ggplot2 is the most elegant and aesthetically pleasing graphics framework available in R. It has a nicely planned structure to it. This tutorial focusses on exposing this underlying structure you can use to make any ggplot. But, the way you make plots in ggplot2 is very different from base graphics making the learning curve steep. So leave what you know about base graphics behind and follow along. You are just 5 steps away from cracking the ggplot puzzle.

Topics

1. The Setup
2. The Layers
3. The Labels
4. The Theme
5. The Facets
6. Commonly Used Features

1. The Setup

First, you need to tell ggplot what dataset to use. This is done using the `ggplot(df)` function, where `df` is a dataframe that contains all features needed to make the plot. This is the most basic step. Unlike base graphics, ggplot doesn’t take vectors as arguments.

Optionally you can add whatever aesthetics you want to apply to your ggplot (inside `aes()` argument) - such as X and Y axis by specifying the respective variables from the dataset. The variable based on which the color, size, shape and stroke should change can also be specified here itself. The aesthetics specified here will be inherited by all the geom layers you will add subsequently.

If you intend to add more layers later on, may be a bar chart on top of a line graph, you can specify the respective aesthetics when you add those layers.

Below, I show few examples of how to setup ggplot using in the `diamonds` dataset that comes with `ggplot2` itself. However, no plot will be printed until you add the geom layers.

Examples:

```
library(ggplot2)
ggplot(diamonds)  # if only the dataset is known.
ggplot(diamonds, aes(x=carat))  # if only X-axis is known. The Y-axis can be
specified in respective geoms.
ggplot(diamonds, aes(x=carat, y=price))  # if both X and Y axes are fixed for all
layers.
ggplot(diamonds, aes(x=carat, color=cut))  # Each category of the 'cut' variable
will now have a distinct  color, once a geom is added.
```

The `aes` argument stands for aesthetics. ggplot2 considers the X and Y axis of the plot to be aesthetics as well, along with color, size, shape, fill etc. If you want to have the color, size etc fixed (i.e. not vary based on a variable from the dataframe), you need to specify it outside the `aes()`, like this.

```
ggplot(diamonds, aes(x=carat), color="steelblue")
```

See this color palette for more colors.

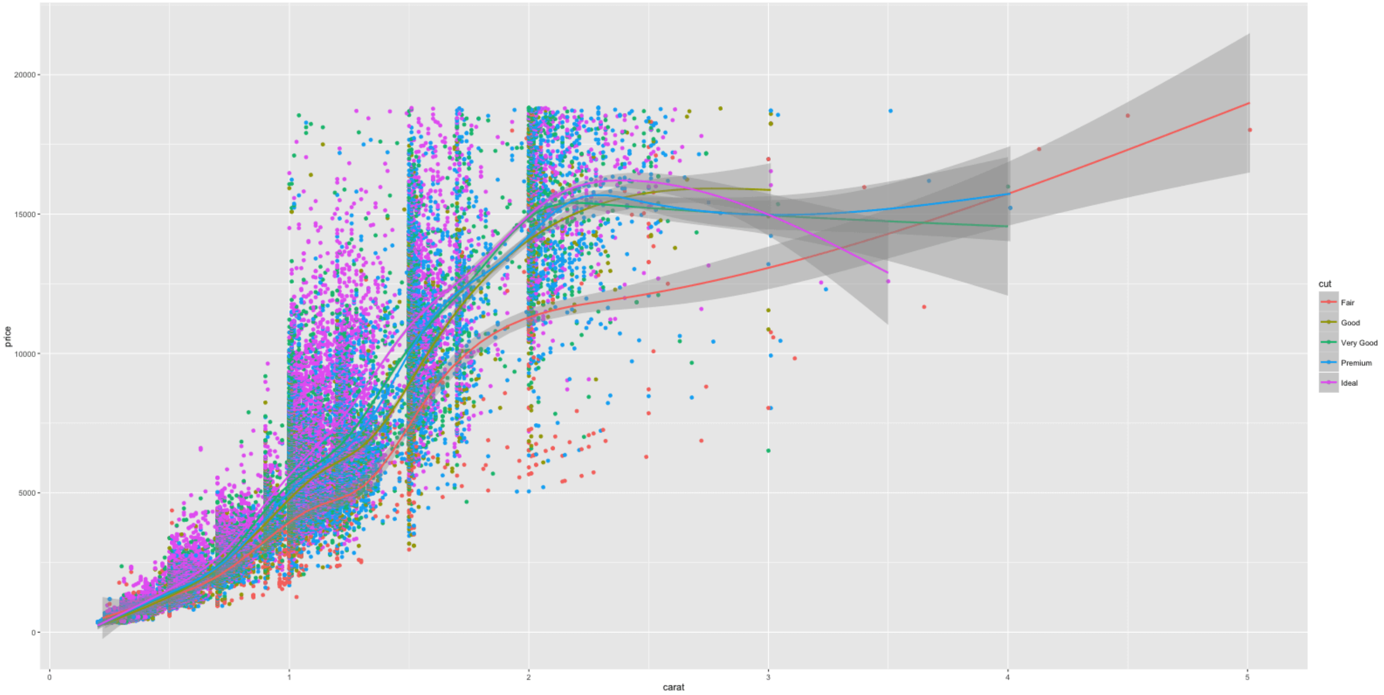
2. The Layers

The layers in ggplot2 are also called '*geoms*'. Once the base setup is done, you can append the geoms one on top of the other. The documentation provides a compehensive list of all available *geoms*.

```
ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + geom_smooth() #
Adding scatterplot geom (layer1) and smoothing geom (layer2).
```

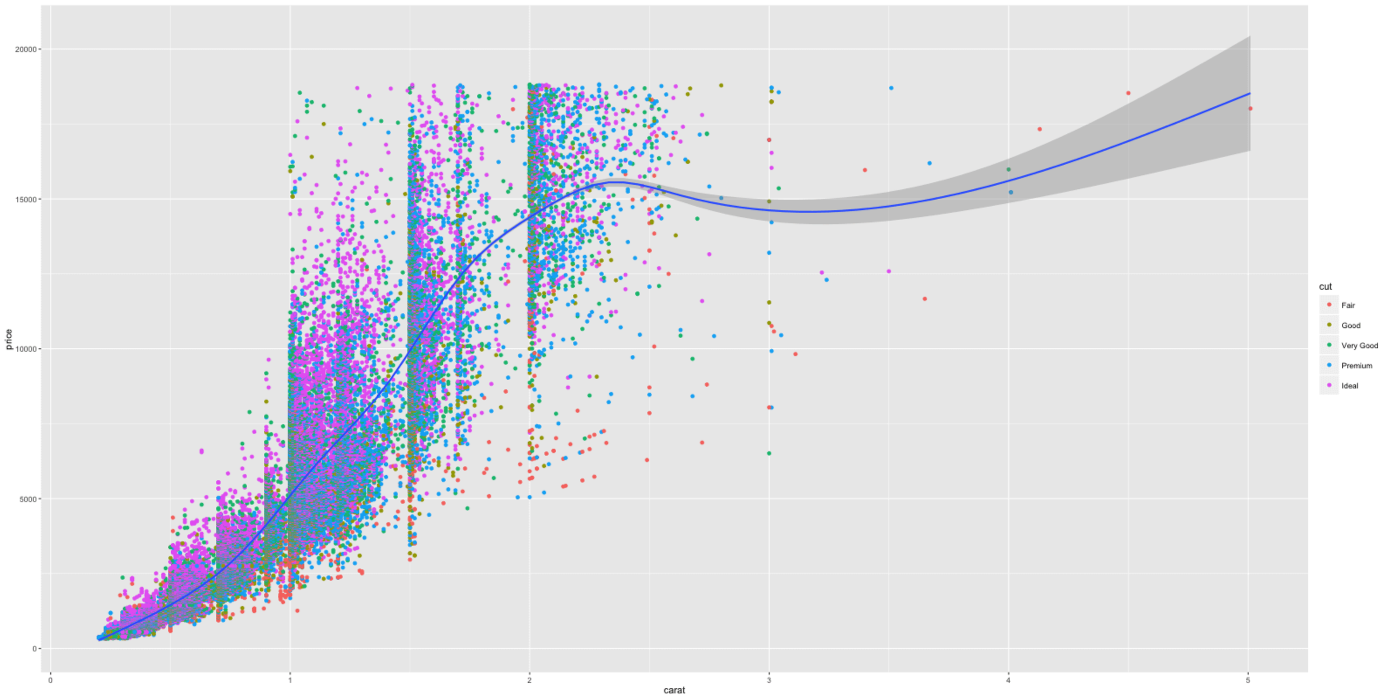
We have added two layers (geoms) to this plot - the `geom_point()` and `geom_smooth()`. Since the X axis Y axis and the color were defined in `ggplot()` setup itself, these two layers inherited those aesthetics. Alternatively, you can specify those aesthetics inside the geom layer also as shown below.

```
ggplot(diamonds) + geom_point(aes(x=carat, y=price, color=cut)) +
geom_smooth(aes(x=carat, y=price, color=cut)) # Same as above but specifying the
aesthetics inside the geoms.
```



Notice the X and Y axis and how the color of the points vary based on the value of `cut` variable. The legend was automatically added. I would like to propose a change though. Instead of having multiple smoothing lines for each level of `cut`, I want to integrate them all under one line. How to do that? Removing the `color` aesthetic from `geom_smooth()` layer would accomplish that.

```
library(ggplot2)
ggplot(diamonds) + geom_point(aes(x=carat, y=price, color=cut)) +
geom_smooth(aes(x=carat, y=price)) # Remove color from geom_smooth
ggplot(diamonds, aes(x=carat, y=price)) + geom_point(aes(color=cut)) +
geom_smooth()  # same but simpler
```



Here is a quick challenge for you. Can you make the shape of the points vary with `color` feature?

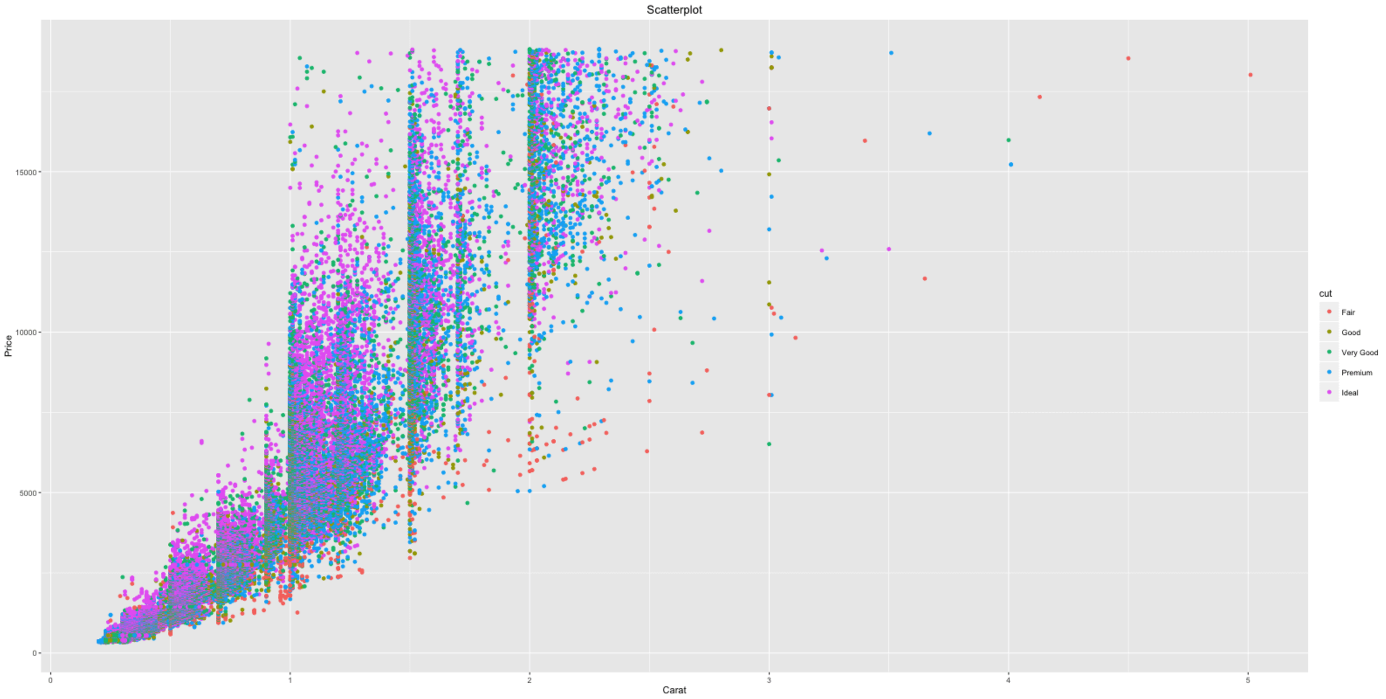
Though setting up took us quite a bit of code, adding further complexity such as the layers, distinct color for each cut etc was easy. Imagine how much code you would have had to write if you were to make this in base graphics? Thanks to ggplot2!

```
# Answer to the challenge.
ggplot(diamonds, aes(x=carat, y=price, color=cut, shape=color)) + geom_point()
```

3. The Labels

Now that you have drawn the main parts of the graph. You might want to add the plot's main title and perhaps change the X and Y axis titles. This can be accomplished using the `labs` layer, meant for specifying the labels. However, manipulating the size, color of the labels is the job of the 'Theme'.

```
library(ggplot2)
gg <- ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() +
labs(title="Scatterplot", x="Carat", y="Price")  # add axis lables and plot title.
print(gg)
```



The plot's main title is added and the X and Y axis labels capitalized.

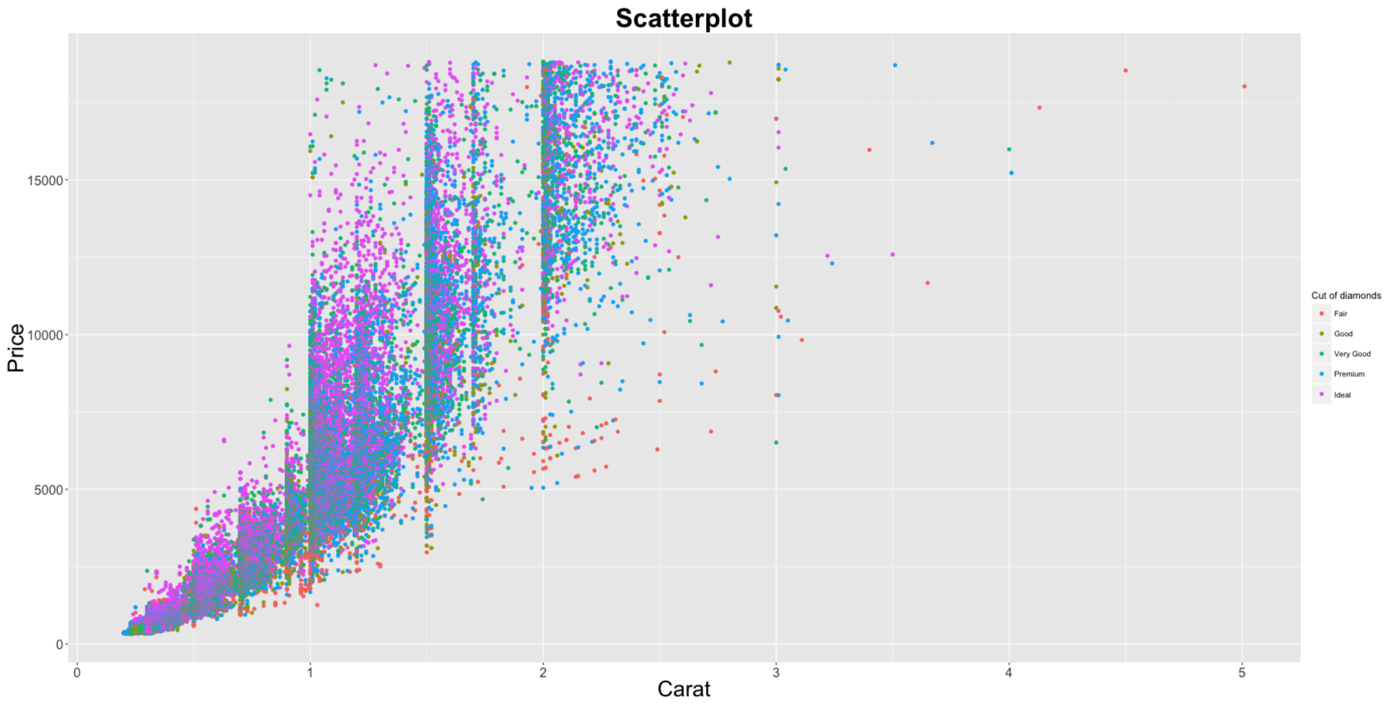
Note: If you are showing a ggplot inside a function, you need to explicitly save it and then print using the `print(gg)`, like we just did above.

4. The Theme

Almost everything is set, except that we want to increase the size of the labels and change the legend title. Adjusting the size of labels can be done using the `theme()` function by setting the `plot.title`, `axis.text.x` and `axis.text.y`. They need to be specified inside the `element_text()`. If you want to remove any of them, set it to `element_blank()` and it will vanish entirely.

Adjusting the legend title is a bit tricky. If your legend is that of a `color` attribute and it varies based in a factor, you need to set the `name` using `scale_color_discrete()`, where the *color* part belongs to the color attribute and the *discrete* because the legend is based on a factor variable.

```
gg1 <- gg + theme(plot.title=element_text(size=30, face="bold"),
                  axis.text.x=element_text(size=15),
                  axis.text.y=element_text(size=15),
                  axis.title.x=element_text(size=25),
                  axis.title.y=element_text(size=25)) +
  scale_color_discrete(name="Cut of diamonds")  # add title and axis text, change
legend title.
print(gg1)  # print the plot
```



If the legend shows a shape attribute based on a factor variable, you need to change it using `scale_shape_discrete(name="legend title")`. Had it been a continuous variable, use `scale_shape_continuous(name="legend title")` instead.

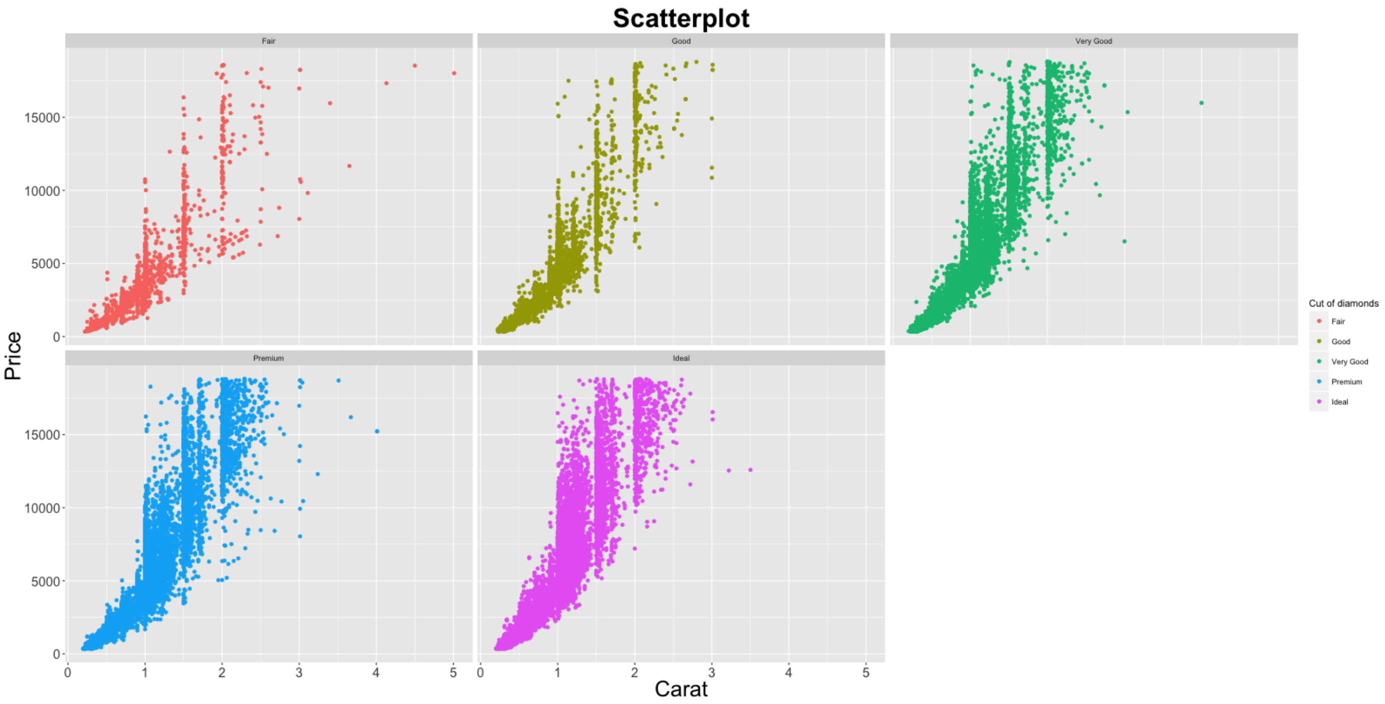
So now, Can you guess the function to use if your legend is based on a `fill` attribute on a continuous variable?

The answer is `scale_fill_continuous(name="legend title")`.

5. The Facets

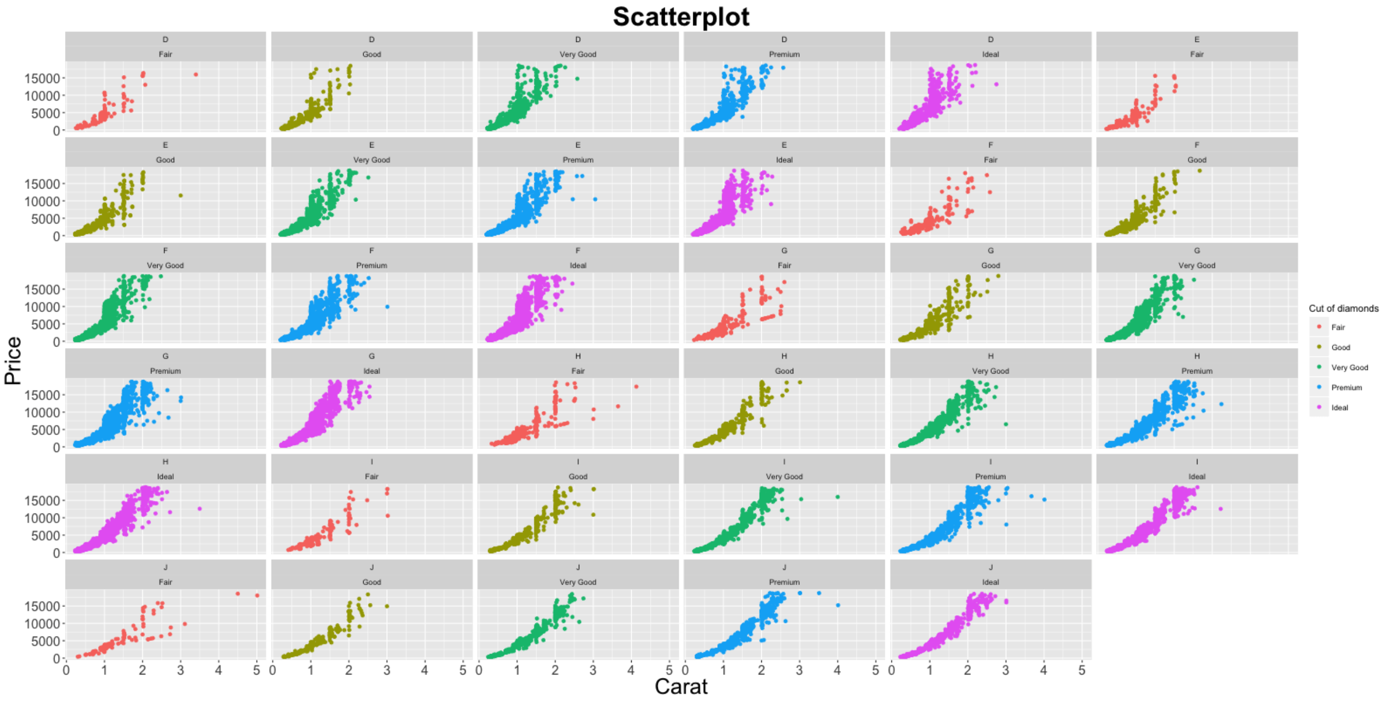
In the previous chart, you had the scatterplot for all different values of `cut` plotted in the same chart. What if you want one chart for one `cut`?

```
gg1 + facet_wrap( ~ cut, ncol=3)  # columns defined by 'cut'
```



`facet_wrap(formula)` takes in a formula as the argument. The item on the RHS corresponds to the column. The item on the LHS defines the rows.

```
gg1 + facet_wrap(color ~ cut)  # row: color, column: cut
```

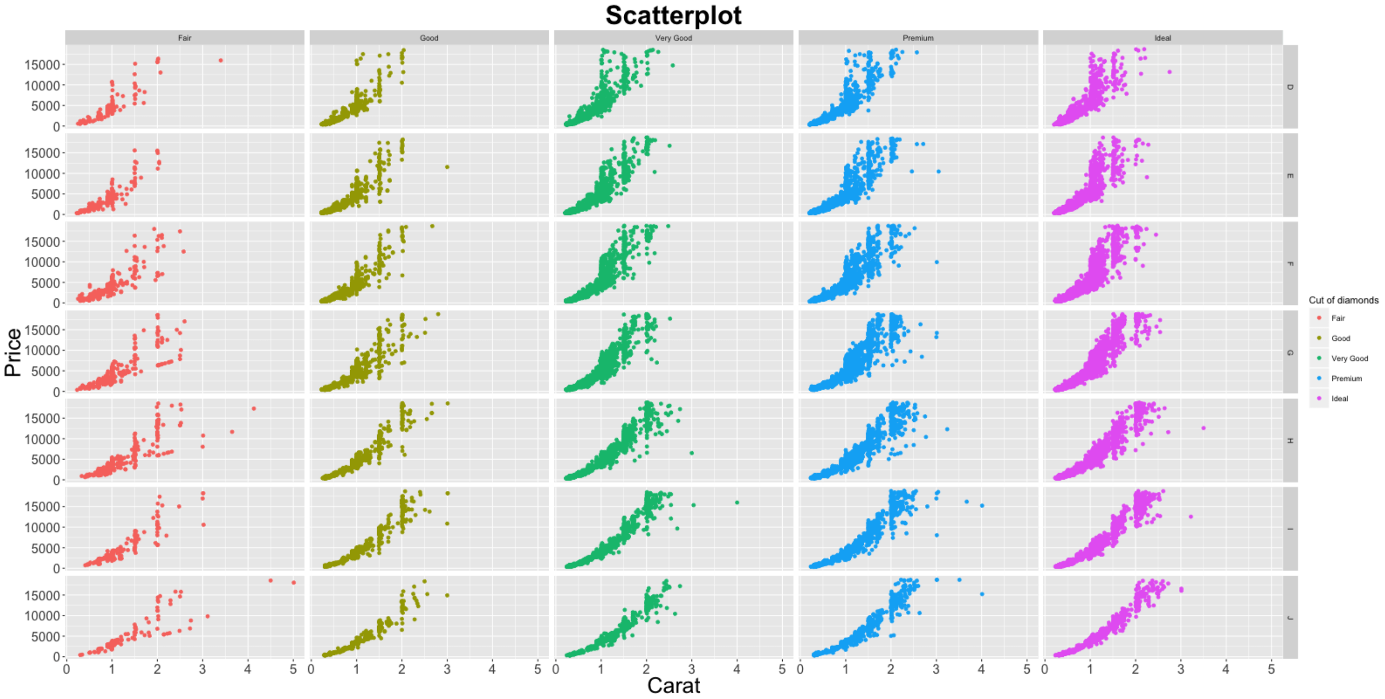


In `facet_wrap`, the scales of the X and Y axis are fixed to accomodate all points by default. This would make comparison of attributes meaningful because they would be in the same scale. However, it is possible to make the scales roam free making the charts look more evenly distributed by setting the argument `scales=free`.

```
gg1 + facet_wrap(color ~ cut, scales="free")  # row: color, column: cut
```

For comparison purposes, you can put all the plots in a grid as well using `facet_grid(formula)`.

```
gg1 + facet_grid(color ~ cut)   # In a grid
```



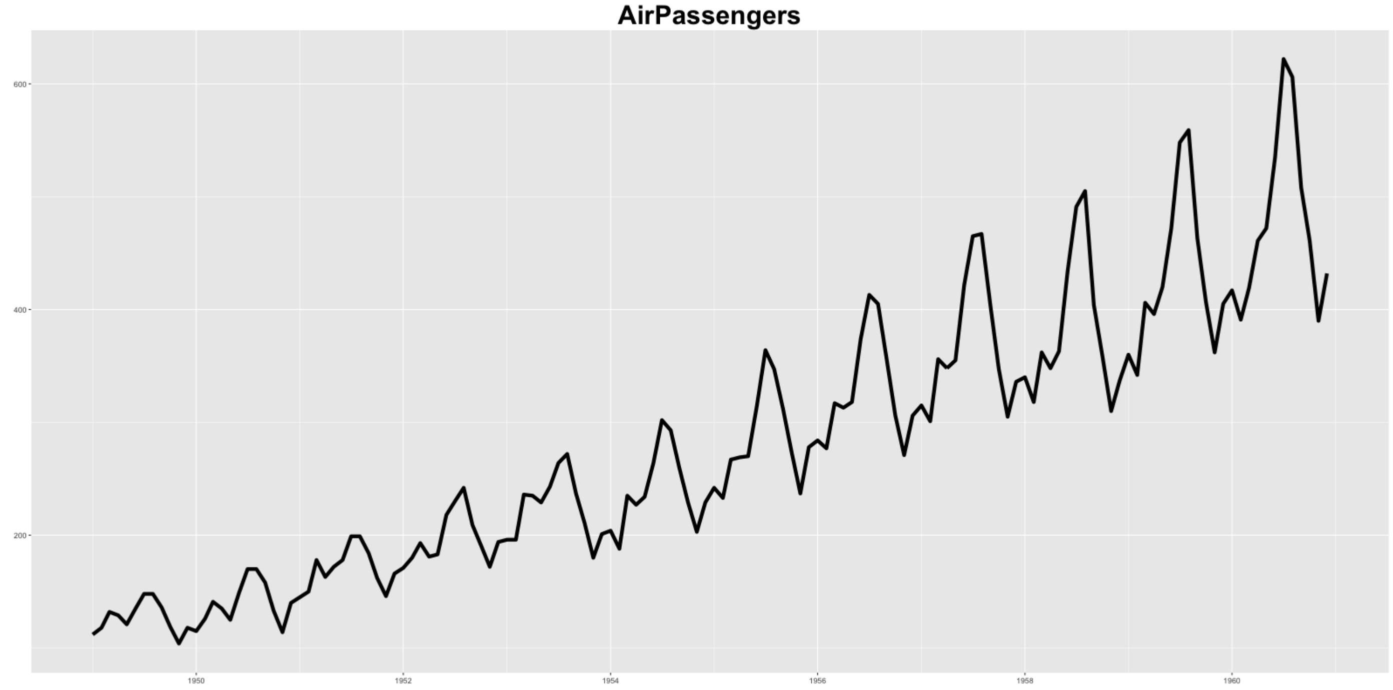
Note, the headers for individual plots are gone leaving more space for plotting area..

6. Commonly Used Features

6.1 Make a time series plot (using ggfortify)

The `ggfortify` package makes it very easy to plot time series directly from a time series object, without having to convert it to a dataframe. The example below plots the `AirPassengers` timeseries in one step. Cool!. See more ggfortify's autoplot options to plot time series here.

```
library(ggfortify)
autoplot(AirPassengers) + labs(title="AirPassengers")  # where AirPassengers is a
'ts' object
```



6.2 Plot multiple timeseries on same ggplot

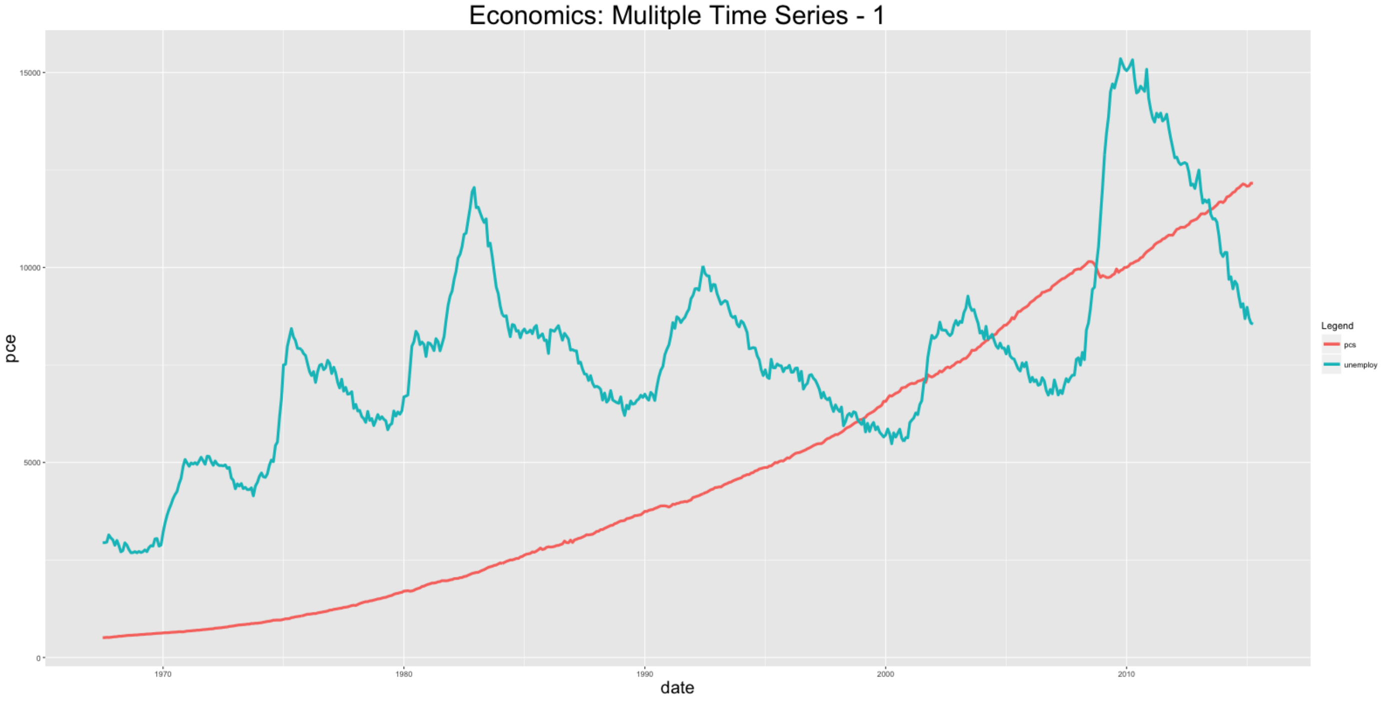
Plotting multiple timeseries requires that you have your data in dataframe format, in which one of the columns is the dates that will be used for X-axis.

Approach 1: After converting, you just need to keep adding multiple layers of time series one on top of the other.

Approach 2: Melt the dataframe using `reshape2::melt` by setting the `id` to the date field. Then just add one `geom_line` and set the color aesthetic to `variable` (which was created during the melt).

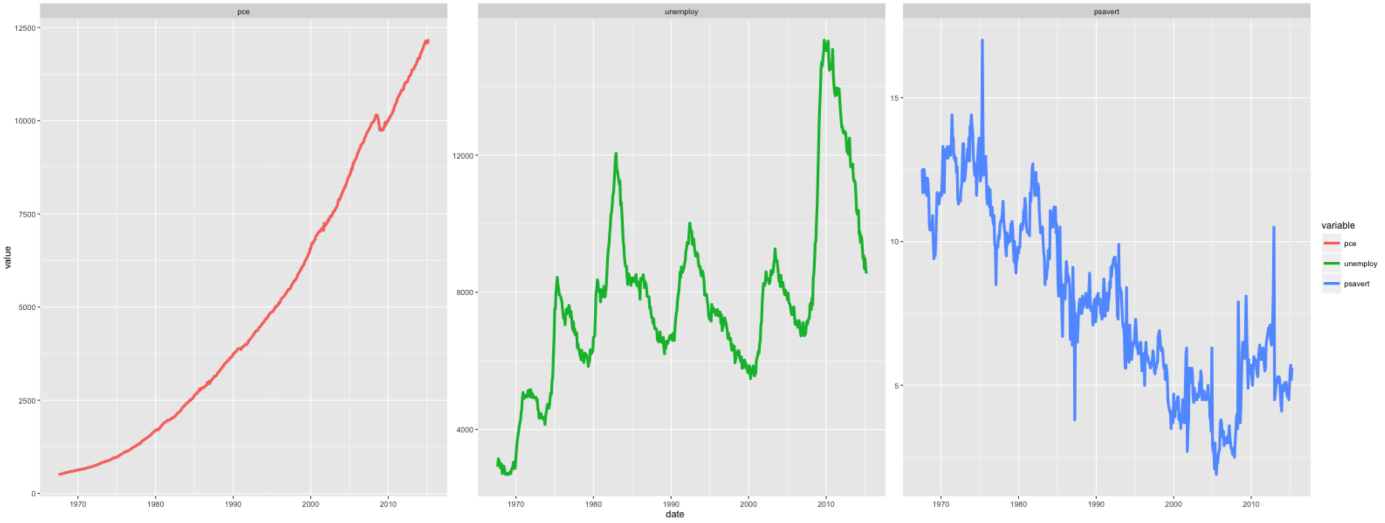
```
# Approach 1:
data(economics, package="ggplot2")  # init data
economics <- data.frame(economics)  # convert to dataframe
ggplot(economics) + geom_line(aes(x=date, y=pce, color="pcs")) +
geom_line(aes(x=date, y=unemploy, col="unemploy")) +
scale_color_discrete(name="Legend") + labs(title="Economics") # plot multiple time
series using 'geom_line's

# Approach 2:
library(reshape2)
df <- melt(economics[, c("date", "pce", "unemploy")], id="date")
ggplot(df) + geom_line(aes(x=date, y=value, color=variable)) +
labs(title="Economics")# plot multiple time series by melting
```



The disadvantage with ggplot2 is that it is not possible to get multiple Y-axis on the same plot. To plot multiple time series on the same scale can make few of the series appear small. An alternative would be to `facet_wrap` it and set the `scales='free'`.

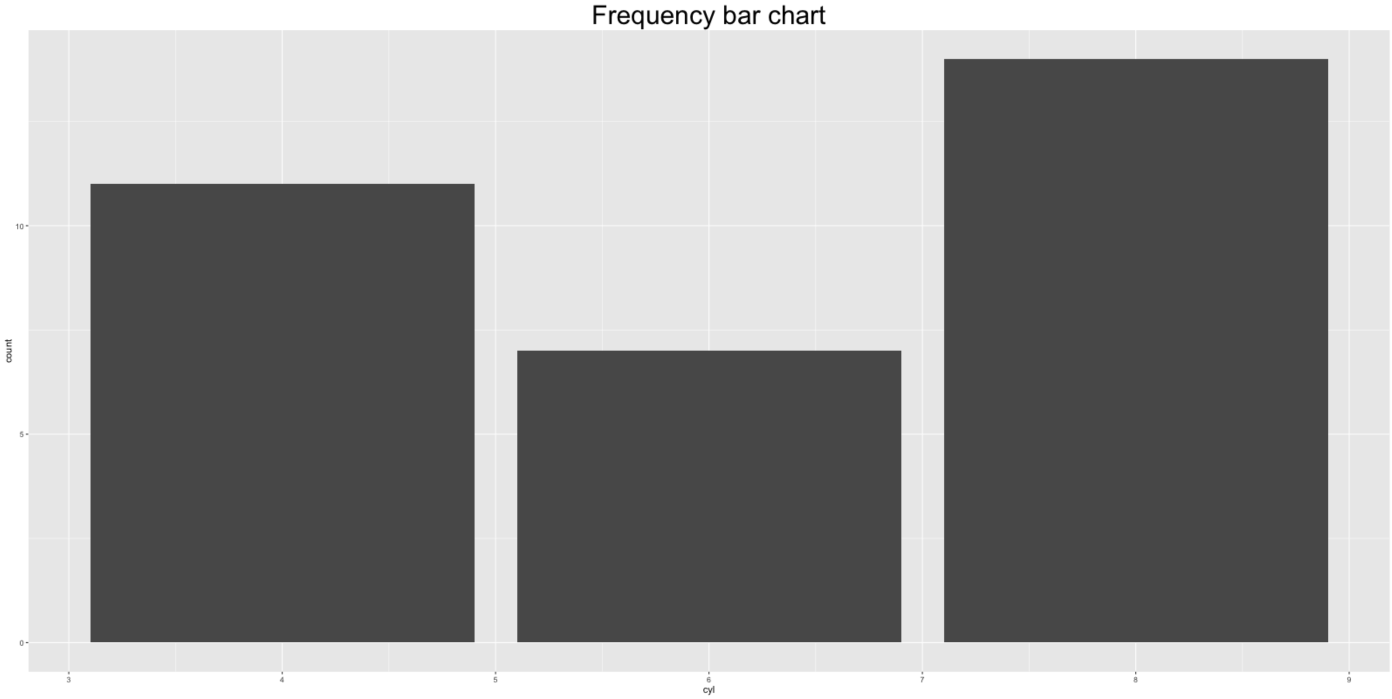
```
df <- melt(economics[, c("date", "pce", "unemploy", "psavert")], id="date")
ggplot(df) + geom_line(aes(x=date, y=value, color=variable))  + facet_wrap( ~
variable, scales="free")
```



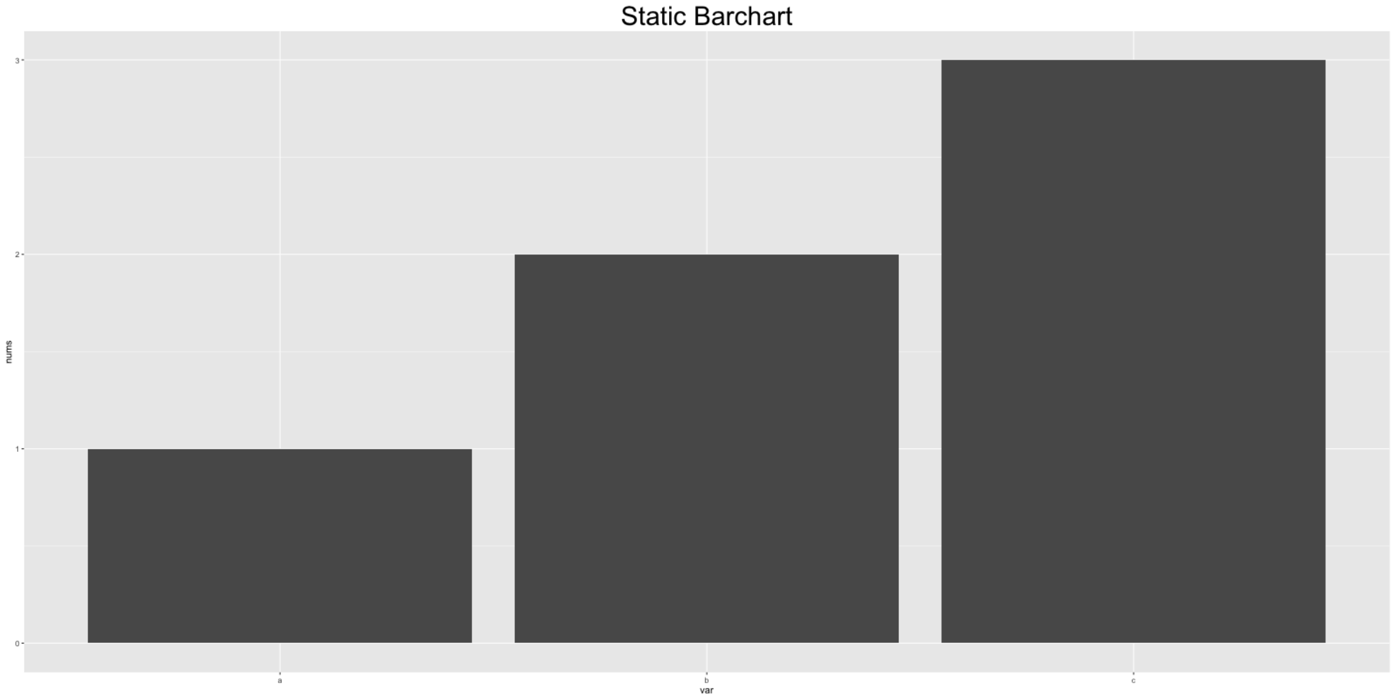
6.3 Bar charts

By default, ggplot makes a ‘counts’ barchart, meaning, it counts the frequency of items specified by the `x` aesthetic and plots it. In this format, you don’t need to specify the Y aesthetic. However, if you would like the make a bar chart of the absolute number, given by Y aesthetic, you need to set `stat="identity"` inside the `geom_bar`.

```
plot1 <- ggplot(mtcars, aes(x=cyl)) + geom_bar() + labs(title="Frequency bar
chart")  # Y axis derived from counts of X item
print(plot1)
```



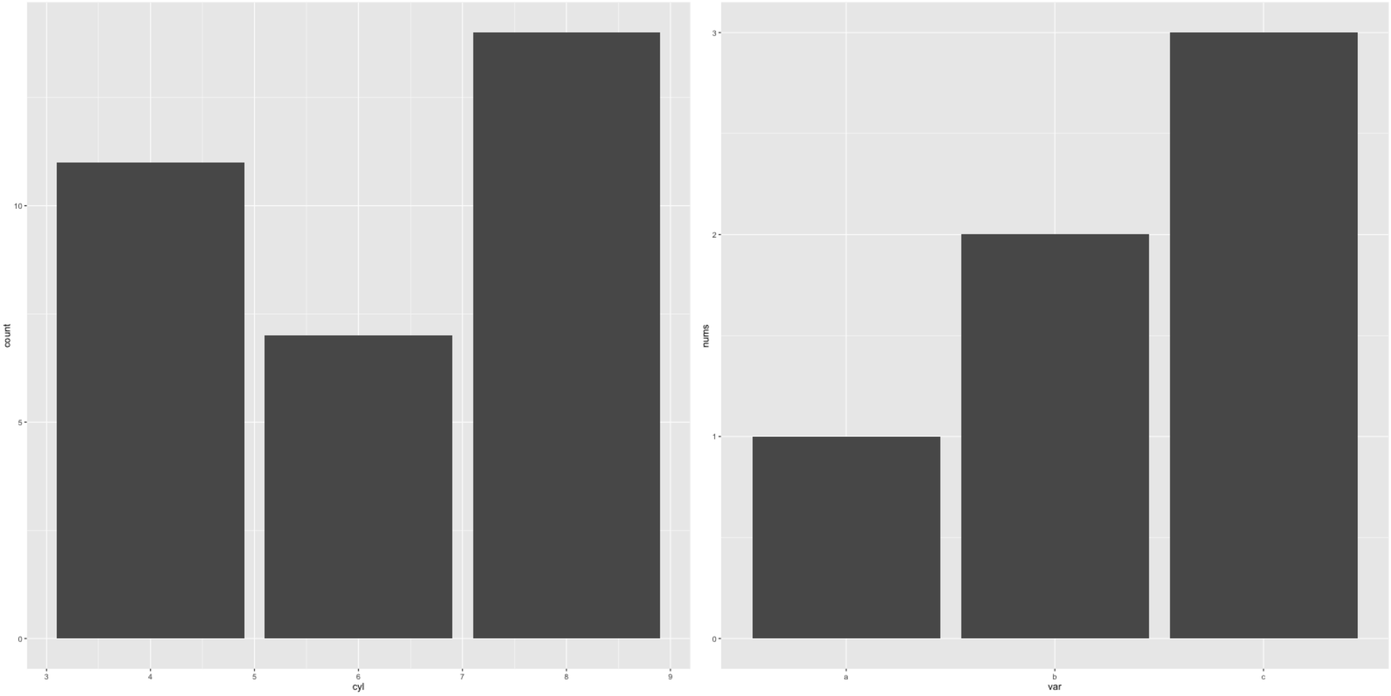
```
df <- data.frame(var=c("a", "b", "c"), nums=c(1:3))
plot2 <- ggplot(df, aes(x=var, y=nums)) + geom_bar(stat = "identity")  # Y axis is
explicit. 'stat=identity'
print(plot2)
```



6.4 Custom layout

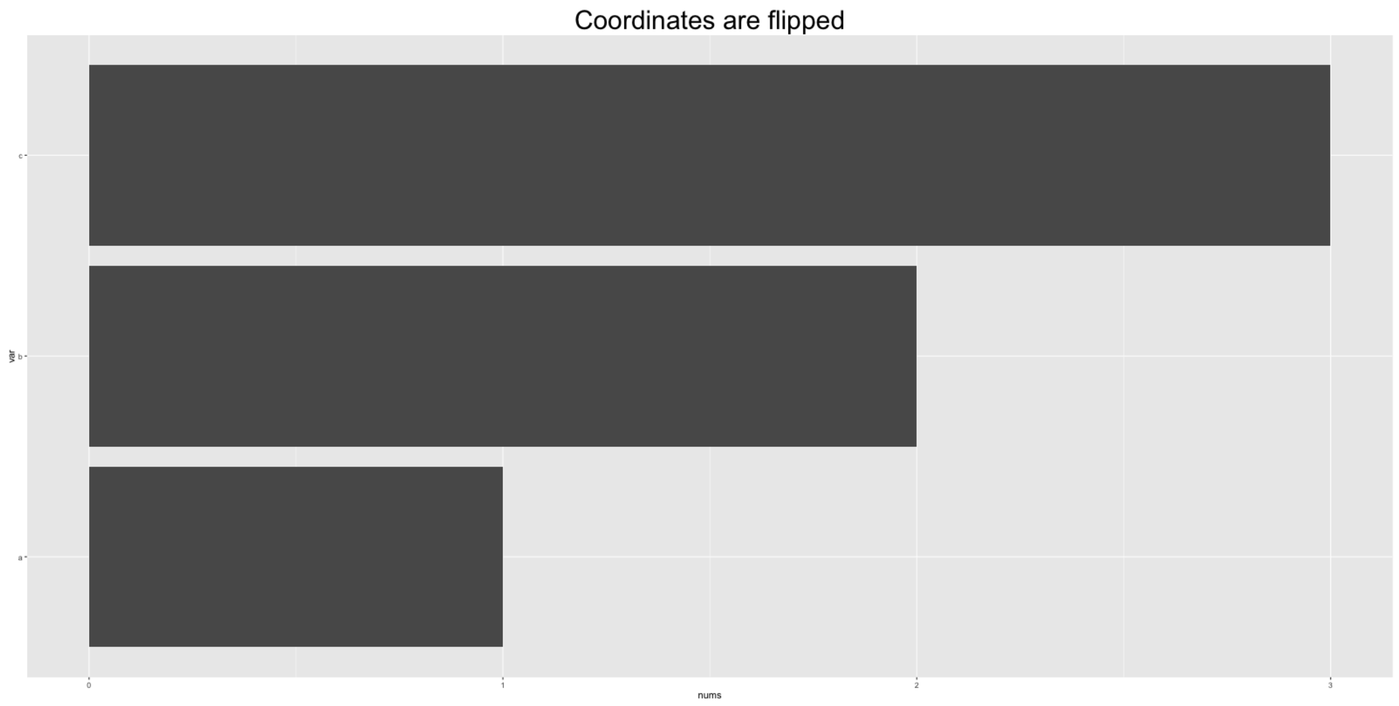
The `gridExtra` package provides the facility to arrage multiple ggplots in a single grid.

```
library(gridExtra)
grid.arrange(plot1, plot2, ncol=2)
```



6.5 Flipping coordinates

```
df <- data.frame(var=c("a", "b", "c"), nums=c(1:3))
ggplot(df, aes(x=var, y=nums)) + geom_bar(stat = "identity") + coord_flip() +
labs(title="Coordinates are flipped")
```



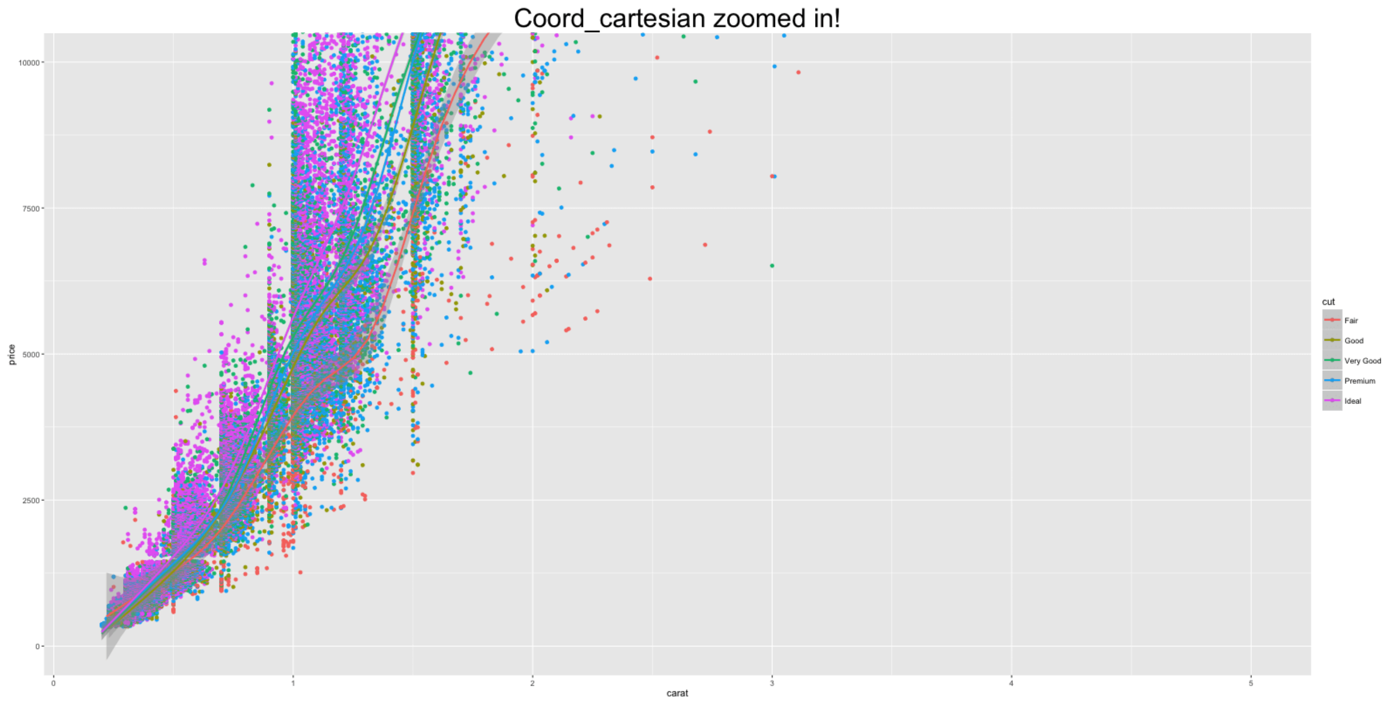
6.6 Adjust X and Y axis limits

There are 3 ways to change the X and Y axis limits.

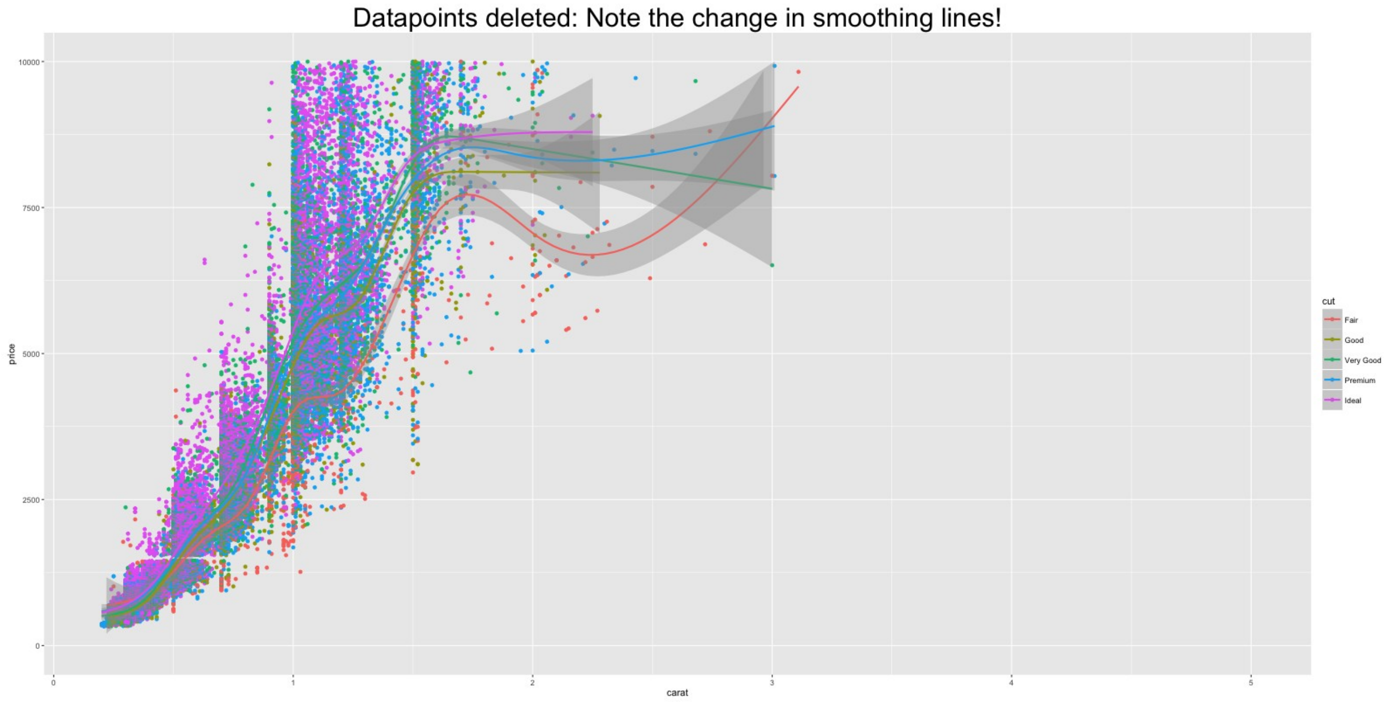
1. Using coord_cartesian(xlim=c(x1,x2))
2. Using xlim(c(x1,x2))
3. Using scale_x_continuous(limits=c(x1,x2))

Warning: Items 2 and 3 will delete the datapoints that lie outisde the limit from the data itself. So, if you add any smoothing line line and such, the outcome will be distorted. Item 1 (coord_cartesian) does not delete any datapoint, but instead zooms in to a specific region of the chart.

```
ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + geom_smooth() +
coord_cartesian(ylim=c(0, 10000)) + labs(title="Coord_cartesian zoomed in!")
```



```
ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + geom_smooth() +
ylim(c(0, 10000)) + labs(title="Datapoints deleted: Note the change in smoothing
lines!")
#> Warning messages:
#> 1: Removed 5222 rows containing non-finite values
#> (stat_smooth).
#> 2: Removed 5222 rows containing missing values
#> (geom_point).
```



6.7 Equal coordinates

Adding `coord_equal()` to ggplot sets the limits of X and Y axis to be equal. Below is a meaningless example. So to save face for not giving a good example, I am not showing you the output.

```
ggplot(diamonds, aes(x=price, y=price+runif(nrow(diamonds), 100, 10000),
color=cut)) + geom_point() + geom_smooth() + coord_equal()
```

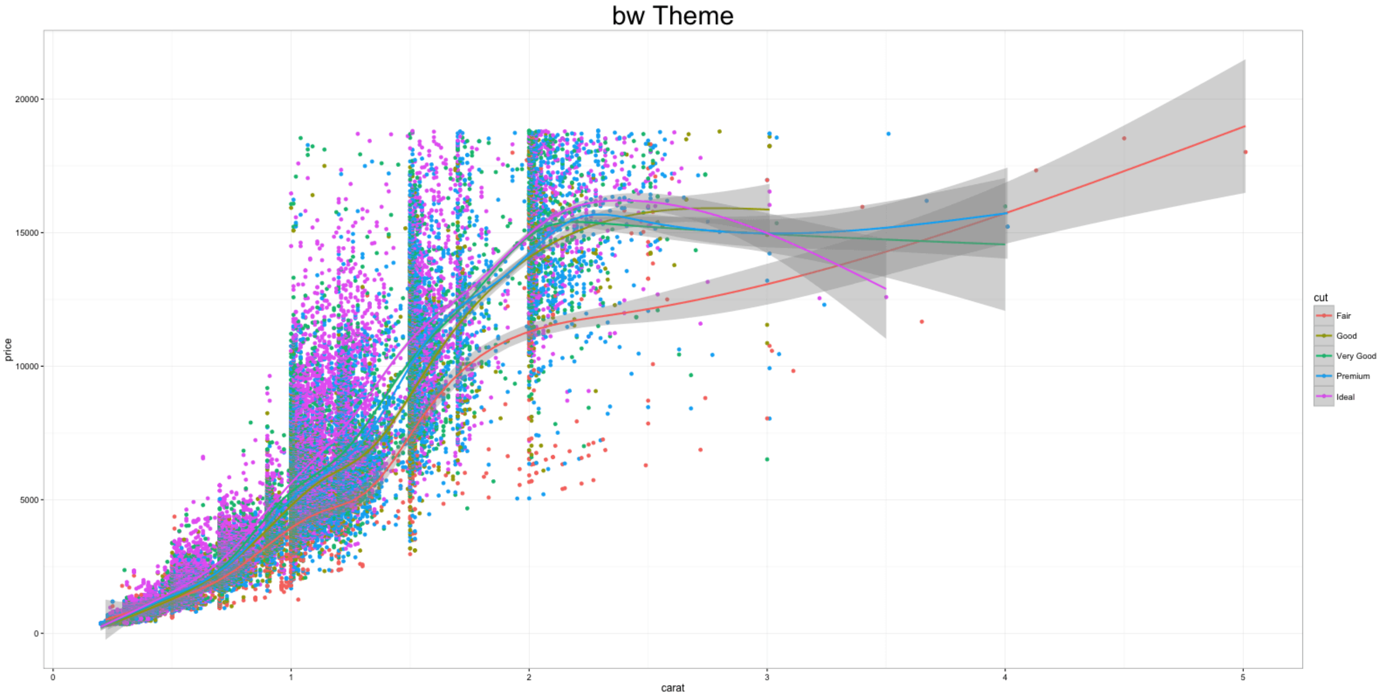
6.8 Change themes

Apart from the basic ggplot2 theme, you can change the look and feel of your plots using one of these builtin themes.

1. theme_gray()
2. theme_bw()
3. theme_linedraw()
4. theme_light()
5. theme_minimal()
6. theme_classic()
7. theme_void()

The `ggthemes` package provides [additional ggplot themes]() that imitates famous magazines and softwares. Here is an example of how to change the theme.

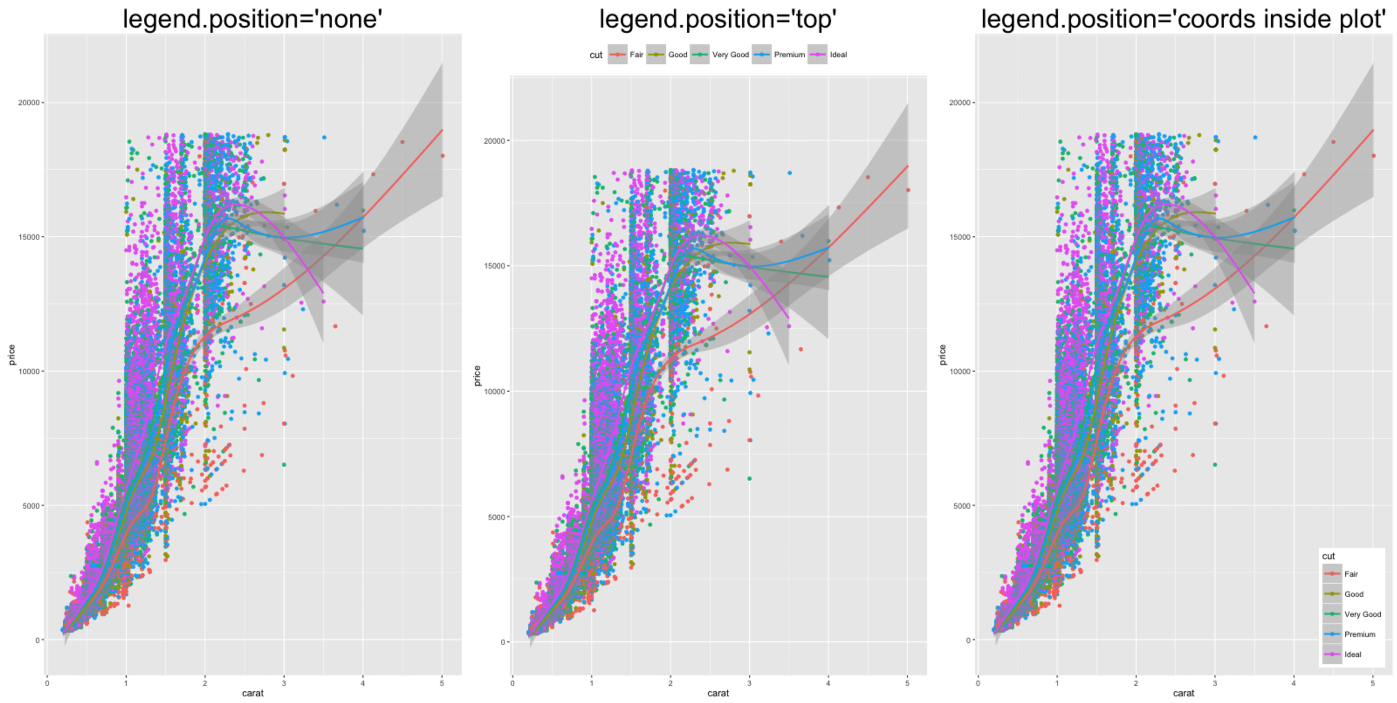
```
ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + geom_smooth()
+theme_bw() + labs(title="bw Theme")
```



6.9 Legend - Deleting and Changing Position

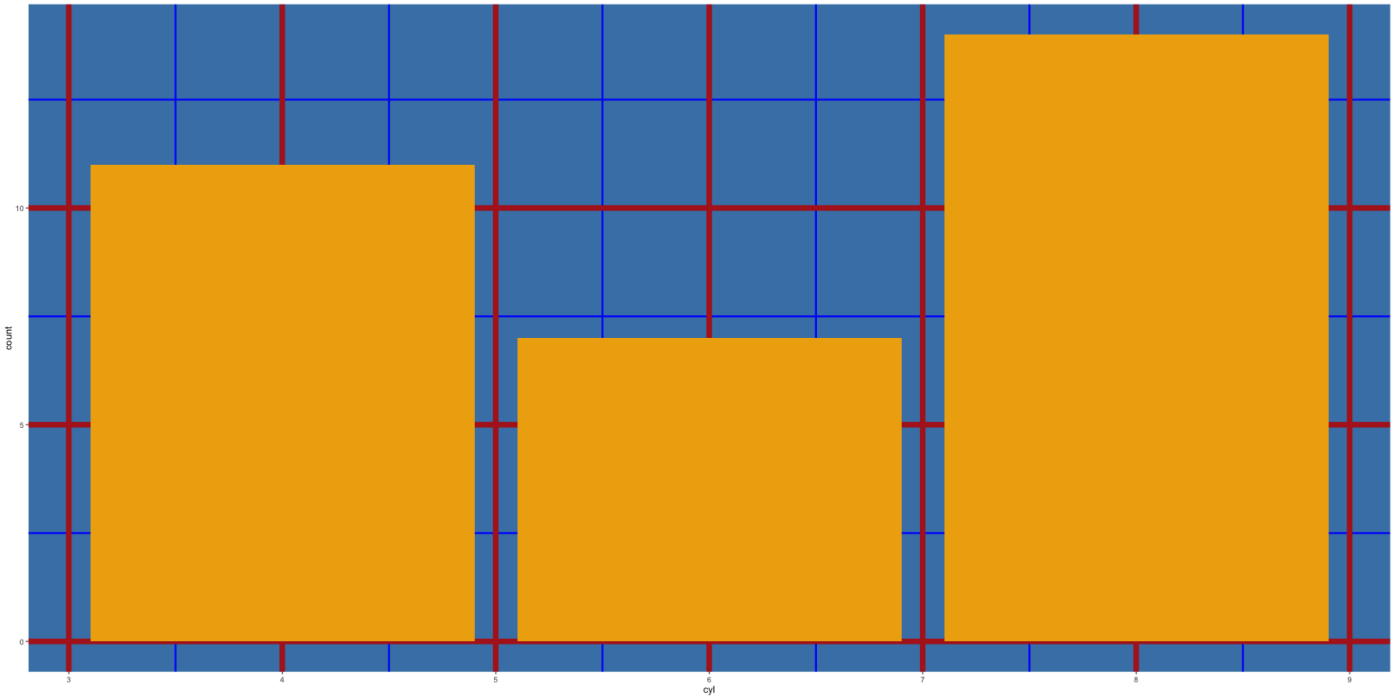
By setting `theme(legend.position="none")`, you can remove the legend. By setting it to ‘top’, i.e. `theme(legend.position="top")`, you can move the legend around the plot. By setting `legend.postion` to a co-ordinate inside the plot you can place the legend inside the plot itself. The `legend.justification` denotes the anchor point of the legend, i.e. the point that will be placed on the co-ordinates given by `legend.position`.

```
p1 <- ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() +
geom_smooth() + theme(legend.position="none") +
labs(title="legend.position='none'")  # remove legend
p2 <- ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() +
geom_smooth() + theme(legend.position="top") + labs(title="legend.position='top'")
# legend at top
p3 <- ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() +
geom_smooth() + labs(title="legend.position='coords inside plot'") +
theme(legend.justification=c(1,0), legend.position=c(1,0))  # legend inside the
plot.
grid.arrange(p1, p2, p3, ncol=3)  # arrange
```



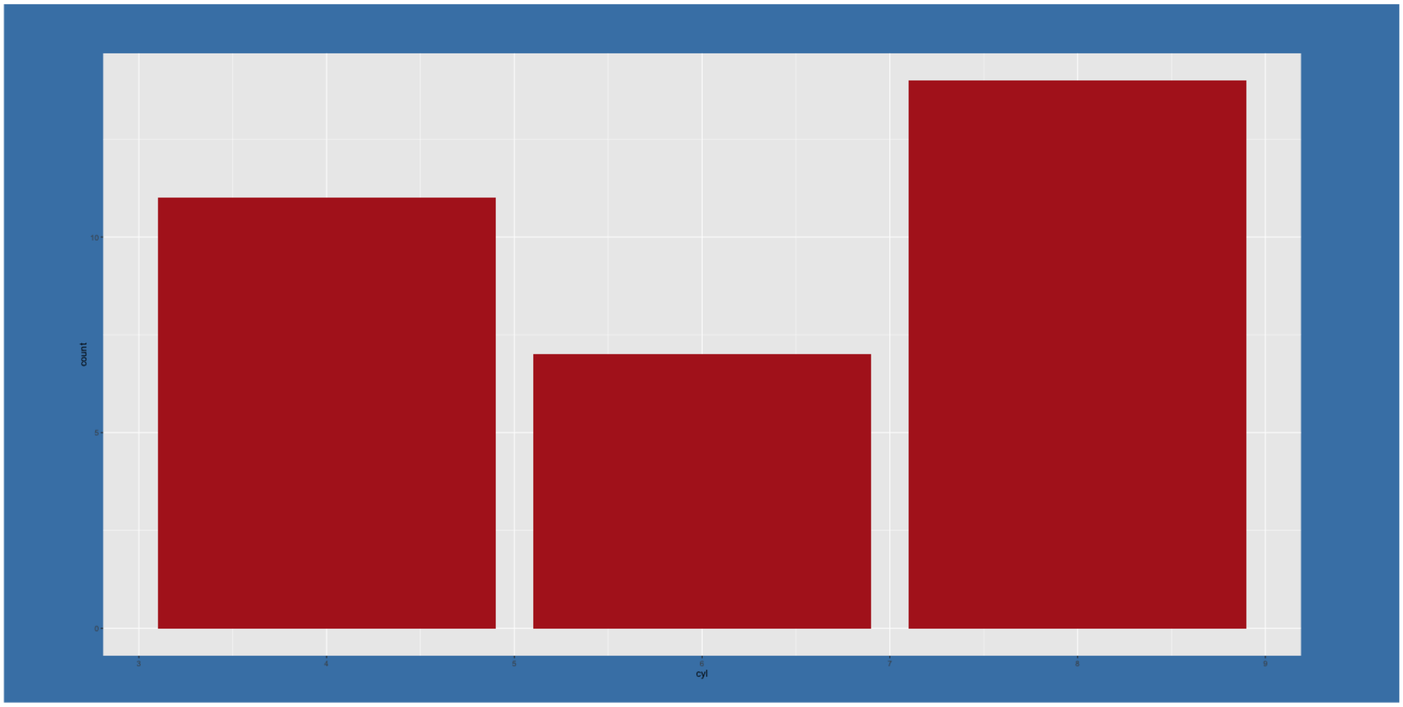
6.10 Grid lines

```
ggplot(mtcars, aes(x=cyl)) + geom_bar(fill='darkgoldenrod2') +
  theme(panel.background = element_rect(fill = 'steelblue'),
  panel.grid.major = element_line(colour = "firebrick", size=3),
  panel.grid.minor = element_line(colour = "blue", size=1))
```



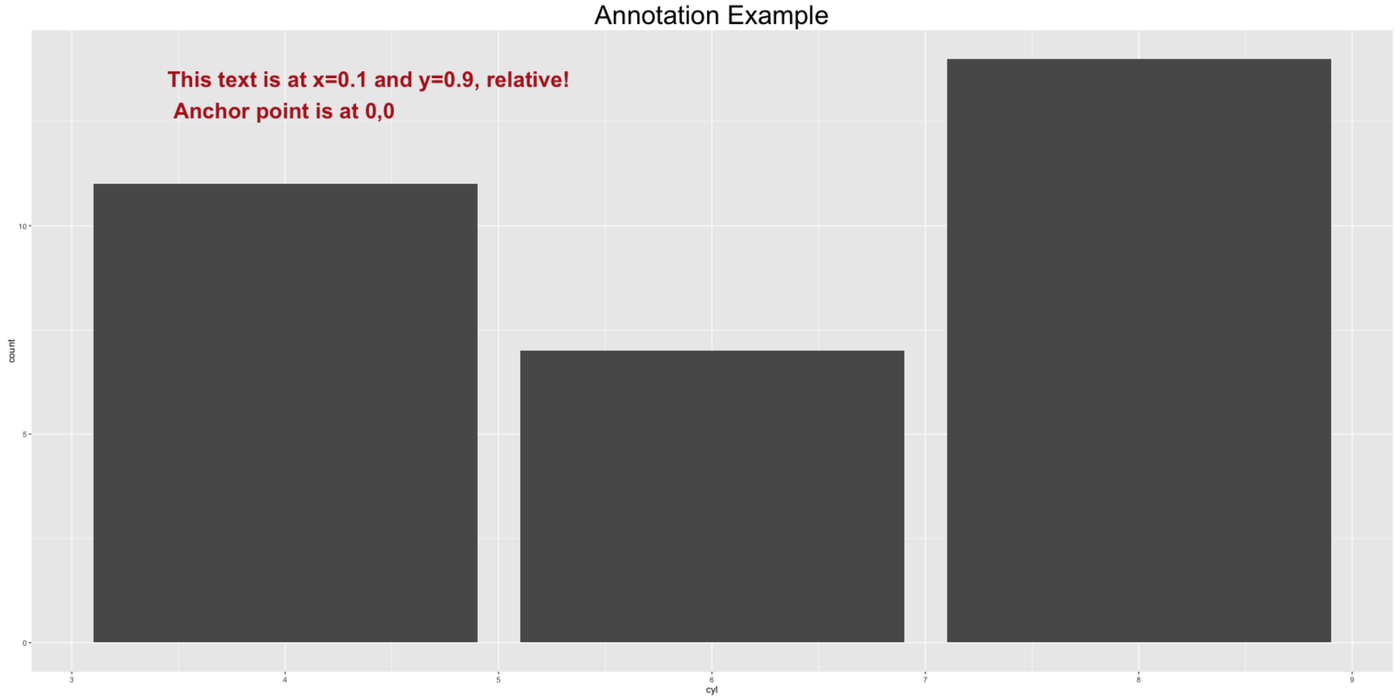
6.11 Plot margin and background

```
ggplot(mtcars, aes(x=cyl)) + geom_bar(fill="firebrick") +
theme(plot.background=element_rect(fill="steelblue"), plot.margin = unit(c(2, 4, 1,
3), "cm")) # top, right, bottom, left
```



6.12 Annotation

```
library(grid)
my_grob = grobTree(textGrob("This text is at x=0.1 and y=0.9, relative!\n Anchor
point is at 0,0", x=0.1,  y=0.9, hjust=0,
  gp=gpar(col="firebrick", fontsize=25, fontface="bold")))
ggplot(mtcars, aes(x=cyl)) + geom_bar() + annotation_custom(my_grob) +
labs(title="Annotation Example")
```



6.13 Saving ggplot

```
plot1 <- ggplot(mtcars, aes(x=cyl)) + geom_bar()
ggsave("myggplot.png")  # saves the last plot.
ggsave("myggplot.png", plot=plot1)  # save a stored ggplot
```