| Por favor, selecione uma lição ou digite 0 para retornar ao menu do curso.

1: Comandos basicos do R 2: Logicos 3: Sequencias numericas 4: Vetores

5: Valores ausentes 6: Data e Hora 7: Filtrando vetores 8: Matrizes e tabelas

9: Amostras e Simulacoes 10: Funcoes 11: Manipulacao com dplyr 12: Graficos Basicos

13: K Medias 14: Graficos Analiticos

Selection: 13

| Tentando carregar as dependências da lição...

| O pacote ‘openssl’ carregou corretamente!

| O pacote ‘jsonlite’ carregou corretamente!

| O pacote ‘ggplot2’ carregou corretamente!

| Esta lição requer o pacote ‘fields’ . Você quer instalá-lo agora?

1: Sim

2: Não

Selection: 1

| Tentando instalar o pacote ‘fields’ agora...

WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropriate version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/

also installing the dependencies ‘dotCall64’, ‘gridExtra’, ‘spam’, ‘viridis’, ‘maps’

package ‘dotCall64’ successfully unpacked and MD5 sums checked

package ‘gridExtra’ successfully unpacked and MD5 sums checked

package ‘spam’ successfully unpacked and MD5 sums checked

package ‘viridis’ successfully unpacked and MD5 sums checked

package ‘maps’ successfully unpacked and MD5 sums checked

package ‘fields’ successfully unpacked and MD5 sums checked

| O pacote ‘fields’ carregou corretamente!

| Esta lição requer o pacote ‘jpeg’ . Você quer instalá-lo agora?

1: Sim

2: Não

Selection: 1

| Tentando instalar o pacote ‘jpeg’ agora...

WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropriate version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/

package ‘jpeg’ successfully unpacked and MD5 sums checked

| O pacote ‘jpeg’ carregou corretamente!

| O pacote ‘datasets’ carregou corretamente!

| | 0%

| Nesta lição, aprenderemos sobre o agrupamento em cluster k-means, outra maneira simples de examinar e

| organizar dados multidimensionais. Tal como acontece com o agrupamento hierárquico, essa técnica é mais útil

| nos estágios iniciais de análise quando você está tentando obter uma compreensão dos dados, por exemplo,

| encontrar algum padrão ou relação entre diferentes fatores ou variáveis.

...

|== | 2%

| A documentação do R nos diz que o método k-means "visa particionar os pontos em k grupos de forma que a soma

| quadrática das distâncias dos pontos aos centros do cluster designados seja minimizada."

...

|=== | 3%

| Como o processo organiza os pontos de dados que estão próximos em grupos, vamos assumir que decidimos uma

| medida de distância, por exemplo, euclidiana.

...

|===== | 5%

| Para ilustrar o método, usaremos esses pontos aleatórios que acabei de gerar. Demonstrarei o agrupamento

| k-means em várias etapas, mas primeiro explicarei a ideia geral.

...

|====== | 6%

| Como disse, k-means é uma abordagem de agrupamento que requer que você adivinhe quantos clusters você

| tem (ou deseja). Depois de corrigir esse número, você cria aleatoriamente um "centróide" (um ponto

| fantasma) para cada cluster e atribui cada ponto ou observação em seu conjunto de dados ao centróide

| ao qual ele está mais próximo. Uma vez que cada ponto é atribuído a um centróide, você reajusta a

| posição do centróide, fazendo com que seja a média dos pontos atribuídos a ele.

...

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

| Uma vez que você tenha reposicionado os centróides, você deve recalcular a distância das observações

| para os centróides e reatribuir, se necessário, ao centróide mais próximo a eles. Novamente, depois

| que as reatribuições forem concluídas, reajuste as posições dos centróides com base na nova associação

| do cluster. O processo é interrompido quando você alcança uma iteração na qual nenhum ajuste é feito

| ou quando você atinge um número máximo predeterminado de iterações.

...

|========== | 10%

| Conforme descrito, o que esse processo exige?

1: Todas as alternativas

2: Um número de clusters

3: Uma métrica de distância definida

4: Um palpite inicial quanto aos centróides de cluster

Selection: 1

| Essa é a resposta que eu esperava.

|=========== | 11%

| Portanto, o agrupamento k-means requer alguma métrica de distância (digamos, euclidiana), um número

| fixo de clusters hipotético e uma estimativa inicial dos centróides de cluster. Como descrito, o que

| esse processo produz?

1: Todas as alternativas

2: Uma atribuição de cada ponto a um cluster

3: Uma estimativa final de centróides de cluster

Selection: 1

| Na mosca! Bom trabalho!

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

| Quando terminar, o agrupamento k-means retorna uma posição final do centróide de cada cluster, bem

| como a atribuição de cada ponto de dados ou observação a um cluster.

...

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

| Agora vamos percorrer este processo usando nossos pontos aleatórios como nossos dados. As coordenadas

| destes 12 pontos foram armazenadas em 2 vetores, x e y. Nós olhamos a tela e só de olhar imaginamos

| que existem 3 clusters. Nós vamos escolher 3 posições de centróides, uma para cada cluster.

...

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

| Criei dois vetores de 3 números para você, cx e cy. Estes, respectivamente, mantêm as coordenadas x e

| y para 3 centróides propostos. Por conveniência, também os armazenei em uma matriz 2 por 3, chamada

| cmat. As coordenadas x estão na primeira linha e as coordenadas y na segunda. Veja a matriz cmat

| agora.

> cmat

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

[1,] 1 1.8 2.5

[2,] 2 1.0 1.5

| Excelente trabalho!

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

| As coordenadas desses pontos são (1, 2), (1.8, 1) e (2.5, 1.5). Vamos adicionar esses centróides ao

| plot dos nossos pontos. Faça isso chamando o comando points com 6 argumentos. Os dois primeiros são cx

| e cy e o terceiro é conjunto igual à concatenação de três cores, "red", "orange" e "purple". O quarto

| argumento é pch=3 (um sinal de mais), o quinto é cex=2 (expansão de caractere), e o final é lwd=2

| (largura da linha).

> points(cx, cy, c('red', 'orange', 'purple', pch=3, cex=2,lwd=2))

Error in plot.xy(xy.coords(x, y), type = type, ...) : invalid plot type

> points(cx, cy, c('red', 'orange', 'purple'), pch=3, cex=2,lwd=2))

Error: unexpected ')' in "points(cx, cy, c('red', 'orange', 'purple'), pch=3, cex=2,lwd=2))"

> points(cx, cy, c('red', 'orange', 'purple'), pch=3, cex=2,lwd=2)

Error in plot.xy(xy.coords(x, y), type = type, ...) : invalid plot type

> points(cx, cy)

| Isso não é exatamente o que espero. Tente novamente. Ou digite info() para mais opções.

| Digite points(cx,cy,col=c("red","orange","purple"),pch=3,cex=2,lwd=2) no console.

> points(cx,cy,col=c("red","orange","purple"),pch=3,cex=2,lwd=2)

| Excelente trabalho!

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

| Nós vemos o primeiro centróide (1, 2) em vermelho. O segundo (1.8, 1), à direita e abaixo do primeiro, e é

| laranja. O centróide final (2.5, 1.5), o mais distante à direita, é roxo.

...

|===================== | 21%

| Agora temos que calcular as distâncias entre cada ponto e cada centróide. Existem 12 pontos de dados e 3

| centróides. Quantas distâncias temos que calcular?

1: 15

2: 108

3: 9

4: 36

Selection: 4

| Excelente trabalho!

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

| Eu escrevi uma função para você chamada mdist que recebe 4 argumentos. Os vetores de pontos de dados (x e y)

| são os dois primeiros e os dois vetores de coordenadas centróides (cx e cy) são os dois últimos. Chame mdist

| agora com esses argumentos.

> mdist(x,y,cx,cy)

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

[1,] 1.392885 0.9774614 0.7000680 1.264693 1.1894610 1.2458771 0.8113513 1.026750 4.5082665 4.5255617

[2,] 1.108644 0.5544675 0.3768445 1.611202 0.8877373 0.7594611 0.7003994 2.208006 1.1825265 1.0540994

[3,] 3.461873 2.3238956 1.7413021 4.150054 0.3297843 0.2600045 0.4887610 1.337896 0.3737554 0.4614472

[,11] [,12]

[1,] 4.8113368 4.0657750

[2,] 1.2278193 1.0090944

[3,] 0.5095428 0.2567247

| Excelente!

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

| Cada linha representa um dos 3 centróides e cada coluna, um dos 12 pontos. Cada célula representa a

| distância euclidiana de um ponto para um centróide.

...

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

| Agora, faça o mesmo comando, mas armazene o resultado na variável distTmp.

> distTmp <- mdist(x,y,cx,cy)

| Perseverança é a resposta.

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

| Confira o conteúdo da variável distTmp.

> distTmp

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

[1,] 1.392885 0.9774614 0.7000680 1.264693 1.1894610 1.2458771 0.8113513 1.026750 4.5082665 4.5255617

[2,] 1.108644 0.5544675 0.3768445 1.611202 0.8877373 0.7594611 0.7003994 2.208006 1.1825265 1.0540994

[3,] 3.461873 2.3238956 1.7413021 4.150054 0.3297843 0.2600045 0.4887610 1.337896 0.3737554 0.4614472

[,11] [,12]

[1,] 4.8113368 4.0657750

[2,] 1.2278193 1.0090944

[3,] 0.5095428 0.2567247

| Está correto!

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

| Agora temos que atribuir um cluster a cada ponto. Para fazer isso, vamos olhar para cada coluna (cada um dos

| 12 pontos)... Mas e aí? O que fazer?

1: Escolher a maior linha

2: Escolher a menor linha

3: Somar as 3 linhas.

Selection: 1

| Tente outra vez.

| Atribuímos cada ponto ao centróide mais próximo a ele. Cada linha representa um centróide e cada coluna um

| dos pontos? O que fazer?

1: Escolher a maior linha

2: Escolher a menor linha

3: Somar as 3 linhas.

Selection: 3

| Boa tentativa, mas não é exatamente o que estou esperando. Tente novamente.

| Atribuímos cada ponto ao centróide mais próximo a ele. Cada linha representa um centróide e cada coluna um

| dos pontos? O que fazer?

1: Escolher a maior linha

2: Somar as 3 linhas.

3: Escolher a menor linha

Selection: 1

| Essa não é a resposta esperada, mas tente novamente.

| Atribuímos cada ponto ao centróide mais próximo a ele. Cada linha representa um centróide e cada coluna um

| dos pontos? O que fazer?

1: Somar as 3 linhas.

2: Escolher a menor linha

3: Escolher a maior linha

Selection: 1

| Não exatamente. Tente outra vez.

| Atribuímos cada ponto ao centróide mais próximo a ele. Cada linha representa um centróide e cada coluna um

| dos pontos? O que fazer?

1: Escolher a maior linha

2: Escolher a menor linha

3: Somar as 3 linhas.

Selection: 2

| Maravilha!

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

| Das entradas de distTmp, para qual cluster o ponto 6 deve ser atribuído?

1: 1

2: 3

3: nenhuma

4: 2

Selection: 1

| Tente outra vez.

| Qual linha que na coluna 6 tem o menor valor?

1: 2

2: 3

3: 1

4: nenhuma

Selection: 2

| Você é muito bom, amig@!

|================================ | 32%

| O R tem uma função útil chamada which.min que diz qual o índice do menor valor de um vetor. Você pode

| aplicar a todas as colunas de distTmp com uma chamada. Simplesmente chame a função R apply com 3 argumentos.

| O primeiro é distTmp, o segundo é 2, ou seja, as colunas de distTmp, e o terceiro é which.min, a função que

| você deseja aplicar às colunas de distTmp. Tente isso agora.

> apply(distTmp, 2, which.min)

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

| Excelente!

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

| Você pode ver que você estava certo! O menor valor da 6ª coluna é de fato o que está na linha 3 como você

| respondeu antes. Vemos que as 3 primeiras entradas foram atribuídas ao segundo cluster (laranja) e apenas 2

| pontos (4 e 8) foram atribuídos ao primeiro cluster (vermelho).

...

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

| Faça o mesmo comando, mas agora armazene o resultado na variável newClust .

> newClust = apply(distTmp, 2, which.min)

| Mas uma vez. Você consegue! Ou digite info() para mais opções.

| Digite newClust<-apply(distTmp, 2, which.min) no console.

> newClust <- apply(distTmp, 2, which.min)

| Você acertou!

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

| Eu armazenei as cores do cluster ("red", "orange", "purple") no vetor cols1 para você. Agora vamos colorir

| os 12 pontos de acordo com suas atribuições. Novamente, use o comando points com 5 argumentos. Os primeiros

| 2 são x e y. O terceiro é pch=19, o quarto é cex=2 e o último, col=cols1[newClust].

> points(x, y, pch=19, cex=2, col=cols1[newClust])

| Você está indo muito bem!

|====================================== | 38%

| Agora temos que recalcular nossos centróides para que eles sejam a média (centro de gravidade) do conjunto

| de pontos atribuídos a eles. Temos que fazer as coordenadas x e y separadamente. Nós vamos fazer a

| coordenada x primeiro. Lembre-se de que os vetores x e y mantêm as respectivas coordenadas de nossos 12

| pontos de dados.

...

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

| Podemos usar a função R tapply que aplica "uma função sobre um array irregular". Isso significa que cada elemento da matriz é atribuído a um fator e a função é aplicada a subconjuntos da matriz (identificados pelo vetor de

| fator). Isso nos permite aproveitar o novo vetor de fator newClust que calculamos. Chame tapply agora com 3 argumentos, x (os dados), newClust (a matriz de fatores) e média (a função a ser aplicada).

> tapply(x, newClust,mean)

1 2 3

1.210767 1.010320 2.498011

| Excelente trabalho!

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

| Estas são as coordenadas centrais no eixo X dos pontos vermelhos, laranjas e roxos. Isso quer dizer que deve ser a nova coordenada X de cada um dos respectivos centróides.

...

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

| Repita o comando tapply, mas agora faça com o vetor y no primeiro parâmetro ao invés de x.

> tapply(y, newClust,mean)

1 2 3

1.730555 1.016513 1.354373

| Continue assim e você chegará lá!

|============================================= | 44%

| Como sou legal, armazenei esse valores que você calculou nos vetores newCx e newCy. Para tanto, eu executei as instruções newCx <- tapply(x, newClust, mean) e newCy <- tapply(y, newClust, mean);

...

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

| Agora que temos novas coordenadas x e y para os 3 centróides, podemos plotá-las. Use o comando points com as variáveis newCx e newCy como os 2 primeiros argumentos. Além disso, use os argumentos col = cols1, pch = 8

| (asterisco), cex = 2 e lwd = 2.

> points(newCx, newCy, col=cols1, pch=8, cex=2, lwd=2)

| Maravilha!

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

| Vemos como os centróides se aproximaram de seus respectivos clusters. Isso é especialmente verdadeiro no segundo cluster (laranja). Agora chame a função que calcula a distância mdist com os 4 argumentos x, y, newCx e newCy.

| Isso nos permitirá reatribuir os pontos de dados a novos clusters, se necessário.

> mdist(x, y, newCx, newCy)

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]

[1,] 0.98911875 0.539152725 0.2901879 1.0286979 0.7936966 0.8004956 0.4650664 1.028698 3.3053706 3.282778 3.5391512 2.9345445

[2,] 0.09287262 0.002053041 0.0734304 0.2313694 1.9333732 1.8320407 1.4310971 2.926095 3.5224442 3.295301 3.5990955 3.2097944

[3,] 3.28531180 2.197487387 1.6676725 4.0113796 0.4652075 0.3721778 0.6043861 1.643033 0.2586908 0.309730 0.3610747 0.1602755

| Excelente trabalho!

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

| Agora, faça o mesmo comando, mas armazene o resultado na variável distTmp2.

> distTmp2 <- mdist(x, y, newCx, newCy)

| Na mosca! Bom trabalho!

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

| Confira o conteúdo da variável distTmp2.

> distTmp2

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]

[1,] 0.98911875 0.539152725 0.2901879 1.0286979 0.7936966 0.8004956 0.4650664 1.028698 3.3053706 3.282778 3.5391512 2.9345445

[2,] 0.09287262 0.002053041 0.0734304 0.2313694 1.9333732 1.8320407 1.4310971 2.926095 3.5224442 3.295301 3.5990955 3.2097944

[3,] 3.28531180 2.197487387 1.6676725 4.0113796 0.4652075 0.3721778 0.6043861 1.643033 0.2586908 0.309730 0.3610747 0.1602755

| Na mosca! Bom trabalho!

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

| Lembre-se de que o primeiro cluster é vermelho, o segundo laranja e o terceiro roxo. Observe atentamente as colunas 4 e 7 do distTmp2. O que acontecerá com os pontos 4 e 7?

1: Nada

2: Eles são os únicos pontos que não mudam de cluster

3: Ambos mudarão para o cluster 2

4: Ambos devem mudar de cluster

Selection: 1

| Quase! Tente novamente.

| Duas das escolhas estão obviamente erradas. Isso deixa duas possibilidades que são semelhantes. Observe atentamente os números nas colunas 4 e 7 para ver onde estão os valores mínimos.

1: Ambos mudarão para o cluster 2

2: Eles são os únicos pontos que não mudam de cluster

3: Ambos devem mudar de cluster

4: Nada

Selection: 1

| Você quase acertou. Tente novamente.

| Duas das escolhas estão obviamente erradas. Isso deixa duas possibilidades que são semelhantes. Observe atentamente os números nas colunas 4 e 7 para ver onde estão os valores mínimos.

1: Nada

2: Ambos devem mudar de cluster

3: Ambos mudarão para o cluster 2

4: Eles são os únicos pontos que não mudam de cluster

Selection: 2

| Mantenha esse bom nível!

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

| Agora, chame apply com 3 argumentos, distTmp2, 2 e which.min para encontrar as novas atribuições de cluster para os pontos.

> apply(distTmp2, 2, which.min)

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

| Você está indo muito bem!

|======================================================== | 56%

| Estes são os novos clusters de cada um dos 12 pontos. Para isso foi usado o critério da menor distância de cada ponto para cada um dos novos centróides.

...

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

| Agora que você viu a saída da instrução apply, execute novamente o mesmo comando e armazene o resultado na variável newClust2 .

> newClust2 <- apply(distTmp2, 2, which.min)

| Essa é a resposta que eu esperava.

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

| Use a função points para recolorir os pontos com suas novas atribuições. Novamente, passe 5 argumentos, x e y são os primeiros, seguidos por pch = 19, cex = 2 e col = cols1[newClust2].

> points(x, y, pch=19, cex=2 col=cols1[newClust2])

Error: unexpected symbol in "points(x, y, pch=19, cex=2 col"

> points(x, y, pch=19, cex=2, col=cols1[newClust2])

| Perseverança é a resposta.

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

| Observe que os pontos 4 e 7 mudaram de cluster, o ponto 4 mudou de 1 para 2 (vermelho para laranja) e o ponto 7 mudou de 3 para 2 (roxo para vermelho).

...

|=============================================================== | 62%

| Agora, use tapply para encontrar a coordenada x do novo centróide. Lembre-se que existem 3 argumentos, x,

| newClust2 e mean.

> tapply(x, newClust2,mean)

1 2 3

1.8878628 0.8904553 2.6001704

| Na mosca! Bom trabalho!

|================================================================ | 63%

| Faça o mesmo para encontrar a nova coordenada y.

> tapply(y, newClust2,mean)

1 2 3

2.157866 1.006871 1.274675

| Na mosca! Bom trabalho!

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

| Eu armazenei essas coordenadas para você nas variáveis finalCx e finalCy. Plote estes novos centróides

| usando a função points com 6 argumentos. Os dois primeiros são finalCx e finalCy. O argumento col = cols1,

| pch = 9, cex = 2 e lwd = 2.

> plot(finalCx, finalCy, col=cols1, pch=9, cex=2, lwd=2)

| Essa não é a resposta esperada, mas tente novamente. Ou digite info() para mais opções.

| Digite points(finalCx, finalCy, col = cols1, pch = 9, cex = 2, lwd = 2) no console.

> points(finalCx, finalCy, col = cols1, pch = 9, cex = 2, lwd = 2)

| Ótimo trabalho!

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

| Deveria ser óbvio que se continuássemos este processo, os pontos de 5 a 8 ficariam vermelhos, os pontos de 1

| a 4 ficariam cor de laranja, e os pontos de 9 a 12 ficariam roxos.

...

|===================================================================== | 68%

| Agora que você passou por um exemplo passo a passo, ficará aliviado ao saber que o R fornece um comando para

| fazer todo esse trabalho para você. Não é novidade que se chama kmeans e, embora tenha vários parâmetros,

| vamos apenas mencionar quatro. Estes são x, (a matriz numérica dos dados), centers, iter.max e nstart. O

| segundo parâmetro (centers) pode ser um número de clusters ou um conjunto de centróides iniciais. O

| terceiro, iter.max, especifica o número máximo de iterações a serem percorridas e nstart é o número de

| iniciações aleatórias que você deseja tentar, se especificar centros como um número.

...

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

| Vamos tentar agora a função kmeans com 2 argumentos, `x=dataFrame` (que contém as coordenadas x e y de

| nossos 12 pontos) e o numero de centróides `centers = 3`.

> kmeans(x=dataFrame, centers=3)

K-means clustering with 3 clusters of sizes 4, 4, 4

Cluster means:

x y

1 0.8904553 1.0068707

2 2.8534966 0.9831222

3 1.9906904 2.0078229

Clustering vector:

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

Within cluster sum of squares by cluster:

[1] 0.34188313 0.03298027 0.34732441

(between\_SS / total\_SS = 93.6 %)

Available components:

[1] "cluster" "centers" "totss" "withinss" "tot.withinss" "betweenss" "size"

[8] "iter" "ifault"

| Você acertou!

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

| Gostei, mas a saída foi toda para o console. Armazene a saída na variável kmObj.

> kmObj <- kmeans(x=dataFrame, centers=3)

| Você está em um bom ritmo!

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

| Antes de explorar as variáveis individualmente, gostaria que você explorasse um pouco a saída da função

| kmeans com o comando View(kmObj).

> View(kmObj)

| Bom trabalho!

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

| O programa retorna a informação de que os dados se agrupam em 3 clusters de tamanho 4. Também retorna as

| coordenadas das 3 médias de cluster, um vetor chamado cluster indicando como os 12 pontos foram

| particionados nos clusters e a soma dos quadrados dentro de cada um. grupo. Também mostra todos os

| componentes disponíveis retornados pela função. Para ver quantas iterações o algoritmo passou, digite

| kmObj$iter (sim, digite isso no console).

> kmObj$iter

[1] 1

| Na mosca! Bom trabalho!

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

| Ei só queria enfatizar como você pode acessar as informações disponíveis no objeto kmObj. Vamos traçar a cor dos 12 pontos de acordo com o cluster deles. Isso foi armazenado em kmObj$cluster. Execute a instrução plot com 5

| argumentos. Os parâmetros x e y, são os dois primeiros; o terceiro parâmetro é col = kmObj$cluster. Os dois últimos argumentos são pch = 19 e cex = 2.

> plot(x,y, col=kmObj$cluster, pch=19, cex=2)

| Ótimo trabalho!

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

| Agora adicione os centróides que estão armazenados em kmObj$centers. Use a função points com 5 argumentos. Os dois primeiros são kmObj$centers e col = c("black","red","green"). Os três últimos, pch, cex e lwd, devem todos

| ser iguais a 3.

> points(kmObj$centers, col=c('black', 'red', 'green'), pch=3, cex=3, lwd=3)

| Na mosca! Bom trabalho!

|================================================================================ | 79%

| Agora para um pouco de diversão!

...

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

| Quero mostrar a você como a saída da função kmeans é afetada devido seu início aleatório (quando você

| solicita apenas um número de clusters). Quero que você execute a função várias vezes para ter uma ideia das

| relações entre suas observações. Vamos chamar kmeans com o mesmos 12 pontos (armazenados em dataFrame), mas

| pediremos 6 clusters em vez de 3.

...

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

| Vamos plotar nossos 12 pontos várias vezes. O argumento col que nos mostrará como a função kmeans está

| agrupando-os de forma diferente. Então, chame plot agora com 5 argumentos. Os primeiros 2 são x e y. O

| terceiro é col = kmeans(dataFrame, 6)$cluster. Os dois últimos são pch = 19 e cex = 2.

> plot(x, y, col=kmeans(dataFrame, 6)$cluster, pch=19, cex=2)

| Excelente trabalho!

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

| Viu como os pontos se agrupam? Agora digite novamente o último comando.

> plot(x, y, col=kmeans(dataFrame, 6)$cluster, pch=19, cex=2)

| Você está em um bom ritmo!

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

| Viu como os clusteres mudaram? Como os Teletubbies diriam, 'De novo!' (e digite mais uma vez)

> plot(x, y, col=kmeans(dataFrame, 6)$cluster, pch=19, cex=2)

| Continue assim e você chegará lá!

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

| Então, os clusteres mudam com diferentes valores iniciais. Me parece que 6 são muitos clusters para este

| dataset, fica como lição de casa identificar um bom número de clusters.

...

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

| Vamos fazer uma revisão ...

...

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

| Verdadeiro ou falso? O K-means clustering requer que você especifique um número de clusters antes de

| começar.

1: Verdadeiro

2: Falso

Selection: 1

| Mantenha esse bom nível!

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

| Verdadeiro ou falso? K-means clustering requer que você especifique um número de iterações antes de começar.

1: Falso

2: Verdadeiro

Selection: 1

| Você está indo muito bem!

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

| Verdadeiro ou falso? Cada conjunto de dados possui um único número fixo de clusters.

1: Falso

2: Verdadeiro

Selection: 1

| Excelente trabalho!

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

| Verdadeiro ou falso? K-means clustering sempre irá parar em 3 iterações.

1: Falso

2: Verdadeiro

Selection: 1

| Excelente trabalho!

|================================================================================================== | 97%

| Verdadeiro ou falso? Ao iniciar kmeans com centróides aleatórios, você sempre terminará com o mesmo cluster

| final.

1: Verdadeiro

2: Falso

Selection: 2

| Bom trabalho!

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

| Parabéns! Espero que você tenha achado útil o que você encontrou esta lição.

...

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

| Gostaria de informar ao professor sobre a conclusão desta lição

1: Sim

2: Não

Selection: 1

| Qual o código da sua turma? (exemplo FIAP-01IA)

24IA

| Qual seu código de aluno?

344154

| Qual seu nome?

Diego Cohen

| O que achou deste exercício?

Bastante confuso!

[1] "Tentando submeter ao professor, tentativa 1 ... (max 5) ..."

[1] "saved"

#############################################################################################################

Seu resultado foi salvo!

#############################################################################################################