How Do `ggplot()` Objects Reflect The Grammar of Graphics?

Zhanyuan Zhang

October 26, 2017

Introduction:

The Grammar of Graphics is a systematic way to describe graphs or plottings. We have brifely mentioned its concept in lectures. In this post, I want to dig deeper into this sophisticated "grammar" via the package *ggplot2* to understand how the package *ggplot2* reflects the key conponents of the "grammar."
This report will focus on the analyses on several `ggplot()` objects.

```
# impost package
library(ggplot2)
```

Example Data

For demonstration purposes, I will use the *iris*, a built-in dataset in R.

```
# import dataset
df <- iris
```

Investigate it a little bit, we find that it is a 150 rows and 5 column.

```
dim(df)
```

```
## [1] 150   5
```

The column names show each flower's different features including the *sepal length*, *sepal width*, *petal length*, *petal width*, and *species*.

```
names(df)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

First 5 rows of the dataset:

```
head(df, n = 5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

1. Building Blocks of a Graph and `ggplot()` object:

Now, let's look at what the grammar of graphics contains. There are 8 key conponents in the grammar of graphics, and they serve as the building blocks for us to develop whatever graphical display we need. source:

- data
- aesthetic mapping
- geometric object
- statistical transformations
- scales
- coordinate system
- position adjustments
- faceting

In *ggplot2*, a `ggplot()` object includes all these conponents.
To demonstrate this, we need to create a `ggplot()` object:

```
ggdf <- ggplot(data = df)
```

If we inspect this `ggplot()` object, we can find out its type:

```
class(ggdf)
```

```
## [1] "gg"     "ggplot"
```

```
typeof(ggdf)
```

```
## [1] "list"
```

So now we can know that a `ggplot()` object is a list.

Moreover, it is a list cotaining the features of a graph:

```
names(ggdf)
```

```
## [1] "data"        "layers"      "scales"      "mapping"     "theme"
## [6] "coordinates" "facet"       "plot_env"    "labels"
```

Therefore, we can try to figure out how these features reflect those building blocks of graphs.

2. Aesthetic Mapping, Geometric Object, and `layers`

A layer contains of the following items Source:

- Data
- Aesthetic mapping (aes)
- A geometric object (geom)
- A statistical transformation (stat)
- Position adjustment (position)

First of all, a single `ggplot()` object by itself can give us nothing, since it misses an important conponent– a *geom* layer. .

```
ggdf
```

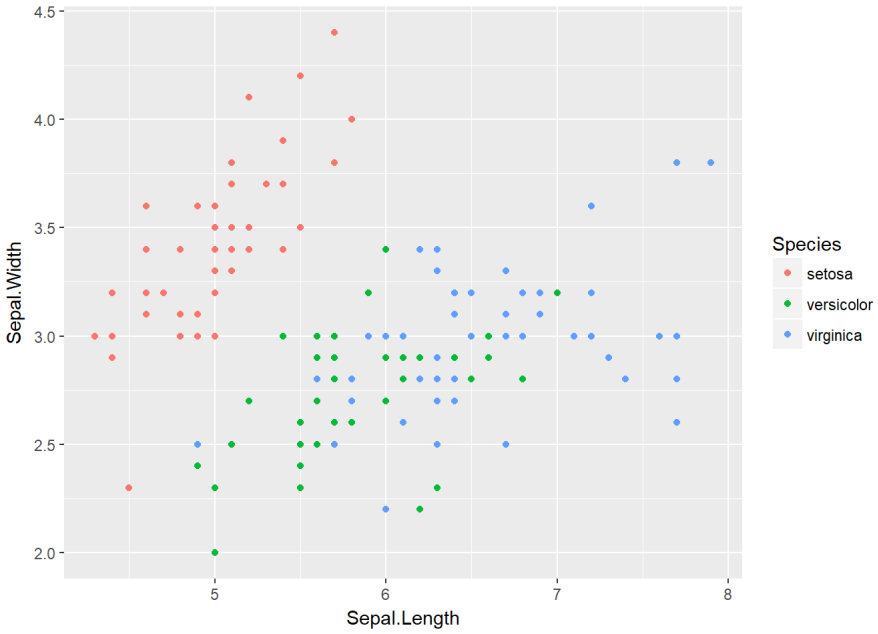

And when we inspect the `layers` of ggdf, we will find that it's an **empty** list:

```
ggdf$layers
```

```
## list()
```

Now, let's add a layer to *ggdf* to plot the sepal width and sepal length. Now *ggdf* become *ggdf_2*

```
ggdf_2 <- ggdf + geom_point(aes(x = Sepal.Length, y = Sepal.Width, color = Species))
ggdf_2
```



Look at the `layers` of *ggdf_2*:

```
ggdf_2$layers
```

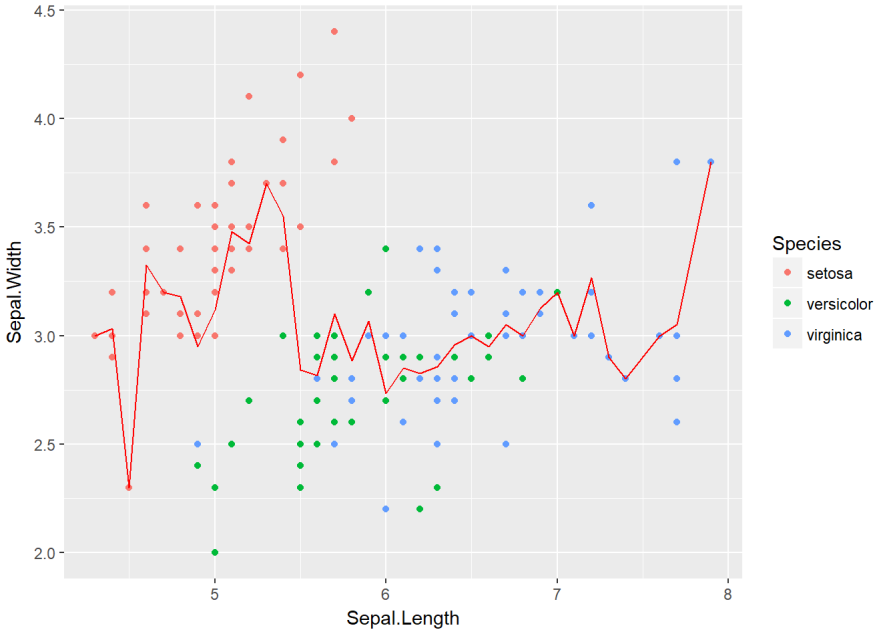
```
## [[1]]
## mapping: x = Sepal.Length, y = Sepal.Width, colour = Species
## geom_point: na.rm = FALSE
## stat_identity: na.rm = FALSE
## position_identity
```

The `layers` shows that we use points as our `geometric object`. Also, we are plotting Sepal.Length along the x-axis, and Sepal.Width along the y-axis. The colors of points are assigned according to their Species. It turns out that `layers` summarises the `aesthetic mapping` and `geometric object`. Again, as we have covered in lectures, geometric object determines what we will see in a graphic display, while aesthetic mapping determines how the points show be plotted. In other words, "the *aes()* is the 'how', and *geom* is the 'what'."

3. Statistical Transformation and `stat` function

In some case, we need some statistical transformations to provide better data visulizations. Some common statistical transformations includes mean, standard deviation, percentile, ect. In *ggplot2* package, we can use "stat function"

```
ggdf_3 <- ggplot(data = df, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point() +
  stat_summary(fun.y = mean, color = "red", geom = "line")
ggdf_3
```



The red line visualizes the mean of sepal width of each sepal length. Adding this line helps us identity which species has sepal widths above or below the means according the sepal lengths.
Now let's look at the layers of the `ggplot()` object *ggdf_3*

```
ggdf_3$layers
```

```
## [[1]]
## geom_point: na.rm = FALSE
## stat_identity: na.rm = FALSE
## position_identity
##
## [[2]]
## geom_line: na.rm = FALSE
## stat_summary: fun.data = NULL, fun.y = function (x, ...)
## UseMethod("mean"), fun.ymax = NULL, fun.ymin = NULL, fun.args = list(), na.rm = FALSE
## position_identity
```

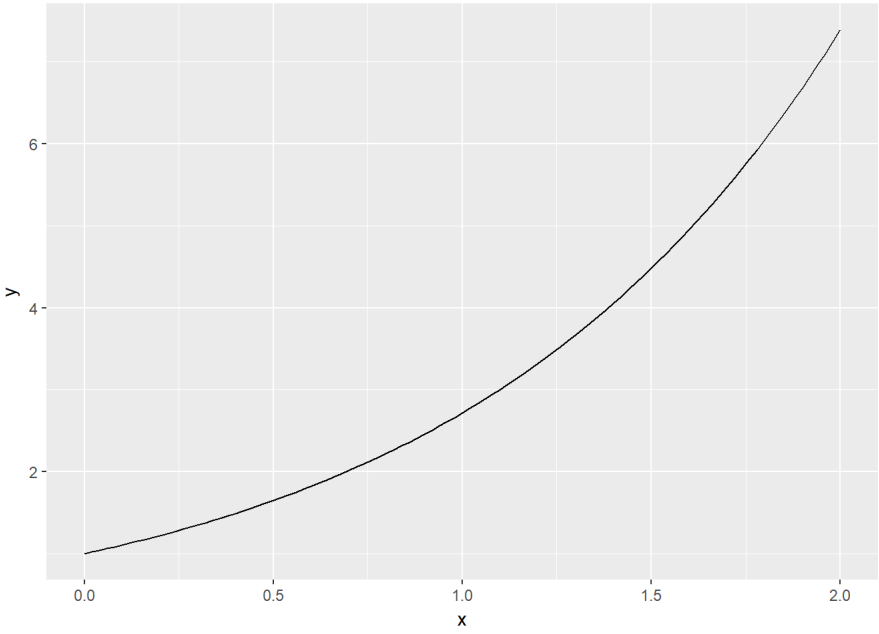
Comparing the `stat` in the two layers, we can find that the first layer set the statistics as identity (doesn't make any change), while the second layer apply a statistical transformation "mean" on the y-value. Hence, the layers also reflect the statistical transformation of the data in a graphic displays, and we can use `stat` function to accomplish this change.

What's more, we can use a handy function called `stat_function()` to generate the graphs of some common functions.

For instance: Source

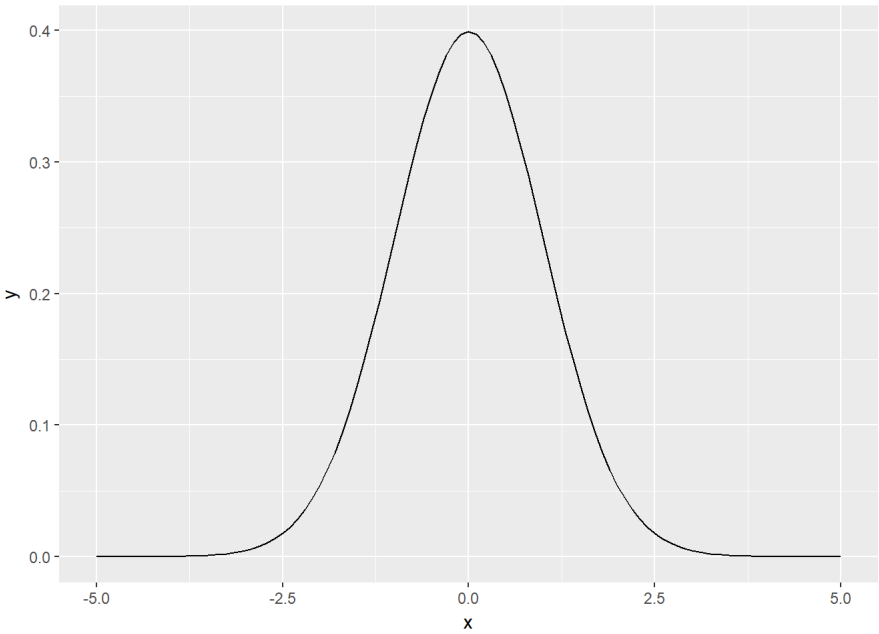
Exponential function

```
ggplot(data = data.frame(x = c(0, 2)), aes(x = x)) +
        stat_function(fun = exp, geom = "line")
```



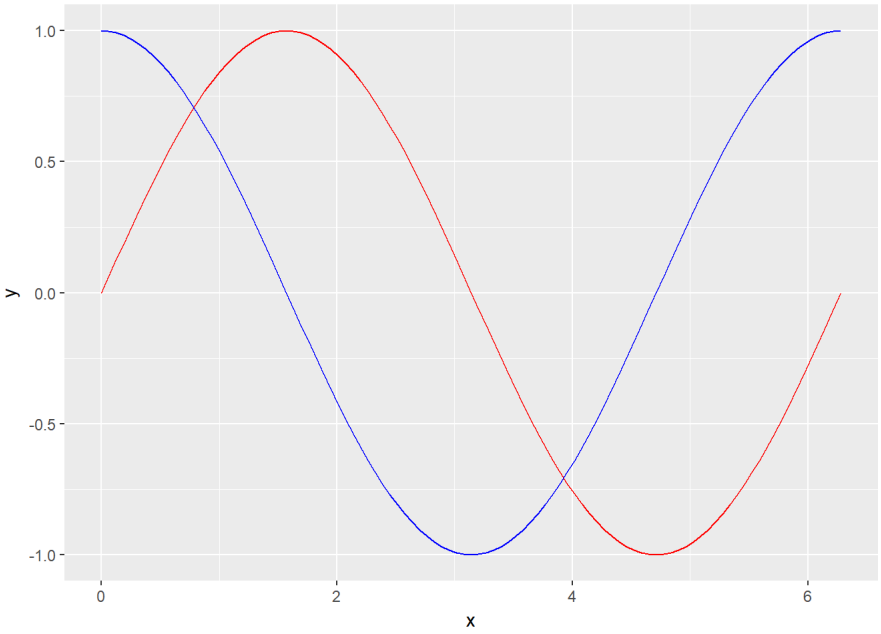
Normal density function

```
ggplot(data = data.frame(x = c(-5, 5)), aes(x = x)) +
  stat_function(fun = dnorm, args = list(mean = 0, sd = 1))
```



Sin & cos functions

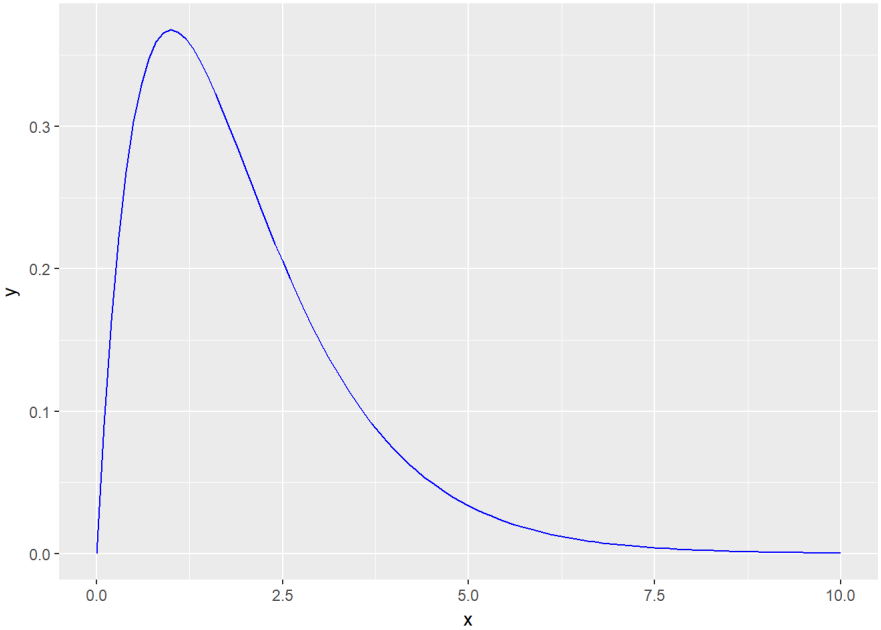
```
ggplot(data = data.frame(x = c(0, 2*pi)), aes(x)) +
  stat_function(fun = sin, color = "red") +
  stat_function(fun = cos, color = "blue")
```



You can even customize your own function.

For instance, the density function of a distribution \(gamma(r = 2, \lambda = 1)\)

```
my_function <- function(x) {x * exp(-x)}
ggplot(data = data.frame(x = c(0, 10)), aes(x)) +
  stat_function(fun = my_function, color = "blue")
```



If we check the `stat` in their `layers`, we can figure out how *ggplot* generates the plot:

```
my_gamma <- ggplot(data = data.frame(x = c(0, 10)), aes(x)) +
  stat_function(fun = my_function, color = "blue")
my_gamma$layers
```

```
## [[1]]
## geom_path: na.rm = FALSE
## stat_function: fun = function (x)
## {
##     x * exp(-x)
## }, n = 101, args = list(), na.rm = FALSE, xlim = NULL
## position_identity
```

Here, we can find that `stat_function()` applys the function \(xe^x\) to transform x-value, and then uses `geom_path()` to draw the line.

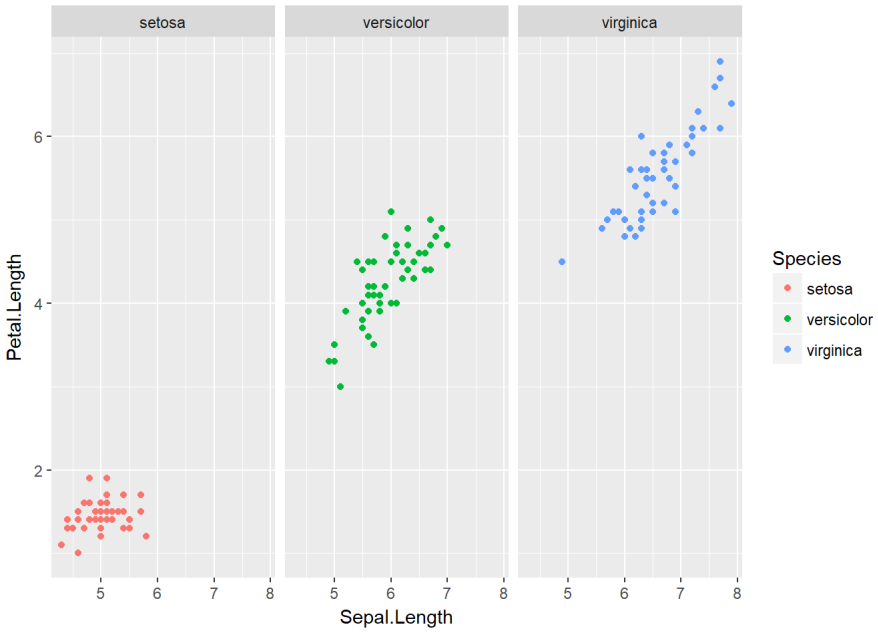
4. Faceting and `facet`

I want to briefly contrast two common facet methonds– `facet_wrap()` and `facet_grid()`. And then focus on `facet_wrap()` by inspecting corresponding `ggplot()` objects. From one of my sources, the differences between these two are stated as following (source):

> facet_wrap(): define subsets as the levels of a single grouping variable
> facet_grid(): define subsets as the crossing of two grouping variables

Looks like `facet_wrap()` can compare elements based on one single varialbes, while `facet_grid()` can base on two variables.
To understand them further, let's generate some plots.

```
ggplot(data = df) +
  facet_wrap(~Species) +
  geom_point(stat = "identity", aes(x = Sepal.Length, y = Petal.Length, color = Species))
```



Now, let's look at the `facet` in the `ggplot()` object

```
my_facet <- ggplot(data = df) +
  facet_wrap(~Species) +
  geom_point(stat = "identity", aes(x = Sepal.Length, y = Petal.Length, color = Species))
my_facet$facet
```

```
## <ggproto object: Class FacetWrap, Facet>
##     compute_layout: function
##     draw_back: function
##     draw_front: function
##     draw_labels: function
##     draw_panels: function
##     finish_data: function
##     init_scales: function
##     map: function
##     map_data: function
##     params: list
##     render_back: function
##     render_front: function
##     render_panels: function
##     setup_data: function
##     setup_params: function
##     shrink: TRUE
##     train: function
##     train_positions: function
##     train_scales: function
##     super:  <ggproto object: Class FacetWrap, Facet>
```

This looks unfamiliar to us, so let's figure out what `my_facet$facet` really is:

```
typeof(my_facet$facet)
```

```
## [1] "environment"
```

```
names(my_facet$facet)
```

```
## [1] "params" "shrink" "super"
```

It turns out that it is an `environment` with objects `params`, `shrink`, and `super`. Explaining what is an "environment" would be out of the scope of this post, but I found a relevant source for whoever is curious:source.
If we do further investigation, we can figure out where our fecet parameter ("Species," in this case) locates:

```
my_facet$facet$params$facets
```

```
## List of 1
##  $ Species: symbol Species
##  - attr(*, "env")=<environment: 0x000000001cd28db8>
##  - attr(*, "class")= chr "quoted"
```

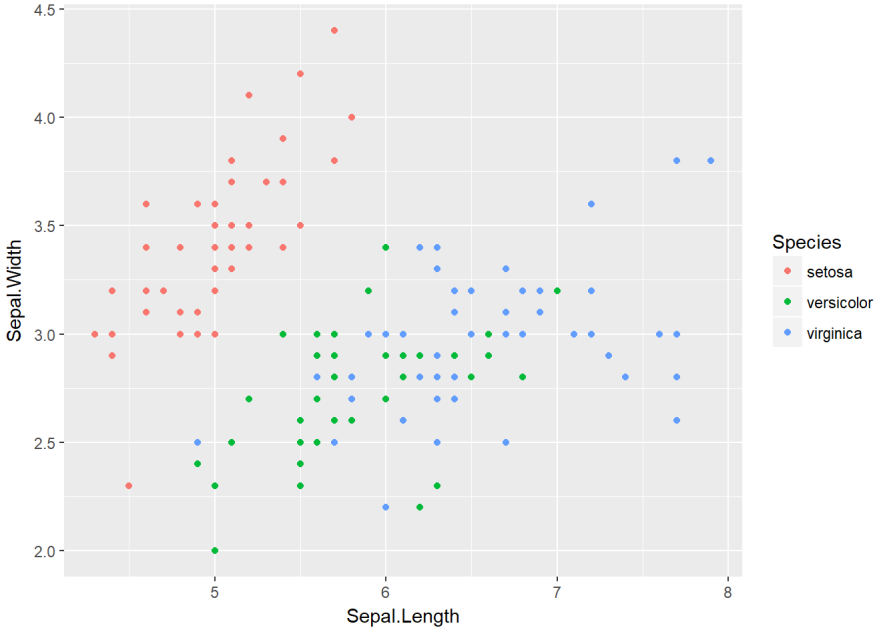
Cool! Now we know that plotting by group requires "ggplot" to define a new environment and then set certain parameters, which can be checked in the `facet` of a `ggplot()` object.

5. Coordinate systems and `coordinates`

As it is introduced here, both scales and coordinates system can finish the work of zooming a graphic display. However, as a tutorial clarifies, the difference is that transforming the coordinate system occurs *before* statistics, while coordinate transformation comes afterwards.
In this section, I want to talk about not only scaling and zooming, but also changing coordinate from Cartesian coordinates to polar coordinate, and then compare the `ggplot()` objects in each situation.

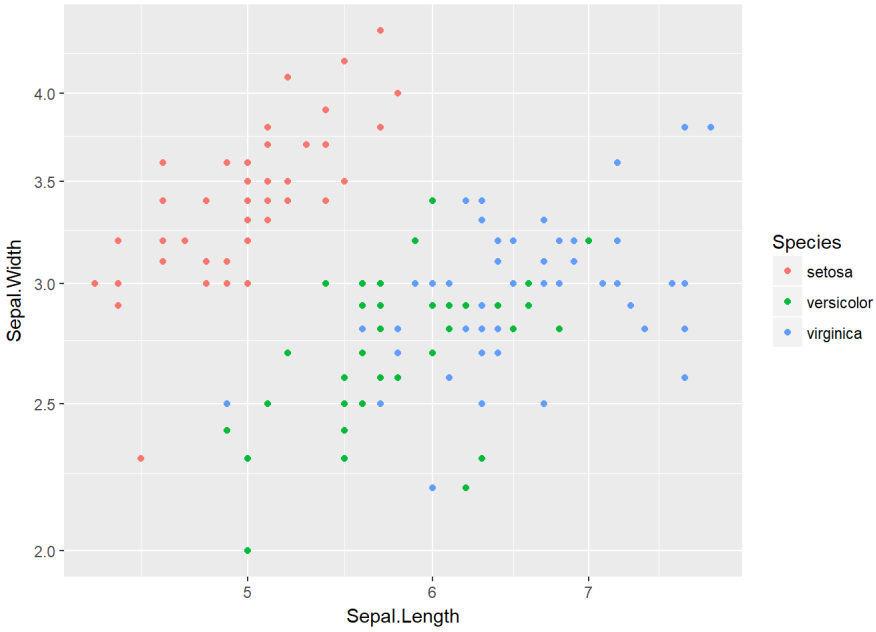
First of all, we have a plot that doesn't do any modification, so the plot is on the Cartesian coordinates.

```
ggdf_2
```



5.1 **Scale the x-axis and y-axis by coordinates**

```
ggdf_scale <- ggdf_2 + coord_trans(x = "log10", y = "log10")
ggdf_scale
```



Even though changing the scale of both x-axis and y-axis doesn't modify the positions of the points too much, we can still find the modification of the `trans` parameters in the `ggplot()` object.

```
ggdf_scale$coordinates$trans
```

```
## $x
## Transformer:  log-10
##
## $y
## Transformer:  log-10
```

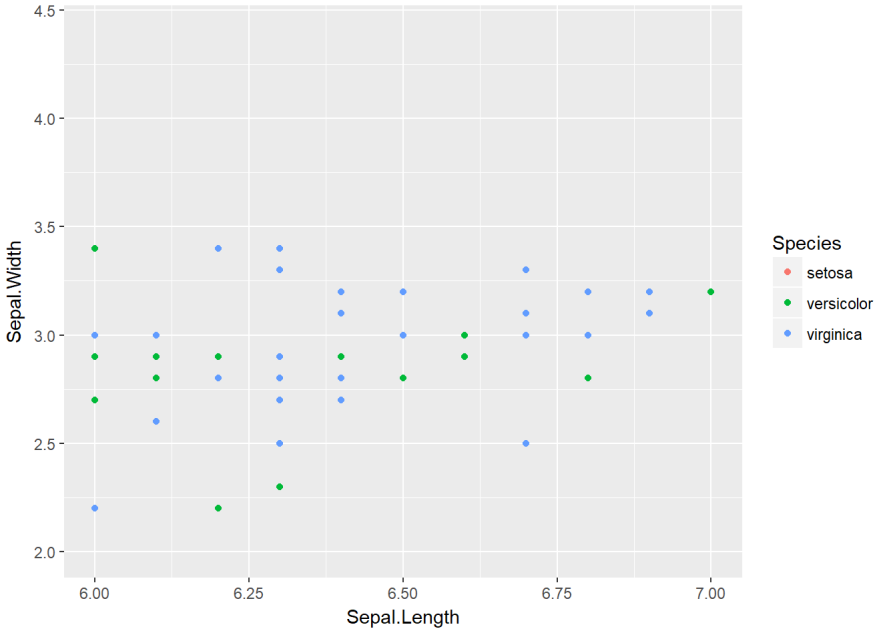
Note: Since we scale the plot by the coordinate system, we can go to the `coordinates` to check this change.

5.2 **Zoom in the plot**

We can zoom in the plot by setting a limitation of x-axis

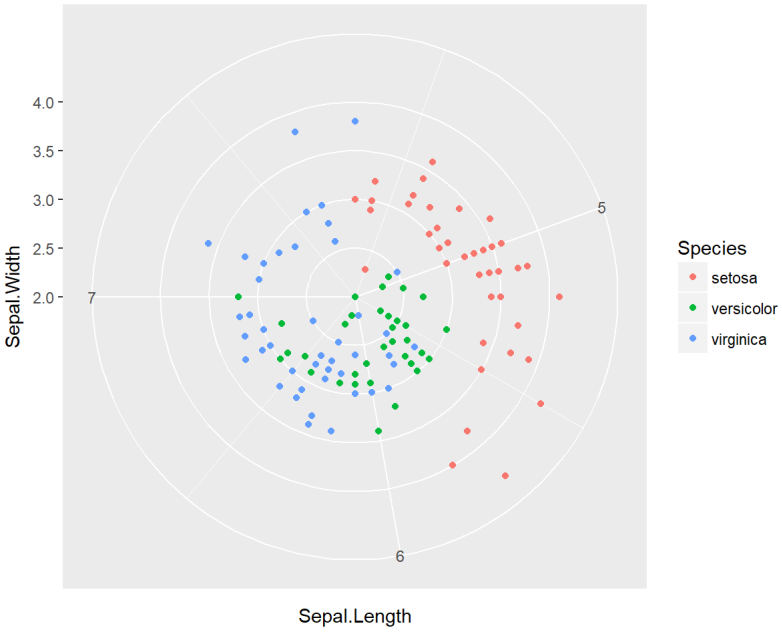
```
ggdf_zoom <- ggdf_2 + scale_x_continuous(limits = c(6, 7))
ggdf_zoom
```

```
## Warning: Removed 95 rows containing missing values (geom_point).
```



5.3 **Change to polar coordinate**

```
ggdf_polar <- ggdf_2 + coord_polar(theta = "x")
ggdf_polar
```




```
ggdf_polar$coordinates
```

```
## <ggproto object: Class CoordPolar, Coord>
##     aspect: function
##     direction: 1
##     distance: function
##     is_linear: function
##     labels: function
##     r: y
##     range: function
##     render_axis_h: function
##     render_axis_v: function
##     render_bg: function
##     render_fg: function
##     start: 0
##     theta: x
##     train: function
##     transform: function
##     super:  <ggproto object: Class CoordPolar, Coord>
```

Here, we can see that `coordinates` can show us what “Class” of coordinate we are using, and the corresponding paramters, like the angle \(\theta\) and radius \(r\).

6. Conclusion

Since the *ggplot2* package is the realization of the Grammar of Graphics, a `ggplot()` object contains all the “building blocks” for constructing any graphic display we need.
In this post, I have demostrated:

1. We can find “aesthetic mapping” and “geometric object” in `layers`;
2. The `stat` functions can realize “Statistical transformation,” and this transformation causes change in the `layers` of a `ggplot` object;
3. The `facet` of a `ggplot()` object reflects the faceting of a graph. Particularly, we can go to `my_facet$facet$params$facets` to check the method of faceting. (Note:‘my_facet’ is a name of a `ggplot()` object)
4. Modifications of the coordinate systems includes scaling, zooming, and changing coordinate systems. These changes are reflected by the `coordinates` in a `ggplot()` object.

7. References

1. http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html#what_is_the_grammar_of_graphics
2. http://stat405.had.co.nz/lectures/21-grammar-of-graphics.pdf
3. https://www.aridhia.com/technical-tutorials/the-fundamentals-of-ggplot-explained/
4. https://www.rdocumentation.org/packages/ggplot2/versions/2.2.1/topics/stat_function
5. http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html
6. http://adv-r.had.co.nz/Environments.html
7. http://ggplot2.tidyverse.org/reference/coord_cartesian.html
8. http://ggplot2.tidyverse.org/reference/coord_trans.html