# R Notebook

**Hierarchical Clustering**

organize your data in a hiearchical order

- find closest two things
- put them together
- find next closest

produces a tree showing how close things

- Defining Closeness
- distance or similarity
  - continuous euclidean space
  - continuous correlation similarity
  - binary - Manhattan distance

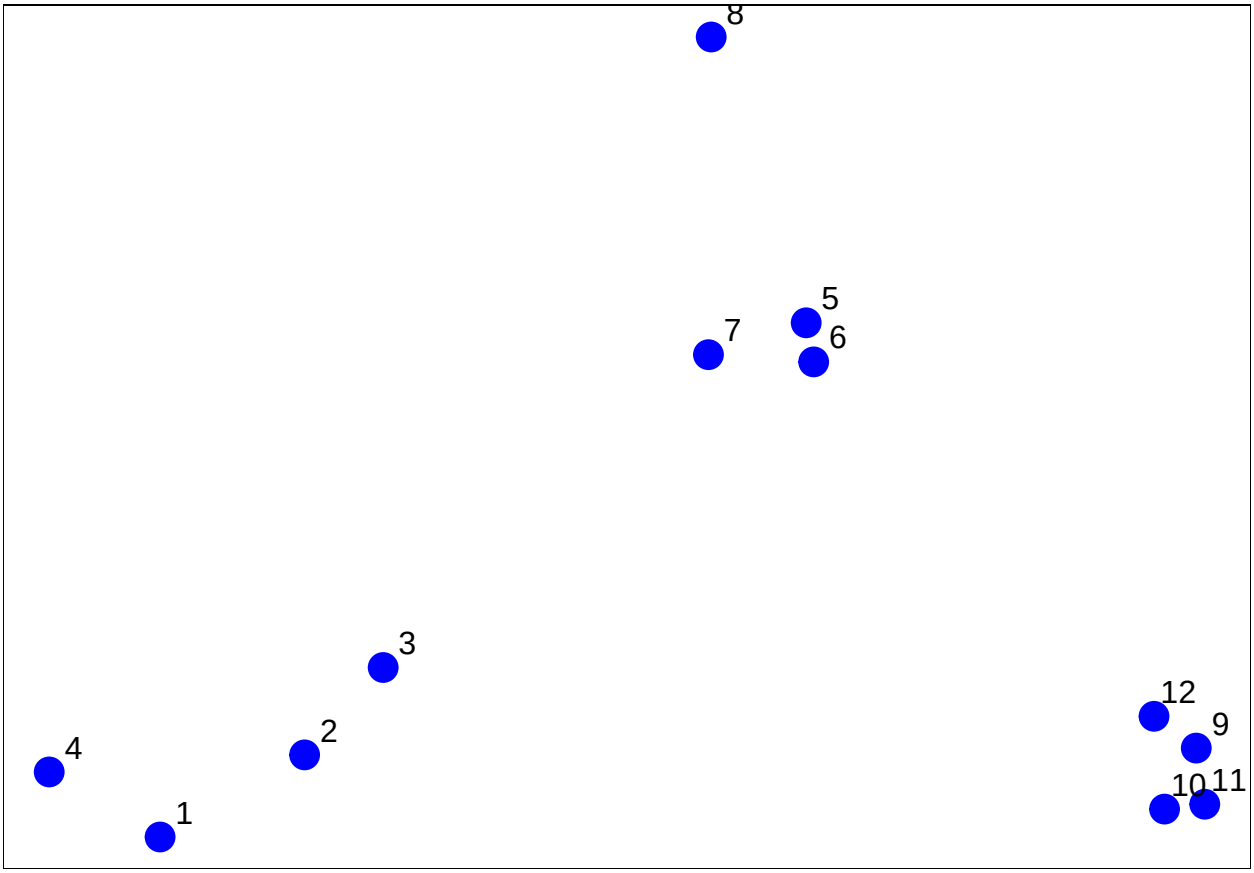
Euclidean Distance- fly over distance

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

Manhattan distance:

Considering roads in a city when you calculate a distance

```
set.seed(1234)
par(mar = c(0,0,0,0))
x <- rnorm(12, mean = rep(1:3, each = 4), sd = 0.2)
y <- rnorm(12, mean = rep(c(1, 2, 1), each = 4), sd = 0.2)
plot(x, y, col = "blue", pch =19, cex = 2)
text(x + 0.05, y + 0.05, labels = as.character(1:12))
```




## Calculating distance

This function computes and returns the distance matrix computed by using the specified distance measure to compute the distances between the rows of a data matrix.

```
str(dist)
```

```
## function (x, method = "euclidean", diag = FALSE, upper = FALSE, p = 2)
```

```
df <- data.frame(x = x, y=y)
dist(df)
```

```
##            1          2          3          4          5          6          7
## 2  0.34120511
## 3  0.57493739 0.24102750
## 4  0.26381786 0.52578819 0.71861759
## 5  1.69424700 1.35818182 1.11952883 1.80666768
## 6  1.65812902 1.31960442 1.08338841 1.78081321 0.08150268
## 7  1.49823399 1.16620981 0.92568723 1.60131659 0.21110433 0.21666557
## 8  1.99149025 1.69093111 1.45648906 2.02849490 0.61704200 0.69791931 0.65062566
## 9  2.13629539 1.83167669 1.67835968 2.35675598 1.18349654 1.11500116 1.28582631
## 10 2.06419586 1.76999236 1.63109790 2.29239480 1.23847877 1.16550201 1.32063059
## 11 2.14702468 1.85183204 1.71074417 2.37461984 1.28153948 1.21077373 1.37369662
## 12 2.05664233 1.74662555 1.58658782 2.27232243 1.07700974 1.00777231 1.17740375
##            8          9         10         11
## 2
## 3
```

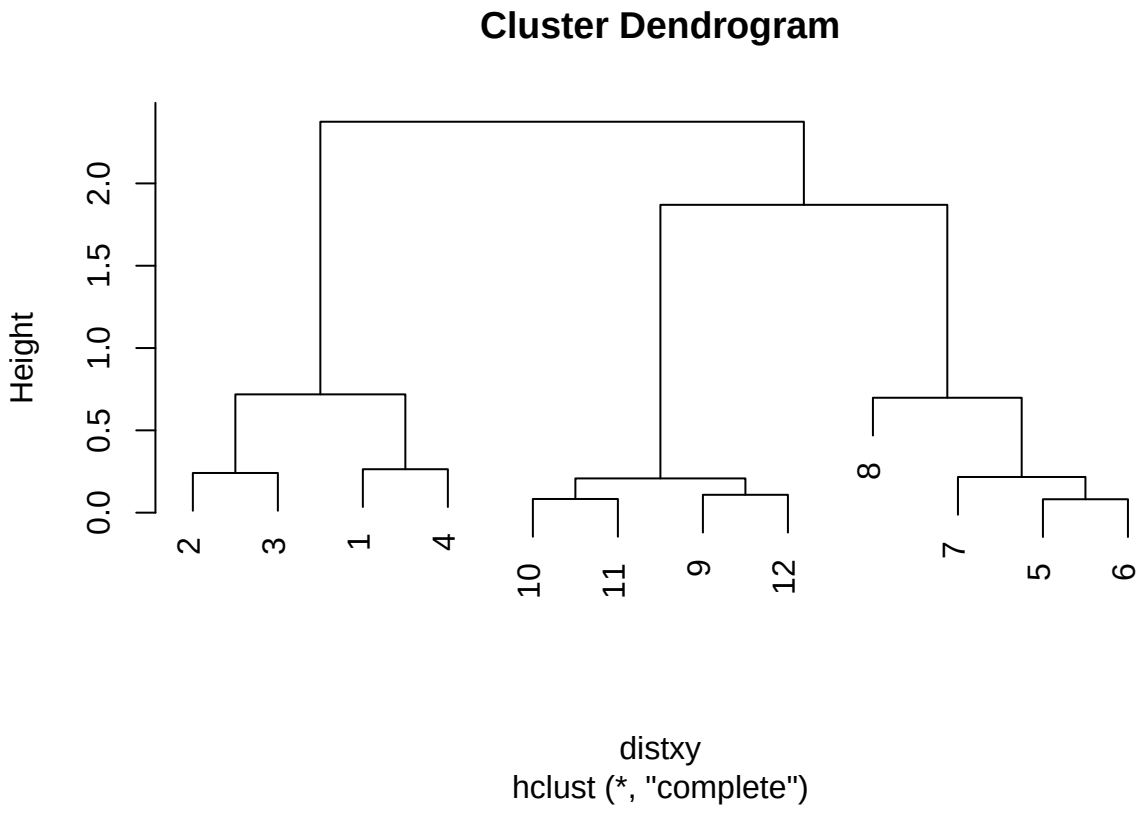
```
## 4
## 5
## 6
## 7
## 8
## 9  1.76460709
## 10 1.83517785 0.14090406
## 11 1.86999431 0.11624471 0.08317570
## 12 1.66223814 0.10848966 0.19128645 0.20802789
```

```
str(hclust)
```

```
## function (d, method = "complete", members = NULL)
```

Hierarchical cluster analysis on a set of dissimilarities and methods for analyzing it.

```
distxy <- dist(df)
hClustering <- hclust(distxy)
plot(hClustering)
```

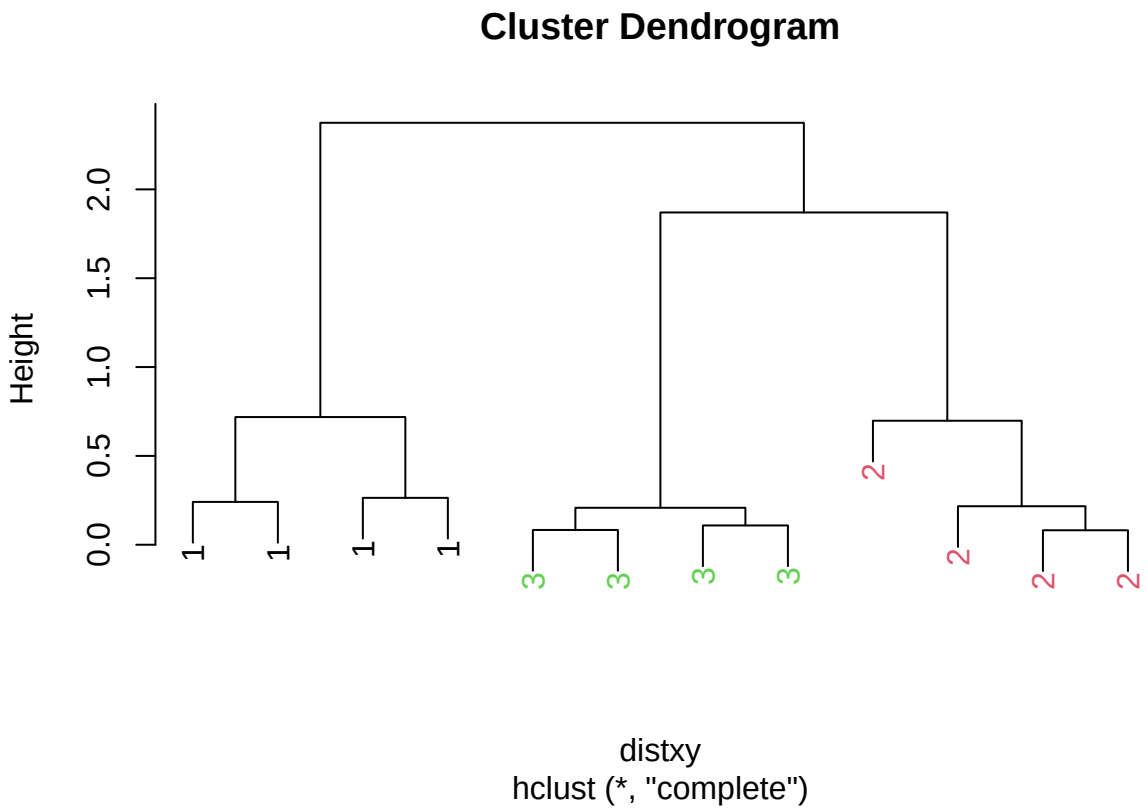



```
myplclust <- function(hclust, lab = hclust$labels,
                      lab.col = rep(1, length(hclust$labels)),
                      hang = 0.1, ...) {
# modification of plclust for plotting hclust objects *in colour*! copyright
# Eva KF Chan 2009 Arguments: hclust: object lab: a character vector
# of labels of the leaves of the tree lab.col: colour for the labels;
# NA = default device foreground colour hang: as in hclust & plclust Side
# effect: A display of hierarchical cluster with coloured leaf labels.
```

```
    y <- rep(hclust$height, 2)
    x <- as.numeric(hclust$merge)
    y <- y[which(x < 0)]
    x <- x[which(x < 0)]
    x <- abs(x)
    y <- y[order(x)]
    x <- x[order(x)]
    plot(hclust, labels = FALSE, hang = hang, ...)
    text(x = x, y = y[hclust$order] - (max(hclust$height) * hang),
         labels = lab[hclust$order], col = lab.col[hclust$order],
         srt = 90, adj = c(1, 0.5), xpd = NA, ...)
    }
```

You have to know the number of clusters before yoou draw the dendrogram

```
myplclust(hClustering, lab = rep(1:3, each = 4),
          lab.col = rep(1:3, each = 4))
```




In R graph Gallery, you can find more graph options.

## merging the points

what represents the new locations? 1. you can merge taking average of axis, distances 2. complete linkage both can result differently.

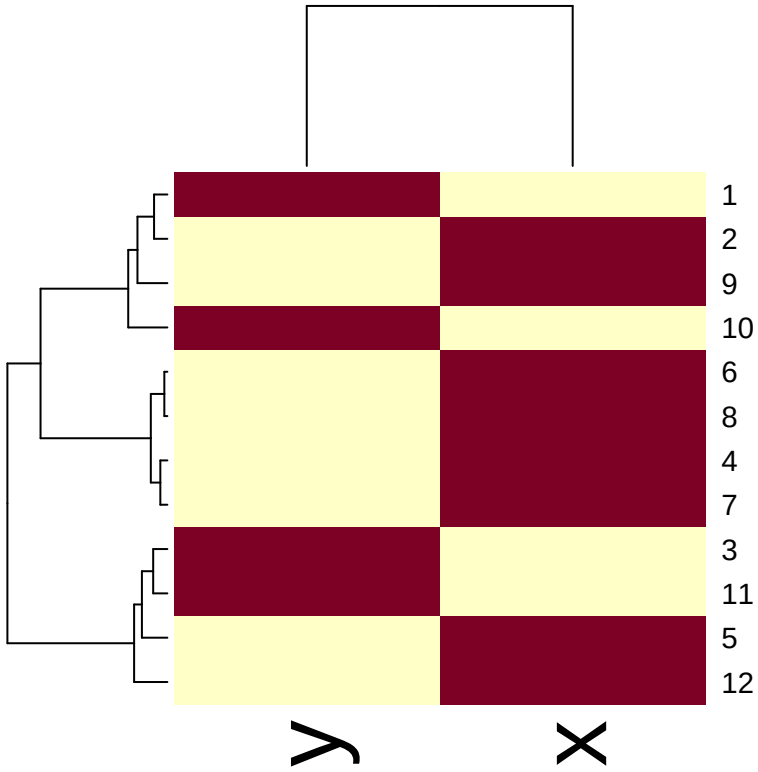
### Heatmap() function

the Idea here is that, heatmap function uses clusters

reorder the columns and row according to the groups

```
x <- rnorm(12, mean = rep(1:3, each = 4), sd = 0.2)
y <- rnorm(12, mean = rep(c(1, 2, 1), each = 4), sd = 0.2)
df <- data.frame(x = x, y = y)

set.seed(143)
dataMatrix <- as.matrix(df)[sample(1:12), ]
heatmap(dataMatrix)
```




### Summary

- Gives an idea of the relationship between variables/observations
- the picture may be unstable
- Should be primarily used for exploration
- Some variables may need to be rescaled

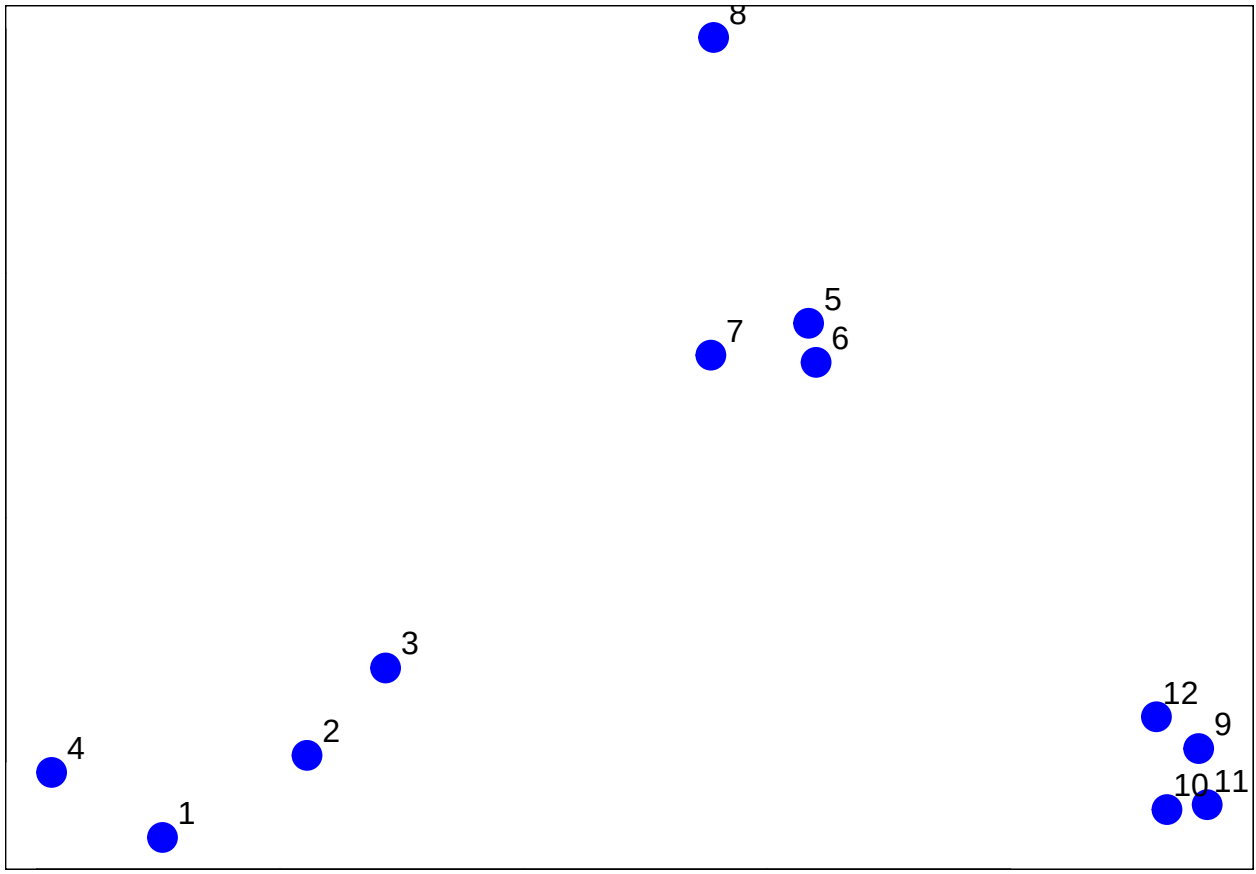
### K-Means Clustering

- what does it means to be close?

```
set.seed(1234)
par(mar = c(0,0,0,0))
```

```
x <- rnorm(12, mean = rep(1:3, each = 4), sd = 0.2)
y <- rnorm(12, mean = rep(c(1, 2, 1), each = 4), sd = 0.2)

plot(x, y, col = "blue", pch = 19, cex = 2)
text(x + 0.05, y + 0.05, labels = as.character(1:12))
```




**An Example**

### Kmeans()

You need to specify number of clusters - pick by eye/intuition - pick by cross validation/information theory - determine the number of clusters

K-means is not deterministic - different # of clusters - different number of iterations

```
df <- data.frame(x, y)
kmeansobj <- kmeans(df, centers = 3)
names(kmeansobj)
```

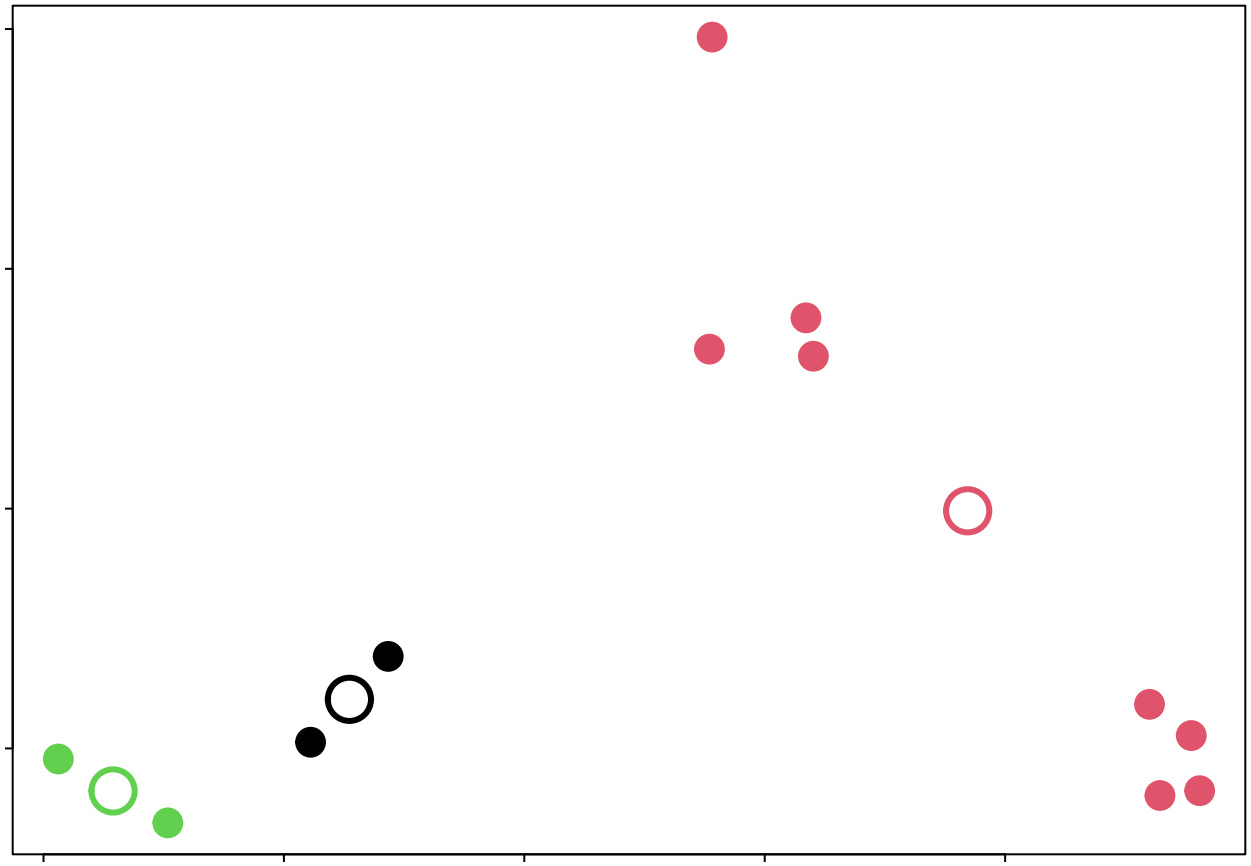
```
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
kmeansobj$cluster
```

```
##  [1] 3 1 1 3 2 2 2 2 2 2 2 2
```

```
par(mar = rep(0.2, 4))

plot(x, y, col = kmeansobj$cluster, pch = 19, cex = 2)
points(kmeansobj$centers, col = 1:3, cex = 3, lwd = 3)
```



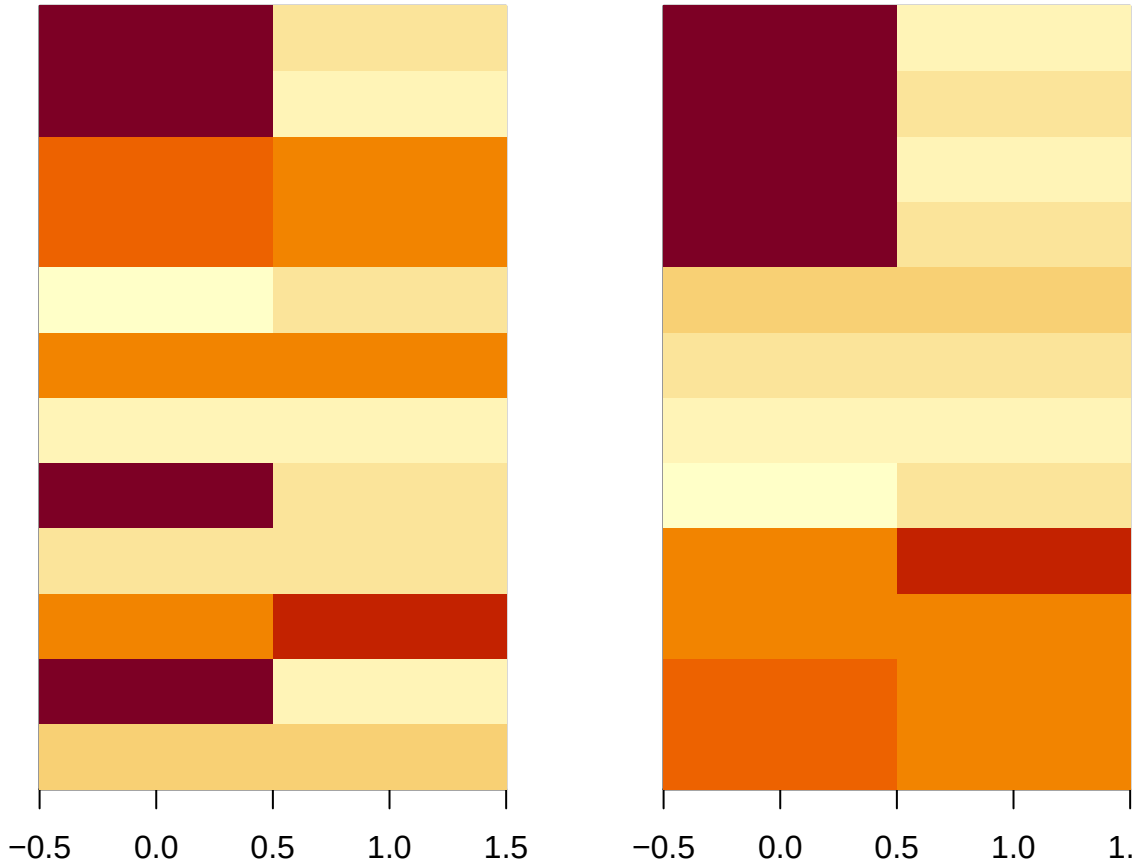
**Heatmap()**

Another way to see clusters

```
set.seed(1234)
dataMatrix <- as.matrix(df)[sample(1:12), ]

kmeansobj <- kmeans(dataMatrix, centers = 3)
par(mfrow = c(1, 2), mar = c(2, 4, 0.1, 0.1))

image(t(dataMatrix)[, nrow(dataMatrix):1], yaxt = "n")
image(t(dataMatrix)[, order(kmeansobj$cluster)], yaxt = "n")
```



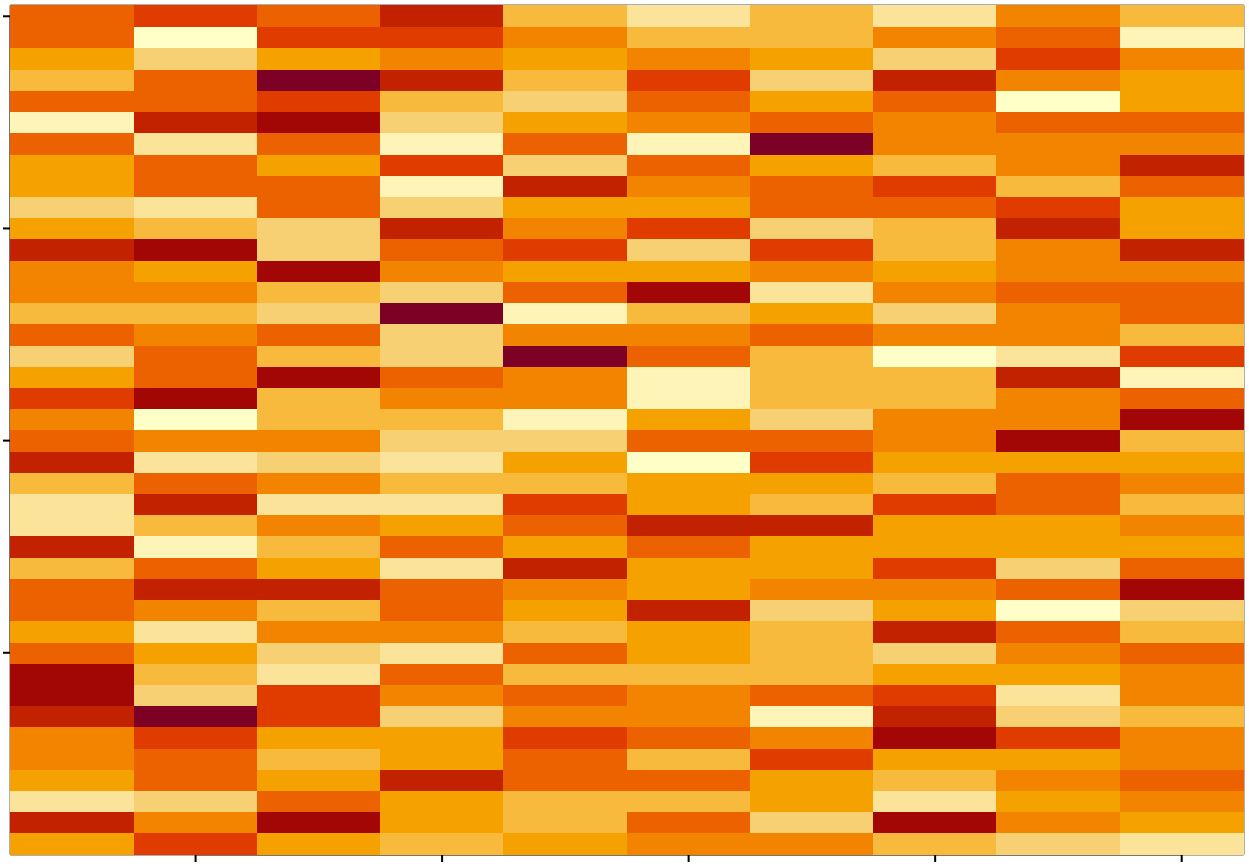

## Summary

- K-means requires a number of clusters
  - pick by intuition
  - pick by cross validation
- K-means is not deterministic
  - Different number of clusters
  - different number of iterations

### Principal Components Analysis

lets start with a random generated matrix

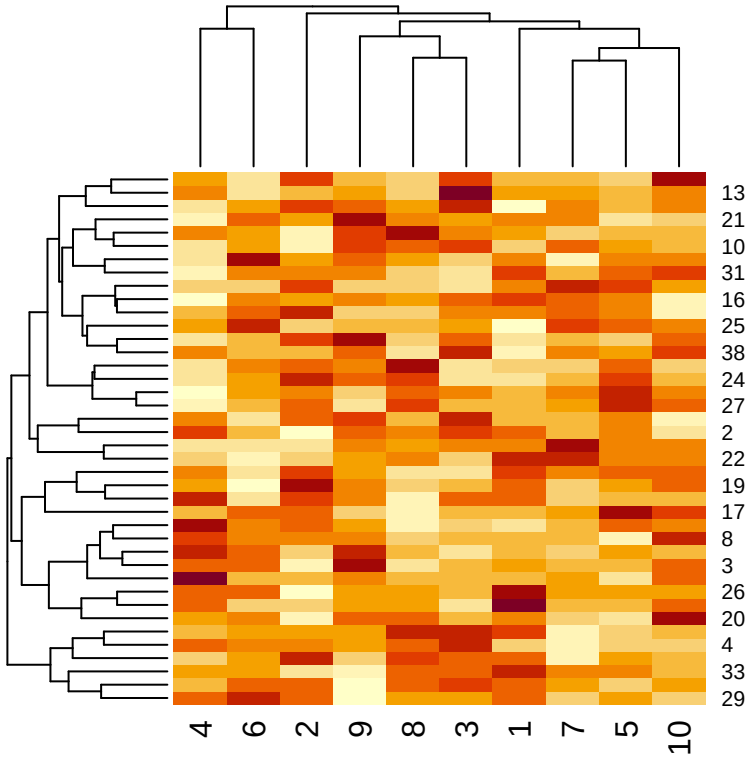
```
set.seed(12345)
par(mar = rep(0.2, 4))
dm <- matrix(rnorm(400), nrow = 40)
image(1:10, 1:40, t(dm)[, nrow(dm):1])
```



There is so much noise above.

## Cluster the data

```
par(mar = rep(0.2, 4))
heatmap(dm)
```

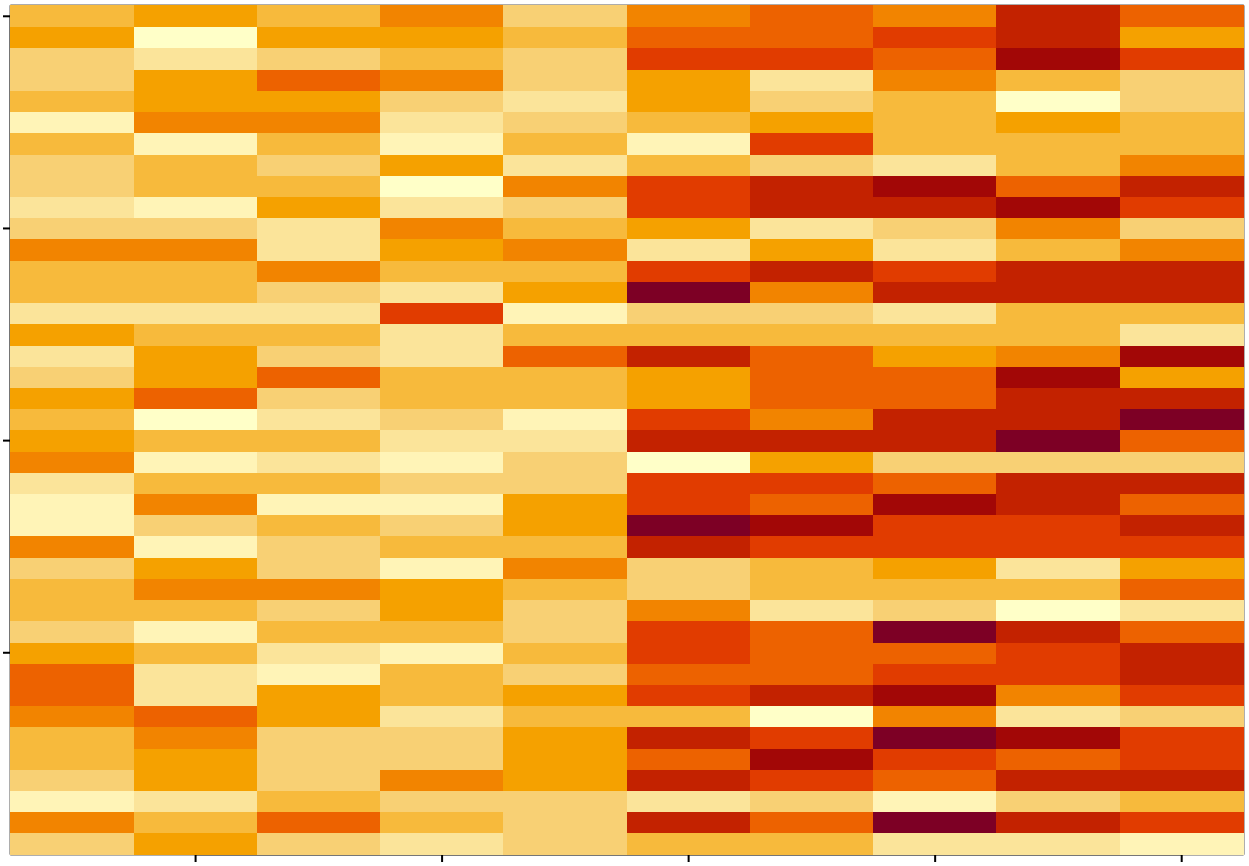



No obvious pattern can be seen above

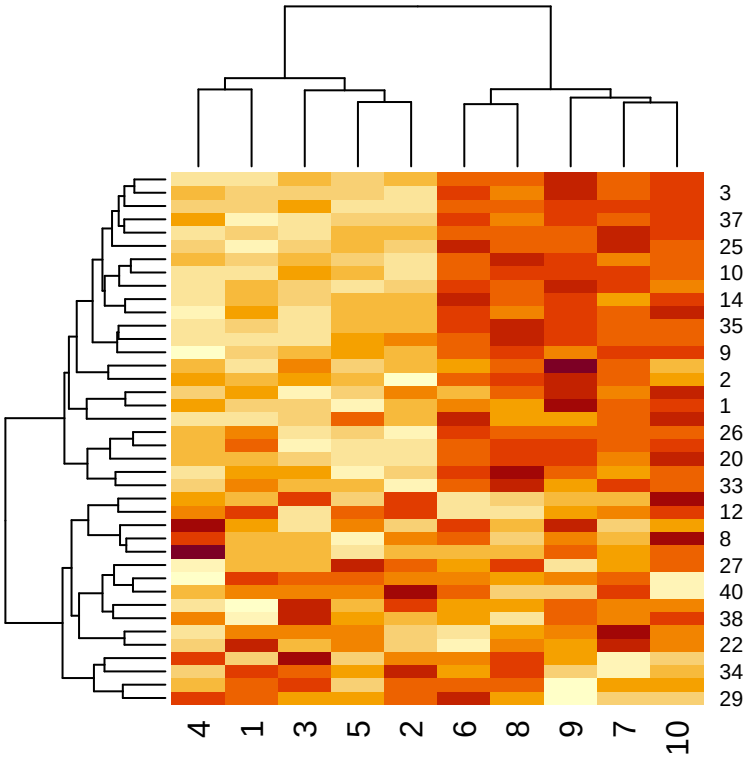
### What if we add a pattern?

```
set.seed(678910)
for (i in 1:40) {
    # flip a cooin
    coinflip <- rbinom(1, size =1, prob = 0.5)
    # if a coin is heads add acommon pattern to that row
    if (coinflip) {
        dm[i, ] <- dm[i, ] + rep(c(0, 3), each = 5)
    }
}

par(mar = rep(0.2, 4))
image(1:10, 1:40, t(dm)[, nrow(dm):1])
```



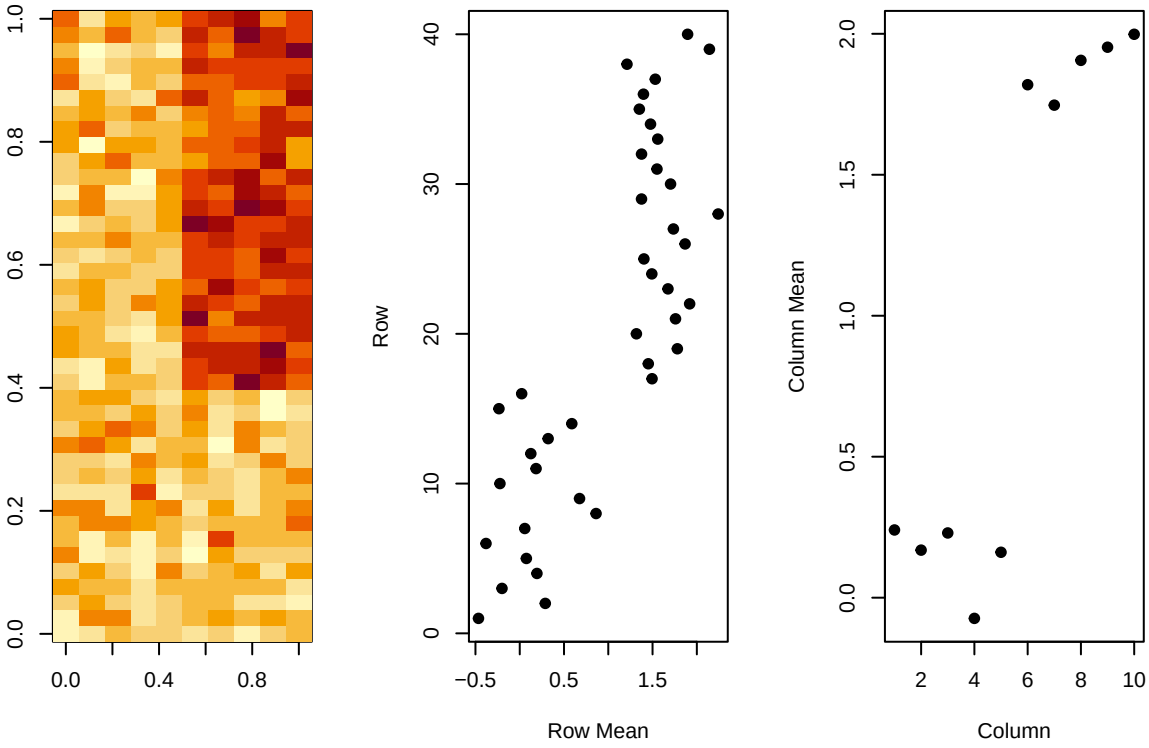
```
par(mar = rep(0.2, 4))
heatmap(dm)
```




### Patterns in row and column

```
hh <- hclust(dist(dm))
dmordered <- dm[hh$order, ]
par(mfrow = c(1, 3))
image(t(dmordered)[, nrow(dmordered):1])

plot(rowMeans(dmordered), 40:1, xlab = "Row Mean", ylab = "Row", pch = 19)
plot(colMeans(dmordered), xlab = "Column", ylab = "Column Mean", pch=19)
```




**Related problems**

You have multivariate variables

Find a new set of multivariate variables that are uncorrelated and explain as much variance as possible

If you put all the variables together in one matrix, find the best matrix created with fewer variables

First goal statistical, then data compression

**PCA/SVD**

**SVD** Let X is a matrix with each variable in a columns, each obs is in a row.

Then SVD is a matrix decomposition

$$X = UVD^T$$

where the columns of U are orthogonal (left singular), the columns of D are orthogonal (right singular) and D is a diagonal matrix (singular values).

**PCA** the principal components are equal to the right singular values if you first scale (substract the mean, divide by the standard deviation) the variables.

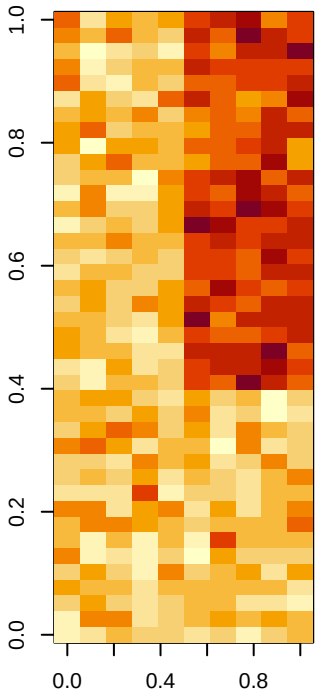
## Dimension Reduction

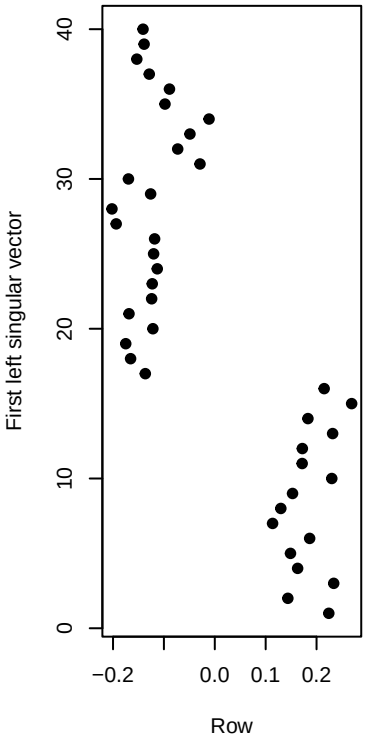
```
svd1 <- svd(scale(dmordered))
```

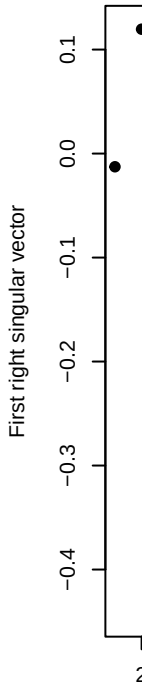
```
par(mfrow = c(1, 3))

image(t(dmordered)[, nrow(dmordered):1])

plot(svd1$u[, 1], 40:1, xlab = "Row", ylab = "First left singular vector", pch =19)
plot(svd1$v[, 1], xlab = "Column", ylab = "First right singular vector", pch = 19)
```




**Components of the SVD -u and -v**