STAT 302 Fall 2024 - Final Project

Maia Czerwonka

K means clustering

In this project, you will implement an unsupervised learning technique called **k-means clustering**. This method is used to partition a dataset into k distinct clusters based on the similarity of the data points. The goal of k-means is to minimize the variance within each cluster, so that points within the same cluster are as similar as possible.

While there is a built-in function in base R that can perform k-means clustering, you are not allowed to use it in this project. Instead, you will manually implement the algorithm from scratch.

The k-means algorithm works as follows:

- Choose k initial centroids (typically randomly selected points from the dataset).
- Assign each data point to the closest centroid, forming KK clusters.
- Recompute the centroids of the clusters by calculating the mean of all points assigned to each cluster.
- Repeat the assignment and update steps until the centroids no longer change (or change very little), indicating convergence.

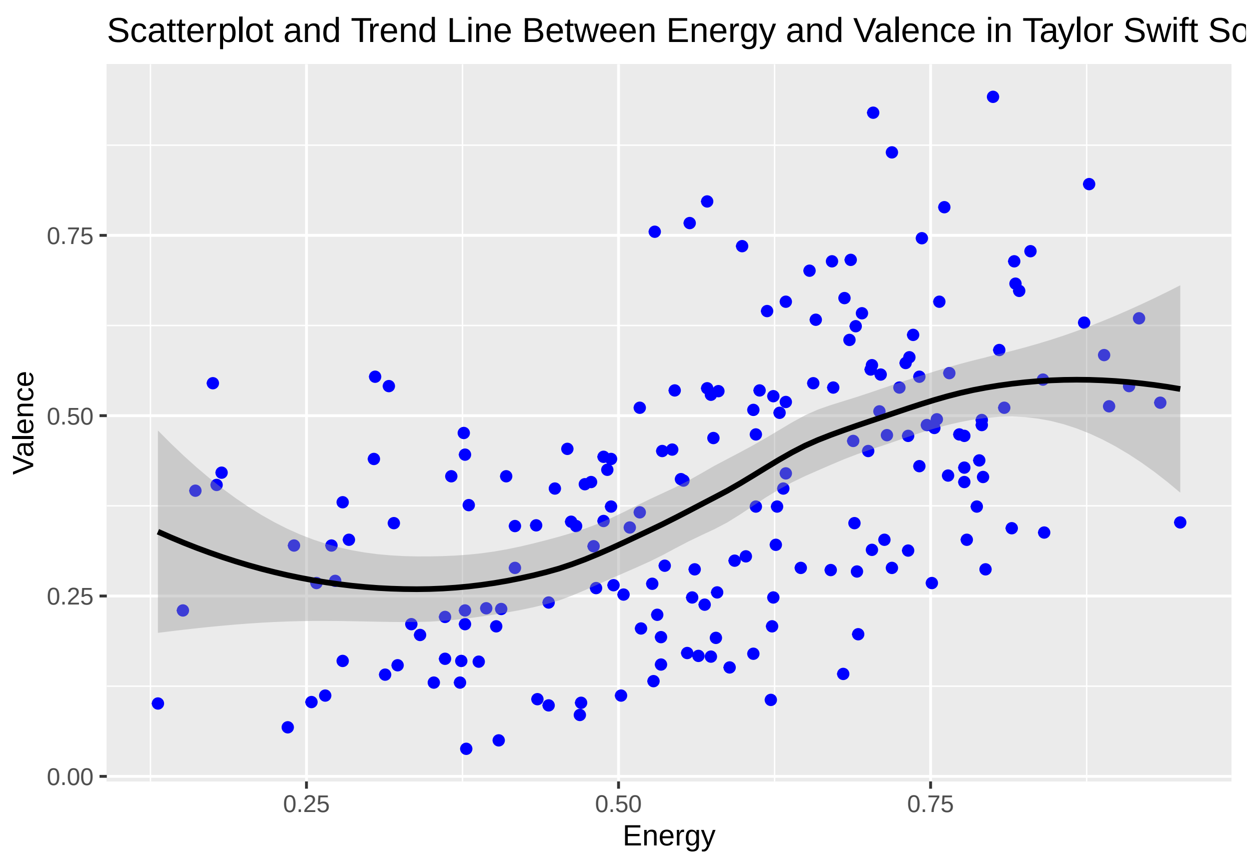
Your task is to replicate this process and implement the algorithm manually using base `R` functions, without relying on any built-in `R` functions (such as the `knn()` function) and without relying on any specialized clustering libraries.

Consider one of the following data set:

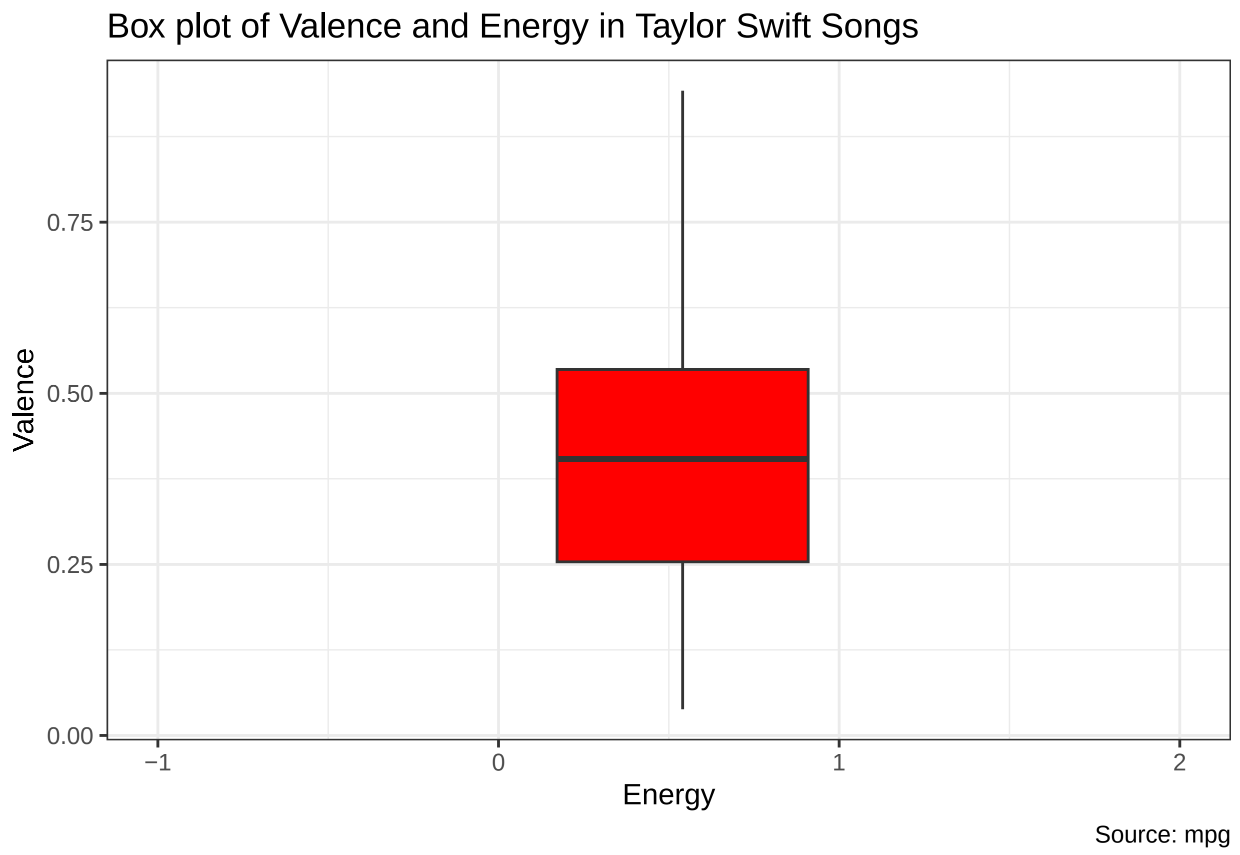
```r
taylor_album_songs <- read.csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/da
songs_reduced <- taylor_album_songs[,c("track_number", "energy", "valence")]
songs_reduced_final <- songs_reduced[rowSums(is.na(songs_reduced))==0,]
```

1. In order to start exploring the dataset, use base `R` or `ggplot` to create some plots (at least 2).

```r
library(ggplot2)
songs<- songs_reduced_final

ggplot(data=songs, mapping=aes(x=energy, y=valence))+
  geom_point(color="blue")+geom_smooth(color="black")+
  labs(title="Scatterplot and Trend Line Between Energy and Valence in Taylor Swift Songs",
       x="Energy", y="Valence")
```



```r
ggplot(data=songs,mapping=aes(x=energy, y=valence))+geom_boxplot(fill="red")+
  theme(axis.text.x = element_text(vjust=0.6)) +
  labs(title="Box plot of Valence and Energy in Taylor Swift Songs",
       caption="Source: mpg",
       x="Energy",
       y="Valence")+ xlim(-1,2)+ theme_bw()
```



2. Randomly choose any three points from the dataset which will be your initial centroids. These will define the centers of the first initial three clusters. This means that you are defining $k = 3$ clusters.

```
songs <- songs[!duplicated(songs$track_number), ]
set.seed(123)
k=3
three_points<- sample(songs$track_number, size=k)
selected_songs<- songs[c(5,8,14),]
```

3. Measure the distance between all the points in the dataset and each of the three initial clusters. Assign the points to the nearest cluster. Do this in the most efficient way.

```
track_num <- c()
centroid <- c()
pt_dist <- c()

unique_tracks <- unique(songs$track_number)

for (track in unique_tracks) {
  min_dist <- Inf
  min_index <- NA

  track_values <- as.numeric(unlist(songs[songs$track_number == track, c("energy", "valence")]))
  for (point in 1:nrow(selected_songs)) {
    current_pt <- as.numeric(selected_songs[point, c("energy", "valence")])
    centroid_dist <- sqrt(sum((track_values - current_pt)^2))

    if (centroid_dist < min_dist) {
      min_dist <- centroid_dist
      min_index <- selected_songs[point, "track_number"]
    }
  }

  track_num <- c(track_num, track)
  centroid <- c(centroid, min_index)
  pt_dist <- c(pt_dist, min_dist)
}

results <- data.frame(Track = track_num, Centroid = centroid,
                      Distance = pt_dist,songs[, c("energy", "valence")])
head(results)
```

```
##   Track Centroid   Distance energy valence
## 1     1        8 0.15901258  0.491   0.425
## 2     2        8 0.40248354  0.877   0.821
## 3     3        5 0.07077429  0.417   0.289
## 4     4       14 0.16209874  0.777   0.428
## 5     5        5 0.00000000  0.482   0.261
## 6     6        8 0.19632881  0.805   0.591
```

4. After assigning all points to the closest centroid, recalculate the centroids. This is achieved by computing the mean of all the variables of the points assigned to each cluster. This will give you the new centroids.

```
centroid_recalc<- function(df){
  centroid_name<- c(1,2,3)
  energy_mean<- c()
  valence_mean<-c()
  for (centroid in (unique(df$Centroid))){
    c<- df[df$Centroid==centroid,]
    c_energy<-mean(df[df$Centroid==centroid, "energy"])
    c_valence<-mean(df[df$Centroid==centroid, "valence"])
    energy_mean<- c(energy_mean, c_energy)
    valence_mean<-c(valence_mean, c_valence)
  }
  centroid_results<- data.frame(Centroid=centroid_name, Energy=energy_mean, Valence=valence_mean)
  print(centroid_results)
}

centroid_results<- centroid_recalc(results)
```

```
##   Centroid    Energy   Valence
## 1        1 0.6914118 0.5281765
## 2        2 0.4627000 0.2324000
## 3        3 0.7690000 0.3413333
```

5. Repeat the process of assigning points to clusters and updating the centroids until the centroids no longer change.

```r
tracks <- results[, !(names(results) %in% c("Distance", "Centroid"))]
repeat{
  track_num <- c()
  centroid <- c()
  pt_dist <- c()

  for (track in tracks$Track) {
    min_dist <- Inf
    min_index <- NA

    track_values <- as.numeric(unlist(tracks[tracks$Track == track, c("energy", "valence")]))

    for (point in 1:nrow(centroid_results)) {
      current_pt <- as.numeric(centroid_results[point, c("Energy", "Valence")])
      centroid_dist <- sqrt(sum((track_values - current_pt)^2))

      if (centroid_dist < min_dist) {
        min_dist <- centroid_dist
        min_index <- centroid_results[point, "Centroid"]
     }
    }

    track_num <- c(track_num, track)
    centroid <- c(centroid, min_index)
    pt_dist <- c(pt_dist, min_dist)
  }

  results <- data.frame(Track = track_num, Centroid = centroid,
                        Distance = pt_dist,tracks[, c("energy", "valence")])

  new_centroid_results<- centroid_recalc(results)

  if (identical(centroid_results, new_centroid_results)){
    break
  }
  centroid_results<- new_centroid_results
}
print(head(results))
print(centroid_results)
```

6. Calculate the Within-Cluster-Sum of Squared Errors (WSS) for different values of k (including the $k = 3$). The WSS is given by:

$$\text{WSS} = \sum_{k=1}^{K} \sum_{i=1}^{n_k} \left(\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^\top \left(\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k\right)$$

where:

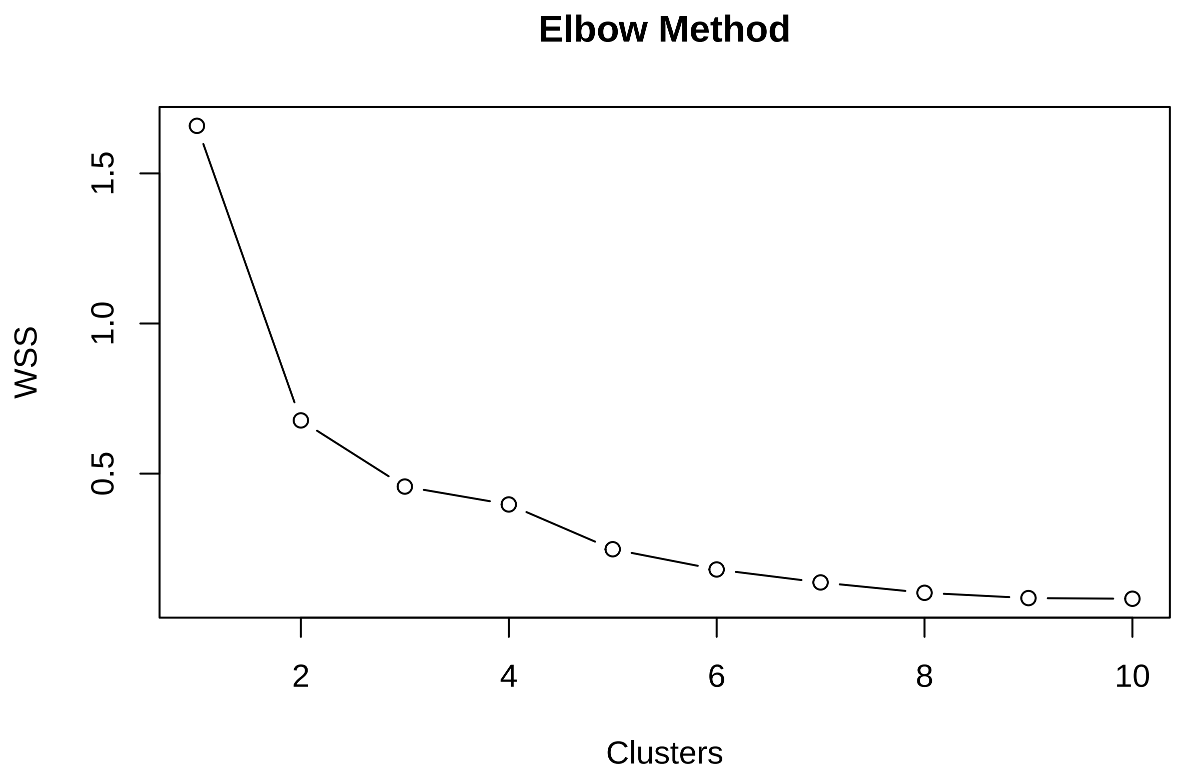
- $\mathbf{x}_i^{(k)} = (x_{i1}^{(k)}, x_{i2}^{(k)}, \ldots, x_{id}^{(k)})$ is the i-th data point in cluster k, where d is the number of dimensions (features).

- $\boldsymbol{\mu}_k = (\mu_{k1}, \mu_{k2}, \ldots, \mu_{kd})$ is the centroid (mean) of cluster k, calculated as the average of the points in the cluster.
- The term $\left(\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^\top \left(\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k\right)$ represents the squared Euclidean distance between the point $\mathbf{x}_i^{(k)}$ and its corresponding centroid $\boldsymbol{\mu}_k$.

```r
k_means <- function(data, k, seed) {
  set.seed(seed)

  initial <- sample(1:nrow(data), size = k)
  centroids <- data[initial, c("energy", "valence")]
  centroids <- data.frame(Centroid = 1:k, centroids)

  repeat {
    assignments <- data.frame(Track = data$track_number, energy = data$energy,
                              valence = data$valence)
    assignments$Centroid <- rep(NA, nrow(data))
    assignments$Distance <- rep(Inf, nrow(data))

    for (i in 1:nrow(data)) {
      point <- as.numeric(data[i, c("energy", "valence")])
      min_dist <- Inf
      closest_centroid <- NA

      for (j in 1:nrow(centroids)) {
        centroid <- as.numeric(centroids[j, c("energy", "valence")])
        dist <- sum((point - centroid)^2)

        if (dist < min_dist) {
          min_dist <- dist
          closest_centroid <- centroids[j, "Centroid"]
        }
      }
      assignments$Centroid[i] <- closest_centroid
      assignments$Distance[i] <- min_dist
    }

    new_centroids <- data.frame(Centroid = 1:k, energy = numeric(k),
                                valence = numeric(k), count = numeric(k))

    for (j in 1:k) {
      cluster_points <- assignments[assignments$Centroid == j, c("energy", "valence")]
      if (nrow(cluster_points) > 0) {
        new_centroids$energy[j] <- sum(cluster_points$energy) / nrow(cluster_points)
        new_centroids$valence[j] <- sum(cluster_points$valence) / nrow(cluster_points)
        new_centroids$count[j] <- nrow(cluster_points)
      }
    }

    centroids_changed <- FALSE
    for (j in 1:k) {
      for (dim in c("energy", "valence")) {
        if (abs(new_centroids[j, dim] - centroids[j, dim]) > 0.000001) {
          centroids_changed <- TRUE
          break
        }
      }
      if (centroids_changed) {
        break
      }
    }

    if (!centroids_changed) {
      break
    }

    centroids <- new_centroids[, c("Centroid", "energy", "valence")]
  }

  wss <- sum(assignments$Distance)
  assignments<- assignments$Centroid
  assgn_wss_df<- data.frame(songs$track_number, assignments,wss)
  print(assgn_wss_df)
  return(assgn_wss_df)
}

wss_list <- c()
assignments<-c()
k_values <- 1:10
for (k in k_values) {
  wss <- k_means(songs, k, 123)
  wss_list <- c(wss_list, unique(wss$wss))
  assignments<- c(assignments,wss)
}
```

7. Using either base R or ggplot2, plot WSS-versus-k. Choose the k for which WSS becomes first starts to diminish. In the plot, this is visible as an elbow.

```
plot(k_values, wss_list, type = "b", xlab = "Clusters", ylab = "WSS", main = "Elbow Method")
```



The k for which the plots first starts to diminish is k=2.

8. Since this procedure started with a random seed, consider 10 different random seeds for each value of k.

```
wss_list=c()
assignments_data<-data.frame(row.names=songs$track_number)
k_list<- 1:6
random_seeds<- sample(1:10000, size=10, replace=FALSE)
for (k in k_list){
  for (seed in random_seeds){
    value<-k_means(songs,k,seed)
    wss_list <- c(wss_list, unique(value$wss))
    assignments_data<- cbind(assignments_data, value$assignments)
  }
}
print(assignments_data)
print(wss_list)
```

9. For each data point, compute the proportion of times the data point ends up in the same cluster. Are there any points which are not always assigned to the same group? If so, try to explain why you think that happens.

```
proportions_calc<- function(row){
  unique_values<- unique(row)
  max_count<-0

  for (val in unique_values){
    count<- sum( row==val)
    if (count> max_count){
      max_count<-count
    }
  }
  return(max_count/length(row))
}

proportions<- apply(assignments_data, 1, proportions_calc)
```

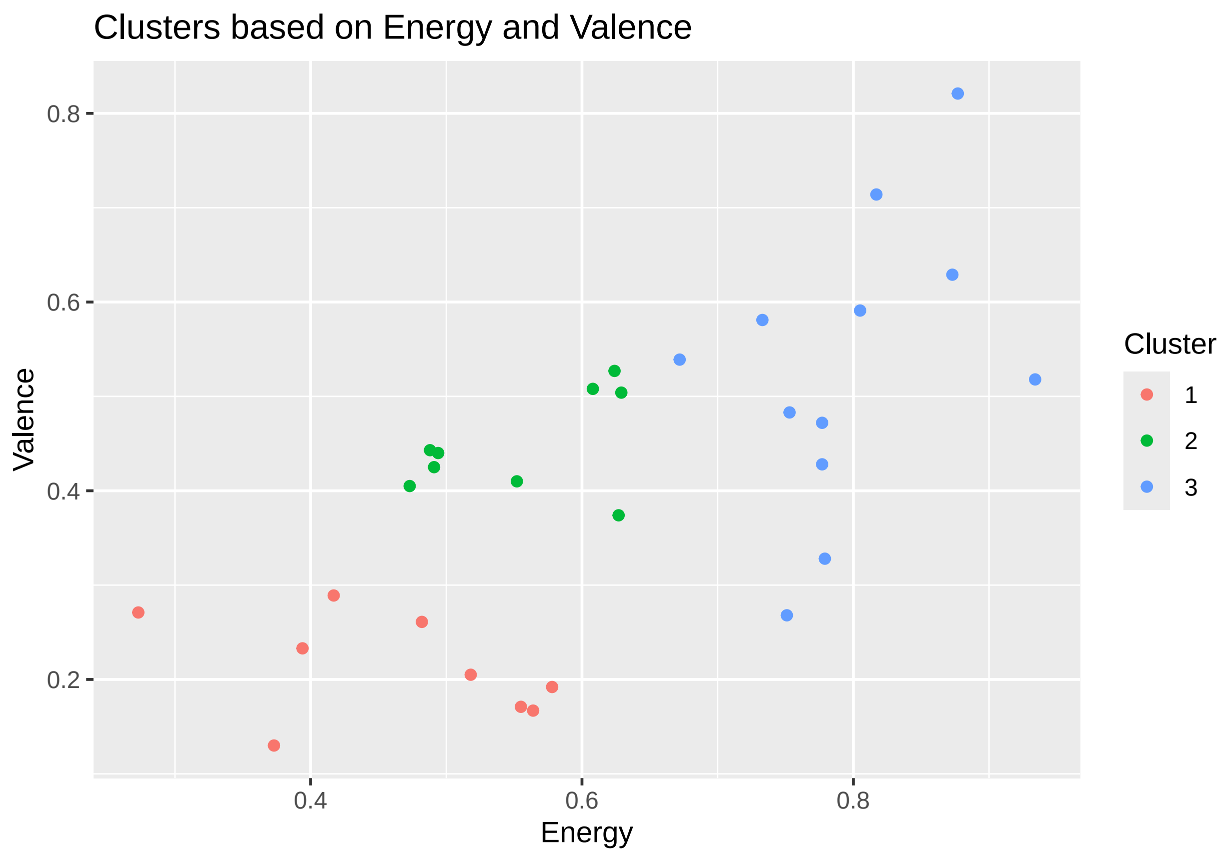
None of my points are always assigned to the same cluster. They all have proportions of being assigned to their most frequent cluster about 40% of the time. I think this is due to the variable nature of initial point sampling and then the increasing of the number of initial points during each iteration.

10. Choose any of the seeds used above and consider your clustering as final. Add a variable called `cluster` to the data set. Use the variables in the data set (both the ones used for clustering and the ones not used for clustering) to visually describe the clusters. Explain what differences you observe.

```
final_clustering<-k_means(songs,3,2758)
```

```
##    songs.track_number assignments       wss
## 1                   1           2 0.4856567
## 2                   2           3 0.4856567
## 3                   3           1 0.4856567
## 4                   4           3 0.4856567
## 5                   5           1 0.4856567
## 6                   6           3 0.4856567
## 7                   7           1 0.4856567
## 8                   8           2 0.4856567
## 9                   9           3 0.4856567
## 10                 10           2 0.4856567
## 11                 11           3 0.4856567
## 12                 12           3 0.4856567
## 13                 13           1 0.4856567
## 14                 14           3 0.4856567
## 15                 15           3 0.4856567
## 16                 16           1 0.4856567
## 17                 17           1 0.4856567
## 18                 18           1 0.4856567
## 19                 19           3 0.4856567
## 20                 20           3 0.4856567
## 21                 21           2 0.4856567
## 22                 22           3 0.4856567
## 23                 23           1 0.4856567
## 24                 24           2 0.4856567
## 25                 25           2 0.4856567
## 26                 26           2 0.4856567
## 27                 27           2 0.4856567
## 28                 28           2 0.4856567
## 29                  29           3 0.4856567
## 30                  30           1 0.4856567
```

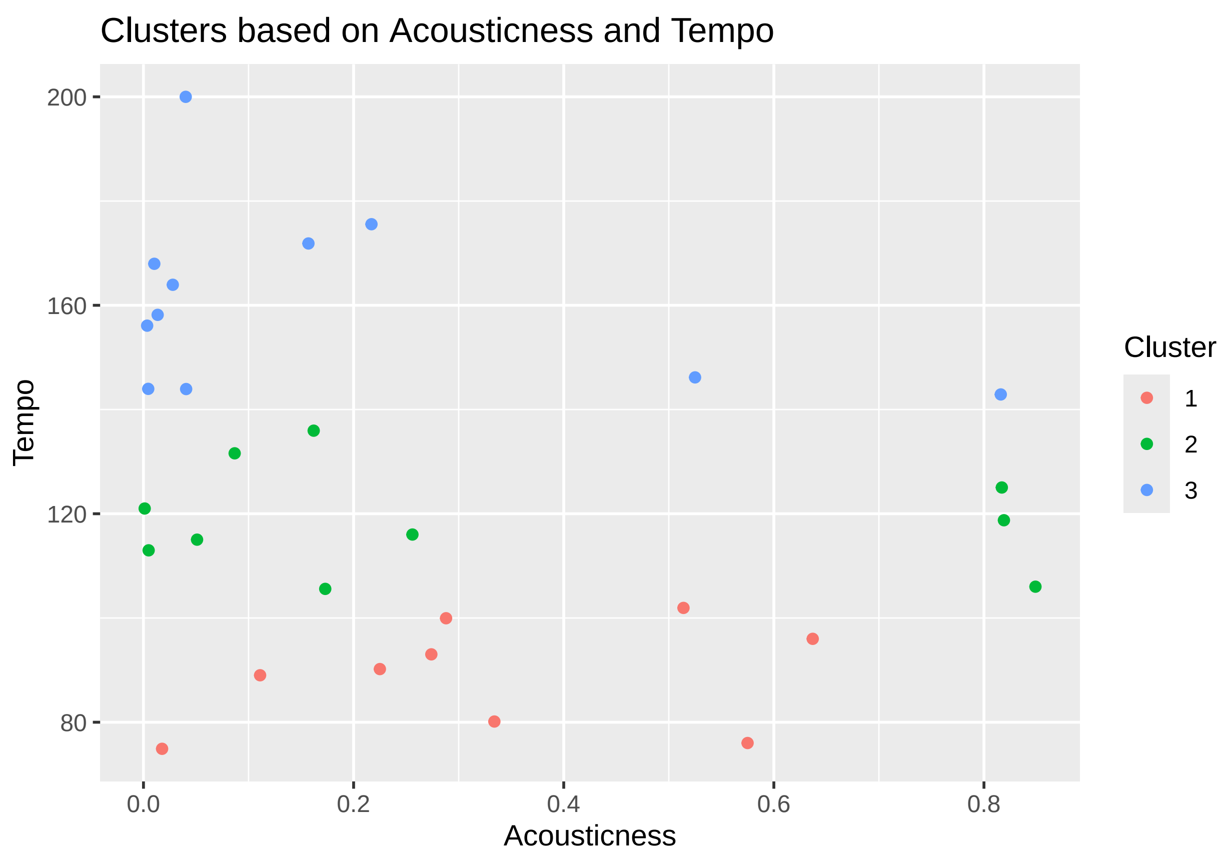
```
final<-cbind(final_clustering, songs[,c("energy", "valence")])

ggplot(songs, aes(x = energy, y = valence, color =factor(as.numeric(unlist(final$assignments))))) +
  geom_point() +
  labs(
    title = "Clusters based on Energy and Valence",
    x = "Energy",
    y = "Valence",
    color = "Cluster"
  )
```



```
songs_reduced <- taylor_album_songs[,c("track_number", "acousticness", "tempo")]
songs_reduced_final <- songs_reduced[rowSums(is.na(songs_reduced))==0,]
songs2 <- songs_reduced_final[!duplicated(songs_reduced_final$track_number), ]
colnames(songs2)[colnames(songs2)=="acousticness"]<-"energy"
colnames(songs2)[colnames(songs2)=="tempo"]<-"valence"
new_characteristics<-k_means(songs2,3,2758)
```

```
##    songs.track_number assignments      wss
## 1                   1           1 4767.368
## 2                   2           2 4767.368
## 3                   3           1 4767.368
## 4                    4           2 4767.368
## 5                    5           3 4767.368
## 6                    6           2 4767.368
## 7                    7           3 4767.368
## 8                    8           2 4767.368
## 9                    9           3 4767.368
## 10                  10           1 4767.368
## 11                  11           1 4767.368
## 12                  12           3 4767.368
## 13                  13           1 4767.368
## 14                  14           3 4767.368
## 15                  15           3 4767.368
## 16                  16           2 4767.368
## 17                  17           3 4767.368
## 18                  18           3 4767.368
## 19                  19           3 4767.368
## 20                  20           3 4767.368
## 21                  21           3 4767.368
## 22                  22           2 4767.368
## 23                  23           2 4767.368
## 24                  24           1 4767.368
## 25                  25           1 4767.368
## 26                  26           1 4767.368
## 27                  27           2 4767.368
## 28                  28           2 4767.368
## 29                  29           2 4767.368
## 30                  30           1 4767.368
```

```r
songs2 <- songs_reduced_final[!duplicated(songs_reduced_final$track_number), ]
new_characteristics<- cbind(new_characteristics, songs2[, c("acousticness", "tempo")])

ggplot(new_characteristics, aes(x = acousticness, y = tempo,
                                color =factor(as.numeric(unlist(new_characteristics$assignments))))) +
  geom_point() +
  labs(
    title = "Clusters based on Acousticness and Tempo",
    x = "Acousticness",
    y = "Tempo",
    color = "Cluster"
  )
```



The difference between these two graphs is quite distinct. The valence and energy graph has clusters that are very near to each other and they make sense. points near each other are in very clear clusters. The acousticness and tempo graph is much different from this. The clusters are very dispersed and not very cohesive. It feels like they did not cluster properly like the valence and energy clusters did.