Clustering

Hierarchical, K-Means, and GMMs

SYS 6018 | Spring 2025

clustering.pdf

Contents

1 Clustering Intro

1.1 Required R Packages

We will be using the R packages of:

- `tidyverse` for data manipulation and visualization
- `mixtools` for mixture modeling
- `mclust` for model-based clustering
- `broom` for tidying model output

Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

1.2 Clustering

Cluster analysis divides data into groups (clusters) that are meaningful, useful, or both. [ITDM 7]

- If meaningful, then clusters should capture the natural structure of the underlying data generating process.
  - biological taxonomy
  - personality profiling
- Alternatively, clustering can be useful for summarizing or reducing the size/complexity of the data.
  - market segmentation for targeted marketing
  - group news articles into topics
  - data compression

1.2.1 Clustering: Two Views

1. Find homogeneous subgroups
   - Partition data into groups such that objects in the same group are *similar* to each other, while objects in different groups are *dissimilar*
2. Statistical Clustering
   - Partition data into groups such that objects in the same groups are from the same *distribution*

1.2.2 The Field of clustering

There are probably more approaches/algorithms for clustering than any other area of data mining.

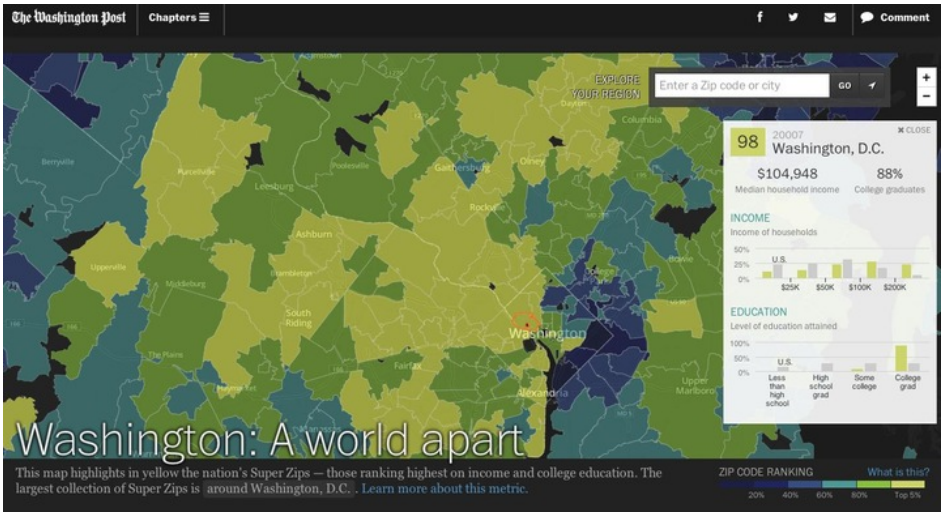
- Different criteria on which to base cluster analysis
  - E.g., Cluster plants on color, size, shape, geography
- Different ways to evaluate a clustering solution
- Different goals of clustering: understanding, data reduction, etc.

- The large variety of data types which can be clustered
  - rectangular, network, time series, functional, etc

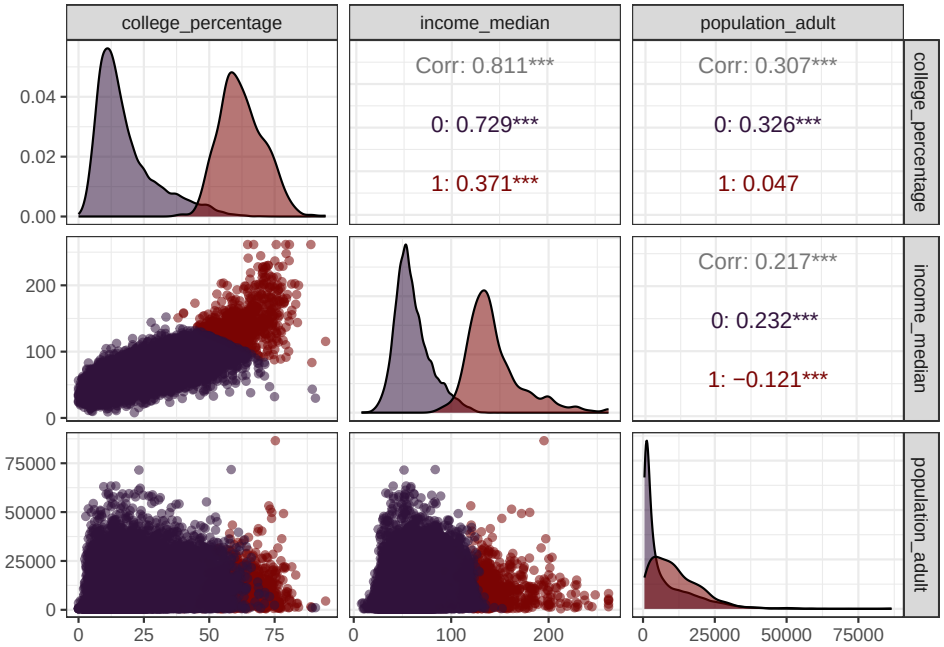
The three most popular methods (in my opinion) which are also the building blocks of the more complex and targeted methods are:

1. Hierarchical Clustering
2. K-means
3. Model Based Clustering

1.3 Example: SuperZip



The Washington Post released an interactive data viewer Washington: A world apart that drew attention to the term *Super Zips* defined by Charles Murray to describe the zipcodes that house the most prosperous, highly educated demographic clusters in the US. Posit produced a Shiny SuperZip Viewer if you want to explore. The data are from the 2000 census and includes zip codes with an adult population exceeding 500.



1.4 Example: *Leptograpsus variegatus*

The *Leptograpsus variegatus*, or purple rock crab, can take on a blue or orange coloring in some parts of Australia.

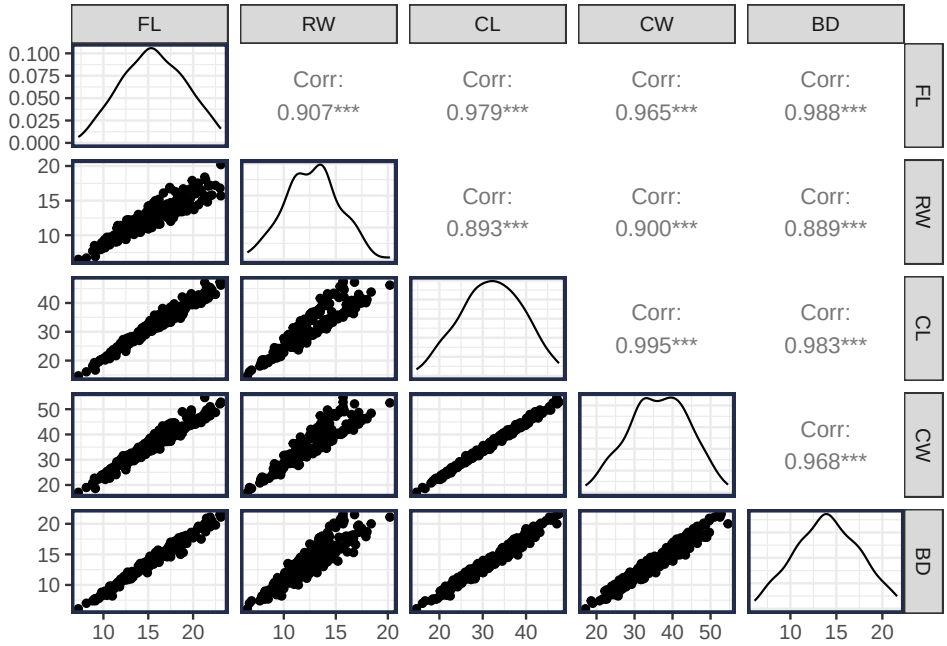
Campbell, N.A. and Mahon, R.J. (1974) collected 5 morphological measurements on 200 crabs from the species *Leptograpsus variegatus*. These are recorded in the `crabs` dataset in the `MASS` R package.

```
crabs = MASS::crabs
```

FL	RW	CL	CW	BD
15.0	11.9	32.5	37.2	13.6
16.3	11.6	31.6	34.2	14.5
17.1	14.5	33.1	37.2	14.6
12.0	11.1	25.4	29.2	11.0
13.1	10.6	28.2	32.3	11.0
14.7	11.1	29.0	32.1	13.1

- columns
  - `FL`, `RW`, `CL`, `CW`, `BD` morphological measurements
  - `index` not important for us
  - see `?MASS::crabs` for details

We will construct clustering based on the five morphological features



1.5 Example: Old Faithful

The old faithful geyser in Yellowstone National Park is one of the most regular geysers in the park. The waiting time between eruptions is between 35 and 120 mins.

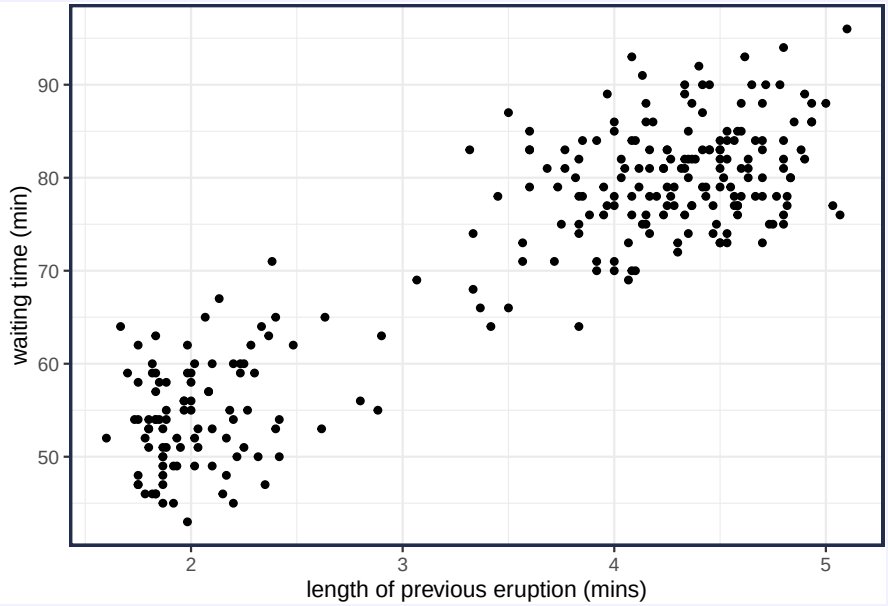
Live Streaming Webcam with eruption predictions

Because the nearby Yellowstone Lodge is nice and warm in the winter, and serves good ice cream in the summer, you may be distracted from stepping outside to watch the eruption. Let's see if we can determine the best time to leave the cozy lodge and go outside to watch the next eruption.

Your Turn #1 : Old Faithful

The `eruptions` column is the *duration of the previous eruption* and the `waiting` column is the *time between eruptions.*

```
#: Load the Old Faithful data
X = datasets::faithful

#: Scatterplot
ggplot(X) +
  geom_point(aes(eruptions, waiting)) +
  labs(
    x = "length of previous eruption (mins)",
    y = "waiting time (min)"
  )
```



1. What patterns do you see?
2. How many clusters are there? Interpret each cluster.
3. How do the clusters relate to density estimation?

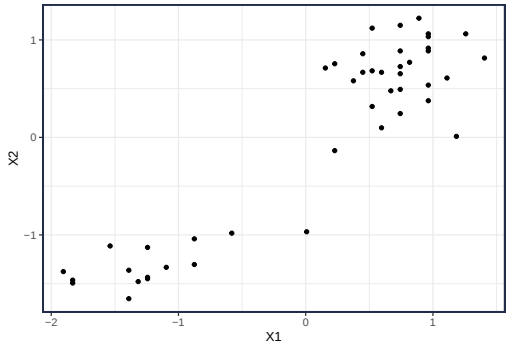
2 Hierarchical Clustering

Hierarchical clustering creates a set of *hierarchical* (or nested) clusters based on a *dissimilarity/distance* metric.

The resulting structure is represented by a dendrogram (*tree-drawing*), which resembles an upside-down tree.

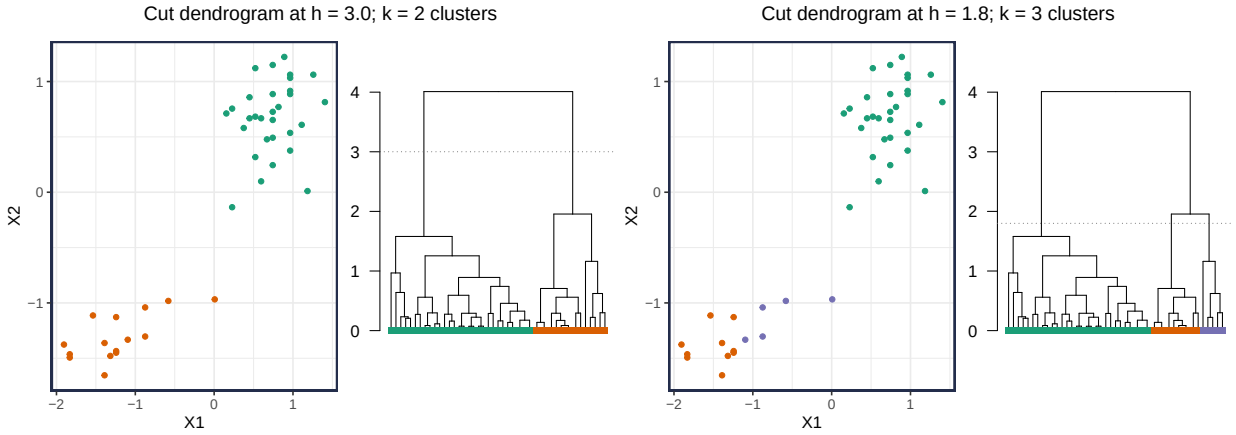
2.1 Dendrogram

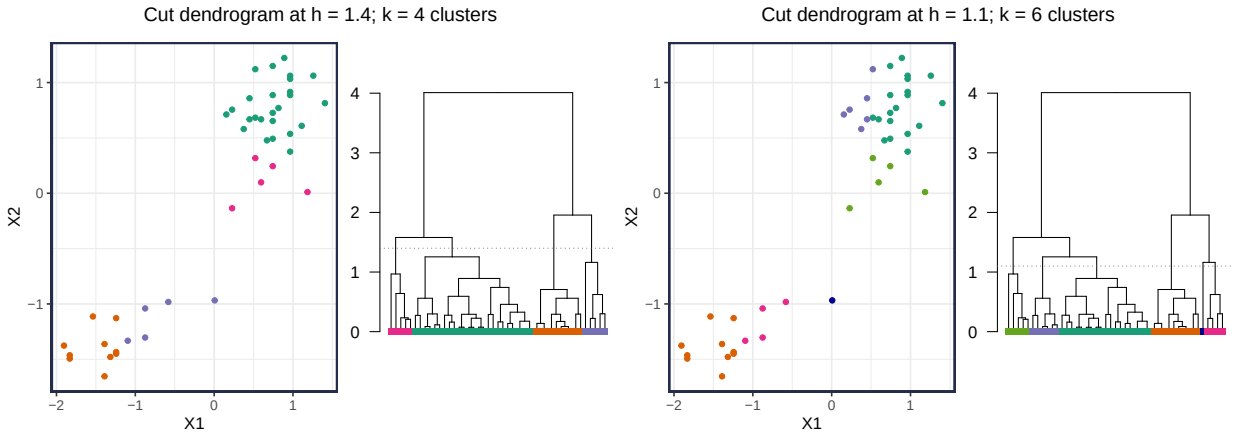
Below are 45 observations, in 2D space.



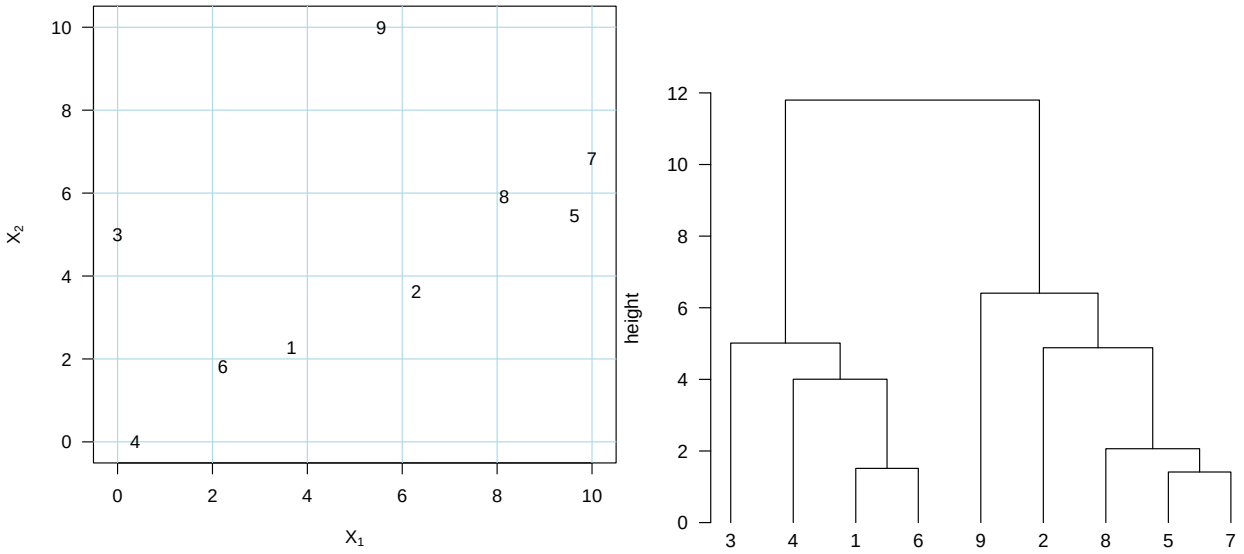
The following dendrograms are constructed from *agglomerative hierarchical clustering*:

- The *pairwise dissimilarity* is defined as the Euclidean distance between points.
- The *cluster dissimilarity* is defined as the largest dissimilarity between two clusters (complete linkage).
- The heights on the y-axis show the dissimilarity between clusters, not the number of clusters.





2.1.1 Dendrogram Interpretation



Distance Matrix

	1	2	3	4	5	6	7	8
2	2.96							
3	4.57	6.44						
4	4.00	6.95	5.01					
5	6.76	3.80	9.64	10.75				
6	1.51	4.46	3.88	2.60	8.25			
7	7.80	4.88	10.16	11.80	1.41	9.25		
8	5.77	2.93	8.20	9.77	1.55	7.20	2.06	
9	7.95	6.41	7.47	11.26	6.10	8.83	5.47	4.84

2.2 Agglomerative (Greedy) Hierarchical Clustering Algorithm

The basic hierarchical clustering algorithm takes a *greedy*, *sequential* approach to forming the nested groups.

Algorithm: Agglomerative Hierarchical Clustering

Initialize

1. Begin with n observations and treat each observation as a unique cluster. Set the number of clusters $k = n$.
2. Calculate the *dissimilarity* between all $\binom{n}{2}$ clusters.

Iterate: For $k = n-1, n-2, \ldots, 2$:

3. Merge the most *similar* (or least *dissimilar*) clusters.
4. Update the pairwise dissimilarity between the new cluster and all existing clusters.
5. Set $k = k - 1$

2.2.1 Cluster Dissimilarity

There are a few common approaches to calculate the dissimilarity between *clusters* (or sets).

The first three are based on the pairwise dissimilarity/similarity between observations.

- Let A and B be two clusters (collection of points) and d_{ij} the dissimilarity/distance between observations i and j.

Single Linkage/Nearest Neighbor

The cluster dissimilarity is the *smallest* dissimilarity between pairs in the two sets

$$D_{\text{single}}(A, B) = \min\{d_{ij} : i \in A, j \in B\}$$

Complete Linkage/Farthest Neighbor

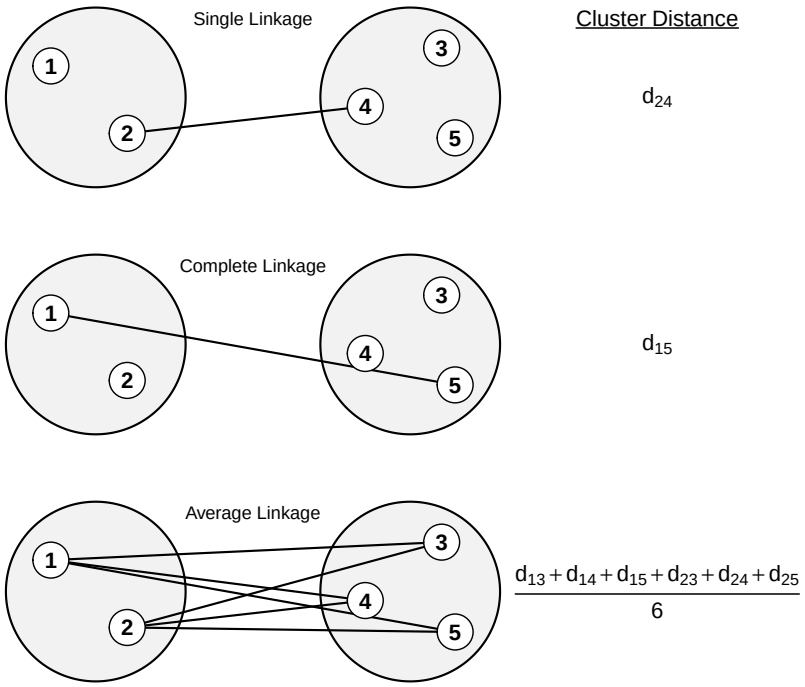
The cluster dissimilarity is the *largest* dissimilarity between pairs in the two sets

$$D_{\text{complete}}(A, B) = \max\{d_{ij} : i \in A, j \in B\}$$

Average Linkage

The cluster dissimilarity is the *average* dissimilarity between pairs in the two sets

$$D_{\text{avg}}(A, B) = \text{avg}\{d_{ij} : i \in A, j \in B\}$$



Another common approach is to base the cluster dissimilarity score on the *centroid* and *compactness* of the observations in the cluster.

Centroid Linkage

The cluster dissimilarity is the *distance between the cluster centroids*.

$$D_{\text{centroid}}(A, B) = \|m_A - m_B\|$$

- m_A is the centroid of set A
- L_2 norm: $\|x\|_2 = \left(\sum_j x_j^2\right)^{1/2}$ (Euclidean distance)

Ward's Linkage

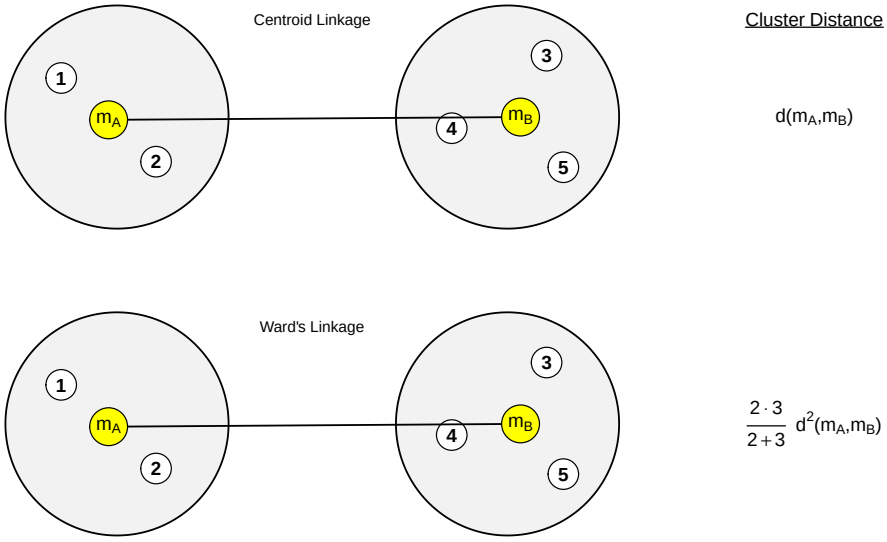
The cluster dissimilarity is the *increase in sum of squares* if the clusters were merged.

Let A and B be two clusters. Ward's linkage uses the dissimilarity score

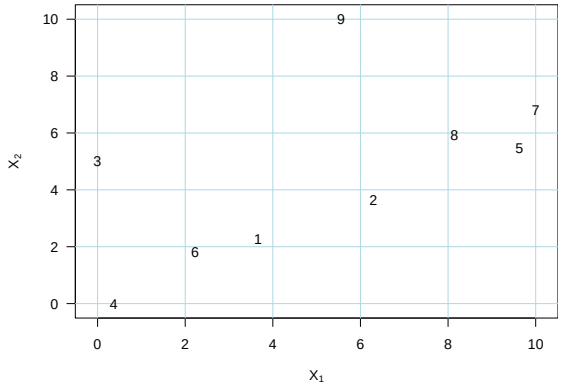
$$\begin{aligned} D_{\text{ward}}(A, B) &= \sum_{i \in A \cup B} \|x_i - m_{A\cup B}\|_2^2 - \sum_{i \in A} \|x_i - m_A\|_2^2 - \sum_{i \in B} \|x_i - m_B\|_2^2 \\ &= \frac{n_A n_B}{n_A + n_B} \|m_A - m_B\|_2^2 \end{aligned}$$

- $x = (x_1, \ldots, x_p)$
- m_A is the centroid of set A
- L_2 norm: $\|x\|_2 = \left(\sum_j x_j^2\right)^{1/2}$
  - $\|x\|_2^2$ is the *squared* Euclidean distance

- See (Murtagh and Legendre 2011) for more details and R implementation (`ward.D` and `ward.D2`)

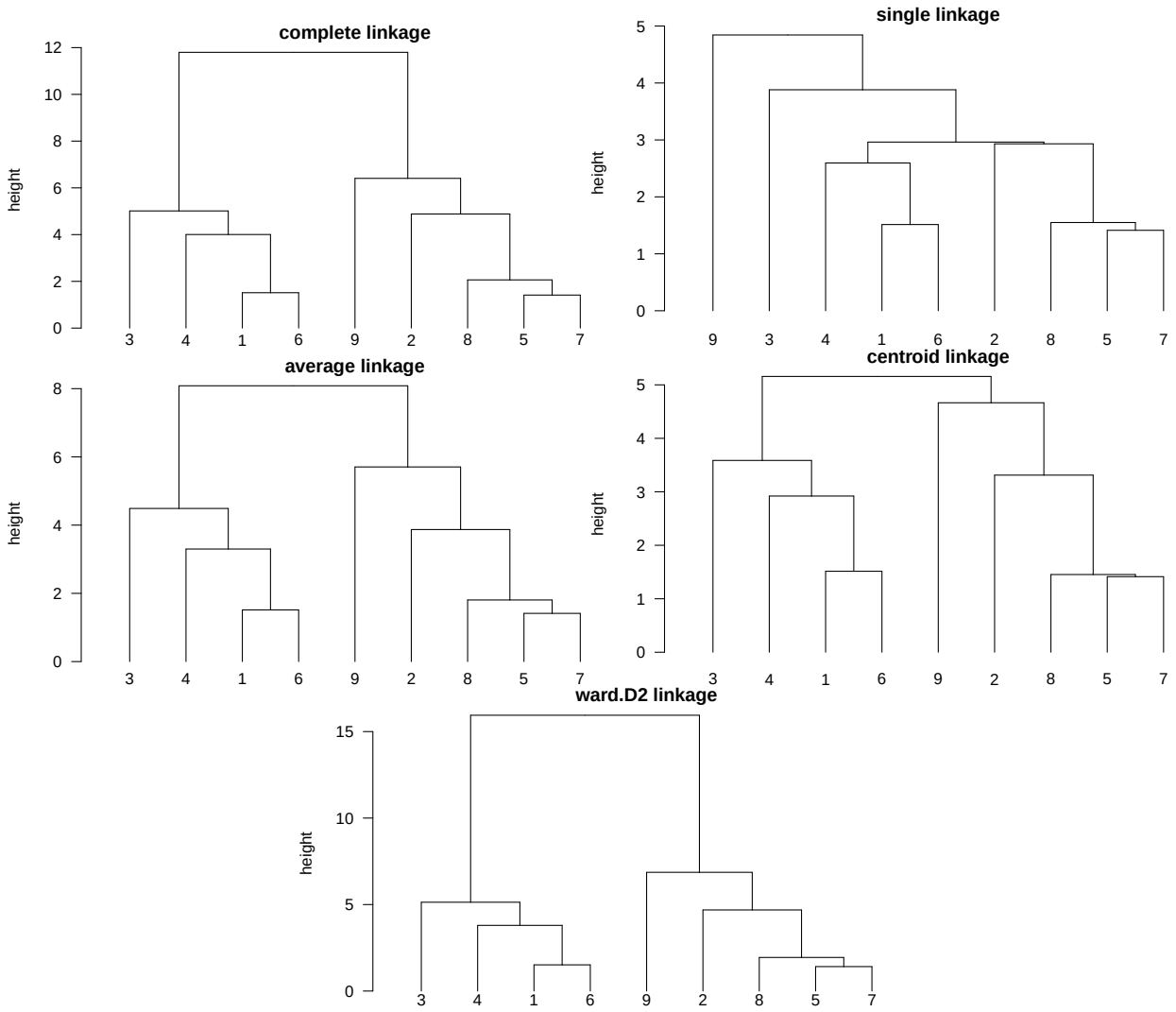


2.2.2 Example



Distance Matrix

	1	2	3	4	5	6	7	8
2	2.96							
3	4.57	6.44						
4	4.00	6.95	5.01					
5	6.76	3.80	9.64	10.75				
6	1.51	4.46	3.88	2.60	8.25			
7	7.80	4.88	10.16	11.80	1.41	9.25		
8	5.77	2.93	8.20	9.77	1.55	7.20	2.06	
9	7.95	6.41	7.47	11.26	6.10	8.83	5.47	4.84



2.2.3 R Implementation

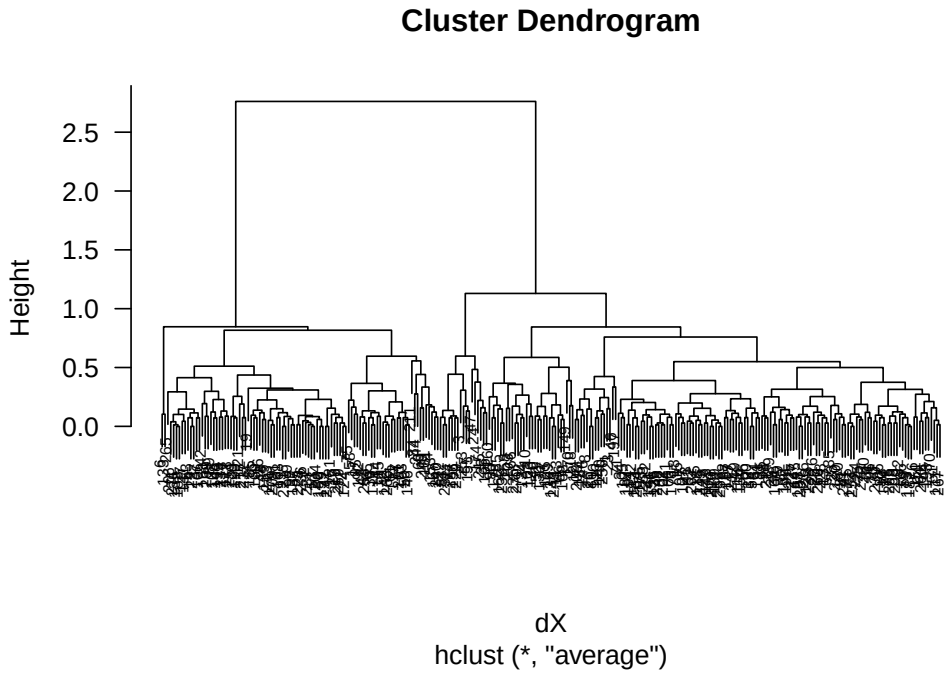
```
#: Load Data and scale (mean=0, sd=1)
X = datasets::faithful %>% scale() %>% as_tibble()
```

- The R function `hclust()` will run basic Hierarchical Clustering.
- It takes a *distance* object. A distance object is generated by calling `dist()` on a matrix or data frame.
  - This contains all pairwise distances between the rows of your data.
  - Data must be numeric data and have no missing values.
  - Several different types of distances are available (e.g., Euclidean, manhattan). Whenever you see distances being used think about scaling!

```
dX = dist(X, method="euclidean")       # calculate distance
hc = hclust(dX, method="average")      # use average linkage
str(hc, 1)
#> List of 7
#>  $ merge      : int [1:271, 1:2] -11 -14 -26 -38 -72 -93 -104 -135 -138 -141 ...
#>  $ height     : num [1:271] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ order      : int [1:272] 6 133 265 206 271 135 188 127 131 63 ...
```

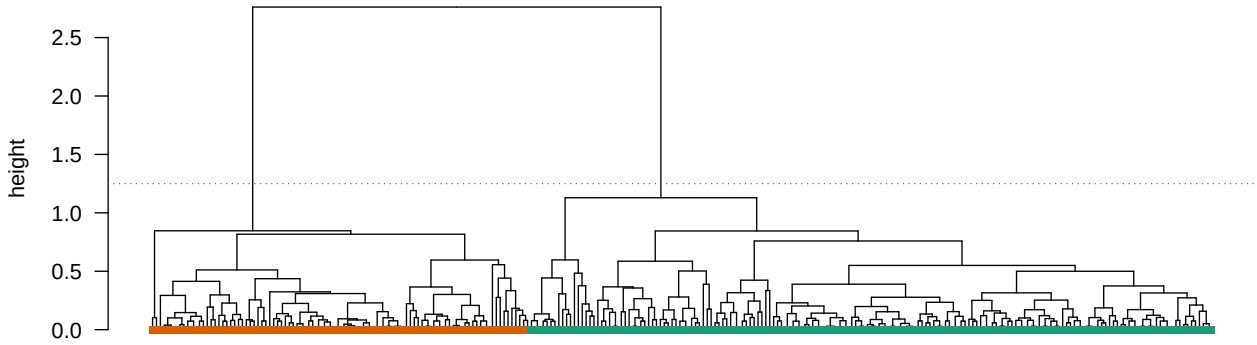
```
#>  $ labels      : NULL
#>  $ method      : chr "average"
#>  $ call        : language hclust(d = dX, method = "average")
#>  $ dist.method: chr "euclidean"
#>  - attr(*, "class")= chr "hclust"
```

A basic dendrogram can be obtained with `plot()`



- Some additional visualization is available if `hc` is converted to a `dendrogram` object

```
colPalette = c('#1b9e77','#d95f02','#7570b3','#e7298a','#66a61e')
clusters = cutree(hc, k=2)

plot(as.dendrogram(hc), las=1, leaflab="none", ylab="height")
ord = hc$order
labels = clusters[ord]
colors = colPalette[labels]
shapes = 15 #ifelse(str_detect(labels, "F"), 15, 17)
n = length(labels)
points(1:n, rep(0, n), col=colors, pch=shapes, cex=.8)
abline(h = 1.25, lty=3, col="grey40")
```



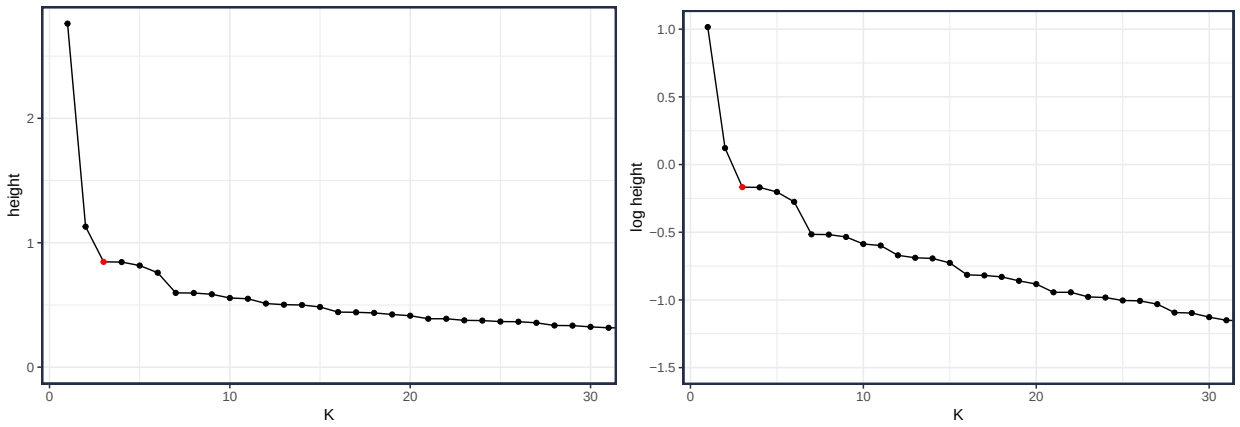
2.3 Choosing the number of clusters, K

- There are **many** approaches; see ITDM Chapter 7.5 for some approaches.

2.3.1 Elbow method

- One approach is to look for regions in the dendrogram where gaps/changes appear in the height (or log height) of merges.
  - The height corresponds to the dissimilarity between the merged clusters, so a large jump in height corresponds to a high dissimilarity between the solutions for $k+1$ and k clusters
  - For Ward's method this corresponds to the change in the Sum of Squared Errors (SSE) for a merge
  - For Complete Linkage, the height corresponds to the dissimilarity between the most dissimilar points in the two clusters

```
tibble(height = hc$height, K = row_number(-height)) %>%
  ggplot(aes(K, height)) +
  geom_line() +
  geom_point(aes(color = ifelse(K == 3, "red", "black"))) +
  scale_color_identity() +
  coord_cartesian(xlim=c(1, 30))

tibble(height = log(hc$height), K = row_number(-height)) %>%
  ggplot(aes(K, height)) +
  geom_line() +
  geom_point(aes(color = ifelse(K == 3, "red", "black"))) +
  scale_color_identity() +
  coord_cartesian(xlim=c(1, 30), y=c(-1.5, NA)) +
  labs(y = "log height")
```



Your Turn #2

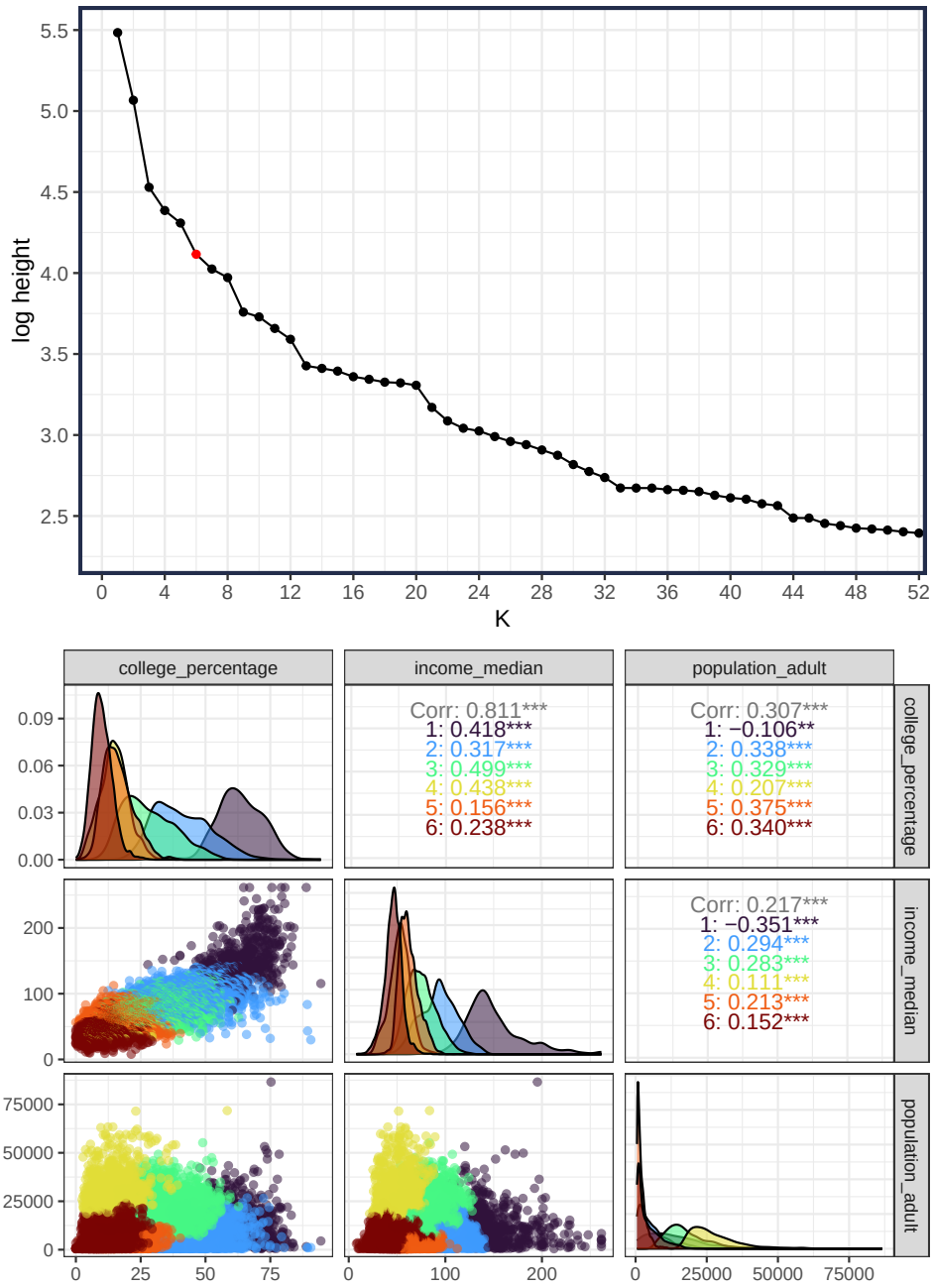
1. Where do you have to cut the dendrogram to get $K = 3$ clusters?

- The function `cutree()` will extract the membership vector for a given *k* clusters or *h* height.

```
#: cluster membership by specifying number of clusters
cutree(hc, k=3)  %>% table()
.
  1   2   3
```

```
158  97  17

#: cluster membership by specifying height
cutree(hc, h=0.8465) %>% table()
.
  1   2   3
158  97  17
```

2.3.2 SuperZip data



2.3.3 Other methods for choosing K

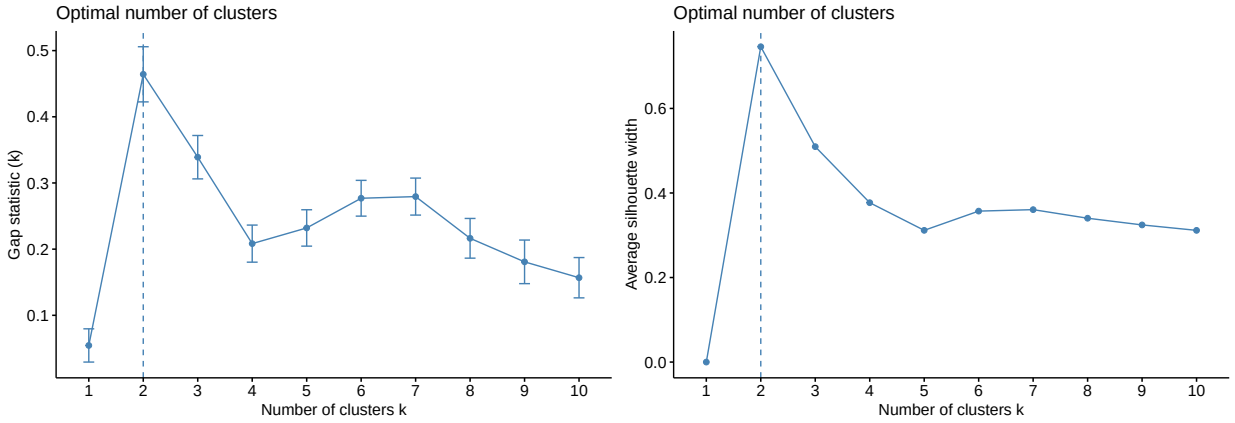
- There are several, more quantitative, methods to select K. However, they all make strong assumptions.
- See Gap Statistic paper for an approach that compares each solution to what is obtained in a *null* model (of no clustering)
- See Silhouette Score for an approached based on comparing the average distances from a point to all other points in the same cluster and to all points in the nearest cluster.

2.3.4 R package: `factoextra`

The `factoextra` package has functions to calculate and plot the gap statistic and silhouette score. First, a helper function is needed that has two arguments, the first is the data and the second is the number of clusters:

```
hier_clusters <- function(X, k = 5){
  dX = dist(X, method="euclidean")
  hc = hclust(dX, method="average")
  tibble(cluster = cutree(hc, k = k))
}
```

```
# Old Faithful data
library(factoextra)
X = datasets::faithful %>% scale() %>% as_tibble()
set.seed(2024)
fviz_nbclust(X, hier_clusters, method = "gap_stat")
fviz_nbclust(X, hier_clusters, method = "silhouette")
```



2.4 Details and Considerations

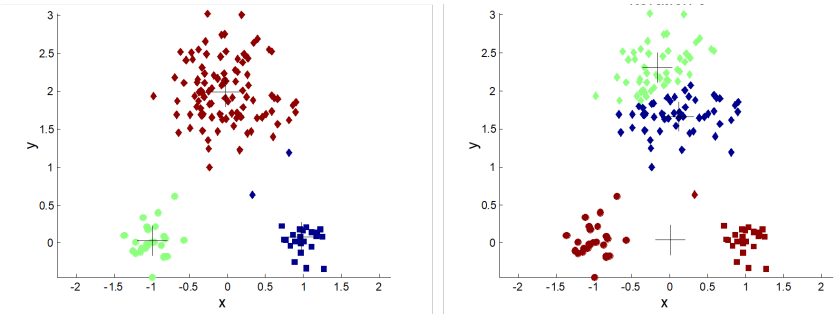
- Choice of dissimilarity/distance can be crucial
- Should the variables/features be standardized? E.g.,
  - Scale so all features have mean of zero and standard deviation of one. In R, this can be performed with the `scale()` function.
  - Scale to be between $[0, 1]$
  - Scale by quantile/rank
- Should all the features/variables be used in to calculate the distance? Or more generally, should the features/variables be weighted?
  - See ESL 14.3.3 for an interesting discussion on feature weights

- Other transformations
  - PCA (Principal component analysis). See ISL chapter 10.1.
  - Autoencoders (based on ANNs)
- What type of linkage should be used?
- What value of K to use?
- **These can have a substantial impact on your resulting analysis**
- View clustering results with skepticism
  - Results may not have much stability. Vary the data or tuning parameters just a little and a very different solution will appear.
  - Most clustering results that you will see are based on trying many different K, distance/dissimilarity, and linkage methods, to get a pleasing solution

3 K-means clustering

3.1 Prototype Methods

- Instead of seeking a hierarchical clustering structure, *prototype methods* seek a set of K vectors $(m_1, \ldots, m_K)$ that best represent the n observations.
  - The prototypes don't have to be existing observations
  - The K prototypes can be thought of as representing K *clusters*
  - Prototype methods can be used for data reduction (see ESL 14.3.9)



- The prototypes ($\{m_k\}$) should be determined by optimization. The prototypes should *best represent* the data.
- *Best* is, of course, determined by the application
  - For data compression, the choice of K and $\{m_k\}$ is based on the trade-off between fidelity and storage size.
  - For clustering, the choice of prototypes can give us different insights into the data structure
- The following concepts emerge:
  1. Each observation should be represented by the prototype that *best represents* it.
  2. The prototypes should be determined so that they *best represent* the observations assigned to it.

3.2 K-means

- K-means formalizes these two concepts into an algorithm
- In K means, the K cluster *centroids* are the prototypes
  1. The kth centroid represents the n_k observations assigned to that cluster
  2. An observation is assigned to the centroid that is *nearest*

Algorithm: K means

Initialize

a. Choose number of clusters K

b. Choose K initial centroids

- $\{m_k\}_{k=1}^K$

Repeat until convergence:

1. Assign the observations to the *nearest* centroid (using squared Euclidean distance).

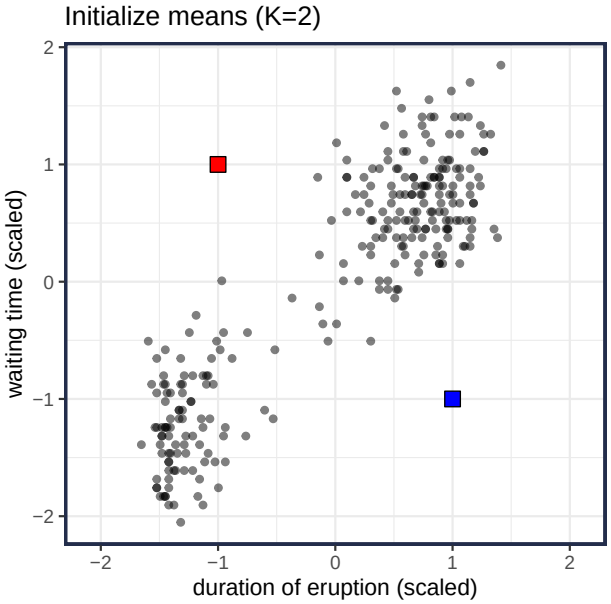
$$g_i = \arg\min_{1 \le k \le K} \|x_i - m_k\|^2$$

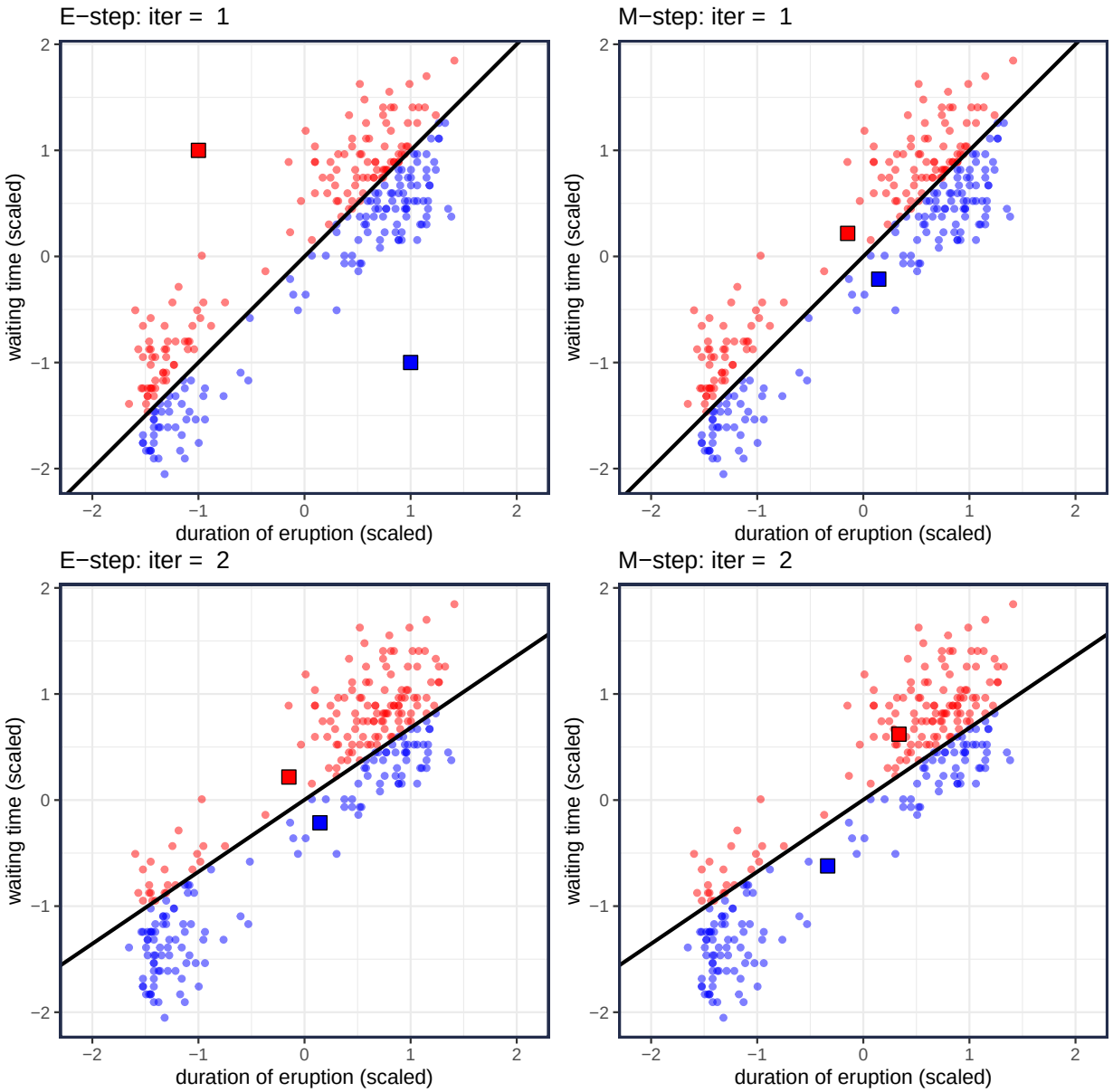
2. Update the centroids according to the points assigned to them.

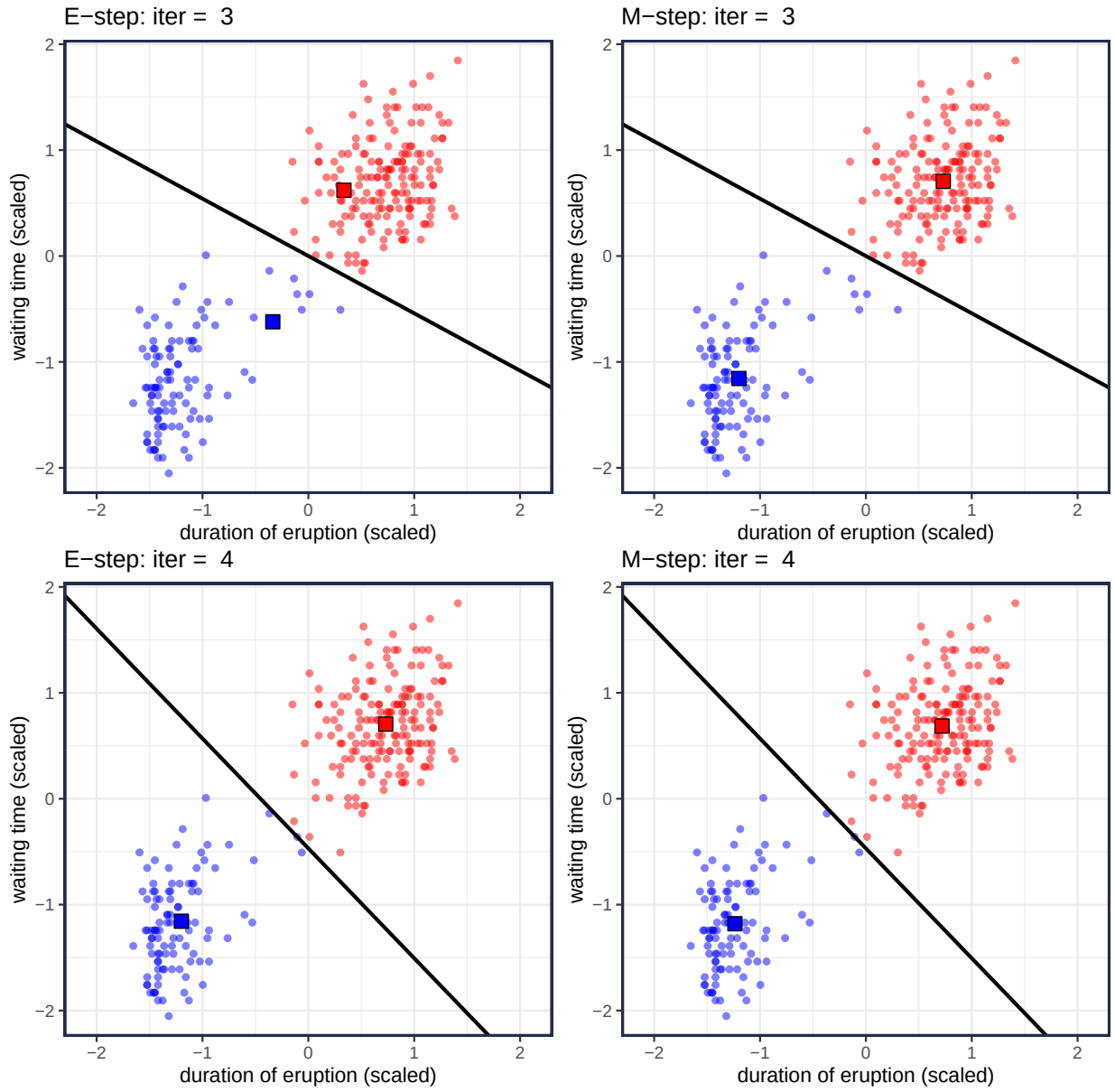
$$m_k = \arg\min_m \sum_{i:g_i=k} \|x_i - m\|^2$$

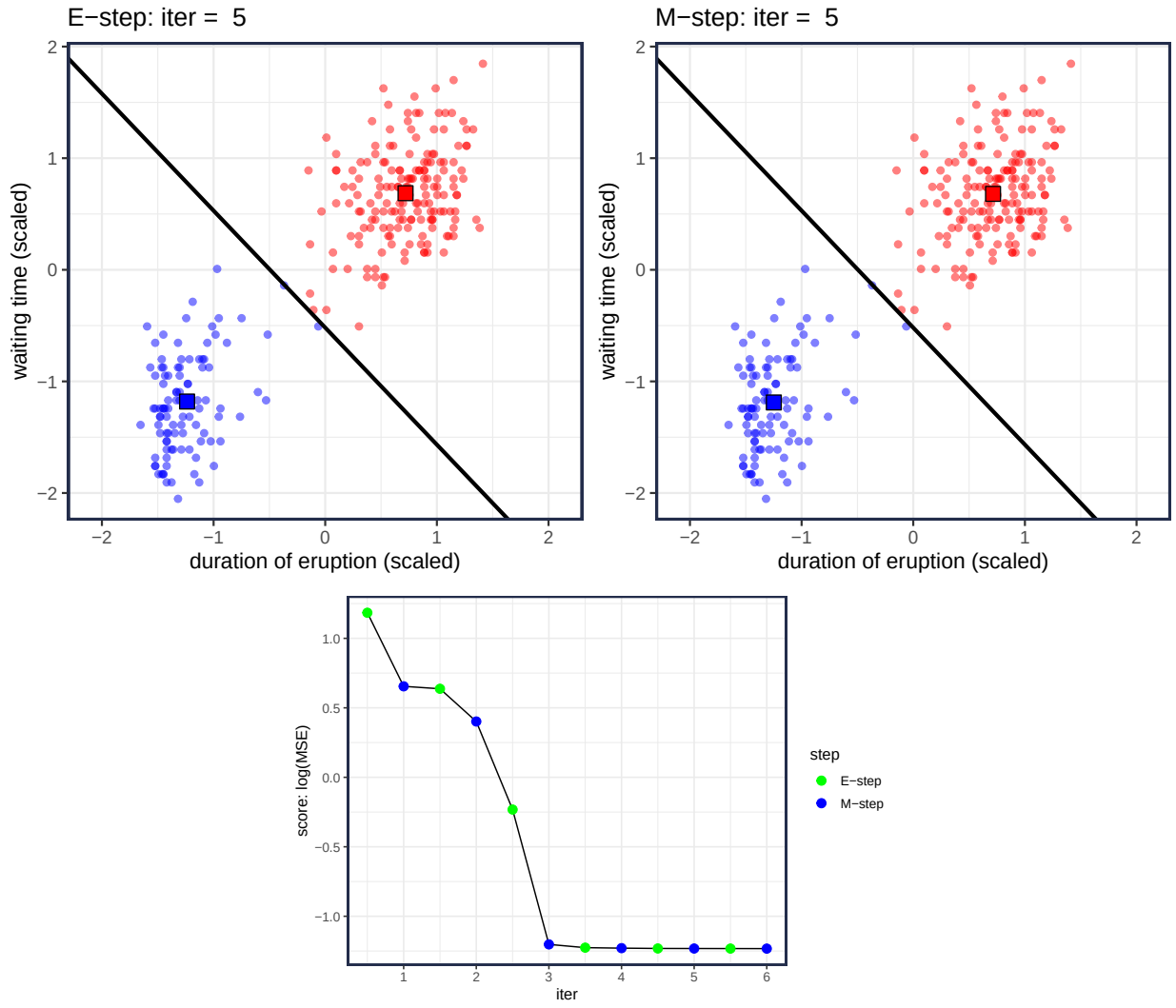
- Notation:
  - $x_i \in \mathbf{R}^p$ is the i^{th} observation (point in p dimensional Euclidean space)
  - $m_k \in \mathbf{R}^p$ is the k^{th} prototype
  - $g_i \in 1, 2, \ldots, K$ is the cluster label for observation i
  - The L_2 norm: $\|x - y\| = \left(\sum_{j=1}^p (x_j - y_j)^2\right)^{1/2}$

3.2.1 Example: Old Faithful



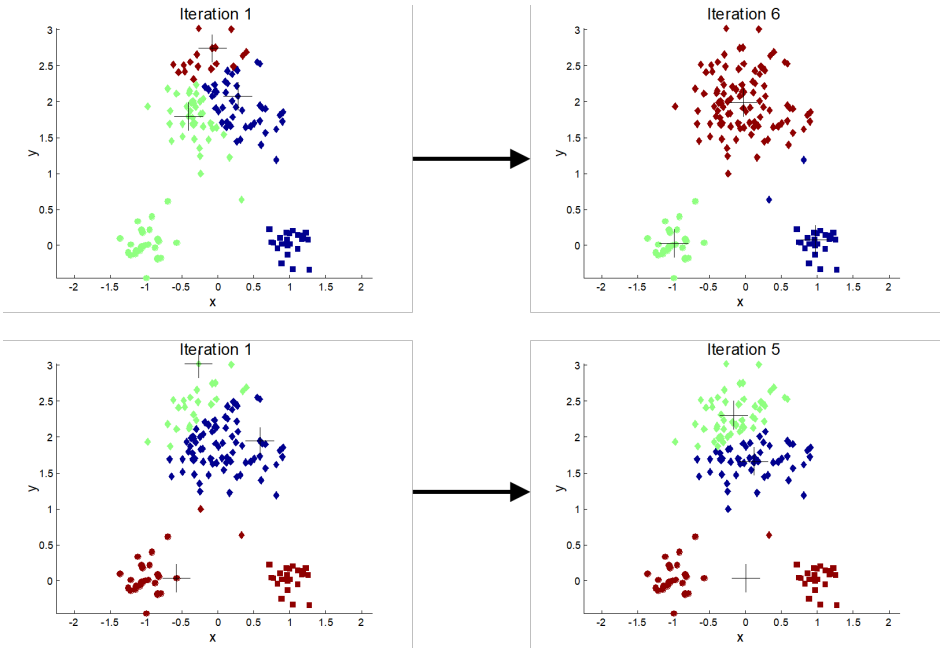

E−step: iter = 1
M−step: iter = 1
E−step: iter = 2
M−step: iter = 2
waiting time (scaled)
duration of eruption (scaled)
−2
−1
0
1
2


E−step: iter = 3
M−step: iter = 3
E−step: iter = 4
M−step: iter = 4
waiting time (scaled)
duration of eruption (scaled)
−2
−1
0
1
2



3.3 Initialization

- K-means may only find *local* solutions
- Important to run with several initializations
  - Use the initialization that gives the best performance
- There are some strategies to help
  - initialize with hierarchical clustering
  - sequentially choose prototypes that are farthest away from existing centroids



3.4 Choosing K

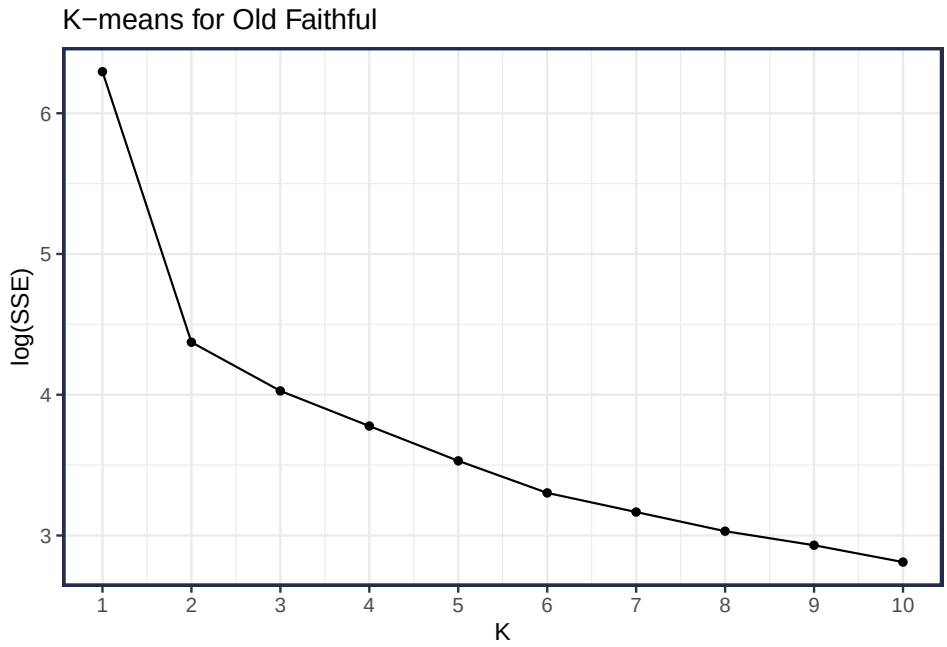
- Run K-means for several values of K and examine the SSE (sum of squared error); also known as *within cluster scatter* or *within group sum of squares*.
  - Or mean squared error (MSE = SSE/n)

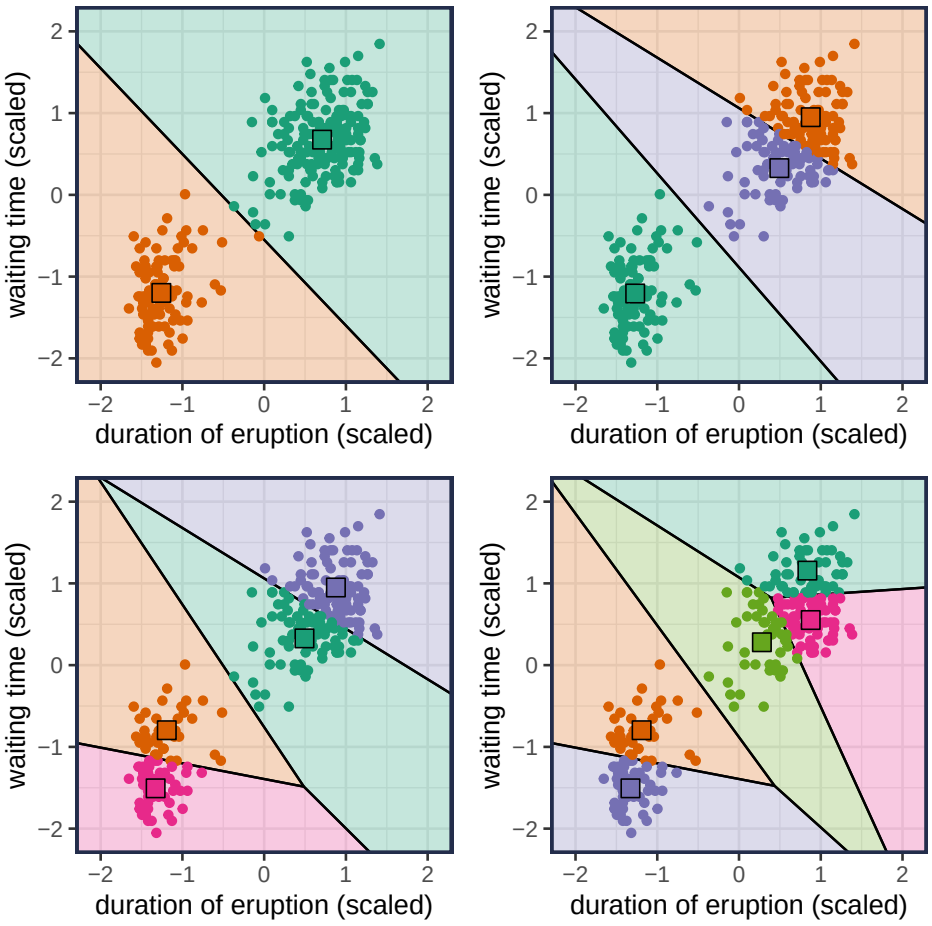
$$\begin{aligned}\text{SSE} &= \sum_{k=1}^{K} \sum_{i:g_i=k} \|x_i - m_k\|^2 \\ &= \sum_{k=1}^{K} \sum_{i=1}^{n} \mathbb{1}(g_i = k) \|x_i - m_k\|^2 \\ &= \sum_{k=1}^{K} W(k)\end{aligned}$$

- $W(k)$ is the SSE for cluster k
  - Notice that each step, both E and M, will reduce (technically, will not increase) the SSE
- Same warnings as with hierarchical clustering

```
#-- Load Data and scale (mean=0, sd=1)
X = datasets::faithful %>% scale() %>% as_tibble()

#-- Run kmeans for multiple K
Kmax = 10                                   # maximum K
SSE = numeric(Kmax)                         # initiate SSE vector
set.seed(2022)                              # set seed for reproducibility
for(k in 1:Kmax){
  km = kmeans(X, centers=k, nstart=25)      # use 25 initializations
  SSE[k] = km$tot.withinss                  # get SSE
}

#-- Plot results
```

```r
tibble(K = 1:Kmax, SSE) %>%
  ggplot(aes(K, log(SSE))) +
  geom_line() +
  geom_point() +
  scale_x_continuous(breaks = 1:Kmax) +
  labs(title = "K-means for Old Faithful")

# plot(1:Kmax, log(SSE), type='o', las=1, xlab="K")
# title("K-means for Old Faithful")
```



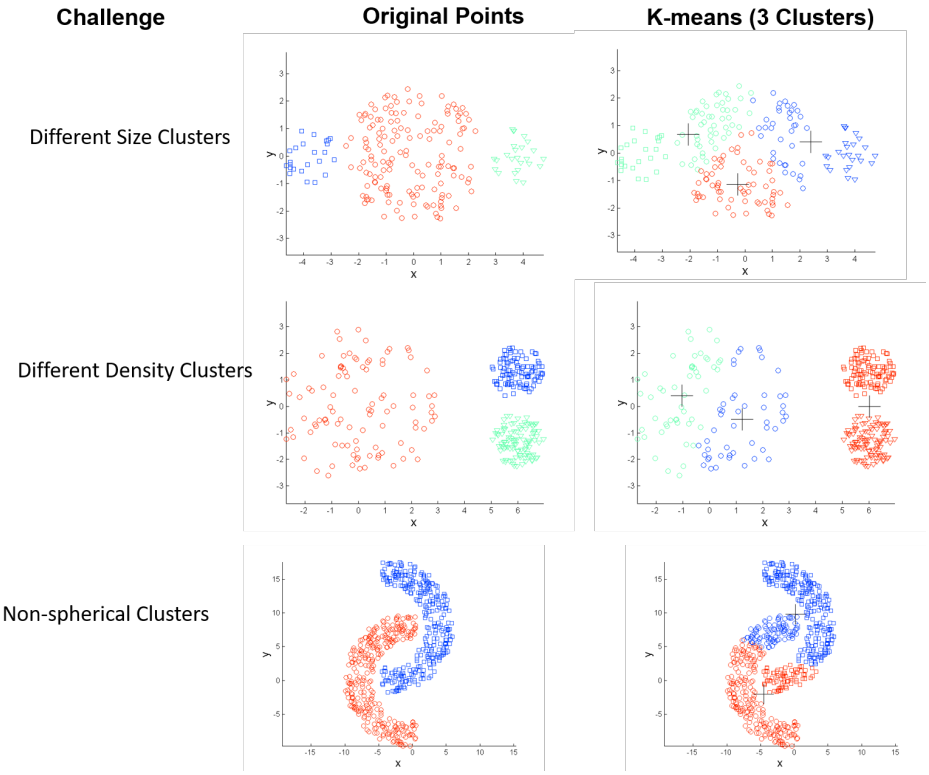


Methods for choosing K

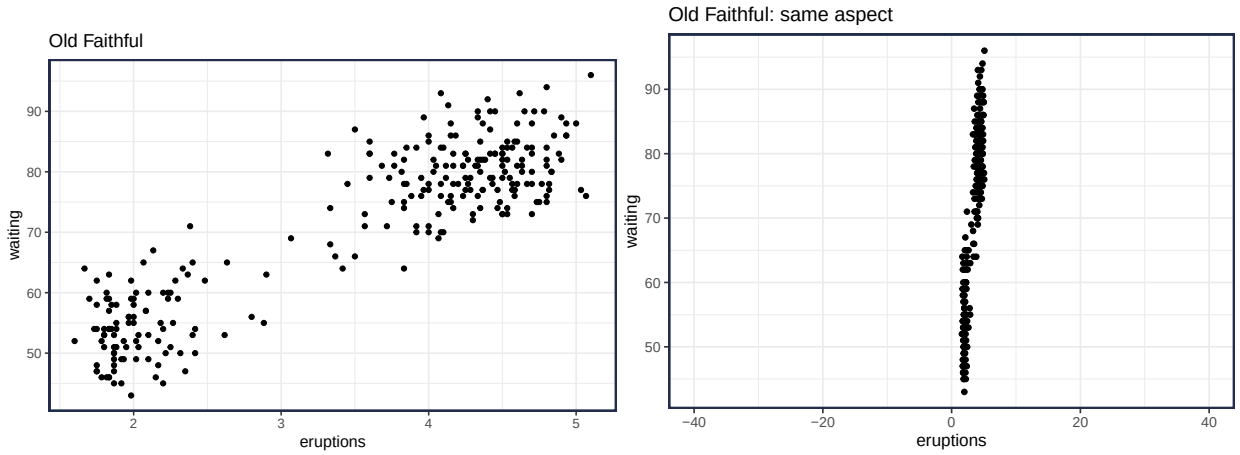
1. Look for *elbow* in plot of log(SSE) vs. K
   - The log usually shows sharper elbows. Stronger connection to Gaussian log-likelihood
2. Use one of many statistical type tests (e.g. [Hamerly & Elkan, 2003])
3. Instead of K means, use the more flexible Gaussian Mixture Models!
   - Then use AIC/BIC and other statistical measures to select K

3.5 K means finds balanced, spherical clusters

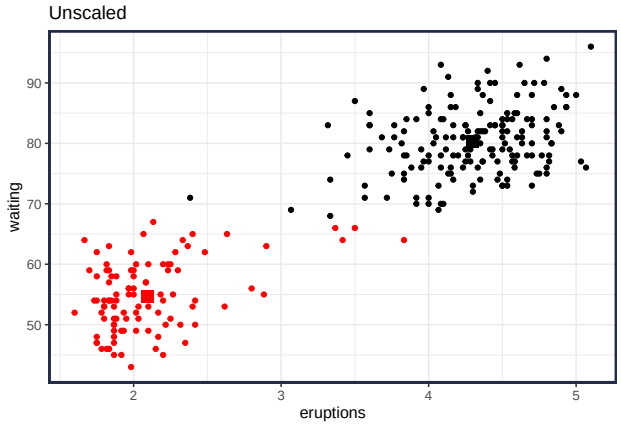
- K means tends to find *balanced* clusters
  - clusters are around the same size (number of observations)
  - there is no mechanism to permit small/large clusters; everything is based on SSE
- K means tends to find *spherical* clusters
  - because Euclidean distance is used, clusters will be more spherical; larger K is needed to fit non-spherical shapes
  - **Importance of scaling to treat each variable equivalently** using Euclidean distance
- Important to scale appropriately. E.g., standardize all columns to have equal variance.
  - In R, the function `scale(X)` will transform all columns to have mean 0 and variance of 1.

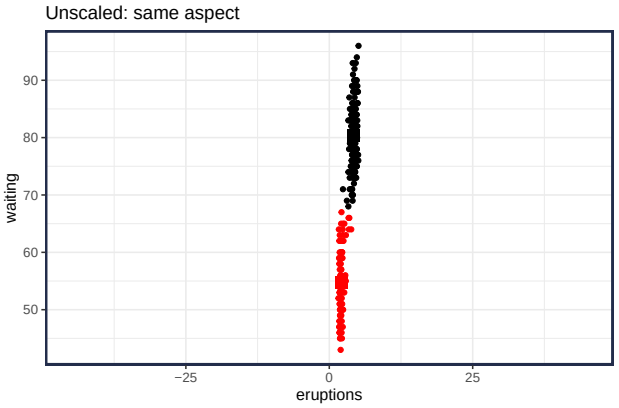


3.5.1 Old Faithful Example

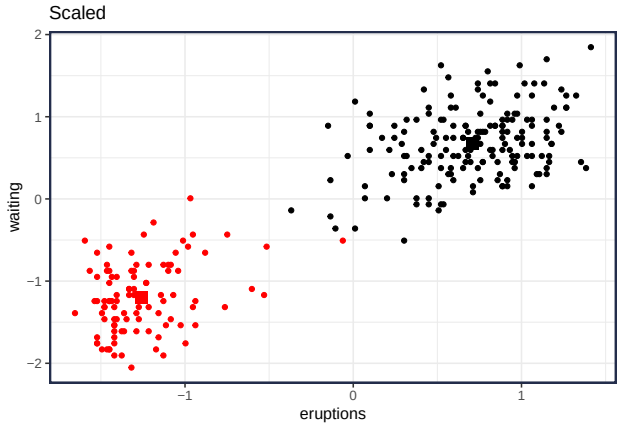


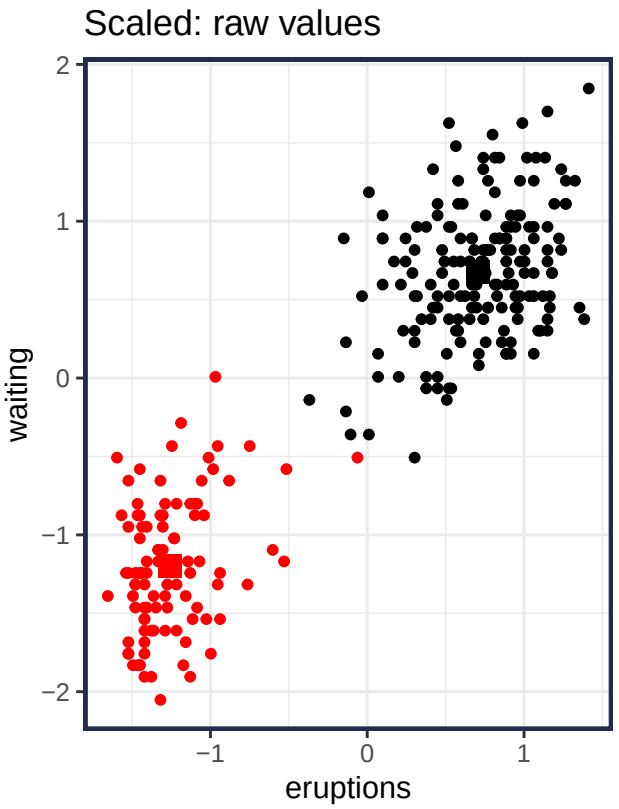
Unscaled Solution





Scaled Solution (mean=0, sd=1)





Note

- Notice when we **do not** scale the data, the waiting time dominates the distance calculation
  - Thus, the *eruption* variable has practically no impact on the resulting clustering (i.e., clustering is only based on *waiting time*)
- By scaling the data (so each feature has the same variance), the resulting clustering can better adapt to the natural structure
- Gaussian Mixture Models provide a natural way to handle input features that have different scales
  - I.e., **use mixture models instead of k-means**

3.6 R Code for K-means

The R function `kmeans()` implements the K means algorithm.

- Inputs (primary):
  - `x`: numeric matrix or data frame. No missing values allowed. You probably want `x` to be scaled.
  - `centers`: Two options: (1) the number of clusters or (2) a set of initial centers. Default uses (1) and selects random rows of `x` to initiate.
  - `nstart`: (only used if `centers` is a number) the number of times to re-run the algorithm with different initialization. Will return the run with the best performance.
- Outputs (primary):
  - `cluster`: integer representing the cluster label
  - `centers`: the cluster centers
  - `size`: number of points in each cluster
  - performance metrics: `withinss`, `tot.wighinss`, `betweenss`, `totss`

```
set.seed(2001)
crabsX %>%                  # data
  scale() %>%               # scale the data
  kmeans(centers = 4, nstart = 100) # run k-means with 4 centers; 100 different initializations
#> K-means clustering with 4 clusters of sizes 28, 41, 60, 71
#>
#> Cluster means:
#>         FL      RW      CL      CW      BD
#> 1 -1.5533 -1.5791 -1.6046 -1.6143 -1.5700
#> 2  1.3767  1.2778  1.3597  1.3388  1.3855
#> 3 -0.6098 -0.5519 -0.5947 -0.5919 -0.5861
#> 4  0.3329  0.3512  0.3502  0.3637  0.3143
#>
#> Clustering vector:
#>   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
#>   1   1   1   1   1   1   1   1   1   3   3   3   3   3   3   3   3   3   3   3
#>  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
#>   3   3   3   3   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
#>  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
#>   4   4   4   2   2   2   2   2   2   2   1   1   1   1   1   1   1   1   1   1
#>  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
#>   1   1   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
#>  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
#>   3   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   2   2   2   2
#> 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
#>   1   1   1   1   1   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
#> 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
#>   3   3   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   2   2   2
#> 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
#>   2   2   2   2   2   2   2   2   2   2   1   1   3   3   3   3   3   3   3   3
#> 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
#>   3   4   4   4   4   4   4   4   4   4   4   4   4   4   4   2   4   4   4   4
#> 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
#>   4   4   4   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
#>
#> Within cluster sum of squares by cluster:
#> [1] 16.55 38.09 27.58 43.49
#>  (between_SS / total_SS =  87.4 %)
#>
#> Available components:
#>
```

```
#> [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
#> [6] "betweenss"    "size"         "iter"         "ifault"
```

The `broom` package provides some nice functions to quickly get relevant info. See the tidy kmeans tutorial for details.

```
library(broom) # for tidy(), augment(), glance()
```

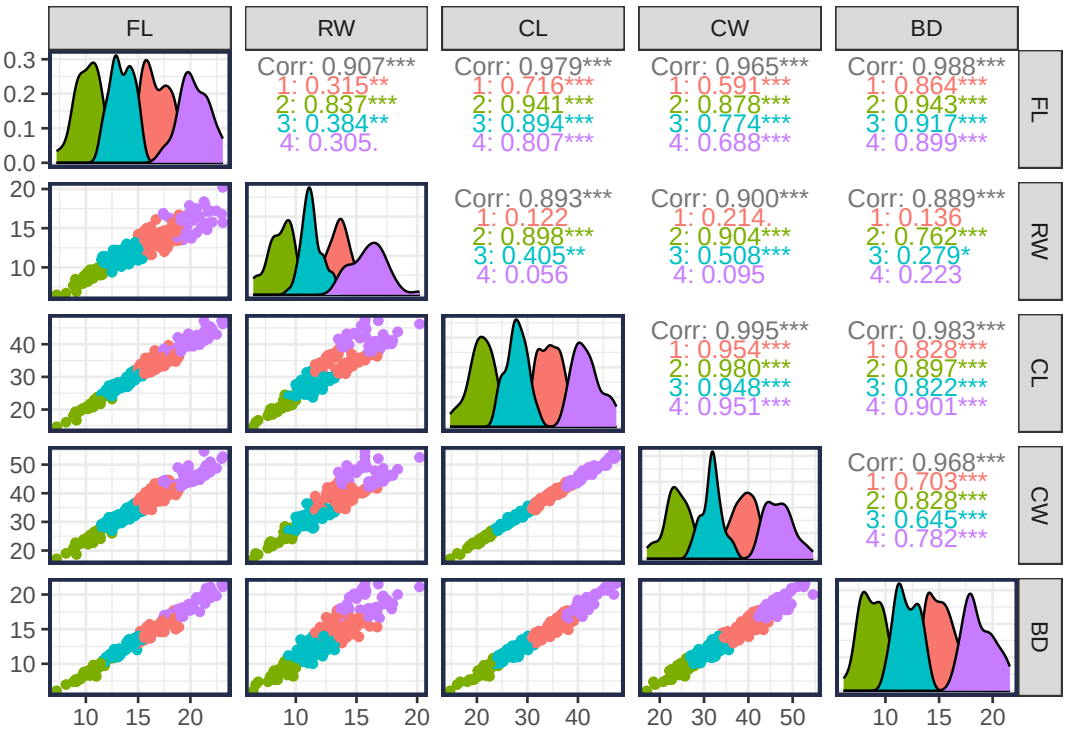
```
#-- Scale data and implement kmeans
scaled_data = scale(crabsX)    # scale data
set.seed(2021)
fit = kmeans(scaled_data, centers = 4, nstart = 100)  # run kmeans
```

```
#-- augment() adds the cluster labels to data
broom::augment(fit, scaled_data)
#> # A tibble: 200 x 6
#>      FL    RW    CL    CW    BD .cluster
#>   <dbl> <dbl> <dbl> <dbl> <dbl> <fct>
#> 1 -2.14 -2.35 -2.25 -2.21 -2.05 2
#> 2 -1.94 -1.96 -1.97 -1.98 -1.94 2
#> 3 -1.83 -1.92 -1.84 -1.78 -1.85 2
#> 4 -1.71 -1.88 -1.69 -1.69 -1.70 2
#> 5 -1.65 -1.84 -1.66 -1.70 -1.70 2
#> 6 -1.37 -1.45 -1.28 -1.26 -1.24 2
#> # i 194 more rows
```

```
#-- tidy() summarizes the info about each cluster
broom::tidy(fit)
#> # A tibble: 4 x 8
#>       FL     RW     CL     CW     BD  size withinss cluster
#>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <int>    <dbl> <fct>
#> 1  0.333  0.351  0.350  0.364  0.314    71     43.5 1
#> 2 -1.55  -1.58  -1.60  -1.61  -1.57     28     16.6 2
#> 3 -0.610 -0.552 -0.595 -0.592 -0.586    60     27.6 3
#> 4  1.38   1.28   1.36   1.34   1.39     41     38.1 4
```

```
#-- glance() summarizes the info about the entire model
broom::glance(fit)
#> # A tibble: 1 x 4
#>   totss tot.withinss betweenss  iter
#>   <dbl>        <dbl>     <dbl> <int>
#> 1   995         126.      869.     3
```

```
library(GGally)
ggpairs(crabsX, aes(color = factor(fit$cluster)))
```

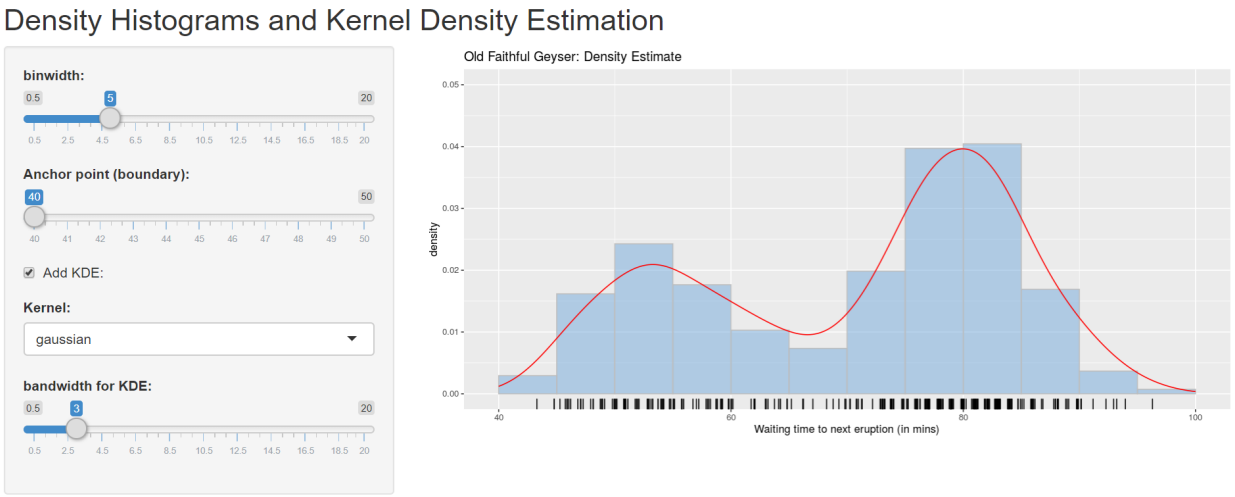

FL
RW
CL
CW
BD
Corr: 0.907***
1: 0.315**
2: 0.837***
3: 0.384**
4: 0.305.
Corr: 0.979***
1: 0.716***
2: 0.941***
3: 0.894***
4: 0.807***
Corr: 0.965***
1: 0.591***
2: 0.878***
3: 0.774***
4: 0.688***
Corr: 0.988***
1: 0.864***
2: 0.943***
3: 0.917***
4: 0.899***
Corr: 0.893***
1: 0.122
2: 0.898***
3: 0.405**
4: 0.056
Corr: 0.900***
1: 0.214.
2: 0.904***
3: 0.508***
4: 0.095
Corr: 0.889***
1: 0.136
2: 0.762***
3: 0.279*
4: 0.223
Corr: 0.995***
1: 0.954***
2: 0.980***
3: 0.948***
4: 0.951***
Corr: 0.983***
1: 0.828***
2: 0.897***
3: 0.822***
4: 0.901***
Corr: 0.968***
1: 0.703***
2: 0.828***
3: 0.645***
4: 0.782***
0.3
0.2
0.1
0.0
20
15
10
40
30
20
50
40
30
20
20
15
10
10 15 20
10 15 20
20 30 40
20 30 40 50
10 15 20

4 Mixture Models

4.1 Example: Old Faithful

The old faithful geyser in Yellowstone National Park is one of the most regular geysers in the park. The waiting time between eruptions is between 35 and 120 mins.

- We used kernel density estimation (KDE) to estimate the density of waiting times in a previous section.
  - See the Shiny App



- We used KDE because we couldn't think of any parametric distribution that could capture the bi-modal structure of the waiting time distribution.
- While KDE can be a great approach, another idea is to use a mixture of parametric distributions.
  - These can also capture complex densities (like multiple modes), but also provide a nice intuitive interpretation (i.e., data comes from several sub-populations) which form the basis for clustering

4.2 Finite Mixture Models

Mixture models combine several parametric models to produce a more complex, yet easy to interpret distribution.

- natural representation when data come from K different clusters/groups

$$\begin{aligned} f(x) &= \sum_{k=1}^{K} \pi_k \, f_k(x) \\ &= \pi_1 \, f_1(x) + \pi_2 \, f_2(x) + \ldots + \pi_K \, f_K(x) \end{aligned}$$

- $0 \leq \pi_k \leq 1$, $\sum_{k=1}^{K} \pi_k = 1$
- $f_k(x)$ is a parametric pdf/pmf

Note

- Think of π_k as the probability that an observation comes from group k
- Think of $f_k(x)$ as the density of group k
- Usually written $f_k(x) = f(x; \theta_k)$
  - i.e., all component densities are of the same family, but have different parameter values

4.3 Gaussian Mixture Model (GMM): Univariate

Consider a two-component ($K = 2$) mixture of univariate Gaussian distributions (GMM):

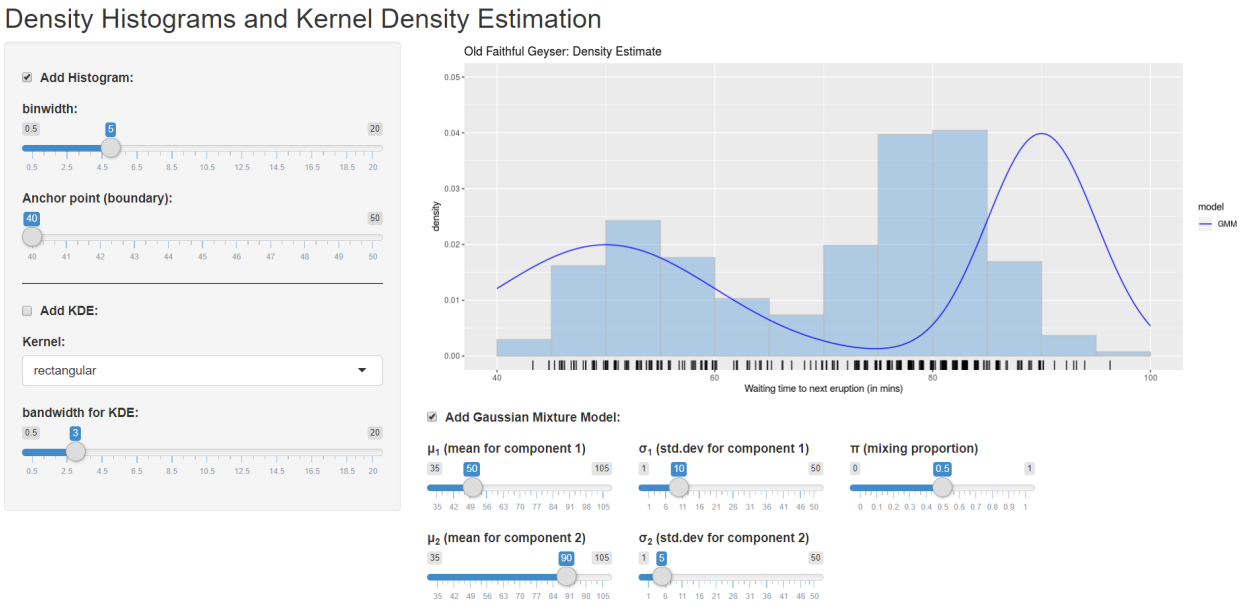
$$\begin{aligned} f(x; \theta) &= \pi f_1(x; \theta_1) + (1 - \pi) f_2(x; \theta_2) \\ &= \pi \mathcal{N}(x; \mu_1, \sigma_1) + (1 - \pi) \mathcal{N}(x; \mu_2, \sigma_2) \end{aligned}$$

- $\mathcal{N}(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$ is the pdf for a univariate Gaussian distribution
- $0 \leq \pi \leq 1$
- $\theta = (\pi, \mu_1, \sigma_1, \mu_2, \sigma_2)$
  - There are 5 parameters to estimate

4.3.1 Example: Old Faithful

Your Turn #3

Manually select parameters for the Gaussian Mixture Model. Use the Shiny app to help you decide.



4.4 Simulation (Generative Model)

- Data can be simulated from mixture models in a straightforward manner

Algorithm: Simulate Data from Gaussian Mixture Model (GMM)

To simulate n observations from a K component Gaussian mixture model (with **known parameters**):

$$\begin{aligned} f(x;\theta) &= \sum_{k=1}^{K} \pi_k f_k(x;\theta_k) \\ &= \sum_{k=1}^{K} \pi_k \mathcal{N}(x;\mu_k,\sigma_k) \end{aligned}$$

For i in $1, 2, \ldots, n$

1. Draw a label from a categorical distribution:

$$g_i \sim Cat(\pi_1, \ldots, \pi_K)$$

2. Draw the value from a Gaussian, conditional on the group label/parameters

$$x_i \mid g_i{=}k \sim N(\mu_k, \sigma_k)$$

To implement this more efficiently in R, I would do the following:

1. Draw all n labels from a multinomial distribution:

$$g \sim MN(n, \pi_1, \ldots, \pi_K)$$

2. For k in $1, \ldots, K$
   - Draw n_k observations from $N(\mu_k, \sigma_k)$
   - where $n_k = \sum_{i=1}^{n} \mathbb{1}(g_i = k)$

Your Turn #4

1. What is the expected value of n_k?
2. What is the expected value of X (i.e., the overall mean)?

4.5 EM Algorithm

The details are found in the assigned readings ESL 8.5.1 and Gaussian Mixture Models: 11.1-11.3, so we will review only the basics.

Notation:

- The observed data is $D = \{X_1, X_2, \ldots, X_n\}$
- Let $g_i \in \{1, 2, \ldots, K\}$ be the (unknown) group/component identifier.
  - π_k is prior probability that any observation is from component k
- The model parameters $\theta = (\pi_1, \ldots, \pi_K, \theta_1, \ldots, \theta_K)$
  - note: because we have the constraint that $\sum_k \pi_k = 1$, we only need to estimate $K - 1$ of the π's
- The **responsibility** r_{ik} is the posterior probability that observation i (X_i) came from component k:

$$\begin{aligned} r_{ik} &= \Pr(g_i = k | X_i, \theta) \\ &= \frac{P(X_i | g_i = k, \theta_k)\pi_k}{\sum_{j=1}^{K} P(X_i | g_i = j, \theta_j)\pi_j} \end{aligned}$$

- The responsibilities are weights: $\sum_{k=1}^{K} r_{ik} = 1 \; \forall i$, which represent the probability that events come from the components, conditional on the parameters θ.
- *EM* stands for *Expectation-Maximization*
  - *E-step*: calculate the responsibilities
  - *M-step*: estimate parameters using new responsibilities as weights
  - Iterate until convergence

Algorithm: EM Algorithm

Initialize

1. Set θ to something reasonable.

Repeat until convergence:

2. E-step: update r_{ik}, using θ, for $i = 1, 2, \ldots, n$ and $k = 1, \ldots, K$.
3. M-step: update θ using r_{ik} (maximizing the *expected* log-likelihood)

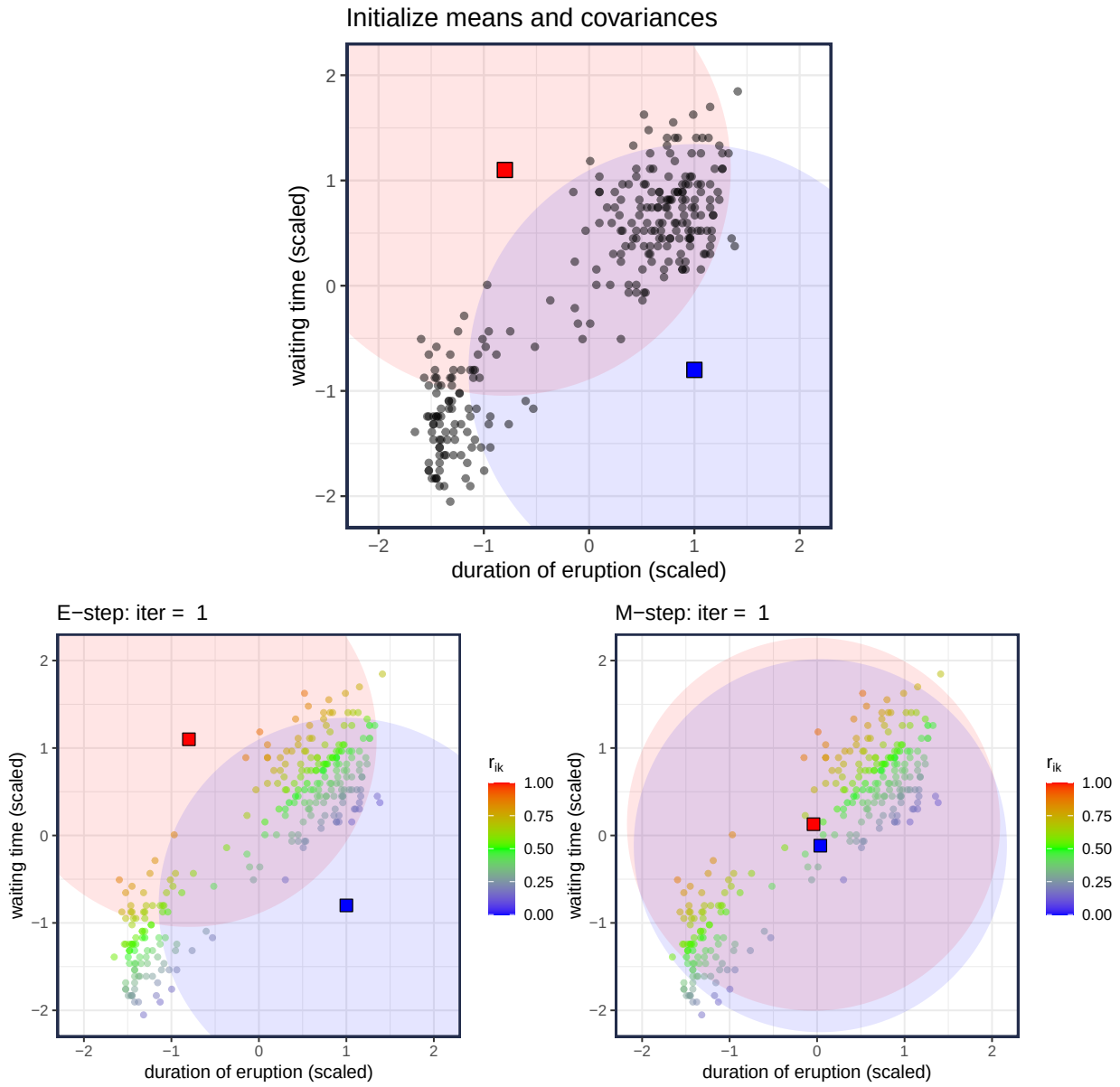
For Gaussian components:

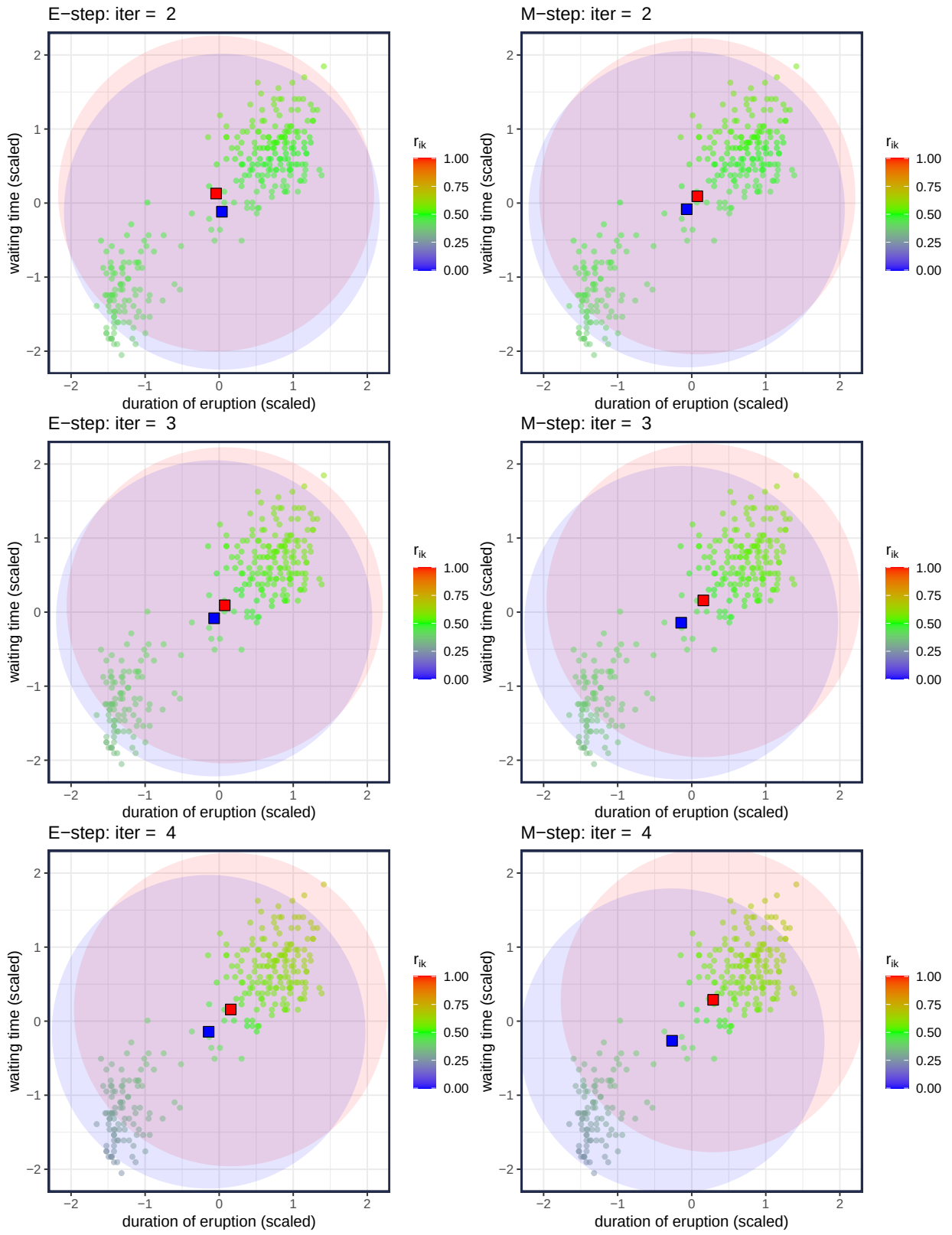
- Estimate like usual (MLE), except with *weighted* observations:

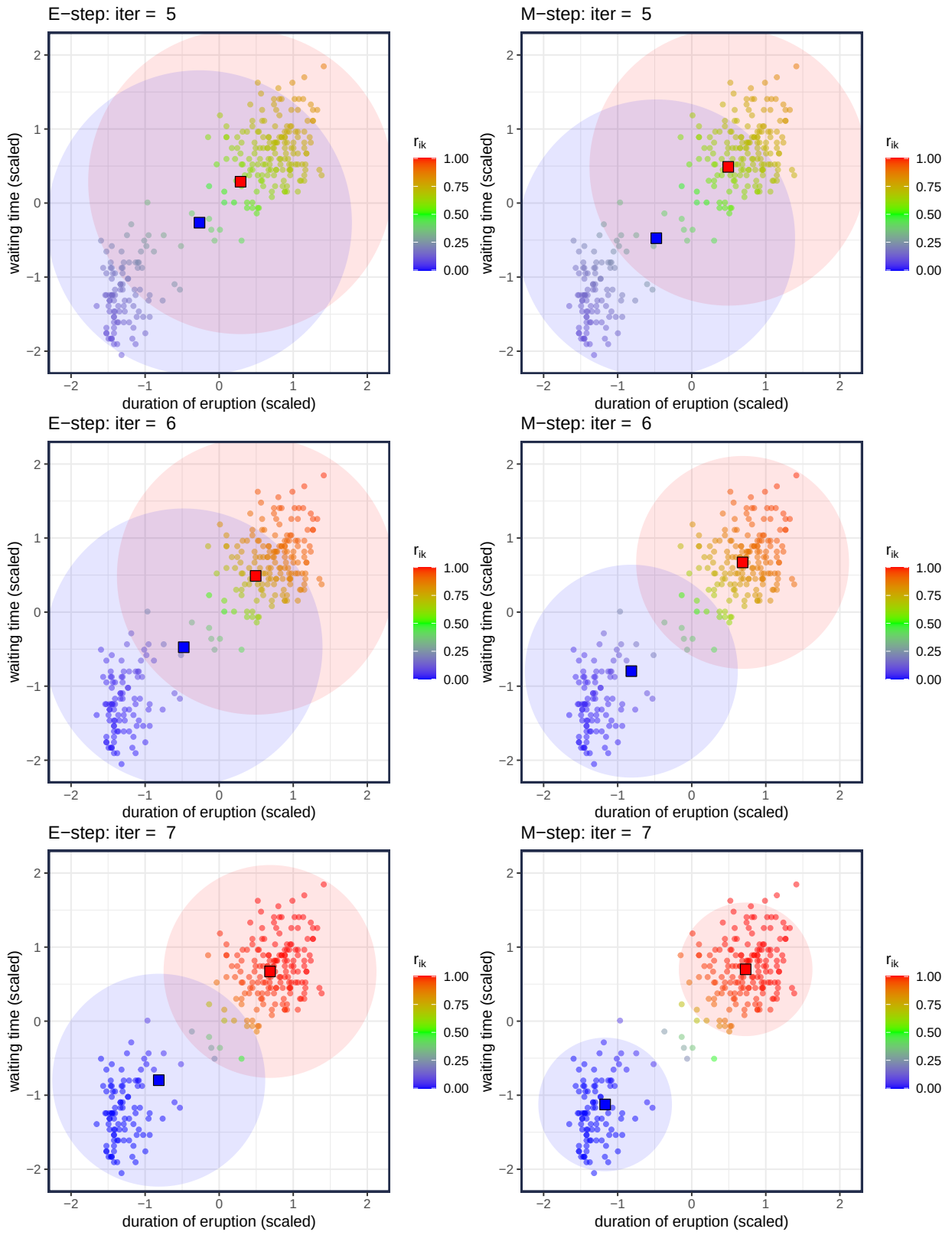
$$\begin{aligned} \hat{n}_k &= \sum_{i=1}^{n} r_{ik} \\ \hat{\pi}_k &= \hat{n}_k / n \\ \hat{\mu}_k &= \frac{1}{\hat{n}_k} \sum_{i=1}^{n} r_{ik} x_i \\ \hat{\sigma}_k^2 &= \frac{1}{\hat{n}_k} \sum_{i=1}^{n} r_{ik} (x_i - \hat{\mu}_k)^2 \end{aligned}$$

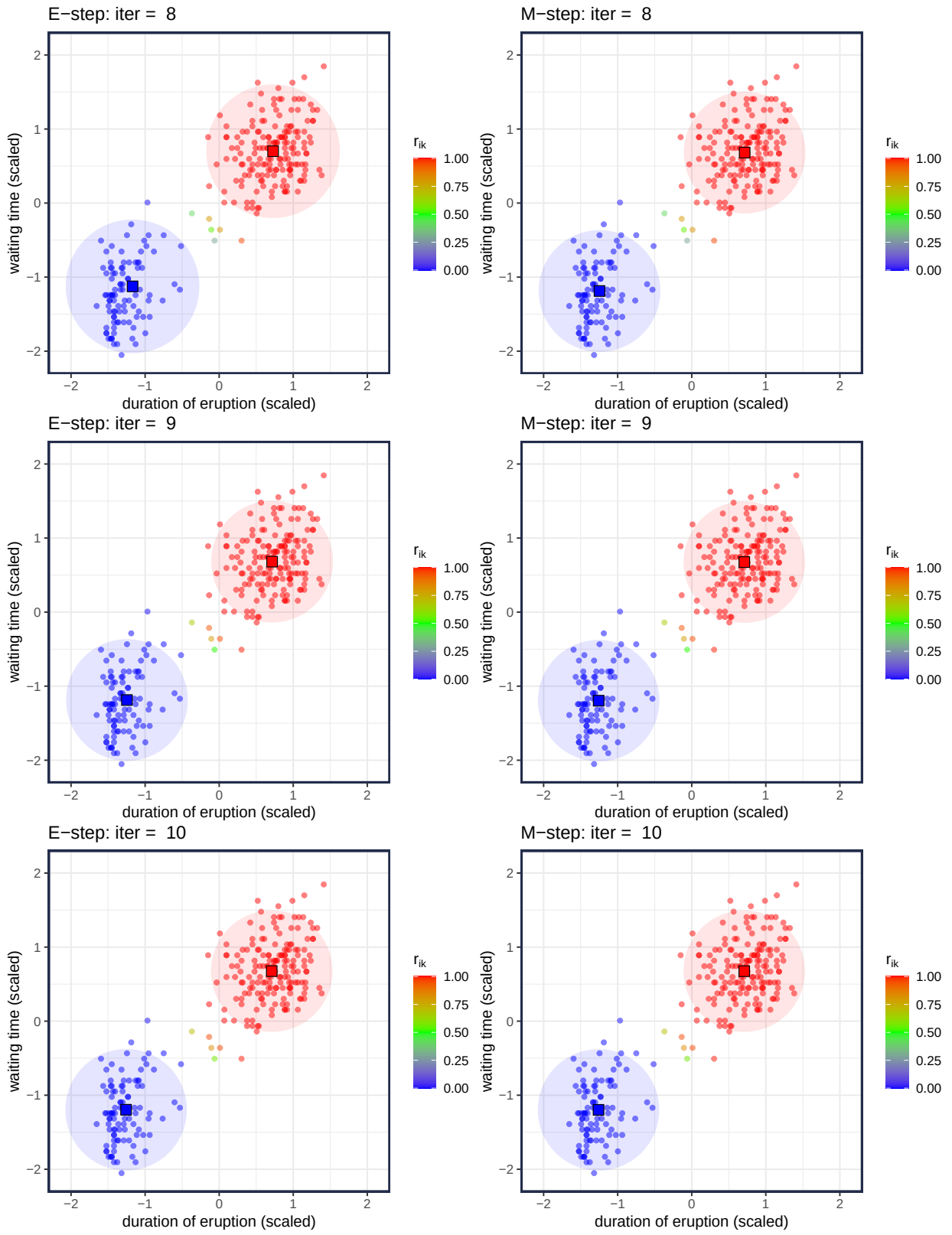
- The above is specific to univariate Gaussian mixture models.
- The assigned reading gives the mathematical details about what exactly EM is doing (e.g., what the *expected likelihood* is and why it is *maximized*)

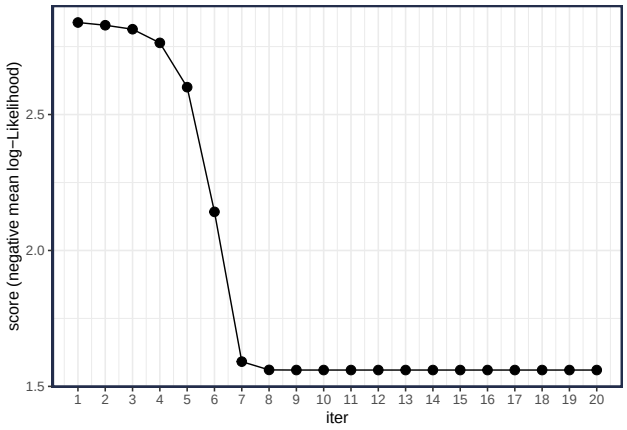
4.5.1 Example: Old Faithful




E−step: iter = 2
M−step: iter = 2
E−step: iter = 3
M−step: iter = 3
E−step: iter = 4
M−step: iter = 4
waiting time (scaled)
duration of eruption (scaled)
r_ik
1.00
0.75
0.50
0.25
0.00


E−step: iter = 5
M−step: iter = 5
E−step: iter = 6
M−step: iter = 6
E−step: iter = 7
M−step: iter = 7
waiting time (scaled)
duration of eruption (scaled)
r_ik
1.00
0.75
0.50
0.25
0.00


E−step: iter = 8
M−step: iter = 8
E−step: iter = 9
M−step: iter = 9
E−step: iter = 10
M−step: iter = 10
waiting time (scaled)
duration of eruption (scaled)
r_ik
1.00
0.75
0.50
0.25
0.00



- This example uses a common (spherical, volume) variance-covariance matrix for all components.
  - This is a strong restriction, but makes GMM very close to k-means
  - Allowing more flexibility in the variance-covariance matrix will give more complex models.

4.5.2 Issues in Fitting Mixture Models

- Like in K-means, the EM algorithm may converge to a *local* instead of global solution
  - Run several times with different initializations for θ
- Degenerate distributions
  - It's possible, especially in higher dimensions, to get component variances close to zero (e.g., when very few observations assigned to a cluster)
  - Just ignore these solutions, and run again with different initializations
  - Most software will do this automatically
- Add constraints (e.g., on the variance matrix) will help limit the complexity of the fitted models and can reduce the chances of finding degenerate component distributions

4.6 Multivariate Gaussian Mixture Models (GMM)

A multivariate normal random variable ($\mathbf{X} = X_1, \ldots, X_p$) has the joint pdf:

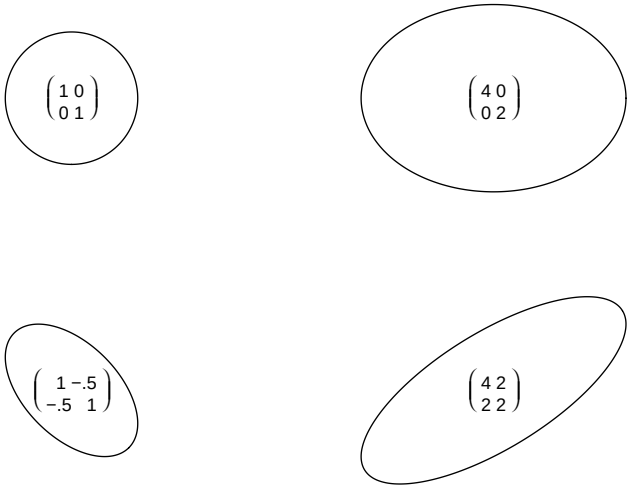
$$f(\mathbf{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^\mathsf{T}\Sigma^{-1}(\mathbf{x}-\mu)\right]$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \qquad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) & \ldots & Cov(X_1, X_p) \\ Cov(X_2, X_1) & Var(X_2) & \ldots & Cov(X_2, X_p) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(X_p, X_1) & Cov(X_p, X_2) & \ldots & Var(X_p) \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \ldots & \sigma_{1p} \\ \sigma_{21} & \sigma_2^2 & \ldots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \ldots & \sigma_p^2 \end{bmatrix}$$

- $|\Sigma|$ is the *determinant* of the variance-covariance matrix Σ
  - $\det(\Sigma) = \prod_{j=1}^{p} \lambda_i$ where λ are eigenvalues of Σ
- Σ controls the orientation, shape, and volume of the density contours

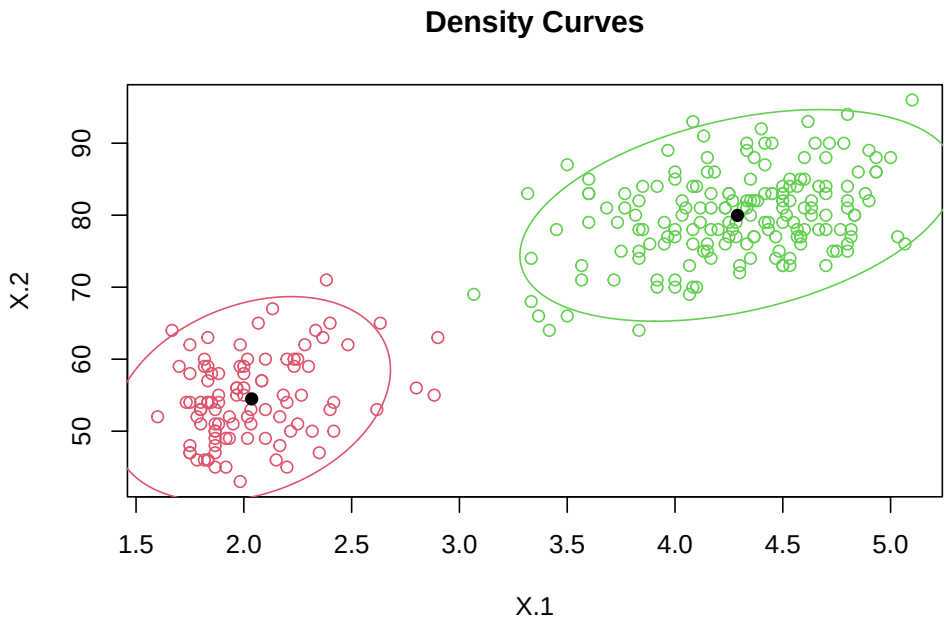


4.6.1 Example: Old Faithful

The R package `mixtools` provides a function `mvnormalmixEM()` to estimate parameters in a Gaussian mixture model using EM.

```r
library(mixtools)

#-- Use old faithful data
X = datasets::faithful
```

```r
#-- Fit K=2 component mixture model
GMM =  mvnormalmixEM(X, k=2)   # use mvnormalmixEM()
#> number of iterations= 12
(w = GMM$lambda)               # estimated weights
#> [1] 0.3559 0.6441
(mu = GMM$mu)                  # estimate means
#> [[1]]
#> [1]  2.036 54.479
#>
#> [[2]]
#> [1]  4.29 79.97
(Sigma = GMM$sigma)            # estimated covariance matrix
#> [[1]]
#>          [,1]    [,2]
#> [1,] 0.06917  0.4352
#> [2,] 0.43517 33.6973
#>
#> [[2]]
#>         [,1]    [,2]
#> [1,] 0.1700  0.9406
#> [2,] 0.9406 36.0462
sapply(Sigma, det)             # determinant
#> [1] 2.141 5.242

#-- Plot 95% contours
plot(GMM, whichplots = 2, alpha=.05) # 95% is 1-alpha
```



4.7 Mixture Models vs. LDA/QDA

In Linear/Quadratic Discriminant Analysis (LDA/QDA) each class was modeled by a Gaussian distribution. In LDA every class shared the same covariance matrix while in QDA each class had its own covariance matrix.

- Because the class labels were available, we could directly estimate the parameters.
- LDA is a simpler model (less parameters to estimate) that results in a linear boundary.

- QDA is more complex (higher variance) and thus more likely to over-fit
- We used hold-out/CV to find which model predicts best

For Gaussian mixture models (and model-based clustering) we aren't able to observe the labels, but choose to use the same type of Gaussian distributions to model the data.

- We can also restrict all components to share the same covariance matrix (simple model) or let the covariance matrices be unconstrained (more complex)
- The same bias/variance trade-off exists in this unsupervised setting

5 Model-Based Clustering

- Model-based clustering takes the *statistical view* of clustering:
  - observations are in a group if they are from the same distribution
- Model based clustering (with Gaussian components) can be thought of as an extension of K-means that relaxes the restrictions
  - Can fit data with non-spherical clusters, different sizes, different variances (densities)
  - While Gaussian components are most common, any other types of distributions are possible

Algorithm: Model Based Clustering

For k in $1, 2, \ldots, K$:

a. Fit a mixture model with k components
b. Record the log-likelihood (of the best solution if multiple initializations)
c. Choose the k and model parameters that optimize some criterion (e.g., BIC)

Using the best model $(\hat{K}, \hat{\theta})$, the *responsibilities*, $r_{ik}|\hat{\theta}$ allow hard or soft classification of each observation into a cluster.

- E.g., hard classification

$$\hat{g}_i = \arg\max_{1 \leq k \leq K} r_{ik}$$

- E.g., soft classification

$$\Pr(g_i = k|\hat{\theta}) = r_{ik}(\hat{\theta}_k)$$

5.1 Choosing K in Model-Based Clustering

- Back to the old problem of choosing the number of components K.
  - We have a few more options in this statistical setting

1. One approach is the *elbow* method with a plot of log-likelihood and K.
   - the log-likelihood will always increase as K increases
2. Use the *Bayesian Information Criterion (BIC)*.
   - m is the *number of parameters estimated by the model*
   - $\hat{L}$ is the maximum likelihood (i.e., the likelihood using the optimal parameter values)
   - Choose the model with the *lowest* BIC

$$BIC(m) = -2 \ln \hat{L} + m \ln(n)$$

- Note: In multivariate settings, m can vary depending on the model constraints (e.g., $\Sigma = \sigma I_p$)
- This penalizes the *complexity* of the model, where complexity is defined by the number of estimated parameters
  - Add constraints in the model will reduce the number of estimated parameters (and hence reduce the model complexity)

3. Use resampling (e.g., cross-validation, bootstrap) along with any loss function

5.2 Example with `mclust`

- The R package `mclust` provides many functions for running Gaussian model based clustering.

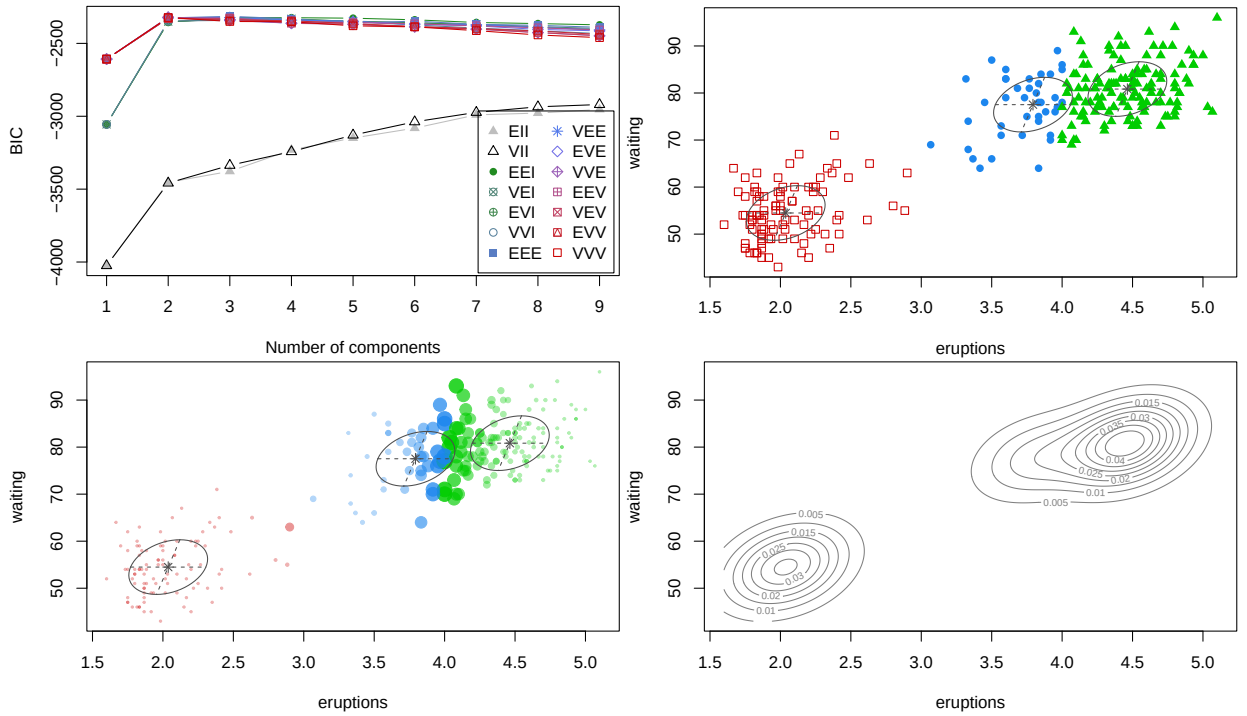
- See the `mclust` tutorial

NOTE: `mclust` uses an alternative definition of BIC (it uses the *negative* of what is stated above):

$$BIC_{\text{mclust}}(m) = 2\ln\hat{L} - m\ln(n)$$

- So for `mclust` the best model is the one that **maximizes** their BIC

```
# Use the mclust package to help search across all K's
library(mclust)
mix = Mclust(X, verbose=FALSE)    # fit series of models
summary(mix)                      # finds 3 clusters with EEE
#> ----------------------------------------------------
#> Gaussian finite mixture model fitted by EM algorithm
#> ----------------------------------------------------
#>
#> Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 3
#> components:
#>
#>  log-likelihood   n df   BIC   ICL
#>           -1126 272 11 -2314 -2358
#>
#> Clustering table:
#>   1   2   3
#>  40  97 135

plot(mix, what="BIC")            # Note: maximize BIC = 2logL - m log n
plot(mix, what="classification")
plot(mix, what="uncertainty")
plot(mix, what="density")
```




```
summary(mix, parameters=TRUE)
#> ----------------------------------------------------
#> Gaussian finite mixture model fitted by EM algorithm
#> ----------------------------------------------------
```

```
#>
#> Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 3
#> components:
#>
#>  log-likelihood   n df   BIC   ICL
#>           -1126 272 11 -2314 -2358
#>
#> Clustering table:
#>   1   2   3
#>  40  97 135
#>
#> Mixing probabilities:
#>      1      2      3
#> 0.1657 0.3564 0.4780
#>
#> Means:
#>              [,1]   [,2]   [,3]
#> eruptions  3.793  2.038  4.463
#> waiting   77.521 54.491 80.833
#>
#> Variances:
#> [,,1]
#>           eruptions waiting
#> eruptions   0.07825  0.4802
#> waiting     0.48020 33.7671
#> [,,2]
#>           eruptions waiting
#> eruptions   0.07825  0.4802
#> waiting     0.48020 33.7671
#> [,,3]
#>           eruptions waiting
#> eruptions   0.07825  0.4802
#> waiting     0.48020 33.7671
```

5.2.1 Constraints in `mclust`

The `mclust` package allows many types of constraints (14 to be precise) on the covariance matrices, $\{\Sigma_k\}$.

- This helps find models that have just the right complexity.
- By default, the `Mclust()` function can search over all possible covariance constraints
- Read the mclust 5 paper for more details

Model	$\boldsymbol{\Sigma}_k$	Distribution	Volume	Shape	Orientation
EII	$\lambda \boldsymbol{I}$	Spherical	Equal	Equal	—
VII	$\lambda_k \boldsymbol{I}$	Spherical	Variable	Equal	—
EEI	$\lambda \boldsymbol{A}$	Diagonal	Equal	Equal	Coordinate axes
VEI	$\lambda_k \boldsymbol{A}$	Diagonal	Variable	Equal	Coordinate axes
EVI	$\lambda \boldsymbol{A}_k$	Diagonal	Equal	Variable	Coordinate axes
VVI	$\lambda_k \boldsymbol{A}_k$	Diagonal	Variable	Variable	Coordinate axes
EEE	$\lambda \boldsymbol{D}\boldsymbol{A}\boldsymbol{D}^\top$	Ellipsoidal	Equal	Equal	Equal
EVE	$\lambda \boldsymbol{D}\boldsymbol{A}_k\boldsymbol{D}^\top$	Ellipsoidal	Equal	Variable	Equal
VEE	$\lambda_k \boldsymbol{D}\boldsymbol{A}\boldsymbol{D}^\top$	Ellipsoidal	Variable	Equal	Equal
VVE	$\lambda_k \boldsymbol{D}\boldsymbol{A}_k\boldsymbol{D}^\top$	Ellipsoidal	Variable	Variable	Equal
EEV	$\lambda \boldsymbol{D}_k\boldsymbol{A}\boldsymbol{D}_k^\top$	Ellipsoidal	Equal	Equal	Variable
VEV	$\lambda_k \boldsymbol{D}_k\boldsymbol{A}\boldsymbol{D}_k^\top$	Ellipsoidal	Variable	Equal	Variable
EVV	$\lambda \boldsymbol{D}_k\boldsymbol{A}_k\boldsymbol{D}_k^\top$	Ellipsoidal	Equal	Variable	Variable
VVV	$\lambda_k \boldsymbol{D}_k\boldsymbol{A}_k\boldsymbol{D}_k^\top$	Ellipsoidal	Variable	Variable	Variable

Table 3: Parameterisations of the within-group covariance matrix $\boldsymbol{\Sigma}_k$ for multidimensional data available in the **mclust** package, and the corresponding geometric characteristics.

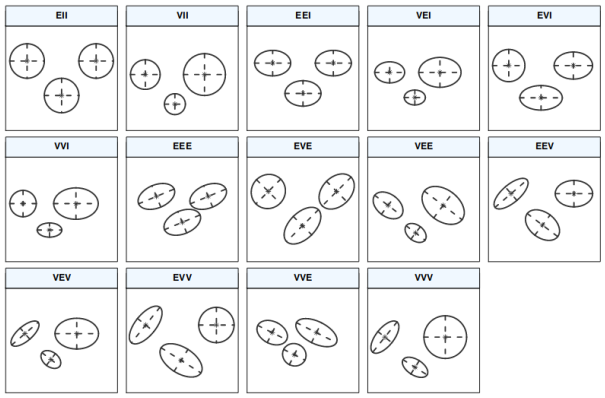


Figure 2: Ellipses of isodensity for each of the 14 Gaussian models obtained by eigen-decomposition in case of three groups in two dimensions.

5.2.2 Tidy Model-based clustering

The `broom` package provides some nice functions to quickly get relevant info. See the tidy Mclust help for details.

```
library(broom) # for augment(), tidy(), glance()

#-- augment() adds MAP class label and associated probability
broom::augment(mix, X)
#> # A tibble: 272 x 4
#>   eruptions waiting .class .uncertainty
#>       <dbl>   <dbl> <fct>         <dbl>
#> 1      3.6       79 1          2.82e- 2
#> 2      1.8       54 2          8.60e-13
#> 3      3.33      74 1          3.26e- 3
#> 4      2.28      62 2          3.14e- 7
#> 5      4.53      85 3          1.17e- 2
#> 6      2.88      55 2          3.09e- 3
#> # i 266 more rows

#-- tidy() summarizes each component (but no var-cov matrix yet)
broom::tidy(mix)
#> # A tibble: 3 x 5
#>   component  size proportion mean.eruptions mean.waiting
#>       <int> <int>      <dbl>          <dbl>        <dbl>
#> 1         1    40      0.166           3.79         77.5
#> 2         2    97      0.356           2.04         54.5
#> 3         3   135      0.478           4.46         80.8

#-- glance() summarizes the best model
broom::glance(mix)
#> # A tibble: 1 x 7
#>   model     G    BIC logLik    df hypvol  nobs
#>   <chr> <int>  <dbl>  <dbl> <dbl>  <dbl> <int>
#> 1 EEE       3 -2314. -1126.    11     NA   272
```

5.3 Gaussian MBC vs. K-means

5.3.1 Univariate Data

Consider first the univariate setting: $x \in \mathbb{R}$

- Recall the updates for K-means
  1. Given the cluster centroids $(\mu_1, \ldots, \mu_K)$, assign cluster label for each event:

$$g_i = \arg\min_{1 \le k \le K} \|x - \mu_k\|^2$$

  2. Given the labels, $(g_1, \ldots, g_n)$, update the cluster centroids:

$$\mu_k = \arg\min_{m} \sum_{i:g_i=k} \|x_i - m\|^2$$

Your Turn #5

1. What are the unknown parameters in K-means?

2. Derive a closed form expression for $\hat{\mu}_k$ (given $\{g_i\}$)

- Consider a GMM (Gaussian Mixture Model) where all components have the same variance.
  1. E-step: update r_{ik}, using θ, for $i = 1, 2, \ldots, n$ and $k = 1, \ldots, K$.
  2. M-step: update θ using r_{ik}

$$n_k = \sum_{i=1}^{n} r_{ik}$$
$$\hat{\pi}_k = n_k/n$$
$$\hat{\mu}_k = \frac{1}{n_k}\sum_{i=1}^{n} r_{ik}x_i$$
$$\hat{\sigma}_k^2 = \frac{1}{n_k}\sum_{i=1}^{n} r_{ik}(x_i - \hat{\mu}_k)^2 \qquad \text{For Multivariate: } \hat{\Sigma}_k = \frac{1}{n_k}\sum_{i=1}^{n} r_{ik}(\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^\mathsf{T}$$

Your Turn #6

1. What are the unknown parameters in the univariate K component GMM?

2. Estimate the GMM parameters with the constraints:
   a. (Equal size) $\pi_k = 1/K$ for all k
   b. (Equal variance) $\sigma_k = \sigma$ for all k

5.3.2 Multivariate Normal

- K-means is roughly equivalent to the setting:
  - $\pi_k = 1/K$ for all k
  - $\Sigma_k = \sigma I_p$
  - use "hard" assignments

Note

6 Appendix: R Code

```r
#-- Install Required Packages
# library(MASS)       # only needed for the crabs data
library(mclust)       # for model-based clustering
library(broom)        # for tidy model output
library(tidyverse)    # load after MASS so dplyr::select() is not overwritten
theme_set(theme_bw()) # set default ggplot theme

#------------------------------------------------------------------------------#
#-- Load crabs data
#------------------------------------------------------------------------------#
crabs = MASS::crabs                                  # get crabs data
crabsX = dplyr::select(crabs, FL, RW, CL, CW, BD) # extract features
crabsY = paste(crabs$sp, crabs$sex, sep=":")      # get true labels
```

```r
#------------------------------------------------------------------------------#
#-- Hierarchical Clustering
#------------------------------------------------------------------------------#

#-- Calculate Distance (dissimilarity matrix)
dX = dist(crabsX, method="euclidean")  # calculate distance

#-- Run hierarchical clustering
hc = hclust(dX, method="average")      # average linkage

#-- Plot
plot(hc)     # basic plot

plot(as.dendrogram(hc), las=1, leaflab="none")
ord = hc$order                 # order of x-axis
labels = crabsY[ord]
colors = ifelse(str_detect(labels, "B"), "blue", "orange")
shapes = ifelse(str_detect(labels, "M"), 17, 15)
n = nrow(crabsX)
points(1:n, rep(0, n), col=colors, pch=shapes, cex=.8)
legend("topright", c("Blue:Male", "Blue:Female", "Orange:Male", "Orange:Female"),
       pch=c(17, 15, 17, 15), col=c("blue", "blue", "orange", "orange"), cex=.8)

#-- Guess number of clusters by height plot
# tidyverse
tibble(height = hc$height, K = row_number(-height)) %>%
  ggplot(aes(K, log(height))) +
  geom_line() +
  geom_point(aes(color = ifelse(K == 5, "red", "black"))) +
  scale_color_identity() +
  coord_cartesian(xlim=c(1, 50), ylim=c(0.5, NA))

# base R
# n = length(hc$height)     # get number of merges
# plot(n:1, hc$height, type='o', xlab="K", ylab="height", las=1,
#      xlim=c(1, 50))
# points(5, hc$height[n-4], col="red", pch=19) # K=5

#-- Extract cluster membership
yhat = cutree(hc, k=5)     # cut so K=5
# yhat = cutree(hc, h=8.2) # cut at height of h=8.2
```

```r
#-- Confusion Matrix
table(true=crabsY, est=yhat)



#-- Scaling
X2 = scale(crabsX)         # each column has mean=0, sd=1
apply(X2, 2, sd)
apply(X2, 2, mean)

dX2 = dist(X2)                        # new distance (after scaling)
hc2 = hclust(dX2, method="complete") # new hclust (with scaled distances)

# tidyverse
tibble(height = hc2$height, K = row_number(-height)) %>%
  ggplot(aes(K, log(height))) +
  geom_line() +
  geom_point(aes(color = ifelse(K == 4, "red", "black"))) +
  scale_color_identity() +
  coord_cartesian(xlim=c(1, 50), ylim=c(-1, NA))

# base R
# n = length(hc2$height)      # get number of merges
# plot(n:1, hc2$height, type='o', xlab="K", ylab="height", las=1,
#      xlim=c(1, 50))
# points(4, hc2$height[n-3], col="red", pch=19) # K=4

# confusion table
table(true=crabsY, est=cutree(hc2, k=4))



#-- PCA (transform X matrix)
X3 = prcomp(crabsX, scale=TRUE)$x[, 1:2]  # first 2 PC
plot(X3,
     col = ifelse(str_detect(crabsY, "B"), "blue", "orange"),
     pch = ifelse(str_detect(crabsY, "M"), 17, 15))

dX3 = dist(scale(X3))                 # new distance
hc3 = hclust(dX3, method="complete") # new hclust

# tidyverse
tibble(height = hc3$height, K = row_number(-height)) %>%
  ggplot(aes(K, log(height))) +
  geom_line() +
  geom_point(aes(color = ifelse(K == 9, "red", "black"))) +
  scale_color_identity() +
  coord_cartesian(xlim=c(1, 50), ylim=c(-1, NA))

# base R
# n = length(hc3$height)      # get number of merges
# plot(n:1, hc3$height, type='o', xlab="K", ylab="height", las=1,
#      xlim=c(1, 50))
# points(9, hc3$height[n-8], col="red", pch=19) # K=9

# confusion table
table(true=crabsY, est=cutree(hc3, k=9))
```

```
#-------------------------------------------------------------------------#
#-- K-means clustering: crabs
#-------------------------------------------------------------------------#

X = scale(crabsX)                           # scale crabs data

#-- Run kmeans for multiple K
Kmax = 20                                   # maximum K
SSE = numeric(Kmax)                         # initiate SSE vector
for(k in 1:Kmax){
  km = kmeans(X, centers=k, nstart=25)      # use 25 starts
  SSE[k] = km$tot.withinss                  # get SSE
}

results = tibble(K = 1:Kmax, SSE = SSE)

#-- Plot results
# plot(1:Kmax, SSE, type='b', las=1, xlab="K")
# qplot(1:Kmax, SSE, geom="point", xlab="K")

results %>% ggplot(aes(K, log(SSE))) +
  geom_point() + geom_line() +
  scale_x_continuous(breaks=seq(0, 100, by=2))

#-- Evaluate to truth (both indicate K=5 isn't bad)
km = kmeans(X, centers=5, nstart=25)  # choose K=5
table(true=crabsY, est=km$cluster)
```

```
#-------------------------------------------------------------------------#
#-- K-means clustering: Old Faithful
#-------------------------------------------------------------------------#
oldf = datasets::faithful

gg = ggplot(oldf, aes(eruptions, waiting)) + geom_point()
gg + labs(title = "Old Faithful")
gg + coord_fixed(xlim=c(-40, 40)) +
  labs(title = "Old Faithful: same aspect")


km = kmeans(oldf, centers=2, nstart=25)

#-- Unscaled Solution
broom::augment(km, oldf) %>%
  ggplot(aes(eruptions, waiting, color=.cluster)) + geom_point() +
  geom_point(data = tidy(km), aes(color=cluster), pch=15, size=4) +
  scale_color_manual(values=c("black", "red"), guide=FALSE) +
  labs(title="Unscaled")

broom::augment(km, oldf) %>%
  ggplot(aes(eruptions, waiting, color=.cluster)) + geom_point() +
  geom_point(data = tidy(km), aes(color=cluster), pch=15, size=4) +
  scale_color_manual(values=c("black", "red"), guide=FALSE) +
  coord_equal(xlim=c(-45, 45)) +
  labs(title="Unscaled: same aspect")

#-- Scaled Solution
X2 = scale(oldf)
km2 = kmeans(X2, centers=2, nstart=25)
```

```
broom::augment(km2, X2) %>%
  ggplot(aes(eruptions, waiting, color=.cluster)) + geom_point() +
  geom_point(data = tidy(km2), aes(color=cluster), pch=15, size=4) +
  scale_color_manual(values=c("black", "red"), guide=FALSE) +
  labs(title="Scaled")

broom::augment(km2, X2) %>%
  ggplot(aes(eruptions, waiting, color=.cluster)) + geom_point() +
  geom_point(data = tidy(km2), aes(color=cluster), pch=15, size=4) +
  scale_color_manual(values=c("black", "red"), guide=FALSE) +
  coord_fixed() +
  labs(title="Scaled: raw values")

#-- Tidy K-means with the broom package
#   [tidy kmeans tutorial](https://www.tidymodels.org/learn/statistics/k-means/) for details.

library(broom) # for tidy(), augment(), glance()

#-- Scale data and implement kmeans
scaled_data = scale(crabsX)   # scale data
fit = kmeans(scaled_data, centers = 4, nstart = 100)  # run kmeans

# augment() adds the cluster labels to data
augment(fit, scaled_data)

# tidy() summarizes the info about each cluster
tidy(fit)

# glance() summarizes the info about the entire model
glance(fit)
```

```
#-------------------------------------------------------------------------#
#-- GMM Example: Old Faithful (Waiting Time)
# See interactive shiny example at: https://pasda.shinyapps.io/Old_Faithful/
#-------------------------------------------------------------------------#
#-- Load the Old Faithful data
oldf = datasets::faithful

#-- Make a ggplot object
pp = ggplot(oldf, aes(x=waiting)) + xlab("waiting time (min)")

#-- Function to calculate Gaussian mixture pdf
dnmix <- function(theta1, theta2, w=.5, x.seq=seq(-4, 4, length=100)){
  f1 = dnorm(x.seq, mean=theta1[1], sd=theta1[2])
  f2 = dnorm(x.seq, mean=theta2[1], sd=theta2[2])
  fmix = f1*w + f2*(1-w)
  return(fmix)
}

#-- Set parameters
theta1 = c(mu=50, sigma=10)       # parameters for component 1
theta2 = c(mu=90, sigma=5)        # parameters for component 2
w = .5                            # mixture weight

#-- Make data for plotting
x.seq = seq(40, 100, length=200)     # make sequence of x values
f = dnmix(theta1, theta2, w, x.seq)  # calculate the density at those values
data.mix = tibble(x.seq, f)          # make into a data frame/tibble
```

```r
#-- Make plot
pp +
  geom_histogram(binwidth = 1, aes(y=stat(density)), alpha=.5) +
  geom_line(data=data.mix, aes(x=x.seq, y=f), color="blue", size=1.25)
```

```r
#-------------------------------------------------------------------------#
#-- Sampling from a two component univariate GMM
#-------------------------------------------------------------------------#

#-- Set parameters
theta1 = c(mu=50, sigma=10)        # parameters for component 1
theta2 = c(mu=90, sigma=5)         # parameters for component 2
w = c(.4, .6)                      # mixture weights (must sum to one)
n = 300                            # number of samples to draw
set.seed(2019)                     # set the random seed for replication

#-- (1) Draw the group labels
g = sample(c(1,2), size=n, replace=TRUE, prob=w)


#-- (2) Sample from the component densities
#   To avoid loops, generate n observations from each density and pick the
#   one according to group label.
X = ifelse(g == 1,
           rnorm(n, mean=theta1[1], sd=theta1[2]),
           rnorm(n, mean=theta2[1], sd=theta2[2]))

qplot(X, bins = 50, geom="histogram")
hist(X, breaks=50)
```

```r
#-------------------------------------------------------------------------#
#-- GMM clustering: Old Faithful
#-------------------------------------------------------------------------#
library(mixtools)

X = datasets::faithful

#-- Fit K=2 component mixture model
GMM = mixtools::mvnormalmixEM(X, k=2)  # use mvnormalmixEM()
(w = GMM$lambda)            # estimated weights
(mu = GMM$mu)               # estimate means
(Sigma = GMM$sigma)         # estimated covariance matrix
sapply(Sigma, det)          # determinant

#-- Plot 95% contour
plot(GMM, whichplots = 2, alpha=.05) # 95% is 1-alpha
```

```r
#-------------------------------------------------------------------------#
#-- MBC clustering: Old Faithful
#-------------------------------------------------------------------------#
# Use the mclust package to help search across all K's
library(mclust)
mix = Mclust(X)
summary(mix)   # finds 3 clusters

plot(mix, what="BIC")
plot(mix, what="classification")
plot(mix, what="uncertainty")
plot(mix, what="density")
```

```r
#-- get parameters
summary(mix, parameters=TRUE)

#-- More detailed analysis: see https://www.stat.washington.edu/sites/default/files/files/reports/20
faithfulBIC = mclustBIC(X)
faithfulSummary  = summary(faithfulBIC, data=X)
faithfulSummary

plot(faithfulBIC, G=1:7,
     ylim=c(-2500,-2300), legendArgs=list(x="bottomright",ncol=5))

#-- tidy eval (using broom package)
library(broom) # for augment(), tidy(), glance()

# augment() adds MAP class label and associated probability
augment(mix, X)

# tidy() summarizes each component (but no var-cov matrix yet)
tidy(mix)

# glance() summarizes the best model
glance(mix)
```