Multivariate Stats Week 03: Clustering & MDS

Noah Silbert

9/8/2017

To Do

- Homework review
- Hierarchical clustering
- k-means clustering
- Multidimensional Scaling

Proximity

Before discussing clustering and multidimensional scaling, we need to talk about proximity (distance) and similarity.

The Euclidean distance between to points $\mathbf{x}$ and $\mathbf{y}$ is:

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &= \sqrt{(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})} \\ &= \sqrt{\sum_{i=1}^{P} (x_i - y_i)^2} \end{aligned}$$

Note that if you have N observations of P variables, calculating the distance between each pair of observations will give you an $N \times N$ square, symmetric matrix.

Proximity continued

The *statistical* distance is:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{S}^{-1} (\mathbf{x} - \mathbf{y})}$$

By "dividing" by the covariance matrix (i.e., putting its inverse in the middle of the product of the difference of the two vectors), distances are normalized with respect to the (co-)variance of the variables.

We could also standardize each variable prior to calculating distances. If the original variables are on very disparate scales, this can be useful.

Similarity

The similarity between two points is a decreasing function of the proximity between the points (i.e., as proximity increases, similarity decreases, and vice versa).

There isn't a single, fixed measure of similarity. If you want to measure the similarity between pairs of *variables* (as opposed to pairs of *observations*), correlation can be used.

For measuring similarity between pairs of observations, there are a number of options. One is just the reciprocal of the distance, i.e., $\frac{1}{d_{ij}}$. Another (from cognitive psychology) is $s_{ij} = \exp\left(-d_{ij}^2\right)$.

We will (probably) return to this issue later on.

Hierarchical Clustering

Hierarchical clustering is a family of algorithms that are useful for exloring the structure of a multidimensional data set. Specific types of hierarchical clustering just require that distances can be calculated between pairs of observations (or variables).

There are two general approaches to clustering: agglomerative and divisive.

In the former, you start with every observation in its own "cluster", and on each step of the algorithm, some rule determines which clusters at step i should be merged into larger clusters at step $i + 1$. At the end, a single cluster contains the entire data set.

Divisive approaches go in the opposite directing, starting with a single cluster and using a rule to determine when and where to split clusters at step i into smaller clusters on step $i + 1$.

Single Linkage or Nearest Neighbor

In single linkage (or nearest neighbor) hierarchical clustering, the distance between two clusters A and B is the minimum distance between a point in A and a point in B:

$$D(A, B) = \min\Big(d(\mathbf{y}_i, \mathbf{y}_j); i \in A, j \in B\Big)$$

At each step of the algorithm, the nearest two clusters are merged into a new cluster.

Crime example adapted from the book

We can read the data in using `read.table()`, which gives us a `data.frame`. The data file doesn't have a header row, so we can use the information from the book to name the columns, and then we can see the first `n` (default 6) rows of the `data.frame` with the function `head()`:

```
df = read.table("T15_1_CITYCRIME.dat")
colnames(df) = c("City","Murder","Rape","Robbery","Assault","Bur
head(df)
```

```
##      City Murder Rape Robbery Assault Burglary Larceny AutoTh
## 1 Atlanta   16.5 24.8     106     147     1112     905
## 2  Boston    4.2 13.3     122      90      982     669
## 3 Chicago   11.6 24.7     340     242      808     609
## 4  Dallas   18.1 34.2     184     293     1668     901
## 5  Denver    6.9 41.5     173     191     1534    1368
## 6 Detroit   13.0 35.7     477     220     1566    1183
```

Calculating distances

We can use the the function `dist()` to calculate distances between each pair of rows in the data frame (after turning the city names into row names for a matrix):

```
nc = ncol(df)
M = as.matrix(df[,2:nc])
rownames(M) = df[,1]
D = dist(M)
D
```

```
##                Atlanta    Boston   Chicago    Dallas    Denver
## Boston       536.6419
## Chicago      516.3700  447.4033
## Dallas       590.1753  833.0708  924.0035
## Denver       693.5741  914.9784 1073.3948  527.6673
## Detroit      716.1962  881.0858  971.5271  464.4677  358.6654
## Hartford     215.1604  493.5120  426.4688  723.5541  893.9948
## Honolulu     443.6810  723.7733  893.9362  420.9614  376.4364
## Houston      497.7503  628.6037  729.0793  264.2640  599.0626
## KC           500.6776  645.5878  782.6467  263.4868  424.6894
## LA          1033.1139 1211.4398 1365.3121  610.0896  426.9086
```

Hierarchical Clustering

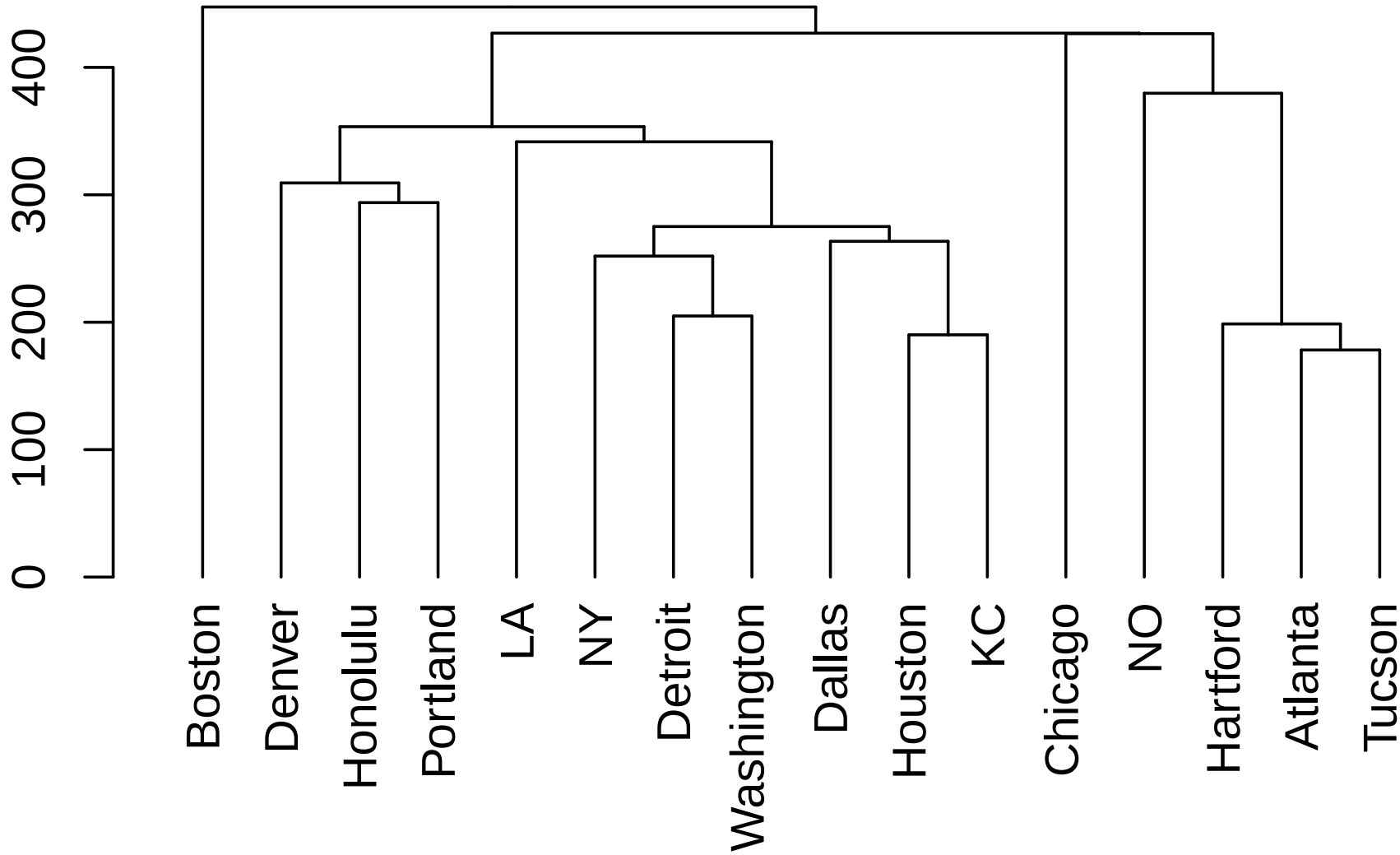
The function `hclust()` takes a distance matrix as input and returns a hierarhical clustering object. In order to do single linkage clustering, we need to also specify the `method` input argument:

```
slc = hclust(D,method="single")
```

The object `slc` contains information about which clusters were merged, when they were merged, and various other things. Conveniently, we can give this object to the `plot()` function, and it will plot a dendrogram (or tree diagram) to illustrate the clusters.

Single Linkage Dendrogram

```
sld = as.dendrogram(slc)
plot(sld)
```



Other Methods

Complete: $D(A, B) =$ *maximum* distance between elements of A and B.

Average: $D(A, B) =$ average distance between all elements of A and B.

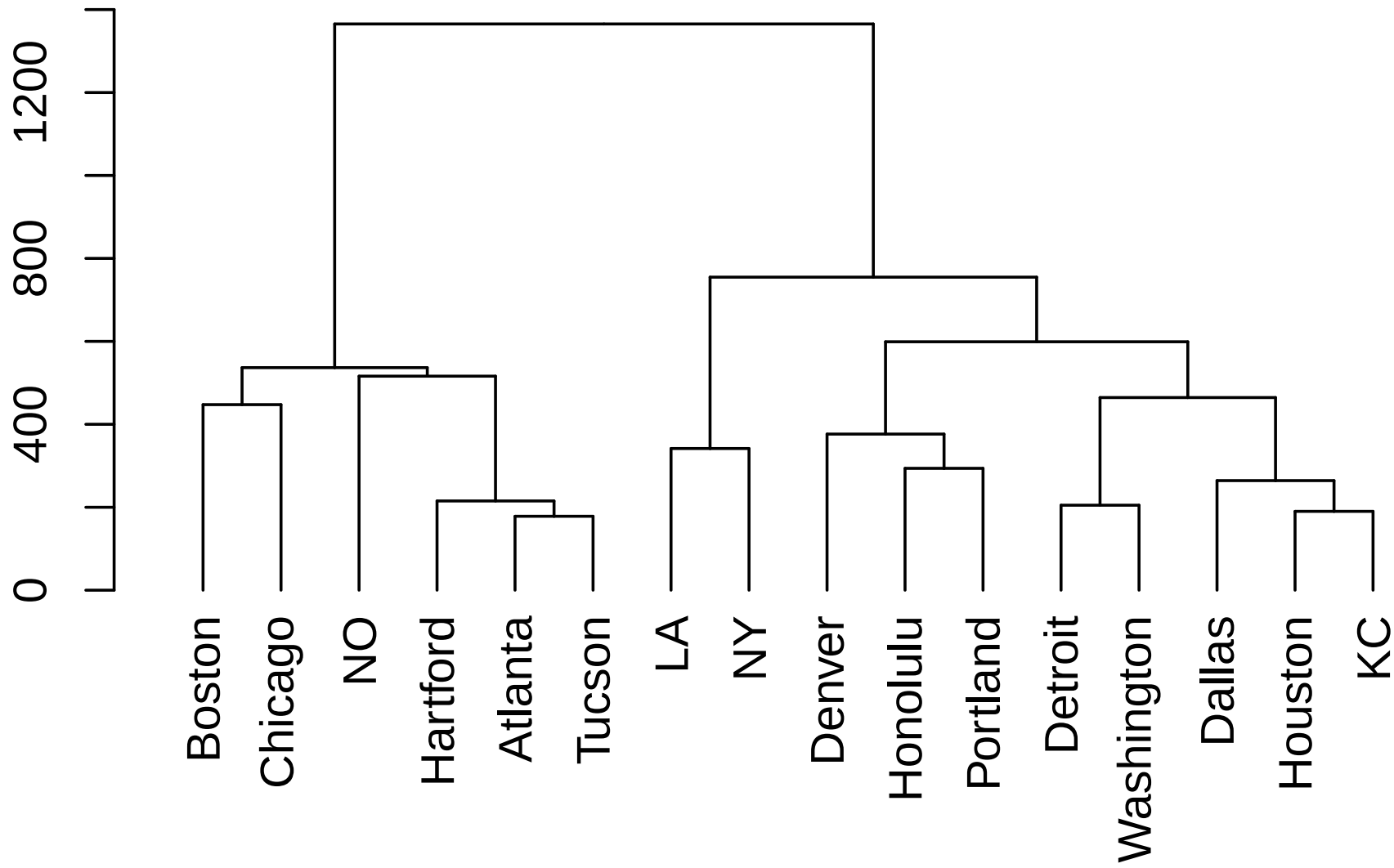
Centroid: $D(A, B) =$ distance between the mean vectors of A and B.

Median: $D(A, B) =$ midpoint of the line joining A and B.

Ward's method determines when to merge clusters based on minimization of the sum of squared error (see p. 518 of the book for the formulas). This method finds compact clusters and produces results similar to the centroid method.

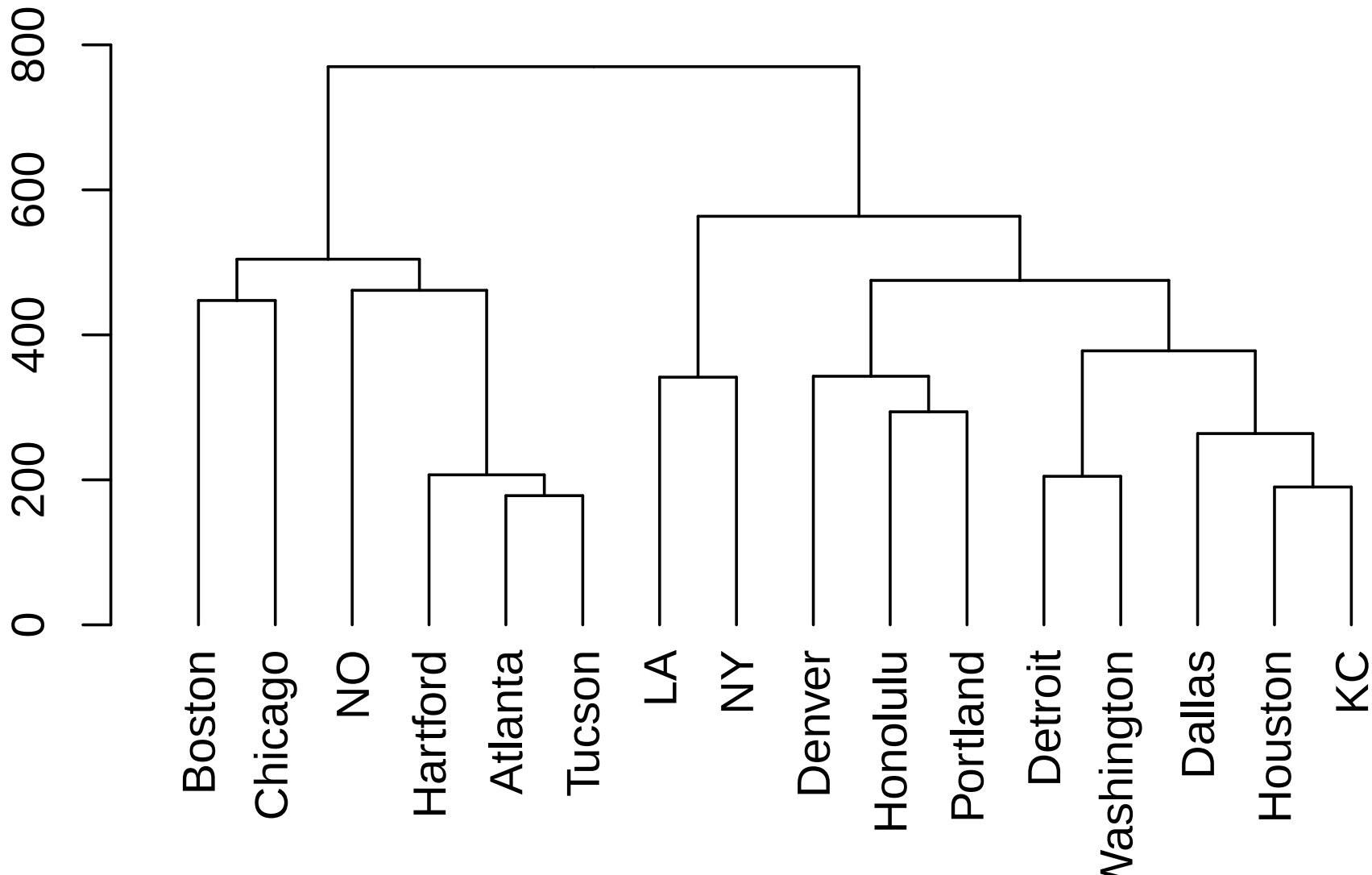
Complete Linkage Dendrogram

```
clc = hclust(D, method="complete")
cld = as.dendrogram(clc)
plot(cld)
```



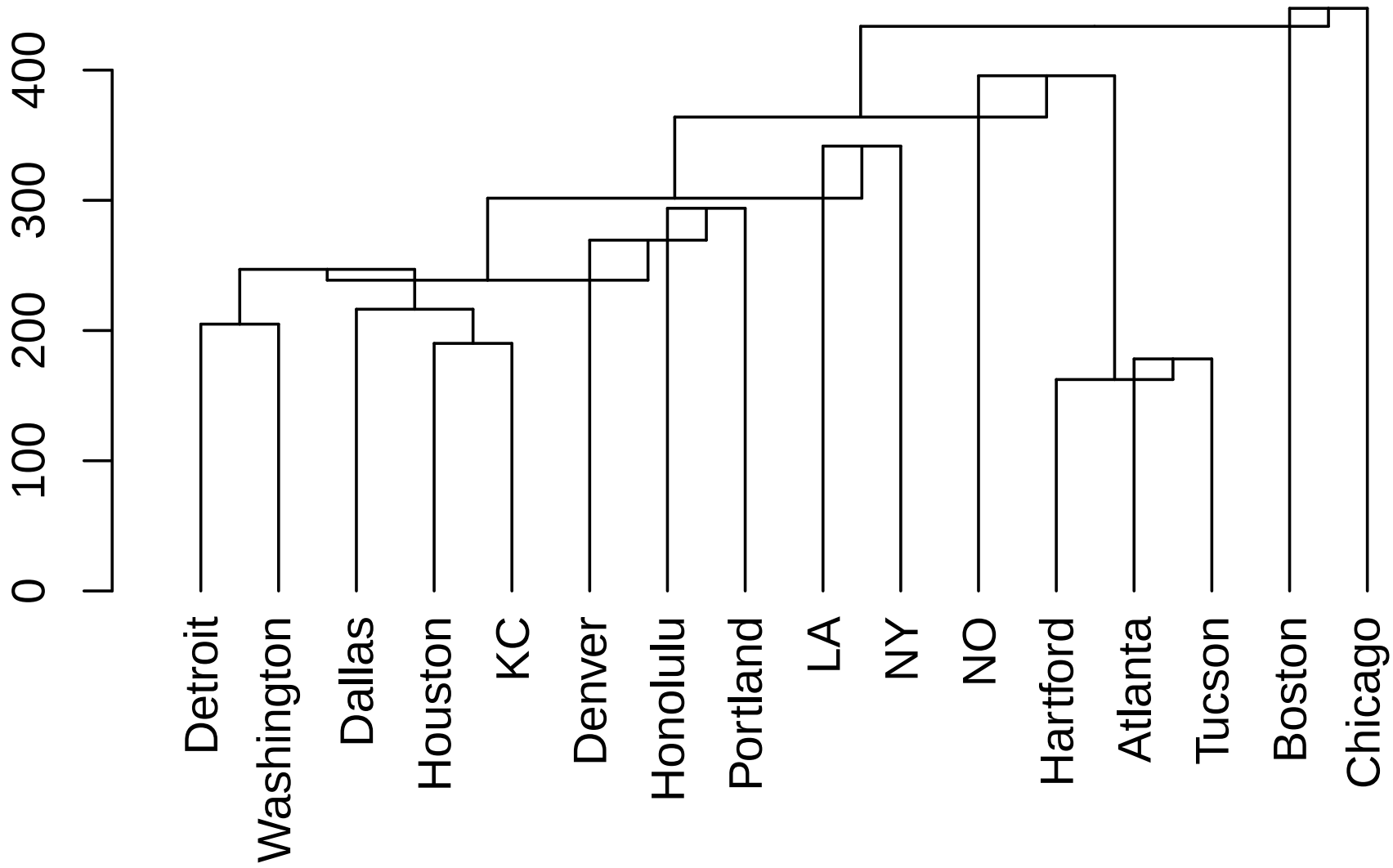
Average Linkage Dendrogram

```
alc = hclust(D, method="average")
ald = as.dendrogram(alc)
plot(ald)
```



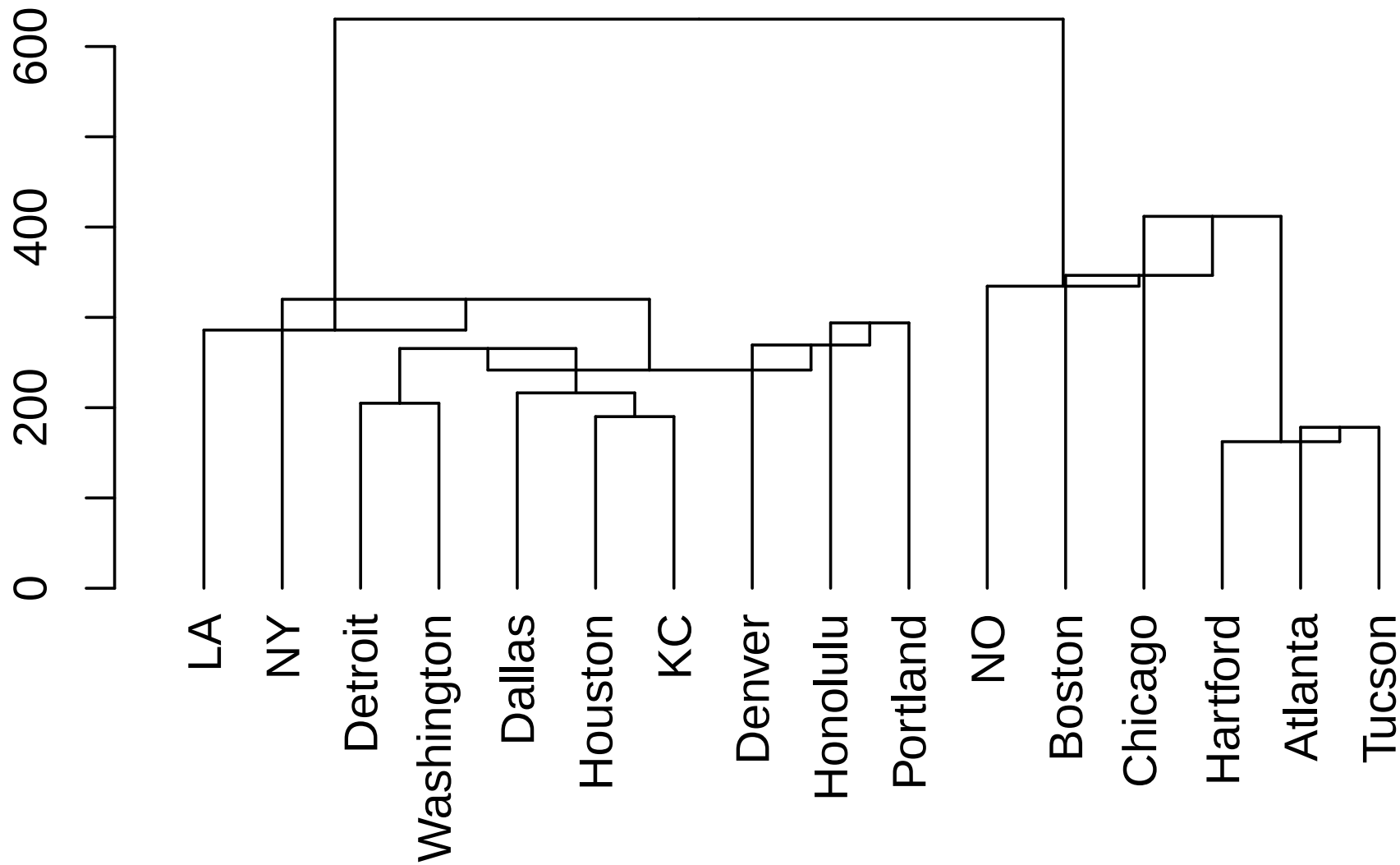
Centroid Linkage Dendrogram

```
nlc = hclust(D, method="centroid")
nld = as.dendrogram(nlc)
plot(nld)
```



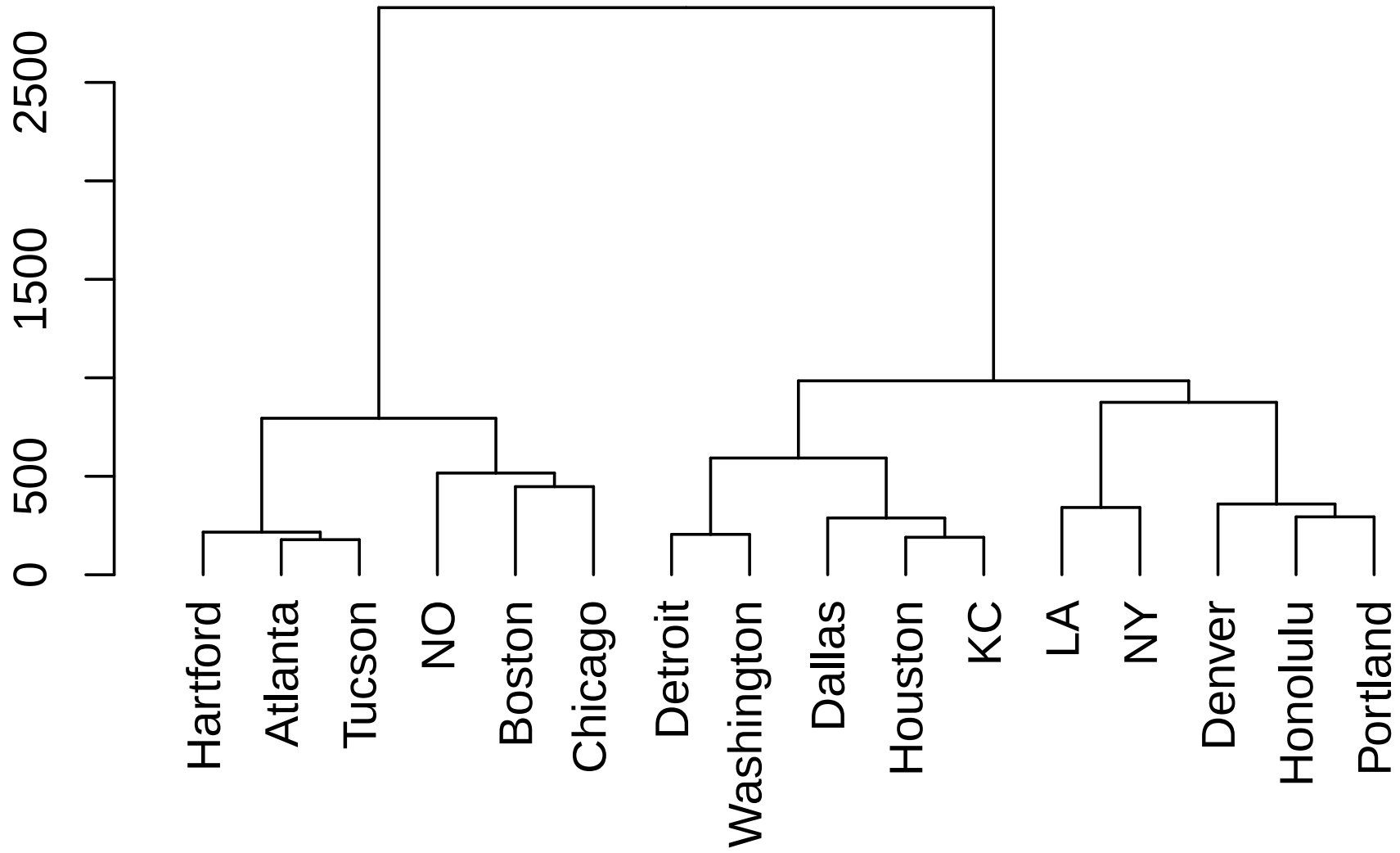
Median Linkage Dendrogram

```
mlc = hclust(D, method="median")
mld = as.dendrogram(mlc)
plot(mld)
```



Ward's Method Dendrogram

```
wlc = hclust(D, method="ward.D")
wld = as.dendrogram(wlc)
plot(wld)
```



Choosing the number of (hierarchical) clusters

You can choose the number of clusters you want by cutting the tree at a given height.

The function `cutree()` takes the object returned by `hclust()` plus either the number of desired clusters `k` or the height at which to cut `h`, and returns a vector indicating the cluster for each observation:

```
w.cut = cutree(wlc, k=2)
w.cut
```

```
##     Atlanta      Boston     Chicago      Dallas      Denver     Det
##           1           1           1           2           2
##    Hartford    Honolulu     Houston          KC          LA
##           1           2           2           2           2
##          NY    Portland      Tucson Washington
##           2           2           1           2
```

Brief interlude about lists

A list in R is a container of any other type of object. The elements of a list can be named or not. If they are named, they can be access by their name. Whether or not they are named, they can be accessed by their index.

```
L = list(n=5,w="string",v=c(1,2,3))
L$v
```

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

```
L[[3]]
```

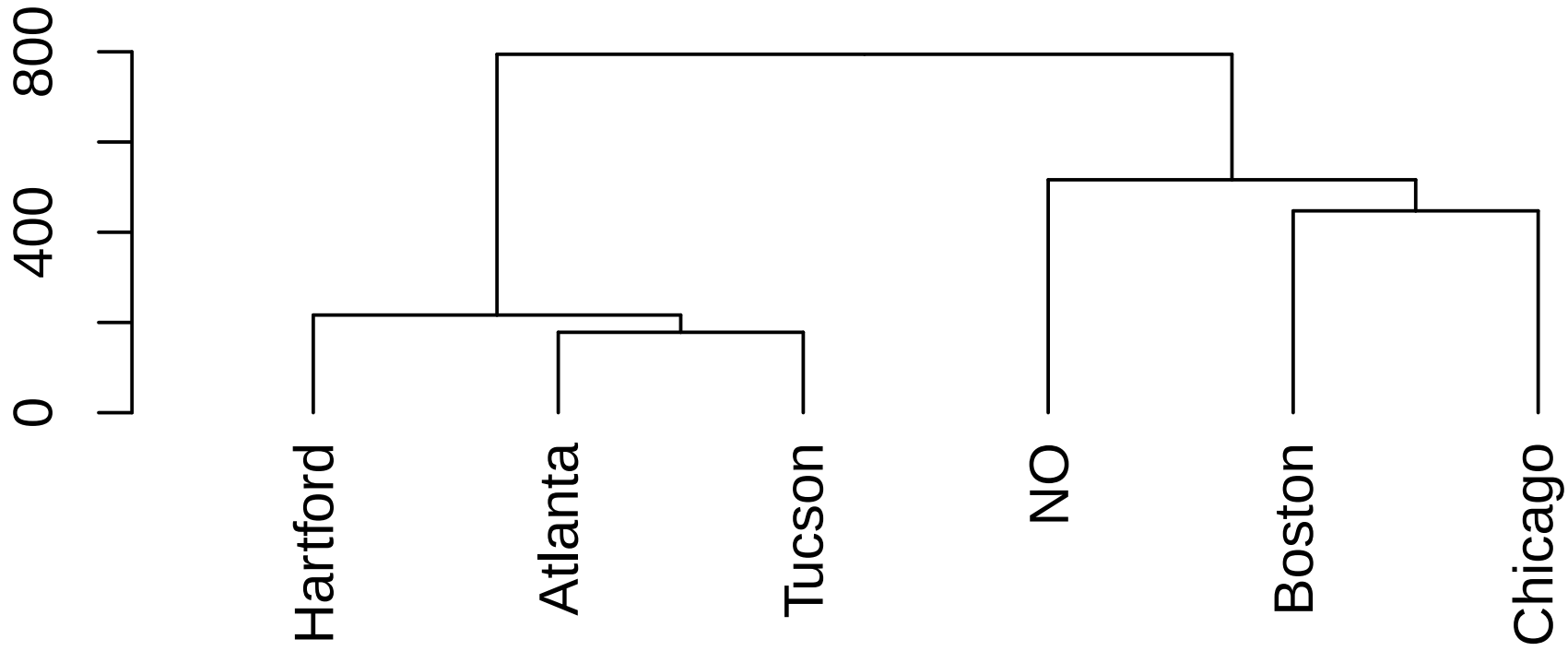
```
## [1] 1 2 3
```

Many functions return lists, so it's useful to know how to get the elements out of the lists.

Choosing the number of (hierarchical) clusters

The function `cut()` (for dendrogram objects) will cut a tree and return a list with the upper and lower clusters:

```
w.cut = cut(wld, h=1250)
plot(w.cut$lower[[1]])
```



k-means Clustering

K-means clustering is a non-hierarchical method that takes pre-selected seeds and a pre-specified number of clusters and iteratively builds clusters.

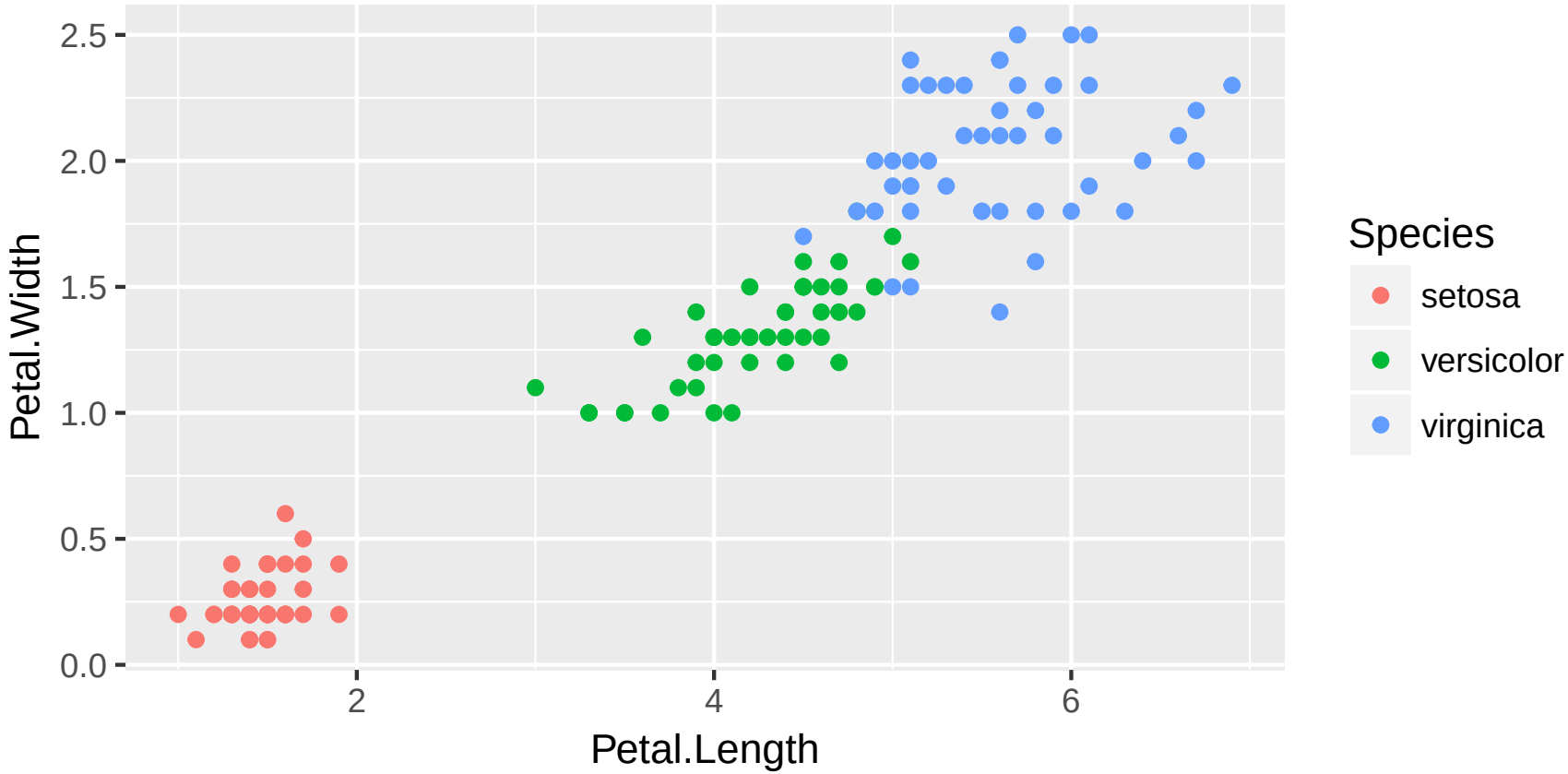
At the first step, each data point is joined with the nearest seed (based on Euclidean distance). Once two or more data points are joined together, the location of that cluster is the centroid of all the elements in the cluster.

Once all data points are in a cluster, the algorithm checks to see if any data point is closer to another cluster's centroid. If so, it reassigns any such data points, and then re-checks.

The k-means algorithm can be very sensitive to the initial choice of seeds. A typical use of k-means will run the algorithm many times, each time with a randomly selected set of seeds.

k-means example

```
library(ggplot2)
library(datasets)
ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) +
  geom_point()
```



k-means example

```
icl <- kmeans(iris[, 3:4], 3, nstart = 200)
icl
```

```
## K-means clustering with 3 clusters of sizes 48, 52, 50
##
## Cluster means:
##   Petal.Length Petal.Width
## 1     5.595833    2.037500
## 2     4.269231    1.342308
## 3     1.462000    0.246000
##
## Clustering vector:
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [106] 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 16.29167 13.05769  2.02200
```

Checking k-means vs known species

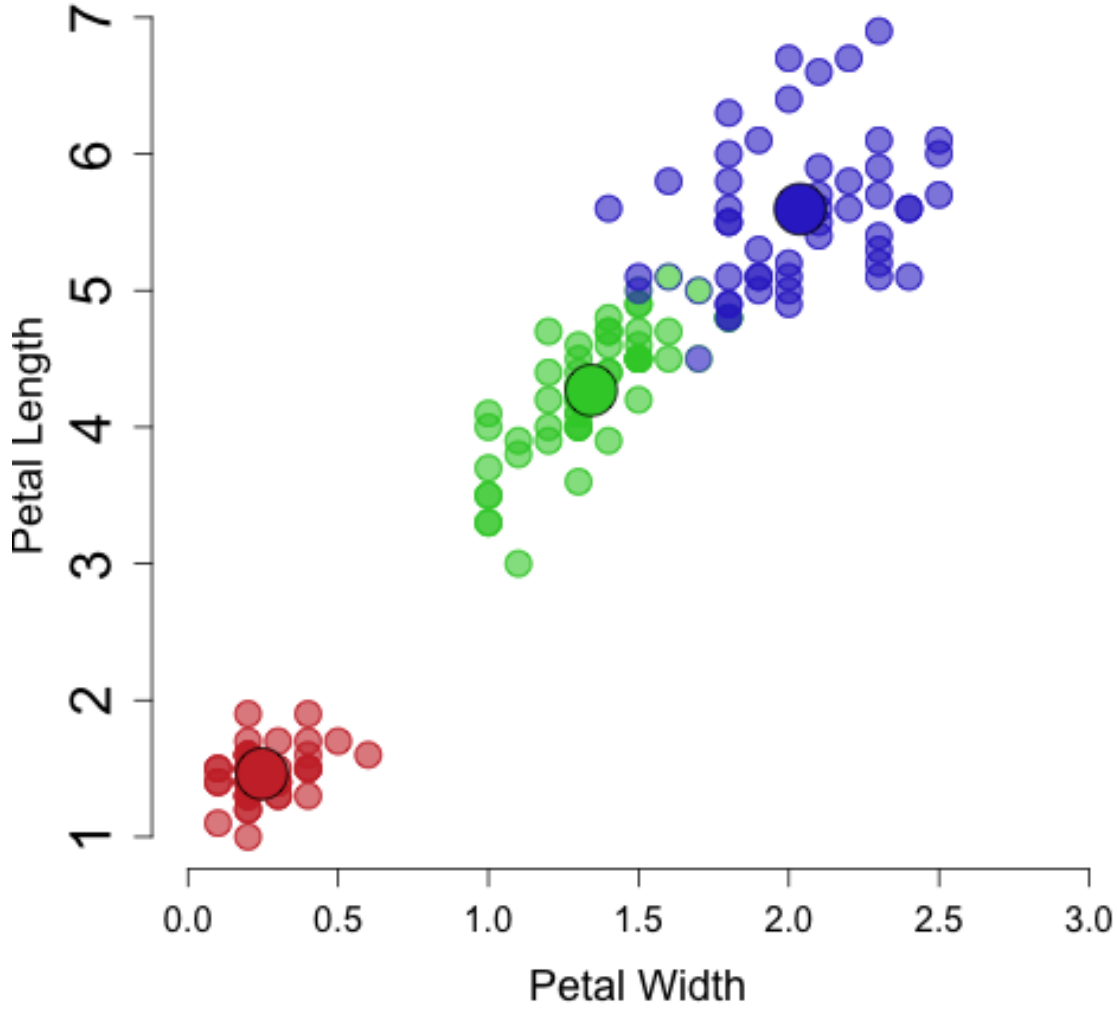
We can check how well the k-means algorithm captured the known species n this case (since we have known species).

First, we add the clusters to the data frame, then we use `xtabs()` to cross-tabulate the species and clusters:

```
iris["Cluster"] = icl$cluster
xtabs(~ Species + Cluster, iris)
```

```
##              Cluster
## Species        1  2  3
##   setosa       0  0 50
##   versicolor   2 48  0
##   virginica   46  4  0
```

Plotting k-means results with the data



Multidimensional Scaling

Hierarhical clustering and k-means clustering look for groups, and they find groups whether or not there are distinct groups to be found.

Multidimensional scaling (MDS) provides another method for exploring multidimensional data sets. MDS is useful for dimensionality reduction, but not for finding clusters.

There are two forms of MDS: metric and non-metric. Both are available in R, and both take a distance matrix as input (as well as a number of optional arguments) and return coordinates in a reduced-dimensionality space.

The difference between the two concerns the measure of goodness of fit (and how the relevant functions behave).

MDS in R

We can use a built-in data set and a related set from the textbook to illustrate MDS in R. In the `datasets` library, there is a distance matrix called `UScitiesD`. This contains 'straight line' distances between 10 US cities. Table 16.1 in the textbook has airline distances between the same 10 cities.

```
M = as.matrix(read.table("T16_1_USAIRDIST.dat"))

cities =c("Atlanta","Chicago","Denver","Houston","LA",
          "Miami","NYC","SF","Seattle","Washington, D.C")
colnames(M) = cities
rownames(M) = cities

D.air = as.dist(M)
D.lin = UScitiesD
```

Non-metric MDS in R

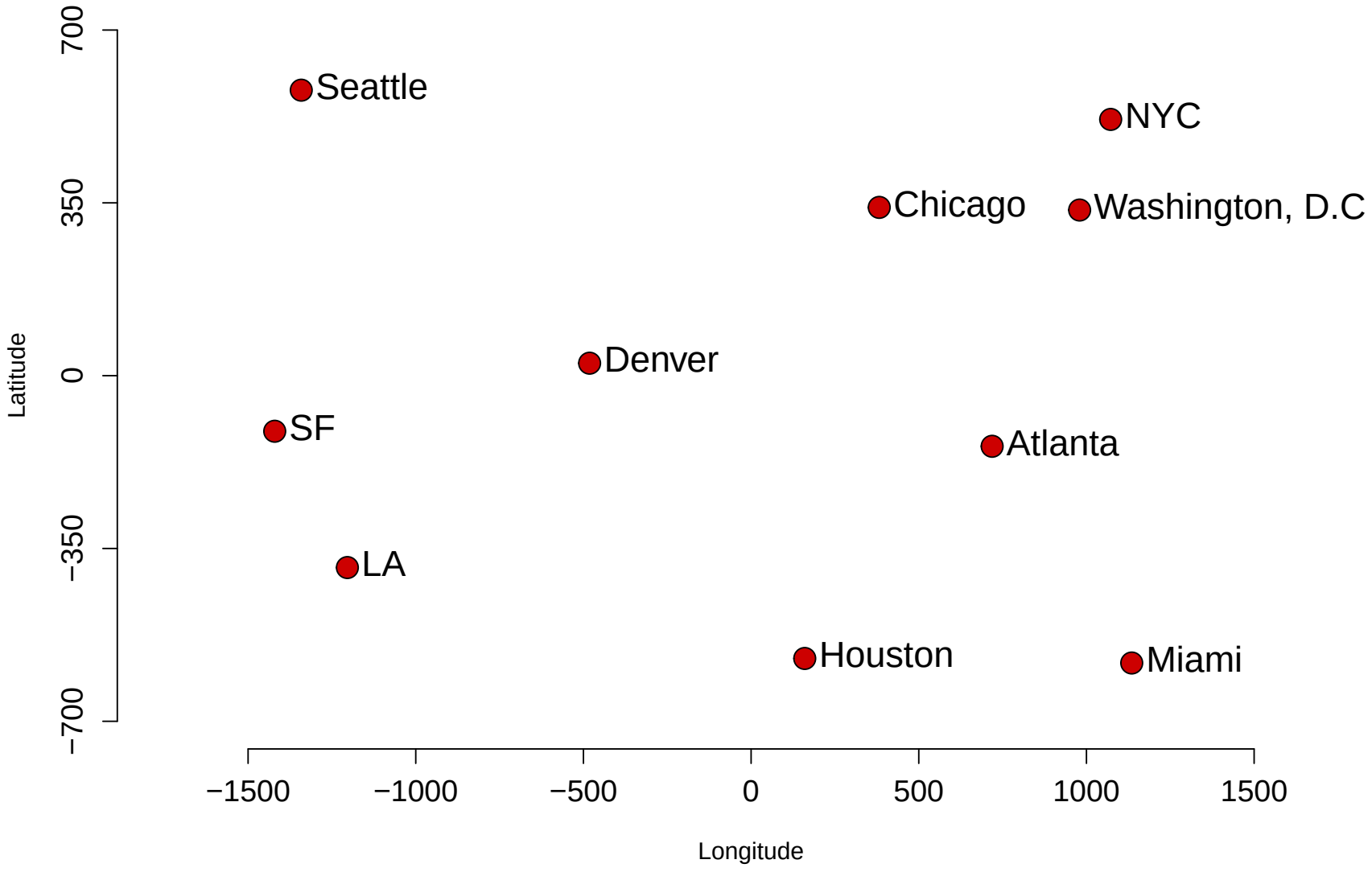
The function `isoMDS()` is available in the `MASS` library, which should already be installed with your R installation. We give it a distance matrix and tell it how many dimensions `k` we want in our MDS solution. It returns a list containing the coordinates of our data points in a new, `k`-dimensional space and the *stress* statistic, which is a measure of how closely the original distances and the MDS-based distances correspond to each other.

```
library(MASS)

nm.air = isoMDS(D.air, k = 2, maxit=500, trace=F)
NM.air = nm.air$points

nm.lin = isoMDS(D.lin, k = 2, maxit=500, trace=F)
NM.lin = nm.lin$points
```

Visualizing the MDS solutions



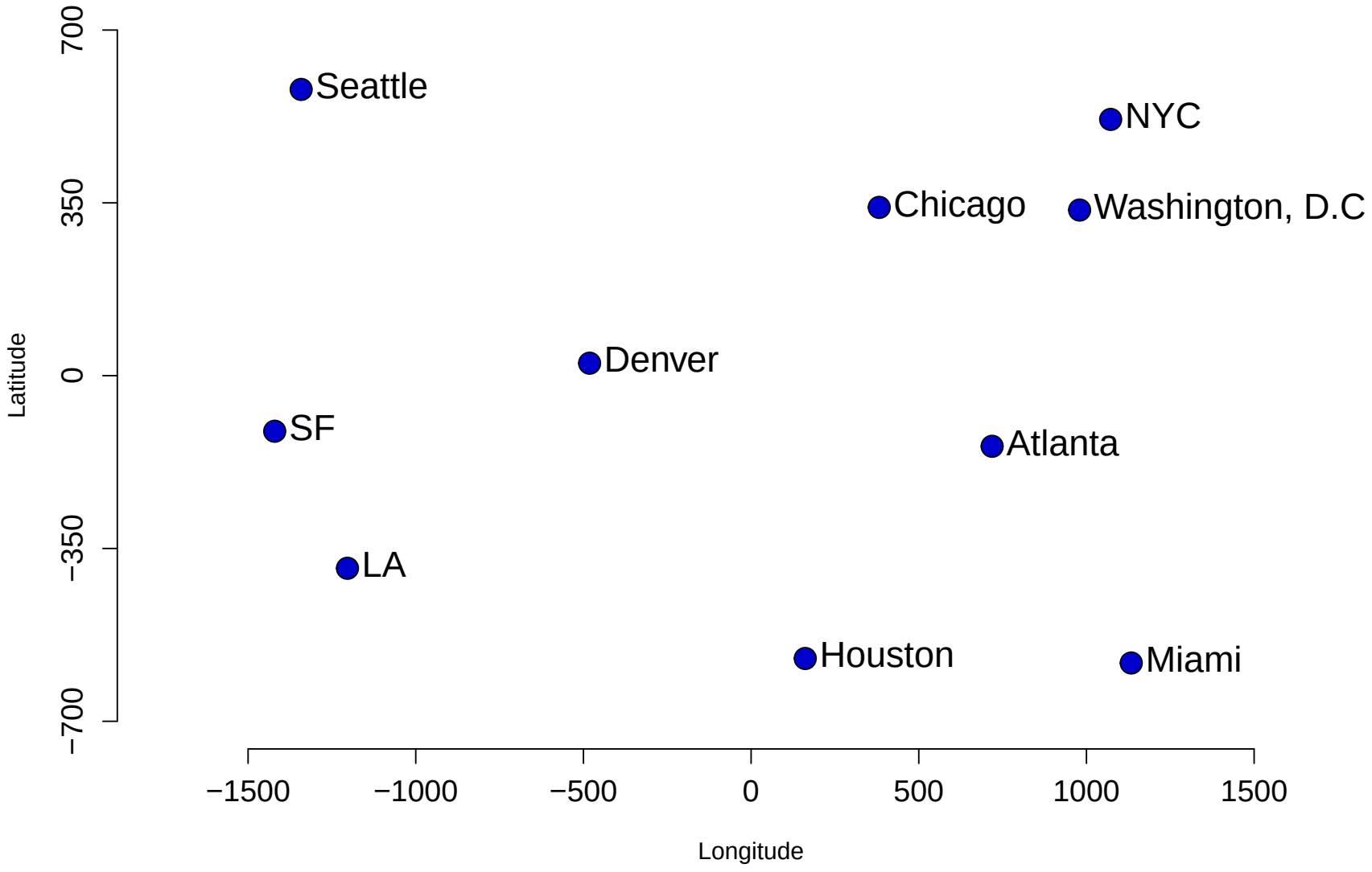
Metric MDS in R

The function `cmdscale()` is available in base R. We give it a distance matrix and tell it how many dimensions `k` we want in our MDS solution. By default, it just returns the coordinates of our data points in a new, `k`-dimensional space. We can get a measure of the goodness of fit if we want.

```
MT.air = cmdscale(D.air, k = 2)

MT.lin = cmdscale(D.lin, k = 2)
```

Visualizing the MDS solutions



MDS solution for crime data

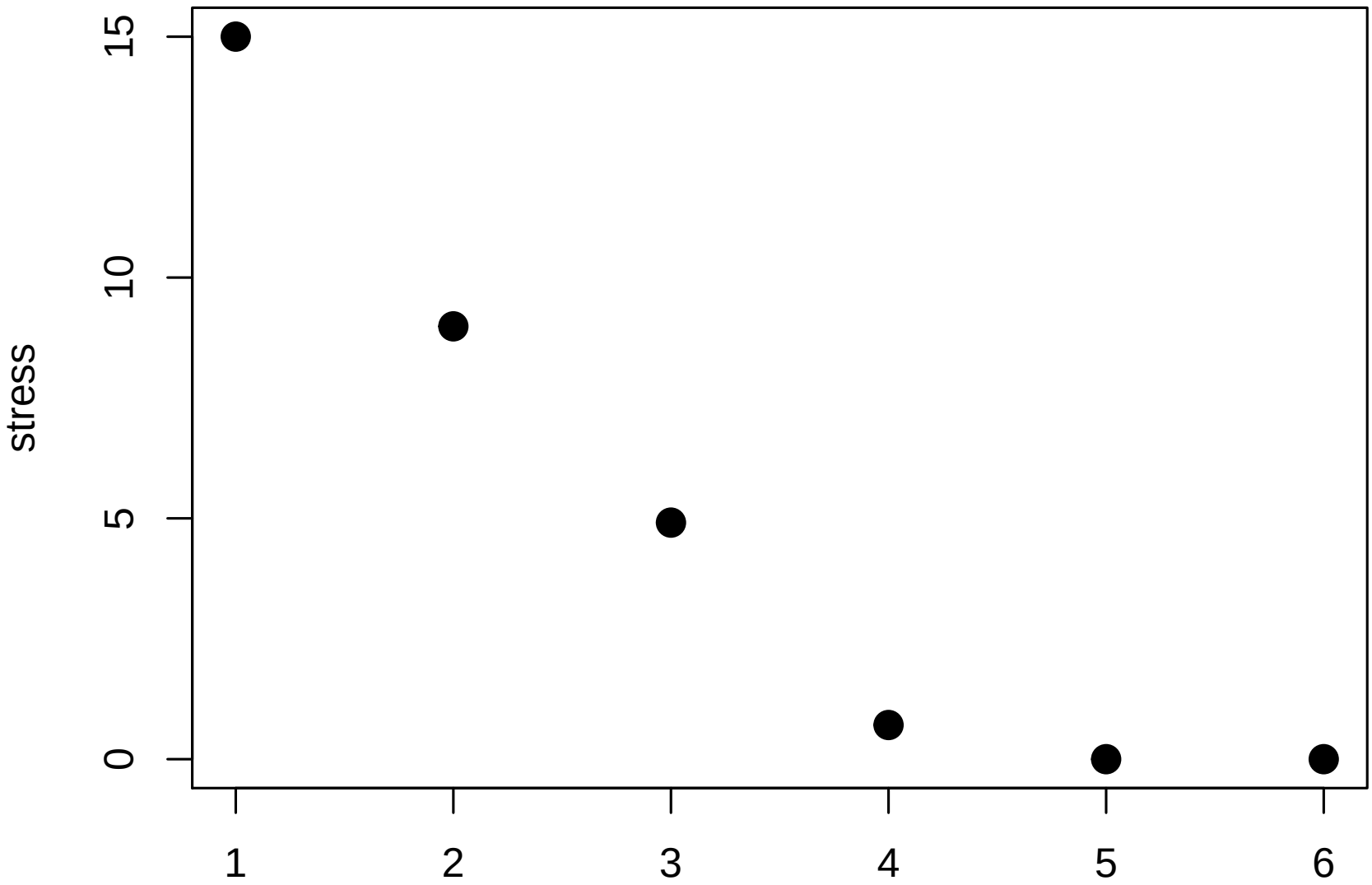
With the airline and straight line distance between cities, we know ahead of time that two dimensions is likely to give us a good solution. It is more typical to not know how many dimensions are appropriate.

One way to decide how many dimensions give the best fit is to look at a plot of stress (non-metric goodness of fit) as a function of the number of dimensions. If there is a good "elbow" (a point at which stress stops decreasing substantially), this can be used as a rule of thumb for the number of dimensions.

Unfortunately, it's just a rule of thumb, and there often isn't a clear "elbow," in which case you can decide how many dimensions to use based on interpretability of the MDS solution.

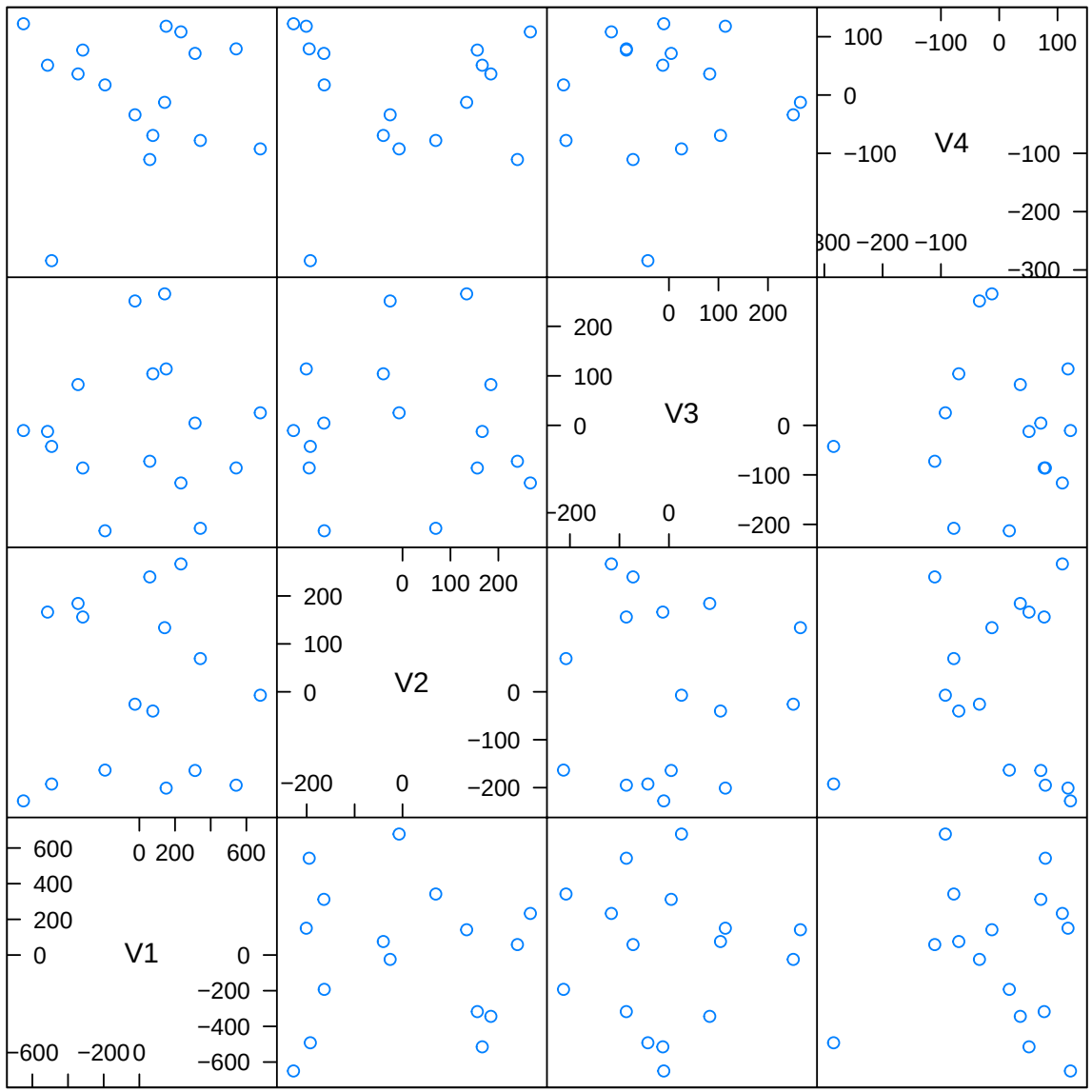
Stress and dimensionality

```
stress = rep(0,6)
for(k in 1:6){ stress[k] = isoMDS(D, k=k, trace=F)$stress }
plot(1:6,stress,xlab="k",main="",pch=19,cex=1.5)
```



Four dimensional MDS solution

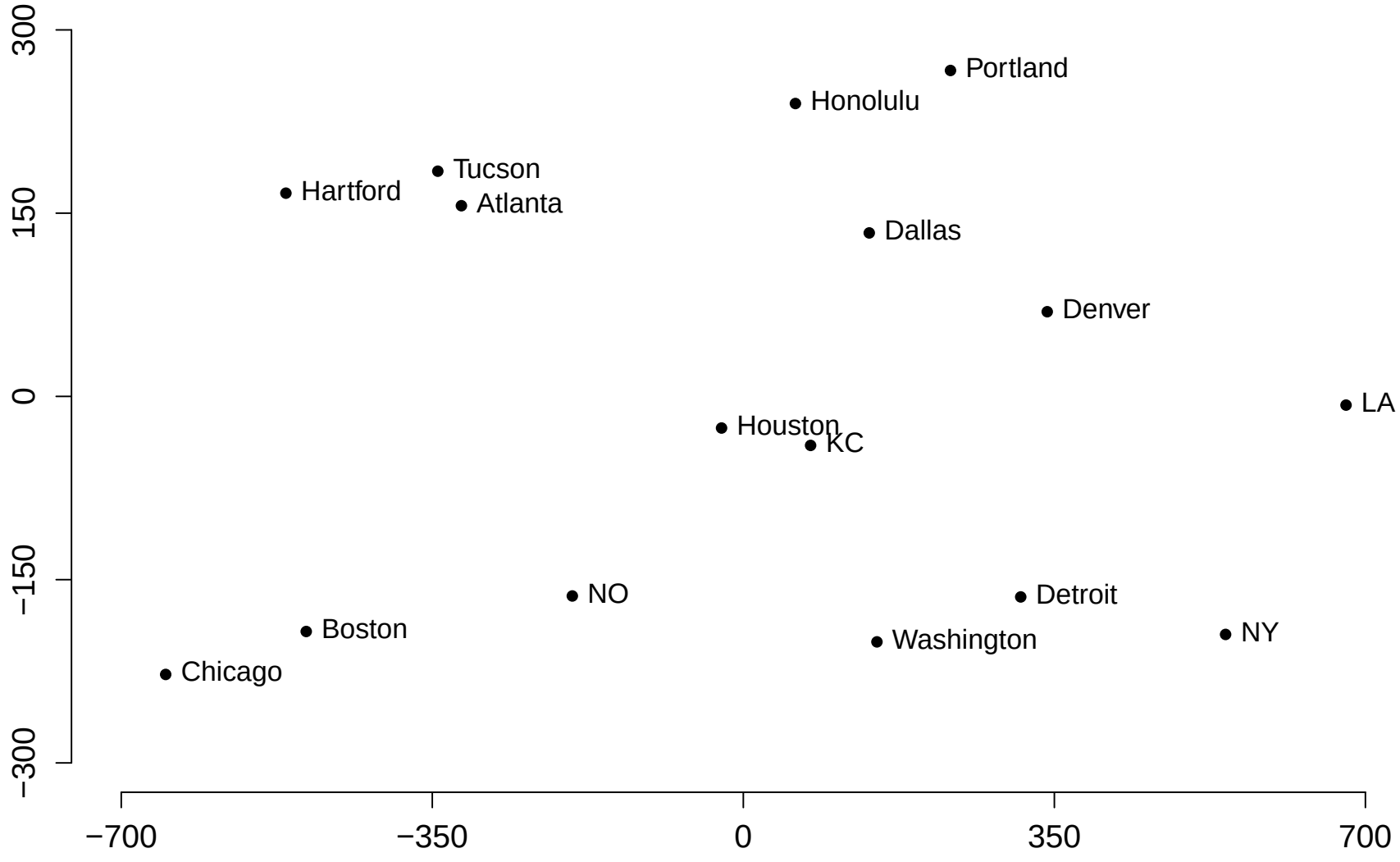
```
mds.crime.4 = isoMDS(D, k=4, trace=F)
```



Scatter Plot Matrix

Two dimensional MDS solution

```
mds.crime = cmdscale(D, k = 2)
```



Comparing MDS and Hierarchical Clustering

```
plot(wld)
```