```
Course: Exploratory_Data_Analysis
Lesson: GGPlot2_Part2

Output: "GGPlot2_Part2. (Slides for this and other Data Science courses may be found at github https://github.com/DataScienceSpecialization/courses/. If you care to use them, they must be downloaded as a zip file and viewed locally. This lesson corresponds to 04_ExploratoryAnalysis/ggplot2.)"


- Class: text
  Output:  In a previous lesson we showed you the vast capabilities of qplot, the basic workhorse function of the ggplot2 package. In this lesson we'll focus on some fundamental components of the package. These  underlie qplot which uses default values when it calls them. If you understand these building blocks, you will be better able to customize your plots. We'll use the second workhorse function in the package, ggplot, as well as other graphing functions.

- Class: mult_question
  Output:  Do you remember what the gg of ggplot2 stands for?
  AnswerChoices: goto graphics; grammar of graphics; great graphics; good grief
  CorrectAnswer: grammar of graphics
  AnswerTests: omnitest(correctVal='grammar of graphics')
  Hint: Think about nouns, verbs, and adjectives.

- Class: text
  Output:  A "grammar" of graphics means that ggplot2 contains building blocks with which you can create your own graphical objects. What are these basic components of ggplot2 plots? There are 7 of them.

- Class: text
  Output: Obviously, there's a DATA FRAME which contains the data you're trying to plot. Then the AESTHETIC MAPPINGS determine how data are mapped to color, size, etc. The GEOMS (geometric objects) are what you see in the plot (points, lines, shapes) and FACETS are the panels used in conditional plots. You've used these or seen them used in the first ggplot2 (qplot) lesson.

- Class: text
  Output: There are 3 more. STATS are statistical transformations such as binning, quantiles, and smoothing which ggplot2 applies to the data. SCALES show what coding an aesthetic map uses (for example, male = red, female = blue). Finally, the plots are depicted on a COORDINATE SYSTEM. When you use qplot these were taken care of for you.

- Class: mult_question
  Output:  Do you remember what the "artist's palette" model means in the context of plotting?
  AnswerChoices: we draw pictures; we mix paints; plots are built up in layers; things get messy
  CorrectAnswer: plots are built up in layers
  AnswerTests: omnitest(correctVal='plots are built up in layers')
  Hint: Think about layers and creating a picture in several steps.

- Class: text
  Output: As in the base plotting system (and in contrast to the lattice system), when building plots with ggplot2,  the plots are built up in layers, maybe in several steps. You can plot the data, then overlay a summary (for instance, a regression line or smoother) and then add any metadata and annotations you need.

- Class: cmd_question
  Output: We'll keep using the mpg data that comes with the ggplot2 package. Recall the versatility of qplot. Just as a refresher, call qplot now with 5 arguments. The first 3 deal with data - displ, hwy, and data=mpg. The fourth is geom set equal to the concatenation of the two strings, "point" and "smooth". The fifth is facets set equal to the formula .~drv. Try this now.
  CorrectAnswer: qplot(displ, hwy, data = mpg, geom=c("point", "smooth"),facets=.~drv)
  AnswerTests: omnitest(correctExpr='qplot(displ, hwy, data = mpg, geom=c("point", "smooth"),facets=.~drv)')
  Hint: Type qplot(displ, hwy, data = mpg, geom=c("point", "smooth"),facets=.~drv) at the command prompt.
```

```
- Class: text
  Output:  We see a 3 facet plot, one for each drive type (4, f, and r). Now we'll see
  how ggplot works. We'll build up a similar plot using the basic components of the
  package. We'll do this in a series of steps.

- Class: cmd_question
  Output: First we'll create a variable g by assigning to it the output of a call to
  ggplot with 2 arguments. The first is mpg (our dataset) and the second will tell
  ggplot what we want to plot, in this case, displ and hwy. These are what we want our
  aesthetics to represent so we enclose these as two arguments to the function aes. Try
  this now.
  CorrectAnswer: g <- ggplot(mpg,aes(displ,hwy))
  AnswerTests: expr_creates_var("g"); omnitest(correctExpr='g <-
  ggplot(mpg,aes(displ,hwy))')
  Hint: Type g <- ggplot(mpg, aes(displ,hwy) ) at the command prompt.

- Class: text
  Output: Notice that nothing happened? As in the lattice system, ggplot created a
  graphical object which we assigned to the variable g.

- Class: cmd_question
  Output: Run the R command summary with g as its argument to see what g contains.
  CorrectAnswer: summary(g)
  AnswerTests: omnitest(correctExpr='summary(g)')
  Hint: Type summary(g) at the command prompt.

- Class: text
  Output: So g contains the mpg data with all its named components in a 234 by 11
  matrix. It also contains a mapping, x (displ) and y (hwy) which you specified, and no
  faceting.
  CorrectAnswer: print(g)
  AnswerTests: omnitest(correctExpr='print(g)')
  Hint: Typeprint(g) at the command prompt.

- Class: cmd_question
  Output: Note that if you tried to print g with the expressions g or print(g) you'd
  get an error! Even though it's a great package, ggplot doesn't know how to display
  the data yet since you didn't specify how you wanted to see it. Now type
  g+geom_point() and see what happens.
  CorrectAnswer: g+geom_point()
  AnswerTests: omnitest(correctExpr='g+geom_point()')
  Hint: Type g+geom_point() at the command prompt.

- Class: cmd_question
  Output: By calling the function geom_point you added a layer. By not assigning the
  expression to a variable you displayed a plot. Notice that you didn't have to pass
  any arguments to the function geom_point. That's because the object g has all the
  data stored in it. (Remember you saw that when you ran summary on g before.) Now use
  the expression you just typed (g + geom_point()) and add to it another layer, a call
  to geom_smooth(). Notice the red message R gives you.
  CorrectAnswer: g+geom_point()+geom_smooth()
  AnswerTests: omnitest(correctExpr='g+geom_point()+geom_smooth()')
  Hint: Type g+geom_point()+geom_smooth() at the command prompt.

- Class: cmd_question
  Output: The gray shadow around the blue line is the confidence band. See how wide it
  is at the right? Let's try a different smoothing function. Use the up arrow to
  recover the expression you just typed, and instead of calling geom_smooth with no
  arguments, call it with the argument method set equal to the string "lm".
  CorrectAnswer: g+geom_point()+geom_smooth(method="lm")
  AnswerTests: omnitest(correctExpr='g+geom_point()+geom_smooth(method="lm")')
  Hint: Type g+geom_point()+geom_smooth(method="lm") at the command prompt.

- Class: cmd_question
  Output: By changing the smoothing function to "lm" (linear model) ggplot2 generated a
  regression line through the data. Now recall the expression you just used and add to
  it another call, this time to the function facet_grid. Use the formula . ~ drv as it
  argument. Note that this is the same type of formula used in the calls to qplot.
```

```
  CorrectAnswer: g+geom_point()+geom_smooth(method="lm") + facet_grid(.~drv)
  AnswerTests: omnitest(correctExpr='g+geom_point()+geom_smooth(method="lm") +
  facet_grid(.~drv)')
  Hint: Type g+geom_point()+geom_smooth(method="lm") + facet_grid(.~drv) at the command
  prompt.

- Class: text
  Output: Notice how each panel is labeled with the appropriate factor. All the data
  associated with 4-wheel drive cars is in the leftmost panel, front-wheel drive data
  is shown in the middle panel, and rear-wheel drive data in the rightmost. Notice that
  this is similar to the plot you created at the start of the lesson using qplot. (We
  used a different smoothing function than previously.)

- Class: cmd_question
  Output: So far you've just used the default labels that ggplot provides. You can add
  your own annotation using functions such as xlab(), ylab(), and ggtitle(). In
  addition, the function labs() is more general and can be used to label either or both
  axes as well as provide a title. Now recall the expression you just typed and add a
  call to the function ggtitle with the argument "Swirl Rules!".
  CorrectAnswer: g+geom_point()+geom_smooth(method="lm") + facet_grid(.~drv)+
  ggtitle("Swirl Rules!")
  AnswerTests: omnitest(correctExpr='g+geom_point()+geom_smooth(method="lm") +
  facet_grid(.~drv)+ ggtitle("Swirl Rules!")')
  Hint: Type g+geom_point()+geom_smooth(method="lm") + facet_grid(.~drv)+
  ggtitle("Swirl Rules!") at the command prompt.

- Class: text
  Output: Now that you've seen the basics we'll talk about customizing. Each of the
  “geom” functions (e.g., _point and _smooth) has options to modify it. Also, the
  function theme() can be used to modify aspects of the entire plot, e.g. the position
  of the legend. Two standard appearance themes are included in ggplot. These are
  theme_gray() which is the default theme (gray background with white grid lines) and
  theme_bw() which is a plainer (black and white) color scheme.

- Class: cmd_question
  Output: Let's practice modifying aesthetics now. We'll use the graphic object g that
  we already filled with mpg data and add a call to the function geom_point, but this
  time we'll give geom_point 3 arguments. Set the argument color equal to "pink", the
  argument size to 4, and the argument alpha to 1/2. Notice that all the arguments are
  set equal to constants.
  CorrectAnswer: g+geom_point(color="pink",size=4,alpha=1/2)
  AnswerTests:
  ANY_of_exprs('g+geom_point(color="pink",size=4,alpha=1/2)','g+geom_point(color="pink",s
  ize=4,alpha=.5)')
  Hint: Type g+geom_point(color="pink",size=4,alpha=1/2) at the command prompt.

- Class: text
  Output: Notice the different shades of pink? That's the result of the alpha aesthetic
  which you set to 1/2. This aesthetic tells ggplot how transparent the points should
  be. Darker circles indicate values hit by multiple data points.

- Class: cmd_question
  Output: Now we'll modify the aesthetics so that color indicates which drv type each
  point represents. Again, use g and add to it a call to the function geom_point with 3
  arguments. The first is size set equal to 4, the second is alpha equal to 1/2. The
  third is a call to the function aes with the argument color set equal to drv. Note
  that you MUST use the function aes since the color of the points is data dependent
  and not a constant as it was in the previous example.
  CorrectAnswer: g + geom_point(aes(color = drv), size = 4, alpha = 1/2)
  AnswerTests: ANY_of_exprs('g + geom_point(aes(color = drv), size = 4, alpha =
  1/2)','g + geom_point(aes(color = drv), size = 4, alpha = .5)')
  Hint: Type g + geom_point(aes(color = drv), size = 4, alpha = 1/2) at the command
  prompt.

- Class: text
  Output: Notice the helpful legend on the right decoding the relationship between
  color and drv.

- Class: cmd_question
```

```
121     Output: Now we'll practice modifying labels. Again, we'll use g and add to it calls
        to 3 functions. First, add a call to geom_point with an argument making the color
        dependent on the drv type (as we did in the previous example). Second, add a call to
        the function labs with the argument title set equal to "Swirl Rules!". Finally, add a
        call to labs with 2 arguments, one setting x equal to "Displacement" and the other
        setting y equal to "Hwy Mileage".
122     CorrectAnswer: g + geom_point(aes(color = drv)) + labs(title="Swirl Rules!") +
        labs(x="Displacement", y="Hwy Mileage")
123     AnswerTests: omnitest(correctExpr='g + geom_point(aes(color = drv)) +
        labs(title="Swirl Rules!") + labs(x="Displacement", y="Hwy Mileage")')
124     Hint: Type g + geom_point(aes(color = drv)) + labs(title="Swirl Rules!") +
        labs(x="Displacement", y="Hwy Mileage") at the command prompt.
125
126   - Class: cmd_question
127     Output: Note that you could have combined the two calls to the function labs in the
        previous example. Now we'll practice customizing the geom_smooth calls. Use g and add
        to it a call to geom_point setting the color to drv type (remember to use the call to
        the aes function), size set to 2 and alpha to 1/2. Then add a call to geom_smooth
        with 4 arguments. Set size equal to 4, linetype to 3, method to "lm", and se to FALSE.
128     CorrectAnswer: g + geom_point(aes(color = drv),size=2,alpha=1/2) +
        geom_smooth(size=4,linetype=3,method="lm",se=FALSE)
129     AnswerTests: ANY_of_exprs('g + geom_point(aes(color = drv),size=2,alpha=1/2) +
        geom_smooth(size=4,linetype=3,method="lm",se=FALSE)','g + geom_point(aes(color =
        drv),size=2,alpha=.5) + geom_smooth(size=4,linetype=3,method="lm",se=FALSE)')
130     Hint: Type g + geom_point(aes(color = drv),size=2,alpha=1/2) +
        geom_smooth(size=4,linetype=3,method="lm",se=FALSE) at the command prompt.
131
132   - Class: text
133     Output: What did these arguments do? The method specified a linear regression (note
        the negative slope indicating that the bigger the displacement the lower the gas
        mileage), the linetype specified that it should be dashed (not continuous), the size
        made the dashes big, and the se flag told ggplot to turn off the gray shadows
        indicating standard errors (confidence intervals).
134
135   - Class: cmd_question
136     Output: Finally, let's do a simple plot using the black and white theme, theme_bw.
        Specify g and add a call to the function geom_point with the argument setting the
        color to the drv type. Then add a call to the function theme_bw with the argument
        base_family set equal to "Times". See if you notice the difference.
137     CorrectAnswer: g + geom_point(aes(color = drv)) + theme_bw(base_family="Times")
138     AnswerTests: omnitest(correctExpr='g + geom_point(aes(color = drv)) +
        theme_bw(base_family="Times")')
139     Hint: Type g + geom_point(aes(color = drv)) + theme_bw(base_family="Times") at the
        command prompt.
140
141   - Class: text
142     Output: No more gray background! Also, if you have good eyesight, you'll notice that
        the font in the labels changed.
143
144   - Class: text
145     Output: One final note before we go through a more complicated, layered ggplot
        example, and this concerns the limits of the axes. We're pointing this out to
        emphasize a subtle difference between ggplot and the base plotting function plot.
146
147   - Class: cmd_question
148     Output: We've created some random x and y data, called myx and myy, components of a
        dataframe called testdat. These represent 100 random normal points, except halfway
        through, we made one of the points be an outlier. That is, we set its y-value to be
        out of range of the other points. Use the base plotting function plot to create a
        line plot of this data. Call it with 4 arguments - myx, myy, type="l", and
        ylim=c(-3,3). The type="l" tells plot you want to display the data as a line instead
        of as a scatterplot.
149     CorrectAnswer: plot(myx, myy, type = "l", ylim = c(-3,3))
150     AnswerTests: omnitest(correctExpr='plot(myx, myy, type = "l", ylim = c(-3,3))')
151     Hint: Type plot(myx, myy, type = "l", ylim = c(-3,3)) at the command prompt.
152
153   - Class: cmd_question
154     Output: Notice how plot plotted the points in the (-3,3) range for y-values. The
        outlier at (50,100) is NOT shown on the line plot. Now we'll plot the same data with
```

```
    ggplot. Recall that the name of the dataframe is testdat. Create the graphical object
    g with a call to ggplot with 2 arguments, testdat (the data) and a call to aes with 2
    arguments, x set equal to myx, and y set equal to myy.
155   CorrectAnswer: g <- ggplot(testdat, aes(x = myx, y = myy))
156   AnswerTests: expr_creates_var("g"); omnitest(correctExpr='g <- ggplot(testdat, aes(x
    = myx, y = myy))')
157   Hint: Type g <- ggplot(testdat, aes(x = myx, y = myy)) at the command prompt.
158
159 - Class: cmd_question
160   Output: Now add a call to geom_line with 0 arguments to g.
161   CorrectAnswer: g + geom_line()
162   AnswerTests: omnitest(correctExpr='g + geom_line()')
163   Hint: Type g + geom_line() at the command prompt.
164
165 - Class: text
166   Output: Notice how ggplot DID display the outlier point at (50,100). As a result the
    rest of the data is smashed down so you don't get to see what the bulk of it looks
    like. The single outlier probably isn't important enough to dominate the graph. How
    do we get ggplot to behave more like plot in a situation like this?
167
168 - Class: cmd_question
169   Output: Let's take a guess that in addition to adding geom_line() to g we also just
    have to add ylim(-3,3) to it as we did with the call to plot. Try this now to see
    what happens.
170   CorrectAnswer: g + geom_line() + ylim(-3,3)
171   AnswerTests: omnitest(correctExpr='g + geom_line() + ylim(-3,3)')
172   Hint: Type g + geom_line() + ylim(-3,3) at the command prompt.
173
174 - Class: cmd_question
175   Output: Notice that by doing this, ggplot simply ignored the outlier point at
    (50,100). There's a break in the line which isn't very noticeable. Now recall that at
    the beginning of the lesson we mentioned 7 components of a ggplot plot, one of which
    was a coordinate system. This is a situation where using a coordinate system would be
    helpful. Instead of adding ylim(-3,3) to the expression g+geom_line(), add a call to
    the function coord_cartesian with the argument ylim set equal to c(-3,3).
176   CorrectAnswer: g + geom_line() + coord_cartesian(ylim=c(-3,3))
177   AnswerTests: omnitest(correctExpr='g + geom_line() + coord_cartesian(ylim=c(-3,3))')
178   Hint: Type g + geom_line() + coord_cartesian(ylim=c(-3,3)) at the command prompt.
179
180 - Class: text
181   Output: See the difference? This looks more like the plot produced by the base plot
    function. The outlier y value at x=50 is not shown, but the plot indicates that it is
    larger than 3.
182
183 - Class: text
184   Output: We'll close with a more complicated example to show you the full power of
    ggplot and the entire ggplot2 package. We'll continue to work with the mpg dataset.
185
186 - Class: cmd_question
187   Output: Start by creating the graphical object g by assigning to it a call to ggplot
    with 2 arguments. The first is the dataset and the second is a call to the function
    aes. This call will have 3 arguments, x set equal to displ, y set equal to hwy, and
    color set equal to factor(year). This last will allow us to distinguish between the
    two manufacturing years (1999 and 2008) in our data.
188   CorrectAnswer: g <- ggplot(mpg,aes(x=displ,y=hwy,color=factor(year)))
189   AnswerTests: expr_creates_var("g"); omnitest(correctExpr='g <-
    ggplot(mpg,aes(x=displ,y=hwy,color=factor(year)))')
190   Hint: Type g <- ggplot(mpg,aes(x=displ,y=hwy,color=factor(year))) at the command
    prompt.
191
192 - Class: text
193   Output: Uh oh! Nothing happened. Does g exist? Of course, it just isn't visible yet
    since you didn't add a layer.
194
195 - Class: mult_question
196   Output:  If you typed g at the command line, what would happen?
197   AnswerChoices: a scatterplot would appear with 2 colors of points; R would return an
    error in red; I would have to try this to answer the question
198   CorrectAnswer: R would return an error in red
```

```
    AnswerTests: omnitest(correctVal='R would return an error in red')
    Hint: You've told ggplot about the data, but have you told it how to display it?

  - Class: cmd_question
    Output: We'll build the plot up step by step. First add to g a call to the function
    geom_point with 0 arguments.
    CorrectAnswer: g + geom_point()
    AnswerTests:  omnitest(correctExpr='g + geom_point()')
    Hint: Type g + geom_point() at the command prompt.

  - Class: cmd_question
    Output: A simple, yet comfortingly familiar scatterplot appears. Let's make our
    display a 2 dimensional multi-panel plot. Recall your last command (with the up
    arrow) and add to it a call the function facet_grid. Give it 2 arguments. The first
    is the formula drv~cyl, and the second is the argument margins set equal to TRUE. Try
    this now.
    CorrectAnswer: g + geom_point() + facet_grid(drv~cyl,margins=TRUE)
    AnswerTests:  omnitest(correctExpr='g + geom_point() +
    facet_grid(drv~cyl,margins=TRUE)')
    Hint: Type g + geom_point() + facet_grid(drv~cyl,margins=TRUE) at the command prompt.

  - Class: text
    Output: A 4 by 5 plot, huh? The margins argument tells ggplot to display the marginal
    totals over each row and column, so instead of seeing 3 rows (the number of drv
    factors) and 4 columns (the number of cyl factors) we see a 4 by 5 display. Note that
    the panel in position (4,5) is a tiny version of the scatterplot of the entire dataset.

  - Class: cmd_question
    Output: Now add to your last command (or retype it if you like to type) a call to
    geom_smooth with 4 arguments. These are method set to "lm", se set to FALSE, size set
    to 2, and color set to "black".
    CorrectAnswer: g + geom_point() +
    facet_grid(drv~cyl,margins=TRUE)+geom_smooth(method="lm",size=2,se=FALSE,color="black")
    AnswerTests:  omnitest(correctExpr='g + geom_point() +
    facet_grid(drv~cyl,margins=TRUE)+geom_smooth(method="lm",size=2,se=FALSE,color="black")
    ')
    Hint: Type g + geom_point() +
    facet_grid(drv~cyl,margins=TRUE)+geom_smooth(method="lm",size=2,se=FALSE,color="black")
     at the command prompt.

  - Class: cmd_question
    Output: Angry Birds? Finally, add to your last command (or retype it if you like to
    type) a call to the function labs with 3 arguments. These are x set to
    "Displacement", y set to "Highway Mileage", and title set to "Swirl Rules!".
    CorrectAnswer: g + geom_point() +
    facet_grid(drv~cyl,margins=TRUE)+geom_smooth(method="lm",size=2,se=FALSE,color="black")
    +labs(x="Displacement",y="Highway Mileage",title="Swirl Rules!")
    AnswerTests:  omnitest(correctExpr='g + geom_point() +
    facet_grid(drv~cyl,margins=TRUE)+geom_smooth(method="lm",size=2,se=FALSE,color="black")
    +labs(x="Displacement",y="Highway Mileage",title="Swirl Rules!")')
    Hint: Typeg + geom_point() +
    facet_grid(drv~cyl,margins=TRUE)+geom_smooth(method="lm",size=2,se=FALSE,color="black")
    +labs(x="Displacement",y="Highway Mileage",title="Swirl Rules!") at the command prompt.

  - Class: text
    Output: You could have done these labels with separate calls to labs but we thought
    you'd be sick of this by now. Anyway, congrats! You've concluded part 2 of ggplot2.
    We hope you got enough mileage out of the lesson. If you like ggplot2 you can do some
    extras with the extra lesson.

  - Class: mult_question
    Output: "Would you like to receive credit for completing this course on
      Coursera.org?"
    CorrectAnswer: NULL
    AnswerChoices: Yes;No
    AnswerTests: coursera_on_demand()
    Hint: ""
```