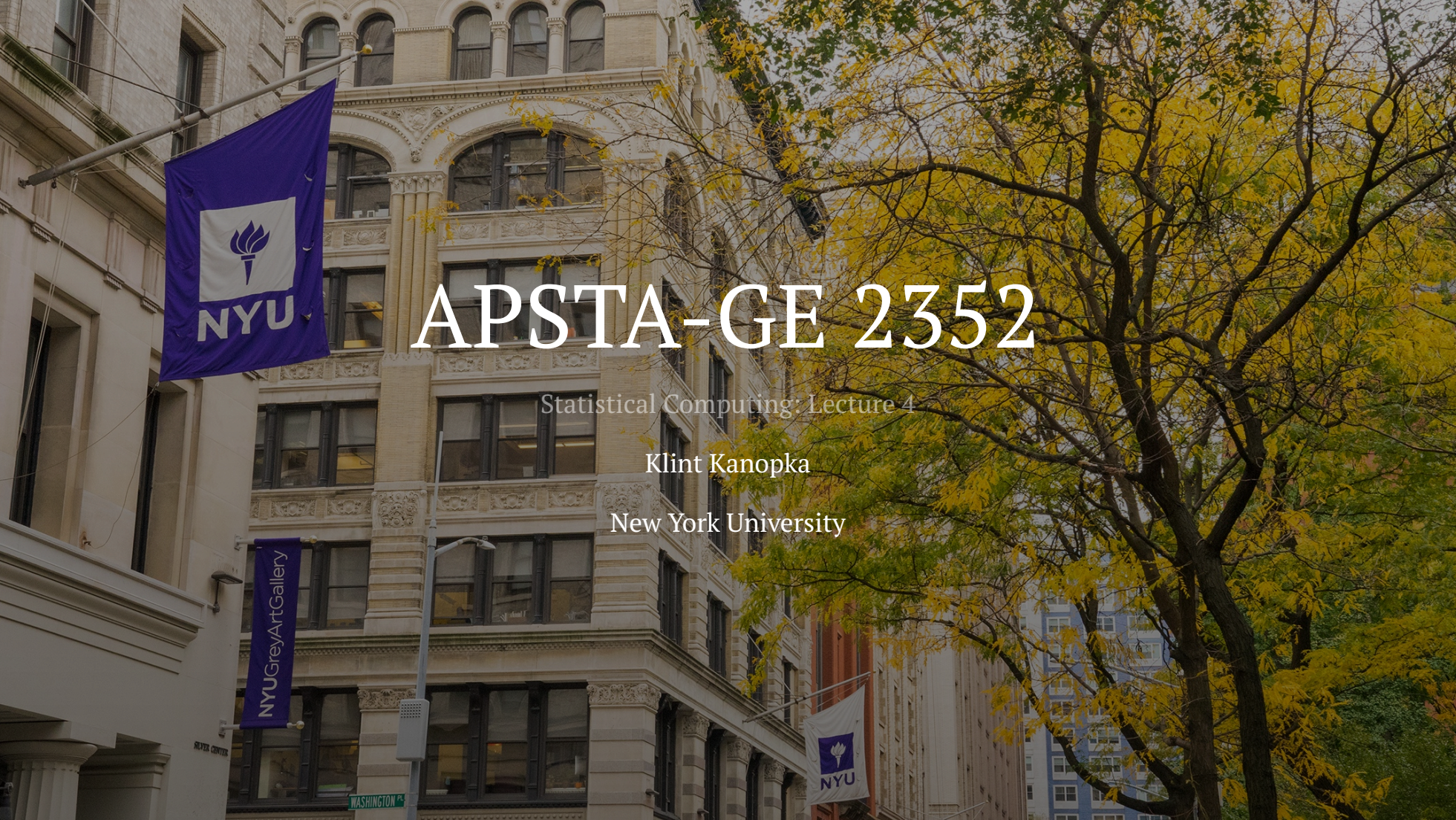


APSTA-GE 2352

Statistical Computing: Lecture 4

Klint Kanopka

New York University

Table of Contents

Announcements

- PS1 Grades released
  - Correlation between PS0 and PS1 grades was low $(r \approx 0.28)$
  - Mean score was 7.7pts higher on PS1 than PS0
  - Seems fine to me
- Answer keys for both PS0 and PS1 are posted in the Week 1 materials
- PS2 is due next week before class

Total PS Points Needed by Desired Grade

Grade	PS Points
A	752
A-	707
B+	680
B	645

Note: this does not account for the final exam!

Check-In

- PollEv.com/klintkanopka

Motivating Problem

Making Groups from Data

- Often called "clustering"
- Big idea:
  - Observations in your dataset belong to *latent* groups
    - *Latent* means unobservable
  - The nature and number of groups is unobservable
  - The assignment of each point to a group is unobservable
  - We also need to figure all of this out at once
- Conceptually, the clustering task takes observations that are "close together" and puts them into groups
- Typically there are three ways to do clustering:
  - *Iterative*: Pick a number of clusters and rearrange them until you get your "best fit"
  - *Agglomerative*: Build up clusters by sticking nearby points together
  - *Divisive*: Start with one big group and repeatedly cut it into pieces

Making Groups from Data

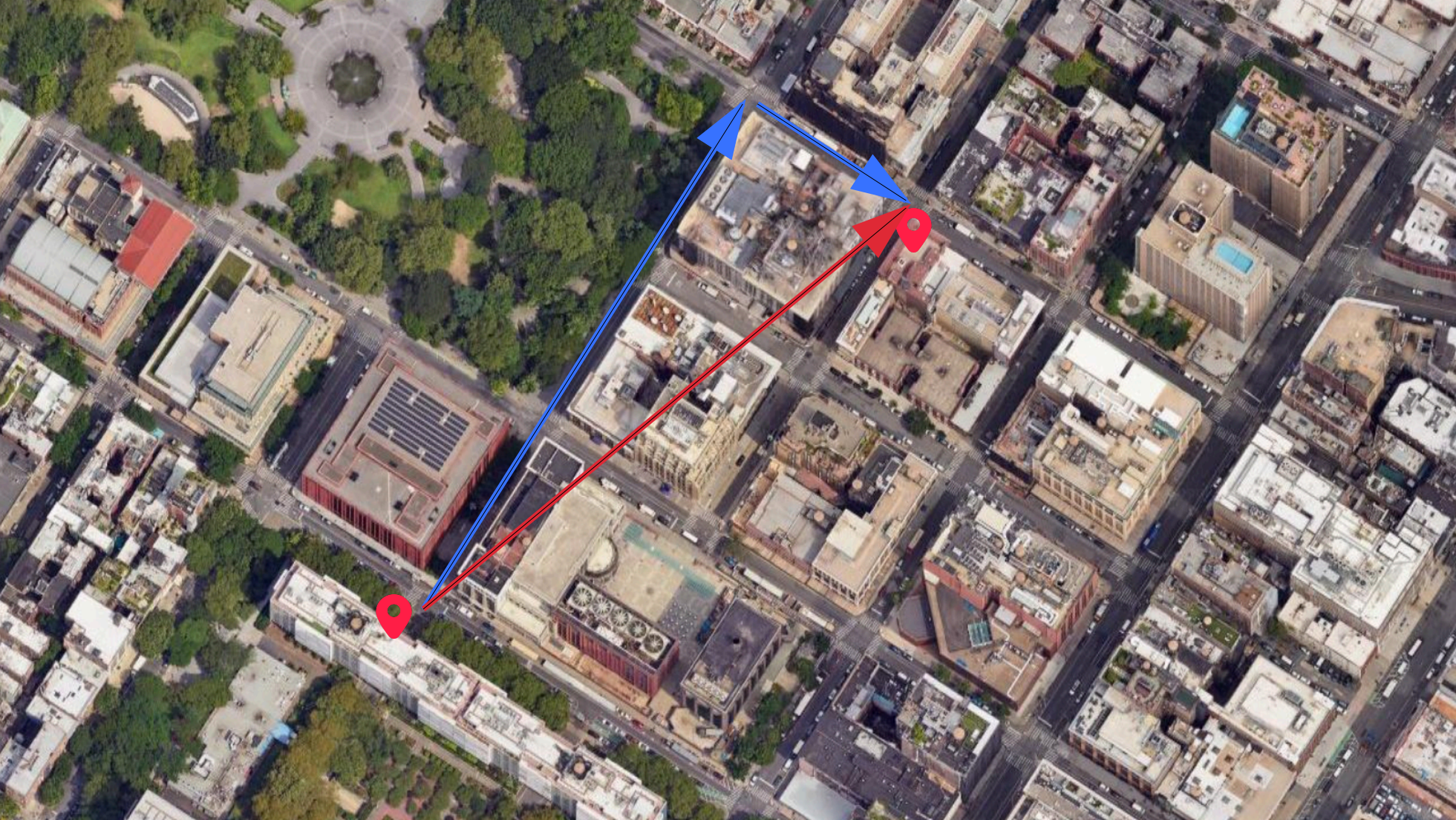
- This is different from building a model to predict known group membership and classifying new observations
  - This type of problem is called "supervised learning"
  - You can train a model to predict a known outcome and check how well it does
- Clustering is, on the other hand, an "unsupervised learning" problem
  - You don’t know that there really are groups
  - If there are groups, you don’t know how many groups there should be
  - You don’t even know that the groups you find are the "right" groups
- This makes clustering really tricky task; you can’t check your work!

Tools

Measuring Distances

Measuring Distances

- If you want to talk about how "close together" two things are, you need some notion of distance
- Distances are *dissimilarity* metrics
  - Larger magnitudes mean less similar (or farther apart) objects
  - Assign a value of 0 to identical objects
- Contrast these with *similarity* metrics
  - Larger magnitudes mean more similar (or closer together) objects
  - Assign maximal values (sometimes 1) to identical objects



Measuring Distances

- Most generically, we often think about *Euclidean distance*
  - Sometimes called "as the crow flies"
  - It’s straight line distance

$$d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

$$d(p, q) = \sqrt{\sum_i (p_i - q_i)^2}$$

Measuring Distances

- Depending on the situation, something else might make more sense!
- Another spatial distance metric is *Manhattan distance*
- It's what you might expect from the name

$$d(p, q) = |x_p - x_q| + |y_p - y_q|$$

$$d(p, q) = \sum_i |p_i - q_i|$$

`while` Loops

`while` Loops

- Recall: loops allow us to specify a chunk of code to be executed repeatedly
  - `for` loops execute a pre-specified number of times
- What if you don't know how many times a loop will take, but you know when you want it to stop?
  - This is the job of the `while` loop
  - When possible, prefer `for` loops to `while` loops!

`while` Loops

- In `R`, a `while` loop has a few main components
  - The call: `while`
  - The condition, specified in `( )` after the call and contain:
    - A logical statement that returns a single value
  - The code, wrapped in `{ }`:
    - At the beginning of the loop, the condition is checked
    - If it evaluates to `TRUE`, all of the code in the loop is executed
    - If it evaluates to `FALSE`, the code in the loop is skipped
    - After this is done, the condition is checked again to decide if the loop should be repeated
    - This continues until the condition evaluates to `FALSE`

Example `while` Loops 1 and 2

- What will be the result of executing these loops?

Example `while` Loops 1 and 2

- What will be the result of executing these loops?

```
i <- 0

while (i <= 5){
  print(i)
}

# [1] 0
# [1] 0
# [1] 0
# [1] 0
# [1] 0

# this will not end...

while (i < 5){
  print(i)
}
```

Example `while` Loops 3 and 4

- What will be the result of executing these loops?

```r
i <- 0

while (i <= 5){
  print(i)
  i <- i + 1
}

# [1] 0
# [1] 1
# [1] 2
# [1] 3
# [1] 4
# [1] 5

while (i < 5){
  print(i)
  i <- i + 1
}

# nothing happens?
```

Example `while` Loop 4 (for real)

```
i <- 0

while (i < 5){
  print(i)
  i <- i + 1
}

# [1] 0
# [1] 1
# [1] 2
# [1] 3
# [1] 4

# all done!
```

Example `while` Loop 5

- You can check for convergence

```
last <- Inf
eps <- 1e-3
current <- 10

while (abs(last - current) > eps){
  last <- current
  current <- sqrt(current)
  print(current)
}

# [1] 3.162278
# [1] 1.778279
# [1] 1.333521
# [1] 1.154782
# [1] 1.074608
# [1] 1.036633
# [1] 1.018152
# [1] 1.009035
# [1] 1.004507
# [1] 1.002251
# [1] 1.001125
# [1] 1.000562
```

Example `while` Loop 6

- You can combine both methods to set a maximum number of iterations while looking for convergence

```
last <- Inf
eps <- 1e-3
max_iter <- 3
current <- 10
i <- 0

while (abs(last - current) > eps && i < max_iter){
  last <- current
  current <- sqrt(current)
  print(current)
  i <- i + 1
}

# [1] 3.162278
# [1] 1.778279
# [1] 1.333521
```

Back to Clustering

The k-Means Algorithm

Iterative Clustering using k-Means

- We specify some number of clusters, k
- Clusters are described by *centroids*, or the center point
- We want to find locations of k centroids that, when we assign our observed points to them, minimize the *within-cluster sum of squares* (or variance)
- We call it k-Means because the location of each centroid is the mean of the coordinates of the points assigned to it!

Iterative Clustering using k-Means

- We follow a two step process:
  1. Assign each point to a cluster by finding the nearest centroid
  2. Move each centroid to the middle of the points assigned to it
- Do this over and over until *convergence*
  - An algorithm *converges* when additional iterations fail to improve the solution
  - For k-Means, convergence is when the centroids stop moving
  - What tool that we have used is a good fit for this task?
  - `while()` loops!

The k-Means Algorithm

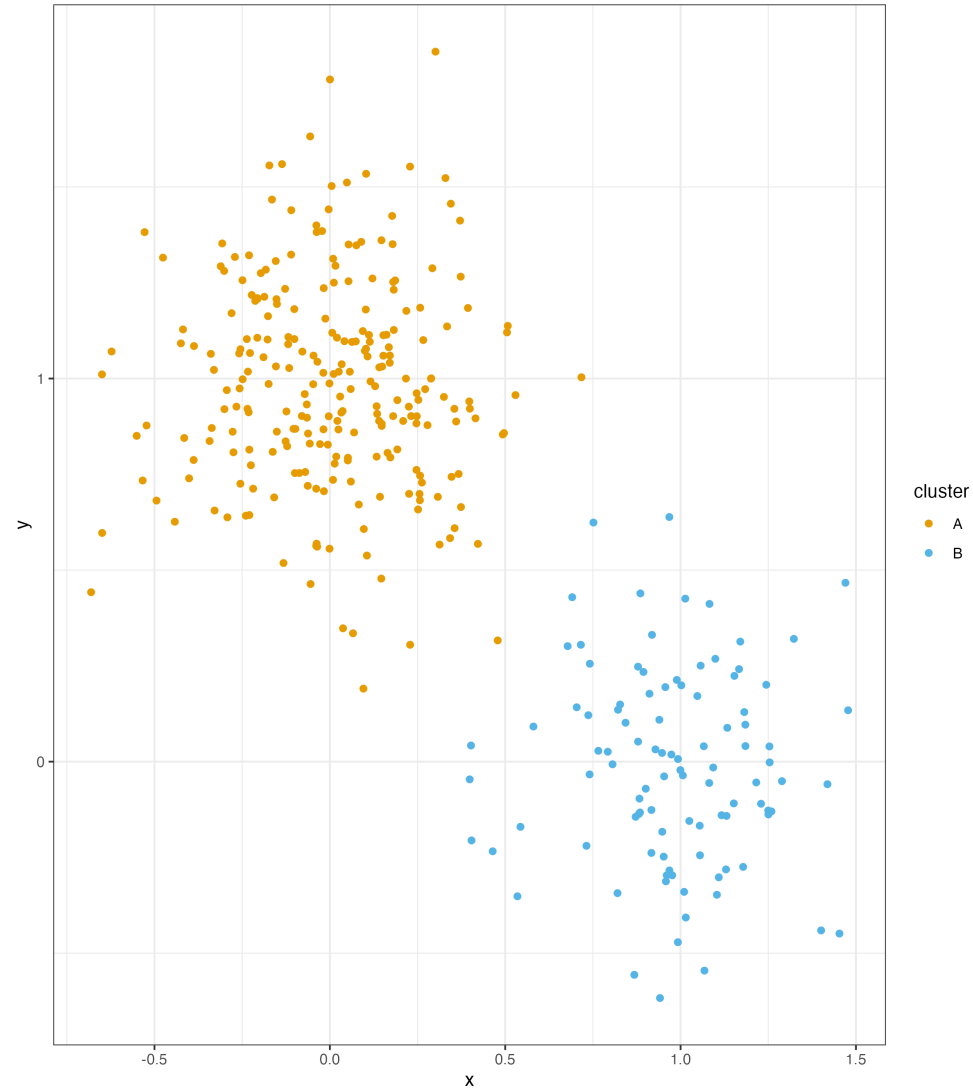
1. Select a number of clusters, k, and initialize the center of each cluster to a different point in space
2. Assign each observation from the dataset its nearest centroid using the Euclidean distance
3. Move the center of each cluster to the mean value of the coordinates of the points assigned to it
4. Repeat steps 2&3 until the centers stop moving

Generating Clustered Data

```
set.seed(8675309)
d <- data.frame(x = c(rnorm(250, 0, 0.25),
                      rnorm(100, 1, 0.25)),
                y = c(rnorm(250, 1, 0.25),
                      rnorm(100, 0, 0.25)),
                cluster = c(rep('A', 250),
                            rep('B', 100)))
```

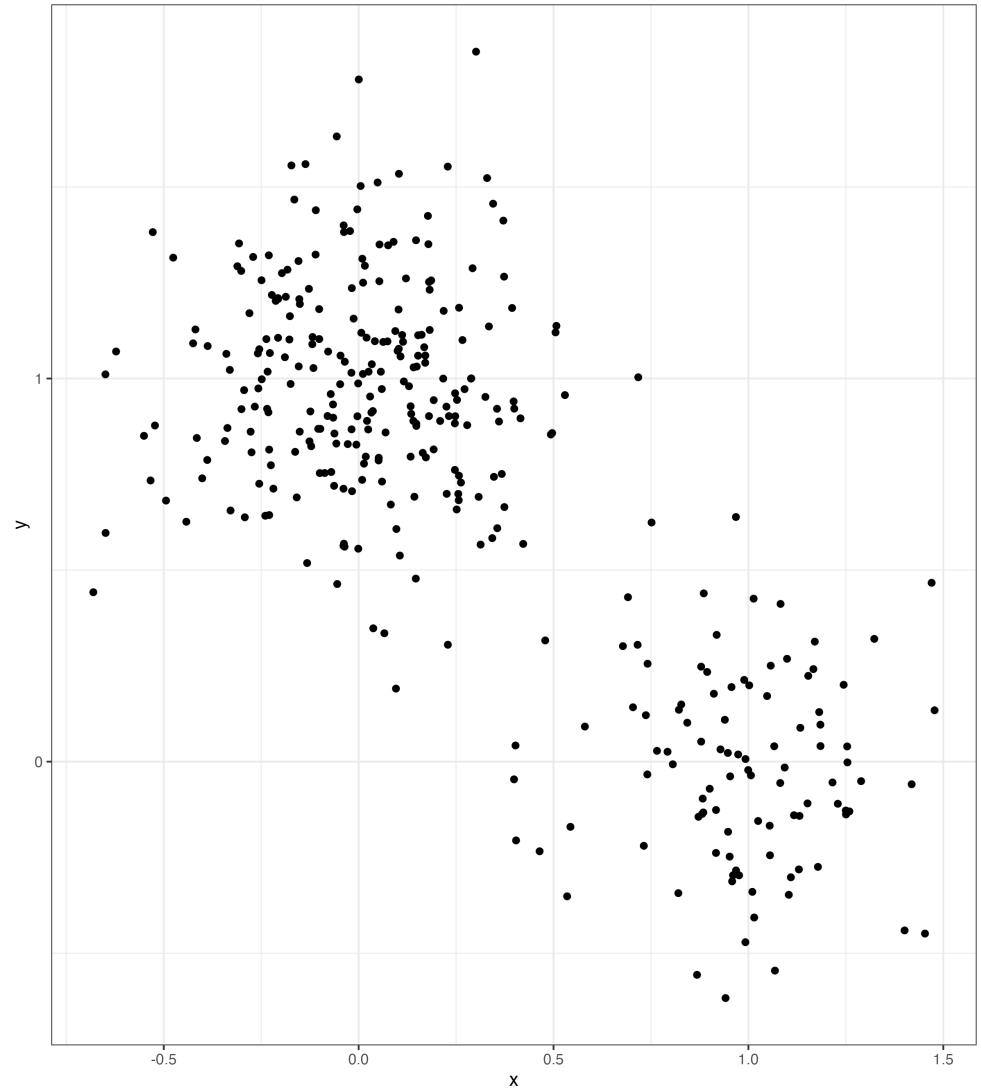
Visualizing Clustered Data

```
ggplot(d, aes(x=x, y=y, color=cluster)) +
  geom_point() +
  scale_color_okabeito() +
  theme_bw()
```



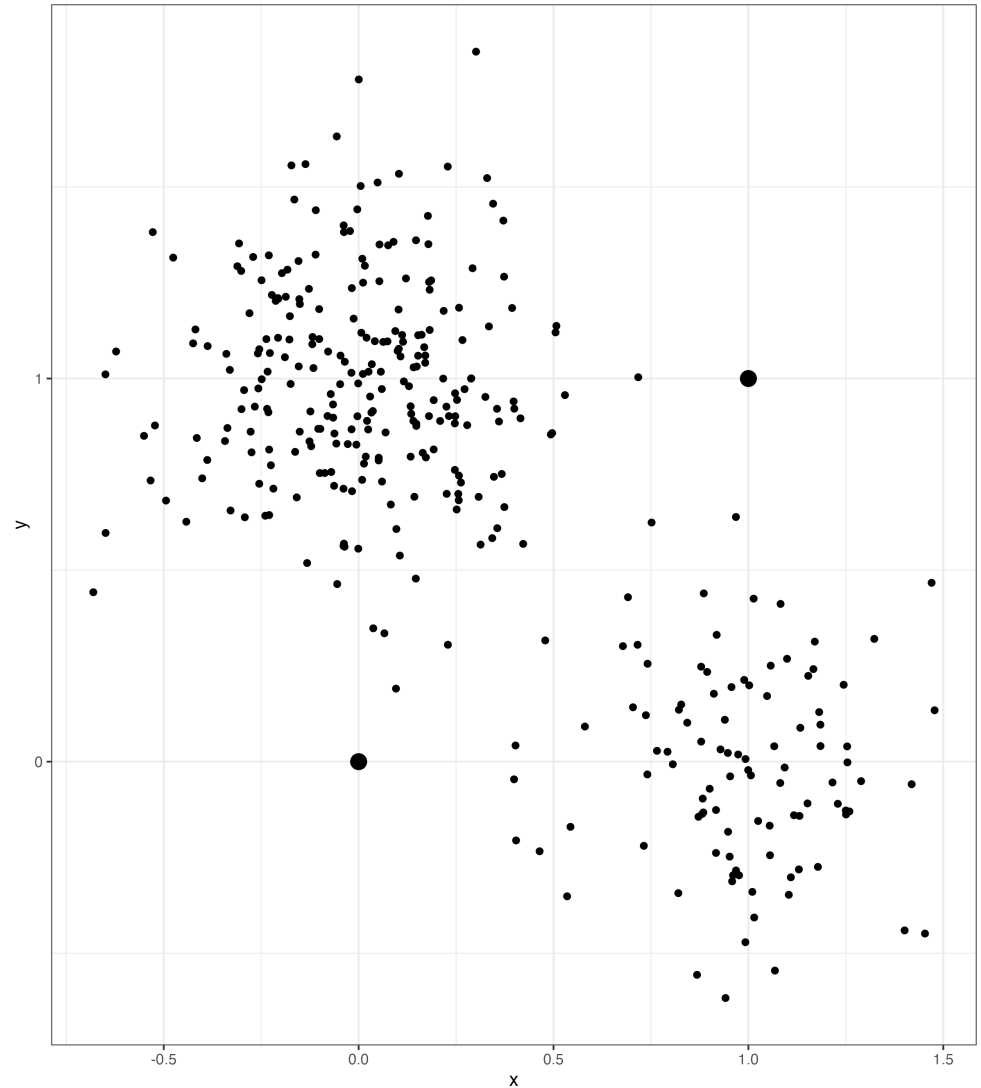
Visualizing Clustered Data

```
ggplot(d, aes(x=x, y=y)) +
  geom_point() +
  theme_bw()
```



Generate and Plot Starting Centroids

```
centroids <- data.frame(x = c(0,1),
                        y = c(0,1))


ggplot(d, aes(x=x, y=y)) +
  geom_point() +
  geom_point(data=centroids, aes(x=x, y=y),
             color="black", size=4) +
  theme_bw()
```

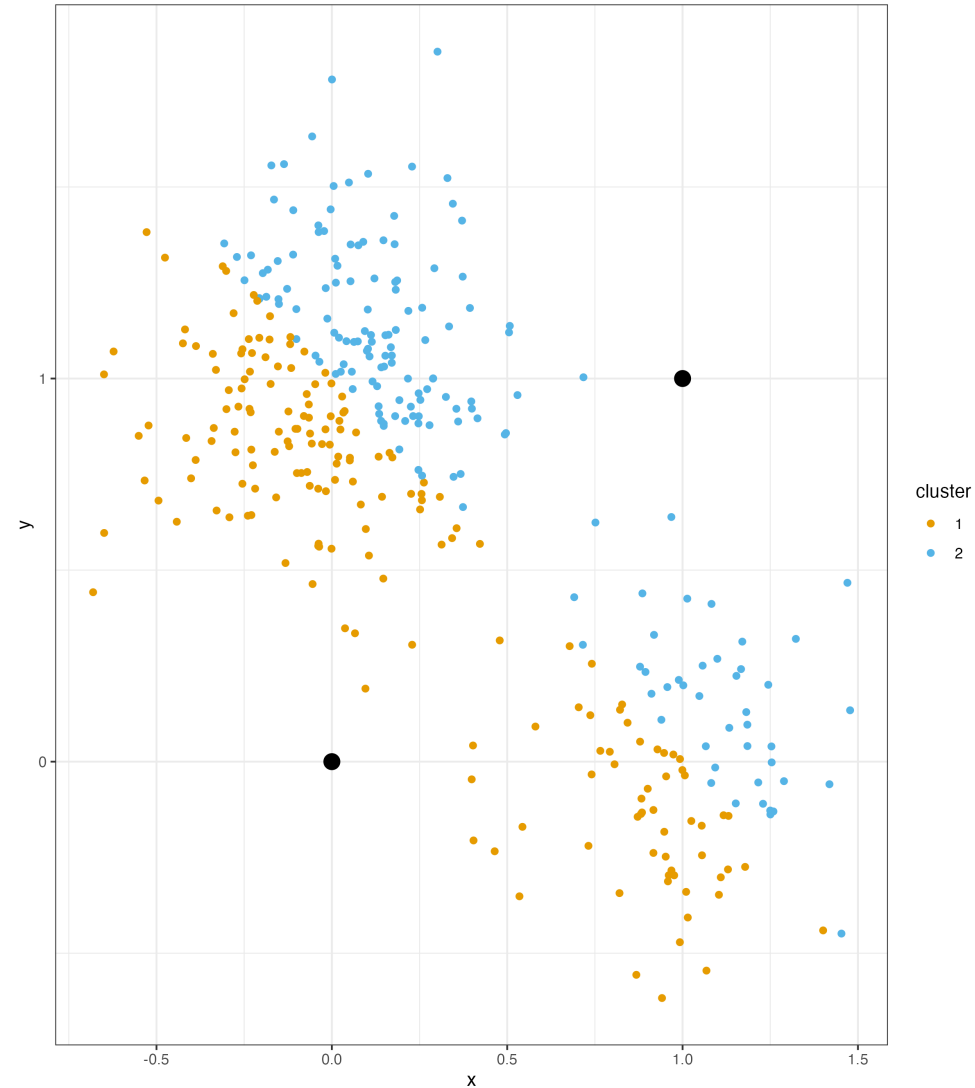


Compute Distances from Each Point to a Centroid

```r
GetDistances <- function(data, centroid){
  # TODO: Compute distance from each point in data
  #  to a single centroid

  distances <- sqrt((data[['x']] - centroid[['x']])^2 +
                      (data[['y']] - centroid[['y']])^2)

  return(distances)
}

GetDistances(d, centroids[2,])[1:5]

# [1] 1.2752070 0.8253962 1.1943523 0.5114553 0.7404843
```

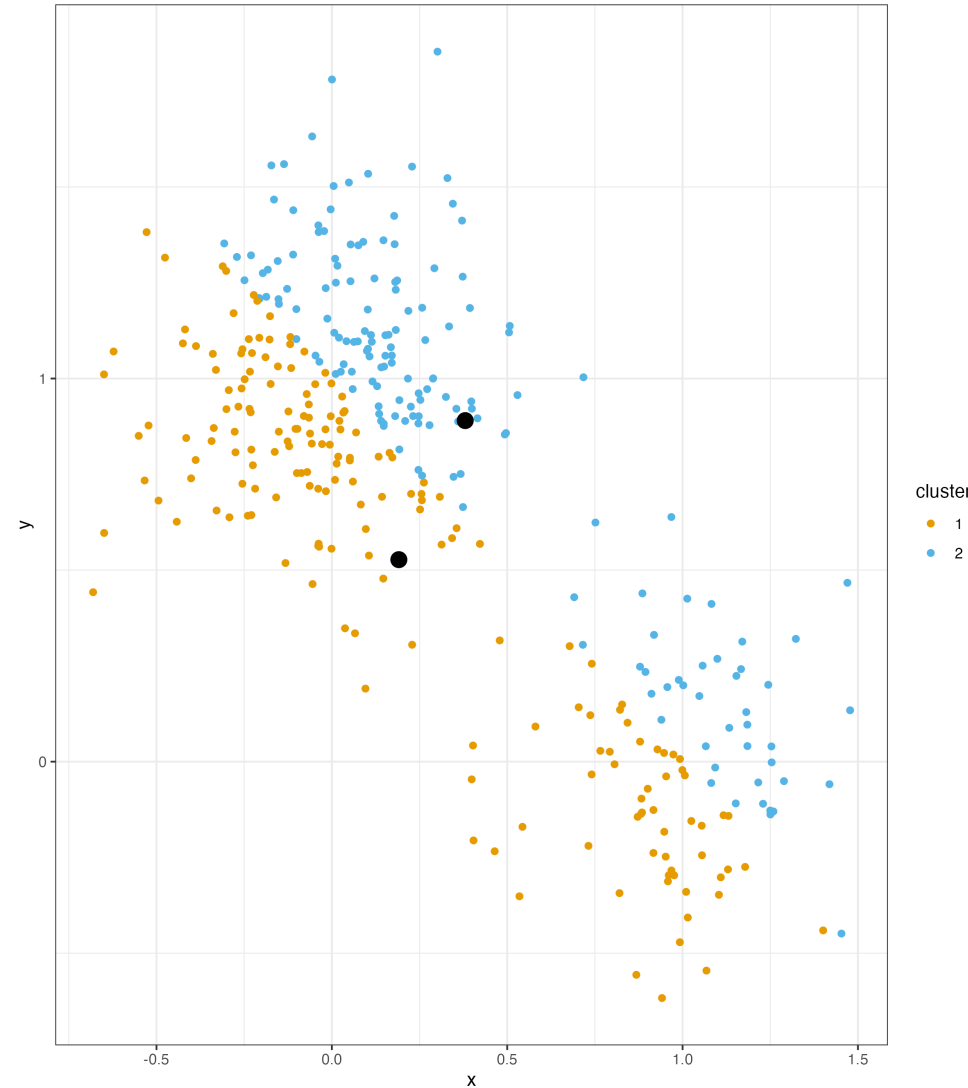
Assign Each Point to the Nearest Centroid

```
d$assignment <-
  ifelse(GetDistances(d, centroids[1,]) <=
           GetDistances(d, centroids[2,]),
         1, 2)

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



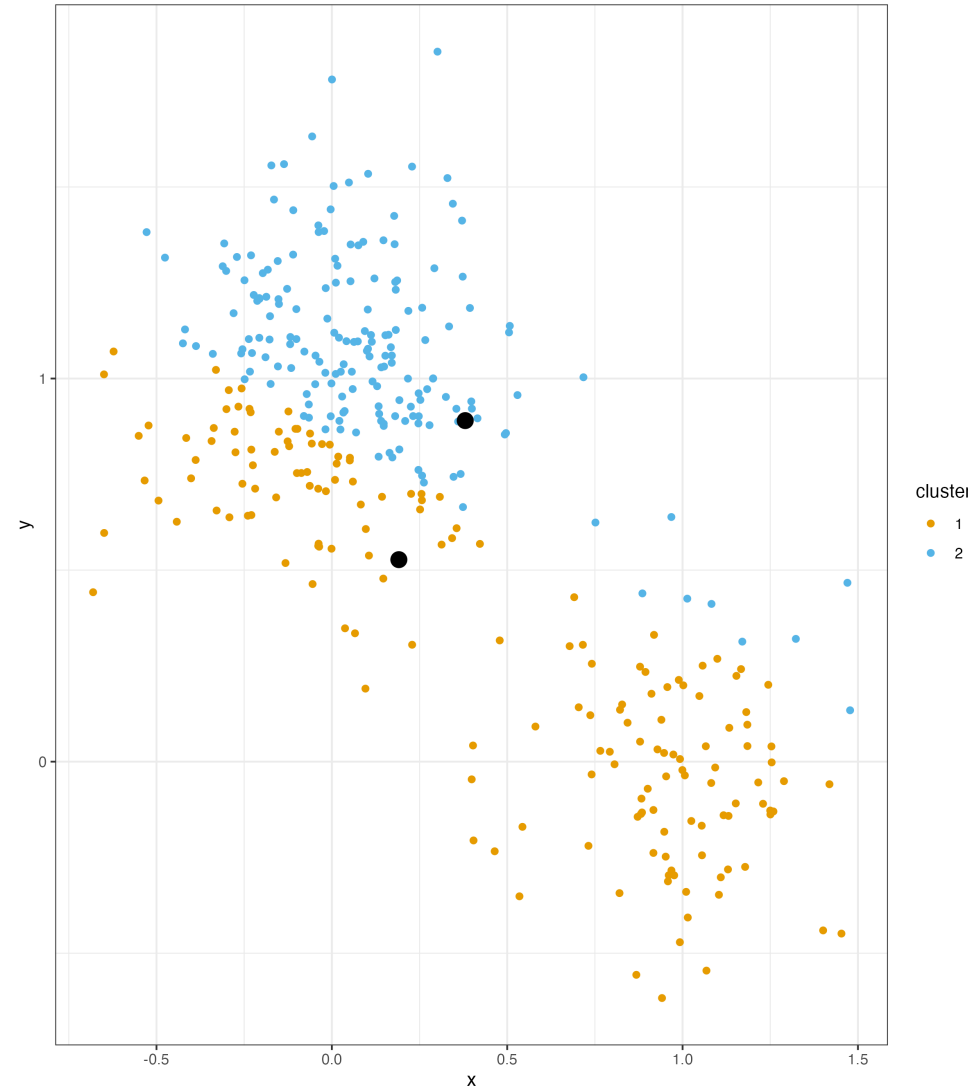
Move Centroids to the Mean of their Cluster

```
centroids <-
  data.frame(x = c(mean(d$x[d$assignment == 1]
                   mean(d$x[d$assignment == 2]
             y = c(mean(d$y[d$assignment == 1]
                   mean(d$y[d$assignment == 2]

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



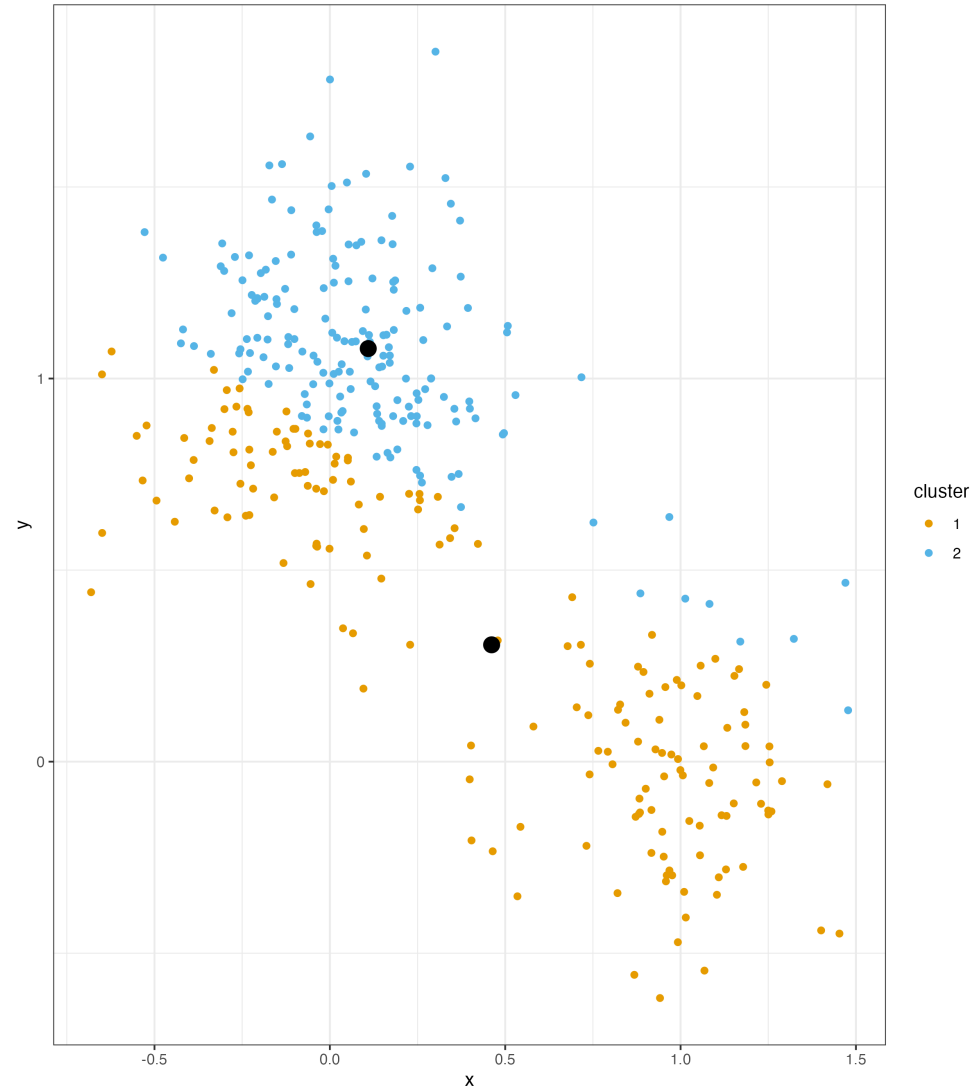
Second Iteration: Assign Points

```
d$assignment <-
  ifelse(GetDistances(d, centroids[1,]) <=
           GetDistances(d, centroids[2,]),
         1, 2)

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



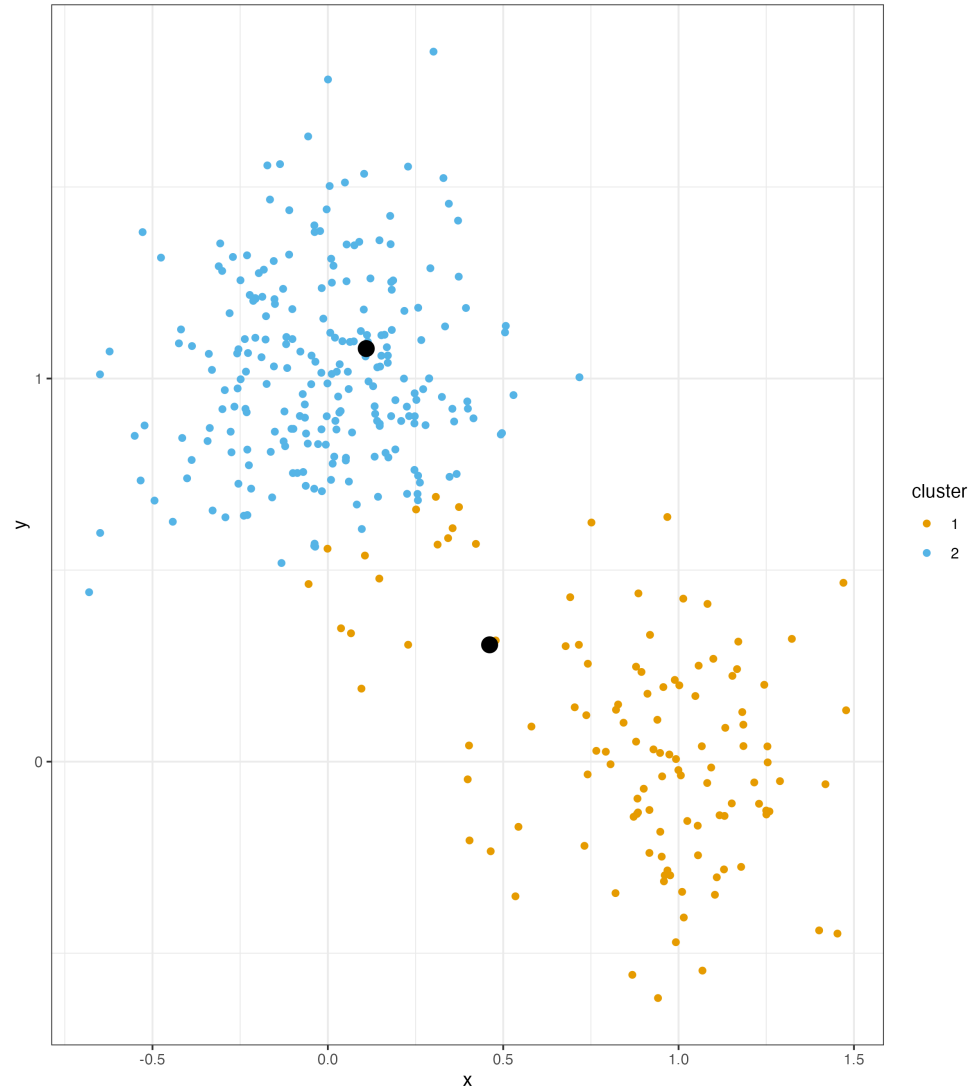
Second Iteration: Move Centroids

```r
centroids <-
  data.frame(x = c(mean(d$x[d$assignment == 1]
                   mean(d$x[d$assignment == 2]
             y = c(mean(d$y[d$assignment == 1]
                   mean(d$y[d$assignment == 2]

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



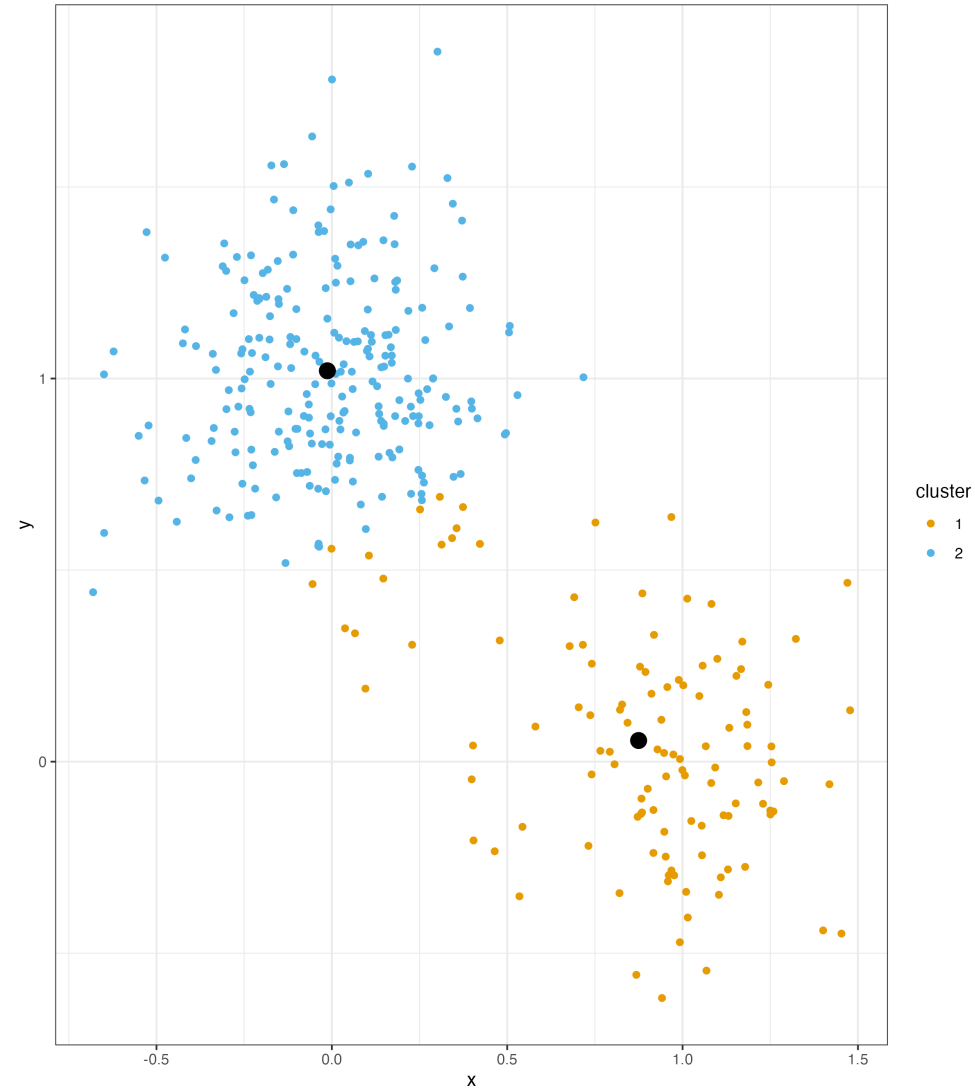
Third Iteration: Assign Points

```
d$assignment <-
  ifelse(GetDistances(d, centroids[1,]) <=
           GetDistances(d, centroids[2,]),
         1, 2)

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



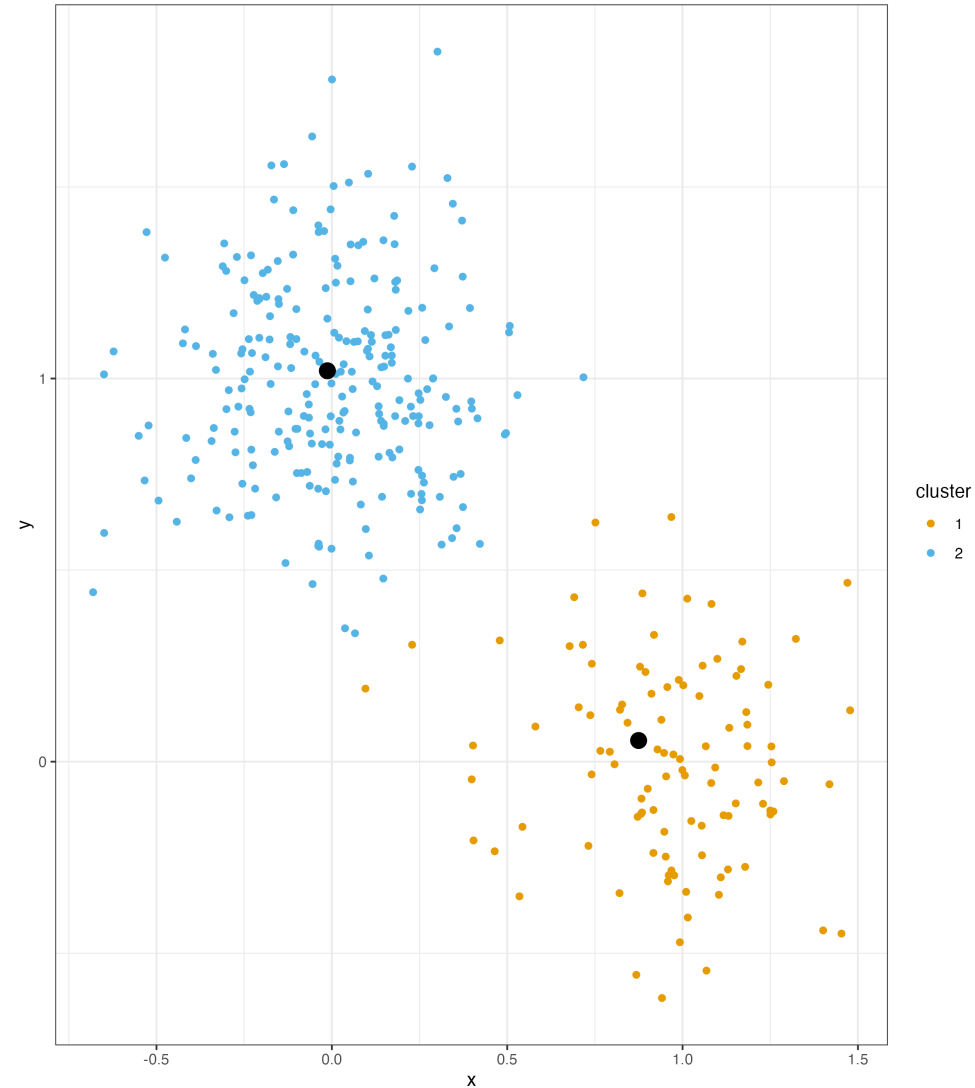
Third Iteration: Move Centroids

```
centroids <-
  data.frame(x = c(mean(d$x[d$assignment == 1]
                   mean(d$x[d$assignment == 2]
             y = c(mean(d$y[d$assignment == 1]
                   mean(d$y[d$assignment == 2]

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



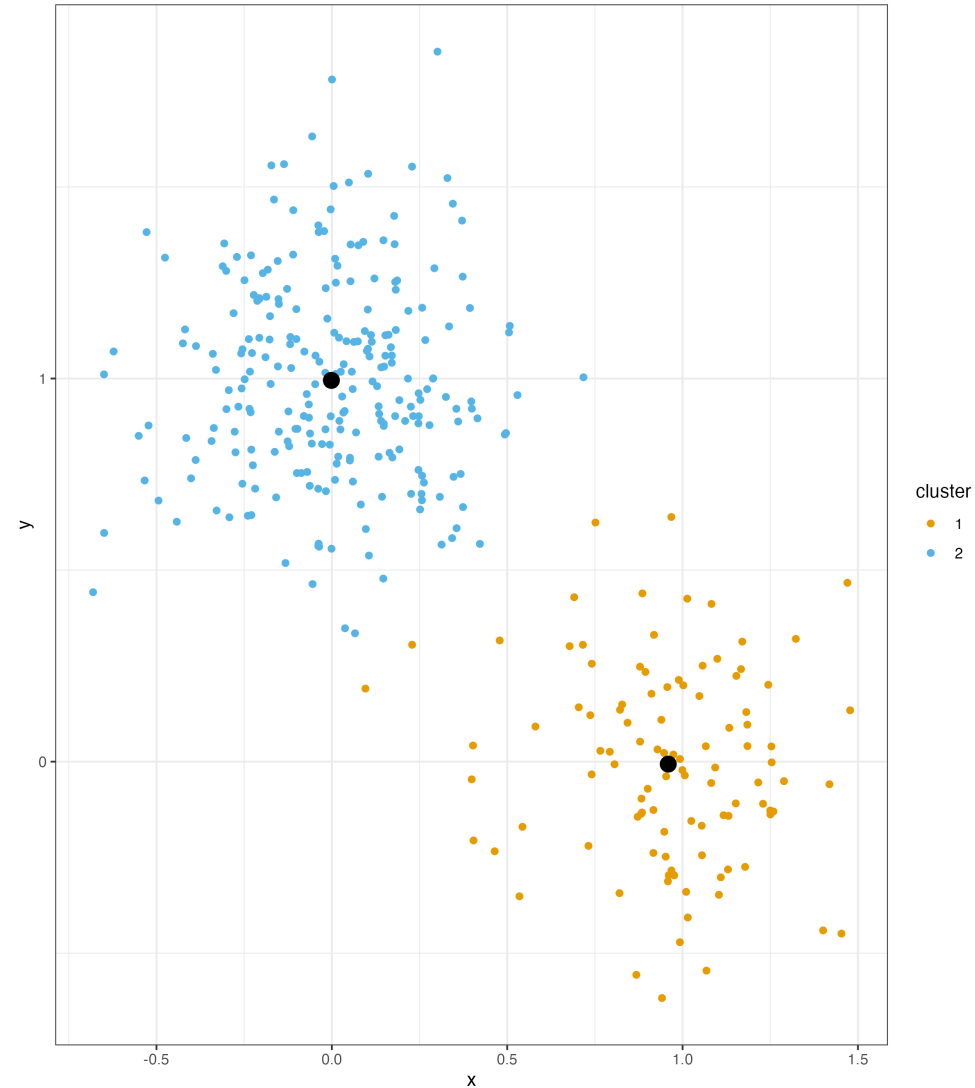
Fourth Iteration: Assign Points

```
d$assignment <-
  ifelse(GetDistances(d, centroids[1,]) <=
           GetDistances(d, centroids[2,]),
         1, 2)

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



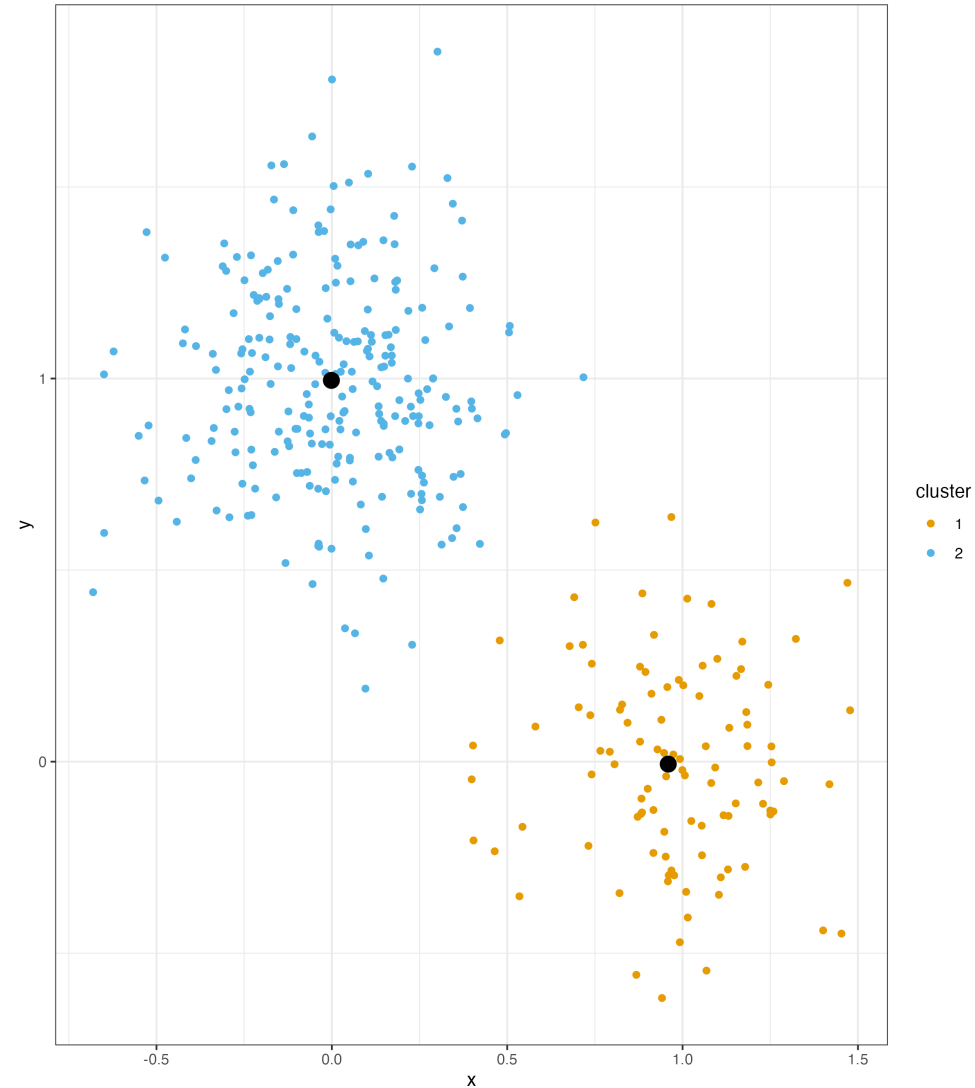
Fourth Iteration: Move Centroids

```r
centroids <-
  data.frame(x = c(mean(d$x[d$assignment == 1]
                   mean(d$x[d$assignment == 2]
             y = c(mean(d$y[d$assignment == 1]
                   mean(d$y[d$assignment == 2]

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



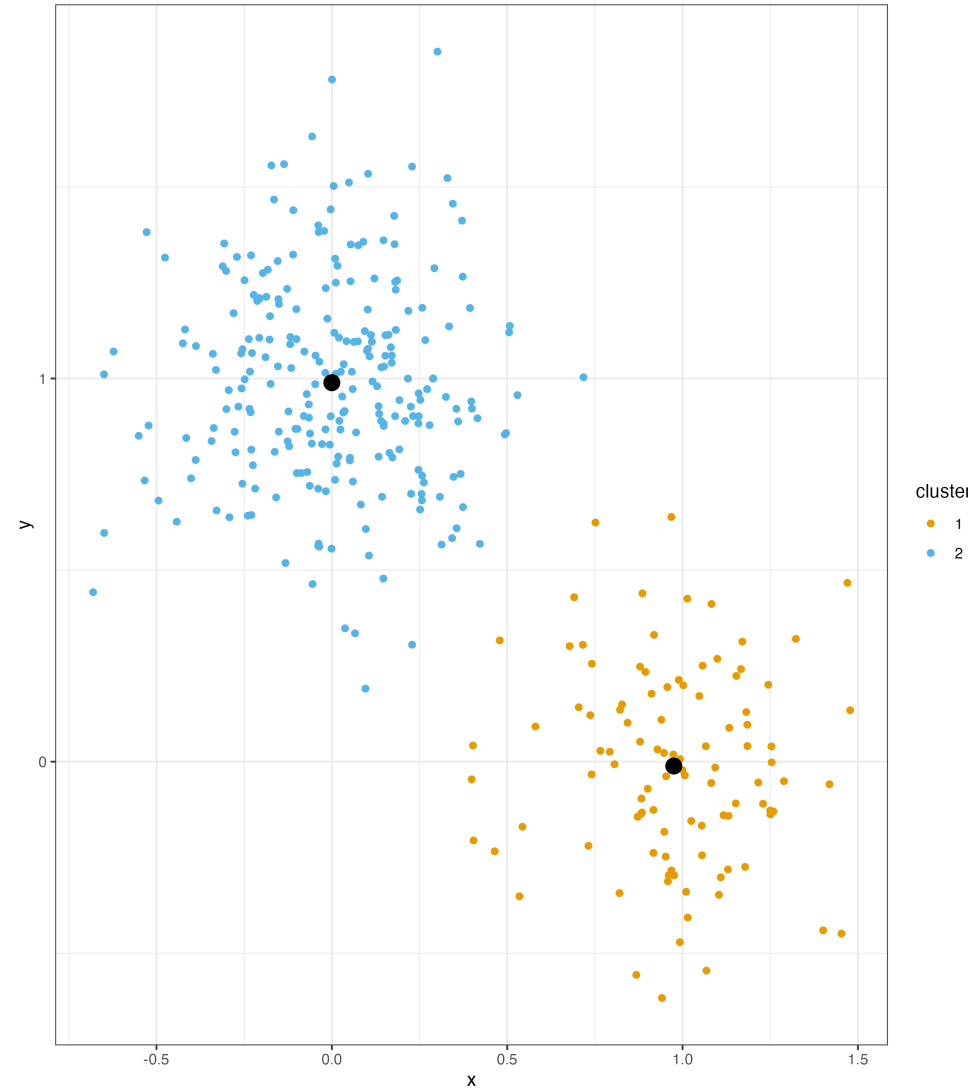
Fifth Iteration: Assign Points

```
d$assignment <-
  ifelse(GetDistances(d, centroids[1,]) <=
           GetDistances(d, centroids[2,]),
         1, 2)

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



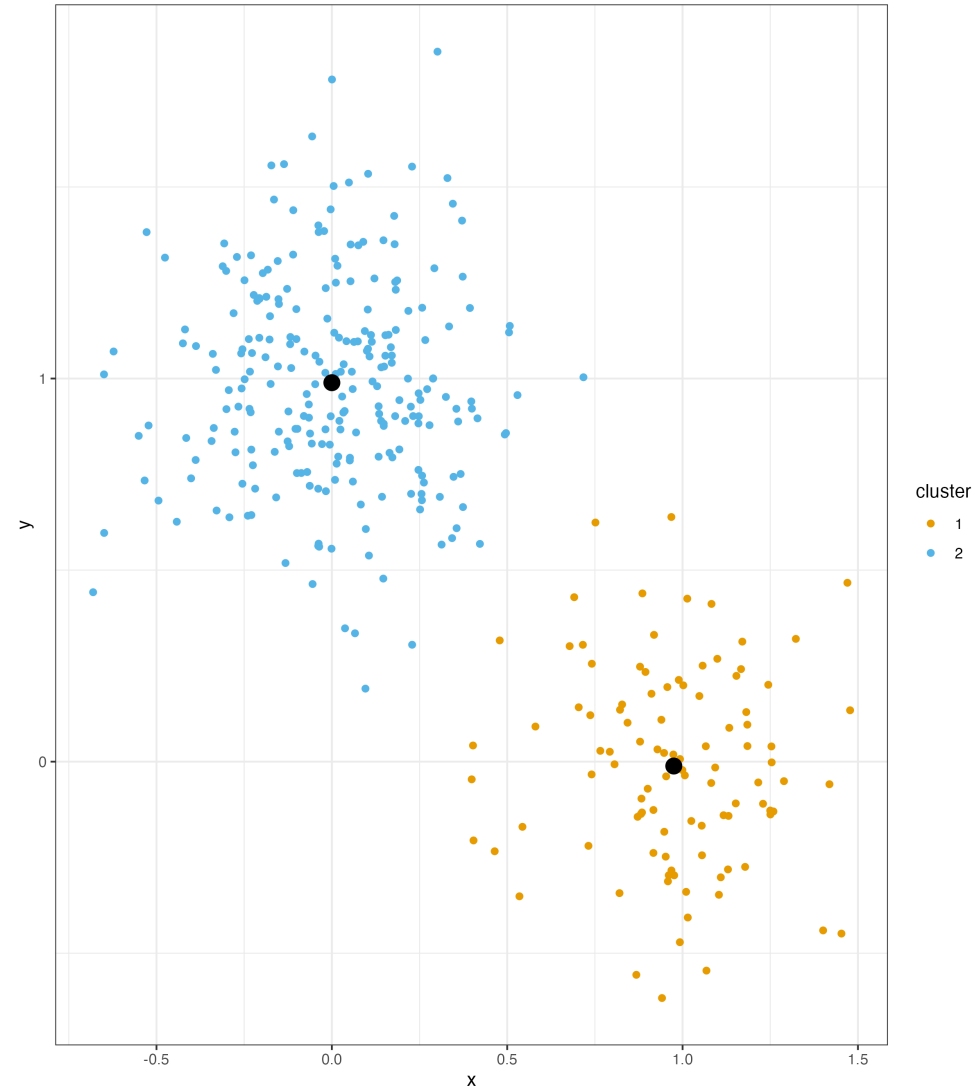
Fifth Iteration: Move Centroids

```
centroids <-
  data.frame(x = c(mean(d$x[d$assignment == 1]
                   mean(d$x[d$assignment == 2]
             y = c(mean(d$y[d$assignment == 1]
                   mean(d$y[d$assignment == 2]

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
 theme_bw()
```



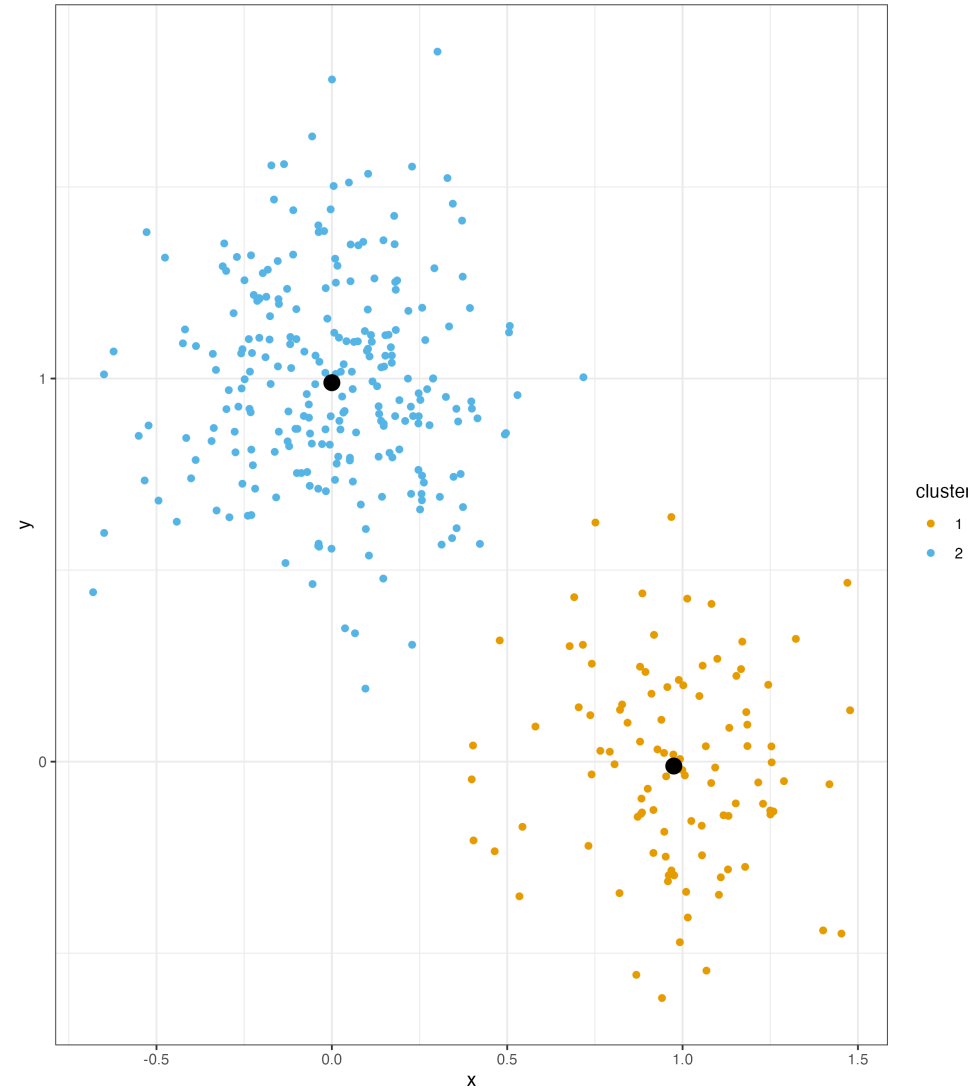
Sixth Iteration: Assign Points

```
d$assignment <-
  ifelse(GetDistances(d, centroids[1,]) <=
           GetDistances(d, centroids[2,]),
         1, 2)

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
 theme_bw()
```



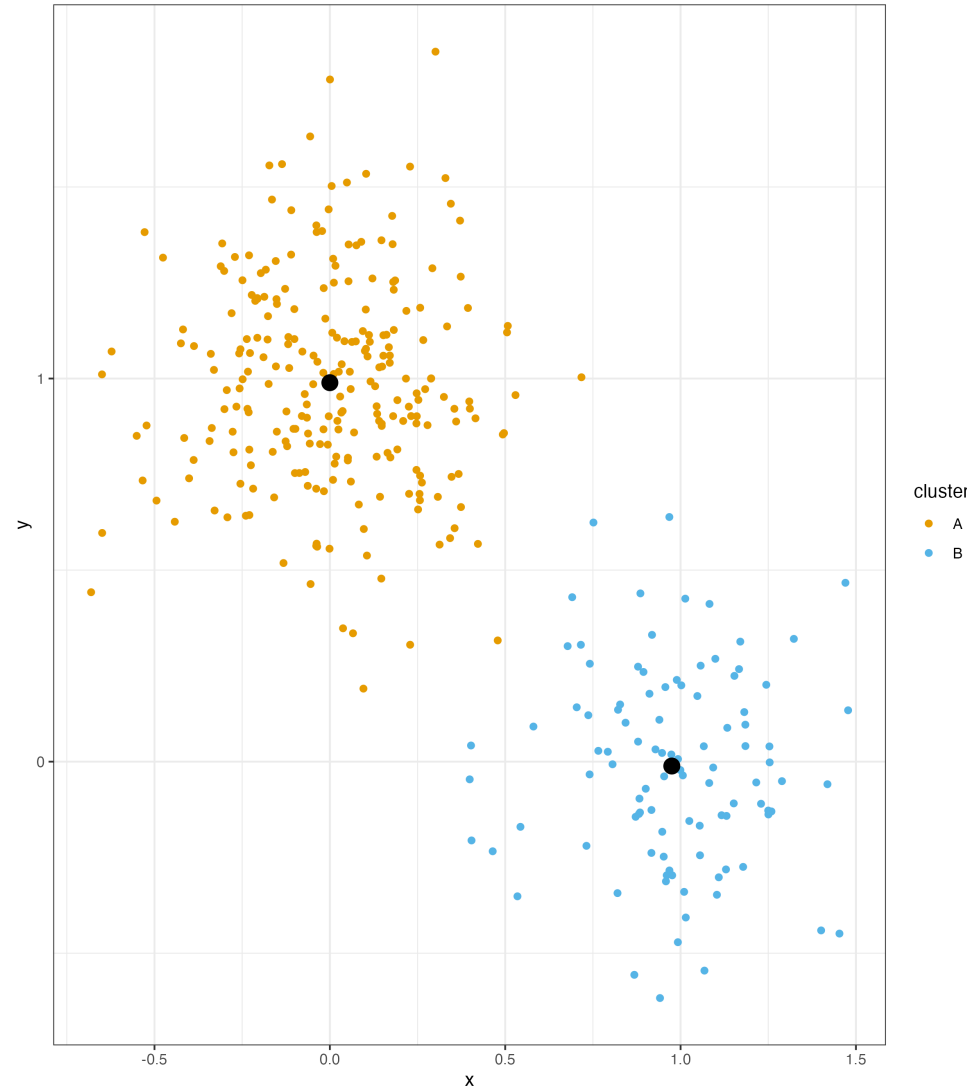
Sixth Iteration: Move Centroids

```
centroids <-
  data.frame(x = c(mean(d$x[d$assignment == 1]
                   mean(d$x[d$assignment == 2]
             y = c(mean(d$y[d$assignment == 1]
                   mean(d$y[d$assignment == 2]

ggplot(d,
       aes(x=x,
           y=y,
           color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



"True" Assignments

```
ggplot(d, aes(x=x, y=y, color=cluster)) +
  geom_point() +
  geom_point(data=centroids,
             aes(x=x, y=y),
             color="black",
             size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



Generalizing our k-Means Implementation

What Do We Need to Do?

1. **Initialize Centroids:**
- *Output*: Dataframe of k centroids
- *Inputs*: The dataframe we are clustering and k
2. **Get Cluster Assignments:**
- *Output*: Vector of cluster assignments
- *Inputs*: Data, centroids, a distance function
3. **Recalculate Centroids:**
- *Output*: A dataframe of new centroid locations
- *Inputs*: Data, centroids, and cluster assignments
4. **Evaluation Function:**
- *Output*: Within-cluster sum of squares
- *Inputs*: Data, centroids, and cluster assignments

Initialize Centroids

```r
InitializeCentroids <- function(data, k){
  # TODO: Return a dataframe of k starting
  #  centroids, one centroid per row
  c <- sample(1:nrow(data), k)
  centroids <- data[c, ]

  return(centroids)
}
```

Get Cluster Assignments

```r
GetAssignments <- function(data, centroids, distfun){
  # TODO: Return a vector of numeric cluster
  #  assignments by matching each point with
  #  its nearest centroid

  dist <- matrix(rep(NA, nrow(data)*nrow(centroids)),
                 ncol=nrow(centroids))

  for (i in 1:nrow(centroids)){
    dist[,i] <- distfun(data, centroids[i,])
  }
  assignment <- max.col(-dist)

  return(assignment)
}
```

Recalculate Centroids

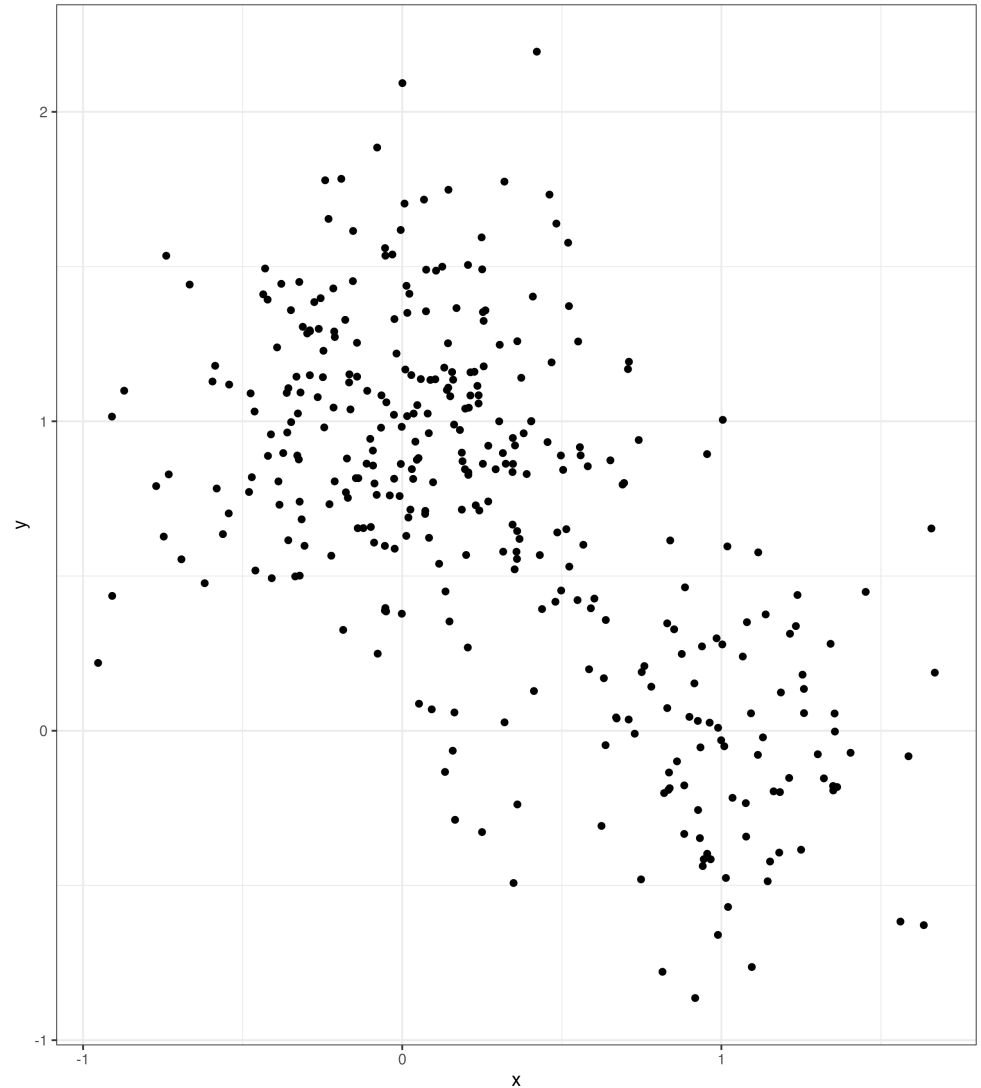
```r
RecalculateCentroids <- function(data, centroids, assignment){
  # TODO: Return a dataframe of new centroids
  #  calculated from the means of the points
  #  in each cluster

  for (i in 1:nrow(centroids)){
    centroids[i,] <- colMeans(data[assignment==i,])
  }

  return(centroids)
}
```

Evaluation Function

```r
WithinClusterSumSquares <- function(data, centroids, assignment){
  # TODO: Return the total within-cluster sum of squares

  WCSS <- 0
  for (i in 1:nrow(centroids)){
    WCSS <- WCSS + sum(GetDistances(data[assignment==i,], centroids[i,])^2)
  }

  return(WCSS)
}
```

Some New Data

```r
set.seed(8675309)

# New data, same solution but larger variance
d <- data.frame(x = c(rnorm(250, 0, 0.35),
                      rnorm(100, 1, 0.35)),
                y = c(rnorm(250, 1, 0.35),
                      rnorm(100, 0, 0.35)))

# shuffle to destroy any "known" solution
d <- d[sample(1:nrow(d)),]

# looks worse!
ggplot(d, aes(x=x, y=y)) +
  geom_point() +
  theme_bw()
```

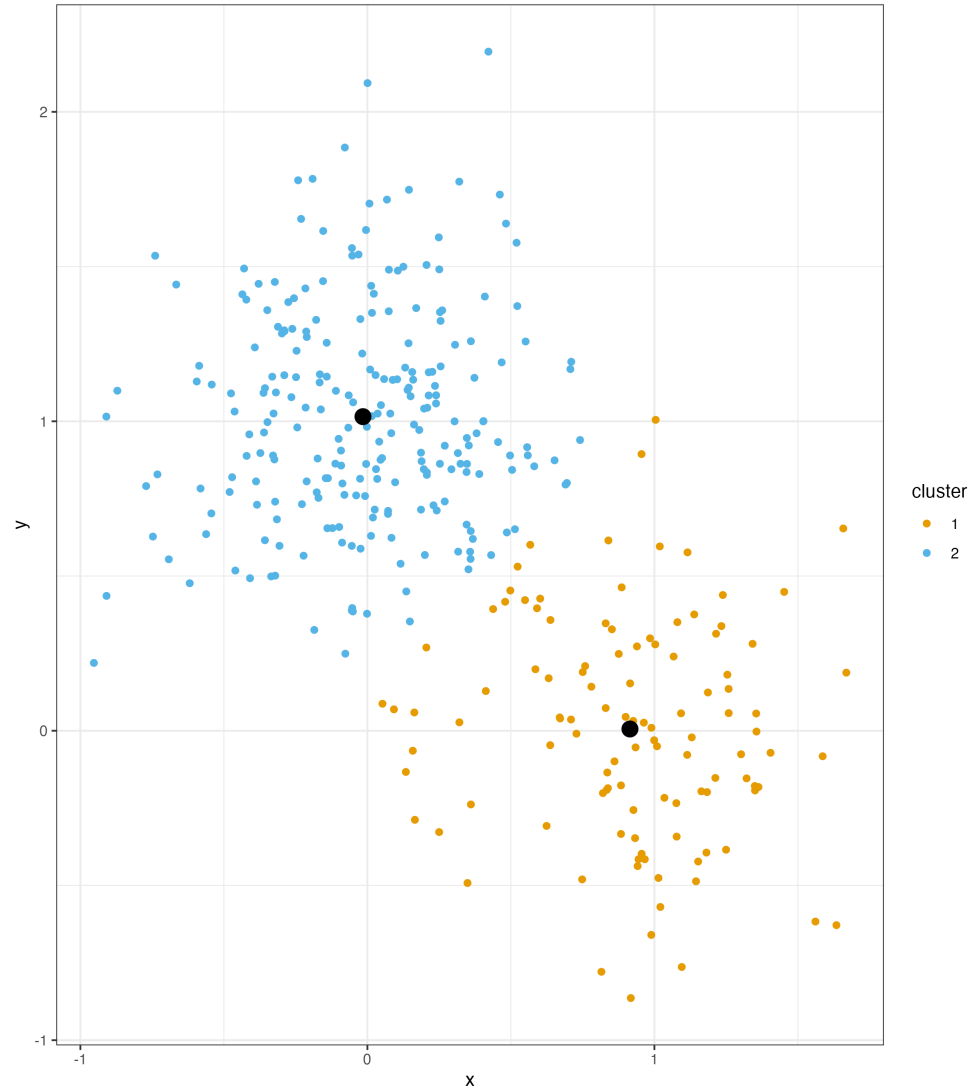


Putting it All Together

```r
# TODO: initialize starting values
k <- 2
centroids <- InitializeCentroids(d, k)
old_centroids <- InitializeCentroids(d, k)

# TODO: run until convergence
while(!identical(old_centroids, centroids)){
  old_centroids <- centroids
  assignment <- GetAssignments(d, centroids, GetDistances)
  centroids <- RecalculateCentroids(d, centroids, assignment)
}

# TODO: get WCSS for cluster solution
WithinClusterSumSquares(d, centroids, assignment)
```

Visualize the Solution: $k = 2$

```
ggplot(d, aes(x=x,
              y=y,
              color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids, aes(x=x, y=y),
             color="black", size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```

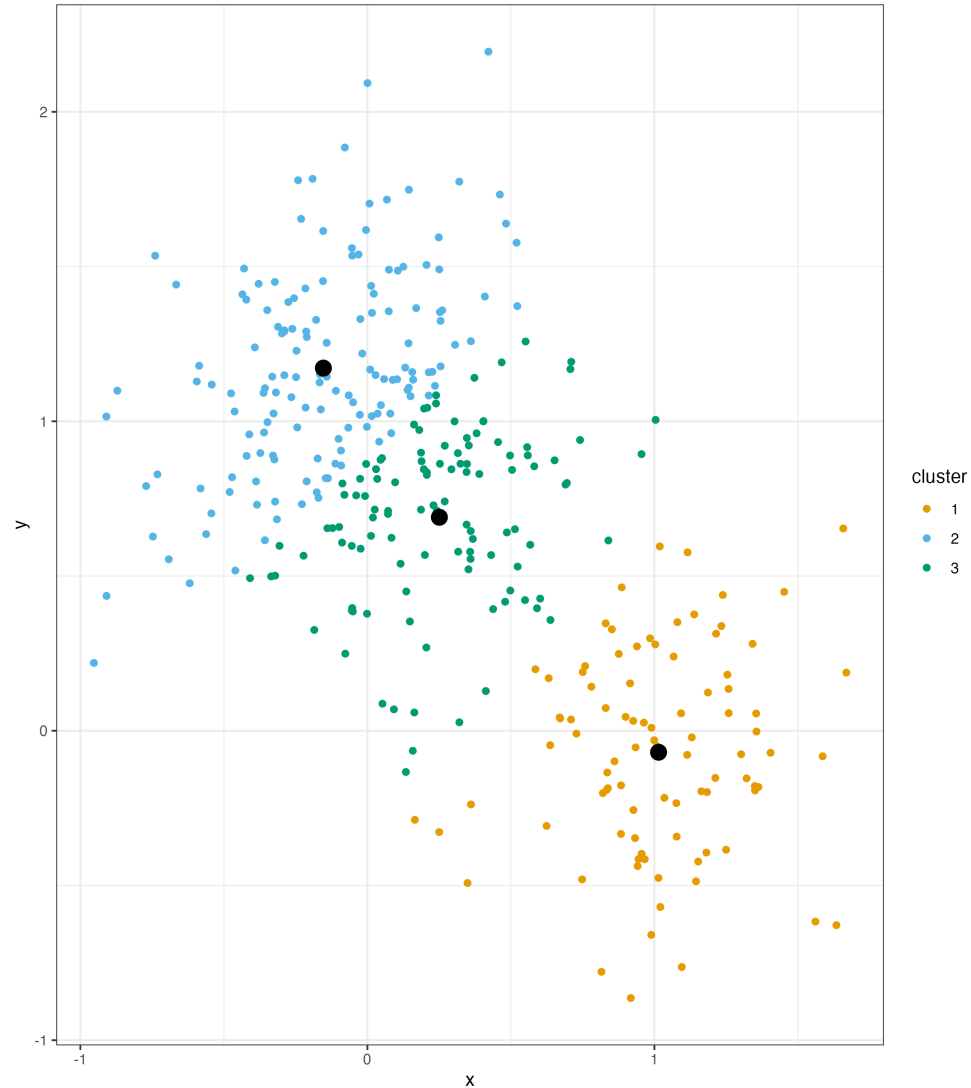


What About $k = 3$?

```r
# TODO: initialize starting values
k <- 3
centroids <- InitializeCentroids(d, k)
old_centroids <- InitializeCentroids(d, k)

# TODO: run until convergence
while(!identical(old_centroids, centroids)){
  old_centroids <- centroids
  assignment <- GetAssignments(d, centroids, GetDistances)
  centroids <- RecalculateCentroids(d, centroids, assignment)
}

# TODO: get WCSS for cluster solution
WithinClusterSumSquares(d, centroids, assignment)
```

Visualize the Solution: $k = 3$

```
ggplot(d, aes(x=x,
              y=y,
              color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids, aes(x=x, y=y),
             color="black", size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```



What About $k = 4$?

```r
# TODO: initialize starting values
k <- 4
centroids <- InitializeCentroids(d, k)
old_centroids <- InitializeCentroids(d, k)

# TODO: run until convergence
while(!identical(old_centroids, centroids)){
  old_centroids <- centroids
  assignment <- GetAssignments(d, centroids, GetDistances)
  centroids <- RecalculateCentroids(d, centroids, assignment)
}

# TODO: get WCSS for cluster solution
WithinClusterSumSquares(d, centroids, assignment)
```

Visualize the Solution: $k = 4$

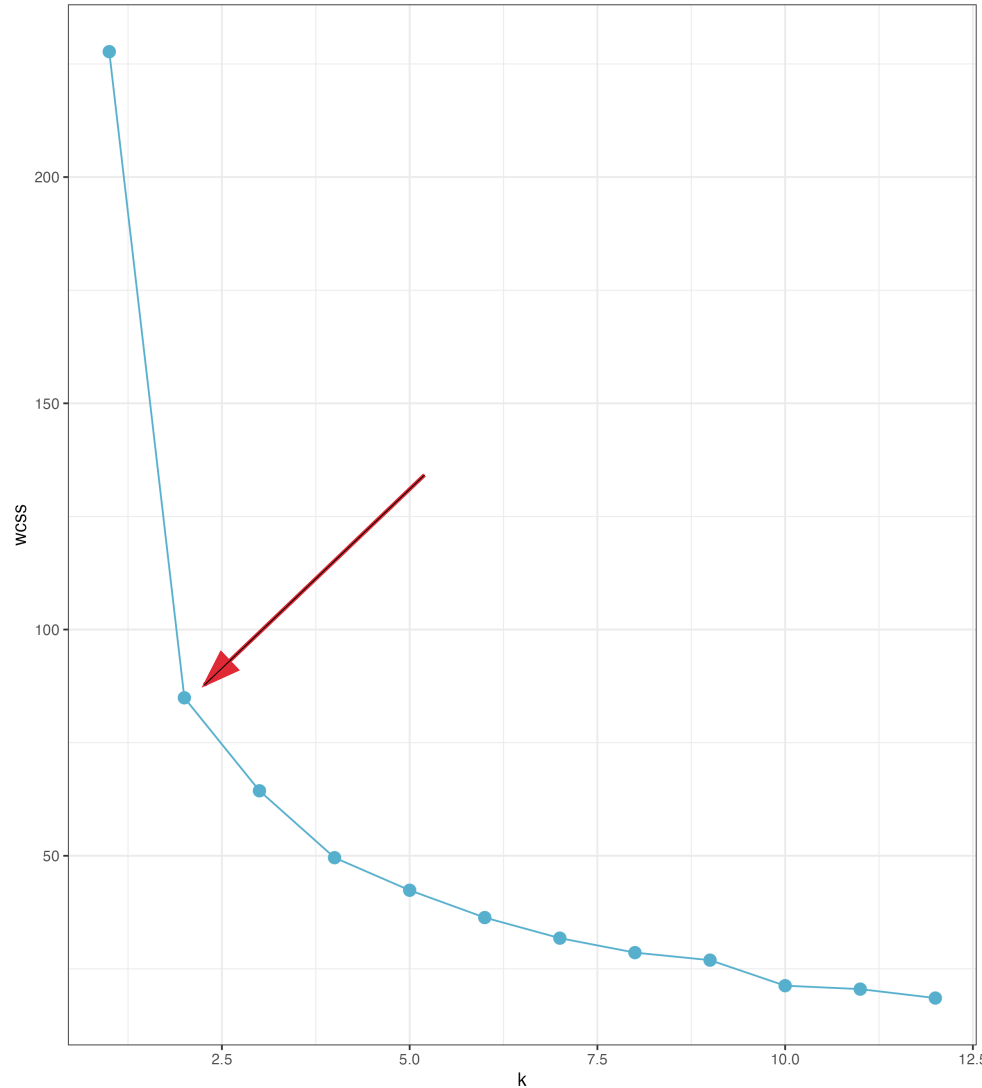
```
ggplot(d, aes(x=x,
              y=y,
              color=as.factor(assignment))) +
  geom_point() +
  geom_point(data=centroids, aes(x=x, y=y),
             color="black", size=4) +
  labs(color='cluster') +
  scale_color_okabeito() +
  theme_bw()
```

How Do You Decide What the Right Answer is?

- You'll never know the right answer!
- Look at some evaluation metric
  - Here we use within-cluster variance
  - Other ones exist (like silhouette scores)
- The problem is that adding an additional cluster always makes the evaluation metric go down
- Ask "when does adding another cluster stop making a big difference in my evaluation metric?"
  - This is a judgement call!
  - Often we look at "scree plot" and try and identify the "elbow"
  - Plot WCSS vs. k
  - Pick the "elbow," or the point where adding another cluster doesn't give much improvement

The Elbow Plot

```
ks <- 1:12
wcss <- vector('numeric', length(ks))
for (k in ks){
  centroids <- InitializeCentroids(d, k)
  old_centroids <- InitializeCentroids(d, k)
  while(!identical(old_centroids, centroids)){
    old_centroids <- centroids
    assignment <- GetAssignments(d,
                                 centroids,
                                 GetDistances)
    centroids <- RecalculateCentroids(d,
                                      centroid
                                      assignme
  }
  wcss[k] <- WithinClusterSumSquares(d,
                                     centroids
                                     assignmen
}
cluster_results <- data.frame(k=ks, wcss=wcss)

ggplot(cluster_results, aes(x=k, y=wcss)) +
  geom_point(size=3, color="#59B2D1") +
  geom_line(color="#59B2D1") +
  theme_bw()
```



Wrap Up

Recap

- Clustering has tons of applications in social science!
  - Putting things into groups is helpful, but sometimes we really don't know what the groups should be
  - If you want to learn more about clustering and similar unsupervised learning paradigms, take a look at my course "Modern Approaches in Measurement" happening in Spring Semester!
- Distance metrics give us a way to quantify how similar (or different) two observations are
  - The distance measure you choose dictates what features of the data are prioritized during this comparison
- `while` loops are our last major tool for control flow
  - Great for checking convergence conditions
  - If you can do a job with a `for` loop, however, prioritize using that

Final Thoughts

- PollEv.com/klintkanopka