Introduction to ggplot2

N. Matloff

January 11, 2013

1 Introduction

Hadley Wickham's **ggplot2** package is a very popular alternative to R's base graphics package. (Others include **lattice**, **ggobi** and so on.)

The **ggplot2** pacakge is an implementation of the ideas in the book, *The Grammar of Graphics*, by Leland Wilkison, whose goal was to set out a set of general unifying principles for the visualization of data. For this reason, **ggplot2** offers a more elegant and arguably more natural approach than does the base R graphics package.

The package has a relatively small number of primitive functions, making it relatively easy to master. But through combining these functions in various ways, a very large number of types of graphs may be produced. It is considered especially good in setting reasonable default values of parameters, and much is done without the user's asking. Legends are automatically added to graphs, for instance.

The package is quite extensive (only a few functions, but lots of options), and thus this document is merely a brief introduction.

2 Installation and Use

Download and install **ggplot2** with the usual **install.packages()** function, and then at each usage, load via **library()**. Here's what I did on my netbook:

```
# did once:
> install.packages("ggplot2","/home/nm/R")
# do each time I use the package (or set in .Rprofile)
> .libPaths("/home/nm/R")
> library(ggplot2)
```

3 Basic Structures

One operates in the following pattern:

- One begins with a call to **ggplot()**:

  ```
  > p <- ggplot(yourdataframe)
  ```

  or

  ```
  > p <- ggplot(yourdataframe,aes(yourargs))
  ```

  Here **yourdataframe** could have been read from a file, say using **read.table()**, or generated within the program.

  The result **p** is an R S3 object of class **"ggplot"**, consisting of a component named **data**, and other components containing information about the plot.

  Note that at this point, though, there is no plot; **p** won't be displayed until we print it.[1]

- One adds features to—or even changes—the plot via the **+** operator, which of course is an overloaded version of R's built-in **+**, the function **"+.ggplot"()**. One can do this repeatedly, e.g. to superimpose several curves on the same graph.

  Each invocation of **+** adds a new **layer** to the graph.

- The function **aes()** ("aesthetics") is used to map your data variables to graph attributes.

  Its first one or two arguments specify the data variables to be plotted, in one dimension (say for a histogram) or two dimensions (say for a scatterplot).

  The remaining arguments are name/value pairs, where the name is an attribute such as color and the value is a data variable. This could be, for example, to indicate a variable that will determine the color of any given point in the graph.

  One can set attributes in this way at various levels:

  - We can set attributes for the entire graph by calling **aes()** within our call to **ggplot()**, e.g. to specify which variables from our dataset we wish to plot.
  - We can set attributes specific to one of the + actions, such as specifying point color in a scatterplot.

  So for instance we could use **aes()** to specify our data variables either when we call **ggplot()**, so these variables will be used in all operations, or when we call, say, **geom_point()**, to indicate data variables for this specific operation.

There are various types of objects that can be used as the second operand for the + operator. Examples are:

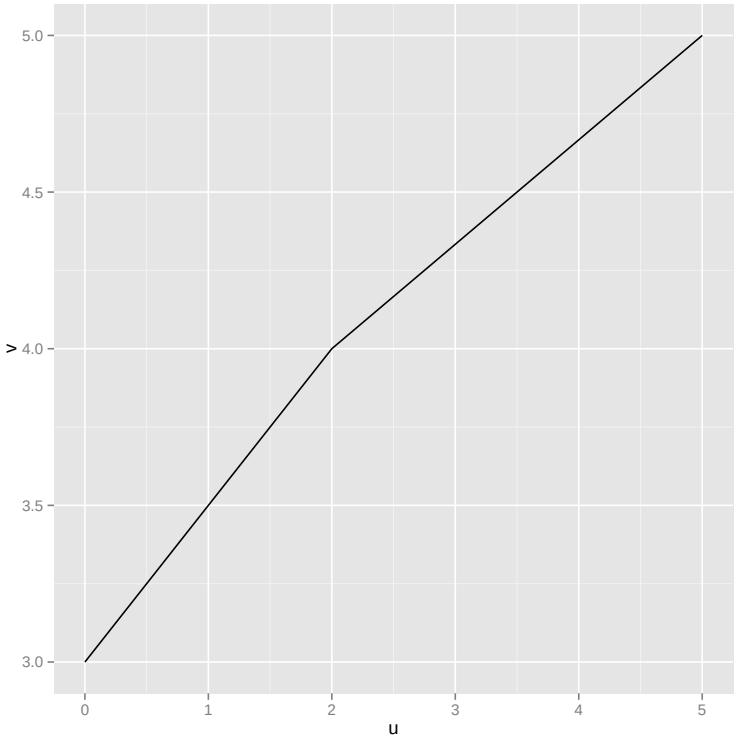
- **geoms** ("geometrics"): Geometric objects to be drawn, such as points, lines, bars, polygons and text.
- **position adjustments**: For instance, in a bar graph, this controls whether bars should be side by side, or stacked on top of each other.
- **facets**: Specifications to draw many graphs together, as panels in a large graph. You can have rows of panels, columns of panels, and rows and columns of panels.
- **themes**: Don't like the gray background in a graph? Want nicer labeling, etc.? You can set each of these individually, but one of the built-in themes, or a user-contributed one, can save you the trouble, or you can write one that you anticipate using a lot.

[1] In R, **print()** is a **generic** function, i.e. a stub for a class-specific one. In this case the latter does a plot.

4 Example: Simple Line Graphs

```
> df1
  u v
1 0 3
2 2 4
3 5 5
> ggplot(df1) + geom_line(aes(x=u,y=v))
```

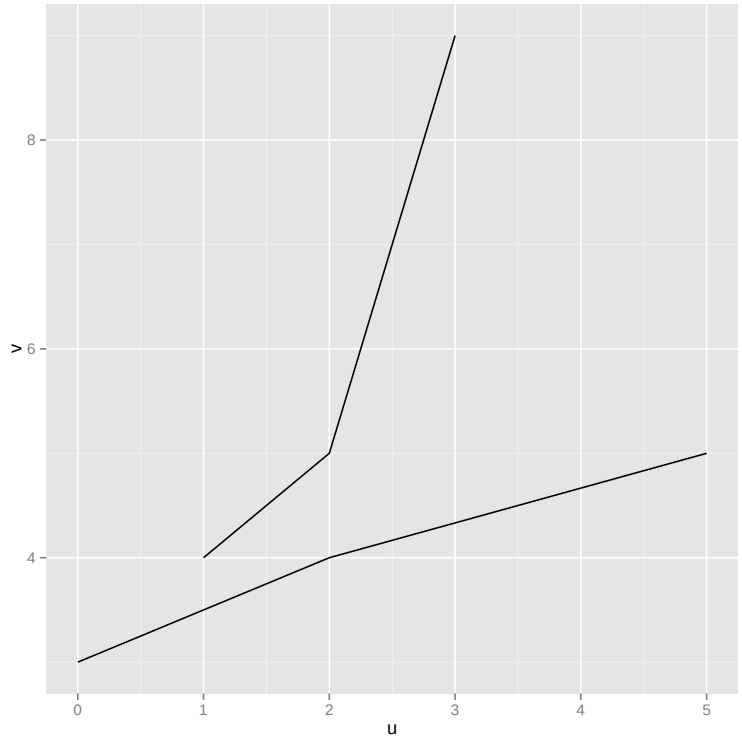
Here **aes()** was called from **geom_line()** rather than from **ggplot()**, so as to apply just to this line. The result is



Now let's add a second line, from a *different* data frame:

```
> df2
  w z
1 1 4
2 2 5
3 3 9
ggplot(df1) + geom_line(aes(x=u,y=v)) + geom_line(data=df2,aes(x=w,y=z))
```

Here is the result:



It worked as long as we specified **data** in the second line.

Note that **ggplot2** automatically adjusted that second graph, to make room for the "taller" second line.

5 Example: Census Data

The data set here consists of programmers (software engineers, etc.) and electrical engineers in Silicon Valley, in the 2000 Census. I've removed those with less than a Bachelor's degree. The R object was a data frame named **pm**.

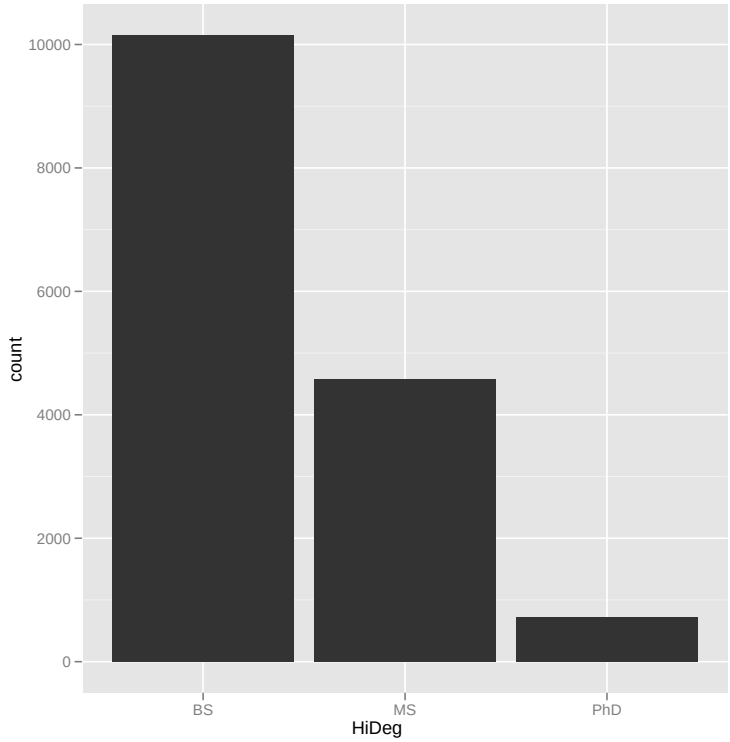
I first ran

```
p <- ggplot(pm)
```

to set up the **ggplot** object. Next, I typed

```
p + geom_histogram(aes(HiDeg))
```

which produced a histogram of the highest-degree values of the workers:



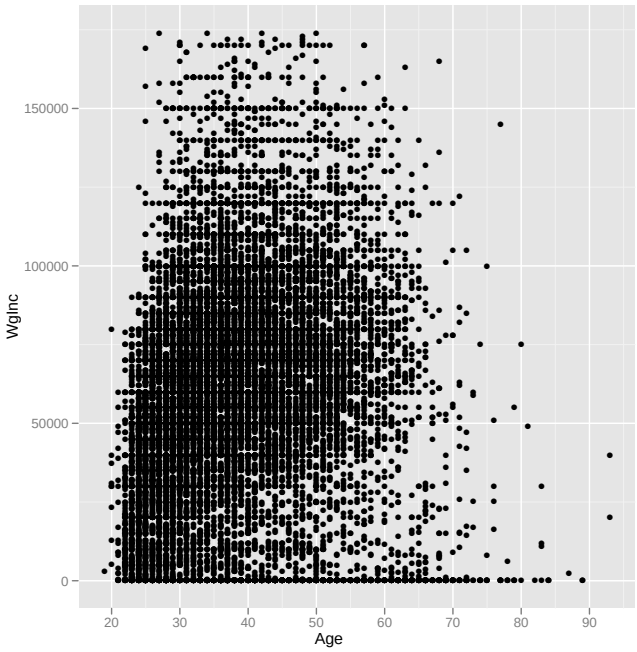
Note that the + operation yields a new object of class **"ggplot"**. Since the generic print function for that class actually plots the graph, the graph did appear on the screen. I could have saved the new object in a variable if needed.

I then decided to do a scatter plot of salary versus age:

```
> p + geom_point(aes(x=Age, y=WgInc))
```

Note the roles of **aes()** here and above; I used it to tell the geom what variable(s) to use, for instance telling **geom_point()** in the second exampl,e which of my data variables would correspond to the X- and Y-axes.

This gave me this graph:



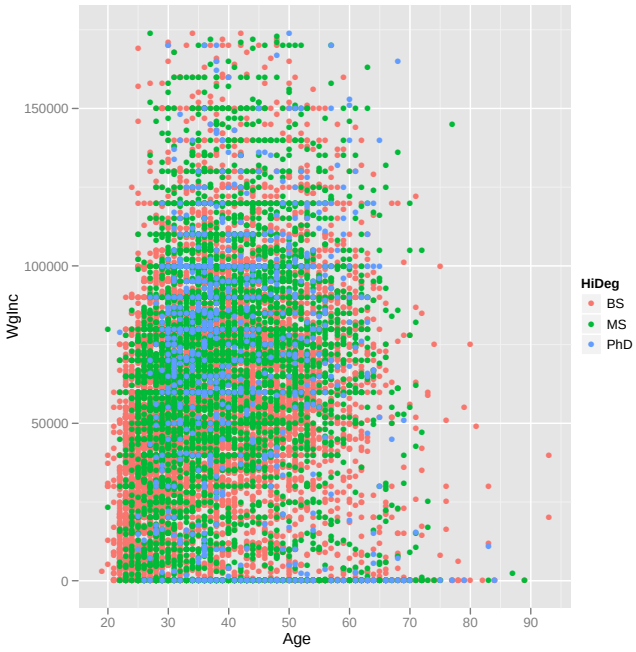
(As is often the case with large data sets, the points tend to "fill in" entire regions, one solution to which is to graph a random subset of the data, not done here. Similar comments apply to some of the graphs below.)

However, I wanted to separate the points according to highest degree level:

```
> p + geom_point(aes(x=Age, y=WgInc, color=HiDeg))
```

Here I have three data variables informing **aes()**: Age, wage income and highest degree. The argument **color** here means that I want the degree to be used for color coding the points:[2]

[2]Note that if I had wanted the same color to be set for *all* points, I'd set the **color** option outside the **aes()** call, as my second argument to **geom_point()**.

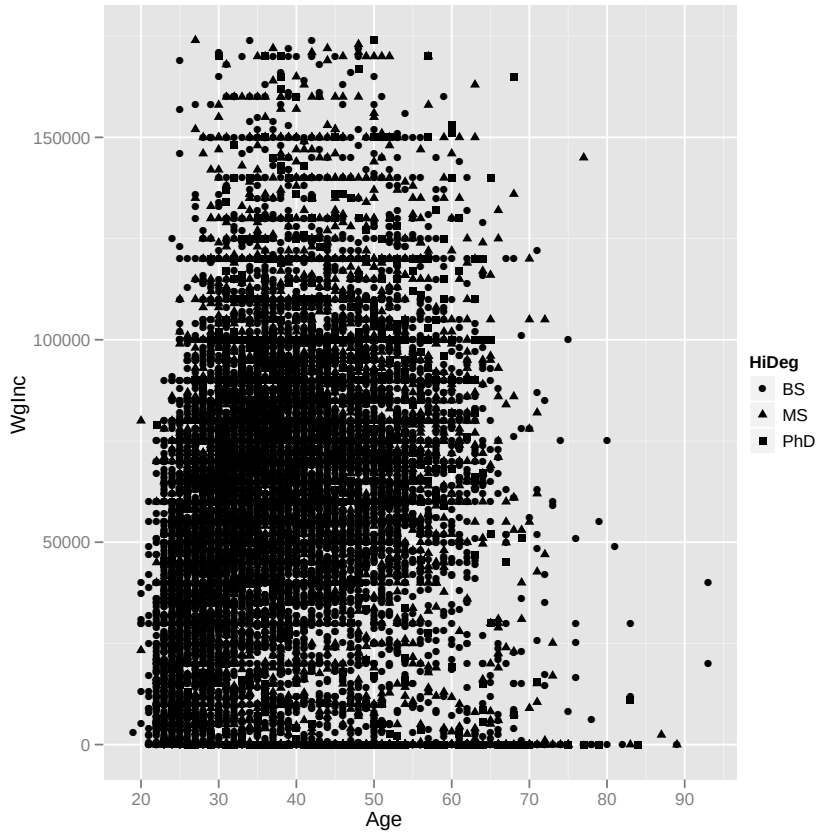


Note the legend that was automatically included on the right.

Since some people might be viewing a black-and-white version of this document, I ran the command again, specifying point shape instead of point color:

```
p + geom_point(aes(x=Age, y=WgInc, shape=HiDeg))
```

Here **ggplot2** decided to use a circle, a triangle and a square to represent Bachelor's, Master's and PhD workers:



Since I'm interested in age discrimination in the industry, I decided to restrict my graph to those over age 40. The **ggplot2** package cleverly exploits the R **subset()** function, allowing me to write

```
p %+% subset(pm, Age > 40) + geom_point(aes(x=Age, y=WgInc, color=HiDeg))
```

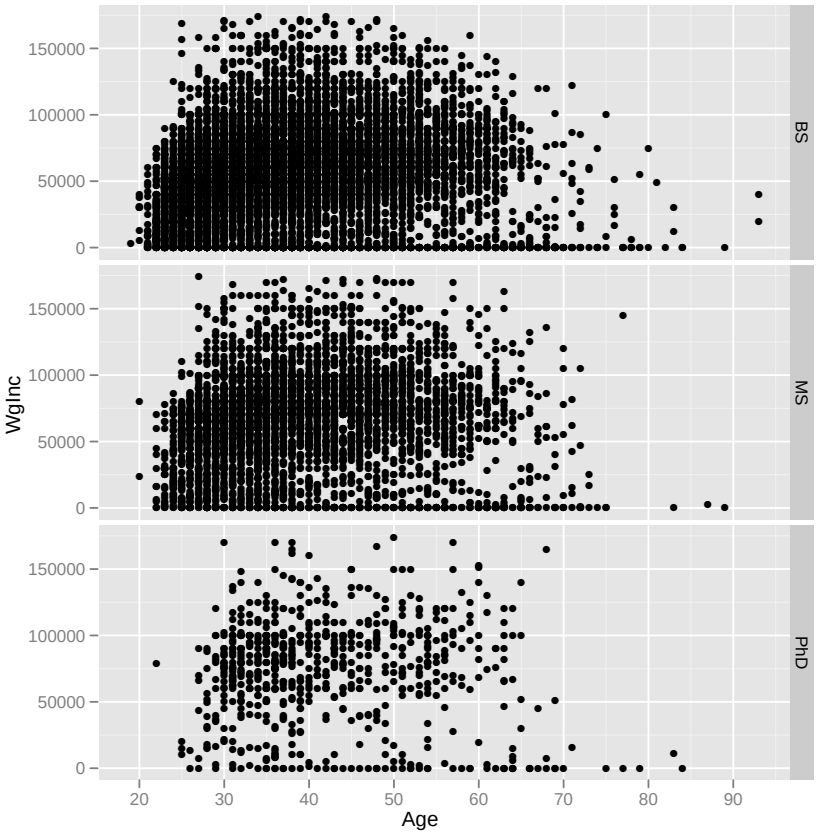
The new operator **%+%** is again mapped to **"+.ggplot"**(). The result was

Look at all those 0-income PhDs! (There was also a business income variable, which I did not pursue here, so maybe some had high incomes after all.)

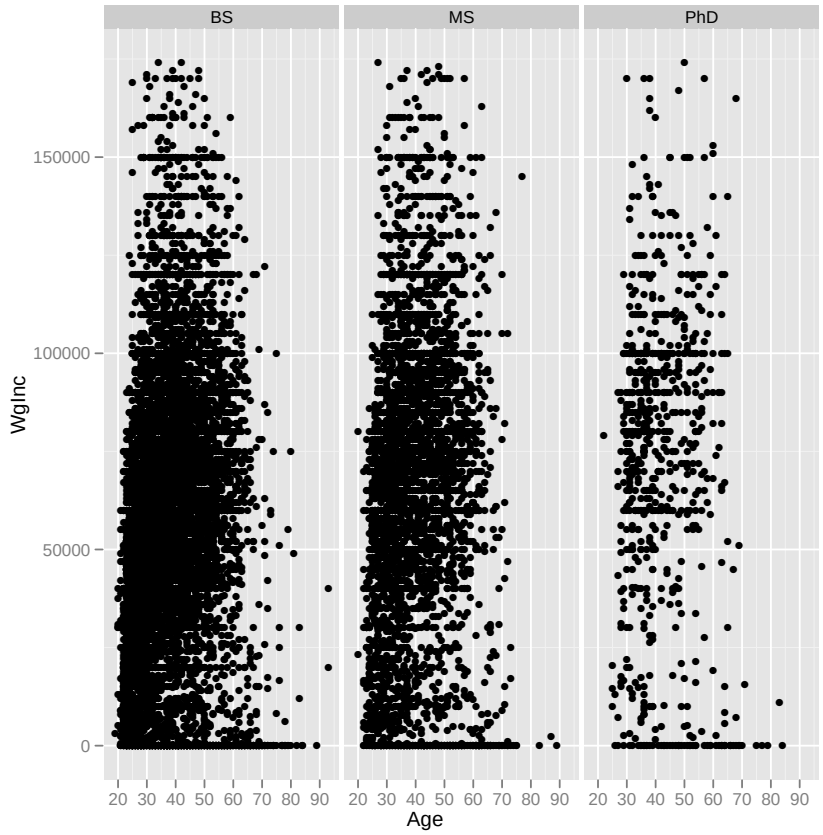
Even with color, it's a little hard to compare the three degree levels, so let's try faceting:

```
> pt <- p + geom_point(aes(x=Age, y=WgInc))
> pt + facet_grid(HiDeg ~ .)
```

Here I saved an overall graph of wage income versus age to **pt**, and then added faceting by degree. This instructed **ggplot2** to draw three graphs, one for each degree level, as three panels in the same large graph. The HiDeg ~ . argument (which you may recognize as similar notation to R's **lm()** function for linear models) states that I want HiDeg to define the rows of panels, without any variable defining columns. The result was:



I tried a horizontal presentation too:



Note that I could have combined operations, e.g.

```
> p + geom_point(aes(x=Age, y=WgInc)) + facet_grid(HiDeg ~ .)
```

to get the vertical set of panels, without directly going through the intermediate step of drawing the non-faceted graph.

If there had been a Gender variable in the data, I could have defined rows and columns of panels:

```
> p + geom_point(aes(x=Age, y=WgInc)) + facet_grid(Gender ~ HiDeg)
```

6 For Further Information

Just plugging "ggplot2 tutorial," "ggplot2 introduction," "ggplot2 examples" and so on into your favorite search engine will give you tons of information.

Hadley's book, *ggplot2: Elegant Graphics for Data Analysis*, is of course the definitive source, but also try his pictorial reference manual, at `http://had.co.nz/ggplot2/`.