ST340 Lab 3: Markov chains & PageRank, K-means

2020–21

1: PageRank

Inspect and run the following code, which sets up each object we need in PageRank.

- `links[,1]` denotes "from" page
- `links[,2]` denotes "to" page

Note: there are no links from a page to itself. `A.sparse` is a sparse matrix with `A[i,j] = 1` whenever `links[k,]` $= (i, j)$ for some `k`.
This links dataset gives links between pages. Where the the first column denotes where the link starts and the second column tell us the page where the link goes.

```
library(Matrix) # Load the sparse Matrix library
load("web-google.rdata") # Load the Google web data:


n <- max(links)              # Equal to the number of pages
numlinks <- dim(links)[1]    # number of links




# Turn the data into an adjacency matrix:
A.sparse <- sparseMatrix(i=links[,1], j=links[,2], x=rep(1,numlinks), dims=c(n,n))         # This the m
```

A.sparse having entry 1 at element [i,j], means there exists a link from page i to page j.

```
outlinks <- rep(0,n)
for (i in 1:numlinks) {
  # Calculate outlinks where `outlinks[i]` is the number of outlinks for page `i`:
  outlinks[links[i,1]] <- outlinks[links[i,1]] + 1
}

d <- outlinks == 0          # Calculate `d`: the binary vector that is 1 when page `i` has no outlinks






vsH <- rep(0,numlinks)
for (k in 1:numlinks) {
  # Calculate the values to assign to `H`. These are nonzero whenever links[k,]` = (i,j)
  # for some k but are normalized so that sum(A[i,]) = 1 whenever sum(A[i,]) > 0
  # (i.e. apart from when i is a dangling node).
  vsH[k] <- 1/outlinks[links[k,1]]        # this gives 1/(the number outlinks)
}

# Construct H as a sparse matrix
```

```
H.sparse <- sparseMatrix(i=links[,1], j=links[,2], x=vsH, dims=c(n,n))

alpha <- .85                # damping factor
w <- rep(1/n,n)             # (these are not sensible choices of w and p, but they "work")
p <- log(1+(1:n))/sum(log(1+(1:n)))          # personalization probability
```

2: Power Iteration

(a) We will do power iteration for `m` iterations:

```
m <- 150
```

(b) We will compute the difference between probability vectors after each iteration:

```
diffs <- rep(0,m)
```

(c) Start with the uniform distribution:

```
muT <- t(rep(1/n,n))
for (i in 1:m) {
  muT.old <- muT # store the old value of muT

  # compute the new value of muT = muT.old%*%G

  ## write your code here
  muT = (alpha*((muT.old)%*%(H.sparse))) + (alpha*(muT.old%*%d)%*%(t(w))) + (1 - alpha)%*%t(p)

  # compute the difference
  diff <- sum(abs(muT-muT.old))
  diffs[i] <- diff
  print(paste("iteration ",i,": ||muT-muT.old||_1 = ",diff,sep="",sum(muT)))
}
```

(d) Rank the pages!

```
ranking <- sort(muT, decreasing=TRUE, index.return=TRUE)$ix
```

3: K-means

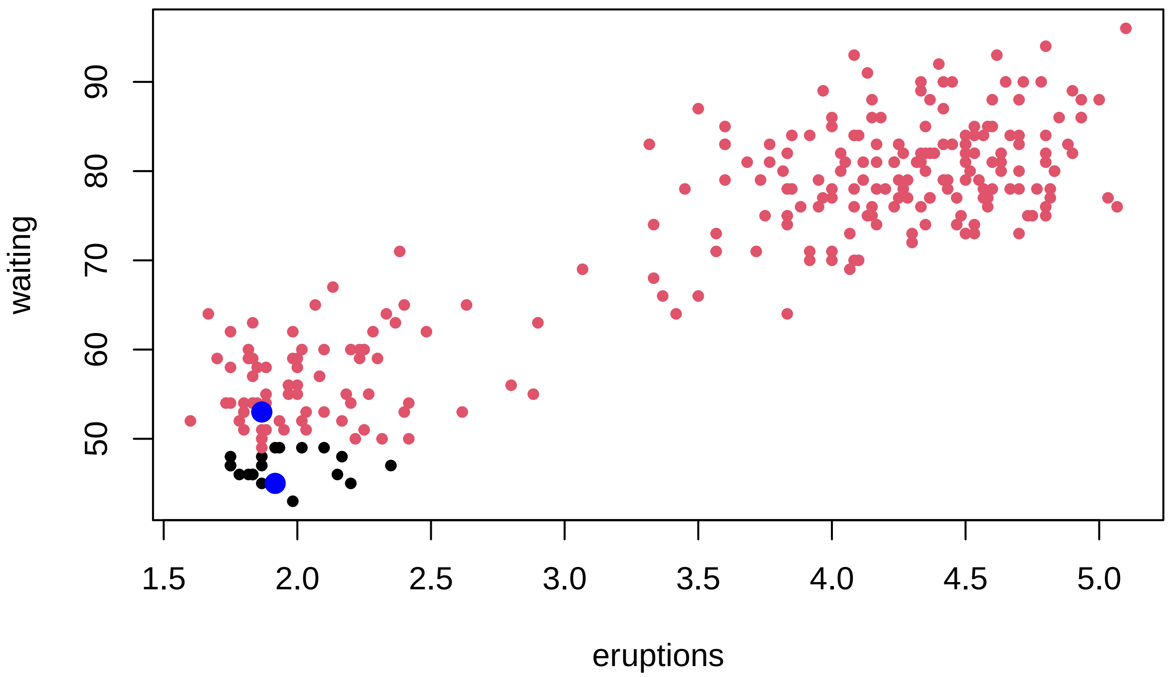
(a) Write a k-means algorithm:

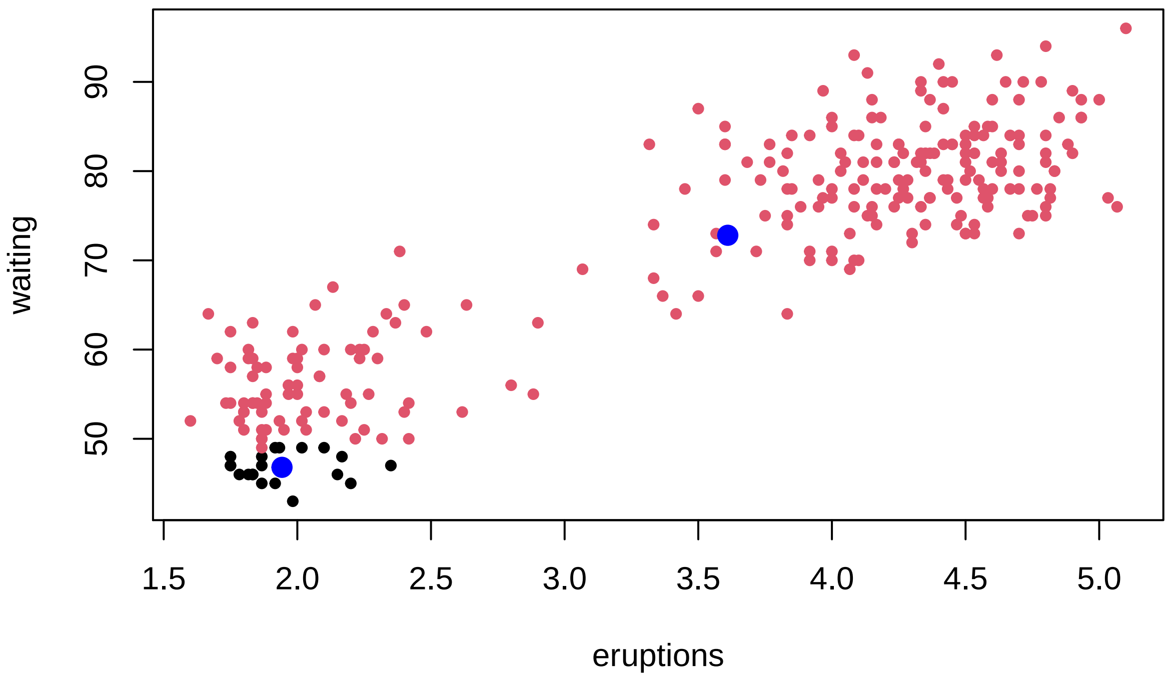
```
my.kmeans <- function(xs,K,ZZZZzzzz=0) {

  xs = as.matrix(xs)

  # find the number of columns and rows of xs
  p <- dim(xs)[2]
  n <- dim(xs)[1]

  # initialize the clusters
  minx <- min(xs)
  maxx <- max(xs)
  cs <- xs[sample(n,K),]     # Chosing K random centres

  # zs will be modified in the code
  zs <- rep(0,n)
```

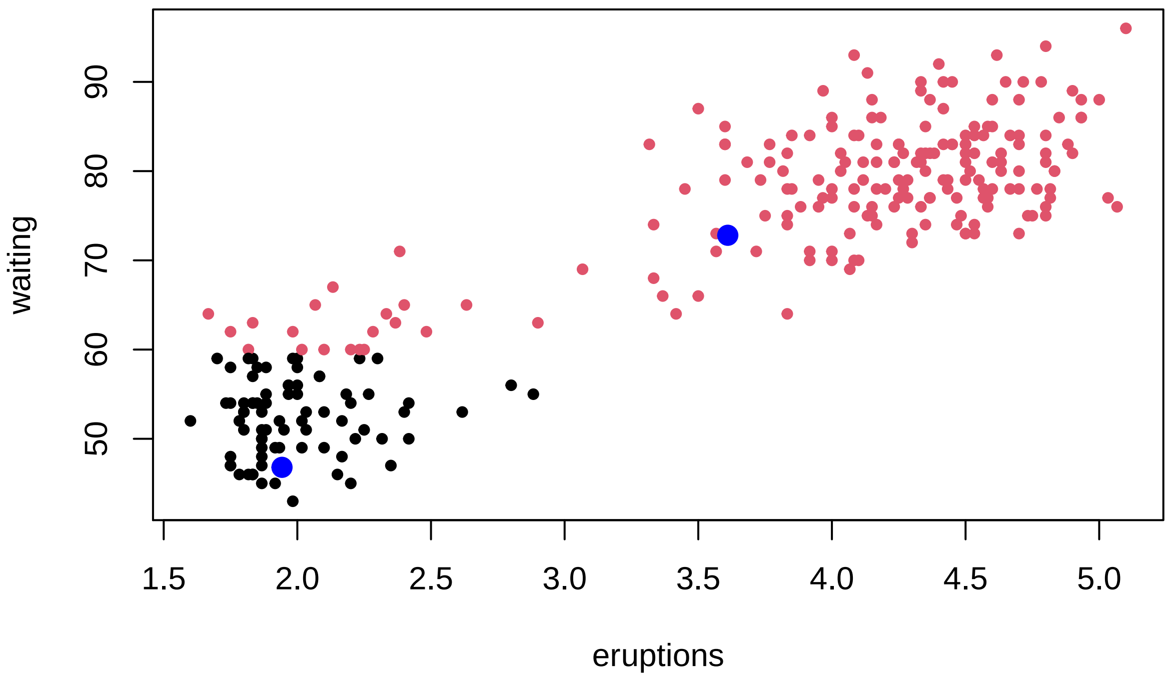
```r
  converged <- FALSE
  while(!converged) {
    cs.old <- cs
    zs.old <- zs

    # update zs
    dist_x_c = matrix(nrow=n, ncol=K)
    for(i in 1:n){
      for(j in 1:K){
        dist_x_c[i,j] = sum(((xs[i,]) - (cs[j,]))^2)
      }
    }

    dist_x_c_min = rep(0,n)
    for(i in 1:n){
      zs[i] = (1:K)[dist_x_c[i,] == min(dist_x_c[i,])]
    }


    plot(xs,col=zs,pch=20)
    points(cs,pch=20,col="blue",cex=2)

    Sys.sleep(ZZZZzzzz)

    # update cs

    for(i in 1:K){
      cs[i,] = cbind(mean(xs[zs == i,1]), mean(xs[zs == i,2]))
    }

    plot(xs,col=zs,pch=20)
    points(cs,pch=20,col="blue",cex=2)

    Sys.sleep(ZZZZzzzz)

    if (all(zs==zs.old)){
      converged <- TRUE
    }
  }
  return(list(cs=cs,zs=zs))
}
```

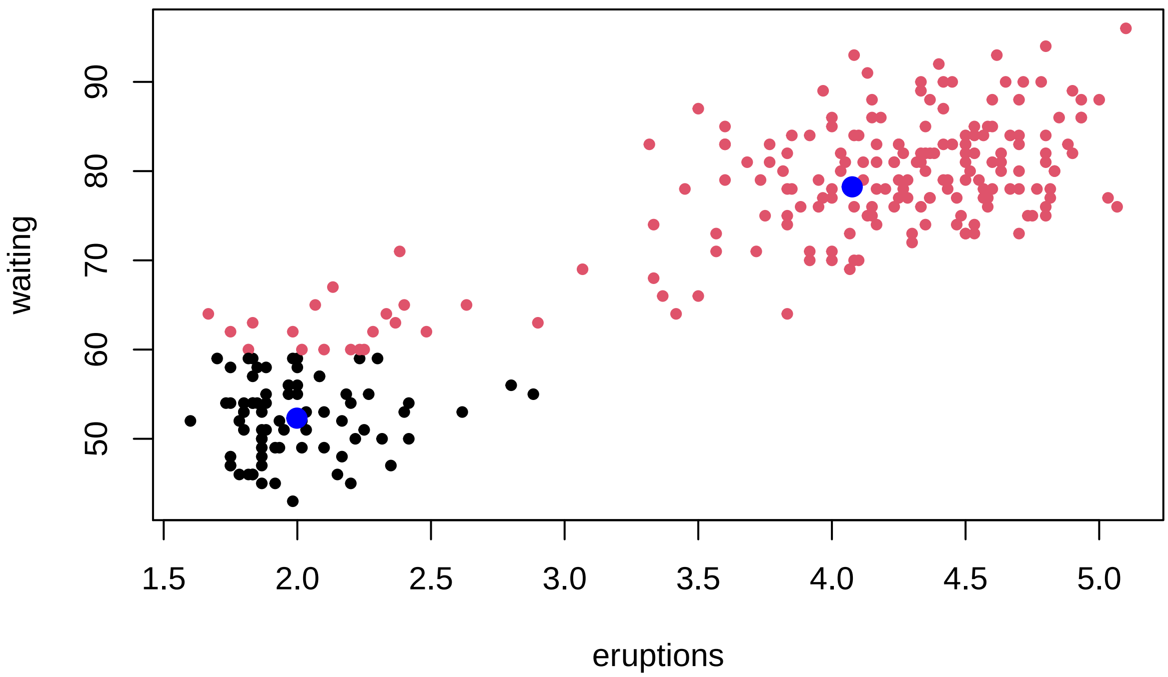
(b) Check that it works:

```r
xs <- faithful
my.kmeans.out <- my.kmeans(xs, K=2, ZZZZzzzz=1)
```

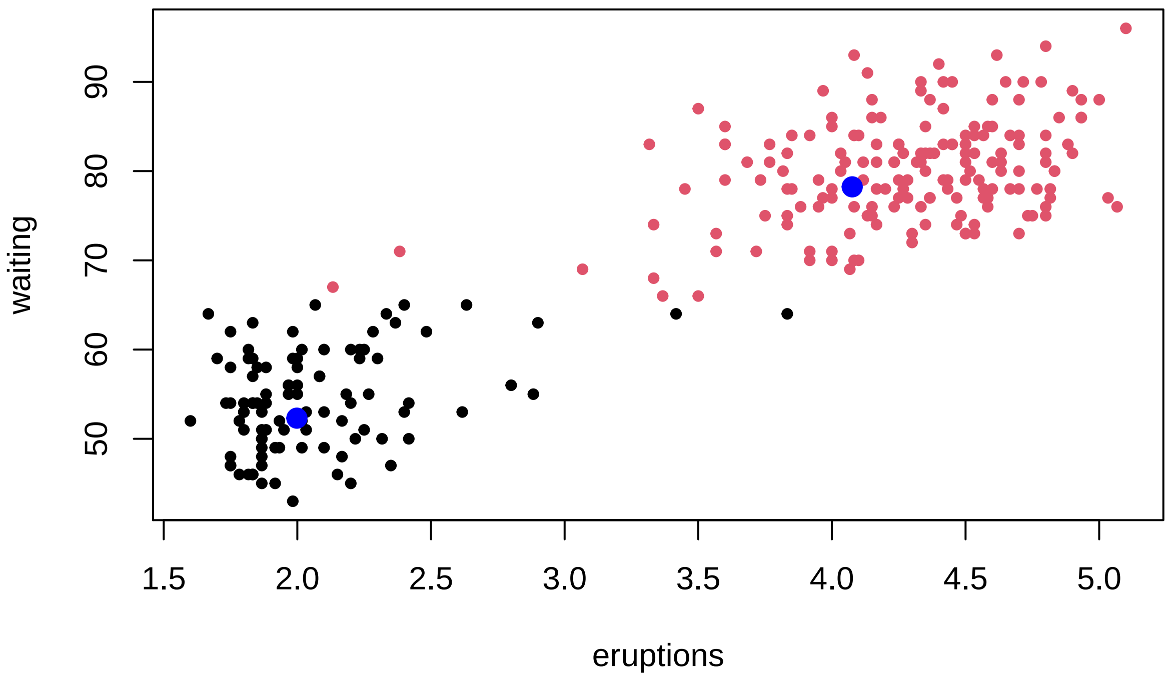

waiting
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
eruptions


waiting
eruptions
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

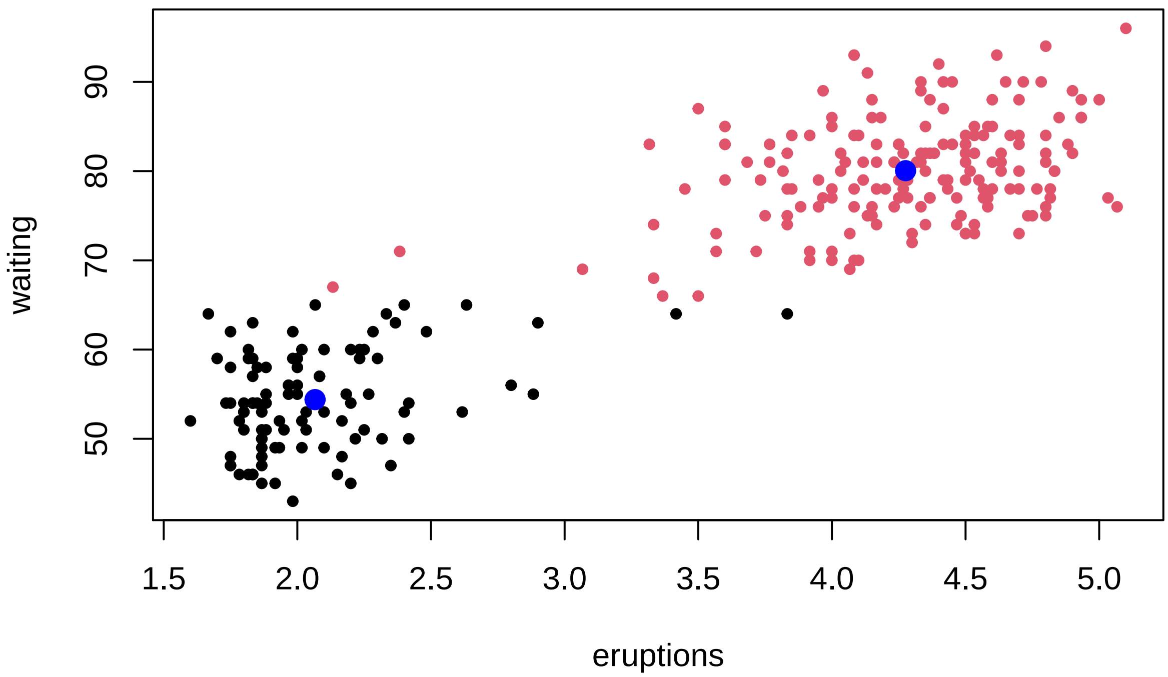

waiting
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
eruptions

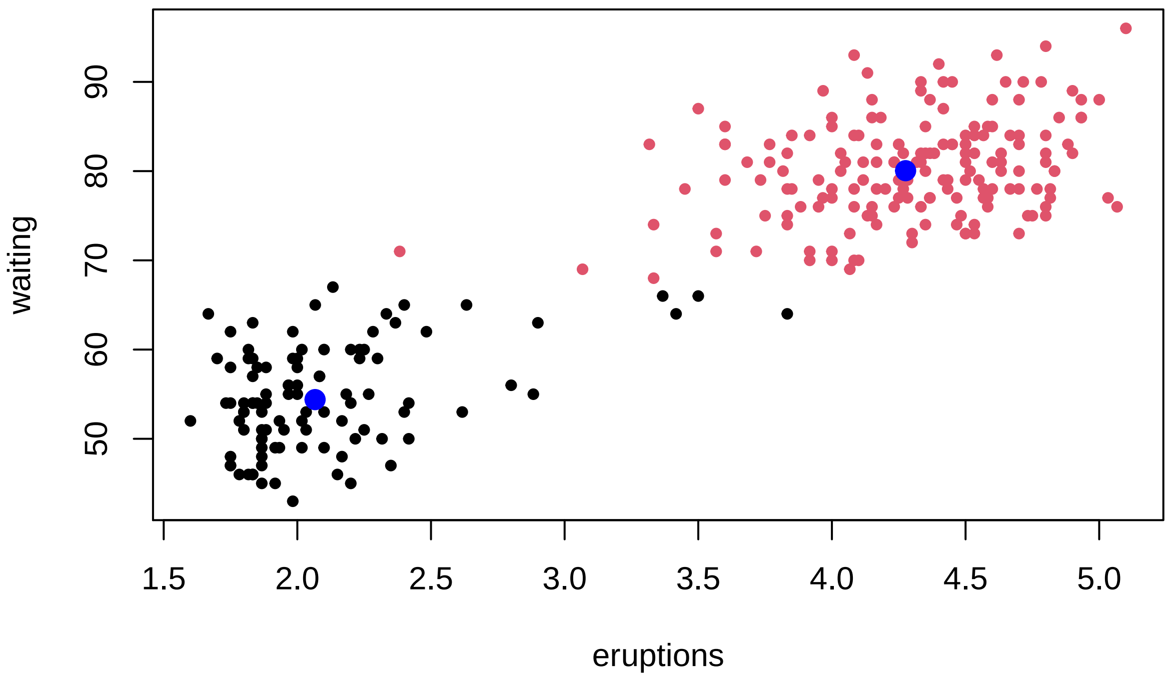


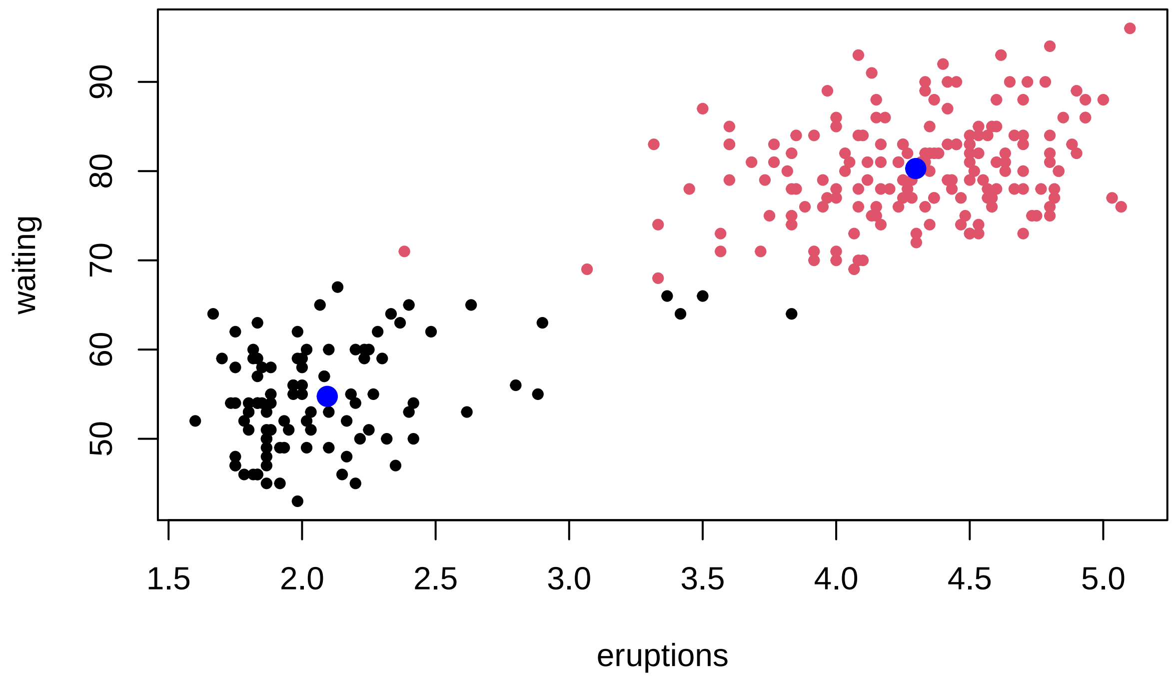
waiting
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
eruptions



waiting
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
eruptions


waiting
eruptions
50
60
70
80
90
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


waiting
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
eruptions

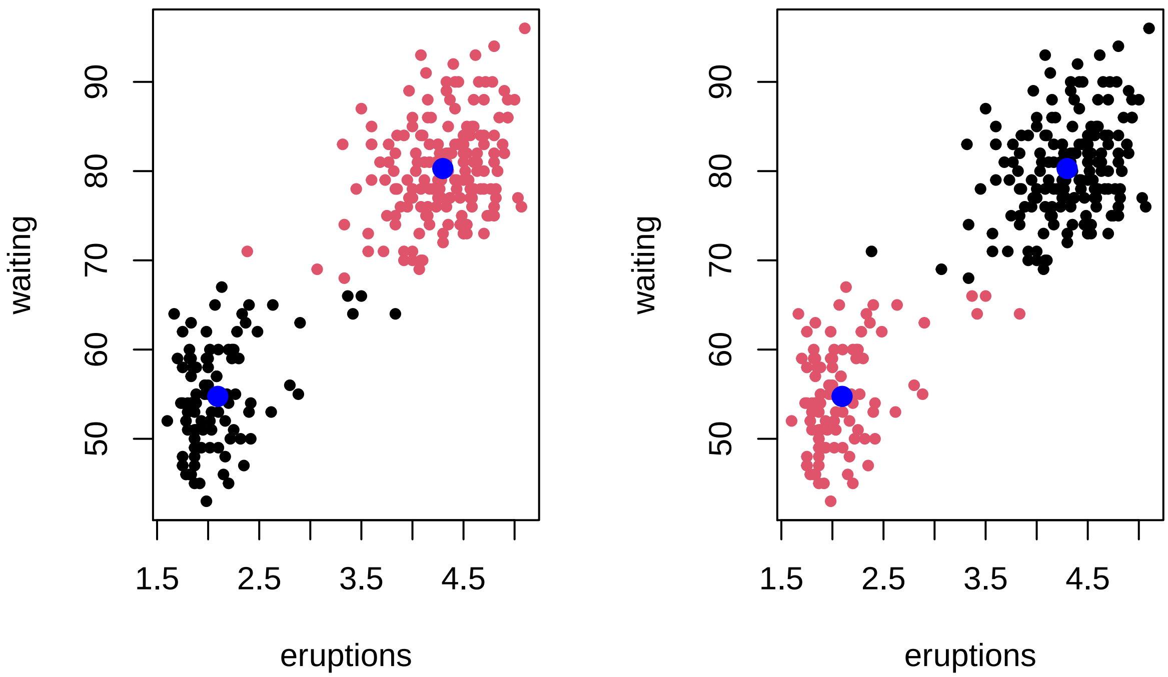



```
xs[1,]
```

```
##   eruptions waiting
## 1       3.6      79
```

(c) Check the output against the output of `kmeans(faithful,2)`:

```
par(mfrow=c(1,2))
plot(xs, col=my.kmeans.out$zs, pch=20)
points(my.kmeans.out$cs ,pch=20, col="blue", cex=2)

builtin.kmeans.out <- kmeans(xs,2)
plot(xs,col=builtin.kmeans.out$cluster,pch=20)
points(builtin.kmeans.out$centers,pch=20,col="blue",cex=2)
```


waiting
eruptions
50
60
70
80
90
1.5
2.5
3.5
4.5
waiting
eruptions
50
60
70
80
90
1.5
2.5
3.5
4.5

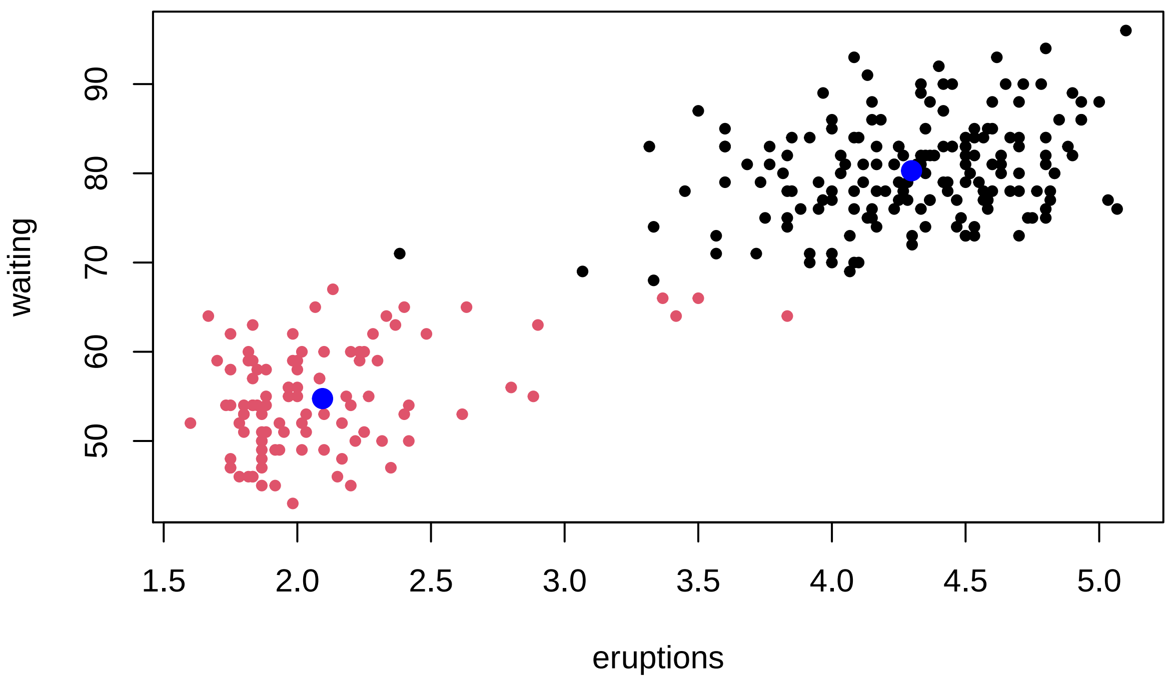


Figure 1: Built-in `kmeans` function for the Old Faithful datasset.