```
Course: Exploratory_Data_Analysis
Lesson: CaseStudy


- Class: text
  Output: "CaseStudy. (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/CaseStudy.)"

- Class: text
  Output:  In this lesson we'll apply some of the techniques we learned in this course to study air pollution data, specifically particulate matter (we'll call it pm25 sometimes), collected by the U.S. Environmental Protection Agency. This website https://www.health.ny.gov/environmental/indoors/air/pmq_a.htm from New York State offers some basic information on this topic if you're interested.

- Class: text
  Output: Particulate matter (less than 2.5 microns in diameter) is a fancy name for dust, and breathing in dust might pose health hazards to the population. We'll study data from two years, 1999 (when monitoring of particulate matter started) and 2012. Our goal is to see if there's been a noticeable decline in this type of air pollution between these two years.

- Class: cmd_question
  Output:  We've read in 2 large zipped files for you using the R command read.table (which is smart enough to unzip the files).  We stored the 1999 data in the array pm0 for you. Run the R command dim now to see its dimensions.
  CorrectAnswer: dim(pm0)
  AnswerTests: omnitest(correctExpr='dim(pm0)')
  Hint: Type dim(pm0) at the command prompt.

- Class: cmd_question
  Output:  We see that pm0 has over 117000 lines, each containing 5 columns. In the original file, at the EPA website, each row had 28 columns, but since we'll be using only a few of these, we've created and read in a somewhat smaller file. Run head on pm0 now to see what the first few lines look like.
  CorrectAnswer: head(pm0)
  AnswerTests: omnitest(correctExpr='head(pm0)')
  Hint: Type head(pm0) at the command prompt.

- Class: text
  Output: We see there's some missing data, but we won't worry about that now. We also see that the column names, V1, V2, etc., are not informative. However, we know that the first line of the original file (a comment) explained what information the columns contained.

- Class: cmd_question
  Output:  We created the variable cnames containing the 28 column names of the original file. Take a look at the column names now.
  CorrectAnswer: cnames
  AnswerTests: ANY_of_exprs('cnames','print(cnames)')
  Hint: Type cnames or print(cnames) at the command prompt.

- Class: cmd_question
  Output:  We see that the 28 column names look all jumbled together even though they're separated by "|" characters, so let's fix this. Reassign to cnames the output of a call to strsplit (string split) with 3 arguments. The first is cnames, the pipe symbol '|' is the second (use the quotation marks), and the third is the argument fixed set to TRUE. Try this now.
  CorrectAnswer:  cnames <- strsplit(cnames, "|", fixed = TRUE)
  AnswerTests:  any_of_exprs('cnames <- strsplit(cnames, "|", fixed = T)', 'cnames <- strsplit(cnames, "|", fixed = TRUE)')
  Hint: Type cnames <- strsplit(cnames, "|", fixed = TRUE) at the command prompt.

- Class: cmd_question
  Output: The variable cnames now holds a list of the column headings. Take another look at the column names.
  CorrectAnswer: cnames
```

```
  AnswerTests: ANY_of_exprs('cnames','print(cnames)')
  Hint: Type cnames or print(cnames) at the command prompt.

- Class: cmd_question
  Output:  Nice, but we don't need all these. Assign to names(pm0) the output of a call to the function make.names with cnames[[1]][wcol] as the argument. The variable wcol holds the indices of the 5 columns we selected (from the 28) to use in this lesson, so those are the column names we'll need. As the name suggests, the function "makes syntactically valid names".
  CorrectAnswer:  names(pm0) <- make.names(cnames[[1]][wcol])
  AnswerTests: omnitest(correctExpr='names(pm0) <- make.names(cnames[[1]][wcol])')
  Hint: Type names(pm0) <- make.names(cnames[[1]][wcol]) at the command prompt.

- Class: cmd_question
  Output:  Now re-run head on pm0 now to see if the column names have been put in place.
  CorrectAnswer: head(pm0)
  AnswerTests: omnitest(correctExpr='head(pm0)')
  Hint: Type head(pm0) at the command prompt.

- Class: cmd_question
  Output:  Now it's clearer what information each column of pm0 holds. The measurements of particulate matter (pm25) are in the column named Sample.Value. Assign this component of pm0 to the variable x0. Use the m$n notation.
  CorrectAnswer: x0 <- pm0$Sample.Value
  AnswerTests: expr_creates_var("x0"); omnitest(correctExpr='x0 <- pm0$Sample.Value')
  Hint: Type x0 <- pm0$Sample.Value at the command prompt.

- Class: cmd_question
  Output:  Call the R command str with x0 as its argument to see x0's structure.
  CorrectAnswer: str(x0)
  AnswerTests:  omnitest(correctExpr='str(x0)')
  Hint: Type str(x0) at the command prompt.

- Class: cmd_question
  Output:  We see that x0 is a numeric vector (of length 117000+) with at least the first 3 values missing.  Exactly what percentage of values are missing in this vector? Use the R function mean with is.na(x0) as an argument to see what percentage of values are missing (NA) in x0.
  CorrectAnswer: mean(is.na(x0))
  AnswerTests:  omnitest(correctExpr='mean(is.na(x0))')
  Hint: Type mean(is.na(x0)) at the command prompt.

- Class: text
  Output: So a little over 11% of the 117000+ are missing. We'll keep that in mind. Now let's start processing the 2012 data which we stored for you in the array pm1.

- Class: cmd_question
  Output:  We'll repeat what we did for pm0, except a little more efficiently. First assign the output of make.names(cnames[[1]][wcol]) to names(pm1).
  CorrectAnswer: names(pm1) <- make.names(cnames[[1]][wcol])
  AnswerTests:  omnitest(correctExpr='names(pm1) <- make.names(cnames[[1]][wcol])')
  Hint: Type names(pm1) <- make.names(cnames[[1]][wcol]) at the command prompt.

- Class: cmd_question
  Output:  Find the dimensions of pm1 with the command dim.
  CorrectAnswer: dim(pm1)
  AnswerTests:  omnitest(correctExpr='dim(pm1)')
  Hint: Type dim(pm1) at the command prompt.

- Class: text
  Output: Wow! Over 1.3 million entries. Particulate matter was first collected in 1999 so  perhaps there weren't as many sensors  collecting data then as in 2012 when the program was more mature. If you ran head on pm1 you'd see that it looks just like pm0. We'll move on though.

- Class: cmd_question
  Output:  Create the variable x1 by assigning to it the Sample.Value component of pm1.
  CorrectAnswer:  x1 <- pm1$Sample.Value
  AnswerTests:  expr_creates_var("x1"); omnitest(correctExpr='x1 <- pm1$Sample.Value')
```

```
    Hint: Type x1 <- pm1$Sample.Value at the command prompt.

  - Class: cmd_question
    Output:  Now let's see what percentage of values are missing in x1. As before, use the R function mean with is.na(x1) as an argument to find out.
    CorrectAnswer: mean(is.na(x1))
    AnswerTests:  omnitest(correctExpr='mean(is.na(x1))')
    Hint: Type mean(is.na(x1)) at the command prompt.

  - Class: text
    Output: So only 5.6% of the particulate matter measurements are missing. That's about half the percentage as in 1999.

  - Class: cmd_question
    Output:  Now let's look at summaries (using the summary command) for both datasets. First, x0.
    CorrectAnswer: summary(x0)
    AnswerTests:  omnitest(correctExpr='summary(x0)')
    Hint: Type summary(x0) at the command prompt.

  - Class: cmd_question
    Output:  The numbers in the vectors x0 and x1 represent measurements taken in micrograms per cubic meter. Now look at the summary of x1.
    CorrectAnswer: summary(x1)
    AnswerTests:  omnitest(correctExpr='summary(x1)')
    Hint: Type summary(x1) at the command prompt.

  - Class: text
    Output: We see that both the median and the mean of measured particulate matter have declined from 1999 to 2012. In fact, all of the measurements, except for the maximum and missing values (Max and  NA's), have decreased. Even the Min has gone down from 0 to -10.00! We'll address what a negative measurment might mean a little later. Note that the Max has increased from 157 in 1999 to 909 in 2012. This is quite high and might reflect an error in the table or malfunctions in some monitors.

  - Class: cmd_question
    Output:  Call the boxplot function with 2 arguments, x0 and x1.
    CorrectAnswer: boxplot(x0, x1)
    AnswerTests:  omnitest(correctExpr='boxplot(x0, x1)')
    Hint: Type boxplot(x0, x1) at the command prompt.

  - Class: cmd_question
    Output:  Huh? Did somebody step on the boxes? It's hard to see what's going on here. There are so many values outside the boxes and the range of x1 is so big that the boxes are flattened. It might be more informative to call boxplot on the logs (base 10) of x0 and x1. Do this now using log10(x0) and log10(x1) as the 2 arguments.
    CorrectAnswer: boxplot(log10(x0), log10(x1))
    AnswerTests:  omnitest(correctExpr='boxplot(log10(x0), log10(x1))')
    Hint: Type boxplot(log10(x0), log10(x1)) at the command prompt.

  - Class: text
    Output: A bonus! Not only do we get a better looking boxplot we also get some warnings from R in Red. These let us know that some values in x0 and x1 were "unloggable", no doubt the 0 (Min) we saw in the summary of x0 and the negative values we saw in the Min of the summary of x1.

  - Class: mult_question
    Output: From the boxplot (x0 on the left and x1 on the right), what can you say about the data?
    AnswerChoices:  The median of x1 is less than the median of x0; The mean of x1 is less than the mean of x0; The range of x0 is greater than the range of x1; The boxes are too small to interpret
    CorrectAnswer:  The median of x1 is less than the median of x0
    AnswerTests: omnitest(correctVal='The median of x1 is less than the median of x0')
    Hint: Recall that the horizontal lines inside the boxes represent the medians of the two datasets.

  - Class: text
    Output: Let's return to the question of the negative values in x1. Let's count how
```

```
  many negative values there are. We'll do this in a few steps.

- Class: cmd_question
  Output:  First, form the vector negative by assigning to it the boolean x1<0.
  CorrectAnswer: negative <- x1 < 0
  AnswerTests:  expr_creates_var("negative"); omnitest(correctExpr='negative <- x1 < 0')
  Hint: Type negative <- x1 < 0 at the command prompt.

- Class: cmd_question
  Output:  Now run the R command sum with 2 arguments. The first is negative, and the second is na.rm set equal to TRUE. This tells sum to ignore the missing values in negative.
  CorrectAnswer: sum(negative, na.rm = TRUE)
  AnswerTests:   omnitest(correctExpr='sum(negative, na.rm = TRUE)')
  Hint: Type sum(negative, na.rm = TRUE)  at the command prompt.

- Class: cmd_question
  Output:  So there are over 26000 negative values. Sounds like a lot. Is it? Run the R command mean with same 2 arguments you just used with the call to sum. This will tell us a percentage.
  CorrectAnswer: mean(negative, na.rm = TRUE)
  AnswerTests:  omnitest(correctExpr='mean(negative, na.rm = TRUE)')
  Hint: Type mean(negative, na.rm = TRUE)  at the command prompt.

- Class: text
  Output: We see that just 2% of the x1 values are negative. Perhaps that's a small enough percentage that we can ignore them. Before we ignore them, though, let's see if they occur during certain times of the year.

- Class: cmd_question
  Output:  First create the array dates by assigning to it the Date component of pm1. Remember to use the x$y notation.
  CorrectAnswer: dates <- pm1$Date
  AnswerTests:  expr_creates_var("dates"); omnitest(correctExpr='dates <- pm1$Date')
  Hint: Type dates <- pm1$Date  at the command prompt.

- Class: cmd_question
  Output:  To see what dates looks like run the R command str on it.
  CorrectAnswer: str(dates)
  AnswerTests:   omnitest(correctExpr='str(dates)')
  Hint: Type str(dates)  at the command prompt.

- Class: cmd_question
  Output:  We see dates is a very long vector of integers. However, the format of the entries is hard to read. There's no separation between the year, month, and day. Reassign to dates the output of a call to as.Date with the 2 arguments as.character(dates) as the first argument and the string "%Y%m%d" as the second.
  CorrectAnswer: dates <- as.Date(as.character(dates), "%Y%m%d")
  AnswerTests:   expr_creates_var("dates"); omnitest(correctExpr='dates <- as.Date(as.character(dates), "%Y%m%d")')
  Hint: Type dates <- as.Date(as.character(dates), "%Y%m%d") at the command prompt.

- Class: cmd_question
  Output:  Now when you run head on dates you'll see the dates in a nicer format. Try this now.
  CorrectAnswer: head(dates)
  AnswerTests:    omnitest(correctExpr='head(dates)')
  Hint: Type head(dates) at the command prompt.

- Class: cmd_question
  Output: Let's plot a histogram of the months when the particulate matter measurements are negative. Run hist with 2 arguments. The first is dates[negative] and the second is the string "month".
  CorrectAnswer: hist(dates[negative],"month")
  AnswerTests:  omnitest(correctExpr='hist(dates[negative],"month")')
  Hint: Type hist(dates[negative],"month") at the command prompt.

- Class: text
  Output: We see the bulk of the negative measurements were taken in the winter months,
```

```
  with a spike in May. Not many of these negative measurements occurred in summer months. We can take a guess that because particulate measures tend to be low in winter and high in summer, coupled with the fact that higher densities are easier to measure, that measurement errors occurred when the values were low. For now we'll attribute these negative measurements to errors. Also, since they account for only 2% of the 2012 data, we'll ignore them.


- Class: figure
  Output: Now we'll change focus a bit and instead of looking at all the monitors throughout the country and the data they recorded, we'll try to find one monitor that was taking measurements in both 1999 and 2012. This will allow us to control for different geographical and environmental variables that might have affected air quality in different areas. We'll narrow our search and look just at monitors in New York State.
  Figure: runSites.R
  FigureType: new

- Class: cmd_question
  Output: We subsetted off the New York State monitor identification data for 1999 and 2012  into 2 vectors, site0 and site1. Look at the structure of site0 now with the R command str.
  CorrectAnswer: str(site0)
  AnswerTests:    obliterate("dates"); omnitest(correctExpr='str(site0)')
  Hint: Type str(site0) at the command prompt.

- Class: text
  Output: We see that site0 (the IDs of monitors in New York State in 1999) is a vector of 33 strings, each of which has the form "x.y". We've created these from the county codes (the x portion of the string) and the monitor IDs (the y portion). If you ran str on site1 you'd see 18 similar values.

- Class: cmd_question
  Output: Use the intersect command with site0 and site1 as arguments and put the result in the variable both.
  CorrectAnswer: both <- intersect(site0, site1)
  AnswerTests:    obliterate("x0"); expr_creates_var("both"); omnitest(correctExpr='both <- intersect(site0, site1)')
  Hint: Type both <- intersect(site0, site1) at the command prompt.

- Class: cmd_question
  Output: Take a look at both now.
  CorrectAnswer: both
  AnswerTests:  obliterate("x1"); ANY_of_exprs('both','print(both)')
  Hint: Type both or print(both) at the command prompt.

- Class: figure
  Output:  We see that 10 monitors in New York State were active in both 1999 and 2012.
  Figure: runModpms.R
  FigureType: new

- Class: cmd_question
  Output: To save you some time and typing, we modified the data frames pm0 and pm1 slightly  by adding to each of them a new component, county.site. This is just a concatenation of two original components County.Code and Site.ID. We did this to facilitate the next step which is to find out how many measurements were taken by the 10 New York monitors working in both of the years of interest. Run head on pm0 to see the first few entries now.
  CorrectAnswer: head(pm0)
  AnswerTests:  obliterate("negative"); omnitest(correctExpr='head(pm0)')
  Hint: Type head(pm0) at the command prompt.

- Class: text
  Output: Now pm0 and pm1 have 6 columns instead of 5, and the last column is a concatenation of two other columns, County and Site.

- Class: text
  Output: Now let's see how many measurements each of the 10 New York monitors that were active in both 1999 and 2012 took in those years. We'll create 2 subsets (one
```

```
  for each year), one of pm0 and the other of pm1.

- Class: text
  Output: The subsets will filter for 2 characteristics. The first is State.Code equal to 36 (the code for New York), and the second is that the county.site (the component we  added) is in the vector both.

- Class: cmd_question
  Output:  First create the variable cnt0 by assigning to it the output of the R command subset, called with 2 arguments. The first is pm0, and the second is a boolean with the 2 conditions we just mentioned. Recall that the testing for equality in a boolean requires ==, intersection of 2 boolean conditions is denoted by  & and membership by %in%.
  CorrectAnswer: cnt0 <- subset(pm0, State.Code == 36 & county.site %in% both)
  AnswerTests:  expr_creates_var("cnt0"); omnitest(correctExpr='cnt0 <- subset(pm0, State.Code == 36 & county.site %in% both)')
  Hint: Type cnt0 <- subset(pm0, State.Code == 36 & county.site %in% both) at the command prompt.

- Class: cmd_question
  Output:  Recall the last command with the up arrow, and create cnt1 (instead of cnt0). Remember to change pm0 to pm1. Everything else can stay the same.
  CorrectAnswer: cnt1 <- subset(pm1, State.Code == 36 & county.site %in% both)
  AnswerTests:  expr_creates_var("cnt1"); omnitest(correctExpr='cnt1 <- subset(pm1, State.Code == 36 & county.site %in% both)')
  Hint: Type  cnt1 <- subset(pm1, State.Code == 36 & county.site %in% both) at the command prompt.

- Class: cmd_question
  Output:  Now run the command sapply(split(cnt0, cnt0$county.site), nrow). This will split  cnt0 into several data frames according to county.site (that is, monitor IDs) and tell us how many measurements each monitor recorded.
  CorrectAnswer: sapply(split(cnt0, cnt0$county.site), nrow)
  AnswerTests:   omnitest(correctExpr='sapply(split(cnt0, cnt0$county.site), nrow)')
  Hint: Type sapply(split(cnt0, cnt0$county.site), nrow) at the command prompt.

- Class: cmd_question
  Output:  Do the same for cnt1. (Recall your last command and change 2 occurrences of cnt0 to cnt1.)
  CorrectAnswer: sapply(split(cnt1, cnt1$county.site), nrow)
  AnswerTests:   omnitest(correctExpr='sapply(split(cnt1, cnt1$county.site), nrow)')
  Hint: Type sapply(split(cnt1, cnt1$county.site), nrow) at the command prompt.

- Class: mult_question
  Output: From the output of the 2 calls to sapply, which monitor is the only one whose number of measurements increased from 1999 to 2012?
  AnswerChoices:  29.5; 85.55; 63.2008; 101.3
  CorrectAnswer:  85.55
  AnswerTests: omnitest(correctVal='85.55')
  Hint: Compare the 2 rows of numbers and look for the one with the bigger bottom number.

- Class: cmd_question
  Output:  We want to examine a monitor with a reasonable number of measurements so let's look at the monitor with ID 63.2008. Create a variable pm0sub which is the subset of cnt0 (this contains just New York data) which has County.Code equal to 63 and Site.ID 2008.
  CorrectAnswer: pm0sub <- subset(cnt0, County.Code==63 & Site.ID==2008)
  AnswerTests:   expr_creates_var("pm0sub"); omnitest(correctExpr='pm0sub <- subset(cnt0, County.Code==63 & Site.ID==2008)')
  Hint: Type pm0sub <- subset(cnt0, County.Code==63 & Site.ID==2008) at the command prompt.

- Class: cmd_question
  Output:  Now do the same for cnt1. Name this new variable pm1sub.
  CorrectAnswer: pm1sub <- subset(cnt1, County.Code==63 & Site.ID==2008)
  AnswerTests:   expr_creates_var("pm1sub"); omnitest(correctExpr='pm1sub <- subset(cnt1, County.Code==63 & Site.ID==2008)')
  Hint: Type pm1sub <- subset(cnt1, County.Code==63 & Site.ID==2008) at the command prompt.
```

```
- Class: mult_question
  Output: From the output of the 2 calls to sapply, how many rows will pm0sub have?
  AnswerChoices: 122; 30; 29; 183
  CorrectAnswer:  122
  AnswerTests: omnitest(correctVal='122')
  Hint: Recall monitor 63.2008 had 122 measurements in 1999 and 30 in 2012.

- Class: cmd_question
  Output:  Now we'd like to compare the pm25 measurements of this particular monitor (63.2008) for the 2 years. First, create the vector x0sub by assigning to it the Sample.Value component of pm0sub.
  CorrectAnswer: x0sub <- pm0sub$Sample.Value
  AnswerTests:   expr_creates_var("x0sub"); omnitest(correctExpr='x0sub <- pm0sub$Sample.Value')
  Hint: Type x0sub <- pm0sub$Sample.Value at the command prompt.

- Class: cmd_question
  Output:  Similarly, create x1sub from pm1sub.
  CorrectAnswer: x1sub <- pm1sub$Sample.Value
  AnswerTests:   expr_creates_var("x1sub"); omnitest(correctExpr='x1sub <- pm1sub$Sample.Value')
  Hint: Type x1sub <- pm1sub$Sample.Value at the command prompt.

- Class: cmd_question
  Output:  We'd like to make our comparison visually so we'll have to create a time series of these pm25 measurements. First, create a dates0 variable by assigning to it the output of a call to as.Date. This will take 2 arguments. The first is a call to as.character with pm0sub$Date as the argument. The second is the format string "%Y%m%d".
  CorrectAnswer: dates0 <- as.Date(as.character(pm0sub$Date),"%Y%m%d")
  AnswerTests:   expr_creates_var("dates0"); omnitest(correctExpr='dates0 <- as.Date(as.character(pm0sub$Date),"%Y%m%d")')
  Hint: Type dates0 <- as.Date(as.character(pm0sub$Date),"%Y%m%d") at the command prompt.

- Class: cmd_question
  Output:  Do the same for the 2012 data. Specifically, create dates1 using pm1sub$Date as your input.
  CorrectAnswer: dates1 <- as.Date(as.character(pm1sub$Date),"%Y%m%d")
  AnswerTests:   expr_creates_var("dates1"); omnitest(correctExpr='dates1 <- as.Date(as.character(pm1sub$Date),"%Y%m%d")')
  Hint: Type dates1 <- as.Date(as.character(pm1sub$Date),"%Y%m%d") at the command prompt.

- Class: cmd_question
  Output:  Now we'll plot these 2 time series in the same panel using the base plotting system. Call par with 2 arguments. The first is mfrow set equal to c(1,2). This will tell the system we're plotting 2 graphs in 1 row and 2 columns. The second argument will adjust the panel's margins. It is mar set to c(4,4,2,1).
  CorrectAnswer: par(mfrow = c(1, 2), mar = c(4, 4, 2, 1))
  AnswerTests:   omnitest(correctExpr='par(mfrow = c(1, 2), mar = c(4, 4, 2, 1))')
  Hint: Type par(mfrow = c(1, 2), mar = c(4, 4, 2, 1)) at the command prompt.

- Class: cmd_question
  Output:  Call plot with the 3 arguments dates0, x0sub, and pch set to 20. The first two arguments are the x and y coordinates. This will show the pm25 values as functions of time.
  CorrectAnswer: plot(dates0, x0sub, pch = 20)
  AnswerTests:   omnitest(correctExpr='plot(dates0, x0sub, pch = 20)')
  Hint: Type plot(dates0, x0sub, pch = 20) at the command prompt.

- Class: figure
  Output: Now we'll mark the median.
  Figure: goodPlot1.R

- Class: cmd_question
  Output:  Use abline to add a horizontal line at the median of the pm25 values. Make the line width 2 (lwd is the argument), and when you call median with x0sub, specify the argument na.rm to be TRUE.
  CorrectAnswer: abline(h = median(x0sub, na.rm = TRUE),lwd=2)
```

```yaml
  AnswerTests:   omnitest(correctExpr='abline(h = median(x0sub, na.rm = TRUE),lwd=2)')
  Hint: Type abline(h = median(x0sub, na.rm = TRUE),lwd=2) at the command prompt.

- Class: cmd_question
  Output:  Now we'll do the same for the 2012 data. Call plot with the 3 arguments dates1, x1sub, and pch set to 20.
  CorrectAnswer: plot(dates1, x1sub, pch = 20)
  AnswerTests:   omnitest(correctExpr='plot(dates1, x1sub, pch = 20)')
  Hint: Type plot(dates1, x1sub, pch = 20) at the command prompt.

- Class: figure
  Output: As before,  we'll mark the median of this 2012 data.
  Figure: goodPlot2.R

- Class: cmd_question
  Output:  Use abline to add a horizontal line at the median of the pm25 values. Make the line width 2 (lwd is the argument). Remember to  specify the argument na.rm to be TRUE when you call median on x1sub.
  CorrectAnswer: abline(h = median(x1sub, na.rm = TRUE),lwd=2)
  AnswerTests:   omnitest(correctExpr='abline(h = median(x1sub, na.rm = TRUE),lwd=2)')
  Hint: Type abline(h = median(x1sub, na.rm = TRUE),lwd=2) at the command prompt.

- Class: mult_question
  Output: Which median is larger - the one for 1999 or the one for 2012?
  AnswerChoices: 1999; 2012
  CorrectAnswer:  1999
  AnswerTests: omnitest(correctVal='1999')
  Hint: Look carefully at the numbers on the y-axis.

- Class: mult_question
  Output: The picture makes it look like the median is higher for 2012 than 1999. Closer inspection shows that this isn't true. The median for 1999 is a little over 10 micrograms per cubic meter and for 2012 its a little over 8. The plots appear this way because the 1999 plot ....
  AnswerChoices: shows a bigger range of y values than the 2012 plot; displays more points than the 2012 plot; shows different months than those in the 2012 plot
  CorrectAnswer:  shows a bigger range of y values than the 2012 plot
  AnswerTests: omnitest(correctVal='shows a bigger range of y values than the 2012 plot')
  Hint: Look at the range of y values in the two plots.

- Class: cmd_question
  Output:  The 1999 plot shows a much bigger range of pm25 values on the y axis, from below 10  to 40, while the 2012 pm25 values are much more restricted, from around 1 to 14. We should really plot the points of both datasets on the same range of values on the y axis. Create the variable rng by assigning to it the output of a call to the R command range with 3 arguments, x0sub, x1sub, and the boolean na.rm set to TRUE.
  CorrectAnswer: rng <- range(x0sub,x1sub,na.rm=TRUE)
  AnswerTests:   expr_creates_var("rng"); omnitest(correctExpr='rng <- range(x0sub,x1sub,na.rm=TRUE)')
  Hint: Type rng <- range(x0sub,x1sub,na.rm=TRUE) at the command prompt.

- Class: cmd_question
  Output:  Look at rng to see the values it spans.
  CorrectAnswer: rng
  AnswerTests:    ANY_of_exprs("rng","print(rng)")
  Hint: Type rng or print(rng) at the command prompt.

- Class: figure
  Output:  Here a new figure we've created showing the two plots side by side with the same range of values on the y axis. We used the argument ylim set equal to rng in our 2 calls to plot. The improvement in the medians between 1999 and 2012 is now clear. Also notice that in 2012 there are no big values (above 15). This shows that not only is there a chronic improvement in air quality, but also there are fewer days with severe pollution.
  Figure: showBoth.R
  FigureType: new

- Class: text
  Output: The last avenue of this data we'll explore (and we'll do it quickly) concerns
```

```
  a comparison of all the states' mean pollution levels. This is important because the states are responsible for implementing the regulations set at the federal level by the EPA.

- Class: text
  Output: Let's first gather the mean (average measurement) for each state in 1999. Recall that the original data for this year was stored in pm0.

- Class: cmd_question
  Output:   Create the vector mn0 with a call to the R command with using 2 arguments. The first is pm0. This is the data in which the second argument, an expression, will be evaluated. The second argument is a call to the function tapply. This call requires 4 arguments. Sample.Value and State.Code are the first two. We want to apply the function mean to Sample.Value, so mean is the third argument. The fourth is simply the boolean na.rm set to TRUE.
  CorrectAnswer: mn0 <- with(pm0,tapply(Sample.Value,State.Code,mean,na.rm=TRUE))
  AnswerTests:    expr_creates_var("mn0"); omnitest(correctExpr='mn0 <- with(pm0,tapply(Sample.Value,State.Code,mean,na.rm=TRUE))')
  Hint: Type  mn0 <- with(pm0,tapply(Sample.Value,State.Code,mean,na.rm=TRUE)) at the command prompt.

- Class: cmd_question
  Output:   Call the function str with mn0 as its argument to see what it looks like.
  CorrectAnswer: str(mn0)
  AnswerTests:     omnitest(correctExpr='str(mn0)')
  Hint: Type  str(mn0) at the command prompt.

- Class: text
  Output: We see mn0 is a 53 long numerical vector. Why 53 if there are only 50 states? As it happens, pm25 measurements for  the District of Columbia (Washington D.C), the Virgin Islands, and Puerto Rico are included in this data. They are coded as 11, 72, and 78 respectively.

- Class: cmd_question
  Output: Recall your command creating mn0 and change it to create mn1 using pm1 as the first input to the call to with.
  CorrectAnswer: mn1 <- with(pm1,tapply(Sample.Value,State.Code,mean,na.rm=TRUE))
  AnswerTests:    expr_creates_var("mn1"); omnitest(correctExpr='mn1 <- with(pm1,tapply(Sample.Value,State.Code,mean,na.rm=TRUE))')
  Hint: Type  mn1 <- with(pm1,tapply(Sample.Value,State.Code,mean,na.rm=TRUE)) at the command prompt.

- Class: cmd_question
  Output:  For fun, call the function str with mn1 as its argument.
  CorrectAnswer: str(mn1)
  AnswerTests:     omnitest(correctExpr='str(mn1)')
  Hint: Type  str(mn1) at the command prompt.

- Class: cmd_question
  Output:  So mn1 has only 52 entries, rather than 53. We checked. There are no entries for the Virgin Islands in 2012. Call summary now with mn0 as its input.
  CorrectAnswer: summary(mn0)
  AnswerTests:     omnitest(correctExpr='summary(mn0)')
  Hint: Type  summary(mn0) at the command prompt.

- Class: cmd_question
  Output:  Now call summary with mn1 as its input so we can compare the two years.
  CorrectAnswer: summary(mn1)
  AnswerTests:     omnitest(correctExpr='summary(mn1)')
  Hint: Type  summary(mn1) at the command prompt.

- Class: cmd_question
  Output:  We see that in all 6 entries, the 2012 numbers are less than those in 1999. Now we'll create 2 new dataframes containing just the state names and their mean measurements for each year. First, we'll do this for 1999. Create the data frame d0 by calling the function data.frame with 2 arguments. The first is state set equal to names(mn0), and the second is mean set equal to mn0.
  CorrectAnswer: d0 <- data.frame(state = names(mn0), mean = mn0)
  AnswerTests:   expr_creates_var("d0");  omnitest(correctExpr='d0 <- data.frame(state
```

```
  = names(mn0), mean = mn0)')
  Hint: Type  d0 <- data.frame(state = names(mn0), mean = mn0) at the command prompt.

- Class: cmd_question
  Output: Recall the last command and create d1 instead of d0 using the 2012 data. (There'll be 3 changes of 0 to 1.)
  CorrectAnswer: d1 <- data.frame(state = names(mn1), mean = mn1)
  AnswerTests:   expr_creates_var("d1");  omnitest(correctExpr='d1 <- data.frame(state = names(mn1), mean = mn1)')
  Hint: Type  d1 <- data.frame(state = names(mn1), mean = mn1) at the command prompt.

- Class: cmd_question
  Output: Create the array mrg by calling the R command merge with 3 arguments, d0, d1, and the argument by set equal to the string "state".
  CorrectAnswer: mrg <- merge(d0, d1, by = "state")
  AnswerTests:   expr_creates_var("mrg");  omnitest(correctExpr='mrg <- merge(d0, d1, by = "state")')
  Hint: Type  mrg <- merge(d0, d1, by = "state") at the command prompt.

- Class: cmd_question
  Output: Run dim with mrg as its argument to see how big it is.
  CorrectAnswer: dim(mrg)
  AnswerTests:    omnitest(correctExpr='dim(mrg)')
  Hint: Type  dim(mrg) at the command prompt.

- Class: cmd_question
  Output: We see merge has 52 rows and 3 columns. Since the Virgin Island data was missing from d1, it is excluded from mrg. Look at the first few entries of mrg using the head command.
  CorrectAnswer: head(mrg)
  AnswerTests:    omnitest(correctExpr='head(mrg)')
  Hint: Type  head(mrg) at the command prompt.

- Class: text
  Output: Each row of mrg has 3 entries - a state identified by number, a state mean for 1999 (mean.x), and a state mean for 2012 (mean.y).

- Class: text
  Output: Now we'll plot the data to see how the state means changed between the 2 years. First we'll plot the 1999 data in a single column at x=1. The y values for the points will be the state means. Again, we'll use the R command with so we don't have to keep typing mrg as the data environment in which to evaluate the second argument, the call to plot. We've already reset the graphical parameters for you.

- Class: cmd_question
  Output: For the first column of points, call with with 2 arguments. The first is mrg, and the second is the call to plot with 3 arguments. The first of these is rep(1,52). This tells the plot routine that the x coordinates for all 52 points are 1. The second argument is the second column of mrg or mrg[,2] which holds the 1999 data. The third argument is the range of x values we want, namely xlim set to c(.5,2.5). This works since we'll be plotting 2 columns of points, one at x=1 and the other at x=2.
  CorrectAnswer: with(mrg, plot(rep(1, 52), mrg[, 2], xlim = c(.5, 2.5)))
  AnswerTests:    omnitest(correctExpr='with(mrg, plot(rep(1, 52), mrg[, 2], xlim = c(.5, 2.5)))')
  Hint: Type  with(mrg, plot(rep(1, 52), mrg[, 2], xlim = c(.5, 2.5))) at the command prompt.

- Class: cmd_question
  Output: We see a column of points at x=1 which represent the 1999 state means. For the second column of points, again call with with 2 arguments. As before, the first is mrg. The second, however,  is a call to the function points with 2 arguments. We need to do this since we're adding points to an already existing plot. The first argument to points is the set of x values, rep(2,52). The second argument is the set of y values, mrg[,3]. Of course, this is the third column of mrg. (We don't need to specify the range of x values again.)
  CorrectAnswer: with(mrg, points(rep(2, 52), mrg[, 3]))
  AnswerTests:    omnitest(correctExpr='with(mrg, points(rep(2, 52), mrg[, 3]))')
  Hint: Type  with(mrg, points(rep(2, 52), mrg[, 3])) at the command prompt.
```

```yaml
- Class: cmd_question
  Output: We see a shorter column of points at x=2. Now let's connect the dots. Use the R function segments with 4 arguments. The first 2 are the x and y coordinates of the 1999 points and the last 2 are the x and y coordinates of the 2012 points. As in the previous calls specify the x coordinates with calls to rep and the y coordinates with references to the appropriate columns of mrg.
  CorrectAnswer: segments(rep(1, 52), mrg[, 2], rep(2, 52), mrg[, 3])
  AnswerTests:    omnitest(correctExpr='segments(rep(1, 52), mrg[, 2], rep(2, 52), mrg[, 3])')
  Hint: Type  segments(rep(1, 52), mrg[, 2], rep(2, 52), mrg[, 3]) at the command prompt.

- Class: text
  Output: We see from the plot that the vast majority of states have indeed improved their particulate matter counts so the general trend is downward. There are a few exceptions. (The topmost point in the 1999 column is actually two points that had very close measurements.)

- Class: cmd_question
  Output: For fun, let's see which states had higher means in 2012 than in 1999. Just use the mrg[mrg$mean.x < mrg$mean.y, ] notation to find the rows of mrg with this particulate property.
  CorrectAnswer: mrg[mrg$mean.x < mrg$mean.y,]
  AnswerTests:    omnitest(correctExpr='mrg[mrg$mean.x < mrg$mean.y,]')
  Hint: Type  mrg[mrg$mean.x < mrg$mean.y,] at the command prompt.

- Class: text
  Output: Only 4 states had worse pollution averages, and 2 of these had means that were very close. If you want to see which states (15, 31, 35, and 40) these are, you can check out this website https://www.epa.gov/enviro/state-fips-code-listing to decode the state codes.

- Class: text
  Output: This concludes the lesson, comparing air pollution data from two years in different ways. First, we looked at measures of the entire set of monitors, then we compared the two measures from a particular monitor, and finally, we looked at the mean measures of the individual states.

- Class: text
  Output: Congratulations! We hope you enjoyed this particulate 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: ""
```