Data Mining with R

วิชา การค้นพบองค์ความรู้และการทำเหมืองข้อมูลชั้นสูง

Veerasak Kritsanapraphan

Chulalongkorn University

Email : veerasak.kr568@cbs.chula.ac.th

@veerasakk

Slide and Sample Data

https://github.com/vkrit/chula_datamining

Agenda

- Overview and Data Visualization
- Data Preparation
- Predictive Data Mining
    - Decision Tree
    - K-Nearest Neighbor
    - Naive Bayes Classifier
    - Neural Network

Slide and Source Codes

https://github.com/vkrit/chula_datamining

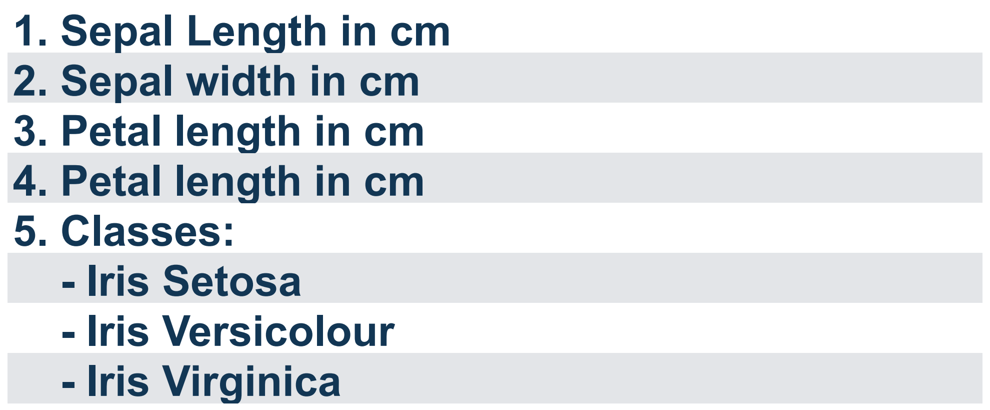
Overview

- Predictive Data Mining
  - Two Phases of Processing
    - Training Phase : Learn a model from training data
    - Predicting Phase : Deploy the model to production and use that to predict the future outcome

Data

- Iris Data Set from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris)

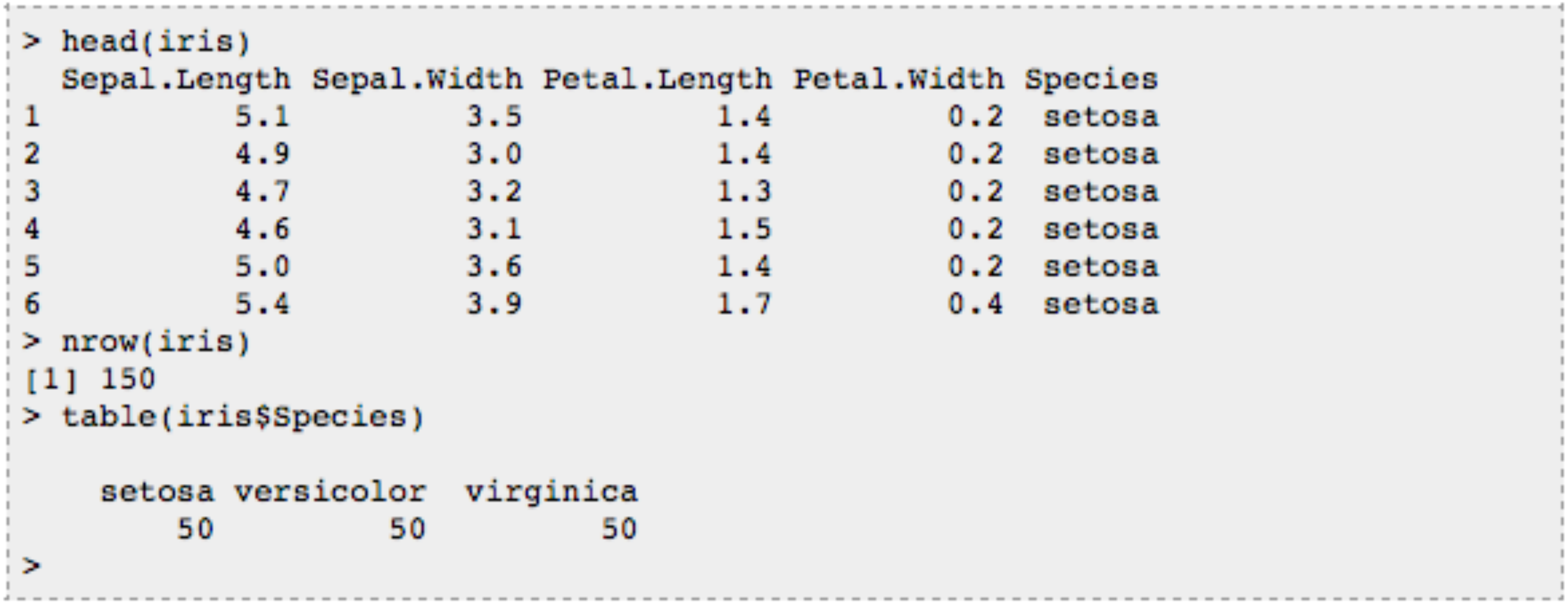
Attribute Information:



1. Sepal Length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal length in cm
5. Classes:
   - Iris Setosa
   - Iris Versicolour
   - Iris Virginica

Getting Data

> iris <- read.csv("iris.data.csv", header=TRUE)



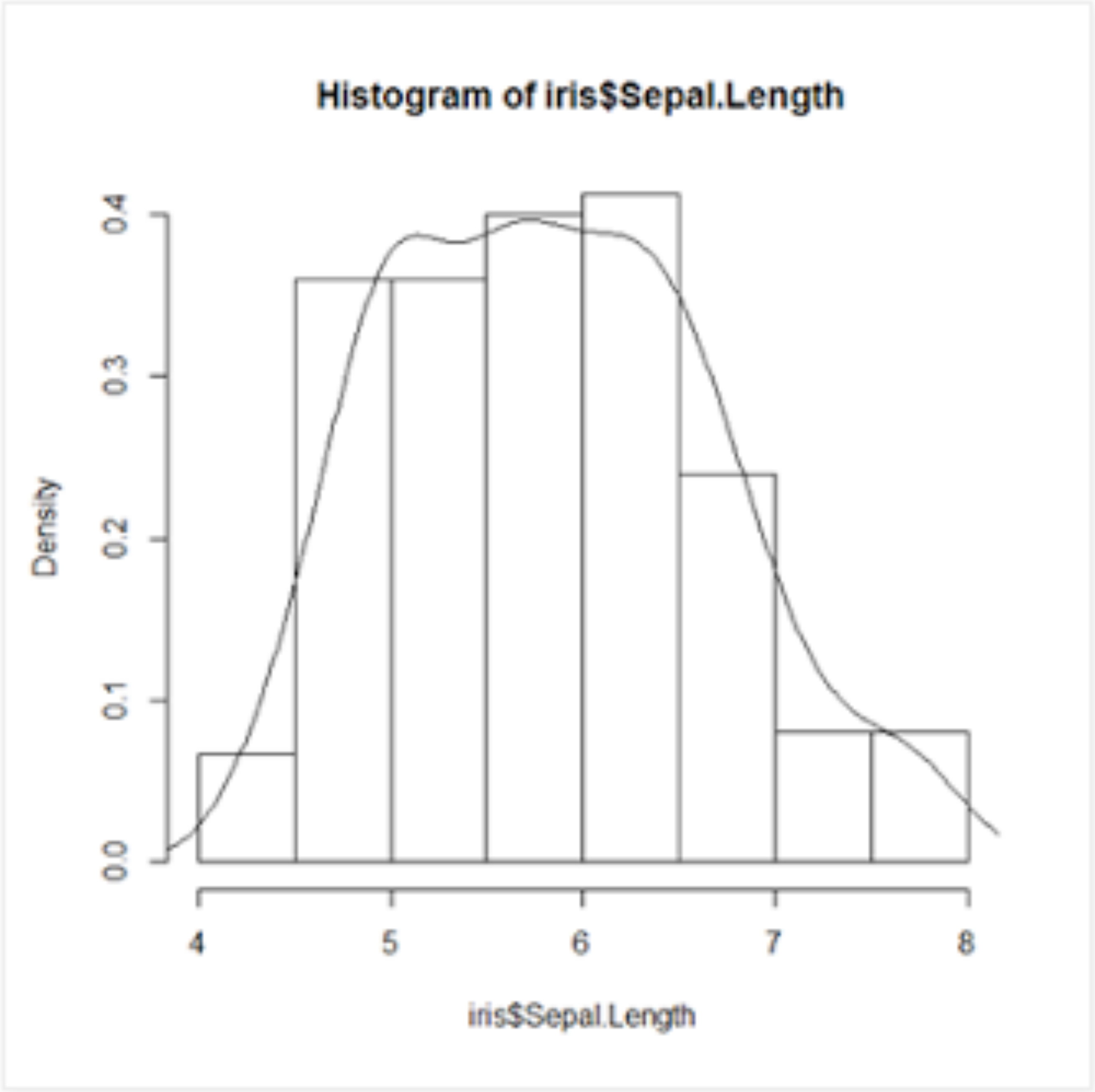
```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> nrow(iris)
[1] 150
> table(iris$Species)

    setosa versicolor  virginica 
        50         50         50 
>
```

Data Visualization

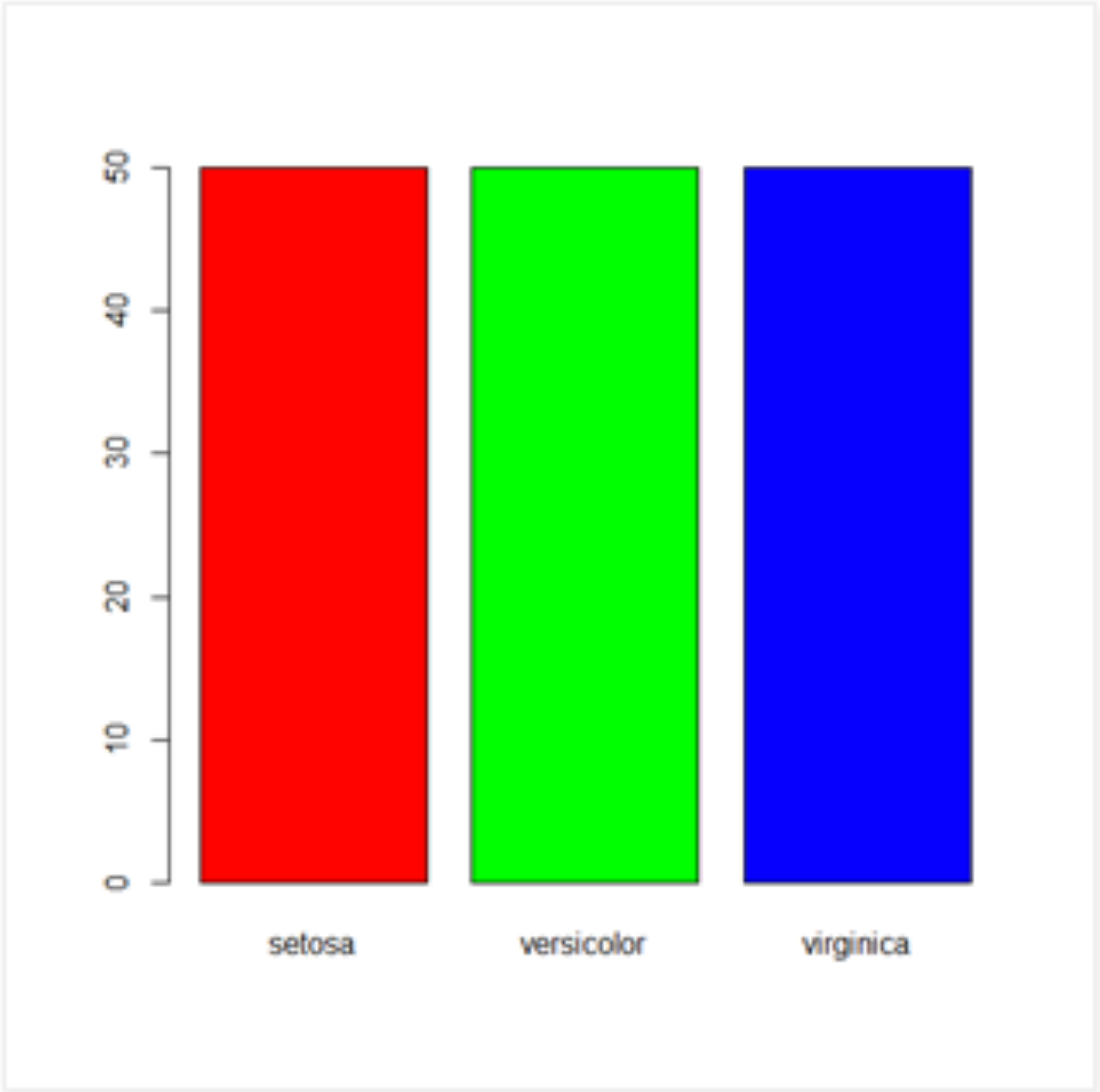
- Visualizing existing data is a very useful way to come up with ideas about what features should be included.

- "Dataframe" in R is a common way where data samples are organized in a tabular structure.

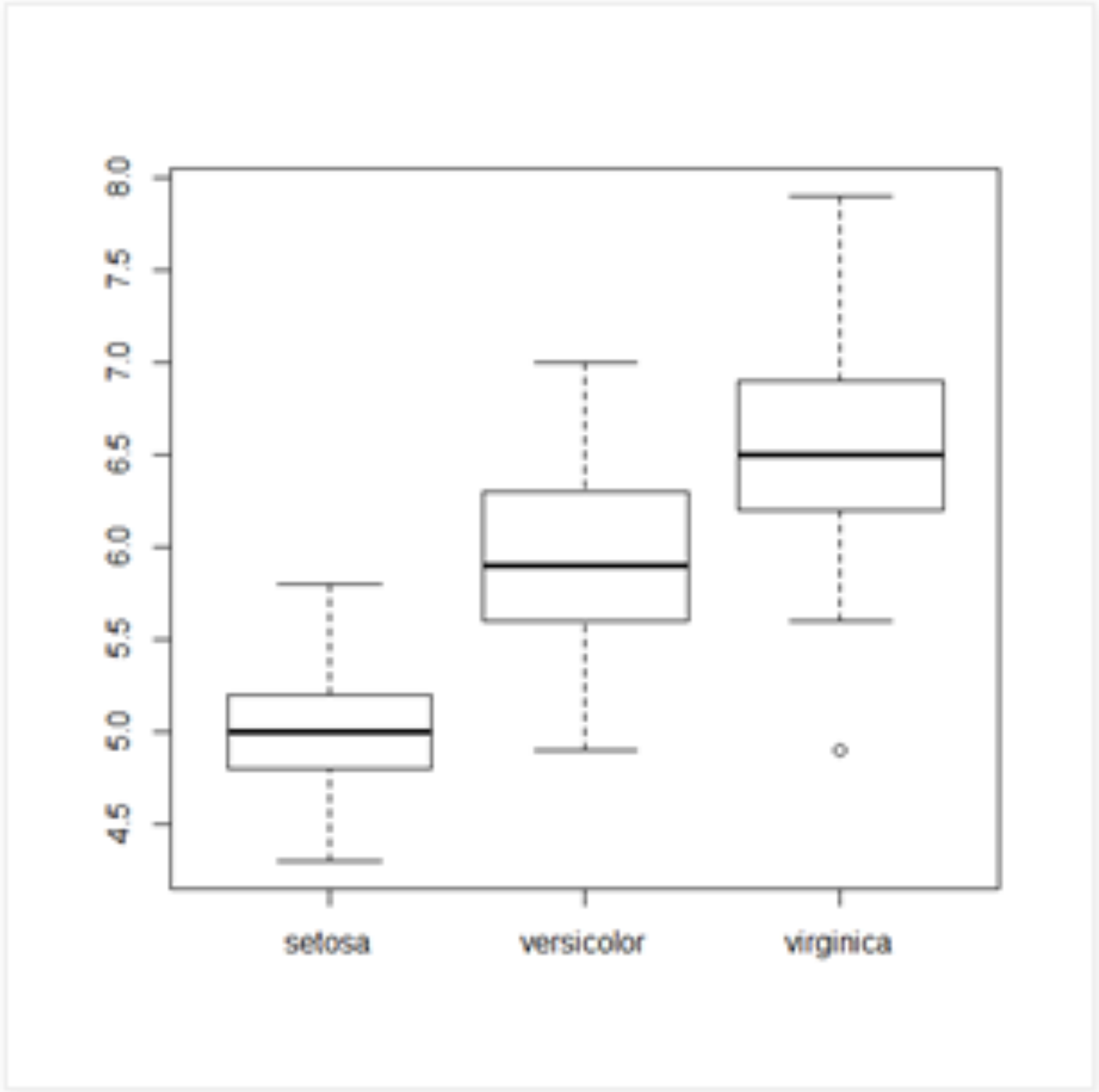
```
> # Plot the histogram
> hist(iris$Sepal.Length, breaks=10, prob=T)
> # Plot the density curve
> lines(density(iris$Sepal.Length))
>
```



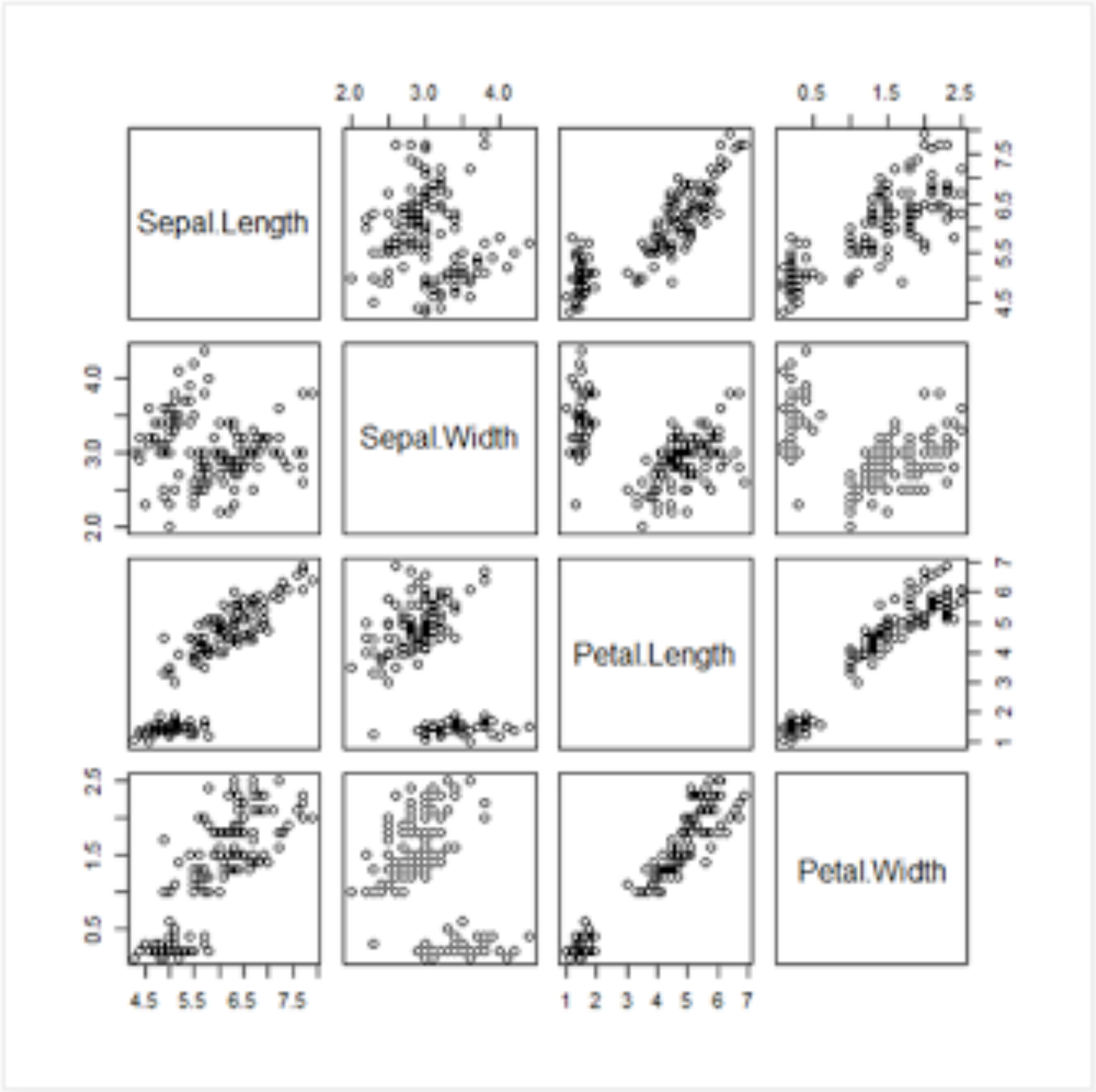
```
> categories <- table(iris$Species)
> barplot(categories, col=c('red', 'green', 'blue'))
>
```



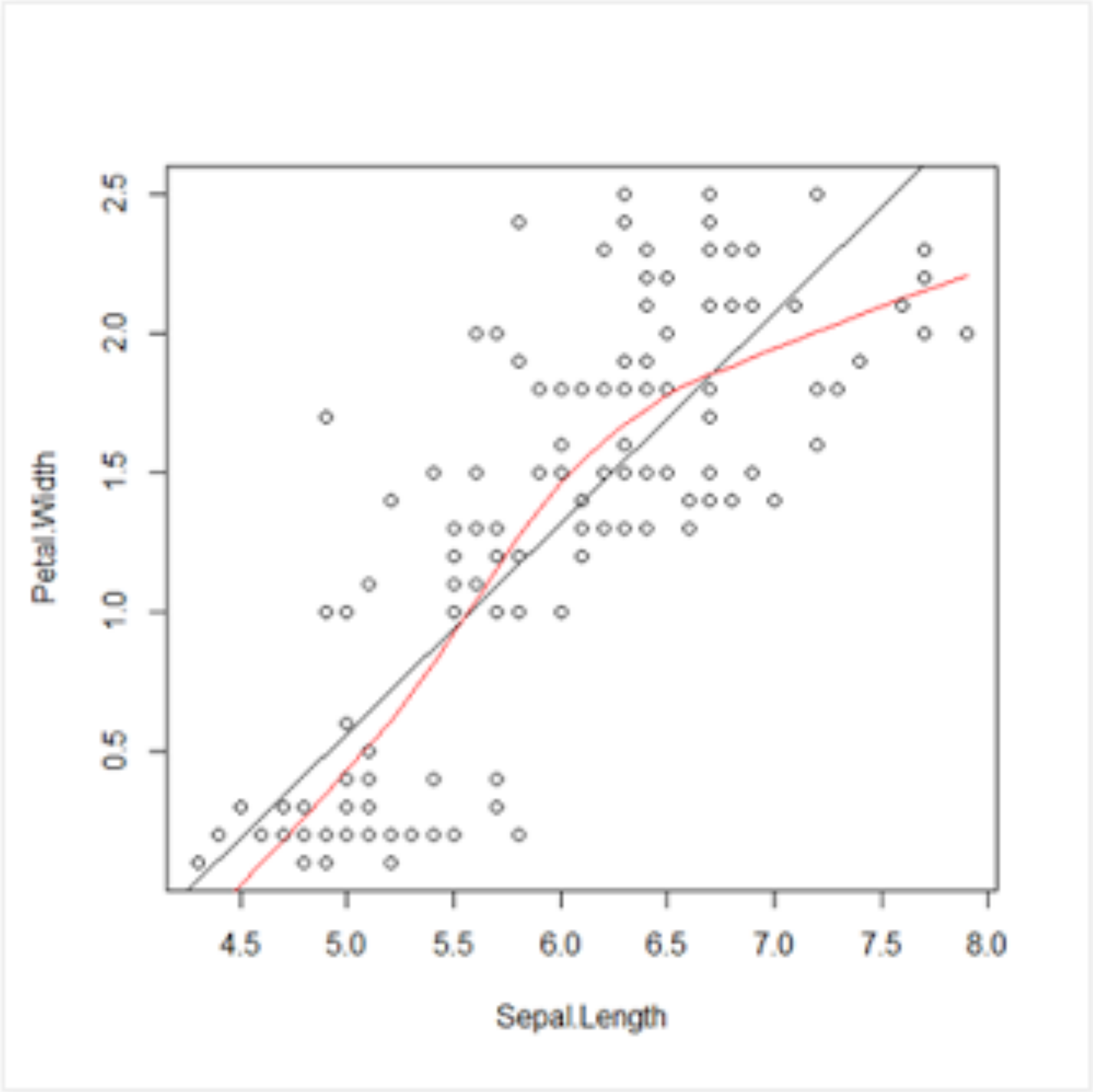
```
> boxplot(Sepal.Length~Species, data=iris)
>
```



```
> # Scatter plot for all pairs
> pairs(iris[,c(1,2,3,4)])
> # Compute the correlation matrix
> cor(iris[,c(1,2,3,4)])
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1170695    0.8716902   0.8179410
Sepal.Width    -0.1170695   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8716902  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179410  -0.3661259    0.9628654   1.0000000
>
```



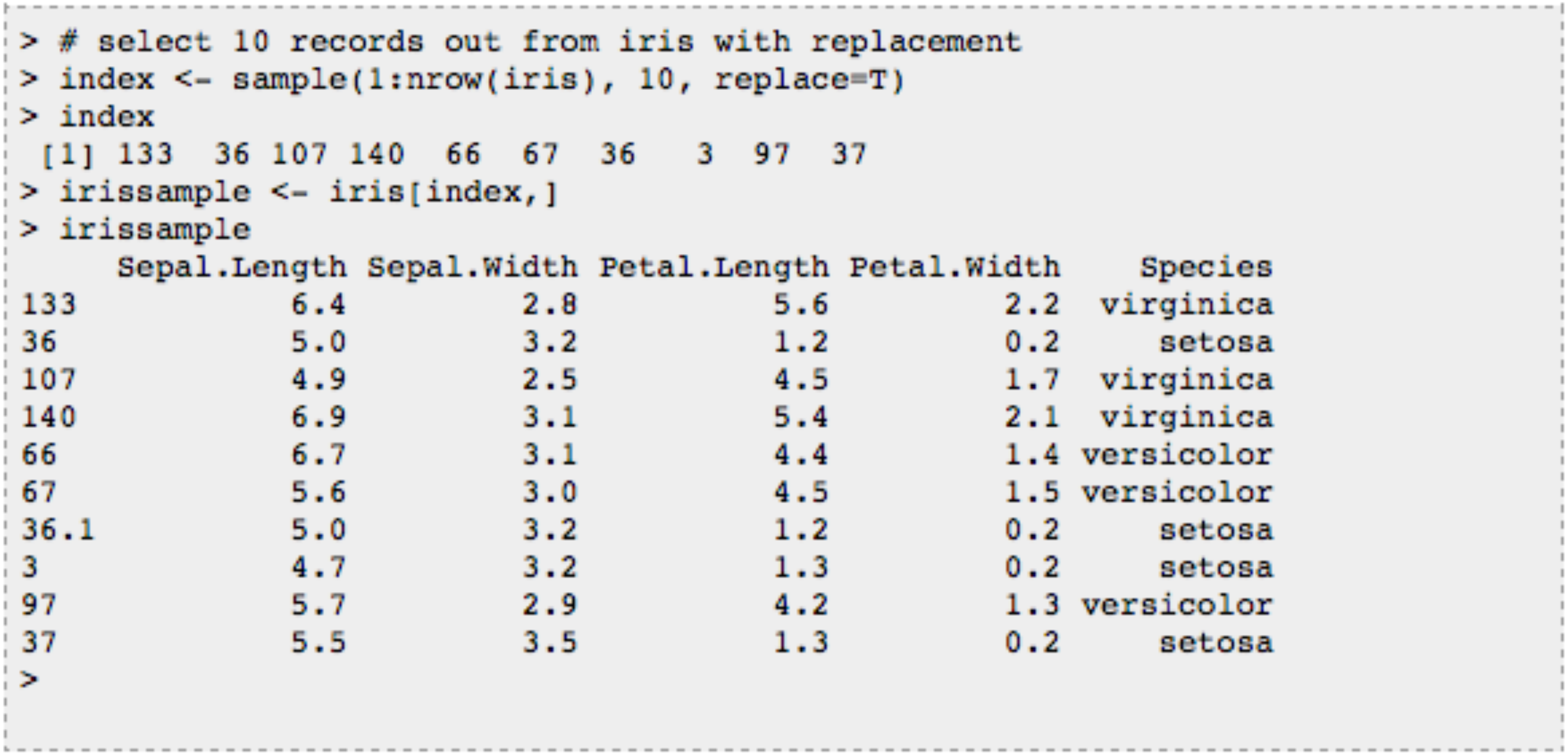
```
> # First plot the 2 variables
> plot(Petal.Width~Sepal.Length, data=iris)
> # Learn the regression model
> model <- lm(Petal.Width~Sepal.Length, data=iris)
> # Plot the regression line
> abline(model)
> # Now learn the local linear model
> model2 <- lowess(iris$Petal.Width~iris$Sepal.Length)
> lines(model2, col="red")
>
```



Preparing Training Data

- At this step, the purpose is to transform the raw data in a form that can fit into the data mining model.
    - Data sampling
    - Data validation and handle missing data
    - Normalize numeric value into a uniform range
    - Compute aggregated value (a special case is to compute frequency counts)
    - Expand categorical field to binary fields
    - Discretize numeric value into categories
    - Create derived fields from existing fields
    - Reduce dimensionality
    - Power and Log transformation

Data Sampling

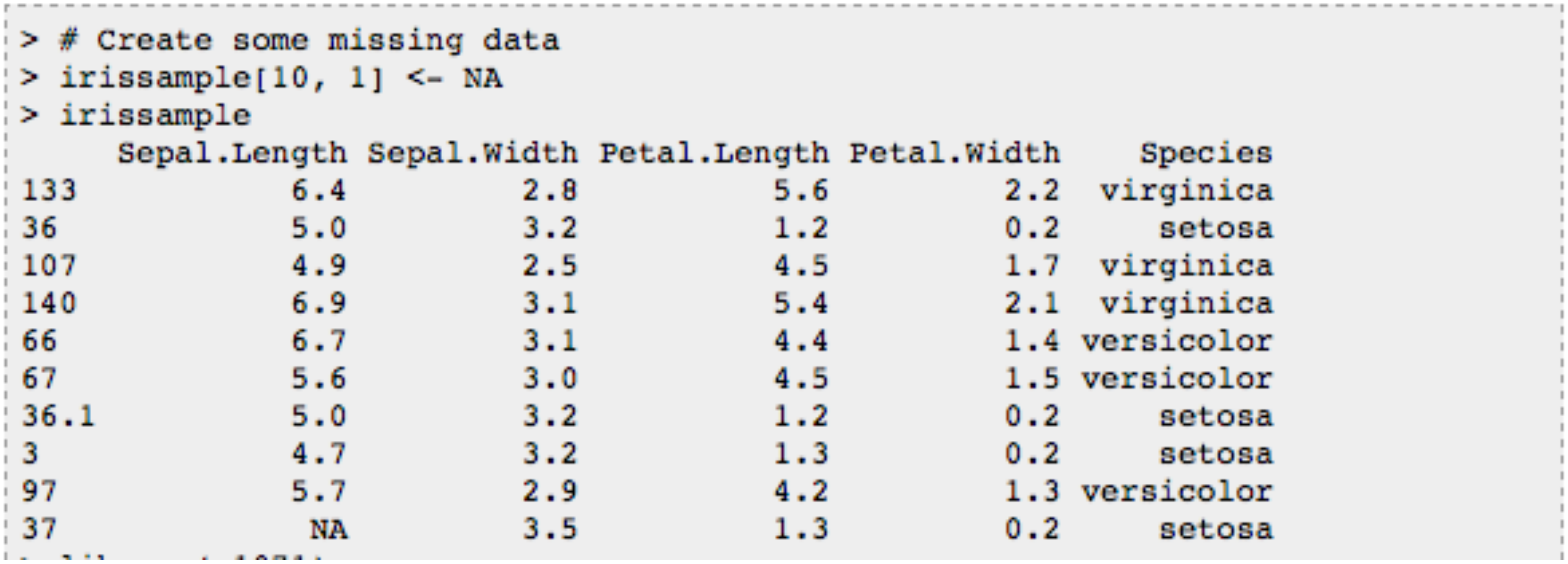


```
> # select 10 records out from iris with replacement
> index <- sample(1:nrow(iris), 10, replace=T)
> index
 [1] 133  36 107 140  66  67  36   3  97  37
> irissample <- iris[index,]
> irissample
     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
133           6.4         2.8          5.6         2.2  virginica
36            5.0         3.2          1.2         0.2     setosa
107           4.9         2.5          4.5         1.7  virginica
140           6.9         3.1          5.4         2.1  virginica
66            6.7         3.1          4.4         1.4 versicolor
67            5.6         3.0          4.5         1.5 versicolor
36.1          5.0         3.2          1.2         0.2     setosa
3             4.7         3.2          1.3         0.2     setosa
97            5.7         2.9          4.2         1.3 versicolor
37            5.5         3.5          1.3         0.2     setosa
>
```

Impute missing data

- Discard the whole record

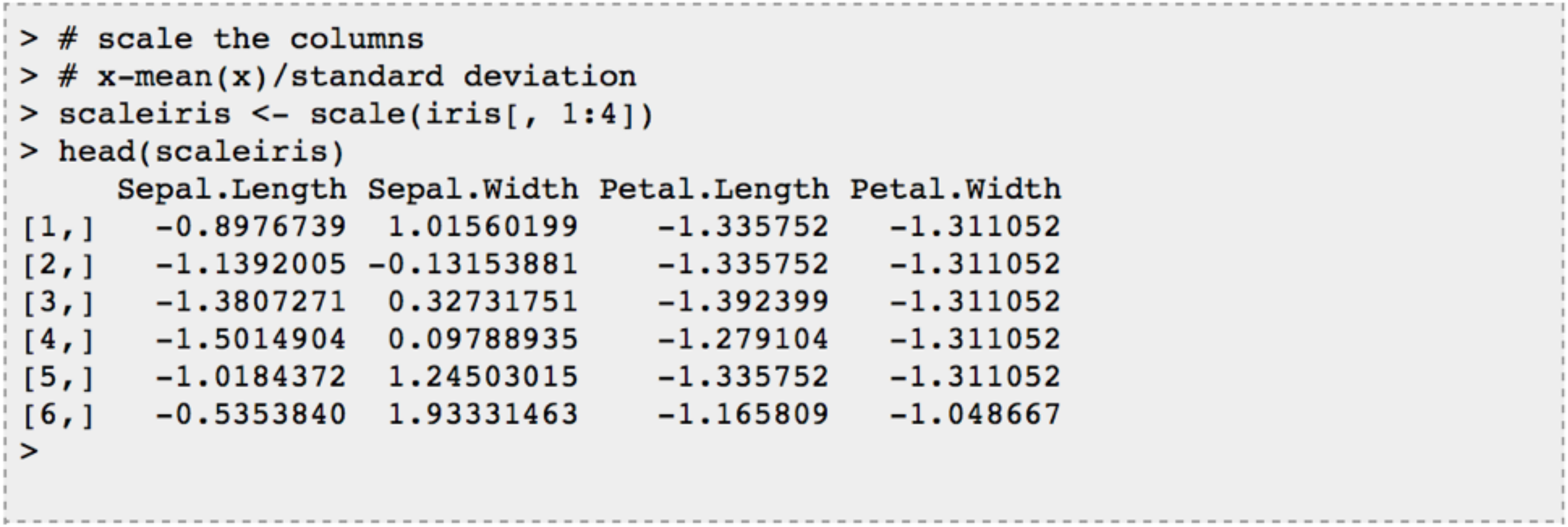
- Infer missing value based on the data of other record. Approach is to fill the missing data with the average or the median.



```
> # Create some missing data
> irissample[10, 1] <- NA
> irissample
     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
133           6.4         2.8          5.6         2.2  virginica
36            5.0         3.2          1.2         0.2     setosa
107           4.9         2.5          4.5         1.7  virginica
140           6.9         3.1          5.4         2.1  virginica
66            6.7         3.1          4.4         1.4 versicolor
67            5.6         3.0          4.5         1.5 versicolor
36.1          5.0         3.2          1.2         0.2     setosa
3             4.7         3.2          1.3         0.2     setosa
97            5.7         2.9          4.2         1.3 versicolor
37             NA         3.5          1.3         0.2     setosa
```

```
> library(e1071)
Loading required package: class
Warning message:
package ‘e1071’ was built under R version 2.14.2
> fixIris1 <- impute(irissample[,1:4], what='mean')
> fixIris1
     Sepal.Length Sepal.Width Petal.Length Petal.Width
133      6.400000         2.8          5.6         2.2
36       5.000000         3.2          1.2         0.2
107      4.900000         2.5          4.5         1.7
140      6.900000         3.1          5.4         2.1
66       6.700000         3.1          4.4         1.4
67       5.600000         3.0          4.5         1.5
36.1     5.000000         3.2          1.2         0.2
3        4.700000         3.2          1.3         0.2
97       5.700000         2.9          4.2         1.3
37       5.655556         3.5          1.3         0.2
> fixIris2 <- impute(irissample[,1:4], what='median')
> fixIris2
     Sepal.Length Sepal.Width Petal.Length Petal.Width
133           6.4         2.8          5.6         2.2
36            5.0         3.2          1.2         0.2
107           4.9         2.5          4.5         1.7
140           6.9         3.1          5.4         2.1
66            6.7         3.1          4.4         1.4
67            5.6         3.0          4.5         1.5
36.1          5.0         3.2          1.2         0.2
3             4.7         3.2          1.3         0.2
97            5.7         2.9          4.2         1.3
37            5.6         3.5          1.3         0.2
>
```

Normalize numeric value



```
> # scale the columns
> # x-mean(x)/standard deviation
> scaleiris <- scale(iris[, 1:4])
> head(scaleiris)
     Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,]   -0.8976739  1.01560199    -1.335752   -1.311052
[2,]   -1.1392005 -0.13153881    -1.335752   -1.311052
[3,]   -1.3807271  0.32731751    -1.392399   -1.311052
[4,]   -1.5014904  0.09788935    -1.279104   -1.311052
[5,]   -1.0184372  1.24503015    -1.335752   -1.311052
[6,]   -0.5353840  1.93331463    -1.165809   -1.048667
>
```

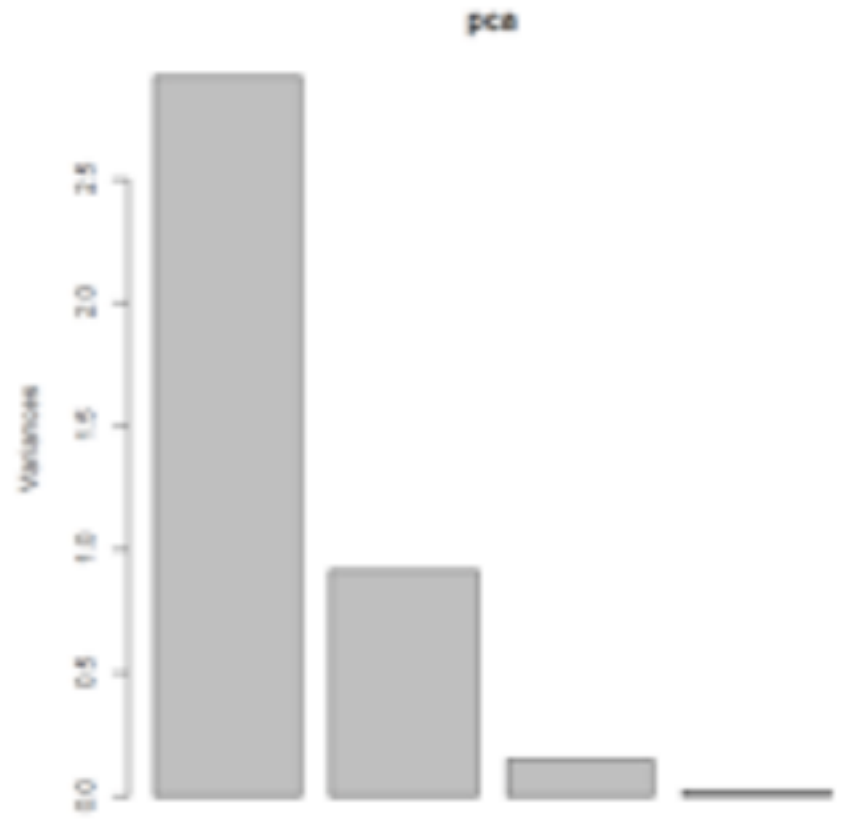
Reduce dimensionality

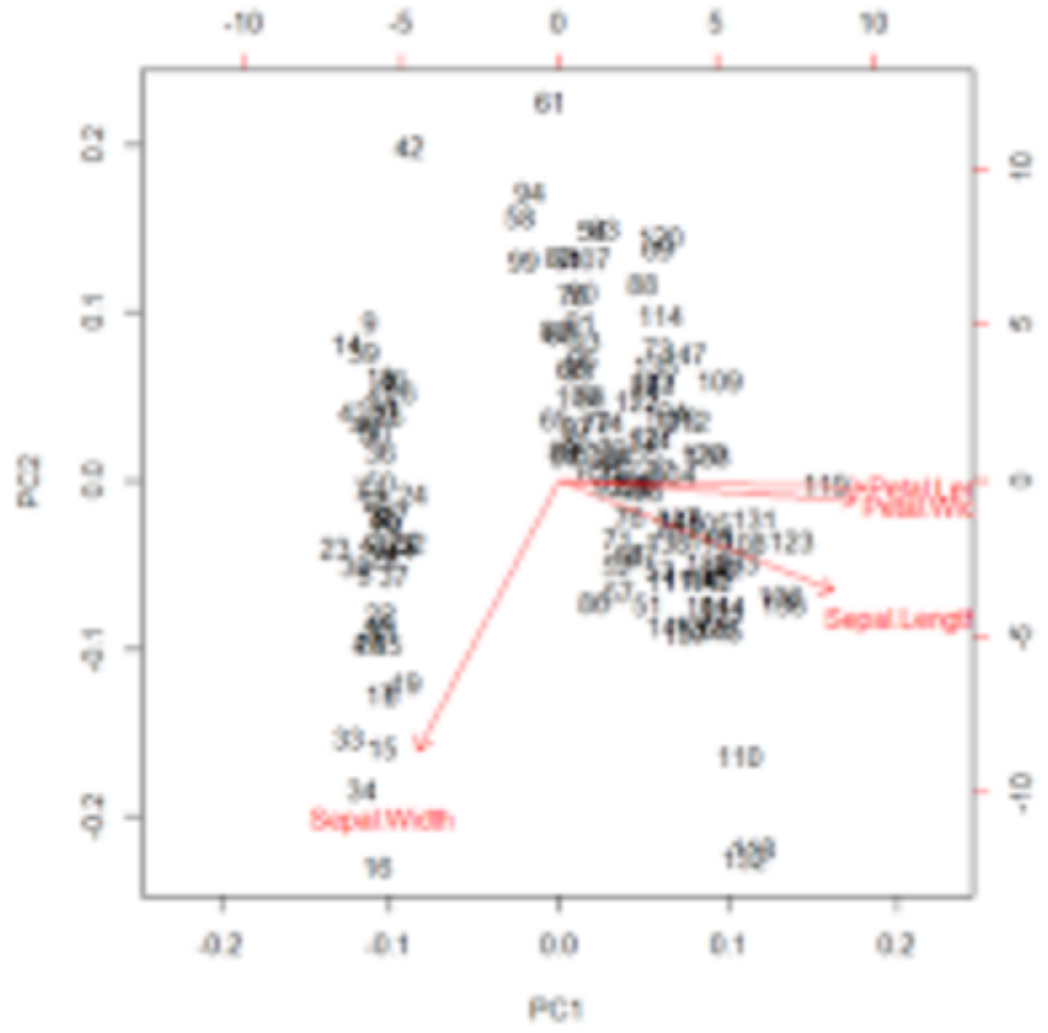
There are two ways to reduce the number of input attributes.

1. Removing irrelevant input variables.
2. Removing redundant input variables.

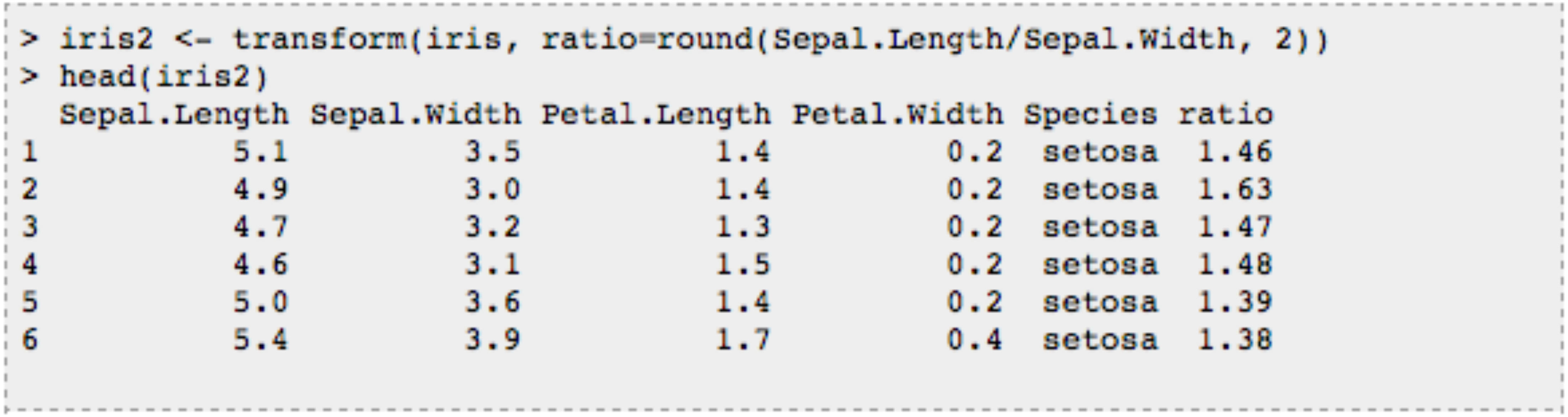
```
> # Use iris data set
> cor(iris[, -5])
              Sepal.Length  Sepal.Width Petal.Length  Petal.Width
Sepal.Length  1.0000000000 -0.1175697841  0.8717537759  0.8179411263
Sepal.Width  -0.1175697841  1.0000000000 -0.4284401043 -0.3661259325
Petal.Length  0.8717537759 -0.4284401043  1.0000000000  0.9628654314
Petal.Width   0.8179411263 -0.3661259325  0.9628654314  1.0000000000
> # Some attributes shows high correlation, compute PCA
> pca <- prcomp(iris[,-5], scale=T)
> summary(pca)
Importance of components:
                            PC1       PC2       PC3       PC4
Standard deviation     1.708361 0.9560494 0.3830886 0.1439265
Proportion of Variance 0.729620 0.2285100 0.0366900 0.0051800
Cumulative Proportion  0.729620 0.9581300 0.9948200 1.0000000
> # Notice PC1 and PC2 covers most variation
> plot(pca)
> pca$rotation
                       PC1            PC2           PC3           PC4
Sepal.Length  0.5210659147 -0.37741761556  0.7195663527  0.2612862800
Sepal.Width  -0.2693474425 -0.92329565954 -0.2443817795 -0.1235096196
Petal.Length  0.5804130958 -0.02449160909 -0.1421263693 -0.8014492463
Petal.Width   0.5648565358 -0.06694198697 -0.6342727371  0.5235971346
> # Project first 2 records in PCA direction
> predict(pca)[1:2,]
              PC1          PC2          PC3           PC4
[1,] -2.257141176 -0.4784238321 0.1272796237 0.02408750846
[2,] -2.074013015  0.6718826870 0.2338255167 0.10266284468
> # plot all points in top 2 PCA direction
> biplot(pca)
```

Pin it


pca
Variances
0.0
0.5
1.0
1.5
2.0
2.5

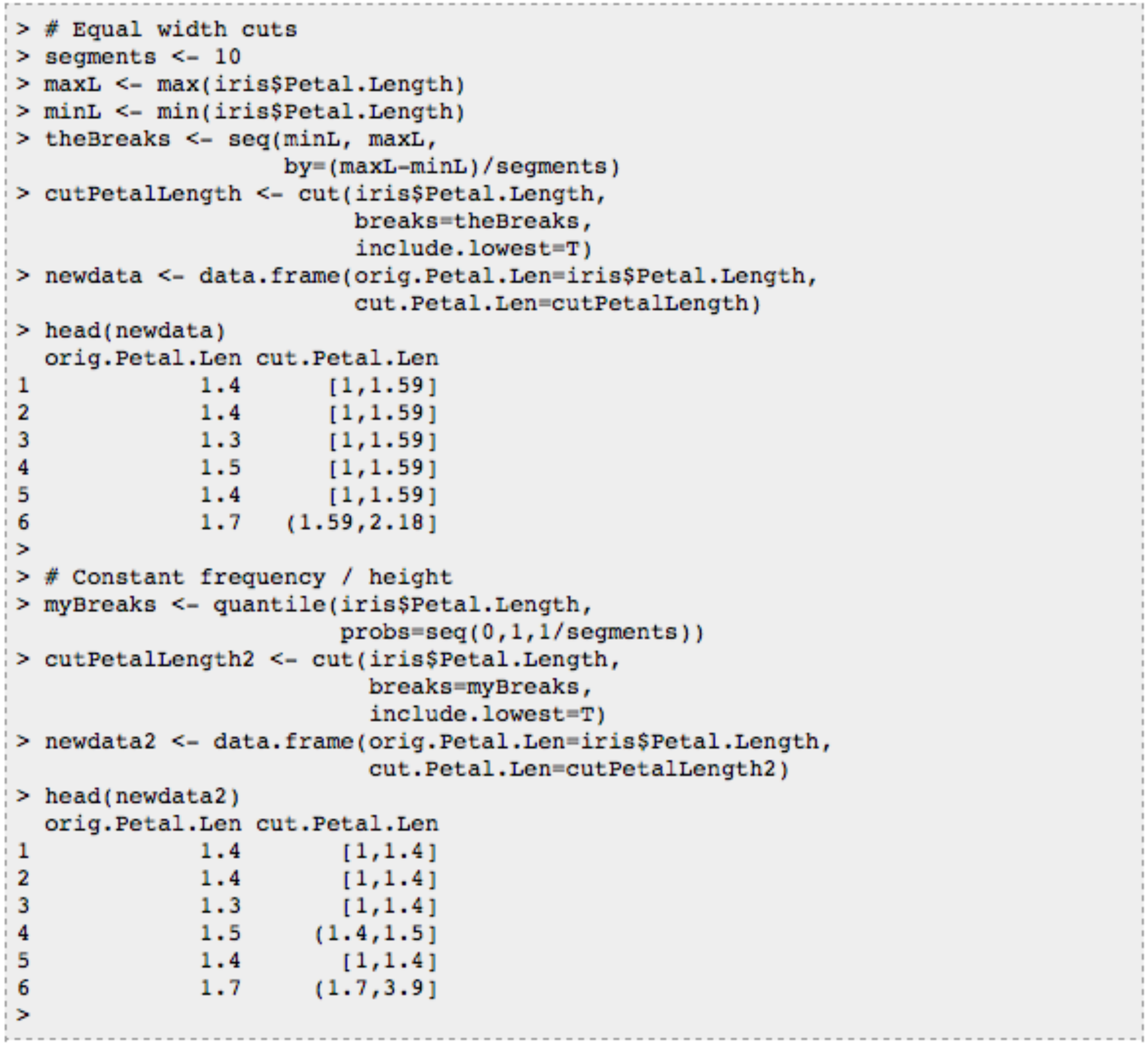

PC1
PC2
Sepal.Width
Sepal.Length
Petal.Length
Petal.Width

Add derived attributes



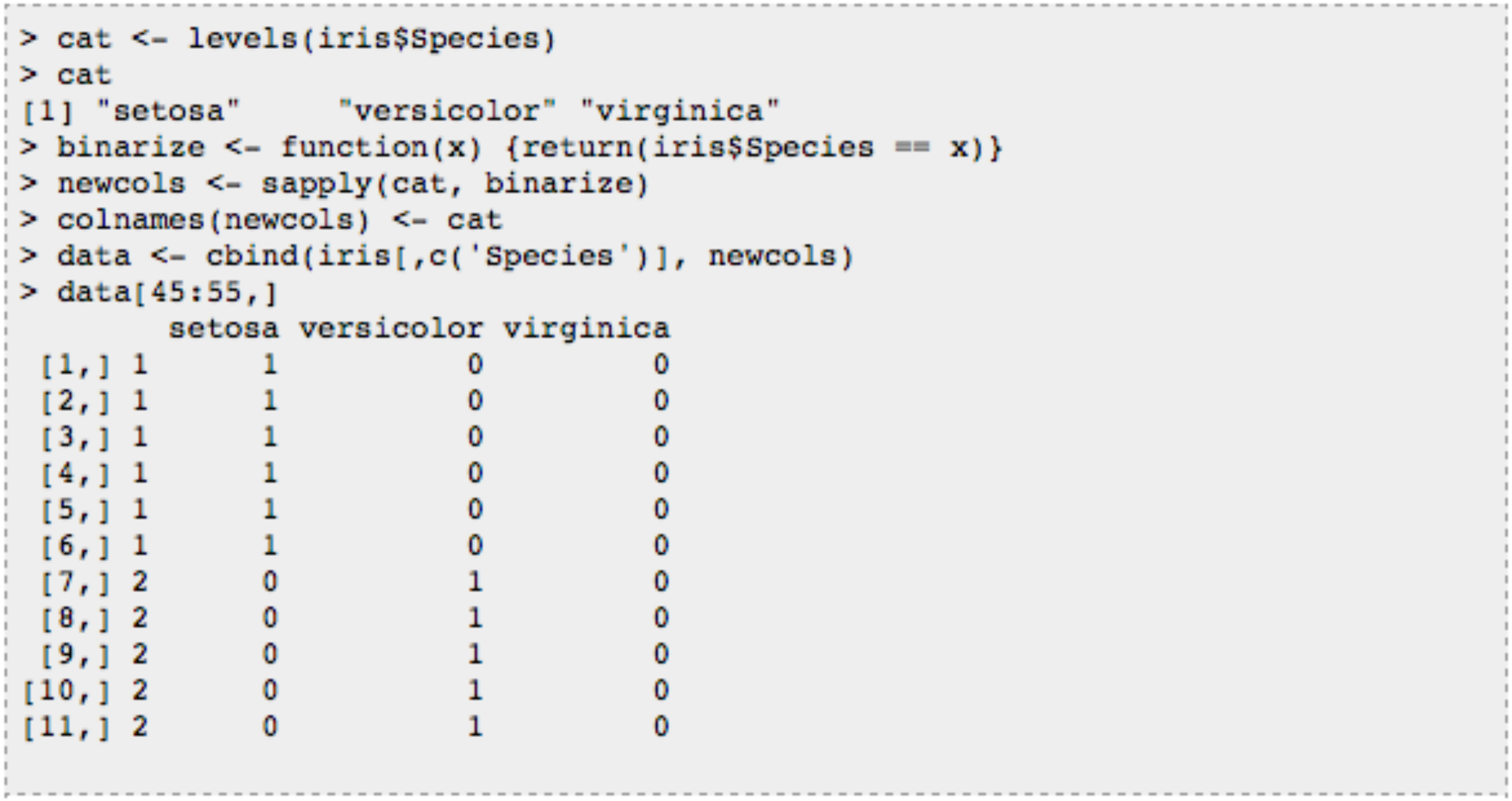
```
> iris2 <- transform(iris, ratio=round(Sepal.Length/Sepal.Width, 2))
> head(iris2)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species ratio
1          5.1         3.5          1.4         0.2  setosa  1.46
2          4.9         3.0          1.4         0.2  setosa  1.63
3          4.7         3.2          1.3         0.2  setosa  1.47
4          4.6         3.1          1.5         0.2  setosa  1.48
5          5.0         3.6          1.4         0.2  setosa  1.39
6          5.4         3.9          1.7         0.4  setosa  1.38
```

Discretize numeric value into categories



```
> # Equal width cuts
> segments <- 10
> maxL <- max(iris$Petal.Length)
> minL <- min(iris$Petal.Length)
> theBreaks <- seq(minL, maxL,
                  by=(maxL-minL)/segments)
> cutPetalLength <- cut(iris$Petal.Length,
                       breaks=theBreaks,
                       include.lowest=T)
> newdata <- data.frame(orig.Petal.Len=iris$Petal.Length,
                       cut.Petal.Len=cutPetalLength)
> head(newdata)
  orig.Petal.Len cut.Petal.Len
1            1.4      [1,1.59]
2            1.4      [1,1.59]
3            1.3      [1,1.59]
4            1.5      [1,1.59]
5            1.4      [1,1.59]
6            1.7   (1.59,2.18]
>
> # Constant frequency / height
> myBreaks <- quantile(iris$Petal.Length,
                      probs=seq(0,1,1/segments))
> cutPetalLength2 <- cut(iris$Petal.Length,
                        breaks=myBreaks,
                        include.lowest=T)
> newdata2 <- data.frame(orig.Petal.Len=iris$Petal.Length,
                        cut.Petal.Len=cutPetalLength2)
> head(newdata2)
  orig.Petal.Len cut.Petal.Len
1            1.4       [1,1.4]
2            1.4       [1,1.4]
3            1.3       [1,1.4]
4            1.5     (1.4,1.5]
5            1.4       [1,1.4]
6            1.7     (1.7,3.9]
>
```

Binarize categorical attributes



```
> cat <- levels(iris$Species)
> cat
[1] "setosa"     "versicolor" "virginica"
> binarize <- function(x) {return(iris$Species == x)}
> newcols <- sapply(cat, binarize)
> colnames(newcols) <- cat
> data <- cbind(iris[,c('Species')], newcols)
> data[45:55,]
         setosa versicolor virginica
 [1,] 1      1          0         0
 [2,] 1      1          0         0
 [3,] 1      1          0         0
 [4,] 1      1          0         0
 [5,] 1      1          0         0
 [6,] 1      1          0         0
 [7,] 2      0          1         0
 [8,] 2      0          1         0
 [9,] 2      0          1         0
[10,] 2      0          1         0
[11,] 2      0          1         0
```

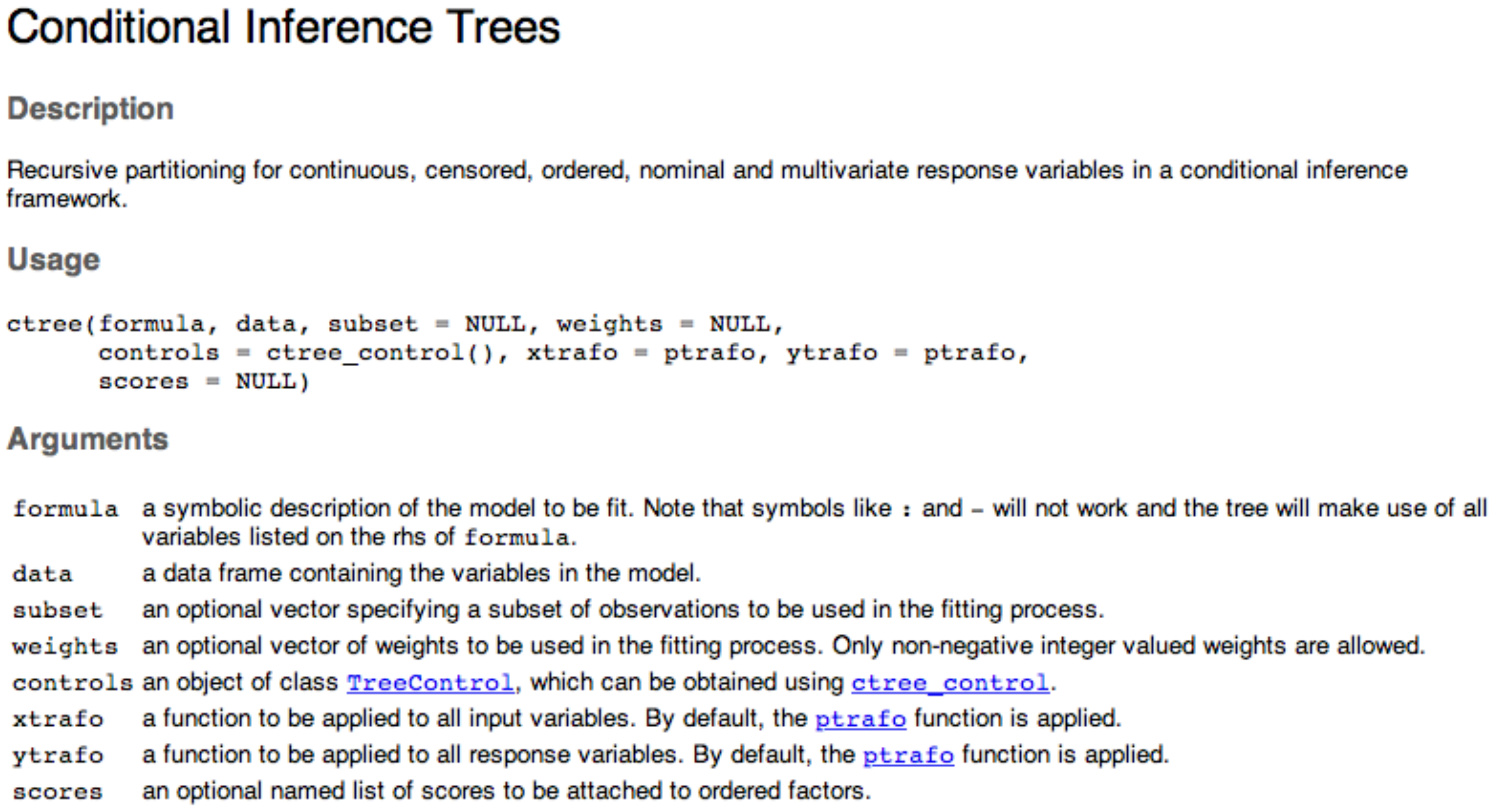
Data Mining

Techniques

Iris Data Preparation

```
> set.seed(1234)

> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))

> trainData <- iris[ind==1,]

> testData <- iris[ind==2,]
```

Decision Tree



Conditional Inference Trees

Description

Recursive partitioning for continuous, censored, ordered, nominal and multivariate response variables in a conditional inference framework.

Usage

```
ctree(formula, data, subset = NULL, weights = NULL,
      controls = ctree_control(), xtrafo = ptrafo, ytrafo = ptrafo,
      scores = NULL)
```

Arguments

`formula`	a symbolic description of the model to be fit. Note that symbols like `:` and `-` will not work and the tree will make use of all variables listed on the rhs of `formula`.
`data`	a data frame containing the variables in the model.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. Only non-negative integer valued weights are allowed.
`controls`	an object of class `TreeControl`, which can be obtained using `ctree_control`.
`xtrafo`	a function to be applied to all input variables. By default, the `ptrafo` function is applied.
`ytrafo`	a function to be applied to all response variables. By default, the `ptrafo` function is applied.
`scores`	an optional named list of scores to be attached to ordered factors.

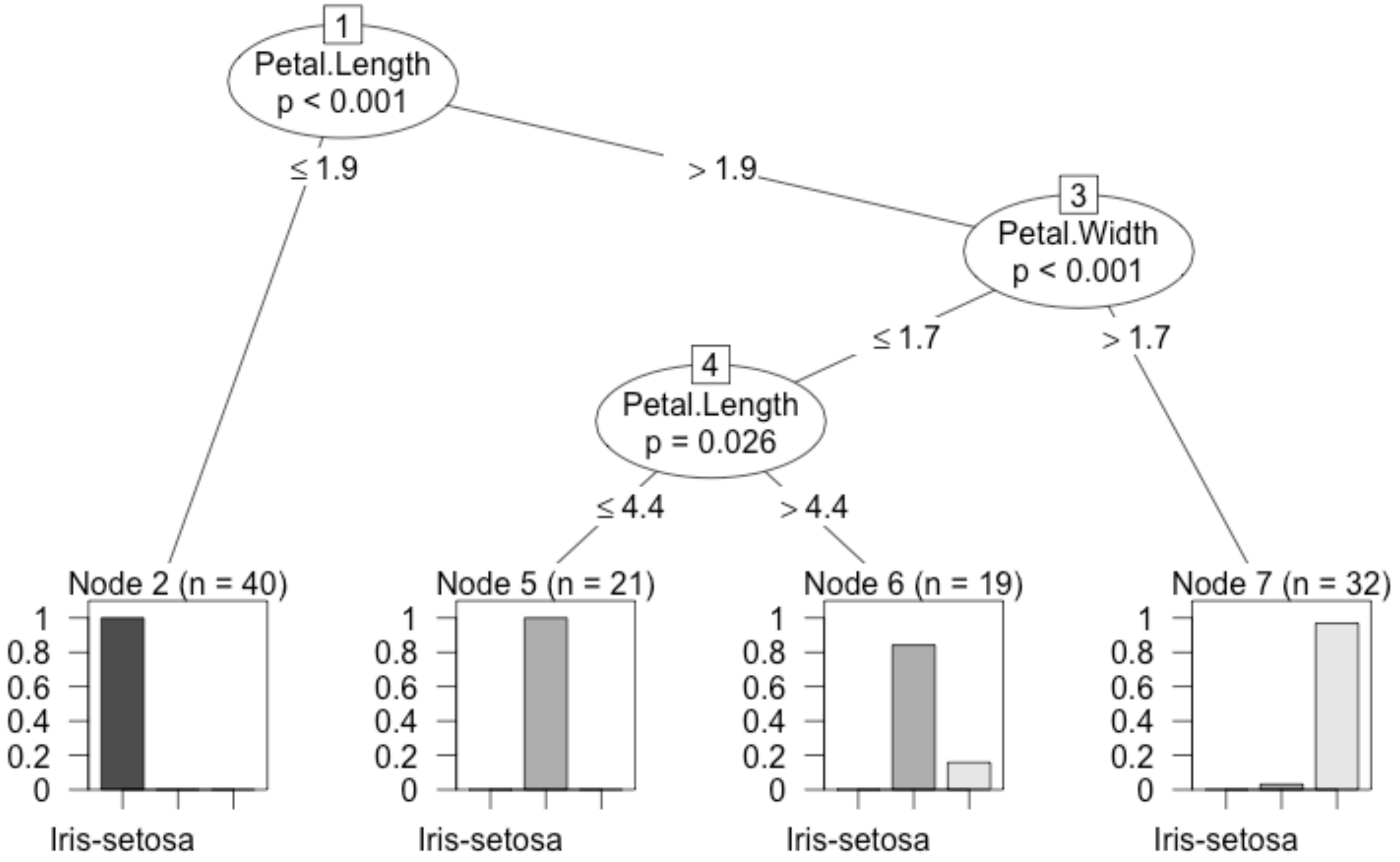
Decision Tree - Create Model

```
> library(party)
> myFormula <- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
> iris_ctree <- ctree(myFormula, data=trainData)
> # check the prediction
> table(predict(iris_ctree), trainData$Species)
```

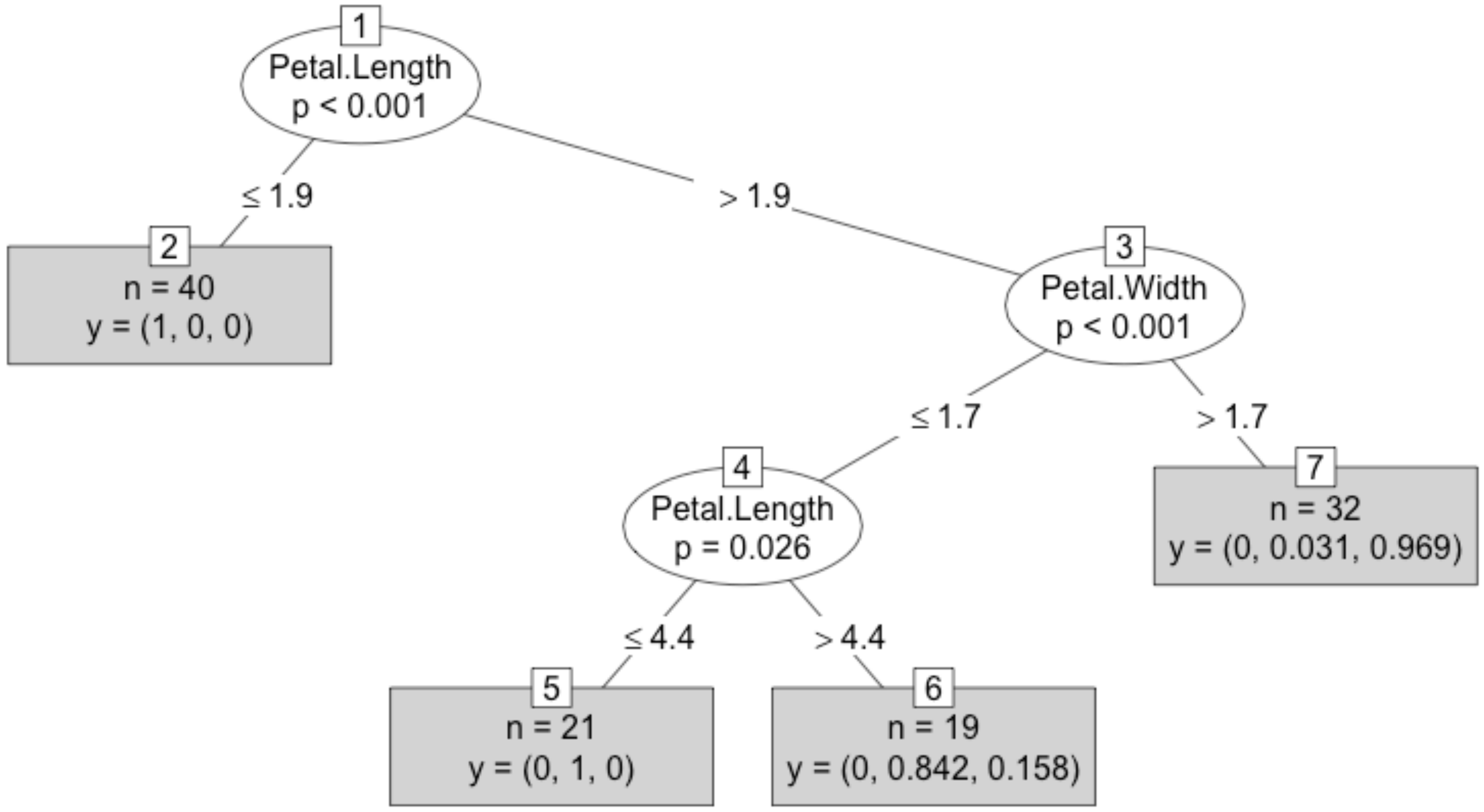
```
             setosa versicolor virginica
  setosa         40          0         0
  versicolor      0         37         3
  virginica       0          1        31
```

```
> print(iris_ctree)

         Conditional inference tree with 4 terminal nodes

Response:  Species
Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations:  112

1) Petal.Length <= 1.9; criterion = 1, statistic = 104.643
  2)*  weights = 40
1) Petal.Length > 1.9
  3) Petal.Width <= 1.7; criterion = 1, statistic = 48.939
    4) Petal.Length <= 4.4; criterion = 0.974, statistic = 7.397
      5)*  weights = 21
    4) Petal.Length > 4.4
      6)*  weights = 19
  3) Petal.Width > 1.7
    7)*  weights = 32
```

```
> plot(iris_ctree)
```



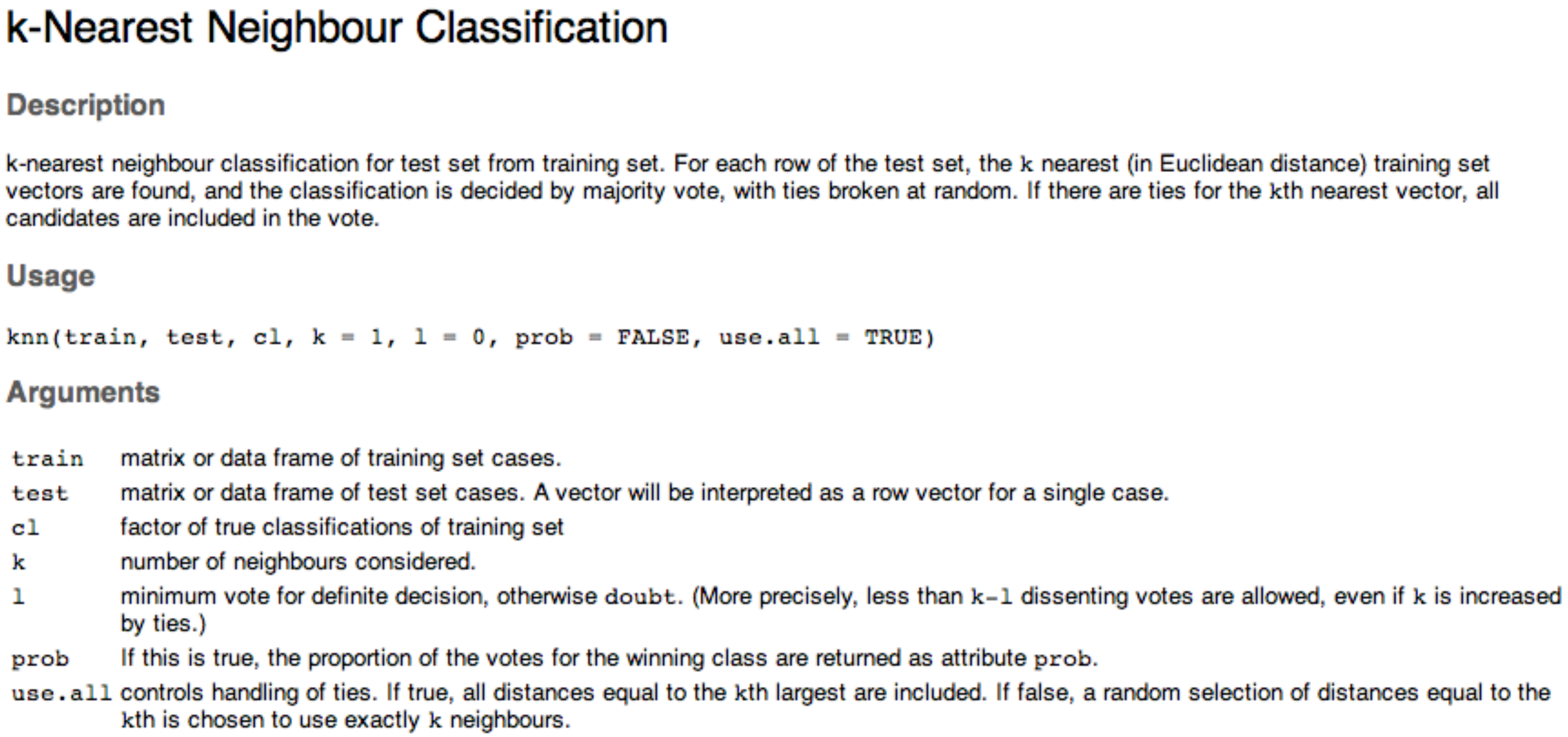
```
> plot(iris_ctree, type="simple")
```



Decision Tree - Prediction

```
> # predict on test data
> testPred <- predict(iris_ctree, newdata = testData)
> table(testPred, testData$Species)

testPred     setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         12         2
  virginica       0          0        14
```

K-NN



k-Nearest Neighbour Classification

Description

k-nearest neighbour classification for test set from training set. For each row of the test set, the `k` nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the `k`th nearest vector, all candidates are included in the vote.

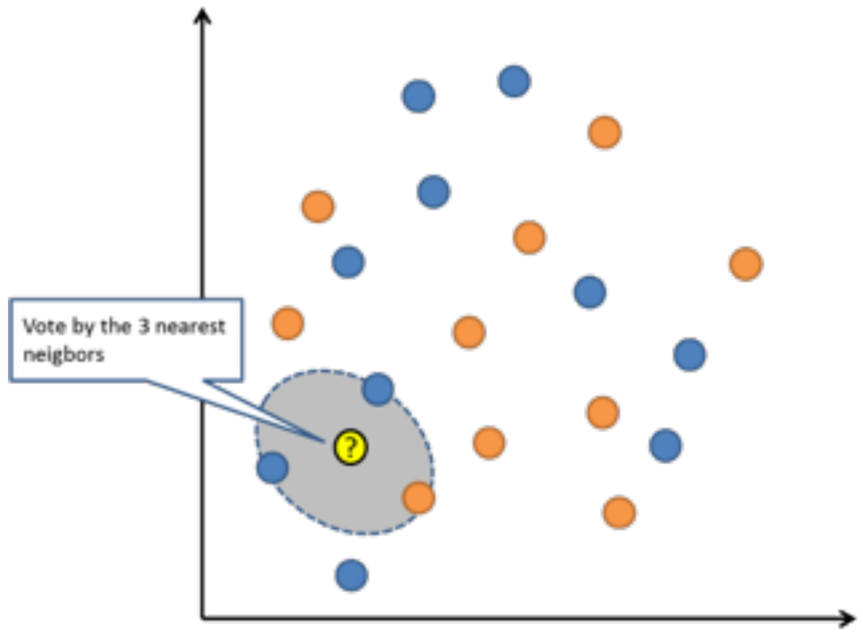
Usage

```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

Arguments

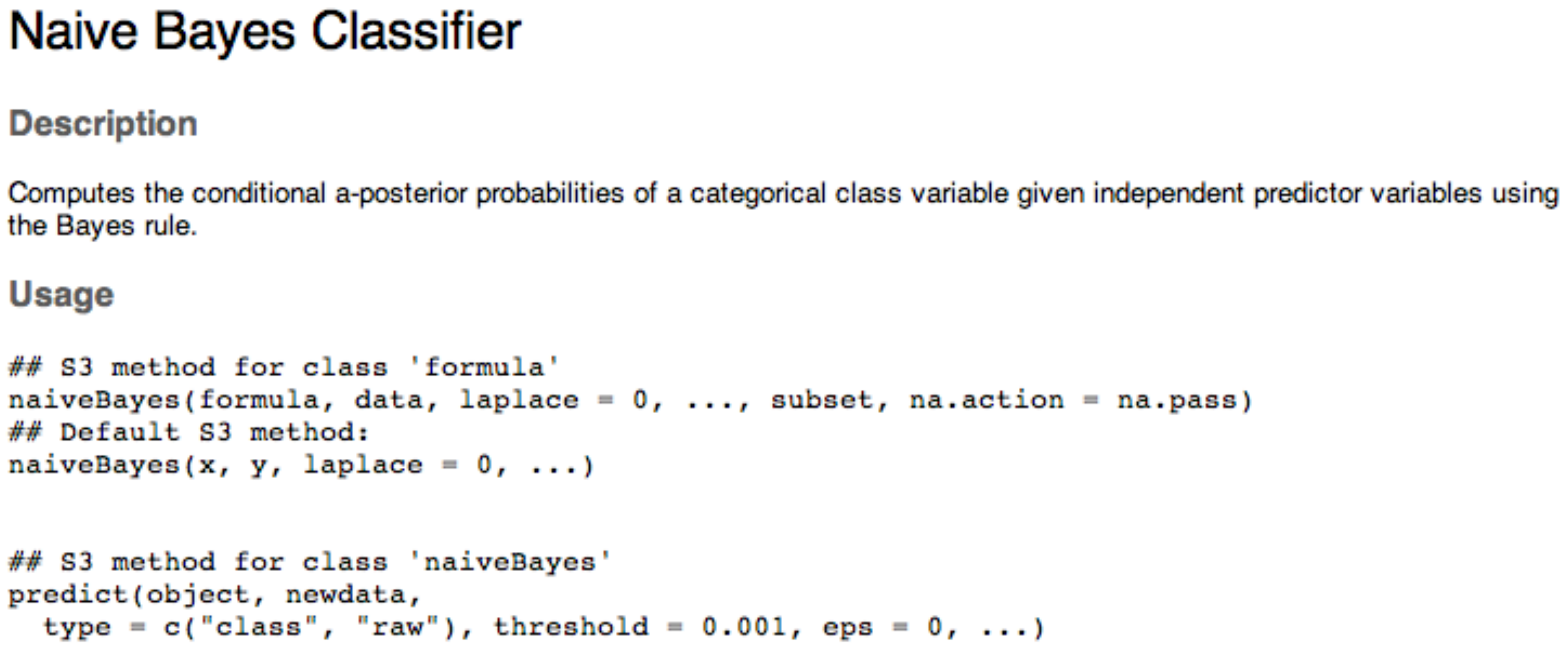
`train`	matrix or data frame of training set cases.
`test`	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
`cl`	factor of true classifications of training set
`k`	number of neighbours considered.
`l`	minimum vote for definite decision, otherwise `doubt`. (More precisely, less than `k-l` dissenting votes are allowed, even if `k` is increased by ties.)
`prob`	If this is true, the proportion of the votes for the winning class are returned as attribute `prob`.
`use.all`	controls handling of ties. If true, all distances equal to the `k`th largest are included. If false, a random selection of distances equal to the `k`th is chosen to use exactly `k` neighbours.

K-NN



```
> library(class)
> train_input <- as.matrix(iristrain[,-5])
> train_output <- as.vector(iristrain[,5])
> test_input <- as.matrix(iristest[,-5])
> prediction <- knn(train_input, test_input,
                    train_output, k=5)
> table(prediction, iristest$Species)

prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         1
  virginica       0          0         9
>
```

Naive Bayes



Naive Bayes Classifier

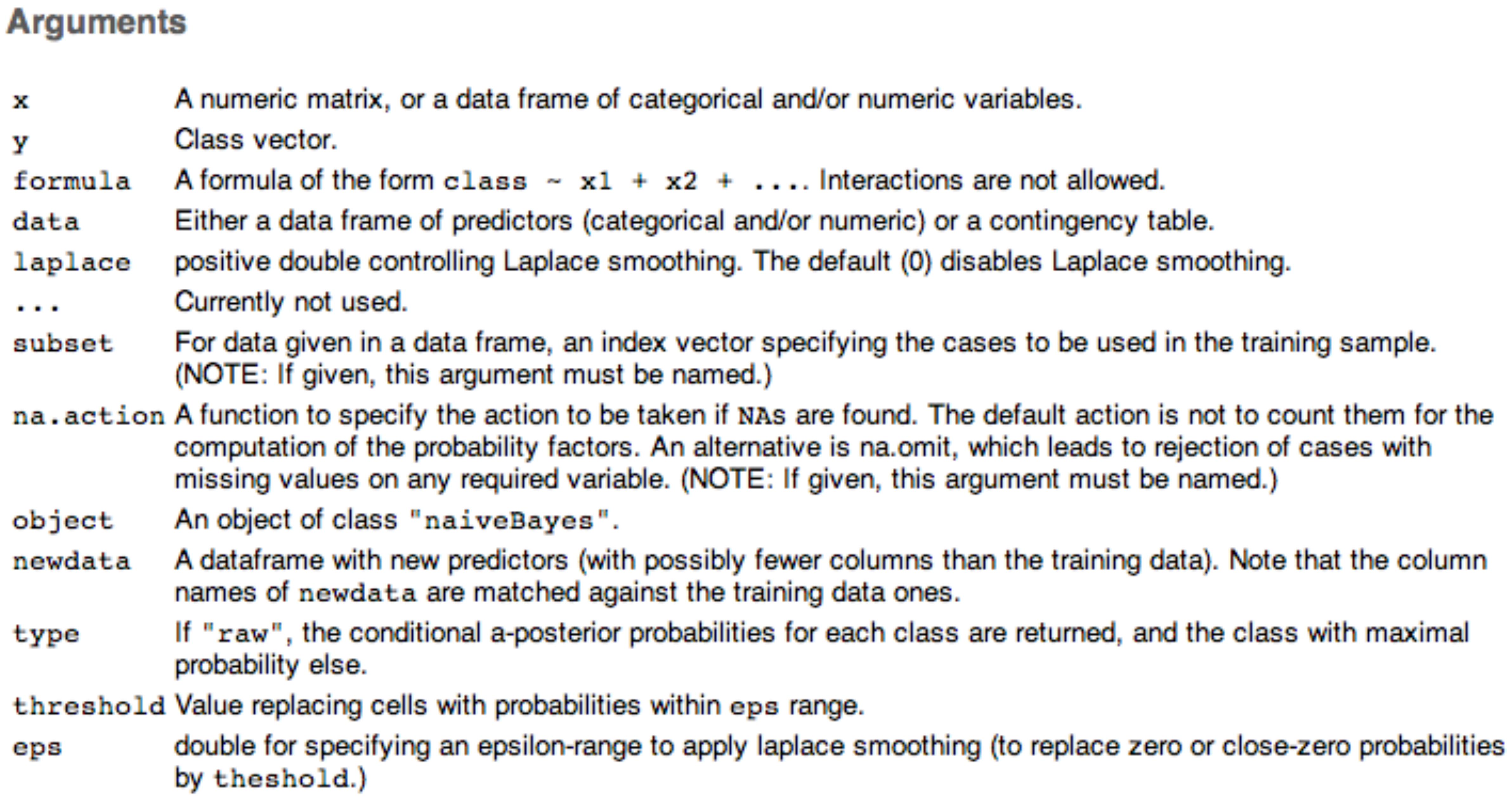
Description

Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.

Usage

```
## S3 method for class 'formula'
naiveBayes(formula, data, laplace = 0, ..., subset, na.action = na.pass)
## Default S3 method:
naiveBayes(x, y, laplace = 0, ...)


## S3 method for class 'naiveBayes'
predict(object, newdata,
  type = c("class", "raw"), threshold = 0.001, eps = 0, ...)
```

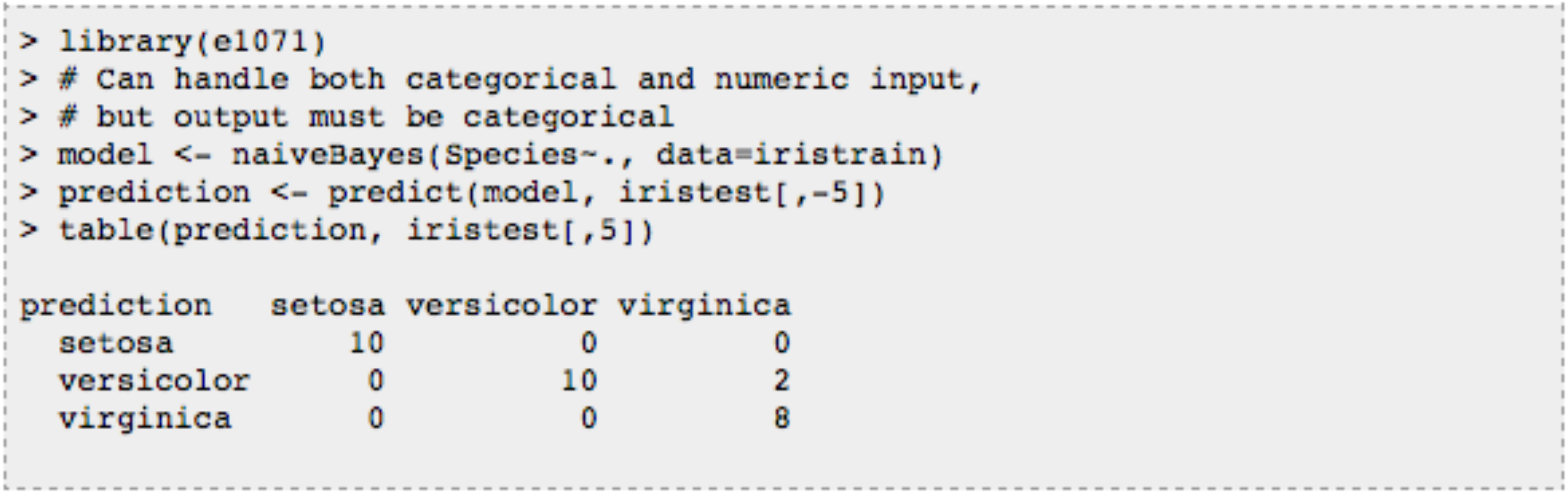
Naive Bayes



Arguments

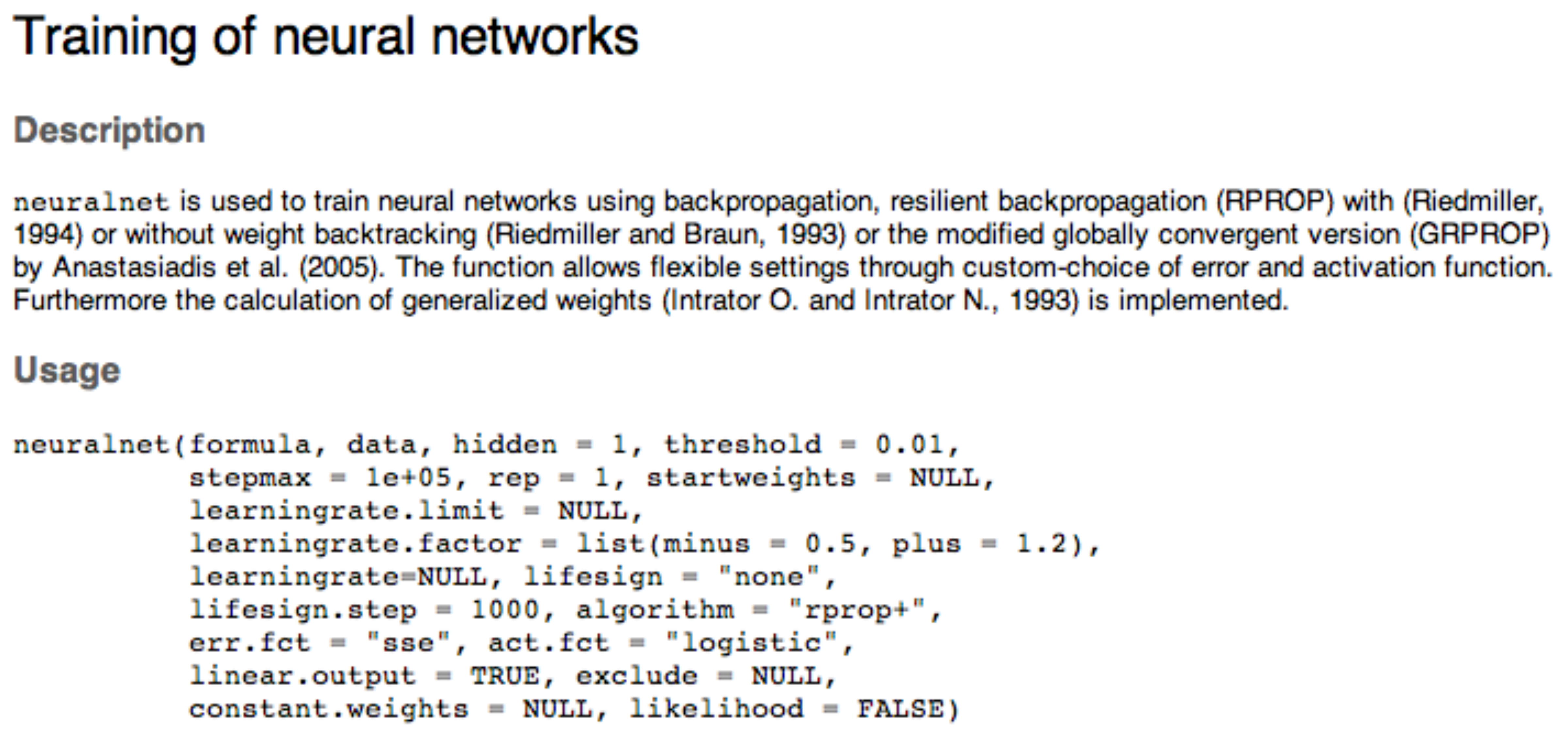
`x`	A numeric matrix, or a data frame of categorical and/or numeric variables.
`y`	Class vector.
`formula`	A formula of the form `class ~ x1 + x2 + ...`. Interactions are not allowed.
`data`	Either a data frame of predictors (categorical and/or numeric) or a contingency table.
`laplace`	positive double controlling Laplace smoothing. The default (0) disables Laplace smoothing.
`...`	Currently not used.
`subset`	For data given in a data frame, an index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is not to count them for the computation of the probability factors. An alternative is na.omit, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
`object`	An object of class `"naiveBayes"`.
`newdata`	A dataframe with new predictors (with possibly fewer columns than the training data). Note that the column names of `newdata` are matched against the training data ones.
`type`	If `"raw"`, the conditional a-posterior probabilities for each class are returned, and the class with maximal probability else.
`threshold`	Value replacing cells with probabilities within `eps` range.
`eps`	double for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by `theshold`.)

Naive Bayes



```
> library(e1071)
> # Can handle both categorical and numeric input,
> # but output must be categorical
> model <- naiveBayes(Species~., data=iristrain)
> prediction <- predict(model, iristest[,-5])
> table(prediction, iristest[,5])

prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         2
  virginica       0          0         8
```

Neural Network



Training of neural networks

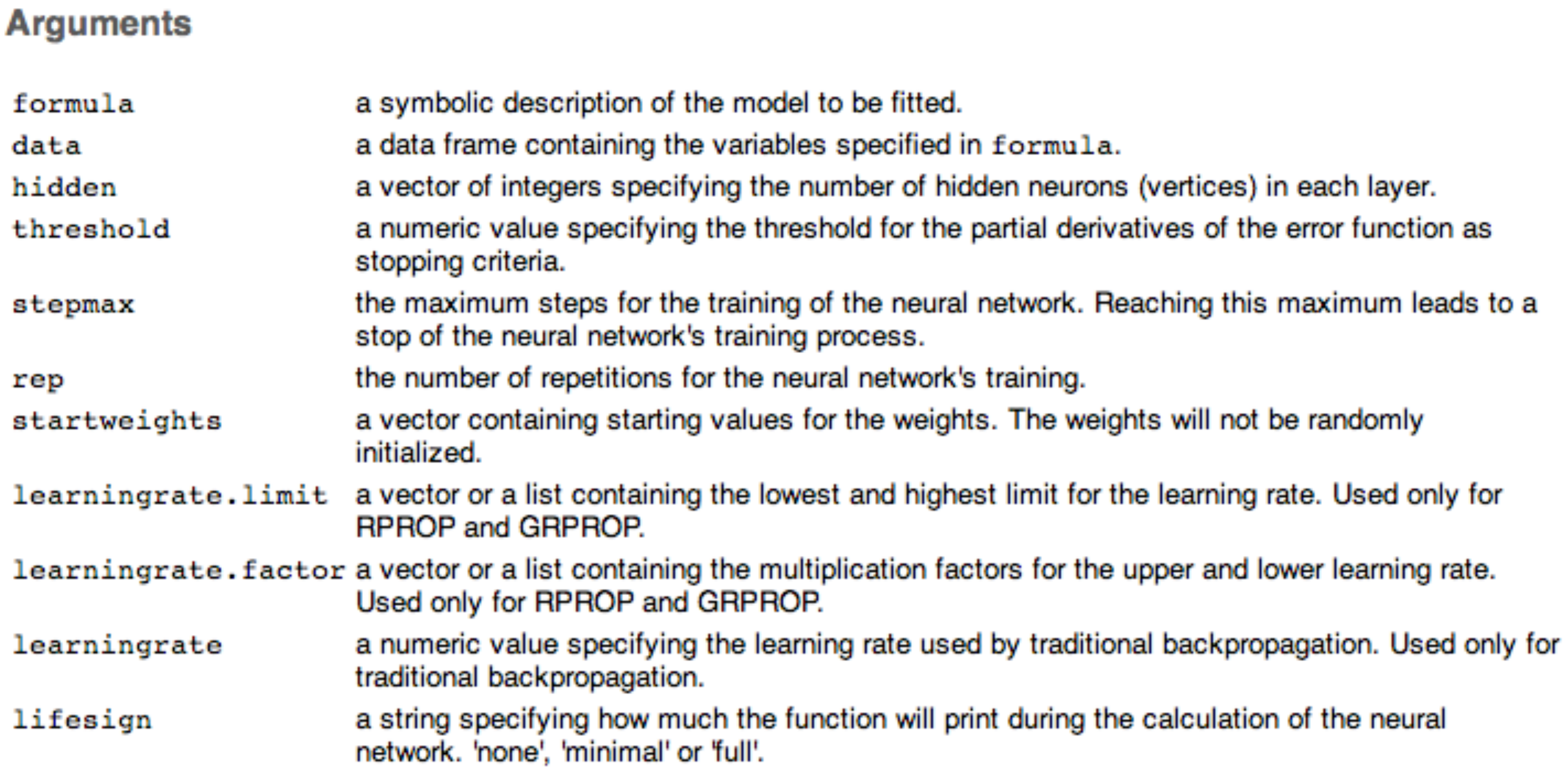
Description

`neuralnet` is used to train neural networks using backpropagation, resilient backpropagation (RPROP) with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993) or the modified globally convergent version (GRPROP) by Anastasiadis et al. (2005). The function allows flexible settings through custom-choice of error and activation function. Furthermore the calculation of generalized weights (Intrator O. and Intrator N., 1993) is implemented.

Usage

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
          stepmax = 1e+05, rep = 1, startweights = NULL,
          learningrate.limit = NULL,
          learningrate.factor = list(minus = 0.5, plus = 1.2),
          learningrate=NULL, lifesign = "none",
          lifesign.step = 1000, algorithm = "rprop+",
          err.fct = "sse", act.fct = "logistic",
          linear.output = TRUE, exclude = NULL,
          constant.weights = NULL, likelihood = FALSE)
```

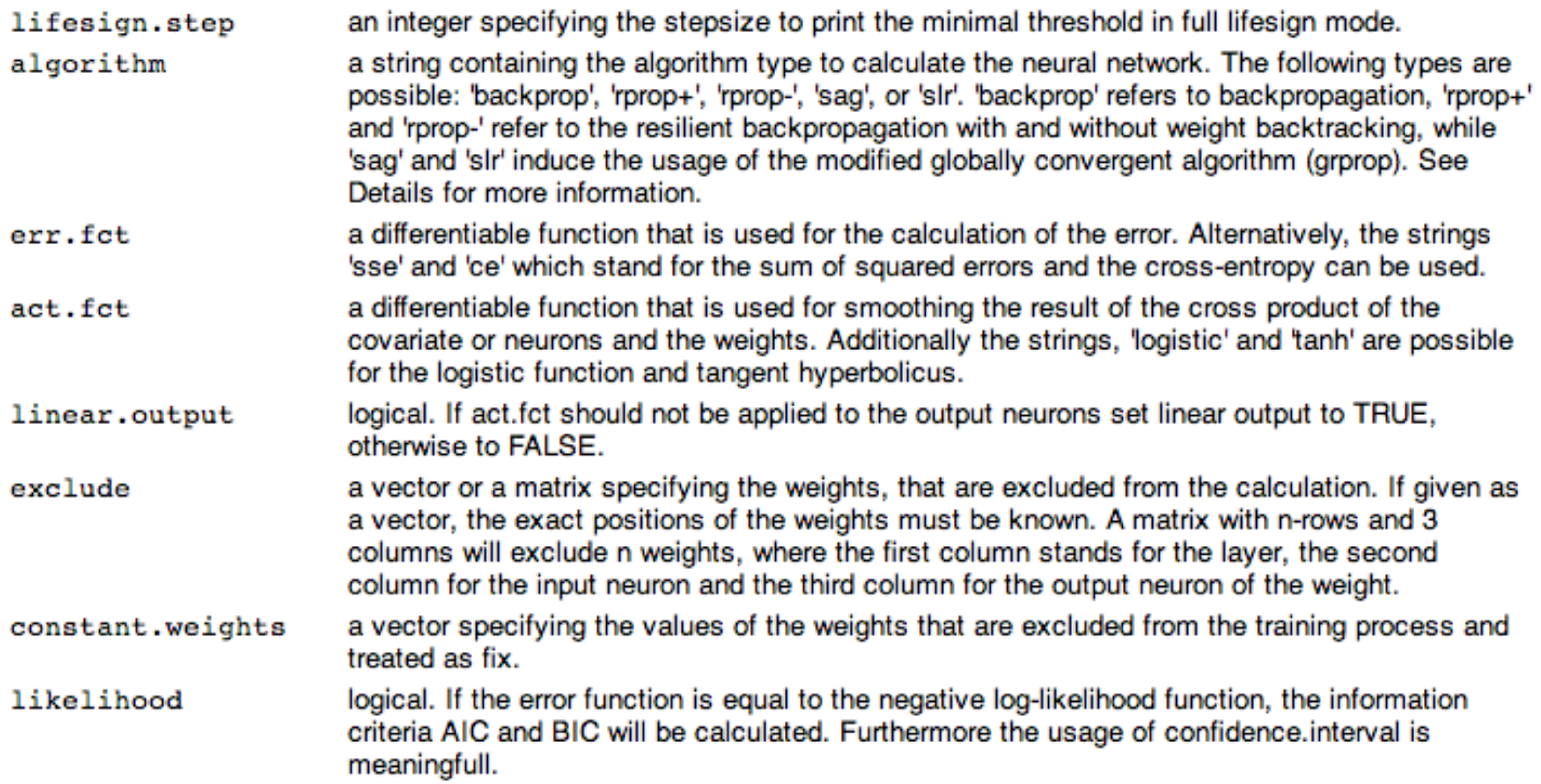
Neural Network



Arguments

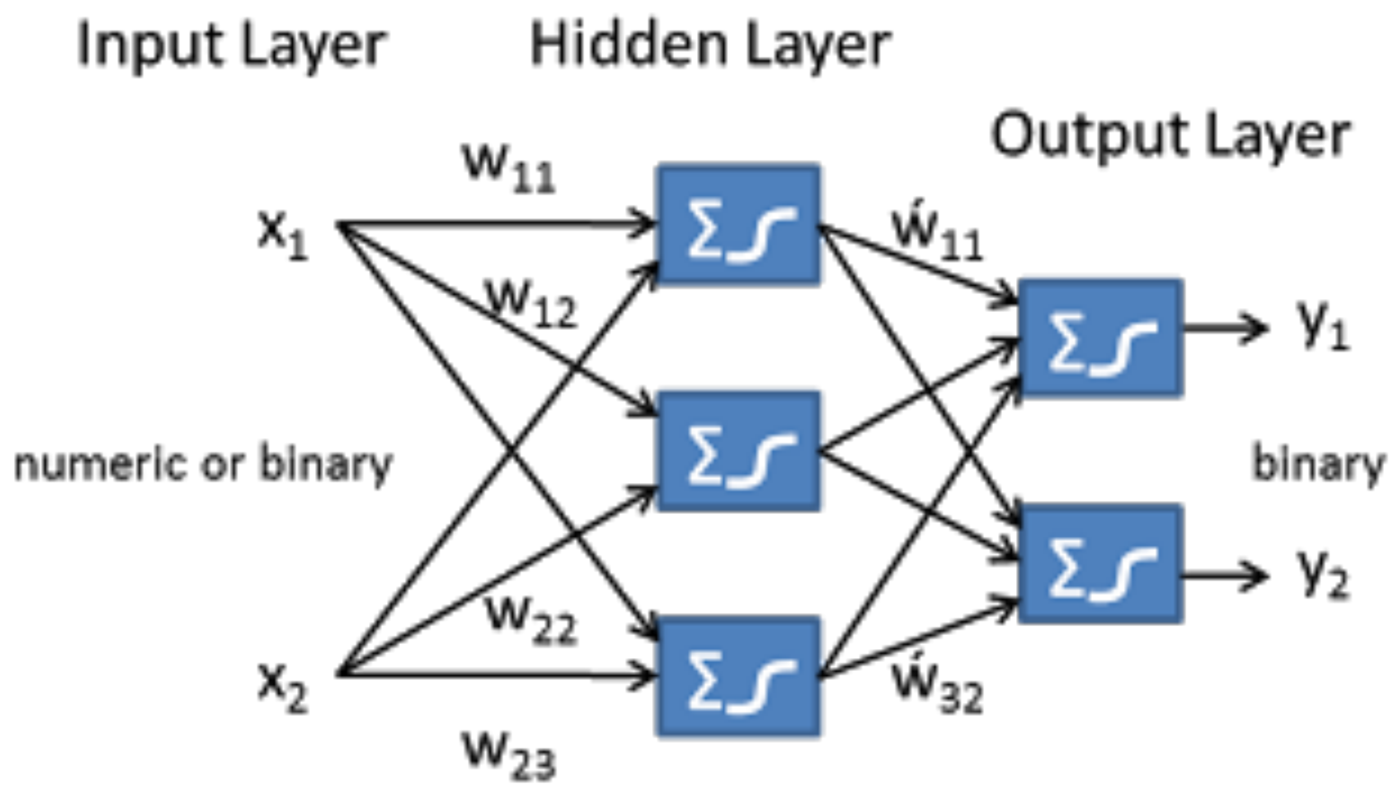
`formula`	a symbolic description of the model to be fitted.
`data`	a data frame containing the variables specified in `formula`.
`hidden`	a vector of integers specifying the number of hidden neurons (vertices) in each layer.
`threshold`	a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.
`stepmax`	the maximum steps for the training of the neural network. Reaching this maximum leads to a stop of the neural network's training process.
`rep`	the number of repetitions for the neural network's training.
`startweights`	a vector containing starting values for the weights. The weights will not be randomly initialized.
`learningrate.limit`	a vector or a list containing the lowest and highest limit for the learning rate. Used only for RPROP and GRPROP.
`learningrate.factor`	a vector or a list containing the multiplication factors for the upper and lower learning rate. Used only for RPROP and GRPROP.
`learningrate`	a numeric value specifying the learning rate used by traditional backpropagation. Used only for traditional backpropagation.
`lifesign`	a string specifying how much the function will print during the calculation of the neural network. 'none', 'minimal' or 'full'.

Neural Network

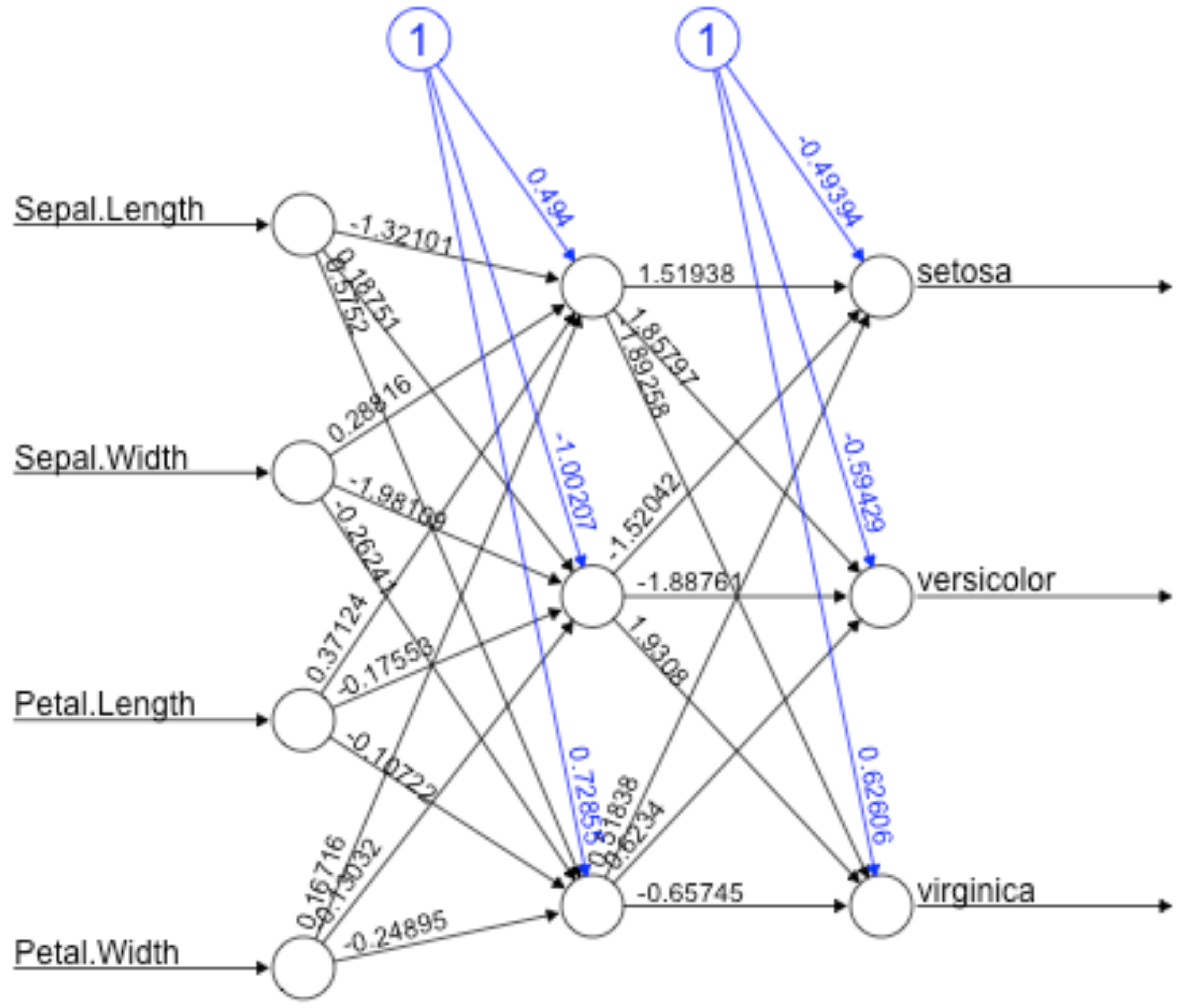


`lifesign.step`	an integer specifying the stepsize to print the minimal threshold in full lifesign mode.
`algorithm`	a string containing the algorithm type to calculate the neural network. The following types are possible: 'backprop', 'rprop+', 'rprop-', 'sag', or 'slr'. 'backprop' refers to backpropagation, 'rprop+' and 'rprop-' refer to the resilient backpropagation with and without weight backtracking, while 'sag' and 'slr' induce the usage of the modified globally convergent algorithm (grprop). See Details for more information.
`err.fct`	a differentiable function that is used for the calculation of the error. Alternatively, the strings 'sse' and 'ce' which stand for the sum of squared errors and the cross-entropy can be used.
`act.fct`	a differentiable function that is used for smoothing the result of the cross product of the covariate or neurons and the weights. Additionally the strings, 'logistic' and 'tanh' are possible for the logistic function and tangent hyperbolicus.
`linear.output`	logical. If act.fct should not be applied to the output neurons set linear output to TRUE, otherwise to FALSE.
`exclude`	a vector or a matrix specifying the weights, that are excluded from the calculation. If given as a vector, the exact positions of the weights must be known. A matrix with n-rows and 3 columns will exclude n weights, where the first column stands for the layer, the second column for the input neuron and the third column for the output neuron of the weight.
`constant.weights`	a vector specifying the values of the weights that are excluded from the training process and treated as fix.
`likelihood`	logical. If the error function is equal to the negative log-likelihood function, the information criteria AIC and BIC will be calculated. Furthermore the usage of confidence.interval is meaningfull.

Neural Network



```
> library(neuralnet)
> nnet_iristrain <-iristrain
> #Binarize the categorical output
> nnet_iristrain <- cbind(nnet_iristrain,
                          iristrain$Species == 'setosa')
> nnet_iristrain <- cbind(nnet_iristrain,
                          iristrain$Species == 'versicolor')
> nnet_iristrain <- cbind(nnet_iristrain,
                          iristrain$Species == 'virginica')
> names(nnet_iristrain)[6] <- 'setosa'
> names(nnet_iristrain)[7] <- 'versicolor'
> names(nnet_iristrain)[8] <- 'virginica'
> nn <- neuralnet(setosa+versicolor+virginica ~
                  Sepal.Length+Sepal.Width
                              +Petal.Length
                              +Petal.Width,
                  data=nnet_iristrain,
                  hidden=c(3))
> plot(nn)
> mypredict <- compute(nn, iristest[-5])$net.result
> # Put multiple binary output to categorical output
> maxidx <- function(arr) {
      return(which(arr == max(arr)))
  }
> idx <- apply(mypredict, c(1), maxidx)
> prediction <- c('setosa', 'versicolor', 'virginica')[idx]
> table(prediction, iristest$Species)

prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         3
  virginica       0          0         7
```



Error: 0.001277   Steps: 281

Clustering

- K-Means Clustering
- Hierarchical Clustering

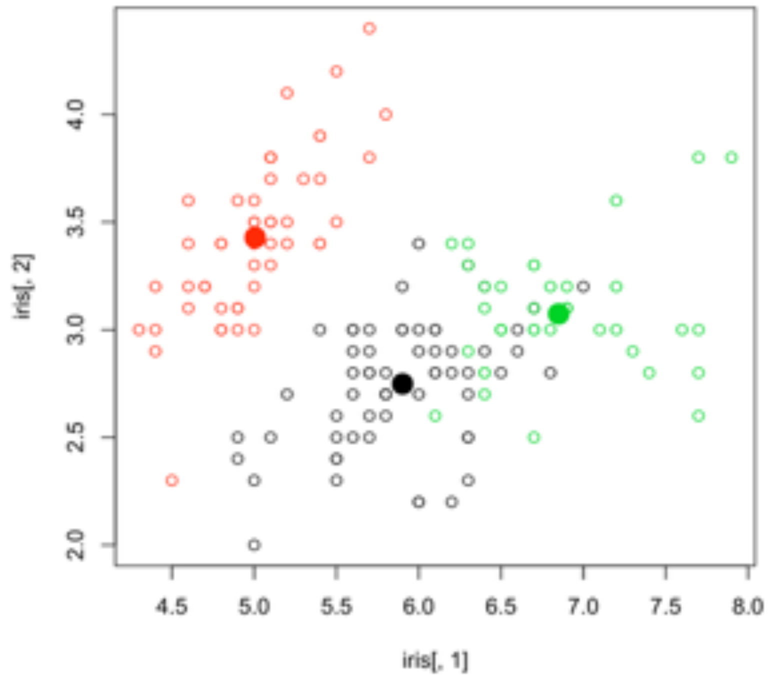
K-Means Clustering

1. Pick an initial set of K centroids (this can be random or any other means)

2. For each data point, assign it to the member of the closest centroid according to the given distance function

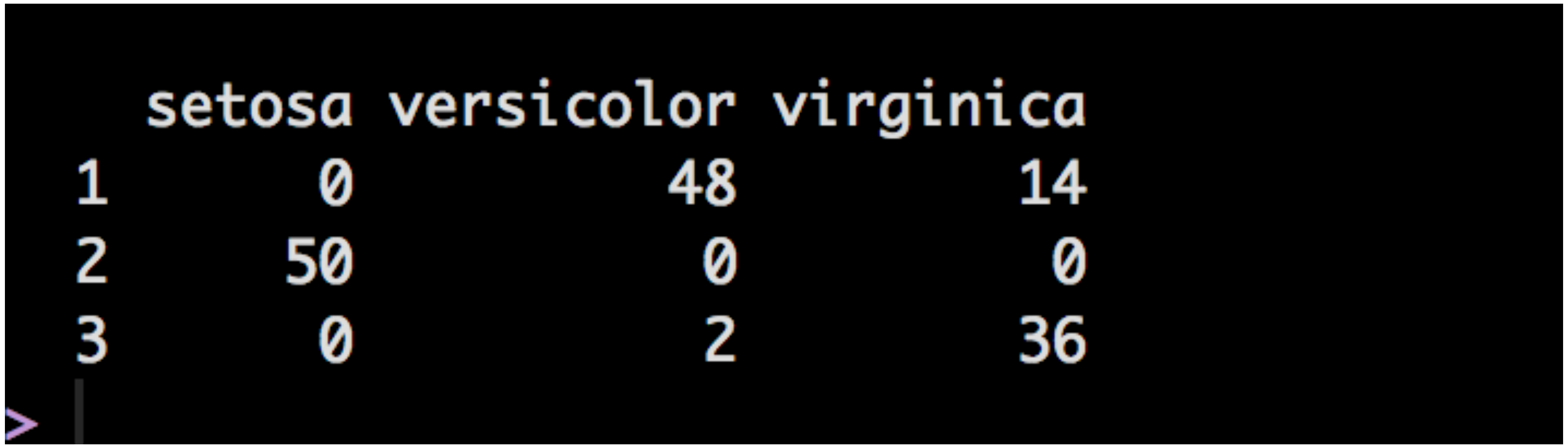
3. Adjust the centroid position as the mean of all its assigned member data points. Go back to (2) until the membership isn't change and centroid position is stable.

4. Output the centroids.



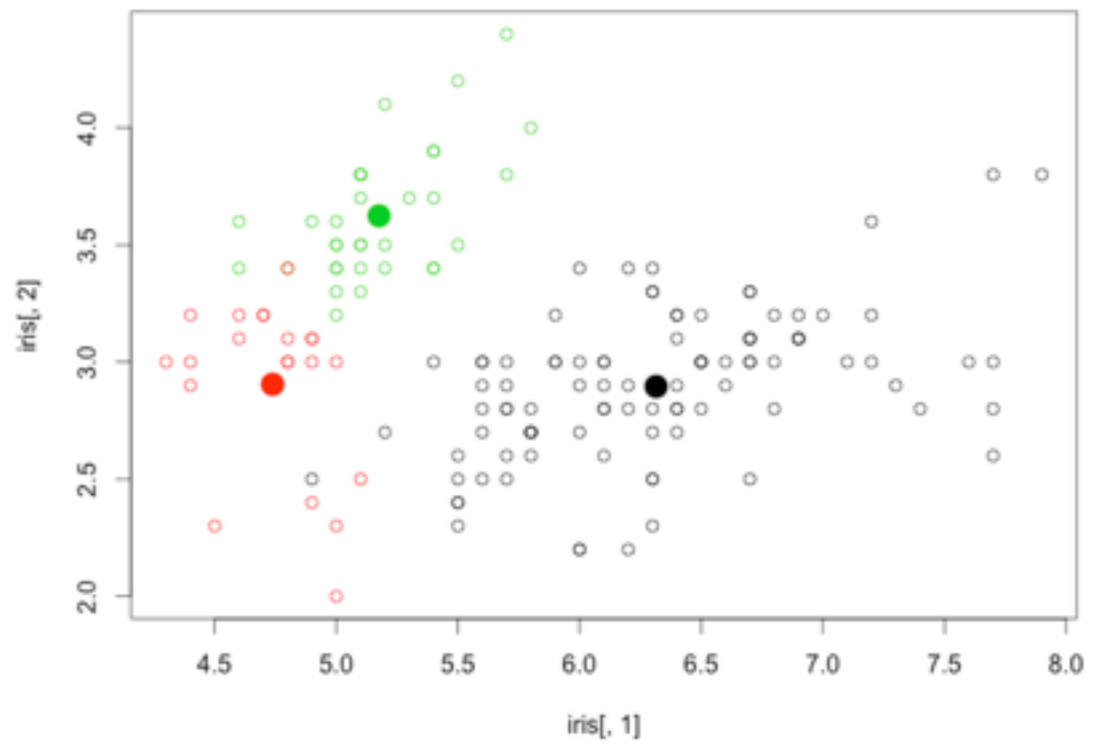
- library(stats)
- set.seed(101)
- km <- kmeans(iris[,1:4], 3)
- plot(iris[,1], iris[,2], col=km$cluster)
- points(km$centers[,c(1,2)], col=1:3, pch=19, cex=2)

- table(km$cluster, iris$Species)

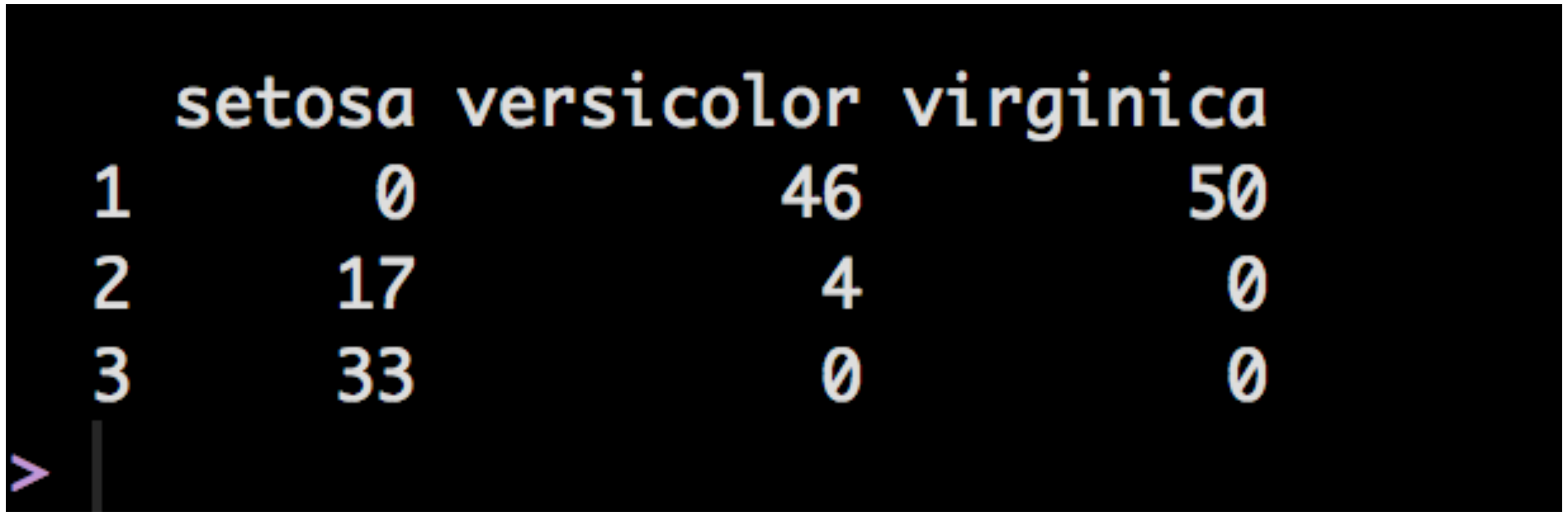


```
   setosa versicolor virginica
 1      0         48        14
 2     50          0         0
 3      0          2        36
>
```

Another round



- set.seed(900)
- km <- kmeans(iris[,1:4], 3)
- plot(iris[,1], iris[,2], col=km$cluster)
- points(km$centers[,c(1,2)], col=1:3, pch=19, cex=2)



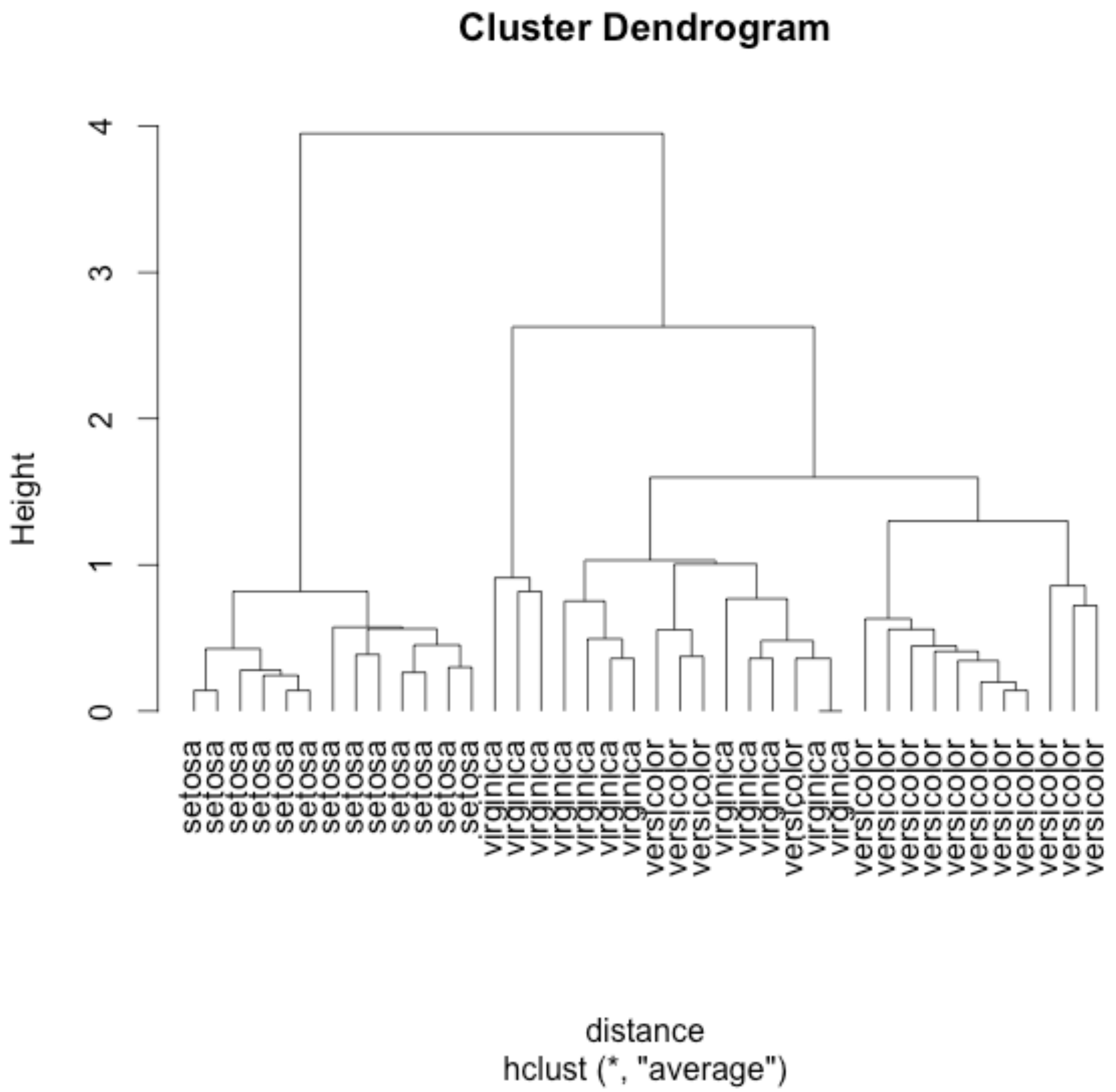
```
   setosa versicolor virginica
  1      0         46        50
  2     17          4         0
  3     33          0         0
>
```

- table(km$cluster, iris$Species)

Hierarchical Clustering

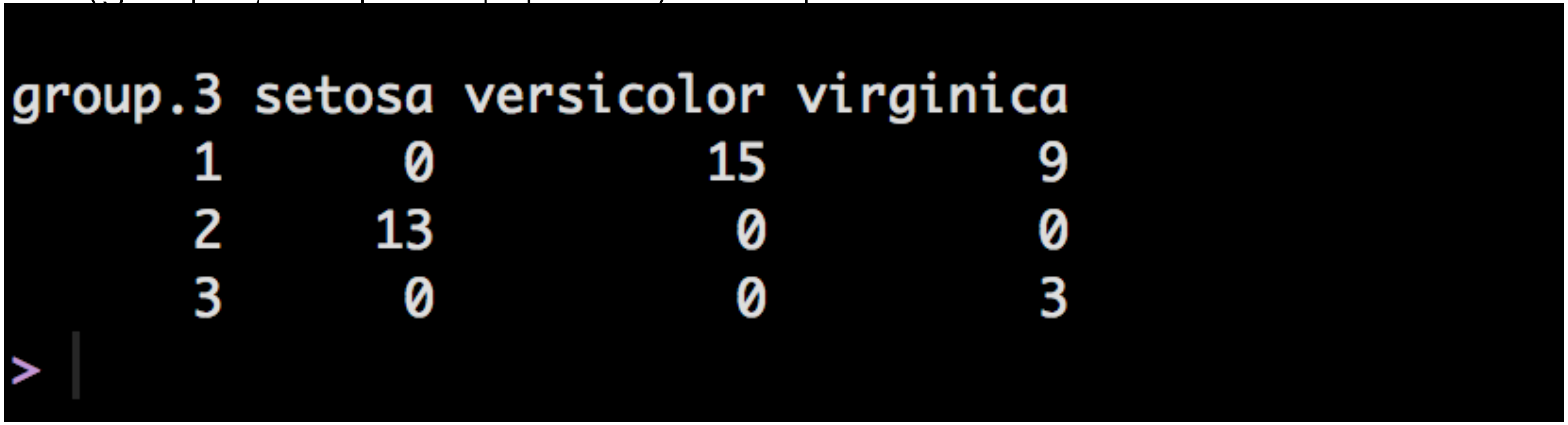
- Compute distance between every pairs of point/cluster.
    - (a) Distance between point is just using the distance function.
    - (b) Compute distance between pointA to clusterB may involve many choices (such as the min/max/avg distance between the pointA and points in the clusterB).
    - (c) Compute distance between clusterA to clusterB may first compute distance of all points pairs (one from clusterA and the other from clusterB) and then pick either min/max/avg of these pairs.
- Combine the two closest point/cluster into a cluster. Go back to (1) until only one big cluster remains

- set.seed(101)
- sampleiris <- iris[sample(1:150, 40),] # get samples from iris dataset
- # each observation has 4 variables, ie, they are interpreted as 4-D points
- distance   <- dist(sampleiris[,-5], method="euclidean")
- cluster   <- hclust(distance, method="average")
- plot(cluster, hang=-1, label=sampleiris$Species)

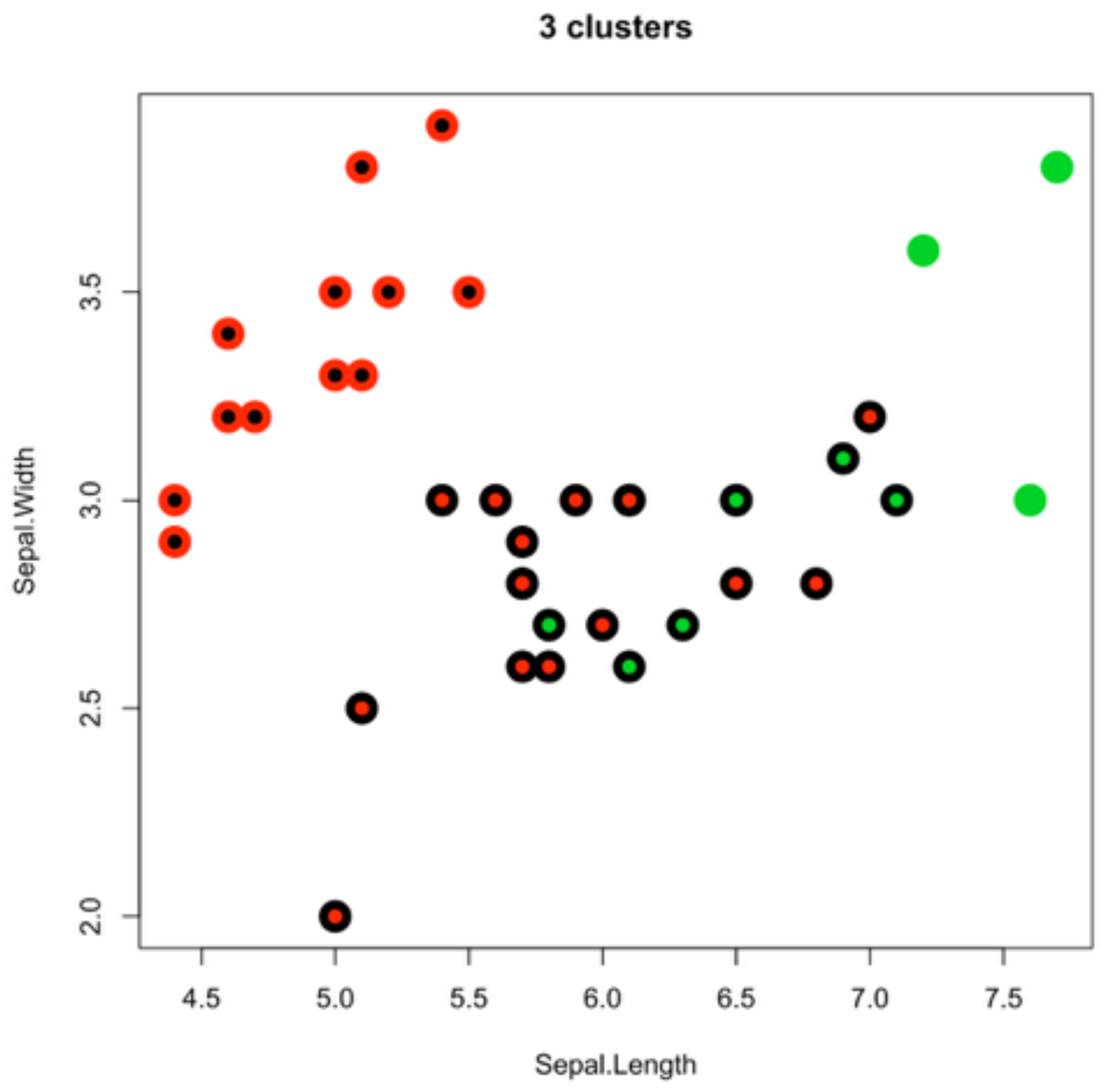

Cluster Dendrogram
Height
0
1
2
3
4
setosa
virginica
versicolor
distance
hclust (*, "average")

It’s possible to prune the result tree.

- par(mfrow=c(1,2))
- group.3 <- cutree(cluster, k = 3) # prune the tree by 3 clusters
- table(group.3, sampleiris$Species) # compare with known classes



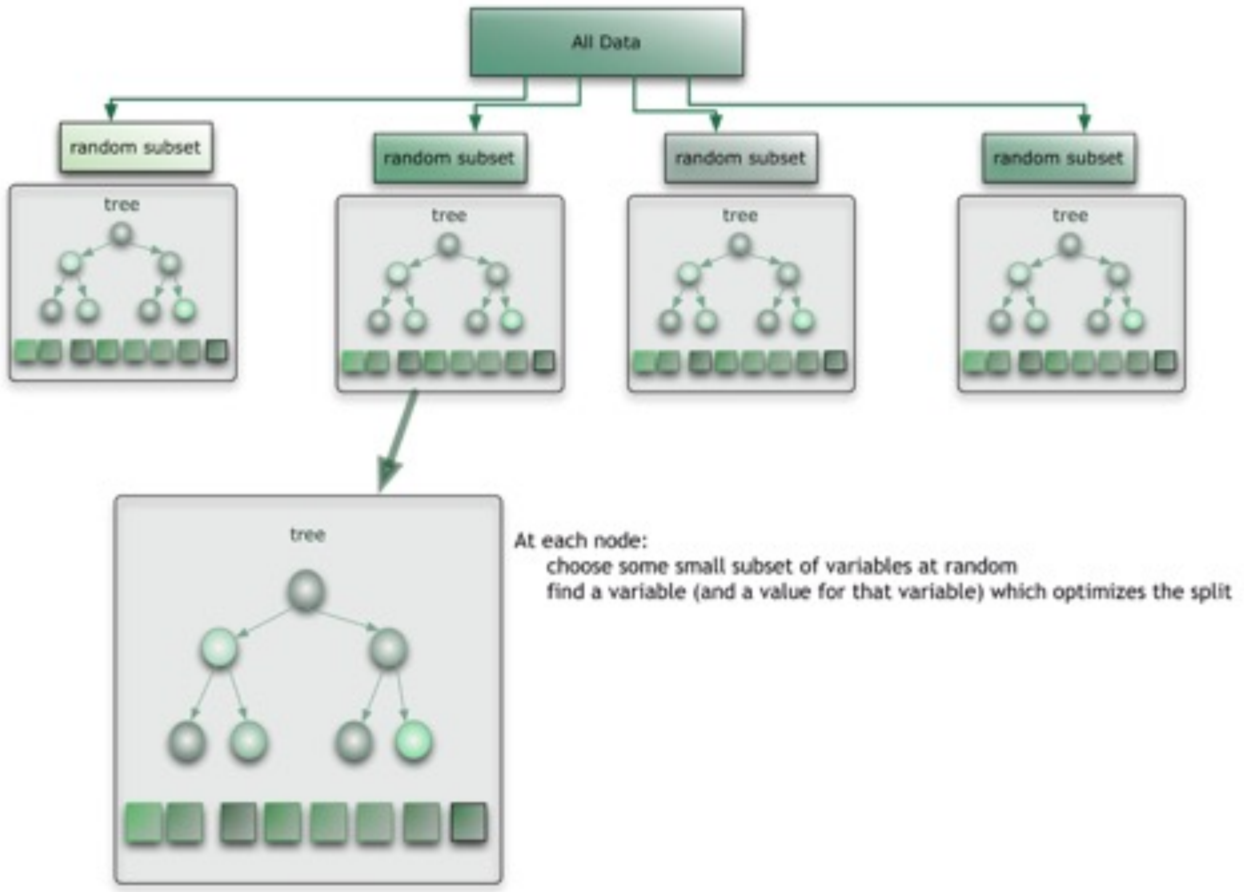
```
group.3 setosa versicolor virginica
      1      0         15         9
      2     13          0         0
      3      0          0         3
> 
```



- plot(sampleiris[,c(1,2)], col=group.3, pch=19, cex=2.5, main="3 clusters")
- points(sampleiris[,c(1,2)], col=sampleiris$Species, pch=19, cex=1)

Ensemble : Bagging

- Random Forest


All Data
random subset
random subset
random subset
random subset
tree
tree
tree
tree
tree
At each node:
choose some small subset of variables at random
find a variable (and a value for that variable) which optimizes the split

Random Forest

- Here is how such a system is trained; for some number of trees T:

- 1) Sample N cases at random with replacement to create a subset of the data. The subset should be about 66% of the total set.
- 2) At each node:
  - a) For some number m (see below), m predictor variables are selected at random from all the predictor variables.
  - b) The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.
  - c) At the next node, choose another m variables at random from all predictor variables and do the same.

Bagging

```
> library(randomForest)
#Train 100 trees, random selected attributes
> model <- randomForest(Species~., data=iristrain, nTree=500)
#Predict using the forest
> prediction <- predict(model, newdata=iristest, type='class')
> table(prediction, iristest$Species)
> importance(model)
             MeanDecreaseGini
Sepal.Length         7.807602
Sepal.Width          1.677239
Petal.Length        31.145822
Petal.Width         38.617223
```

Boosting

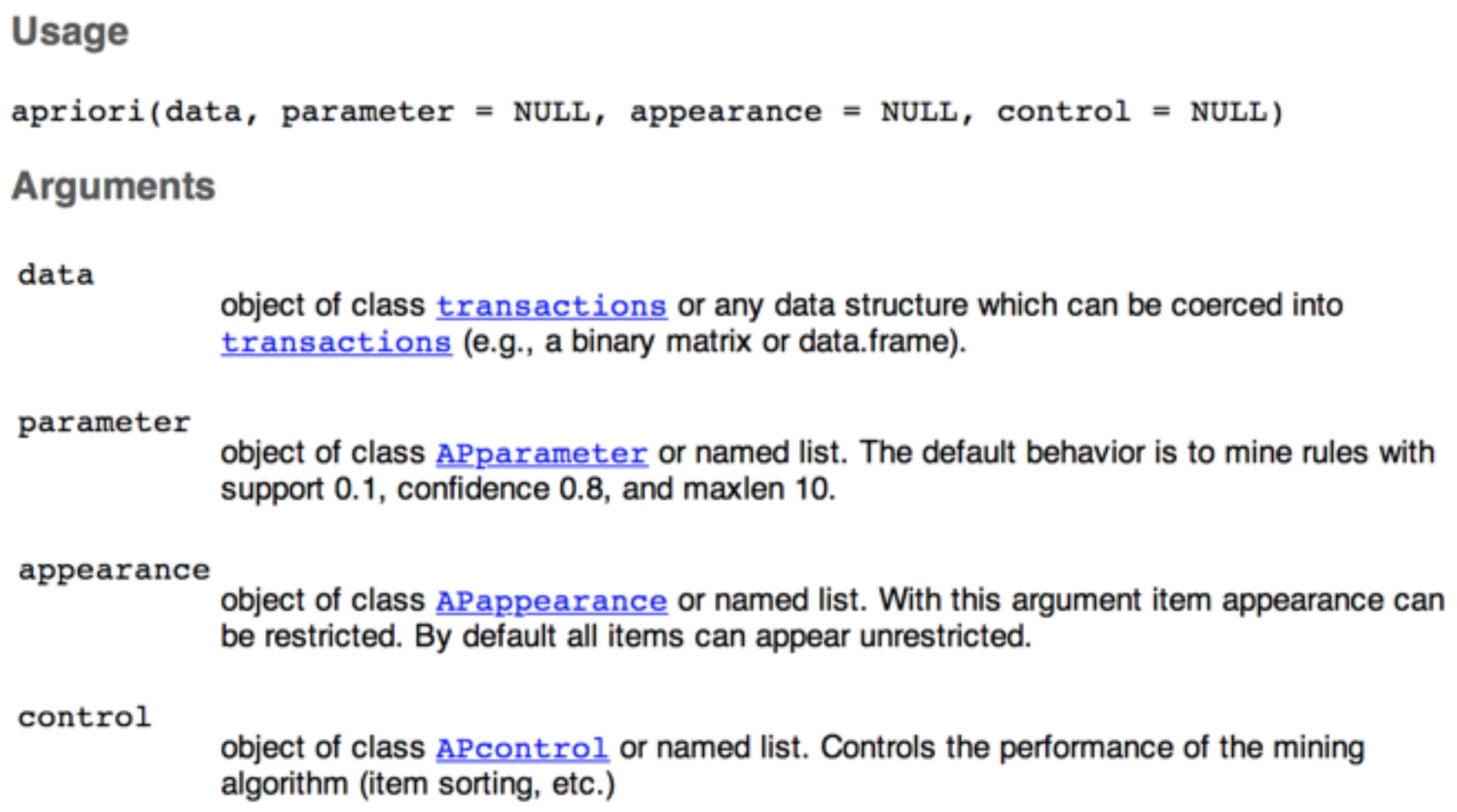
- > library(adabag)
- > iris.adaboost <- boosting(Species~., data=iristrain, boost=TRUE, mfinal=5)
- > iris.adaboost

Association Rules (Market Basket Analysis)

- **Support**: The fraction of which our item set occurs in our dataset.
- **Confidence**: probability that a rule is correct for a new transaction with items on the left.
- **Lift**: The ratio by which by the confidence of a rule exceeds the expected confidence.
- **Note**: if the lift is 1 it indicates that the items on the left and right are independent

Apriori Algorithm

- ?apriori



Usage

```
apriori(data, parameter = NULL, appearance = NULL, control = NULL)
```

Arguments

`data`	object of class `transactions` or any data structure which can be coerced into `transactions` (e.g., a binary matrix or data.frame).
`parameter`	object of class `APparameter` or named list. The default behavior is to mine rules with support 0.1, confidence 0.8, and maxlen 10.
`appearance`	object of class `APappearance` or named list. With this argument item appearance can be restricted. By default all items can appear unrestricted.
`control`	object of class `APcontrol` or named list. Controls the performance of the mining algorithm (item sorting, etc.)

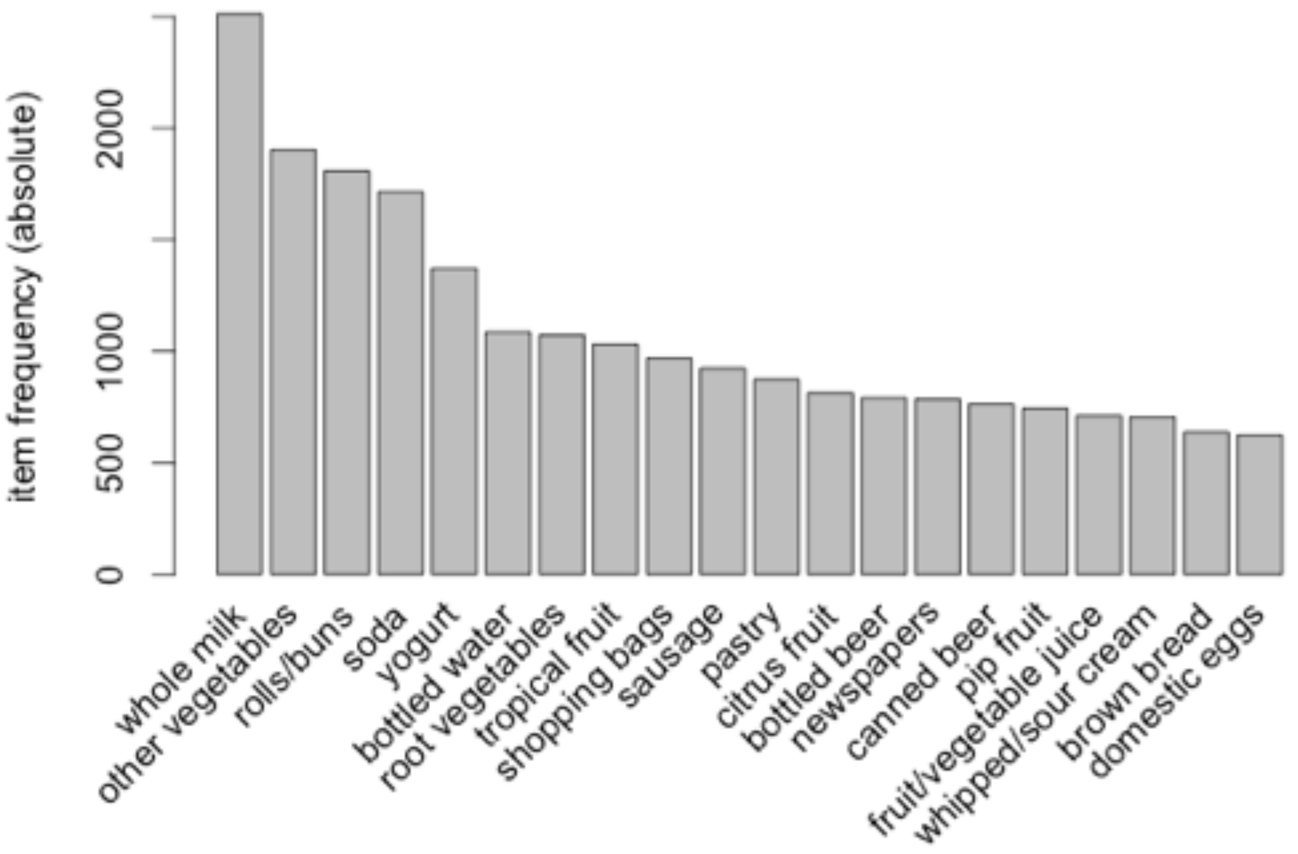
Apriori Algorithm

So lets get started by loading up our libraries and data set.

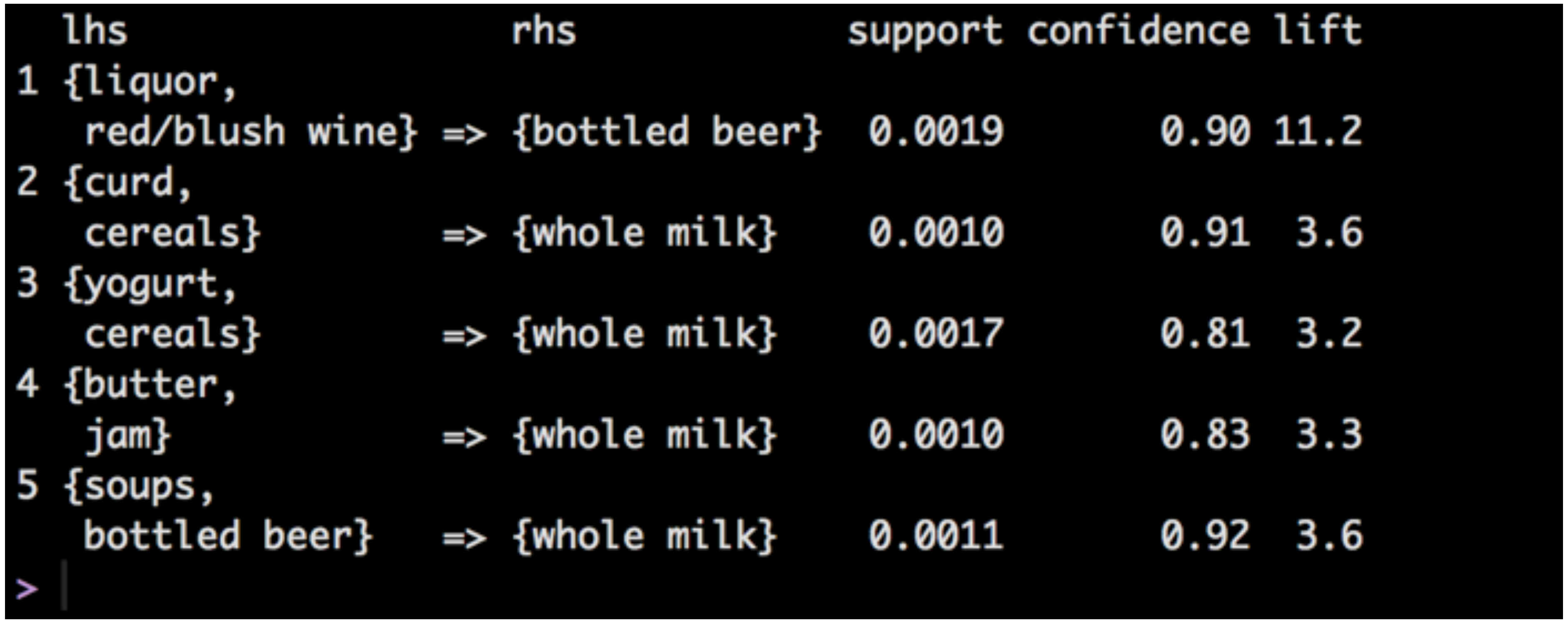
- # Load the libraries
- library(arules)
- library(arulesViz)
- library(datasets)
- # Load the data set
- data(Groceries)

Explore Data

- # Create an item frequency plot for the top 20 items
- itemFrequencyPlot(Groceries, topN=20,type="absolute")

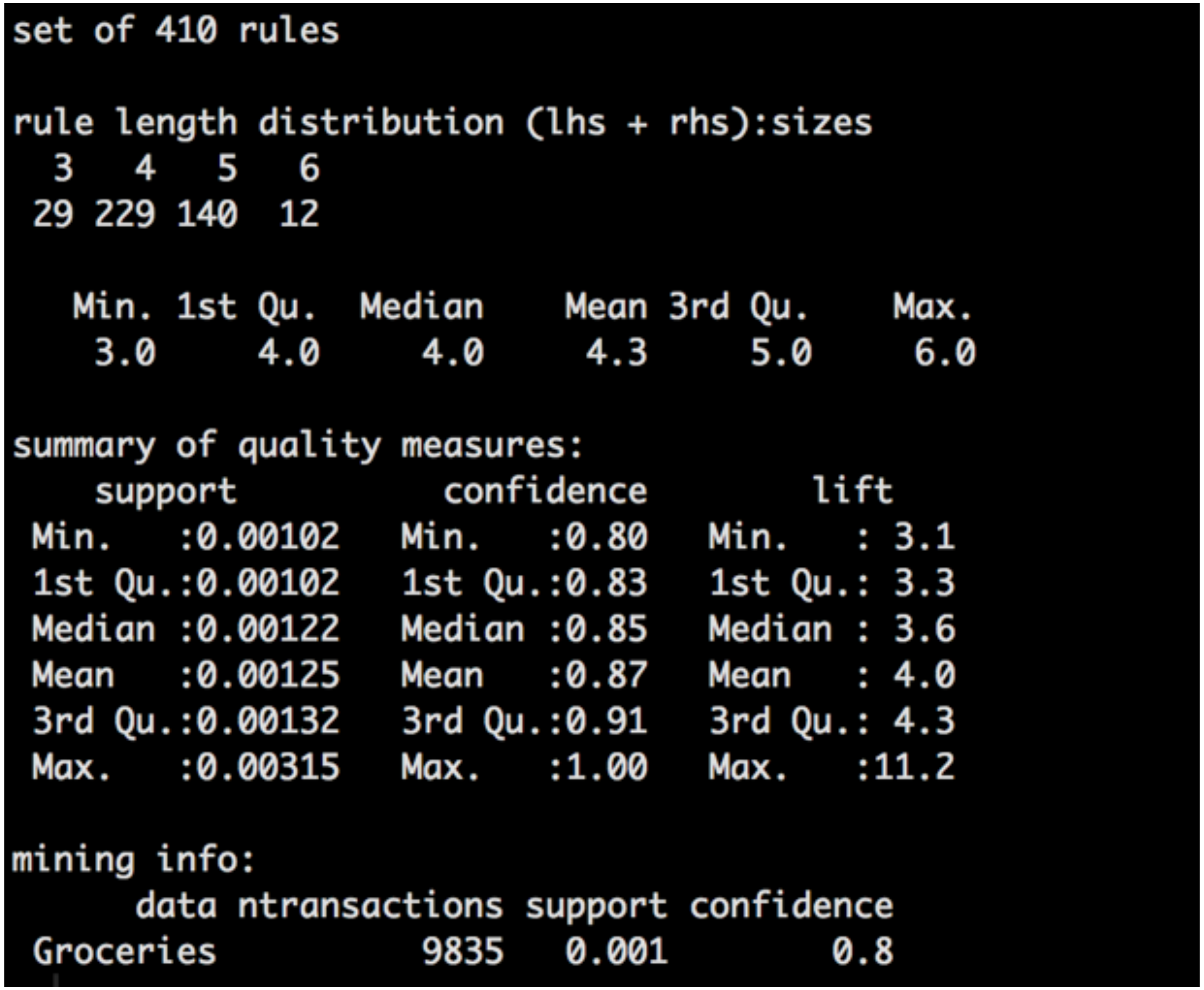


- **rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))**
- # Show the top 5 rules, but only 2 digits
- options(digits=2)
- inspect(rules[1:5])



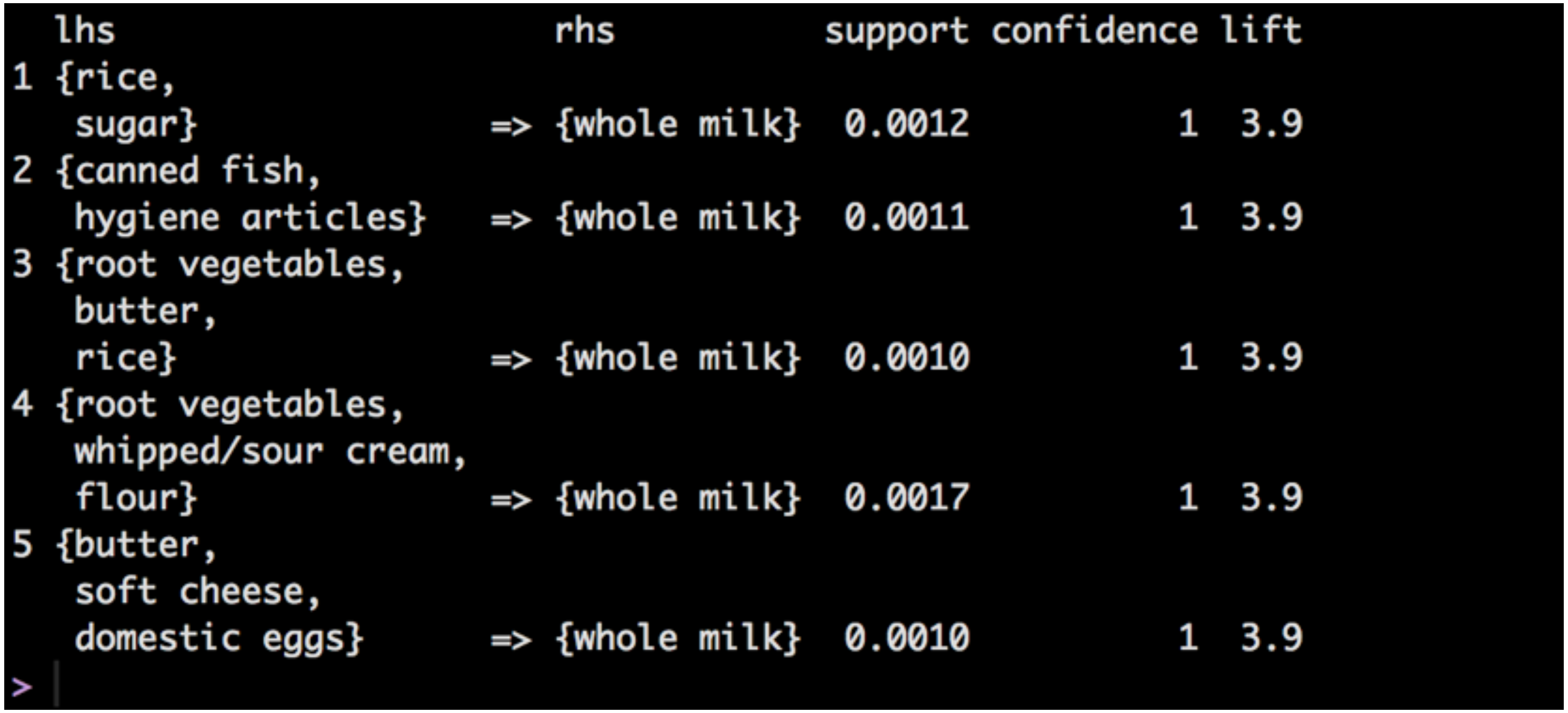
```
  lhs                    rhs              support confidence lift
1 {liquor,
   red/blush wine} => {bottled beer}  0.0019       0.90 11.2
2 {curd,
   cereals}        => {whole milk}    0.0010       0.91  3.6
3 {yogurt,
   cereals}        => {whole milk}    0.0017       0.81  3.2
4 {butter,
   jam}            => {whole milk}    0.0010       0.83  3.3
5 {soups,
   bottled beer}   => {whole milk}    0.0011       0.92  3.6
> 
```

- summary(rules)



```
set of 410 rules

rule length distribution (lhs + rhs):sizes
  3   4   5   6
 29 229 140  12

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    3.0     4.0     4.0     4.3     5.0     6.0

summary of quality measures:
    support           confidence          lift
 Min.   :0.00102   Min.   :0.80   Min.   : 3.1
 1st Qu.:0.00102   1st Qu.:0.83   1st Qu.: 3.3
 Median :0.00122   Median :0.85   Median : 3.6
 Mean   :0.00125   Mean   :0.87   Mean   : 4.0
 3rd Qu.:0.00132   3rd Qu.:0.91   3rd Qu.: 4.3
 Max.   :0.00315   Max.   :1.00   Max.   :11.2

mining info:
      data ntransactions support confidence
 Groceries          9835   0.001        0.8
```

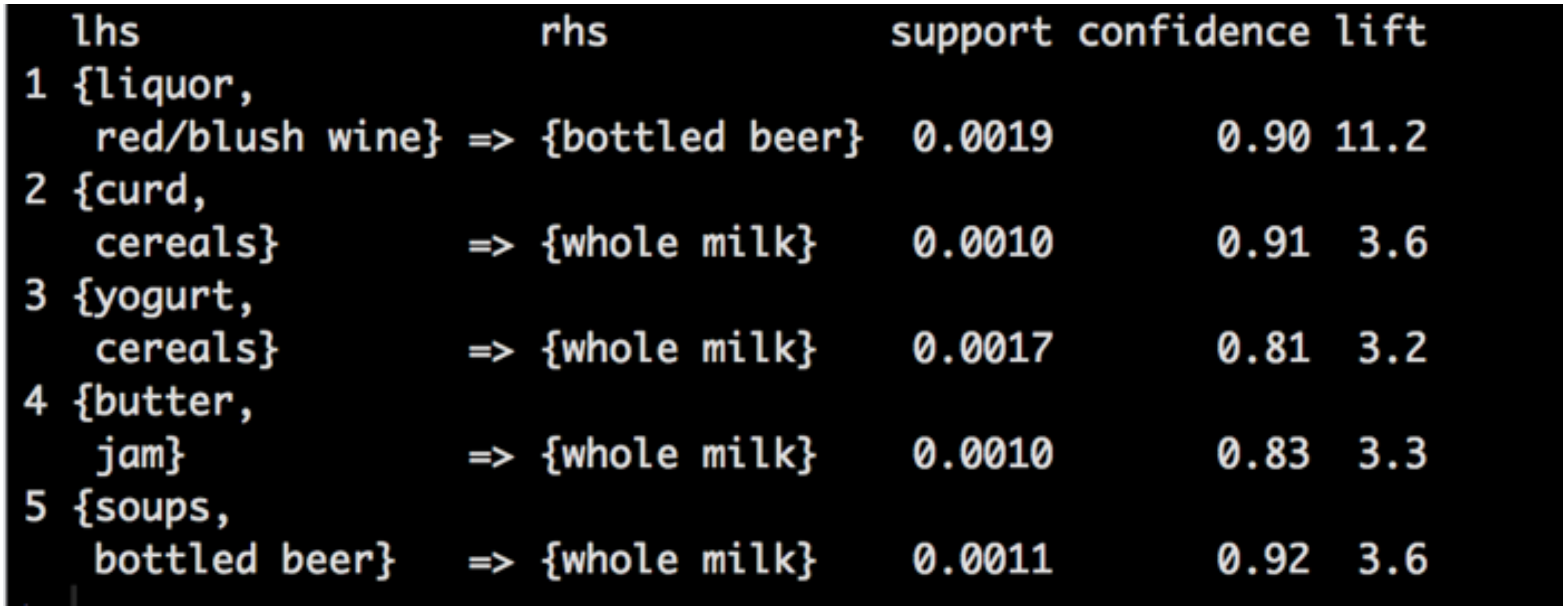
- # Sort Rules
- rules<-sort(rules, by="confidence", decreasing=TRUE)
- inspect(rules[1:5])



```
  lhs                         rhs            support confidence lift
1 {rice,
   sugar}                  => {whole milk}  0.0012          1  3.9
2 {canned fish,
   hygiene articles}       => {whole milk}  0.0011          1  3.9
3 {root vegetables,
   butter,
   rice}                   => {whole milk}  0.0010          1  3.9
4 {root vegetables,
   whipped/sour cream,
   flour}                  => {whole milk}  0.0017          1  3.9
5 {butter,
   soft cheese,
   domestic eggs}          => {whole milk}  0.0010          1  3.9
>
```

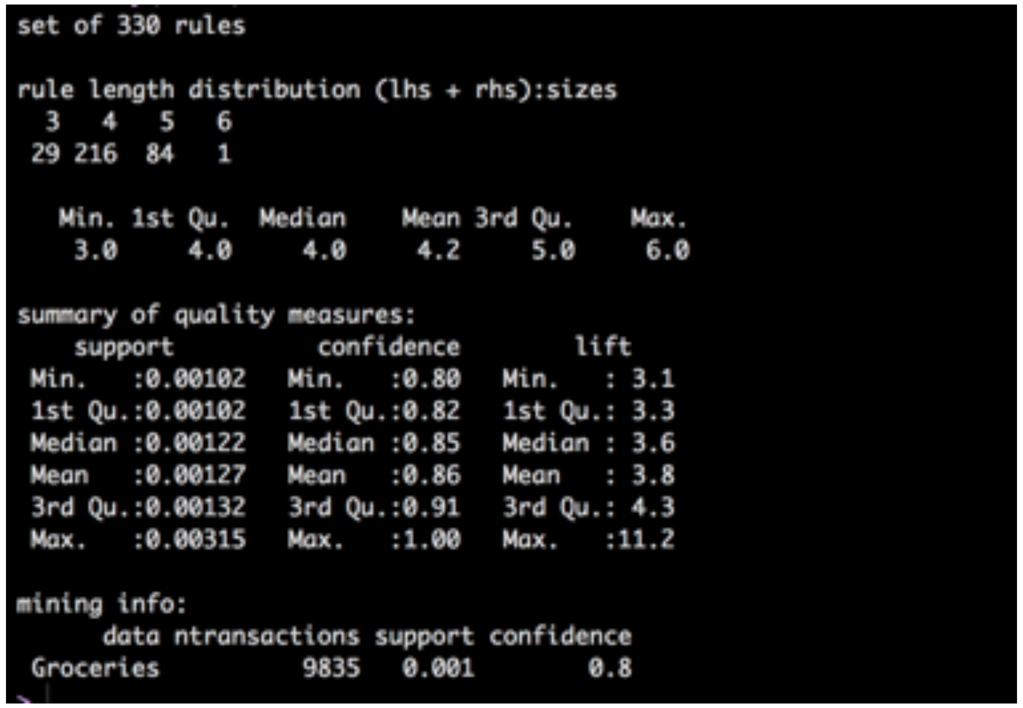
Change to have limit association in one rule

- # change to have maximum of 3
- rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8,maxlen=3))
- inspect(rules[1:5])



```
  lhs                    rhs             support confidence lift
1 {liquor,
   red/blush wine} => {bottled beer}  0.0019       0.90 11.2
2 {curd,
   cereals}        => {whole milk}    0.0010       0.91  3.6
3 {yogurt,
   cereals}        => {whole milk}    0.0017       0.81  3.2
4 {butter,
   jam}            => {whole milk}    0.0010       0.83  3.3
5 {soups,
   bottled beer}   => {whole milk}    0.0011       0.92  3.6
```

- **# Rules pruned**
- subset.matrix <- is.subset(rules, rules)
- subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
- redundant <- colSums(subset.matrix, na.rm=T) >= 1
- rules.pruned <- rules[!redundant]
- rules<-rules.pruned
- summary(rules)



```
set of 330 rules

rule length distribution (lhs + rhs):sizes
  3   4   5   6
 29 216  84   1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    3.0     4.0     4.0     4.2     5.0     6.0

summary of quality measures:
    support           confidence        lift
 Min.   :0.00102   Min.   :0.80   Min.   : 3.1
 1st Qu.:0.00102   1st Qu.:0.82   1st Qu.: 3.3
 Median :0.00122   Median :0.85   Median : 3.6
 Mean   :0.00127   Mean   :0.86   Mean   : 3.8
 3rd Qu.:0.00132   3rd Qu.:0.91   3rd Qu.: 4.3
 Max.   :0.00315   Max.   :1.00   Max.   :11.2

mining info:
      data ntransactions support confidence
 Groceries          9835   0.001        0.8
```

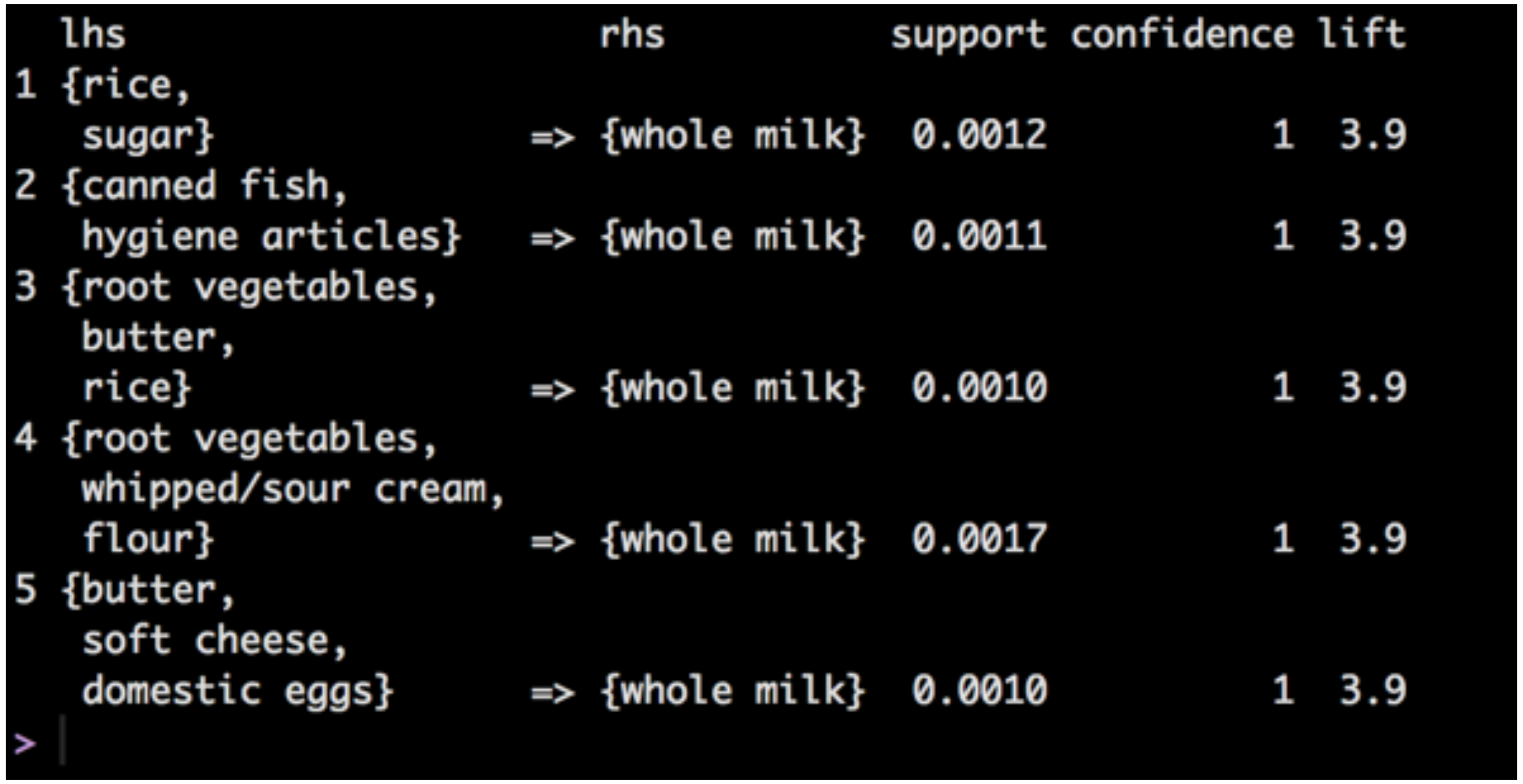
Targeting Items

- What are customers likely to buy before buying whole milk?
- What are customers likely to buy if they purchase whole milk?
- This essentially means we want to set either the Left Hand Side and Right Hand Side. This is not difficult to do with R!

- rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.08), appearance = list(default="lhs",rhs="whole milk”), control = list(verbose=F))

- rules<-sort(rules, decreasing=TRUE,by="confidence")

- inspect(rules[1:5])



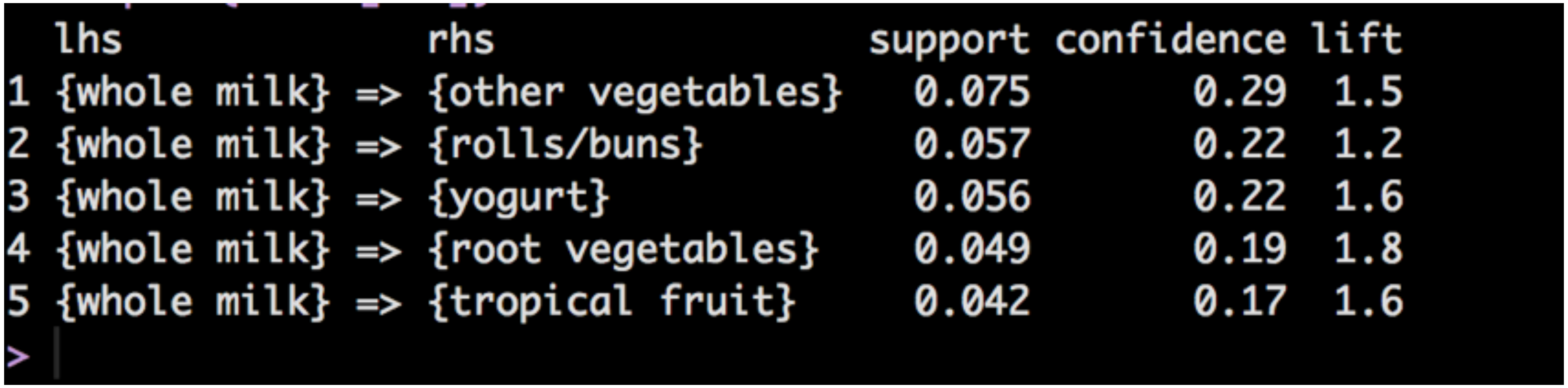
```
  lhs                        rhs          support confidence lift
1 {rice,
   sugar}                 => {whole milk}  0.0012          1  3.9
2 {canned fish,
   hygiene articles}      => {whole milk}  0.0011          1  3.9
3 {root vegetables,
   butter,
   rice}                  => {whole milk}  0.0010          1  3.9
4 {root vegetables,
   whipped/sour cream,
   flour}                 => {whole milk}  0.0017          1  3.9
5 {butter,
   soft cheese,
   domestic eggs}         => {whole milk}  0.0010          1  3.9
>
```

Find whole milk’s antecedents

- rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.15,minlen=2), appearance = list(default="rhs",lhs="whole milk”), control = list(verbose=F))

- rules<-sort(rules, decreasing=TRUE,by="confidence")

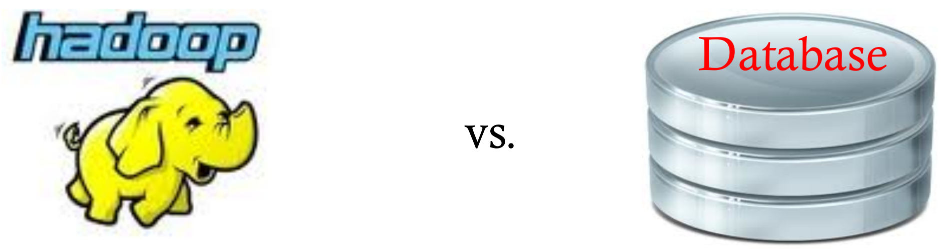
- inspect(rules[1:5])



```
  lhs              rhs                  support confidence lift
1 {whole milk} => {other vegetables}   0.075        0.29  1.5
2 {whole milk} => {rolls/buns}         0.057        0.22  1.2
3 {whole milk} => {yogurt}             0.056        0.22  1.6
4 {whole milk} => {root vegetables}    0.049        0.19  1.8
5 {whole milk} => {tropical fruit}     0.042        0.17  1.6
>
```

Hadoop an Map-reduce Paradigm

- MapReduce computing paradigm (E.g., Hadoop) vs. Traditional database systems



- Many enterprises are turning to Hadoop
  - Especially applications generating big data
  - Web applications, social networks, scientific applications

Why Hadoop is able to compete?

hadoop

vs.

Scalability (petabytes of data, thousands of machines)

Flexibility in accepting all data formats (no schema)

Efficient and simple fault-tolerant mechanism

Commodity inexpensive hardware

Performance (tons of indexing, tuning, data organization tech.)

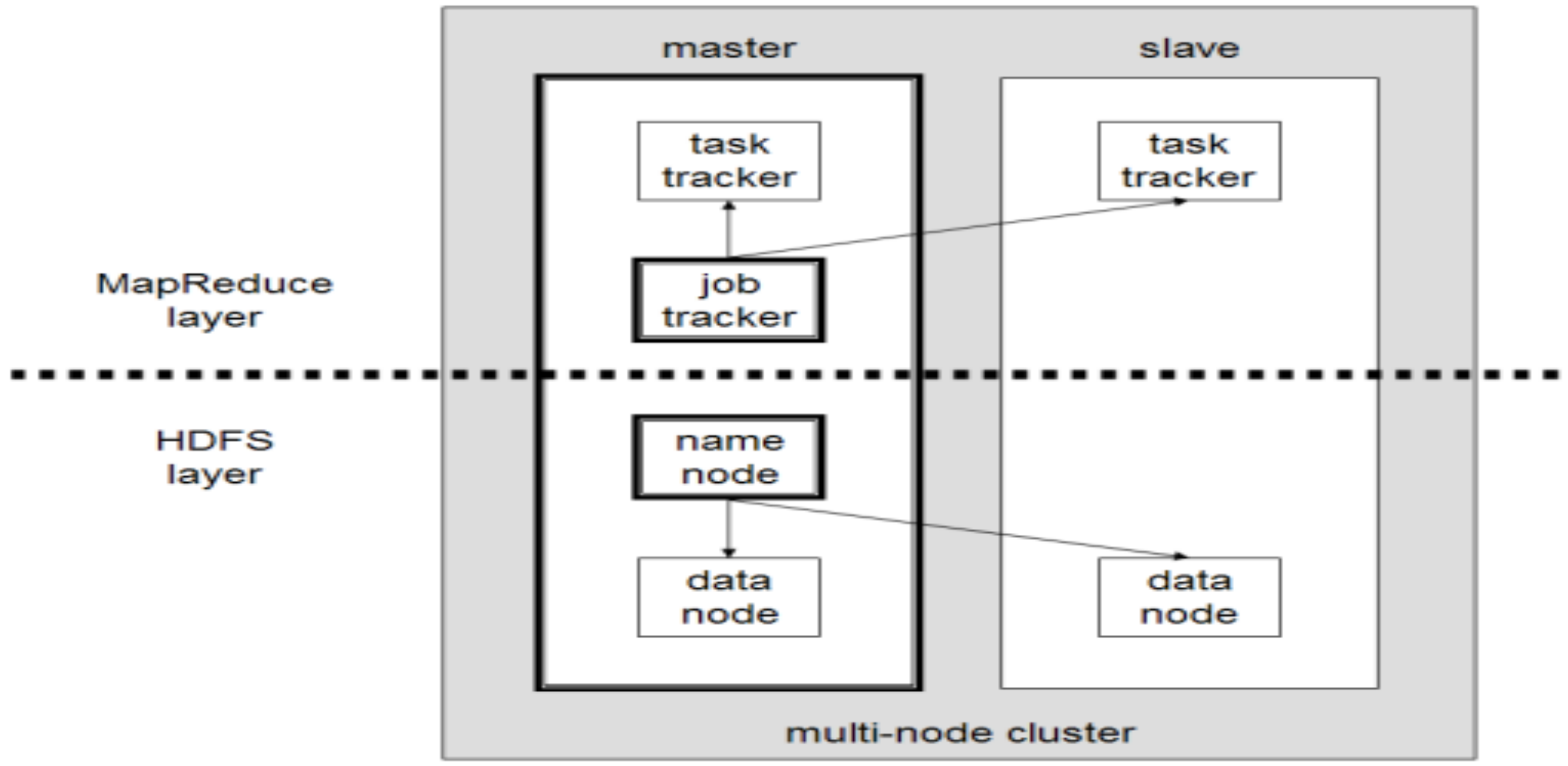
Features:
- Provenance tracking
- Annotation management
- ….

What is Hadoop?

- Hadoop is a software framework for *distributed processing* of *large datasets* across *large clusters* of computers
  - ***Large datasets*** → Terabytes or petabytes of data
  - ***Large clusters*** → hundreds or thousands of nodes
- Hadoop is open-source implementation for Google ***MapReduce***
- Hadoop is based on a simple programming model called *MapReduce*
- Hadoop is based on a simple data model, *any data will fit*

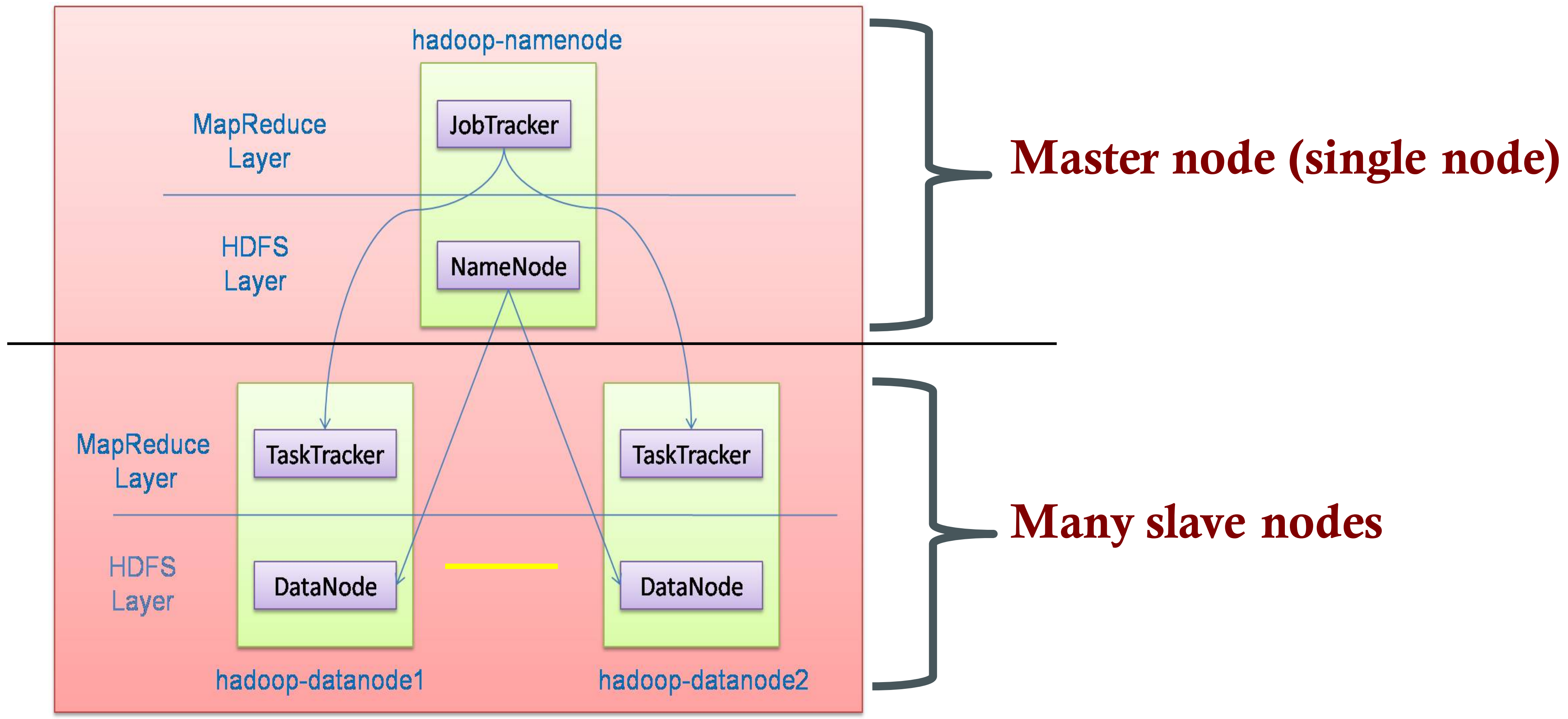
What is Hadoop?

- **Hadoop framework consists on two main layers**
  - Distributed file system (HDFS)
  - Execution engine (MapReduce)



Hadoop Architecture

- Hadoop is designed as a *master-slave* *shared-nothing* architecture



Design principles of Hadoop

- Need to process big data
- Need to parallelize computation across thousands of nodes
- **Commodity hardware**
  - Large number of low-end cheap machines working in parallel to solve a computing problem
- This is in contrast to **Parallel DBs**
  - Small number of high-end expensive machines

Design principles of Hadoop

- **Automatic parallelization & distribution**
  - Hidden from the end-user
- **Fault tolerance and automatic recovery**
  - Nodes/tasks will fail and will recover automatically
- **Clean and simple programming abstraction**
  - Users only provide two functions “map” and “reduce”

RHadoop

- install.packages(c('rJava','RJSONIO', 'itertools', 'digest','Rcpp','httr','functional','devtools', 'plyr','reshape2')

- Sys.setenv("HADOOP_CMD"="/usr/local/Cellar/hadoop/2.7.1/bin/hadoop")

- Sys.setenv("HADOOP_STREAMING"="/usr/local/Cellar/hadoop/2.7.1/libexec/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar")

- Sys.getenv("HADOOP_CMD")

- Sys.setenv("HADOOP_HOME"="/usr/local/Cellar/hadoop/2.7.1")

Install RHadoop

- Installing RHadoop [rhdfs, rmr, rhbase]
  1. Download RHadoop packages from GitHub repository of Revolution Analytics: https://github.com/RevolutionAnalytics/RHadoop
    - rmr: [rmr-2.2.2.tar.gz]
    - rhdfs: [rhdfs-1.6.0.tar.gz]
    - rhbase: [rhbase-1.2.0.tar.gz]
- 2. Installing packages.
- For rmr we use: R CMD INSTALL rmr-2.2.2.tar.gz
- For rhdfs we use: R CMD INSTALL rmr-2.2.2.tar.gz
- For rhbase we use: R CMD INSTALL rhbase-1.2.0.tar.gz

gdp data

- library(rmr2)
- library(rhdfs)
- gdp <- NA
- gdp <- read.csv("~/Downloads/GDP.csv")
- gdp <- gdp[,1:4]
- gdp$GDP <- as.double(gsub(",","",gdp$GDP))
- head(gdp)

Setup Map-Reduce Function

- hdfs.init()
- gdp.values <- to.dfs(gdp)
- aaplRevenue = 181890

- gdp.map.fn <- function(k,v) {
-     key <- ifelse(v[4] < aaplRevenue, "less", "greater")
-     keyval(key, 1)
- }
- count.reduce.fn <- function(k,v) {
-     keyval(k, length(v))
- }

Run Map-Reduce Function

- count <- mapreduce(input=gdp.values, map = gdp.map.fn, reduce = count.reduce.fn)
- from.dfs(count)