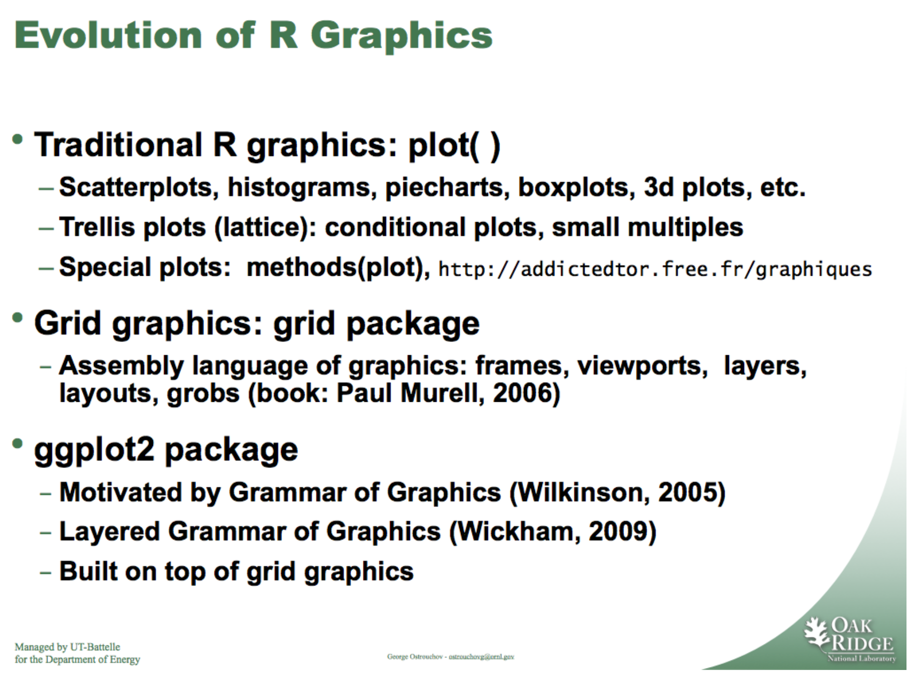
OLCF and NICS/RDAV Tutorial: Graphics with R using ggplot2

Prepared by Amy Szczepanski, Pragnesh Patel, and George Ostrouchov.

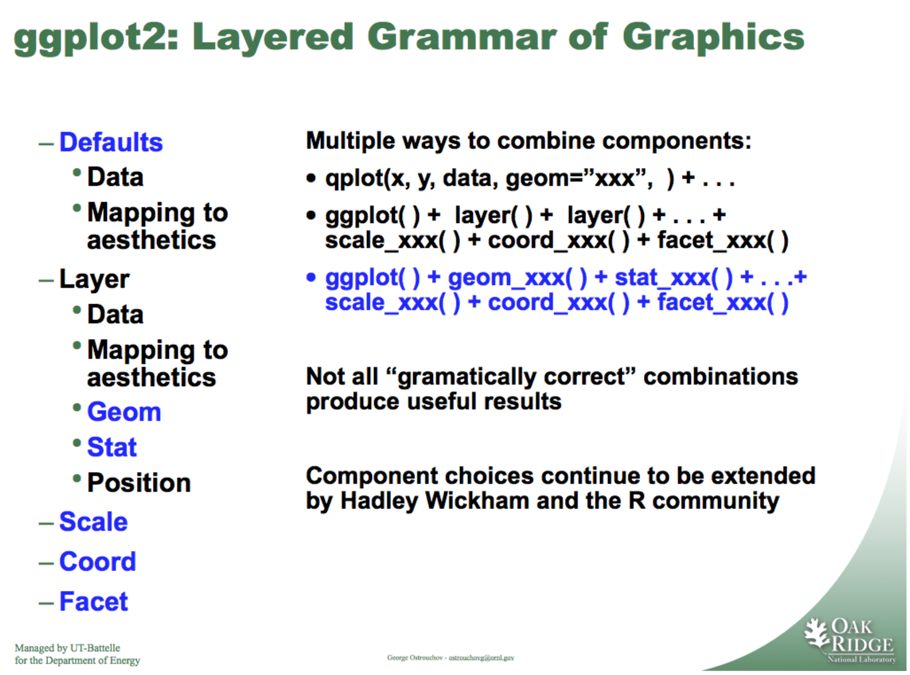
`http://olcf.ornl.gov/` and `http://rdav.nics.tennessee.edu/`

For this tutorial you should have installed R from `www.r-project.org`. Also, install the ggplot2 package if you do not have it already. Installation will depend on your OS.



Evolution of R Graphics

- **Traditional R graphics: plot( )**
  - Scatterplots, histograms, piecharts, boxplots, 3d plots, etc.
  - Trellis plots (lattice): conditional plots, small multiples
  - Special plots: methods(plot), `http://addictedtor.free.fr/graphiques`
- **Grid graphics: grid package**
  - Assembly language of graphics: frames, viewports, layers, layouts, grobs (book: Paul Murell, 2006)
- **ggplot2 package**
  - Motivated by Grammar of Graphics (Wilkinson, 2005)
  - Layered Grammar of Graphics (Wickham, 2009)
  - Built on top of grid graphics



ggplot2: Layered Grammar of Graphics

- Defaults
  - Data
  - Mapping to aesthetics
- Layer
  - Data
  - Mapping to aesthetics
  - Geom
  - Stat
  - Position
- Scale
- Coord
- Facet

Multiple ways to combine components:

- qplot(x, y, data, geom=”xxx”,) + . . .
- ggplot( ) + layer( ) + layer( ) + . . . + scale_xxx( ) + coord_xxx( ) + facet_xxx( )
- ggplot( ) + geom_xxx( ) + stat_xxx( ) + . . .+ scale_xxx( ) + coord_xxx( ) + facet_xxx( )

Not all “gramatically correct” combinations produce useful results

Component choices continue to be extended by Hadley Wickham and the R community

1. Launch R on your computer. Once R is running, give the command:

   ```
   library(ggplot2)
   ```

   Almost every example today will be based on the **diamonds** dataset that is included with the ggplot2 package.

2. We'll load the data set with `data(diamonds)`. This is a dataset that is built in to the ggplot2 package.

3. Let's try to understand this data. Remember that last time we used the following commands to get started.

   ```
   head(diamonds)
   summary(diamonds)
   str(diamonds)
   ```

4. There are several ways to make plots with ggplot2. You can make "quick" plots with `qplot()` or you can use the full power of the grammar with commands that build your graphs up in layers.

5. Remember, to get help on `qplot()`, give the command `?qplot`. To see some examples of graphs done with `qplot()`, you can give the command `example(qplot)`.

6. Let's get to know the data with some really basic graphs. As we saw from the results of the `str()` function, some of our variables are numbers and some of them are factors. We start with one variable at a time.

   ```
   qplot(cut, data=diamonds)
   qplot(price, data=diamonds)
   qplot(price, data=diamonds, binwidth=100)
   ```

7. Let's examine the relationship between the **price** of the diamond as a function of the "four Cs."

   ```
   qplot(carat, price, data=diamonds)
   qplot(carat, price, data=diamonds, alpha=I(1/10))
   ```

   Notice there are not many "almost" two carat diamonds but lots of two carat diamonds!

8. Let's keep adding more to the plot. A log transformation will bring carat and price distribution closer to uniform.

   ```
   qplot(carat, price, data=diamonds, log="xy")
   qplot(carat, price, data=diamonds, log="xy", facets = cut ~ color)
   qplot(carat, price, data=diamonds, log="xy", facets = cut ~ color, color=clarity)
   ```

9. To understand better what is going on, let's switch to the `ggplot()` function and the Layered Grammar of Graphics viewpoint.

   ```
   p <- ggplot(diamonds, aes(carat, price))
   p
   p + layer(geom="point")
   p + layer(geom="point", alpha=I(1/10)
   p + layer(geom="point") + scale_x_log10() + scale_y_log10()
   p + layer(geom="point") + scale_x_log10() + scale_y_log10() +
   facet_grid(cut ~ color)
   ```

   To map `clarity` to color, we first change the mapping via `aes()`

   ```
   p <- ggplot(diamonds, aes(carat, price, color=clarity))
   p + layer(geom="point") + scale_x_log10() + scale_y_log10()
   ```

10. So far we duplicated what we did earlier via `qplot()`. Let's add another layer.

```
p + layer(geom="point") + layer(stat="smooth") + scale_x_log10() + scale_y_log10()
p + layer(geom="point",stat="smooth") + scale_x_log10() + scale_y_log10()
p + layer(stat="identity", geom="point") + layer(stat="smooth", geom="smooth") + scale_x_log10() + sca
```

11. Because geoms and stats are the main components of a layer, there is another way to specify these. This is my favorite way to work with ggplot.

    ```
    p + geom_point() + stat_smooth() + scale_x_log10() + scale_y_log10()
    p <- p + scale_x_log10() + scale_y_log10()
    p + geom_point() + stat_smooth(method="lm)
    ```

12. The **geom**s that I use most frequently are: **point**, **jitter**, **smooth**, **histogram**, **boxplot**, and **bar**.

13. We can set the **color**, **fill**, **size**, and **shape**.

14. Let's use `qplot()` to look at the relationship between **clarity** and **price**, once with **jitter** as the **geom** and once with **boxplot**.

    ```
    p <- ggplot(diamonds, aes(clarity, carat))
    p + geom_point()
    p + geom_jitter()
    p + geom_boxplot()
    p + stat_boxplot()
    ```

15. Instead of using the entire data set, we can also work with a **subset**. Instead of having `data=diamonds`, we could have instead **data = subset(diamonds, carat==1)**. Try these on your own:

    ```
    p <- ggplot(subset(diamonds, carat == 1), aes(clarity, price))
    p + geom_boxplot()
    p + geom_boxplot() + geom_jitter()
    p + geom_jitter() + geom_boxplot()
    p + geom_jitter() + geom_boxplot() + facet_wrap(˜cut)
    p + geom_jitter() + geom_boxplot() + facet_grid(color˜cut)
    ```

16. As we talked about last time, everything in R is an object. These graphs are no different!

    ```
    p <- ggplot(diamonds, aes(clarity, price))
    str(p)
    summary(p)
    p <- p + geom_point()
    summary(p)
    ```

17. The **geom**s include: abline, area, bar, bin2d, blank, boxplot, contour, crossbar, density, density2d, errorbar, errorbarh, freqpoly, hex, histogram, hline, jitter, line, linerange, path, point, pointrange, polygon, quantile, rect, ribbon, rug, segment, smooth, step, text, tile, and vline. If your R installation has tab completion, you can see the whole list of **geom**s by typing `geom` and then tab. Not every **geom** works for every type of data.

18. Some interesting available **scale** include: alpha, size, and shape.

19. The **coord** include: cartesian, flip, polar, equal, map, and trans. Some of these require further parameters.

20. the available **aesthetics** depend on the **geom**. They often include: x, y, color (colour), fill, linetype, size, weight, xmin, xmax, ymin, and ymax.

21. What kind of adjustments can we make to our graph, especially as we are preparing it for publication? Consider the following graph:

```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point()
```

We can change the theme from a grey background to a white background:

```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point() + theme_bw()
```

The two built-in themes are `theme_bw()` and `theme_gray()`. We can set the theme for all graphs from this point forward with `theme_set()`.

```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point() + opts(title="This is our
  graph", plot.title=theme_text(colour="red", size=20))
ggplot(diamonds, aes(x=cut)) + geom_bar() + opts(axis.text.x=theme_text(hjust=0,
  angle=-90))
```

22. Recall from last time that we have several options for saving our graphs, including the `pdf()` and `png()` functions.