# A concise guide to ggplot2

Abhishek Sathe and Ashwin Karanam

July 13, 2020

## Contents

# 1 The `ggplot2` package

ggplot2 is a data visualization package in R created by Hadley Wickham in 2005. It is an implementation of “The Grammar of Graphics” (Leland Wilkinson). It uses a layered structure to graphing which simplifies the process of coding plots in R. Intuitively, we know that any graph looks like Figure 1. ggplot adopts this concept.

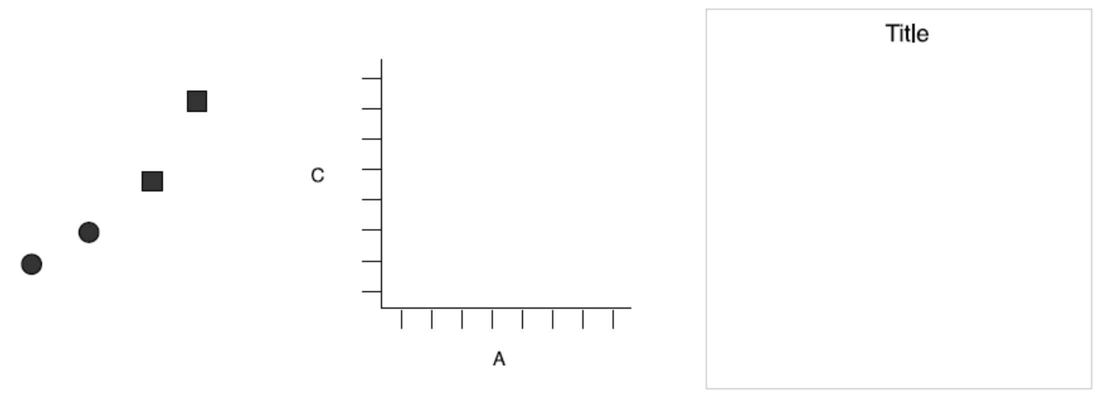


Figure 1. Graphics objects produced by (from left to right): geometric objects, scales and coordinate system, plot annotations.

Figure 1:

## 1.1 The Grammar in ggplot2

Any graph can be depicted using a set of layers. Seven layers can be used in ggplot2 but the first 3 layers listed below are required for ggplot to work:

1. Data, the dataframe that you wish to use for plot
2. Aesthetics (aes), specifies the desired visual properties
3. A geom_function, to map out the x and y values and how to represent them
4. A stat_function, that allows statistical depiction of data
5. Position, to determine where elements are located relative to each other
6. A coordinate_function, to manipulate the coordinate system used to graph
7. The two facet_functions to display multiple panels stratified by specific variables

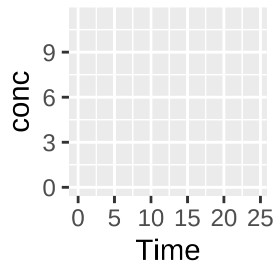
## 2 Hands on with ggplot2

As an exampe of PK data lets use the thoephylline dataset. In addition we can also use the pre-exisiting mtcars and diamonds datasets.

```r
library(ggplot2)
library(dplyr)
library(gridExtra)
library(mrgsolve)
library(ggpubr)

##Loading datasets

df <- data.frame(Theoph)
df2 <- data.frame(mtcars)
df3 <- data.frame(diamonds)
```

As we have already worked with this data before, let's skip the introductions and get right into the coding. Use the `ggplot()` function to create the first layer which is the base plot.

```r
ggplot(df,
       aes(Time,conc))
```



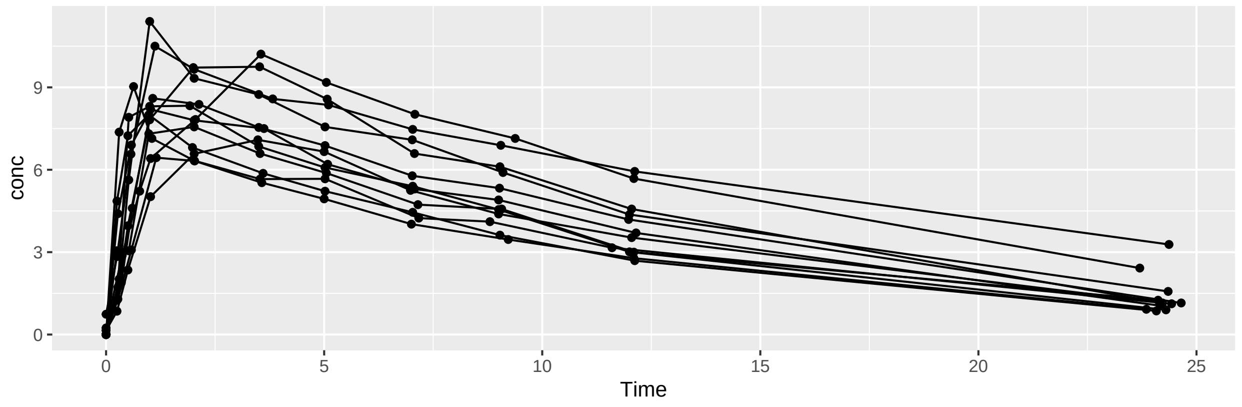
Every plot should start with `ggplot()` as this maps the x and y axis along with which data to use. To this we add layers that inform what type of plot to create using "geoms".

## 2.1 Scatterplots/Trendplots

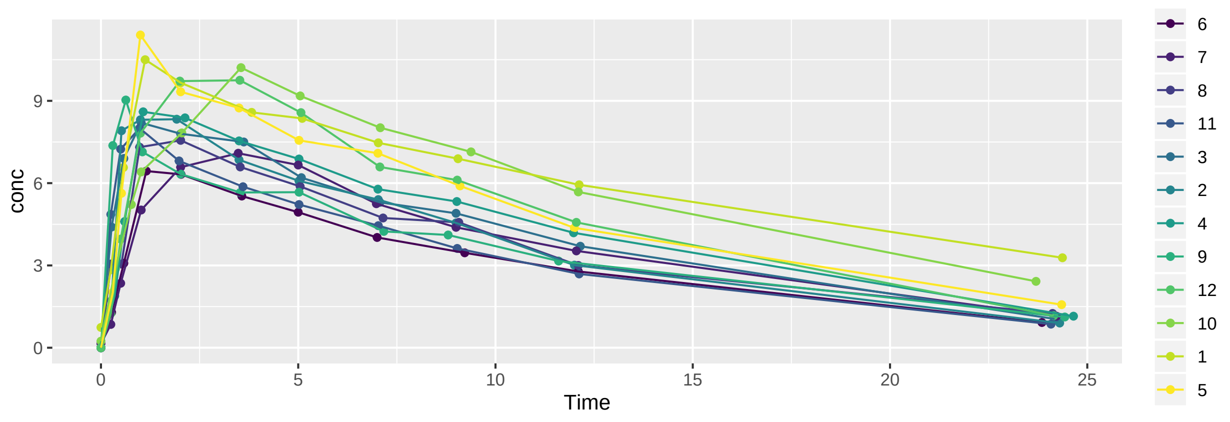
### 2.1.1 Sphagetti plots

Trendplots are one of the most important type of plots in PMx. `geom_line` is used for connecting observations in the order they appear in. The "group" argument allows you to map a group of factors. The other way of doing this would be to use the color and linetype arguments.

```r
a <- ggplot(df,aes(Time,conc,group=Subject))+
  geom_point()+
  geom_line()

a
```



```r
b <- ggplot(df,aes(Time,conc,color=Subject))+
  geom_point()+
  geom_line()
b
```

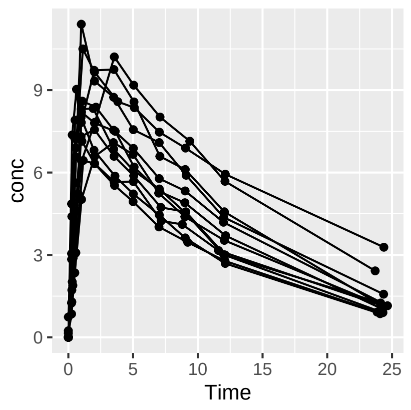


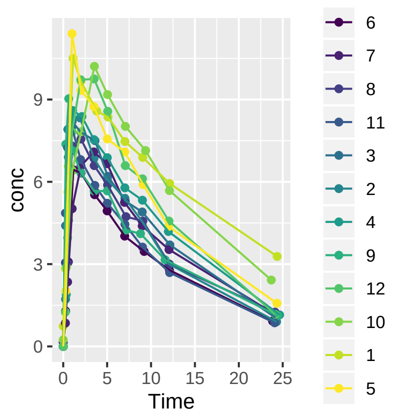
```r
c <- ggplot(df,aes(Time,conc,group=Subject))+
  geom_point(aes(shape=Subject))+
  geom_line()

grid.arrange(a,b,c,ncol=3)
```

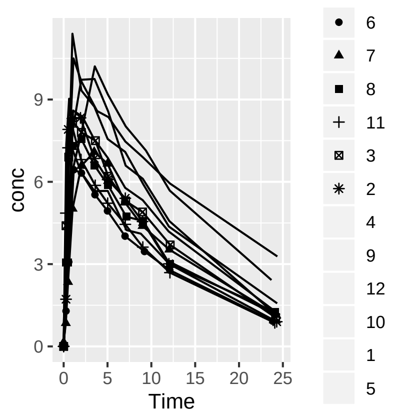
```
## Warning: Using shapes for an ordinal variable is not advised
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values
## because more than 6 becomes difficult to discriminate; you have
## 12. Consider specifying shapes manually if you must have them.
```

```
## Warning: Removed 66 rows containing missing values (geom_point).
```



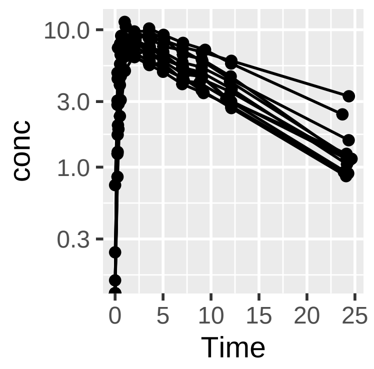




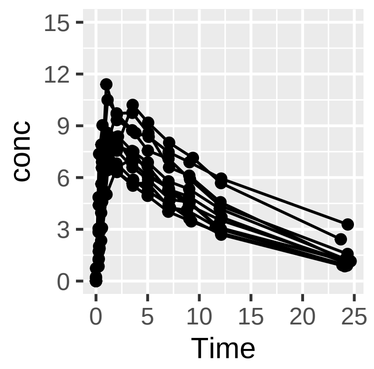
As you see they do the same thing as group, but assigns a different color/linetype to each group. Of course this is only useful if you have limited number of subgroups (as shapes/ linetype options are limited)

Another commonly used transformation is for the y-axis. For log scale Y-axis, we use the `scale_y_log10()` function which is a variant of `scale_y_continuous()`. Similar functions are available for x-axis transformations.

```
ggplot(df,aes(Time,conc,group=Subject))+
  geom_point()+
  geom_line()+
  scale_y_log10()
```

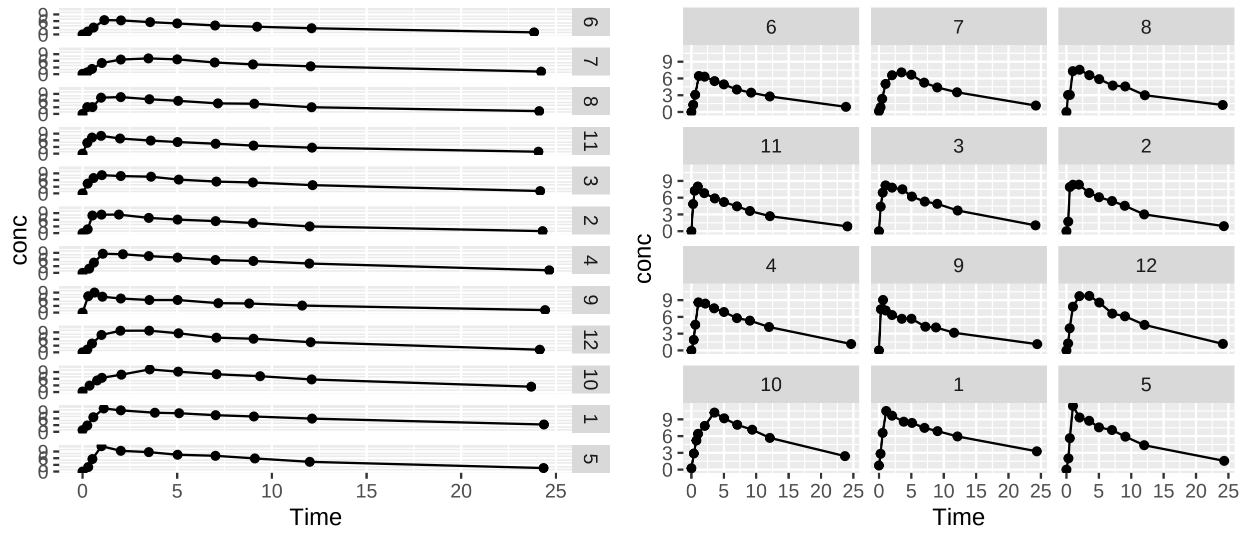


```
## Adjusting axis limits and breaks

ggplot(df, aes(Time, conc, group=Subject))+
  geom_point()+
  geom_line()+
scale_y_continuous(limits=c(0,15), breaks=seq(0,15,3))
```



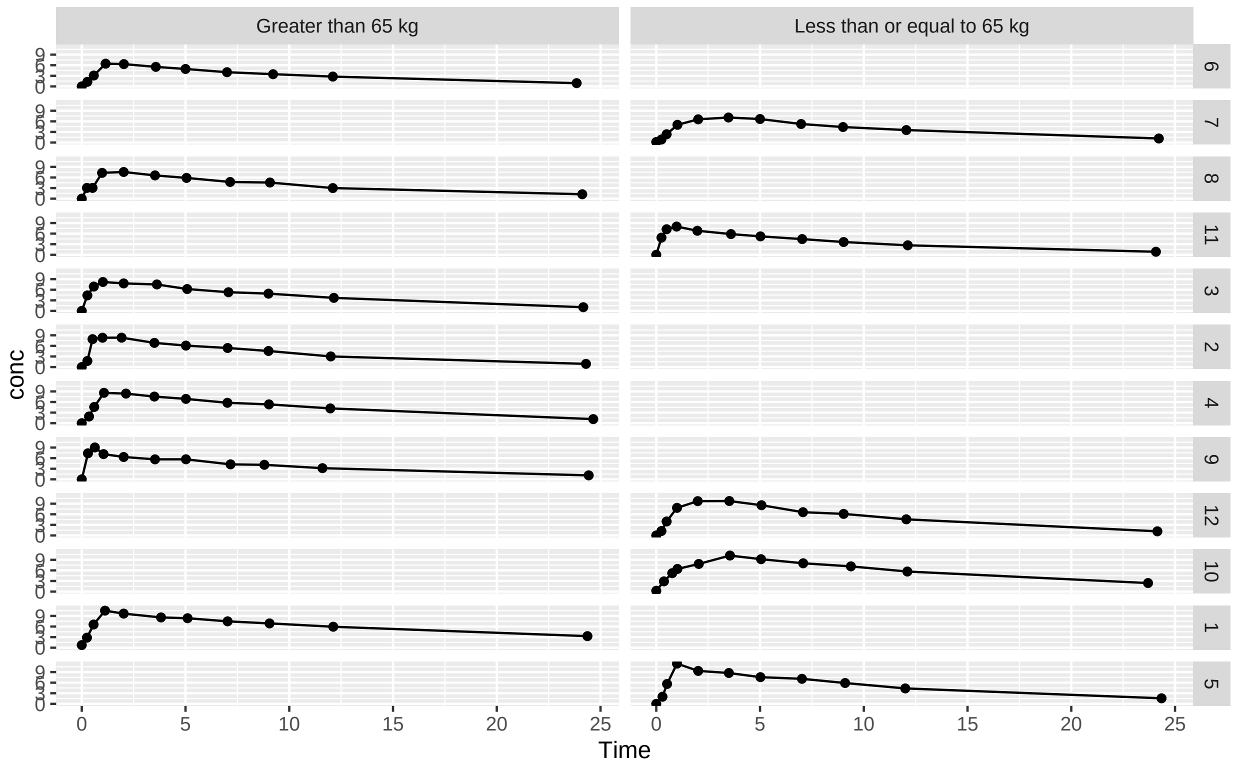
Now, let's create one plot for each individual. `facet_grid()` is one way to facet your data. This is a good option for small number of IDs, but becomes less useful with larger dataset. The other verb useful here is `facet_wrap()` which allows you to customize your grid.

```
c <- ggplot(df,aes(Time,conc))+
  geom_point()+
  geom_line()+
  facet_grid(Subject~.)

d <- ggplot(df,aes(Time,conc))+
  geom_point()+
  geom_line()+
  facet_wrap(~Subject,ncol=3)

grid.arrange(c,d,ncol=2)
```



We can add a second layer of faceting to this.

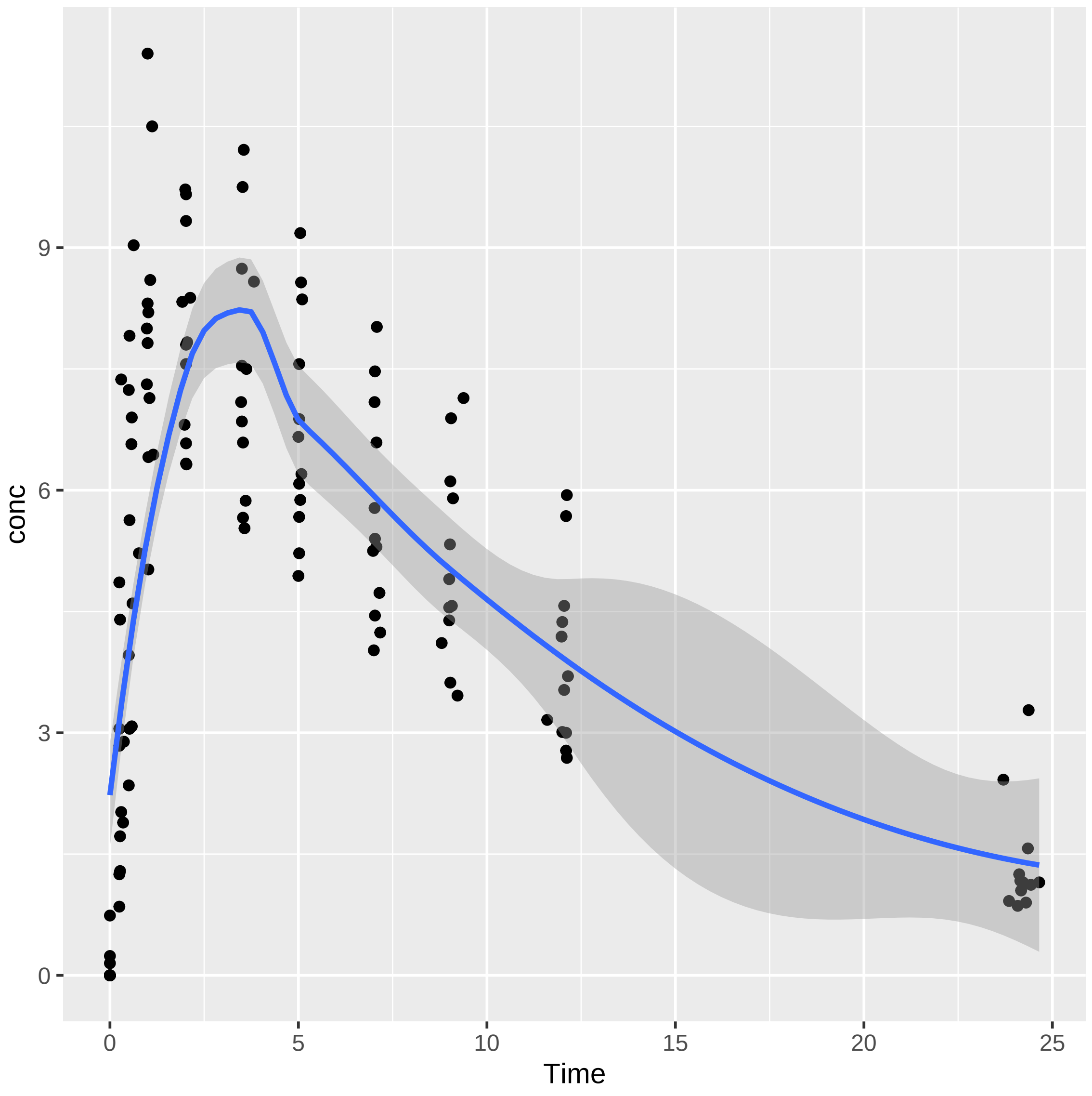
```
df4 <- df %>%
  mutate(Category = if_else(Wt <= 65,
                            "Less than or equal to 65 kg",
                            "Greater than 65 kg"))

ggplot(df4,aes(Time,conc))+
  geom_point()+
  geom_line()+
  facet_grid(Subject~Category)
```


Greater than 65 kg
Less than or equal to 65 kg
conc
Time
0
3
5
9
0
5
10
15
20
25
6
7
8
11
3
2
4
9
12
10
1
5

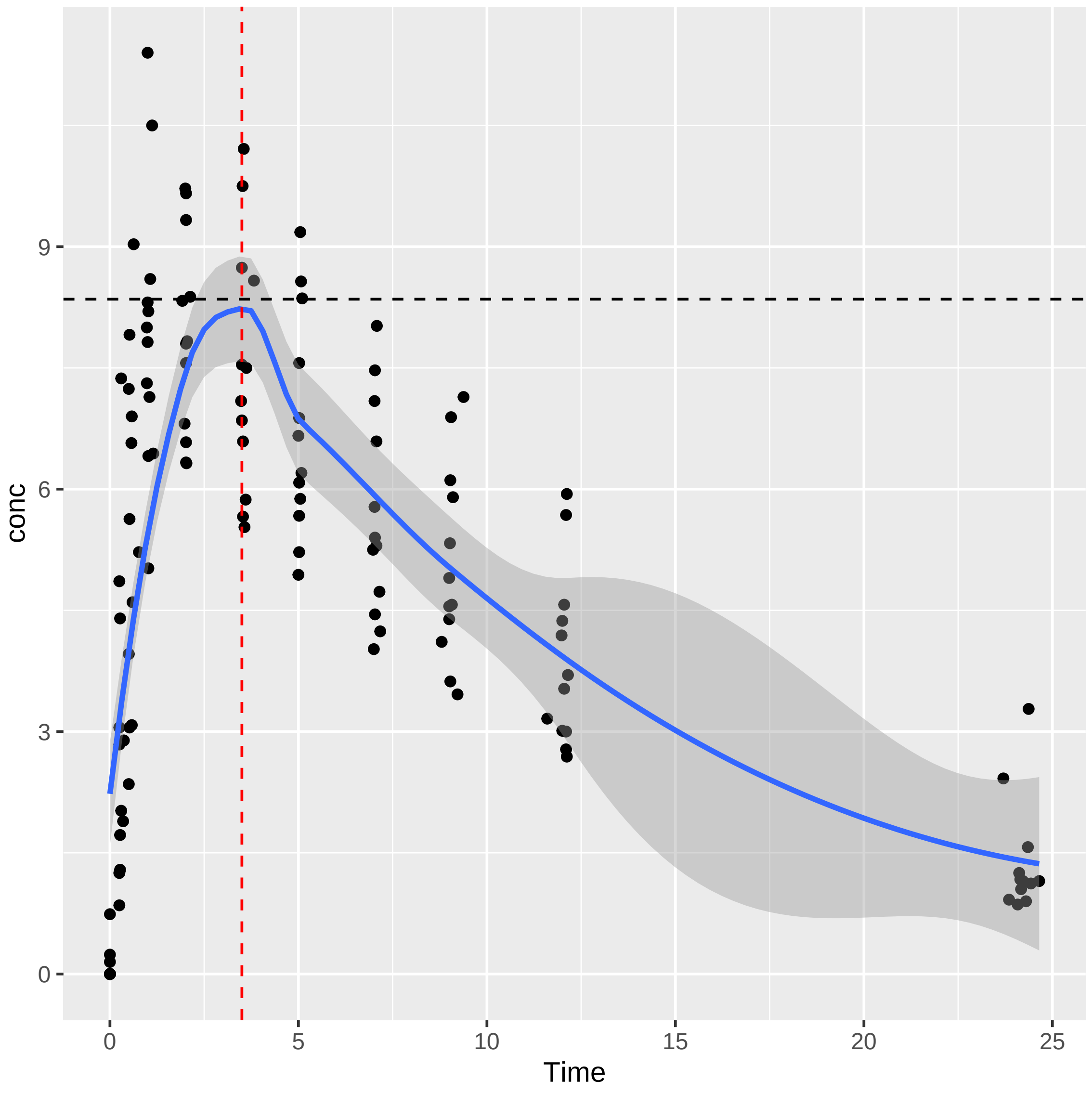
### 2.1.2 Trendplots

With the same concentration time plot, you can add a trend line eg. a non-parametric smoother. The `geom_smooth` verb allows for this in ggplot.

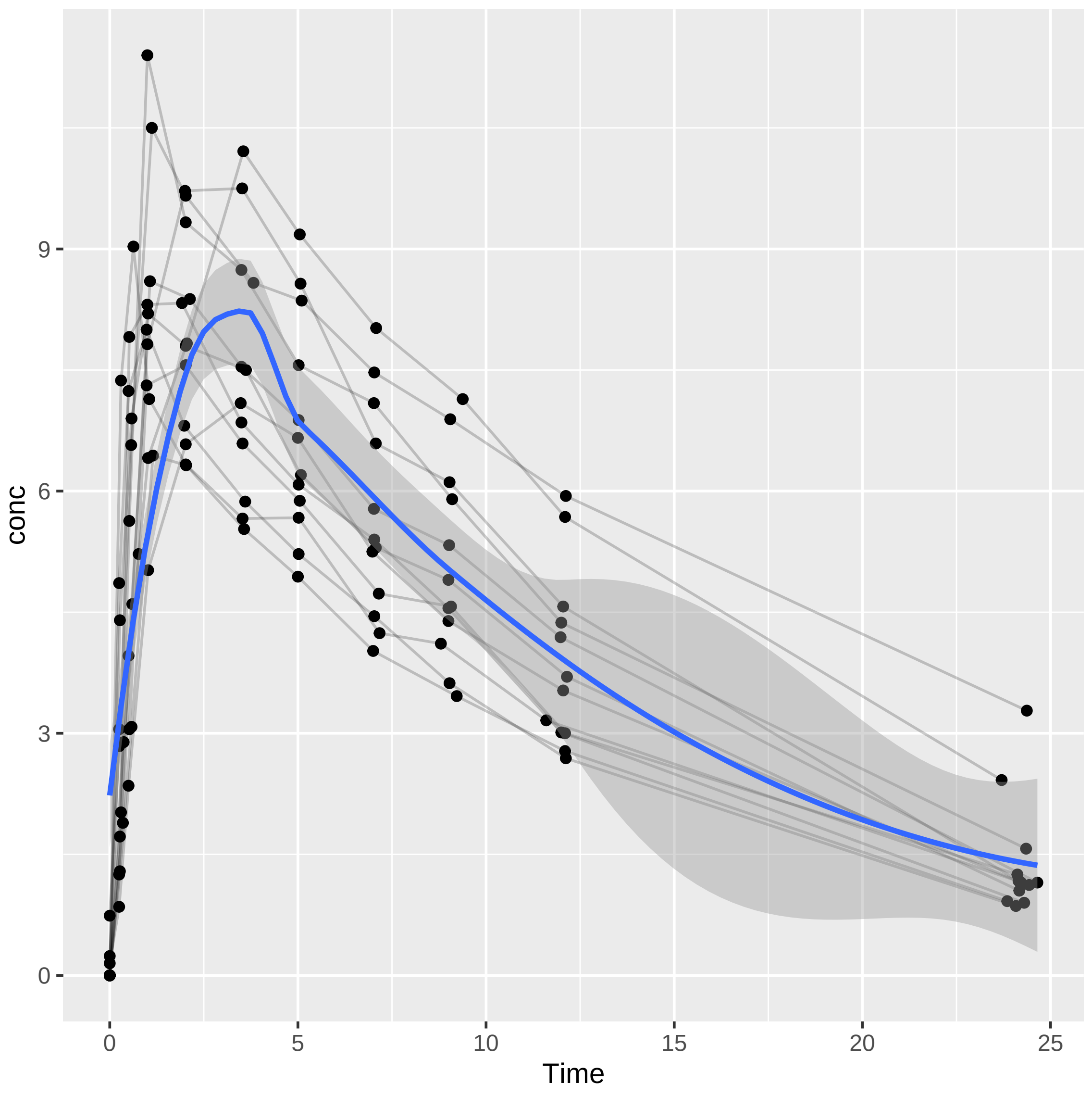
```
e <- ggplot(df,aes(Time,conc))+
  geom_point()+
  geom_smooth(se=TRUE,method="loess")

e
```



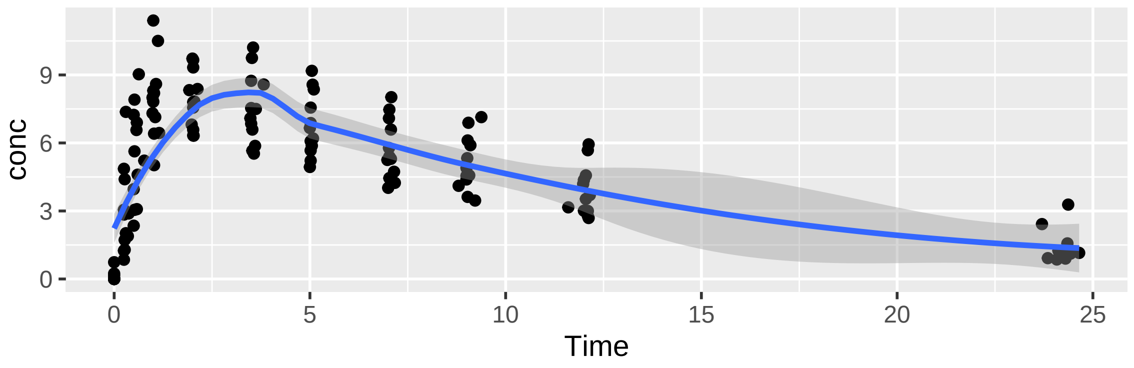
```
## Adding lines to the plot

f <- ggplot(df,aes(Time,conc))+
  geom_point()+
  geom_smooth(se=TRUE,method="loess")+
  geom_vline(xintercept = 3.5, linetype=2,color="red")+
  geom_hline(yintercept = 8.35, linetype=2,color="black")

f
```

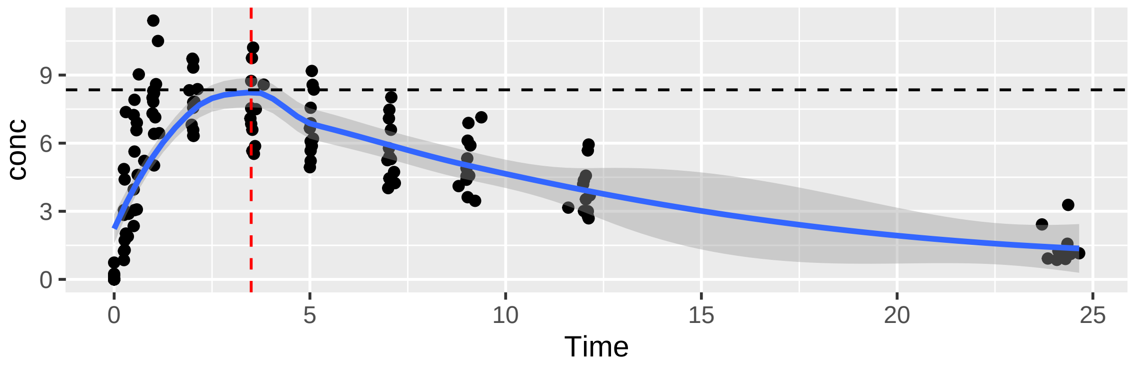


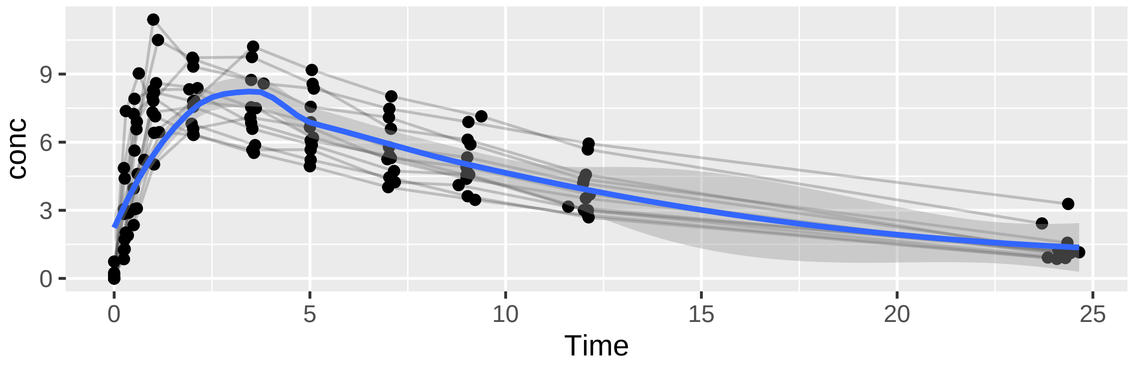
```
## Adjusting transparency
g <- ggplot(df,aes(Time,conc))+
  geom_point()+
  geom_line(aes(group=Subject),alpha=0.2)+
  geom_smooth(se=TRUE,method="loess")
g
```



```r
grid.arrange(e,f,g,ncol=1)
```







Couple of things to notice:

For plot e, the grouping aesthetic is shifted to geom_point as geom_smooth does not need the grouping aesthetic. By default geom_smooth plots the SE and uses the loess method. Additional methods include lm, glm and gam.

For plot f, using `geom_hline()` and `geom_vline()` creates reference lines. Can create any line using geom_abline() and providing values of intercept and slope.

For plot g, we are adding individual sphagetti plots to the average plot. I am using the `alpha` argument in the geom_line to make the individual lines lighter. You can use this for any geom available.

Another type of plot is an average trend plot with standard deviation. Use `geom_ribbon()` for creating a shaded region in your plot. We can also use `geom_errorbar()` for this which would give you error bars around the mean.

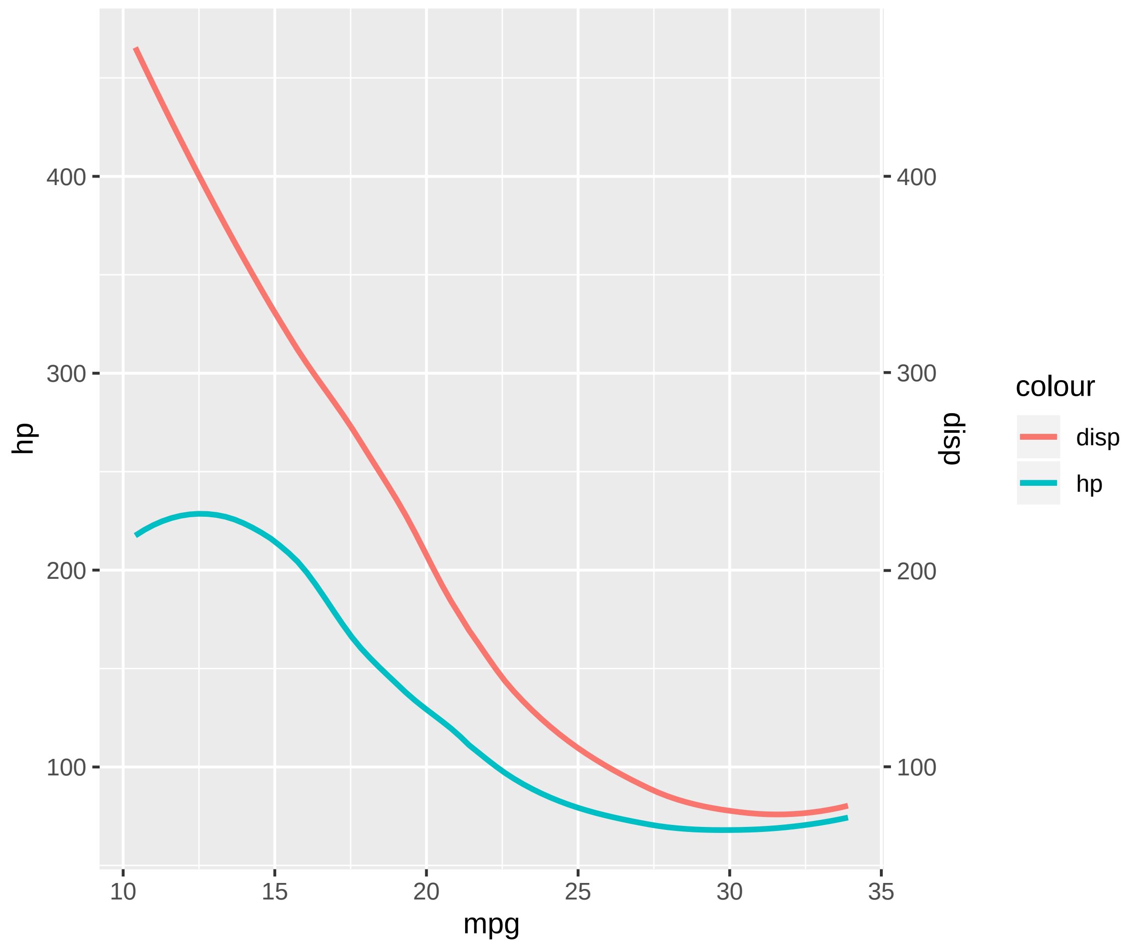
### 2.2 Adding a second axis and multiple trend lines

You can use `sec_axis()` to create a secondary axis, be it for x or y axes.

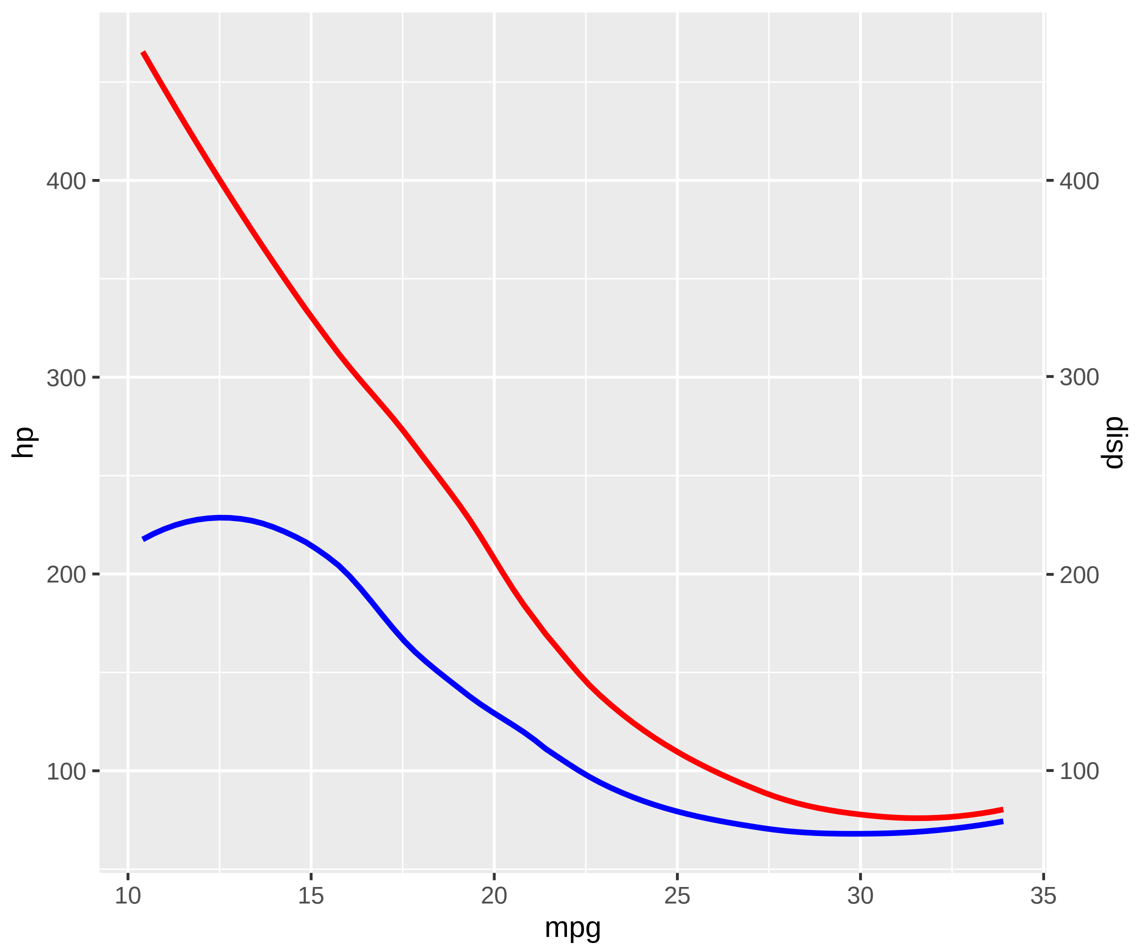
Use `geom_smooth` for adding multiple trends. Just add a seperate `geom_smooth` with new mappings.

Here is an example using the "mtcars" dataset

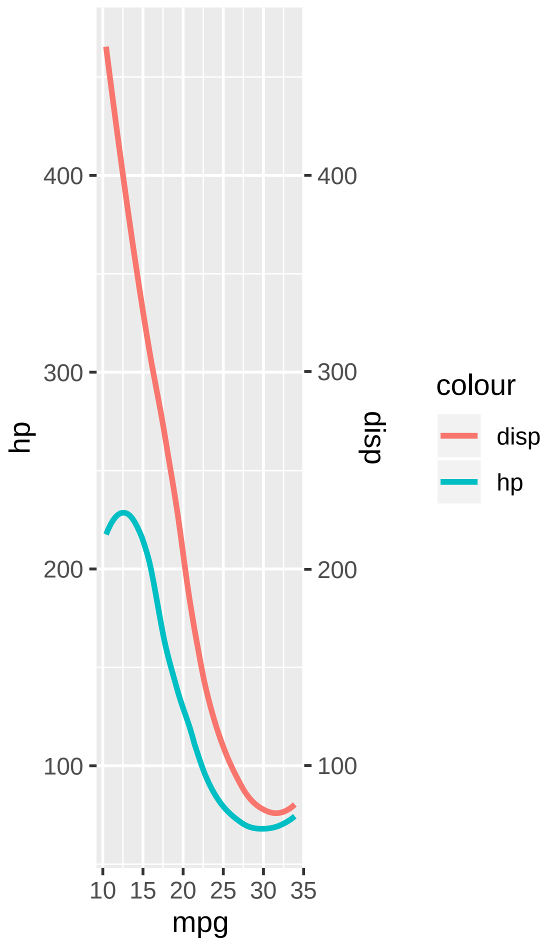
```r
i <- ggplot(df2, aes(mpg,hp))+
  geom_smooth(aes(color="hp"), se=FALSE)+
  scale_y_continuous("hp", sec.axis=sec_axis(trans= ~.*1,name="disp"))+
  geom_smooth(aes(mpg,disp,color="disp"), se=FALSE)

i
```

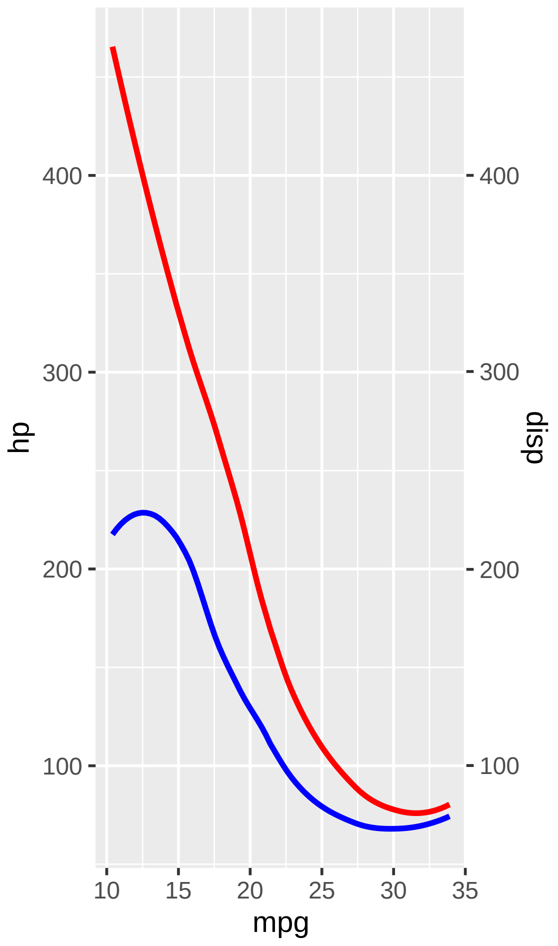


```r
j<- ggplot(df2, aes(mpg,hp))+
  geom_smooth(color="blue", se=FALSE)+
  scale_y_continuous("hp", sec.axis=sec_axis(trans= ~.*1,name="disp"))+
  geom_smooth(aes(mpg,disp),color="red", se=FALSE)

j
```



```r
grid.arrange(i,j,nrow=1)
```





If you notice, in this plot the color argument is within the aesthetics argument. This forces ggplot to consider all mappings in that geom as one entity. Here we are forcing ggplot to call the first layer as hp and second as disp.

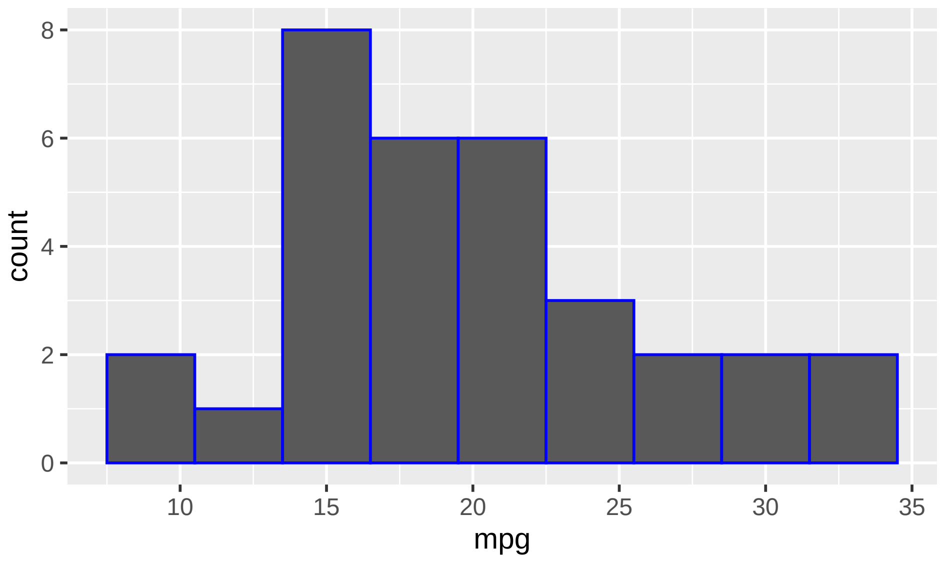
In the second plot, we do not use the aesthetics mapping for color but use the manual color selection. This does not give us a legend because you manually specified which layer has which color.

---

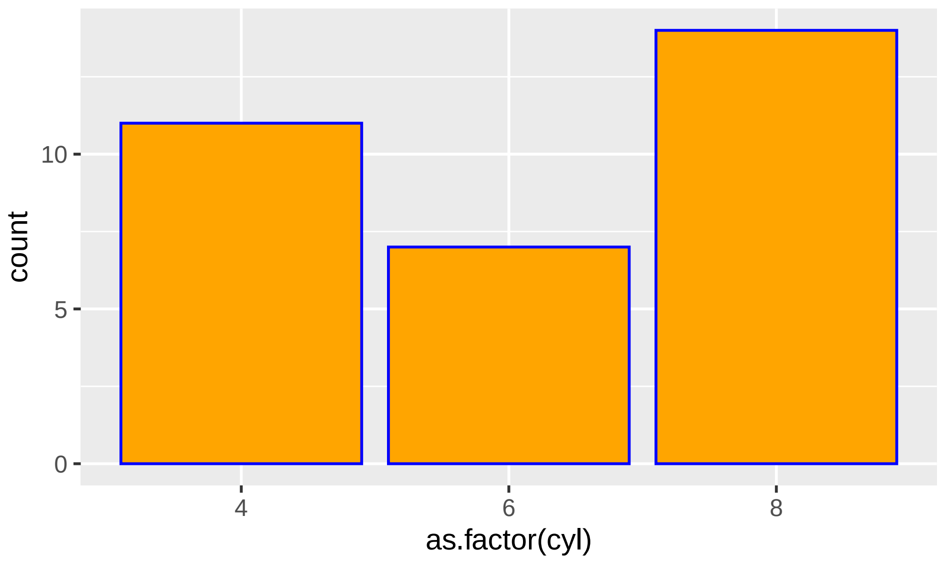
### 2.3 Histograms

Let's try a basic histogram plot of distribution of cars by displacement `geom_histogram()` for this.

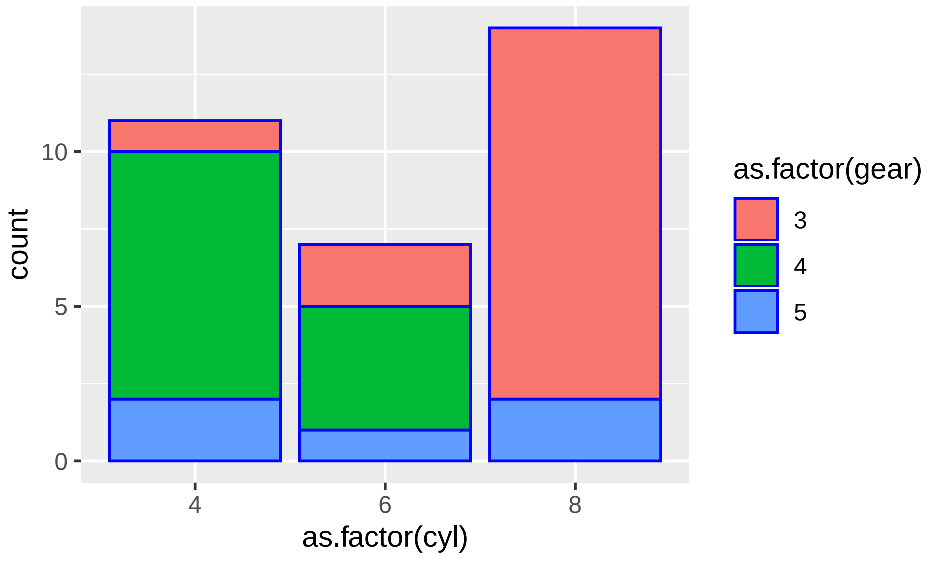
```
ggplot(df2,aes(x=mpg))+
  geom_histogram(binwidth=3, color="blue")
```



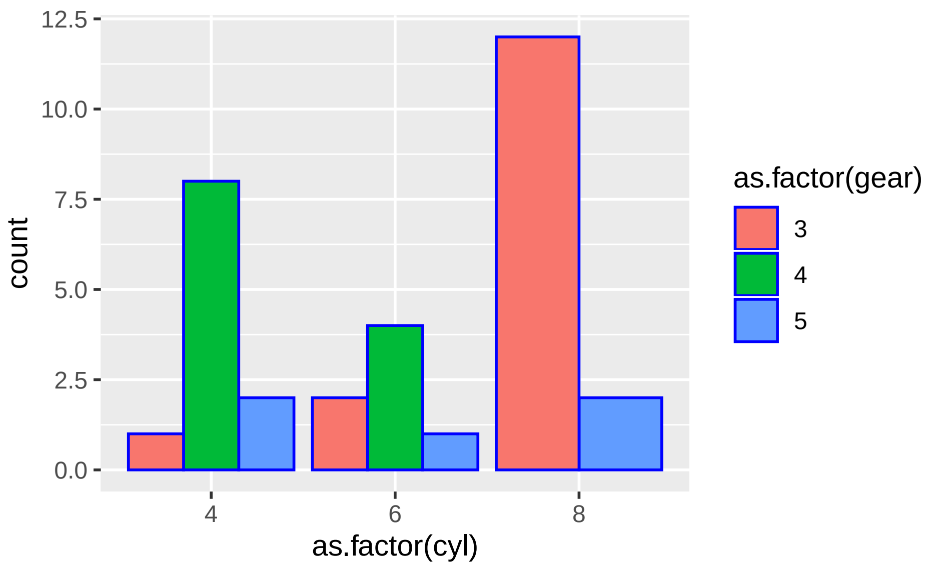
```
## Adding borders and fill
ggplot(df2,aes(x=as.factor(cyl)))+
  geom_histogram(color="blue", fill="orange", stat="count")
```



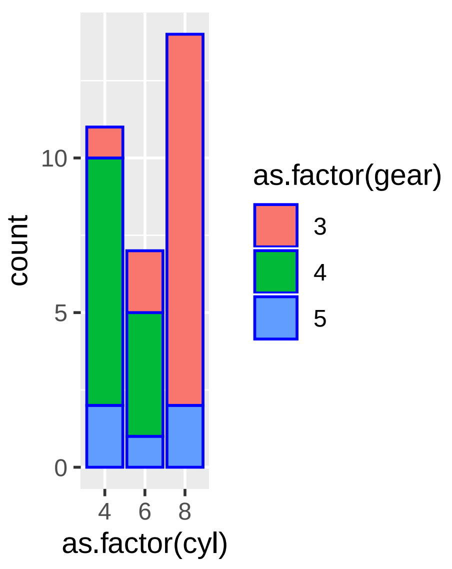
```
## Stratifying based on another variable

hist1 <- ggplot(df2,aes(as.factor(cyl)))+
  geom_histogram(aes(fill=as.factor(gear)), color="blue", position="stack", stat="count")

hist1
```



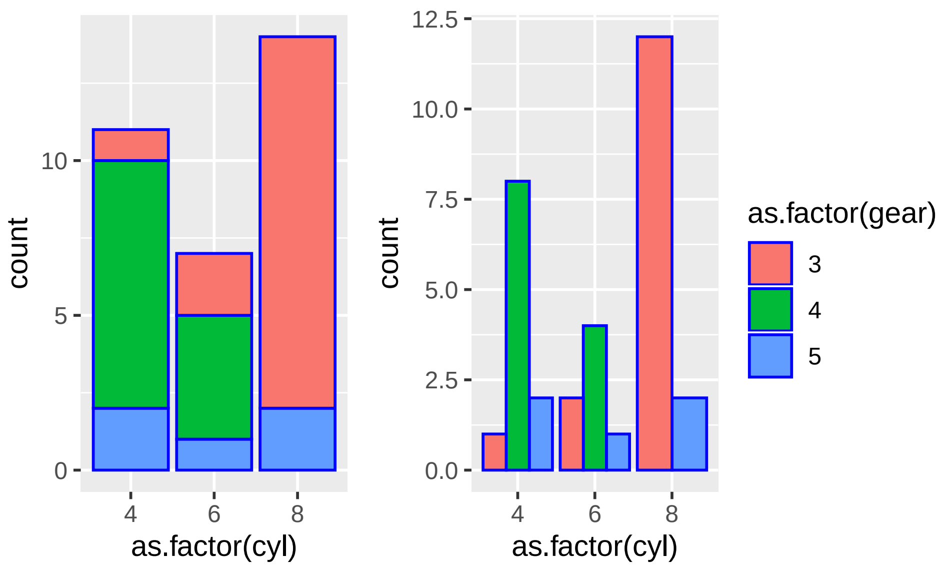
```
hist2 <- ggplot(df2,aes(as.factor(cyl)))+
  geom_histogram(aes(fill=as.factor(gear)), color="blue", position="dodge", stat="count")

hist2
```



```
ggarrange (hist1,hist2,nrow=1, legend="right")
```



```r
ggarrange (hist1,hist2,nrow=1,common.legend=T, legend="right")
```

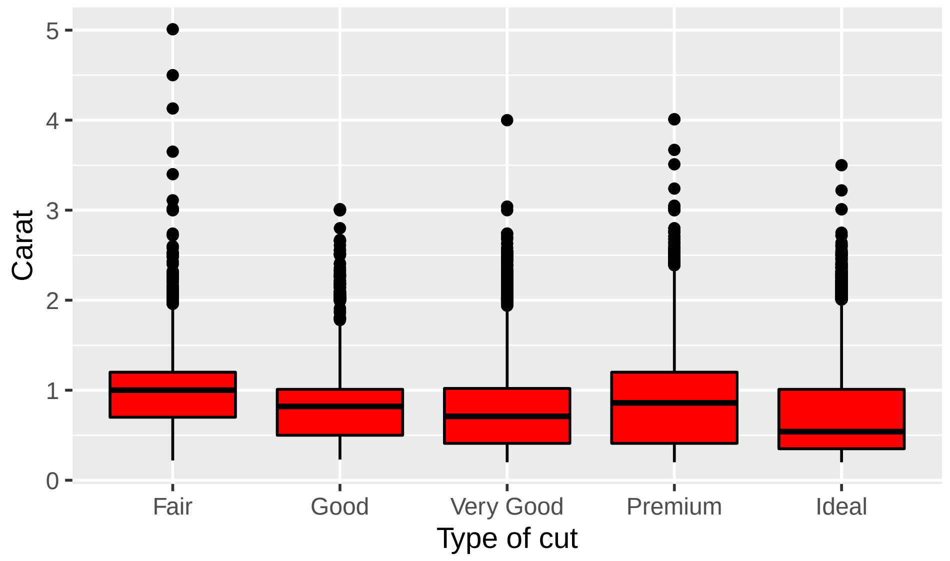


There are several options within `geom_histogram()` that let you modify the histograms.

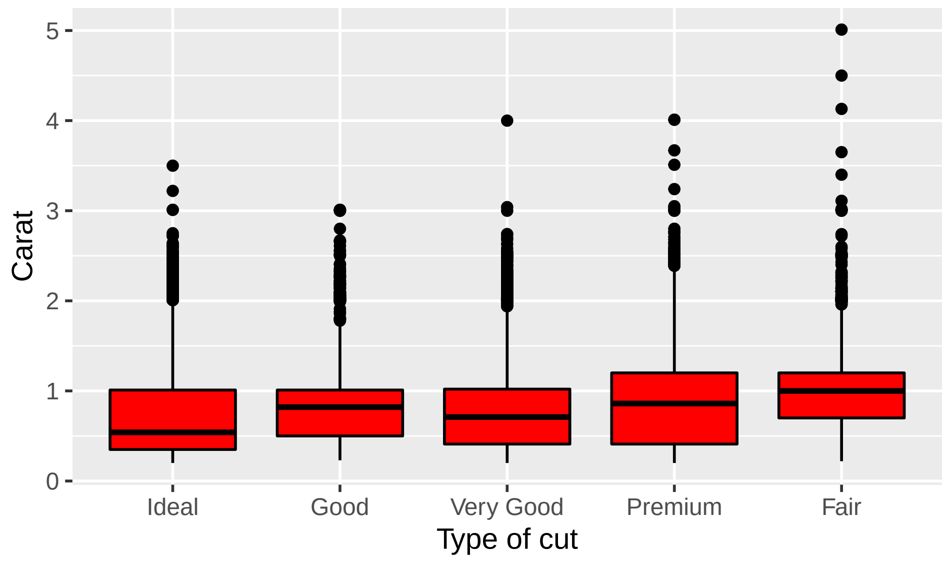
## 2.4 Boxplots

Here is an example of boxplot using the diamonds dataset

```r
ggplot(df3, aes(x=as.factor(cut), y=carat))+
  geom_boxplot(color="black", fill="red")+
  labs(x="Type of cut", y="Carat")
```



```
### Specifying order

ggplot(df3, aes(x=factor(cut, levels=c("Ideal","Good","Very Good","Premium","Fair"), ordered=TRUE),
                y=carat))+
  geom_boxplot(color="black", fill="red")+
    labs(x="Type of cut", y="Carat")
```

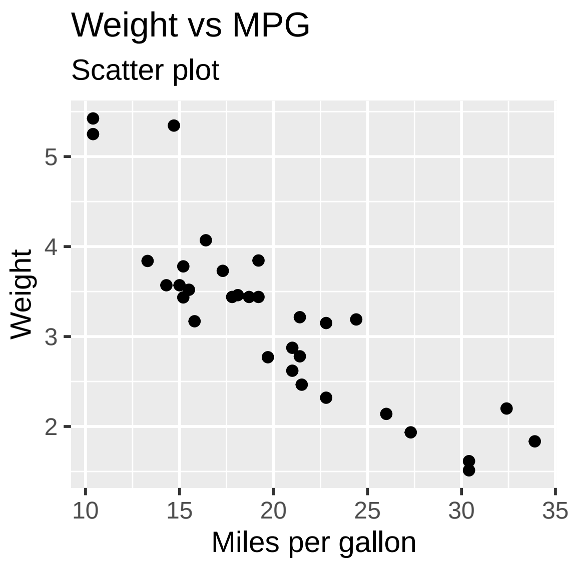


## 2.5 Formatting in ggplot

### 2.5.1 Formatting the axis labels

Use the `labs()` to edit the plot title, and axis labels. You can also use `xlab()`, `ylab()` and `ggtitle()` to individually edit these. For this exercise let us again use the "mtcars" dataset

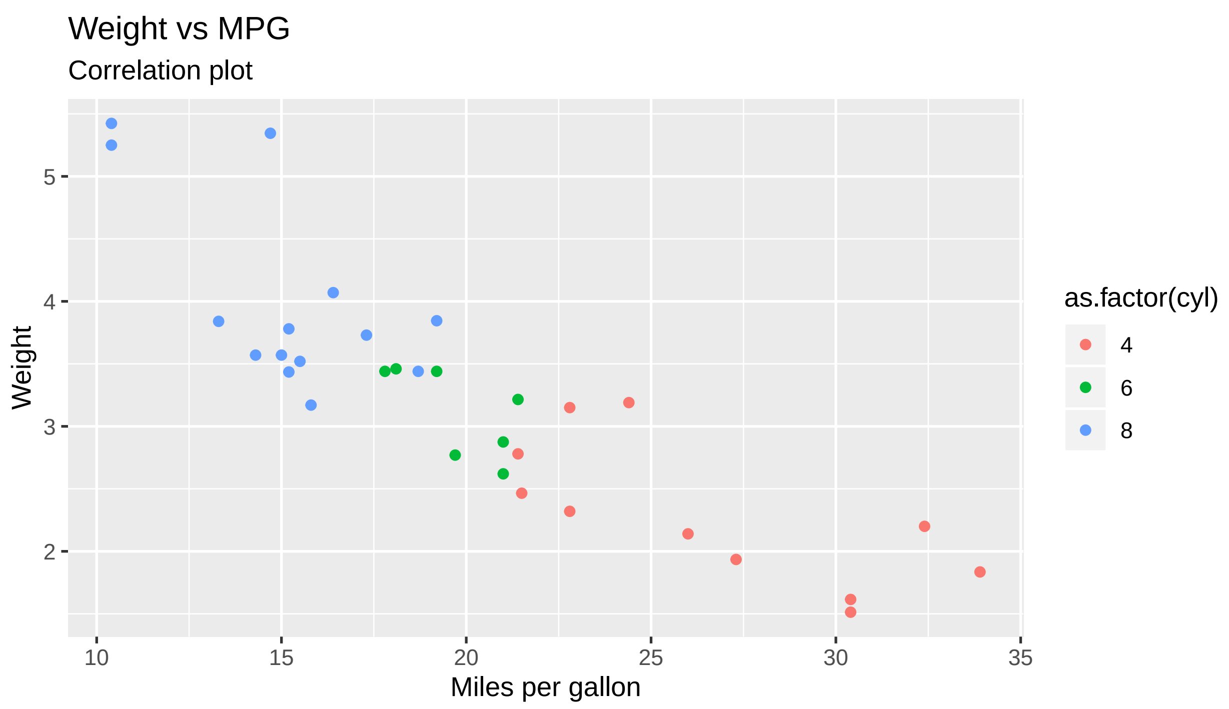
```r
ggplot(df2, aes(mpg, wt))+
  geom_point()+
  labs(title="Weight vs MPG",
       subtitle="Scatter plot",
       x="Miles per gallon",
       y="Weight")
```



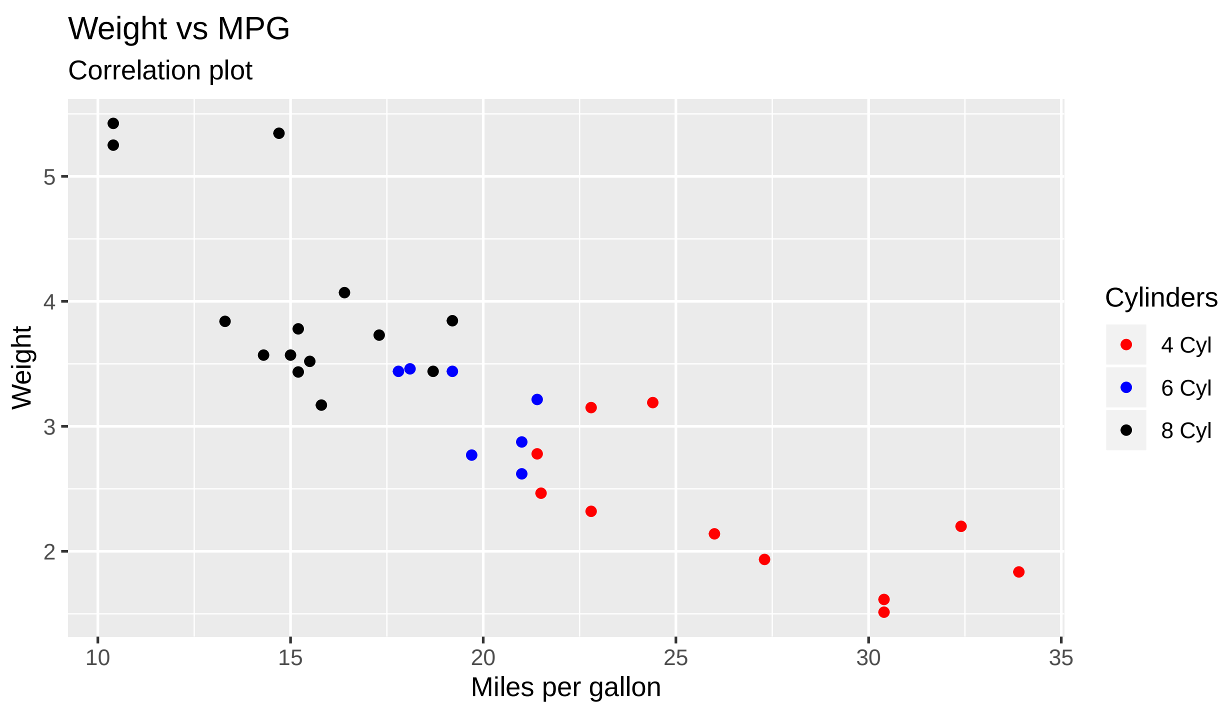
### 2.5.2 Formatting legends

ggplot uses the names of your columns or factors in said colums to determine the names of the legends. An easy way to override those is to use the `scale_color_manual()` verb. This allows you to choose customize not only the names and titles, but also the color. Similar functions are available for shape `scale_shape_manual()`, fill `scale_fill_manual()`, size `scale_size_manual()`, linetype `scale_linetype_manual()` and alpha `scale_alpha_manual()`. Similar off-shoots are available for discrete variables `scale_*_discrete()` and continious variables `scale_*_continuous()`.

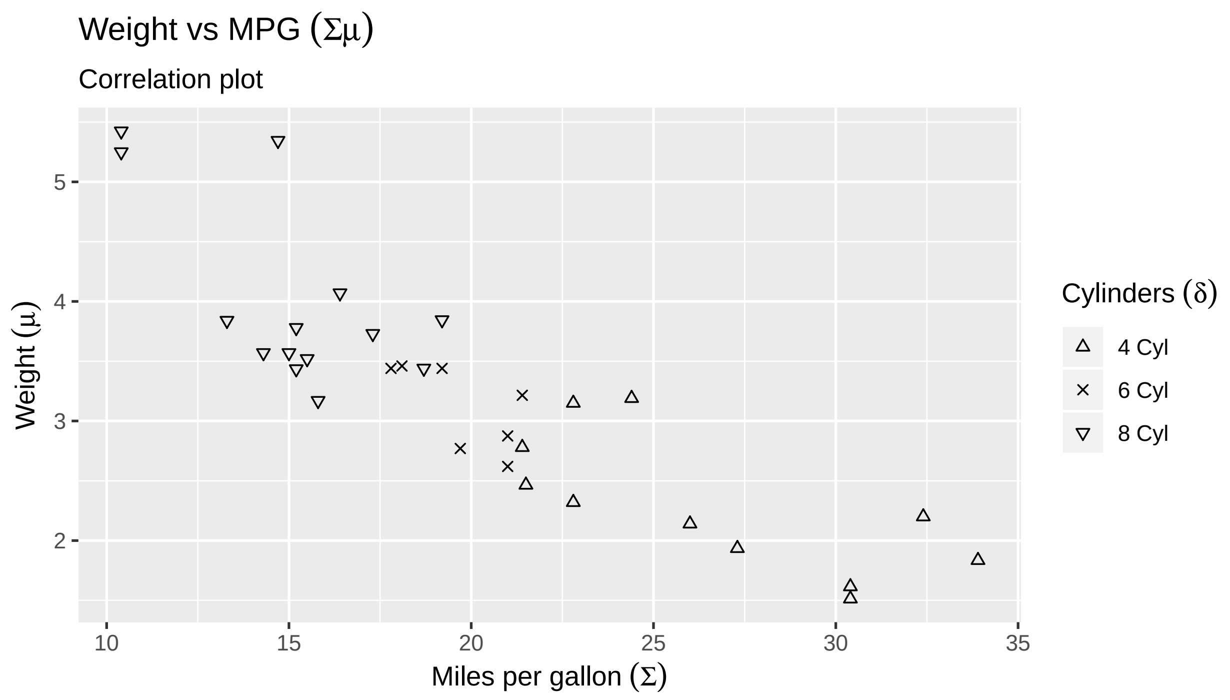
```r
## Title and subtitle

k <- ggplot(df2, aes(mpg, wt, color=as.factor(cyl)))+
  geom_point()+
  labs(title="Weight vs MPG",
       subtitle="Correlation plot",
       x="Miles per gallon",
       y="Weight")

k
```



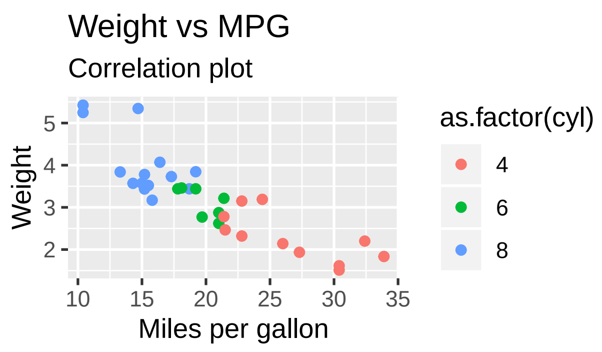
```
## Customizing colors and legend

l <- ggplot(df2, aes(mpg, wt, color=as.factor(cyl)))+
  geom_point()+
  labs(title="Weight vs MPG",
       subtitle="Correlation plot",
       x="Miles per gallon",
       y="Weight")+
  scale_colour_manual(values=c("red", "blue", "black"),
                      name="Cylinders",
                      breaks=c("4", "6", "8"),
                      labels=c("4 Cyl", "6 Cyl", "8 Cyl"))
l
```

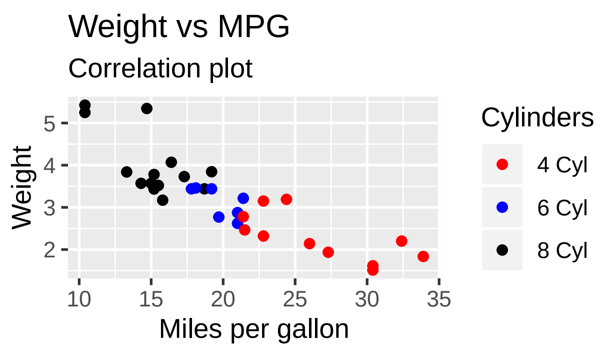


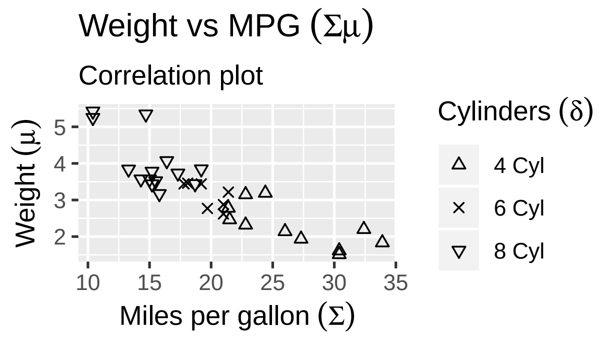
```
## Customizing shape and legend


m <- ggplot(df2, aes(mpg, wt, shape=as.factor(cyl)))+
  geom_point()+
  labs(title="Weight vs MPG"~(Sigma*mu),
       subtitle="Correlation plot",
       x="Miles per gallon"~(Sigma),
       y="Weight"~(mu))+
  scale_shape_manual(values=c(2,4,6),
                       name="Cylinders"~(delta),
                       breaks=c("4", "6", "8"),
                       labels=c("4 Cyl","6 Cyl","8 Cyl"))
m
```



```r
grid.arrange(k,l,m,ncol=2)
```



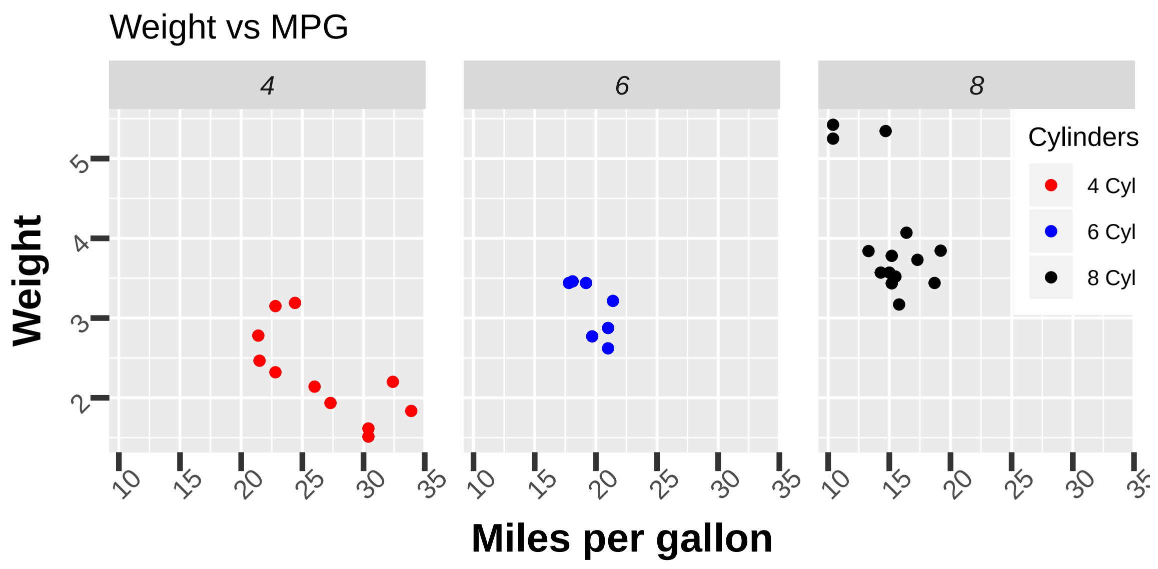




### 2.5.3 Text size formatting

`theme()` is a great way of formatting all your text in one place. There are around 50 arguments within `theme()` all aimed at providing user control over the formatting.

```r
## Formatting minute details of the plot

ggplot(df2, aes(mpg, wt, color=as.factor(cyl)))+
  geom_point()+
  facet_grid(.~as.factor(cyl))+
  labs(title="Weight vs MPG",
       x="Miles per gallon",
       y="Weight")+
  scale_colour_manual(values=c("red", "blue", "black"),
                      name="Cylinders",
                      breaks=c("4", "6", "8"),
                      labels=c("4 Cyl", "6 Cyl", "8 Cyl")) +
  theme(axis.title.x=element_text(size=15,face="bold"),
        axis.title.y=element_text(size=15,face="bold"),
        axis.text = element_text(size=10,angle = 45),
        axis.ticks = element_line(size = 1),
        axis.ticks.length = unit(.25, "cm"),
        strip.text.x = element_text(size=10,face="italic"),
        legend.position = c(0.95,0.7),
        legend.title=element_text(size=10),
        legend.text=element_text(size=8),
        panel.spacing = unit(1, "lines"))
```



# 3 Colors, themes and exporting with ggplot2

## 3.1 R Color

This is a short section on colors and how to custmomize colors in R and in ggplot2

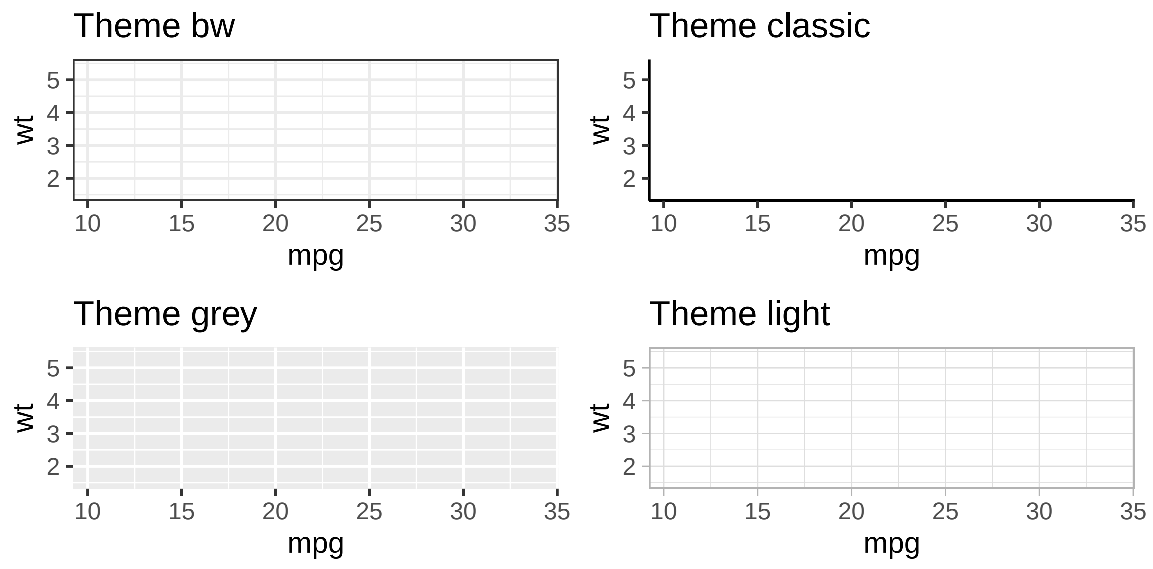
There are several help or how-to sections on the web about R colors. There are countless times when the defaults in ggplot look decent for a presentation, but when you print it out, they look really bad. Overcome them by:

1. Here is a consize guide to the color choices are given by `Columbia-Stats`. These colors go hand in hand with `scale_color_manual()`
2. If you're a fan of using the color pallete instead of the names, then I suggest use this `link`
3. If you want to be adventurous, there's several packages that give fun palletes like `wesanderson`

## 3.2 Themes in ggplot

There are several built in themes availabe for ggplot. Use the `theme_*()` verb to explore the options. Here are some for demostration.

```r
o <- ggplot(df2, aes(mpg, wt, shape=as.factor(cyl)))+
  theme_bw()+
  labs (title="Theme bw")

p <- ggplot(df2, aes(mpg, wt, shape=as.factor(cyl)))+
  theme_classic()+
  labs (title="Theme classic")

q <- ggplot(df2, aes(mpg, wt, shape=as.factor(cyl)))+
  theme_grey()+
  labs (title="Theme grey")

r <- ggplot(df2, aes(mpg, wt, shape=as.factor(cyl)))+
  theme_light()+
  labs (title="Theme light")

grid.arrange(o,p,q,r,ncol=2)
```



### 3.3 Exporting publication quality images

You can output almost any type of image file. The verbs are `tiff()`, `bmp()`, `jpeg()`, and `png()`.

```
tiff("test.tiff", width = 9, height = 8, units = 'in', res = 300)
1
dev.off()
```

## 4 Helpful References

1. `R in Action` by Robert Kabacoff
2. `Hands-On Programming` with R by Garrett Grolemund
3. `R Cookbook` by Paul Teetor
4. `THE ART OF R PROGRAMMING` by Norman Matloff
5. `The Grammar of Graphics` by Leland Wilkinson
6. `A Layered Grammar of Graphics` by Hadley Wickham

## 5 Session Information

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18362)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggpubr_0.2.4    magrittr_1.5    mrgsolve_0.10.0 gridExtra_2.3
## [5] dplyr_0.8.3     ggplot2_3.2.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.2                plyr_1.8.4
##  [3] pillar_1.4.2              compiler_3.6.1
##  [5] tools_3.6.1               digest_0.6.22
##  [7] evaluate_0.14             tibble_2.1.3
##  [9] gtable_0.3.0              viridisLite_0.3.0
## [11] pkgconfig_2.0.3           rlang_0.4.1
## [13] yaml_2.2.0                xfun_0.10
## [15] RcppArmadillo_0.9.800.1.0 withr_2.1.2
## [17] stringr_1.4.0             knitr_1.25
## [19] cowplot_1.0.0             grid_3.6.1
## [21] tidyselect_0.2.5          glue_1.3.1
## [23] R6_2.4.0                  rmarkdown_2.1
## [25] reshape2_1.4.3            purrr_0.3.3
## [27] scales_1.0.0              htmltools_0.4.0
## [29] assertthat_0.2.1          colorspace_1.4-1
## [31] ggsignif_0.6.0            labeling_0.3
## [33] stringi_1.4.3             lazyeval_0.2.2
## [35] munsell_0.5.0             crayon_1.3.4
```