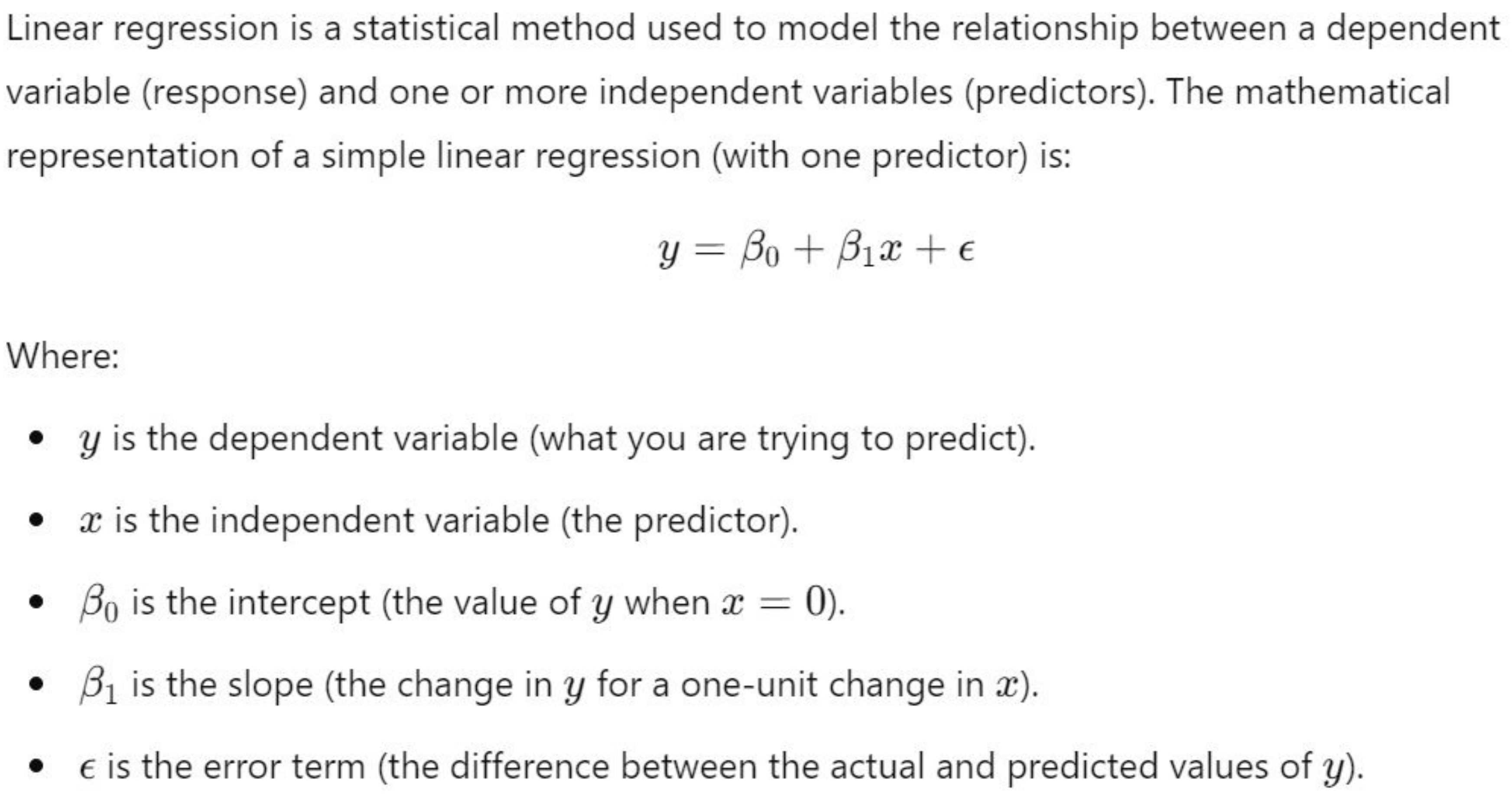
# Linear Regression in R



Linear regression is a statistical method used to model the relationship between a dependent variable (response) and one or more independent variables (predictors). The mathematical representation of a simple linear regression (with one predictor) is:

$$y = \beta_0 + \beta_1 x + \epsilon$$

Where:

- $y$ is the dependent variable (what you are trying to predict).
- $x$ is the independent variable (the predictor).
- $\beta_0$ is the intercept (the value of $y$ when $x = 0$).
- $\beta_1$ is the slope (the change in $y$ for a one-unit change in $x$).
- $\epsilon$ is the error term (the difference between the actual and predicted values of $y$).

# Steps to implement Linear Regression in R

Step 1: Load the data into R

Step 2: Make sure your data meet the assumptions

Step 3: Perform the linear regression analysis

Step 4: Visualize the results with a graph

## Step 1 : Load the data into R

1. In RStudio, go to File > Import dataset  > From Text (base).
2. Choose the data file , and an Import Dataset window pops up.
3. Click on the Import button and the file should appear in your Environment tab on the upper right side of the RStudio screen.

## Step 2: Make sure your data meet the assumptions

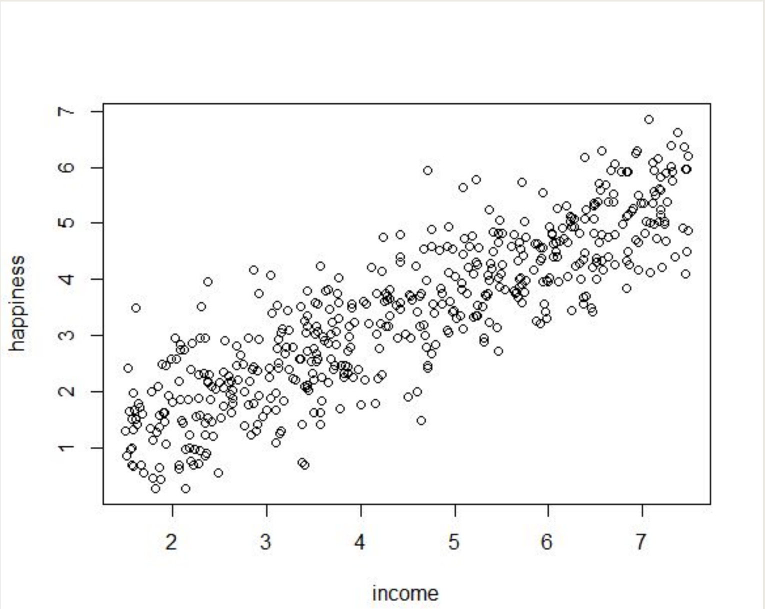
Simple linear regression is a parametric test, meaning that it makes certain assumptions about the data. These assumptions are:

1. Homogeneity of variance (homoscedasticity):This means that the prediction error doesn’t change significantly over the range of prediction of the model
2. Independence of observations: The independent variables are not highly correlated with each other.(no multicollinearity)
3. Normality: The residual data follows a normal distribution.
4. The relationship between the independent and dependent variable is linear: the line of best fit through the data points is a straight line (rather than a curve or some sort of grouping factor).

# Linearity

The relationship between the independent and dependent variable must be linear. We can test this visually with a scatter plot to see if the distribution of data points could be described with a straight line.

```
plot(happiness ~ income, data = income.data)
```



## Multiple regression

Independence of observations (aka no autocorrelation) Use the **cor()** function to test the relationship between your independent variables and make sure they aren't too highly correlated.

```
cor(heart.data$biking, heart.data$smoking)
```

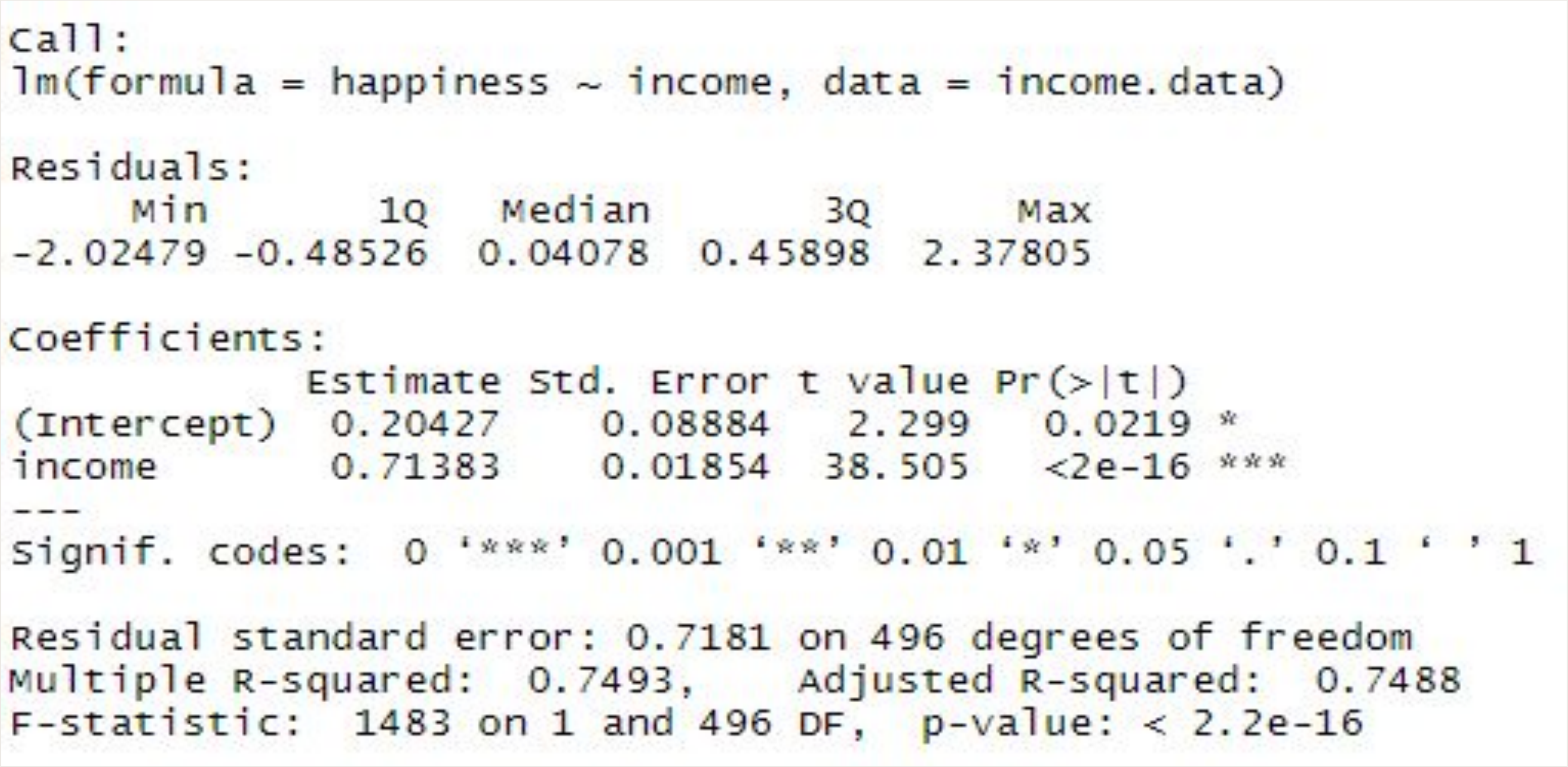
## Step 3: Perform the linear regression analysis

Now that you’ve determined your data meet the assumptions, you can perform a linear regression analysis to evaluate the relationship between the independent and dependent variables.

### Simple regression: income and happiness

```
income.happiness.lm <- lm(happiness ~ income, data = income.data)

summary(income.happiness.lm)
```

# The output looks like this:



```
Call:
lm(formula = happiness ~ income, data = income.data)

Residuals:
     Min       1Q   Median       3Q      Max 
-2.02479 -0.48526  0.04078  0.45898  2.37805 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.20427    0.08884   2.299   0.0219 *  
income       0.71383    0.01854  38.505   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7181 on 496 degrees of freedom
Multiple R-squared:  0.7493,	Adjusted R-squared:  0.7488 
F-statistic:  1483 on 1 and 496 DF,  p-value: < 2.2e-16
```

# Interpretation of Regression Output

## Residuals

The residuals are the difference between the actual values and the predicted values.

## Coefficients

### Coefficients – Estimate

Estimates for the coefficients provided in the output above, we can now build out the equation for our model.

### Coefficients – Std. Error

The standard error of the coefficient is an estimate of the standard deviation of the coefficient. In effect, it is telling us how much uncertainty there is with our coefficient.

## Coefficients — t value

The t-statistic is simply the coefficient divided by the standard error. In general, we want our coefficients to have large t-statistics, because it indicates that our standard error is small in comparison to our coefficient.

## Coefficients — Pr(>|t|) and Signif. codes

The p-value is calculated using the t-statistic from the T distribution. The p-value, in association with the t-statistic, help us to understand how *significant* our coefficient is to the model. In practice, any p-value below 0.05 is usually deemed as *significant*.

## Residual Standard Error

The residual standard error is a measure of how well the model fits the data.

## Multiple R-squared and Adjusted R-squared

The Multiple R-squared value is most often used for simple linear regression (one predictor). It tells us what percentage of the variation within our dependent variable that the independent variable is explaining. In other words, it’s another method to determine how well our model is fitting the data.

The Adjusted R-squared value shows what percentage of the variation within our dependent variable that all independent variables are explaining.

## F-statistic and p-value

When running a regression model, either simple or multiple, a hypothesis test is being run on the global model. The null hypothesis is that there is no relationship between the dependent variable and the independent variable(s) and the alternative hypothesis is that there is a relationship. Said another way, the null hypothesis is that the coefficients for all of the variables in your model are zero. The alternative hypothesis is that at least one of them is **not** zero. The F-statistic and overall p-value help us determine the result of this test

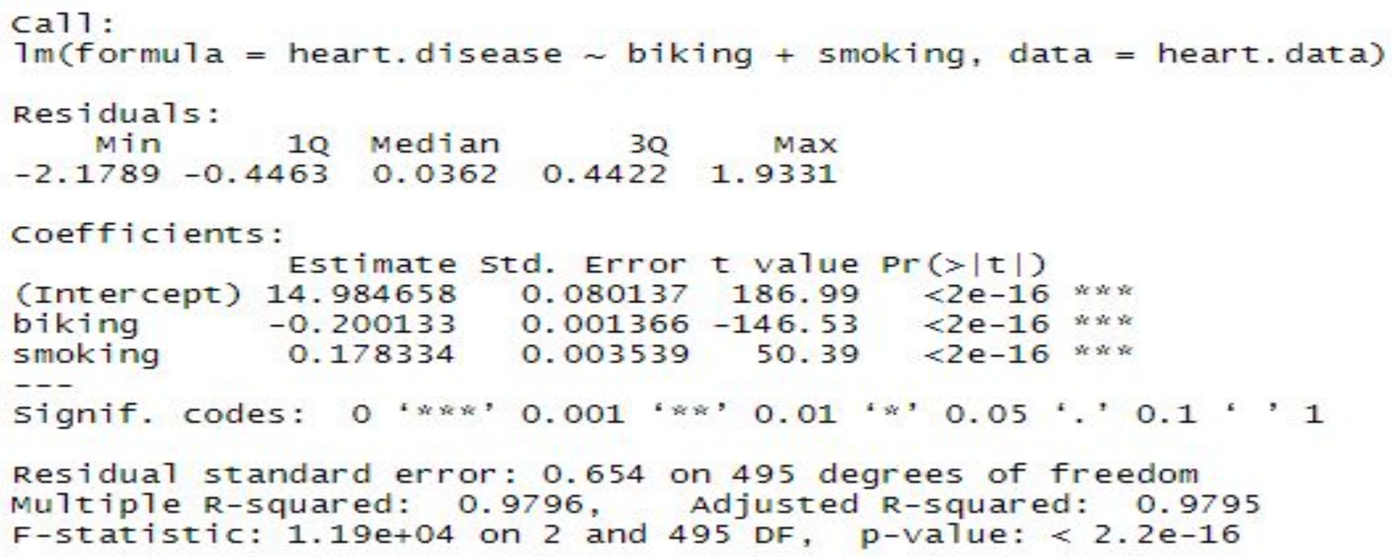
However, for smaller models, a larger F-statistic generally indicates that the null hypothesis should be rejected. A better approach is to utilize the p-value that is associated with the F-statistic. Again, in practice, a p-value below 0.05 generally indicates that you have at least one coefficient in your model that isn't zero.

## Multiple regression: biking, smoking, and heart disease

heart.disease.lm<-lm(heart.disease ~ biking + smoking, data = heart.data)

summary(heart.disease.lm)

The output looks like this:



```
Call:
lm(formula = heart.disease ~ biking + smoking, data = heart.data)

Residuals:
    Min      1Q  Median      3Q     Max 
-2.1789 -0.4463  0.0362  0.4422  1.9331 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 14.984658   0.080137  186.99   <2e-16 ***
biking      -0.200133   0.001366 -146.53   <2e-16 ***
smoking      0.178334   0.003539   50.39   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.654 on 495 degrees of freedom
Multiple R-squared:  0.9796,	Adjusted R-squared:  0.9795 
F-statistic: 1.19e+04 on 2 and 495 DF,  p-value: < 2.2e-16
```

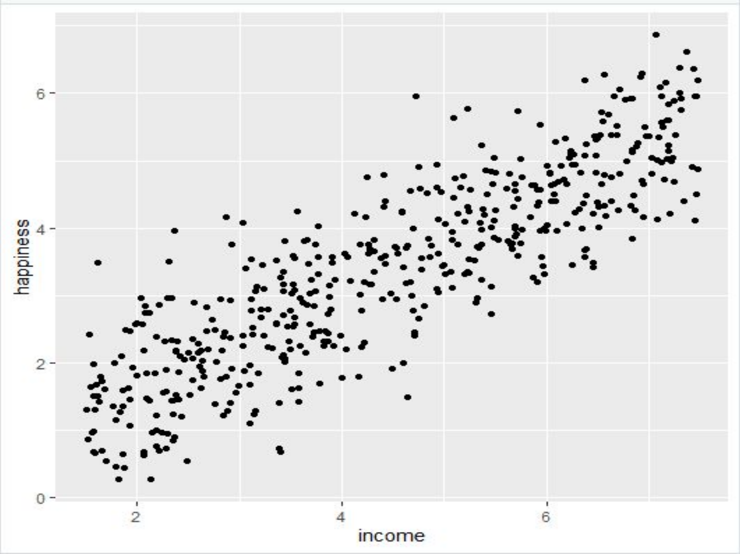
# Step 5: Visualize the results with a graph

## Simple regression

Plot the data points on a graph

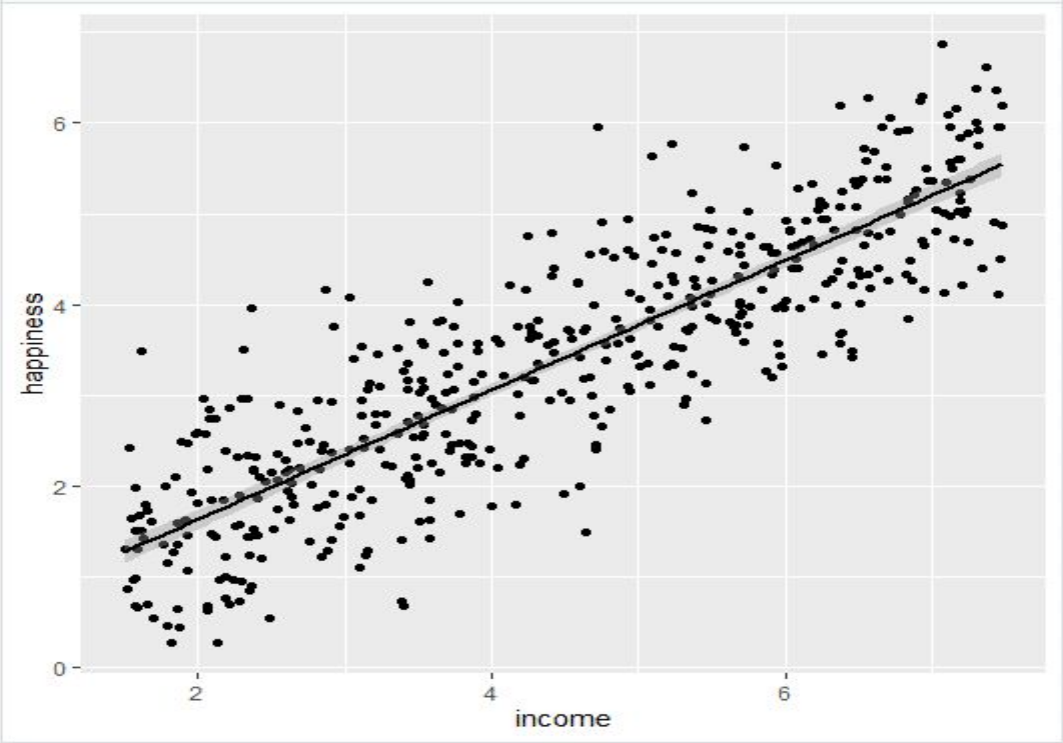
```
income.graph<-ggplot(income.data, aes(x=income, y=happiness))
```

```
income.graph
```



# Add the linear regression line to the plotted data

```
income.graph <- income.graph + geom_smooth(method="lm", col="black")

income.graph
```



Once we’ve fit a model, we can then use the **predict()** function to predict the response value of a new observation.

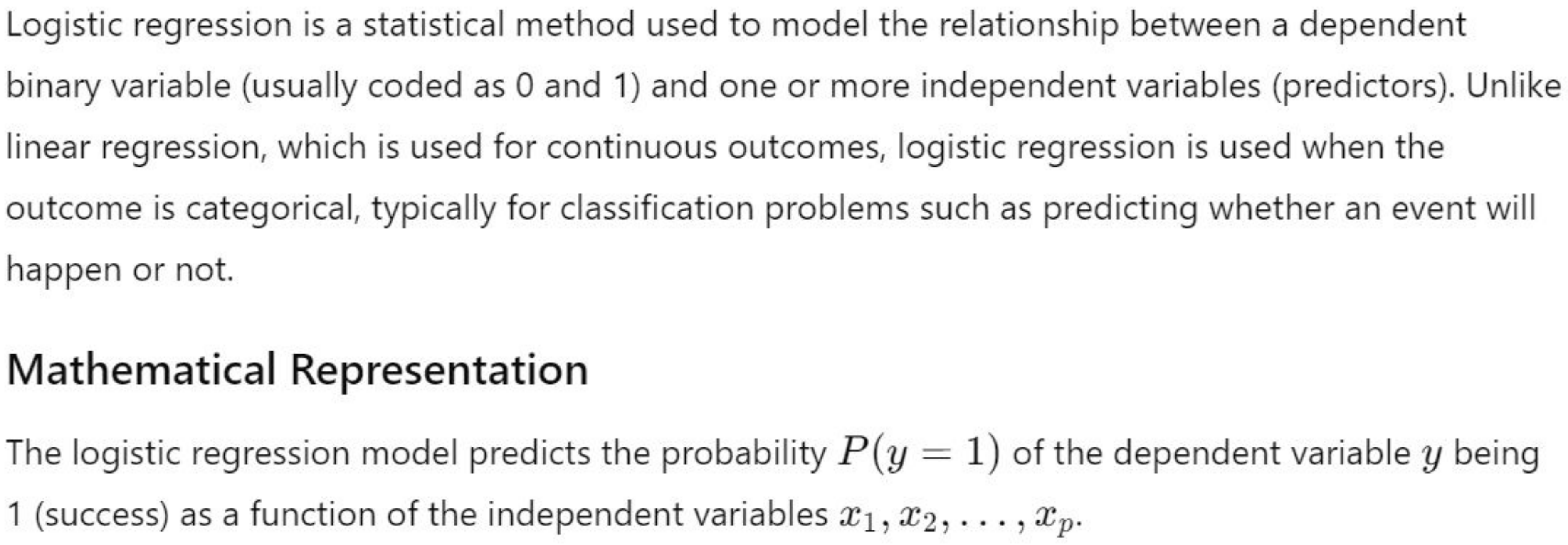
This function uses the following syntax:

**predict(object, newdata)**

where:

- **object:** The name of the model fit using the glm() function
- **newdata:** The name of the new data frame to make predictions for

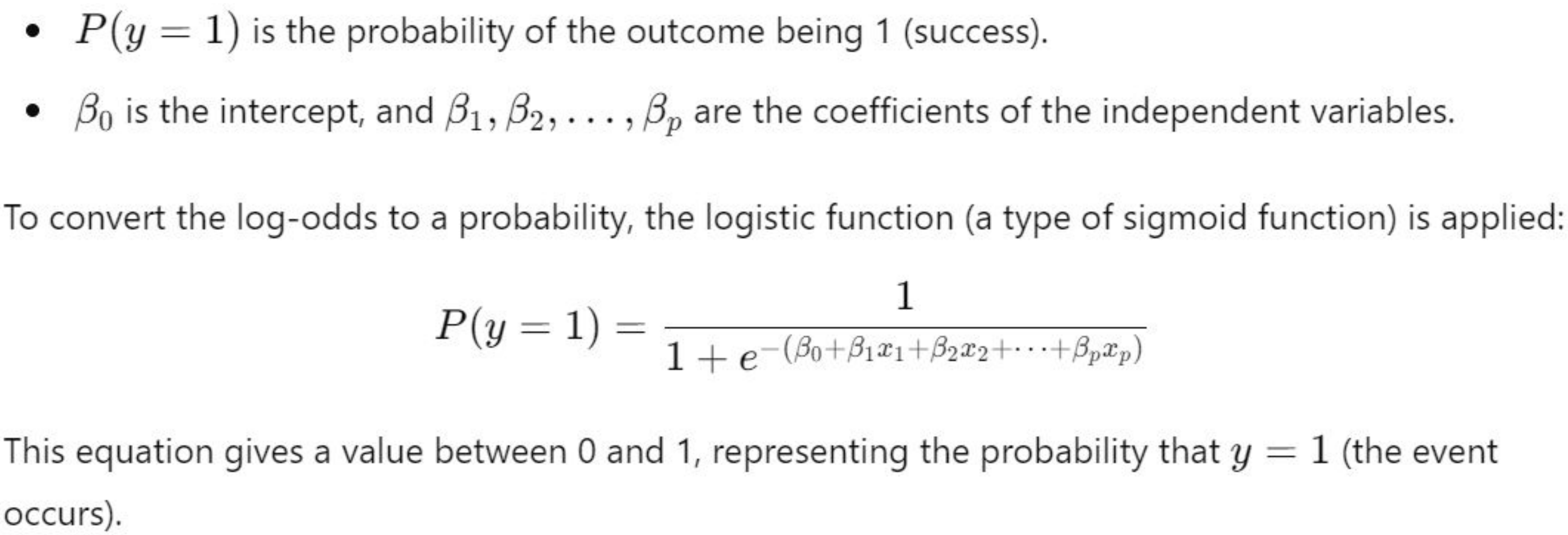
# What is Logistic Regression?



Logistic regression is a statistical method used to model the relationship between a dependent binary variable (usually coded as 0 and 1) and one or more independent variables (predictors). Unlike linear regression, which is used for continuous outcomes, logistic regression is used when the outcome is categorical, typically for classification problems such as predicting whether an event will happen or not.

## Mathematical Representation

The logistic regression model predicts the probability $P(y = 1)$ of the dependent variable $y$ being 1 (success) as a function of the independent variables $x_1, x_2, \ldots, x_p$.



- $P(y=1)$ is the probability of the outcome being 1 (success).
- $\beta_0$ is the intercept, and $\beta_1, \beta_2, \ldots, \beta_p$ are the coefficients of the independent variables.

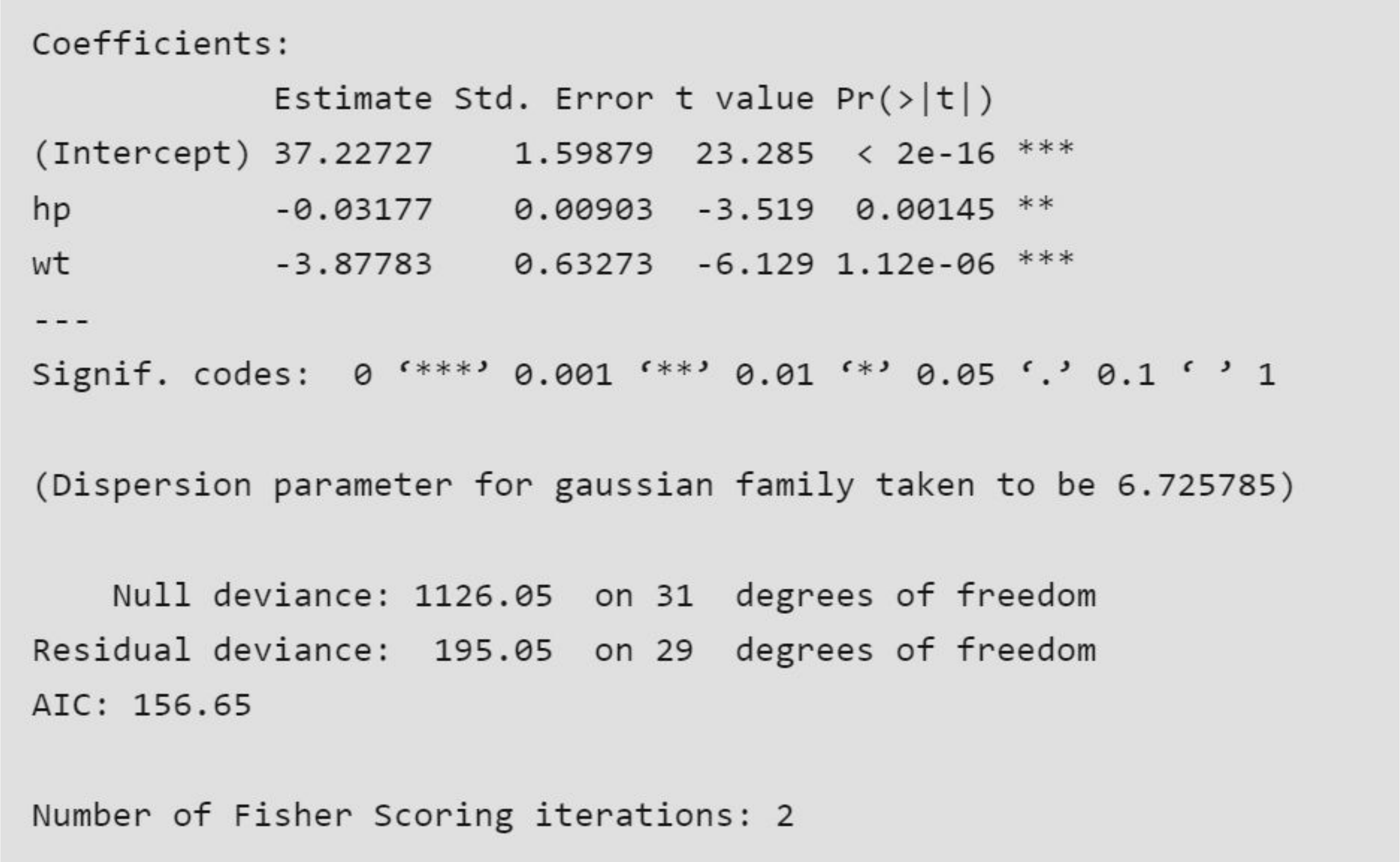
To convert the log-odds to a probability, the logistic function (a type of sigmoid function) is applied:

$$P(y=1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p)}}$$

This equation gives a value between 0 and 1, representing the probability that $y = 1$ (the event occurs).

In R, you can use the `glm()` function (Generalized Linear Model) to perform logistic regression. The `glm()` function allows you to fit a variety of generalized linear models, including logistic regression, by specifying the appropriate family of distribution and link function.

```
model <- glm(binary_response_variable ~ predictor_variable1
 + predictor_variable2,     family = binomial(link = "logit"), data = data)

#Calculate summary of the model
summary(model)
```



```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
hp          -0.03177    0.00903  -3.519  0.00145 ** 
wt          -3.87783    0.63273  -6.129 1.12e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 6.725785)

    Null deviance: 1126.05  on 31  degrees of freedom
Residual deviance:  195.05  on 29  degrees of freedom
AIC: 156.65

Number of Fisher Scoring iterations: 2
```

**Coefficients:**

- This table provides estimates of the regression coefficients for the intercept and the variables (predictors). Each row represents a different variable in the model.

**Std. Error (Standard Error):**

- The standard error of each estimate, which measures the accuracy of the coefficient's estimate.
- For example, the standard error for hp is 0.00903, indicating the variability of the hp coefficient's estimate.

**t value:**

- The **t-statistic** for each coefficient is calculated as the estimate divided by its standard error.
- For example, the `t value` for `hp` is -3.519, indicating how many standard deviations the estimate is away from 0.

**Pr(>|t|) (p-value):**

- The **p-value** tests the hypothesis that the coefficient is significantly different from 0.
- A small p-value (typically ≤ 0.05) indicates that the coefficient is statistically significant.
    - `hp` has a p-value of 0.00145, meaning it is significant at the 0.01 level.
    - `wt` has a very small p-value (1.12e-06), meaning it is highly significant.

## significance Codes:

- `***`: Highly significant (p-value < 0.001).
- `**`: Moderately significant (p-value < 0.01).
- `*`: Significant (p-value < 0.05).
- `.`: Marginally significant (p-value < 0.1).
- Blank: Not significant.

## Null Deviance:

- The **null deviance** measures the goodness-of-fit of a model with only the intercept (no predictors).

**Residual Deviance:**

- The **residual deviance** measures the goodness-of-fit of the full model (with predictors). Lower residual deviance indicates a better fit.

**AIC (Akaike Information Criterion):**

- The **AIC** is a metric used to compare models. Lower AIC values indicate a better fit, considering both the goodness-of-fit and model complexity (penalizing models with more parameters).

**Number of Fisher Scoring Iterations:**

- This refers to the number of iterations the algorithm used to estimate the model’s coefficients. In this case, it converged in 2 iterations, which indicates the model fit successfully.

# The End

# Unit -3

## Topic: Unsupervised methods

### 1.Introduction:

- **Unsupervised methods** - discovering hidden relationships in data.
- No specific outcome or prediction is involved.
- Focus is on finding **patterns** or **groupings** within the data.
- Examples include:
    - **Grouping customers** with similar purchase behaviors.
    - Identifying **correlations** between population movement and socioeconomic factors.
- Used to explore and understand data structure rather than making predictions.

### Two classes of unsupervised methods:

★ **Cluster analysis -** finds groups with similar characteristics.

★ **Association rule mining -** finds elements or properties in the data that tend to occur together.

### Cluster Analysis:

**Cluster analysis** groups observations into clusters where each datum is more similar to others in the same cluster than to those in different clusters.

**Example**: A tour company could cluster clients based on:

- **Preferred destinations** (countries they like to visit).
- **Tour preferences** (adventure, luxury, or educational tours).
- **Types of activities** clients engage in.

To help the company design appealing travel packages and better target specific client segments.

**Two approaches to clustering:**

1. **K-means clustering**: A **fast and popular** method for identifying clusters in **quantitative data**.
2. **Hierarchical clustering**: Finds **nested groups** of clusters, similar to plant taxonomy (family, then genus, then species).

## 1.1 Distances

**Different notions of distance:**

- Euclidean distance
- Hamming distance
- Manhattan (city block) distance
- Cosine similarity

### EUCLIDEAN DISTANCE :

- Euclidean distance is a good choice for clustering when measurements are numerical and continuous.
- K-means clustering is based on optimizing squared Euclidean distance.
- For categorical data, especially binary, other distance metrics should be used.
- Formula for Euclidean distance between two vectors :

$$\mathrm{edist}(x, y) = \sqrt{(x[1] - y[1])^2 + (x[2] - y[2])^2 + \ldots}$$

### HAMMING DISTANCE

When all variables are **categorical**, use Hamming distance, which counts mismatches

$$\mathrm{hdist}(x, y) = \sum((x[1] \neq y[1]) + (x[2] \neq y[2]) + \ldots)$$

For **categorical variables** (e.g., recipe ingredients, gender, size), you can define the distance as:

- 0 if two points are in the same category.
- 1 if they are in different categories.

**Hamming Distance (for unordered categorical variables):**

Let’s say we have:

- Recipe 1: chicken, spicy, medium
- Recipe 2: beef, mild, medium

Now, compare the categories:

- Ingredient: chicken ≠ beef → 1
- Spice level: spicy ≠ mild → 1
- Serving size: medium = medium → 0

**The Hamming distance is:**

$$\text{hdist}(x, y) = 1 + 1 + 0 = 2$$

## MANHATTAN (CITY BLOCK) DISTANCE

Manhattan distance measures distance in the number of horizontal and vertical units it takes to get from one (real-valued) point to the other (no diagonal moves). This is also known as L1 distance.

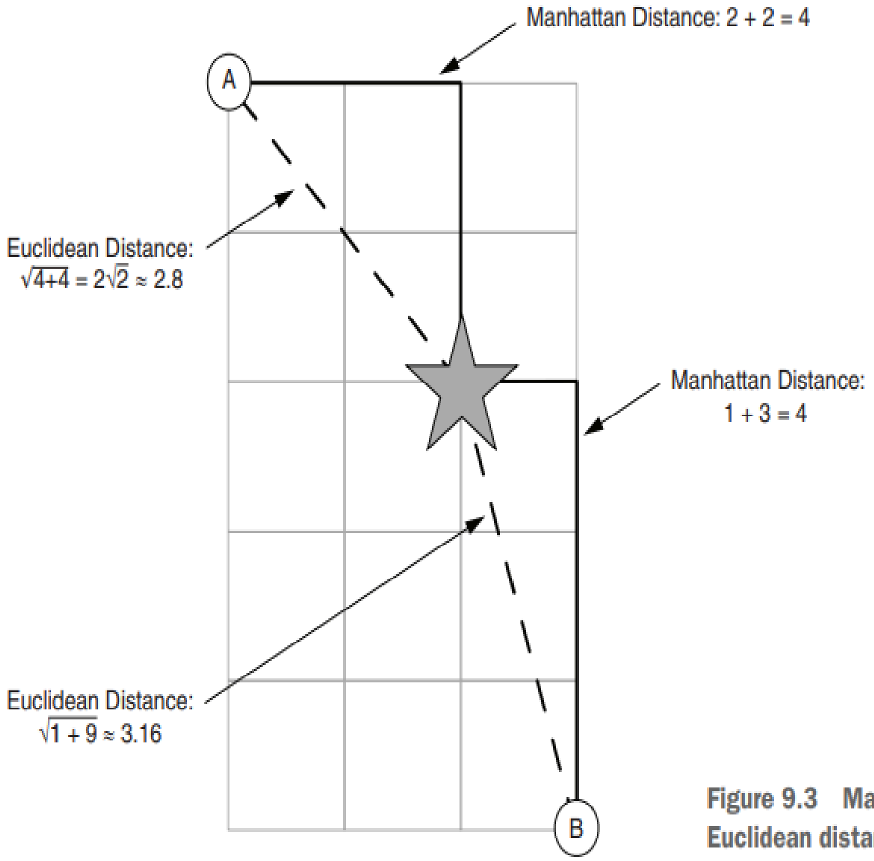
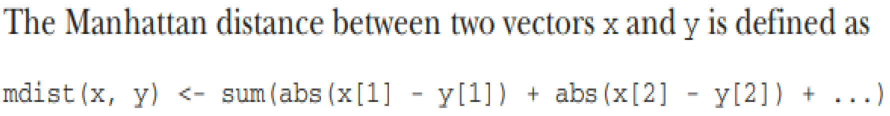
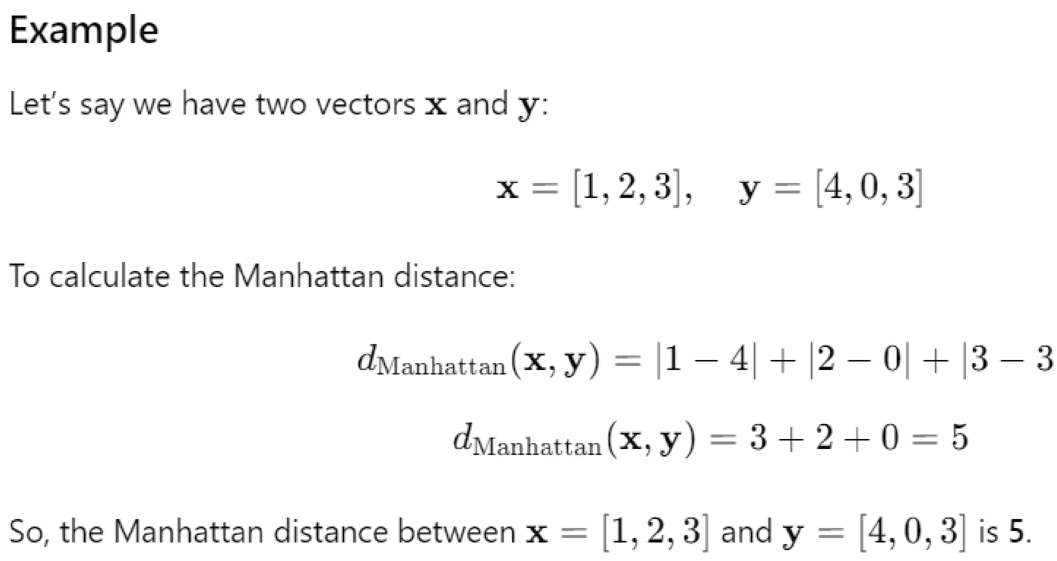


**Figure 9.3 Manhattan vs. Euclidean distance**



The Manhattan distance between two vectors x and y is defined as

```
mdist(x, y) <- sum(abs(x[1] - y[1]) + abs(x[2] - y[2]) + ...)
```



## Example

Let's say we have two vectors $\mathbf{x}$ and $\mathbf{y}$:

$$\mathbf{x} = [1, 2, 3], \quad \mathbf{y} = [4, 0, 3]$$

To calculate the Manhattan distance:

$$d_{\text{Manhattan}}(\mathbf{x}, \mathbf{y}) = |1 - 4| + |2 - 0| + |3 - 3|$$

$$d_{\text{Manhattan}}(\mathbf{x}, \mathbf{y}) = 3 + 2 + 0 = 5$$

So, the Manhattan distance between $\mathbf{x} = [1, 2, 3]$ and $\mathbf{y} = [4, 0, 3]$ is **5**.

### COSINE SIMILARITY

Cosine similarity is a common similarity metric in text analysis. It measures the smallest angle between two vectors

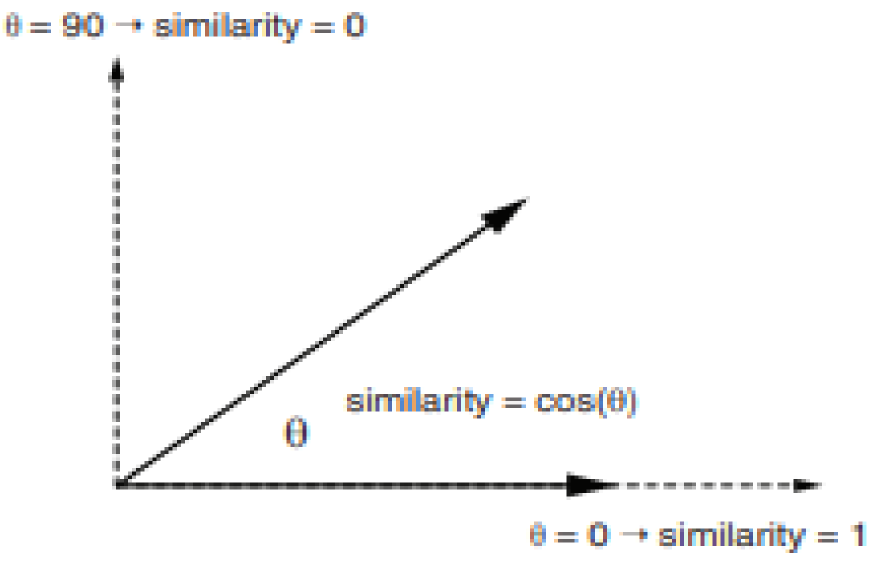


**Figure 9.4 Cosine similarity**

## K - Means Clustering:

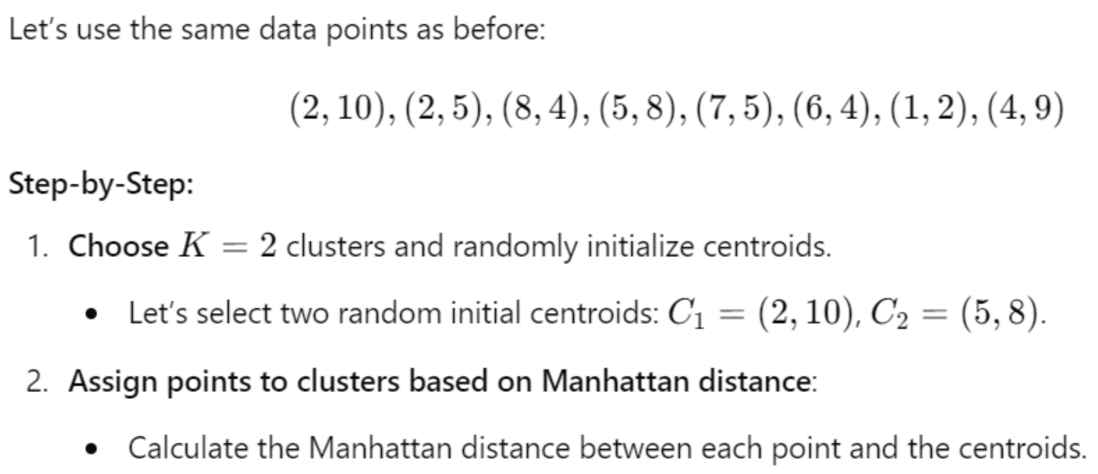
K-Means is a partition-based clustering algorithm used to divide a dataset into **K distinct non-overlapping clusters** based on feature similarity. It is an iterative algorithm that minimizes the **within-cluster sum of squares (WCSS)**, a measure of how close the data points in a cluster are to the cluster centroid.

### Steps of the K-Means Algorithm:

1. **Initialize**:
   - Choose the number of clusters K.
   - Randomly initialize K cluster centroids from the data points.
2. **Assign**:
   - For each data point, assign it to the cluster whose centroid is closest to it.
3. **Update**:
   - Recalculate the centroid of each cluster by taking the mean of all the points assigned to that cluster.
4. **Repeat**:
   - Repeat the **Assign** and **Update** steps until the centroids no longer change significantly or the algorithm reaches the maximum number of iterations.
5. **Terminate**:
   - The algorithm terminates when the centroids stabilize (i.e., do not change between iterations) or the maximum number of iterations is reached.

.

### Numerical Example with Manhattan distance:

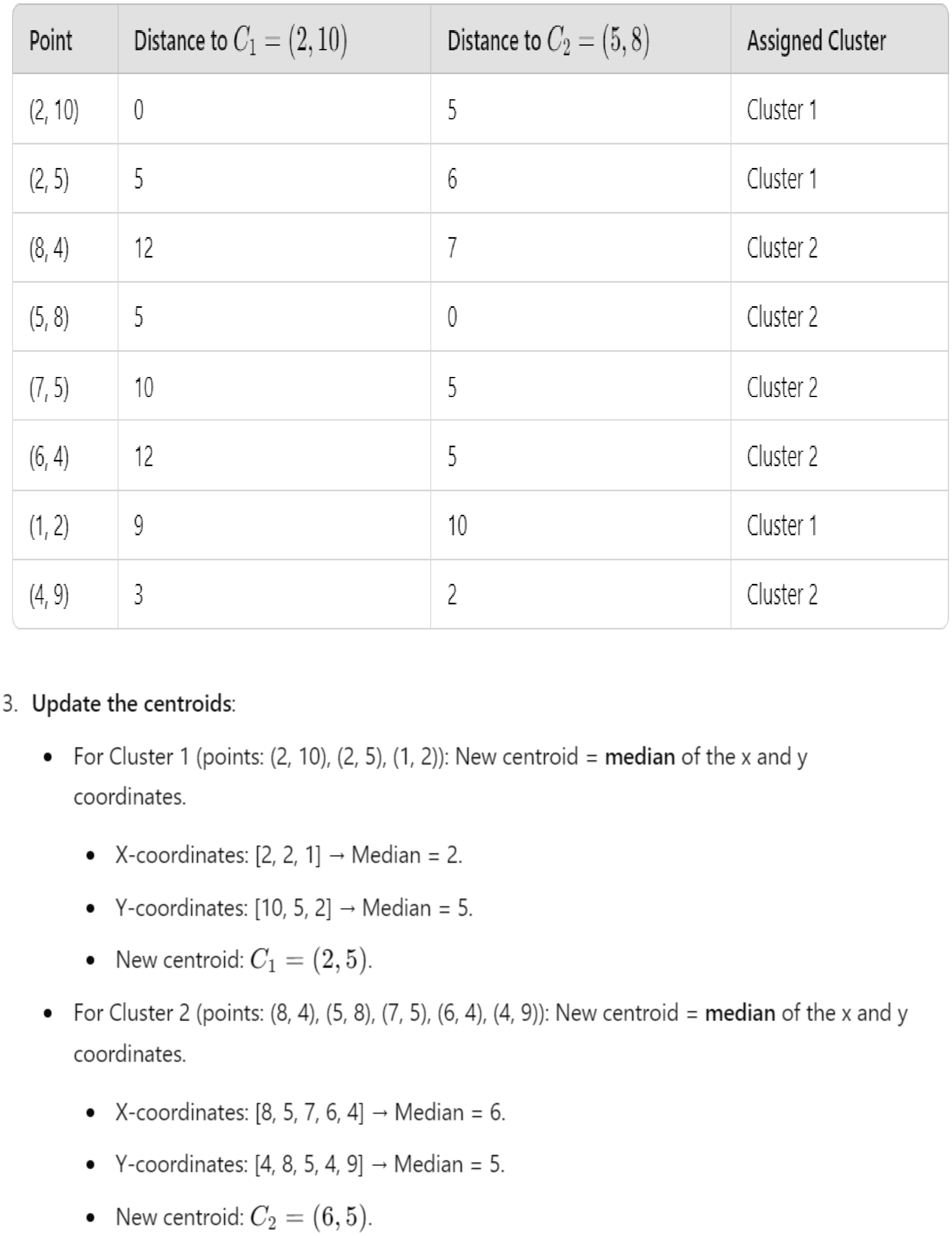


Let's use the same data points as before:

$$(2,10),(2,5),(8,4),(5,8),(7,5),(6,4),(1,2),(4,9)$$

**Step-by-Step:**

1. **Choose $K=2$ clusters and randomly initialize centroids.**
   - Let's select two random initial centroids: $C_1=(2,10)$, $C_2=(5,8)$.
2. **Assign points to clusters based on Manhattan distance**:
   - Calculate the Manhattan distance between each point and the centroids.



| Point | Distance to $C_1 = (2, 10)$ | Distance to $C_2 = (5, 8)$ | Assigned Cluster |
|---|---|---|---|
| (2, 10) | 0 | 5 | Cluster 1 |
| (2, 5) | 5 | 6 | Cluster 1 |
| (8, 4) | 12 | 7 | Cluster 2 |
| (5, 8) | 5 | 0 | Cluster 2 |
| (7, 5) | 10 | 5 | Cluster 2 |
| (6, 4) | 12 | 5 | Cluster 2 |
| (1, 2) | 9 | 10 | Cluster 1 |
| (4, 9) | 3 | 2 | Cluster 2 |

3. **Update the centroids**:
   - For Cluster 1 (points: (2, 10), (2, 5), (1, 2)): New centroid = **median** of the x and y coordinates.
     - X-coordinates: [2, 2, 1] → Median = 2.
     - Y-coordinates: [10, 5, 2] → Median = 5.
     - New centroid: $C_1 = (2, 5)$.
   - For Cluster 2 (points: (8, 4), (5, 8), (7, 5), (6, 4), (4, 9)): New centroid = **median** of the x and y coordinates.
     - X-coordinates: [8, 5, 7, 6, 4] → Median = 6.
     - Y-coordinates: [4, 8, 5, 4, 9] → Median = 5.
     - New centroid: $C_2 = (6, 5)$.

4. **Repeat the assignment step** using the updated centroids.

After a few iterations, the centroids will stabilize and the clusters will be formed.

# K-Means Clustering in R

## Step 1: Load the Necessary Packages

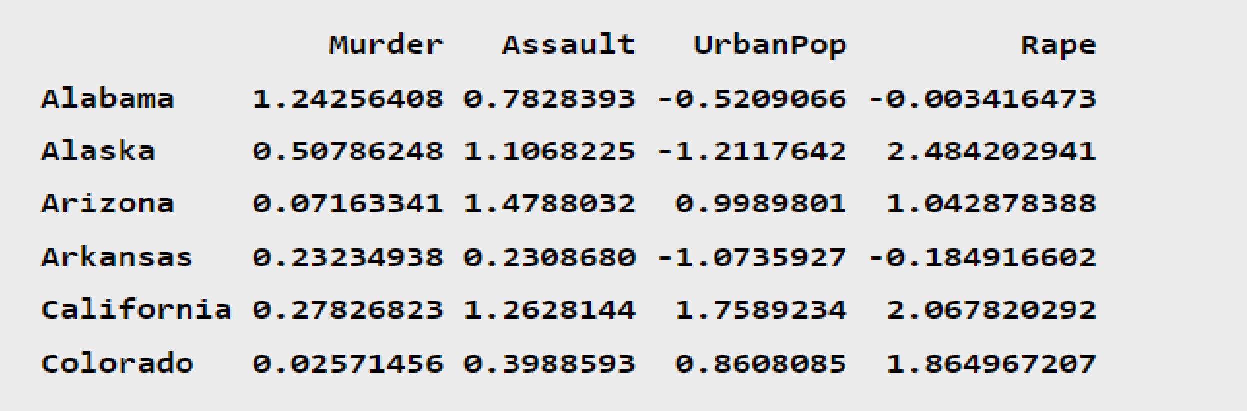
library(factoextra)

library(cluster)

## Step 2: Load and Prep the Data

- Load the  dataset
- Remove any rows with missing values
- Scale each variable in the dataset to have a mean of 0 and a standard deviation of 1

```
#load data
df <- USArrests

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

#scale each variable to have a mean of 0 and sd of 1
df <- scale(df)

#view first six rows of dataset
head(df)
```



```
                Murder   Assault    UrbanPop         Rape
Alabama     1.24256408 0.7828393 -0.5209066 -0.003416473
Alaska      0.50786248 1.1068225 -1.2117642  2.484202941
Arizona     0.07163341 1.4788032  0.9989801  1.042878388
Arkansas    0.23234938 0.2308680 -1.0735927 -0.184916602
California  0.27826823 1.2628144  1.7589234  2.067820292
Colorado    0.02571456 0.3988593  0.8608085  1.864967207
```

### Step 3: Find the Optimal Number of Clusters

To perform k-means clustering in R we can use the built-in kmeans() function, which uses the following syntax:

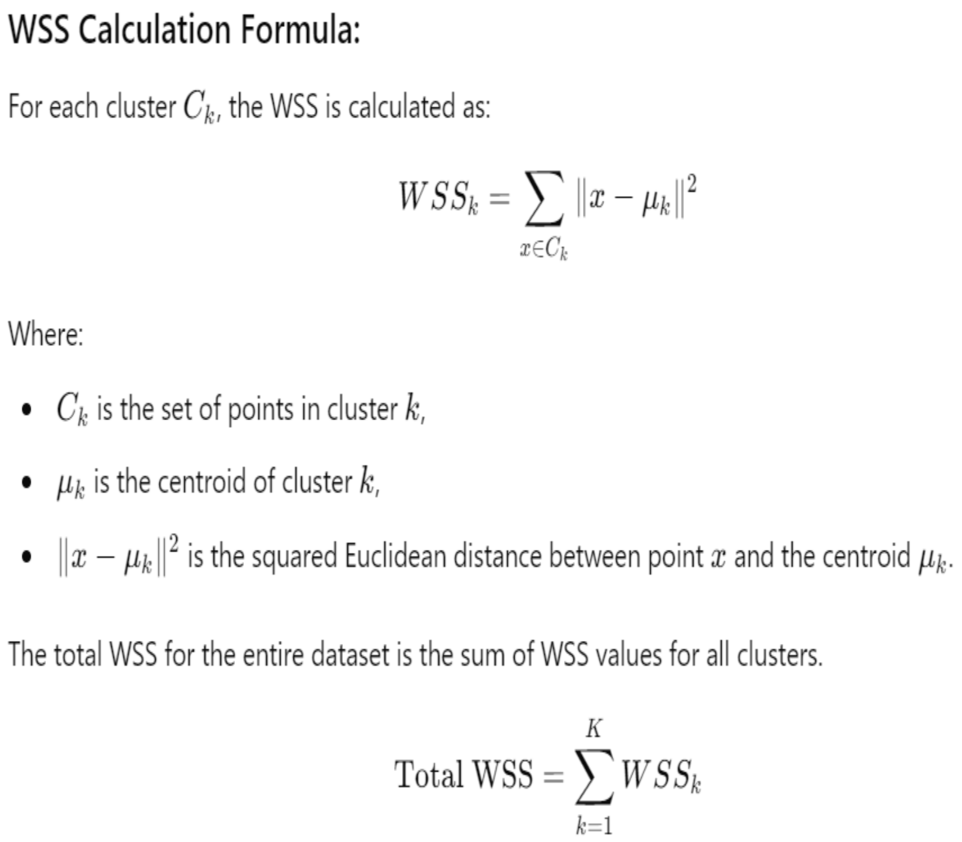
kmeans(data, centers, nstart, iter.max)

where:

- data: Name of the dataset.
- centers: The number of clusters, denoted *k*.
- nstart: This tells the `kmeans()` function to run the algorithm for specific no. of  times with different initial cluster centers and choose the best solution based on the total within-cluster sum of squares.
- Iter.max: maximum number of iterations in each configuration allowed to converge

### Method 1: Number of Clusters vs. the Total Within Sum of Squares (Elbow Method)

The **Within-Cluster Sum of Squares (WSS)** is a metric used in k-means clustering to measure the compactness of the clusters. It calculates the sum of the squared distances between each data point and the centroid of the cluster it belongs to. **Lower WSS** values indicate that the data points within a cluster are close to each other, implying well-defined clusters.



### WSS Calculation Formula:

For each cluster $C_k$, the WSS is calculated as:

$$WSS_k = \sum_{x \in C_k} \|x - \mu_k\|^2$$

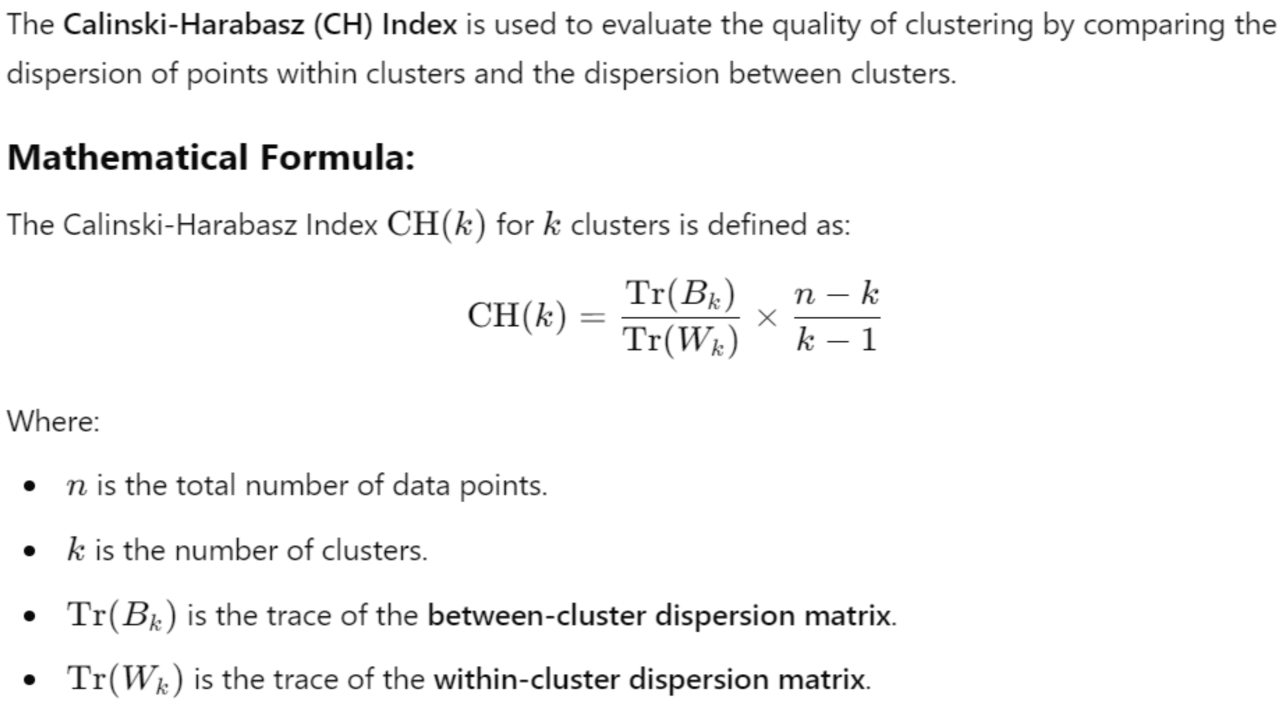
Where:

- $C_k$ is the set of points in cluster $k$,
- $\mu_k$ is the centroid of cluster $k$,
- $\|x - \mu_k\|^2$ is the squared Euclidean distance between point $x$ and the centroid $\mu_k$.

The total WSS for the entire dataset is the sum of WSS values for all clusters.

$$\text{Total WSS} = \sum_{k=1}^{K} WSS_k$$

# Method2 :



The **Calinski-Harabasz (CH) Index** is used to evaluate the quality of clustering by comparing the dispersion of points within clusters and the dispersion between clusters.

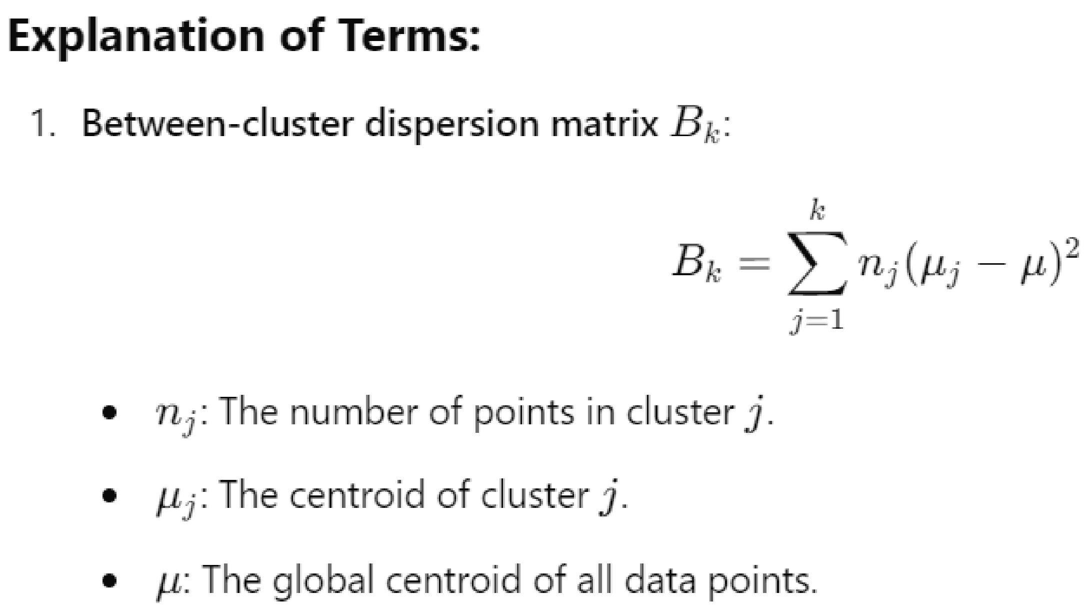
### Mathematical Formula:

The Calinski-Harabasz Index $\mathrm{CH}(k)$ for $k$ clusters is defined as:

$$\mathrm{CH}(k) = \frac{\mathrm{Tr}(B_k)}{\mathrm{Tr}(W_k)} \times \frac{n-k}{k-1}$$

Where:

- $n$ is the total number of data points.
- $k$ is the number of clusters.
- $\mathrm{Tr}(B_k)$ is the trace of the **between-cluster dispersion matrix**.
- $\mathrm{Tr}(W_k)$ is the trace of the **within-cluster dispersion matrix**.

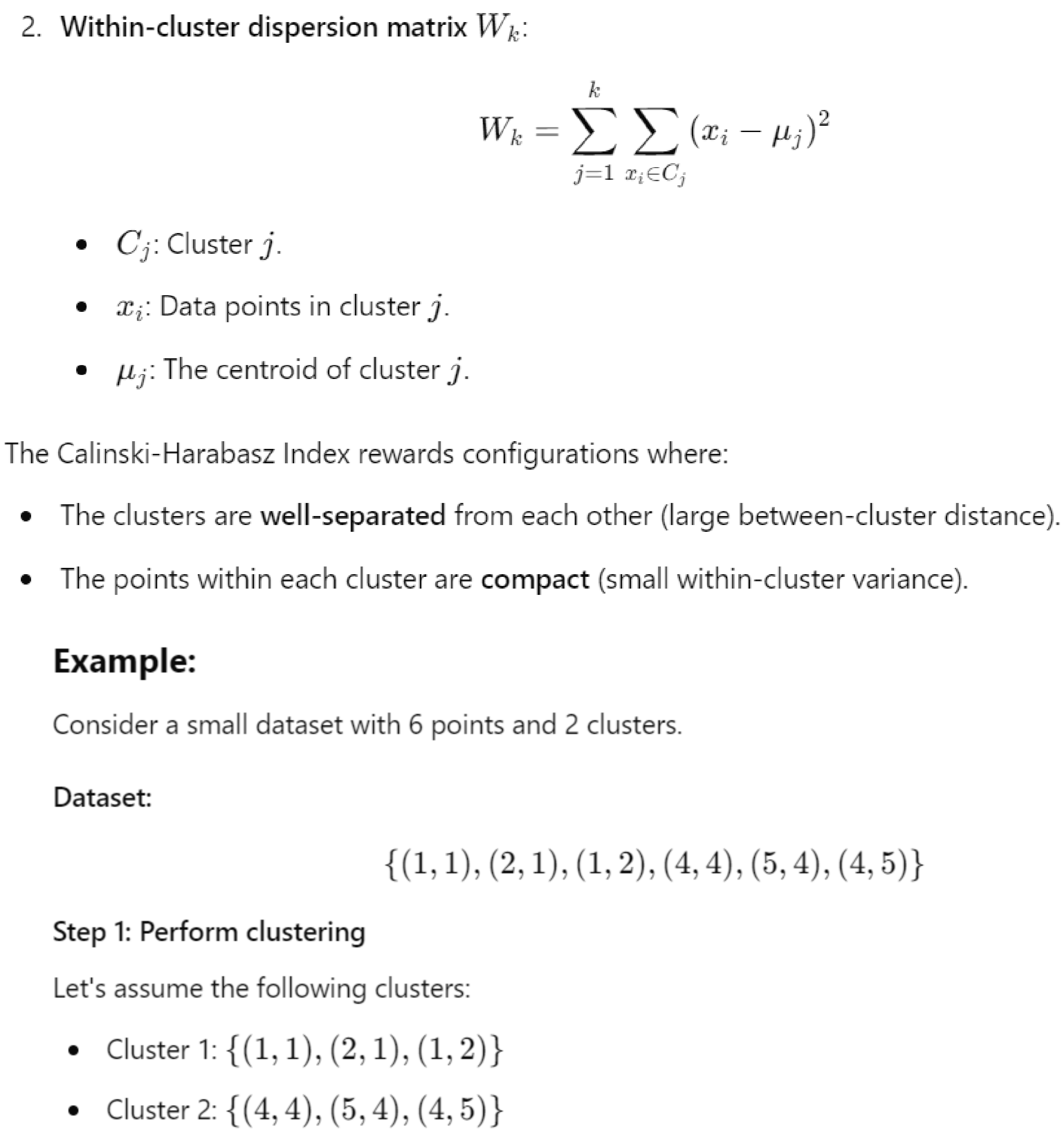


## Explanation of Terms:

1. **Between-cluster dispersion matrix $B_k$:**

$$B_k = \sum_{j=1}^{k} n_j (\mu_j - \mu)^2$$

- $n_j$: The number of points in cluster $j$.
- $\mu_j$: The centroid of cluster $j$.
- $\mu$: The global centroid of all data points.



2. **Within-cluster dispersion matrix $W_k$:**

$$W_k = \sum_{j=1}^{k} \sum_{x_i \in C_j} (x_i - \mu_j)^2$$

- $C_j$: Cluster $j$.
- $x_i$: Data points in cluster $j$.
- $\mu_j$: The centroid of cluster $j$.

The Calinski-Harabasz Index rewards configurations where:

- The clusters are **well-separated** from each other (large between-cluster distance).
- The points within each cluster are **compact** (small within-cluster variance).

### Example:

Consider a small dataset with 6 points and 2 clusters.

**Dataset:**

$$\{(1,1),(2,1),(1,2),(4,4),(5,4),(4,5)\}$$

**Step 1: Perform clustering**

Let's assume the following clusters:

- Cluster 1: $\{(1,1),(2,1),(1,2)\}$
- Cluster 2: $\{(4,4),(5,4),(4,5)\}$

### Step 2: Calculate centroids

- Centroid of Cluster 1 $\mu_1$:

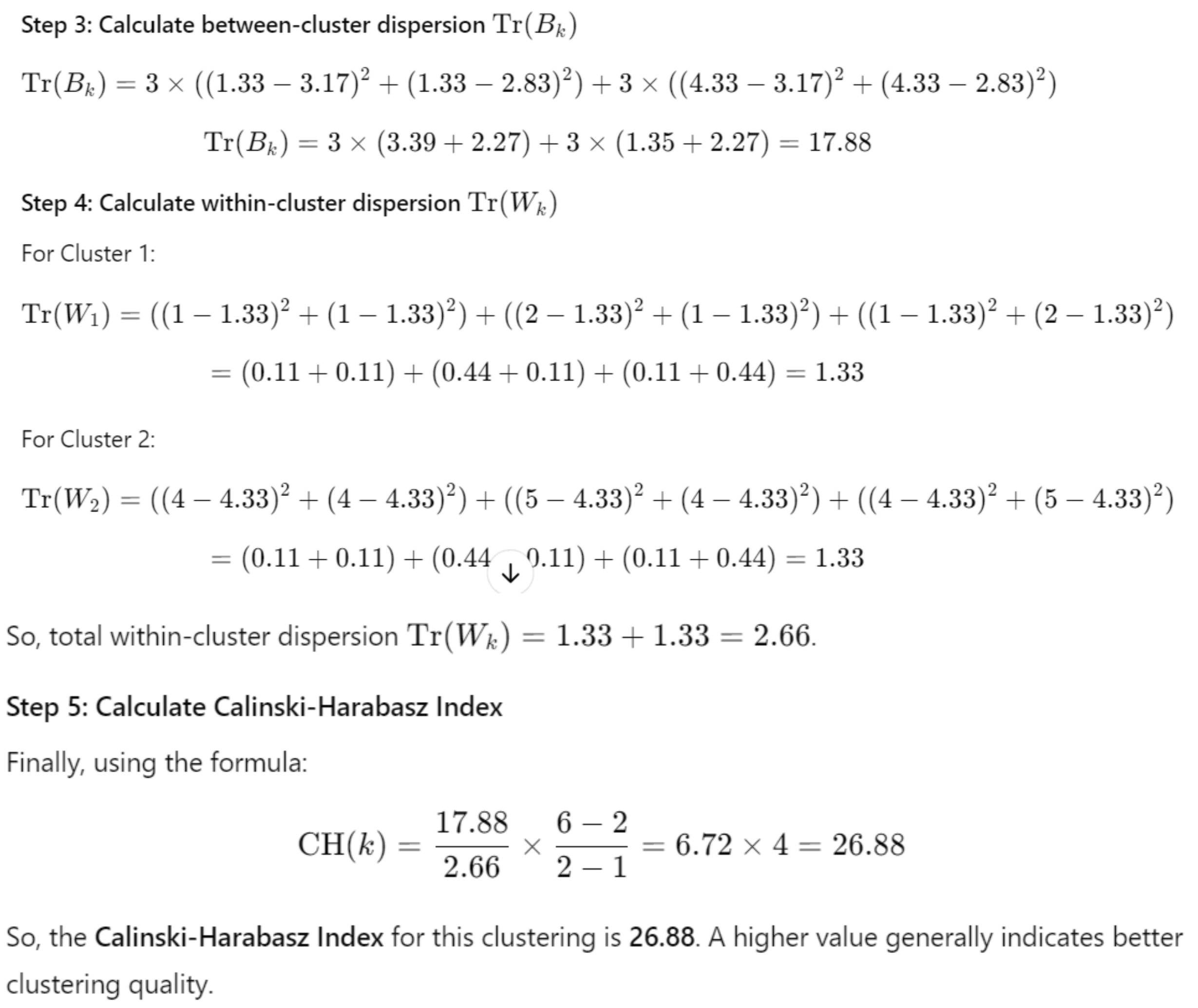
$$\mu_1 = \left(\frac{1+2+1}{3}, \frac{1+1+2}{3}\right) = (1.33, 1.33)$$

- Centroid of Cluster 2 $\mu_2$:

$$\mu_2 = \left(\frac{4+5+4}{3}, \frac{4+4+5}{3}\right) = (4.33, 4.33)$$

- Global centroid $\mu$ (mean of all points):

$$\mu = \left(\frac{1+2+1+4+5+4}{6}, \frac{1+1+2+4+4+5}{6}\right) = (3.17, 2.83)$$



**Step 3: Calculate between-cluster dispersion $\mathrm{Tr}(B_k)$**

$$\mathrm{Tr}(B_k) = 3 \times ((1.33 - 3.17)^2 + (1.33 - 2.83)^2) + 3 \times ((4.33 - 3.17)^2 + (4.33 - 2.83)^2)$$

$$\mathrm{Tr}(B_k) = 3 \times (3.39 + 2.27) + 3 \times (1.35 + 2.27) = 17.88$$

**Step 4: Calculate within-cluster dispersion $\mathrm{Tr}(W_k)$**

For Cluster 1:

$$\mathrm{Tr}(W_1) = ((1 - 1.33)^2 + (1 - 1.33)^2) + ((2 - 1.33)^2 + (1 - 1.33)^2) + ((1 - 1.33)^2 + (2 - 1.33)^2)$$

$$= (0.11 + 0.11) + (0.44 + 0.11) + (0.11 + 0.44) = 1.33$$

For Cluster 2:

$$\mathrm{Tr}(W_2) = ((4 - 4.33)^2 + (4 - 4.33)^2) + ((5 - 4.33)^2 + (4 - 4.33)^2) + ((4 - 4.33)^2 + (5 - 4.33)^2)$$

$$= (0.11 + 0.11) + (0.44 + 0.11) + (0.11 + 0.44) = 1.33$$

So, total within-cluster dispersion $\mathrm{Tr}(W_k) = 1.33 + 1.33 = 2.66$.

**Step 5: Calculate Calinski-Harabasz Index**

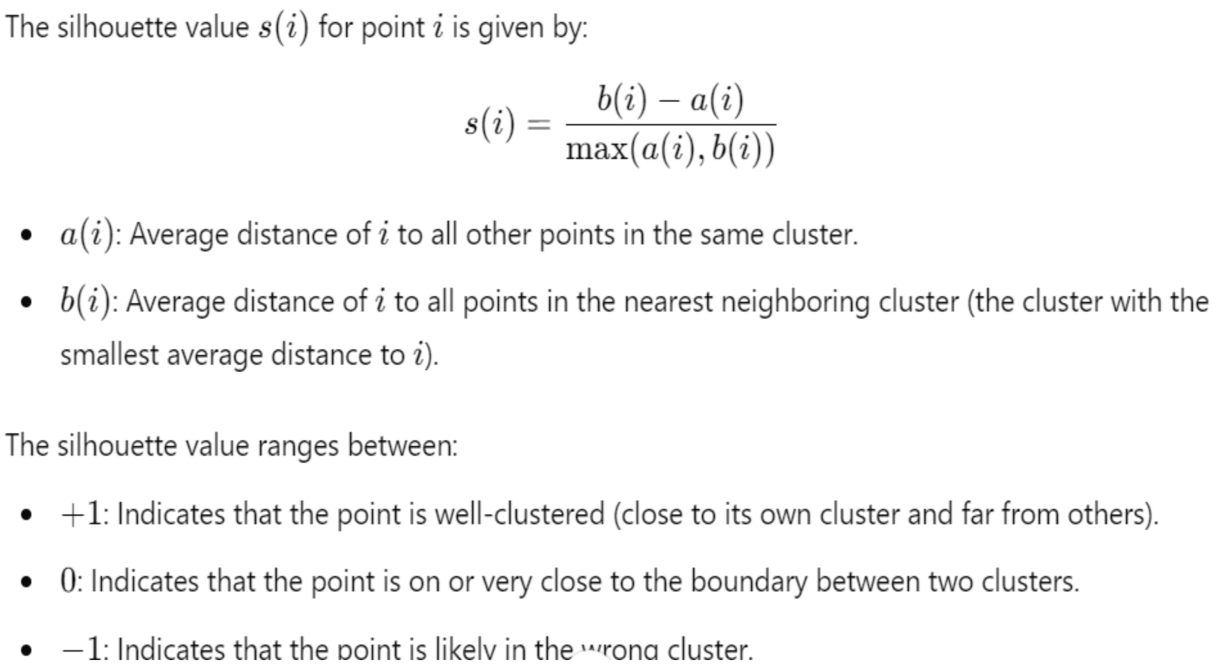
Finally, using the formula:

$$\mathrm{CH}(k) = \frac{17.88}{2.66} \times \frac{6 - 2}{2 - 1} = 6.72 \times 4 = 26.88$$

So, the **Calinski-Harabasz Index** for this clustering is **26.88**. A higher value generally indicates better clustering quality.

## Average Silhouette Method

The **average silhouette** is a method used to evaluate the quality of clustering results. It helps determine how well data points are clustered and provides a way to assess the optimal number of clusters (commonly used with **k-means** or hierarchical clustering).



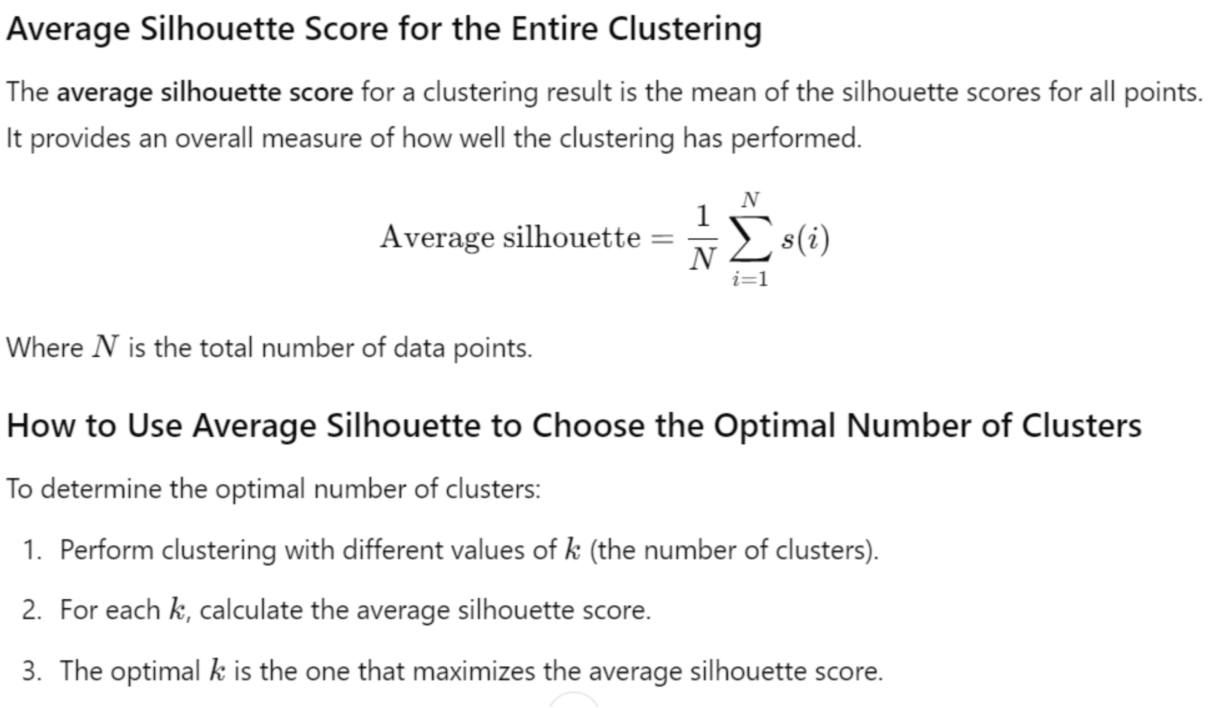
The silhouette value $s(i)$ for point $i$ is given by:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

- $a(i)$: Average distance of $i$ to all other points in the same cluster.
- $b(i)$: Average distance of $i$ to all points in the nearest neighboring cluster (the cluster with the smallest average distance to $i$).

The silhouette value ranges between:

- $+1$: Indicates that the point is well-clustered (close to its own cluster and far from others).
- $0$: Indicates that the point is on or very close to the boundary between two clusters.
- $-1$: Indicates that the point is likely in the wrong cluster.



### Average Silhouette Score for the Entire Clustering

The **average silhouette score** for a clustering result is the mean of the silhouette scores for all points. It provides an overall measure of how well the clustering has performed.

$$\text{Average silhouette} = \frac{1}{N} \sum_{i=1}^{N} s(i)$$

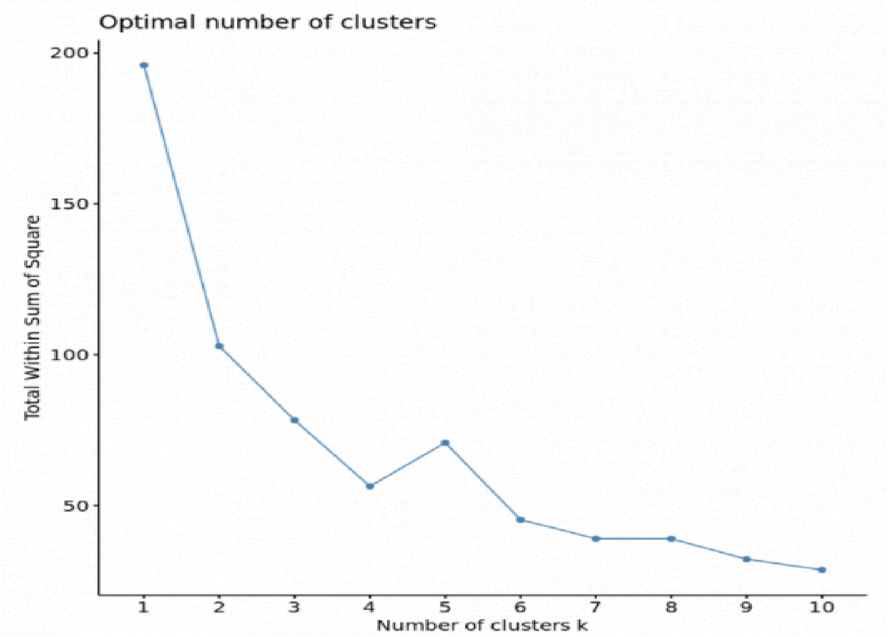
Where $N$ is the total number of data points.

### How to Use Average Silhouette to Choose the Optimal Number of Clusters

To determine the optimal number of clusters:

1. Perform clustering with different values of $k$ (the number of clusters).
2. For each $k$, calculate the average silhouette score.
3. The optimal $k$ is the one that maximizes the average silhouette score.

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

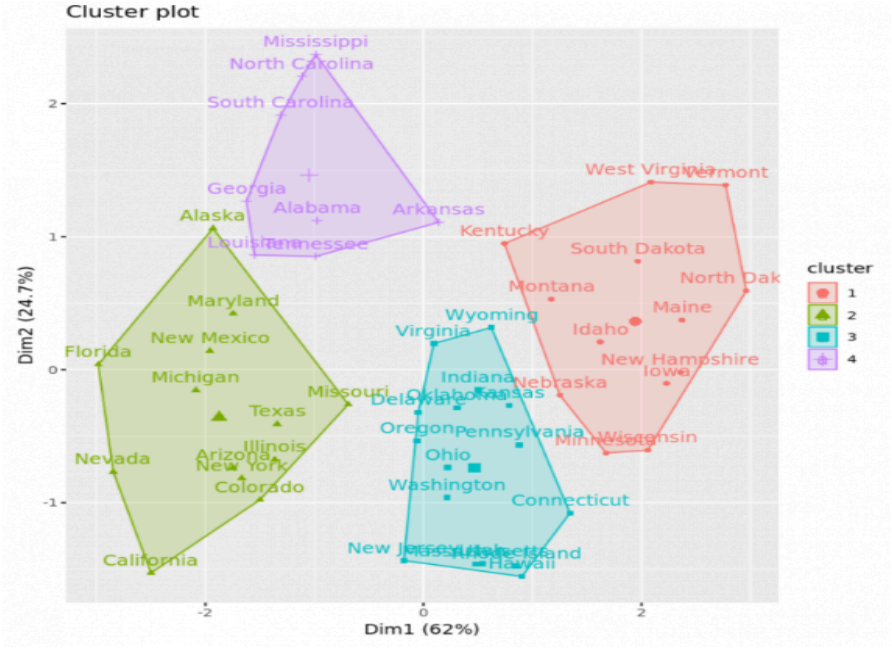


**Step 4: Perform K-Means Clustering with Optimal *K***

```
#make this example reproducible
set.seed(1)

#perform k-means clustering with k = 4 clusters
km <- kmeans(df, centers = 4, nstart = 25)

# View the size of each cluster
km$size
```

We can visualize the clusters on a scatterplot that displays the first two principal components on the axes using the **fivz_cluster()** function:

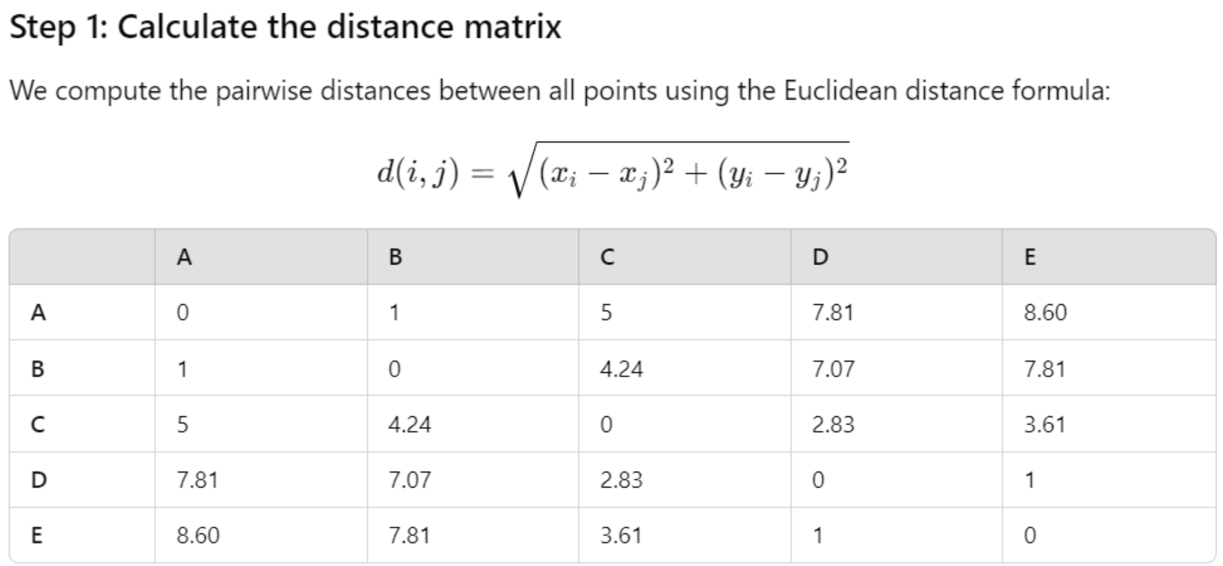
```
fviz_cluster(km, data = df)
```



## Hierarchical Clustering:

Hierarchical clustering is a method of clustering that builds a hierarchy of clusters either by starting with individual points and merging them (agglomerative) or starting with all points in one cluster and splitting them (divisive)

A: (2, 3) B: (3, 3)C: (6, 6)D: (8, 8)E: (8, 9)

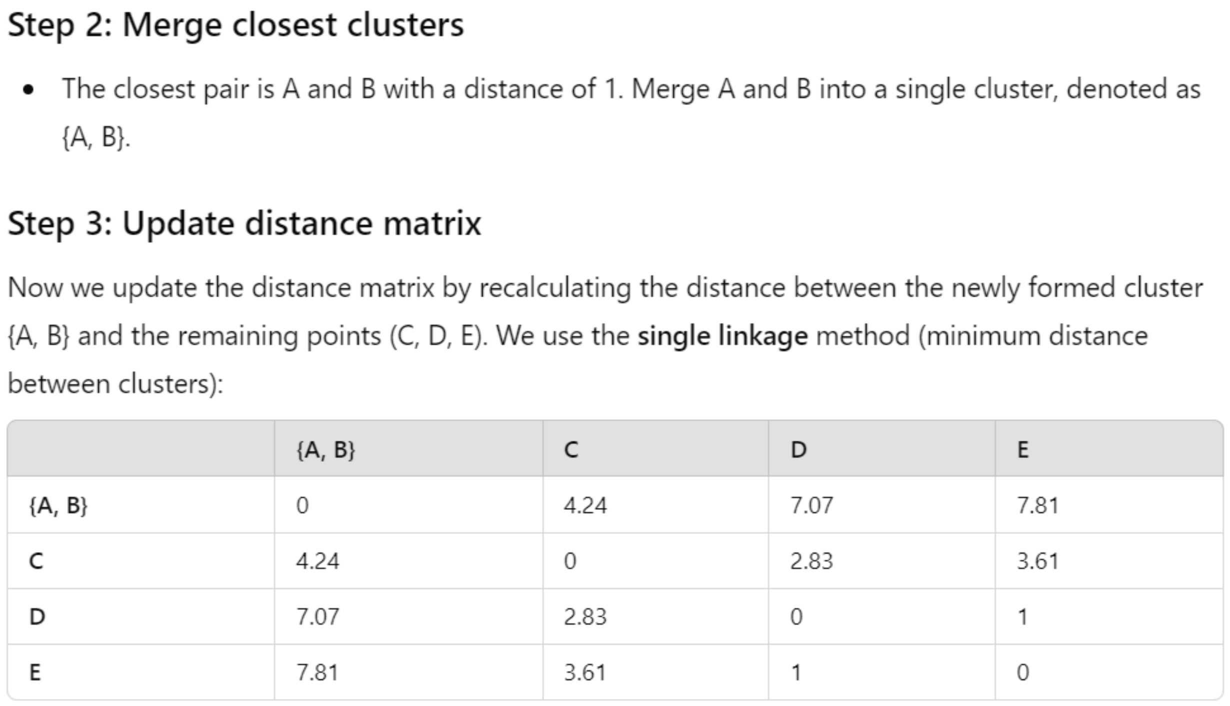


### Step 1: Calculate the distance matrix

We compute the pairwise distances between all points using the Euclidean distance formula:

$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 5 | 7.81 | 8.60 |
| B | 1 | 0 | 4.24 | 7.07 | 7.81 |
| C | 5 | 4.24 | 0 | 2.83 | 3.61 |
| D | 7.81 | 7.07 | 2.83 | 0 | 1 |
| E | 8.60 | 7.81 | 3.61 | 1 | 0 |



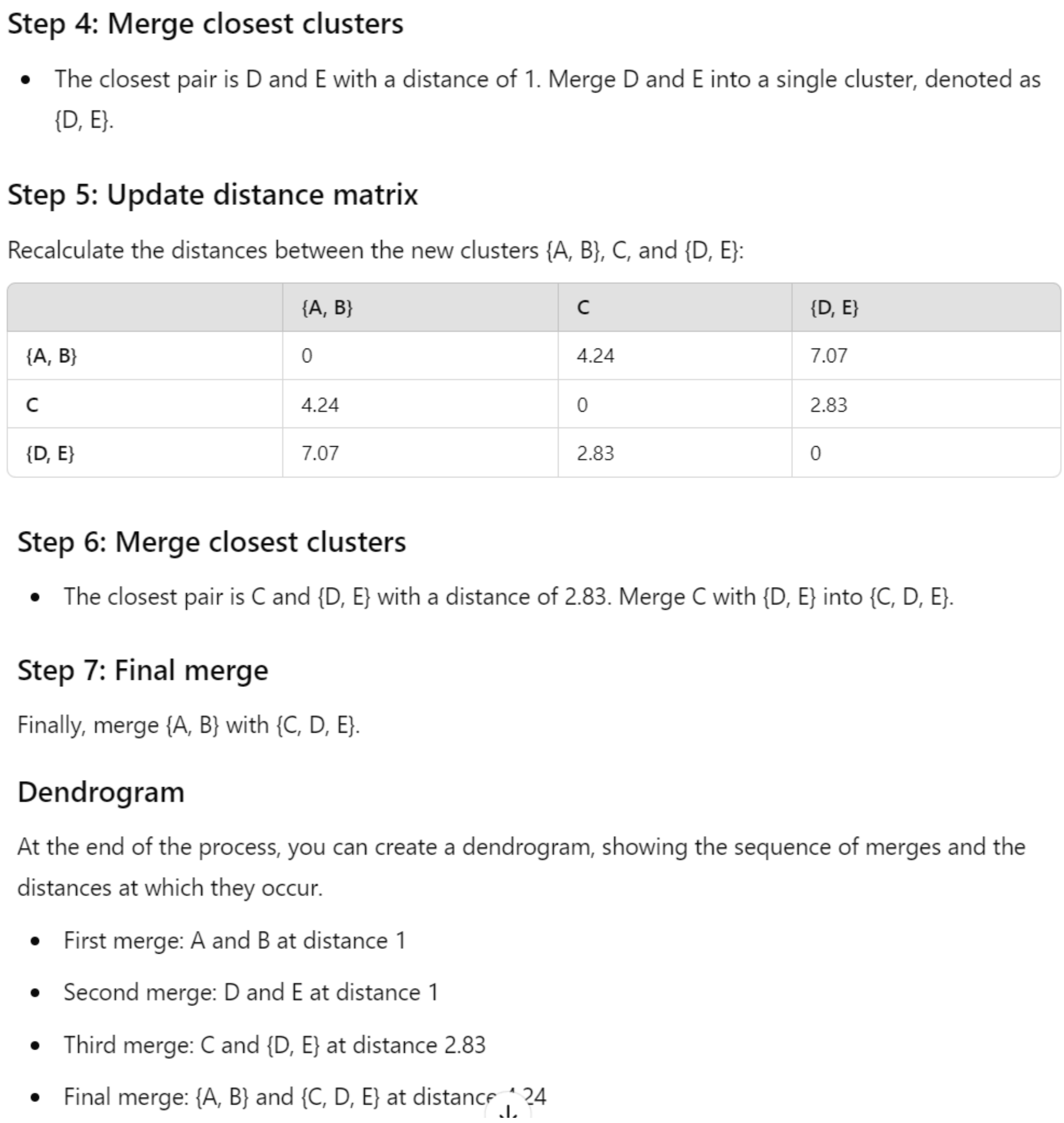
### Step 2: Merge closest clusters

- The closest pair is A and B with a distance of 1. Merge A and B into a single cluster, denoted as {A, B}.

### Step 3: Update distance matrix

Now we update the distance matrix by recalculating the distance between the newly formed cluster {A, B} and the remaining points (C, D, E). We use the **single linkage** method (minimum distance between clusters):

| | {A, B} | C | D | E |
|---|---|---|---|---|
| {A, B} | 0 | 4.24 | 7.07 | 7.81 |
| C | 4.24 | 0 | 2.83 | 3.61 |
| D | 7.07 | 2.83 | 0 | 1 |
| E | 7.81 | 3.61 | 1 | 0 |



### Step 4: Merge closest clusters

- The closest pair is D and E with a distance of 1. Merge D and E into a single cluster, denoted as {D, E}.

### Step 5: Update distance matrix

Recalculate the distances between the new clusters {A, B}, C, and {D, E}:

| | {A, B} | C | {D, E} |
|---|---|---|---|
| {A, B} | 0 | 4.24 | 7.07 |
| C | 4.24 | 0 | 2.83 |
| {D, E} | 7.07 | 2.83 | 0 |

### Step 6: Merge closest clusters

- The closest pair is C and {D, E} with a distance of 2.83. Merge C with {D, E} into {C, D, E}.

### Step 7: Final merge

Finally, merge {A, B} with {C, D, E}.

### Dendrogram

At the end of the process, you can create a dendrogram, showing the sequence of merges and the distances at which they occur.

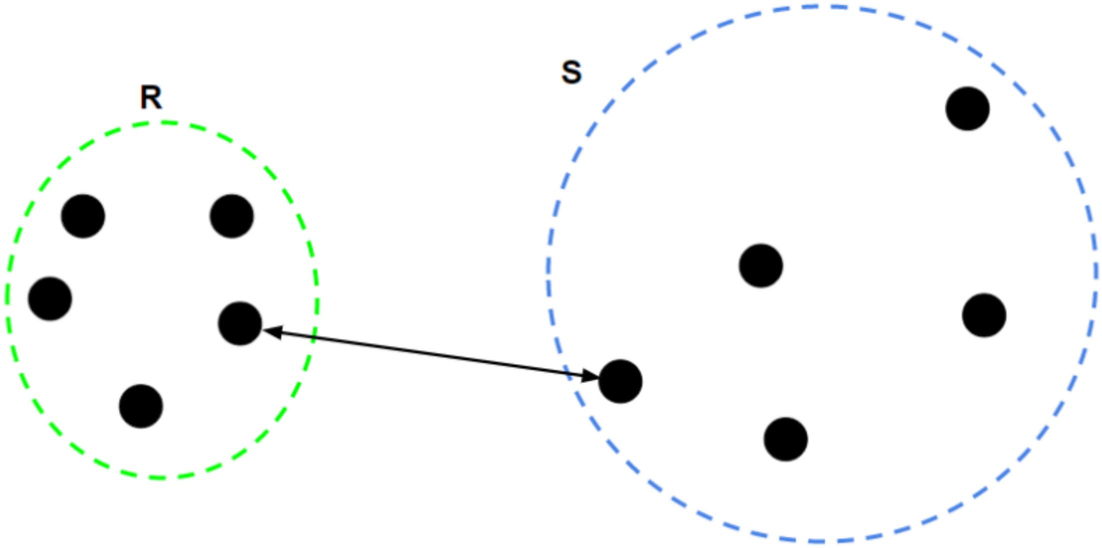
- First merge: A and B at distance 1
- Second merge: D and E at distance 1
- Third merge: C and {D, E} at distance 2.83
- Final merge: {A, B} and {C, D, E} at distance 4.24

## Methods to Merge Clusters:

### Single Linkage:

For two clusters R and S, the single linkage returns the minimum distance between two points i and j such that i belongs to R and j belongs to S.

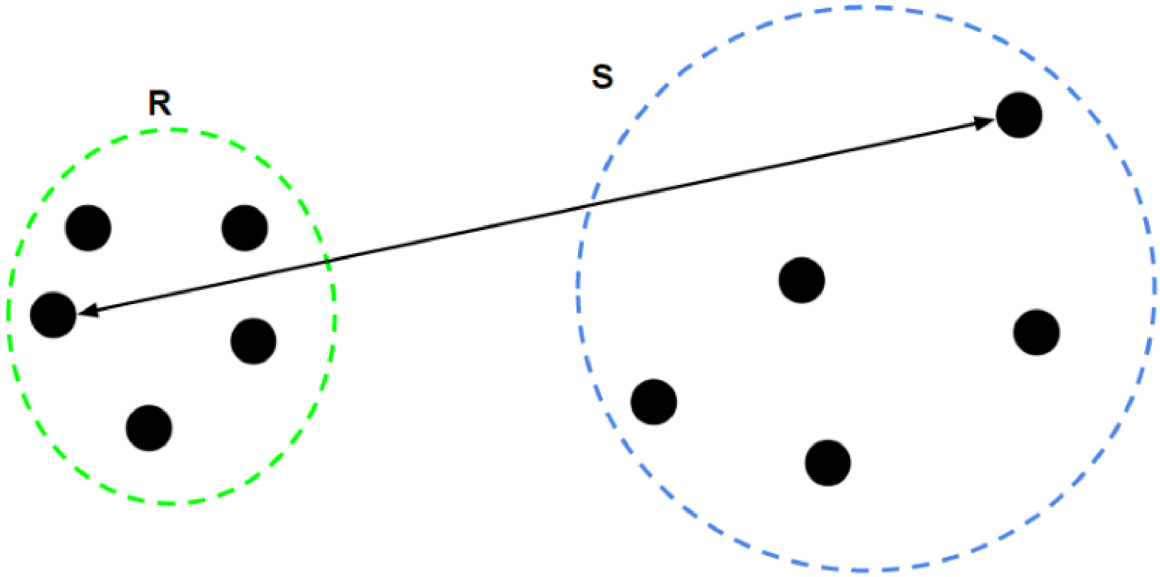
$$L(R,S) = min(D(i,j)), i\epsilon R, j\epsilon S$$



### Complete Linkage:

For two clusters R and S, the complete linkage returns the maximum distance between two points i and j such that i belongs to R and j belongs to S.

$$L(R,S) = max(D(i,j)), i\epsilon R, j\epsilon S$$



**Ward's Method:**

Ward's method minimizes the total within-cluster variance (also known as error sum of squares, or ESS). When merging two clusters, the method selects the pair of clusters whose merging results in the smallest increase in the total within-cluster variance.

**Steps:**

- Calculate the initial distance between every pair of points.
- For each merge, select the pair of clusters whose union leads to the smallest increase in the total variance.
- Recalculate distances after every merge.

## Example Data Points:

Let's consider the following four data points in 1-dimensional space:

- $A = 1$
- $B = 3$
- $C = 5$
- $D = 7$

Initially, each point is its own cluster: $\{A\}, \{B\}, \{C\}, \{D\}$.

### Step 1: Calculate the Initial Distances

We calculate the Euclidean distance (simple subtraction in this case):

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 2 | 4 | 6 |
| B | 2 | 0 | 2 | 4 |
| C | 4 | 2 | 0 | 2 |
| D | 6 | 4 | 2 | 0 |

### Step 2: Initial Merge (Find Clusters to Merge)

Ward's method minimizes the total within-cluster variance (or ESS).

**Formula for ESS**:

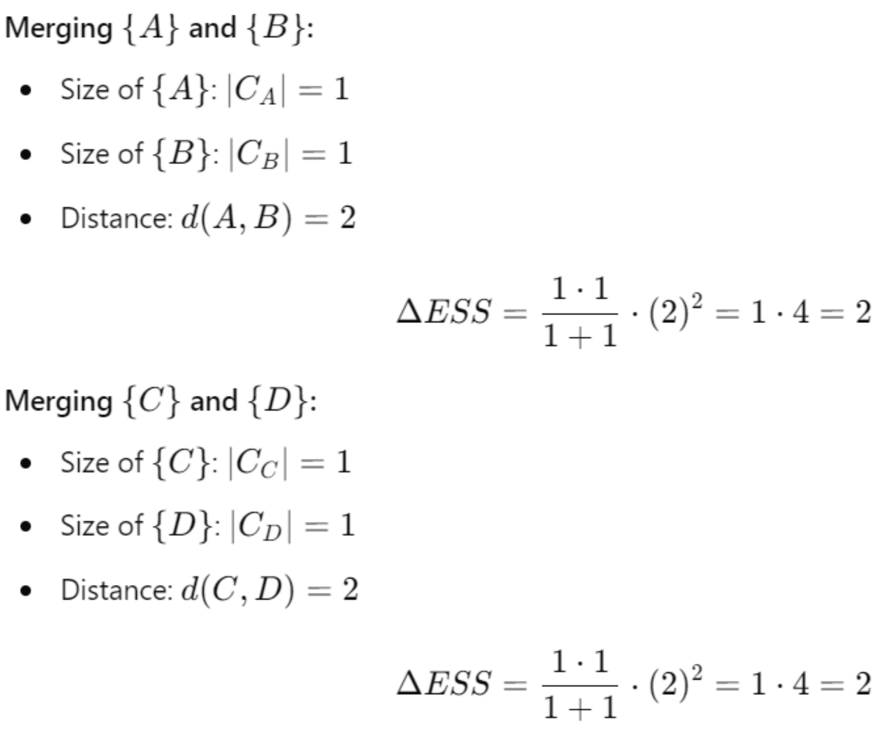
- When two clusters $C_i$ and $C_j$ are merged, the increase in variance is calculated as:

$$\Delta ESS = \frac{|C_i| \cdot |C_j|}{|C_i| + |C_j|} \cdot d(C_i, C_j)^2$$

Where:

- $|C_i|$ and $|C_j|$ are the sizes (number of points) in clusters $C_i$ and $C_j$.
- $d(C_i, C_j)$ is the distance between the centroids of clusters $C_i$ and $C_j$.

Let's now calculate the $\Delta ESS$ for each possible pair of clusters.



Merging $\{A\}$ and $\{B\}$:

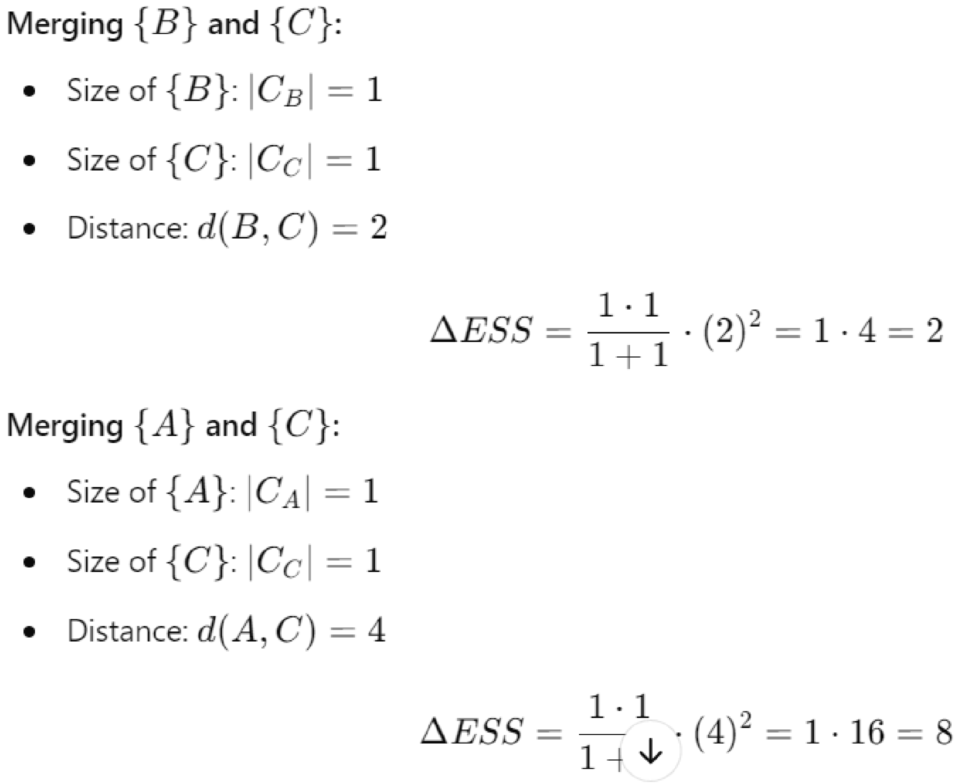
- Size of $\{A\}$: $|C_A| = 1$
- Size of $\{B\}$: $|C_B| = 1$
- Distance: $d(A, B) = 2$

$$\Delta ESS = \frac{1 \cdot 1}{1 + 1} \cdot (2)^2 = 1 \cdot 4 = 2$$

Merging $\{C\}$ and $\{D\}$:

- Size of $\{C\}$: $|C_C| = 1$
- Size of $\{D\}$: $|C_D| = 1$
- Distance: $d(C, D) = 2$

$$\Delta ESS = \frac{1 \cdot 1}{1 + 1} \cdot (2)^2 = 1 \cdot 4 = 2$$



Merging $\{B\}$ and $\{C\}$:

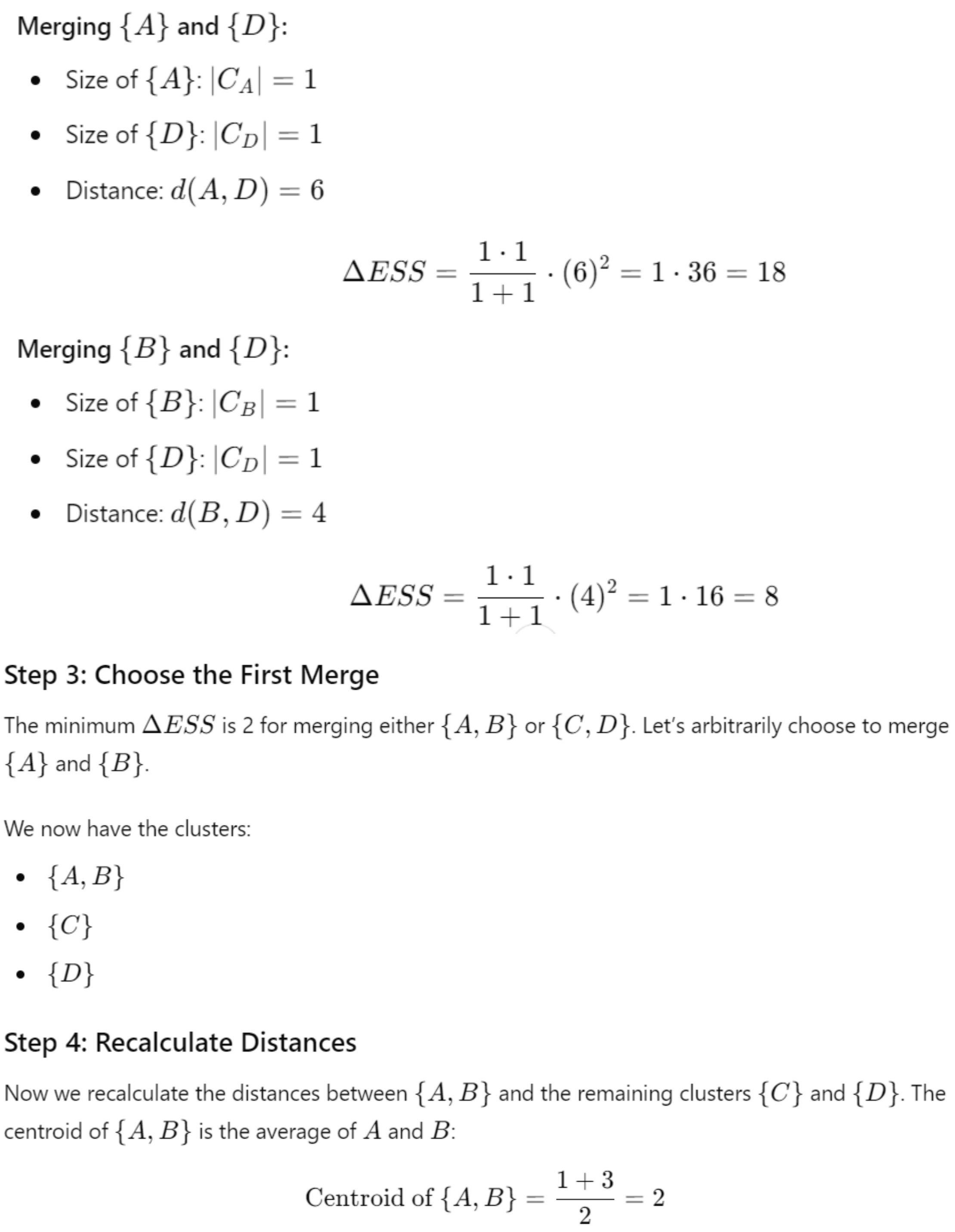
- Size of $\{B\}$: $|C_B| = 1$
- Size of $\{C\}$: $|C_C| = 1$
- Distance: $d(B, C) = 2$

$$\Delta ESS = \frac{1 \cdot 1}{1 + 1} \cdot (2)^2 = 1 \cdot 4 = 2$$

Merging $\{A\}$ and $\{C\}$:

- Size of $\{A\}$: $|C_A| = 1$
- Size of $\{C\}$: $|C_C| = 1$
- Distance: $d(A, C) = 4$

$$\Delta ESS = \frac{1 \cdot 1}{1 +} \cdot (4)^2 = 1 \cdot 16 = 8$$



**Merging $\{A\}$ and $\{D\}$:**

- Size of $\{A\}$: $|C_A| = 1$
- Size of $\{D\}$: $|C_D| = 1$
- Distance: $d(A, D) = 6$

$$\Delta ESS = \frac{1 \cdot 1}{1+1} \cdot (6)^2 = 1 \cdot 36 = 18$$

**Merging $\{B\}$ and $\{D\}$:**

- Size of $\{B\}$: $|C_B| = 1$
- Size of $\{D\}$: $|C_D| = 1$
- Distance: $d(B, D) = 4$

$$\Delta ESS = \frac{1 \cdot 1}{1+1} \cdot (4)^2 = 1 \cdot 16 = 8$$

### Step 3: Choose the First Merge

The minimum $\Delta ESS$ is 2 for merging either $\{A, B\}$ or $\{C, D\}$. Let's arbitrarily choose to merge $\{A\}$ and $\{B\}$.

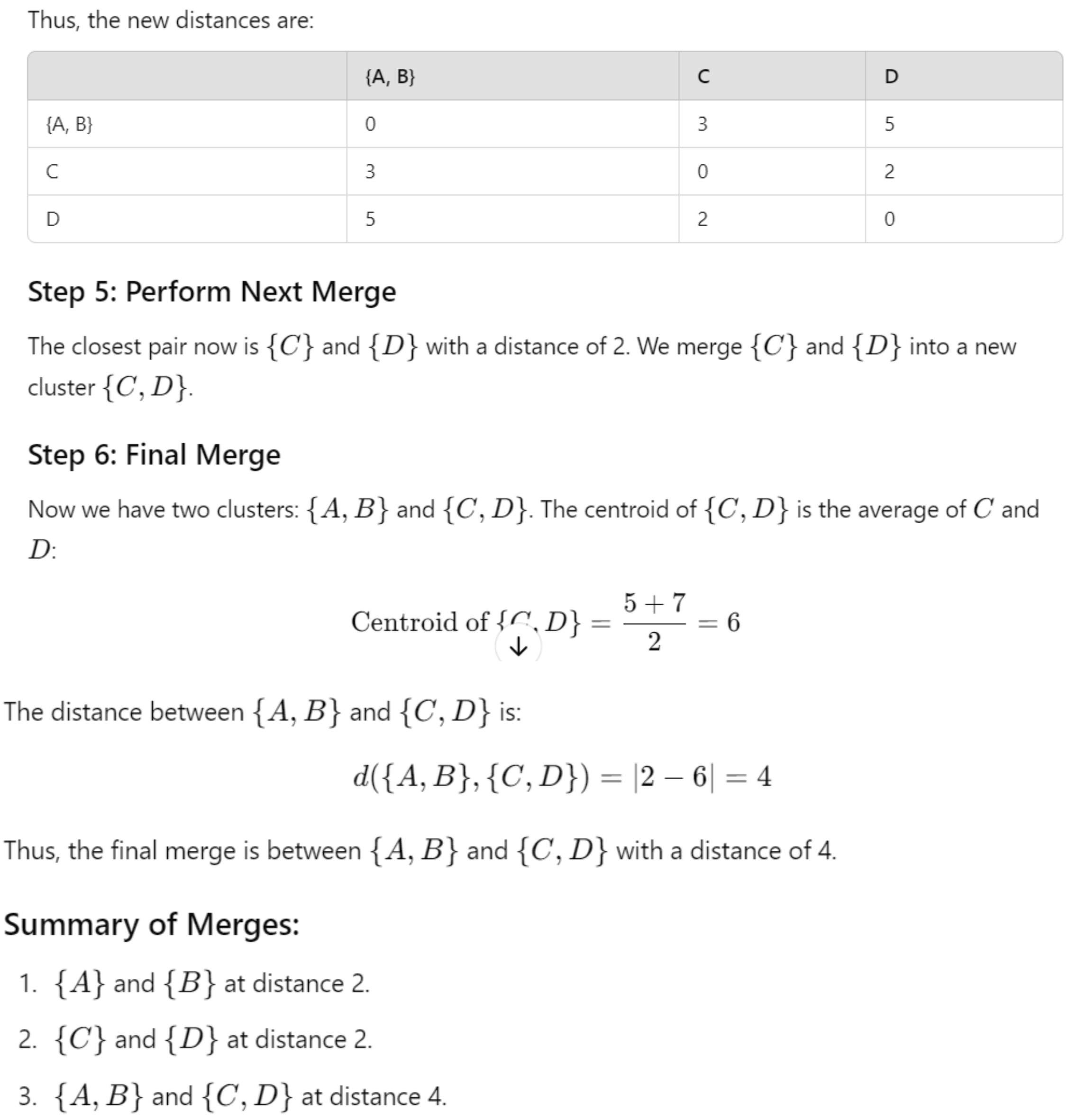
We now have the clusters:

- $\{A, B\}$
- $\{C\}$
- $\{D\}$

### Step 4: Recalculate Distances

Now we recalculate the distances between $\{A, B\}$ and the remaining clusters $\{C\}$ and $\{D\}$. The centroid of $\{A, B\}$ is the average of $A$ and $B$:

$$\text{Centroid of } \{A, B\} = \frac{1+3}{2} = 2$$



Thus, the new distances are:

| | {A, B} | C | D |
|---|---|---|---|
| {A, B} | 0 | 3 | 5 |
| C | 3 | 0 | 2 |
| D | 5 | 2 | 0 |

### Step 5: Perform Next Merge

The closest pair now is $\{C\}$ and $\{D\}$ with a distance of 2. We merge $\{C\}$ and $\{D\}$ into a new cluster $\{C, D\}$.

### Step 6: Final Merge

Now we have two clusters: $\{A, B\}$ and $\{C, D\}$. The centroid of $\{C, D\}$ is the average of $C$ and $D$:

$$\text{Centroid of } \{C, D\} = \frac{5+7}{2} = 6$$

The distance between $\{A, B\}$ and $\{C, D\}$ is:

$$d(\{A, B\}, \{C, D\}) = |2 - 6| = 4$$

Thus, the final merge is between $\{A, B\}$ and $\{C, D\}$ with a distance of 4.

## Summary of Merges:

1. $\{A\}$ and $\{B\}$ at distance 2.
2. $\{C\}$ and $\{D\}$ at distance 2.
3. $\{A, B\}$ and $\{C, D\}$ at distance 4.

## Bootstrap Evaluation

**Purpose of Evaluation**:

- Assess whether a cluster genuinely represents a structure in the data or if it's merely an artifact of the clustering algorithm.
- Particularly relevant for clustering algorithms like k-means, where the number of clusters must be specified beforehand.

**Cluster Characteristics**:

- Clusters often reveal actual relationships in the data.
- Clusters of "other" or "miscellaneous" are composed of data points with no real relationship, merely fitting into an arbitrary category.

**Assessment Method**:

- The **fpc** package provides the `clusterboot()` function for evaluating cluster stability using bootstrap resampling.
- This function integrates clustering and evaluation for various algorithms, including `hclust` and `kmeans`.

**Jaccard Coefficient**:

- A similarity measure between sets defined as:

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

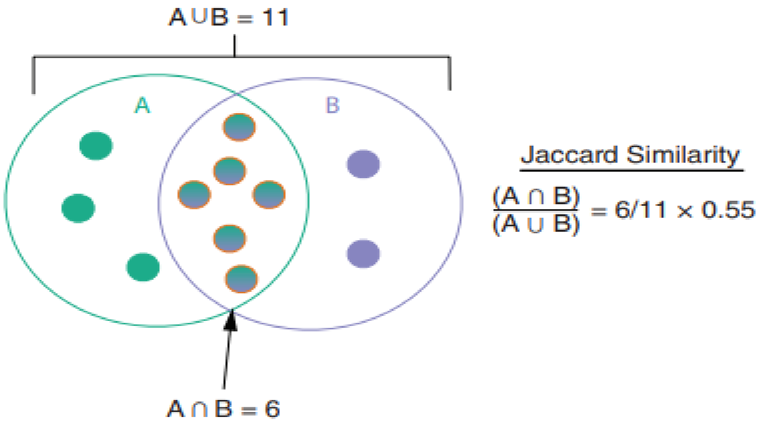


Figure 9.9 Jaccard similarity

It quantifies the similarity of two clusters by comparing the intersection and union of their members.

**Evaluation Strategy**:

- **Step 1**: Perform the initial clustering on the original dataset.
- **Step 2**: Create a new dataset through bootstrap resampling (sampling with replacement) and cluster this new dataset.
- **Step 3**: For each original cluster, find the most similar cluster in the new clustering using the maximum Jaccard coefficient.
    - If this coefficient is less than 0.5, the original cluster is considered "dissolved," indicating instability.
- **Step 4**: Repeat steps 2 and 3 multiple times to obtain a comprehensive assessment of cluster stability.

**Interpretation of Results**:

- Clusters that frequently dissolve (low Jaccard coefficients) are likely not representative of true structure in the data and should be treated with caution.
- High stability (high Jaccard coefficients) indicates that the cluster is more likely to reflect genuine patterns in the data

# Exploring Advance Methods

## Drawbacks of basic ML models:

1. **Training variance**— Training variance is when small changes in the makeup of the training set result in models that make substantially different predictions. Decision trees can exhibit this effect. Both **bagging and random forests** can reduce training variance and sensitivity to overfitting.
2. **Non-monotone effects** —-Occur when the relationship between a predictor variable and the outcome is not consistently increasing or decreasing. This means that increasing a variable may lead to a better outcome up to a certain point, after which it may have a negative effect. Traditional linear and logistic regression models assume a monotonic relationship, meaning they predict that as an independent variable increases, the dependent variable either consistently increases or consistently decreases.
3. **Linearly inseparable** data-Kernel methods allow the data scientist to introduce new nonlinear combination terms to models , and s**upport vector machines (SVMs) use both kernels** and training data to build useful decision surfaces.

## Using bagging and random forests to reduce training variance

Decision trees are an attractive method for a number of reasons:

1. They take any type of data, **numerical or categorical, without any distributional assumptions and without preprocessing.**
2. The algorithm is easy to use, and the output (the tree) is relatively easy to understand.

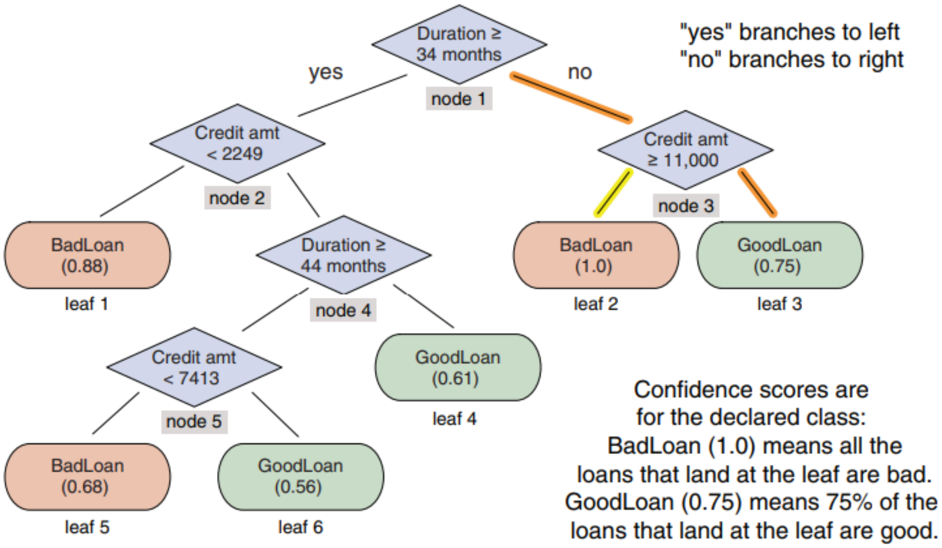


**Figure 10.3 Example decision tree (from chapter 1)**

3.Once the model is fit, scoring is fast

On the other hand, decision trees do have some drawbacks:

1. They have a tendency to overfit, especially without pruning.
2. They have high training variance: samples drawn from the same population can produce trees with different structures and different prediction accuracy.
3. Prediction accuracy can be low, compared to other methods.

## Using bagging to improve prediction

**Definition of Bagging**:

- Bagging is a technique to improve the performance of decision tree models by combining the predictions of multiple individual trees.
- It involves drawing random samples with replacement (bootstrap samples) from the original dataset to create multiple models.

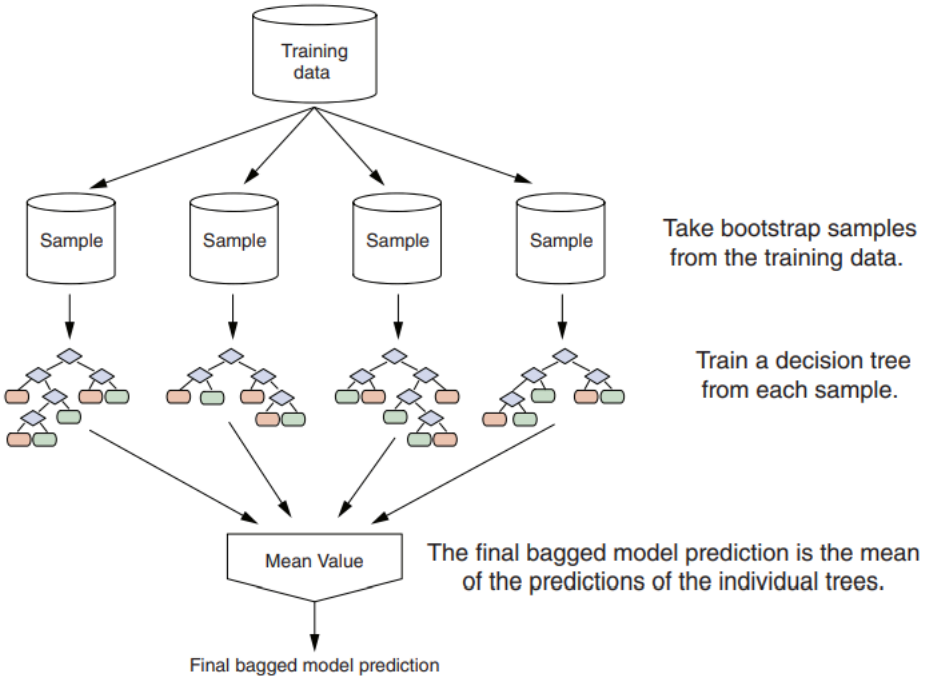

Training data
Sample
Sample
Sample
Sample
Take bootstrap samples from the training data.
Train a decision tree from each sample.
Mean Value
The final bagged model prediction is the mean of the predictions of the individual trees.
Final bagged model prediction

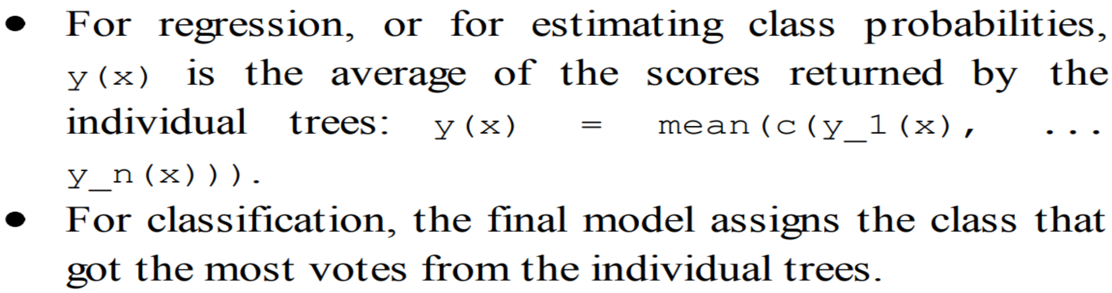
**Figure 10.5 Bagging decision trees**

**Individual Decision Trees**:

- For each bootstrap sample, a separate decision tree model is built.
- The trees are trained independently on different subsets of the data.

**Final Prediction**:

- The final output of the bagged model is an aggregate (average) of the outputs of the individual trees.



- For regression, or for estimating class probabilities, `y(x)` is the average of the scores returned by the individual trees: `y(x) = mean(c(y_1(x), ... y_n(x)))`.
- For classification, the final model assigns the class that got the most votes from the individual trees.

**Improvement of Model**:

- Bagging helps reduce variance and prevents overfitting, leading to more robust predictions.
- It mitigates the shortcomings of single decision trees, which are prone to high variance.

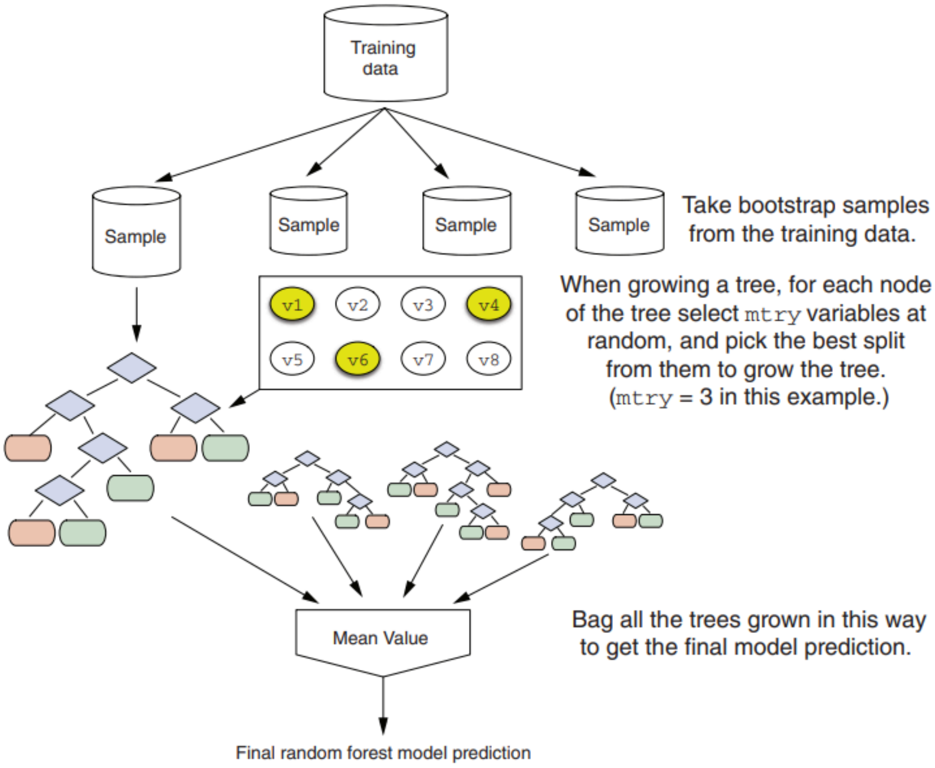
## Using random forests to further improve prediction

**Limitation of Bagging**:

- In bagging, each tree is built using the same set of features, which can lead to high correlation between trees.
- This correlation results in similar mistakes being made across all trees, reducing the effectiveness of bagging in correcting errors.

**Random Forest Approach**:

- Random forests **reduce correlation among trees** by introducing **additional randomness in feature selection.**
- It builds on the bagging technique **but adds an extra layer of randomization in choosing the variables (features) at each tree node.**



Training data
Sample
Sample
Sample
Sample
Take bootstrap samples from the training data.
v1
v2
v3
v4
v5
v6
v7
v8
When growing a tree, for each node of the tree select mtry variables at random, and pick the best split from them to grow the tree. (mtry = 3 in this example.)
Mean Value
Bag all the trees grown in this way to get the final model prediction.
Final random forest model prediction

**Figure 10.6 Growing a random forest**

**Process for Random Forest**:

1. **Bootstrap Sampling**: A bootstrapped sample is drawn from the training data for each tree.
2. **Random Feature Selection**: At each node in the decision tree:
   - A random subset of features (of size `mtry`) is selected from the total set of features.

    - The best feature and split are chosen from this subset, not from all available features.
3. **Tree Growth**: The decision tree is fully grown without pruning.

**Final Prediction**:

- The final ensemble of trees is aggregated (usually by averaging or majority voting) to make the random forest prediction.

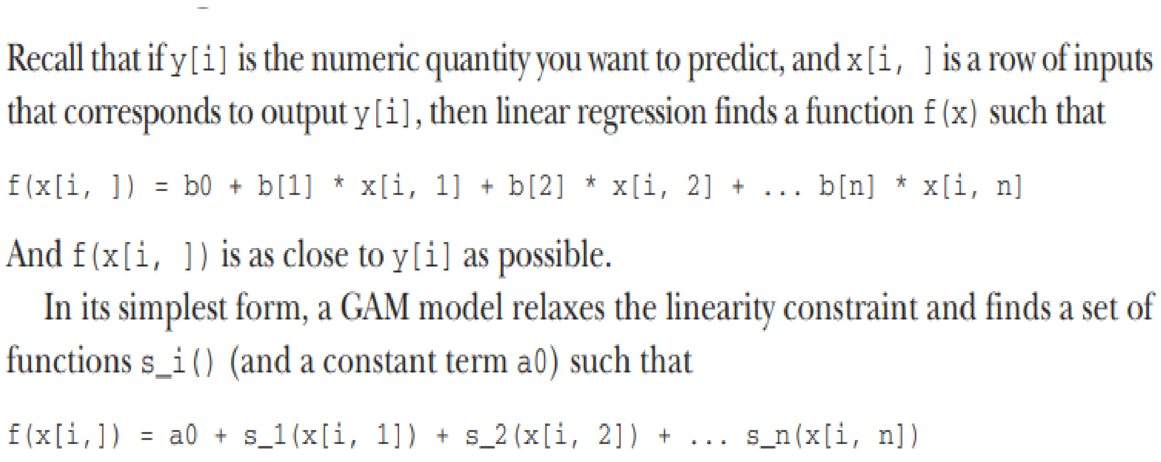
**Outcome**:

- Random forests create more diverse and de-correlated trees, leading to a more robust and accurate prediction model than simple bagging.

## Using generalized additive models (GAMs) to learn non-monotone relationships

Generalized additive models (GAMs) provide a more flexible approach by allowing for non-linear relationships.

**Understanding GAMs**



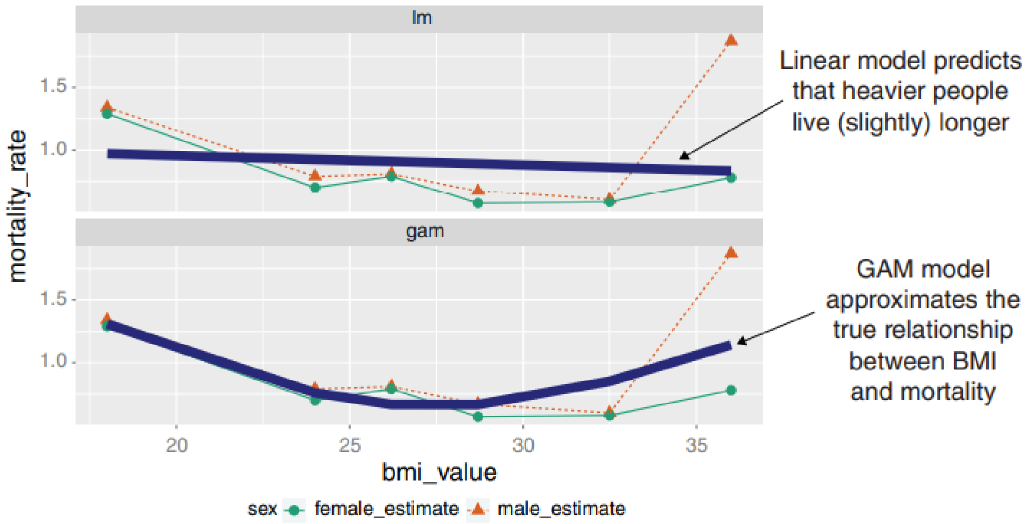
Recall that if y[i] is the numeric quantity you want to predict, and x[i, ] is a row of inputs that corresponds to output y[i], then linear regression finds a function f(x) such that

```
f(x[i, ]) = b0 + b[1] * x[i, 1] + b[2] * x[i, 2] + ... b[n] * x[i, n]
```

And f(x[i, ]) is as close to y[i] as possible.

In its simplest form, a GAM model relaxes the linearity constraint and finds a set of functions s_i() (and a constant term a0) such that

```
f(x[i,]) = a0 + s_1(x[i, 1]) + s_2(x[i, 2]) + ... s_n(x[i, n])
```



**The functions s_i() are smooth curve fits that are built up from polynomials. The curves are called splines and are designed to pass as closely as possible through the data without being too “wiggly” (without overfitting)**

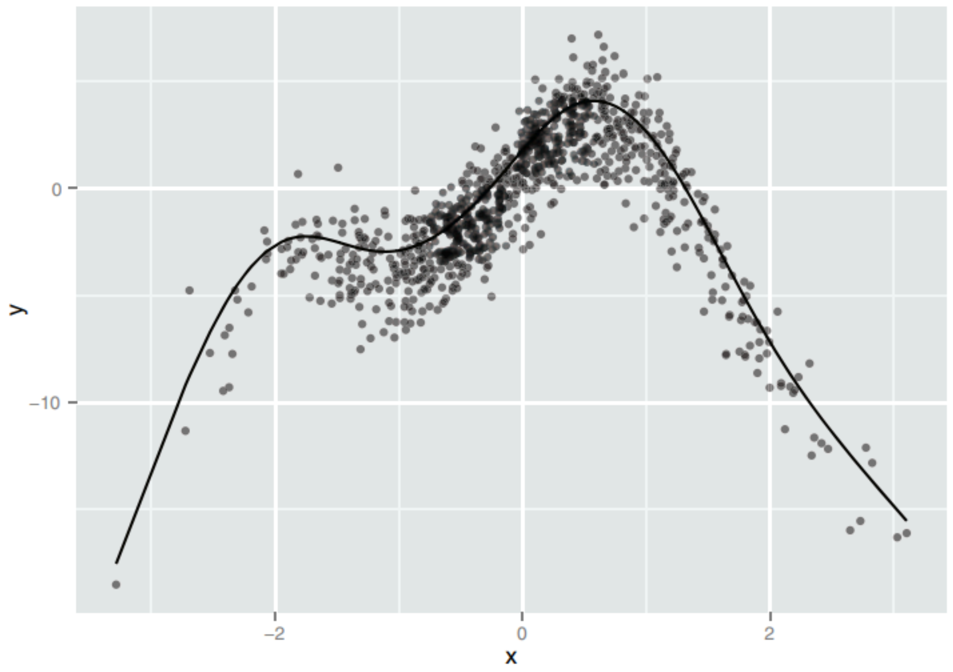
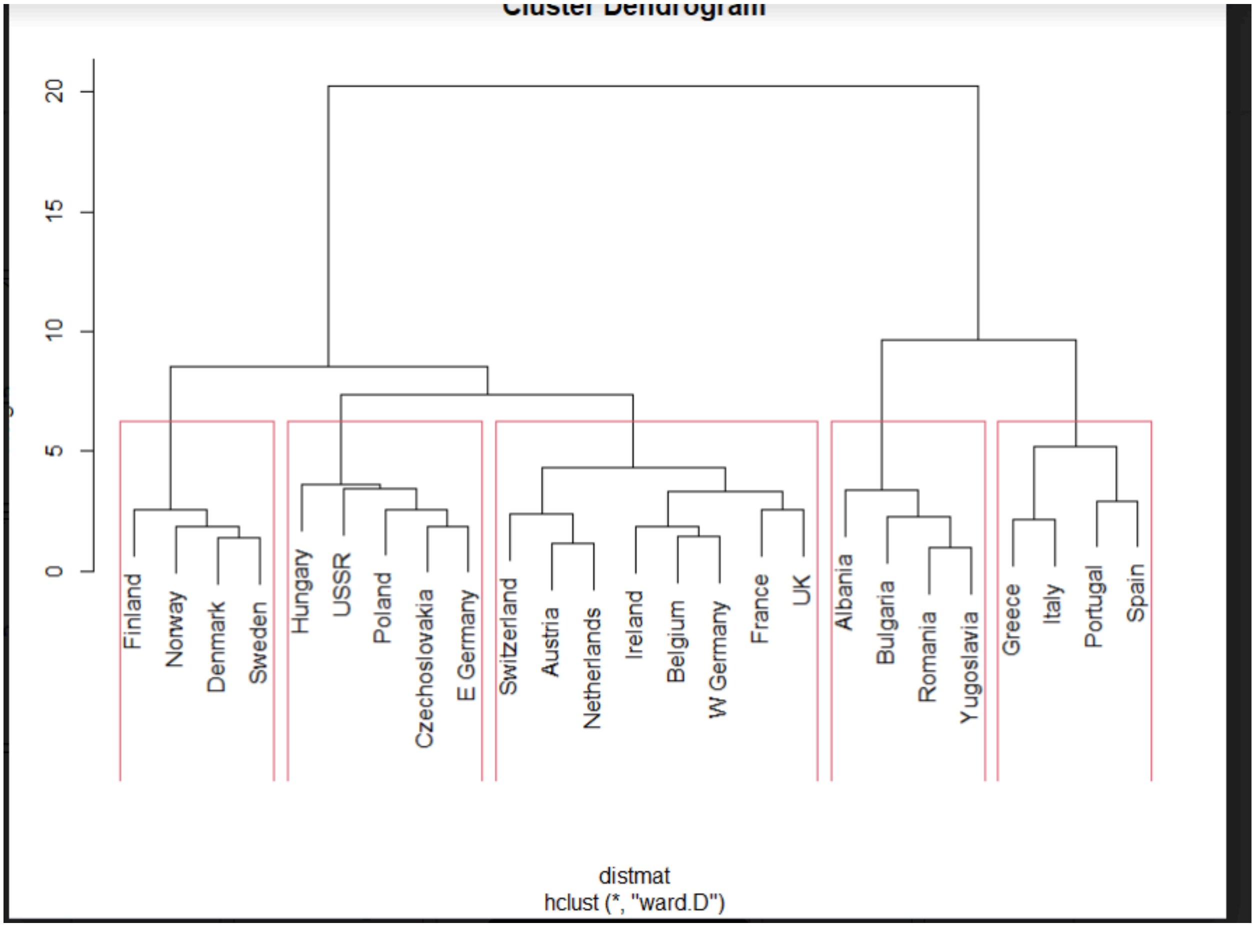


Figure 10.13 A spline that has been fit through a series of points

**Implementation of Heirarchial Clustering:**

```
protein <- read.table(file.choose(), sep = "\t", header=TRUE)

vars_to_use <- colnames(protein)[-1]

pmatrix <- scale(protein[, vars_to_use])

distmat <- dist(pmatrix, method = "euclidean")

pfit <- hclust(distmat, method = "ward.D")

plot(pfit, labels = protein$Country)

rect.hclust(pfit, k=5)
```



# To extract the members of each cluster from the hclust object, use cutree().

groups <- cutree(pfit, k = 5)

print_clusters <- function(data, groups, columns) {

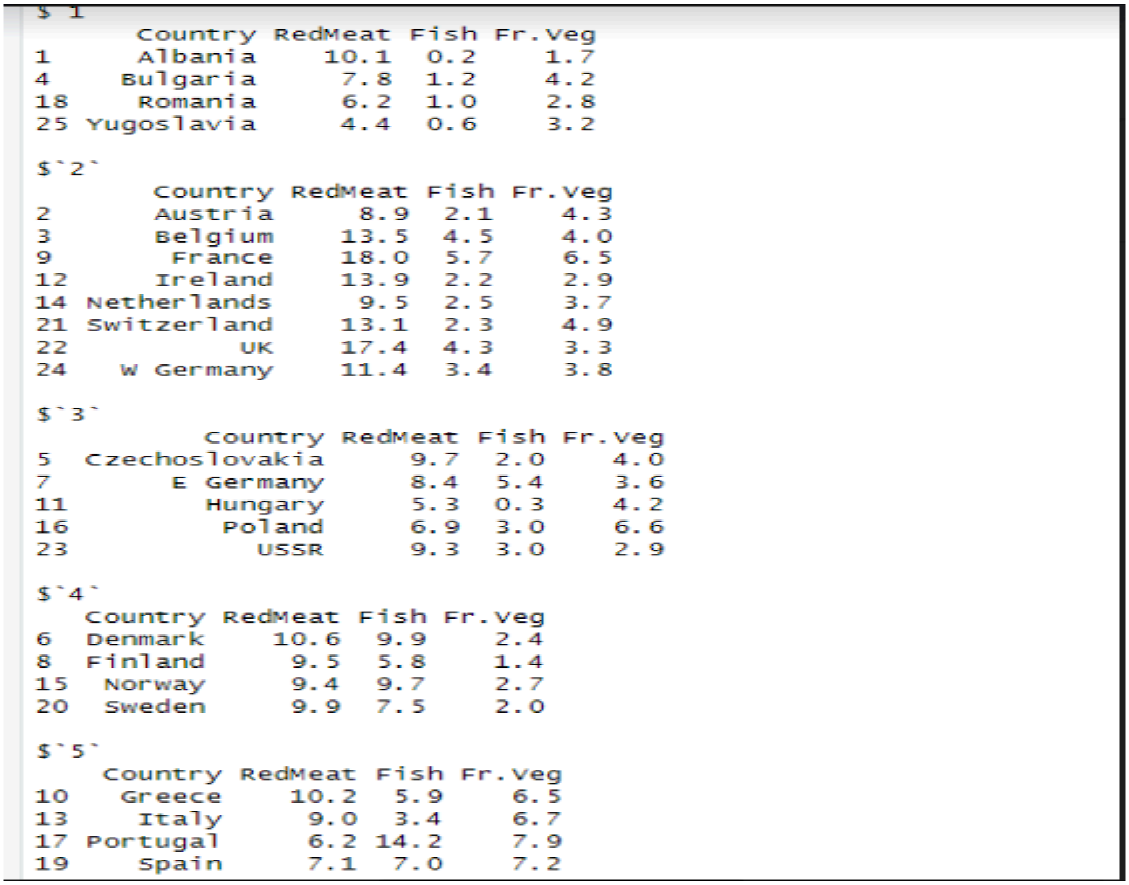
  groupedD <- split(data, groups)

  lapply(groupedD, function(df) df[, columns])

}

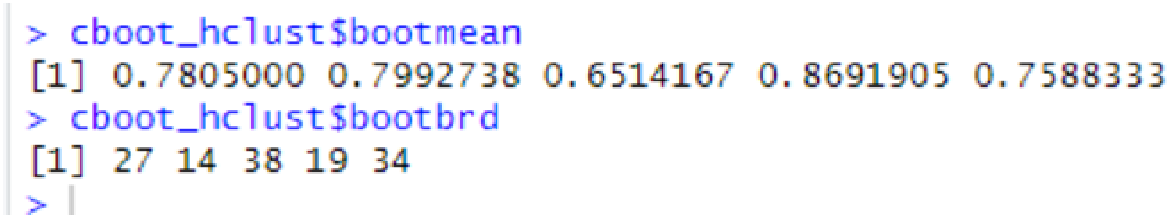
cols_to_print <- wrapr::qc(Country, RedMeat, Fish, Fr.Veg)

print_clusters(protein, groups, cols_to_print)



```
$`1`
      Country RedMeat Fish Fr.Veg
1     Albania    10.1  0.2    1.7
4    Bulgaria     7.8  1.2    4.2
18    Romania     6.2  1.0    2.8
25 Yugoslavia     4.4  0.6    3.2

$`2`
       Country RedMeat Fish Fr.Veg
2      Austria     8.9  2.1    4.3
3      Belgium    13.5  4.5    4.0
9       France    18.0  5.7    6.5
12     Ireland    13.9  2.2    2.9
14 Netherlands     9.5  2.5    3.7
21 Switzerland    13.1  2.3    4.9
22          UK    17.4  4.3    3.3
24   W Germany    11.4  3.4    3.8

$`3`
          Country RedMeat Fish Fr.Veg
5  Czechoslovakia     9.7  2.0    4.0
7       E Germany     8.4  5.4    3.6
11        Hungary     5.3  0.3    4.2
16         Poland     6.9  3.0    6.6
23           USSR     9.3  3.0    2.9

$`4`
   Country RedMeat Fish Fr.Veg
6  Denmark    10.6  9.9    2.4
8  Finland     9.5  5.8    1.4
15  Norway     9.4  9.7    2.7
20  Sweden     9.9  7.5    2.0

$`5`
    Country RedMeat Fish Fr.Veg
10   Greece    10.2  5.9    6.5
13    Italy     9.0  3.4    6.7
17 Portugal     6.2 14.2    7.9
19    Spain     7.1  7.0    7.2
```

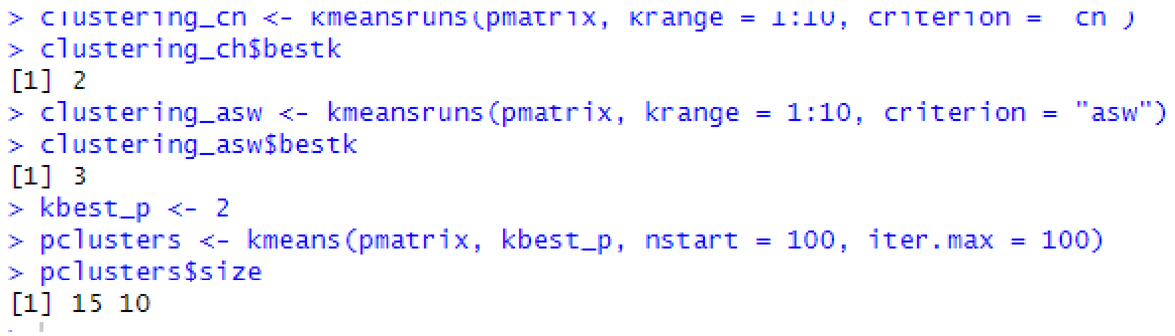
```
# BOOTSTRAP EVALUATION OF CLUSTERS

install.packages("fpc")

library(fpc)

kbest_p <- 5

clustermethod <- function(x) hclust(dist(x), method = "ward.D")

cboot_hclust <- clusterboot(pmatrix,

                     clustermethod = hclustCBI,

                     method = "ward.D",

                     k = kbest_p)

groups <- cboot_hclust$result$partition

print_clusters(protein, groups, cols_to_print)

cboot_hclust$bootmean

cboot_hclust$bootbrd
```



```
> cboot_hclust$bootmean
[1] 0.7805000 0.7992738 0.6514167 0.8691905 0.7588333
> cboot_hclust$bootbrd
[1] 27 14 38 19 34
> 
```

## Implementation of K Means

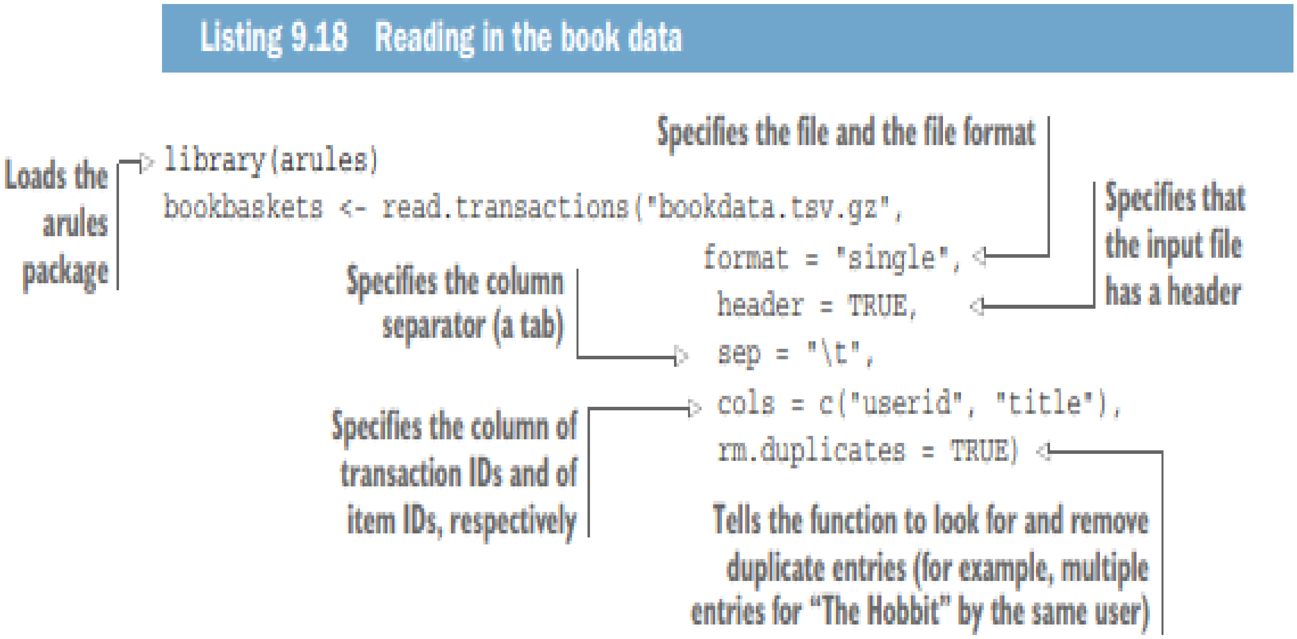
```
clustering_ch <- kmeansruns(pmatrix, krange = 1:10, criterion = "ch")

clustering_ch$bestk

clustering_asw <- kmeansruns(pmatrix, krange = 1:10, criterion = "asw")

clustering_asw$bestk

kbest_p <- 2

pclusters <- kmeans(pmatrix, kbest_p, nstart = 100, iter.max = 100)

pclusters$size
```



```
> clustering_ch <- kmeansruns(pmatrix, krange = 1:10, criterion = "ch")
> clustering_ch$bestk
[1] 2
> clustering_asw <- kmeansruns(pmatrix, krange = 1:10, criterion = "asw")
> clustering_asw$bestk
[1] 3
> kbest_p <- 2
> pclusters <- kmeans(pmatrix, kbest_p, nstart = 100, iter.max = 100)
> pclusters$size
[1] 15 10
```

**Implementation of Association Rule Mining**

**Dataset: https://github.com/WinVector/PDSwR2/tree/master/Bookdata**



Listing 9.18 Reading in the book data

```
library(arules)
bookbaskets <- read.transactions("bookdata.tsv.gz",
                                 format = "single",
                                  header = TRUE,
                                  sep = "\t",
                                  cols = c("userid", "title"),
                                  rm.duplicates = TRUE)
```

Loads the arules package

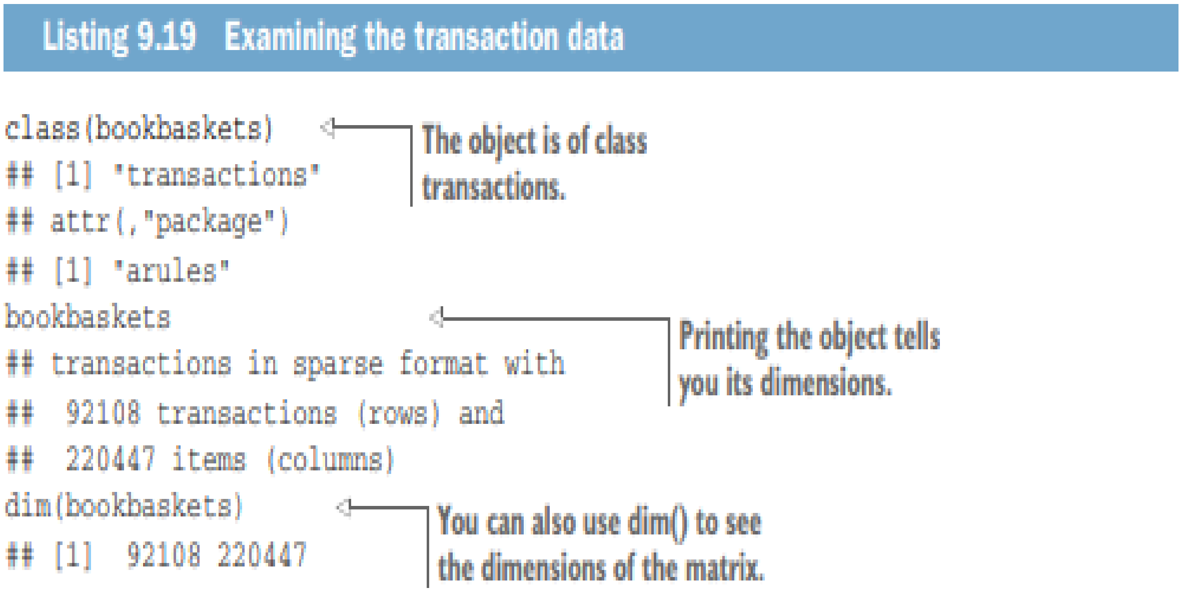
Specifies the file and the file format

Specifies that the input file has a header

Specifies the column separator (a tab)

Specifies the column of transaction IDs and of item IDs, respectively

Tells the function to look for and remove duplicate entries (for example, multiple entries for "The Hobbit" by the same user)



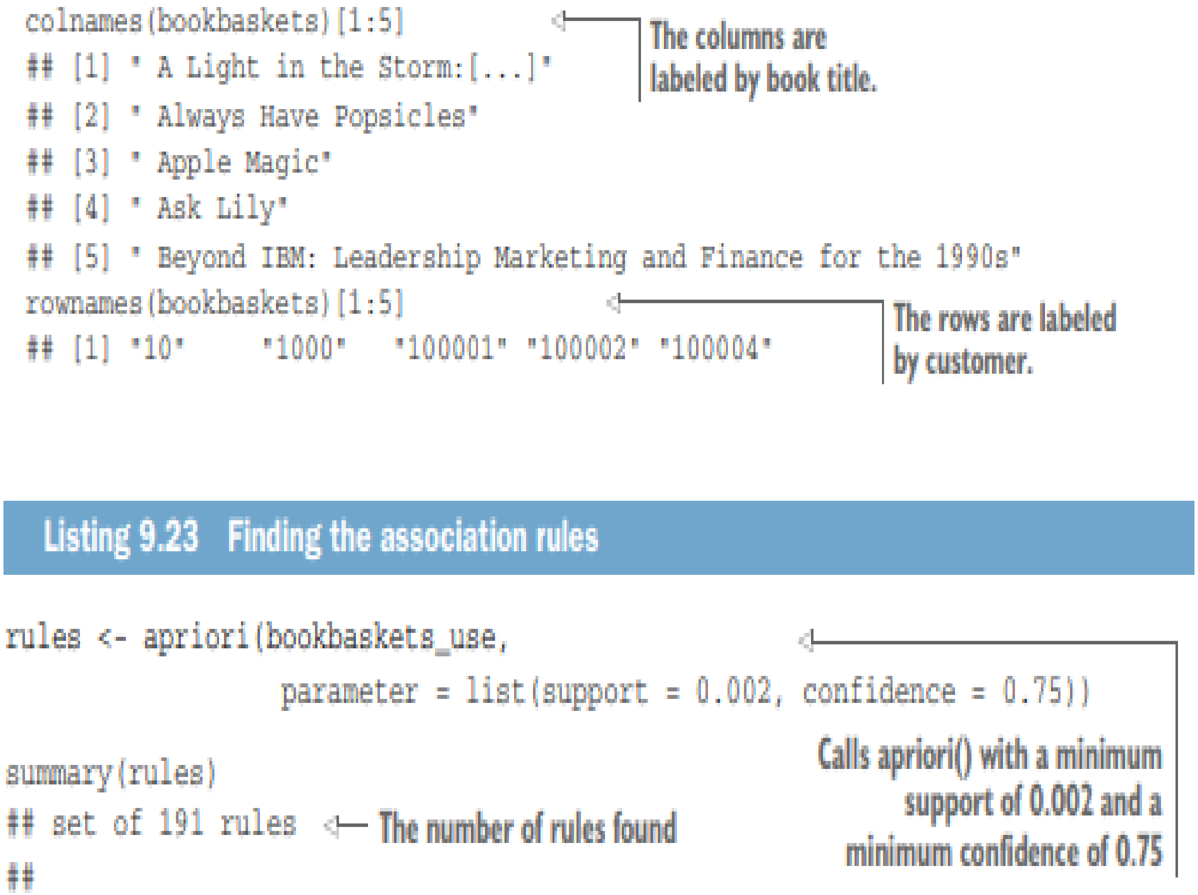
Listing 9.19 Examining the transaction data

```
class(bookbaskets)
## [1] "transactions"
## attr(,"package")
## [1] "arules"
bookbaskets
## transactions in sparse format with
##  92108 transactions (rows) and
##  220447 items (columns)
dim(bookbaskets)
## [1]  92108 220447
```

The object is of class transactions.

Printing the object tells you its dimensions.

You can also use dim() to see the dimensions of the matrix.



```
colnames(bookbaskets)[1:5]
## [1] " A Light in the Storm:[...]"
## [2] " Always Have Popsicles"
## [3] " Apple Magic"
## [4] " Ask Lily"
## [5] " Beyond IBM: Leadership Marketing and Finance for the 1990s"
rownames(bookbaskets)[1:5]
## [1] "10"     "1000"   "100001" "100002" "100004"
```

The columns are labeled by book title.

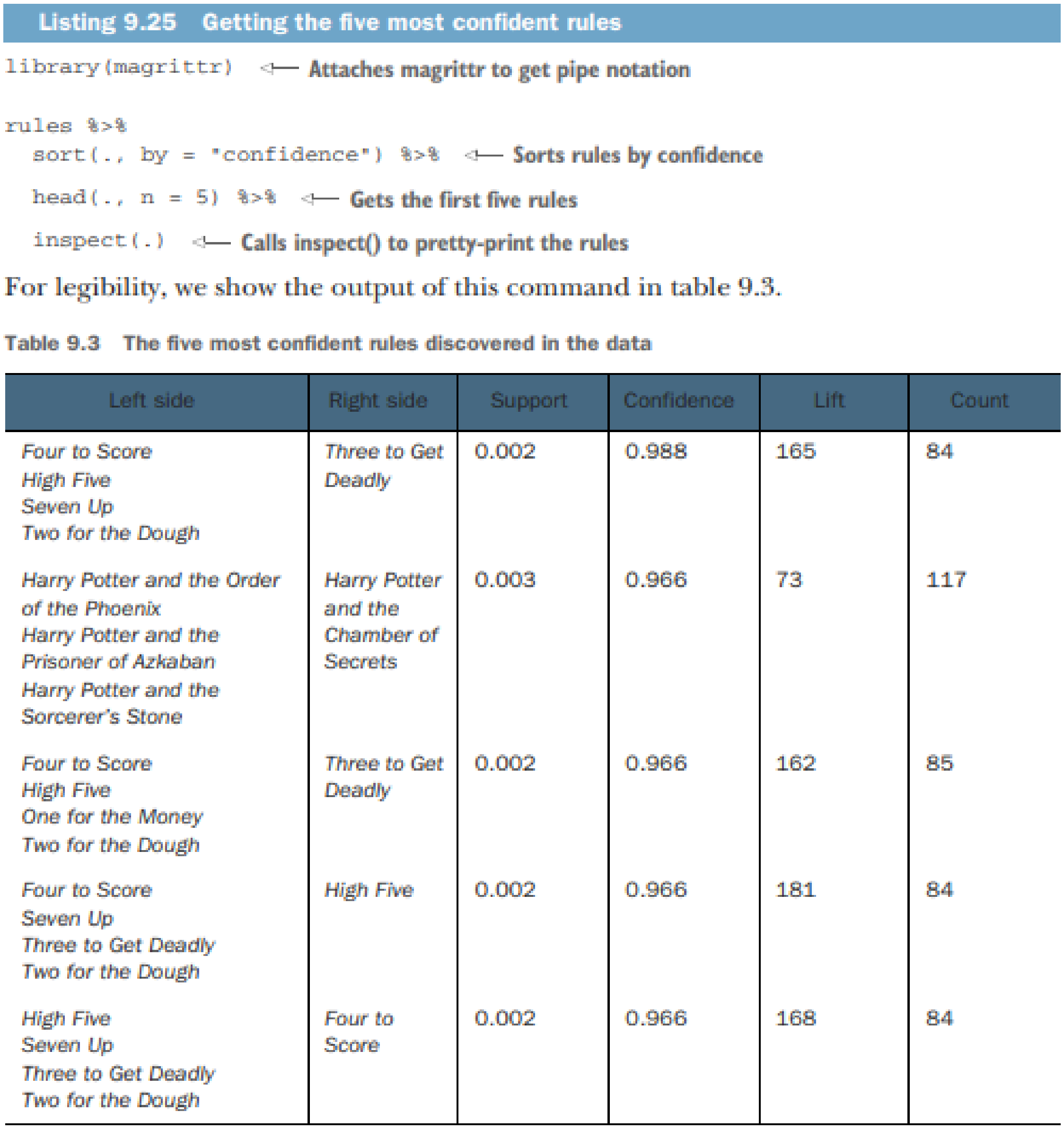
The rows are labeled by customer.

### Listing 9.23 Finding the association rules

```
rules <- apriori(bookbaskets_use,
                 parameter = list(support = 0.002, confidence = 0.75))

summary(rules)
## set of 191 rules
##
```

Calls apriori() with a minimum support of 0.002 and a minimum confidence of 0.75

The number of rules found



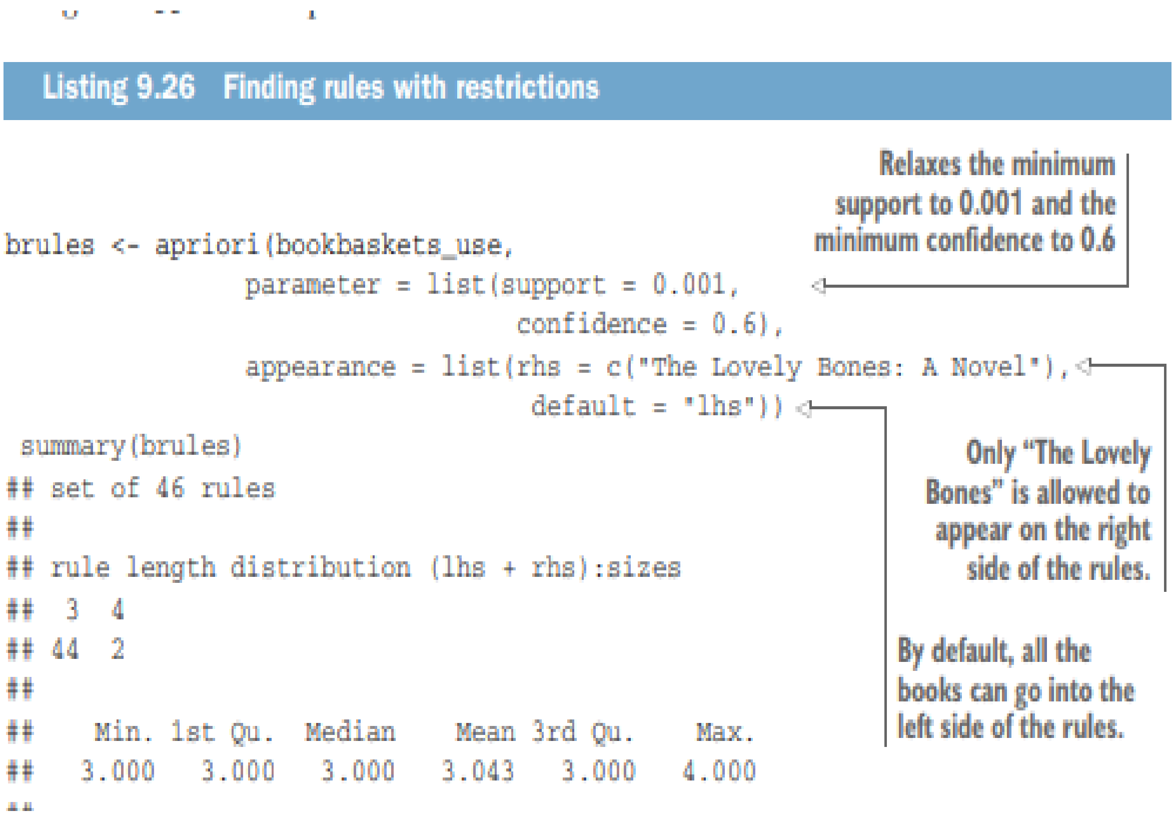
**Listing 9.25 Getting the five most confident rules**

```
library(magrittr)   ◁— Attaches magrittr to get pipe notation

rules %>%
  sort(., by = "confidence") %>%   ◁— Sorts rules by confidence
  head(., n = 5) %>%   ◁— Gets the first five rules
  inspect(.)   ◁— Calls inspect() to pretty-print the rules
```

For legibility, we show the output of this command in table 9.3.

**Table 9.3 The five most confident rules discovered in the data**

| Left side | Right side | Support | Confidence | Lift | Count |
|---|---|---|---|---|---|
| *Four to Score*<br>*High Five*<br>*Seven Up*<br>*Two for the Dough* | *Three to Get Deadly* | 0.002 | 0.988 | 165 | 84 |
| *Harry Potter and the Order of the Phoenix*<br>*Harry Potter and the Prisoner of Azkaban*<br>*Harry Potter and the Sorcerer's Stone* | *Harry Potter and the Chamber of Secrets* | 0.003 | 0.966 | 73 | 117 |
| *Four to Score*<br>*High Five*<br>*One for the Money*<br>*Two for the Dough* | *Three to Get Deadly* | 0.002 | 0.966 | 162 | 85 |
| *Four to Score*<br>*Seven Up*<br>*Three to Get Deadly*<br>*Two for the Dough* | *High Five* | 0.002 | 0.966 | 181 | 84 |
| *High Five*<br>*Seven Up*<br>*Three to Get Deadly*<br>*Two for the Dough* | *Four to Score* | 0.002 | 0.966 | 168 | 84 |



Listing 9.26 Finding rules with restrictions

```
brules <- apriori(bookbaskets_use,
                  parameter = list(support = 0.001,
                                   confidence = 0.6),
                  appearance = list(rhs = c("The Lovely Bones: A Novel"),
                                    default = "lhs"))
 summary(brules)
## set of 46 rules
##
## rule length distribution (lhs + rhs):sizes
##  3  4
## 44  2
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   3.000   3.000   3.043   3.000   4.000
##
```

Relaxes the minimum support to 0.001 and the minimum confidence to 0.6

Only "The Lovely Bones" is allowed to appear on the right side of the rules.

By default, all the books can go into the left side of the rules.

## Implementation of Bagging and Boosting:

# Load necessary libraries

install.packages("rpart")   # For decision trees

install.packages("randomForest")  # For random forest

install.packages("ipred")   # For bagging

library(rpart)

library(randomForest)

library(ipred)

# Load the iris dataset

data(iris)

```
# Set seed for reproducibility
set.seed(42)

# Split the data into training and testing sets
set.seed(123)
sample_index <- sample(1:nrow(iris), 0.7 * nrow(iris))
train_data <- iris[sample_index, ]
test_data <- iris[-sample_index, ]
# Train decision tree model
decision_tree_model <- rpart(Species ~ ., data = train_data, method = "class")
# Plot the decision tree
plot(decision_tree_model)
text(decision_tree_model, use.n = TRUE)
# Predict on test data
predictions_tree <- predict(decision_tree_model, test_data, type = "class")
# Calculate accuracy
accuracy_tree <- mean(predictions_tree == test_data$Species)
print(paste("Decision Tree Accuracy: ", accuracy_tree))

# Train bagging model using the ipred package
bagging_model <- bagging(Species ~ ., data = train_data, nbagg = 50)

# Predict on test data
predictions_bagging <- predict(bagging_model, test_data)
```

```
# Calculate accuracy

accuracy_bagging <- mean(predictions_bagging == test_data$Species)

print(paste("Bagging Accuracy: ", accuracy_bagging))


# Train random forest model

random_forest_model <- randomForest(Species ~ ., data = train_data,ntree = 100,mtry = 2)

# Syntax for random forest

 rf_model <- randomForest(formula, data = dataset, ntree = 100, mtry = 2, importance = TRUE)

 # Arguments:

# formula: The formula in the form of `response ~ predictors` (e.g., Species ~ .).

# data: The dataset on which to apply the random forest model.

# ntree: The number of trees to grow (default is 500, but smaller values like 100 work fine).

 # mtry: The number of features randomly selected at each split (default is the square root of the number of predictors for classification).

# importance: Whether to calculate variable importance (TRUE/FALSE).

# Predict on test data

predictions_rf <- predict(random_forest_model, test_data)

# Calculate accuracy

accuracy_rf <- mean(predictions_rf == test_data$Species)

print(paste("Random Forest Accuracy: ", accuracy_rf))
```

### Implementation of GLM:

```
set.seed(602957)
x <- rnorm(1000)
noise <- rnorm(1000, sd = 1.5)
y <- 3 * sin(2 * x) + cos(0.75 * x) - 1.5 * (x^2) + noise
select <- runif(1000)
frame <- data.frame(y = y, x = x)
train <- frame[select > 0.1, ]
test <-frame[select <= 0.1, ]
lin_model <- lm(y ~ x, data = train)
summary(lin_model)
train$pred_lin <- predict(lin_model, train)
install.packages("ggplot2")
library(ggplot2)
ggplot(train, aes(x = pred_lin, y = y)) +
  geom_point(alpha = 0.3) +
  geom_abline()
install.packages("mgcv")
library(mgcv)
gam_model <- gam(y ~ s(x), data = train)
gam_model$converged
summary(gam_model)
```

```
sx <- predict(gam_model, type = "terms")

xframe <- cbind(train, sx = sx[,1])

ggplot(xframe, aes(x = x)) +

  geom_point(aes(y = y), alpha = 0.4) +

  geom_line(aes(y = sx))
```

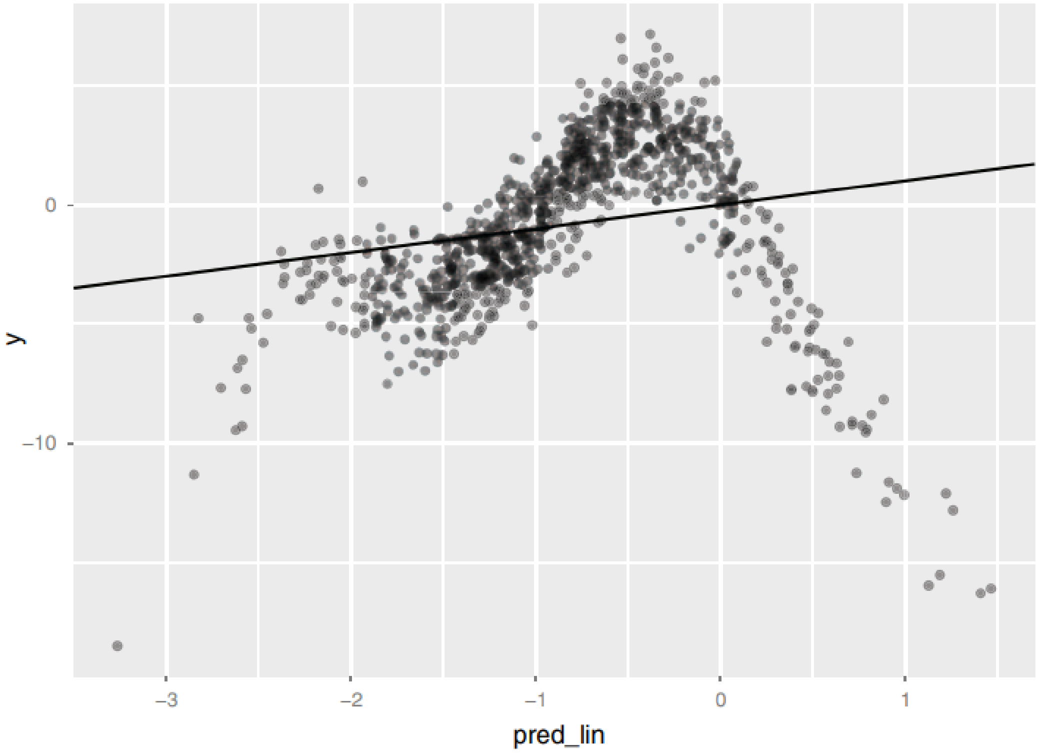


**Figure 10.14** Linear model's predictions vs. actual response. The solid line is the line of perfect prediction (`prediction == actual`).

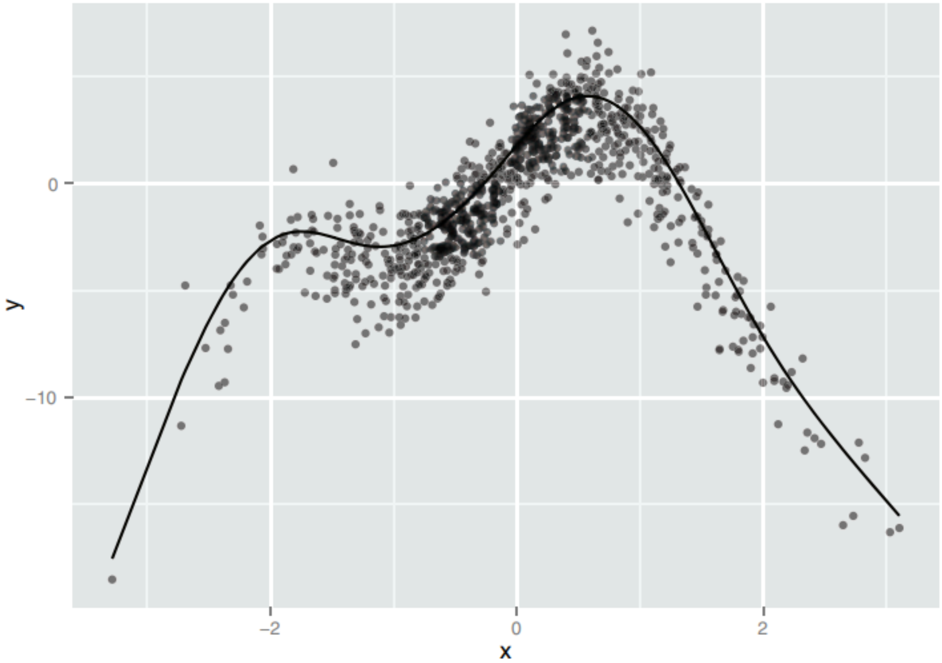
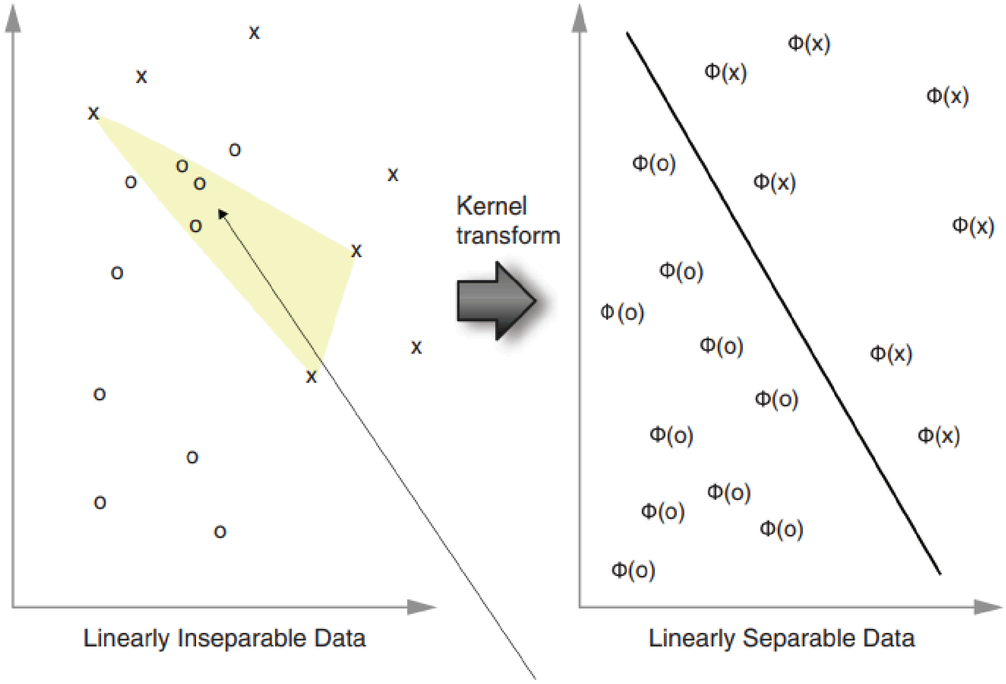


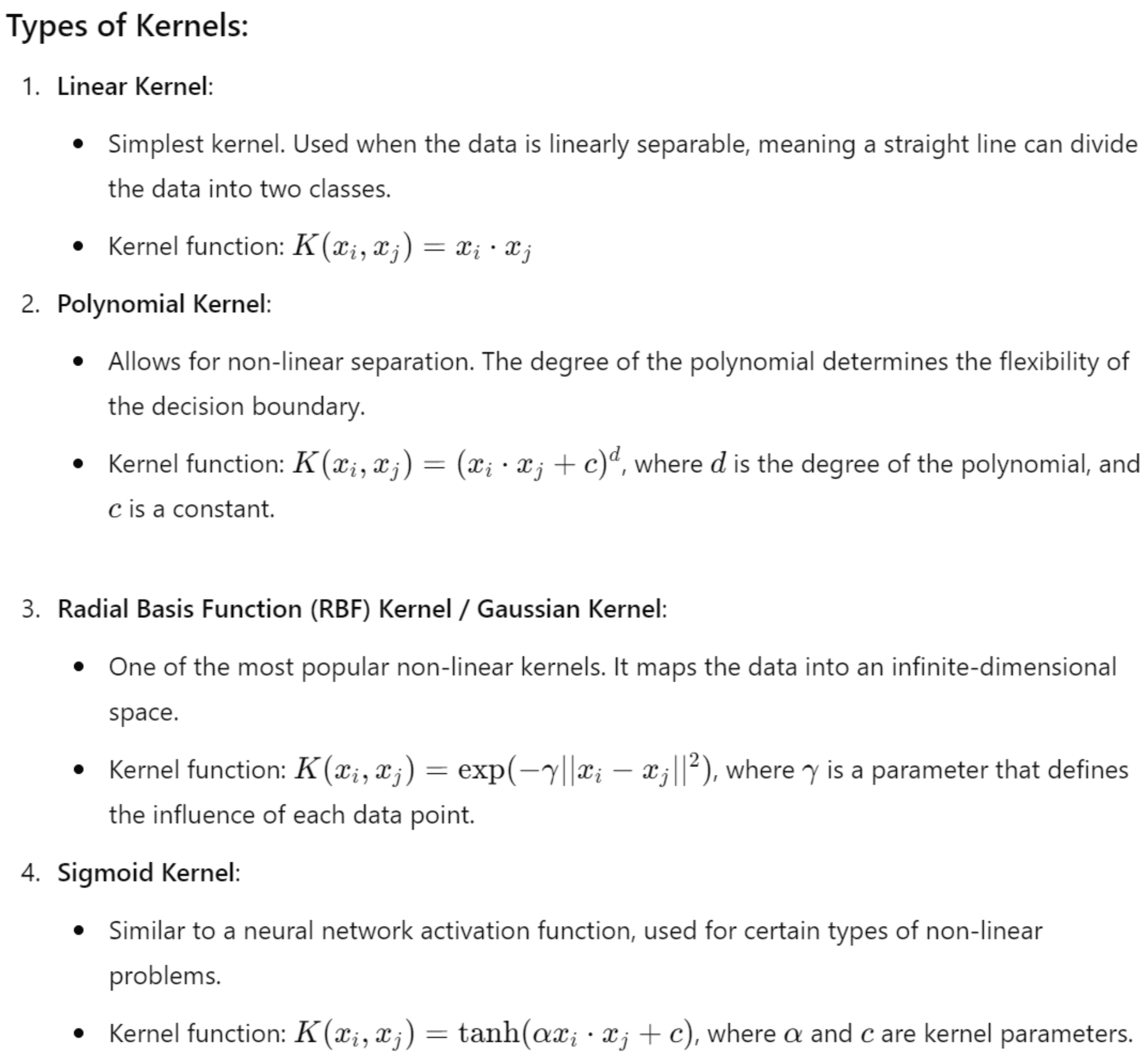
Figure 10.13 A spline that has been fit through a series of points

## Solving “inseparable” problems using support vector machines:



An o inside triangle of x’s means there is
no way to separate the x’s from the o’s using
a single straight line (or a linear separator).

In machine learning, a **kernel** is a function used in algorithms like Support Vector Machines (SVMs) to transform the data into a higher-dimensional space where it becomes easier to separate or classify the data points. Essentially, kernels allow SVMs to handle complex relationships between data points without explicitly working in higher dimensions, which can be computationally expensive.



### Types of Kernels:

1. **Linear Kernel**:
   - Simplest kernel. Used when the data is linearly separable, meaning a straight line can divide the data into two classes.
   - Kernel function: $K(x_i, x_j) = x_i \cdot x_j$
2. **Polynomial Kernel**:
   - Allows for non-linear separation. The degree of the polynomial determines the flexibility of the decision boundary.
   - Kernel function: $K(x_i, x_j) = (x_i \cdot x_j + c)^d$, where $d$ is the degree of the polynomial, and $c$ is a constant.
3. **Radial Basis Function (RBF) Kernel / Gaussian Kernel**:
   - One of the most popular non-linear kernels. It maps the data into an infinite-dimensional space.
   - Kernel function: $K(x_i, x_j) = \exp(-\gamma ||x_i - x_j||^2)$, where $\gamma$ is a parameter that defines the influence of each data point.
4. **Sigmoid Kernel**:
   - Similar to a neural network activation function, used for certain types of non-linear problems.
   - Kernel function: $K(x_i, x_j) = \tanh(\alpha x_i \cdot x_j + c)$, where $\alpha$ and $c$ are kernel parameters.

**svm(x, y = NULL, data = NULL, scale = TRUE, type = NULL, kernel = "radial", degree = 3, gamma = 1, cost = 1, ...)**

**x**: Input features (independent variables).

**y**: Target labels (dependent variable).

**data**: Data frame containing `x` and `y`.

**kernel**: Type of kernel to use (e.g., `"linear"`, `"polynomial"`, `"radial"`, `"sigmoid"`).

**cost**: Regularization parameter (default = 1).

**degree**: Degree of polynomial (used only with polynomial kernel).

**gamma**: Kernel coefficient for RBF, polynomial, and sigmoid kernels.