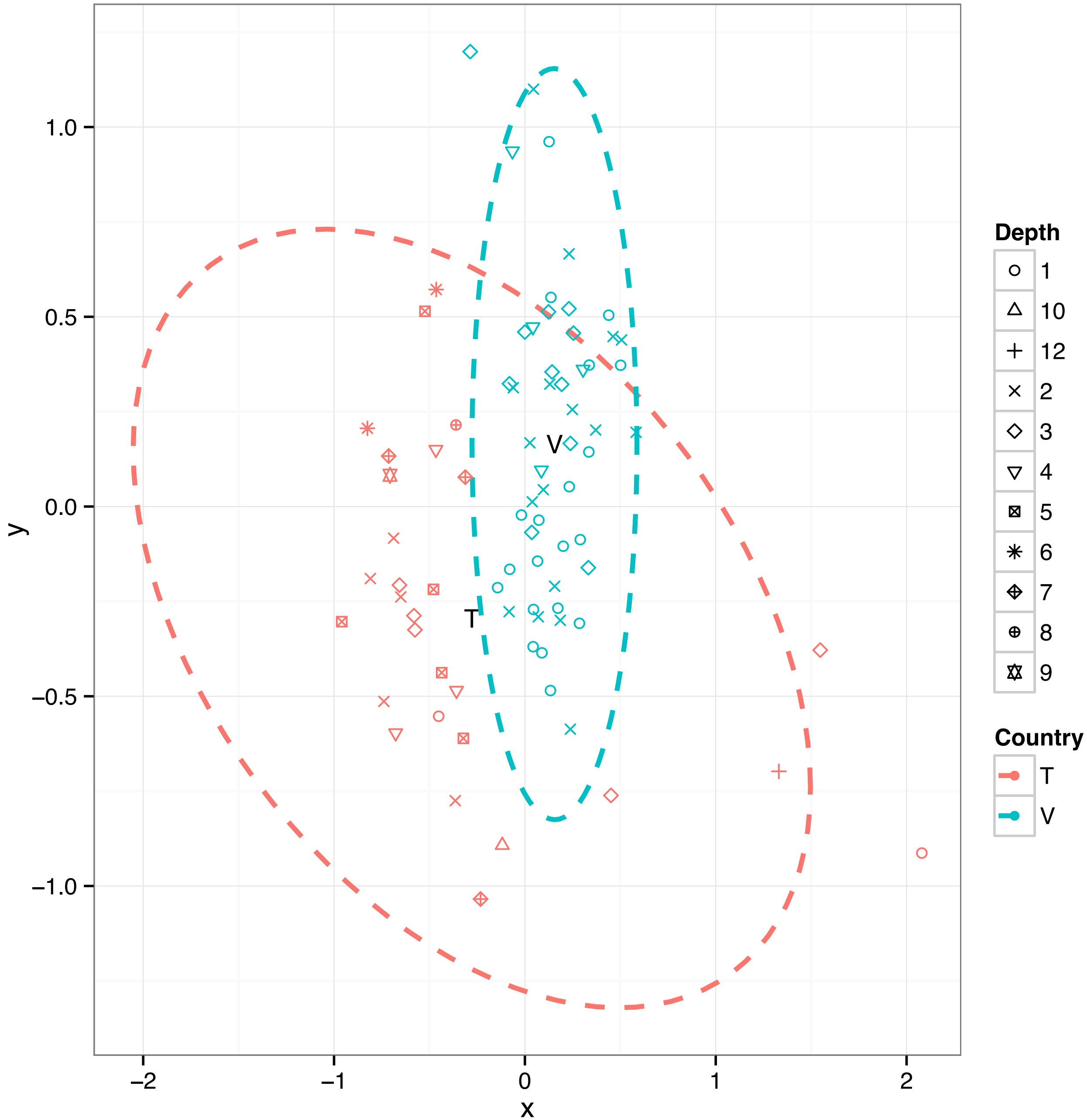
ggplot2: Introduction and exercises

Umer Zeeshan Ijaz

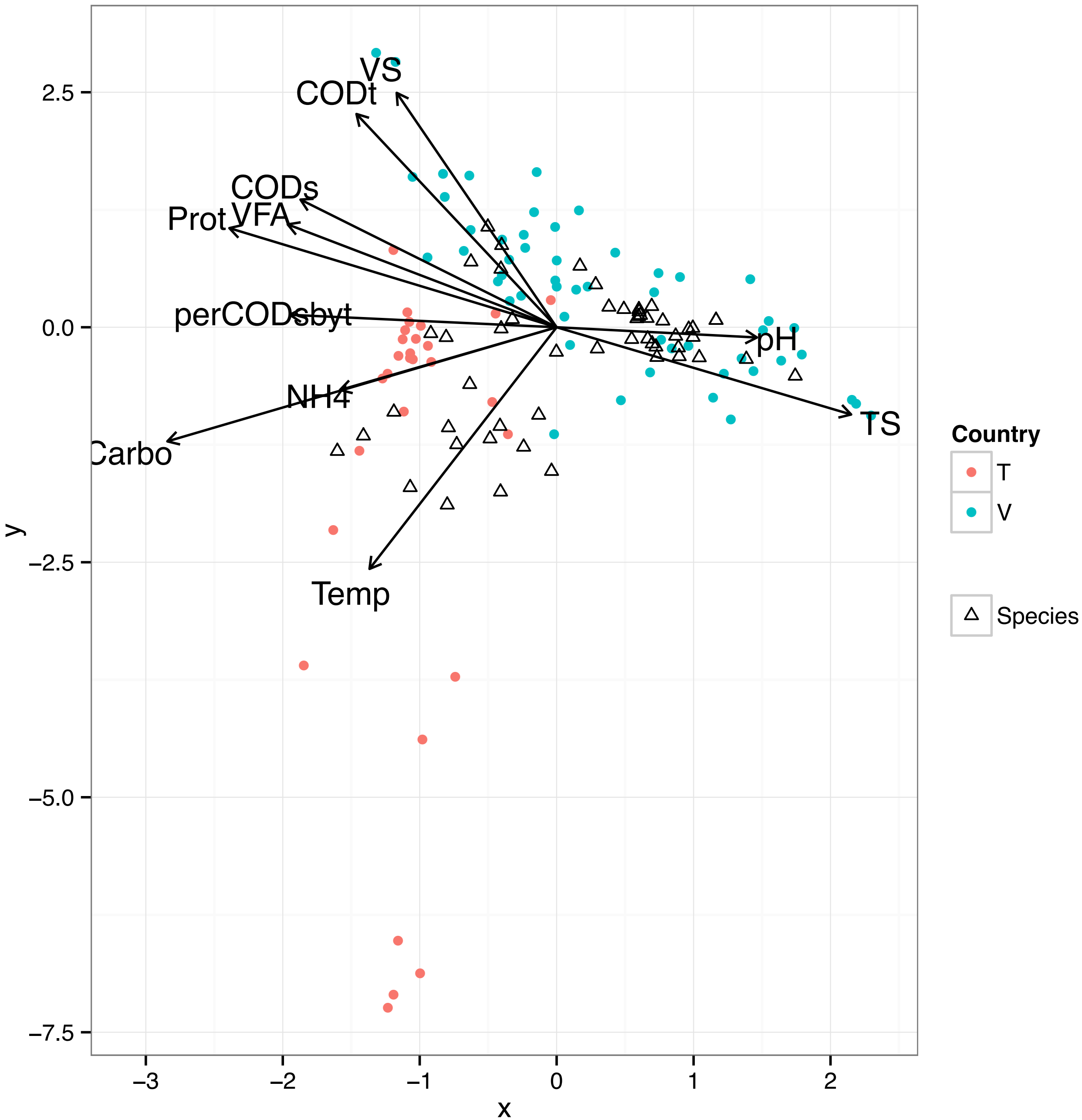
http://userweb.eng.gla.ac.uk/umer.ijaz

Motivation

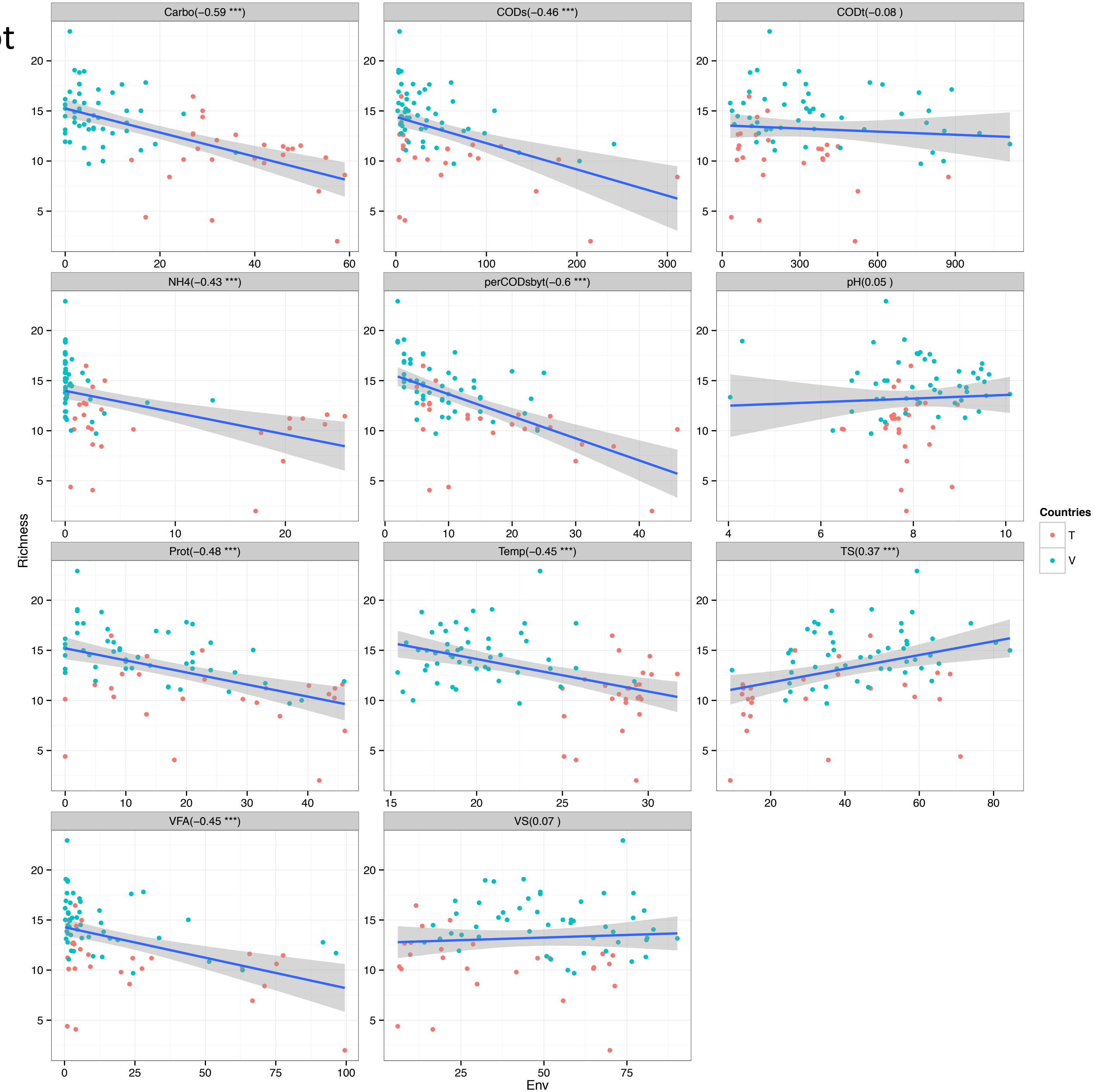
NMDS plot
(NMDS.R)



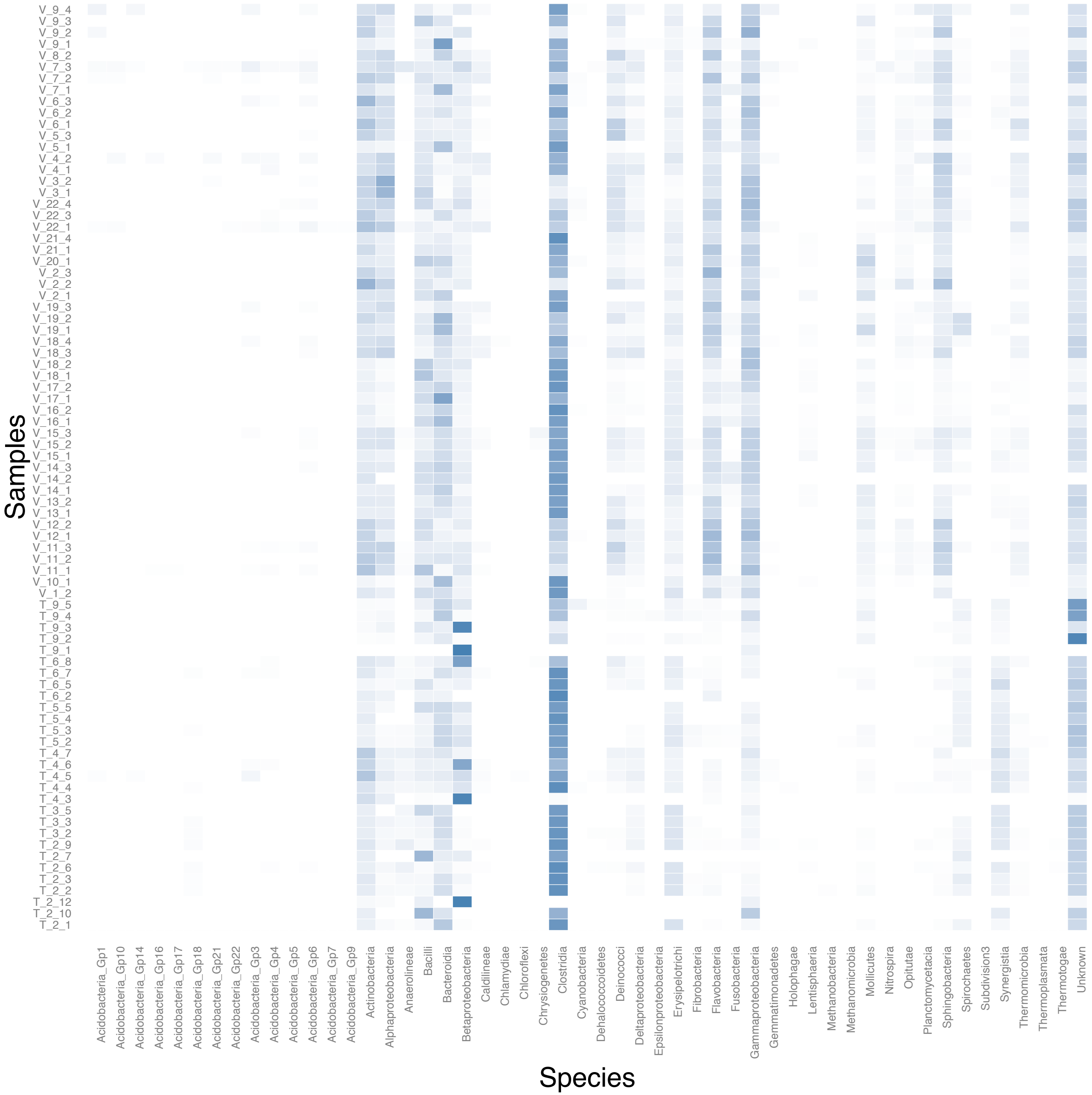
CCA plot
(CCA.R)



Richness plot (richness.R)



Heatmap
(heatmap.R)



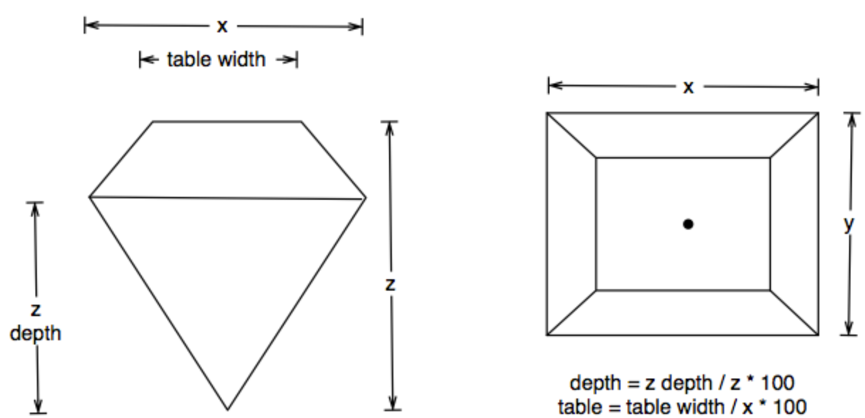
ggplot2 basics
ggplot_basics.R

ggplot2

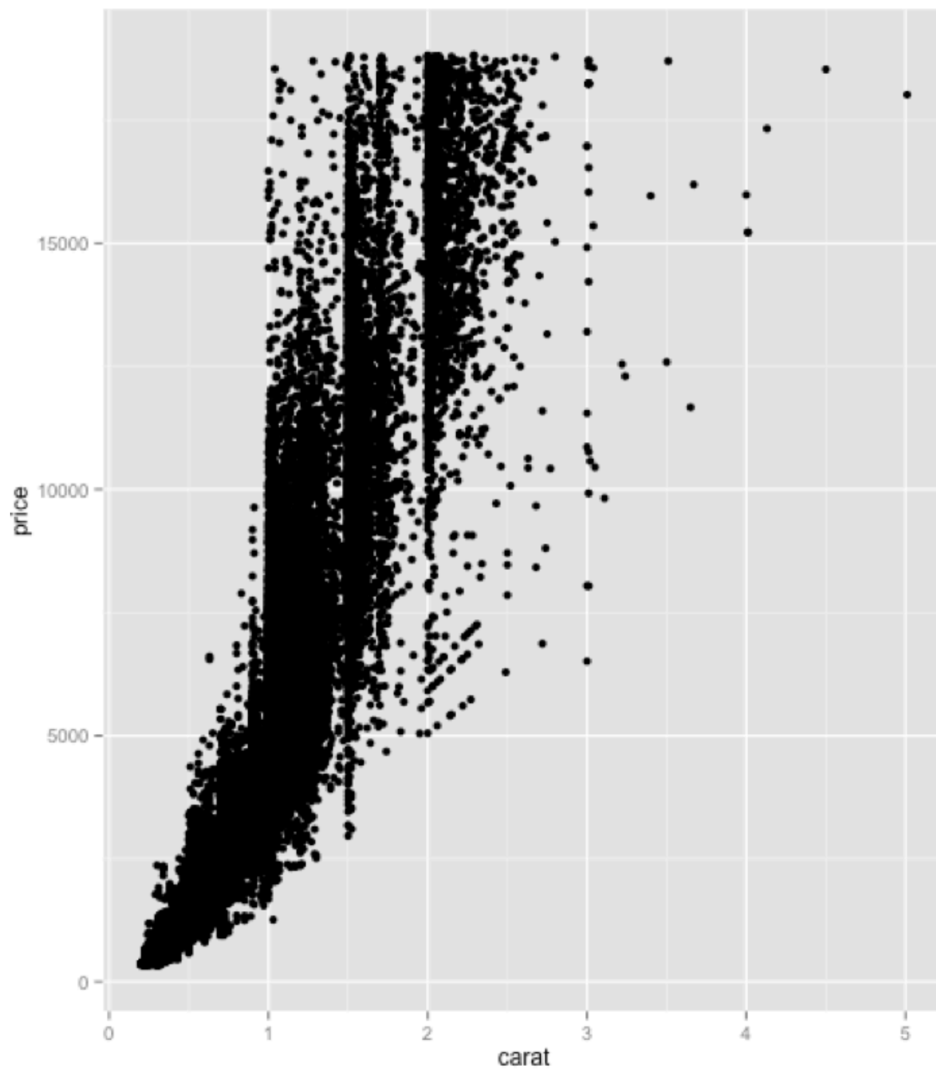
- Use just qplot(), without any understanding of the underlying grammar
- Theoretical basis of ggplot2: layered grammar is based on Wilkinson's grammar of graphics (Wilkinson, 2005)

Data

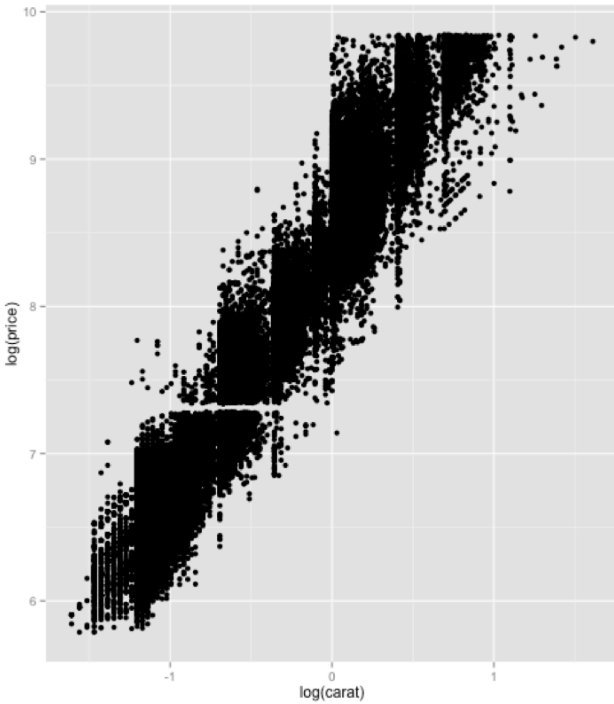
```
head(diamonds)
A dataset containing the prices and other attributes of almost 54,000 diamonds.
The variables are as follows:
# price   = price in US dollars ($326–$18,823)
# carat   = weight of the diamond (0.2–5.01)
# cut     = quality of the cut (Fair, Good, Very Good, Premium, Ideal)
# colour  = diamond colour, from J (worst) to D (best)
# clarity = a measurement of how clear the diamond is (I1 (worst), SI1, SI2, VS1,
#            VS2, VVS1, VVS2, IF (best))
# x       = length in mm (0–10.74)
# y       = width in mm (0–58.9)
# z       = depth in mm (0–31.8)
# depth   = total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
# table   = width of top of diamond relative to widest point (43–95)
```

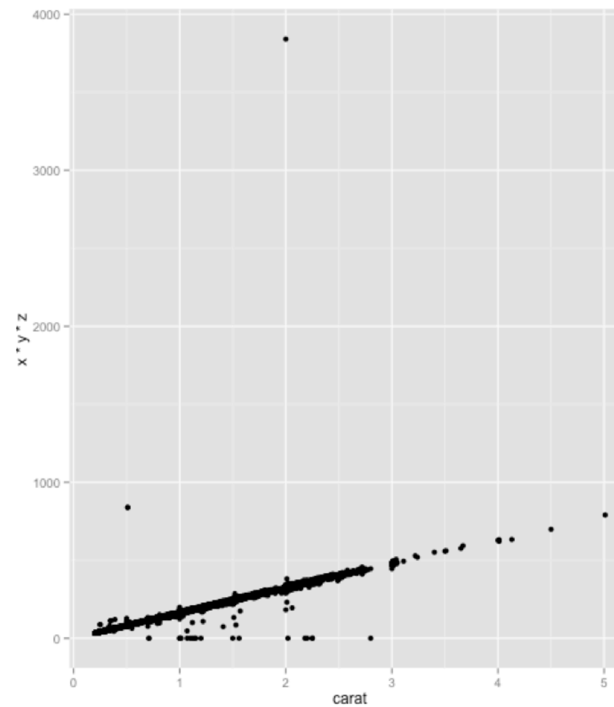


```
qplot(carat, price, data = diamonds)
```



```
qplot(log(carat), log(price), data = diamonds)
```

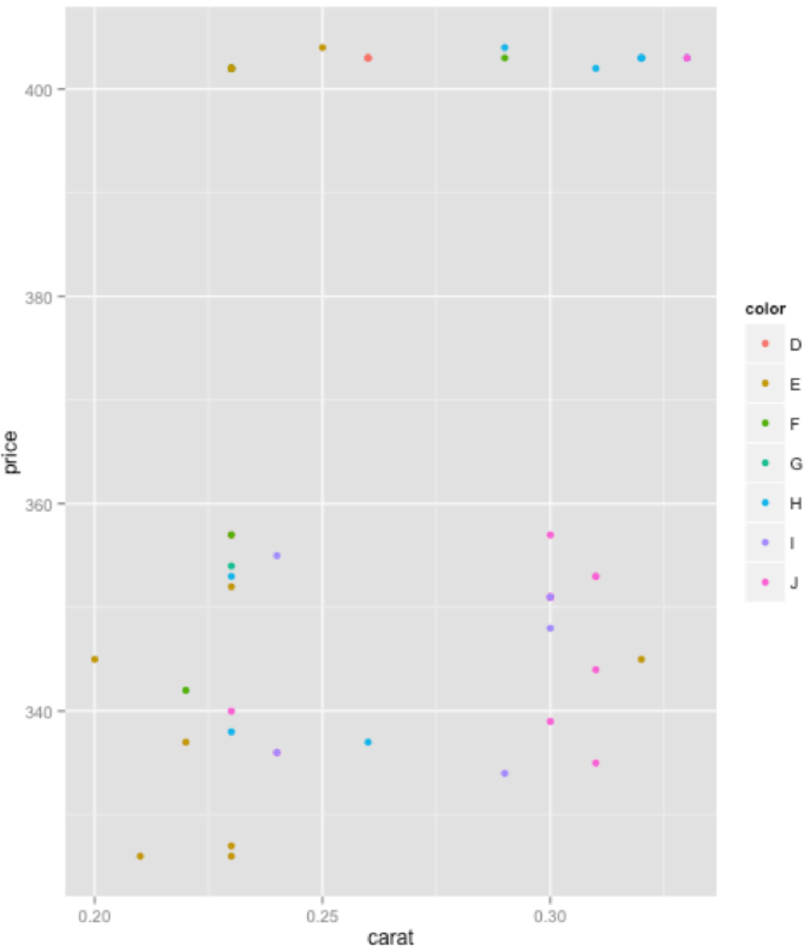


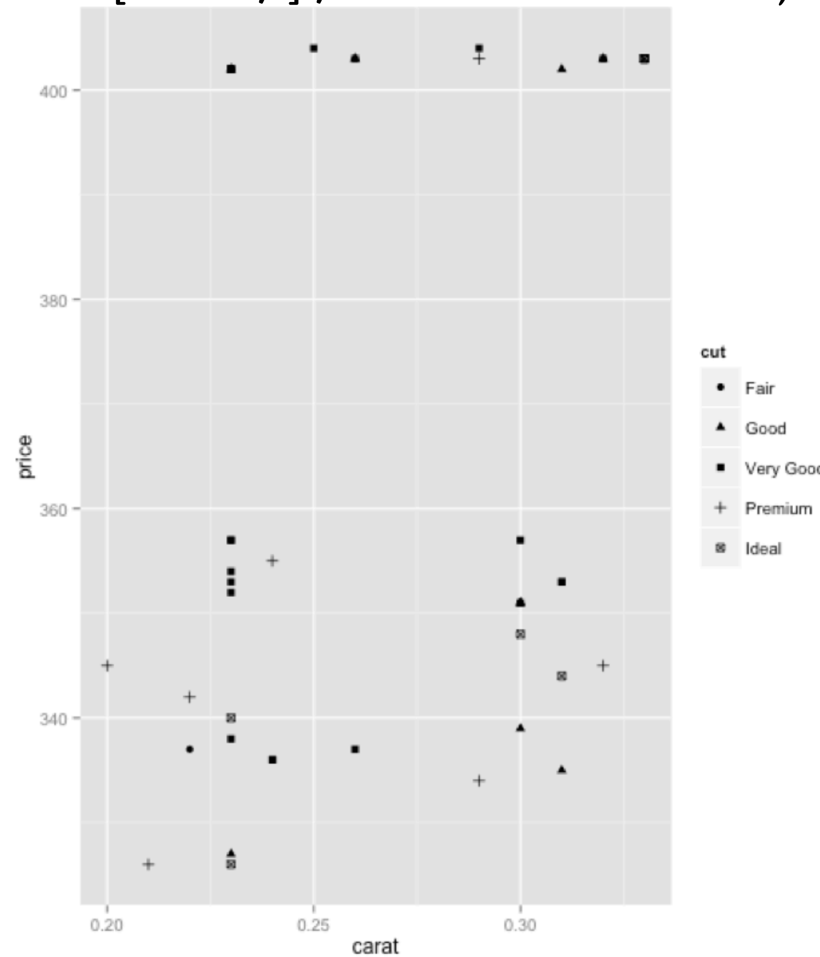



```
qplot(carat, x * y * z, data = diamonds)
```

Aesthetic Attributes

```
qplot(carat, price, data = diamonds[1:50,], colour = color)
```



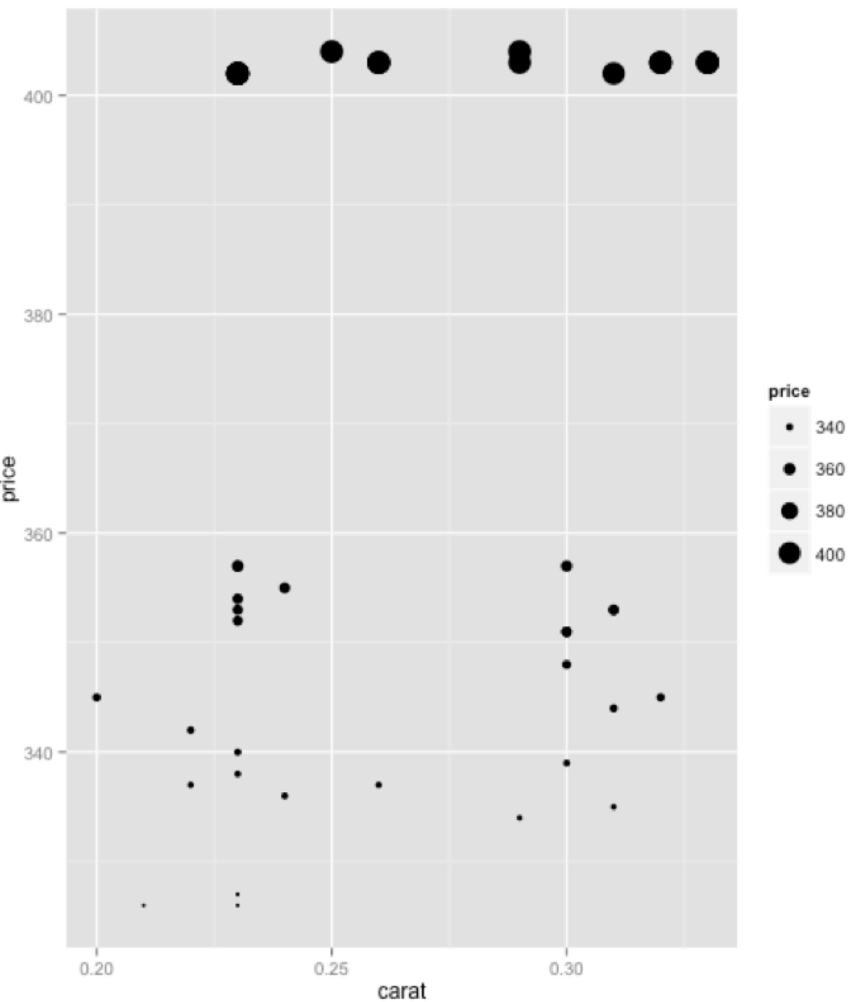


http://www.handprint.com/HP/WCL/color7.html

```
qplot(carat, price, data = diamonds[1:50,], shape = cut)
```

Aesthetic Attributes (3)

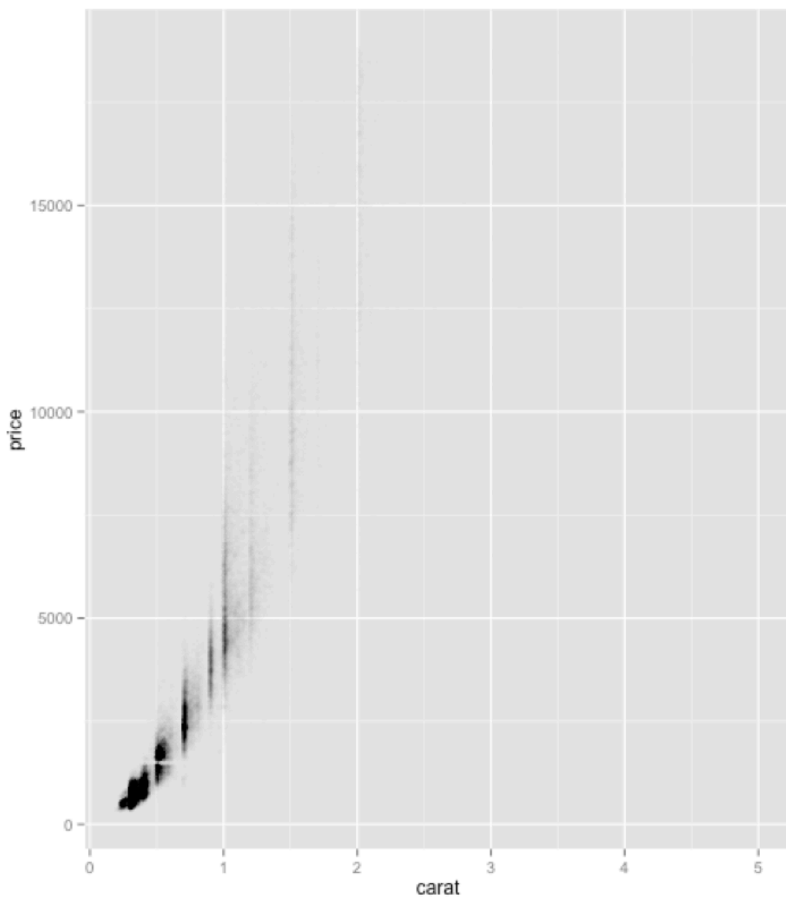
```
qplot(carat, price, data = diamonds[1:50,], size = price)
```



Aesthetic Attributes (3)

- **colour**, **size** and **shape** are all examples of aesthetic attributes, visual properties that affect the way observations are displayed.
- For every aesthetic attribute, there is a function, called a **scale**, which maps data values to valid values for that aesthetic.

```
library(scales)
qplot(carat, price, data = diamonds, colour = I(alpha("black", 1/200)))
```



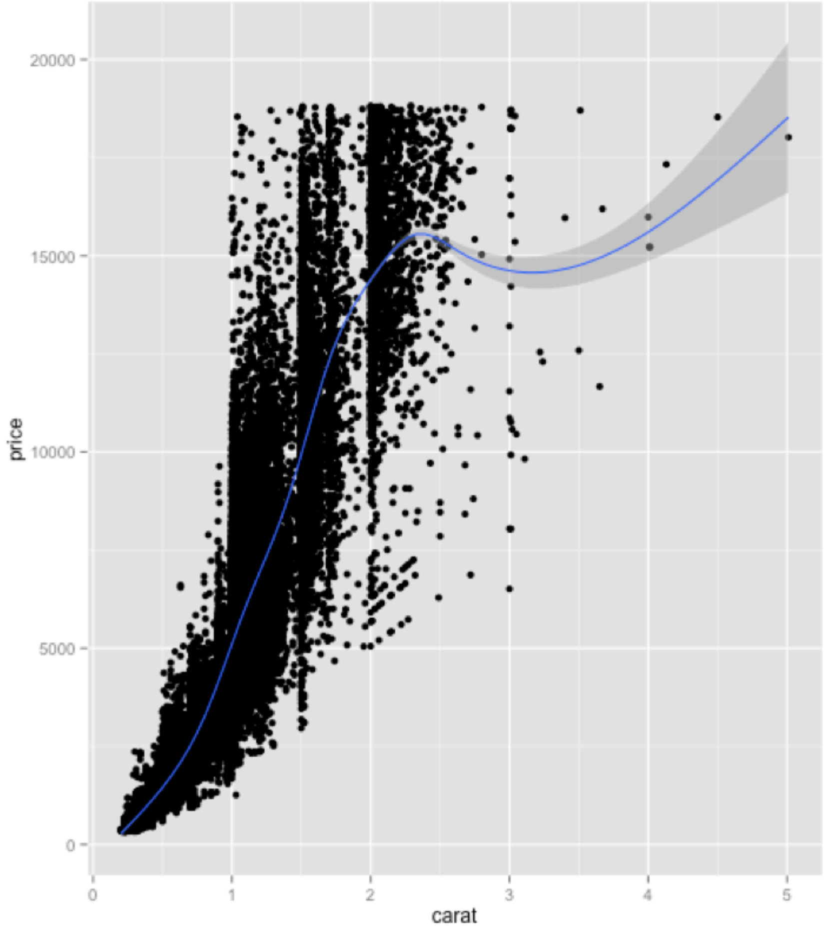
You can also manually set the aesthetics using I()

Plot geoms(1)

- **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.
- For continuous variables, **geom = "histogram"** draws a histogram, **geom = "freqpoly"** a frequency polygon, and **geom = "density"** creates a density plot
- For discrete variables, **geom = "bar"** makes a barchart

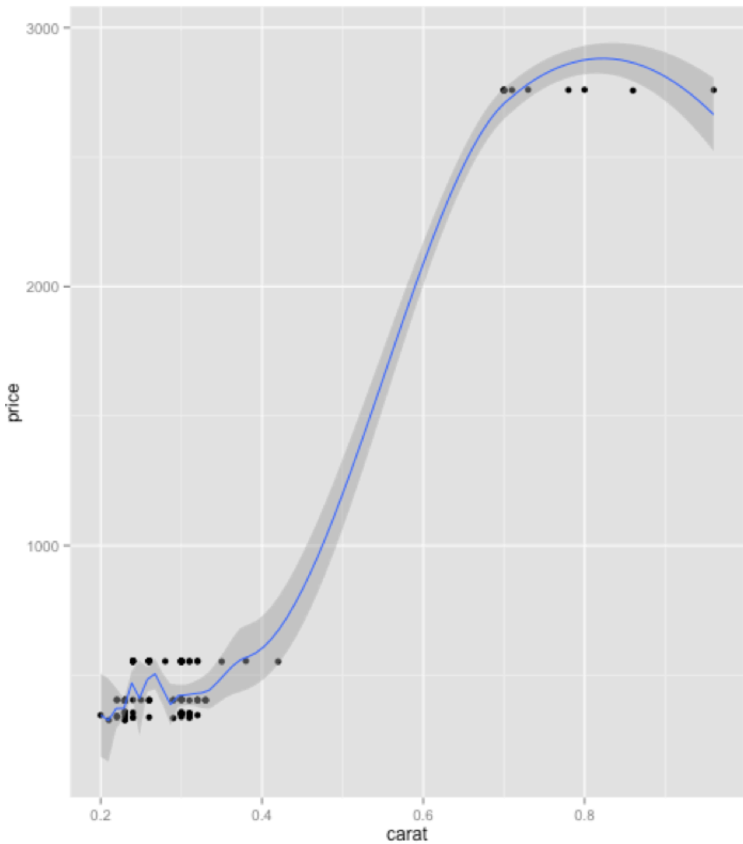
Plot geoms(2)

```
qplot(carat, price, data = diamonds, geom = c("point", "smooth"))
```



Plot geoms(3)

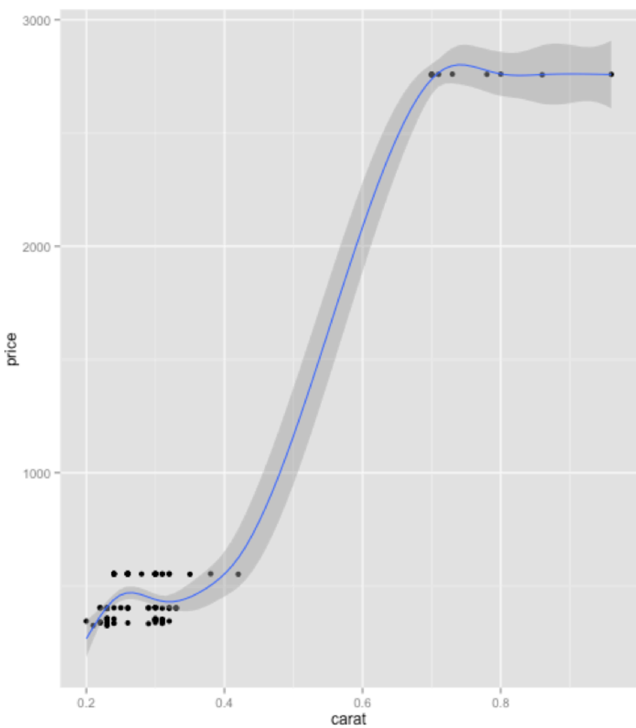
```
qplot(carat, price, data = diamonds[1:100,], geom = c("point",
"smooth"),span=0.2,se=TRUE)
```

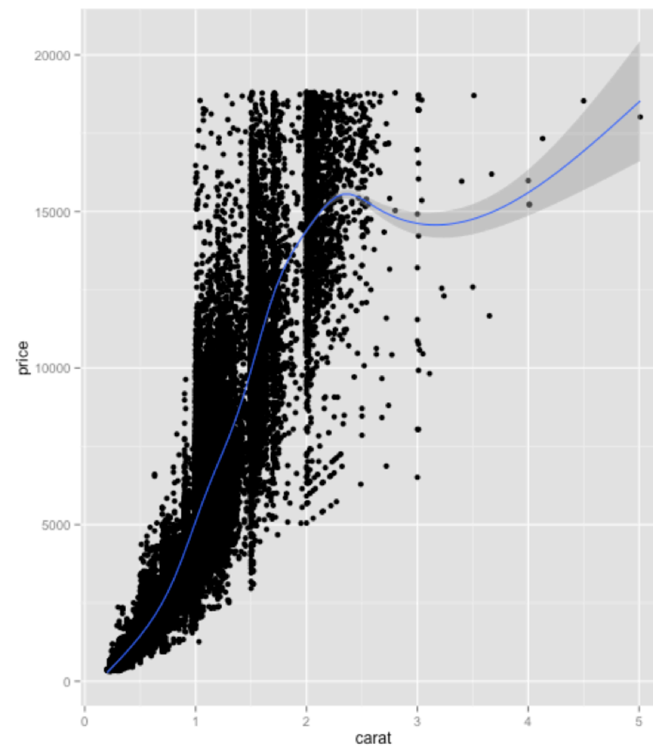


- **method = "loess"**, the default for small n, uses a smooth local regression.
- If you want to turn the confidence interval off, use **se = FALSE**
- The wiggliness of the line is controlled by the **span** parameter, which ranges from 0 (exceeding wiggly) to 1 (not so wiggly)

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.

```
library(mgcv)
qplot(carat, price, data = diamonds[1:100,], geom = c("point",
"smooth"),method="gam", formula= y ~ s(x))
qplot(carat, price, data = diamonds, geom = c("point",
"smooth"),method="gam", formula= y ~ s(x,bs="cs"))
```





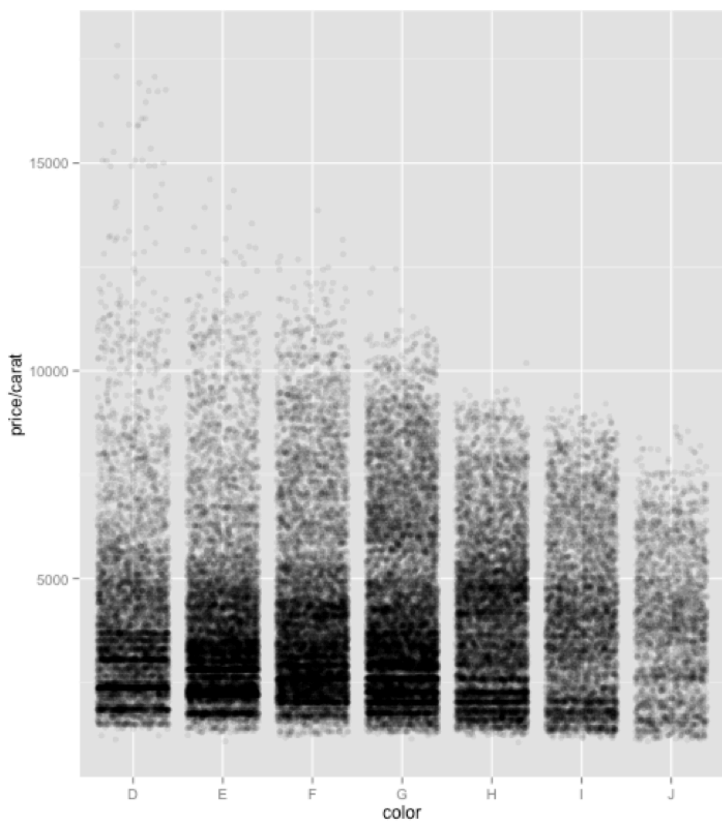
Plog geom(5)

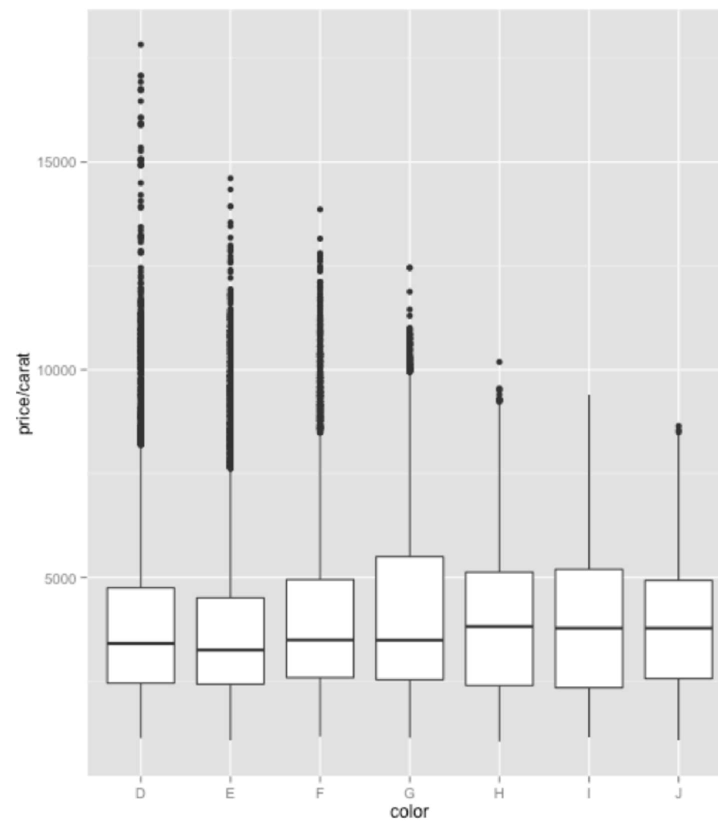
- **method = "lm"** fits a linear model. The default will fit a straight line to your data, or you can specify formula = **y ~ poly(x, 2)** to specify a degree 2 polynomial, or better, load the splines package and use a natural spline: formula = **y ~ ns(x, 2)**. The second parameter is the degrees of freedom: a higher number will create a wigglier curve.
- **method = "rlm"** works like lm, but uses a robust fitting algorithm so that outliers don't affect the fit as much

```
library(splines)
qplot(carat, price, data = diamonds[1:100,], geom=c("point", "smooth"),
method = "lm")
qplot(carat, price, data = diamonds[1:100,], geom=c("point", "smooth"),
method = "lm", formula=y ~ poly(x,2))
qplot(carat, price, data = diamonds[1:100,], geom=c("point", "smooth"),
method = "lm", formula=y ~ ns(x,3))
library(MASS)
qplot(carat, price, data = diamonds[1:100,], geom=c("point", "smooth"),
method = "rlm")
```

Boxplots and jittered points

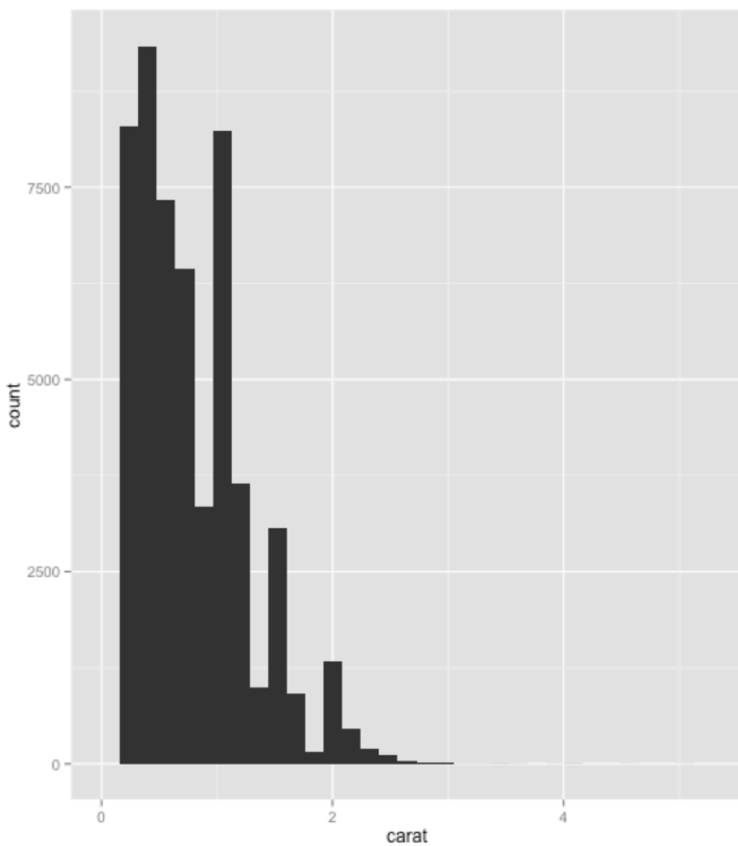
```
qplot(color, price / carat, data = diamonds, geom = "jitter", colour =
I(alpha("black", 1 / 10)))
qplot(color, price / carat, data = diamonds, geom = "boxplot")
```

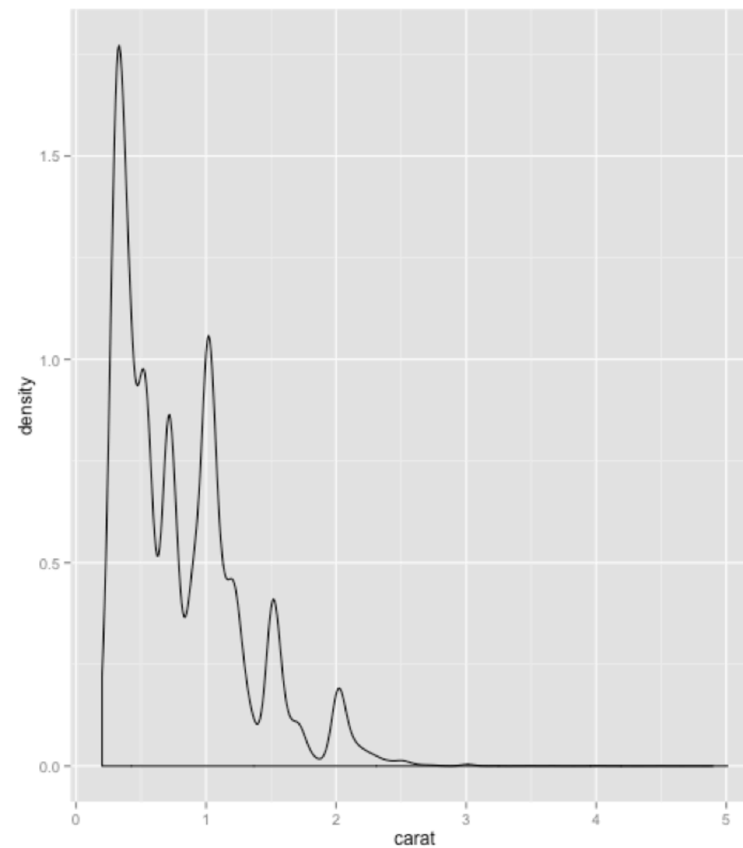




Histogram and density plots(1)

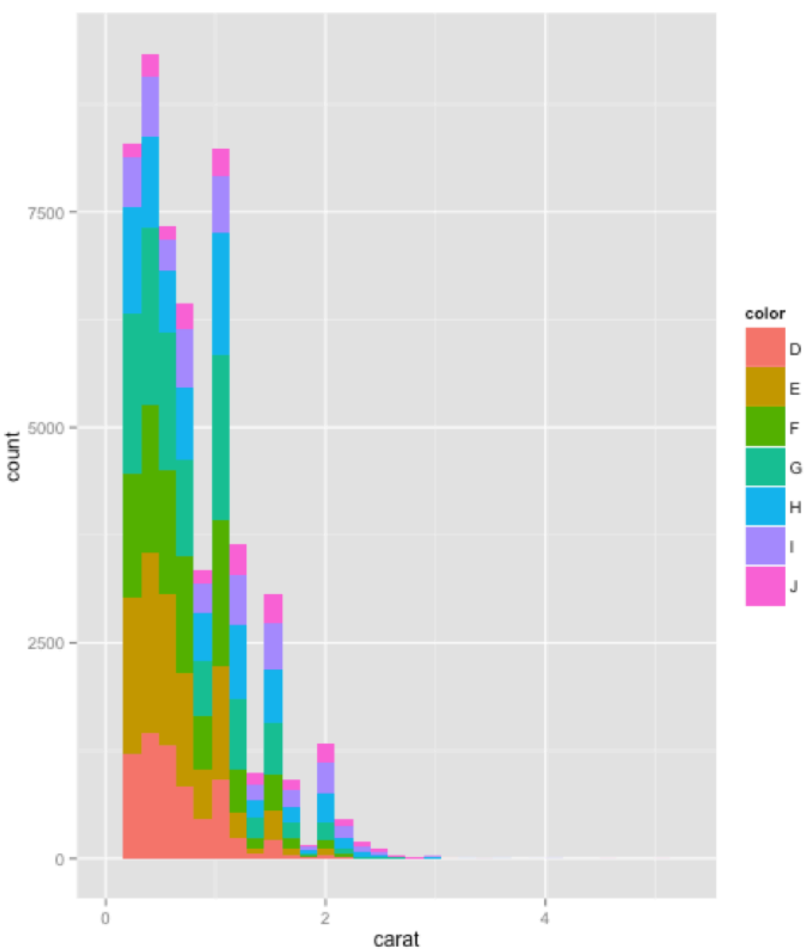
```
qplot(carat, data = diamonds, geom = "histogram")
qplot(carat, data = diamonds, geom = "density")
```

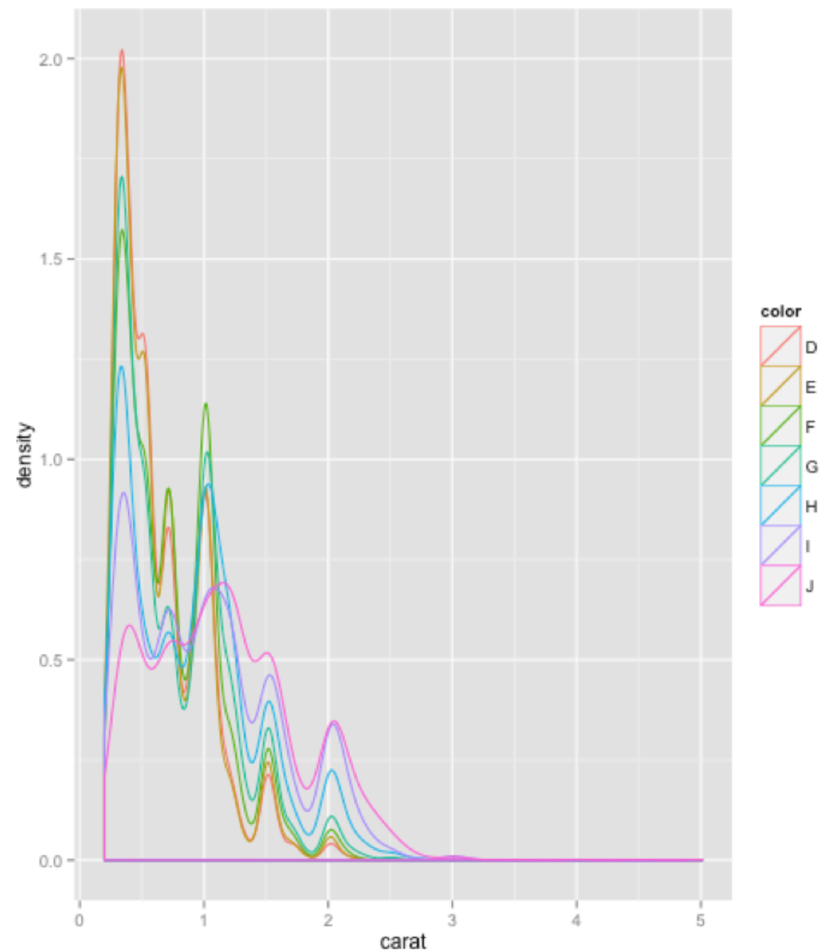




Histogram and density plots(2)

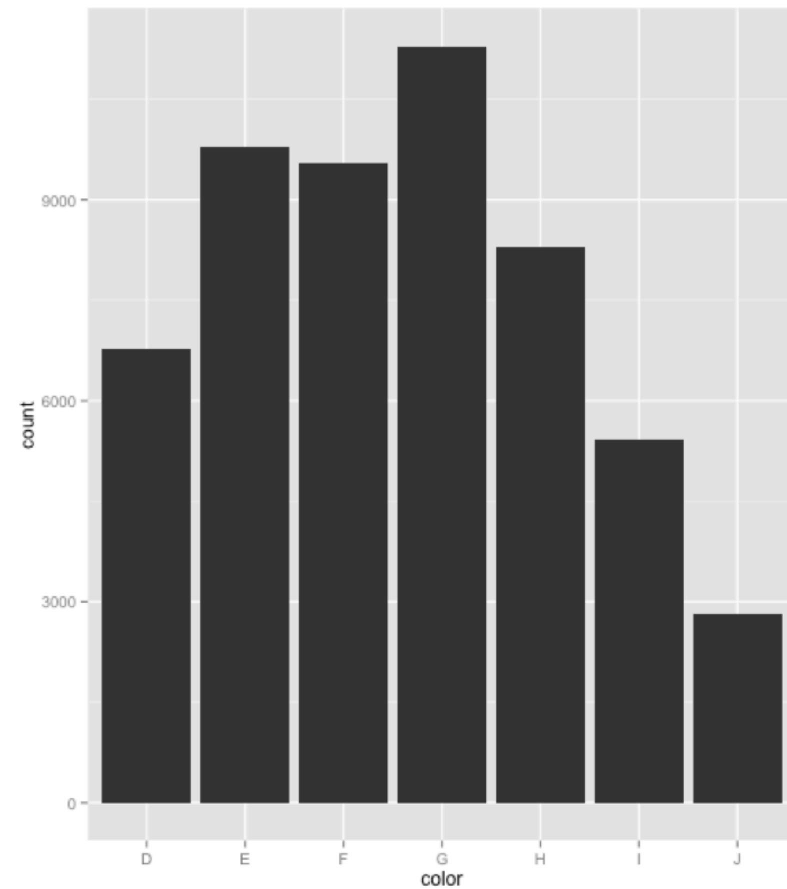
```
qplot(carat, data = diamonds, geom = "histogram", fill = color)
qplot(carat, data = diamonds, geom = "density", colour = color)
```





Bar charts(1)

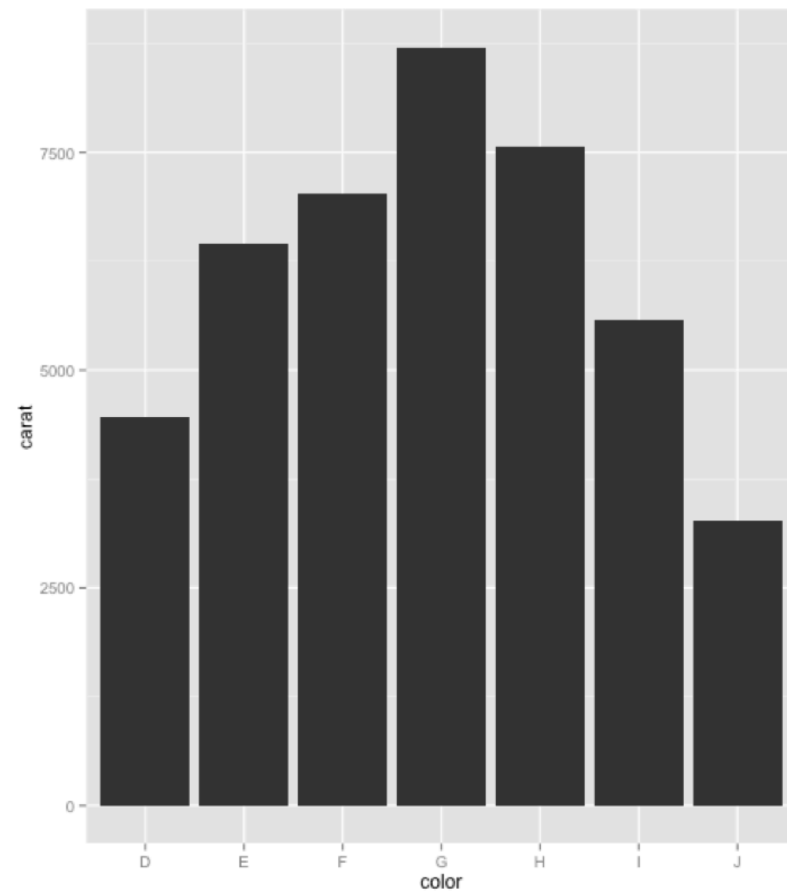
```
qplot(color, data=diamonds, geom="bar")

> dim(diamonds[diamonds$color=="D",])
[1] 6775   10
> dim(diamonds[diamonds$color=="E",])
[1] 9797   10
> dim(diamonds[diamonds$color=="E",])
[1] 9797   10
> dim(diamonds[diamonds$color=="F",])
[1] 9542   10
> dim(diamonds[diamonds$color=="G",])
[1] 11292    10
> dim(diamonds[diamonds$color=="H",])
[1] 8304   10
> dim(diamonds[diamonds$color=="I",])
[1] 5422   10
> dim(diamonds[diamonds$color=="J",])
[1] 2808   10
```



Bar charts(2)

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

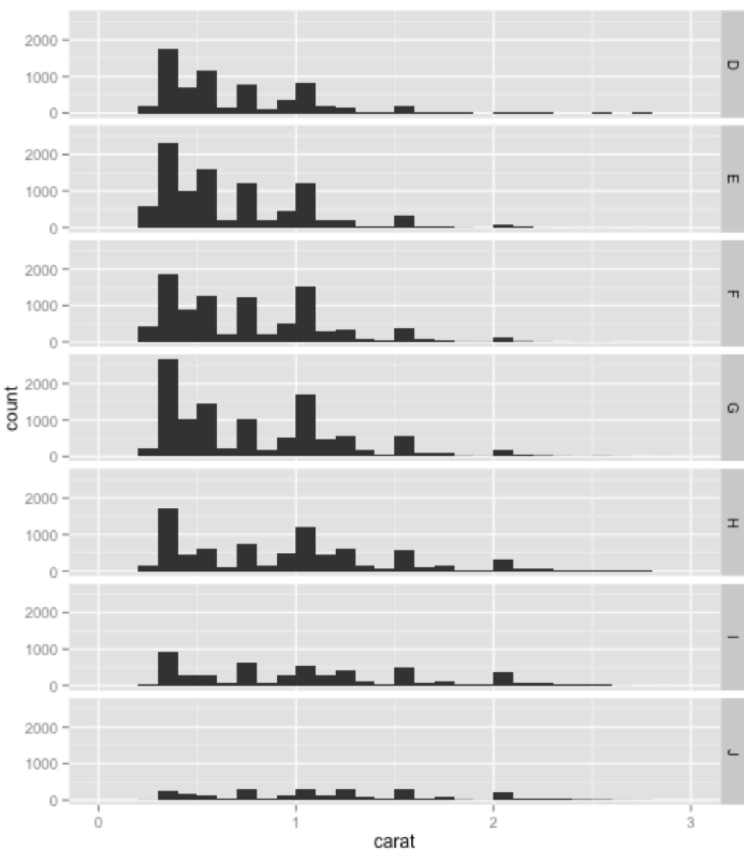
```
> sum(diamonds[diamonds$color=="D",
"carat"])
[1] 4456.56
> sum(diamonds[diamonds$color=="E",
"carat"])
[1] 6445.12
> sum(diamonds[diamonds$color=="F",
"carat"])
[1] 7028.05
> sum(diamonds[diamonds$color=="G",
"carat"])
[1] 8708.28
> sum(diamonds[diamonds$color=="H",
"carat"])
[1] 7571.58
> sum(diamonds[diamonds$color=="I",
"carat"])
[1] 5568
> sum(diamonds[diamonds$color=="J",
"carat"])
[1] 3263.28
```

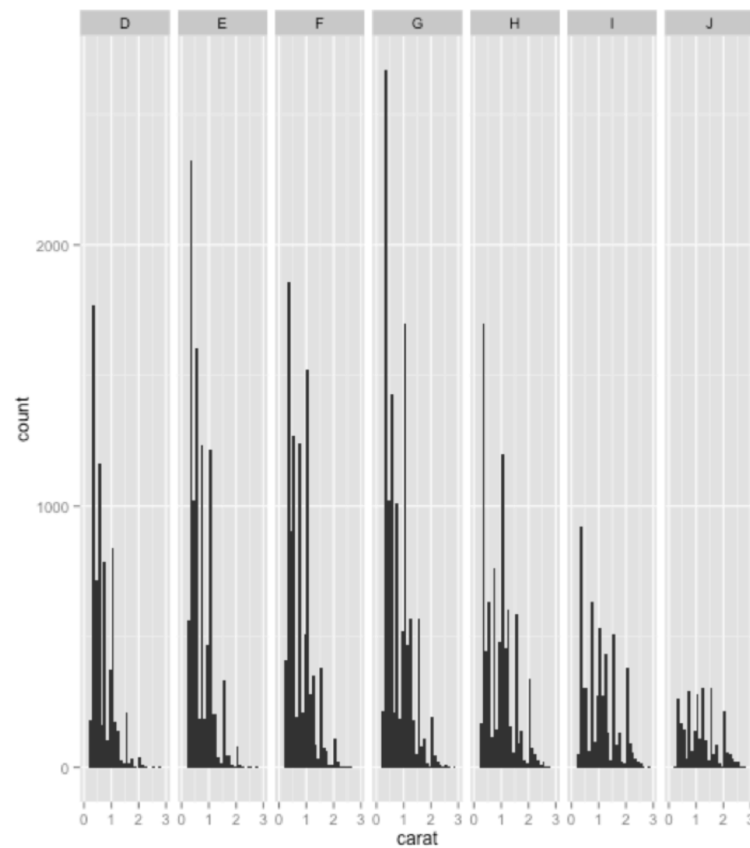


Faceting(1)

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

arranged on a grid specified by a faceting formula which looks like **row var ~ col var**

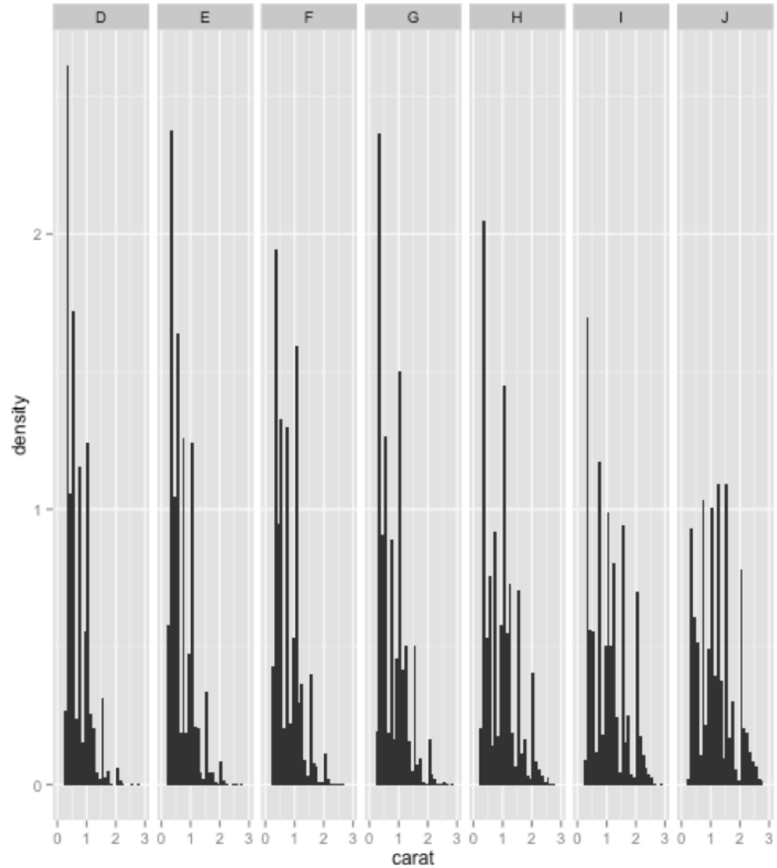




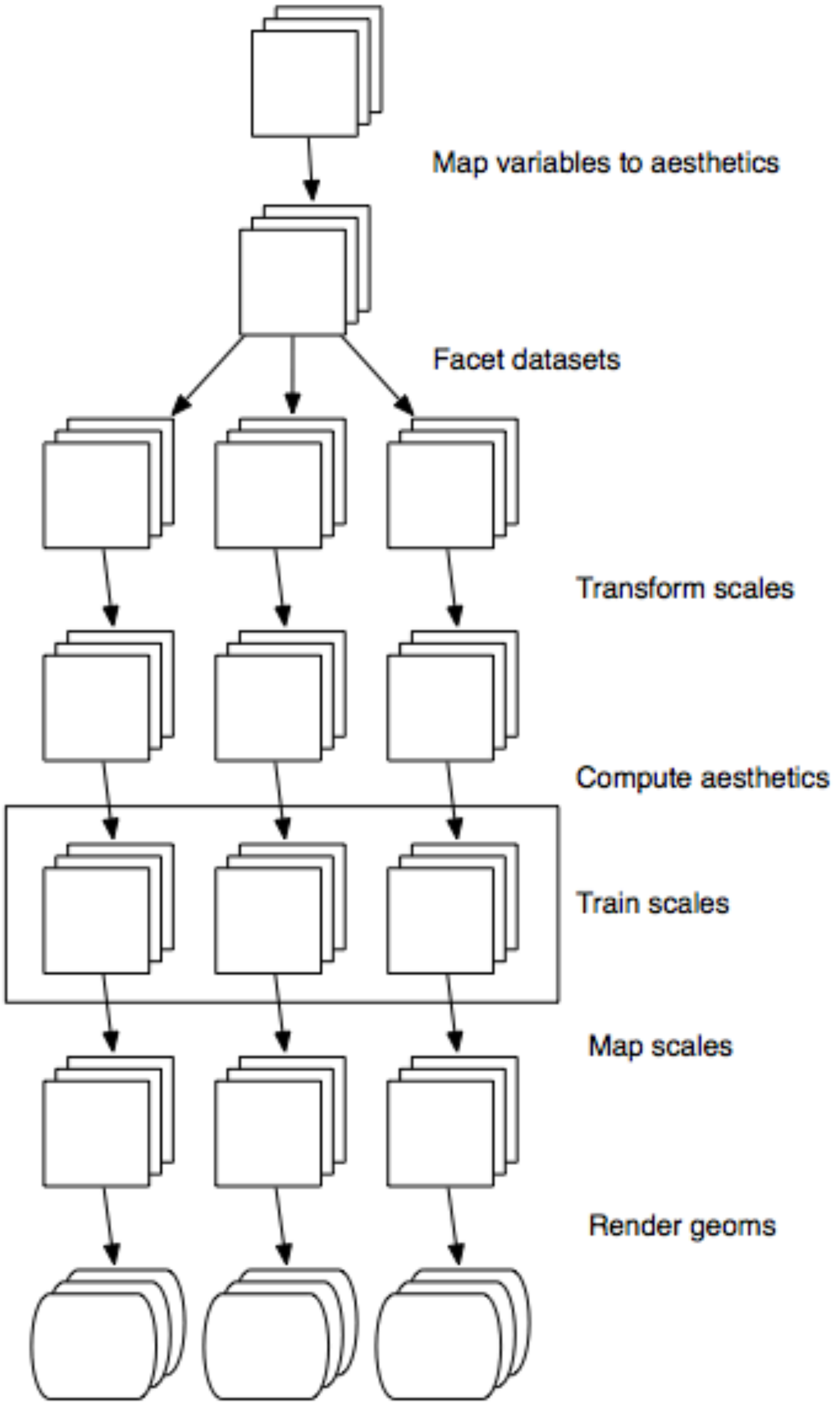
Faceting(2)

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

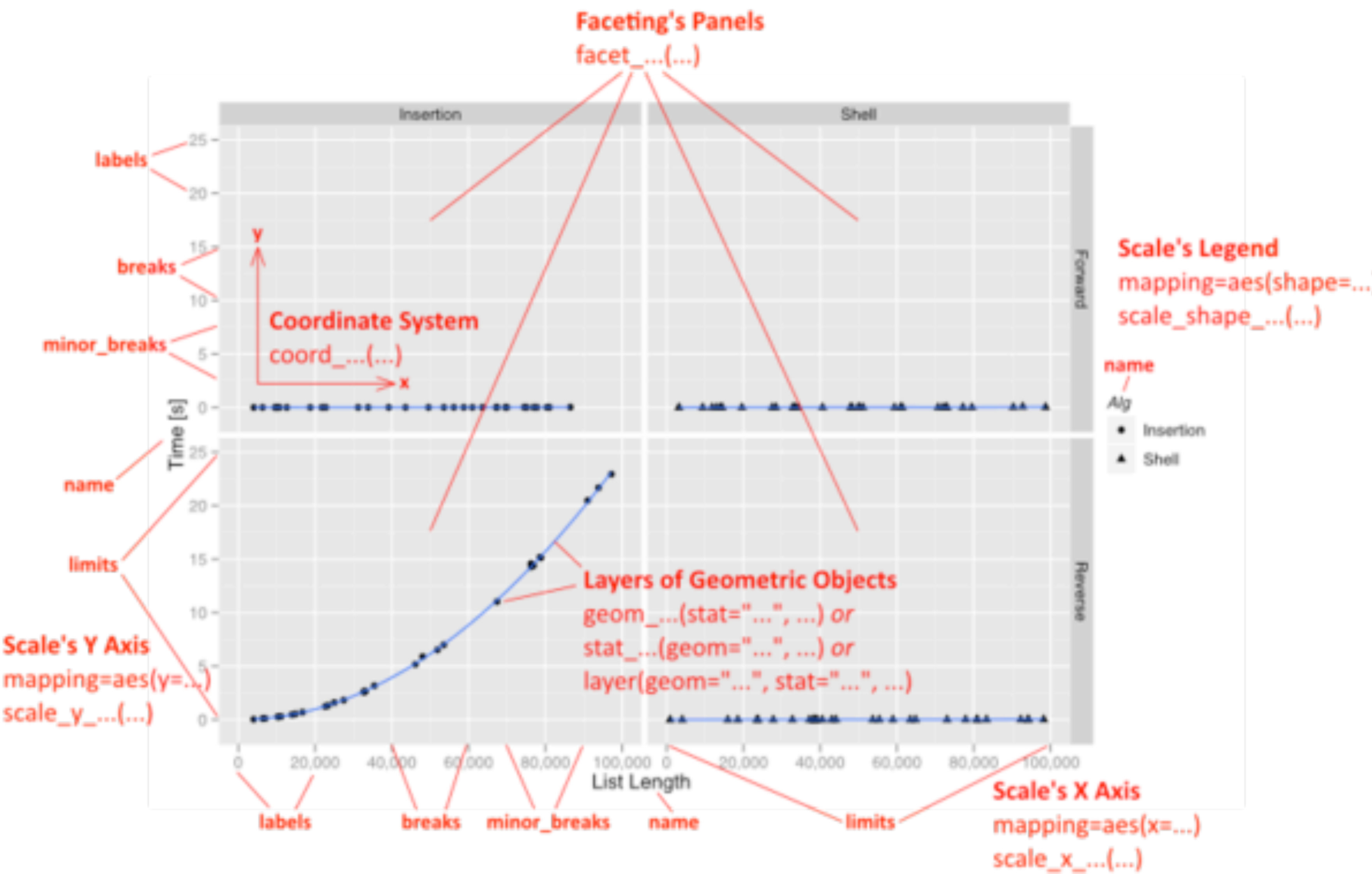
The **..density..** syntax is new. The y-axis of the histogram does not come from the original data, but from the statistical transformation that counts the number of observations in each bin. Using ..density.. tells ggplot2 to map the density to the y-axis instead of the default use of count.



Plot generation process



- Each square represents a layer, and this schematic represents a plot with three layers and three panels.
- All steps work by transforming individual data frames, except for training scales which doesn’t affect the data frame and operates across all datasets simultaneously.



Faceting's Panels
facet_...(...)
Insertion
Shell
Forward
Reverse
labels
breaks
minor_breaks
name
limits
Coordinate System
coord_...(...)
y
x
Time [s]
List Length
Scale's Legend
mapping=aes(shape=...)
scale_shape_...(...)
name
Alg
Insertion
Shell
Layers of Geometric Objects
geom_...(stat="...", ...) or
stat_...(geom="...", ...) or
layer(geom="...", stat="...", ...)
Scale's Y Axis
mapping=aes(y=...)
scale_y_...(...)
labels
breaks
minor_breaks
name
limits
Scale's X Axis
mapping=aes(x=...)
scale_x_...(...)

Layers

- Plots can be created in two ways: all at once with qplot(), as shown previously
- or piece-by-piece with ggplot() and layer functions

```
> p<-qplot(carat, price, data = diamonds[1:50,], colour = color)
> summary(p)
data: carat, cut, color, clarity, depth, table, price, x, y, z
[50x10]
mapping:  colour = color, x = carat, y = price
faceting: facet_null()
-----------------------------------
geom_point:
stat_identity:
position_identity: (width = NULL, height = NULL)
```

Creating plot(1)

- 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.

```
p <- ggplot(diamonds, aes(carat, price, colour = cut))
```

- This plot cannot be displayed until we add a layer

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

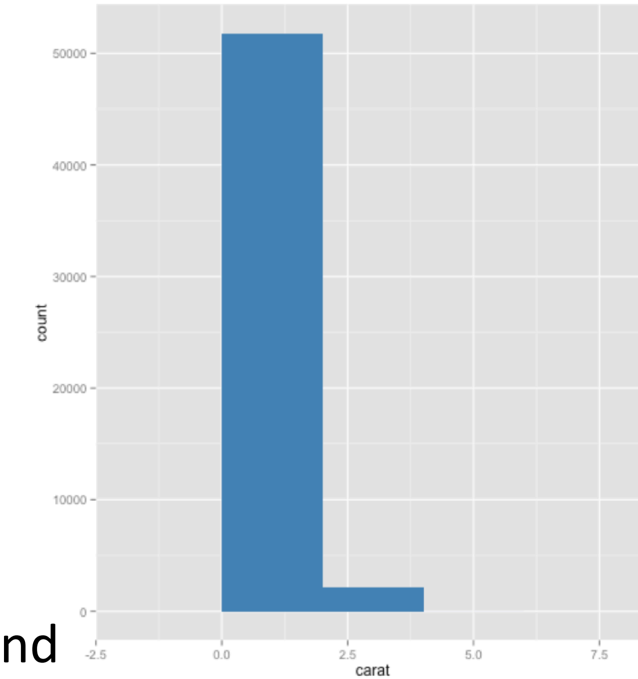
Creating plot(2)

- 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:

```
layer(geom, geom_params, stat, stat_params, data, mapping, position)
```

Creating plot(3)

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



- Simplify it by using shortcuts: every **geom** is associated with a default statistic and position, and every statistic with a default **geom**.
- 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.

```
geom_histogram(binwidth = 2, fill = "steelblue")
```

Creating plot(4)

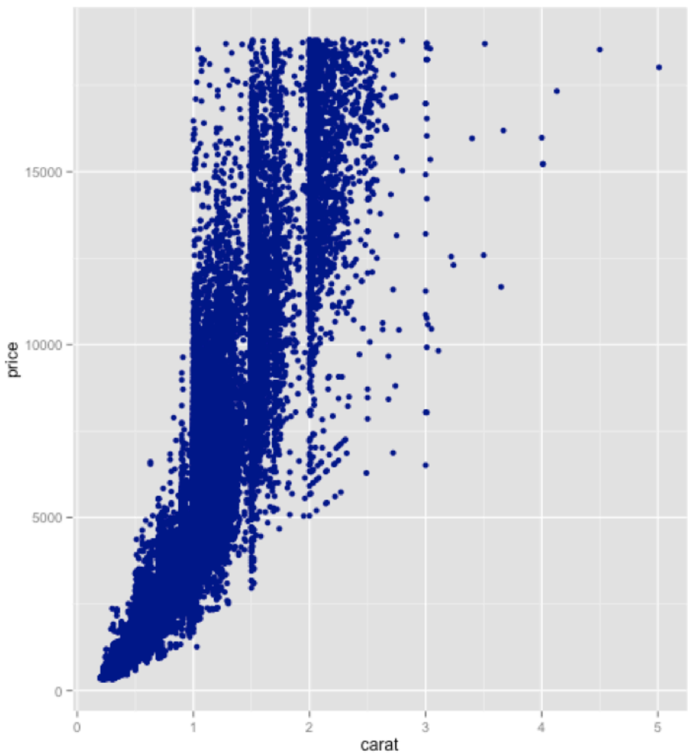
```
geom_XXX(mapping, data, ..., geom, position)
stat_XXX(mapping, data, ..., stat, position)
```

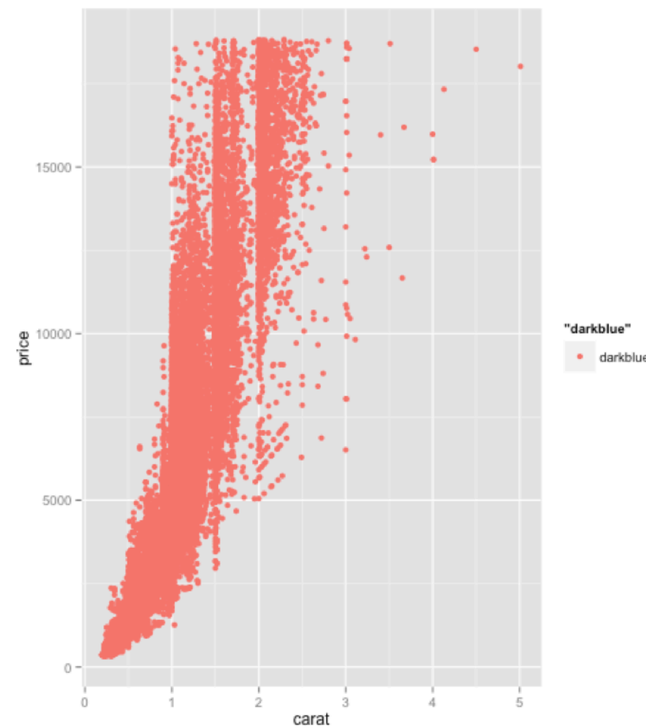
- **mapping** (optional): A set of aesthetic mappings, specified using the aes() function and combined with the plot defaults
- **data** (optional): A data set which overrides the default plot data set.
- **...** : Parameters for the geom or stat, such as bin width in the histogram or bandwidth for a loess smoother.
- **geom** or **stat** (optional): You can override the default stat for a geom, or the default geom for a stat.
- **position** (optional): Choose a method for adjusting overlapping objects

Creating plot(5)

```
p<-ggplot(diamonds,aes(carat,price))+geom_point(colour="darkblue")
p
p<-ggplot(diamonds,aes(carat,price))+geom_point(aes(colour="darkblue"))
p
```

This maps DOES NOT SET the colour to the value **“darkblue”**. It creates a new variable containing only the value “darkblue” and then maps colour to that new variable. Because this value is discrete, the default colour scale uses evenly spaced colours on the colour wheel, and since there is only one value this colour is **pinkish**.



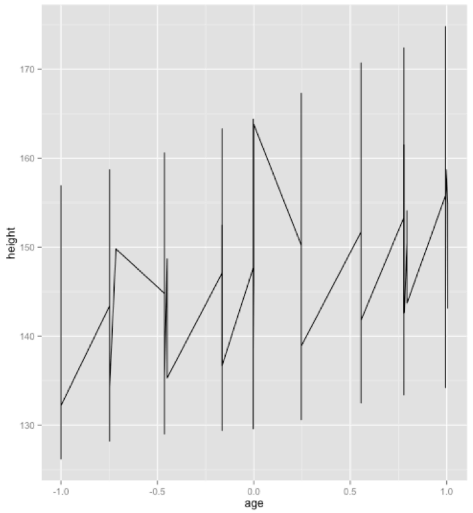


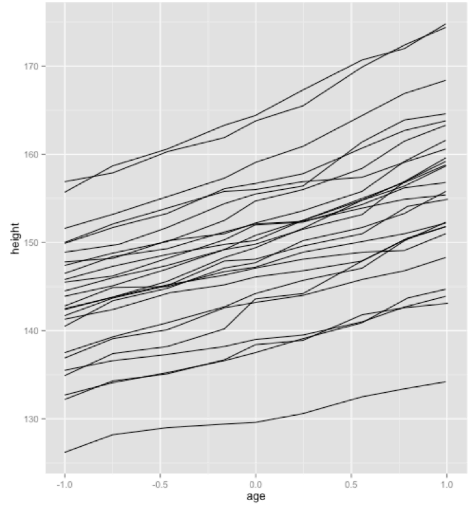
Creating plot(6): Grouping

- **geoms** can be individual and collective geoms
- By default, **group** is set to the interaction of all discrete variables in the plot
- When it doesn't, explicitly define the grouping structure, by mapping group to a variable that has a different value for each group
- **interaction()** is useful if a single pre-existing variable doesn't cleanly separate groups

Creating plot(7): Grouping

```
>head(Oxboys)
Grouped Data: height ~ age | Subject
  Subject     age height Occasion
1       1 -1.0000  140.5        1
2       1 -0.7479  143.4        2
3       1 -0.4630  144.8        3
4       1 -0.1643  147.1        4
5       1 -0.0027  147.7        5
6       1  0.2466  150.2        6
```

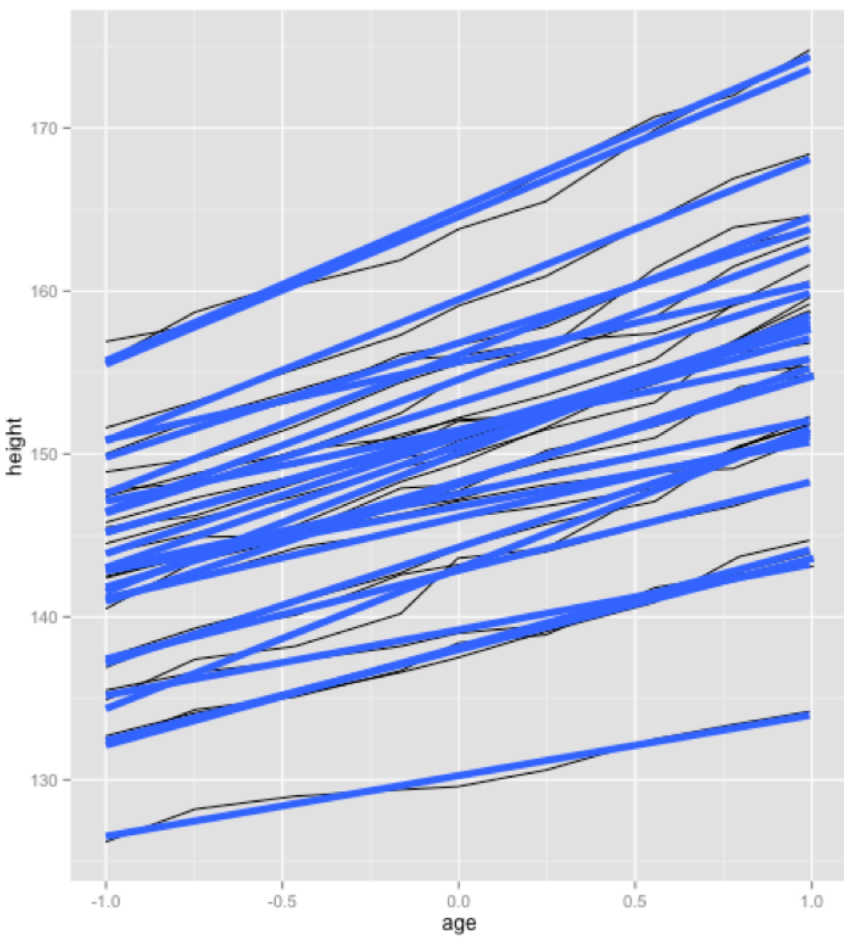


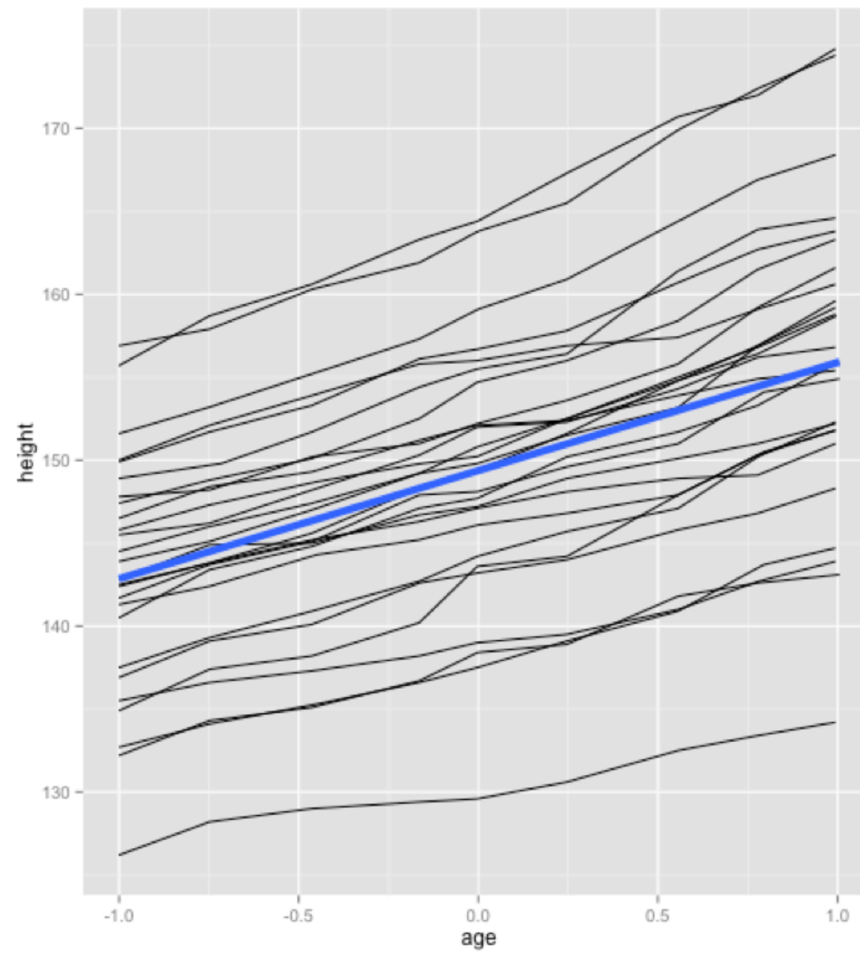



```
p <- ggplot(Oxboys, aes(age, height)) + geom_line()
p
p <- ggplot(Oxboys, aes(age, height, group = Subject)) + geom_line()
p
```

Creating plot(8): Grouping

```
p <- ggplot(Oxboys, aes(age, height, group = Subject)) + geom_line()
p + geom_smooth(method="lm", size=2, se=F)
p <- ggplot(Oxboys, aes(age, height, group = Subject)) + geom_line()
p + geom_smooth(aes(group="dummy"),method="lm", size=2, se=F)
```





Geoms

- Geometric objects (**geoms**)
  - Perform the actual rendering of the layer
  - Control the type of plot that you create
  - Has a set of aesthetics
  - Differ in the way they are parameterised
  - Have a default statistic

Default statistics and aesthetics. Emboldened aesthetics are required

Name	Default stat	Aesthetics
abline	abline	colour, linetype, size
area	identity	colour, fill, linetype, size, **x**, **y**
bar	bin	colour, fill, linetype, size, weight, **x**
bin2d	bin2d	colour, fill, linetype, size, weight, **xmax**, **xmin**, **ymax**, **ymin**
blank	identity	
boxplot	boxplot	colour, fill, **lower**, **middle**, size, **upper**, weight, **x**, **ymax**, **ymin**
contour	contour	colour, linetype, size, weight, **x**, **y**
crossbar	identity	colour, fill, linetype, size, **x**, **y**, **ymax**, **ymin**
density	density	colour, fill, linetype, size, weight, **x**, **y**
density2d	density2d	colour, linetype, size, weight, **x**, **y**
errorbar	identity	colour, linetype, size, width, **x**, **ymax**, **ymin**
freqpoly	bin	colour, linetype, size
hex	binhex	colour, fill, size, **x**, **y**
histogram	bin	colour, fill, linetype, size, weight, **x**
hline	hline	colour, linetype, size
jitter	identity	colour, fill, shape, size, **x**, **y**
line	identity	colour, linetype, size, **x**, **y**
linerange	identity	colour, linetype, size, **x**, **ymax**, **ymin**
path	identity	colour, linetype, size, **x**, **y**
point	identity	colour, fill, shape, size, **x**, **y**
pointrange	identity	colour, fill, linetype, shape, size, **x**, **y**, **ymax**, **ymin**
polygon	identity	colour, fill, linetype, size, **x**, **y**
quantile	quantile	colour, linetype, size, weight, **x**, **y**
rect	identity	colour, fill, linetype, size, **xmax**, **xmin**, **ymax**, **ymin**
ribbon	identity	colour, fill, linetype, size, **x**, **ymax**, **ymin**
rug	identity	colour, linetype, size
segment	identity	colour, linetype, size, **x**, **xend**, **y**, **yend**
smooth	smooth	alpha, colour, fill, linetype, size, weight, **x**, **y**
step	identity	colour, linetype, size, **x**, **y**
text	identity	angle, colour, hjust, **label**, size, vjust, **x**, **y**
tile	identity	colour, fill, linetype, size, **x**, **y**
vline	vline	colour, linetype, size

Stat(1)

- Statistical transformation (**stat**)
  - Transforms the data by summarising it in some manner
  - e.g., smoother calculates the mean of y, condition of x
  - A stat must be location-scale invariant (transformation stays same when scale is changed)
    $f(x+a)=f(x)+a$; $f(b.x)=b.f(x)$
  - Takes a dataset as input, returns a dataset as output and introduces new variables
  - e.g., **stat_bin** (statistic used to make histograms, produces)
    - **count**: number of observation in each bin
    - **density:** density of observation in each bin (percentage of total/bar width)
    - **x**: the centre of bin
  - The names of generated variables must be surrounded with .. when used

Stat(2)

Name	Description
bin	Bin data
boxplot	Calculate components of box and whisker plot
contour	Contours of 3d data
density	Density estimation, 1D
density_2d	Density estimation, 2D
function	Superimpose a function
identity	Don't transform data
qq	Calculation for quantile-quantile plot
quantile	Continuous quantiles
smooth	Add a smoother
spoke	Convert angle and radius to xend and yend
step	Create stair steps
sum	Sum unique values. Useful for overplotting on scatterplots
summary	Summarise y values at every unique x
unique	Remove duplicates

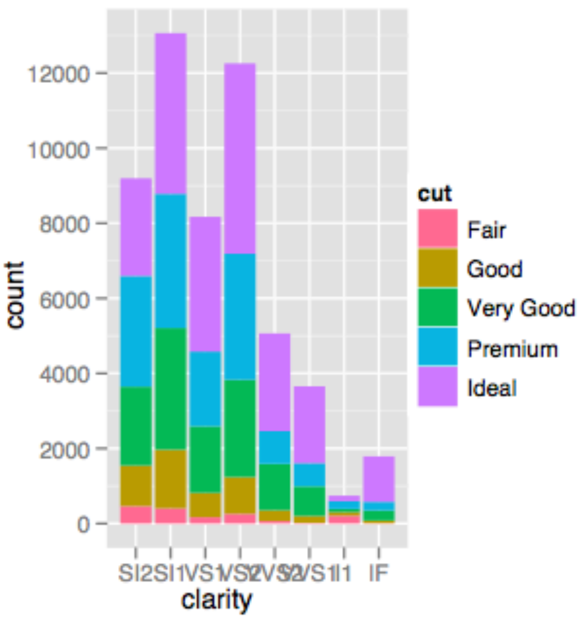
Position adjustments(1)

- Apply minor tweaks to the position of elements within a layer
- Normally used with **discrete** data
- Continuous data typically don’t overlap and when do, **jittering** is sufficient

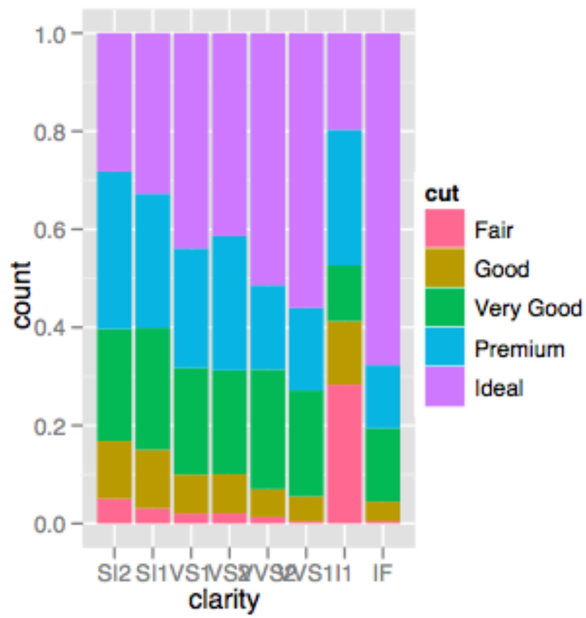
Position adjustments(2)

Adjustment	Description
dodge	Adjust position by dodging overlaps to the side
fill	Stack overlapping objects and standardise have equal height
identity	Don't adjust position
jitter	Jitter points to avoid overplotting
stack	Stack overlapping objects on top of one another

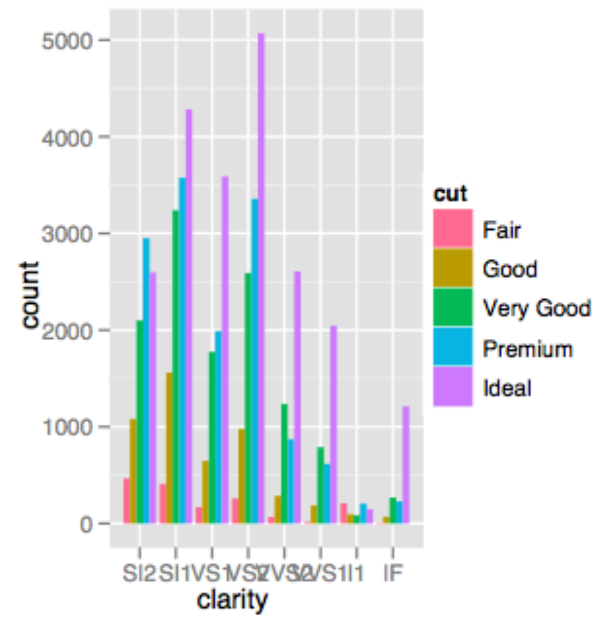
Position adjustments(3)



stacking



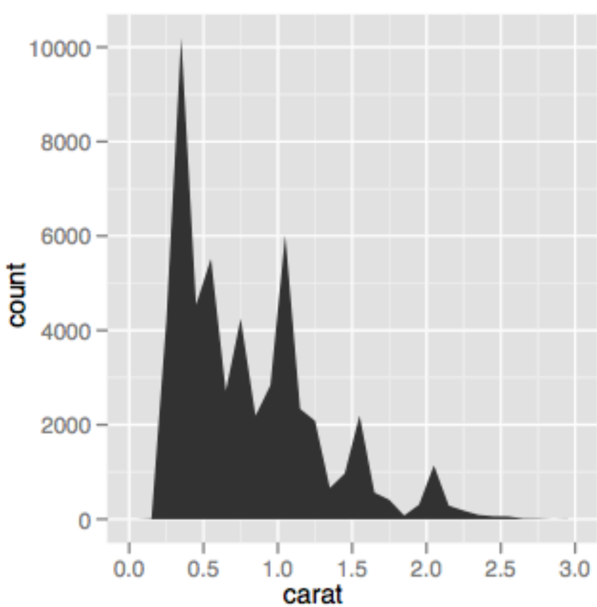
filling

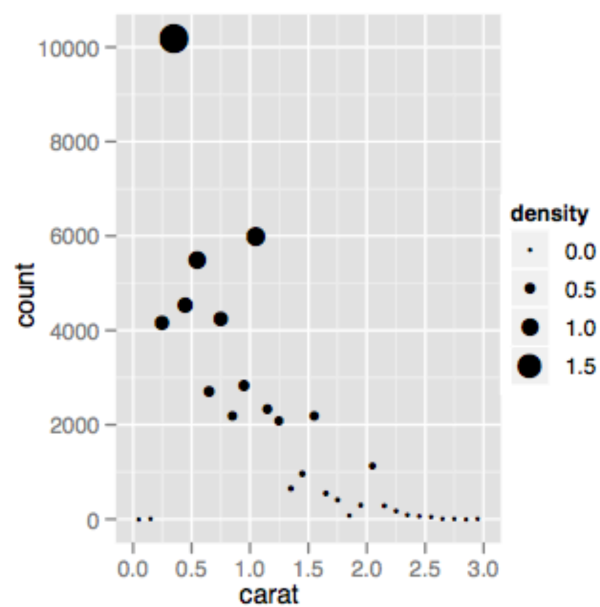


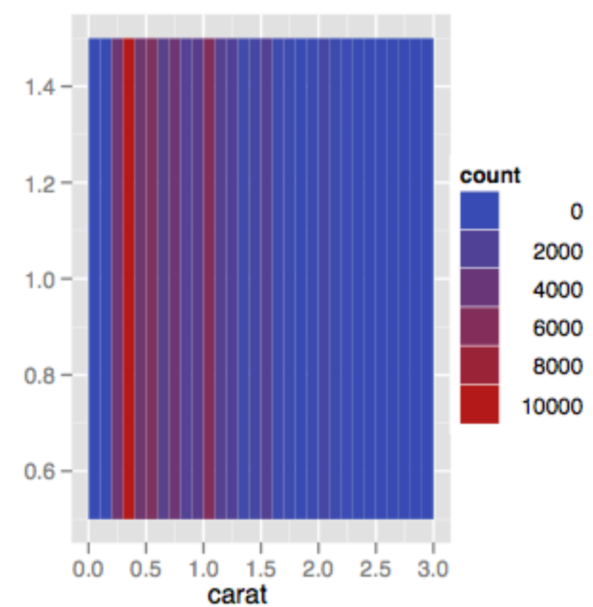
dodging

Position adjustments(4)

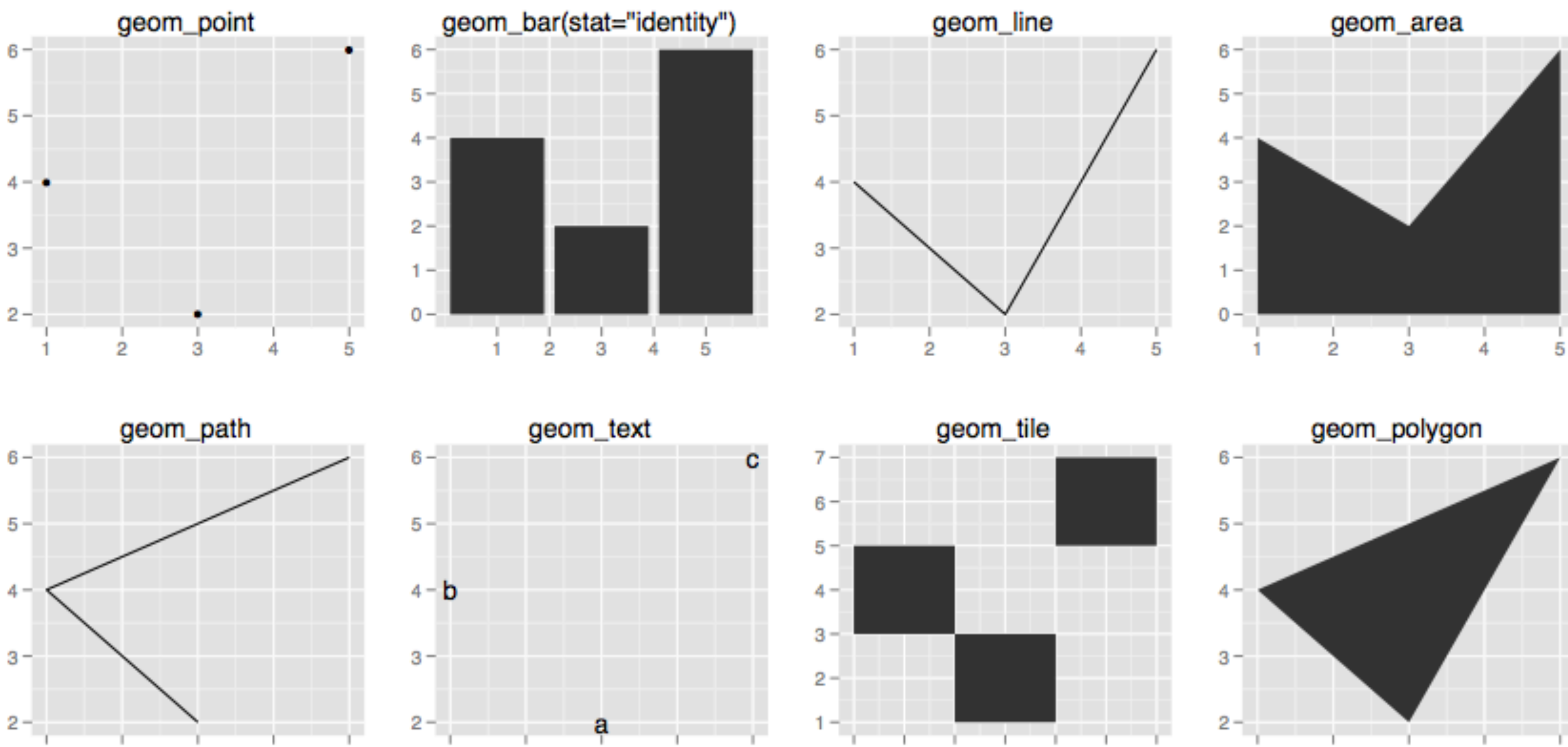
```
d <- ggplot(diamonds, aes(carat)) + xlim(0, 3)
d + stat_bin(aes(ymax = ..count..), binwidth = 0.1, geom = "area")
d + stat_bin(
  aes(size = ..density..), binwidth = 0.1,
  geom = "point", position="identity”)
d + stat_bin(
  aes(y = 1, fill = ..count..), binwidth = 0.1,
  geom = "tile", position="identity”)
```







```r
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")
```



Scales, axes and legends(1)

- Scales control the mapping from data to aesthetics
- Data → size, colour, position or shape
- data space (domain) → scale → aesthetic space (range)
- Process of scaling: **Transformation** (log transformation?), **Training** (minimum?maximum? Of a continuous variable; unique levels? of a categorical variable), and **Mapping**
- Four categories:
  – position scales
  – colour scales
  – manual discrete scales
  – identity scales
- **guide**: perform the inverse mapping from aesthetic space to data space
  – For position aesthetics, axes are the guides
  – Any other aesthetics, legends are the guides
- Every aesthetic has a default scale: **set_default_scale()**

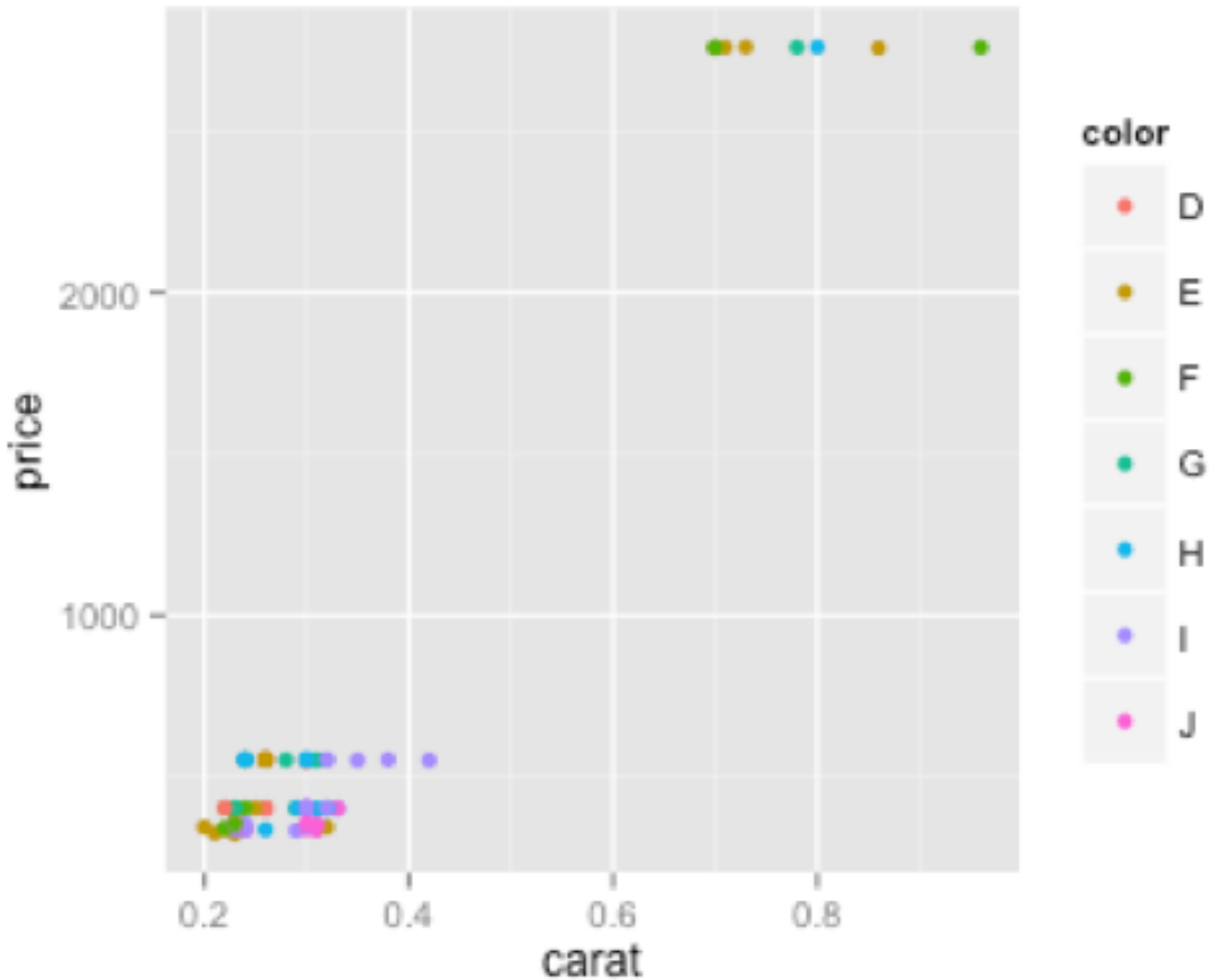
Scales, axes and legends(2)

- All scale constructors start with **scale_**
- Followed by the name of the aesthetic (e.g., **colour_**, **shape_**, or **x_**)
- Finally name of the scale (e.g., **gradient**, **hue**, or **manual**),
- E.g., scale_colour_hue(), scale_fill_brewer()

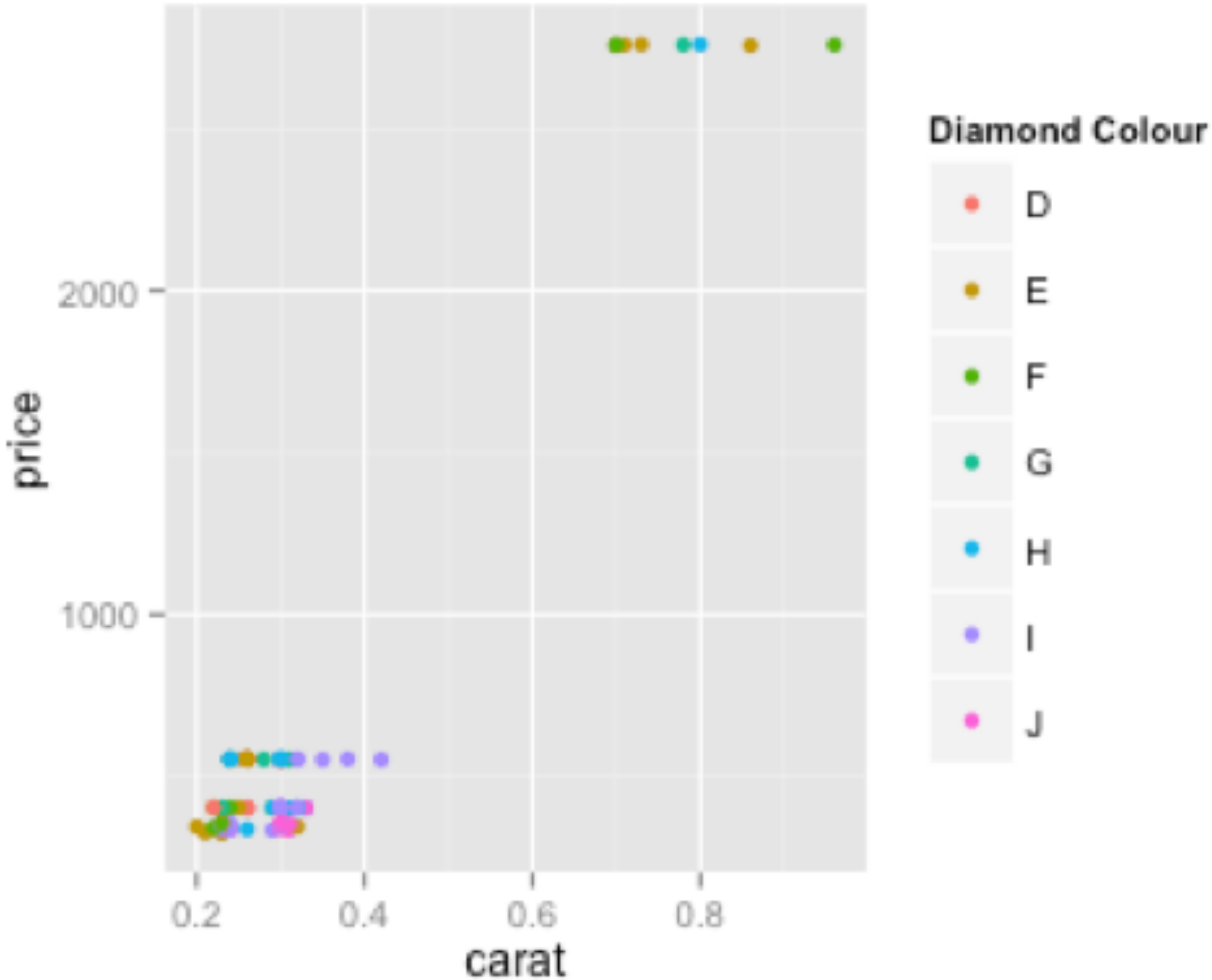
Aesthetic	Discrete	Continuous
Colour and fill	brewer grey **hue** identity manual	**gradient** gradient2 gradientn
Position (x, y)	**discrete**	**continuous** date
Shape	**shape** identity manual	
Line type	**linetype** identity manual	
Size	identity manual	**size**

Scale example

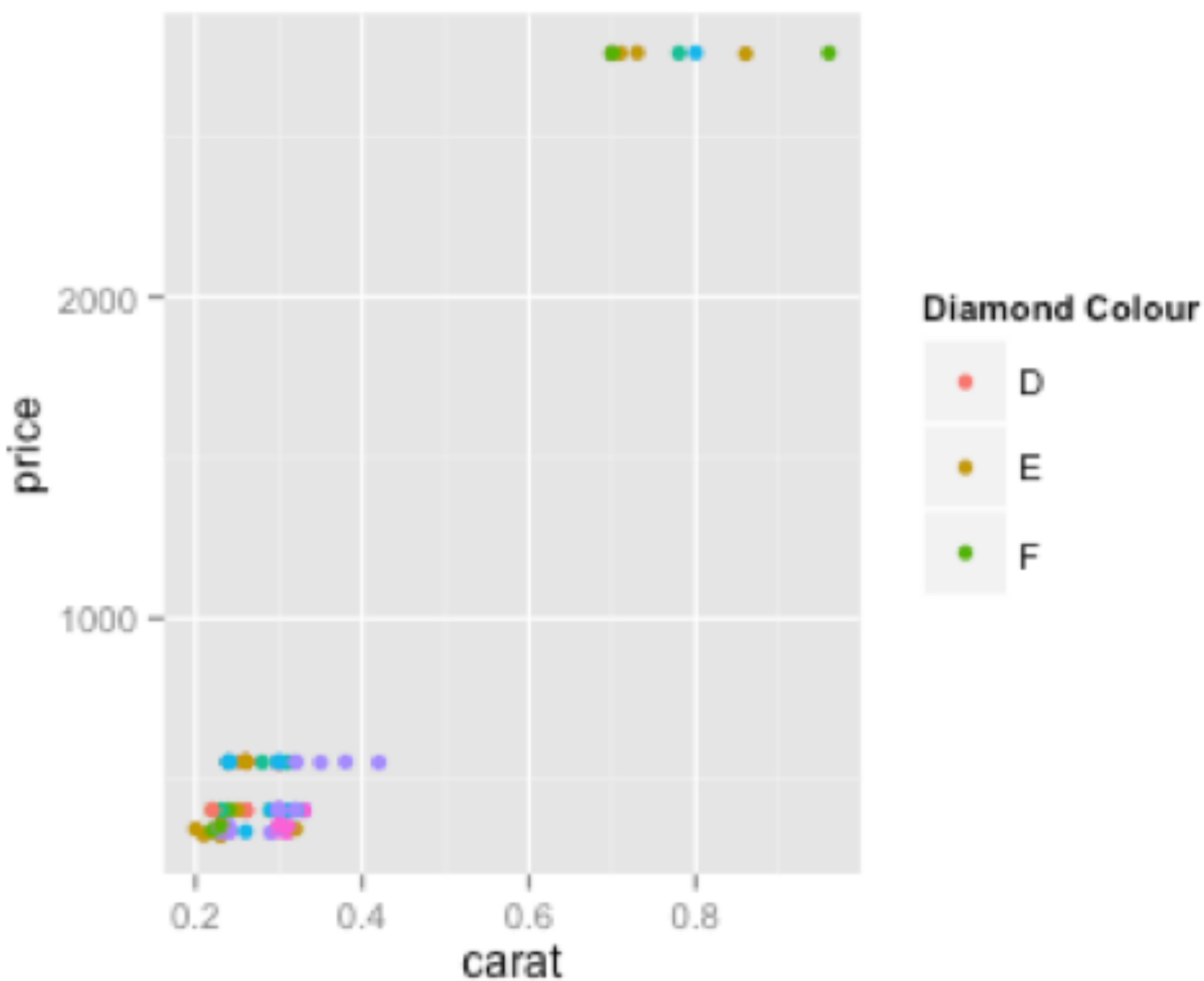
```
qplot(carat, price, data = diamonds[1:100,], colour = color)
```



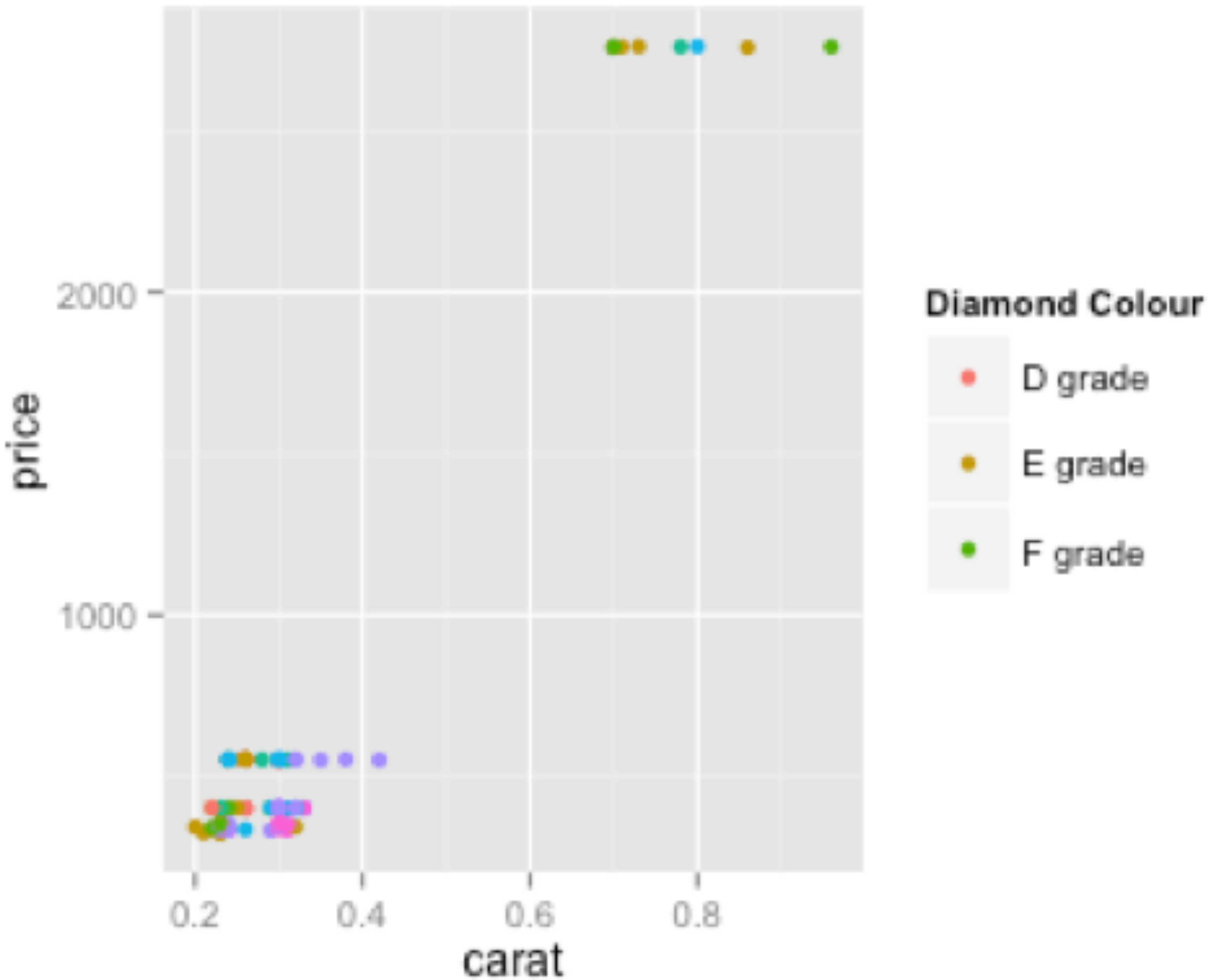
```
qplot(carat, price, data = diamonds[1:100,], colour = color)  +
scale_color_hue("Diamond Colour")
```



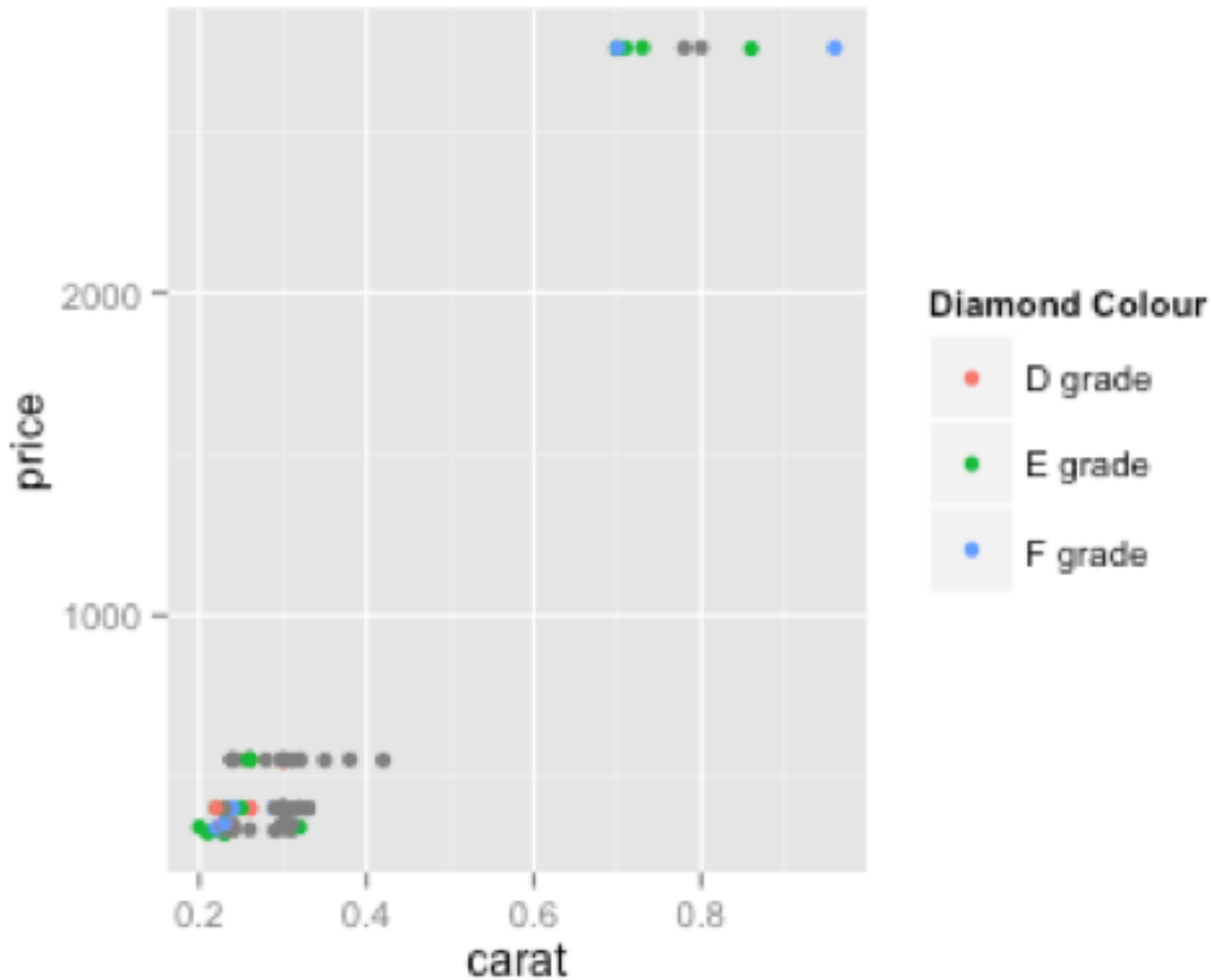
```
qplot(carat, price, data = diamonds[1:100,], colour = color)  +
  scale_color_hue("Diamond Colour", breaks=c("D","E","F"))
```



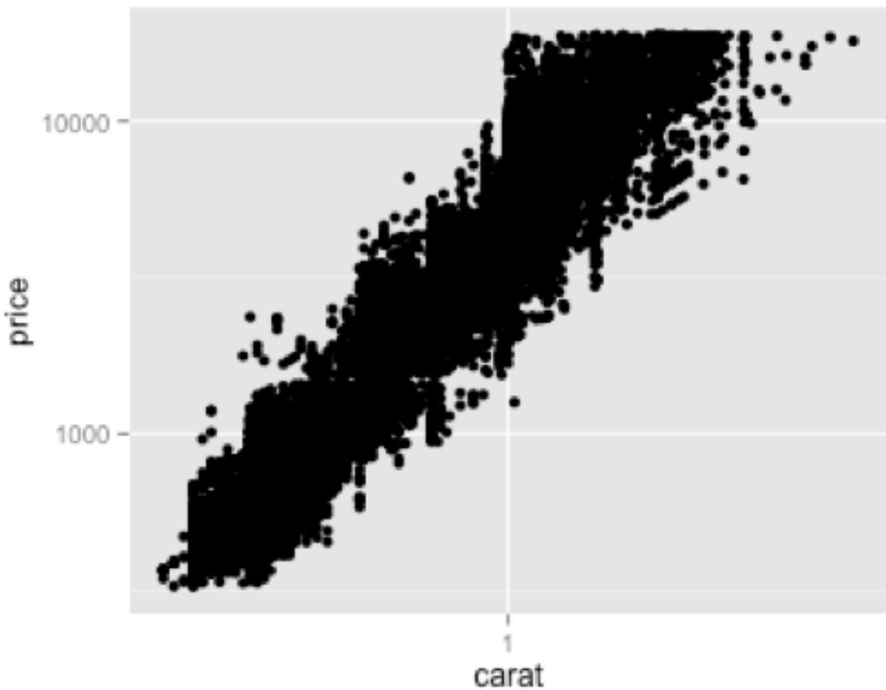
```
qplot(carat, price, data = diamonds[1:100,], colour = color)  +
  scale_color_hue("Diamond Colour", breaks=c("D","E","F"),
labels=c("D grade","E grade","F grade"))
```



```
qplot(carat, price, data = diamonds[1:100,], colour = color)  +
  scale_color_hue("Diamond Colour", limits=c("D","E","F"),
labels=c("D grade","E grade","F grade"))
```



Continuous scales(1)



Name	Function $f(x)$	Inverse $f^{-1}(y)$
asn	$\tanh^{-1}(x)$	$\tanh(y)$
exp	e^x	$\log(y)$
identity	x	y
log	$\log(x)$	e^y
log10	$\log_{10}(x)$	10^y
log2	$\log_2(x)$	2^y
logit	$\log(\frac{x}{1-x})$	$\frac{1}{1+e(y)}$
pow10	10^x	$\log_{10}(y)$
probit	$\Phi(x)$	$\Phi^{-1}(y)$
recip	x^{-1}	y^{-1}
reverse	$-x$	$-y$
sqrt	$x^{1/2}$	y^2

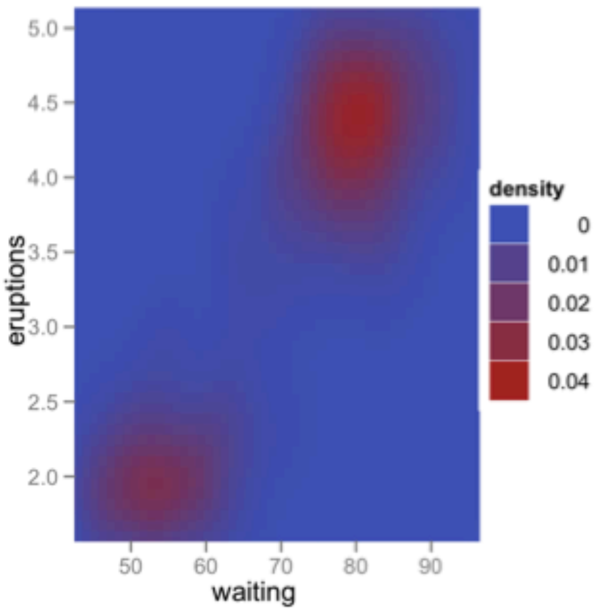
```
qplot(log10(carat), log10(price), data = diamonds)
qplot(carat, price, data = diamonds) + scale_x_log10() + scale_y_log10()
qplot(carat, price, data = diamonds) + scale_x_continuous(trans="log10") +
scale_y_log10()
```

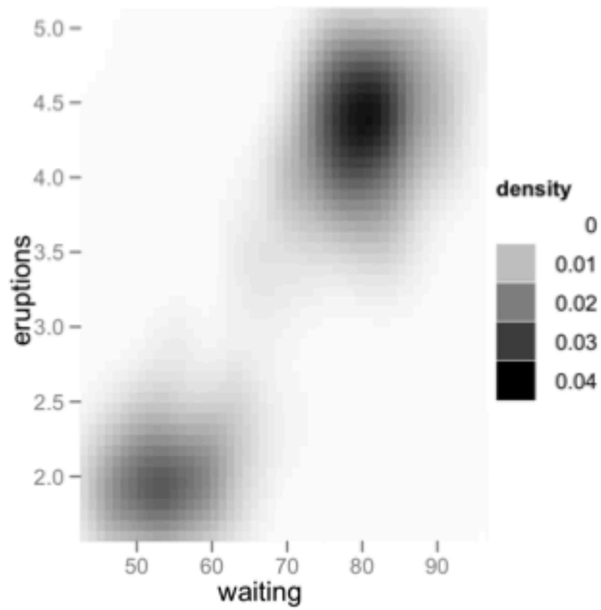
Continuous scales(2)

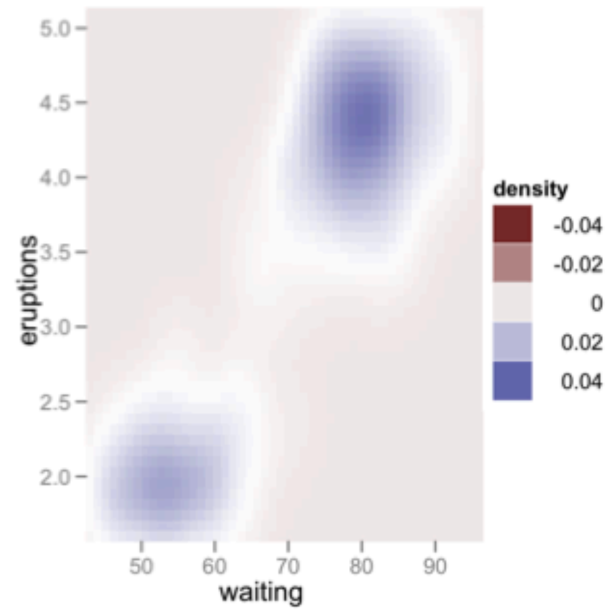
- **scale_colour_gradient()** and **scale_fill_gradient()**: a two–colour gradient, low–high. Arguments low and high control the colours at either end of the gradient.
- **scale_colour_gradient2()** and **scale_fill_gradient2():** a three–colour gradient, low–med–high.
- **scale_colour_gradientn()** and **scale_fill_gradientn()**: a custom n–colour gradient.

Continuous scales(3)

```
f2d <- with(faithful, MASS::kde2d(eruptions, waiting, h = c(1, 10), n = 50))
df <- with(f2d, cbind(expand.grid(x, y), as.vector(z)))
names(df) <- c("eruptions", "waiting", "density")
erupt <- ggplot(df, aes(waiting, eruptions, fill = density)) +
  geom_tile() +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(expand = c(0, 0))
erupt + scale_fill_gradient(limits = c(0, 0.04))
erupt + scale_fill_gradient(limits = c(0, 0.04), low="white", high="black")
erupt + scale_fill_gradient2(limits = c(-0.04, 0.04),midpoint = mean(df$density))
```







Continuous scales(4)

```
library(colorspace)
fill_gradn <- function(pal) {
  scale_fill_gradientn(colours = pal(7), limits = c(0, 0.04))
}
erupt + fill_gradn(rainbow_hcl)
erupt + fill_gradn(diverge_hcl)
erupt + fill_gradn(heat_hcl)
```