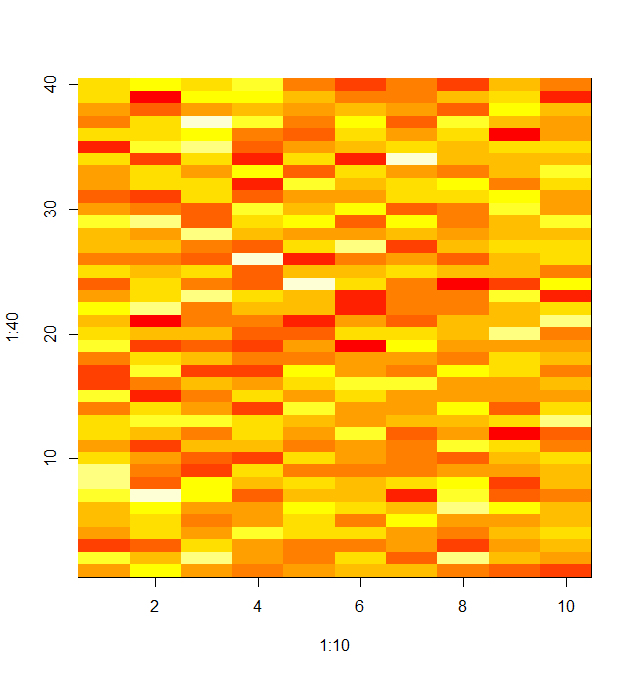
|  |
| --- |
| | Please choose a lesson, or type 0 to return to course menu.  1: Principles of Analytic Graphs 2: Exploratory Graphs 3: Graphics Devices in R  4: Plotting Systems 5: Base Plotting System 6: Lattice Plotting System  7: Working with Colors 8: GGPlot2 Part1 9: GGPlot2 Part2  10: GGPlot2 Extras 11: Hierarchical Clustering 12: K Means Clustering  13: Dimension Reduction 14: Clustering Example 15: CaseStudy  Selection: 13  | Attemping to load lesson dependencies...  | Package ‘fields’ loaded correctly!  | Package ‘jpeg’ loaded correctly!  | Package ‘datasets’ loaded correctly!  | | 0%  | 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.)  ...  |= | 1%  | 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.  ...  |=== | 2%  | We'll begin with a motivating example - random data. |
|  |
| |  | | --- | | ... | |



|==== | 4%

| 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.

> head(dataMatrix)

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]

[1,] 0.5855288 1.1285108 0.6453831 1.5448636 -0.4876385 -1.4361457 -0.7000758 -1.5138641 0.3803157 -0.37582344

[2,] 0.7094660 -2.3803581 1.0431436 1.3214520 0.3031512 -0.6292596 -0.5674016 0.1642810 0.6051368 -1.81283376

[3,] -0.1093033 -1.0602656 -0.3043691 0.3221516 -0.2419740 0.2435218 -0.2613939 -0.8708652 1.0196741 0.28860021

[4,] -0.4534972 0.9371405 2.4771109 1.5309551 -0.4817336 1.0583622 -1.0638850 1.5933290 0.4749430 -0.18962258

[5,] 0.6058875 0.8544517 0.9712207 -0.4212397 -0.9918029 0.8313488 -0.1063687 0.6465975 -2.1859464 0.01786021

[6,] -1.8179560 1.4607294 1.8670992 -1.1588210 -0.2806491 0.1052118 0.7711037 0.3573697 0.9331922 0.65043024

| You are really on a roll!

|====== | 5%

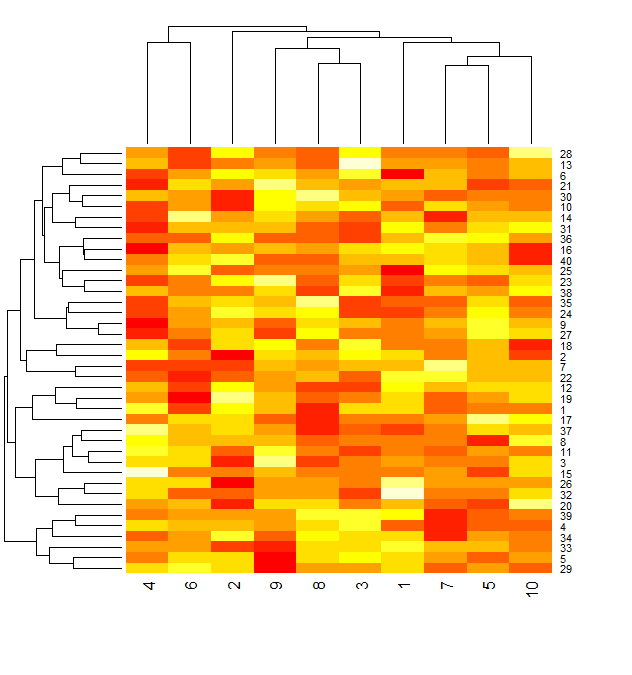
| So we see that dataMatrix has 10 columns (and hence 40 rows) of random numbers. The image here looks pretty random. Let's

| see how the data clusters. Run the R command heatmap with dataMatrix as its only argument.

> heatmap(dataMatrix)

| That's a job well done!

|======= | 6%



| We can see that even with the clustering that heatmap provides, permuting the rows (observations) and columns (variables)

| independently, the data still looks random.

...

|======== | 7%

| 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.

> myedit("addPatt.R")

| Great job!

|========== | 8%

“addPatt.R”

set.seed(678910)

for(i in 1:40){

# flip a coin

coinFlip <- rbinom(1,size=1,prob=0.5)

# if coin is heads add a common pattern to that row

if(coinFlip){

dataMatrix[i,] <- dataMatrix[i,] + rep(c(0,3),each=5)

}

}

|  |
| --- |
| | Look at the code. Will every row of the matrix have a pattern added to it?  1: Yes  2: No  Selection: 2  | You nailed it! Good job!  |=========== | 10%  | 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?  1: Yes  2: No  Selection: 2  | You're the best!  |============= | 11%  | 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. |
|  |
| |  | | --- | | ... | |

|============== | 12%

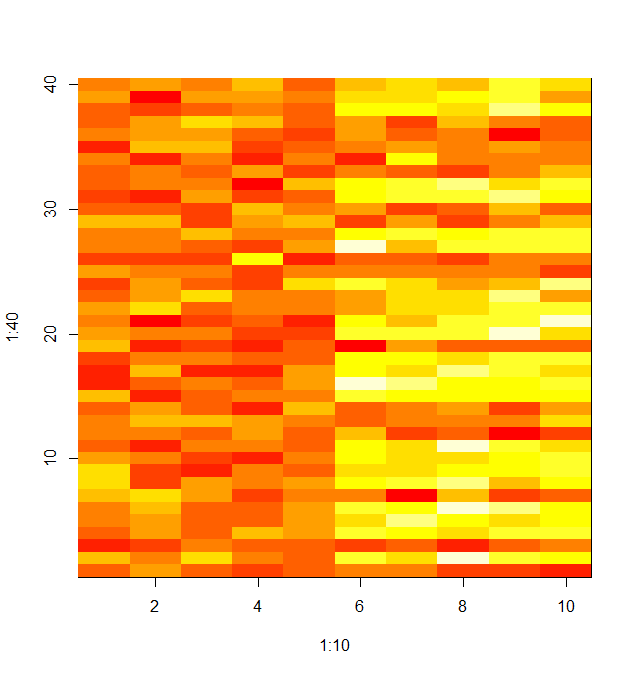
| Now to execute this code, run the R command source with 2 arguments. The first is the filename (in quotes), "addPatt.R",

| and the second is the argument local set equal to TRUE.

> source("addPatt.R", local = TRUE)

| You're the best!

|=============== | 13%



| 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.

...

|================= | 14%

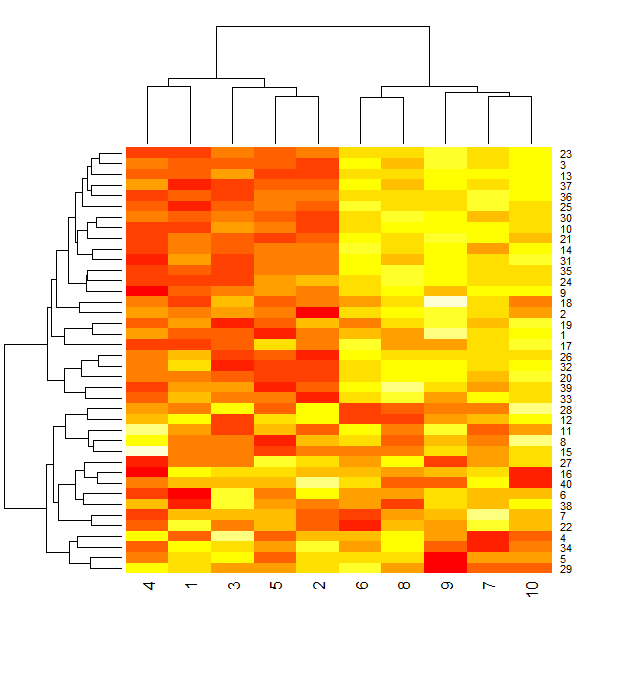
| Now run the R command heatmap again with dataMatrix as its only argument. This will perform a hierarchical cluster analysis

| on the matrix.

> heatmap(dataMatrix)

| Excellent job!

|================== | 16%



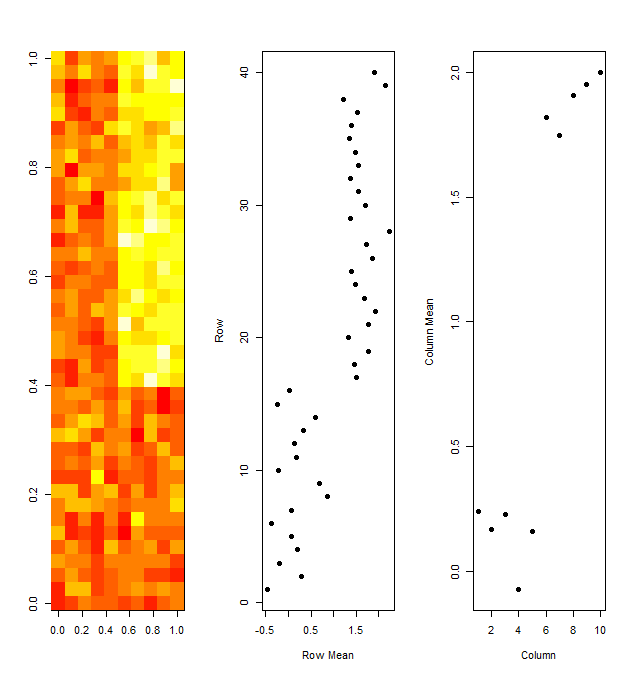
| Again we see the pattern in the columns of the matrix. As shown in the dendrogram at the top of the display, these split

| 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.

...

|==================== | 17%



| 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).

...

|===================== | 18%

| 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.

...

|====================== | 19%

| 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.

...

|======================== | 20%

| 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.

...

|========================= | 22%

| Which of the following would be an example of variables correlated to one another?

1: Heights and weights of members of a family

2: The depth of the Atlantic Ocean and what you eat for breakfast

3: Today's weather and a butterfly's wing position

Selection: 1

| You are amazing!

|=========================== | 23%

| 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.

...

|============================ | 24%

| 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.

...

|============================= | 25%

| 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.

...

|=============================== | 27%

| 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.

...

|================================ | 28%

| To illustrate this idea we created a simple example matrix called mat. Look at it now.

> mat

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 2 5 7

| You are really on a roll!

|================================== | 29%

| 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.

> svd(mat)

$d

[1] 9.5899624 0.1806108

$u

[,1] [,2]

[1,] -0.3897782 -0.9209087

[2,] -0.9209087 0.3897782

$v

[,1] [,2]

[1,] -0.2327012 -0.7826345

[2,] -0.5614308 0.5928424

[3,] -0.7941320 -0.1897921

| You got it!

|=================================== | 30%

| 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 %\*%.

> matu%\*%diag%\*%t(matv)

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 2 5 7

| Great job!

|==================================== | 31%

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

...

|====================================== | 33%

| 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.

...

|======================================= | 34%

| 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.

...

|========================================= | 35%

| We'll demonstrate this now. First we have to scale mat, our simple example 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.

> svd(scale(mat))

$d

[1] 1.732051 0.000000

$u

[,1] [,2]

[1,] -0.7071068 0.7071068

[2,] 0.7071068 0.7071068

$v

[,1] [,2]

[1,] 0.5773503 -0.5773503

[2,] 0.5773503 0.7886751

[3,] 0.5773503 -0.2113249

| You're the best!

|========================================== | 36%

| Now run the R program prcomp on scale(mat). This will give you the principal components of mat. See if they look familiar.

> prcomp(scale(mat))

Standard deviations:

[1] 1.732051 0.000000

Rotation:

PC1 PC2

[1,] 0.5773503 -0.5773503

[2,] 0.5773503 0.7886751

[3,] 0.5773503 -0.2113249

| You are quite good my friend!

|=========================================== | 37%

| 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.

...

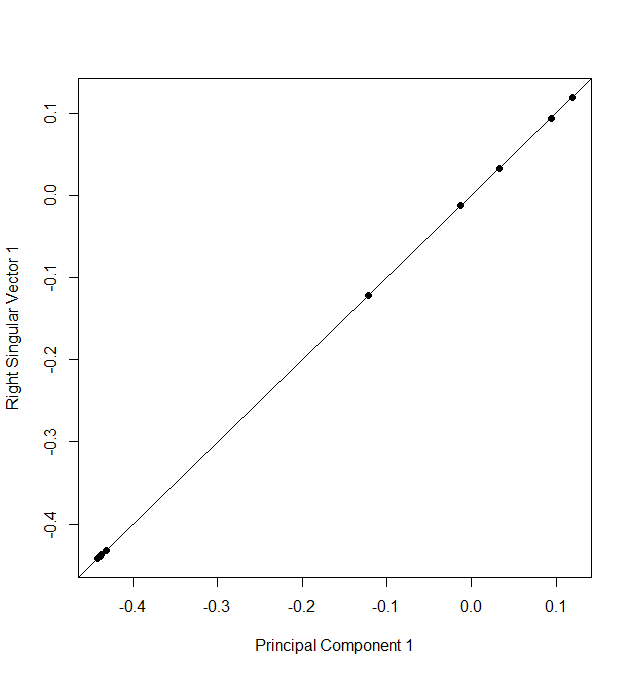
|============================================= | 39%

| 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.

...

|============================================== | 40%



| 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.

...

|================================================ | 41%

| 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.

> svd1$v[,1]

[1] -0.01269600 0.11959541 0.03336723 0.09405542 -0.12201820 -0.43175437 -0.44120227 -0.43732624 -0.44207248 -0.43924243

| You are quite good my friend!

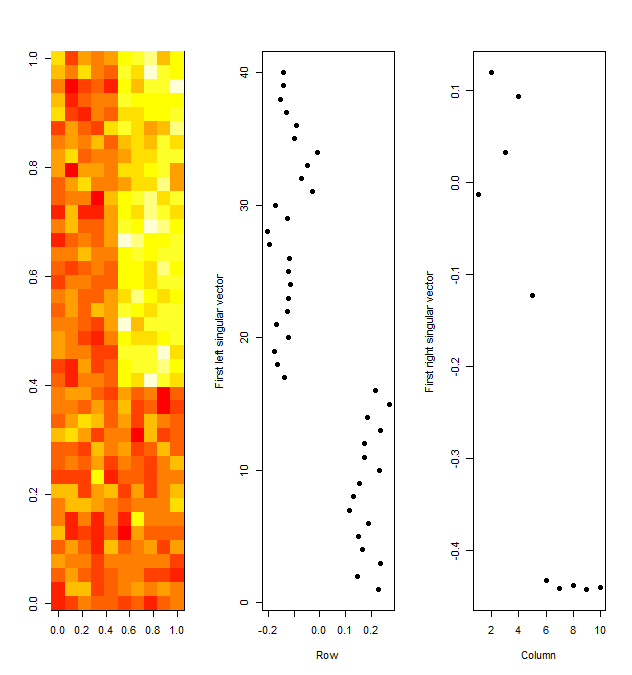
|================================================= | 42%

| 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)).

...

|================================================== | 43%



| 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.

...

|==================================================== | 45%

| 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.

...

|===================================================== | 46%

| So the singular value decomposition automatically picked up these patterns, the differences in the row and column means.

...

|======================================================= | 47%

| 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

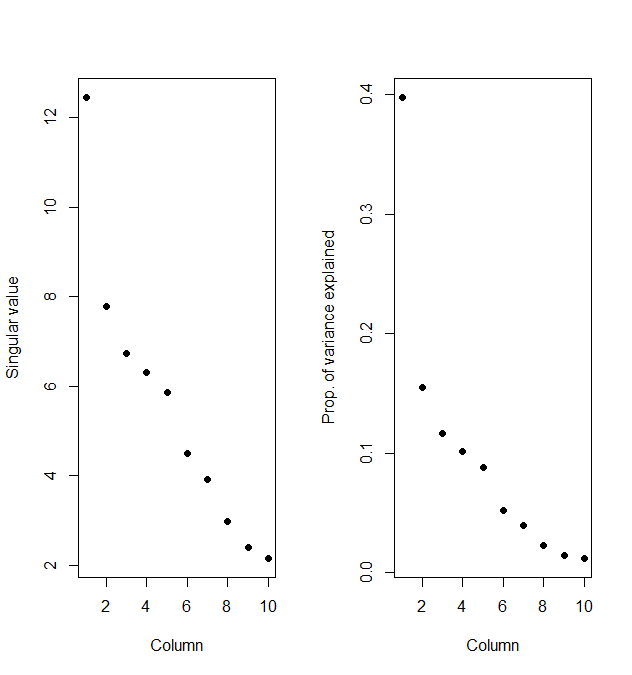
| diagonal entries now. Recall that they're stored in svd1$d.

> svd1$d

[1] 12.458121 7.779798 6.732595 6.301878 5.860013 4.501826 3.921267 2.973909 2.401470 2.152848

| Excellent work!

|======================================================== | 48%



| 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.

...

|========================================================= | 49%

| 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.

> head(constantMatrix)

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]

[1,] 0 0 0 0 0 1 1 1 1 1

[2,] 0 0 0 0 0 1 1 1 1 1

[3,] 0 0 0 0 0 1 1 1 1 1

[4,] 0 0 0 0 0 1 1 1 1 1

[5,] 0 0 0 0 0 1 1 1 1 1

[6,] 0 0 0 0 0 1 1 1 1 1

| That's correct!

|=========================================================== | 51%

| 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.

> svd2$d

[1] 1.414214e+01 1.293147e-15 2.515225e-16 8.585184e-31 9.549693e-32 3.330034e-32 2.022600e-46 4.362170e-47 1.531252e-61

[10] 0.000000e+00

| You are amazing!

|============================================================ | 52%

| Which index holds the largest entry of the svd2$d?

1: 10

2: 9

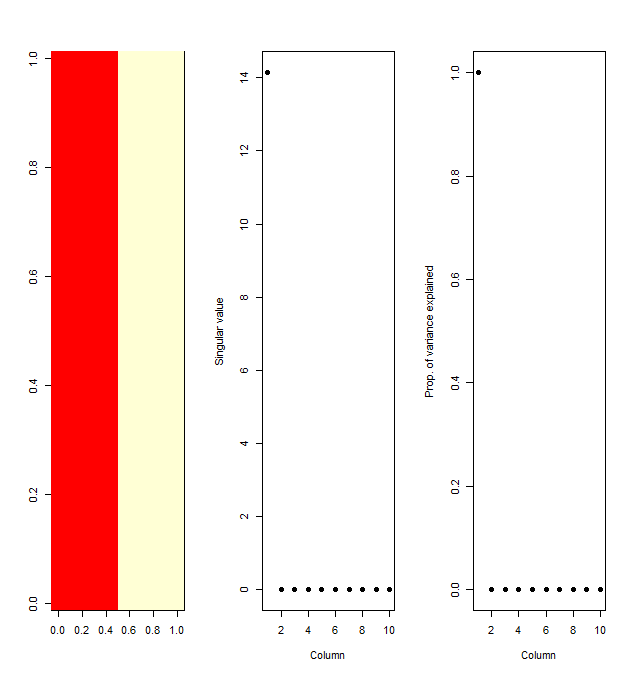
3: 5

4: 1

Selection: 4

| That's correct!

|============================================================= | 53%



| 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.

...

|=============================================================== | 54%

| According to the plot, what percentage of the total variation does the first diagonal element account for?

1: 0%

2: 100%

3: 90%

4: 50%

Selection: 2

| That's correct!

|================================================================ | 55%

| So what does this mean? Basically that the data is one-dimensional. Only 1 piece of information, namely which column an

| entry is in, determines its value.

...

|================================================================== | 57%

| Now let's return to our random 40 by 10 dataMatrix and consider a slightly more complicated example in which we add 2

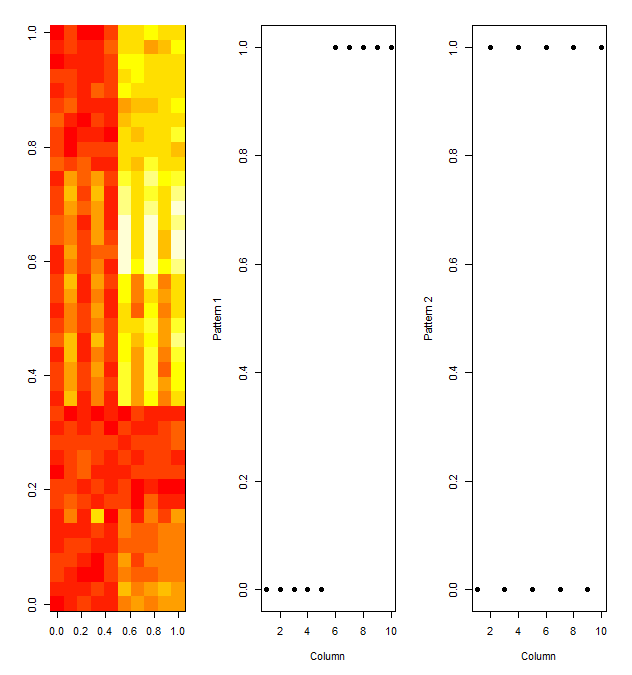
| patterns to it. Again we'll choose which rows to tweak using coinflips. Specifically, for each of the 40 rows we'll flip 2

| 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.

...

|=================================================================== | 58%



| 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.

...

|==================================================================== | 59%

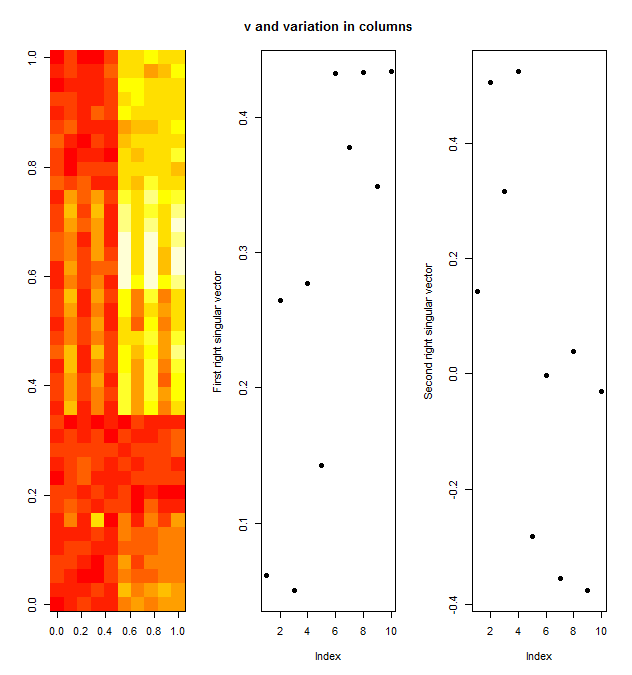
| The question is, "Can our analysis detect these patterns just from the data?" Let's see what SVD shows. Since we're

| 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.

...

|====================================================================== | 60%



| 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.

...

|======================================================================= | 61%

| 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.

|  |
| --- |
| > svd2$v[,1:2]  [,1] [,2]  [1,] 0.06154540 0.142468636  [2,] 0.26433096 0.504510087  [3,] 0.04987554 0.316470664  [4,] 0.27693897 0.524499356  [5,] 0.14275820 -0.282921362  [6,] 0.43252652 -0.002280468  [7,] 0.37724057 -0.354403893  [8,] 0.43280767 0.039226153  [9,] 0.34912246 -0.376485206  [10,] 0.43379723 -0.031422705  | Great job!  |========================================================================= | 63%  | 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. |
|  |
| |  | | --- | | ... | |

|========================================================================== | 64%

| Now look at the entries of the diagonal matrix d resulting from the svd. Recall that we stored this output for you in the

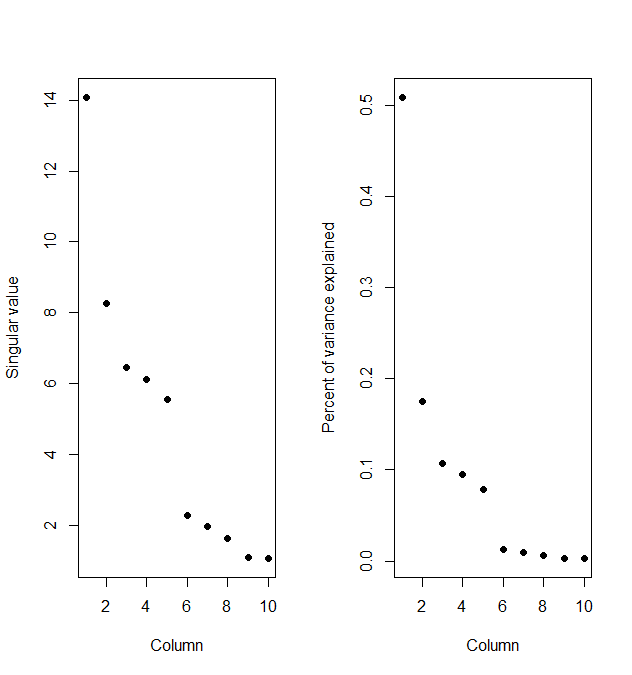
| svd object svd2.

> svd2$d

[1] 14.084918 8.257842 6.458184 6.100500 5.538262 2.266358 1.955790 1.609508 1.078566 1.054146

| You got it right!

|=========================================================================== | 65%



| 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.

...

|============================================================================= | 66%

| According to the plot, how much of the variance does the second element account for?

1: 18%

2: 11%

3: 53%

4: .1%

Selection: 1

| That's a job well done!

|============================================================================== | 67%

| 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.

...

|================================================================================ | 69%

| 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.

...

|================================================================================= | 70%

| 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.

...

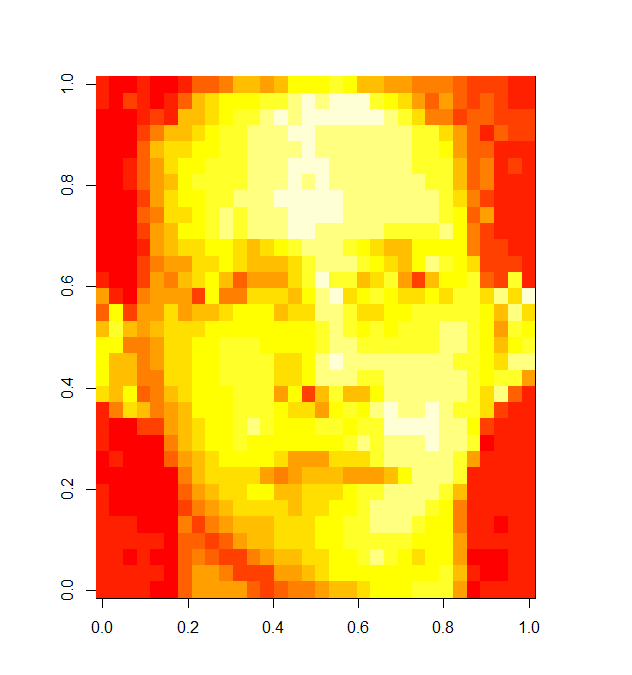
|================================================================================== | 71%

| 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.

...

|==================================================================================== | 72%



| 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.

...

|===================================================================================== | 73%

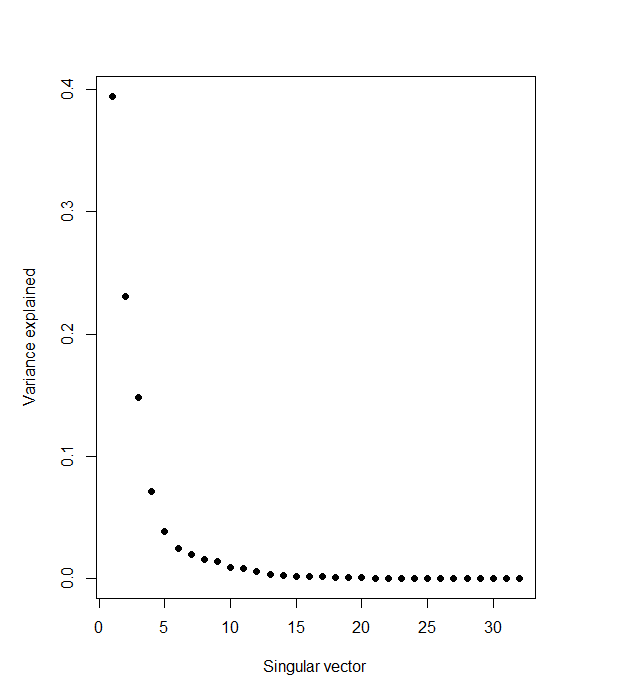
| The image data is stored in the matrix faceData. Run the R command dim on faceData to see how big it is.

> dim(faceData)

[1] 32 32

| You got it right!

|======================================================================================= | 75%



| 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.

...

|======================================================================================== | 76%

| According to the plot what percentage of the variance is explained by the first singular value?

1: 15

2: 100

3: 40

4: 23

Selection: 3

| Excellent work!

|========================================================================================= | 77%

| 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?

...

|=========================================================================================== | 78%

| 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.

...

|============================================================================================ | 80%

| 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.)

...

|============================================================================================== | 81%

| 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 \*.

...

|=============================================================================================== | 82%

| 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.

> a1 <- (svd1$u[,1] %\*% svd1$d[1])

| 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.

> a1 <- svd1$u[,1] %\*% t(svd1$v[,1]) \* svd1$d[1]

| You are amazing!

|================================================================================================ | 83%

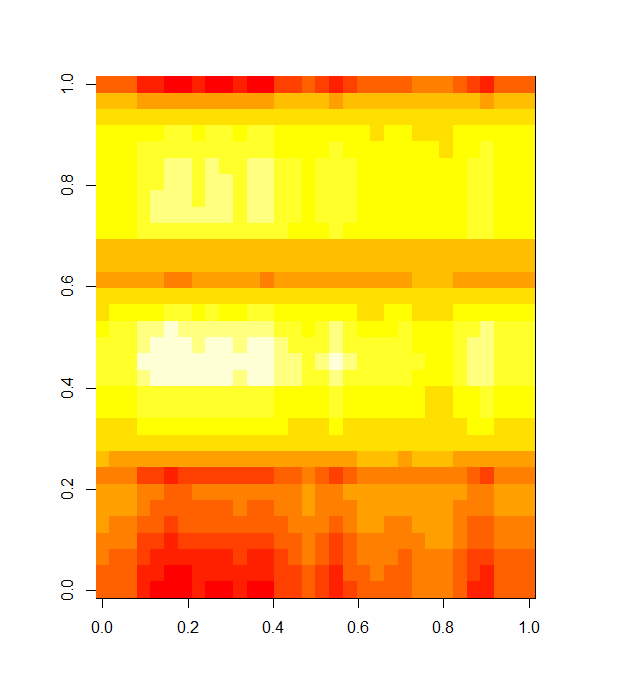
| Now to look at it as an image. We wrote a function for you called myImage which takes a single argument, a matrix of data

| to display using the R function image. Run it now with a1 as its argument.

> myImage(a1)

| Excellent job!

|================================================================================================== | 84%



| 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.

...

|=================================================================================================== | 86%

| 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.

> a2 <- svd1$u[,1:2] %\*% diag(svd1$d[1:2]) %\*% t(svd1$v[,1:2])

| All that practice is paying off!

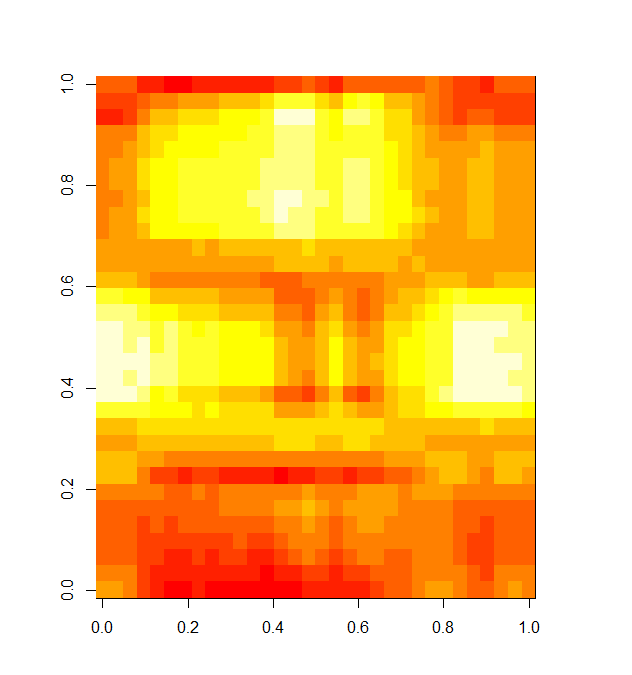
|===================================================================================================== | 87%

| Use myImage again to see how a2 displays.

> myImage(a2)

| That's a job well done!

|====================================================================================================== | 88%



| 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

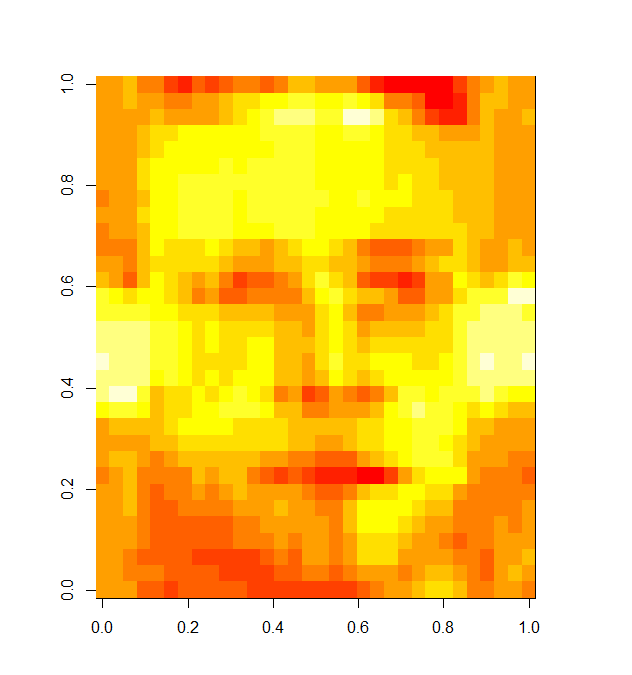
| variation. To save typing, use the up arrow to recall the command which created a2 and replace the a2 and assignment arrow

| with the call to myImage, and change the three occurrences of 2 to 5.

> myImage(svd1$u[,1:5] %\*% diag(svd1$d[1:5]) %\*% t(svd1$v[,1:5]))

| All that hard work is paying off!

|======================================================================================================= | 89%



| Certainly much better. Clearly a face is appearing with eyes, nose, ears, and mouth recognizable. Again, use the up arrow

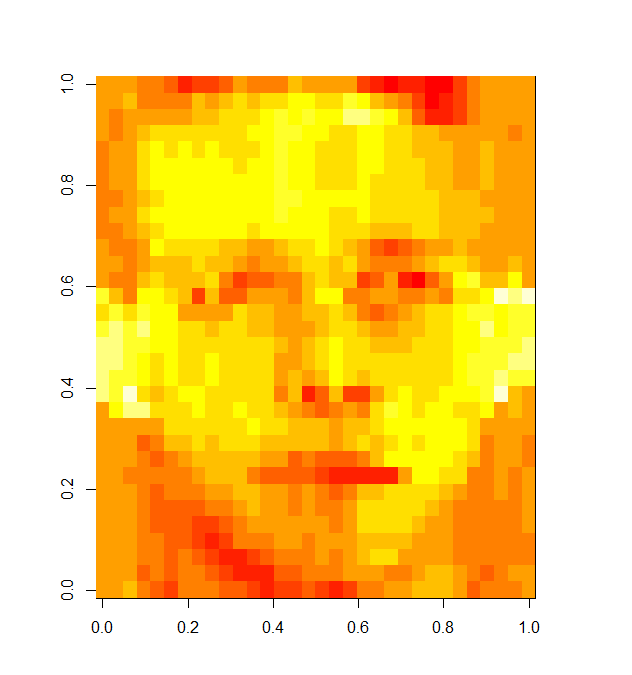
| to recall the last command (calling myImage with a matrix product argument) and change the 5's to 10's. We'll see how this

| image looks.

> myImage(svd1$u[,1:10] %\*% diag(svd1$d[1:10]) %\*% t(svd1$v[,1:10]))

| Great job!

|========================================================================================================= | 90%



| 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.

...

|========================================================================================================== | 92%

| 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.

...

|============================================================================================================ | 93%

| Which of the following cliches LEAST captures the essence of dimension reduction?

1: separate the wheat from the chaff

2: find the needle in the haystack

3: see the forest through the trees

4: a face that could launch a 1000 ships

Selection: 4

| Keep working like that and you'll get there!

|============================================================================================================= | 94%

| A matrix X has the singular value decomposition UDV^t. The principal components of X are ?

1: the columns of V

2: the rows of V

3: the columns of U

4: the rows of U

Selection: 1

| Excellent job!

|============================================================================================================== | 95%

| A matrix X has the singular value decomposition UDV^t. The singular values of X are found where?

1: the columns of D

2: the columns of V

3: the diagonal elements of D

4: the columns of U

Selection: 4

| Give it another try.

| Recall that U and V give us vectors and D gave us values.

1: the diagonal elements of D

2: the columns of V

3: the columns of U

4: the columns of D

Selection: 1

| You are amazing!

|================================================================================================================ | 96%

| True or False? PCA and SVD are totally unrelated.

1: False

2: True

Selection: 1

| That's the answer I was looking for.

|================================================================================================================= | 98%

| True or False? D gives the singular values of a matrix in decreasing order of weight.

1: False

2: True

Selection: 2

| Nice work!

|=================================================================================================================== | 99%

| Congratulations! We hope you enjoyed making faces and that this lesson didn't reduce the dimensions of your understanding.

...

|====================================================================================================================| 100%