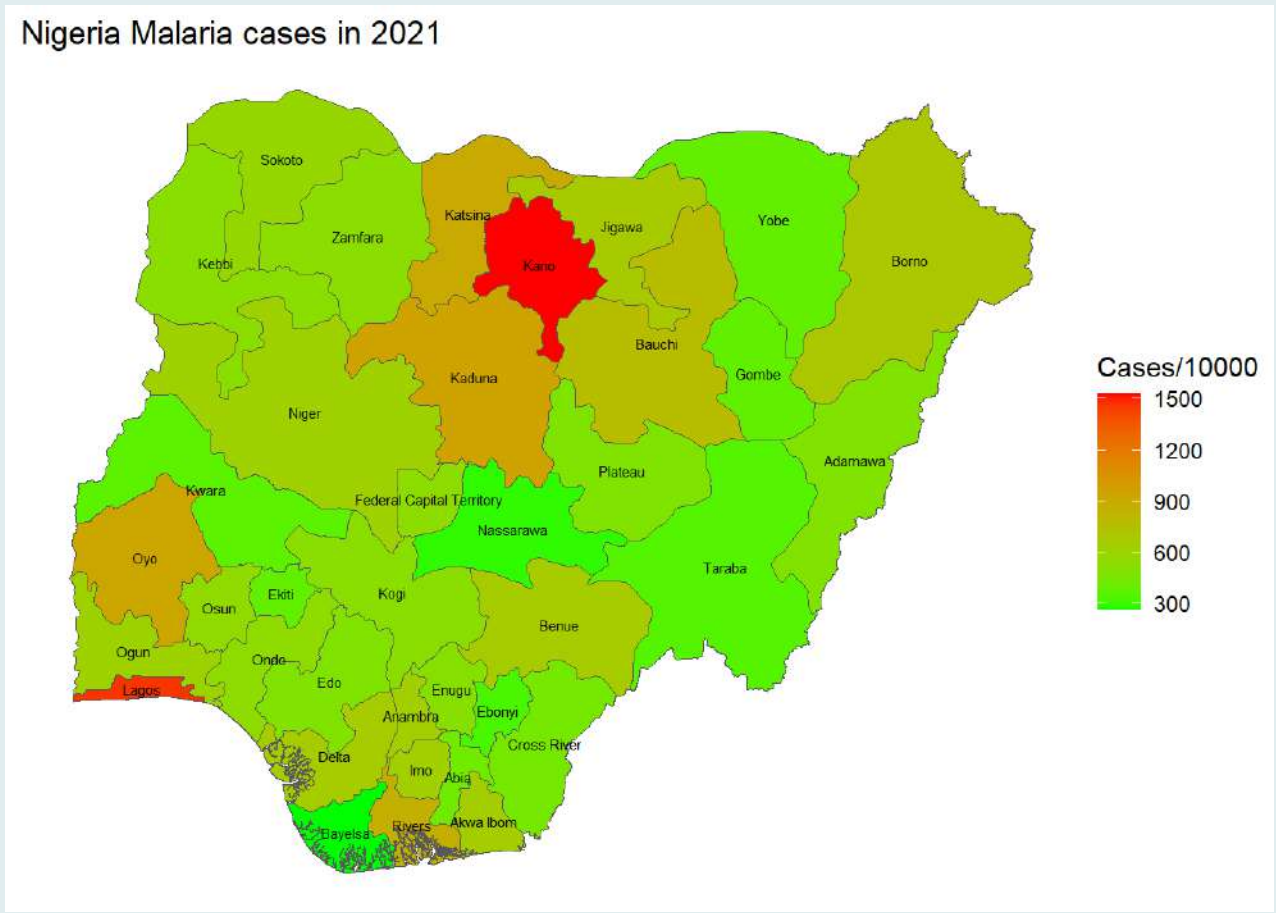
Through this visualization, stakeholders can pinpoint regions where malaria is on the rise and potentially allocate resources more effectively.

Labeling the Choropleth Map with State Names

In a choropleth map, while color gradients offer a visual cue to understand the distribution of a variable across regions, adding labels can significantly enhance the clarity of the visualization. This is especially true when audiences might not be familiar with all geographical boundaries shown. In the case of our malaria dataset, adding state names to the map makes the data more accessible and understandable.

Let's break down the code:

```
# Constructing a choropleth map with state names
ggplot(data = malaria3) +
  # Fill each state based on the number of malaria cases in 2021, scaled per
      10,000
  geom_sf(aes(fill = cases_2021/10000)) +
  # Add the name of each state to its centroid
  geom_text(aes(x = centroid_coords[, 1], y = centroid_coords[, 2], label =
      state_name), size = 2, check_overlap = TRUE)+
  # Add titles and labels
  labs(title = "Nigeria Malaria cases in 2021", fill = "Cases/10000")+
  # Use a continuous color scale from green (low case numbers) to red (high
      case numbers)
  scale_fill_continuous(low = "green", high = "red")+
  theme_void()
```



Here's a detailed explanation of the visualization:

- `geom_sf(aes(fill = cases_2021/10000))`: This creates the choropleth map where each state's color represents the number of malaria cases in 2021, scaled down by a factor of 10,000.

- `geom_text(aes(...))`: This function places text on the plot. In this instance, it's used to add the name of each state to its geographical centroid. The `check_overlap = TRUE` parameter ensures that ggplot checks for overlapping text labels and tries to prevent labels from overlapping.

- `labs(title = "Nigeria Malaria cases in 2021", fill = "Cases/10000")`: This function adds a title to the plot and a label to the color scale.

- `scale_fill_continuous(low = "green", high = "red")`: This sets the color scale for the map, transitioning from green for states with fewer cases to red for those with more cases.

The resulting visualization is a clear and informative map of malaria cases across Nigeria in 2021, with each state labeled for easy reference.

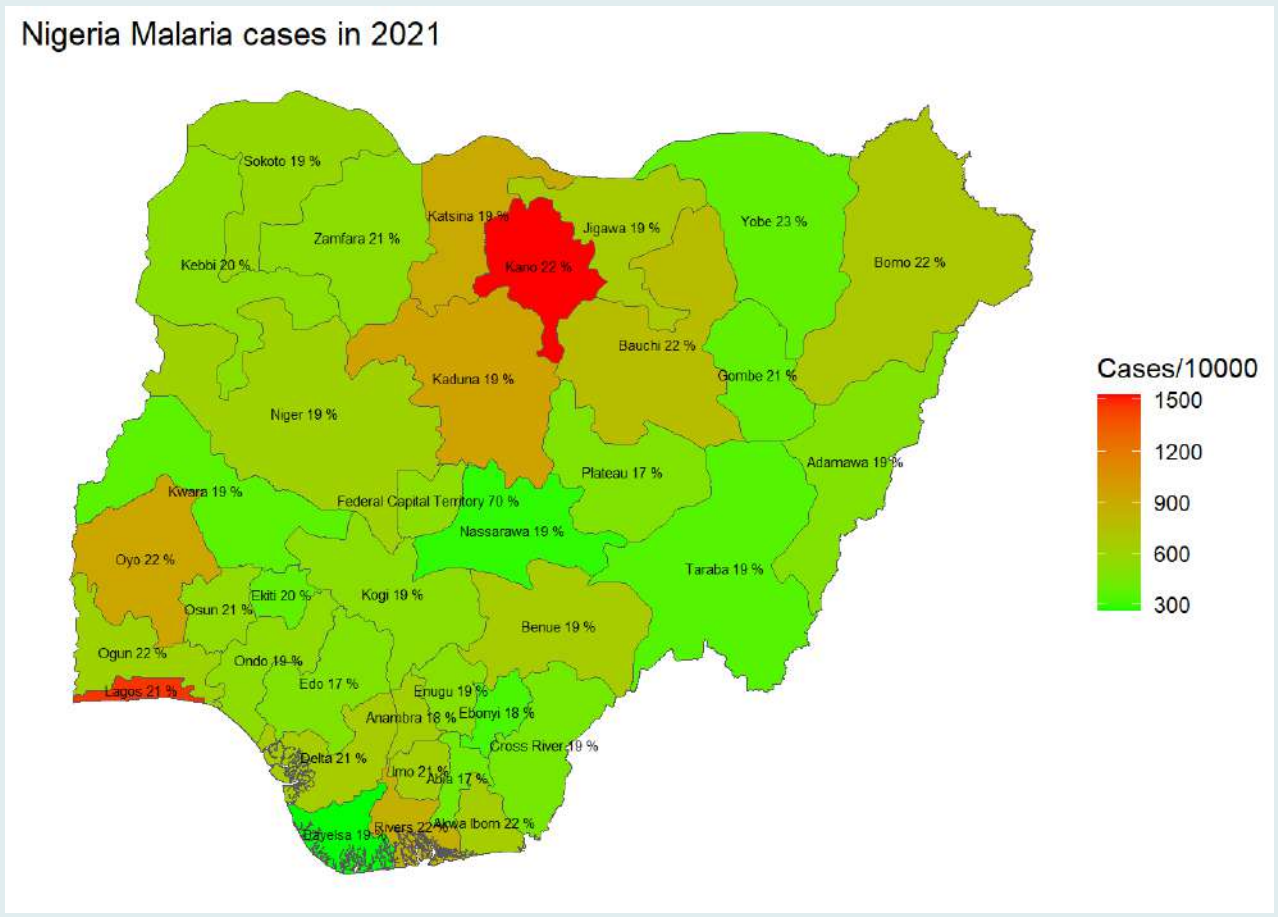
Displaying Combined State Names and Increase Rates on the Choropleth Map

Visualizations can convey a wealth of information when they incorporate multiple data points in an intuitive manner. By pairing state names with their corresponding increase rates, we can provide a richer, more detailed view of the data without overwhelming the audience.

Let's delve into this visualization:

We want to present a choropleth map showcasing the malaria cases per 10,000 residents across Nigerian states in 2021, with labels that combine state names and their respective increase rates from 2015.

```
# Combine the state names and their respective increase rates into a single
        label
malaria3$label_text <- paste(malaria3$state_name, malaria3$increase_rate, "%")

# Calculate the centroid of each state to serve as reference points for the
        labels
centroid_coords <- st_coordinates(st_centroid(malaria3$geometry))

# Visualize the data
ggplot(data = malaria3) +
  # Create a choropleth map shaded based on the number of malaria cases in
        2021 per 10,000 residents
  geom_sf(aes(fill = cases_2021/10000)) +
  # Add combined labels (state name and increase rate) to each state's
        centroid
  geom_text(aes(x = centroid_coords[, 1],
                y = centroid_coords[, 2],
                label = label_text), size = 2)+

  # Add titles and customize the color legend
  labs(title = "Nigeria Malaria cases in 2021", fill = "Cases/10000")+
  scale_fill_continuous(low = "green", high = "red") +  # Set color gradient
  theme_void()  # Apply a minimal theme for clarity
```



In this visualization:

- The line `malaria3$label_text <- paste(malaria3$state_name, malaria3$increase_rate, "%")` constructs our combined labels by concatenating the state name with its increase rate, followed by a percentage sign.
- `geom_text()` adds these combined labels to the plot. It positions each label at the centroid of the corresponding state, ensuring the labels are centered and easily associated with their respective regions.
- `scale_fill_continuous(low = "green", high = "red")` assigns a color gradient based on the number of malaria cases. States with fewer cases will be colored green, transitioning to red for states with more cases.

This approach allows us to efficiently communicate two vital pieces of information (cases per 10,000 and increase rate) within the same visualization while keeping the name of states for easy identification.

Practice 2: Create a single choropleth map that showcases the malaria increase rates across different states in Nigeria with each state labeled by its name

Instructions

1. Start by setting up the base plot using the malaria3 dataset.
2. Create a choropleth map where the fill color represents the increase rate from 2015 to 2021.
3. Label each state with its name using the centroids.
4. Customize the color scale to transition from green (low increase) to red (high increase).
5. Add appropriate titles and labels to the plot.

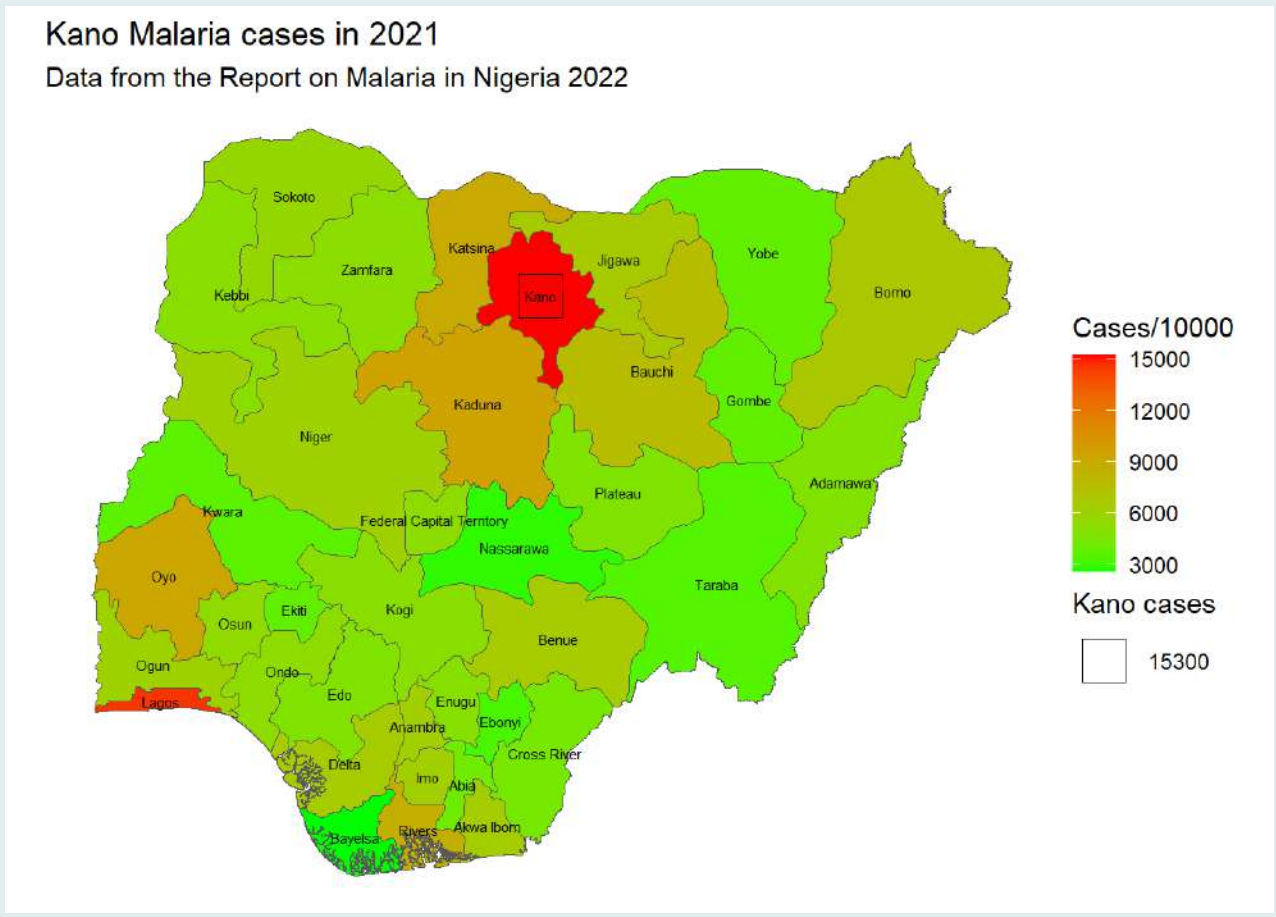
Bellow a starter code:


```
# Combining visualization of increase rates with state names
ggplot(data = malaria3) +
  _______ +  # Fill in the code to generate the choropleth map
  _______ +  # Add state names to each region
  _______ +  # Specify the color scale for increase rates
  _______    # Add titles and labels
```

Highlighting a Specific Region on the Map while Preserving Context

Sometimes, you might want to draw attention to a particular area or region on your map, without omitting details from the surrounding areas. By using specific graphical elements, like larger markers or distinctive colors, you can emphasize certain regions while still showcasing the broader data. In this example, we're focusing on the "Kano" region of Nigeria.

```r
# Calculate centroid coordinates for labeling
centroid_coords <- st_coordinates(st_centroid(malaria3$geometry))

# Visualize the malaria cases across Nigeria with an emphasis on Kano
ggplot(data = malaria3) +
  # Create a choropleth map with color based on malaria cases in 2021
  geom_sf(aes(fill = cases_2021/1000)) +
  # Set a continuous color gradient from green to red
  scale_fill_continuous(low = "green", high = "red")+

  # Overlay a point on Kano to emphasize it. The size of the point corresponds
        to the number of cases
  geom_point(data = subset(malaria3, state_name == "Kano"),
             aes(x = st_coordinates(st_centroid(geometry))[1],
                 y = st_coordinates(st_centroid(geometry))[2], size =
        round(cases_2021/1000)),
             color = "black", shape = 22,  fill = "transparent") +

  # Label each region with its name
  geom_text(aes(x = centroid_coords[, 1], y = centroid_coords[, 2], label =
        state_name), size = 2, check_overlap = TRUE)+

  # Customize the scale of the size of the emphasized point
  scale_size_continuous(range = c(1, 12)) +

  # Add title and legends
  labs(title = "Kano Malaria cases in 2021", subtitle = "Data from the Report
        on Malaria in Nigeria 2022", size = "Kano cases", fill =
        "Cases/10000",)+

  # Apply a minimal theme for clarity
  theme_void()
```



In this visualization:

- The `geom_point()` function is utilized to lay a circle over the Kano region. The size of the circle signifies the number of cases in Kano in 2021. The circle is designed transparent (`fill = "transparent"`) with a bold black boundary (`color = "black"`) to set it apart.
- `geom_text()` adds the names of all regions to the map. It places each label near the geographical center of the respective region. The parameter `check_overlap = TRUE` prevents labels from overlapping when possible, ensuring readability.

RECAP

While “Kano” is emphasized, all other regions are also displayed with their respective color shading based on malaria cases. This offers a holistic view of the situation across Nigeria, enabling viewers to compare Kano with other regions.

Such an approach is invaluable when you wish to spotlight specific details or areas of interest without sidelining the broader dataset, enriching your data presentations.

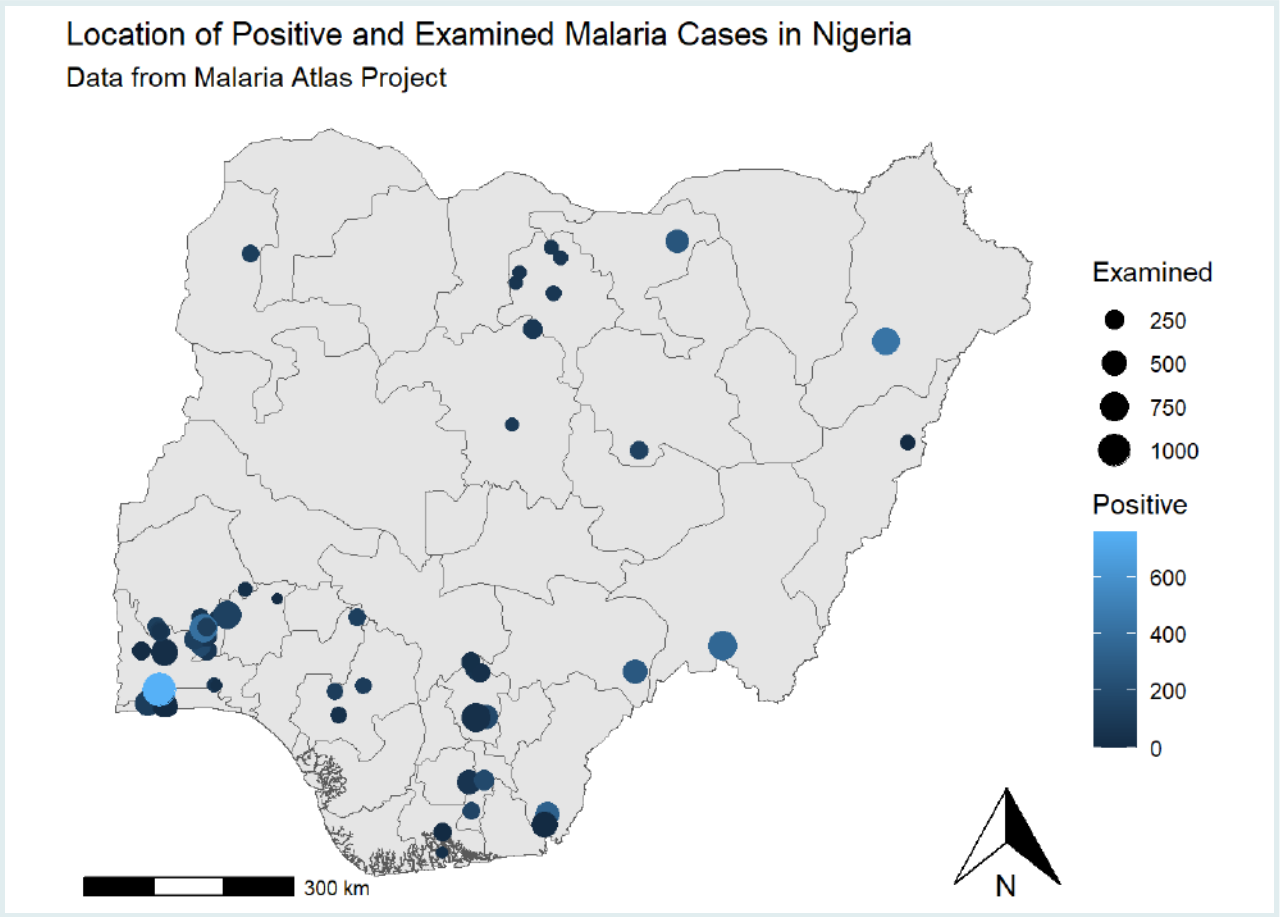
Labeling Point Locations: Exploring Malaria Positive Rate and Incidence

Mapping and visualizing specific data points on a geographical map can provide crucial insights, especially when dealing with epidemiological data. Let's delve into the code to understand the processes and the visualization we're aiming to achieve:

```
# Data Retrieval
# The malariaAtlas package provides the `getPR` function to access the
        parasite rate (PR).
# PR is an essential indicator of malaria prevalence.

# Fetching data for Nigeria for both malaria species
nigeria_pr <- malariaAtlas::getPR(ISO = "NGA", species = "both") %>%
  # Filtering out records with missing PR values
  filter(!is.na(pr)) %>%
  # Removing any rows with missing longitude or latitude
  drop_na(longitude, latitude) %>%
  # Converting the data into a spatial dataframe to facilitate mapping
  st_as_sf(coords = c("longitude", "latitude"), crs = 4326)
```

This chunk fetches malaria prevalence data for Nigeria. After retrieval, the data is cleansed by filtering out missing values. It's then transformed into a format suitable for geospatial visualization (`sf` object).

```
# Setting up a geospatial visualization using ggplot2

# Starting the plot
ggplot() +
  # Plotting the administrative boundaries of Nigeria
  geom_sf(data = nga_adm1) +
  # Adding points for each testing location
  # The color of each point indicates if the test was positive, and the size
        represents the number of people tested
  geom_sf(aes(size = examined, color = positive), data = nigeria_pr)+
  # Setting titles, axis labels, and legends for the plot
  labs(title = "Location of Positive and Examined Malaria Cases in Nigeria",
       subtitle = "Data from Malaria Atlas Project",
       color = "Positive",
       size = "Examined")+
  # Adding labels for longitude and latitude
  xlab("Longitude")+
  ylab("Latitude")+
  # Incorporating a north arrow to provide orientation
  ggspatial::annotation_north_arrow(location = "br")+
  # Incorporating a scale bar to assist in distance interpretation
  ggspatial::annotation_scale(location = "bl")+
  # Applying a minimalistic theme for visual clarity
  theme_void()
```



RECAP

This code chunk visualizes the malaria data on a map. It highlights regions based on the number of malaria tests and their outcomes. Specific tools from the `ggspatial` package are used to add cartographic elements, ensuring the map is informative.

Practice 3: Reflecting the positive rate by size and color

CHALLENGE

Create a new visualization that represents the positive rate (number of positive cases divided by the number examined) for each location. Modify the size of the points to reflect the positive rate.

Final Thoughts

Geospatial data visualization is more than just plotting data on a map. It's about narrating a story that's rooted in location and space. This lesson delved deep into a

layers-based narrative approach, from the importance of clear annotations to the integration of diverse data types for a richer understanding.

References

- ggplot2: Elegant Graphics for Data Analysis (3e)
- R for Data Science
- Geocomputation with R

WRAP UP!

In this lesson on maps labeling, explored in depth the following points:

- **Annotations & Labels**: Clear annotations transform maps, making complex spatial data easily comprehensible.
- **Advanced Mapping Techniques**: We explored how to integrate continuous indicators, overlay labels, emphasize regions, and determine optimal point placements on maps.
- **Practical Application**: Using a malaria dataset, we demonstrated data preparation, visualization, and interpretation in R.
- **R packages**: Packages like `ggplot2`, `sf`, and `ggspatial` facilitate advanced geospatial visualizations.
- **Highlight with Context**: It's essential to provide a holistic view, even when emphasizing specific areas.

Answer Key

Solution for practice 1

To visualize the distribution of malaria cases in Nigeria for the year 2015, follow the instructions provided in the exercise. Here's the complete solution:

```
# Constructing the choropleth map for 2015 using ggplot2
ggplot(data = malaria3) +
  geom_sf(aes(fill = cases_2015 / 1000)) +  # Updated data column to reflect
        2015
  labs(title = "Nigeria Malaria Distributed Cases in 2015", fill = "Cases per
        1000") +  # Updated title for 2015
  scale_fill_continuous(low = "blue", high = "yellow") +  # Modified color
        scale to blue-to-yellow gradient
  theme_void()
```

Solution for practice 2

To combine the visualization of increase rates with state names, follow the steps below:

```
# Combining visualization of increase rates with state names
ggplot(data = malaria3) +
  # Create a choropleth map with fill color based on the increase rate
  geom_sf(aes(fill = increase_rate), color="white", size = 0.2) +

  # Label each region with its state name
  geom_text(aes(x = centroid_coords[, 1],
                y = centroid_coords[, 2],
                label = state_name), size = 2, check_overlap = TRUE) +

  # Specify the color scale for increase rates
  scale_fill_continuous(name="Increase rate (%)",
                        limits=c(0,70),
                        low = "green",
                        high = "red",
                        breaks=c(0, 20, 40, 60)) +

  # Add a title and legend to the plot
  labs(title = "Nigeria Malaria Increase Rate from 2015 to 2021",
       fill = "Increase Rate (%)") +

  theme_void()  # Apply a minimalistic theme for clarity
```

In this solution, the choropleth map is created based on the increase rate of malaria cases from 2015 to 2021. Each state in Nigeria is labeled by its name, and the color gradient (from green to red) showcases the magnitude of the increase rate. The map is enhanced with a title and a legend to ensure clarity and comprehension.

Solution for practice 3

Create a visualization that represents the positive rate:

```
# Adding a positive rate column
nigeria_pr$positive_rate <- (nigeria_pr$positive / nigeria_pr$examined) * 100

ggplot() +
  geom_sf(data = nga_adm1) +
  geom_sf(aes(size = positive_rate, color = positive_rate), data =
          nigeria_pr)+
  labs(title = "Location and Positive Rate of Malaria Cases in Nigeria",
       subtitle = "Data from Malaria Atlas Project",
       color = "Positive Rate (%)",
       size = "Positive Rate (%)")+
  xlab("Longitude")+
  ylab("Latitude")+
  ggspatial::annotation_north_arrow(location = "br")+
  ggspatial::annotation_scale(location = "bl")+
  theme_void()
```

Contributors

The following team members contributed to this lesson:

IMAD EL BADISY

Data Science Education Officer
Deeply interested in health data

JOY VAZ

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