Intro to ggplot2

Justin Baumann

Table of contents

1 Tutorials and Resources for graphs in ggplot

Basics of ggplot

Colors with ggsci

Many plots, 1 page w/ Patchwork

---

2 1: What makes a good graph vs a bad graph?

Take a look at some graphs of data for your field of interest. You may have a look at papers you have recently read or graphs you find in textbooks or assignments. Consider what you like or don't like about these graphs. What looks good and/or makes a graph easy to interpret? What doesn't? Making figures is both an art and a science.

To learn more about what makes graphs good (or bad), read Chapter 1 of Kieran Healy's online data visualization book –> What makes figures bad?

To continue your learning, have a look at this more detailed data visualization book by Claus Wilke Fundamentals of Data Visualization

2: Let's make some graphs!

Making nice looking graphs is a key feature of R and of data science in general. The best way to do this in R is through use of the ggplot2 package. This package is the most user friendly and flexible way to make nice plots in R. Notably, ggplot2 is a package that is contained within the tidyverse package, which is more of a style of R usage than a package. So, let's load tidyverse and a few other useful packages for today.

```
#Load packages
library(tidyverse)
library(ggsci) #for easy color scales
library(patchwork) #to make multi-panel plots
library(palmerpenguins) # our fave penguin friends :)
```

3 3: ggplot basics

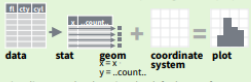
3.1 Introduction

ggplot2 is the preferred graphics package for most R users. It allows users to build a graph piece by piece from your own data through mapping of aesthetics. It is much easier to make pretty (publication and presentation quality) plots with ggplot2 than it is with the base plot function in R. If you prefer base plot() that is ok. You can use whatever you'd like but when we talk about graphs we will be using the language of ggplot.

Attached here are the Tidyverse Cheat Sheets for ggplot2

Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geommappings

i + stat_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() **x, y, size** | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun.y = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

n <- d + geom_bar(aes(fill = fl))

scale_ | aesthetic to adjust | prepackaged scale to use | scale-specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

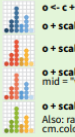
Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)

n <- d + geom_bar(aes(fill = fl))
n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()
n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)



o <- c + geom_dotplot(aes(fill = ..x..))
o + scale_fill_distiller(palette = "Blues")
o + scale_fill_gradient(low="red", high="yellow")
o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)
o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(xtrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0))projection, orienztation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))
s + geom_bar(position = "dodge")
Arrange elements side by side
s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height
e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting
e + geom_label(position = "nudge")
Nudge labels away from points
s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines
r + theme_gray()
Grey background (default theme)
r + theme_dark()
dark for contrast
r + theme_classic()
r + theme_light()
r + theme_linedraw()
r + theme_minimal()
Minimal themes
r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(cols = vars(fl))
facet into columns based on fl

t + facet_grid(rows = vars(year))
facet into rows based on year

t + facet_grid(rows = vars(year), cols = vars(fl))
facet into both rows and columns

t + facet_wrap(vars(fl))
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(rows = vars(drv), cols = vars(fl), scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(cols = vars(fl), labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(rows = vars(fl), labeller = label_bquote(alpha ^ .(fl)**))**

α^{c} α^{d} α^{e} α^{p} α^{r}

Labels

t + labs(**x** = "New x axis label", **y** = "New y axis label", **title** ="Add a title above the plot", **subtitle** = "Add a subtitle below title", **caption** = "Add a caption below plot", <AES> = "New <AES> legend title") **t + annotate**(geom = "text", x = 8, y = 9, label = "A")

Use scale functions to update legend labels

geom to place | manual values for geom's aesthetics

Legends

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"

n + guides(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

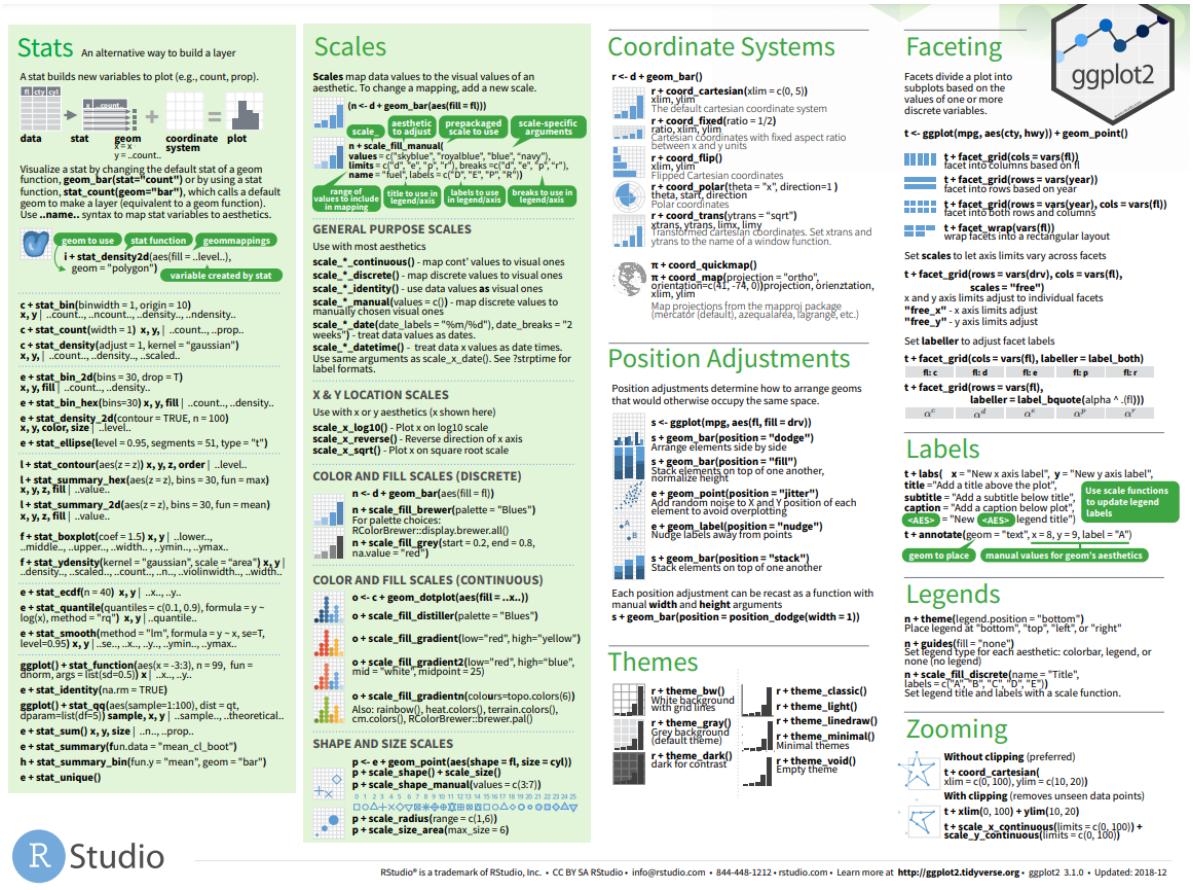
n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)
t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)
t + xlim(0, 100) **+ ylim**(10, 20)
t + scale_x_continuous(limits = c(0, 100)) **+ scale_y_continuous**(limits = c(0, 100))

RStudio



3.2 ggplot()

The ggplot() function is the base of the ggplot2 package. Using it creates the space that we use to build a graph. If we run just the ggplot() function we will get a gray rectangle. This is the space (and background) of our plot!

```
ggplot()
```

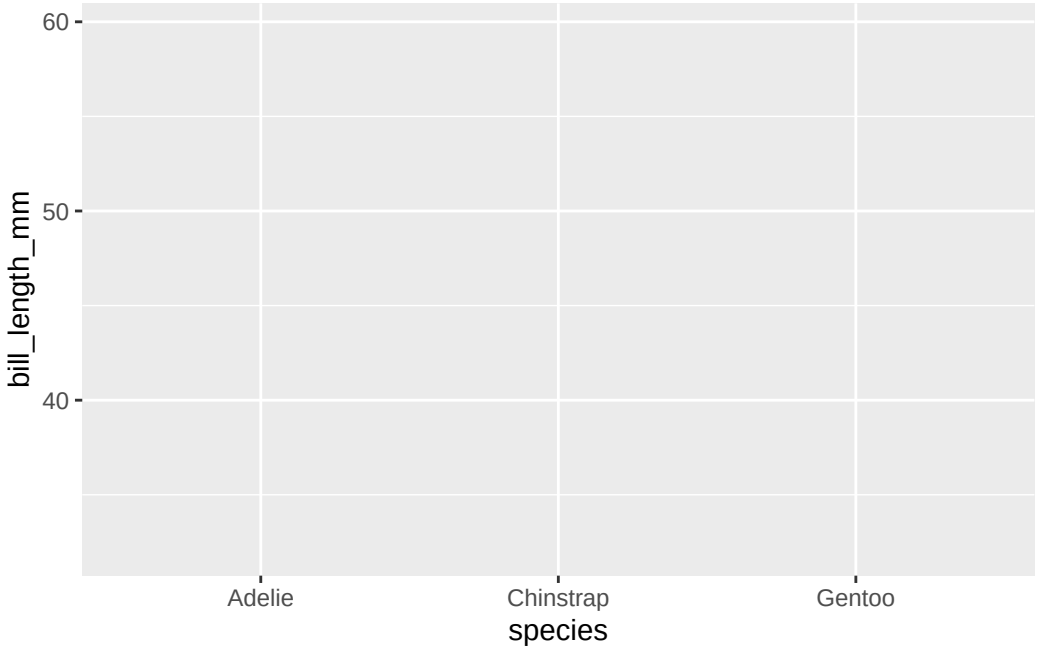
To build a plot on the background, we must add to the ggplot call. First, we need to tell it what data to use. Next, we need to tell it where in the data frame to pull data from to build the axes and data points. The part of the ggplot() function we use to build a graph is called aes() or aesthetics.

Here is an example using penguins: I am telling ggplot that the data we are using is 'penguins' and then defining the x and y axis in the aes() call with column names from penguins

```
head(penguins)
```

```
# A tibble: 6 x 8
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
1 Adelie  Torgersen           39.1          18.7               181        3750
2 Adelie  Torgersen           39.5          17.4               186        3800
3 Adelie  Torgersen           40.3          18                 195        3250
4 Adelie  Torgersen           NA            NA                  NA          NA
5 Adelie  Torgersen           36.7          19.3               193        3450
6 Adelie  Torgersen           39.3          20.6               190        3650
# i 2 more variables: sex <fct>, year <int>
```

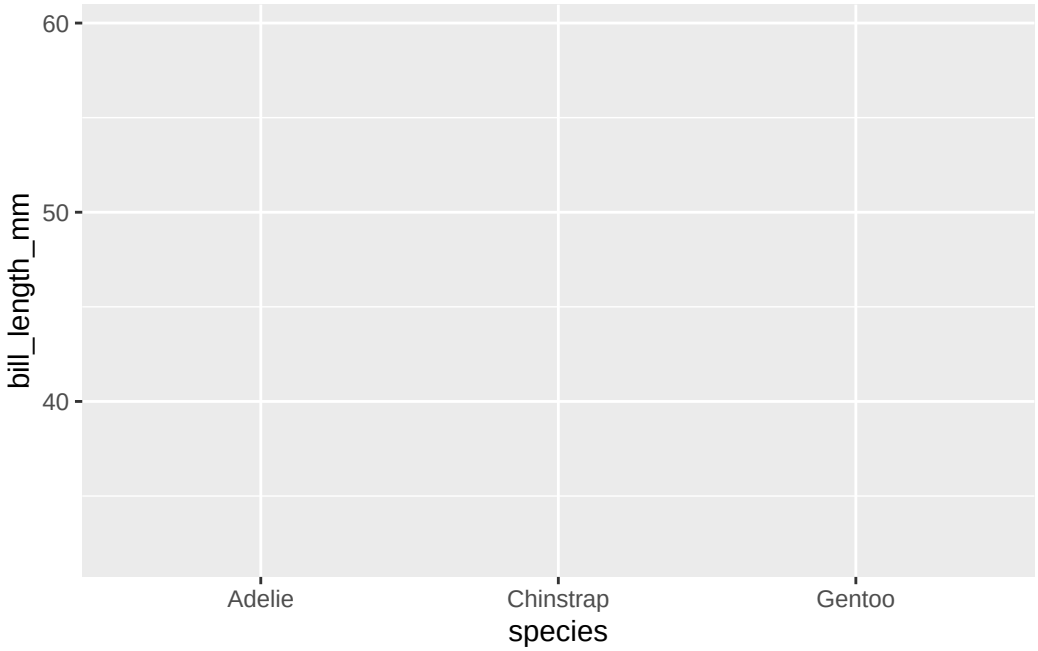
```
ggplot(data=penguins, aes(x=species, y= bill_length_mm))
```



Like anything in R, we can give our plot a name and call it later

```
plot1<-ggplot(data=penguins, aes(x=species, y= bill_length_mm))

plot1
```

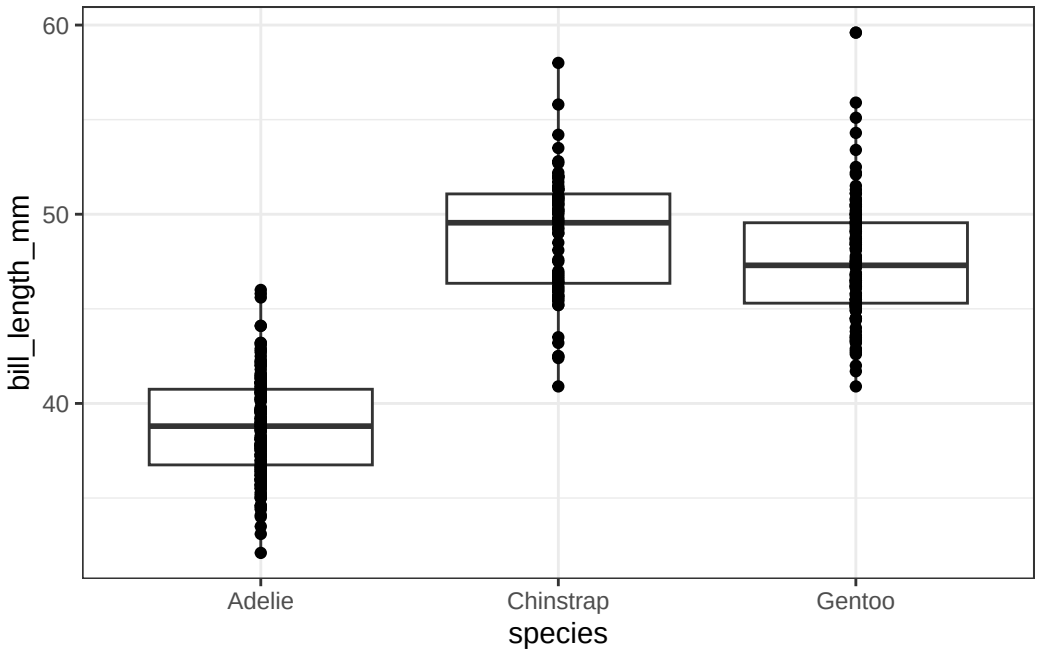


This is incredibly useful in ggplot as we can essentially add pieces to make a more complete graph

```
plot1+
  geom_boxplot()+
  geom_point()+
  theme_bw()
```

```
Warning: Removed 2 rows containing non-finite values (`stat_boxplot()`).
```

```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



Before we get too excited about making perfect graphs, let's take a look at the types of graphs we have available to us...

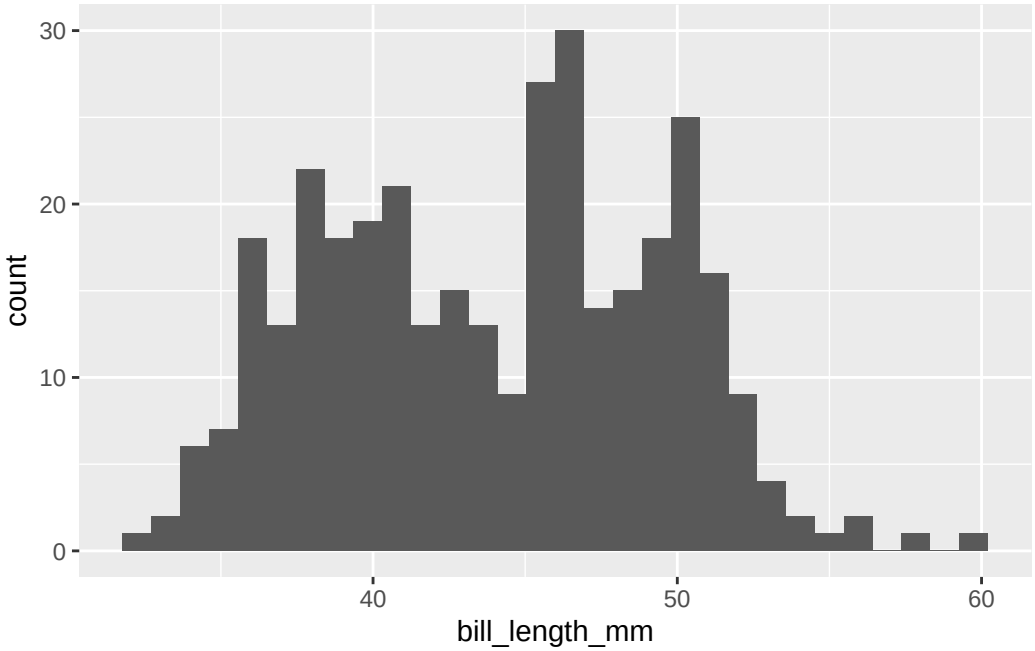
3.3 histogram

Histograms are used to explore the frequency distribution of a single variable. We can check for normality (a bell curve) using this feature. We can also look for means, skewed data, and other trends.

```
ggplot(data=penguins, aes(bill_length_mm))+
  geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

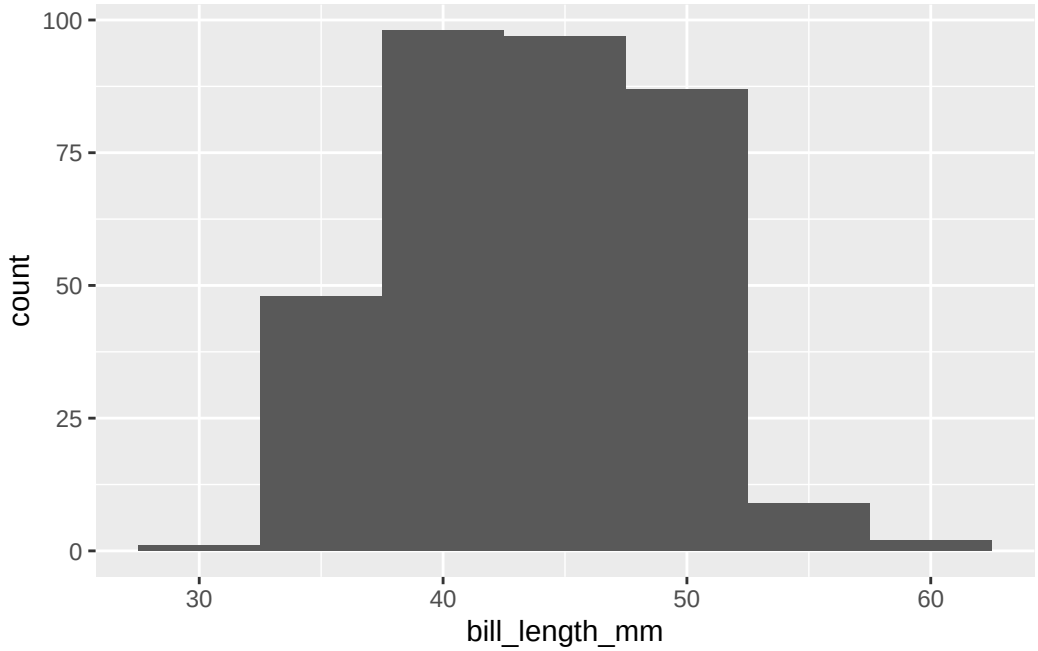
```
Warning: Removed 2 rows containing non-finite values (`stat_bin()`).
```



Within geom_histogram we can use bin_width to change the width of our x-axis groupings.

```
ggplot(data=penguins, aes(bill_length_mm))+
  geom_histogram(binwidth=5)
```

```
Warning: Removed 2 rows containing non-finite values (`stat_bin()`).
```



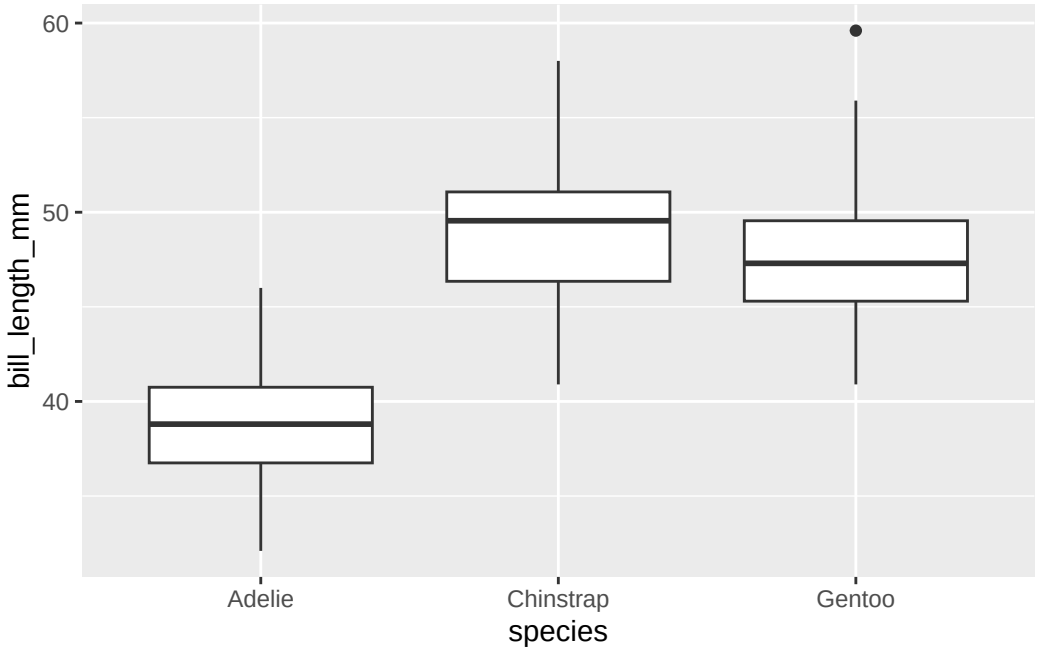
3.4 boxplot

A boxplot is a really useful plot to assess median and range of data. It can also identify outliers! The defaults for a boxplot in ggplot produce a median and interquartile range (IQR). The 1st quartile is the bottom of the box and the 3rd quartile is the top. The whiskers show the spread of the data where the ends of the whiskers represent the data points that are the furthest from the median in either direction. Notably, if a data point is 1.5 * IQR from the box (either the 1st or 3rd quartile) it is an outlier. Outliers are excluded from whiskers and are presented as points. There

Here's an example

```
ggplot(data=penguins, aes(x=species, y= bill_length_mm)) +
  geom_boxplot()
```

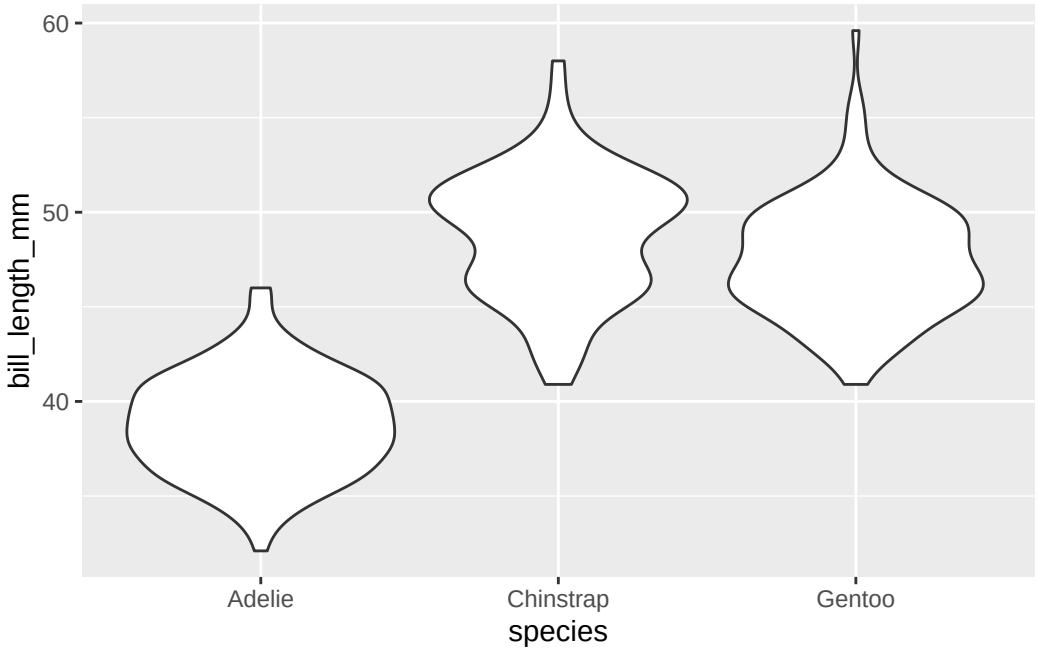
```
Warning: Removed 2 rows containing non-finite values (`stat_boxplot()`).
```



We can use geom_violin to combine boxplot with a density plot (similar to a histogram) Here we can see the distribution of values within bill length by species.

```
ggplot(data=penguins, aes(x=species, y= bill_length_mm)) +
  #geom_boxplot()+
  geom_violin()
```

```
Warning: Removed 2 rows containing non-finite values (`stat_ydensity()`).
```

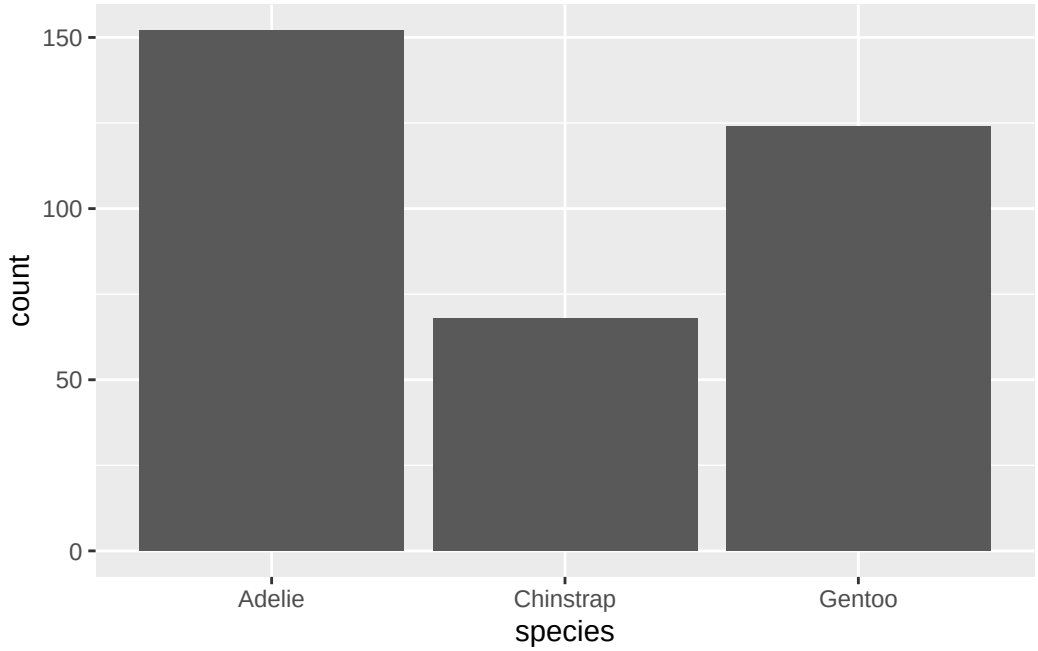


3.5 bar graph

We can make bar graphs in ggplot using geom_bar(). There are some tricks to getting bar graphs to work exactly right, which I will try to detail below. **NOTE** Bar graphs are very rarely useful. If we want to show means, why not just use points + error bars? What does the bar actually represent? There aren't that many cases where we really need bar graphs. There are exceptions, like when we have a population and we want to see the demographics of that population by count or percentage (see example below)

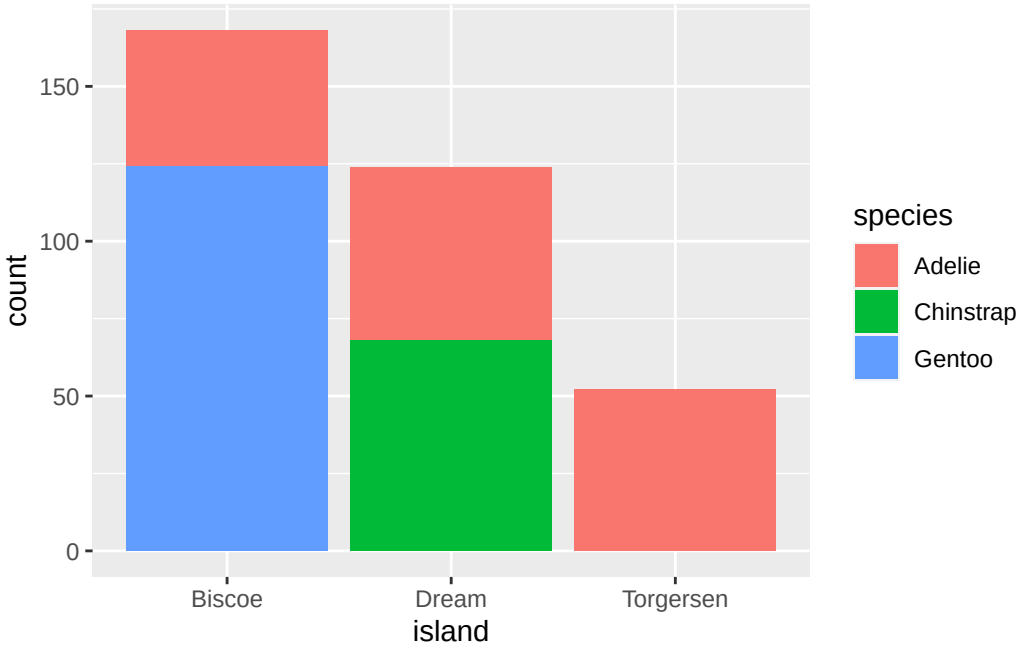
Here is a simple bar chart.

```
ggplot(data=penguins, aes(species)) +
  geom_bar()
```



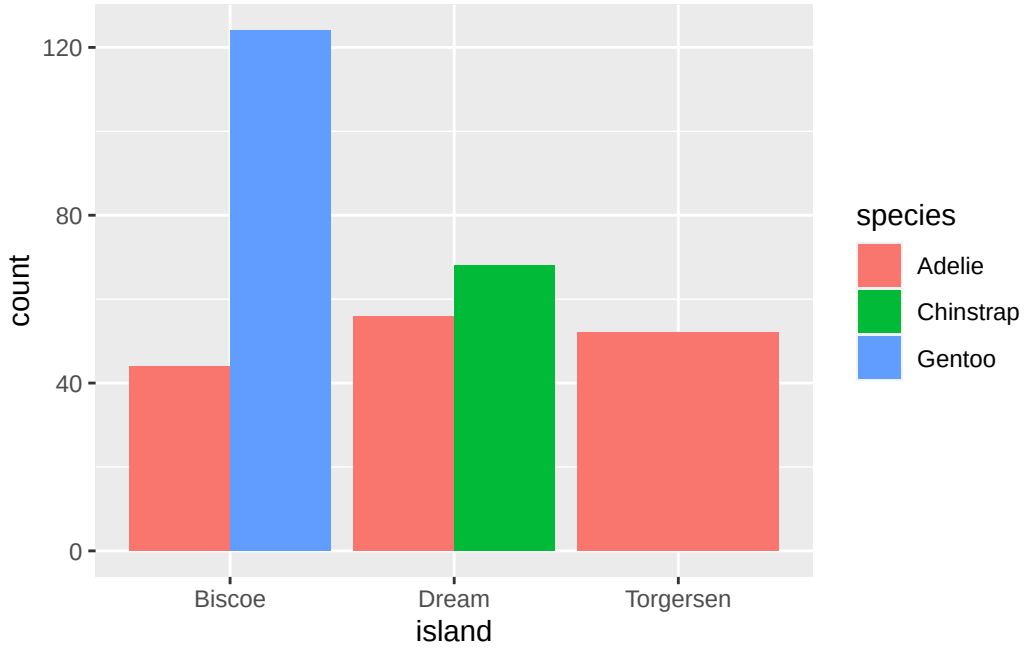
Here is a more elaborate boxplot that shows species breakdown by island! Note that we use an aes() call within geom_bar to define a fill. That means fill by species, or add a color for each species.

```
ggplot(data=penguins, aes(island)) +
  geom_bar(aes(fill=species))
```



And here is that same plot with the bars unstacked. Instead of stacking, we have used "dodged" each color to be its own bar.

```
ggplot(data=penguins, aes(island)) +
  geom_bar(aes(fill=species), position= position_dodge())
```



We learned when the best (only) times to use bar graphs are. Do you remember what those were? Are the examples above representative of that?

3.6 line graph

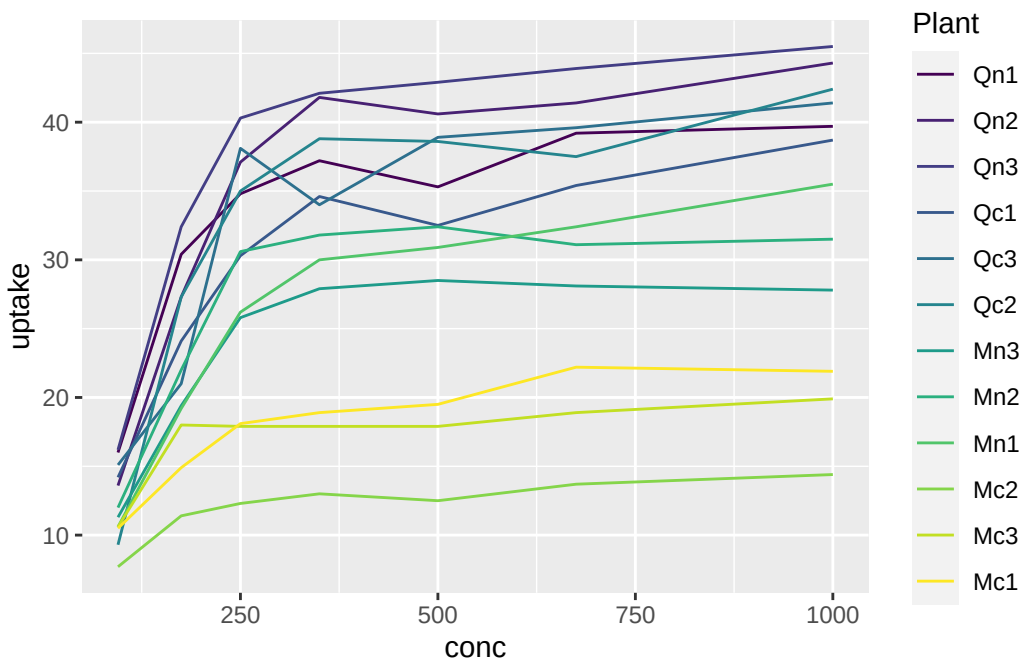
A line graph can be extremely useful, especially if we are looking at time series data or rates!

Here is an example of CO2 uptake vs concentration in plants. Each color represents a different plant. NOTE: the dataset called 'CO2' is built into R, so we can just use it without loading anything :)

```
head(CO2)
```

```
  Plant   Type  Treatment conc uptake
1   Qn1 Quebec nonchilled   95   16.0
2   Qn1 Quebec nonchilled  175   30.4
3   Qn1 Quebec nonchilled  250   34.8
4   Qn1 Quebec nonchilled  350   37.2
5   Qn1 Quebec nonchilled  500   35.3
6   Qn1 Quebec nonchilled  675   39.2
```

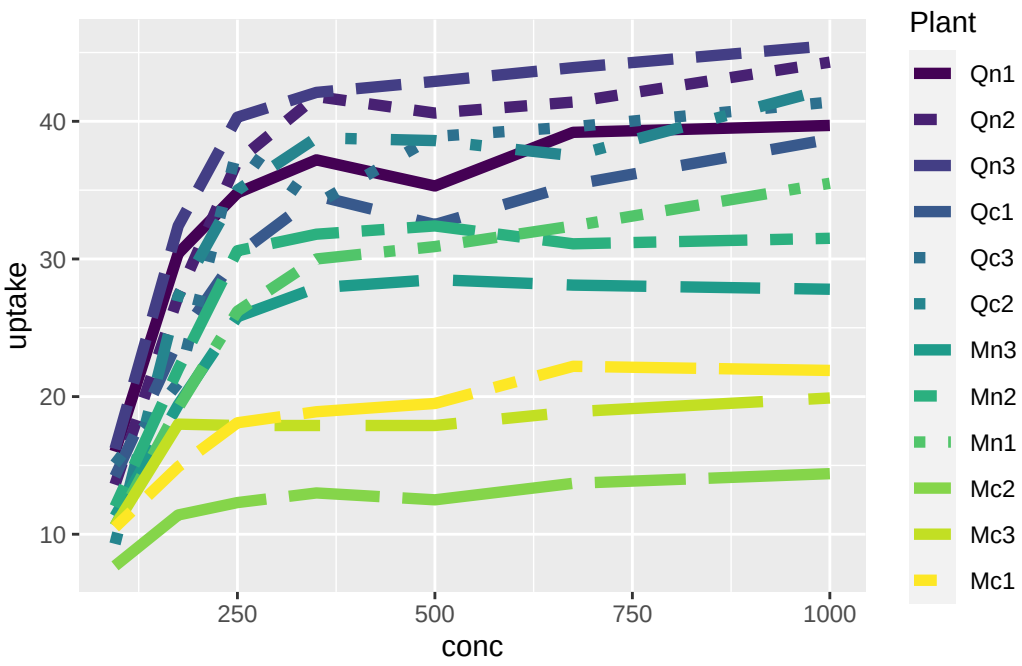
```
ggplot(data=CO2, aes(x=conc, y= uptake, color=Plant)) +
  geom_line()
```



We can change the aesthetics of the lines using color, linetype, size, etc. Here I am changing the linetype based on the plant species and increasing the size of ALL lines to 2. This is a good example of how aes() works. Anything within the aes() call is conditional. That means, I give it a name (such as a column or variable name) and it changes based on that column or variable. To change an aesthetic across all lines, points, etc, I just put the code outside of the aes(). As I did for size. That makes the size of ALL lines = 2.

```
ggplot(data=CO2, aes(x=conc, y= uptake, color=Plant)) +
  geom_line(aes(linetype=Plant),size=2)
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```

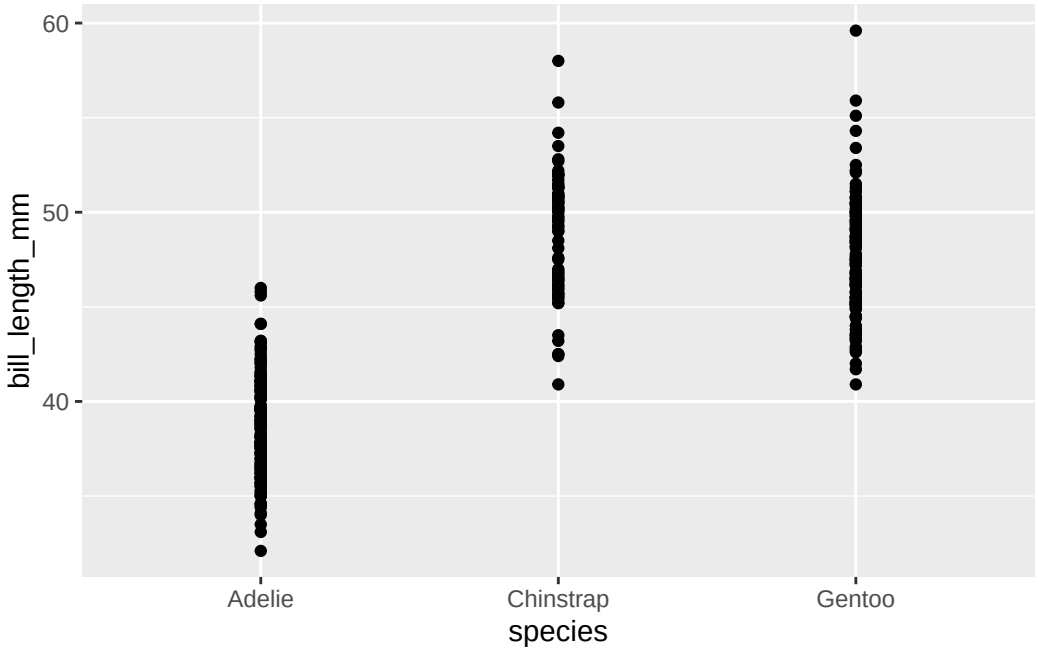


3.7 scatter plot

The scatter plot is probably the most commonly used graphical tool in ggplot. It is based on the geom_point() function

```
ggplot(data=penguins, aes(x=species, y= bill_length_mm)) +
  geom_point()
```

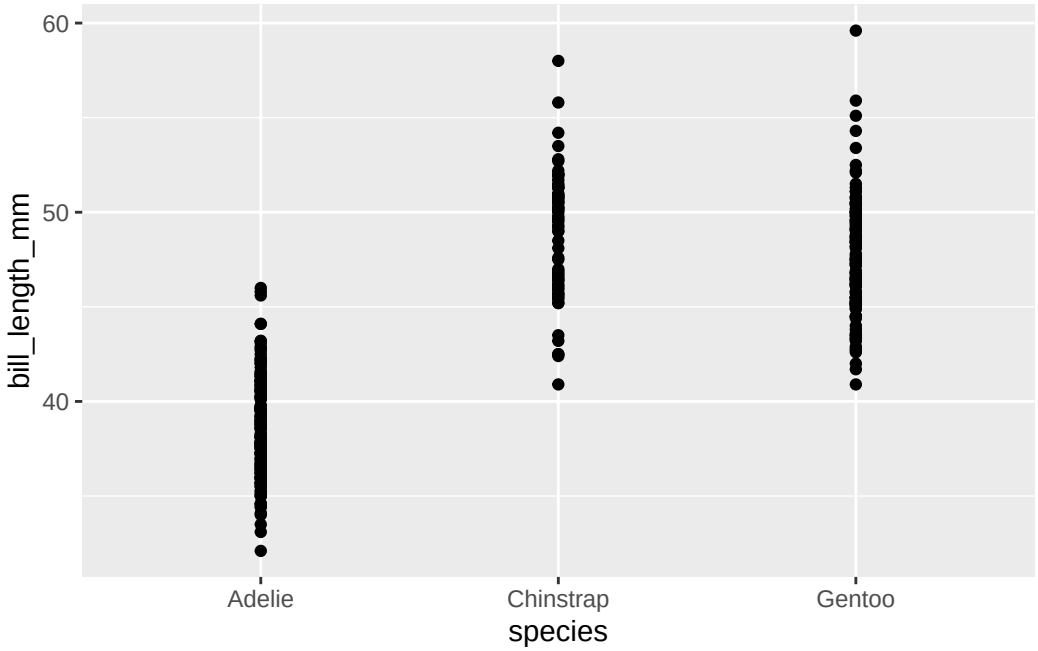
```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



Importantly, we can use the data= and aes() calls within geom_point() or any other geom instead of within ggplot() if needed. Why might this be important?

```
ggplot() +
  geom_point(data=penguins, aes(x=species, y= bill_length_mm))
```

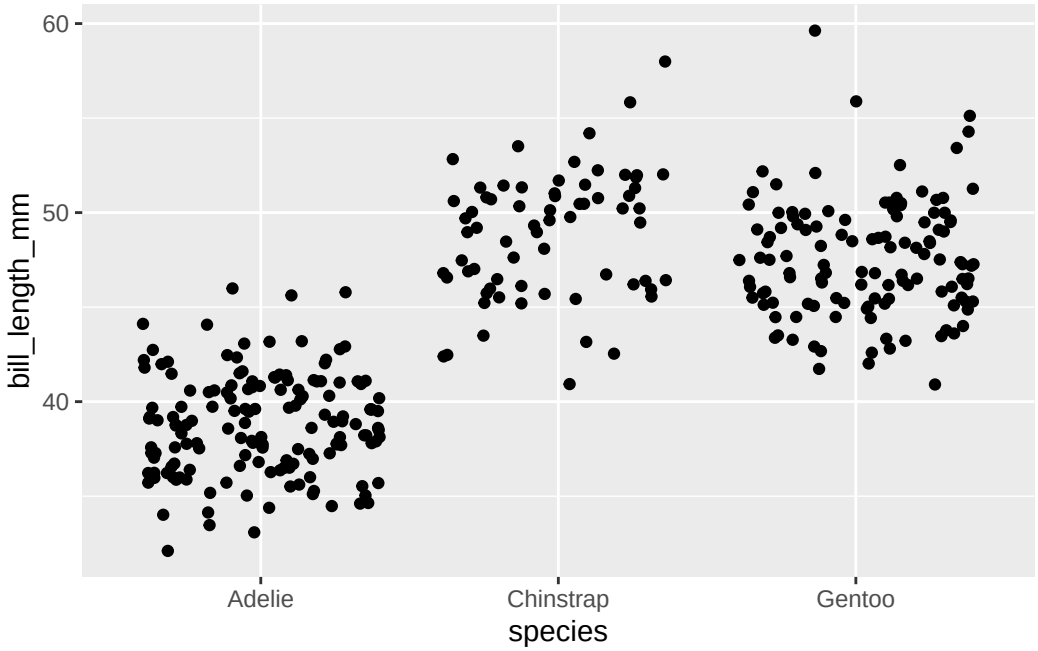
```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



Sometimes we don't want to plot all of our points on the same vertical line. If that is the case, we can use geom_jitter()

```
ggplot(data=penguins, aes(x=species, y= bill_length_mm)) +
  geom_jitter()
```

```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



3.8 Adding error bars

We often want to present means and error in our visualizations. This can be done through the use of geom_boxplot() or through combining geom_point() with geom_errorbar()

Here is an example of the later…

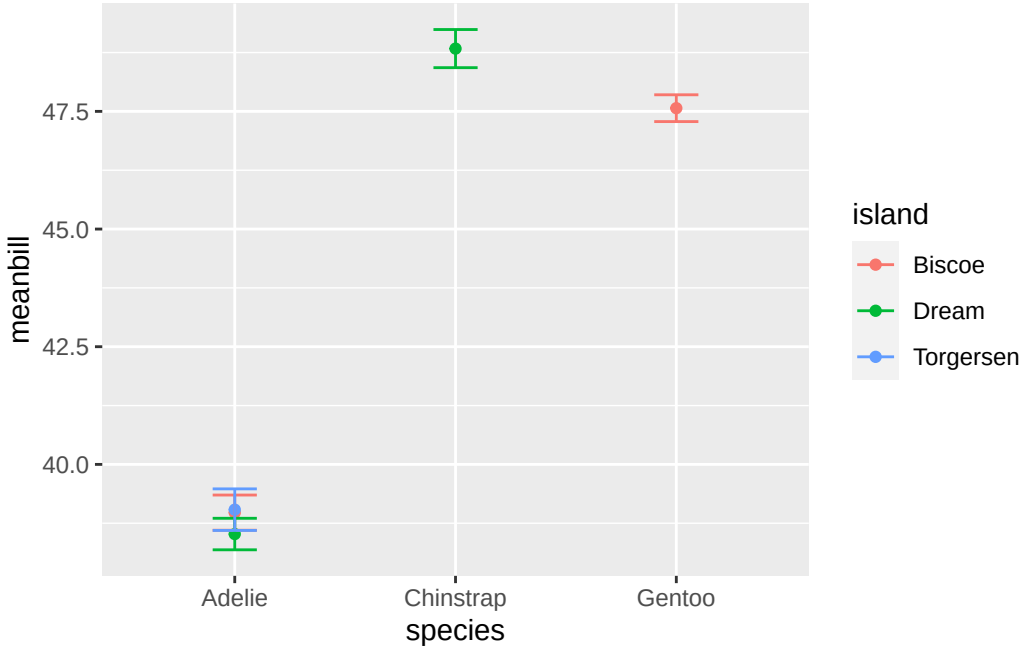
```
#First, we need to calculate a mean bill length for our penguins by species and island
sumpens<- penguins %>%
  group_by(species, island) %>%
  na.omit() %>% #removes rows with NA values (a few rows may otherwise have NA due to samp
  summarize(meanbill=mean(bill_length_mm), sd=sd(bill_length_mm), n=n(), se=sd/sqrt(n))

sumpens
```

```
# A tibble: 5 x 6
# Groups:   species [3]
  species   island    meanbill    sd     n    se
  <fct>     <fct>        <dbl> <dbl> <int> <dbl>
1 Adelie    Biscoe        39.0  2.48    44 0.374
```

```
2 Adelie    Dream         38.5  2.48    55 0.335
3 Adelie    Torgersen     39.0  3.03    47 0.442
4 Chinstrap Dream         48.8  3.34    68 0.405
5 Gentoo    Biscoe        47.6  3.11   119 0.285
```

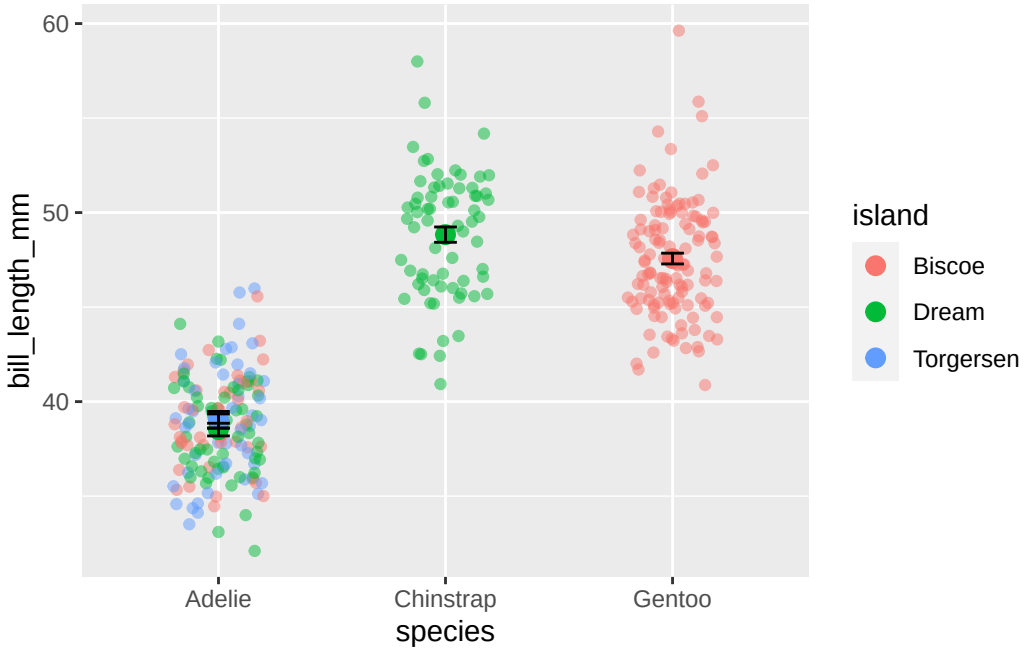
```
# Now we can plot!
ggplot(data=sumpens, aes(x=species, y=meanbill, color=island))+
  geom_point()+
  geom_errorbar(data=sumpens, aes(x=species, ymin=meanbill-se, ymax=meanbill+se), width=0.
```



And if we want to be extra fancy (and rigorous), we can plot the raw data behind the mean+error This is considered a **graphical best practice** as we can see the mean, error, and the true spread of the data!

```
ggplot()+
  geom_jitter(data= penguins, aes(x=species, y=bill_length_mm, color=island), alpha=0.5, w
  geom_point(data=sumpens, aes(x=species, y=meanbill, color=island), size=3)+ #this is the
  geom_errorbar(data=sumpens, aes(x=species, ymin=meanbill-se, ymax=meanbill+se), width=0.
```

```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



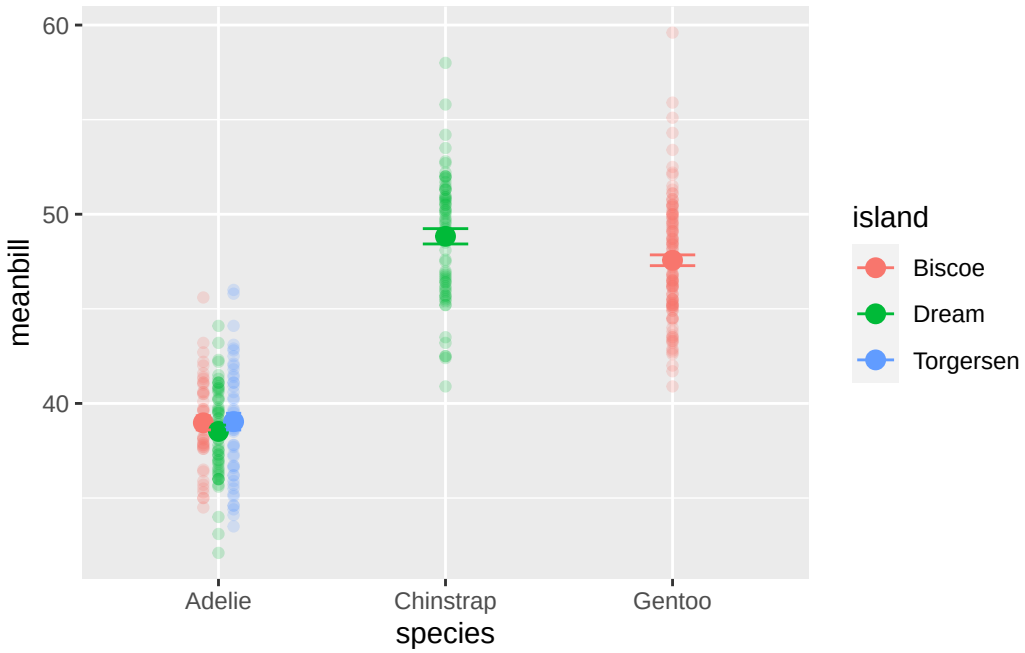
An alternative to geom_jitter, which doesn't always work, is to use geom_point but force the points to not overlap with position_dodge. Here is an example

```
#first we should define the distance of our position_dodge
pd<-position_dodge(width=0.2)

ggplot(data=sumpens, aes(x=species, y=meanbill, color=island))+
  geom_point(data= penguins, aes(x=species, y=bill_length_mm, color=island), alpha=0.2, wi
  geom_point(size=3, position=pd)+ #averages
  geom_errorbar(aes(ymin=meanbill-se, ymax=meanbill+se), width=0.2, position=pd)
```

```
Warning in geom_point(data = penguins, aes(x = species, y = bill_length_mm, :
Ignoring unknown parameters: `width`
```

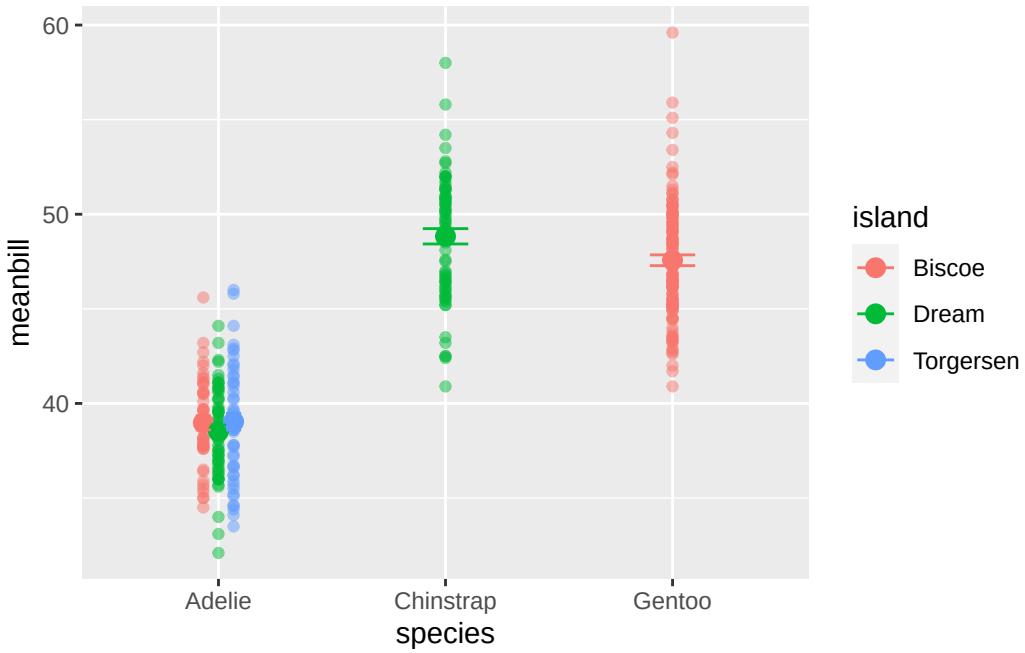
```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



This code will produce the same graph as above. Note that in geom_jitter we just replaced width = with position =

```
ggplot(sumpens, aes(x=species, y= meanbill, color=island))+
  geom_jitter(data= penguins, aes(x=species, y=bill_length_mm, color=island), alpha=0.5, p
  geom_point(size=3,position=pd)+ #this is the averages
  geom_errorbar(aes(ymin=meanbill-se, ymax=meanbill+se), width=0.2, position=pd)
```

```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```


60
50
40
meanbill
Adelie
Chinstrap
Gentoo
species
island
Biscoe
Dream
Torgersen