Hierarchical Clustering

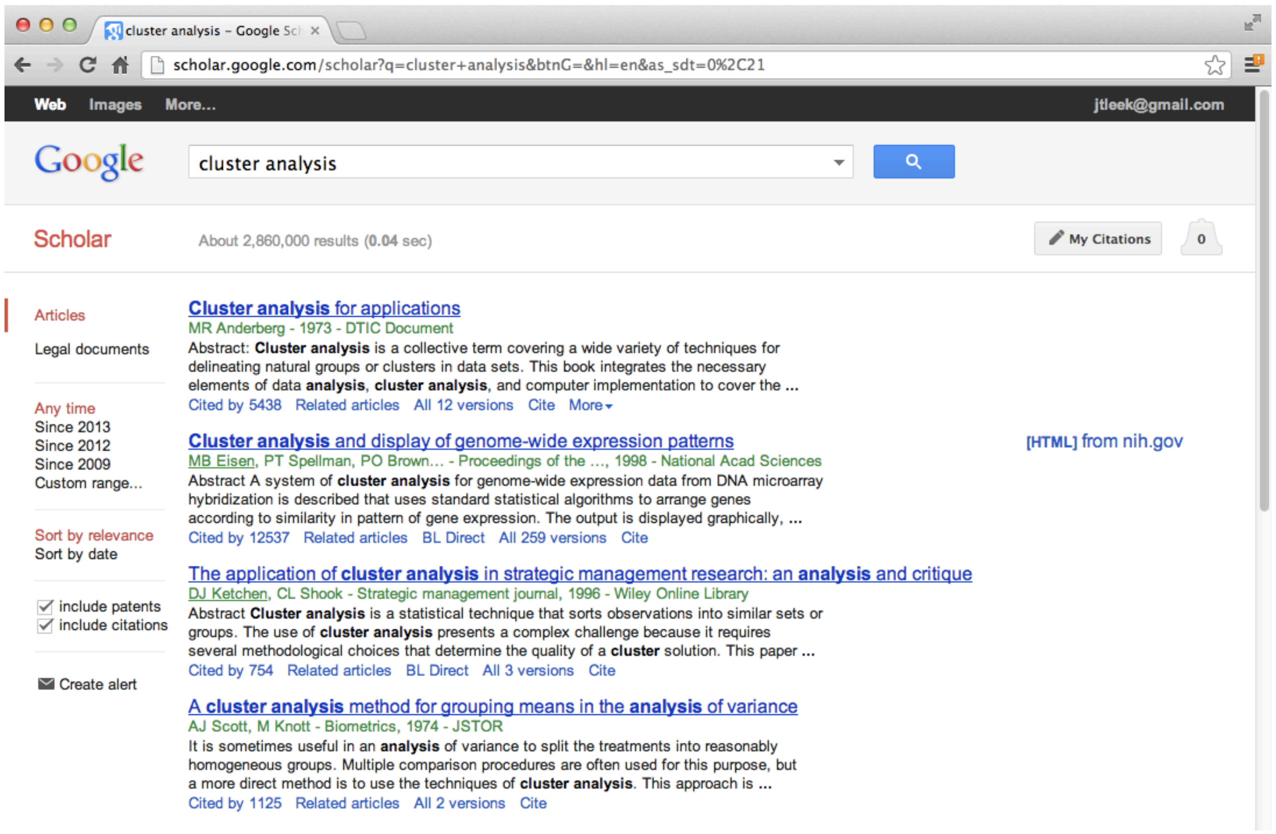
Roger D. Peng, Associate Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

Can we find things that are close together?

Clustering organizes things that are **close** into groups

- How do we define close?
- How do we group things?
- How do we visualize the grouping?
- How do we interpret the grouping?

Hugely important/impactful



About 2,860,000 results (0.04 sec)

Cluster analysis for applications
MR Anderberg - 1973 - DTIC Document
Abstract: **Cluster analysis** is a collective term covering a wide variety of techniques for delineating natural groups or clusters in data sets. This book integrates the necessary elements of data **analysis**, **cluster analysis**, and computer implementation to cover the ...
Cited by 5438 Related articles All 12 versions Cite More

Cluster analysis and display of genome-wide expression patterns — [HTML] from nih.gov
MB Eisen, PT Spellman, PO Brown... - Proceedings of the ..., 1998 - National Acad Sciences
Abstract A system of **cluster analysis** for genome-wide expression data from DNA microarray hybridization is described that uses standard statistical algorithms to arrange genes according to similarity in pattern of gene expression. The output is displayed graphically, ...
Cited by 12537 Related articles BL Direct All 259 versions Cite

The application of **cluster analysis** in strategic management research: an **analysis** and critique
DJ Ketchen, CL Shook - Strategic management journal, 1996 - Wiley Online Library
Abstract **Cluster analysis** is a statistical technique that sorts observations into similar sets or groups. The use of **cluster analysis** presents a complex challenge because it requires several methodological choices that determine the quality of a **cluster** solution. This paper ...
Cited by 754 Related articles BL Direct All 3 versions Cite

A **cluster analysis** method for grouping means in the **analysis** of variance
AJ Scott, M Knott - Biometrics, 1974 - JSTOR
It is sometimes useful in an **analysis** of variance to split the treatments into reasonably homogeneous groups. Multiple comparison procedures are often used for this purpose, but a more direct method is to use the techniques of **cluster analysis**. This approach is ...
Cited by 1125 Related articles All 2 versions Cite

http://scholar.google.com/scholar?hl=en&q=cluster+analysis&btnG=&as_sdt=1%2C21&as_sdtp=

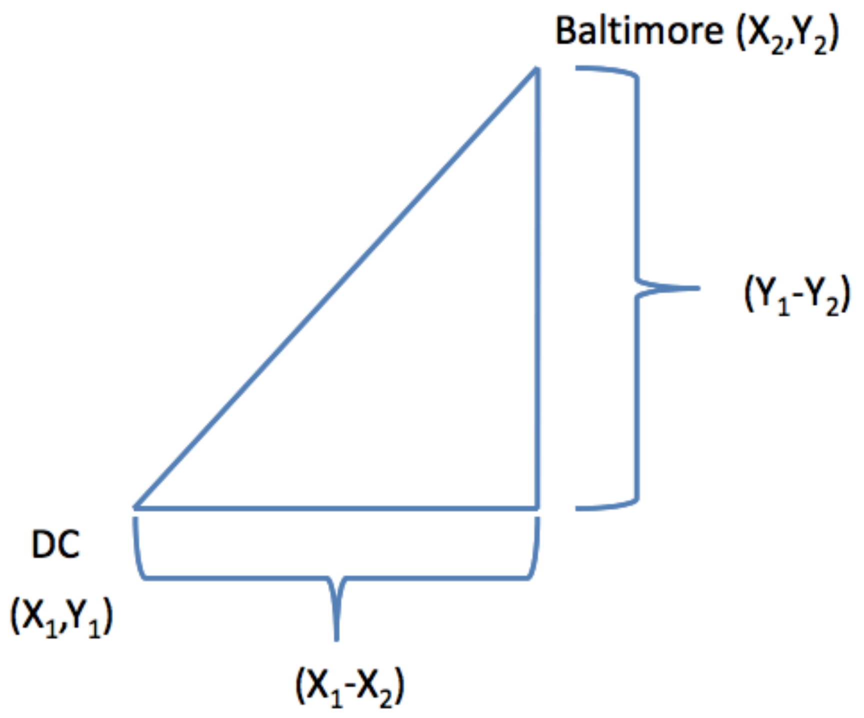
Hierarchical clustering

- An agglomerative approach
  - Find closest two things
  - Put them together
  - Find next closest
- Requires
  - A defined distance
  - A merging approach
- Produces
  - A tree showing how close things are to each other

How do we define close?

- Most important step
  - Garbage in -> garbage out
- Distance or similarity
  - Continuous - euclidean distance
  - Continuous - correlation similarity
  - Binary - manhattan distance
- Pick a distance/similarity that makes sense for your problem

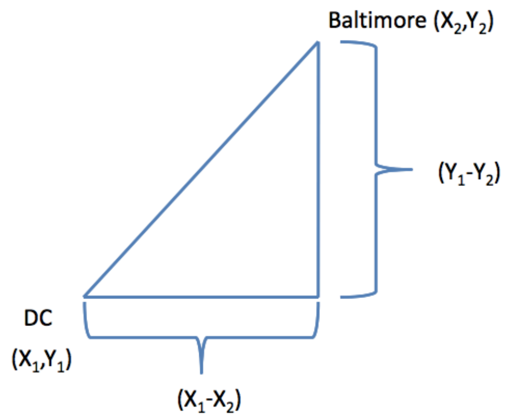
Example distances - Euclidean



http://rafalab.jhsph.edu/688/lec/lecture5-clustering.pdf

Example distances - Euclidean

$$\sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$$

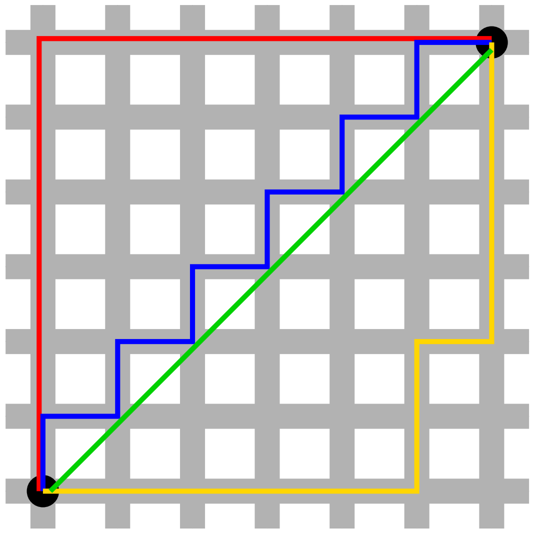


In general:

$$\sqrt{(A_1 - A_2)^2 + (B_1 - B_2)^2 + \ldots + (Z_1 - Z_2)^2}$$

http://rafalab.jhsph.edu/688/lec/lecture5-clustering.pdf

Example distances - Manhattan



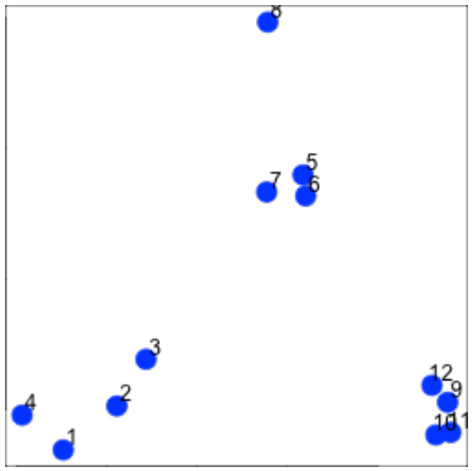
In general:

$$|A_1 - A_2| + |B_1 - B_2| + \ldots + |Z_1 - Z_2|$$

http://en.wikipedia.org/wiki/Taxicab_geometry

Hierarchical clustering - example

```r
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))
```



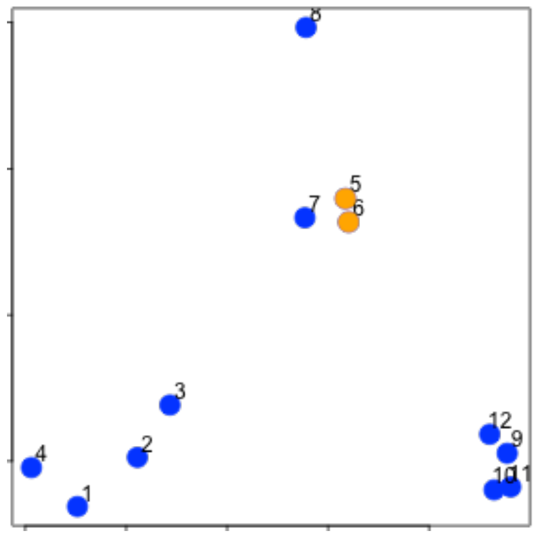
Hierarchical clustering - `dist`

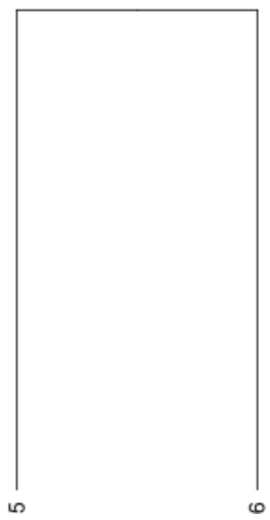
- Important parameters: *x*,*method*

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

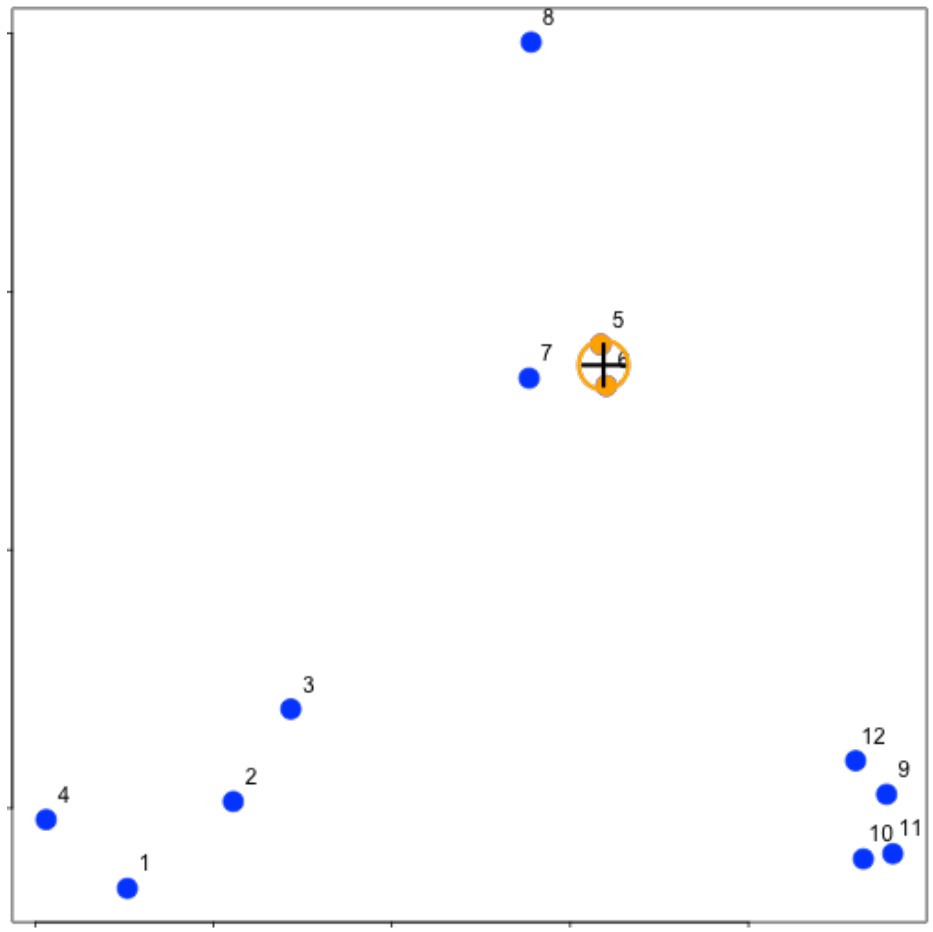
```
##          1       2       3       4       5       6       7       8       9
## 2  0.34121
## 3  0.57494 0.24103
## 4  0.26382 0.52579 0.71862
## 5  1.69425 1.35818 1.11953 1.80667
## 6  1.65813 1.31960 1.08339 1.78081 0.08150
## 7  1.49823 1.16621 0.92569 1.60132 0.21110 0.21667
## 8  1.99149 1.69093 1.45649 2.02849 0.61704 0.69792 0.65063
## 9  2.13630 1.83168 1.67836 2.35676 1.18350 1.11500 1.28583 1.76461
## 10 2.06420 1.76999 1.63110 2.29239 1.23848 1.16550 1.32063 1.83518 0.14090
## 11 2.14702 1.85183 1.71074 2.37462 1.28154 1.21077 1.37370 1.86999 0.11624
## 12 2.05664 1.74663 1.58659 2.27232 1.07701 1.00777 1.17740 1.66224 0.10849
##         10      11
## 2
## 3
## 4
## 5
```

Hierarchical clustering - #1

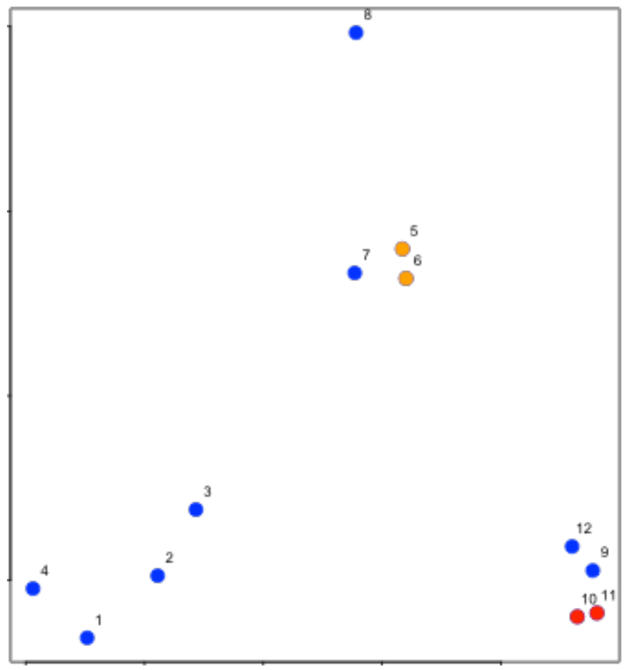


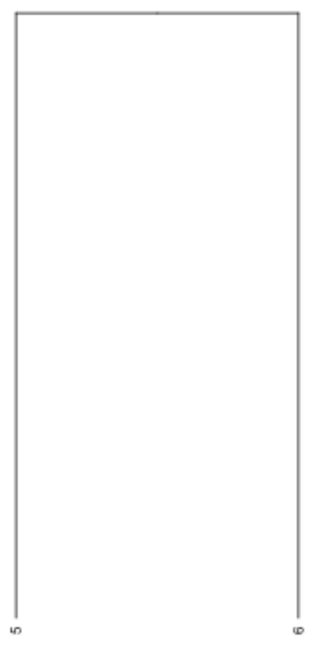


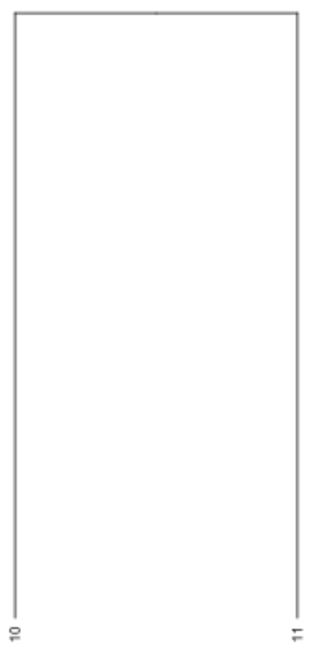
Hierarchical clustering - #2



Hierarchical clustering - #3

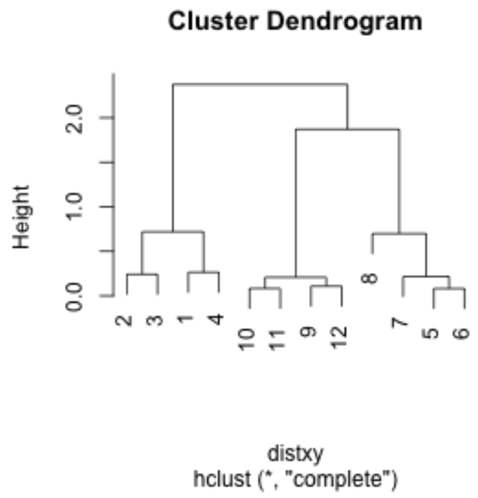






Hierarchical clustering - hclust

```
dataFrame <- data.frame(x = x, y = y)
distxy <- dist(dataFrame)
hClustering <- hclust(distxy)
plot(hClustering)
```

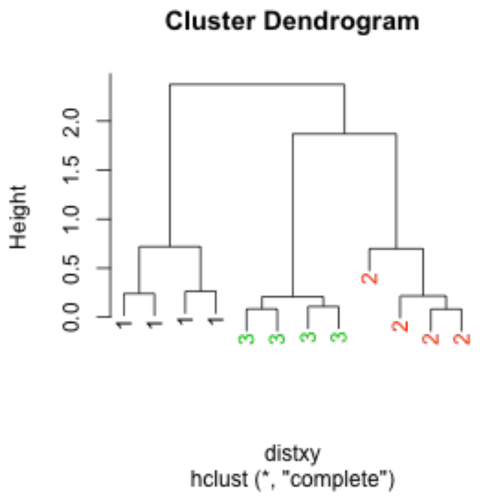


Prettier dendrograms

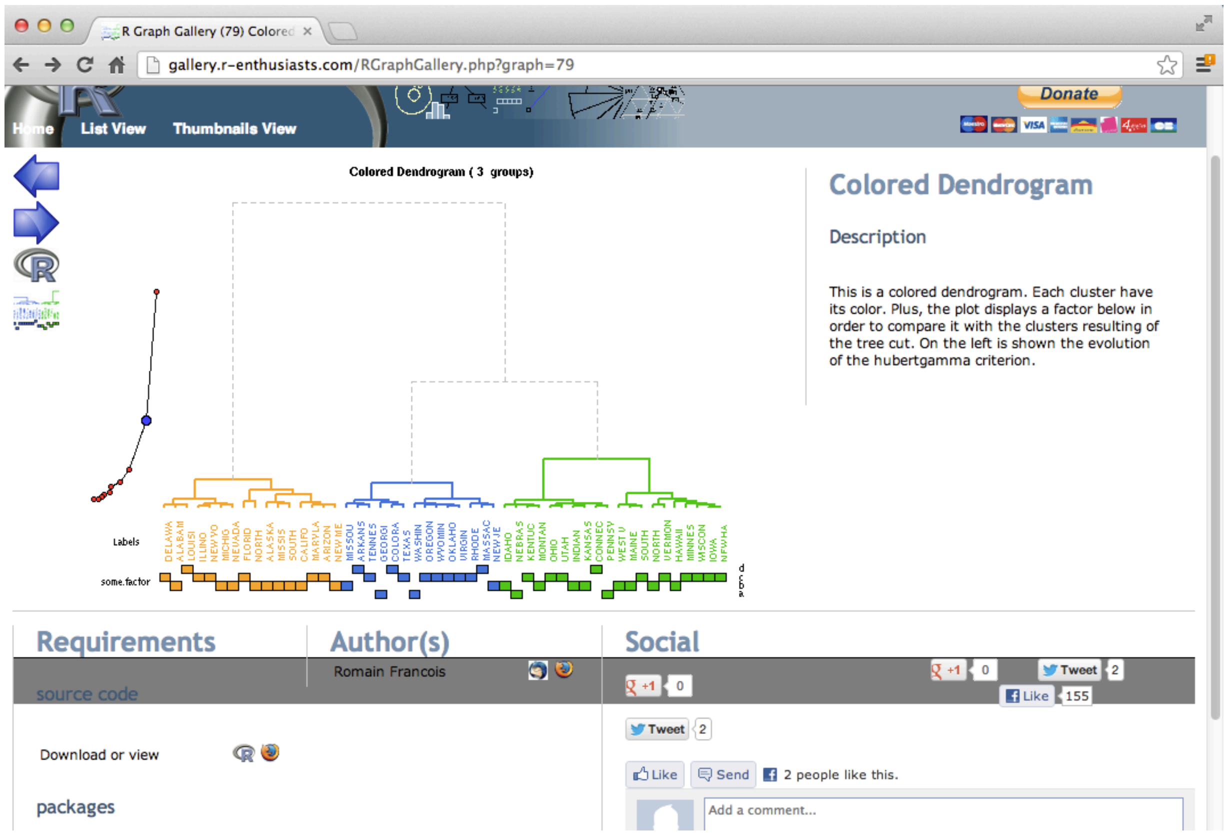
```r
myplclust <- function(hclust, lab = hclust$labels, lab.col = rep(1, length(hclust$labels)),
    hang = 0.1, ...) {
    ## modifiction of plclust for plotting hclust objects *in colour*!  Copyright
    ## Eva KF Chan 2009 Arguments: hclust: hclust object lab: a character vector
    ## of labels of 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, ...)
}
```

Pretty dendrograms

```r
dataFrame <- data.frame(x = x, y = y)
distxy <- dist(dataFrame)
hClustering <- hclust(distxy)
myplclust(hClustering, lab = rep(1:3, each = 4), lab.col = rep(1:3, each = 4))
```



Even Prettier dendrograms



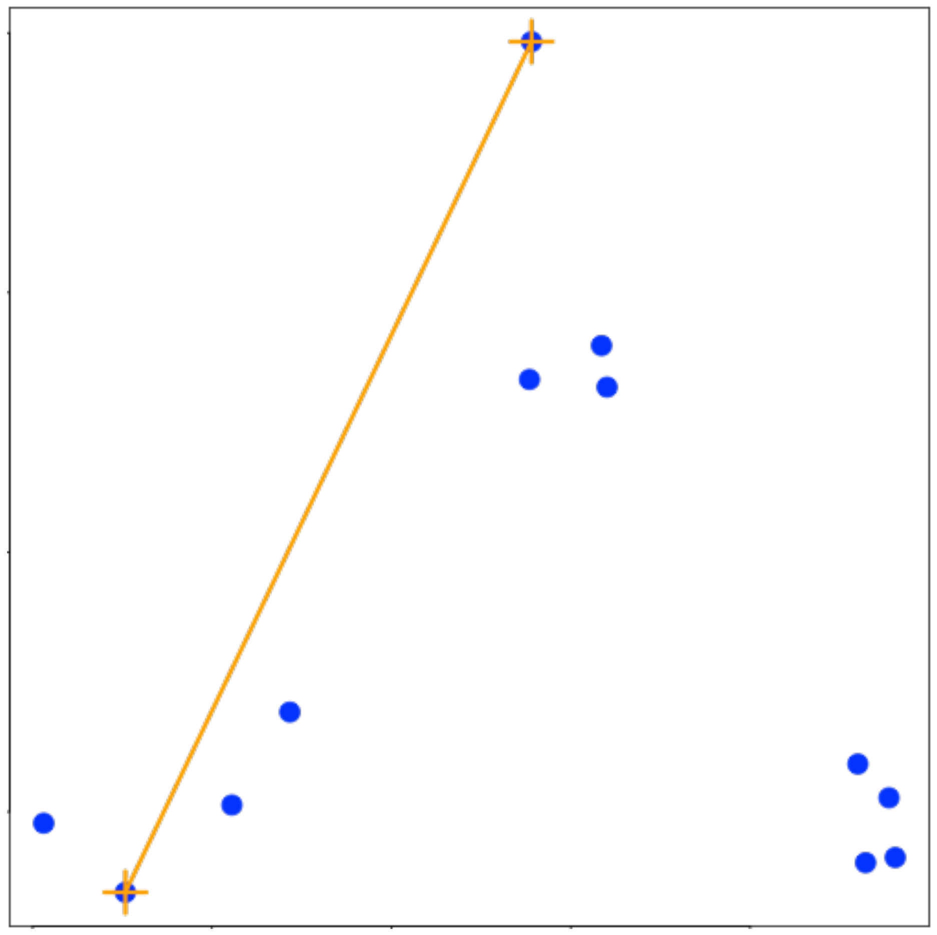
Colored Dendrogram

Description

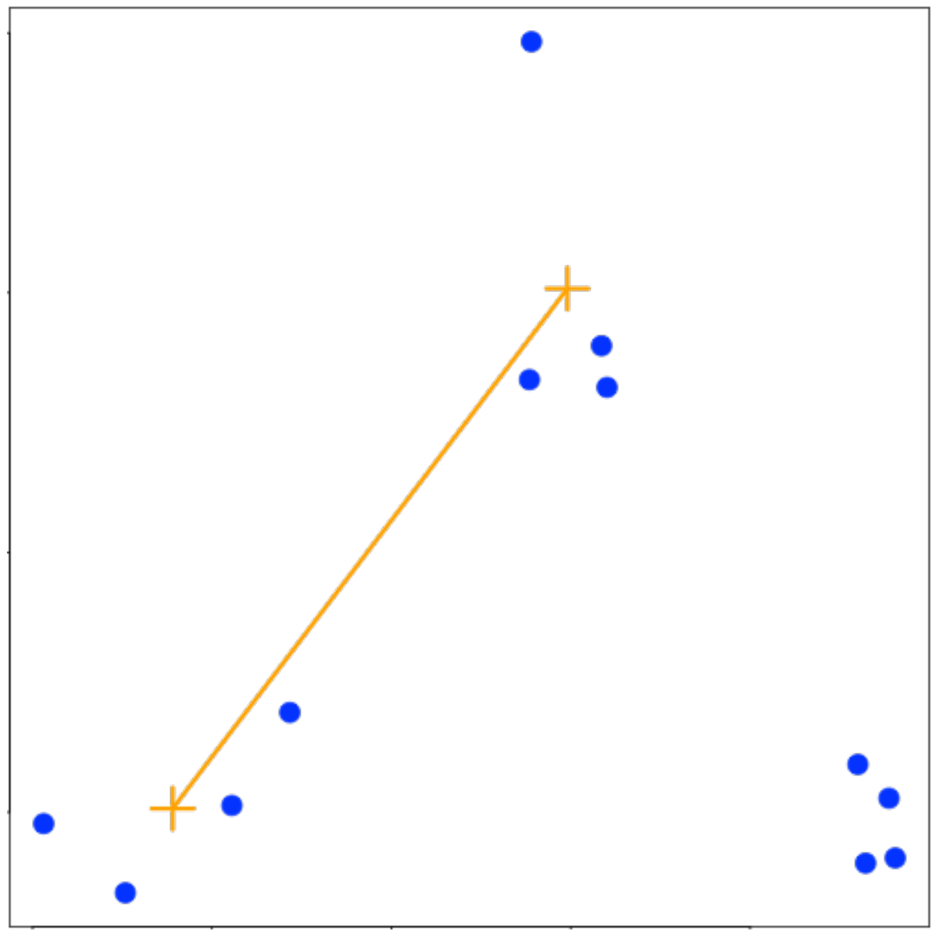
This is a colored dendrogram. Each cluster have its color. Plus, the plot displays a factor below in order to compare it with the clusters resulting of the tree cut. On the left is shown the evolution of the hubertgamma criterion.

http://gallery.r-enthusiasts.com/RGraphGallery.php?graph=79

Merging points - complete

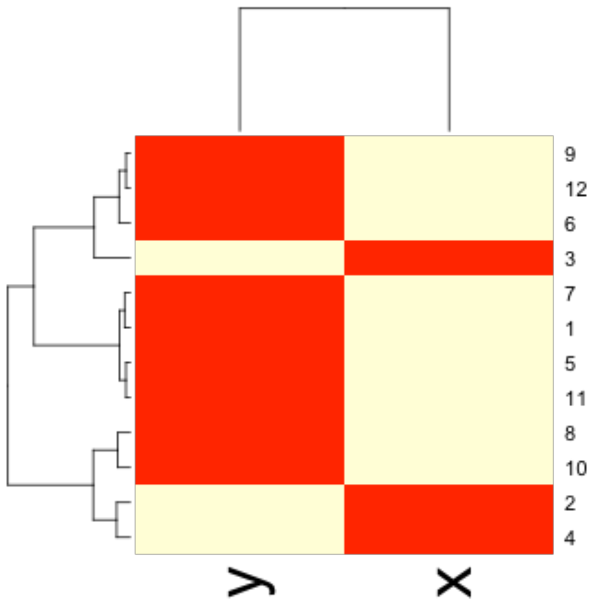


Merging points - average



heatmap()

```
dataFrame <- data.frame(x = x, y = y)
set.seed(143)
dataMatrix <- as.matrix(dataFrame)[sample(1:12), ]
heatmap(dataMatrix)
```



Notes and further resources

- Gives an idea of the relationships between variables/observations
- The picture may be unstable
  - Change a few points
  - Have different missing values
  - Pick a different distance
  - Change the merging strategy
  - Change the scale of points for one variable
- But it is deterministic
- Choosing where to cut isn't always obvious
- Should be primarily used for exploration
- Rafa's Distances and Clustering Video
- Elements of statistical learning

K-means Clustering

Roger D. Peng, Associate Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

Can we find things that are close together?

- How do we define close?
- How do we group things?
- How do we visualize the grouping?
- How do we interpret the grouping?

How do we define close?

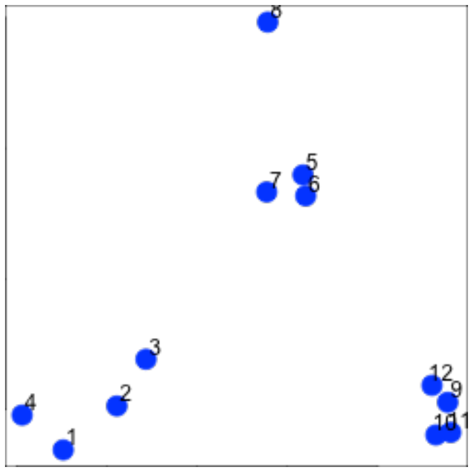
- Most important step
  - Garbage in $\longrightarrow$ garbage out
- Distance or similarity
  - Continuous - euclidean distance
  - Continous - correlation similarity
  - Binary - manhattan distance
- Pick a distance/similarity that makes sense for your problem

K-means clustering

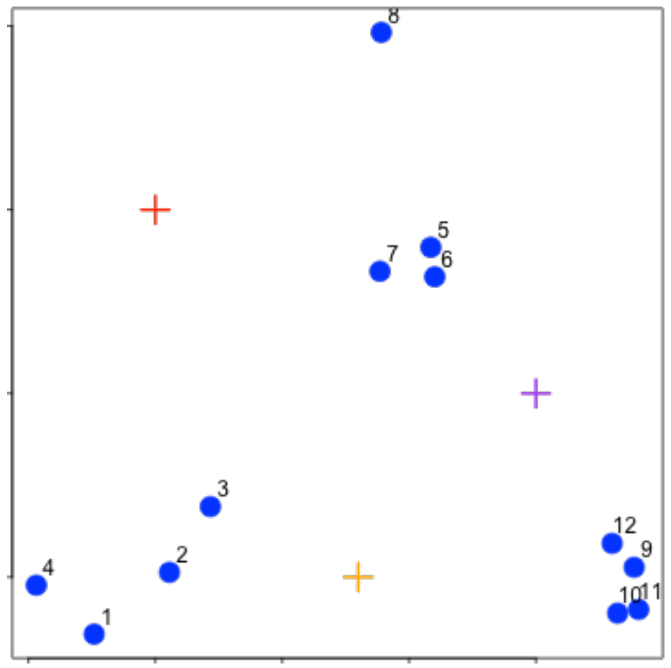
- A partioning approach
  - Fix a number of clusters
  - Get "centroids" of each cluster
  - Assign things to closest centroid
  - Reclaculate centroids
- Requires
  - A defined distance metric
  - A number of clusters
  - An initial guess as to cluster centroids
- Produces
  - Final estimate of cluster centroids
  - An assignment of each point to clusters

K-means clustering - example

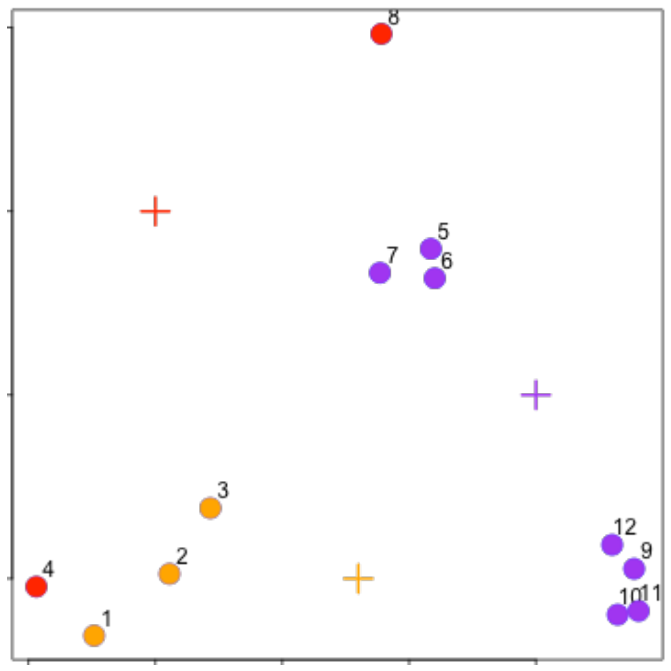
```r
set.seed(1234)
par(mar = c(0, 0, 0, 0))
x <- rnorm(12, mean = rep(1:3, each = 4), sd = 0.2)
y <- rnorm(12, mean = rep(c(1, 2, 1), each = 4), sd = 0.2)
plot(x, y, col = "blue", pch = 19, cex = 2)
text(x + 0.05, y + 0.05, labels = as.character(1:12))
```



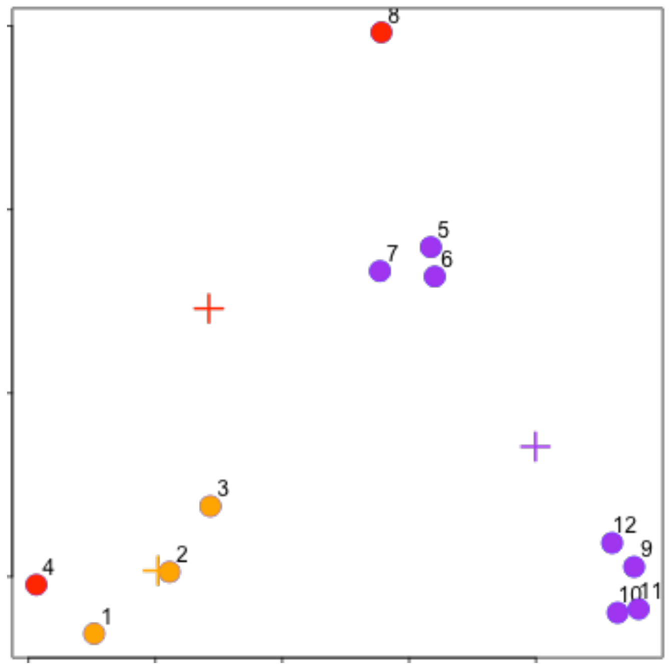
K-means clustering - starting centroids



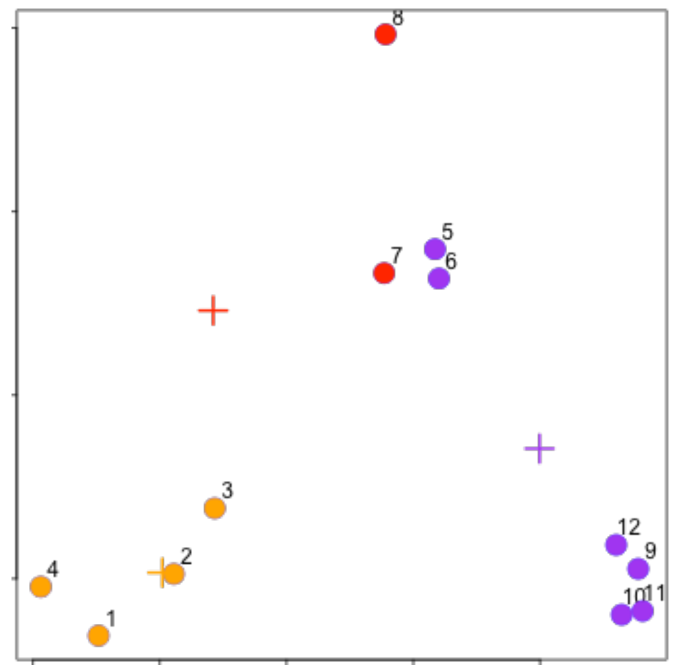
K-means clustering - assign to closest centroid



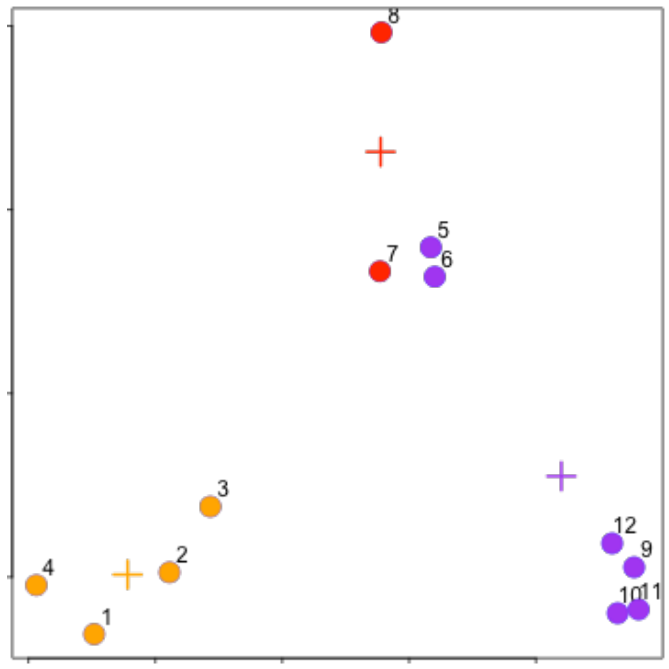
K-means clustering - recalculate centroids



K-means clustering - reassign values



K-means clustering - update centroids



`kmeans()`

- Important parameters: *x*, *centers*, *iter.max*, *nstart*

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

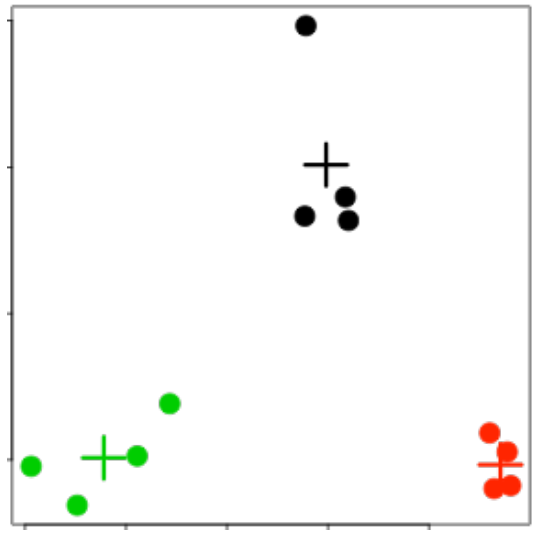
```
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
kmeansObj$cluster
```

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

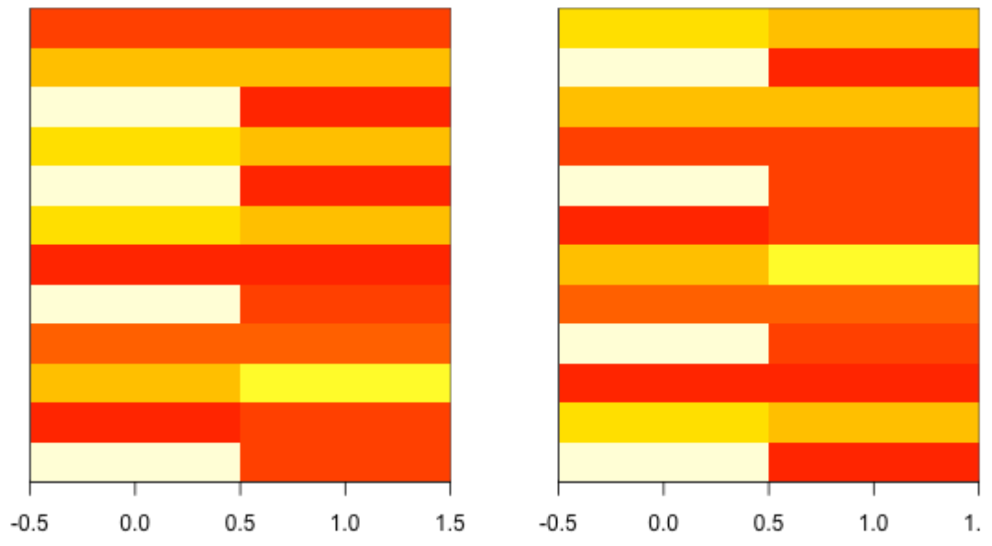
kmeans()

```
par(mar = rep(0.2, 4))
plot(x, y, col = kmeansObj$cluster, pch = 19, cex = 2)
points(kmeansObj$centers, col = 1:3, pch = 3, cex = 3, lwd = 3)
```



Heatmaps

```r
set.seed(1234)
dataMatrix <- as.matrix(dataFrame)[sample(1:12), ]
kmeansObj2 <- 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")
```



Notes and further resources

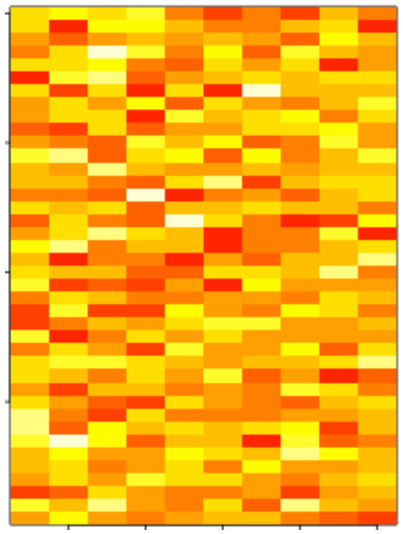
- K-means requires a number of clusters
  - Pick by eye/intuition
  - Pick by cross validation/information theory, etc.
  - Determining the number of clusters
- K-means is not deterministic
  - Different # of clusters
  - Different number of iterations
- Rafael Irizarry's Distances and Clustering Video
- Elements of statistical learning

Principal Components Analysis and Singular Value Decomposition

Roger D. Peng, Associate Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

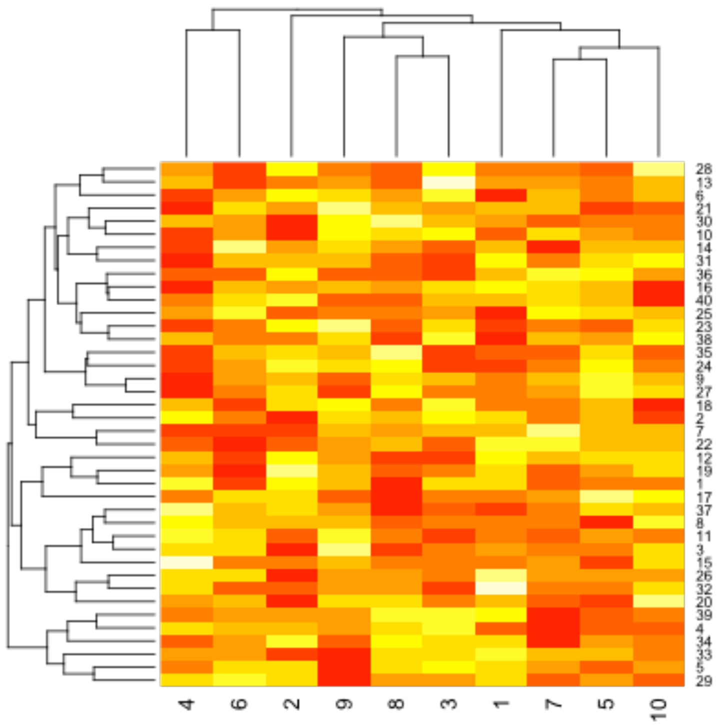
Matrix data

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



Cluster the data

```r
par(mar = rep(0.2, 4))
heatmap(dataMatrix)
```

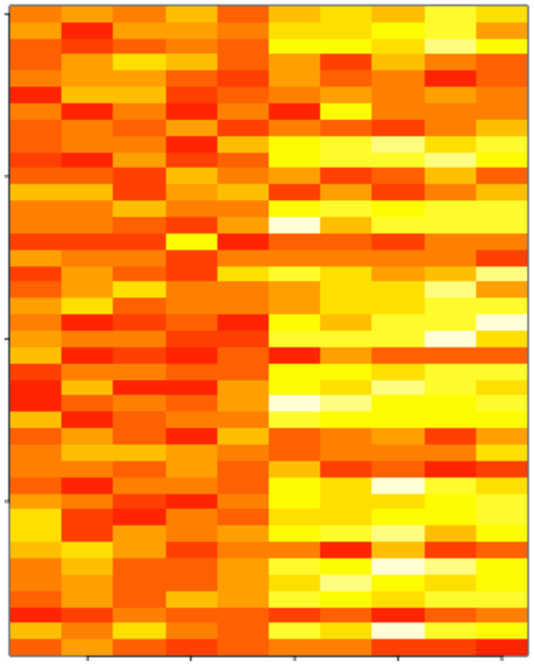


What if we add a pattern?

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

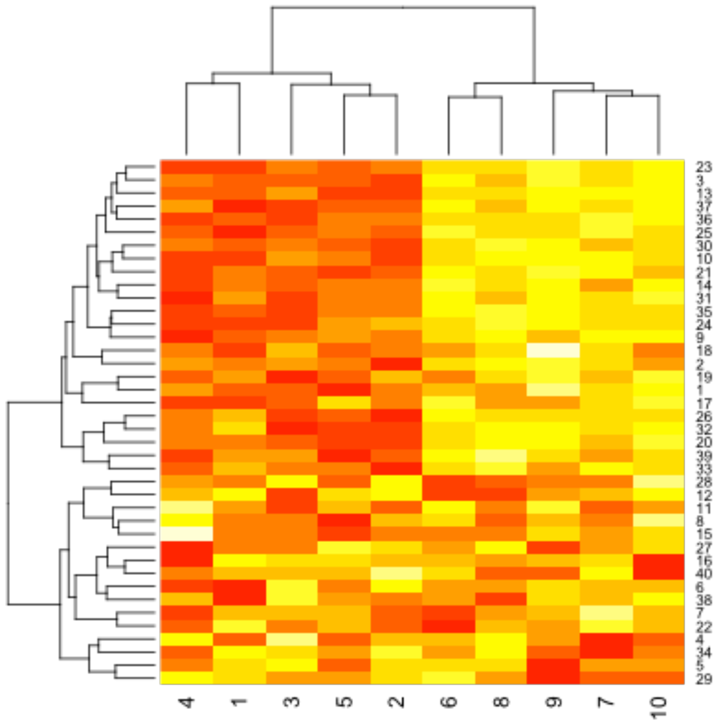
What if we add a pattern? - the data

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



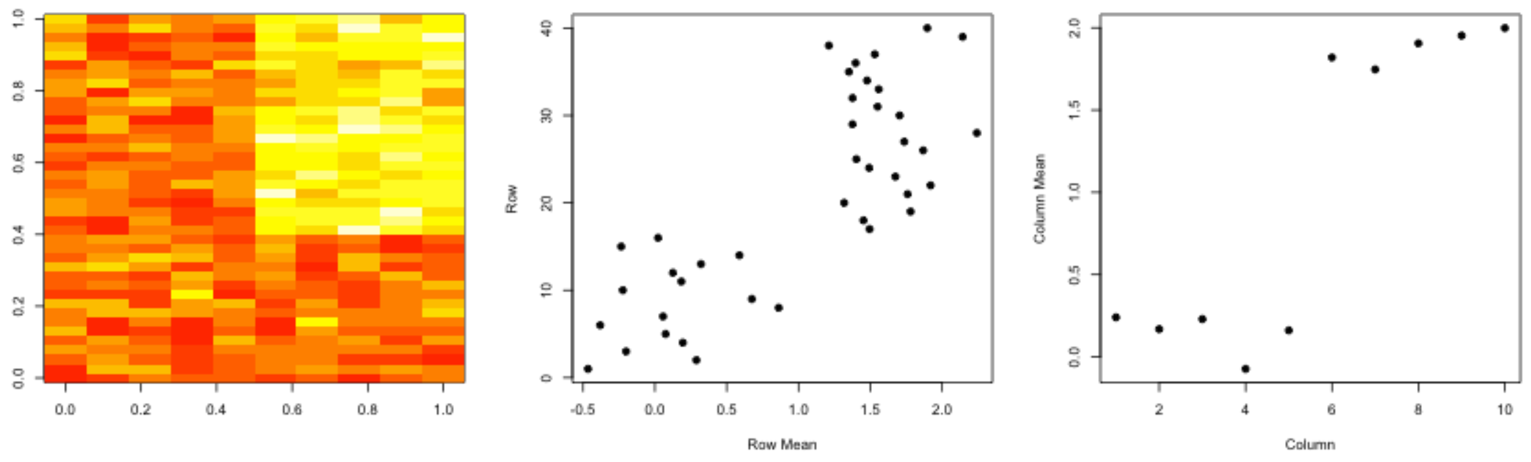
What if we add a pattern? - the clustered data

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



Patterns in rows and columns

```
hh <- hclust(dist(dataMatrix))
dataMatrixOrdered <- dataMatrix[hh$order, ]
par(mfrow = c(1, 3))
image(t(dataMatrixOrdered)[, nrow(dataMatrixOrdered):1])
plot(rowMeans(dataMatrixOrdered), 40:1, , xlab = "Row Mean", ylab = "Row", pch = 19)
plot(colMeans(dataMatrixOrdered), xlab = "Column", ylab = "Column Mean", pch = 19)
```



Related problems

You have multivariate variables $X_1, \ldots, X_n$ so $X_1 = (X_{11}, \ldots, X_{1m})$

- 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 (lower rank) that explains the original data.

The first goal is statistical and the second goal is data compression.

Related solutions - PCA/SVD

SVD

If X is a matrix with each variable in a column and each observation in a row then the SVD is a "matrix decomposition"

$$X = UDV^T$$

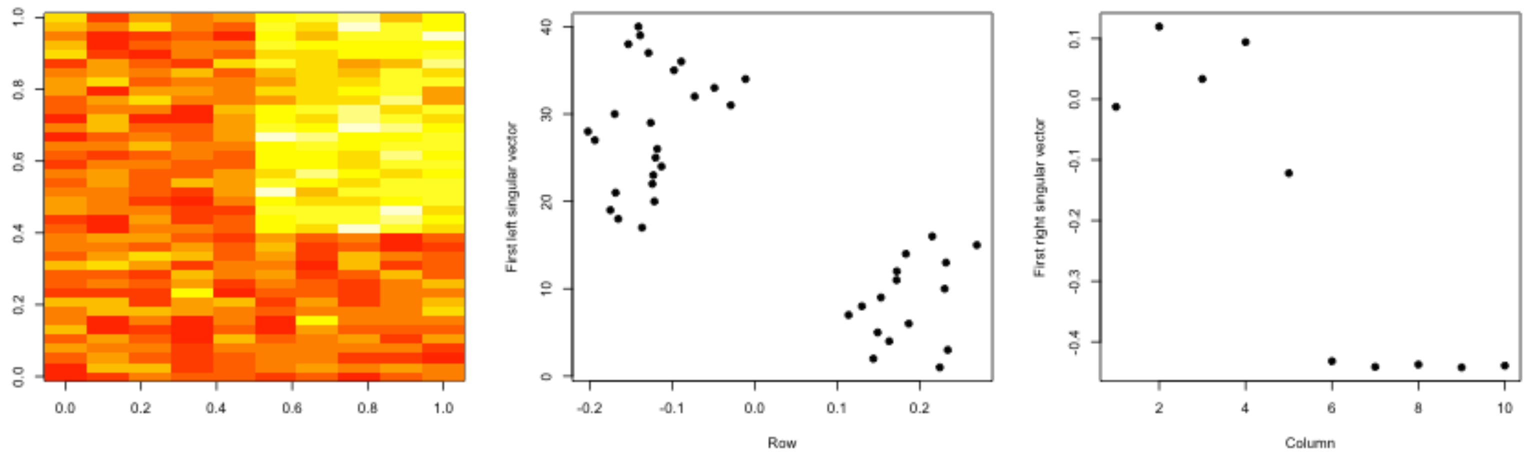
where the columns of U are orthogonal (left singular vectors), the columns of V are orthogonal (right singular vectors) and D is a diagonal matrix (singular values).

PCA

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

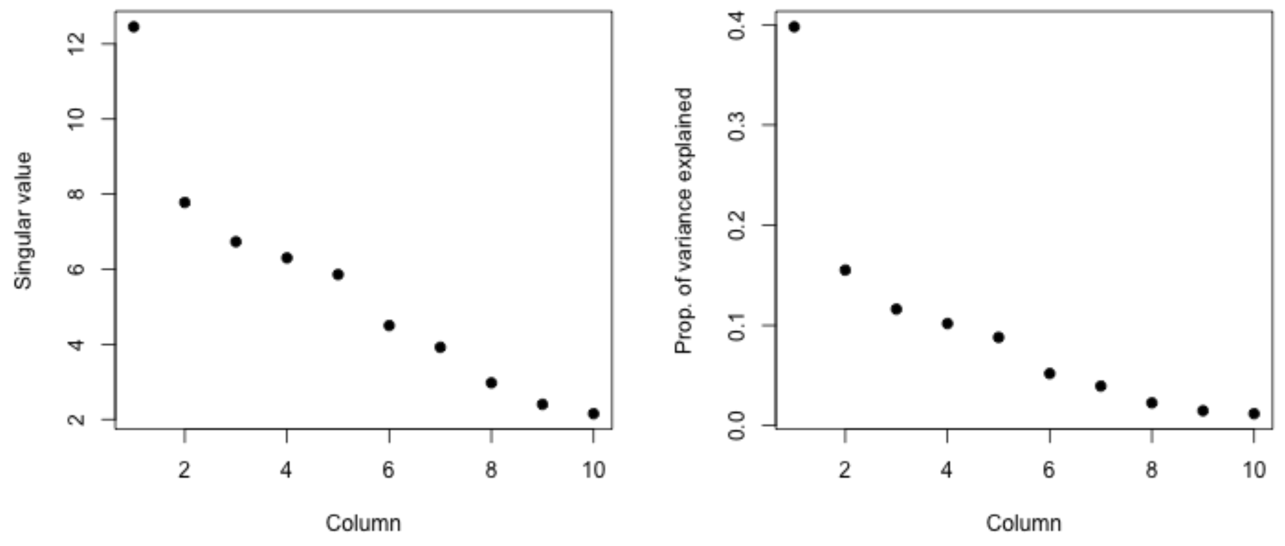
Components of the SVD - u and v

```
svd1 <- svd(scale(dataMatrixOrdered))
par(mfrow = c(1, 3))
image(t(dataMatrixOrdered)[, nrow(dataMatrixOrdered):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)
```



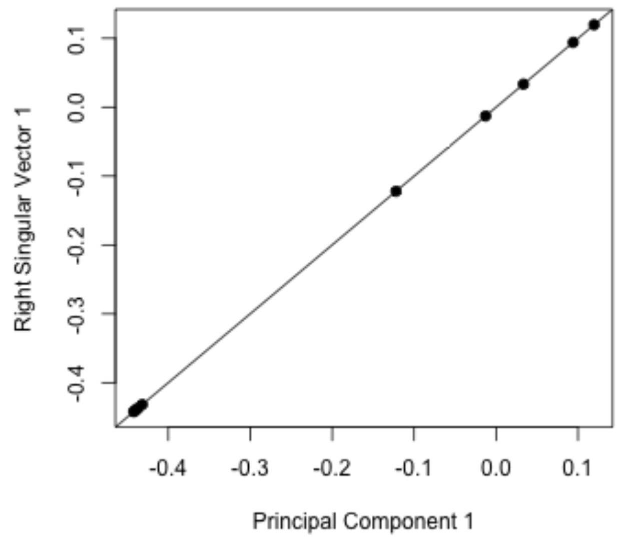
Components of the SVD - Variance explained

```r
par(mfrow = c(1, 2))
plot(svd1$d, xlab = "Column", ylab = "Singular value", pch = 19)
plot(svd1$d^2/sum(svd1$d^2), xlab = "Column", ylab = "Prop. of variance explained",
    pch = 19)
```



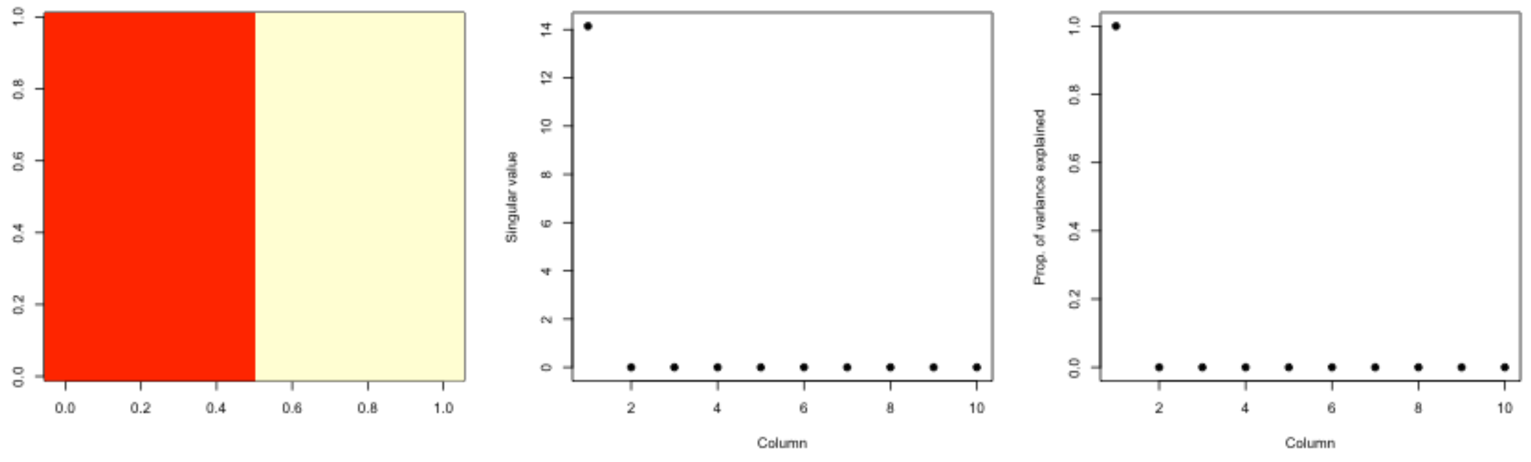
Relationship to principal components

```
svd1 <- svd(scale(dataMatrixOrdered))
pca1 <- prcomp(dataMatrixOrdered, scale = TRUE)
plot(pca1$rotation[, 1], svd1$v[, 1], pch = 19, xlab = "Principal Component 1",
    ylab = "Right Singular Vector 1")
abline(c(0, 1))
```



Components of the SVD - variance explained

```r
constantMatrix <- dataMatrixOrdered*0
for(i in 1:dim(dataMatrixOrdered)[1]){constantMatrix[i,] <- rep(c(0,1),each=5)}
svd1 <- svd(constantMatrix)
par(mfrow=c(1,3))
image(t(constantMatrix)[,nrow(constantMatrix):1])
plot(svd1$d,xlab="Column",ylab="Singular value",pch=19)
plot(svd1$d^2/sum(svd1$d^2),xlab="Column",ylab="Prop. of variance explained",pch=19)
```

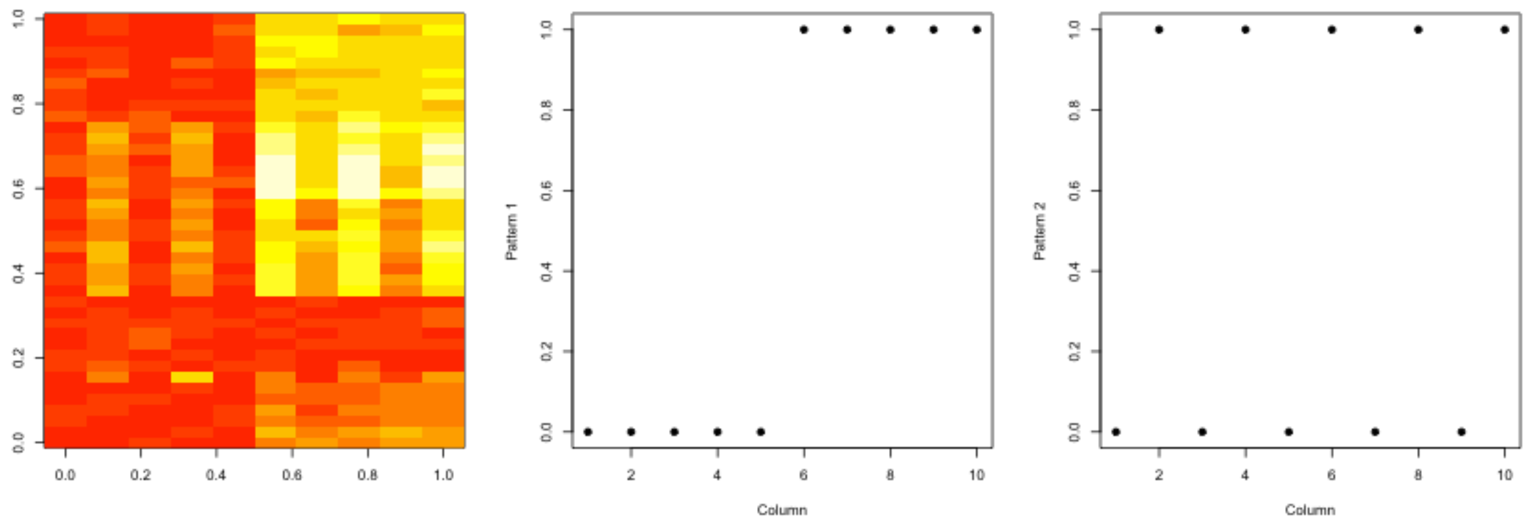


What if we add a second pattern?

```r
set.seed(678910)
for (i in 1:40) {
    # flip a coin
    coinFlip1 <- rbinom(1, size = 1, prob = 0.5)
    coinFlip2 <- rbinom(1, size = 1, prob = 0.5)
    # if coin is heads add a common pattern to that row
    if (coinFlip1) {
        dataMatrix[i, ] <- dataMatrix[i, ] + rep(c(0, 5), each = 5)
    }
    if (coinFlip2) {
        dataMatrix[i, ] <- dataMatrix[i, ] + rep(c(0, 5), 5)
    }
}
hh <- hclust(dist(dataMatrix))
dataMatrixOrdered <- dataMatrix[hh$order, ]
```

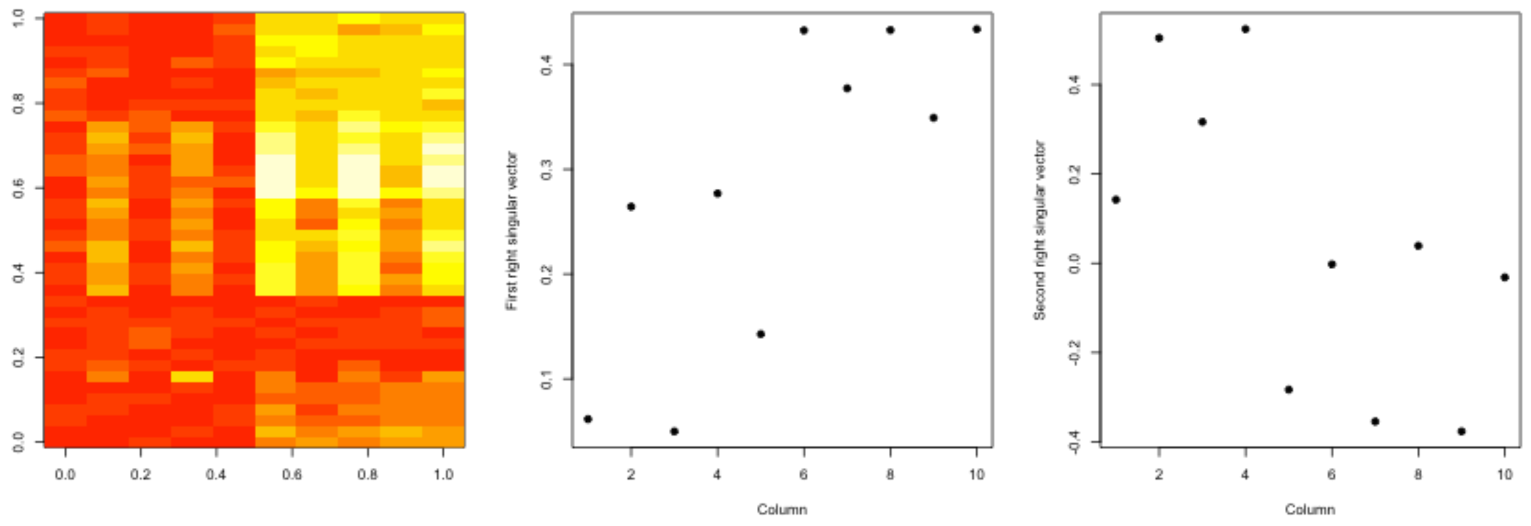
Singular value decomposition - true patterns

```
svd2 <- svd(scale(dataMatrixOrdered))
par(mfrow = c(1, 3))
image(t(dataMatrixOrdered)[, nrow(dataMatrixOrdered):1])
plot(rep(c(0, 1), each = 5), pch = 19, xlab = "Column", ylab = "Pattern 1")
plot(rep(c(0, 1), 5), pch = 19, xlab = "Column", ylab = "Pattern 2")
```



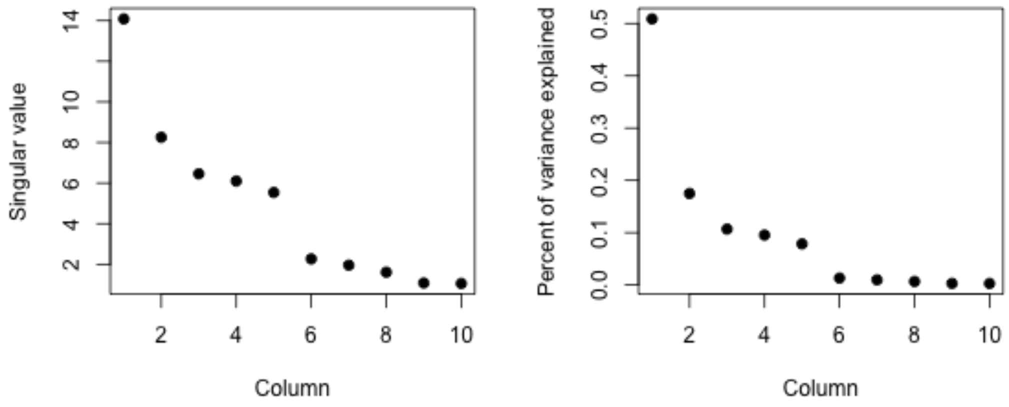
v and patterns of variance in rows

```
svd2 <- svd(scale(dataMatrixOrdered))
par(mfrow = c(1, 3))
image(t(dataMatrixOrdered)[, nrow(dataMatrixOrdered):1])
plot(svd2$v[, 1], pch = 19, xlab = "Column", ylab = "First right singular vector")
plot(svd2$v[, 2], pch = 19, xlab = "Column", ylab = "Second right singular vector")
```



d and variance explained

```r
svd1 <- svd(scale(dataMatrixOrdered))
par(mfrow = c(1, 2))
plot(svd1$d, xlab = "Column", ylab = "Singular value", pch = 19)
plot(svd1$d^2/sum(svd1$d^2), xlab = "Column", ylab = "Percent of variance explained",
    pch = 19)
```



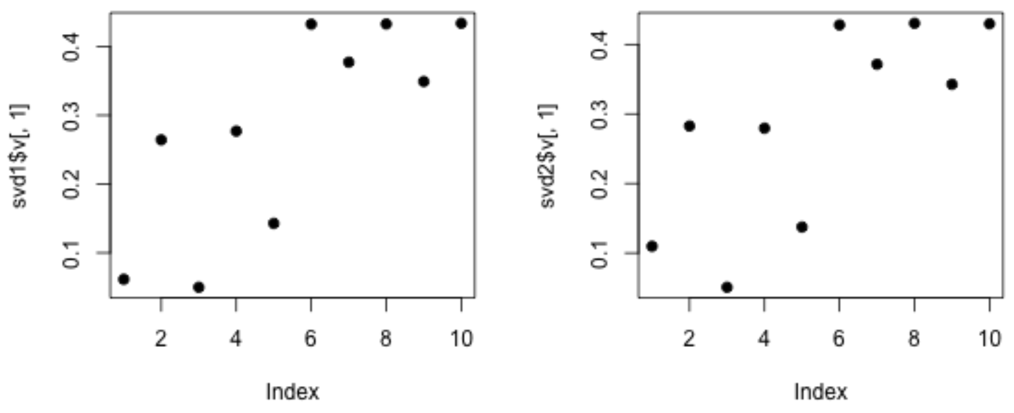
Missing values

```
dataMatrix2 <- dataMatrixOrdered
## Randomly insert some missing data
dataMatrix2[sample(1:100, size = 40, replace = FALSE)] <- NA
svd1 <- svd(scale(dataMatrix2))  ## Doesn't work!
```

```
## Error: infinite or missing values in 'x'
```

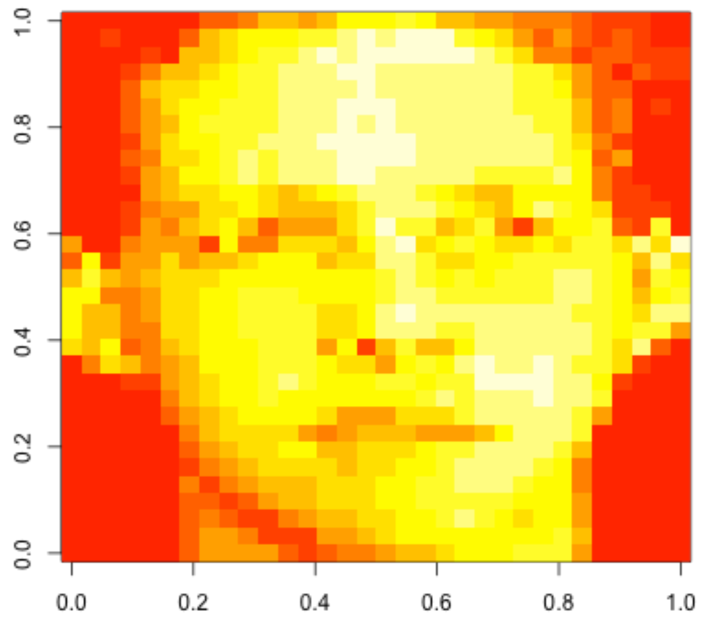
Imputing {impute}

```
library(impute)  ## Available from http://bioconductor.org
dataMatrix2 <- dataMatrixOrdered
dataMatrix2[sample(1:100,size=40,replace=FALSE)] <- NA
dataMatrix2 <- impute.knn(dataMatrix2)$data
svd1 <- svd(scale(dataMatrixOrdered)); svd2 <- svd(scale(dataMatrix2))
par(mfrow=c(1,2)); plot(svd1$v[,1],pch=19); plot(svd2$v[,1],pch=19)
```



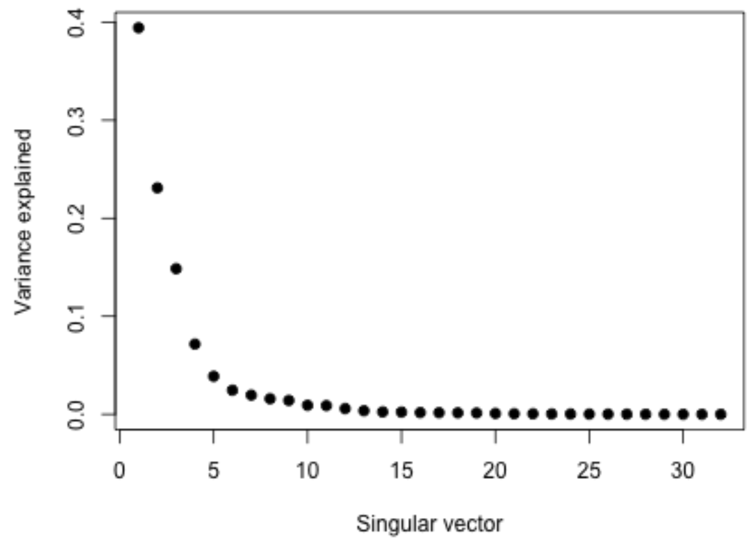
Face example

```r
load("data/face.rda")
image(t(faceData)[, nrow(faceData):1])
```



Face example - variance explained

```
svd1 <- svd(scale(faceData))
plot(svd1$d^2/sum(svd1$d^2), pch = 19, xlab = "Singular vector", ylab = "Variance explained")
```

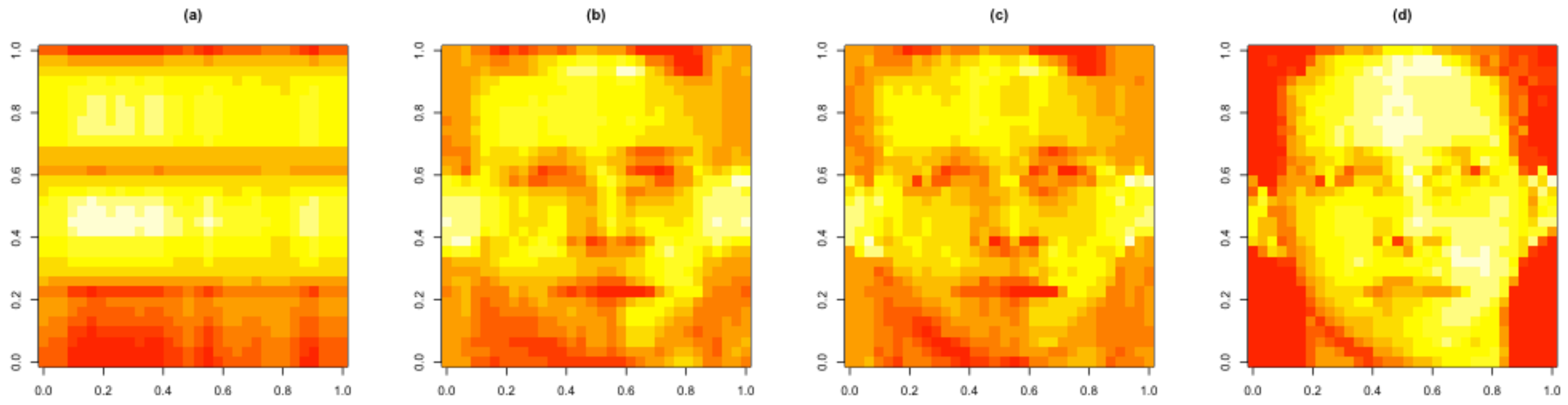


Face example - create approximations

```r
svd1 <- svd(scale(faceData))
## Note that %*% is matrix multiplication

# Here svd1$d[1] is a constant
approx1 <- svd1$u[, 1] %*% t(svd1$v[, 1]) * svd1$d[1]

# In these examples we need to make the diagonal matrix out of d
approx5 <- svd1$u[, 1:5] %*% diag(svd1$d[1:5]) %*% t(svd1$v[, 1:5])
approx10 <- svd1$u[, 1:10] %*% diag(svd1$d[1:10]) %*% t(svd1$v[, 1:10])
```

Face example - plot approximations

```r
par(mfrow = c(1, 4))
image(t(approx1)[, nrow(approx1):1], main = "(a)")
image(t(approx5)[, nrow(approx5):1], main = "(b)")
image(t(approx10)[, nrow(approx10):1], main = "(c)")
image(t(faceData)[, nrow(faceData):1], main = "(d)")  ## Original data
```



Notes and further resources

- Scale matters
- PC's/SV's may mix real patterns
- Can be computationally intensive
- Advanced data analysis from an elementary point of view
- Elements of statistical learning
- Alternatives
  - Factor analysis
  - Independent components analysis
  - Latent semantic analysis

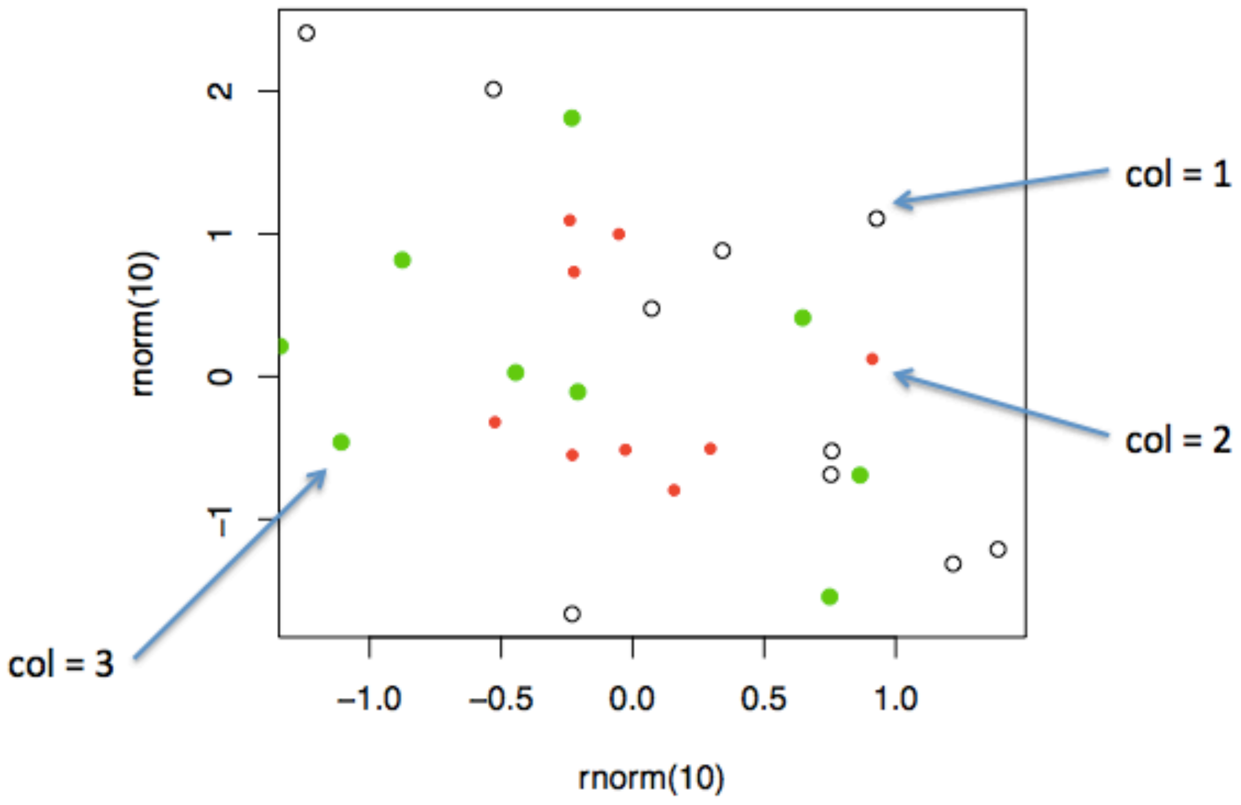
Plotting and Color in R

Roger D. Peng, Associate Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

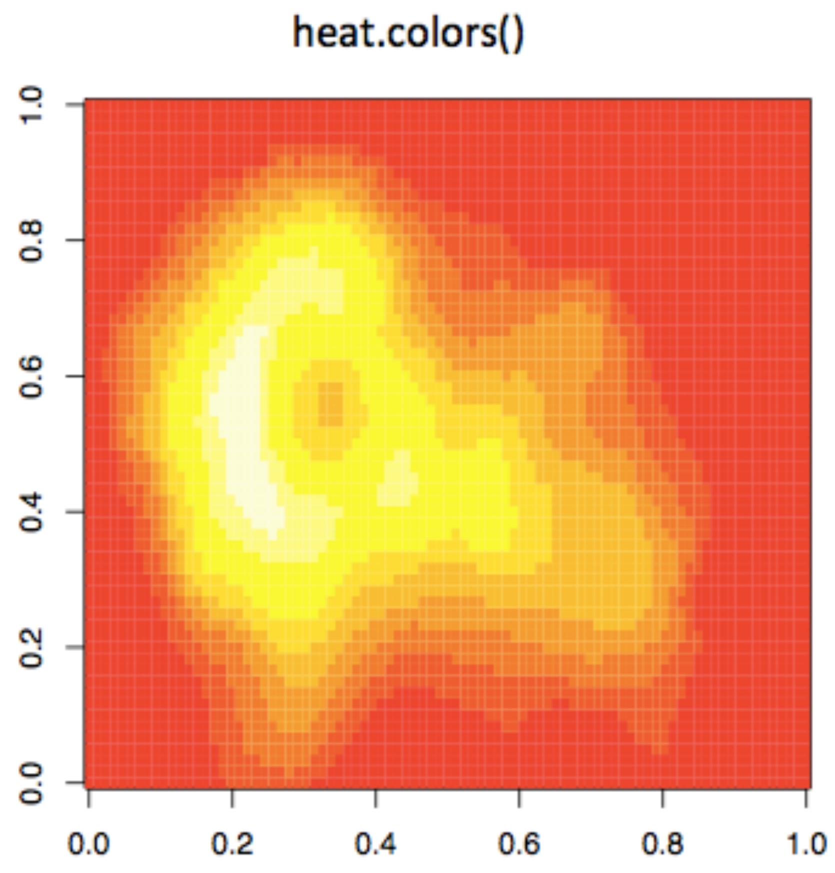
Plotting and Color

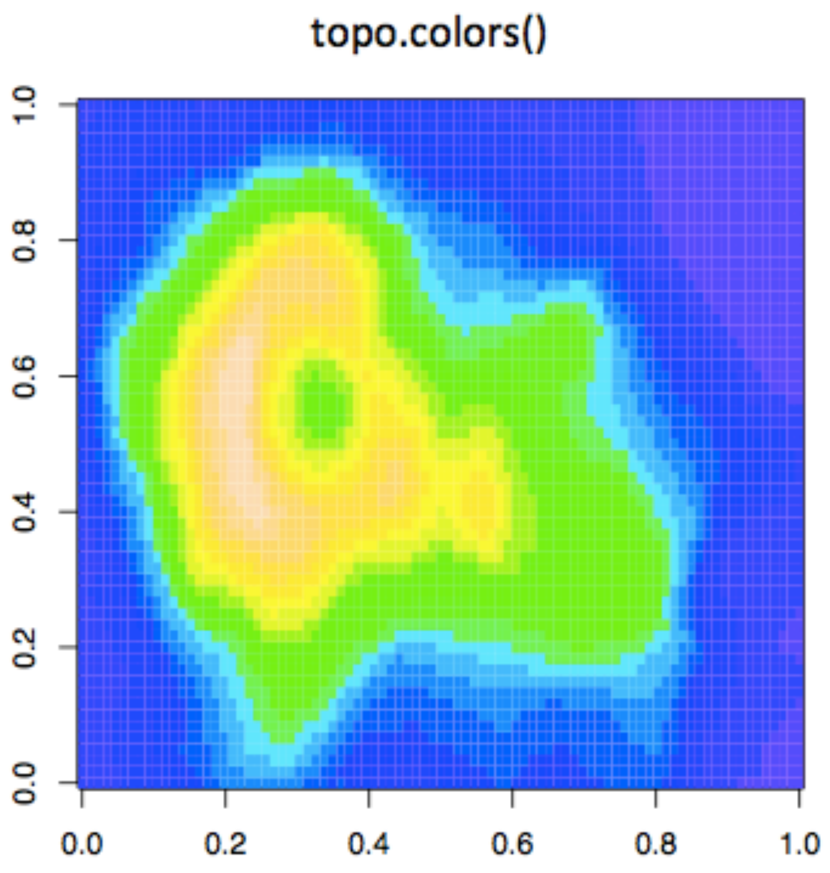
- The default color schemes for most plots in R are horrendous
  - I don't have good taste and even I know that
- Recently there have been developments to improve the handling/specifica1on of colors in plots/graphs/etc.
- There are functions in R and in external packages that are very handy

Colors 1, 2, and 3



Default Image Plots in R





Color U1li1es in R

- The `grDevices` package has two functions
  - `colorRamp`
  - `colorRampPalette`
- These functions take palettes of colors and help to interpolate between the colors
- The function colors() lists the names of colors you can use in any plotting function

Color Palette Utilities in R

- `colorRamp`: Take a palette of colors and return a function that takes valeus between 0 and 1, indicating the extremes of the color palette (e.g. see the 'gray' function)
- `colorRampPalette`: Take a palette of colors and return a function that takes integer arguments and returns a vector of colors interpolating the palette (like `heat.colors` or `topo.colors`)

colorRamp

`[,1] [,2] [,3]` corresponds to `[Red] [Blue] [Green]`

```
> pal <- colorRamp(c("red", "blue"))

> pal(0)
     [,1] [,2] [,3]
[1,]  255    0    0

> pal(1)
     [,1] [,2] [,3]
[1,]    0    0  255

> pal(0.5)
      [,1] [,2]  [,3]
[1,] 127.5    0 127.5
```

colorRamp

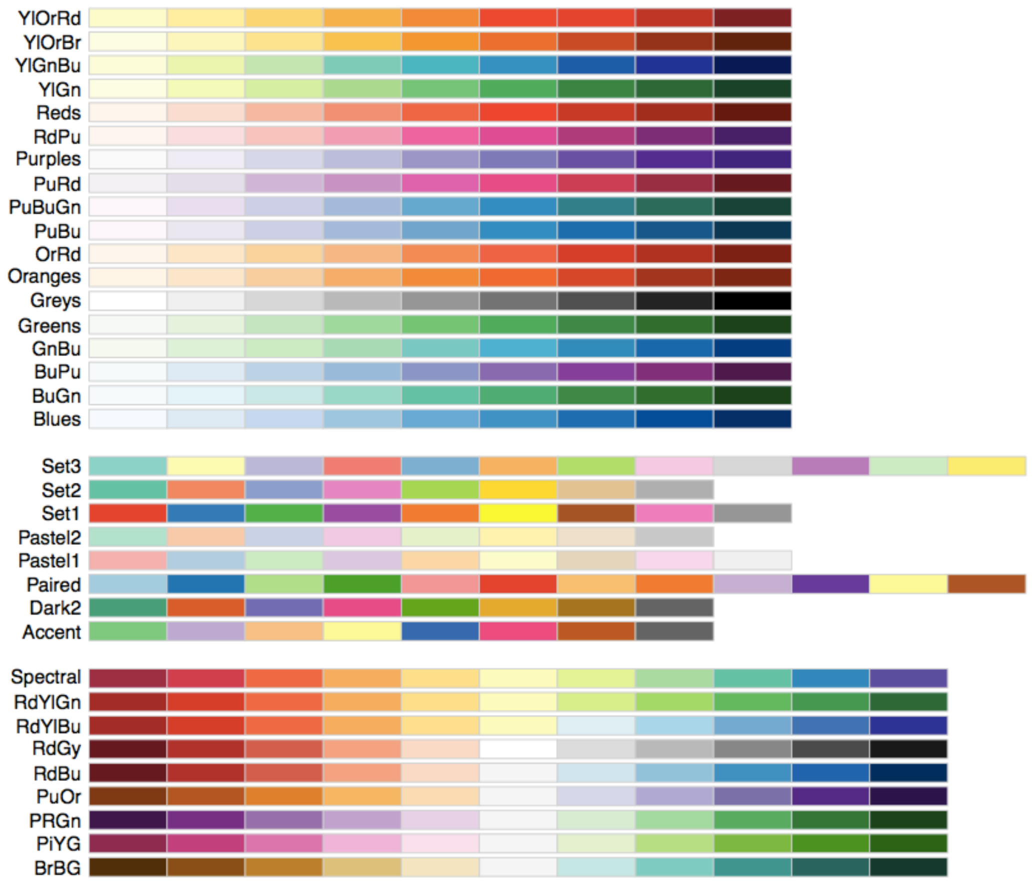
```
> pal(seq(0, 1, len = 10))
                   [,1] [,2]       [,3]
        [1,] 255.00000    0          0
        [2,] 226.66667    0   28.33333
        [3,] 198.33333    0   56.66667
        [4,] 170.00000    0   85.00000
        [5,] 141.66667    0  113.33333
        [6,] 113.33333    0  141.66667
        [7,]  85.00000    0  170.00000
        [8,]  56.66667    0  198.33333
        [9,]  28.33333    0  226.66667
        [10,]  0.00000    0  255.00000
```

colorRampPalette

```
> pal <- colorRampPalette(c("red", "yellow"))

> pal(2)
[1] "#FF0000" "#FFFF00"

> pal(10)
 [1] "#FF0000" "#FF1C00" "#FF3800" "#FF5500" "#FF7100"
 [6] "#FF8D00" "#FFAA00" "#FFC600" "#FFE200" "#FFFF00”
```

RColorBrewer Package

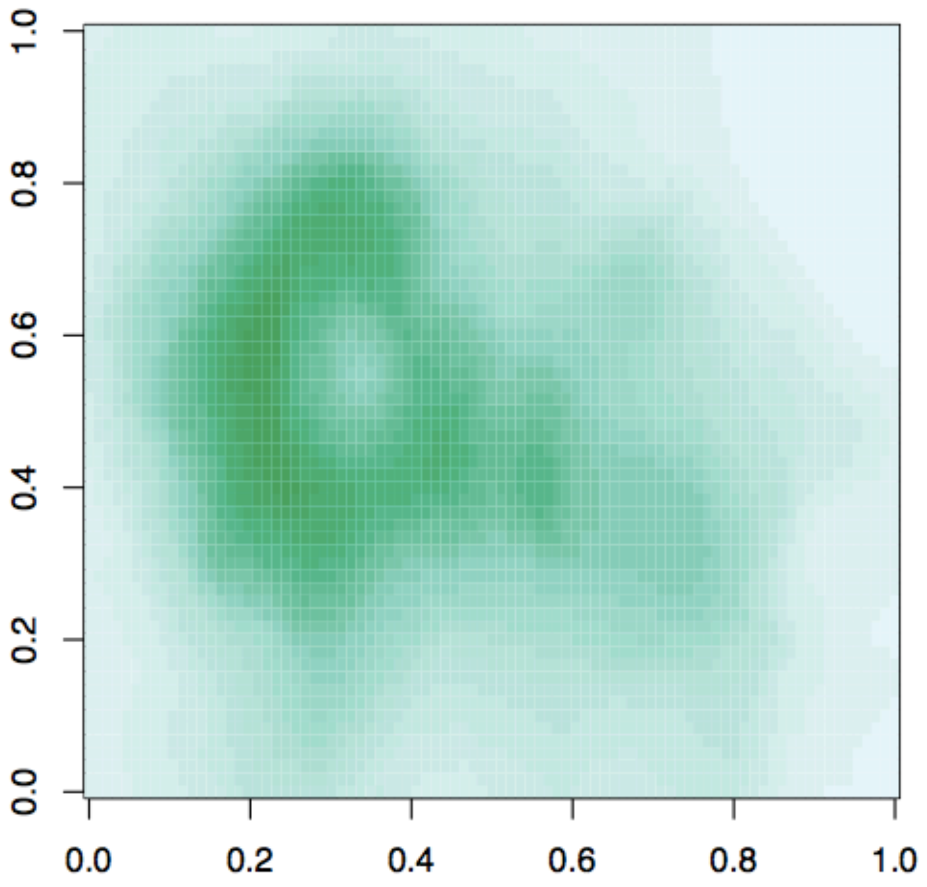
- One package on CRAN that contains interes1ng/useful color palettes
- There are 3 types of palettes
  - Sequential
  - Diverging
  - Qualitative
- Palette informa1on can be used in conjunction with the `colorRamp()` and `colorRampPalette()`


YlOrRd
YlOrBr
YlGnBu
YlGn
Reds
RdPu
Purples
PuRd
PuBuGn
PuBu
OrRd
Oranges
Greys
Greens
GnBu
BuPu
BuGn
Blues
Set3
Set2
Set1
Pastel2
Pastel1
Paired
Dark2
Accent
Spectral
RdYlGn
RdYlBu
RdGy
RdBu
PuOr
PRGn
PiYG
BrBG

RColorBrewer and colorRampPalette

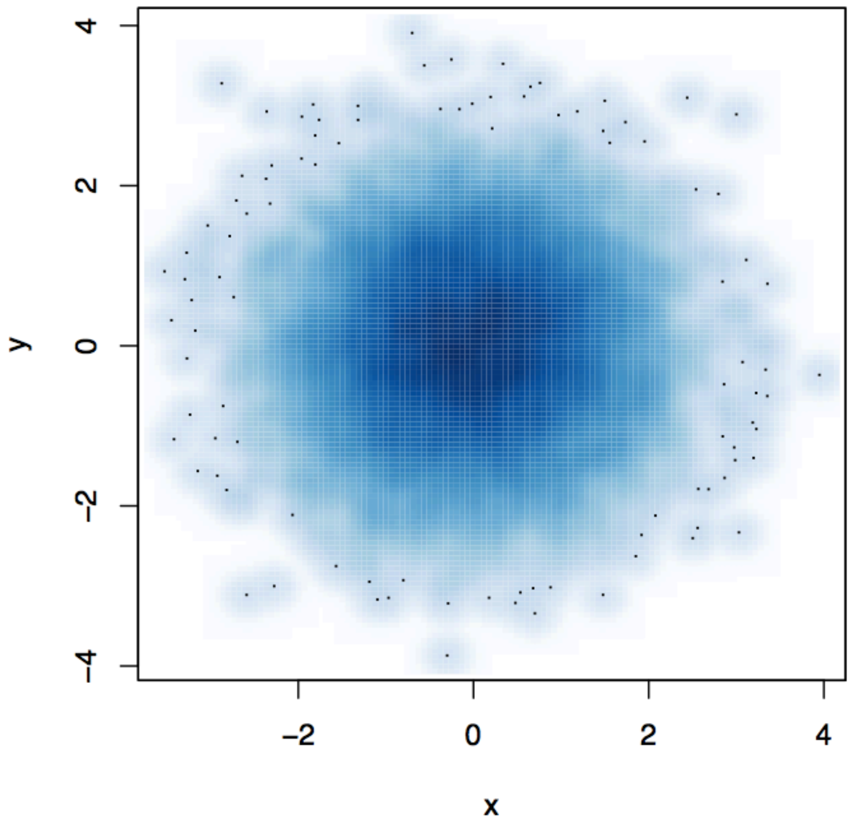
```
> library(RColorBrewer)

> cols <- brewer.pal(3, "BuGn")

> cols
[1] "#E5F5F9" "#99D8C9" "#2CA25F"

> pal <- colorRampPalette(cols)

> image(volcano, col = pal(20))
```

RColorBrewer and colorRampPalette



The smoothScatter function

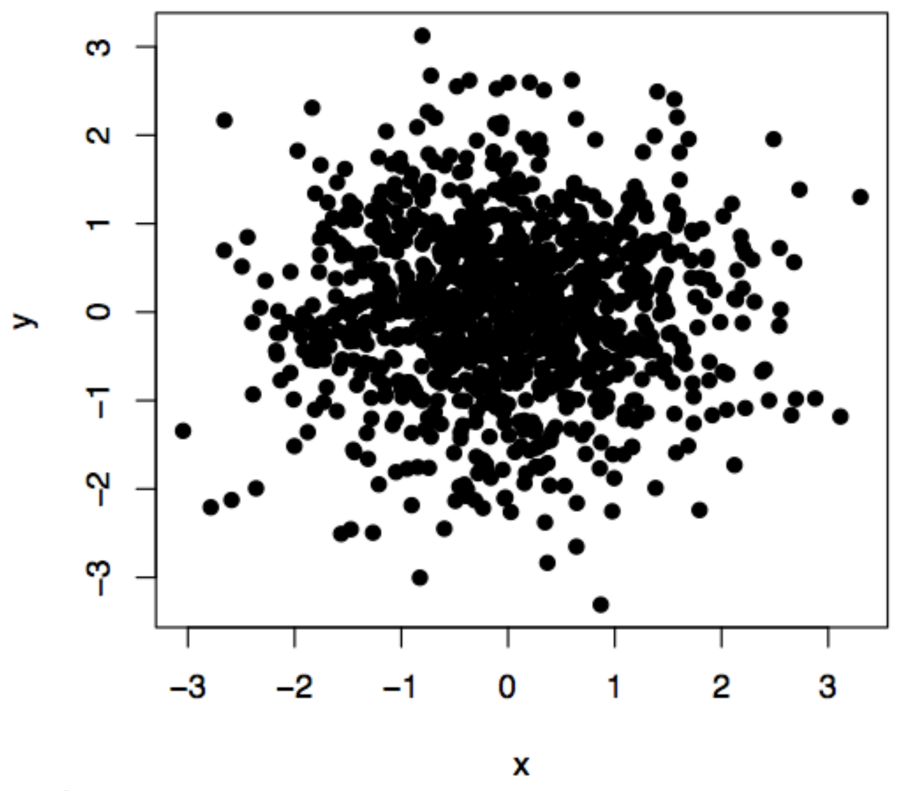
```
x <- rnorm(10000)
y <- rnorm(10000)
smoothScatter(x, y)
```



Some other plotting notes

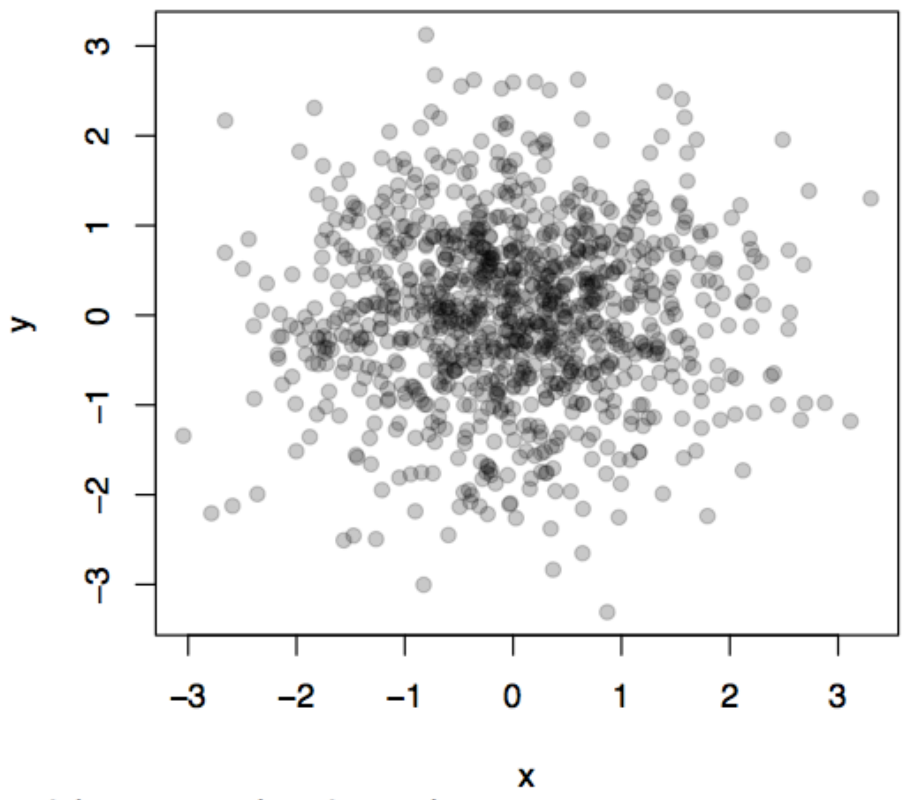
- The `rgb` function can be used to produce any color via red, green, blue proportions
- Color transparency can be added via the `alpha` parameter to `rgb`
- The `colorspace` package can be used for a different control over colors

Scatterplot with no transparency



```
plot(x, y, pch = 19)
```

Scatterplot with transparency



```
plot(x, y, col = rgb(0, 0, 0, 0.2), pch = 19)
```

Summary

- Careful use of colors in plots/maps/etc. can make it easier for the reader to get what you're trying to say (why make it harder?)
- The `RColorBrewer` package is an R package that provides color palettes for sequential, categorical, and diverging data
- The `colorRamp` and `colorRampPalette` functions can be used in conjunction with color palettes to connect data to colors
- Transparency can sometimes be used to clarify plots with many points