1. **Linear Models**
- Simple Linear Regression
- Multiple Regression
- Logistic Regression
- Poisson Regression

2. **Nonlinear Models**
- Nonlinear least squares
- Generalized additive models
- Decision trees
- Random forests

3. **Time Series:**
- Autoregressive Moving Average
- VAR
- GARCH

4. **Clustering**
- K –Means
- PAM
- Hierarchical Clustering

Regression Analysis

- Regression analysis is very widely used statistical tool to establish a relationship model between two variables.
- One of these variable is called predictor variable whose value is gathered through experiments.
- The other variable is called response variable whose value is derived from the predictor variable.

1. Linear Regression

- In linear regression these two variables are related through an equation, where exponent (power) of both these variables is 1.
- Mathematically a linear relationship represents a straight line when plotted as a graph.
- A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.
- The general mathematical equation for a linear regression is −

$$\mathbf{y = ax + b}$$

Following is the description of the parameters used :

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are constants which are called the coefficients.

Steps to Establish a Regression

- A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is −

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- To predict the weight of new persons, use the **predict()**function in R.

Input Data

Below is the sample data representing the observations −

- # Values of height 151, 174, 138, 186, 128, 136, 179, 163, 152, 131
- # Values of weight. 63, 81, 56, 91, 47, 57, 76, 72, 62, 48

lm() Function

- This function creates the relationship model between the predictor and the response variable.

Syntax

- The basic syntax for **lm()** function in linear regression is −
- lm(formula,data) Following is the description of the parameters used −
- **formula** is a symbol presenting the relation between x and y.
- **data** is the vector on which the formula will be applied.

Create relationship model and get the coefficients

```
>x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
>y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
>relation <- lm(y~x)
>print(relation)
```

```
Call:
lm(formula = y ~ x)
Coefficients:
(Intercept)        x
-38.4551          0.6746
```

Get the summary of the relationship

```
>print(summary(relation))
Call:
lm(formula = y ~ x)
Residuals:
    Min      1Q  Median      3Q
-6.3002 -1.6629  0.0412  1.8944
    Max
 3.9775
Coefficients:
            Estimate Std. Error
(Intercept) -38.45509    8.04901
x             0.67461    0.05191
            t value Pr(>|t|)
(Intercept)  -4.778  0.00139 **
x            12.997 1.16e-06 ***
---
Signif. codes:
  0 '***' 0.001 '**' 0.01 '*' 0.05
  '.' 0.1 ' ' 1
Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,  Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

predict()

Syntax

predict(object, newdata)

Following is the description of the parameters used:

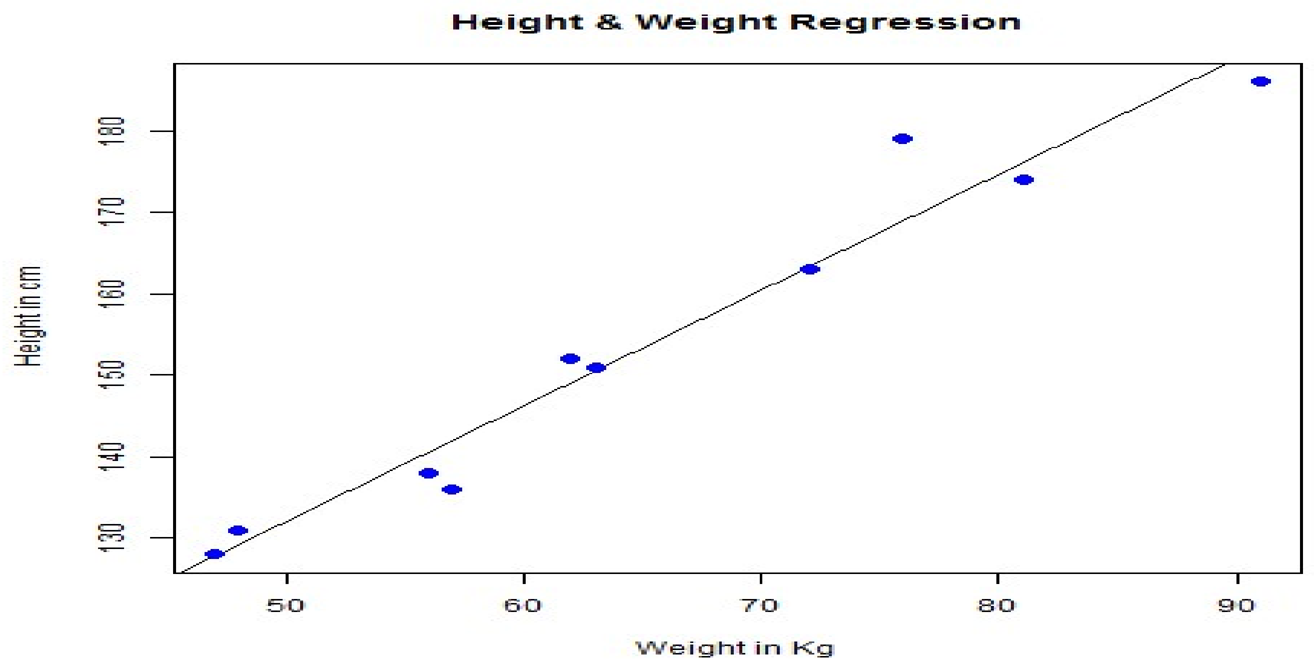
- object is the formula which is already created using the lm() function.
- newdata is the vector containing the new value for predictor variable.

Predict the weight of new persons

```
# The predictor vector.
>x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
# The resposne vector.
>y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm() function.
>relation <- lm(y~x)
# Find weight of a person with height 170.
>a <- data.frame(x = 170)
>result <-  predict(relation,a)
>print(result)
 1
76.22869
```

Visualize the Regression Graphically

```
# Create the predictor and response variable.
>x<- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
>y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
>relation <- lm(y~x)
# Give the chart file a name.
>png(file = "linearregression.png")
# Plot the chart.
>plot(y,x,col = "blue",main = "Height & Weight Regression",
abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")
# Save the file.
>dev.off()
```



2. Multiple Regression

- Multiple regression is an extension of linear regression into relationship between more than two variables. In simple linear relation we have one predictor and one response variable, but in multiple regression we have more than one predictor variable and one response variable.

The general mathematical equation for multiple regression is

$$y = a + b1x1 + b2x2 + ...bnxn$$

Following is the description of the parameters used

- y is the response variable.
- a, b1, b2...bn are the coefficients.
- x1, x2, ...xn are the predictor variables.

- We create the regression model using the lm() function in R. The model determines the value of the coefficients using the input data. Next we can predict the value of the response variable for a given set of predictor variables using these coefficients.

lm()

- This function creates the relationship model between the predictor and the response variable.

Syntax

lm(y ~ x1+x2+x3...,data)

Following is the description of the parameters used

- **formula** is a symbol presenting the relation between the response variable and predictor variables.
- **data** is the vector on which the formula will be applied.

Example

Input Data

- Consider the data set "mtcars" available in the R environment. It gives a comparison between different car models in terms of mileage per gallon (mpg), cylinder displacement("disp"), horse power("hp"), weight of the car("wt") and some more parameters.
- The goal of the model is to establish the relationship between "mpg" as a response variable with "disp","hp" and "wt" as predictor variables. We create a subset of these variables from the mtcars data set for this purpose.

•

```
>input <- mtcars[,c("mpg","disp","hp","wt")]
>print(head(input))
                        mpg      disp    hp       wt
Mazda RX4               21.0     160     110      2.620
Mazda RX4 Wag           21.0     160     110      2.875
Datsun 710              22.8     108      93      2.320
Hornet 4 Drive          21.4     258     110      3.215
Hornet Sportabout       18.7     360     175      3.440
Valiant                 18.1     225     105      3.460
```

Create Relationship Model & get the Coeffients

```
input <- mtcars[,c("mpg","disp","hp","wt")]
# Create the relationship model.
model <- lm(mpg~disp+hp+wt, data = input)
# Show the model.
print(model)
```

```
# Get the Intercept and coefficients as vector elements.
cat("# # # # The Coefficient Values # # # ","\n")
a <- coef(model)[1]
print(a)
Xdisp <- coef(model)[2]
Xhp <- coef(model)[3]
Xwt <- coef(model)[4]
print(Xdisp)
print(Xhp)
print(Xwt)
```

Result

```
Call:
lm(formula = mpg ~ disp + hp + wt, data = input)
Coefficients:
(Intercept)       disp          hp         wt
  37.105505    -0.000937    -0.031157   -3.800891
```

```
# # # # The Coefficient Values # # #
(Intercept)
  37.10551
        disp
-0.0009370091
        hp
-0.03115655
      wt
-3.800891
```

3. Logistic Regressions

- The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It actually measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables.

The general mathematical equation for logistic regression is

y = 1/(1+e^-(a+b1x1+b2x2+b3x3+...))

Following is the description of the parameters used

- y is the response variable.
- x is the predictor variable.
- a and b are the coefficients which are numeric constants.

The function used to create the regression model is the glm() function.

Syntax

```
glm(formula,data,family)
```

Following is the description of the parameters used

- formula is the symbol presenting the relationship between the variables.
- data is the data set giving the values of these variables.
- family is R object to specify the details of the model. It's value is binomial for logistic regression.

Example

- The in-built data set "mtcars" describes different models of a car with their various engine specifications. In "mtcars" data set, the transmission mode (automatic or manual) is described by the column am which is a binary value (0 or 1). We can create a logistic regression model between the columns "am" and 3 other columns - hp, wt and cyl.

```
# Select some columns form mtcars.
>input <- mtcars[,c("am","cyl","hp","wt")]
>print(head(input))
```

Result

```
                    am   cyl   hp    wt
Mazda RX4            1    6   110  2.620
Mazda RX4 Wag        1    6   110  2.875
Datsun 710           1    4    93  2.320
Hornet 4 Drive       0    6   110  3.215
Hornet Sportabout    0    8   175  3.440
Valiant              0    6   105  3.460
```

Create Regression Model

- We use the **glm()** function to create the regression model and get its summary for analysis.

```
>input <- mtcars[,c("am","cyl","hp","wt")]
>am.data = glm(formula = am ~ cyl + hp + wt, data = input, family = binomial)
>print(summary(am.data))
```

Result

```
Call:
glm(formula = am ~ cyl + hp + wt, family = binomial, data = input)
Deviance Residuals:
    Min        1Q      Median       3Q       Max
-2.17272     -0.14907  -0.01464     0.14116   1.27641
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 19.70288    8.11637   2.428   0.0152 *
cyl          0.48760    1.07162   0.455   0.6491
hp           0.03259    0.01886   1.728   0.0840 .
wt          -9.14947    4.15332  -2.203   0.0276 *
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for binomial family taken to be 1)
    Null deviance: 43.2297  on 31  degrees of freedom
Residual deviance:  9.8415  on 28  degrees of freedom
AIC: 17.841
Number of Fisher Scoring iterations: 8
```

Conclusion

- In the summary as the p-value in the last column is more than 0.05 for the variables "cyl" and "hp", we consider them to be insignificant in contributing to the value of the variable "am". Only weight (wt) impacts the "am" value in this regression model.

4. Poisson Regression

- Poisson Regression involves regression models in which the response variable is in the form of counts and not fractional numbers. For example, the count of number of births or number of wins in a football match series. Also the values of the response variables follow a Poisson distribution.

The general mathematical equation for Poisson regression is −

$$\log(y) = a + b1x1 + b2x2 + bnxn.....$$

Following is the description of the parameters used −

- y is the response variable.
- a and b are the numeric coefficients.
- x is the predictor variable.

The function used to create the Poisson regression model is the glm() function.

Syntax:

```
glm(formula,data,family)
```

- **formula** is the symbol presenting the relationship between the variables.
- **data** is the data set giving the values of these variables.
- **family** is R object to specify the details of the model. It's value is 'Poisson' for Logistic Regression.

Example

- We have the in-built data set "warpbreaks" which describes the effect of wool type (A or B) and tension (low, medium or high) on the number of warp breaks per loom. Let's consider "breaks" as the response variable which is a count of number of breaks. The wool "type" and "tension" are taken as predictor variables.

Input Data

```
>input<-warpbreaks
>print(head(input))
```

Result:

```
   breaks     wool  tension
1  26         A       L
2  30         A       L
3  54         A       L
4  25         A       L
5  70         A       L
6  52         A       L
```

Create Regression Model

```
>output <-glm(formula = breaks ~ wool+tension, data = warpbreaks, family =
    poisson)
>print(summary(output))
```

Result:

```
Call:
glm(formula = breaks ~ wool + tension, family = poisson, data = warpbreaks)

Deviance Residuals:
    Min       1Q    Median       3Q      Max
 -3.6871  -1.6503  -0.4269   1.1902   4.2616

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.69196    0.04541  81.302  < 2e-16 ***
woolB       -0.20599    0.05157  -3.994 6.49e-05 ***
tensionM    -0.32132    0.06027  -5.332 9.73e-08 ***
tensionH    -0.51849    0.06396  -8.107 5.21e-16 ***
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 297.37  on 53  degrees of freedom
Residual deviance: 210.39  on 50  degrees of freedom
AIC: 493.06

Number of Fisher Scoring iterations: 4
```

Non-linear Models

- When modeling real world data for regression analysis, we observe that it is rarely the case that the equation of the model is a linear equation giving a linear graph.
- Most of the time, the equation of the model of real world data involves mathematical functions of higher degree like an exponent of 3 or a sin function.
- In such a scenario, the plot of the model gives a curve rather than a line. The goal of both linear and non-linear regression is to adjust the values of the model's parameters to find the line or curve that comes closest to your data.
- On finding these values we will be able to estimate the response variable with good accuracy.

1. Non- Linear Least Squares

- In Least Square regression, we establish a regression model in which the sum of the squares of the vertical distances of different points from the regression curve is minimized.
- We generally start with a defined model and assume some values for the coefficients. We then apply the nls() function of R to get the more accurate values along with the confidence intervals.

4. Clustering

- Play's a big role in modern machine learning, is the partitioning of data into groups.
- This can be done in a number of ways, the two most popular being K-means and hierarchical clustering.
- In terms of a data.frame, a clustering algorithm finds out which rows are similar to each other.
- Rows that are grouped together are supposed to have high similarity to each other and low similarity with rows outside the grouping.

1. K-means

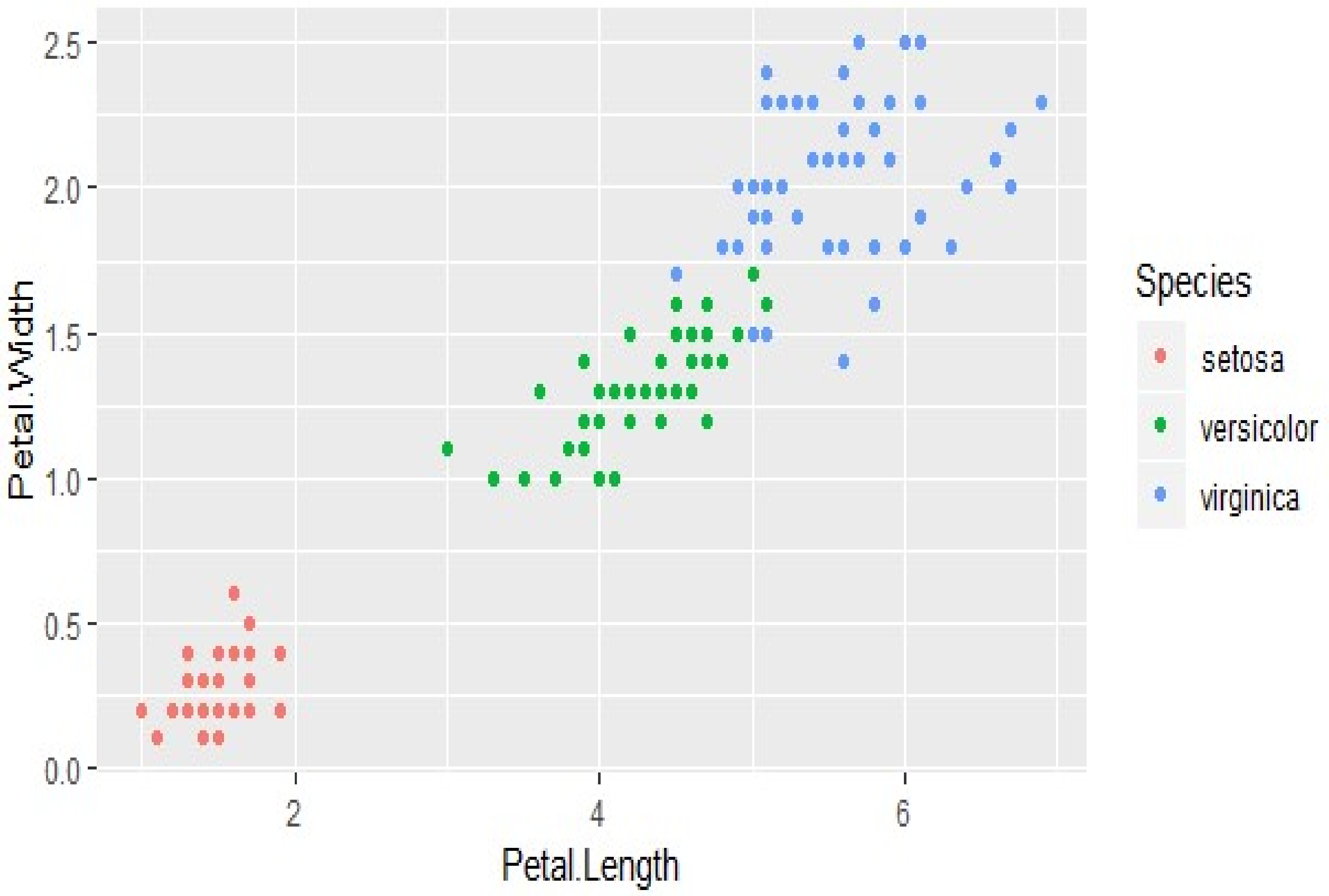
- One of the more popular algorithms for clustering is K-means.
- It divides the observations into discrete groups based on some distance metric.
- For K-means we need to specify the number of clusters, and then the algorithm assigns observations into that many clusters.
- In R, K-means is done with the aptly named K-means function. The first 2 arguments are the data to be clustered, which must be all numeric(K-means does not work with categorical data), and the number of centers(clusters).

- Printing the K-means objects displays the size of the clusters, the cluster mean for each column, the cluster membership for each row and similarity measures.
- Plotting the results of K-means clustering can be difficult because of high dimensional nature of the data. To overcome this, the plot.kmeans function in useful performs multidimensional scaling to project the data into two dimensions, and then color codes the points according to cluster membership.

Code:

```
> library(datasets)
>head(iris)
Petal.Width Species
1         0.2  setosa
2         0.2  setosa
3         0.2  setosa
4         0.2  setosa
5         0.2  setosa
6         0.4  setosa
```

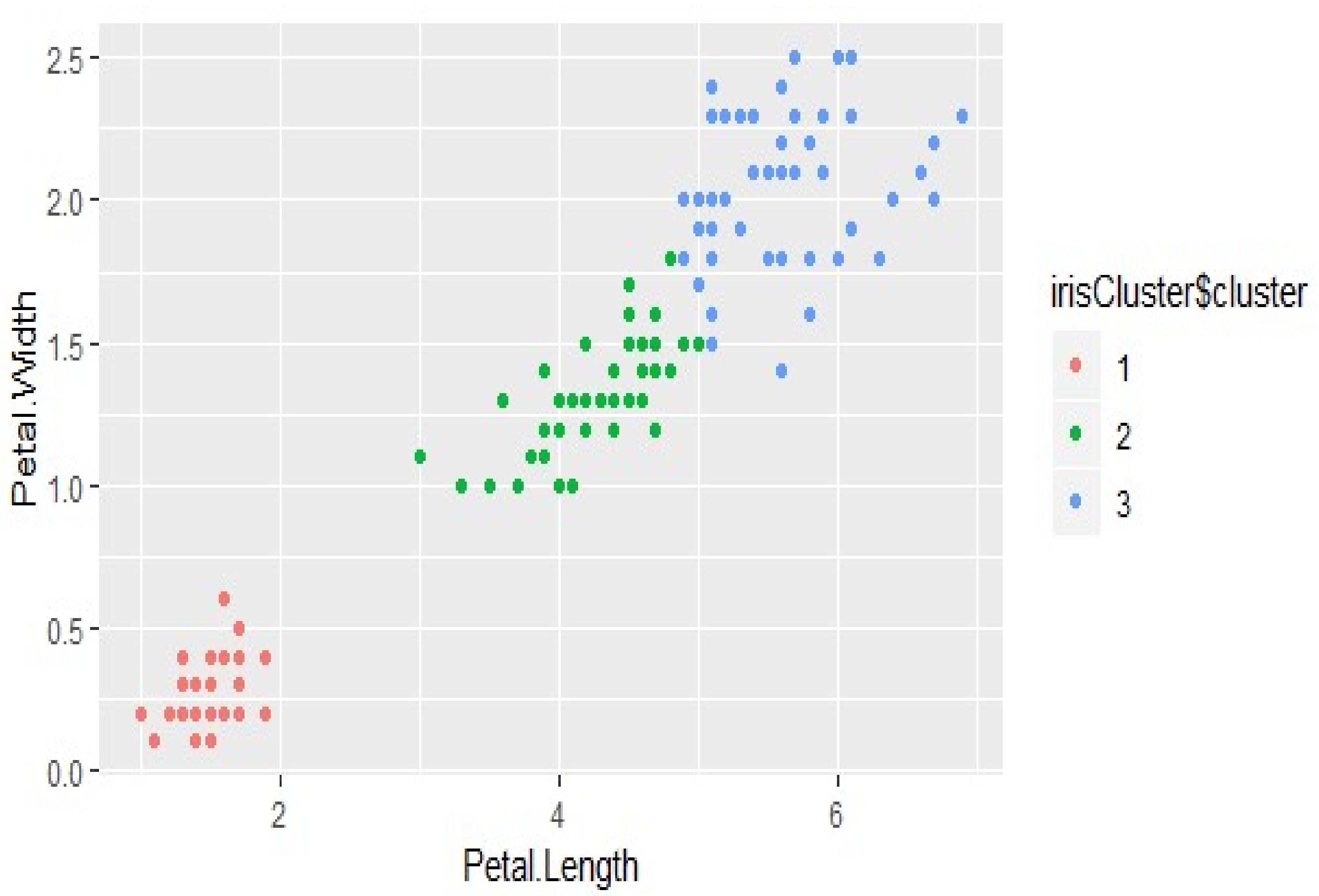
```
>library(ggplot2)
>ggplot(iris,aes(Petal.Length,Petal.Width,color=Species))+geom_point()
```



```
>set.seed(20)
>irisCluster<-kmeans(iris[,3:4],3,nstart=20)
>irisCluster
Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [25] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [49] 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [73] 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2
 [97] 2 2 2 2 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2
[121] 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3
[145] 3 3 3 3 3 3
Within cluster sum of squares by cluster:
[1]  2.02200 13.05769 16.29167
 (between_SS / total_SS =  94.3 %)
Available components:
[1] "cluster"      "centers"      "totss"
[4] "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
>
```

```
>table(irisCluster$cluster,iris$Species)
      setosa          versicolor          virginica
  1    50               0                 0
  2     0               48                4
  3     0               2                 46
> irisCluster$cluster<-as.factor(irisCluster$cluster)
>irisCluster$cluster
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [25] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [49] 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [73] 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2
 [97] 2 2 2 2 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2
[121] 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3
[145] 3 3 3 3 3 3
Levels: 1 2 3
```

```
>ggplot(iris,aes(Petal.Length,Petal.Width,
     color=irisCluster$cluster))+geom_point()
```



2. PAM

- Two problems with K-means clustering are that it does not work with categorical data and it is susceptible to outliers.
- An alternative is K-medoids. Instead of the center of a cluster being the mean of the cluster, the center is one of the actual observations in the cluster.
- The most common K-mediods algorithm is Partitioning Around Mediods(PAM). The cluster package contains the pam().
- If the data has a few missing values, pam handles missing values well. Before we run the clustering algorithm we clean up the data some more.
- Now we fit the clustering using pam from the cluster package. Each line represents an observation, and each grouping of lines is a cluster.
- Observations that fit the cluster well have large positive lines and observations that do not fit well have small or negative lines.
- A bigger average width for a cluster means a better clustering.

Code:

```
>x<-rbind(cbind(rnorm(10,0,0.5),rnorm(10,0,0.5)),
        cbind(rnorm(15,5,0.5),rnorm(15,5,0.5)))
```

```
>x
            [,1]         [,2]
 [1,] -0.10815576 -0.409241508
 [2,]  0.79507288 -0.771283919
 [3,]  0.77807164  0.277941073
 [4,]  0.55422545 -0.184514485
 [5,] -0.54867092 -0.523669140
 [6,] -0.93030286  0.009089958
 [7,] -0.45678942  0.440938753
 [8,]  0.62278446  0.440930748
 [9,]  0.04392736  0.513121593
[10,]  0.21174095 -0.190654590
[11,]  5.54971762  4.169400227
[12,]  4.98454143  5.582606039
[13,]  5.09516971  4.465169011
[14,]  5.66760327  5.454441770
[15,]  5.36527617  4.341246290
[16,]  5.02810095  5.327714978
[17,]  5.66465281  4.669814088
[18,]  4.79594003  5.495790384
[19,]  4.59087145  4.658388232
[20,]  5.17947283  5.560660598
[21,]  5.02935803  5.194275645
[22,]  4.96528171  4.924616605
[23,]  4.80759187  4.767011671
[24,]  4.56283827  4.319742181
[25,]  5.59816533  4.985207120
```

```
>library(pamr)
>pamx<-pam(x,2)
>pamx
Medoids:
     ID
[1,] 10 0.2117409 -0.1906546
[2,] 22 4.9652817  4.9246166
Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Objective function:
    build      swap
0.8236515 0.6037261

Available components:
 [1] "medoids"    "id.med"     "clustering" "objective"
 [5] "isolation"  "clusinfo"   "silinfo"    "diss"
 [9] "call"       "data"
```

```
>summary(pamx)
Medoids:
     ID
[1,] 10 0.2117409 -0.1906546
[2,] 22 4.9652817  4.9246166
Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Objective function:
    build      swap
0.8236515 0.6037261
Numerical information per cluster:
     size  max_diss   av_diss diameter separation
[1,]   10 1.1593800 0.6674374 1.893649   5.528942
[2,]   15 0.9549435 0.5612518 1.583682   5.528942
Isolated clusters:
 L-clusters: character(0)
 L*-clusters: [1] 1 2
```

Silhouette plot information:

```
   cluster neighbor sil_width
10       1        2 0.8956637
1        1        2 0.8881902
4        1        2 0.8783324
9        1        2 0.8655514
5        1        2 0.8633453
7        1        2 0.8573630
8        1        2 0.8474618
3        1        2 0.8442775
6        1        2 0.8406352
2        1        2 0.8265019
21       2        1 0.9142036
22       2        1 0.9137526
16       2        1 0.9092210
23       2        1 0.9010146
25       2        1 0.8990212
20       2        1 0.8938403
13       2        1 0.8921443
12       2        1 0.8898658
17       2        1 0.8866756
18       2        1 0.8865927
14       2        1 0.8801415
15       2        1 0.8787858
19       2        1 0.8751181
11       2        1 0.8520542
24       2        1 0.8412541
```

```
Average silhouette width per cluster:
[1] 0.8607322 0.8875790
Average silhouette width of total data set:
[1] 0.8768403

300 dissimilarities, summarized :
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1335  0.8335  3.7113  3.9761  7.0918  8.6244
Metric :  euclidean
Number of objects : 25

Available components:
 [1] "medoids"    "id.med"     "clustering" "objective"
 [5] "isolation"  "clusinfo"   "silinfo"    "diss"
 [9] "call"       "data"
```

```
>(p2m<-pam(x,2,medoids=c(1,16)))
Medoids:
     ID
[1,] 10 0.2117409 -0.1906546
[2,] 22 4.9652817  4.9246166
Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Objective function:
    build      swap
0.6728364 0.6037261

Available components:
 [1] "medoids"    "id.med"     "clustering" "objective"
 [5] "isolation"  "clusinfo"   "silinfo"    "diss"
 [9] "call"       "data"
```

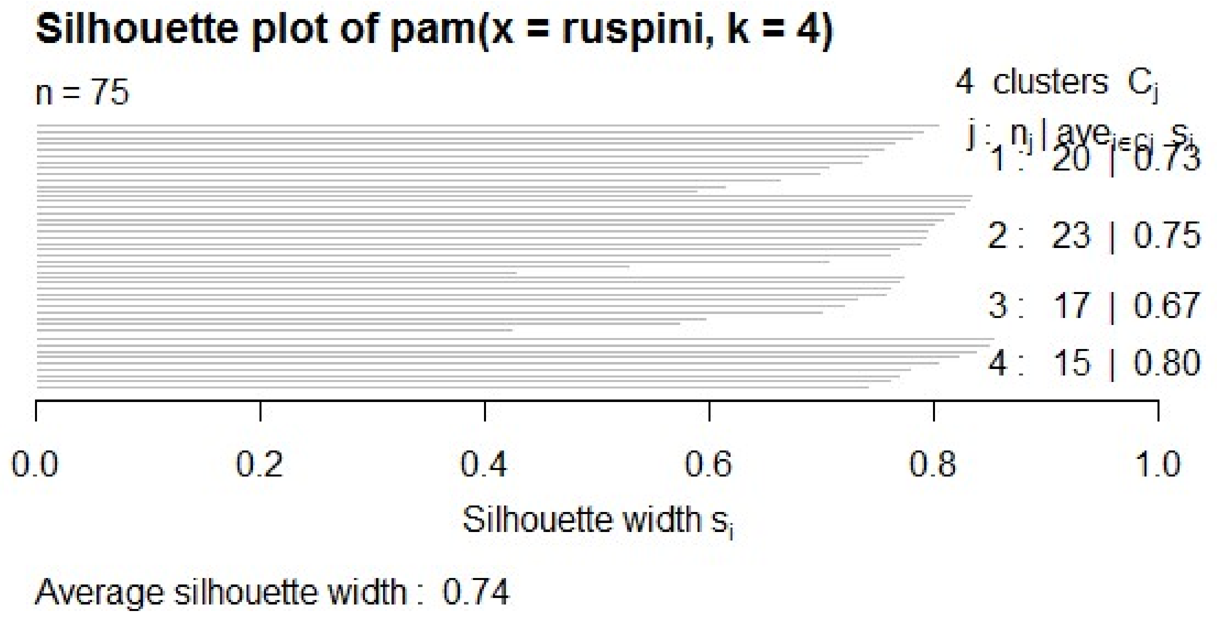
```
>p2.s<-pam(x,2,medoids=c(1,16),do.swap=FALSE)
>p2.s
Medoids:
     ID
[1,]  1 -0.1081558 -0.4092415
[2,] 16  5.0281009  5.3277150
Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Objective function:
   build      swap
0.6728364 0.6728364

Available components:
 [1] "medoids"    "id.med"     "clustering" "objective"
 [5] "isolation"  "clusinfo"   "silinfo"    "diss"
 [9] "call"       "data"
```

```
>p3m<-pam(x,3,trace=2)
C pam(): computing 301 dissimilarities from  25 x 2  matrix: [Ok]
pam()'s bswap(*, s=8.62438, pamonce=0): build 3 medoids:
    new repr. 24
    new repr. 10
    new repr. 16
  after build: medoids are 10 16 24
   swp new 13 <-> 24 old; decreasing diss. 13.4737 by -1.16452
end{bswap()}, end{cstat()}
>(p3m.<-pam(x,3,medoids = 3:1,trace=1))
C pam(): computing 301 dissimilarities from  25 x 2  matrix: [Ok]
pam()'s bswap(*, s=8.62438, pamonce=0): medoids given
  after build: medoids are  1  2  3
end{bswap()}, end{cstat()}
```

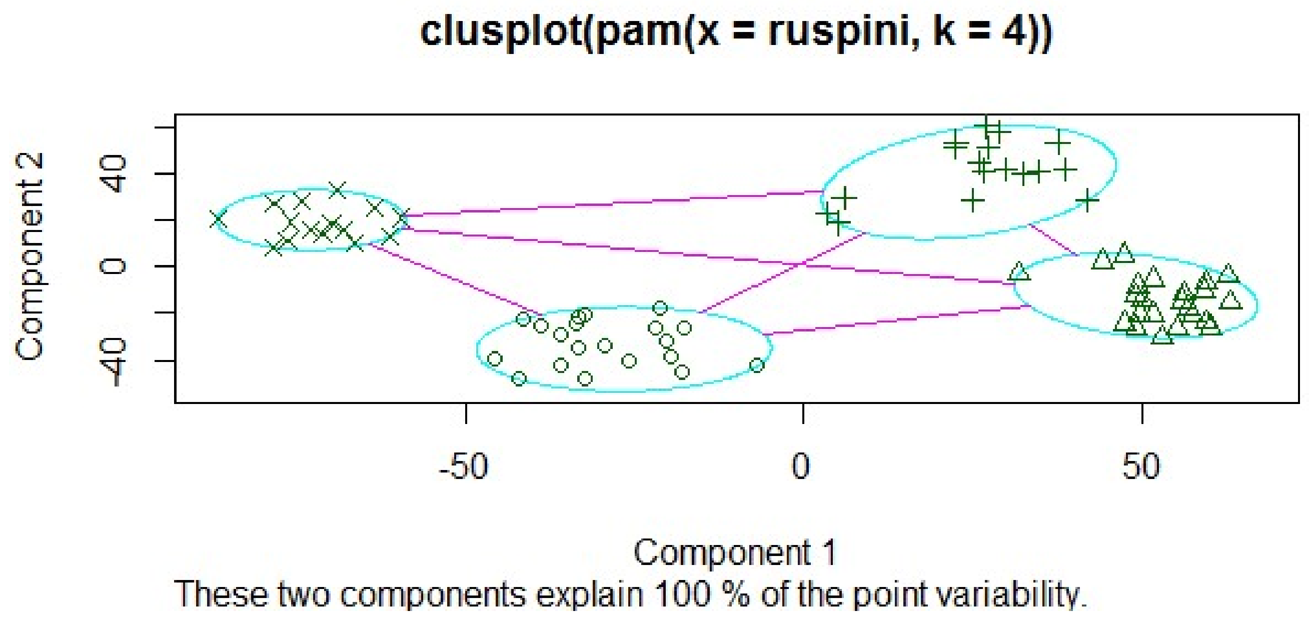
```
C pam(): computing 301 dissimilarities from  25 x 2  matrix: [Ok]
pam()'s bswap(*, s=8.62438, pamonce=0): medoids given
  after build: medoids are  1  2  3
end{bswap()}, end{cstat()}
Medoids:
     ID
[1,]  1 -0.1081558 -0.4092415
[2,]  8  0.6227845  0.4409307
[3,] 22  4.9652817  4.9246166
Clustering vector:
 [1] 1 1 2 2 1 1 1 2 2 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
Objective function:
    build      swap
3.9953828 0.5405376
Available components:
 [1] "medoids"    "id.med"     "clustering" "objective"
 [5] "isolation"  "clusinfo"   "silinfo"    "diss"
 [9] "call"       "data"
```

```
>pam(daisy(x,metric = "manhattan"),2,diss=TRUE)
Medoids:
     ID
[1,] 10 10
[2,] 22 22
Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Objective function:
    build      swap
1.0639279 0.7823442

Available components:
[1] "medoids"    "id.med"     "clustering" "objective"
[5] "isolation"  "clusinfo"   "silinfo"    "diss"
[9] "call"
```

```
>data("ruspini")
>plot(pam(ruspini,4),ask=TRUE)
Make a plot selection (or 0 to exit):
 1: plot  All
2: plot  Clusplot
3: plot  Silhouette Plot

Selection: 1
```



Selection:2


clusplot(pam(x = ruspini, k = 4))
Component 2
40
0
-40
-50
0
50
Component 1
These two components explain 100 % of the point variability.

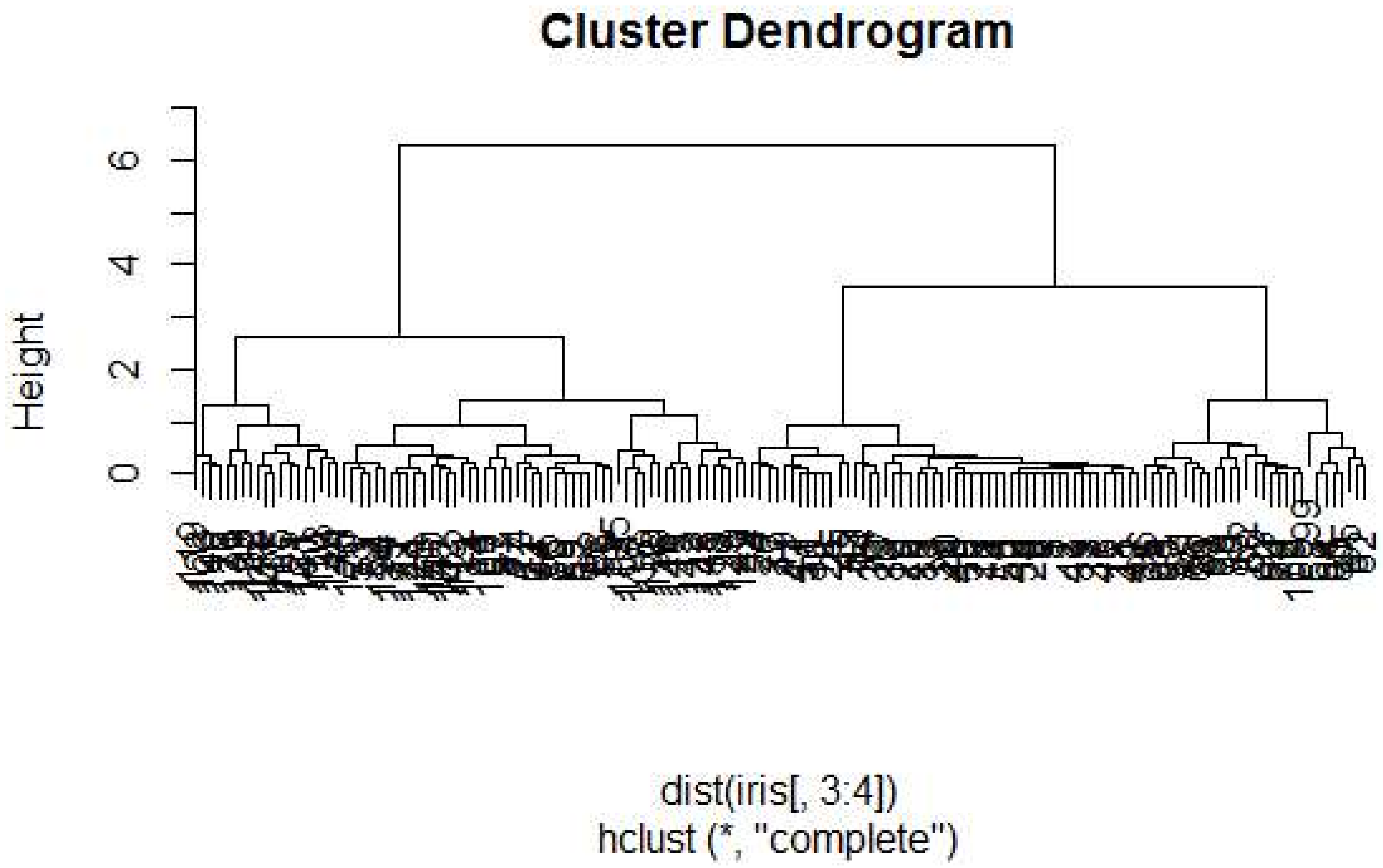
3. Hierarchical Clustering:

- Hierarchical Clustering builds clusters within clusters, and does not require a prespecified number of cluster like K-means and K-medoids do.
- A Hierarchical Clustering can be thought of as a tree and displayed as a dendogram, at the top there is just one cluster consisting of all the observations, and at the bottom each observation is an entire cluster.
- In between are varying levels of clustering.
- Hierarchical clustering also works on categorical data like the country information data. However, its dissimilarity matrix must be calculated differently.
- There are a number of ways to compute the distance between clusters and they can have a significant impact on the results of a hierarchical clustering.

Code:

```
>clusters<-hclust(dist(iris[,3:4]))

>clusters

Call:

hclust(d = dist(iris[, 3:4]))
```

```
Cluster method   : complete
Distance         : euclidean
Number of objects: 150
>plot(clusters)
```


Cluster Dendrogram
Height
dist(iris[, 3:4])
hclust (*, "complete")