```
Course: Exploratory_Data_Analysis
Lesson: K_Means_Clustering

- Class: text
  Output: "K_Means_Clustering. (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/kmeansClustering.)"


- Class: text
  Output:  In this lesson we'll learn about k-means clustering, another simple way of examining and organizing multi-dimensional data. As with hierarchical clustering, this technique is  most useful in the early stages of analysis when you're trying to get an understanding of the data, e.g., finding some pattern or relationship between different factors or variables.

- Class: text
  Output:  R documentation tells us that the k-means method "aims to partition the points into k groups such that the sum of squares from points to the assigned cluster centres is minimized."

- Class: text
  Output: Since clustering organizes data points that are close into groups we'll assume we've decided on a measure of distance, e.g., Euclidean.

- Class: figure
  Output: To illustrate the method, we'll use these random points we generated, familiar to you if you've already gone through the hierarchical clustering lesson. We'll  demonstrate k-means clustering in several steps, but first we'll explain the general idea.
  Figure: ranPoints.R
  FigureType: new

- Class: text
  Output: As we said, k-means is a partioning approach which requires that you first guess how many clusters you have (or want). Once you fix this number, you randomly create a "centroid" (a phantom point) for each cluster and assign each point or observation in your dataset to the centroid to which it is closest. Once each point is assigned a centroid, you readjust the centroid's position by making it the average of the points assigned to it.

- Class: text
  Output: Once you have repositioned the centroids, you must recalculate the distance of the observations to the centroids and reassign any, if necessary, to the centroid closest to them. Again, once the reassignments are done, readjust the positions of the centroids based on the new cluster membership. The process stops once you reach an iteration in which no adjustments are made or when you've reached some predetermined maximum number of iterations.

- Class: mult_question
  Output: As described, what does this process require?
  AnswerChoices:  A defined distance metric; A number of clusters; An initial guess as to cluster centroids; All of the others
  CorrectAnswer:  All of the others
  AnswerTests: omnitest(correctVal='All of the others')
  Hint: Which choice includes all the others.

- Class: mult_question
  Output: So k-means clustering requires some distance metric (say Euclidean), a hypothesized fixed number of clusters, and an initial guess as to cluster centroids. As described, what does this process produce?
  AnswerChoices:  A final estimate of cluster centroids; An assignment of each point to a cluster; All of the others
  CorrectAnswer:  All of the others
  AnswerTests: omnitest(correctVal='All of the others')
  Hint: Which choice includes all the others.

- Class: text
```

```yaml
  Output: When it's finished k-means clustering returns a final position of each cluster's centroid as well as the assignment of each data point or observation to a cluster.

- Class: text
  Output: Now we'll step through this process using our random points as our data. The coordinates of these are stored in 2 vectors, x and y. We eyeball the display and guess that there are 3 clusters. We'll pick 3 positions of centroids, one for each cluster.

- Class: cmd_question
  Output:  We've created two 3-long vectors for you, cx and cy. These respectively hold the x- and y- coordinates for 3 proposed centroids. For convenience, we've also stored them in a 2 by 3 matrix cmat. The x coordinates are in the first row and the y coordinates in the second. Look at cmat now.
  CorrectAnswer: cmat
  AnswerTests: omnitest(correctExpr='cmat')
  Hint: Type cmat at the command prompt.

- Class: cmd_question
  Output:  The coordinates of these points are (1,2), (1.8,1) and (2.5,1.5). We'll add these centroids to the plot of our points. Do this by calling the R command points with 6 arguments. The first 2 are cx and cy, and the third is col set equal to the concatenation of 3 colors, "red", "orange", and "purple". The fourth argument is pch set equal to 3 (a plus sign), the fifth is cex set equal to 2 (expansion of character), and the final is lwd (line width) also set equal to 2.
  CorrectAnswer: points(cx,cy,col=c("red","orange","purple"),pch=3,cex=2,lwd=2)
  AnswerTests: omnitest(correctExpr='points(cx,cy,col=c("red","orange","purple"),pch=3,cex=2,lwd=2)')
  Hint: Type points(cx,cy,col=c("red","orange","purple"),pch=3,cex=2,lwd=2) at the command prompt.

- Class: text
  Output: We see the first centroid (1,2) is in red. The second (1.8,1), to the right and below the first, is orange, and the final centroid (2.5,1.5), the furthest to the right, is purple.

- Class: mult_question
  Output: Now we have to calculate distances between each point and every centroid. There are 12 data points and 3 centroids. How many distances do we have to calculate?
  AnswerChoices:  15; 36; 9; 108
  CorrectAnswer: 36
  AnswerTests: omnitest(correctVal='36')
  Hint: The distance between each point and one centroid means 12 distances have to be calculated for each centroid. This has to be done for all 3 centroids.

- Class: cmd_question
  Output:  We've written a function for you called mdist which takes 4 arguments. The vectors of data points (x and y) are the first two and the two vectors of centroid coordinates (cx and cy) are the last two. Call mdist now with these arguments.
  CorrectAnswer: mdist(x,y,cx,cy)
  AnswerTests: omnitest(correctExpr='mdist(x,y,cx,cy)')
  Hint: Type mdist(x,y,cx,cy) at the command prompt.


- Class: mult_question
  Output: We've stored these distances in the matrix distTmp for you. Now we have to assign a cluster to each point. To do that we'll look at each column and ?
  AnswerChoices:  pick the minimum entry; pick the maximum entry; add up the 3 entries.
  CorrectAnswer:  pick the minimum entry
  AnswerTests: omnitest(correctVal='pick the minimum entry')
  Hint: We assign each point to the centroid closest to it. Recall that the matrix holds distances.

- Class: mult_question
  Output: From the distTmp entries, which cluster would point 6 be assigned to?
  AnswerChoices:  1; 2; 3; none of the above
  CorrectAnswer:  3
  AnswerTests: omnitest(correctVal='3')
```

```
    Hint: Which row in column 6 has the lowest value?


  - Class: cmd_question
    Output:  R has a handy function which.min which you can apply to ALL the columns of distTmp with one call. Simply call the R function apply with 3 arguments. The first is distTmp, the second is 2 meaning the columns of distTmp, and the third is which.min, the function you want to apply to the columns of distTmp. Try this now.
    CorrectAnswer: apply(distTmp,2,which.min)
    AnswerTests: omnitest(correctExpr='apply(distTmp,2,which.min)')
    Hint: Type apply(distTmp,2,which.min) at the command prompt.

  - Class: text
    Output: You can see that you were right and the 6th entry is indeed 3 as you answered before. We see the first 3 entries were assigned to the second (orange) cluster and only 2 points (4 and 8) were assigned to the first (red) cluster.

  - Class: cmd_question
    Output:  We've stored the vector of cluster colors ("red","orange","purple") in the array cols1 for you and we've also stored the cluster assignments in the array newClust. Let's color the 12 data points  according to their assignments. Again, use the command points with 5 arguments. The first 2 are x and y. The third is pch set to 19, the fourth is cex set to 2, and the last, col is set to cols1[newClust].
    CorrectAnswer: points(x,y,pch=19,cex=2,col=cols1[newClust])
    AnswerTests: omnitest(correctExpr='points(x,y,pch=19,cex=2,col=cols1[newClust])')
    Hint: Type points(x,y,pch=19,cex=2,col=cols1[newClust]) at the command prompt.

  - Class: text
    Output: Now we have to recalculate our centroids so they are the average (center of gravity) of the cluster of points assigned to them. We have to do the x and y coordinates separately. We'll do the x coordinate first. Recall that the vectors x and y hold the respective coordinates of our 12 data points.

  - Class: cmd_question
    Output: We can use the R function tapply which applies "a function over a ragged array". This means that every element of the array is assigned a factor and the function is applied to  subsets of the array (identified by the factor vector). This allows us to take advantage of the factor vector newClust we calculated. Call tapply now with 3 arguments, x (the data), newClust (the factor array), and mean (the function to apply).
    CorrectAnswer: tapply(x,newClust,mean)
    AnswerTests: omnitest(correctExpr='tapply(x,newClust,mean)')
    Hint: Type tapply(x,newClust,mean) at the command prompt.

  - Class: cmd_question
    Output: Repeat the call, except now apply it to the vector y instead of x.
    CorrectAnswer: tapply(y,newClust,mean)
    AnswerTests: omnitest(correctExpr='tapply(y,newClust,mean)')
    Hint: Type tapply(y,newClust,mean) at the command prompt.


  - Class: cmd_question
    Output: Now that we have  new x and new y coordinates for the 3 centroids we can plot them. We've stored off the coordinates for you in variables newCx and newCy. Use the R command points with these as the first 2 arguments. In addition, use the arguments col set equal to cols1, pch equal to 8, cex equal to 2 and lwd also equal to 2.
    CorrectAnswer: points(newCx,newCy,col=cols1,pch=8,cex=2,lwd=2)
    AnswerTests: omnitest(correctExpr='points(newCx,newCy,col=cols1,pch=8,cex=2,lwd=2)')
    Hint: Type points(newCx,newCy,col=cols1,pch=8,cex=2,lwd=2) at the command prompt.

  - Class: cmd_question
    Output: We see how the centroids have moved closer to their respective clusters. This is especially true of the second (orange) cluster. Now call the distance function mdist with the 4 arguments x, y, newCx, and newCy. This will allow us to reassign the data points to new clusters if necessary.
    CorrectAnswer: mdist(x,y,newCx,newCy)
    AnswerTests: omnitest(correctExpr='mdist(x,y,newCx,newCy)')
    Hint: Type mdist(x,y,newCx,newCy) at the command prompt.
```

```yaml
- Class: mult_question
  Output: We've stored off this new matrix of distances in the matrix distTmp2 for you. Recall that the first cluster is red, the second orange and the third purple. Look closely at columns 4 and 7 of distTmp2. What will happen to points 4 and 7?
  AnswerChoices:  Nothing; They will both change to cluster 2; They will both change clusters; They're the only points that won't change clusters
  CorrectAnswer:  They will both change clusters
  AnswerTests: omnitest(correctVal='They will both change clusters')
  Hint: Two of the choices are obviously wrong. That leaves two possibilities which are similar. Look carefully at the numbers in columns 4 and 7 to see where the minimum values are.

- Class: cmd_question
  Output: Now call apply with 3 arguments, distTmp2, 2, and which.min to find the new cluster assignments for the points.
  CorrectAnswer:  apply(distTmp2,2,which.min)
  AnswerTests: omnitest(correctExpr='apply(distTmp2,2,which.min)')
  Hint: Type  apply(distTmp2,2,which.min) at the command prompt.

- Class: cmd_question
  Output: We've stored off the new cluster assignments in a vector of factors called newClust2. Use the R function points to recolor the points with their new assignments. Again, there are 5 arguments, x and y are first, followed by pch set to 19, cex to 2, and col to cols1[newClust2].
  CorrectAnswer:  points(x,y,pch=19,cex=2,col=cols1[newClust2])
  AnswerTests: omnitest(correctExpr='points(x,y,pch=19,cex=2,col=cols1[newClust2])')
  Hint: Type  points(x,y,pch=19,cex=2,col=cols1[newClust2]) at the command prompt.

- Class: text
  Output: Notice that points 4 and 7 both changed clusters, 4 moved from 1 to 2 (red to orange), and point 7 switched from 3 to 2 (purple to red).

- Class: cmd_question
  Output: Now use tapply to find the x coordinate of the new centroid. Recall there are 3 arguments, x, newClust2, and mean.
  CorrectAnswer:  tapply(x,newClust2,mean)
  AnswerTests: omnitest(correctExpr='tapply(x,newClust2,mean)')
  Hint: Type  tapply(x,newClust2,mean) at the command prompt.

- Class: cmd_question
  Output: Do the same to find the new y coordinate.
  CorrectAnswer:  tapply(y,newClust2,mean)
  AnswerTests: omnitest(correctExpr='tapply(y,newClust2,mean)')
  Hint: Type  tapply(y,newClust2,mean) at the command prompt.

- Class: cmd_question
  Output: We've stored off these coordinates for you in the variables finalCx and finalCy. Plot these new centroids using the points function with 6 arguments. The first 2 are finalCx and finalCy. The argument col should equal cols1, pch should equal 9, cex 2 and lwd 2.
  CorrectAnswer:  points(finalCx,finalCy,col=cols1,pch=9,cex=2,lwd=2)
  AnswerTests: omnitest(correctExpr='points(finalCx,finalCy,col=cols1,pch=9,cex=2,lwd=2)')
  Hint: Type  points(finalCx,finalCy,col=cols1,pch=9,cex=2,lwd=2) at the command prompt.

- Class: text
  Output: It should be obvious that if we continued this process points 5 through 8 would all turn red, while points 1 through 4 stay orange, and points 9 through 12 purple.

- Class: text
  Output: Now that you've gone through an example step by step, you'll be relieved to hear that R provides a command to do all this work for you. Unsurprisingly it's called kmeans and, although it has several parameters, we'll just mention four. These are x, (the numeric matrix of data), centers, iter.max, and nstart. The second of these (centers) can be either a number of clusters or a set of initial centroids. The third, iter.max, specifies the maximum number of iterations to go through, and nstart is the number of random starts you want to try if you specify centers as a number.
```

```yaml
- Class: cmd_question
  Output: Call kmeans now with 2 arguments, dataFrame (which holds the x and y coordinates of our 12 points) and centers set equal to 3.
  CorrectAnswer:  kmeans(dataFrame,centers=3)
  AnswerTests: omnitest(correctExpr='kmeans(dataFrame,centers=3)')
  Hint: Type  kmeans(dataFrame,centers=3) at the command prompt.

- Class: cmd_question
  Output: The program returns the information that the data clustered into 3 clusters each of size 4. It also returns the coordinates of the 3 cluster means, a vector named cluster indicating how the 12 points were partitioned into the clusters, and the sum of squares within each cluster. It also shows all the available components returned by the function. We've stored off this data for you in a kmeans object called kmObj. Look at kmObj$iter to see how many iterations the algorithm went through.
  CorrectAnswer:  kmObj$iter
  AnswerTests: omnitest(correctExpr='kmObj$iter')
  Hint: Type kmObj$iter at the command prompt.

- Class: cmd_question
  Output: Two iterations as we did before. We just want to emphasize how you can access the information available to you. Let's plot the data points color coded according to their cluster. This was stored in kmObj$cluster. Run plot with 5 arguments. The data, x and y, are the first two; the third, col is set equal to kmObj$cluster, and the last two are pch and cex. The first of these should be set to 19 and the last to 2.
  CorrectAnswer:  plot(x,y,col=kmObj$cluster,pch=19,cex=2)
  AnswerTests: omnitest(correctExpr='plot(x,y,col=kmObj$cluster,pch=19,cex=2)')
  Hint: Type plot(x,y,col=kmObj$cluster,pch=19,cex=2) at the command prompt.

- Class: cmd_question
  Output: Now add the centroids which are stored in kmObj$centers. Use the points function with 5 arguments. The first two are  kmObj$centers and col=c("black","red","green"). The last three, pch, cex, and lwd, should all equal 3.
  CorrectAnswer:  points(kmObj$centers,col=c("black","red","green"),pch=3,cex=3,lwd=3)
  AnswerTests: omnitest(correctExpr='points(kmObj$centers,col=c("black","red","green"),pch=3,cex=3,lwd=3)')
  Hint: Type points(kmObj$centers,col=c("black","red","green"),pch=3,cex=3,lwd=3) at the command prompt.


- Class: text
  Output: Now for some fun! We want to show you how the output of the kmeans function is affected by its random start (when you just ask for a number of clusters). With random starts you might want to run the function several times to get an idea of the relationships between your observations. We'll call kmeans with the same data points (stored in dataFrame), but ask for 6 clusters instead of 3.


- Class: cmd_question
  Output: We'll plot our data points several times and each time we'll just change the argument col which will show us how the R function kmeans is clustering them. So, call plot now with 5 arguments. The first 2 are x and y. The third is col set equal to the call kmeans(dataFrame,6)$cluster. The last two (pch and cex) are set to 19 and 2 respectively.
  CorrectAnswer:  plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2)
  AnswerTests: omnitest(correctExpr='plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2)')
  Hint: Type plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2) at the command prompt.

- Class: cmd_question
  Output: See how the points cluster? Now recall your last command and rerun it.
  CorrectAnswer:  plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2)
  AnswerTests: omnitest(correctExpr='plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2)')
  Hint: Type plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2) at the command prompt.

- Class: cmd_question
```

```
  Output: See how the clustering has changed? As the Teletubbies would say, "Again! Again!"
  CorrectAnswer:  plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2)
  AnswerTests: omnitest(correctExpr='plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2)')
  Hint: Type plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2) at the command prompt.

- Class: text
  Output: So the clustering changes with different starts. Perhaps 6 is too many clusters? Let's review!

- Class: mult_question
  Output: True or False? K-means clustering  requires you to specify a number of clusters before you begin.
  AnswerChoices:  True; False
  CorrectAnswer:  True
  AnswerTests: omnitest(correctVal='True')
  Hint: What did you provide when you called the R function?

- Class: mult_question
  Output:  True or False? K-means clustering  requires you to specify a number of iterations before you begin.
  AnswerChoices:  True; False
  CorrectAnswer:  False
  AnswerTests: omnitest(correctVal='False')
  Hint: What did you provide when you called the R function?

- Class: mult_question
  Output:  True or False? Every data set has a single fixed number of clusters.
  AnswerChoices:  True; False
  CorrectAnswer:  False
  AnswerTests: omnitest(correctVal='False')
  Hint: The number of clusters depends on your eye.

- Class: mult_question
  Output:  True or False? K-means clustering will always stop in 3 iterations
  AnswerChoices:  True; False
  CorrectAnswer:  False
  AnswerTests: omnitest(correctVal='False')
  Hint: The number of iterations depends on your data.


- Class: mult_question
  Output:  True or False? When starting kmeans with random centroids, you'll always end up with the same final clustering.
  AnswerChoices:  True; False
  CorrectAnswer:  False
  AnswerTests: omnitest(correctVal='False')
  Hint: Recall the last experiment we did in the lesson, rerunning the same routine.

- Class: text
  Output: Congratulations! We hope this means you found this lesson oK.

- 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: ""
```