Introduction to ggplot2

Version 1.1

Table of Contents

Introduction

R is a popular language and environment that allows powerful and fast manipulation of data, offering many statistical and graphical options.

In this course, we will focus on a package called `ggplot2`, which is very versatile when it comes to drawing graphs.

Drawing graphs with *ggplot2*

If you need help with `ggplot2`, you can look at the **R Graphics Cookbook.** A pdf version is available online at:

To draw graphs, you are going to use a package called `ggplot2` which is more powerful and more versatile than `plot()` which comes with the basic version of R. During this course you will also need to download the `plyr` and `reshape2` packages.

The first time you want to use a package, you need to install it. To do so, you type:

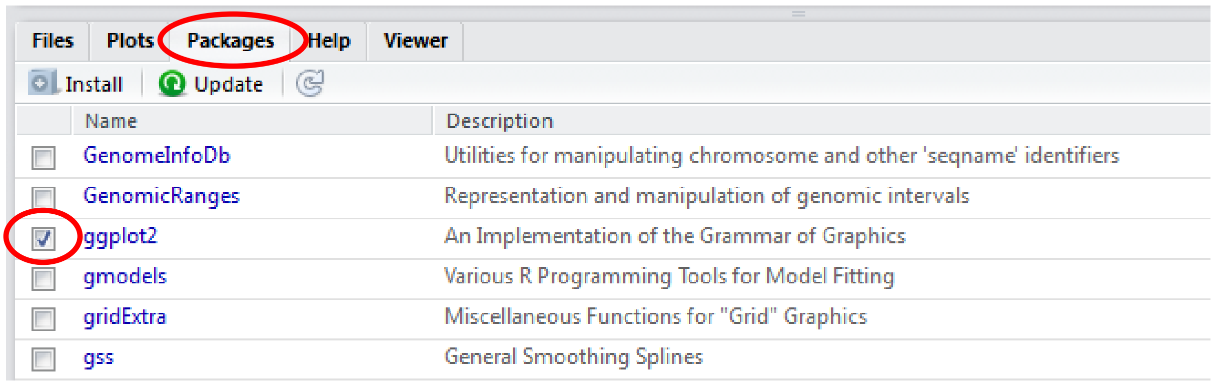
```
install.packages(“ggplot2”)
```

You then need to activate by executing the command:

```
library(ggplot2)
```

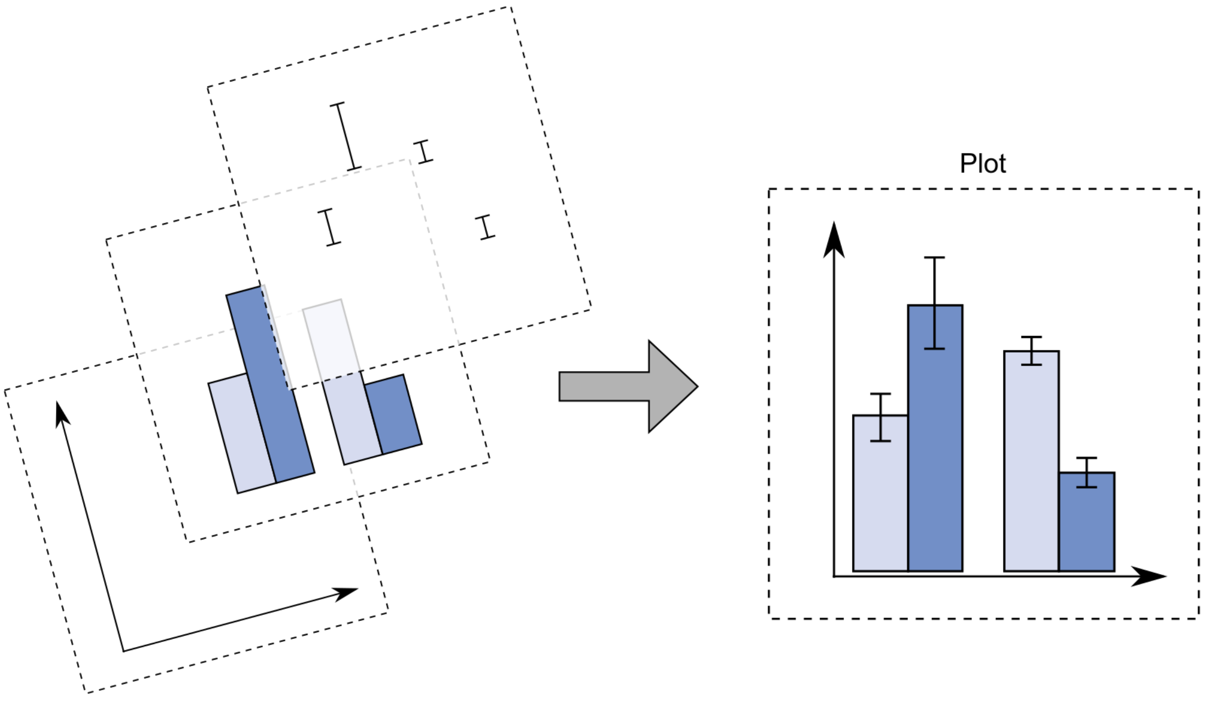
Each time you start an R session you will need to activate the package(s) you need, but you don’t need to install them again. To activate packages, you can either use the command above (`library(ggplot2)`), or click on Packages in the graph panel. The list of downloaded packages will appear and all you have to do is select the tick box next to the one you want.

You can also install packages by clicking on Packages.

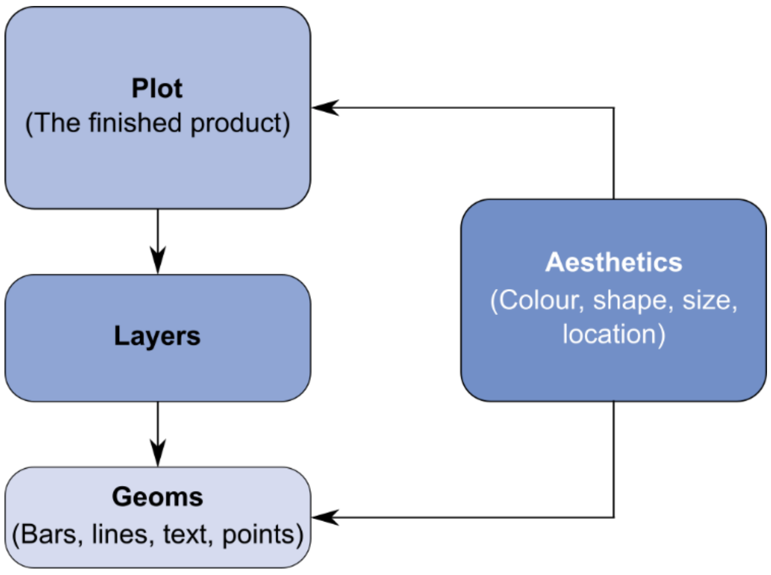


	Name	Description
☐	GenomeInfoDb	Utilities for manipulating chromosome and other 'seqname' identifiers
☐	GenomicRanges	Representation and manipulation of genomic intervals
☑	ggplot2	An Implementation of the Grammar of Graphics
☐	gmodels	Various R Programming Tools for Model Fitting
☐	gridExtra	Miscellaneous Functions for "Grid" Graphics
☐	gss	General Smoothing Splines

In `ggplot2`, a graph is made up of a series of layers.



Visual elements in `ggplot2` are called `geoms` (as in geometric objects: bars, points …) and the appearance and location of these `geoms` (size, colours …) are controlled by the aesthetics properties. Variables that you want to plot to referred to as `aes()`.



Geometric objects (`geom`)

`geom_bar()` : creates a layer with bars representing different statistical properties

`geom_point()` : creates a layer with data points (as on a scatterplot)

`geom_line()` : creates a layer with a straight line

`geom_smooth()` : creates a layer with a 'smoother' (a line that summarizes the data as a whole rather than connecting individual points)

`geom_histogram()` : creates a layer with a histogram

`geom_boxplot()` : creates a layer with a box-whisker diagram

`geom_text()` : creates a layer with text in it

`geom_errorbar()` : creates a layer with error bars in it

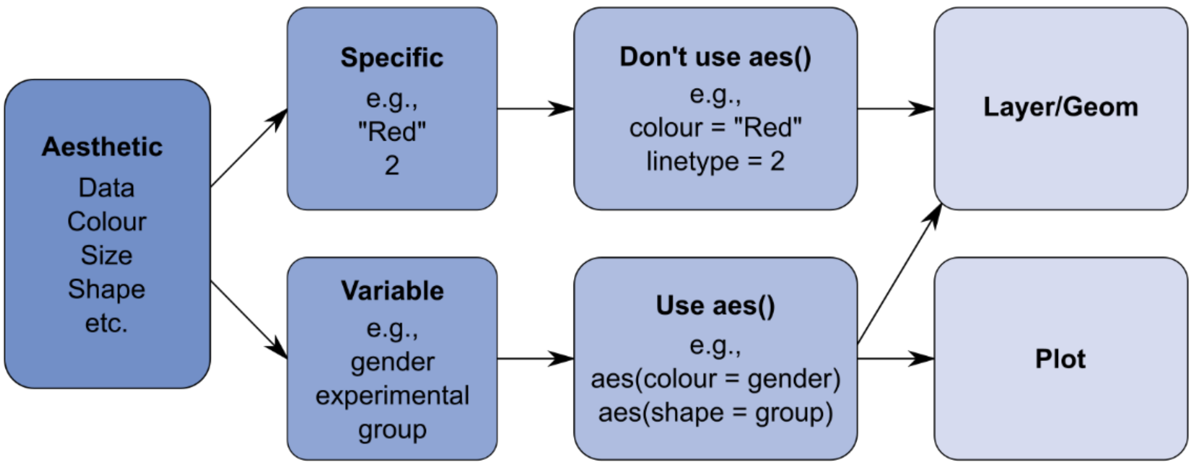
`geom_hline()` and `geom_vline()` : creates a layer with a user-defined horizontal or vertical line respectively.

Between the brackets we need to specify aesthetics:

- required : the variable the `geom` represent
- optional: attributes such as colour, size …

geom	required	optional
`geom_bar()`	x: the variable to plot on the x-axis	colour, size, fill, linetype, alpha (transparency)
`geom_point()`	x: the variable to plot on the x-axis y: the variable to plot on the y-axis	shape, colour, size, fill, alpha
`geom_line()`	x: the variable to plot on the x-axis y: the variable to plot on the y-axis	colour, size, linetype, alpha
`geom_smooth()`	x: the variable to plot on the x-axis y: the variable to plot on the y-axis	colour, size, fill, linetype, alpha
`geom_histogram()`	x: the variable to plot on the x-axis	colour, size, fill linetype, alpha
`geom_boxplot()`	x: the variable to plot on the x-axis	colour, size, fill, alpha

`geom_text()`	x: the horizontal coordinate of where the text should be placed y: the vertical coordinate of where the text should be placed	colour, size, angle, hjust(horizontal adjustment), vjust (vertical adjustment), alpha
`geom_errorbar()`	x: the variable to plot ymin, ymax: lower and upper value for error bar	colour, size, linetype, width, alpha
`geom_hline()` `geom_vline()`	yintercept = value xintercept = value	colour, size, linetype, alpha



The general form of a command will look like this:

```
MyGraph<-ggplot(MyData, aes(variable for x axis, variable for y
axis))+geom()
```

That will work if you are plotting the raw data. If you want to plot means or error bars, then the form of the command will be different as you will see later.

With the command above you have created a graph object but with no graphical elements. You need to add layers, and to do so you just use the symbol '`+`'.

Now, some of these graphs require more than data. For example a box plot requires median, minimum and maximum values, same thing for error bars. The last thing you want to do is having to calculate those as you just want R to draw your graphs from raw data. Luckily, `ggplot2` has built-in functions called '`stats`' which do exactly that.
One last thing `ggplot2` works with a dataframe.

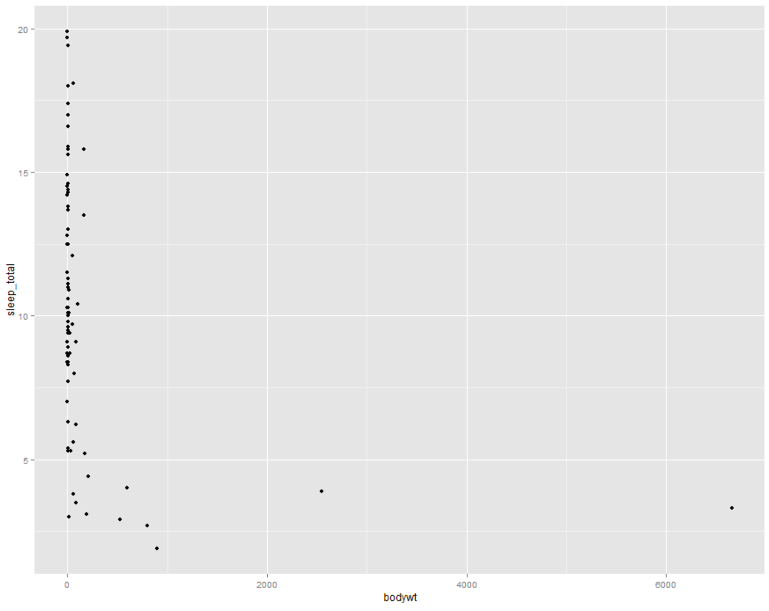
Scatterplot

You are going to plot data from the file: `msleep` which is built-in `ggplot2`. This file contains sleep habits of different animal species. It is a dataframe with 83 rows and 11 variables.

```
msleep
```

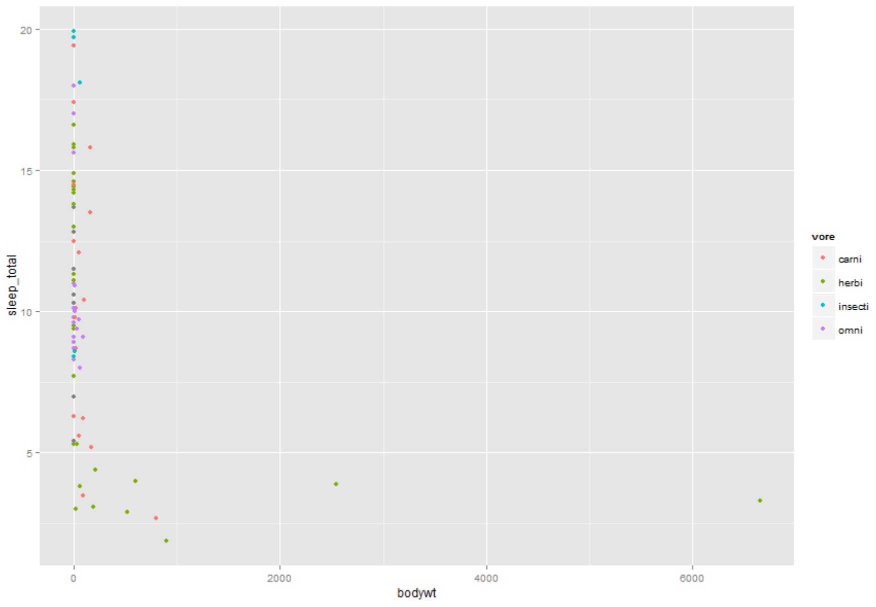
So say you want to look at the relationship between numbers of hours slept per day (`sleep_total`) and the weight of the animal (`bodywt`).

```
scatterplot<-ggplot(data=msleep, aes(x=bodywt, y=sleep_total))+geom_point()
```



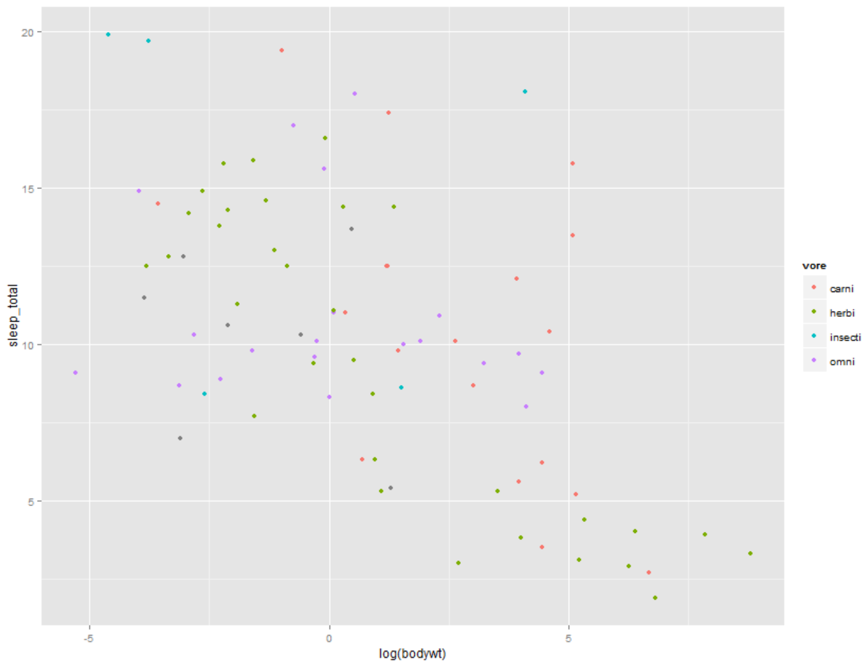
Now to have a different colour for each trophic level:

```
scatterplot<-ggplot(data=msleep, aes(x=bodywt, y=sleep_total,
col=vore))+geom_point()
```



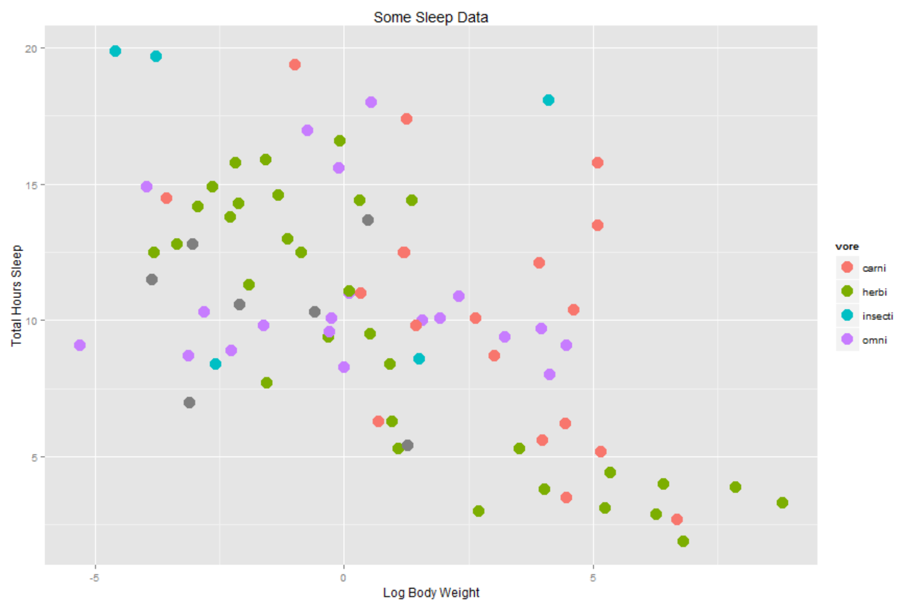
Now, without doing any fancy stats it is pretty obvious that the relationship between the 2 variables is not linear. You can directly transform variables in the `ggplot` call.

```
scatterplot<-ggplot(data=msleep, aes(x=log(bodywt), y=sleep_total,
col=vore)+geom_point()
```



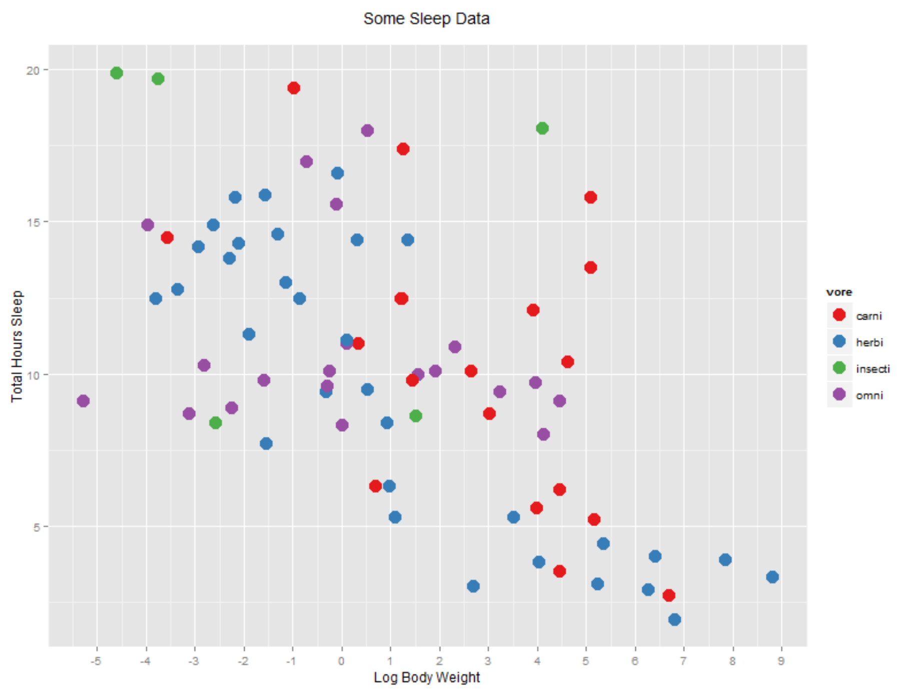
The next changes you can make are more in the ‘cosmetic’ area, like the size of the points and prettier titles for the axes.

```
Scatterplot<-scatterplot+geom_point(size=5)+xlab("Log Body
Weight")+ylab("Total Hours Sleep")+ggtitle("Some Sleep Data")
```



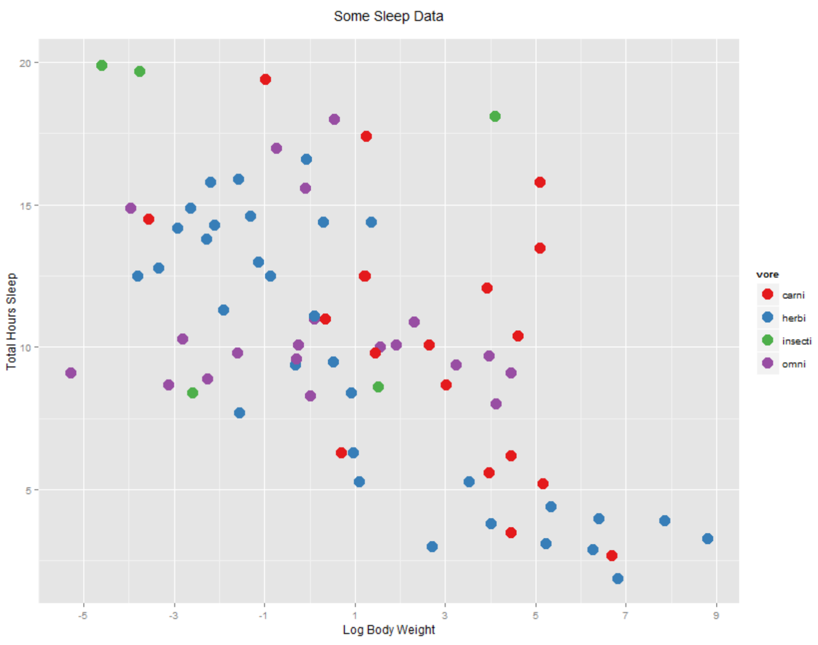
Next you can adjust the tick marks on the x-axis and fiddle with the title position.

```
scatterplot+scale_colour_brewer(palette="Set1")+theme(plot.title=element_te
xt(vjust=+2))+scale_x_continuous(breaks=-5:10)
```



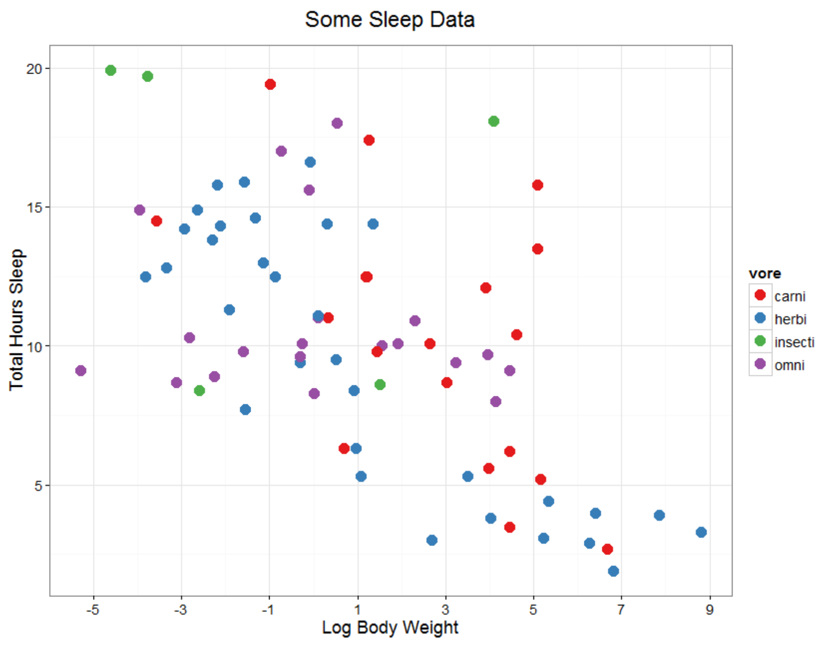
The breaks above are the default, which is 1 unit. If you want to specify the breaks, you could go:

```
+scale_x_continuous(breaks=seq(-5, 10, 2))
```



Finally you may want to change the general theme for `ggplot`. The default is `theme_grey()` but it can be changed to a black and white one. NOTE: once you have changed the theme setting, it will remain changed for the rest of the R session.

```
theme_set(theme_bw(base_size=18))
```



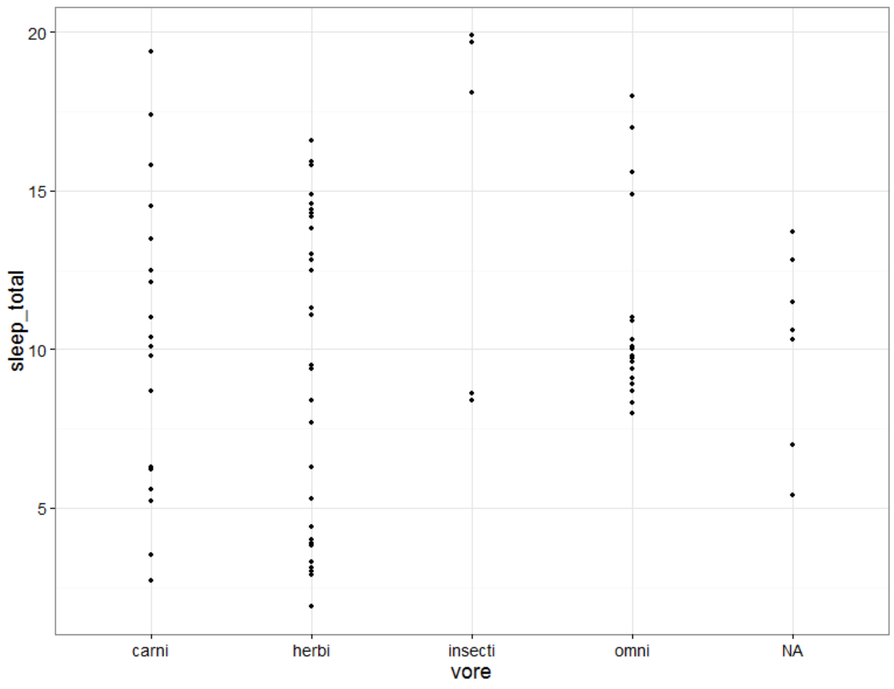
Tadah!

Issue with missing values: if the colour palette is not specified, `ggplot` will plot missing values (NA) in grey by default. If the palette is specified, because NAs are not identified as a category, they are not plotted.

Stripchart

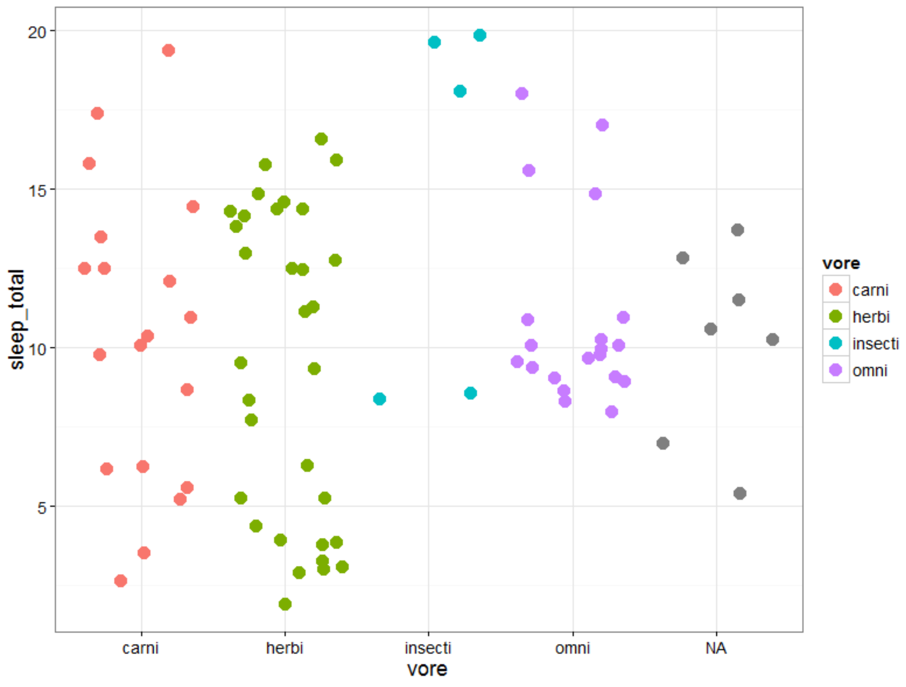
Another way to look at individual values is stripchart. Say you want to look at the average amount of sleep (`sleep_total`) per trophic level (`vore`). The command line below is the basic one to set the graph. You can see that in way of simplification you don't have to specify `data=`, `x=` and `y=`.

```
stripchart<-ggplot(msleep, aes(vore, sleep_total))+geom_point()
```



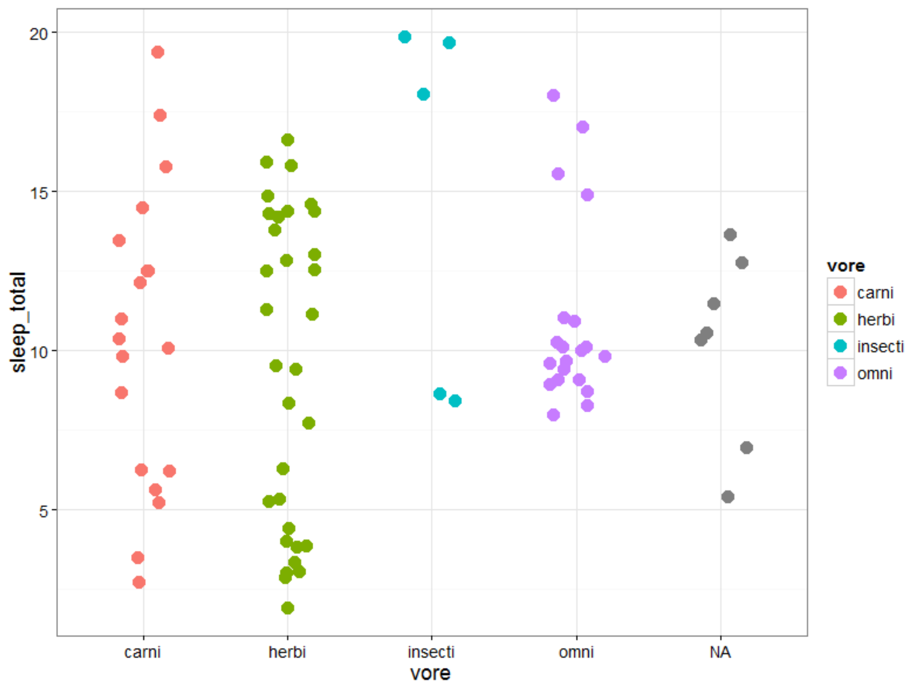
There is definitely room for improvement here. First you can introduce some random noise `(position="jitter")` in the data so that the data points are not of top of each other. You can also add some colour and make the points bigger.

```
stripchart<-ggplot(msleep,aes(vore,sleep_total,col=vore))
+geom_point(size=5,position="jitter")
```



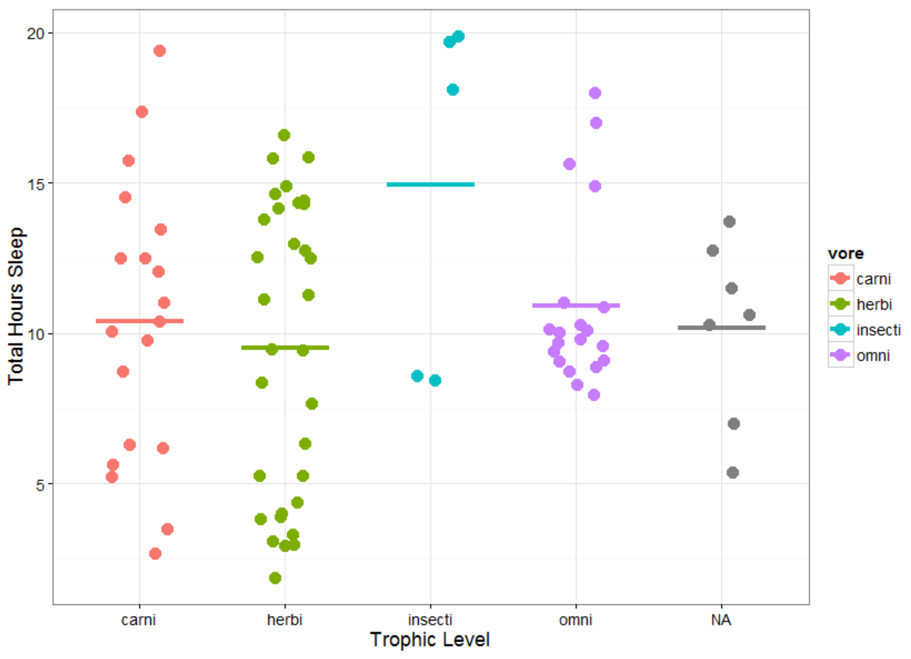
Depending on the sample size and the number of categories, the default might work but in this case, we want to choose the width of the jitter. To do that, you use a less usual `geom` called `geom_jitter`. To be able to manipulate the jitter, we need to make it a layer so it has to become a proper `geom`. Because the introduction of random noise is supposed to be applied on points, the `geom_point` is not needed anymore.

```
stripchart<-ggplot(msleep, aes(vore,sleep_total,col=vore))
+geom_jitter(position = position_jitter(width = 0.2),size=5)
stripchart
```



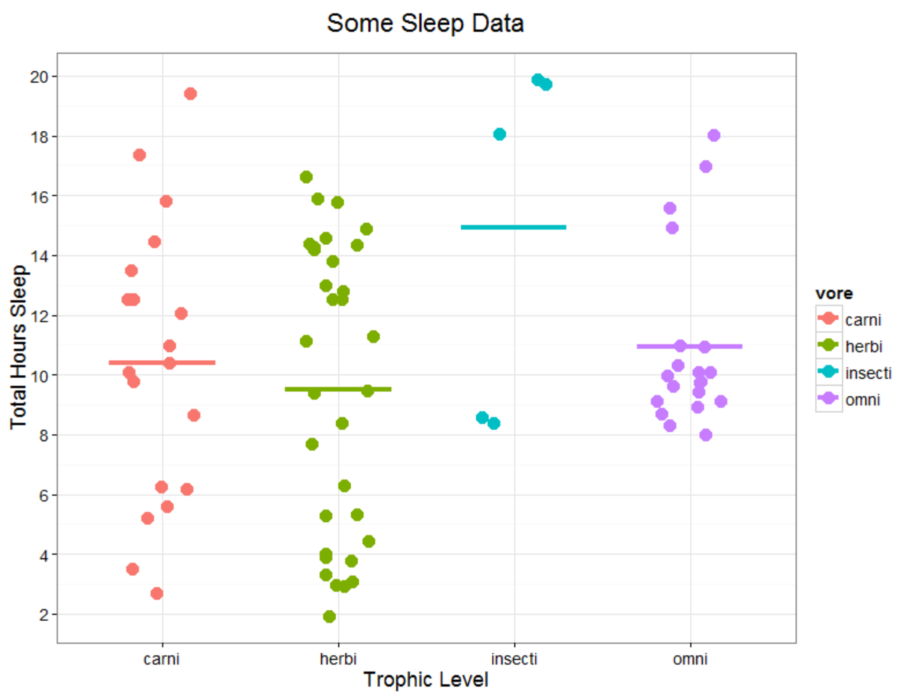
Now the next thing would be to plot the mean of each trophic level. For that you are going to use the stats behind `geom_errorbar`. It is primarily designed for error bars but it also calculates the mean which is what you are after here. `geom_hline` would also work but it would be much more cumbersome in this context. By setting `ymax` and `ymin` at the intercept (`..y..` in `ggplot` language), you are basically asking `ggplot` to only draw a horizontal line where the y coordinate is the mean. The size is for the thickness of the line and the width for its length.

```
stripchart<-stripchart+geom_errorbar(stat="hline", yintercept="mean",
width=0.6, aes(ymax=..y.., ymin=..y..), size=2) +ylab("Total Hours
Sleep")+xlab("Trophic Level")
```



Finally you can remove the NA category, add a title and play with the y-axis.

```
stripchart<-stripchart+scale_x_discrete(limits
=c("carni","herbi","insecti","omni"))
stripchart+ggtitle("Some Sleep Data")
+theme(plot.title=element_text(vjust=+2))+scale_y_continuous(breaks=seq(0,
20, 2))
```

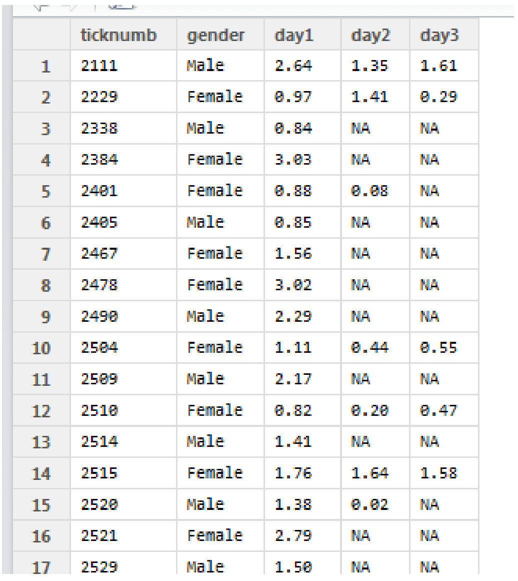


Ok so now, you are going to use another data file to explore other graphs: histogram, boxplots, violin plot (=bean plots), bar charts and line graphs.

The file is called `DownloadFestival.dat` and it contains the level of hygiene (measured as a score) of 810 concert-goers over 3 days (!).

```
setwd("wherever/is/the/file")
festival.data<-read.table("DownloadFestival.dat", sep="\t", header=T)
```

```
View(festival.data)
```

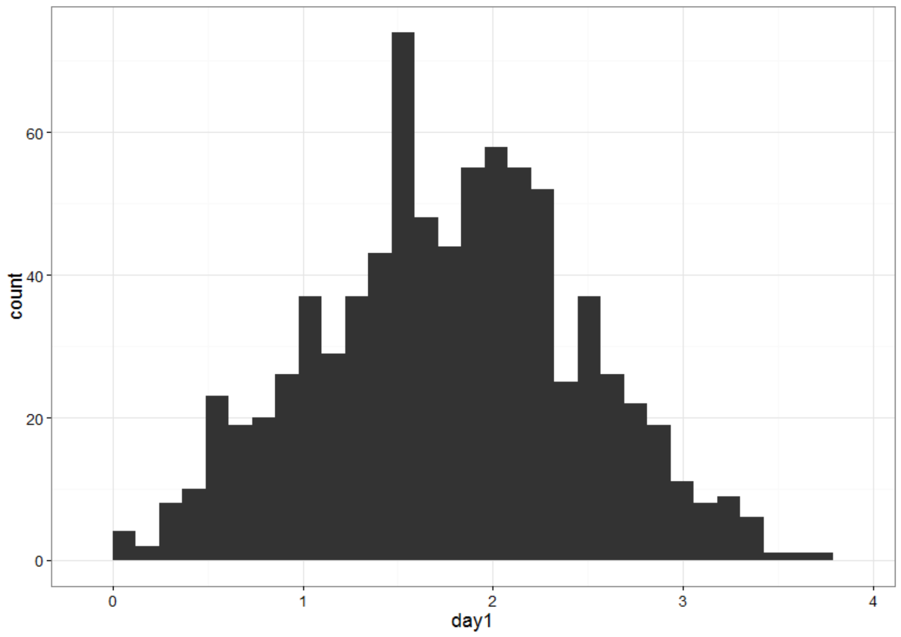


	ticknumb	gender	day1	day2	day3
1	2111	Male	2.64	1.35	1.61
2	2229	Female	0.97	1.41	0.29
3	2338	Male	0.84	NA	NA
4	2384	Female	3.03	NA	NA
5	2401	Female	0.88	0.08	NA
6	2405	Male	0.85	NA	NA
7	2467	Female	1.56	NA	NA
8	2478	Female	3.02	NA	NA
9	2490	Male	2.29	NA	NA
10	2504	Female	1.11	0.44	0.55
11	2509	Male	2.17	NA	NA
12	2510	Female	0.82	0.20	0.47
13	2514	Male	1.41	NA	NA
14	2515	Female	1.76	1.64	1.58
15	2520	Male	1.38	0.02	NA
16	2521	Female	2.79	NA	NA
17	2529	Male	1.50	NA	NA

Now let's say you want to look at the distribution of the values in day 1.

Histogram

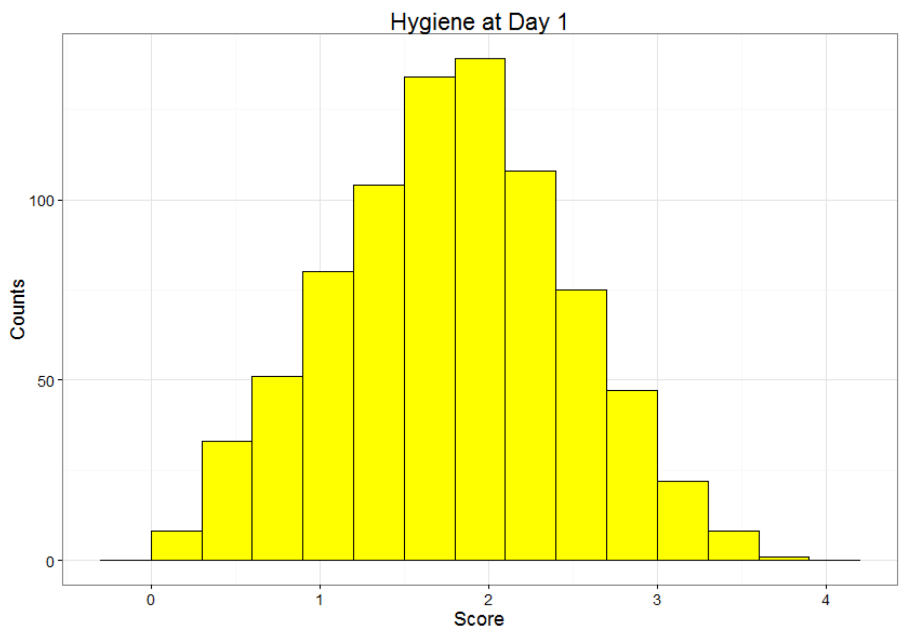
```
Day1Histogram <- ggplot(festival.data, aes(day1))+geom_histogram()
```



There are several things to play with:

- the width of the bins: `+geom_histogram(binwidth=0.3)`
- the colour of the bins: `+geom_histogram(color="black", fill="yellow")`
- labels for the axis: `+labs(x="Score", y="Counts")`
- title of the graph: `+ggtitle("Hygiene at Day 1")`

```
Day1Histogram <- ggplot(festival.data, aes(day1))+geom_histogram()
Day1Histogram <- Day1Histogram
+geom_histogram(binwidth=0.3,color="black",fill="yellow")
Day1Histogram<- Day1Histogram +labs(x="Score", y="Counts")+ggtitle("Hygiene
at Day 1")
```



You can find a list of colours: http://sape.inf.usi.ch/quick-reference/ggplot2/colour

Now if you want to look at the 3 days and the 2 genders at the same time, you will have to restructure the file to be able to use the `facet()` function. To do that you need to download a package called `reshape2`.

Once `reshape2` has been downloaded and activated, before plotting the graph, you need to use the `melt()` function which will basically restructure the file or, as the help (`?melt`) puts it 'melt an object into a form suitable for easy casting'. You need to tell R that you want to stack the day's variable but we want to keep the '`ticknumb`' and the '`gender`' as identifier.

```
festival.data.stack<-melt(festival.data, id = c("ticknumb","gender"))
```

Or:

```
festival.data.stack<-melt(festival.data, id = 1:2)
```

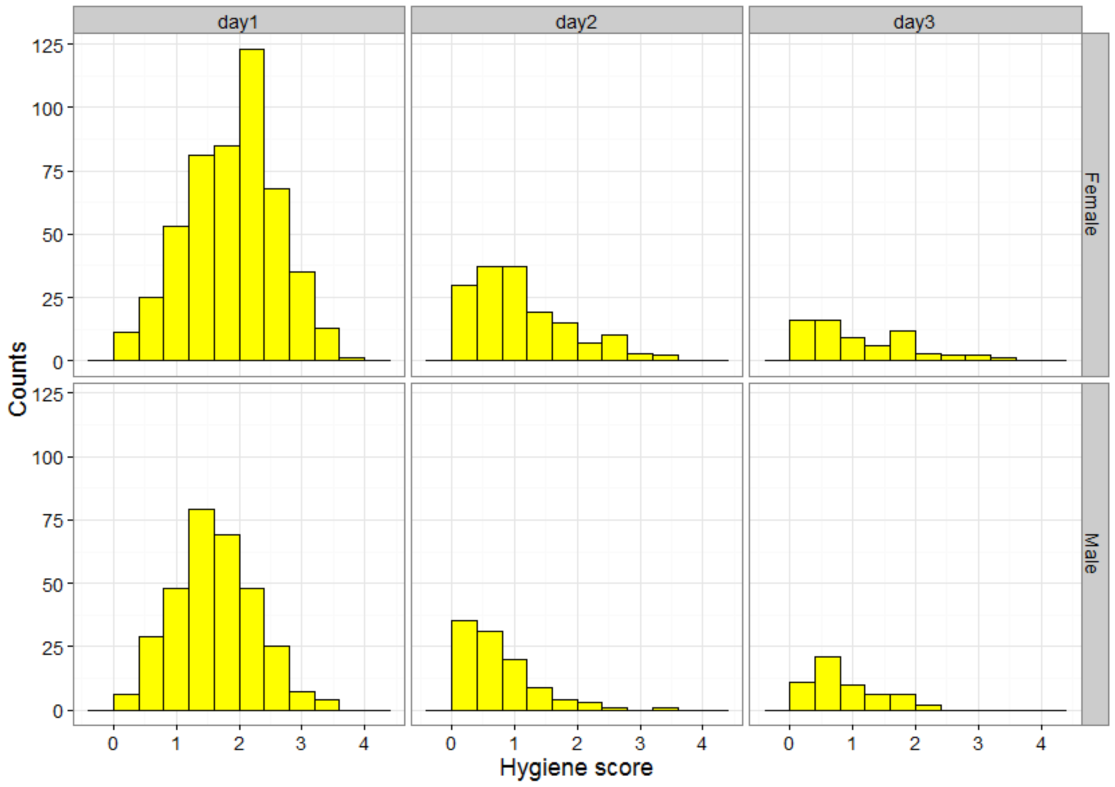
To rename the column which contains the days:

```
colnames(festival.data.stack)[3:4]<-c("day",”score”)
```

Now the data are ready, you can apply the `facet()` function. It comes in 2 flavours: `facet_grid()` or `facet_wrap()`. Today you will use the first one.

```
Histogram.3days<-ggplot(festival.data.stack,aes(score))
+geom_histogram(binwidth=0.4, color="black", fill="yellow")
+labs(x="Score", y="Counts")
+facet_grid(gender~day)
Histogram.3days
```

Basically you recognise the line you typed earlier, and you have simply added the `facet()` function and asked for the rows as gender and the columns as days.



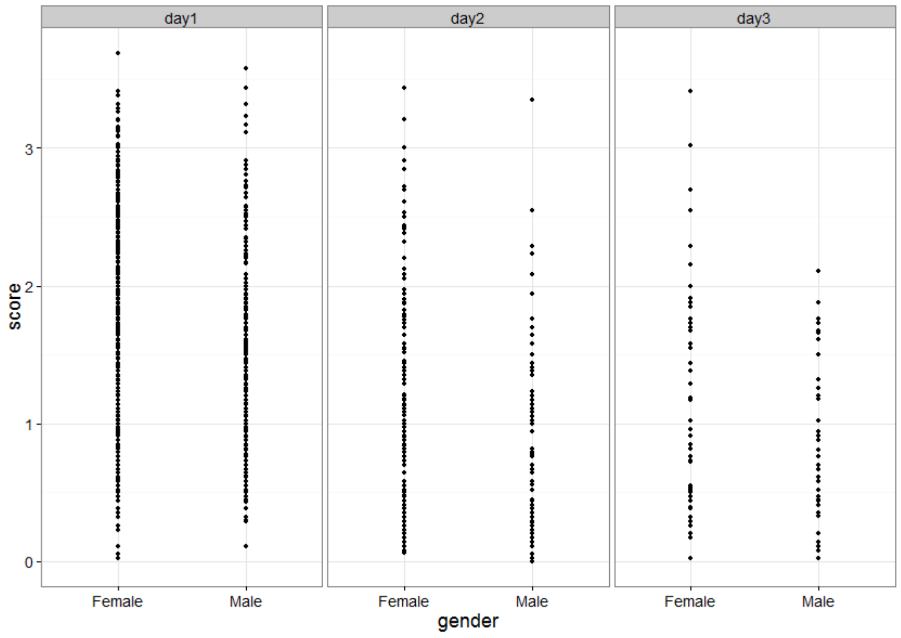
Before moving on with new graphs, let's try to plot these data as a stripchart. One thing we have to account for this time, is the function *mean* which we use to draw the horizontal line. This function does not like missing values (default: `na.rm = FALSE`) and we have such values in our file (NA). One of the ways to deal with it, is to remove them altogether.

```
festival.data.stack<-
festival.data.stack[!is.na(festival.data.stack$score),]
```

Stripchart

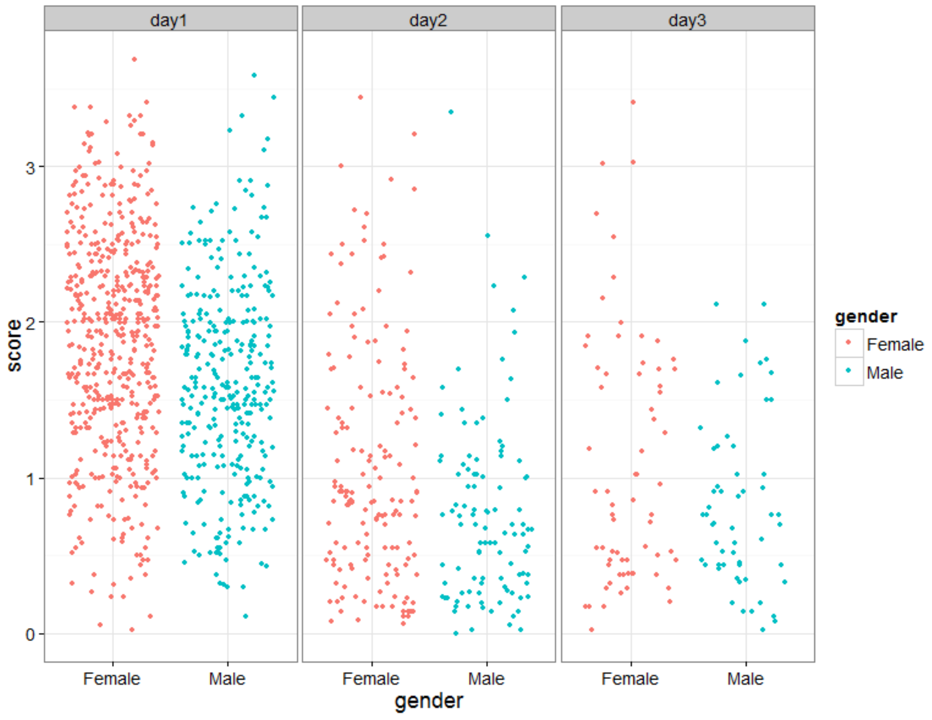
Basic:

```
Scatterplot<-ggplot(festival.data.stack,aes(gender,score))+geom_point()
+facet_grid(~day)
Scatterplot
```



We can definitely improve on that:

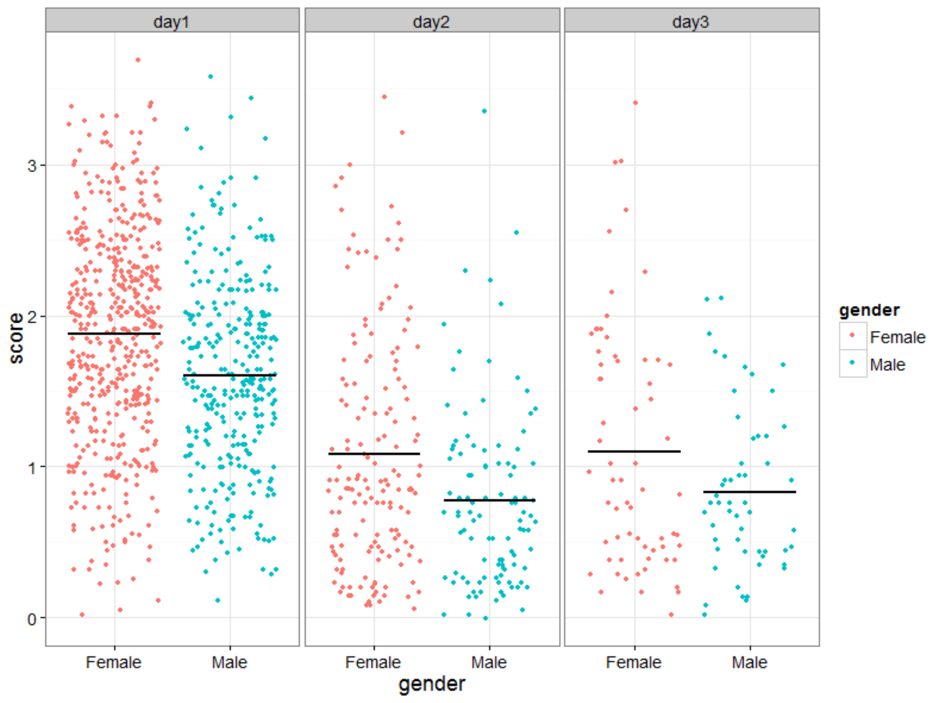
```
Scatterplot<-ggplot(festival.data.stack, aes(gender, score,
colour=gender))+geom_point(position="jitter")+facet_grid(~day)
```



And, like before, we can add a line for the mean:

```
Scatterplot<-ggplot(festival.data.stack, aes(gender, score,
col=gender))+geom_point(position="jitter")+facet_grid(~day)
+geom_errorbar(stat= "hline", yintercept="mean", width=0.8, colour="black",
aes(ymax=..y.., ymin=..y..))
```

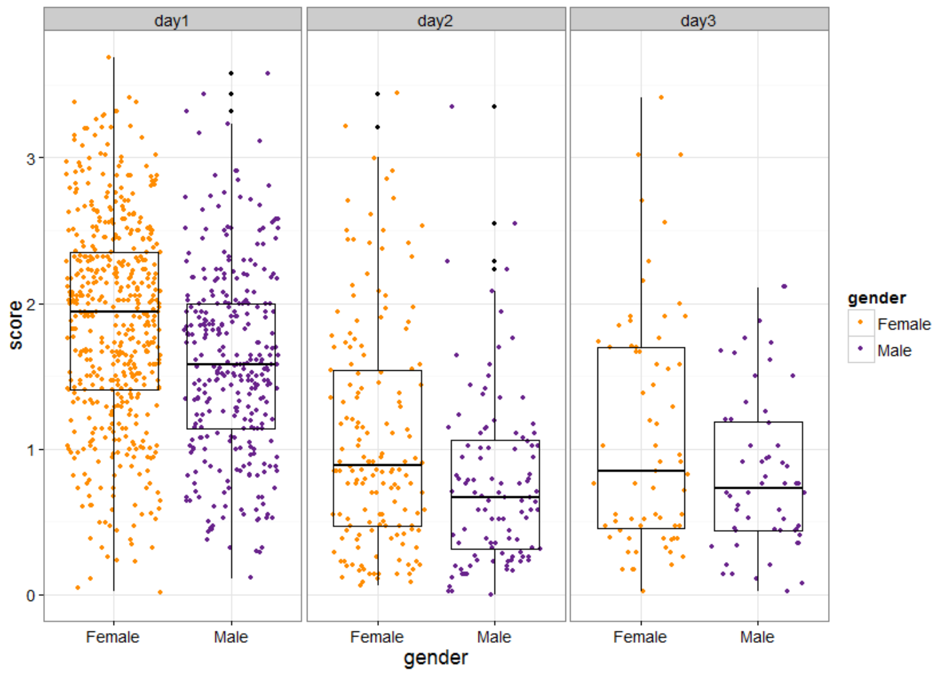
As explained before, with the command above, we are basically telling R that we want to plot error bars (`geom_errorbar`) but as a horizontal line (`hline`) which y-coordinate will be “mean” and the error bars themselves will also be the mean y.



One last thing, we can choose to overlay a boxplot on the stripchart rather than draw the mean. We can also choose other colour than the default while we are at it.

```
Scatterplot<-ggplot(festival.data.stack,aes(gender,score, colour=gender))
+geom_point(position="jitter")+facet_grid(~day)+scale_colour_manual(values=
c("darkorange", "darkorchid4"))
Scatterplot+geom_boxplot(alpha=0, colour="black")
```

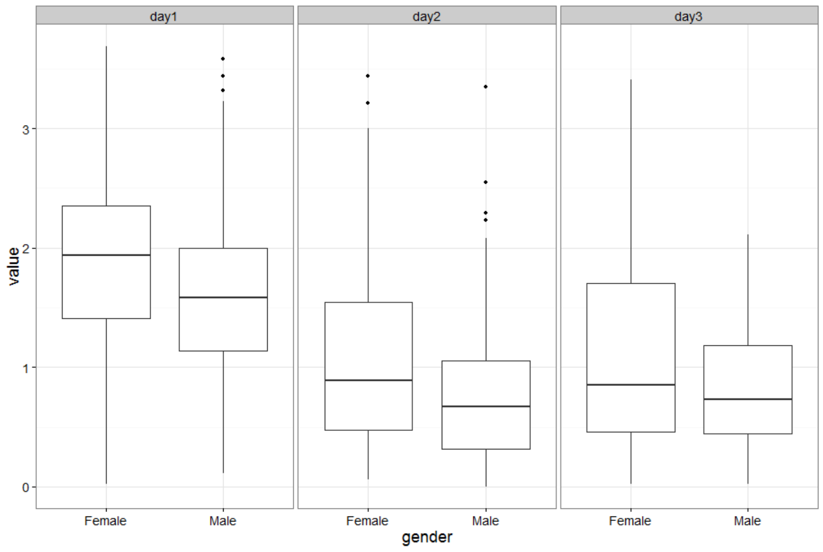
For the boxplot to be the top layer and the points to be still visible, we need to make the boxplots transparent (`alpha=0)`.



Boxplots

Let's keep on drawing boxplots.

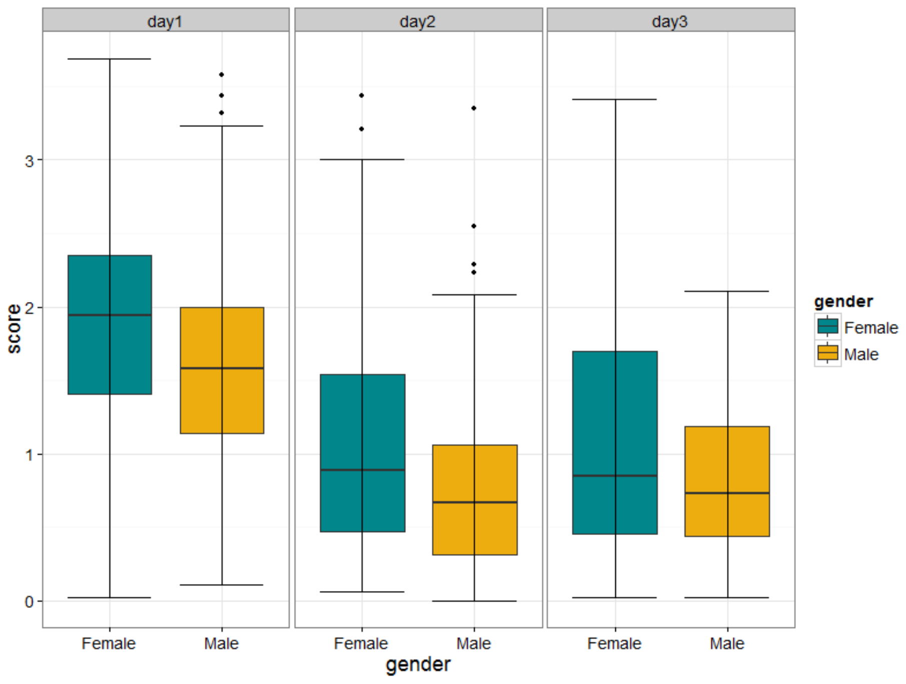
```
Boxplot<-ggplot(festival.data.stack, aes(gender,score))+geom_boxplot()
 +facet_grid(~day)
```



The black dots are 'official' outliers but we will not deal with them today.

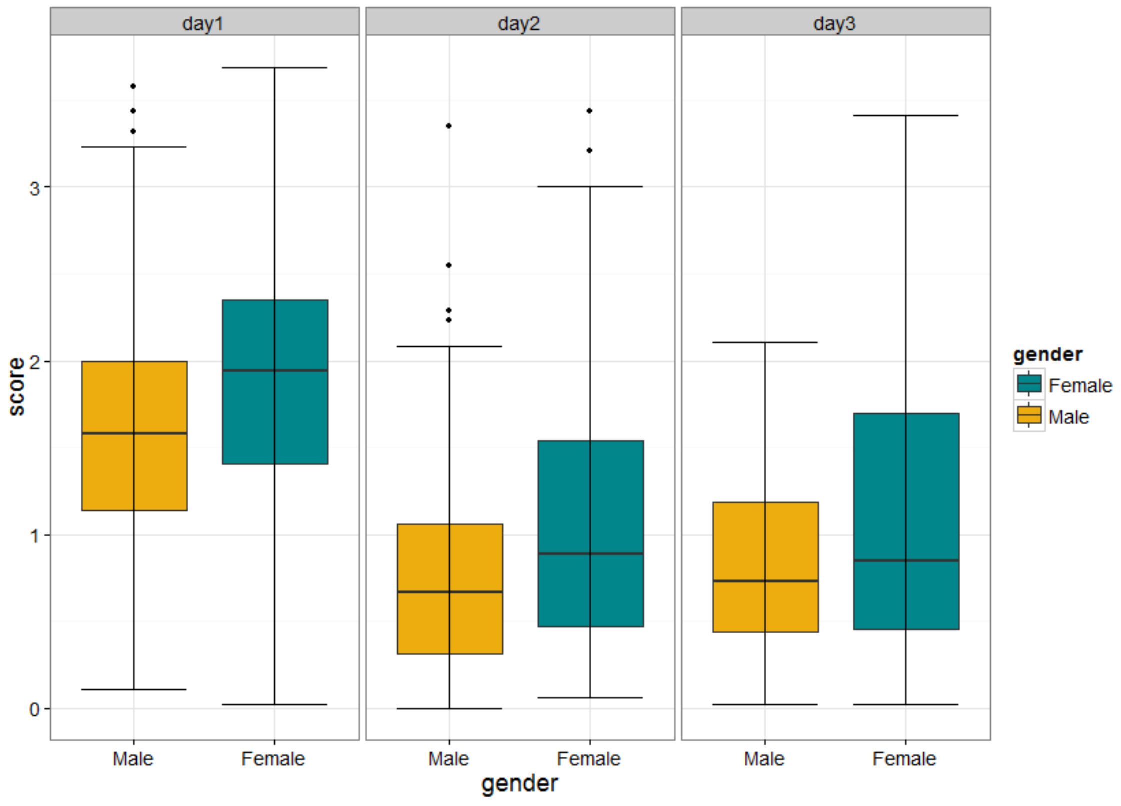
If you want to make the boxplots look prettier, like changing the look of the whiskers for instance, you can do:

```
Boxplot<-ggplot(festival.data.stack, aes(gender,score, fill=gender)
+geom_boxplot() +stat_boxplot(geom="errorbar")
+scale_fill_manual(values=c("turquoise4","darkgoldenrod2"))
+facet_grid(~day)
```



If you want to change the order of the levels on the x-axis:

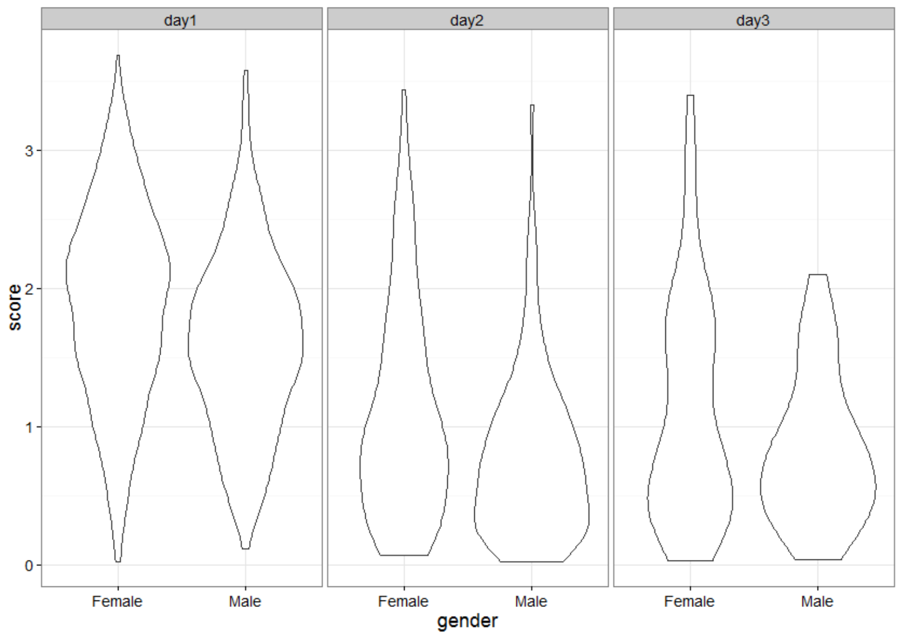
```
Boxplot+scale_x_discrete(limits=c("Male","Female"))
```



Violin/bean plot

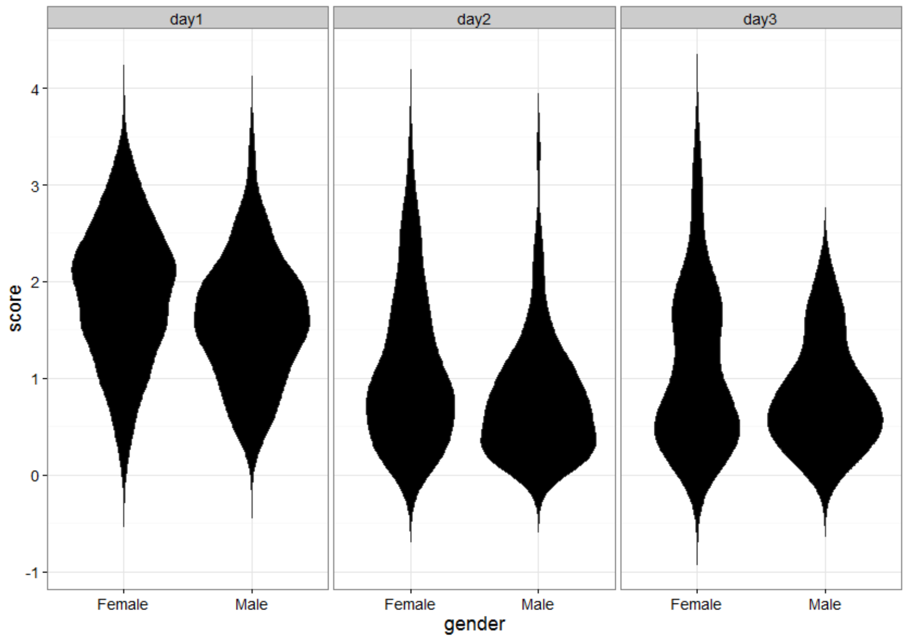
Violin/bean plots are a variation on the theme of the boxplot. They have the same structure but data density is mirrored by the shape of the 'bean'. Plus they can look really cool!

```
Violinplot<-ggplot(festival.data.stack, aes(gender,score))
+geom_violin()+facet_grid(~day)
Violinplot
```



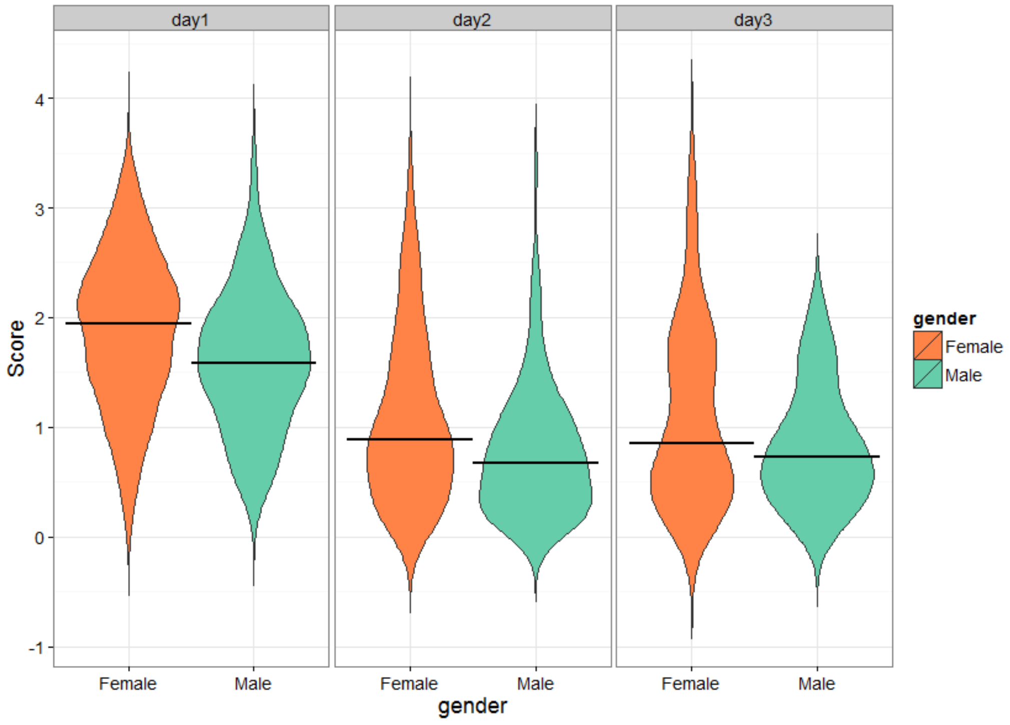
As usual, there is plenty of room for improvement here. One thing that you may want to remove is the default setting for trimming, which trims the tails of the violins to the range of the data.

```
Violinplot<-ggplot(festival.data.stack, aes(gender,score))+geom_violin(trim
= FALSE, fill="black")+facet_grid(~day)
Violinplot
```



Finally, you can play a little with the violin plots by adding a line for the median and changing the colours.

```
Violinplot<-ggplot(festival.data.stack, aes(gender,score, fill=gender))
+geom_violin(trim = FALSE)+facet_grid(~day)
Violinplot<-Violinplot+geom_errorbar(stat= "hline", yintercept="median",
width=0.9, aes(ymax=..y.., ymin=..y..), size=1,colour="black")
Violinplot+scale_fill_manual(values=c("sienna1", "aquamarine3"))
```



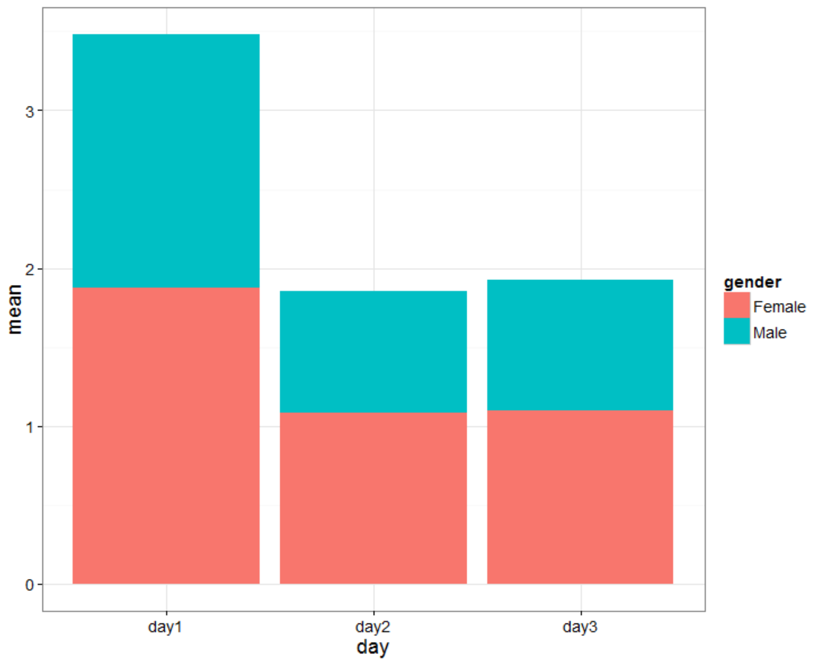
Bar charts

The graphs above are very useful if you want to explore your data, but then you might want to plot them in a more 'classical' way, for example in a bar chart.
The graph is a little bit more cumbersome to draw as you actually have to calculate the values you want to plot: means and standard errors. To do that you can use the function `ddply` from the `plyr` package.

```
score.sem<-ddply(festival.data.stack,c("gender","day"),
summarise,mean=mean(score), sem=sd(score)/sqrt(length(score)))
score.sem
```

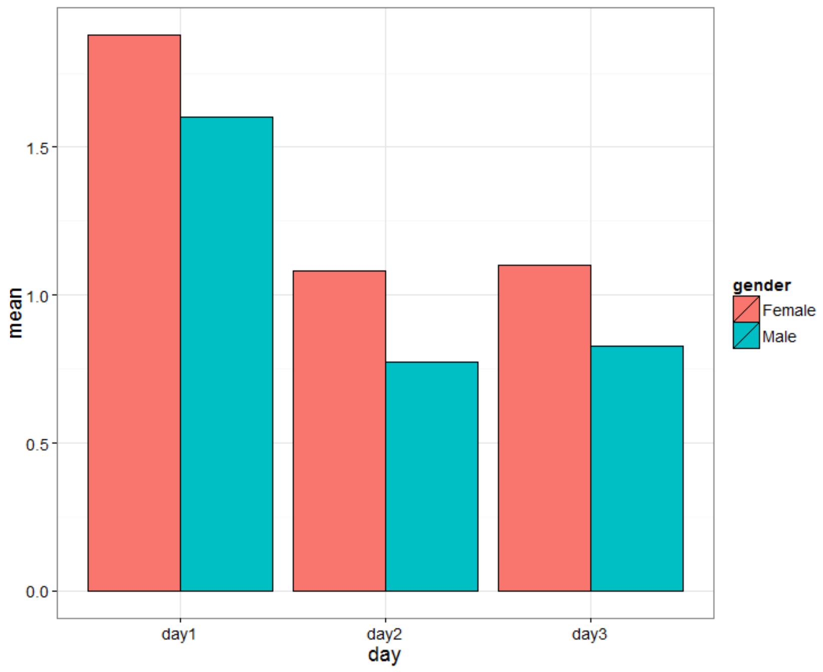
```
  gender  day      mean        sem
1 Female day1 1.8787273 0.03164061
2 Female day2 1.0828750 0.06077612
3 Female day3 1.0997015 0.09895861
4   Male day1 1.6020635 0.03619580
5   Male day2 0.7732692 0.05847218
6   Male day3 0.8291071 0.07209944
```

```
BarFestival<-ggplot(score.sem, aes(day,mean,
fill=gender))+geom_bar(stat="identity")
BarFestival
```



This a pretty ugly graph and quite useless as well as the genders on stacked on one another. What you want is the bars to be next to each other (**`position="dodge"`**).

```
BarFestival<-ggplot(score.sem, aes(day,mean, fill=gender))
+geom_bar(position="dodge", colour="black”)
BarFestival
```

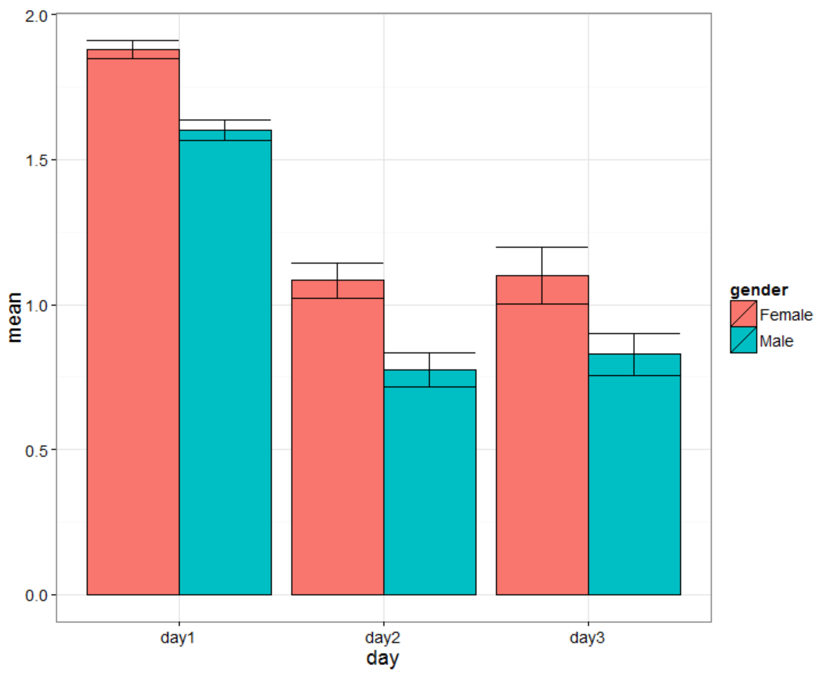


Now you are getting there, all that you need now are error bars (`geom_errorbar`).

Again for this `geom`, the default is ‘`stack`’ for the position of the error bars which is not very useful so you need to specify `position=”dodge”`.

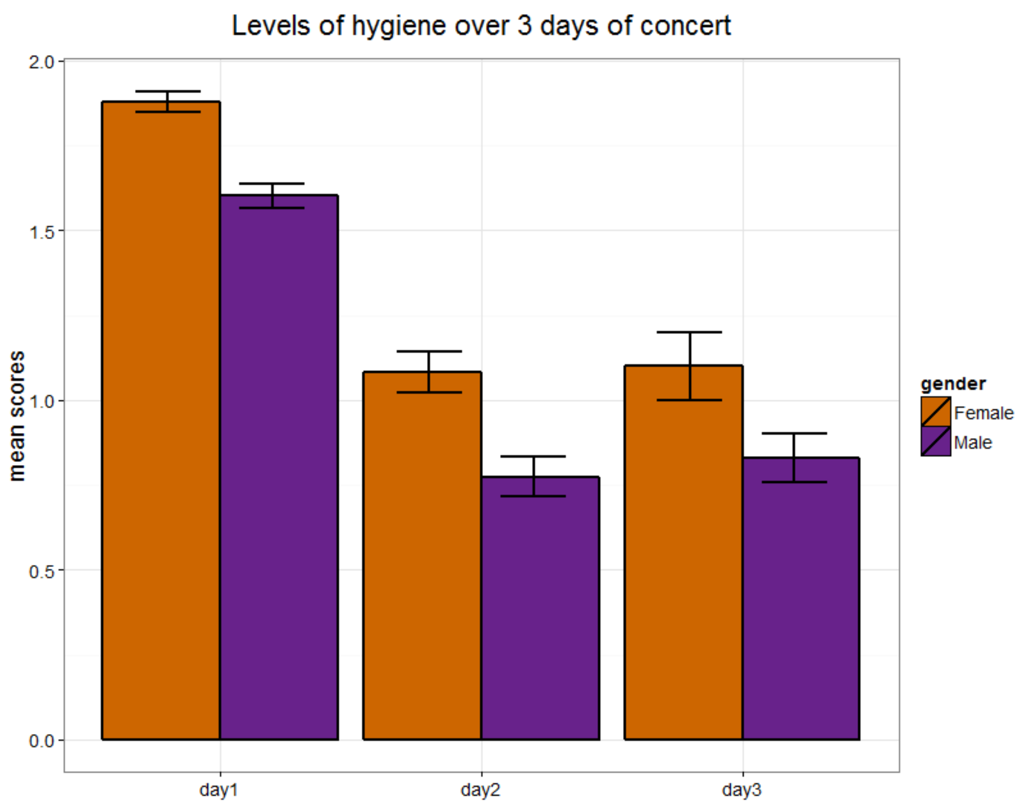
Also `geom_errorbar` needs to know with which stats it is working, so you need to specify in the `geom_bar stat=”identity”` as in it is taking the values as they are from the file and not calculating anything with it like mean or median. If you don’t specify it, it will work but it might mess things up later on.

```
BarFestival<-ggplot(score.sem,aes(day,mean, fill=gender))
+geom_bar(position="dodge", colour="black",stat="identity")
+geom_errorbar(aes(ymin=mean-sem, ymax=mean+sem), position="dodge")
BarFestival
```



Not bad but it could be better.

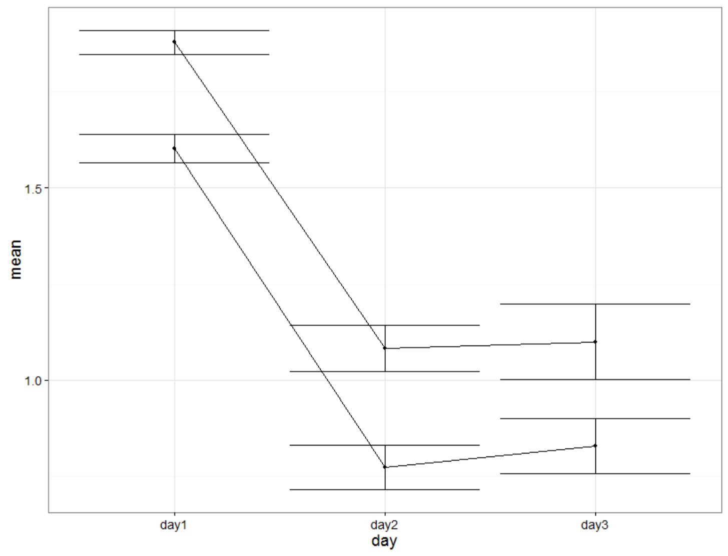
```
BarFestival<-ggplot(score.sem, aes(day,mean, fill=gender))
+geom_bar(position="dodge", colour="black",stat="identity",size=1)
+geom_errorbar(aes(ymin=mean-sem, ymax=mean+sem),
width=.5,position=position_dodge(width=0.8),size=1)
Barfestival<-BarFestival+ ylab("mean scores") + ggtitle("Levels of hygiene
over 3 days of concert")+theme(axis.title.x=element_blank())
Barfestival+scale_fill_manual(values=c("darkorange3", "darkorchid4"))
+theme(plot.title=element_text(vjust=+2))
```


Levels of hygiene over 3 days of concert
mean scores
2.0
1.5
1.0
0.5
0.0
day1
day2
day3
gender
Female
Male

Line graphs

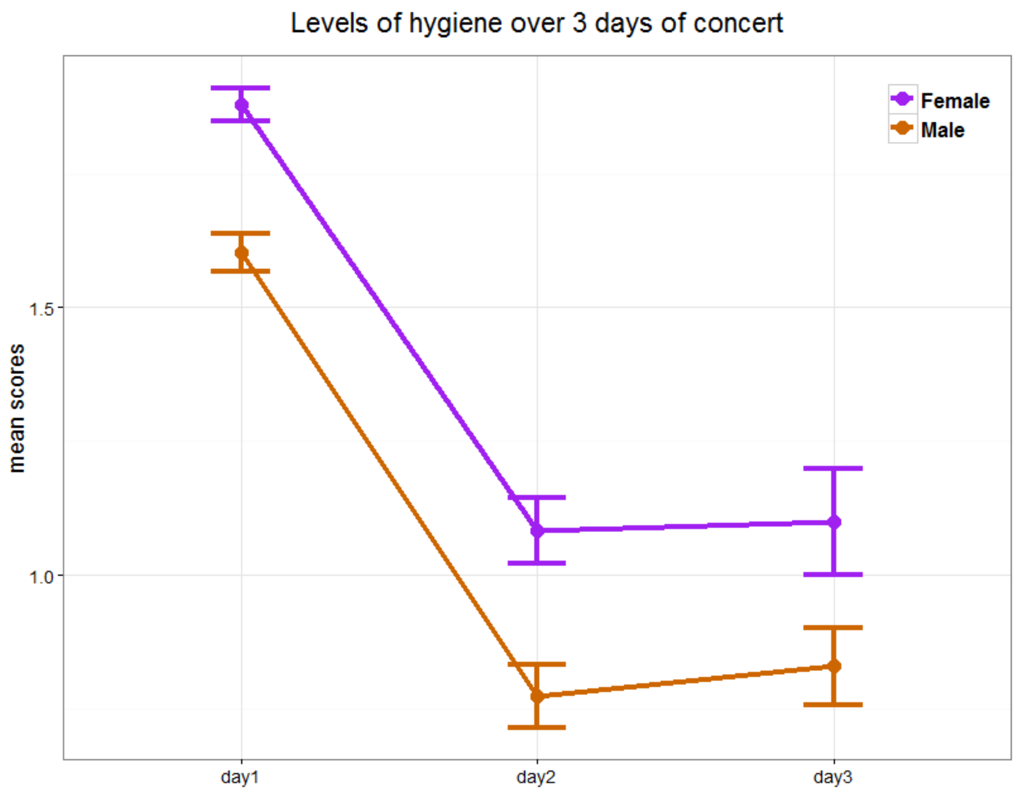
We basically want to plot the same information (mean and standard error) but in another format so we can use again the `score.sem` dataframe.

```
Linegraph<-ggplot(score.sem, aes(day,mean, group=gender))+geom_line()
+geom_point()+geom_errorbar(aes(ymin=mean-sem, ymax=mean+sem))
Linegraph
```



And if you want it prettier:

```
Linegraph<-ggplot(score.sem, aes(day,mean, colour=gender, group=gender))
+geom_line(size=1.5)+geom_point(size=5)+geom_errorbar(aes(ymin=mean-sem,
ymax=mean+sem), width=.2, size=1.5)
Linegraph<-Linegraph+ylab("mean scores")+ggtitle("Levels of hygiene over 3
days of concert")+theme(axis.title.x=element_blank())
Linegraph+scale_colour_manual(values=c("purple","darkorange3"))
+theme(legend.justification=c(1,1),legend.position=c(1, 1))
+theme(legend.text=element_text(size=16, face="bold"))
+theme(legend.title= element_blank())
+theme(plot.title=element_text(vjust=+2))
```



OK, you have done a few things to get the graph above.

To start with, you have removed the x-axis title `theme(axis.title.x=element_blank())`.

Next, manually changed the colours:
`scale_colour_manual(values=c("purple","darkorange3"))` and for this to work, you had to state in the `aes()` at the beginning that you wanted different colours for genders `colour=gender`.

Then you changed the position of the legend from outside of the graph to inside the graph. In order to do that you needed first to set the “anchoring point” of the legend (bottom-left is 0,0 and top-right is 1,1)
`legend.justification=c(1,1`)

and then stated where you wanted to put the legend:
`legend.position=c(1, 1)`

Finally, you changed the font `theme(legend.text=element_text(size=16, face="bold"))` and removed the legend title `theme(legend.title= element_blank()).`

Stacked Barcharts

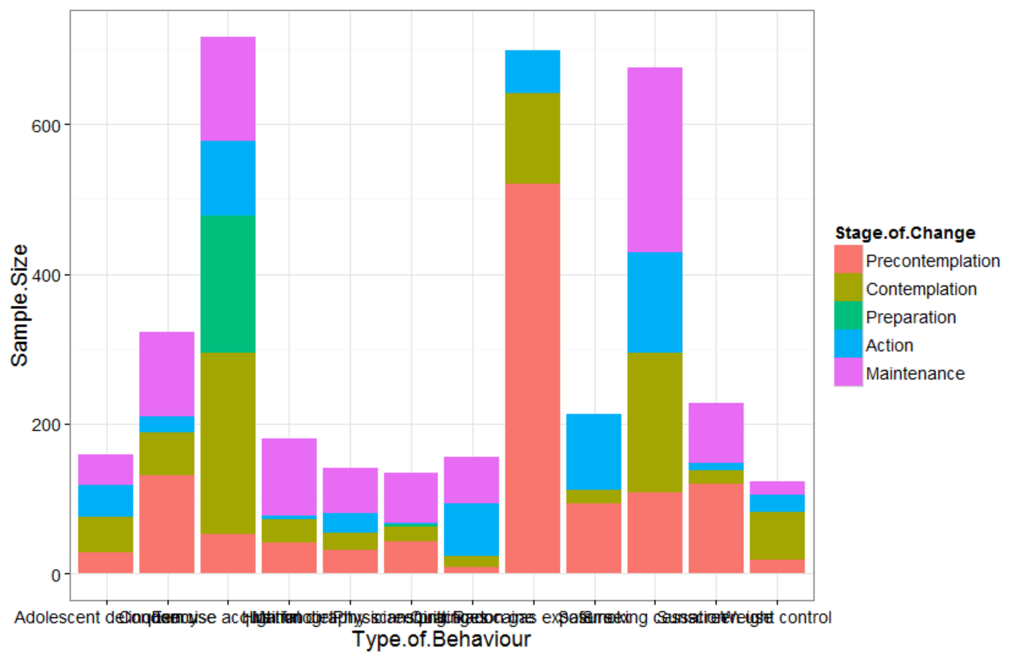
There is one last type of graph to explore: stacked bar chart. You have already plotted bar charts but this time we are going to plot categorical data which means you need to do a bit of data manipulation.

The data set (`Changing.csv`) you are going to plot contains data from 12 problem behaviours (`Type.of.Behaviour`) and the different stages of change subjects are going through (`Stage.of.Change`).

```
Changing<-read.csv("Changing.csv")
head(Changing)
```

```
   Type.of.Behavior Sample.Size  Stage.of.Change
1 Smoking cessation         108 Precontemplation
2 Smoking cessation         187    Contemplation
3 Smoking cessation           0      Preparation
4 Smoking cessation         134           Action
5 Smoking cessation         247      Maintenance
6  Quitting cocaine           8 Precontemplation
```

```
StackedBar<-ggplot(Changing, aes(Type.of.Behaviour, Sample.Size, fill =
Stage.of.Change))+geom_bar()
StackedBar
```



The first thing you are going to do is flip the chart `coord_flip()` to make the labels more readable.

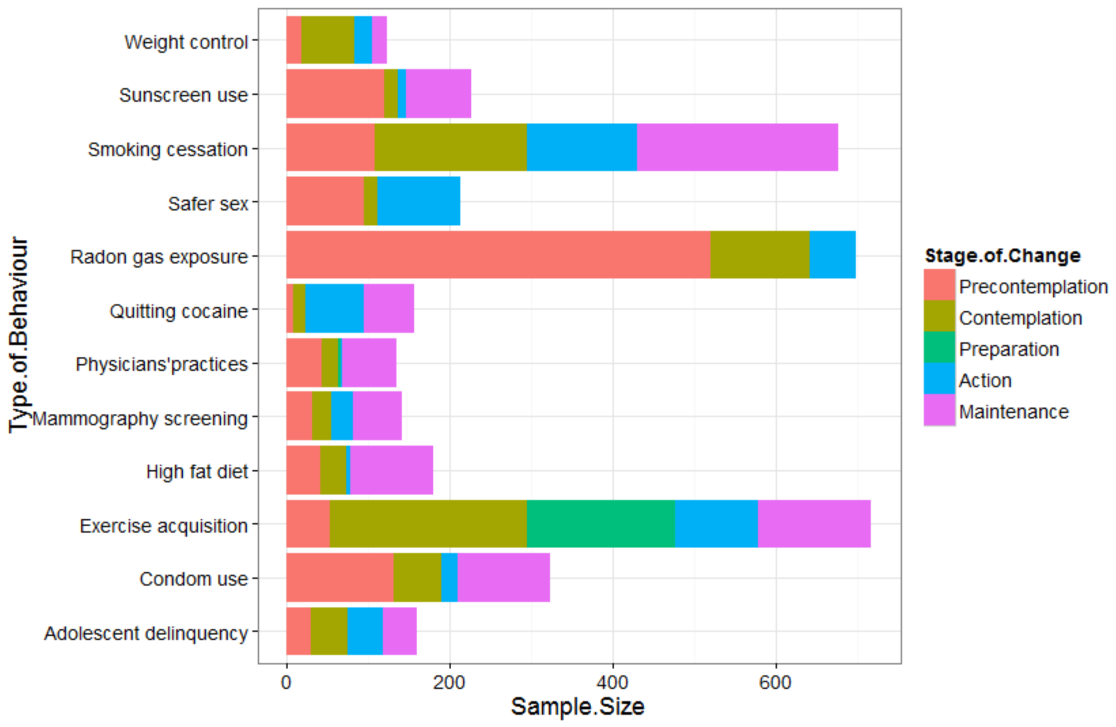
```
StackedBar<-ggplot(Changing, aes(Type.of.Behaviour, Sample.Size, fill =
Stage.of.Change))+geom_bar()+coord_flip()
StackedBar
```

The order of the stages of changes is not very informative as it is presented alphabetically and not logically. So first, make it more logical and to do that you need to ‘State.of.Change’ a factor.

```
Changing$Stage.of.Change <- factor(Changing$Stage.of.Change, levels =
c("Precontemplation", "Contemplation", "Preparation", "Action",
"Maintenance"))
```

Then you can plot again:

```
StackedBar<-ggplot(Changing, aes(Type.of.Behaviour, Sample.Size, fill =
factor(Stage.of.Change)))+geom_bar(stat="identity",colour="black")
+coord_flip()
```

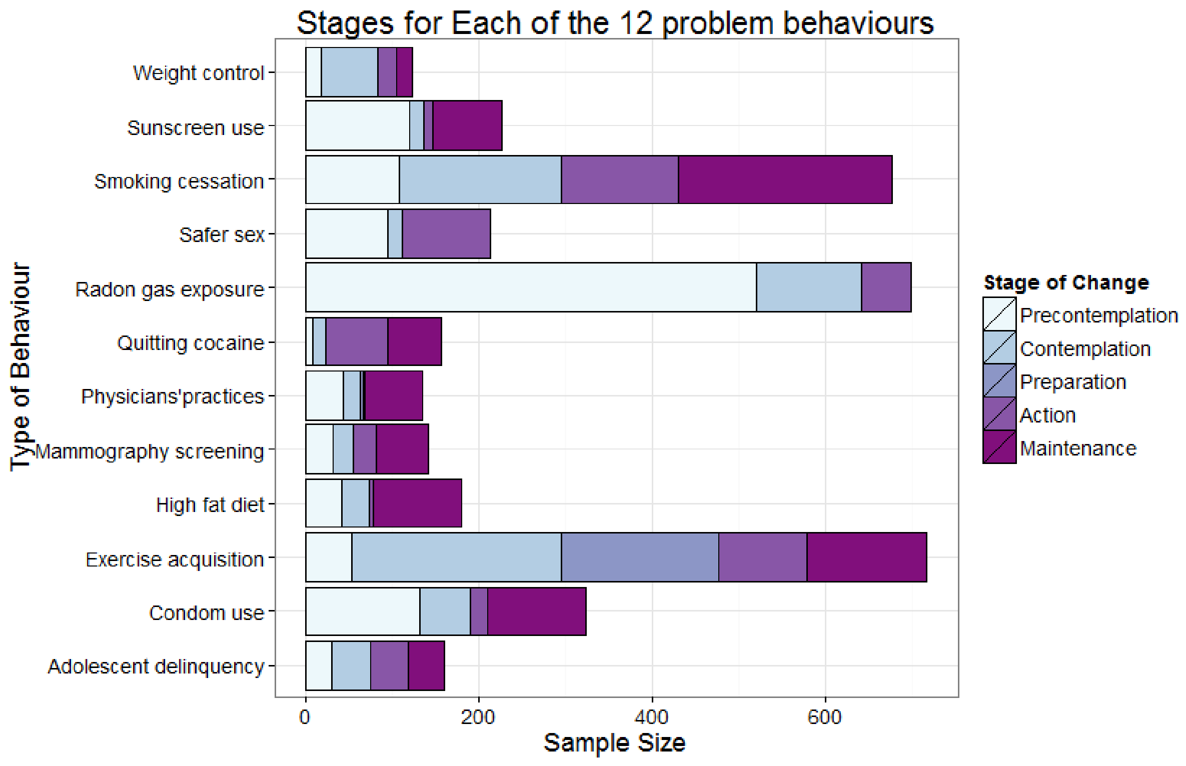


Not bad, but you could now change the colours of the factors to make it prettier.

```
StackedBar+scale_fill_brewer(palette = 3)
```

Then to finish, add some titles.

```
StackedBar+scale_fill_brewer(palette=3)+labs(title="Stages for Each of the
12 Problem Behaviours", y="Sample Size", x="Type of Behaviour", fill="Stage
of Change")
```



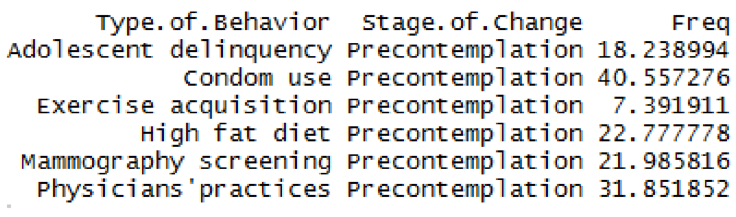
Now, you might want to plot percentages rather than raw counts. To do that, you need to create a contingency table (`xtab`) first and then transform the values into frequencies (`prop.table`).

```
contingency.table<-
xtabs(Sample.Size~Type.of.Behaviour+Stage.of.Change,Changing)
contingency.table
contingency.table100<-prop.table(contingency.table,1)
contingency.table100<-contingency.table100*100
```

That will give we the raw percentages; for the column we would type: `(contingency.table,2)`

And then you have to change it to a dataframe as `ggplot2` likes to work with dataframes.

```
Changing.percent<-as.data.frame(contingency.table100)
head(Changing.percent)
```



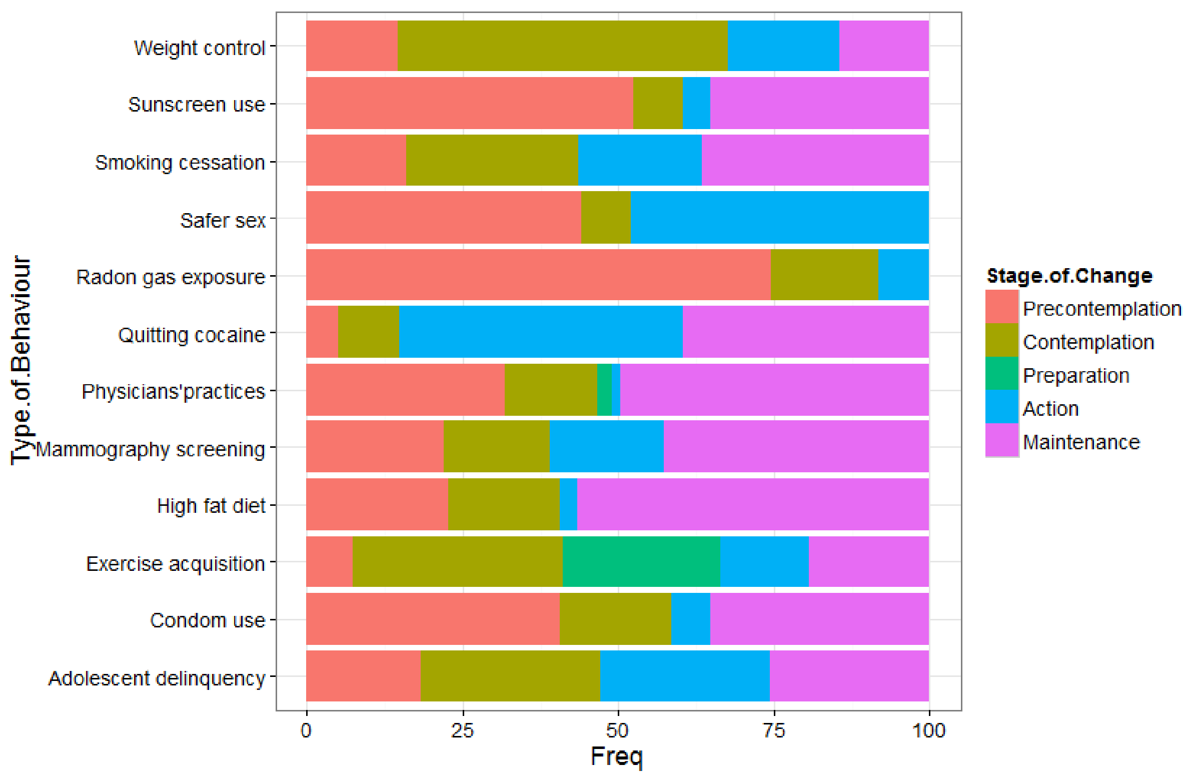
```
        Type.of.Behavior   Stage.of.Change      Freq
Adolescent delinquency Precontemplation 18.238994
           Condom use Precontemplation 40.557276
  Exercise acquisition Precontemplation  7.391911
         High fat diet Precontemplation 22.777778
 Mammography screening Precontemplation 21.985816
  Physicians'practices Precontemplation 31.851852
```

Now you can plot your new data.

```
StackedBar.percent<-ggplot(Changing.percent,aes(Type.of.Behaviour, Freq,
fill=Stage.of.Change))+geom_bar()+coord_flip()
StackedBar.percent
```

```
Changing.percent$Stage.of.Change <-
factor(Changing.percent$Stage.of.Change, levels = c("Precontemplation",
"Contemplation", "Preparation", "Action", "Maintenance"))
```

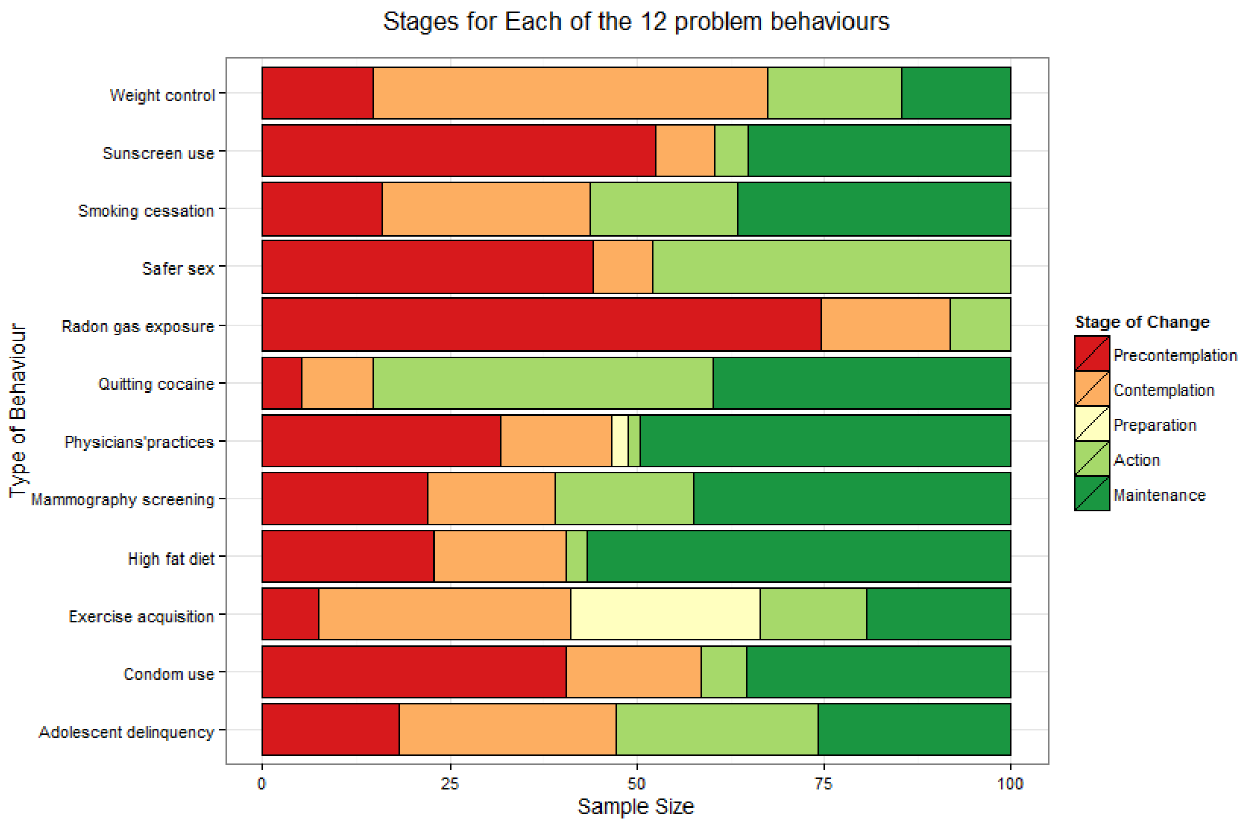
```
StackedBar<-ggplot(Changing.percent, aes(Type.of.Behaviour, Freq, fill =
factor(Stage.of.Change)))+geom_bar(stat="identity",colour="black")+coord_fl
ip()
```



This is OK, but there is room for improvement. You can find some more suitable colours and add labels for the x and y axis, as well as a main title for the plot.

To do this you could do:

```
StackedBar+scale_fill_brewer(palette = "RdYlGn")+labs(title="Stages for
Each of the 12 Problem Behaviours", y ="Frequency", x="Type of Behaviour",
fill="Stage of Change")+theme(plot.title=element_text(vjust=+2))
```



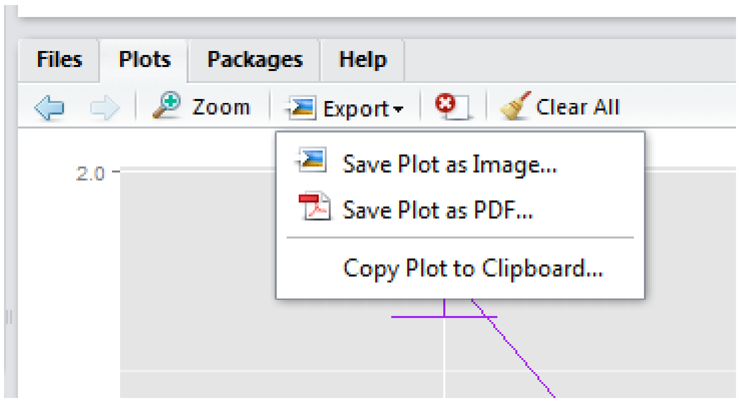
Saving Graphs

To save a graph:

```
StackedBar.saved<-ggsave(StackedBar, file=" StackedBar.png")
```

Exporting graphs from RStudio

Graphs can be exported from RStudio using the item Export from the menu bar in the graphics window. The graph can be saved as a pdf or as a range of image formats. The size of the graph can be altered.


Files
Plots
Packages
Help
Zoom
Export
Clear All
2.0
Save Plot as Image...
Save Plot as PDF...
Copy Plot to Clipboard...