# FEL results preliminary

sadie

3/4/2020

## check one alignment

### zeiformes nd1 FEL results

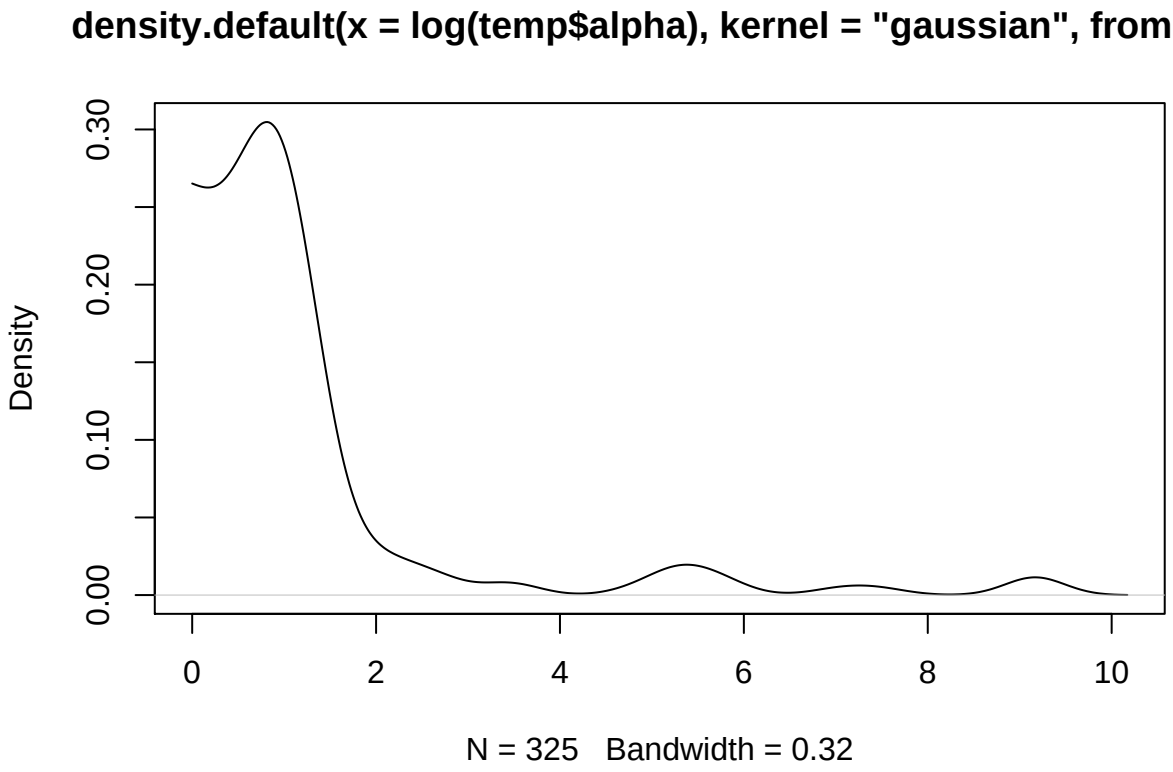
```
filepath <- read_json("~/bin/mtDNA_redo/data/FEL/zeiformes-nd1-align-dna.fas.FEL.json") #read in json

heads <- filepath$MLE$headers  %>% unlist() %>% .[c(TRUE,FALSE)] #get headers and ignore header descrip
```

```
#get MLE contents and make them a data frame
temp <- filepath$MLE$content$`0` %>% unlist %>% matrix(ncol = 6, byrow = TRUE) %>% as.data.frame()
#make the headers the variable names
 names(temp)  <- heads
```

## kernel density and plots?

### Kernel density and plot for alpha for single FEL result

```
d_alpha <- density(log(temp$alpha), kernel = "gaussian", from = 0) #Kernel density for alpha rate estim
#start from 0 and log transform to control outliers

d_alpha %>% plot() #plot the density
```

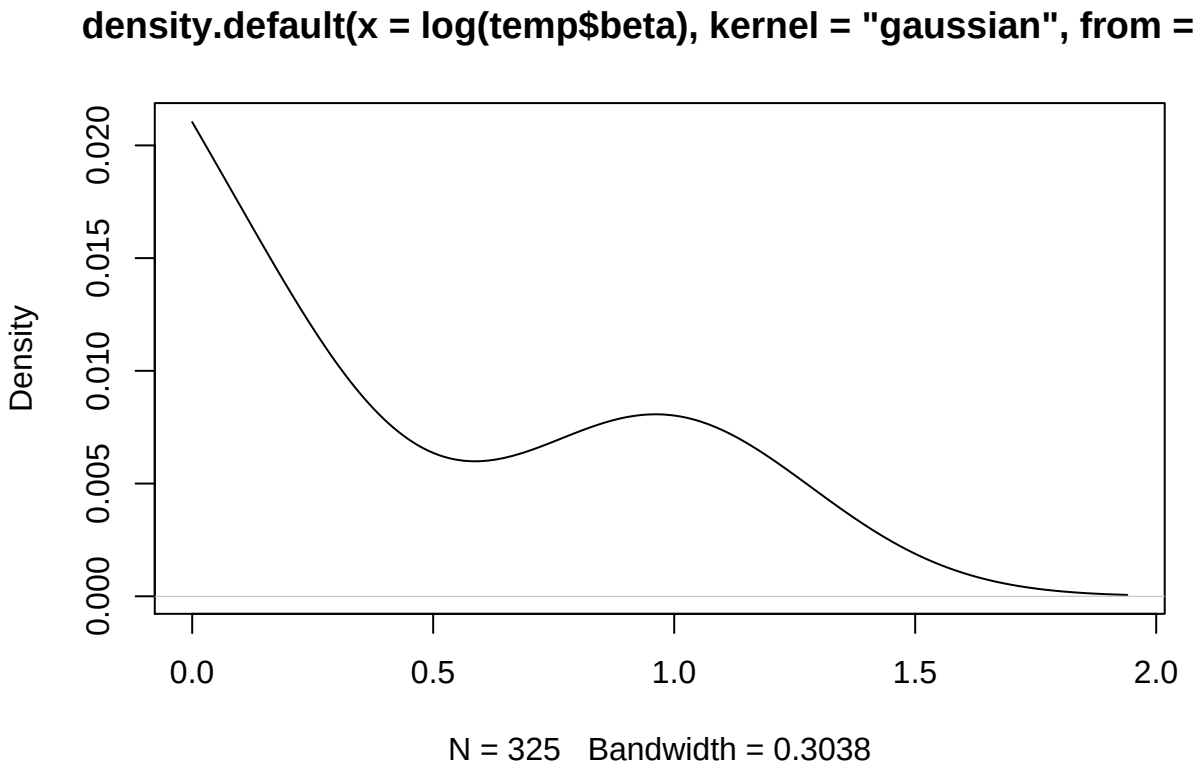



what is the variability of the distribution?

```
d_a_var <- d_alpha$x %>% var()

# t.a <- temp$alpha
# t.a[which(temp$alpha<=0)]  <- 0
# t.a[which(temp$alpha== -Inf)]  <- 0
#   log(t.a)%>% var()
```

### Kernel density and plot for beta for single FEL result

```
d_beta <- density(log(temp$beta), kernel = "gaussian", from = 0)  #Kernel density for the beta rate est

d_beta %>% plot() #plot the density
```




## check a second alignment:

### acipenseriformes at8 FEL results

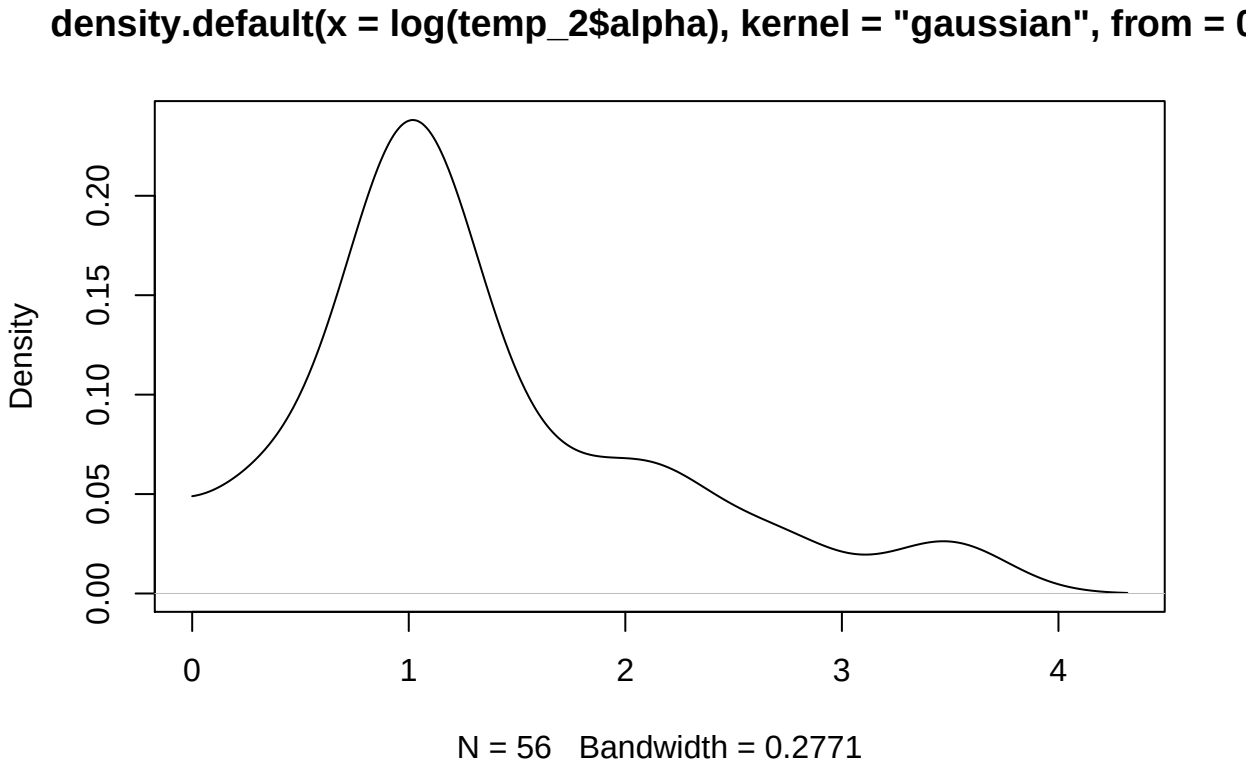
```
filepath <- read_json("~/bin/mtDNA_redo/data/FEL/acipenseriformes-atp8-align-dna.fas.FEL.json") #read i

heads <- filepath$MLE$headers  %>% unlist() %>% .[c(TRUE,FALSE)] #get headers and ignore header descrip
```

```
#get MLE contents and make them a data frame
temp_2 <- filepath$MLE$content$`0` %>% unlist %>% matrix(ncol = 6, byrow = TRUE) %>% as.data.frame()
#make the headers the variable names
 names(temp_2)  <- heads
```

## kernel density and plots?

### Kernel density and plot for alpha for single FEL result

```
d_alpha_2 <- density(log(temp_2$alpha), kernel = "gaussian", from = 0)

d_alpha_2 %>% plot()
```

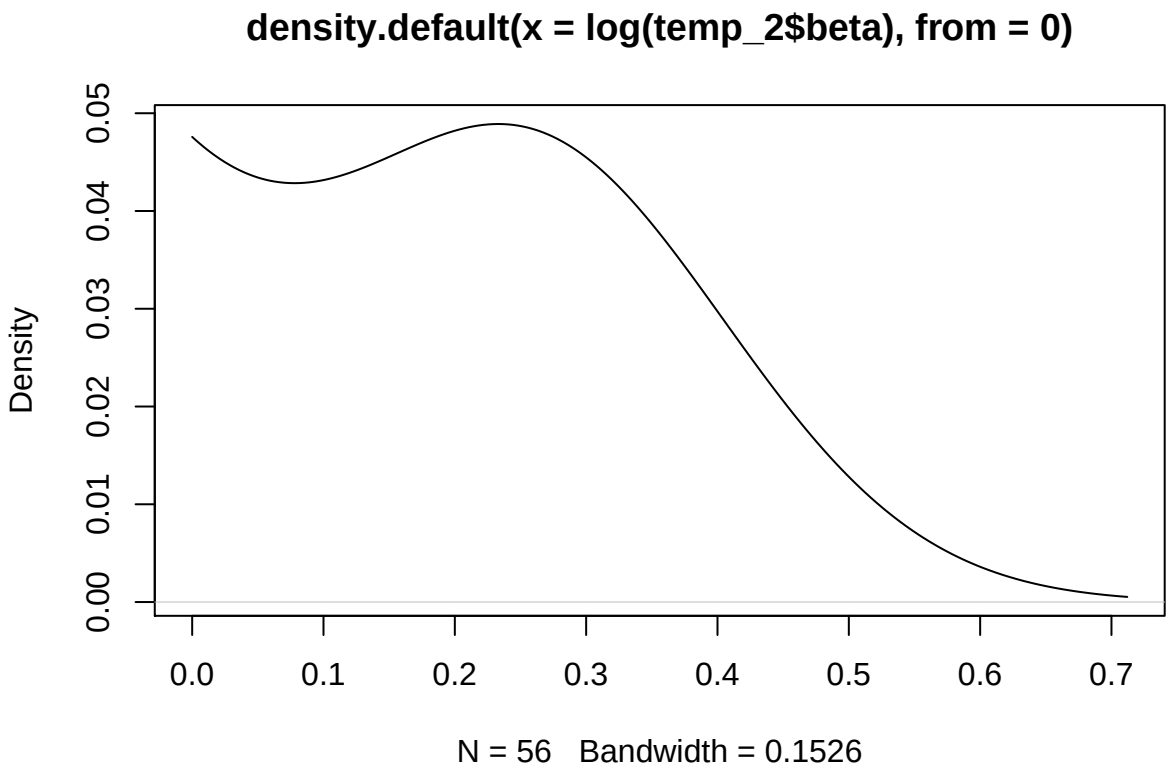



what is the variability of the distribution?

```
d_a_2_var <- d_alpha_2$x %>% var()
```

### Kernel density and plot for beta for single FEL result

```
d_beta_2 <- density(log(temp_2$beta), from = 0)
d_beta_2 %>% plot()
```




#try different ways of calculating the KL or JSD ##using the entropy library:

```
#uses entropy library to calculate Kullback-Leibler divergence (KL) as it is needed to do the JSD
library("entropy")

KL <- KL.plugin(freqs1 = d_alpha$x, freqs2 = d_beta$x)
```

```
## Warning in KL.plugin(freqs1 = d_alpha$x, freqs2 = d_beta$x): Vanishing value(s)
## in argument freqs2!
```

```
KL_2 <- KL.plugin(freqs1 = d_alpha_2$x, freqs2 = d_beta_2$x)
```

```
## Warning in KL.plugin(freqs1 = d_alpha_2$x, freqs2 = d_beta_2$x): Vanishing
## value(s) in argument freqs2!
```

```
#both of these return an error: Vanishing value(s) in argument freqs2!
```

### Using a calculation from stackoverflow:

```
#from stackoverflow https://stackoverflow.com/questions/11226627/jensen-shannon-divergence-in-r
p <- d_beta_2$x
q <- d_alpha_2$x
n <- 0.5 * (p + q)
JS <- 0.5 * (sum(p * log(p / n)) + sum(q * log(q / n))) #returns an NaN
```

### Using a formula from a website:

```
#actually, lets try the way this site does it: https://enterotype.embl.de/enterotypes.html

JSD<- function(x,y) sqrt(0.5 * KLD(x, (x+y)/2) + 0.5 * KLD(y, (x+y)/2))
```

```
KLD <- function(x,y) sum(x * log(x/y))

KLD(d_beta_2$x, d_alpha_2$x)
```

```
## [1] NaN
```

```
JSD(d_beta_2$x, d_alpha_2$x)
```

```
## [1] NaN
```

```
 dist.JSD <- function(inMatrix, pseudocount=0.000001, ...) {
    KLD <- function(x,y) sum(x *log(x/y))
    JSD<- function(x,y) sqrt(0.5 * KLD(x, (x+y)/2) + 0.5 * KLD(y, (x+y)/2))
    matrixColSize <- length(colnames(inMatrix))
    matrixRowSize <- length(rownames(inMatrix))
    colnames <- colnames(inMatrix)
    resultsMatrix <- matrix(0, matrixColSize, matrixColSize)

  inMatrix = apply(inMatrix,1:2,function(x) ifelse (x==0,pseudocount,x))

    for(i in 1:matrixColSize) {
        for(j in 1:matrixColSize) {
            resultsMatrix[i,j]=JSD(as.vector(inMatrix[,i]),
            as.vector(inMatrix[,j]))
        }
    }
    colnames -> colnames(resultsMatrix) -> rownames(resultsMatrix)
    as.dist(resultsMatrix)->resultsMatrix
    attr(resultsMatrix, "method") <- "dist"
    return(resultsMatrix)
 }
```

**calculate the JSD between alpha and beta**

```
d.temp <- dist.JSD(temp %>% select(alpha, beta))

d.temp
```

```
##         alpha
## beta 108.0882
```

this last method works and gives me the JSD between the nonsynonymous (beta) and synonymous (alpha) rate distributions.

so now what if we want the distance for every FEL result?

first, get list of all FEL results:

```
dir <- "~/bin/mtDNA_redo/data/FEL"
FEL_jsons <- list.files(path = dir,
                        pattern = '*FEL.json', recursive = TRUE, full.names = TRUE)
```

write function for JSD calculation:

```
d.temp <- NULL
JSD.calc <- function(filename){

  results <- read_json(filename) #read in json
```

```
  heads <- results$MLE$headers  %>% unlist() %>% .[c(TRUE,FALSE)] #get headers and ignore header descri

  #get MLE contents and make them a data frame
  temp <- results$MLE$content$`0` %>% unlist %>% matrix(ncol = 6, byrow = TRUE) %>% as.data.frame()
#make the headers the variable names
  names(temp)  <- heads

#Kernel density for alpha rate estimation
#start from 0 and log transform to control outliers
   d_alpha <- density(log(temp$alpha), kernel = "gaussian", from = 0)
   d_beta <- density(log(temp$beta), kernel = "gaussian", from = 0)

  d_alpha_var<- d_alpha$x %>% var()
  d_beta_var<- d_beta$x %>% var()

  dist.temp <- dist.JSD(temp %>% select(alpha, beta))
  d.temp <- cbind(d.temp, filename, dist.temp, d_alpha_var, d_beta_var)

  return(d.temp)
}
```

check that it works:

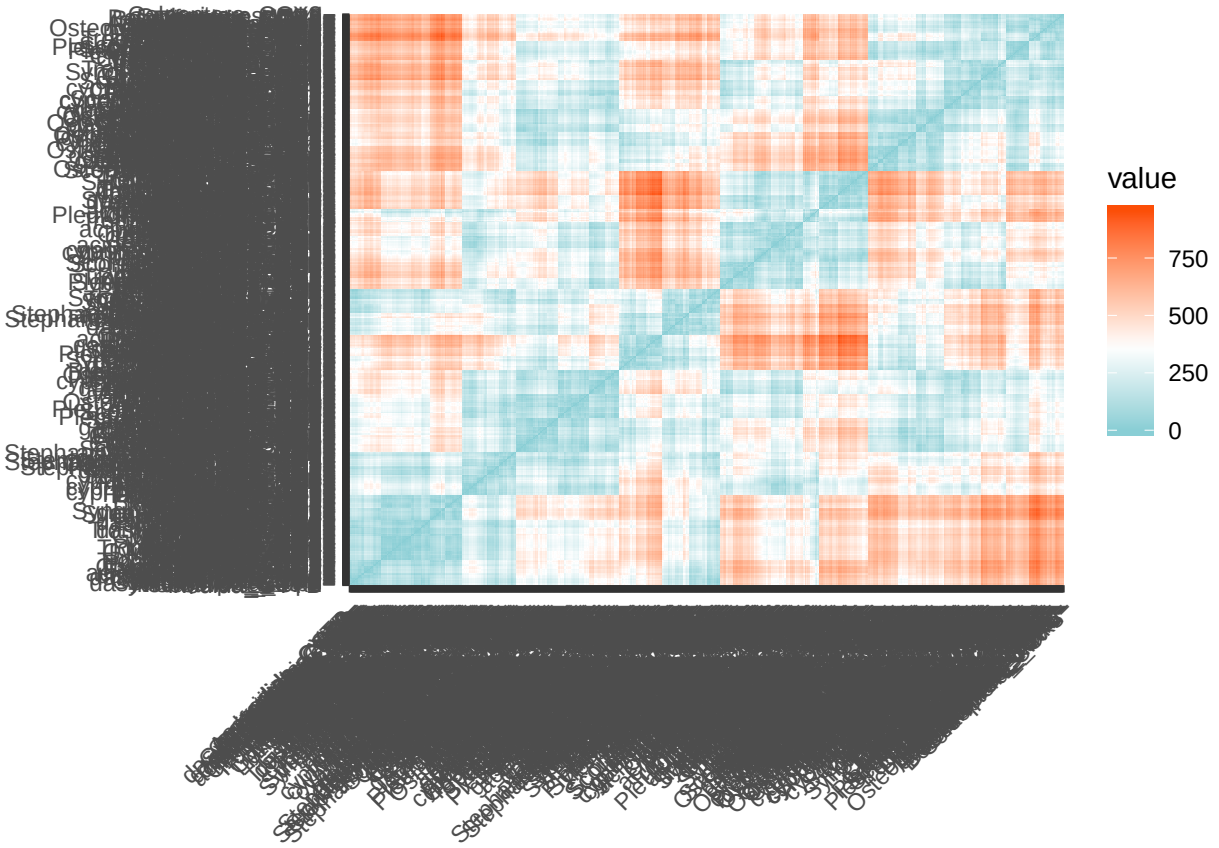
```
JSD.calc(paste0(FEL_jsons[1]))
```

```
##      filename
## [1,] "/Users/Sadie/bin/mtDNA_redo/data/FEL/acipenseriformes-atp6-align-dna.fas.FEL.json"
##      dist.temp          d_alpha_var        d_beta_var
## [1,] "60.1693123631767" "8.64937340783265" "0.136574441064125"
```

this returns a named number of JSD calculations

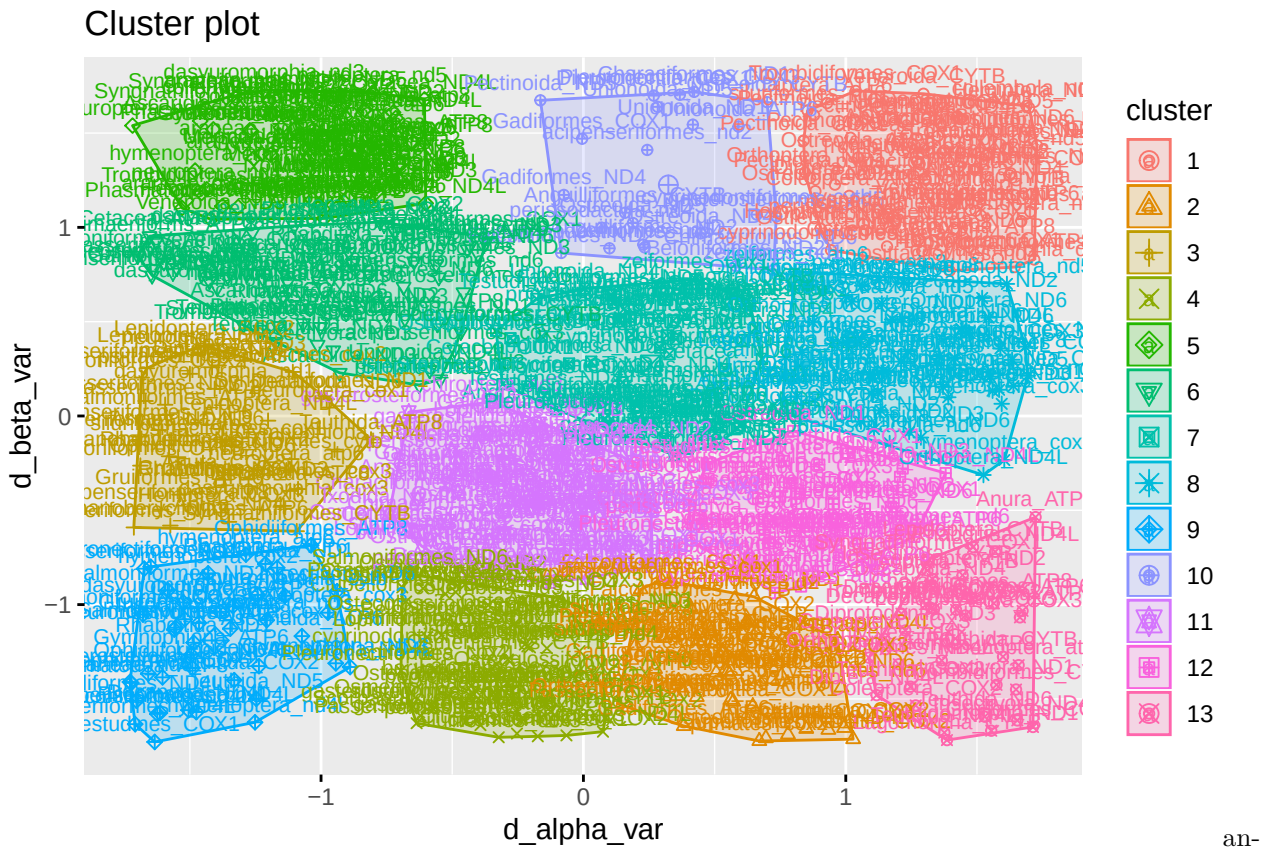
```
t <- sapply(FEL_jsons, JSD.calc)
#t %>% head()
t <- t %>% t()
colnames(t) <- c("file", "dist.temp", "d_alpha_var", "d_beta_var")
t.df<- t %>% as.data.frame()
```

```
t.1 <- t.df %>% select(d_alpha_var, d_beta_var)
t.1$d_alpha_var <- t.1$d_alpha_var %>% as.numeric()
t.1$d_beta_var <- t.1$d_beta_var %>% as.numeric()
cleannames<- t.1 %>% rownames()  %>% str_extract_all(., "\\w+(?=-)", simplify = T)
rownames(t.1) <-paste(cleannames[,1], cleannames[,2], sep = "_")

distance <- get_dist(t.1) #computes distance matrix, default is euclidean
fviz_dist(distance, gradient = list(low = "#00AFBB", mid = "white", high = "#FC4E07"))
```



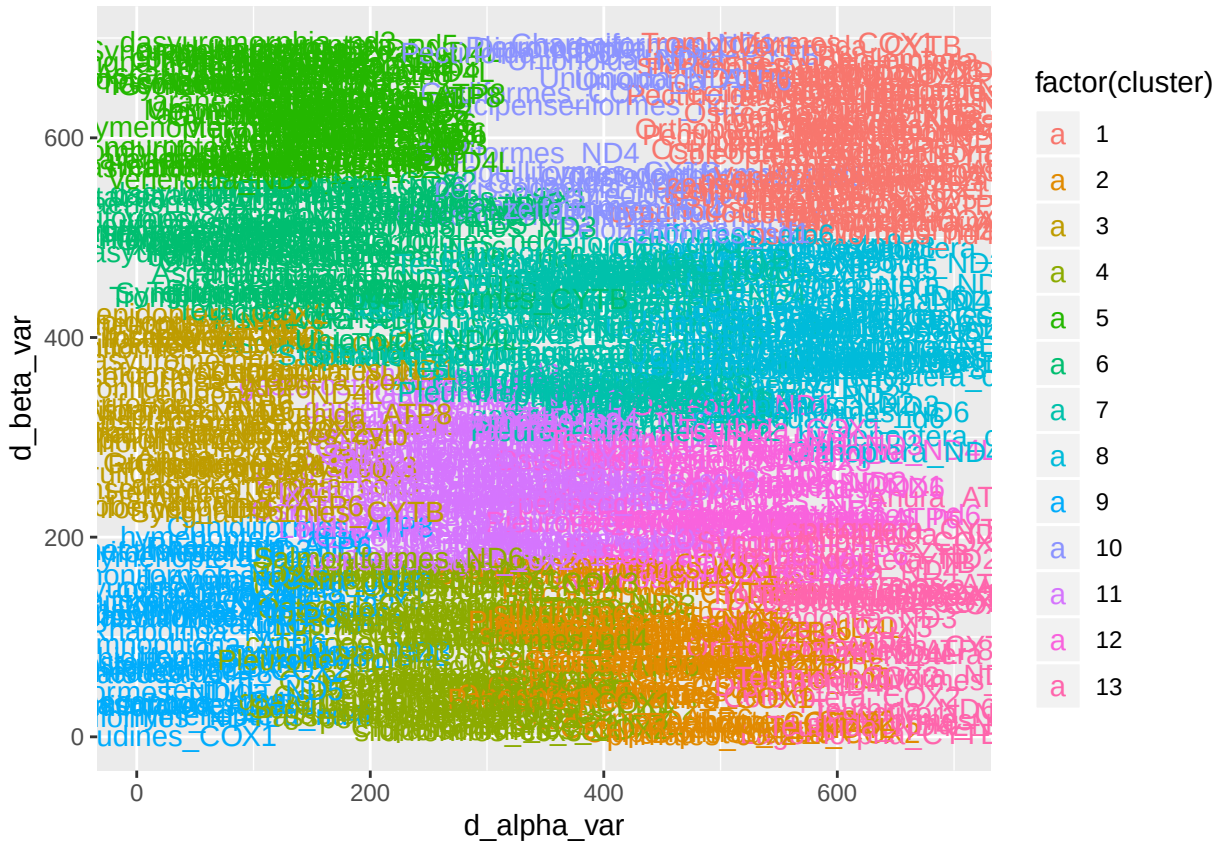

cluster with 13 clusters to see is gene groups emerge

```
k2 <- kmeans(t.1, centers = 13, nstart = 25)
fviz_cluster(k2, data = t.1, labelsize = 8)
```



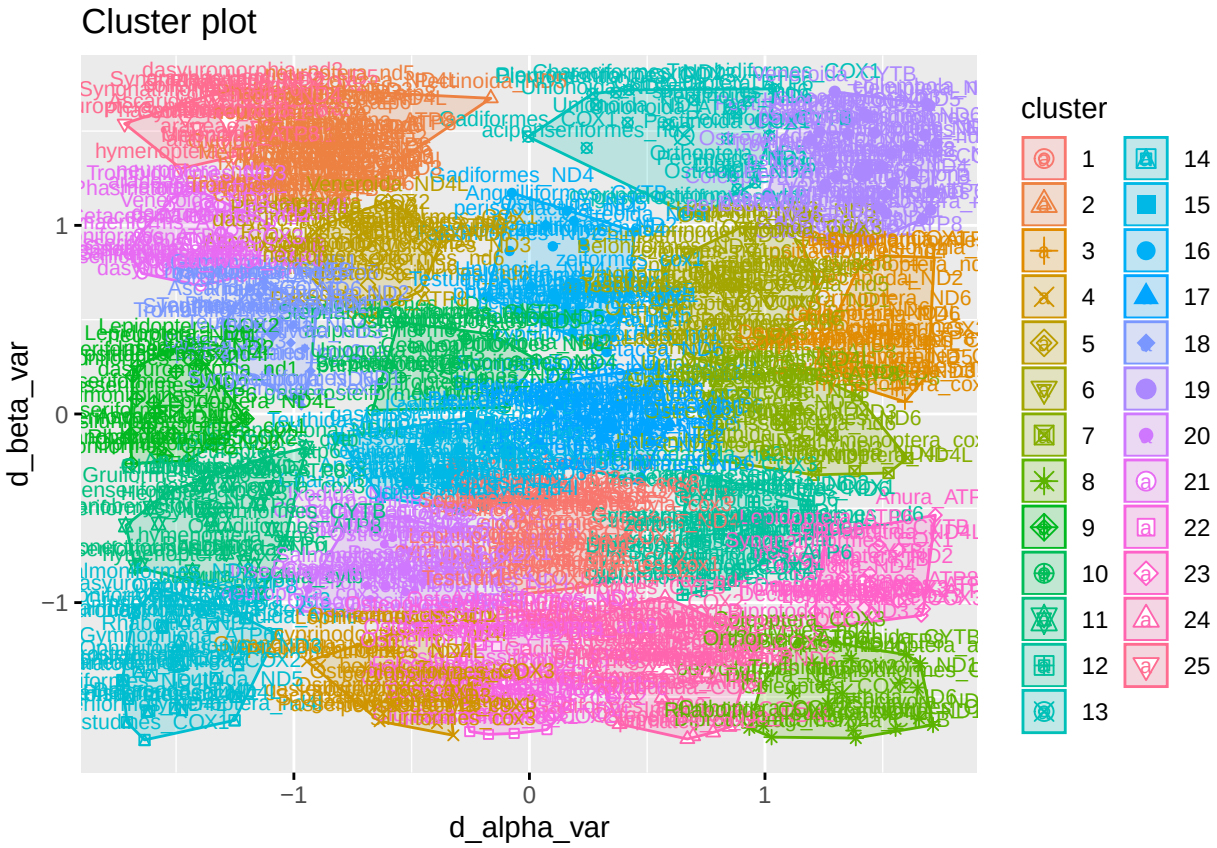

another way of plotting the k means clustering:

```
t.1 %>%
  as_tibble() %>%
  mutate(cluster = k2$cluster,
         gene_order = row.names(t.1)) %>%
  ggplot(aes(d_alpha_var, d_beta_var, color = factor(cluster), label = gene_order)) +
  geom_text()
```




cluster with 25 clusters to see if order clusters emerge

```
k2 <- kmeans(t.1, centers = 25, nstart = 25)
fviz_cluster(k2, data = t.1, labelsize = 8)
```

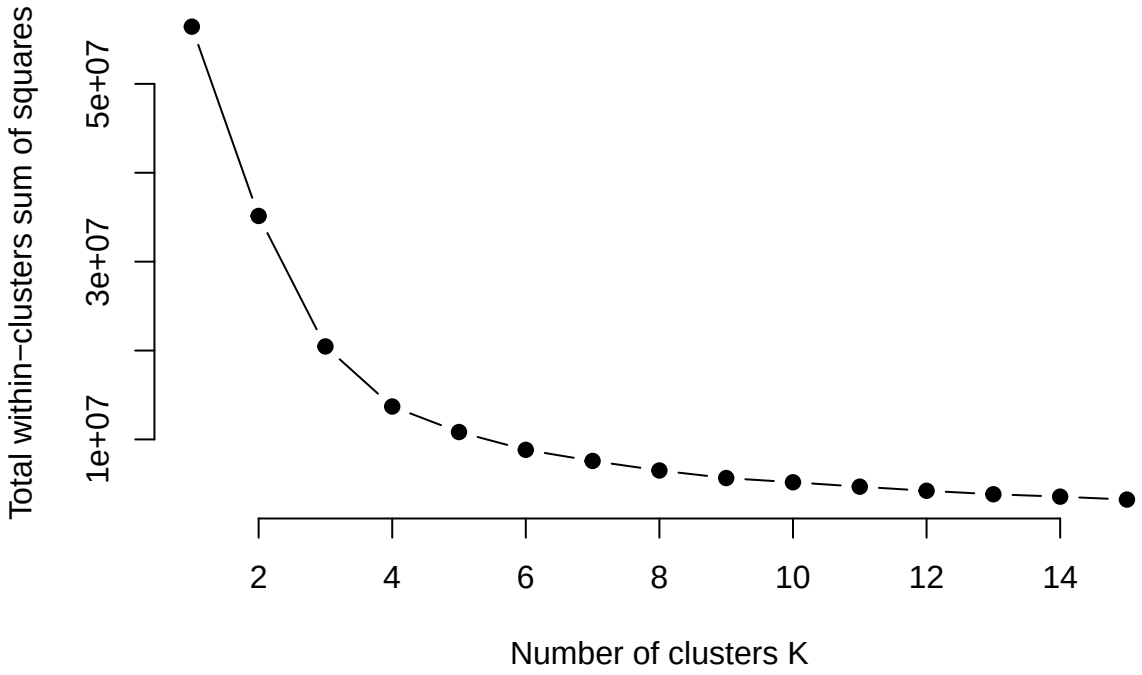



don't notice anything

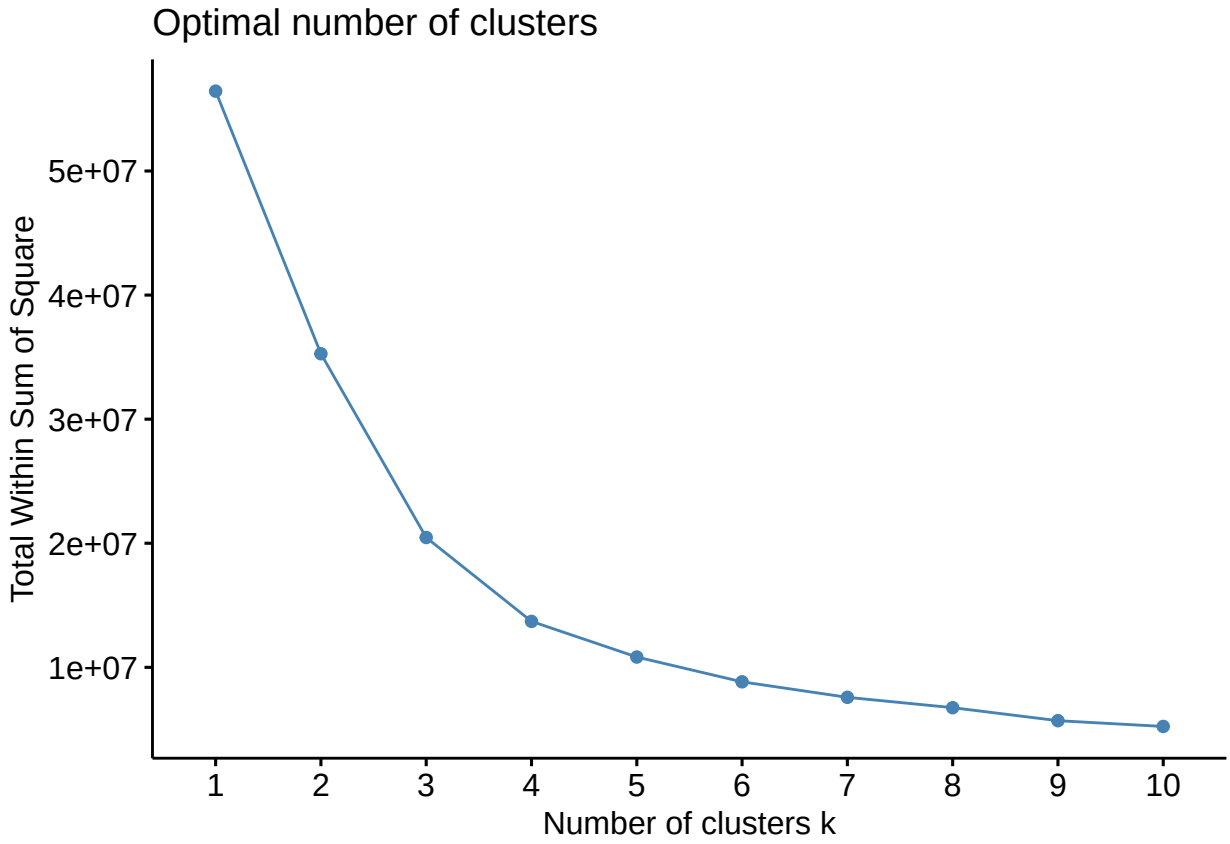
### elbow method of finding clusters

from this tutorial still http://uc-r.github.io/kmeans_clustering

```
set.seed(123)

# function to compute total within-cluster sum of square
wss <- function(k) {
  kmeans(t.1, k, nstart = 10 )$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k.values <- 1:15

# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss)

plot(k.values, wss_values,
       type="b", pch = 19, frame = FALSE,
       xlab="Number of clusters K",
       ylab="Total within-clusters sum of squares")
```

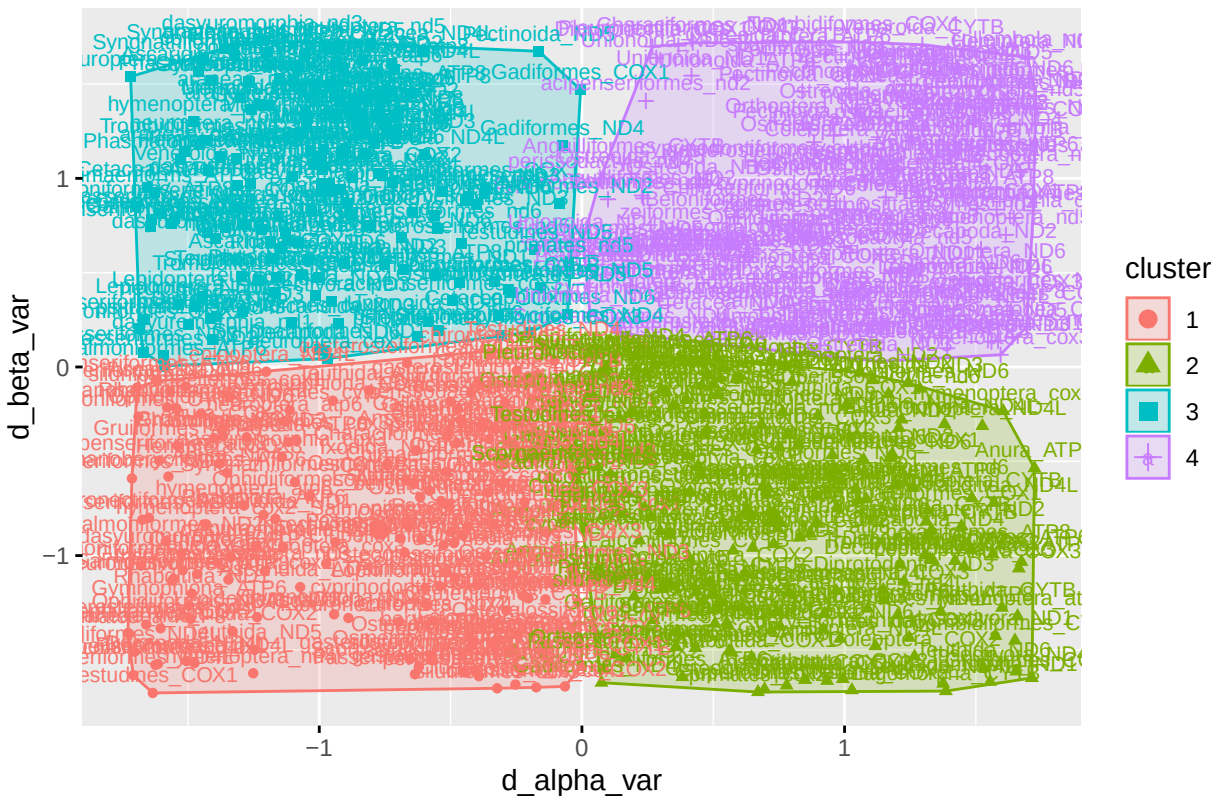



```
##or this way:
fviz_nbclust(t.1, kmeans, method = "wss")
```



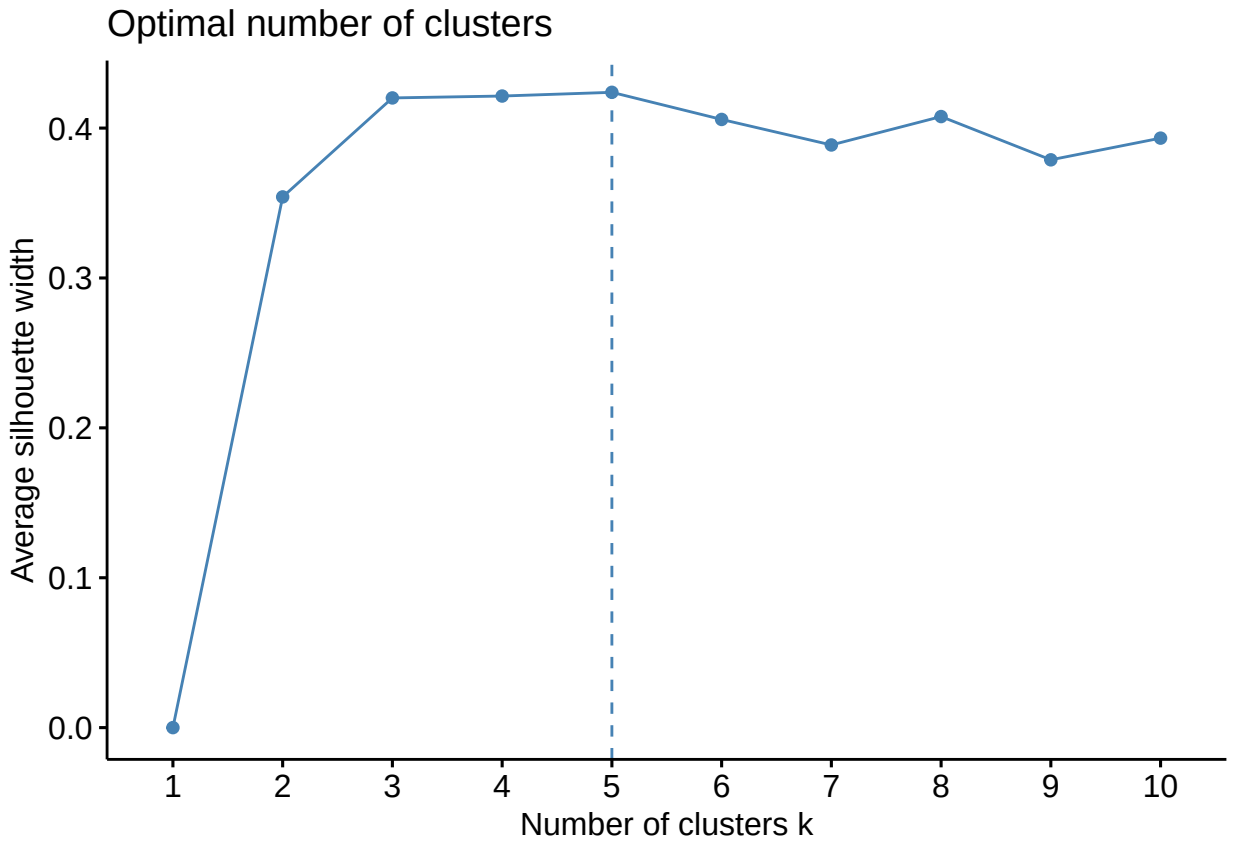

looks like 4 is the best number of clusters by this method:

```
k2 <- kmeans(t.1, centers = 4, nstart = 25)
fviz_cluster(k2, data = t.1, labelsize = 8)
```



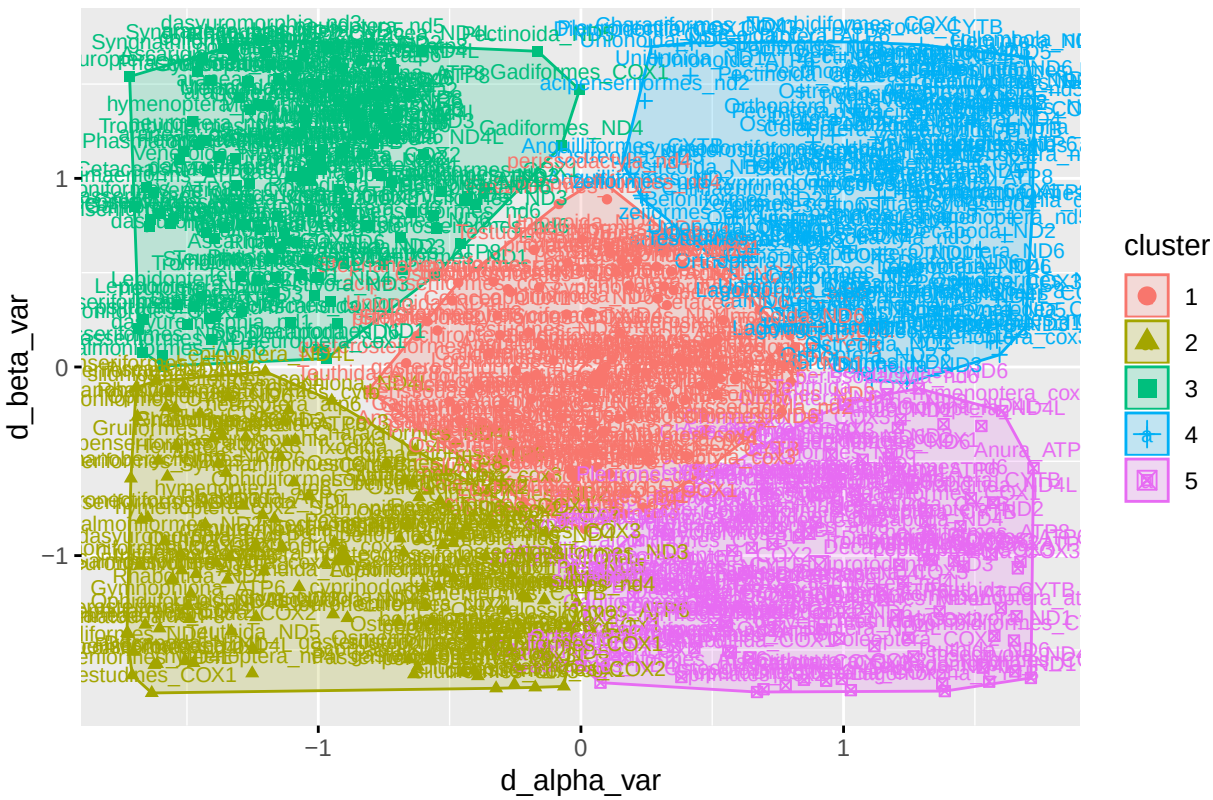

### average silloute method:

```
fviz_nbclust(t.1, kmeans, method = "silhouette")
```




looks like 5 is the best number of clusters by this method:

```
k2 <- kmeans(t.1, centers = 5, nstart = 25)
fviz_cluster(k2, data = t.1, labelsize = 8)
```

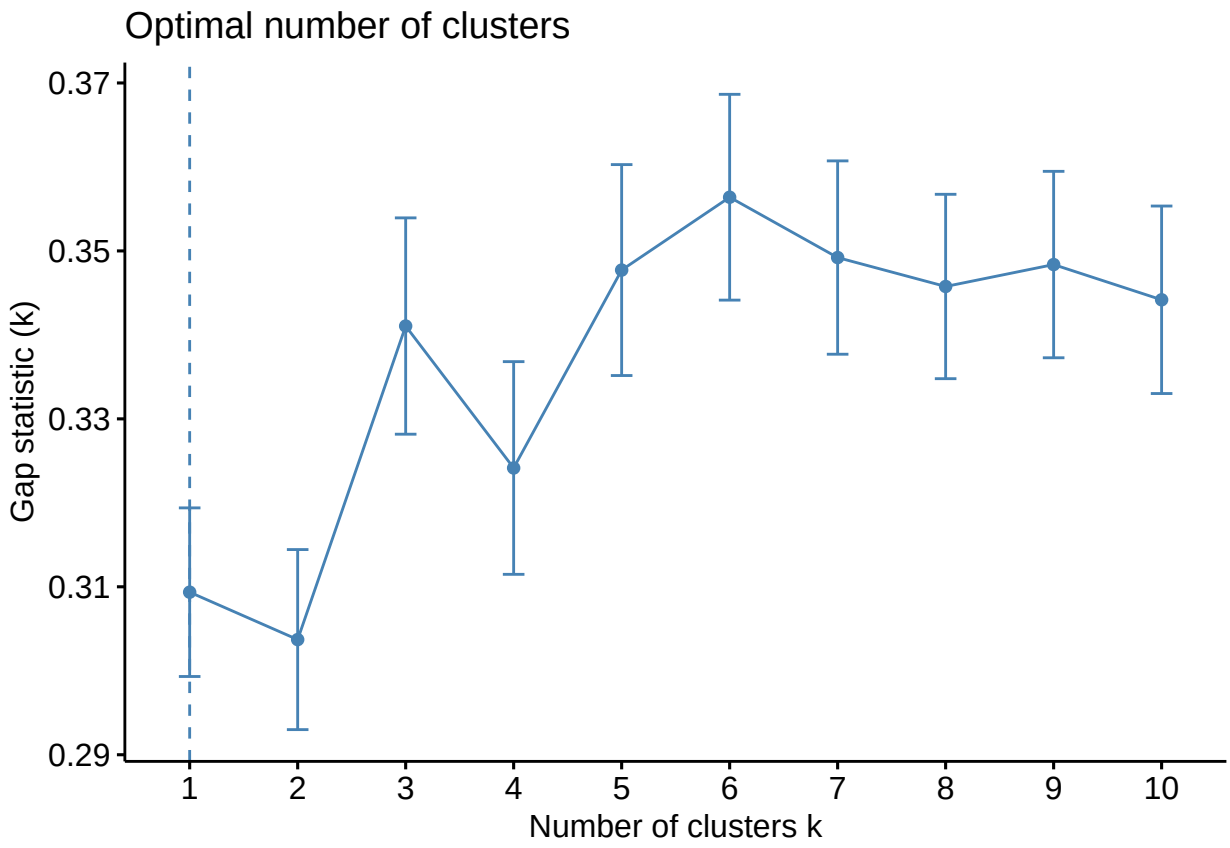



### gap methond

```
set.seed(123)
gap_stat <- clusGap(t.1, FUN = kmeans, nstart = 25,
                    K.max = 10, B = 50)
# Print the result
print(gap_stat, method = "firstmax")
```

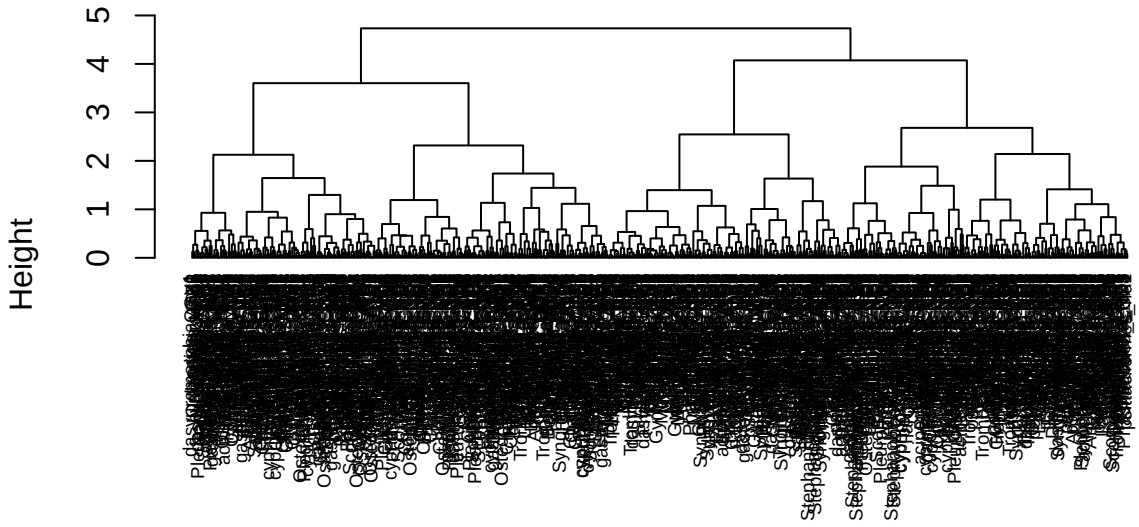
```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = t.1, FUNcluster = kmeans, K.max = 10, B = 50, nstart = 25)
## B=50 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstmax'): 1
##            logW   E.logW       gap     SE.sim
##  [1,] 11.052865 11.36222 0.3093584 0.01003995
##  [2,] 10.789719 11.09343 0.3037126 0.01072197
##  [3,] 10.519531 10.86058 0.3410535 0.01288010
##  [4,] 10.337989 10.66214 0.3241496 0.01266540
##  [5,] 10.214038 10.56176 0.3477198 0.01256155
##  [6,] 10.112792 10.46919 0.3563938 0.01225475
##  [7,] 10.034930 10.38414 0.3492092 0.01151272
##  [8,]  9.960093 10.30586 0.3457644 0.01097864
##  [9,]  9.890794 10.23917 0.3483749 0.01110079
## [10,]  9.839862 10.18404 0.3441801 0.01116492
```

```
fviz_gap_stat(gap_stat)
```




## heirarchical clustering

```
t.1.s <- t.1 %>% scale()
# Dissimilarity matrix
d <- dist(t.1.s, method = "euclidean")

# Hierarchical clustering using Complete Linkage
hc1 <- hclust(d, method = "complete" )

# Plot the obtained dendrogram
plot(hc1, cex = 0.6, hang = -1)
```




use agnes

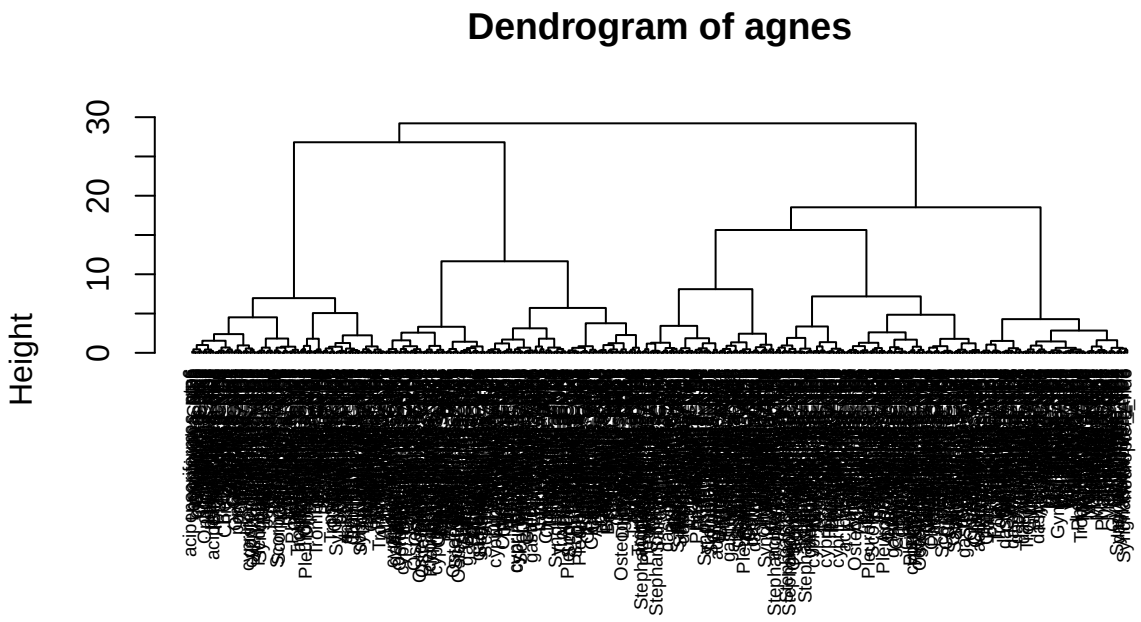
to assess different clustering functions:

```
# methods to assess
m <- c( "average", "single", "complete", "ward")
names(m) <- c( "average", "single", "complete", "ward")

# function to compute coefficient
ac <- function(x) {
  agnes(t.1.s, method = x)$ac
}

map_dbl(m, ac)
```

```
##   average    single  complete      ward
## 0.9663688 0.7478243 0.9837773 0.9973810
```

see that ward is the strongest:

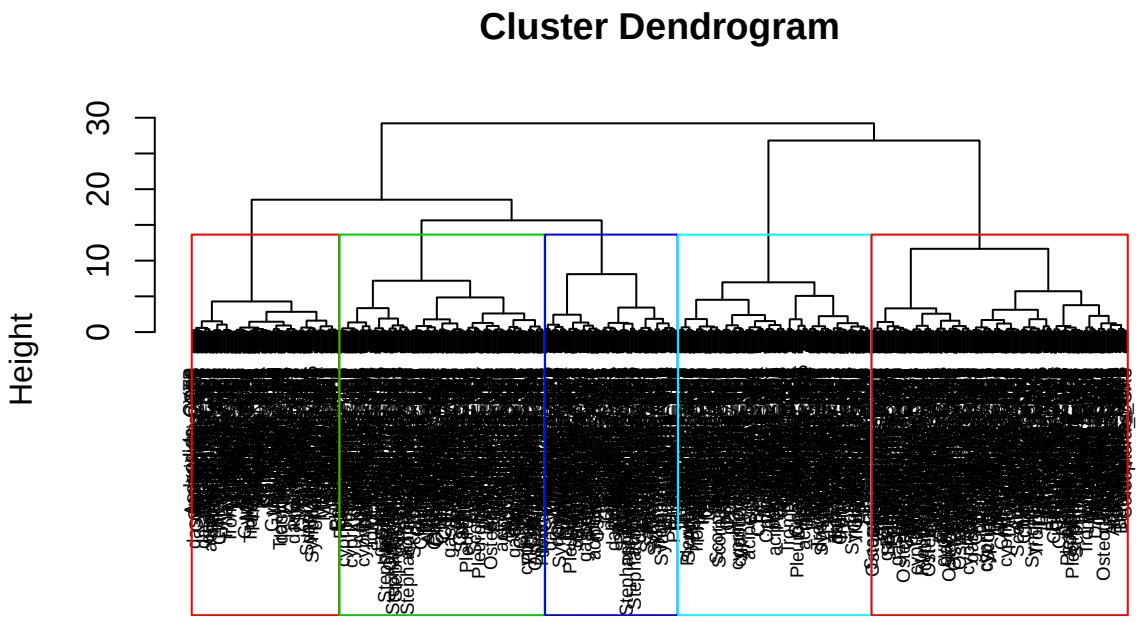
```
hc3 <- agnes(t.1.s, method = "ward")
pltree(hc3, cex = 0.6, hang = -1, main = "Dendrogram of agnes")
```




```
# Ward's method
hc5 <- hclust(d, method = "ward.D2" )

# Cut tree into 4 groups
sub_grp <- cutree(hc5, k = 5)

# Number of members in each cluster
table(sub_grp)
```

```
## sub_grp
##   1   2   3   4   5
## 144  99 153 110 191
```

```
plot(hc5, cex = 0.6)
rect.hclust(hc5, k = 5, border = 2:5)
```




wanna see if I can color the labels by genes: taking a lot from: http://talgalili.github.io/dendextend/articles/dendextend.html#how-to-change-a-dendrogram-1

```
#make a dendrogram
rownames(t.1.s) <- rownames(t.1.s) %>% toupper()
dend15 <- t.1.s %>% dist() %>% hclust(method= "ward.D2")%>% as.dendrogram

 labels(dend15) <- labels(dend15) %>% toupper()

#plot(dend)
invert <- read.delim("~/bin/mtDNA_redo/data/Invert-orders.txt", header = FALSE, sep = "\n", as.is = TRU
invert <- invert$V1 %>% paste(collapse = "|")
is.invert <- ifelse(str_detect(labels(dend15), invert), 2, 3 )
k_5 <- cutree(dend15,k = 5, order_clusters_as_data = FALSE)
```

```
library("colorspace")
#hcl_palettes(plot = TRUE)
```

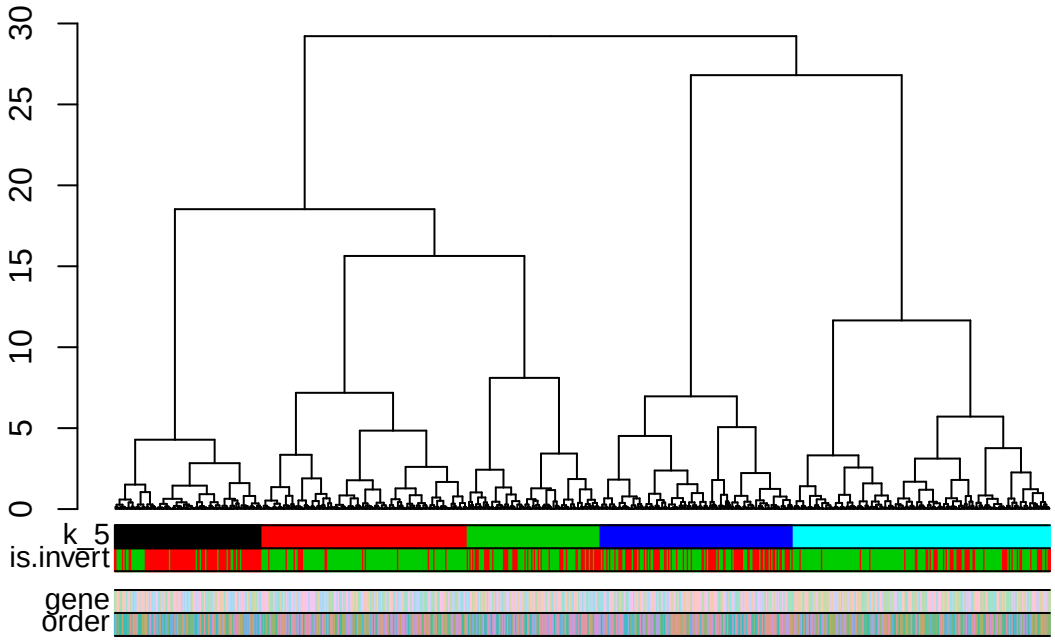
```
clean <- t.1.s %>% rownames() %>% str_split(pattern = "_", simplify= TRUE)

t.1.s<- as.data.frame(t.1.s) %>% mutate(order = as.factor(clean[,1]), gene = as.factor(clean[,2])) %>% r

q4 <- qualitative_hcl(max(t.1.s$order), palette = "Dynamic")
q3 <- qualitative_hcl(max(t.1.s$gene), palette = "Pastel 1")
order <- t.1.s$order %>% mapvalues(.,from = 1:61, to = q4)
gene <- t.1.s$gene%>% mapvalues(.,from = 1:13, to = q3)
```

```
the_bars_1 <- cbind(is.invert, k_5)
the_bars_2 <- cbind( order, gene)
#the_bars[the_bars==2] <- 8
```

```
dend15 %>%set("labels", "")  %>% plot()
```

```
## Warning in `labels<-.dendrogram`(dend, value = value, ...): The lengths of the
## new labels is shorter than the number of leaves in the dendrogram - labels are
## recycled.
```

```
colored_bars(colors = the_bars_1, dend = dend15, y_shift = -1, sort_by_labels_order = FALSE)
colored_bars(colors = the_bars_2, dend = dend15, y_shift = -5)
```




take that clean it up so you have gene, order, alpha/beta JSD.

```
# cleanup <- str_extract_all(names(t), "\\w+(?=-)", simplify = T)
# f <- tibble(order = cleanup[,1], gene = cleanup[,2], JSD_alpha_beta = t)
```

stopped here

```
# get_dist()
# fviz_dist()
```