DATA VISUALIZATION IN R USING ggplot2

Statistical Computing in R

Lecturer:Ivan Innocent Sekibenga

1. Introduction to Data Visualisation in R

Data visualisation is the graphical representation of information and data. By using visual elements like charts, graphs, and maps. Data visualisation tools provide an accessible way to see and understand trends, outliers, and patterns in data.

1.1 Why Data Visualisation?

- To communicate information clearly and effectively through graphical means.
- To identify patterns, trends, and outliers in data.
- To make data-driven decisions.
- To explore and understand complex datasets.

1.2 Different visualization packages in R

R has many visualisation packages that will help you do almost anything you want with your data from making simple pie charts, to creating more complex visuals like interactive graphs and maps. Some of the most popular packages include:

- `ggplot2` (the most popular package for creating static graphics)
- `Plotly` (do a wide range of visualization functions)
- `Lattice` (for creating trellis graphs)
- `RGL` (focus on specific solutions like 3D visuals)
- `Dygraphs`
- `Leaflet` (for interactive maps)
- `Highcharter`
- `Patchwork`
- `gganimate`
- `ggridges`

2. The ggplot2 package

It's the most popular visualization package in R. A lot of data analysts prefer to use `ggplot2`. You can use `ggplot2` on its own or extend its powers with other packages.

`ggplot2` was originally created by the statistician and developer Hadley Wickham in 2005. Wickham's inspiration for creating `ggplot2` came from the 1999 book, *The Grammar of Graphics, a scholarly study of* *data visualization* by computer scientist Leland Wilkinson. The first two letters of `ggplot2` actually stand for *grammar of graphics*. And in the same way the grammar of a human language gives us rules to build any kind of sentence, the grammar of graphics gives us rules to build any kind of visual.

2.1 The Grammar of Graphics

The grammar of graphics is a framework for data visualization that breaks down graphs into semantic components such as layers, scales, and coordinates. It provides a systematic way to describe and create visualizations by defining the relationships between data and graphical elements. The key components of the grammar of graphics include:

- **Data**: The dataset being visualized.
- **Aesthetics**: The visual properties of the data points, such as position, color, size, and shape.
- **Geometries**: The geometric objects used to represent data points, such as points, lines, and bars.
- **Statistics**: The statistical transformations applied to the data, such as summarization or smoothing.
- **Facets**: The division of data into subsets for creating multiple plots based on a categorical variable.
- **Coordinates**: The coordinate system used to map data points to the plotting area.
- **Themes**: The overall visual style and appearance of the plot.

2.2 Loading the ggplot2 package and the dataset

You load the tidyverse package which includes `ggplot2` using the code below:

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.1     v stringr   1.5.2
## v ggplot2   4.0.0     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
data("penguins") # load the penguins dataset.

# It is part of the palmerpenguins package which is included in the tidyverse package.

# it contains data about three species of penguins (Adelie, Chinstrap and Gentoo)

View(penguins)   # view the entire penguins dataset as a dataframe
```

2.3 Merits of the ggplot2 package

- Create different types of plots including scatter plots,pie charts, bar charts, line diagrams and many more.
- Customize the look and feel of plots (change the colors, layout and dimensions of your plots and add text elements like titles, captions and labels).
- Create high-quality visuals.

- Combine data manipulation and visualization using the pipe operator.
- You can add or remove layers of detail to your plot without changing its basic structure or the underlying data.

2.4 The core concepts of the ggplot2 package

The core concepts of the ggplot2 package include:

- **Aesthetics:** Aesthetic is a visual property of an object in your plot. For example, in a scatter plot aesthetics include things like the size, shape or color of your data points. Think of an aesthetic as a connection or mapping between a visual feature in your plot and a variable in your data.
- **Geoms:** A geom refers to the geometric object used to represent your data. For example, you can use points to create a scatter plot, bars to create a bar chart, or lines to create a line diagram.
- **Facets:** Facets let you display smaller groups or subsets of your data. With facets, you can create separate plots for all the variables in your dataset.
- **Labels and Annotations:** Label and annotate functions let you customize your plot. You can add text like titles, subtitles and captions to communicate the purpose of your plot or highlight important data.

2.5 The ggplot2 cheatsheet

The ggplot2 cheatsheet is a quick reference guide that provides an overview of the key functions available in `ggplot2` . It can be found at: https://ggplot2.tidyverse.org/

3. Getting started with plotting using ggplot2

The code below uses functions from ggplot2 to plot the relationship between `body mass` and `flipper length` using the `penguins` in built R dataset. The end product is a **scatter plot**.

In R, a function's name is followed by a set of parentheses. Lots of functions require special information to do their jobs. You write this information called the function's argument inside the parentheses.

The three functions in the code are the `ggplot` function, the `geom_point` function, and the `aes` function.

STEP 1: Every `ggplot2` plot starts with the `ggplot` function. The `data` argument of the ggplot function tells R what data to use for your plot. So the first thing to do is choose a data frame to work with. You can set up the code like this. Inside the parentheses of the function write the word `data`, then an equal sign, then `penguins`. This code initializes or starts the plot. This is just the first step in creating a plot.

STEP 2: The next thing you might notice about this code is the plus sign at the end of the first line. You use the plus sign to add layers to your plot. In ggplot2, plots are built through combinations of layers. First, we start with our data.

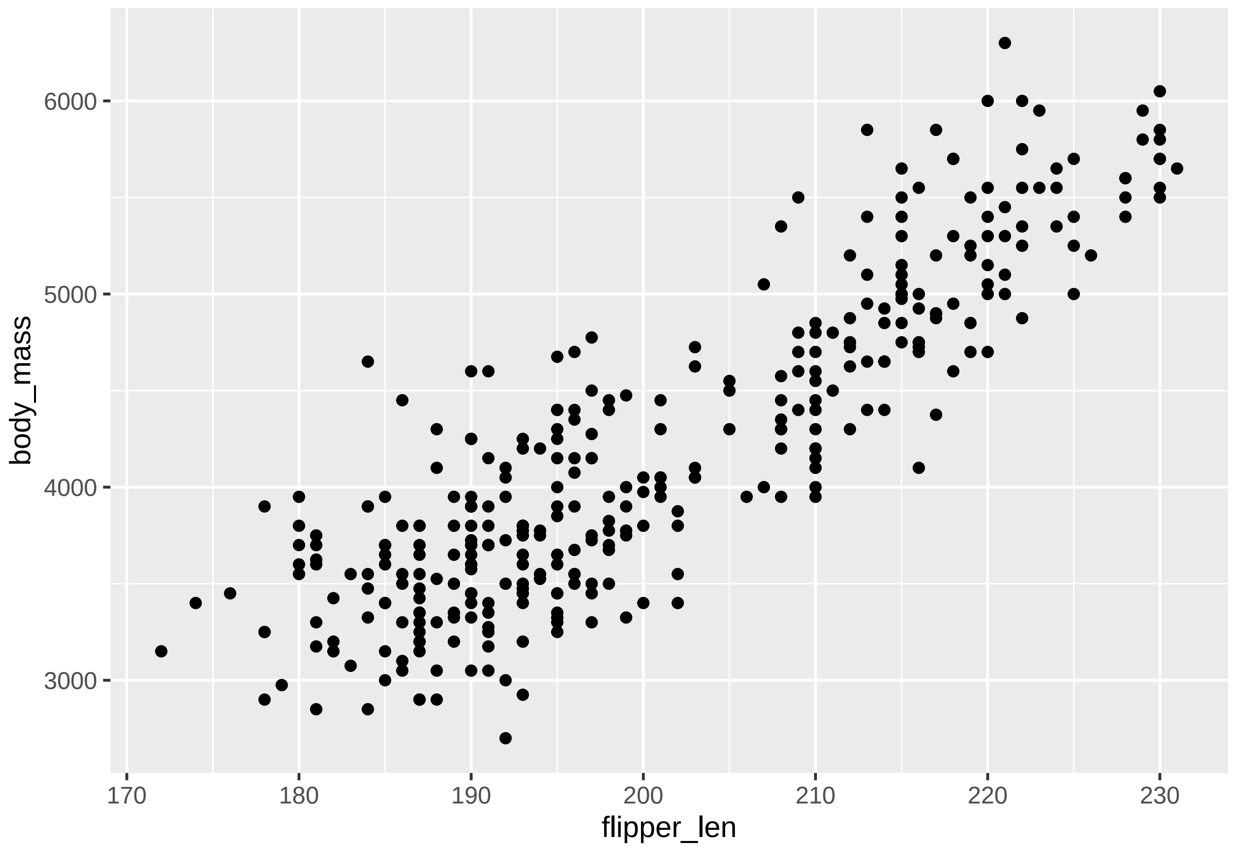
STEP 3: Then we add a layer to our plot by choosing a geom to represent our data. The function `geom_point` tells R to use points to represent our data. Keep in mind that the plus sign must be placed at the end of each line to add a layer. Adding a geom function is the second step in creating a plot. As a reminder, ***a geom is a geometric object used to represent your data. Geoms include points, bars, lines, and more***. In our code, the function geom_point tells R to use points and create a scatter plot.

STEP 4: Next, we need to choose specific variables from our dataset and tell R how we want these variables to look in our plot. In ggplot2, the way a variable looks is called its aesthetic. An ***aesthetic*** is a visual property of an object in your plot, like its position, color, shape, or size. The **mapping equals aes** part of the code tells R what aesthetics to use for the plot. You use the aes function to define the mapping between your data and your plot. Mapping means matching up a specific variable in your dataset with a specific aesthetic. For example, you can map a variable to the x- axis of your plot, or you can map a variable to the y-axis of your plot.

STEP 5: In a scatter plot, you can also map a variable to the color, size, and shape of your data points. Mapping aesthetics to variables is the third step in creating a plot. In our code, we map the variable flipper length to the x-axis and the variable body mass to the y-axis. Inside the parentheses of the aes function, we write the name of the aesthetic then the equal sign, then the name of the variable. Run the code and then the scatter plot appears showing the relationship between flipper length and body mass of penguins.

```
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len , y = body_mass ))
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



NB: To learn more about any R function, just run the code question mark function_name. For example, if you want to learn more about the geom_point function, type in question mark geom_point i.e

```
? geom_point
```

As a new learner, you might not understand all the concepts in the help page. At the bottom of the page,you can find specific examples of code that may show you how to solve your problem.

3.1 Summary of steps to create a plot using ggplot2

1. Start with the ggplot function and choose a data frame to work with.

2. Add a geom function to choose a geometric object to represent your data.
3. Use the aes function to map variables in your data to aesthetics in your plot.
4. Run the code to create your plot.

3.2 Aesthetics in ggplot2

Aesthetics are visual properties of the objects in your plot. In ggplot2, you can map variables in your data to different aesthetics to change how your data looks in the plot. Here are some common aesthetics you can use in ggplot2:

- **x and y**: The position of your data points on the x-axis and y-axis.
- **color**: The color of your data points. This allows you to change the color of all of the points on your plot, or the color of each data group
- **size**: The size of your data points.This allows you to change the size of the points on your plot by data group
- **shape**: The shape of your data points (e.g., circle, square, triangle). This allows you to change the shape of the points on your plot by data group
- **fill**: The fill color of shapes (e.g., bars in a bar chart).
- **alpha**: The transparency of your data points (0 = fully transparent, 1 = fully opaque).
- **linetype**: The type of line used in line plots (e.g., solid, dashed, dotted).
- **stroke**: The thickness of the outline of shapes.
- **group**: Used to group data points for certain geoms (e.g., lines in a line plot).
- **label**: Text labels for data points.
- **facet**: Used to create multiple plots based on a categorical variable.

You can combine multiple aesthetics in a single plot to create more complex visualizations. For example, you can map one variable to the x-axis, another variable to the y-axis, and a third variable to the color of the data points.

4. Enhancing visualizations in R using ggplot2

In a scatter plot, we can make a point small, triangular or blue or a combination of these. Let's go back to our `penguins` dataset and review the code for our plot that shows the relationship between `body mass` and `flipper length`.

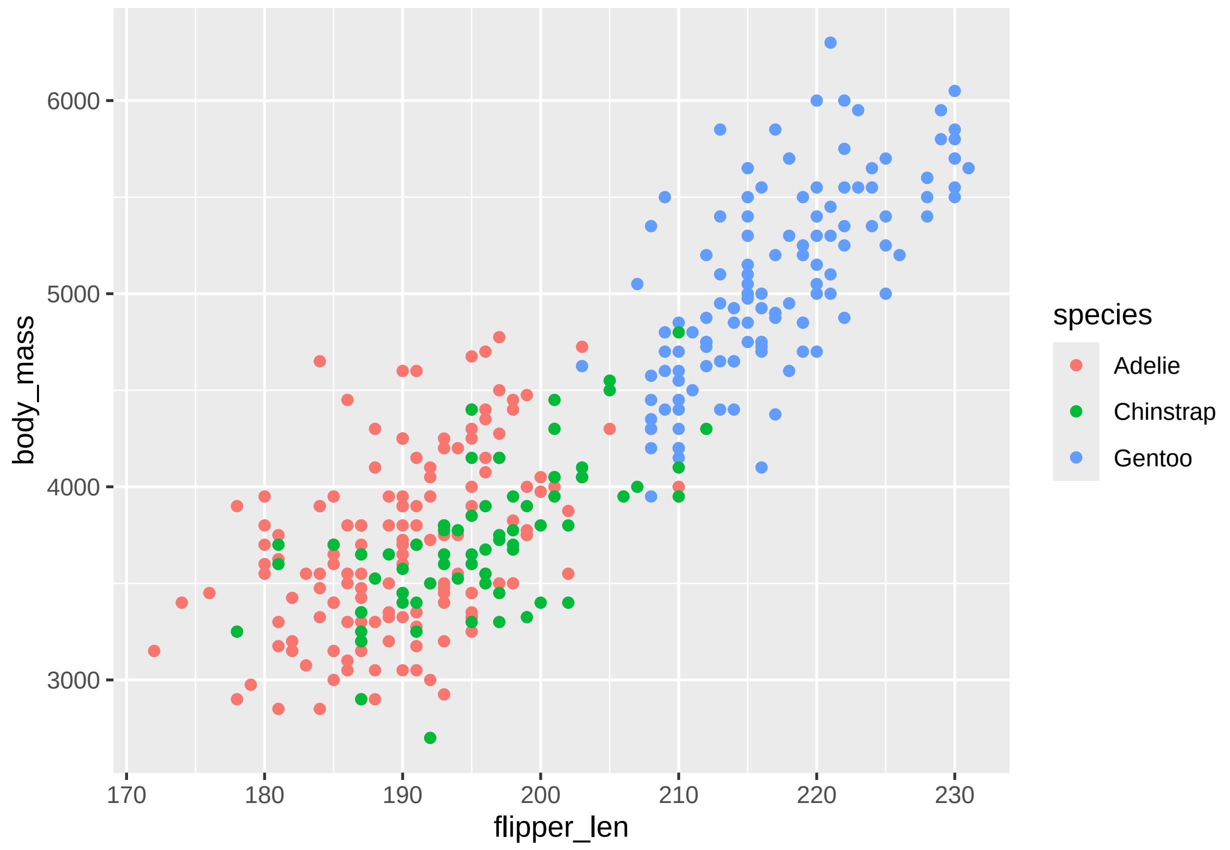
You can also map data to other aesthetics, like color, size and shape. Right now, the plot of penguins data is in black and white. It clearly shows the positive relationship between the two variables. As the values on the x-axis increase, the values on the y-axis increase. But it's also got some limitations. For example, we can't tell which data points refer to each of the three penguin species.

4.1 Color aesthetic

To solve this problem, we can map a new variable to a new aesthetic. Let's add a third variable to our scatter plot by mapping it to a new aesthetic. We'll map the variable `species` to the aesthetic `color` by adding some code inside the parentheses of the `aes` function. We'll add a comma after the `body mass` variable and type color equals sign species. Our code tells R to assign a different color to each `species` of `penguin`. Let's check it out.

```r
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass, color = species))
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



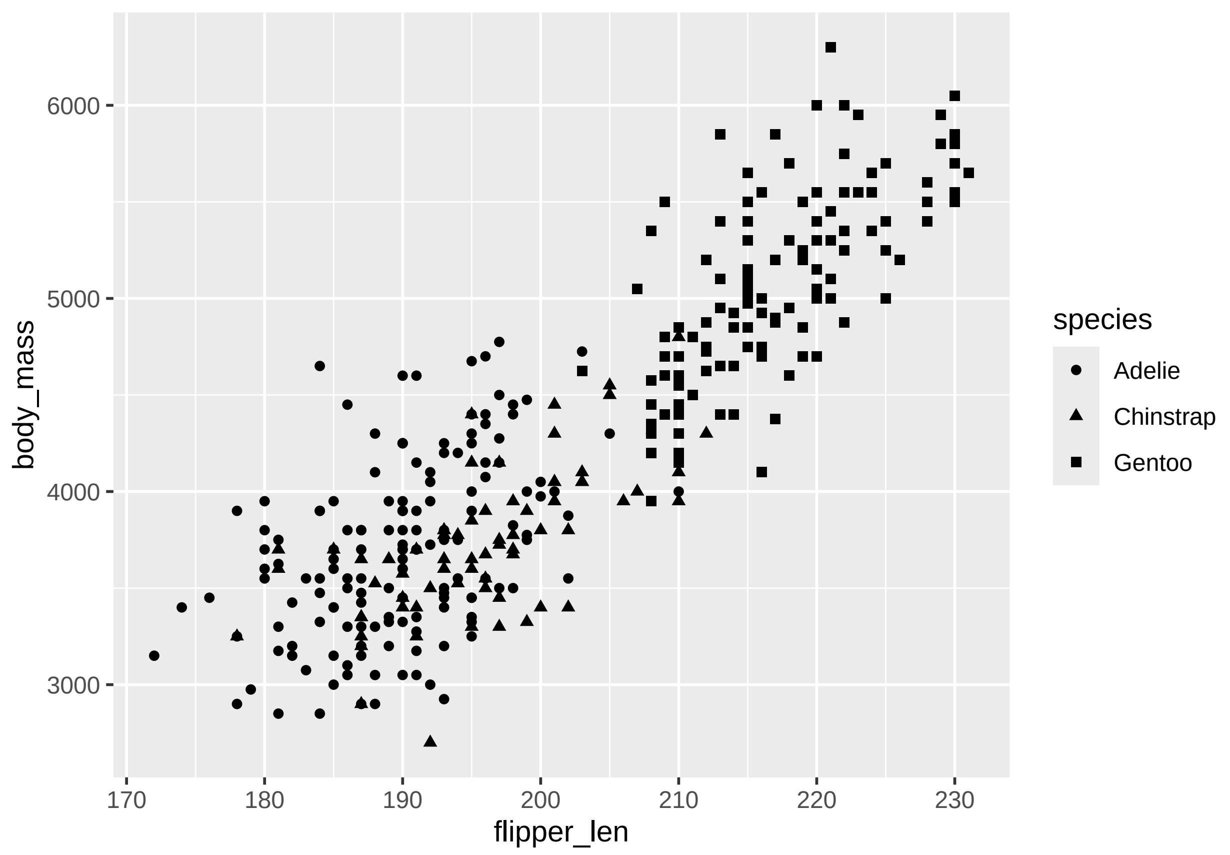
The *Gentoo* are the largest of the three penguin species. The legend just to the right of the plot shows us that the blue points refer to the *Gentoo* penguins. Not only does R automatically apply different colors to each data point, it also gives a legend to show us the color-coding.

4.2 Shape aesthetic

We can also use `shape` to highlight the different penguin species. Let's map the variable species to the aesthetic `shape`. To do this, we can change the code from `color` equal `species` to `shape` equal `species`. Instead of colored points, R assigns different shapes to each species.

```r
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass, shape = species))
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



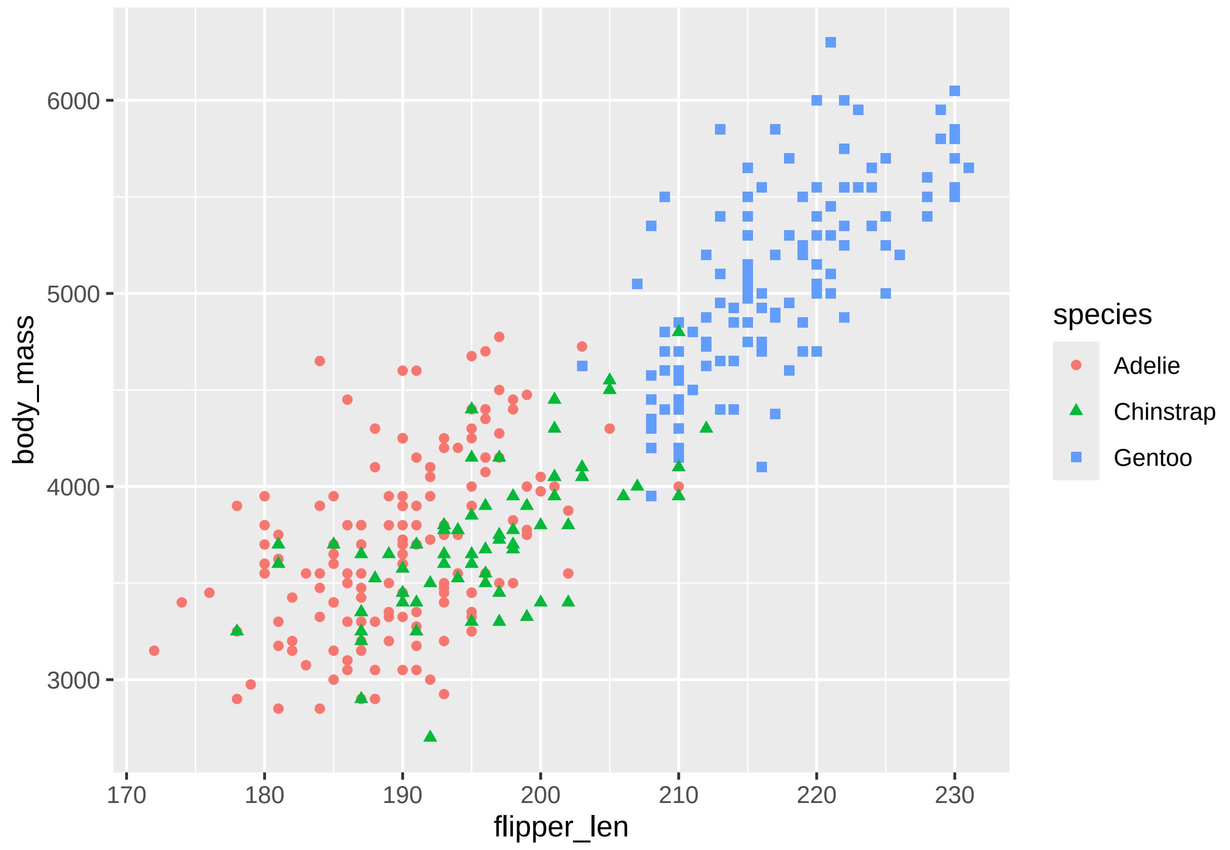
Now the legend shows us a circle for the Adelie species, a triangle for the Chinstraps and a square for the Gentoos. You might notice that our plot's in black and white again because we removed the code for color. Let's put some color back into our plot.

4.3 Combining color and shape aesthetics

If we want we can map more than one aesthetic to the same variable. Let's map both `color` and `shape` to `species`. We'll add the code color equals species while keeping the code shape equal species.

```
ggplot(data = penguins) +
geom_point(mapping = aes(x = flipper_len, y = body_mass,
           color = species, shape = species))
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



Now our plot shows a different color and a different shape for each species.

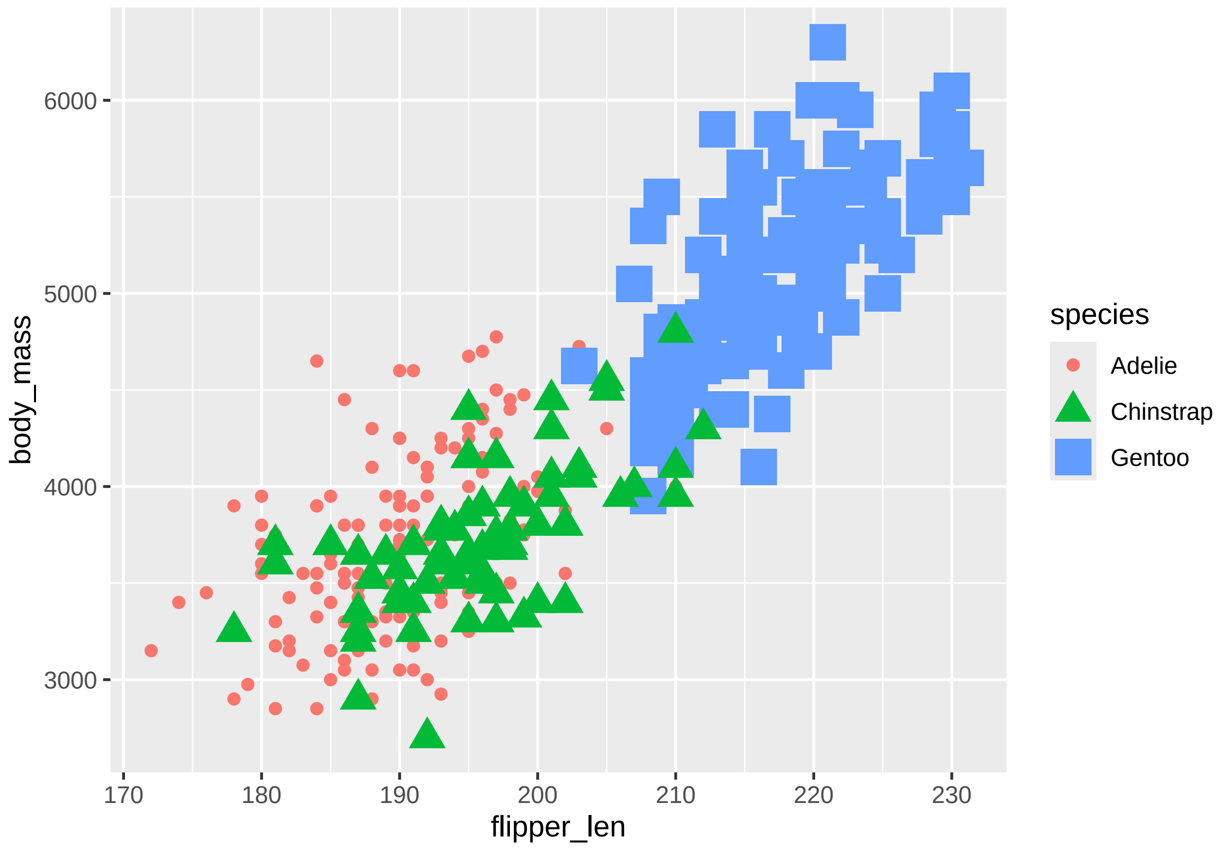
4.4 Combining color, shape and size aesthetics

Let's add size as well and map three aesthetics to species. If we add size equal species, each colored shape will also be a different size.

```
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass,
             color = species, shape = species, size = species))
```

```
## Warning: Using size for a discrete variable is not advised.
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



Using more than one aesthetic can also be a way to make your visuals more accessible because it gives your viewers more than one way to make sense of your data.

4.5 Alpha aesthetic

Another aesthetic you can use is `alpha`. The alpha aesthetic controls the transparency of your data points. You can map a variable to alpha to make your points more or less transparent. This is especially useful when you have a lot of data points that overlap.

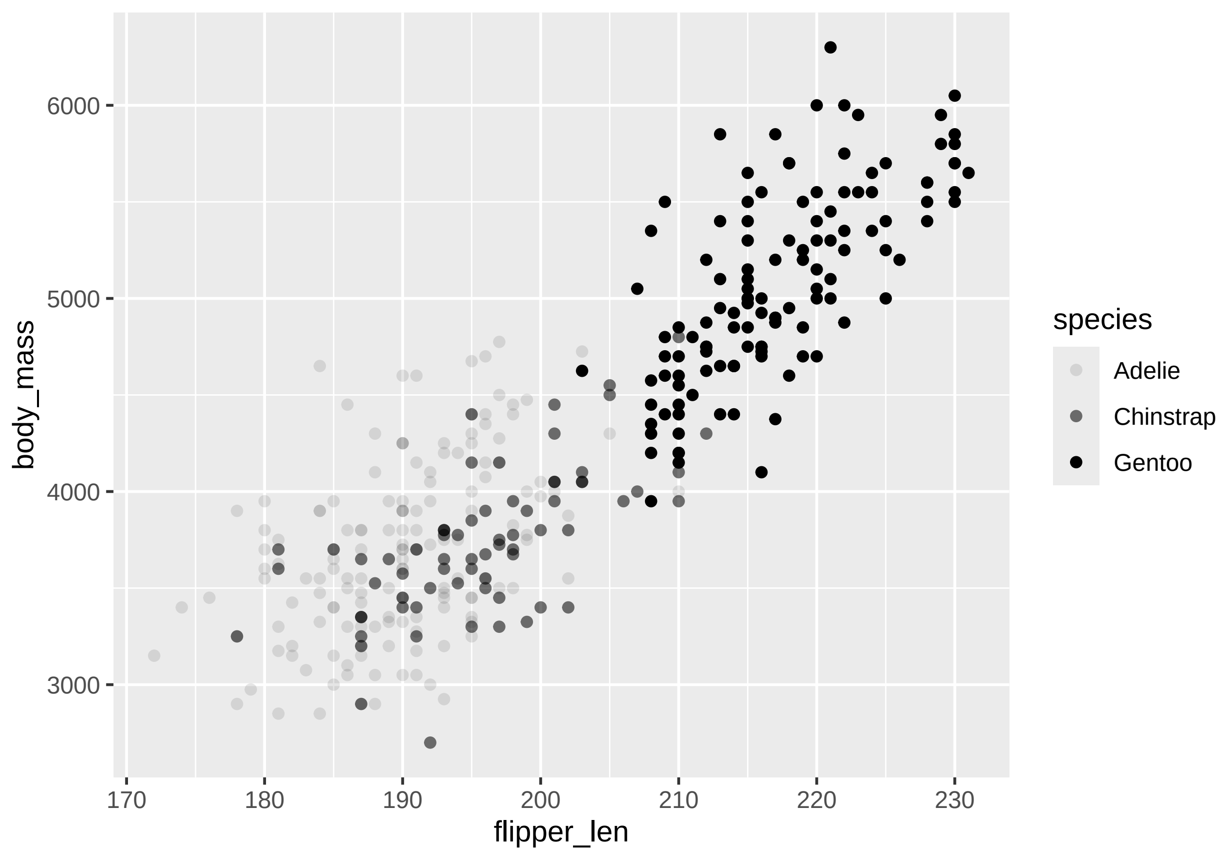
We can also map species to the alpha aesthetic, which controls the transparency of the points. Our first plot showed the relationship between body mass and flipper length in black and white. Then we mapped the variable species to the aesthetic color to show the difference between each of the three penguin species.

If we want to keep our graph in black and white, we can map the alpha aesthetic to species. This will make some points more transparent or see-through than others. This gives us another way to represent each penguin species.

```r
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass, alpha = species))
```

```
## Warning: Using alpha for a discrete variable is not advised.
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



Now the points for the Adelie species are the most transparent, the points for the Chinstrap species are less transparent, and the points for the Gentoo species are the least transparent. Alpha is a good option when you've got a dense plot with lots of data points.

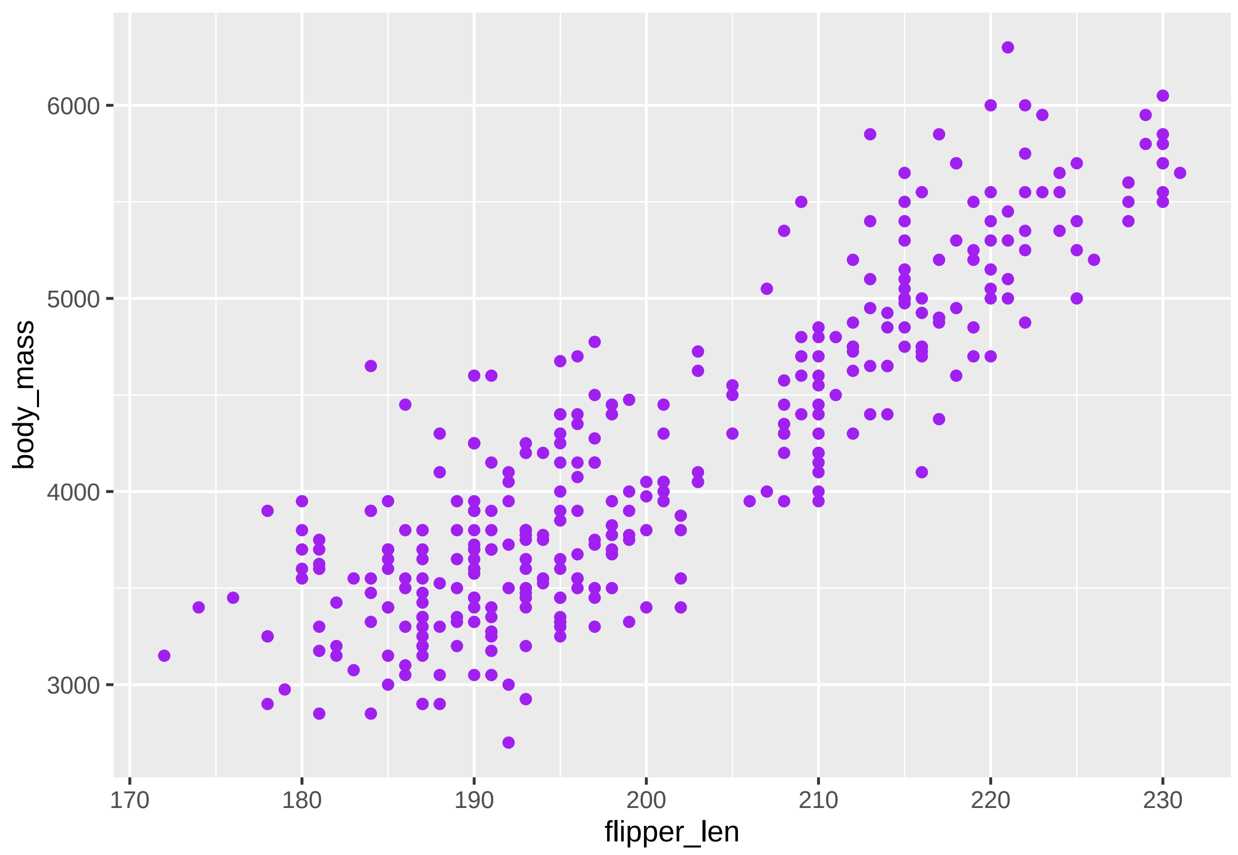
4.6 Working outside the aes function

You can also set the aesthetic apart from a specific variable. Let's say we want to change the color of all the points to purple. Here we don't want to map color to a specific variable like species. We just want every point in our scatter plot to be purple.

So we need to set our new piece of code outside of the aes function and use quotation marks for our color value. This is because all the code inside of the aes function tells R how to map aesthetics to variables. For example, mapping the aesthetic color to the variable species. If we want to change the appearance of our overall plot without regard to specific variables, we write code outside of the aes function. Let's write the code and run it.

```r
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass), color = "purple")
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



5. Create Different Graphs/Plots Using Different Geom function

There are lots of different geoms available. You can choose a specific geom based on how you want to represent your data and your goals for communicating it. This lets you tell the story of your data in different ways and communicate effectively to different audiences. In ggplot2, a geom is the geometrical object used to represent your data.

Geoms include points, bars, lines, and more. The geom_point function uses points to create scatter plots. The geom_bar function uses bars to create bar charts and so on. To change the geom in our plot, we need to change the geom function in our code. The list of geoms is long. Here are some of the most common ones:

- **geom_point()**: Creates scatter plots using points to represent data.
- **geom_line()**: Creates line plots using lines to connect data points.
- **geom_bar()**: Creates bar charts using bars to represent data. For pie charts, you can use the `geom_bar` function with the `coord_polar` function to create a pie chart.
- **geom_histogram()**: Creates histograms to show the distribution of a single variable.
- **geom_boxplot()**: Creates box plots to show the distribution of a variable and identify outliers.
- **geom_smooth()**: Adds a smoothed line to a scatter plot to show trends.
- **geom_area()**: Creates area plots to show the cumulative values of a variable over time.
- **geom_violin()**: Creates violin plots to show the distribution of a variable.
- **geom_density()**: Creates density plots to show the distribution of a variable.
- **geom_text()**: Adds text labels to a plot.

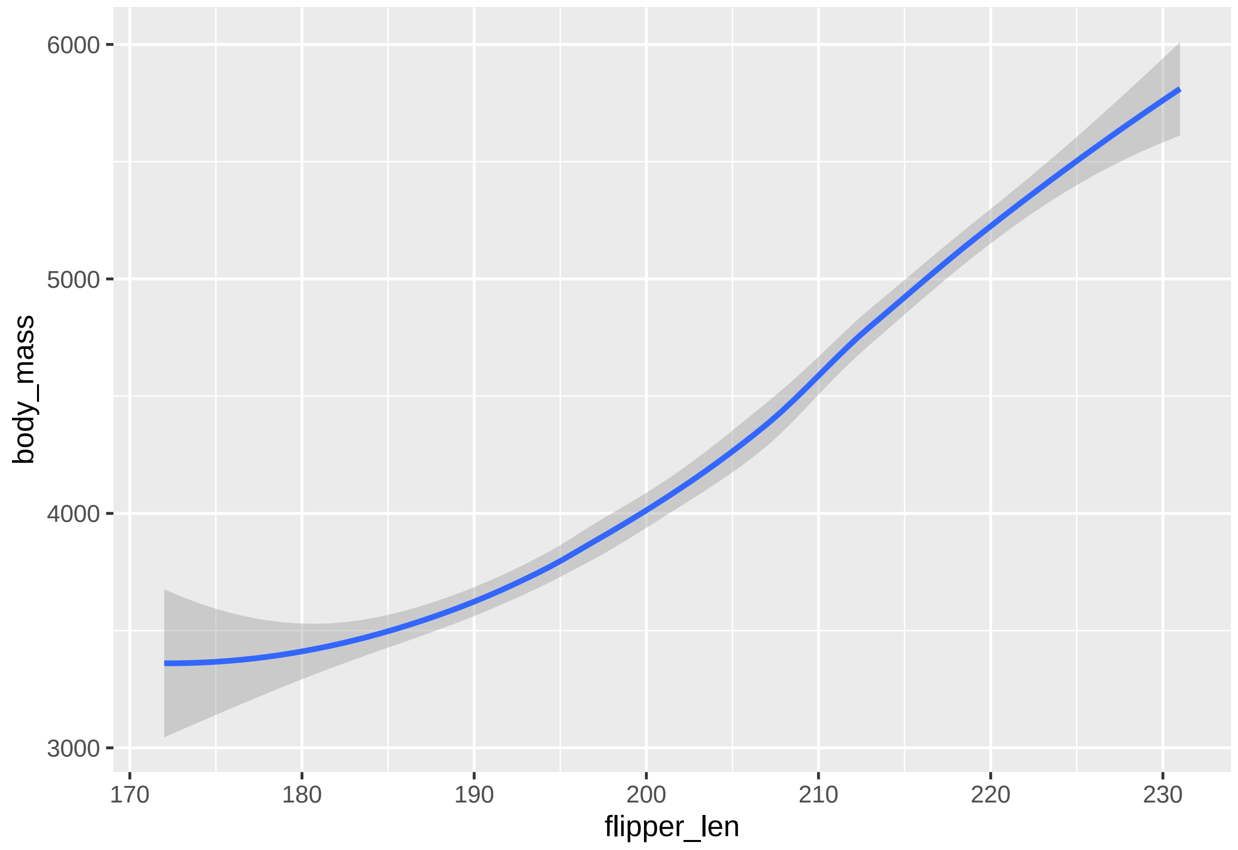
- **geom_tile()**: Creates heatmaps to show the intensity of values in a matrix.
- **geom_ribbon()**: Creates ribbon plots to show the range of values for a variable.
- **geom_path()**: Creates path plots to connect data points in a specific order.
- **geom_jitter()**: Adds random noise to data points in a scatter plot to reduce overplotting.
- **geom_step()**: Creates step plots to show changes in a variable over time.
- **geom_segment()**: Creates segment plots to show relationships between two variables.
- **geom_curve()**: Creates curved lines to connect data points.
- **geom_polygon()**: Creates polygon plots to show areas defined by multiple points.
- **geom_map()**: Creates maps to visualize spatial data.
- **geom_sf()**: Creates plots for spatial data using simple features.
- **geom_hex()**: Creates hexbin plots to show the density of data points in a scatter plot.
- **geom_contour()**: Creates contour plots to show the density of data points in a scatter plot.

For creating a **Smooth line chart** we use the following code:

```r
ggplot(data = penguins) + geom_smooth(mapping = aes(x = flipper_len, y = body_mass))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```



We still have the same data, but now the data's got a different visual appearance. Instead of points, there's a smooth line that fits the data. The `geom_smooth` function's useful for showing general trends in our data. The line clearly shows the positive relationship between body mass and flipper length. The larger the penguin, the longer the flipper.

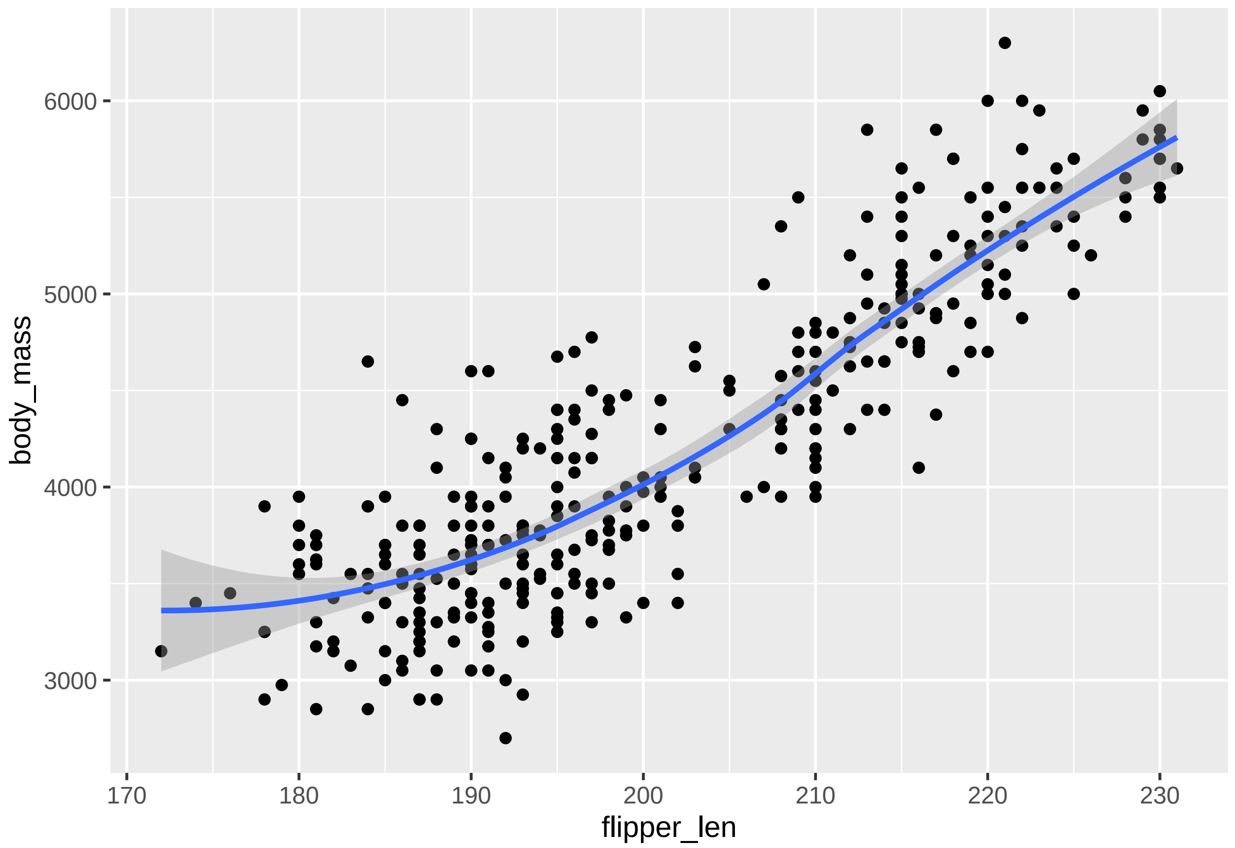
We can even use two geoms in the same plot. Let's say we want to show the relationship between the trend line and the data points more clearly. We can combine the code for `geom_point` and the code for `geom_smooth` by adding a plus symbol after geom underscore smooth. Let's write the code and run it.

```
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass)) +
  geom_smooth(mapping = aes(x = flipper_len, y = body_mass))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

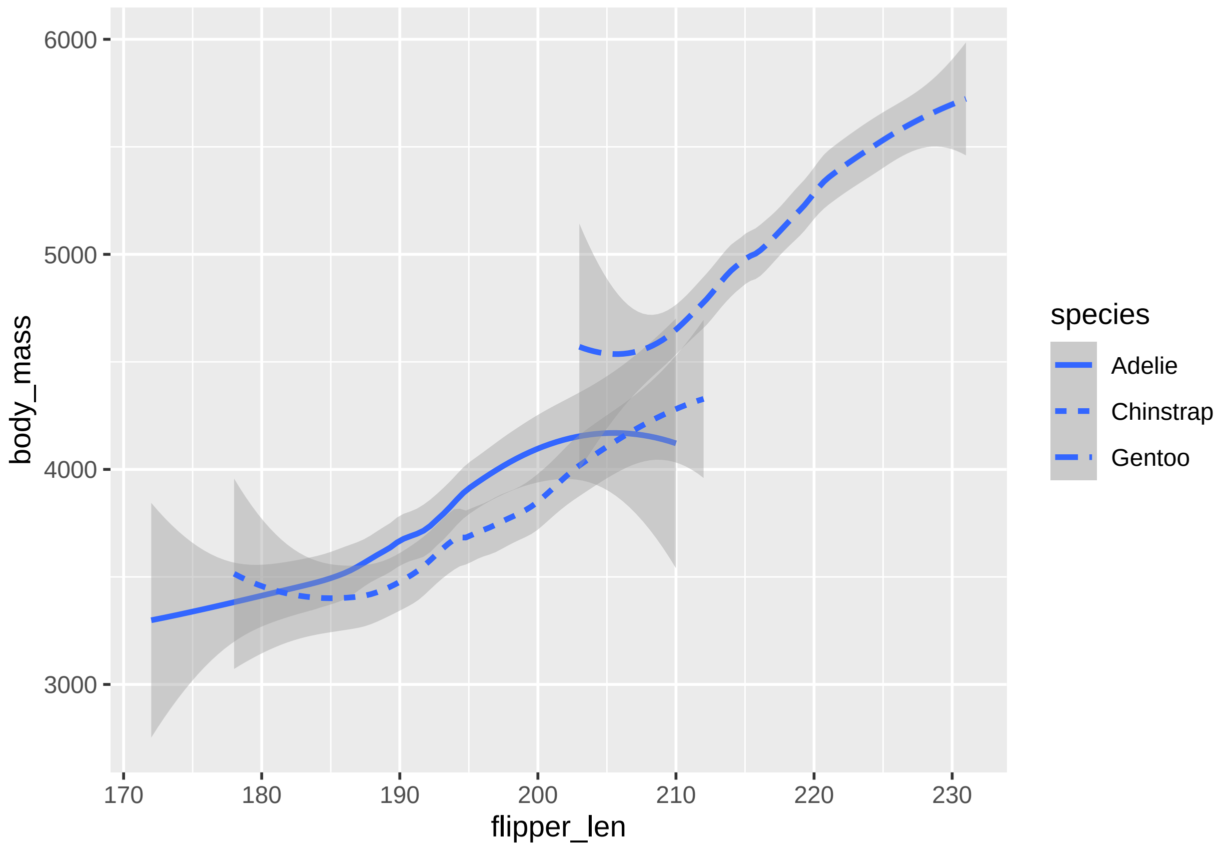


Now we want to plot a separate line for each species of penguin. We can add the line type aesthetic to our code and map it to the variable species.

```
ggplot (data = penguins) +geom_smooth(mapping = aes(x = flipper_len, y = body_mass, linetype = species)
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```

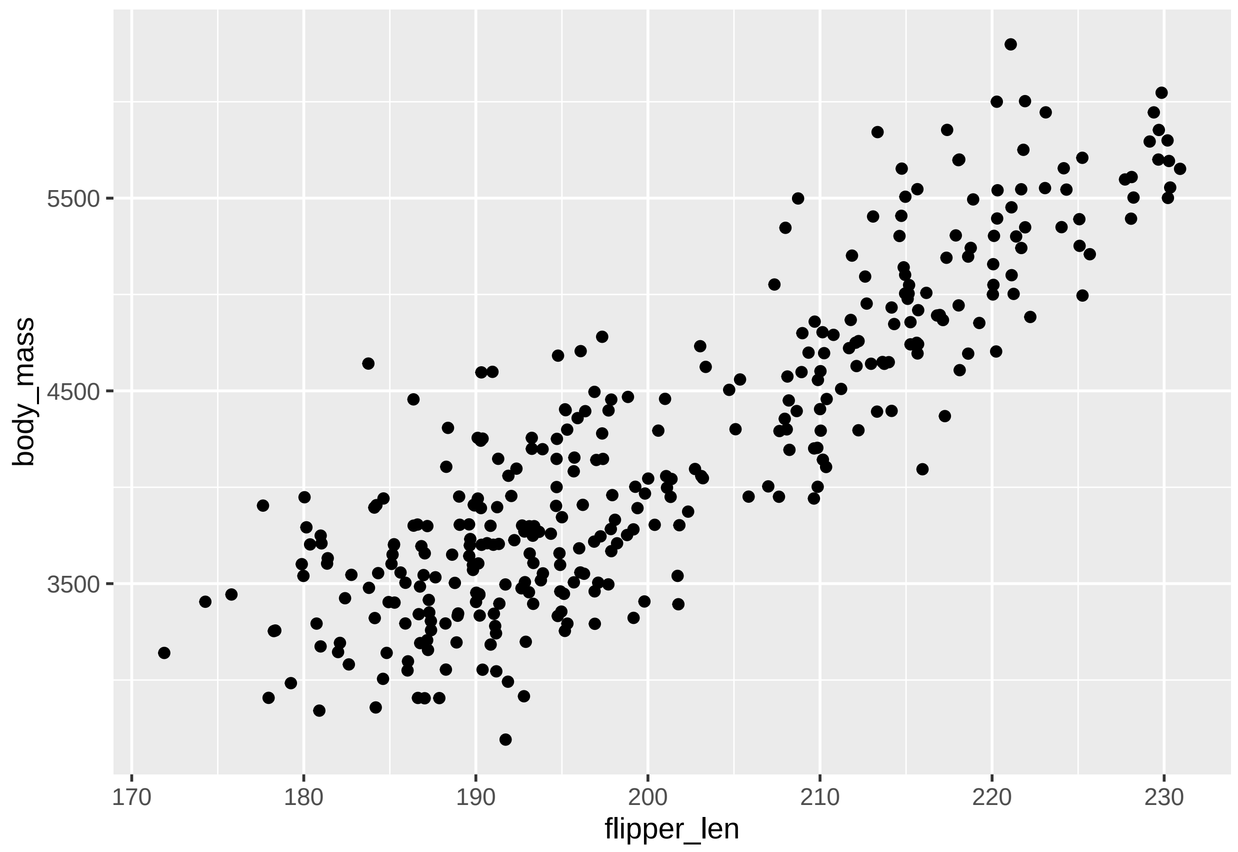


`geom_smooth` will draw a different line with a different line type for each species of penguin. The legend shows how each line type matches with each species. The plot clearly shows the trend for each species.

The `geom_jitter` function creates a scatter plot and then adds a small amount of random noise to each point in the plot. *Jittering* helps us deal with over-plotting, which happens when the data points in a plot overlap with each other. Jittering makes the points easier to find. And for that we run the following code

```r
ggplot(data = penguins) +
  geom_jitter(mapping = aes(x = flipper_len, y = body_mass))
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



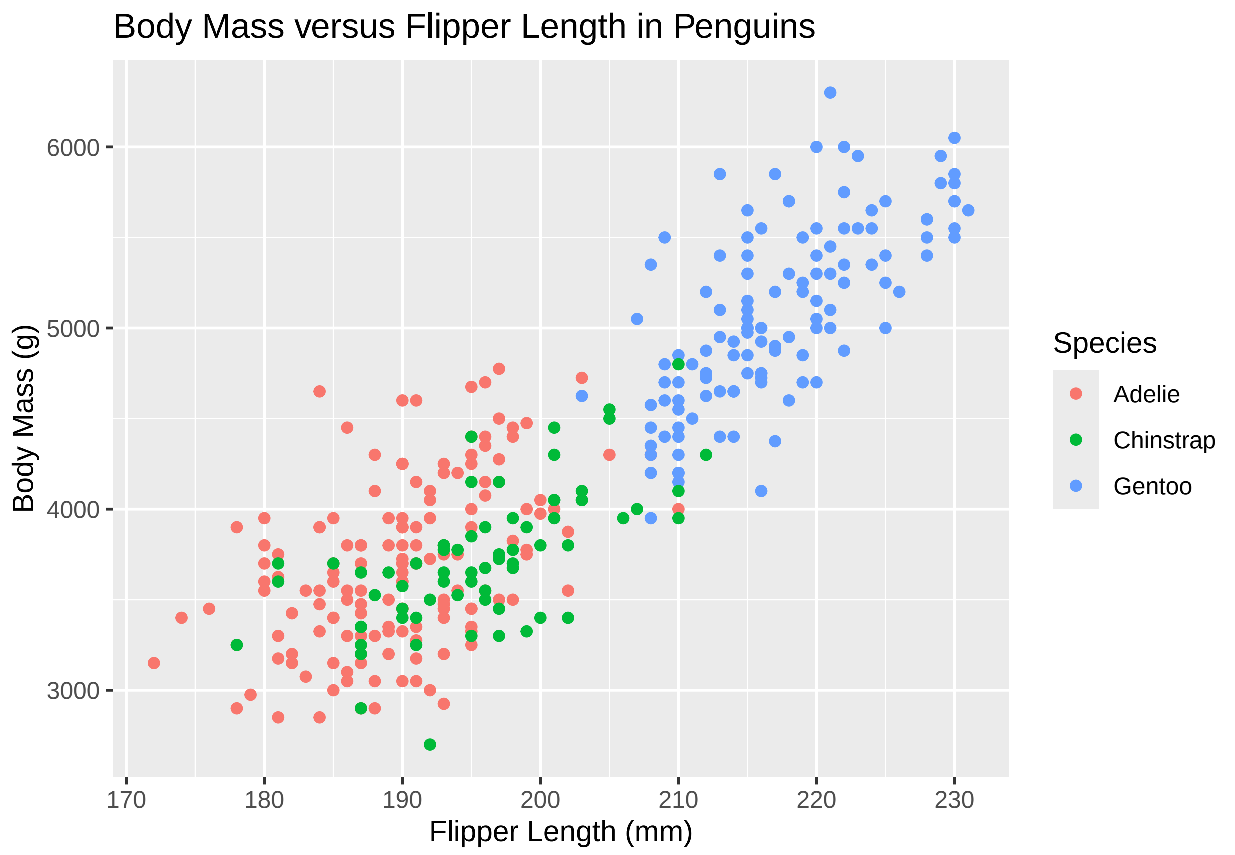
6. Labels and Annotations

Annotate means to add notes to a document or diagram to explain or comment upon it. In ggplot 2 adding annotations to your plot can help explain the plot's purpose or highlight important data.

1. LABELS:
Labels include titles, subtitles, and captions. To add a title to our plot that shows the relationship between body mass and flipper length for the three penguin species, we use the following code:

```
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass, color = species)) +
  labs(title = "Body Mass versus Flipper Length in Penguins",
       x = "Flipper Length (mm)",
       y = "Body Mass (g)",
       color = "Species")
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

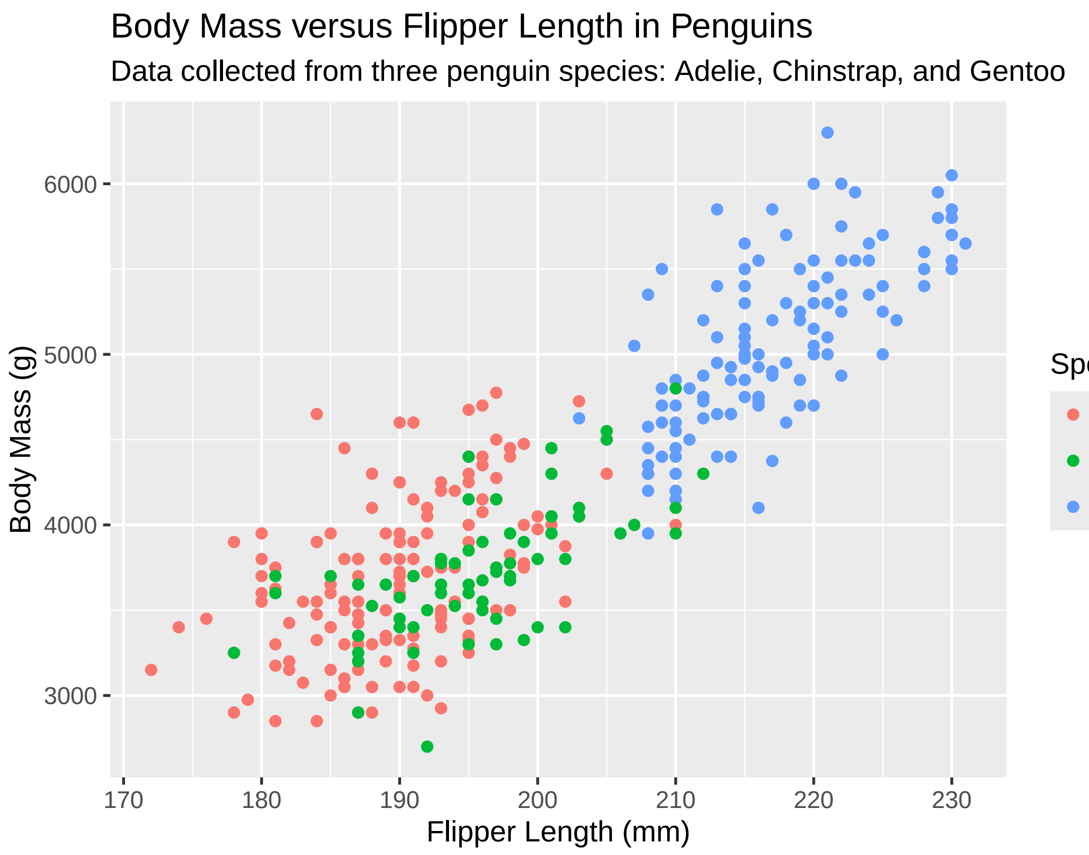


R automatically displays the title at the top of the plot.

We can also add a subtitle to our plot to highlight important information about our data by using the following code:

```
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass, color = species)) +
  labs(title = "Body Mass versus Flipper Length in Penguins",
       subtitle = "Data collected from three penguin species: Adelie, Chinstrap, and Gentoo",
       x = "Flipper Length (mm)",
       y = "Body Mass (g)",
       color = "Species")
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

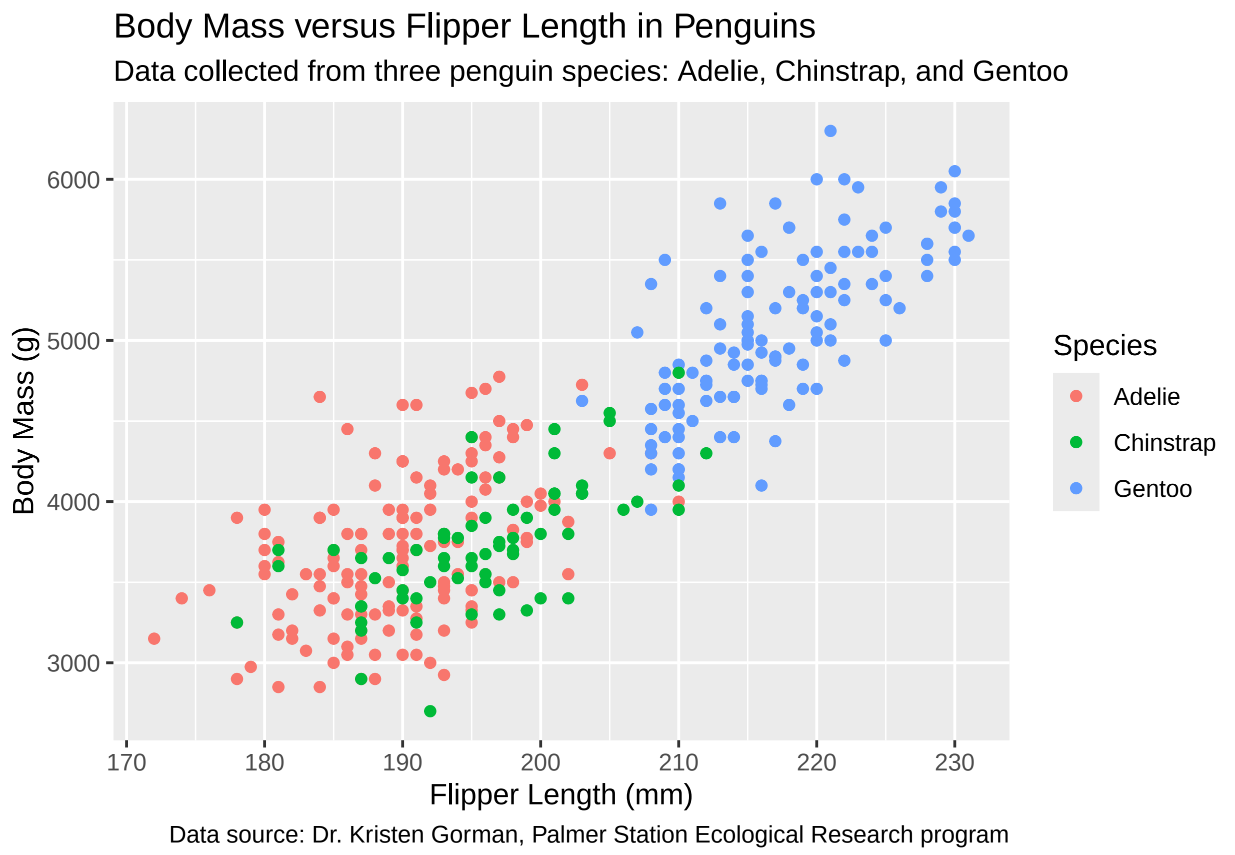


R automatically displays the subtitle just below the title.

We can add a caption to our plot in the same way. Captions let us show the source of our data. The palmer penguins data was collected from 2007 to 2009 by Dr.Kristen Gorman, a member of the Palmer Station Long Term Ecological Research program. Let's cite Dr. Gorman in our caption.

```
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass, color = species)) +
  labs(title = "Body Mass versus Flipper Length in Penguins",
       subtitle = "Data collected from three penguin species: Adelie, Chinstrap, and Gentoo",
       caption = "Data source: Dr. Kristen Gorman, Palmer Station Ecological Research program",
       x = "Flipper Length (mm)",
       y = "Body Mass (g)",
       color = "Species")
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



R automatically displays the caption at the bottom right of our plot.

2. ANNOTATIONS:

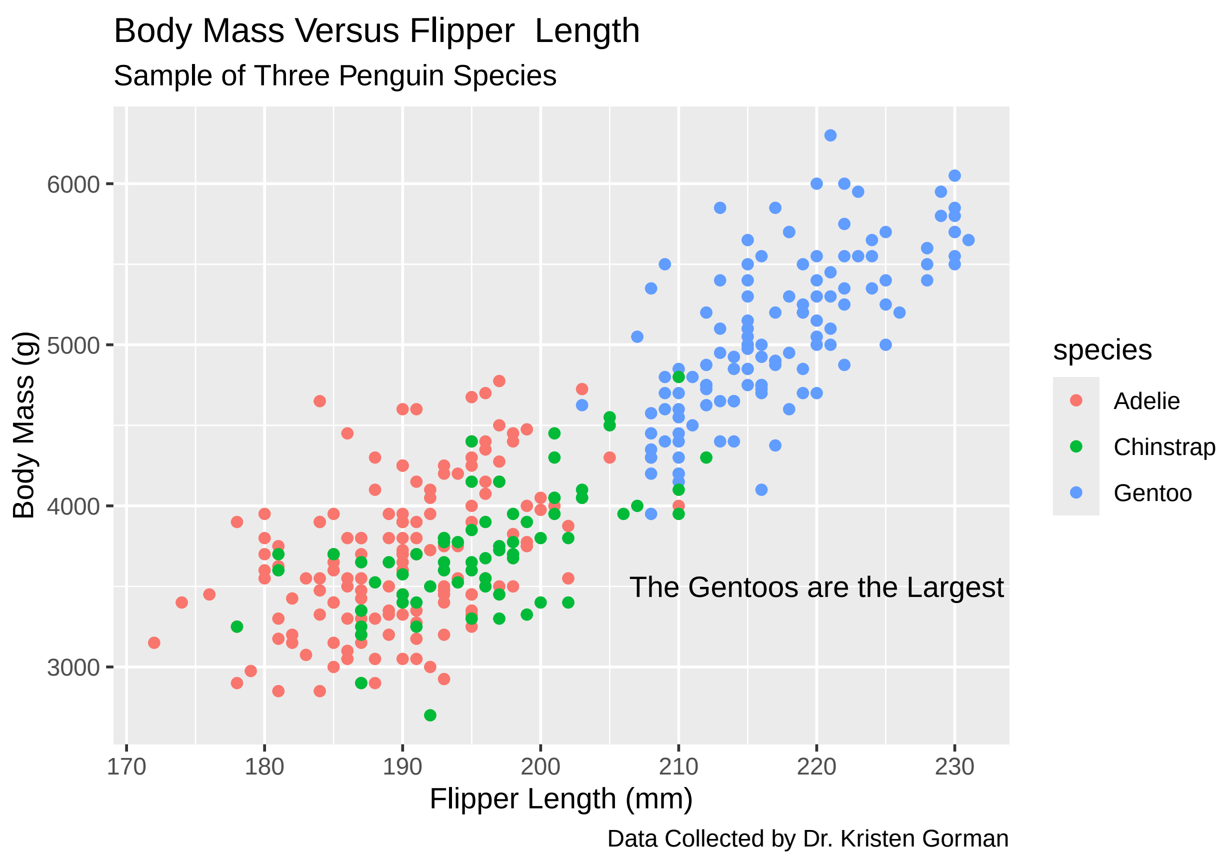
Titles, subtitles, and captions are labels that we put outside of the grid of our plot to indicate important information. If we want to put text inside the grid to call out specific data points, we can use the `annotate` function.

For example, let's say we want to highlight the data from the Gentoo penguins. We can use the annotate function to add some text next to the data points that refer to the Gentoos. This text will clearly communicate what the plot shows and reinforce an important part of our data.

In the parentheses of the annotate function, we've got information on the type of label, the specific location of the label and the context of the label. In this case, we want to write a text label. We also want to place it near the Gentoo data points. Let's put it at the following coordinates: x-axis equals 220 millimeters and y-axis equals 3,500 grams.

```r
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y =body_mass, color = species)) +
  labs(title = "Body Mass Versus Flipper  Length",
  subtitle = "Sample of Three Penguin Species",
  x = "Flipper Length (mm)",
       y = "Body Mass (g)",
       caption = "Data Collected by Dr. Kristen Gorman") +
  annotate("text", x = 220, y = 3500, label = "The Gentoos are the Largest")
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

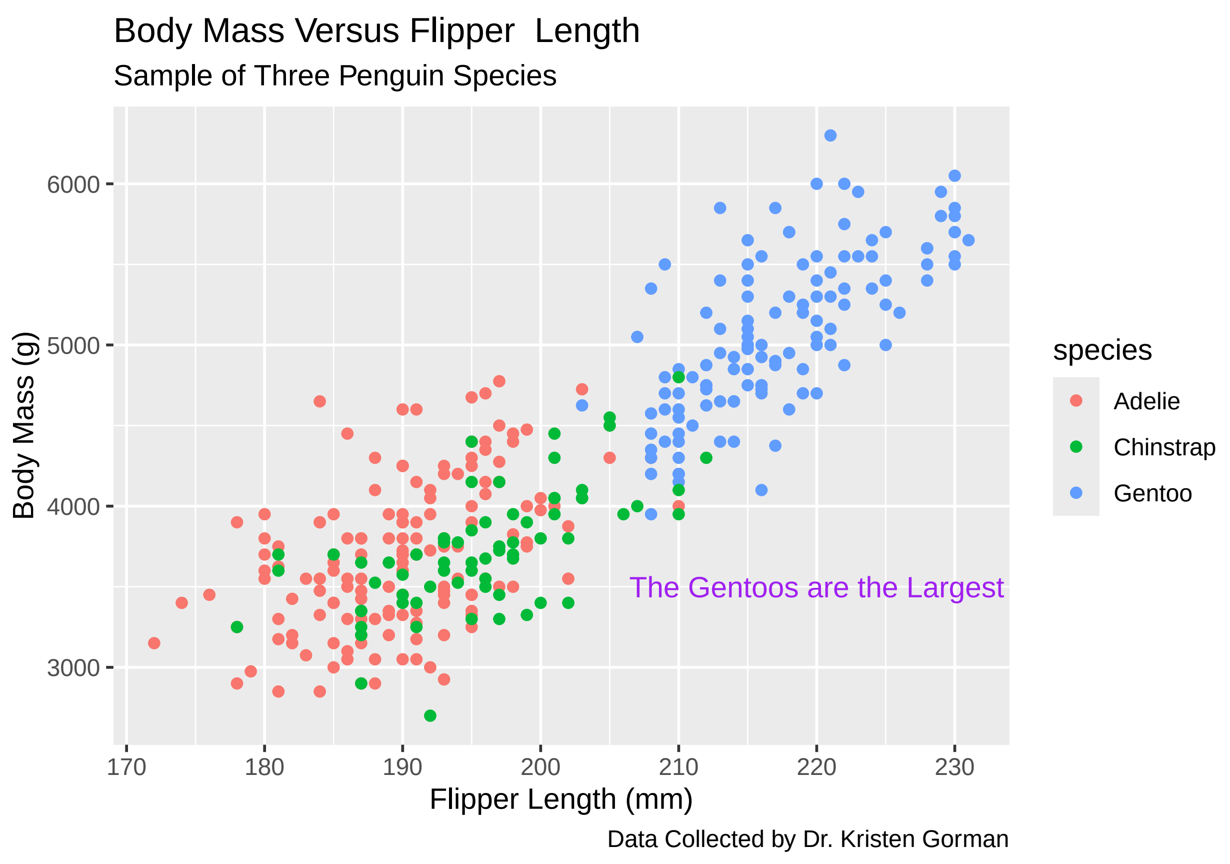


R automatically places the text label on the correct coordinates in our plot.

We can customize our annotation even more. Let's say we want to change the color of our text. Well, we can add color equals followed by the name of the color. Let's try purple.

```r
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y =body_mass, color = species)) +
  labs(title = "Body Mass Versus Flipper  Length",
  subtitle = "Sample of Three Penguin Species",
  x = "Flipper Length (mm)",
       y = "Body Mass (g)",
       caption = "Data Collected by Dr. Kristen Gorman") +
  annotate("text", x = 220, y = 3500, label = "The Gentoos are the Largest", color = "purple")
```

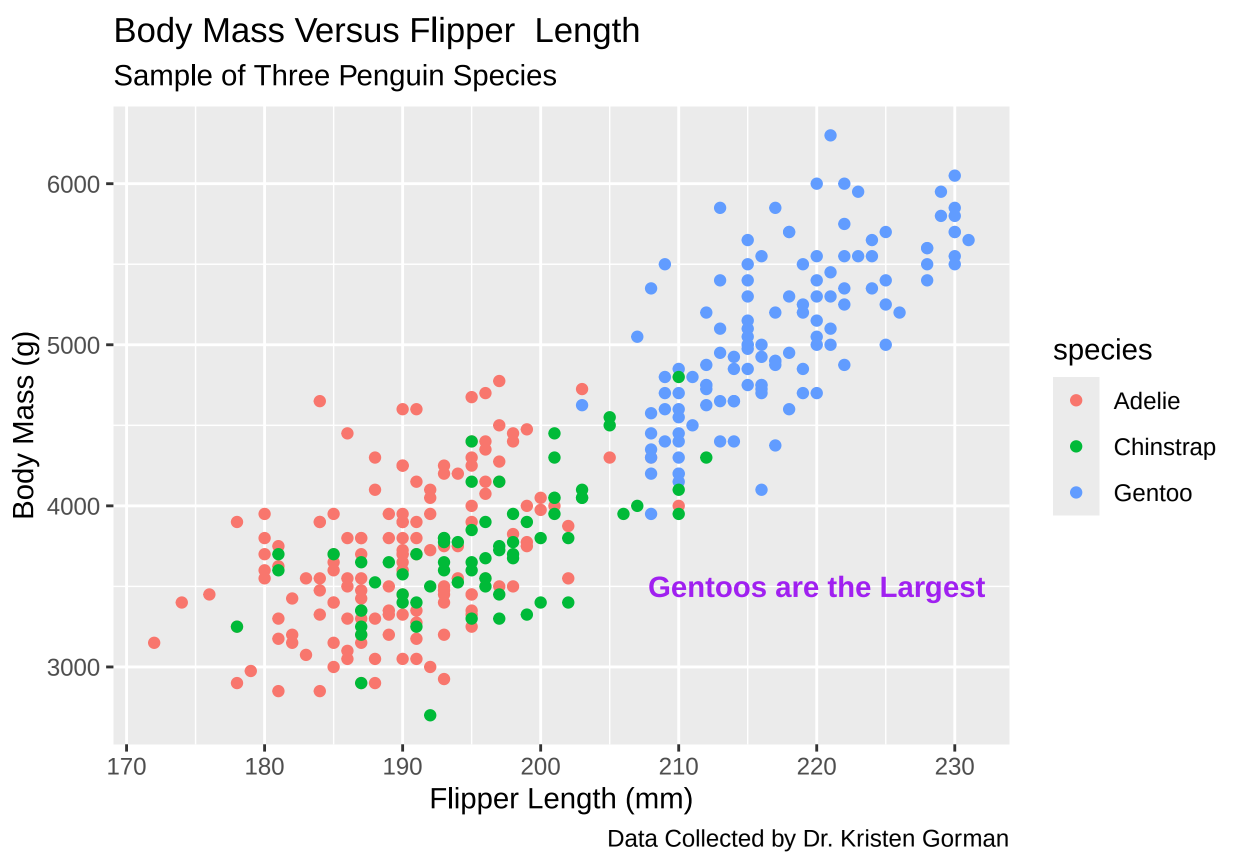
```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



We can also change the font style and size of our text. Use font face and size to write the code. Let's bold our text and make it a little larger.

```
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y =body_mass, color = species)) +
  labs(title = "Body Mass Versus Flipper  Length",
  subtitle = "Sample of Three Penguin Species",
  x = "Flipper Length (mm)",
       y = "Body Mass (g)",
       caption = "Data Collected by Dr. Kristen Gorman") +
  annotate("text", x = 220, y = 3500, label = "Gentoos are the Largest",
           color = "purple", fontface = "bold", size = 4)
```

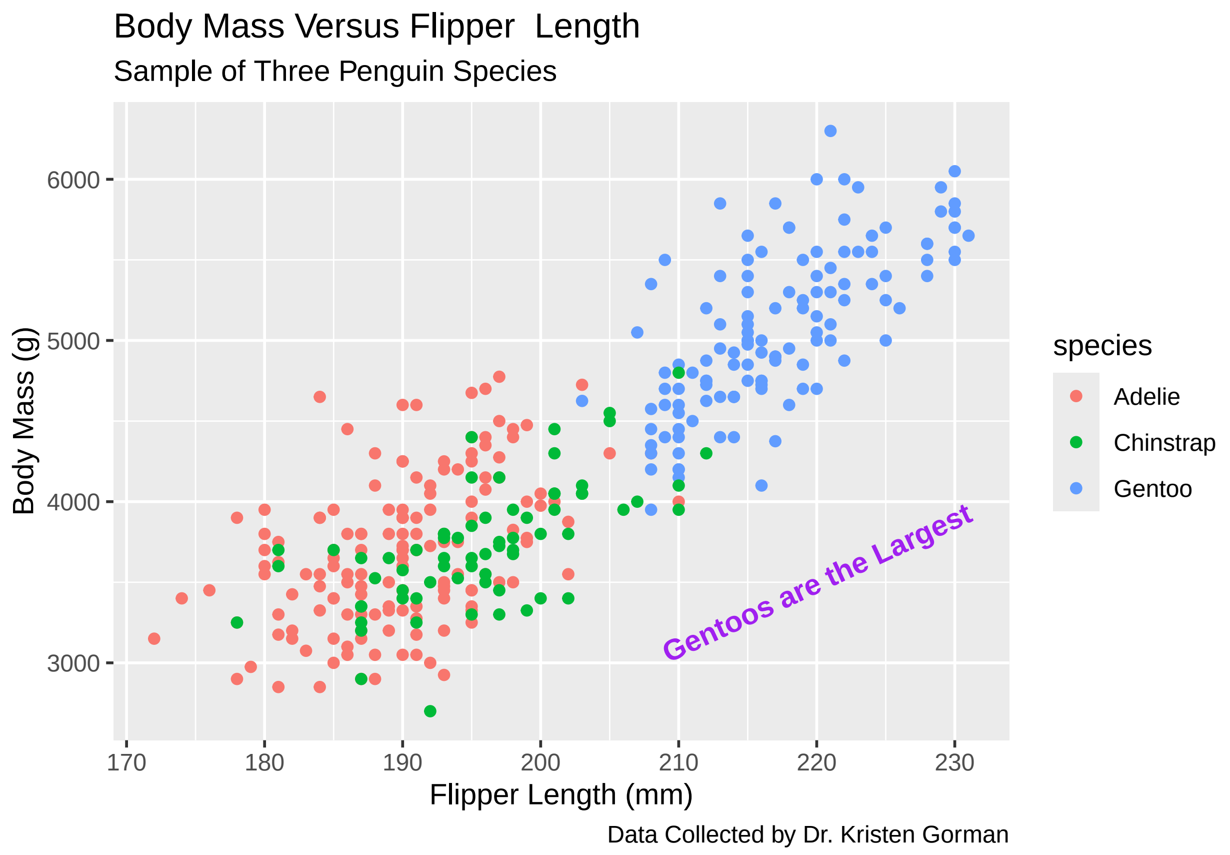
```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



We can even change the angle of our text. For example, we can tilt our text at a 25 degree angle to line it up with our data points.

```
ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y =body_mass, color = species)) +
  labs(title = "Body Mass Versus Flipper  Length",
  subtitle = "Sample of Three Penguin Species",
  x = "Flipper Length (mm)",
       y = "Body Mass (g)",
       caption = "Data Collected by Dr. Kristen Gorman") +
  annotate("text", x = 220, y = 3500, label = "Gentoos are the Largest",
           color = "purple", fontface = "bold", size = 4, angle = 25)
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



7. Store your plot as a variable in R

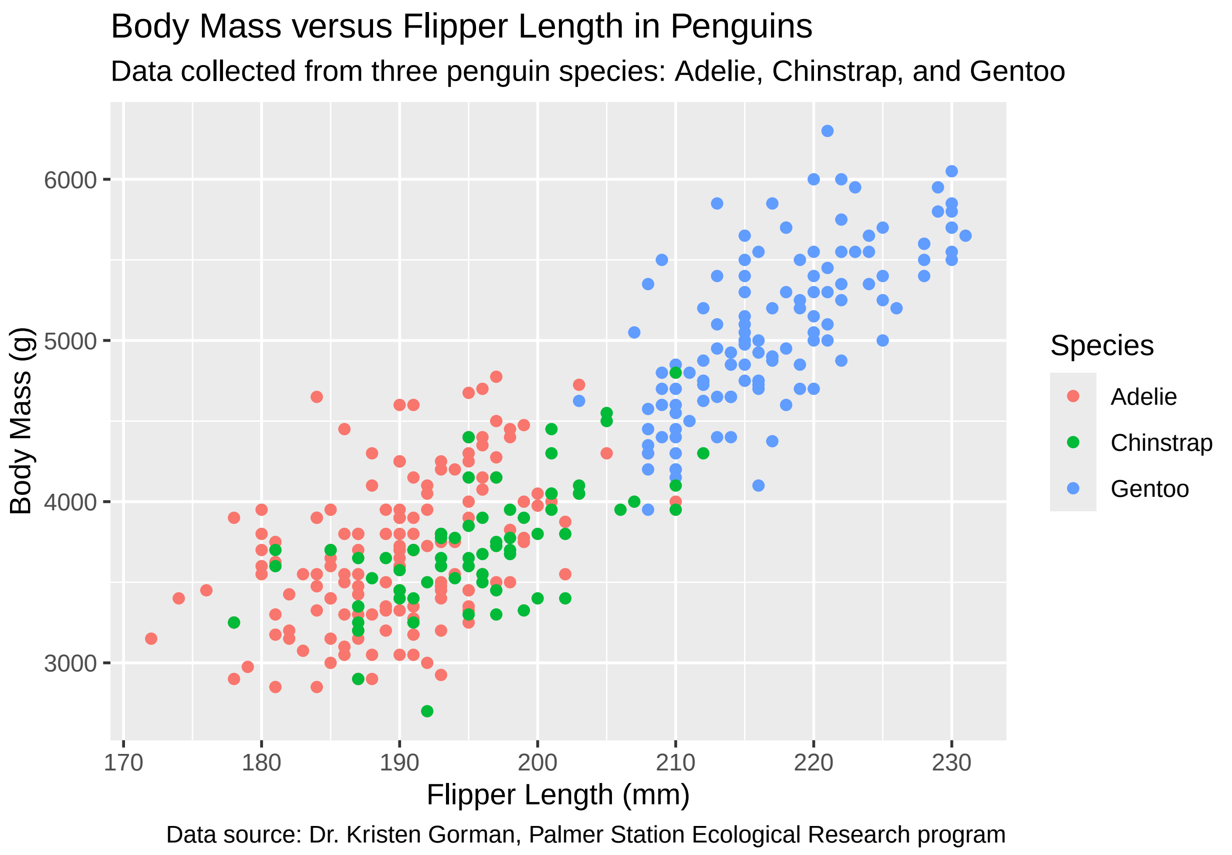
You can store your plot as a variable in R. This is useful when you want to save your plot and use it later or when you want to make changes to your plot without having to rewrite all the code. You can store your plot as a variable by assigning it to a name using the assignment operator, which is the less than sign followed by the minus sign. For example, let's store our scatter plot of body mass versus flipper length as a variable called `penguin_plot`.

```
penguin_plot <- ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass, color = species)) +
  labs(title = "Body Mass versus Flipper Length in Penguins",
       subtitle = "Data collected from three penguin species: Adelie, Chinstrap, and Gentoo",
       caption = "Data source: Dr. Kristen Gorman, Palmer Station Ecological Research program",
       x = "Flipper Length (mm)",
       y = "Body Mass (g)",
       color = "Species")
```

Now we've got a variable called `penguin_plot` that contains our plot. To display the plot, we just need to type the name of the variable and run the code.

```
penguin_plot
```

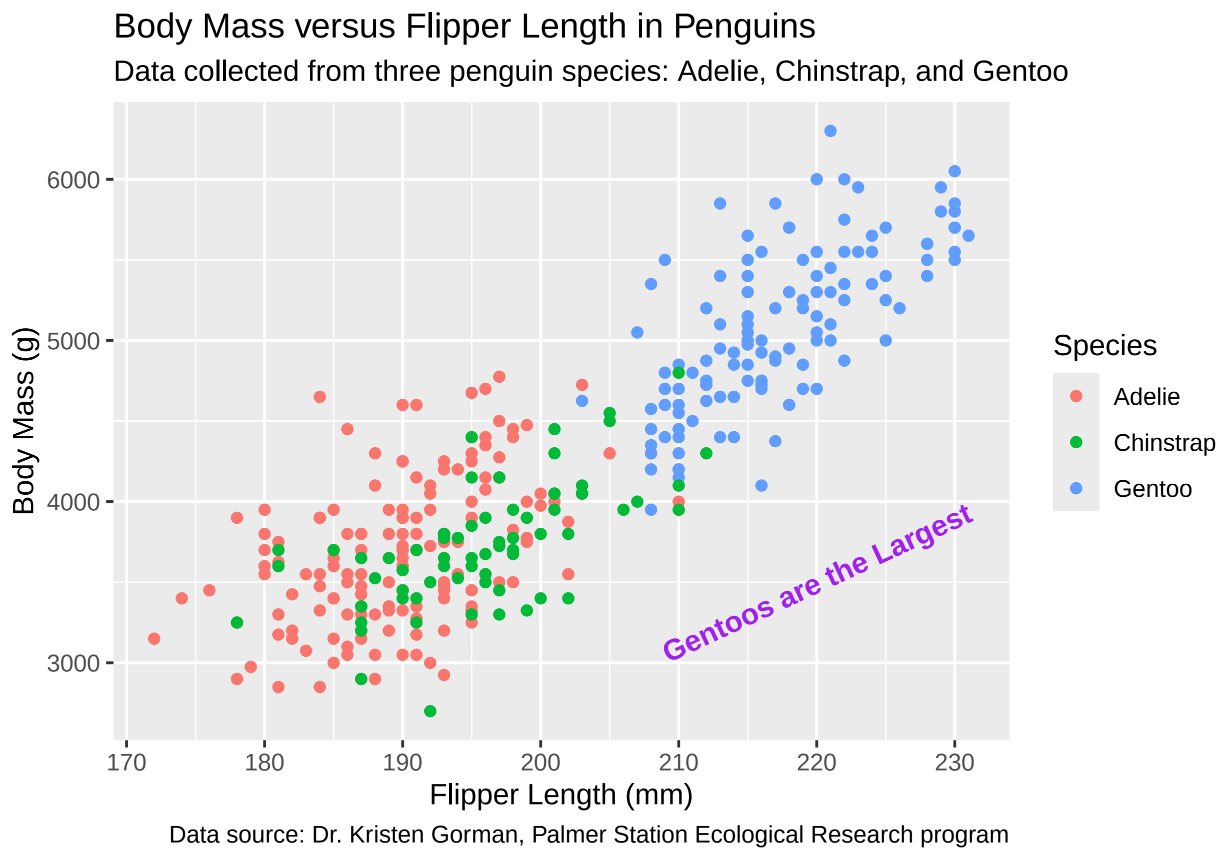
```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



Now, instead of writing all the code again, we can just call `penguin_plot` and add an annotation to it like this:

```
penguin_plot +
  annotate("text", x = 220, y = 3500, label = "Gentoos are the Largest",
           color = "purple", fontface = "bold", size = 4, angle = 25)
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



8. Saving your visualisations using ggplot2

You can save your visualizations in R using the `ggsave` function from the ggplot2 package. The `ggsave` function allows you to save your plots in various formats, including PNG, PDF, JPEG, and more. Here's how to use it:

1. Create your plot and store it as a variable (optional but recommended).

```
#  Creating the plot and storing it as a variable
penguin_plot <- ggplot(data = penguins) +
  geom_point(mapping = aes(x = flipper_len, y = body_mass, color = species)) +
  labs(title = "Body Mass versus Flipper Length in Penguins",
       subtitle = "Data collected from three penguin species: Adelie, Chinstrap, and Gentoo",
       caption = "Data source: Dr. Kristen Gorman, Palmer Station Ecological Research program",
       x = "Flipper Length (mm)",
       y = "Body Mass (g)",
       color = "Species")
```

2. Use the `ggsave` function to save your plot to a file.The saved files will appear in your working directory unless you specify a different path.You can then click on the file to open it. Here's an example of how to save a plot using `ggsave`:

```
# (a). Saving the plot as a PNG file
ggsave("penguin_plot.png", plot = penguin_plot, width = 8, height = 6, dpi = 300)
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

```r
# dpi is dots per inch, it controls the resolution of the image.You can adjust it based on your needs.
# width and height are in inches, you can adjust them based on your needs.
# width and height control the size of the saved image.

# (b). Saving the plot as a PDF file
ggsave("penguin_plot.pdf", plot = penguin_plot, width = 8, height = 6)
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

```r
# (c). Saving the plot as a JPEG file
ggsave("penguin_plot.jpg", plot = penguin_plot, width = 8, height = 6, dpi = 300)
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

9. Working with other geometries in ggplot2

Now that we've covered the basics of creating plots in `ggplots` by using scatter plots to explain in detail, let's explore some other types of plots you can create using different geoms. I now present to you some other geoms in ggplot2 that you can use to create other types of plots on top of the scatter plots we have treated. I will cover bargraphs, piecharts, histograms, boxplots, violin plots and line graphs.

9.1 Bar Graphs

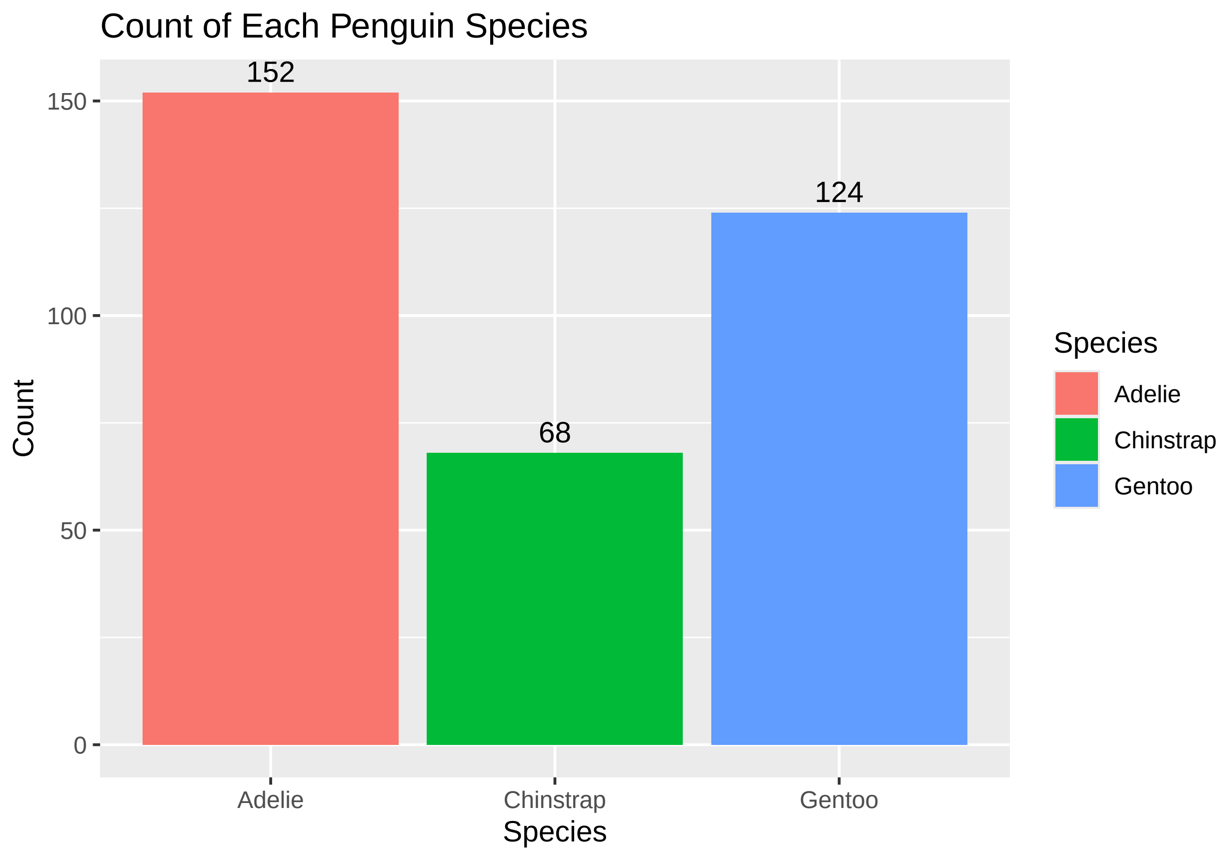
Bar graphs are used to display and compare the frequency, count, or other measures (like mean, sum) of different categories. Each bar represents a category, and the height of the bar corresponds to the value of that category. Bar graphs are useful for visualizing categorical data and making comparisons between different groups.

We shall use the `geom_bar` function to create bar graphs. The `geom_bar` function has two main ways of working: one is to count the number of occurrences of each category in a single variable, and the other is to summarize a second variable for each category in the first variable.

(a). Creating a simple bar graph to show the count of each penguin species

```r
ggplot(data = penguins) +
  geom_bar(mapping = aes(x = species, fill = species)) +
  geom_text(stat = "count", aes(x = species, label = ..count..), vjust = -0.5) +
  labs(title = "Count of Each Penguin Species",
       x = "Species",
       y = "Count",
       fill = "Species")
```

```
## Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(count)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

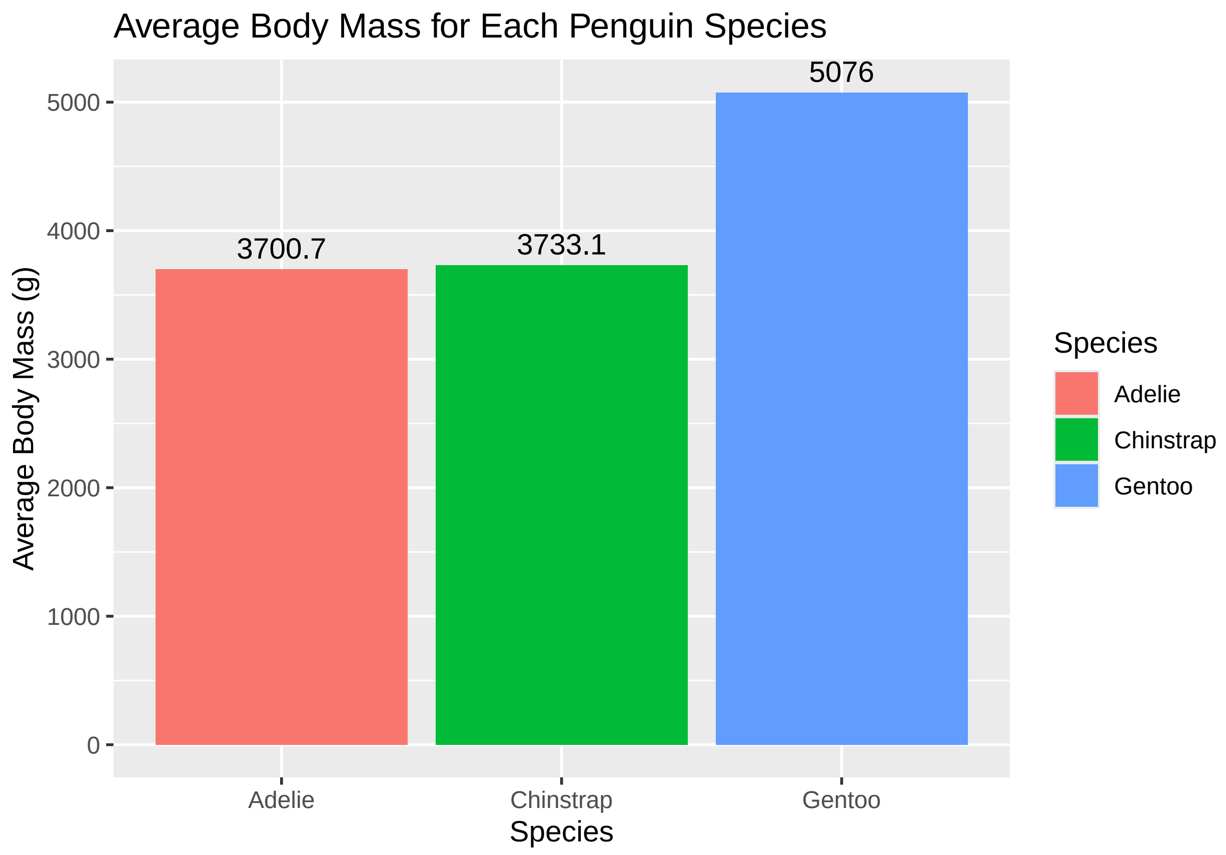


```
# fill aesthetic is used to fill the bars with different colors based on the species.
# count is the default statistic for geom_bar, so we don't need to specify it explicitly.
# In the geom_text function, we use stat = "count" when we want to use the count statistic for
# the text labels.
# The aes function inside geom_text maps the x aesthetic  to species and
# the label aesthetic to ..count.. to display the count of each species on top of the bars.
# ..count.. is a special variable in ggplot2 that represents the count of observations in
# each category.
# vjust is used to adjust the vertical position of the text labels.
# A negative value of vjust moves the labels above the bars.
```

(b). Creating a bar graph to show the average body mass for each penguin species

```r
ggplot(data = penguins) +
geom_bar(mapping = aes(x = species, y = body_mass, fill = species), stat = "summary", fun = "mean") +
geom_text(stat = "summary", aes(x = species, y = body_mass, label = round(..y.., 1)), fun = "mean",
          vjust = -0.5) +
labs(title = "Average Body Mass for Each Penguin Species",
       x = "Species",
       y = "Average Body Mass (g)",
       fill = "Species")
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_summary()`).
## Removed 2 rows containing non-finite outside the scale range
## (`stat_summary()`).
```



```
# stat = "summary" tells ggplot2 to summarize the data instead of counting it.
# fun = "mean" specifies that we want to calculate the mean (average) of body_mass for each species.
# y aesthetic is used to specify the variable we want to summarize (body_mass).
# label aesthetic is used to display the average body mass on top of the bars.
# ..y.. is a special variable in ggplot2 that represents the summarized value (the mean body mass) for
# each category.
# 1 is used to round the average body mass to one decimal place for better readability.
```

9.2 Pie Charts

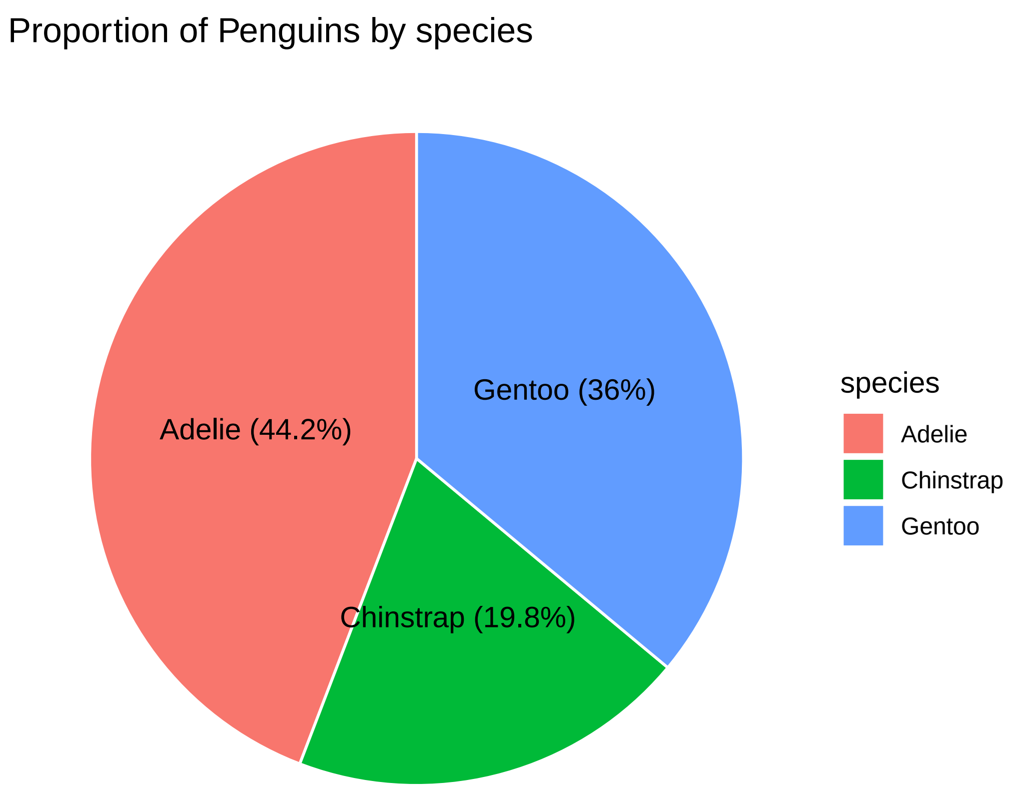
Pie charts are circular bar charts divided into sectors, where each sector represents a proportion of the whole. The size of each sector corresponds to the percentage or fraction of the total that the category represents. Pie charts are useful for showing the relative proportions of different categories in a dataset.

We can create pie charts in ggplot2 by using the `geom_bar` function along with the `coord_polar` function to transform a bar chart into a pie chart. Here's how to do it:

Creating a piechart to show the proportion of penguins by species

```
# (a). Prepare data for pie chart by data wrangling it first,assign it to a variable called pie_data
pie_data <- penguins %>%
  filter(!is.na(species)) %>% # remove any missing values in the species column
  count(species) %>% # count the number of occurrences of each species
  mutate(perc = n / sum(n) * 100,
         label = paste0(species, " (", round(perc, 1), "%)"))
# mutate function is used to create new columns "perc" and "label".
```

```r
# calculate the percentage(perc) and create labels for each species
# paste0 function is used to concatenate the species name with its percentage.
# round function is used to round the percentage to one decimal place for better readability.

# (b). Create the pie chart from wrangled data
ggplot(pie_data, aes(x = "", y = perc, fill = species)) +
  geom_col(width = 1, color = "white") +
  coord_polar(theta = "y") +
  geom_text(aes(label = label), position = position_stack(vjust = 0.5)) +
  labs(title = "Proportion of Penguins by species") +
  theme_void()
```



```r
# geom_col function is used to create the bars for the pie chart.
# width = 1 makes the bars full width to create a complete pie chart.
# color = "white" adds a white border between the pie slices for better visibility.
# coord_polar function transforms the bar chart into a pie chart.
# theta = "y" specifies that the y aesthetic should be used to determine the angle of the pie slices.
# geom_text function is used to add labels to each pie slice.
# position = position_stack(vjust = 0.5) centers the labels within each pie slice.
# theme_void function removes the background, gridlines, and axes for a cleaner look.
```

9.3 Histograms

Histograms are graphical representations of the distribution of a continuous variable. They display the frequency or count of data points that fall within specified ranges or bins. Each bar in a histogram represents the number of observations that fall within a particular bin.

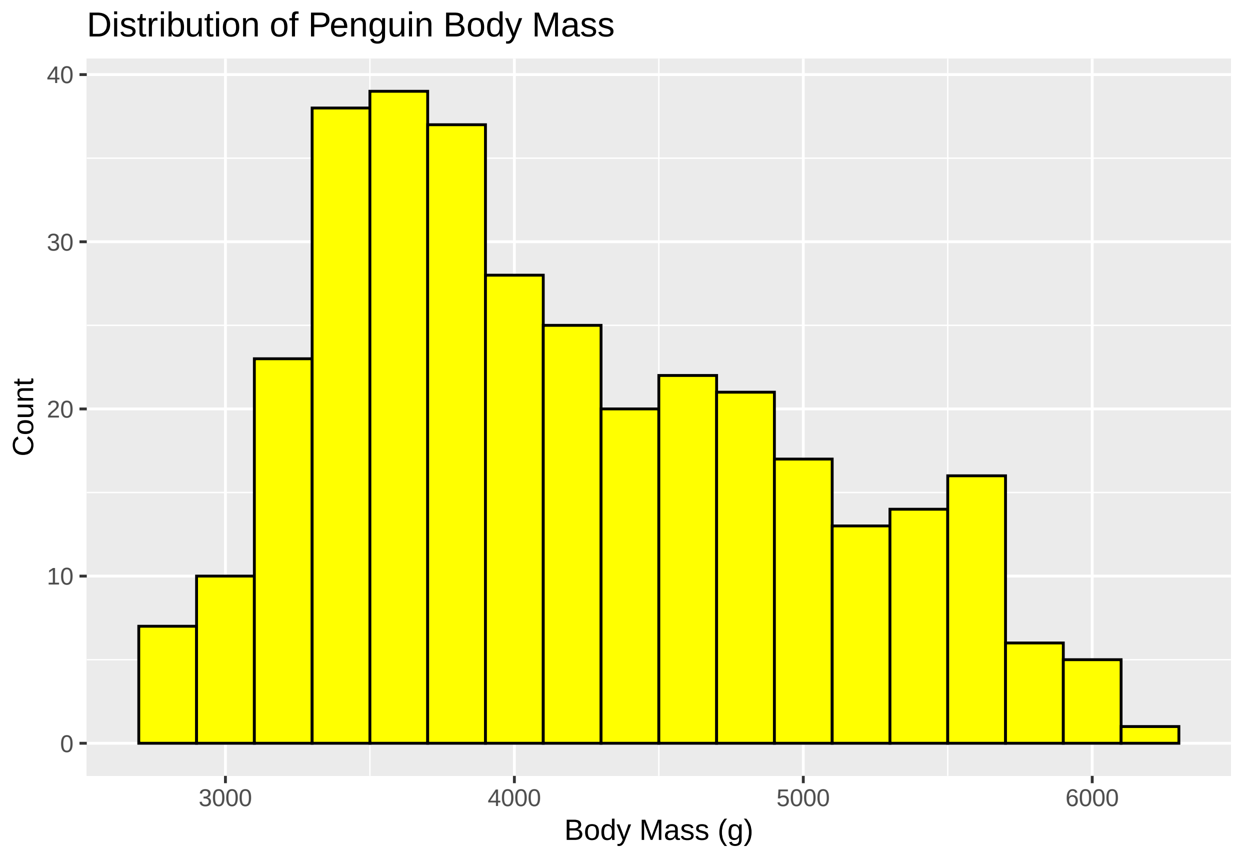
Bins are intervals that divide the range of the data into smaller segments. The choice of bin width can affect the appearance of the histogram and the insights you can gain from it.Histograms are useful for visualizing the shape,spread, and central tendency of a dataset.

We can create histograms in ggplot2 using the `geom_histogram` function. Here's how to do it:

(a). Creating a histogram to show the distribution of penguin body mass

```
ggplot(data = penguins) +
  geom_histogram(mapping = aes(x = body_mass), binwidth = 200, fill = "yellow", color = "black") +
  labs(title = "Distribution of Penguin Body Mass",
       x = "Body Mass (g)",
       y = "Count")
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_bin()`).
```

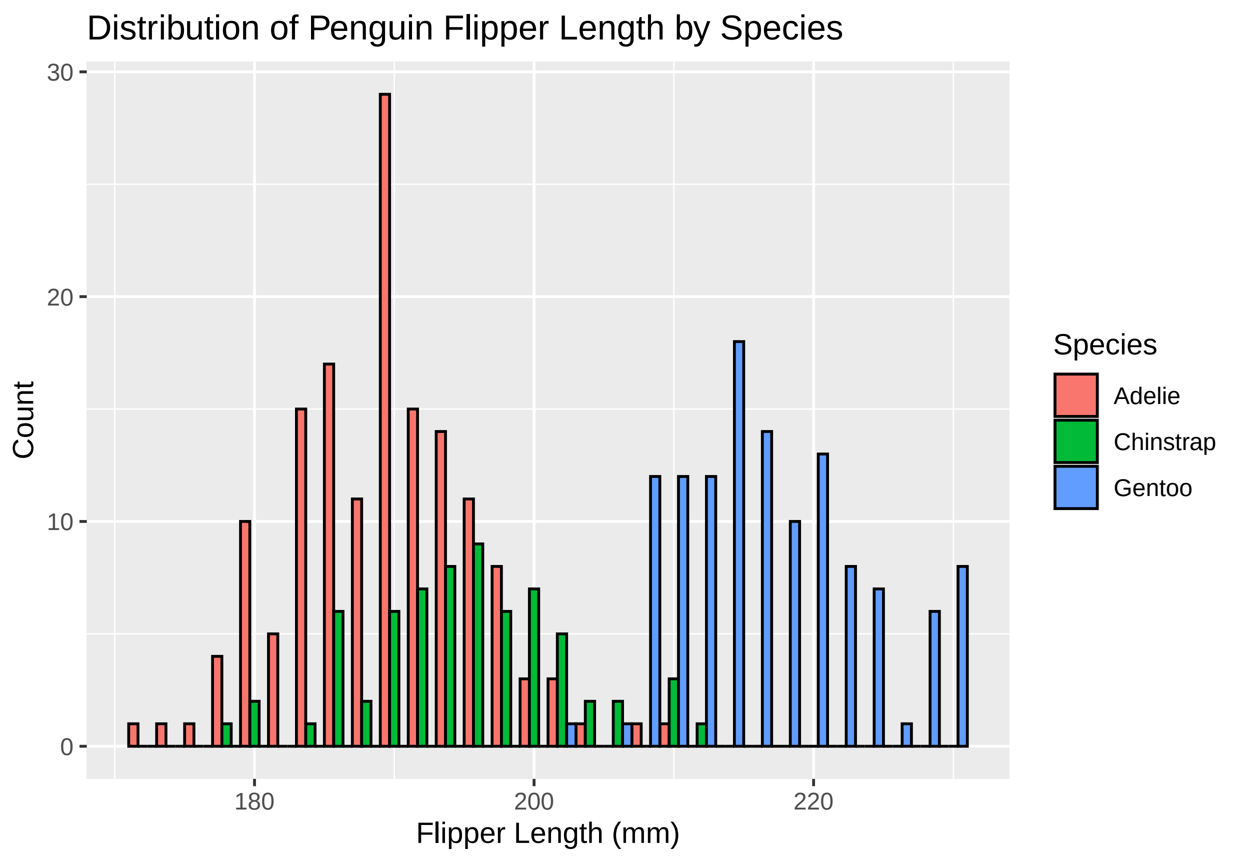


```
# binwidth = 200 specifies the width of each bin in the histogram.
# fill aesthetic is used to fill the bars with yellow color.
# color aesthetic is used to add a black border to each bar for better visibility.
```

(b). Creating a histogram to show the distribution of penguin flipper length by species

```r
ggplot(data = penguins) +
  geom_histogram(mapping = aes(x = flipper_len, fill = species), position = "dodge",
                 binwidth = 2, color = "black") +
  labs(title = "Distribution of Penguin Flipper Length by Species",
       x = "Flipper Length (mm)",
       y = "Count",
       fill = "Species")
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_bin()`).
```

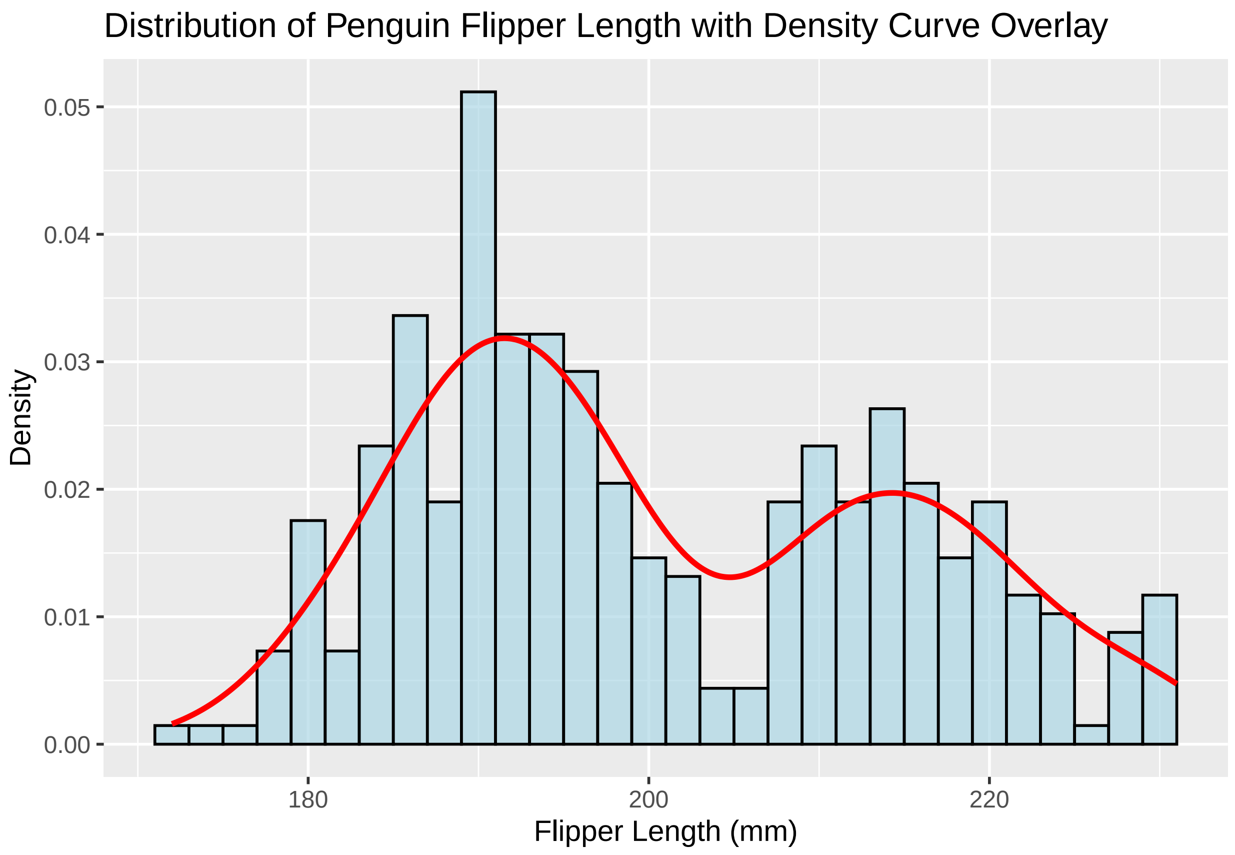


```r
# position = "dodge" places the bars for each species side by side for better comparison.
# dodge means to separate the bars for each species.
```

(c). Creating a histogram to show the distribution of penguin flipper length with density curve overlay

```r
ggplot(data = penguins) +
  geom_histogram(mapping = aes(x = flipper_len, y = ..density..), binwidth = 2,
           fill = "lightblue", color = "black", alpha = 0.7) +
  geom_density(mapping = aes(x = flipper_len), color = "red", size = 1) +
  labs(title = "Distribution of Penguin Flipper Length with Density Curve Overlay",
       x = "Flipper Length (mm)",
       y = "Density")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_bin()`).

## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_density()`).
```



```
# y = ..density.. scales the y-axis to show density instead of count.
# alpha = 0.7 sets the transparency of the bars to 70% for better visibility of the density curve.
# geom_density function is used to add a density curve overlay to the histogram.
# color = "red" sets the color of the density curve to red.
# size = 1 sets the thickness of the density curve.
```

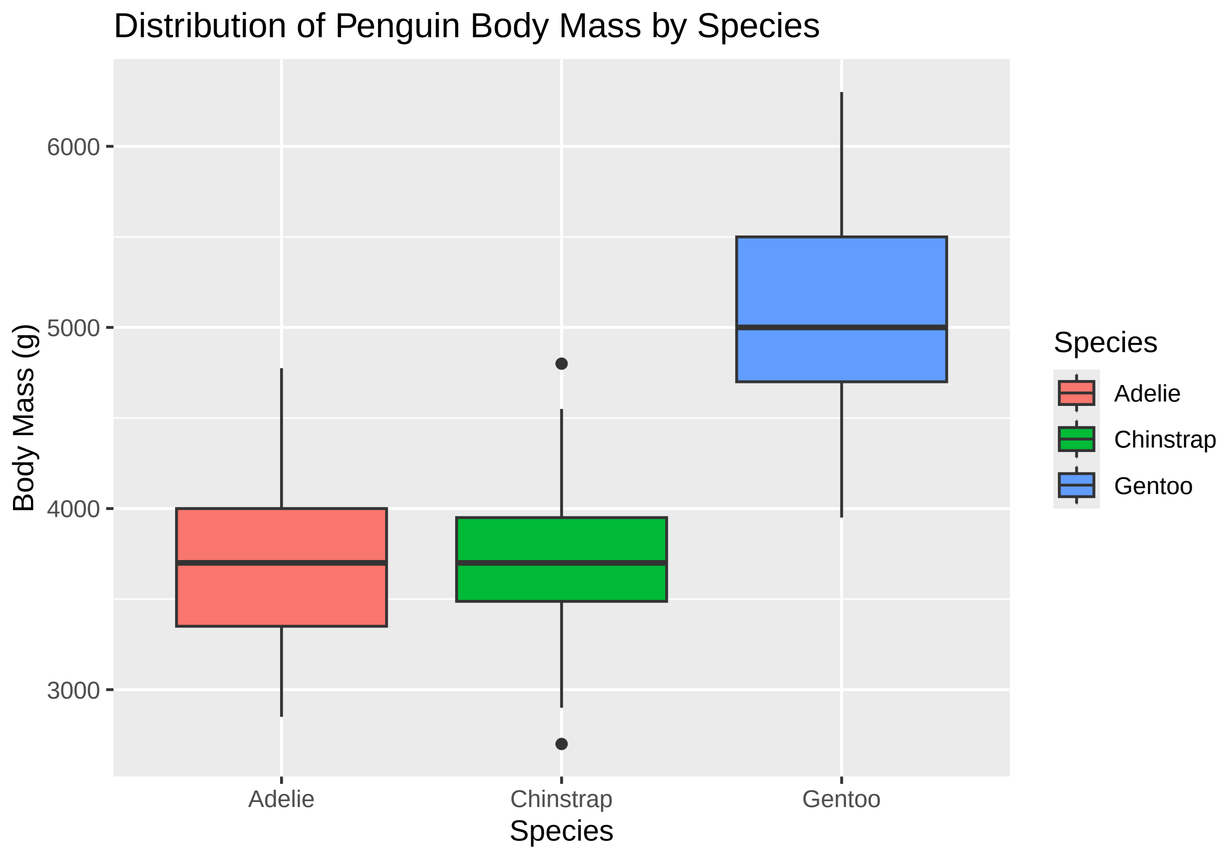
9.4 Boxplots

Boxplots, also known as box-and-whisker plots, are graphical representations of the distribution of a continuous variable. They display the minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum of the data. Boxplots are useful for identifying outliers, comparing distributions between groups, and visualizing the spread and central tendency of the data. We can create boxplots in ggplot2 using the `geom_boxplot` function. Here's how to do it:

(a). Creating a boxplot to show the distribution of penguin body mass by species

```r
ggplot(data = penguins) +
  geom_boxplot(mapping = aes(x = species, y = body_mass, fill = species)) +
  labs(title = "Distribution of Penguin Body Mass by Species",
       x = "Species",
       y = "Body Mass (g)",
       fill = "Species")
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

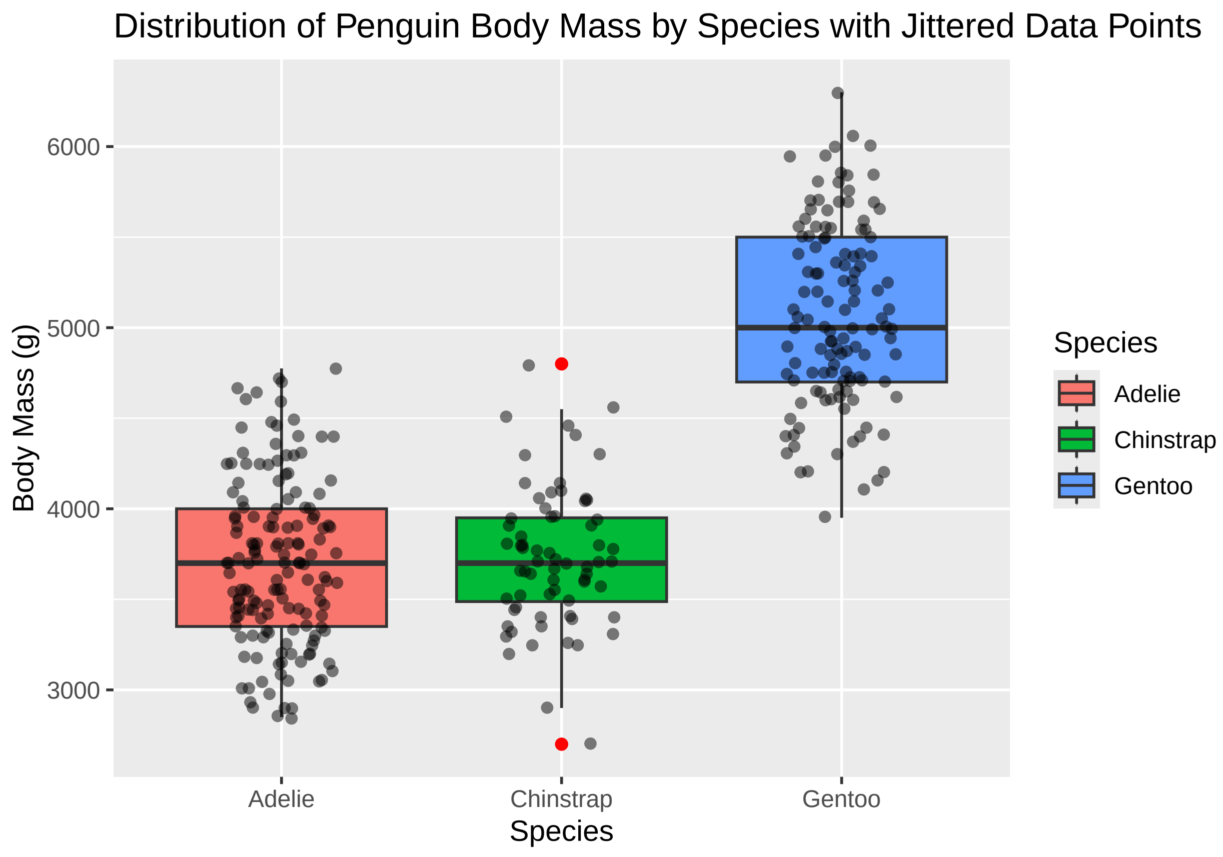


(b). Creating a boxplot to show the distribution of penguin body mass by species with jittered data points overlay

```r
ggplot(data = penguins) +
  geom_boxplot(mapping = aes(x = species, y = body_mass, fill = species),
    outlier.color = "red", outlier.shape = 16, outlier.size = 2) +
  geom_jitter(mapping = aes(x = species, y = body_mass), width = 0.2, alpha = 0.5, color = "black") +
  labs(title = "Distribution of Penguin Body Mass by Species with Jittered Data Points",
       x = "Species",
       y = "Body Mass (g)",
       fill = "Species")
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



```r
# outlier.color = "red" sets the color of the outlier points to red.
# outlier.shape = 16 sets the shape of the outlier points to filled circles.
# outlier.size = 2 sets the size of the outlier points to 2.
# geom_jitter function is used to add jittered data points overlay to the boxplot.
# width = 0.2 sets the amount of horizontal jitter to 0.2 for better visibility.
# alpha = 0.5 sets the transparency of the jittered points to 50% for better visibility.
```

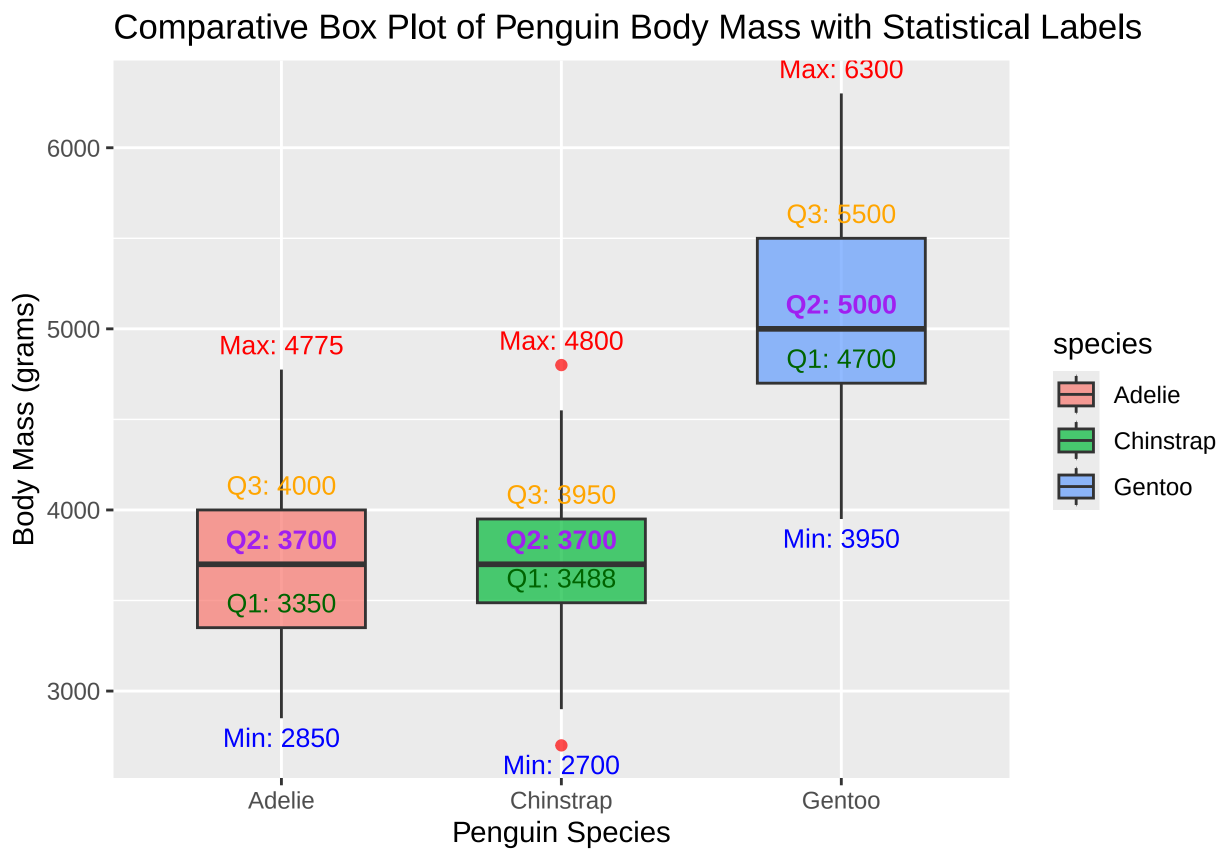
(c). Creating a comparative boxplot to show the distribution of penguin body mass by species with five-number summary labels

```r
#------------------------------------------------------------
# STEP 1: Wrangle data to compute five-number summary for each species
#------------------------------------------------------------
# We'll calculate minimum, first quartile(Q1), median (Q2),third quartile (Q3) and maximum values
#for body_mass for each penguin species.

box_stats <- penguins %>%
  filter(!is.na(body_mass)) %>%          # Remove missing values
  group_by(species) %>%                  # Group by species
  summarise(Min = min(body_mass),
    Q1 = quantile(body_mass, 0.25),
    Median = median(body_mass),
    Q3 = quantile(body_mass, 0.75),
    Max = max(body_mass))

# View the computed summary table
print(box_stats)
```

```
## # A tibble: 3 x 6
##   species     Min    Q1 Median    Q3   Max
##   <fct>     <int> <dbl>  <dbl> <dbl> <int>
## 1 Adelie     2850 3350    3700  4000  4775
## 2 Chinstrap  2700 3488.   3700  3950  4800
## 3 Gentoo     3950 4700    5000  5500  6300
```

```r
#------------------------------------------------------------
# STEP 2: Create a Comparative Box Plot with Labels
#------------------------------------------------------------
# The plot will show each species' body mass distribution,
# and overlay text labels for the five-number summary.

ggplot(penguins, aes(x = species, y = body_mass, fill = species)) +
  geom_boxplot(width = 0.6, alpha = 0.7, outlier.color = "red") +  # Standard box plot
  # Add text labels for each statistic, placed near the corresponding Y-value
  geom_text(
    data = box_stats,
    aes(x = species, y = Min, label = paste0("Min: ", round(Min, 0))),
    vjust = 1.5, color = "blue", size = 3.5
  ) +
  geom_text(
    data = box_stats,
    aes(x = species, y = Q1, label = paste0("Q1: ", round(Q1, 0))),
    vjust = -0.8, color = "darkgreen", size = 3.5
  ) +
  geom_text(
    data = box_stats,
    aes(x = species, y = Median, label = paste0("Q2: ", round(Median, 0))),
    vjust = -0.8, color = "purple", size = 3.5, fontface = "bold"
  ) +
  geom_text(
    data = box_stats,
    aes(x = species, y = Q3, label = paste0("Q3: ", round(Q3, 0))),
    vjust = -0.8, color = "orange", size = 3.5
  ) +
  geom_text(
    data = box_stats,
    aes(x = species, y = Max, label = paste0("Max: ", round(Max, 0))),
    vjust = -0.8, color = "red", size = 3.5
  ) +
  labs(
    title = "Comparative Box Plot of Penguin Body Mass with Statistical Labels",
    x = "Penguin Species",
    y = "Body Mass (grams)"
  )
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```



(d). Creating a comparative boxplot to show the distribution of penguin flipper length by species with five-number summary labels

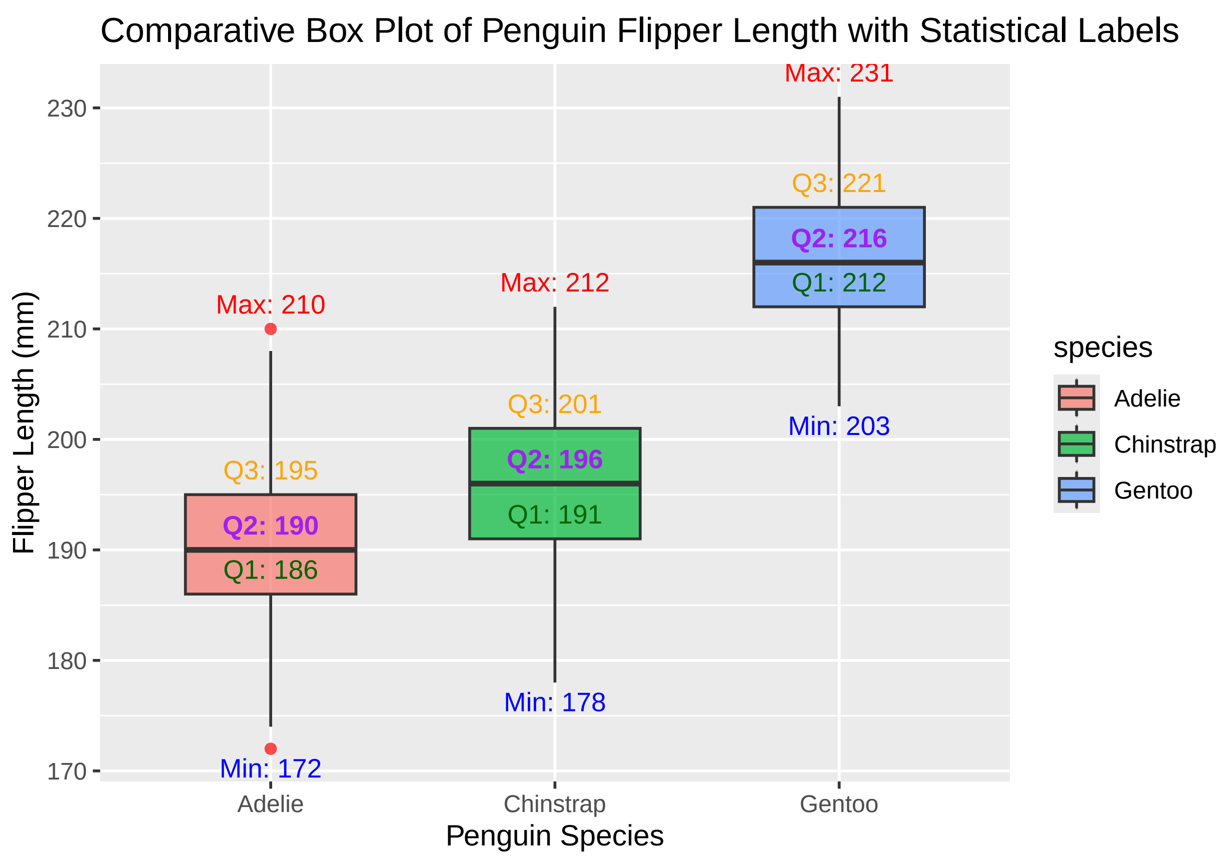
```
#-----------------------------------------------------------
#  Repeat for Flipper Length
#-----------------------------------------------------------
# We do the same for flipper_len:

flipper_stats <- penguins %>%
  filter(!is.na(flipper_len)) %>%
```

```r
  group_by(species) %>%
  summarise(
    Min = min(flipper_len),
    Q1 = quantile(flipper_len, 0.25),
    Median = median(flipper_len),
    Q3 = quantile(flipper_len, 0.75),
    Max = max(flipper_len)
  )

# View the summary for verification
print(flipper_stats)
```

```
## # A tibble: 3 x 6
##   species     Min    Q1 Median    Q3   Max
##   <fct>     <int> <dbl>  <dbl> <dbl> <int>
## 1 Adelie      172   186    190   195   210
## 2 Chinstrap   178   191    196   201   212
## 3 Gentoo      203   212    216   221   231
```

```r
# Create box plot for flipper length with summary labels
ggplot(penguins, aes(x = species, y = flipper_len, fill = species)) +
  geom_boxplot(width = 0.6, alpha = 0.7, outlier.color = "red") +
  geom_text(
    data = flipper_stats,
    aes(x = species, y = Min, label = paste0("Min: ", round(Min, 1))),
    vjust = 1.5, color = "blue", size = 3.5
  ) +
  geom_text(
    data = flipper_stats,
    aes(x = species, y = Q1, label = paste0("Q1: ", round(Q1, 1))),
    vjust = -0.8, color = "darkgreen", size = 3.5
  ) +
  geom_text(
    data = flipper_stats,
    aes(x = species, y = Median, label = paste0("Q2: ", round(Median, 1))),
    vjust = -0.8, color = "purple", size = 3.5, fontface = "bold"
  ) +
  geom_text(
    data = flipper_stats,
    aes(x = species, y = Q3, label = paste0("Q3: ", round(Q3, 1))),
    vjust = -0.8, color = "orange", size = 3.5
  ) +
  geom_text(
    data = flipper_stats,
    aes(x = species, y = Max, label = paste0("Max: ", round(Max, 1))),
    vjust = -0.8, color = "red", size = 3.5
  ) +
  labs(
    title = "Comparative Box Plot of Penguin Flipper Length with Statistical Labels",
    x = "Penguin Species",
    y = "Flipper Length (mm)"
  )
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```



```
# The result: Each box plot now displays the five key statistics
# directly on the graph for each penguin species.
```

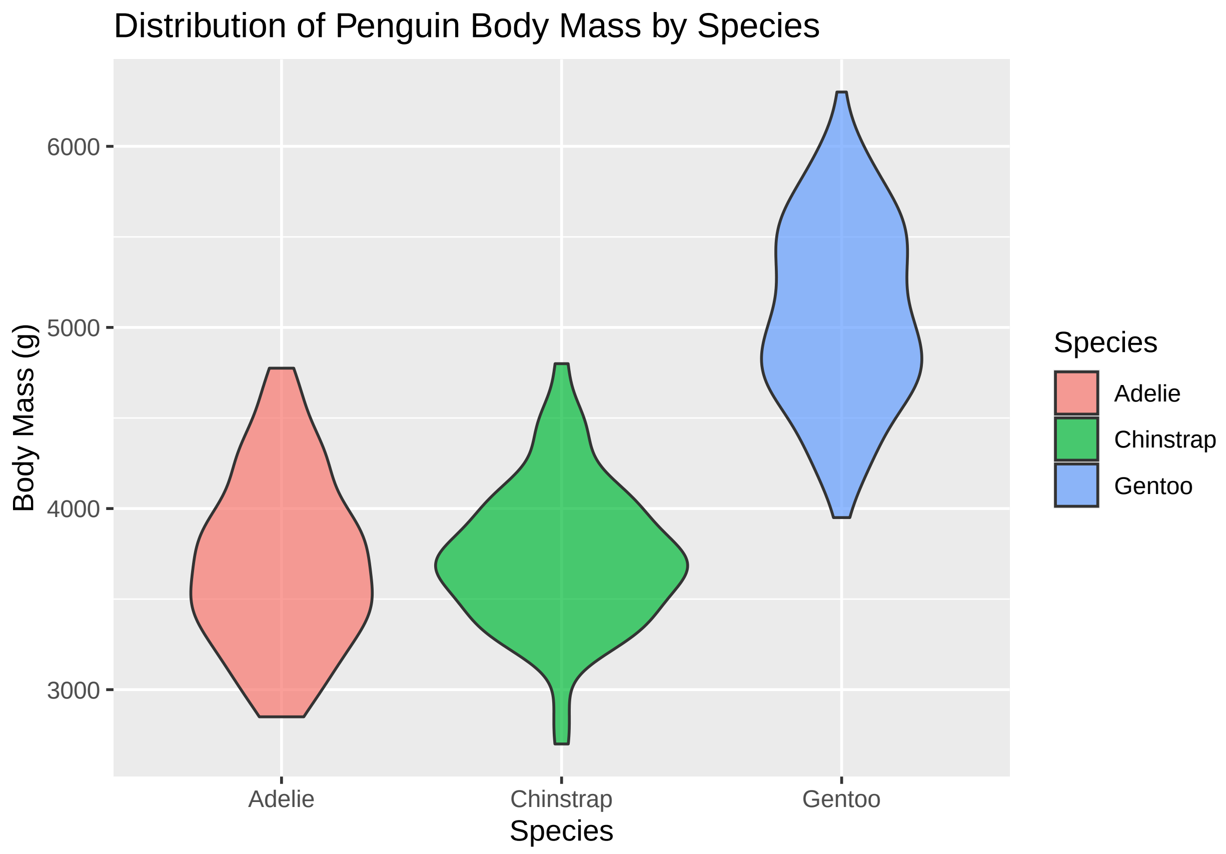
9.5 Violin Plots

Violin plots are a combination of box plots and kernel density plots. They display the distribution of a continuous variable and provide insights into the density of the data at different values. Violin plots are useful for visualizing the shape, spread, and central tendency of a dataset, as well as identifying potential multimodal distributions. We can create violin plots in ggplot2 using the `geom_violin` function. Here's how to do it:

(a). Creating a violin plot to show the distribution of penguin body mass by species

```r
ggplot(data = penguins) +
  geom_violin(mapping = aes(x = species, y = body_mass, fill = species), alpha = 0.7) +
  labs(title = "Distribution of Penguin Body Mass by Species",
       x = "Species",
       y = "Body Mass (g)",
       fill = "Species")
```

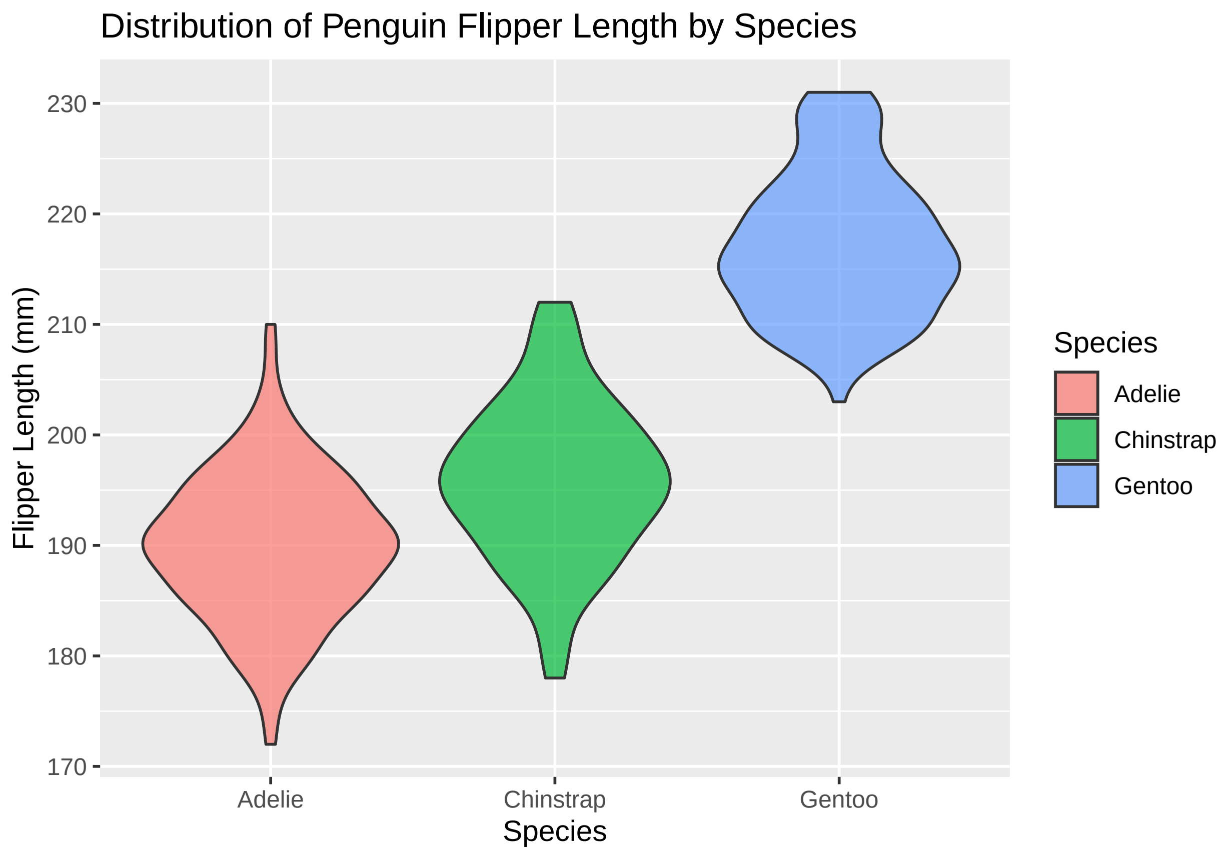
```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_ydensity()`).
```



(b). Creating a violin plot to show the distribution of penguin flipper length by species

```r
ggplot(data = penguins) +
  geom_violin(mapping = aes(x = species, y = flipper_len, fill = species), alpha = 0.7) +
  labs(title = "Distribution of Penguin Flipper Length by Species",
       x = "Species",
       y = "Flipper Length (mm)",
       fill = "Species")
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_ydensity()`).
```

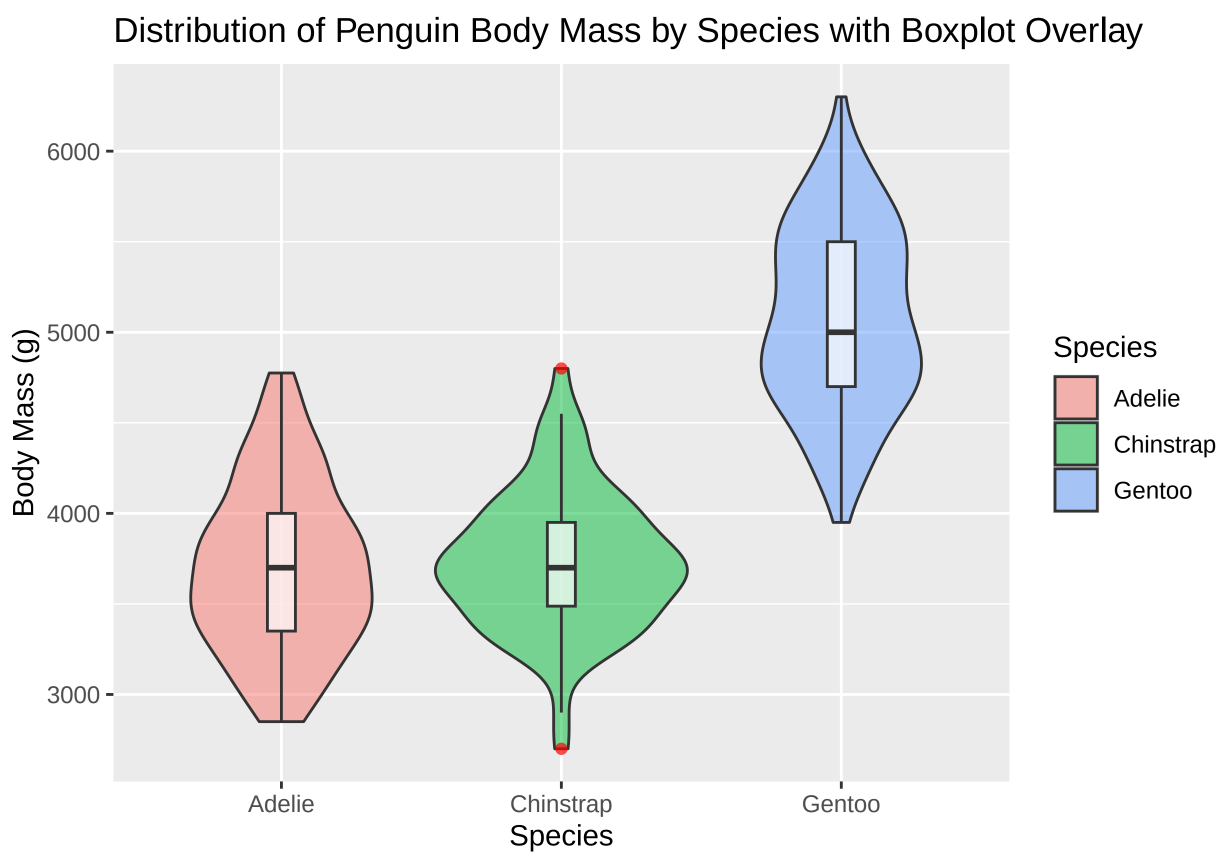


(c). Creating a violin plot to show the distribution of penguin body mass by species with boxplot overlay

```
ggplot(data = penguins) +
  geom_violin(mapping = aes(x = species, y = body_mass, fill = species), alpha = 0.5) +
  geom_boxplot(mapping = aes(x = species, y = body_mass),
  width = 0.1, outlier.color = "red", alpha = 0.7) +
  labs(title = "Distribution of Penguin Body Mass by Species with Boxplot Overlay",
       x = "Species",
       y = "Body Mass (g)",
       fill = "Species")
```

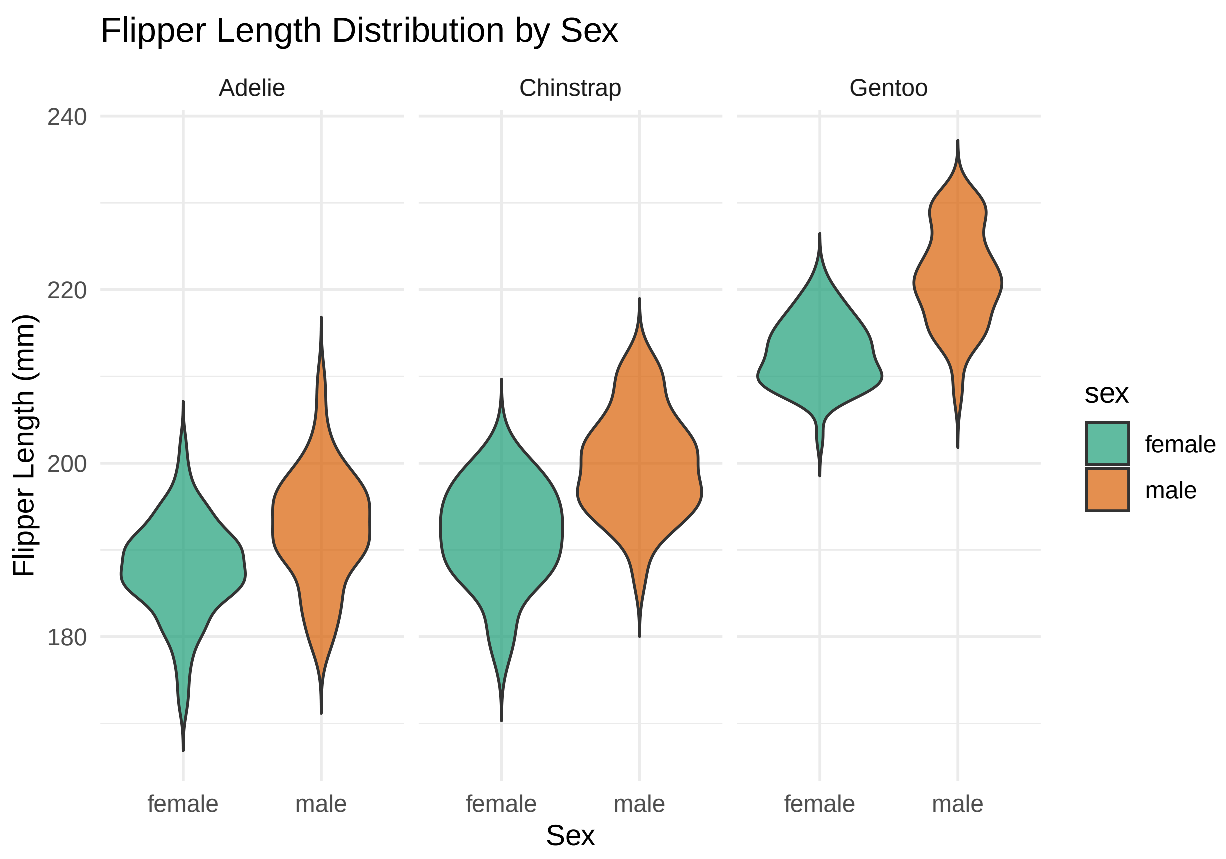
```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_ydensity()`).
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```



(d). Creating a violin plot to show flipper length distribution by sex

```
#------------------------------------------------------
# Flipper Length by Sex (Faceted by Species)
#------------------------------------------------------
# Violin plots show the full distribution of numeric data by category

# Remove rows where flipper_len and sex is NA(missing data)
penguins_clean <-penguins %>% filter(!is.na(flipper_len)) %>% filter(!is.na(sex))

ggplot(data = penguins_clean, aes(x = sex, y = flipper_len, fill = sex)) +   # Aesthetic mapping
  geom_violin(trim = FALSE, alpha = 0.7) +    # Draw violins, show full tails
  labs(title = "Flipper Length Distribution by Sex",      # Plot title
       x = "Sex",                                          # X-axis label
       y = "Flipper Length (mm)") +                        # Y-axis label
  theme_minimal() +                                      # Use minimal theme
  scale_fill_brewer(palette = "Dark2") +             # Dark contrasting colors
  facet_wrap(~species)                           # Create one plot per species
```



EXERCISE

Research and create plots for other geometries that we have not covered here, such as:

- **geom_density**: geom_density is used to create density plots, which show the distribution of a continuous variable. Density plots are useful for visualizing the shape of the data and identifying patterns or trends.
- **geom_area**: geom_area is used to create area plots, which show the cumulative values of a variable over time. Area plots are useful for visualizing trends and changes in data over time.
- **geom_path**: geom_path is used to create path plots, which connect data points in a specific order. Path plots are useful for visualizing the trajectory of a variable over time or space.
- **geom_step**: geom_step is used to create step plots, which show changes in a variable over time. Step plots are useful for visualizing discrete changes in data.
- **geom_segment**: geom_segment is used to create segment plots, which show relationships between two variables. Segment plots are useful for visualizing connections or correlations between variables.
- **geom_curve**: geom_curve is used to create curved lines that connect data points. Curved lines can be useful for visualizing relationships or trends in data.
- **geom_polygon** : geom_polygon is used to create polygon plots, which show areas defined by multiple points. Polygon plots are useful for visualizing regions or shapes in data.