

R Introductory Session

Michael Hahsler

March 4, 2009

Contents

1	Introduction	2
1.1	Getting Help	2
2	Basic Data Types	2
2.1	Vector	2
2.2	Matrix	4
2.3	List	6
2.4	Coercion Between Data Types	8
3	Reading/Writing Data and Saving Plots	9
3.1	Reading/Writing Data	9
3.2	Creating Plots for Documents	9
4	Programming Elements	9
4.1	Control Structures	9
4.2	Functions	11
4.3	Object orientation	11
5	Applications	12
5.1	Inspecting Data	12
5.2	Linear Regression	16
5.3	Cluster Analysis	18
5.4	Hierarchical Clustering	20
5.5	Classification	21
6	Writing Extension Packages	23

1 Introduction

R is a free software environment for statistical computing and graphics.

<http://www.r-project.org/>

Manuals and extension packages can be obtained from CRAN at

<http://cran.r-project.org/>

1.1 Getting Help

Read “An Introduction to R” from the CRAN manuals section.

In R online documentation is available with `?`, `??`, `help()`, etc.

For a description of packages that could be interesting for a particular task (e.g., machine learning) go to the “Task Views” subsection on the CRAN web site.

2 Basic Data Types

2.1 Vector

```
R> a <- c(1, 3, 5:10)
```

```
R> a
```

```
[1] 1 3 5 6 7 8 9 10
```

```
R> length(a)
```

```
[1] 8
```

```
R> a[1]
```

```
[1] 1
```

```
R> a[-2]
```

```
[1] 1 5 6 7 8 9 10
```

```
R> a[1:3]
```

```
[1] 1 3 5
```

```
R> mean(a)
```

```
[1] 6.125
```

```
R> summary(a)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
1.000  4.500   6.500   6.125  8.250  10.000
```

```
R> a <- a - 1
```

```
R> a
```

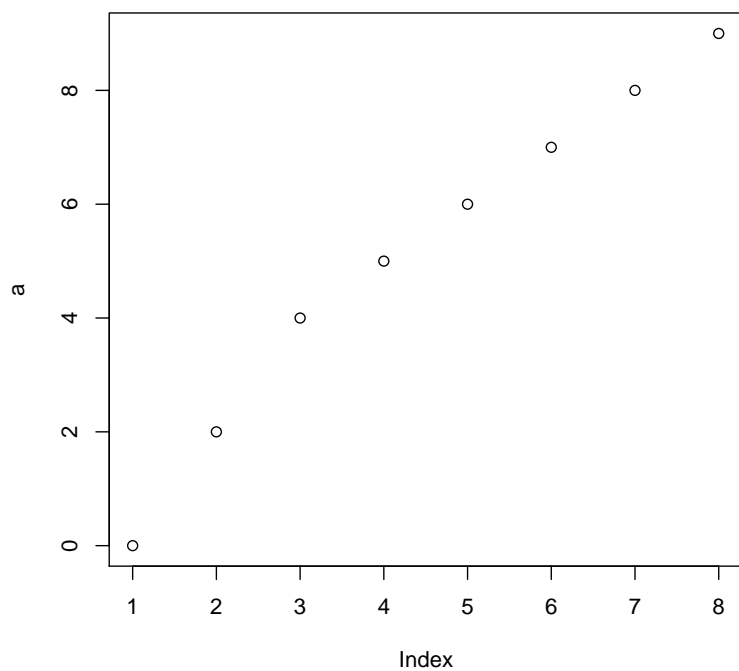
```
[1] 0 2 4 5 6 7 8 9
```

```
R> names(a) <- 1:length(a)
```

```
R> a
```

```
 1 2 3 4 5 6 7 8
0 2 4 5 6 7 8 9
```

```
R> plot(a)
```



2.2 Matrix

```
R> m <- matrix(1:25, ncol = 5, byrow = FALSE)
R> dim(m)
```

```
[1] 5 5
```

```
R> colnames(m) <- paste("col", 1:ncol(m))
R> rownames(m) <- paste("row", 1:nrow(m))
R> m[1, 1:3]
```

```
col 1 col 2 col 3
     1     6    11
```

```
R> m[1, ]
```

```
col 1 col 2 col 3 col 4 col 5
     1     6    11    16    21
```

```
R> m
```

```
      col 1 col 2 col 3 col 4 col 5
row 1     1     6    11    16    21
row 2     2     7    12    17    22
row 3     3     8    13    18    23
row 4     4     9    14    19    24
row 5     5    10    15    20    25
```

```
R> t(m)
```

```
      row 1 row 2 row 3 row 4 row 5
col 1     1     2     3     4     5
col 2     6     7     8     9    10
col 3    11    12    13    14    15
col 4    16    17    18    19    20
col 5    21    22    23    24    25
```

```
R> m * m
```

```
      col 1 col 2 col 3 col 4 col 5
row 1     1    36   121   256   441
row 2     4    49   144   289   484
row 3     9    64   169   324   529
row 4    16    81   196   361   576
row 5    25   100   225   400   625
```

```
R> m %*% m
```

```
      col 1 col 2 col 3 col 4 col 5
row 1  215  490  765 1040 1315
row 2  230  530  830 1130 1430
row 3  245  570  895 1220 1545
row 4  260  610  960 1310 1660
row 5  275  650 1025 1400 1775
```

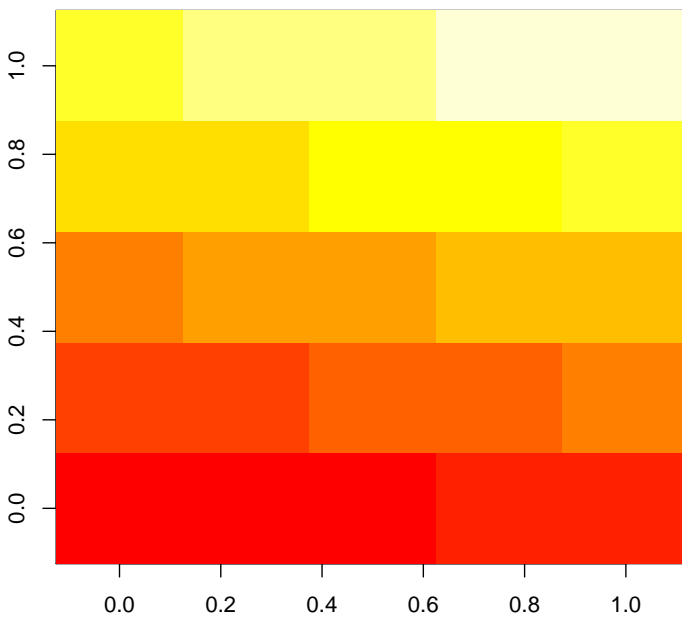
```
R> colSums(m)
```

```
col 1 col 2 col 3 col 4 col 5
  15   40   65   90  115
```

```
R> diag(m)
```

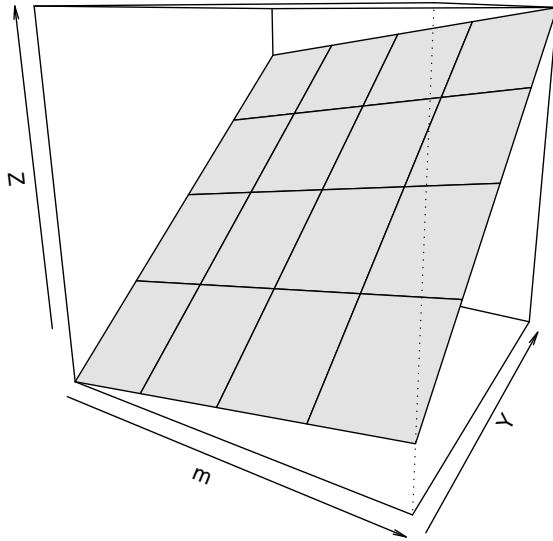
```
[1]  1  7 13 19 25
```

```
R> image(m)
```



3D plot

```
R> persp(m, shade = 0.1, theta = 30)
```



2.3 List

```
R> l <- list(a = 1:5, b = c("hello", "world"), c = matrix(1:10,  
+ ncol = 2))  
R> l
```

```
$a  
[1] 1 2 3 4 5
```

```
$b  
[1] "hello" "world"
```

```
$c  
  [,1] [,2]  
[1,]  1   6  
[2,]  2   7  
[3,]  3   8  
[4,]  4   9  
[5,]  5  10
```

```
R> l[[2]]
```

```
[1] "hello" "world"
```

```
R> l$b
```

```
[1] "hello" "world"
R> l[["b"]]
[1] "hello" "world"
R> str(l)
List of 3
 $ a: int [1:5] 1 2 3 4 5
 $ b: chr [1:2] "hello" "world"
 $ c: int [1:5, 1:2] 1 2 3 4 5 6 7 8 9 10
```

```
R> rev(l)
$c
  [,1] [,2]
[1,]   1   6
[2,]   2   7
[3,]   3   8
[4,]   4   9
[5,]   5  10
```

```
$b
[1] "hello" "world"
```

```
$a
[1] 1 2 3 4 5
```

```
R> lapply(l, rev)
```

```
$a
[1] 5 4 3 2 1
```

```
$b
[1] "world" "hello"
```

```
$c
[1] 10 9 8 7 6 5 4 3 2 1
```

Data.frames are special lists where each element has the same length

```
R> df <- data.frame(number = 1:3, letter = c("A", "B", "C"))
R> df
```

```
  number letter
1      1      A
2      2      B
3      3      C
```

```
R> df$letter
```

```
[1] A B C
Levels: A B C
```

Typically input data is stored as a data.frame.

2.4 Coercion Between Data Types

```
R> m
```

```
      col 1 col 2 col 3 col 4 col 5
row 1     1     6    11    16    21
row 2     2     7    12    17    22
row 3     3     8    13    18    23
row 4     4     9    14    19    24
row 5     5    10    15    20    25
```

```
R> class(m)
```

```
[1] "matrix"
```

```
R> m_df <- as.data.frame(m)
```

```
R> m_df
```

```
      col 1 col 2 col 3 col 4 col 5
row 1     1     6    11    16    21
row 2     2     7    12    17    22
row 3     3     8    13    18    23
row 4     4     9    14    19    24
row 5     5    10    15    20    25
```

```
R> class(m_df)
```

```
[1] "data.frame"
```

```
R> as.vector(m)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
[23] 23 24 25
```

```
R> as.character(m)
```

```
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12" "13"
[14] "14" "15" "16" "17" "18" "19" "20" "21" "22" "23" "24" "25"
```

```
R> as.logical(m)
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[14] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

3 Reading/Writing Data and Saving Plots

3.1 Reading/Writing Data

try ? read.table and ? write.table

```
R> write.csv(m, file = "matrix.csv")
R> m2 <- read.csv("matrix.csv")
R> m2
```

```
      X col.1 col.2 col.3 col.4 col.5
1 row 1     1     6    11    16    21
2 row 2     2     7    12    17    22
3 row 3     3     8    13    18    23
4 row 4     4     9    14    19    24
5 row 5     5    10    15    20    25
```

3.2 Creating Plots for Documents

Create a pdf file

```
R> data_rand <- cbind(x = rnorm(100), y = rnorm(100))
R> pdf(file = "myplot.pdf")
R> plot(data_rand, xlim = c(-4, 4), ylim = c(-4, 4))
R> dev.off()
```

```
pdf
  2
```

Create a png file

```
R> png(file = "myplot.png")
R> plot(data_rand, xlim = c(-4, 4), ylim = c(-4, 4))
R> dev.off()
```

```
pdf
  2
```

4 Programming Elements

4.1 Control Structures

```
R> a <- c()
R> for (i in 1:10) {
+   a <- append(a, 100 + i)
+ }
R> a
```

```
[1] 101 102 103 104 105 106 107 108 109 110
```

Better

```
R> a <- numeric(10)
R> for (i in 1:10) {
+   a[i] <- 100 + i
+ }
R> a
```

```
[1] 101 102 103 104 105 106 107 108 109 110
```

while and if work as expected (see ?Control)

Comparisons and logical vectors

```
R> a > 103
```

```
[1] FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
R> which(a > 103)
```

```
[1] 4 5 6 7 8 9 10
```

```
R> a[a > 103] <- NA
R> a
```

```
[1] 101 102 103 NA NA NA NA NA NA NA
```

Try to prevent loops. Most things in R are vectorized. For example to add all numbers in a vector which are divisible by 3 together.

```
R> v <- as.integer(runif(10000, 1, 100))
```

Don't use a loop like this

```
R> s <- 0
R> system.time(for (i in 1:length(v)) {
+   if (v[i]%%3 == 0)
+     s <- s + v[i]
+ })
```

```
user system elapsed
0.036 0.004 0.040
```

But do this

```
R> system.time(s <- sum(v[v%%3 == 0]))
```

```
user system elapsed
0.004 0.000 0.002
```

4.2 Functions

Defining a function

```
R> dec <- function(x) {  
+   x - 1  
+ }  
R> dec
```

```
function (x)  
{  
  x - 1  
}
```

```
R> dec(1:10)
```

```
[1] 0 1 2 3 4 5 6 7 8 9
```

A function returning a function

```
R> gen_dec <- function(i) {  
+   function(x) {  
+     x - i  
+   }  
+ }  
R> my_dec <- gen_dec(5)  
R> my_dec(1:10)
```

```
[1] -4 -3 -2 -1 0 1 2 3 4 5
```

4.3 Object orientation

R supports two ways of implementing oo called S3 and S4. In S3 a generic functions can have implementations for different objects and be dispatched depending on the object they operate on (typically the first argument).

The name convention for methods is
<function name>.<object type>

For example see plot. Type plot. and hit tab twice. Then type e.g. ? plot.default.

Defining and using a class

```
R> v <- 1:5  
R> v
```

```
[1] 1 2 3 4 5
```

```
R> class(v)
```

```
[1] "integer"
```

```
R> class(v) <- "myclass"
R> v
```

```
[1] 1 2 3 4 5
attr(,"class")
[1] "myclass"
```

```
R> attributes(v)
```

```
$class
[1] "myclass"
```

```
R> print.myclass <- function(x) {
+   cat("My class: ", unclass(x), "\n")
+ }
R> v
```

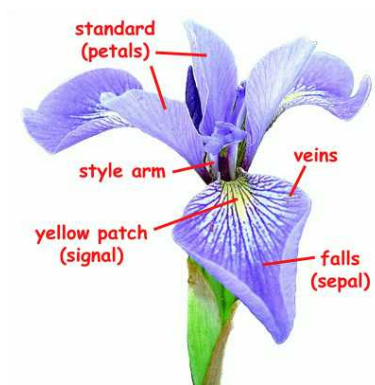
```
My class: 1 2 3 4 5
```

S4 provides oo-support with formal class descriptions.

5 Applications

5.1 Inspecting Data

This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are *Iris setosa*, *versicolor*, and *virginica*.



```
R> data(iris)
R> 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

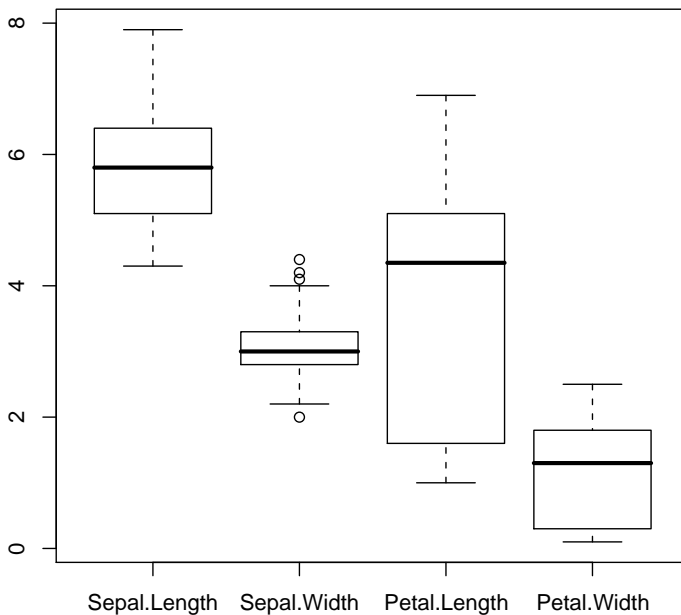
```
R> summary(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300
Median :5.800	Median :3.000	Median :4.350	Median :1.300
Mean :5.843	Mean :3.057	Mean :3.758	Mean :1.199
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500

Species

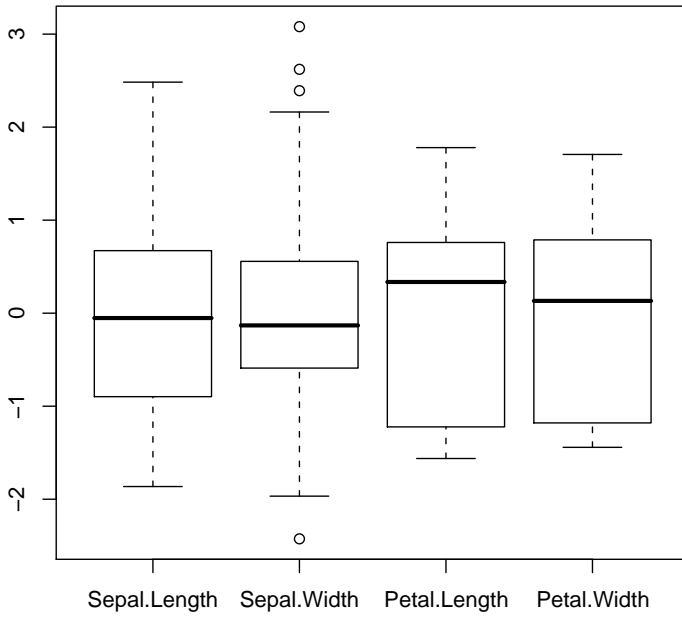
setosa	:50
versicolor	:50
virginica	:50

```
R> data <- iris[, -5]
R> class <- iris[, 5]
R> boxplot(data)
```



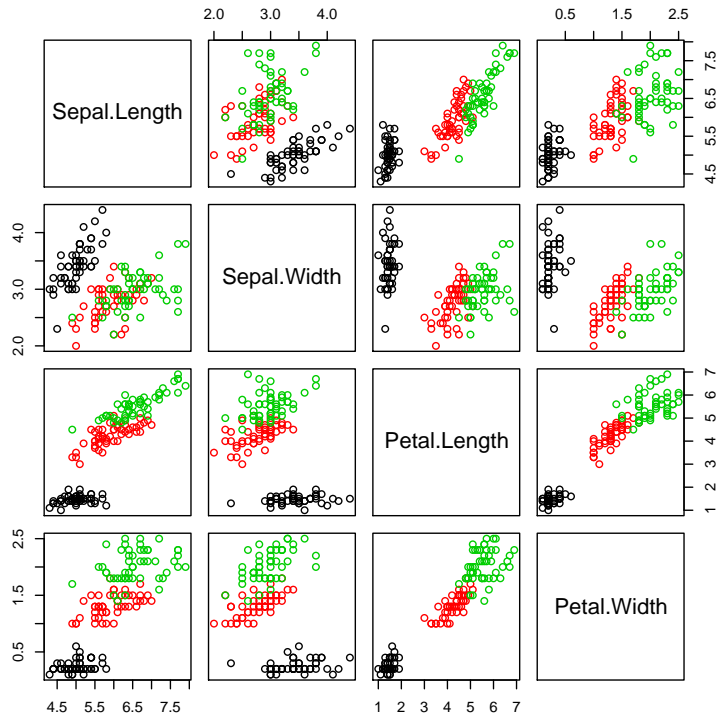
Data can be scaled

```
R> data_s <- scale(data)
R> boxplot(as.data.frame(data_s))
```



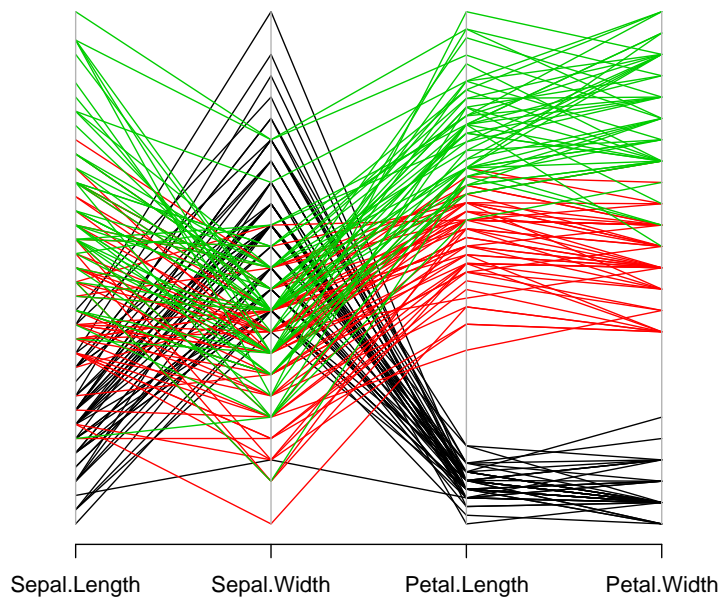
We can look at the data as a trellis plot.

```
R> plot(data, col = class)
```



We can look at a parallel coordinates plot

```
R> library("MASS")
R> parcoord(data, col = class, pch = 0)
```



5.2 Linear Regression

We fit a linear model for sepal length based on petals on part of the data (train).

```
R> train.ind <- sample(1:nrow(iris), 100)
R> train <- iris[train.ind, ]
R> test <- iris[-train.ind, ]
```

lm uses a formula interface (many model building functions in R support this kind of interface; see <http://cran.r-project.org/doc/manuals/R-intro.html#Formulae-for-statistical-models>).

```
R> model <- lm(Sepal.Length ~ Petal.Length + Petal.Width,
+ data = train)
R> model
```

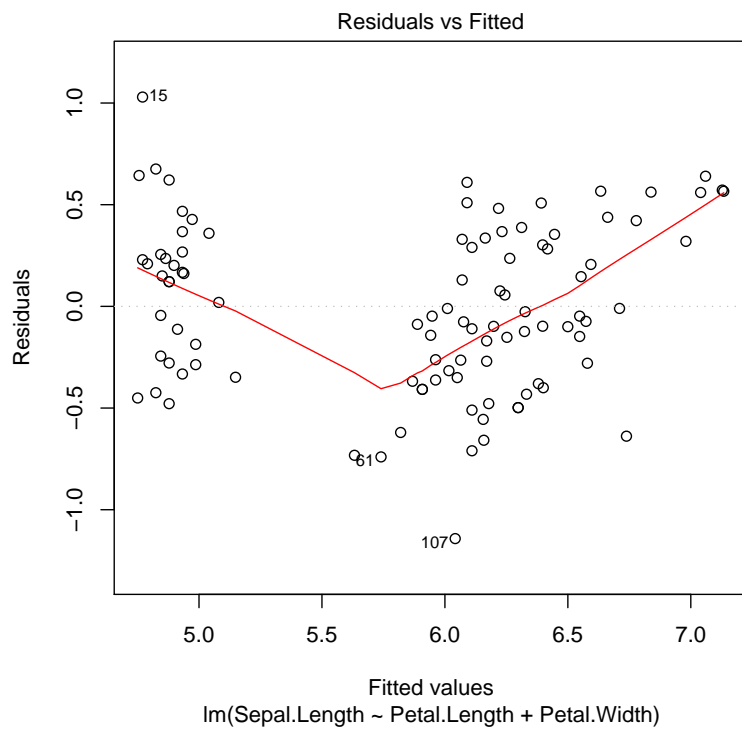
Call:

```
lm(formula = Sepal.Length ~ Petal.Length + Petal.Width, data = train)
```

Coefficients:

(Intercept)	Petal.Length	Petal.Width
4.1904	0.5402	-0.3408

```
R> plot(model, which = 1)
```



Use the linear model to predict sepal length in the test data.

```
R> Sepal.Length.predicted <- predict(model, test)
R> plot(Sepal.Length.predicted, test$Sepal.Length)
R> abline(0, 1, col = "red")
```



```
R> table(class, cl$cluster)
```

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

Use PCA for display

```
R> pc <- prcomp(data)
R> pc
```

Standard deviations:

```
[1] 2.0562689 0.4926162 0.2796596 0.1543862
```

Rotation:

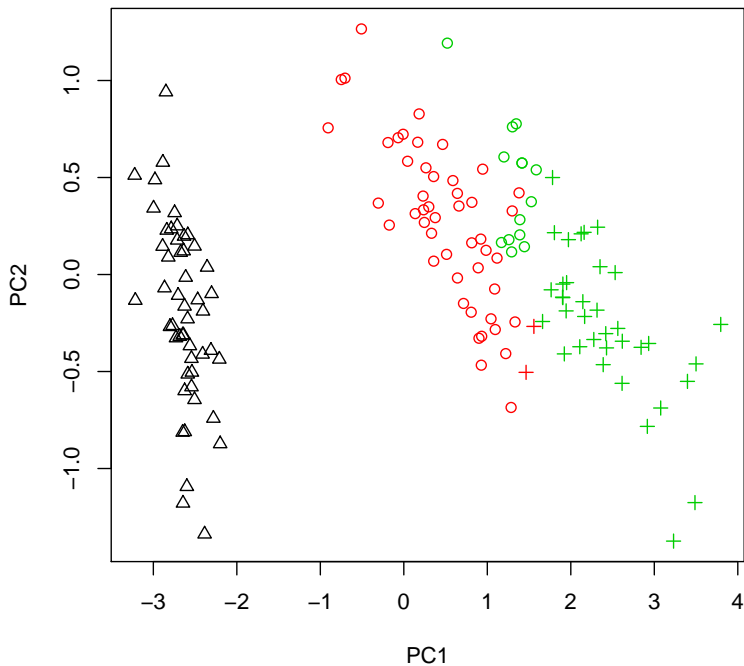
	PC1	PC2	PC3	PC4
Sepal.Length	0.36138659	-0.65658877	0.58202985	0.3154872
Sepal.Width	-0.08452251	-0.73016143	-0.59791083	-0.3197231
Petal.Length	0.85667061	0.17337266	-0.07623608	-0.4798390
Petal.Width	0.35828920	0.07548102	-0.54583143	0.7536574

```
R> summary(pc)
```

Importance of components:

	PC1	PC2	PC3	PC4
Standard deviation	2.056	0.4926	0.2797	0.15439
Proportion of Variance	0.925	0.0531	0.0171	0.00521
Cumulative Proportion	0.925	0.9777	0.9948	1.00000

```
R> plot(pc$x, col = class, pch = cl$cluster)
```



5.4 Hierarchical Clustering

```
R> d <- dist(data)
R> str(d)
```

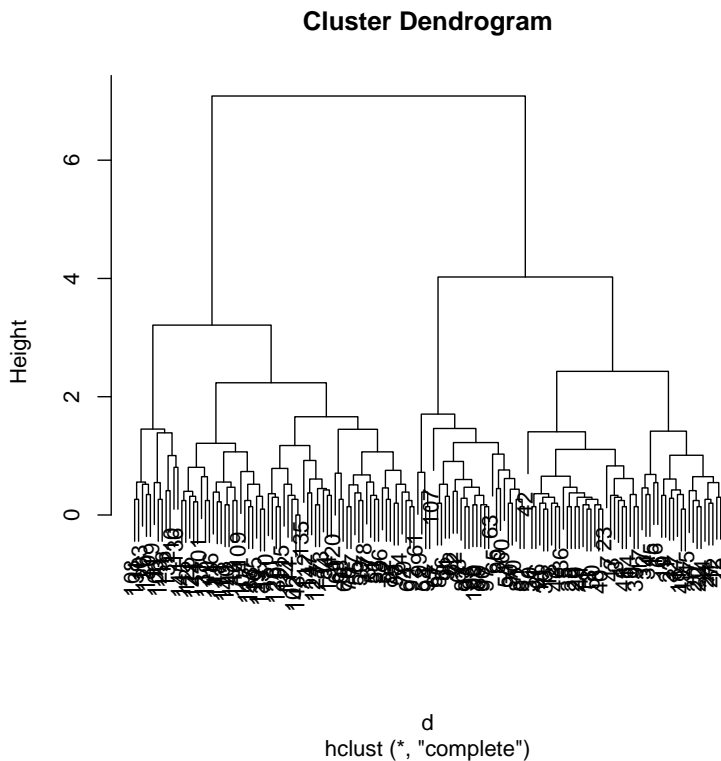
```
Class 'dist'  atomic [1:11175] 0.539 0.51 0.648 0.141 0.616 ...
..- attr(*, "Size")= int 150
..- attr(*, "Diag")= logi FALSE
..- attr(*, "Upper")= logi FALSE
..- attr(*, "method")= chr "euclidean"
..- attr(*, "call")= language dist(x = data)
```

```
R> hc <- hclust(d)
R> hc
```

```
Call:
hclust(d = d)
```

```
Cluster method : complete
Distance       : euclidean
Number of objects: 150
```

```
R> plot(hc)
```



```
R> d <- dist(data)
R> labels <- cutree(hc, 3)
R> table(class, labels)
```

	labels		
class	1	2	3
setosa	50	0	0
versicolor	0	23	27
virginica	0	49	1

5.5 Classification

Here we use the package “caret” so we have to install it by typing `install.packages("caret", depend = TRUE)`.

We use 90% as the training set and 10% as the test set.

```
R> train_index <- sample(1:nrow(data), 0.9 * nrow(data),
+   replace = FALSE)
R> train_data <- data[train_index, ]
R> train_class <- class[train_index]
R> test_data <- data[-train_index, ]
R> test_class <- class[-train_index]
R> library("caret")
R> knn <- knn3(as.matrix(train_data), train_class, k = 5)
R> knn
```

5-nearest neighbor classification model

Call:

```
knn3.matrix(x = as.matrix(train_data), y = train_class, k = 5)
```

Training set class distribution:

	setosa	versicolor	virginica
	41	46	48

```
R> class(knn)
```

```
[1] "knn3"
```

Now we can use the model to make predictions for our test set (most model types in R have a predict method).

```
R> pred <- predict(knn, test_data, type = "class")
R> conf_tab <- table(test_class, pred)
R> conf_tab
```

	pred		
test_class	setosa	versicolor	virginica
setosa	9	0	0
versicolor	0	4	0
virginica	0	0	2

```
R> sum(diag(conf_tab))/sum(conf_tab)
```

```
[1] 1
```

Since knn is of class knn3 use ? predict.knn3 to get help for predict for class knn3.

Caret supports also a formula interface Let's use caret to search for parameters and do 10-fold cross validation.

```
R> knn_opt <- train(Species ~ ., data = iris, method = "knn",  
+   trControl = trainControl(method = "cv"), tuneLength = 5)
```

```
Model 1: k= 5  
Model 2: k= 7  
Model 3: k= 9  
Model 4: k=11  
Model 5: k=13
```

```
R> knn_opt
```

Call:

```
train.formula(form = Species ~ ., data = iris, method = "knn",  
  trControl = trainControl(method = "cv"), tuneLength = 5)
```

```
150 samples  
4 predictors
```

```
summary of cross-validation (10 fold) sample sizes:  
 135, 135, 135, 135, 135, 135, ...
```

```
cv resampled training results across tuning parameters:
```

k	Accuracy	Kappa	Accuracy SD	Kappa SD	Selected
5	0.967	0.95	0.0351	0.0527	
7	0.967	0.95	0.0471	0.0707	
9	0.973	0.96	0.0466	0.0699	
11	0.973	0.96	0.0466	0.0699	
13	0.973	0.96	0.0466	0.0699	*

Accuracy was used to select the optimal model using the largest value.

The final values used in the model were k = 13.

Only use Petal.Length and Sepal.Length for classification

```
R> knn_opt <- train(Species ~ Petal.Length + Sepal.Length,  
+   data = iris, method = "knn", trControl = trainControl(method = "cv"),  
+   tuneLength = 5)
```

```
Model 1: k= 5  
Model 2: k= 7  
Model 3: k= 9  
Model 4: k=11  
Model 5: k=13
```

```
R> knn_opt
```

```
Call:
```

```
train.formula(form = Species ~ Petal.Length + Sepal.Length, data = iris,  
  method = "knn", trControl = trainControl(method = "cv"),  
  tuneLength = 5)
```

```
150 samples
```

```
2 predictors
```

```
summary of cross-validation (10 fold) sample sizes:
```

```
135, 135, 135, 135, 135, 135, ...
```

```
cv resampled training results across tuning parameters:
```

k	Accuracy	Kappa	Accuracy SD	Kappa SD	Selected
5	0.947	0.92	0.0689	0.103	
7	0.953	0.93	0.0706	0.106	
9	0.953	0.93	0.0706	0.106	
11	0.953	0.93	0.0706	0.106	
13	0.953	0.93	0.0706	0.106	*

```
Accuracy was used to select the optimal model using the largest value.
```

```
The final values used in the model were k = 13.
```

```
For more models try ? train
```

6 Writing Extension Packages

```
We will do that another time.
```

7 Contact Information

```
For questions contact me in my office or per email at mhahsler@lyle.smu.edu
```