# Plotting data with R

*Ian Handel*

# Contents

# Chapter 1

# Preface

Here we'll talk about...

## 1.1 Learning objectives

1. Be able to explain the purpose of statistical graphics
2. Be able to list and explain the principles of good statistical graphic design
3. Be able to explain the principles of 'The Grammar of Graphics'
4. Be able to design, on paper, appropriate plots for simple data sets
5. Be able to use the `ggplot` package to produce basic plots
6. Be able to list sources of help for `ggplot`

## 1.2 Prerequisites

In order to complete this module you'll need to have R and Rstudio (free desktop version) installed.

Download R

Download RStudio

You'll need to have a basic ability with R so that you are familiar with data types and data.frames. You'll need to be able to run functions and be able to give arguments to R functions. You'll need to be able to enter R code into RStudio, ideally by writing a script file. These skills are covered in [list relevant modules] modules.

## 1.3 How we'll be teaching this

This module will make most sense if you start at the beginning and work though to the end. Once you've done that it should be a helpful reference for making plots for your own projects. As you work though it will help to copy and paste code into either the RStudio console or, even better, to build an R script that contains the code as you work though with your own comments where appropriate.

## 1.4 Getting help

Online help sources....

Course help....

# Chapter 2

# Introduction

# Chapter 3

# Principles

Here we'll talk about...

Why plot stuff? Perhaps Anscombe & Challenger

Principles of good graphics

Grammar of graphics

Defo need an exanple to talk through

# Chapter 4

# Building a Plot

Vizualising data is extremely important both in terms of data checking and understanding, but also as a very powerful way of describing your data/results to your audience (in a presentation, your thesis, in a paper etc).

One of the really exciting things about R is the amazing graphics that you can produce that are very quickly journal ready. They are infinitely superior to Excel, SAS, Stata, or Minitab. However there is a small hurdle to get over with R but this is made easier by learning the grammar of graphics and using the package ggplot2 in R.

NB: There are simpler ways to plot things with R, but are much uglier so this is worth the pain!

## 4.1 Import dataset

So first of all you need to install the package and then run the library function to then run it in the current environment and start using its functions.

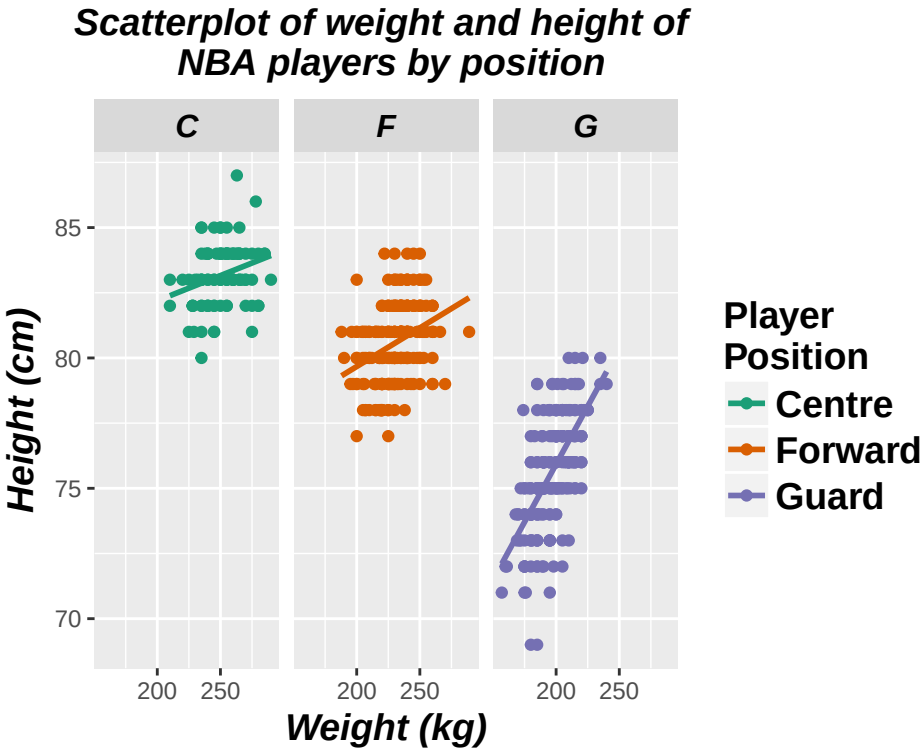
```
#install.packages("ggplot2")
library(ggplot2)
```

Lets import a dataset and see what we can do with it in ggplot. The "NBA.csv" dataset (http://www.stat.ufl.edu/~winner/datasets.html) contains the height, weight, age of NBA players and their positions. As we have done already have a look at the data frame and check how large it is and summarise it.

```
nba <- read.csv("data/NBA.csv")
```

## 4.2 An example of a plot by ggplot

Now lets visualize the relationship between the Weight and Height of NBA players according to their position.




The code for this plot is the following!!

```
ggplot(nba, aes(x = Weight, y = Height, colour = Pos)) +
geom_point() +
stat_smooth(method = "lm", se = FALSE) +
scale_colour_brewer(palette="Dark2",
                    name  ="Player \nPosition",
                    breaks=c("C", "F", "G"),
                    labels=c("Center", "Forward", "Guard")) +
facet_grid(. ~ Pos) +
ggtitle("Scatterplot of weight and height of \n nba players by position") +
xlab("Weight (kg)") +
ylab("Height (cm)") +
theme(axis.title = element_text(colour = "black", size = 14, face = "bold.italic"),
strip.text = element_text(colour = "black", face = "bold.italic", size = 12),
plot.title = element_text(colour = "black", size = 14, face = "bold.italic", hjust = 0.5),
legend.title = element_text(colour="black", size=14, face="bold"),
legend.text = element_text(colour="black", size = 14, face = "bold") )
```

## 4.3 The ggplot grammar

It might look a bit scary, but we will now explain how to build a ggplot step by step by understanding the ggplot grammar. Basically, ggplots are composed of building blocks that are added to the plot one after the other using the + sign. The diagram below shows the most important building blocks. We start building a plot from the bottom!


### 4.3.1 DATA


Anything you try to plot with ggplot needs to belong to a dataframe. The variables we want to visualize belong to the *NBA* dataset.

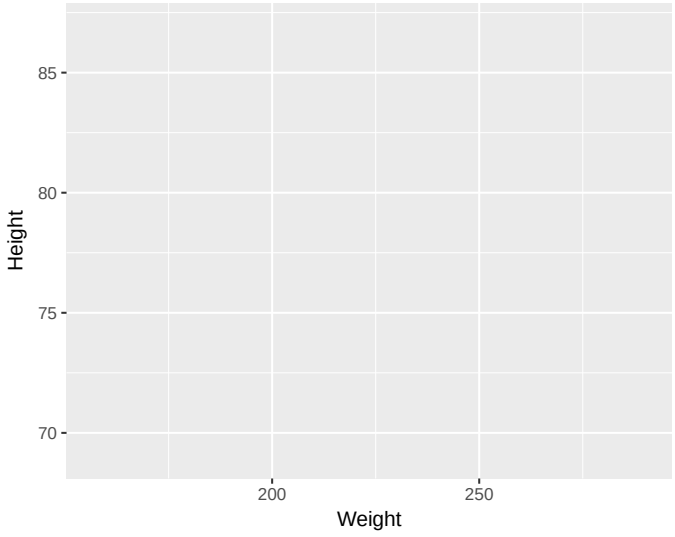
```
head(nba)
```

```
##   X.1 X            Player Pos Height Weight Age Age21
## 1   1 1  Nate\xa0Robinson   G     69    180  29   >21
## 2   2 2  Isaiah\xa0Thomas   G     69    185  24   >21
## 3   3 3   Phil\xa0Pressey   G     71    175  22   >21
## 4   4 4   Shane\xa0Larkin   G     71    176  20  <=21
## 5   5 5      Ty\xa0Lawson   G     71    195  25   >21
## 6   6 6 John\xa0Lucas III   G     71    157  30   >21
```

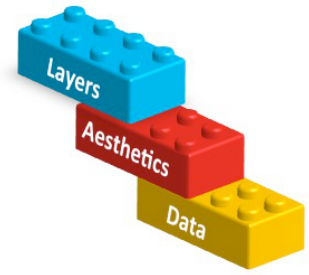
### 4.3.2 Aesthetics mapping


Aesthetics refer to the variables we want to see. In this case weight and height! So let's start building our plot using the *ggplot* function.

```
ggplot(data = nba, aes(x = Weight, y = Height))
```

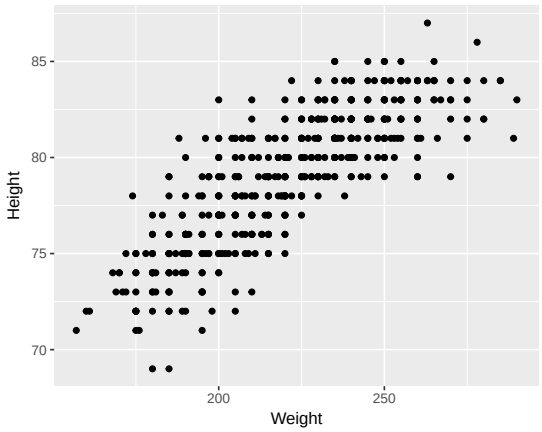



### 4.3.3 Layers



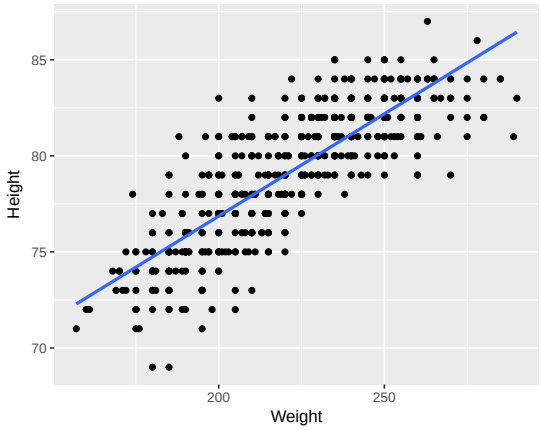

In order to see something on our plot we need to add layers. Layers include geometric elements (geoms) and statistical transformations (stats). Since we want to build a scatterplot our first layer will be a layer of points (geom_point):

```
ggplot(data = nba, aes(x = Weight, y = Height)) +
  geom_point() # Layer 1
```

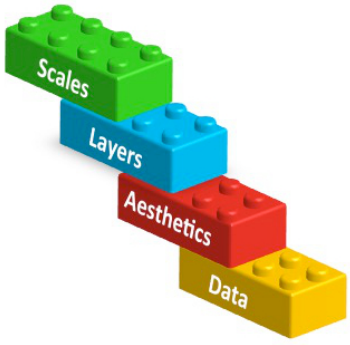



We also want to see the statistical relationship between weight and height so we will add a regression line as our second layer.

```
ggplot(data = nba, aes(x = Weight, y = Height)) +
  geom_point() + # Layer 1
stat_smooth(method = "lm", se = FALSE) # Layer 2
```

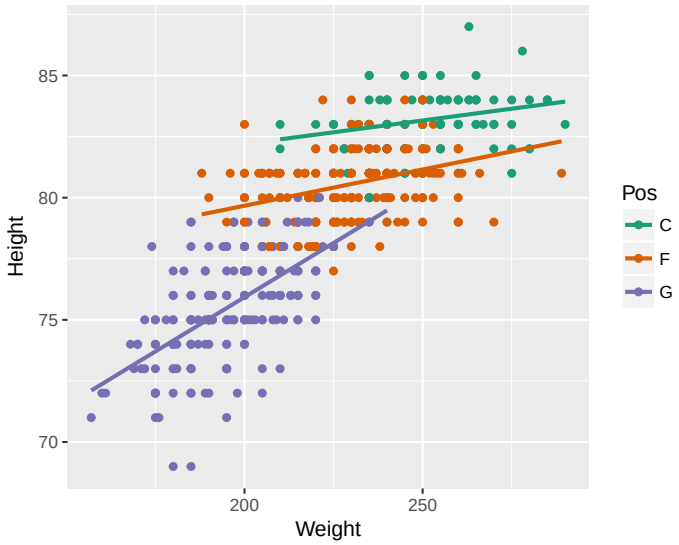



### 4.3.4 Scales



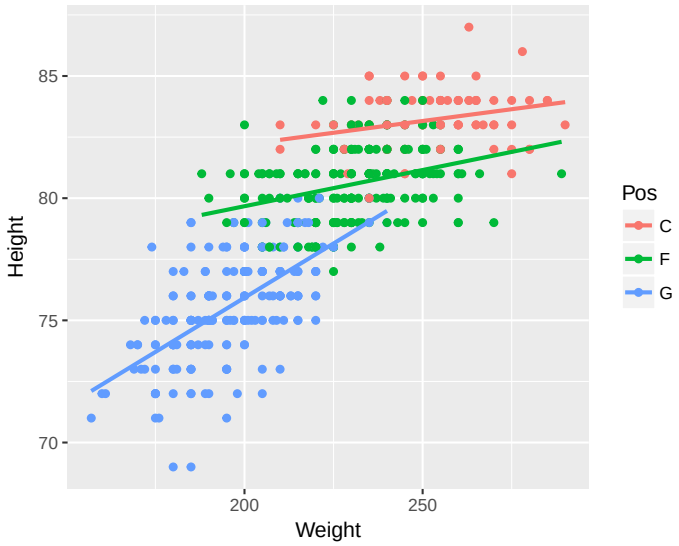

As we said at the beginning we are also interested in the position of each player. As a first step we want to colour each point by the player's position. Scales map values in the data space to values in an aesthetic space. This can be colour, size or shape. This will also automatically create a legend to explain the colours on the plot.

```
ggplot(data = nba, aes(x = Weight, y = Height, colour = Pos)) + # added colour = POS
  geom_point() +
stat_smooth(method = "lm", se = FALSE) +
scale_colour_brewer(palette="Dark2") # scale maps aes data values (x and y) to aes colour
```

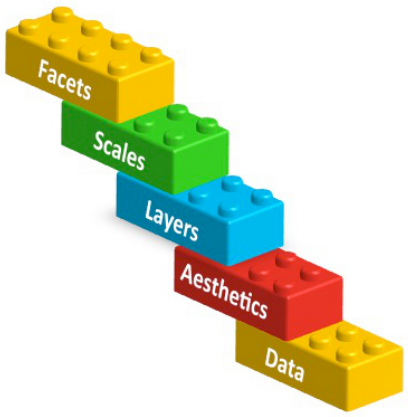



NB: You can obtain a similar plot just by adding a colour variable in the aesthetics. Scales give you the ability to have control over the colours chosen.

```
ggplot(data = nba, aes(x = Weight, y = Height, colour = Pos)) +
  geom_point() +
stat_smooth(method = "lm", se = FALSE)
```

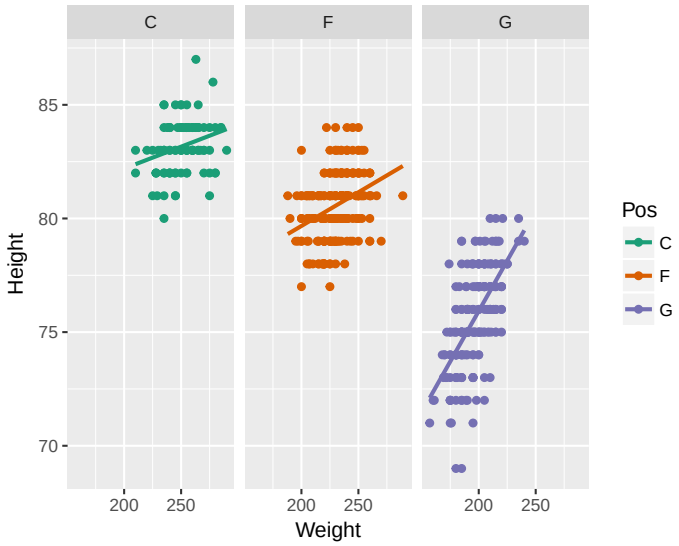


### 4.3.5 Facets




If you remember the original plot, we actually wanted to see a separate plot for each player position. Using facets we can display our data split by the chosen variable, in this case position.

```
ggplot(data = nba, aes(x = Weight, y = Height, colour = Pos)) +
  geom_point() +
stat_smooth(method = "lm", se = FALSE) +
scale_colour_brewer(palette="Dark2") +
facet_grid(. ~ Pos) # split grid by the variable Pos
```



### 4.3.6 Themes and other useful tricks!

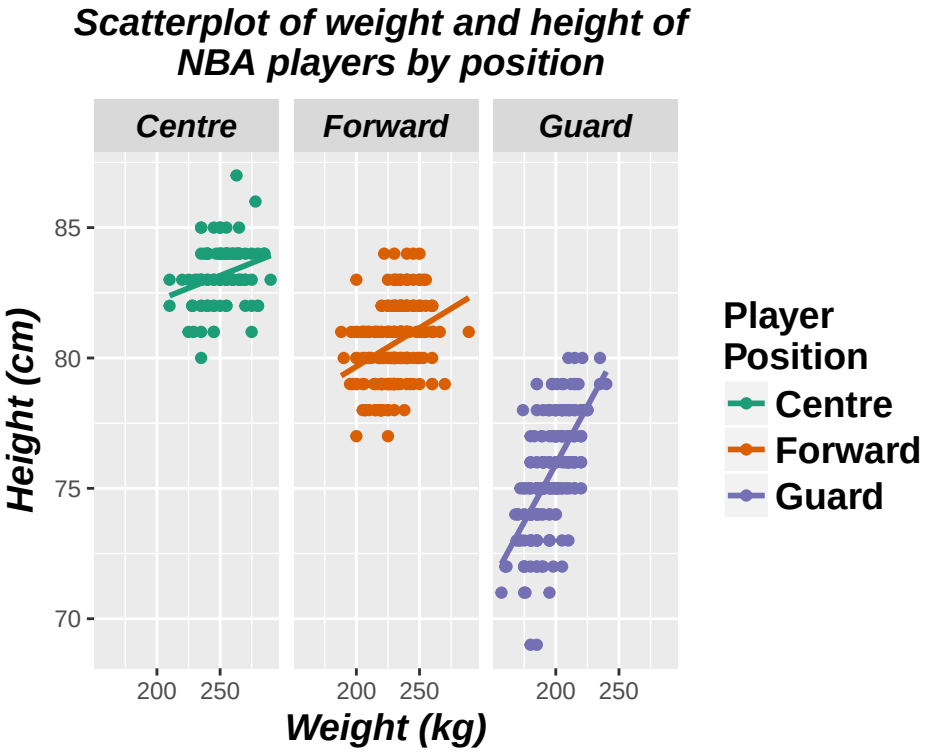

ggplot is very flexible and you can adjust pretty much every aspect of the plot to your preference. In our original plot we had added a plot title using **ggtitle**, changed the x and y axis labels by **xlab** and **ylab** as shown in the code below.

We also changed the title and labels of our legend by adding information to **scale** and the facet labels by adding information to **facet**.

Lastly, we used *theme* to change font size, colour and style of many elements of the plot. You can use theme to change pretty much everything you like on your plot.

Here is the whole code again, with some useful websites to understand each building block better!

```
ggplot(nba, aes(x = Weight, y = Height, colour = Pos)) +
# Visit http://docs.ggplot2.org/current/ for lists of geoms, stats etc
geom_point() +
stat_smooth(method = "lm", se = FALSE) +
# Visit http://docs.ggplot2.org/current/scale_brewer.html for more colour options
scale_colour_brewer(palette="Dark2",
                    name  ="Player \nPosition",
                    breaks=c("C", "F", "G"),
                    labels=c("Centre", "Forward", "Guard")) +
facet_grid(. ~ Pos, labeller=labeller(Pos = c("C"="Centre", "F"="Forward", "G"="Guard"))) +
ggtitle("Scatterplot of weight and height of \n NBA players by position") +
xlab("Weight (kg)") +
ylab("Height (cm)") +
# Visit http://docs.ggplot2.org/current/theme.html and http://www.sthda.com/english/wiki
#/ggplot2-themes-and-background-colors-the-3-elements for info about themes
theme(axis.title = element_text(colour = "black", size = 14, face = "bold.italic"),
strip.text = element_text(colour = "black", face = "bold.italic", size = 12),
plot.title = element_text(colour = "black", size = 14, face = "bold.italic", hjust = 0.5),
#legend.position = "none", # can add this line to remove legend from plot
legend.title = element_text(colour="black", size=14, face="bold"),
legend.text = element_text(colour="black", size = 14, face = "bold") )
```




## 4.4 Other plot types

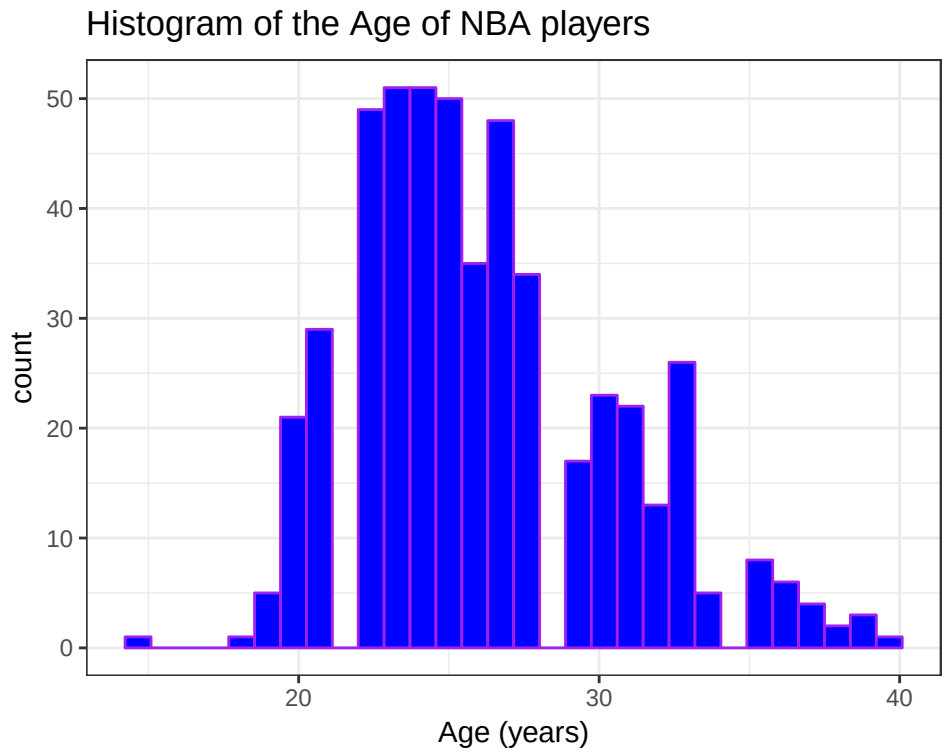
So far we have seen how to draw a scatter plot with ggplot. You can plot pretty much any type of plot you like and you can change that by changing the *geom* type used. There are a number of geoms and you can see these when you use Rstudio as you type. But some of the common ones are below and also in the cheat sheet provided.

| **geom** | description |
|---|---|
| **geom_point** | Points, eg. a scatterplot |
| **geom_line** | lines |
| **geom_ribbon** | Ribbons, y range with continuous x |
| **geom_polygon** | Polygon, a filled path |
| **geom_pointrange** | vertical line with point in the middle |
| **geom_path** | connect observations in original order |
| **geom_histogram** | Histograms |
| **geom_text** | Textural annotations |
| **geom_violin** | Violin plots |
| **geom_map** | Polygons from map |

## 4.5 Histograms

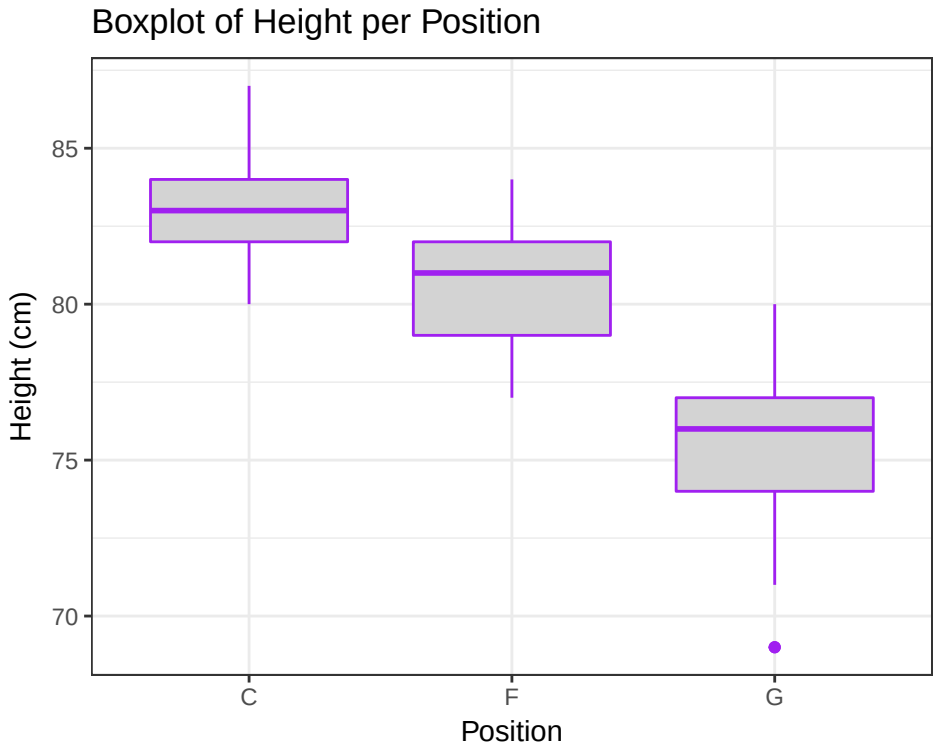
```
p <- ggplot(data=nba, aes(x=Age)) +
      geom_histogram( fill="blue", colour="purple") +
      labs(title="Histogram of the Age of NBA players", x="Age (years)") +
      theme_bw()
p
```

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



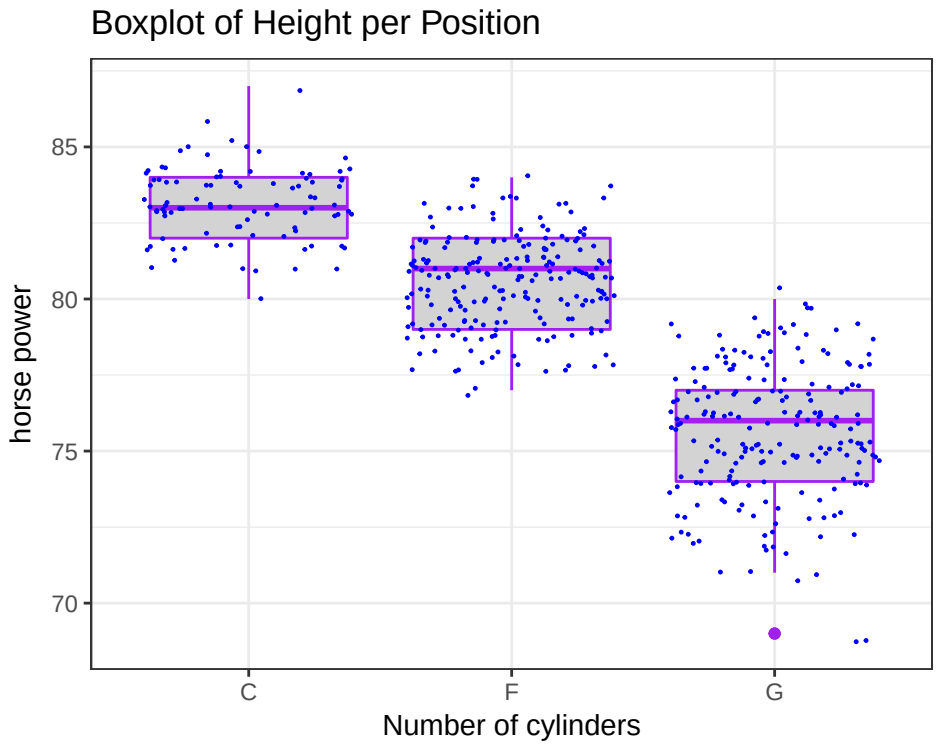
## 4.6 Boxplots

```
ggplot(data=nba) +
      geom_boxplot(aes(y=Height, x=Pos), fill="lightgrey", colour="purple") +
      labs(title="Boxplot of Height per Position", y="Height (cm)", x="Position") +
# Set title and axis labels
      theme_bw()
```



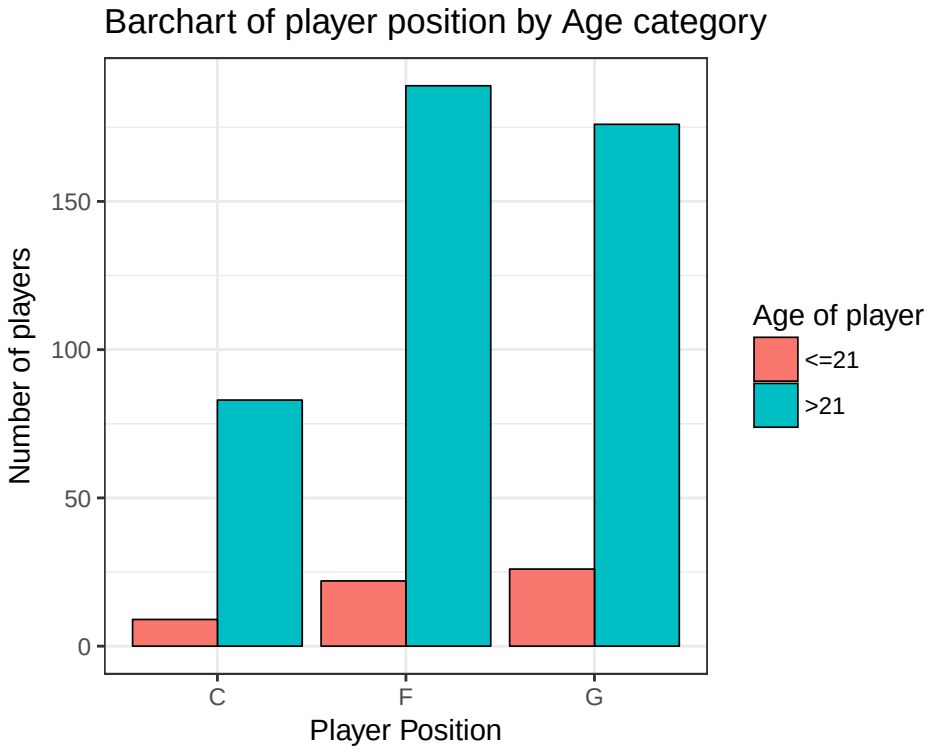
One of the things about box plots though is that you loose a lot of information so you can get fancy and overlay the raw dots like this...

```
ggplot(data=nba) +
geom_boxplot(aes(y=Height, x=Pos), fill="lightgrey", colour="purple") +
geom_jitter(aes(y=Height, x=Pos), colour="blue", size=0.2) + # geom_jitter spreads points
#out so that they don't overlap
labs(title="Boxplot of Height per Position", y="horse power", x="Number of cylinders") +
theme_bw()
```



## 4.7 Barcharts

```
ggplot(data=nba, aes(x=Pos, fill=Age21)) +
    geom_bar(colour="black", stat="count",
             position=position_dodge(), # split bars by Age21 variable.
                                        #try commenting this line out
             size=.3) +                         # Thinner lines
    scale_fill_discrete(name="Age of player") +      # Set legend title
    xlab("Player Position") + ylab("Number of players") + # Set axis labels
    ggtitle("Barchart of player position by Age category") +     # Set title
    theme_bw() # Set theme
```



## 4.8 More info

- R Cookbook Graphs
  http://www.cookbook-r.com/Graphs/
- Line plots tutorial
  http://t-redactyl.io/blog/2015/12/creating-plots-in-r-using-ggplot2-part-1-line-plots.html
- Bar plots tutorials
  http://t-redactyl.io/blog/2016/01/creating-plots-in-r-using-ggplot2-part-3-bar-plots.html
  http://t-redactyl.io/blog/2016/01/creating-plots-in-r-using-ggplot2-part-4-stacked-bar-plots.html
- Scatter plots tutorials
  http://t-redactyl.io/blog/2016/02/creating-plots-in-r-using-ggplot2-part-5-scatterplots.html
  http://t-redactyl.io/blog/2016/02/creating-plots-in-r-using-ggplot2-part-6-weighted-scatterplots.html
- Histograms tutorial
  http://t-redactyl.io/blog/2016/02/creating-plots-in-r-using-ggplot2-part-7-histograms.html
- Boxplots tutorial
  http://t-redactyl.io/blog/2016/04/creating-plots-in-r-using-ggplot2-part-10-boxplots.html

## 4.9 Exercises

### 4.9.1 Ex1.

Using dataset *mtcars* plot a box-plot of Gross horse power (*hp*) against number of cylinders (*cyl*). Give the plot the title *Boxplot by (Your Name)*. Add the real horse power values using dots coloured by Number of carburetors (*carb*) faceted by Number of forward gears (*gear*). Change the labels of gear to "3 Gears", "4 Gears", "5 Gears".

Hint: To find more info about an R dataset try *?mtcars*.

## 4.10 Ex2.

Using the same dataset plot a stacked bar chart of number of cylinders (*cyl*) by Transmission (*am*). Change colours corresponding to *am* manually to blue for 0 and red for 1. Change the legend labels to Autumatic and Manual and the legend title to Transmission.

Hint: Check the bar plots tutorial websites above for help

# Chapter 5

# Common Graphics

In this section we will show the R code used to generate some common statistical graphics. The graphics will be based on built-in R datasets so you can test them easily and then change the dataset and variable (column headings) parts of the code to easily plot your own data.

## 5.1 Barchart

Barcharts are sometimes used to plot numerical data, including counts, for a set of categories. It is good practice with a barchart to show the bar from zero rather that cutting off the axis. For our first example of a barchart we'll use the `mpg` dataset. This is available once you have loaded in the `ggplot` or `tidyverse` package. Do that now...

```
library(tidyverse) # NB This loads in ggplot as well as other packages
```

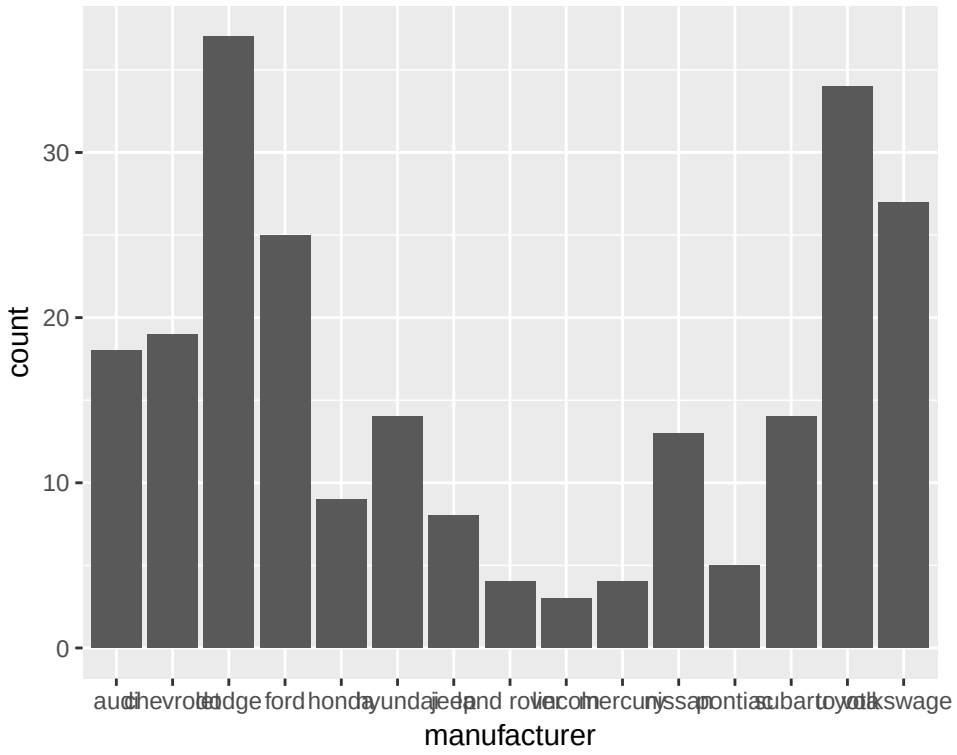
The `mpg` dataset lists 234 cars and includes data on their manufacturer and fuel efficiency. We can look at the top of the dataset with this...

```
print(mpg, width = Inf)
```

```
## # A tibble: 234 x 11
##    manufacturer      model displ  year   cyl      trans   drv   cty   hwy    fl   class
##           <chr>      <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>   <chr>
##  1         audi         a4   1.8  1999     4   auto(l5)     f    18    29     p compact
##  2         audi         a4   1.8  1999     4 manual(m5)     f    21    29     p compact
##  3         audi         a4   2.0  2008     4 manual(m6)     f    20    31     p compact
##  4         audi         a4   2.0  2008     4   auto(av)     f    21    30     p compact
##  5         audi         a4   2.8  1999     6   auto(l5)     f    16    26     p compact
##  6         audi         a4   2.8  1999     6 manual(m5)     f    18    26     p compact
##  7         audi         a4   3.1  2008     6   auto(av)     f    18    27     p compact
##  8         audi a4 quattro   1.8  1999     4 manual(m5)     4    18    26     p compact
##  9         audi a4 quattro   1.8  1999     4   auto(l5)     4    16    25     p compact
## 10         audi a4 quattro   2.0  2008     4 manual(m6)     4    20    28     p compact
## # ... with 224 more rows
```

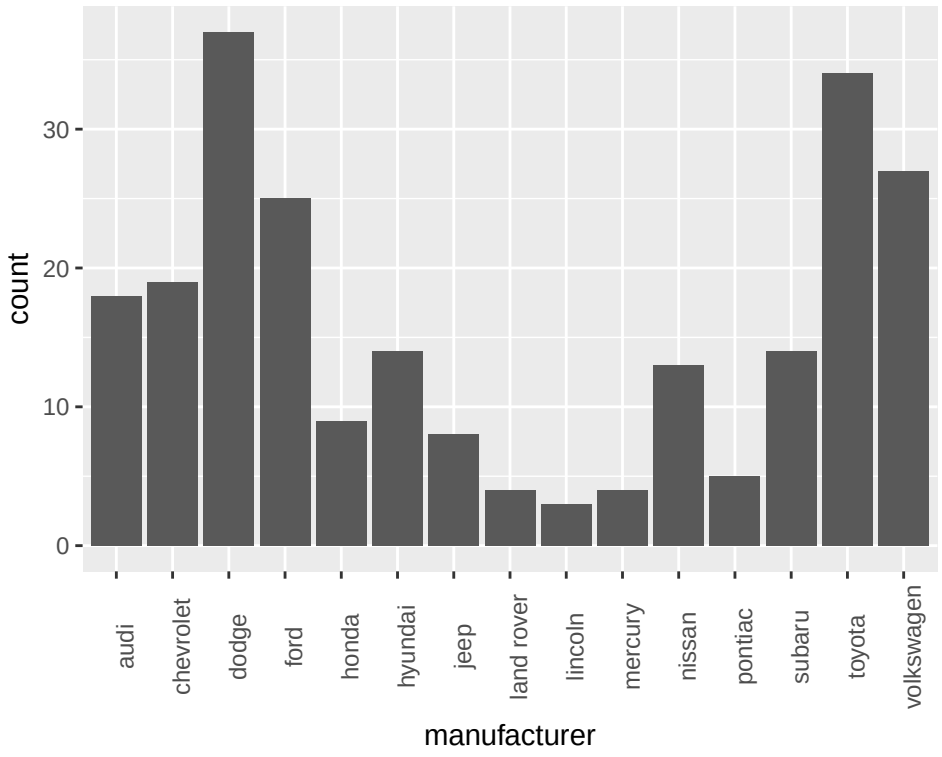
To plot a barchart showing the number of cars in the dataset from each manufacturer we can use the `ggplot()` function with manufacturer as the `x` aesthetic and using the `geom_bar` geom.

```
ggplot(mpg, aes(x = manufacturer)) +
  geom_bar()
```



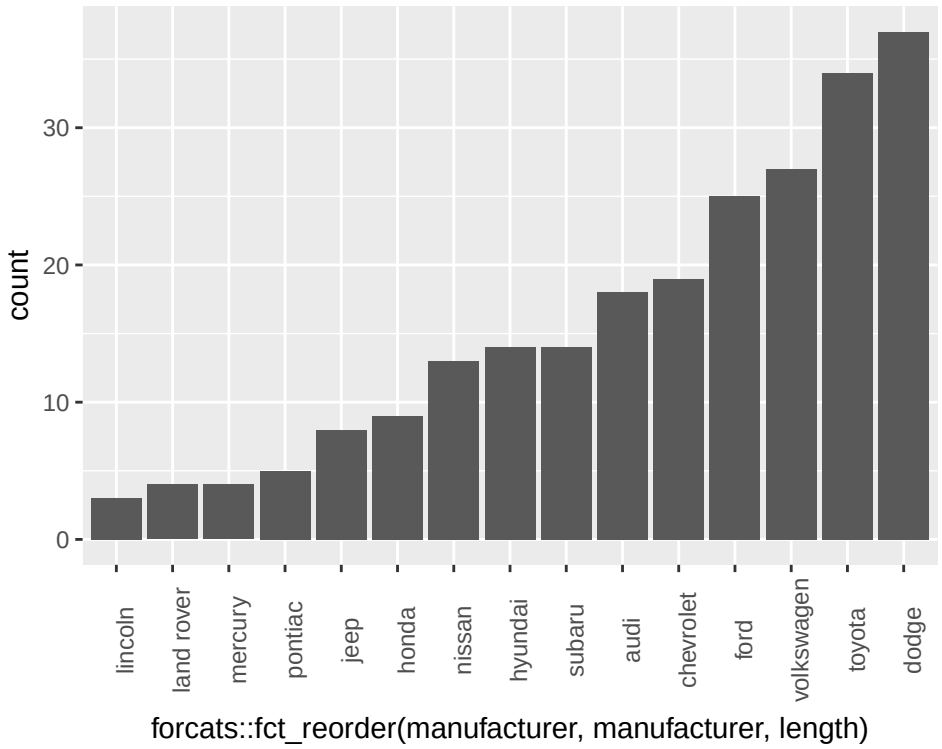

The `geom_bar` geom is clever. If you just give it a factor (categories) as the `x` aesthetic it will default to counting each category and plotting the counts. So each bar height shows the number of rows for that manufacturer. Lets tidy up the `x` axis labels by rotating them through 90 degrees. We add a new line of code with a `theme()` function and tell it to set the angle of the `x` axis to 90 degrees...

```
ggplot(mpg, aes(x = manufacturer)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle = 90))
```



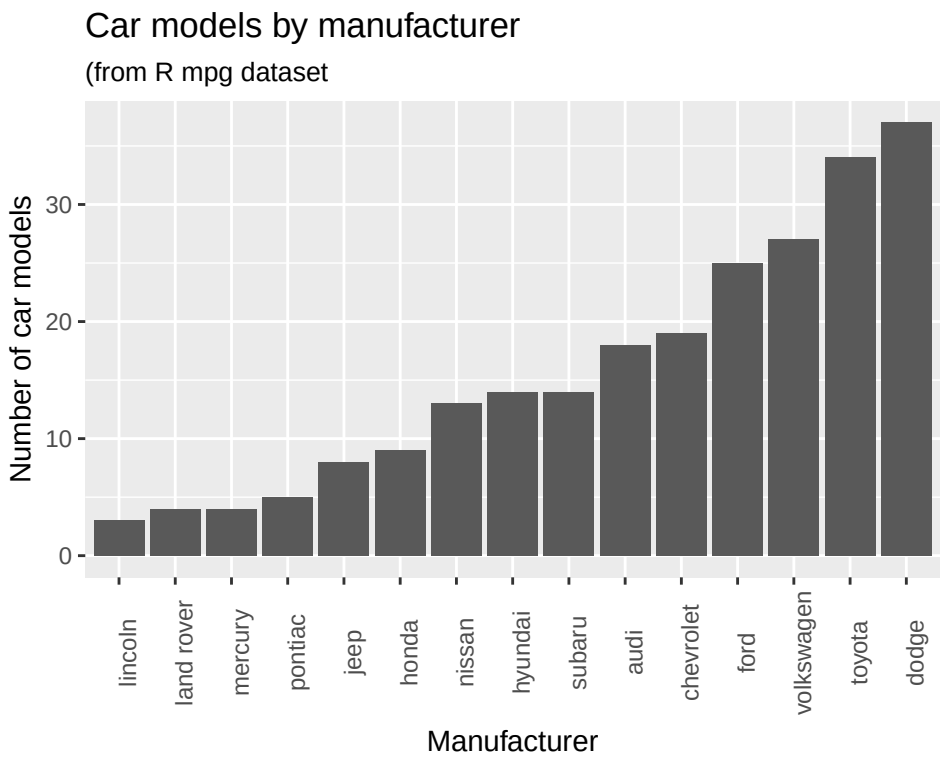

The manufacturers here appear in alphabetical order. It would be interesting to sort the plot so the bars are sorted by the number of car models each manufacturer produces. We can do this by changing the factor levels of the manufacturer column (don't worry about details in the code - it uses the fct_reorder function in the forcats package to sort on the number of cars)...

```
ggplot(mpg, aes(x = forcats::fct_reorder(manufacturer, manufacturer, length))) +
  geom_bar() +
  theme(axis.text.x = element_text(angle = 90))
```




Finally lets tidy up the axis labels and give the plot a title...

```
ggplot(mpg, aes(x = forcats::fct_reorder(manufacturer, manufacturer, length))) +
  geom_bar() +
  theme(axis.text.x = element_text(angle = 90)) +
  labs(x = "Manufacturer",
       y = "Number of car models",
       title = "Car models by manufacturer",
       subtitle = "(from R mpg dataset")
```



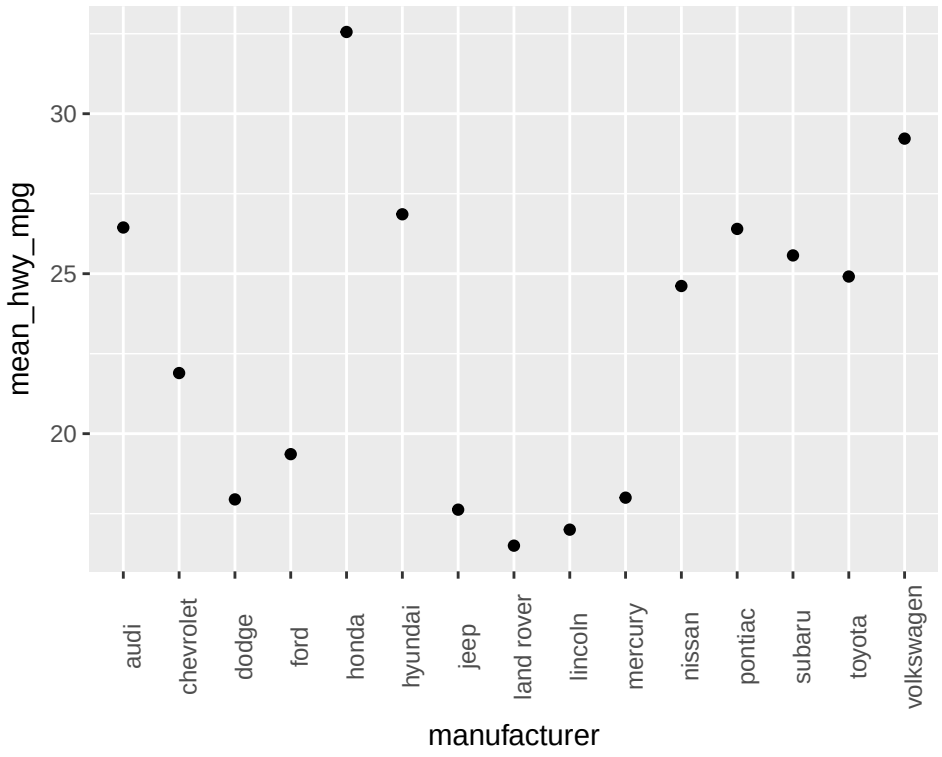

## 5.2 Dot chart

Dot charts are also used to display numerical values for a set of categories. They work well when we wish to truncate an axis and not include zero. We'll show you what we mean by that. First we'll summarise the `mpg` data to make a small dataset that has the mean highway mpg (miles per gallon of fuel) for each manufacturer. You can run the following code to make this dataset (don't follow if you don't understand it - we cover that elsewhere)...

```
mean_mpg <- mpg %>%
  group_by(manufacturer) %>%
  summarise(mean_hwy_mpg = mean(hwy)) %>%
  ungroup()
```

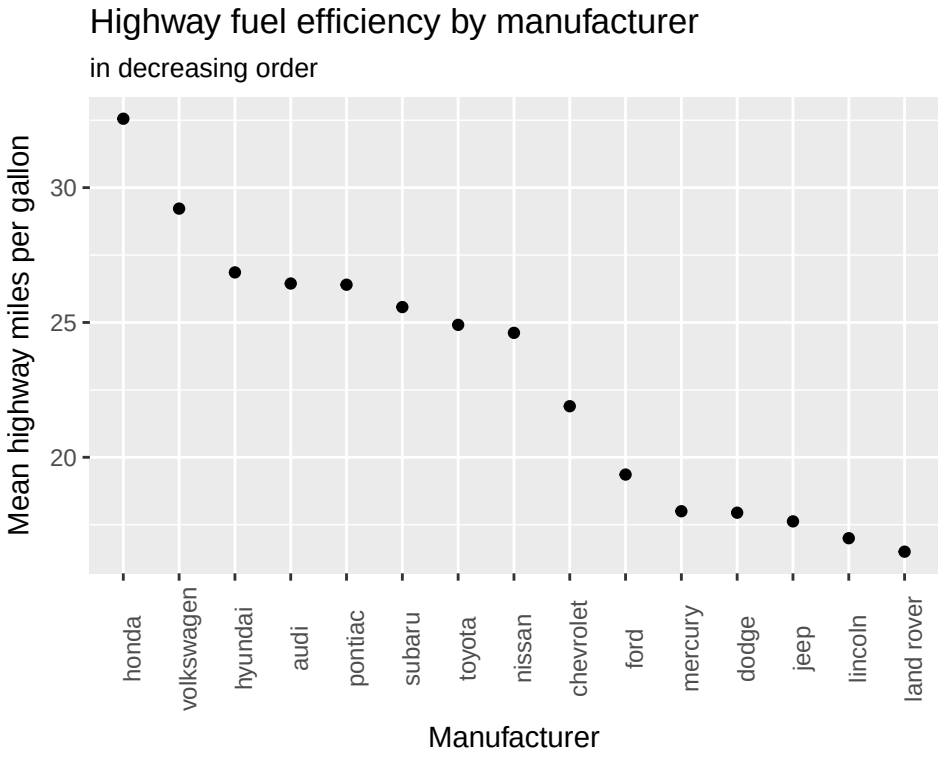
Now we'll plot a dot chart for this data. We'll put the mnaufacture on the `x` axis and mean mpg o the `y` axis using a `geom_point`. We'll also use the `theme` function to rotate the `x` axis labels - like we did for the barchart

```
ggplot(mean_mpg, aes(x = manufacturer, y = mean_hwy_mpg)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90))
```



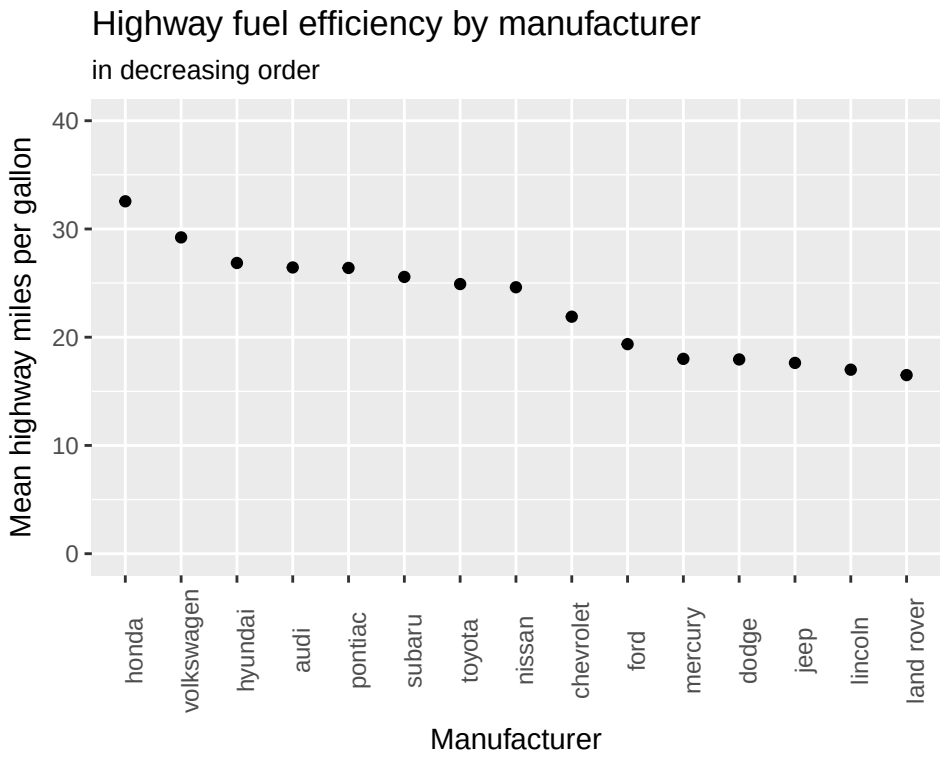

Again it would make sense to sort the manufacturers by the result we are plotting. We'll use simialr code to the the code we used with the barchart but we'll add in `.desc = TRUE` to sort in decresaing order. Finally we'll also add some better axis labels and a title.

```
ggplot(mean_mpg, aes(x = forcats::fct_reorder(manufacturer,
                                              mean_hwy_mpg,
                                              .desc = TRUE),
                                              y = mean_hwy_mpg)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90)) +
  labs(x = "Manufacturer",
       y = "Mean highway miles per gallon",
       title = "Highway fuel efficiency by manufacturer",
       subtitle = "in decreasing order")
```




You'll see that ggplot has automatically truncated the axis to give the clearest comaprison. This is fine with a dot chart. If we included zero we'd lose detail in the data. Here we'll use `last_plot()` as a shortcut to take our last plot and modify it. Adding `ylim(c(0, 40))` fixes the limits of the `y` axis from 0 to 40. It's not as easy to see the difference between the mean fuel efficiencies of the different manufactuers.

```
last_plot() +
  ylim(c(0, 40))
```




## 5.3 Histogram

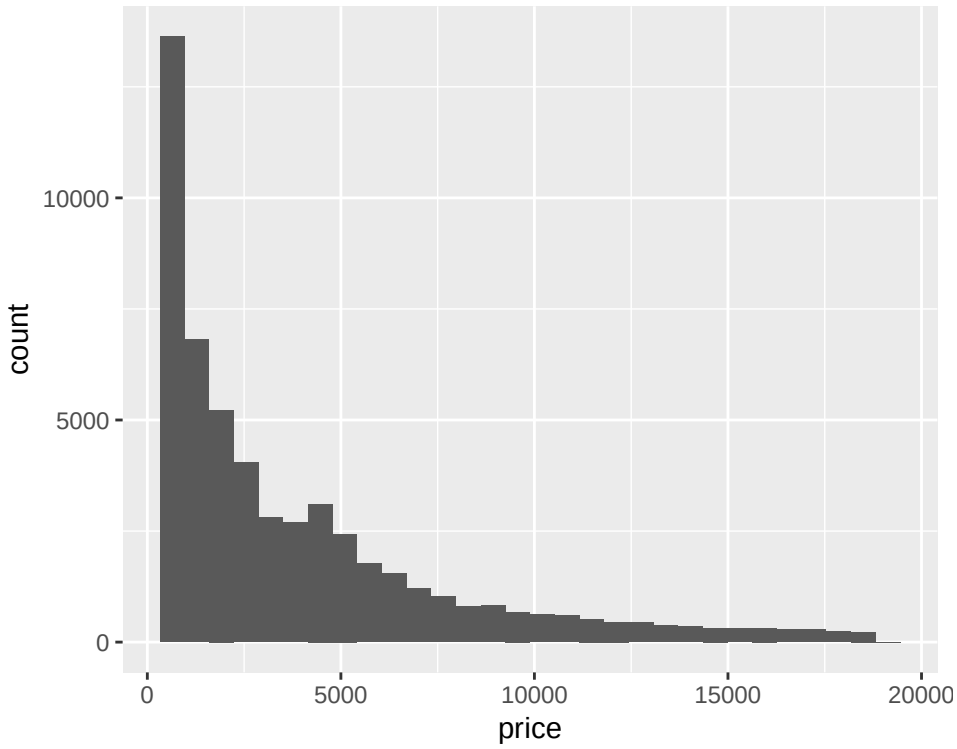
Histograms show a summary of the distribution of a numerical value. In this example we'll use the `diamonds` dataset that's built in to `ggplot` and should be already loaded if you've typed `library(tidyverse)`. First lets look at the dataset...

```
print(diamonds, width = Inf)
```

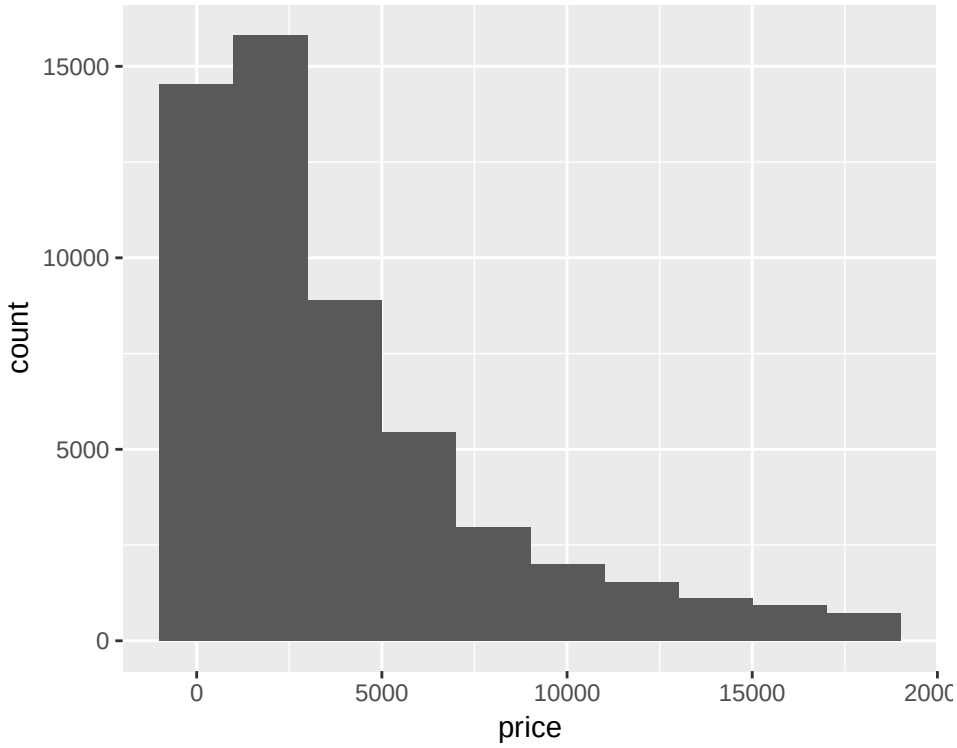
```
## # A tibble: 53,940 x 10
##    carat       cut color clarity depth table price     x     y     z
##    <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
##  2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
##  3  0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
##  4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
##  5  0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good     I    VVS1  62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good     H     SI1  61.9    55   337  4.07  4.11  2.53
##  9  0.22      Fair     E     VS2  65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good     H     VS1  59.4    61   338  4.00  4.05  2.39
## # ... with 53,930 more rows
```

The `price` column records the diamond's price in dollars. Let's plot a basic histogram by mapping the `x` aesthetic to the `price` column and adding `geom_histigram`...

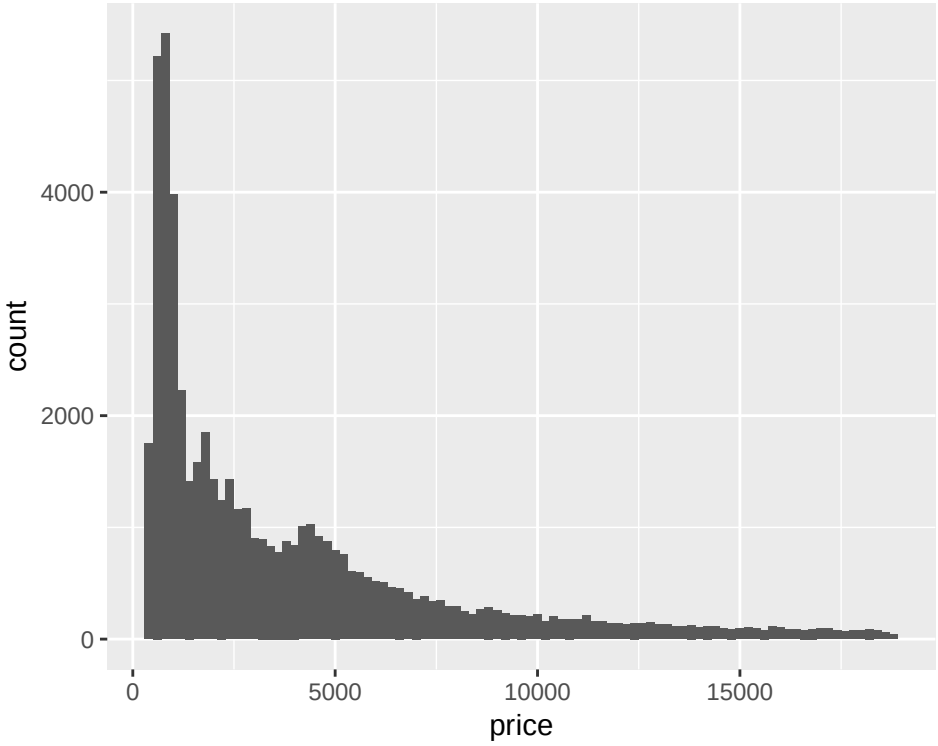
```
ggplot(diamonds, aes(x = price)) +
  geom_histogram()
```




```
ggplot(diamonds, aes(x = price)) +
  geom_histogram(binwidth = 2000)
```

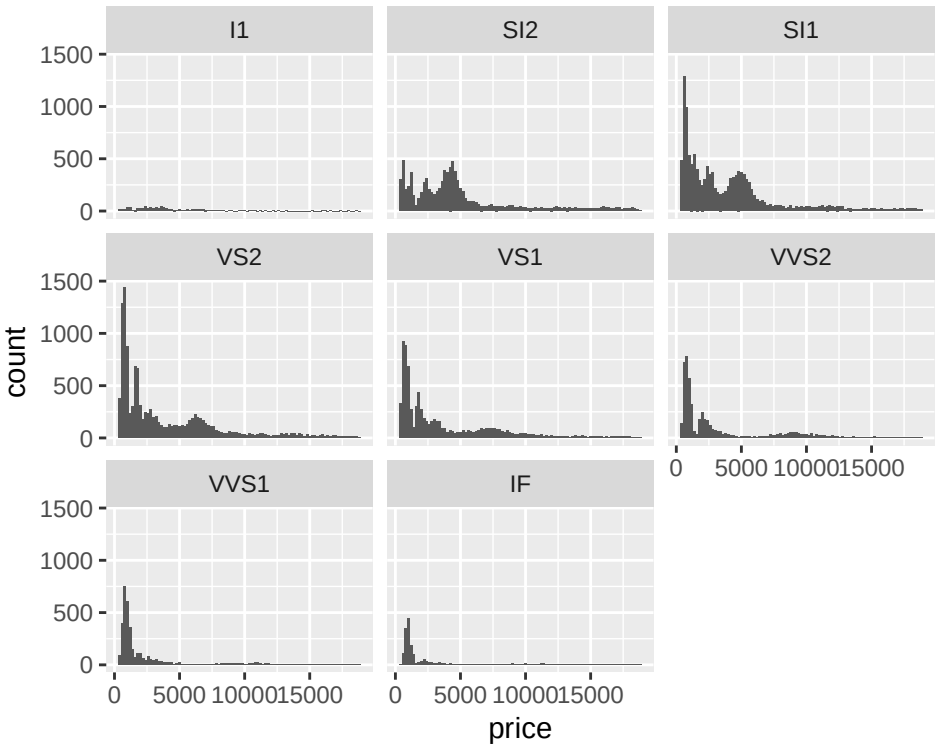



```
ggplot(diamonds, aes(x = price)) +
  geom_histogram(binwidth = 200)
```




Now we have a resonable looking overall histogram we can dig deeper and look at the distribution of prioces within different groups of diamonds. The `clarity` column in the `diamonds` dataset contains a code for, you guessed it, the diamond's clarity. Lets 'facet' the plot by that variable to do a histogram for each clarity class...

```
ggplot(diamonds, aes(x = price)) +
  geom_histogram(binwidth = 200) +
  facet_wrap(~ clarity)
```




## 5.4 Frequency polygon

## 5.5 Scatterplot

## 5.6 Scatterplot with smoother