Week 7. Big Data Analytics
Visualization with ggplot2

Hyeonsu B. Kang
hyk149@eng.ucsd.edu

May 2016

1 The Layered Grammer of ggplot2

"ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics." http://ggplot2.org/

Many people love ggplot2. It makes graphics easier, provides a set of grammar that facilitates research into new types of display, and it consists of orthogonal components and minimal special cases.

In this session, we will recreate some ggplot2 examples. More could be found at http://docs.ggplot2.org/current/.

1.1 Line plots with ggplot2

We can plot horizontal, vertical, and slanted lines by specifying slope and intercept using ggplot2. The syntax:

```
geom_hline(mapping = NULL, data = NULL, ..., \
yintercept, na.rm = FALSE, show.legend = NA)

geom_vline(mapping = NULL, data = NULL, ..., \
xintercept, na.rm = FALSE, show.legend = NA)

geom_abline(mapping = NULL, data = NULL, ..., \
slope, intercept, na.rm = FALSE, show.legend = NA)
```

Let's use a default data set in R called mtcars.

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Now creating a dot plot of wt and mpg is very simple:

```
> p = ggplot(mtcars, aes(wt, mpg)) + geom_point()
> p
```

It is important to "feel" the layered grammar of visualization here. First, we have created a plot data plane using ggplot() and then added the dot-plot visualization specification using geom_point(). The + operator serves as an indicator that we are "adding" more visualization specifications on top of the data plane.

Adding one or more lines to the plot is also straightforward, we can call functions such as geom_vline(), geom_hline() or geom_abline(). Please note that we have saved the original dot plot in p, so that we can reuse it in the layered grammar manner.

```
> # adding a fixed line using geom_line
> p + geom_vline(xintercept = 5)
> # adding multiple fixed lines
> p + geom_vline(xintercept = 1:5)
> p + geom_hline(yintercept = c(15, 20, 25))
> p + geom_abline(intercept = 20)
```

How to fit a linear line to the data displayed? First, we can think of using `geom_abline()`. As in its syntax shown in the beginning of this tutorial, we can specify the slope and intercept of the line – thus we will first need to calculate the linear line's slope and intercept.

```
> # calculate slope and intercept of a line of the best fit
> coef(lm(mpg ~ wt, data = mtcars))
(Intercept)          wt
  37.285126   -5.344472
> p + geom_abline(intercept= 37.285, slope = -5.344)
```

In fact, we do not have to calculate the intercept and the slope beforehand for the visualization purpose, as fitting a linear line can be done using the following command:

```
> p + geom_smooth(method = "lm", se = FALSE)
```

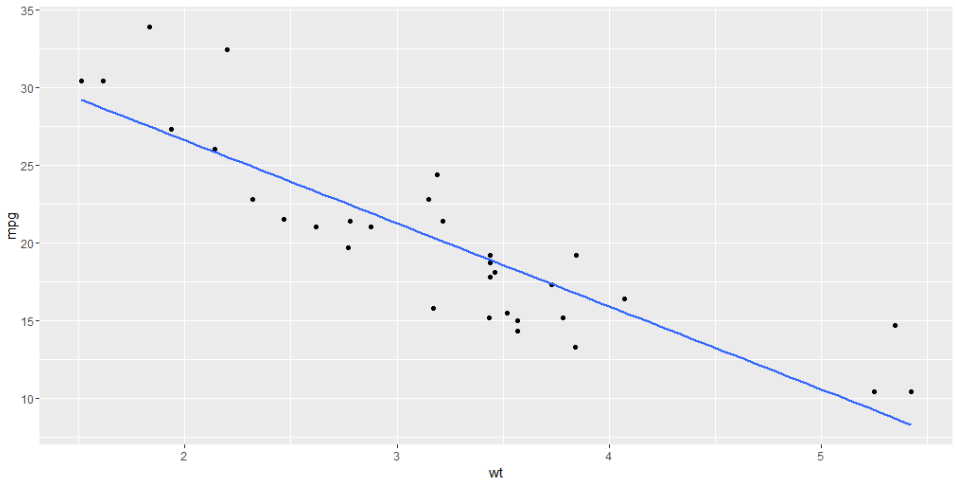

Figure 1: The fitted linear line

How about showing different lines from different facets of the data, thus creating small multiples of it? For example, in `mtcars`, the data points can be classified into three different groups depending on their number of cylinders. Can we separate them out and plot? Using `facet_wrap()` and specifying `aes()` can do exactly that.

```
> p = ggplot(mtcars, aes(mpg, wt)) + geom_point() + facet_wrap(~ cyl)
> cyl4 = mtcars$cyl == 4
> cyl6 = mtcars$cyl == 6
> cyl8 = mtcars$cyl == 8
> meanWT4 = mean(mtcars[cyl4,]$wt)
> meanWT6 = mean(mtcars[cyl6,]$wt)
> meanWT8 = mean(mtcars[cyl8,]$wt)
> meanWT = data.frame(cyl = c(4,6,8), wt = c(meanWT4, meanWT6, meanWT8))
> p + geom_hline(aes(yintercept = wt), meanWT)
```

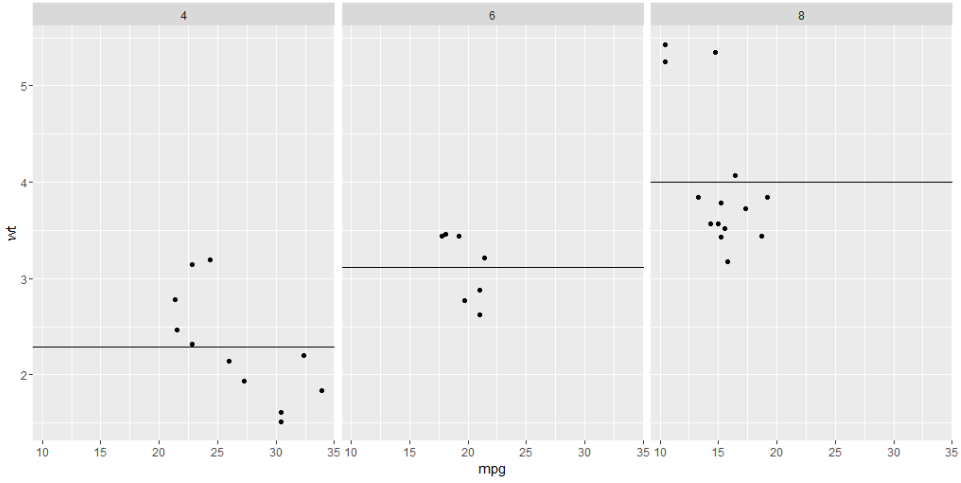

Figure 2: Dot plot separated based on the number of cylinders in `mtcars`

Adding some shades and making the plot more visually explicit is also possible by specifying the `color` option in `aes()`:

```
> p = ggplot(mtcars, aes(mpg, wt, color = wt)) + geom_point() + facet_wrap(~cyl)
> p + geom_hline(aes(yintercept = wt, color = wt), meanWT)
```

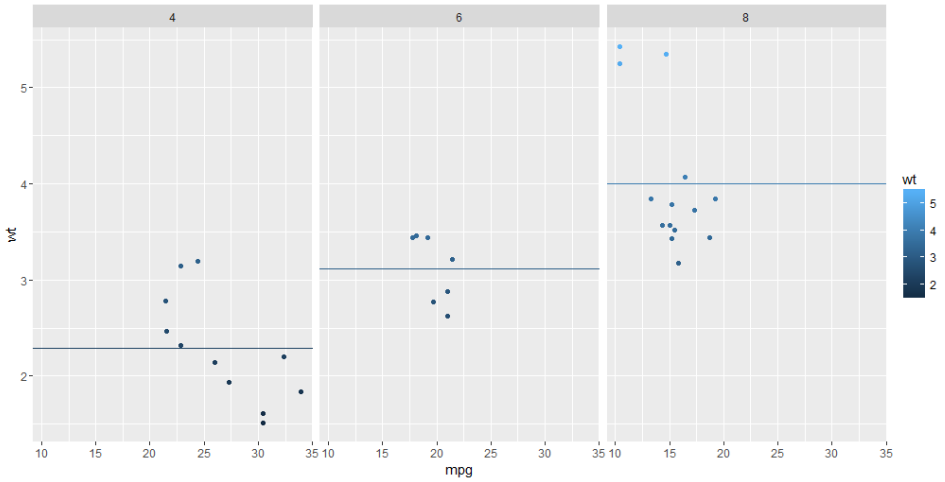

Figure 3: Colored plot

1.2 Data ellipses with ggplot2

The method for calculating ellipses can be found from `http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/`. Using another default data set called `faithful`, we can manually specify some bounding ellipses such that visual groupings of data points are explicit. The basic syntax of `stat_ellipse()` is as follows:

```
> head(faithful)
  eruptions waiting
1     3.600      79
2     1.800      54
3     3.333      74
4     2.283      62
5     4.533      85
6     2.883      55

stat_ellipse(mapping = NULL, data = NULL, geom = "path", \
position = "identity", ..., type = "t", level = 0.95, \
segments = 51, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

> ggplot(faithful, aes(waiting, eruptions)) + geom_point() + stat_ellipse()
```

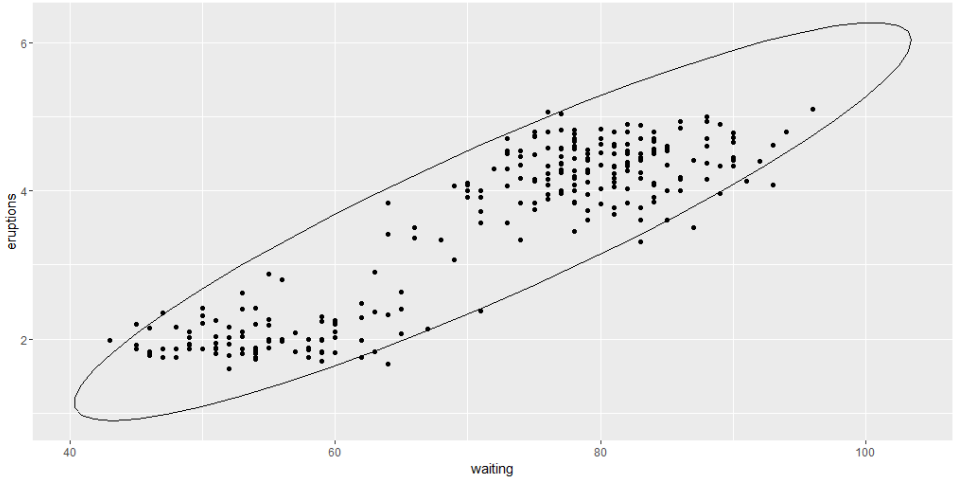

Figure 4: Dot plot with a default ellipse

Or sub-grouping into smaller sets of points is also possible:

```
> ggplot(faithful, aes(waiting, eruptions, color = eruptions > 3)) +
+   geom_point() + stat_ellipse()
```

Figure 5: Dot plot with small ellipses

Finally, specifying the type of ellipses is also possible (for more details about this specification, please visit `http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/`). Here, we will assume that the default data set follows a multivariate t-distribution (in solid lines) and add new ellipses that indicate multivariate normal distribution of the data.

```
> ggplot(faithful, aes(waiting, eruptions, color = eruptions > 3)) +
+     geom_point() + stat_ellipse(type = "norm", linetype = 2) + stat_ellipse(type = "t")
```

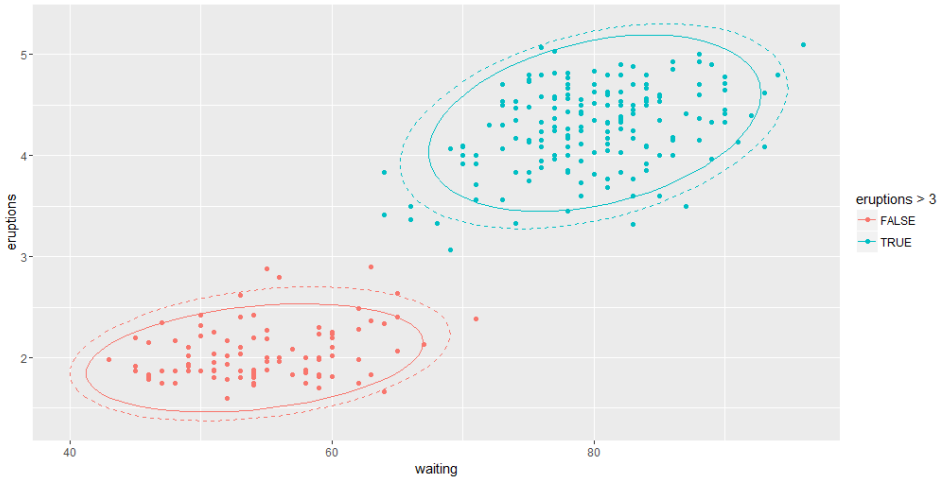

Figure 6: Dot plot with different multivariate ellipses

1.3 Vertical intervals: lines, crossbars and error bars

Showing various vertical lines and error bars can be easily done with `ggplot2` as well. The basic syntax is as follows:

```
geom_crossbar(mapping = NULL, data = NULL, stat = "identity", position = "identity", \
..., fatten = 2.5, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

geom_errorbar(mapping = NULL, data = NULL, stat = "identity", position = "identity", \
..., na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

geom_linerange(mapping = NULL, data = NULL, stat = "identity", position = "identity", \
..., na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

```
geom_pointrange(mapping = NULL, data = NULL, stat = "identity", position = "identity", \
..., fatten = 4, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

In this example, we use a simple data frame for the purpose of visualization. Download it from `https://bigdata452.slack.com/files/hyk149/F17QVMGM8/data.csv` and locate it in the current project folder. Drawing a vertical line plot using `resp` and `trt` of the data set is as simple as

```
> df
  trt resp group upper lower
1   1    1     1   1.1   0.8
2   1    5     2   5.3   4.6
3   2    3     1   3.3   2.4
4   2    4     2   4.2   3.6
> p = ggplot(df, aes(trt, resp, color = group))
> p + geom_linerange(aes(ymin = lower, ymax = upper))
```

Adding observation points is

```
> p + geom_pointrange(aes(ymin = lower, ymax = upper))
```

Adding cross bars or error bars is also simple:

```
> p + geom_crossbar(aes(ymin = lower, ymax = upper), width = 0.2)
> p + geom_errorbar(aes(ymin = lower, ymax = upper), width = 0.2)
```

Cross line connections can also be made using

```
> p + geom_line(aes(group = group)) +
+     geom_errorbar(aes(ymin = lower, ymax = upper), width = 0.2)

> p = ggplot(df, aes(trt, resp, fill = group))
> dodge = position_dodge(width = 0.9)
> p + geom_bar(position = dodge, stat = "identity") +
+     geom_errorbar(aes(ymin = lower, ymax = upper), position = dodge, width = 0.25)
```

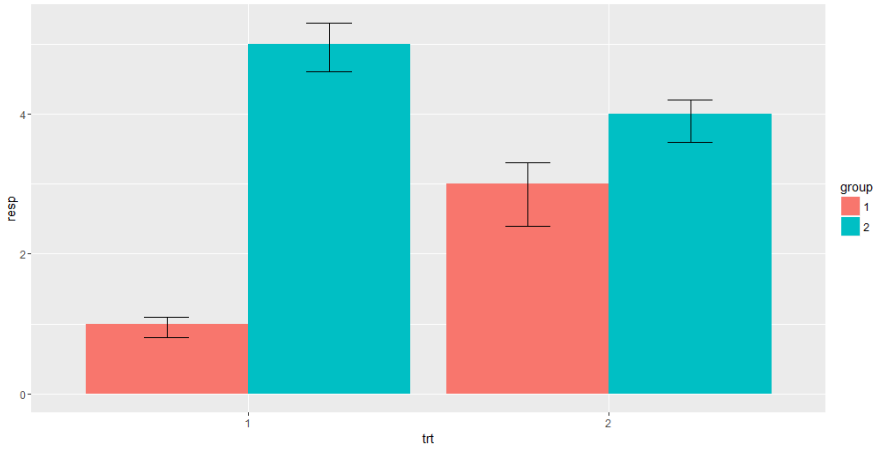

Figure 7: Error bars with bar chart