Unit 9: Bootstrap and Clustering

Agenda

1. Bootstrap method
2. Clustering

Next Week

- Network theory (application)
- GLM-Classification-image recognition

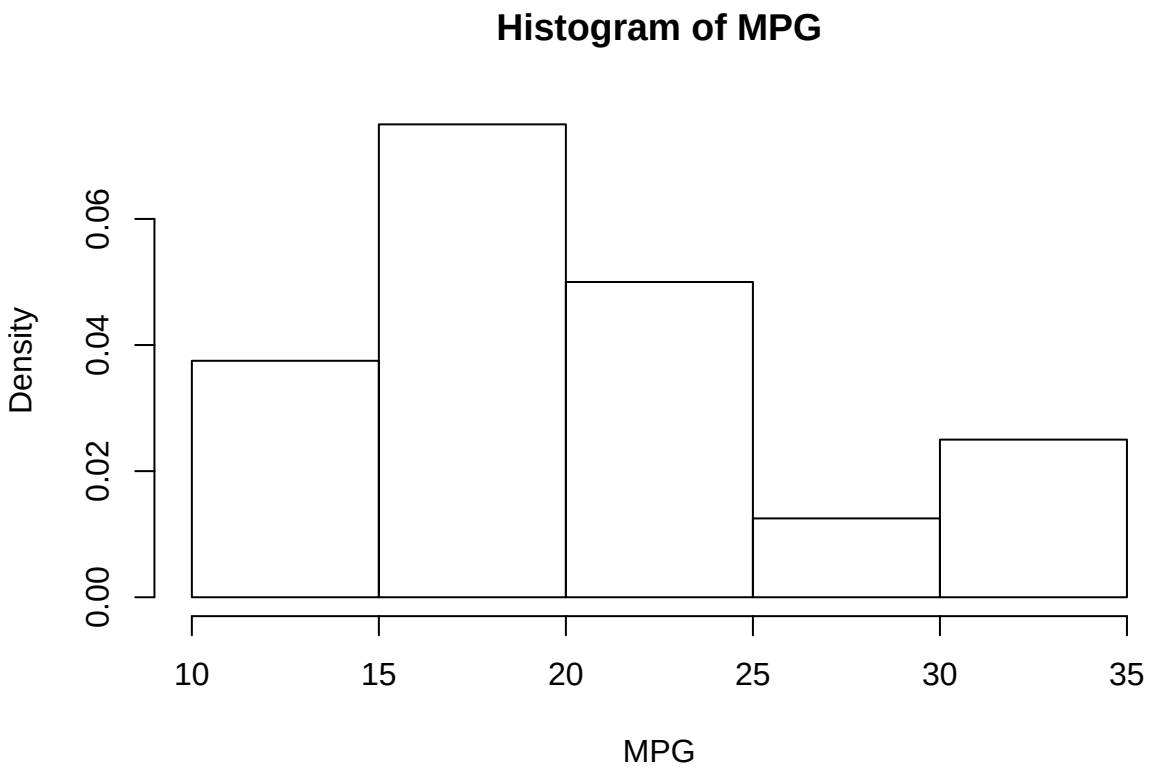
Bootstrap Resampling

- Sample from the dataset with replacement to generate new samples
- the data estimates its own distribution
- well grounded theoretically, should have $n > 20$
- let's do a simple example with `mtcars` dataset

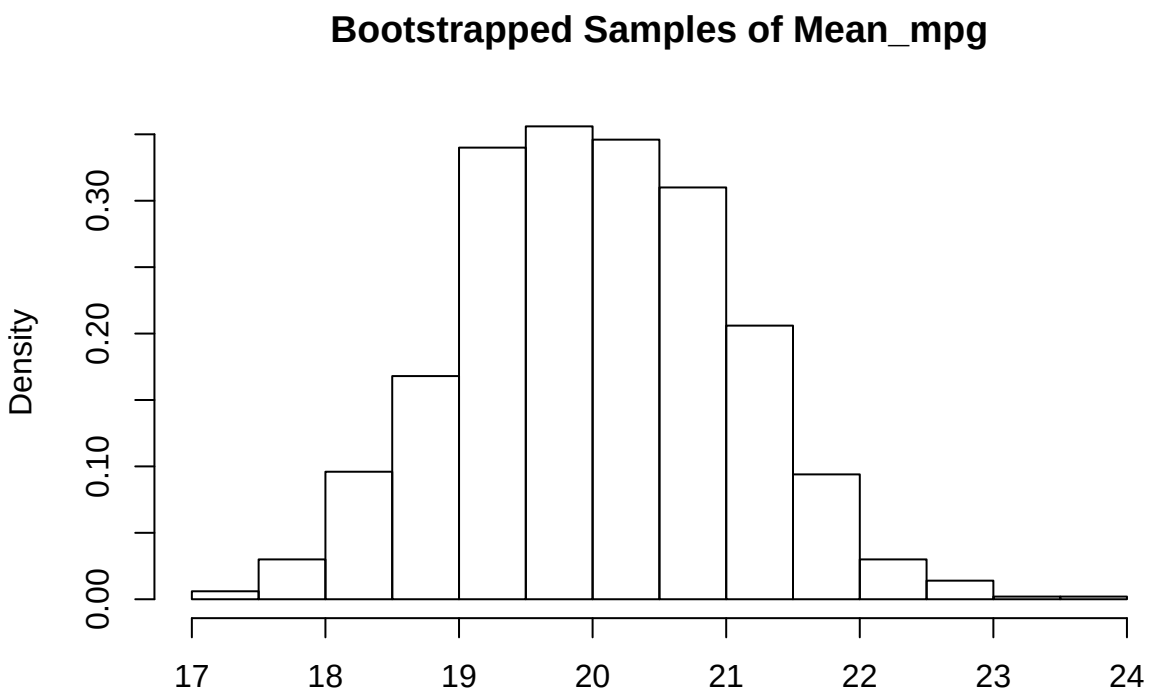
```
data("mtcars")
mpg = mtcars$mpg
n = length(mpg)
print(mean(mpg))
```

```
## [1] 20.09062
```

```
hist(x = mpg, probability = TRUE, xlab = "MPG", main = "Histogram of MPG")
```

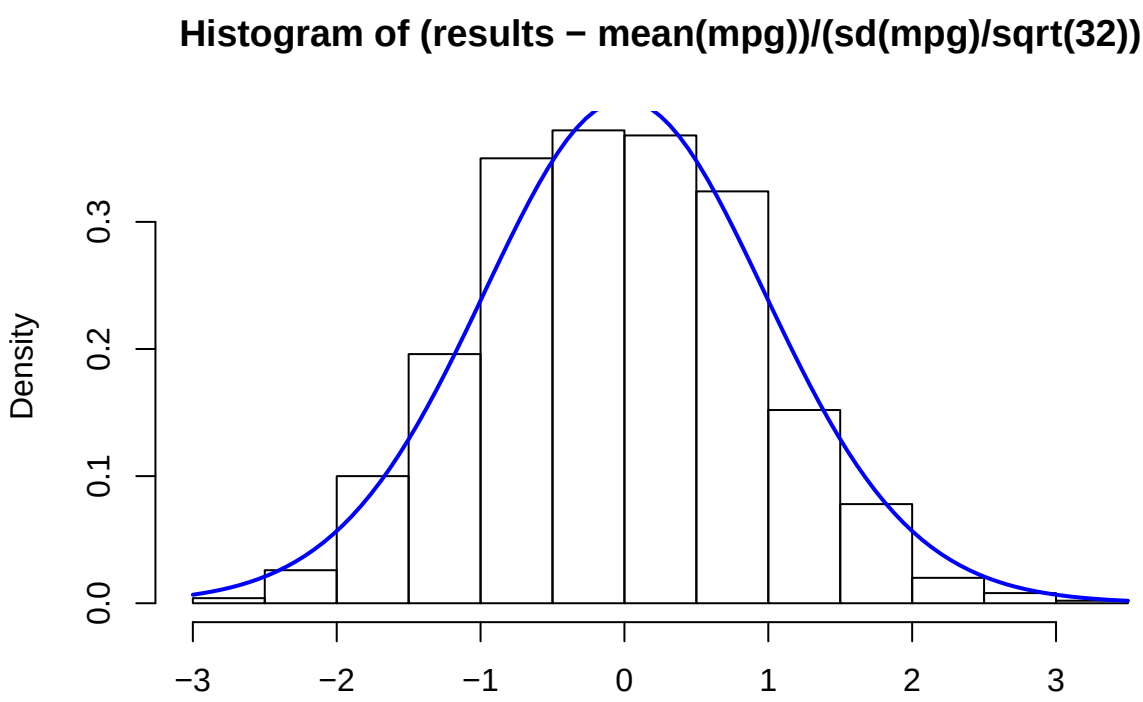


```
results = numeric(1000) ## vector to hold results of 1000 bootstraps
for(b in 1:1000){
  i = sample(x = 1:n, size = n, replace = TRUE) ## sample indices
  bootSample = mpg[i] ## get data
  xHat = mean(bootSample) ## calculate the mean for bootstrap sample
  results[b] = xHat} ## store results
```



This bootstrap result is actually very close to what you would get if you took the theoretical approach with $\hat{\sigma} = S$

$\frac{\overline{X} - \mu_0}{\frac{\sigma}{\sqrt{n}}} \sim t(n-1)$



curve(dt(x,4), col="blue", lwd=2, add=TRUE, yaxt="n")

Clustering

1. Clustering is a broad set of techniques for finding subgroups of observations within a data set.
2. When we cluster observations, we want observations in the same group to be similar and observations in different groups to be dissimilar.
3. Because **there isn't a response variable**, this is an unsupervised method, which implies that it seeks to find relationships between the n observations without being trained by a response variable.
4. K-means clustering is the simplest and the most commonly used clustering method for splitting a dataset into a set of k groups.

Clustering in R

To setup you will need

```
library(tidyverse)  # data manipulation
library(cluster)    # clustering algorithms
library(factoextra) # clustering algorithms & visualization
```

To perform a cluster analysis in R, generally,

1. Rows are observations (individuals) and columns are variables
2. Any missing value in the data must be removed or estimated

3. The data must be standardized (i.e., scaled) to make variables comparable. Recall that, standardization consists of transforming the variables such that they have mean zero and standard deviation one.

---

data: we'll use the built-in R data set `USArrests` which contains the arrests per 100,000 residents in each state as well as the percentage of the population which is urban

```
df <- USArrests

df <- na.omit(df) # remove any missing values

head(df)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

Now we use the `scale()` function to standardize data

```
df <- scale(df)
head(round(df,2))
```

```
##            Murder Assault UrbanPop  Rape
## Alabama      1.24    0.78    -0.52  0.00
## Alaska       0.51    1.11    -1.21  2.48
## Arizona      0.07    1.48     1.00  1.04
## Arkansas     0.23    0.23    -1.07 -0.18
## California   0.28    1.26     1.76  2.07
## Colorado     0.03    0.40     0.86  1.86
```

Defining distance measure

The choice of distance measures is a critical step in clustering. It defines how the similarity of two elements (x, y) is calculated and it will influence the shape of the clusters.

Euclidean distance:

$$d_e(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Manhattan distance:

$$d_m(x,y) = \sum_{i=1}^{n}|x_i - y_i|$$

Pearson correlation distance:

$$d_c(x,y) = 1 - \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

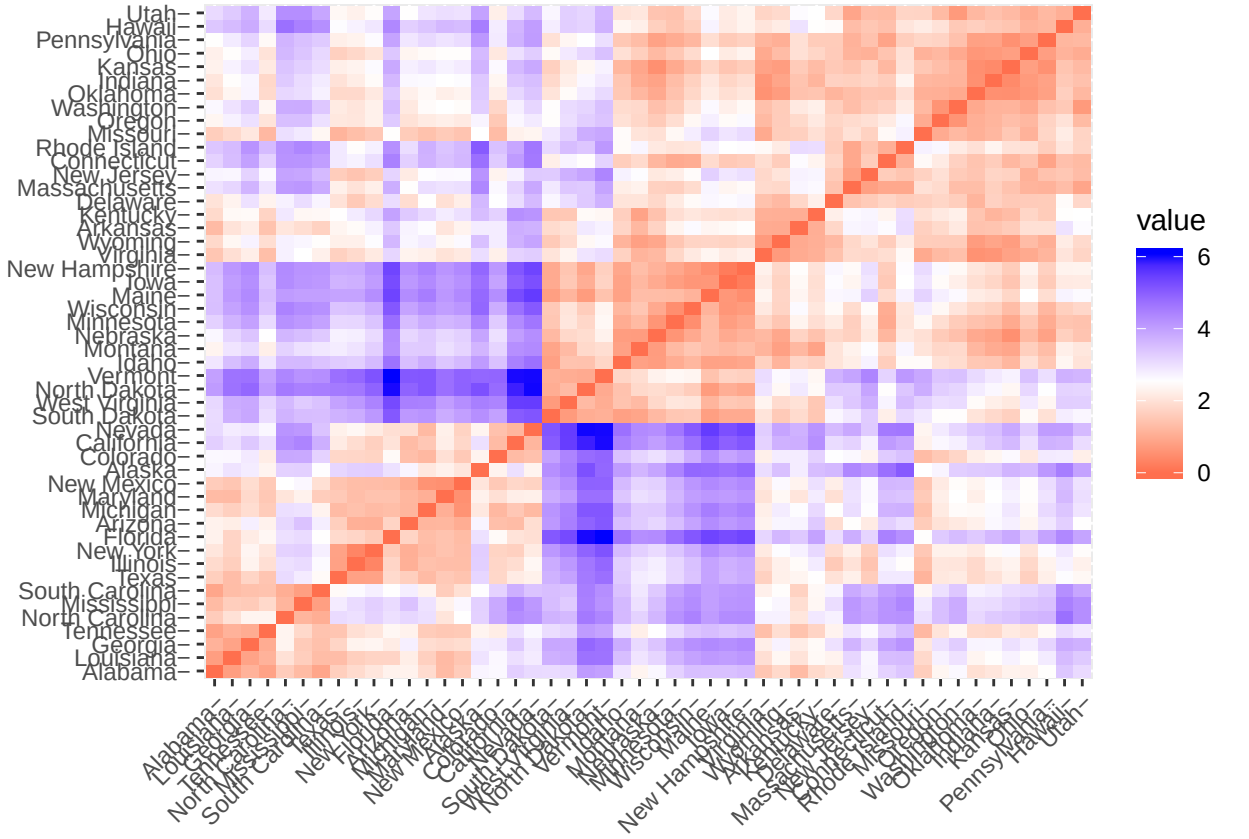
notice that for this correlation metric, the summation index is over the columns, whereas previously when we calculated the correlation **betweeen two variables** we summed over rows instead of columns!

For most common clustering software, the default distance measure is the Euclidean distance, with `factoextra` you can use all of the above distance measures

```
distance <- get_dist(df)
```

You can see a visualiztaion of the distances with `fviz_dist()`

```
fviz_dist(distance)
```



k-means clustering

- algorithm for partitioning data set into a set of k groups
- k represents the number of groups specified by analyst
- Each cluster is represented by its center (i.e, centroid) which corresponds to the mean of points assigned to the cluster
- The basic idea behind k-means clustering consists of defining clusters so that the total intra-cluster variation (known as total within-cluster variation) is minimized

$$W(C_k) = \sum_{x_i \in C_k} (x_i - \mu_k)^2$$

where μ_k is the centroid of the observations in kth cluster

---

we want to minimize the total SS over all clusters C_k

$$TWCSS = \sum_{k=1}^{k} W(C_k) = \sum_{k=1}^{k} \sum_{x_i \in C_k} (x_i - \mu_k)^2$$

the total-within-cluster-sum-square distances from centroids, is a measure of *goodness of clustering* similar to sum of square residuals in regression. We want this to be small but not have too many clusters (e.g. don't overfit/overcluster)!!

k means is an algorithm to find this minimum given k

the steps of the algorithm are...

---

1. Randomly assign the k centroids (means) to k observations from data
2. Assigns each observation to their closest centroid, based on the Euclidean distance between the object and the centroid
3. For each of the k clusters update the cluster centroid by calculating the new mean values of all the data points in the cluster. The centroid of the kth cluster is a vector of length p containing the means of all variables for the observations in the kth cluster; p is the number of variables
4. Iterate steps 3 and 4 until the cluster assignments stop changing or the maximum number of iterations is reached.

Computing k-means in R

We can compute k-means in R with the `kmeans` function

- Here will group the data into two clusters (centers = 2)
- The kmeans function also has an `nstart` option that attempts multiple initial configurations and reports on the best one.

The output `k2$cluster` gives a list of which cluster each observation belongs

```
k2 <- kmeans(df, centers = 2, nstart = 25)
k2$cluster
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              1              1              2              1
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              1              2              2              1              1
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              2              2              1              2              2
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              2              2              1              2              1
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              2              1              2              1              1
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
```

```
##              2              2              1              2              2
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              1              1              1              2              2
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              2              2              2              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              2              1              1              2              2
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              2              2              2              2              2
```

```
k2$centers # locations of centroids (means)
```

```
##      Murder    Assault   UrbanPop       Rape
## 1  1.004934  1.0138274  0.1975853  0.8469650
## 2 -0.669956 -0.6758849 -0.1317235 -0.5646433
```

`totss`: total sum of squares from mean of data (e.g. $k = 1$)

`withinss`: Vector of within-cluster sum of squares, one component per cluster.

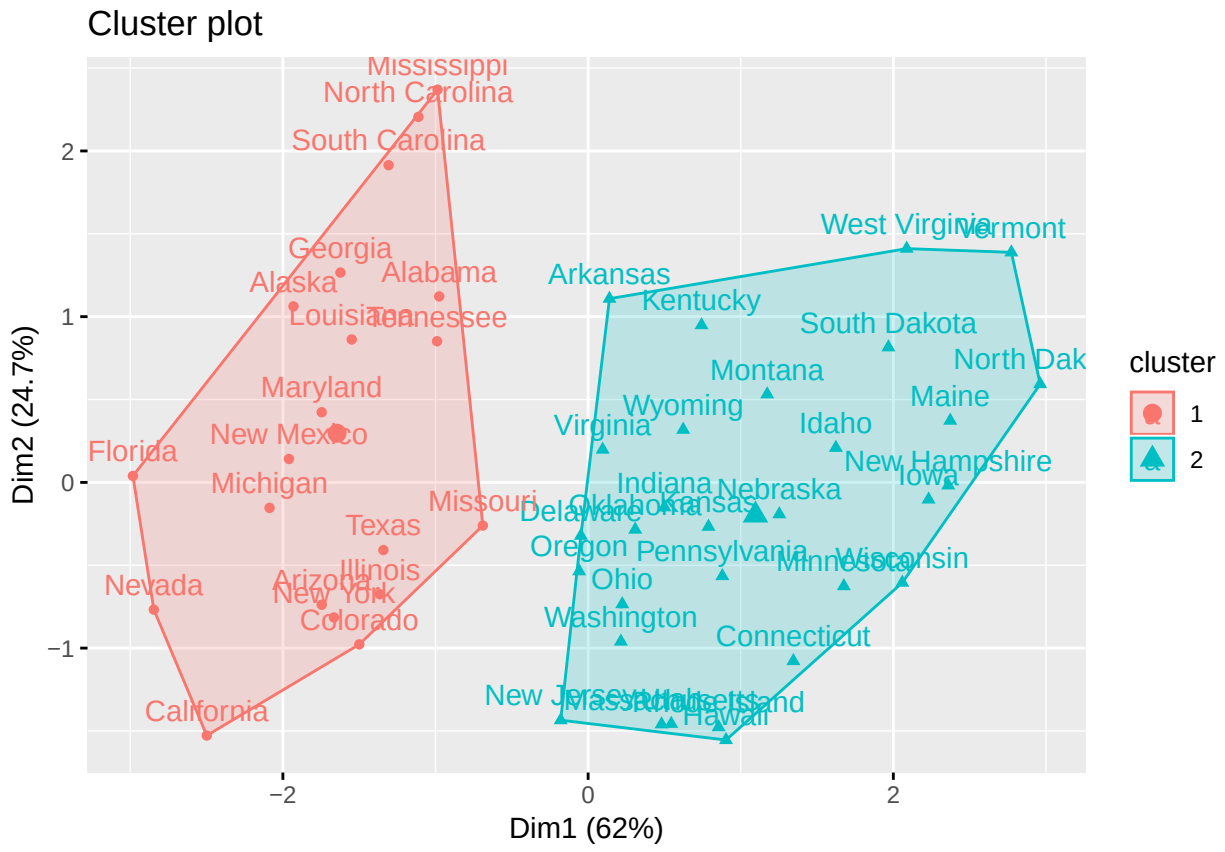
`tot.withinss`: Total within-cluster sum of squares, i.e. sum(withinss).

`betweenss`: The between-cluster sum of squares, i.e. $totss - tot.withinss$.

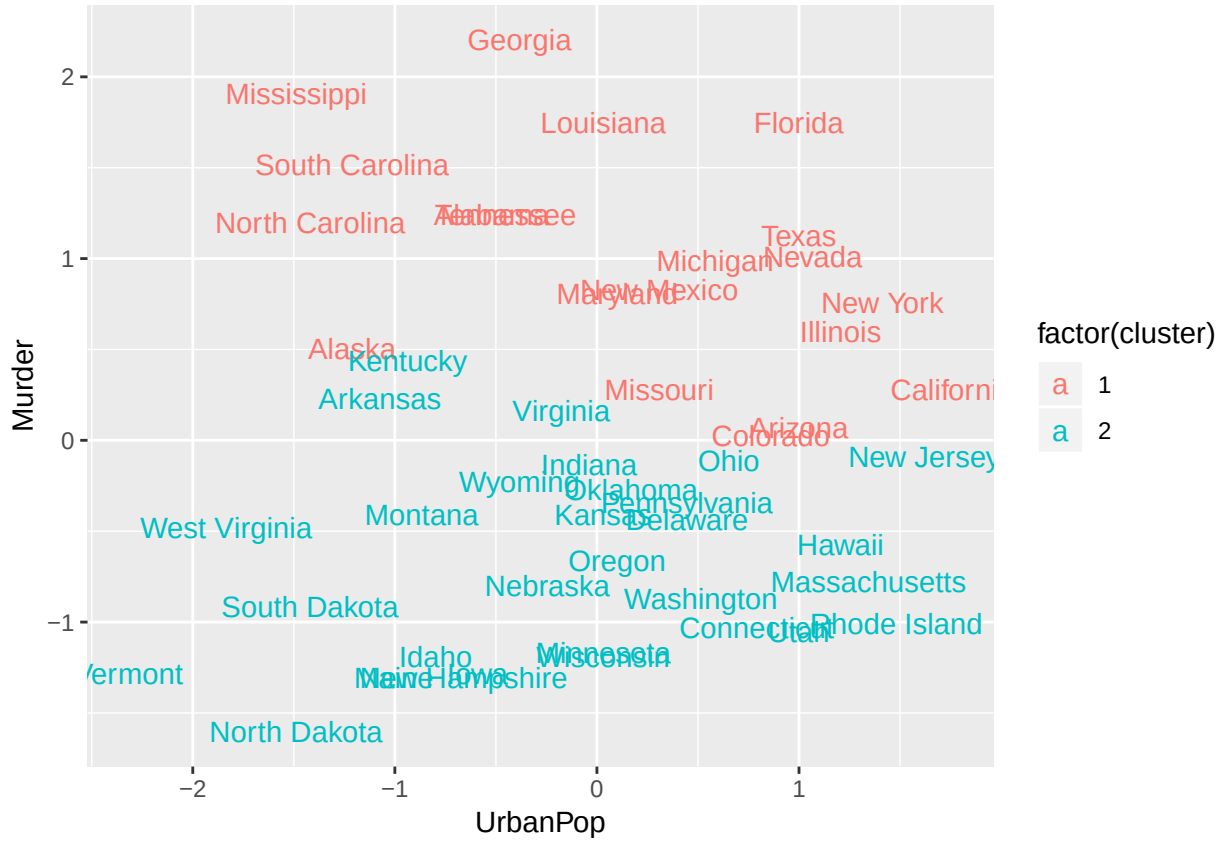
`size`: The number of points in each cluster.

You can visualize the result with `fviz_cluster()`

```
fviz_cluster(k2, data = df)  # axes are top two comp from PCA
```



```
df %>% as_tibble() %>% mutate(cluster = k2$cluster,
   state = row.names(USArrests)) %>% ggplot(aes(UrbanPop, Murder,
    color = factor(cluster), label = state)) + geom_text()
```



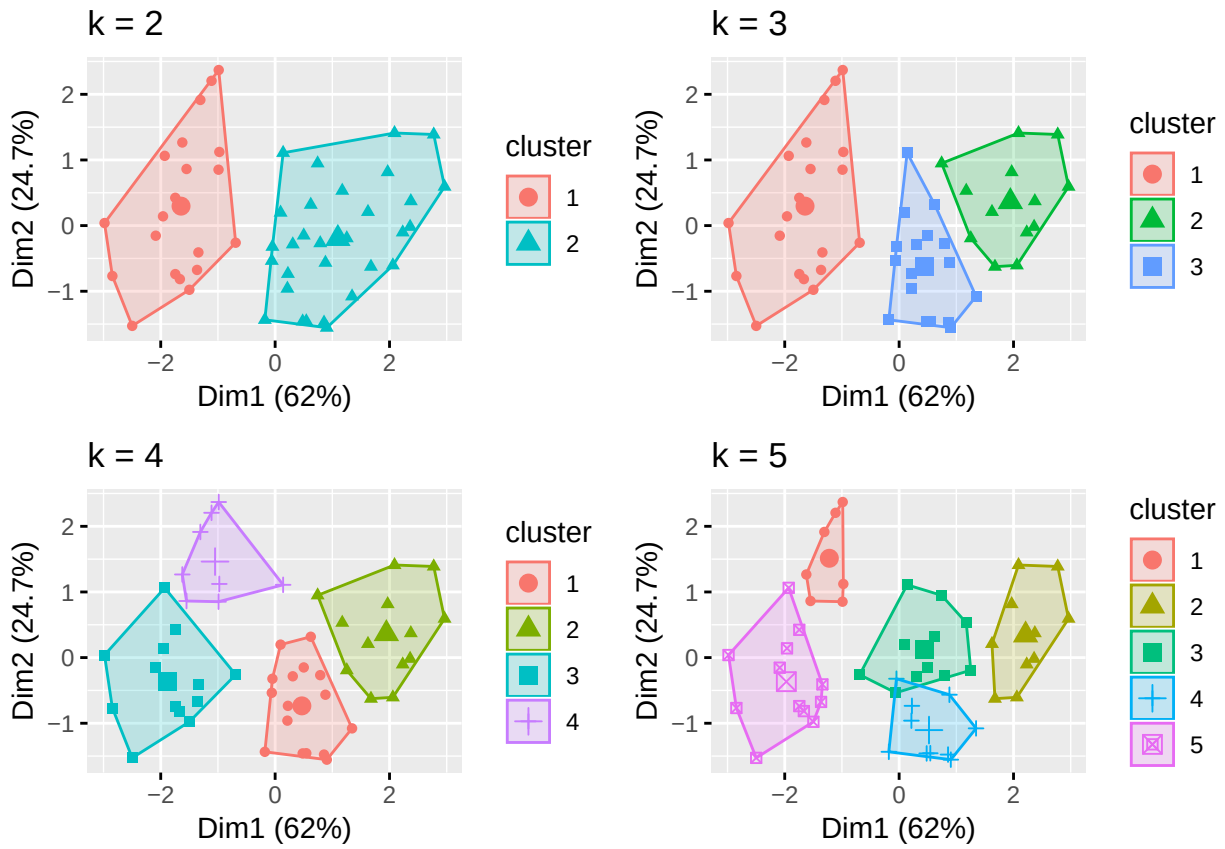
Because the number of clusters (k) must be set before we start the algorithm, it is often advantageous to use several different values of k and examine the differences in the results.

We can execute the same process for 3, 4, and 5 clusters, and the results are shown in the figure on next slide.

```
k3 <- kmeans(df, centers = 3, nstart = 25)
k4 <- kmeans(df, centers = 4, nstart = 25)
k5 <- kmeans(df, centers = 5, nstart = 25)

# plots to compare
p1 <- fviz_cluster(k2, geom = "point", data = df) + ggtitle("k = 2")
p2 <- fviz_cluster(k3, geom = "point",  data = df) + ggtitle("k = 3")
p3 <- fviz_cluster(k4, geom = "point",  data = df) + ggtitle("k = 4")
p4 <- fviz_cluster(k5, geom = "point",  data = df) + ggtitle("k = 5")
```

```
library(gridExtra)
grid.arrange(p1, p2, p3, p4, nrow = 2)
```



What should k be?

Similar to choosing how many parameters β we need some criteria, the following is the *elbow method*

1. Compute clustering algorithm (e.g., k-means clustering) for different values of k. For instance, by varying k from 1 to 10 clusters
2. For each k, calculate the total within-cluster sum of square (wss)
3. Plot the curve of wss according to the number of clusters k.
4. The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

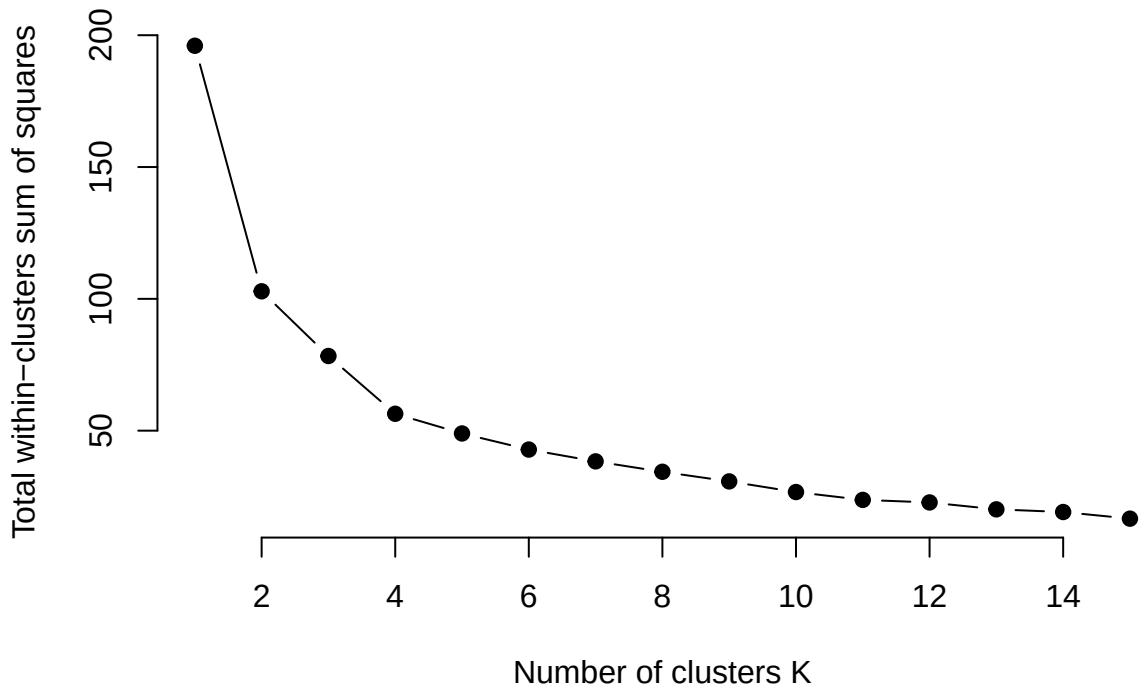
Here is some R code to execute the elbow method

```
# function to compute total within-cluster sum of square
wss <- function(k) {
  kmeans(df, k, nstart = 10 )$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k.values <- 1:15

# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss) # applies wss function
```

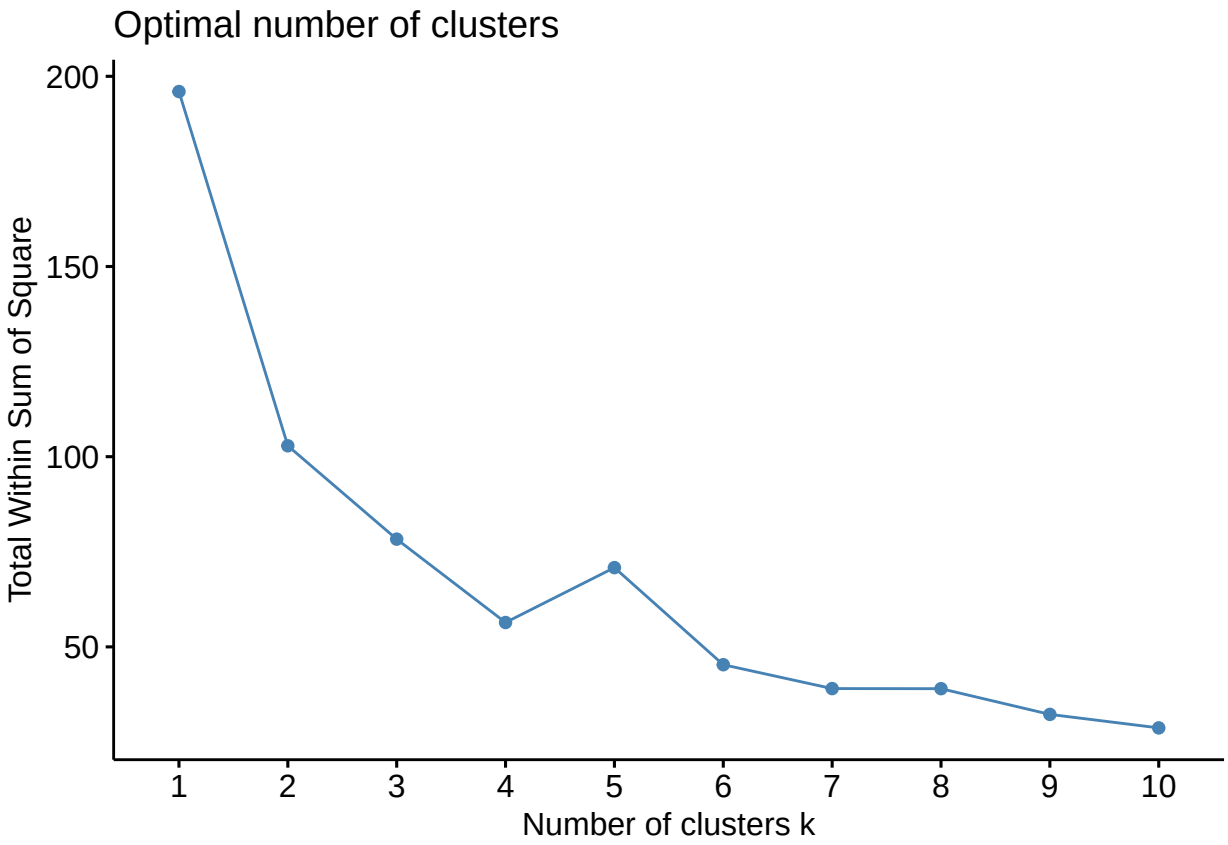
```r
plot(k.values, wss_values,
       type="b", pch = 19, frame = FALSE,
       xlab="Number of clusters K",
       ylab="Total within-clusters sum of squares")
```



This suggests that $k = 4$ is the optimal number of clusters.

Fortunately, this process is wrapped up in `fviz_nbclust()`

```r
fviz_nbclust(df, kmeans, method = "wss")
```



Heierarchal Clustering

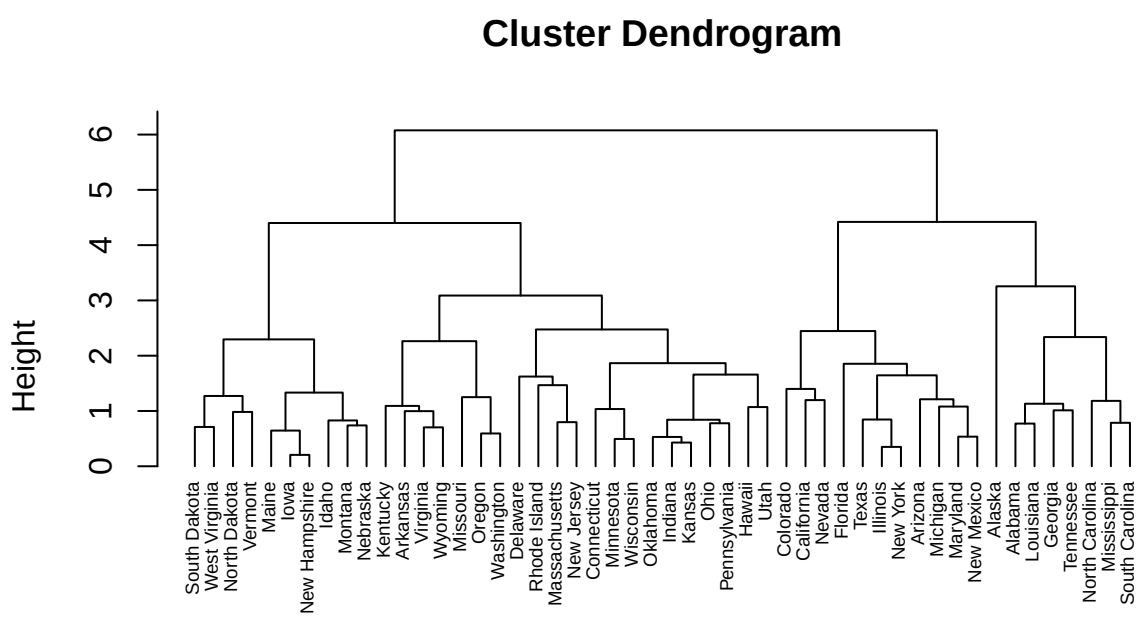
- **Agglomerative clustering**:

This is a "bottom-up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. A notion of dissimilarity is needed between clusters. **There are various definitions based on the set of pairwise inter-distances between two clusters**. The most similar two clusters based on the criteria are linked first.

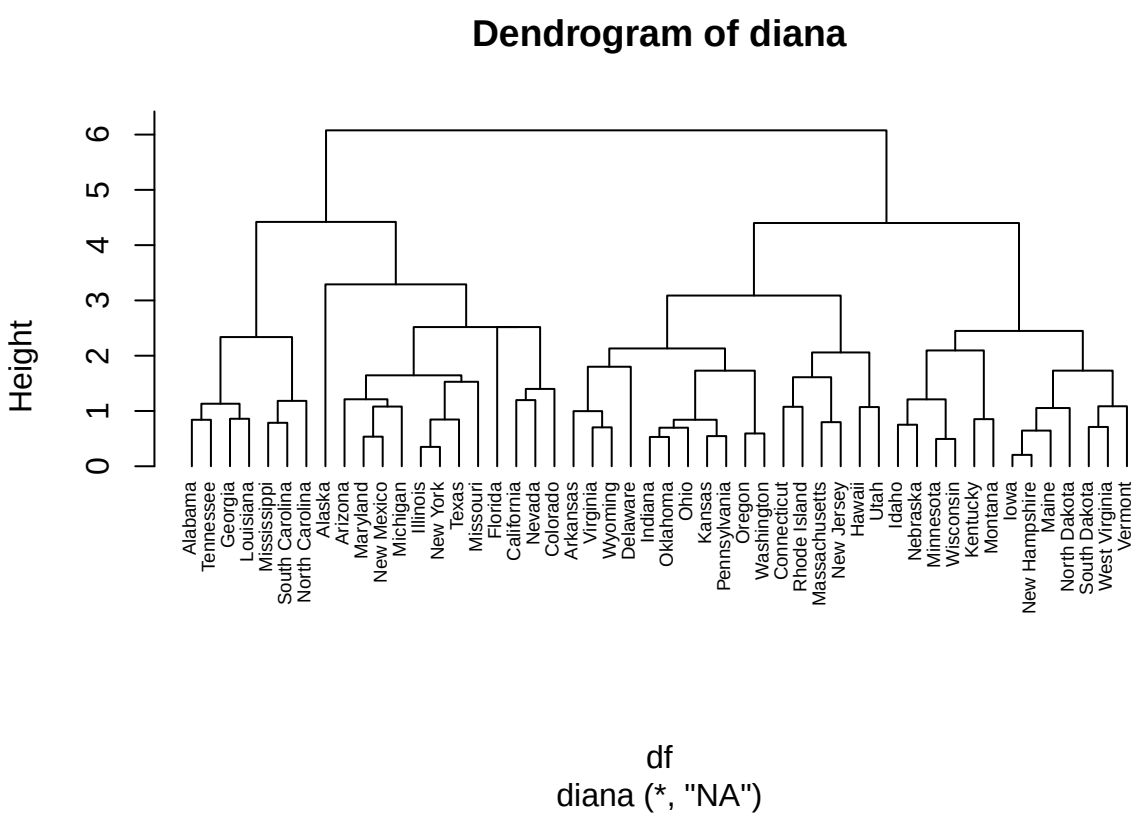
- **Divisive hierarchical clustering(DIANA)**:

This is a "top-down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy. DIANA chooses the object with the maximum average dissimilarity and then moves all objects to this cluster that are more similar to the new cluster than to the remainder. The most dissimilar groups are split first.

```
d <- dist(df, method = "euclidean")
hc1 <- hclust(d, method = "complete" ) # link metric (max pairwise)
plot(hc1, cex = 0.6, hang = -1)  # plot dendrogram
```



```r
hc4 <- diana(df) # divisive
pltree(hc4, cex = 0.6, hang = -1, main = "Dendrogram of diana")
```



Clustering Exercises

Suppose you were given a large numerical dataset with 10 million observations and each observations has 10,000 features. All values are real numbers.

1. Suppose that you want to find all the vectors in the dataset that are at distance less than or equal to unity from a given vector. How would you write an efficient algorithm to find all such vectors? (say, using Euclidean distance)
2. Suppose you wanted to reduce the dataset to a smaller data set. How would you go about that?