```
  Course: Exploratory_Data_Analysis
  Lesson: Dimension_Reduction

- Class: text
  Output: "Dimension_Reduction. (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/dimensionReduction.)"


- Class: text
  Output:  In this lesson we'll discuss principal component analysis (PCA) and singular value decomposition (SVD), two important and related techniques of dimension reduction. This last entails processes which finding subsets of variables in datasets that contain their essences. PCA and SVD are used in both the exploratory phase and the more formal modelling stage of analysis. We'll focus on the exploratory phase and briefly touch on some of the underlying theory.

- Class: figure
  Output:  We'll begin with a motivating example - random data.
  Figure: showRanMat.R
  FigureType: new

- Class: cmd_question
  Output:  This is dataMatrix, a matrix of 400 random normal numbers (mean 0 and standard deviation 1).  We're displaying it with the R command image. Run the R command head with dataMatrix as its argument to see what dataMatrix looks like.
  CorrectAnswer: head(dataMatrix)
  AnswerTests: omnitest(correctExpr='head(dataMatrix)')
  Hint: Type head(dataMatrix) at the command prompt.

- Class: cmd_question
  Output:  So we see that dataMatrix has 10 columns (and hence 40 rows) of random numbers. The image here looks pretty random. Let's see how the data clusters. Run the R command heatmap with dataMatrix as its only argument.
  CorrectAnswer: heatmap(dataMatrix)
  AnswerTests: omnitest(correctExpr='heatmap(dataMatrix)')
  Hint: Type heatmap(dataMatrix) at the command prompt.

- Class: text
  Output: We can see that even with the clustering that heatmap provides, permuting the rows (observations) and columns (variables) independently, the data still looks random.

- Class: cmd_question
  Output: Let's add a pattern to the data. We've put some R code in the file addPatt.R for you. Run the command myedit with the single argument "addPatt.R" (make sure to use the quotation marks) to see the code. You might have to click your cursor in the console after you do this to keep from accidentally changing the file.
  CorrectAnswer: myedit("addPatt.R")
  AnswerTests: omnitest(correctExpr='myedit("addPatt.R")')
  Hint: Type myedit("addPatt.R") at the command prompt.

- Class: mult_question
  Output: Look at the code. Will every row of the matrix have a pattern added to it?
  AnswerChoices:  Yes; No
  CorrectAnswer:  No
  AnswerTests: omnitest(correctVal='No')
  Hint: What  does the coinflip do?

- Class: mult_question
  Output: So whether or not a row gets modified by a pattern is determined by a coin flip. Will the added pattern affect every column in the affected row?
  AnswerChoices:  Yes; No
  CorrectAnswer:  No
  AnswerTests: omnitest(correctVal='No')
  Hint: The expression rep(c(0,3),each=5) creates the 10-long vector (0,0,0,0,0,3,3,3,3,3) which is added to the rows chosen by the coin flip.

```

```yaml
- Class: text
  Output: So in rows affected by the coin flip, the 5 left columns will still have a mean of 0 but the right 5 columns will have a mean closer to 3.

- Class: cmd_question
  Output: Now to execute this code, run the R command source with 2 arguments. The first is the filename (in quotes), "addPatt.R", and the second is the argument local set equal to TRUE.
  CorrectAnswer: source("addPatt.R", local=TRUE)
  AnswerTests: omnitest(correctExpr='source("addPatt.R", local=TRUE)')
  Hint: Type source("addPatt.R", local=TRUE) at the command prompt.

- Class: figure
  Output:  Here's the image of the altered dataMatrix after the pattern has been added. The pattern is clearly visible in the columns of the matrix. The right half is yellower or hotter, indicating higher values in the matrix.
  Figure: showRanMat.R
  FigureType: new

- Class: cmd_question
  Output:  Now run the R command heatmap again with dataMatrix as its only argument. This will perform    a hierarchical cluster analysis on the matrix.
  CorrectAnswer: heatmap(dataMatrix)
  AnswerTests: omnitest(correctExpr='heatmap(dataMatrix)')
  Hint: Type heatmap(dataMatrix) at the command prompt.

- Class: text
  Output: Again we see the pattern in the columns of the matrix. As shown in the dendrogram at the top of the display, these split into 2 clusters, the lower numbered columns (1 through 5) and the higher numbered ones (6 through 10). Recall from the code in addPatt.R that for rows selected by the coinflip the last 5 columns had 3 added to them. The rows still look random.

- Class: figure
  Output: Now consider this picture. On the left is an image similar to the heatmap of dataMatix you just plotted. It is an image plot of the output of hclust(), a hierarchical clustering function applied to dataMatrix. Yellow indicates "hotter" or higher values than red. This is consistent with the pattern we applied to the data (increasing the values for some of the rightmost columns).
  Figure: showPatt.R
  FigureType: new

- Class: text
  Output: The middle display shows the mean of each of the 40 rows (along the x-axis). The rows are shown in the same order as the rows of the heat matrix on the left. The rightmost display shows the mean of each of the 10 columns. Here the column numbers are along the x-axis and their means along the y.

- Class: text
  Output: We see immediately the connection between the yellow (hotter) portion of the cluster image and the higher row means, both in the upper right portion of the displays. Similarly, the higher valued column means are in the right half of that display and lower colummn means are in the left half.

- Class: text
  Output: Now we'll talk a little theory. Suppose you have 1000's of multivariate variables X_1, ... ,X_n. By multivariate we mean that each X_i contains many components, i.e., X_i = (X_{i1}, ... , X_{im}. However, these variables (observations) and their components might be correlated to one another.

- Class: mult_question
  Output: Which of the following would be an example of variables correlated to one another?
  AnswerChoices:  Heights and weights of members of a family; Today's weather and a butterfly's wing position; The depth of the Atlantic Ocean and what you eat for breakfast
  CorrectAnswer:  Heights and weights of members of a family
  AnswerTests: omnitest(correctVal='Heights and weights of members of a family')
  Hint: Which choice is the only one that makes sense?
```

```
- Class: text
  Output: As data scientists, we'd like to find a smaller set of multivariate variables that are uncorrelated AND explain as much variance (or variability) of the data as possible. This is a statistical approach.

- Class: text
  Output: In other words, we'd like to find the best matrix created with fewer variables (that is, a lower rank matrix) that explains the original data. This is related to data compression.

- Class: text
  Output: Two related solutions to these problems are PCA which stands for Principal Component Analysis and SVD, Singular Value Decomposition. This latter simply means that we express a matrix X of observations (rows) and variables (columns) as the product of 3 other matrices, i.e., X=UDV^t. This last term (V^t) represents the transpose of the matrix V.

- Class: text
  Output: Here U and V each have orthogonal (uncorrelated) columns. U's columns are the left singular vectors of X and V's columns are the right singular vectors of X.  D is a diagonal matrix, by which we mean that all of its entries not on the diagonal are 0. The diagonal entries of D are  the singular values of X.

- Class: cmd_question
  Output:  To illustrate this idea we created a simple example matrix  called  mat. Look at it now.
  CorrectAnswer: mat
  AnswerTests: omnitest(correctExpr='mat')
  Hint: Type mat at the command prompt.

- Class: cmd_question
  Output:  So mat is a 2 by 3 matrix. Lucky for us R provides a function to perform singular value decomposition. It's called, unsurprisingly, svd. Call it now with a single argument, mat.
  CorrectAnswer: svd(mat)
  AnswerTests: omnitest(correctExpr='svd(mat)')
  Hint: Type svd(mat) at the command prompt.

- Class: cmd_question
  Output:  We see that the function returns 3 components, d which holds 2 diagonal elements, u, a 2 by 2 matrix, and v, a 3 by 2 matrix. We stored the diagonal entries in a diagonal matrix for you, diag, and we also stored u and v in the variables matu and matv respectively. Multiply matu by diag by      t(matv) to see what you get. (This last expression represents the transpose of matv in R). Recall that in R matrix multiplication requires you to use the operator %*%.
  CorrectAnswer: matu %*% diag %*% t(matv)
  AnswerTests: omnitest(correctExpr='matu %*% diag %*% t(matv)')
  Hint: Type matu %*% diag %*% t(matv) at the command prompt.

- Class: text
  Output: So we did in fact get mat back. That's a relief! Note that this type of decomposition is NOT unique.

- Class: text
  Output: Now we'll talk a little about PCA, Principal Component Analysis, "a simple, non-parametric method for extracting relevant information from confusing data sets." We're quoting here from a very nice concise paper on this subject which can be found at http://arxiv.org/pdf/1404.1100.pdf. The paper by Jonathon Shlens of Google Research is called, A Tutorial on Principal Component Analysis.

- Class: text
  Output: Basically, PCA is a method to reduce a high-dimensional data set to its essential elements (not lose information) and explain the variability in the data. We won't go into the mathematical details here, (R has a function to perform PCA), but you should know that SVD and PCA are closely related.

- Class: cmd_question
  Output:  We'll demonstrate this now. First we have to scale mat, our simple example
```

```yaml
  data matrix.  This means that we subtract the column mean from every element and
  divide the result by the column standard deviation. Of course R has a command, scale,
  that does this for you. Run svd on scale of mat.
  CorrectAnswer: svd(scale(mat))
  AnswerTests: omnitest(correctExpr='svd(scale(mat))')
  Hint: Type svd(scale(mat)) at the command prompt.

- Class: cmd_question
  Output: Now run the R program prcomp on scale(mat). This will give you the principal
  components of mat. See if they look familiar.
  CorrectAnswer: prcomp(scale(mat))
  AnswerTests: omnitest(correctExpr='prcomp(scale(mat))')
  Hint: Type prcomp(scale(mat)) at the command prompt.

- Class: text
  Output: Notice that the principal components of the scaled matrix, shown in the
  Rotation component of the prcomp output, ARE the columns of V, the right singular
  values. Thus, PCA of a scaled matrix yields the V matrix (right singular vectors) of
  the same scaled matrix.

- Class: text
  Output: Now that we covered the theory let's return to our bigger matrix of random
  data into which we had added a fixed pattern for some rows selected by coinflips. The
  pattern effectively shifted the means of the rows and columns.

- Class: figure
  Output: Here's a picture showing the relationship between PCA and SVD for that bigger
  matrix.  We've plotted 10 points (5 are squished together in the bottom left corner).
  The x-coordinates are the elements of the first principal component (output from
  prcomp), and the y-coordinates are the elements of the first column of V, the first
  right singular vector (gotten from running svd). We see that the points all lie on
  the 45 degree line represented by the equation y=x.   So the first column of V IS the
  first principal component of our bigger data matrix.
  Figure: showRel.R
  FigureType: new

- Class: cmd_question
  Output: To prove we're not making this up, we've run svd on dataMatrix and stored the
  result in the object svd1. This has 3 components, d, u and v. look at the first
  column of V now. It can be viewed by using the svd1$v[,1] notation.
  CorrectAnswer: svd1$v[,1]
  AnswerTests: omnitest(correctExpr='svd1$v[,1]')
  Hint: Type svd1$v[,1] at the command prompt.

- Class: text
  Output: See how these values correspond to those plotted? Five of the entries are
  slightly to the left of the point (-0.4,-0.4), two more are negative (to the left of
  (0,0)), and three are positive (to the right of (0,0)).

- Class: figure
  Output: Here we again show the clustered data matrix on the left. Next to it we've
  plotted the first column of the U matrix associated with the scaled data matrix. This
  is the first LEFT singular vector and it's associated with the ROW means of the
  clustered data. You can see the clear separation between the top 24 (around -0.2) row
  means and the bottom 16 (around 0.2). We don't show them but note that the other
  columns of U don't show this pattern so clearly.
  Figure: showUV.R
  FigureType: new

- Class: text
  Output: The rightmost display shows the first column of the V matrix associated with
  the scaled and clustered data matrix. This is the first RIGHT singular vector and
  it's associated with the COLUMN means of the clustered data. You can see the clear
  separation between the left 5 column means (between -0.1 and 0.1) and the right 5
  column means (all below -0.4). As with the left singular vectors, the other columns
  of V don't show this pattern as clearly as this first one does.

- Class: text
  Output: So the singular value decomposition automatically picked up these patterns,
```

```yaml
  the differences in the row and column means.

- Class: cmd_question
  Output: Why were the first columns of both the U and V matrices so special?  Well as it happens, the D matrix of the SVD explains this phenomenon. It is an aspect of SVD called variance explained. Recall that D is the diagonal matrix sandwiched in between U and V^t in the SVD representation of the data matrix. The diagonal entries of D are like weights for the U and V columns accounting for the variation in the data. They're given in decreasing order from highest to lowest. Look at these diagonal entries now. Recall that they're stored in svd1$d.
  CorrectAnswer: svd1$d
  AnswerTests: omnitest(correctExpr='svd1$d')
  Hint: Type svd1$d at the command prompt.

- Class: figure
  Output: Here's a display of these values (on the left). The first one (12.46) is significantly bigger than the others. Since we don't have any units specified, to the right we've plotted the proportion of the variance each entry represents. We see that the first entry accounts for about 40% of the variance in the data. This explains why the first columns of the U and V matrices respectively showed the distinctive patterns in the row and column means so clearly.
  Figure: showVar.R
  FigureType: new

- Class: cmd_question
  Output:  Now we'll show you another simple example of how SVD explains variance. We've created a 40 by 10 matrix, constantMatrix. Use the R command head with constantMatrix as its argument to see the top rows.
  CorrectAnswer: head(constantMatrix)
  AnswerTests: omnitest(correctExpr='head(constantMatrix)')
  Hint: Type head(constantMatrix) at the command prompt.

- Class: cmd_question
  Output: The rest of the rows look just like these. You can see that the left 5 columns are all 0's and the right 5 columns are all 1's. We've run svd with constantMatrix as its argument for you and stored the result in svd2. Look at the diagonal component,  d, of svd2 now.
  CorrectAnswer: svd2$d
  AnswerTests: omnitest(correctExpr='svd2$d')
  Hint: Type svd2$d at the command prompt.

- Class: mult_question
  Output: Which index holds  the largest entry of the svd2$d?
  AnswerChoices:  9; 1; 10; 5
  CorrectAnswer:  1
  AnswerTests: omnitest(correctVal='1')
  Hint: Which choice has a positive exponent? Notice that the entries are given in scientific notation, where xey means x * 10^y.

- Class: figure
  Output: So the first entry by far dominates the others. Here the picture on the left shows the heat map of  constantMatrix. You can see how the left columns differ from the right ones. The middle plot shows the values of the singular values of the matrix, i.e., the diagonal elements which are the entries of svd2$d. Nine of these are 0 and the first is a little above 14. The third plot shows the proportion of the total each diagonal element represents.
  Figure: showSimple.R
  FigureType: new

- Class: mult_question
  Output: According to the plot, what percentage of the total variation does the first diagonal element account for?
  AnswerChoices:  100%; 90%; 50%; 0%
  CorrectAnswer:  100%
  AnswerTests: omnitest(correctVal='100%')
  Hint: The first element is at 1.0. Which percent does the number 1.0 represent?

- Class: text
  Output: So what does this mean? Basically that the data is one-dimensional. Only 1
```

```yaml
  piece of information, namely which column an entry is in, determines its value.

- Class: figure
  Output: Now let's return to our random 40 by 10 dataMatrix and consider a slightly more complicated example in which we add 2 patterns to it. Again we'll choose which rows to tweak using coinflips. Specifically, for each of the 40 rows we'll flip 2 coins. If the first coinflip is heads, we'll add 5 to each entry in the right 5 columns of that row, and if the second coinflip is heads, we'll add  5 to just the even columns of that row.
  Figure: twoPatts.R
  FigureType: new

- Class: figure
  Output: So here's the image of the scaled data matrix on the left. We can see both patterns, the clear difference between the left 5 and right 5 columns, but also, slightly less visible, the alternating pattern of the columns. The other plots show the true patterns that were added into the affected rows. The middle plot shows the true difference between the left and right columns, while the rightmost plot shows the true difference between the odd numbered and even-numbered columns.
  Figure: showTrue.R
  FigureType: new

- Class: text
  Output: The question is, "Can our analysis detect these patterns just from the data?" Let's see what SVD shows. Since we're interested in patterns on columns we'll look at the first two right singular vectors (columns of V) to see if they show any evidence of the patterns.

- Class: figure
  Output: Here we see the 2 right singular vectors plotted next to the image of the data matrix. The middle plot shows the first column of V and the rightmost plot the second. The middle plot does show that the last 5 columns have higher entries than the first 5. This picks up, or at least alludes to, the first pattern we added in which affected the last 5 columns of the matrix. The rightmost plot, showing the second column of V, looks more random. However, closer inspection shows that the entries alternate or bounce up and down as you move from left to right. This hints at the second pattern we added in which affected only even columns of selected rows.
  Figure: showVs.R
  FigureType: new

- Class: cmd_question
  Output:  To see this more closely, look at the first 2 columns of the v component. We stored the SVD output in the svd object svd2.
  CorrectAnswer: svd2$v[,1:2]
  AnswerTests: ANY_of_exprs('svd2$v[,1:2]','svd2$v[,c(1,2)]','svd2$v[,c(1:2)]')
  Hint: Type svd2$v[,1:2] at the command prompt.

- Class: figure
  Output: Seeing the 2 columns side by side, we see that the values in both columns alternately increase and decrease. However, we knew to look for this pattern, so chances are, you might not have noticed this pattern if you hadn't known if was there. This example is meant to show you that it's hard to see patterns, even straightforward ones.
  Figure: clearPlot.R

- Class: cmd_question
  Output:  Now look at the entries of the diagonal matrix d resulting from the svd. Recall that we stored this output for you in the svd object svd2.
  CorrectAnswer: svd2$d
  AnswerTests: omnitest(correctExpr='svd2$d')
  Hint: Type svd2$d at the command prompt.

- Class: figure
  Output:  We see that the first element, 14.55, dominates the others. Here's the plot of these diagonal elements of  d. The left shows the numerical entries and the right show the percentage of variance each entry explains.
  Figure: showDvar.R
  FigureType: new

```

```
- Class: mult_question
  Output: According to the plot, how much of the variance does the second element account for?
  AnswerChoices:  18%; 53%; 11%; .1%
  CorrectAnswer: 18%
  AnswerTests: omnitest(correctVal='18%')
  Hint: At what height does the second point fall in the chart on the right?

- Class: text
  Output: So the first element which showed the difference between the left and right halves of the matrix accounts for roughly 50% of the variation in the matrix, and the second element which picked up the alternating pattern accounts for 18% of the variance. The remaining elements account for smaller percentages of the variation. This indicates that the first pattern is much stronger than the second. Also the two patterns confound each other so they're harder to separate and see clearly. This is what often happens with real data.

- Class: text
  Output: Now you're probably convinced that SVD and PCA are pretty cool and useful as tools for analysis, but one problem with them that you should be aware of, is that they cannot deal with MISSING data. Neither of them will work if any data in the matrix is missing. (You'll get error messages from R in red if you try.) Missing data is not unusual, so luckily we have ways to work around this problem. One we'll just mention is called imputing the data.

- Class: text
  Output:  This uses the k nearest neighbors to calculate a values to use in place of the missing data. You may want to specify an integer k which indicates how many neighbors you want to average to create this replacement value. The bioconductor package (http://bioconductor.org) has an impute package which you can use to fill in missing data. One specific function in it is impute.knn.

- Class: text
  Output: We'll move on now to a final example of the power of singular value decomposition and principal component analysis and how they work as a data compression technique.

- Class: figure
  Output: Consider this low resolution image file showing a face. We'll use SVD and see how the first several components contain most of the information in the file so that storing a huge matrix might not be necessary.
  Figure: showFace.R
  FigureType: new

- Class: cmd_question
  Output:  The image data is stored in the matrix faceData. Run the R command dim on faceData to see how big it is.
  CorrectAnswer: dim(faceData)
  AnswerTests: omnitest(correctExpr='dim(faceData)')
  Hint: Type dim(faceData) at the command prompt.

- Class: figure
  Output: So it's not that big of a file but we want to show you how to use what you learned in this lesson. We've done the SVD and stored it in the object svd1 for you. Here's the plot of the variance explained.
  Figure: showFaceSVD.R
  FigureType: new

- Class: mult_question
  Output: According to the plot what percentage of the variance is explained by the first singular value?
  AnswerChoices:  23; 15; 40; 100
  CorrectAnswer: 40
  AnswerTests: omnitest(correctVal='40')
  Hint: At what height does the first (leftmost) point fall in the plot?

- Class: text
  Output: So 40% of the variation in the data matrix is explained by the first component, 22% by the second, and so forth. It looks like most of the variation is
```

```
contained in the first 10 components. How can we check this out? Can we try to create an approximate image using only a few components?

  - Class: text
    Output: Recall that the data matrix X is the product of 3 matrices, that is X=UDV^t. These are precisely what you get when you run  svd on the  matrix X.

  - Class: text
    Output: Suppose we create the product of pieces of these, say the first columns of U and V and the first element of D. The first column of U can be interpreted as a 32 by 1 matrix  (recall that faceData was a 32 by 32 matrix), so we can multiply it by the first element of D, a 1 by 1 matrix, and get a 32 by 1 matrix result. We can multiply that by the transpose of the first column of V, which is the first principal component. (We have to use the transpose of V's column to make it a 1 by 32 matrix in order to do the matrix multiplication properly.)

  - Class: text
    Output: Alas, that is how we do it in theory, but in R using only one element of d means it's a constant. So we have to do the matrix multiplication with the %*% operator and the multiplication by the constant (svd1$d[1]) with the regular multiplication operator *.

  - Class: cmd_question
    Output: Try this now and put the result in the variable a1. Recall that svd1$u, svd1$d, and svd1$v contain all the information you need. NOTE that because of the peculiarities of R's casting, if you do the scalar multiplication with the * operator first (before the matrix multiplication with the %*% operator) you MUST enclose the 2 arguments (svd1$u[,1] and svd1$d[1]) in parentheses.
    CorrectAnswer: a1 <- svd1$u[,1] %*% t(svd1$v[,1]) * svd1$d[1]
    AnswerTests: expr_creates_var("a1"); ANY_of_exprs('a1 <- svd1$u[,1] %*% t(svd1$v[,1]) * svd1$d[1]','a1 <- (svd1$d[1] * svd1$u[,1]) %*% t(svd1$v[,1])','a1 <- (svd1$u[,1] * svd1$d[1]) %*% t(svd1$v[,1])')
    Hint: Type a1 <- (svd1$u[,1] * svd1$d[1]) %*% t(svd1$v[,1]) OR a1 <- svd1$u[,1] %*% t(svd1$v[,1]) * svd1$d[1] at the command prompt.

  - Class: cmd_question
    Output:  Now to look at it as an image. We wrote a function for you called myImage which takes a single argument, a matrix of data to display using the R function image. Run it now with a1 as its argument.
    CorrectAnswer: myImage(a1)
    AnswerTests: omnitest(correctExpr='myImage(a1)')
    Hint: Type myImage(a1) at the command prompt.

  - Class: text
    Output: It might not look like much but it's a good start. Now we'll try the same experiment but this time we'll use 2 elements from each of the 3 SVD terms.

  - Class: cmd_question
    Output:  Create the matrix a2 as the product of the first 2 columns of svd1$u, a diagonal matrix using the first 2 elements of svd1$d, and the transpose of the first 2 columns of svd1$v. Since all of your multiplicands are matrices you have to use only the operator %*% AND you DON'T need parentheses. Also, you must use the R function diag with svd1$d[1:2] as its sole argument to create the proper diagonal matrix. Remember, matrix multiplication is NOT commutative so you have to put the multiplicands in the correct order. Please use the 1:2 notation and not the c(m:n), i.e., the concatenate function, when specifying the columns.
    CorrectAnswer: a2 <- svd1$u[,1:2] %*%  diag(svd1$d[1:2])  %*% t(svd1$v[,1:2])
    AnswerTests: expr_creates_var("a2"); omnitest(correctExpr='a2 <- svd1$u[,1:2] %*% diag(svd1$d[1:2])  %*% t(svd1$v[,1:2])')
    Hint: Type a2 <- svd1$u[,1:2] %*%  diag(svd1$d[1:2])  %*% t(svd1$v[,1:2]) at the command prompt.

  - Class: cmd_question
    Output:  Use myImage again to see how a2 displays.
    CorrectAnswer: myImage(a2)
    AnswerTests: omnitest(correctExpr='myImage(a2)')
    Hint: Type myImage(a2) at the command prompt.

  - Class: cmd_question
```

```
  Output: We're starting to see slightly more detail, and maybe if you squint you see a grimacing mouth. Now let's see what image results using 5 components. From our plot of the variance explained 5 components covered a sizeable percentage of the variation. To save typing, use the up arrow to recall the command which created a2 and replace the a2 and assignment arrow with the call to myImage, and change the three occurrences of 2 to 5.
  CorrectAnswer: myImage(svd1$u[,1:5] %*%  diag(svd1$d[1:5])  %*% t(svd1$v[,1:5]))
  AnswerTests:  omnitest(correctExpr='myImage(svd1$u[,1:5] %*%  diag(svd1$d[1:5])  %*% t(svd1$v[,1:5]))')
  Hint: Type myImage(svd1$u[,1:5] %*%  diag(svd1$d[1:5])  %*% t(svd1$v[,1:5])) at the command prompt.

- Class: cmd_question
  Output: Certainly much better. Clearly a face is appearing with eyes, nose, ears, and mouth recognizable. Again, use the up arrow to recall the last command  (calling myImage with a matrix product argument) and change the 5's to 10's. We'll see how this image looks.
  CorrectAnswer: myImage(svd1$u[,1:10] %*%  diag(svd1$d[1:10])  %*% t(svd1$v[,1:10]))
  AnswerTests:  omnitest(correctExpr='myImage(svd1$u[,1:10] %*%  diag(svd1$d[1:10])  %*% t(svd1$v[,1:10]))')
  Hint: Type myImage(svd1$u[,1:10] %*%  diag(svd1$d[1:10])  %*% t(svd1$v[,1:10])) at the command prompt.

- Class: text
  Output: Now that's pretty close to the original which was low resolution to begin with, but you can see that 10 components really do capture the essence of the image. Singular value decomposition is a good way to approximate data without having to store a lot.

- Class: text
  Output: We'll close now with a few comments. First, when reducing dimensions you have to pay attention to the scales on which different variables are measured and make sure that all your data is in consistent units. In other words, scales of your data matter. Second, principal components and singular values may mix real patterns, as we saw in our simple 2-pattern example, so finding and separating out the real patterns require some detective work. Let's do a quick review now.

- Class: mult_question
  Output: Which of the following cliches LEAST captures the essence of dimension reduction?
  AnswerChoices:  find the needle in the haystack; see the forest through the trees; separate the wheat from the chaff; a face that could launch a 1000 ships
  CorrectAnswer: a face that could launch a 1000 ships
  AnswerTests: omnitest(correctVal='a face that could launch a 1000 ships')
  Hint: Which choice fails to deal with discerning differences between the valuable and the invaluable.

- Class: mult_question
  Output: A matrix X has the singular value decomposition UDV^t. The principal components of X are ?
  AnswerChoices:  the columns of U; the rows of U; the columns of V; the rows of V
  CorrectAnswer: the columns of V
  AnswerTests: omnitest(correctVal='the columns of V')
  Hint: Recall the simple example where we ran prcomp and svd on the same scaled matrix and saw that the columns of V matched the rotations of the prcomp output.

- Class: mult_question
  Output: A matrix X has the singular value decomposition UDV^t. The singular values of X are found where?
  AnswerChoices:  the columns of U; the columns of D; the columns of V; the diagonal elements of D;
  CorrectAnswer: the diagonal elements of D
  AnswerTests: omnitest(correctVal='the diagonal elements of D')
  Hint: Recall that U and V give us vectors and D gave us values.

- Class: mult_question
  Output: True or False? PCA and SVD are totally unrelated.
  AnswerChoices:  False; True
  CorrectAnswer: False
```

```
  AnswerTests: omnitest(correctVal='False')
  Hint: Recall the question about principal components and their relationship to V.

- Class: mult_question
  Output: True or False? D gives the singular values of a matrix in decreasing order of weight.
  AnswerChoices:  True; False
  CorrectAnswer: True
  AnswerTests: omnitest(correctVal='True')
  Hint: Recall that the first value accounted for the highest percentage of variation in the data.


- Class: text
  Output: Congratulations! We hope you enjoyed making faces and that this lesson didn't reduce the dimensions of your understanding.

- 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: ""
```