SOLES codeRs

The principles and practice of datavis with `ggplot2`

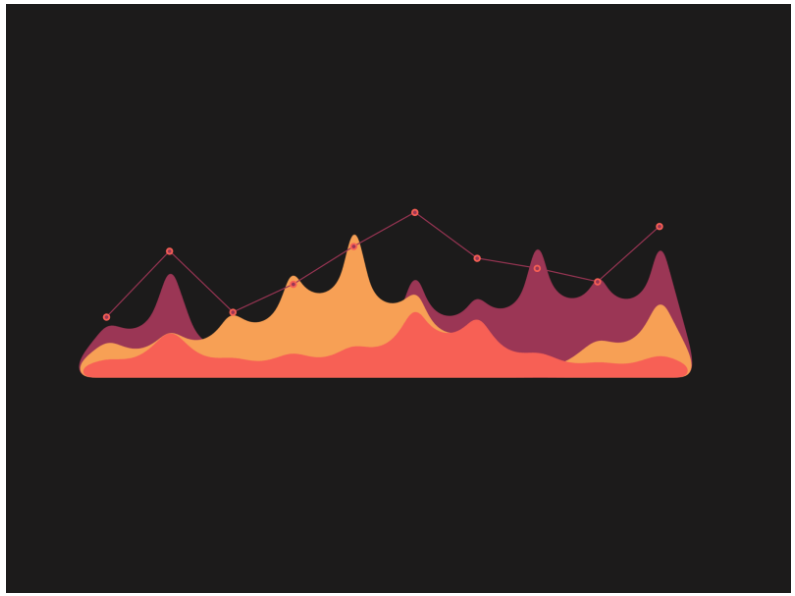
Thomas White

Outline

- Articulate the principles of effective data visualisation
- Understand the foundational place of the ‘big 5’ graphics for datavis
- Use `ggplot2` to create and modify common forms of data visualisation to publication standard

Basic principles of data visualisation

1. Show the data in the rawest form possible
2. Create the simplest graphic that conveys the information you want to convey. Maximise the "data-to-ink" ratio.
3. Everything should be intentional



How to violate all datavis principles

Problems typically fall into:

1. Aesthetics
2. Substance
3. Perception

The grammar of graphics

The grammar of graphics

- Languages are comprised of different elements: nouns, verbs, articles, subjects, objects, etc.
- The rules defining their arrangement into a meaningful whole define the *grammar* of the language
- The grammar of *graphics* works similarly, by defining a set of rules for constructing visualisations by combining different types of *layers* (Wilkinson 2005)

In short, the grammar tells us that:

> A statistical graphic is a mapping of **data** variables to **aes**thetic attributes of **geom**etric objects.

ggplot2

> A statistical graphic is a mapping of **data** variables to **aes**thetic attributes of **geom**etric objects.

ggplot2

> A statistical graphic is a mapping of **data** variables to **aes**thetic attributes of **geom**etric objects.

`ggplot2` implements this grammar in R, which breaks graphics into three essential components:

1. `data`: the dataset containing the variables of interest.
2. `geom`: the geometric object in question. The *type* of object we can observe in a plot, such as points, lines, and bars.
3. `aes`: the aesthetic attributes of the geometric object — their appearance. Colour, shape, size, position. Aesthetic attributes are mapped to variables in the `data`.

Building a ggplot

The basic template:

```
ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) +
      <GEOMS>() +
      <CUSTOMISATIONS>
```

Building a ggplot

Step 1

We use the `ggplot()` function to bind the plot to a specific data frame using the `data` argument

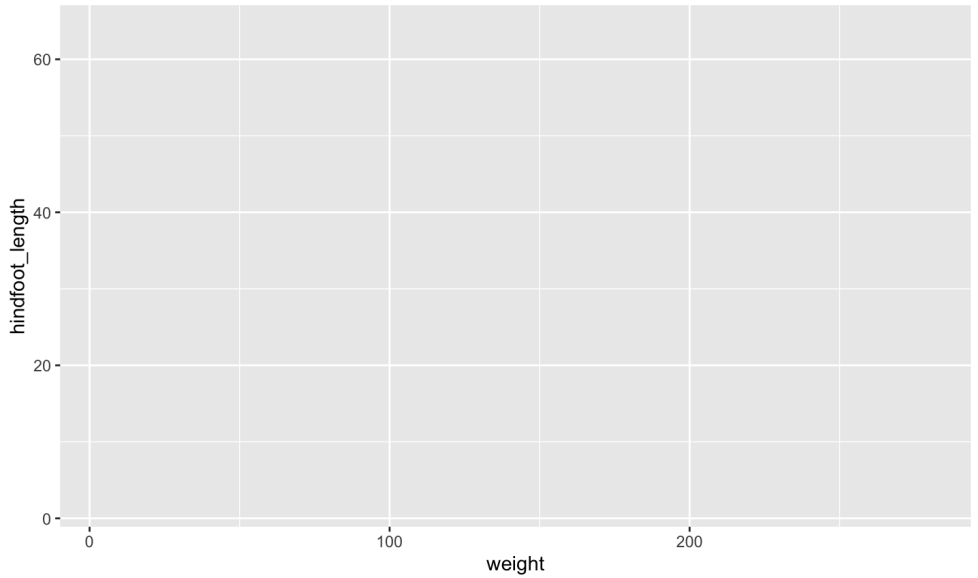
```
ggplot(data = dat_eco_clean)
```


Note the plot is just a large grey blob, because we've only given it the data (& haven't told it how to use it).

Step 2

We then extend it to define an aesthetic mapping using the `aes` function

```
ggplot(data = dat_eco_clean, mapping = aes(x = weight, y = hindfoot_length))
```



Note we have a little more structure here. We've told it *what*, but not *how*.

Step 3

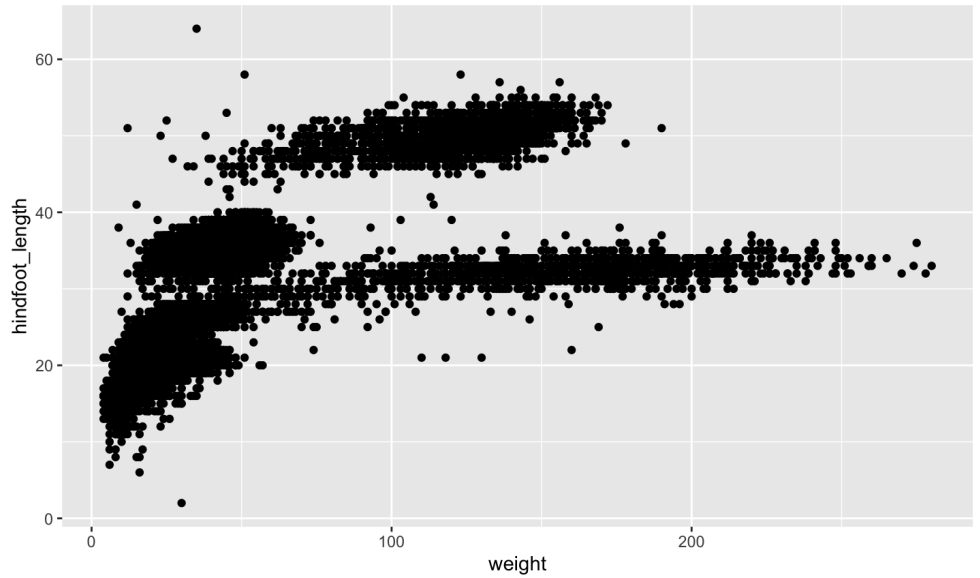
Finally (well, not really) we add a `geom` – our graphical representation of the data in the plot. Here we're use `geom_point` to create a scatterplot (i.e. a plot of points), which sensible for the data at hand. And….

```
ggplot(data = dat_eco_clean, mapping = aes(x = weight, y = hindfoot_length)) +
      geom_point()
```

Step 3

We have a (fairly ugly) plot!

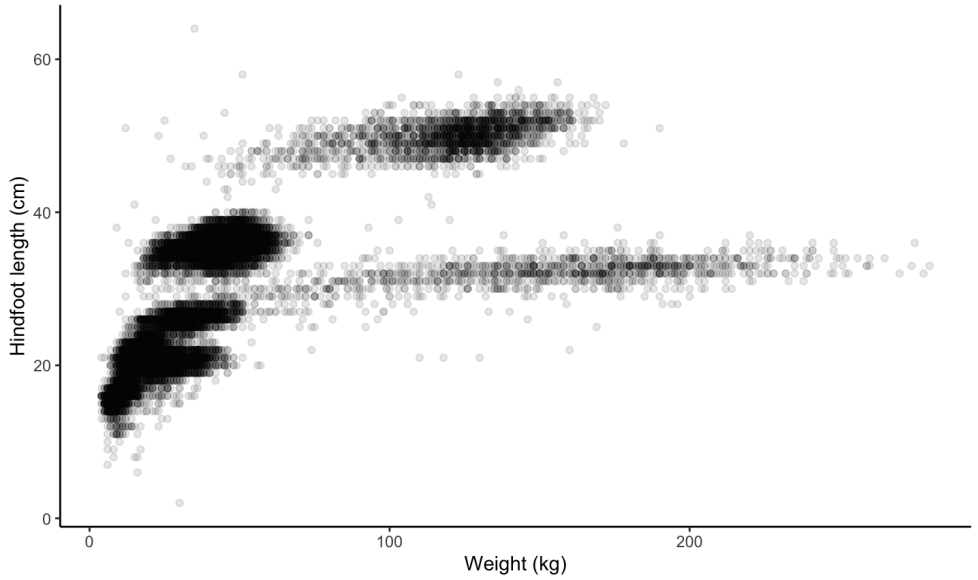
```
ggplot(data = dat_eco_clean, mapping = aes(x = weight, y = hindfoot_length)) +
      geom_point()
```



Step 4 -> inf.

We customise the heck out of it in line with our datavis principles.

```
ggplot(data = dat_eco_clean, mapping = aes(x = weight, y = hindfoot_length)) +
      geom_point(alpha = 0.1) +
      ylab('Hindfoot length (cm)') +
      xlab('Weight (kg)') +
      theme_classic()
```



Hot plot tips 1

- Plots are built iteratively by specifying **data**, **aes**thetics, and **geom**s
- Layers are added using the + at the *end* of a line
- You can assign plots to variables using <–

The big five graphics

The big five

- scatterplots
- linegraphs
- boxplots
- histograms
- barplots

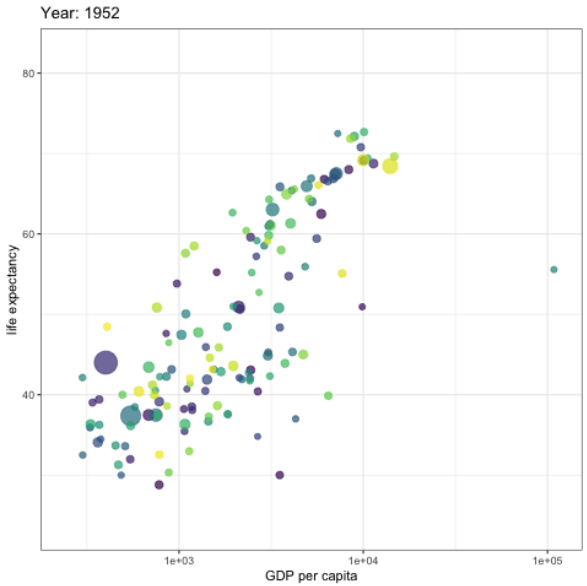
The big five in `ggplot2`

- scatterplots: `geom_point()`
- linegraphs: `geom_line()`
- boxplots: `geom_boxplot()`
- histograms: `geom_histogram()`
- barplots: `geom_bar()` or `geom_col()`

Scatterplots

Scatterplots

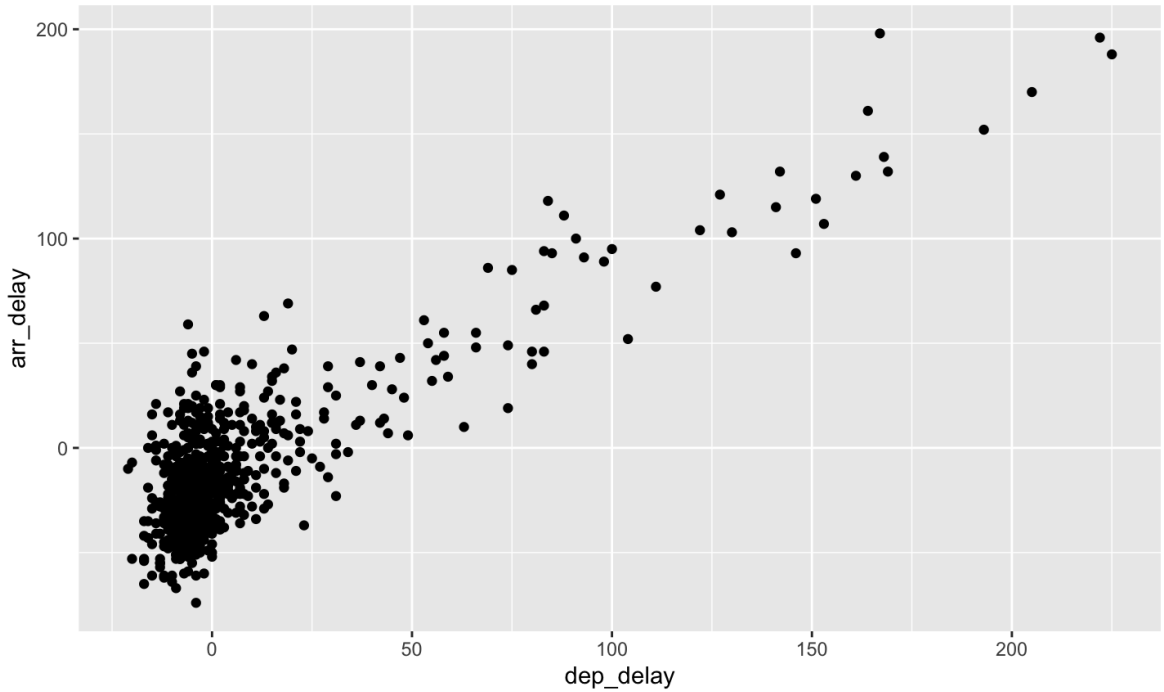
For visualising the relationship between two numerical variables



- Specified via `geom_point()`.
- Let's use it to look at some data on the relationship between flight *departure delays* and *arrival delays* in the US:

Scatterplots

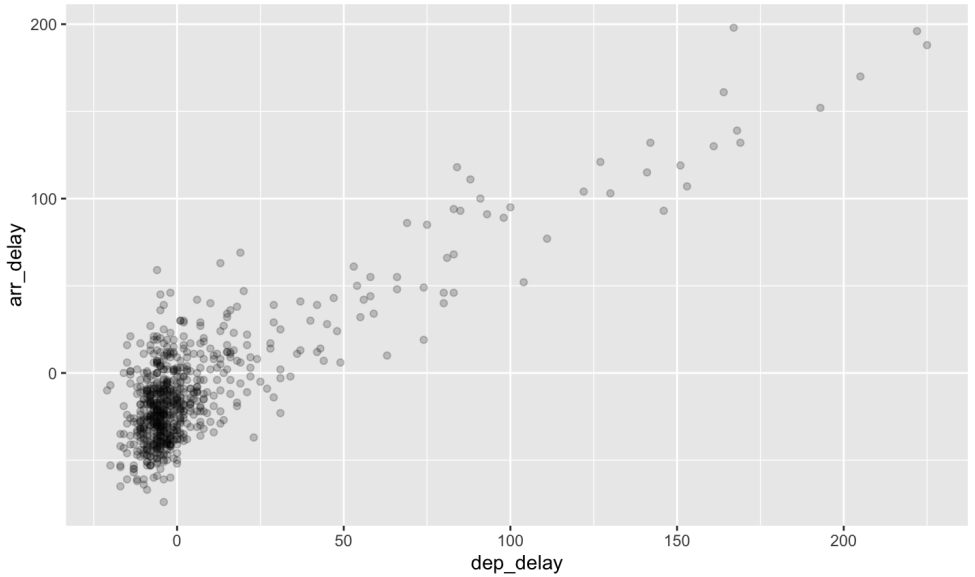
```
ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay)) +
  geom_point()
```



Scatterplots

Let's take a moment to clean things up & learn some new tricks. Specifically, let's deal with the **overplotting** at 0, 0. First: we can try changing the transparency of points.

```
ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay)) +
  geom_point(alpha = 0.2)
```

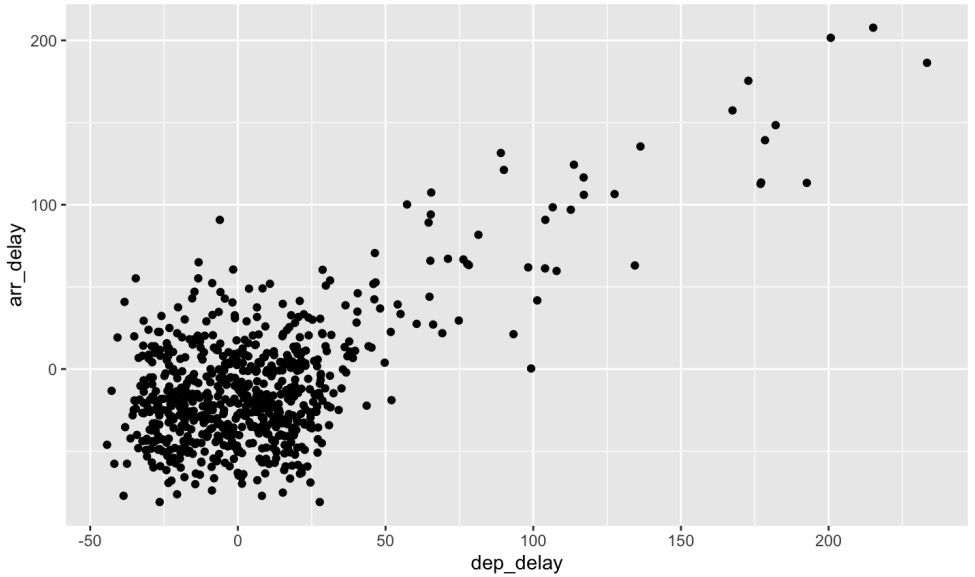


Note that transparency is *cumulative*, which is useful as it adds information.

Scatterplots

Alternatively, we can 'jitter' the points to randomly nudge them away from one another & create some space. Let's replace our `geom_point()` with `geom_jitter()`.

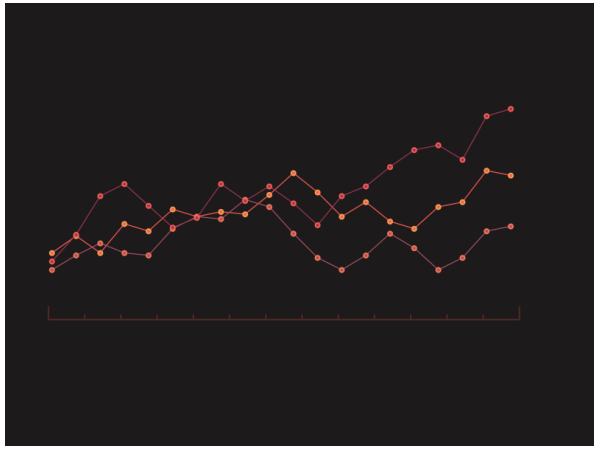
```
ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay)) +
  geom_jitter(width = 30, height = 30)
```



Linegraphs

Linegraphs

For viewing the relationship between numerical variables when the x is *ordered*



- Most commonly used to view temporal relationship (x = minutes, hours, years etc.)
- via `geom_line()`

Linegraphs

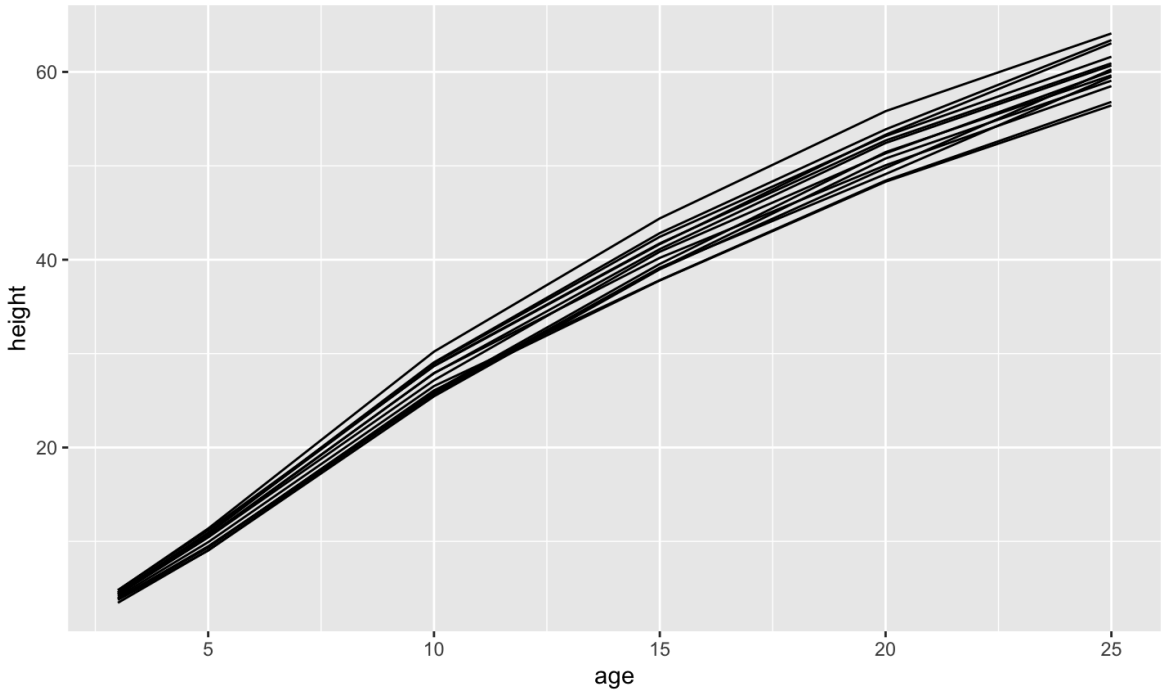
Let's take a look at some data on the growth rates of Loblolly pine trees via via the `Loblolly` dataset.

```
head(Loblolly)
```

```
##    height age Seed
## 1    4.51   3  301
## 15  10.89   5  301
## 29  28.72  10  301
## 43  41.74  15  301
## 57  52.70  20  301
## 71  60.92  25  301
```

Linegraphs

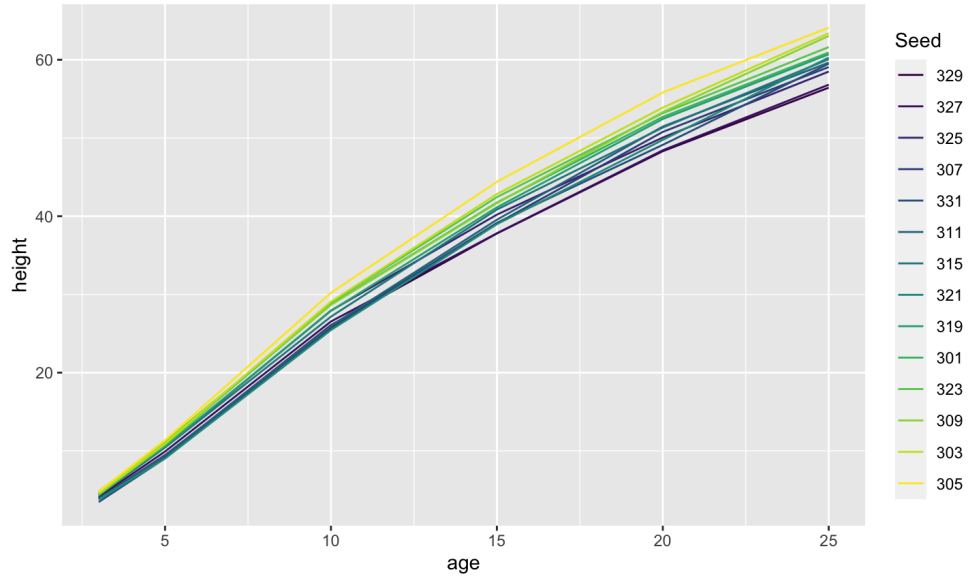
```
ggplot(data = Loblolly, mapping = aes(x = age, y = height, group = Seed)) +
       geom_line()
```



What if we want to visually separate those trajectories a bit? We could consider using `colour` instead of `group`:

Linegraphs

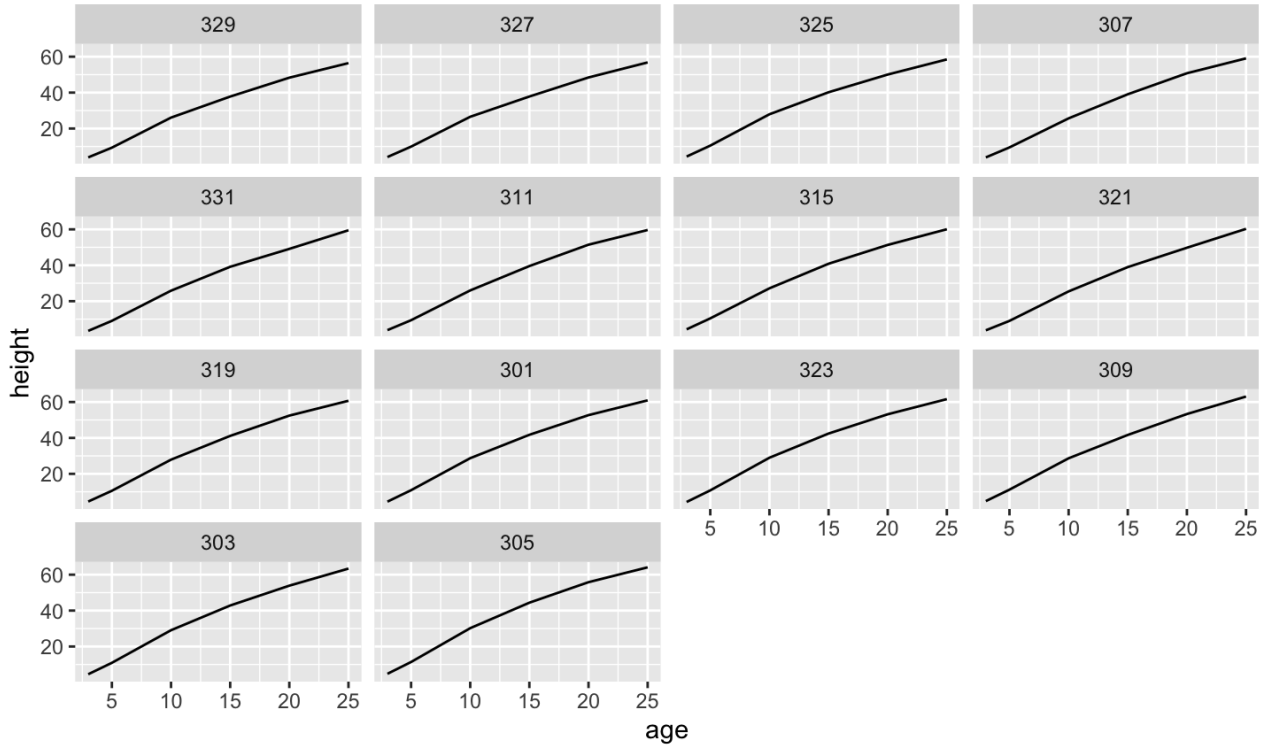
```
ggplot(data = Loblolly, mapping = aes(x = age, y = height, colour = Seed)) +
       geom_line()
```



We could also consider using `facet_wrap()`, which *facets* the plot based on a variable that we specify (here, `Seed`).

Linegraphs

```
ggplot(data = Loblolly, mapping = aes(x = age, y = height)) +
       geom_line() +
       facet_wrap(~Seed)
```



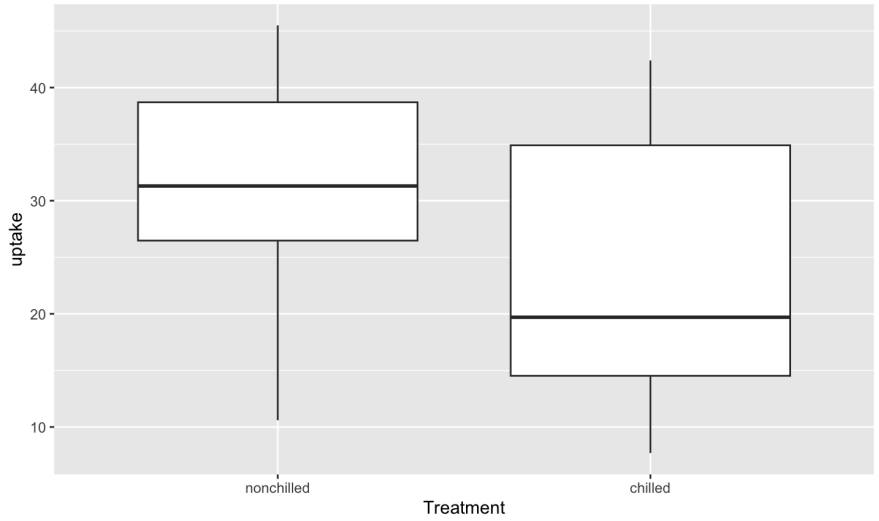
Hot plot tips 2

- `geom_jitter()` can replace `geom_point()` to create some visual separation between points
- Grouping structure in the data can be specified by both `group` and `colour` aesthetic arguments, to different ends
- `facet_wrap()` is a handy layer for splitting up groups into their constituent elements

Boxplots

Boxplots

For representing distribution of a numerical variable split by the values of another variable.



- Constructed from the information provided by a five-number summary

Boxplots

Let's build one from scratch using CO2 data:

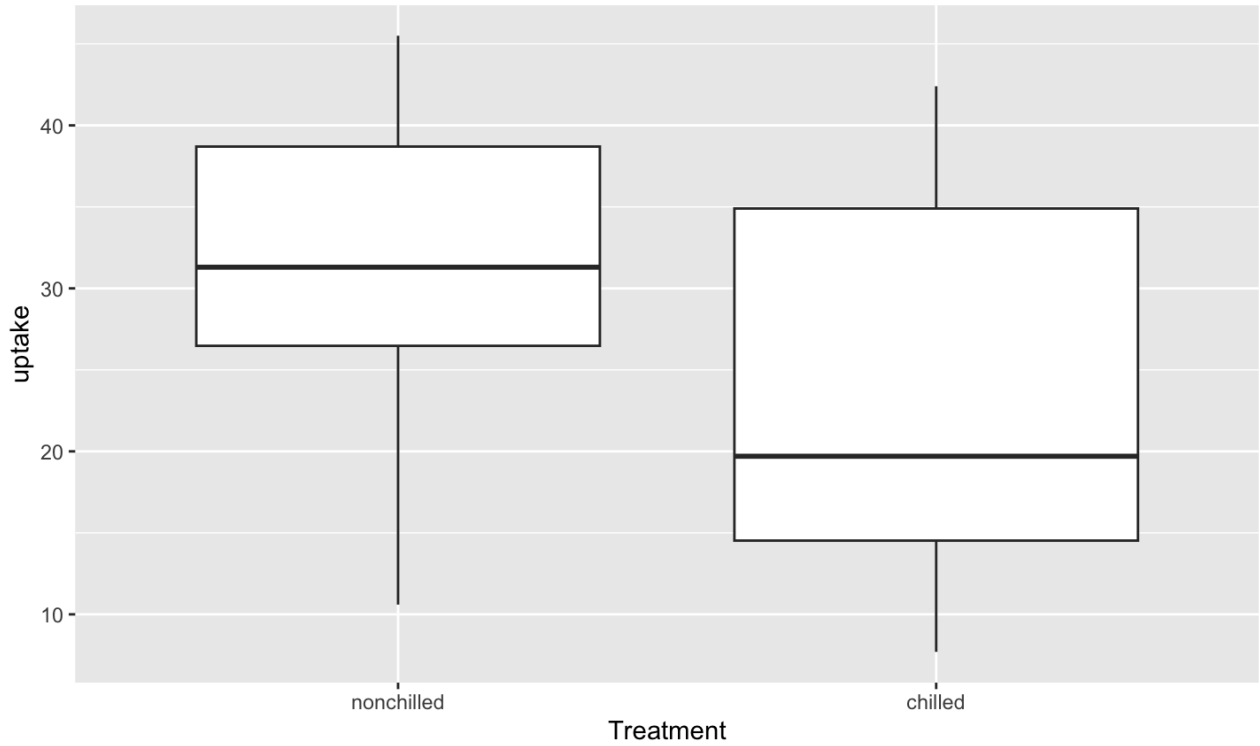
```
head(CO2)
```

```
##   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
```

Boxplots

Let’s build one from scratch using CO2 data:

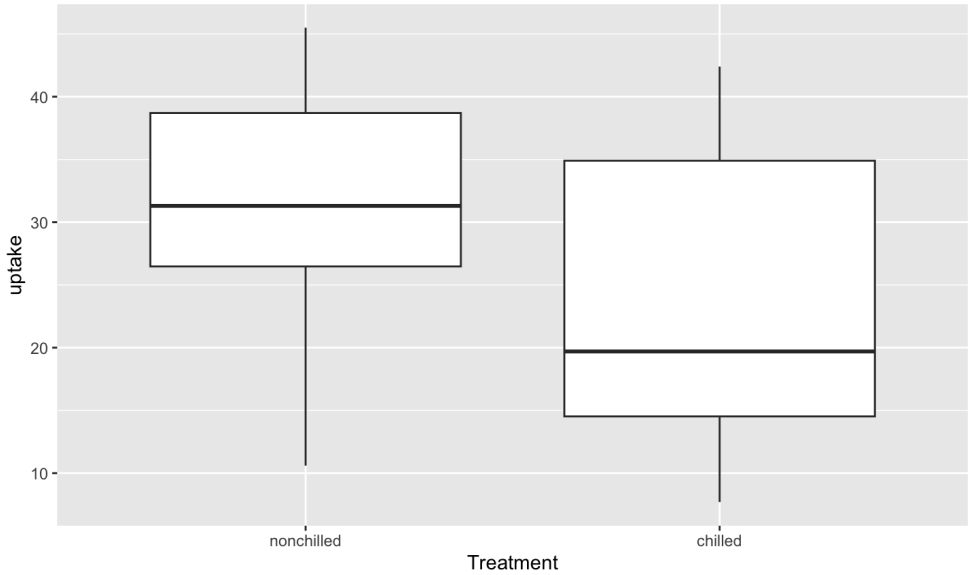
```
ggplot(data = CO2, mapping = aes(x = Treatment, y = uptake)) +
       geom_boxplot()
```



Boxplots

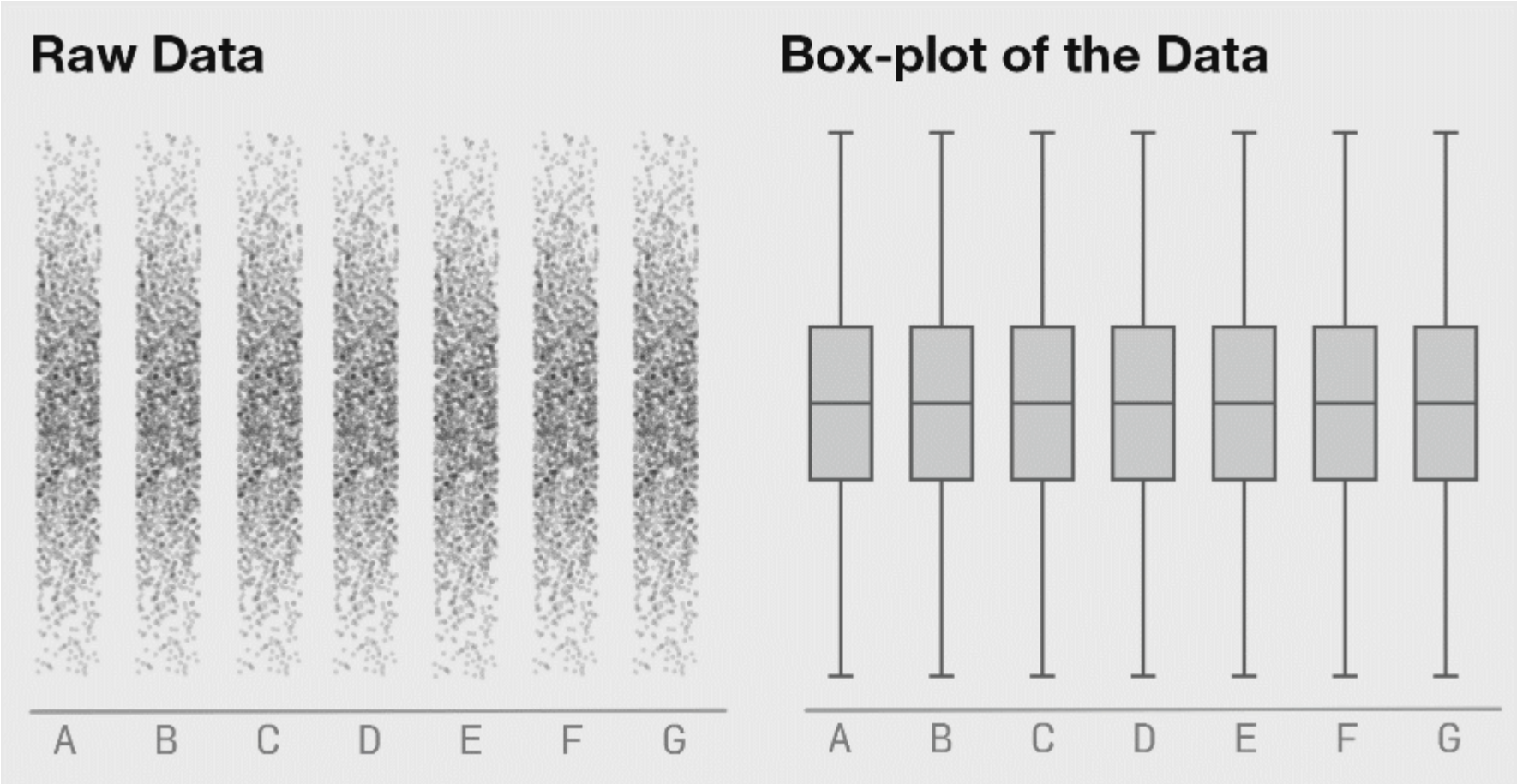
Let's build one from scratch using CO2 data:

```
ggplot(data = CO2, mapping = aes(x = Treatment, y = uptake)) +
       geom_boxplot()
```



Great! But eh, it's a bit ugly, and it's also defying one of our **basic principles**. Let's start wandering further down the customisation rabbit-hole.

Boxplots

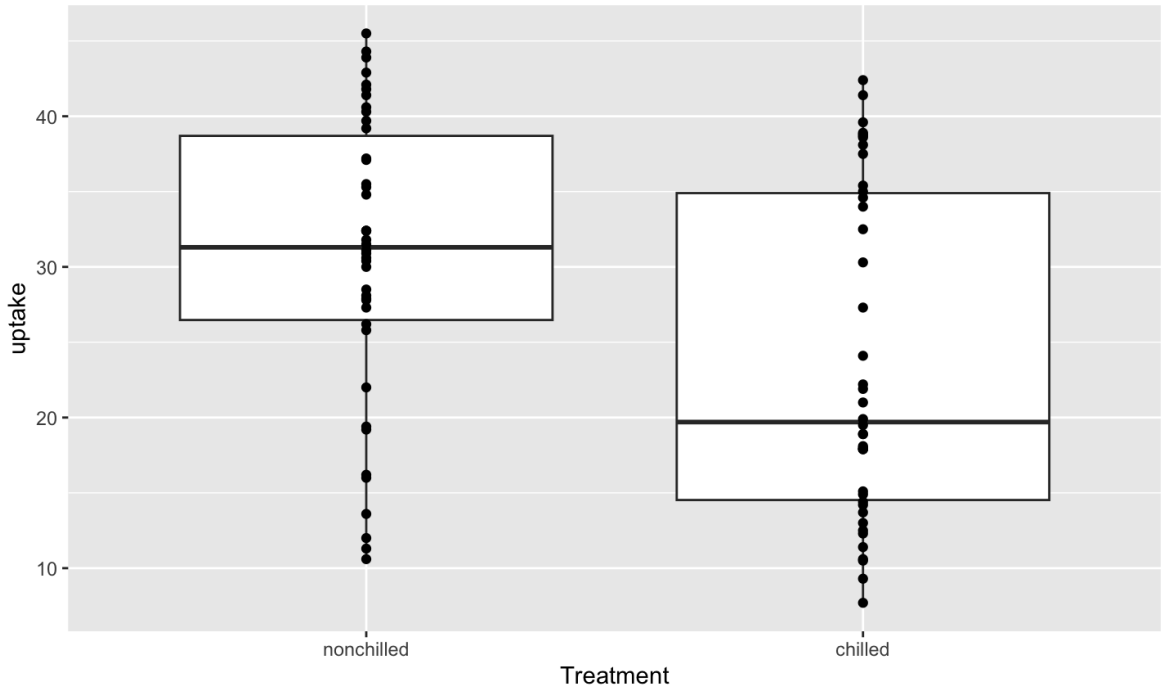


Boxplots

Let’s try combining multiple `geoms` for a more information-rich plot.

Boxplots

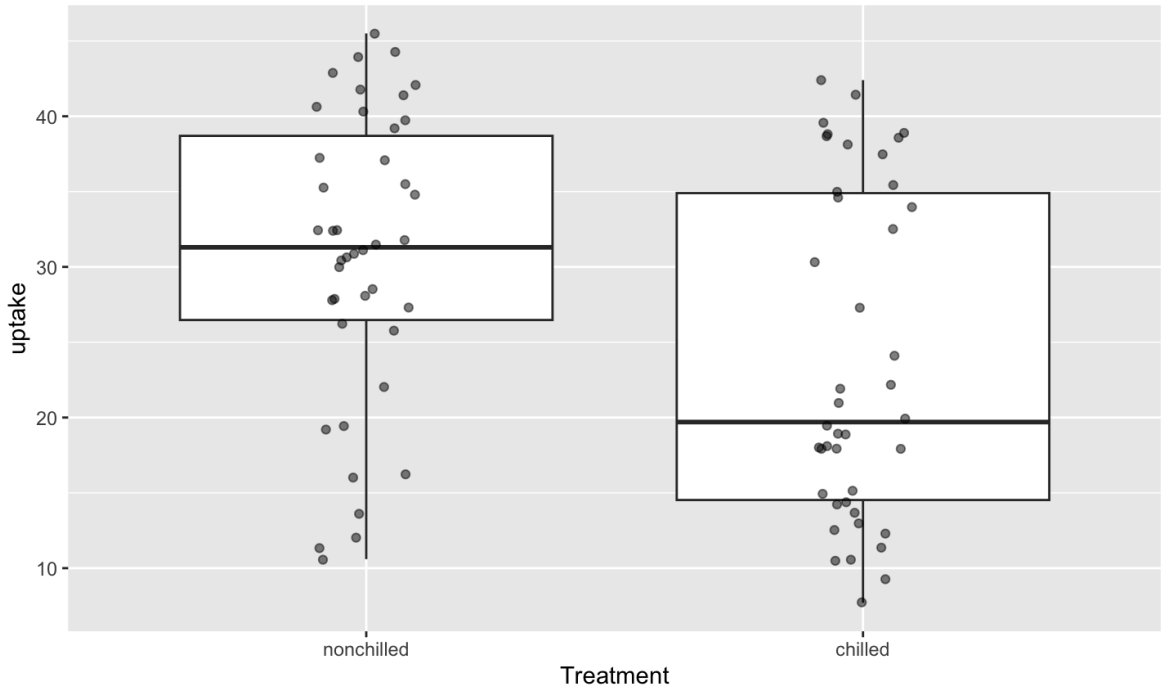
```
ggplot(data = CO2, mapping = aes(x = Treatment, y = uptake)) +
       geom_boxplot() +
       geom_point()
```



Can we do better?

Boxplots

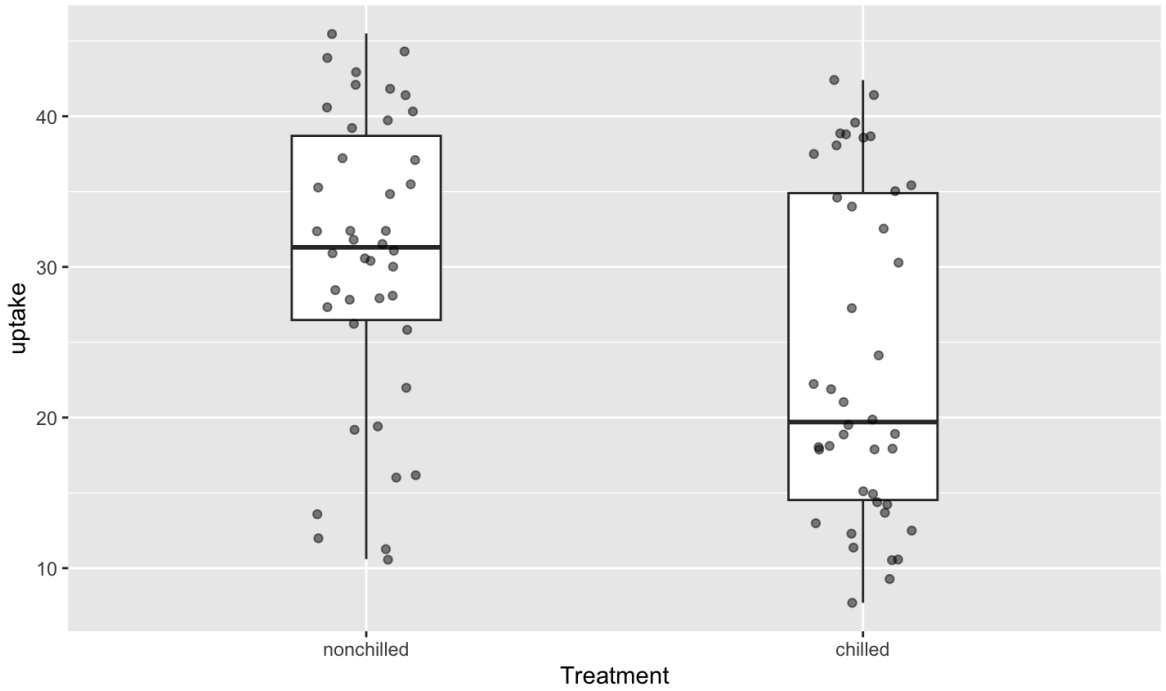
```
ggplot(data = CO2, mapping = aes(x = Treatment, y = uptake)) +
       geom_boxplot() +
       geom_jitter(width = 0.1, alpha = 0.5)
```



What about those ugly wide boxes?

Boxplots

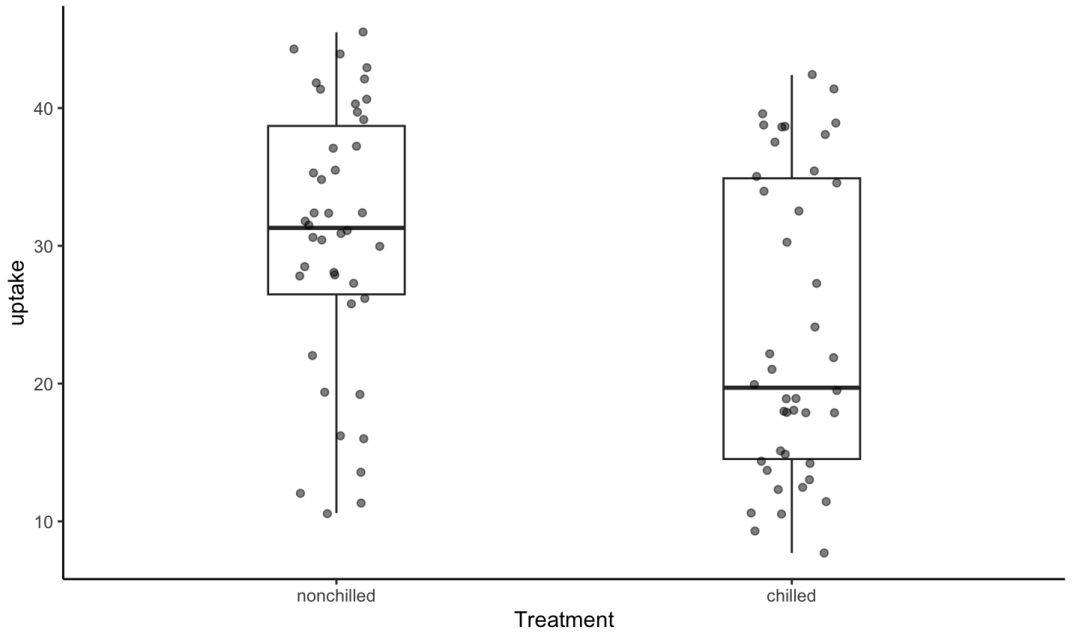
```
ggplot(data = CO2, mapping = aes(x = Treatment, y = uptake)) +
       geom_boxplot(width = 0.3) +
       geom_jitter(width = 0.1, alpha = 0.5)
```



Nice! And now for another new trick, can we fix that noisy background?

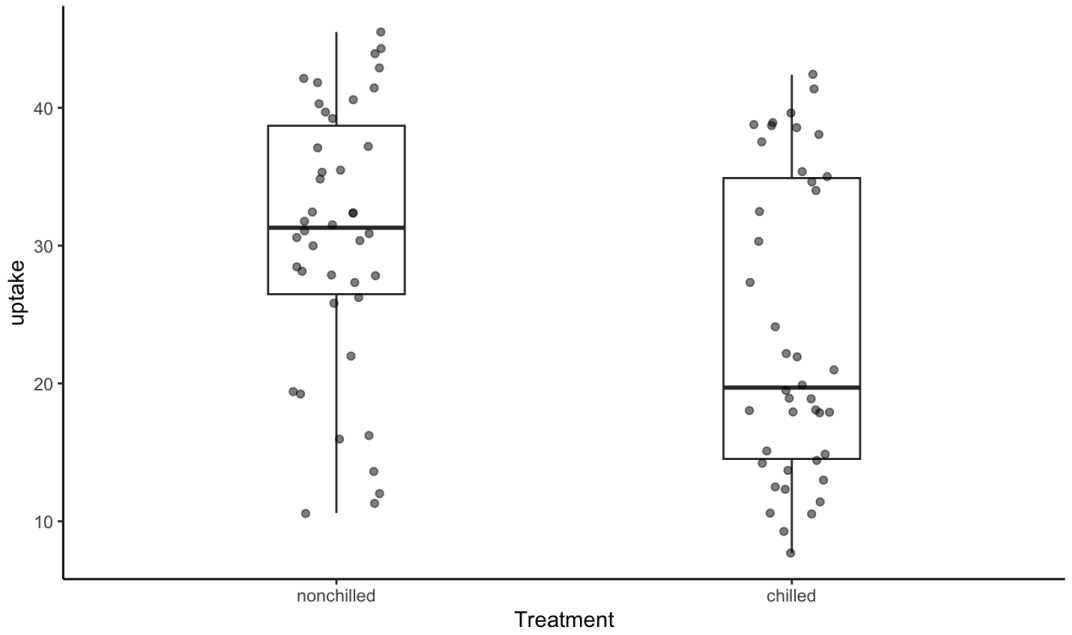
Boxplots

```
ggplot(data = CO2, mapping = aes(x = Treatment, y = uptake)) +
       geom_boxplot(width = 0.3) +
       geom_jitter(width = 0.1, alpha = 0.5) +
       theme_classic()
```



Boxplots

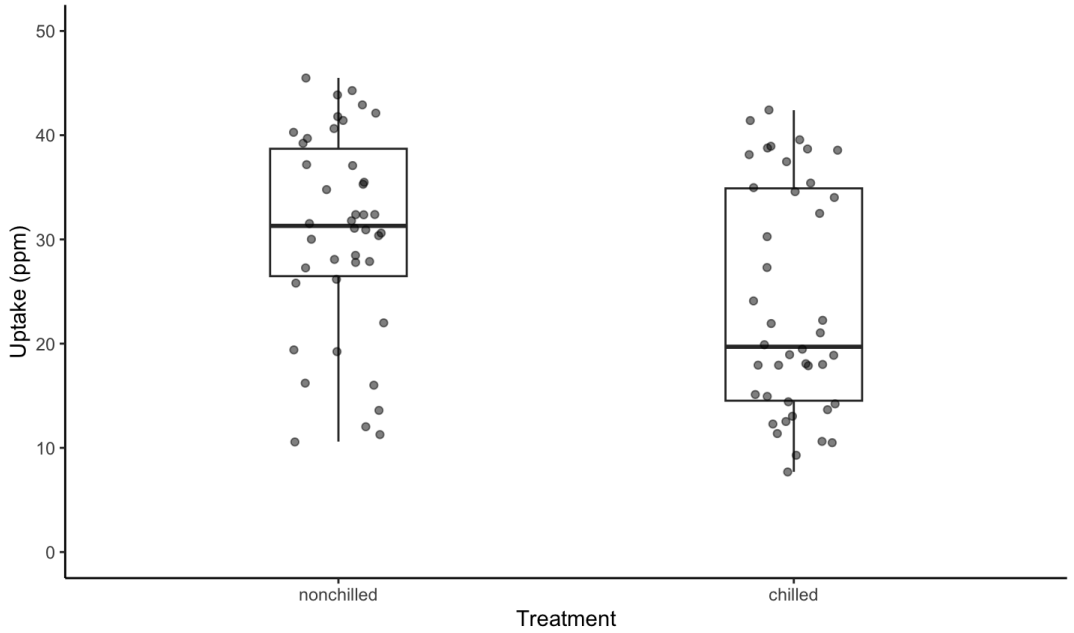
```
ggplot(data = CO2, mapping = aes(x = Treatment, y = uptake)) +
       geom_boxplot(width = 0.3) +
       geom_jitter(width = 0.1, alpha = 0.5) +
       theme_classic()
```



There are still **two things** that bug me about this plot.

Boxplots

```
ggplot(data = CO2, mapping = aes(x = Treatment, y = uptake)) +
       geom_boxplot(width = 0.3) +
       geom_jitter(width = 0.1, alpha = 0.5) +
       ylim(0, 50) +
       ylab('Uptake (ppm)') +
       theme_classic()
```



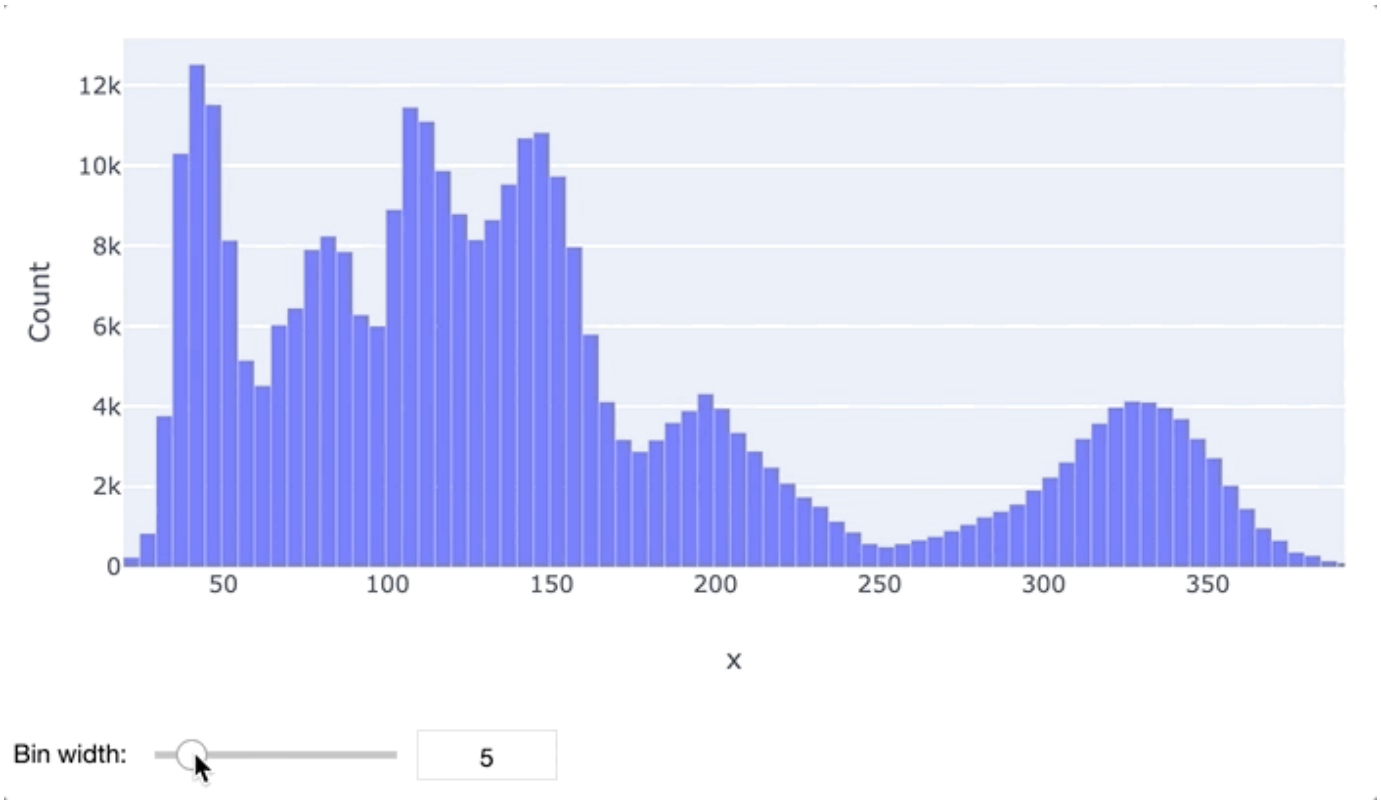
Hot plot tips 3

- `xlim()` and `ylim()` allow simple control of axis limits (there are also more fine-grained methods)
- `xlab()` and `ylab()` do the same for axis labels
- Use pre-built `theme_` to exercise fine control over the non-data elements on your plot

Histograms

Histograms

Valuable for examining the distribution of *continuous* values



Histograms

Valuable for examining the distribution of *continuous* values

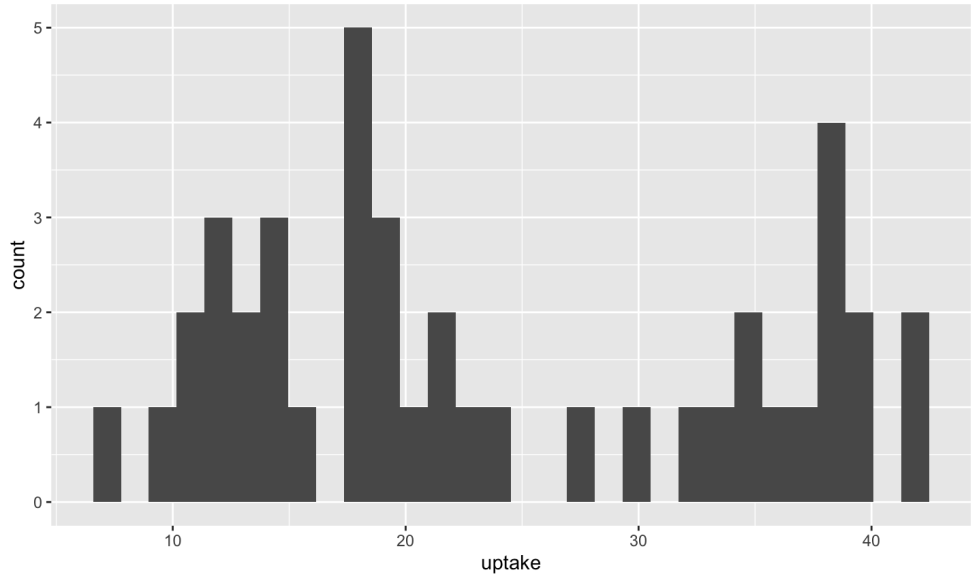
Powerful diagnostic tool:

- What are the smallest and largest values?
- What is the “center” or “most typical” value?
- How do the values spread out?
- What are frequent and infrequent values?
- via `geom_histogram()`

Histograms

Let’s take a look at our CO2 data again. We’ll just begin with one of our treatments to get a feel for it.

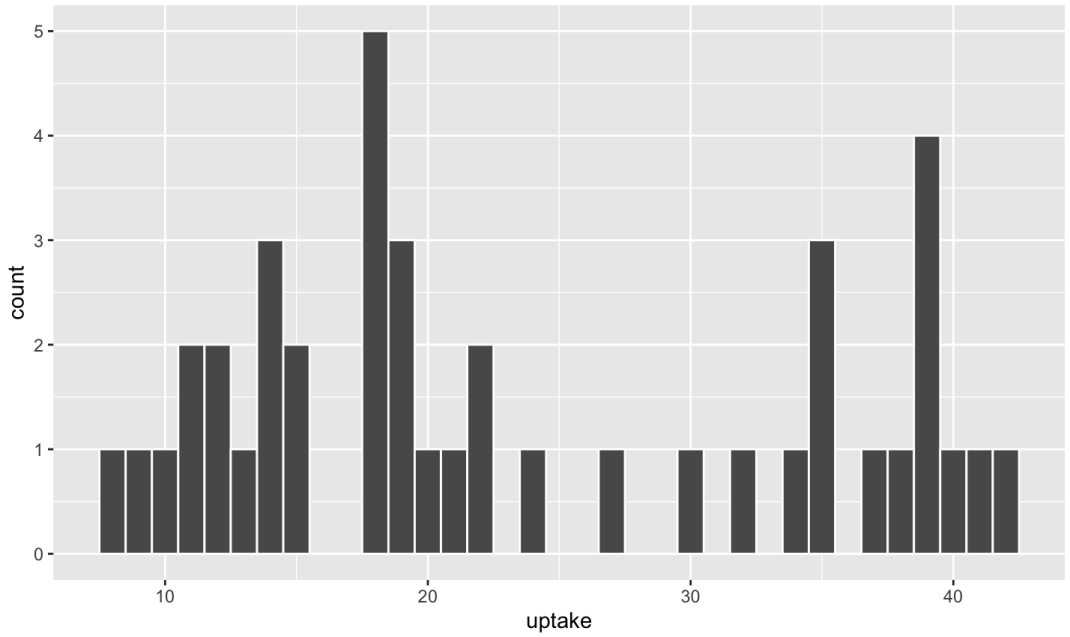
```
ggplot(data = filter(CO2, Treatment == 'chilled'), mapping = aes(x = uptake)) +
  geom_histogram()
```



Histograms

As before, it's a little ugly and hard to see, but there are simple tweaks we can make

```
ggplot(data = filter(CO2, Treatment == 'chilled'), mapping = aes(x = uptake)) +
  geom_histogram(binwidth = 1, color = "white")
```

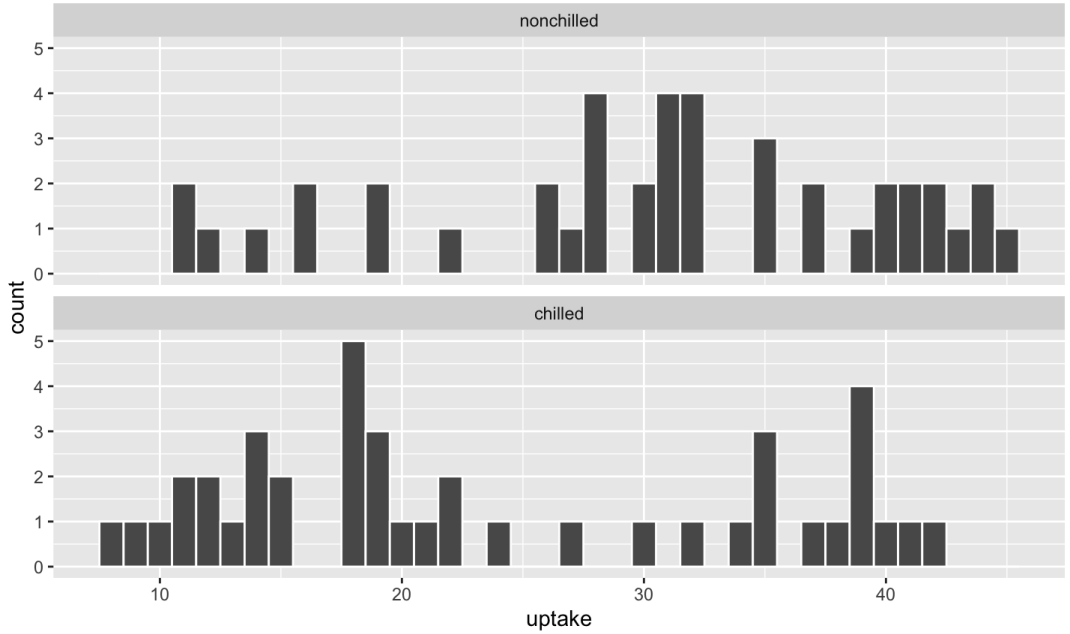


Here *binwidth* and *color* make for an easier, more information-rich plot.

Histograms

We have more than one treatment though! Now what?

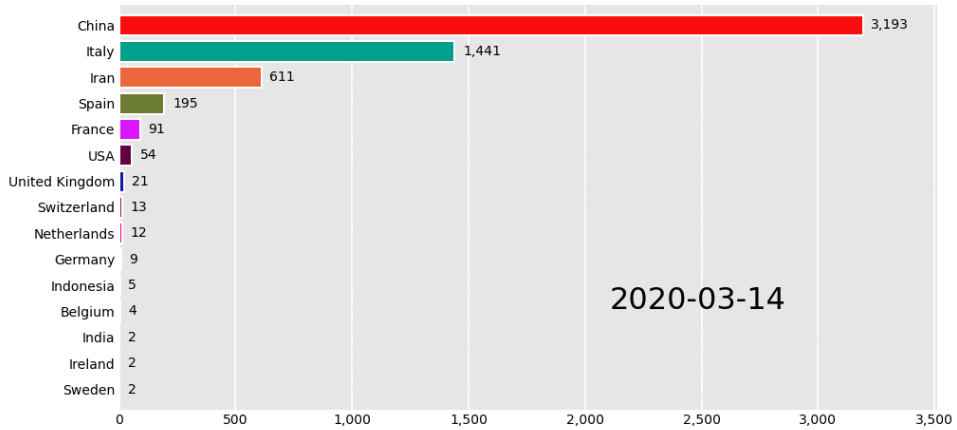
```
ggplot(data = CO2, mapping = aes(x = uptake)) +
 geom_histogram(binwidth = 1, color = "white") +
        facet_wrap(~Treatment, nrow = 2)
```



Barplots

Barplots

For visualising the distribution of *categorical* variables



- via `geom_bar()` OR `geom_col`, depending on the data
- Most common for simple counts or proportions
- Often improperly used where more information-rich plots would be better (e.g. boxplots)

Barplots

Use `geom_col()` when data are ‘pre-counted’, otherwise `geom_bar()`. Let’s try both.

Barplots

Raw:

```
insects
```

```
## # A tibble: 5 × 1
##   order
##   <chr>
## 1 Lepidoptera
## 2 Lepidoptera
## 3 Orthoptera
## 4 Orthoptera
## 5 Hymenoptera
```

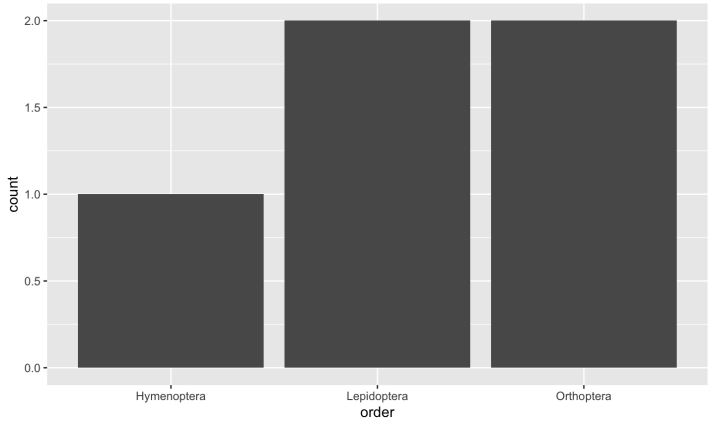
Pre-counted:

```
insects_counted
```

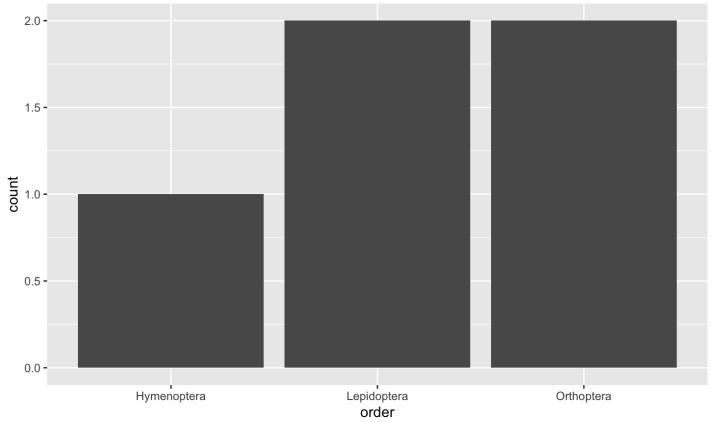
```
## # A tibble: 3 × 2
##   order       count
##   <chr>       <dbl>
## 1 Lepidoptera     2
## 2 Orthoptera      2
## 3 Hymenoptera     1
```

Barplots

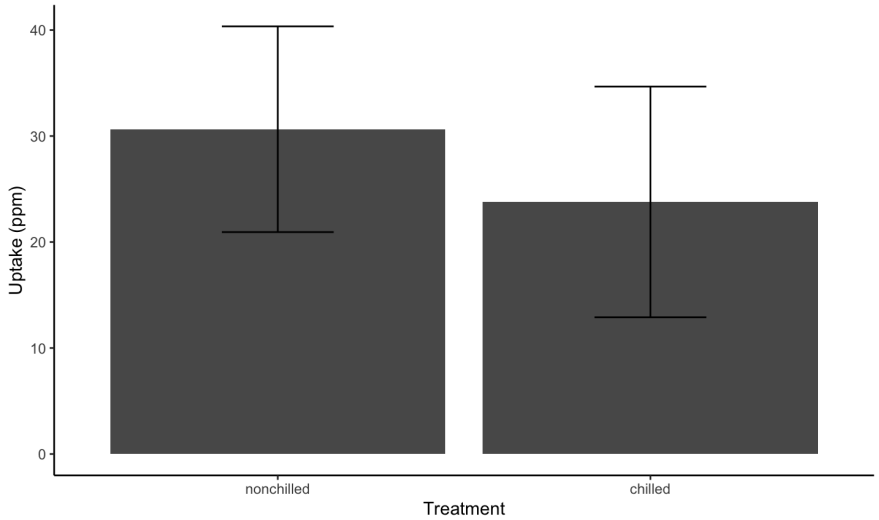
```
ggplot(insects,
       aes(x = order)) +
       geom_bar()
```

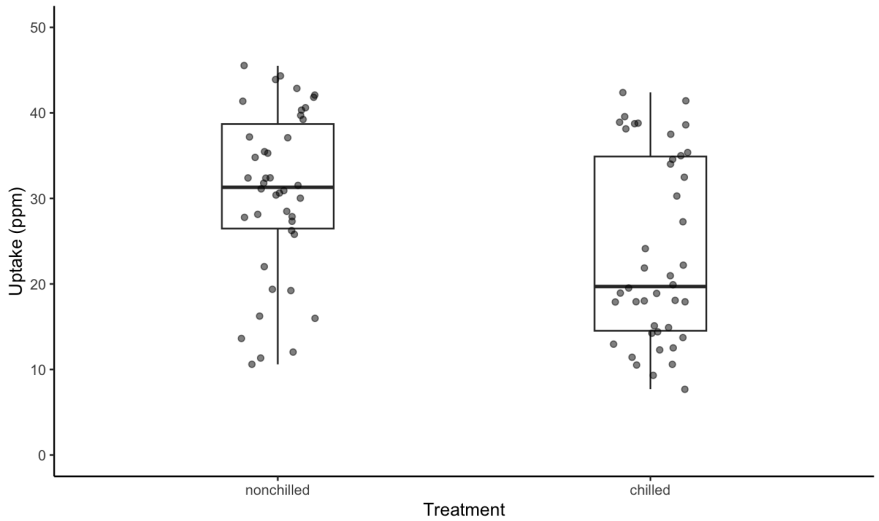



```
ggplot(insects_counted,
       aes(x = order, y = count)) +
       geom_col()
```



Bars vs Boxes





Final tricks

Composing figures with `patchwork`

`library(patchwork)` makes it simple to combine multiple plots into multi-panel figures

1. Assign your plots to objects with `<-`
2. Use `+` and `/` to combine them. `+` combines horizontally, `/` vertically, and `()` for more complex arrangements.

Composing figures with `patchwork`

1. Assign your plots to objects with <-

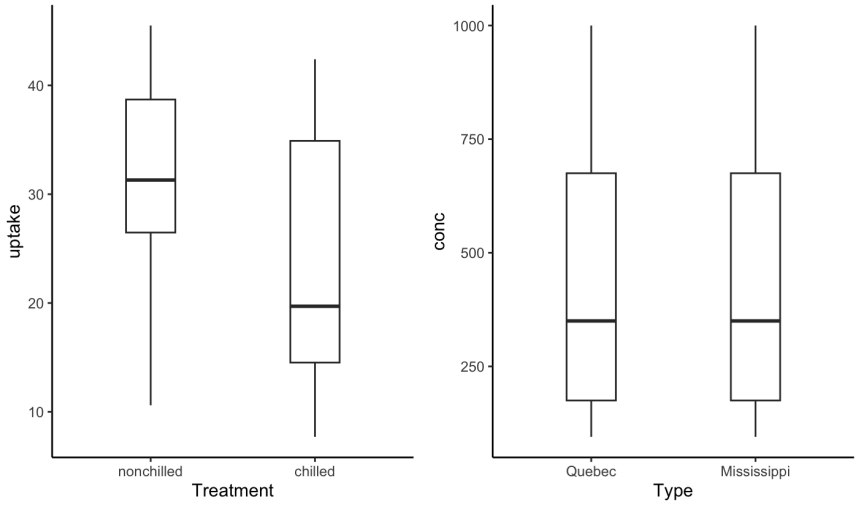
```
plot_1 <-
  ggplot(data = CO2, mapping = aes(x = Treatment, y = uptake)) +
         geom_boxplot(width = 0.3) +
         theme_classic()

plot_2 <-
  ggplot(data = CO2, mapping = aes(x = Type, y = conc)) +
         geom_boxplot(width = 0.3) +
         theme_classic()
```

Composing figures with `patchwork`

1. Use `+` and `/` to combine them. `+` combines horizontally, `/` vertically, and `()` for more complex arrangements.

```
plot_1 + plot_2
```



Exporting plots with `ggsave()`

`ggsave()` saves your ggplot2s, and makes some sane guesses about what you want. By default, it'll save the last plot your displayed, or a saved plot can be specified.

- `ggsave('my_barplot.png')`: creates a local `.png` file called `my_barplot` with your last plot in it
- `ggsave('my_barplot.jpg', insect_barplot)`: creates a `.jpg` file called `my_barplot`, containing a plot saved in the object `insect_barplot`
- `ggsave('my_barplot.tiff', width = 10, height = 10, units = 'cm')`: creates a 10 cm x 10 cm plot in `.tiff` format, with your last plot in it.
- Etc. See `?ggsave` for options.

Outline

- Articulate the principles of effective data visualisation
- Understand the foundational place of the ‘big 5’ graphics for datavis
- Use `ggplot2` to create and modify common forms of data visualisation

Thanks!