```
  Course: Exploratory_Data_Analysis
  Lesson: Lattice_Plotting_System


- Class: text
  Output: "Lattice_Plotting_System. (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/PlottingLattice.)"

- Class: text
  Output:  In another lesson, we gave you an overview of the three plotting systems in
  R. In this lesson we'll focus on the lattice plotting system.  As we did with the
  base plotting system, we'll focus on using lattice  to create graphics on the screen
  device rather than another graphics device.

- Class: text
  Output:  The lattice plotting system is completely separate and independent of the
  base plotting system. It's an add-on package so it has to be explicitly loaded with a
  call to the R function library. We've done this for you. The R Documentation tells us
  that lattice "is an implementation of Trellis graphics for R. It is a powerful and
  elegant high-level data visualization system with an emphasis on multivariate data."

- Class: text
  Output: Lattice is implemented using two packages. The first is called, not
  surprisingly, lattice, and it contains code for producing Trellis graphics. Some of
  the functions in this package are the higher level functions which you, the user,
  would call. These include xyplot, bwplot, and levelplot.


- Class: mult_question
  Output: If xyplot produces a scatterplot, what kind of plot does bwplot produce?
  AnswerChoices:  black and white; big and whittle; box and whisker; bad and wonderful
  CorrectAnswer: box and whisker
  AnswerTests: omnitest(correctVal='box and whisker')
  Hint: Which choice is the only one without opposites?

- Class: text
  Output: The second package in the lattice system is grid which contains the
  low-level  functions upon which the lattice package is built. You, the user, seldom
  call functions from the grid package directly.

- Class: text
  Output: Unlike base plotting,  the lattice system does not have a "two-phase" aspect
  with separate plotting and annotation. Instead all plotting and annotation is done at
  once with a single function call.

- Class: text
  Output: The lattice system, as the base does, provides several different plotting
  functions. These include xyplot for creating scatterplots, bwplot for
  box-and-whiskers plots or boxplots, and  histogram for  histograms. There are several
  others (stripplot, dotplot, splom and levelplot), which we won't cover here.

- Class: text
  Output: Lattice functions generally take a formula for their first argument, usually
  of the form y ~ x. This indicates that y depends on x, so in a scatterplot y would be
  plotted on the y-axis and x on the x-axis.

- Class: text
  Output: Here's an example of  typical lattice plot call,  xyplot(y ~ x | f * g,
  data). The f and g represent the optional conditioning variables. The * represents
  interaction between them. Remember when we said that lattice is good for plotting
  multivariate data? That's where these conditioning variables come into play.

- Class: text
  Output: The second argument is the data frame or list from which the variables in the
  formula should be looked up.  If no data frame or list is passed, then the parent
  frame is used. If no other arguments are passed, the default values are used.

```

```
- Class: cmd_question
  Output:  Recall the airquality data we've used before. We've loaded it again for you. To remind yourself what it looks like run the R command head with airquality as an argument to see what the data looks like.
  CorrectAnswer: head(airquality)
  AnswerTests: omnitest(correctExpr='head(airquality)')
  Hint: Type head(airquality) at the command prompt.

- Class: cmd_question
  Output:  Now try running xyplot with the formula Ozone~Wind as the first argument and the second argument data set equal to airquality.
  CorrectAnswer: xyplot(Ozone ~ Wind, data = airquality)
  AnswerTests: omnitest(correctExpr='xyplot(Ozone ~ Wind, data = airquality)')
  Hint: Type xyplot(Ozone ~ Wind, data = airquality) at the command prompt.

- Class: text
  Output: Look vaguely familiar? The dots are blue, instead of black, but lattice labeled the axes for you. You can use some of the same graphical parameters (e.g., pch and col) that you used in the base package in calls to lattice functions.

- Class: cmd_question
  Output:  Now rerun xyplot with the formula Ozone~Wind as the first argument and the second argument data set equal to airquality (use the up arrow to save typing). This time add the arguments col set equal to "red", pch set equal to 8, and main set equal to "Big Apple Data".
  CorrectAnswer: xyplot(Ozone ~ Wind, data = airquality, pch=8, col="red", main="Big Apple Data")
  AnswerTests: omnitest(correctExpr='xyplot(Ozone ~ Wind, data = airquality, pch=8, col="red", main="Big Apple Data")')
  Hint: Type xyplot(Ozone ~ Wind, data = airquality, pch=8, col="red", main="Big Apple Data") at the command prompt.

- Class: text
  Output: Red snowflakes are cool, right? Now that you've seen the basic xyplot() and some of its arguments, you might want to experiment more by yourself when you're done with the lesson to discover what other arguments and colors are available. (If you can't wait to experiment, recall that swirl has play() and nxt() functions. At a command prompt, typing play() allows you to leave swirl temporarily so  you can try different R commands at the console. Typing nxt() when you're done playing brings you back to swirl and you can resume your lesson.)

- Class: text
  Output: Now you'll see how easy it is to generate a multipanel plot using a single lattice command.

- Class: cmd_question
  Output:  Run xyplot with the formula Ozone~Wind | as.factor(Month) as the first argument and the second argument data set equal to airquality (use the up arrow to save typing). So far, not much is different, right? Add a third argument, layout, set equal to c(5,1).
  CorrectAnswer: xyplot(Ozone ~ Wind | as.factor(Month), data = airquality, layout=c(5,1))
  AnswerTests: omnitest(correctExpr='xyplot(Ozone ~ Wind | as.factor(Month), data = airquality, layout=c(5,1))')
  Hint: Type xyplot(Ozone ~ Wind | as.factor(Month), data = airquality, layout=c(5,1)) at the command prompt.

- Class: mult_question
  Output: Note that the default color and plotting character are back. What did the as.factor(Month) do?
  AnswerChoices:  Displayed and labeled each subplot with the month's integer; Displayed the data by individual months; Huh?; Randomly divided the data into 5 panels
  CorrectAnswer:  Displayed and labeled each subplot with the month's integer
  AnswerTests: omnitest(correctVal='Displayed and labeled each subplot with the month\'s integer')
  Hint: Obviously the data is broken up and displayed by month. The as.factor made sure each panel was labeled correctly.

- Class: cmd_question
```

```
  Output:  Since Month is a named column of the airquality dataframe we had to tell R
  to treat it as a factor. To see how this affects the plot, rerun the xyplot command
  you just ran, but use Ozone ~ Wind | Month instead of Ozone ~ Wind | as.factor(Month)
  as the first argument.
  CorrectAnswer: xyplot(Ozone ~ Wind | Month, data = airquality, layout=c(5,1))
  AnswerTests: omnitest(correctExpr='xyplot(Ozone ~ Wind | Month, data = airquality,
  layout=c(5,1))')
  Hint: Type xyplot(Ozone ~ Wind | Month, data = airquality, layout=c(5,1)) at the
  command prompt.

- Class: text
  Output: Not as informative, right? The word Month in each panel really doesn't tell
  you much if it doesn't identify which month it's plotting. Notice that the actual
  data is the same between the two plots, though.

- Class: text
  Output: Lattice functions behave differently from base graphics functions in one
  critical way. Recall that base graphics functions plot data directly to the graphics
  device (e.g., screen, or file such as a PDF file). In contrast, lattice graphics
  functions return an object of class trellis.

- Class: figure
  Output: The print methods for lattice functions actually do the work of plotting the
  data on the graphics device. They return "plot objects" that can be stored (but it's
  usually better to just save the code  and data). On the command line, trellis objects
  are auto-printed so that it appears the function is plotting the data.
  Figure: clearPlot.R
  FigureType: new

- Class: cmd_question
  Output:  To see this, create a variable p which is assigned the output of this simple
  call to xyplot, xyplot(Ozone~Wind,data=airquality).
  CorrectAnswer: p <- xyplot(Ozone~Wind,data=airquality)
  AnswerTests: expr_creates_var("p"); omnitest(correctExpr='p <-
  xyplot(Ozone~Wind,data=airquality)')
  Hint: Type p <- xyplot(Ozone~Wind,data=airquality) at the command prompt.

- Class: text
  Output: Nothing plotted, right? But the object p is around.

- Class: cmd_question
  Output:  Type p or print(p) now to see it.
  CorrectAnswer: p
  AnswerTests: ANY_of_exprs('p','print(p)')
  Hint: Type p or print(p) at the command prompt.

- Class: cmd_question
  Output: Like magic, it appears. Now run the R command names with p as its argument.
  CorrectAnswer: names(p)
  AnswerTests: omnitest(correctExpr='names(p)')
  Hint: Type names(p) at the command prompt.

- Class: cmd_question
  Output: We see that the trellis object p has 45 named properties, the first of which
  is "formula" which isn't too surprising. A lot of these properties are probably NULL
  in value. We've done some behind-the-scenes work for you and created two vectors. The
  first, mynames, is a character vector of the names in p. The second is a boolean
  vector, myfull,  which has TRUE values for nonnull entries of p. Run mynames[myfull]
  to see which entries of p are not NULL.
  CorrectAnswer: mynames[myfull]
  AnswerTests: omnitest(correctExpr='mynames[myfull]')
  Hint: Type mynames[myfull] at the command prompt.

- Class: cmd_question
  Output: Wow! 29 nonNull values for one little plot. Note that a lot of them are like
  the ones we saw in the base plotting system. Let's look at the values of some of
  them. Type p[["formula"]] now.
  CorrectAnswer: p[["formula"]]
  AnswerTests: omnitest(correctExpr='p[["formula"]]')
```

```
    Hint: Type p[["formula"]] at the command prompt.

  - Class: cmd_question
    Output: Not surprising, is it? It's a familiar formula. Now look at p's x.limits.
    Remember the double square brackets and quotes.
    CorrectAnswer: p[["x.limits"]]
    AnswerTests: omnitest(correctExpr='p[["x.limits"]]')
    Hint: Type p[["x.limits"]] at the command prompt.

  - Class: text
    Output: They match the plot, right? The x values are indeed between .37 and 22.03.

  - Class: text
    Output: Again, not surprising. Before we wrap up, let's talk about lattice's panel
    functions which control what happens inside each panel of the plot. The ease of
    making multi-panel plots makes lattice very appealing. The lattice package comes with
    default panel functions, but you can customize what happens in each panel.

  - Class: cmd_question
    Output: Panel functions receive the x and y coordinates of the data points in their
    panel (along with any optional arguments). To see this, we've created some data for
    you - two 100-long vectors, x and y. For its first 50 values y is a function of x,
    for the last 50 values, y is random. We've also defined a 100-long factor vector f
    which distinguishes between the first and last 50 elements of the two vectors. Run
    the R command table with f as it argument.
    CorrectAnswer: table(f)
    AnswerTests: omnitest(correctExpr='table(f)')
    Hint: Type table(f) at the command prompt.

  - Class: cmd_question
    Output: The first 50 entries of f are "Group 1" and the last 50 are "Group 2". Run
    xyplot with two arguments. The first is the formula y~x|f, and the second is layout
    set equal to c(2,1). Note that we're not providing an explicit data argument, so
    xyplot will look in the environment and see the x and y that we've generated for you.
    CorrectAnswer: xyplot(y ~ x | f, layout = c(2, 1))
    AnswerTests: omnitest(correctExpr='xyplot(y ~ x | f, layout = c(2, 1))')
    Hint: Type xyplot(y ~ x | f, layout = c(2, 1)) at the command prompt.

  - Class: cmd_question
    Output: To understand this a little better look at the variable v1 we've created for
    you.
    CorrectAnswer: v1
    AnswerTests: omnitest(correctExpr='v1')
    Hint: Type v1 at the command prompt.

  - Class: cmd_question
    Output: The first two numbers are the range of the x values of Group 1 and the last
    two numbers are the range of y values of Group 1. See how they match the values of
    the left panel (Group 1) in the plot. Now look at v2 which holds the comparable
    numbers for Group 2.
    CorrectAnswer: v2
    AnswerTests: omnitest(correctExpr='v2')
    Hint: Type v2 at the command prompt.

  - Class: cmd_question
    Output: Again, the values match the plot. That's reassuring. We've copied some code
    from the slides for you. To see it, type myedit("plot1.R"). This will open your
    editor and display the R code in it.
    CorrectAnswer: myedit("plot1.R")
    AnswerTests: omnitest(correctExpr='myedit("plot1.R")')
    Hint: Type myedit("plot1.R") at the command prompt.

  - Class: mult_question
    Output: How many calls to basic lattice plotting functions are there in plot1.R?
    AnswerChoices: 1;2;3
    CorrectAnswer: 1
    AnswerTests: omnitest(correctVal='1')
    Hint: How many calls to xyplot are there?
```

```
- Class: mult_question
  Output: Note the panel function. How many formal arguments does it have?
  AnswerChoices: 1;2;3
  CorrectAnswer: 3
  AnswerTests: omnitest(correctVal='3')
  Hint: You have to count the ... as an argument?

- Class: figure
  Output: The panel function has 3 arguments, x, y and ... . This last stands for all other arguments (such as graphical parameters) you might want to include. There are 2 lines in the panel function. Each invokes a panel method, the first to plot the data in each panel (panel.xyplot), the second to draw a horizontal line in each panel (panel.abline). Note the similarity of this last call to that of the base plotting function of the same name.
  Figure: clearPlot.R

- Class: cmd_question
  Output: We've defined a function for you, pathtofile, which takes a filename as its argument. This makes sure R can find the file on your computer. Now run the R command source with two arguments. The first is the call to pathtofile with the string "plot1.R" as its argument and the second is the argument local set equal to TRUE. This command will run the code contained in plot1.R within the swirl environment so you can see what it does.
  CorrectAnswer: source(pathtofile("plot1.R"),local=TRUE)
  AnswerTests: omnitest(correctExpr='source(pathtofile("plot1.R"),local=TRUE)')
  Hint: Type source(pathtofile("plot1.R"),local=TRUE) at the command prompt.

- Class: mult_question
  Output: See how the lines appear. The plot shows two panels because...?
  AnswerChoices: f contains 2 factors; there are 2 calls to panel methods; lattice can handle at most 2 panels; there are 2 variables
  CorrectAnswer: f contains 2 factors
  AnswerTests: omnitest(correctVal='f contains 2 factors')
  Hint: The number of panels depends on the conditioning variable, and in this case it is f.

- Class: cmd_question
  Output: We've copied another piece of similar code, i.e., a call to xyplot with a custom panel function, from the slides. To see it, type myedit("plot2.R"). This will open your editor and display the R code in it.
  CorrectAnswer: myedit("plot2.R")
  AnswerTests: omnitest(correctExpr='myedit("plot2.R")')
  Hint: Type myedit("plot2.R") at the command prompt.

- Class: text
  Output: You can see how plot2.R differs from plot1.R, right?

- Class: cmd_question
  Output: Again, run the R command source with  the two arguments pathtofile("plot2.R") and local=TRUE. This will run the code in plot2.R.
  CorrectAnswer: source(pathtofile("plot2.R"),local=TRUE)
  AnswerTests: omnitest(correctExpr='source(pathtofile("plot2.R"),local=TRUE)')
  Hint: Type source(pathtofile("plot2.R"),local=TRUE) at the command prompt.

- Class: mult_question
  Output: The regression lines are red because ...?
  AnswerChoices: R always plots regression lines in red; R is the first letter of the word red; the custom panel function specified a col argument
  CorrectAnswer: the custom panel function specified a col argument
  AnswerTests: omnitest(correctVal='the custom panel function specified a col argument')
  Hint: Look carefully at the arguments in the call to panel.lmline.

- Class: figure
  Output: Before we close we'll look at how easily lattice can handle a plot with a great many panels. (The sky's the limit.) We've loaded some diamond data for you. It comes with the ggplot2 package. We'll use it just to show off lattice's panel plotting capability.
  Figure: clearPlot.R
  FigureType: new
```

```
232
233 - Class: cmd_question
234   Output: The data is in the data frame diamonds. Use the R command str to see what it
      looks like.
235   CorrectAnswer: str(diamonds)
236   AnswerTests: omnitest(correctExpr='str(diamonds)')
237   Hint: Type str(diamonds) at the command prompt.
238
239 - Class: cmd_question
240   Output: So the data frame contains 10 pieces of information for each of 53940
      diamonds. Run the R command table with diamonds$color as an argument.
241   CorrectAnswer: table(diamonds$color)
242   AnswerTests: omnitest(correctExpr='table(diamonds$color)')
243   Hint: Type table(diamonds$color) at the command prompt.
244
245 - Class: cmd_question
246   Output: We see 7 colors each represented by a letter. Now  run the R command table
      with two arguments, diamonds$color and diamonds$cut.
247   CorrectAnswer: table(diamonds$color,diamonds$cut)
248   AnswerTests: omnitest(correctExpr='table(diamonds$color,diamonds$cut)')
249   Hint: Type table(diamonds$color,diamonds$cut) at the command prompt.
250
251
252 - Class: mult_question
253   Output:  We see a 7 by 5 array with counts indicating how many diamonds in the data
      frame have a particular color and cut. From the table, which is the most frequent
      combination?
254   AnswerChoices: Premium cut of color G; Ideal color of cut G; Ideal cut of color G;
      Ideal cut of color F.
255   CorrectAnswer: Ideal cut of color G
256   AnswerTests: omnitest(correctVal='Ideal cut of color G')
257   Hint: Colors are depicted by letters, so one choice is eliminated. Which letter
      appears in most of the choices?
258
259 - Class: cmd_question
260   Output:  To save you some trouble we've defined three character strings for you,
      labels for the x- and y-axes and a main title. They're in the file myLabels.R, so run
      myedit on this file to see  them. Remember to put the file name in quotes when you
      call myedit.
261   CorrectAnswer: myedit("myLabels.R")
262   AnswerTests: omnitest(correctExpr='myedit("myLabels.R")')
263   Hint: Type myedit("myLabels.R") at the command prompt.
264
265 - Class: cmd_question
266   Output:  Now run source with pathtofile("myLabels.R") and local set equal to TRUE.
267   CorrectAnswer: source(pathtofile("myLabels.R"),local=TRUE)
268   AnswerTests: omnitest(correctExpr='source(pathtofile("myLabels.R"),local=TRUE)')
269   Hint: Type source(pathtofile("myLabels.R"),local=TRUE) at the command prompt.
270
271 - Class: cmd_question
272   Output: Now call xyplot with the formula price~carat | color*cut  and data set equal
      to diamonds. In addition, set the argument strip equal to  FALSE,  pch set equal to
      20, xlab to myxlab, ylab to myylab, and main to mymain. The plot may take longer than
      previous plots because it is bigger.
273   CorrectAnswer:
      xyplot(price~carat|color*cut,data=diamonds,strip=FALSE,pch=20,xlab=myxlab,ylab=myylab,m
      ain=mymain)
274   AnswerTests:
      omnitest(correctExpr='xyplot(price~carat|color*cut,data=diamonds,strip=FALSE,pch=20,xla
      b=myxlab,ylab=myylab,main=mymain)')
275   Hint: Type
      xyplot(price~carat|color*cut,data=diamonds,strip=FALSE,pch=20,xlab=myxlab,ylab=myylab,m
      ain=mymain) at the command prompt.
276
277 - Class: text
278   Output: Pretty cool, right? 35 panels, one for each combination of color and cut. The
      dots (pch=20) show how prices for the diamonds in each category (panel) vary
      depending on carat.
279
```

```
- Class: mult_question
  Output: Are colors defining the rows or columns of the plot?
  AnswerChoices: rows; columns
  CorrectAnswer: columns
  AnswerTests: omnitest(correctVal='columns')
  Hint: Recall that there were 7 colors and 5 cuts in the data.

- Class: cmd_question
  Output: Were you curious about that argument strip? I know I was. Now  rerun the
  xyplot command you just ran (use the up arrow key to retrieve it), this time without
  the strip argument.
  CorrectAnswer:
  xyplot(price~carat|color*cut,data=diamonds,pch=20,xlab=myxlab,ylab=myylab,main=mymain)
  AnswerTests:
  omnitest(correctExpr='xyplot(price~carat|color*cut,data=diamonds,pch=20,xlab=myxlab,yla
  b=myylab,main=mymain)')
  Hint: Type
  xyplot(price~carat|color*cut,data=diamonds,pch=20,xlab=myxlab,ylab=myylab,main=mymain)
  at the command prompt.

- Class: mult_question
  Output: The plot shows that the strip argument ....
  AnswerChoices: labels each panel; removes information from the plot; makes the plot
  less intelligible; has a default value of FALSE
  CorrectAnswer: columns
  AnswerTests: omnitest(correctVal='labels each panel')
  Hint: Do the words in the colorful stripes convey useful information?


- Class: text
  Output:  Review time!!!

- Class: mult_question
  Output: True or False? Lattice plots are constructed by a series of calls to core
  functions.
  AnswerChoices: True; False
  CorrectAnswer: False
  AnswerTests: omnitest(correctVal='False')
  Hint: Recall the long call you just made to plot 35 panels in one picture. We would
  have broken it up if we could have, but we didn't, so we can't.

- Class: mult_question
  Output: True or False? Lattice plots are constructed with a single function call to a
  core lattice function (e.g. xyplot)
  AnswerChoices: True; False
  CorrectAnswer: True
  AnswerTests: omnitest(correctVal='True')
  Hint: This is the opposite of the last question which was false.

- Class: mult_question
  Output: True or False? Aspects like margins and spacing are automatically handled and
  defaults are usually sufficient.
  AnswerChoices: True; False
  CorrectAnswer: True
  AnswerTests: omnitest(correctVal='True')
  Hint: In any of our examples, did we mention margins?

- Class: mult_question
  Output: True or False? The lattice system is ideal for creating conditioning plots
  where you examine the same kind of plot under many different conditions.
  AnswerChoices: True; False
  CorrectAnswer: True
  AnswerTests: omnitest(correctVal='True')
  Hint: Think of the ease with which lattice handles multi-panel plots.

- Class: mult_question
  Output: True or False? The lattice system, like the base plotting system, returns a
  trellis plot object.
  AnswerChoices: True; False
```

```
    CorrectAnswer: False
    AnswerTests: omnitest(correctVal='False')
    Hint: This is the key difference between the two systems. Lattice DOES return a plot
    object but base doesn't.

  - Class: mult_question
    Output: True or False?  Panel functions can NEVER be customized to modify what is
    plotted in each of the plot panels.
    AnswerChoices: True; False
    CorrectAnswer: False
    AnswerTests: omnitest(correctVal='False')
    Hint: Recall our advice to NEVER trust questions with NEVER and ALWAYS in them.

  - Class: mult_question
    Output: True or False?  Lattice plots can display at most 20 panels in a single plot.
    AnswerChoices: True; False
    CorrectAnswer: False
    AnswerTests: omnitest(correctVal='False')
    Hint: Recall the sparkly diamonds.

  - Class: text
    Output: Congrats! We hope this lesson didn't leave you climbing the trellis.

  - 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: ""

```