Visualizing Comparisons and Compositions

Introduction

Welcome to our tutorial on visualizing comparisons and compositions using {ggplot2} in R. Analyzing and comparing categories, as well as visually representing the composition of elements, are fundamental aspects of data visualization in R.

In this lesson, we'll explore a variety of visualization techniques for these two applications!

We will start our journey by examining bar charts, a foundational element in data visualization. Bar charts offer exceptional versatility, allowing straightforward comparisons across diverse categories. However, when our focus shifts to understanding proportions or compositions, pie charts and donut charts come into play. These charts prove invaluable in visually representing data where the cumulative sum of all parts constitutes a whole.

This tutorial aims to equip you with the skills to effectively visualize aggregated data and make comparisons. By the end, you'll have a solid understanding of how to use {ggplot2} functions to visualize and compare groups in your data, enhancing your ability to draw meaningful insights from your datasets. Let's get started!

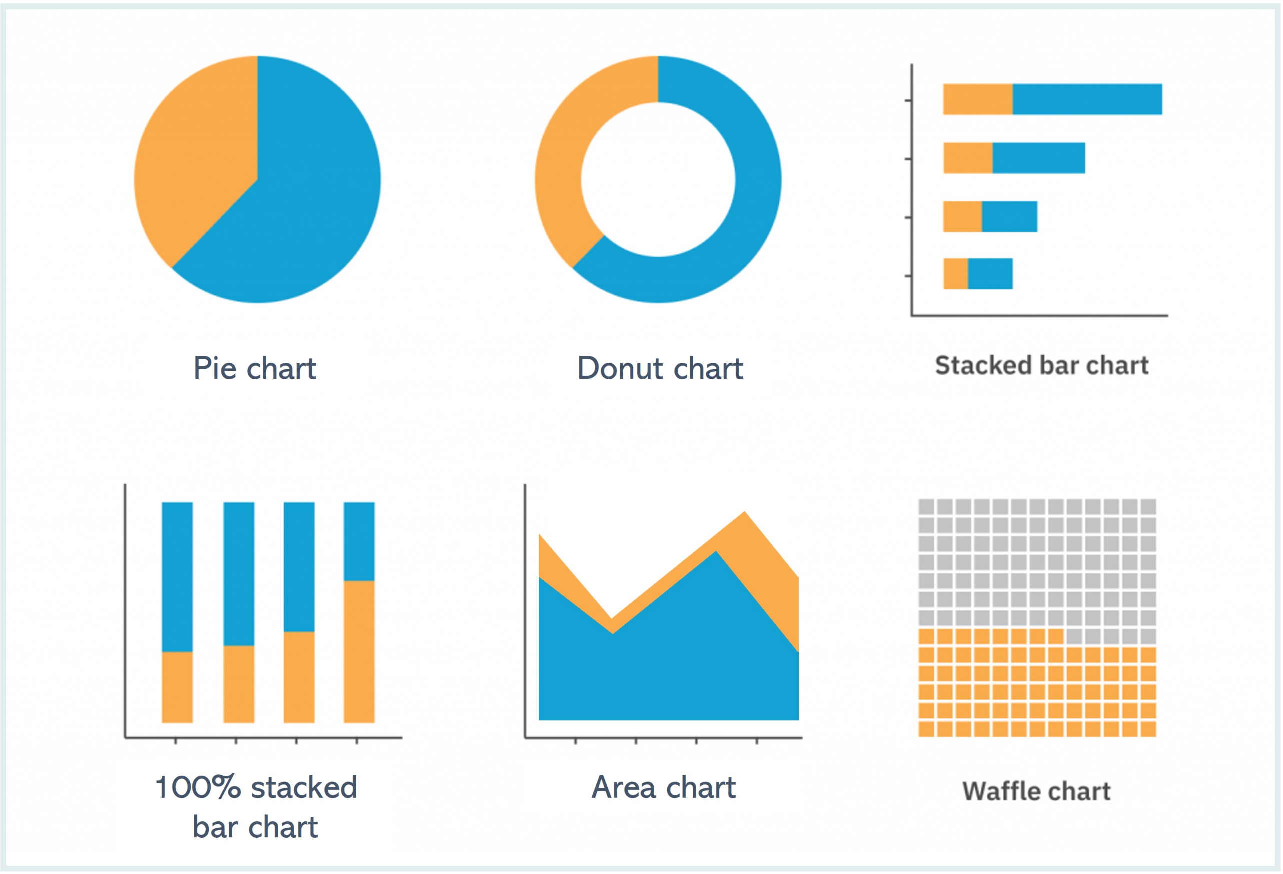
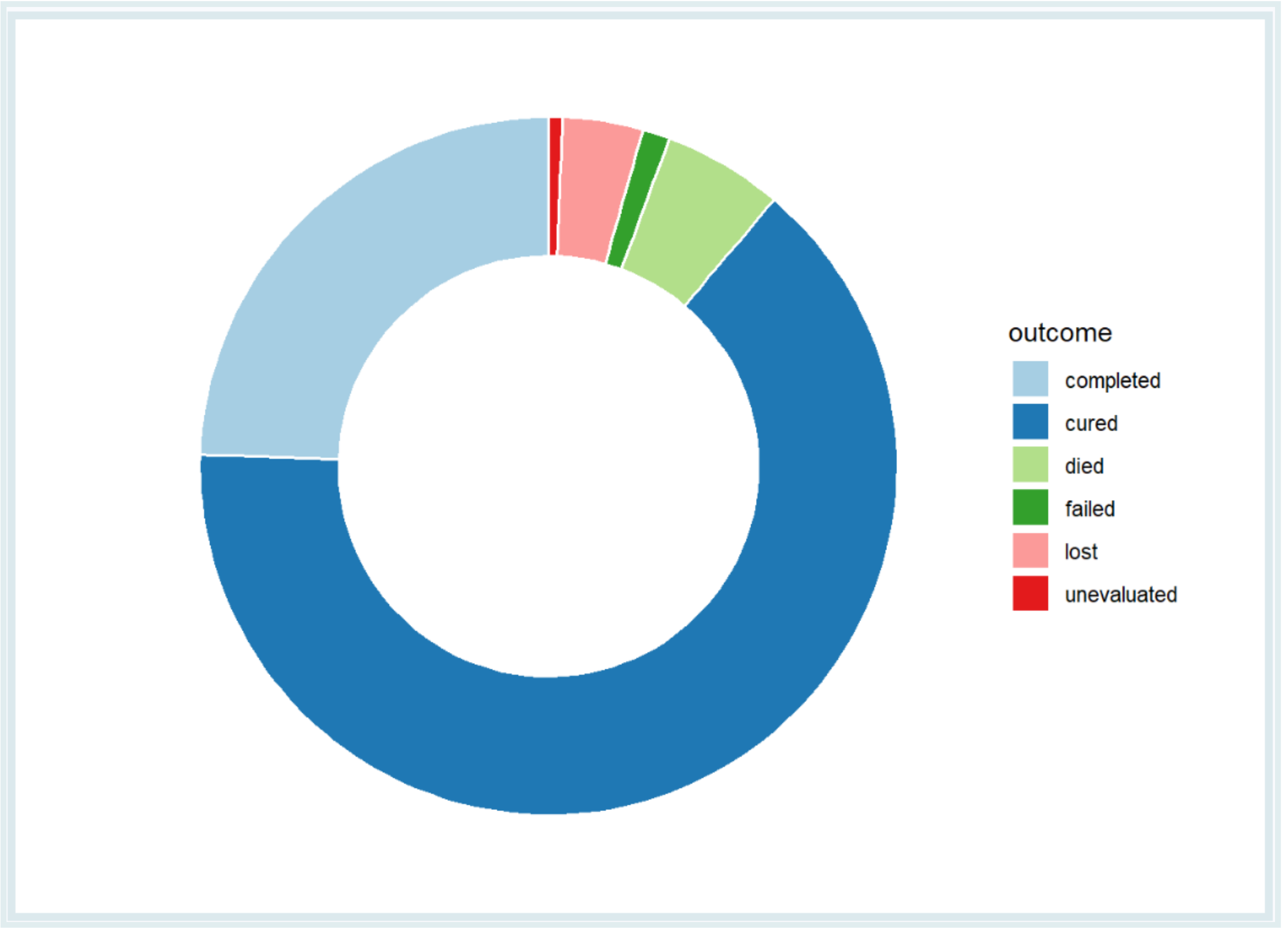


Chart types covered in this lesson.

Learning objectives

1. Understand the difference between visualizing comparisons and visualizing compositions, and recall the appropriate chart types for these two types of analysis.
2. Create and customize bar charts using {ggplot2} for comparing categorical data, with `geom_col()`, `geom_errorbar()`, and `position` adjustments.
3. Create area plots with `geom_area()` for visualizing comparisons.
4. Understand the principles of creating effective pie and donut charts for visualizing compositions.
5. Create and customize pie and donut charts using `coord_polar()` with `geom_col()`.



Load packages

In this lesson we will use the following packages:

- `{tidyverse}` for data wrangling and data visualization
- `{here}` for project-relative file paths

```
pacman::p_load("tidyverse", "here")
```

Data: TB treatment outcomes in Benin

In the realm of public health, data often comes in forms that require us to compare metrics between subgroups, or understand the relative contributions to a total count.

Today we'll be looking at sub-national tuberculosis (TB) data from Benin. The data was provided by WHO and hosted on this DHIS2 dashboard. We will be looking at a subset of this data, with records of treatment outcomes for TB patients in hospitals, from 2015 to 2017.

Let's import the `tb_outcomes` data subset.

```
# Import data from CSV
tb_outcomes <- read_csv(here::here('data/benin_tb.csv'))

# Print data frame
tb_outcomes
```

The data contains records of the number of new and relapse cases started on treatment (`cases`). The case counts are disaggregated by a number of groupings: time period, health facility, treatment outcome, and diagnosis type.

Here are the detailed variable definitions for each column:

1. **`period`** and **`period_date`**: records the time frame for each entry in the dataset. The periods are marked quarterly, starting from the first quarter of 2015 (represented as `2015Q1`) up to the last quarter of 2017 (`2017Q4`). This allows us to track the progression and changes in TB cases over time.

2. **`hospital`**: indicates the specific health facility where the TB cases were recorded. These facilities represent different geographical and administrative areas, each with unique characteristics and capabilities. This subset of the data contains treatment outcome records from five health facilities: St Jean De Dieu, CHPP Akron, CS Abomey-Calavi, Hopital Bethesda, Hopital Savalou, and Hopital St Luc. This information can be used to analyze and compare the prevalence and treatment outcomes of TB across different facilities.

3. **`outcome`:** This column categorizes the TB cases based on the diagnosis type and the stage of their treatment journey. Each variable corresponds to a different aspect of the patient's diagnosis and treatment progress:

   - **`completed`**: started on treatment and the respective outcome is indicated as completed.

   - **`cured`**: started on treatment and for whom the respective outcome is indicated as cured (and backed by at least two clear sputum smear results)

    - `died`: Represents the TB cases that resulted in the death of the patient during treatment, including both bacteriologically confirmed and clinically diagnosed cases. died while under treatment

    - `failed`: These are the cases where treatment failed, which is confirmed for bacteriologically tested patients and observed in clinically diagnosed cases.

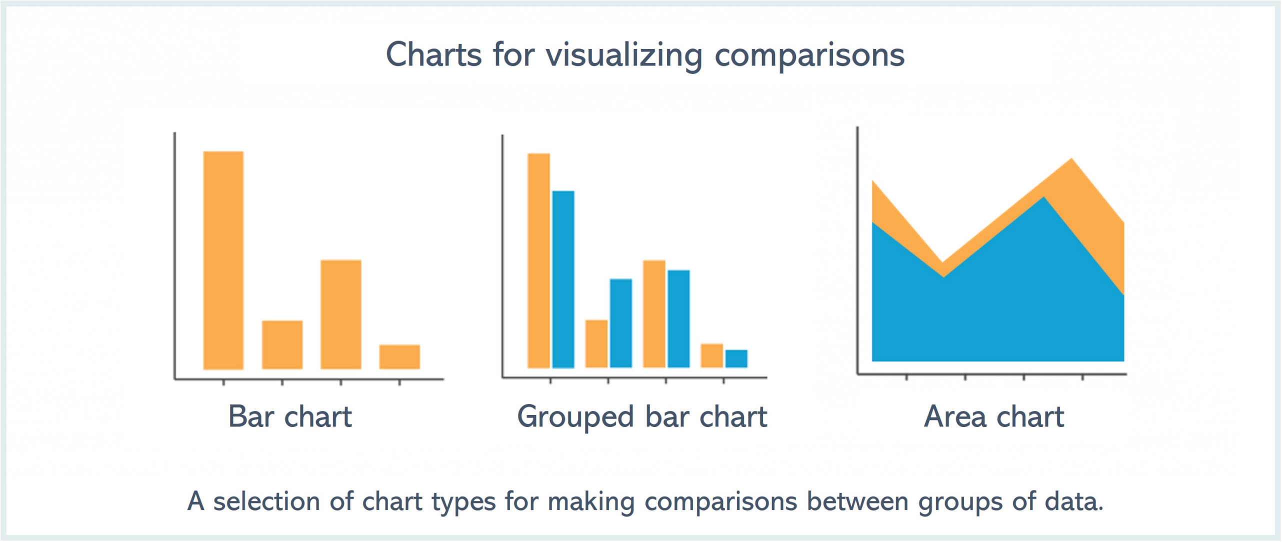
    - `unevaluated`: These are the TB cases that started treatment but do not have an evaluated treatment outcome available. This applies to both bacteriologically confirmed and clinically diagnosed cases.

4. `diagnosis_type`: This column categorizes the TB cases based on the method of diagnosis. There are two types of diagnosis included in this dataset:
    - `bacteriological`: These are the cases where the presence of TB bacteria is confirmed through bacteriological examination methods, such as sputum smear microscopy or culture methods.
    - `clinical`: These are the cases where TB diagnosis is made based on clinical signs and symptoms, without bacteriological confirmation. This usually happens when bacteriological tests are either unavailable or inconclusive, and the patient presents TB symptoms.

5. `cases`: the number of new or relapse cases started on treatment. This allows for quantitative analysis of the TB cases, such as the total number of new cases over a specific period or the number of cases that completed treatment in a particular health facility.

Visualizing comparisons

Visualizations allow you to ask questions of your data, and the type of question you want to ask will determine the chart type. A ton of questions you might ask about your data will hinge on a comparison between values. Which region was responsible for the most cases or deaths? What year had the best treatment outcomes? These questions are answered quickly by gathering and comparing different values in your data.


Charts for visualizing comparisons
Bar chart
Grouped bar chart
Area chart
A selection of chart types for making comparisons between groups of data.

Bar charts

One of the most common chart types are bar charts, and for good reason! They are often the most efficient and effective way of conveying counts per group or category so that comparisons can be made between the bars.

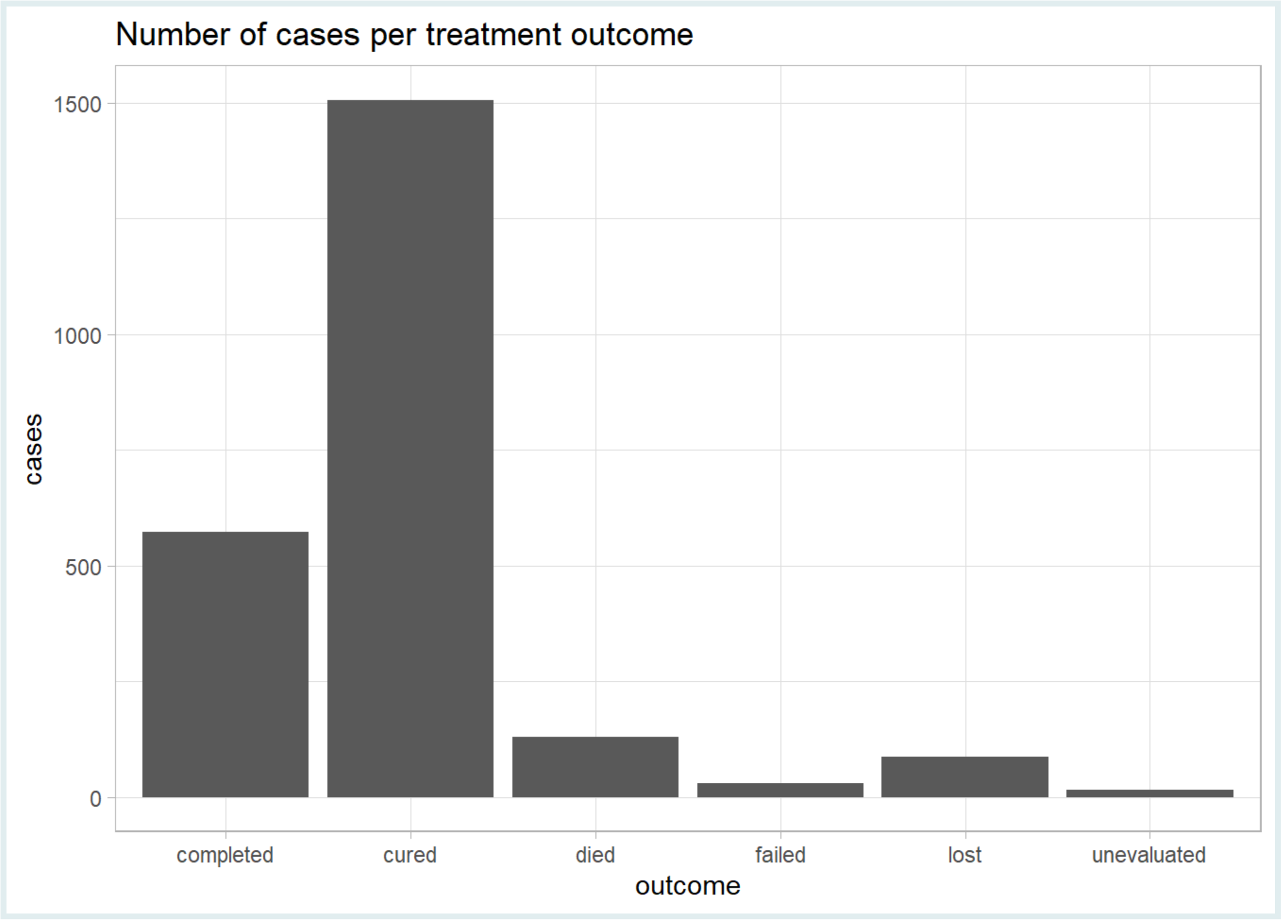
They are especially effective when the categories are ordinal (i.e. contain an inherent order; e.g., age groups), or time based (e.g., months of the year).

But when should we consider choosing a bar/column chart? Bar charts are ideally used when dealing with data that can be **grouped into categories** and when wanting to make **comparisons** between these categories.

When using {ggplot2}, we can use the `geom_col()` function to create a bar plot of a categorical variable against a numerical variable.

Let's exemplify this by visualizing the *Number of cases per treatment outcomes* in the `tb_outcomes` dataset:

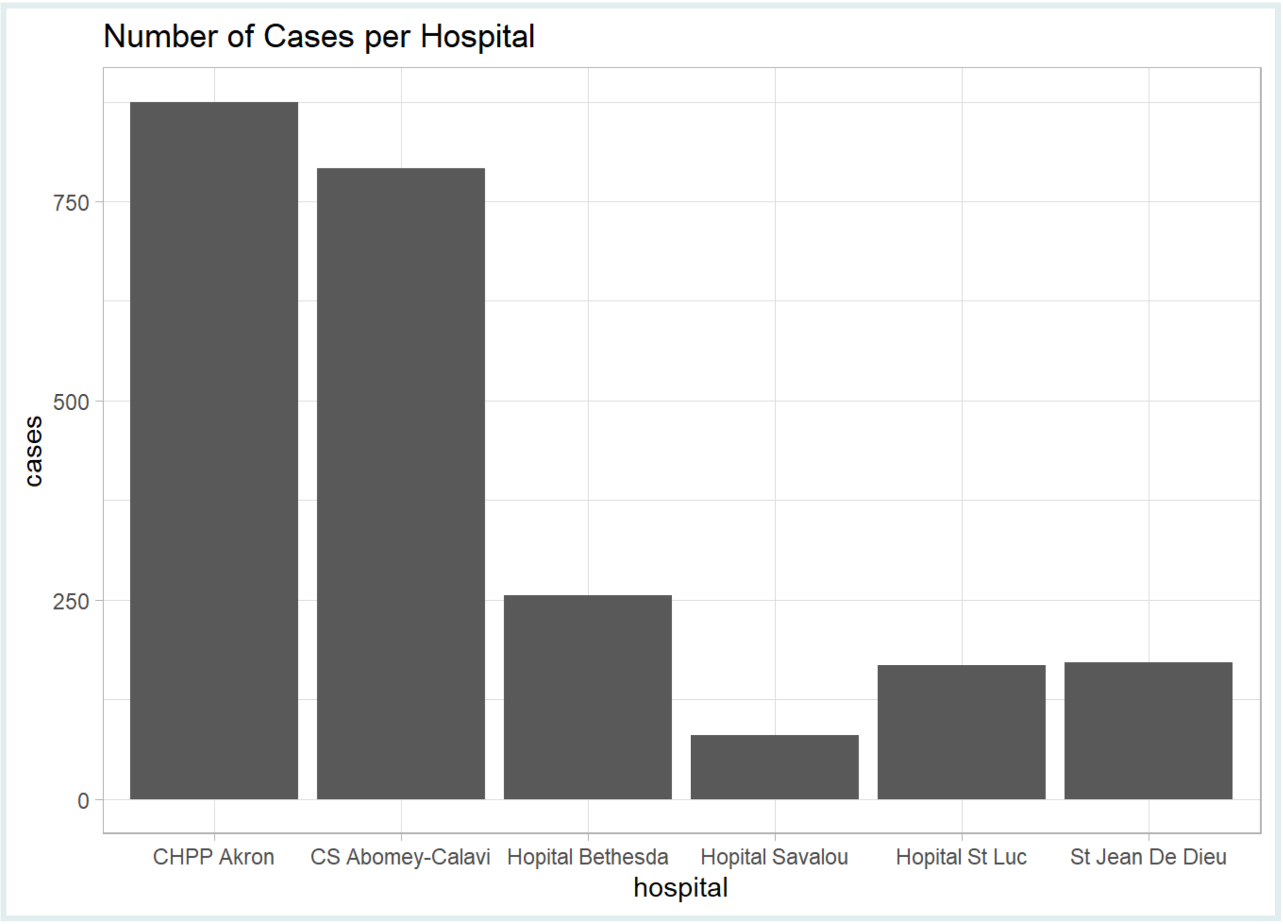
```
# Basic bar plot example 1: Frequency of treatment outcomes
tb_outcomes %>%
  # Pass the data to ggplot as a basis for creating the visualization
  ggplot(
    # Specify that the x and y axis variables
    aes(x = outcome, y = cases)) +
  #  geom_col() creates a bar plot
  geom_col() +
  labs(title = "Number of cases per treatment outcome")
```



As we can see from the graph above, `geom_col()` has automatically tallied the total number of cases per outcomes, across all periods, hospitals, and diagnosis types.

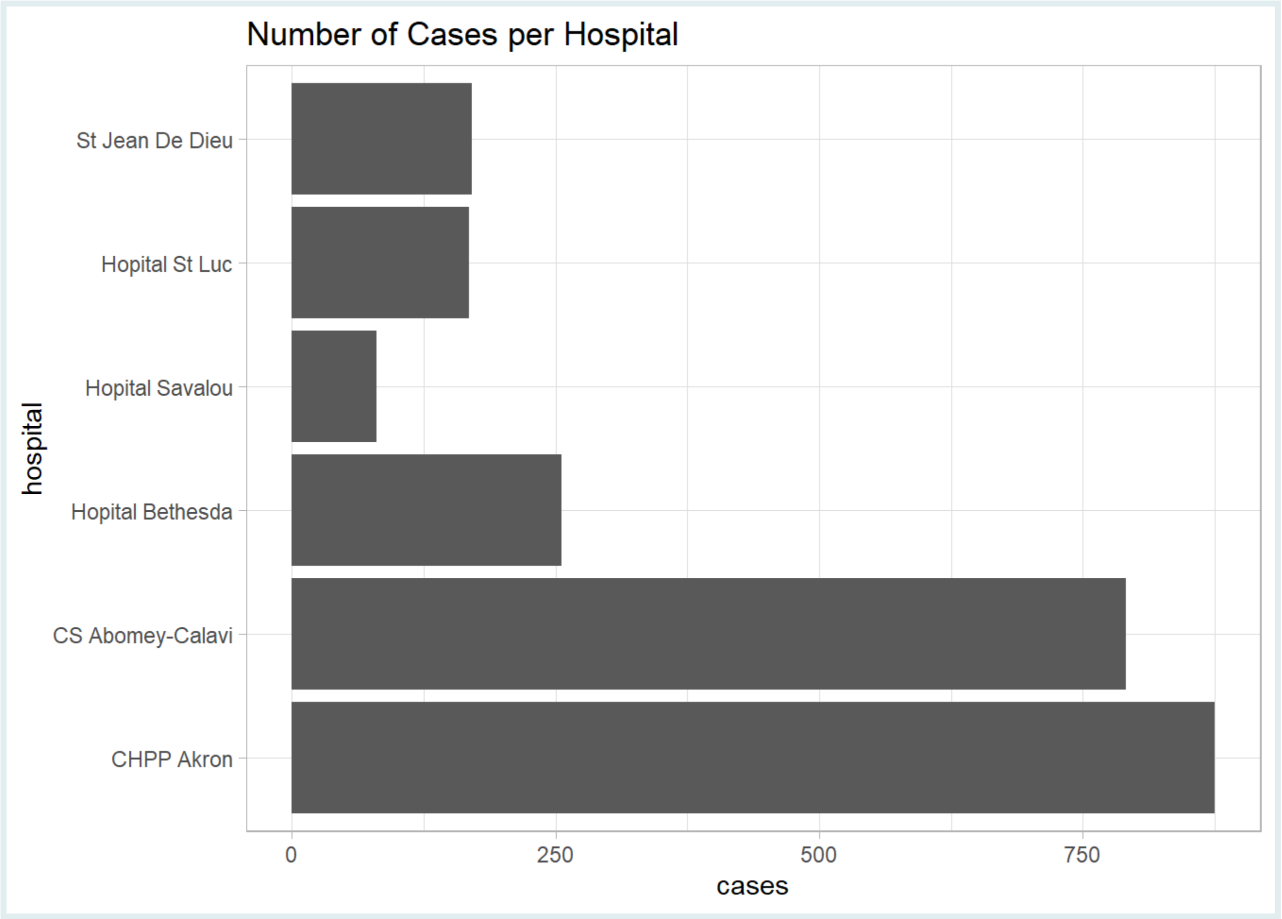
We can also change the x-axis variable to any other categorical variable in the data as below:

```
# Basic bar plot example 2: Case counts per hospital
tb_outcomes %>%
  ggplot(aes(x = hospital, y = cases)) +
  geom_col() +
  labs(title = "Number of Cases per Hospital")
```



In order to properly visualize the various categories, we can generate a horizontal bar plot, by integrating the `coord_flip()` function into our previous code.

```
# Basic bar plot example 3: Horizontal bars
tb_outcomes %>%
  ggplot(aes(x = hospital, y = cases)) +
  geom_col() +
  labs(title = "Number of Cases per Hospital") +
  coord_flip()
```



SIDE NOTE

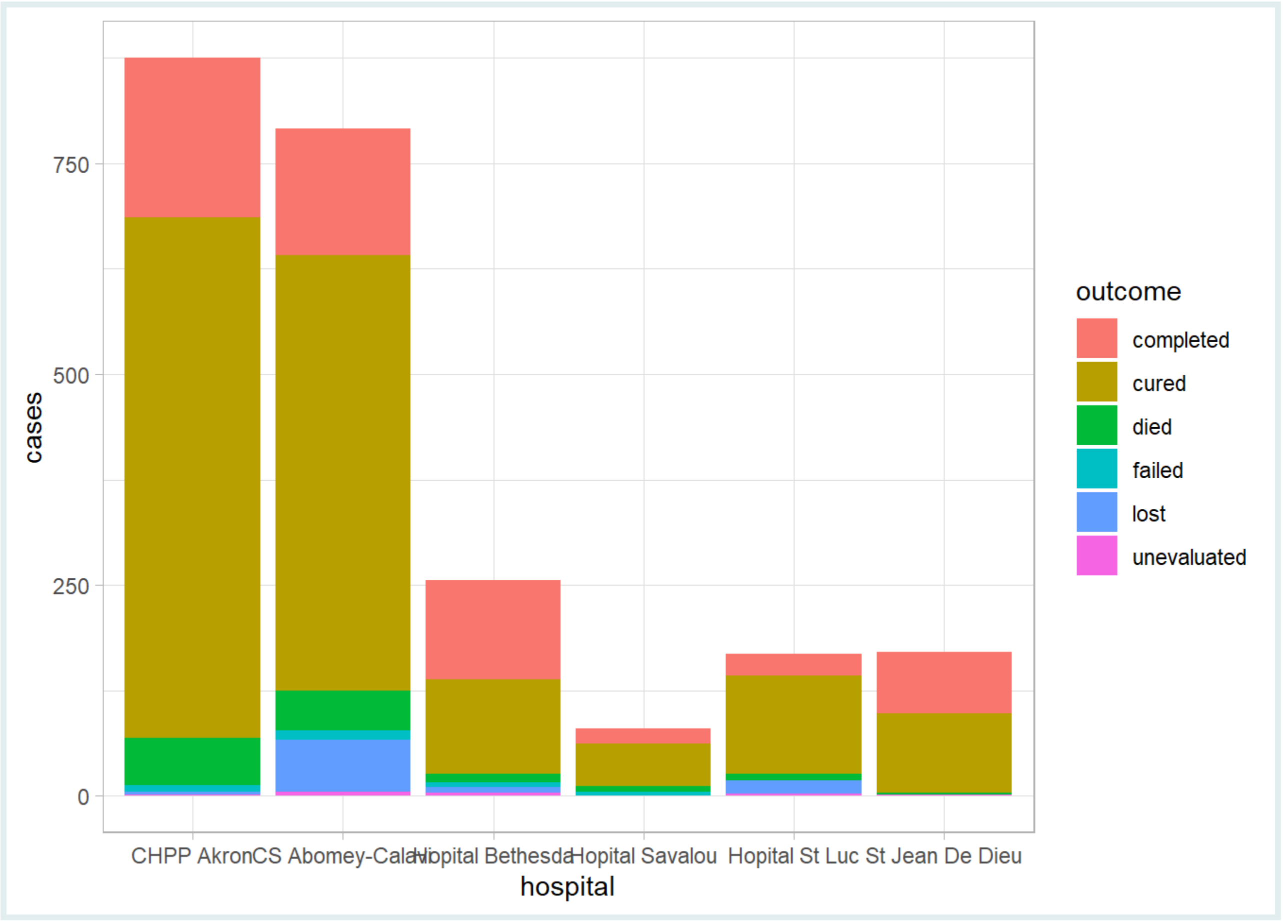
We will use `coord_*()` functions later in the lesson to create circular plots.

Stacked bar charts

While the previous bar charts depicted case distribution across a *single categorical variable*, we can elevate our understanding by introducing a *second categorical variable* and creating **stacked bar charts**.

This can be done in `ggplot()` by setting fill color to a categorical variable:

```
# Stacked bar plot:
tb_outcomes %>%
  ggplot(
    # Fill color of bars by the 'outcome' variable
    aes(x = hospital,
        y = cases,
        fill = outcome)) +
  geom_col()
```

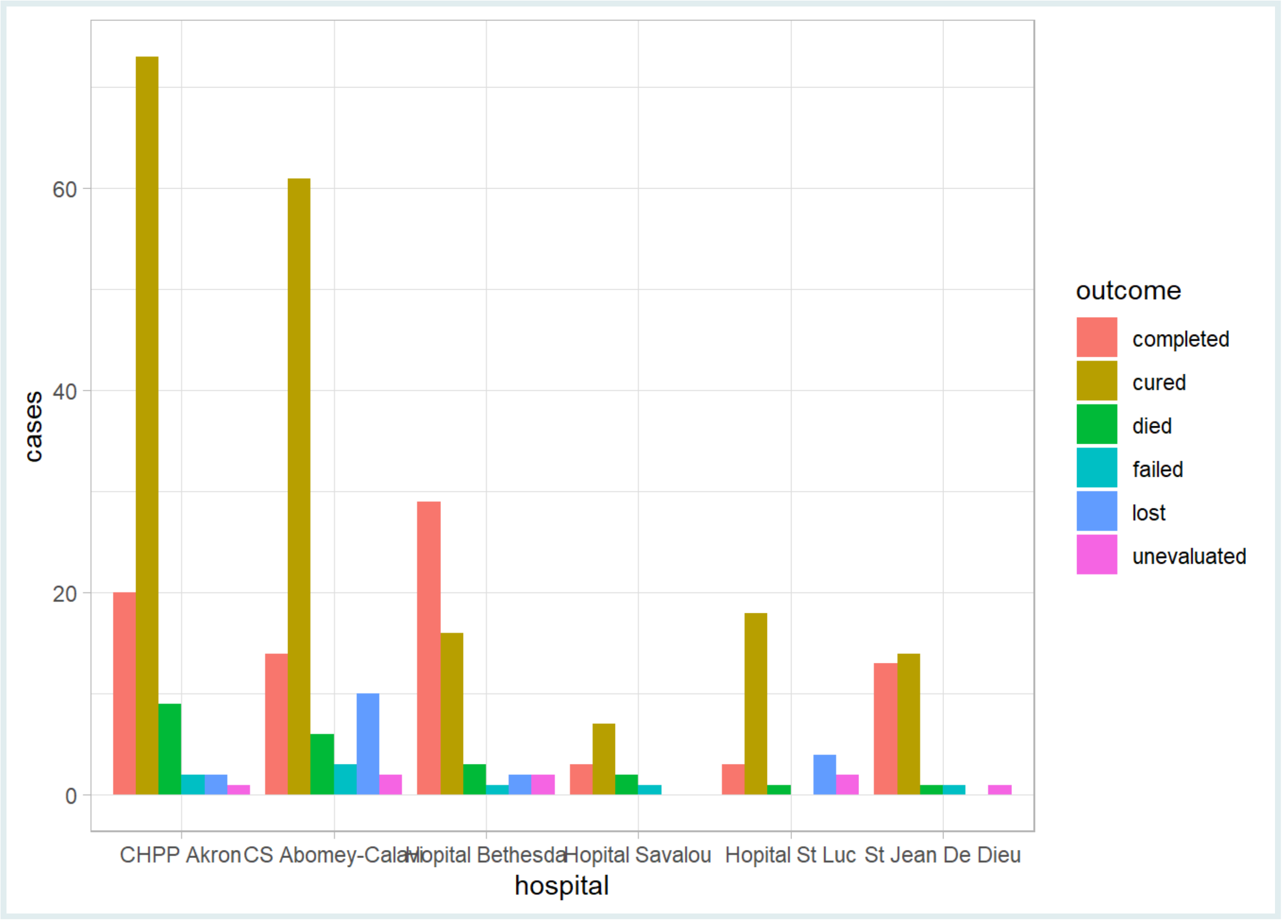


As seen above, stacked plots maintain our *primary categories* on the axis while visually **segregating contributions** from different subgroups by splitting the bars into smaller segments.

Grouped bar charts

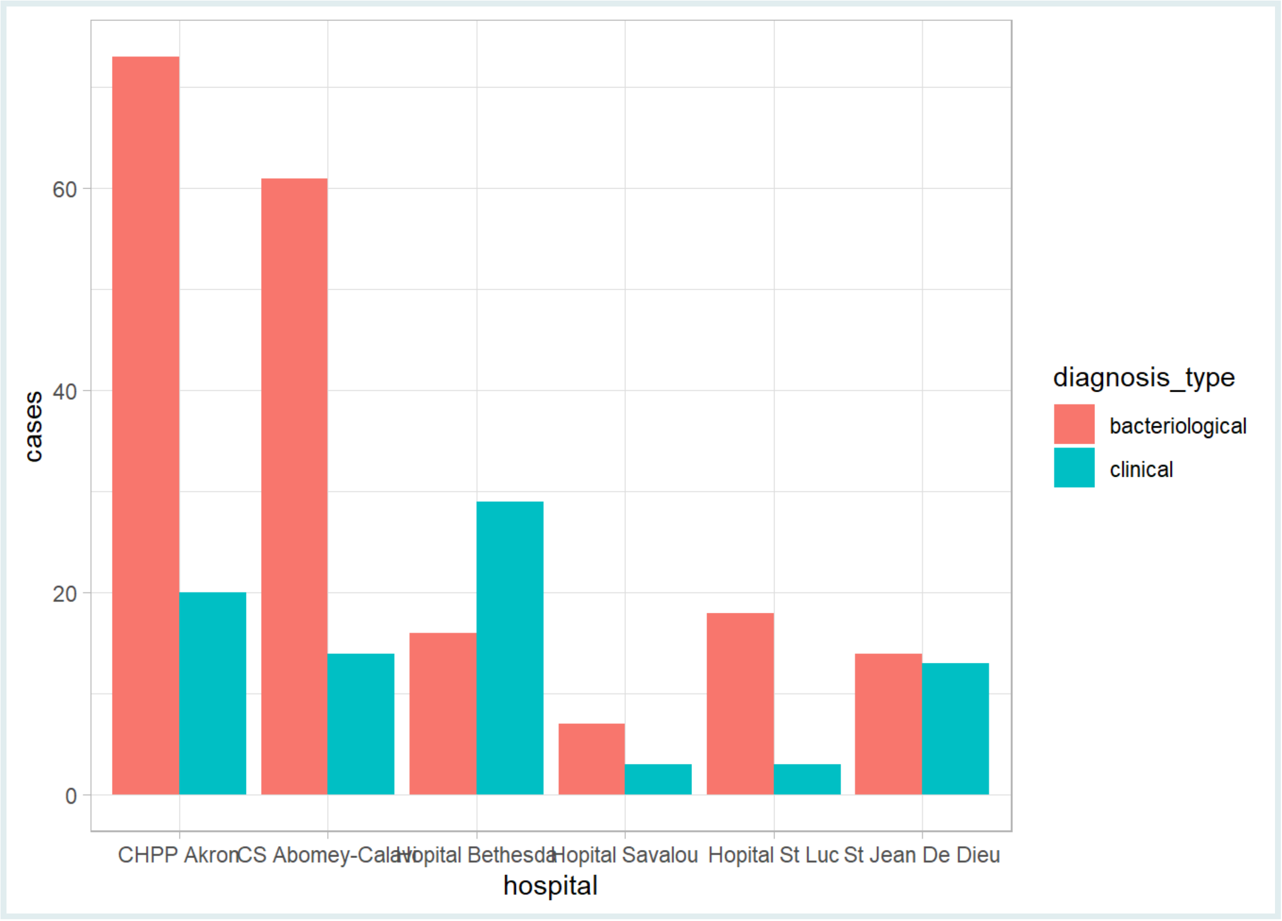
- Grouped bar plots provide a side-by-side representation of subgroups within each main category.
- We can set the `position` argument to `"dodge"` in `geom_col()` to display bars side by side:

```
# Grouped bar plot:
tb_outcomes %>%
  ggplot(
    aes(x = hospital,
        y = cases,
        fill = outcome)) +
  # Add position argument for side-by-side bars
  geom_col(position = "dodge")
```



- Grouped bar charts are not ideal when there are too many groups.
- We can try this again but with a different grouping variable that has fewer categories:

```
# Grouped bar plot: split into 2 bars
tb_outcomes %>%
  ggplot(
    # Fill color of bars by the 'diagnosis_type'
    aes(x = hospital,
        y = cases,
        fill = diagnosis_type)) +
  geom_col(position = "dodge")
```



PRACTICE (in RMD)

Question 1: Basic bar plot

Write the adequate code that generates a basic bar chart of the number of `cases` per quarter with `period_date` on the x axis

```
# PQ1 answer:
tb_outcomes %>%
  ggplot(aes(------, ------)) +
  geom_col()
```

```
## Error: <text>:3:20: unexpected ','
## 2: tb_outcomes %>%
## 3:   ggplot(aes(------,
##                        ^
```

Question 2: Stacked bar plot

PRACTICE

(in RMD)

Create a stacked bar chart to display treatment outcomes over different time periods

```
# PQ2 answer:
tb_outcomes %>%
  ggplot(
    aes(-----, ------, ------)) +
  geom_col()
```

```
## Error: <text>:4:14: unexpected ','
## 3:   ggplot(
## 4:     aes(-----,
##                  ^
```

Adding error bars

Visualizing data with error bars allows for a clearer understanding of the variability or uncertainty inherent in the dataset. Error bars can indicate the reliability of a mean score or an individual data point, providing context to the plotted values.

To implement error bars in {ggplot2}, we use the `geom_errorbar()` function. This requires a value for the range of your error, typically defined by the standard deviation, standard error, or confidence intervals.

Here's an example of how to add error bars to our grouped `geom_col()` plot above.

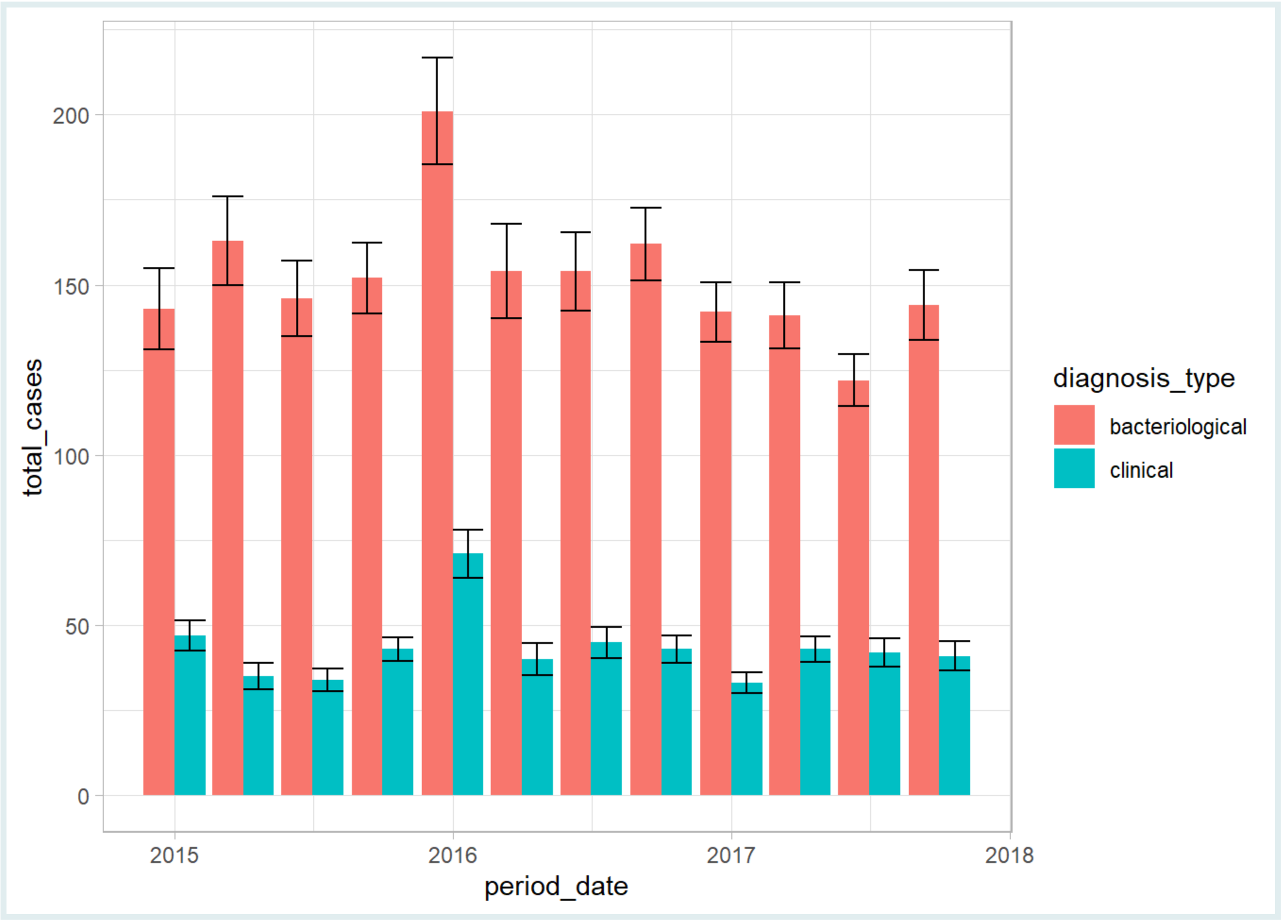
First, let's create the necessary summary data since we need to have some kind of error measurement. In our case we will compute the standard deviation:

```
hosp_dx_error <- tb_outcomes %>%
  group_by(period_date, diagnosis_type) %>%
  summarise(
    total_cases = sum(cases, na.rm = T),
    error = sd(cases, na.rm = T))

hosp_dx_error
```

```
## # A tibble: 24 × 4
## # Groups:   period_date [12]
##    period_date diagnosis_type  total_cases error
##    <date>      <chr>                 <dbl> <dbl>
##  1 2015-01-01  bacteriological         143 11.9
##  2 2015-01-01  clinical                 47  4.40
##  3 2015-04-01  bacteriological         163 13.0
##  4 2015-04-01  clinical                 35  3.84
##  5 2015-07-01  bacteriological         146 11.2
##  6 2015-07-01  clinical                 34  3.33
##   7 2015-10-01  bacteriological         152 10.4
##   8 2015-10-01  clinical                 43  3.55
##   9 2016-01-01  bacteriological         201 15.7
##  10 2016-01-01  clinical                 71  7.01
## # ℹ 14 more rows
```

```
# Recreate grouped bar chart and add error bars
hosp_dx_error %>%
  ggplot(
    aes(x = period_date,
        y = total_cases,
        fill = diagnosis_type)) +
  geom_col(position = "dodge") +  # Dodge the bars
  #  geom_errorbar() adds error bars
  geom_errorbar(
    # Specify upper and lower limits of the error bars
    aes(ymin = total_cases - error, ymax = total_cases + error),
    position = "dodge"  # Dodge the error bars to align them with side-by-side
        bars
  )
```

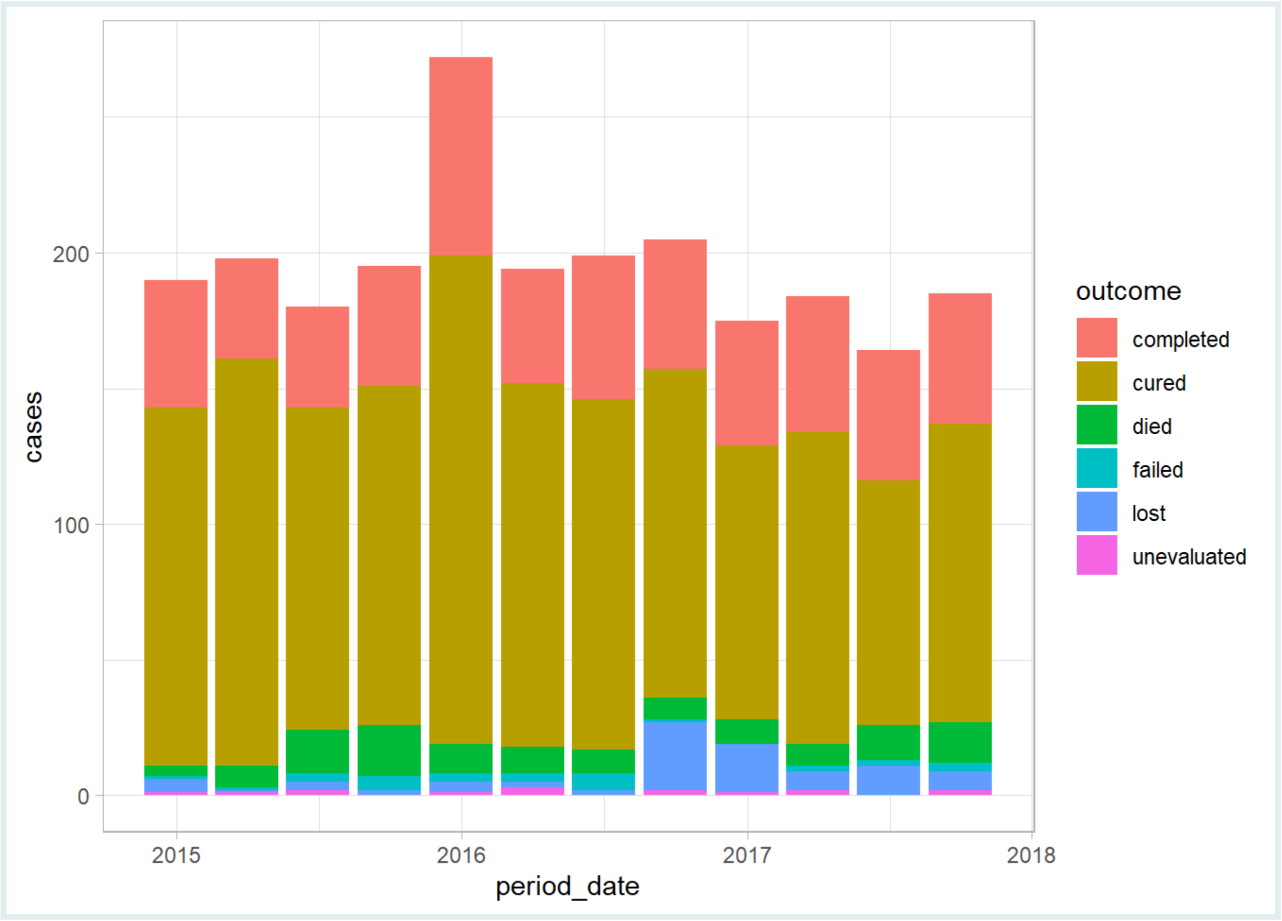


Area charts

In this section, we'll be exploring area plots with `geom_area()`. These plots are especially advantageous for visualizing a range of data values or the progression of data across time, making them ideal for use cases like tracking the number of new treatment cases per quarter.

Let's start by visualizing this distribution with a stacked bar chart:

```r
tb_outcomes %>%
  ggplot(
    aes(x = period_date,
        y = cases,
        fill = outcome)) +
  geom_col()
```



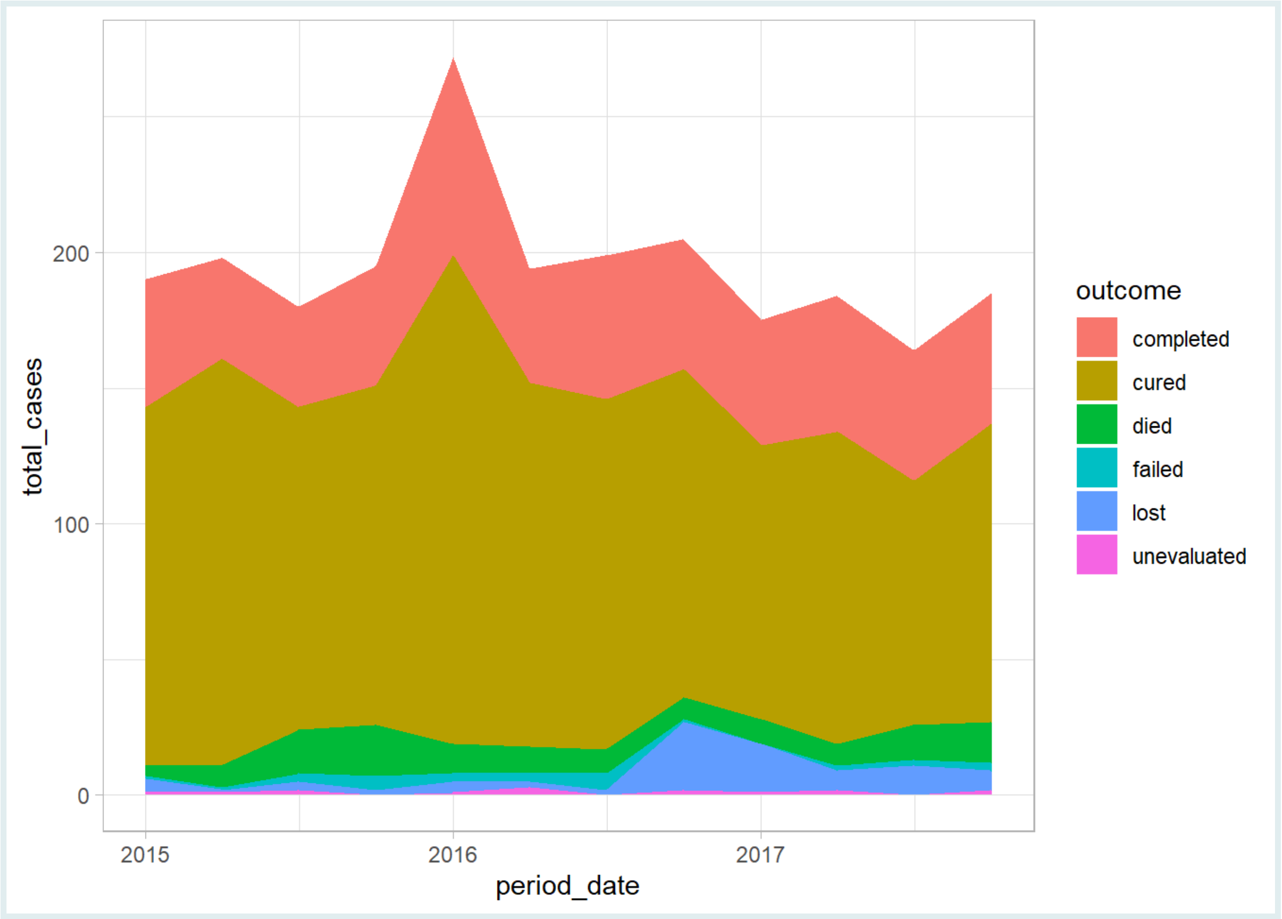
- Unlike `geom_col()`, `geom_area()` will not automatically compute the sums for each category.
- We need to group summarize our data so we have a data frame with the sum of cases for every treatment outcome per quarter:

```r
# Create summary data frame
outcome_by_period <- tb_outcomes %>%
  group_by(period_date, outcome) %>%
  summarise(
    total_cases = sum(cases, na.rm = T))

outcome_by_period
```

```
## # A tibble: 72 × 3
## # Groups:   period_date [12]
##    period_date outcome     total_cases
##    <date>      <chr>             <dbl>
##  1 2015-01-01  completed            47
##  2 2015-01-01  cured               132
##  3 2015-01-01  died                  4
##  4 2015-01-01  failed                1
##  5 2015-01-01  lost                  5
##  6 2015-01-01  unevaluated           1
##  7 2015-04-01  completed            37
##  8 2015-04-01  cured               150
##  9 2015-04-01  died                  8
## 10 2015-04-01  failed                1
## # ℹ 62 more rows
```

```r
# Area plot of treated cases over time
outcome_by_period %>%
ggplot(
  aes(
    x = period_date,    # Map 'period' to the x-axis
    y = total_cases,     # Map 'value' to the y-axis
    fill = outcome    # Map 'variable_name' to the fill aesthetic
  )) +
  #  geom_area() creates an area plot
  geom_area()
```

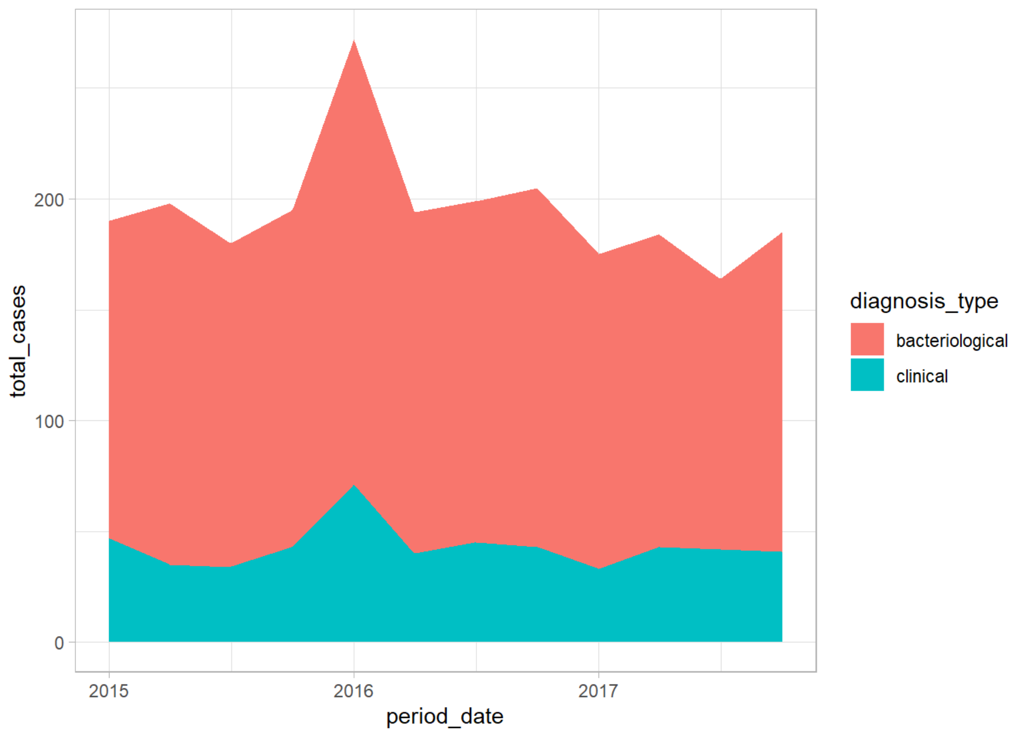


Suppose we now want to ask what outcome contributes most to each section, or track the proportion of cured cases in each period over time. These questions also involve comparisons, but they are subtly different. Instead of comparing total values, we want to compare the relative contribution of those values to a bigger whole as proportions or percentages. In other words, we are interested in looking at the composition, or part-to-whole relationships. In the next section we will learn how to visualize these relationships.

PRACTICE

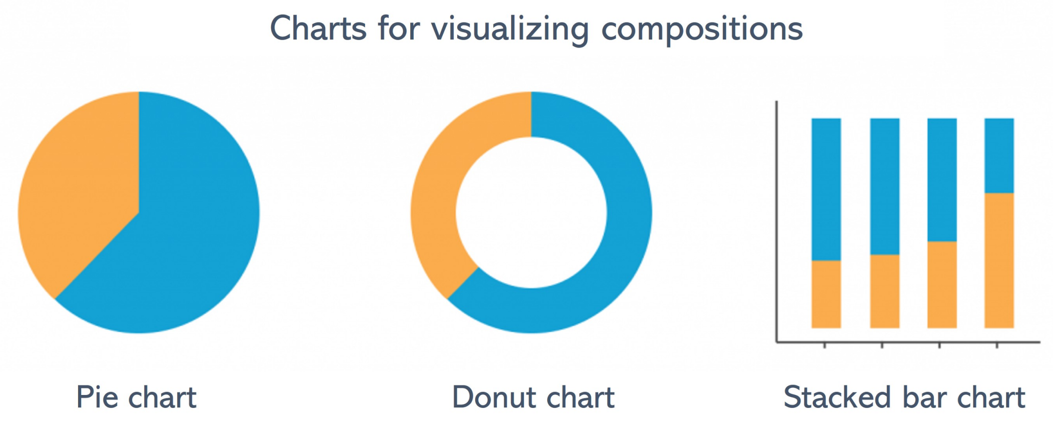
(in RMD)

```
# PQ 3 answer
tb_outcomes %>%
  group_by(period_date, diagnosis_type) %>%
  summarise(
    total_cases = sum(cases, na.rm = T)) %>%
ggplot(
  aes(
    x = period_date,
    y = total_cases,
    fill = diagnosis_type)) +
  geom_area()
```



Visualizing comparisons with normalized bar charts, pie charts, and donut charts

With compositions, we want to show how individual parts make up the whole. We could try to answer these questions using charts types discussed above, but there are many chart types devoted to compositions that do a much better job. These part-to-whole chart types immediately focus our visual attention on the relative importance of every part to the total value in the data.



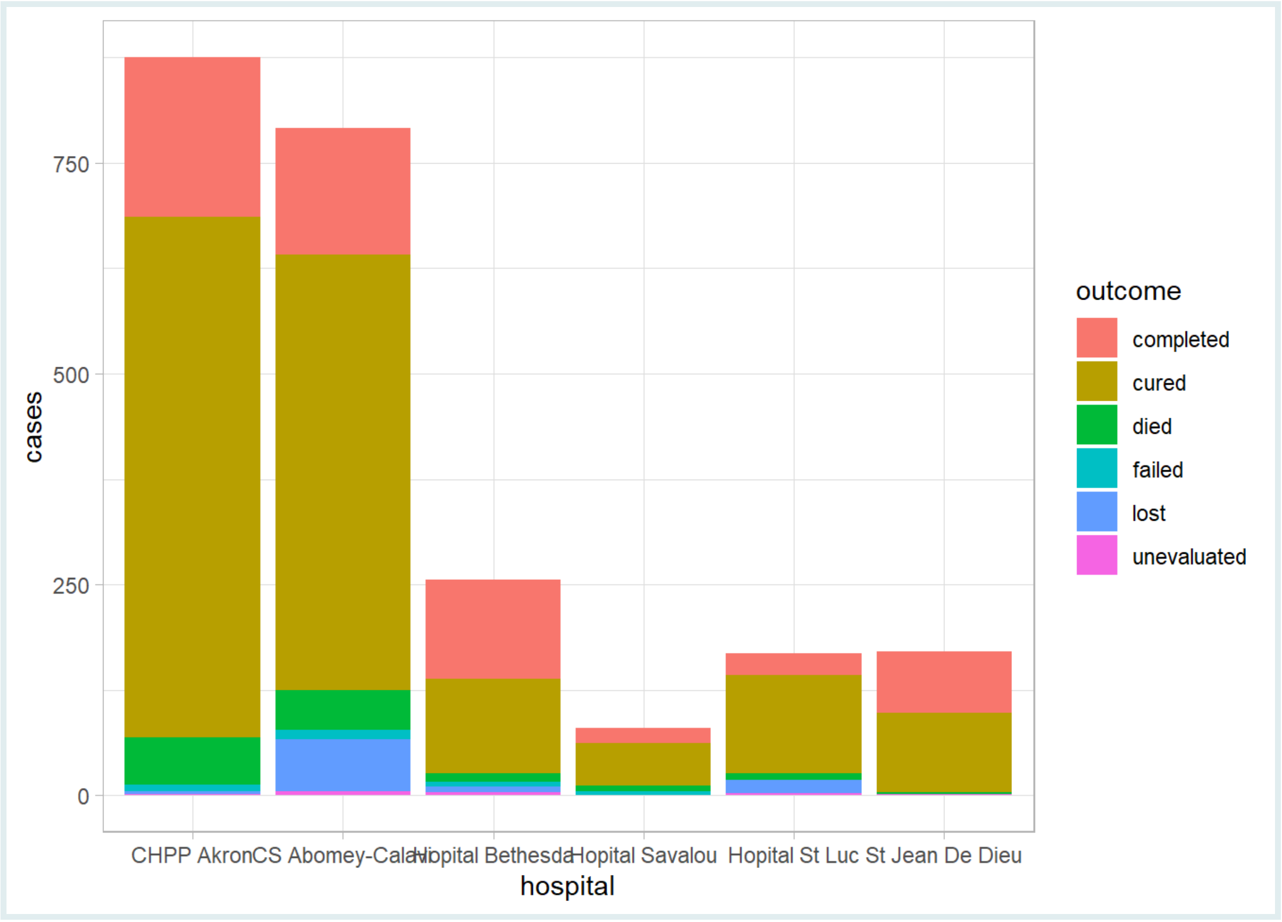
A selection of chart types for visualizing parts-to-whole relationships.

Percent-stacked bar chart

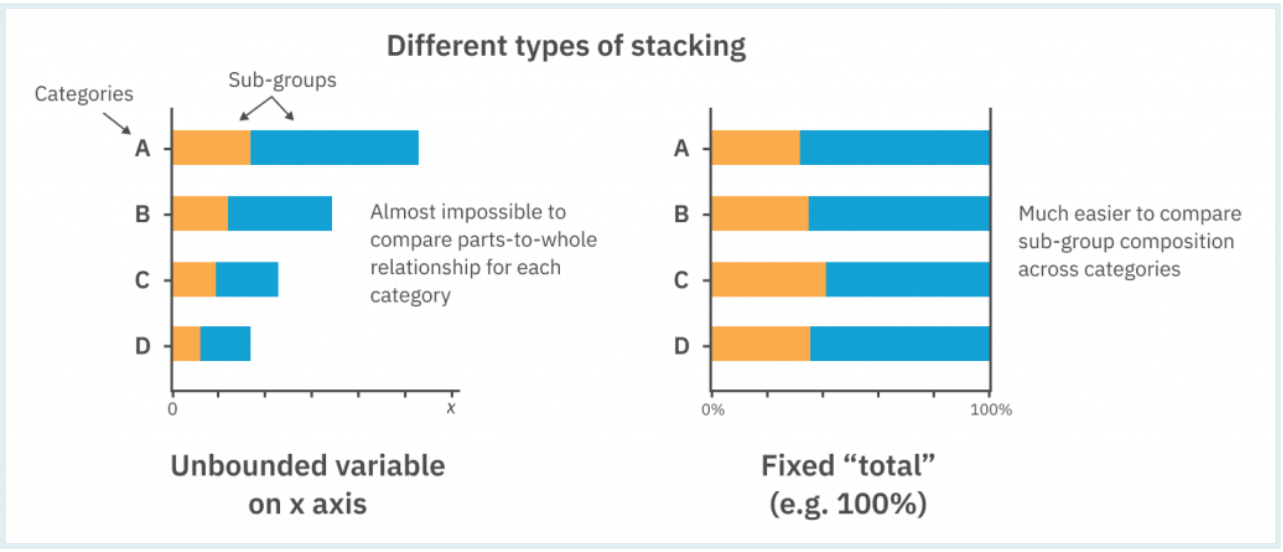
To visualize a composition, or part-to-whole relationship, we need two ingredients: the parts, and the whole.

The stacked bar charts we created earlier do a somewhat okay job of this:

```r
# Regular stacked bar plot
tb_outcomes %>%
  ggplot(
    aes(x = hospital,
        y = cases,
        fill = outcome)) +
  geom_col()
```

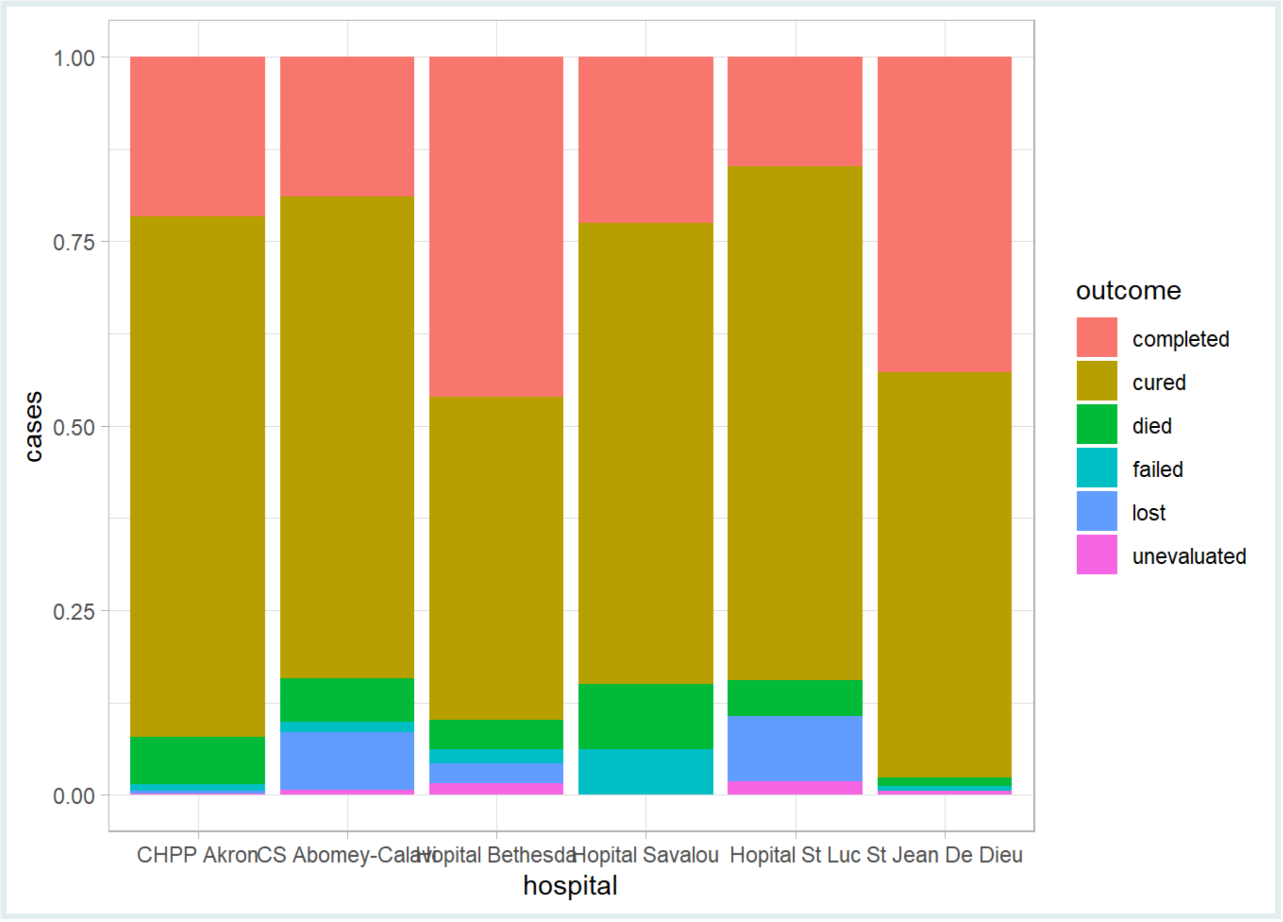


- They show us parts-of-wholes, but all the wholes are different sizes.
- The height of the bars represent the total number of cases, which is different at every location.
- Looking at the *relative* distribution of outcomes would be much easier if every bar were the same size.
- We can do this by creating a 100% stacked bar chart, where the total height of each bar is standardized to the same size, effectively showing proportions rather than counts or absolute values.


Different types of stacking
Categories
Sub-groups
A
B
C
D
0
x
Almost impossible to compare parts-to-whole relationship for each category
Unbounded variable on x axis
A
B
C
D
0%
100%
Much easier to compare sub-group composition across categories
Fixed "total" (e.g. 100%)

This is achieved by setting the `position` argument to `"fill"` in `geom_col()`.

```
# Percent-stacked bar plot
tb_outcomes %>%
  ggplot(
    aes(x = hospital,
        y = cases,
        fill = outcome)) +
  # Add position argument for normalized bars
  geom_col(position = "fill")
```



- All bars are now the same length, meaning all the wholes are now the same size. This now allows us to easily evaluate the contributions of the different parts to the whole.

Circular plots: Pie and Donut charts

In this section, we will delve into circular data visualizations, particularly pie charts and donut plots, to demonstrate categorical data distribution. These types of plots can be quite polarizing in the data visualization community due to their tendency to distort data interpretation. However, when employed judiciously, they can offer an intuitive snapshot of proportions within a dataset.

WATCH OUT

- Delve into donut and pie charts with a note of caution: they're visually enticing but can mislead.
- Recognize that bar plots often surpass these charts in delivering precise data interpretations.
- Understand that our brains prefer comparing lengths (like bars) over angles or areas (like pie slices).
- Remember, as data categories grow, pie and donut charts can get overly crowded and lose clarity.

WATCH OUT

- Acknowledge their inability to effectively display changes over time, unlike the adept bar plot.
- Use pie and donut charts sparingly, with a mindful eye on their potential to obscure data's true story.

Before we can visualize the data, we must first aggregate it to get the total counts for each treatment outcome category, ensuring we have a clear representation of each segment of our dataset.

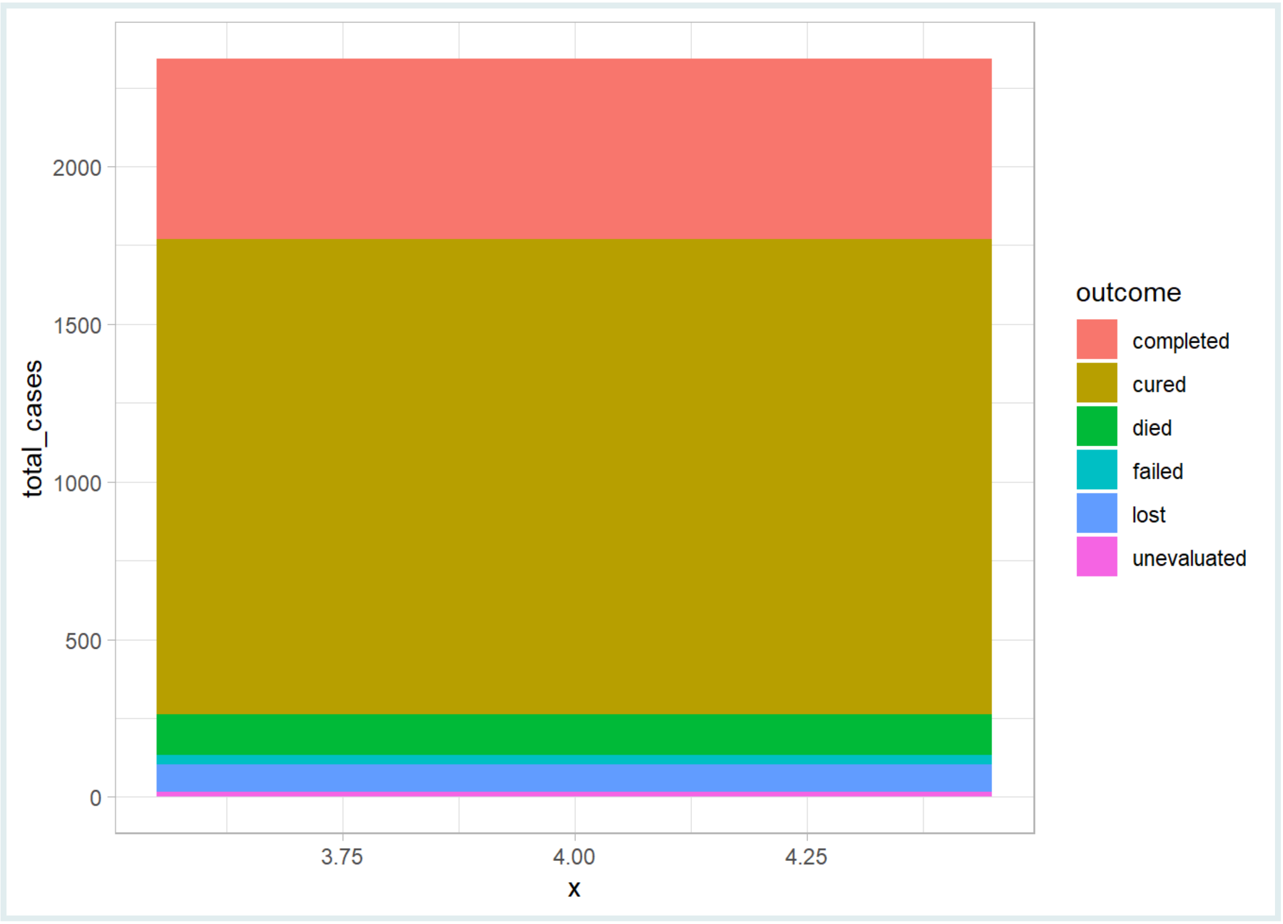
```
outcome_totals <- tb_outcomes %>%
  group_by(outcome) %>%
  summarise(
    total_cases = sum(cases, na.rm = T))

outcome_totals
```

```
## # A tibble: 6 × 2
##   outcome      total_cases
##   <chr>              <dbl>
## 1 completed            573
## 2 cured               1506
## 3 died                 130
## 4 failed                30
## 5 lost                  87
## 6 unevaluated           15
```

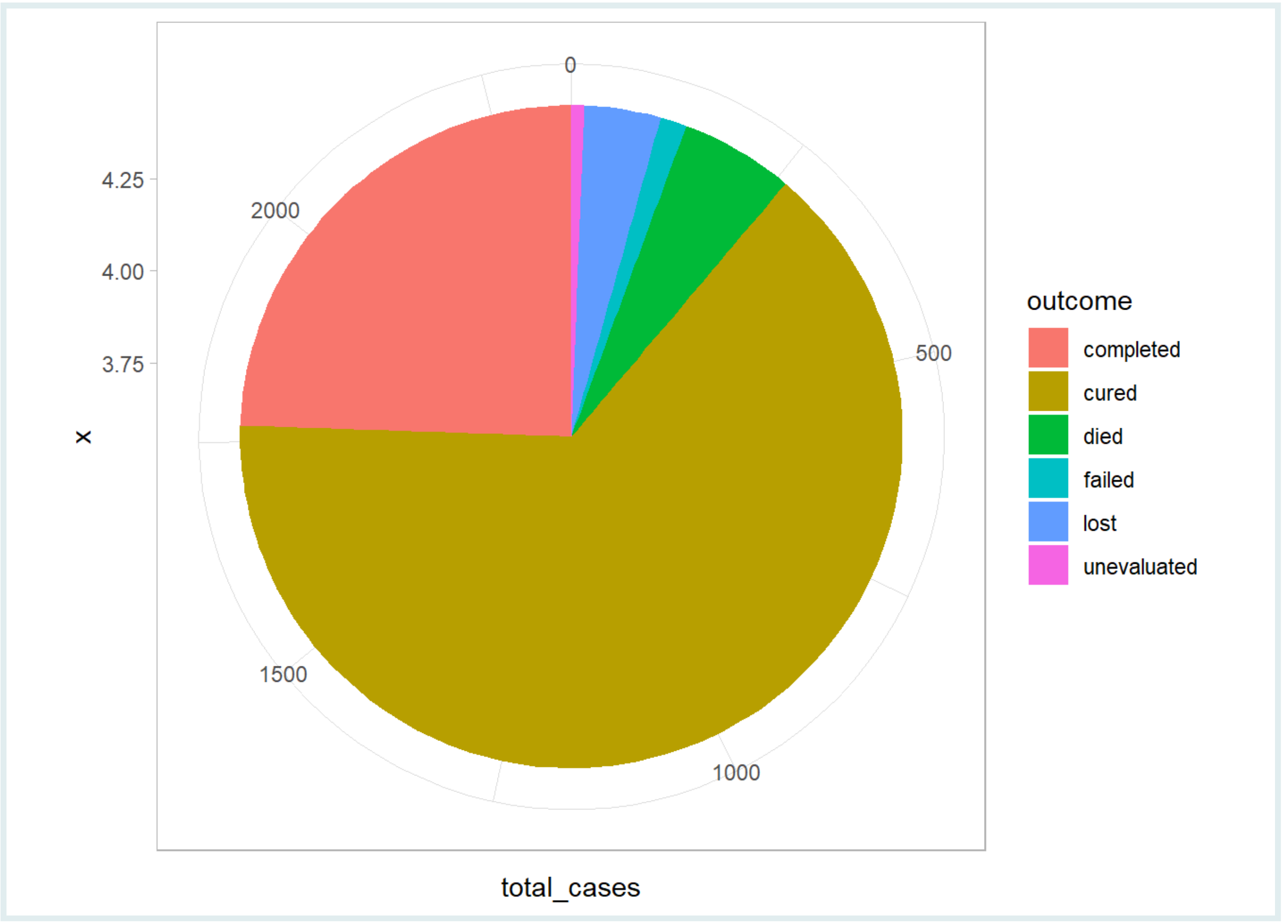
A pie chart is basically a round version of a single 100% stacked bar.

```
# Single-bar chart (precursor to pie chart)
ggplot(outcome_totals,
       aes(x=4, # Set arbitrary x value
           y=total_cases,
           fill=outcome)) +
  geom_col()
```



- In ggplot2, we'll explore how `coord_*()` functions can change a plot's perspective, like tweaking aspect ratios or axis limits.
- We'll transform our plot from linear to polar coordinates using `coord_polar()`, which will shape our data into slices for a pie chart.
- By mapping the `y` aesthetic to angles (using the `theta` argument), we'll collaboratively create a visual that clearly displays the distribution of our categorical data.

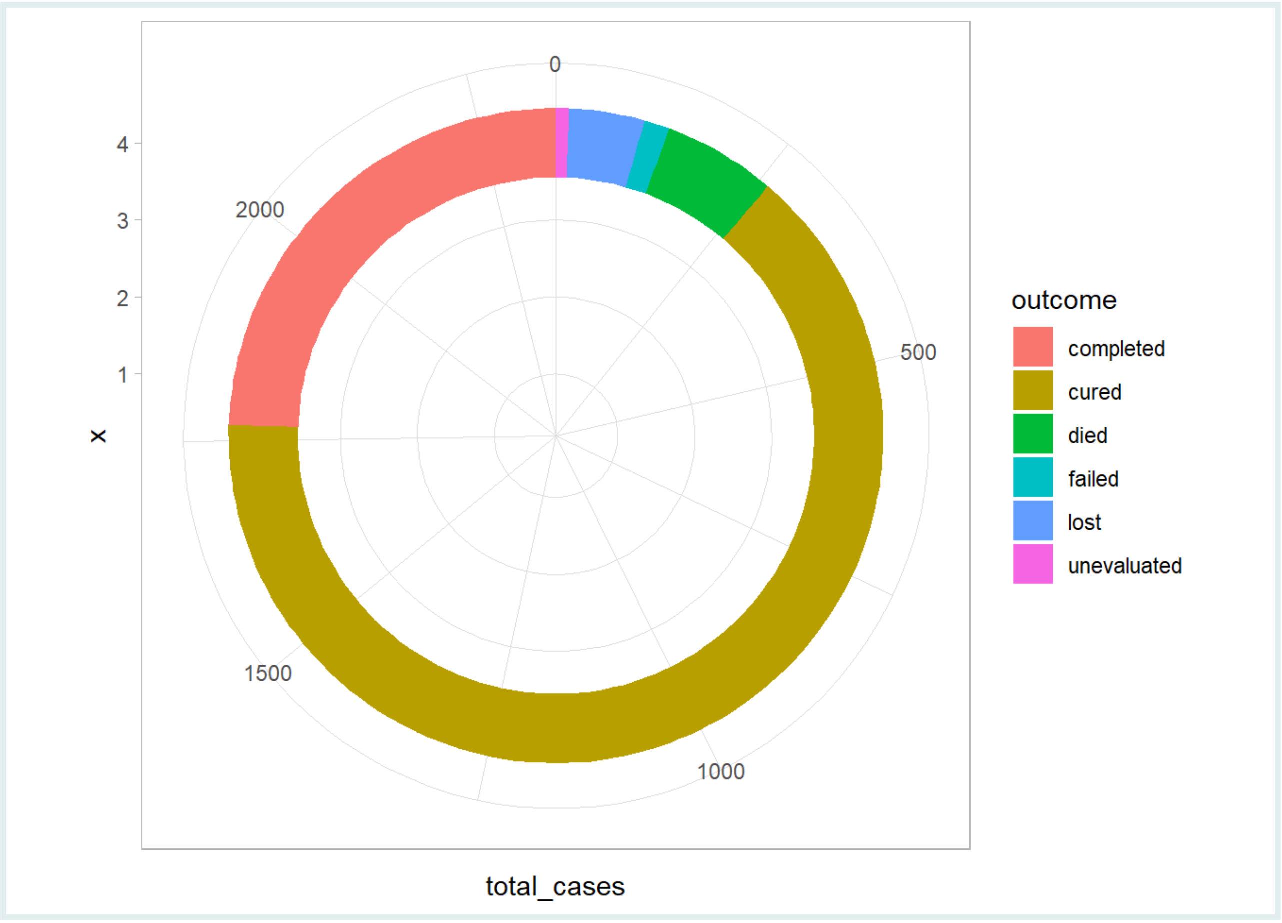
```
# Basic pie chart
ggplot(outcome_totals,
       aes(x=4,
           y=total_cases,
           fill=outcome)) +
  geom_col() +
  coord_polar(theta = "y") # Change y axis to be circular
```



- Donut charts, or ring charts, offer a compelling twist on traditional pie charts and are within our reach using ggplot2 in R.
- We build these charts with a process akin to pie charts, but we add distinctive touches.
- By employing `geom_col()` we lay the groundwork with a bar chart.
- The transformation into a circle comes next with `coord_polar(theta = "y")`.
- With `xlim(c(0.2, 4 + 0.5))`, we're setting the stage for the donut's signature feature: its central void.
- The lower limit of `0.2` carves out the space for the donut's hole, while `4.5` on the upper end ensures every category has its place.

A donut or ring chart can be created using `geom_col()`, `coord_polar(theta = "y")` and setting a x-axis limit with `xlim` like this:

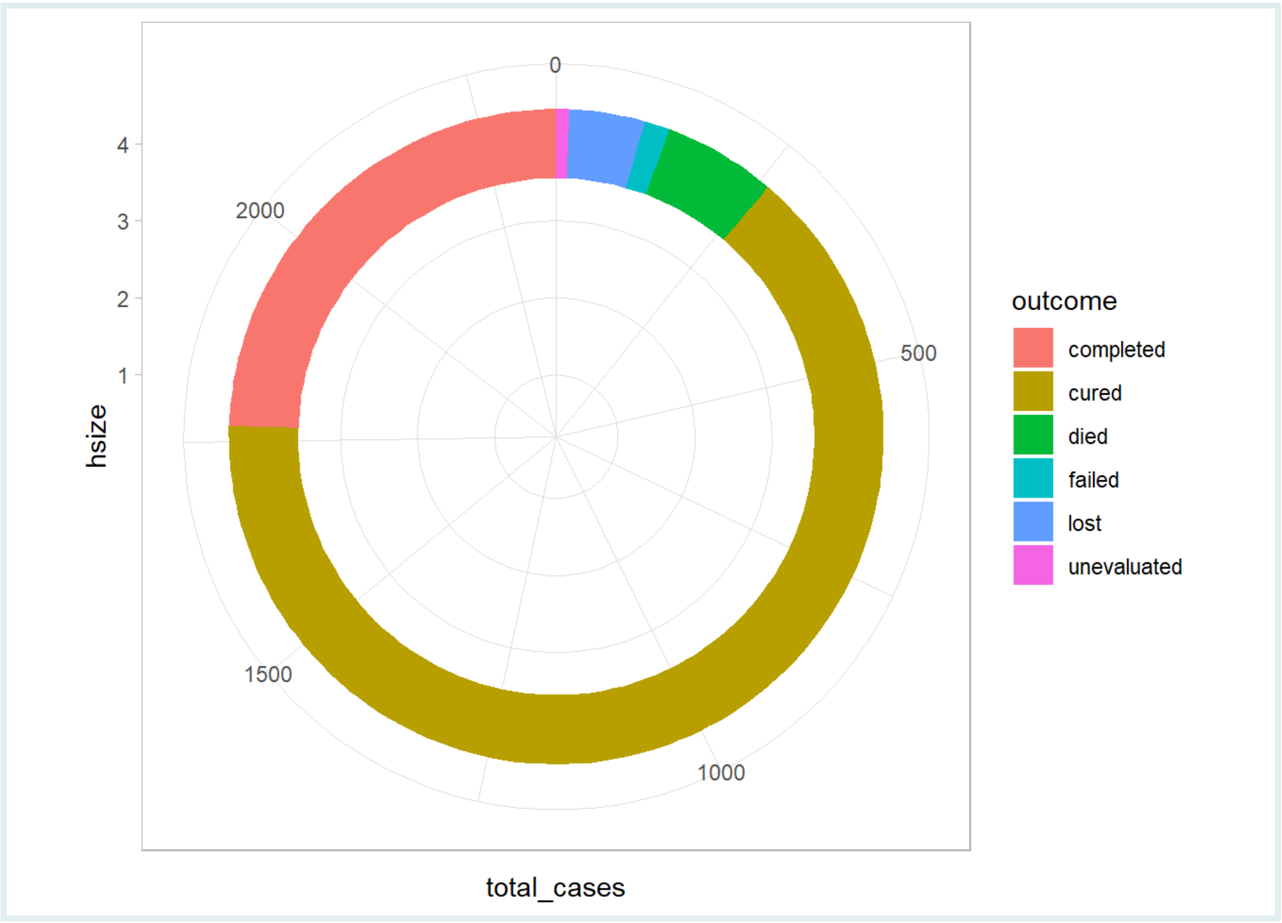
```
# Make it a donut chart
ggplot(outcome_totals,
       aes(x=4,
           y=total_cases,
           fill=outcome)) +
  geom_col() +
  coord_polar(theta = "y") +
  xlim(c(0.2, 4 + 0.5)) # Set x-axis limits
```



- The x value controls hole size.
- Replace 4 with placeholder `hsize` which can be varied.
- The bigger the value the bigger the hole size. Note that the hole size must be bigger than 0.

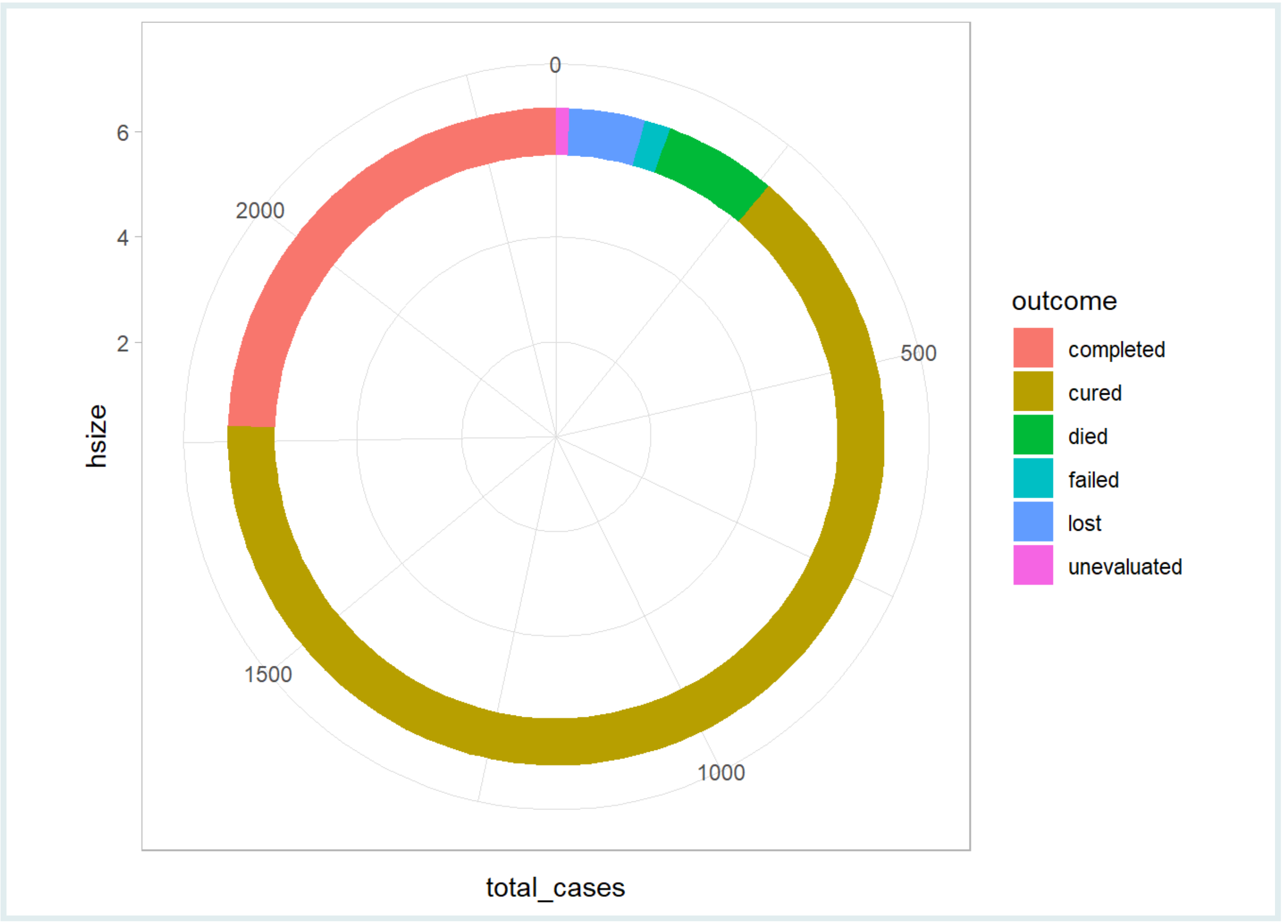
```
# Set hole width
hsize <- 4

ggplot(outcome_totals,
       aes(x=hsize, y=total_cases, fill=outcome)) +
  geom_col() +
  coord_polar(theta = "y") +
  xlim(c(0.2, hsize + 0.5))
```



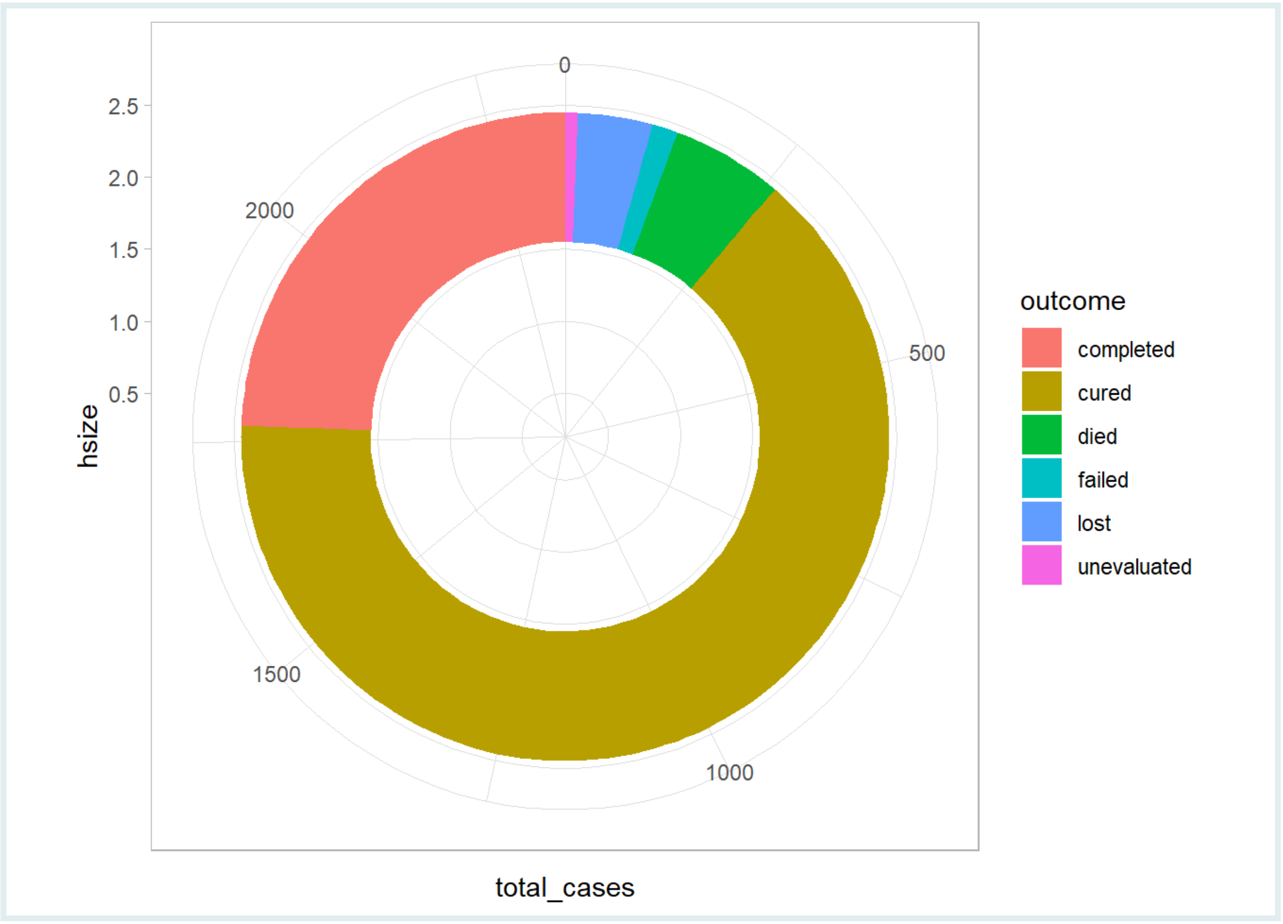
```
# Increase the value to make the hole bigger
hsize <- 6

ggplot(outcome_totals, aes(x=hsize, y=total_cases, fill=outcome)) +
  geom_col() +
  coord_polar(theta = "y") +
  xlim(c(0.2, hsize + 0.5))
```



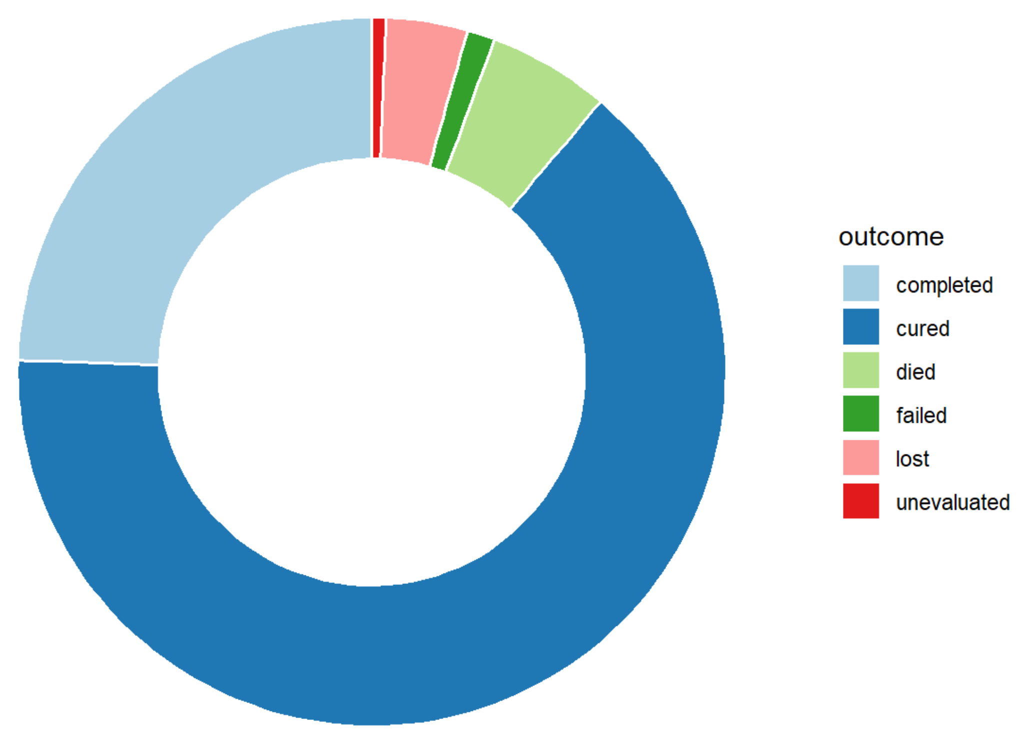
```
# Decrease the value to make the hole smaller
hsize <- 2

ggplot(outcome_totals, aes(x=hsize, y=total_cases, fill=outcome)) +
  geom_col() +
  coord_polar(theta = "y") +
  xlim(c(0.2, hsize + 0.5))
```



The border, fill colors and the theme of the donut plot can be customized in several ways. The code below illustrates some customization options:

```
# Improved aesthetics
ggplot(outcome_totals,
       aes(x=hsize, y=total_cases, fill=outcome)) +
  geom_col(color = "white", linewidth = 0.7) + # Add white borders
  coord_polar(theta = "y") +
  xlim(c(0.2, hsize + 0.5)) +
  scale_fill_brewer(palette = "Paired") + # Set color palette
  theme_void() # Remove background, grid, numeric labels
```



RECAP

What do stacked bars, pies, and donuts have in common? They are all types of parts-to-whole charts! Along with making comparisons, visualizing what things are composed of is one of the most important applications of data visualization. These visualizations work for data that is grouped into multiple subcategories, where our aim is to see how much of the total each category makes up. Examining these "parts-to-whole" relationships can provide insight into demographics, budget allocation, levels of agreement, and more.

PRACTICE (in RMD)

Practice Question: The whole picture

Using the `tb_outcomes` dataset in R, create a visual representation that compares the proportion of tuberculosis cases by outcome across different types of diagnoses. Your visualization should:

1. Group the data by `outcome` and `diagnosis_type`.

2. Summarize the total number of `cases` for each group, accounting for possible missing values (`NA`s).
3. Display this information in a series of donut charts, one for each diagnosis type, showing the proportion of each outcome within the diagnosis type.
4. Ensure the charts have a clean look with `**white**` borders around the segments and no background or gridlines.
5. Arrange the individual donut charts for each diagnosis type using `facet_wrap`.

Write the R code to create this visualization.

PRACTICE

(in RMD)

```r
# PQ4 answer
tb_outcomes %>%
  group_by(------, -------) %>%
  summarise(
    total_cases = sum(-----, -----)) %>%
ggplot(aes(x=----, y=------, fill=-------)) +
  geom_col(color = ------, position = -------) +
  coord_polar(theta = "y") +
  xlim(c(0.2, ------ + 0.5)) +
  theme_void() +
  -------------
```

```
## Error: <text>:3:18: unexpected ','
## 2: tb_outcomes %>%
## 3:   group_by(------,
##                     ^
```

Wrap Up!

Solutions

Contributors

References

Some material in this lesson was adapted from the following sources:

- Horst, Allison. "Allisonhorst/Dplyr-Learnr: A Colorful Introduction to Some Common Functions in Dplyr, Part of the Tidyverse." GitHub. Accessed April 6, 2022. https://github.com/allisonhorst/dplyr-learnr.