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)

- Carbo (-0.59 ***)
- CODs (-0.46 ***)
- CODi (-0.08 )
- NH4 (-0.43 ***)
- perCODshy (-0.6 ***)
- pH (0.05 )
- Prot (-0.48 ***)
- Temp (-0.45 ***)
- TS (0.37 ***)
- VFA (-0.45 ***)
- VS (0.07 )
Heatmap (heatmap.R)
ggplot2 basics
ggplot_basis.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

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

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

http://www.handprint.com/HP/WCL/color7.html
Aesthetic Attributes (3)

```r
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 = \( y \sim s(x) \) 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 \( y \sim s(x, bs=\text{"cs"}) \). This is used by default when there are more than 1,000 points.

```r
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 = \( y \sim \text{poly}(x, 2) \) to specify a degree 2 polynomial, or better, load the splines package and use a natural spline: formula = \( y \sim \text{ns}(x, 2) \). 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.

```r
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)

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

![Histogram and density plots](image-url)
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$color=="F",])
[1] 9542 10
> dim(diamonds[diamonds$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)

```r
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.
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

```r
> 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.

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

- This plot cannot be displayed until we add a layer

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

```r
layer(geom, geom_params, stat, stat_params, data, mapping, position)
```
Creating plot(3)

```r
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.

```r
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)

```r
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

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

```r
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 \textit{(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
<table>
<thead>
<tr>
<th>Name</th>
<th>Default stat</th>
<th>Aesthetics</th>
</tr>
</thead>
<tbody>
<tr>
<td>abline</td>
<td>abline</td>
<td>colour, linetype, size</td>
</tr>
<tr>
<td>area</td>
<td>identity</td>
<td>colour, fill, linetype, size, $x, y$</td>
</tr>
<tr>
<td>bar</td>
<td>bin</td>
<td>colour, fill, linetype, size, weight, $x$</td>
</tr>
<tr>
<td>bin2d</td>
<td>bin2d</td>
<td>colour, fill, linetype, size, weight, $xmax, x_{\text{min}}, y_{\text{max}}, y_{\text{min}}$</td>
</tr>
<tr>
<td>blank</td>
<td>identity</td>
<td>colour, fill, $lower, middle, size, upper$, weight, $x, y_{\text{max}}, y_{\text{min}}$</td>
</tr>
<tr>
<td>boxplot</td>
<td>boxplot</td>
<td>colour, linetype, size</td>
</tr>
<tr>
<td>contour</td>
<td>contour</td>
<td>colour, linetype, size, weight, $x, y$</td>
</tr>
<tr>
<td>crossbar</td>
<td>identity</td>
<td>colour, fill, linetype, size, $x, y, y_{\text{max}}, y_{\text{min}}$</td>
</tr>
<tr>
<td>density</td>
<td>density</td>
<td>colour, fill, linetype, size, $x, y$</td>
</tr>
<tr>
<td>density2d</td>
<td>density2d</td>
<td>colour, fill, linetype, size, weight, $x, y$</td>
</tr>
<tr>
<td>errorbar</td>
<td>identity</td>
<td>colour, linetype, size, width, $x, y_{\text{max}}, y_{\text{min}}$</td>
</tr>
<tr>
<td>freqpoly</td>
<td>bin</td>
<td>colour, linetype, size</td>
</tr>
<tr>
<td>hex</td>
<td>binhex</td>
<td>colour, fill, size, $x, y$</td>
</tr>
<tr>
<td>histogram</td>
<td>bin</td>
<td>colour, fill, linetype, size, weight, $x$</td>
</tr>
<tr>
<td>hline</td>
<td>hline</td>
<td>colour, linetype, size</td>
</tr>
<tr>
<td>jitter</td>
<td>identity</td>
<td>colour, fill, shape, size, $x, y$</td>
</tr>
<tr>
<td>line</td>
<td>identity</td>
<td>colour, linetype, size, $x, y$</td>
</tr>
<tr>
<td>linerange</td>
<td>identity</td>
<td>colour, linetype, size, $x, y_{\text{max}}, y_{\text{min}}$</td>
</tr>
<tr>
<td>path</td>
<td>identity</td>
<td>colour, linetype, size, $x, y$</td>
</tr>
<tr>
<td>point</td>
<td>identity</td>
<td>colour, fill, shape, size, $x, y$</td>
</tr>
<tr>
<td>pointrange</td>
<td>identity</td>
<td>colour, fill, linetype, shape, size, $x, y, y_{\text{max}}, y_{\text{min}}$</td>
</tr>
<tr>
<td>polygon</td>
<td>identity</td>
<td>colour, fill, linetype, size, $x, y$</td>
</tr>
<tr>
<td>quantile</td>
<td>quantile</td>
<td>colour, fill, linetype, size, weight, $x, y$</td>
</tr>
<tr>
<td>rect</td>
<td>identity</td>
<td>colour, fill, linetype, size, $x, y_{\text{max}}, y_{\text{min}}$</td>
</tr>
<tr>
<td>ribbon</td>
<td>identity</td>
<td>colour, fill, linetype, size, $x, y_{\text{max}}, y_{\text{min}}$</td>
</tr>
<tr>
<td>rug</td>
<td>identity</td>
<td>colour, linetype, size</td>
</tr>
<tr>
<td>segment</td>
<td>identity</td>
<td>colour, linetype, size, $x, x_{\text{end}}, y, y_{\text{end}}$</td>
</tr>
<tr>
<td>smooth</td>
<td>smooth</td>
<td>alpha, colour, fill, linetype, size, weight, $x, y$</td>
</tr>
<tr>
<td>step</td>
<td>identity</td>
<td>colour, linetype, size, $x, y$</td>
</tr>
<tr>
<td>text</td>
<td>identity</td>
<td>angle, colour, hjust, label, size, vjust, $x, y$</td>
</tr>
<tr>
<td>tile</td>
<td>identity</td>
<td>colour, fill, linetype, size, $x, y$</td>
</tr>
<tr>
<td>vline</td>
<td>vline</td>
<td>colour, linetype, size</td>
</tr>
</tbody>
</table>
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)
    \[ f(x+a) = f(x) + a \; ; \; f(b \cdot x) = b \cdot f(x) \]
  - 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
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>Bin data</td>
</tr>
<tr>
<td>boxplot</td>
<td>Calculate components of box and whisker plot</td>
</tr>
<tr>
<td>contour</td>
<td>Contours of 3D data</td>
</tr>
<tr>
<td>density</td>
<td>Density estimation, 1D</td>
</tr>
<tr>
<td>density.2d</td>
<td>Density estimation, 2D</td>
</tr>
<tr>
<td>function</td>
<td>Superimpose a function</td>
</tr>
<tr>
<td>identity</td>
<td>Don't transform data</td>
</tr>
<tr>
<td>qq</td>
<td>Calculation for quantile-quantile plot</td>
</tr>
<tr>
<td>quantile</td>
<td>Continuous quantiles</td>
</tr>
<tr>
<td>smooth</td>
<td>Add a smoother</td>
</tr>
<tr>
<td>spoke</td>
<td>Convert angle and radius to xend and yend</td>
</tr>
<tr>
<td>step</td>
<td>Create stair steps</td>
</tr>
<tr>
<td>sum</td>
<td>Sum unique values. Useful for overplotting on scatterplots</td>
</tr>
<tr>
<td>summary</td>
<td>Summarise y values at every unique x</td>
</tr>
<tr>
<td>unique</td>
<td>Remove duplicates</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>Adjustment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dodge</td>
<td>Adjust position by dodging overlaps to the side</td>
</tr>
<tr>
<td>fill</td>
<td>Stack overlapping objects and standardise have equal height</td>
</tr>
<tr>
<td>identity</td>
<td>Don’t adjust position</td>
</tr>
<tr>
<td>jitter</td>
<td>Jitter points to avoid overplotting</td>
</tr>
<tr>
<td>stack</td>
<td>Stack overlapping objects on top of one another</td>
</tr>
</tbody>
</table>
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")
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 $\rightarrow$ size, colour, position or shape
- data space (domain) $\rightarrow$ scale $\rightarrow$ 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()`

<table>
<thead>
<tr>
<th>Aesthetic</th>
<th>Discrete</th>
<th>Continuous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colour and fill</td>
<td>brewer</td>
<td>gradient</td>
</tr>
<tr>
<td></td>
<td>grey</td>
<td>gradient2</td>
</tr>
<tr>
<td></td>
<td>hue</td>
<td>gradientn</td>
</tr>
<tr>
<td></td>
<td>identity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>manual</td>
<td></td>
</tr>
<tr>
<td>Position (x, y)</td>
<td>discrete</td>
<td>continuous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>date</td>
</tr>
<tr>
<td>Shape</td>
<td>shape</td>
<td></td>
</tr>
<tr>
<td></td>
<td>identity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>manual</td>
<td></td>
</tr>
<tr>
<td>Line type</td>
<td>linetype</td>
<td></td>
</tr>
<tr>
<td></td>
<td>identity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>manual</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>identity</td>
<td>size</td>
</tr>
<tr>
<td></td>
<td>manual</td>
<td></td>
</tr>
</tbody>
</table>
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

\begin{align*}
\text{qplot}(\log_{10}(\text{carat}), & \log_{10}(\text{price}), \text{data} = \text{diamonds}) \\
\text{qplot}(\text{carat}, \text{price}, & \text{data} = \text{diamonds}) + \text{scale}_x\_\log10() + \text{scale}_y\_\log10() \\
\text{qplot}(\text{carat}, \text{price}, & \text{data} = \text{diamonds}) + \text{scale}_x\_\text{continuous}(\text{trans} = "\log10") + \text{scale}_y\_\log10() \\
\end{align*}
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)

```r
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(3)](image)
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)