

OLCF and NICS/RDAV Tutorial: Graphics with R using ggplot2*

George Ostrouchov^{†‡}, Pragnesh Patel[‡], and Drew Schmidt[‡]

June 28, 2012

Contents

1	Introduction	1
1.1	Plotting Data in R	1
1.2	What Is ggplot2 and why should I use it?	2
1.3	Ways of Plotting in ggplot2	2
2	ggplot2 Basics	2
2.1	Some Simple Plots	4
2.2	Saving	5
2.3	Exercises	5
3	Graphically Exploring the Diamonds Dataset with ggplot2	6
3.1	Exercises	6
4	Creating “new” Plots	7
4.1	Polar Bars and Pies	7
4.2	More about <code>opts()</code>	7
4.3	Plotting Regression Lines by Group	7
4.4	Heatmaps	8
4.5	Multiple Plots Without Faceting	9
5	Where to Learn More?	9
6	Suggested Solutions to Exercises	10
6.1	Solutions to Section 2 Exercises	10
6.2	Solutions to Section 3 Exercises	10

1 Introduction

1.1 Plotting Data in R

In addition to being a fantastic analytics engine, R is, in the author’s opinion, the *gold standard* in plotting.

1. Base Graphics: Simple for simple things, hard for complicated things. For a scatterplot `plot(x,y)`, barplot, `barplot(x)`, histogram `hist(x)`, piechart `piechart(x)`, . . .
2. Grid: “Assembly language for graphics”. Grid is cumbersome for pretty much everything; generally should be avoided unless you feel you have a good reason (you should not yet if you are reading this document)
3. ggplot2: More power and flexibility than base graphics, somewhat more cumbersome on simple plots
4. Lattice: “Like” ggplot2 in its scope, but operating under wildly different paradigms
5. Also 20 other packages: <http://cran.r-project.org/web/views/Graphics.html>

With so many excellent options for plotting our data, it is easy to get analysis paralysis in trying to decide which to use. Luckily for you, we have already decided to only talk about ggplot2. There are many good reasons in doing this, many of which would be difficult to make you appreciate just yet. Perhaps the most compelling argument is that it is *far* more powerful than base graphics while not being all that much less popular in use. More on that in the next section.

Further, we will only be using ggplot2 from the R console by learning the ggplot2 syntax. It is worth mentioning, however,

*Copyright © 2012 George Ostrouchov, Pragnesh Patel, and Drew Schmidt. All Rights Reserved.

[†]Oak Ridge National Laboratory, Computer Science and Mathematics Division, Scientific Data Group

[‡]National Institute for Computational Sciences, Remote Analysis and Visualization Center

that there is a very nice gui called Deducer (which is vaguely similar to SPSS) which can very easily make simple plots with ggplot2. To use Deducer, first start an R session and enter:

```
1 #Windows, Linux
2 install.packages(c("JGR", "Deducer", "DeducerExtras")) # only run
  this once
3
4 library(JGR)
5 JGR()
6
7 #####
8
9 # Mac
10 install.packages(c("JGR", "Deducer", "DeducerExtras", "CarbonEL")) #
  only run this once
11
12 Sys.setenv(NOAWT=1)
13 library(JGR)
14 Sys.unsetenv("NOAWT")
15 JGR()
```

Once the JGR console is loaded, you can load Deducer either by entering

```
1 library(Deducer)
```

in the JGR console¹, or by navigating the JGR drop-down menus to “Packages & Data” → ‘Package Manager’, and then by selecting Deducer and DeducerExtras.

For more details about Deducer, see the Deducer wiki: <http://www.deducer.org/pmwiki/pmwiki.php?n=Main.DeducerManual>

1.2 What Is ggplot2 and why should I use it?

The package ggplot2 is a package for R written by Hadley Wickham in 2005. It is an implementation of Leland Wilkinson’s “grammar of graphics”, which is a highly abstract data visualization “language”.

One of the most compelling reasons to use ggplot2 is that the high level of abstraction allows you to think about your data in new ways. The grammar of graphics allows one to go from thinking about canned routines for plotting data, to thinking about displaying relationships and associations in any way you can imagine. Of course, ggplot2 offers basic functionality as well. With ggplot2, if you can envision it, you can plot it.

1.3 Ways of Plotting in ggplot2

There are currently four main ways to produce plots with ggplot2:

1. qplot syntax — Shorthand. Meant to make learning ggplot2 easier, but not entirely equivalent (in some subtle respects) with the “longhand”; can make transitioning very confusing.
2. layer syntax — Nothing wrong with this approach, but it is extremely rare ‘in the wild’. If you learn this, you may find it difficult to get help, or make sense of the help you get.
3. geom/stat syntax — Probably the most popular way of using ggplot2.
4. autoplot syntax — Methods for plotting objects (in the OOP sense). None are implemented by default (you have to create them at the moment). Likely to become a prominent feature in the future, but for now very barebones.

For the remainder, we will only be discussing the geom syntax.

2 ggplot2 Basics

Before diving in to ggplot2 syntax, it would be wise to take a quick, abstract overview of ggplot2.

First, never forget that **ggplot2 operates on dataframes**. Not matrices, not vectors, and not lists (unless those lists

¹If you instead start deducer from the ordinary R console, you will experience some *very* odd behavior.

are actually dataframes). Dataframes and *only* dataframes. Once you have your dataframe (which might require some reshaping; the reshape2 package can be useful for this), the general procedure to produce a plot with ggplot2 is:

1. Declare which dataframe ggplot should use and set your aesthetics
2. Add layers via geom and/or stat functions
3. Options and extras

So what are aesthetics and geoms? A quick explanation is that in the context of the grammar of graphics, a “graphic” has a very well-defined meaning. Specifically, *a graphic is a mapping from **data** to **visual properties of geometric shapes***. There are four main components to a graphic:²

1. Geoms: Graphical objects such as point, path, line, polygon, ...
2. Aesthetics: For lack of a better term, these form the “visual cues”. They are geometric properties such as size, rotation, thickness, gradient, shape, color, ...
3. Coordinates: Just what it sounds like; e.g., rectangular, polar, ...
4. Faceting: Coplotting — more on this later

We will shortly see examples of how each of these components can affect a graph.

It is not an exaggeration to say that you can change *every* aspect of a ggplot2 plot. Figure 1 below³ details the differ-

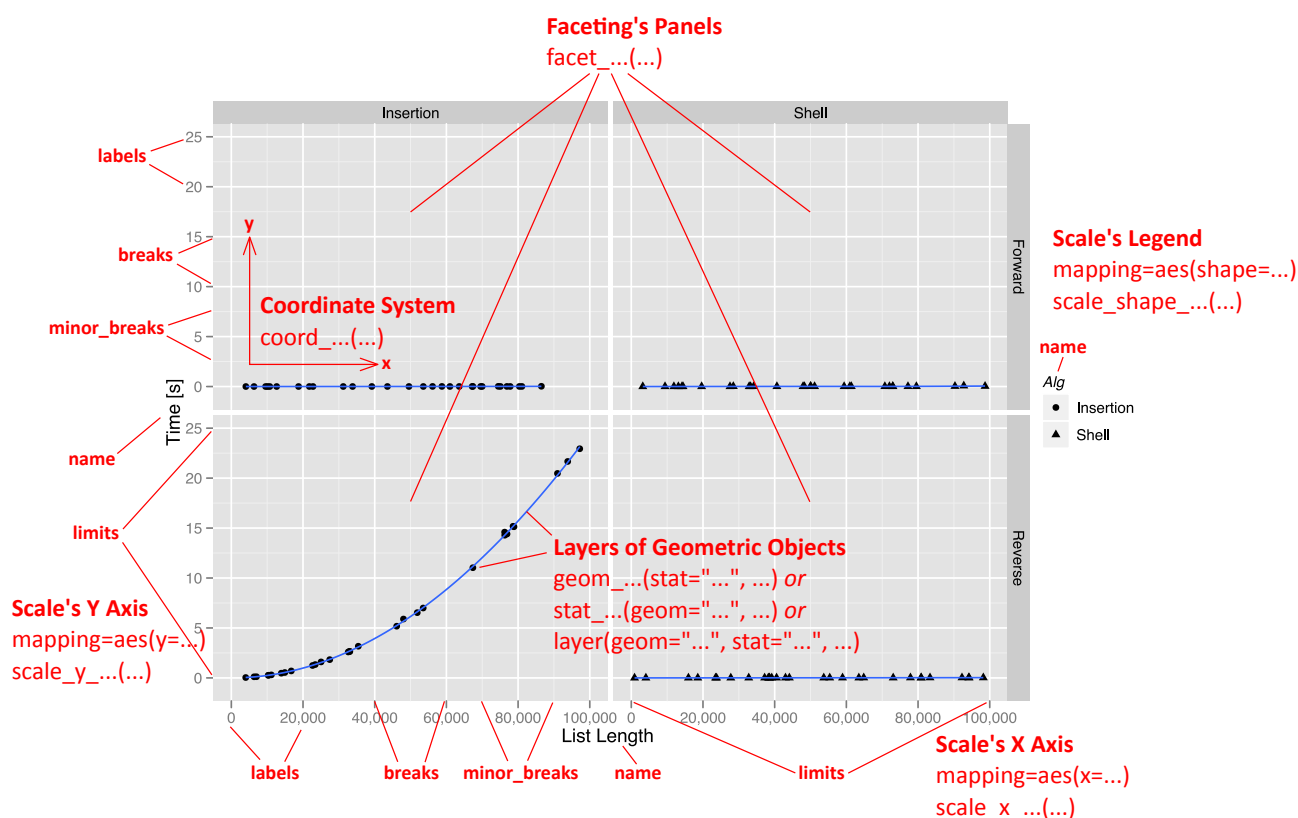


Figure 1: Anatomy of a ggplot2 Plot

ent components of a general ggplot2 plot. Many of these may seem unclear to you for now, but this plot is a fantastic reference for when you become more comfortable with ggplot2.

The way you add layers to a plot is by (literally) adding geom or stat functions. How many geom and stat functions are there? The table below provides a complete list for ggplot v. 0.9.1: So for example, to produce a barplot, you might make a call to `geom_bar()`. If there is a major downside to using ggplot, it's that it is so feature rich, it can almost be overwhelming even getting started. However, we will have many examples in the sections to follow which should hopefully alleviate some of this early anxiety.

When applying geom or stat functions, it is worth mentioning that the placement of layers in ggplot is not (necessarily) commutative, meaning that $A + B$ is not necessarily entirely equivalent to $B + A$. The ggplot2 package relies on a layering system, whereby new things sit on top of old things. This becomes especially important when you are doing lots of

²From Hadley Wickham <http://had.co.nz/ggplot/2006-auckland.pdf>

³Image reproduced with permission from the Sape Research Group. See <http://sape.inf.usi.ch/quick-reference/ggplot2>

Geom Functions					
geom_abline	geom_bar	geom_blank	geom_contour	geom_density	geom_errorbar
geom_freqpoly	geom_histogram	geom_jitter	geom_linerange	geom_point	geom_polygon
geom_rect	geom_rug	geom_smooth	geom_text	geom_vline	geom_area
geom_bin2d	geom_boxplot	geom_crossbar	geom_density2d	geom_errorbarh	geom_hex
geom_hline	geom_line	geom_path	geom_pointrange	geom_quantile	geom_ribbon
geom_segment	geom_step	geom_tile			
Stat Functions					
stat_abline	stat_bin2d	stat_boxplot	stat_density	stat_function	stat_identity
stat_quantile	stat_spoke	stat_summary	stat_vline	stat_bin	stat_binhex
stat_contour	stat_density2d	stat_hline	stat_qq	stat_smooth	stat_sum
stat_unique					

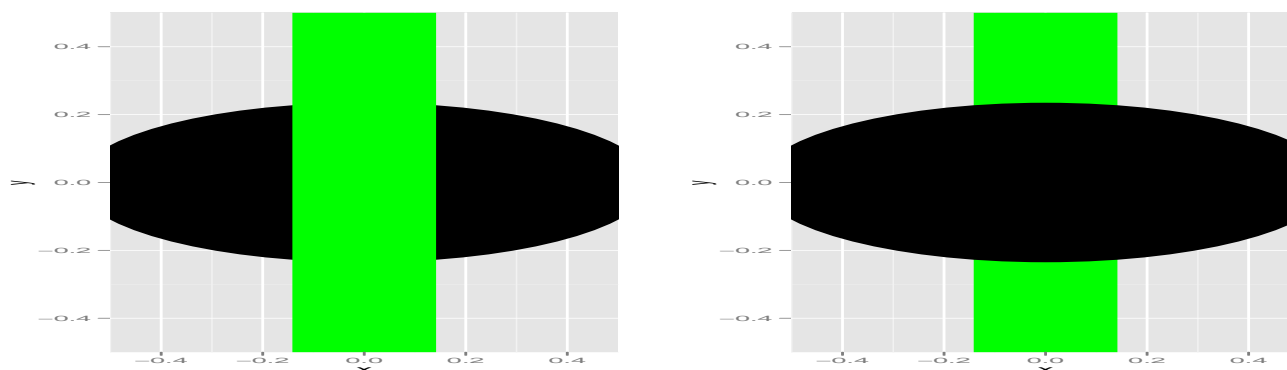
Layer Functions: <http://had.co.nz/ggplot2/>

interesting things with colors.

The process is much akin to stacking transparencies with different drawings on them. The sheets that get laid down first will, if there is overlap, be covered by the sheets that get laid down later. In ggplot2 syntax, this looks like

```
1 # Plot points layer then lines layer on top
2 g + geom_point() + geom_line()
3
4 # Plot lines layer then points layer on top
5 g + geom_line() + geom_point()
```

The figure below illustrates this effect. Here we draw a black point and a green vertical line. Notice that the order in which



we call the geom and stat functions has the obvious effect on the placement of objects in the plot. However, beyond this particular aspect, ggplot2 syntax is generally commutative, meaning that the order in which you call geom and stat functions is largely unimportant.

2.1 Some Simple Plots

Finally, let's look at some examples. Recall that we must first load the ggplot2 package before we can use it. Do this by entering

```
1 library(ggplot2)
```

into an R terminal. The ggplot2 package has several datasets built into it which are very useful for experimenting and learning ggplot2. To see the datasets available to you, enter

```
1 data() # see all available base datasets
2 data(package="ggplot2") # see datasets available via ggplot2
3 data(diamonds) # load the diamonds dataset
```

Before we start plotting, we should familiarize ourselves with the diamonds dataset. We can do this by examining the structure of the dataframe in several ways:

```

1 head(diamonds) # see the first few rows
2 nrow(diamonds) # count the number of rows
3 summary(diamonds) # get summary statistics
4 str(diamonds) # show the structure of the dataframe

```

We see that the diamonds dataset has 7 continuous variables: carat, depth, table, price, x, y, z, and 3 categorical variables: cut, color, clarity. Let's first investigate the 'clarity' variable. We can make a simple barplot with

Barplot

```

1 ggplot(data=diamonds, aes(x=clarity)) + geom_bar()

```

or see how the variable 'color' spreads across the 'clarity' variable

Barplot with Bars Filled by Another Categorical Variable

```

1 ggplot(data=diamonds, aes(x=clarity, fill=color)) + geom_bar()

```

Moving on to continuous variables, we can easily create a histogram:

Histogram

```

1 ggplot(data=diamonds, aes(x=price)) + geom_histogram()

```

Or get slightly more complicated, by looking at how a diamond's carat relates to its clarity

Plotting a Continuous by a Categorical Variable

```

1 g <- ggplot(data=diamonds, aes(x=clarity, y=carat))
2 g + geom_point() # scatterplot
3 g + geom_jitter() # scatterplot with jitter
4 g + geom_boxplot() # a boxplot

```

2.2 Saving

There are two ways to save your ggplot plots from the R console. Say you have a ggplot plot stored in the R object `g`. Then to save your plot, you could do either of the following:

```

1 # Option 1: ggplot2 independent
2 pdf("location/filename.pdf") # see help("device") for other
  filetypes
3 g # or last_plot() to save the last plot created by ggplot2
4 dev.off()
5
6 # Option 2: only for ggplot2 plots
7 ggsave("location/filename.pdf", g)

```

Both of these methods have many options, so you are encouraged to read the help files.

```

1 ?pdf
2 ?ggsave

```

2.3 Exercises

1. Create a histogram of the "carat" variable.
2. Save the plot you just made as both a pdf and a png.
3. Begin with:

```

1 g <- ggplot (data=diamonds, aes(x=clarity))

```

Produce a barplot and a histogram with `g` (remember, “clarity” is categorical). Is there a difference?

3 Graphically Exploring the Diamonds Dataset with ggplot2

To get more familiar with ggplot, let’s graphically investigate the relationship between the continuous variables ‘carat’ and ‘price’. First, we will store the basic information ggplot needs:

```
1 g <- ggplot(data=diamonds, aes(x=carat, y=price) )
```

Next, let’s look at some scatterplots.

Scatterplots of Price by Carat

```
1 # a simple scatterplot
2 g + geom_point()
3
4 # scatterplot with different colors/shapes by group(s)
5 g + geom_point(aes(color=color))
6 g + geom_point(aes(color=color, shape=cut))
7
8 # scatterplots with only part of the data
9 g_subset <- ggplot(data=subset(diamonds, color=="J"), aes(x=carat,
10 y=price) )
11 g_subset + geom_point()
12 g_subset + geom_point(aes(color=color))
```

We can even easily add a LOESS (local polynomial regression) curve to our plots

Scatterplot with a LOESS Fit

```
1 g + geom_point() +
2   geom_smooth()
3
4 g + geom_point() +
5   geom_smooth(se=FALSE) # no error band
```

Finally, let’s decompose the graph into individual *facets*. Say we aren’t just interested in the global relationship between carat and price, but the relationship between carat and price across individual cuts of diamond. There are 5 different cuts for a diamond, in increasing order of quality: fair, good, very good, premium, and ideal.

So for instance, we might want to see the scatterplot of carat and price for diamonds with an ideal cut. But instead of having to subset by each cut, we can plot them all at once in different sub-plots, using standardized axes using just one (of two) simple command(s).

Faceting by Cut

```
1 g + geom_point(aes(color=color)) +
2   facet_wrap(~cut)
3
4 g + geom_point(aes(color=color)) +
5   facet_grid(.~cut)
6
7 g + geom_point(aes(color=color)) +
8   facet_grid(clarity~cut)
```

3.1 Exercises

1. Create scatterplots of price by carat faceted by color. How would you describe the relationship between price and carat across groups?
2. Every plot should tell a story. What story do our scatterplots tell about a diamond’s carat and its price? (Just a short, one sentence explanation)
3. Refer to the subset plot above where we restricted the data only to those diamonds with color “J”. Produce a scatterplot

with a LOESS fit in the same plot. Do you notice anything striking in this plot (you may have noticed it in another plot above)?

4 Creating “new” Plots

Throughout this section, we are going to be looking at how one creates interesting plots for which just a single quick `geom_` or `stat_` function either will not do, or does not exist. We will also be introducing some of the great wealth of customization options available in `ggplot2`.

4.1 Polar Bars and Pies

The “Polar Bar” chart, also known as the “consultant’s plot”, is common in certain domains. Thinking about producing this plot in “pencil and paper” methods requires thinking about our data in an entirely new way. But really, at the end of the day, the plot is just a barplot wrapped in polar coordinates. And that’s exactly how you make it in `ggplot2`: by using the `coord_polar()` function, like so:

Polar Bar

```
1 ggplot(data=diamonds, aes(x=clarity, fill=color)) +  
2   geom_bar() +  
3   coord_polar()
```

This has just the fundamentals of a polar bar chart, really. We can make it look much more presentable by tweaking some `ggplot2` options:

A Fancy Polar Bar

```
1 ggplot(data=diamonds, aes(x=clarity, fill=color)) +  
2   geom_bar() +  
3   coord_polar() +  
4   scale_x_discrete(name="") + # no label on x-axis  
5   scale_y_continuous(name="") + # no label on y-axis  
6   theme_bw() + # remove the ggplot2 background  
7   opts(  
8     axis.ticks=theme_blank(), # no axis ticks  
9     legend.position="none", # remove legend  
10    axis.text.x=theme_blank(), # remove r-axis labels  
11    axis.text.y=theme_blank() # remove theta-axis labels  
12  )
```

Finally, we can make an ordinary piechart with just a few minor modifications

Piechart

```
1 ggplot(data=diamonds, aes(x=factor(1), fill=cut)) +  
2   geom_bar(width=1) + # fatter bars  
3   coord_polar(theta="y") # (x,y) -> (r,theta)
```

4.2 More about `opts()`

Before proceeding further, we would briefly take a moment to describe the `opts()` function. This is used for controlling theming options; and since just about anything you can think of is an option, it’s quite an extensive function. This is where you will be spending a lot of time “cleaning up” your plot for presentation, really giving it that extra edge.

For many years, the `ggplot2` help files within R (e.g., `help(opts)`) was notoriously useless. In one of the more recent editions, that has thankfully changed, so referring to the help file for `opts()` is good practice. You can also learn more about the many options at the `ggplot2` github wiki: <https://github.com/hadley/ggplot2/wiki/%2Bopts%28%29-List>

4.3 Plotting Regression Lines by Group

Recall that in section 3 we were interested in exploring the relationship between price and carat. We might be interested in fitting linear regression models to this data for each different color of diamond. Whether or not this type of modeling is appropriate for this data, we could proceed by first using R’s `lm` function to build our model:

Regression Lines by Group Part 1: Build the model

```
1 model <- lm(price ~ carat + color, data=diamonds)
```

That was easy! Next, we need to slightly restructure the data so ggplot2 knows what to do with it.

To do this, we, loosely speaking, have to construct a bunch of points which are going to be on each of the lines (one line for each group). The pairs will consist of a carat value and the corresponding price estimated from our linear regression model (really these are triplets, since we will also be storing with the aforementioned pair which color this pair corresponds to). We can do this easily with the functions `expand.grid()` and `predict()`.

Regression Lines by Group Part 2: Restructure the Data

```
1 grid <- with(diamonds ,
2   base::expand.grid(
3     carat = seq(min(carat), max(carat), length = 20),
4     color = levels(factor(color))
5   )
6 )
7
8 grid$price <- stats::predict(model, newdata=grid)
```

Finally, we can throw the data into ggplot. Notice that we are actually using multiple 2 dataframes: diamonds and grid.

Regression Lines by Group Part 3: Creating the plot

```
1 g <- ggplot(data=diamonds, aes(x=carat, y=price, color=color)) +
2   geom_point() +
3   geom_line(data=grid) +
4   ylim(0,22000)
5
6 g
```

This plot looks cluttered, so it might be best to facet out by our group variable “color”:

Faceted Regression Lines

```
1 g + facet_wrap(~color)
```

We can even add confidence bands:

Faceted Regression Lines with Confidence Bands

```
1 err <- stats::predict(model, newdata=grid, se = TRUE)
2 grid$ucl <- err$fit + 1.96 * err$se.fit
3 grid$lcl <- err$fit - 1.96 * err$se.fit
4
5 g + facet_wrap(~color) +
6   geom_smooth(aes(ymin = lcl, ymax = ucl), data=grid,
7     stat="identity")
```

4.4 Heatmaps

Using `geom_tile` allows us to quickly create a heatmap:

Fancy Heatmap of Depth by Cut and Color

```
1 g <- ggplot(diamonds, aes(cut, color)) +
2   geom_tile(aes(fill=depth), color="white")
```

We can make the heatmap look much more presentable with some of the many options in ggplot2:

Fancy Heatmap of Depth by Cut and Color

```
1 g + theme_bw() +
2   scale_fill_gradient(low="#2E2EFE", high="#FA5858") +
3   scale_x_discrete(expand=c(0,0)) +
4   scale_y_discrete(expand=c(0,0)) +
5   labs(x="", y="") +
6   opts(legend.position="none",
7     axis.ticks=theme_blank(),
8     axis.text.x=theme_text(size = 9,
```



```

9         angle = 320,
10        hjust = 0,
11        colour = "grey10"
12    ),
13    title="Depth of Diamond"
14 )

```

4.5 Multiple Plots Without Faceting

Using multiple plots in the same plotting window (not faceting) is not as easy as faceting. Generally, to do this you would use some functions from the package “grid”.

Multiple Plots Without Faceting

```

1 # Setting up the plots
2 g1 <- ggplot(data=diamonds, aes(x=clarity, y=price) )
3 g2 <- ggplot(data=diamonds, aes(x=price))
4
5
6 first <- g1 +
7     geom_boxplot() +
8     opts(title="Boxplots") +
9     scale_y_log10("Log-scale Price") +
10    scale_x_discrete("Clarity")
11
12 second <- g1 +
13     geom_jitter(size=1, aes(color=color)) +
14     opts(title="Jitter Plot") +
15     scale_x_discrete("Clarity") +
16     scale_y_continuous("Price")
17
18 third <- g2 +
19     geom_histogram(aes(y=..density..),
20                    color="black",
21                    fill="gray") +
22     geom_density(color="black") +
23     facet_wrap(~clarity) +
24     opts(title="Densities") +
25     scale_x_continuous("Price") +
26     scale_y_continuous("Density")
27
28 # Using grid to control placement of plots
29 library(grid) # load grid
30 grid.newpage() # ready the plotting device
31 pushViewport(viewport(layout = grid.layout(2, 2))) # set up plot
   layout, here a 2x2 grid
32 print(first, vp=viewport(layout.pos.row=1, layout.pos.col = 1)) #
   plot 'first' in square (1,1)
33 print(second, vp=viewport(layout.pos.row=2, layout.pos.col = 1)) #
   plot 'second' in square (2,1)
34 print(third, vp=viewport(layout.pos.row=1:2, layout.pos.col = 2)) #
   plot 'third' in rectangle (1-2, 2)

```

5 Where to Learn More?

The ggplot2 system is incredibly popular, so luckily if you are having trouble

Reference Manual:	http://had.co.nz/ggplot2/
CRAN page:	http://cran.r-project.org/web/packages/ggplot2/index.html
Wiki:	https://github.com/hadley/ggplot2/wiki/
Google Group:	https://groups.google.com/group/ggplot2
Tag on stackoverflow:	http://stackoverflow.com/questions/tagged/ggplot2
Blog:	http://blog.ggplot2.org/
Official Book:	http://tinyurl.com/ggplot2-book

6 Suggested Solutions to Exercises

6.1 Solutions to Section 2 Exercises

1.

```
1 ggplot(data=diamonds, aes(x=carat)) + geom_hist()
```

2.

```
1 ggsave("plot_pdf.pdf", last_plot()) # pdf
2 ggsave("plot_png.png", last_plot()) # png
```

3.

```
1 g + geom_bar()
2 g + geom_histogram()
```

6.2 Solutions to Section 3 Exercises

1.

```
1 ggplot(diamonds, aes(carat, price)) + geom_point() +
  facet_wrap(~color)
```

The relationship between price and carat looks similar across all colors.

2. There appears to be a fairly strong, positive association between a diamond's carat and its price.

3.

```
1 g_subset + geom_point() + geom_smooth()
```

The data appears to contain some outliers with extremely high leverage.