Stats 102B - Clustering

Miles Chen, PhD

Department of Statistics

Week 8 Monday

Section 1

Unsupervised Learning - Clustering

Supervised vs Unsupervised Learning

So far the methods we covered have been supervised learning methods.

Supervised learning models take input values and try to produce output values that match or get close to target values.

The existence of target values allow us to measure things like loss (the difference between the predicted and the target values).

Unsupervised learning, on the other hand, do not have target values. Instead, we search for structure within the input values alone.

Two broad applications of unsupervised learning methods are clustering (grouping similar observations together) and dimension reduction (reducing redundant predictor variables)

Clustering

The goal of clustering is to create grouping of objects so that objects within a group are similar to each other and objects in different groups are not similar to each other.

There are many ways to define similarity and many ways to perform the grouping.

Some applications include grouping similar products together, grouping similar documents together, and grouping physical locations together (e.g. ride-share pickup locations).

K-Means Clustering

K-means clustering is a simple clustering algorithm. It defines similar objects as objects that are close each other in terms of Euclidean distance.

Do not mix up K-means with KNN. K-means is an unsupervised clustering method. KNN is a supervised classification method.

Clusters are defined by centroids. The centroid is the center (mean) of the points assigned to the cluster.

The centroid of cluster k is $\boldsymbol{\mu}_k$.

$$\boldsymbol{\mu}_k = \frac{\sum_n z_{nk}\mathbf{x}_n}{\sum_n z_{nk}}$$

z_{nk} is an indicator variable. It equals 1 if object n is in cluster k and 0 if not.

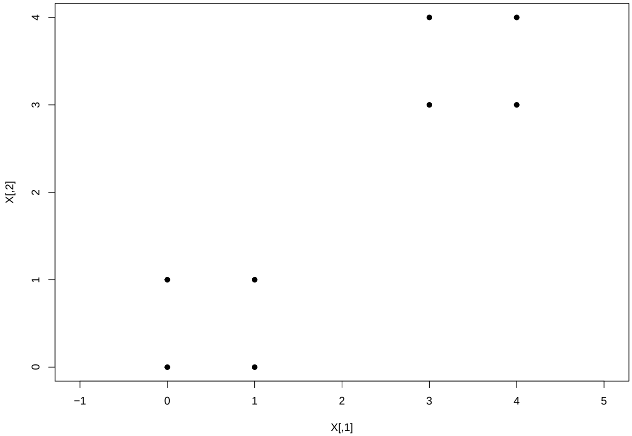
K-Means Clustering Algorithm

The algorithm can be described as follows:

0) Determine how many (k) clusters you will search for.
1) Randomly assign points in your data to each of the clusters.
2) Once all values have been assigned to a cluster, calculate the centroid of the values in each cluster.
3) Reassign values to clusters by associating values in the data set to the nearest (euclidean distance) centroid.
4) Repeat steps 2 and 3 until convergence. Convergence occurs when no values are reassigned to a new cluster.

Toy Example

```
par(mar=c(4.1, 4.1, 2.1, 2.1))
X <- matrix(c(0,0,  0,1,  1,0,  1,1,  3,3,  3,4,  4,3,  4,4), ncol = 2, byrow = TRUE)
plot(X, asp = 1, pch = 19)
```



Toy Example

Step 0: Determine how many (k) clusters you will search for.

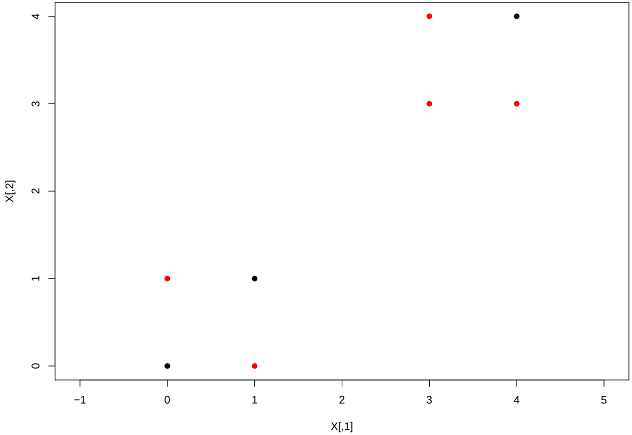
Let's say I'll search for 2 clusters. (Later, we'll discuss how to choose how many clusters to search for.)

Step 1: Randomly assign points to a cluster.

```
set.seed(3)
assignments <- factor(sample(c(1,2), nrow(X), replace = TRUE))
z1 <- as.integer(assignments == 1)
z2 <- as.integer(assignments == 2)
```

Assignments plotted

```
par(mar=c(4.1, 4.1, 2.1, 2.1))
plot(X, col = assignments, asp = 1, pch = 19)
```



Centroids

Step 2: Calculate the centroids of the values in each cluster.

```
##   X1 X2 assignments z1 z2
## 1  0  0           1  1  0
## 2  0  1           2  0  1
## 3  1  0           2  0  1
## 4  1  1           1  1  0
## 5  3  3           2  0  1
## 6  3  4           2  0  1
## 7  4  3           2  0  1
## 8  4  4           1  1  0
```

$$\boldsymbol{\mu}_k = \frac{\sum_n z_{nk}\mathbf{x}_n}{\sum_n z_{nk}}$$

- Cluster 1: $x_1 = (0 + 1 + 4)/3 = 1.667$, $x_2 = (0 + 1 + 4)/3 = 1.667$
- Cluster 2: $x_1 = (0 + 1 + 3 + 3 + 4)/5 = 2.2$, $x_2 = (1 + 0 + 3 + 4 + 3)/5 = 2.2$

Centroids

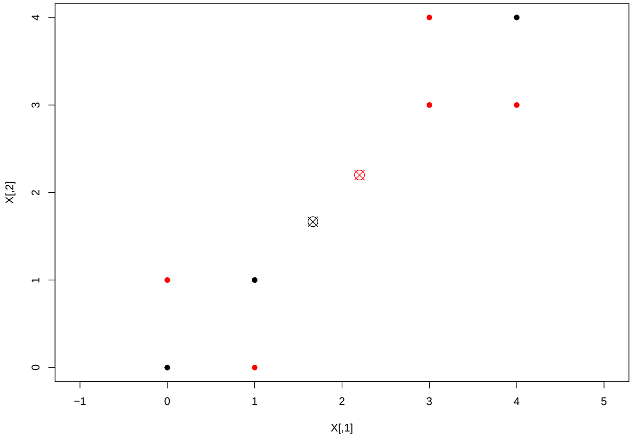
Step 2: Calculate the centroids of the values in each cluster.

```r
library(dplyr)
dat <- data.frame(X, assignments, z1, z2)
centroids <- dat %>% group_by(assignments) %>%
  summarise(x1 = mean(X1), x2 = mean(X2))
print(centroids)
```

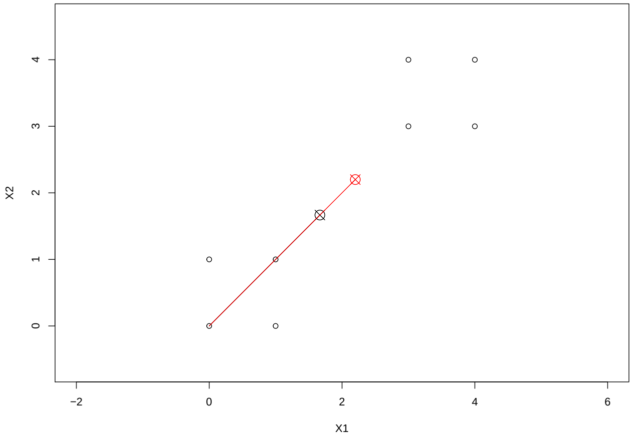
```
## # A tibble: 2 x 3
##   assignments    x1    x2
##   <fct>       <dbl> <dbl>
## 1 1            1.67  1.67
## 2 2            2.2   2.2
```

Centroids

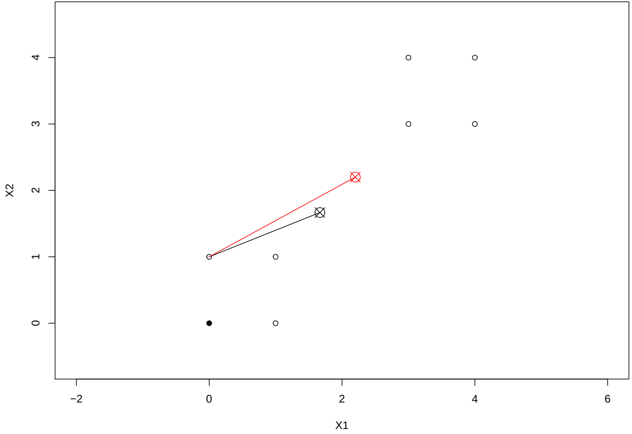
```
par(mar=c(4.1, 4.1, 2.1, 2.1))
plot(X, col = assignments, asp = 1, pch = 19)
points(centroids$x1, centroids$x2, col = centroids$assignments, cex = 2, pch = 13)
```



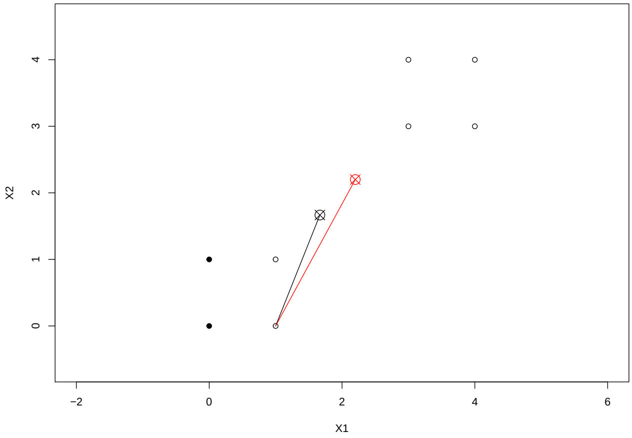
Reassign points to clusters based on Euclidean distance to centroid



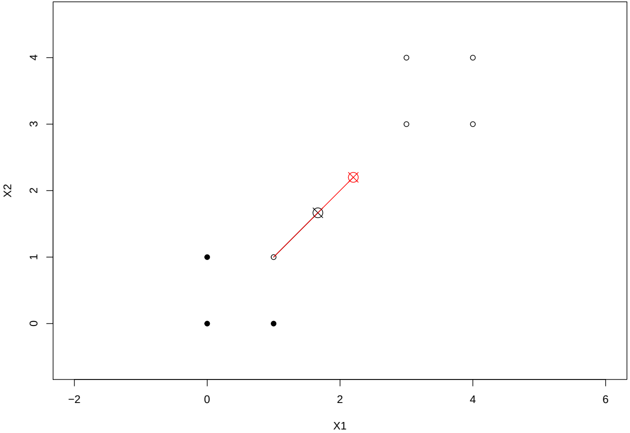
Reassign points



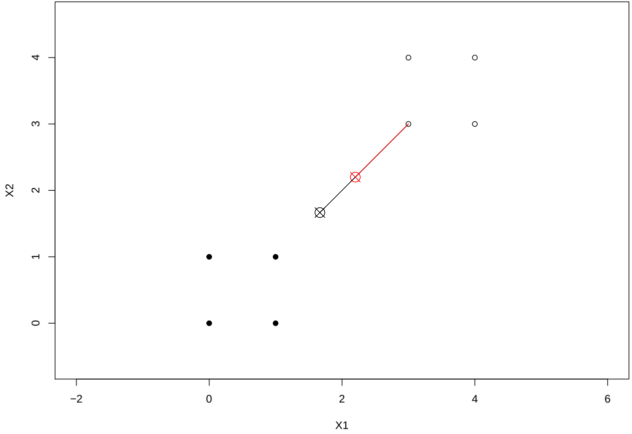
Reassign points



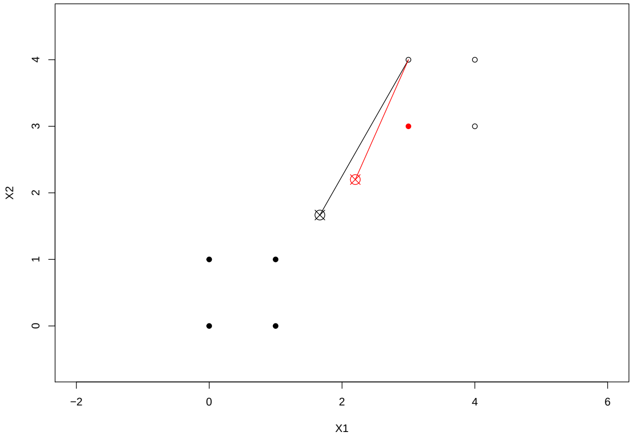
Reassign points



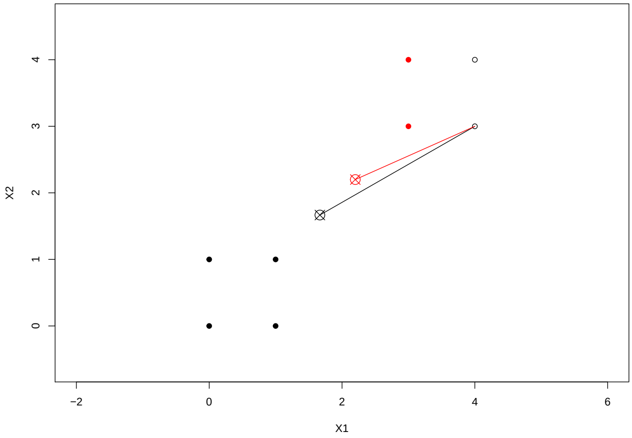
Reassign points



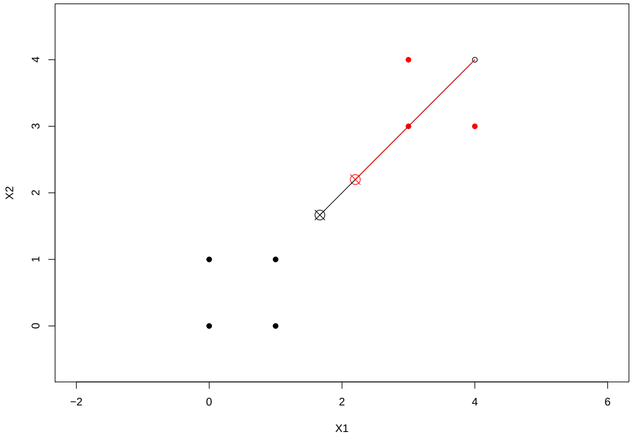
Reassign points



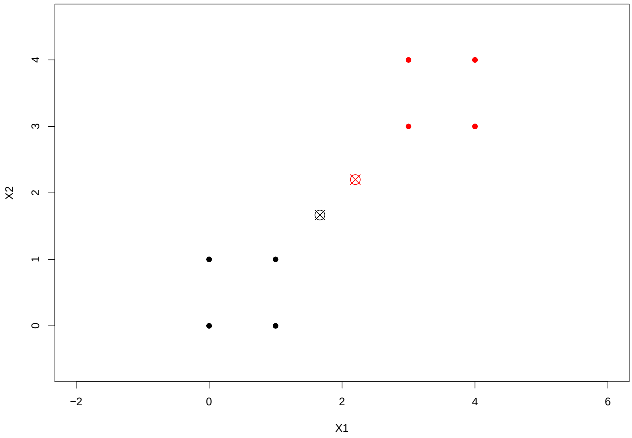
Reassign points



Reassign points



Points after reassignment

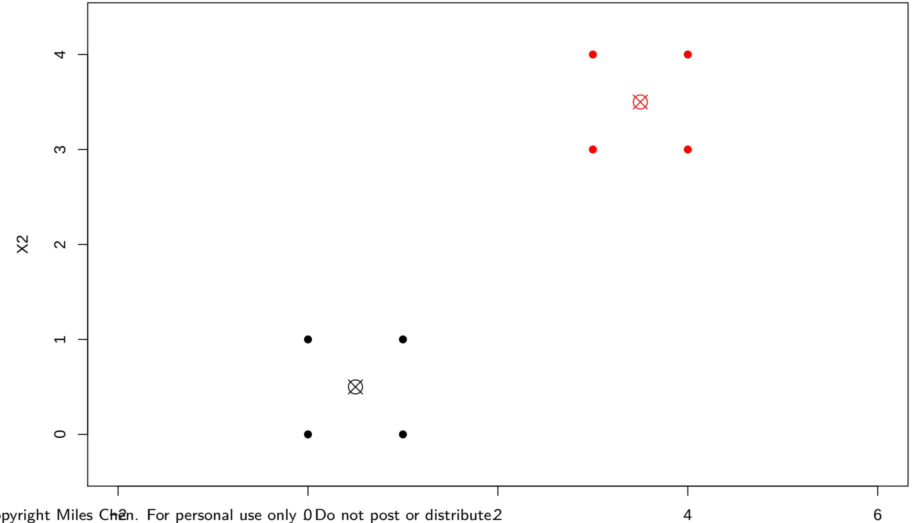


Recalculate Centroids

```r
dat <- data.frame(X, assignments)
centroids <- dat %>% group_by(assignments) %>%
  summarise(x1 = mean(X1), x2 = mean(X2))
print(centroids)
```

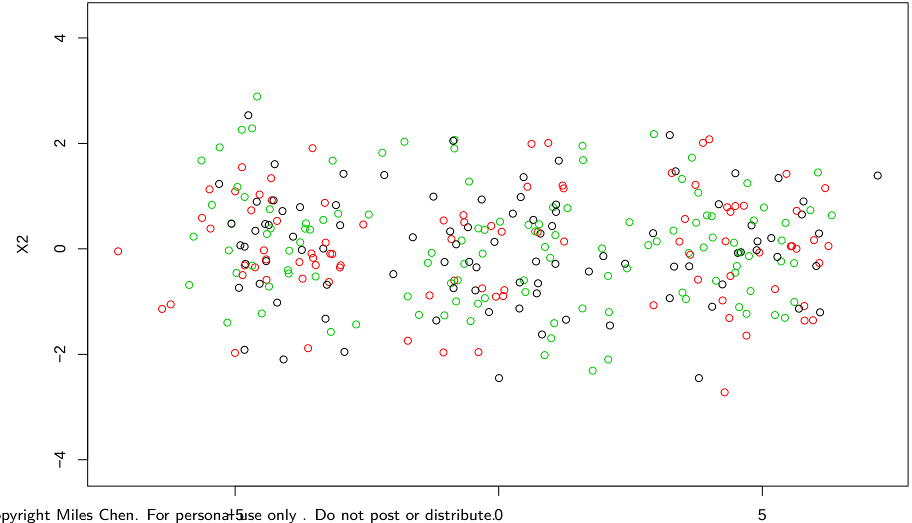
```
## # A tibble: 2 x 3
##   assignments    x1    x2
##   <fct>       <dbl> <dbl>
## 1 1             0.5   0.5
## 2 2             3.5   3.5
```

Clusters after one iteration (converged)



Slightly more complex example

Random assignments

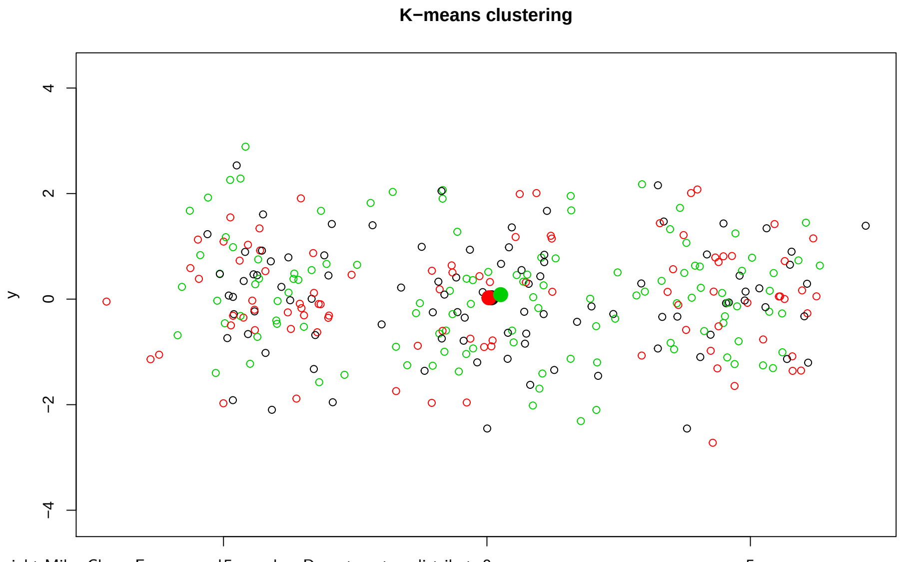


Calculate Centroids

```
print(means)
```

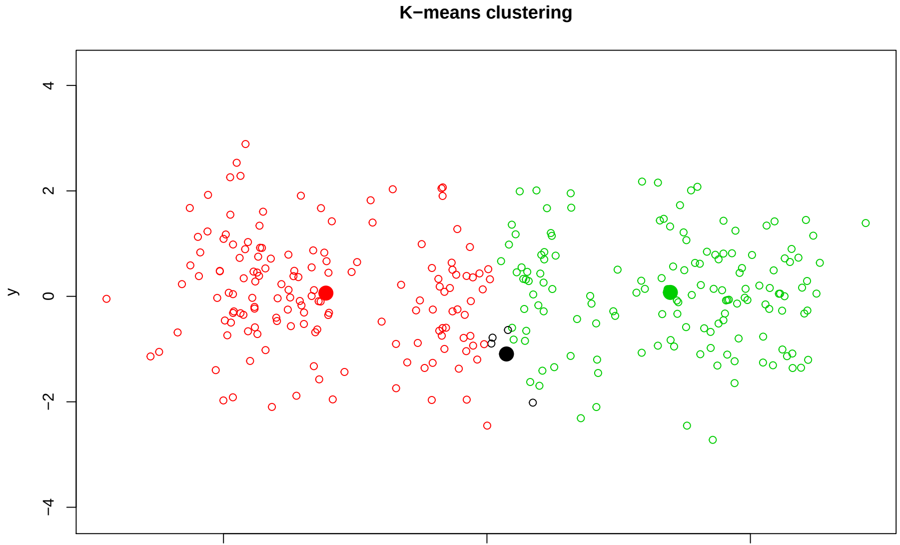
```
##                [,1]       [,2]
## [1,] 0.08600675 0.02554589
## [2,] 0.03725896 0.02542326
## [3,] 0.25991618 0.08461058
```

Calculate Centroids

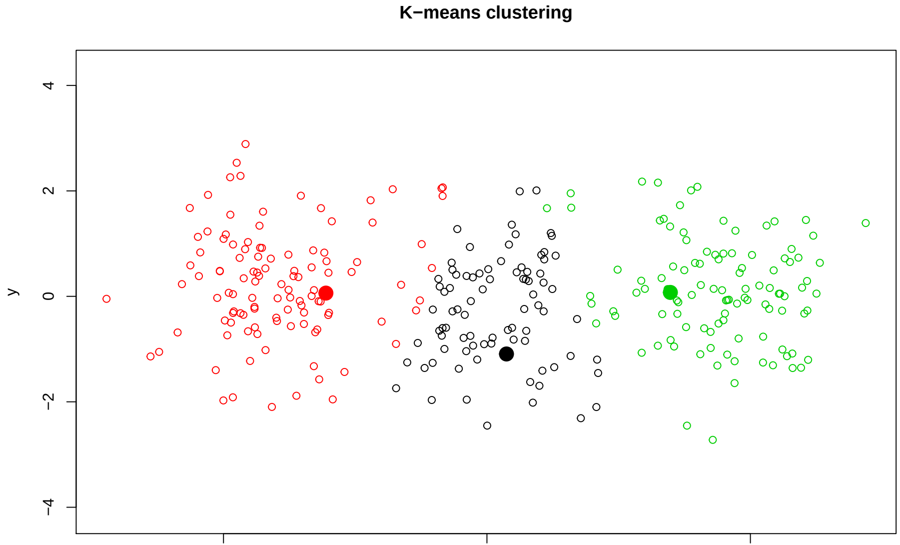


Reassign Points

Recalculate Centroids

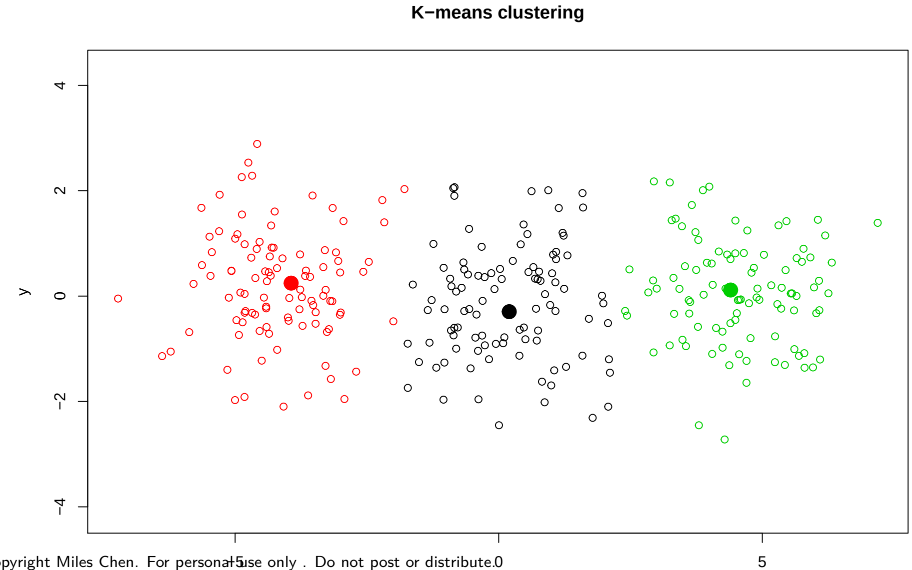


Reassign Points

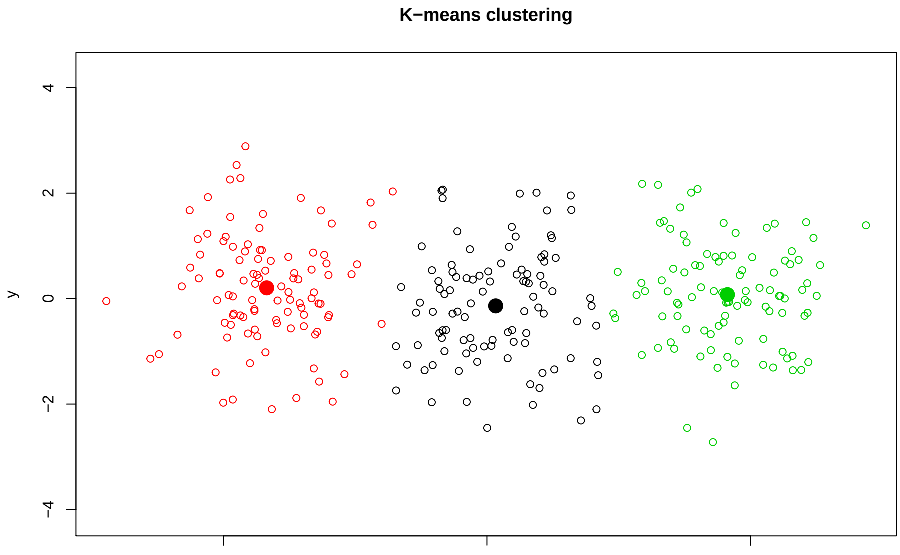


Recalculate Centroids

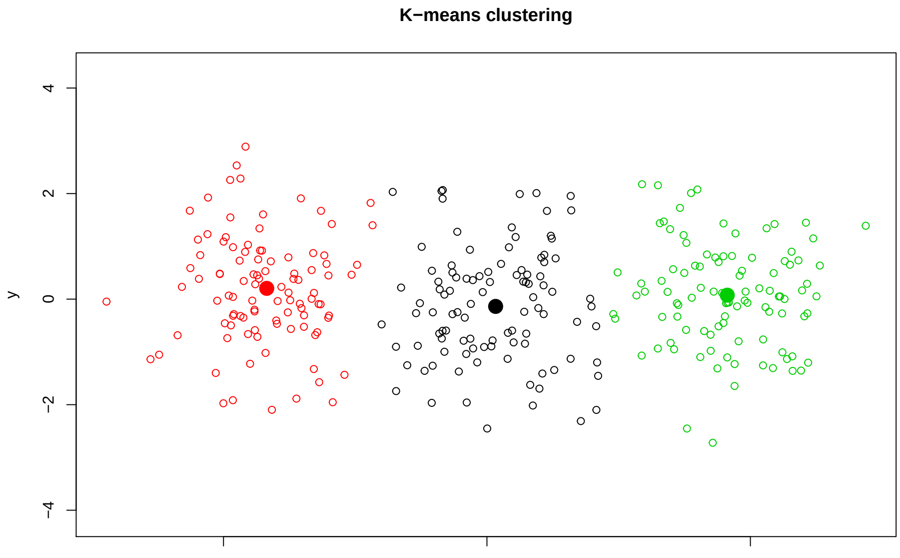
Reassign Points



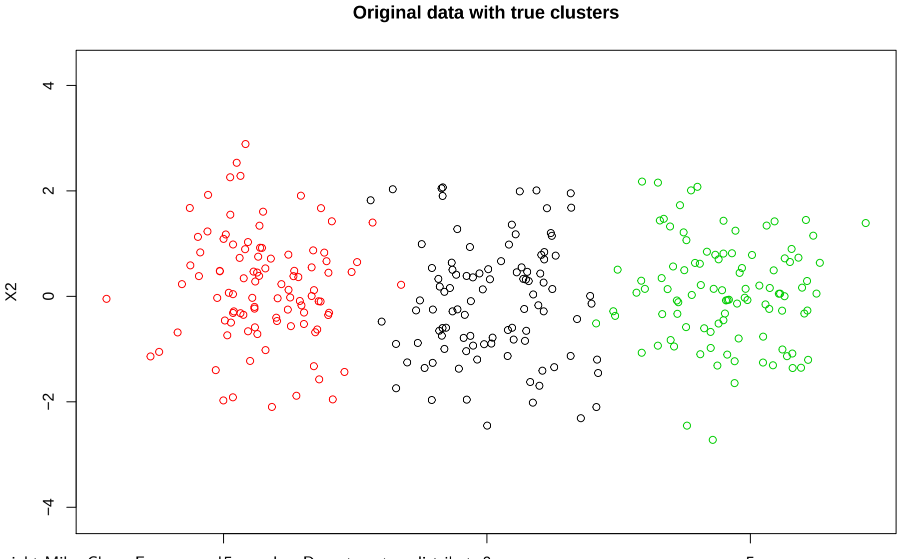
Recalculate Centroids



Reassign Points (no changes means it has converged)

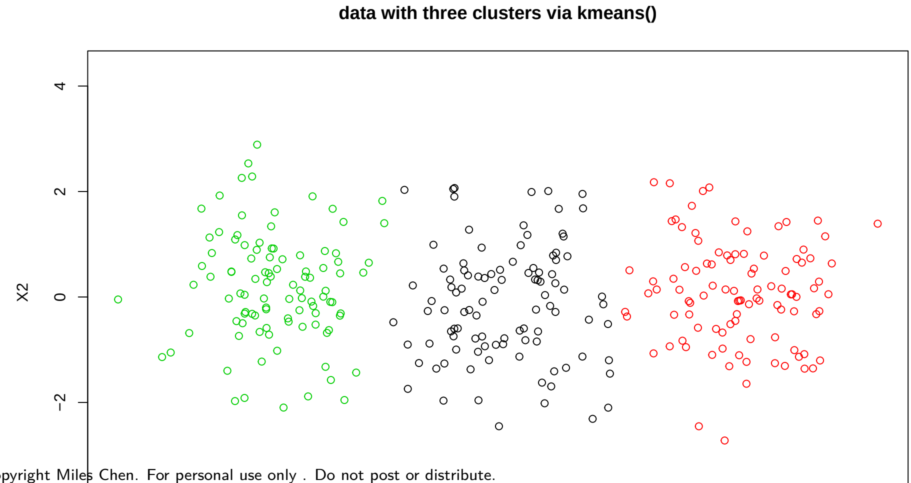


Comparison to Original Cluster assignments



Function kmeans() in R

```
results <- kmeans(dat, 3)  # super simple command
plot(dat, col = results$cluster, asp = 1, xlab = 'X1', ylab = 'X2', main = 'data with
```



Other results from kmeans()

```r
print(results)
```

```
## K-means clustering with 3 clusters of sizes 101, 99, 100
##
## Cluster means:
##         [,1]        [,2]
## 1  0.1239934 -0.12016322
## 2  4.5613159  0.07482233
## 3 -4.2250539  0.19233309
##
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 3 3 3 3 3 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [186] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [223] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [260] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [297] 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 225.6128 198.3275 192.9474
##  (between_SS / total_SS =  86.2 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Other results from kmeans()

```
str(results)
```

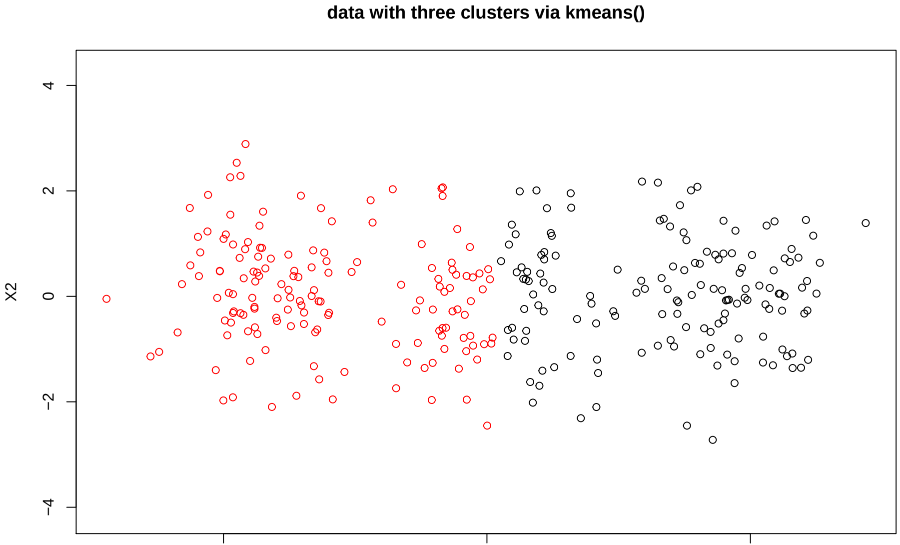
```
## List of 9
##  $ cluster     : int [1:300] 1 1 1 1 1 1 1 1 1 1 ...
##  $ centers     : num [1:3, 1:2] 0.124 4.5613 -4.2251 -0.1202 0.0748 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "1" "2" "3"
##   .. ..$ : NULL
##  $ totss       : num 4463
##  $ withinss    : num [1:3] 226 198 193
##  $ tot.withinss: num 617
##  $ betweenss   : num 3846
##  $ size        : int [1:3] 101 99 100
##  $ iter        : int 2
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

Trying other values of k, k = 2

```
results_2 <- kmeans(dat, 2); print(results_2)
```

```
## K-means clustering with 2 clusters of sizes 147, 153
##
## Cluster means:
##        [,1]       [,2]
## 1  3.421289 0.04684780
## 2 -3.015303 0.04978829
##
## Clustering vector:
##   [1] 2 1 2 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 1 2 2 2 2
##  [38] 2 1 1 1 2 2 2 2 1 2 2 1 2 1 2 2 1 2 2 1 1 1 2 1 1 2 2 2 1 1 1 1 1 2 2 1 1
##  [75] 2 1 1 1 1 1 1 2 2 1 2 1 2 2 2 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [186] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [297] 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 666.1363 690.4256
##  (between_SS / total_SS =  69.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

results with k = 2

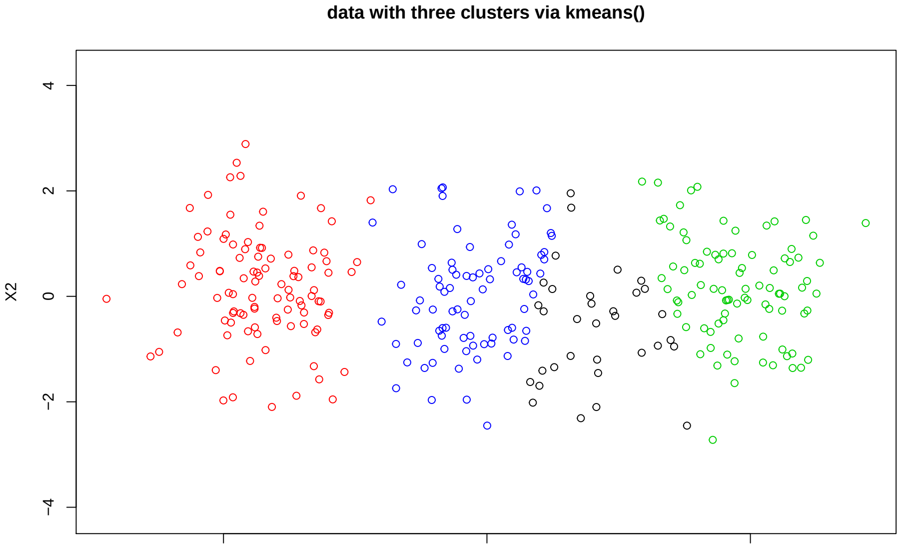


Trying other values of k, k = 4

```
results_4 <- kmeans(dat, 4); print(results_4)
```

```
## K-means clustering with 4 clusters of sizes 33, 99, 87, 81
##
## Cluster means:
##         [,1]        [,2]
## 1  2.0506930 -0.58241992
## 2 -4.2458046  0.18014025
## 3  4.7718626  0.15657585
## 4 -0.2580499  0.02800133
##
## Clustering vector:
##   [1] 4 1 4 4 1 4 4 1 4 4 1 1 4 4 4 4 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [38] 4 1 1 1 4 2 4 4 1 4 4 1 4 1 4 4 4 4 4 4 4 4 4 1 1 4 4 4 4 1 4 4 1 4 4 4 4
##  [75] 4 1 4 1 4 4 4 4 4 1 4 4 4 4 4 1 4 4 4 4 4 4 4 4 4 4 4 2 4 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 2
## [186] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 1 3 1 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 1 1
## [223] 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 1 1 3 3 3 1 3 3 3 3 3 3 1 3 3 3
## [260] 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [297] 3 3 3 1
##
## Within cluster sum of squares by cluster:
## [1]  59.64527 187.21276 152.28998 143.38663
##  (between_SS / total_SS =  87.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

results with k = 4

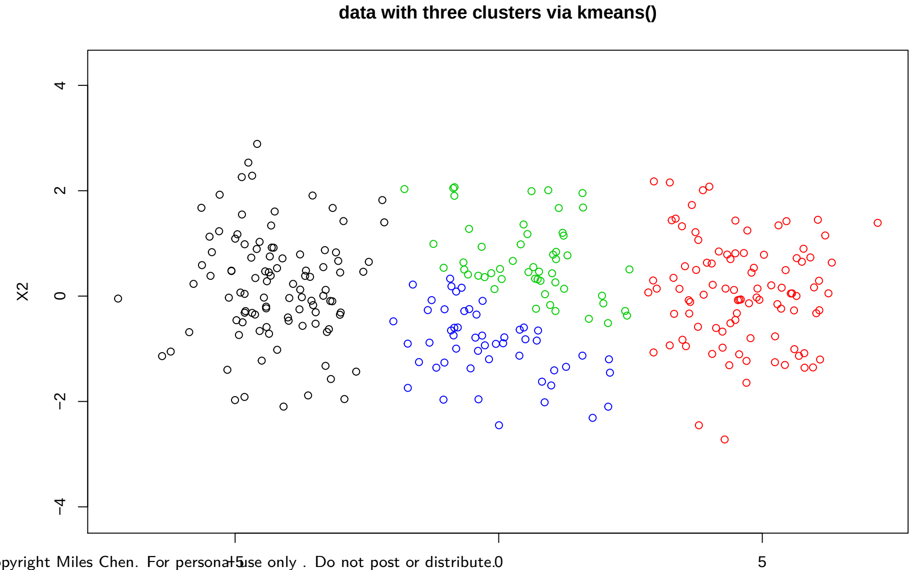


Trying other values of k, k = 4, different run

```
results_4 <- kmeans(dat, 4); print(results_4)
```

```
## K-means clustering with 4 clusters of sizes 100, 96, 52, 52
##
## Cluster means:
##         [,1]        [,2]
## 1 -4.2250539  0.19233309
## 2  4.6276954  0.07869407
## 3  0.5877396  0.68834080
## 4 -0.2063001 -0.92456591
##
## Clustering vector:
##   [1] 4 4 3 3 3 3 4 4 3 3 4 3 4 4 3 3 3 4 4 4 4 3 3 4 4 3 3 4 3 3 3 4 4 4 4 4 3
##  [38] 4 3 4 4 4 1 4 3 3 4 4 3 4 3 4 3 3 4 4 3 3 3 4 3 3 3 3 4 3 4 3 3 4 3 4 4 3
##  [75] 4 3 4 4 3 3 4 3 4 4 3 3 3 4 4 4 4 4 3 3 3 4 4 3 4 4 1 4 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2
## [223] 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [260] 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [297] 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 192.94740 183.85169  78.04110  79.61404
##  (between_SS / total_SS =  88.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

results with k = 4, different run = different results



Choosing K

```r
df <- data.frame( k = double(0), withinss = double(0)) # an empty df to store
for(k in 1:10){
  for(i in 1:20){
    kmresults <- kmeans(dat, k)
    df <- rbind(df, data.frame(k = k, withinss = kmresults$tot.withinss))
  }
}
```

Choosing K

```r
head(df, 30)
```

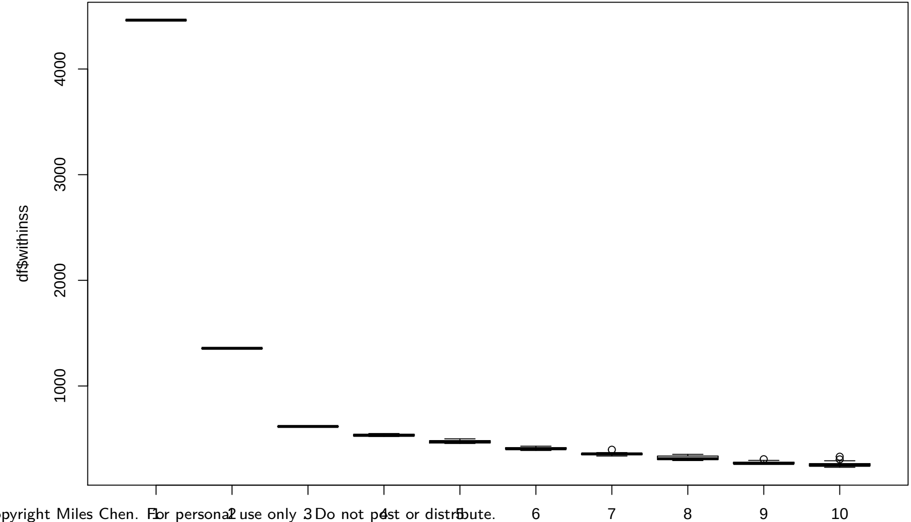
```
##     k withinss
## 1   1 4462.549
## 2   1 4462.549
## 3   1 4462.549
## 4   1 4462.549
## 5   1 4462.549
## 6   1 4462.549
## 7   1 4462.549
## 8   1 4462.549
## 9   1 4462.549
## 10  1 4462.549
## 11  1 4462.549
## 12  1 4462.549
## 13  1 4462.549
## 14  1 4462.549
## 15  1 4462.549
## 16  1 4462.549
## 17  1 4462.549
## 18  1 4462.549
## 19  1 4462.549
## 20  1 4462.549
## 21  2 1356.562
## 22  2 1356.562
## 23  2 1356.562
## 24  2 1356.562
## 25  2 1356.562
## 26  2 1356.562
## 27  2 1356.562
## 28  2 1356.562
## 29  2 1356.562
```

Choosing K

```r
tail(df, 30)
```

```
##       k withinss
## 171   9 307.0581
## 172   9 262.5156
## 173   9 282.5861
## 174   9 278.0010
## 175   9 295.7408
## 176   9 260.8046
## 177   9 261.7311
## 178   9 284.7416
## 179   9 279.3047
## 180   9 263.3226
## 181  10 258.0981
## 182  10 230.5012
## 183  10 244.0878
## 184  10 235.3907
## 185  10 263.0980
## 186  10 260.9706
## 187  10 242.8999
## 188  10 272.5117
## 189  10 267.6635
## 190  10 246.8170
## 191  10 245.0873
## 192  10 260.0689
## 193  10 234.6894
## 194  10 305.3552
## 195  10 330.8811
## 196  10 248.3713
## 197  10 248.6633
## 198  10 249.1468
## 199  10 292.0882
```

Choosing K



Choosing K

We see big drops in the within SS when we go from 1 to 2 to 3.

After $k = 3$, we don't see very much improvement in the within SS measure.