Data Visualization using R

Week 4: Working with ggplot graphs

Categorical variables

What are categorical variables?

Categorical variables are variables that

- have values defining categories of things
- typically have a few unique values
- may or may not be ordered
- are not interval-scaled, i.e., their differences don't make sense *per se*

What are categorical variables?

Non-ordered

1. Race (White, Black, Hispanic, Asian, Native American)
2. Gender (Male, Female, Other)
3. Geographic regions (Africa, Asia, Europe, North America, South America)
4. Genes/Proteins

Ordered

1. Income levels (< $10K, $10K - $25K, $25K - $75K, $75K - $100K)
2. BMI categories (Underweight, Normal, Overweight, Obese)
3. Number of bedrooms in houses (1 BR, 2BR, 3BR, 4BR)

Cateogical variables in R

The `factor` data type

R stores categorical variables as type `factor`.

- You can coerce a character or numeric object into a factor using `as.factor`.
- You can check if an object is a factor with `is.factor`.
- You can create a factor with the function `factor`.

The `factor` data type

> `factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA)`
>
> factor returns an object of class "factor" which has a set of integer codes the length of x with a "levels" attribute of mode character and unique

---

- Internally, each level of a factor is coded as an integer
- Each such integer has a corresponding `level` which is a character, describing the level.
- You can add `labels` to each level to change the printed form of the factor.

The `factor` data type

```
x <- c('Maryland','Virginia', 'District', 'Maryland','Virginia')  # a character vector
xf <- as.factor(x)
xf
```

```
[1] Maryland Virginia District Maryland Virginia
Levels: District Maryland Virginia
```

There are three levels, that by default are in alphabetical order

---

```
as.integer(xf)
```

```
[1] 2 3 1 2 3
```

- District = 1, Maryland = 2, Virginia = 3

```
as.character(xf)
```

```
[1] "Maryland" "Virginia" "District" "Maryland" "Vi
```

- Get original characters back

The `factor` data type

```
y <- c(5, 3, 9, 4, 5, 3)
yf <- as.factor(y)
yf
```

```
[1] 5 3 9 4 5 3
Levels: 3 4 5 9
```

Levels are still in alphanumeric order

```
as.numeric(yf)
```

```
[1] 3 1 4 2 3 1
```

- Note, we don't get original integers back!!
- 3 = 1, 4 = 2, 5 = 3, 9 = 4

```
as.numeric(as.character(yf))
```

```
[1] 5 3 9 4 5 3
```

- This is how you get numbers back

The `factor` data type

```
x <- c('MD','DC','VA','MD','DC')
xf <- factor(x)
unclass(xf)
```

```
[1] 2 1 3 2 1
attr(,"levels")
[1] "DC" "MD" "VA"
```

```
x <- c('MD','DC','VA','MD','DC')
xf <- factor(x, levels = c('MD','DC','VA'))
unclass(xf)
```

```
[1] 1 2 3 1 2
attr(,"levels")
[1] "MD" "DC" "VA"
```

- If I change the level designation, the underlying coding changes
- This is important when a factor is an independent variable in a regression model

The `factor` data type

The `drv` variable in the `mpg` dataset tells us the kind of drive (front, rear or 4-wheel) each car has. However it's coded as `f`, `r`, and `4`, which is not great for display purposes. We can re-label these levels, but we have to be a bit careful

```
x <- mpg$drv
xf <- factor(x,
             levels = c('4-wheel','Front wheel',
                        'Rear wheel'))
head(xf)
```

```
  [1] <NA> <NA> <NA> <NA> <NA> <NA>
  Levels: 4-wheel Front wheel Rear wheel
```

```
x <- mpg$drv
xf <- factor(x,
             levels = c('4', 'f','r'),
             labels = c('4-wheel','Front wheel',
                        'Rear wheel'))
head(xf)
```

```
  [1] Front wheel Front wheel Front wheel Front wheel
  Levels: 4-wheel Front wheel Rear wheel
```

Levels have to match what's actually in the original data, but you can re-label the levels.

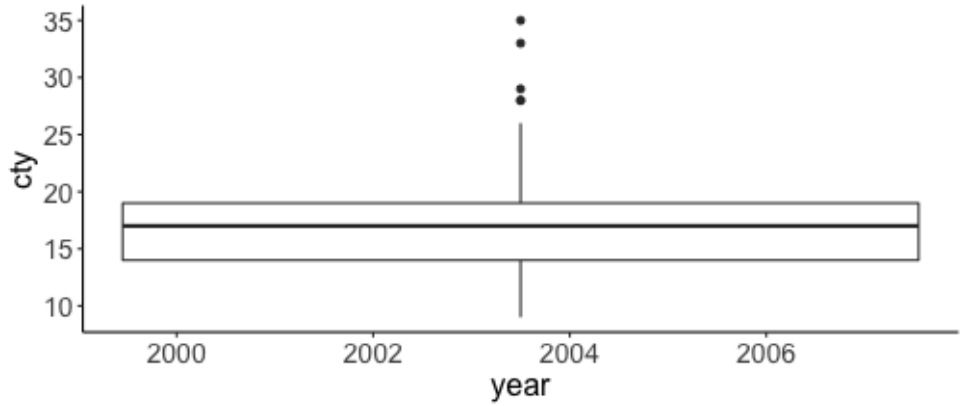
Why factors?

Factors are R's discrete data type

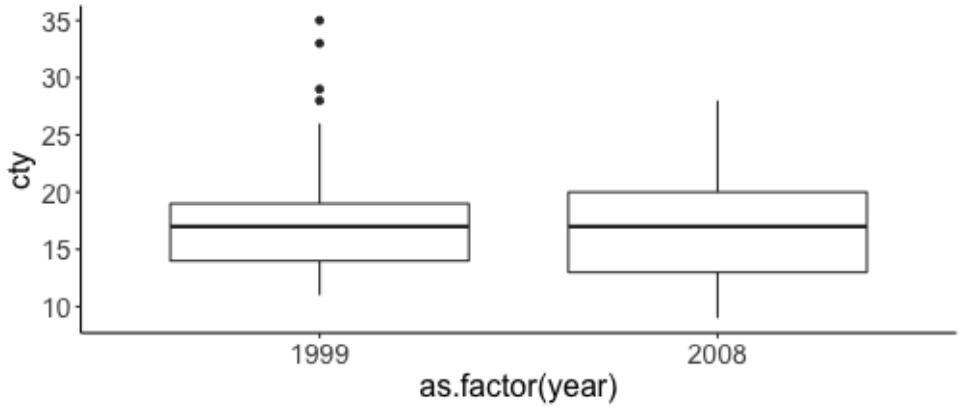
- Factors are interpreted as discrete by R's functions

```
ggplot(mpg,
       aes(year, cty))+
  geom_boxplot()
```

```
  Warning: Continuous x aesthetic -- did you forget a
```




```
ggplot(mpg,
       aes(as.factor(year), cty))+
  geom_boxplot()
```



Dummy variables are automatically created from factors

```
model.matrix(~species, data = palmerpenguins::penguin
```

```
  (Intercept) speciesChinstrap speciesGentoo
1           1                0             0
2           1                0             0
3           1                1             0
4           1                1             0
5           1                0             1
6           1                0             1
attr(,"assign")
[1] 0 1 1
attr(,"contrasts")
attr(,"contrasts")$species
[1] "contr.treatment"
```

- If a factor has *n* levels, you get *n-1* dummy variables
- The level corresponding to integer code 1 is omitted as the reference level

---

Changing the base level (integer code 1) changes model interpretation since it changes the reference level against which all other levels are compared.

Manipulating factors

The `forcats` package (part of `tidyverse`)

Effect in models

```
library(palmerpenguins)
m <- lm(body_mass_g ~ species, data = penguins)
broom::tidy(m)
```

```
# A tibble: 3 x 5
  term             estimate std.error statistic   p
  <chr>               <dbl>     <dbl>     <dbl>
1 (Intercept)        3701.       37.6    98.4   2.4
2 speciesChinstrap     32.4      67.5     0.480 6.3
3 speciesGentoo      1375.       56.1    24.5   5.4
```

Compare with Adele

```
p1 <- penguins %>%
  mutate(species = fct_relevel(species, 'Gentoo'))
m1 <- lm(body_mass_g ~ species, data=p1 )
broom::tidy(m1)
```

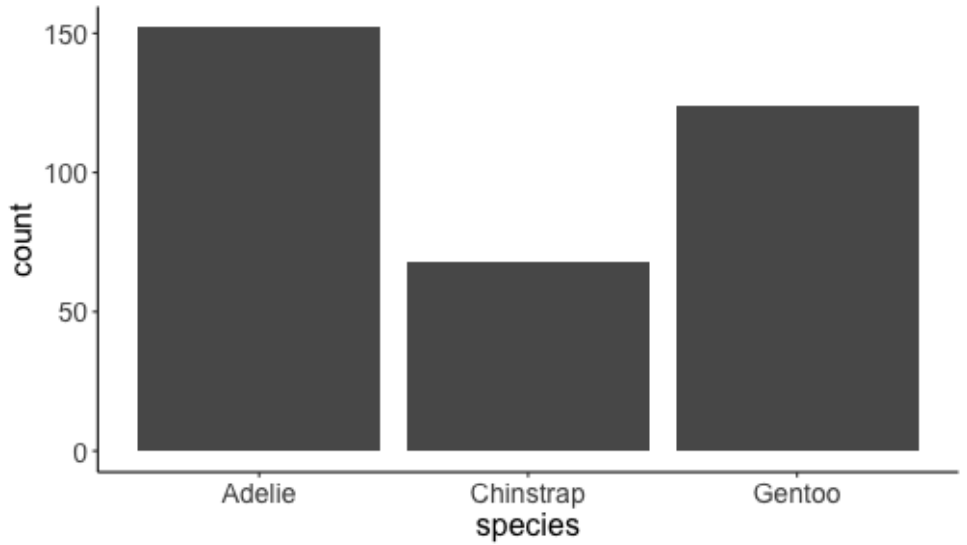
```
# A tibble: 3 x 5
  term             estimate std.error statistic   p
  <chr>               <dbl>     <dbl>     <dbl>
1 (Intercept)        5076.       41.7     122.  6.8
2 speciesAdelie     -1375.       56.1     -24.5 5.4
3 speciesChinstrap  -1343.       69.9     -19.2 3.2
```

Compare with Gentoo

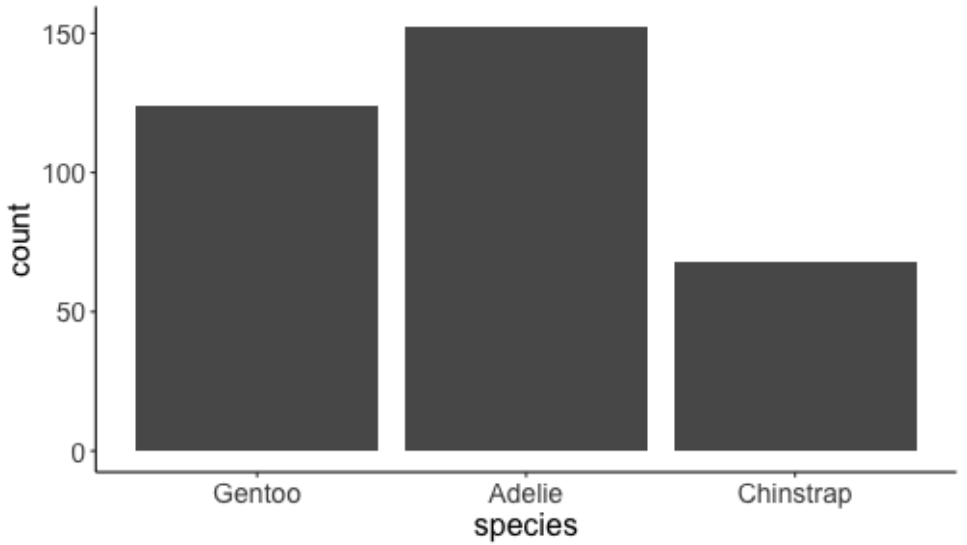
Providing only one level to `fct_relevel` makes that the base level (integer code 1).
You can also fully specify all the levels in order, or partially specify them. If you partially specify them, the remaining levels will be put in alphabetical order after the ones you specify.

Effect in plots

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




```
ggplot(p1,
       aes(x = species))+
  geom_bar()
```



Changes the order in which bars are plotted

Extra levels

```
x <- factor(str_split('statistics', '')[[1]],
            levels = letters)
x
```

```
 [1] s t a t i s t i c s
Levels: a b c d e f g h i j k l m n o p q r s t u v
```

```
p1 <- penguins %>% filter(species != 'Gentoo')
fct_count(p1$species)
```

```
# A tibble: 3 x 2
  f             n
  <fct>     <int>
1 Adelie      152
2 Chinstrap    68
3 Gentoo        0
```

```
fct_drop(x)
```

```
 [1] s t a t i s t i c s
Levels: a c i s t
```

```
p1 <- p1 %>% mutate(species = fct_drop(species))
fct_count(p1$species)
```

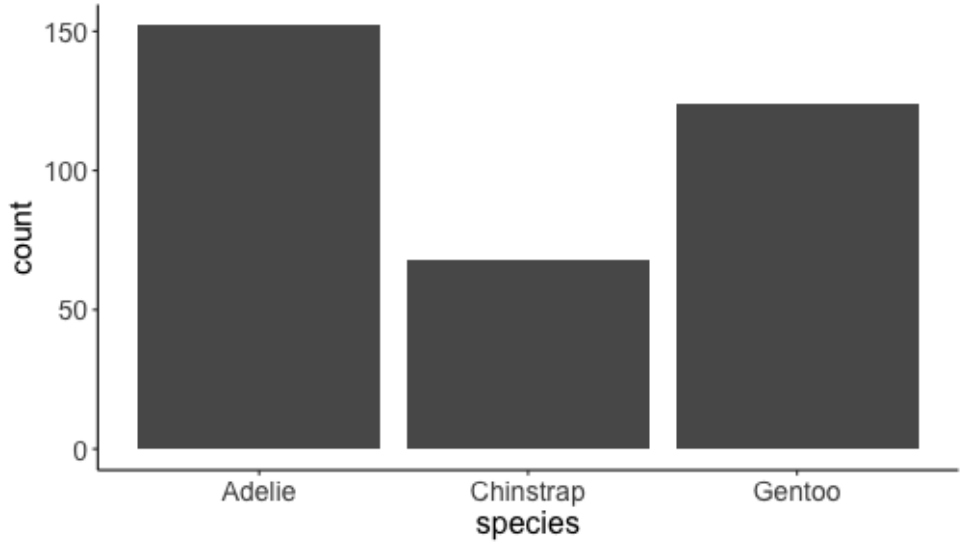
```
# A tibble: 2 x 2
  f             n
  <fct>     <int>
1 Adelie      152
2 Chinstrap    68
```

Getting rid of extra levels

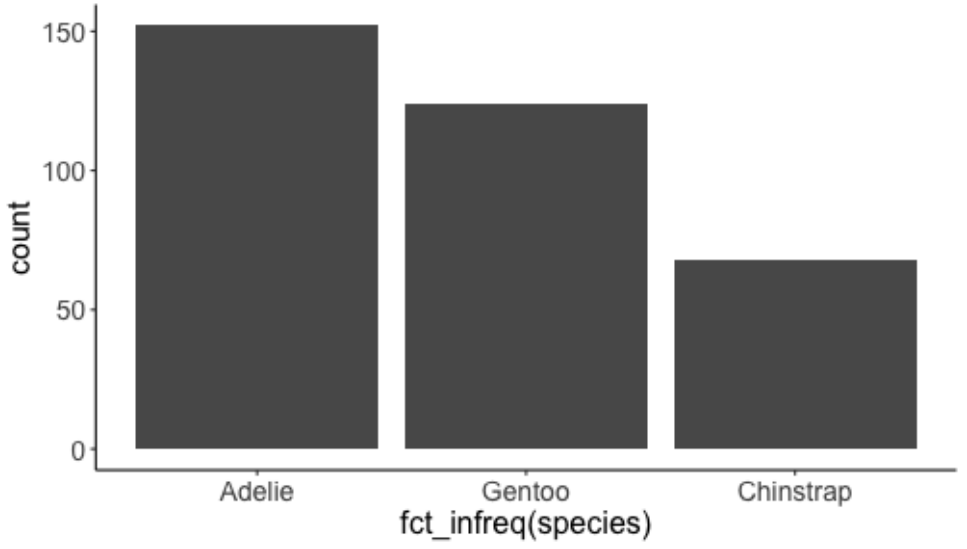
Sometimes levels with no data show up in summaries or plots

Ordering levels by frequency

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



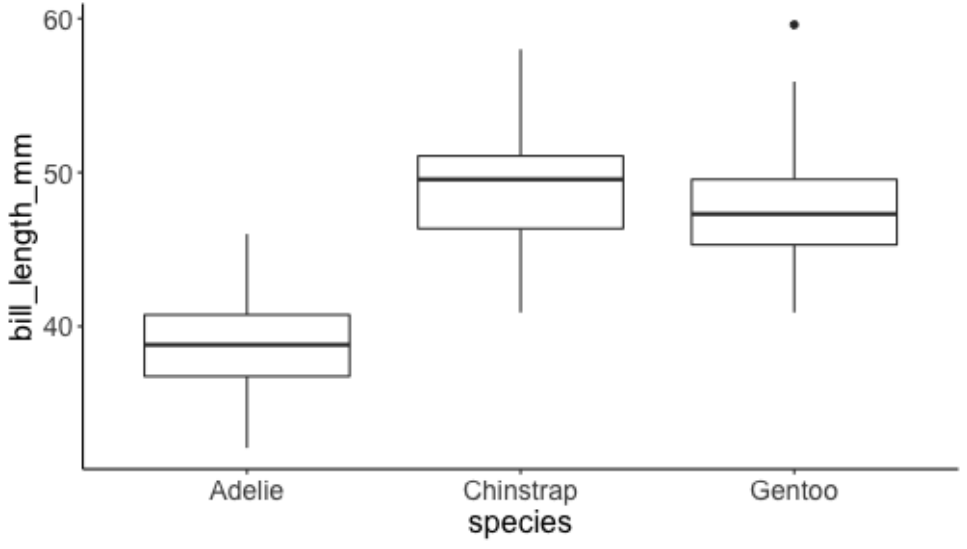

```
ggplot(penguins,
       aes(x = fct_infreq(species)))+
  geom_bar()
```



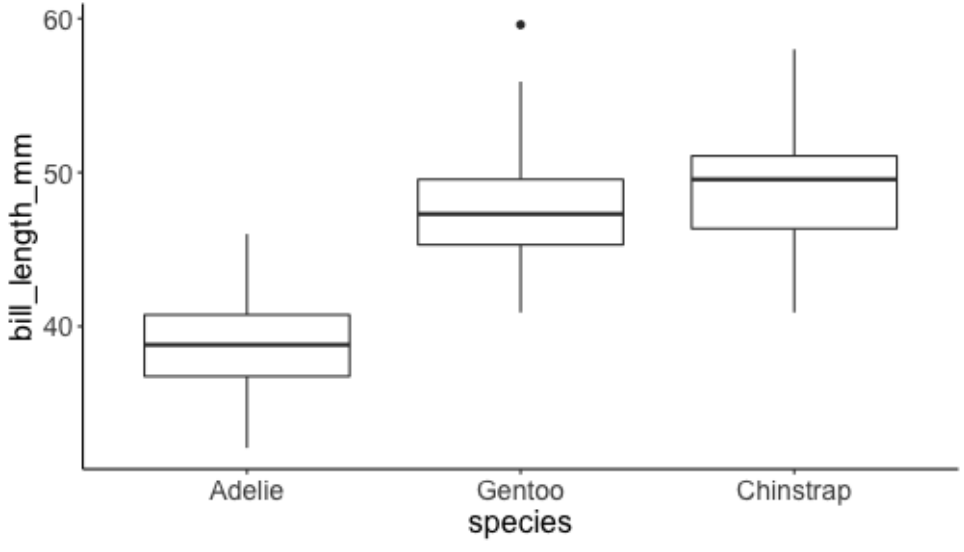
Ordering levels from most to least frequent

Ordering levels by values of another variable

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




```
ggplot(penguins,
       aes(x = fct_reorder(species, bill_length_mm,
                           .fun=median, na.rm=T),
           y = bill_length_mm))+
  geom_boxplot() + labs(x = 'species')
```

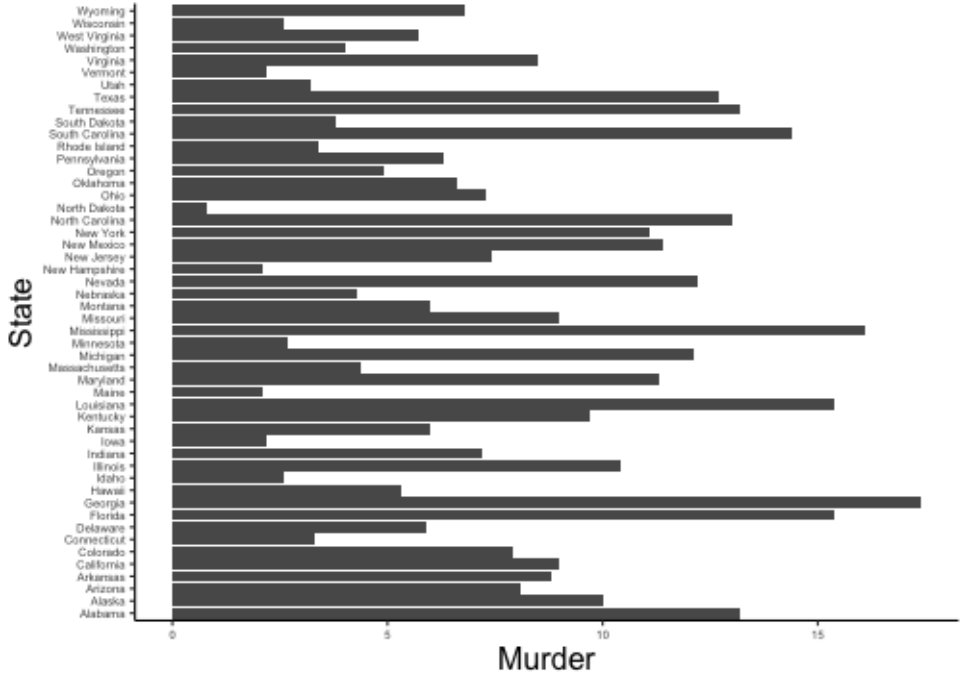


`fct_reorder` is useful for ordering a plot by ascending or descending levels. This makes the plot easier to read.

Ordering levels by values of another variable

```
USArrests <- USArrests %>% rownames_to_column('State')
```

```
ggplot(USArrests, aes(x=State, y = Murder))+
  geom_bar(stat = 'identity') +
  theme(axis.text = element_text(size=6))+
  coord_flip()
```

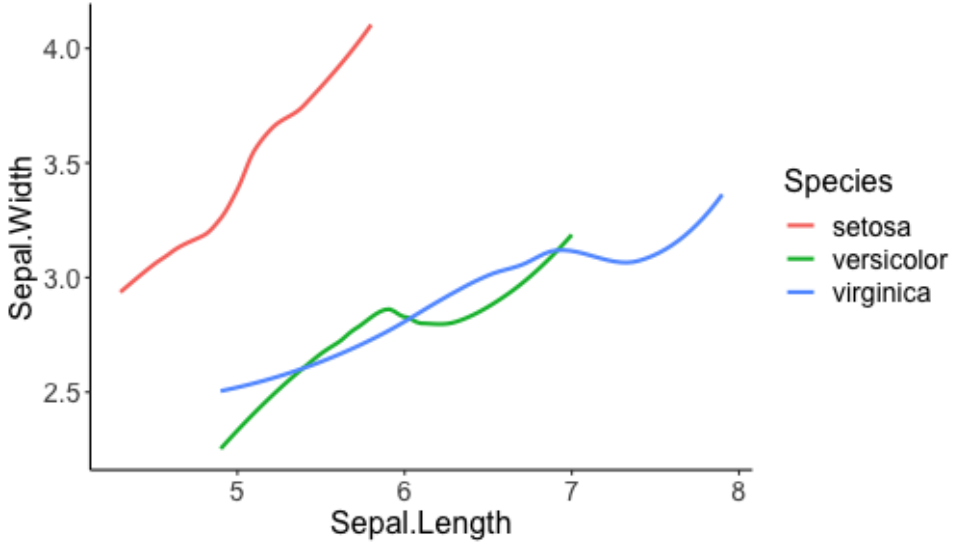



```
ggplot(USArrests, aes(
  x = fct_reorder(State, Murder),
  y = Murder))+
  geom_bar(stat = 'identity')+
  theme(axis.text = element_text(size=6))+
  coord_flip()
```

Order levels based on last values when plotting 2 variables

The level ordering also shows up in the order of levels in the legends of plots. Suppose you are plotting two variables, grouped by a factor.

```
ggplot(iris, aes(
  x = Sepal.Length, y = Sepal.Width, color = Species)
  geom_smooth(se=F)
```




```
ggplot(iris, aes(
  x = Sepal.Length, y = Sepal.Width,
  color = fct_reorder2(Species,
                       Sepal.Length, Sepal.Width)))+
  geom_smooth(se=F) + labs(color = 'Species')
```

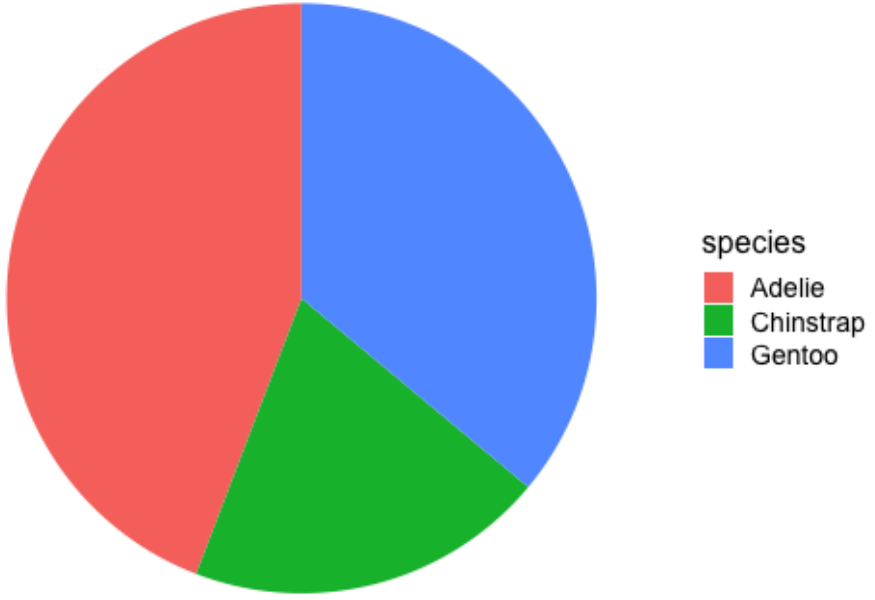
Further exploration

1. forcats cheatsheet
2. Chapter 15 of R4DS

Pie chart

```
penguins %>% count(species) %>%
  ggplot(aes(x=1,y=n, fill=species))+
    geom_bar(stat='identity')+
    coord_polar('y') +
    theme(axis.text = element_blank(),
          axis.title = element_blank(),
          axis.ticks=element_blank(),
          panel.grid = element_blank(),
          axis.line = element_blank())
```

See here and here for more examples.



The nature of a data set

Data characteristics

Some of the things we care about in a data set are

- Nature of each column
- Missing data patterns
- Correlation patterns

The **visdat** package and the **naniar** package help us with visualizing these.

Without visualization

```
summary(airquality)
```

```
     Ozone           Solar.R          Wind      
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700  
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400  
 Median : 31.50   Median :205.0   Median : 9.700  
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958  
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500  
 Max.   :168.00   Max.   :334.0   Max.   :20.700  
 NA's   :37       NA's   :7                       
     Month            Day      
 Min.   :5.000   Min.   : 1.0  
 1st Qu.:6.000   1st Qu.: 8.0  
 Median :7.000   Median :16.0  
 Mean   :6.993   Mean   :15.8  
 3rd Qu.:8.000   3rd Qu.:23.0  
 Max.   :9.000   Max.   :31.0  
```

```
glimpse(airquality, width=40)
```

```
Rows: 153
Columns: 6
$ Ozone   <int> 41, 36, 12, 18, NA, 28…
$ Solar.R <int> 190, 118, 149, 313, NA…
$ Wind    <dbl> 7.4, 8.0, 12.6, 11.5, …
$ Temp    <int> 67, 72, 74, 62, 56, 66…
$ Month   <int> 5, 5, 5, 5, 5, 5, 5, 5…
$ Day     <int> 1, 2, 3, 4, 5, 6, 7, 8…
```

These give us a variable-by-variable view.

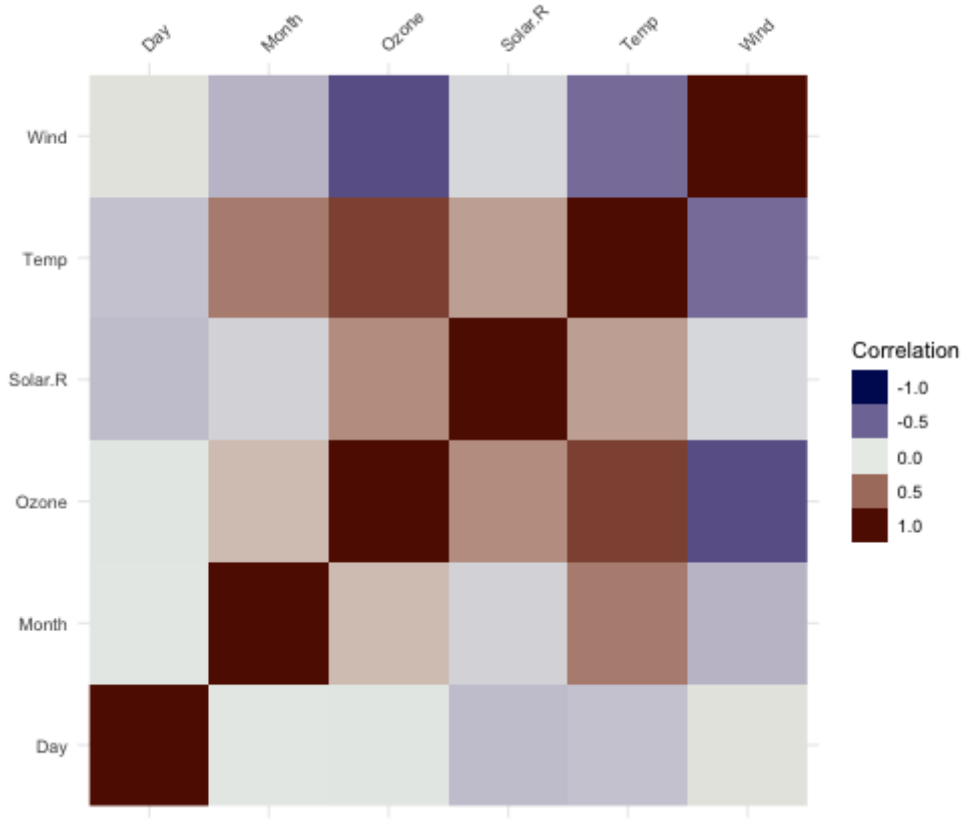
Visualizing a dataset

```
visdat::vis_dat(airquality)
```

- What kinds of variables are in the dataset
- Which elements are missing
- A sense of missing patterns

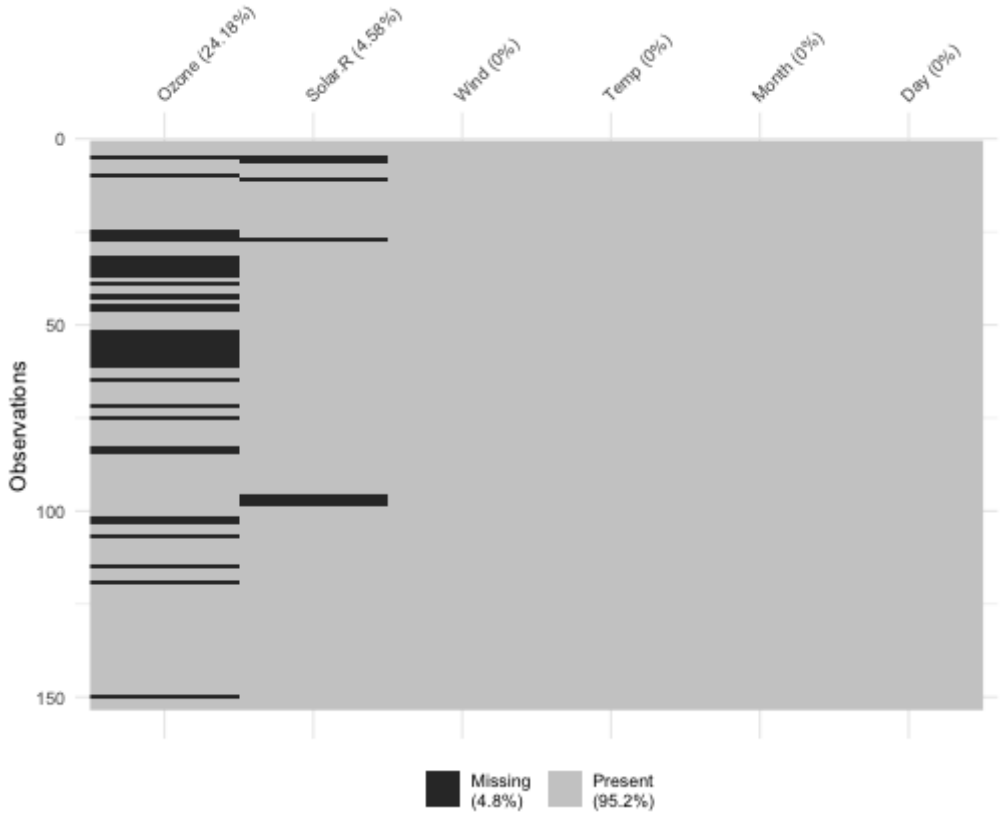
Correlation patterns

```
visdat::vis_cor(airquality)
```



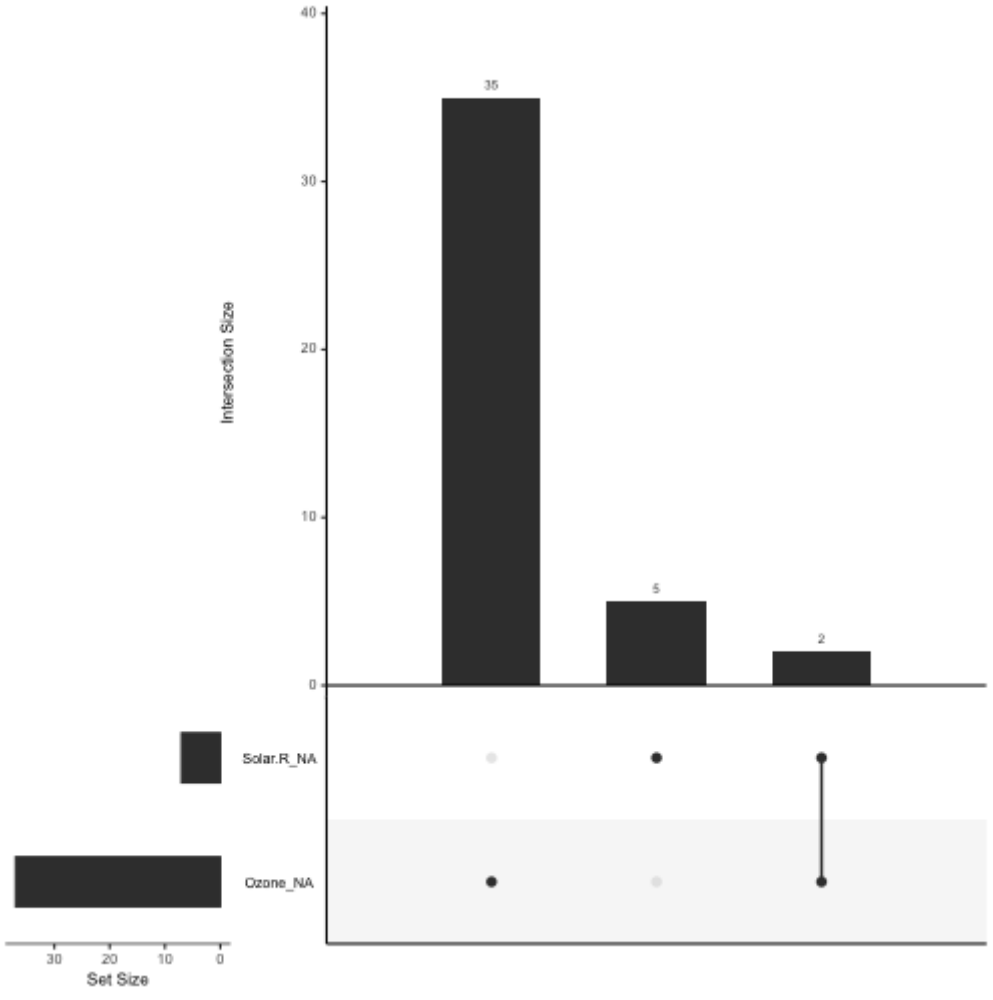
Focus on missing data patterns

```
visdat::vis_miss(airquality)
```

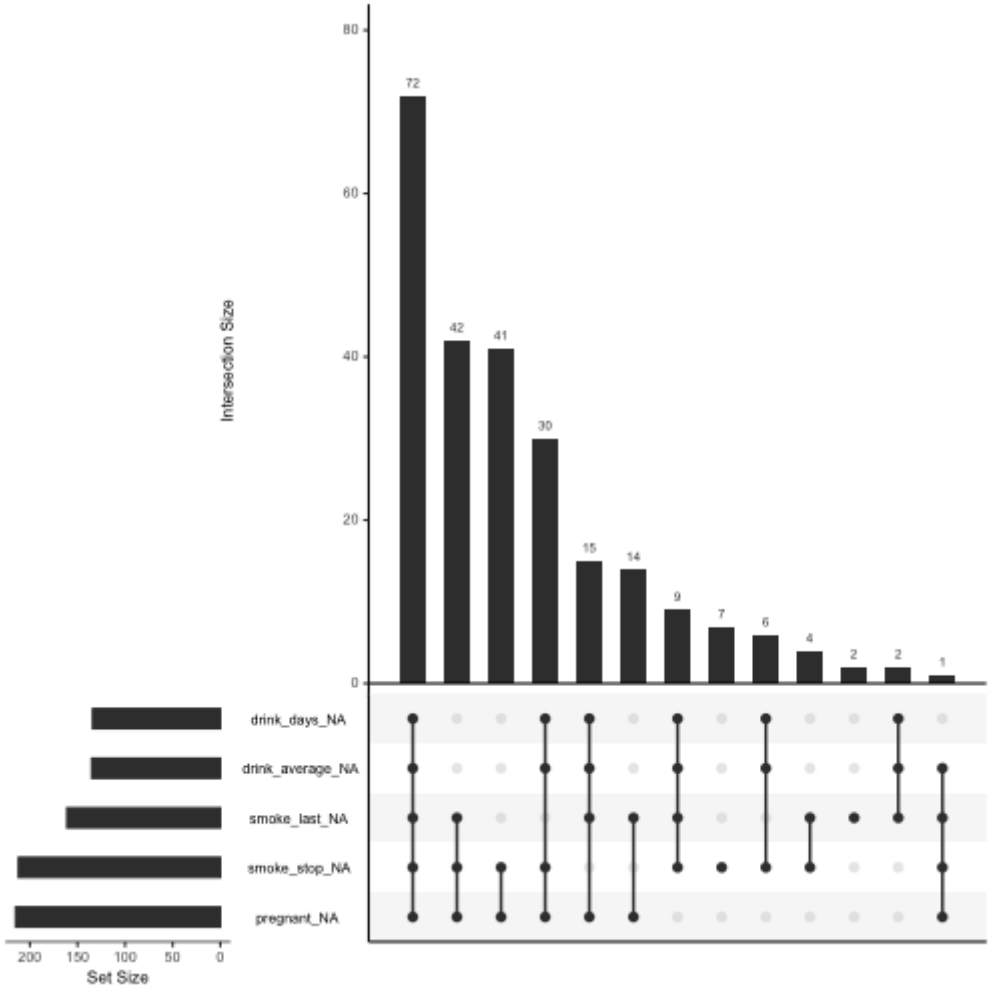


A deeper look at missing data

```
library(naniar)
gg_miss_upset(airquality)
```



```
gg_miss_upset(riskfactors)
```



Missing at random?

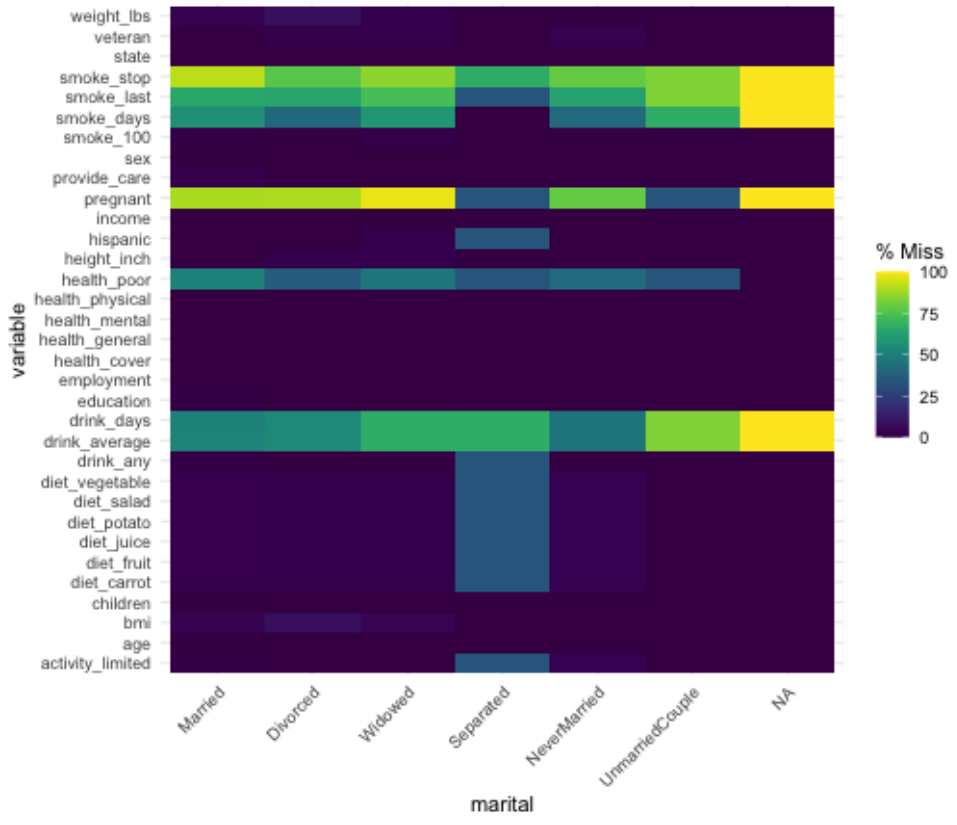
Does missingness in one variable depend on values of another variable?

```
ggplot(airquality,
       aes(Ozone, Solar.R))+
  geom_miss_point()
```

The red points are the values of one variable when the other variable is missing

Missing at random?

```
gg_miss_fct(x = riskfactors, fct=marital)
```



Percent missing in each variable by levels of a factor

What you're looking for is relatively even colors across

Further exploration

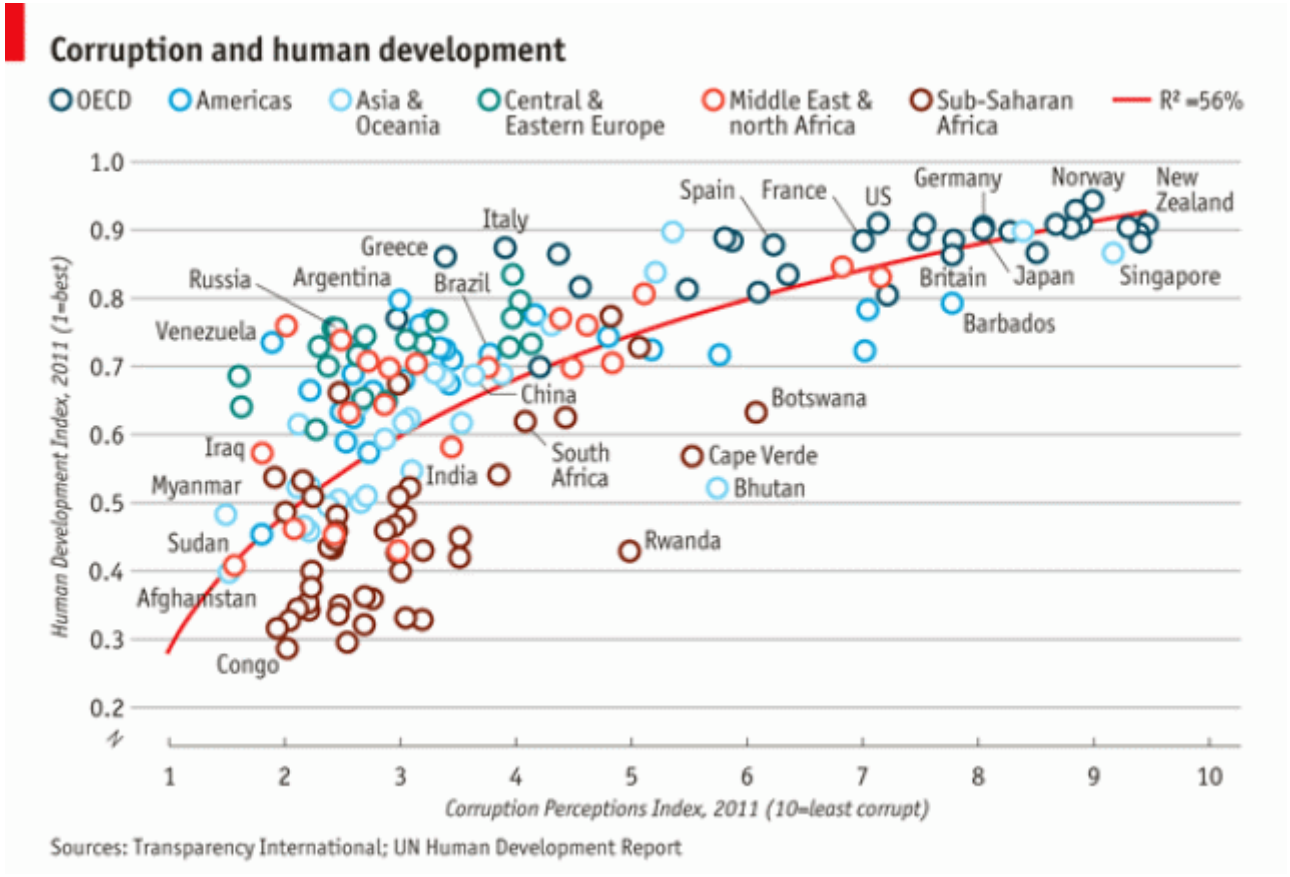
1. The **naniar** website

Annotations

Stand-alone stories

- You would like a data visualization to stand on its own
- Relevant information should be placed on the graph
- However, you need to balance the information content with real estate
    - Don't clutter the graph and make it not readable

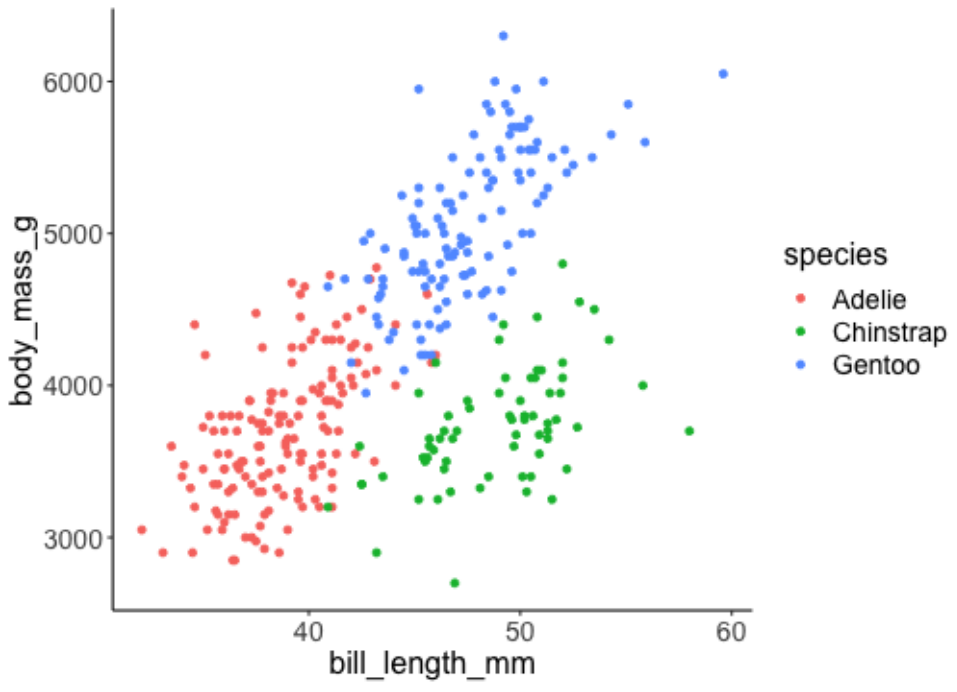
An example



After this week, you can try to reproduce this plot. The data is available at EconomistData.csv in the data folder.

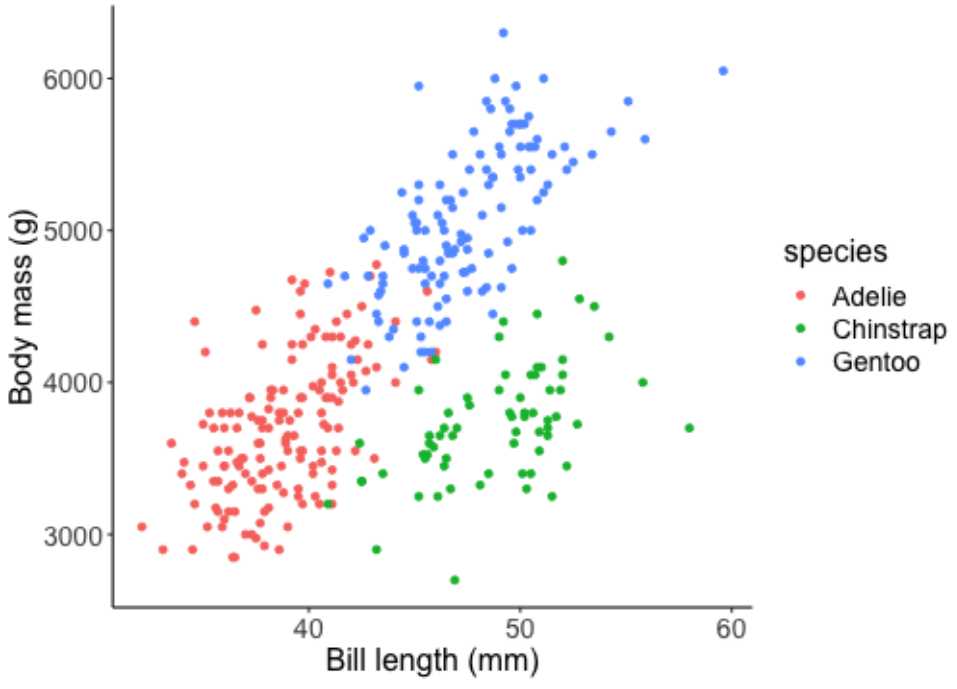
Titles, subtitles and captions

```
library(palmerpenguins)
(plt <- ggplot(penguins,
               aes(bill_length_mm, body_mass_g,
                   color=species))+
  geom_point())
```



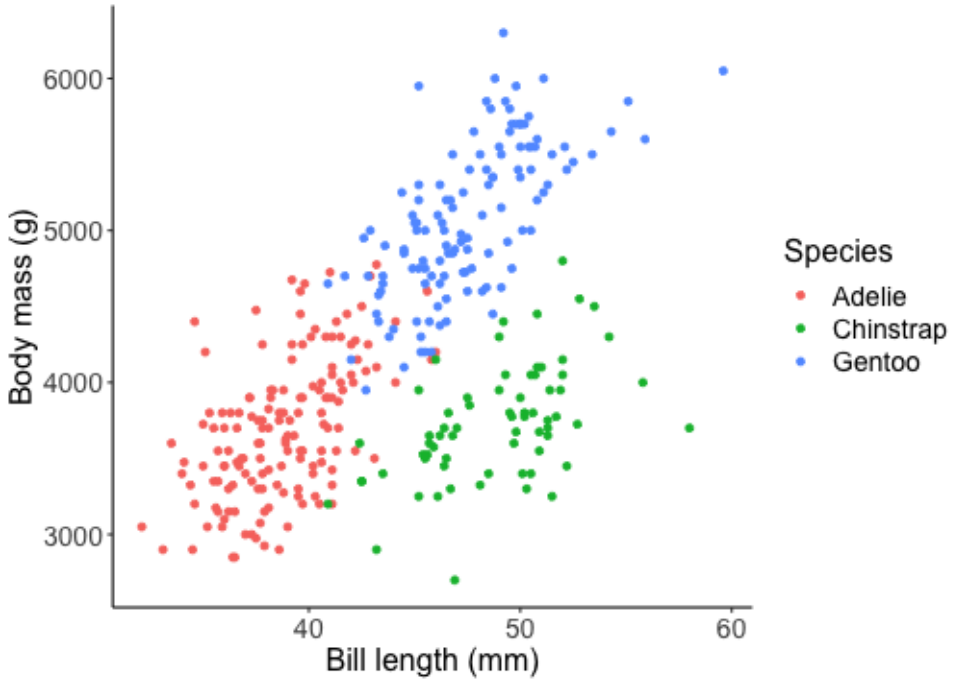
Titles, subtitles and captions

```
library(palmerpenguins)
plt <- ggplot(penguins,
              aes(bill_length_mm, body_mass_g,
                  color=species))+
  geom_point()
plt +
  labs(x = 'Bill length (mm)',
       y = 'Body mass (g)')
```



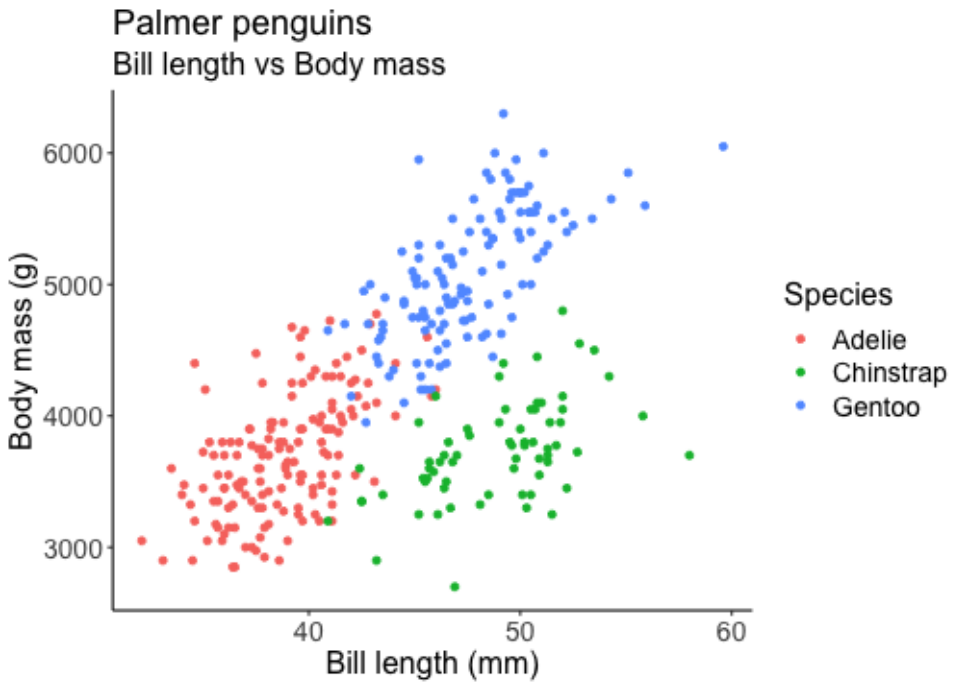
Titles, subtitles and captions

```
library(palmerpenguins)
plt <- ggplot(penguins,
              aes(bill_length_mm, body_mass_g,
                  color=species))+
  geom_point()
plt +
  labs(x = 'Bill length (mm)',
       y = 'Body mass (g)',
       color = 'Species')
```



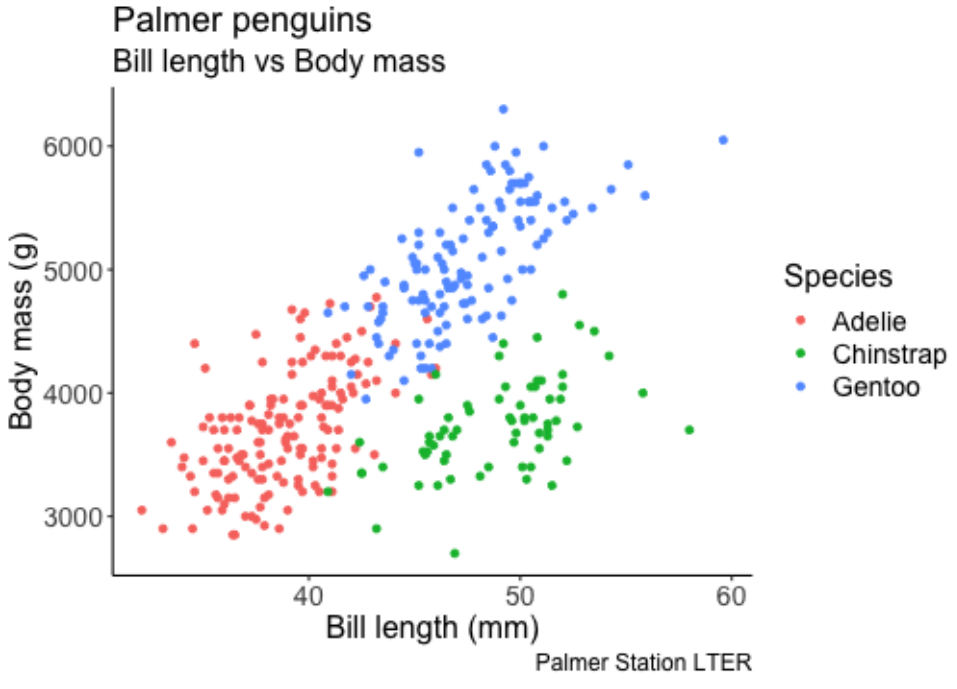
Titles, subtitles and captions

```
library(palmerpenguins)
plt <- ggplot(penguins,
              aes(bill_length_mm, body_mass_g,
                  color=species))+
  geom_point()
plt +
  labs(x = 'Bill length (mm)',
       y = 'Body mass (g)',
       color = 'Species',
       title = "Palmer penguins",
       subtitle = "Bill length vs Body mass")
```



Titles, subtitles and captions

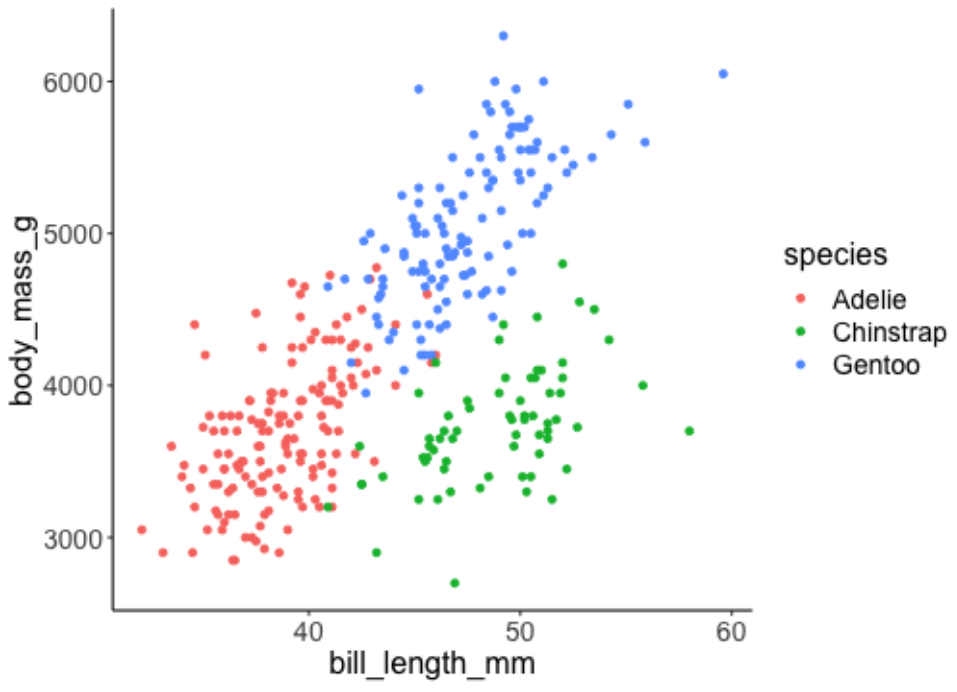
```
library(palmerpenguins)
plt <- ggplot(penguins,
              aes(bill_length_mm, body_mass_g,
                  color=species))+
  geom_point()
plt +
  labs(x = 'Bill length (mm)',
       y = 'Body mass (g)',
       color = 'Species',
       title = "Palmer penguins",
       subtitle = "Bill length vs Body mass",
       caption = "Palmer Station LTER")
```



Adding derived statistics to a plot

Adding group means

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

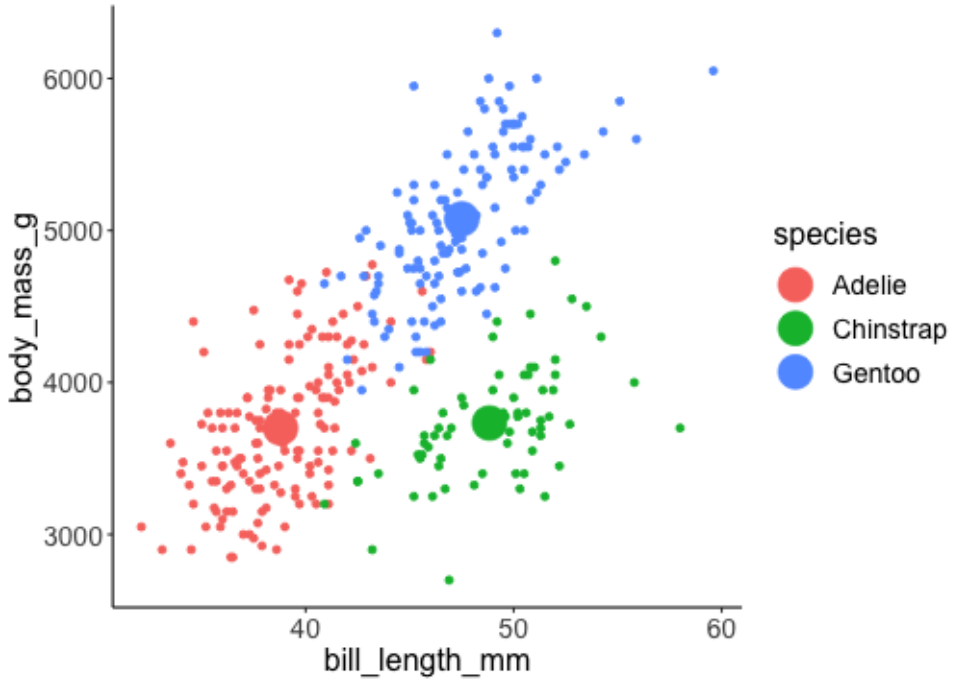


Adding group means

```
means <- penguins %>% group_by(species) %>%
  summarize_at(vars(bill_length_mm, body_mass_g),
               mean, na.rm=TRUE)
means
```

```
  # A tibble: 3 x 3
    species   bill_length_mm body_mass_g
    <fct>              <dbl>       <dbl>
  1 Adelie              38.8       3701.
  2 Chinstrap           48.8       3733.
  3 Gentoo              47.5       5076.
```

```
ggplot(penguins,
       aes(x = bill_length_mm,
           y = body_mass_g,
           color = species))+
  geom_point()+
  geom_point(data = means,
             size=8)
```

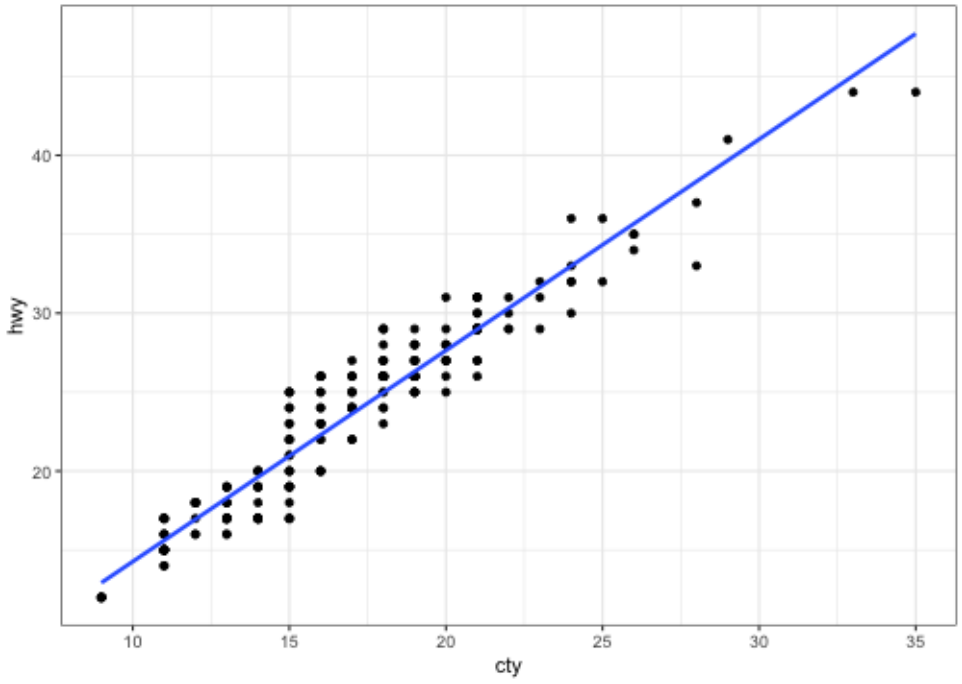


Adding data from a different dataset

Adding regression metrics

Regress highway mileage on city mileage (data: mpg)

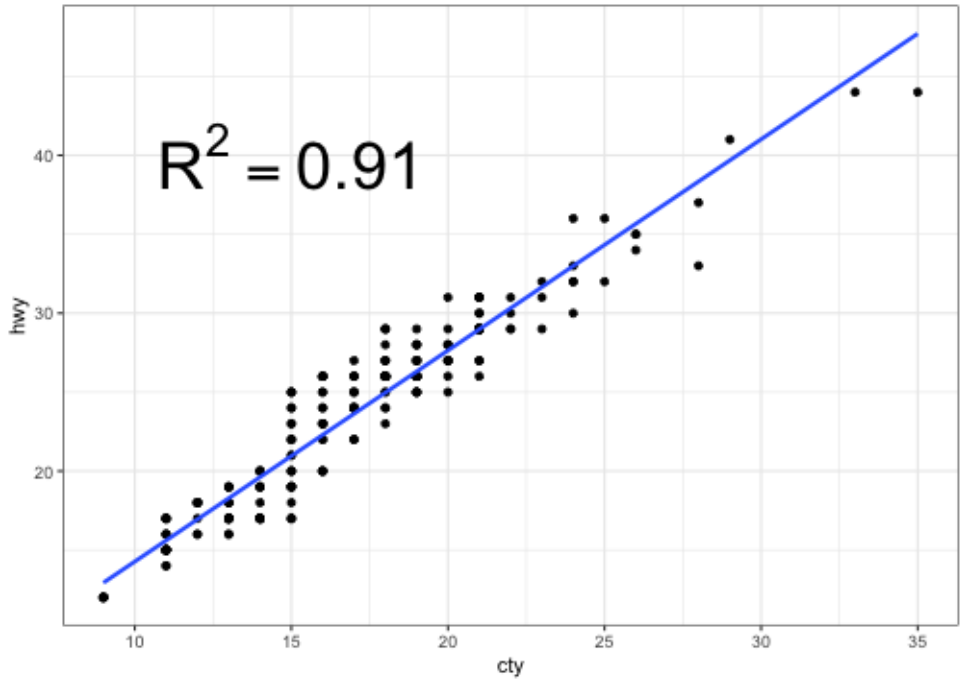
```
mod1 <- lm(hwy ~ cty, data = mpg)
r2 <- broom::glance(mod1) %>% pull(r.squared)

ggplot(mpg,
       aes(x = cty, y = hwy))+
  geom_point() +
  geom_smooth(method = 'lm', se=F) +
  theme_bw()
```



Adding regression metrics

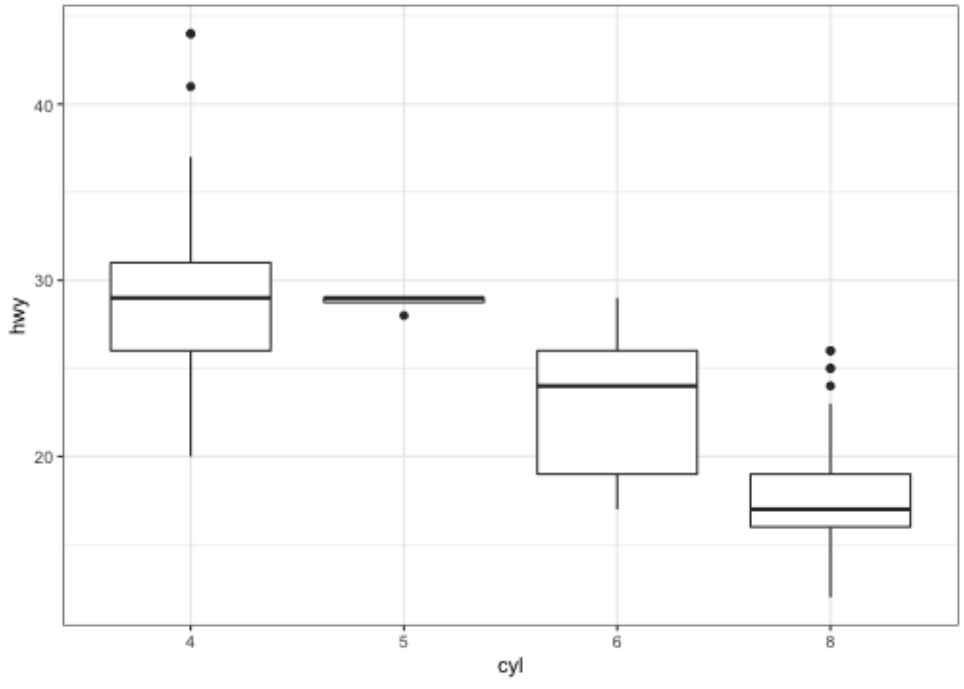
Regress highway mileage on city mileage (data: mpg)

```
mod1 <- lm(hwy ~ cty, data = mpg)
r2 <- broom::glance(mod1) %>% pull(r.squared) %>%
  round(., 2)

ggplot(mpg,
       aes(x = cty, y = hwy))+
  geom_point() +
  geom_smooth(method = 'lm', se=F)+
  annotate(geom='text',
           x = 15, y = 40,
           label=glue::glue("R^2 == {r}",r=r2),
           size=12,
           parse=T) +
  theme_bw()
```



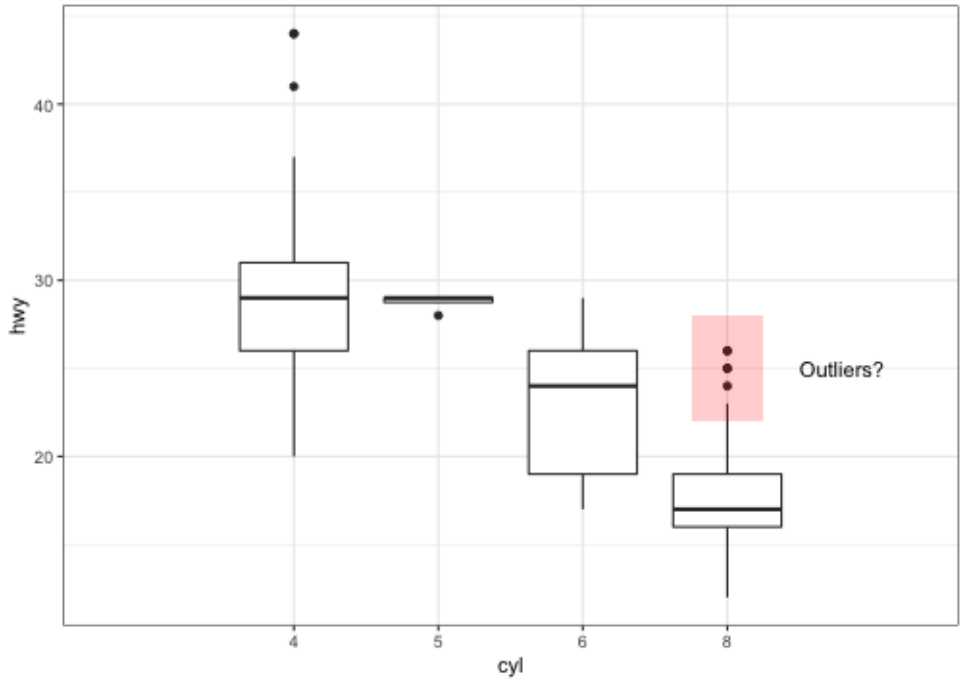
Highlighting regions

```
mpg %>%
  mutate(cyl = as.factor(cyl)) %>%
  ggplot(aes(x = cyl, y = hwy)) +
  geom_boxplot() +
  theme_bw()
```



Highlighting regions

```
mpg %>%
  mutate(cyl = as.factor(cyl)) %>%
  ggplot(aes(x = cyl, y = hwy)) +
  geom_boxplot() +
  theme_bw()+
  annotate(geom = 'rect',
           xmin=3.75,xmax=4.25,
           ymin = 22, ymax = 28,
           fill = 'red',
           alpha = 0.2) +
  annotate('text',
           x = 4.5, y = 25,
           label = 'Outliers?',
           hjust = 0)+
  coord_cartesian(xlim = c(0,5))+
  theme_bw()
```



Note: If you have a factor on the x-axis, they are plotted at 1, 2, 3, ...

layout: true

Putting plots together

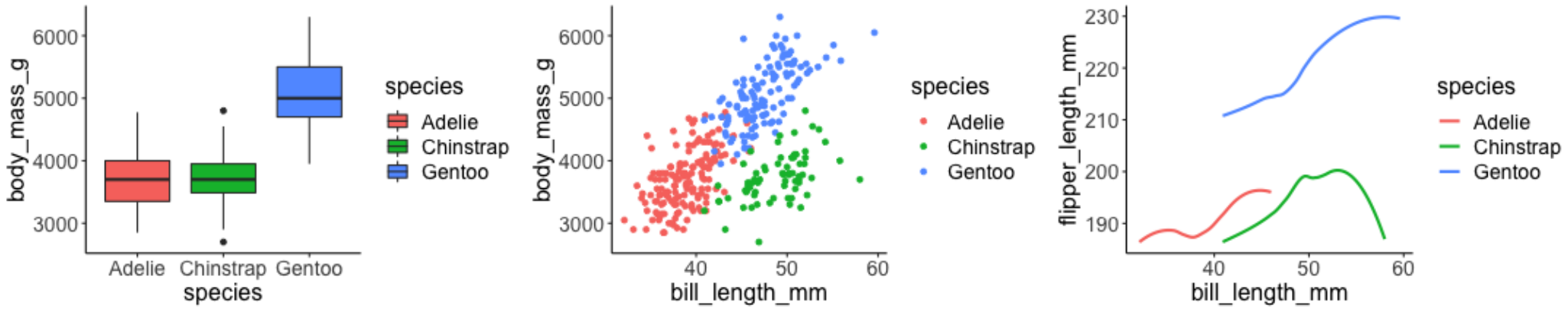
The packages

There are three excellent packages for putting separate `ggplot` graphs together in panels.

1. **ggpubr**
2. **cowplot**
3. **patchwork**

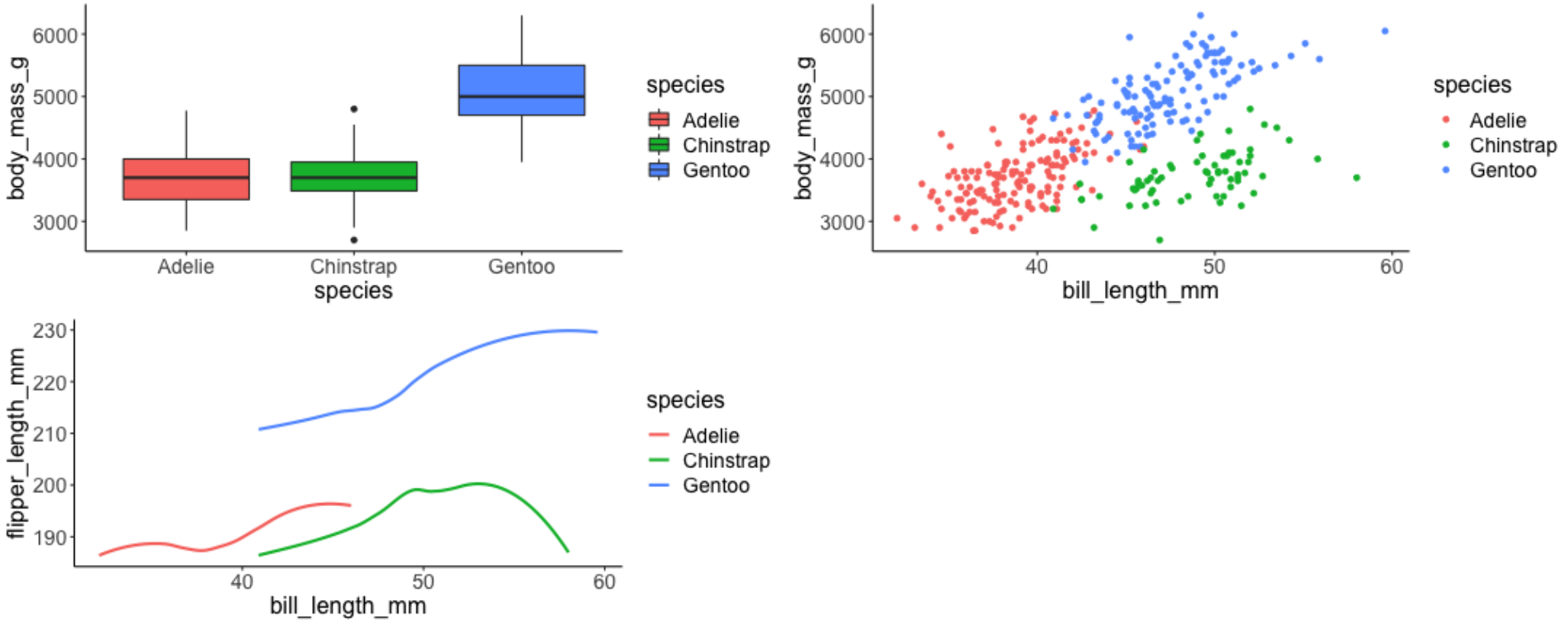
The graphs

```
plt1 <- ggplot(penguins, aes(x = species, y = body_mass_g, fill=species)) +
  geom_boxplot()

plt2 <- ggplot(penguins, aes(x = bill_length_mm, y = body_mass_g,
                             color = species))+
  geom_point()

plt3 <- ggplot(penguins, aes(x = bill_length_mm, y = flipper_length_mm,
                             color = species))+
  geom_smooth(se=F)
```



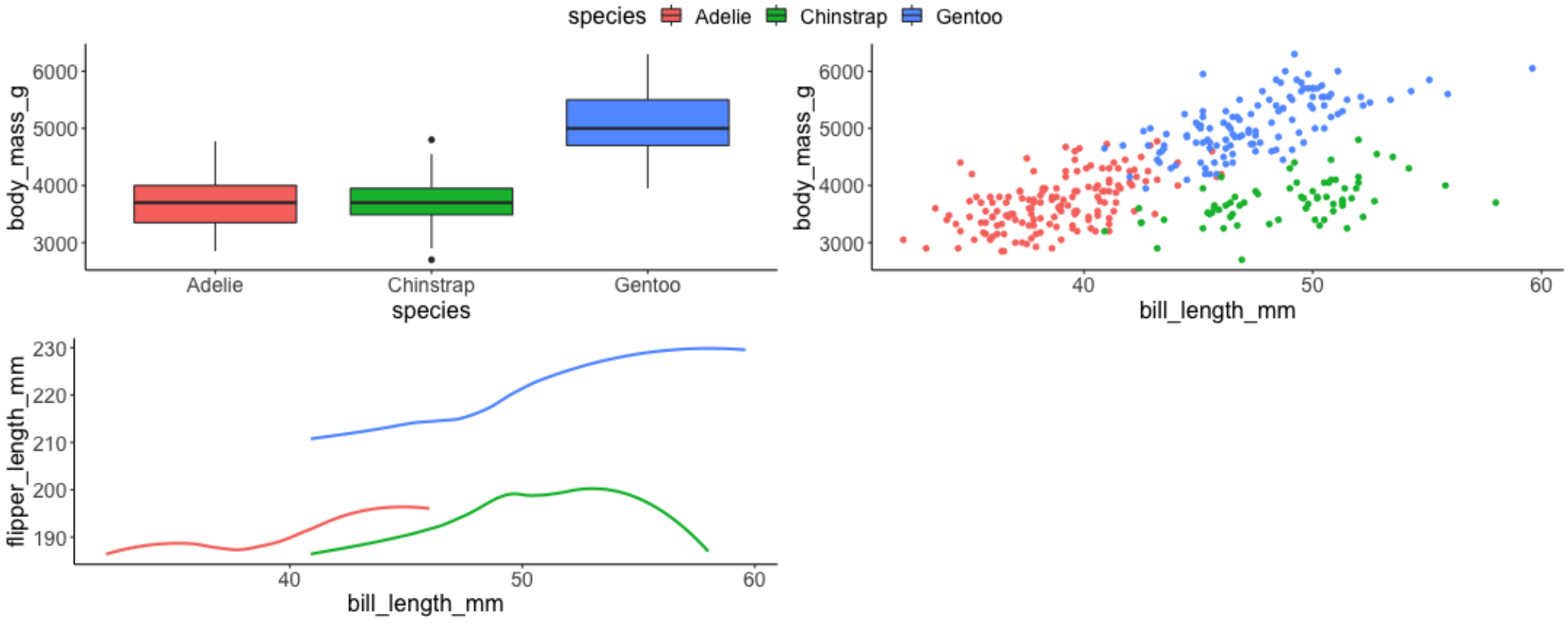
ggpubr

```
ggarrange(plt1, plt2, plt3, ncol = 2, nrow=2)
```



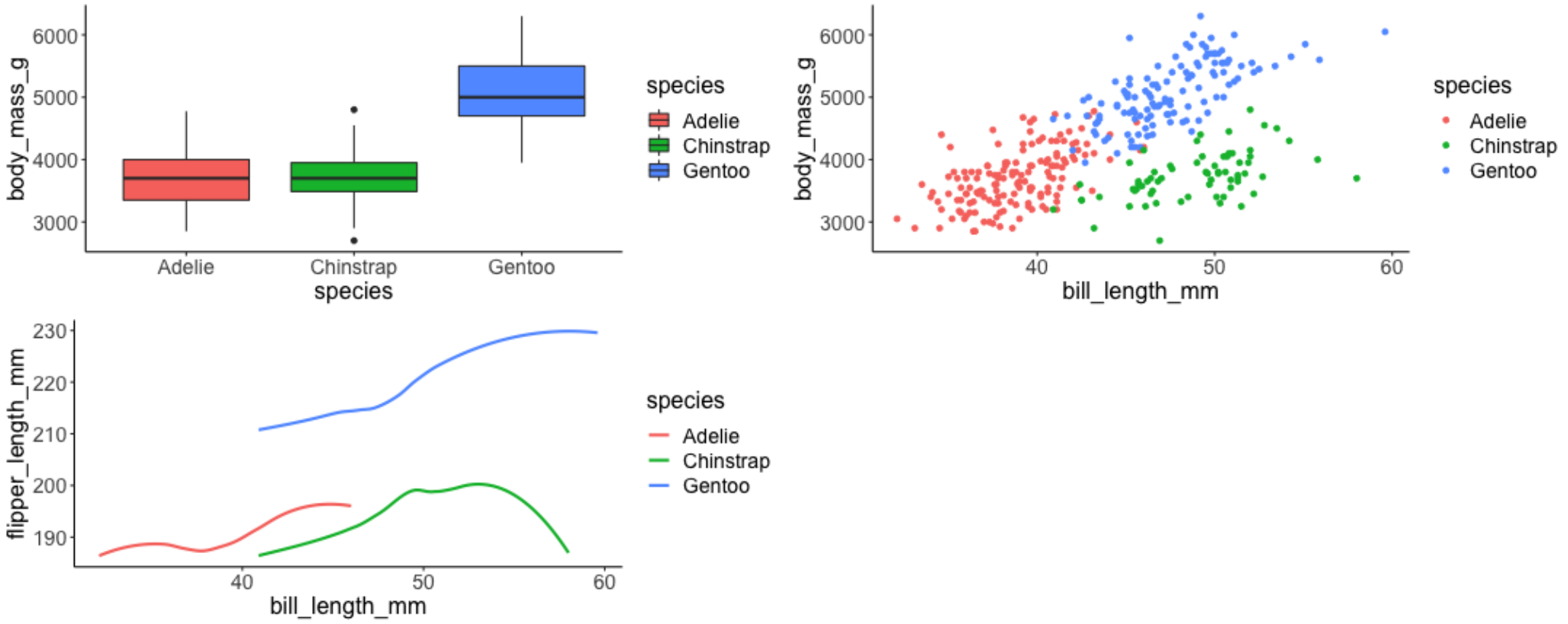
ggpubr

```
ggarrange(plt1, plt2, plt3, ncol = 2, nrow=2, common.legend = TRUE)
```



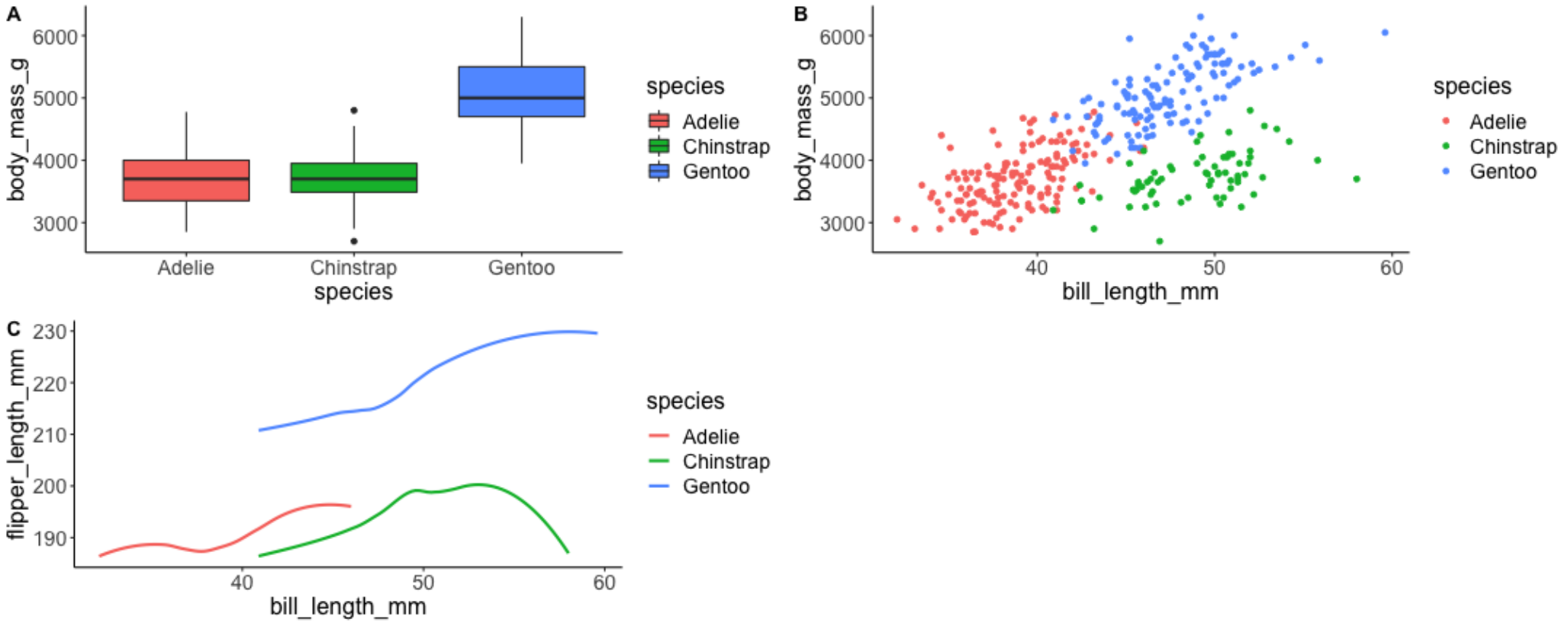
cowplot

```
cowplot::plot_grid(plt1, plt2, plt3, nrow = 2, ncol = 2)
```



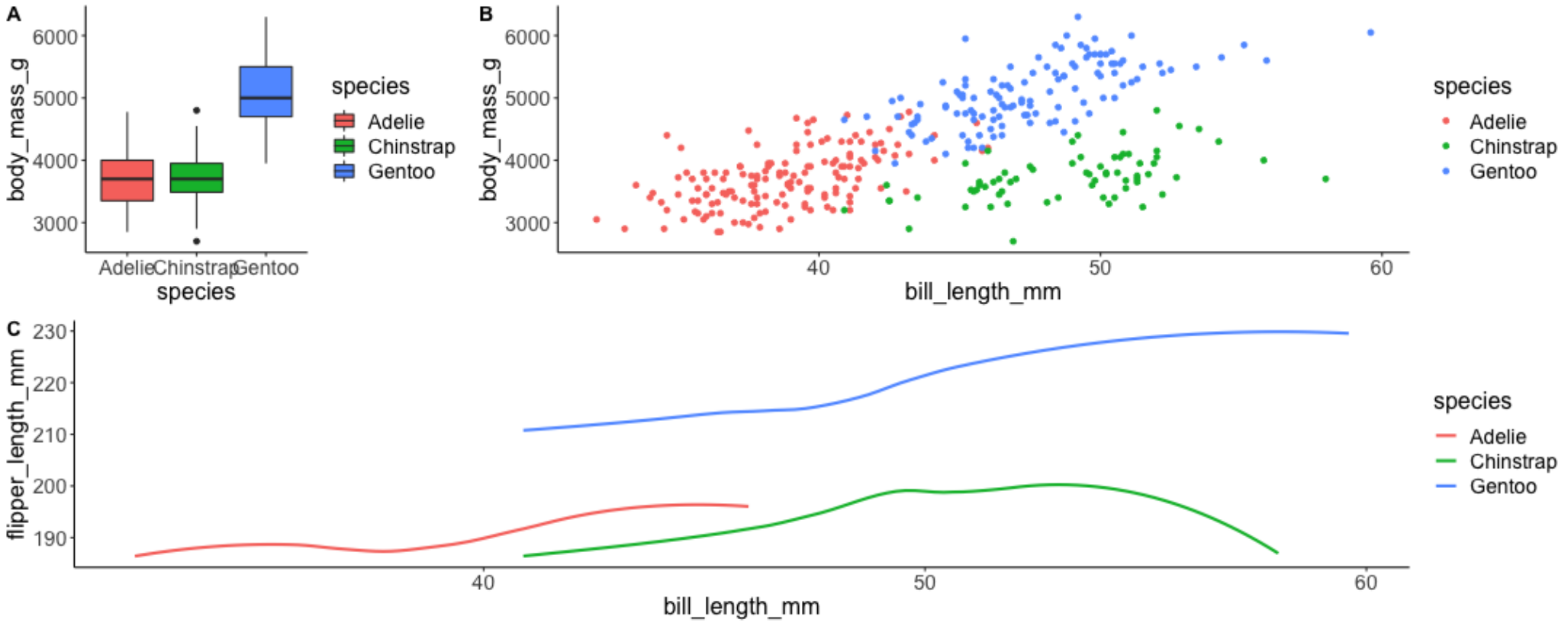
cowplot

```
cowplot::plot_grid(plt1, plt2, plt3, nrow = 2, ncol = 2, labels = c('A','B','C'))
```



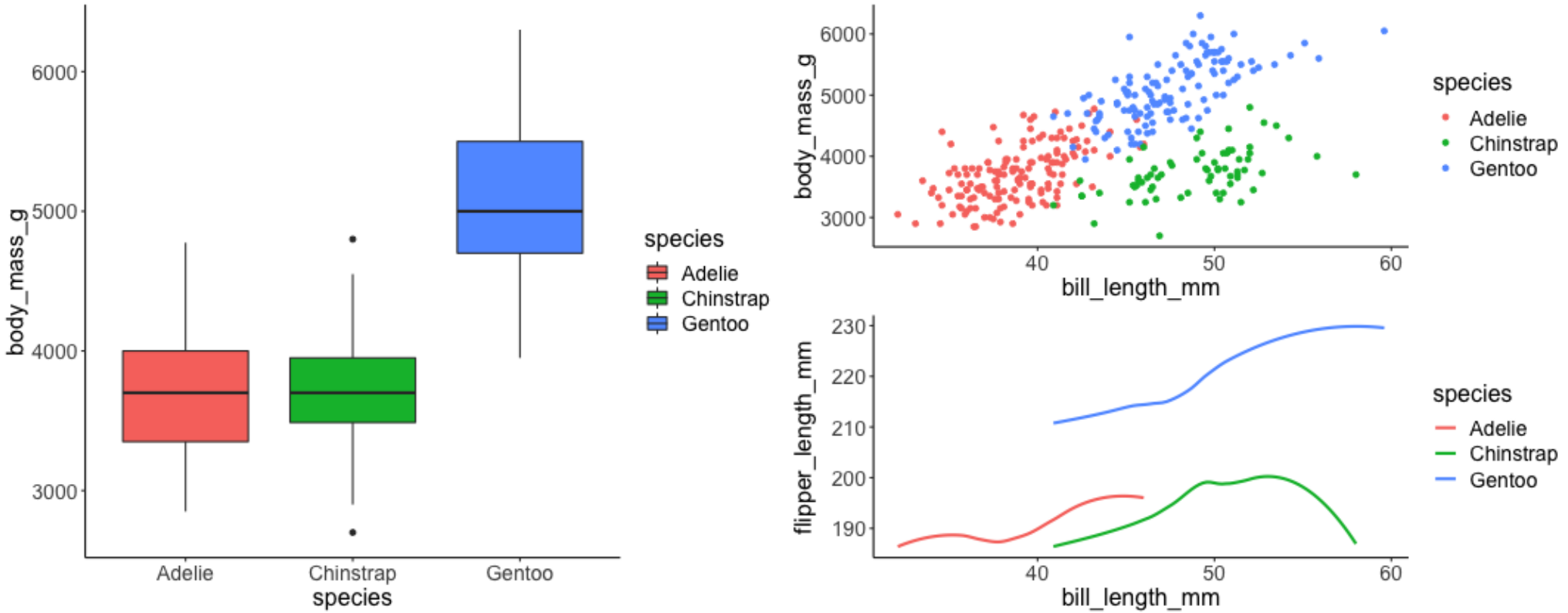
cowplot

```
grid1 = cowplot::plot_grid(plt1, plt2, nrow = 1, rel_widths=c(0.3, 0.7),
                           labels=c('A','B'))
cowplot::plot_grid(grid1, plt3, nrow=2, labels = c('', 'C'))
```



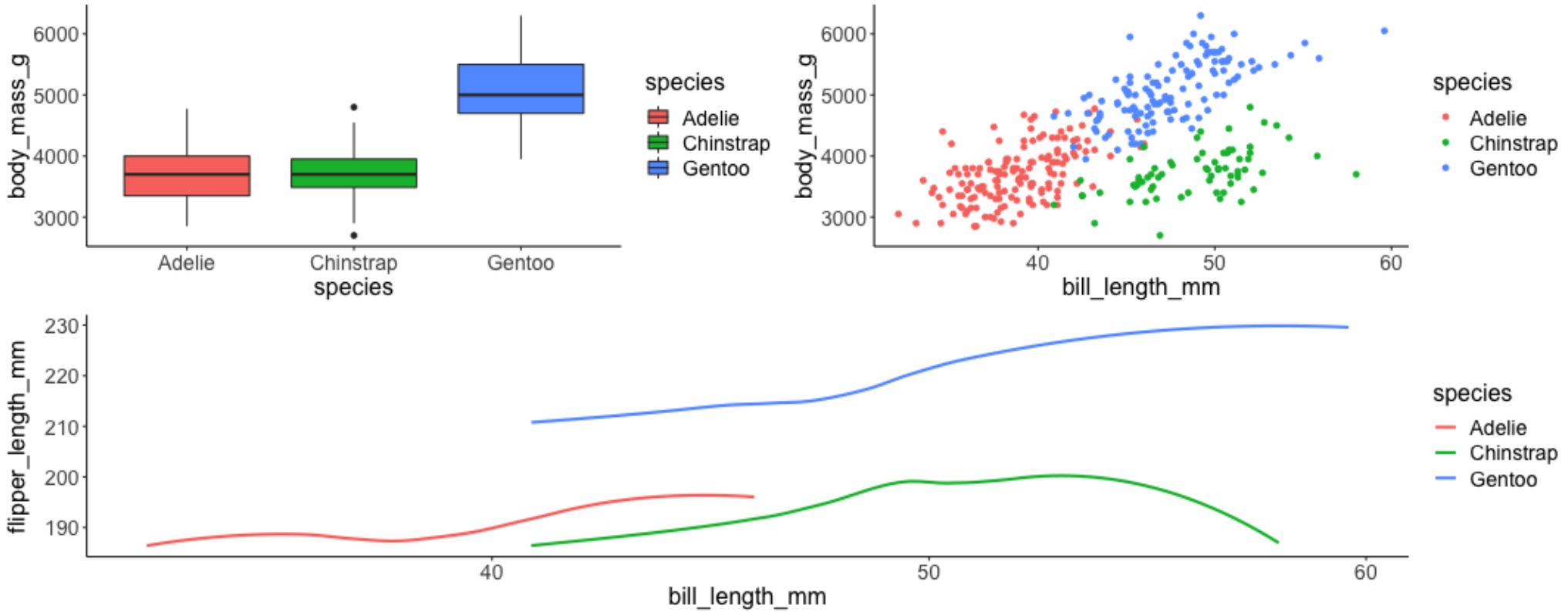
patchwork

```
plt1 | plt2 / plt3
```



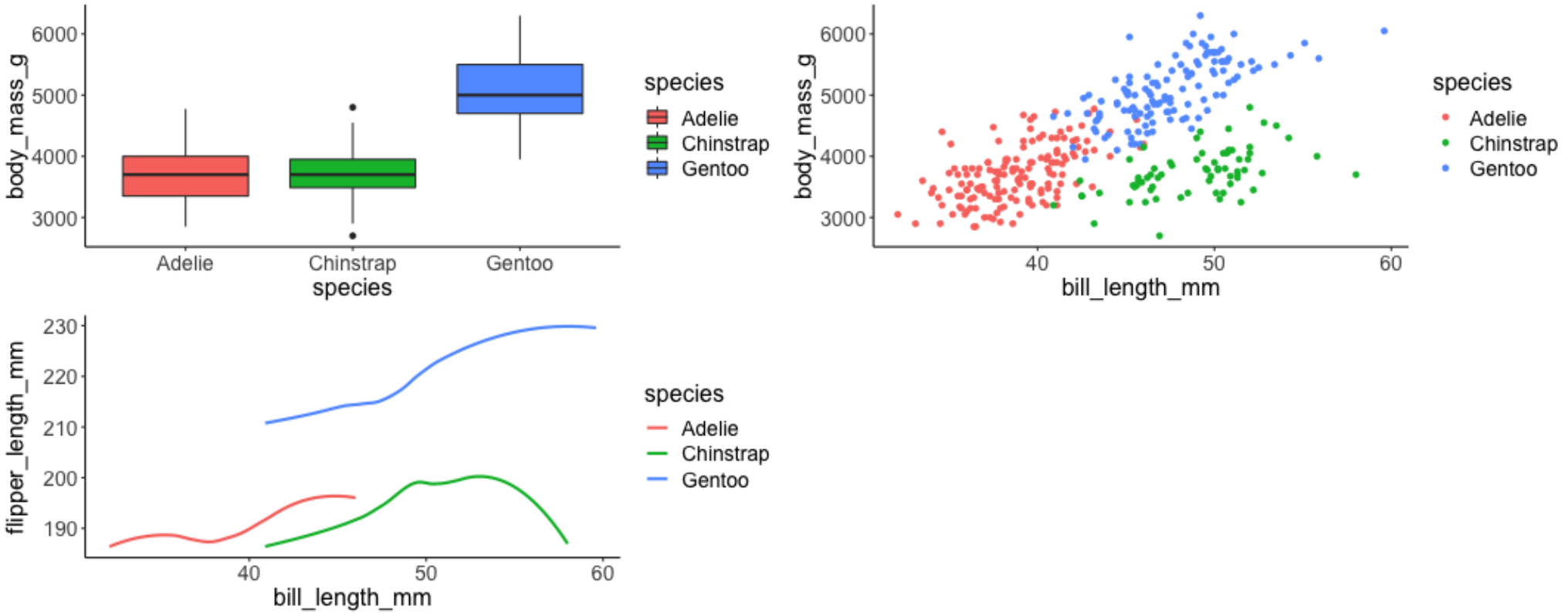
patchwork

```
(plt1 | plt2) / plt3
```



patchwork

```
plt1 + plt2 + plt3 + plot_layout(ncol = 2)
```



Further references

ggpubr: https://rpkgs.datanovia.com/ggpubr/
cowplot: https://wilkelab.org/cowplot/index.html and Fundamentals of Data Visualization
patchwork: https://patchwork.data-imaginist.com/index.html

Saving your work

You want to actually use the visualizations you make

- Save to file
  - PNG for web
  - PDF for print
  - High resolution PNG for Word (600-1200 dpi)
  - Journals often want high resolution TIFF (300+ dpi)
- Save to document
  - Create a Word file from R Markdown
  - Create a PowerPoint file from R Markdown.

Save to file

Printers in R

R allows you to save graphics by using **printers** for PDF, PNG and the like.

```
pdf('temp.pdf', width=5, height=5) # inches
ggplot(penguins, aes(bill_length_mm, body_mass_g, color=species))+
  geom_point() +
  labs(x = 'Bill length (mm)',
       y = 'Body mass (g)',
       color = 'Species')
dev.off()
```

Printers in R

R allows you to save graphics by using **printers** for PDF, PNG and the like.

```
png('temp.png', width=5, height=5, units='in', res=300) # 300 dpi
ggplot(penguins, aes(bill_length_mm, body_mass_g, color=species))+
  geom_point() +
  labs(x = 'Bill length (mm)',
       y = 'Body mass (g)',
       color = 'Species')
dev.off()
```

Printers in R

R allows you to save graphics by using **printers** for PDF, PNG and the like.

```
tiff('temp.tif', width=5, height=5, units='in', res=300, compression='lzw') # 300 dpi
ggplot(penguins, aes(bill_length_mm, body_mass_g, color=species))+
  geom_point() +
  labs(x = 'Bill length (mm)',
       y = 'Body mass (g)',
       color = 'Species')
dev.off()
```

Printers in R

Issues with tiff on a Mac

The `tiff` printer doesn't annotate the TIFF file properly, so Preview thinks it's at 72 dpi, regardless of the setting.

The workaround is to print to PDF, and convert to TIFF, either via Preview, or using the **pdftools** package.

```
pdf('temp.pdf', width=5, height=5) # inches
ggplot(penguins, aes(bill_length_mm, body_mass_g, color=species))+
  geom_point() +
  labs(x = 'Bill length (mm)',
       y = 'Body mass (g)',
       color = 'Species')
dev.off()
pdftools::pdf_convert('temp.pdf', format='tiff', dpi=300)
```

ggplot2 savings

The previous slides showed the basic R way of printing a plot to a file. **ggplot2** makes it a bit easier.

```
ggplot(penguins, aes(bill_length_mm, body_mass_g, color=species))+
  geom_point() +
  labs(x = 'Bill length (mm)',
       y = 'Body mass (g)',
       color = 'Species')
ggsave('temp.pdf', width=5, height=5)
```

`ggsave` figures out the type from the ending. If you use `temp.png` it will create a PNG file.

Note, in all of the examples, the file gets saved to the working directory (`getwd()`).

Practice

Save to PDF by default

Why?

- PDF is *infinite resolution*. As a vector format, it can be infinitely magnified.
- PNG, TIFF are raster formats, so if you magnify too much, you'll see pixels
- Convert from PDF to other raster formats saves both resolution and disk space.

Save to document

Saving to a document

From the same R Markdown where you create the plot, you can save to Word or PowerPoint (even if you don't have it on your computer) by changing the *front matter* on top (between the `---`)

- For Word, use `output: word_document`
- For PowerPoint, use `output: powerpoint_presentation`

You can also learn the excellent **officer** package to directly create Word and PowerPoint presentations from R programmatically. See the website at https://davidgohel.github.io/officer/index.html.
