
Applied geostatistics

Exercise 1a: Using the R Environment for Statistical Computing

Supplement: **ggplot2** graphics

*D G Rossiter
Cornell University*

April 24, 2015

Contents

1 Introduction	1
2 Exploratory graphics	2
2.1 Univariate exploratory graphics	2
2.2 Multiple geometries in one plot	8
2.3 Adding to a plot	9
2.4 Bivariate exploratory graphics: faceting by classifying factors	11
2.5 Bivariate exploratory graphics: scatterplots	13
3 Beyond the quick plot	16
3.1 Colours and palettes	18
3.2 Symbol size and type	21
3.3 Controlling the axes	25
3.4 Faceting	26
4 Displaying polygons	28
5 Using open-source maps	28
References	31
Index of R concepts	32

Version 1. Copyright © D G Rossiter. All rights reserved. Reproduction and dissemination of the work as a whole (not parts) freely permitted if this original copyright notice is included. Sale or placement on a web site where payment must be made to access this document is strictly prohibited. To adapt or translate please contact the author (dgr2@cornell.edu).

子曰：工欲善其事，必先利其器
- 《论文, 15:10 (卫灵公)》，孔子
“If a workman wishes to do a good job, he must first sharpen
his tools.”
- Confucious, Analects 15:10

1 Introduction

This exercise is a supplement to Exercise 1 “Using the R Environment for Statistical Computing”. It introduces a different graphics system for R, the “grammar of graphics” approach of Wilkinson [3], as implemented in the `ggplot2` R package of Wickham [2]. The cited textbooks give full details on the grammar [3] and its R implementation [2]; the `ggplot2` website [1] is also a useful reference. Here we just repeat and extend the graphics from Exercise 1, to show how `ggplot2` provides an intuitive grammar to produce excellent statistical graphics.

Note: The code in these exercises was tested with `knitr` package Version: 1.8 [4] on R version 3.1.1 (2014-07-10), `sp` package Version: 1.0-19 and `ggplot2` package Version: 1.0.0, running on Mac OS X 10.10.3. So, the text and graphical output you see here was automatically generated and incorporated into \LaTeX by running the code through R and its packages. Then the \LaTeX document was compiled into the PDF version you are now reading. Your output may be slightly different on different versions and on different platforms.

TASK 1 : Load the `meuse` example dataset of the `sp` package. •

```
> require(sp)
> data(meuse)
```

TASK 2 : Load the `ggplot2` package. •

```
> require(ggplot2)
```

We begin with the `qplot` function of `ggplot2`; this can also be written `quickplot`, which explains its function. This is analogous to the base graphics `plot` function, but is much more powerful, since it uses a consistent grammar to allow many kinds of plots. We will explain its features as we go along. In a later section we show the more sophisticated and powerful functions.

The grammar considers the following aspects of a graphic:

- data : the dataset containing the variables to visualize;
- mapping : how the data are mapped to aesthetics, i.e., what the viewer can perceive;
- geom : shorthand for “geometries”, the geometric objects that are displayed to the viewer, e.g., points, lines, or polygons;

- scale : how values in data space are mapped to aesthetic space, i.e., how the value of a variable is represented. This could be by its colour, by its size, by its shape, etc.;
- coord : shorthand for “coördinate system”, how data coordinates (1D, 2D etc.) are mapped to the 2D plane of the graphic; this system is shown to the reader by axes and gridlines; examples are Cartesian, polar, and map projections; this also controls plot aspects, i.e., scale ratios;
- stat : shorthand for “statistical transformations”, which summarize the data, rather than show all of it; examples are binning for histograms or a smooth curve for a bivariate relation;
- facet : shorthand for “faceting”, i.e., how to divide the data into groups (subsets); this is also called “conditioning”; this is presented in §2.4

The idea here is that the different logical features of a graphic are separated:

1. What is the data to present?
2. How can the viewer see these?
3. What objects are shown?
4. How are these shown?
5. How are the dimensions of the data shown on the necessarily 2-D graphic?
6. Are the data transformed or summarized before presentation?
7. Are the data presented in subsets, on multiple plots?

This allows a very powerful and flexible approach.

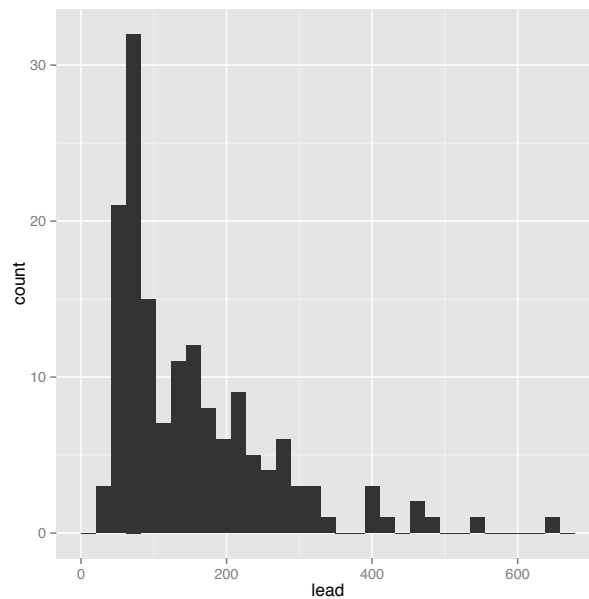
2 Exploratory graphics

2.1 Univariate exploratory graphics

TASK 3 : Visualise the **frequency distribution** of lead content in the soil samples with a frequency (count) histogram. •

We know that a histogram shows the distribution; however with the default behaviour of `qplot` function we don't have to explicitly ask for a histogram, all we need to do is specify that a single variable (here, lead) is to be plotted. The function uses reasonable defaults: the best representation for one variable is a histogram.

```
> qplot(lead, data=meuse)
stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```



Notice how the bin width defaulted to 1/30th of the data range:

```
> diff(range(meuse$lead))/30
[1] 20.56667
```

Behind the scenes `qplot` is making a lot of assumptions like this. This simplest call to `qplot` function only specified the data grammar to find the variable (here the `lead` field of the `meuse` dataframe), and `qplot` did the rest. Here, since the supplied data is a single continuous variable with no grouping factor, the default `geom` is "histogram", the default statistical transformation is histogram binning, and the default coordinate system is then 2D: data values on the x-axis and count on the y-axis. For the histogram, a bin width must be calculated, because we didn't supply one. Further, a histogram has default labelling of the x-axis (value of the variable and its name) and y-axis (count in the bin). These defaults can all be found with `?qplot`.

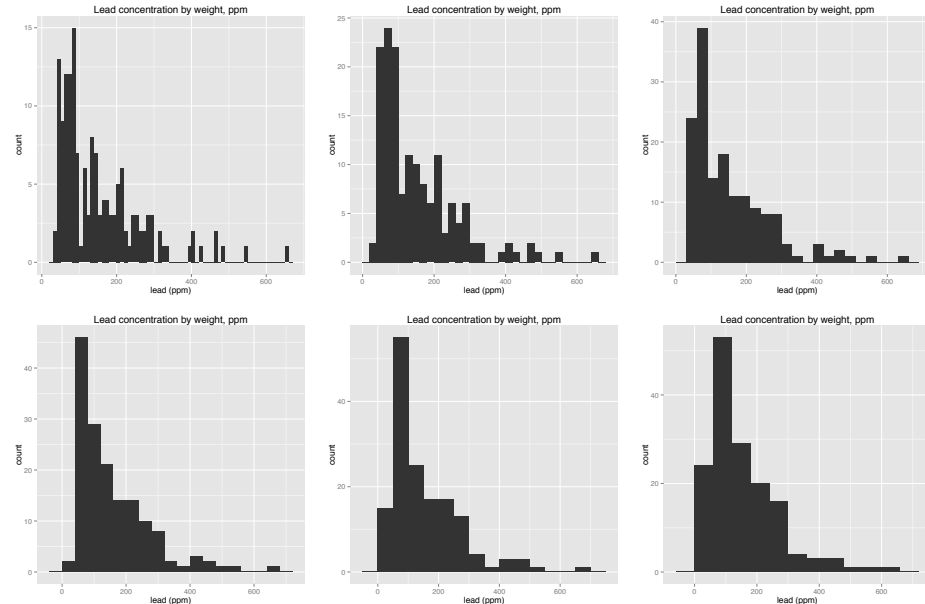
The defaults are reasonable, and allow for quick creation of useful graphics. Details of the appearance can be changed by modifying some defaults, for example the axis labels and the bin width of a histogram.

TASK 4 : Display the histogram with binwidths of 10, 20, 30, 40, 50 and 60 mg kg⁻¹ lead. Add an x-axis label and a graph title. •

The bin width is specified with the `binwidth` argument to `qplot`; this is in fact an argument to the underlying histogram geometry function `geom_histogram`. We also make the geometry explicit, by using argument `geom`. The `xlab`, `ylab`, `main` and `sub` annotations have the same meaning as for base graphics `plot`.

A sequence of plots can be produced with the `for` command, stepping through the desired bin widths.

```
> for (w in seq(10,60, by=10)) {
+   print(qplot(lead, data=meuse, geom="histogram", binwidth=w,
+             main="Lead concentration by weight, ppm",
+             sub=paste(w, "ppm bins"), xlab="lead (ppm)"))
+ }
```



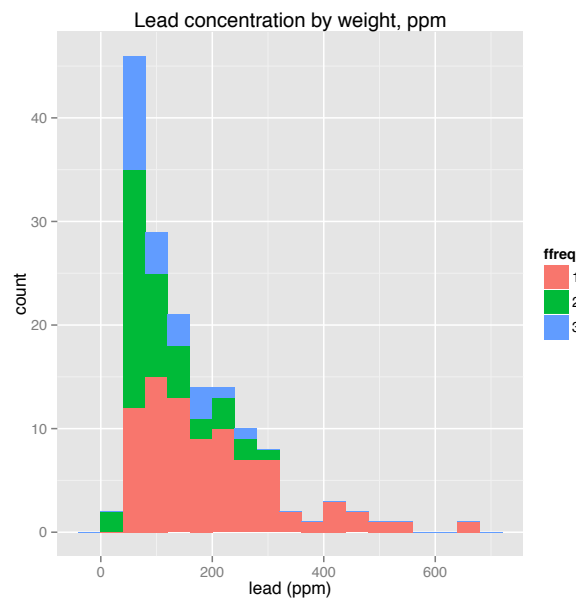
Note: The `print` function must be given explicitly inside a control structure such as a `for`-loop; only at the outer level does a function that produces a graphic, such as `qplot`, also display that graphic.

In this case all the histograms give similar information; however the narrower bins ($< 40 \text{ mg kg}^{-1}$) show a fine feature near 80 mg kg^{-1} that is obscured with wider bins. With the widest (60 mg kg^{-1}) bin the long right tail looks continuous, not interrupted as we see with the others.

TASK 5 : Repeat the histogram, but also show the flood-frequency class.

This is not yet “faceting”, because we only produce one graph; however, by specifying that the bars should be filled according to a *categorical* variable (here, `ffreq`), we can get an impression of how the univariate distribution depends on the categorical variable. The `fill` argument is shorthand for a *mapping* of a categorical variable (here, `ffreq`) to an *aesthetic* (here, the colours of the histogram bars). The `qplot` function automatically adds a plot legend showing the mapping.

```
> print(qplot(lead, data=meuse, geom="histogram", binwidth=40,
+           fill=ffreq, main="Lead concentration by weight, ppm",
+           xlab="lead (ppm)"))
```

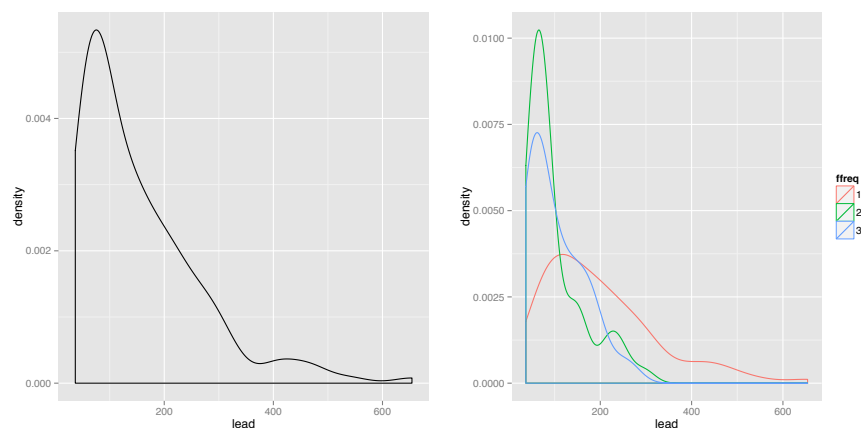


There are other aesthetics to represent a univariate distribution.

TASK 6 : Visualise the frequency distribution of lead content in the soil samples with a density plot, both for all observations together and with a mapping from flood frequency to colour. •

This is now a different geometry than the default, so we make explicit reference to the desired geometry, here `geom_density`, which as an argument to `qplot` is just written `density`. In the second example, the `colour` argument (which can also be written `color`, naming a categorical variable as the “colour”, is shorthand for a mapping from a categorical variable to an aesthetic. Since there is no way to show categories in a single density, the densities must be shown per-category.

```
> qplot(lead, data=meuse, geom="density")
> qplot(lead, data=meuse, geom="density", colour=ffreq)
```

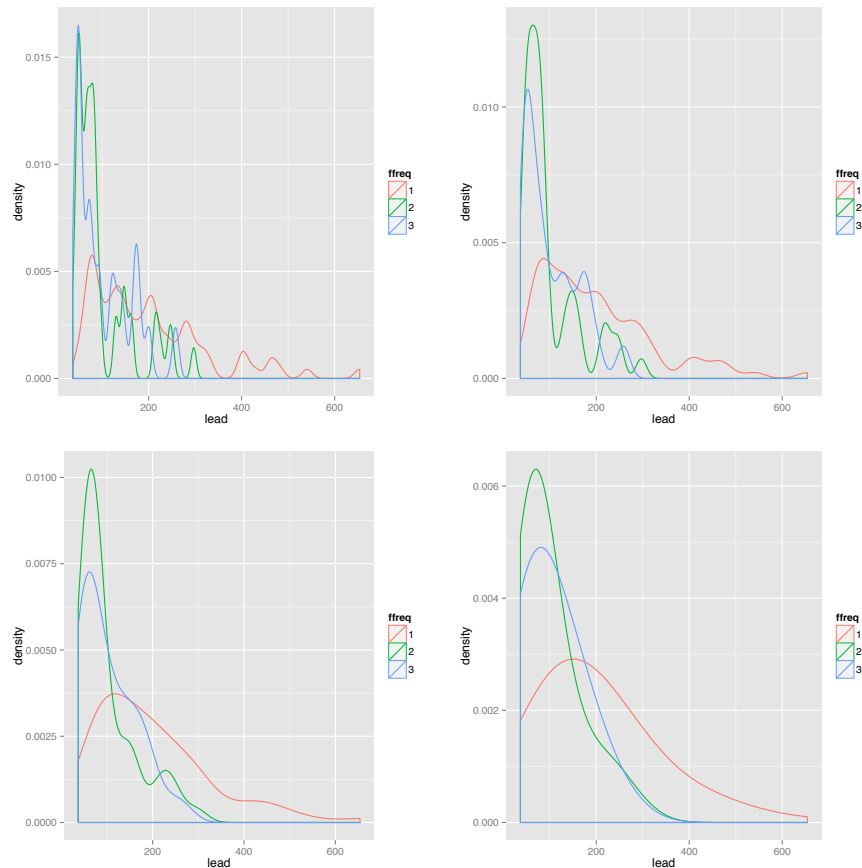


Notice how the vertical (density) scale is adjusted so that the area under the curve is 1.

The analogy of a histogram binwidth for density plots is the degree of smoothness; this is specified with the `adjust` argument; the higher the value the smoother the plot.

Note: The `adjust` argument is actually an argument to the `stat_density` density estimate function; this is in the “statistics” group.

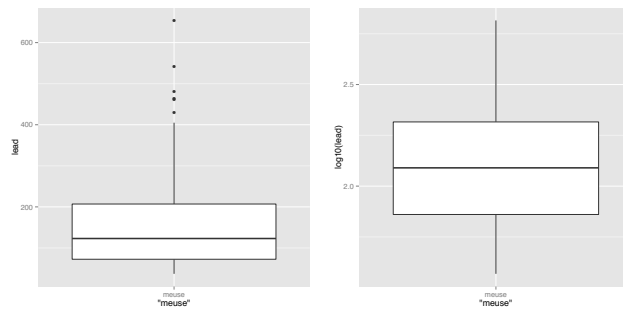
```
> for (w in c(0.25, 0.5, 1, 2))
+   print(qplot(lead, data=meuse, geom="density",
+             colour=ffreq, adjust=w))
```



TASK 7: Visualise the quantiles of distribution of the lead concentration with a box plot; repeat for the base-10 log of the lead concentration. •

The boxplot is a different geometry, `geom_boxplot`. So we again specify one continuous variable, but with this geometry.

```
> qplot(x="meuse", y=lead, data=meuse, geom="boxplot")
> qplot(x="meuse", y=log10(lead), data=meuse, geom="boxplot")
```

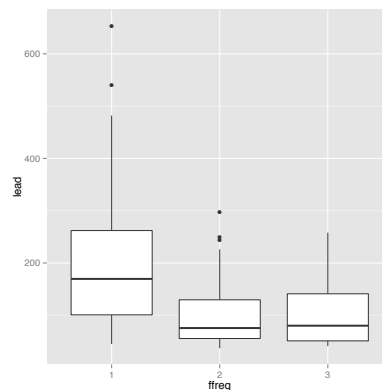


This strange syntax, i.e., the need to specify "meuse" as the first argument, is because the boxplot of a continuous variable (here, the y variable) is considered to be **conditioned** on a classifying variable (the x variable); in the univariate case there isn't one, so we specify a dummy, here a character string. The more natural use of the boxplot is indeed to examine the distribution, conditioned on a classifying variable; we show an example now.

TASK 8: Visualise the quantiles of distribution of the lead concentration with a box plot, conditioned by flood frequency class. •

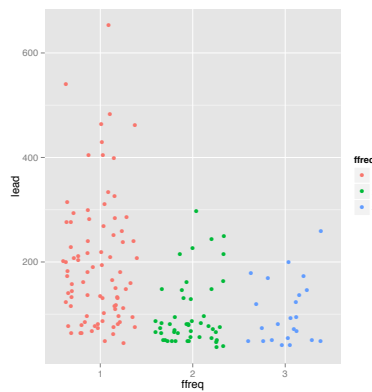
With `geom_boxplot` geometry the x-axis is a categorical variable: there will be one boxplot for each category. The other axis is the continuous variable to be shown in the boxplot.

```
> qplot(x=ffreq, y=lead, data=meuse, geom="boxplot")
```



Another way to visualize a distribution is with the `geom_jitter` geometry. This shows *all* the values, not just the five-number summary and any outliers as in the boxplot. To do this, it uses the second dimension to show all the points at that particular value of the continuous variable.

```
> qplot(x=ffreq, y=lead, data=meuse, geom="jitter", colour=ffreq)
```

Note how we also used `colour` to also colour the jittered points; that could have been used on the boxplot also.

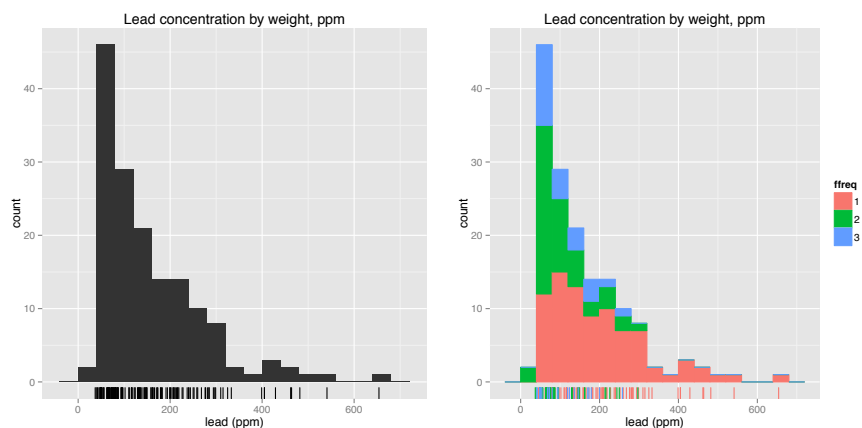
2.2 Multiple geometries in one plot

The `qplot` function can plot multiple compatible geometries on a single plot, by specifying a *list* of geometries as the `geom` argument. The list is built as usual, using the `c` function. By “compatible” we mean that the two geometries represent the same data but with different mappings.

TASK 9 : Plot the univariate distribution of Pb concentration as a histogram, with a “rug” plot on the x-axis to show actual values. Also show the flood frequency classes. •

Both the histogram and rug plot have the same x-axis (the Pb concentration); the histogram then maps this to bars of variable heights covering some range of Pb, whereas the rug plot places a small vertical bar at each actual Pb concentration.

```
> print(qplot(lead, data=meuse, geom=c("histogram", "rug"), binwidth=40,
+       main="Lead concentration by weight, ppm",
+       sides="b", xlab="lead (ppm)"))
> print(qplot(lead, data=meuse, geom=c("histogram", "rug"), binwidth=40,
+       main="Lead concentration by weight, ppm",
+       fill=ffreq, colour=ffreq,
+       sides="b", xlab="lead (ppm)"))
```

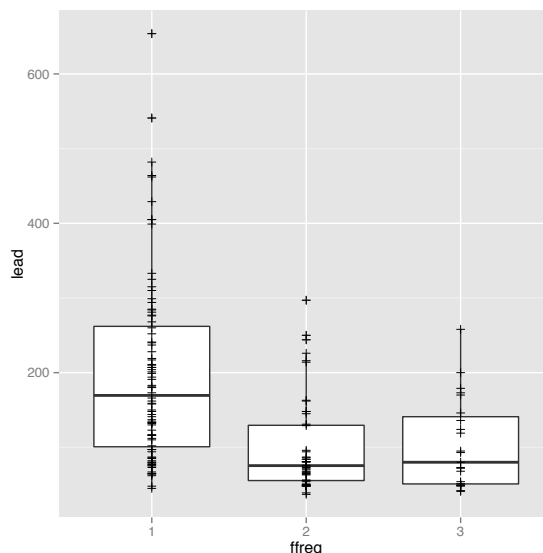


A compatible geometry for a boxplot are the points giving the actual values, this time on the y-axis.

TASK 10 : Make a boxplot of Pb concentration, conditioned on the flood frequency class, with the actual values shown as tick marks. •

We take this chance to introduce the first *aesthetic*: the shape of a symbol. This is similar to the `pch` argument in base graphics. The `shape` argument is shorthand for the `shape` argument to the `aes` “aesthetics” function (see below). There are 25 symbols; we pick the third, a cross, which will look like a tick mark on the vertical boxplot axis. We have to use the `I` “as is” function so that the argument 3 is interpreted as the number 3 and not as a variable.

```
> qplot(x=ffreq, y=lead, data=meuse, geom=c("boxplot", "point"),  
+       shape=I(3))
```



2.3 Adding to a plot

A major feature of `ggplot2` is that plots are built up in *layers*. In the simple use of `qplot` we did not use this; here we show examples of how additional elements can be added to a plot.

TASK 11 : Enhance the histogram with a title, axis labels, coloured histogram bars, and a rug plot. •

This is the first example of creating a plot, adding to it, and then displaying:

1. One call to `qplot` is used to specify a histogram; its aesthetic elements are specified as optional arguments. This call includes explicit reference to the desired geometry, even though it would be automatically selected.

The `<-` assignment operator is used to save the results of the `qplot` function as graphics objects; the result of an assignment expression (right-hand side `<-`) is not displayed.

2. The object is modified by adding the histogram bars with the `geom_bar` function. Although the default would have produced bars in the histogram, calling it explicitly allows for modification of the bar specifications. Here we specify a colour and bin width.

Note the use of the `+` operator to add to a plot. This operator is overloaded, and when both arguments can be interpreted by `ggplot2`.

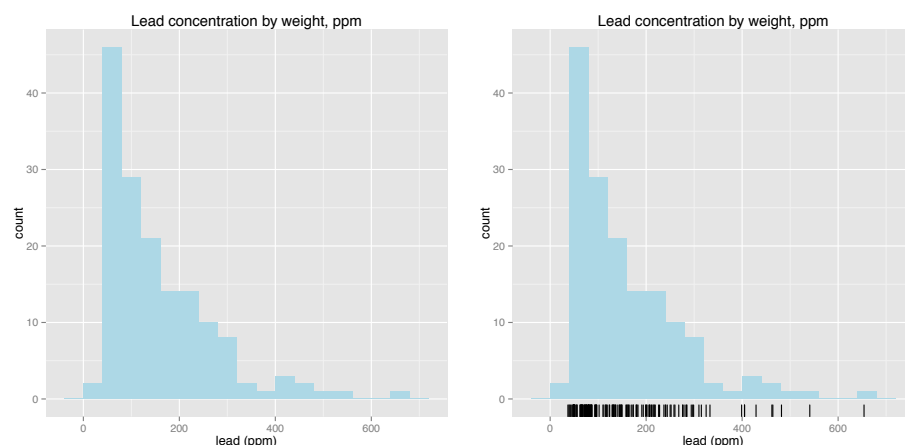
3. The rug plot is added to the histogram. This shows a major feature of the graphics grammar: the use of **low-level** graphics functions to build up plots layer-by-layer.
4. The plotting object is now displayed with an explicit call to the `print` function.
5. The `last_plot` function recalls the previous plot, to which we add some further graphic elements, in this case with a low-level function `geom_rug` to specify that the *same* data from the previous plot should now be displayed with an *additional* geometry, i.e., a rug. This is the same 1D data, we're just seeing it two ways on the same plot.

Since this expression is not assigned to an object, its results, i.e., a plot, are immediately displayed.

```
> tmp <- qplot(lead, data=meuse, geom="histogram", binwidth=40,
+             main="Lead concentration by weight, ppm",
+             xlab="lead (ppm)")
> class(tmp)

[1] "gg"      "ggplot"

> tmp <- tmp + geom_bar(fill="lightblue", binwidth=40)
> print(tmp)
> last_plot() + geom_rug()
```



2.4 Bivariate exploratory graphics: faceting by classifying factors

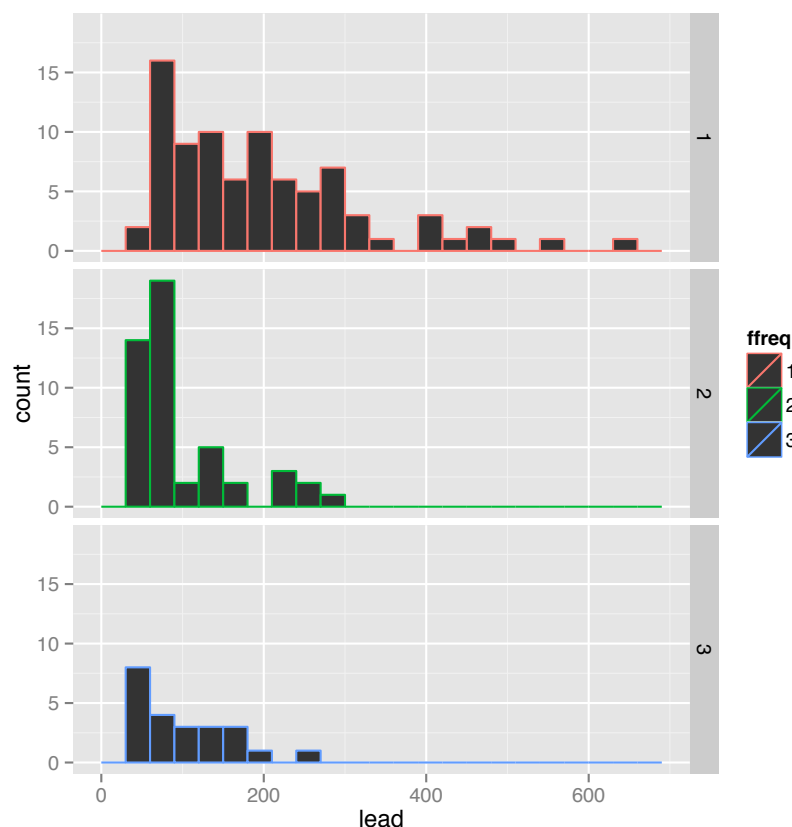
In the previous sections we showed a continuous variable grouped by a classifying factor on one plot. Another approach is to use what **ggplot2** terms **faceting**, to split the variable according to the factor and show separate plots for each subset.

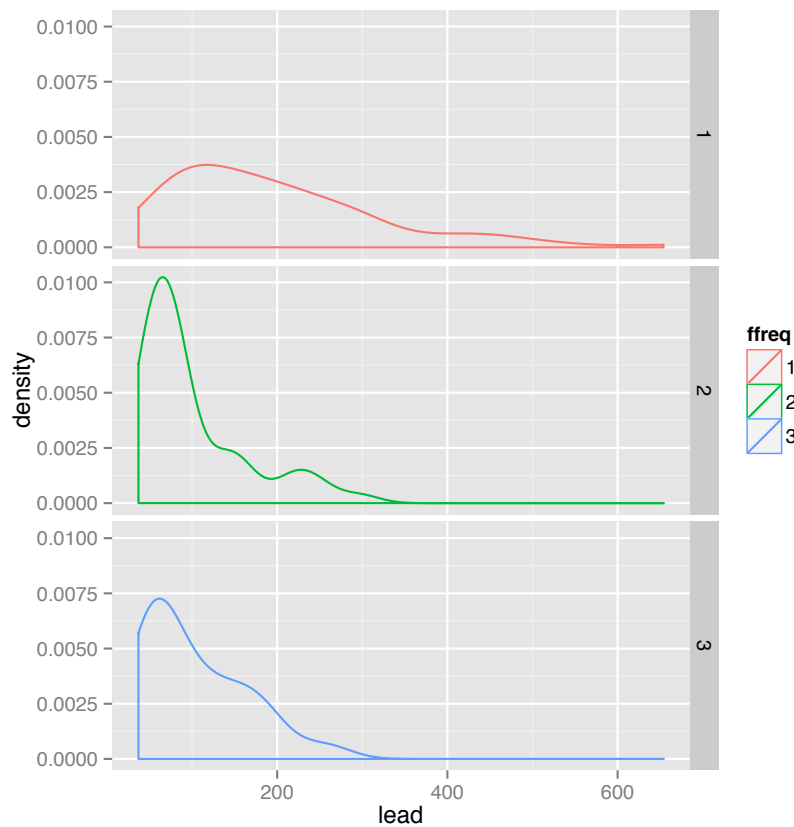
The default faceting in **qplot** is to arrange the facets on a grid, using an R formula `row_variable_name ~ col_variable_name`. If there is only one faceting variable, the “missing” one is specified as the dummy variable name “.”.

TASK 12 : Display histograms and density plots for the lead concentration, faceted by flood frequency class, as rows. •

To show these as rows we specify the column in the formula with the dummy variable “.”.

```
> qplot(lead, data=meuse, facets=ffreq ~ .,
+       geom="histogram", binwidth=30, colour=ffreq)
> qplot(lead, data=meuse, facets=ffreq ~ .,
+       geom="density", adjust=1, colour=ffreq)
```



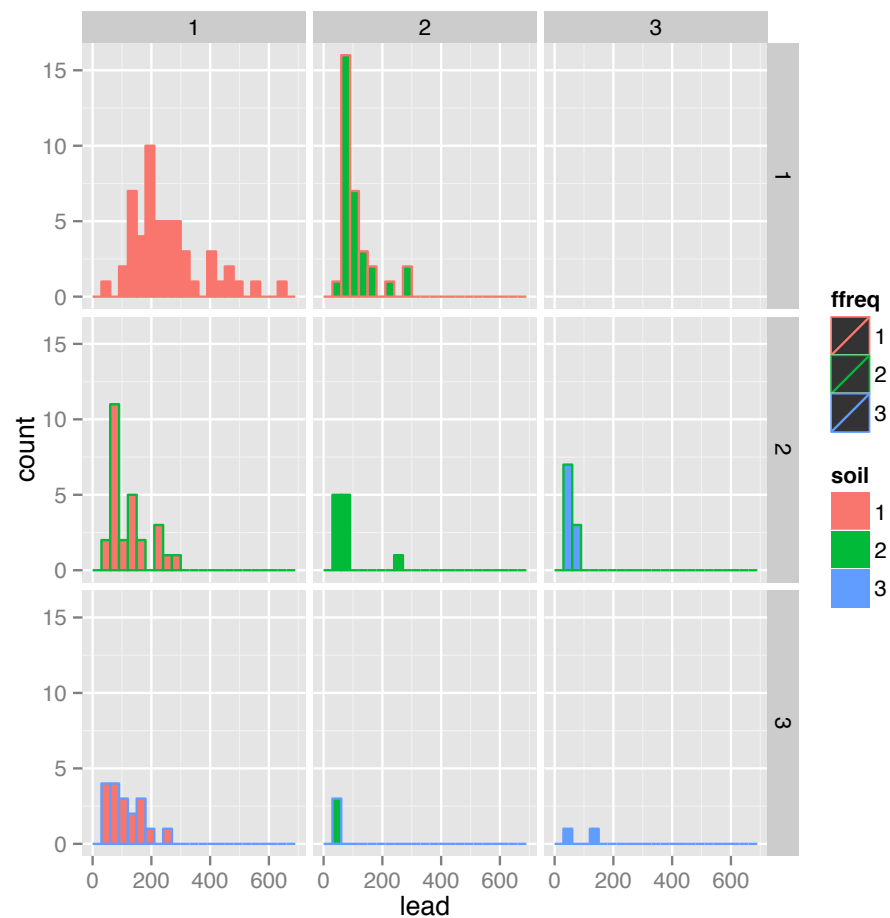


Notice that `qplot` uses the same axis scales for all subgroups; this allows easy comparison. Here we clearly see that the most frequently-flooded soils have all the high values, but that all three classes have skewed distributions. Here the use of colour is optional, since each group is already a separate facet.

We can show a matrix of histograms, classified two ways:

TASK 13 : Display histograms and density plots for the lead concentration, faceted by flood frequency class, as rows, and by soil type, as columns. Colour both for easy interpretation. •

```
> qplot(lead, data=meuse, facets=ffreq ~ soil, geom="histogram",
+       colour=ffreq, fill=soil, binwidth=30)
```



Here we use the `colour` argument to show the flood frequency, and the `fill` argument to show the soil type. These are both mappings of values to aesthetics.

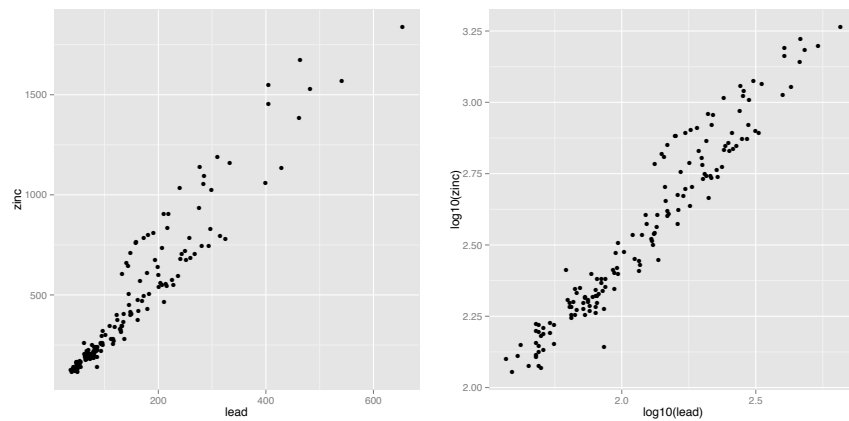
2.5 Bivariate exploratory graphics: scatterplots

The principal tool is the scatterplot.

TASK 14 : Display a scatterplots of Zn concentration (y-axis) vs. Pb concentration (x-axis), both untransformed and log-transformed. •

If `qplot` is presented with two variables, by default it produces a scatterplot. The variables can be the result of expressions, for example `log10`.

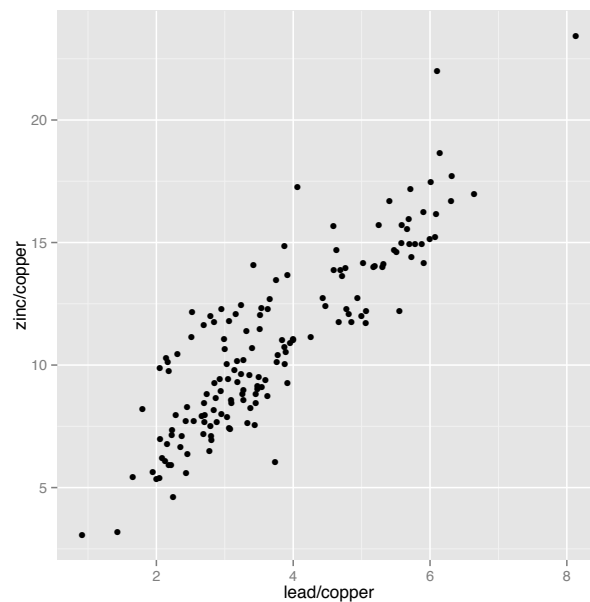
```
> qplot(x=lead, y=zinc, data=meuse)
> qplot(x=log10(lead), y=log10(zinc), data=meuse)
```



The expressions can be any valid arithmetic expression involving the variables in the named data frame.

TASK 15 : Plot the ratios of the two metals standardized by the copper content,

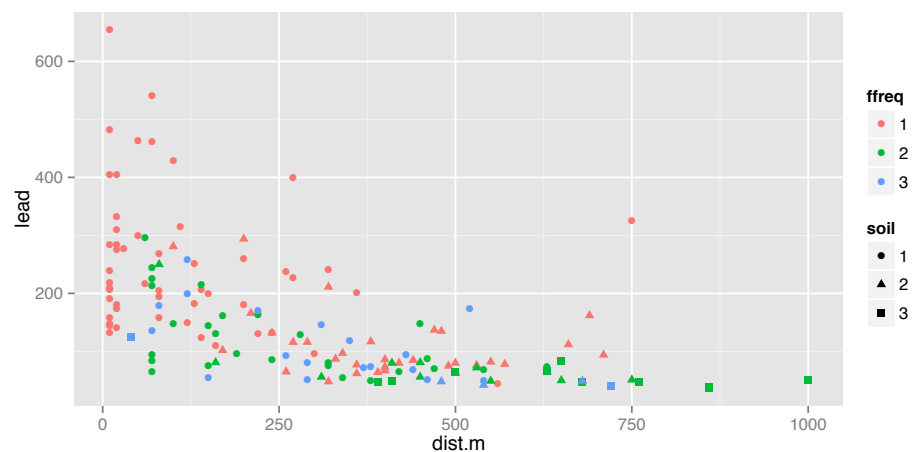
```
> qplot(x=lead/copper, y=zinc/copper, data=meuse)
```



As with the univariate plots, the points can be coloured by some classifying factor; they can also have a different symbol for a (possibly different) factor.

TASK 16 : Plot the Pb concentration against the distance to the nearest river, with the points coloured by the flood frequency class and their shape representing the soil type.

```
> qplot(x=dist.m, y=lead, data=meuse, colour=ffreq, shape=soil)
```



There are several ways to visualize a 2D relation; the scatterplot is the most common. `qplot` can be called with different geometries; we've already seen the default `geom_points` geometry, which produces a scatterplot. Another frequently-used geometry is `geom_smooth`; this fits an empirical smoother and displays the smoother and its standard error. There are various smoothing methods which can be specified with the `method` argument; the default "loess" smooth local regression is the default.

TASK 17 : Add an empirical smoother to the plot of Pb concentration against the distance to the nearest river. •

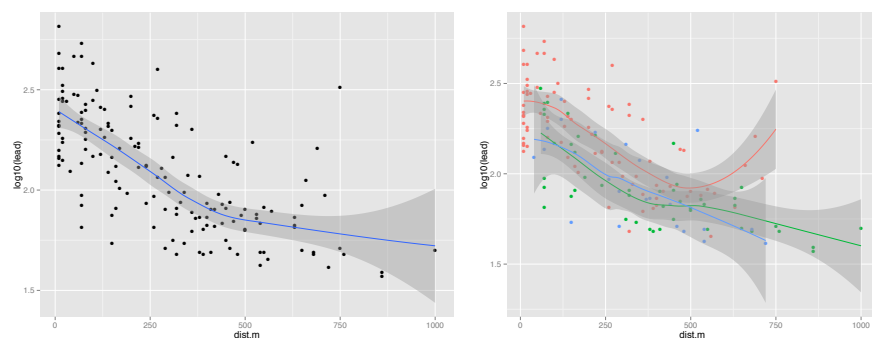
Several geometries can be provided together to the `geom` argument to `qplot`, joined into a list with the `c` "catenate" function. We do this first for all points, and then by flood frequency class.

```
> qplot(x=dist.m, y=log10(lead), data=meuse, geom=c("point", "smooth"))
```

geom_smooth: method="auto" and size of largest group is <1000, so using loess. Use 'method = x' to change the smoothing method.

```
> qplot(x=dist.m, y=log10(lead), data=meuse, colour=ffreq, geom=c("point", "smooth"))
```

geom_smooth: method="auto" and size of largest group is <1000, so using loess. Use 'method = x' to change the smoothing method.

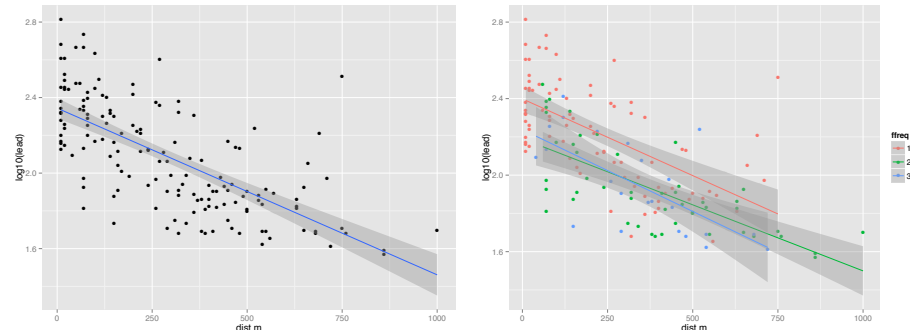


If you suspect a linear or polynomial relation over the whole range, you can fit a linear model with the "lm" "linear model" method:


```

> qplot(x=dist.m, y=log10(lead), data=meuse,
+       geom=c("point", "smooth"), method="lm")
> qplot(x=dist.m, y=log10(lead), data=meuse, colour=ffreq,
+       geom=c("point", "smooth"), method="lm")

```



3 Beyond the quick plot

To enjoy the full power of the grammar of graphics, we introduce the concept of building up a graphic by **layers**. The idea here is that a figure can be built up in layers – this is a natural way to think about graphics. For example, we can think of first drawing the axes of a scatterplot, then the points, then colouring the points according to some category, then adding some summary such as density contours, and so forth. These are all layers, each of which have a grammar that describes them and their options.

In outline, the full specification of one layer of a graphic is:

```

> ggplot(data, mapping) +
+   layer(
+     stat = "",
+     geom = "",
+     position = "",
+     geom_params = list(),
+     stat_params = list(),
+   )

```

where **ggplot** initializes the plot and supplies the data source and the **layer** function lists the specification of one layer. In practice, there are various convenient abbreviations.

As an example, the scatterplot of Pb vs. distance from river, points coloured by flood frequency and with a size proportional to their organic matter content, and with a best-fit linear regression line, can be displayed by (1) initializing the plotting system with a call to **ggplot** without any arguments, (2) adding layers, each separated with the + operator':

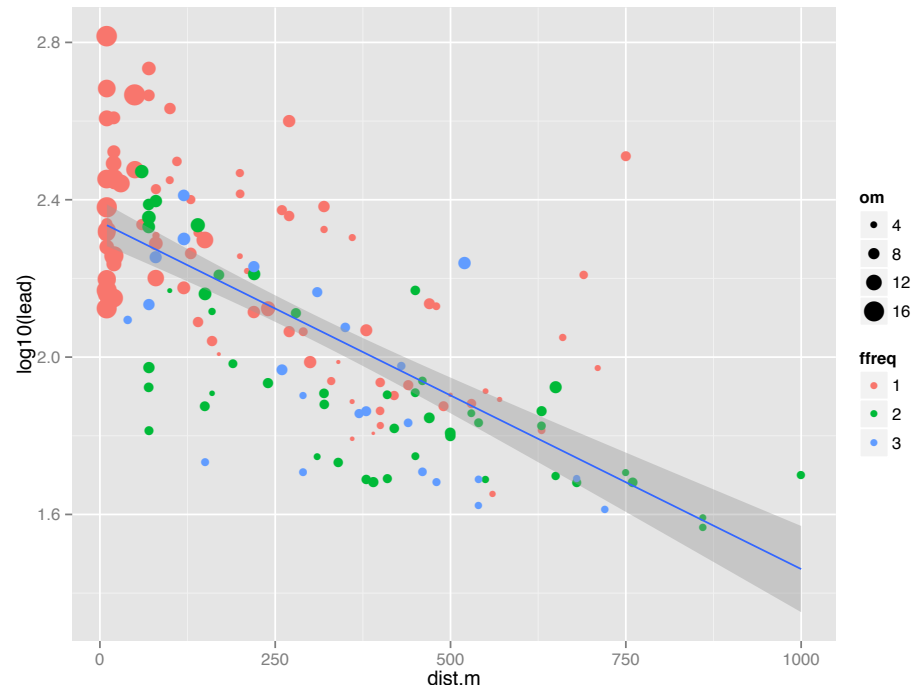
```

> ggplot() +
+   geom_point(
+     aes(x=dist.m, y=log10(lead), colour=ffreq, size=om),
+     data=meuse
+   ) +
+   geom_smooth(
+     aes(x=dist.m, y=log10(lead)),
+     data=meuse,

```

```
+ method="lm"
+ )
```

Warning: Removed 2 rows containing missing values (geom_point).



The call to the `ggplot` function, without any arguments, just creates a skeleton for layers to be added; it is referred to as the “base” layer. The `geom_point` function builds a point layer as specified; similarly for `geom_smooth`. The `+` operator adds a layer to an existing plot. There is no limit to the number of layers, but they must be compatible. In the example the point scatterplot (two axes) is compatible with a smooth line also in 2D. Each of the geometry functions has its own required and optional arguments. All of them require the `data` argument – we can’t plot anything without knowing the data!

Here we see that both require an `aes` “aesthetic” argument. Recall, this specifies an **mapping** from **values** of the variables to **visual properties**, also known as “aesthetics”, within the specified geometry.

In the first layer of this example, we know that some data in the `meuse` dataset should be represented as points on a scatterplot – this from the definition of the `geom_point` geometry.

1. The `x` and `y` arguments specify which variables, or transformations of these, should be mapped to the `x`- and `y`-axes; these are required to place the points.
2. The `colour` argument is optional; for the point geometry it specifies the colours of the points. This is another visual variable (“aesthetic”) – the first two are the geometric position of the point. Here we name another variable in the dataframe, a categorical variable.

This has three values, so we have three colours; the mapping is from three classes to three colours.

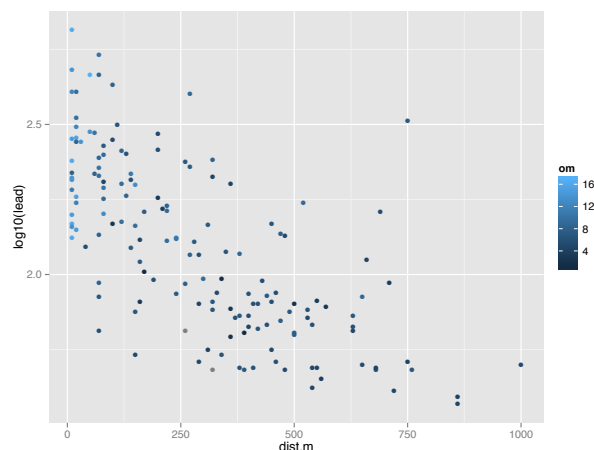
3. The `size` argument is also optional; for the point geometry it specifies the size of the points. This is another visual variable. Here we name another variable in the dataframe, this time a continuous variable (organic matter), and the mapping then scales the point size accordingly. By default the legend shows four levels.

Note that although the points are coloured by flooding frequency, the linear model is for all the points together.

3.1 Colours and palettes

It is also possible to specify a continuous variable as the `colour` argument, in which case the values are shown in a **colour ramp**. For example, we may suspect that the Pb content is also influenced by organic matter; we can include it in the plot as a colour ramp, while still showing the Pb-distance to river relation.

```
> ggplot() +  
+   geom_point(  
+     aes(x=dist.m, y=log10(lead), colour=om),  
+     data=meuse  
+   )
```



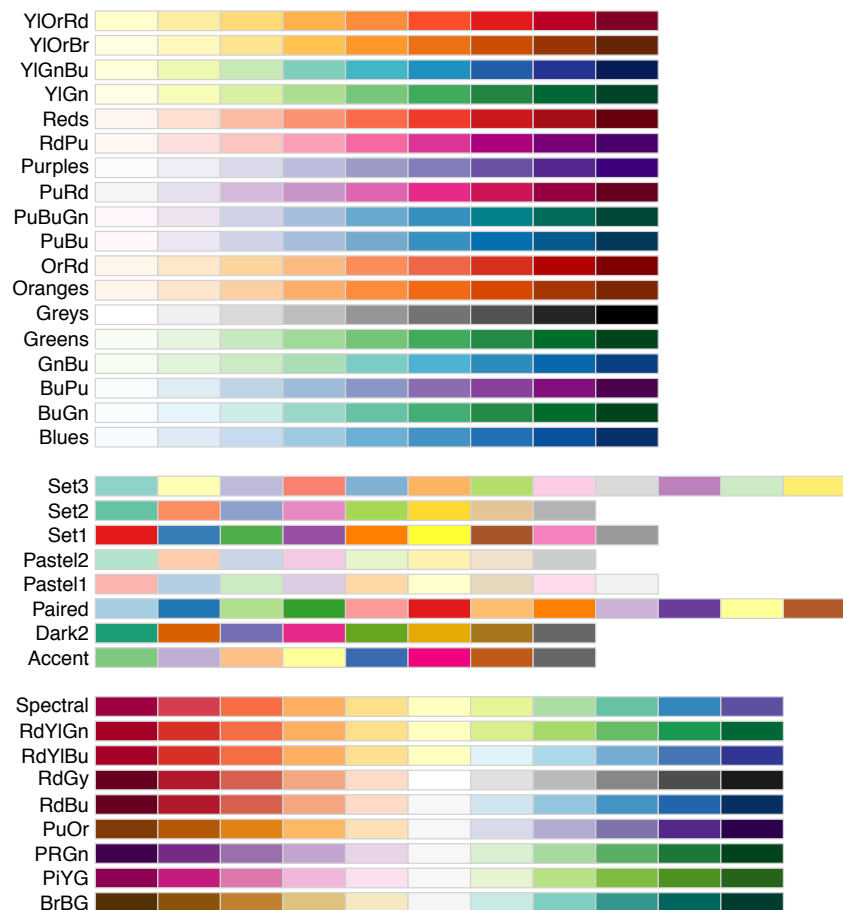
Obviously, there is a default colour ramp – here shades of blue. This is actually a **scale**, in `ggplot2` terminology. It maps numbers (here, organic matter content) to a scale in colour space, in the same way there is a mapping from numbers to a position on an axis of a scatterplot. It is a **function** from a **domain** (data values, which can be classes or continuous) to a **range** (the colour for that value). This is a quite tricky – the user's visual perception must match the change in colour. One package that has dealt with this is `RColorBrewer`, which palettes (colour choices) can be accessed with the `scale_colour_brewer` function for categorical variables and `scale_colour_distiller` for continuous variables. This package provides ready-made sequential, diverging, and qualitative palettes.

Note: The project RColorBrewerwebsite¹ allows you to experiment with palettes and then download them in a format suitable for use in R. You can then build your own palettes with `scale_colour_gradient` and similar functions in the RColorBrewer package.

TASK 18 : Load the RColorBrewer package and display all the ready-made palettes. •

We can see the ready-made palettes available in the RColorBrewer package as follows:

```
> require(RColorBrewer)
> display.brewer.all()
```

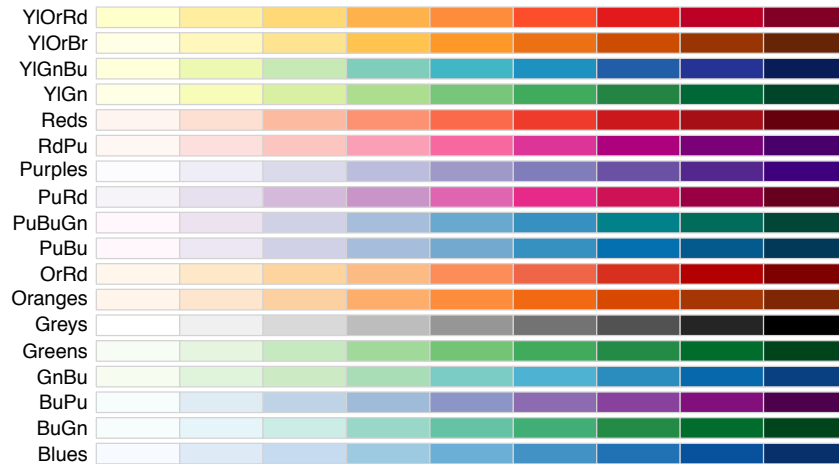


To see only the palettes appropriate for sequences, i.e., continuous vari-

¹ <http://colorbrewer2.org/>

ables, we specify the `type` argument:

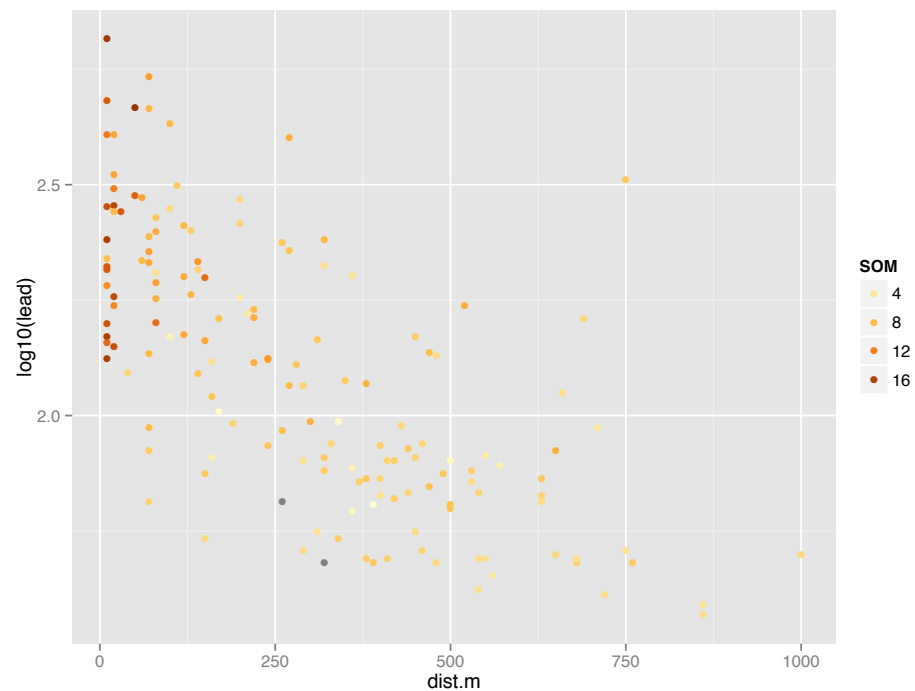
```
> display.brewer.all(type="seq")
```



TASK 19 : Select an appropriate palette and use it to colour the scatter-plot points by their relative SOM contents. •

We select one of the sequential palettes, and use the `palette` argument of the `scale_colour_distiller` function to select a palette. We also need to specify a colour space in which to calculate the gradient, i.e., colour ramp, with the `space` “colour space” argument. The function suggests the Lab colour space; this is a tristimulus (three-axis) perceptual space with lightness and two orthogonal colours **a** (green to red) and **b** (blue to yellow). The interpolation takes place in this space.

```
> ggplot() +
+   geom_point(
+     aes(x=dist.m, y=log10(lead), colour=om),
+     data=meuse
+   ) +
+   scale_colour_distiller(name="SOM",
+                           space="Lab", palette="YlOrBr")
```



3.2 Symbol size and type

Scales also apply to sizes of symbols. Again, this is a mapping, from value to some aesthetic element, here size.

TASK 20 : Display the scatterplot with symbol size proportional to SOM.

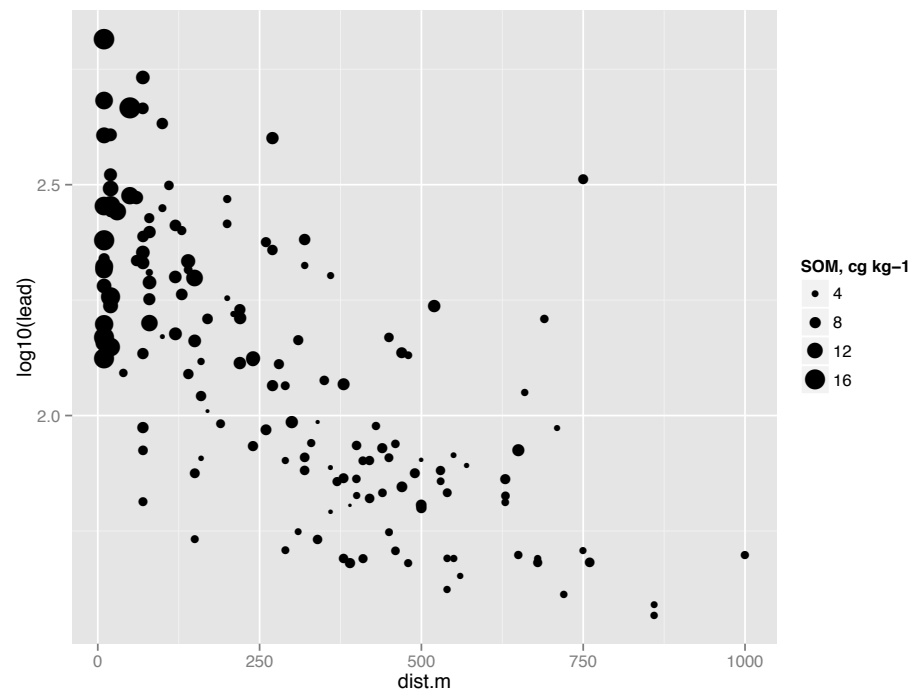
The `scale_size_continuous` function is the default for the `scale_size` function applied to a continuous variable; for discrete variables the function is `scale_size_discrete`. The `size` argument to the `aes` “aesthetic mapping” function specifies the variable that maps to the size; this is analogous to the `colour` function used in the previous example.

```
> p <- ggplot() +
+   geom_point(
+     aes(x=dist.m, y=log10(lead), size=om),
+     data=meuse
+   ) +
+   scale_size(name="SOM, cg kg-1")
> class(p)

[1] "gg"      "ggplot"

> print(p)

Warning: Removed 2 rows containing missing values (geom_point).
```



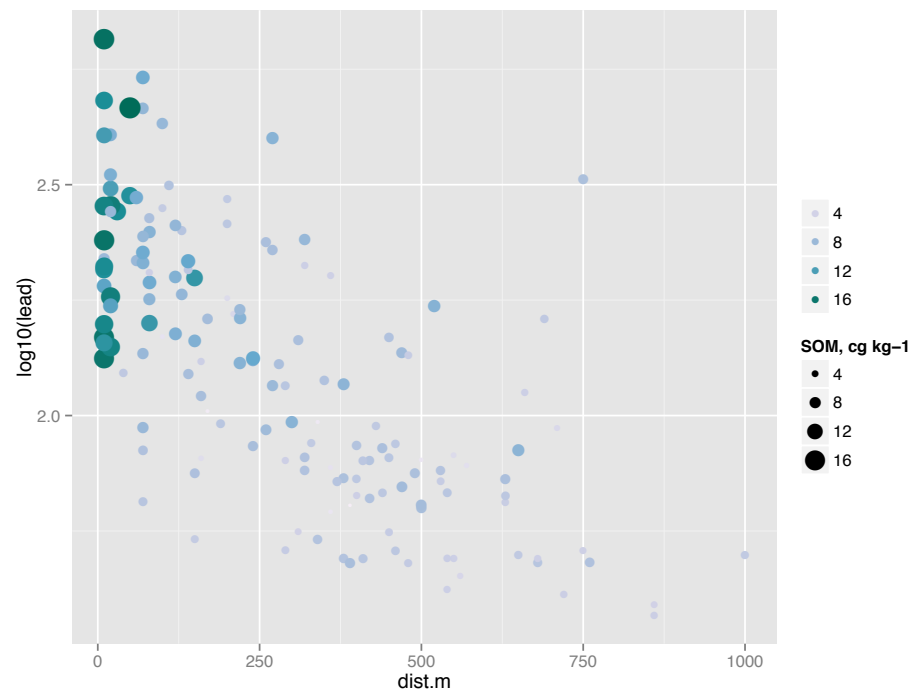
Note that in this example we save the plot as a workspace object of class `ggplot` and then explicitly print it. This is to re-use the plot with additional elements, without having to specify it in full.

TASK 21 : Display the previous plot, but also with colours representing the SOM concentration. •

We first name the saved object, but then add an additional aesthetic mapping (colour) and specify its scale. Note there is no need for a title for the color legend, since we have one already for the size legend.

```
> p +
+   aes(colour=som) +
+   scale_colour_distiller(name="", space="Lab", palette="PuBuGn")
```

Warning: Removed 2 rows containing missing values (geom_point).

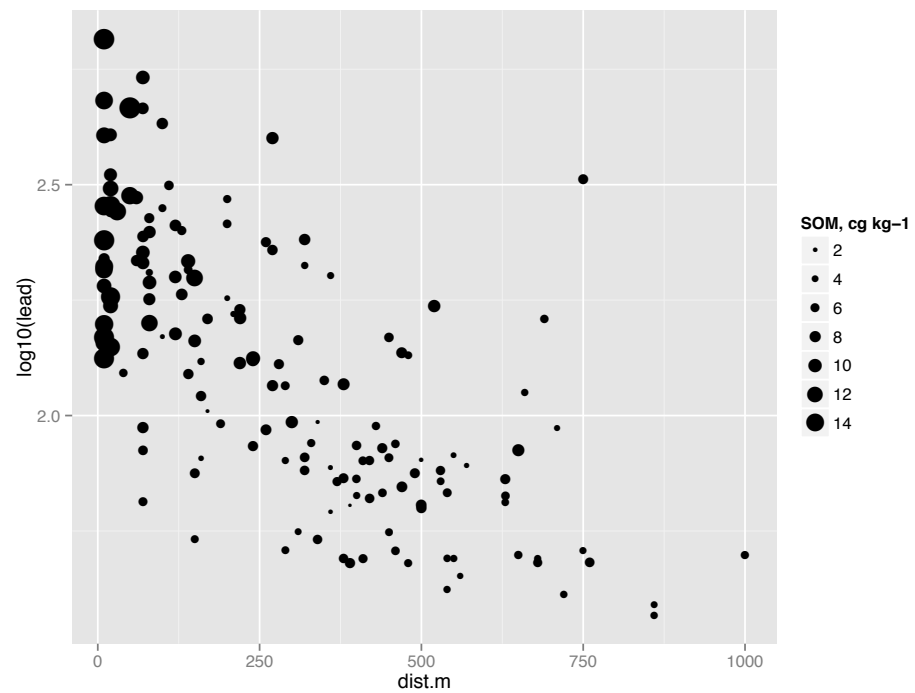


The aesthetics of continuous scales have many options, see `?continuous_scale`. For example, the number and position of legend breaks can be directly specified with the `breaks` argument.

TASK 22 : Repeat the previous graph but show a legend with SOM in classes of 2 cg kg^{-1} . •

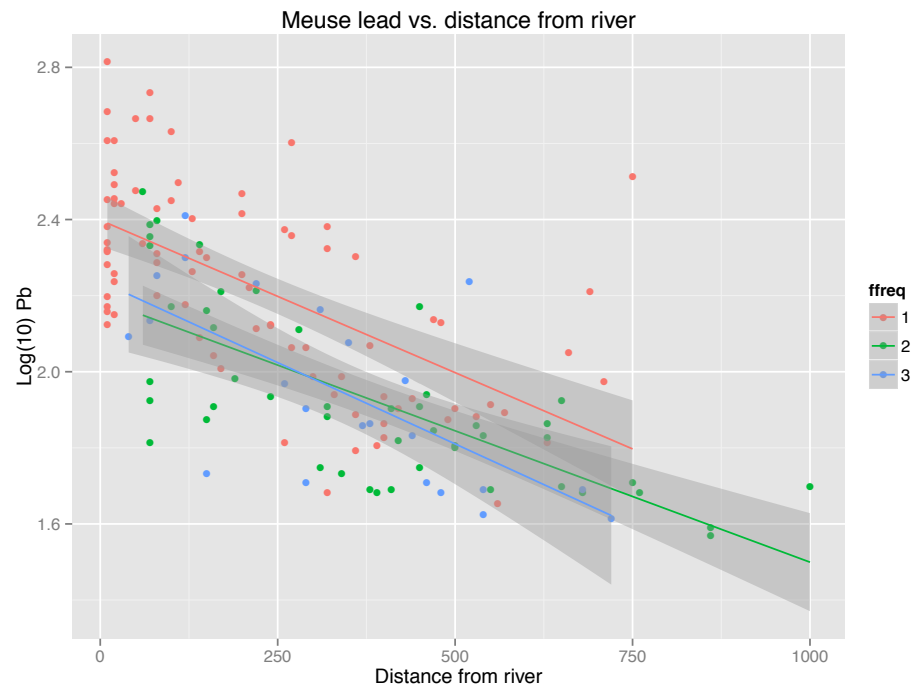
```
> ggplot() +
+   geom_point(
+     aes(x=dist.m, y=log10(lead), size=om),
+     data=meuse
+   ) +
+   scale_size(name="SOM, cg kg-1", breaks=seq(2,14,2))
```

Warning: Removed 2 rows containing missing values (geom_point).



We saw how to colour the points according to a categorical variable (flood frequency) using `qplot`. We can do the same with `ggplot`, but in a more systematic way, i.e., by building up layers.

```
> g <- ggplot(meuse, aes(x=dist.m, y=log10(lead)))
> g +
+   geom_point(
+     aes(colour=ffreq, symbol=3)
+   ) +
+   geom_smooth(aes(colour=ffreq), se=TRUE, method="lm") +
+   ggtitle("Meuse lead vs. distance from river") +
+   xlab("Distance from river") +
+   ylab("Log(10) Pb")
```

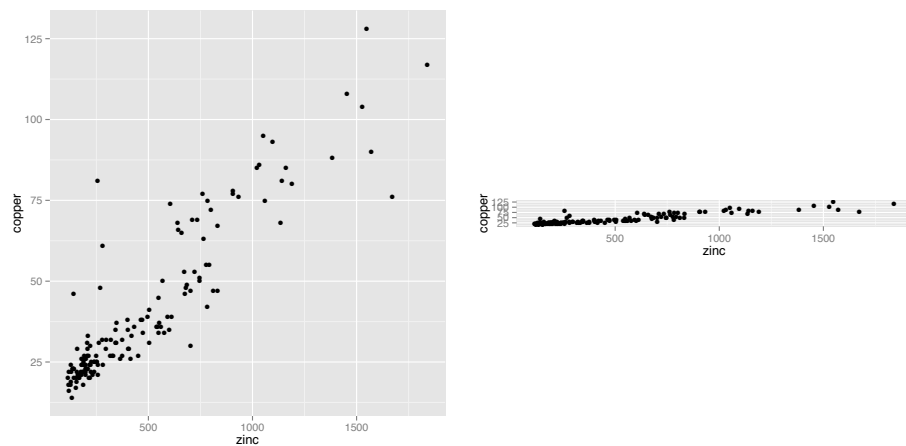


Note the use of the `ggtitle` function to add a graph title.

3.3 Controlling the axes

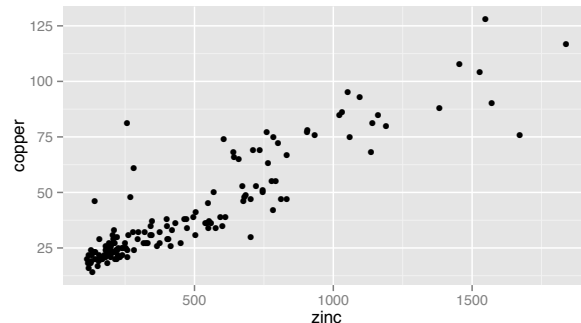
In the above scatterplots the aspect ratio of the axes was computed based on the data range, to produce a square plot. To preserve the original aspect ratio, we add a `coord_fixed` layer. We can visualize the difference in a scatterplot of lead vs. copper, which have quite different absolute values:

```
> ggplot() +
+   geom_point(
+     aes(x=zinc, y=copper),
+     data=meuse
+   )
> ggplot() +
+   geom_point(
+     aes(x=zinc, y=copper),
+     data=meuse
+   ) +
+   coord_fixed()
```



The aspect ratio is by default 1, but can be adjusted with the `ratio` argument, expressed as `y/x`:

```
> ggplot() +
+   geom_point(
+     aes(x=zinc, y=copper),
+     data=meuse
+   ) +
+   coord_fixed(ratio=8)
```



3.4 Faceting

Recall the faceting with `qplot`:

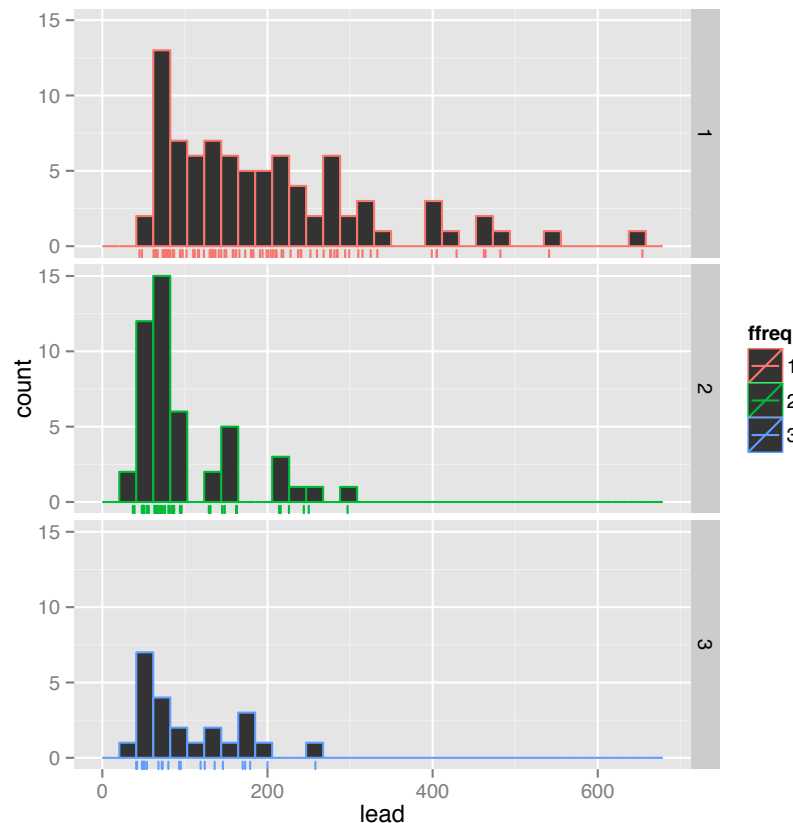
```
> qplot(lead, data=meuse, facets=ffreq ~ .,
+       geom="histogram", binwidth=30, colour=ffreq)
```

We can also facet with `ggplot`, using the `facet_grid` function to specify the form of the faceting:

```
> g <- ggplot(meuse, mapping=aes(x=lead))
> g +
+   facet_grid(ffreq ~ .) +
```

```
+ geom_histogram() + geom_rug()+
+ aes(colour=ffreq)
```

stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.



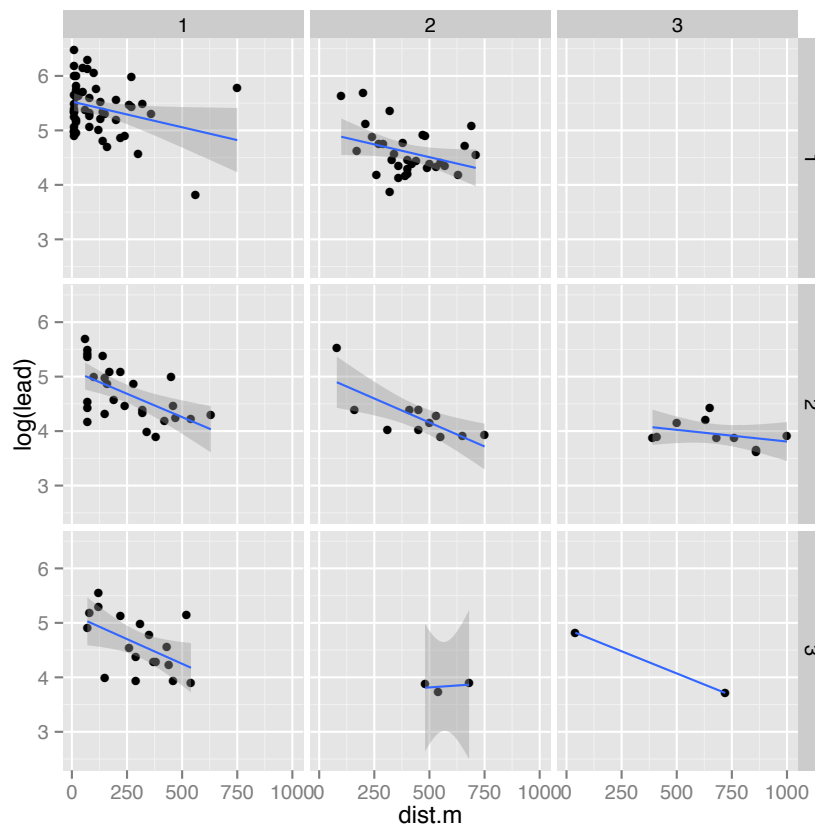
Here the mapping argument specifies how dimensions of the graphic (here, the x dimension) are mapped from fields in the data (here, the lead field). This one-dimensional mapping can be displayed many ways; the `geom_histogram` function specifies a histogram and the `geom_rug` function specifies a “rug” plot under the histogram.

Here are empirical relations between lead and distance to river, but displayed separately for each combination of flood-frequency classes and soil type.

```
> g <- ggplot(meuse, mapping=aes(y=log(lead), x=dist.m))
> g +
+ facet_grid(ffreq ~ soil) +
+ geom_point() + geom_smooth(se=TRUE, method="lm")

Warning in qt((1 - level)/2, df): NaNs produced
> aes(colour=ffreq)
```

```
List of 1
 $ colour: symbol ffreq
```



Note how both the horizontal and vertical scales are the same, allowing direct comparison of the scatterplots and lines.

4 Displaying polygons

5 Using open-source maps

The `ggmap` package provides an interface between `ggplot2` and public maps sources. It uses these maps as the base layers, to which other georeferenced information may be added.

```
> require(ggmap)
```

The base layer is obtained by the `get_map` method, from one of the public map sources; this can be further specialized to name the map source, e.g., `get_googlemap`. The services use geocoding, which may be called directly as `geocode`. The location query can be specified with any string that can be understood by Google Maps in interactive mode.

Note: Note: this will not work in the People's Republic of China or other countries that are blocking Google.

```
> geocode('71 Beijing East Road, Nanjing, China')
```

```
Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=71+Beijing+East+Road+Nanjing+China
Google Maps API Terms of Service : http://developers.google.com/maps/terms
```

```
      lon      lat
1 118.8092 32.05725
```

```

> geocode('Cornell University')

Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Cornell+Unive
Google Maps API Terms of Service : http://developers.google.com/maps/terms

      lon      lat
1 -76.4735 42.45345

> (place <- geocode('meers, netherlands', output='more'))

Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=meers,+nether
Google Maps API Terms of Service : http://developers.google.com/maps/terms

      lon      lat      type      loctype      address
1 5.740819 50.96181 locality approximate 6181 meers, netherlands
  north      south      east      west postal_code      country
1 50.96316 50.96046 5.742168 5.73947      <NA> netherlands
  administrative_area_level_2 administrative_area_level_1 locality
1      stein      limburg      meers
  street streetNo point_of_interest      query
1      <NA>      NA      <NA> meers, netherlands

> geocodeQueryCheck()

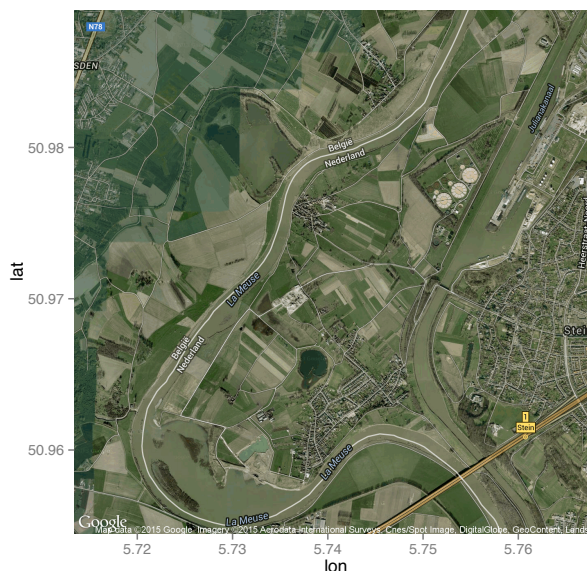
2472 geocoding queries remaining.

> meuse.map <- ggmap(get_map(
+   loc=c(lon=place$lon, lat=place$lat+0.01),
+   zoom=14, maptype="hybrid", source="google"))

Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=50.971807,5.740819&zoom=1
Google Maps API Terms of Service : http://developers.google.com/maps/terms

> print(meuse.map)

```



We recognize this as the Meuse study area.

To overlay our points, they must be transformed to the same coördinate reference system (CRS) as Google Maps.

```

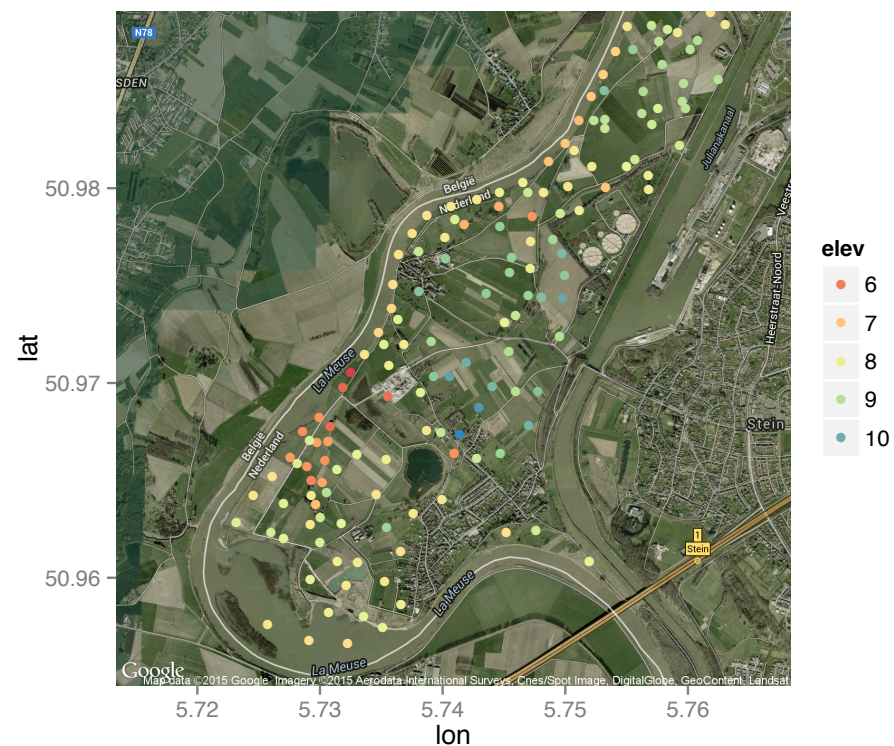
> meuse.sp <- meuse
> coordinates(meuse.sp) <- ~ x + y
> proj4string(meuse.sp) <- CRS("+init=epsg:28992") ## see ?meuse
> require(rgdal)

```

```

> meuse.wgs84 <- spTransform(meuse.sp, CRS("+init=epsg:4326"))
> meuse.wgs84.df <- as(meuse.wgs84, "data.frame")
> meuse.map +
+   geom_point(
+     aes(x=x, y=y, colour=elev),
+     data=meuse.wgs84.df) +
+     scale_colour_distiller(name="elev",
+                             space="Lab",
+                             palette="Spectral")
Warning: Removed 7 rows containing missing values (geom_point).

```



References

- [1] Hadley Wickham. `ggplot2`. <http://ggplot2.org/>. URL <http://ggplot2.org/>. 1
- [2] Hadley Wickham. *ggplot2: Elegant graphics for data analysis*. Use R! Springer, August 2009. ISBN 0387981403. 1
- [3] Leland Wilkinson. *The grammar of graphics*. Statistics and computing. Springer, New York, 2nd ed edition, 2005. ISBN 9780387286952. 1
- [4] Yihui Xie. `knitr`: Elegant, flexible and fast dynamic report generation with R, 2011. URL <http://yihui.name/knitr/>. Accessed 30-Dec-2012. 1

Index of R Concepts

+ operator, 10, 16, 17
<- operator, 10

adjust argument (qplot function), 6
adjust argument (stat_density function), 6
aes (ggplot2 package), 9, 17, 21

binwidth argument (qplot function), 3
breaks argument (continuous_scale function), 23

c, 8, 15
color argument (qplot function), 5
colour argument (aes function), 17, 21
colour argument (geom_point function), 18
colour argument (qplot function), 5, 8, 13
coord_fixed (l package), 25

data (ggplot2 package), 17
density argument (qplot function), 5

facet_grid (ggplot2 package), 26
fill argument (qplot function), 4, 13
for, 3, 4

geocode (ggmap package), 28
geom argument (qplot function), 3, 8, 15
geom_bar (ggplot2 package), 10
geom_boxplot (ggplot2 package), 6, 7
geom_density (ggplot2 package), 5
geom_histogram (ggplot2 package), 3, 27
geom_jitter (ggplot2 package), 7
geom_point (ggplot2 package), 17
geom_points (ggplot2 package), 15
geom_rug (ggplot2 package), 10, 27
geom_smooth (. package), 17
geom_smooth (ggplot2 package), 15
get_ggolemap (ggmap package), 28
get_map (ggmap package), 28
ggmap package, 28
ggplot (ggplot2 package), 16, 24, 26
ggplot (ggplot package), 17
ggplot class, 22
ggplot2 package, 1, 9-11, 28
ggtitle (ggplot2 package), 25

I, 9

knitr package, 1

last_plot (ggplot2 package), 10
layer (ggplot2 package), 16
log10, 13

main argument (qplot function), 3
mapping argument (ggplot2 function), 27
method argument (qplot function), 15
meuse dataset, 1, 17

palette argument (scale_colour_distiller function), 20
pch graphics argument, 9
plot, 1, 3
print, 4, 10

qplot (ggplot2 package), 1-5, 8-13, 15, 24, 26
quickplot (ggplot2 package), 1

ratio argument (coord_fixed function), 26
RColorBrewer package, 18, 19

scale_colour_brewer (ggplot2 package), 18
scale_colour_distiller (ggplot2 package), 18, 20
scale_colour_gradient (ggplot2 package), 19
scale_size (ggplot2 package), 21
scale_size_continuous (ggplot2 package), 21
scale_size_discrete (ggplot2 package), 21
shape argument (aes function), 9
shape argument (qplot function), 9
size argument (aes function), 18, 21
sp package, 1
space argument (scale_colour_distiller function), 20
stat_density (ggplot2 package), 6

sub argument (qplot function), 3

type argument (display.brewer.all function),
20

x argument (aes function), 17

xlab argument (qplot function), 3

y argument (aes function), 17

ylab argument (qplot function), 3