ggplot2 notes

Daijiang Li

January 18, 2013

1 Getting started with qplot (quick-plot)

1.1 Basic use

```
library("ggplot2", lib.loc = "C:/Program Files/R/library")
library(mgcv)
set.seed(1410)
dsmall <- diamonds[sample(nrow(diamonds), 100), ]
```

```
## Basic use
qplot(carat, price, data = diamonds)
qplot(log(carat), log(price), data = diamonds)
qplot(carat, x * y * z, data = diamonds)
# aesthetic attributes
qplot(carat, price, data = diamonds, alpha = I(1/10))
# alpha aesthetics using I(), value from 0-comp transparent to 1,
# e.g.1/10.
# colour, shape
qplot(carat, price, data = dsmall, colour = color)
qplot(carat, price, data = dsmall, shape = cut)
```

1.2 Plot geom

Geom, short for geometric object, describes the type of object that is used to display the data. Here we'll introduce the most common and useful geoms, organised by the dimensionality of data that they work with.

The following geoms enable you to investigate two-dimensional relationships:

- geom = "point" draws points to produce a scatterplot. This is the default when you supply both x and y arguments to qplot().
- geom = "smooth" fits a smoother to the data and displays the smooth and its standard error.
- geom = "boxplot" produces a box and whisker plot to summarise the distribution of a set of points.
- geom = "path" and geom = "line" draw lines between the data points. Traditionally these are used to explore relationships between time and another variable, but lines may be used to join observations connected in some other way. A line plot is constrained to produce lines that travel from left to right, while paths can go in any direction.

For 1d distributions, your choice of geoms is guided by the variable type:

- For continuous variables, geom = "histogram" draws a histogram, geom = "freqpoly" a frequency polygon, and geom = "density" creates a density plot. The **histogram geom is the default when you only supply an x value to qplot()**.
- For discrete variables, geom = "bar" makes a bar chart.

```
# points and smooth
qplot(carat, price, data = dsmall, geom = c("point", "smooth"))
qplot(carat, price, data = dsmall, geom = c("point", "smooth"), se = FALSE)  # turn off CI
qplot(carat, price, data = dsmall, geom = c("point", "smooth"), span = 0.5)  # wiggliness
```

Methods for smooth:

- for small n, default is: method= "loess". Wiggliness of the line is controlled by span from 0 (exceedingly wiggly) to 1 (not so wiggly, smooth).
- You could also load the `mgcv` library and use $method = "gam", formula = y\ s(x)$ to fit a generalised additive model. This is similar to using a spline with lm, but the degree of smoothness is estimated from the data. For large data, use the formula $y\ s(x, bs = "cs")$. This is used by default when there are more than 1,000 points.
- $method = "lm"$fits a linear model. The default will fit a straight line to your data, or you can specify $formula = y\ poly(x, 2)$ to specify a degree 2 polynomial, or better, load splines package and use a natural spline: $formula = y\ ns(x, 2)$.
- $method = "rlm"$works like lm, but uses a robust fitting algorithm so that outliers don't affect the fit as much.

```
library(mgcv)
qplot(carat, price, data = dsmall, geom = c("point", "smooth"), method = "gam",
    formula = y ~ s(x, bs = "cs"))
```

How the values of the continuous variables vary with the levels of the categorical variable? We can use boxplots and jittered points. For jittered points, qplot offers the same control over aesthetics as it does for a normal scatterplot: size, colour and shape. For boxplots you can control the outline colour, the internal fill colour and the size of the lines.

```
# jitter and boxplot
qplot(color, price/carat, data = diamonds, geom = "jitter", alpha = I(1/10))
qplot(color, price/carat, data = diamonds, geom = "boxplot")
qplot(color, price/carat, data = diamonds, geom = c("jitter", "boxplot"), alpha = I(0.5),
    colour = color)
```

Histogram and density plots show the distribution of a single variable. For the density plot, the adjust argument controls the degree of smoothness (high values of adjust produce smoother plots). For the histogram, the binwidth argument controls the amount of smoothing by setting the bin size. (Break points can also be specified explicitly, using the breaks argument.)

```
# histogram and density
qplot(carat, data = diamonds, geom = "histogram", binwidth = 0.1, xlim = c(0,
    3), colour = I("darkgreen"), fill = I("white"))
qplot(carat, data = diamonds, geom = "histogram", fill = color)
qplot(carat, data = diamonds, geom = "density")
qplot(carat, data = diamonds, geom = "density", adjust = 0.1)
qplot(carat, data = diamonds, geom = "density", colour = color)
```

Bar charts. The discrete analogue of histogram is the bar chart, geom = "bar". The bar geom counts the number of instances of each class so that you don't need to tabulate your values beforehand, as with barchart in base R. If the data has already been tabulated or if you'd like to tabulate class members in some other way, such as by summing up a continuous variable, you can use the weight geom.

```
# barchart
qplot(color, data = diamonds, geom = "bar")  # try fill=color
qplot(color, data = diamonds, geom = "bar", weight = carat) + scale_y_continuous("carat")
```

Line and path plots are typically used for time series data. Line plots join the points from left to right, while path plots join them in the order that they appear in the dataset (a line plot is just a path plot of the data sorted by x value). Line plots usually have time on the x-axis, showing how a single variable has changed over time. Path plots show how two variables have simultaneously changed over time, with time encoded in the way that the points are joined together.

```
# Line and path graph
qplot(date, unemploy/pop, data = economics, geom = "line")
year <- function(x) as.POSIXlt(x)$year + 1900  # e.g.1967
qplot(unemploy/pop, uempmed, data = economics, geom = c("point", "path"))
qplot(unemploy/pop, uempmed, data = economics, geom = "path", colour = year(date)) +
    scale_area()
```

1.3 Faceting

We have already discussed using aesthetics (colour and shape) to compare subgroups, drawing all groups on the same plot. Faceting takes an alternative approach: It creates tables of graphics by splitting the data into subsets and displaying the same graph for each subset in an arrangement that facilitates comparison. The default faceting method in qplot() creates plots arranged on a grid specified by a faceting formula which looks like row var~col var.To facet on only one of columns or rows, use . as a place holder. For example, row var ~ . will create a single column with multiple rows. The ..density.. syntax is new. The y-axis of the histogram does not come from the original data, but from the statistical transformation that counts the number of observations in each bin. Using ..density.. tells ggplot2 to map the density to the y-axis instead of the default use of count.

```
# Facet
qplot(carat, data = diamonds, facets = color ~ ., geom = "histogram", binwidth = 0.1,
    xlim = c(0, 3))
qplot(carat, ..density.., data = diamonds, facets = color ~ ., geom = "histogram",
    binwidth = 0.1, xlim = c(0, 3))
```

1.4 Other options

These are a few other qplot options to control the graphic's appearance. These all have the same effect as their plot equivalents: xlim, ylim, log, main, xlab, ylab, etc.

2 Mastering the grammer

2.1 Scatterplot

```
# scatterplot
qplot(displ, hwy, data = mpg, colour = factor(cyl))
```

A scatterplot represents each observation as a point (•), positioned according to the value of two variables. As well as a horizontal and vertical position, each point also has *a size, a colour and a shape*. These attributes are called **aesthetics**, and are the properties that can be perceived on the graphic.

Points, lines and bars are all examples of geometric objects, or **geoms**. Geoms determine the "type" of the plot.

We need to convert them from data units (e.g., litres, miles per gallon and number of cylinders) to physical units (e.g., pixels and colours) that the computer can display. This conversion process is called **scaling** and performed by scales.

A final step determines how the two positions (x and y) are combined to form the final location on the plot. This is done by the coordinate system, or **coord**. In most cases this will be Cartesian coordinates, but it might be polar coordinates, or a spherical projection used for a map.

Finally, we need to render this data to create the graphical objects that are displayed on the screen. To create a complete plot we need to combine graphical objects from three sources: the *data*, represented by the point geom; the *scales and coordinate system*, which generate axes and legends so that we can read values from the graph; and *plot annotations*, such as the background and plot title.

2.2 A more complex plot

```
qplot(displ, hwy, data = mpg, facets = . ~ year) + geom_smooth()
```

This plot adds three new components to the mix: facets, multiple layers and statistics. The facets and layers expand the data structure described above: each facet panel in each layer has its own dataset. You can think of this as a 3d array: the panels of the facets form a 2d grid, and the layers extend upwards in the 3rd dimension. In this case the data in the layers is the same, but in general *we can plot different datasets on different layers*.

The smooth layer is different to the point layer because it doesn't display the raw data, but instead displays a statistical transformation of the data. Specifically, the smooth layer fits a smooth line through the middle of the data. This requires an additional step in the process described above: after mapping the data to aesthetics, the data is passed to a statistical transformation, or **stat**, which manipulates the data in some useful way.

As well as adding an additional step to summarise the data, we also need some extra steps when we get to the scales. This is because we now have multiple datasets (for the different facets and layers) and we need to make sure that the scales are the same across all of them. Scaling actually occurs in three parts: transforming, training and mapping. Scale transformation occurs before statistical transformation so that statistics are computed on the scale-transformed data. This ensures that a plot of log(x) vs. log(y) on linear scales looks the same as x vs. y on log scales. There are many different transformations that can be used, including taking square roots, logarithms and reciprocals.After the statistics are computed, each scale is trained on every dataset from all the layers and facets. The training operation combines the ranges of the individual datasets to get the range of the complete data. Without this step, scales could only make sense locally and we wouldn't be able to overlay different layers because their positions wouldn't line up. Finally the scales map the data values into aesthetic values.

Faceting, a general case of the conditioned or trellised plots, makes it easy to create small multiples each showing a different subset of the whole dataset.

2.3 Data structures

A plot object is a list with components *data, mapping* (the default aesthetic mappings), *layers, scales, coordinates* and *facet*. The plot object has one other component we haven't discussed yet: *options*.

Plots can be created in two ways: all at once with `qplot()`, as shown in the previous chapter, or piece-by-piece with `ggplot()` and layer functions. Once you have a plot object, there are a few things you can do with it:

- Render it on screen, with `print()`. This happens automatically when running interactively, but inside a loop or function, you'll need to `print()` it yourself.
- Render it to disk, with `ggsave()`.
- Briefly describe its structure with `summary()`.
- Save a cached copy of it to disk, with `save()`. This saves a complete copy of the plot object, so you can easily re-create that exact plot with `load()`. Note that data is stored inside the plot, so that if you change the data outside of the plot, and then redraw a saved plot, it will not be updated.

```
p <- qplot(displ, hwy, data = mpg, colour = factor(cyl))
summary(p)
save(p, file = "plot.rdata")  # Save plot object to disk
load("plot.rdata")  # Load from disk
ggsave("plot.png", width = 5, height = 5)  # Save png to disk
```

3 Build a plot layer by layer

Layering is the mechanism by which additional data elements are added to a plot. Each layer can come from a different dataset and have a different aesthetic mapping, allowing us to create plots that could not be generated using `qplot()`, which permits only a single dataset and a single set of aesthetic mappings.

3.1 Creating a plot

When we used `qplot()`, it did a lot of things for us: it created a plot object, added layers, and displayed the result, using many default values along the way. To create the plot object ourselves, we use `ggplot()`. This has two arguments: **data** and aesthetic **mapping**. These arguments set up defaults for the plot and can be omitted if you specify data and aesthetics when adding each layer. You are already familiar with aesthetic mappings from `qplot()`, and the syntax here is quite similar, although you need to wrap the pairs of aesthetic attribute and variable name in the `aes()` function.

```
p <- ggplot(diamonds, aes(carat, price, colour = cut))
# This plot object cannot be displayed until we add a layer: there is
# nothing to see!
```

3.2 Layers

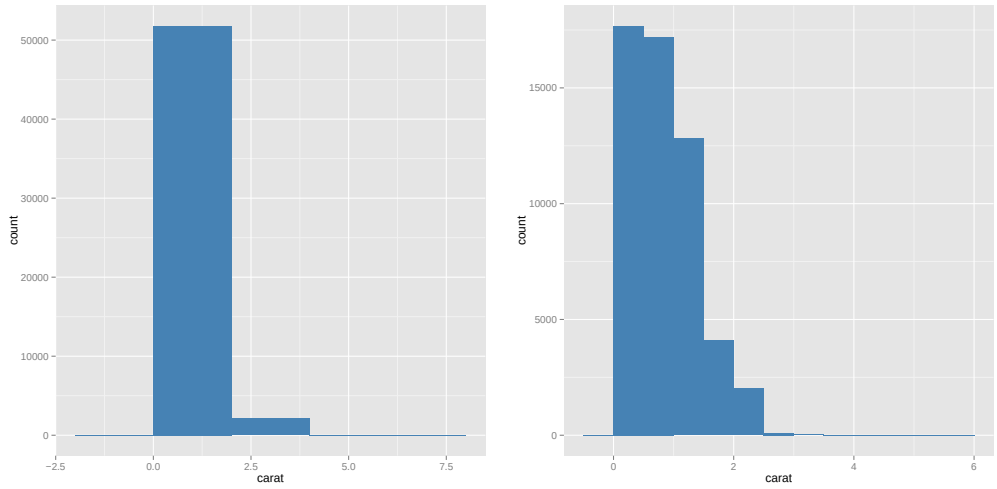
A minimal layer may do nothing more than specify a geom, a way of visually representing the data.

```
p <- p + layer(geom = "point")
```

This layer uses the plot defaults for data and aesthetic mapping and it uses default values for two optional arguments: the statistical transformation (the stat) and the position adjustment. A more fully specified layer can take any or all of these arguments:

```
# arguments for layers
layer(geom, geom_params, stat_params, data, mapping, position)
```

```
p <- ggplot(diamonds, aes(x = carat))
p1 <- p + layer(geom = "bar", geom_params = list(fill = "steelblue"), stat = "bin",
    stat_params = list(binwidth = 2))
q <- ggplot(diamonds, aes(x = carat))
q <- p + layer(geom = "bar", geom_params = list(fill = "steelblue"), stat = "bin",
    stat_params = list(binwidth = 0.5))
p1
q
```



This layer specification is precise but verbose. We can simplify it by using shortcuts that rely on the fact that *every geom is associated with a default statistic and position, and every statistic with a default geom*. This means that you only need to specify one of stat or geom to get a completely specified layer, with parameters passed on to the geom or stat as appropriate. This expression generates the same layer as the full layer command above:

```
p <- ggplot(diamonds, aes(x = carat))
p <- p + geom_histogram(binwidth = 2, fill = "steelblue")
p
```

All the shortcut functions have the same basic form, beginning with geom_ or stat_:

```
# arguments for geom_ or stat_
geom_XXX(mapping, data, ..., geom, position)
stat_XXX(mapping, data, ..., stat, position)
# mapping first, since almost always use the same data.
```

- mapping (optional): A set of aesthetic mappings, specified using the `aes()` function and combined with the plot defaults.
- data (optional): A dataset which overrides the default plot dataset. It is most commonly omitted, in which case the layer will use the default plot data.
- ...: Parameters for the geom or stat, such as bin width in the histogram or bandwidth for a loess smoother. You can also use aesthetic properties as parameters. When you do this you set the property to a fixed value, not map it to a variable in the dataset. The example above showed setting the fill colour of the histogram to "steelblue".
- geom or stat (optional): You can override the default stat for a geom, or the default geom for a stat. This is a text string containing the name of the geom to use. Using the default will give you a standard plot; overriding the default allows you to achieve something more exotic

- position (optional): Choose a method for adjusting overlapping objects.

Layers can be added to plots created with `ggplot()` or `qplot()`. Remember, behind the scenes, `qplot()` is doing exactly the same thing: it creates a plot object and then adds layers. The following example shows the equivalence among these ways of making plots.

```
qplot(sleep_rem/sleep_total, awake, data = msleep) + geom_smooth()
qplot(sleep_rem/sleep_total, awake, data = msleep, geom = c("point", "smooth"))
ggplot(msleep, aes(sleep_rem/sleep_total, awake)) + geom_point() + geom_smooth()
# store layer as variable, to reduce duplication
bestfit <- geom_smooth(method = "lm", se = F, colour = "steelblue", alpha = 1,
    size = 1)
qplot(sleep_rem, sleep_total, data = msleep) + bestfit
qplot(awake, brainwt, data = msleep, log = "y") + bestfit
```

3.3 Data

The restriction on the data is simple: it must be a data frame. You can replace the old dataset with %+%, as shown in the following example.

```
p <- ggplot(mtcar, aes(mpg, wt, colour = cyl)) + geom_point()
p
mtcar <- transform(mtcars, mpg = mpg^2)
p %+% mtcar
```

Any change of values or dimensions is legitimate. However, if a variable changes from discrete to continuous (or vice versa), you will need to change the default scales, It is not necessary to specify a default dataset except when using faceting; faceting is a global operation (i.e., it works on all layers) and it needs to have a base dataset which defines the set of facets for all datasets.

The data is stored in the plot object as a copy, not a reference. This has two important consequences: if your data changes, the plot will not; and ggplot2 objects are entirely self-contained so that they can be save()d to disk and later load()ed and plotted without needing anything else from that session.

3.4 Aesthetic mappings

We use the `aes` function to do this. You should never refer to variables outside of the dataset (e.g., with diamonds$carat). The functions of variables can be used, `e.g. aes(x=weight, y=height, colour=sqrt(age))`. The default aesthetic mappings can be set when the plot is initialised or modified later using +, as in this example:

```
library(ggplot2)
# aesthetic mapping initially or later
p <- ggplot(mecars)
p <- p + aes(wt, hp)
p <- ggplot(mtcars, aes(x = mpg, y = wt))  # initialised
p + geom_point()
```

The default mappings in the plot p can be extended or overridden in the layers, as with the following code. Aesthetic mappings specified in a layer affect only that layer. For that reason, unless you modify the default scales, axis labels and legend titles will be based on the plot defaults.

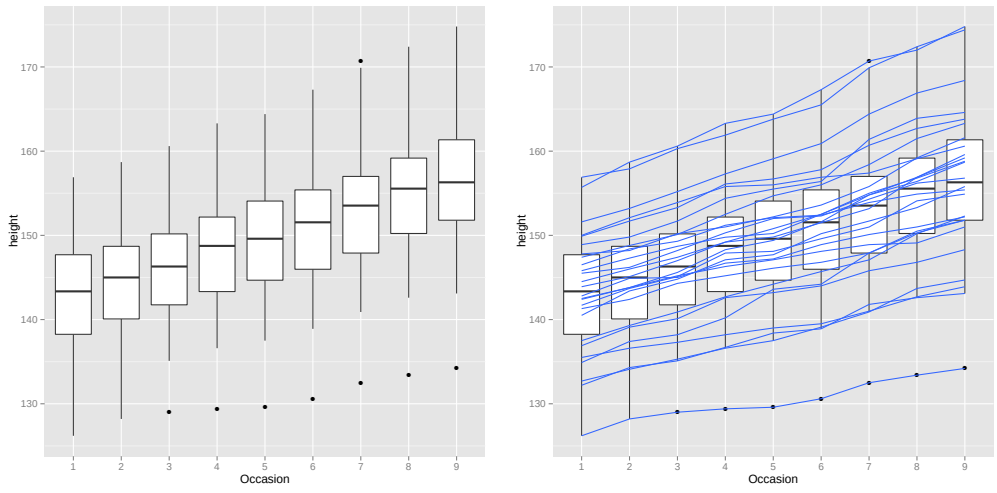
```
# aesthetic mapping for layers only
p + geom_point(aes(colour = factor(cyl)))
p + geom_point(aes(y = disp))
```

```
## Setting VS. Mapping!
p <- ggplot(mtcars, aes(mpg, wt))
p + geom_point(colour = "darkblue")  #setting
p + geom_point(aes(colour = "darkblue"))
# mapping, creat a new variable containing the value 'darkblue'.
qplot(mpg, wt, data = mtcars, colour = I("darkblue"))  #use I() in qplot
```

3.5 Grouping

```
## Multiple groups, one aesthetic
library(nlme)
p <- ggplot(Oxboys, aes(age, height, group = Subject)) + geom_line()
## Different groups on different layers.
p + geom_smooth(aes(group = Subject), method = "lm", se = F)
# each subject has one smooth line
p + geom_smooth(aes(group = 1), method = "lm", se = F)
# group=1: use all data, one smooth line for all subjects.
```

```
## Overriding the default grouping.
library(ggplot2)
library(nlme)
boysbox <- ggplot(Oxboys, aes(Occasion, height)) + geom_boxplot()
# Occasion is discrete
boysbox
boysbox + geom_line(aes(group = Subject), colour = "#3366FF")
# new group for new layer
```



3.6 Geoms

Geometric objects, or geoms for short, perform the actual rendering of the layer, control the type of plot that you create. For example, using a point geom will create a scatterplot, while using a line geom will create a line plot. Every geom has a default statistic, and every statistic a default geom. For example, the bin statistic defaults to using the bar geom to produce a histogram.

3.7 Stat

A statistical transformation, or stat, transforms the data, typically by summarising it in some manner. For example, a useful stat is the smoother, which calculates the mean of y, conditional on x, subject to some restriction that ensures smoothness. A stat takes a dataset as input and returns a dataset as output, and so a stat can add new variables to the original dataset. It is possible to map aesthetics to these new variables. For example, stat_bin, the statistic used to make histograms, produces the following variables: `count`, the number of observations in each bin ; `density`, the density of observations in each bin (percentage of total / bar width); `x`, the centre of the bin These generated variables can be used instead of the variables present in the original dataset. The names of generated variables must be surrounded with `..` when used. This prevents confusion in case the original dataset includes a variable with the same name as a generated variable.

```
ggplot(diamonds, aes(carat)) + geom_histogram(binwidth = 0.1)  #default use count as y axis
ggplot(diamonds, aes(carat)) + geom_histogram(aes(y = ..density..), binwidth = 0.1)
qplot(carat, ..density.., data = diamonds, geom = "histogram", binwidth = 0.1)  #qplot
```

3.8 Pulling all together

```
## Combining geoms and stats
d <- ggplot(diamonds, aes(carat)) + xlim(0, 3)
d + stat_bin(aes(ymax = ..count..), binwidth = 0.1, geom = "area")
d + stat_bin(aes(size = ..density..), binwidth = 0.1, geom = "point", position = "identity")
```

```
model <- lme(height ~ age, data = Oxboys, random = ~1 + age | Subject)  #linear mixed model
oplot <- ggplot(Oxboys, aes(age, height, group = Subject)) + geom_line()
age_grid <- seq(-1, 1, length = 10)
subjects <- unique(Oxboys$Subject)
preds <- expand.grid(age = age_grid, Subject = subjects)
# creat a data frame from all combinations of factors
Oxboys$fitted <- predict(model)
Oxboys$resid <- with(Oxboys, fitted - height)
oplot %+% Oxboys + aes(y = resid) + geom_smooth(aes(group = 1))  #apply to different data
```

4 Toolbox

4.1 Basic plot types

```
geom_area()
geom_bar(stat = "identity")  #default multiple bars in same location stacked
geom_line()
geom_path()  # in the order
geom_point()  #scatterplot
geom_polygon()
geom_text()  #need another aesthetic: label. hjust, vjust, angle
geom_tile()
# examples
df <- data.frame(x = c(3, 1, 5), y = c(2, 4, 6))
p <- ggplot(df, aes(x, y)) + xlab(NULL) + ylab(NULL)
p + geom_point() + ggtitle("geom_point")
```

```
p + geom_bar(stat = "identity") + opts(title = "geom_bar(stat=\"identity\")")
p + geom_text(label = c("a", "b", "c")) + ggtitle("geom_text")
```

4.2 Displaying distributions

```
depthdist <- ggplot(diamonds, aes(depth)) + xlim(58, 68)
depthdist + geom_histogram(aes(y = ..density..), binwidth = 0.1) + facet_grid(cut ~
    .)
depthdist + geom_freqpoly(aes(y = ..density.., colour = cut), binwidth = 0.1)
```

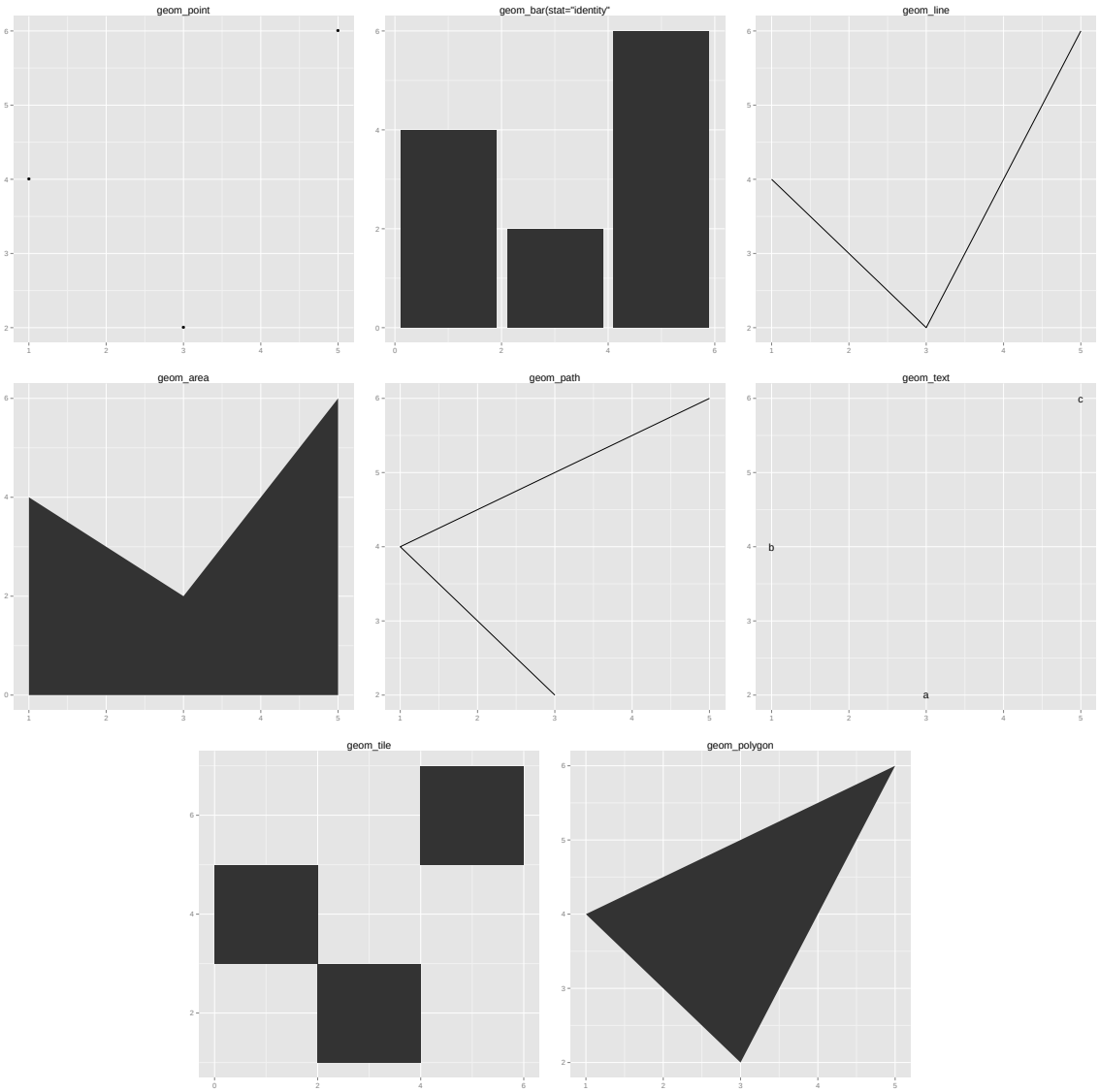
4.3 Drawing maps

```
#maps
library(maps)
data(us.cities)
bigcities=subset(us.cities, pop>500000)
p=qplot(long,lat,data=bigcities) p+borders("state",size=.5)
wicities=subset(us.cities,country.etc=="WI")
ggplot(wicities,aes(long,lat))+borders("county","wisconsin",colour="grey70")+geom_point()
wi=map_data("county","wisconsin")
mid_range=function(x) mean(range(x,na.rm=TRUE))
library(plyr)
centres=ddply(wi, .(subregion),colwise(mid_range,.(lat,long)))
ggplot(wi, aes(long, lat)) + geom_polygon(aes(group = group),fill = NA, colour = "grey60") + geom_text(a
```

5 Toolbox

5.1 Basic plot types

```
df <- data.frame(x = c(3, 1, 5), y = c(2, 4, 6), label = c("a", "b", "c"))
p <- ggplot(df, aes(x, y, label = label)) + xlab(NULL) + ylab(NULL)
p + geom_point() + ggtitle("geom_point")
p + geom_bar(stat = "identity") + ggtitle(("geom_bar(stat=\"identity\""))
p + geom_line() + ggtitle("geom_line")
p + geom_area() + ggtitle("geom_area")
p + geom_path() + ggtitle("geom_path")
p + geom_text() + ggtitle("geom_text")
p + geom_tile() + ggtitle("geom_tile")
p + geom_polygon() + ggtitle("geom_polygon")
```




```
# xlim(low,high)
depth_dist <- ggplot(diamonds, aes(depth)) + xlim(58, 68)
depth_dist + geom_histogram(aes(y = ..density..), binwidth = 0.1) + facet_grid(cut ~
    .)
depth_dist + geom_histogram(aes(fill = cut), binwidth = 0.1, position = "fill")
depth_dist + geom_freqpoly(aes(y = ..density.., colour = cut), binwidth = 0.1)
qplot(cut, depth, data = diamonds, geom = "boxplot")
qplot(carat, depth, data = diamonds, geom = "boxplot", group = round_any(carat,
    0.1, floor), xlim = c(0, 3))
```

5.2 Dealing with overplotting

```
# shape or fuzzing
df <- data.frame(x = rnorm(2000), y = rnorm(2000))
norm <- ggplot(df, aes(x, y))
norm + geom_point()
```

```
norm + geom_point(shape = 1)  # hollow points
norm + geom_point(shape = ".")  # Pixel sized
norm + geom_point(colour = "black", alpha = 1/5)
```

```
# Jitter
td <- ggplot(diamonds, aes(table, depth)) + xlim(50, 70) + ylim(50, 70)
td + geom_point()
td + geom_jitter()
jit <- position_jitter(width = 0.5)
td + geom_jitter(position = jit)
td + geom_jitter(position = jit, colour = "black", alpha = 1/5)
```

6 Scales, axes and legends

6.1 Scales: common arguments

```
p <- qplot(cty, hwy, data = mpg, colour = displ)
p + scale_x_continuous("City mpg")
p + xlab("City mpg")
p + ylab("Highway mpg")
p + labs(x = "City mpg", y = "Highway", colour = "Displacement")
p + xlab(expression(frac(miles, gallon)))
```

Breaks controls which values appear on the axis or legend, i.e., what values tick marks should appear on an axis or how a continuous scale is segmented in a legend. labels specifies the labels that should appear at the breakpoints. If labels is set, you must also specify breaks, so that the two can be matched up correctly. To distinguish breaks from limits, remember that breaks affect what appears on the axes and legends, while limits affect what appears on the plot.

```
p <- qplot(cyl, wt, data = mtcars)
p + scale_x_continuous(breaks = c(5.5, 6.5))
p + scale_x_continuous(limits = c(5.5, 6.5))
p <- qplot(wt, cyl, data = mtcars, colour = cyl)
p + scale_colour_gradient(breaks = c(5.5, 6.5))
p + scale_colour_gradient(limits = c(5.5, 6.5))
```

6.2 Facet

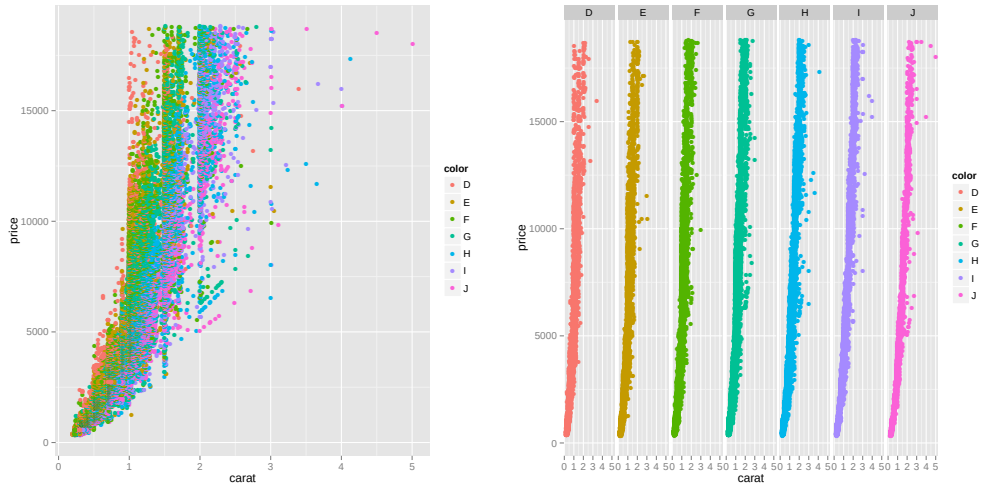
There are two types of faceting provided by ggplot2: `facet_grid` and `facet_wrap`. Facet grid produces a 2d grid of panels defined by variables which form the rows and columns, while facet wrap produces a 1d ribbon of panels that is wrapped into 2d.

You can access either faceting system from qplot(). A 2d faceting specification (e.g., x ~ y) will use facet_grid, while a 1d specification (e.g., ~ x) will use facet_wrap. For facet_grid there is an additional parameter called space, which takes values "free" or "fixed". When the space can vary freely, each column (or row) will have width (or height) proportional to the range of the scale for that column (or row). This makes the scaling equal across the whole plot. `facet_grid(manufacturer ~ ., scales = "free", space = "free")`

```
mpg2 <- subset(mpg, cyl != 5 & drv %in% c("4", "f"))
qplot(cty, hwy, data = mpg2) + facet_grid(. ~ cyl)  #a single row with multiple column
qplot(cty, data = mpg2, geom = "histogram") + facet_grid(cyl ~ .)  #a single col with multiple row
# facet_grid(variable ~ ., scales = 'free_y')
qplot(cty, hwy, data = mpg2) + facet_grid(drv ~ cyl)  #multiple rows and cols
p <- qplot(displ, hwy, data = mpg2) + geom_smooth(method = "lm", se = F)
p + facet_grid(cyl ~ drv)
p + facet_grid(cyl ~ drv, margins = T)  #Margin like contingency table
qplot(displ, hwy, data = mpg2) + geom_smooth(aes(colour = drv), method = "lm",
    se = F) + facet_grid(cyl ~ drv, margins = T)
```

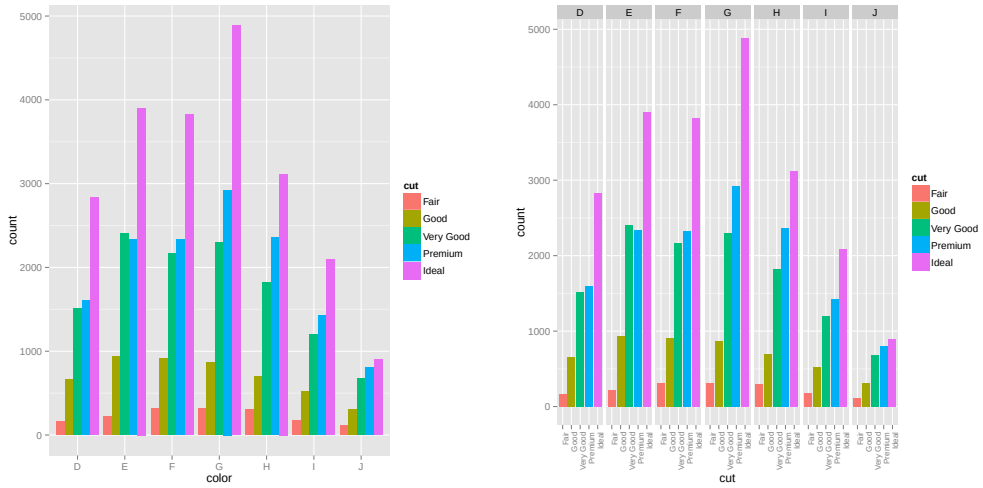
6.2.1 Group VS facet

```
dplot <- ggplot(diamonds, aes(carat, price, colour = color))
dplot + geom_point()
dplot + geom_point() + facet_grid(. ~ color)
```



6.2.2 Dodging vs. faceting

```
qplot(color, data = diamonds, geom = "bar", fill = cut, position = "dodge")
qplot(cut, data = diamonds, geom = "bar", fill = cut) + facet_grid(. ~ color) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 8, colour = "grey50"))
```



6.3 Legend

```
## change default point color for discrete variables.
cbPalette <- c("blue", "red")
p + geom_point(data = data, aes(long, lat, colour = type, shape = type)) + theme_bw()
+theme(legend.position = "top") + scale_colour_manual(values = cbPalette)
```

6.4 Theme

```
ggplot() + theme_bw()
```

6.5 plyr package

```
library(plyr)
head(ddply(diamonds, .(color), subset, carat == min(carat)))  #subset to select data
```

```
##   carat     cut color clarity depth table price    x    y    z
## 1   0.2   Ideal     D     VS2  61.5    57   367 3.81 3.77 2.33
## 2   0.2 Premium     D     VS2  62.3    60   367 3.73 3.68 2.31
## 3   0.2 Premium     D     VS2  61.7    60   367 3.77 3.72 2.31
## 4   0.2 Premium     E     SI2  60.2    62   345 3.79 3.75 2.27
## 5   0.2 Premium     E     VS2  59.8    62   367 3.79 3.77 2.26
## 6   0.2 Premium     E     VS2  59.0    60   367 3.81 3.78 2.24
```

```
head(ddply(diamonds, .(color), transform, price = scale(price)))  #transform to change
```

```
##   carat       cut color clarity depth table   price    x    y    z
## 1  0.23 Very Good     D     VS2  60.5    61 -0.8380 3.96 3.97 2.40
## 2  0.23 Very Good     D     VS1  61.9    58 -0.8246 3.92 3.96 2.44
## 3  0.26 Very Good     D     VS2  60.8    59 -0.8243 4.13 4.16 2.52
## 4  0.26      Good     D     VS2  65.2    56 -0.8243 3.99 4.02 2.61
## 5  0.26      Good     D     VS1  58.4    63 -0.8243 4.19 4.24 2.46
## 6  0.22   Premium     D     VS2  59.3    62 -0.8240 3.91 3.88 2.31
```

```
head(ddply(diamonds, .(color), transform, price = price - mean(price)))
```

```
##   carat       cut color clarity depth table price    x    y    z
## 1  0.23 Very Good     D     VS2  60.5    61 -2813 3.96 3.97 2.40
## 2  0.23 Very Good     D     VS1  61.9    58 -2768 3.92 3.96 2.44
## 3  0.26 Very Good     D     VS2  60.8    59 -2767 4.13 4.16 2.52
## 4  0.26      Good     D     VS2  65.2    56 -2767 3.99 4.02 2.61
## 5  0.26      Good     D     VS1  58.4    63 -2767 4.19 4.24 2.46
## 6  0.22   Premium     D     VS2  59.3    62 -2766 3.91 3.88 2.31
```

```
# colwise() returns a new function
nmissing <- function(x) sum(is.na(x))
nmissing(msleep$brainwt)
```

```
## [1] 27
```

```
nmissing_df <- colwise(nmissing)
nmissing_df(msleep)
```

```
##   name genus vore order conservation sleep_total sleep_rem sleep_cycle
## 1    0     0    7     0           29           0        22          51
##   awake brainwt bodywt
## 1     0      27      0
```

```
colwise(nmissing)(msleep)  # a shortcut
```

```
##   name genus vore order conservation sleep_total sleep_rem sleep_cycle
## 1    0     0    7     0           29           0        22          51
##   awake brainwt bodywt
## 1     0      27      0
```

```
# numcolwise(). catcolwise() only works with categorical columns
ddply(msleep, .(vore), numcolwise(median), na.rm = T)
```

```
##      vore sleep_total sleep_rem sleep_cycle awake brainwt bodywt
## 1   carni        10.4      1.95      0.3833  13.6 0.04450 20.490
## 2   herbi        10.3      0.95      0.2167  13.7 0.01229  1.225
## 3 insecti        18.1      3.00      0.1667   5.9 0.00120  0.075
## 4    omni         9.9      1.85      0.5000  14.1 0.00660  0.950
## 5    <NA>        10.6      2.00      0.1833  13.4 0.00300  0.122
```

```
my_summary <- function(df) {
    with(df, data.frame(pc_cor = cor(price, carat, method = "spearman"), lpc_cor = cor(log(price),
        log(carat))))
}
ddply(diamonds, .(cut), my_summary)
```

```
##         cut pc_cor lpc_cor
## 1      Fair 0.9056  0.9085
## 2      Good 0.9600  0.9688
## 3 Very Good 0.9689  0.9717
## 4   Premium 0.9605  0.9698
## 5     Ideal 0.9537  0.9662
```