

Summarising and Plotting Data in R

Visualising Data

Glenn Williams

University of Sunderland

2020-09-29 (updated: 2020-10-16)

Data Visualisation

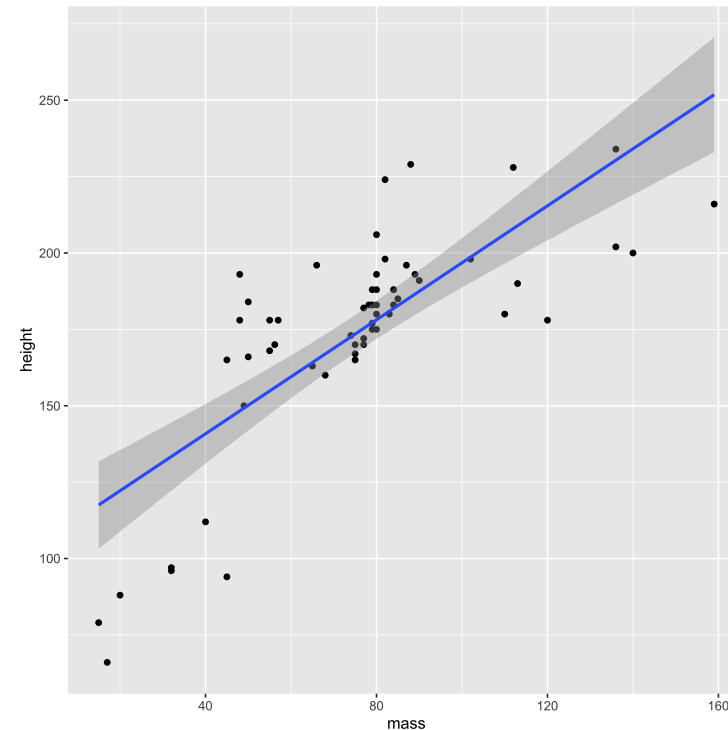
What's a better way to summarise the relationship between height and mass in our star wars data set?

Note below I excluded Jabba the Hutt who is **short and massive**.

We could report a correlation and some averages, though that's going to be hard to picture.

```
## # A tibble: 31 x 3
##   species  mean_height mean_weight
##   <chr>      <dbl>      <dbl>
## 1 Aleena         79         15
## 2 Besalisk       198        102
## 3 Cerean         198         82
## 4 Clawdite       168         55
## # ... with 27 more rows
```

So plots do a lot of the work.



A Grammar of Graphics

R has an inbuilt (base) plotting system, but we'll use **ggplot2** from the tidyverse.

ggplot2 has a consistent method of building up plots layer by layer.

We have to:

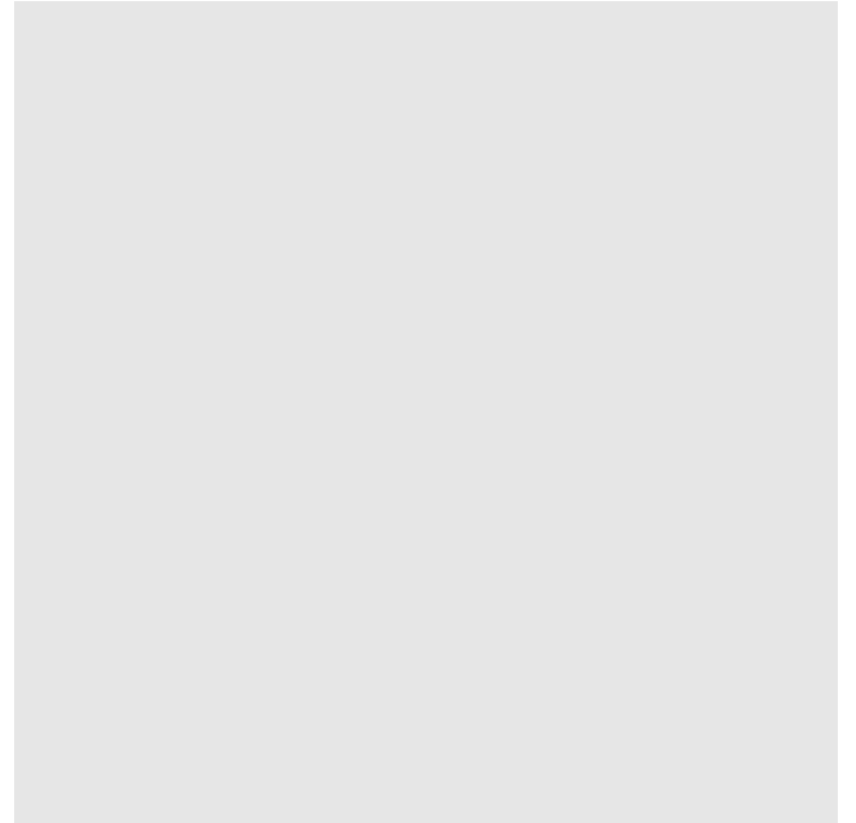
- Define how we want to **map** the data onto the plot.
- Define the **geometric** shapes we want to use to draw the data.
- Define any **aesthetics** for the look for our plot.

Grammar of Graphics with ggplot2

As with all R functions, we need to pass arguments to ggplot2.
Let's start with our data:

```
ggplot(data = starwars)
```

That's not too exciting, but we can see we've made the beginnings of a plot (the background).



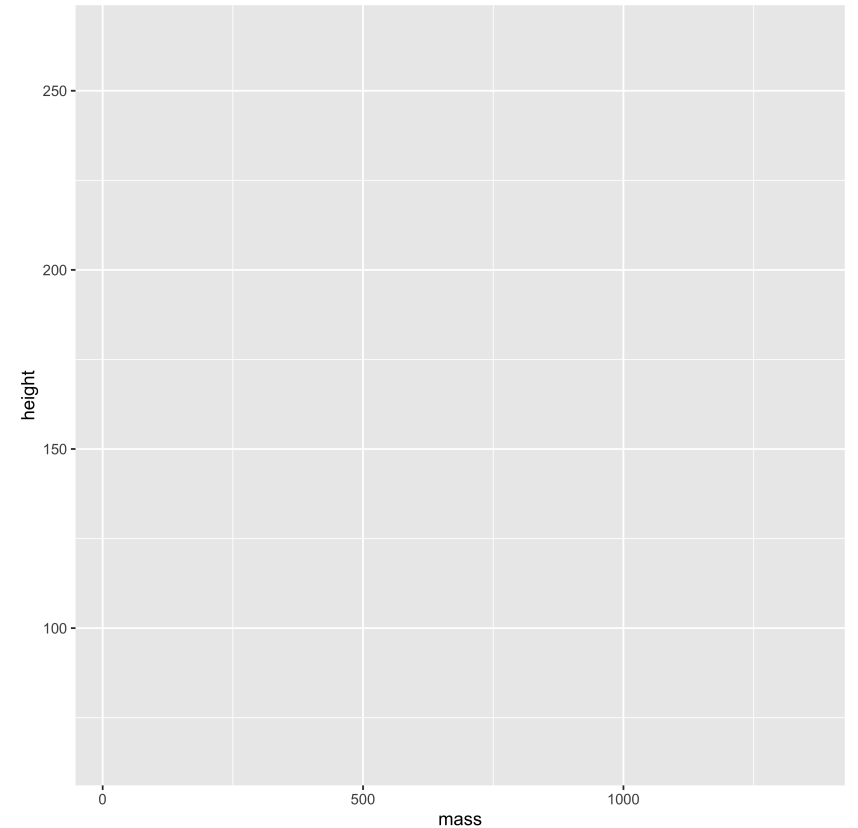
Mapping our Data

Then we can define how we want to map data onto aesthetics.

Here, we tell R to map the mass and height data from starwars onto the X and Y axes respectively.

```
ggplot(  
  data = starwars,  
  mapping = aes(x = mass, y = height)  
)
```

That looks a little better. Now we know the ranges of our data!



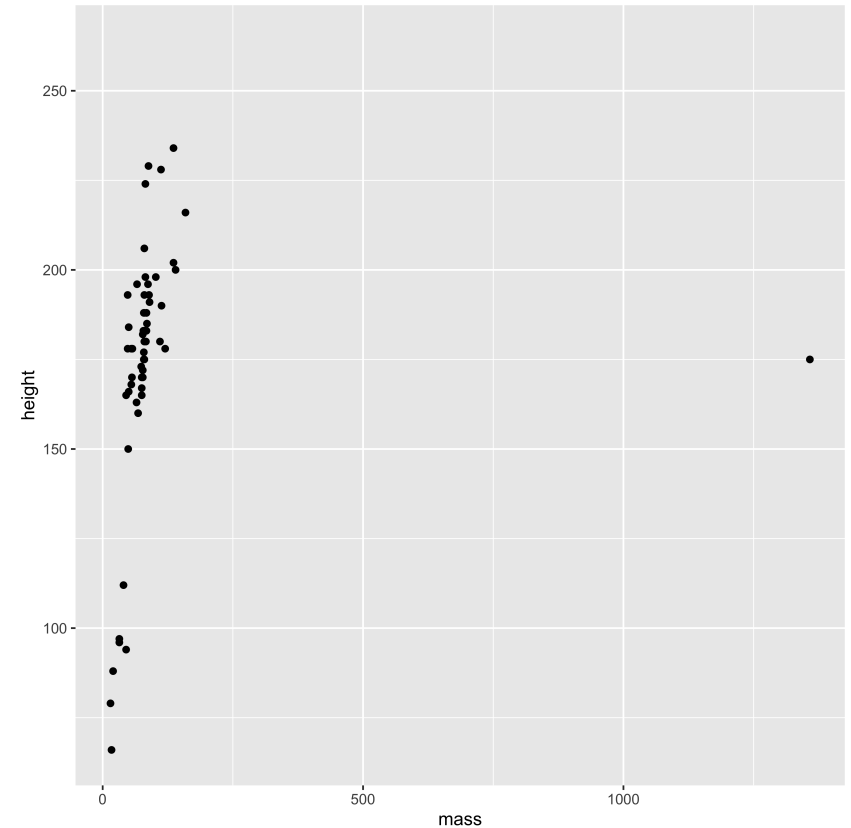
Adding Geoms

Next, we can tell R how we want to present the data on screen in a new **layer**.

We do this by defining the **geometric shapes** of the data, or geoms.

```
ggplot(  
  data = starwars,  
  mapping = aes(x = mass, y = height)  
) +  
  geom_point()
```

Now we can see how mass and height relate to one another.

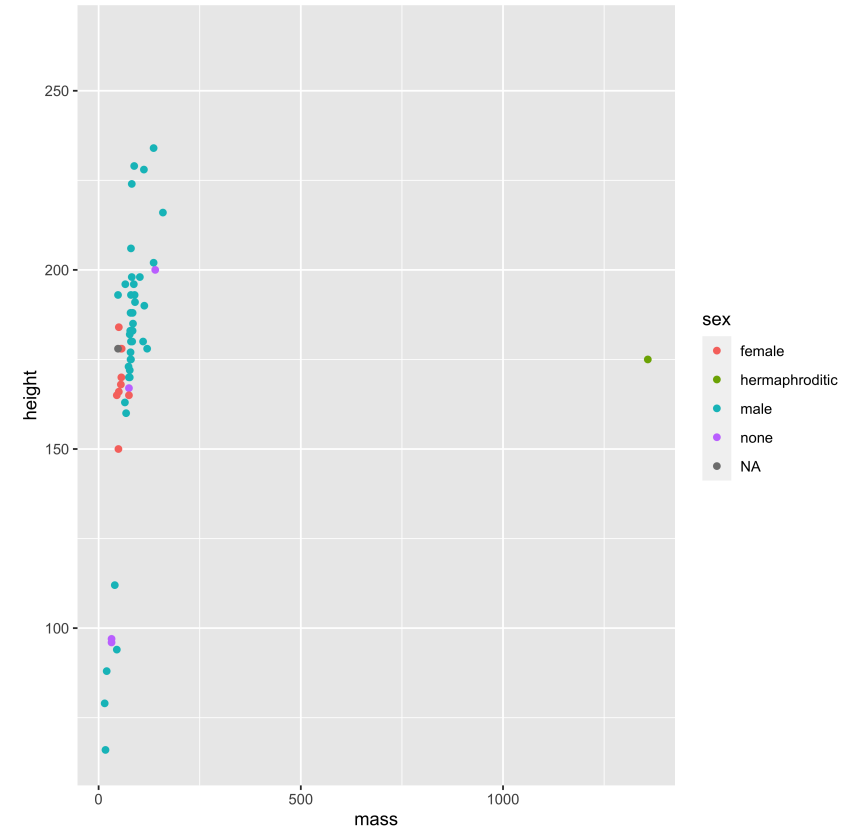


Differentiating Data

We can add additional arguments to our mappings, for example **mapping colour onto sex**.

```
ggplot(  
  data = starwars,  
  mapping = aes(x = mass, y = height, colour = sex)  
) +  
  geom_point()
```

As you can see, we get different groups of data depending upon the sex of the individual.

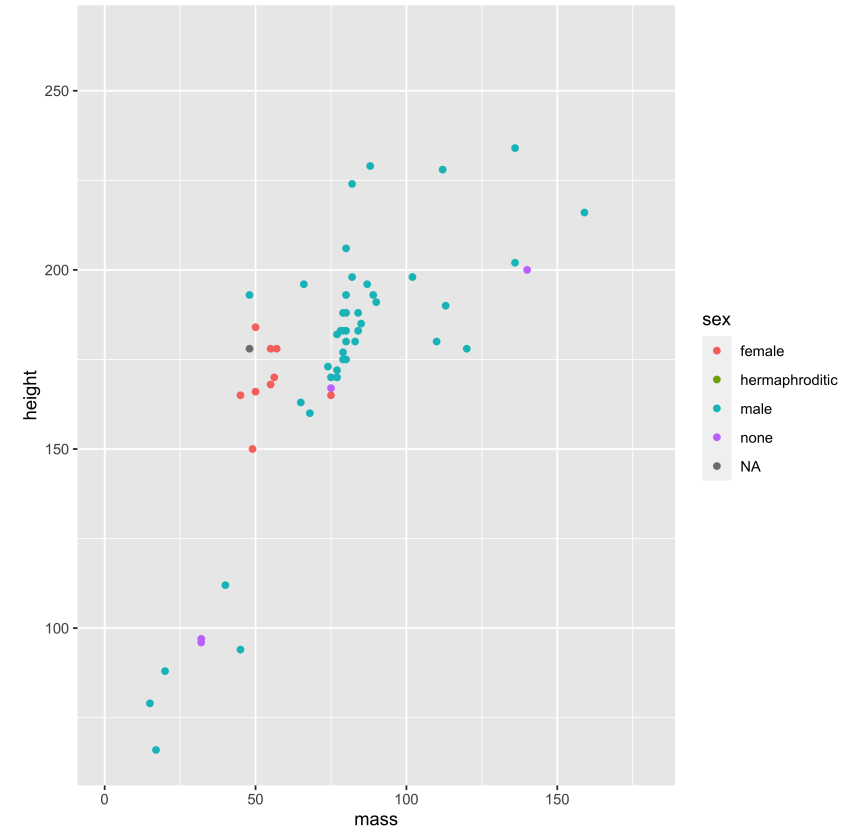


Filtering and Zooming

It's difficult to see the pattern between height and mass due to an extreme outlier.

We can exclude this individual either by (a) Filtering observations first, (b) restricting limits on the axis of the plot using, e.g. `xlim()` in `coord_cartesian()`.

```
ggplot(  
  data = starwars,  
  mapping = aes(x = mass, y = height, colour = sex)  
) +  
  geom_point() +  
  coord_cartesian(xlim = c(0, 180))
```



Filtering and Zooming

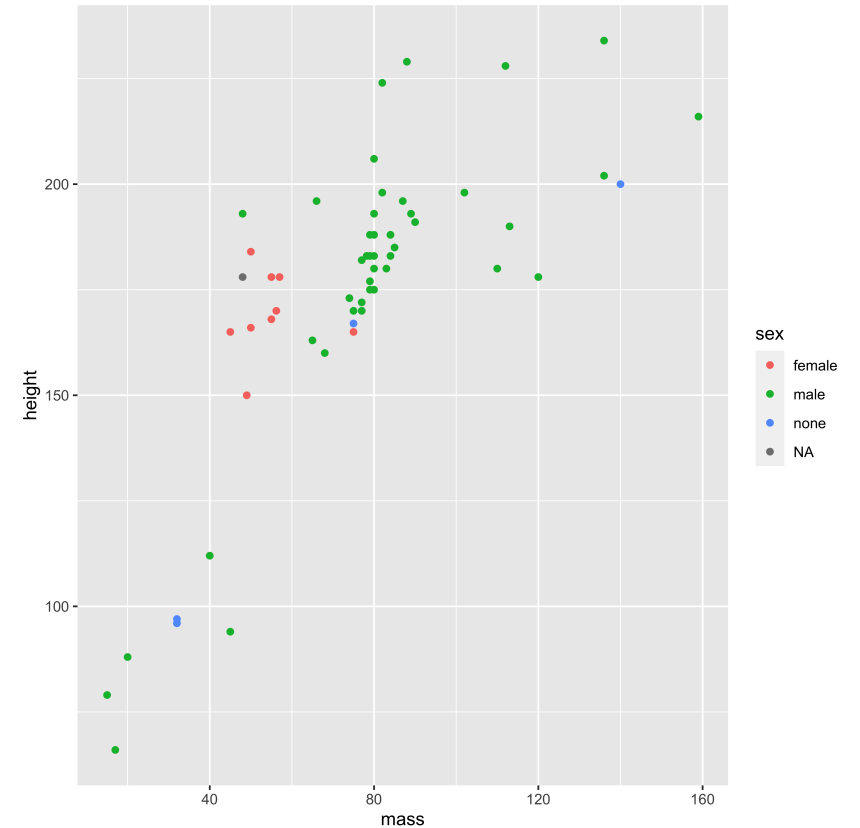
We can exclude this individual either by:

- Chaining tidyverse functions together. Here, we'll filter Jabba from the data before plotting.

```
starwars %>%  
  filter(mass < 180) %>%  
  ggplot(  
    mapping = aes(x = mass, y = height, colour = sex)  
  ) +  
  geom_point()
```

We pipe the data into the filter function which passes the filtered data to ggplot.

Warning: we pipe data using %>% but build plot layers with +. Don't confuse the two.



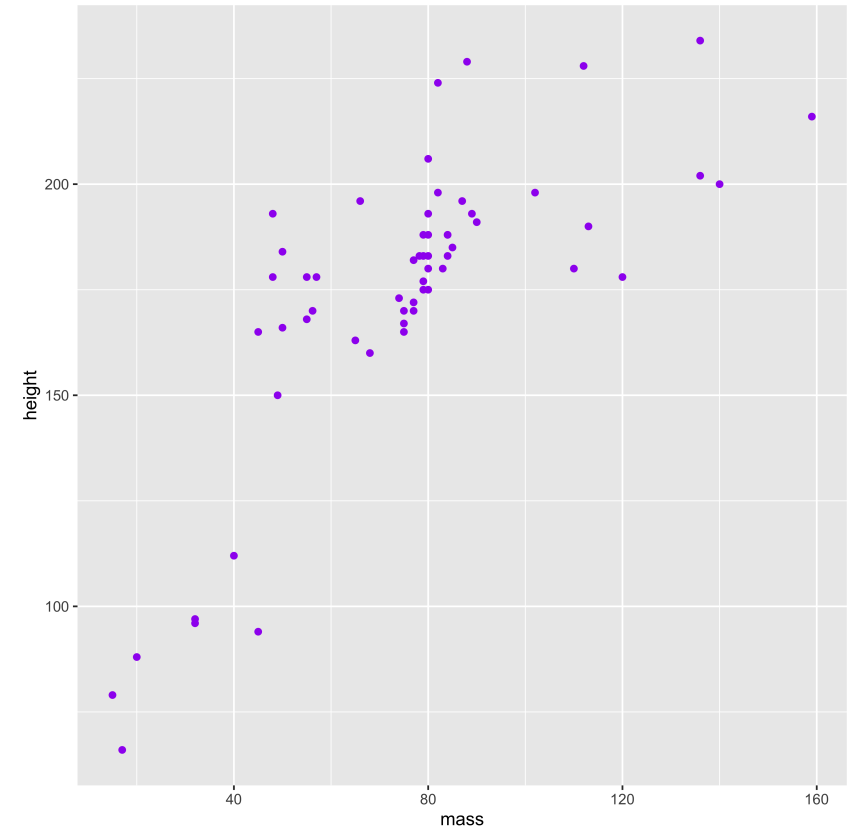
Setting Fixed Colours

What if you want all points to be the same colour? Don't map it to a group. Set it outside the mapping.

```
starwars %>%  
  filter(mass < 180) %>%  
  ggplot(mapping = aes(x = mass, y = height)) +  
  geom_point(colour = "purple")
```

Here, colour is defined in the `geom_point()` call. Colour only applies to this geometric shape.

This overrides any specific group mappings.

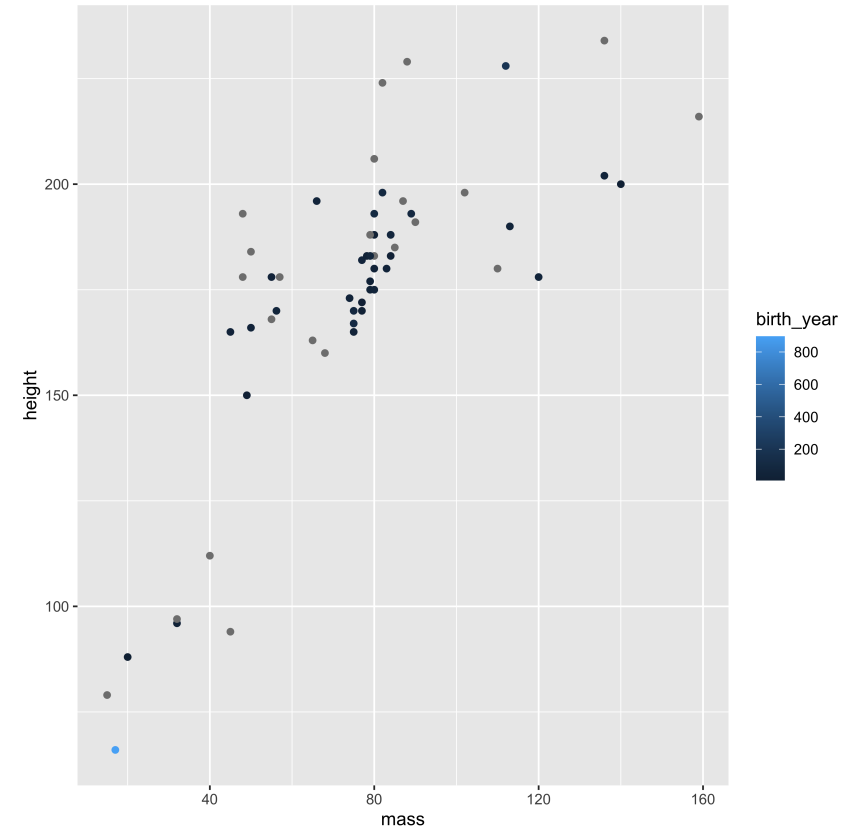


Continuous Colour

ggplot is pretty smart in how it handles colour. Categorical variables get individual colours, while scales get continuous palettes.

```
starwars %>%  
  filter(mass < 180) %>%  
  ggplot(  
    mapping = aes(  
      x = mass,  
      y = height,  
      colour = birth_year  
    )  
  ) +  
  geom_point()
```

Now the legend doesn't have categories, but a scale.

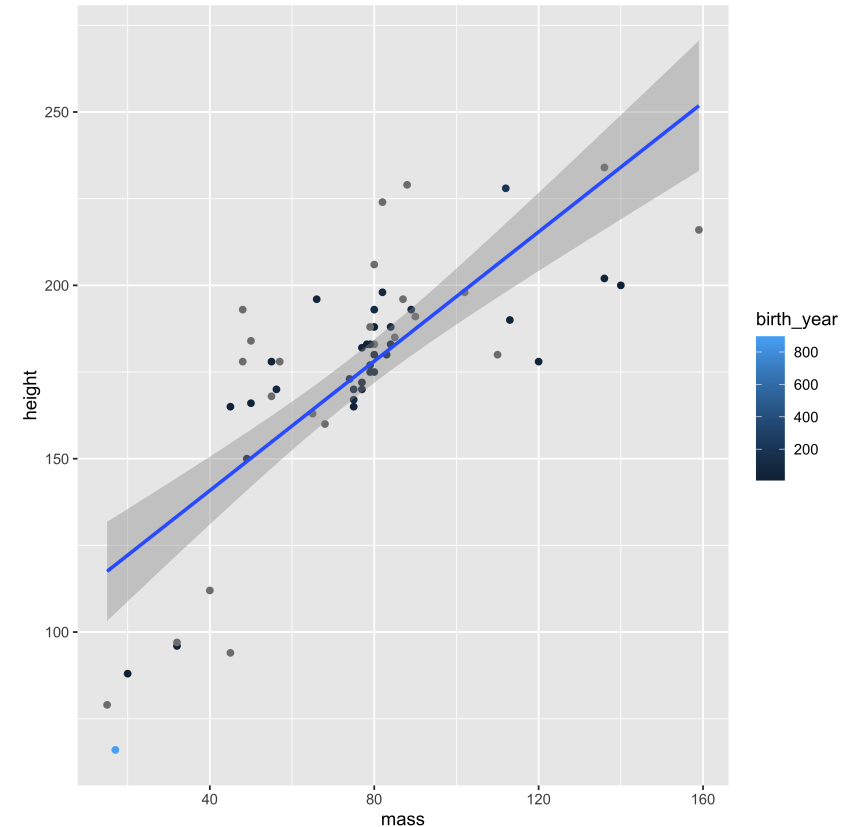


Bonus: Combining Geoms

We can add geoms like a line of best fit with 95% confidence intervals!

Without getting too into it, this is defined as a **linear model** in a **smoothing function**, hence we use `geom_smooth()`.

```
starwars %>%  
  filter(mass < 180) %>%  
  ggplot(  
    mapping = aes(  
      x = mass,  
      y = height,  
      colour = birth_year  
    )  
  ) +  
  geom_point() +  
  geom_smooth(  
    method = "lm",  
    formula = "y ~ x"  
  )
```



Recap

We've learned...

- About the importance of graphs for communicating findings.
- About a **grammar of graphics** for defining plots in code.
- How to **map** data onto a plot, how to display that data with **geometric** shapes, and how to control the **aesthetics** of the plot.
- How to build up a plot layer by layer in `ggplot2`.
- How to use ggplot functions to **differentiate data**.
- How to use ggplot functions to filter data, or how to **chain functions together** from other packages.