Please see the online version of this article for supplementary materials.

A Layered Grammar of Graphics

Hadley WICKHAM

A grammar of graphics is a tool that enables us to concisely describe the components of a graphic. Such a grammar allows us to move beyond named graphics (e.g., the "scatterplot") and gain insight into the deep structure that underlies statistical graphics. This article builds on Wilkinson, Anand, and Grossman (2005), describing extensions and refinements developed while building an open source implementation of the grammar of graphics for R, `ggplot2`.

The topics in this article include an introduction to the grammar by working through the process of creating a plot, and discussing the components that we need. The grammar is then presented formally and compared to Wilkinson's grammar, highlighting the hierarchy of defaults, and the implications of embedding a graphical grammar into a programming language. The power of the grammar is illustrated with a selection of examples that explore different components and their interactions, in more detail. The article concludes by discussing some perceptual issues, and thinking about how we can build on the grammar to learn how to create graphical "poems."

Supplemental materials are available online.

Key Words: Grammar of graphics; Statistical graphics.

1. INTRODUCTION

What is a graphic? How can we succinctly describe a graphic? And how can we create the graphic that we have described? These are important questions for the field of statistical graphics.

One way to answer these questions is to develop a grammar: "the fundamental principles or rules of an art or science" (OED Online 1989). A good grammar will allow us to gain insight into the composition of complicated graphics, and reveal unexpected connections between seemingly different graphics (Cox 1978). A grammar provides a strong foundation for understanding a diverse range of graphics. A grammar may also help guide us on what a well-formed or correct graphic looks like, but there will still be many grammatically correct but nonsensical graphics. This is easy to see by analogy to the English language: good grammar is just the first step in creating a good sentence.

Hadley Wickham is Assistant Professor of Statistics, Rice University, Houston, TX 77030 (E-mail: *h.wicham@gmail.com*).

The most important modern work in graphical grammars is "The Grammar of Graphics" by Wilkinson, Anand, and Grossman (2005). This work built on earlier work by Bertin (1983) and proposed a grammar that can be used to describe and construct a wide range of statistical graphics. This article proposes an alternative parameterization of the grammar, based around the idea of building up a graphic from multiple layers of data. The grammar differs from Wilkinson's in its arrangement of the components, the development of a hierarchy of defaults, and in that it is embedded inside another programming language. These three aspects form the core of the article, comparing and contrasting the layered grammar to Wilkinson's grammar. These sections are followed by a discussion of some of the implications of the grammar, how we might use it to build higher level tools for data analysis.

The ideas presented in this article have been implemented in the open-source R package, ggplot2, available from CRAN. More details about the grammar and implementation, including a comprehensive set of examples, can be found on the package website *http://had.co.nz/ggplot2*. The code used to produce the figures in this article is available online in the supplemental materials.

2. HOW TO BUILD A PLOT

When creating a plot we start with data. For this example, to focus on the essence of drawing a graphic, without getting distracted by more complicated manipulations of the data, we will use the trivial dataset shown in Table 1. It has four variables, A, B, C, and D, and four observations.

2.1 A BASIC PLOT

Let us draw a scatterplot of A versus C. What exactly is a scatterplot? One way to describe it is that we are going to draw a point for each observation, and we will position the point horizontally according to the value of A, and vertically according to C. For this example, we will also map categorical variable D to the shape of the points.

The first step in making this plot is to create a new dataset that reflects the mapping of x-position to A, y-position to C, and shape to D. x-position, y-position, and shape are examples of aesthetics, things that we can perceive on the graphic. We will also remove all other variables that do not appear in the plot. This is shown in Table 2.

Table 1.   Simple dataset.

A	B	C	D
2	3	4	a
1	2	1	a
4	5	15	b
9	10	80	b

Table 2. Simple dataset with variables named according to the aesthetic that they use.

x	y	Shape
2	4	a
1	1	a
4	15	b
9	80	b

We can create many different types of plots using this same basic specification. For example, if we were to draw lines instead of points, we would get a line plot. If we used bars, we would get a bar plot. Bars, lines, and points are all examples of geometric objects.

The next thing we need to do is to convert these numbers measured in data units to numbers measured in physical units, things that the computer can display. To do that we need to know that we are going to use linear scales and a Cartesian coordinate system. We can then convert the data units to aesthetic units, which have meaning to the underlying drawing system. For example, to convert from a continuous data value to a horizontal pixel coordinate, we need a function like the following:

$$\text{floor}\left(\frac{x - \min(x)}{\text{range}(x)} * \text{screen width}\right).$$

In this example, we will scale the x-position to [0, 200] and the y-position to [0, 300]. The procedure is similar for other aesthetics, such as shape: here we map "a" to a circle, and "b" to a square. The results of these scalings are shown in Table 3. These transformations are the responsibility of *scales*, described in detail in Section 3.2.

In general, there is another step that we have skipped in this simple example: a statistical transformation. Here we are using the identity transformation, but there are many others that are useful, such as binning or aggregating. Statistical transformations, or stats, are described in detail in Section 3.1.2.

Finally, we need to render these data to create the graphical objects that are displayed on screen or paper. To create a complete plot we need to combine graphical objects from three sources: the *data*, represented by the point geom; the *scales and coordinate system*, which generates axes and legends so that we can read values from the graph; and the *plot annotations*, such as the background and plot title. These components are shown in Figure 1. Combining and displaying these graphical objects produces the final plot, as in Figure 2.

Table 3. Simple dataset with variables mapped into aesthetic space.

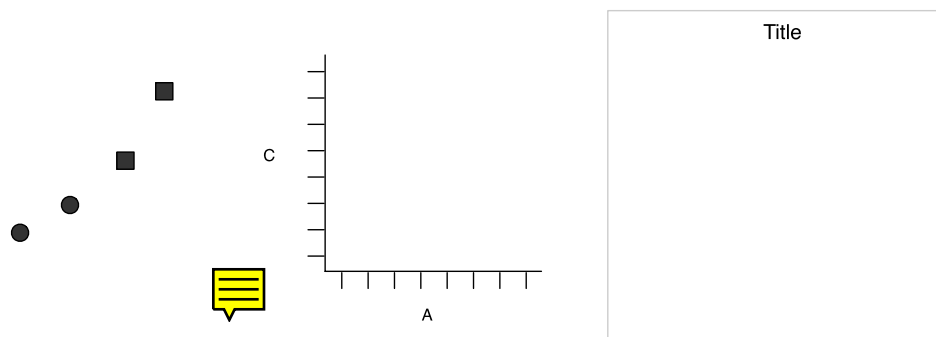| x	y	Shape
25	11	circle
0	0	circle
75	53	square
200	300	square

Figure 1.   Graphics objects produced by (from left to right): geometric objects, scales and coordinate system, plot annotations.
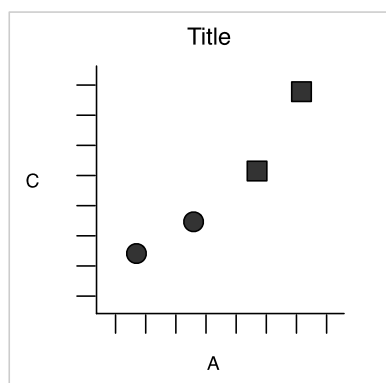

Figure 2.   The final graphic, produced by combining the pieces in Figure 1.

2.2   A More Complicated Plot

Now that you are acquainted with drawing a simple plot, we will create a more complicated plot that uses faceting. Faceting is a more general case of the techniques known as conditioning, trellising, and latticing, and produces small multiples showing different subsets of the data. If we facet the previous plot by D we will get a plot that looks like Figure 3, where each value of D is displayed in a different panel.

Faceting splits the original dataset into a dataset for each subset, so the data that underlie Figure 3 look like Table 4.

The first steps of plot creation proceed as before, but new steps are necessary when we get to the scales. Scaling actually occurs in three parts: transforming, training, and mapping.

- Scale transformation occurs before statistical transformation so that statistics are computed on the scale-transformed data. This ensures that a plot of $\log(x)$ versus $\log(y)$ on linear scales looks the same as x versus y on log scales. See Section 6.3 for more details. Transformation is only necessary for nonlinear scales, because all statistics are location-scale invariant.
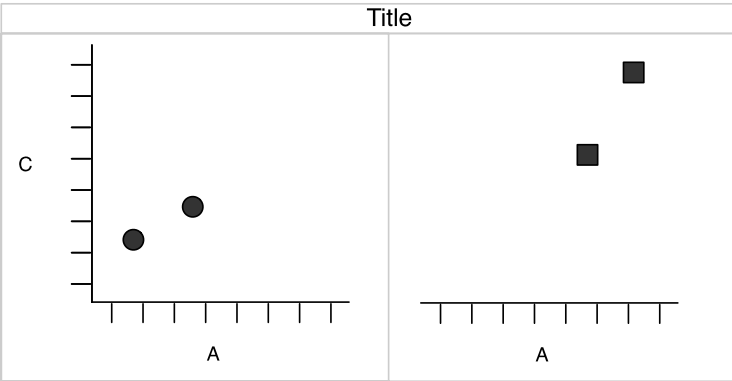
Figure 3. A more complicated plot, faceted by variable *D*. Here the faceting uses the same variable that is mapped to shape so that there is some redundancy in our visual representation. This allows us to easily see how the data have been broken into panels.

Table 4. Simple dataset faceted into subsets.

	x	*y*	Shape
a	2	4	circle
a	1	1	circle
b	4	15	square
b	9	80	square

- After the statistics are computed, each scale is trained on every faceted dataset (a plot can contain multiple datasets, e.g., raw data and predictions from a model). The training operation combines the ranges of the individual datasets to get the range of the complete data. If scales were applied locally, comparisons would only be meaningful within a facet. This is shown in Table 5.

- Finally the scales map the data values into aesthetic values. This gives Table 6, which is essentially identical to Table 2 apart from the structure of the datasets. Given that we end up with an essentially identical structure, you might wonder why we do not simply split up the final result. There are several reasons for this. It makes writing

Table 5. Local scaling, where data are scaled independently within each facet. Note that each facet occupies the full range of positions, and only uses one shape. Comparisons across facets are not necessarily meaningful.

	x	*y*	Shape
a	200	300	circle
a	0	0	circle
b	0	0	circle
b	200	300	circle

Table 6.   Faceted data correctly mapped to aesthetics. Note the similarity to Table 3.

	x	y	Shape
a	25	11	circle
a	0	0	circle
b	75	53	square
b	200	300	square

statistical transformation functions easier, as they only need to operate on a single facet of data, and some need to operate on a single subset, for example, calculating a percentage. Also, in practice we may have a more complicated training scheme for the position scales so that different columns or rows can have different x and y scales.

3. COMPONENTS OF THE LAYERED GRAMMAR

In the examples above, we have seen some of the components that make up a plot:

- data and aesthetic mappings,

- geometric objects,

- scales, and

- facet specification.

We have also touched on two other components:

- statistical transformations, and

- the coordinate system.

Together, the data, mappings, statistical transformation, and geometric object form a layer. A plot may have multiple layers, for example, when we overlay a scatterplot with a smoothed line.

To be precise, the layered grammar defines the components of a plot as:

- a default dataset and set of mappings from variables to aesthetics,

- one or more layers, with each layer having one geometric object, one statistical trans-formation, one position adjustment, and optionally, one dataset and set of aesthetic mappings,

- one scale for each aesthetic mapping used,

- a coordinate system,

- the facet specification.

These high-level components are quite similar to those of Wilkinson's grammar, as shown in Figure 4. In both grammars, the components are independent, meaning that we

GPL	ggplot2

DATA ⟶ Defaults

TRANS   Data
           Mapping

ELEMENT ⟶ Layer
           Data
           Mapping
           Geom
           Stat
           Position

SCALE ⟶ Scale
GUIDE ⟶
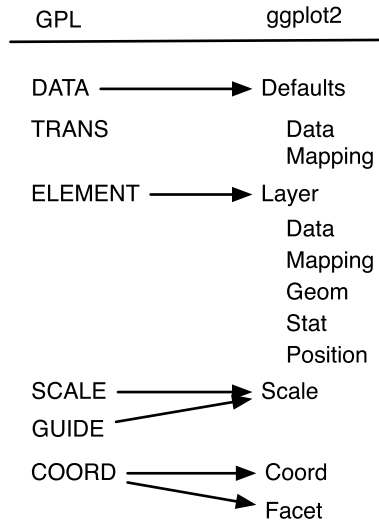
COORD ⟶ Coord
        ⟶ Facet

Figure 4.   Mapping between components of Wilkinson's grammar (left) and the layered grammar (right). TRANS has no correspondence in `ggplot2`: its role is played by built-in R features.

can generally change a single component in isolation. There are more differences within the individual components, which are described in the details that follow.

The layer component is particularly important as it determines the physical representation of the data, with the combination of stat and geom defining many familiar named graphics: the scatterplot, histogram, contourplot, and so on. In practice, many plots have (at least) three layers: the data, context for the data, and a statistical summary of the data. For example, to visualize a spatial point process, we might display the points themselves, a map giving some spatial context, and the contours of a two-dimensional density estimate.

This grammar is useful for both the user and the developer of statistical graphics. For the user, it makes it easier to iteratively update a plot, changing a single feature at a time. The grammar is also useful because it suggests the high-level aspects of a plot that *can* be changed, giving us a framework to think about graphics, and hopefully shortening the distance from mind to paper. It also encourages the use of graphics customized to a particular problem rather than relying on generic named graphics.

For the developer, it makes it much easier to create new capabilities. You only need to add the one component that you need, and you can continue to use all the other existing components. For example, you can add a new statistical transformation, and continue to use the existing scales and geoms. It is also useful for discovering new types of graphics, as the grammar defines the parameter space of statistical graphics.

3.1   LAYERS

Layers are responsible for creating the objects that we perceive on the plot. A layer is composed of four parts:

- data and aesthetic mapping,

```
line(position(smooth.linear(x * y)), color(z))

layer(aes(x = x, y = y, color = z), geom="line",
  stat="smooth")
```

Figure 5.   Difference between GPL (top) and `ggplot2` (bottom) parameterizations.

- a statistical transformation (stat),

- a geometric object (geom), and

- a position adjustment.

These parts are described in detail below.

Usually all the layers on a plot have something in common, typically that they are different views of the same data, for example, a scatterplot with overlaid smoother.

A layer is the equivalent of Wilkinson's ELEMENT, although the parameterization is rather different. In Wilkinson's grammar, all the parts of an element are intertwined, whereas in the layered grammar they are separate, as illustrated by Figure 5. This makes it possible to omit parts from the specification and rely on defaults: if the stat is omitted, the geom will supply a default; if the geom is omitted, the stat will supply a default; if the mapping is omitted, the plot default will be used. These defaults are discussed further in Section 4. In Wilkinson's grammar, the dataset is implied by the variable names, whereas in the layered grammar it can be specified separately.

3.1.1   Data and Mapping

Data are obviously a critical part of the plot, but it is important to remember that they are independent from the other components: we can construct a graphic that can be applied to multiple datasets. Data are what turns an abstract graphic into a concrete graphic.

Along with the data, we need a specification of which variables are mapped to which aesthetics. For example, we might map weight to x position, height to y position, and age to size. The details of the mapping are described by the scales; see Section 3.2. Choosing a good mapping is crucial for generating a useful graphic, as described in Section 7.

3.1.2   Statistical Transformation

A statistical transformation, or **stat**, transforms the data, typically by summarizing them in some manner. For example, a useful stat is the smoother, which calculates the mean of y, conditional on x, subject to some restriction that ensures smoothness. Table 7 lists some of the stats available in `ggplot2`. To make sense in a graphical context a stat must be location-scale invariant: $f(x+a) = f(x)+a$ and $f(b \cdot x) = b \cdot f(x)$. This ensures that the transformation is invariant under translation and scaling, common operations on a graphic.

A stat takes a dataset as input and returns a dataset as output, and so a stat can add new variables to the original dataset. It is possible to map aesthetics to these new variables. For

Table 7.   Some statistical transformations provided by `ggplot2`. The user is able to supplement this list in a
straightforward manner.

Name	Description
bin	Divide continuous range into bins, and count number of points in each
boxplot	Compute statistics necessary for boxplot
contour	Calculate contour lines
density	Compute 1d density estimate
identity	Identity transformation, $f(x) = x$
jitter	Jitter values by adding small random value
qq	Calculate values for quantile-quantile plot
quantile	Quantile regression
smooth	Smoothed conditional mean of y given x
summary	Aggregate values of y for given x
unique	Remove duplicated observations

example, one way to describe a histogram is as a binning of a continuous variable, plotted
with bars whose height is proportional to the number of points in each bin, as described in
Section 6.1. Another useful example is mapping the size (or color, or texture) of the lines
in a contour plot to the height of the contour.

The statistical method used by a stat should be conditional on the coordinate system.
For example, a smoother in polar coordinates should use circular regression, and in three
dimensions should return a two-dimensional surface rather than a one-dimensional curve.
However, many statistical operations have not been derived for non-Cartesian coordinates
and so we use Cartesian coordinates for calculation, which, while not strictly correct, will
normally be a fairly close approximation. This issue is not discussed in Wilkinson's gram-
mar.

3.1.3   Geometric Object

Geometric objects, or **geom**s for short, control the type of plot that you create. For
example, using a point geom will create a scatterplot, whereas using a line geom will
create a line plot. We can classify geoms by their dimensionality:

- 0d: point, text,

- 1d: path, line (ordered path),

- 2d: polygon, interval.

Geometric objects are an abstract component and can be rendered in different ways.
Figure 6 illustrates four possible renderings of the interval geom.

Geoms are mostly general purpose, but do require certain outputs from a statistic. For
example, the boxplot geom requires the position of the upper and lower fences, upper and
lower hinges, the middle bar, and the outliers. Any statistic used with the boxplot needs to
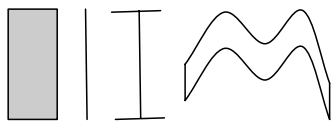provide these values.

Figure 6.   Four representations of an interval geom. From left to right: as a bar, as a line, as an error bar, and (for continuous x) as a ribbon.

Every geom has a default statistic, and every statistic a default geom. For example, the bin statistic defaults to using the bar geom to produce a histogram. Overriding these defaults will still produce a valid plot, but it may violate graphical conventions.

Each geom can only display certain aesthetics. For example, a point geom has position, color, shape, and size aesthetics. A bar geom has position, height, width, and fill color. Different parameterizations may be useful. For example, instead of location and dimension, we could parameterize the bar with locations representing the four corners. Parameterizations that involve dimension (e.g., height and width) only make sense for Cartesian coordinate systems. For example, height of a bar geom in polar coordinates corresponds to radius of a segment. For this reason location-based parameterizations are used internally.

3.1.4 Position Adjustment

Sometimes we need to tweak the position of the geometric elements on the plot, when otherwise they would obscure each other. This is most common in bar plots, where we stack or dodge (place side-by-side) the bars to avoid overlaps. In scatterplots with few unique x and y values, we sometimes randomly jitter (Chambers et al. 1983) the points to reduce overplotting. Wilkinson called these collision modifiers.

3.2 SCALES

A **scale** controls the mapping from data to aesthetic attributes, and so we need one scale for each aesthetic property used in a layer. Scales are common across layers to ensure a consistent mapping from data to aesthetics. The legends associated with some scales are illustrated in Figure 7.

A scale is a function, and its inverse, along with a set of parameters. For example, the color gradient scale maps a segment of the real line to a path through a color space. The
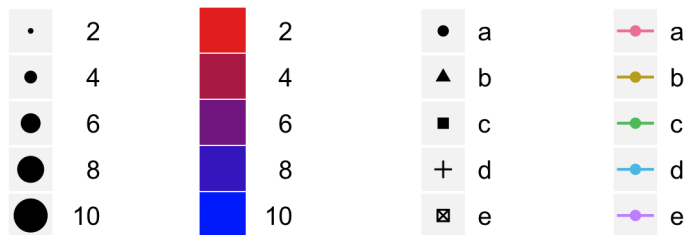

Figure 7.   Examples of legends from four different scales. From left to right: continuous variable mapped to size and color, discrete variable mapped to shape and color. The legend automatically responds to the geoms used in the plot, from left to right: points, tiles, points, points and lines.

parameters of the function define whether the path is linear or curved, which color space to use (e.g., LUV or RGB), and the start and end colors.

The inverse function is used to draw a guide so that you can read values from the graph. Guides are either axes (for position scales) or legends (for everything else). Most mappings have a unique inverse (i.e., the mapping function is one-to-one), but many do not. A unique inverse makes it possible to recover the original data, but this is not always desirable if we want to focus attention on a single aspect.

Scales typically map from a single variable to a single aesthetic, but there are exceptions. For example, we can map one variable to hue and another to saturation, to create a single aesthetic, color. We can also create redundant mappings, mapping the same variable to multiple aesthetics. This is particularly useful when producing a graphic that works in both color and black and white.

The scale of the layered grammar is equivalent to the SCALE and GUIDE of Wilkinson's grammar. There are two types of guides: scale guides and annotation guides. In the layered grammar, the scale guides (axes and legends) are largely drawn automatically based on options supplied to the relative scales. Annotation guides, used to highlight important data points, are not needed because they can be constructed with creative use of geoms if data dependent, or if not, the underlying drawing system can be used directly. Scales are also computed somewhat differently as it is possible to map a variable produced by a statistic to an aesthetic. This requires two passes of scaling, before and after the statistical transformation.

3.3 COORDINATE SYSTEM

A coordinate system, **coord** for short, maps the position of objects onto the plane of the plot. Position is often specified by two coordinates (x, y), but could be any number of coordinates. The Cartesian coordinate system is the most common coordinate system for two dimensions, whereas polar coordinates and various map projections are used less frequently. For higher dimensions, we have parallel coordinates (a projective geometry), mosaic plots (a hierarchical coordinate system), and linear projections onto the plane.

Coordinate systems affect all position variables simultaneously and differ from scales in that they also change the appearance of the geometric objects. For example, in polar coordinates, bar geoms look like segments of a circle. Additionally, scaling is performed before statistical transformation, whereas coordinate transformations occur afterward. The consequences of this are shown in Section 6.3.

Coordinate systems control how the axes and grid lines are drawn. Figure 8 illustrates three different types of coordinate systems. Very little advice is available for drawing these for non-Cartesian coordinate systems, so a lot of work needs to be done to produce polished output.

3.4 FACETING

There is also another graphical tool that turns out to be sufficiently useful that we should include it in our general framework: faceting (a more general case of the plots known as conditioned or trellis plots). Faceting makes it easy to create small multiples of different
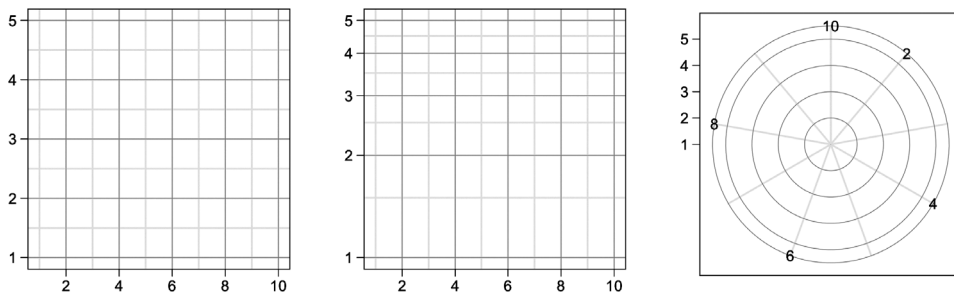
Figure 8.   Examples of axes and grid lines for three coordinate systems: Cartesian, semi-log, and polar. The polar coordinate system illustrates the difficulties associated with non-Cartesian coordinates: it is hard to draw the axes well.

```
COORD: rect(dim(3), dim(1,2))
ELEMENT: point(position(x * y * z))
geom_point(aes(x, y)) + facet_grid(. ~ z)
```

Figure 9.   Difference between GPL (top) and ggplot2 (bottom) parameterizations. Note that z is included in the position specification for the GPL element.

subsets of an entire dataset. This is a powerful tool when investigating whether patterns are the same or different across conditions. The faceting specification describes which variables should be used to split up the data, and how they should be arranged.

In Wilkinson's grammar, faceting is an aspect of the coordinate system, with a somewhat complicated parameterization: the faceting variable is specified within the ELEMENT and a separate COORD specifies that the coordinate system should be faceted by this variable. This is less complicated in the layered grammar as the faceting is independent of the layer and within-facet coordinate system. This is less flexible, as the layout of the facets always occurs in a Cartesian coordinate system, but in practice is not limiting. Figure 9 illustrates the specification.

4. A HIERARCHY OF DEFAULTS

The five major components of the layered grammar allow us to completely and explicitly describe a wide range of graphics. However, having to describe every component, every time, quickly becomes tiresome. This section describes the hierarchy of defaults that simplify the work of making a plot. There are defaults present in GPL, but they are not described in the Grammar of Graphics (Wilkinson 2005). This section will also serve to demonstrate the syntax of the ggplot2 package.

To illustrate how the defaults work, I will show how to create the two graphics of Figure 10. These plots show the relationship between the price and weight (in carats) of 1000 diamonds.
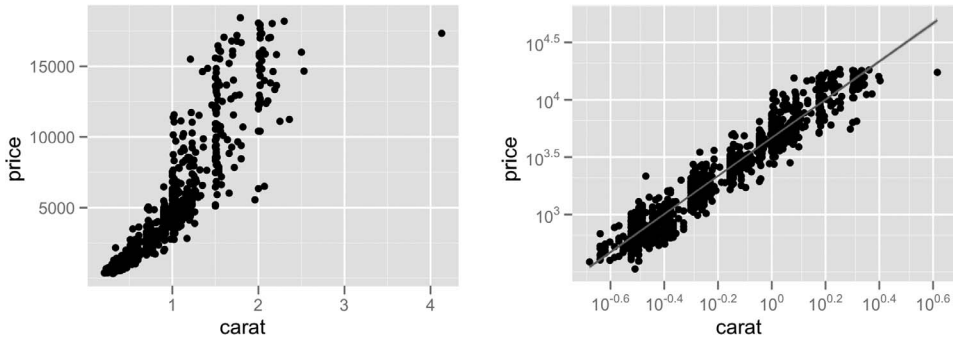
Figure 10.   (Left) Scatterplot of price versus carat. (Right) Scatterplot of price versus carat, with log-transformed scales, and a linear smooth laid on top.

The full `ggplot2` specification of the scatterplot of price versus weight is:

```
ggplot() +
  layer(
    data = diamonds, mapping = aes(x = carat, y = price),
    geom = "point", stat = "identity", position = "identity"
  ) +
  scale_y_continuous() +
  scale_x_continuous() +
  coord_cartesian()
```

We start with `ggplot()` to create a new plot object, and then add the other components: a single layer, specifying the data, mapping, geom, and stat, the two continuous position scales, and a Cartesian coordinate system. The layer is the most complicated and specifies that we want to:

- use the diamonds dataset,
- map carat to horizontal (x) position, and price to vertical (y) position, and
- display the raw data (the identity transformation) with points.

Intelligent defaults allow us to simplify this specification in a number of ways. First, we only need specify one of geom and stat, as each geom has a default stat (and vice versa). Second, the Cartesian coordinate system is used by default, so does not need to be specified. Third, default scales will be added according to the aesthetic and type of variable. For position, continuous values are transformed with a linear scaling, and categorical values are mapped to the integers; for color, continuous variables are mapped to a smooth path in the HCL color space, and discrete variables to evenly spaced hues with equal luminance and chroma. The choice of a good default scale is difficult.

This leads us to the following specification:

```
ggplot() +
  layer(
```

```
  data = diamonds, mapping = aes(x = carat, y = price),
  geom = "point"
)
```

Typically, we will specify a default dataset and mapping in the `ggplot` call, use position-based matching in `aes`, and use a shorthand for the layer:

```
ggplot(diamonds, aes(carat, price)) +
  geom_point()
```

Any aesthetics specified in the layer will override the defaults. Similarly, if a dataset is specified in the layer, it will override the plot default. To get to the second plot of Figure 10 we need to add another layer, and override the default linear-transforms with log-transformations:

```
ggplot(diamonds, aes(carat, price)) +
  geom_point() +
  stat_smooth(method = lm) +
  scale_x_log10() +
  scale_y_log10()
```

which is shorthand for:

```
ggplot() +
  layer(
    data = diamonds, mapping = aes(x = carat, y = price),
    geom = "point", stat = "identity", position = "identity"
  ) +
  layer(
    data = diamonds, mapping = aes(x = carat, y = price),
    geom = "smooth", position = "identity",
    stat = "smooth", method = lm
  ) +
  scale_y_log10() +
  scale_x_log10() +
  coord_cartesian()
```

Even with these many defaults, the explicit grammar syntax is rather verbose, and usually spans multiple lines. This makes it difficult to rapidly experiment with different plots, very important when searching for revealing graphics. For this reason the verbose grammar is supplemented with `qplot`, short for quick plot, which makes strong assumptions to reduce the amount of typing needed. It also mimics the syntax of the `plot` function, making `ggplot2` easier to use for people already familiar with base R graphics.

The `qplot` function assumes that multiple layers will use the same data and aesthetic mappings, and defaults to creating a scatterplot. This allows us to recreate the first plot with this concise code:

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

The second plot is not much longer:

```
qplot(carat, price, data = diamonds,
  geom = c("point", "smooth"),
  method = "lm", log = "xy"
)
```

Note the use of the `log` argument, which mimics the `plot` function in base graphics and specifies which axes should be log-transformed. The geom argument can take a vector of geoms, which are added sequentially to the plot. Everything else is taken to be a layer parameter. The limitations to this approach are obvious: to which layer does the `method` argument apply?

5. AN EMBEDDED GRAMMAR

The previous section hints at an important point: the specification of the layered grammar is tightly embedded within an existing programming language, R. This section discusses some consequences of such an approach. The discussion is centered around R, but many of the issues apply regardless of language. To make these issues concrete, we will attempt to reproduce the top part of Minard's famous depiction of Napoleon's march on Russia, as shown in Figure 11. This graphic can be thought of as a compound graphic: the
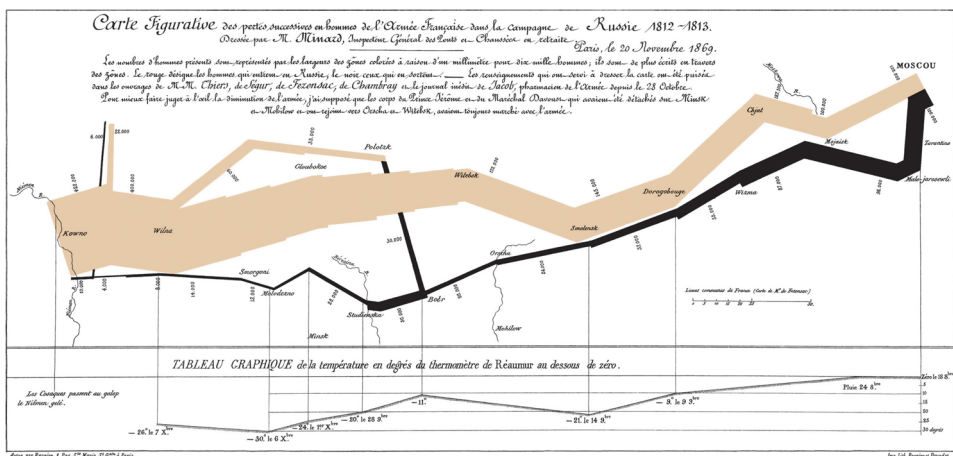

Figure 11.   "Carte figurative des pertes successives en hommes de l'Armee Français dans la campagne de Russe 1812–1813" by Charles Joseph Minard. Public domain image from *http://en.wikipedia.org/wiki/File:Minard.png*.

top part displays the number of troops during the advance and retreat and the bottom part shows the temperature during the advance.

For this example, we will focus on the top part of the graphic. Close study reveals that this part displays two datasets: cities and troops. Each city has a position (a latitude and longitude) and a name, and each troop observation has a position, a direction (advance or retreat), and number of survivors. An additional variable that is not obvious from the graphic is a group variable which separates the "arms" of the advance and retreat. These data are described more fully, and made available in digital form, in Wilkinson (2005, table 20.1, p. 624) and in the supplementary material.

How would we create this graphic with the layered grammar? The following code shows one way. We start with the essence of the graphic: a path plot of the troops data, mapping direction to color and number of survivors to size. We then take the position of the cities as an additional layer, and then polish the plot by carefully tweaking the default scales:

```
plot_troops <- ggplot(troops, aes(long, lat)) +
  geom_path(aes(size = survivors, color = direction,
    group = group))

plot_both <- troops_plot +
  geom_text(aes(label = city), size = 4, data = cities)

plot_polished <- both +
  scale_size(to = c(1, 10),
    breaks = c(1, 2, 3) * 10^5,
    labels = comma(c(1, 2, 3) * 10^5)) +
  scale_color_manual(values = c("grey50","red")) +
  xlab(NULL) +
  ylab(NULL)
```

Figure 12 shows this progression. The final plot is not as polished as Minard's, but it is produced with much less effort. This means we have more time to experiment with alternative representations of the data.

This example demonstrates two important advantages of embedding a graphical grammar into a programming language. We can use R variables to label each component as we create it, building up the final plot piece-by-piece; and we can use other functions like `comma()` to create nicely formatted axis labels. In other situations, we may find it useful to use loops to iterate over variables or subsets, or to create our own functions to encapsulate common tasks. For example, the `ggplot2` package includes a set of functions to generate plot specifications for more complicated graphics, like the parallel coordinates plot and scatterplot matrix.

With R, we also gain a wide range of prepackaged statistics useful for graphics: loess smoothing, quantile regression, density estimation, etc. Additionally, we can also use R's facilities for data import and manipulation: to have data to plot, any graphics tool must
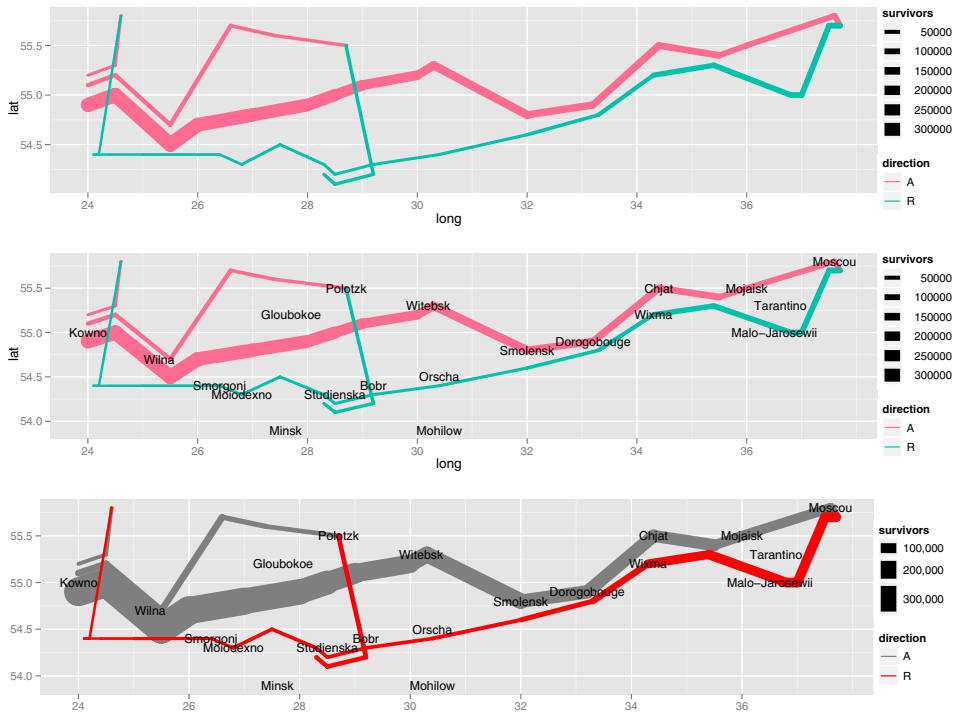
Figure 12. Iteratively reproducing the depiction of Napoleon's March by Minard. (Top) Displaying the key troop movement data. (Center) Adding town locations as reference points. (Bottom) Tweaking scales to produce polished plot.

have at least rudimentary tools for importing, restructuring, transforming, and aggregating data. By relying on other tools in R, ggplot2 does not need three elements of Wilkinson's grammar: DATA, TRANS, and the algebra.

DATA is no longer needed because data are stored as R data frames; they do not need to be described as part of the graphic. TRANS can be dropped because variable transformations are already so easy in R; they do not need to be part of the grammar. The algebra describes how to reshape data for display and, in R, can be replaced by the reshape package (Wickham 2005). Separating data manipulation from visualization allows you to check the data, and the same restructuring can be used in multiple plots. Additionally, the algebra cannot easily perform aggregation or subsetting, but reshape can.

The disadvantages of embedding the grammar are somewhat more subtle, and center around the grammatical restrictions applied by the host language. One of the most important features of the grammar is its declarative nature. To preserve this nature in R, ggplot2 uses + to create a plot by adding pieces of the definition together. The ggplot function creates a base object, to which everything else is added. This base object is not necessary in a stand-alone grammar.

6. IMPLICATIONS OF THE LAYERED GRAMMAR

What are some of the implications of defining a graphic as described above? What is now easy that previously was hard? This section discusses three interesting aspects of the grammar:

- the histogram, which maps bar height to a variable not in the original dataset, and raises questions of parameterization and defaults,

- polar coordinates, which generate pie charts from bar charts,

- variable transformations, and the three places in which they can occur.

6.1 HISTOGRAMS

One of the most useful plots for looking at one-dimensional distributions is the histogram, as shown in Figure 13. The histogram is rather special as it maps an aesthetic (bar height) to a variable created by the statistic (bin count), and raises some issues regarding parameterization and choice of defaults.

The histogram is a combination of a binning stat and a bar geom, and could be written as:

```
ggplot(data = diamonds, mapping = aes(price)) +
  layer(geom = "bar", stat = "bin",
     mapping = aes(y = ..count..))
```

One interesting thing about this definition is that it does not contain the word histogram: a histogram is not elemental, but is a combination of bars and binning. However, we do want to create histograms, without having to specify their composition every time, and so we add the histogram geom as an alias of the bar geom, with a default bin stat, and a default mapping of bar-height to bin count. This lets us produce the same plot in either of
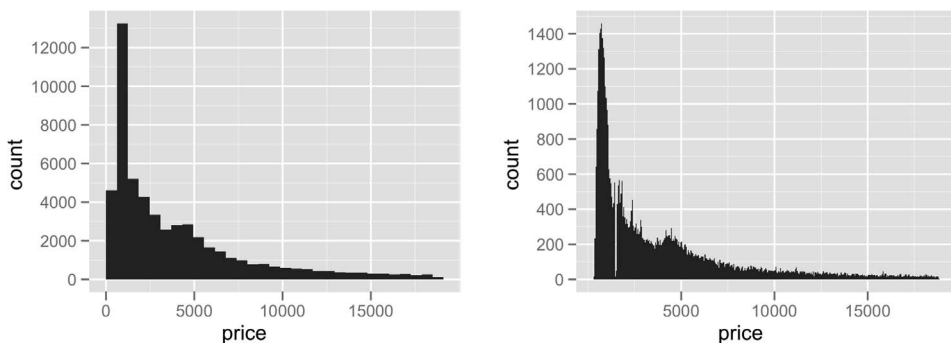

Figure 13.   Two histograms of diamond price produced by the histogram geom. (Left) Default bin width, 30 bins. (Right) Custom $50 bin width reveals missing data.

the following two more succinct ways:

```
ggplot(diamonds, aes(x = price)) + geom_histogram()
qplot(price, data = diamonds, geom = "histogram")
```

The choice of bins is very important for histograms. Figure 13 illustrates the difference changing bin widths can make. In `ggplot2`, a very simple heuristic is used for the default number of bins: it uses 30, regardless of the data. This is perverse, and ignores all of the research on selecting good bin sizes automatically, but sends a clear message to the user that he or she needs to think about, and experiment with, the bin width. This message is reinforced with a warning that reminds the user to manually adjust the bin width.

The histogram geom facilitates this experimentation by being parameterized by bin width (as opposed to number of bins, as is common elsewhere in R). This is preferable as it is directly interpretable on the graph. If you need more control you can use an alternative specification: the breaks argument specifies exactly where bin breaks should occur.

The separation of statistic and geom enforced by the grammar allows us to produce variations on the histogram, as shown in Figure 14. Using a ribbon instead of bars produces a frequency polygon, and using points produces a graphic that does not have its own name.

The histogram is interesting for another reason: one of the aesthetics, the y-position, is not present in the original data, but is the count computed from the binning statistic. In the `ggplot2` this is specified as `aes(y = ..count..)`. The two dots are a visual indicator highlighting that variable is not present in the original data, but has been computed by the statistic. There are other variables produced by the binning statistic that we might want to use instead, for example, `aes(y = ..density..)` or `aes(y = ..density../sum(..density..))`. A less common mapping is `aes(y = ..count.. / max(..count..))`, which creates a histogram with a vertical range of [0, 1], useful for faceted and interactive graphics.

The histogram also knows how to use the `weight` aesthetic. This aesthetic does not directly affect the display of the data, but will affect the stat: instead of counting the number of observations in each bin, it will sum the weights in each bar.
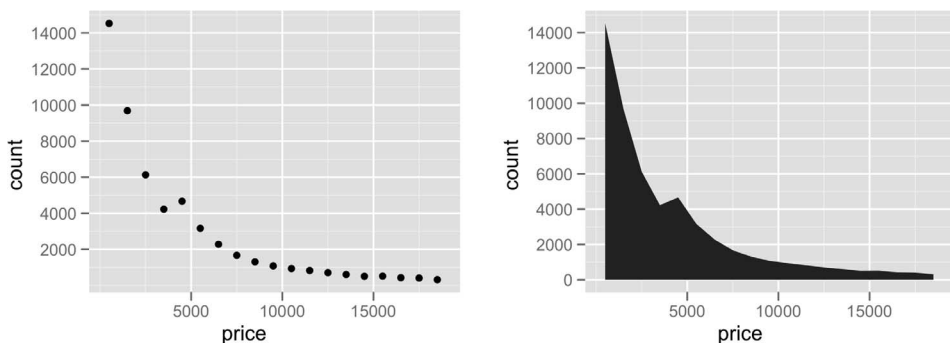

Figure 14. Variations on the histogram. Using a ribbon (left) to produce a frequency polygon, or points (right) to produce an unnamed graphic.

6.2   POLAR COORDINATES

In `ggplot2`, the user is offered the choice of a number of coordinate systems. One coordinate system used very commonly in business graphics is the polar coordinate system, used to produce pie charts and radar plots. The polar coordinate system parameterizes the two-dimensional plane in terms of angle, θ, and distance from the origin, or radius, r. We can convert to regular Cartesian coordinates using the following equations:

$$x = r \cdot \cos(\theta),$$

$$y = r \cdot \sin(\theta).$$

As with regular Cartesian coordinates, we can choose which variable is mapped to angle and which to radius. It can also be useful to tweak the range of the radius: having a minimum radius that is greater than zero can be useful because it avoids some of the compression effects around the origin.

The angle component is particularly useful for cyclical data because the starting and ending points of a single cycle are adjacent. Common cyclical variables are components of dates, like days of the year or hours of the day, and angles, like wind direction. When not plotted on polar coordinates, particular care needs to be taken with cyclical data to ensure that we do not miss patterns occurring across the arbitrary start and end points.

In the grammar, a pie chart is a stacked bar geom drawn in a polar coordinate system. Figure 15 shows this, as well as a bullseye plot, which arises when we map the height to radius instead of angle. The pie chart was created with `ggplot(diamonds,aes(x = "", fill=clarity)) + geom_bar(width = 1) + coord_polar (theta="y")`, and the only change for the bullseye chart was to map angle to the x-axis: `coord_polar(theta="x")`.

A regular bar chart converted into polar coordinates produces another type of graphic: the Coxcomb plot, popularized by Florence Nightingale (Nightingale 1857b). A comparison between a bar chart and a Coxcomb chart is shown in Figure 16. The code for these plots corresponds to the code above, but the x-axis is mapped to clarity instead of a constant.
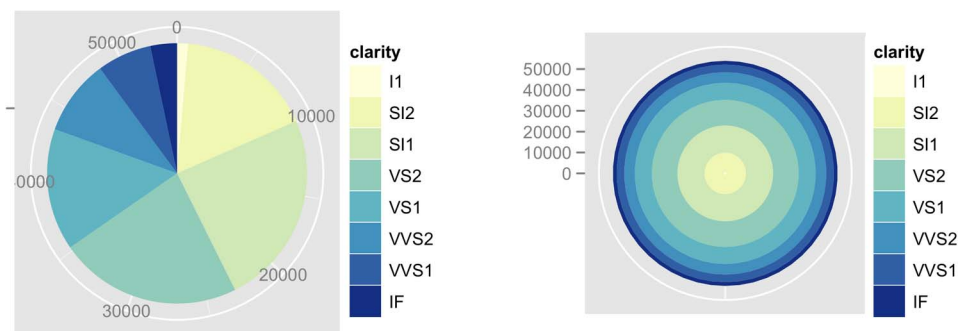

Figure 15.   Pie chart (left) and bullseye chart (right) showing the distribution of diamonds across clarity (I1 is worst, IF is best). A bullseye chart is the polar equivalent of the spineplot: in the pie chart, categories have equal radius and variable angle; in the radial chart, categories have equal angle and variable radius.
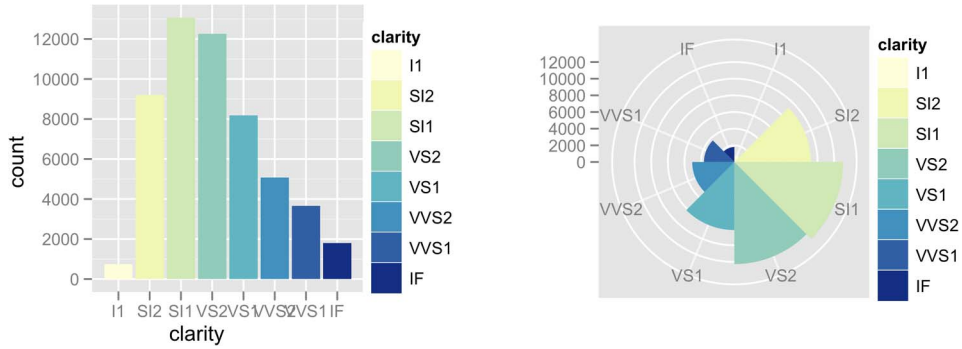
Figure 16.   Bar chart (left) and equivalent Coxcomb plot (right) of clarity distribution. The Coxcomb plot is a bar chart in polar coordinates. Note that the categories abut in the Coxcomb, but are separated in the bar chart: this is an example of a graphical convention that differs in different coordinate systems.

6.3   TRANSFORMATIONS

There are three ways to transform values in `ggplot2`: by transforming the data, by transforming the scales, or by transforming the coordinate system. Transforming the data or the scales produces graphs that look very similar, but the axes (and grid lines) are different: everything else remains the same. This is because statistics operate on data that has been transformed by the scale. Figure 17 shows an example of this using a scatterplot with a smoothed fit.

Transforming the coordinate system does something quite different, as shown in Figure 18. The coordinate transformation occurs at the very end of the plotting process, and alters the appearance of geoms. Here, the smooth is performed on the raw data, producing a straight line, but is then stretched into the new logarithmic coordinate system.

We can also use the scales and coordinate system together to display a smooth calculated on the logged data and then back-transform it to the original scale. This is shown in Figure 19. This is equivalent to fitting the model $\log(y) \sim \log(x)$ and then back-transforming predicted values.
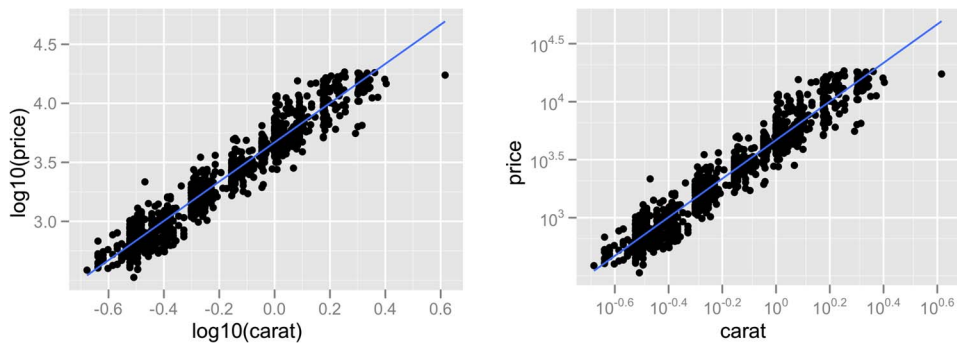

Figure 17.   Transforming the data (left) versus transforming the scale (right). From a distance the plots look identical. Close inspection is required to see that the scales and minor grid lines are different. Transforming the scale is to be preferred as the axes are labeled according to the original data, and so are easier to interpret. The presentation of the labels still requires work.
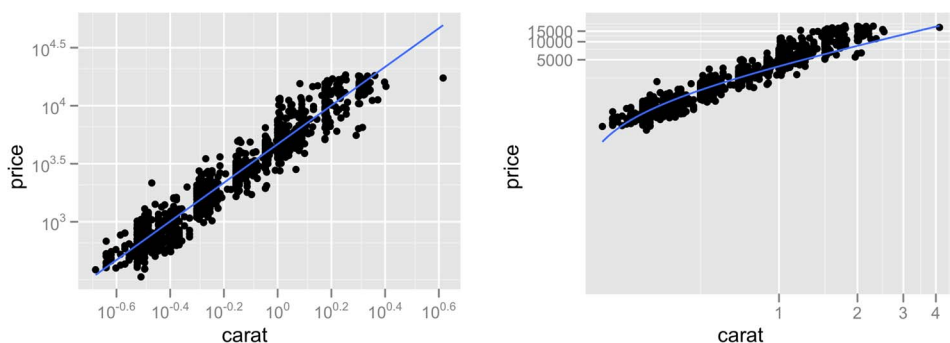
Figure 18.   Transforming the scales (left) versus transforming the coordinate system (right). Coordinate system transformations are the final step in drawing the graphic, and affect the shape of geometric objects. Here a straight line becomes a curve, and demonstrates why fitting a linear model to the untransformed data is such a bad idea.
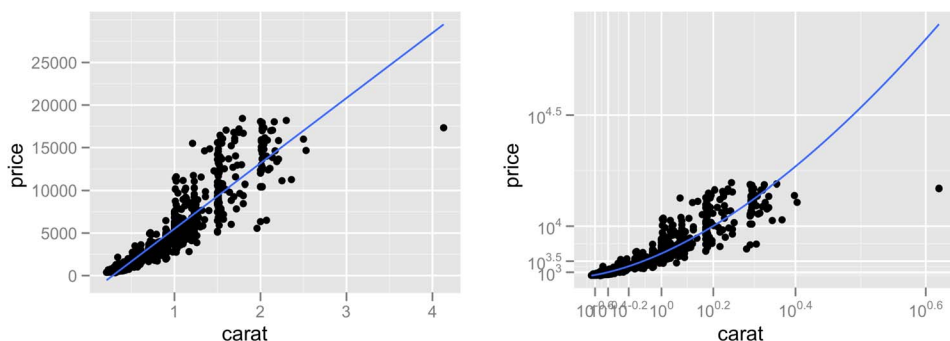

Figure 19.   Linear model fit to raw data (left) versus linear model fit to logged data, then back-transformed to original scale (right).

Currently the placement of the tick marks is suboptimal, as raw versus transformation plus back-transformation generate different tick positions when arguably they should be the same. This would mean that the scales in Figure 18 should be the same. This has the drawback of concealing whether the transformation was applied to the scale or the coordinate system. The nature of the transformation would need to be indicated in some other way, either in the caption of the graphic or elsewhere on the plot. This raises an interesting philosophical question: should we be able to uniquely identify the specification of a graphic from its rendering?

7. A POETRY OF GRAPHICS?

The grammar tells us what words make up our graphical "sentences," but offers no advice on how to write well. How can we build on top of the grammar to help data analysts build compelling, revealing graphics? What would a poetry of graphics look like? The ideas in this section draw inspiration from the tools that word processors provide to try and help us write better.

With `ggplot2`, we already have the equivalent of a spellchecker: the plot simply will not work unless the components have been correctly specified. How about a grammar checker? This tool would identify common mistakes and warn the user. This is a helpful first step: a grammar checker ensures that you have not made any obvious mistakes, but will not significantly improve your prose, and sometimes the warnings may be incorrect. For graphics, some readily detected problems and possible solutions are:

- Too many variables. It is hard to see the relationship between more than three variables in a single panel: two position and one other. We should warn the user when too many aesthetics are used, and suggest alternatives, such as faceting.

- Overplotting. It is easy to draw incorrect conclusions about the distribution of points in the presence of overplotting. There are many possible solutions. We could supplement the plot with the contours of a two-dimensional density estimate, or color each point by the local density (Unwin et al. 1996). Other solutions are suggested in Cleveland and McGill (1984), Carr et al. (1987), Huang, McDonald, and Stuetzle (1997).

- Alphabetical ordering. Categorical variables are often displayed in alphabetical ordering. Wainer (2004) called this mistake "Alabama first!" due to the propensity for states to be listed in alphabetical order. Ordering by some property of the data (e.g., mean or median) often makes the plot more useful (Becker, Cleveland, and Shyu 1996; Friendly and Kwan 2003).

- Polar coordinates. We know that humans are better at judging length than angle or area (Cleveland and McGill 1987), and polar coordinates have a singularity at the origin that makes it difficult to judge angle for objects with small radius. Should we always warn users when they choose to use this coordinate system?

We can also provide tools to help the analyst get started; instead of a blank page, we could start with a template, containing some potentially useful views. One approach is to calculate a measure of interest for a wide number of possible plots and then show the most interesting. This is the basic idea behind scagnostics (Wilkinson, Anand, and Grossman 2005) and projection pursuit (Friedman 1987; Cook et al. 2008), which explore the space of scatterplots generated by projection. We can also explore other parameter spaces, as the aspect ratio of the plot, as in Heer and Agrawala (2006), which create multiple plots each banking a different component to 45° (as recommended by Cleveland (1993)). Similarly, instead of displaying the histogram with the optimal bin width, we could display the five or ten most revealing.

Beyond that, it seems difficult to see how to do much more algorithmically, and we need to turn to education and training. The grammar provides a good foundation for teaching, as it helps students to see the deeper themes underlying different graphics, as well as providing a tool for describing and drawing graphics.

8. CONCLUSIONS

The aim of the grammar is to "bring together in a coherent way things that previously appeared unrelated and which also will provide a basis for dealing systematically with new situations" (Cox 1978). How well have we succeeded?

With the histogram, we saw how a familiar graphic can be broken into its component parts, and then those parts changed independently to get variations on an old friend. In the polar coordinates examples, we saw the deep similarities that underlie the bar chart, pie chart, and Coxcomb chart. We also accidentally created an unusual chart, the bullseye plot, and saw how it is like the polar equivalent of the spineplot. The transformation examples showed the usefulness of multiple transformation stages, and how we can mimic the common statistical practice of back-transformation using graphics alone.

One area where this grammar is not so strong is area plots: bar charts, spineplots, histograms, mosaic plots, etc. (Hartigan and Kleiner 1981; Hofmann 2003). Whereas they can be constructed with the grammar, it gives no great insight into their underlying structure, or how we can create new, useful, variations. This suggests that it may be fruitful to describe "subgrammars" for particular families of plots.

The layered grammar does not include user interaction with the plot; all plots are static and separate. Clearly there is huge scope for adding interaction to this grammar. Some types of interaction will be easier to add than others, for example, connecting a slider to the bin width of a histogram. Here we connect some user interface element to some specification of the plot. This type of interaction includes zooming and panning (changing scale limits), but not linked brushing. Linked brushing is an example of interaction with the underlying data, and incorporating this type of interaction into the grammar will require deeper thought. Speed is also a challenge; to be seamlessly perceived an interactive graphic must be updated multiple times a second, whereas many `ggplot2` graphics take over a second to draw.

The grammar gives no insight into how to display objects other than data frames. How can we visualize mixed effects models, MANOVAs, clusterings, or classifications? One can use the default visualization provided by many R packages, but these methods generally offer little flexibility, only allowing you to produce the single plot that the package author thought would be most useful. Alternatively, you can create a custom visualization by piecing together the necessary data and using `ggplot2`, but this quickly grows tiresome. Some thoughts on how to achieve a happy medium between these two extremes are included in Wickham (2009), but there is much work yet to be done.

The grammar is powerful and useful, but not all encompassing. As well as specializing the grammar for particular families of graphics, we also need to support the creation of interesting and revealing plots. The ideas outlined in the last section suggest ways to build on top of the grammar to support graphical data analysis and exploration. `ggplot2` is available from *http://had.co.nz/ggplot2*.

SUPPLEMENTAL MATERIALS

R code and data: R code to reproduce "Napoleon's march" (minard.r), location of cities
in "Napoleon's March" by Minard (minard-cities.txt), troop movement data (minard-
troops.txt), and R code to reproduce all figures in this article (layered-grammar.r). (Sup-
plements.zip)

ACKNOWLEDGMENT

[Received August 2007. Revised April 2009.]

REFERENCES

Becker, R. A., Cleveland, W. S., and Shyu, M.-J. (1996), "The Visual Design and Control of Trellis Display,"
Journal of Computational and Graphical Statistics, 5 (2), 123–155. [25]

Bertin, J. (1983), *Semiology of Graphics*, Madison, WI: University of Wisconsin Press. [4]

Carr, D. B., Littlefield, R. J., Nicholson, W. L., and Littlefield, J. S. (1987), "Scatterplot Matrix Techniques for
Large *N*," *Journal of the American Statistical Association*, 82 (398), 424–436. [25]

Chambers, J., Cleveland, W., Kleiner, B., and Tukey, P. (1983), *Graphical Methods for Data Analysis*, New York:
Chapman and Hall. [12]

Cleveland, W. (1993), "A Model for Studying Display Methods of Statistical Graphics," *Journal of Computational
and Graphical Statistics*, 2, 323–364; available at *http://stat.bell-labs.com/doc/93.4.ps*. [25]

Cleveland, W. S., and McGill, R. (1984), "Graphical Perception: Theory, Experimentation and Application to the
Development of Graphical Methods," *Journal of the American Statistical Association*, 79 (387), 531–554.
[25]

——— (1987), "Graphical Perception: The Visual Decoding of Quantitative Information on Graphical Displays
of Data," *Journal of the Royal Statistical Society, Ser. A*, 150 (3), 192–229. [25]

Cook, D., Lee, E.-K., Buja, A., and Wickham, H. (2008), "Grand Tours, Projection Pursuit Guided Tours and
Manual Controls," in *Handbook of Data Visualization. Springer Handbooks of Computational Statistics*, eds.
C. Chen, W. Härdle, and A. Unwin, Heidelberg: Springer, chapter III.2, pp. 295–314. [25]

Cox, D. R. (1978), "Some Remarks on the Role in Statistics of Graphical Methods," *Applied Statistics*, 27 (1),
4–9. [3,26]

Friedman, J. H. (1987), "Exploratory Projection Pursuit," *Journal of American Statistical Association*, 82, 249–
266. [25]

Friendly, M., and Kwan, E. (2003), "Effect Ordering for Data Displays," *Computational Statistics & Data Analy-
sis*, 43 (4), 509–539, doi: *http://dx.doi.org/10.1016/S0167-9473(02)00290-6*. [25]

Hartigan, J. A., and Kleiner, B. (1981), "Mosaics for Contingency Tables," in *Computer Science and Statistics:
Proceedings of the 13th Symposium on the Interface*, Fairfax Station, VA: Interface Foundation of North
America, pp. 268–273. [26]

Heer, J., and Agrawala, M. (2006), "Multi-Scale Banking to 45 Degrees," *IEEE Transactions on Visualization
and Computer Graphics*, 12 (5), available at *http://vis.berkeley.edu/papers/banking/*. [25]

Hofmann, H. (2003), "Constructing and Reading Mosaicplots," *Computational Statistics and Data Analysis*, 43
(4), 565–580. [26]

Huang, C., McDonald, J. A., and Stuetzle, W. (1997), "Variable Resolution Bivariate Plots," *Journal of Computational and Graphical Statistics*, 6, 383–396. [25]

Nightingale, F. (1857), *Notes on Matters Affecting the Health, Efficiency and Hospital Administration of the British Army*, London: Private publication. [22]

OED Online (1989), *The Oxford English Dictionary* (2nd ed.), Oxford University Press. Available at *http://dictionary.oed.com/cgi/entry/50097652*. [3]

Unwin, A. R., Hawkins, G., Hofmann, H., and Siegl, B. (1996), "Interactive Graphics for Data Sets With Missing Values—MANET," *Journal of Computational and Graphical Statistics*, 5 (2), 113–122. [25]

Wainer, H. (2004), *Graphic Discovery: A Trout in the Milk and Other Visual Adventures*, Princeton University Press. [25]

Wickham, H. (2005), "reshape: Flexibly Reshape Data," R package version 0.7.1, available at *http://had.co.nz/reshape/*. [19]

——— (2008), "Practical Tools for Exploring Data and Models," Ph.D. thesis, Iowa State University, available at *http://had.co.nz/thesis*. [27]

——— (2009), *ggplot2: Elegant Graphics for Data Analysis: Use R*, Springer. [26]

Wilkinson, L. (2005), *The Grammar of Graphics* (2nd ed.). *Statistics and Computing*, New York: Springer. [14,18]

Wilkinson, L., Anand, A., and Grossman, R. (2005), "Graph-Theoretic Scagnostics," in *IEEE Symposium on Information Visualization*, pp. 157–164. [4,25]