Lecture Notes for Statistics for Biology FS2020

Stephan Peischl

1/10/2020

Contents

Introduction

These notes should help you get started with the statistics software R, guide you through the practical part of the course and provide additional information on selected topics. Before you continue reading, make sure you have R and R Studio installed on your computer. Try to use R as an calcualtor as shown in the lecture slides and try to get yourelf acustomed to the different windows in the RStudio envrionment. No prior knwoledge about R is necessary and once everything is up and running yu can dive right in.

We start by loading the relevant packages for this course. An R package is a collection of functions, data, and documentation that extends the capabilities of base R. We are going to make use of the so-called tidyverse.

```
# make sure you have th packages installed using
# install.packages(packagename)
```

```
library(tidyverse)
library(Rmisc)
library(tibble)
library(dplyr)
```

Visualizations with ggplot2

We are going start with some examples of how to visulaize data. This should show you how easy it is to produce high quality visulaizations using R and motivate you to learn more about programming and data analysis with R. You will be learning the basic structure of a ggplot2 plot, and powerful techniques for turning data into plots.

R has several systems for making graphs, and I have chosen to start with ggplot2 beacuse it is elegant, versatile and is easy to leanr if you have no prior knowledge about programming. For those of you who are already familiar wiht programming langauges, doing graphics in "base R" may seem more intuitive initially but I hope I can convince you that the grammar of plotting used by ggplot has lots of advantages and is very readable once you get the hang of it.

If you'd like to learn more about the theoretical underpinnings of ggplot2 before you start, I'd recommend reading "The Layered Grammar of Graphics", http://vita.had.co.nz/papers/layered-grammar.pdf.

The MPG dataset

Let's use our first graph to answer a question: Do cars with big engines use more fuel than cars with small engines? What does the relationship between engine size and fuel efficiency look like? Is it positive? Negative? Linear? Nonlinear?

We can find an answer with the mpg data set found in ggplot2 (aka ggplot2::mpg). A data frame is the R data format to store data in the format of a spreadsheet: rectangular collection of variables (in the columns) and observations (in the rows). mpg contains observations collected by the US Environmental Protection Agency on 38 models of car.

```
mpg
```

```
## # A tibble: 234 x 11
##    manufacturer model displ  year   cyl trans drv     cty   hwy fl    cla~
##    <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <ch>
##  1 audi         a4      1.8  1999     4 auto~ f        18    29 p     com~
##  2 audi         a4      1.8  1999     4 manu~ f        21    29 p     com~
##  3 audi         a4      2    2008     4 manu~ f        20    31 p     com~
##  4 audi         a4      2    2008     4 auto~ f        21    30 p     com~
##  5 audi         a4      2.8  1999     6 auto~ f        16    26 p     com~
##  6 audi         a4      2.8  1999     6 manu~ f        18    26 p     com~
##  7 audi         a4      3.1  2008     6 auto~ f        18    27 p     com~
##  8 audi         a4 q~   1.8  1999     4 manu~ 4        18    26 p     com~
##  9 audi         a4 q~   1.8  1999     4 auto~ 4        16    25 p     com~
## 10 audi         a4 q~   2    2008     4 manu~ 4        20    28 p     com~
## # ... with 224 more rows
```

Among the variables in mpg are:

displ, a car's engine size, in litres.

hwy, a car's fuel efficiency on the highway, in miles per gallon (mpg). A car with a low fuel efficiency consumes more fuel than a car with a high fuel efficiency when they travel the same distance.

To learn more about mpg, open its help page by running ?mpg. in the next seciton we learn how to visualize such data.

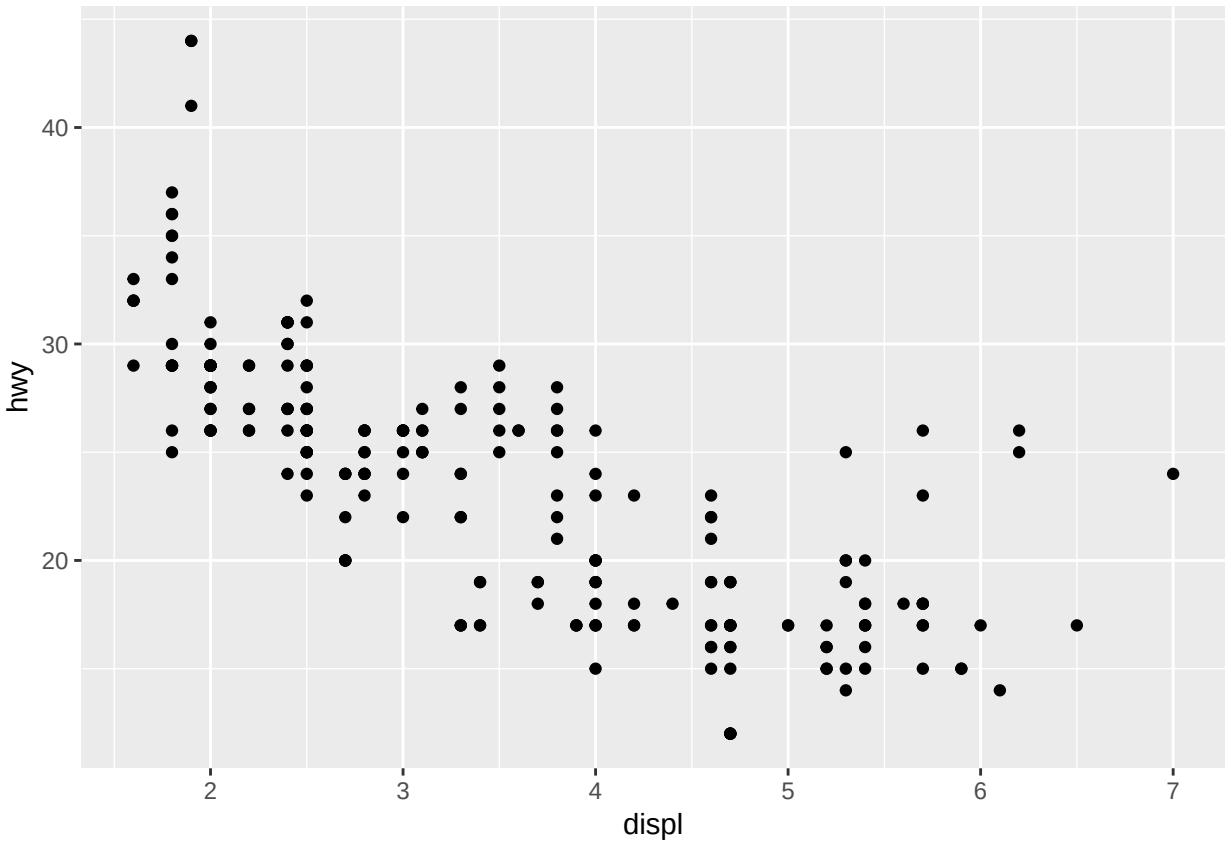
The grammar of ggplot2

In general you begin a plot with the function *ggplot()*. ggplot() creates a coordinate system that you can add layers to. The first argument of ggplot() is the dataset to use in the graph. For instance, typing

```
ggplot(data = mpg)
```


into the console creates an empty graph (which is boring so it is not shown here). You can now add layers to your plot. For example, the function geom_point() adds a layer of points to your plot creating a a scatterplot.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

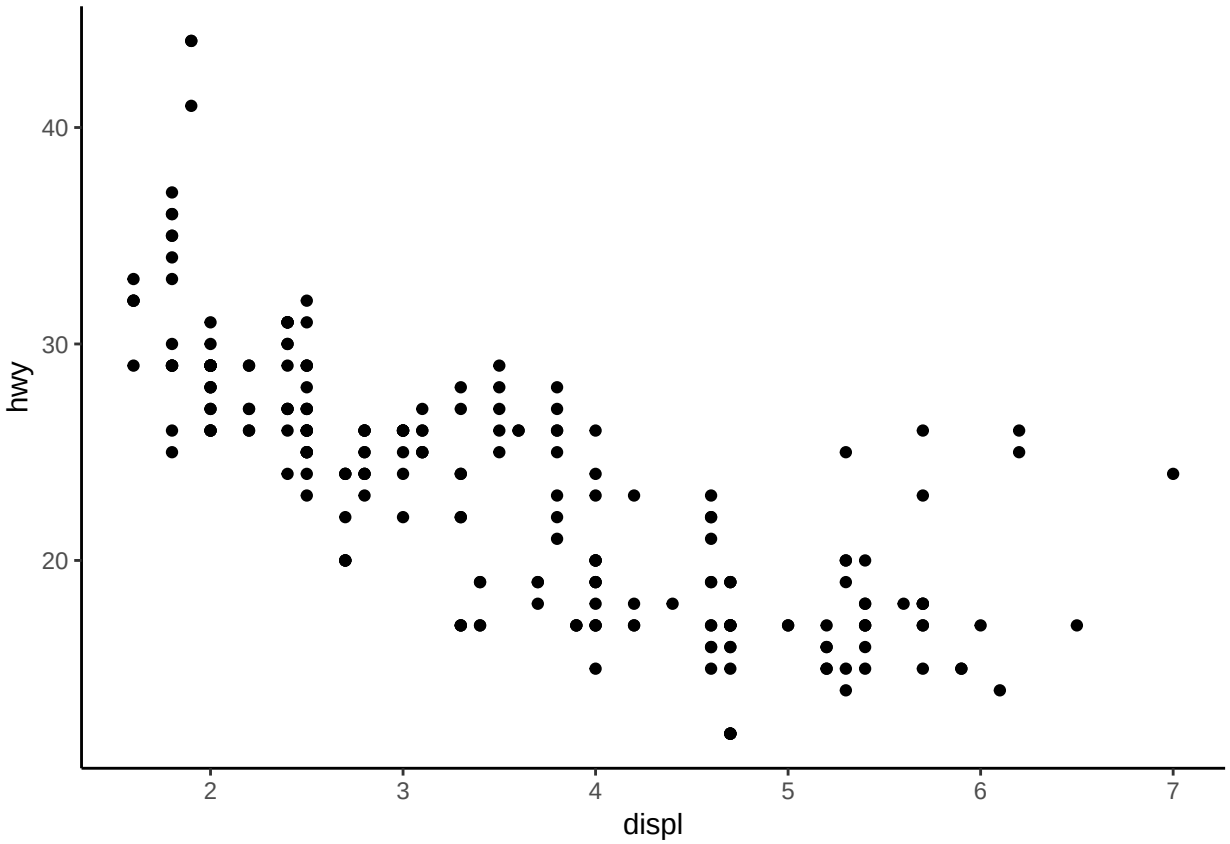


There are many other such *geometirc functions" withing the ggplot2 universe and you can combine them in a single plot. You'll learn a whole bunch of them throughout this course, but be aware that we will only scratch the surface of what is available. After this course, you should be equipped with everthing you need to equip your self with new tools for data analysis whenever you need them.

Each geom function in ggplot2 takes a mapping argument. This defines how variables in your dataset are mapped to visual properties. The mapping argument is always paired with aes(), and the x and y arguments of aes() specify which variables to map to the x and y axes. ggplot2 looks for the mapped variables in the data argument, in this case, mpg.

Next, we use a different theme beacuse this gray background in the standard design is ugly.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  theme_classic()
```



The general grammar of ggplot is

ggplot(data = <DATA>) +

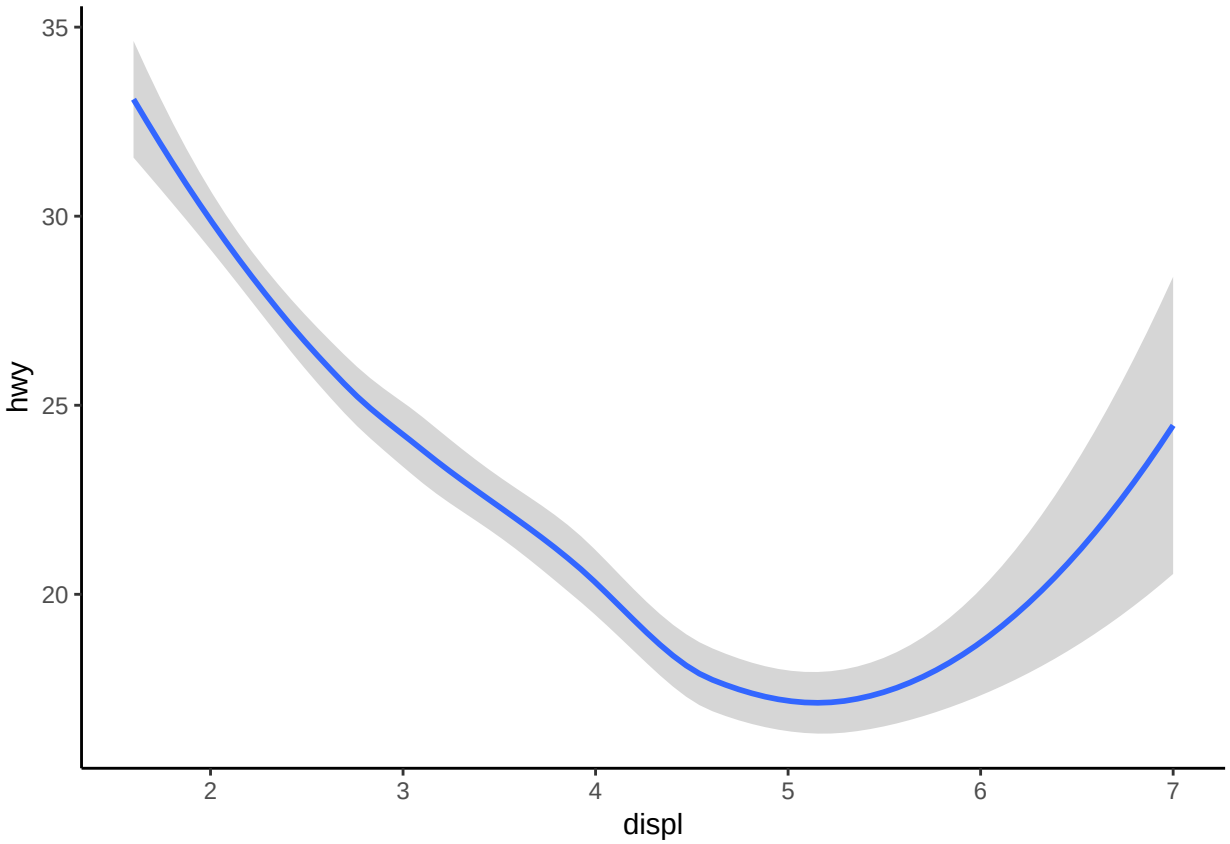
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

We will explain each part step by step in examples. The DATA argument should be quite clear: here you specify the dataframe (we will learn more about that alter) that you want to work with.

The second part, GEOM_FUNCTION, specifies what kind of plot you want. for instance, geom_point gives you a classic scatter plot, whereas geom_smooth would give you a smooth line theat best resembles the cloud of points (more on that later):

```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy)) +
  theme_classic()
```

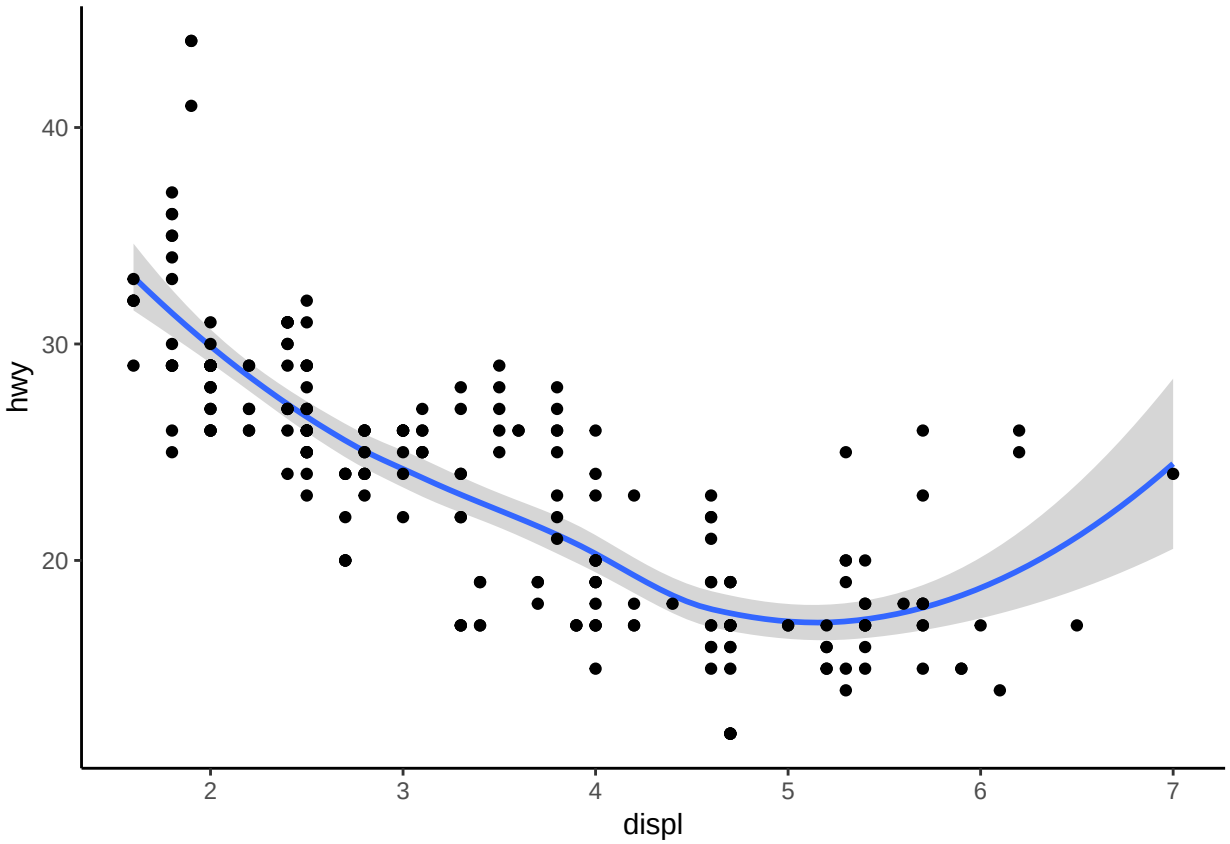
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Of course, we can put points and a smooth line on the same plot:

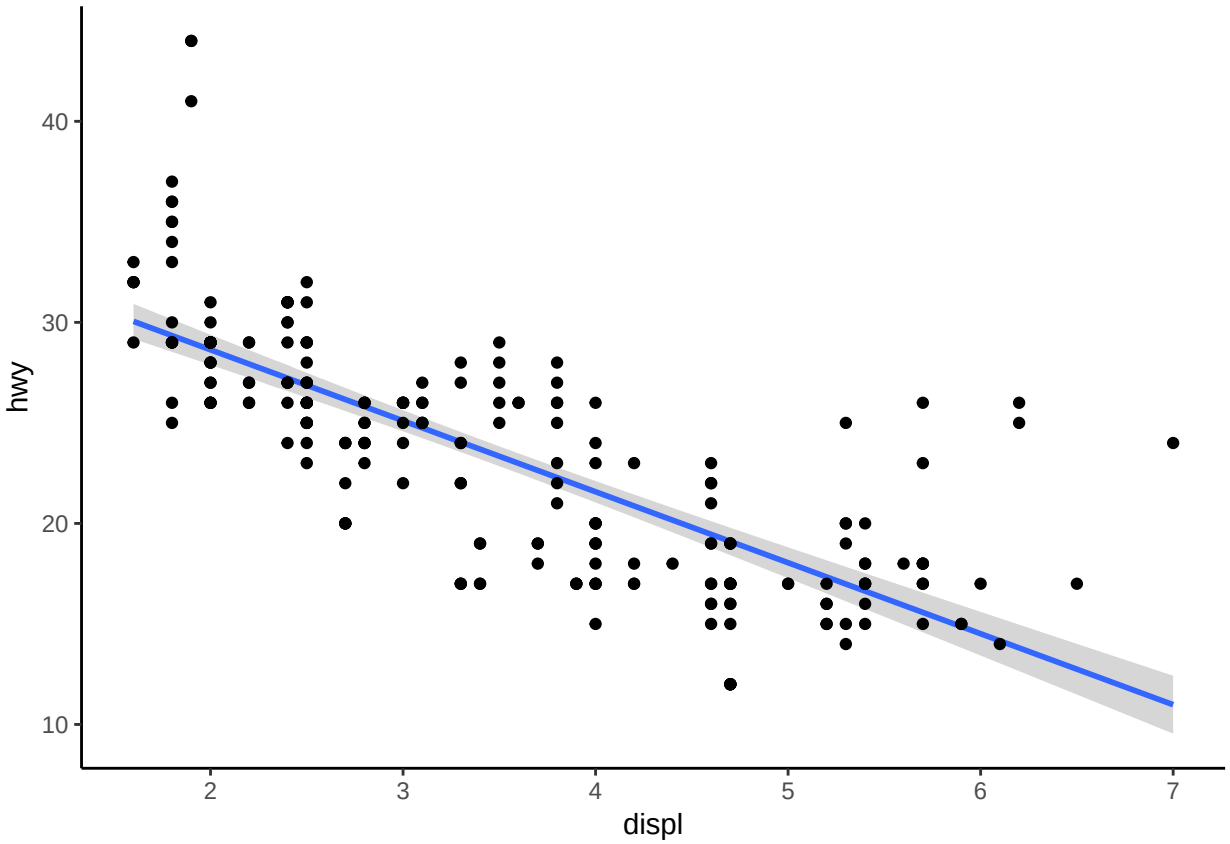
```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  theme_classic()
```

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



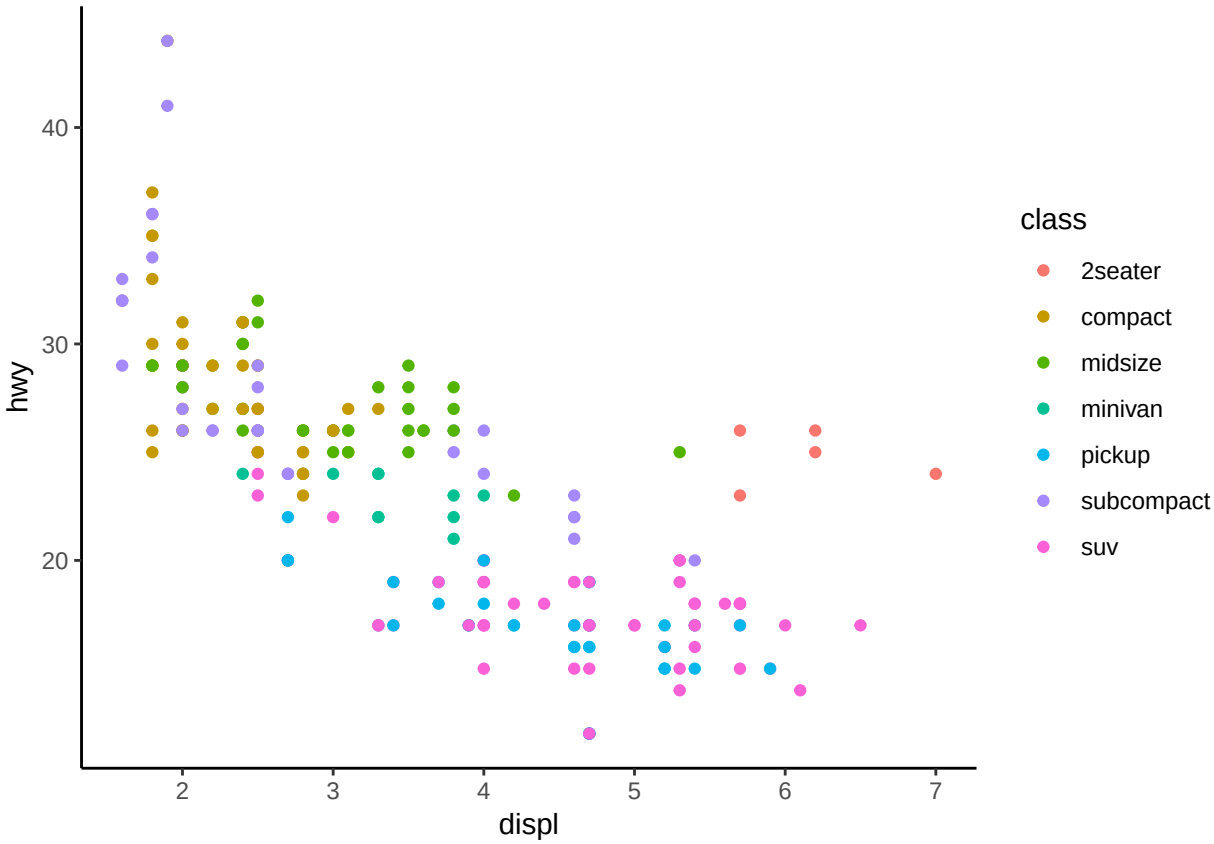
You may wonder how the smooth line is determined. We will come back to this later in the chapter on linear regression. For now, we just mention that we can modify the method used for the plot, e.g., to get a linear regression line:

```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy),method='lm') +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  theme_classic()
```



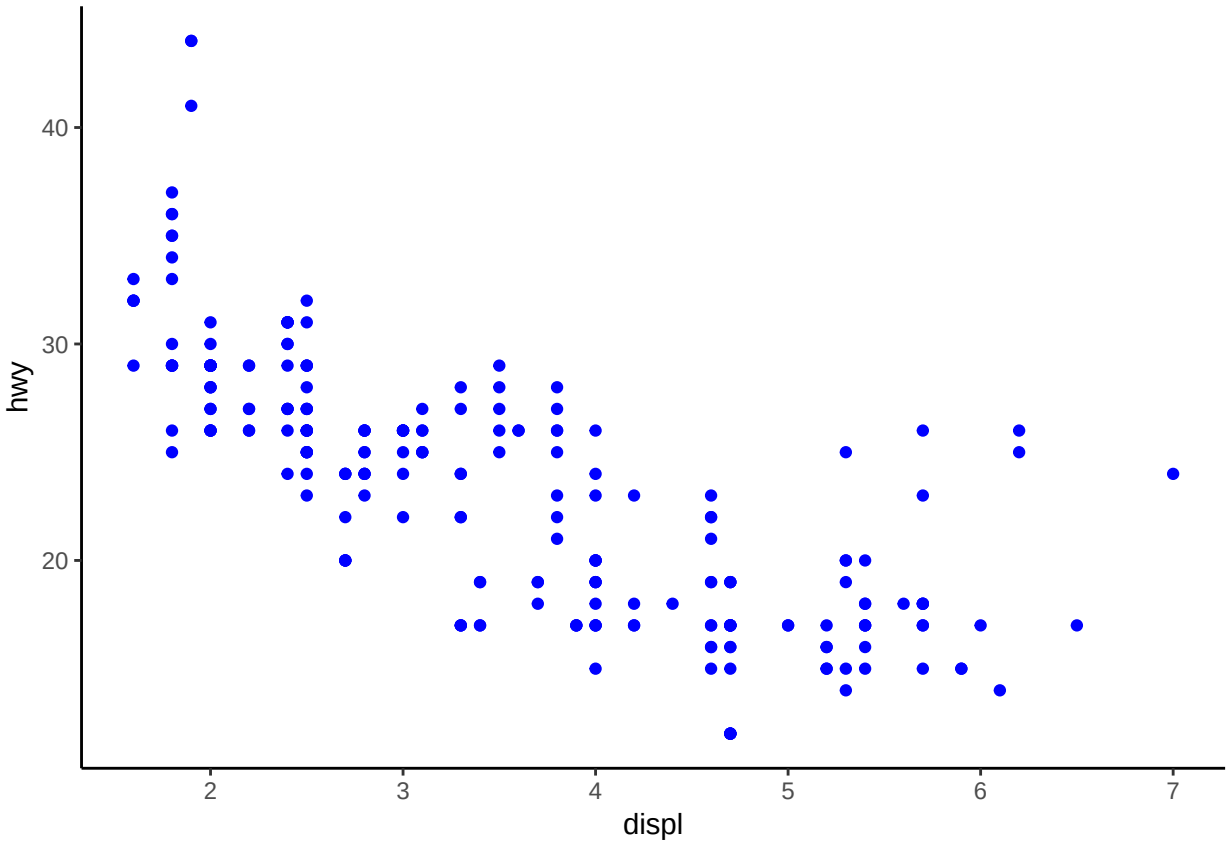
Finally, MAPPING specifies which variables hould be plotted. We ussually specify this with an aesthetics mapping of the form mapping = aes(x = ..., y =...) to deterimne the variables that give us the x- and y-coordinates. We can however also change the aesthetics of our plot in various way, for instance by giving data points different colors that indicate the value of a third variable:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy,color = class)) +
  theme_classic()
```



Another important option is to set a clor manually:

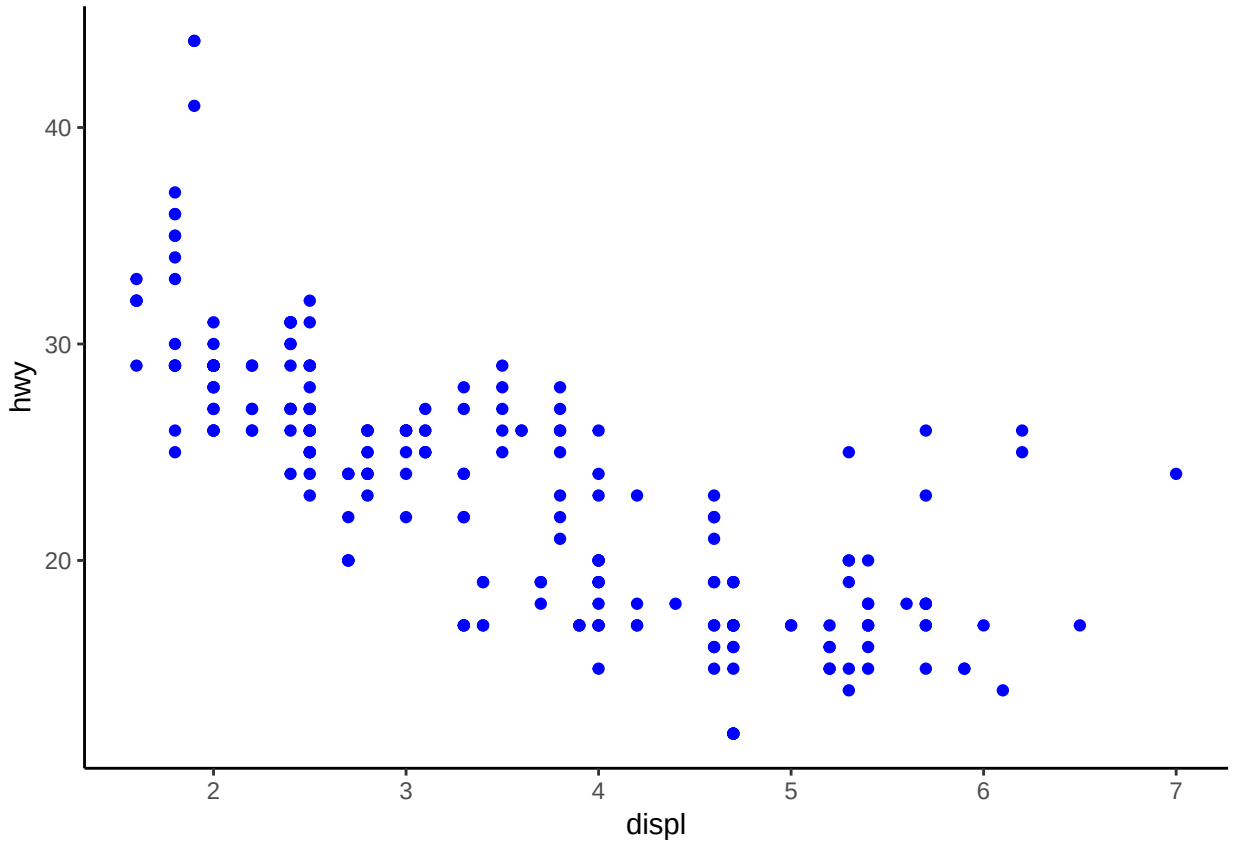
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue") +
  theme_classic()
```



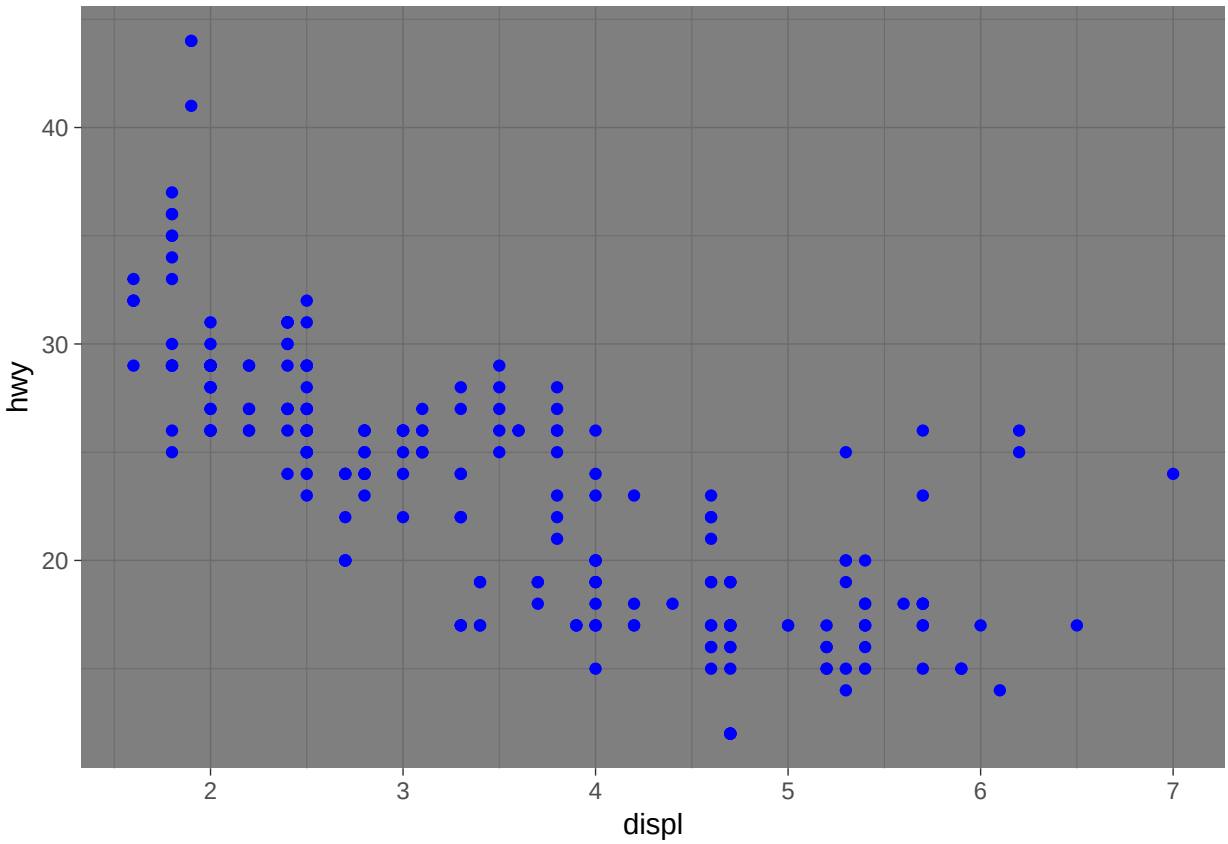
As a final remark, you can also create a plot, store it in a variable and then explore themes after that:

```
p = ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue")

p+theme_classic()
```




```
p+theme_dark()
```



Loading datasets

First we need to make sure R knows where our working directory is:

```
setwd("/Users/stephan/Dropbox/Teaching/StatisticsForBiology/")
```

Now we can load a data frame using the function *read.csv* and store it in the variable stickleback

```
stickleback = read.csv("chap03e3SticklebackPlates.csv")
```

The variable *stickleback* now contains a so called data frame. We can have a quick summary of the content of this data frame using the command *str* (short for STRucture):

```
str(stickleback)
```

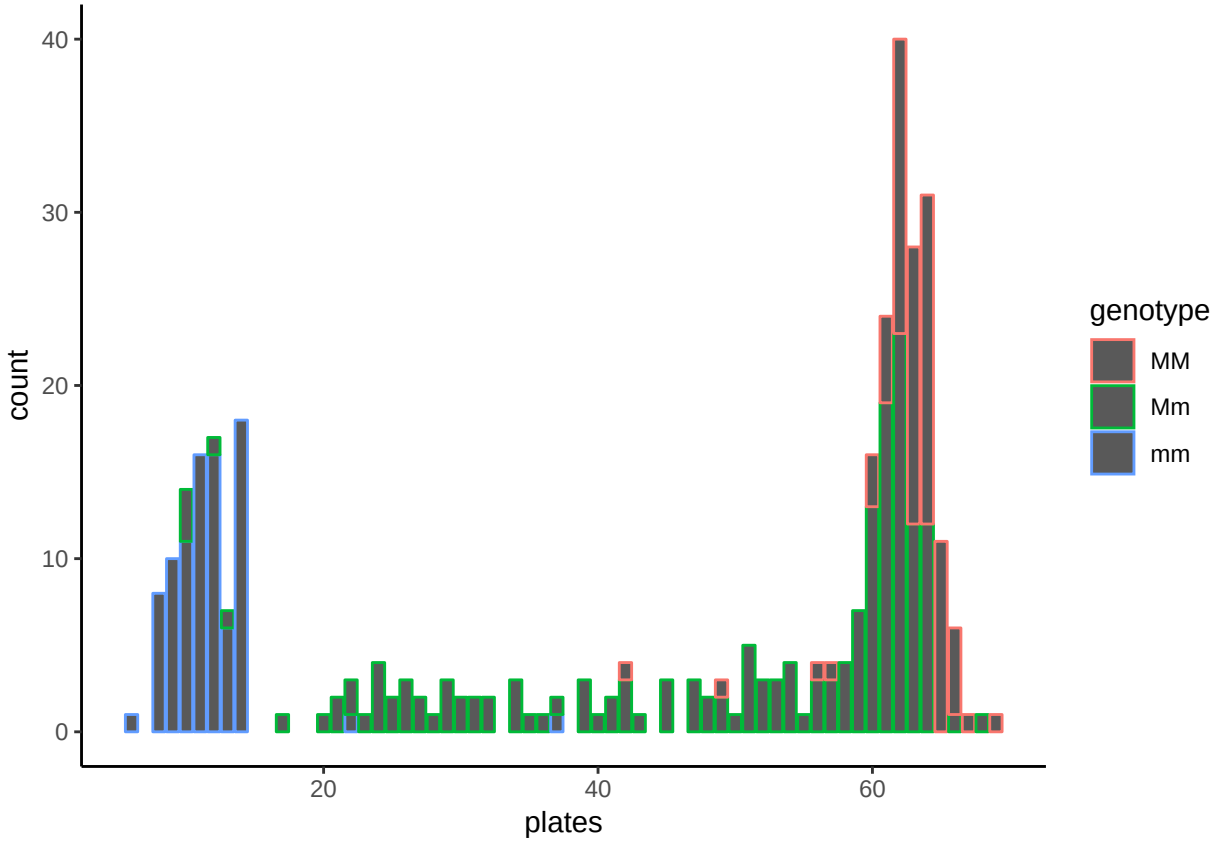
```
## 'data.frame':    344 obs. of  3 variables:
##  $ id      : Factor w/ 344 levels "4-1","4-10","4-100",..: 1 102 282 293 2 22 42 131 212 280 ...
##  $ plates  : int  11 63 22 10 14 11 58 36 31 61 ...
##  $ genotype: Factor w/ 3 levels "MM","Mm","mm": 3 2 2 2 3 3 2 2 2 2 ...
```

We can see form the output that we have 344 observations (= sample size) and for each individual we have measured 3 variables (id, plates and genotypes). ID and genotpye are so-called factors, that is, a catergorical variable and plates is of type integer.

We used the function *read.csv* because our data is stored in the csv format (short for comma seperate values - open the csv file to see what that means). CSV is a common format and you can export your data from software like Excel in that format. Of course, R provides functions for other formats as well.

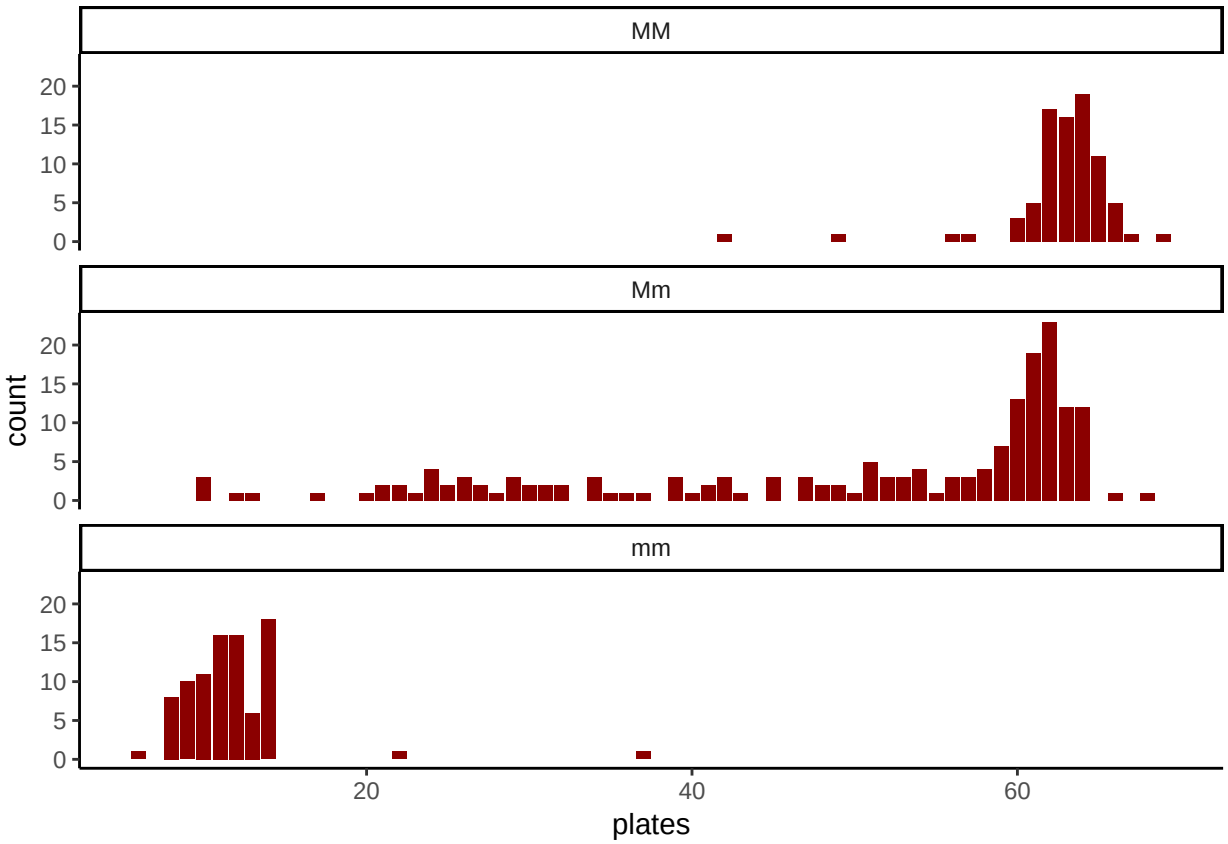
Let us use our ggplot skills to create a nice figure that visualizes our data set. For this we introduce the fucntion *geom_bar()* which allows us to make a bar plot of the number of plates per indivudal, colored by genotype:

```
ggplot(data = stickleback) +
  geom_bar(mapping = aes(x = plates,color=genotype)) +
  theme_classic()
```



This figure is a bit messy. A better way would be to show the distribution for each genotype in its own window. This can be done using the function *facet_wrap*. As before, we can simply add this to our ggplot command:

```
ggplot(data = stickleback) +
  geom_bar(mapping = aes(x = plates),fill="darkred") +
  facet_wrap(~genotype,nrow = 3) +
  theme_classic()
```



We used a new symbol here: the ~. This is part of an R object called formula and it will alter become clear why we used it here like this. For now, just remeber to use it in front of the varibale name in facet wrap. The second aprameter is simply the number of rows in the plot. Try changing it and see what happens. Note: the variable that you pass to facet_wrap shoudl **always** be categorical!

Descriptive Statistics

We continue using the stickleback data set and calcualte some statistics on it. A simple way to get a quick overview about the properties of your data frame is using the funciton *summary*:

```
summary(stickleback)
```

```
##        id          plates      genotype
##  4-1    :  1   Min.   : 6.00   MM: 82
##  4-10   :  1   1st Qu.:14.00   Mm:174
##  4-100  :  1   Median :57.00   mm: 88
##  4-101  :  1   Mean   :43.43
##  4-103  :  1   3rd Qu.:62.00
##  4-105  :  1   Max.   :69.00
##  (Other):338
```

If you want to calcualte a specific statistic, like the mean of a variable, this can be done easily:

```
mean(stickleback$plates)
```

```
## [1] 43.43314
```

There is again something new here, the $ operator: it allows you to access a specific column of a data frame.

Try typing the name of a dataframe, followed by the $ operator in RStudio; it should automatically suggest you the availbale variable names in your data frame.

You may have noticed that we have no calcualted the mean number of plates for *ALL* fish in our data set. What if we want to calcualte the number of plates for a specific genotype? We can use the function filter (as always in R, there are other ways to do this - I chose to present filter because I think it is easy to undersand and very "readable"):

```
stickleback.mm = filter(stickleback,genotype=="mm")
mean(stickleback.mm$plates)
```

```
## [1] 11.67045
```

You could now calcualte the mean plate numbers for each genotpye in this way:

```
mean.mm = mean((filter(stickleback,genotype=="mm"))$plates)
mean.mM = mean((filter(stickleback,genotype=="mM"))$plates)
mean.MM = mean((filter(stickleback,genotype=="MM"))$plates)
```

Note: The tidyverse offers an elegant way to write a long series of commands in a very readable format. To show you how to combine multiple commands quickly, we first need two more functions, *group_by* and *summarize*, and a new operator called the pipe: %>% (because we create a pipeline thorugh which our data "flows"). The above code to calculate the mean for each genotype would become:

```
stickleback %>%
group_by(genotype) %>%
summarise(avg = mean(plates))
```

```
##        avg
## 1 43.43314
```

This is very readable: you take the dataframe stickleback, group it by genotye and summarize it by calcualting the mean number of plates.

A bit of workflow

Manipulating data frames

Loops

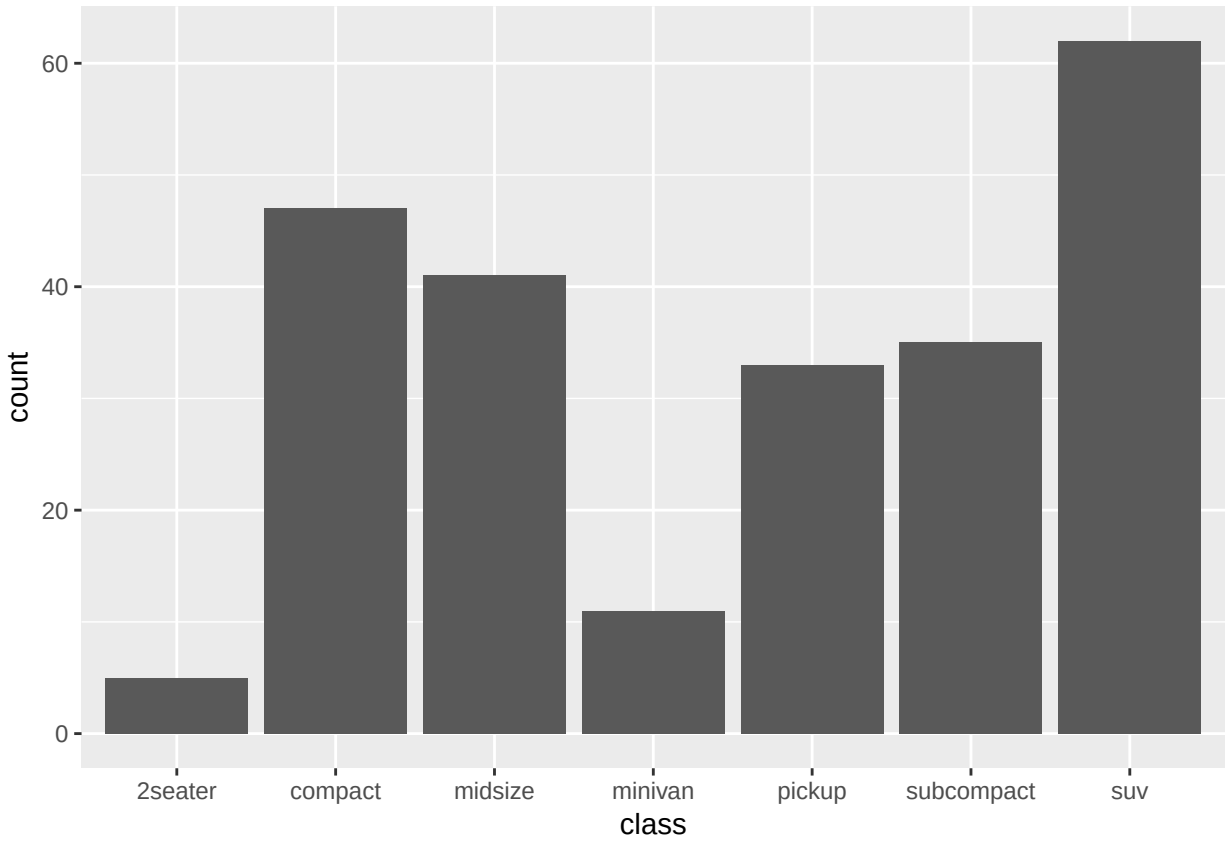
If / else

Distributions

Visualizing distributions

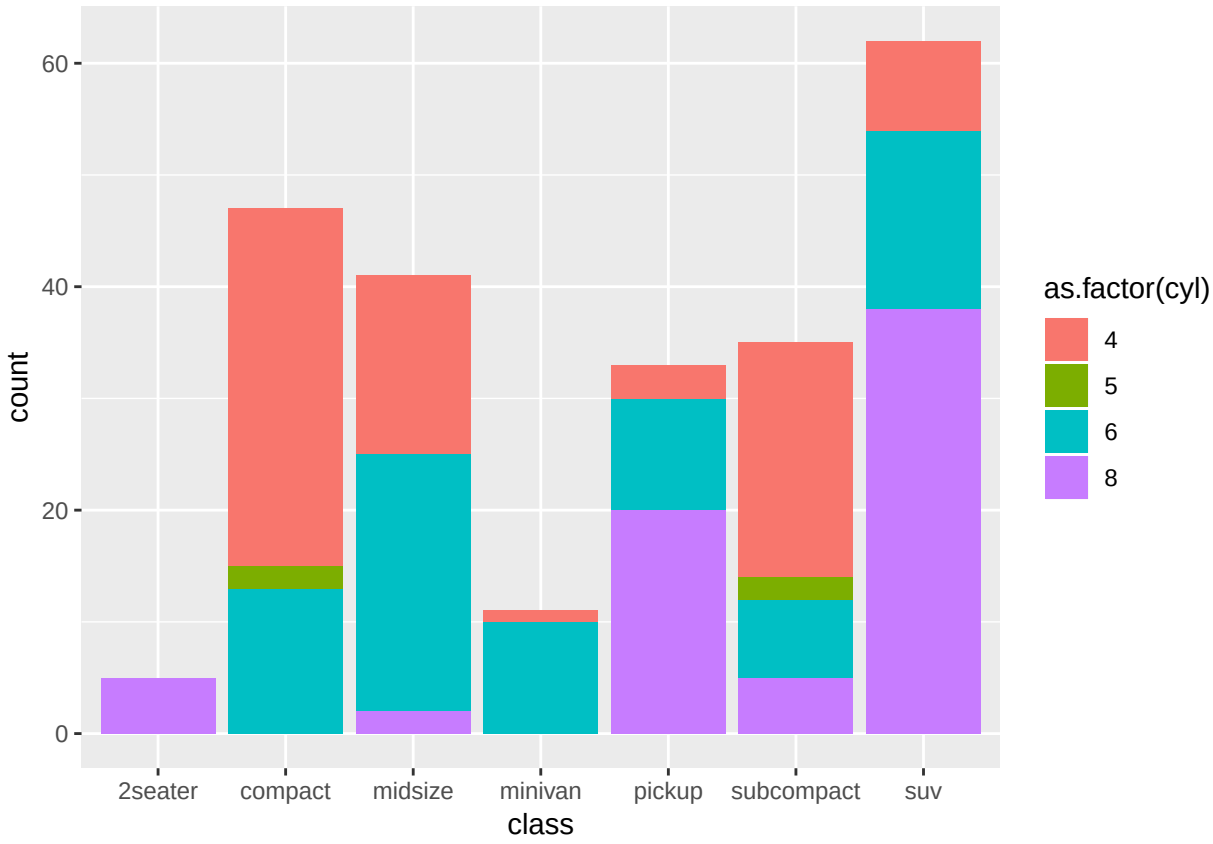
We can visualize continous data in a histogram, for instnace how many cars we have in each class:

```
ggplot(data = mpg) +
  geom_bar(mapping = aes(x = class))
```



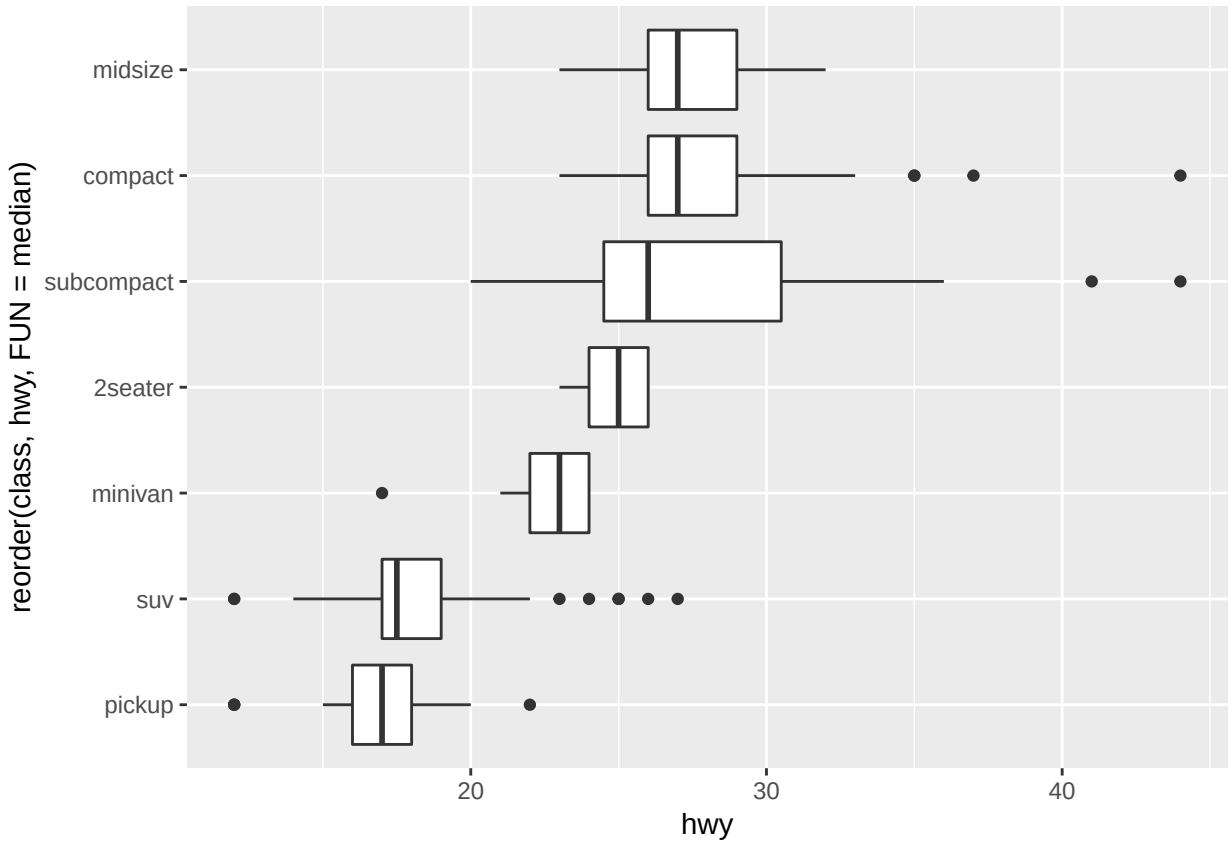
We can make stacked graphs using the argument *fill* and a discrete variable such as number of cylinders

```
ggplot(data = mpg) +
  geom_bar(mapping = aes(x = class, fill = as.factor(cyl)))
```



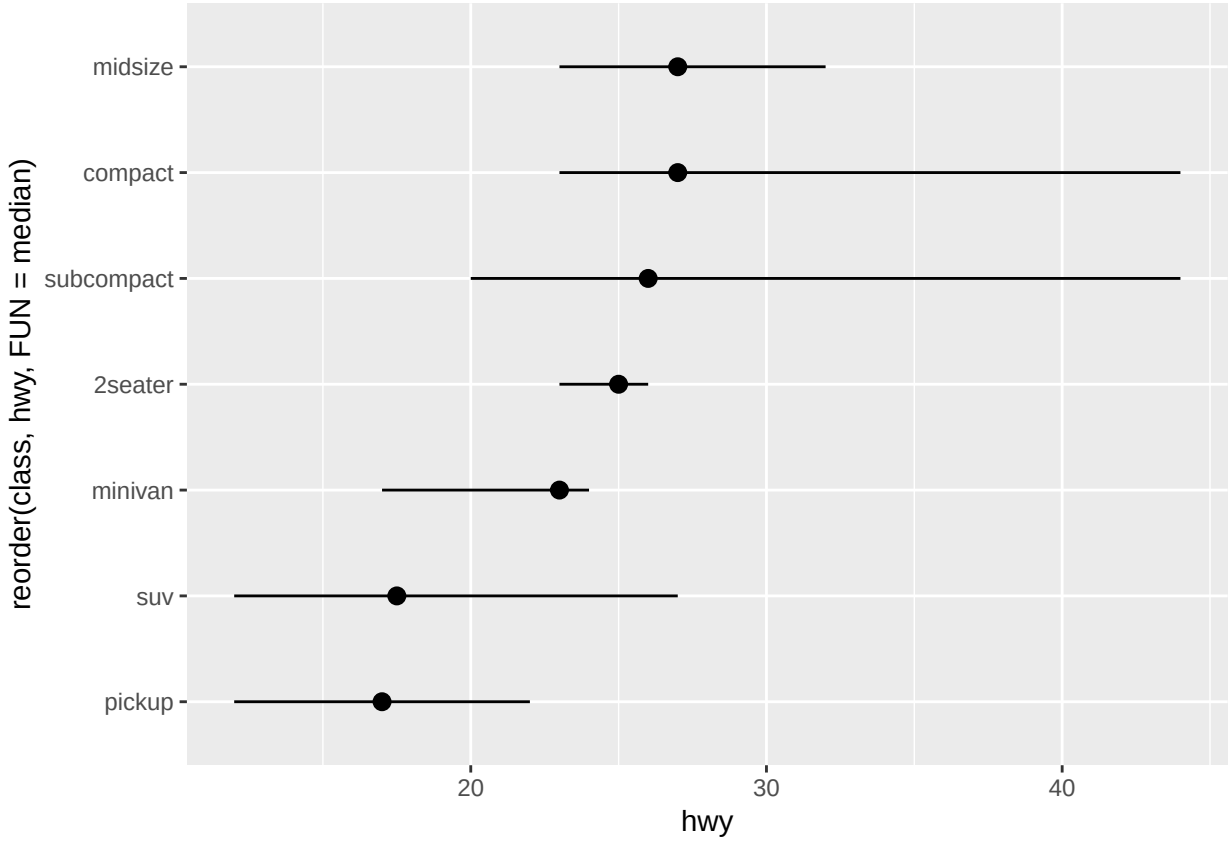
If we want to show the distribution of a continous variable for each class, we can do boxplots, e.g., showing the mileage of each class

```
ggplot(data = mpg) +
geom_boxplot(mapping = aes(x = reorder(class, hwy, FUN = median), y = hwy)) +
coord_flip()
```



Another way of summarizing data is by showing several statistics using whisker plots. A simple solution is using the stat_summary function. In the example below the dots show the median, and the whiskers extend to the minium and maximum in each category (class).

```
ggplot(data = mpg) +
  stat_summary(
    mapping = aes(x = reorder(class, hwy, FUN = median), y = hwy),
    fun.ymin = min,
    fun.ymax = max,
    fun.y = median
  ) +
  coord_flip()
```



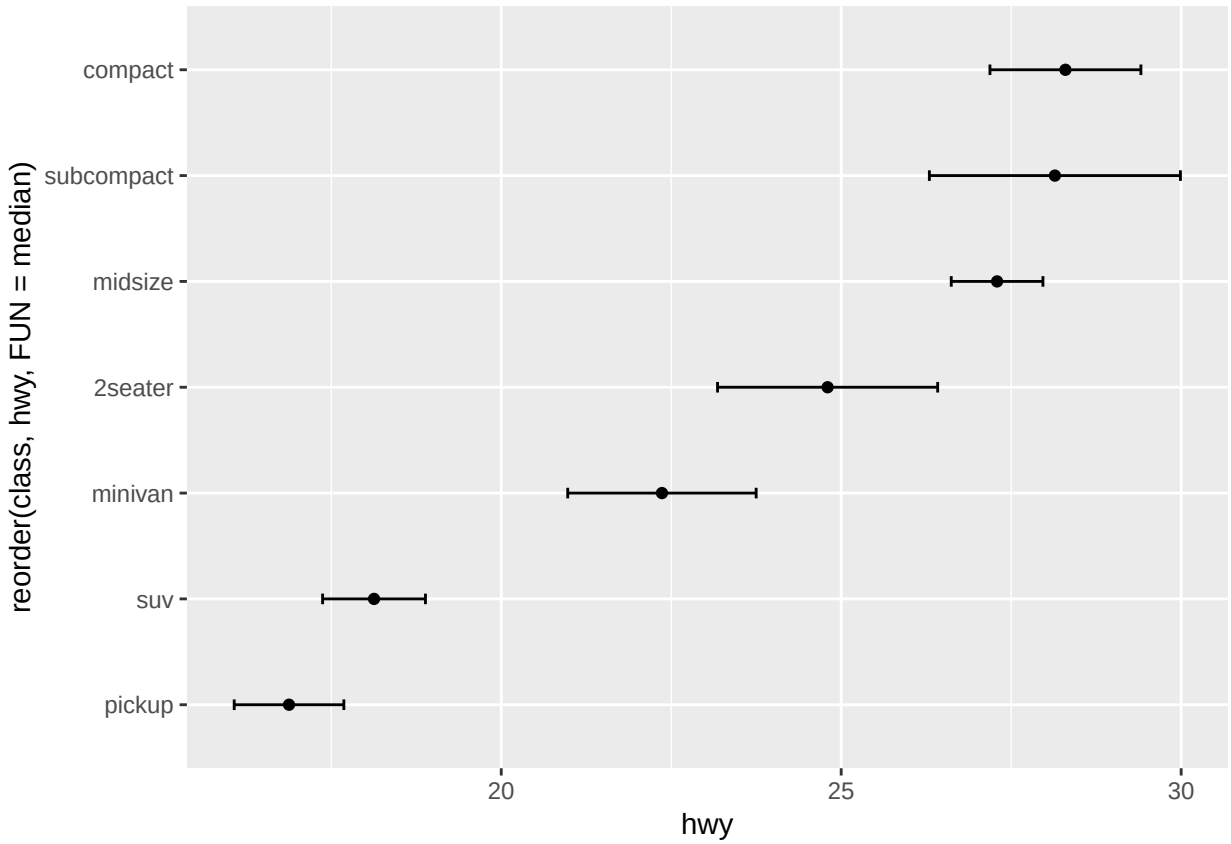
If we want to show confidnece intervalls (or any otehr quantity), a clean way to do this is to simply create a data set containing this information and then use it to plot the quantitites.

```
mpgSE <- summarySE(mpg, measurevar="hwy", groupvars=c("class"))

mpgSE
```

```
##        class  N      hwy       sd        se        ci
## 1    2seater  5 24.80000 1.303840 0.5830952 1.6189318
## 2    compact 47 28.29787 3.781620 0.5516059 1.1103251
## 3    midsize 41 27.29268 2.135930 0.3335762 0.6741826
## 4    minivan 11 22.36364 2.062655 0.6219139 1.3857105
## 5     pickup 33 16.87879 2.274280 0.3959013 0.8064245
## 6 subcompact 35 28.14286 5.375012 0.9085429 1.8463813
## 7        suv 62 18.12903 2.977973 0.3782030 0.7562636
```

```
ggplot(mpgSE, aes(x = reorder(class, hwy, FUN = median), y = hwy)) +
    geom_errorbar(aes(ymin=hwy-ci, ymax=hwy+ci), width=.1) +
    geom_point() +
    coord_flip()
```


compact
subcompact
midsize
2seater
minivan
suv
pickup
reorder(class, hwy, FUN = median)
20
25
30
hwy

Hypothesis Tests

Binomial-test, t-test

Deviations from Normailty

QQ Plots

Transformations

Non-Parametric Alterantives

Permutation Test

Testing for independence

Chi-square test

ANOVA

Linear Regression