Agrupamento de Dados

Inteligência na Web e Big Data

Fabricio Olivetti de França e Thiago Ferreira Covões
folivetti@ufabc.edu.br, thiago.covoes@ufabc.edu.br

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

CMCC
Centro de Matemática, Computação e Cognição
Center for Mathematics, Computation and Cognition

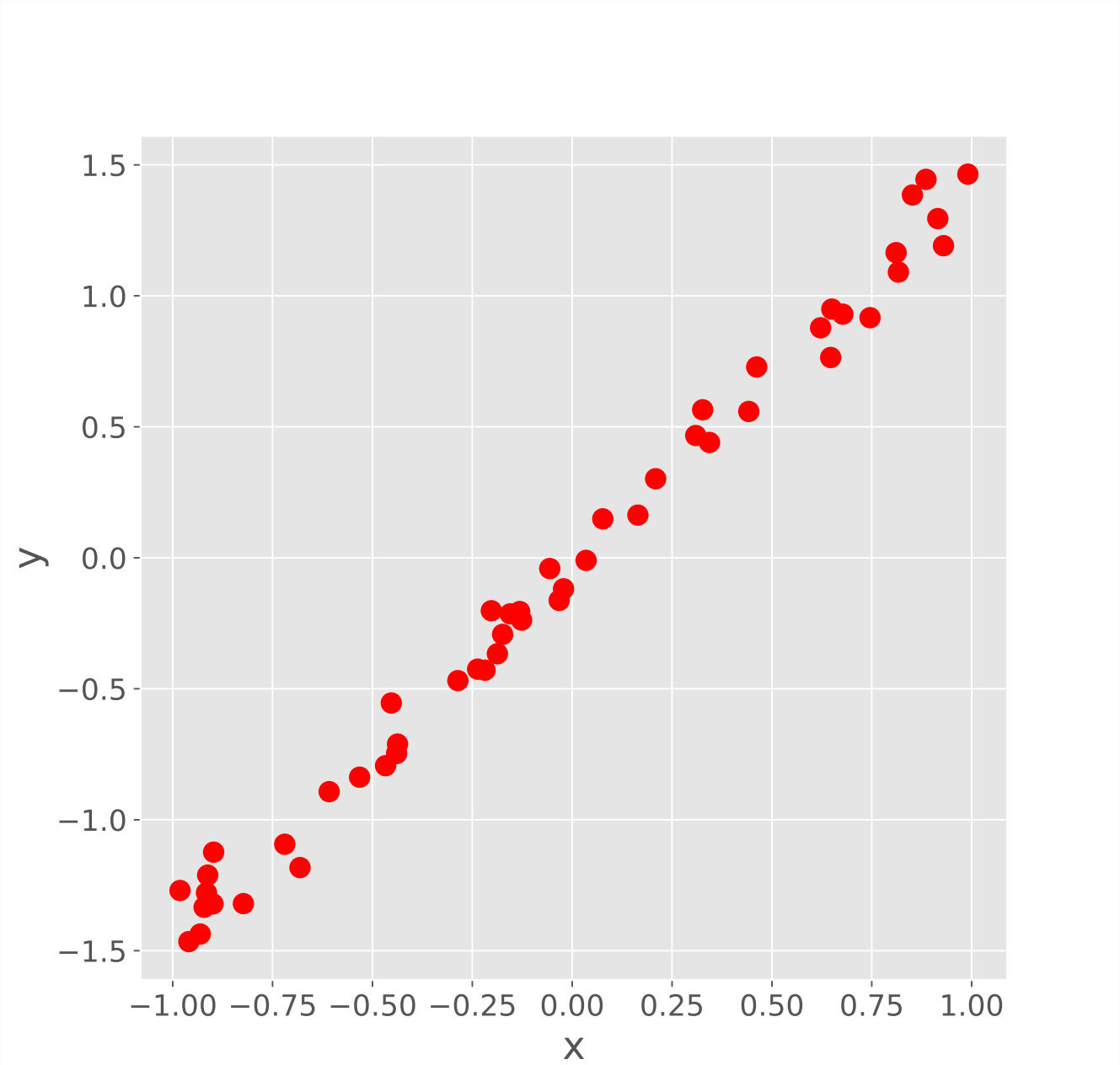
Análise de Componentes Principais

Principal Components Analysis (PCA)

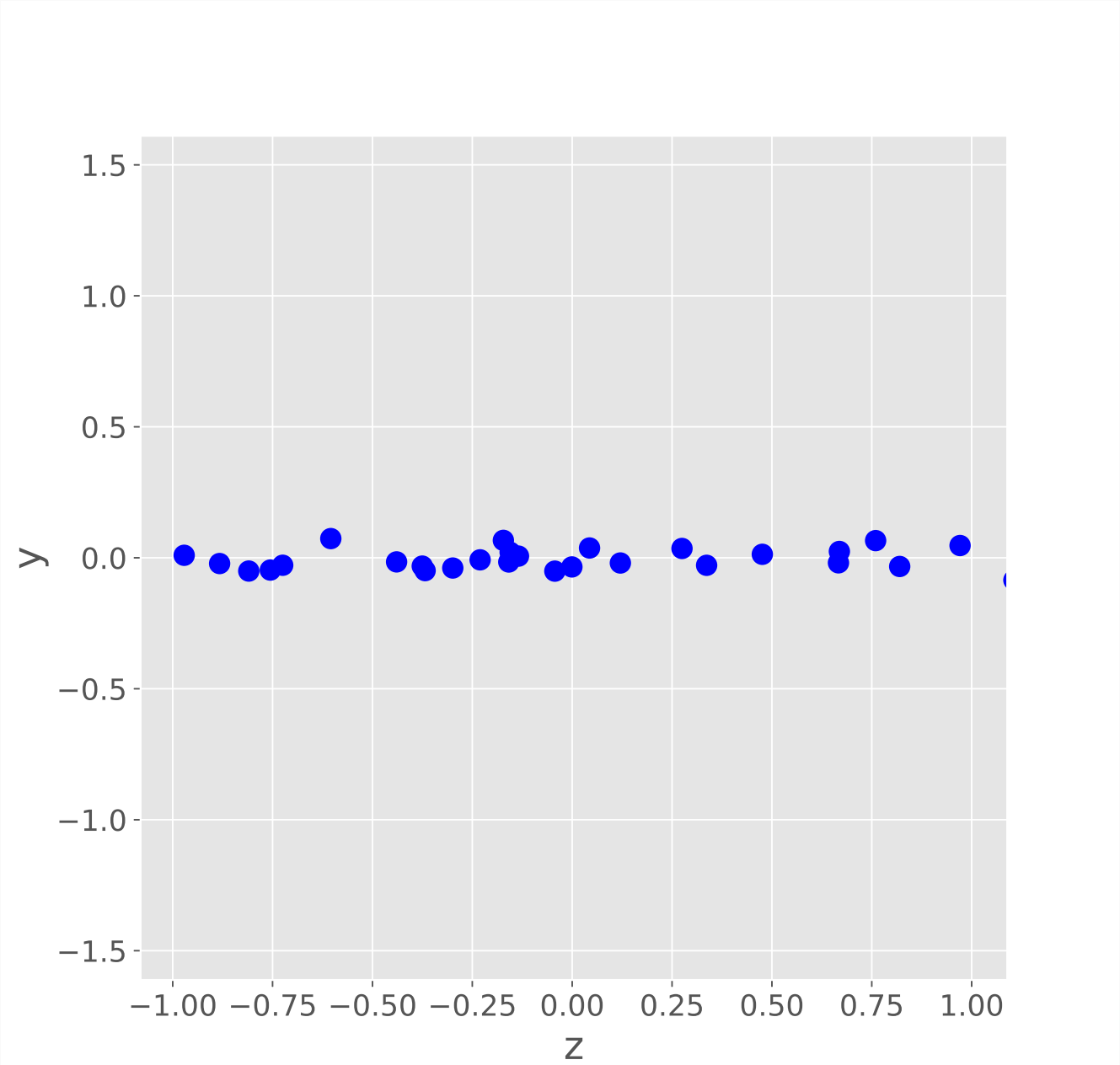
Identifica as direções de maior variação de valores.

Rotaciona o eixo para que cada eixo rotacionado represente da maior para a menor variação.

Principal Components Analysis (PCA)



Principal Components Analysis (PCA)



Principal Components Analysis (PCA)

Para isso utiliza a informação de autovalores e autovetores da matriz de covariância dos atributos.

Covariância dos Atributos

Dada uma matriz de dados $X \in \Re^{n \times d}$, centralizamos os pontos para que fiquem com média zero:

$$x'_{i,j} = x_{i,j} - \hat{x}_j,$$

com $\hat{x}_j$ sendo a média dos valores do atributo j.

Covariância dos Atributos

A covariância dos atributos pode ser calculada como:

$$Cov = \frac{1}{n} X'^T X',$$

que resulta em uma matriz $Cov \in \Re^{d \times d}$.

Covariância dos Atributos

O elemento i, j dessa matriz representa a correlação entre o atributo i e o atributo j.

A diagonal indica a variância do respectivo atributo.

Autovalores e Autovetores

Dessa matriz podemos extrair um total de $\mathbf{d}$ autovalores (λ) e autovetores (e) tal que:

$$Cov \cdot e = \lambda \cdot e$$

Autovalores e Autovetores

Se ordenarmos todos os autovalores/autovetores pela ordem do maior autovalor para o menor, temos que:

- Cada autovetor i representa a i-ésima direção de maior variação
- O autovalor correspondente quantifica essa variação

Principal Components Analysis (PCA)

Cada autovetor representa uma combinação linear dos atributos originais de tal forma a capturar a variação descrita pelo autovalor.

Basicamente a matriz de autovetores é uma base de dados rotacionada que captura a variação em ordem crescente.

Principal Components Analysis (PCA)

Se um autovalor for muito pequeno, significa que não existe variação naquele eixo e, portanto, ele pode ser descartado.

Principal Components Analysis (PCA)

Imagine um problema de classificação utilizando apenas uma variável x_j com variância baixa. É fácil perceber que tal variável não tem poder discriminatório pois, para toda classe, ela apresenta um valor muito similar.

Principal Components Analysis (PCA)

Rotacionando a base de dados De posse da matriz $E \in \Re^{d \times k}$ dos k primeiros autovetores com um valor significativo de λ, é possível transformar a matriz de dados centralizada X' com:

$$X^* = X' \times E.$$

Principal Components Analysis (PCA)

Isso transforma a matriz em uma matriz $X^* \in \Re^{n \times k}$ com $k < d$.

Principal Components Analysis (PCA)

Usos do PCA:

- Transformar a base de dados mantendo a dimensão.
- Reduzir a base de dados, eliminando eixos com pouca variância.
- Reduzir para duas dimensões para visualizar os dados.

Principal Components Analysis (PCA)

Quando utilizado para redução de dimensionalidade, verifique o custo-benefício de remover um eixo dado sua variância.

Principal Components Analysis (PCA)

Além disso, uma vez que a base é transformada, os atributos perdem totalmente seu significado original.

Principal Components Analysis (PCA)

Se temos os atributos m^2, garagens, andar, bairro; uma vez aplicado o PCA não sabemos que combinação linear cada novo atributo representa.

PCA funcional

```haskell
-- Recebe dados e devolve a matriz de rotação
pca :: Int -> Data -> RotMtx
pca k x = subMatrix k eigenvectors
  where
    (eigenvalues, eigenvectors) = descSort $ H.eig
    ↪  covariance
    covariance = multMtx (transpose x') x'
    x' = center x
```

PCA funcional

A matriz de covariância tem dimensão $d \times d$, costumeiramente cabendo na memória.

A matriz de dados e a de dados centralizadas não cabem em memória, portanto devemos pensar em opções de MapReduce para as funções *covariance* e *center*.

PCA funcional distribuído

Lembrando do algoritmo *Ordinary Least Square*, podemos calcular a matriz de covariância como a somatória dos produtos externos de cada linha de X, a centralização de dados também vimos em uma aula passada, no cálculo do Desvio-Padrãoo.

PCA - PySpark

Dada uma RDD X:

```
n = X.lenght()

# 1 x d
xmean = (X
          .map(lambda xi: xi/n)
          .reduce(lambda (xi, yj): xi+yj)
         )

Xcenter = X.map(lambda xi: xi - xmean)

Covar = (Xcenter
          .map(lambda (k1,xi): np.outer(xi, xi))
          .reduce(lambda x,y: x+y)
          )
```

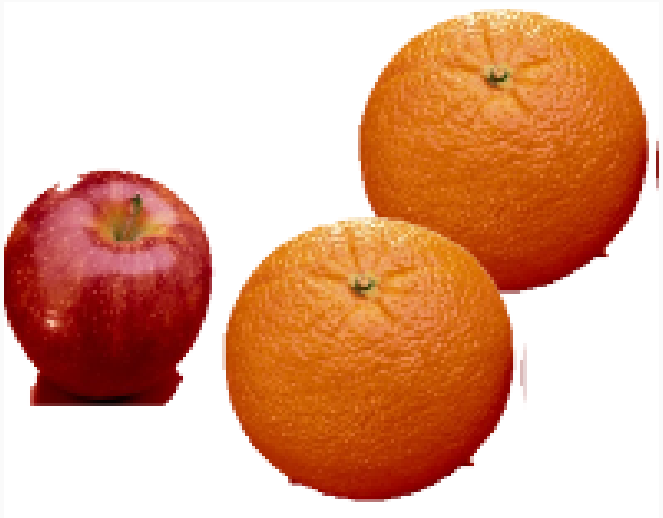
PCA - PySpark

Com *Covar* calculado basta fazer:

```
1 eva, eve = np.linalg.eigh(Covar)
2 idx = np.argsort(-eva)
3 pcaMtx = eve[:,idx[:k]]
```

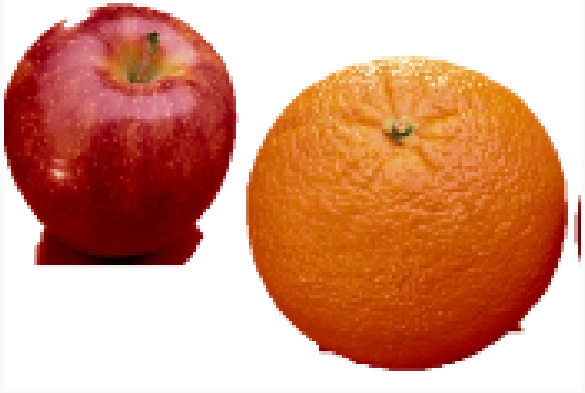
Agrupamento de Dados

Aprendizado Não-Supervisionado Muitas vezes não temos um rótulo pré-definido que queremos extrair de nossos dados.



<2->Contagem? 1 e 2

<3->Tipo?



Aprendizado Não-Supervisionado

O aprendizado não-supervisionado especifica as técnicas para extrair conhecimento de um conjunto de dados sem que tenhamos:

- Informação do que queremos encontrar.
- Qualquer retorno indicando corretude do que encontramos.

Aprendizado Não-Supervisionado

Essas técnicas se baseiam em:

- Formação de grupos de dados similares.
- Modelo de criação de um dado.
- Busca de objetos representativos.

Formação de grupos de dados

Define-se uma medida de similaridade entre nossos dados.

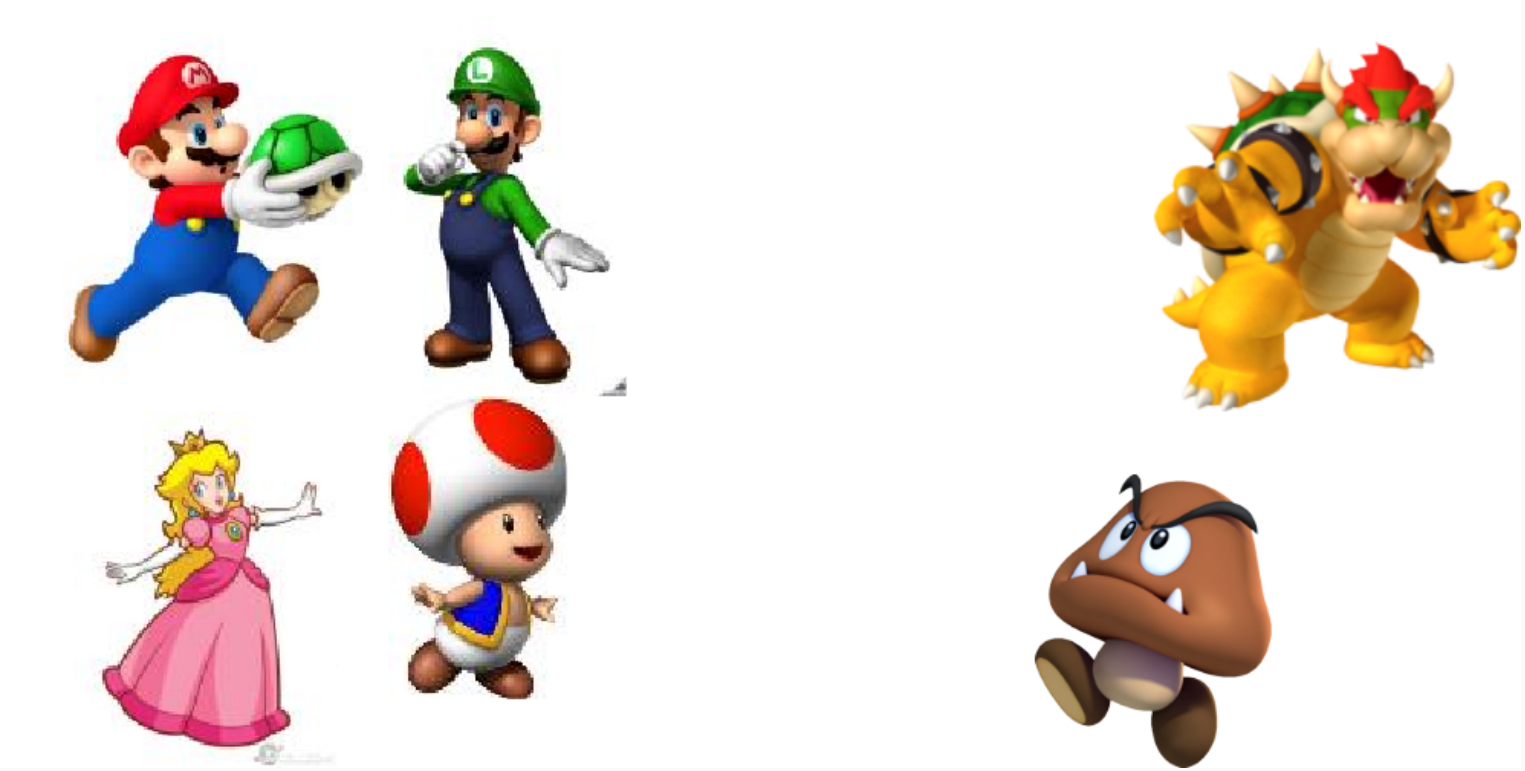
Forme grupos baseados na similaridade entre objetos do mesmo grupo.

Formação de grupos de dados

Nas imagens abaixo:

Formação de grupos de dados

Podemos formar os grupos ”bonzinhos”e ”malvados”:



Formação de grupos de dados

Podemos formar os grupos ”masculino”, ”feminino”e ”indefinido”:

Formação de grupos de dados

Podemos formar os grupos ”heróis”, ”vítimas”e ”vilões”:

Modelo de Criação

Procura encontrar um modelo que explica a geração dos dados:

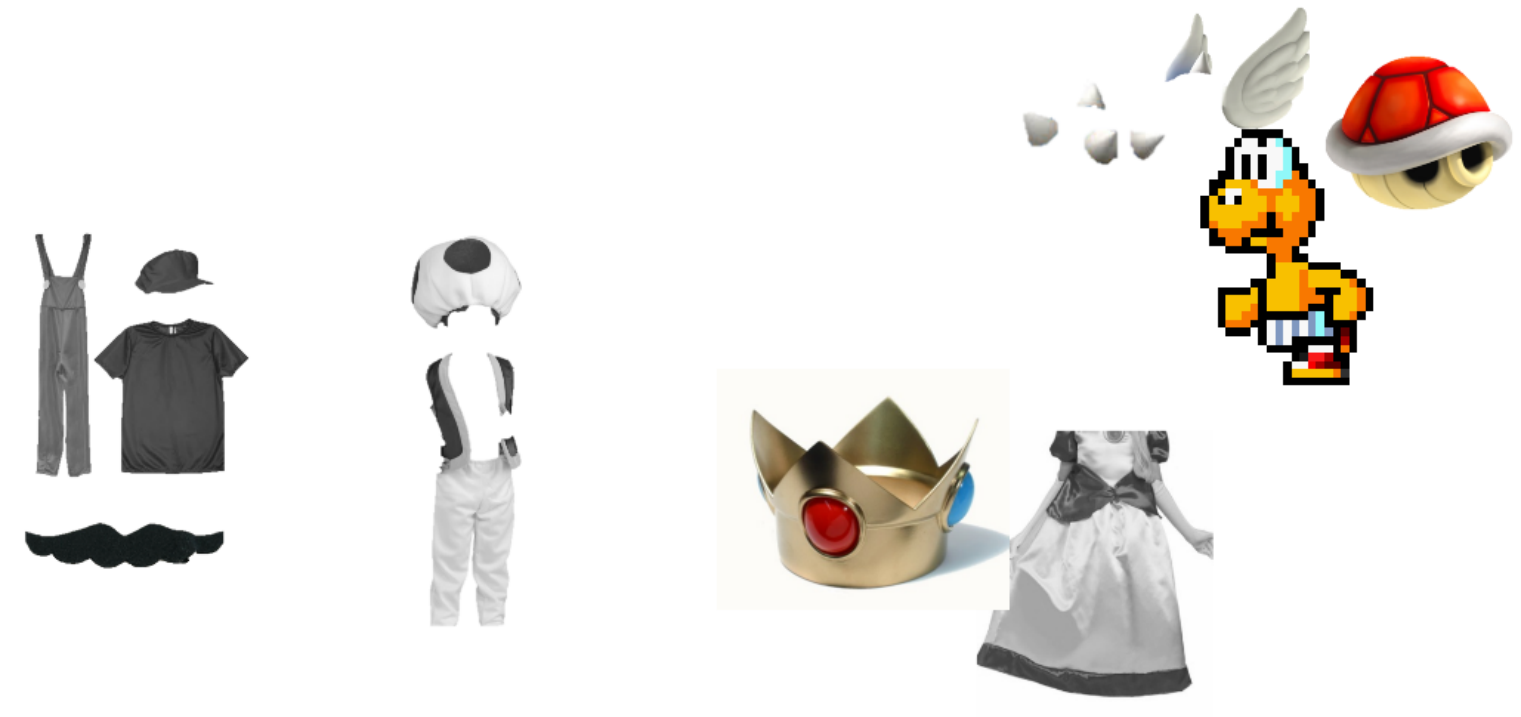
$$P(\theta \mid x) = ?$$

Modelo de Criação

Vamos tomar como exemplo agora as seguintes imagens:

Modelo de Criação

Com elas podemos encontrar os seguintes modelos de criação:

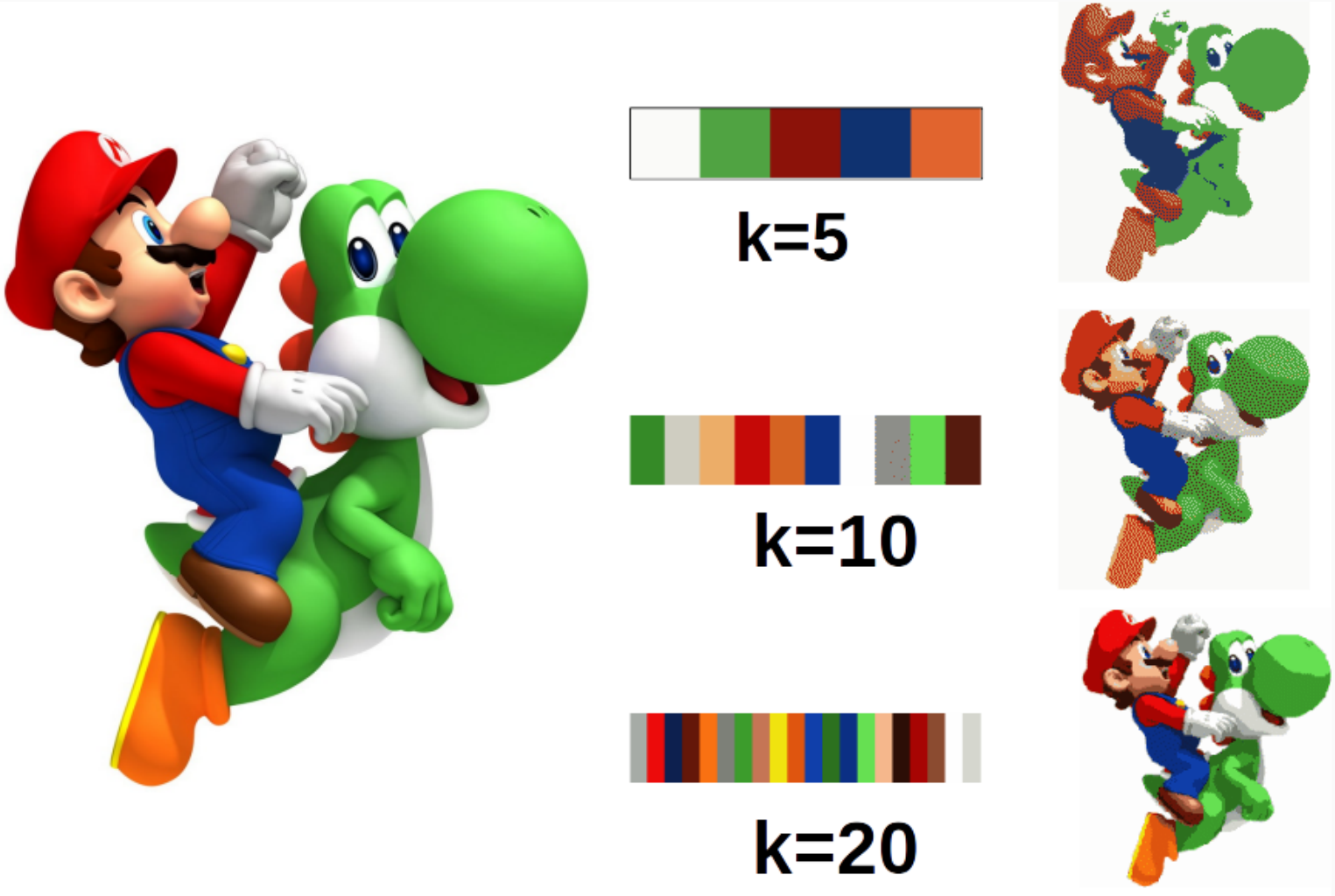


Dados Representativos

Vamos considerar que uma imagem é um conjunto de pixels, e as cores desses pixels são nossos dados.

Se encontrarmos as k cores mais representativa dessa imagem, podemos redesenhá-la utilizando essas cores, economizando bits!

Dados Representativos



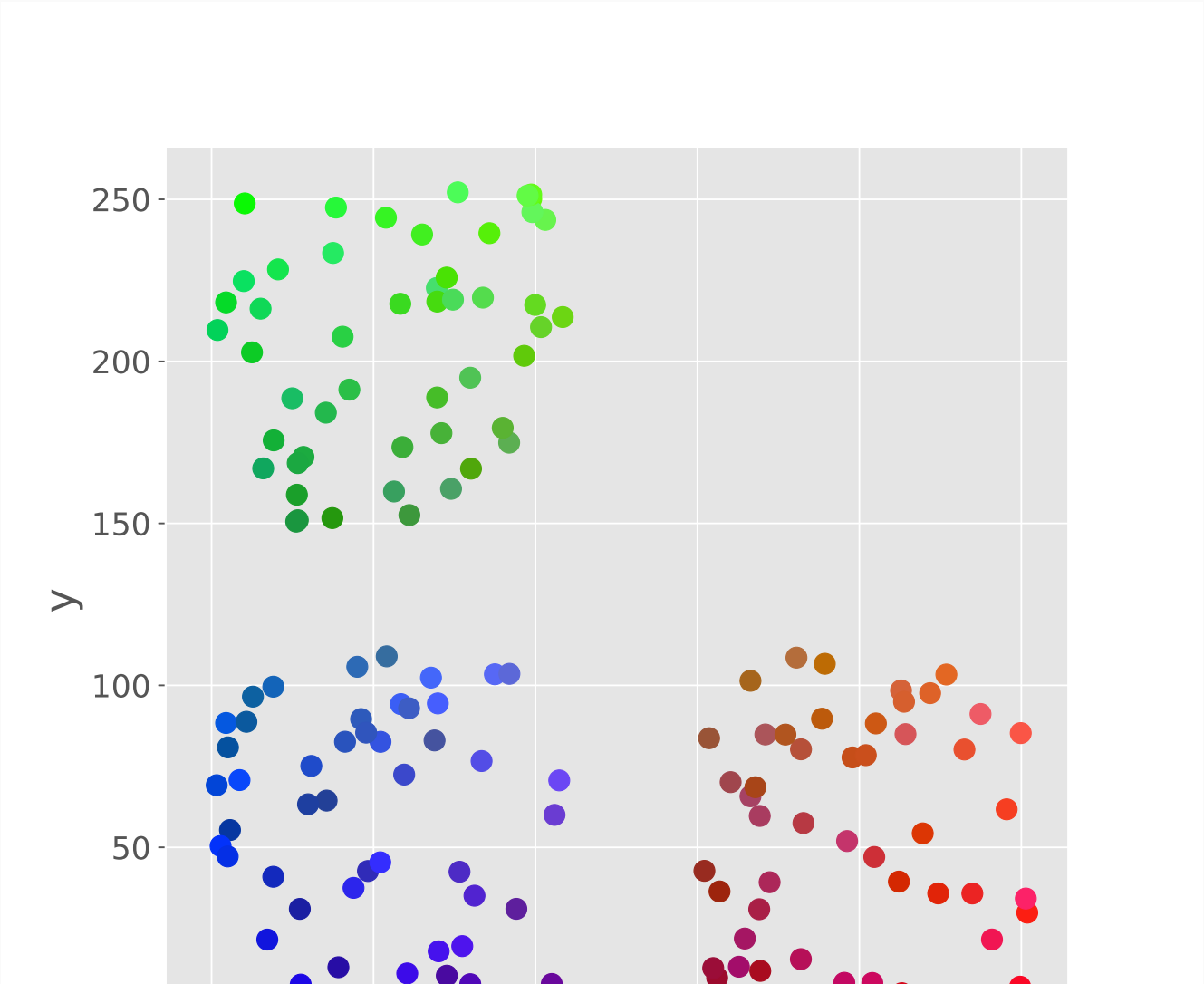
Objetos Representativos

Vamos estudar a busca pelas k cores mais representativas.

Os resultados ilustrados anteriormente utilizam o conceito de agrupamento, especificamente a técnica k-means.

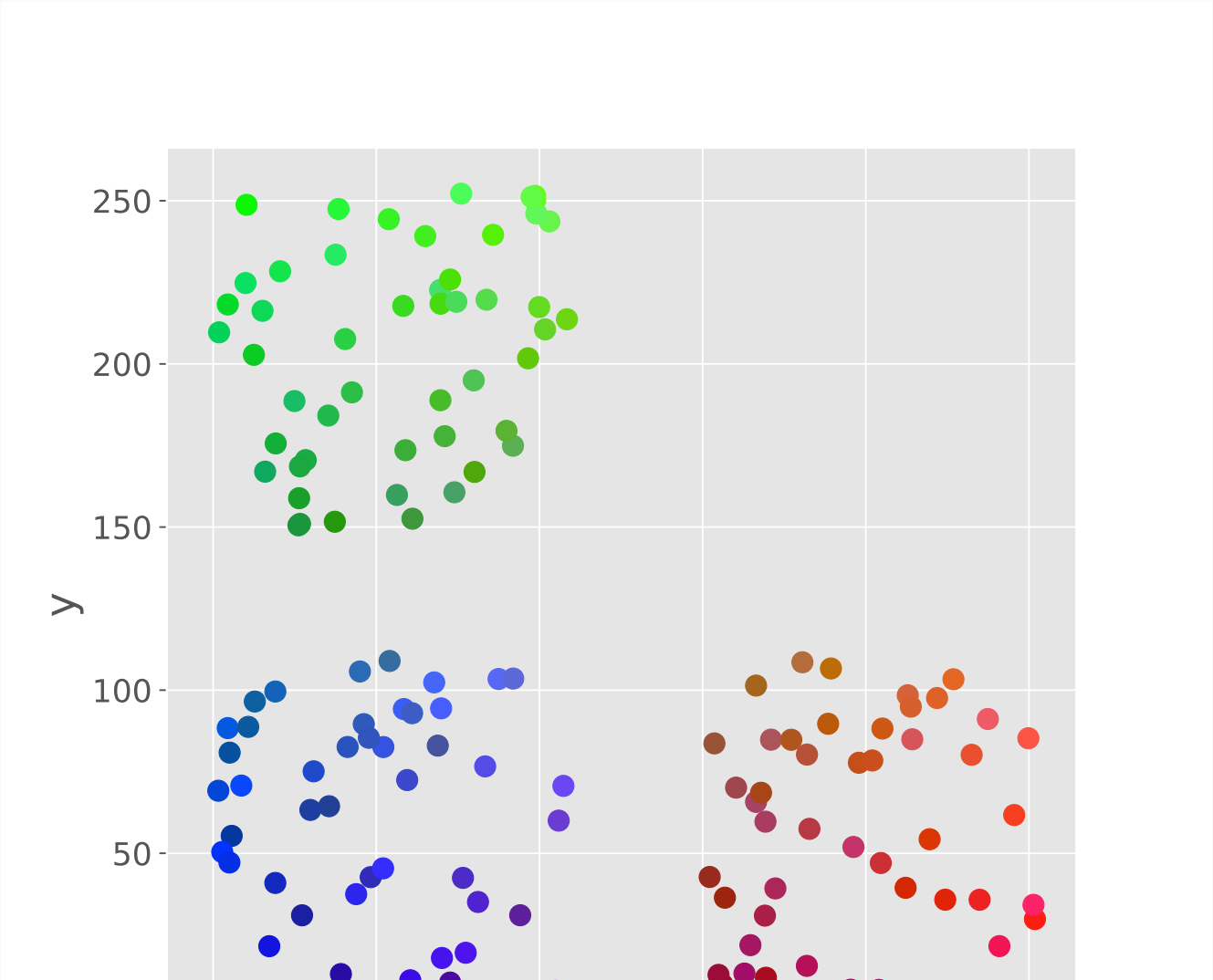
k-Means

Digamos que temos diversas cores que são tons de vermelho, verde e azul.



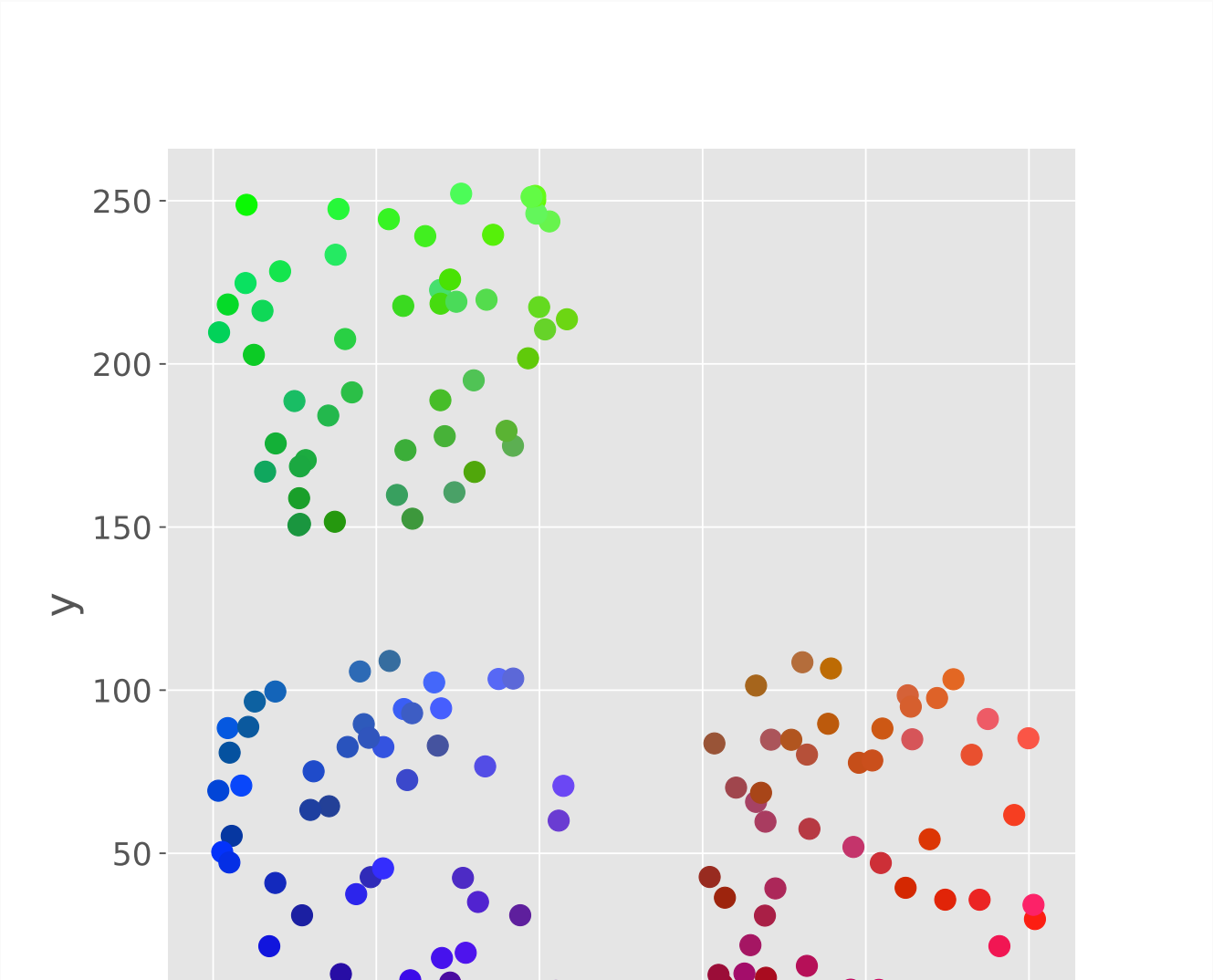
k-Means

Queremos escolher as 3 cores mais representativas desse conjunto!



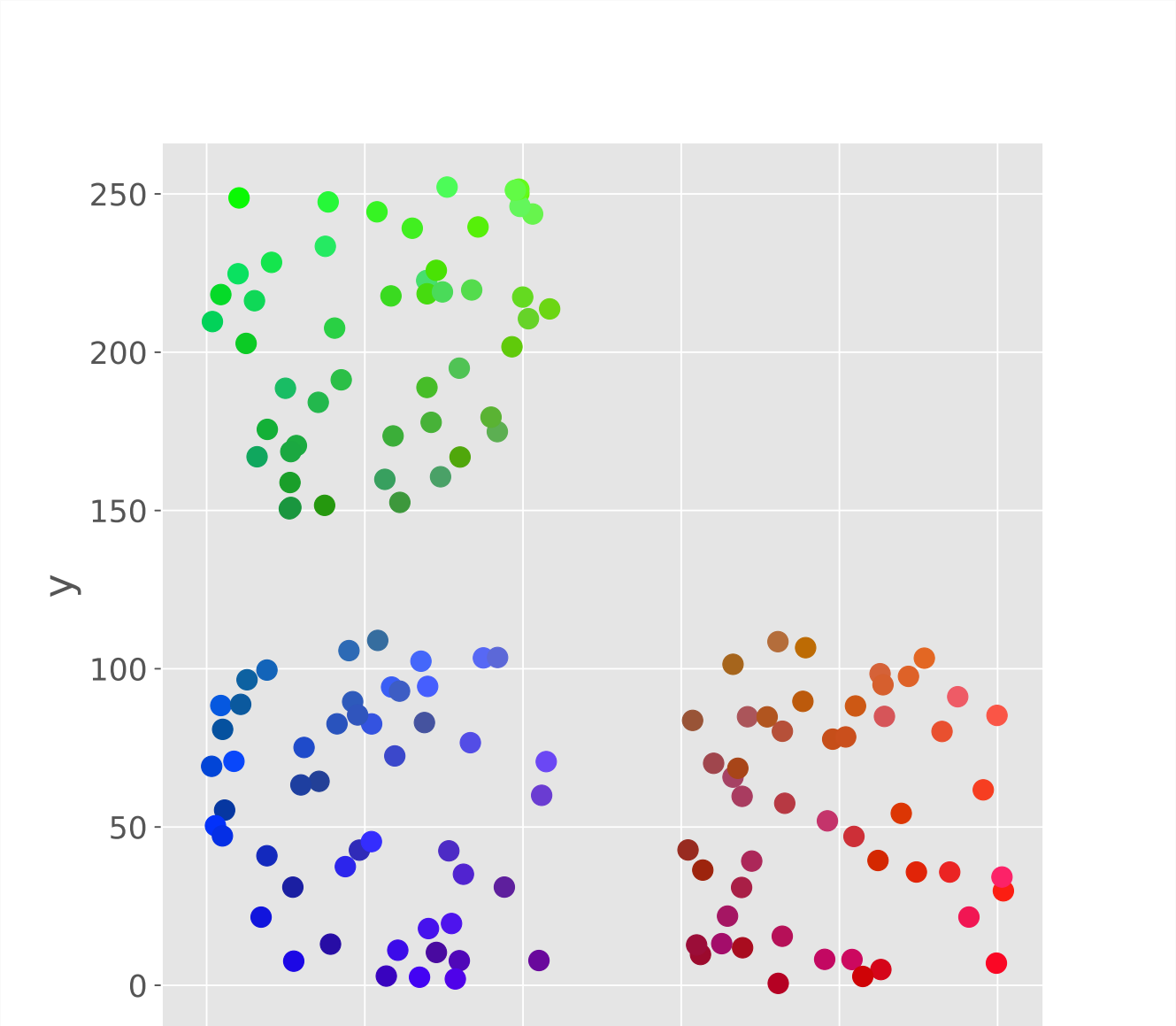
k-Means

E não necessariamente essas 3 cores devem fazer parte do conjunto de dados, podemos criar cores novas!



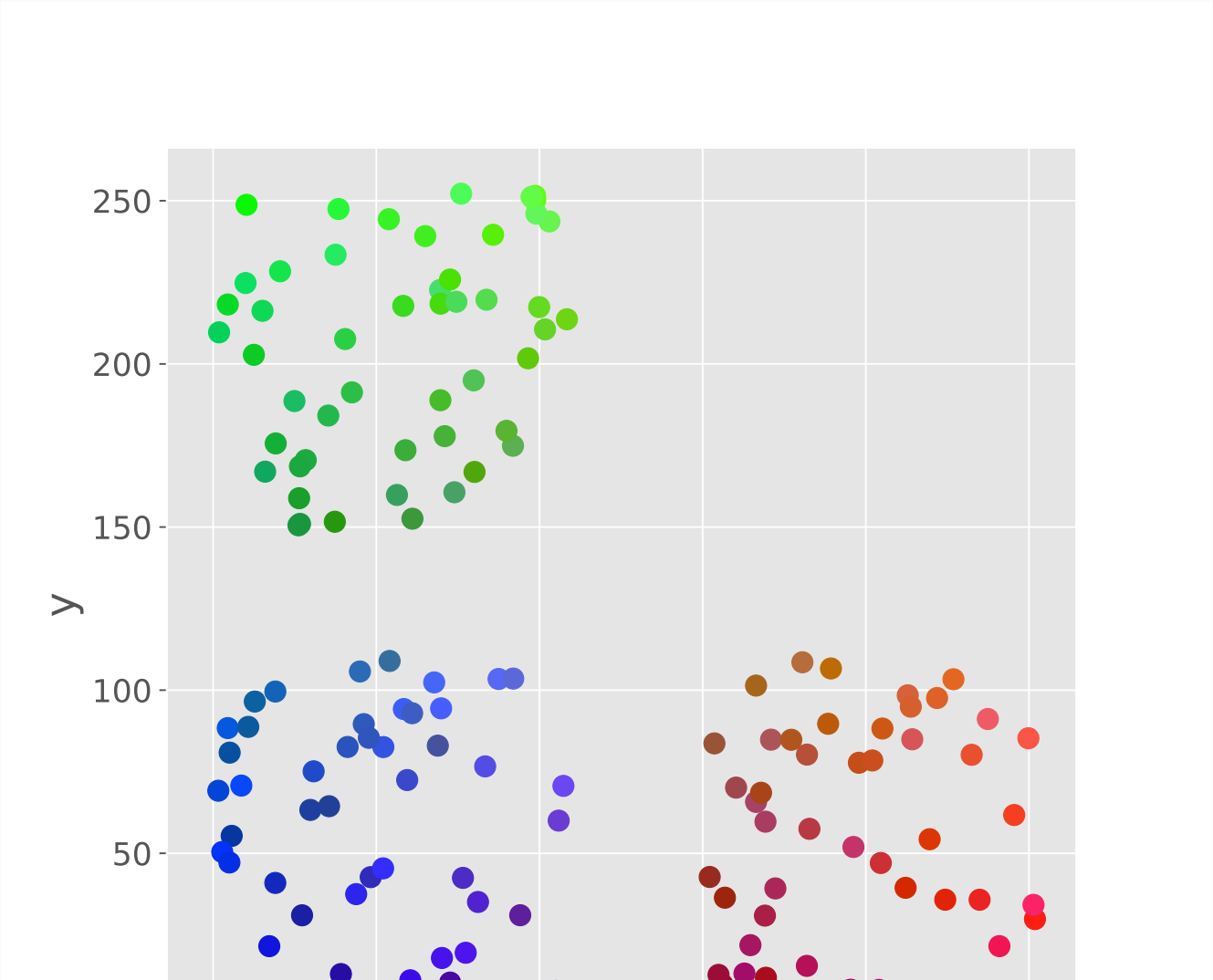
k-Means

O que caracteriza uma cor representativa de um conjunto?



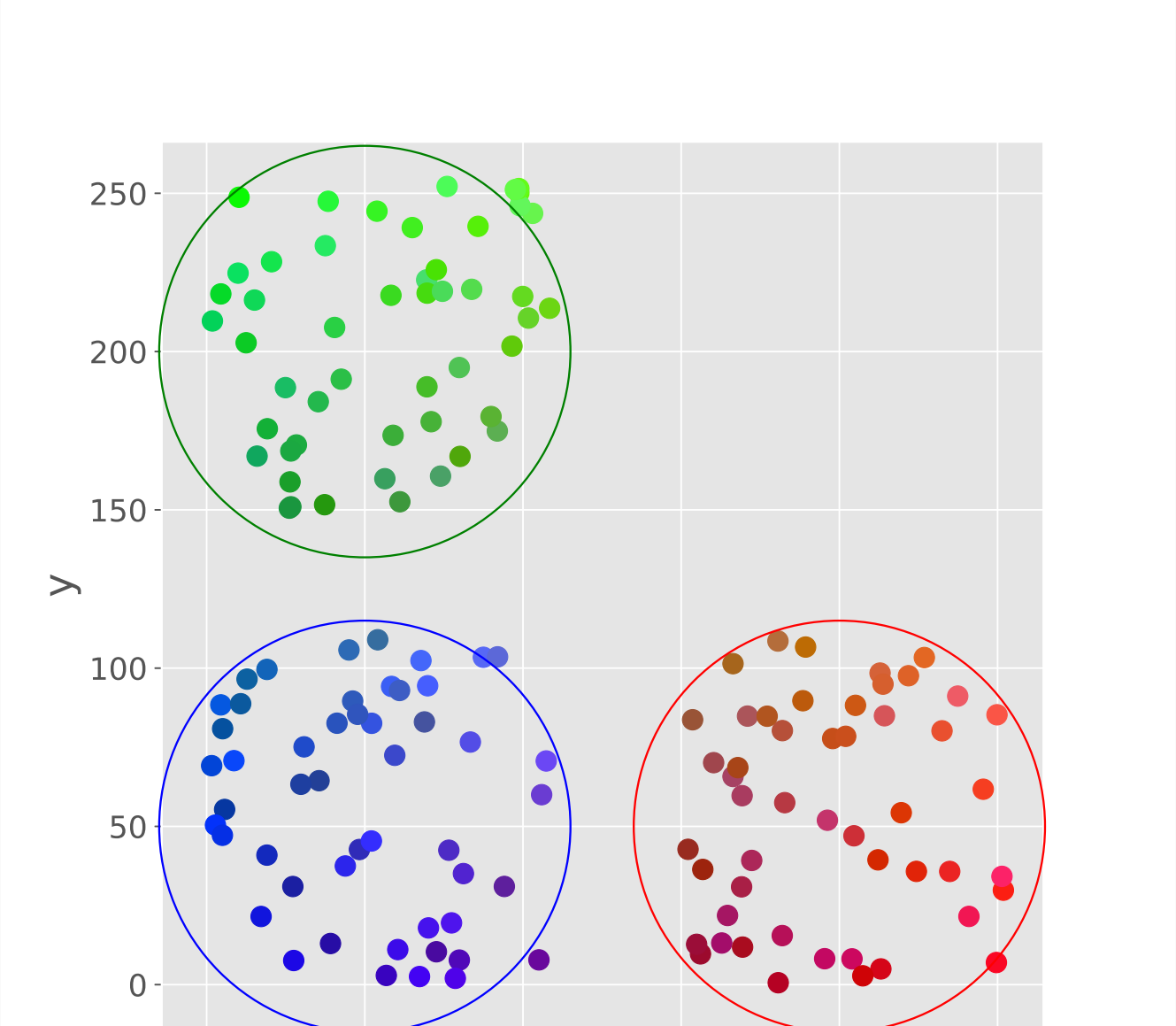
k-Means

Bom, para saber isso primeiro precisamos conhecer os conjuntos!



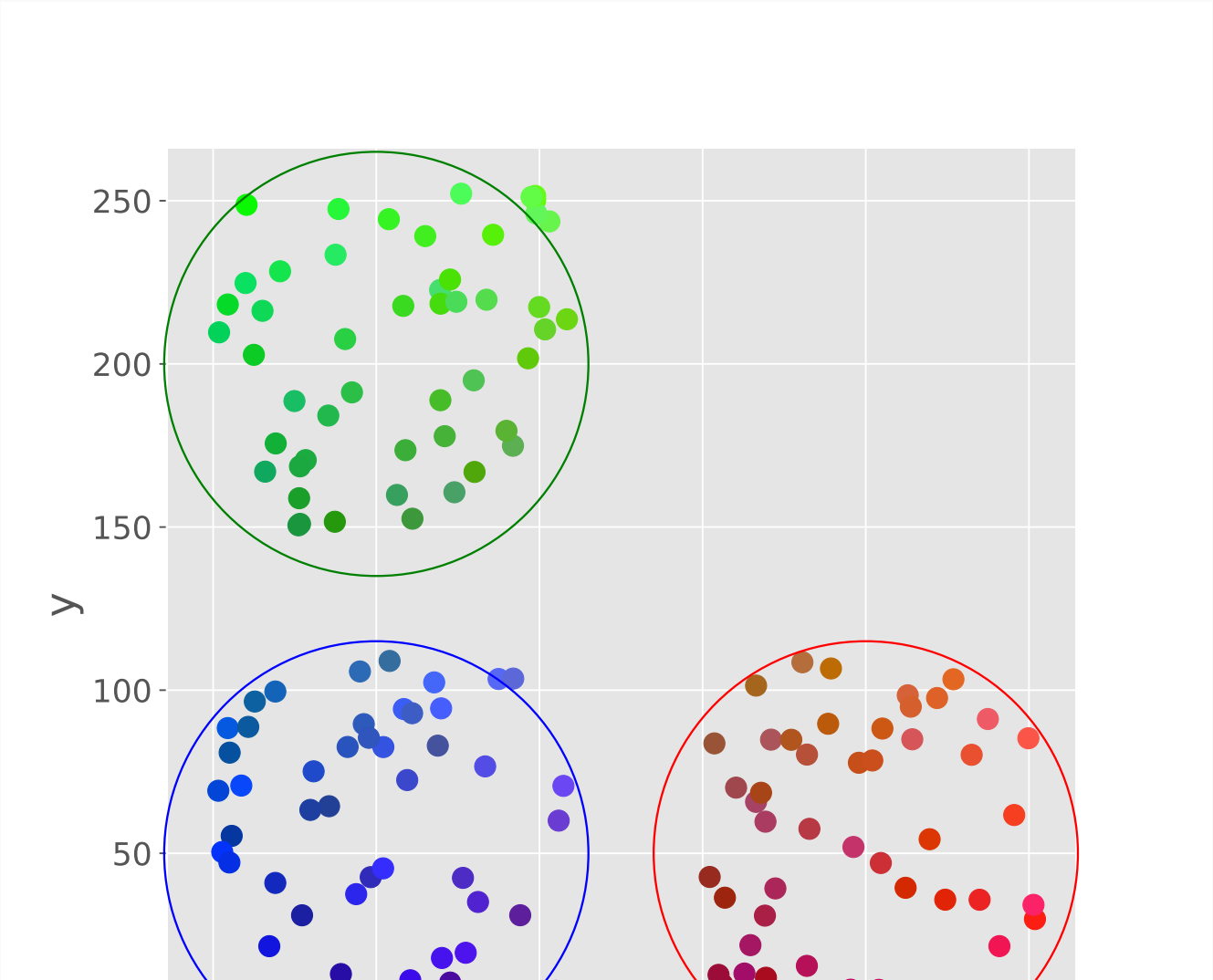
k-Means

Para nosso exemplo é fácil:



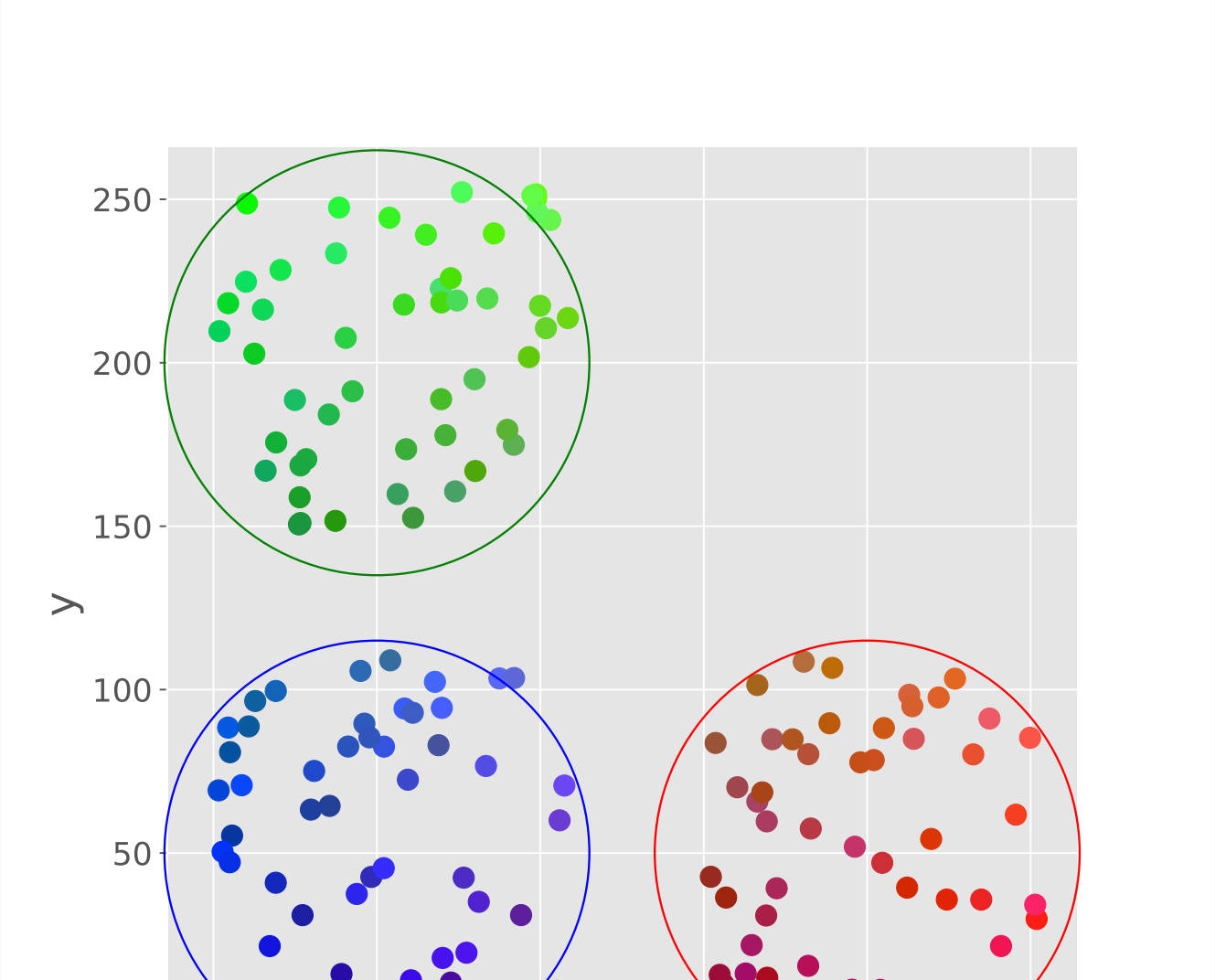
k-Means

A cor representativa de cada grupo é aquela que mais se aproxima de todas do grupo.



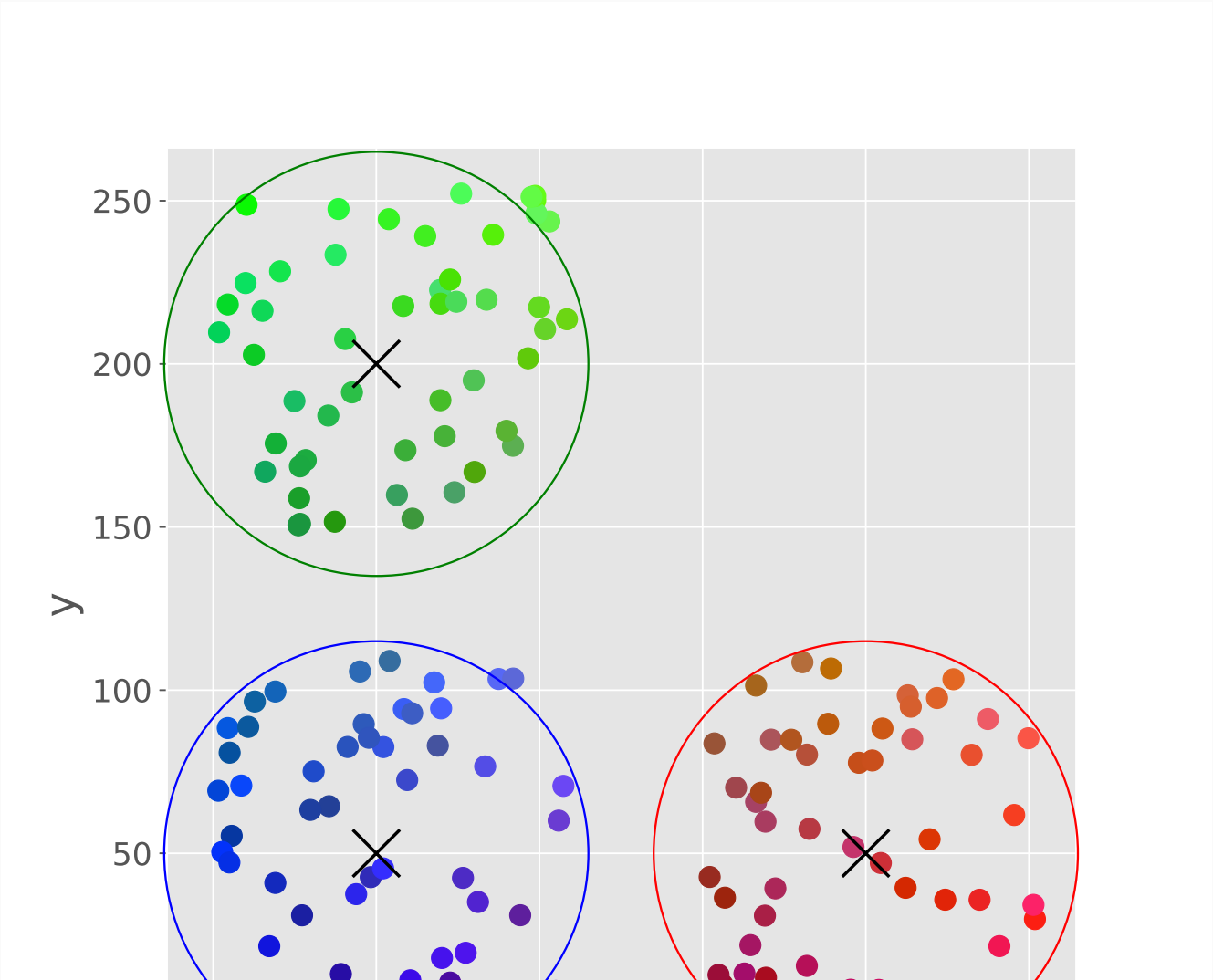
k-Means

Ou seja, é a que tem maior similaridade média com os elementos do grupo.



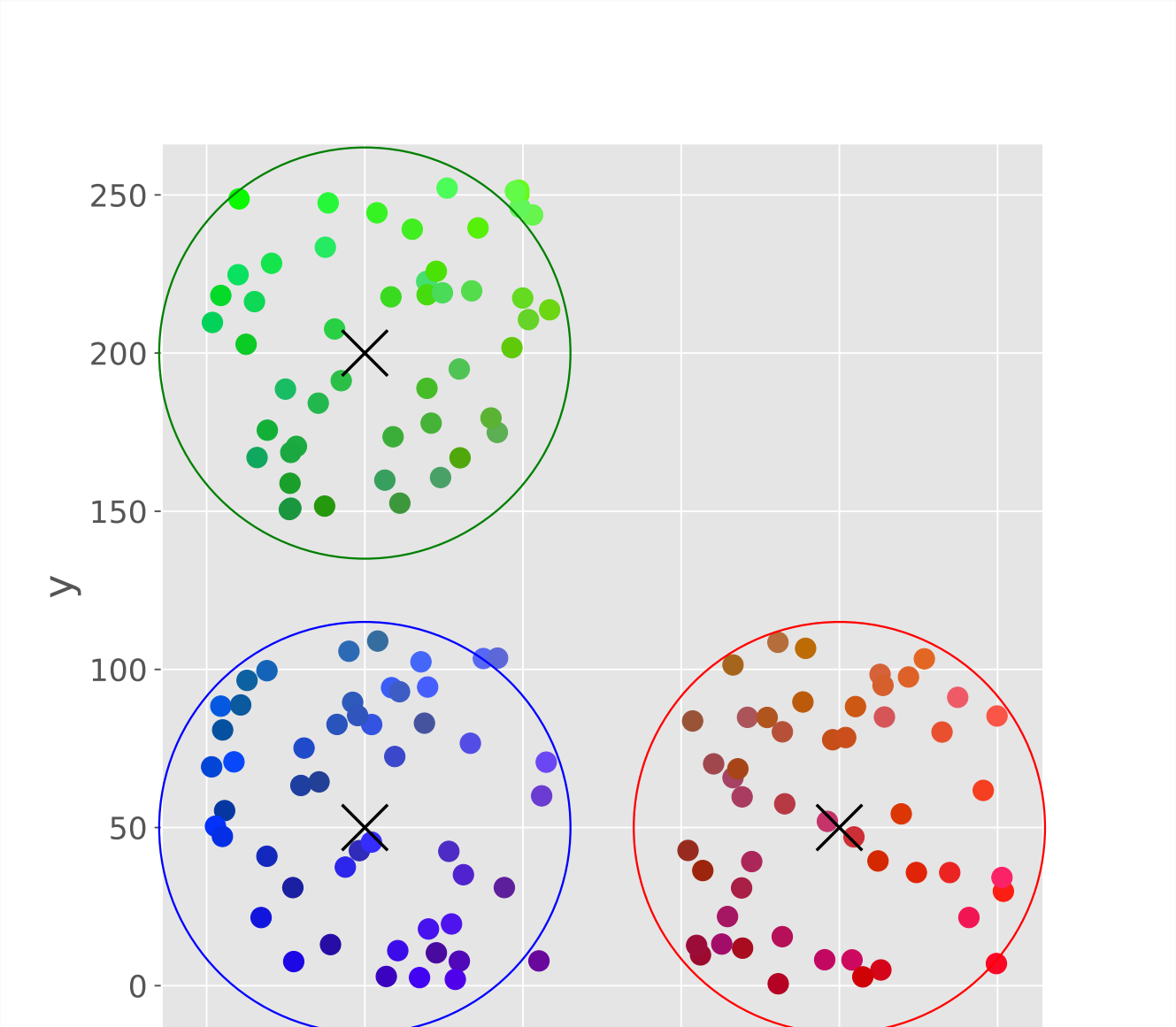
k-Means

Ou seja, é a que tem maior similaridade média com os elementos do grupo.



k-Means

Matematicamente, quem são esses elementos?



k-Means

ado um conjunto de pontos X n-dimensionais, queremos determinar o ponto c, representando o centro, que minimiza:

$$J(X, \mathbf{c}) = \frac{1}{m} \sum_{i=1}^{m} dist(\mathbf{x_i}, \mathbf{c})$$

definimos que a distância é o inverso da similaridade.

k-Means

O mínimo pode ser encontrado com:

$$\nabla J(X, \mathbf{c}) = \frac{1}{m} \sum_{i=1}^{m} \nabla dist(\mathbf{x_i}, \mathbf{c}) = 0$$

k-Means

Utilizando a distância Euclidiana temos:

$$\frac{1}{m}\sum_{i=1}^{m}\nabla(\mathbf{x_i}-\mathbf{c})^2 = \frac{2}{m}\sum_{i=1}^{m}(\mathbf{c}-\mathbf{x_i}) = 0$$

k-Means

$$\frac{2}{m}(mc - \sum_{i=1}^{m} \mathbf{x_i}) = 0$$

$$mc = \sum_{i=1}^{m} \mathbf{x_i}$$

$$c = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x_i}$$

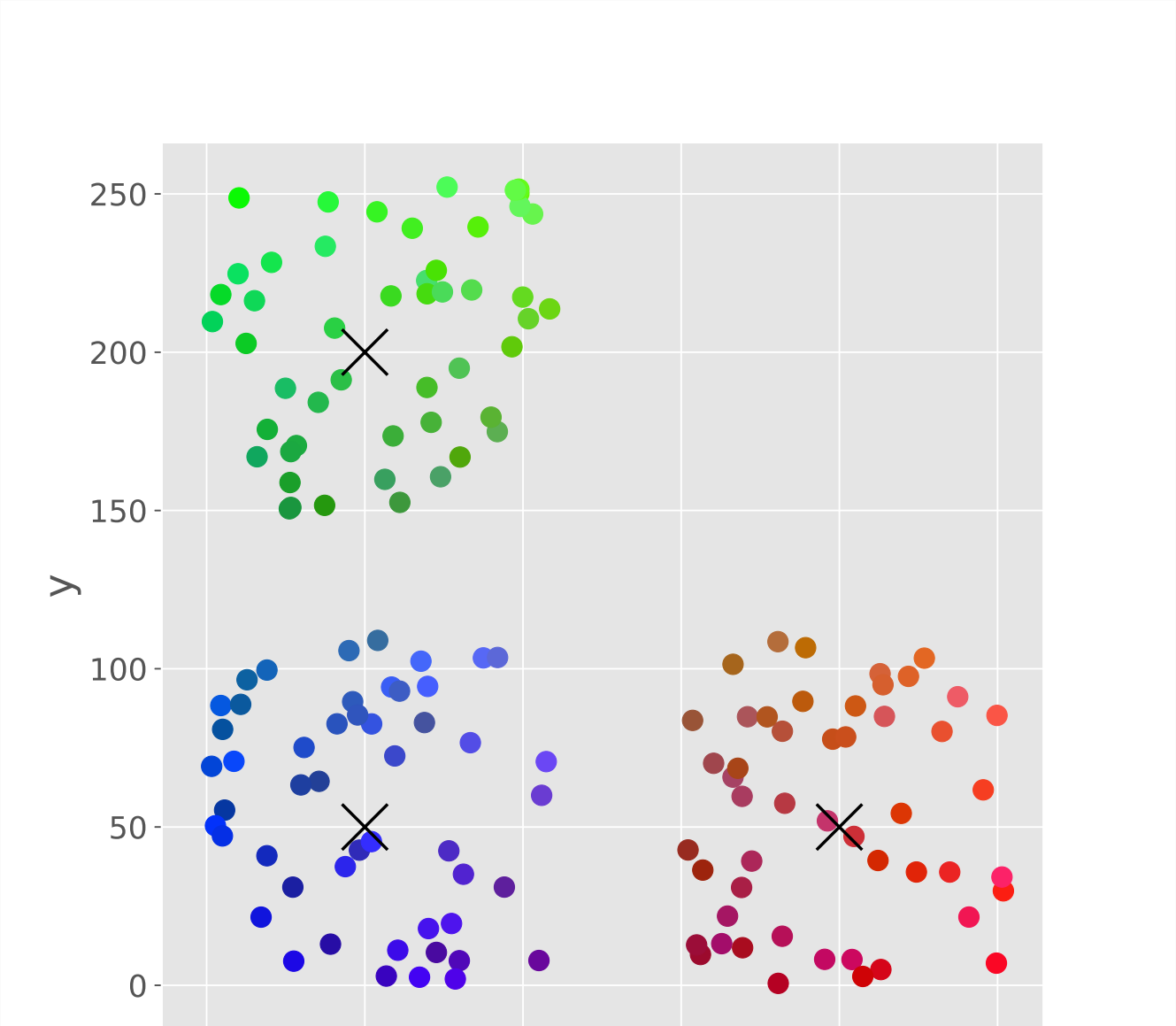
k-Means

O centro ótimo para um conjunto de pontos X é a média desses pontos.

$$c = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x_i}$$

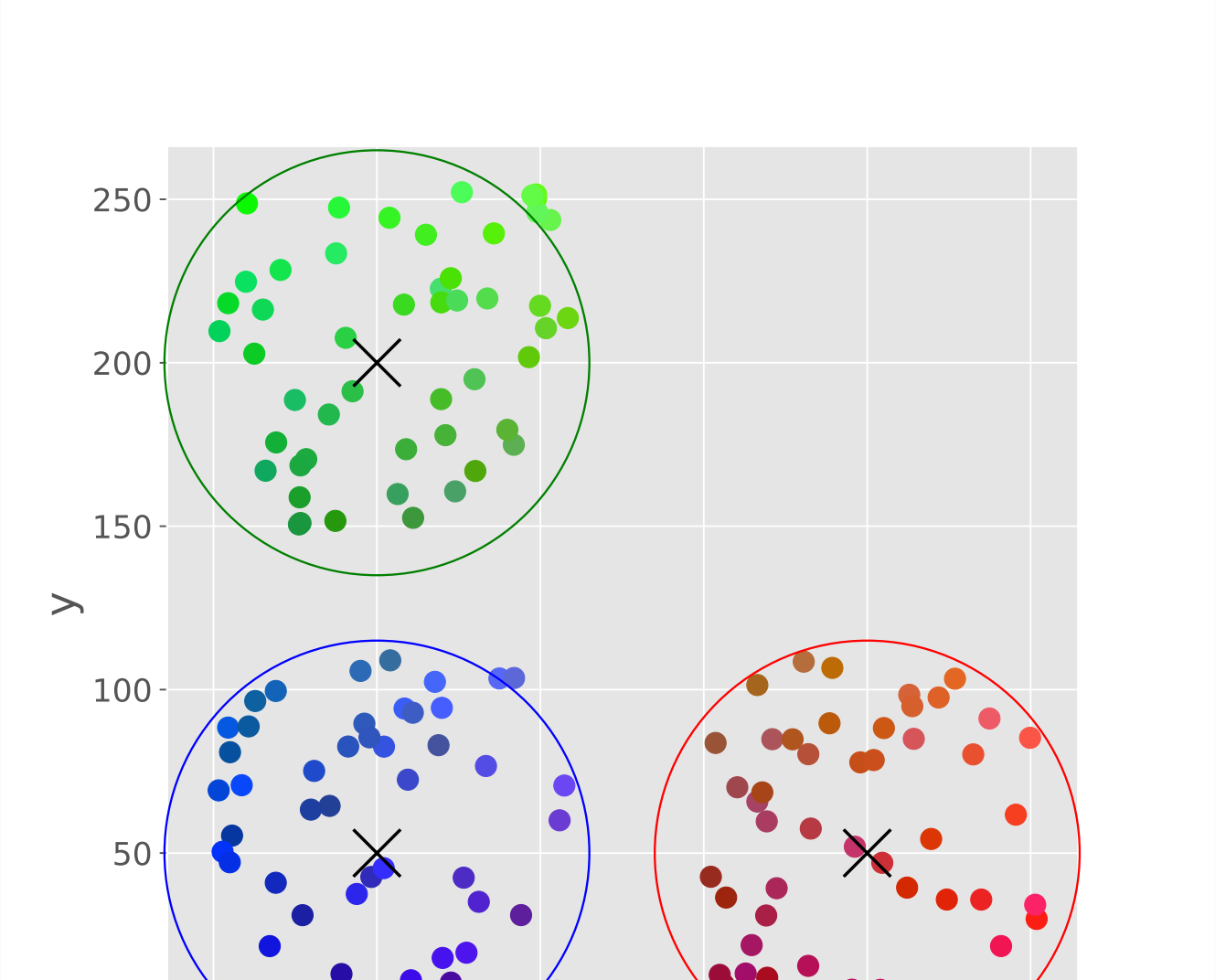
k-Means

Temos os centros…como definir os grupos?



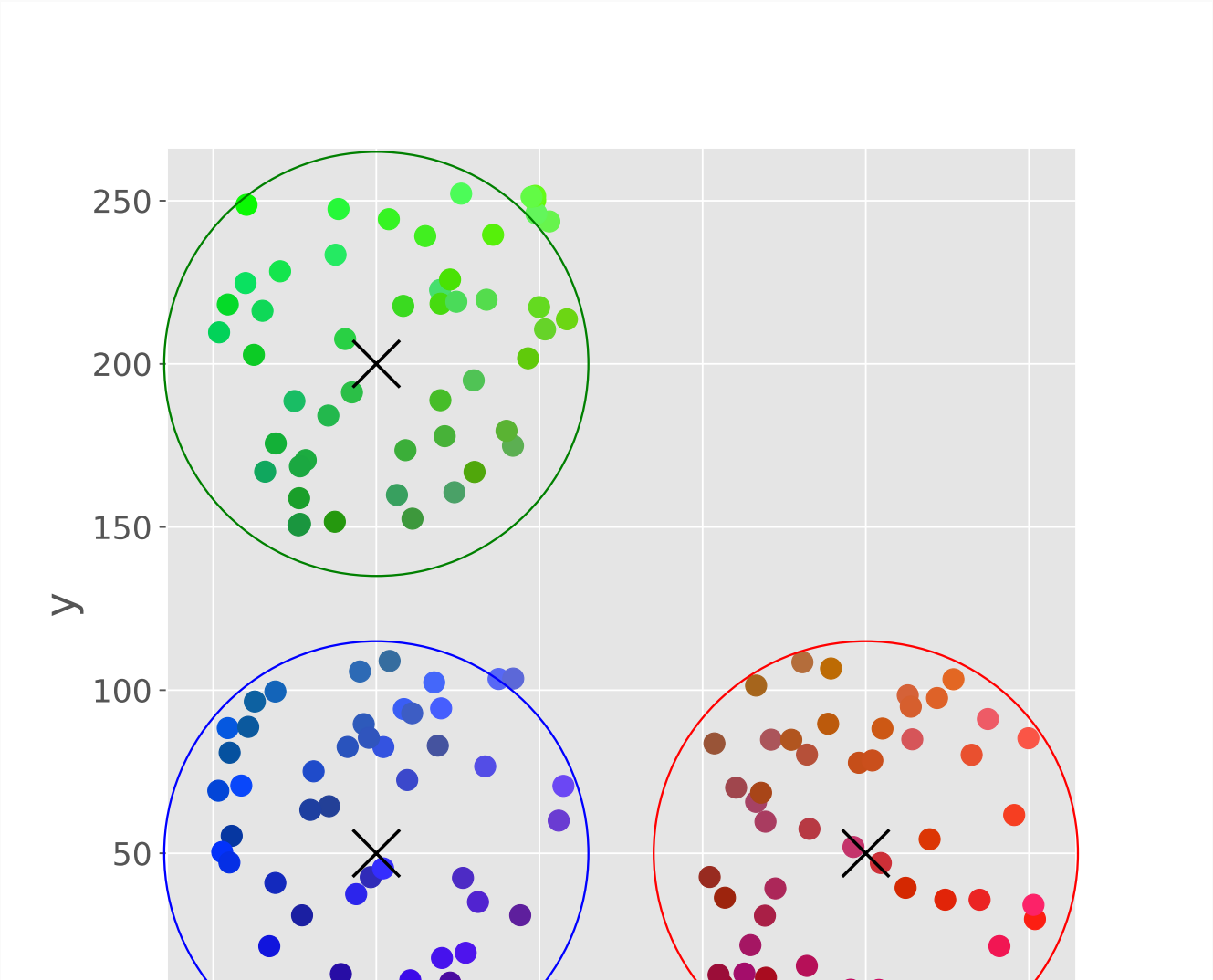
k-Means

Para cada amostra x, verificamos o centro mais próximo…essa amostra fará parte desse grupo.



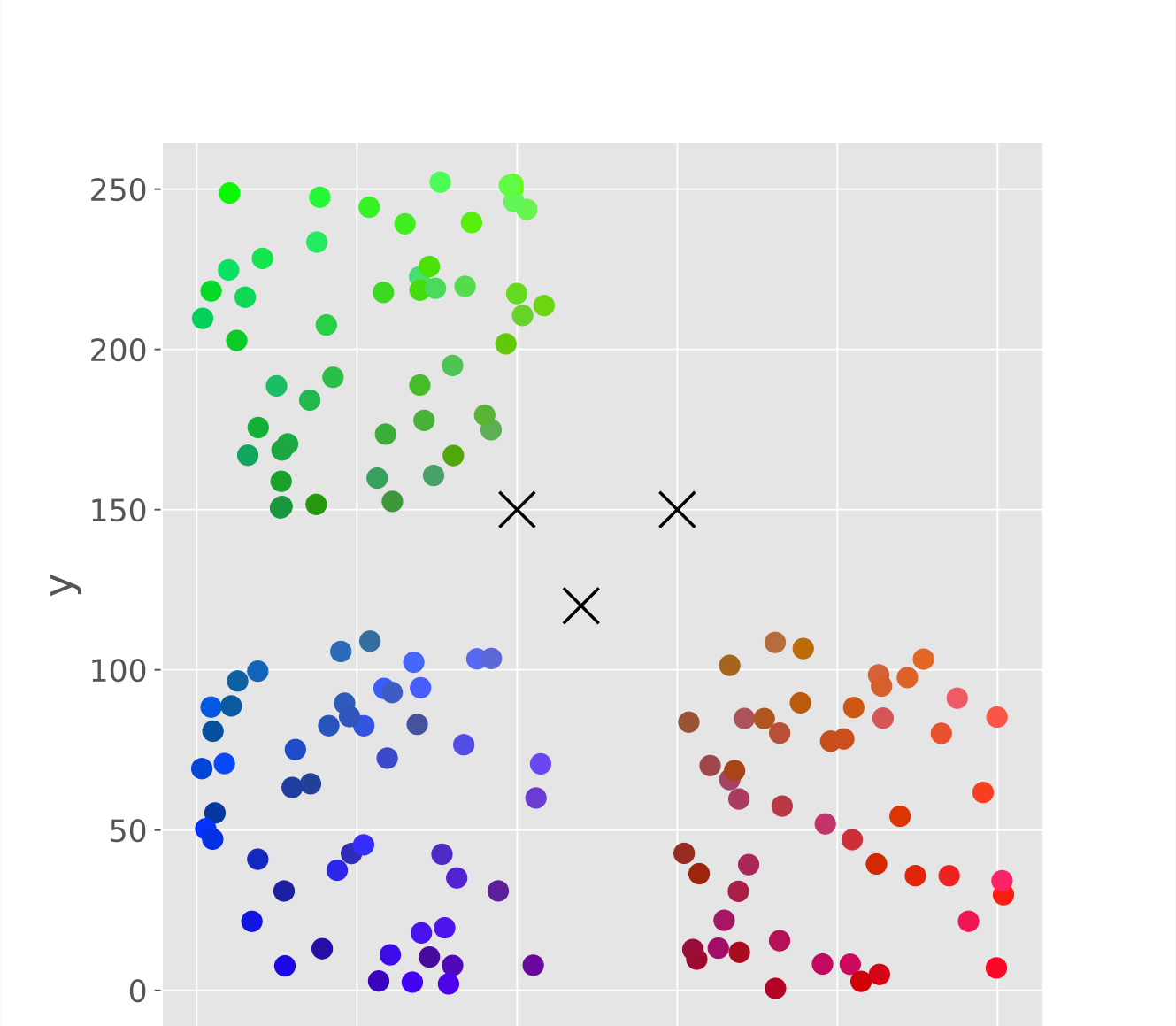
k-Means

Sabemos como calcular o centro, dado os grupos…e como definir os grupos dado os centros.



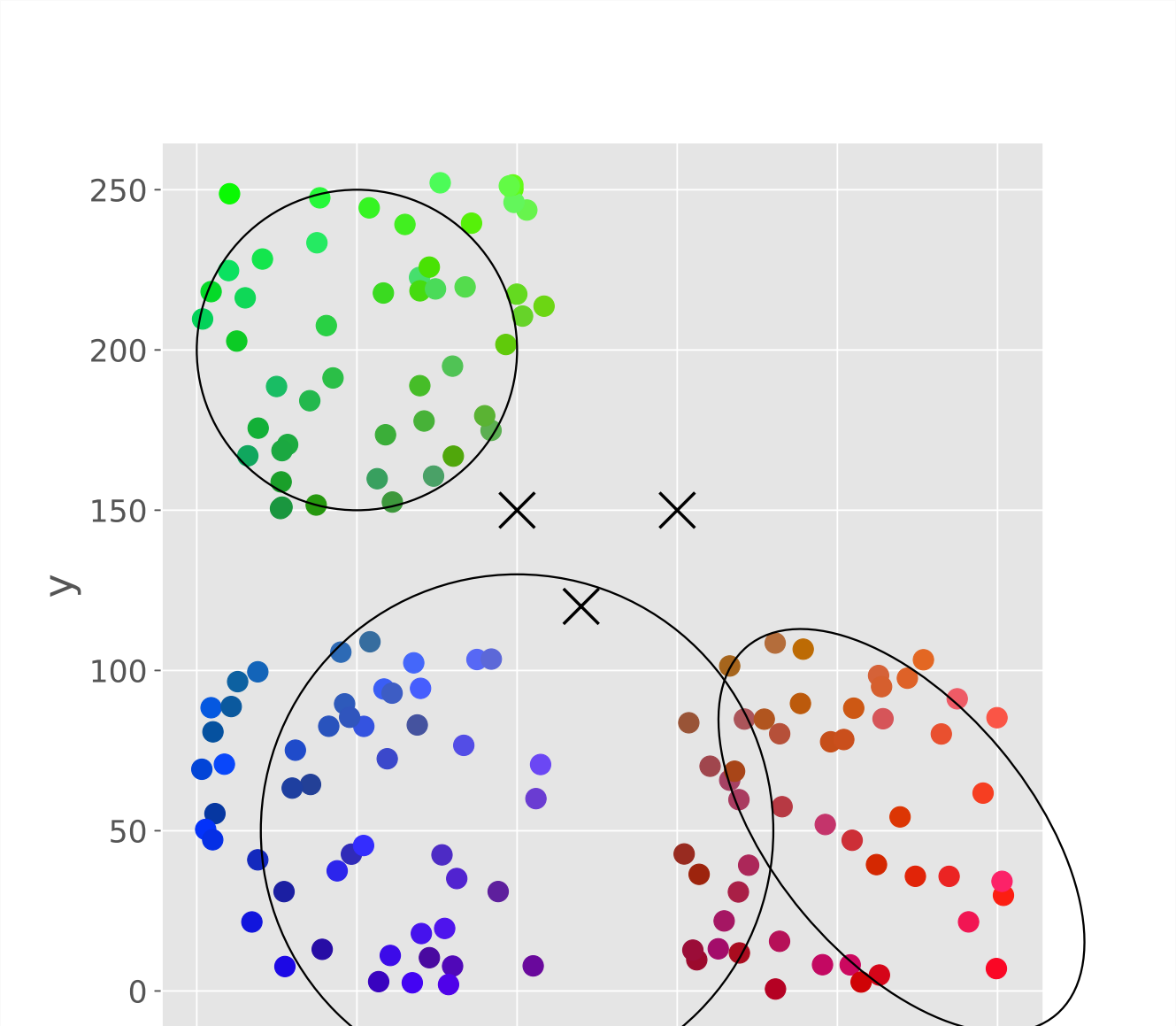
k-Means

Começamos chutando pontos iniciais para os centros.



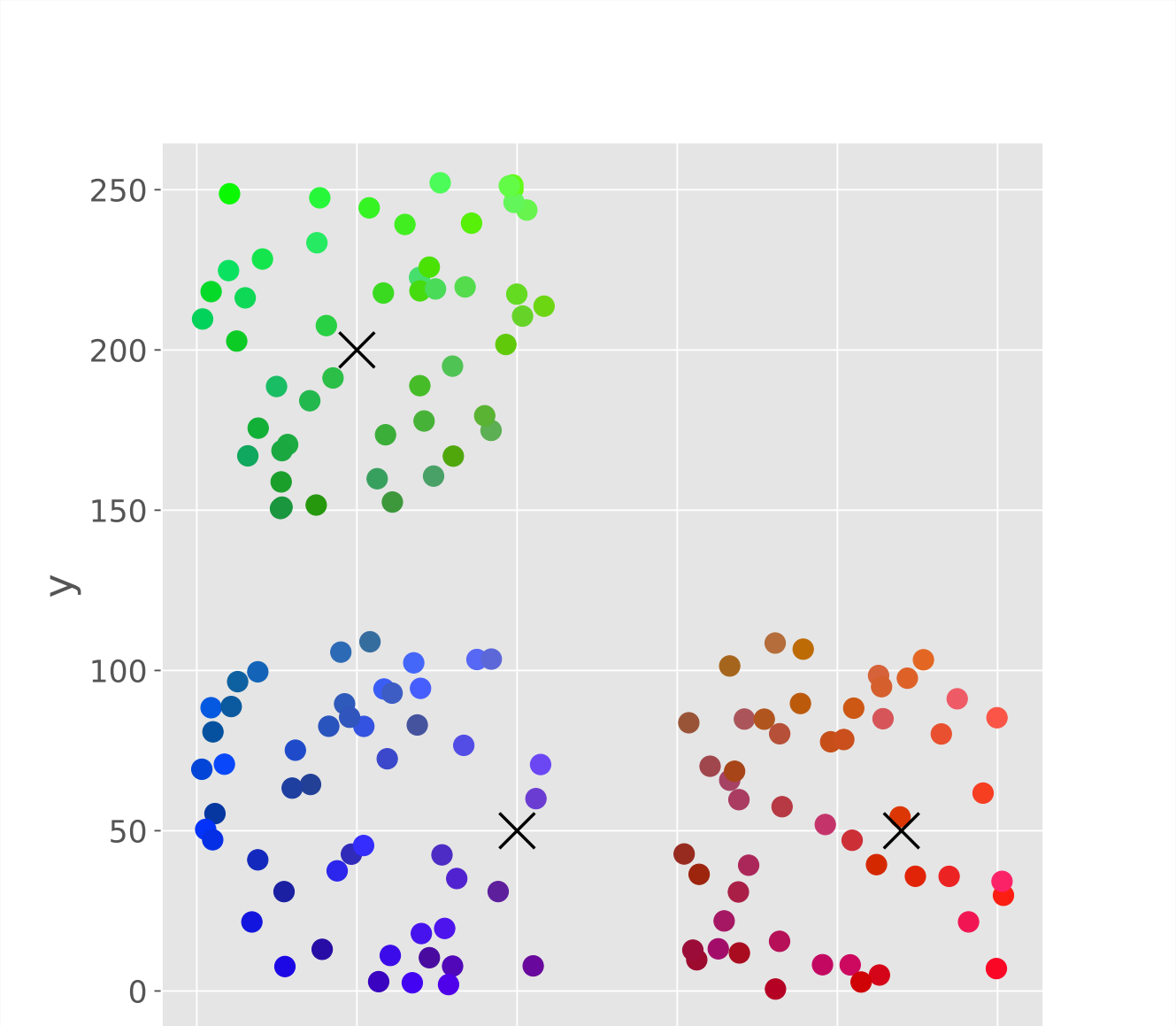
k-Means

Definimos os grupos de acordo com esses centros:



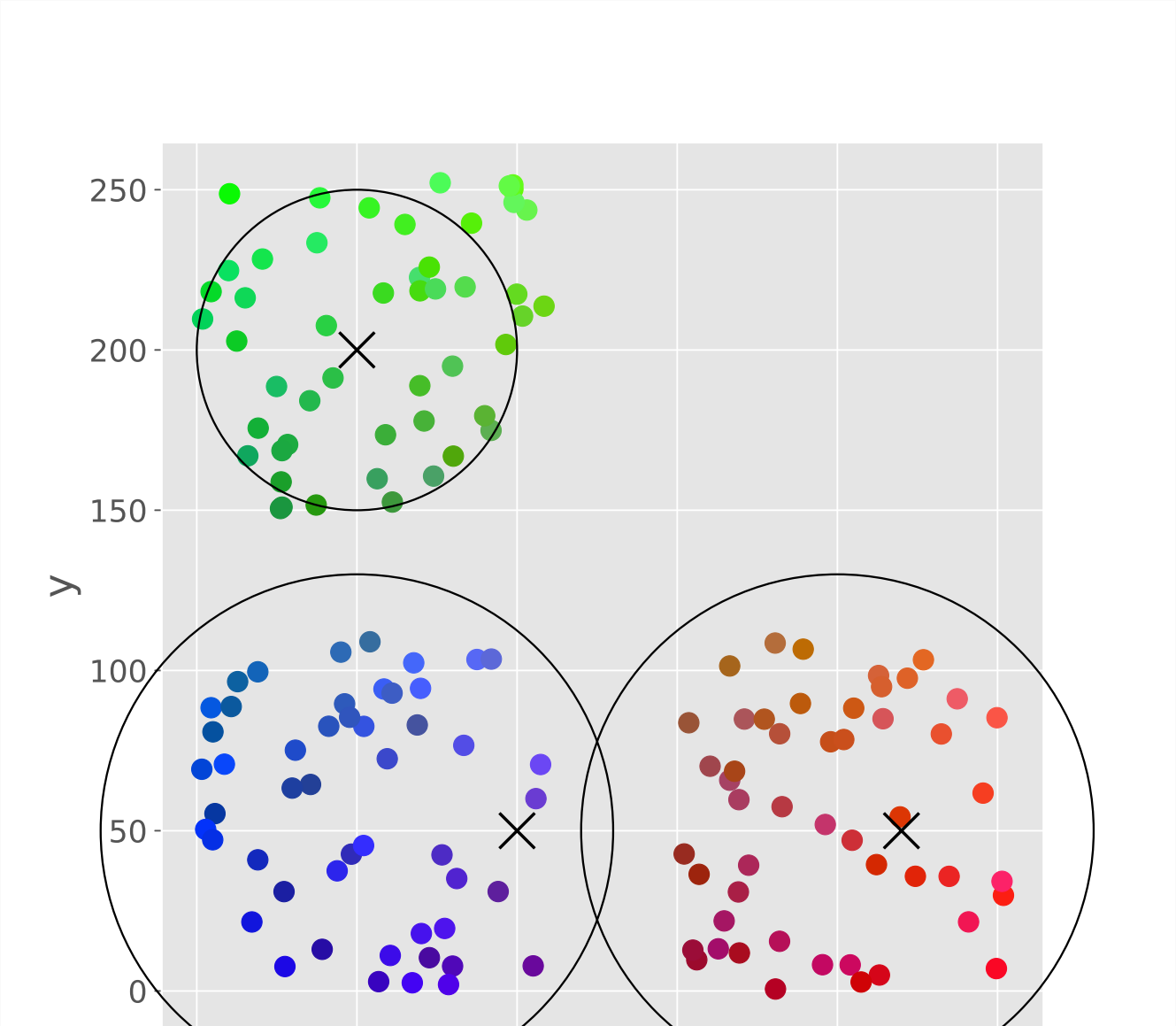
k-Means

Para cada grupo, calculamos o novo centro:



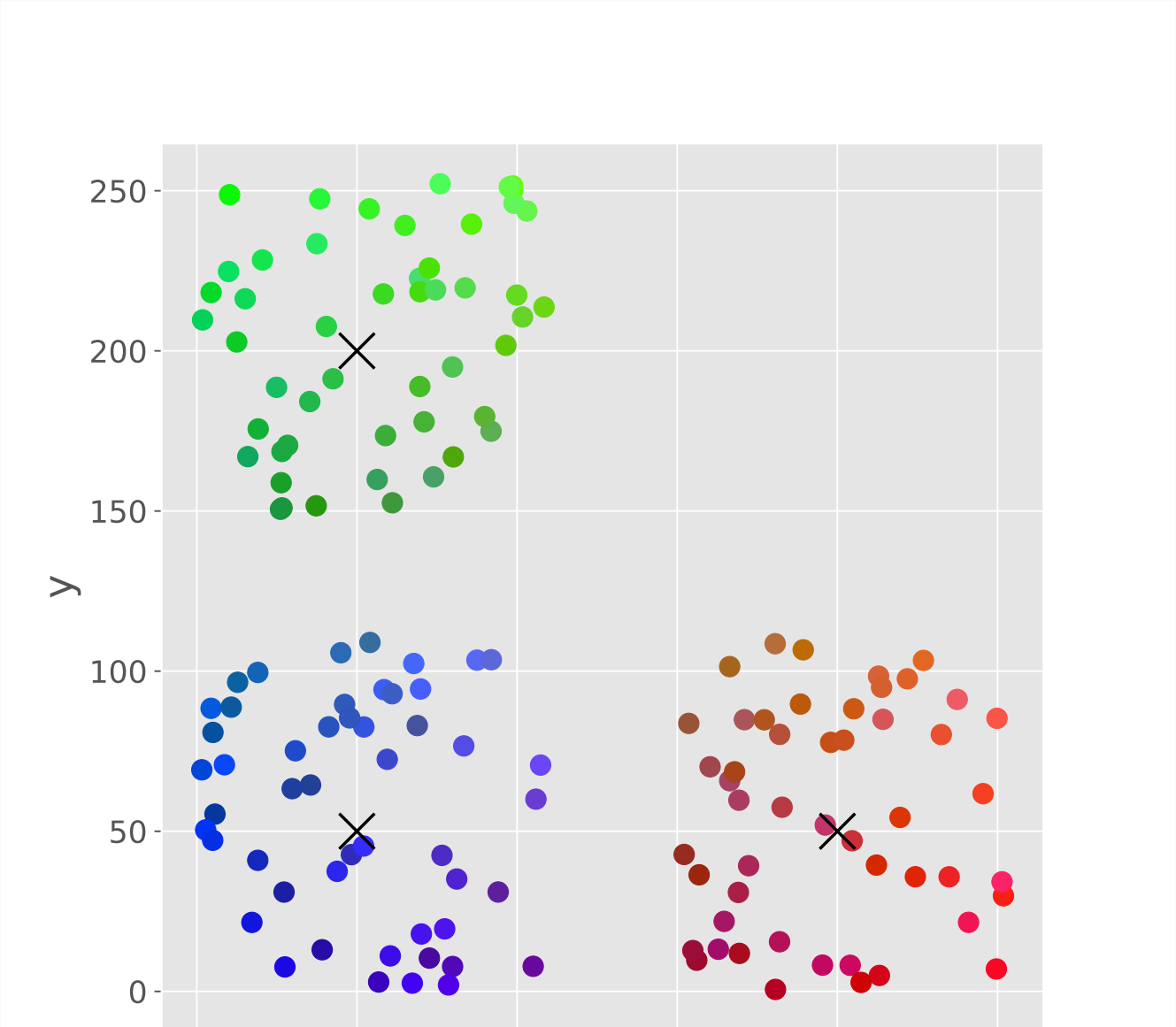
k-Means

E remontamos os grupos:



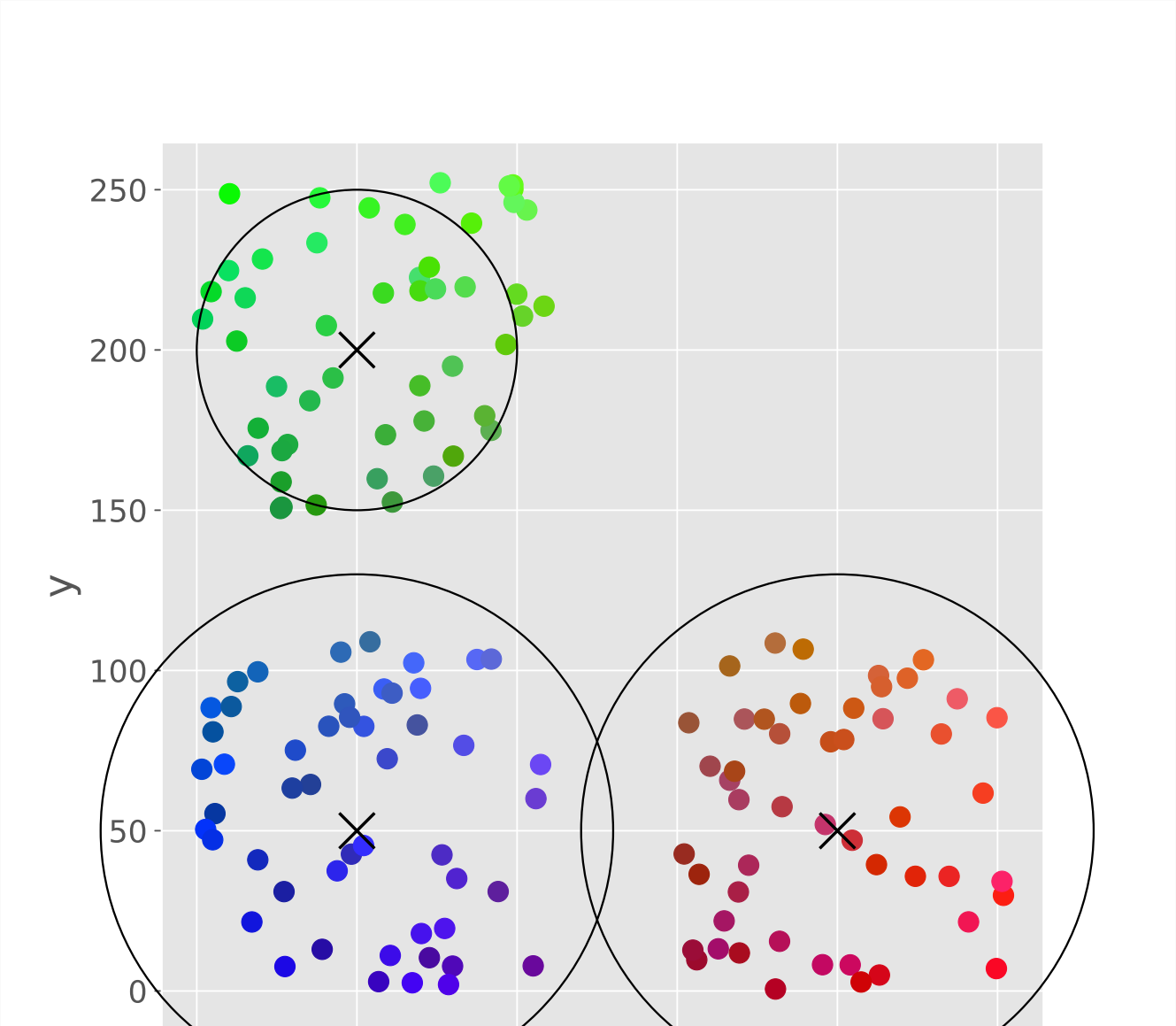
k-Means

Repetimos esse procedimento…



k-Means

…até os centros ou os grupos não se alterarem mais.



** k-Means

```haskell
kmeans it points clusters
  | it == 0                = clusters
  | clusters' == clusters = clusters
  | otherwise              = kmeans (it-1) points
  ↪ clusters'
  where
    clusters' = emStep points clusters
```

k-Means

```haskell
emStep points clusters = maximizationStep
                       $ estimationStep points clusters
```

k-Means

```haskell
estimationStep :: [[Double]] -> [[Double]]
               -> [[[Double]]]
estimationStep points clusters = map (\(k,v) -> v)
                               $ groupByKey $ sortByKey
                               $ assign points clusters
```

k-Means

```haskell
assign points clusters = map closestTo points
  where
    closestTo p = (argmin $ map (euclid p) clusters, p)
```

k-Means

```haskell
maximizationStep :: [[[Double]]] -> [[Double]]
maximizationStep candidates = map meanVec candidates
```

k-Means - Paralelo

Temos dois procedimentos a serem realizados em cada iteração:

- Alocar os elementos a um centro.
- Calcular um novo centro.

Considere que os centros residem em memória.

k-Means - Paralelo

```
centers = RDD.map(lambda xi: (assign(xi,c), xi))
             .reduceByKey(lambda (pi1, n1), (pi2, n2)
                                  : (pi1+pi2, n1+n2))
             .map(lambda (idx, (pi, n)): (pi/n))
             .collect()
```