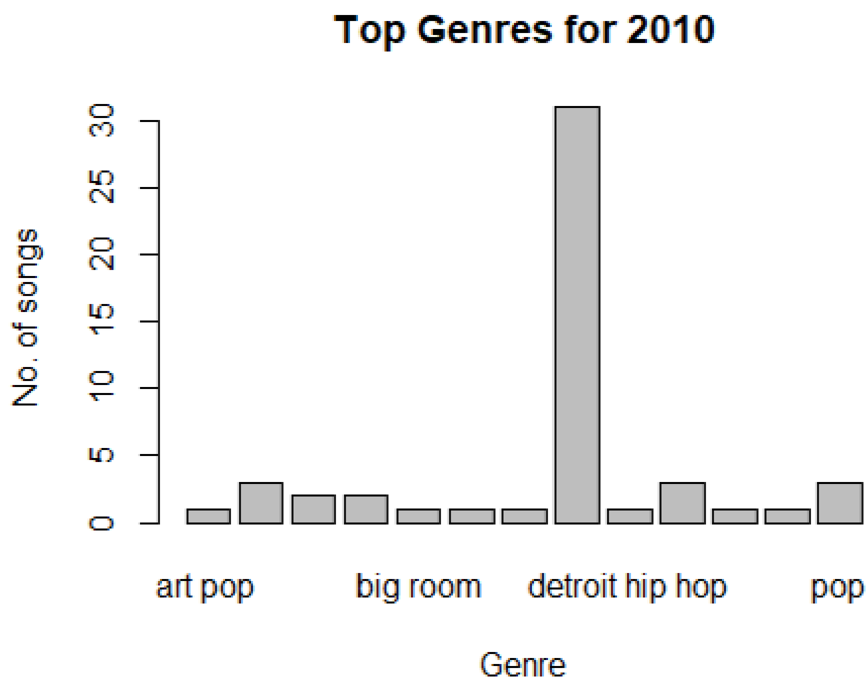
cleaning.R

Apurva Sarode

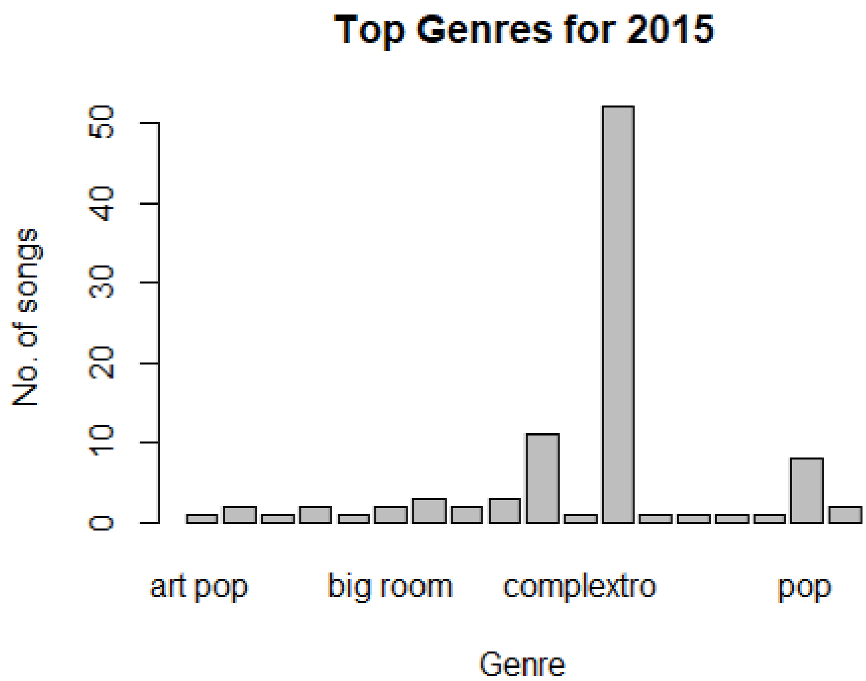
2020-03-05

```
#Top Songs Analysis

#importing dataset top10s and copying it to test data
data <- read.csv'C:\\Users\\Apurva Sarode\\Desktop\\Spotify_mva.csv')
View(data)

#Data Cleaning

#Adding column Rank which will denote rank of a song based on it popularity.
# popularity from 90 - 100 is Rank 10 and so
on for(x in 1:length(data$pop)){
  if(data[x,15] <= 100 && data[x,15] >= 80){
    data[x,16] = 5
  }else if(data[x,15] < 80 && data[x,15] >=
    60){ data[x,16] = 4
  }else if(data[x,15] < 60 && data[x,15] >=
    40){ data[x,16] = 3
  }else if(data[x,15] < 40 && data[x,15] >=
    20){ data[x,16] = 2
  }else if(data[x,15] < 20 && data[x,15] >=
    0){ data[x,16] = 1
  }
}
data$pop <- NULL
dim(data)

## [1] 603  15

#removing values with 0 BPM and duration as 0 seconds

data_clean <- data[-c(433),]
names(data_clean)[15]<- "rating"

View(data_clean)

#EDA
#checking the ranges for all columns
dim(data_clean)

## [1] 602  15
```

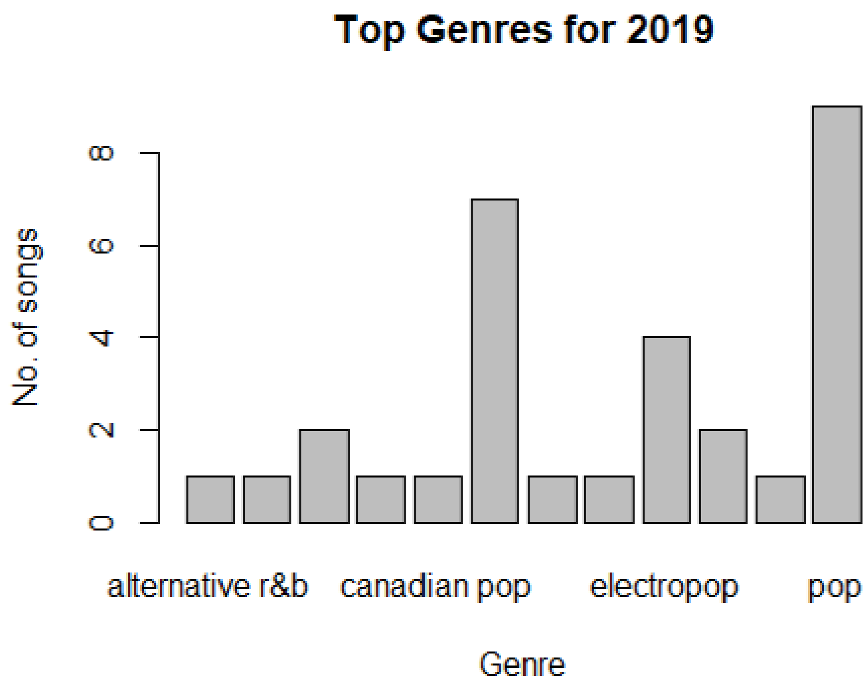
```
library(plyr)
library(ggplot2)

#Finding top genre for 3 years
year1 = data_clean[data_clean$year ==
2010,] gen1 = count(year1$top.genre)
barplot(gen1$freq, names.arg = gen1$x,main = 'Top Genres for 2010',xlab = 'Ge
nre',ylab = 'No. of songs')
```



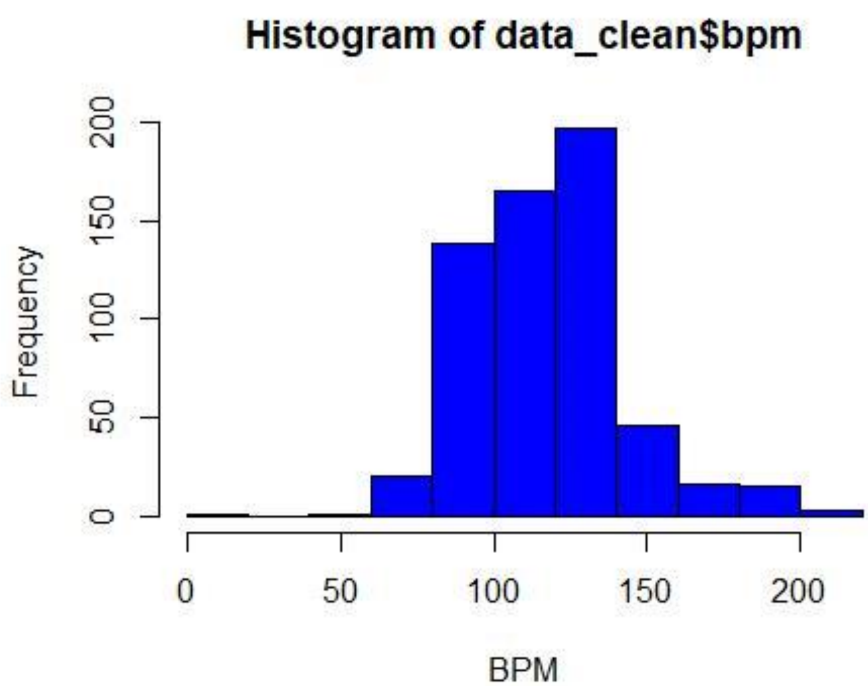

```
year2 = data_clean[data_clean$year ==
2015,] gen2 = count(year2$top.genre)
barplot(gen2$freq, names.arg = gen2$x,main = 'Top Genres for 2015',xlab = 'Ge
nre',ylab = 'No. of songs')
```



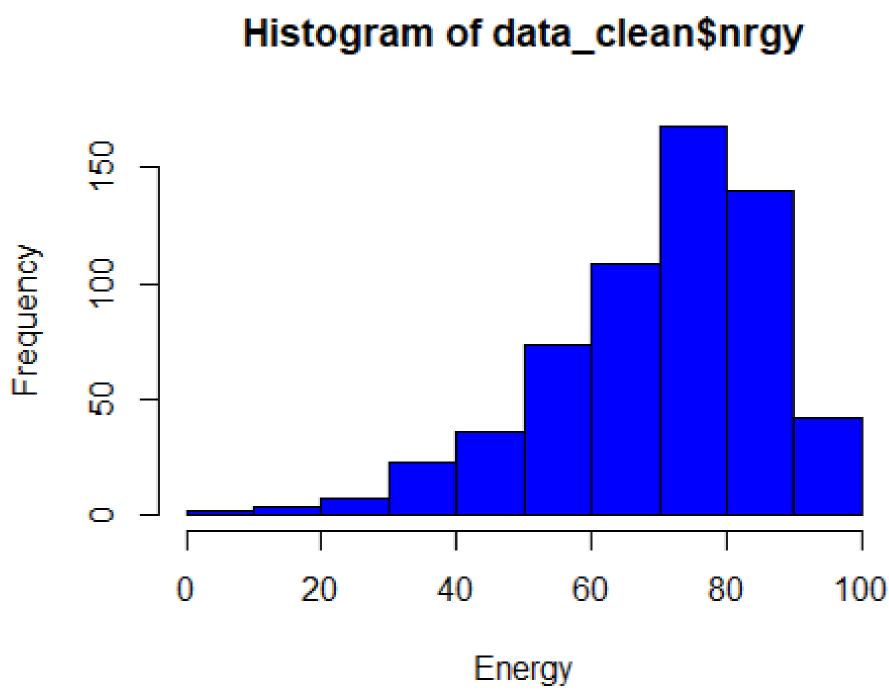

```
year3 = data_clean[data_clean$year ==
2019,] gen3 = count(year3$top.genre)
barplot(gen3$freq, names.arg = gen3$x,main = 'Top Genres for 2019',xlab = 'Ge
nre',ylab = 'No. of songs')
```



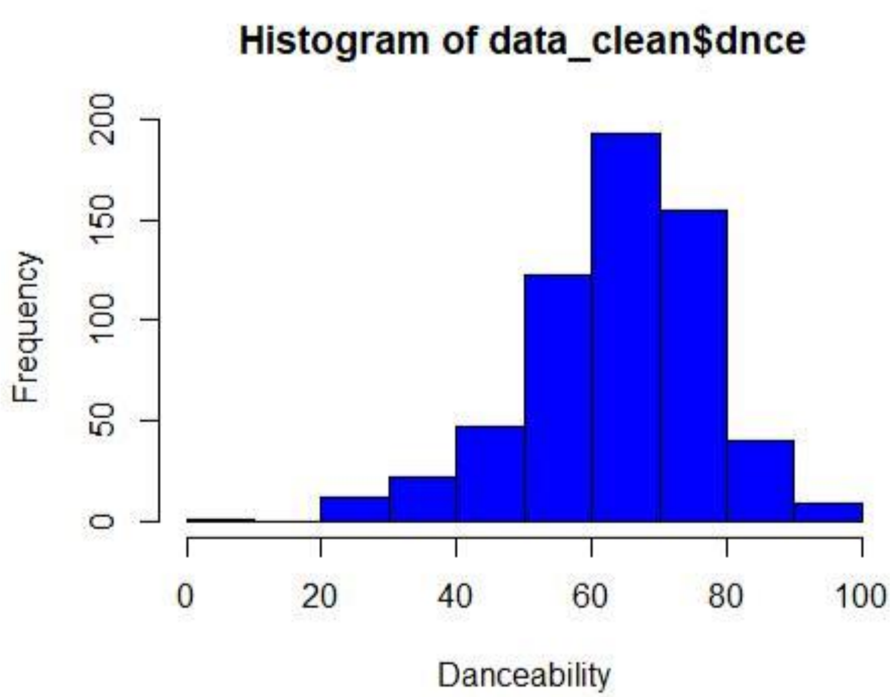

```r
#Histogram view of audio properties
hist(data_clean$bpm, breaks=12,col="blue",xlab="BPM")
```



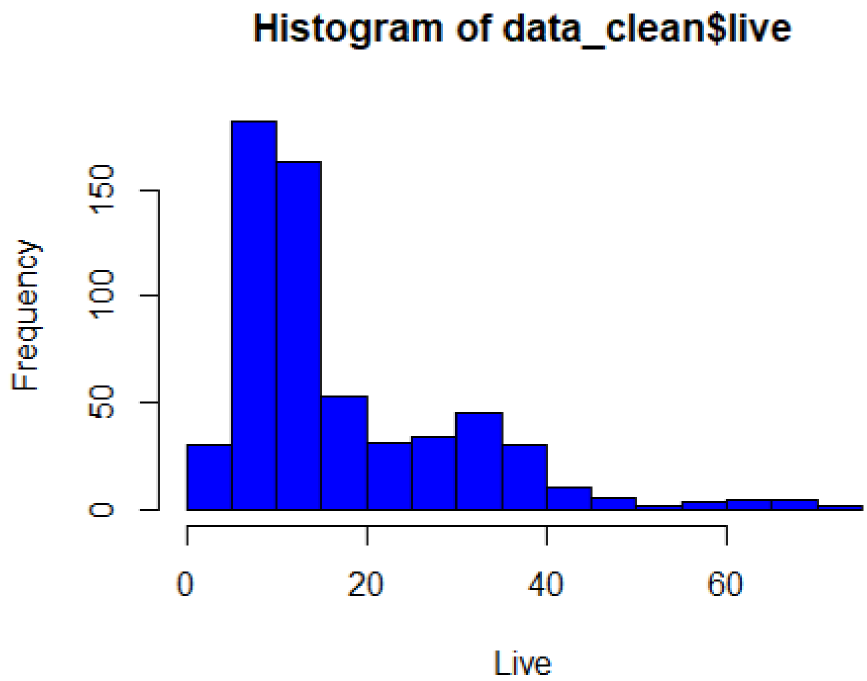
```r
hist(data_clean$nrgy, breaks=12,col="blue",xlab="Energy")
```



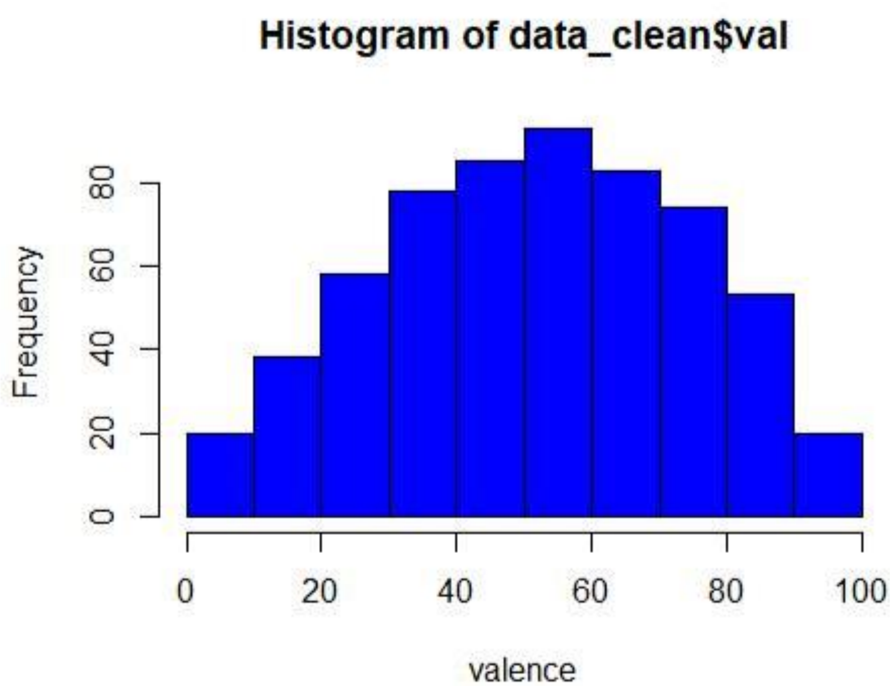

```r
hist(data_clean$dnce, breaks=12,col="blue",xlab="Danceability")
```



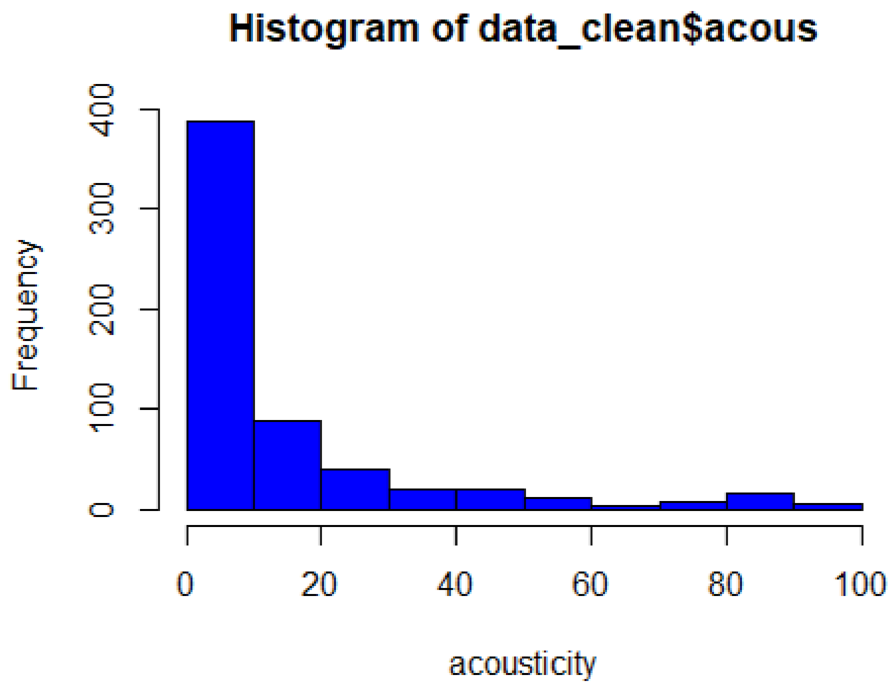
```
hist(data_clean$live, breaks=12,col="blue",xlab="Live")
```



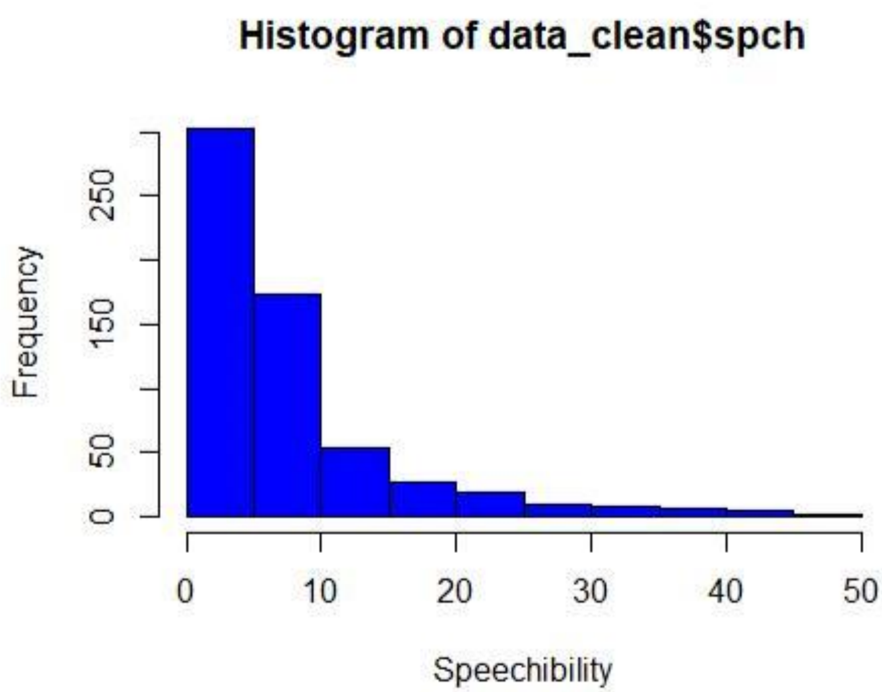

```
hist(data_clean$val, breaks=12,col="blue",xlab="valence")
```



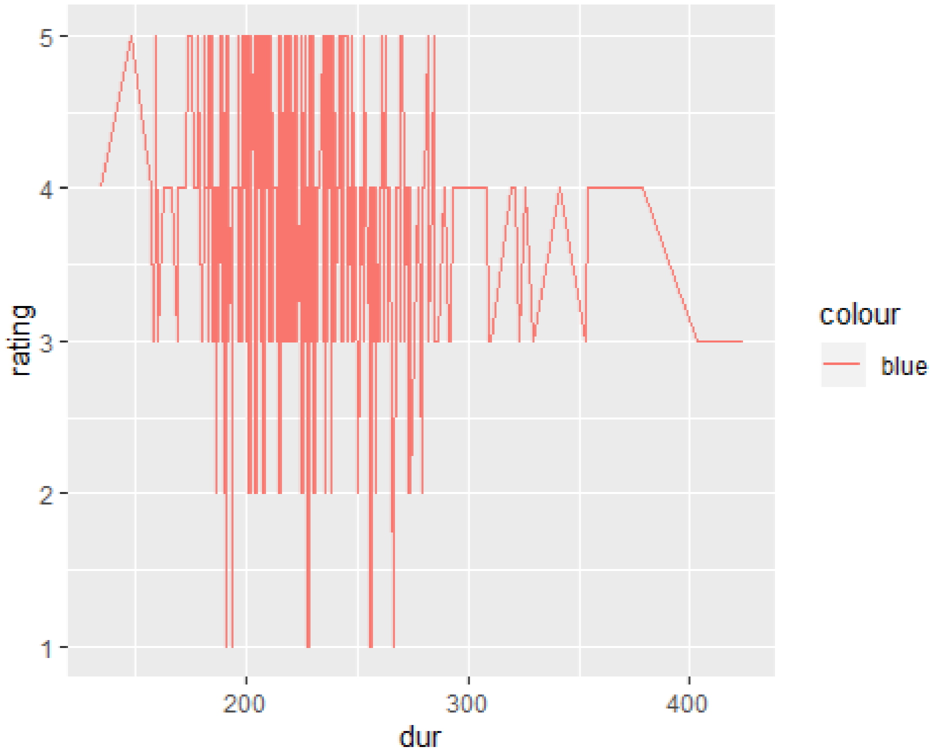
```r
hist(data_clean$acous, breaks=12,col="blue",xlab="acousticity")
```




```r
hist(data_clean$spch, breaks=12,col="blue",xlab="Speechibility")
```



```r
#Line chart for popularity and Duration
ggplot(data_clean) +geom_line(aes(x = dur, y = rating, color = "blue"))
```




```r
# T-Test on dataset columns Duration and rating
t.test(data_clean$dur,data_clean$rating, var.equal = TRUE, paired=FALSE)
```

```
##
##  Two Sample t-test
##
## data:  data_clean$dur and data_clean$rating
## t = 158.71, df = 1202, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  218.0532 223.5116
## sample estimates:
##  mean of x  mean of y
## 224.611296   3.828904
```

```r
#Comparing relation between two top genre from 2010 to 2019.
star5 = data_clean[which(data_clean$rating==5),]
```

```r
with(star5,t.test(dnce[top.genre=="dance pop"],dnce[top.genre=="pop"],var.equ
al=TRUE))
```

```
##
##  Two Sample t-test
##
```

```
## data:  dnce[top.genre == "dance pop"] and dnce[top.genre == "pop"]
## t = -1.0029, df = 40, p-value = 0.3219
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -13.676389   4.604961
## sample estimates:
## mean of x mean of y
##  67.03571  71.57143
```

```
with(star5,t.test(nrgy[top.genre=="dance pop"],nrgy[top.genre=="pop"],var.equ
al=TRUE))
```

```
##
##  Two Sample t-test
##
## data:  nrgy[top.genre == "dance pop"] and nrgy[top.genre == "pop"]
## t = 1.7587, df = 40, p-value = 0.08629
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.433565 20.647851
## sample estimates:
## mean of x mean of y
##  66.67857  57.07143
```

```
with(star5,t.test(bpm[top.genre=="dance pop"],bpm[top.genre=="pop"],var.equal
=TRUE))
```

```
##
##  Two Sample t-test
##
## data:  bpm[top.genre == "dance pop"] and bpm[top.genre == "pop"]
## t = 2.1881, df = 40, p-value = 0.03456
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   1.147886 28.923542
## sample estimates:
## mean of x mean of y
##  119.3929  104.3571
```

```
with(star5,t.test(val[top.genre=="dance pop"],val[top.genre=="pop"],var.equal
=TRUE))
```

```
##
##  Two Sample t-test
##
## data:  val[top.genre == "dance pop"] and val[top.genre == "pop"]
## t = -1.4541, df = 40, p-value = 0.1537
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -27.825938   4.540224
## sample estimates:
```

```
## mean of x mean of y
##  48.78571  60.42857

#------------------PCA-----------------------

#Splitting the rating column in 2 groups as we need 2 levels for t test
#and var test (f test) calculation, so rating 1 has ratings in range 1 to
3 #and rating 5 has ratings in range from 4 to 5.
#A new column v16 stores this new rating value which is used for above
mentio ned tests

for(y in 1:length(data_clean$rating)){
  if(data_clean[y,15] >= 1 & data_clean[y,15] <= 3){
    data_clean[y,16] = 1
  }else{
    data_clean[y,16] = 5
  }
}
View(data_clean)
#We are selecting audio properties to check if any correlation
#exist between them and does that affect the rating energy, danceability, val
ence, acoustics
#and speechability is observed.



aud_prop_cor = cor(data_clean[c(7,8,11,13,14)])
##               nrgy         dnce        val       acous        spch
## nrgy    1.0000000  0.16685024  0.4102908 -0.5625564  0.10711812
## dnce    0.1668502  1.00000000  0.5049296 -0.2413363 -0.02922118
## val     0.4102908  0.50492963  1.0000000 -0.2486811  0.12284677
## acous -0.5625564  -0.24133632 -0.2486811  1.0000000  0.00246410
## spch    0.1071181  -0.02922118  0.1228468  0.0024641  1.00000000

# Correlation is low but danceability and valence are closely related

# Calculating PCA for the cleaned data
data_pca = prcomp(aud_prop_cor,scale. = TRUE)
data_pca

## Standard deviations (1, .., p=5):
## [1] 1.4439153 1.0176814 1.0011165 0.7365874 0.5784789
##
## Rotation (n x k) = (5 x 5):
##                PC1         PC2         PC3         PC4          PC5
## nrgy  -0.53106816  0.3018103 -0.3408606 -0.3818033 -0.60408400
## dnce  -0.43372652 -0.5131816  0.3929811  0.4823965 -0.40172805
## val   -0.52681796 -0.1571937  0.3907000 -0.5388521  0.50472255
## acous  0.49239464 -0.1382874  0.5100046 -0.5094188 -0.46777338
## spch  -0.09928882  0.7757074  0.5626977  0.2676767 -0.01184546

summary(data_pca)
```

```
## Importance of components:
##                          PC1    PC2    PC3    PC4     PC5
## Standard deviation     1.444 1.0177 1.0011 0.7366  0.57848
## Proportion of Variance  0.417 0.2071 0.2004 0.1085  0.06693
## Cumulative Proportion   0.417 0.6241 0.8246 0.9331  1.00000
```

```
data_pca$x
```

```
##              PC1          PC2           PC3          PC4            PC5
## 1    -1.168320396 -0.435362099  -0.039151263 -1.271456683 -0.2412041809
## 2     -1.317131044  1.378217388   1.385378953  -0.136793273 -1.1308962645
## 3     -1.432924832  0.285364744   0.704262305 -0.036255619  -0.3415971627
## 4    -1.602879141 -0.305790987 -0.636065986 -0.552231502  -0.2164379874
## 5    -0.445434523 -0.041214120 -1.082152129  0.039428584  -0.4127259666
```

```
data_pca1 = cbind(data.frame(data_clean$V16),data_pca$x)
data_pca1
```

```
##      data_clean.V16          PC1          PC2           PC3          PC4
## 1                  5 -1.168320396 -0.435362099  -0.039151263 -1.271456683
## 2                  5 -1.317131044  1.378217388   1.385378953 -0.136793273
## 3                  5 -1.432924832  0.285364744   0.704262305 -0.036255619
## 4                  5 -1.602879141 -0.305790987  -0.636065986 -0.552231502
## 5                  5 -0.445434523 -0.041214120 -1.082152129   0.039428584


##             PC5
## 1    -0.2412041809
## 2    -1.1308962645
## 3    -0.3415971627
## 4    -0.2164379874
## 5    -0.4127259666
```

```
var.test(PC3~data_clean$V16,data=data_pca1)
```

```
##
##  F test to compare two variances
##
## data:  PC3 by data_clean$V16
## F = 1.022, num df = 146, denom df = 454, p-value = 0.8534
## alternative hypothesis: true ratio of variances is not equal to
1 ## 95 percent confidence interval:
##  0.7915999 1.3436023
## sample estimates:
## ratio of variances
##          1.021978
```

```r
#t.test(PC1~data_clean$V16,data=data_pca)
#t.test(PC2~data_clean$V16,data=data_pca)
t.test(PC3~data_clean$V16,data=data_pca1)
```

```
##
##  Welch Two Sample t-test
##
## data:  PC3 by data_clean$V16
## t = -0.065215, df = 245.03, p-value = 0.9481
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.1945103  0.1820429
## sample estimates:
## mean in group 1 mean in group 5
##    -0.0047115020.001522178
```

```r
#Taking out all numerical values
data_clean_num = data_clean[c(7,8,11,13,14)]

#scaling the data and finding generalized euclidean
distance scale_data = scale(data_clean_num) scale_data
```

```
##            nrgy        dnce         val       acous        spch
## 11.13355998  0.19831789  1.23201789  0.22549588 -0.58122490
## 21.37861002  0.79675083  0.52143129  0.46609943  1.95619892
## 30.82724742  0.87155495  0.83231293 -0.20759050  0.75426132
## 41.31734751  0.42273025  0.83231293 -0.68879760 -0.58122490
## 50.82724742 -0.02609446 -0.41121364 -0.59255618 -0.58122490
## 60.94977244  0.64714260  0.07731466 -0.49631476  0.75426132
## 70.45967236  0.79675083  1.32084122 -0.68879760  0.08651821
## 80.33714734 -0.92374387 -0.63327195 -0.35195263 -0.58122490
## 9   -2.05209058 -1.22296034 -1.69915187  2.87213490 -0.71477352
## 10   0.09209730  1.09596730  0.38819630 -0.06322837 -0.58122490
## 11   1.01103496 -0.17570269 -0.23356699 -0.54443547 -0.71477352
## 12   0.76598491 -0.17570269 -0.18915532  0.89918581 -0.58122490
## 13   0.70472240  0.94635907  0.47701962  0.17737517 -0.44767628
## 14   0.76598491  1.39518377  0.83231293 -0.64067689 -0.58122490
## 15   0.82724742 -1.52217681  1.14319457 -0.64067689  4.89426861
## 16   0.27588483  1.02116318 -0.85533027  0.27361659 -0.71477352
## 17  -0.58179032  0.57233848  1.36525288 -0.44819405 -0.71477352
## 18   0.64345989  1.32037965 -0.36680197 -0.35195263 -0.18057903
## 19   0.58219738 -0.10089858 -0.67768362 -0.68879760 -0.44767628
## 20  -0.58179032  1.39518377  1.05437124 -0.64067689  0.22006683
## 21  -0.45926530  1.17077142 -0.54444863 -0.68879760 -0.18057903
## 22  -0.15295275 -1.52217681 -0.32239031 -0.64067689 -0.44767628
## 23  -0.15295275  0.64714260  0.96554792 -0.68879760 -0.71477352
## 24   0.64345989  0.42273025  0.92113625 -0.68879760 -0.58122490
## 25   1.50113504 -0.84893975  0.56584295  0.56234085 -0.44767628
## 26   0.76598491 -1.22296034  0.96554792 -0.68879760 -0.58122490
```

```
## 27   1.19482249 -0.25050681  1.58731120 -0.59255618 -0.44767628
## 28   1.43987253  0.04870966  0.92113625 -0.59255618  0.08651821
## 29   0.58219738  0.57233848  0.29937297 -0.59255618 -0.58122490
## 30   0.27588483  1.09596730 -0.54444863 -0.68879760 -0.44767628
## 31   0.03083479  0.42273025  1.18760623 -0.44819405 -0.44767628
## 32   1.13355998  0.27312201  1.36525288 -0.64067689 -0.04703041
## 33   0.82724742  0.79675083 -0.01150867 -0.30383192  0.48716408
## 34  -1.13315292 -0.32531093 -0.50003696  0.17737517  3.82587963
## attr(,"scaled:center")
##      nrgy      dnce       val     acous      spch
## 70.496678 64.348837 52.259136 14.313953  8.352159
## attr(,"scaled:scale")
##     nrgy     dnce      val    acous     spch
## 16.32320 13.36825 22.51661 20.78107  7.48791
```

```r
dist_data = dist(scale_data,method ="euclidean")
dist_data
```

```
##             1         2         3         4         5         6         7
## 2   2.7238786
## 3   1.6364372 1.5181422
## 4   1.0391513 2.8306534 1.5673945
## 5   1.8744619 3.0676005 2.0697475 1.4132454
## 6   1.9680181 1.6656587 0.8477956 1.6050133 1.5810459
## 7   1.4498102 2.5125676 1.0280117 1.2490151 2.0657549 1.5138922
## 8   2.3887337 3.5335273 2.7231597 2.2439226 1.0738587 2.2698659 2.7116449
## 9   5.2709288 5.8054311 5.5433444 5.7591492 4.8377453 5.4008163 5.7298214
## 10  1.6388458 2.9122336 1.6100963 1.5946983 1.6488519 1.7334315 1.3898511
## 11  1.7068897 3.1314652 2.1302944 1.2754313 0.3284352 1.7140169 2.0805673
## 12  1.6578885 2.9072310 2.2698846 2.0560183 1.5168190 2.1243361 2.5074200
## 13  1.1548335 2.5180265 1.3189746 1.2425057 1.5363158 1.4858857 1.3528067
## 14  1.5740453 2.9143700 1.4996758 1.1189201 1.8900949 1.7227100 1.0671484
## 15  5.8131836 3.9906419 4.8118420 5.8397649 5.8853880 4.7976619 5.3535852
## 16  2.4062116 3.2144952 2.3588443 2.2880084 1.5382065 2.0529735 2.5274831
## 17  1.8899108 3.5462081 2.1389007 1.9972124 2.3532995 2.4840378 1.3553362
## 18  2.1329345 2.6150756 1.6024559 1.7238785 1.4377587 1.2802664 1.8278156
## 19  2.2122972 3.1607479 2.2275590 1.7642442 0.4046592 1.6572275 2.2583738
## 20  2.4080691 2.9537138 1.6678765 2.2904332 2.6072480 2.0411011 1.2385167
## 60
## 61
## 62
## 63
## 64
## 65
## 66
## 67
## 68
## 69
## 70
## 71
```

```
## 72
## 73
## 74
## 75
## 76
## 77
## 78
## 79
#  [ reached getOption("max.print") -- omitted 435 rows ]
```

```r
#As we have a column of rating which classifies the songs from 1-5. We can as
sume that K = 5
(kmeans5 <- kmeans(scale_data,5,nstart = 20))
```

```
## K-means clustering with 5 clusters of sizes 224, 106, 176, 45, 51
##
## Cluster means:
##         nrgy       dnce        val      acous        spch
## 1  0.6412719  0.4674791  0.8616563 -0.3302553 -0.19011822
## 2 -0.7667337  0.7995736 -0.1619218 -0.1553841 -0.07852773
## 3  0.1951297 -0.6708549 -0.7695350 -0.2753970 -0.28074050
## 4  0.1601668 -0.2538314  0.3595754  0.2586457  2.87323279
## 5 -2.0376759 -1.1760244 -1.1096088  2.4956611 -0.56813190
##
## Clustering vector:
##    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  1
9   20
##    1   4   1   1   3   1   1   3   5   1   3   3   1   1   4   2   1   1
3    2
##   21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  3
9   40
##    2   3   1   1   1   1   1   1   1   2   1   1   1   4   3   1   5   1
1    3
##   41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  5
9   60
##    4   2   3   1   5   1   1   1   4   1   1   5   5   1   3   1   1   1
1    2
##   61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  7
9   80
##    1   1   3   3   3   3   2   1   1   1   2   1   1   1   3   4   3   1
2    2
##   81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  9
9  100
##    1   3   1   3   1   1   3   5   3   1   1   1   1   1   5   4   5   4
2    1
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120
##    4   1   1   4   3   1   3   1   1   2   2   1   3   1   1   2   1   1
1    1
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 13
```

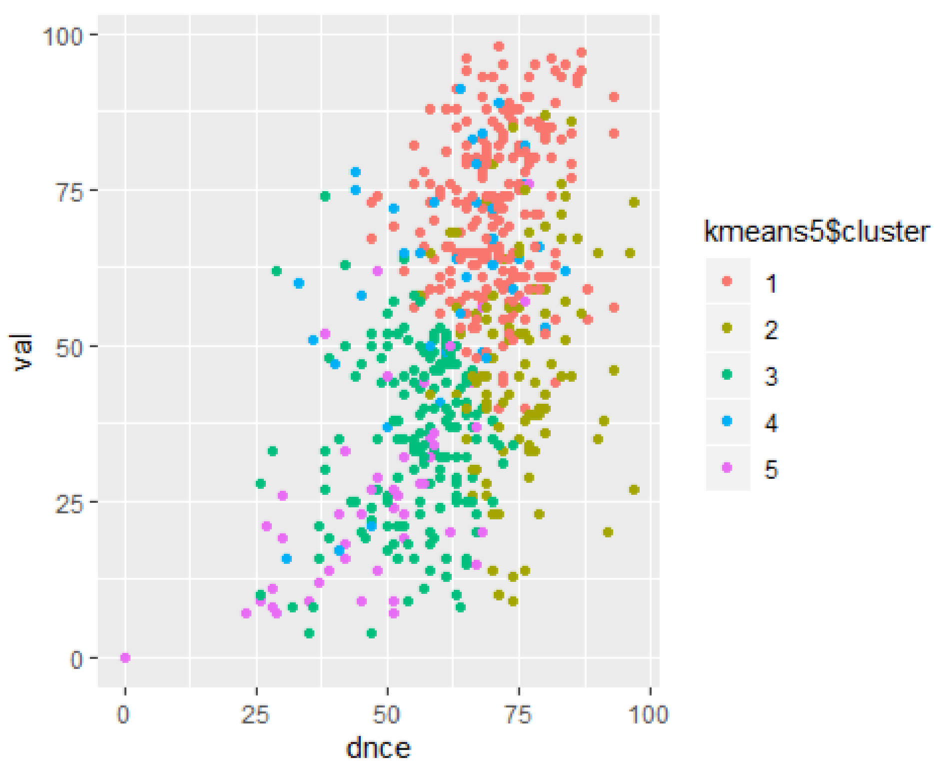
```
9  140
##   3   1   3   2   1   2   1  3   1   1   1   1   3   3   1   1   1   3
1   1
## 141  142  143 144 145 146  147 148  149  150 151  152 153  154 155 156 157 158  15
9  160
##   3   3   2   3   3   3   3  1   5   3   1   3   1   1   3   3   3   1
2   1
##
## Within cluster sum of squares by cluster:
## [1]  336.7368 217.3462 303.3063 197.3781193.3288
##  (between_SS / total_SS =  58.5  %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withi
nss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
kmeans5
```

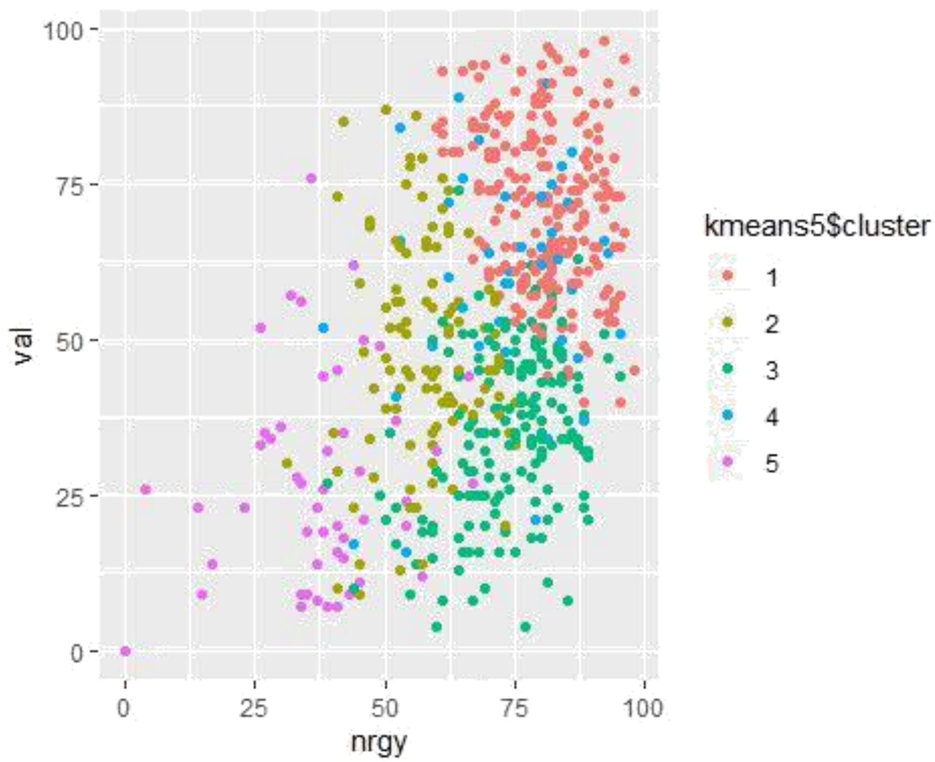
```
## K-means clustering with  5 clusters of sizes 224, 106, 176, 45, 51
##
## Cluster means:
##         nrgy       dnce         val      acous         spch
## 1  0.6412719  0.4674791  0.8616563 -0.3302553 -0.19011822
## 2 -0.7667337  0.7995736 -0.1619218 -0.1553841 -0.07852773
## 3   0.1951297 -0.6708549 -0.7695350 -0.2753970 -0.28074050
## 4  0.1601668 -0.2538314  0.3595754  0.2586457  2.87323279
## 5 -2.0376759 -1.1760244 -1.1096088  2.4956611 -0.56813190
##
## Clustering vector:
##   1   2   3   4   5   6   7  8   9  10  11  12  13  14  15  16  17  18  1
9  20
##   1   4   1   1   3   1   1  3   5   1   3   3   1   1   4   2   1   1
3   2
##  21  22  23  24  25  26  27 28  29  30  31  32  33  34  35  36  37  38  3
9  40
##   2   3   1   1   1   1   1  1   1   2   1   1   1   4   3   1   5   1
1   3
##  41  42  43  44  45  46  47 48  49  50  51  52  53  54  55  56  57  58  5
9  60
##   4   2   3   1   5   1   1  1   4   1   1   5   5   1   3   1   1   1
1   2
##  61  62  63  64  65  66  67 68  69  70  71  72  73  74  75  76  77  78  7
9  80
##   1   1   3   3   3   3   2  1   1   1   2   1   1   1   3   4   3   1
2   2
##  81  82  83  84  85  86  87 88  89  90  91  92  93  94  95  96  97  98  9
9  100
##   1   3   1   3   1   1   3  5   3   1   1   1   1   1   5   4   5   4
```

```
2   1
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120
##    4   1   1   4   3   1   3   1   1   2   2   1   3   1   1   2   1   1
1    1
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 13
9 140
##
## Within cluster sum of squares by cluster:
## [1] 336.7368 217.3462 303.3063 197.3781 193.3288
##  (between_SS / total_SS =  58.5 %)
##
## Available components:
##
##  [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withi
nss"
##  [6] "betweenss"    "size"         "iter"         "ifault"
```

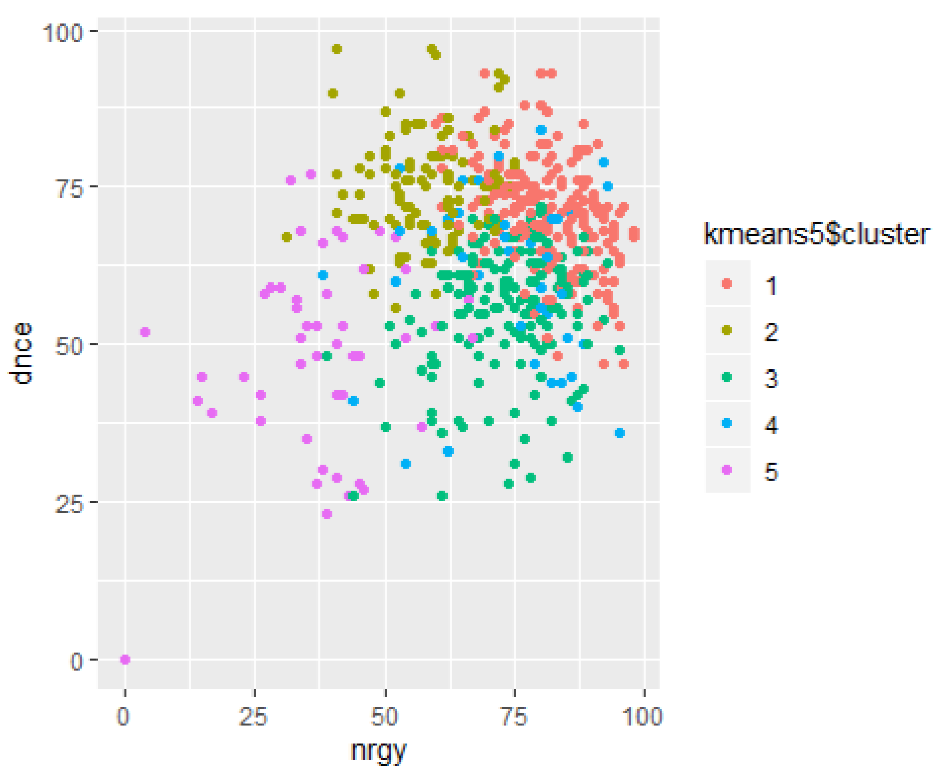
```r
library(ggplot2)
kmeans5$cluster <- as.factor(kmeans5$cluster)
ggplot(data_clean_num, aes(dnce,val,color = kmeans5$cluster)) + geom_point()
```




```r
ggplot(data_clean_num, aes(nrgy,val,color = kmeans5$cluster)) + geom_point()
```

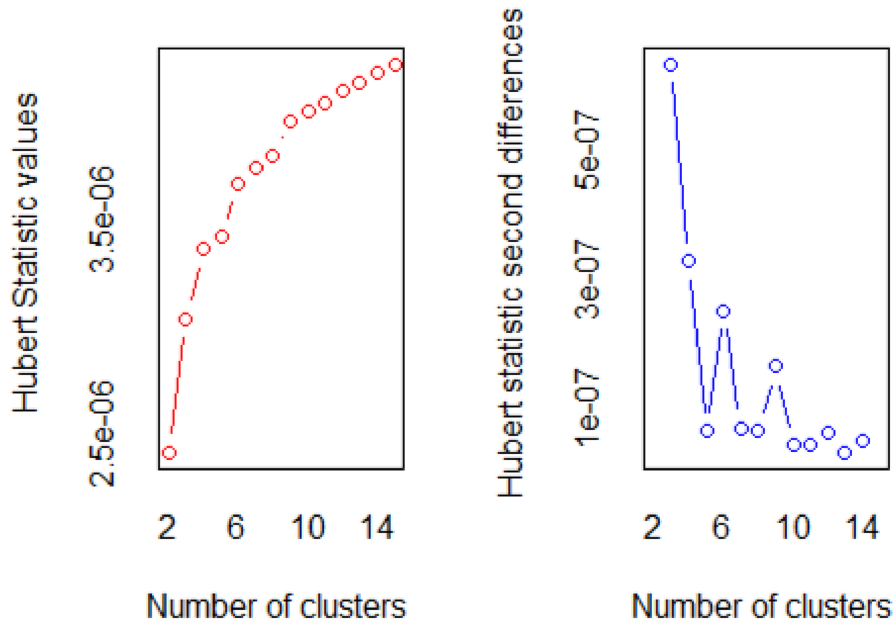



```
ggplot(data_clean_num, aes(nrgy,dnce,color = kmeans5$cluster)) + geom_point()
```

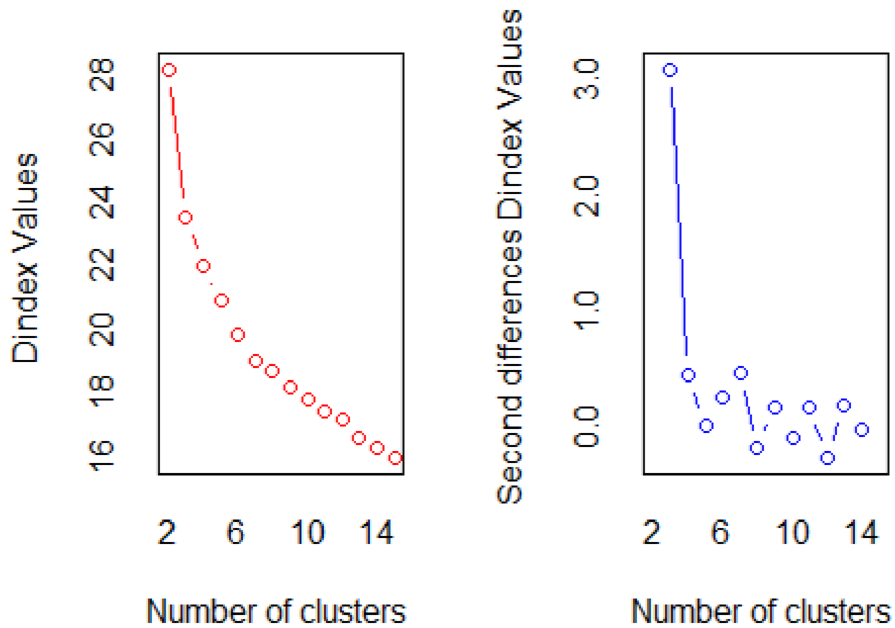



```
kmeans5$size
```

```
## [1] 224 106 176  45  51

#To validate our assumption we took the help of the nbclust function to
find optimal no. of clusters
library(NbClust)
nb_clust = NbClust(data_clean_num, distance="euclidean", method = 'kmeans')
```




```
## *** : The Hubert index is a graphical method of determining the number
of clusters.
##                 In the plot of Hubert index, we seek a significant knee th
at corresponds to a
##                 significant increase of the value of the measure i.e the s
ignificant peak in Hubert
##                 index second differences plot.
##
```




```
## *** : The D index is a graphical method of determining the number of
clust ers.
##                 In the plot of D index, we seek a significant knee (the si
gnificant peak in Dindex
##                 second differences plot) that corresponds to a significant
increase of the value of
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 6 proposed 2 as the best number of clusters
## * 13 proposed 3 as the best number of clusters
## * 1 proposed 7 as the best number of clusters
## * 1 proposed 10 as the best number of clusters
## * 1 proposed 13 as the best number of clusters
## * 1 proposed 14 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  3
##
##
## *******************************************************************

nb_clust
```

```
## $All.index
##         KL       CH Hartigan     CCC    Scott      Marriot      TrCovW   T
raceW
## 2   0.6566 294.1695 278.8921 38.8293 3091.213 2.655594e+25 18945387440 580
752.9
## 3  12.2434 354.3072  86.8032 35.6981 3859.559 1.667418e+25  6769059795 396
467.0
## 4   0.8921 298.8687  71.1807 36.3337 4218.346 1.633376e+25  5675880069 346
285.5
## 5   1.2556 268.1786  57.4013 34.9222 4534.861 1.508574e+25  4549079185 309
451.2
## 6   0.8257 246.2382  52.8810 33.9192 4761.104 1.491807e+25  3625268843 282
307.4
## 7   2.6016 231.8289  26.9968 33.4130 4951.100 1.480946e+25  2769864703 259
300.6
## 8   1.0319 211.2276  34.1323 32.0636 5062.802 1.606717e+25  2580254660 248
046.1
## 9   3.9593 199.3747  24.1545 31.4793 5255.879 1.475553e+25  2132325227 234
567.4
## 10  0.1270 186.8090  33.9347 30.6615 5375.461 1.493485e+25  1912586374 225
386.8
## 11  9.0068 180.8530  19.7266 30.5906 5523.029 1.414255e+25  1903663532 213
167.6
## 12  0.0869 171.4024  33.3160 29.9571 5649.976 1.363086e+25  1811327964 206
282.3
## 13  2.2771 168.4813  23.9604 30.1924 5778.992 1.291137e+25  1551426335 195
256.6
## 14  1.3389 163.4130  21.1317 34.3777 5875.120 1.276420e+25  1397944036 187
624.0
## 15  0.9953 158.4335  10.7028 34.2755 5970.065 1.251483e+25  1258871094 181
115.1
##     Friedman   Rubin Cindex     DB Silhouette   Duda   Pseudot2   Beale Ratk
owsky
## 2    84.5401 14.0498 0.2656 1.4003     0.3518 0.8061    80.3538  0.7512    0
.3199
## 3    96.5565 20.5804 0.2317 1.1729     0.3200 1.5172  -151.6981 -1.0631    0
.3311
## 4   110.5763 23.5628 0.2524 1.4071     0.2432 1.1898   -41.1539 -0.4969    0
.3028
## 5   113.7982 26.3675 0.2370 1.3359     0.2472 1.0026    -0.4342 -0.0080    0
.2803
## 6   125.7339 28.9027 0.2490 1.4294     0.2122 1.6023   -87.9566 -1.1673    0
.2674
## 7   139.9383 31.4671 0.2450 1.3213     0.2230 1.3023   -14.3928 -0.7108    0
.2564
## 8   146.0831 32.8949 0.2424 1.3892     0.2160 1.6201   -88.0353 -1.1885    0
.2457
## 9   154.4269 34.7851 0.2310 1.3657     0.1999 1.2011   -17.2464 -0.5185    0
.2390
## 10  162.8369 36.2020 0.2250 1.4351     0.1986 1.6357   -44.3053 -1.2021    #
```

```
# $All.CriticalValues
##    CritValue_Duda CritValue_PseudoT2 Fvalue_Beale
## 2          0.7826            92.7817       0.5852
## 3          0.7612           139.6301       1.0000
## 4          0.7469            87.4204       1.0000
## 5          0.7439            58.5210       1.0000
## 6          0.7102            95.4866       1.0000
## 7          0.6025            40.9127       1.0000
## 8          0.7083            94.7267       1.0000
## 9          0.6836            47.6729       1.0000
## 10         0.6729            55.4136       1.0000
## 11         0.6642            46.5028       1.0000
## 12         0.6528            70.2005       1.0000
## 13         0.6616            75.2030       1.0000
## 14         0.6602            51.4755       1.0000
## 15         0.6573            47.9622       1.0000
##
## $Best.nc
##                      KL       CH Hartigan    CCC    Scott      Marriot
## Number_clusters   3.0000   3.0000   3.0000 2.0000    3.0000 3.000000e+00
## Value_Index      12.2434 354.3072 192.0889 38.8293 768.3461 9.541344e+24
##                      TrCovW   TraceW Friedman   Rubin Cindex      DB Silhou
ette
## Number_clusters           3      3.0   7.0000  3.0000 10.000 3.0000     2.
0000
## Value_Index     12176327645 134104.4  14.2044 -3.5482  0.225 1.1729     0.
3518
##                   Duda PseudoT2  Beale Ratkowsky     Ball PtBiserial Frey
## Number_clusters 2.0000   2.0000 2.0000    3.0000      3.0     3.0000    1
## Value_Index     0.8061  80.3538 0.7512    0.3311 158220.8     0.5354   NA
##                 McClain    Dunn  Hubert SDindex Dindex   SDbw
## Number_clusters  2.0000 13.0000       0  3.0000      0 14.0000
## Value_Index      0.4221  0.0523       0  0.1106      0  0.2508
##
## $Best.partition
##    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  1
9  20
##    1   1   1   1   2   1   1   2   3   1   2   2   1   1   1   2   1   2
2   1
##   21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  3
9  40
##    2   2   1   1   1   1   1   1   1   2   1   1   1   2   2   1   3   1
1   2
##   41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  5
9  60
##    1   2   2   1   3   1   1   1   1   1   2   3   3   1   2   2   1   1
1   2
##   61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  7
9  80
##    1   1   2   2   2   2   2   1   1   1   2   1   1   1   2   1   2   1
```

```r
# for 3 clusters
(kmeans3 <- kmeans(scale_data,3,nstart = 10))
```

```
## K-means clustering with 3 clusters of sizes 57, 313, 232
##
## Cluster means:
##          nrgy       dnce        val      acous        spch
## 1 -1.98115504 -1.0116715 -1.0205105  2.4550888 -0.37270161
## 2  0.35084822  0.5649298  0.7334154 -0.2569411  0.10102509
## 3  0.01340666 -0.5136109 -0.7387497 -0.2565409 -0.04472785
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  1
9  20
##   2   2   2   2   3   2   2   3   1   2   3   3   2   2   2   3   2   2
3   2
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  3
9  40
##   2   3   2   2   2   2   2   2   2   2   2   2   2   3   3   2   1   2
2   3
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  5
9  60
##   2   3   3   2   1   2   2   2   3   2   2   1   1   2   3   2   2   2
2   3
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  7
9  80
##   2   2   3   3   3   3   3   2   2   2   3   2   2   2   3   2   3   2
3   2
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  9
9 100
##   2   3   2   3   2   2   3   1   3   2   2   2   2   2   1   3   1   2
2   2
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120
##   3   2   2   2   3   2   3   2   2   2   2   2   3   2   2   2   2   2
2   2
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 13
9 140
##   3   2   3   3   2   2   2   3   2   2   2   2   3   3   2   2   2   3
2   2
## 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 15
9 160
##   3   3   2   3   3   3   3   2   1   3   2   3   2   2   3   3   3   2
3   2
## 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 17
9 180
##   2   2   2   2   2   2   2   3   3   2   3   2   2   2   2   2   3   3
2   2
## 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 19
9 200
```

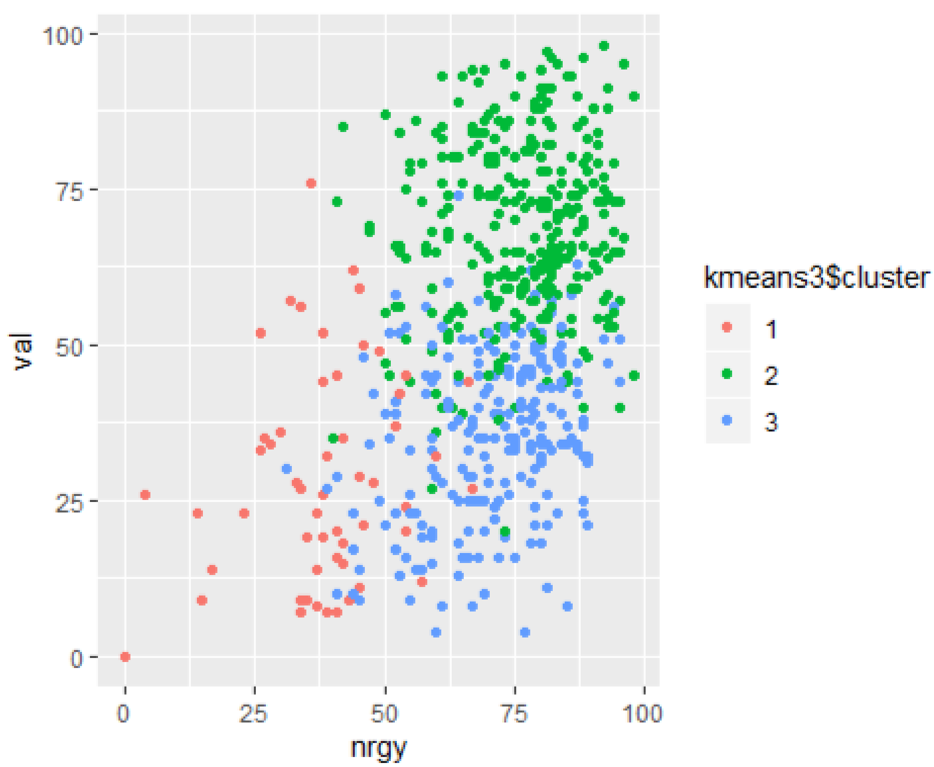
```
##   3   2   3   3   3   2   1   3   3   3   2   3   1   3   3   2   3   2##
## Within cluster sum of squares by cluster:
## [1] 277.5808 846.0713 654.2098
##  (between_SS / total_SS =  40.8 %)
##
## Available components:
##
##  [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withi
nss"
##  [6] "betweenss"    "size"         "iter"         "ifault"
```

```
kmeans3
```

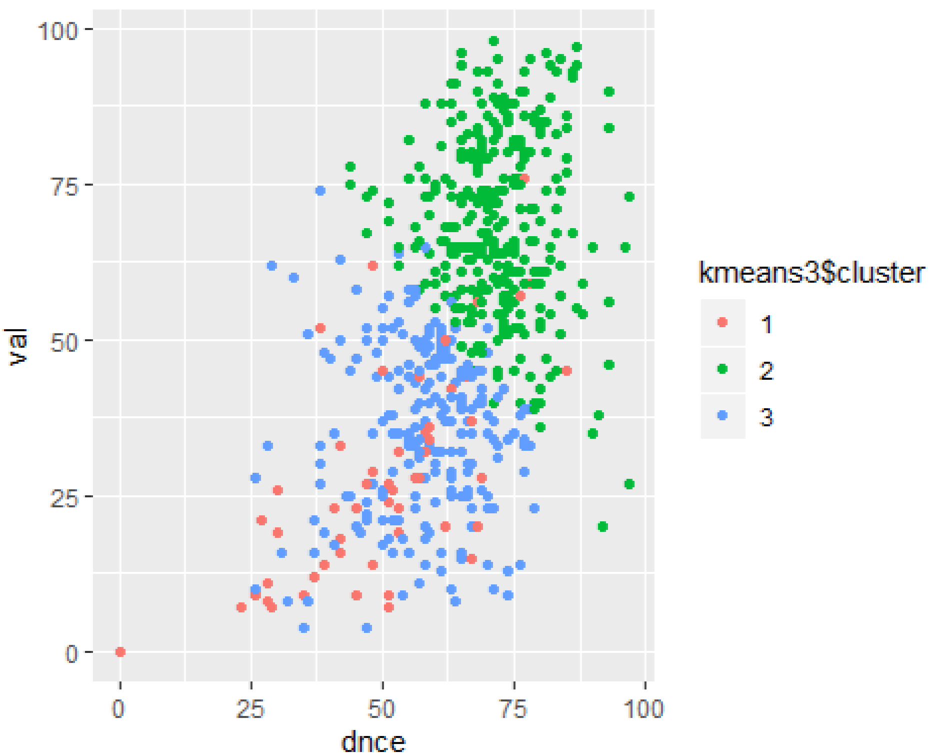
```
## K-means clustering with 3 clusters of sizes 57, 313, 232
##
## Cluster means:
##          nrgy       dnce        val      acous        spch
## 1  -1.98115504 -1.0116715 -1.0205105  2.4550888 -0.37270161
## 2   0.35084822  0.5649298  0.7334154 -0.2569411  0.10102509
## 3   0.01340666 -0.5136109 -0.7387497 -0.2565409 -0.04472785
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  1
9  20
##   2   2   2   2   3   2   2   3   1   2   3   3   2   2   2   3   2   2
3   2
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  3
9  40
##   2   3   2   2   2   2   2   2   2   2   2   2   2   3   3   2   1   2
2   3
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  5
9  60
##   2   3   3   2   1   2   2   2   3   2   2   1   1   2   3   2   2   2
2   3
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  7
9  80
##   2   2   3   3   3   3   3   2   2   2   3   2   2   2   3   2   3   2
3   2
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  9
9 100
##   2   3   2   3   2   2   3   1   3   2   2   2   2   2   1   3   1   2
2   2
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120
##   3   2   2   2   3   2   3   2   2   2   2   2   3   2   2   2   2   2
2   2
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 13
9 140
##   3   2   3   3   2   2   2   3   2   2   2   2   3   3   2   2   2   3
2   2
```

```
## Within cluster sum of squares by cluster:
## [1] 277.5808 846.0713 654.2098
##  (between_SS / total_SS =  40.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withi
nss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

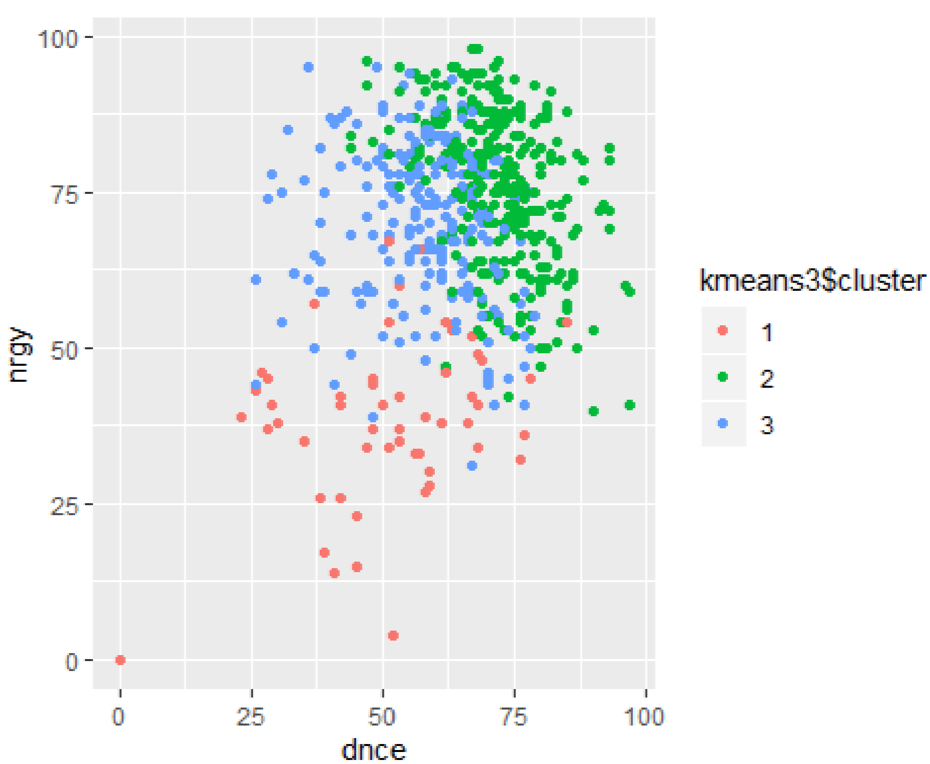
```
kmeans3$cluster <- as.factor(kmeans3$cluster)
ggplot(data_clean, aes(nrgy,val,color = kmeans3$cluster)) + geom_point()
```



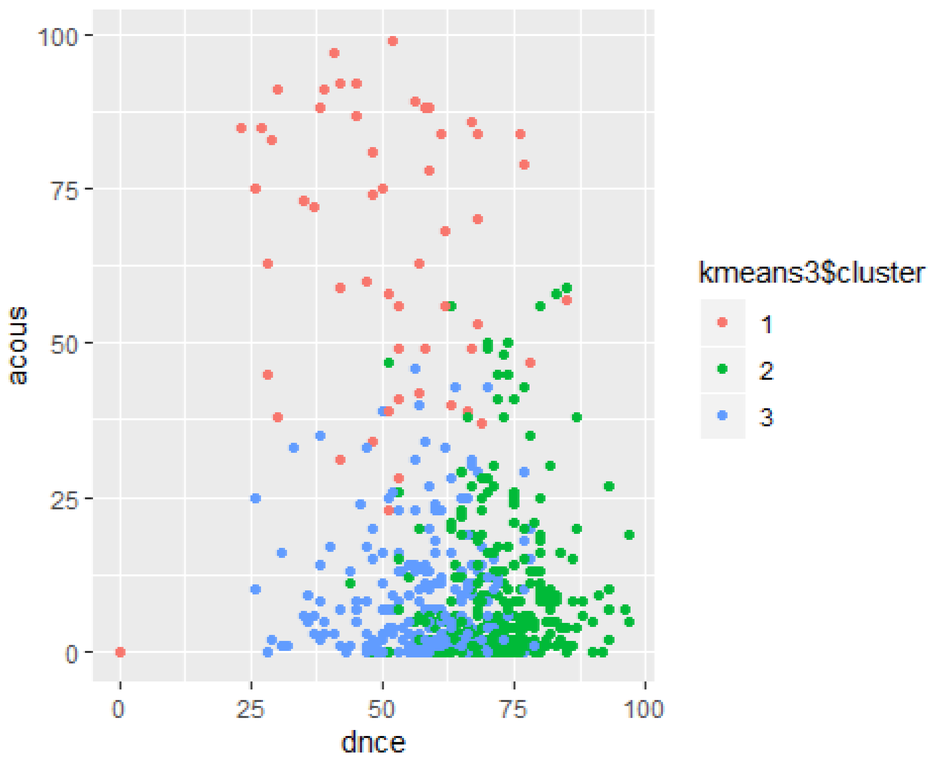

```
ggplot(data_clean, aes(dnce,val,color = kmeans3$cluster)) + geom_point()
```




```
ggplot(data_clean, aes(dnce,nrgy,color = kmeans3$cluster)) + geom_point()
```




```
ggplot(data_clean, aes(dnce,acous,color = kmeans3$cluster)) + geom_point()
```




```
kmeans3$withinss

## [1] 277.5808 846.0713

654.2098 kmeans3$size

## [1]  57 313 232


Conclusion:-
```

Based on the above visualizations we can conclude that we can cluster our dat a based on audio properties in 3 clusters.

Cluster 1: High acousticness, Low danceability, energy, valence

Cluster 2: Low acousticness, high danceability, energy, valence

Cluster 3 : Low acousticness, moderate danceability, energy, valence