Clustering Protein consumption data

We will analyse the *Protein* dataset. The dataset contains information about protein consumption (divided into various subgroups) for 25 European countries in 1973. Our goal will be to cluster countries together based on their protein consumption.

Prepare the data

Begin with the data:

```
head(df)
```

```
## Source: local data frame [6 x 10]
##
##          Country RedMeat WhiteMeat  Eggs  Milk  Fish Cereals Starch  Nuts
##            (chr)   (dbl)     (dbl) (dbl) (dbl) (dbl)   (dbl)  (dbl) (dbl)
## 1        Albania    10.1       1.4   0.5   8.9   0.2    42.3    0.6   5.5
## 2        Austria     8.9      14.0   4.3  19.9   2.1    28.0    3.6   1.3
## 3        Belgium    13.5       9.3   4.1  17.5   4.5    26.6    5.7   2.1
## 4       Bulgaria     7.8       6.0   1.6   8.3   1.2    56.7    1.1   3.7
## 5 Czechoslovakia     9.7      11.4   2.8  12.5   2.0    34.3    5.0   1.1
## 6        Denmark    10.6      10.8   3.7  25.0   9.9    21.9    4.8   0.7
## Variables not shown: Fr&Veg (dbl)
```

Scale the data:

```
df_scaled = data.frame( scale(dplyr::select(df, -Country))   )
rownames(df_scaled) = unlist(dplyr::select(df, Country))
df_scaled[1:3, 1:3]
```

```
##            RedMeat  WhiteMeat      Eggs
## Albania  0.0812649 -1.7584889 -2.179639
## Austria -0.2772567  1.6523731  1.220454
## Belgium  1.0970762  0.3800675  1.041502
```

Use distance-based clustering

create a distance matrix from the data:

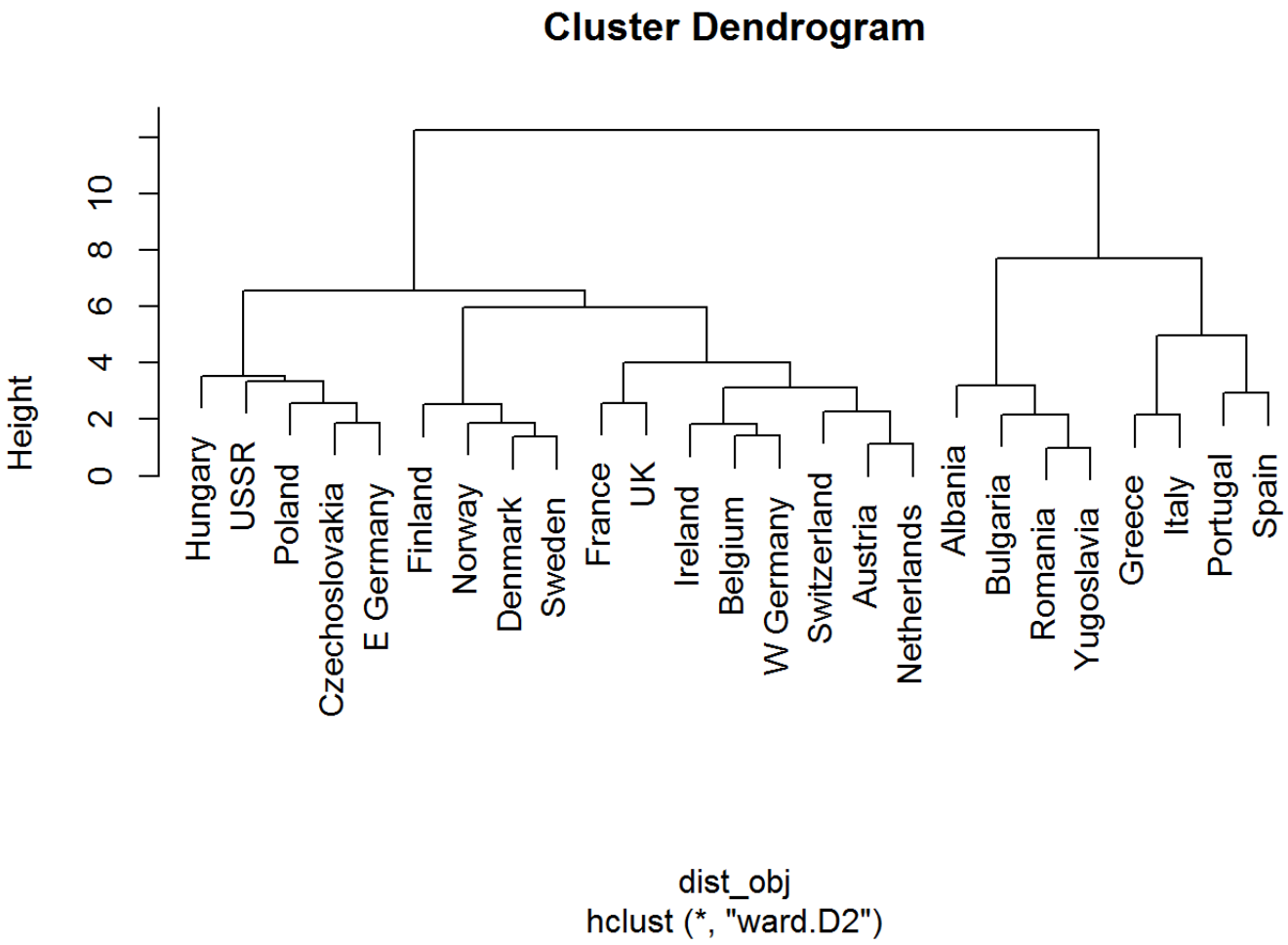
```
dist_obj = dist(df_scaled, method="euclidean")
```

Create a convenience function to print clusters:

```
print_clusters = function(labels, k)
{
  for(i in 1:k)
  {
    print(paste("cluster", i))
    print(df[labels == i, c("Country", "RedMeat", "Fish", "Fr&Veg")])
  }
}
```

Use *hclust()* to cluster the data:

```
cluster_obj = cutree(  hclust(dist_obj, method = "ward.D2" )  , k=4  )

plot(hclust(dist_obj, method = "ward.D2" ))
```



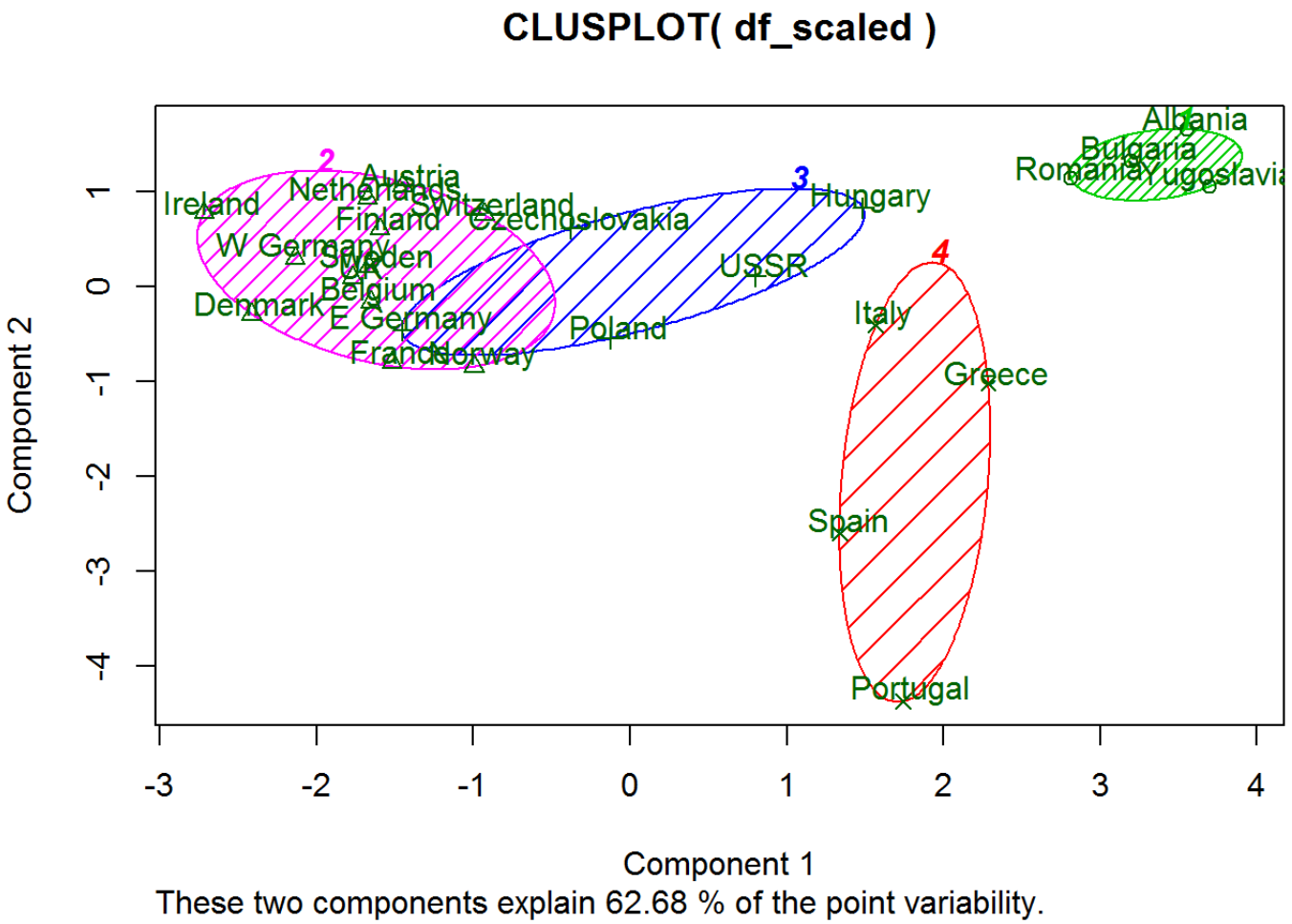
Print the clusters:

```
print_clusters(cluster_obj, 3)
```

```
## [1] "cluster 1"
## Source: local data frame [4 x 4]
##
##      Country RedMeat  Fish Fr&Veg
##        (chr)   (dbl) (dbl)  (dbl)
## 1    Albania    10.1   0.2    1.7
## 2   Bulgaria     7.8   1.2    4.2
## 3    Romania     6.2   1.0    2.8
## 4 Yugoslavia     4.4   0.6    3.2
## [1] "cluster 2"
## Source: local data frame [12 x 4]
##
##        Country RedMeat  Fish Fr&Veg
##          (chr)   (dbl) (dbl)  (dbl)
## 1      Austria     8.9   2.1    4.3
## 2      Belgium    13.5   4.5    4.0
## 3      Denmark    10.6   9.9    2.4
## 4      Finland     9.5   5.8    1.4
## 5       France    18.0   5.7    6.5
## 6      Ireland    13.9   2.2    2.9
## 7  Netherlands     9.5   2.5    3.7
## 8       Norway     9.4   9.7    2.7
## 9       Sweden     9.9   7.5    2.0
## 10 Switzerland    13.1   2.3    4.9
## 11          UK    17.4   4.3    3.3
## 12   W Germany    11.4   3.4    3.8
## [1] "cluster 3"
## Source: local data frame [5 x 4]
##
##          Country RedMeat  Fish Fr&Veg
##            (chr)   (dbl) (dbl)  (dbl)
## 1 Czechoslovakia     9.7   2.0    4.0
## 2      E Germany     8.4   5.4    3.6
## 3        Hungary     5.3   0.3    4.2
## 4         Poland     6.9   3.0    6.6
## 5           USSR     9.3   3.0    2.9
```

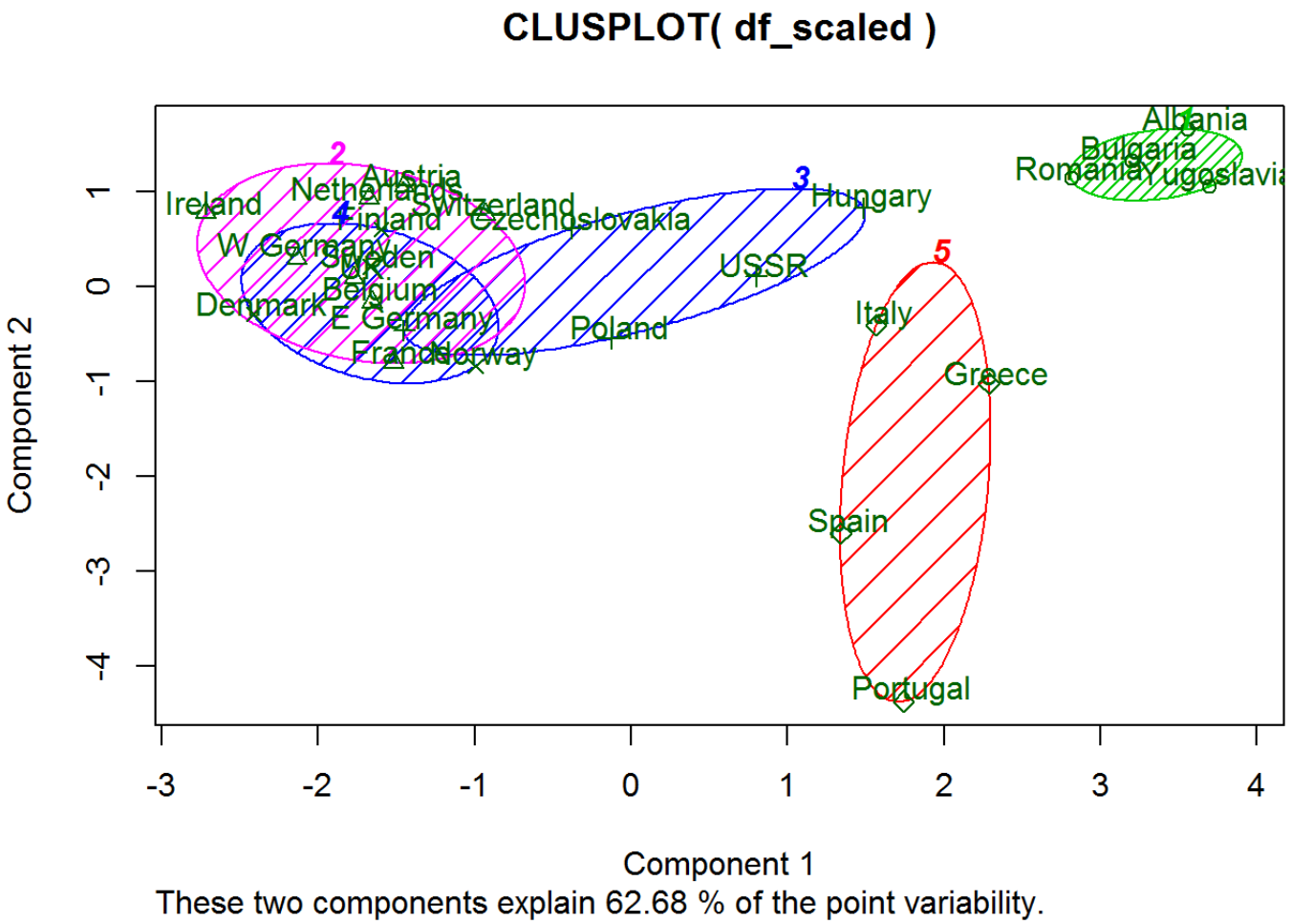
plot the clusters:

```
clusplot(df_scaled, cluster_obj, color=TRUE, shade=TRUE, labels=2, lines=0  )
```



Create a function which plots clusters, taking in settings as parameters:

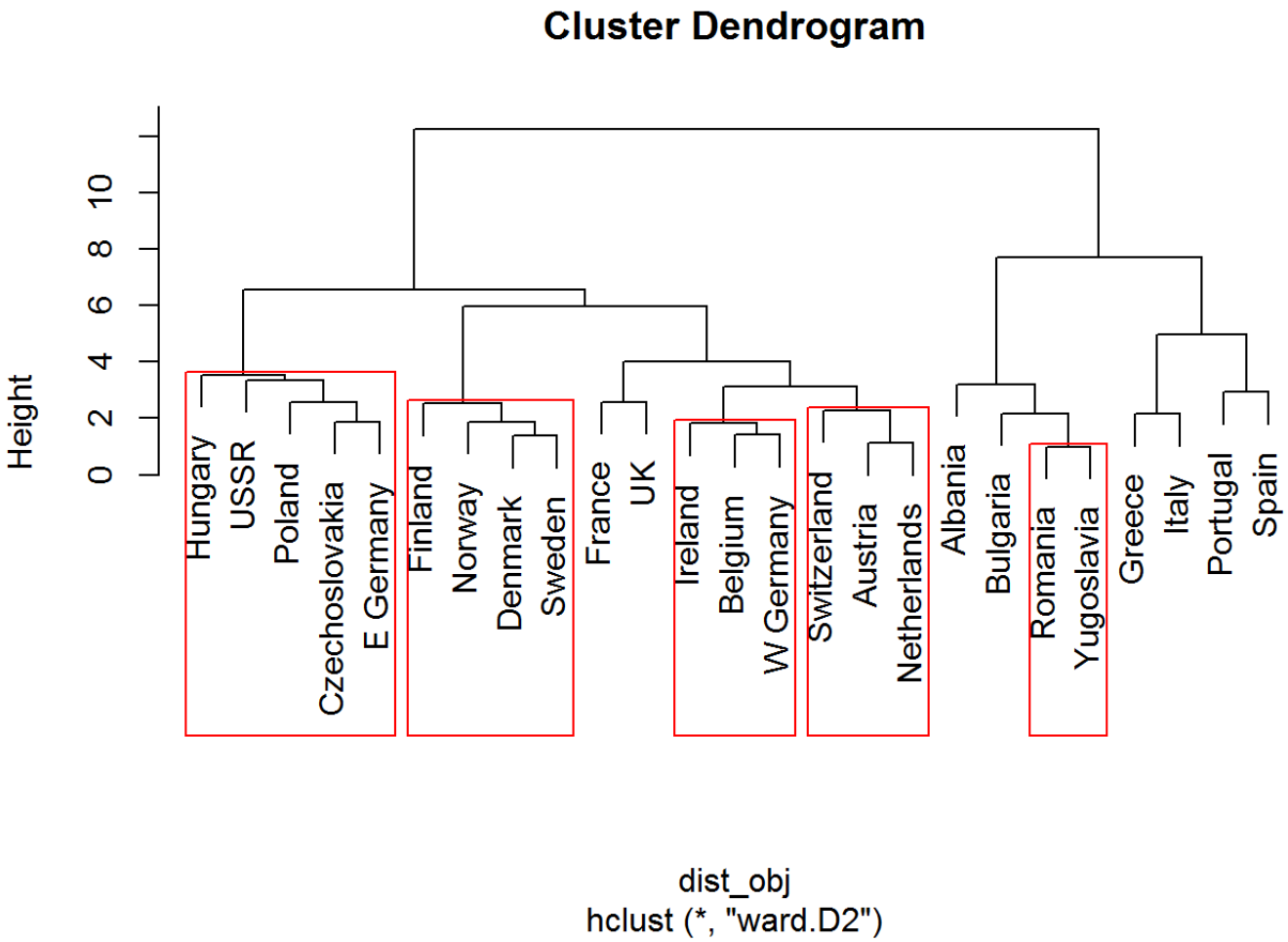
```
hclust_plot = function(d, method, k)
{
  cluster_obj = cutree(  hclust(d=d, method = method )  ,  k=k  )
  clusplot(df_scaled, cluster_obj, color=TRUE, shade=TRUE, labels=2, lines=0  )
}

hclust_plot(d = dist_obj, method = "ward.D2" , k = 5 )
```



These two components explain 62.68 % of the point variability.

Can we assign some level of statistical confidence to our clusters?

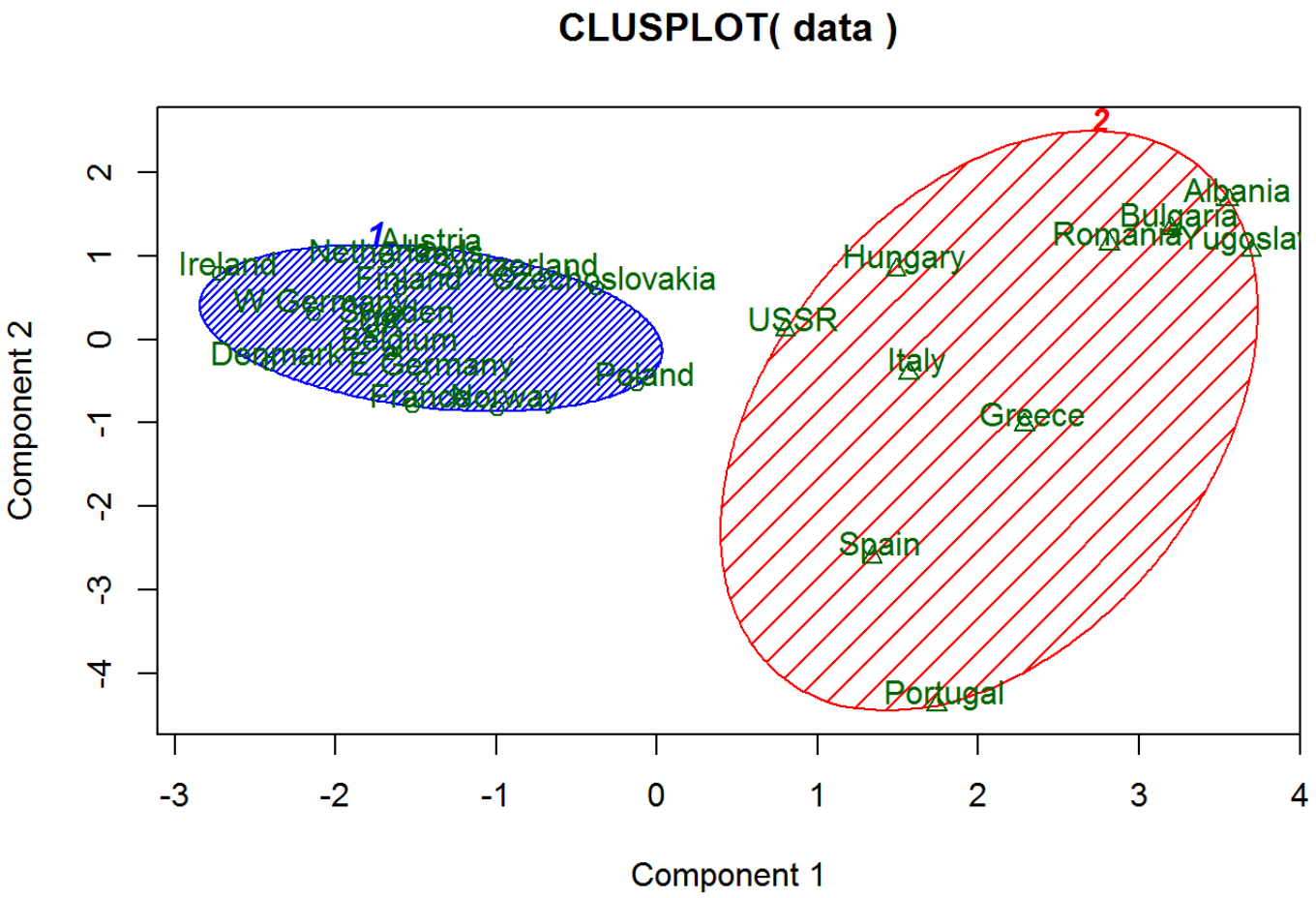
```
df_scaled_tr = t(data.matrix(df_scaled))

pv_obj = pvclust(df_scaled_tr,   method.hclust="ward.D2", method.dist="euclidean")

plot(hclust(dist_obj, method = "ward.D2" ))
pvrect(pv_obj, alpha = 0.95)
```



The highlighted clusters are significant at a 5% level.

K-means clustering

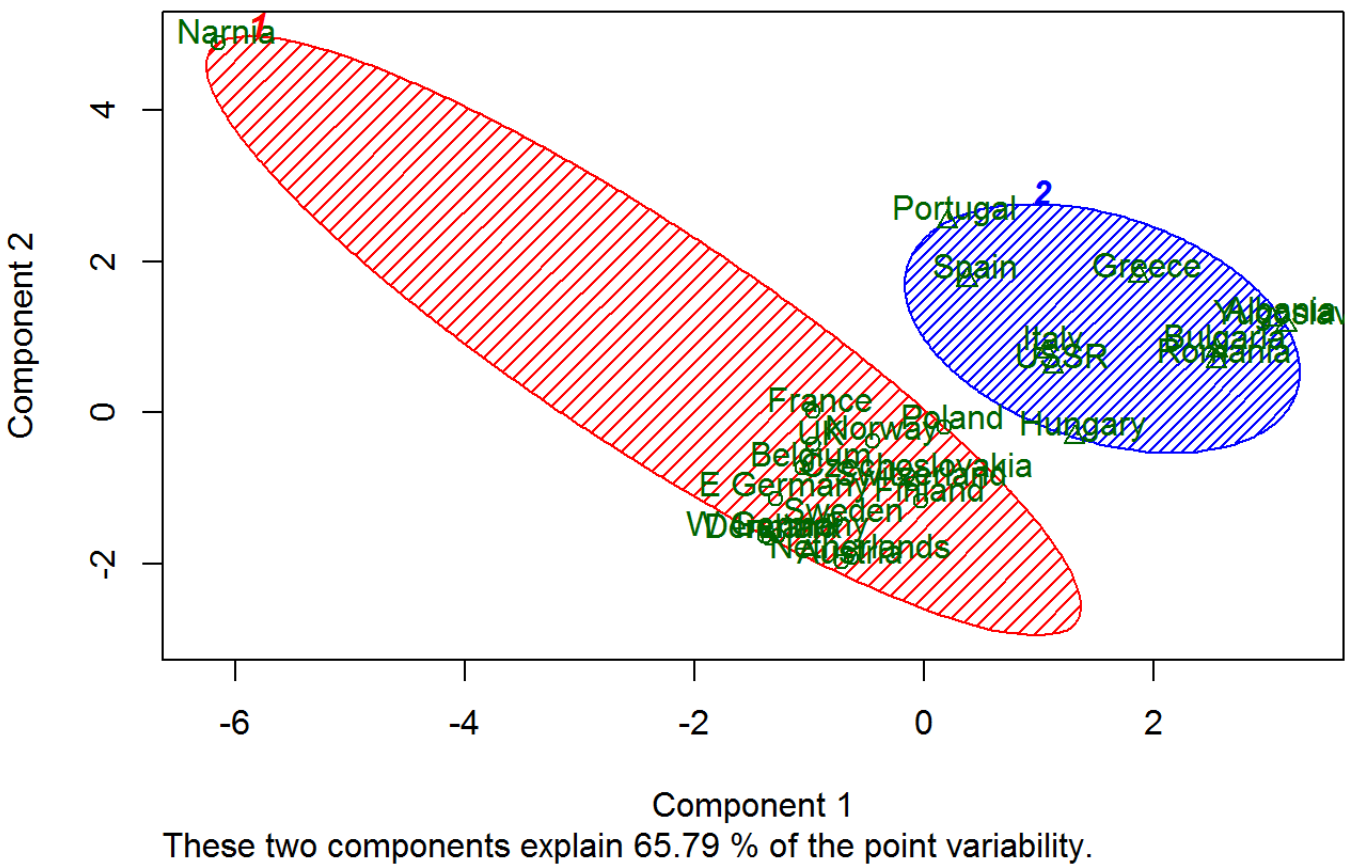
```
kmeans_plot = function(data, k)
{
  kmeans_obj = kmeans(data, k)
  kmeans_obj$cluster
  clusplot(data , kmeans_obj$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
}

kmeans_plot(df_scaled, 2)
```



The clusters from K-means clustering are not particularly stable with 5 clusters, but they are with two clusters.

We can add an artificial outlier:

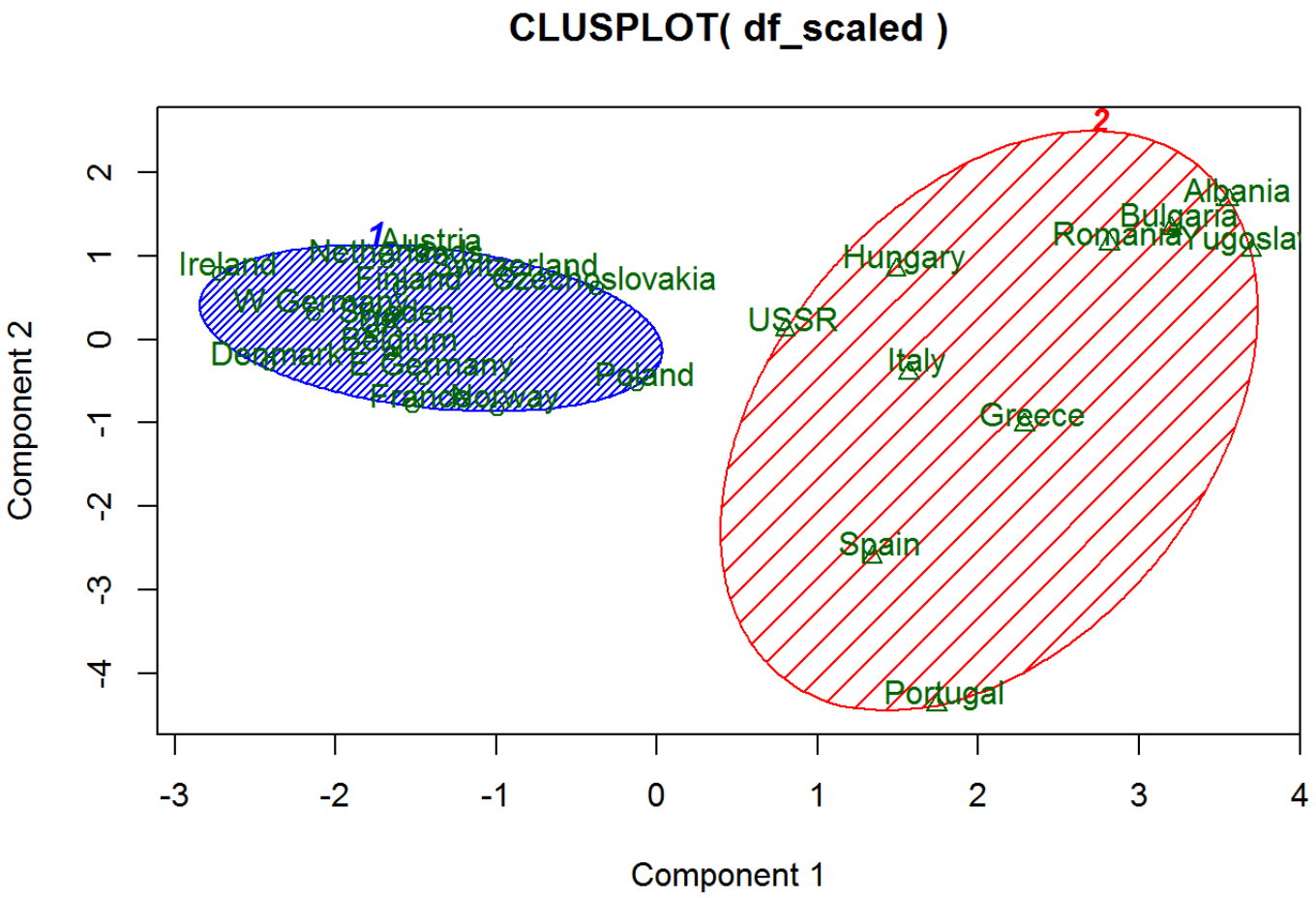
```
Narnia = c(10,-1,1,-10,1,-1,10,-1,1)
df_w_outlier = scale(rbind(df_scaled, Narnia))
rownames(df_w_outlier)[26] = "Narnia"
kmeans_plot(df_w_outlier, 2)
```



With 2 components, we get pretty close to the correct answer in the presence of an outlier.

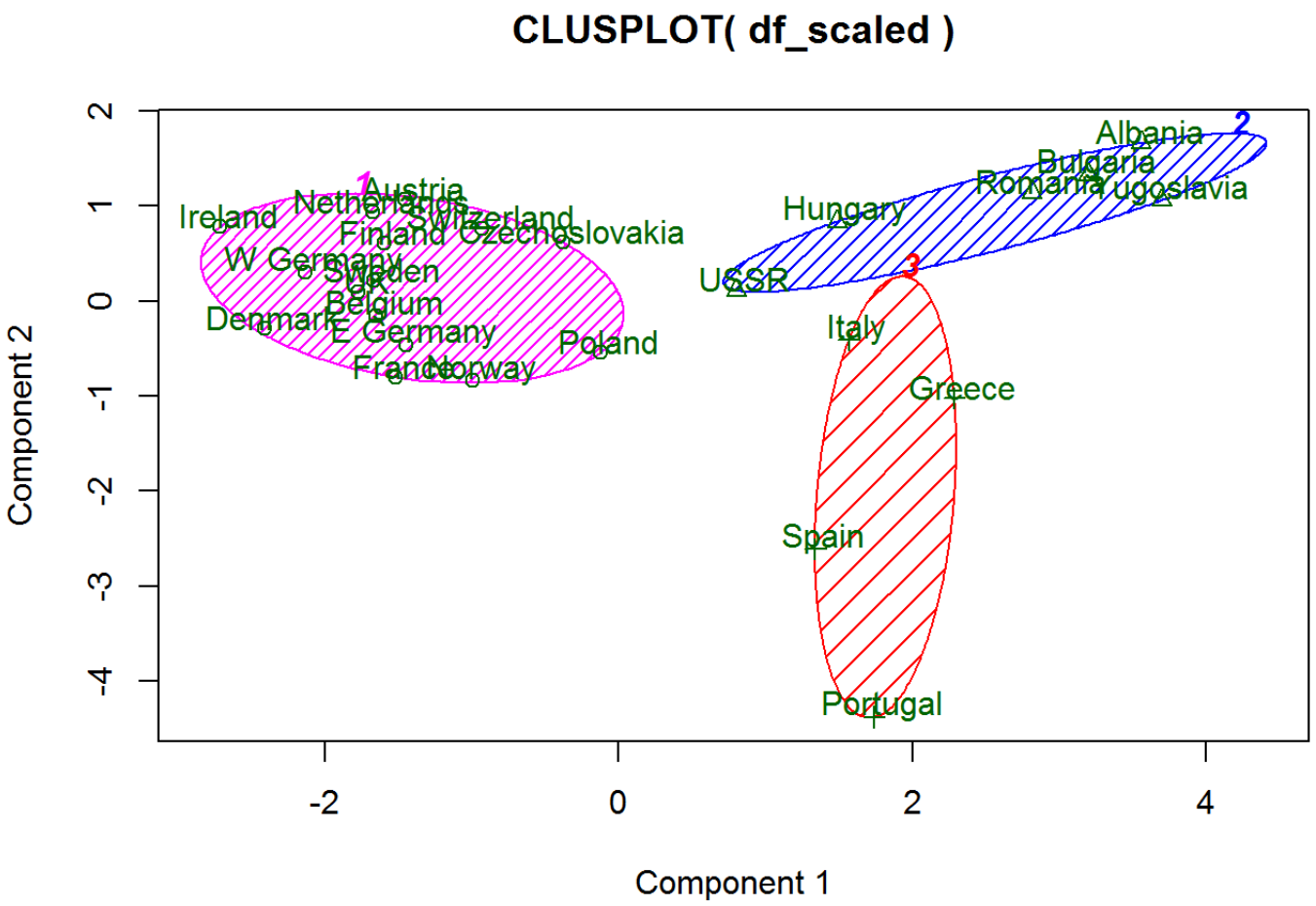
kmeansruns Will give us a way of choosing the number of clusters:

```
kmr_obj_ch = kmeansruns(df_scaled, criterion="ch")
clusplot(df_scaled , kmr_obj_ch$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```



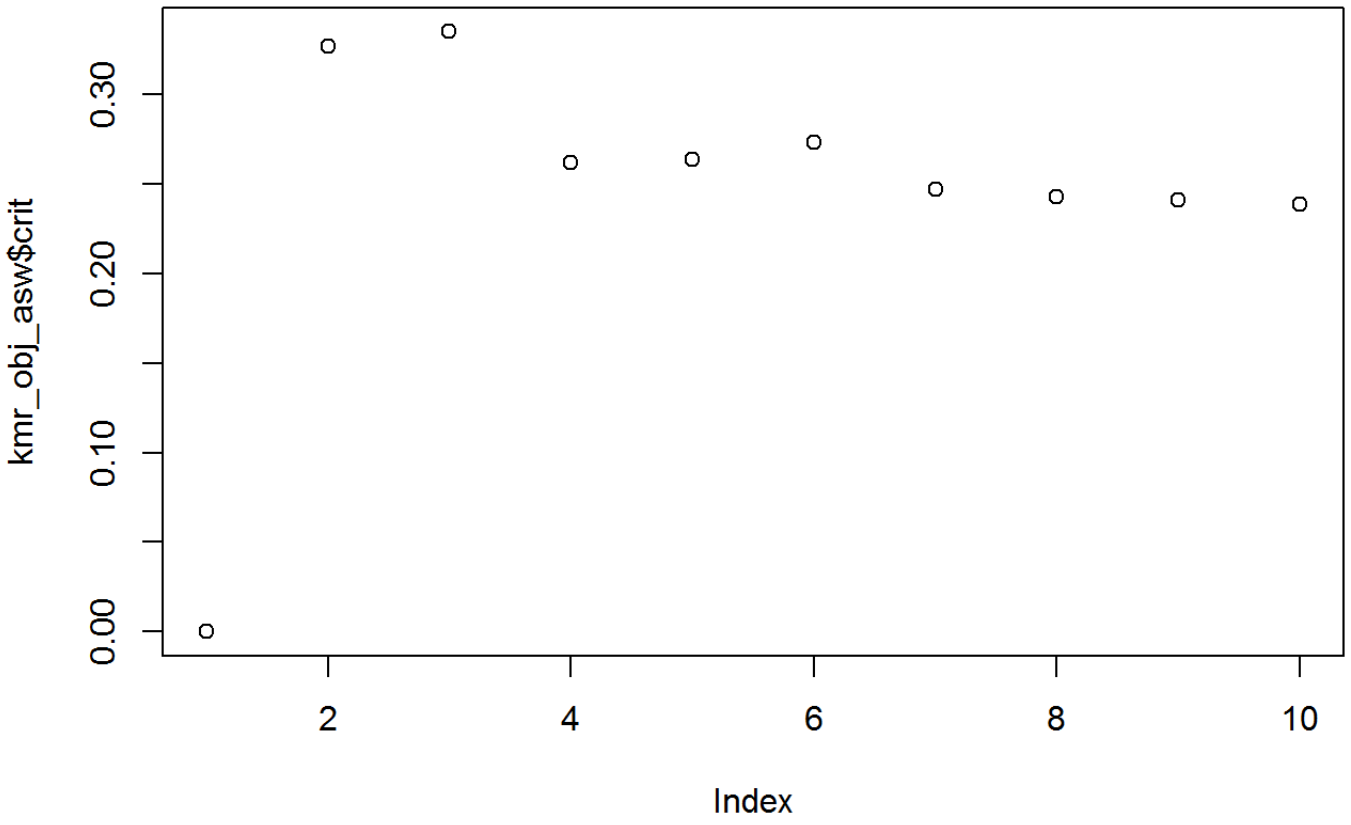
We can use a different criterion in *kmeansruns*

```
kmr_obj_asw = kmeansruns(df_scaled, criterion="asw")
clusplot(df_scaled , kmr_obj_asw$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```



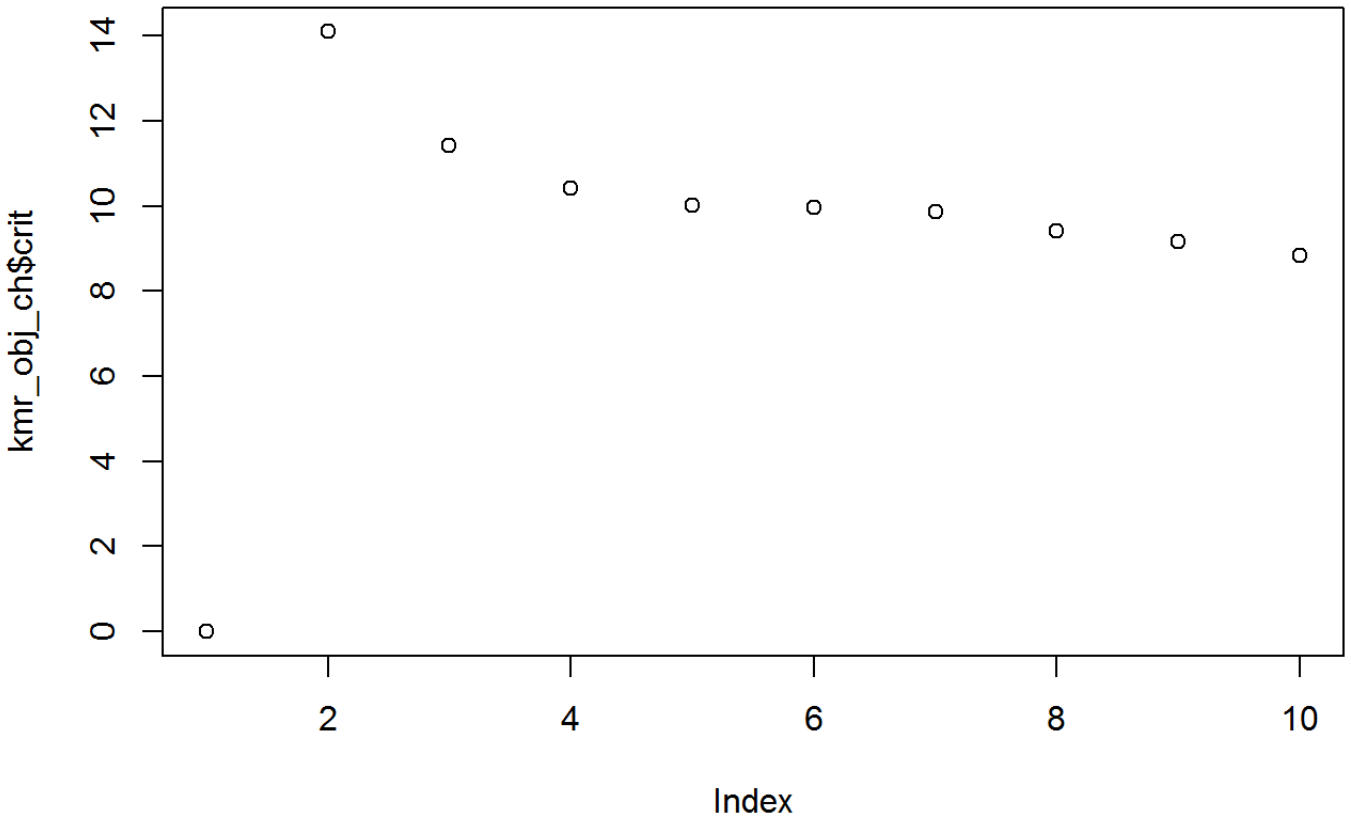
the *asw* criterion stably produces three clusters. We can plot the value of the criterion we are trying to optimise against the number of clusters, K. As we can see, the *asw* criterion is maximized at 3 clusters:

```
plot(kmr_obj_asw$crit)
```



And for the *ch* criterion:

```
plot(kmr_obj_ch$crit)
```

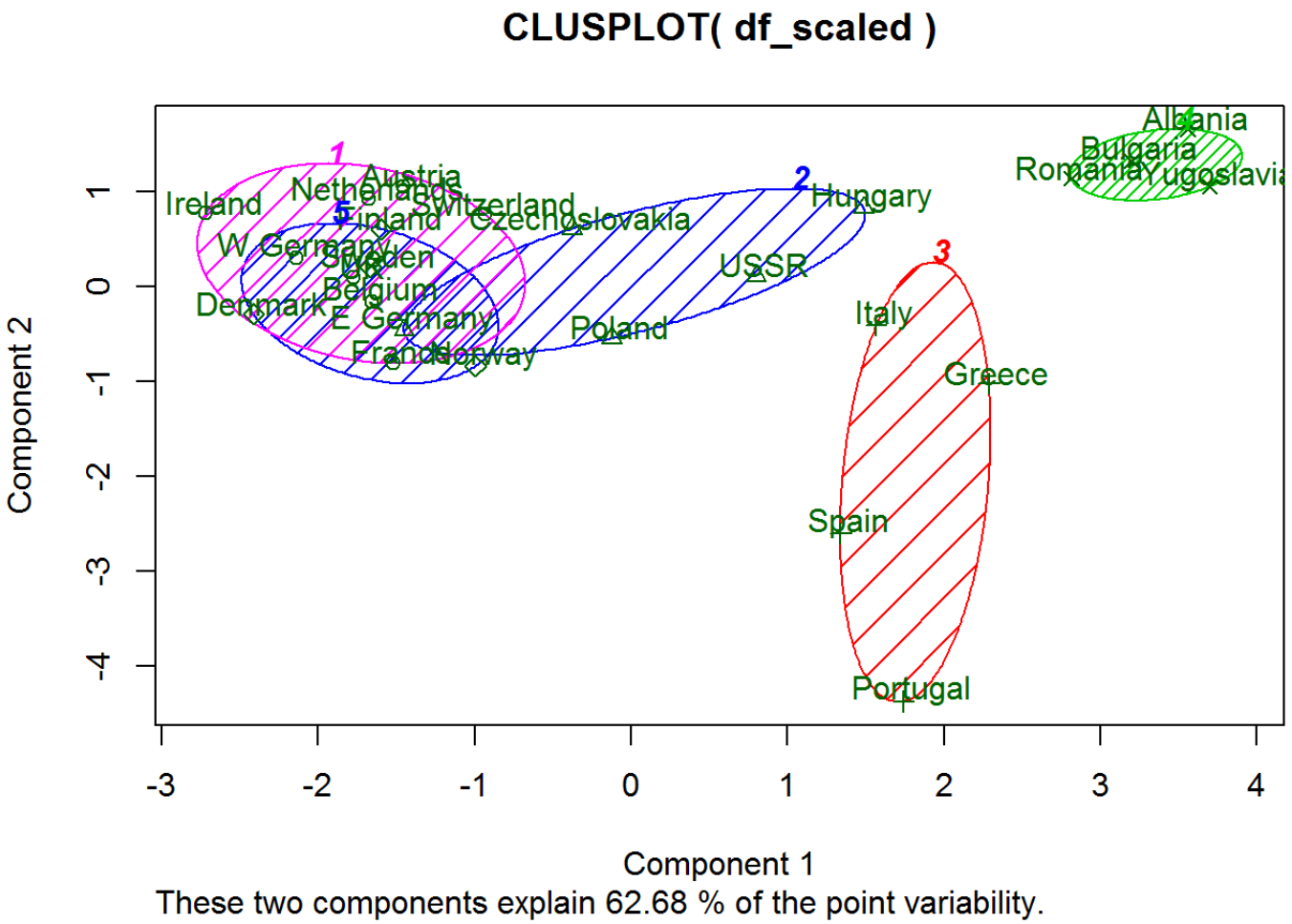


The *ch* criterion is maximized at 2 clusters.

Robustness analysis

We can also use clusterboot() from the fpc package to repeatedly resample our data with replacement, run K-means clustering on each bootstrapped sample, and test how often clusters are dissolved in the bootstrapped clusters. This gives us a notion of how “real” the clusters are.

```
cb_obj = invisible(clusterboot(df_scaled, clustermethod=kmeansCBI, runs=100, iter.max=100,
krange=5 ))
clusplot(df_scaled , cb_obj$result$partition, color=TRUE, shade=TRUE, labels=2, lines=0)
```




```
cb_obj$bootmean
```

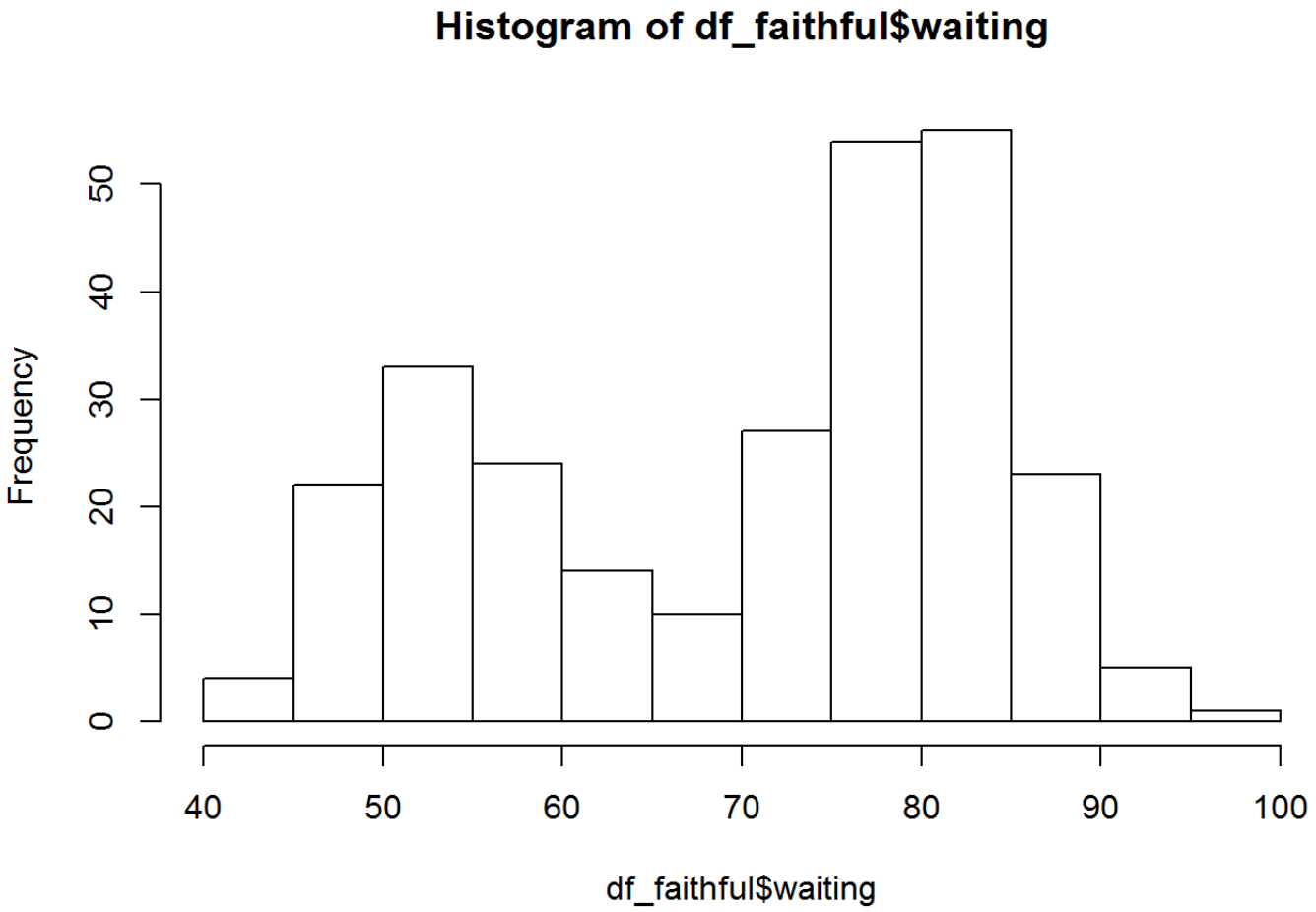
Cluster 3 is the most robust, cluster 4 the least.

Mixture models

old faithful eruptions

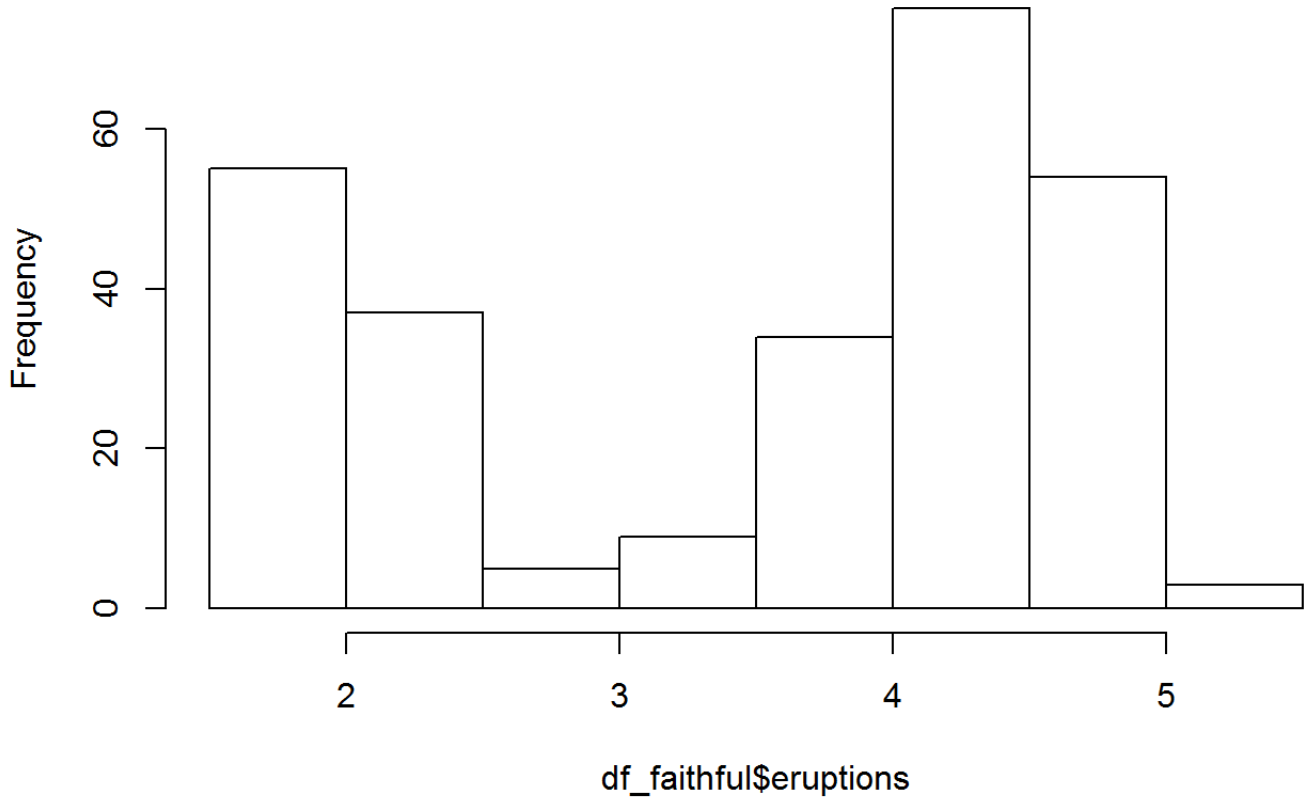
We will examine old faithful eruption times, looking to model them as a mixture model. First consider the waiting times:

```
df_faithful = faithful
hist(df_faithful$waiting)
```



And also the duration of the eruptions:

```
hist(df_faithful$eruptions )
```



Model the waiting times as a mixture of normals:

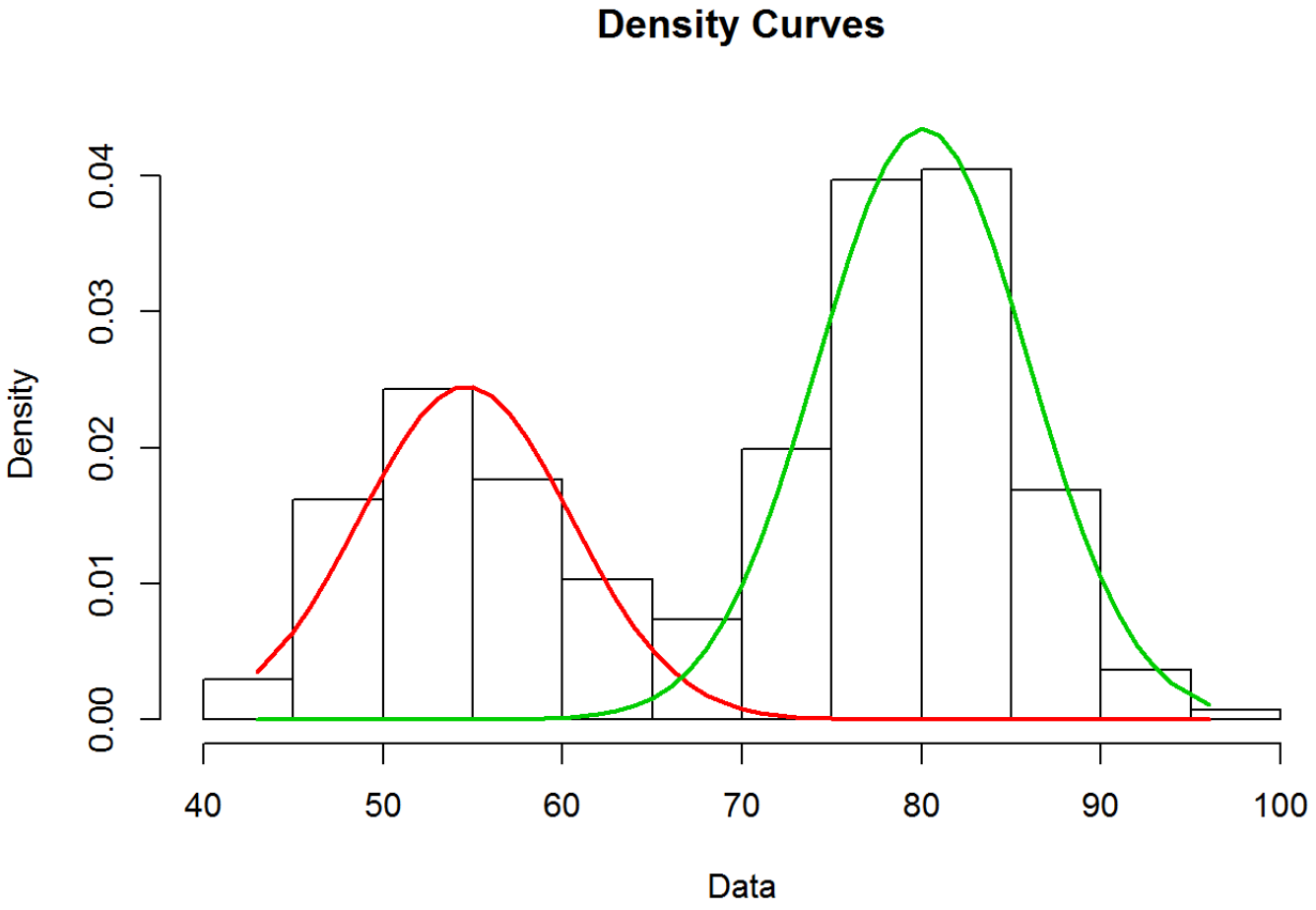
```
mix_obj = normalmixEM(df_faithful$waiting)
```

```
## number of iterations= 29
```

```
summary(mix_obj)
```

```
## summary of normalmixEM object:
##           comp 1    comp 2
## lambda  0.360887  0.639113
## mu     54.614888 80.091089
## sigma   5.871241  5.867718
## loglik at estimate:  -1034.002
```

```
plot(mix_obj, density=TRUE, which=2 )
```

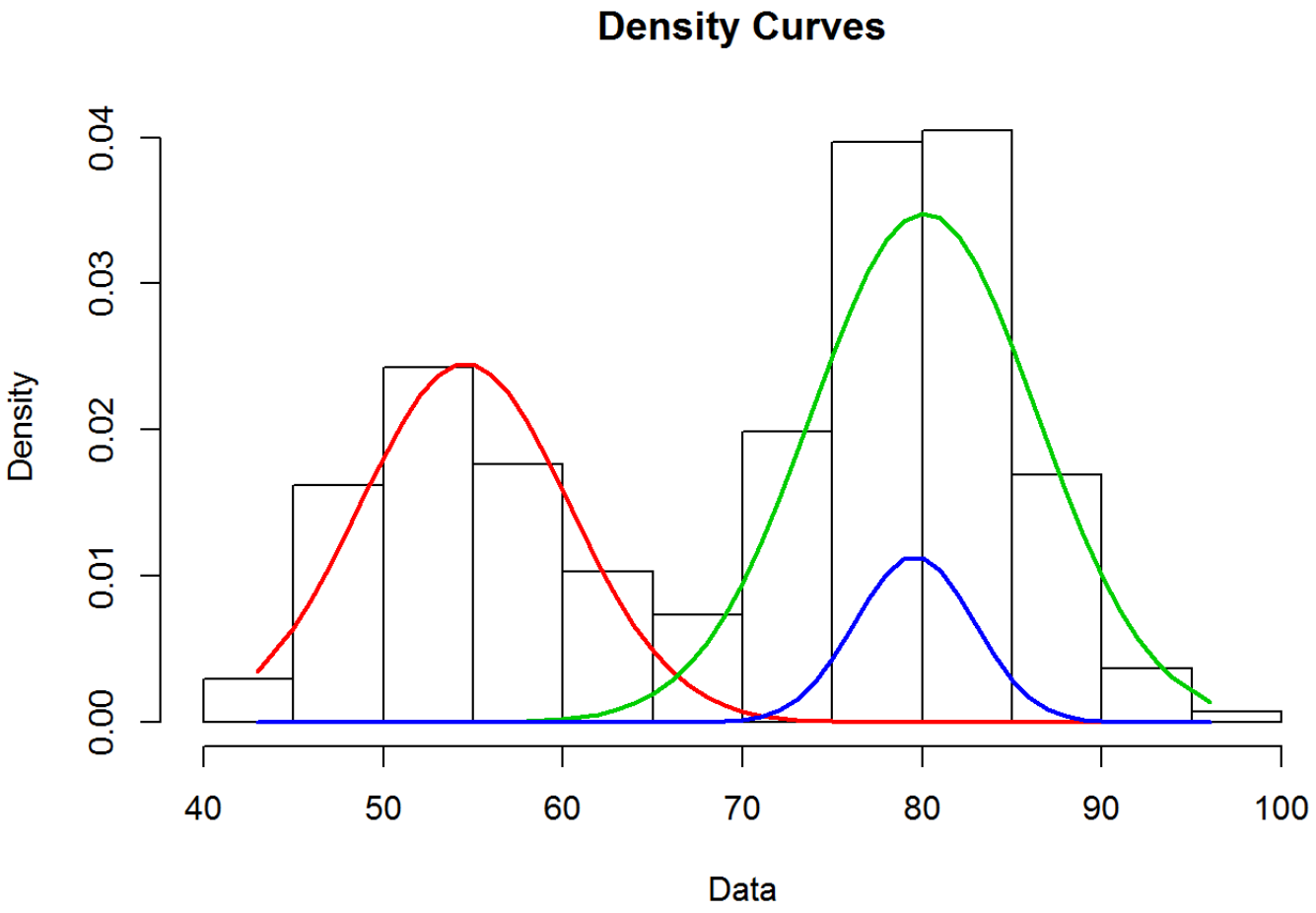


Try a mixture model with the “wrong” number of components:

```
mix_obj_3 = normalmixEM(df_faithful$waiting, k=3)
```

```
## number of iterations= 434
```

```
plot(mix_obj_3, density=TRUE, which=2 )
```

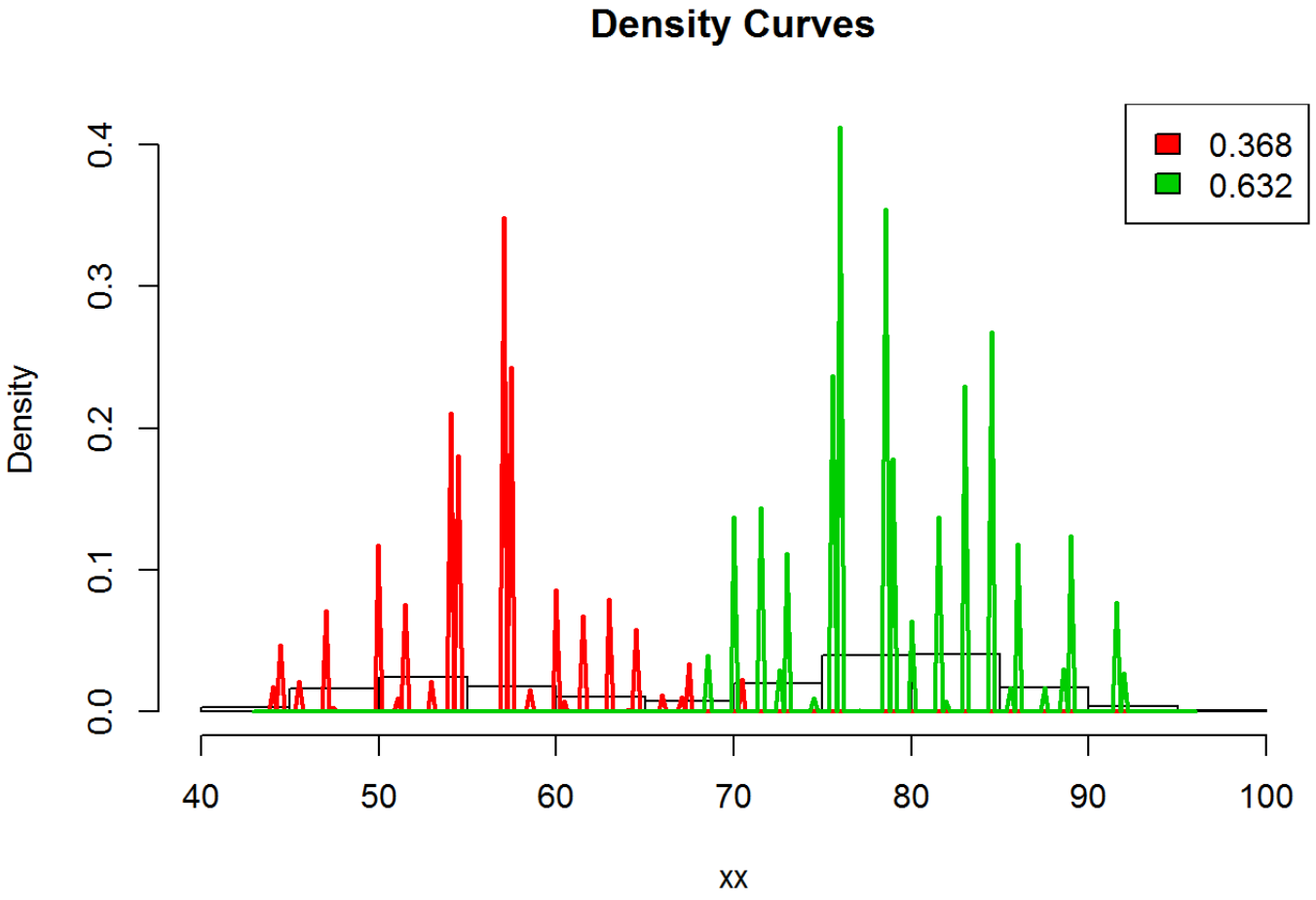


We can see that the result is unstable - you get wildly different parameters every time.

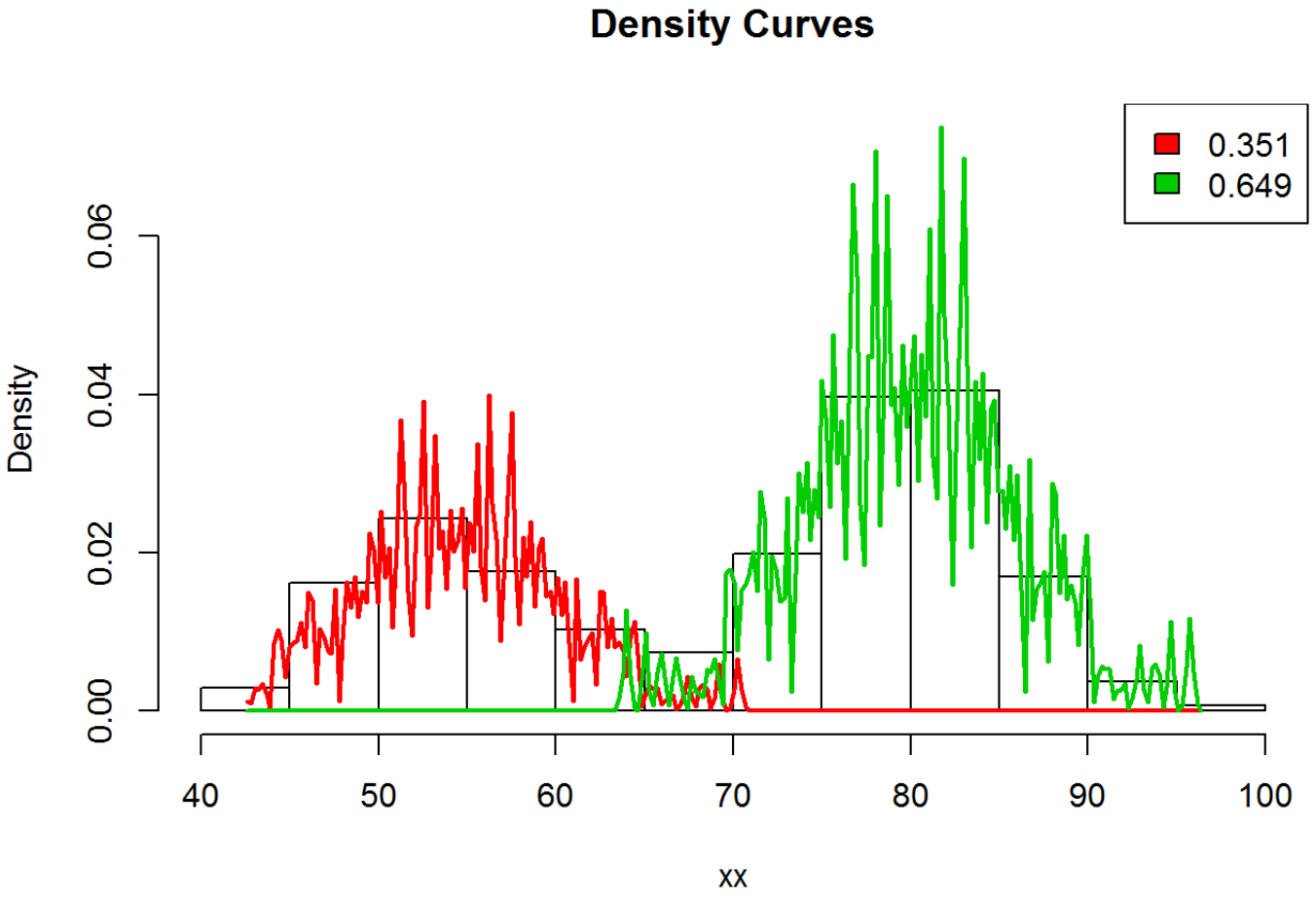
Semiparametric methods

We can use a semiparametric model:

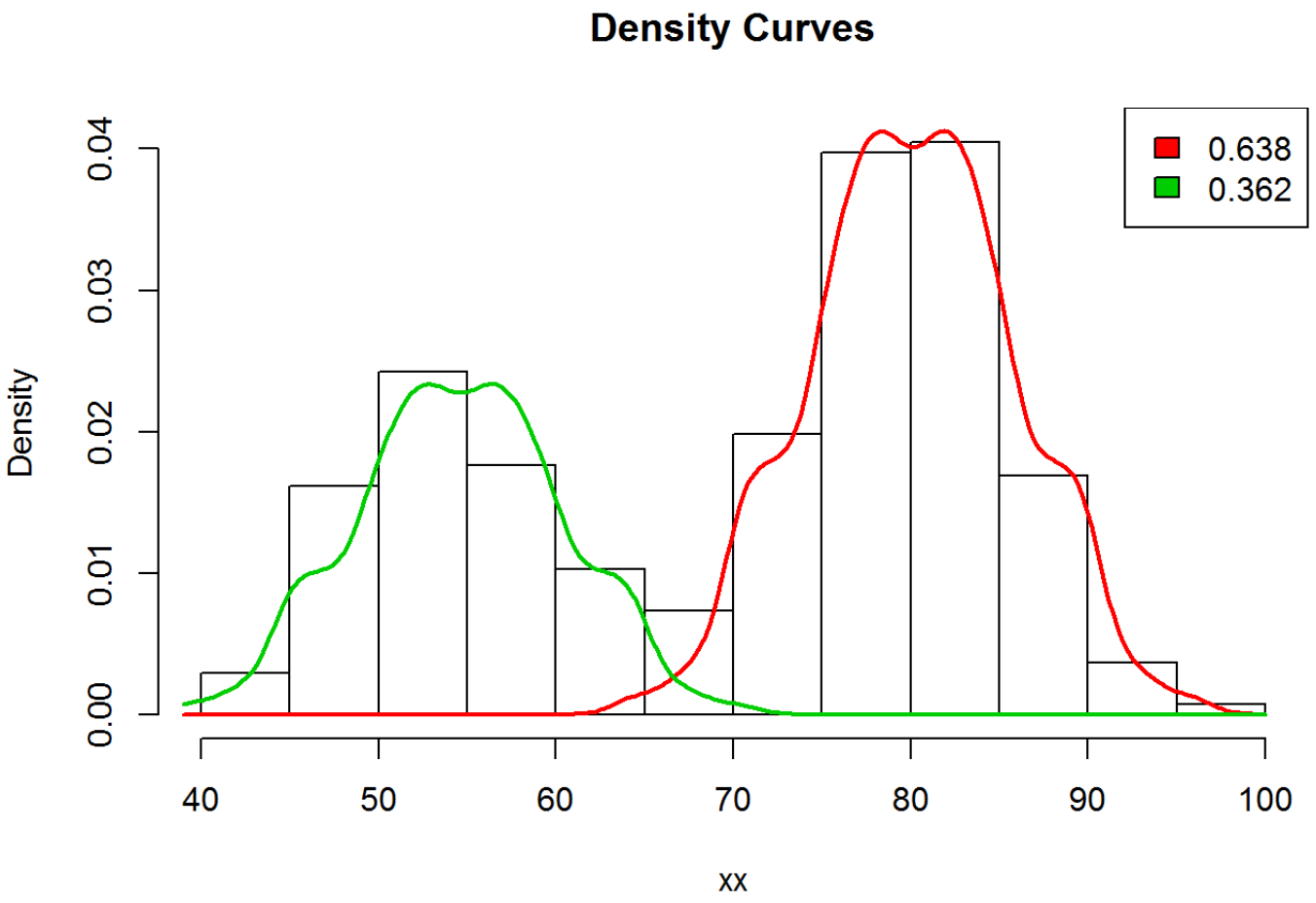
```
sp_obj = spEMsymloc(df_faithful$waiting, mu0 = 2, bw = 0.01)
plot(sp_obj)
```



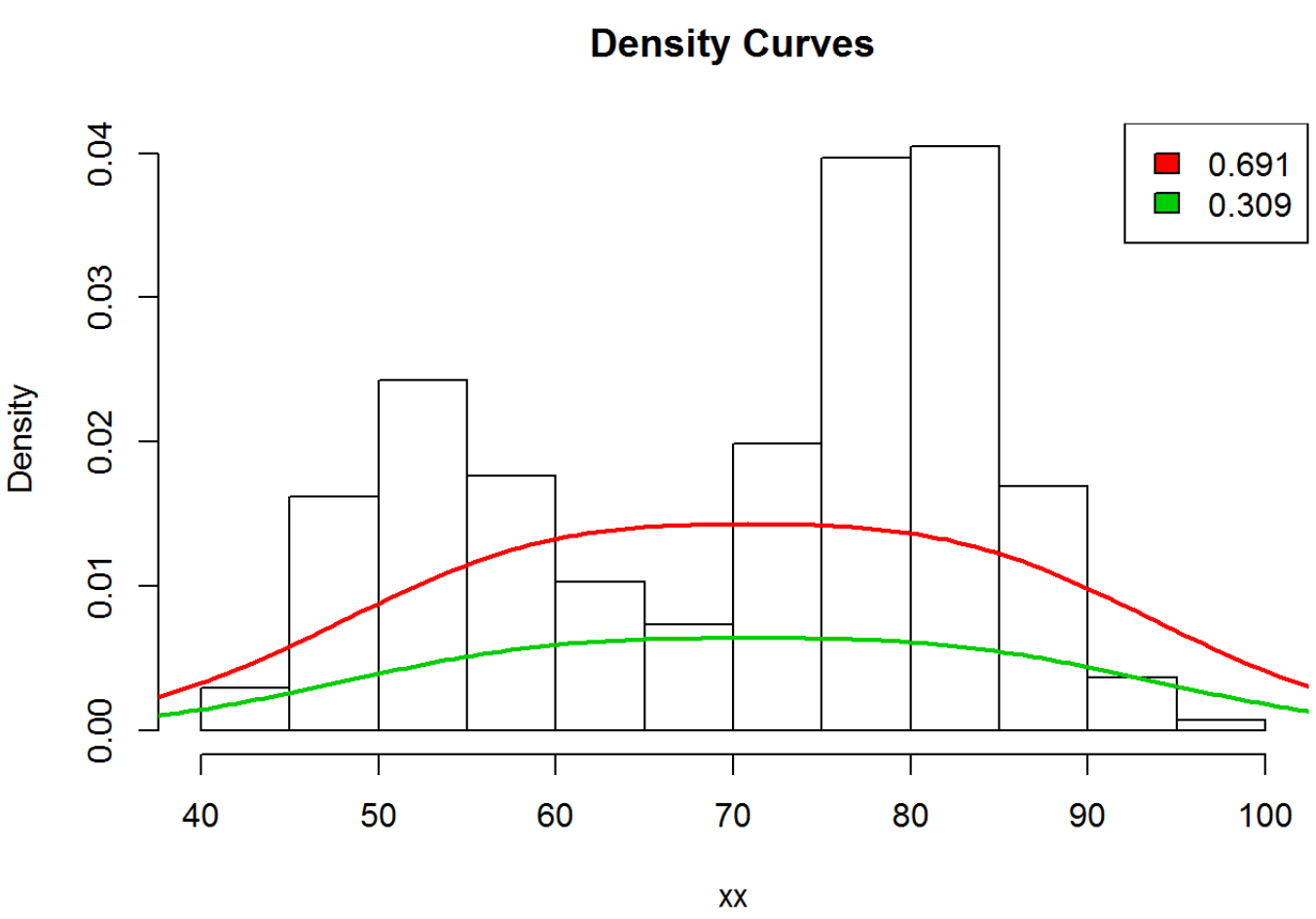

```
sp_obj = spEMsymloc(df_faithful$waiting, mu0 = 2, bw = 0.1)
plot(sp_obj)
```




```
sp_obj = spEMsymloc(df_faithful$waiting, mu0 = 2, bw = 1)
plot(sp_obj)
```




```
sp_obj = spEMsymloc(df_faithful$waiting, mu0 = 2, bw = 10)
plot(sp_obj)
```

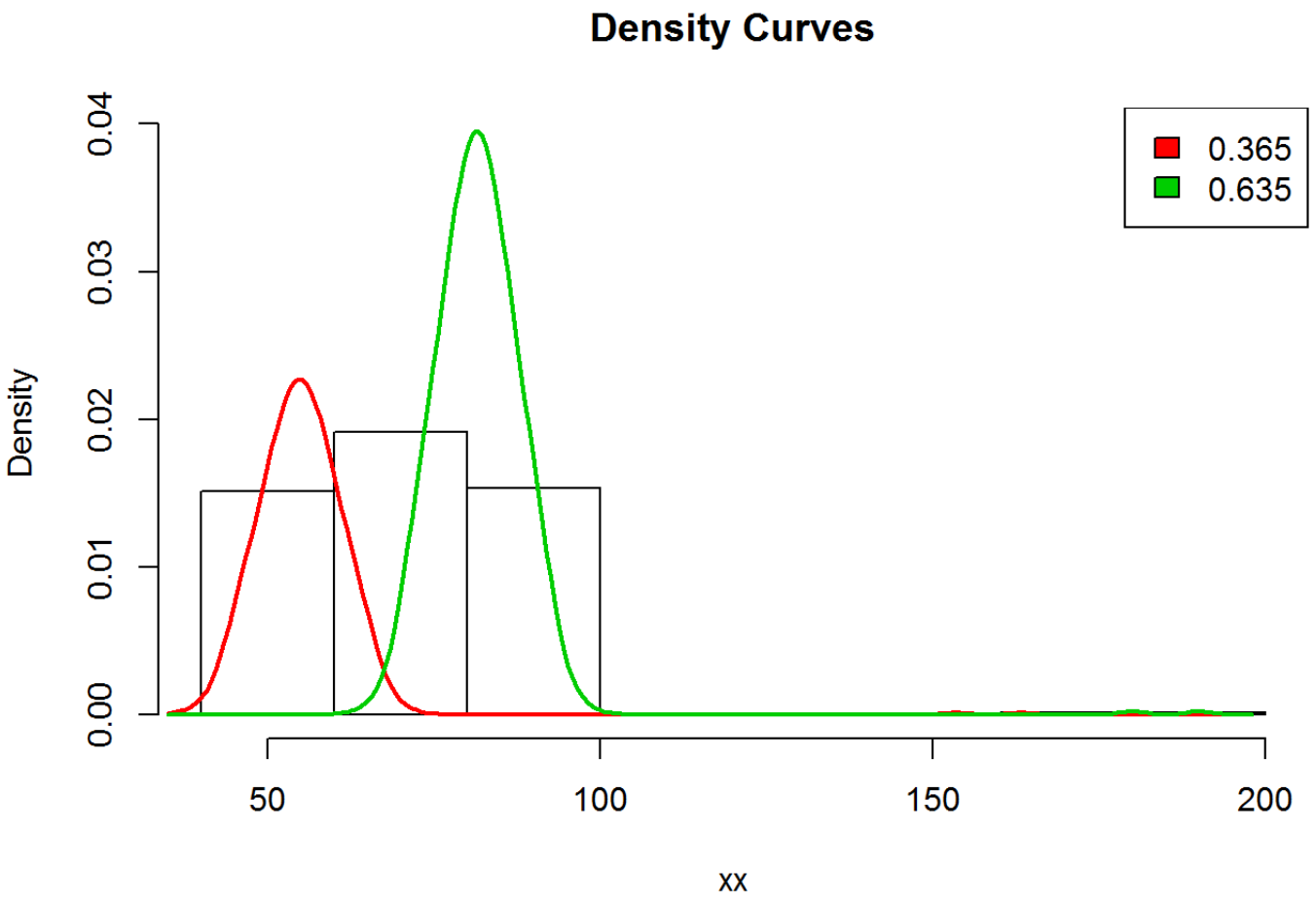


We can test the robustness of a semiparametric model to outliers:

```
df_faithful_ol = rbind(df_faithful, c(19, 190), c(23, 180) )
```

the semiparamatric model can be vulnerable to outliers with a bad bandwidth parameter

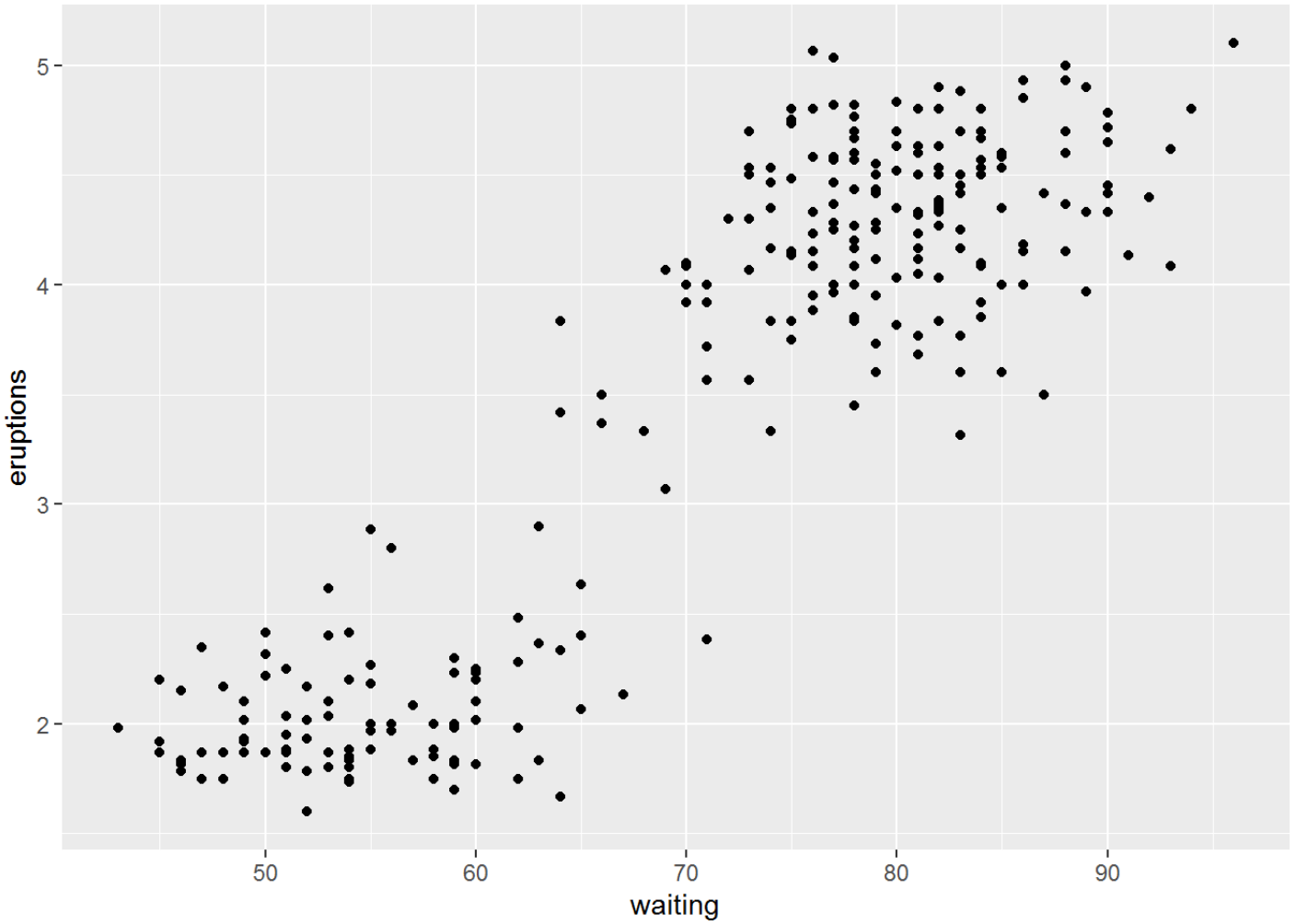
```
sp_obj_ol = spEMsymloc(df_faithful_ol$waiting, mu0 = 2, bw = 2)
plot(sp_obj_ol)
```



Multivariate mixture models

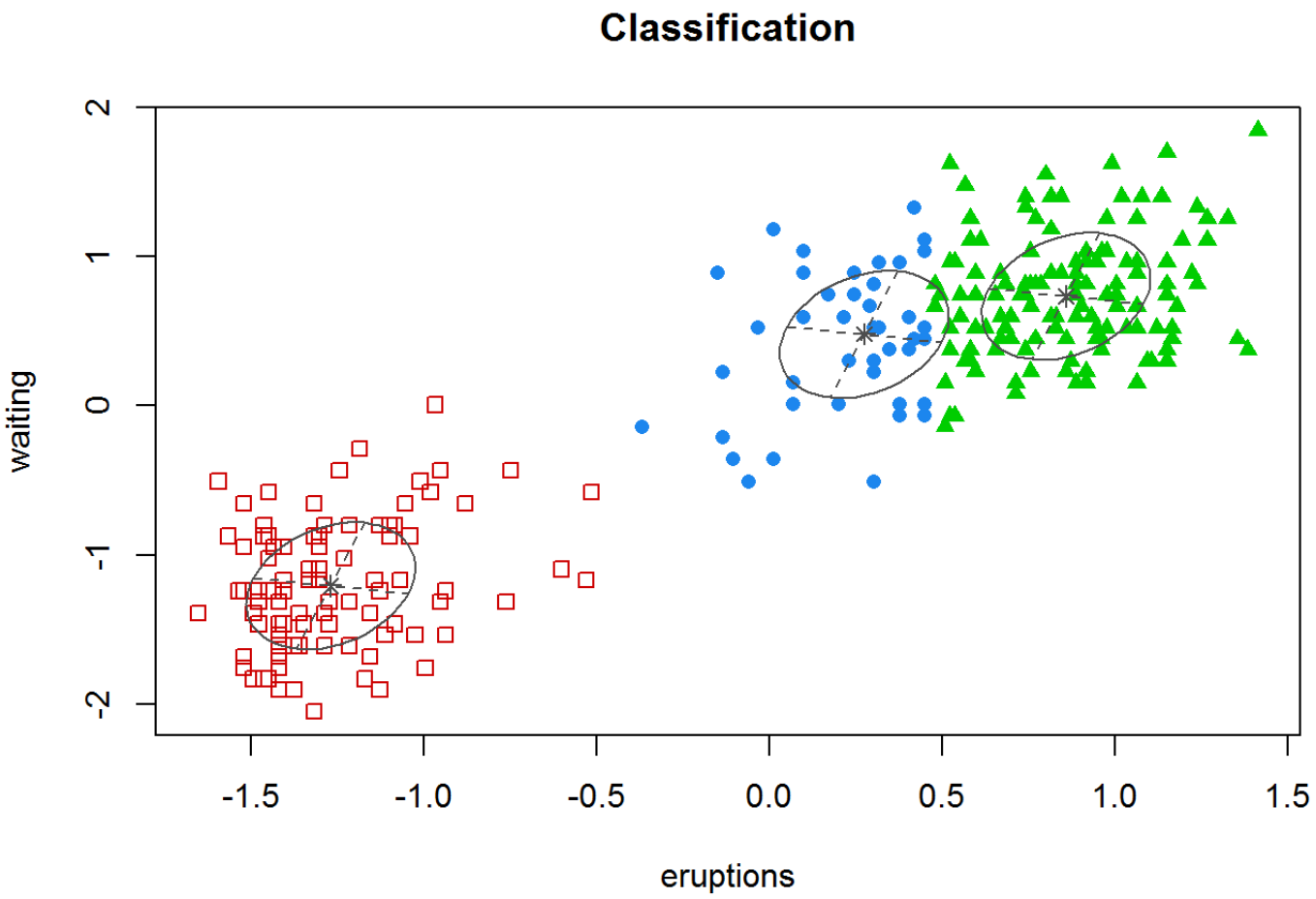
Looking at the old faithful data, there are clearly two clusters:

```
ggplot(data = df_faithful, aes(x = waiting, y = eruptions ) ) + geom_point()
```



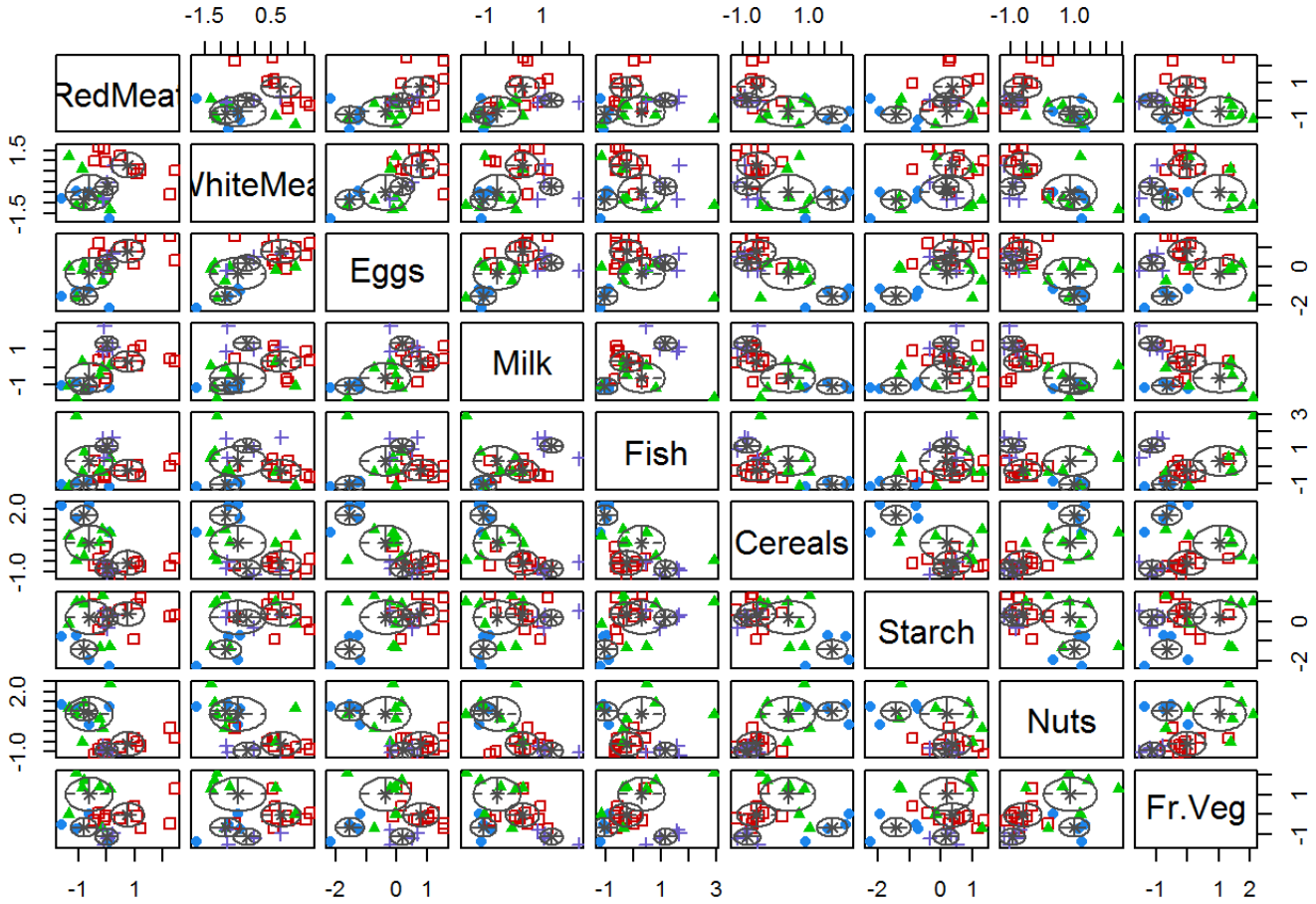
We would like to cluster them into two clusters. First we should scale the data.

```
df_fs = scale(df_faithful)
Lust_obj = Mclust(df_fs)
plot(Lust_obj, what = "classification")
```



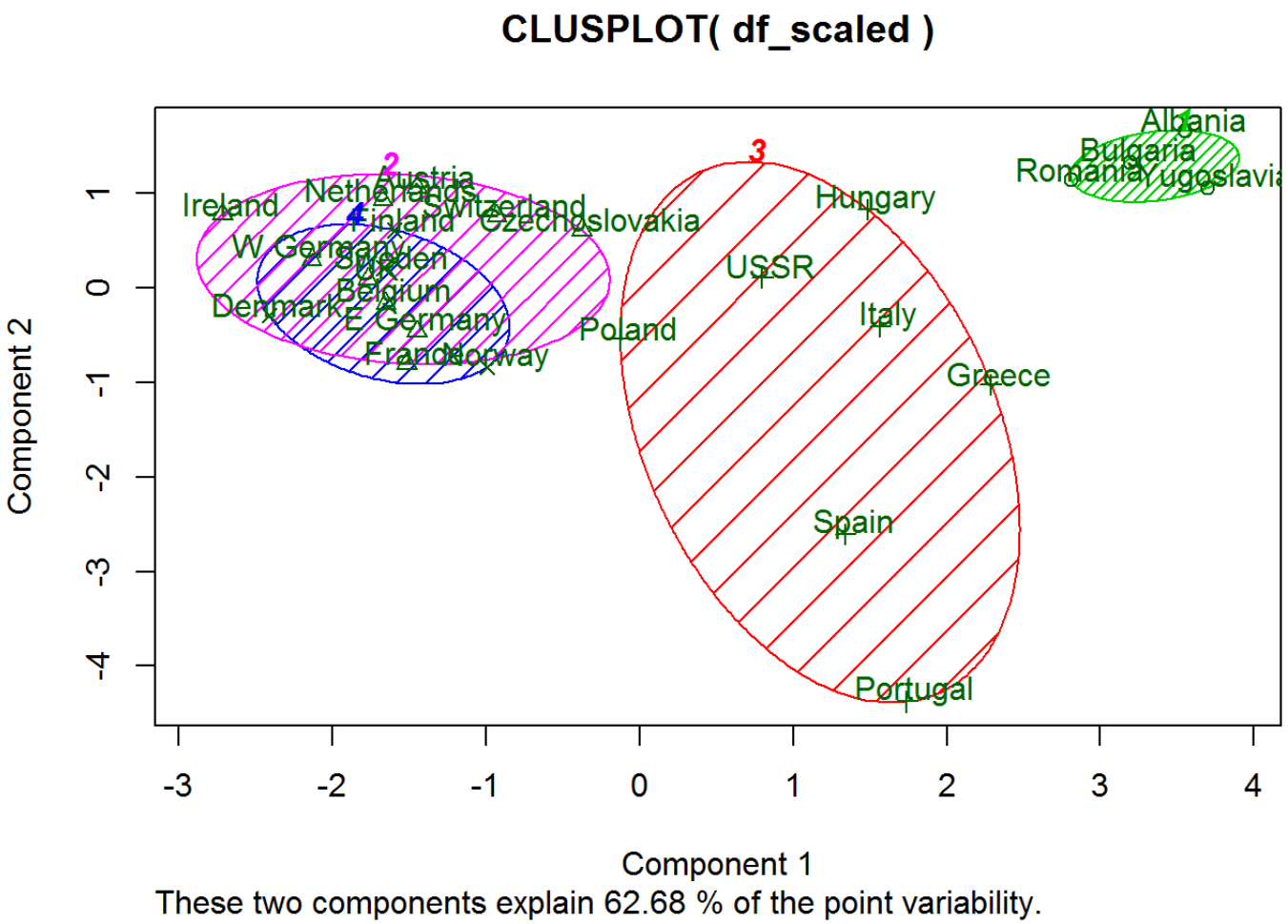
Try the same thing with the protein consumption data. The issue with this data is the larger number of dimensions makes it hard to understand the resulting clusters - since they are in high-dimensional space.

```
Lust_obj_pr = Mclust(df_scaled)
plot(Lust_obj_pr, what = "classification")
```



clusplot allows us to visualize the clusters from *mclust*:

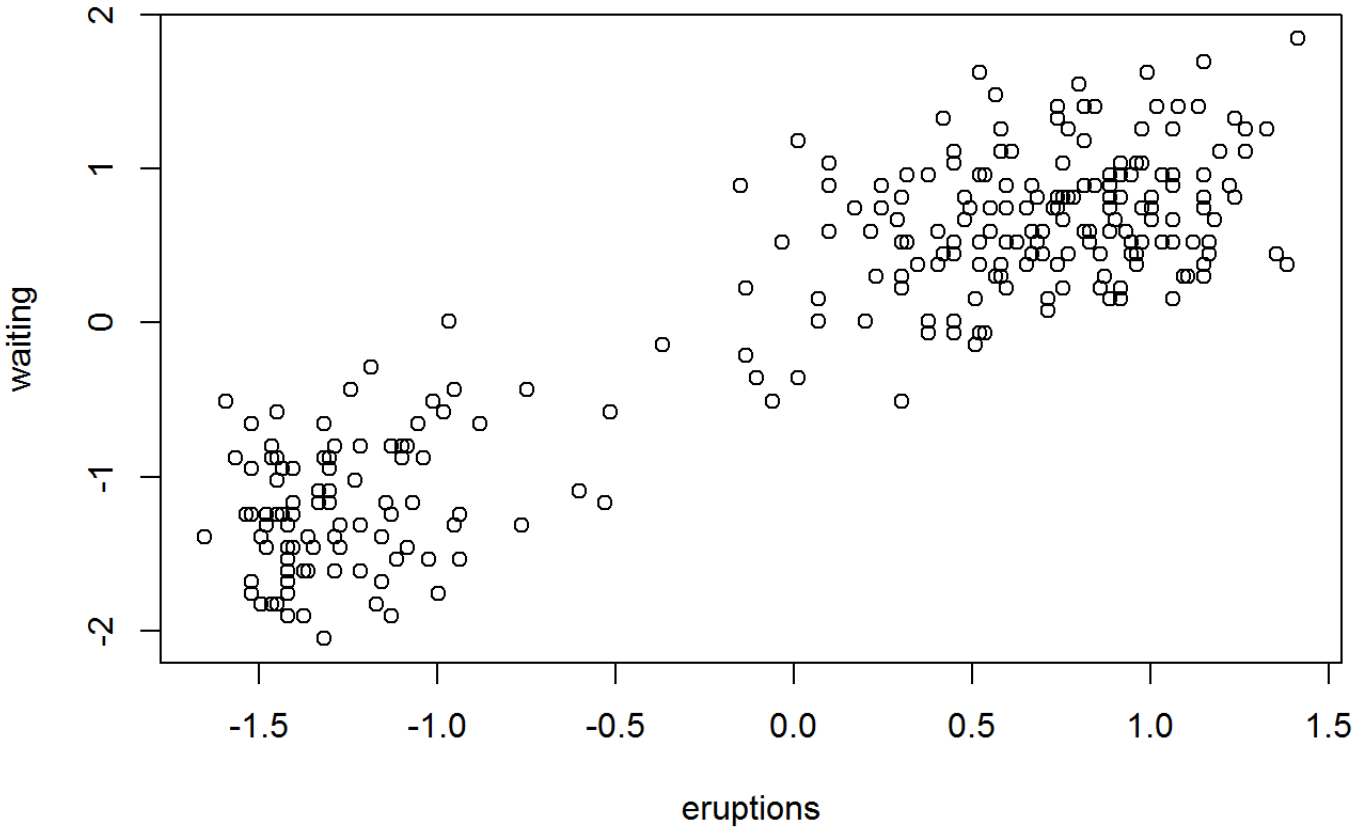
```
clusplot(df_scaled , Lust_obj_pr$classification, color=TRUE, shade=TRUE, labels=2, lines=0)
```



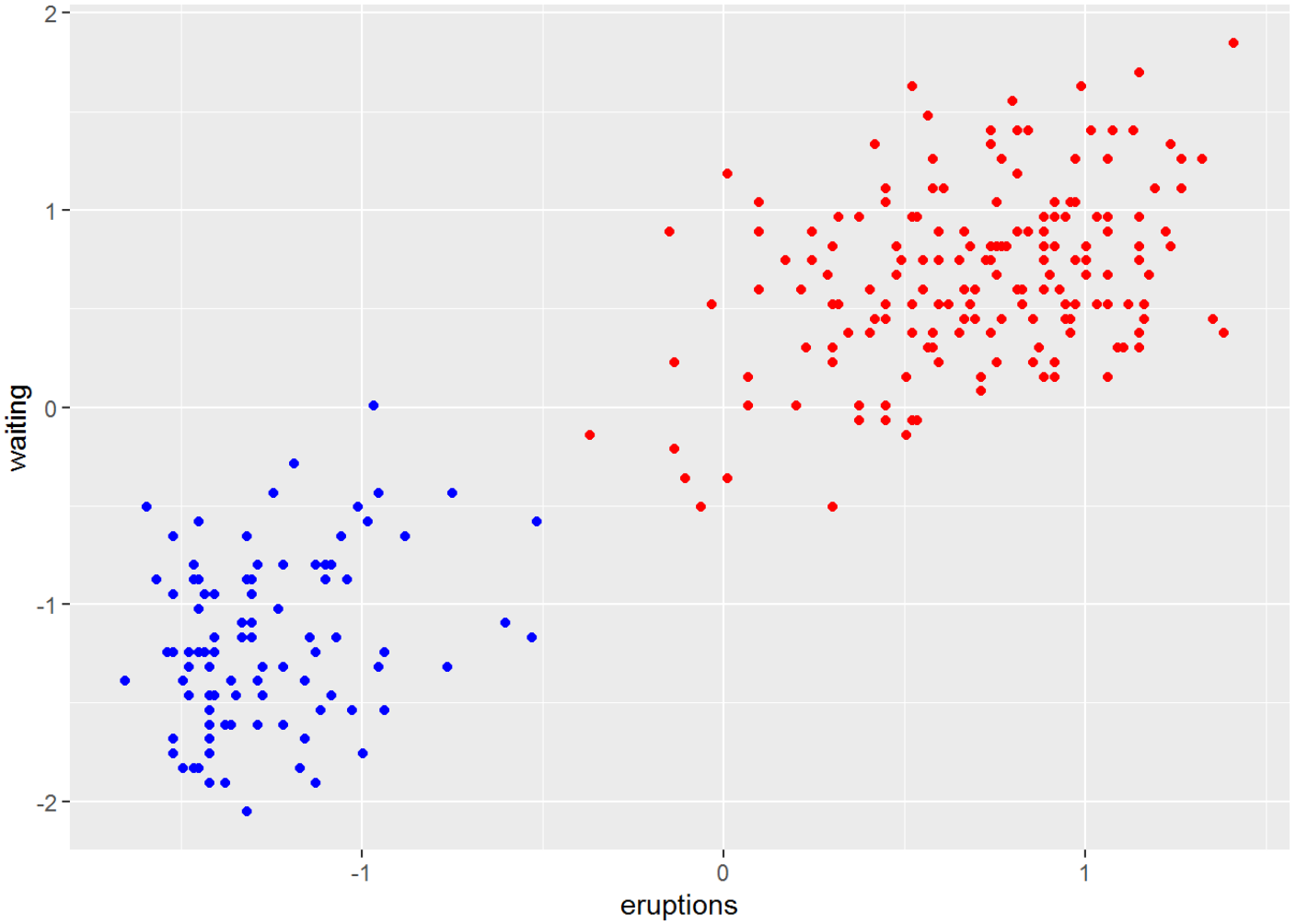
These two components explain 62.68 % of the point variability.

We can do the same thing with *npEM*:

```
plot(df_fs)
```




```
np_obj = npEM(df_fs, mu0 = 2 )
np_obj$posteriors
labels = apply(np_obj$posteriors, 1, which.max)
mypal = colorRampPalette( c( "red", "blue" ) )( 2 )
np_data = cbind( data.frame(df_fs) ,  labels  )
ggplot(data = np_data, aes(x=eruptions, y=waiting)) + geom_point( color = mypal[labels] )
```


2
1
0
-1
-2
waiting
-1
0
1
eruptions