MTH 3270 Notes 7

12 Unsupervised Learning (12)

- Recall that **unsupervised learning** is performed when data contain *explanatory variables* (X's), but there's no response variable (Y). The two most common goals are:
  - **Identifying groups** (**clusters**) of individuals based on the values of the *explanatory variables*. This is called ***cluster analysis***.
  - **Reducing the number of variables** (or "dimensionality") of a data set while retaining most of the information. This is called ***dimension reduction***.
- We'll look at two **cluster analysis** procedures,
  - Hierarchical clustering
  - K means clustering

  and one **dimension reduction** procedure,
  - Principal components analysis (PCA)

Data Set: `USArrests`

The `USArrests` data set (built into R) contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. Also given is the percent of the population living in urban areas. The four variables are:

`Murder`	Murder arrests (per 100,000).
`Assault`	Assault arrests (per 100,000).
`UrbanPop`	Percent urban population.
`Rape`	Rape arrests (per 100,000).

In addition, the data frame has a *row names* attribute containing the names of the states.

12.1 Hierarchical Clustering

- Hierarchical clustering can be used identify **groups** (**clusters**) of individuals when the ***p*** *explanatory* variables (Xs) are all **numerical**. Recall that in this case, an **observation** (**row** of the data frame) can be thought of as a point in a ***p*-dimensional** coordinate system.
- We'll see if we can identify **groups** (**clusters**) of states based on their crime and urban population rates using the (built-in) `USArrests` data.

```
head(USArrests)

##            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
```

```
rownames(USArrests)
```

(Output not shown.)

Fig. 1 below shows the result of **hierarchical clustering** of the 50 states using the data. We'll see how interpret this so-called ***dendrogram*** later.

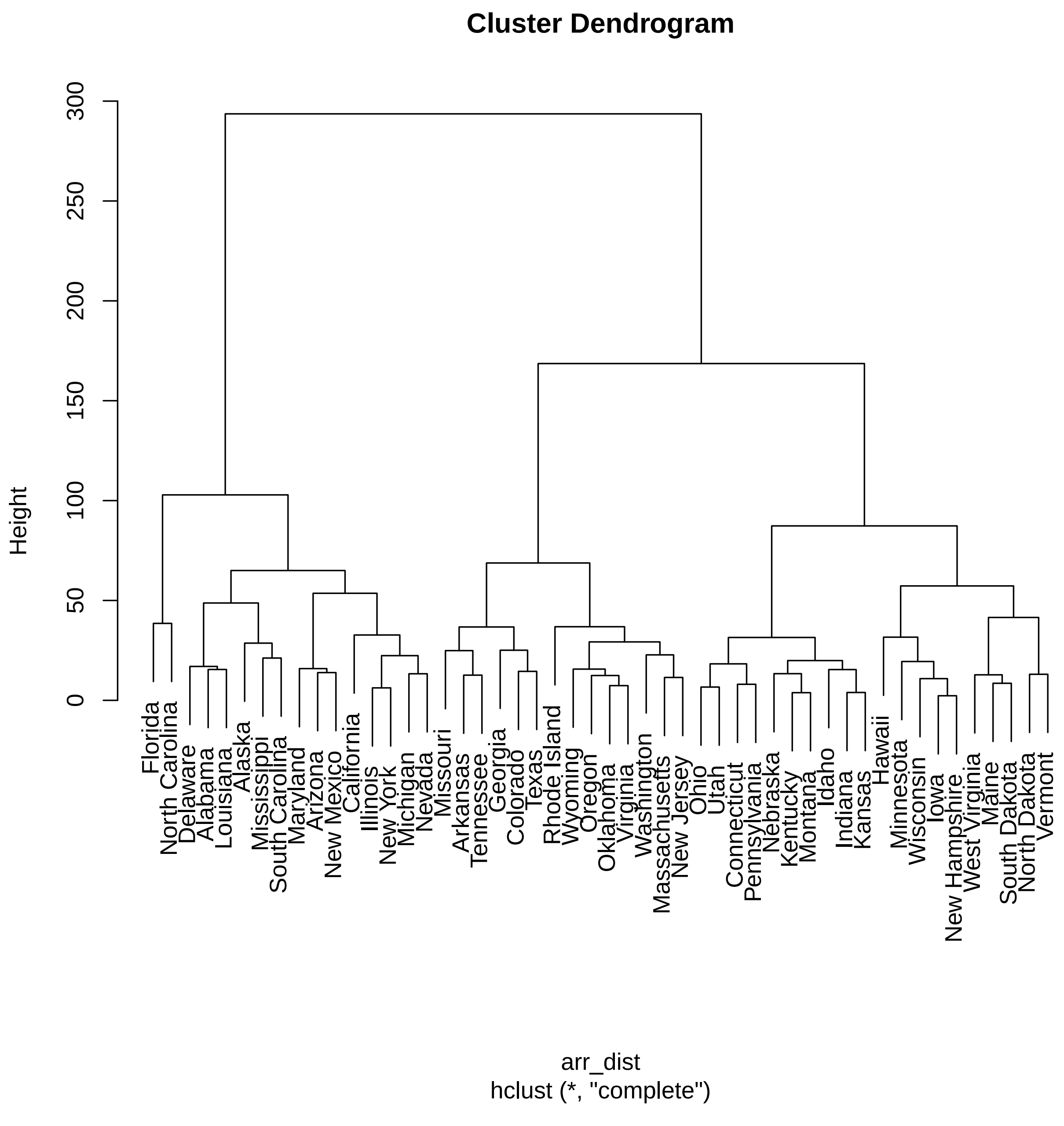

Cluster Dendrogram
Height
0
50
100
150
200
250
300
Florida
North Carolina
Delaware
Alabama
Louisiana
Alaska
Mississippi
South Carolina
Maryland
Arizona
New Mexico
California
Illinois
New York
Michigan
Nevada
Missouri
Arkansas
Tennessee
Georgia
Colorado
Texas
Rhode Island
Wyoming
Oregon
Oklahoma
Virginia
Washington
Massachusetts
New Jersey
Ohio
Utah
Connecticut
Pennsylvania
Nebraska
Kentucky
Montana
Idaho
Indiana
Kansas
Hawaii
Minnesota
Wisconsin
Iowa
New Hampshire
West Virginia
Maine
South Dakota
North Dakota
Vermont
arr_dist
hclust (*, "complete")

Figure 1

- To carry out ***hierarchical clustering***:

  1. Let n denote the number of **rows**, i.e. **observations**, in the data frame.
  2. Begin with each observation representing a **"singleton" cluster** (i.e. begin with $\boldsymbol{n}$ **clusters**, each consisting of a single observation).
  3. Merge the two clusters (observations) that are "closest" in $\boldsymbol{p}$**-dimensional** space (**least dissimilar**) into a single cluster, resulting in $n-1$ clusters (one of which now has two observations). A measure of **dissimilarity** between clusters is defined below.
  4. At each of the remaining steps, merge the two "closest" (**least dissimilar**) clusters into a single cluster, producing one less cluster at the next higher level of the tree.
  5. The last $((n-1)$st) step produces **one cluster** consisting of **all** $\boldsymbol{n}$ **observations** in the data frame.

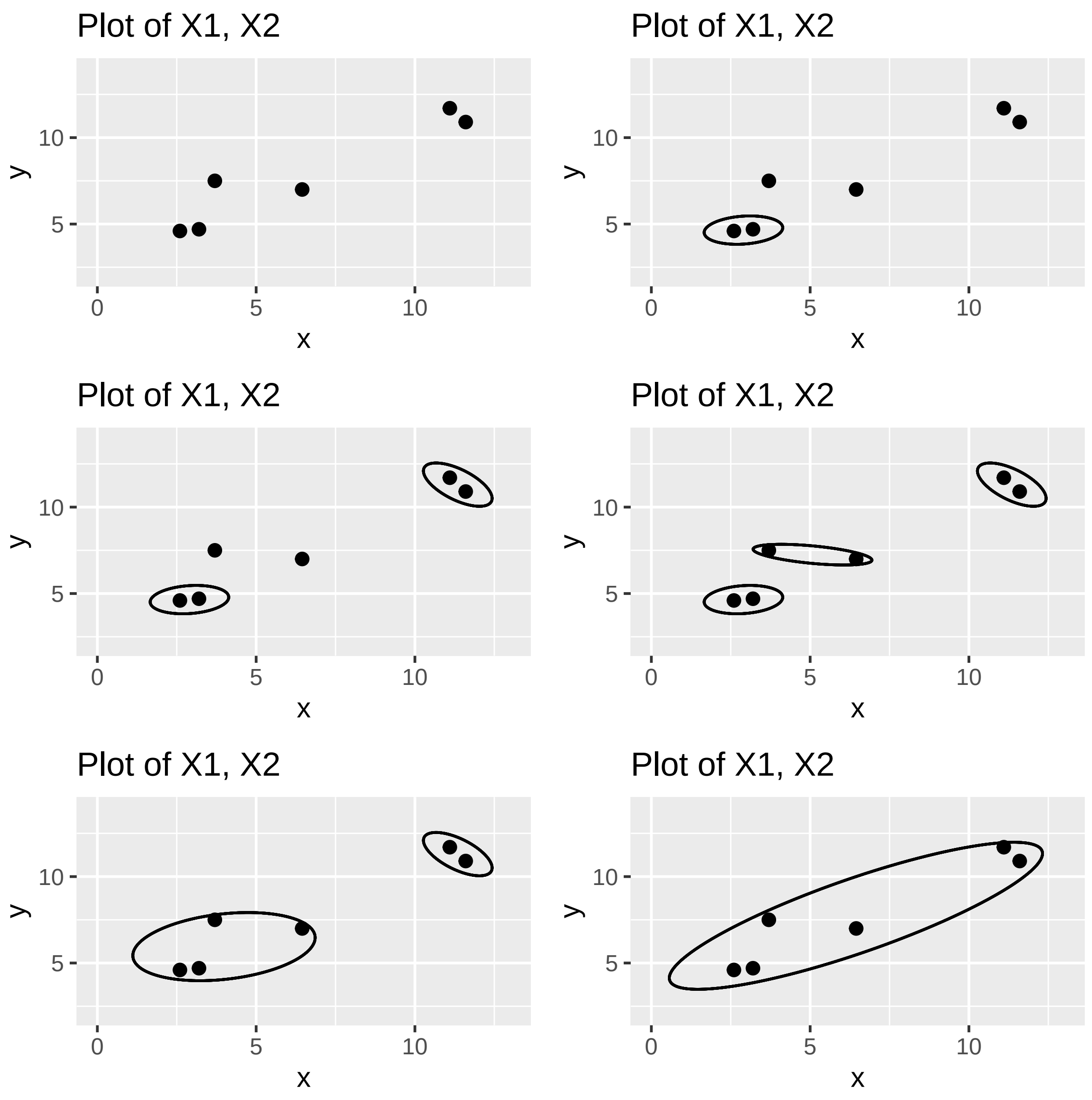


Figure 2: Six singleton clusters (upper left); Five clusters (upper right); Four clusters (middle left); Three clusters (middle right); Two clusters (bottom left).

- **Dissimilarity**: If $\boldsymbol{I}$ and $\boldsymbol{J}$ are two **clusters** (groups of observations in $\boldsymbol{p}$**-dimensional** space), and $\boldsymbol{d_{i,j}}$ is the **Euclidean distance** between observation i in cluster I and observation j in cluster J, three methods of measuring of **dissimilarity** between I and J are:

  1. ***Nearest-neighbor*** (or ***single linkage***): The **dissimilarity** between I and J is the distance between their two **closest** points, i.e.
  $$\text{Dissimilarity}(I,\ J) \ = \ \min(d_{i,j}).$$
  2. ***Furthest-neighbor*** (or ***complete linkage***): The **dissimilarity** between I and J is the distance between their two **farthest** points, i.e.
  $$\text{Dissimilarity}(I,\ J) \ = \ \max(d_{i,j}).$$
  3. ***Group average***: The **dissimilarity** between I and J is the **average** distance between their points, i.e.
  $$\text{Dissimilarity}(I,\ J) \ = \ \text{avg}(d_{i,j}).$$

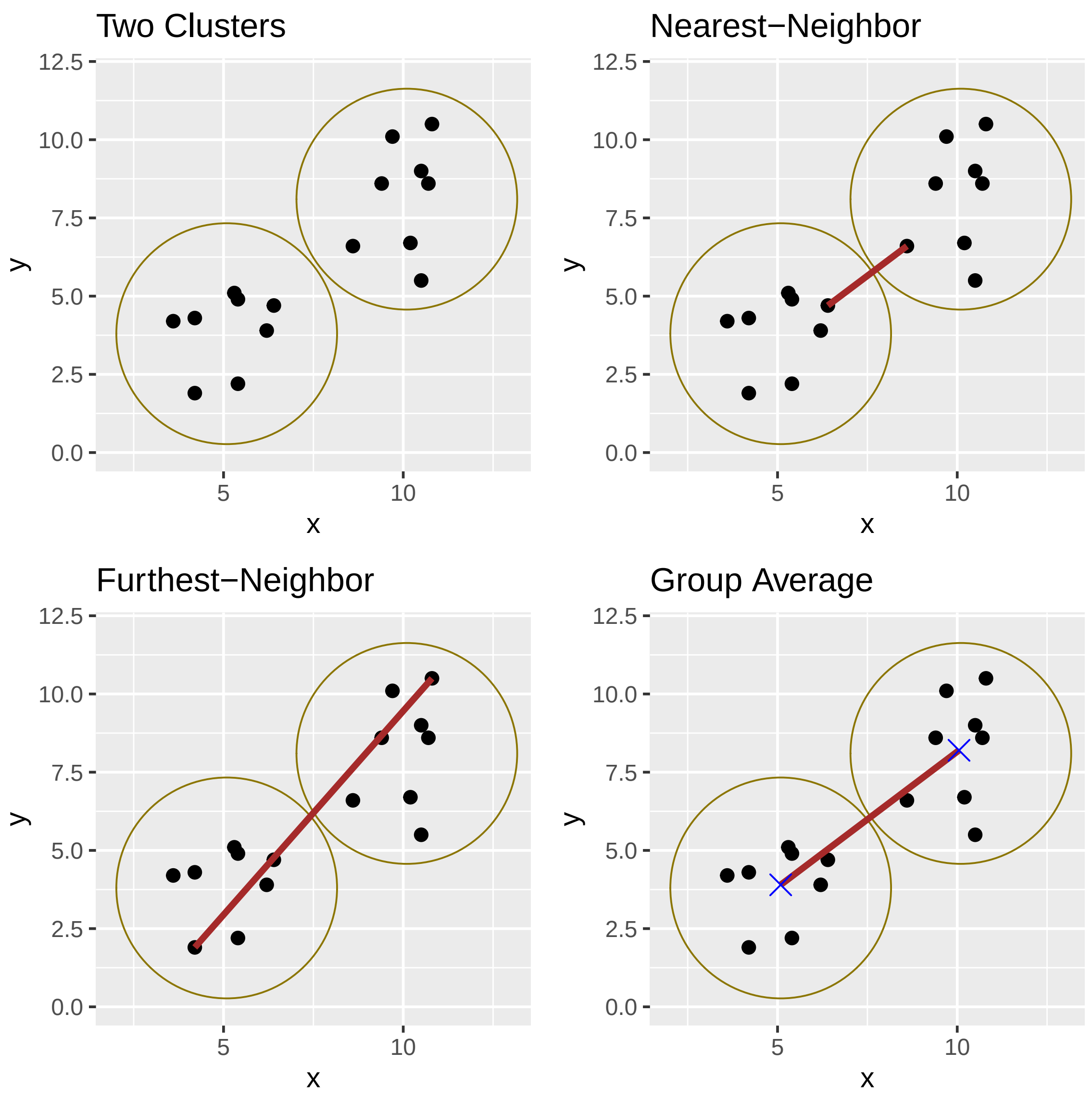


Figure 3: Two clusters (upper left); Nearest-neighbor dissimilarity measure (upper right); furthest-neighbor dissimilarity measure (bottom left); Group average dissimilarity measure (bottom right).

- **Comments**:
  - Each **distance** ($\boldsymbol{d_{i,j}}$ above) is a Euclidean distance in $\boldsymbol{p}$**-dimensional** space, where **each coordinate axis** represents an explanatory (X) variable (column of the data frame).
  - If the variables are measured on very **different scales**, consider **re-scaling** them, for example by ***standardizing*** each one, so that distances along each coordinate axis are comparable and reasonably reflect how different the two observations are.

- These functions (in base R) can be used to carry out **hierarchical clustering**.

```
hclust()       # Carry out hierarchical clustering on an object of class "dist".
               # Returns an object of class "hclust".
plot.hclust()  # Method for the (generic) plot() function that plots objects
               # of class "hclust".
rect.hclust()  # Draw rectangles around clusters in a dendrogram.
cutree()       # Obtain sets of observations corresponding to clusters.
```

- The following function will compute **distances** between observations in ***p*-dimensional** space, where p is the number of explanatory variables (X's) in the data set.

```
dist()   # Compute pairwise distances between observations (rows) in
         # a data frame.  Returns an object of class "dist".
```

- For example, to carry out a **hierarchical cluster analysis** of the (built-in) `USArrests` data and produce the so-called ***dendrogram*** of Fig. 1, type:

```r
arr_dist <- dist(USArrests, method = "euclidean")
arr_hclust <- hclust(arr_dist)
plot(arr_hclust, cex = 0.7)
```

- **Interpretation** of a **Dendrogram**:
  - The **terminal nodes** at the bottom of the **dendrogram** are **"singleton clusters"**, i.e. individual observations (rows of the data frame).
  - Each **parent node** of the **dendrogram** (i.e. each branch of the depicted tree), represented by a **vertical line**, is a **cluster** of observations (group of rows of the data frame) formed by **merging** the two clusters of its **child nodes**.
  - Thus the **merges** proceed **bottom-to-top** in the **dendrogram**.
  - The **height** (i.e. y-axis value) of a **horizontal line** is the **dissimilarity** between the two **clusters** (**nodes**) being **merged**. As the **merges** proceed, the clusters **merged** are more and more **dissimilar**.

    A **long vertical line** indicates the node's daughter nodes are substantially more **similar** to each other than the node itself is to its sibling.

- **Identify Clusters** in a **Dendrogram**:
  - Draw a *horizontal line* through the **dendrogram**.
  - The **nodes** (vertical lines) it crosses are the **clusters**.
  - The *vertical position* of the *horizontal line* you draw controls **how many clusters** the data set will be split into (for example 2 and 3 below).

```r
plot(arr_hclust, cex = 0.7)
abline(h = c(135, 232), lty = "dashed", col = "red")
```

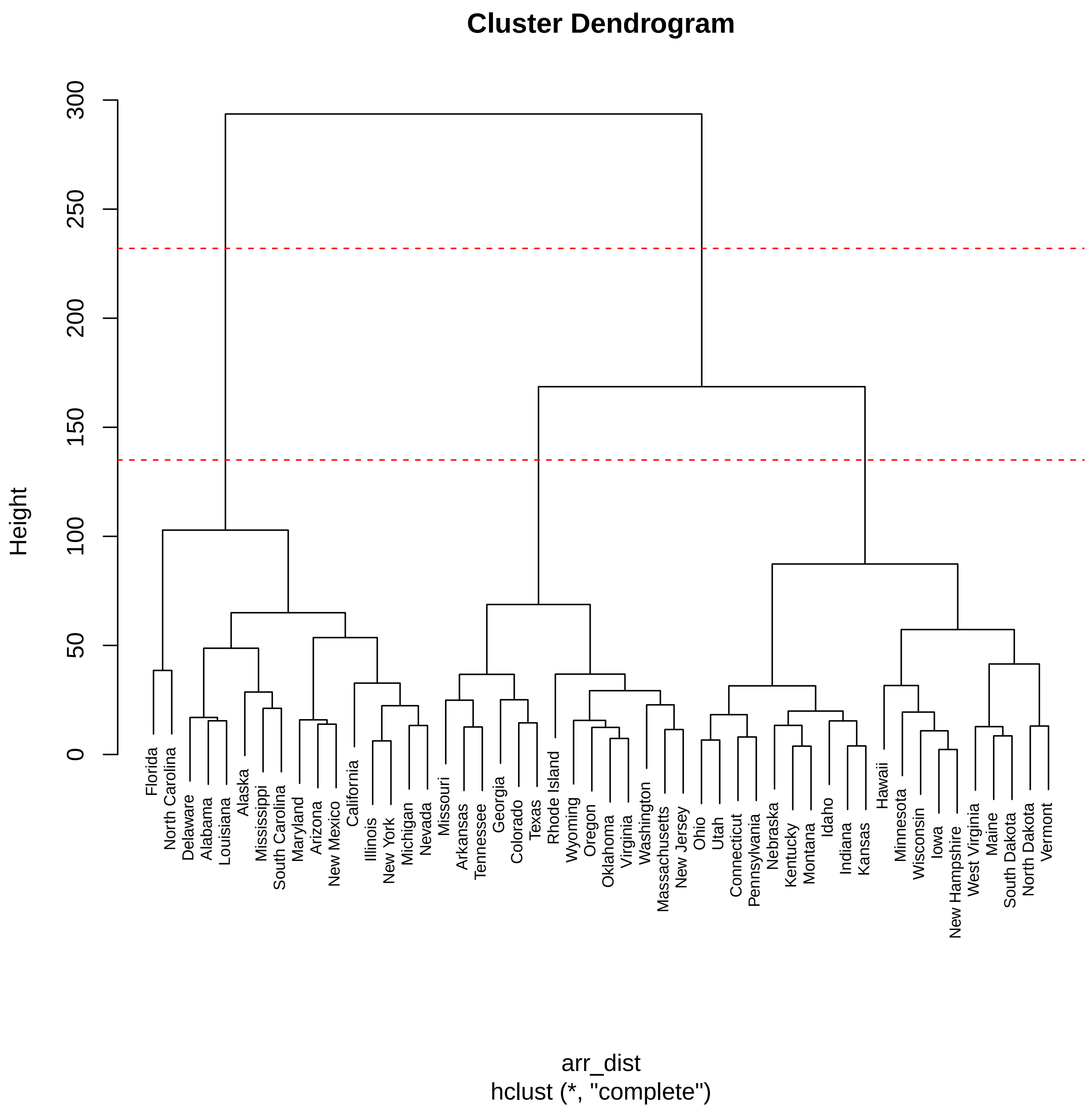

Cluster Dendrogram
Height
0
50
100
150
200
250
300
Florida
North Carolina
Delaware
Alabama
Louisiana
Alaska
Mississippi
South Carolina
Maryland
Arizona
New Mexico
California
Illinois
New York
Michigan
Nevada
Missouri
Arkansas
Tennessee
Georgia
Colorado
Texas
Rhode Island
Wyoming
Oregon
Oklahoma
Virginia
Washington
Massachusetts
New Jersey
Ohio
Utah
Connecticut
Pennsylvania
Nebraska
Kentucky
Montana
Idaho
Indiana
Kansas
Hawaii
Minnesota
Wisconsin
Iowa
New Hampshire
West Virginia
Maine
South Dakota
North Dakota
Vermont
arr_dist
hclust (*, "complete")

Figure 4

The `rect.hclust()` function will plot red rectangles around **clusters** in the **dendrogram**. The argument `k` is used to specify the desired number of **clusters**:

```
plot(arr_hclust, cex = 0.7)
rect.hclust(arr_hclust, k = 2, border = "red")
```

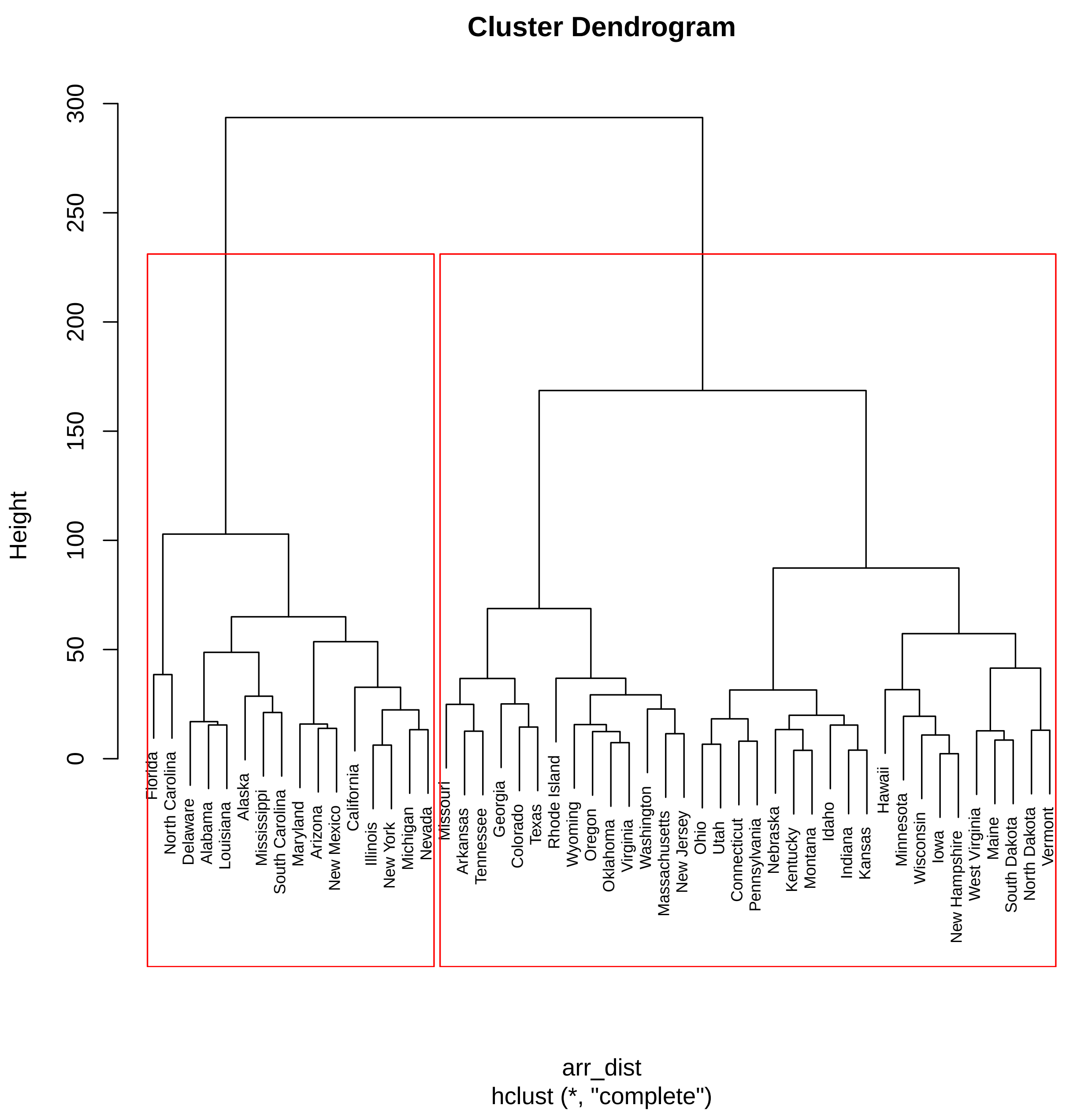


Figure 5

The function `cutree()` is used to obtain sets of observations (rows of the data frame) corresponding to **clusters**. It returns a *vector* indicating **cluster membership** for each observation. For example:

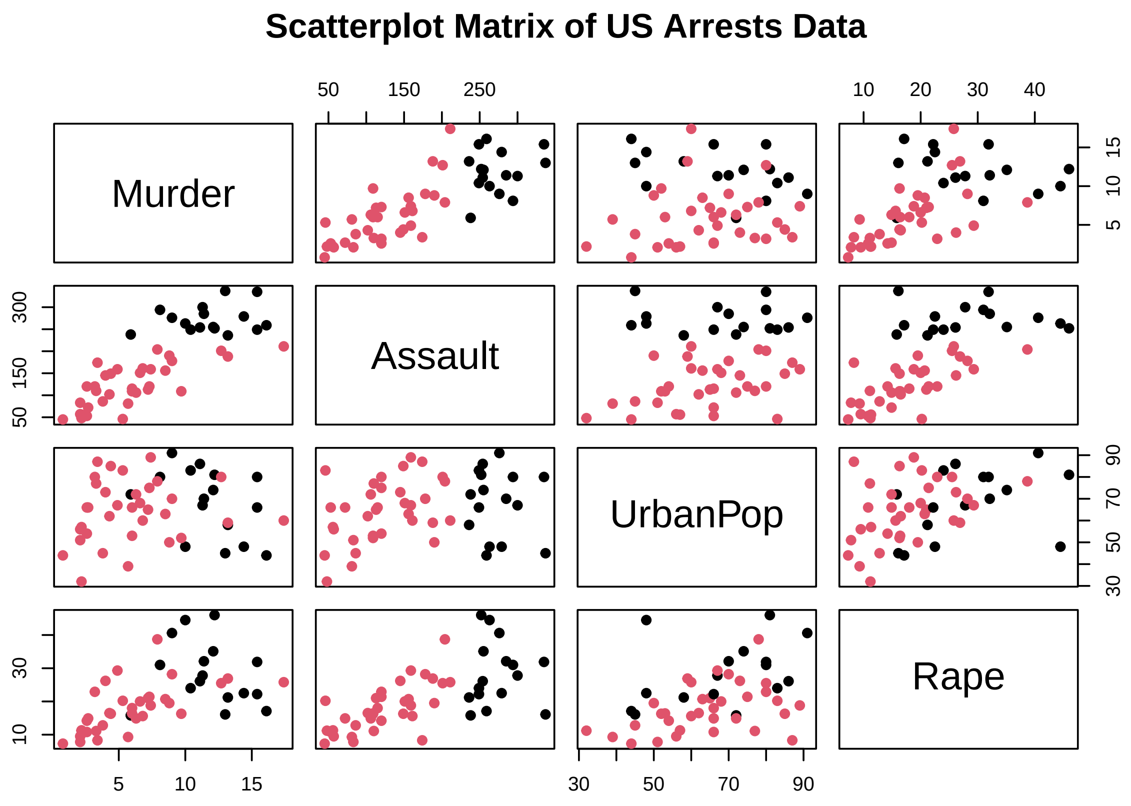
```
my.clusters <- cutree(arr_hclust, k = 2)
my.clusters

##        Alabama          Alaska         Arizona        Arkansas
##              1               1               1               2
##     California        Colorado     Connecticut        Delaware
##              1               2               2               1
##        Florida         Georgia          Hawaii           Idaho
##              1              2              2              2
##       Illinois        Indiana           Iowa         Kansas
##              1              2              2              2
##       Kentucky      Louisiana          Maine       Maryland
##              2              1              2              1
##  Massachusetts       Michigan      Minnesota    Mississippi
##              2              1              2              1
##       Missouri        Montana       Nebraska         Nevada
##              2              2              2              1
##  New Hampshire     New Jersey     New Mexico       New York
##              2              2              1              1
## North Carolina   North Dakota           Ohio       Oklahoma
##              1              2              2              2
##         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              2              2              1
##   South Dakota      Tennessee          Texas           Utah
##              2              2              2              2
##        Vermont       Virginia     Washington  West Virginia
##              2              2              2              2
##      Wisconsin        Wyoming
##              2              2
```

One use of `cutree()` is to **filter clusters** out of the full data frame, for example (using the `my.clusters` *cluster membership* vector from above):

```
library(dplyr)        # For filter()

clust1.data <- filter(USArrests, my.clusters == 1)     # Cluster 1
clust2.data <- filter(USArrests, my.clusters == 2)     # Cluster 2
```

Another use of `cutree()` is to be able to identify **clusters** in a **scatterplot matrix**, for example (using the `my.clusters` *cluster membership* vector from above):

```
pairs(USArrests,
      col = my.clusters,
      main = "Scatterplot Matrix of US Arrests Data",
      pch = 19)
```



Data Set: `wine`

The `wine` data set (from the `"rattle"` package) contains the results of a chemical analysis of wines grown in a specific area of Italy. Three types of wine are represented in the 178 specimens, with the results of 13 chemical analyses recorded for each specimen. The `Type` variable has been transformed into a categorical variable.

The data contains no missing values and consists of only numeric data, with a three-class target variable (`Type`) for classification. The 14 variables are:

`Type`	The type of wine, one of three classes: 1 (59 obs), 2 (71 obs), and 3 (48 obs).
`Alcohol`	Alcohol
`Malic`	Malic acid
`Ash`	Ash
`Alcalinity`	Alcalinity of ash
`Magnesium`	Magnesium
`Phenols`	Total phenols
`Flavanoids`	Flavanoids.
`Nonflavanoids`	Nonflavanoid phenols
`Proanthocyanins`	Proanthocyanins
`Color`	Color intensity
`Hue`	Hue
`Dilution`	D280/OD315 of diluted wines
`Proline`	Proline

Section 12.1 Exercises

Exercise 1 Here's a small data set.

```
my.data <- data.frame(X1 = c(3, 5, 4, 7),
                      X2 = c(6, 4, 9, 9),
                      X3 = c(1, 7, 2, 1))
rownames(my.data) <- c("Obs1", "Obs2", "Obs3", "Obs4")
my.data

##      X1 X2 X3
## Obs1  3  6  1
## Obs2  5  4  7
## Obs3  4  9  2
## Obs4  7  9  1
```

Compute the pairwise **Euclidean distances** (in a **3-dimensional** space whose coordinates are `X1`, `X2`, and `X3`) between observations (rows) from `my.data`:

```
my.data_dist <- dist(my.data, method = "euclidean")
my.data_dist
```

a) What's the **distance** between `Obs1` and `Obs2`?

b) Which two observations are "closest" (**least dissimilar**) to each other?

c) Which two observations would be **merged** in the **first step** of a **hierarchical clustering** procedure?

Exercise 2 Compute the pairwise **Euclidean distances** (in a **4-dimensional** space whose coordinates are `Murder`, `Assault`, `UrbanPop`, and `Rape`) between states in the `USArrests` data set:

```
arr_dist <- dist(USArrests, method = "euclidean")
arr_dist
```

What's the **distance** between **Florida** and **Alabama**?

Exercise 3 The `"rattle"` package contains a data set named `wine` (described above):

```
library(rattle)

head(wine)
```

The first column (`Type`) is a categorical variable, so it shouldn't be included in the cluster analysis:

```
library(dplyr)     # For select()

# The Type column gets removed:
wine2 <- select(wine, -Type)
```

Use `dist` to compute the **distances** (in **13-dimensional** space) between the wine specimens:

```
wine_dist <- dist(wine2, method = "euclidean")
wine_dist
```

a) Now use `wine_dist` to carry out a **hierarchical cluster analysis** on the `wines2` data set (which excludes `Type`), and produce the **dendrogram**. **Report your R command(s)**.

b) Next, use `rect.hclust()` to plot red rectangles around $\boldsymbol{k} = \mathbf{2}$ **clusters** in the **dendrogram**. **Report your R command(s)**.

c) Finally, use `cutree()` to obtain $\boldsymbol{k} = \mathbf{2}$ sets of observations (rows of the `wines2` data frame) corresponding to the **two clusters** of part *b*.

   How many observations are in each of the **two clusters**?

12.2 *K* Means Clustering

- ***k means clustering*** is another method of **identifying groups** (**clusters**) of observations (rows of the data frame) based *only* on the values of *explanatory variables* (X's).

  Unlike *hierarchical clustering*, ***k* means clustering** requires **knowing in advance** the number of clusters ***k*** into which the observations will be partitioned.

- To carry out ***k means clustering***:

  1. Guess the **"centers"** (in ***p*-dimensional** space, where ***p*** is the number of explanatory variables, or X's) of the ***k* clusters**, either *subjectively* or *randomly*. The **"cluster centers"** are the ***cluster means***, a definition of which will be given later.
  2. Given a current set of **"cluster centers"**, **assign** each observation to the **closest cluster center** (in ***p*-dimensional** space). Each observation will now be in **one** of the ***k* clusters**.
  3. For a given set of **assignments** of observations to **clusters**, compute the **"centers"** of these **clusters**. Note that these newly computed **"centers"** may have **shifted** a bit from their previous positions (in ***p*-dimensional** space).
  4. Repeat Steps 2 and 3 until the **assignments** of observations to clusters **don't change**, in which case the **"cluster centers" won't change** either.

- For example, consider the task of identifying $\boldsymbol{k = 3}$ clusters (groups) in the data shown and plotted below.

```r
my.x1 <- c(5.2, 4.6, 5.9, 6.8, 10.5, 10.7, 8.6, 10.5, 14.1, 16.4, 14.3, 12.4)
my.x2 <- c(3.6, 4.7, 2.2, 4.5, 7.2, 7.3, 7.1, 9.9, 6.3, 4.2, 6.2, 3.3)
my.data <- data.frame(x1 = my.x1, x2 = my.x2)
```

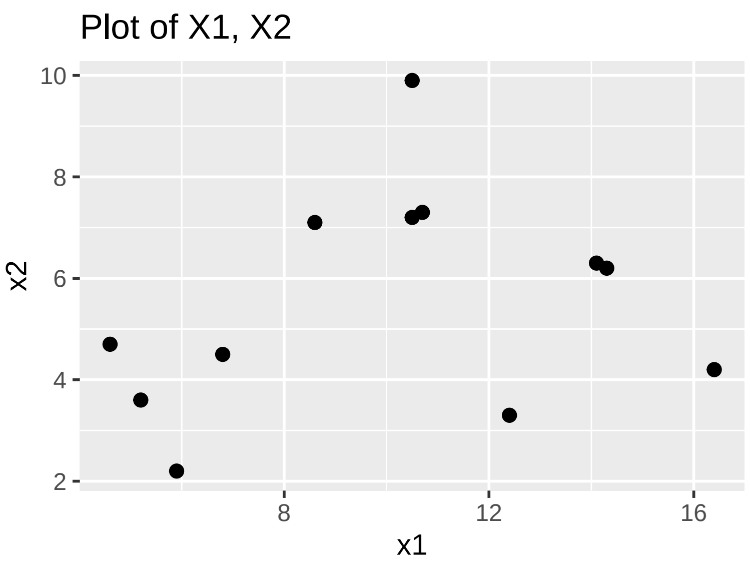


Figure 6

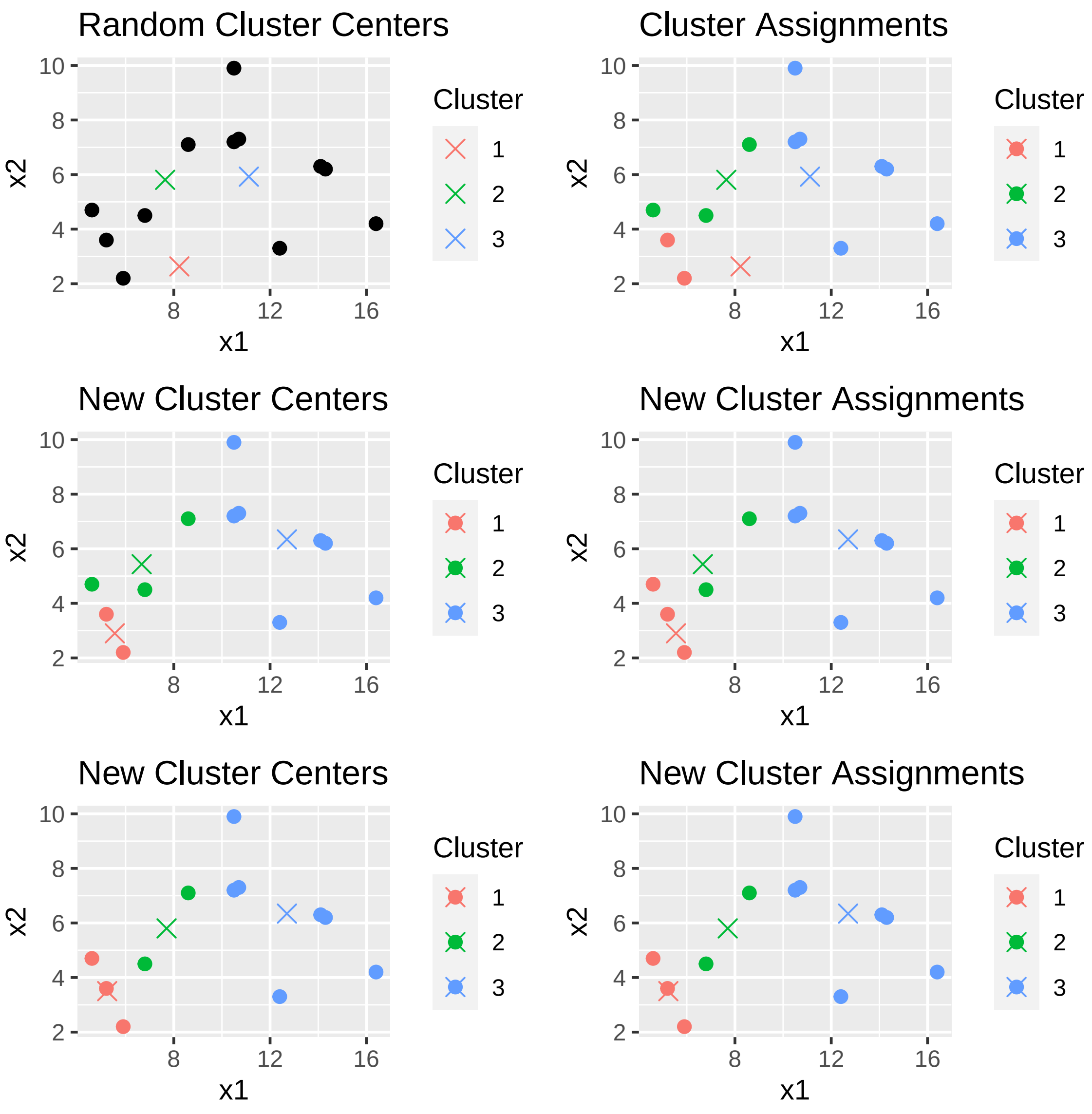


Figure 7: The X's are the cluster centers. Step 1 (upper left); Step 2 (upper right); Step 3 (middle left); Step 2 again (middle right); Step 3 again (bottom left); Step 4 (bottom right).

- The **coordinates** (in a ***p*-dimensional** coordinate system) of the ***cluster centers*** are obtained by **averaging** each of the p explanatory variables using only the observations in the cluster.

  For example, suppose these data are four observations in **one** of ***k* clusters**:

```
my.data <- data.frame(X1 = c(3, 5, 4, 7),
                      X2 = c(6, 4, 9, 9),
                      X3 = c(1, 7, 2, 1))
rownames(my.data) <- c("Obs1", "Obs2", "Obs3", "Obs4")
my.data

##      X1 X2 X3
## Obs1  3  6  1
## Obs2  5  4  7
## Obs3  4  9  2
## Obs4  7  9  1
```

  Then the **cluster center** would be a point in a **3-dimensional** (`X1`, `X2`, `X3`) coordinate system having coordinates:

```
summarize(my.data,
          meanX1 = mean(X1),
          meanX2 = mean(X2),
          meanX3 = mean(X3))

##   meanX1 meanX2 meanX3
## 1   4.75      7   2.75
```

- The function below, from the `"mclust"` package, can be used to carry out ***k* means clustering**.

```
kmeans()       # Carry out k means clustering. Returns an object
               # of class "kmeans".
```

- For example, to carry out a ***k* means cluster analysis** to identify $k = 2$ clusters (groups of states) using the (built-in) `USArrests` data set, type:

```
library(mclust)

# Set seed for random selection of initial cluster centers
# so that the results can be duplicated later, if necessary:
set.seed(25)

# Carry out the k means cluster analysis with k = 2:
arr_kmclust <- kmeans(USArrests, centers = 2)
arr_kmclust

## K-means clustering with 2 clusters of sizes 21, 29
##
## Cluster means:
##      Murder  Assault UrbanPop     Rape
## 1 11.857143 255.0000 67.61905 28.11429
## 2  4.841379 109.7586 64.03448 16.24828
##
## Clustering vector:
##        Alabama         Alaska        Arizona       Arkansas
##              1              1              1              1
##     California       Colorado    Connecticut       Delaware
##              1              1              2              1
##        Florida        Georgia         Hawaii          Idaho
##              1              1              2              2
##       Illinois        Indiana           Iowa         Kansas
##              1              2              2              2
##       Kentucky      Louisiana          Maine       Maryland
##              2              1              2              1
##  Massachusetts       Michigan      Minnesota    Mississippi
##              2              1              2              1
##       Missouri        Montana       Nebraska         Nevada
##              2              2              2              1
##  New Hampshire     New Jersey     New Mexico       New York
##              2              2              1              1
## North Carolina   North Dakota           Ohio       Oklahoma
##              1              2              2              2
##         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              2              2              1
##   South Dakota      Tennessee          Texas           Utah
##              2              1              1              2
##        Vermont       Virginia     Washington  West Virginia
##              2              2              2              2
##      Wisconsin        Wyoming
##              2              2
##
## Within cluster sum of squares by cluster:
## [1] 41636.73 54762.30
##  (between_SS / total_SS =  72.9 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

Note that there are **two clusters** containing **21** and **29** states, respectively.

The object `arr_kmclust` belongs to the `"kmeans"` class of objects, but it's really just a *list* (type `is.list(arr_kmclust)`) whose component names can be viewed using `names()`:

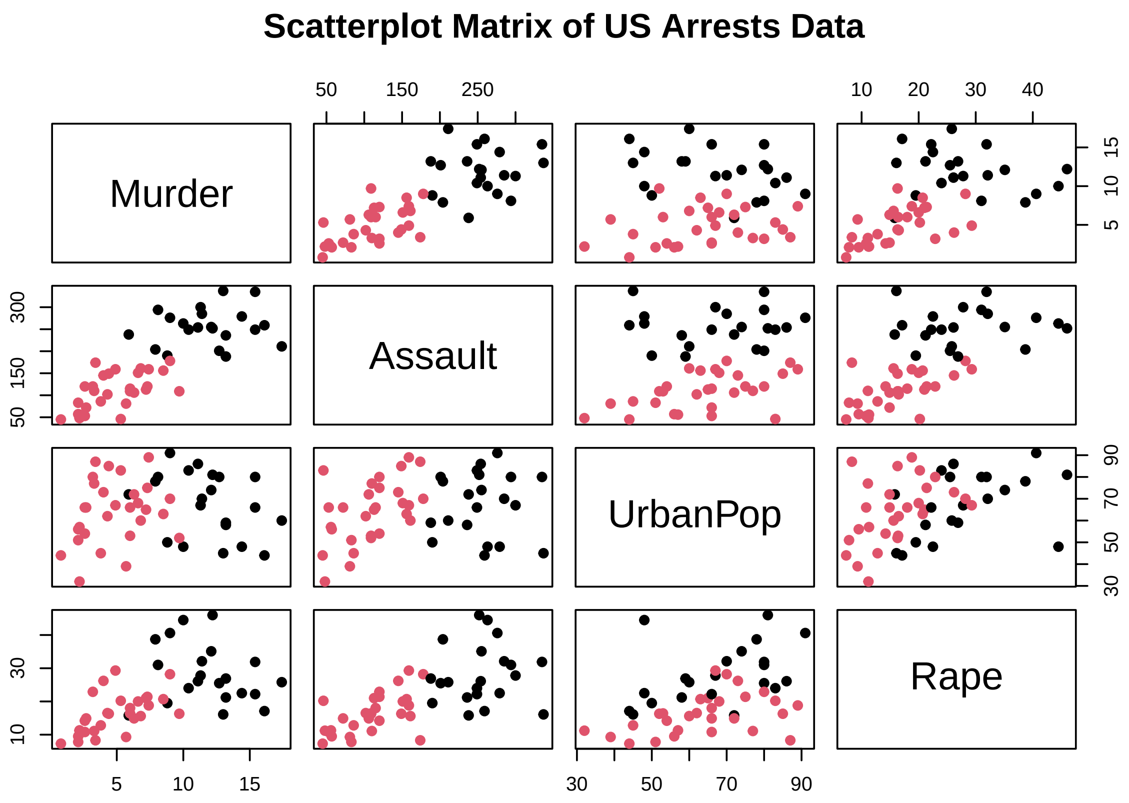
```
names(arr_kmclust)

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

We use the dollar sign operator `$` to access components of the `"kmeans"` object `arr_kmclust`:

```
my.clusters <- arr_kmclust$cluster
```

We can identify **clusters** in a **scatterplot matrix**, using the `my.clusters` *cluster membership* vector from above):

```
pairs(USArrests,
      col = my.clusters,
      main = "Scatterplot Matrix of US Arrests Data",
      pch = 19)
```



Section 12.2 Exercises

Exercise 4 Here are the data from above containing the variables `x1` and `x2` shown in Figs. 6 and 7:

```
my.x1 <- c(5.2, 4.6, 5.9, 6.8, 10.5, 10.7, 8.6, 10.5, 14.1, 16.4, 14.3, 12.4)
my.x2 <- c(3.6, 4.7, 2.2, 4.5, 7.2, 7.3, 7.1, 9.9, 6.3, 4.2, 6.2, 3.3)
my.data <- data.frame(x1 = my.x1, x2 = my.x2)
```

Carry out a ***k* means cluster analysis** on `my.data`, with $\boldsymbol{k = 3}$:

```
# So that everyone has the same randomly selected starting cluster centers:
set.seed(27)

# Carry out the k means cluster analysis with k = 3:
my_kmclust <- kmeans(my.data, centers = 3)
my_kmclust
```

How many observations are in each of the three **clusters** (groups) identified by the k means procedure?

Exercise 5 Recall that the `"rattle"` package contains a data set named `wine` (see description above):

```
# install.packages("rattle")
library(rattle)

head(wine)
```

The first column (`Type`) is a categorical variable, so it shouldn't be included in the cluster analysis:

```r
# For select():
library(dplyr)

# The Type column gets removed:
wine2 <- select(wine, -Type)
```

a) Carry out a ***k* means cluster analysis** on the `wine2` data set, with $\boldsymbol{k = 3}$:

```r
# So that everyone has the same randomly selected starting cluster centers:
set.seed(20)

# Carry out the k means cluster analysis with k = 3:
wine_kmclust <- kmeans(wine2, centers = 3)
wine_kmclust
```

Compare the **scatterplot matrix** showing the identified **clusters** with one showing the wine **types**:

```r
my.clusters <- wine_kmclust$cluster

pairs(wine2,
      col = my.clusters,
      main = "Scatterplot Matrix of Wine Data With Clusters",
      pch = 19)
```

```r
pairs(wine2,
      col = wine$Type,
      main = "Scatterplot Matrix of Wine Data With Types",
      pch = 19)
```

Do the **clusters** correspond to wine **types**?

b) For **cluster analysis**, if the variables are measured on very **different scales**, it's best to ***standardize*** each one so that distances along each coordinate axis in p-dimensional space are comparable.

Re-run the **cluster analysis** after ***standardizing*** each of the 13 variables in `wine2`:

```r
# Standardize each of the 13 variables:
wine2_std <- scale(wine2, center = TRUE, scale = TRUE)

# So that everyone has the same randomly selected starting cluster centers:
set.seed(20)

# Carry out the k means cluster analysis with k = 3:
wine_kmclust_std <- kmeans(wine2_std, centers = 3)
wine_kmclust_std
```

Now compare the **scatterplot matrix** showing the identified **clusters** with the one showing the wine **Types**:

```r
my.clusters_std <- wine_kmclust_std$cluster

pairs(wine2,
      col = my.clusters_std,
      main = "Scatterplot Matrix of Wine Data With Clusters",
      pch = 19)
```

```
pairs(wine2,
      col = wine$Type,
      main = "Scatterplot Matrix of Wine Data With Types",
      pch = 19)
```

Now do the **clusters** correspond (at least approximately) to wine **Types**?

12.3 Dimension Reduction: SVD and PCA

- ***Dimension reduction*** refers to reducing the number of variables (columns) of a data set to a smaller number of variables that carry the most information.

  This sometimes facilitates prediction or cluster identification.

- **Dimension reduction** *might* involve simply **discarding** existing variables.

  This will be the case when a variable is *uninformative*, either because it *doesn't distinguish* between individuals or because it's *redundant*.

  But often it's better to first **"derive" new variables** from the existing ones *before* discarding any.

- Here's an example in which existing variables can simply be **discarded**.

  In this (hypothetical) data set on eight people, `Age` *doesn't distinguish* between people (everyone is the same age), and `WtKg` (weights in kilograms) is *redundant* when `WtLb` (weights in pounds) is present.

```
my.data <- data.frame(Age = c(25, 25, 25, 25, 25, 25, 25, 25),
                      WtLb  = c(160, 155, 165, 170, 180, 155, 165, 175),
                      WtKg = c(72.6, 70.3, 74.8, 77.1, 81.6, 70.3, 74.8, 79.4))
my.data

##   Age WtLb WtKg
## 1  25  160 72.6
## 2  25  155 70.3
## 3  25  165 74.8
## 4  25  170 77.1
## 5  25  180 81.6
## 6  25  155 70.3
## 7  25  165 74.8
## 8  25  175 79.4
```

We can think of *eliminating* the `Age` and `WtKg` columns as **"deriving"** a new variable `V` via a ***linear combination*** of the columns in `my.data`, with ***weights*** (coefficients) **0**, **1**, and **0**:

$$\texttt{V} \;=\; 0 \times \texttt{Age} + 1 \times \texttt{WtLb} + 0 \times \texttt{WtKg},$$

and then **discarding** all variables *except* `V`, e.g. (using `mutate()` and `select()` from `"dplyr"`):

```
my.new.data <- my.data %>%
                 mutate(V = 0*Age + 1*WtLb + 0*WtKg) %>%
                 select(V)
my.new.data

##     V
## 1 160
## 2 155
## 3 165
## 4 170
## 5 180
## 6 155
## 7 165
## 8 175
```

- In practice, it's rare for a variable to be *entirely* uninformative (as `Age` and `WtKg` are), but usually some are *less* informative than others.

  We can still **"derive"** new variables via **linear combinations** in this case.

  The *less informative* variables get *smaller weights* in the linear combination (but not necessarily zeros) .

- ***Singular value decomposition*** (or ***SVD***), a procedure from matrix algebra, is used to **"derive"** new variables $\boldsymbol{V_1, V_2, \ldots, V_p}$, each of which is a **linear combination** of explanatory variables $\boldsymbol{X_1, X_2, \ldots, X_p}$, i.e.

  $$V_j \;=\; a_{j1}X_1 + \cdots + a_{jp}X_p \qquad \text{for } j = 1, 2, \ldots, p\,,$$

  such that $\boldsymbol{V_1}$ is the *most* informative, $\boldsymbol{V_2}$ the *second most* informative, ..., $\boldsymbol{V_p}$ the *least* informative, and there **aren't redundancies** among the $\boldsymbol{V_j}$'s (i.e. they carry completely independent information).[1]

  Dimension reduction is then done by **discarding** all but the first few (most informative) $\boldsymbol{V_j}$'s. The *retained* $\boldsymbol{V_j}$'s can then be used as explanatory variables in random forests, cluster analyses, etc.

  [1]"Most informative" means its values *vary* the most across individuals (subject to a certain constraint on how large the weights can be: in each V, they're constrained so that the sum of their squares equals one). "Second most informative" means its values *vary* second most, etc. "There aren't redundancies" means the V's are *uncorrelated* with each other.

- ***Principal components analysis*** (or ***PCA***) is **SVD** performed **after *centering*** each of the $\boldsymbol{X}$ variables (by subtracting the mean of the variable from each of its values). **PCA** gives **more meaningful** results than SVD.[2] An example of PCA is given below.

  [2]Without centering, V_1 would weight heavily the X's whose values vary away from *zero*. With centering, it weights heavily X's whose values vary away from *their mean*.

- We'll use the function:

```
svd()      # Perform a singular value decomposition (or a principal
           # components analysis) on a set of explanatory variables
           # (columns) in a data frame (or matrix).
scale()    # Used to center variables (columns) in a data frame (or
           # matrix) by subtracting their means.  Can also be used to
           # standardize variables.
```

- As an example of **PCA** (i.e. **SVD** on the **centered variables**), we'll use `my.data` (from above). The means of `Age`, `WtLb`, and `WtKg` are:

```
summarize(my.data,
          meanAge = mean(Age),
          meanWtLb = mean(WtLb),
          meanWtKg = mean(WtKg))

##   meanAge meanWtLb meanWtKg
## 1      25  165.625  75.1125
```

  We can **center** the variables (subtract their means) using `scale()`:

```
# Center the variables in my.data:
data_cntr <- scale(my.data, center = TRUE, scale = FALSE) %>%
                as.data.frame()
data_cntr

##   Age    WtLb    WtKg
## 1   0  -5.625 -2.5125
## 2   0 -10.625 -4.8125
## 3   0  -0.625 -0.3125
## 4   0   4.375  1.9875
## 5   0  14.375  6.4875
## 6   0 -10.625 -4.8125
## 7   0  -0.625 -0.3125
## 8   0   9.375  4.2875
```

Now we perform **PCA**:

```
# Perform PCA (SVD on the centered variables):
my.pca <- svd(data_cntr)
```

The object `my.pca` is a *list* containing three components, `d`, `u`, and `v`:

```
names(my.pca)

## [1] "d" "u" "v"
```

The **weights** in the optimal **linear combinations** of `Age`, `WtLb`, and `WtKg` that generate $\boldsymbol{V_1}$, $\boldsymbol{V_2}$, and $\boldsymbol{V_3}$ are the columns of `my.pca$v`:

```
my.pca$v

##           [,1]       [,2] [,3]
## [1,] 0.0000000  0.0000000    1
## [2,] 0.9109678  0.4124774    0
## [3,] 0.4124774 -0.9109678    0
```

Thus

$$\begin{aligned} V_1 &= 0.00 \times \texttt{Age} + 0.91 \times \texttt{WtLb} + 0.41 \times \texttt{WtKg} \qquad (1) \\ V_2 &= 0.00 \times \texttt{Age} + 0.41 \times \texttt{WtLb} - 0.91 \times \texttt{WtKg} \qquad (2) \\ V_3 &= 1 \times \texttt{Age} + 0 \times \texttt{WtLb} + 0 \times \texttt{WtKg} \qquad (3) \end{aligned}$$

(where `Age`, `WtLb`, and `WtKg` are their *centered* versions). Note that V_1 is analogous to a "weighted average" of `WtLb` and `WtKg` with *weights* 0.91 and 0.41 (whose squares sum to one).

The $\boldsymbol{d}$ values in the vector `my.pca$d`:

```
my.pca$d

## [1] 26.25108372  0.06598016  0.00000000
```

can be used as measures of the relative amounts of information in V_1, V_2, and V_3. In particular, they measure **variation** of values within each V_j.[3] Larger values indicate more information. Thus almost all of the information in `my.data` is contained in V_1, and none of it is contained in V_3.

We can obtain a *re-scaled version* of the **"derived"** variables $\boldsymbol{V_1}$, $\boldsymbol{V_2}$, and $\boldsymbol{V_3}$ via `my.pca$u`:[4]

```
# Re-scaled version of "derived" variables:
as.data.frame(my.pca$u)

##            V1          V2            V3
## 1 -0.23467768 -0.47558005  8.477912e-01
## 2 -0.44432759  0.02212828 -1.105815e-01
## 3 -0.02659906  0.40738085  2.211629e-01
## 4  0.18305085 -0.09032748  3.382017e-14
## 5  0.60077938  0.29492509  3.317444e-01
## 6 -0.44432759  0.02212828 -1.105815e-01
## 7 -0.02659906  0.40738085  2.211629e-01
## 8  0.39270075 -0.58803582 -2.211629e-01
```

The *actual* variables $\boldsymbol{V_1}$, $\boldsymbol{V_2}$, and $\boldsymbol{V_3}$ given by (1)-(3) are obtained by multiplying each column of `my.pca$u` by its `my.pca$d` value. We can do this with `mutate()`:

```
# Multiply each variable (column) of u by its d value, overwriting
# the original V1, V2, and V2 variables:
V <- mutate(as.data.frame(my.pca$u),
            V1 = V1*my.pca$d[1],
            V2 = V2*my.pca$d[2],
            V3 = V3*my.pca$d[3])
V

##            V1           V2 V3
## 1  -6.1605435 -0.031378850  0
## 2 -11.6640807  0.001460028  0
## 3  -0.6982541  0.026879055  0
## 4   4.8052831 -0.005959822  0
## 5  15.7711097  0.019459205  0
## 6 -11.6640807  0.001460028  0
## 7  -0.6982541  0.026879055  0
## 8  10.3088203 -0.038798699  0
```

Ordered left to right, `V1` is the *most* informative, `V2` the *second most*, and `V3` the *least* informative.

In this example, we'd **discard** `V2` and `V3` because essentially all of the information in the data is contained in `V1` (based on the `my.pca$d` values). We could then use this one variable, `V1`, for prediction, identifying clusters, etc.

[3] The $\boldsymbol{d}$ values in `my.pca$d` are the square roots of the sums of squares for each $\boldsymbol{V_j}$, e.g. `my.pca$d[1]` is `sqrt(sum(V$V1^2))`, and they measure how much the values in each $\boldsymbol{V_j}$ *vary*.

[4] Sometimes these *re-scaled* $\boldsymbol{V_j}$'s are plotted and used for prediction, cluster identification, etc. instead of the *actual* $\boldsymbol{V_j}$'s. This is what the textbook does.

Section 12.3 Exercises

Exercise 6 The goal of **PCA** is to obtaing *substantial* **dimension reduction** in a data set while at the same time *retaining* most of the information contained in the data.

Consider again the `wine` data set (described above) from the `"rattle"` package:

```
library(rattle)

head(wine)
```

The first column (`Type`) is a categorical variable, so it shouldn't be included in the PCA analysis:

```
library(dplyr)     # For select() and summarize()

# The Type column gets removed:
wine2 <- select(wine, -Type)
```

a) The means of the thirteen variables, `Alcohol`, `Malic`, ..., `Proline`, can be viewed using:

```
summarise(wine2, across(everything(), list(mean = mean)))
```

Center the thirteen variables (by subtracting their means) by typing:

```
wine2_cntr <- scale(wine2, center = TRUE, scale = FALSE) %>%
                     as.data.frame()

head(wine2_cntr, n = 3)
```

(Output not shown.)

Now perform **PCA** (i.e. **SVD** on the **centered** variables):

```
my.pca <- svd(wine2_cntr)
```

The **weights** in the **"derived"** variables $\boldsymbol{V_1, V_2, V_3,}$ ... $\boldsymbol{V_{13}}$ are the columns of `my.pca$v`:

```
my.pca$v
```

(Output not shown.)

Thus for example, the thirteen **"derived"** variables are:

$$\begin{aligned}
V_1 &= -0.0017 \times \texttt{Alcohol} + 0.0007 \times \texttt{Malic} + \cdots - 0.9998 \times \texttt{Proline} \\
V_2 &= -0.0012 \times \texttt{Alcohol} - 0.0022 \times \texttt{Malic} + \cdots + 0.0178 \times \texttt{Proline} \\
\vdots &\ \ \vdots \ \ \vdots \\
V_{13} &= -0.0012 \times \texttt{Alcohol} - 0.0022 \times \texttt{Malic} + \cdots + 0.0178 \times \texttt{Proline}
\end{aligned}$$

where `Alcohol`, `Malic`, ..., `Proline` are their *centered* versions.

Of these, $\boldsymbol{V_1}$ is the *most* informative, $\boldsymbol{V_2}$ the *second most* informative, etc. The relative amounts of information contained in the V_j's are given by the $\boldsymbol{d}$ values:

```
my.pca$d

##  [1] 4190.312249  174.753375   40.872315   29.722695   14.748071
##  [6]   12.201160    7.026970    5.176339    4.454338    3.562494
## [11]    2.578943    1.931271    1.205013
```

In particular, they measure **variation** of values within each V_j. Larger values indicate more information.

For **dimension reduction**, we'd *discard* the *less informative* $\boldsymbol{V_j}$'s (i.e. $\boldsymbol{V_{13}}$, $\boldsymbol{V_{12}}$, ...), and just keep the *more informative* ones ($\boldsymbol{V_1}$, $\boldsymbol{V_2}$, ...) for use as explanatory variables in random forests, cluster analyses, etc.

To decide **how many** of the $\boldsymbol{V_j}$'s to *keep* and how many to *discard*, we can **plot** the values in `my.pca$d`:

```
library(ggplot2)

ggplot(data = data.frame(d = my.pca$d, j = 1:13),
       mapping = aes(x = j, y = d)) +
  geom_point() +
  geom_line() +
  ggtitle("Plot of d vs j")
```

The $\boldsymbol{V_j}$'s whose $\boldsymbol{d}$ values are *close to zero* carry very *little information* and can be *discarded.* **How many** of the *more informative* $\boldsymbol{V_j}$'s would you suggest *keeping*?

b) For **PCA**, if the variables are measured on very **different scales**, it's best to ***standardize*** each one first so that their **scales** don't influence the results.

Re-run the **PCA** after ***standardizing*** each of the 13 variables in `wine2`:

```
# Standardize each of the 13 variables:
wine2_std <- scale(wine2, center = TRUE, scale = TRUE)

my.pca_std <- svd(wine2_std)

my.pca_std$d
```

To decide **how many** of the V_j's to *keep* and how many to *discard*, we can **plot** the values in `my.pca_std$d`:

```
ggplot(data = data.frame(d = my.pca_std$d, j = 1:13),
       mapping = aes(x = j, y = d)) +
  geom_point() +
  geom_line() +
  ggtitle("Plot of d vs j")
```

The V_j's whose d values are *close to zero* carry very *little information* and can be *discarded.* Now **how many** of the *more informative* V_j's would you suggest *keeping*?

Exercise 7 Sometimes the **weights** used to form a **"derived"** variable in **PCA** can help us **interpret** that variable.

Consider again the built-in `iris` data set. We'll use just the *Virginica* species:

```
library(dplyr)      # For filter(), select().

virginica <- iris %>%
               filter(Species == "virginica") %>%
               select(-Species)
```

The means of the four variables, `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`, are:

```
colMeans(virginica)

## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##        6.588        2.974        5.552        2.026
```

Center the four variables (by subtracting their means) by typing:

```
virginica_cntr <- scale(virginica, center = TRUE, scale = FALSE) %>%
                    as.data.frame()

head(virginica_cntr, n = 3)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1       -0.288       0.326        0.448       0.474
## 2       -0.788      -0.274       -0.452      -0.126
## 3        0.512       0.026        0.348       0.074
```

Now perform **PCA** (i.e. **SVD** on the **centered** variables):

```
my.pca <- svd(virginica_cntr)
```

The **weights** in the **"derived"** variables $V_1, V_2, V_3,$ and V_4 are the columns of `my.pca$v`:

```
my.pca$v

##           [,1]       [,2]       [,3]       [,4]
## [1,] 0.7410168 -0.1652590  0.5344502  0.3714117
## [2,] 0.2032877  0.7486428  0.3253749 -0.5406841
## [3,] 0.6278918 -0.1694278 -0.6515236 -0.3905934
## [4,] 0.1237745  0.6192880 -0.4289653  0.6458723
```

Thus the **"derived"** variables are:

$$\begin{aligned}
V_1 &= 0.74 \times \texttt{Sepal.Length} + 0.20 \times \texttt{Sepal.Width} + 0.63 \times \texttt{Petal.Length} \\
&\quad +0.12 \times \texttt{Petal.Width} \\
V_2 &= -0.17 \times \texttt{Sepal.Length} + 0.75 \times \texttt{Sepal.Width} - 0.17 \times \texttt{Petal.Length} \\
&\quad +0.62 \times \texttt{Petal.Width} \\
V_3 &= 0.53 \times \texttt{Sepal.Length} + 0.33 \times \texttt{Sepal.Width} - 0.65 \times \texttt{Petal.Length} \\
&\quad -0.43 \times \texttt{Petal.Width} \\
V_4 &= 0.37 \times \texttt{Sepal.Length} - 0.54 \times \texttt{Sepal.Width} - 0.39 \times \texttt{Petal.Length} \\
&\quad +0.65 \times \texttt{Petal.Width}
\end{aligned}$$

where `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width` are their *centered* versions.

Of these, $\boldsymbol{V_1}$ is the *most* informative, $\boldsymbol{V_2}$ the *second most* informative, etc. The relative amounts of information contained in the V_j's are given by:

```
my.pca$d

## [1] 5.836736 2.284953 1.600774 1.295773
```

For **dimension reduction**, we'd probably *discard* $\boldsymbol{V_3}$ and $\boldsymbol{V_4}$, and just keep $\boldsymbol{V_1}$ and $\boldsymbol{V_2}$ (for use as explanatory variables in random forests, cluster analyses, etc.).

Sometimes the **weights** can help us **interpret** a **"derived"** variable. Look at the **weights** in $\boldsymbol{V_1}$ and $\boldsymbol{V_2}$. One of these two **"derived"** variables is largely reflecting *length* aspects of the flower and the other *width* aspects. Which variable, $\boldsymbol{V_1}$ or $\boldsymbol{V_2}$, is reflecting *length* and which is reflecting *width*?