Segmentation

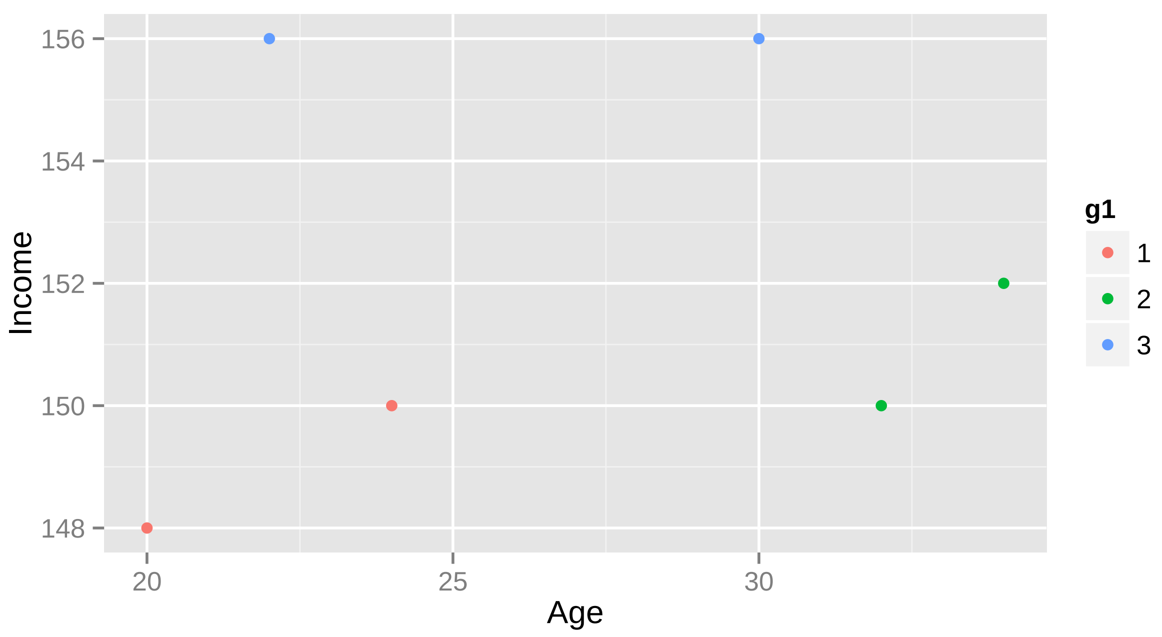
So far what we have seen in terms of pattern finding in the data has always been about developing some sort of equation to predict response. Or in simple words, we always had a response. It might come as a surprise but finding pattern in the data isn't always coupled with having a response against your observations [collection of people or object characteristic]. Have a look at these business problems and you'll have a better understanding about what are we talking about here:

- How should we customize our marketing campaign so that our different marketing campaigns are able to better target different segments in the population?
- Government's current development plan for education doesn't address different demographics properly. What are these demographics group present in the population which do not fare well with an umbrella program?
- One box fits for all days are gone from packaging industry with growth in e-commerce industry and requirement of different mediums and methods of delivery for various products. But in order to manufacture sustainable packaging solutions , we need to find out which products can have similar solutions. How to find that out?

You'll realize that undertone of all these problems is about finding similar observations in your data. And that concept is not that hard to understand. We are already vaguely familiar with this in our day to day life. For example we see that toy companies put age groups on toys that they sell. Different toys are targeting different Age segments. Similarly you'd notice , famed barbie dolls adapt according to different cultures and countries that they cater to.

What we need to do is to formalize this already intuitive concept of finding "groups" with in our data or observations. In order to start developing that framework, I'll ask you to a questions for me. Below given is a small data set containing Age and Income of few people. Accompanying that data is visual representation of the same data.

```
## Source: local data frame [6 x 2]
##
##   Age Income
## 1  32    150
## 2  30    156
## 3  34    152
## 4  20    148
## 5  22    156
## 6  24    150
```



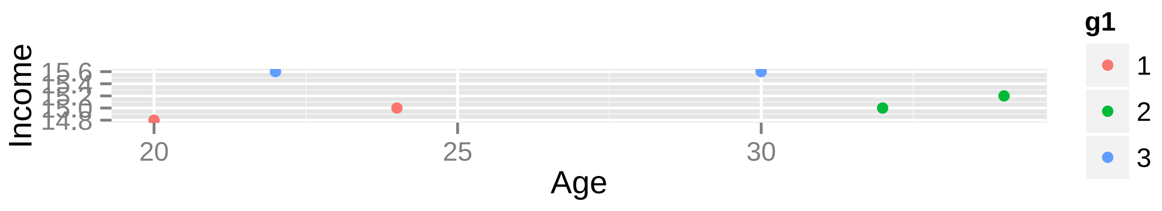
You can identify three groups in the data. I have coloured them accrodingly as well. Now , how did we say that these different people were in these groups. Based on how far apart and close they were . Isn't it? So this part we have figured out, we make groups based on distances between the observations.

That's all!. That's all there is to segmentation in theory. We'll figure out ways to exploit distance in various manners, but at the very basic level , its all about distances between observations.

So before we go ahead and start doing all kinds of numerical jugglery with distances, we need to understand one more fundamental concept here. That is distances depend on scale of the characteristics invloved.

To clearly understand this let's change scale of those incomes and see how these groups change.

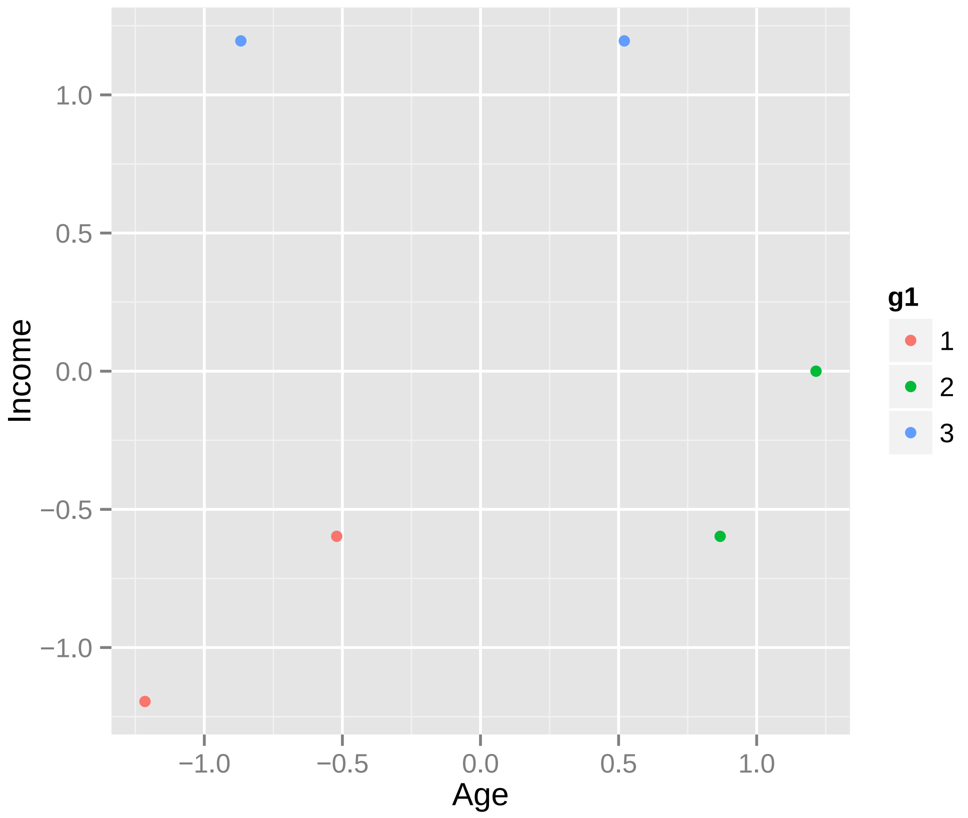
```
## Source: local data frame [6 x 2]
##
##   Age Income
## 1  32   15.0
## 2  30   15.6
## 3  34   15.2
## 4  20   14.8
## 5  22   15.6
## 6  24   15.0
```



So to get rid of this effect of scales we can standardize our data. [Subtract the mean and divide by standard deviation]. Standardzing our data removes effect of scale, which makes it easier to consider Age[small scale variables] and Income [Big scale variables] together and not worry about differnece in "Income" overshadowing difference in "Age".

```
## Source: local data frame [6 x 2]
```

```
##
##          Age     Income
## 1  0.8677627 -0.5976143
## 2  0.5206576  1.1952286
## 3  1.2148677  0.0000000
## 4 -1.2148677 -1.1952286
## 5 -0.8677627  1.1952286
## 6 -0.5206576 -0.5976143
```



There are some cases where you'd not want to use standardization blindly:

- Where standardization might result in loss of information. For example you have data which contains latitude and longitude information . If you standardize these columns , they wont represent the same geographical points any more, essentially data would have become garbage.
- When you might want to use the inherent weightage present in the original data. As we discussed statdardization removes effect of scale and brings down all factors at the same level. In some cases by your business experiences you know that one of the factor should have higher weightage in comparison to others. Then after standardization you can multiple that variable/factor values with wieghtage quantity.

Lets represnt what we just said , mathematically. Consider that we have hypothetical observations with characeteristics x and y. distance between obserations $(x_1, y_1)\, and\, (x_2, y_2)$

$$= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

you can see here that if y takes values with higher scale in comparison to x [e.g. Income taking values in lacs in comparison to Age taking values in]. we need to standardize them to remove effect of scale.So new variables are:

$$x' = \frac{(x - \bar{x})}{S_x}$$

and

$$y' = \frac{(y - \bar{y})}{S_y}$$

where $\bar{x}$ and $\bar{y}$ are means of x & y and S_x and S_y are standard deviation.You can chose median and MADs also to standardize if your data is skewed. distance formula remains same with new variable put in place

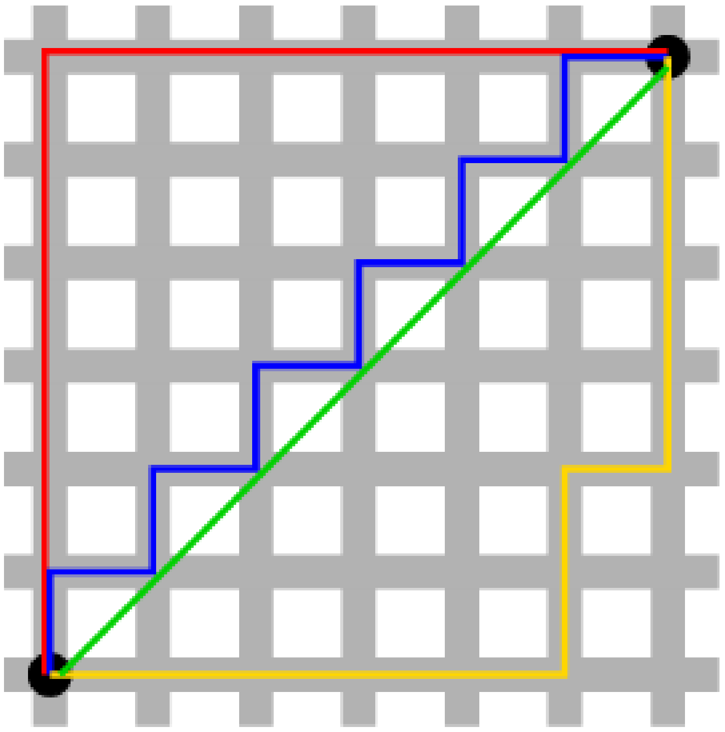
$$= \sqrt{(x_1' - x_2')^2 + (y_1' - y_2')^2}$$

Now when weightage are same for x and y, lets say you want to put weightage w on y, owing to your business experience , you can do that by simply multiplying values of y by w. Distance becomes

$$= \sqrt{(x_1' - x_2')^2 + (w * y_1' - w * y_2')^2}$$

Distances that we have been talking about are called eucledian distances. There are other distances as well which are rarely used. For completeness sake we'll mention them here, but we'll keep on using eucledian distances for general purposes.

$$ManhattanDistance = |x_1 - x_2| + |y_1 - y_2|$$



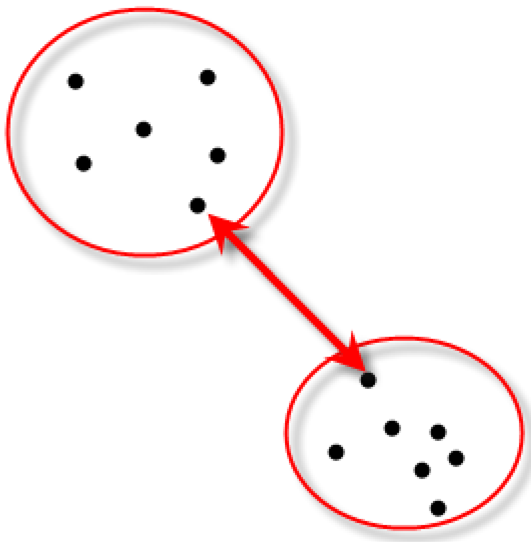
green line here represents traditional euclidian distance, whereas Red represnts manhattan distance. Blue and yellow are equivalent to manhattan distance.

$$MinkowskiDistance = (|x_1 - x_2|^p + |y_1 - y_2|^p)^{1/p}$$

Where p is called order of minkowski distance. Manhattan Distance and Eucledian Distances are two special instances of minkowski distance for p=1 and p=2.

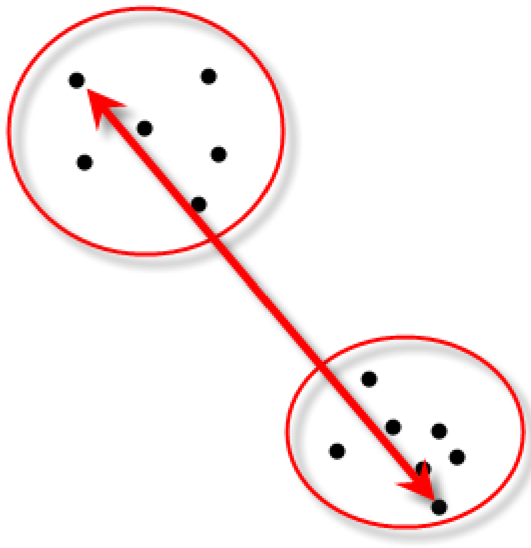
So we understand distances and it makes sense to pair up points based on that. If you haven't realised the coming predicament, think about this : "Groups can have more than two points, which requires us combining groups. How do we combine groups?"

Answer still is : **based on distances**. Problem is how to calculate distances between groups? There are many answers to this question. *[Combining groups is referred as linkage]*



Single Linkage

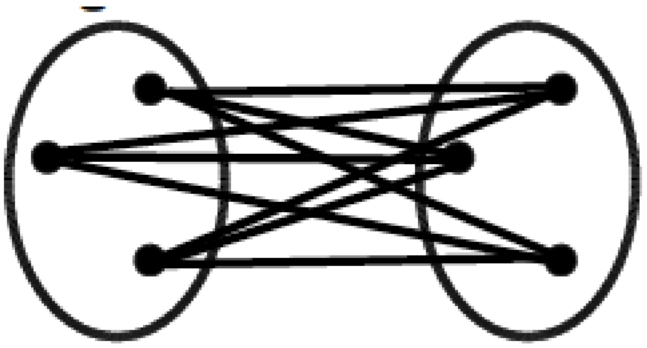
In this scenario distance between groups is considered as distance between closest points in the groups. This results in eventually diffused groups.



Complete Linkage

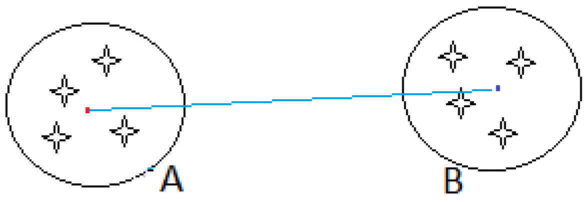
In this scenario distance between groups is considered as distance between farthest points in the groups. This results in eventually compact groups.

Single linkage and Complete linkage are tow extreme examples of linkage methods. There are others which are somewhere in between



Average Linkage

In this scenario , distance is defined as average of all pairwise distance between points.



Centroid Linkage

In this scenario, distance is defined as distance between centroid of each group.

Remember that all these methods are are to calcualte distance between groups. Different business requirement will call for different linkage methods to be used. Once you have calculated distances between grousp, You simply combine the ones which are nearest to each other according to the distances that you calculated.

We will not formally discuss two clustering or segmentation methods.

Hierarchical Clustering

This is the most primitive technique . It starts with calculating all pairwise distances between all data points and then combining them bottoms up to come up with groups in the data.

We have hypothetically represented points here by letters. The resulting dendogram here is nothing but representation of distances between points. Points connected by immediate branches of the tree are nearest. As you move up on the tree distance threshhold increases and groups size increases and number of groups go down. You can cut the tree horizontally to at any distance threshhold to get groups in your data.

If There are n observations/points in your data, while doing hierarchical clustering, you will have to calculate **n*n** distances. This become resource intesive very soon , very fast. Imagine a decent sized dataset with 10000 observation.You'll have to calculate 100 million distances for it.

Also the pretty dendogram that you see here becomes illegible very fast beyond 100 or so observations. Due to these limitations, Hierarchical clustering is rarely used with modern data. Its alternative is K-means algorithm.

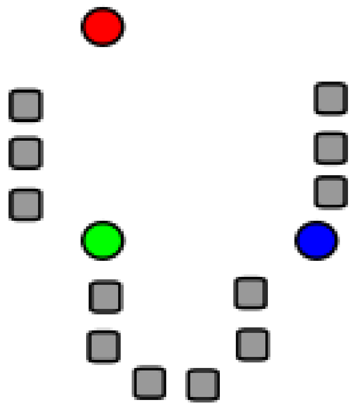
K-means Clustering

In this case , you already tell you algortihm, how many groups there are in your data. Then it follows following steps:

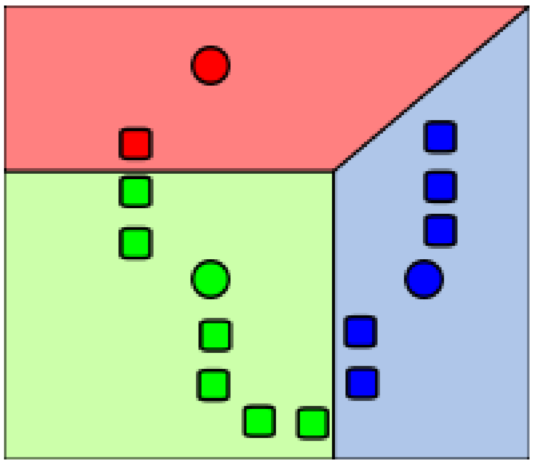
1. Chose K points[lets call them seed] randomly in your data *[There are some minimum distance criterion between the points]*
2. Assign membership to points for groups depending on which seed they are closest to.
3. Centroid of these groups become new seed. Repeat step 2 and 3 until convergence is achieved.

We'll discuss "convergence" in detail when we are learning how to decide "K" or Number of groups in your data.

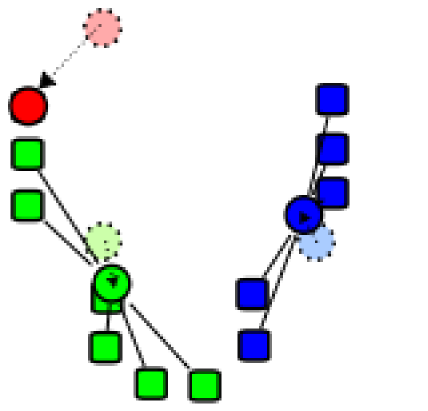
Here are a couple of figures explaining iterations of K-Means Algorithm.



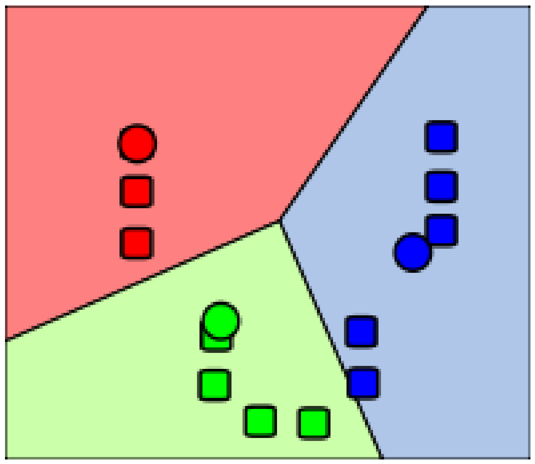
Step 1 :3 Random points [seeds] are chosen



Step 2: 3 Clusters are created by associating points with nearest mean [or seed]



Step 3: Centroid of these clusters become new means [or seed]



Step 4: Step 2 and 3 are repeated until convergence is reached.

Finding K There is no definite answer to "How many groups there are in my data?", you can have as many groups as you want. However we can find an "optimum" number of clusters/groups in the data. To understand that we need to find out what happens when we break our data into groups.

Total variance in your data can be measured in terms of total sum of squares , which is nothing but sum of squares of differences of data points from the overall mean.

Sum of squares are additive , we'll show formulas here for one variable. This can be generalised over multiple variables under some considerations which we'll discuss as limitations of K-means algorithms.

$$Totalsumofsquares = SST = \sum i = 1^n (x_i - \bar{x})^2$$

If we break this data into two groups. We can calculate within *[clusters]* sum of squares and between *[Clusters]* sum of squares.

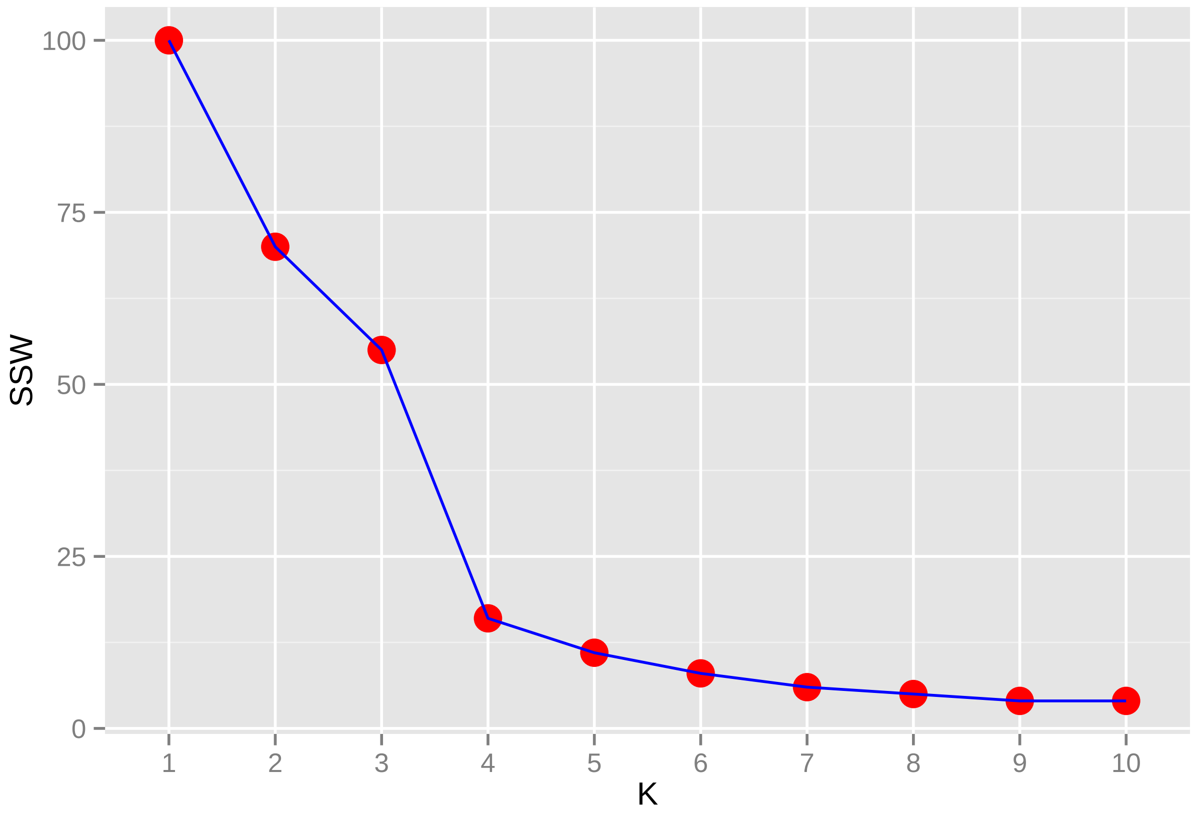
$$SST = SSW + SSB$$

where SSW and SSB are within sum of squares and between sum of squares respectively.

If we break our data, and calculate within group variance [SSW], it will go down and between group variance[SSB] will go up as we increase number of groups.

When we have no partition of groups in our data; SSB=0 and SSW=SST. More and more groups we make in our data, SSW will keep on going down and SSB will keep on going up. Eventually when you have as many groups as there are observations ; SSW=0 and SSB=T.

Initially when groups are well differentiated , increasing number of groups will result in good decrease in SSW . After a while this trend of significant change will give way to minimal change. We'll chose optimum K where this change happens significantly for the last time.



You can see here that beyond K=4, decrease in SSW is not remarkable . We can chose optimum number of clusters to be 4.

Convergence We can now discuss convergence of the K-means algorithm. If you recall, in step 3 , means/seed get re-assigned to new group means, and step 2 is repeated. Each Reassignment and repetition will result in within group variances until group membership of the observations stops changing . This is achieved in few iterations. Which implies that K-means algrotihm scales linearly with number of observations involved as opposed to Hierarchical clustering which scales exponentially.

Limitations of K-means Algorithm

- K-means can not be used with distance measures other than euclidean. Which is not a great limitation as you'd rarely use other distance measures.
- Presence of outliers will lead in sub-optimal grouping.
- If variables are correlated, variance changes will be over-estimated and might result in much less differentiated groups than expected.

Alternative to K-means is EM algortihm, which we will not be discussing here. You can explore that on your own.

General Limitations of any Clustering Algorithm

These limitation are more like fact of the trade rather than real limitations.

- No algortihm can really decide which variables to include while creating clusters. There is no p-value like measure here which tells you to drop/include variables. You'll have to take that decision based on your business experience and trial and error.
- It will not always be possible to lable resulting groups. You might have to go through before you are able to come up with meaningful labels for your resulting groups

Case Study

We'll first look at hierarchical clustering example. In this example we are going to find clusters in the mtcars data, where we'd be grouping same kind of cars based on their various characteristics.

```
data(mtcars)
cars=mtcars
```

As we discussed , we need to standardize the data before we can begin clustering if we dont want inherrent weightages present in the data to prevail.

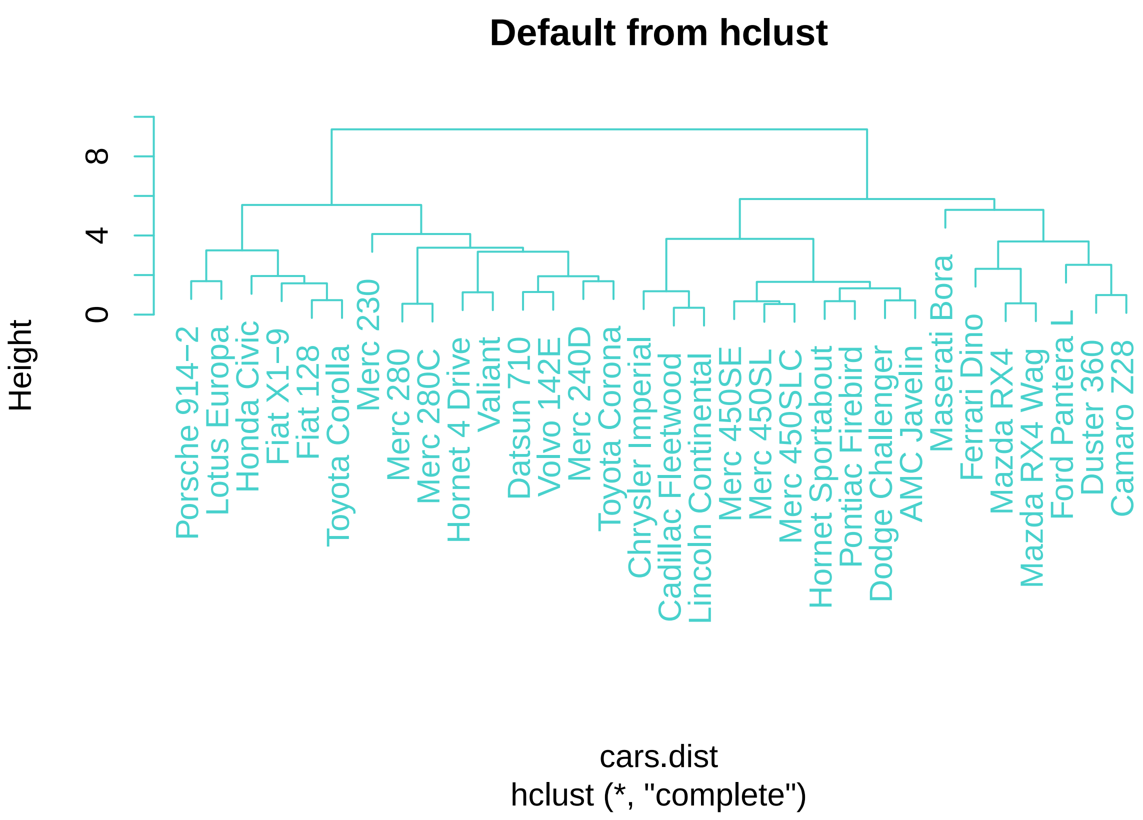
```
medians = apply(cars,2,median)
mads = apply(cars,2,mad)
cars.std=data.frame(scale(cars,center=medians,scale=mads))
```

Next we will create a distance matrix for all the observations in the data. this is the resource consuming step which becomes insanley slow as the size of the data increases.

```
cars.dist = dist(cars.std)
```

Using this distance matrix we can now built a hierarchical clustering model with function `hclust`.

```
cars.hclust = hclust(cars.dist)
plot(cars.hclust,labels=rownames(cars),main='Default from hclust',col="mediumturquoise")
```



You can various information regarding each group of the cluster using function `cutree`. Here is an example.

```
groups.3=cutree(cars.hclust,3)

groups.3
```

```
##           Mazda RX4       Mazda RX4 Wag          Datsun 710
##                   1                   1                   2
##      Hornet 4 Drive   Hornet Sportabout             Valiant
##                   2                   3                   2
##          Duster 360           Merc 240D            Merc 230
##                   1                   2                   2
##            Merc 280           Merc 280C          Merc 450SE
##                   2                   2                   3
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood
##                   3                   3                   3
## Lincoln Continental   Chrysler Imperial            Fiat 128
##                   3                   3                   2
##         Honda Civic      Toyota Corolla       Toyota Corona
##                   2                   2                   2
##    Dodge Challenger         AMC Javelin          Camaro Z28
##                   3                   3                   1
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2
##                   3                   2                   2
##        Lotus Europa      Ford Pantera L        Ferrari Dino
##                   2                   1                   1
##       Maserati Bora          Volvo 142E
##                   1                   2
```

```
table(groups.3)
```

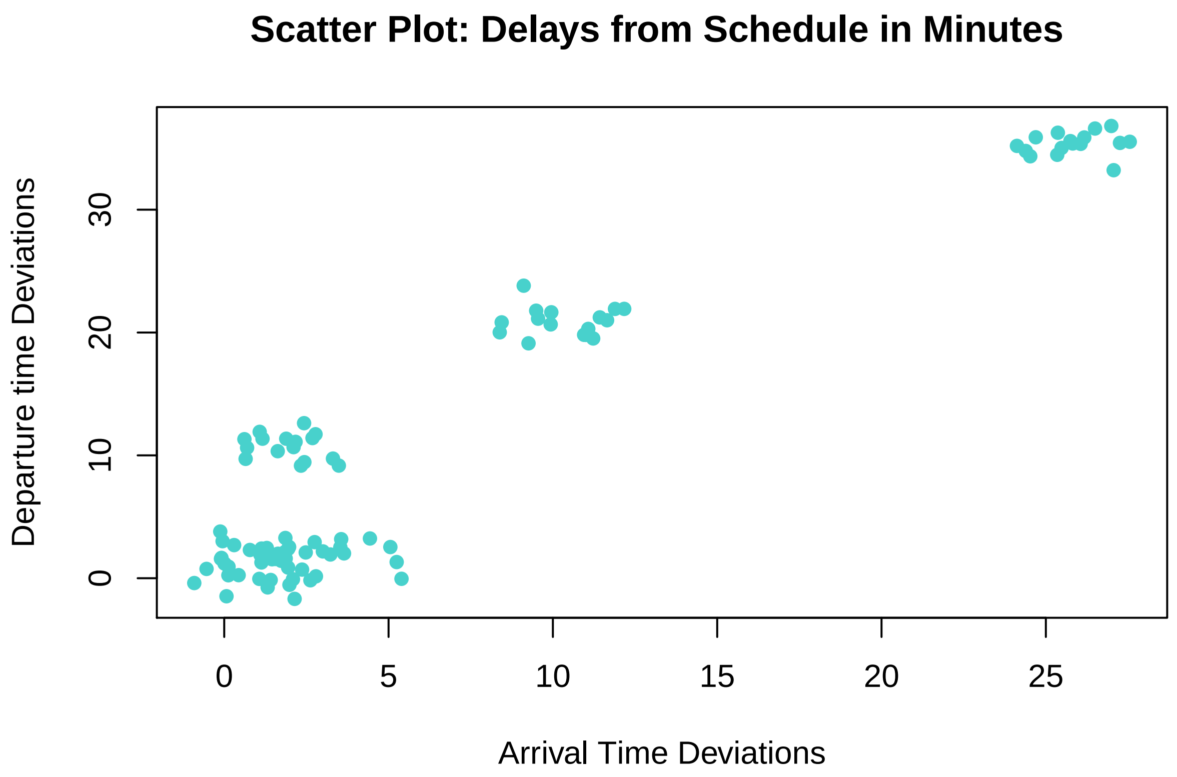
```
## groups.3
##  1  2  3
##  7 15 10
```

```
rownames(cars)[groups.3==1]
```

```
## [1] "Mazda RX4"      "Mazda RX4 Wag"  "Duster 360"     "Camaro Z28"
## [5] "Ford Pantera L" "Ferrari Dino"   "Maserati Bora"
```

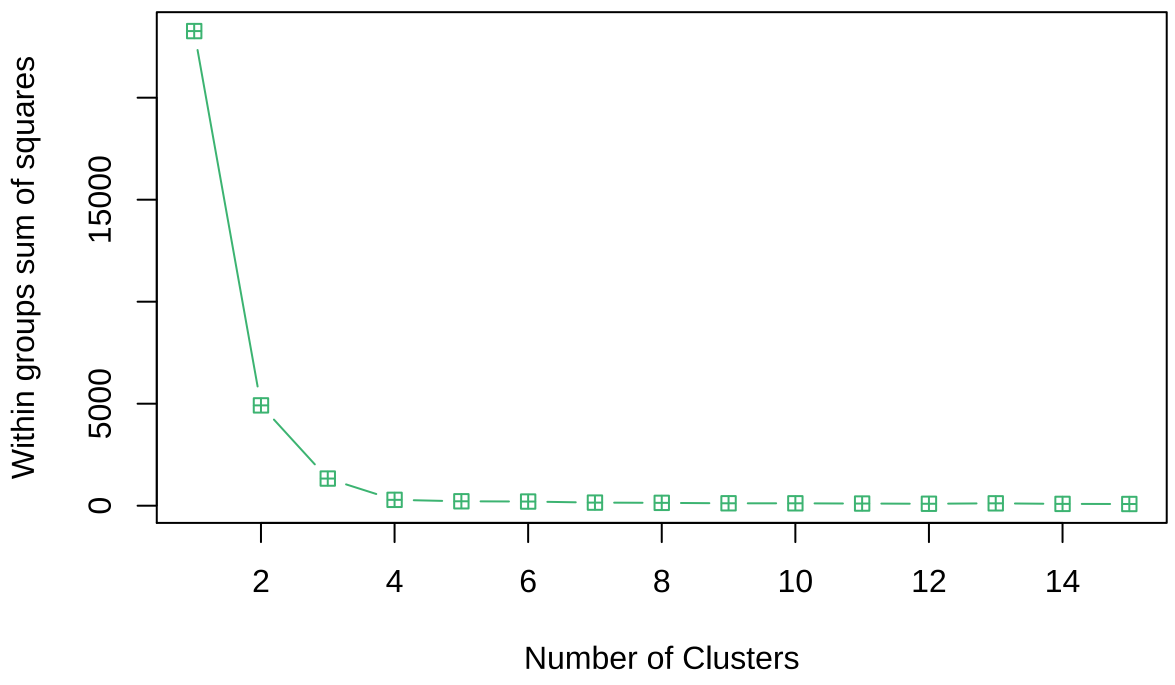
Lets prepare data for K-means clustering with natural groups present in the data. We'll later run K-mean algorithm to see whether it can identify those groups or not.

```
n = 100
g = 6
set.seed(g)
d <- data.frame(
  x = unlist(lapply(1:g, function(i) rnorm(n/g, runif(1)*i^2))),
  y = unlist(lapply(1:g, function(i) rnorm(n/g, runif(1)*i^2))))
plot(d,col="mediumturquoise",pch=16,
     xlab="Arrival Time Deviations",
     ylab="Departure time Deviations",
     main="Scatter Plot: Delays from Schedule in Minutes ")
```



We are not going to standardize this data and let the inherrent weightages prevail. [Also, all the variables are on same scale.]. Lets run kmeans for various values of K and record wss everytime.

```r
mydata <- d

wss <- (nrow(mydata)-1)*sum(apply(mydata,2,var))

for (i in 2:15) wss[i] <- sum(kmeans(mydata,centers=i)$withinss)

plot(1:15, wss, type="b", xlab="Number of Clusters",
     ylab="Within groups sum of squares",col="mediumseagreen",pch=12)
```



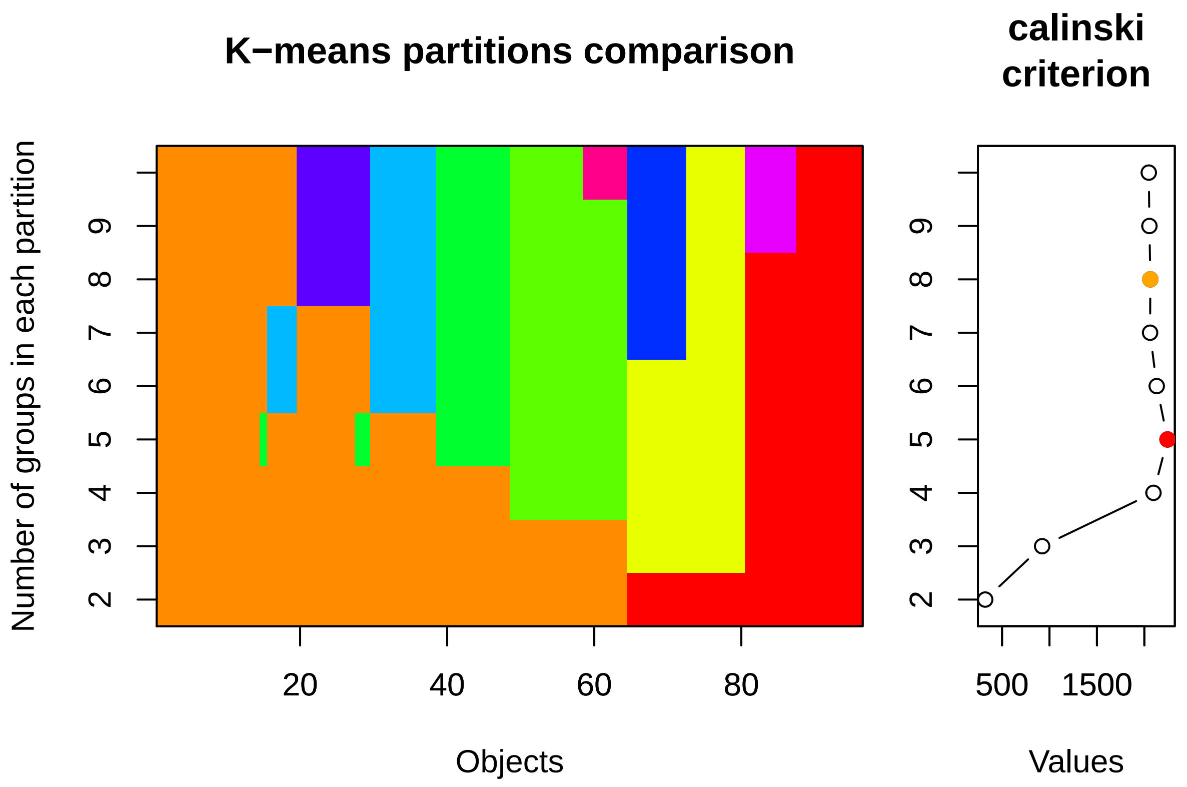
This tells us that there might be 4 groups present in the data. Although if you go back ,try and look closely on the data plot, there might be more group possible. How groups you want to break your data into , is eventually your call as we discussed.

There is another way of deciding the value of K by calinski criterion. We'll discuss maths of the method.

```r
require(vegan)
```

```
## Loading required package: vegan
## Loading required package: permute
## Loading required package: lattice
## This is vegan 2.2-1
```

```r
fit <- cascadeKM(scale(d, center = TRUE,  scale = TRUE), 1, 10, iter = 1000)
plot(fit, sortg = TRUE, grpmts.plot = TRUE)
```



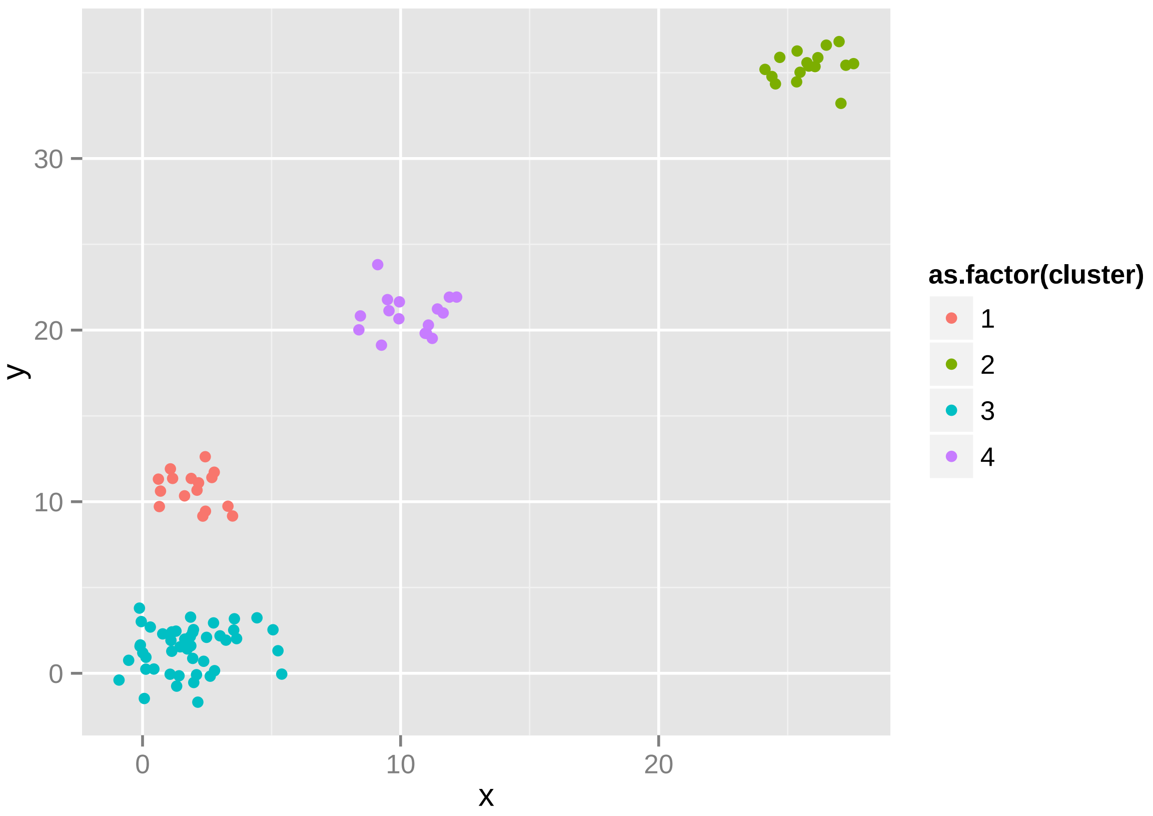
```
calinski.best <- as.numeric(which.max(fit$results[2,]))
cat("Calinski criterion optimal number of clusters:", calinski.best, "\n")
```

```
## Calinski criterion optimal number of clusters: 5
```

This gives best choice is 5. Again this is a subjective call.

For profiling you can add the cluster membership back to the data and plot pair wise 2d plots, colored by cluster membership. You can look at numerical summarise cluster wise as well.

```
fit <- kmeans(mydata,4 )

d$cluster=fit$cluster

library(ggplot2)

ggplot(d,aes(x,y,color=as.factor(cluster)))+geom_point()
```

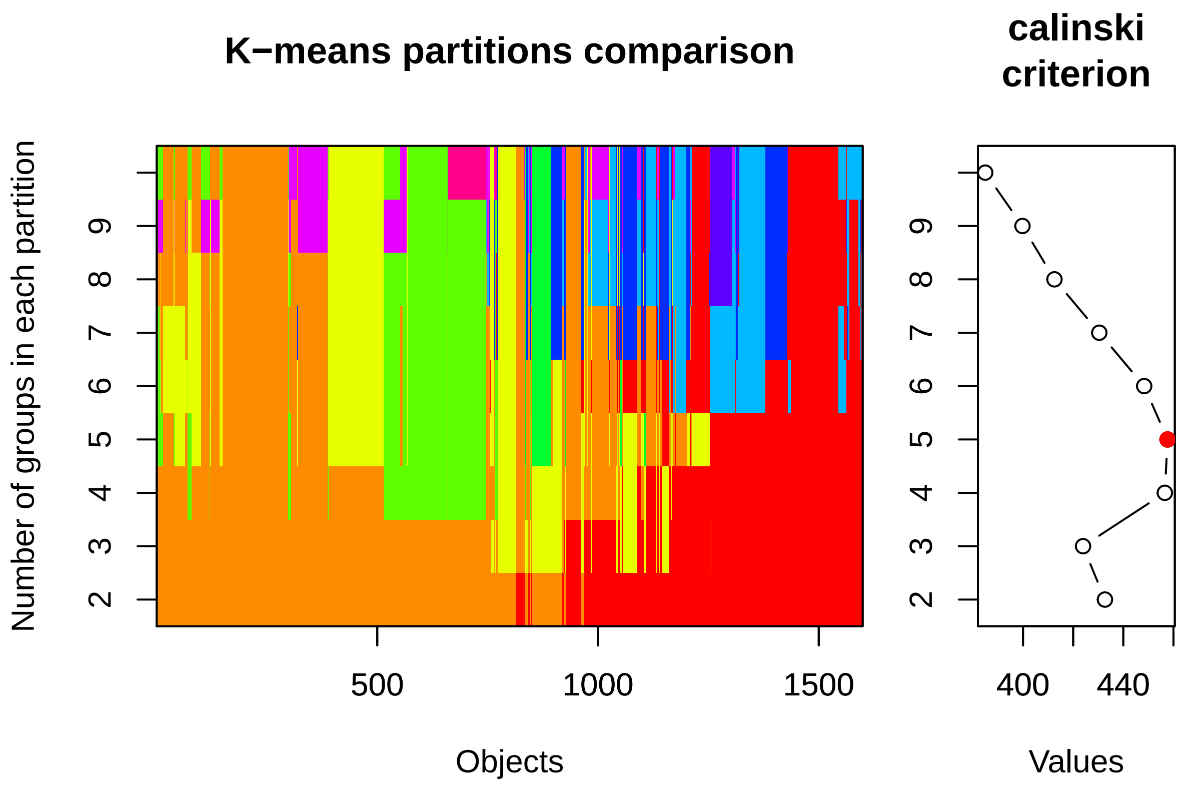


Lets do this for another dataset redwines. We have already selected 4 variable which we'll be using for clustering out of many other present in the data.`ph`, `sulphates`, `alcohol`, `total.sulfur.dioxide`

```
wine=read.csv("winequality-red.csv",sep=";")
wine=wine %>%
  select(pH,sulphates,alcohol,total.sulfur.dioxide)
```

Lets standardize this data

```
md=function(x){
  return((x-mean(x))/sd(x))
}
wine_std=wine %>%
  mutate(pH=md(pH),
         sulphates=md(sulphates),
         alcohol=md(alcohol),
         total.sulfur.dioxide=md(total.sulfur.dioxide))

require(vegan)
fit <- cascadeKM(wine_std, 1, 10, iter = 100)
plot(fit, sortg = TRUE, grpmts.plot = TRUE)
```

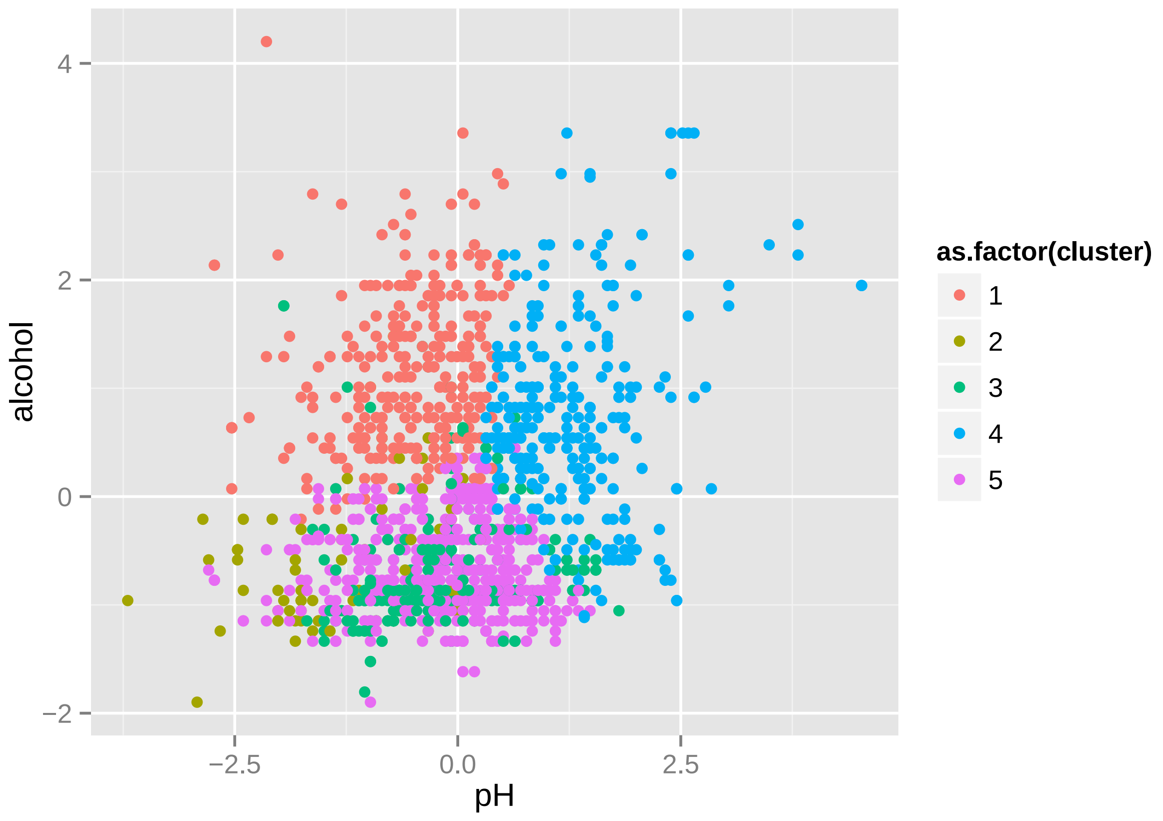


```
calinski.best <- as.numeric(which.max(fit$results[2,]))
cat("Calinski criterion optimal number of clusters:", calinski.best, "\n")
```

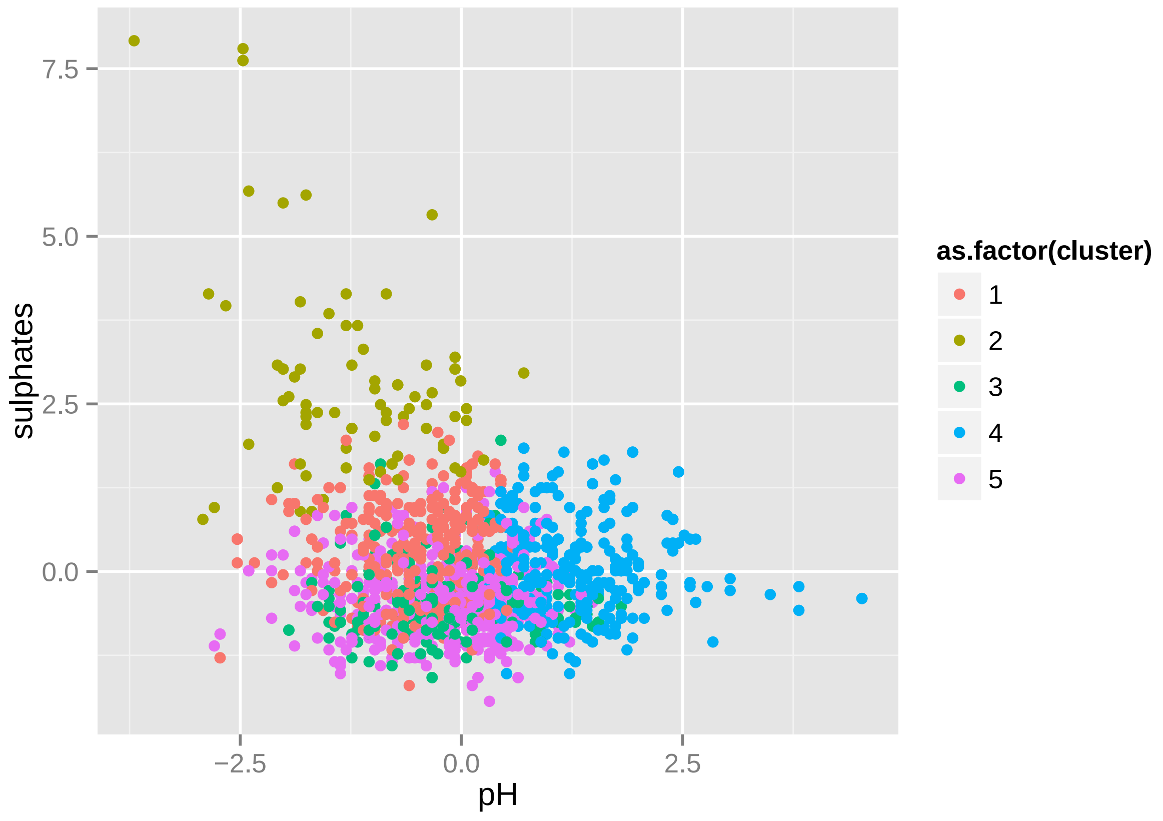
```
## Calinski criterion optimal number of clusters: 5
```

We'll make 5 groups.

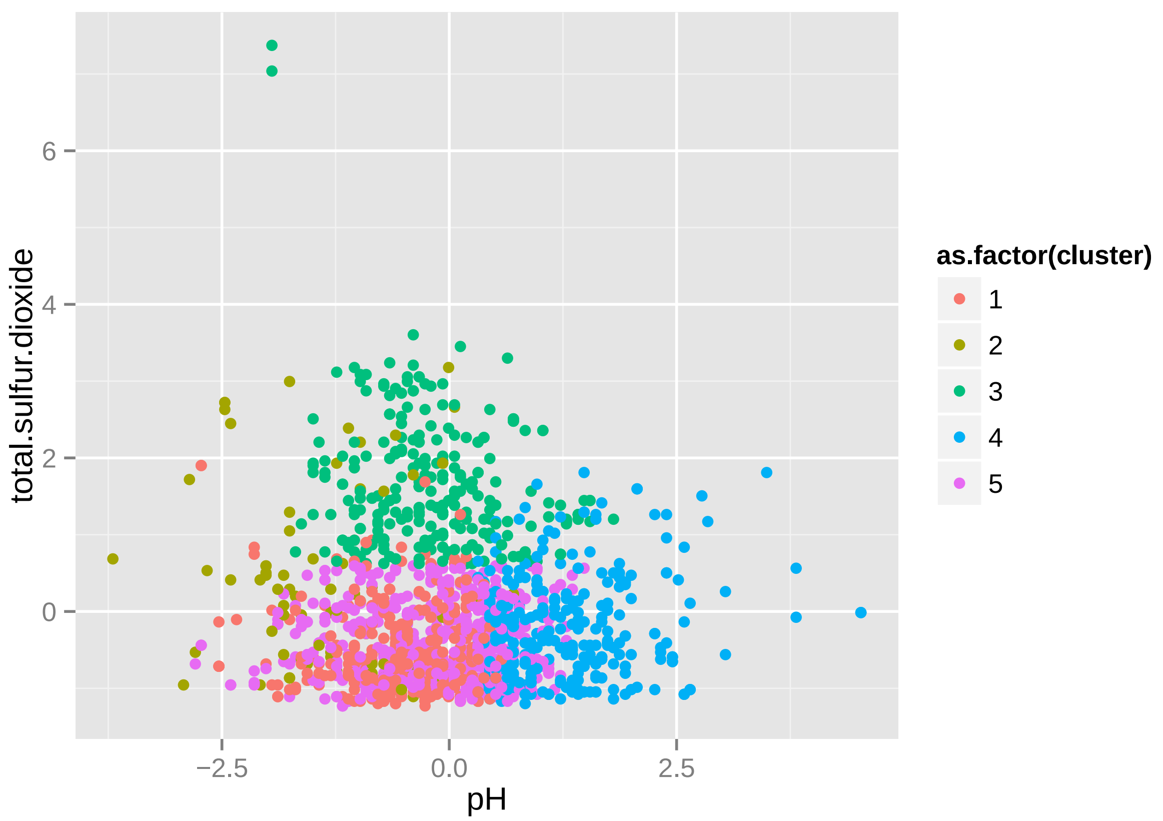
```
fit <- kmeans(wine_std,5)

wine_std$cluster=fit$cluster

library(ggplot2)

#pair wise profiling plots
ggplot(wine_std,aes(pH,alcohol,color=as.factor(cluster)))+geom_point()
```



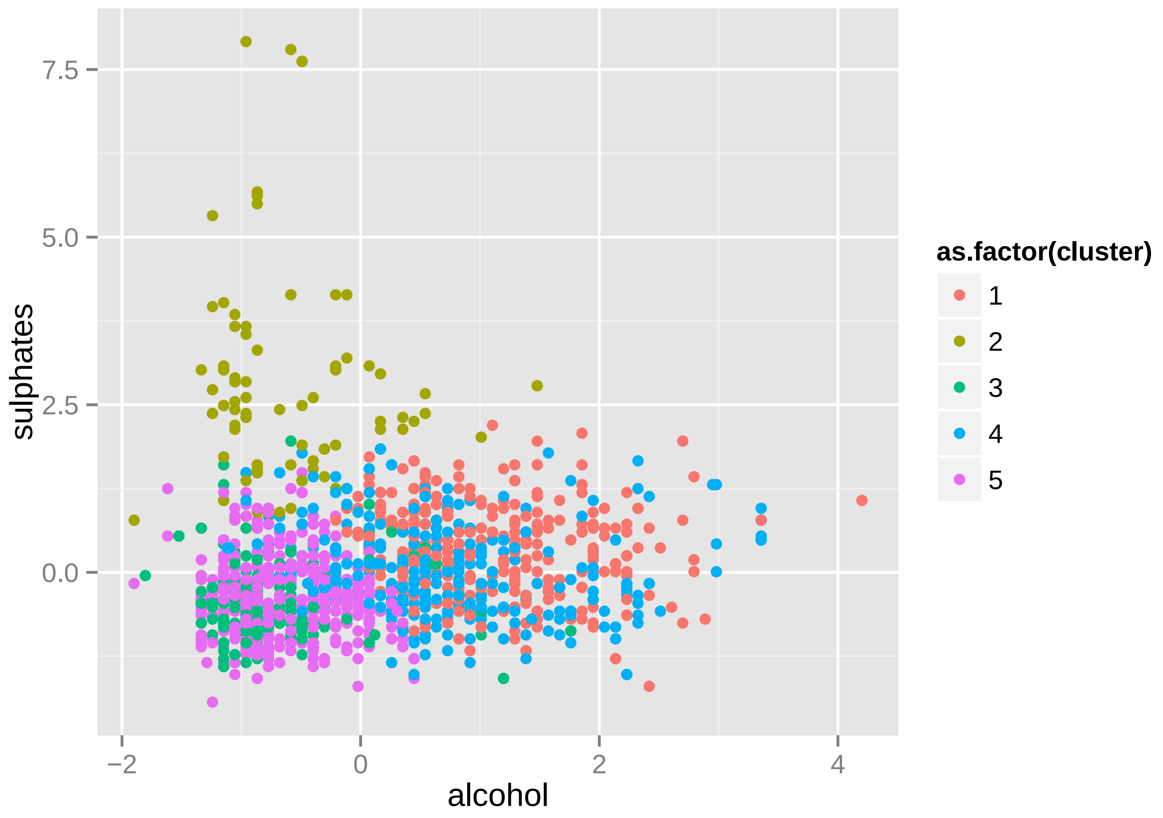
```
ggplot(wine_std,aes(pH,sulphates,color=as.factor(cluster)))+geom_point()
```



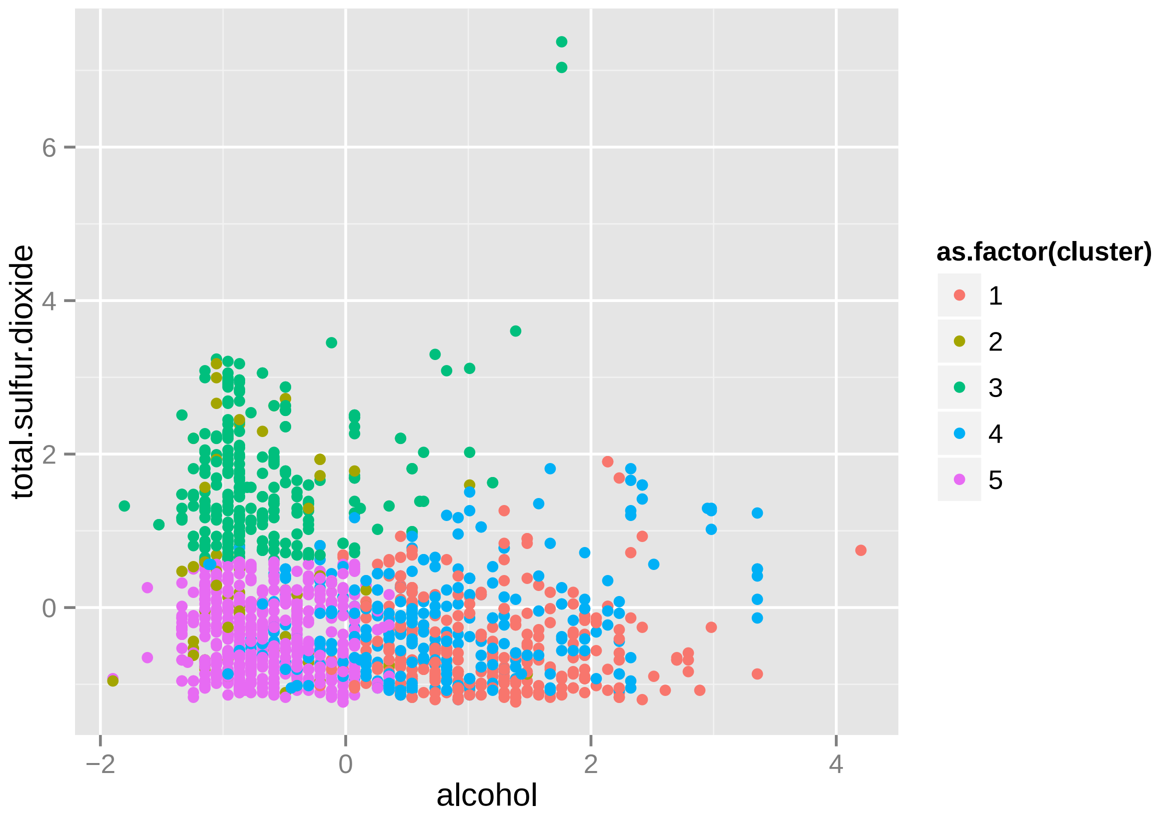
```
ggplot(wine_std,aes(pH,total.sulfur.dioxide,color=as.factor(cluster)))+geom_point()
```



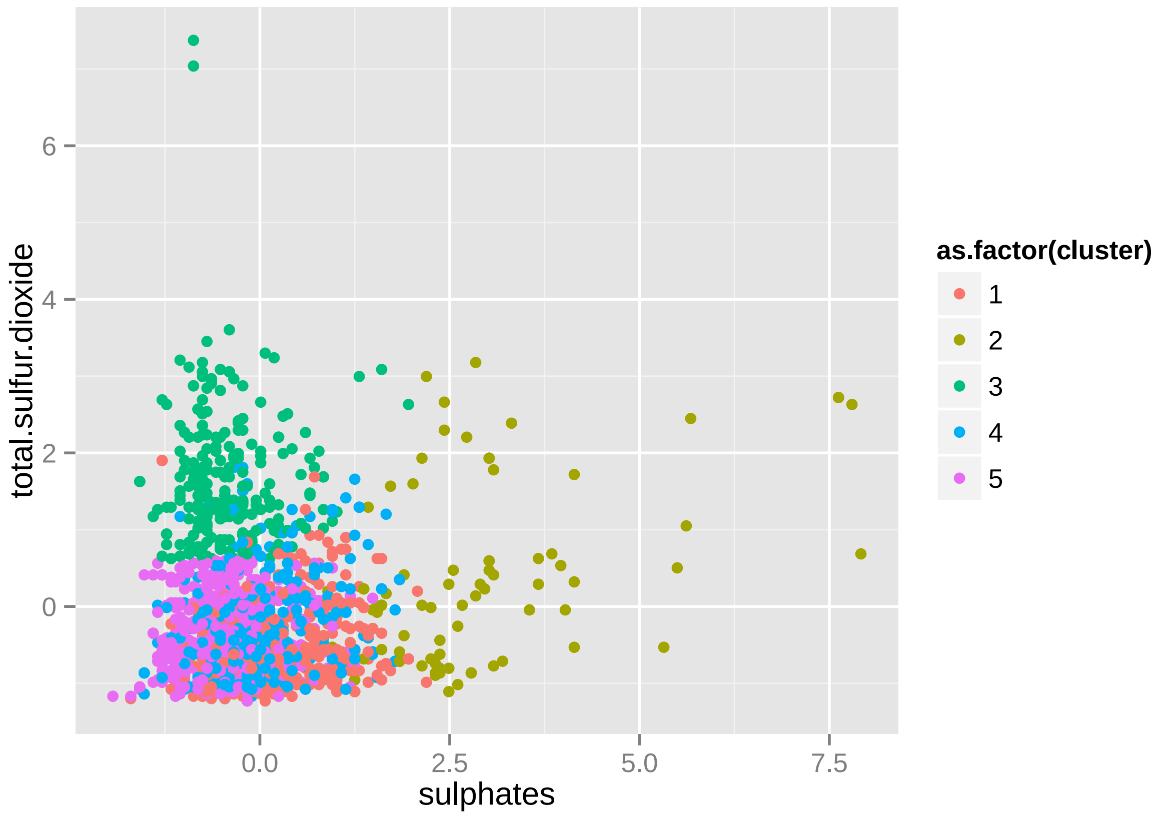
```
ggplot(wine_std,aes(alcohol,sulphates,color=as.factor(cluster)))+geom_point()
```



```
ggplot(wine_std,aes(alcohol,total.sulfur.dioxide,color=as.factor(cluster)))+geom_point()
```



```r
ggplot(wine_std,aes(sulphates,total.sulfur.dioxide,color=as.factor(cluster)))+geom_point()
```



As you can see, once you have increased dimnesion, there are going to be overlap in 2D plots. However in some plots , you'll see better differentiation between certain groups, you can label them accordingly. For example group wih high total.sulfur.dioxide [or if you can associate that with some property of the wine , say strong scent.]

We'll leave numerical summary profiling to you. It can be done with methods discussed in univariate statistics module. We'll conclude here.

Prepared By : Lalit Sachan

Contact : lalit.sachan@edvancer.in