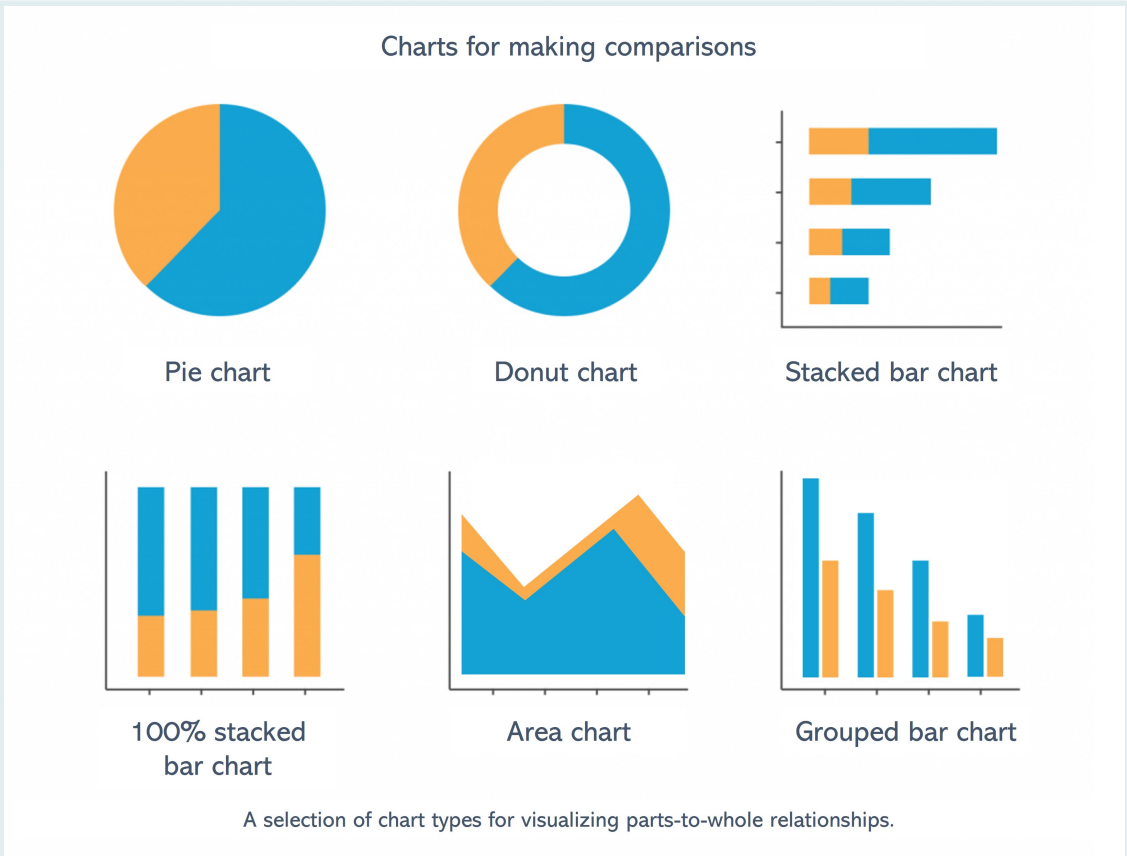
# Plot Labels with ggplot2

## Introduction

Bar plots are one the most common chart type out there and come in several varieties. In the previous lesson, we learned how to make bar plots and their circular counterparts with `{ggplot2}`.




In this lesson, we'll delve into the intricacies of labeling in ggplot2, focusing on `geom_label()` and `geom_text()` functions from `{ggplot2}`.

## Learning Objectives

After this lesson, you will be able to:

1. **Use two different text geoms to label ggplots:**
    - `geom_text()` for simple labels
    - `geom_label()` for emphasized labels
2. Appropriately transform and summarize data in the appropriate format for different chart types.
3. Adjust text placement to position labels on stacked, Dodged, and percent-stacked bar plots.
4. Adjust text placement to position labels on pie charts and donut plots.

## Packages

Run the code below to load the packages for the lesson.

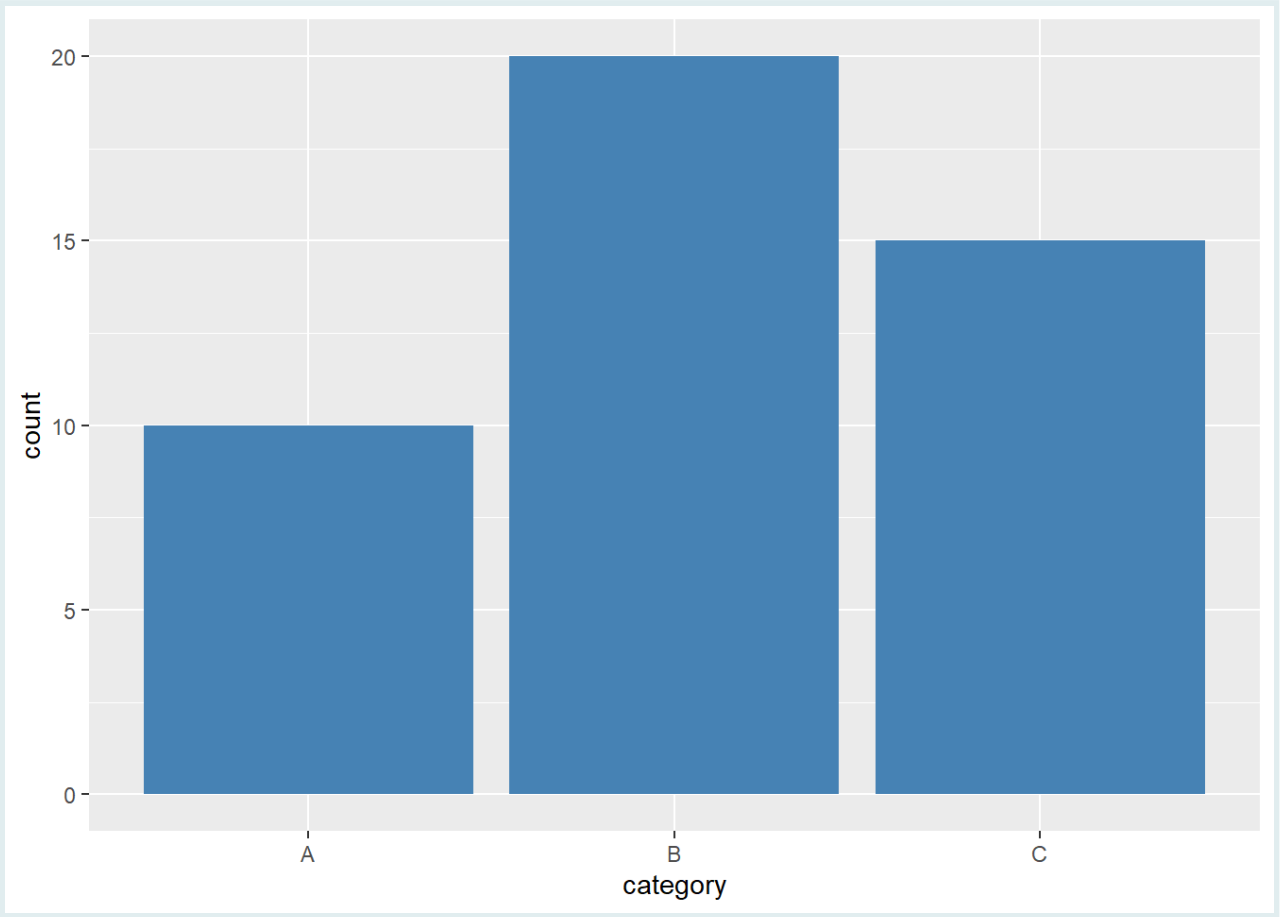
```
pacman::p_load(tidyverse, here, patchwork, medicaldata)
```

## Introduction to text geoms in {ggplot2}

We'll start with `geom_text()` for simple labeling and then move to `geom_label()` for labels with more emphasis. We will show how to use these geoms on simple bar plots, then we will get into more details on how to leverage them for stacked bars, Dodged bars, normalized stacked bars, and circular plots.

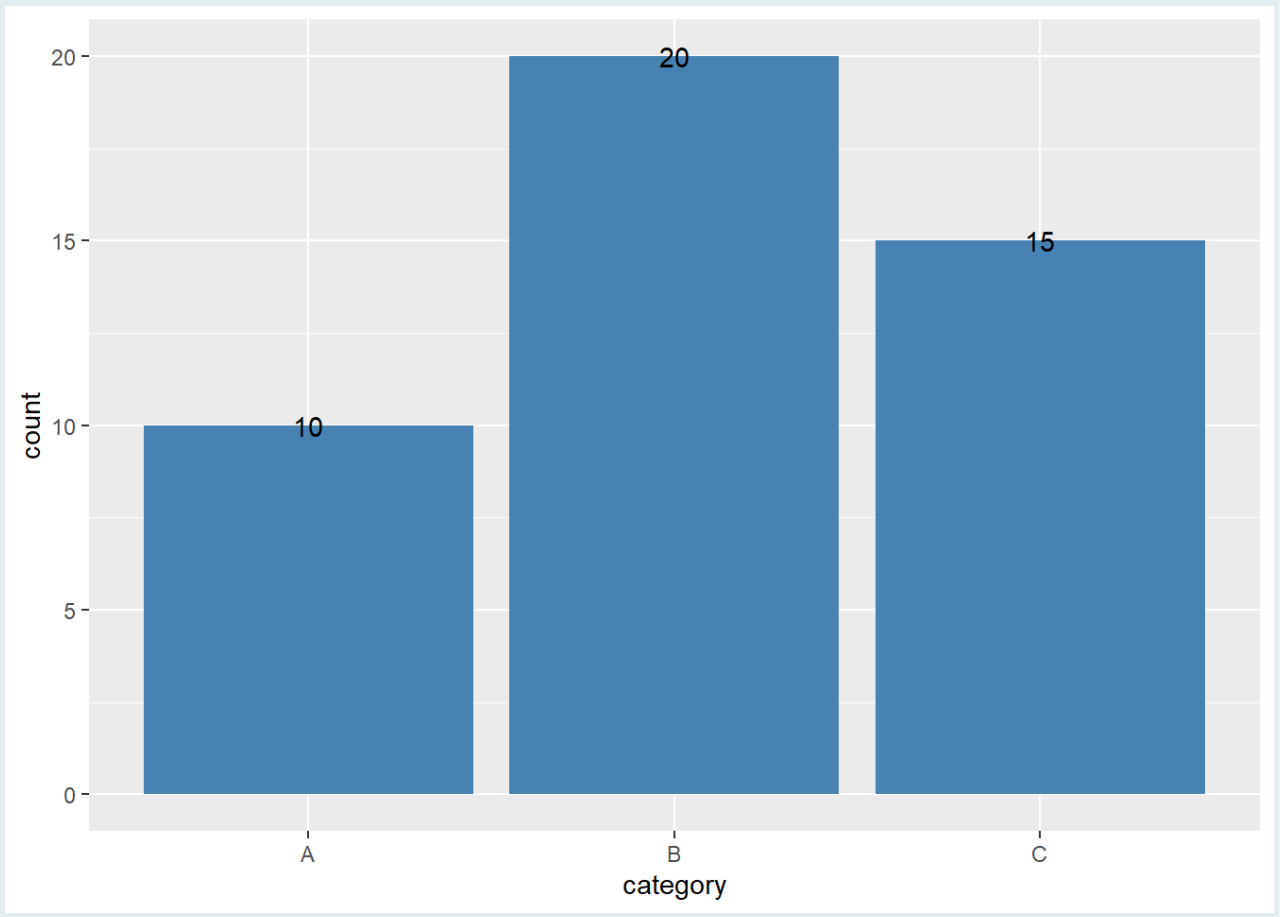
First let's practice using these functions on a simple bar plot made with fake data. Once we cover the fundamentals of the labeling syntax, we will apply these to real epidemiology data.

```
# Create example data frame
data <- data.frame(
  category = c("A", "B", "C"),
  count = c(10, 20, 15)
)

# Create the bar plot
ggplot(data, aes(x = category, y = count)) +
  geom_col(fill = "steelblue")
```




We can easily add labels to our bars with the **`geom_text()`** function and telling the `aes()` function which column to extract **`label`** text from:

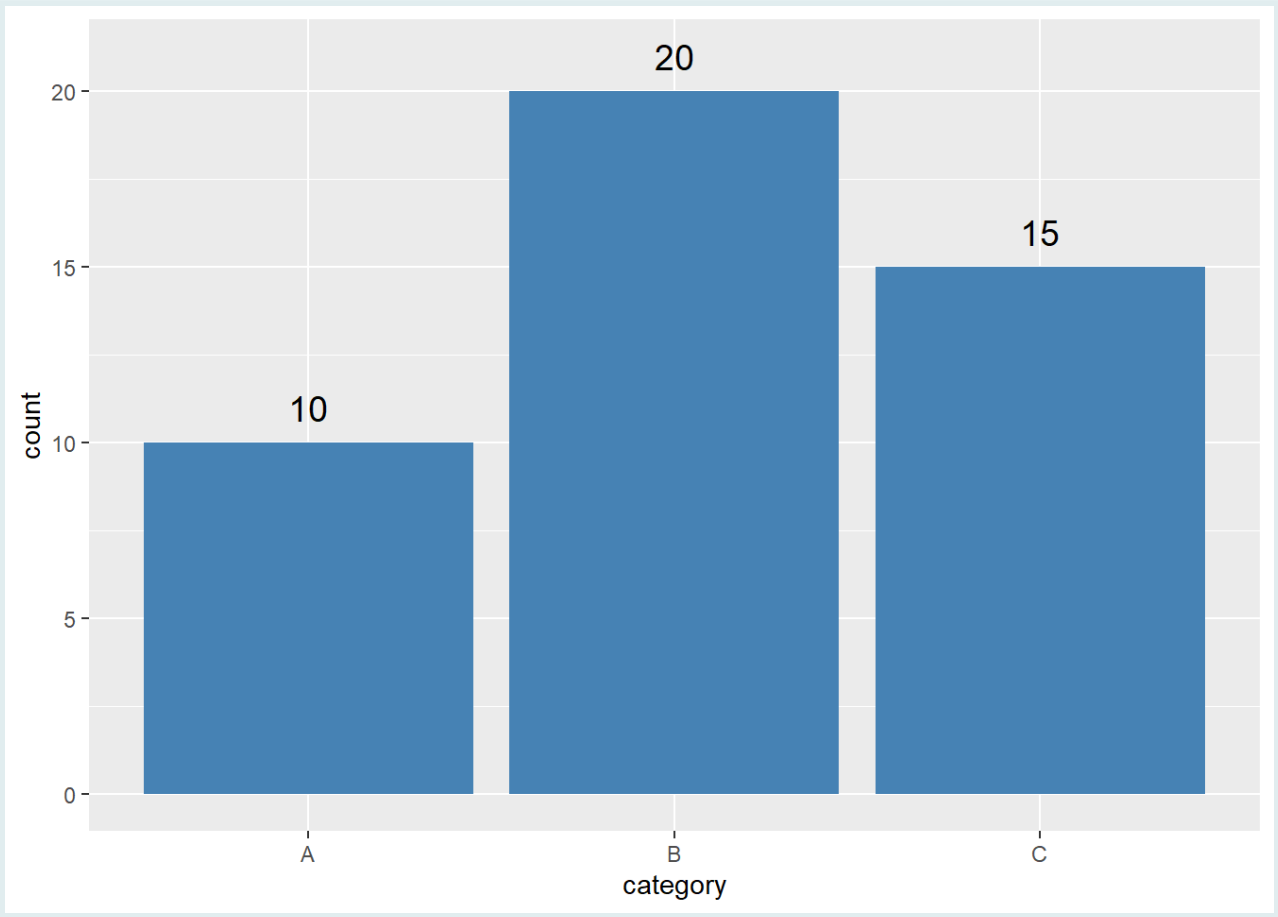
```
ggplot(data, aes(x = category, y = count)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = count)) # provide variable to `label` argument
```




As you can see however, the placement of our text is odd – neither on the bar, nor under the bar. Additionally, they are quite small and difficult to make out. We can address this by making them bigger, and vertically adjusting their placement.

To do this, we will nudge the text upwards using the y_nudge argument. We will also increase the size of the text using the size argument.

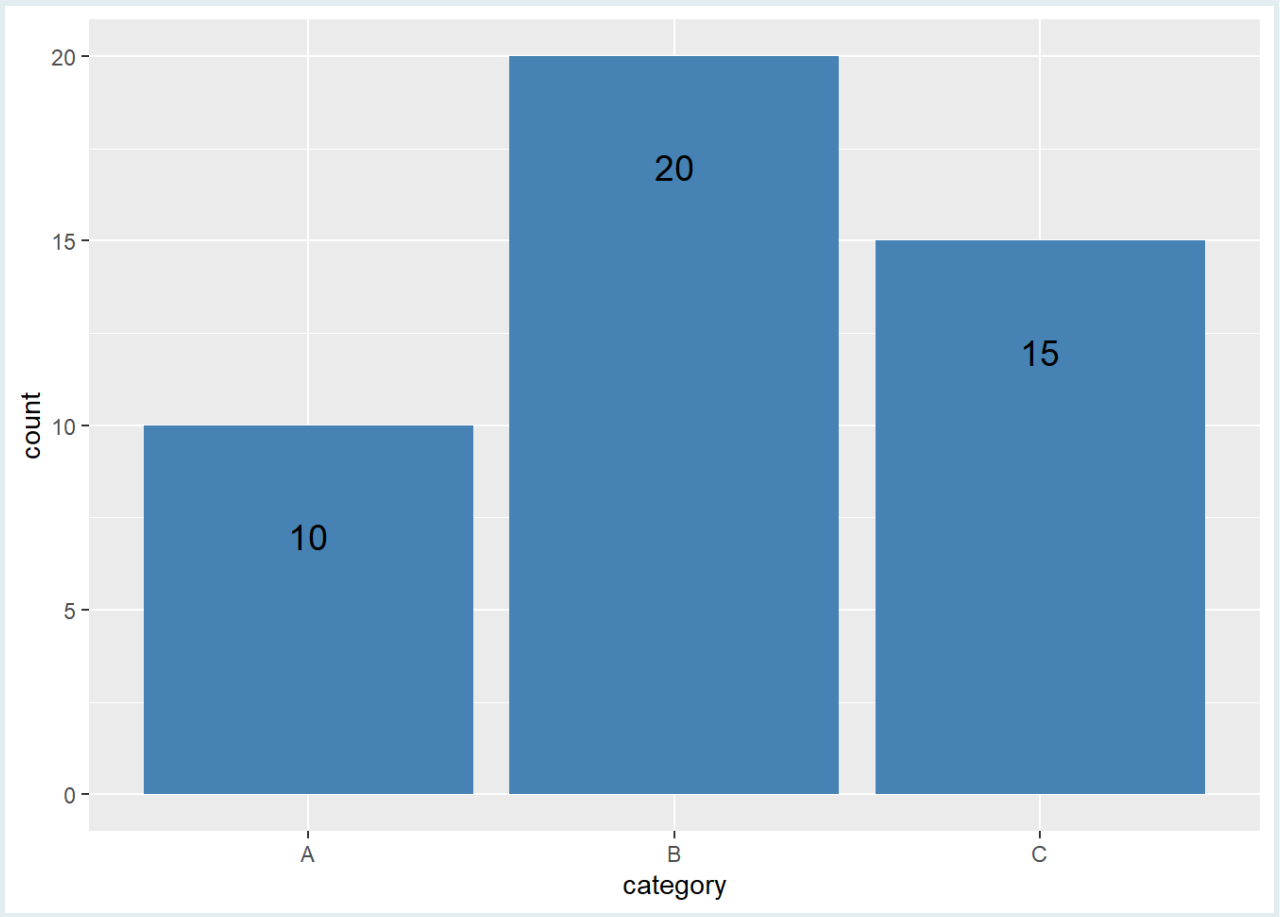
```
ggplot(data, aes(x = category, y = count)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = count),
            nudge_y = 1,
            size = 5) # move text up
```




Note that the value of `nudge_y` is in the same units as the y-axis.

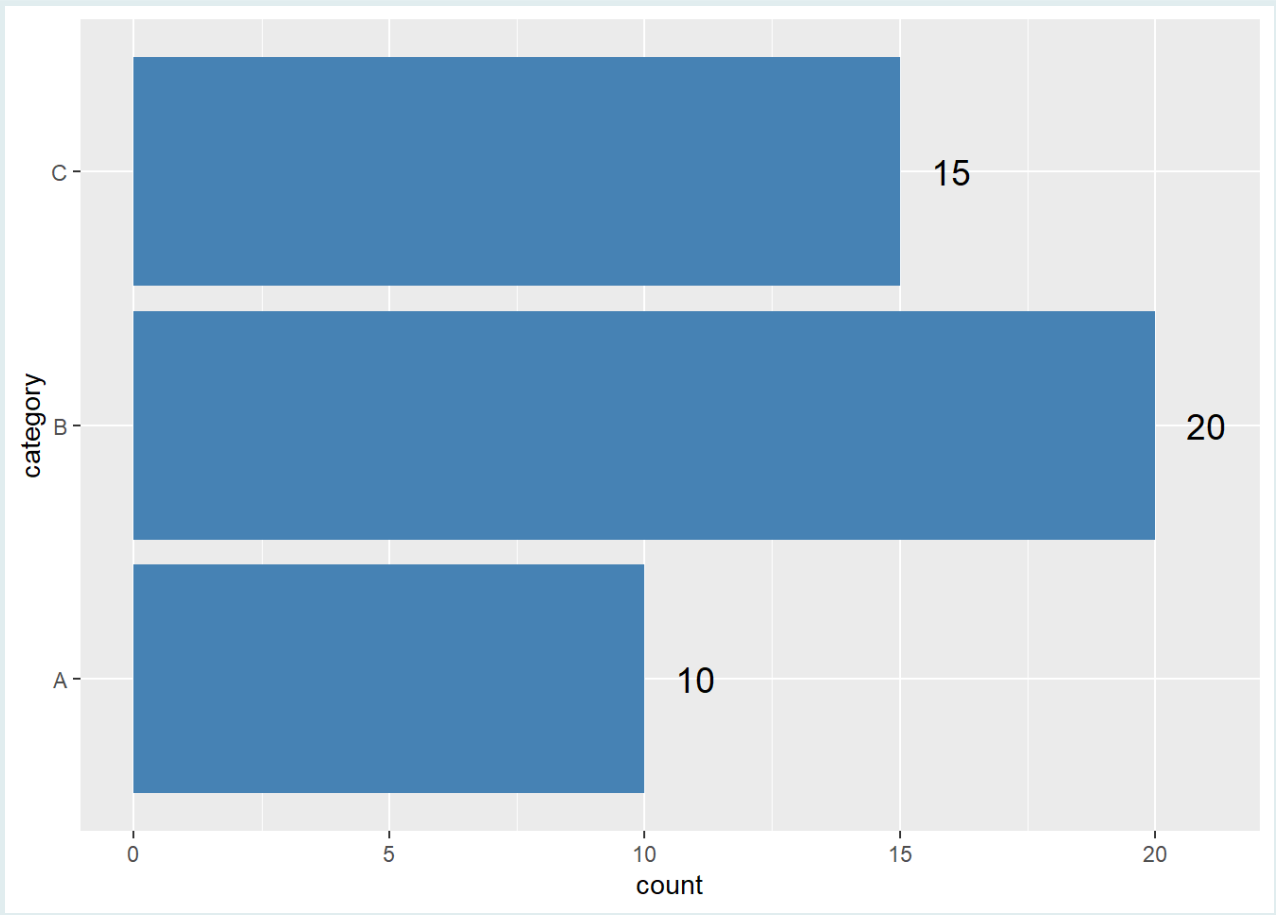
Let's try nudging the text down by setting `nudge_y` to a negative value:

```
ggplot(data, aes(x = category, y = count)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = count),
            nudge_y = -3,
            size = 5) # move text down
```



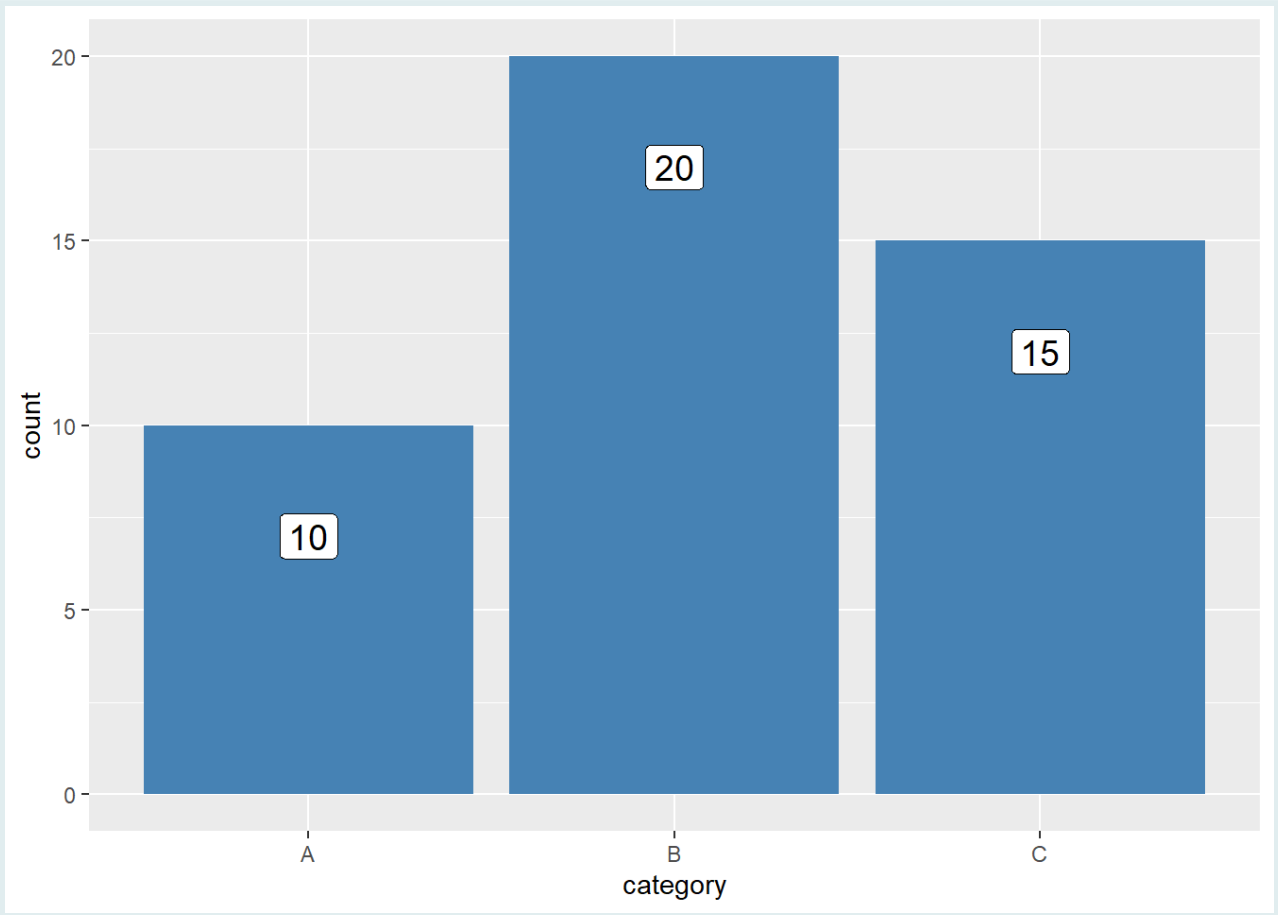

If we made a horizontal bar plot, we would need to nudge the text to the right or left using the `nudge_x` argument instead of `nudge_y`:

```
ggplot(data, aes(x = count, y = category)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = count),
            nudge_x = 1,
            size = 5) # move text to the right
```




Now let's see how the `geom_label()` function works. We can use the same code as above, but replace `geom_text()` with `geom_label()`:

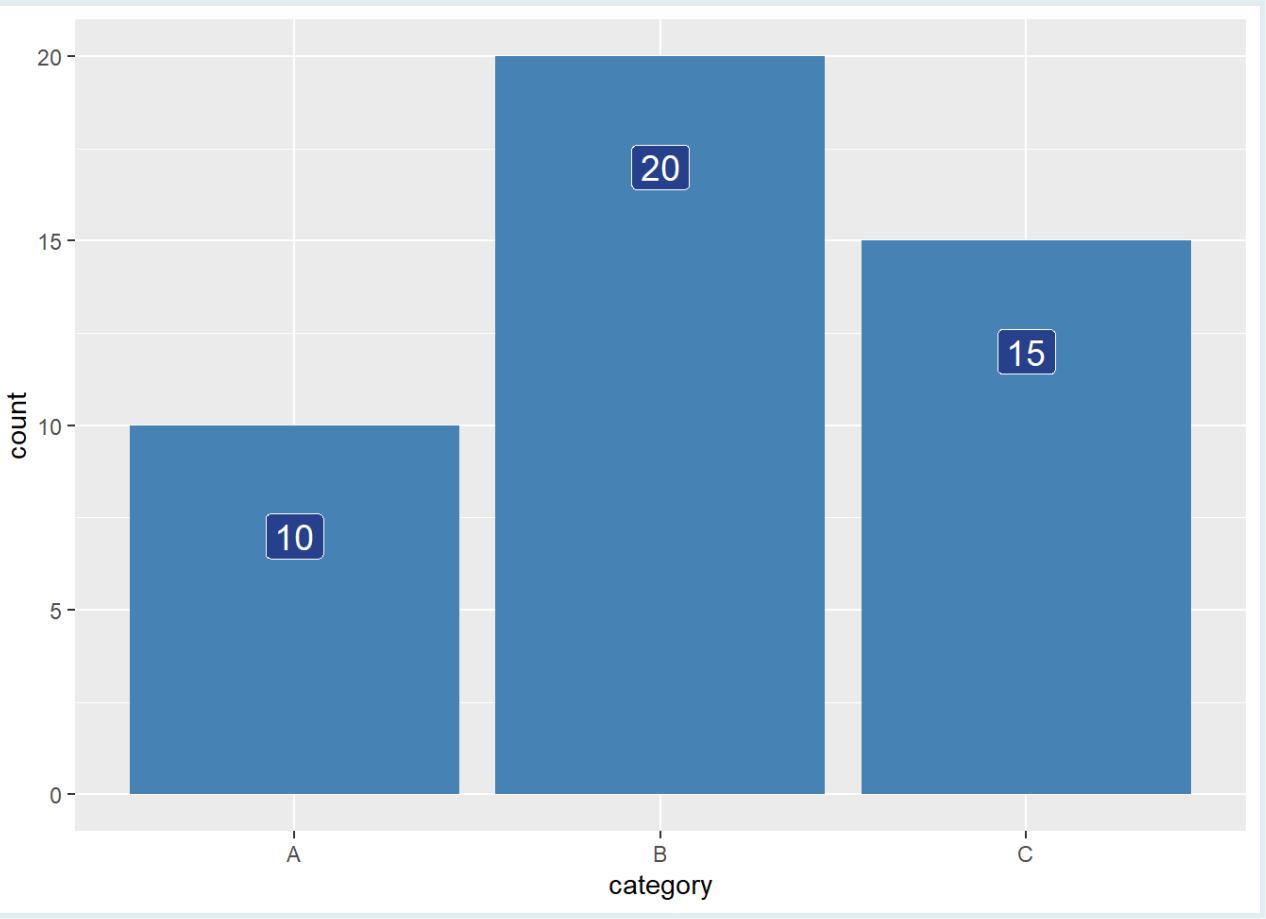
```
ggplot(data, aes(x = category, y = count)) +
  geom_col(fill = "steelblue") +
  geom_label(aes(label = count),
             nudge_y = -3,
             size = 5)
```




As you can see, `geom_label()` draws a rectangle behind the text, making it easier to read.

In this code, the `fill` aesthetic in `geom_label()` can be adjusted to control the background fill color of the labels. For example, let's make the background dark blue, and the text white:

```
ggplot(data, aes(x = category, y = count)) +
  geom_col(fill = "steelblue") +
  geom_label(aes(label = count),
             nudge_y = -3,
             fill = "royalblue4",
             color = "white",
             size = 5)
```




## Q: Simple labeling

### Consider the following sample data frame:


```
# Create example data frame
district_cases <- data.frame(
  district = c("A", "B", "C"),
  cases = c(10, 20, 15)
)

district_cases
```

```
##   district cases
## 1        A    10
## 2        B    20
## 3        C    15
```


Create a labeled bar plot of the data frame above, where the x-axis is the district and the y-axis is the number of cases. The labels should be the number of cases, and should be placed above the bars. The labels should have “darkblue” text with a “lightblue” background. The bar color should be “steelblue”

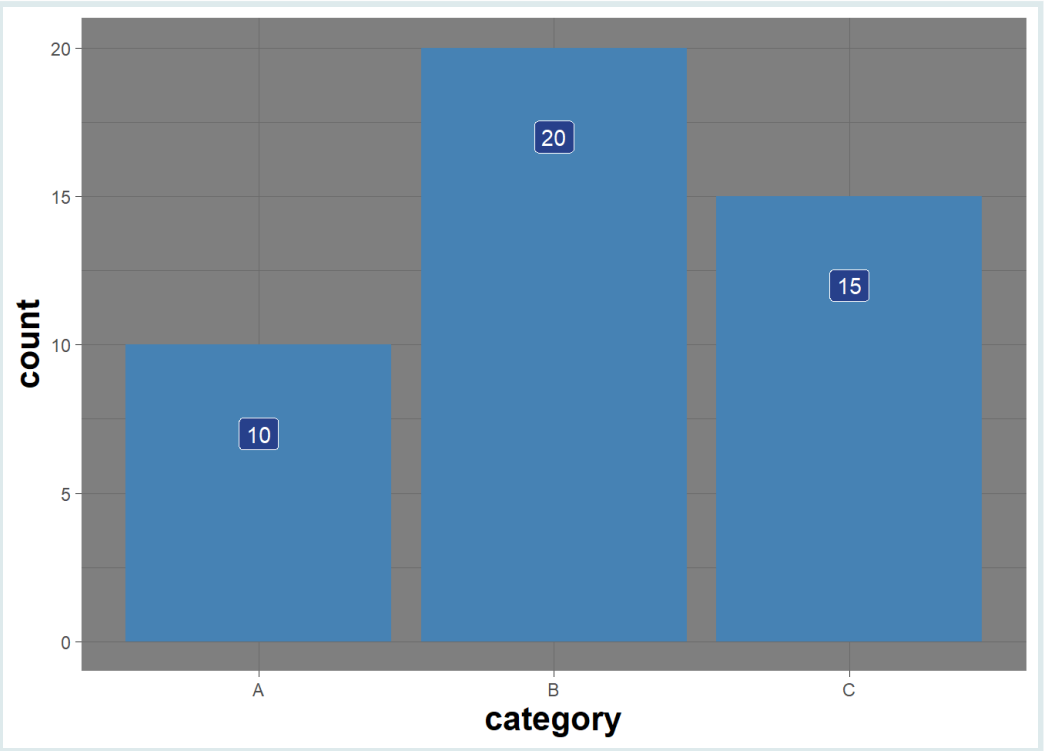
**Setting a custom {ggplot2} theme** So far, we’ve added a theme function to each of our bar plots. Let’s learn how to create our own custom theme functions, and how to use `theme_set()` function to set a global theme for all plots.

We’ll define a custom theme that is a combination of theme_dark and large bold axis labels:


```
theme_dark_custom <-
  theme_dark() +
  theme(
    axis.title = element_text(size = 16, face = "bold")
  )
```

Now we can set use this these for specific plot like this:

```
ggplot(data, aes(x = category, y = count)) +
  geom_col(fill = "steelblue") +
  geom_label(aes(label = count),
             nudge_y = -3,
             fill = "royalblue4",
             color = "white") +
  theme_dark_custom
```




PRO TIP

Note the lack of parentheses after `theme_dark_custom`.

We can set this theme as the default for all plots:

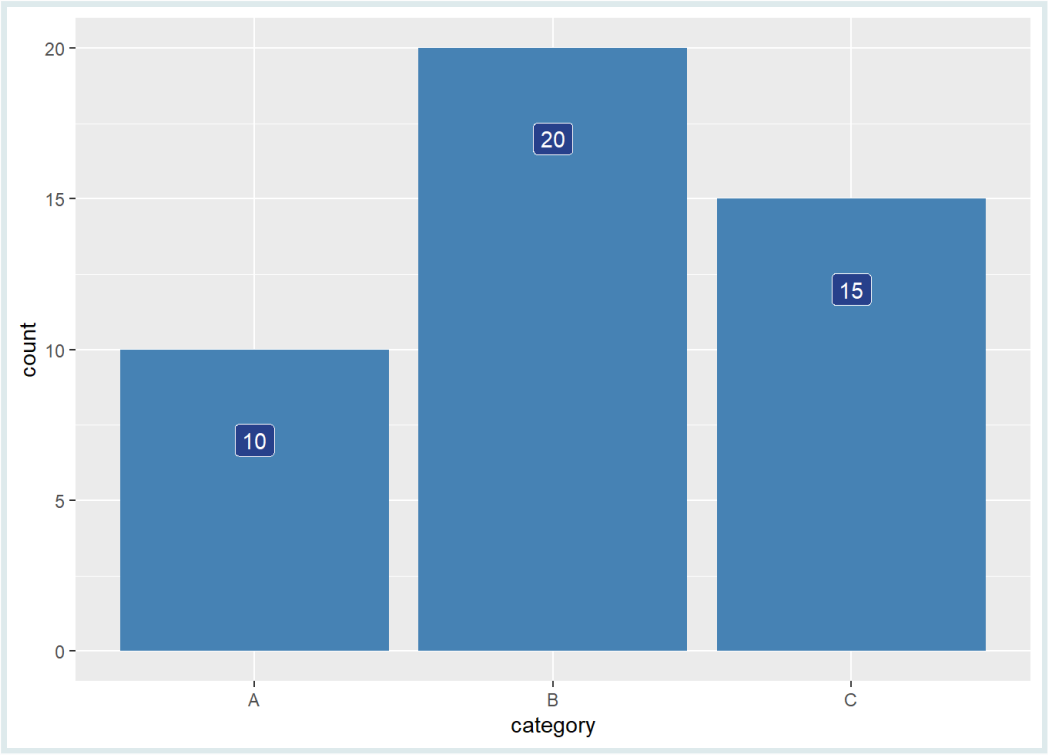
```
theme_set(theme_light_custom)
```

```
## Error in eval(expr, envir, enclos): object
'theme_light_custom' not found
```

Now `theme_light_custom()` will be automatically applied to every plot you draw.

For example, let's redraw the plot we made earlier:

```
ggplot(data, aes(x = category, y = count)) +
  geom_col(fill = "steelblue") +
  geom_label(aes(label = count),
             nudge_y = -3,
             fill = "royalblue4",
             color = "white")
```


This is a great way to ensure that all of your plots have a consistent look and feel.

To set the default theme back to the original, use `theme_set(theme_gray())`.

```
theme_set(theme_gray())
```

## The `vjust` and `hjust` arguments

Rather than use `nudge_x` and `nudge_y`, to adjust the position of text, we can use the `vjust` and `hjust` arguments. These arguments adjust the vertical and horizontal justification of the text, respectively. It is notoriously difficult to understand exactly how these work, but we will introduce their basic functionality here.

### Understanding `hjust` (horizontal justification)

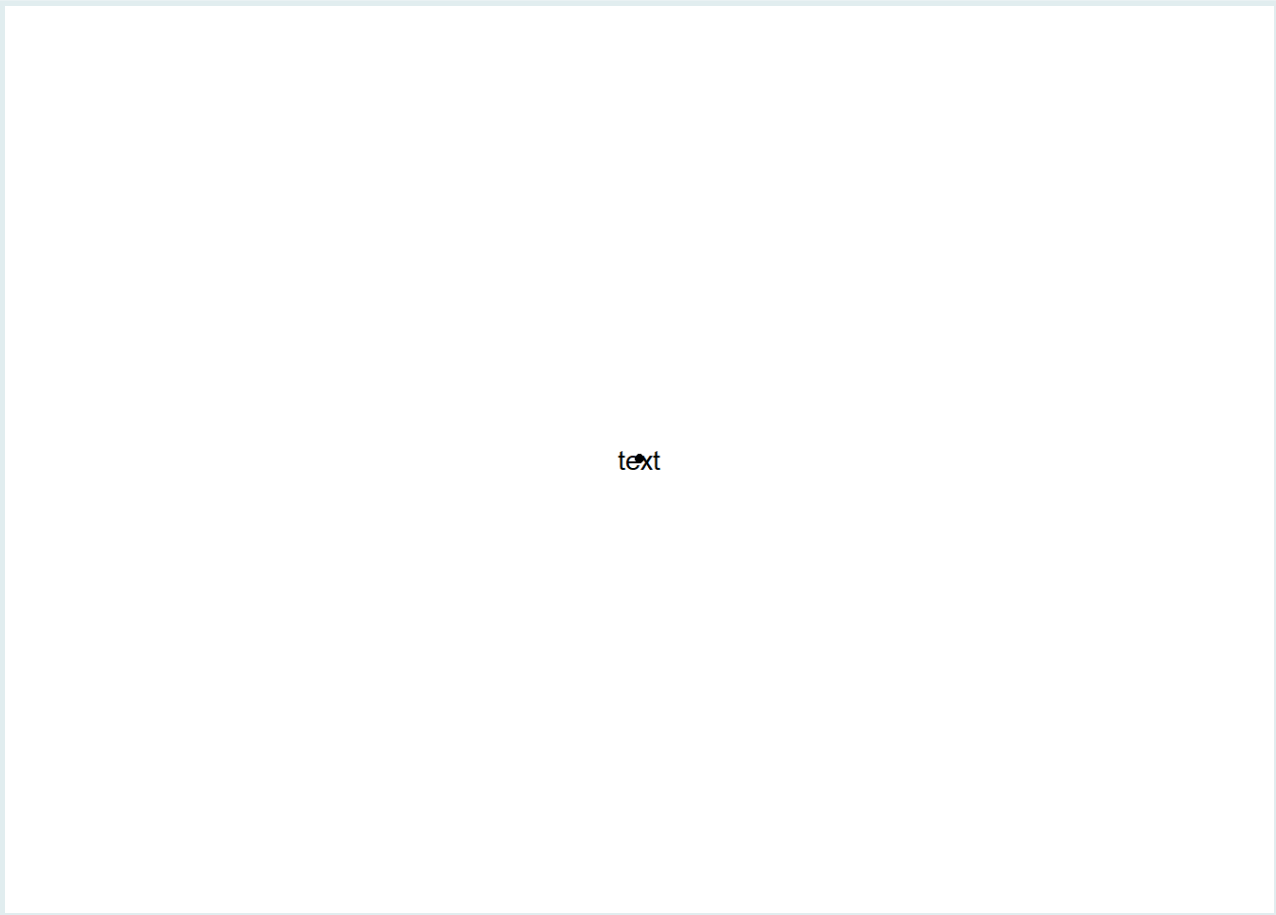
The `hjust` argument in `ggplot2` adjusts the horizontal position of text labels relative to their anchor points (the actual data points). `hjust` values range from 0 to 1, where:

- `hjust = 0` aligns the text label's left edge with the anchor point.
- `hjust = 0.5` centers the text label on the anchor point.

- `hjust = 1` aligns the text label's right edge with the anchor point.

Here's a simple example to illustrate this. First, let's make a plot with a single point and text with no `hjust` argument:

```
# Example data
df <- data.frame(x = 1, y = 1)

# Base plot with a point
base_p <- ggplot(df, aes(x, y)) + geom_point() + theme_void()

base_p + geom_text(aes(label = "text"))
```




With no `hjust` argument, the text is centered on the point, which means that the default value of `hjust` is 0.5.

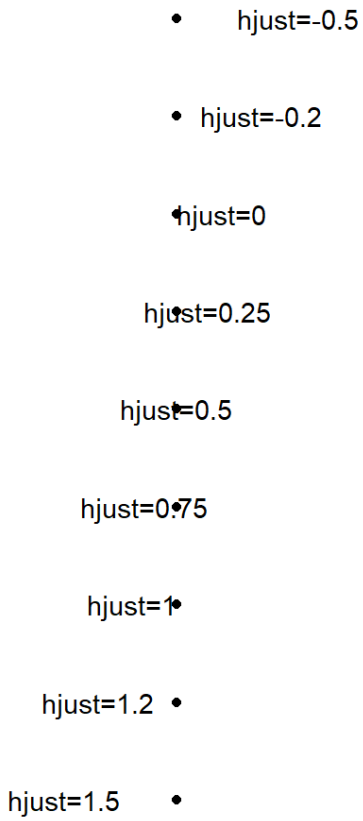
Now let's try setting `hjust` to a variety of values:

```
p_hjust_0 <- base_p + geom_text(aes(label = "hjust=0"), hjust = 0)
p_hjust_0.25 <- base_p + geom_text(aes(label = "hjust=0.25"), hjust = 0.25)
p_hjust_0.5 <- base_p + geom_text(aes(label = "hjust=0.5"), hjust = 0.5)
p_hjust_0.75 <- base_p + geom_text(aes(label = "hjust=0.75"), hjust = 0.75)
p_hjust_1 <- base_p + geom_text(aes(label = "hjust=1"), hjust = 1)

# Combine plots with patchwork
p_hjust_0 / p_hjust_0.25 / p_hjust_0.5 / p_hjust_0.75 / p_hjust_1
```


As you can see, the text is aligned to the left edge of the point when `hjust = 0`, to the right edge of the point when `hjust = 1`, and moves closer to the center as `hjust` approaches 0.5.

While `hjust` was originally meant to be used between 0 and 1, you can actually use any value for `hjust`, above or below 0 and 1. For example, if you set `hjust = -0.2`, the text will be left-aligned, but with an additional 20% of the text width added to the left of the anchor point, and if you set `hjust = 1.2`, the text will be right-aligned, but with an additional 20% of the text width added to the right of the anchor point:

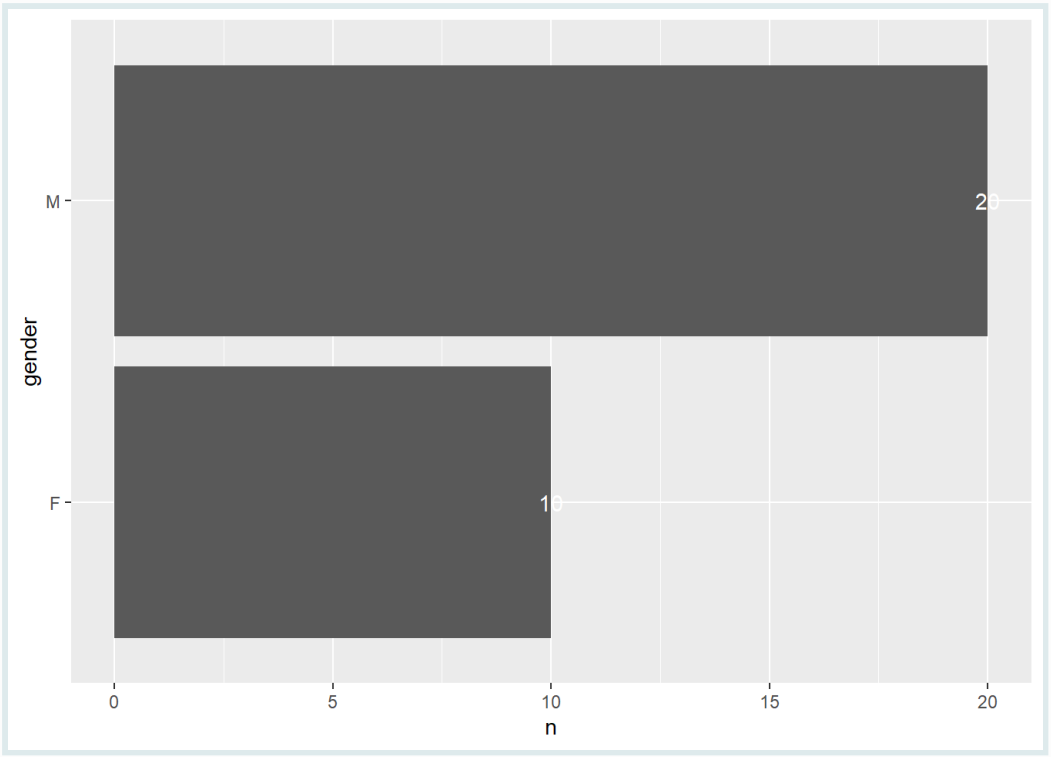
```
p_hjust_neg0.5 <- base_p + geom_text(aes(label = "hjust=-0.5"), hjust = -0.5)
p_hjust_neg0.2 <- base_p + geom_text(aes(label = "hjust=-0.2"), hjust = -0.2)
p_hjust_1.2 <- base_p + geom_text(aes(label = "hjust=1.2"), hjust = 1.2)
p_hjust_1.5 <- base_p + geom_text(aes(label = "hjust=1.5"), hjust = 1.5)

# Combine plots with patchwork
p_hjust_neg0.5 / p_hjust_neg0.2 / p_hjust_0 / p_hjust_0.25 / p_hjust_0.5 /
        p_hjust_0.75 / p_hjust_1 / p_hjust_1.2 / p_hjust_1.5
```




## Q: Horizontal adjusment practice


Consider the following horizontal bar plot with text labels added:

```
# sample data
sample_gender <-
  data.frame(gender = c("F", "M"),
             n = c(10, 20))

ggplot(sample_gender, aes(x = n, y = gender)) +
  geom_col() +
  geom_text(aes(label = n), color = "white")
```

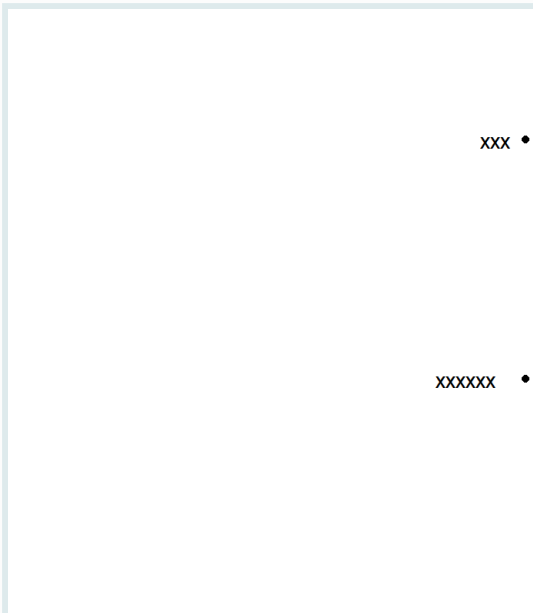




Use the `hjust` or `vjust` arguments to adjust the position of the text label so that it is inside the bar, with some padding on the right side.


Using `hjust` and `vjust` values outside the 0-1 range can be problematic when your labels are not the same length. For example, if you have labels of different lengths, setting `hjust = 1.2` will cause the longer labels to extend further to the right than the shorter labels.

For example:

```
# Different text labels with varying lengths
p_xx <- base_p + geom_text(aes(label = "xxx"), hjust = 1.5)
p_xxxx <- base_p + geom_text(aes(label = "xxxxxx"), hjust = 1.5)
p_xxxxxx <- base_p + geom_text(aes(label = "xxxxxxxxx"), hjust =
        1.5)

# Combine plots with patchwork
p_xx / p_xxxx / p_xxxxxx
```





As you can see, the longer labels have more extra space added to the right of the anchor point than the shorter labels. This is because `hjust` is adding 50% of the text width to the right of the anchor point, so longer labels get more padding.

If this is a problem for you, you can use the `nudge_x` argument to adjust the position of the labels instead. There *are* certain times when using nudges can be problematic though, which is why `hjust` and `vjust` are still useful.

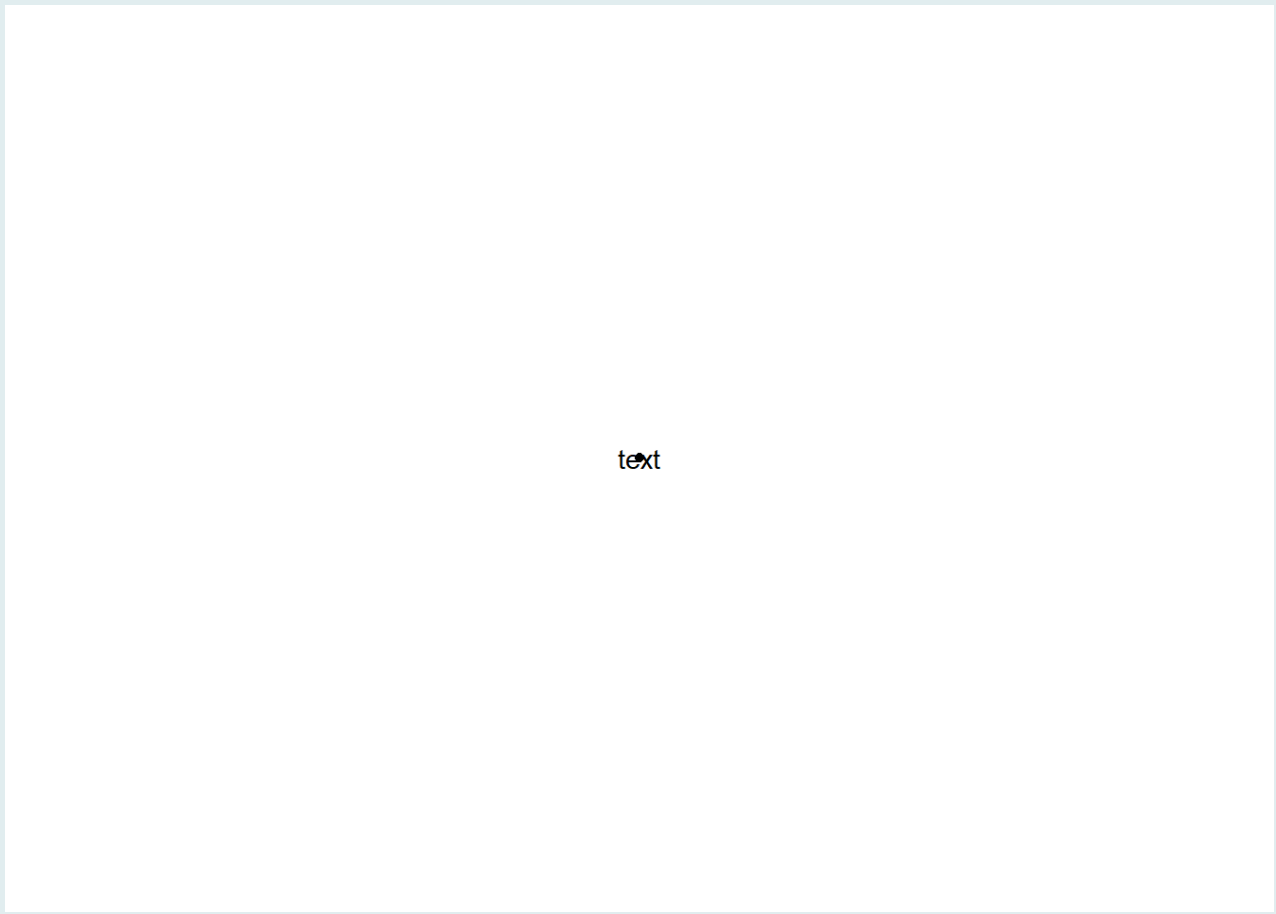
### Understanding `vjust` (vertical justification)

Similarly, the `vjust` argument in `ggplot2` adjusts the vertical position of text labels in relation to their anchor points. `vjust` values also range from 0 to 1, where:

- `vjust = 0` aligns the bottom edge of the text label with the anchor point.
- `vjust = 0.5` centers the text label vertically on the anchor point.
- `vjust = 1` aligns the top edge of the text label with the anchor point.

Here's an example to illustrate `vjust`. We'll start with the same base plot and add text with no `vjust` argument:

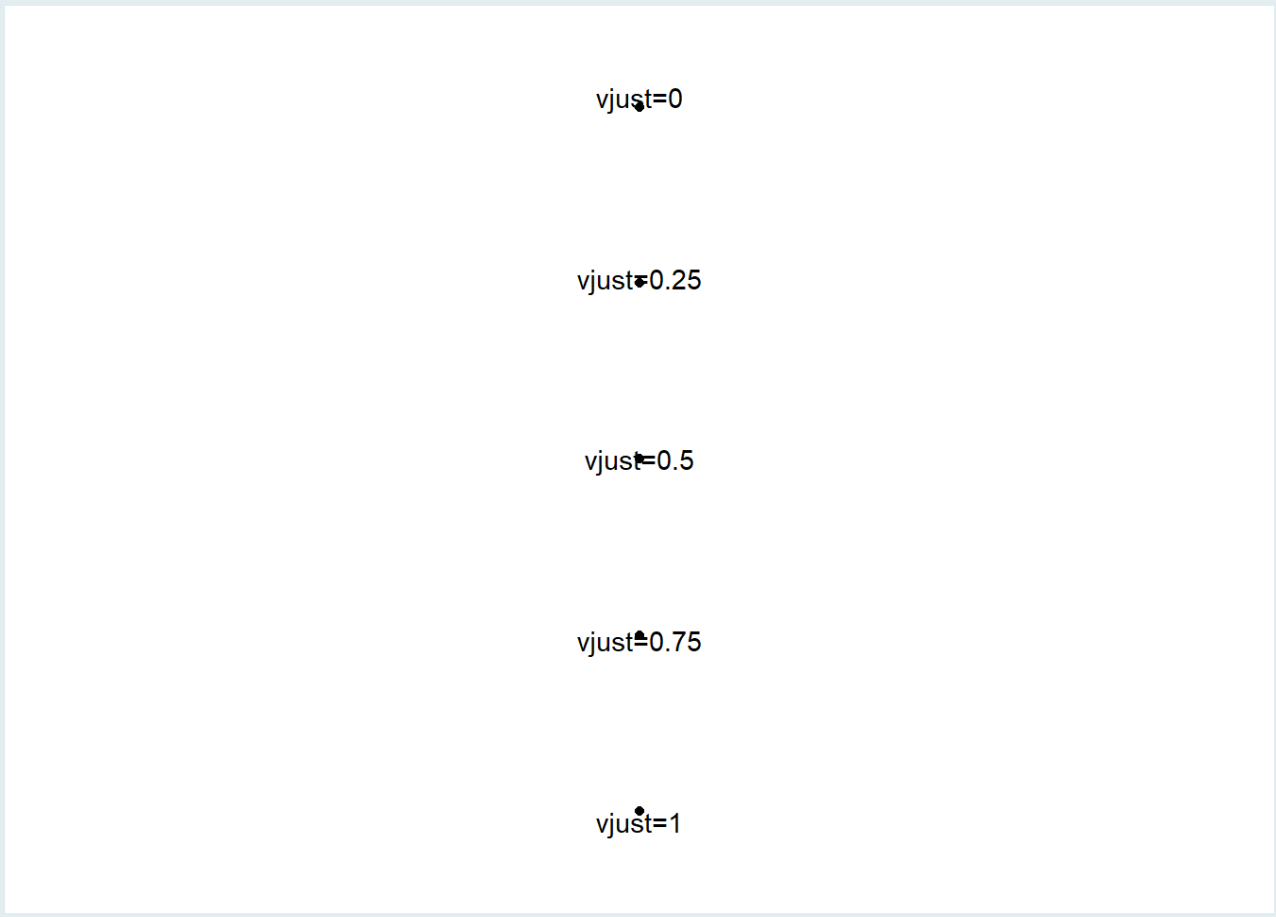
```r
# Base plot with a point
p <- ggplot(df, aes(x, y)) + geom_point() + theme_void()

p + geom_text(aes(label = "text"))
```




By default, with no `vjust` specified, the text is vertically centered on the point, indicating the default value of `vjust` is 0.5.

Now, let's experiment with different `vjust` values:

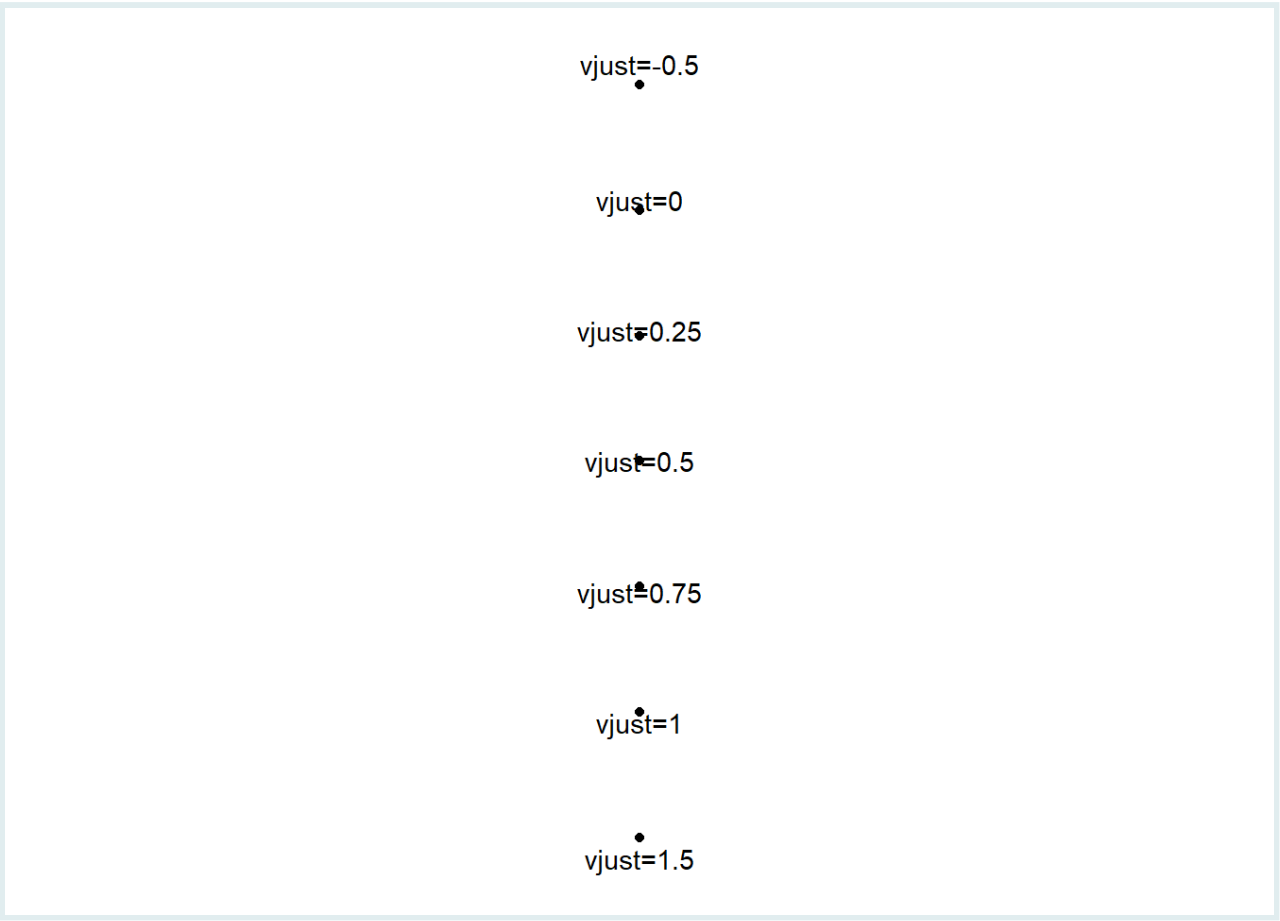
```r
p_vjust_0 <- p + geom_text(aes(label = "vjust=0"), vjust = 0)
p_vjust_0.25 <- p + geom_text(aes(label = "vjust=0.25"), vjust = 0.25)
p_vjust_0.5 <- p + geom_text(aes(label = "vjust=0.5"), vjust = 0.5)
p_vjust_0.75 <- p + geom_text(aes(label = "vjust=0.75"), vjust = 0.75)
p_vjust_1 <- p + geom_text(aes(label = "vjust=1"), vjust = 1)

# Combine plots with patchwork
p_vjust_0 / p_vjust_0.25 / p_vjust_0.5 / p_vjust_0.75 / p_vjust_1
```




Here, `vjust = 0` aligns the text to the bottom of the point, `vjust = 1` aligns it to the top, and as `vjust` approaches 0.5, the text moves closer to the vertical center.

Like `hjust`, `vjust` can also take values outside the 0 to 1 range. For example, `vjust = -0.2` would place the text slightly below the anchor point, and `vjust = 1.2` would place it slightly above. Let's see how these values affect text positioning:

```
p_vjust_neg0.5 <- p + geom_text(aes(label = "vjust=-0.5"), vjust = -0.5)
p_vjust_1.5 <- p + geom_text(aes(label = "vjust=1.5"), vjust = 1.5)

# Combine plots with patchwork
p_vjust_neg0.5 / p_vjust_0 / p_vjust_0.25 / p_vjust_0.5 / p_vjust_0.75 /
        p_vjust_1 / p_vjust_1.5
```

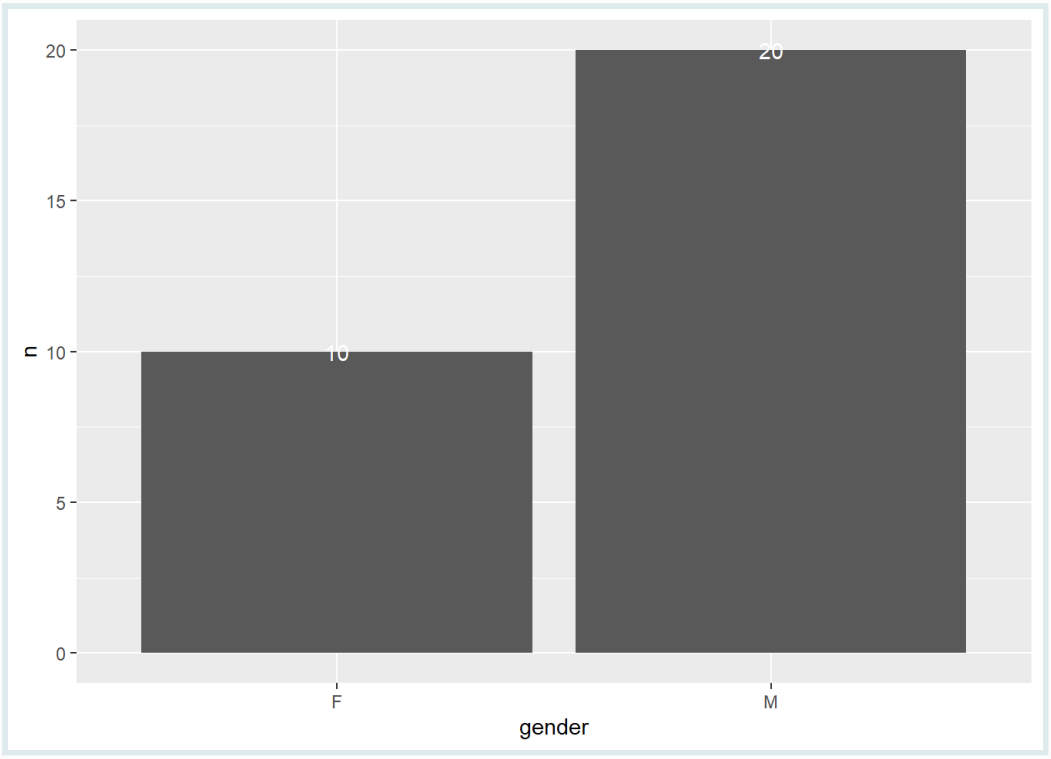



As with `hjust`, using `vjust` values beyond the typical 0 to 1 range can be useful for fine-tuning the placement of your text labels, allowing them to extend slightly above or below the anchor point.

### Q: Vertical adjusment practice


Consider the following bar plot with text labels added:

```
# sample data
sample_gender <-
  data.frame(gender = c("F", "M"),
             n = c(10, 20))


ggplot(sample_gender,
       aes(x = gender, y = n)) +
  geom_col() +
  geom_text(aes(label = n), color = "white")
```


Use the `hjust` or `vjust` arguments to adjust the position of the text label so that it is inside the bar, with some padding on the top.

## Data Example: TB treatment outcomes in Benin

Let's apply what we've learned to a real dataset.

The `tb_outcomes` dataset, which we used in the previous lesson, will serve as the foundation for our examples.

```
tb_outcomes <- read_csv(here::here('data/benin_tb.csv'))
tb_outcomes
```

We'll be trying to plot the number of cases per hospital.

Unlike with our initial practice dataset, we do not already have the total number of cases per hospital; this information is stored in the `cases` column, but we need to summarize it first.

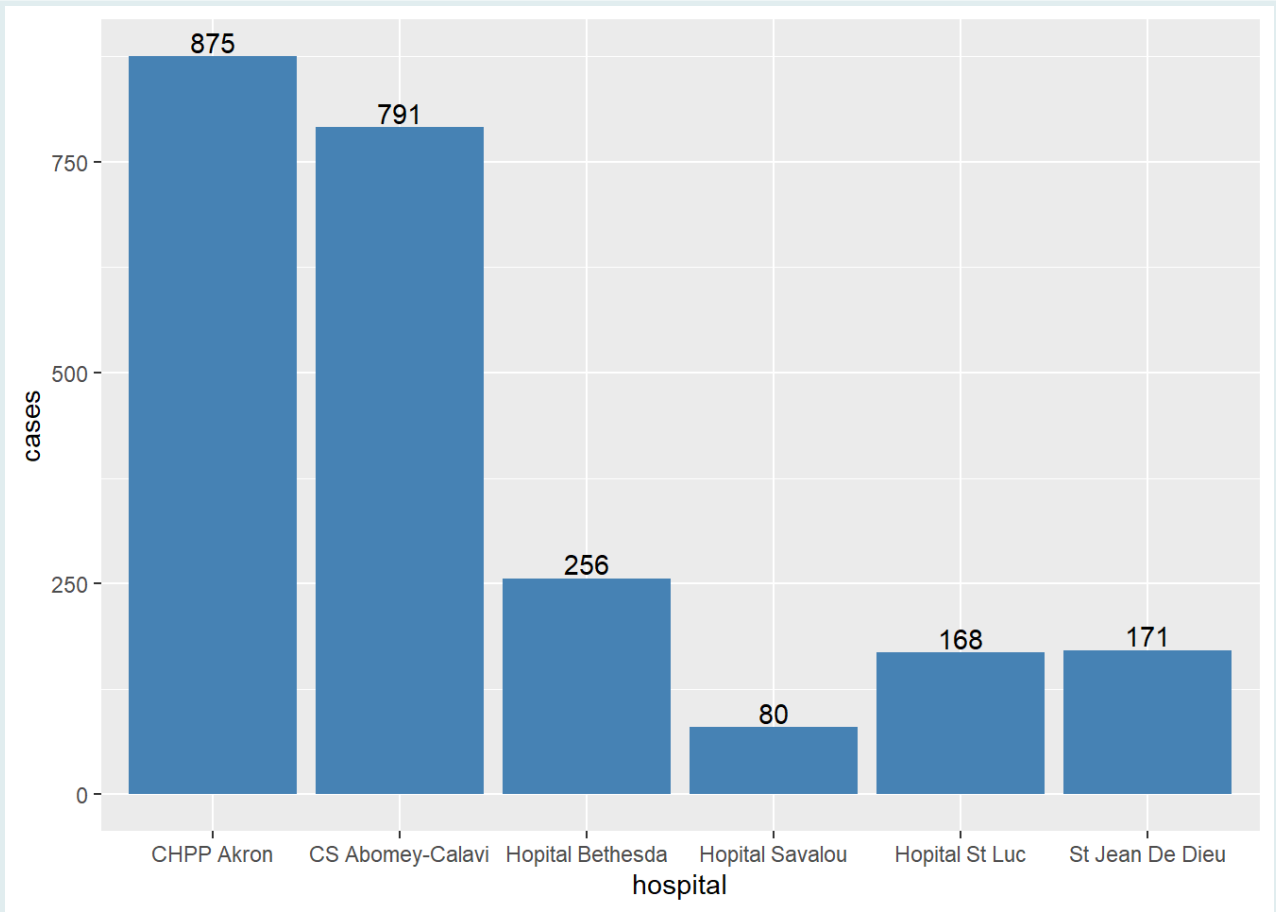
Let's calculate the total number of cases per hospital using the `group_by()` and `summarize()` function:

```r
hospital_sums <-
  tb_outcomes %>%
  group_by(hospital) %>%
  summarize(cases = sum(cases))

hospital_sums
```

```
## # A tibble: 6 × 2
##   hospital          cases
##   <chr>             <dbl>
## 1 CHPP Akron          875
## 2 CS Abomey-Calavi    791
## 3 Hopital Bethesda    256
## 4 Hopital Savalou      80
## 5 Hopital St Luc      168
## 6 St Jean De Dieu     171
```

Now let's use `hospital_sums` to visualize each hospital's total number of cases and use `geom_text()` to annotate the bars:

```r
ggplot(hospital_sums, aes(x = hospital, y = cases)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = cases),
            vjust = -0.2)
```



Great, now you see how to use the `summarize()` function to calculate group totals, and how to use `geom_text()` to annotate your plots.


Q: Summarize then plot

Consider the `aus_tb_notifs` dataset imported below, which shows the number of TB cases in urban and rural areas per quarter:

```
aus_tb_notifs <-
        read_csv(here::here('data/aus_tb_notifs_modified.csv'))
aus_tb_notifs
```

```
## # A tibble: 52 × 4
##     year quarter rural urban
##    <dbl> <chr>   <dbl> <dbl>
##  1  2010 Q1          4    87
##  2  2010 Q2          4    98
##  3  2010 Q3          5   101
##  4  2010 Q4         10   124
##  5  2011 Q1          5    81
##  6  2011 Q2          4    52
##  7  2011 Q3          9   102
##  8  2011 Q4          5   100
##  9  2012 Q1          9    80
## 10  2012 Q2          4    63
## # ℹ 42 more rows
```

Create a simple bar plot to visualize the total number of TB cases in urban areas for **each year**. Label each bar with the total number of cases using `geom_text()` just below the bar.

**Hint**: First, aggregate the data by year and sum up the urban cases. Then use `ggplot()` with `geom_col()` for the bar plot and `geom_text()` for the labels.


### Further Aesthetic modifications

So far we have only used some of the possible aesthetics for `geom_text()`. The minimum three aesthetics are `x`, `y`, and `label`. These must be mapped to a variable defined inside `aes()`.

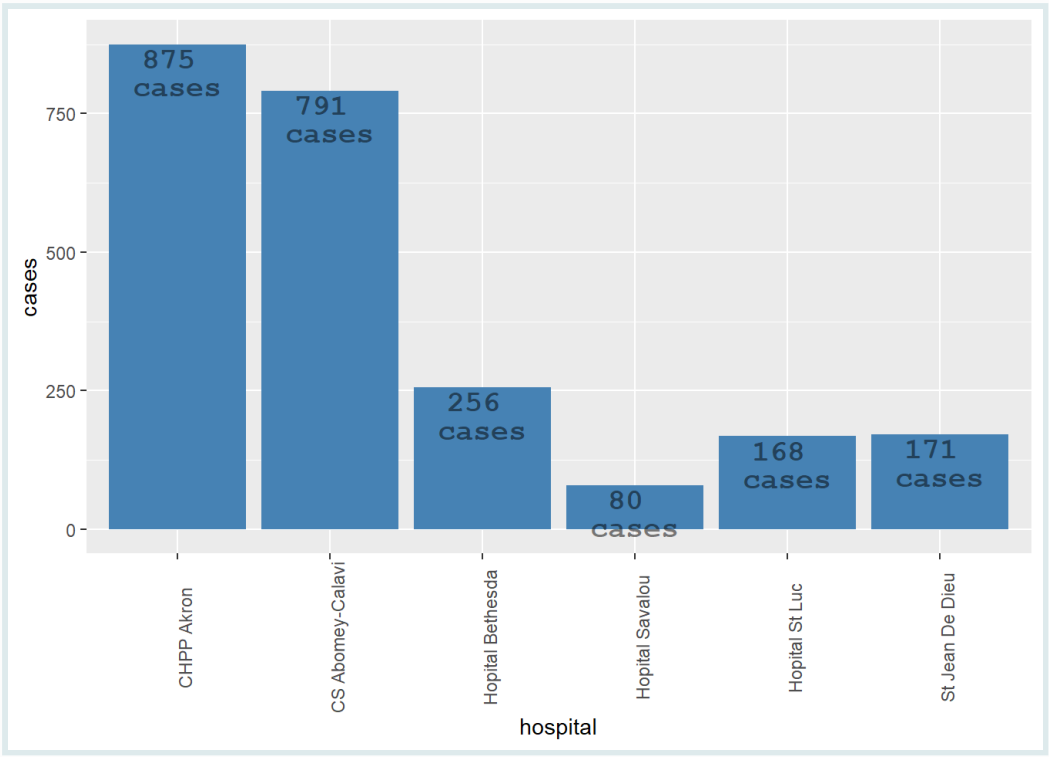
Additional aesthetics include:

- `size`: the size of the text, in mm
- `angle`: the angle of the text, from 0 to 360
- `alpha`: the transparency of the text, from 0 to 1
- `color`: the color of the text
- `family`: the font family of the text, such as “sans”, “serif”, “mono”
- `fontface`: the font face of the text, including “plain”, “bold”, “italic”, “bold.italic”
- `group`: a grouping variable for the text
- `hjust`: horizontal justification of the text
- `vjust`: vertical justification of the text
- `lineheight`: the line height of the text


`nudge_y` and `nudge_x` are also available, but are not formally considered aesthetics, as they cannot be mapped to a variable inside `aes()`, and must be set outside of it.

Here is an example plot with most of these aesthetics set. It’s not a very beautiful plot. Try modifying the code to see how each aesthetic changes the plot:

```
ggplot(hospital_sums, aes(x = hospital, y = cases)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = paste(cases, "\ncases")),
            size = 5,
            angle = 0,
            alpha = 0.5,
            color = "black",
            family = "mono",
            fontface = "bold",
            hjust = 0.5,
            vjust = 1,
            nudge_y = -10,
            lineheight = 0.8) +
  theme(axis.text.x = element_text(angle = 90))
```




## Labeling stacked bar plots

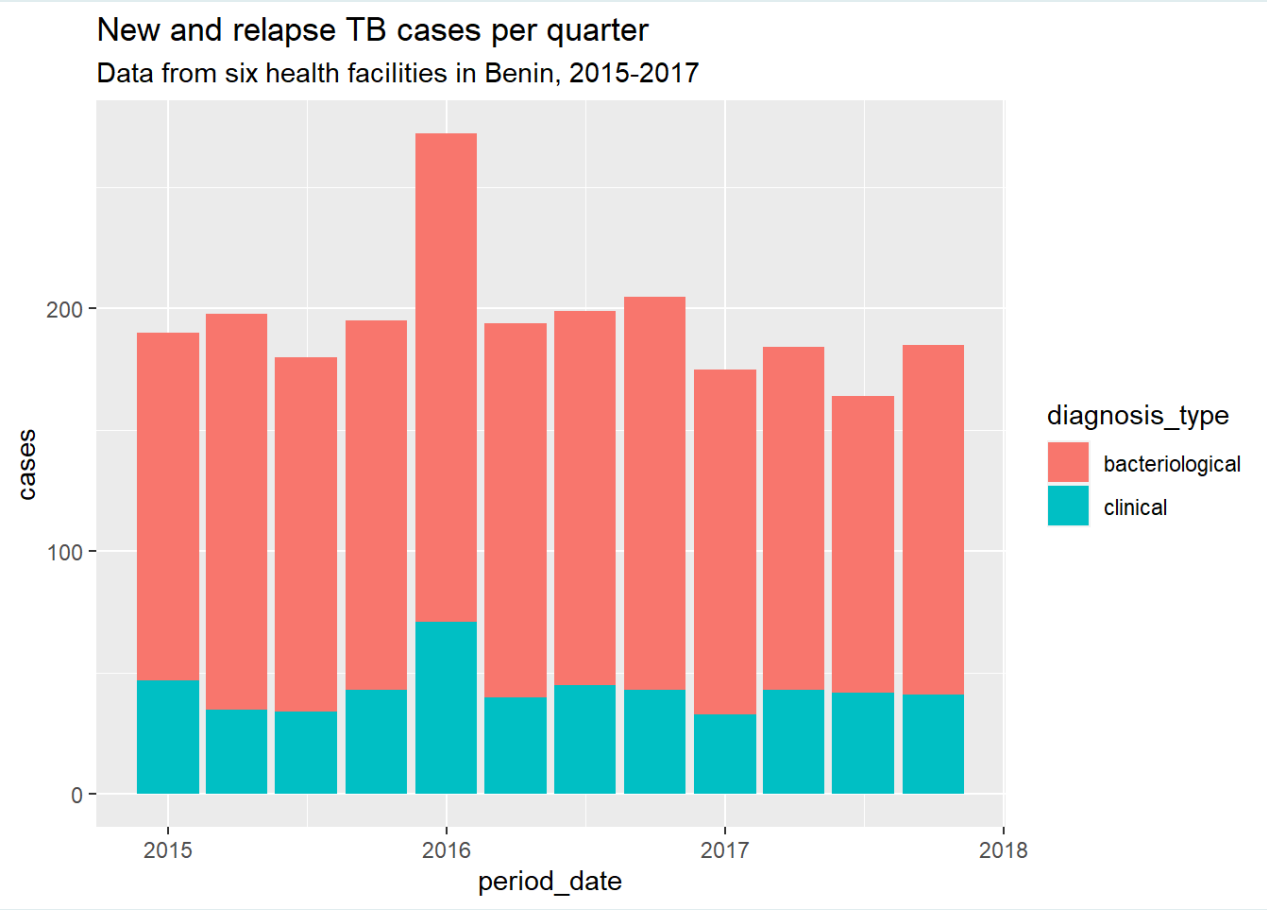
So far, we've only looked at bar plots with a single categorical variable. Let's build plots with two categorical variables and add labels to each subgroup. We'll start with stacked bar plots.

We summarize the `tb_outcomes` dataset by `period_date` and `diagnosis_type`, calculating the sum of cases (`cases`) for each group.

```
# Summarize the data by period and diagnosis type
tb_sum <- tb_outcomes %>%
  group_by(period_date, diagnosis_type) %>%
  summarise(cases = sum(cases))

tb_sum
```

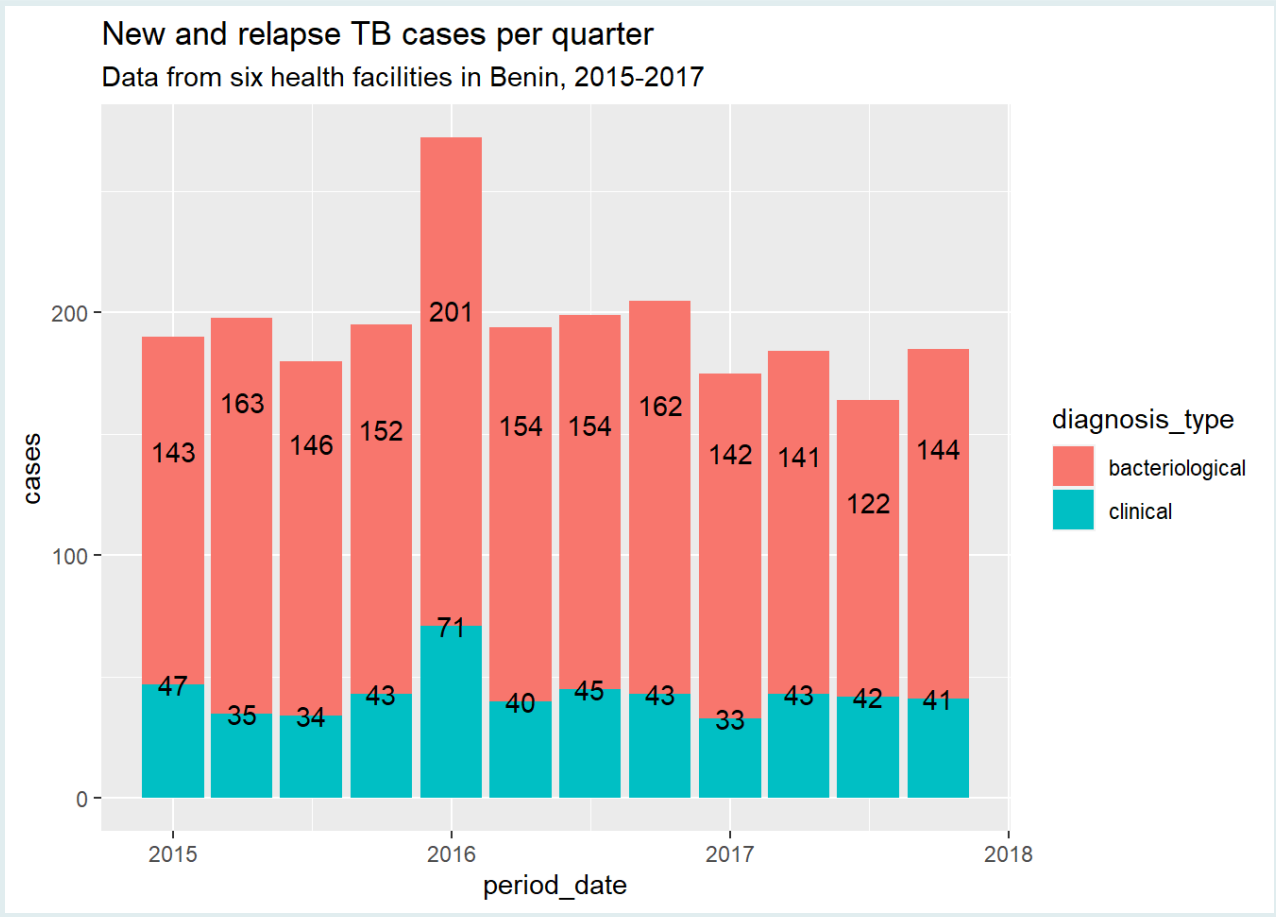
Now, let's create a **simple stacked bar plot** and see how to add labels to it:

```r
# Create a basic bar plot using the summarized data
quarter_dx_bar <- tb_sum %>%
  ggplot(aes(x = period_date, y = cases, fill = diagnosis_type)) +
  geom_col() +
  labs(title = "New and relapse TB cases per quarter",
       subtitle = "Data from six health facilities in Benin, 2015-2017")

quarter_dx_bar
```




We'll use the `cases` column for labeling each bar:

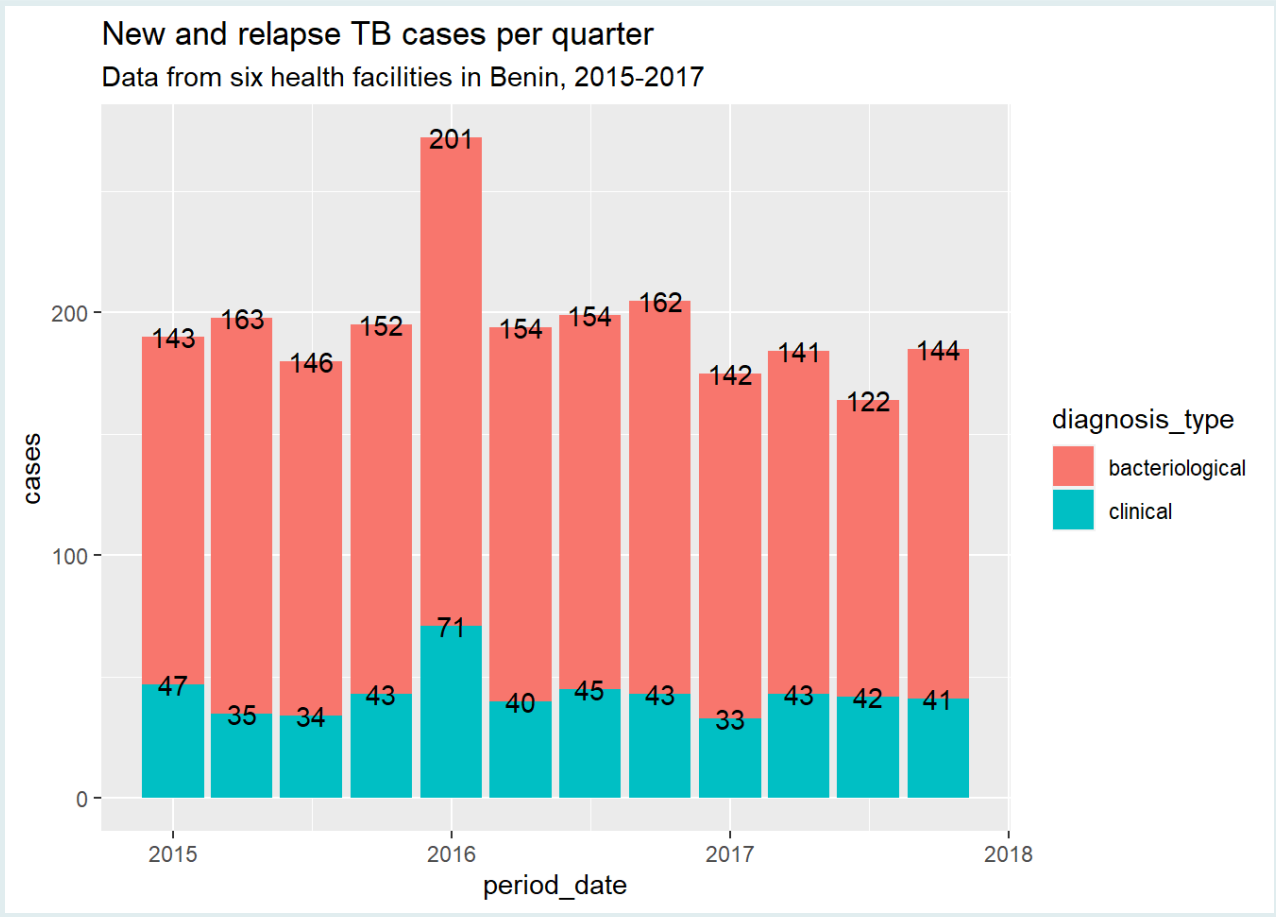
```r
# Add text labels to the bar plot
quarter_dx_bar +
  geom_text(aes(label = cases))
```




Oops, the labels are not in the right place! They don't align with the height of the bars in our plot.

The issue is that `geom_text()` does not stack positions by default like `geom_col()`. We must explicitly set `position = "stack"` in `geom_text()`:

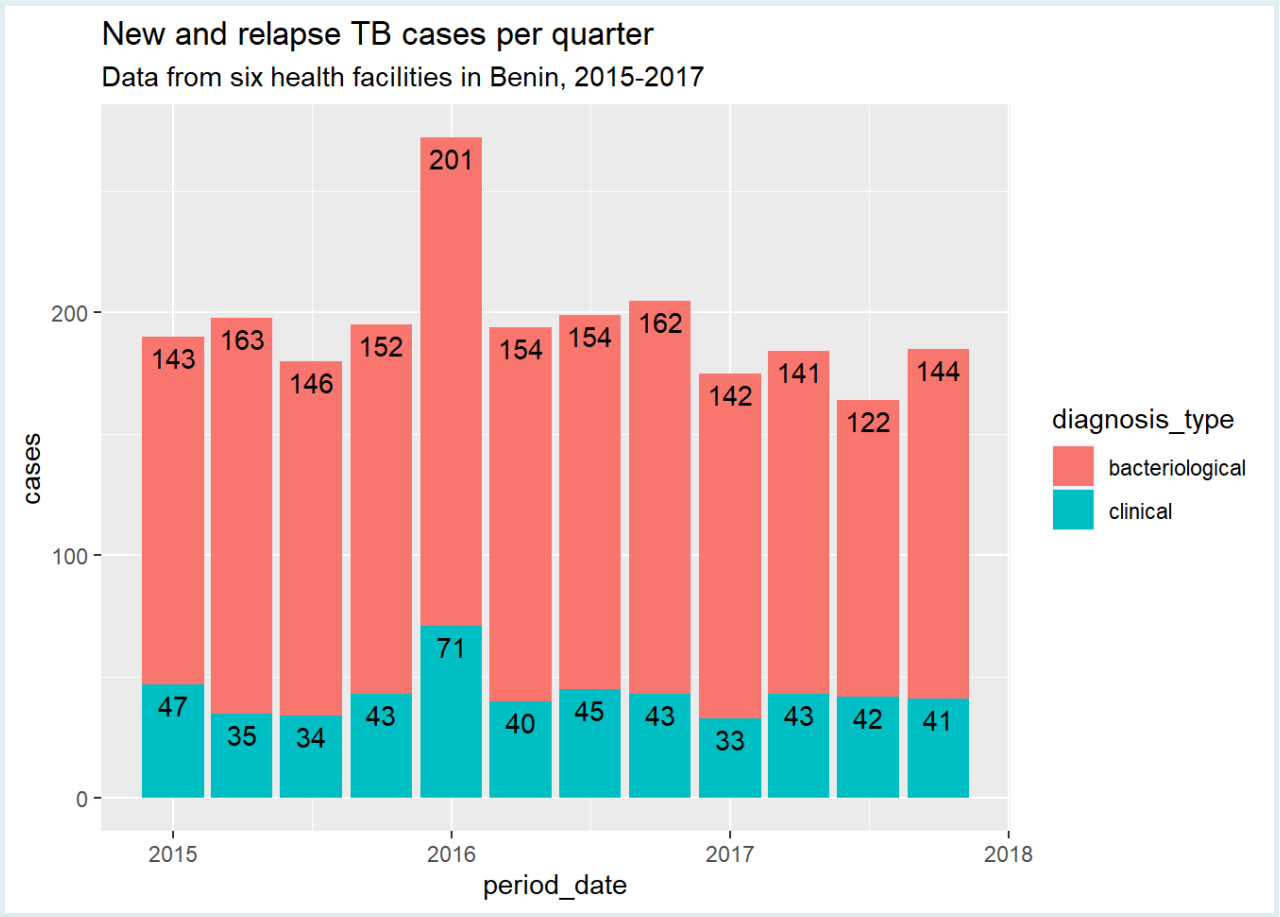
```
# Place text at the top of each bar segment
quarter_dx_bar +
  geom_text(aes(label = cases),
            position = "stack") # Set position to stack
```




Great!

To vertically align the text inside the bars, we can add `vjust` to `geom_text()`:

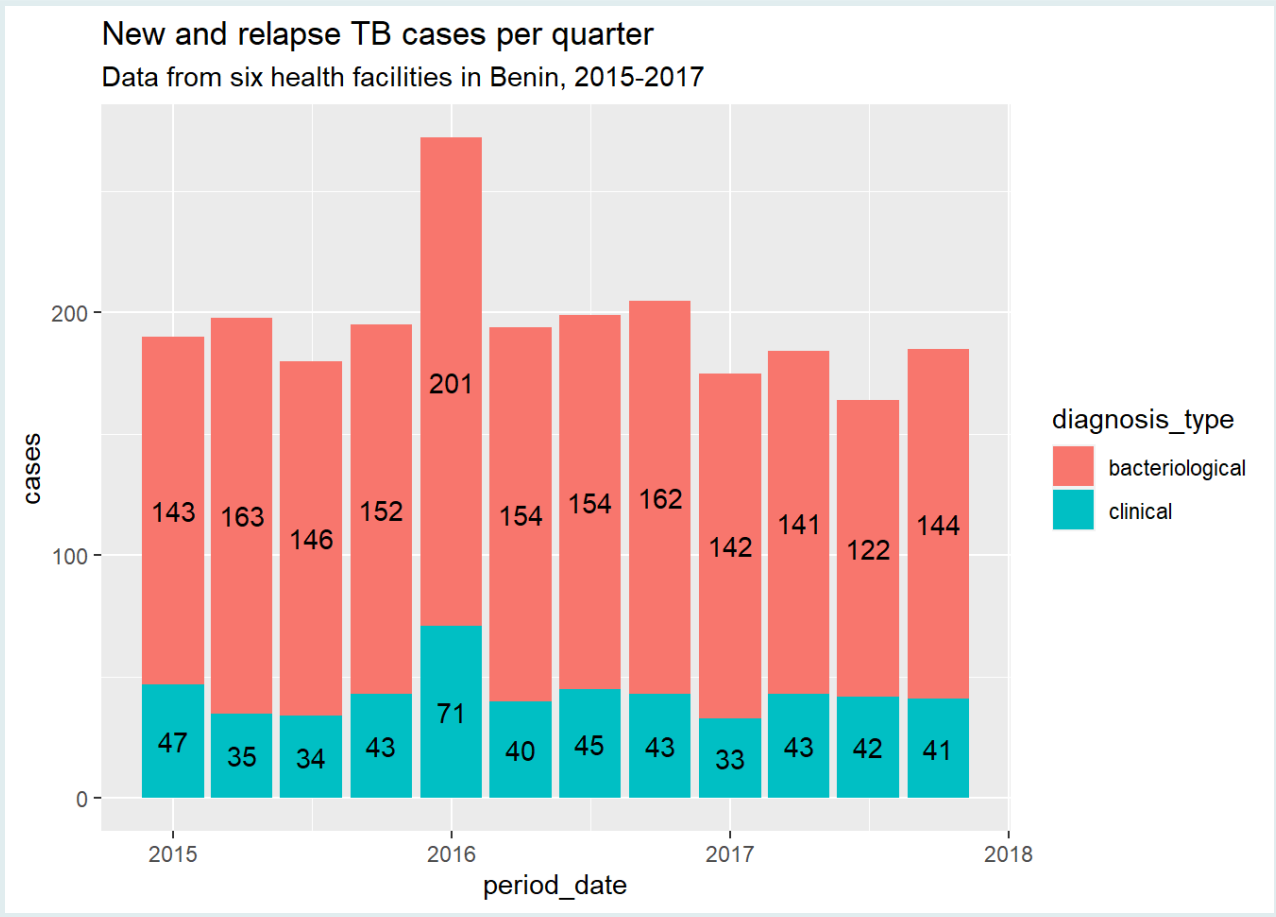
```
# Reposition labels inside the stacks for clarity and change the font style
quarter_dx_bar +
  geom_text(aes(label = cases),
    position = "stack",
    vjust = 1.5)
```




This works well, the labels are now inside the bars, and setting `vjust = 1.5` adds an extra 50% of label height as padding between the label and the top of the bar.

But what if we want to center the labels vertically within each bar segment? To do this, we switch from `position = "stack"` to the more customizable `position_stack()` function, and set `vjust = 0.5` *within* `position_stack()`:

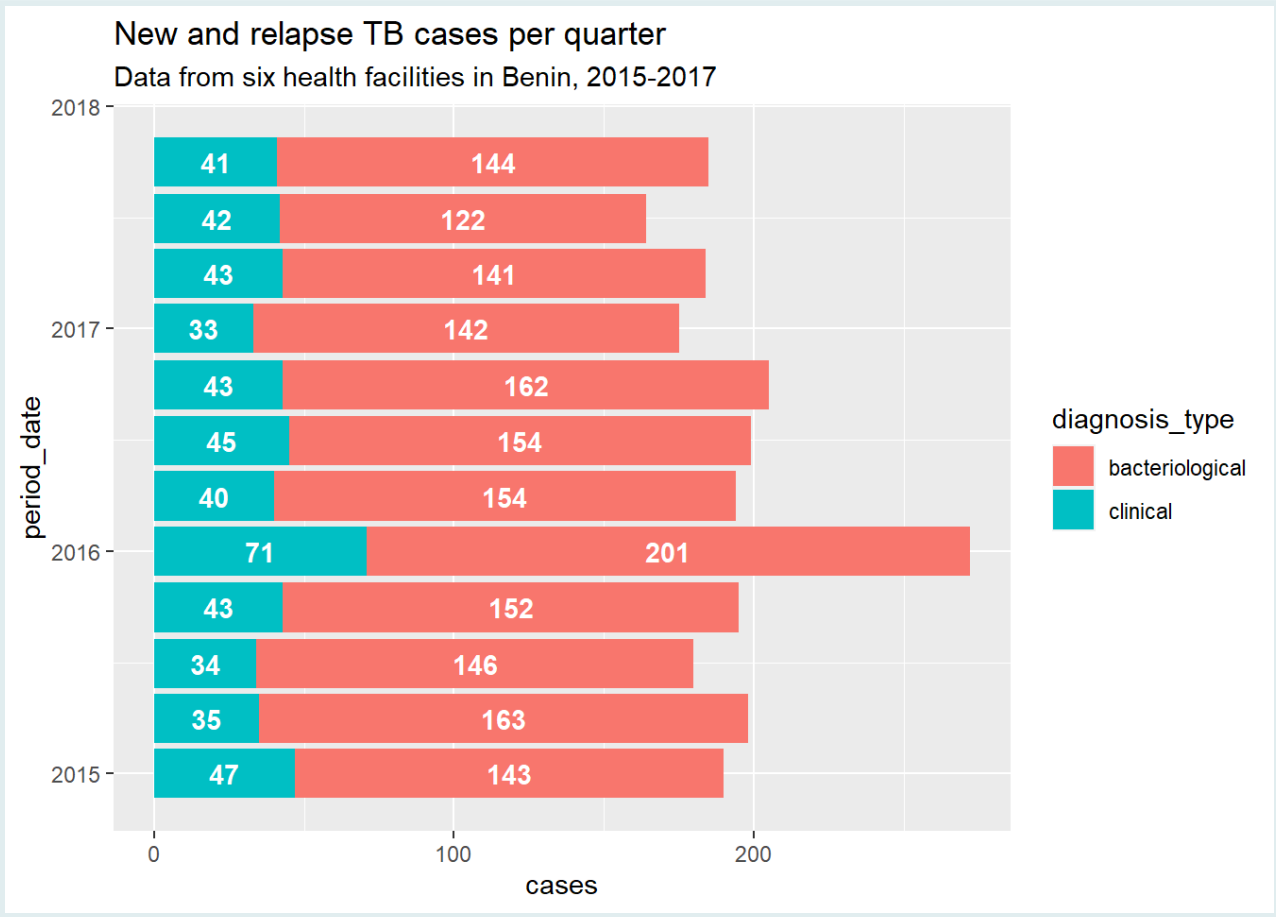
```
quarter_dx_bar +
  geom_text(aes(label = cases),
            position = position_stack(vjust = 0.5))
```




Now the labels are vertically centered within each bar segment.

This label placement is especially nice for horizontal bar plots. Below we flip the axes of our plot using `coord_flip()` to create a horizontal bar plot, and add some extra aesthetic modifications to make the plot more readable:

```
quarter_dx_bar +
  geom_text(aes(label = cases),
            position = position_stack(vjust = 0.5),
            # some extra adjustments
            color = "white",
            fontface = "bold") +
  coord_flip()
```




That looks great! Let's move on to dodged bar charts now.


Q: Practice with labeling stacked plots

Create a stacked bar plot showing the distribution per year of TB cases in rural and urban areas using the `aus_tb_notifs` dataset. Use `geom_text()` and adjust the position of the labels for clarity.

**Hint**: Pivot the data so that `area_type` is a column, then summarize the data by `year` and `area_type`, calculating the sum of cases (`cases`) for each group. The pivoting is done for you in the code below.

```
# Pivot the data
aus_tb_notifs %>%
  pivot_longer(cols = c(rural, urban),
               names_to = "area_type",
               values_to = "cases")
```

```
## # A tibble: 104 × 4
##     year quarter area_type cases
##    <dbl> <chr>   <chr>     <dbl>
##  1  2010 Q1      rural         4
##  2  2010 Q1      urban        87
##  3  2010 Q2      rural         4
##  4  2010 Q2      urban        98
##  5  2010 Q3      rural         5
##  6  2010 Q3      urban       101
##  7  2010 Q4      rural        10
##  8  2010 Q4      urban       124
##  9  2011 Q1      rural         5
## 10  2011 Q1      urban        81
## # ℹ 94 more rows
```


```
# Summarize the data by year and area type


# Create the stacked bar plot
```

## Labeling dodged bar plots

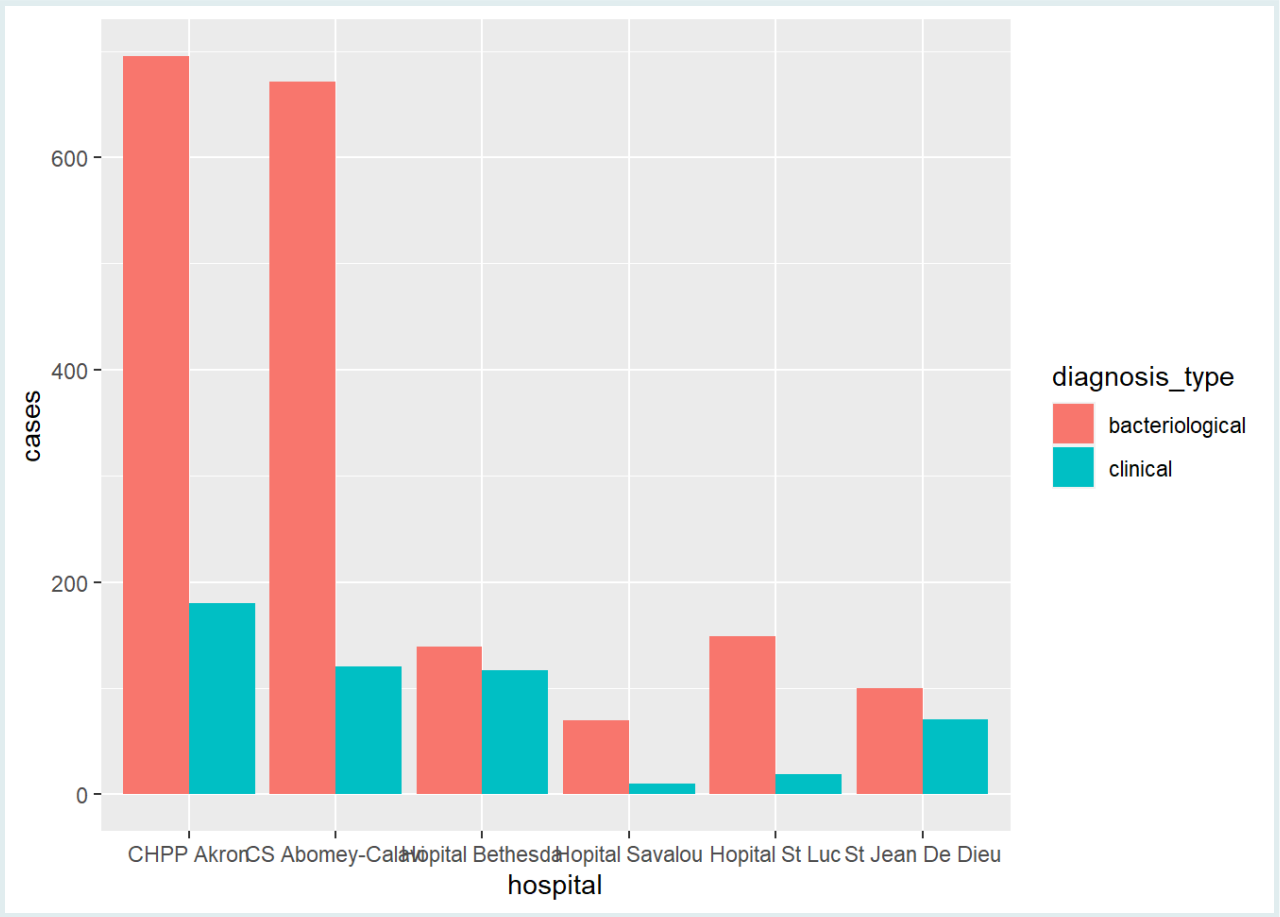
Dodged bar charts display multiple categories side by side. Let's explore how to group the data and properly position labels for clear interpretation.

To begin, we'll group our dataset `tb_outcomes` by `hospital` and `diagnosis_type`, calculating the sum of cases (`cases`) for each group.

```
hospital_dx_cases <- tb_outcomes %>%
  group_by(hospital, diagnosis_type) %>%
  summarise(cases = sum(cases))

hospital_dx_cases
```

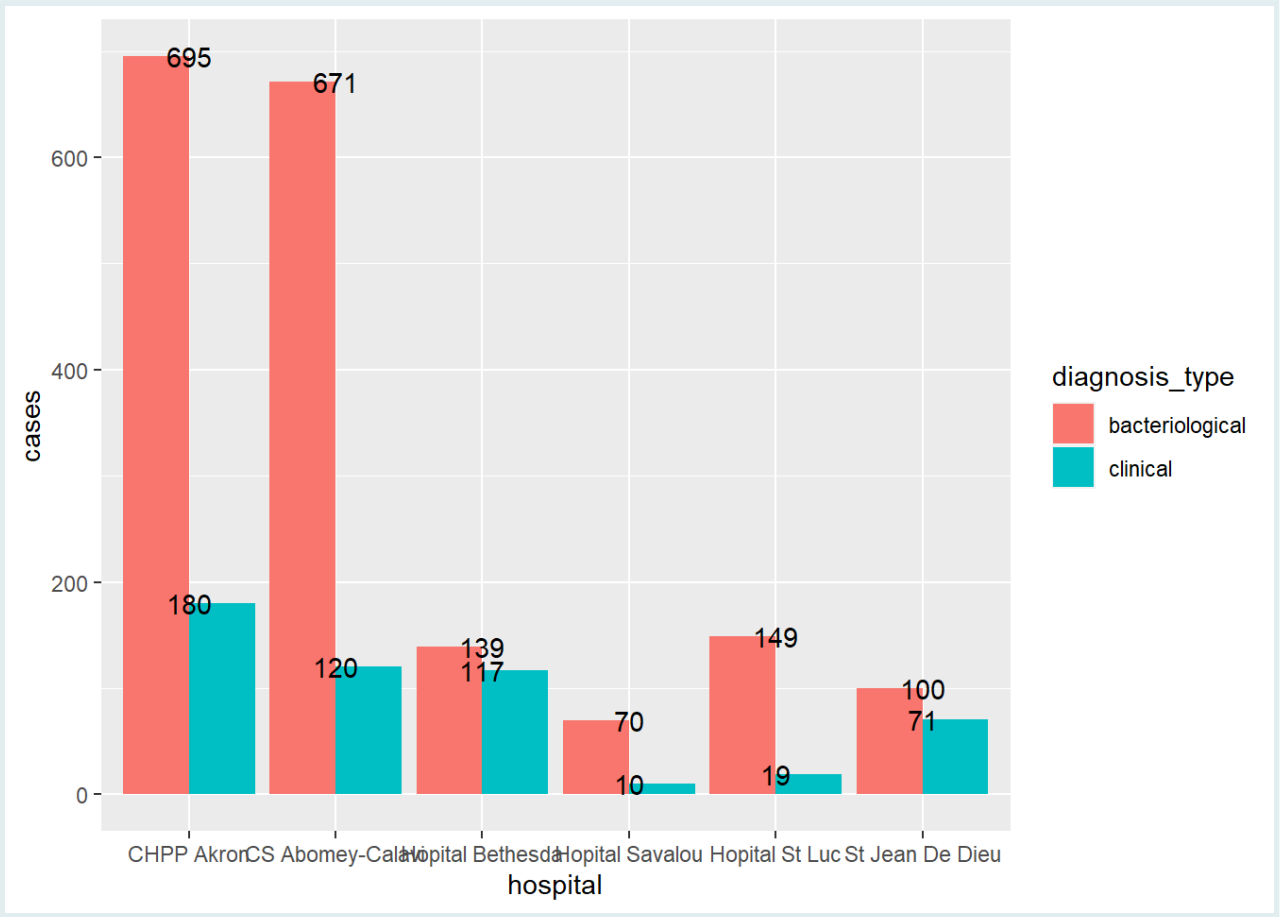
Next, let's create a simple **dodged bar chart**, where the height of each bar signifies the total number of cases for a specific diagnosis in each hospital. Since the default parameter for `geom_col` is `stack`, we must explicitly set `position = "dodge"` to create a dodged bar chart.

```
hospital_dx_bar <- hospital_dx_cases %>%
  ggplot(aes(x = hospital, y = cases, fill = diagnosis_type)) +
  geom_col(position = "dodge")

hospital_dx_bar
```




Now, we can annotate the chart with `geom_text()` to display the labels, just as we've done before.

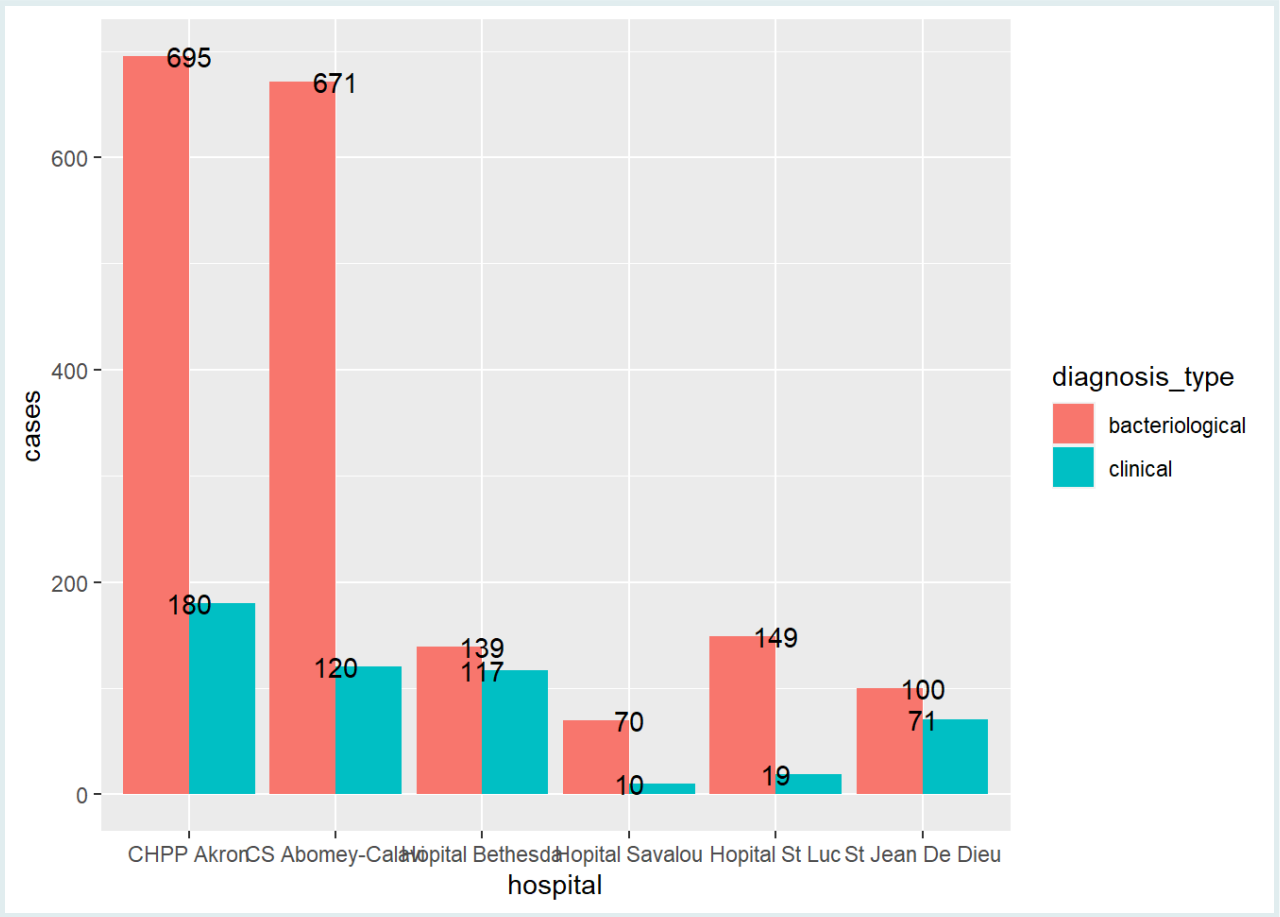
```
hospital_dx_bar +
  geom_text(aes(label = cases))
```




Oops, that's not quite right! The labels are vertically centered in a straight line, and they're not aligned with the bars. Let's take a look at how we can fix that.

Just as with our stacked bar chart in the previous section, we need to add the position adjustment to `geom_text()`. This time we're going to specify `position = position_dodge()`.

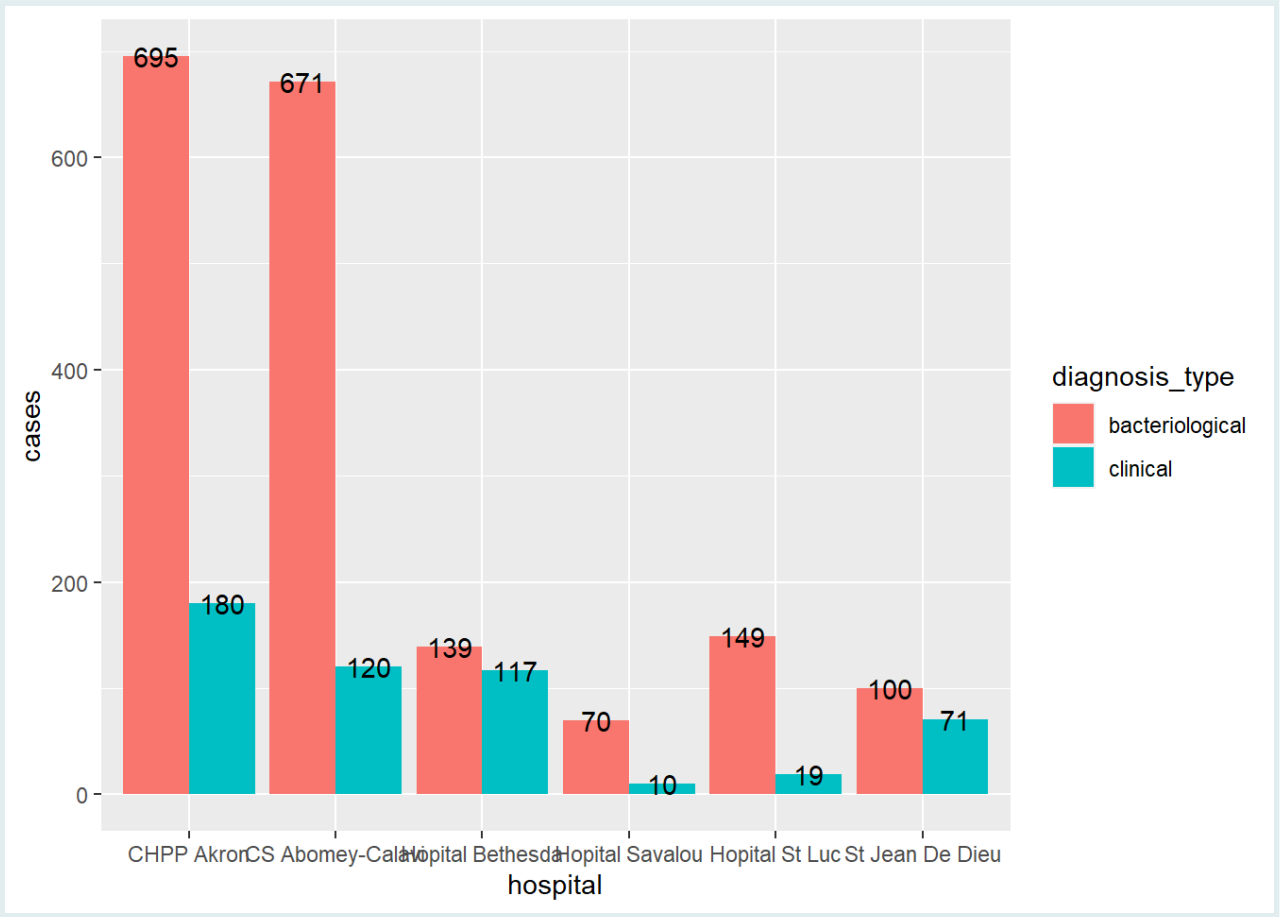
```
hospital_dx_bar +
  geom_text(aes(label = cases),
            position = position_dodge())
```




Oh no. We get the same chart as before. This is because a width argument is required for `position_dodge()`.

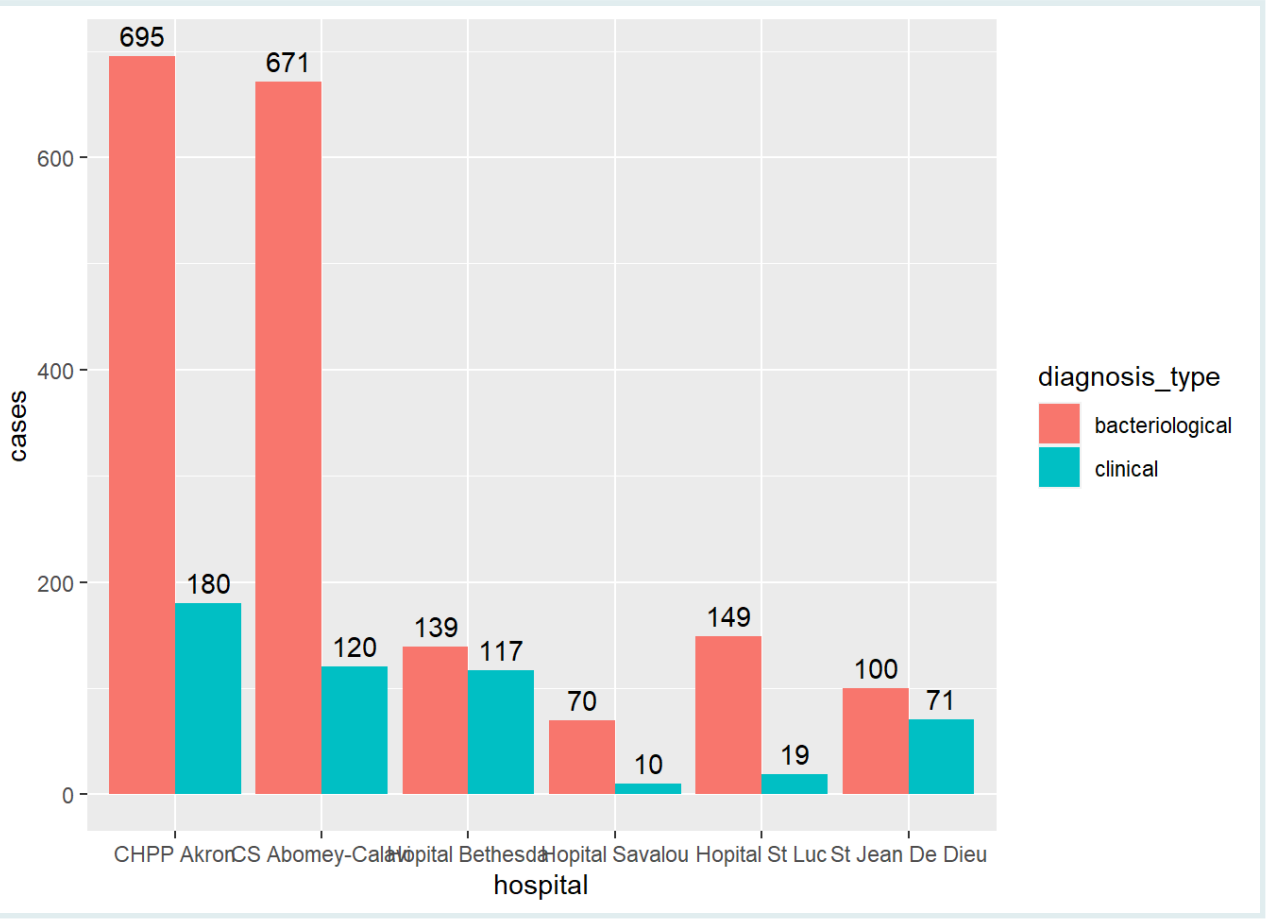
For `geom_col()`, the default value of `width` is `0.9`. We'll also use `0.9` for `geom_text()` to ensure the bars and labels are aligned:

```
hospital_dx_bar +
  geom_text(aes(label = cases),
            position = position_dodge(width = 0.9))
```




Now all that's left to do is shift the labels up a bit with vjust.

```
hospital_dx_bar +
  geom_text(aes(label = cases),
            position = position_dodge(width = 0.9),
            vjust = -0.5)
```




That looks great! Next, we'll move on to percent-stacked bar plots.


Q: Practice with labeling dodged bar plots

Generate a dodged bar plot that displays rural and urban TB cases side by side for each year using the `aus_tb_notifs` dataset. Label each bar using `geom_text()`, ensuring the labels are correctly aligned.

You can use the code and comments below as a guide:

```
# Pivot the data
aus_tb_notifs %>%
  pivot_longer(cols = c(rural, urban),
               names_to = "area_type",
               values_to = "cases")
```

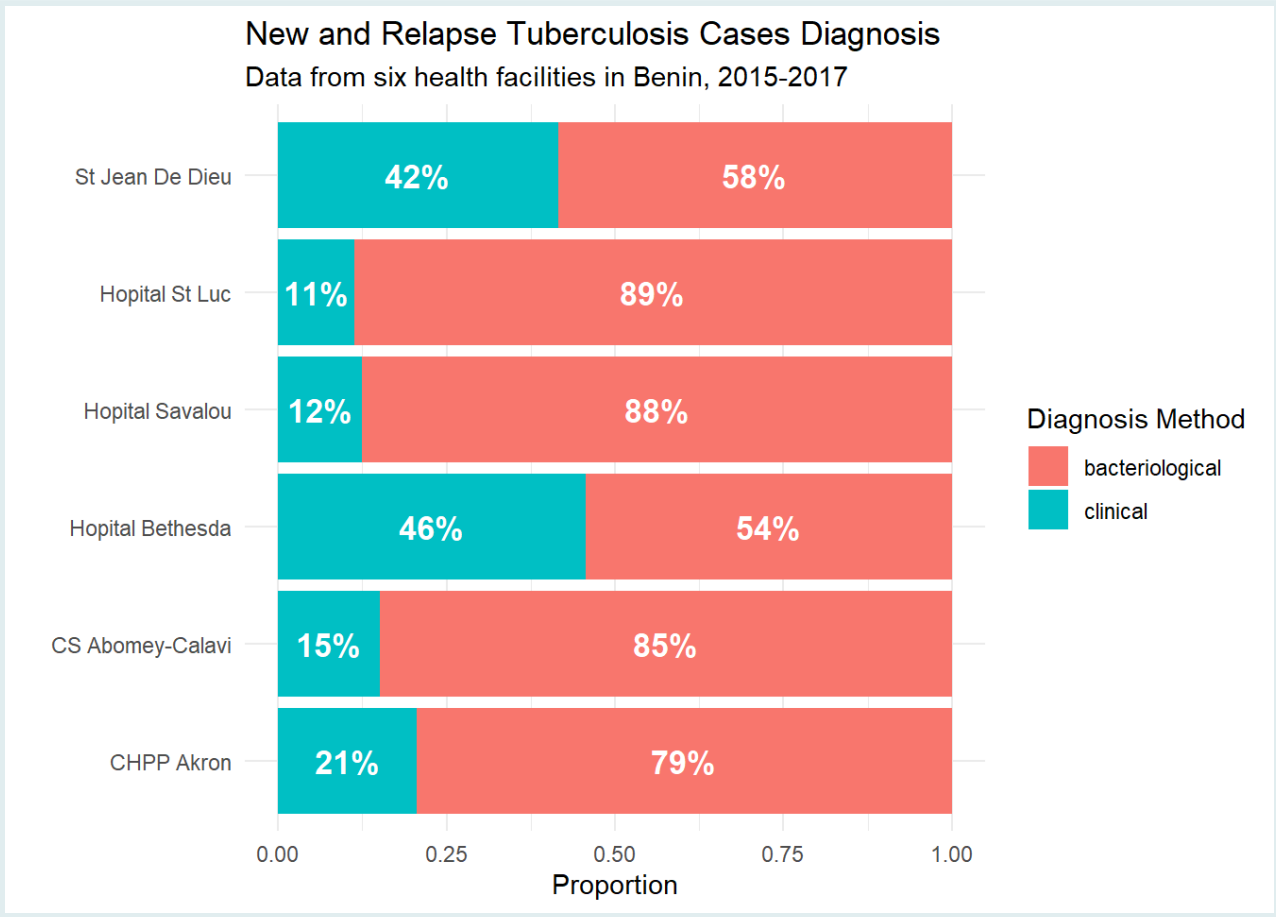
```
## # A tibble: 104 × 4
##    year quarter area_type cases
```

```
##    <dbl> <chr>   <chr>      <dbl>
##  1  2010 Q1      rural          4
##  2  2010 Q1      urban         87
##  3  2010 Q2      rural          4
##  4  2010 Q2      urban         98
##  5  2010 Q3      rural          5
##  6  2010 Q3      urban        101
##  7  2010 Q4      rural         10
##  8  2010 Q4      urban        124
##  9  2011 Q1      rural          5
## 10  2011 Q1      urban         81
## # ℹ 94 more rows
```


```
# then summarize the data by year and area type


# then create the dodged bar plot
# for the text, use position = position_dodge(width = 0.9)
```

## Labeling percent-stacked bar plots

When labeling percent-stacked bar plots, the labels should reflect the percentages of each category. This means we need to format the labels into percentages to ensure they match the segments on the chart. By the end of this section, you'll know how to create a graph like the one below!

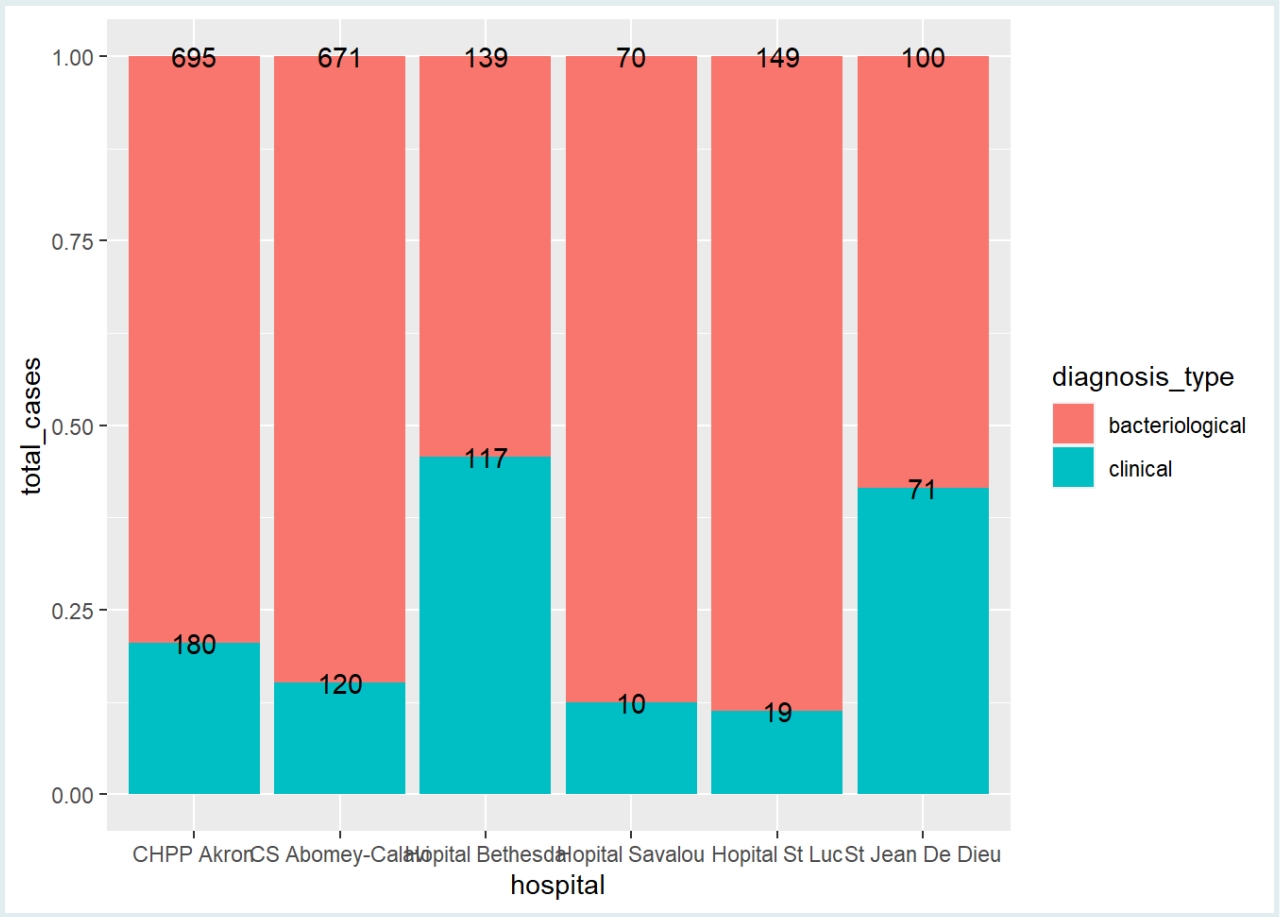



To get started, let's calculate the total number of cases for each health facility (`hospital`) by diangostic type.

```
hosp_dx_sum <- tb_outcomes %>%
  group_by(hospital, diagnosis_type) %>%
  summarise(total_cases = sum(cases))

hosp_dx_sum
```

We could use this dataset to create a percent-stacked bar plot. You may remember from the last lesson that for percent stacked plots, we need to use the `fill` position. By now, you should recognize that we want to use the more customizable `position_fill()` instead of the simpler `position = "fill"`. Let's apply this position to both the bars and the labels.

```
hosp_dx_sum %>%
  ggplot(aes(x = hospital, y = total_cases, fill = diagnosis_type)) +
  geom_col(position = position_fill()) +
  geom_text(aes(label = total_cases),
            position = position_fill())
```




This is a good start but it need some improvements. For starters, we want percentages, not raw values.

In order to prepare our data for this, we need to calculate the proportion of cases for each hospital and diagnosis type before we create the plot:

```
hosp_dx_prop <- tb_outcomes %>%
  group_by(hospital, diagnosis_type) %>%
  summarise(total_cases = sum(cases)) %>%
  mutate(prop = total_cases / sum(total_cases))

hosp_dx_prop
```

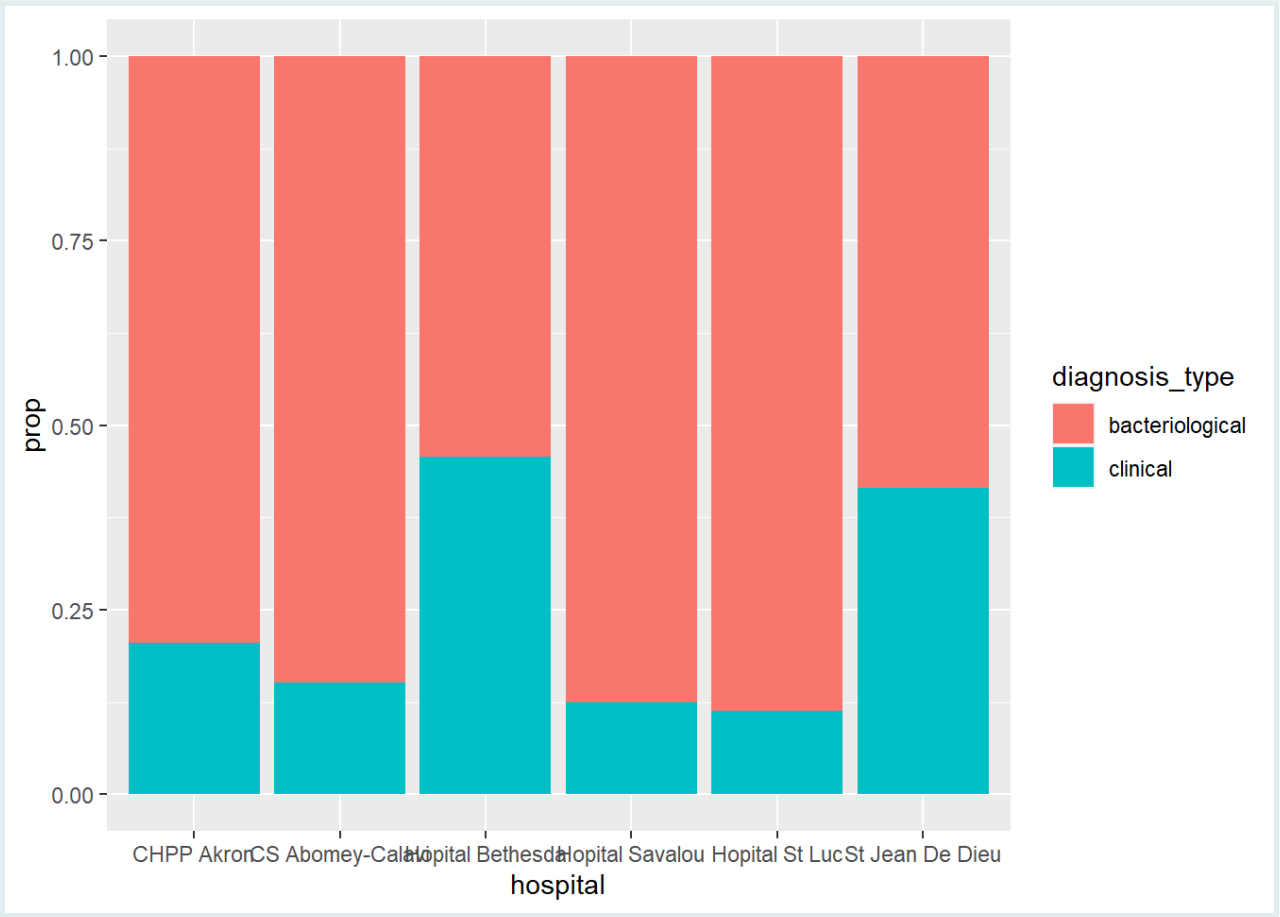
```
## # A tibble: 12 × 4
## # Groups:   hospital [6]
##    hospital         diagnosis_type  total_cases  prop
##    <chr>            <chr>                 <dbl> <dbl>
##  1 CHPP Akron       bacteriological         695 0.794
##  2 CHPP Akron       clinical                180 0.206
##  3 CS Abomey-Calavi bacteriological         671 0.848
##  4 CS Abomey-Calavi clinical                120 0.152
##  5 Hopital Bethesda bacteriological         139 0.543
##  6 Hopital Bethesda clinical                117 0.457
##  7 Hopital Savalou  bacteriological          70 0.875
```

```
##  8 Hopital Savalou  clinical                10 0.125
##  9 Hopital St Luc   bacteriological        149 0.887
## 10 Hopital St Luc   clinical                19 0.113
## 11 St Jean De Dieu  bacteriological        100 0.585
## 12 St Jean De Dieu  clinical                71 0.415
```

Now we have a proportion column, `prop`, that we can use to create our percent-stacked bar plot.

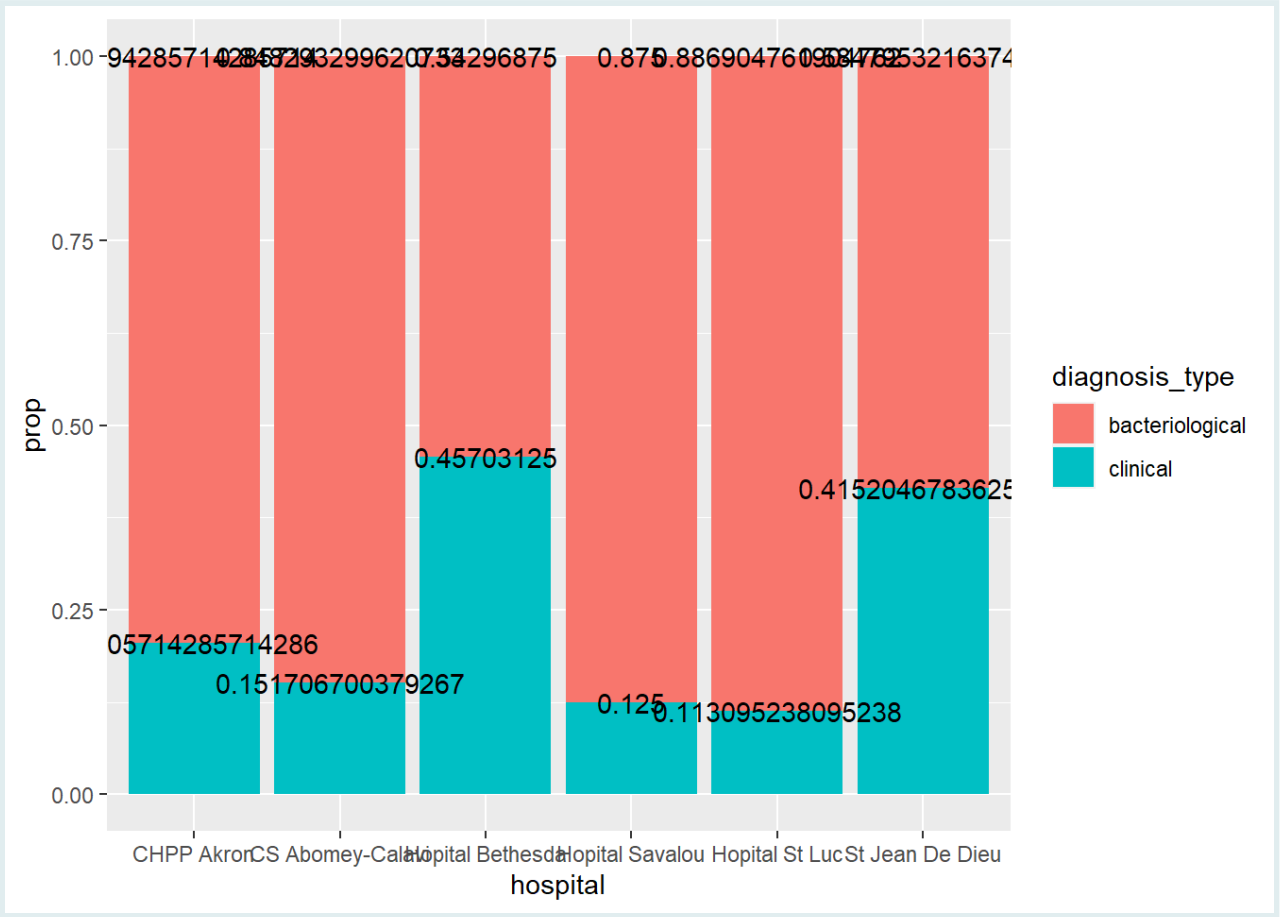
Let's create a bar chart using our new dataset `hosp_dx_prop` with `prop` as our new y variable:

```
hosp_dx_fill <- hosp_dx_prop %>%
  ggplot(aes(x = hospital, y = prop, fill = diagnosis_type)) +
  geom_col(position = position_fill())

hosp_dx_fill
```




Now, we can use `geom_text()` and specify the position to the labels:

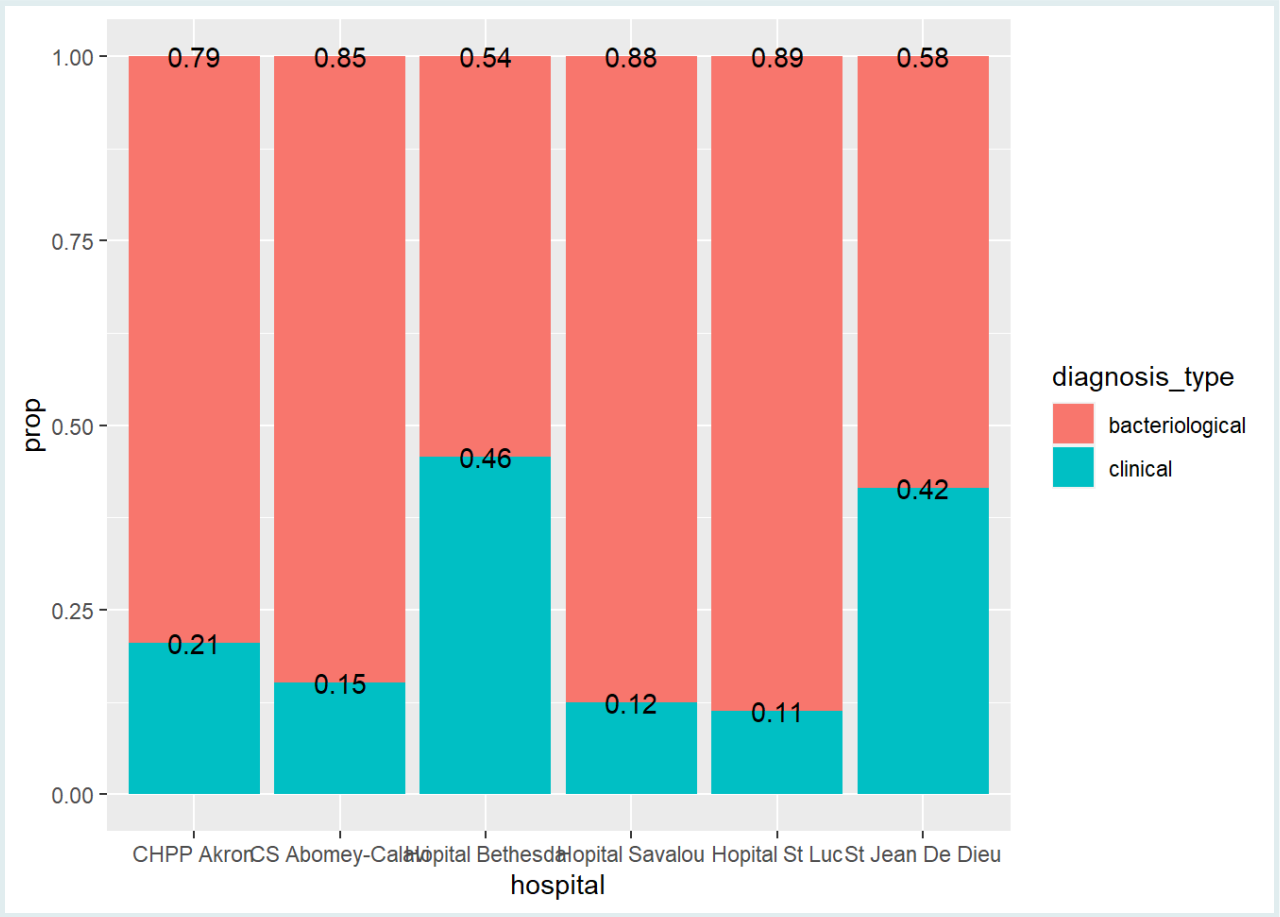
```
hosp_dx_fill +
  geom_text(aes(label = prop),
            position=position_fill())
```




It's a good start, but obviously, we still have some work to do to make it look nicer!

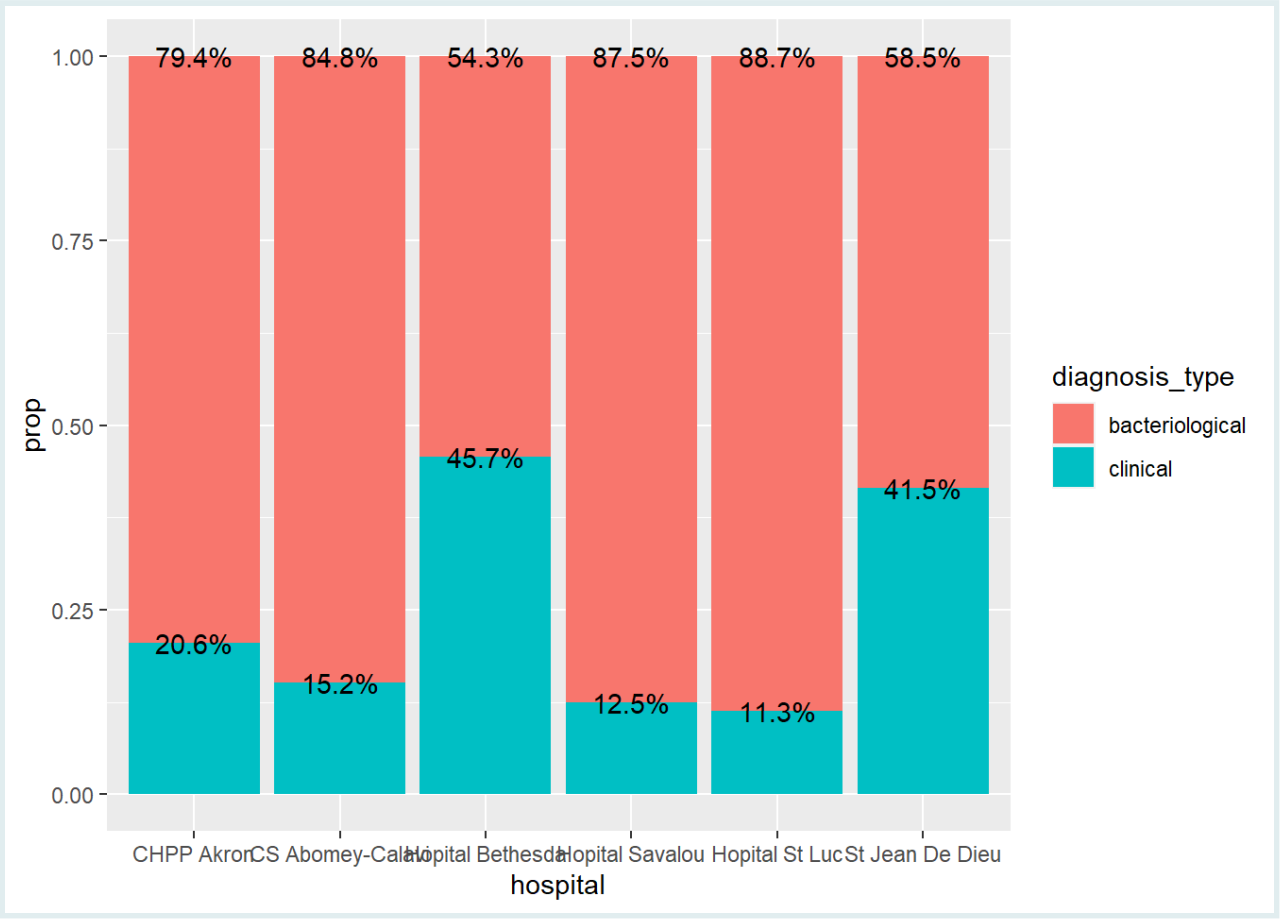
Before adjusting our labels, let's handle those decimals. We could reduce the number of decimals like this:

```
hosp_dx_fill +
  geom_text(aes(label = round(prop,2)),
            position = position_fill())
```




However, the better method is this:

```
hosp_dx_fill +
  geom_text(aes(label = scales::percent(prop)),
            position = position_fill())
```



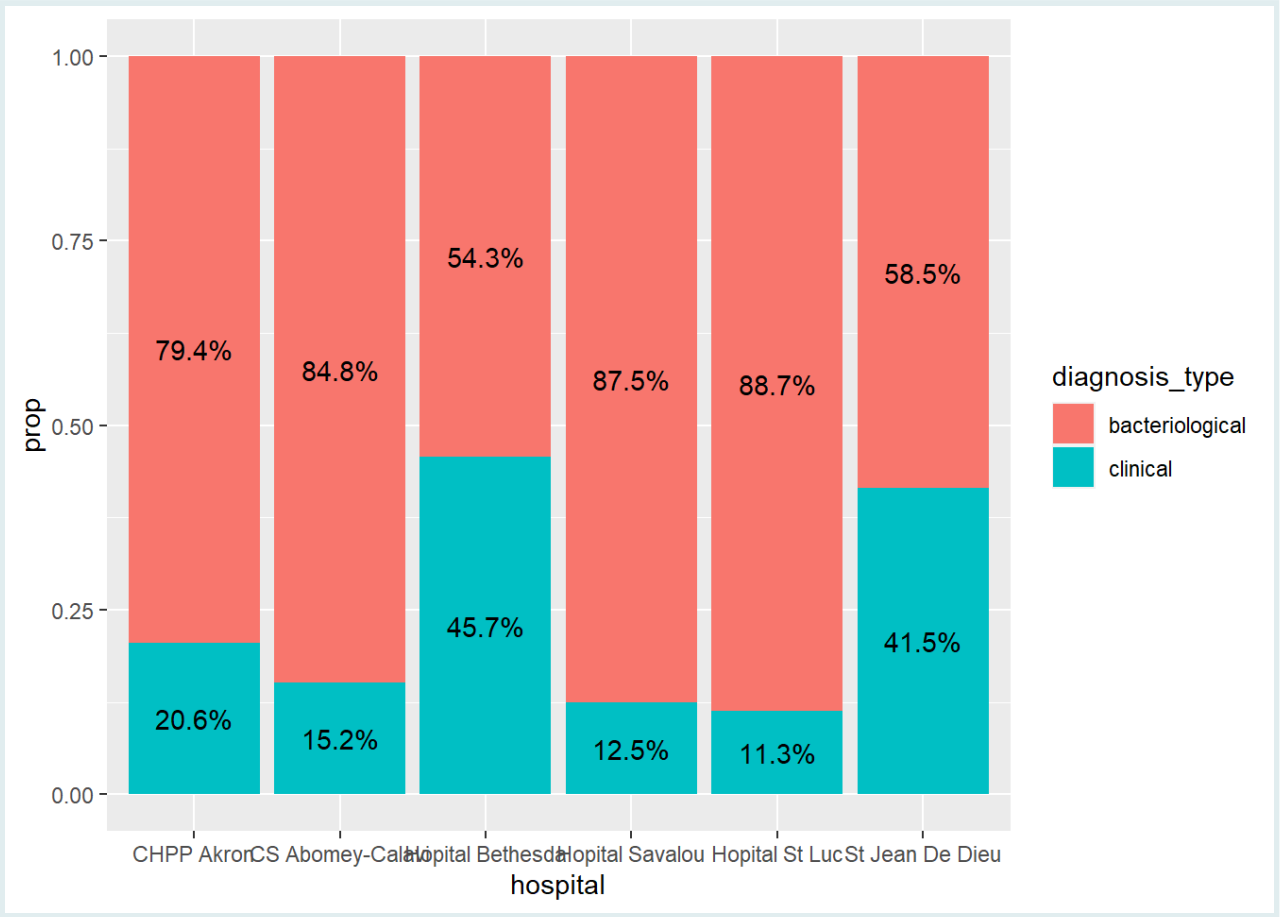

**SIDE NOTE**

The `{scales}` package is commonly used with `{ggplot2}` for customizing aesthetics, transforming axis scales, formatting labels, defining color palettes, and more.

The `scales::percent(prop)` function we used in the code above with `geom_text()` converts the proportions (values from our `prop` variable) into a percentage format and adds percentage signs. We can also control the number of displayed digits using the `accuracy` argument (see below).

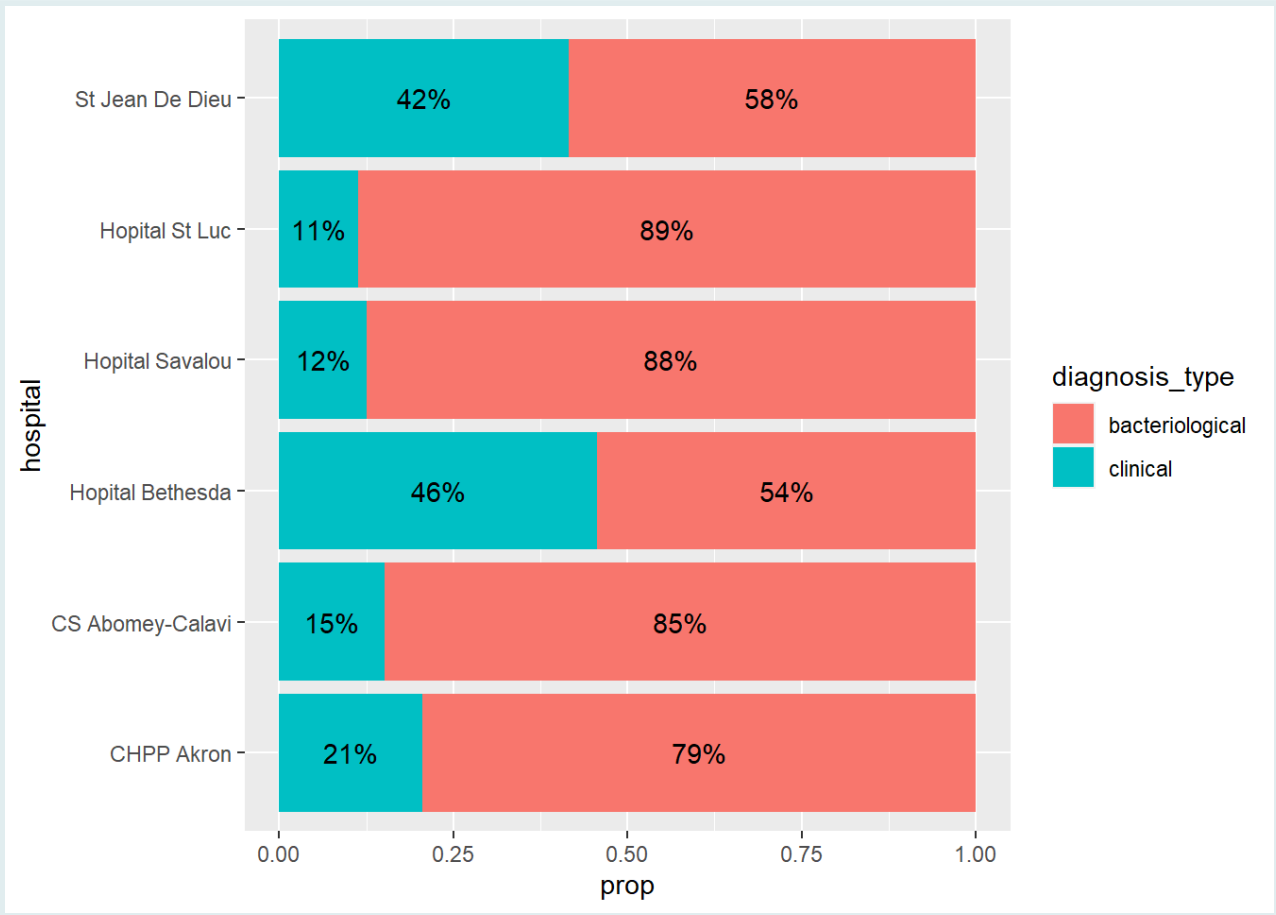
Next, we can center the labels using `vjust` in the `position_fill()` function

```
hosp_dx_fill +
  geom_text(aes(label = scales::percent(prop)),
            position = position_fill(vjust = 0.5)) # center labels
```



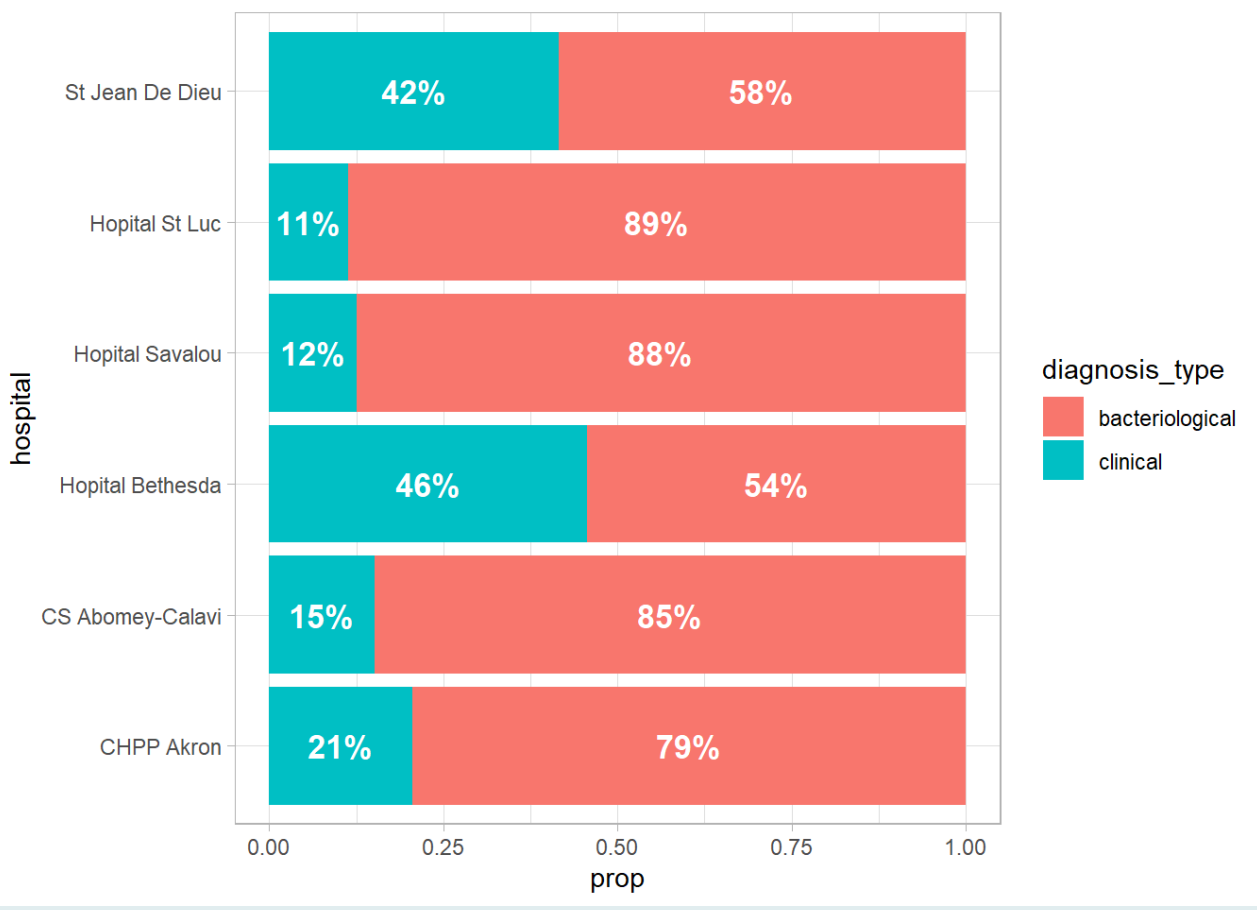

It looks great, but we can do better! Using flipped coordinates in bar charts can greatly improve readability:

```
hosp_dx_fill +
  geom_text(aes(label = scales::percent(prop, accuracy = 1)),
            position = position_fill(vjust = 0.5)) +
  coord_flip()
```




Great, now we can add some additional aesthetic tweaks:

```
hosp_dx_fill +
  geom_text(aes(label = scales::percent(prop, accuracy = 1)),
            position = position_fill(vjust = 0.5),
            color = "white", # Change text color
            fontface = "bold", # Make it bold
            size = 4.5) + # Change font size
  theme_light() +
  coord_flip()
```




Amazing! Let's move on to our last section where we'll take a look at circular plots.


### Q: Creating Percent-Stacked Bar Plots with Labels

Transform the `aus_tb_notifs` data into a percent-stacked bar plot, with a bar for each year, and the fill aesthetic mapped to the area type (rural vs urban).

Label each segment with the percentage of cases using `geom_text()`. Format the labels as percentages.

You can use the code and comments below as a guide:

```
# Pivot the data
aus_tb_notifs %>%
  pivot_longer(cols = c(rural, urban),
               names_to = "area_type",
               values_to = "cases")
```

```
## # A tibble: 104 × 4
##     year quarter area_type cases
##    <dbl> <chr>   <chr>     <dbl>
##  1  2010 Q1      rural         4
##  2  2010 Q1      urban        87
##  3  2010 Q2      rural         4
##  4  2010 Q2      urban        98
##  5  2010 Q3      rural         5
##  6  2010 Q3      urban       101
##  7  2010 Q4      rural        10
##  8  2010 Q4      urban       124
##  9  2011 Q1      rural         5
## 10  2011 Q1      urban        81
## # ℹ 94 more rows
```


```
# Then summarize and calculate proportions



# Next create the percent-stacked bar plot
# For the label, use the scales::percent() function with an
        accuracy of 1
# Use position_fill() to center the labels
```

## Labeling circular plots

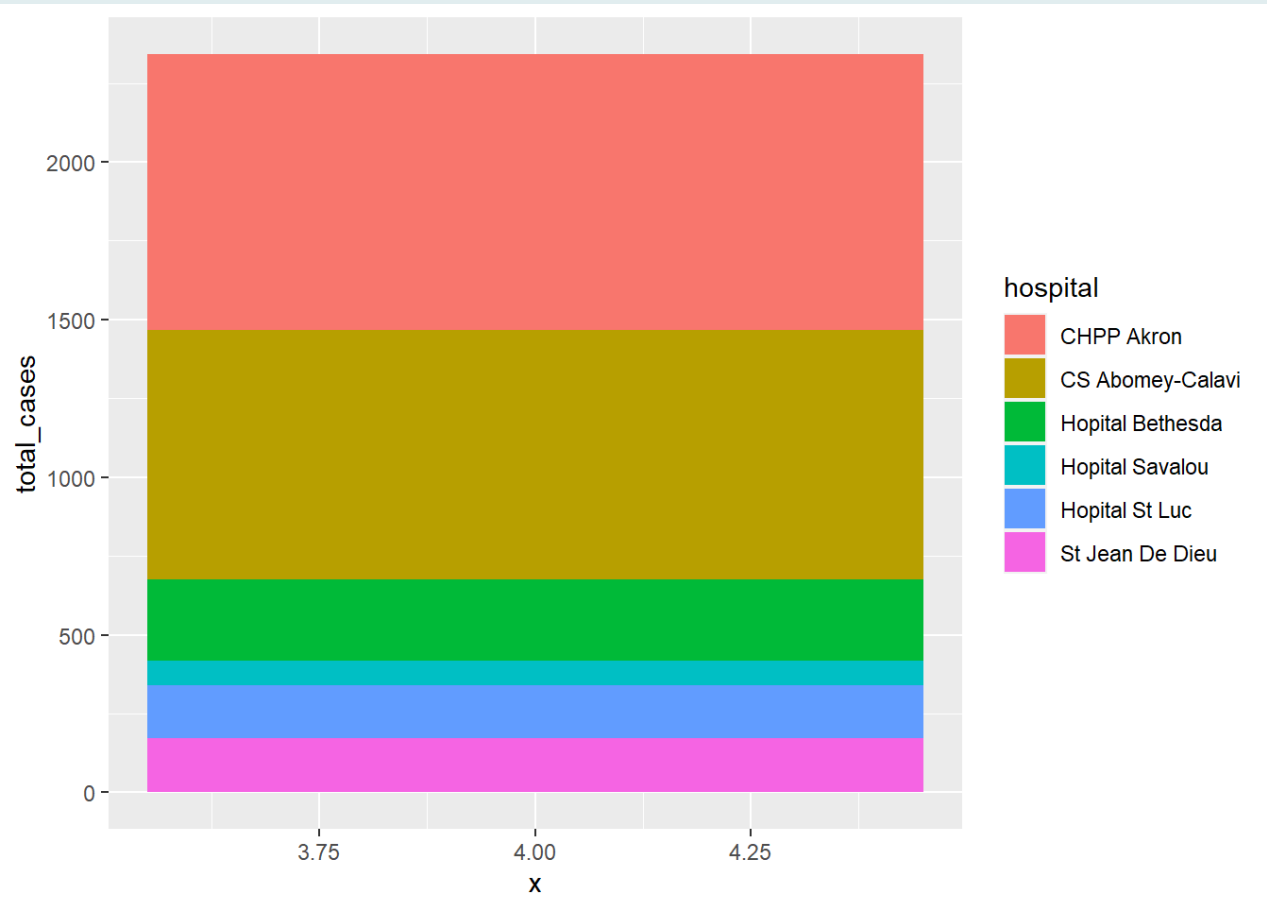
As usual, let's begin by summarizing the data.

We'll calculate the total number of cases for each hospital by grouping the data based on the `hospital` variable and then calculating the sum of cases in each group.

```
total_results <- tb_outcomes %>%
  group_by(hospital) %>%
  summarise(
    total_cases = sum(cases))

total_results
```

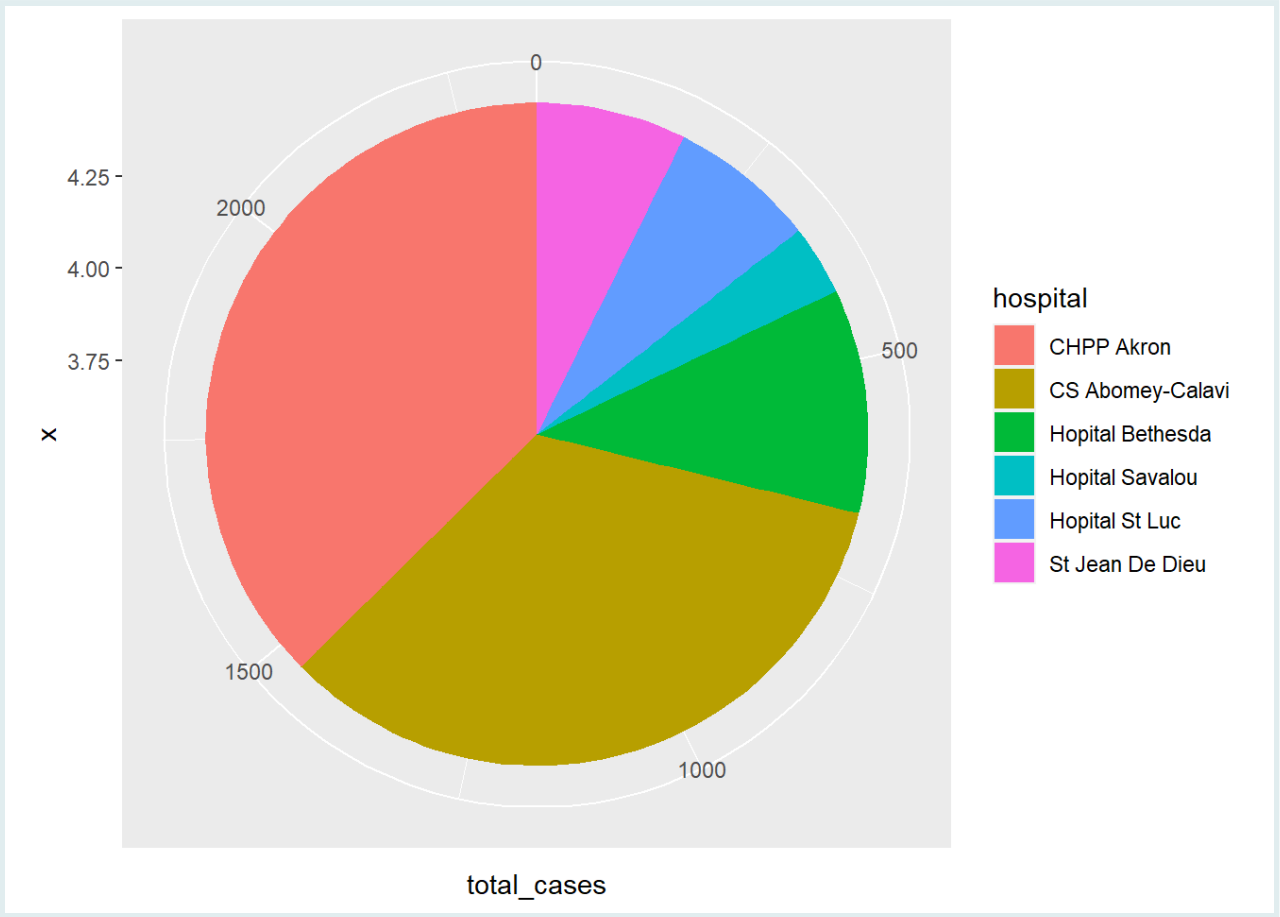
Now that we have our new dataset, let's start by creating a simple bar chart. You may recall from the previous lesson that a pie chart is essentially a round version of a 100% stacked bar chart.

```
results_stack <- ggplot(total_results,
       aes(x=4, # Set an arbitrary x value
           y=total_cases,
           fill=hospital)) +
  geom_col()

results_stack
```



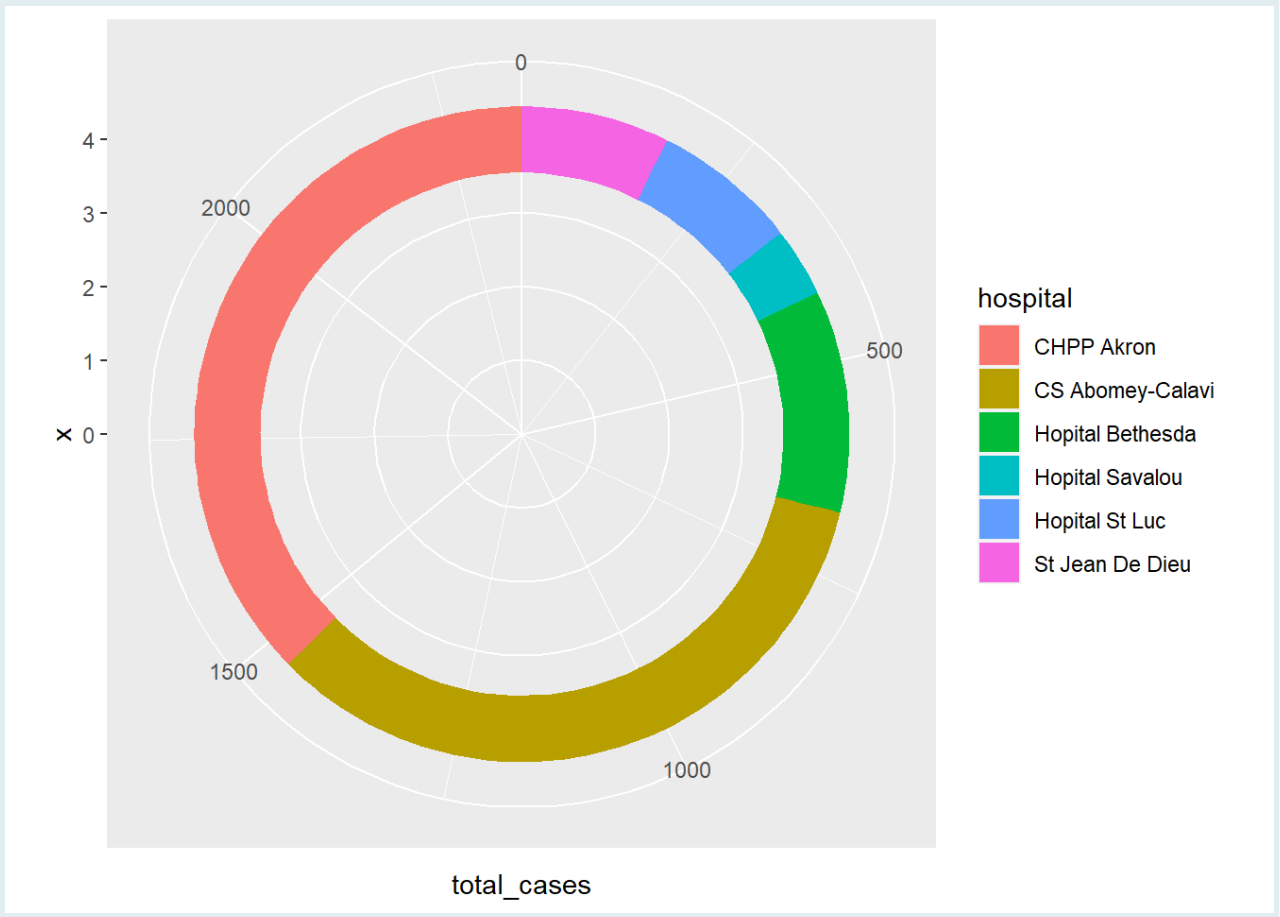

Now, we can create our basic pie chart. As we learned in the last lesson, to transform linear coordinates into polar coordinates, we use the `coord_polar()` function. The `theta` parameter defines which aesthetic variable should be mapped to the angular coordinate in the polar coordinate system. By specifying `"y"`, we use the height of the bars to determine the angle of each slice in our pie chart.

```
outcome_pie <- results_stack +
  coord_polar(theta = "y")

outcome_pie
```




Great! This will serve as our base pie chart. Next, let's create a base donut chart using `xlim()`.

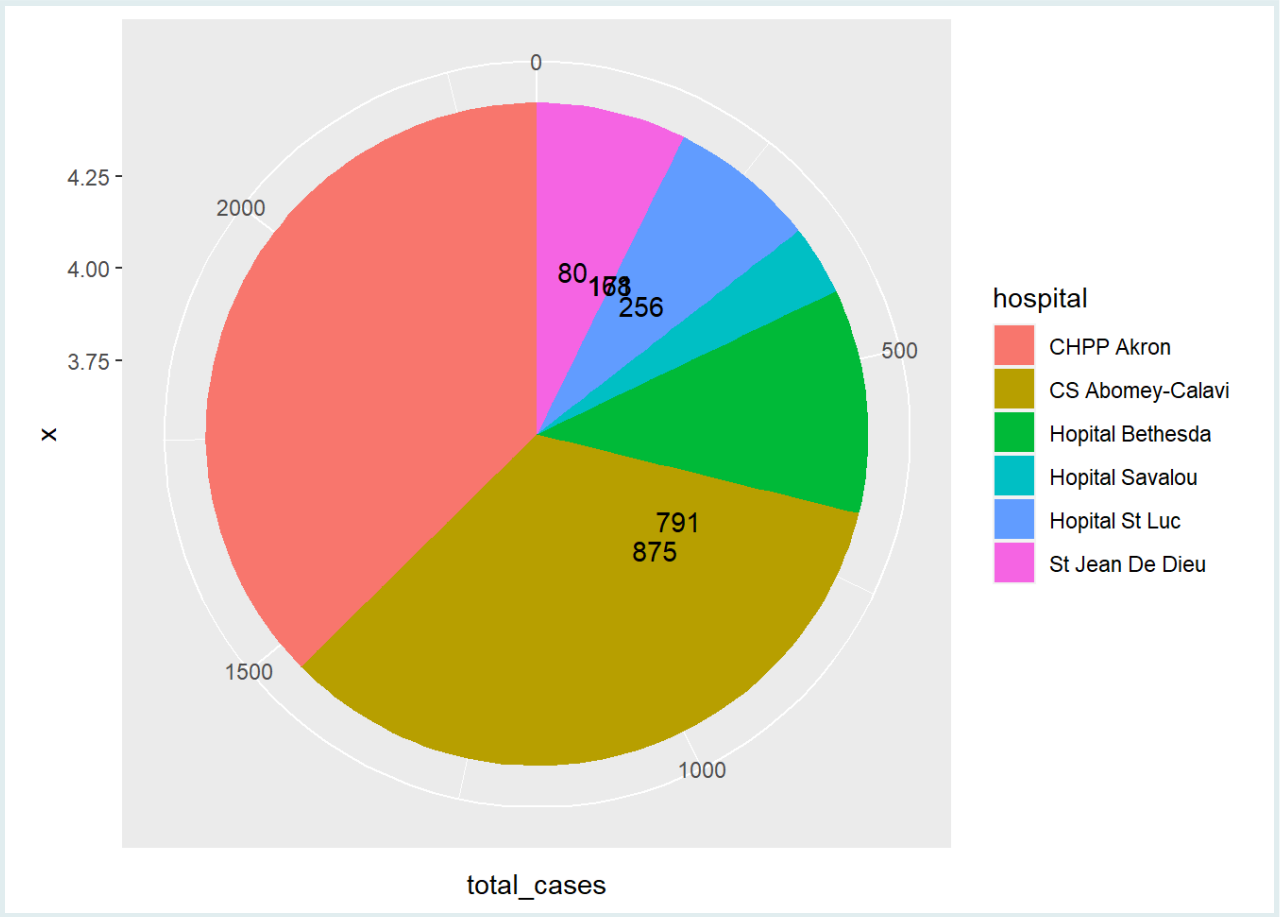
```
outcome_donut <- ggplot(total_results,
       aes(x = 4,
           y = total_cases,
           fill = hospital)) +
  geom_col() +
  xlim(c(0, 4.5)) +
  coord_polar(theta = "y")

outcome_donut
```




Alright, we're ready to move on to labelling!

Let's add labels to our pie chart using `geom_text()`.

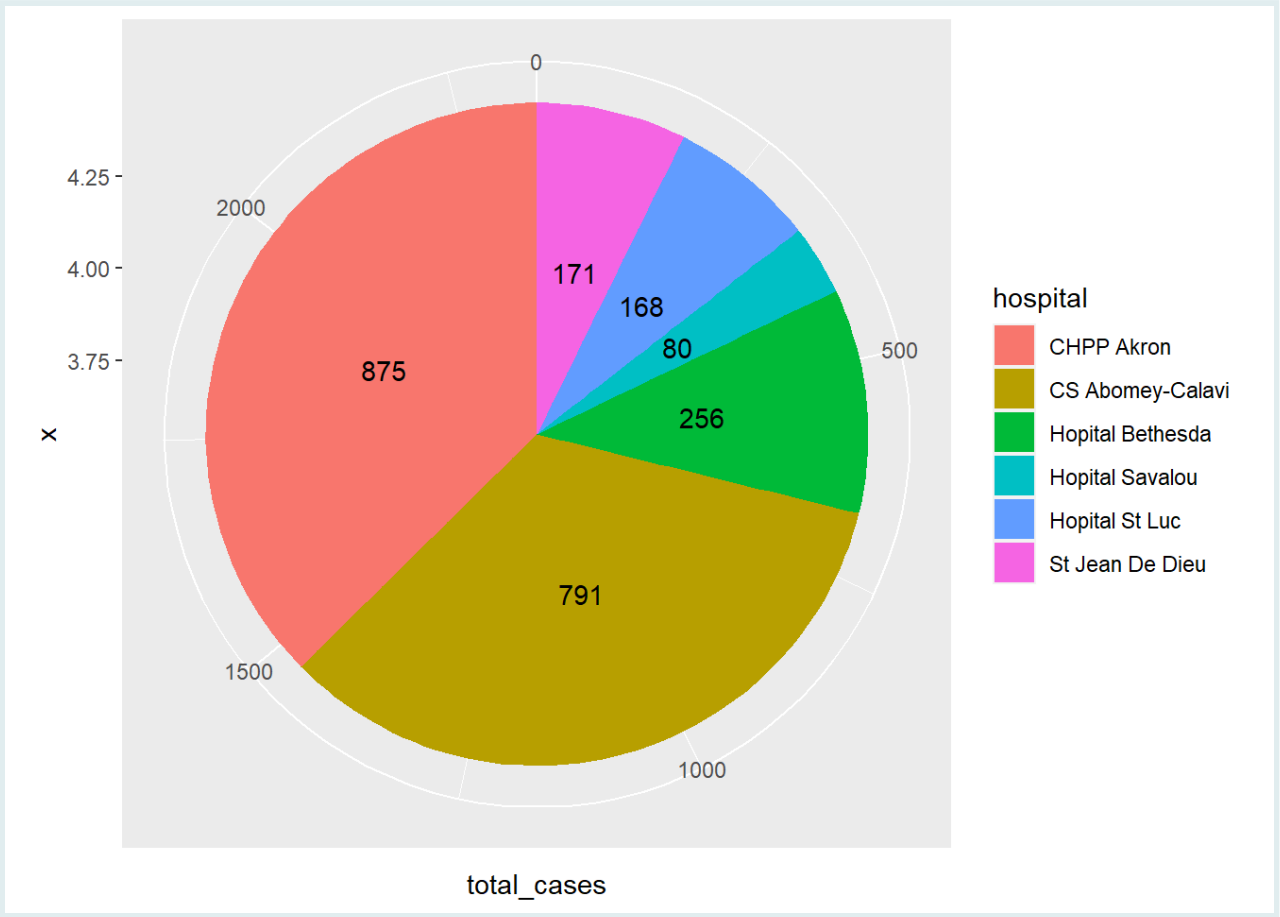
```
outcome_pie +
  geom_text(aes(label = total_cases))
```




You'll notice that the numbers appear in the wrong segments because we haven't added a `position` adjustment to the labeling geometry yet.

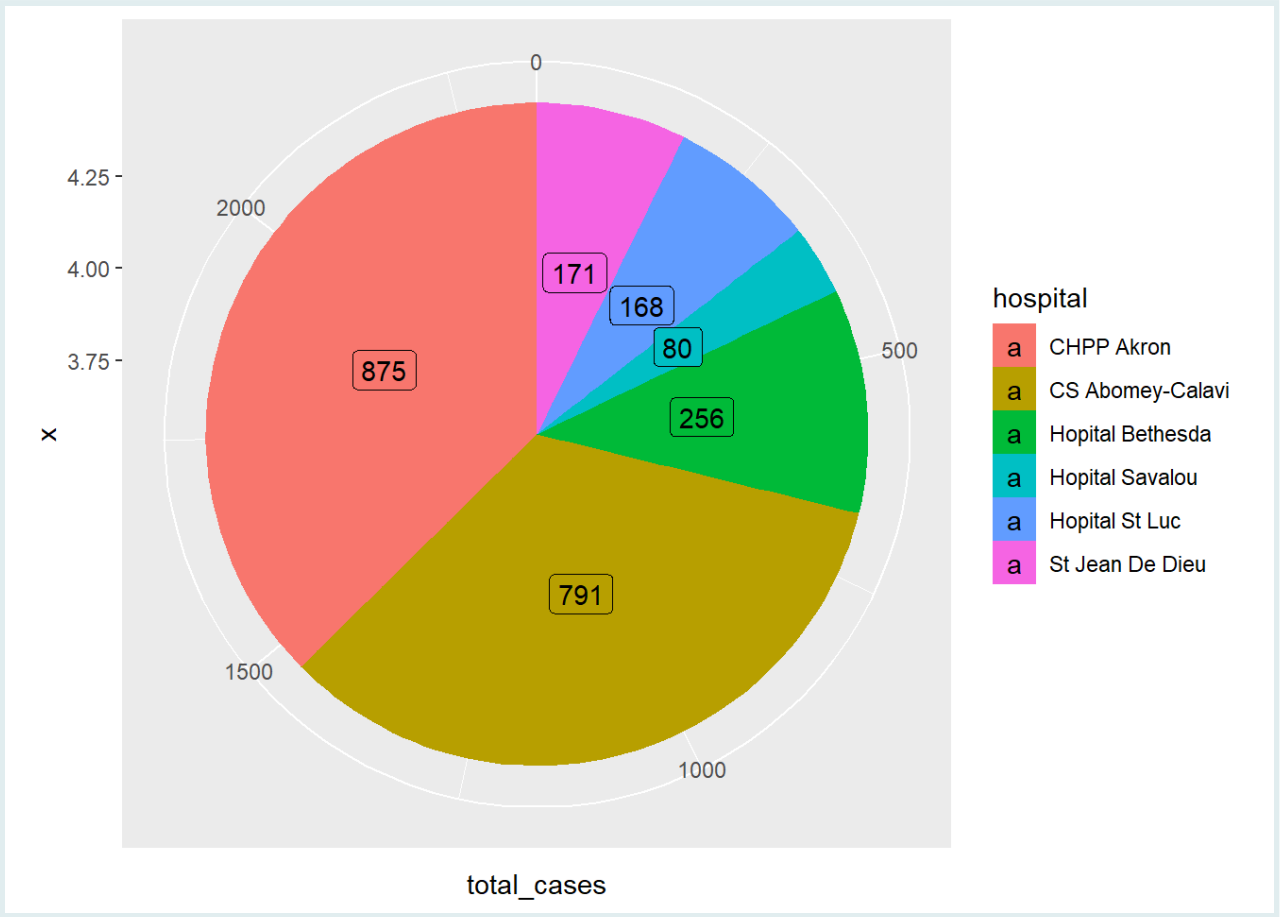
Now, just as we did previously, we will use the `position_stack()` argument with `vjust` to center the labels.

```
outcome_pie +
  geom_text(aes(label = total_cases),
            position = position_stack(vjust = 0.5)) # Center the labels
```



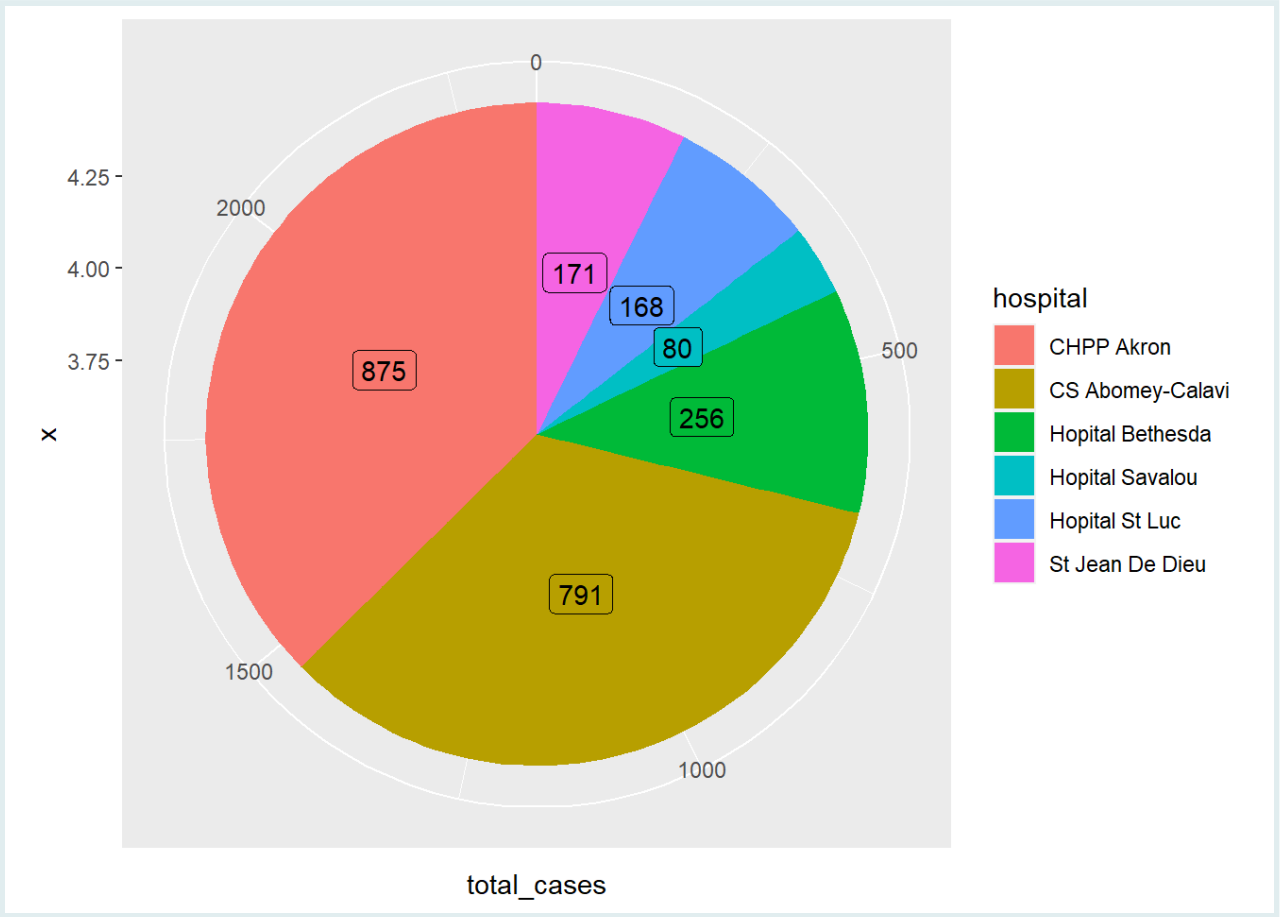

We can do the same with `geom_label()`.

```
# Similar adjustment with geom_label()
outcome_pie +
  geom_label(aes(label = total_cases),
             position = position_stack(vjust = 0.5))
```



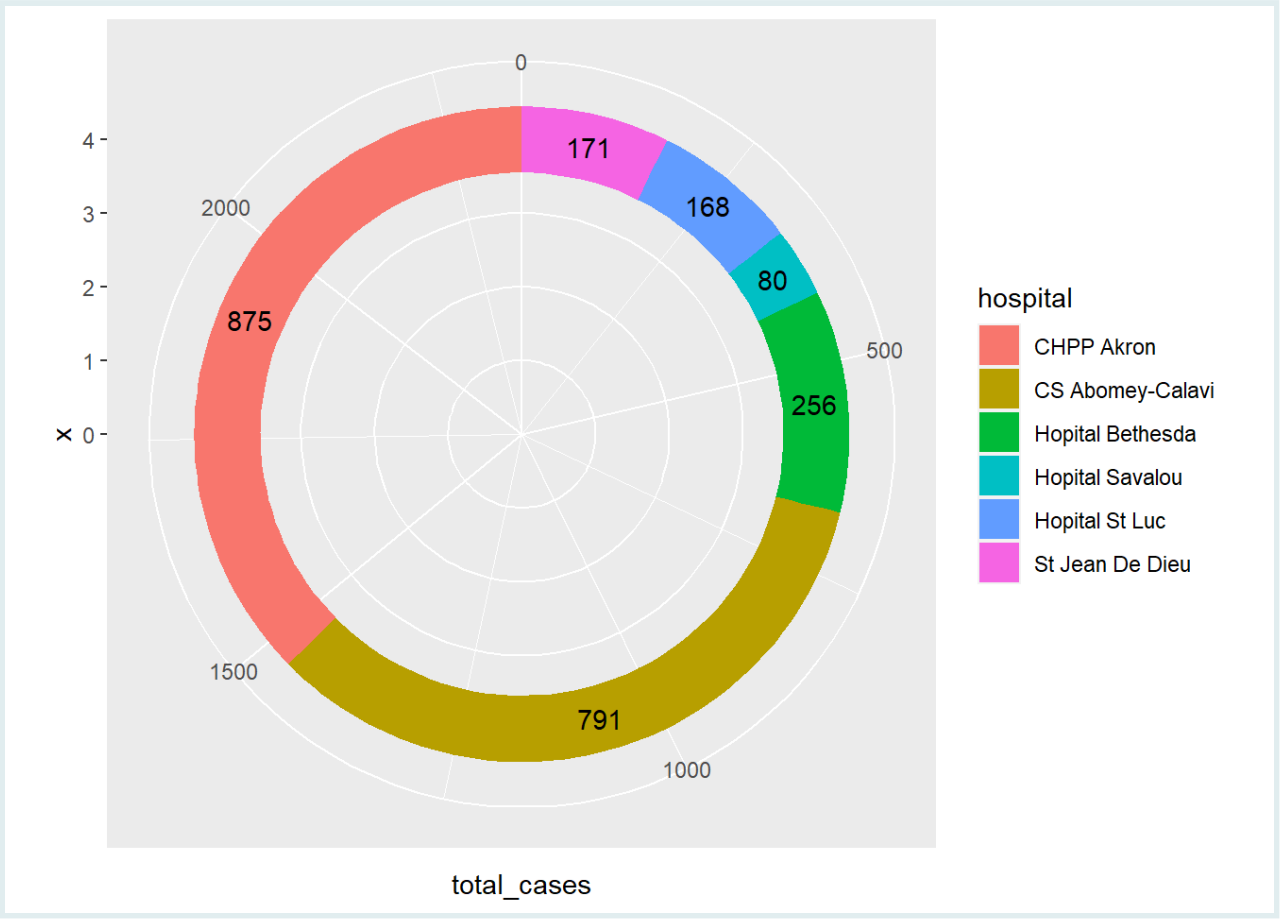

To remove the letter “a” from the legend, we can use `show.legend = FALSE`:

```
outcome_pie +
  geom_label(aes(label = total_cases),
             position = position_stack(vjust = 0.5),
             show.legend = FALSE)
```



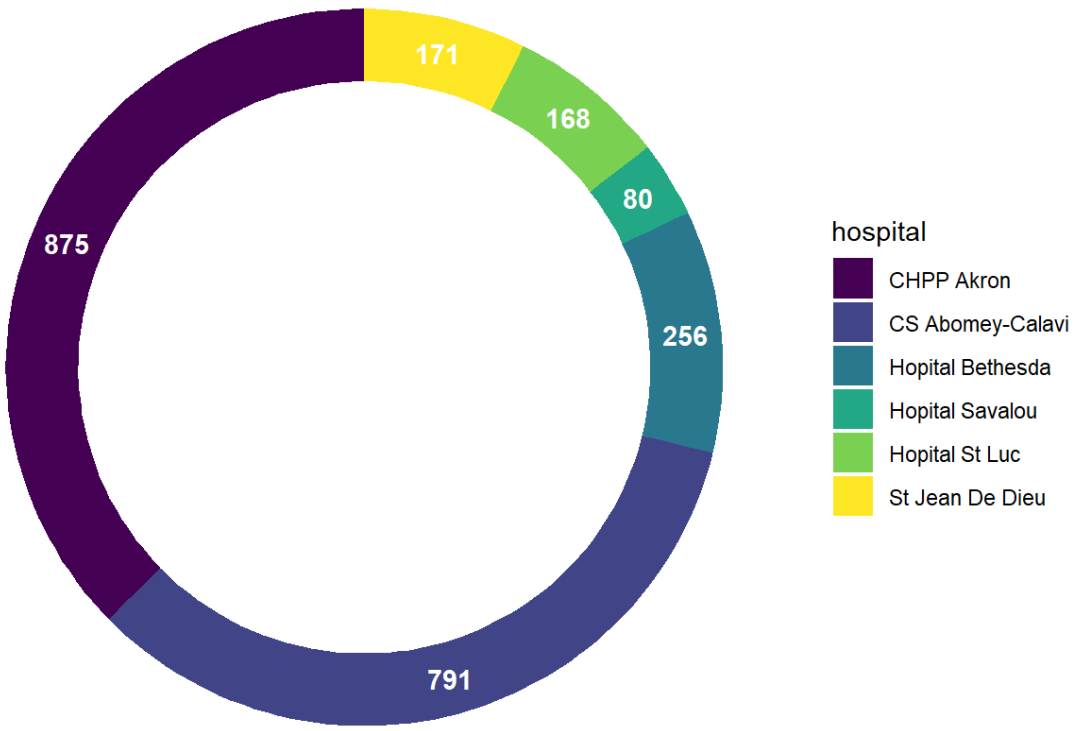

Next, let's move on to our basic donut chart. We'll label it using `geom_text()`:

```
outcome_donut +
  geom_text(aes(label = total_cases),
            position = position_stack(vjust = 0.5))
```




To finish, we can make some additional aesthetic adjustments. Here, we enhance the chart's aesthetics by applying `theme_void()` to remove cluttered background elements, introducing a new color palette with `scale_fill_viridis_d()`, and adjusting the text labels using `geom_text()` with white and bold text for better visibility and contrast.

```
# Additional aesthetic modifications
outcome_donut +
  geom_text(aes(label = total_cases),
            position = position_stack(vjust = 0.5),
            color = "white",
            fontface = "bold") +
  theme_void() +
  scale_fill_viridis_d()
```




Congratulations, it looks great!


### Q: Labeling Pie Charts

Plot total TB cases in all rural vs urban areas in the `aus_tb_notifs` dataset as a pie chart. Use `geom_text()` to place labels correctly, indicating the number of cases in that area.

You can use the code and comments below as a guide:

```
# Pivot then summarize the total cases per area type
aus_tb_notifs %>%
  pivot_longer(cols = c(rural, urban),
               names_to = "area_type",
               values_to = "cases") %>%
  group_by(area_type) %>%
  summarise(total_cases = sum(cases))
```


```
## # A tibble: 2 × 2
##   area_type total_cases
##   <chr>           <dbl>
## 1 rural             394
## 2 urban            4981
```

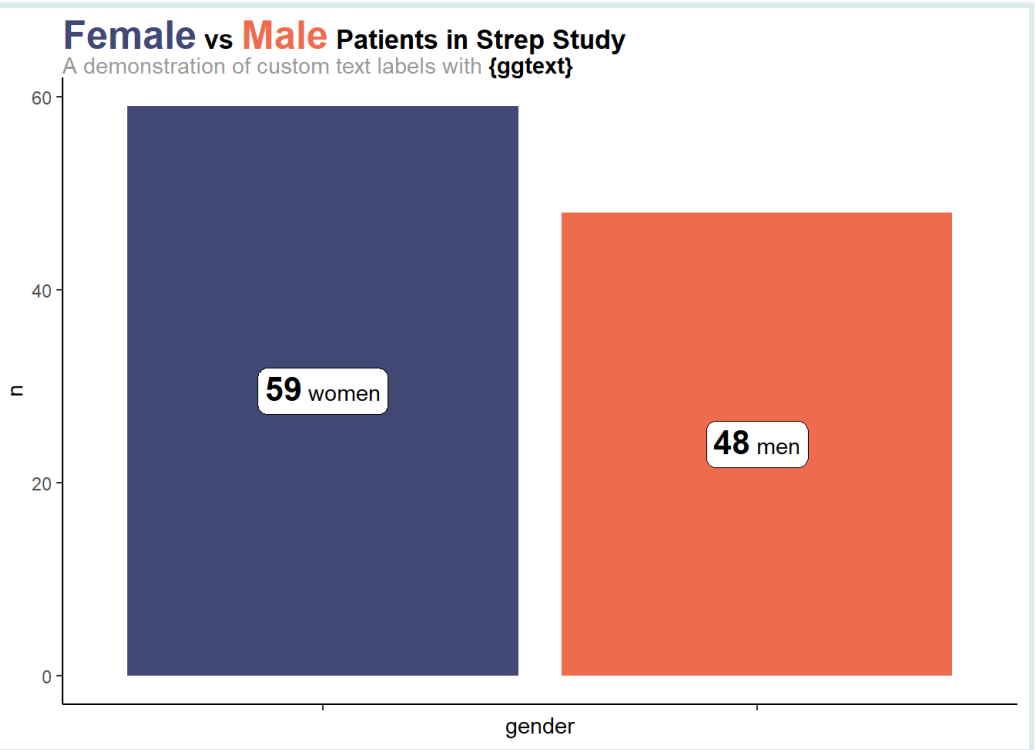
```
# Now, create the pie chart
# For the text labels, use geom_text() and position_stack(vjust
        = 0.5)
```


**Pro-Tip: Enhancing Text Labels with ggtext**

For advanced plotters seeking even more sophisticated control over text formatting in ggplot2, the {ggtext} package may come in handy. It allows the use of CSS to precisely format text elements, including options to embolden, italicize, change color and size, add superscripts/subscripts, and even embed images. Notably, you can apply multiple styles within the same text element, opening up new levels of creativity and customization.

Consider the example below, which uses {ggtext} for the plot title, subtitle and bar labels:


```
pacman::p_load(tidyverse, ggtext, medicaldata)

# Data and Plot
medicaldata::strep_tb %>%
  count(gender) %>%
  mutate(gender_label = paste0("**<span style='font-
        size:16pt'>", n, "</span>**",
                               if_else(gender == "M", " men", "
        women"))) %>%
  ggplot(aes(x = gender, fill = gender, y = n)) +
  geom_col() +
  scale_fill_manual(values = c("M" = "#ee6c4d", "F" =
        "#424874")) +
  labs(
    title = "<b><span style='color:#424874; font-
        size:19pt'>Female</span> vs
    <span style='color:#ee6c4d; font-size:19pt'>Male</span>
    Patients in Strep Study</b>",
    subtitle = "<span style='color:gray60'>A demonstration of
        custom text labels with </span>**{ggtext}**") +
  theme_classic() +
  theme(plot.title = element_textbox_simple(),
        plot.subtitle = element_textbox_simple(),
        legend.position = "none",
        axis.text.x = element_blank()) +
  geom_richtext(aes(label = gender_label, y = n/2),
                label.r = grid::unit(5, "pt"), fill = "white")
```




To learn more about {ggtext}, visit the package website.

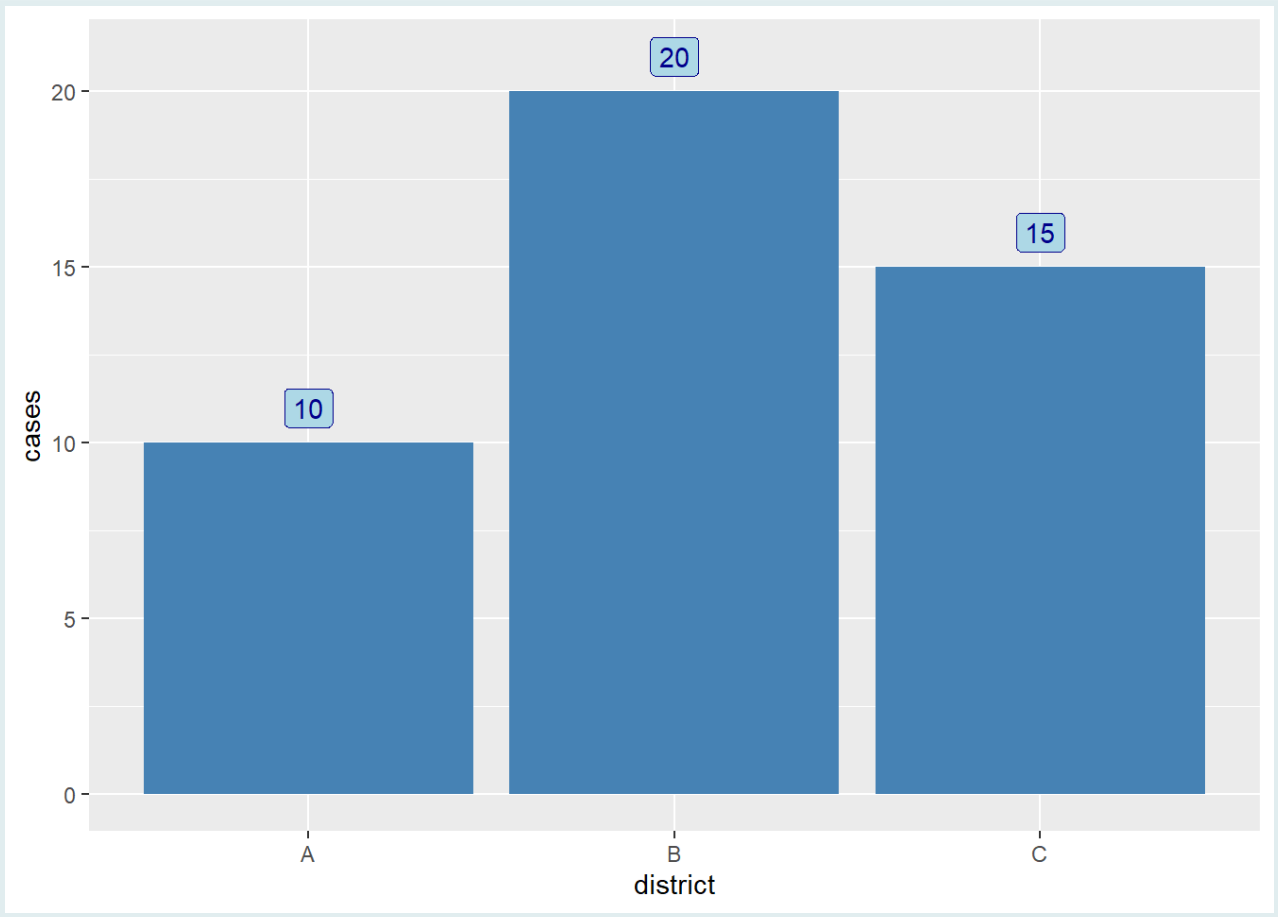
## Wrap Up!

This lesson covered the use of `geom_text()` and `geom_label()` in ggplot2, demonstrating their application in various plot types. We learned how to add and adjust text labels for clarity and effective data presentation, from basic bar plots to more complex formats like stacked and circular plots. These skills are essential for enhancing the readability and informative value of graphical data representations in R.

## Solutions

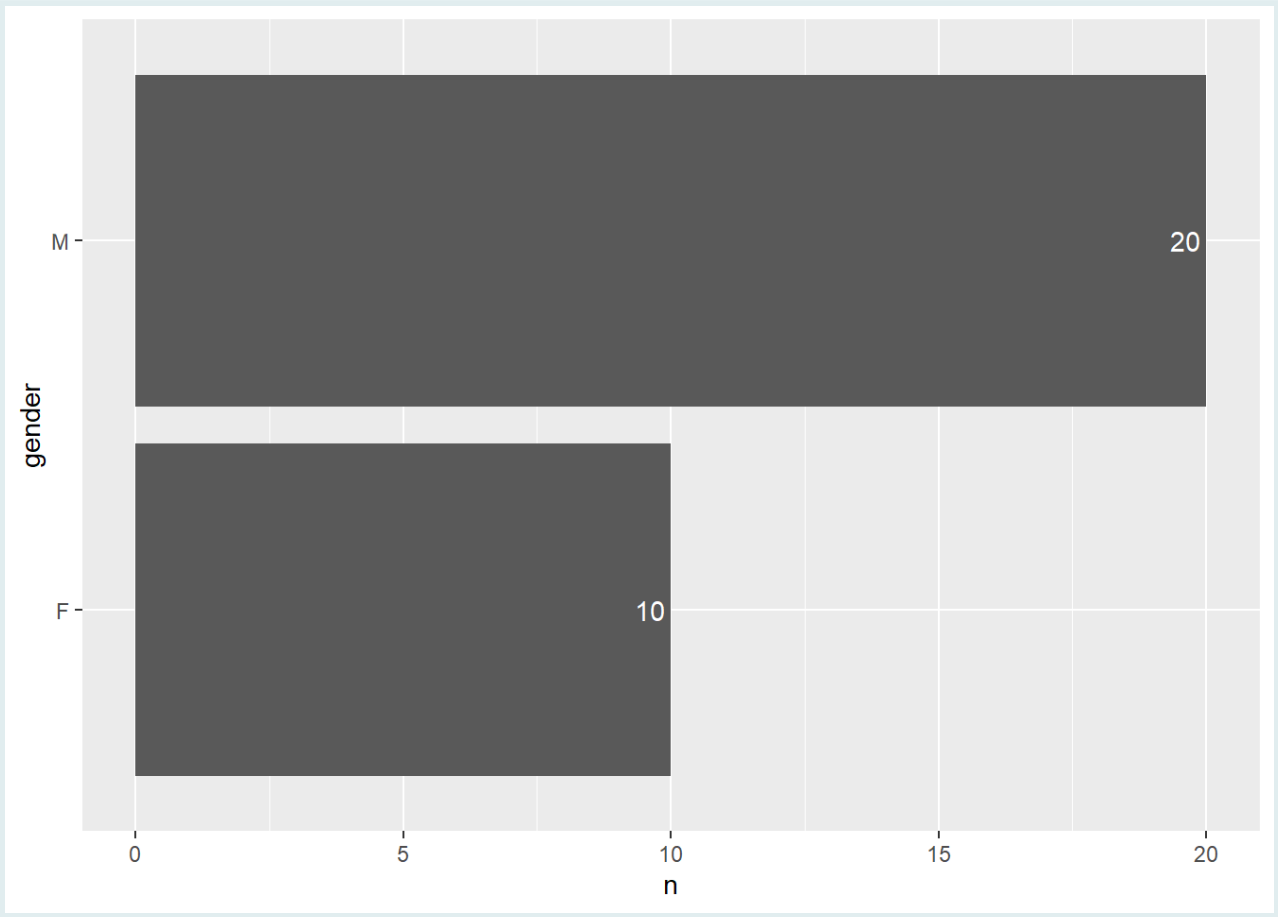
### Q: Simple labeling

```
ggplot(district_cases, aes(x = district, y = cases)) +
  geom_col(fill = "steelblue") +
  geom_label(aes(label = cases),
             nudge_y = 1,  # Adjust to position the labels above the bars
             fill = "lightblue",  # Background color of the labels
             color = "darkblue"  # Text color
             )
```



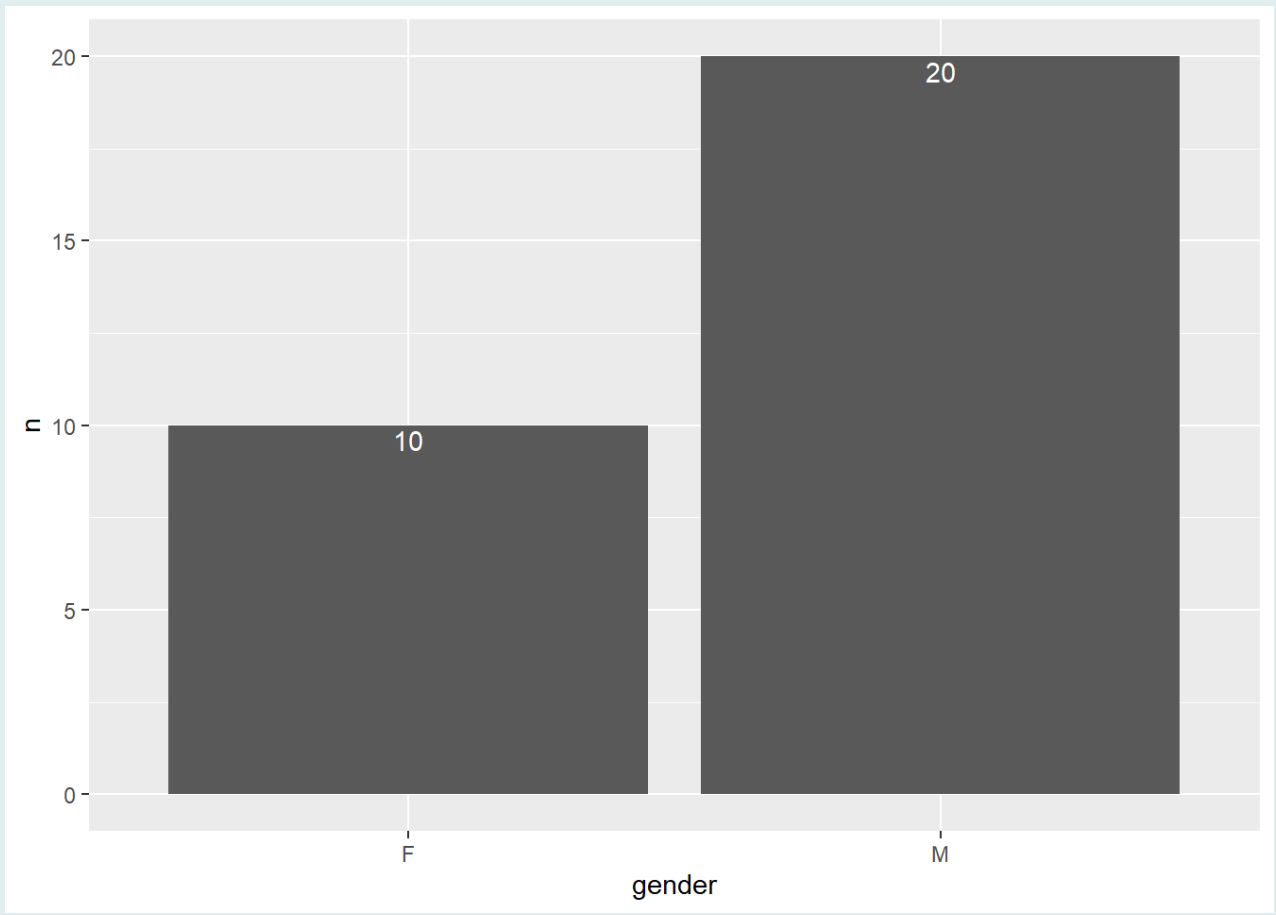

## Q: Horizontal adjusment practice

```
# Adjusting text position inside the bar
ggplot(sample_gender, aes(x = n, y = gender)) +
  geom_col() +
  geom_text(aes(label = n), color = "white", hjust = 1.2)
```



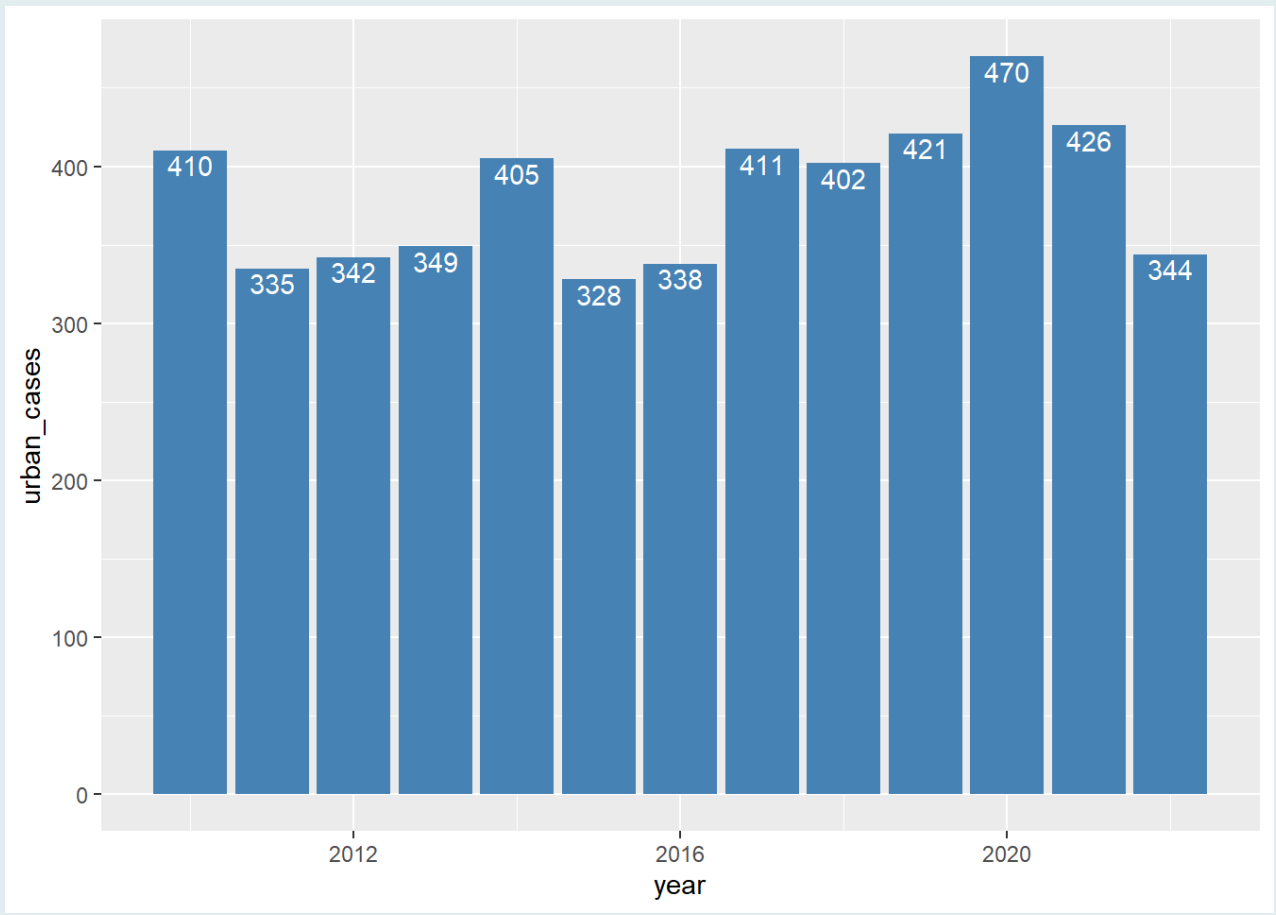

## Q: Vertical adjusment practice

```
# Adjusting text position inside the bar
ggplot(sample_gender,
       aes(x = gender, y = n)) +
  geom_col() +
  geom_text(aes(label = n), color = "white", vjust = 1.2)
```



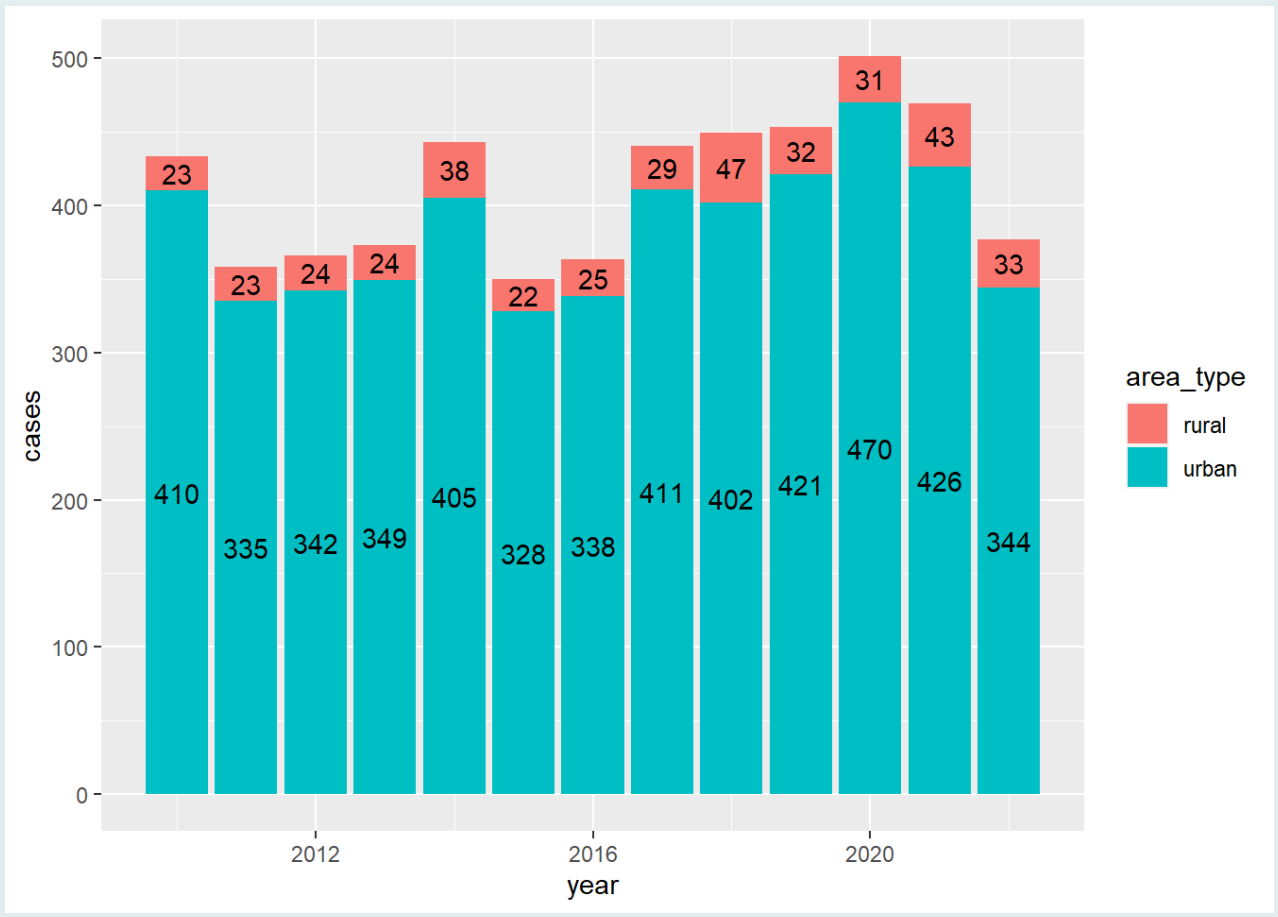

## Q: Summarize then plot

```
# aggregate the data by year and sum up the urban cases
urban_cases <-
  aus_tb_notifs %>%
  group_by(year) %>%
  summarize(urban_cases = sum(urban))

# plot the data
ggplot(urban_cases, aes(x = year, y = urban_cases)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = urban_cases),
            color = "white",
            vjust = 1.2)
```



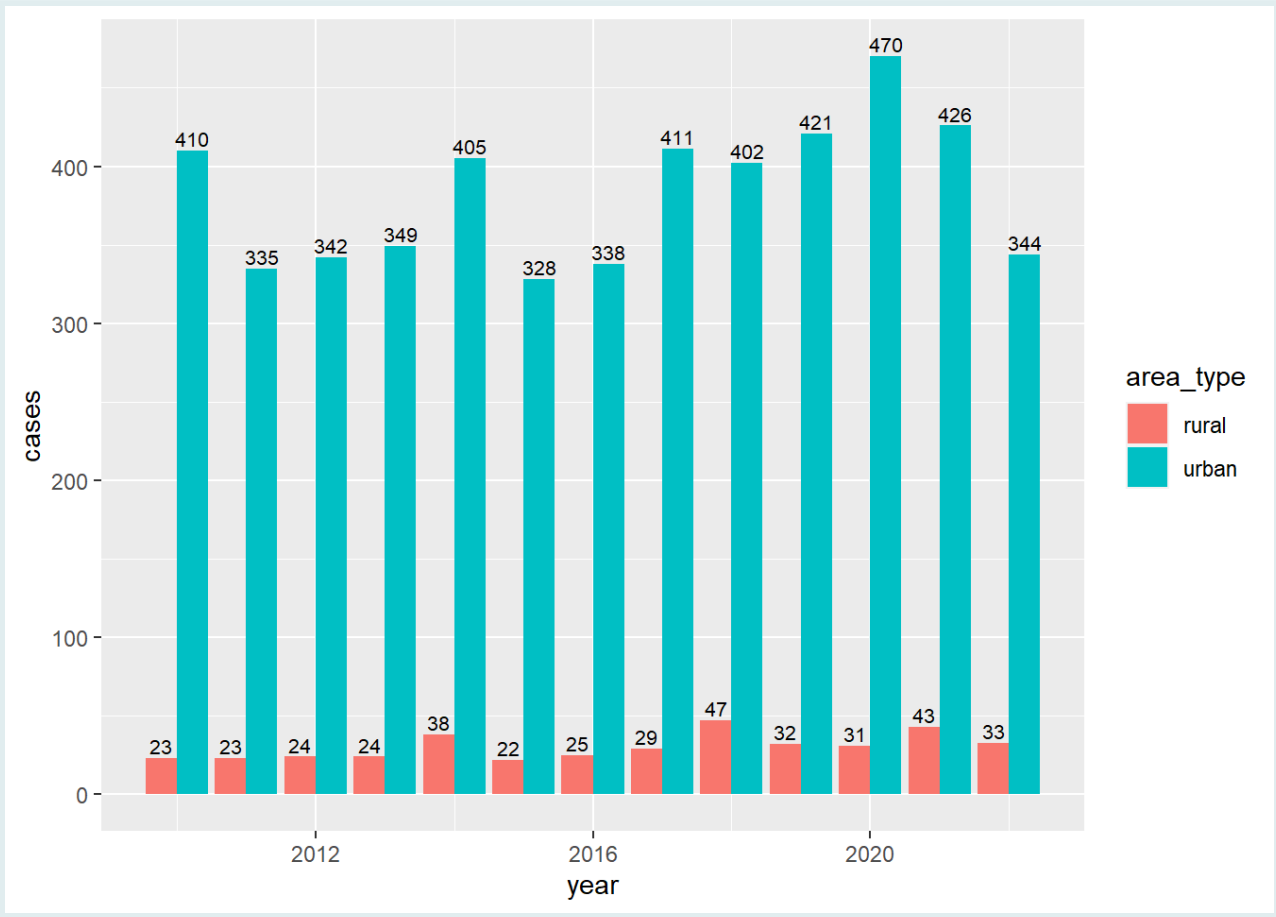

## Q: Practice with labeling stacked plots

```
# Pivot the data
aus_tb_pivoted <- aus_tb_notifs %>%
  pivot_longer(cols = c(rural, urban),
               names_to = "area_type",
               values_to = "cases")

# Summarize the data by year and area type
aus_tb_summarized <- aus_tb_pivoted %>%
  group_by(year, area_type) %>%
  summarise(cases = sum(cases))

# Create the stacked bar plot
ggplot(aus_tb_summarized, aes(x = year, y = cases, fill = area_type)) +
  geom_col() +
  geom_text(aes(label = cases),
            position = position_stack(vjust = 0.5))
```



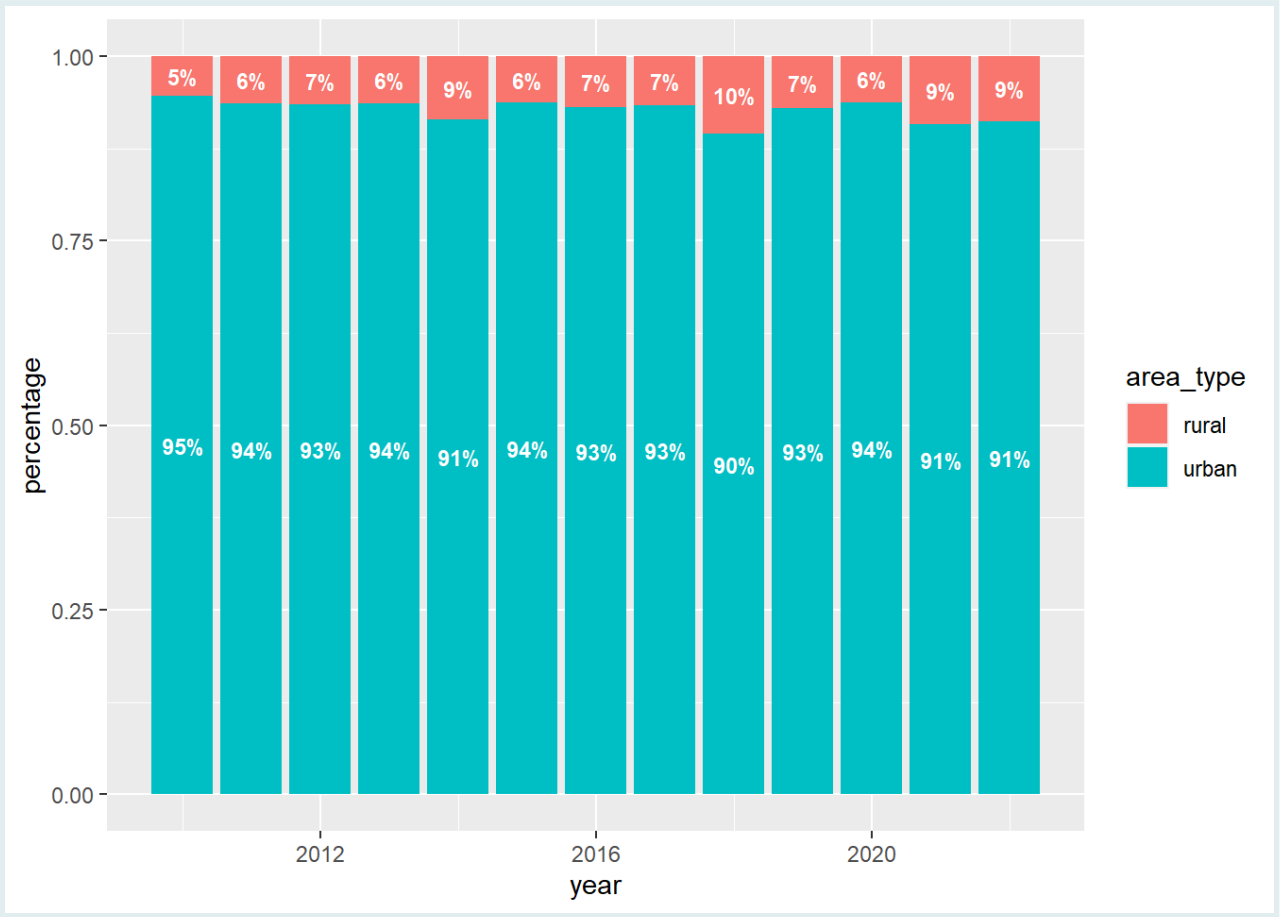

## Q: Practice with labeling dodged bar plots

```
# Pivot the data
aus_tb_pivoted <- aus_tb_notifs %>%
  pivot_longer(cols = c(rural, urban),
               names_to = "area_type",
               values_to = "cases")

# Summarize the data by year and area type
aus_tb_summarized <- aus_tb_pivoted %>%
  group_by(year, area_type) %>%
  summarise(cases = sum(cases))

# Create the dodged bar plot
ggplot(aus_tb_summarized, aes(x = year, y = cases, fill = area_type)) +
  geom_col(position = "dodge") +
  geom_text(aes(label = cases),
            position = position_dodge(width = 0.9),
            vjust = -0.3,
            size = 2.8)
```



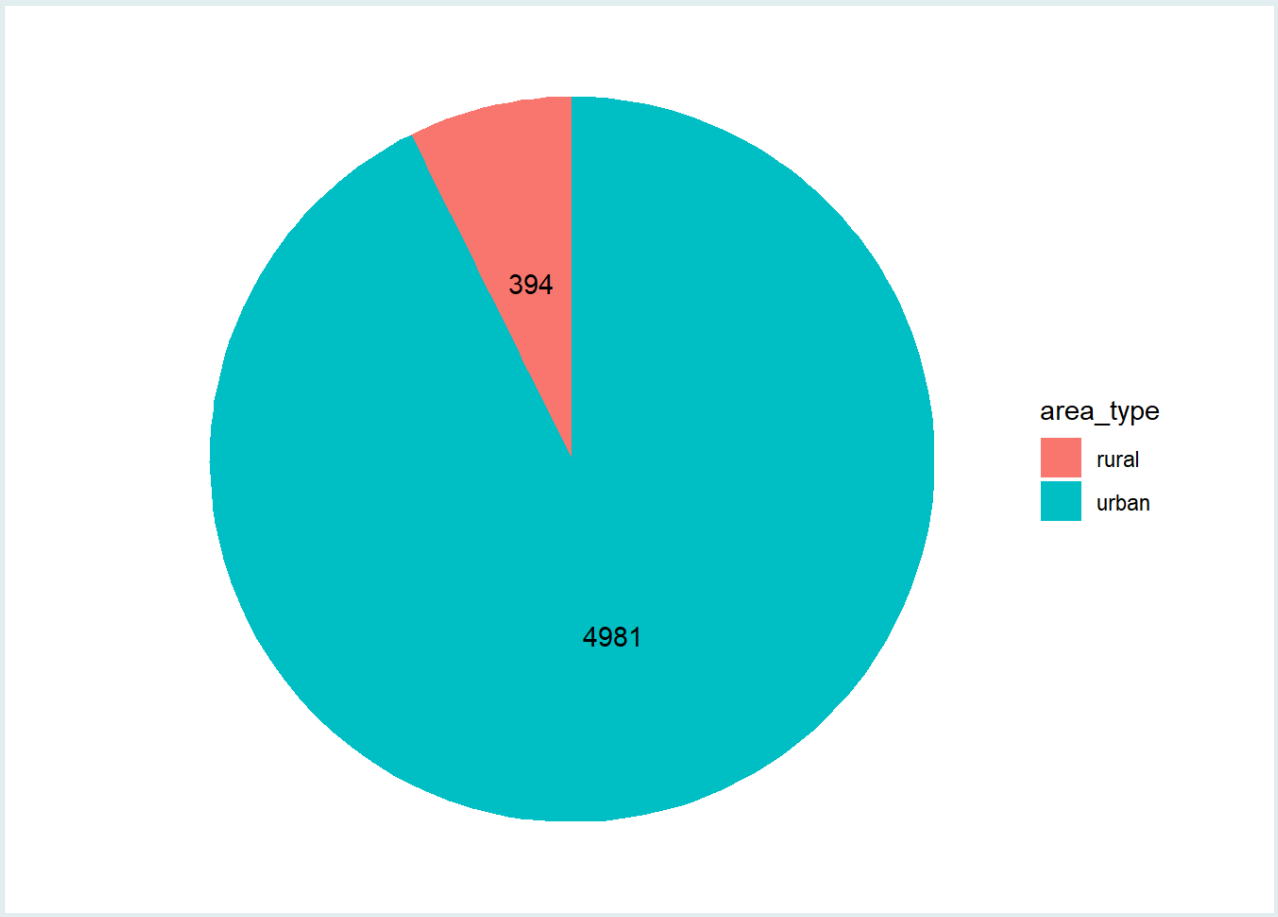

## Q: Creating Percent-Stacked Bar Plots with Labels

```r
# Pivot the data
aus_tb_pivoted <- aus_tb_notifs %>%
  pivot_longer(cols = c(rural, urban),
               names_to = "area_type",
               values_to = "cases")

# Summarize and calculate proportions
aus_tb_percent <- aus_tb_pivoted %>%
  group_by(year, area_type) %>%
  summarise(cases = sum(cases)) %>%
  mutate(percentage = cases / sum(cases))

# Create the percent-stacked bar plot
ggplot(aus_tb_percent, aes(x = year, y = percentage, fill = area_type)) +
  geom_col(position = "fill") +
  geom_text(aes(label = scales::percent(percentage, accuracy = 1)),
            position = position_fill(vjust = 0.5),
            size = 3,
            color = "white",
            fontface = "bold")
```




## Q: Labeling Pie Charts

```
# Summarize the total cases per quarter
aus_tb_rural_urban <- aus_tb_notifs %>%
  pivot_longer(cols = c(rural, urban),
               names_to = "area_type",
               values_to = "cases") %>%
  group_by(area_type) %>%
  summarise(total_cases = sum(cases))


# Create the pie chart
ggplot(aus_tb_rural_urban, aes(x = "", y = total_cases, fill = area_type)) +
  geom_col() +
  geom_text(aes(label = total_cases), position = position_stack(vjust = 0.5))
      +
  coord_polar(theta = "y") +
  theme_void()
```




## Contributors

The following team members contributed to this lesson:

### BENNOUR HSIN

Data Science Education Officer
Data Visualization enthusiast

### 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

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.