Data Visualization with lessR and ggplot2

David W. Gerbing
The School of Business
Portland State University

April 3, 2026

Contents

1 Visualization Systems

The transformation of variables to visual aesthetics defines a *mapping*. At a minimum, the mapping associates the variables plotted with the corresponding axes, and then applies visual aesthetics such as color and size, with default choices if not explicitly stated.

mapping: Associate elements of one set, values of variables, with elements of another set, aesthetic properties of a visualization.

1.1 The Three Questions

Except for visualizations such as drawing geographical maps, data analysis typically begins seeking answers to three fundamental questions. Answering these questions results in three corresponding types of visualizations (Gerbing, *submitted manuscript*).

1. *Question*: How are numerical values for one or two categorical variables organized?
   Answer: Distribution of the values of a categorical variable.
   Visualization: Count the values in each category or combination of categories.
   Visualization: Compute a summary statistic of a continuous variable, such as the mean, for each category or combination of categories.
   e.g., Bar chart, dot plot, bubble plot of counts

2. *Question*: What is the shape and variability of a continuous variable?
   Answer: Distribution of the values of a continuous variable.
   Visualization: Display the distribution of a continuous variable.
   e.g., Violin plot, boxplot, scatterplot, density plot, histogram

3. *Question*: To what extent are two or more continuous variables related?
   Answer: Joint distribution of the values of the continuous variables.
   Visualization: e.g., Scatterplot, contour plot.
   Answer: Time series of a continuous variable.
   Visualization: Scatterplot of the values of the variable over time with adjacent points connected by a line segment.
   e.g., Run chart, time series chart

This book includes extensive discussions and visualizations from what has become the standard source of `R` visualizations, `ggplot2`, as well as equivalent visualizations obtained from the more straightforward, though not as customizable, `lessR` package.

1.2 lessR

The defining characteristic of `lessR` visualization is that, apart from geographic maps and related displays, it reduces routine visualizations to three high-level functions, each of which corresponds to one of these three basic questions. The general instructions for conducting a wide variety of routine data visualizations can be concisely stated.

The three fundamental visualization questions each correspond to a `lessR` data visualization function.

Within `lessR`, implement visualization corresponding to a distinct analytic task:

1. Aggregated values: `Chart()`, based on a categorical variable `x`
   How many or to what extent across categories?

2. Distributions of continuous variables: `X()`
   What is a variable's shape and variability?
3. Relationships between continuous variables: `XY()`, mapping continuous variables `x` and `y` within an `x-y` coordinate system
   How are continuous variables related?

The three data view functions share a common argument structure for the visualization type and the grouping and configuration.

Visualization type. The `type` argument determines the particular graphical form from the available alternatives for rendering given the specified data. The selection is from *analytically equivalent geometric forms* rather than from graphical primitives, such as points, lines, and bars. A bar chart, radar chart, and treemap all answer the first of the three listed questions: How many or how much in each category? However, these different visualizations apply different perceptual strategies from which the user can choose the desired form.

This distinction separates analytic intent, the question being pursued, from visual form, which is chosen according to perceptual or aesthetic preference. The analytical intent is set by the choice of the visualization function, and then the specific visual form is chosen by the user. For example, the respective defaults for `Chart()`, `X()`, and `XY()` are bar charts, histograms, and scatterplots, but multiple alternatives exist for each. The function choice establishes the conceptual frame that sets the general structure of the visualization and the subsequent choice of the `type` argument determines the visual form within it.

stratification: Do separate analyses for different sub-groups of data.

Grouping variables. Two arguments support *stratification* within and across panels:

- `by`: overlay or nest the visualizations of the same geometric form for different groups *within* a panel, distinguished by visual aesthetics such as color, symbol, or position.
- `facet`: partition the visualizations of the same geometric form *across* panels, in the style of Trellis displays.

In `Chart()`, hierarchical or part-whole visualizations, such as the pie chart, the argument `by` nests the levels of aggregation, resulting in a sunburst chart. Within this context, define `by` as a vector for increasingly deeper levels of nesting. In comparative views, such as the bar chart, the `by` argument overlays groups. The other stratification technique, faceting, yields side-by-side panels with coordinated axes and shared scales for each level or combination of levels for the specified categorical variables.

Data view grammar. The instructions to create a visualization within this framework follow a unified syntactic pattern. Although many formatting and aesthetic customizations are available regarding the size, color, and placement of graphical elements beyond the defaults, the data view grammar reduces to five structural parameters available to each of the three visualization functions (here, assuming the variables are in the default data frame `d`):

```
Function(x, y, by, facet, type)
```

The required variable `x` is categorical for `Chart()` and continuous for `X()` and `XY()`. The continuous variable `y` is optional for `Chart()`, enabling aggregation of a numeric variable according to a specified statistic via the `stat` argument. The continuous variable `y` is required for `XY()` and not available for `X()`.

For example, given variable *Dept* in data frame *d*, create the pie chart of the count of employees in each department of a company, and then the mean salary of employees for each department, with:

```
Chart(Dept, type="pie")
Chart(Dept, type="pie", y=Salary, stat="mean")
```

If *Dept* is not in data frame *d*, then specify the name of the data frame as the argument to the parameter `data`. If *Dept* is in the user's workspace instead of a data frame, specify the argument a `NULL`.

By shifting from a grammar for composing geometric forms to a concise grammar of analytic structure, this minimal interface, three functions and five parameters, supports a wide range of common value visualizations.

1.3 ggplot2

layer: A plotted set of geometric objects that correspond to specified data values.

ggplot2 function, `ggplot()`: Create a `ggplot2` object to plot, perhaps identifying the data frame and associated variables to plot.

aes() function, `ggplot2`: Specify the visual aesthetics.

All visualization functions plot the various components of a visualization one at a time. The compelling feature of `ggplot2` is explicitly plotting the layers, with much control provided to the user for the construction of these layers. For example, each layer may be constructed from potentially different data, with different geometric objects plotted, thereby blending multiple visualizations into a single composite. This flexibility allows analysts to add meaningful specifications in separate layers, rather than being constrained to the specific structure provided by generic plotting functions.

A `ggplot2` visualization begins with the `ggplot()` function. Specify the data frame that contains the variables for analysis. Next, use the embedded `aes()` function to specify the data values of the one or more variables to visualize. Finally, specify the geometric object to plot, referred to as a `geom_histogram`, such as points or bars. Figure 1 shows how to create a bar chart, which consists of a single layer, the bars of a bar chart.

```
data frame  variable   object to plot
ggplot(d,  aes(Dept))  +  geom_bar()
```

Figure 1: `ggplot2` function calls to generate a bar chart from the tabulation of the count for each level of the categorical variable *Dept* in the *d* data frame.

If the same data frame and associated variables apply to all the layers in the plot, as in this example, then generally specify the data frame and its constituent variables within the call to `ggplot()`.

A visualization consists of visual aesthetics such as coordinates on the x and y-axes, border color and fill color of plotted objects, and the shape of the plotted objects

such as a dashed line, or a diamond for a plotted point. The grammar of graphics explicitly associates these aesthetics, what the viewer perceives, to the properties of the data for one or more variables.

As Figure 1 illustrates, specify the variables for which to apply the visual aesthetics in `ggplot2` with the `aes()` function. The default first argument to the `aes()` function is the variable on the x-axis. So a variable name as the first argument in the parameter list not preceded with an `x=` is implicitly defined as the variable mapped to the x-axis. Any second unnamed argument, if present, is the y-axis variable.

geom(), `ggplot2`: A function that specifies a geometric object to plot.

Typical geometric objects to plot include bars, lines, and points. Plot at least one geometric object or `geom` in a layer that corresponds to some property of the data specified by the aesthetics. Specify a specific `geom` by its own function, such as `geom_bar()` to plot the bars of a bar chart. Figure 1 includes all three functions that specify the resulting bar chart. Many other default values are also invoked.

The function calls in Figure 1 follow the useful R convention that the parameters do not need to be named if they follow the order of the parameters in the function definition. The first two `ggplot()` parameters are respectively named `data` and `mapping`. The `aes()` function maps, that is, transforms, the data values to the aesthetics as plotted with a specific geometric object, e.g., a gray bar. The first parameter for the `aes()` function is the variable x.

The function call that generates a bar chart with all named given parameter values follows.

> *ggplot2*: `ggplot(data=d, mapping=aes(x=Dept)) + geom_bar()`

For aesthetics unique to a layer in a multi-layer plot, call the `aes()` function within the layer's `geom)` function. For example, because there is only one layer in the bar chart, the following statement generates the same figure. Here, the call to `ggplot()` specifies the default data table (frame) for all of the layers, *d*, but the call to `aes()` specifies the variable *Dept* only within the `geom_bar()` layer. Without a reference to a data frame in `geom_bar()`, the referenced variable, *Dept*, must be in the previously referenced *d*.

> *ggplot2*: `ggplot(d) + geom_bar(aes(Dept))`

In the following example, there is no default data table for the layers of this visualization because there is no `data` argument for `ggplot()`. Here, specify both the variable plotted and the name of the data table in the specific `geom_histogram` function that defines the plotted object for the layer. The following line of code also generates the same bar chart because this visualization only contains a single layer.

> *ggplot2*: `ggplot() + geom_bar(aes(Dept), data=d)`

To apply these specifications to subsequent layers, unless explicitly overridden by new specifications, include in the originating `ggplot()` function call both the variable(s) with the data values to plot, as well as the data frame that contains these variables.

This flexibility provides for different data frames and variables specified as needed in different layers, a topic explored later.

1.3 ggplot2 Cleveland Dot Plot

Prepare the data for the dotplot of the

- categorical variable `Name`
- continuous variable `Pre`

`Read()` is the `lessR` function for reading data. Create the variable name from the row names of the Employee data frame named `d`.

```
d <- Read("Employee")
d$Name <- rownames(d)
```

Do the dot plot. This illustrates how multiple geometric objects can be displayed on the same visualization by specifying each individual `geom` function.

`|>` is the Base R pipe operator, which pipes (moves) the indicated data source on the left of the operator to the first position of the function call to the left of the operator. Then the reference to that data is not included in the function call. For example, replace `slice(d, 1:8)` with `d |> slice(1:8)`.

1. To save space, work with only the first eight rows of data instead of all 36 by Using the `dplyr` function `slice()`.
2. Sort the data with the Base R function `reorder()`.
3. Draw the line segments with `geom_segment()`, specify the beginning and ending x and y coordinates for all the points with `geom_point()`.
4. Use the `labs()` function to suppress the y-axis label.

```
d |> slice(1:8) |>
ggplot(aes(x=Pre, y=reorder(Name, Pre))) +
  geom_segment(aes(x=0, xend=Pre, y=reorder(Name, Pre),
                   yend=reorder(Name, Pre))) +
  geom_point() +
  labs(y=NULL)
```

1.3 ggplot2 Ring (Pie) Chart

The strategy is to plot the ring chart as a bar chart in radial coordinates with the function `coord_radial()`.

```
d |> filter(!is.na(Dept)) |>
ggplot(aes(x = factor(1), fill = Dept)) +
  geom_bar(width = 1, color = "white") +
  coord_radial(theta = "y", inner.radius = 0.68, expand = FALSE) +
  theme_void() +
  theme(legend.title = element_text(face = "bold")) +
  scale_fill_grey(start = .8, end = .3)
```

1.4 Compare ggplot2 with lessR

Hadley Wickham's ggplot2 package is the most downloaded R contributed package, an impressive accomplishment with over 23,000 available contributed R packages on the CRAN servers. With ggplot2, Hadley Wickham implemented the conceptual work of Leland Wilkinson, which articulates a grammar of graphics, the gg in ggplot2.

All visualization software creates a visualization in successive layers, step-by-step, one feature at a time. For ggplot2, the analyst explicitly controls the implementation of this general grammar with a separate function call to implement each layer. The result is a toolkit, a set of building blocks available to the user that provides the basis for an extraordinarily wide range of both standard and highly customized visualizations.

lessR motivation: Minimal computer input should provide the most beneficial computer output.

lessR and ggplot2 implement complementary approaches to data visualization. By design, lessR flips some of the perspectives of ggplot2, and, indeed, the perspective of R in general. The result is a different user experience to obtain the same quality of output across a wide variety of routine visualizations. The primary distinctions include the following.

ggplot2 theme: Implement a wide variety of visualization functions for maximum flexibility and creativity.

1. Flexibility vs. minimal code. ggplot2 is a true graphics programming language. Some visualizations can be created with as few as three function calls on a single line of code. More complex, customized visualizations, however, may require many additional ggplot2 layers and parameter settings spread across dozens of lines. In contrast, the core motivation of lessR is to let users write as little code as possible while still obtaining useful and comprehensive output. lessR provides a broad range of visualizations from just three core functions: `Chart()`, `X()`, and `XY()`, along with `getColors()` for generating color palettes.

The help files for the three core lessR visualization functions provide guidance for customizing a wide variety of visualizations. Although the parameter list for each function is extensive, each help file groups related parameters together to simplify access. The tradeoff is reduced flexibility beyond routine data visualizations when compared with ggplot2, which excels at constructing highly customized visualizations when the user is willing to provide the necessary code.

2. Bottom-up vs. top-down. The traditional approach to constructing visualizations, such as with ggplot2, may be called bottom-up: Construct the visualization from its most basic components. Enhance by invoking additional options. For ggplot2, these additions consist of calling other functions to add additional visualization layers, or customizing a given layer. The flipped perspective of lessR follows a top-down approach: The form of the visualization perceived as the most desired is offered by

default, or as a simple option that offers many enhancements simultaneously. To change the default, the user typically assigns parameter values to remove unwanted components, without the need to search the help manuals for multiple functions.

The traditional bottom-up approach requires the user to focus on constructing the visualization from lower-level components. The user explicitly codes for each aspect of the visualization. The advantage is much potential for customization. The disadvantage is that the bottom-up approach requires more work to obtain the same result if available as an equivalent top-down visualization.

3. Visualization only vs. plus statistics. Unlike ggplot2, lessR is not a visualization package per se, but a data analysis system for a core set of analyses that includes visualizations and corresponding statistical analyses. For example, to conduct a full regression analysis in R – which includes collinearity analysis, outlier/influence analysis, and prediction intervals, as well as several visualizations – requires up to 20 different R functions. In contrast, the lessR function Regression() provides the entire analysis, including multiple visualizations, with one function call.

lessR also includes other features. The lessR functions replace many of the usually cryptic R error messages with more extensive instructions that explain more fully the nature of the error and its remedy. A default data table (frame) name, *d*, requires only the function name and variable name(s) contained within that data frame to generate the analysis. Again, however, ggplot2 remains the ultimate tool for user flexibility for generating visualizations beyond those typically encountered visualizations.