Data Pre-Processing in R

L. Torgo

ltorgo@knoyda.com
KNOYDA, Know Your Data!

Jul, 2019

What is Data Pre-Processing?

Data Pre-Processing

Set of steps that may be necessary to carry out before any further analysis takes place on the available data

Some Motivations for Data Pre-Processing

- Several data mining methods are sensitive to the scale and/or type of the variables
  - Different variables (columns of our data sets) may have rather different scales
  - Some methods are not able to handle either nominal or numeric variables
- We may need to “create” new variables to achieve our objectives
  - Sometimes we are more interested in relative values (variations) than absolute values
  - We may be aware of some domain-specific mathematical relationship among two or more variables that is important for the task
- Frequently we have data sets with unknown variable values
- Our data set may be too large for some methods to be applicable

Some of the Main Classes of Data Pre-Processing

- Data cleaning
  - Given data may be hard to read or require extra parsing efforts
- Data transformation
  - It may be necessary to change/transform some of the values of the data
- Variable creation
  - E.g. to incorporate some domain knowledge
- Dimensionality reduction
  - To make modeling possible

Illustrations of Data Cleaning in R

Making your data tidy

- Properties of tidy data sets:
  - each value belongs to a variable and an observation
  - each variable contains all values of a certain property measured across all observations
  - each observation contains all values of the variables measured for the respective case
- The properties lead to data tables where each row represents an observation and the columns represent different properties measured for each observation

A non tidy data set

StudentName	Math	English	DegreeYear
Anna	86	90	Bio|2014
John	43	75	Math|2013
Catherine	80	82	Bio|2012

- This data is about the grades on some courses of students that entered some degree in some year
- The rows are students
- The columns are the properties measured for each student:
  - name
  - subject
  - grade
  - degree
  - entrance year

Reading the data

StudentName Math English DegreeYear
Anna 86 90 Bio|2014
John 43 75 Math|2013
Catherine 80 82 Bio|2012

The contents of this file could be read as follows:

```
library(readr)
std <- read_table2("stud2.txt", col_types = cols())
std

## # A tibble: 3 x 4
##   StudentName  Math English DegreeYear
##   <chr>       <dbl>   <dbl> <chr>
## 1 Anna           86      90 Bio|2014
## 2 John           43      75 Math|2013
## 3 Catherine      80      82 Bio|2012
```

Making this data tidy

```
library(tidyr)
tstd <- gather(std, Math:English,
               key="Subject", value="Grade")
tstd

## # A tibble: 6 x 4
##   StudentName DegreeYear Subject Grade
##   <chr>       <chr>      <chr>   <dbl>
## 1 Anna        Bio|2014   Math       86
## 2 John        Math|2013  Math       43
## 3 Catherine   Bio|2012   Math       80
## 4 Anna        Bio|2014   English    90
## 5 John        Math|2013  English    75
## 6 Catherine   Bio|2012   English    82
```


Data Cleaning Tidy Data

Making this data tidy - 2

```
tstd <- separate(tstd, col="DegreeYear",
                 into=c("Degree","Year"),
                 convert = TRUE)
tstd

## # A tibble: 6 x 5
##   StudentName Degree  Year Subject Grade
##   <chr>       <chr>  <int> <chr>   <dbl>
## 1 Anna        Bio     2014 Math       86
## 2 John        Math    2013 Math       43
## 3 Catherine   Bio     2012 Math       80
## 4 Anna        Bio     2014 English    90
## 5 John        Math    2013 English    75
## 6 Catherine   Bio     2012 English    82
```

Handling Dates

- Date/time information are very common types of data
- With real-time data collection (e.g. sensors) this is even more common
- Date/time information can be provided in several different formats
- Being able to read, interpret and convert between these formats is a very frequent data pre-processing task

Package **lubridate**

- Package with many functions related with handling dates/time
- Handy for parsing and/or converting between different formats
- Some examples:

```
library(lubridate)
ymd("20151021")

## [1] "2015-10-21"

ymd("2015/11/30")

## [1] "2015-11-30"

myd("11.2012.3")

## [1] "2012-11-03"

dmy_hms("2/12/2013 14:05:01")

## [1] "2013-12-02 14:05:01 UTC"
```

Examples of using package **lubridate**

```
dates <- c(20120521, "2010-12-12", "2007/01/5", "2015-2-04",
           "Measured on 2014-12-6", "2013-7+ 25")
dates <- ymd(dates)
dates

## [1] "2012-05-21" "2010-12-12" "2007-01-05" "2015-02-04" "2014-12-06"
## [6] "2013-07-25"

data.frame(Dates=dates,WeekDay=wday(dates),nWeekDay=wday(dates,label=TRUE),
           Year=year(dates),Month=month(dates,label=TRUE))

##        Dates WeekDay nWeekDay Year Month
## 1 2012-05-21       2      Mon 2012   May
## 2 2010-12-12       1      Sun 2010   Dec
## 3 2007-01-05       6      Fri 2007   Jan
## 4 2015-02-04       4      Wed 2015   Feb
## 5 2014-12-06       7      Sat 2014   Dec
## 6 2013-07-25       5      Thu 2013   Jul
```


Conversions between time zones

- Sometimes we get dates from different time zones
- **lubridate** can help with that too
- Some examples:

```
date <- ymd_hms("20150823 18:00:05", tz="Europe/Berlin")
date

## [1] "2015-08-23 18:00:05 CEST"

with_tz(date, tz="Pacific/Auckland")

## [1] "2015-08-24 04:00:05 NZST"

force_tz(date, tz="Pacific/Auckland")

## [1] "2015-08-23 18:00:05 NZST"
```

String Processing

- Processing and/or parsing strings is frequently necessary when reading data into R
- This is particularly true when data is received in a non-standard format

String Processing - some useful packages

- Base R contains several useful functions for string processing
  - E.g. `grep`, `strsplit`, `nchar`, `substr`, etc.
- Package **stringi** provides an extensive set of useful functions for string processing
- Package **stringr** builds upon the extensive set of functions of **stringi** and provides a simpler interface covering the most common needs

String Processing - a concrete example

- Let us work through a concrete example
  - Reading the name of the variables of a problem that are provided within a text file
  - Avoiding having to type them by hand
- The UCI repository contains a large set of data sets
  - Data sets are typically provided in two separate files: one with the data, the other with information on the data set, including the names of the variables
  - This latter file is a text file in a free format
- Let us try to read the information on the names of the variables of the data set named **heart-disease**
  - Information (text file) available at `https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/heart-disease.names`

Reading in the file

- Let us start by reading the file

```
library(readr)
d <- read_lines(url("https://archive.ics.uci.edu/ml/machine-learning-d
```

- As you may check the useful information is between lines 127 and 235

```
d <- d[127:235]
head(d,2)

## [1] "      1 id: patient identification number"
## [2] "      2 ccf: social security number (I replaced this with a du

tail(d,2)

## [1] "     75 junk: not used"
## [2] "     76 name: last name of patient "
```

Processing the lines

- Trimming white space

```
library(stringr)
d <- str_trim(d)
```

- Looking carefully at the lines (strings) you will see that the lines containing some variable name all follow the pattern
  `ID name ....`
- Where `ID` is a number from 1 to 76
- So we have a number, followed by the information we want (the name of the variable), plus some optional information we do not care
- There are also some lines in the midle that describe the values of the variables and not the variables

Processing the lines (cont.)

- Regular expressions are a powerful mechanism for expressing string patterns
- They are out of the scope of this subject
  - Tutorials on regular expressions can be easily found around the Web
- Function `grep()` can be used to match strings against patterns expressed as regular expressions

```
## e.g. line (string) starting with the number 26
d[grep("^26",d)]

## [1] "26 pro (calcium channel blocker used during exercise ECG: 1 =
```

Processing the lines (cont.)

- Lines starting with the numbers 1 till 76

```
tgtLines <- sapply(1:76,function(i) d[grep(paste0("^",i),d)[1]])
head(tgtLines,2)

## [1] "1 id: patient identification number"
## [2] "2 ccf: social security number (I replaced this with a dummy va
```

- Throwing the IDs out...

```
nms <- str_split_fixed(tgtLines," ",2)[,2]
head(nms,2)

## [1] "id: patient identification number"
## [2] "ccf: social security number (I replaced this with a dummy valu
```


Processing the lines (cont.)

- Grabbing the name

```
nms <- str_split_fixed(nms,":",2)[,1]
head(nms,2)

## [1] "id"  "ccf"
```

- Final touches to handle some extra characters (e.g. check `nms[6:8]`)

```
nms <- str_split_fixed(nms," ",2)[,1]
head(nms,2)

## [1] "id"  "ccf"

tail(nms,2)

## [1] "junk" "name"
```

Dealing with Missing/Unknown Values

- Missing variable values are a frequent problem in real world data sets

Some Possible Strategies

- Remove all lines in a data set with some unknown value
- Fill-in the unknowns with the most common value (a statistic of centrality)
- Fill-in with the most common value on the cases that are more "similar" to the one with unknowns
- Explore eventual correlations between variables
- etc.

Some illustrations in R

```
load("carInsurance.Rdata") # car insurance dataset (get it from class web page)
```

```
library(DMwR2)
head(ins[!complete.cases(ins),],3)

##   symb normLoss        make fuelType aspiration nDoors   bodyStyle
## 1    3       NA alfa-romero      gas        std    two convertible
## 2    3       NA alfa-romero      gas        std    two convertible
## 3    1       NA alfa-romero      gas        std    two   hatchback
##   driveWheels engineLocation wheelBase length width height curbWeight
## 1         rwd          front      88.6  168.8  64.1   48.8       2548
## 2         rwd          front      88.6  168.8  64.1   48.8       2548
## 3         rwd          front      94.5  171.2  65.5   52.4       2823
##   engineType nrCylinds engineSize fuelSystem bore stroke compressionRatio
## 1       dohc      four        130       mpfi 3.47   2.68                9
## 2       dohc      four        130       mpfi 3.47   2.68                9
## 3       ohcv       six        152       mpfi 2.68   3.47                9
##   horsePower peakRpm cityMpg highwayMpg price
## 1        111    5000      21         27 13495
## 2        111    5000      21         27 16500
## 3        154    5000      19         26 16500
```

Some illustrations in R (2)

```
nrow(ins[!complete.cases(ins),])

## [1] 46

noNA.ins <- na.omit(ins)   # Option 1
nrow(noNA.ins[!complete.cases(noNA.ins),])

## [1] 0

noNA.ins <- centralImputation(ins)   # Option 2
nrow(noNA.ins[!complete.cases(noNA.ins),])

## [1] 0

noNA.ins <- knnImputation(ins,k=10)   # Option 3
nrow(noNA.ins[!complete.cases(noNA.ins),])

## [1] 0
```


Transformations of Variables in R

Standardizing Numeric Variables

Goal

Make all variables have the same scale - usually a scale where all have mean 0 and standard deviation 1

$$y = \frac{x - \bar{x}}{\sigma_x}$$

```
load("carInsurance.Rdata") # car insurance data (check course web page)
```

```
norm.ins <- ins
norm.ins[,c(10:14,17,19:26)] <- scale(norm.ins[,c(10:14,17,19:26)])
```


Discretization of Numeric Variables

- Sometimes it makes sense to discretize a numeric variable
- This can also reduce computational complexity in some cases
- Let us see an example of discretizing a variable into 4 intervals.
- Two examples of possible strategies
  - Equal-width

```
data(Boston, package="MASS") # The Boston Housing data set
Boston$age <- cut(Boston$age,4)
table(Boston$age)

##
##  (2.8,27.2] (27.2,51.4] (51.4,75.7]  (75.7,100]
##          51          97          96         262
```

  - Equal-frequency

```
data(Boston, package="MASS") # The Boston Housing data set
Boston$age <- cut(Boston$age,quantile(Boston$age,probs=seq(0,1,.25)))
table(Boston$age)

##
##    (2.9,45]   (45,77.5] (77.5,94.1]  (94.1,100]
##         126         126         126         127
```

Creating Variables

Creating Variables

- May be necessary to properly address our data mining goals
- Several factors may motivate variable creation:
  - Express known relationships between existing variables
  - Overcome limitations of some data mining tools, like for instance:
    - dependencies between cases (rows)
    - etc.

Handling Case Dependencies

- Observations in a data set sometimes are not independent
- Frequent dependencies include time, space or even space-time
- These effects may have a strong impact on the data mining process
- Two main ways of handling this issue:
  - Constrain ourselves to tools that handle these dependencies directly
  - Create variables that express the dependency relationships

Working with relative values instead of absolute values

Why?

Frequent technique that is used in time series analysis to avoid trend effects

$$y_i = \frac{x_i - x_{i-1}}{x_{i-1}}$$

```
x <- rnorm(100,mean=100,sd=3)
head(x)

## [1]  97.52625 100.19782  99.16785 100.23747 100.38753 101.75377

vx <- diff(x)/x[-length(x)]
head(vx)

## [1]  0.027393332 -0.010279347  0.010785978  0.001496962  0.013609686
## [6] -0.031358624
```

An example with real-world time series data

The S&P 500 stock market index

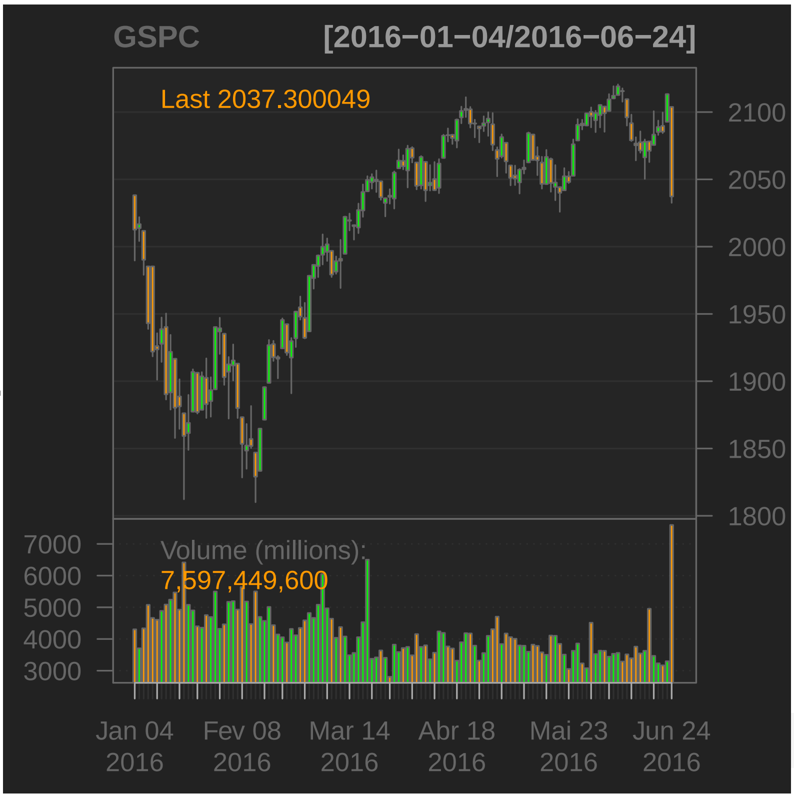
```
library(quantmod)  # extra package
getSymbols('^GSPC',from='2016-01-01')

## [1] "GSPC"

head(GSPC,3)

##             GSPC.Open GSPC.High GSPC.L
## 2016-01-04   2038.20   2038.20  1989.
## 2016-01-05   2013.78   2021.94  2004.
## 2016-01-06   2011.71   2011.71  1979.
##             GSPC.Adjusted
## 2016-01-04        2012.66
## 2016-01-05        2016.71
## 2016-01-06        1990.26

candleChart(GSPC )
```



An example with real-world time series data (2)

The S&P 500 stock market index

```
head(Cl(GSPC))

##             GSPC.Close
## 2016-01-04     2012.66
## 2016-01-05     2016.71
## 2016-01-06     1990.26
## 2016-01-07     1943.09
## 2016-01-08     1922.03
## 2016-01-11     1923.67

head(Delt(Cl(GSPC)))

##             Delt.1.arithmetic
## 2016-01-04                 NA
## 2016-01-05       0.0020122261
## 2016-01-06      -0.0131153966
## 2016-01-07      -0.0237004430
## 2016-01-08      -0.0108383746
## 2016-01-11       0.0008532723
```

Handling Time Order Between Cases

Why?

- There is a time order between the cases
- Some tools shuffle the cases, or are not able to use the information about this order

Time Delay Embedding

- Create variables whose values are the value of the time series in previous time steps
- Standard tools find relationships between variables
- If we have variables whose values are the value of the same variable but on different time steps, the tools will be able to model the time relationships with these embeddings
- Note that similar “tricks” can be done with space and space-time dependencies

An example of creating an embed data set in R

```
library(DMwR2)
library(quantmod)
dat <- getSymbols('^GSPC',from=Sys.Date()-90,auto.assign=FALSE)
ts <- na.omit(Delt(Cl(dat))) # because 1st return is NA
embTS <- createEmbedDS(ts, emb = 3)
head(embTS)
```

```
##                        T           T_1           T_2
## 2017-07-28 -0.0013411155 -0.0009726882  0.0002826638
## 2017-07-31 -0.0007281457 -0.0013411155 -0.0009726882
## 2017-08-01  0.0024491150 -0.0007281457 -0.0013411155
## 2017-08-02  0.0004926484  0.0024491150 -0.0007281457
## 2017-08-03 -0.0021836541  0.0004926484  0.0024491150
## 2017-08-04  0.0018891035 -0.0021836541  0.0004926484
```

```
head(ts)
```

```
##            Delt.1.arithmetic
## 2017-07-26      0.0002826638
## 2017-07-27     -0.0009726882
## 2017-07-28     -0.0013411155
## 2017-07-31     -0.0007281457
## 2017-08-01      0.0024491150
## 2017-08-02      0.0004926484
```


Feature Selection

Motivations

- Some data mining methods may be unable to handle very large data sets
- The computation time to obtain a certain model may be too large for the application
- We may want simpler models
- We may suspect some features are irrelevant
- We may suspect that some features are highly correlated
- etc.

Some strategies

- Filter methods
  - looking at variables individually and asserting their value using some metric
  - rank and / or filter based on these scores
- Wrapper methods
  - Take into consideration what we are going to do with the data (e.g. the models we are going to learn)
  - Carry out an iterative search process where we try different subsets of features, apply the analysis, and check the results
  - Based on these results select the best subset

Other possible taxonomy of the methods

- Unsupervised methods
  - Use only the values of each variable to score it
- Supervised methods
  - Use some metric that relates the values of a feature with the values of some target variable (e.g. how they are correlated)

Feature selection in R

- R contains several packages related with feature selection
- Some good examples
  - Package **FSelector**
  - Package **CORElearn**

Some illustrations with CORElearn

Classification tasks

```
library(CORElearn)
data(iris)
attrEval(Species ~ ., iris, estimator="GainRatio")

## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.5919339    0.3512938    1.0000000    1.0000000

attrEval(Species ~ ., iris, estimator="InfGain")

## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.5572327    0.2831260    0.9182958    0.9182958

attrEval(Species ~ ., iris, estimator="Gini")

## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.2277603    0.1269234    0.3333333    0.3333333
```

Many more metrics!

Classification tasks

```r
infoCore(what="attrEval")
```

```
##  [1] "ReliefFequalK"      "ReliefFexpRank"     "ReliefFbestK"
##  [4] "Relief"             "InfGain"            "GainRatio"
##  [7] "MDL"                "Gini"               "MyopicReliefF"
## [10] "Accuracy"           "ReliefFmerit"       "ReliefFdistance"
## [13] "ReliefFsqrDistance" "DKM"                "ReliefFexpC"
## [16] "ReliefFavgC"        "ReliefFpe"          "ReliefFpa"
## [19] "ReliefFsmp"         "GainRatioCost"      "DKMcost"
## [22] "ReliefKukar"        "MDLsmp"             "ImpurityEuclid"
## [25] "ImpurityHellinger"  "UniformDKM"         "UniformGini"
## [28] "UniformInf"         "UniformAccuracy"    "EqualDKM"
## [31] "EqualGini"          "EqualInf"           "EqualHellinger"
## [34] "DistHellinger"      "DistAUC"            "DistAngle"
## [37] "DistEuclid"
```


Regression tasks illustrations with CORElearn

```r
data(algae, package ="DMwR2")
attrEval(a1 ~ ., algae[,1:12], estimator="MSEofMean")
```

```
##    season      size     speed      mxPH      mnO2        Cl       NO3
## -453.2142 -395.9696 -413.5873 -413.3519 -395.2823 -252.7300 -380.6412
##       NH4      oPO4       PO4      Chla
## -291.0525 -283.3738 -272.9903 -303.5737
```

```r
attrEval(a1 ~ ., algae[,1:12], estimator="RReliefFexpRank")
```

```
##       season         size        speed         mxPH         mnO2
## -0.031203465 -0.028139035 -0.035271926  0.080825823 -0.072103230
##           Cl          NO3          NH4         oPO4          PO4
## -0.152077352 -0.011462467 -0.009879109 -0.134034483 -0.076488066
##         Chla
## -0.142442935
```

Other measures for regression

```
infoCore(what="attrEvalReg")

## [1] "RReliefFequalK"       "RReliefFexpRank"      "RReliefFbestK"
## [4] "RReliefFwithMSE"      "MSEofMean"            "MSEofModel"
## [7] "MAEofModel"           "RReliefFdistance"     "RReliefFsqrDistance"
```


Reducing the number of variables through PCA

Principal Component Analysis (PCA)

- General Idea : replace the set of variables by a new (smaller) set where most of the “information” on the problem is still expressed
- Goal : find a new set of axes onto which we will project the original data points

- On PCA the new set of axes are formed linear combinations of the original variables
- We search for the linear combinations that “explain” most of the variability that existed among the data points on the original axes
- If we are “lucky” with a few of these new axes (ideally two for easy data visualization), we are able to explain most of the variability on the original data
- Each original observation is then “projected” into these new axes

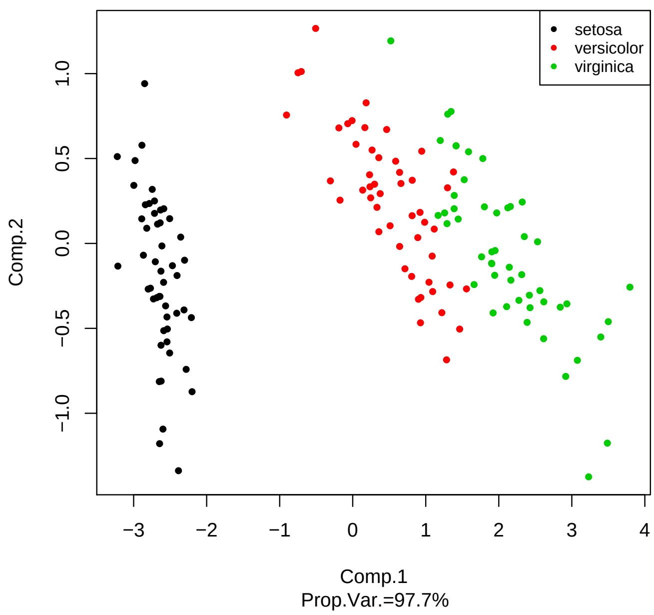
PCA - the method

- Find a first linear combination which better captures the variability in the data
- Move to the second linear combination to try to capture the variability not explained by the first one
- Continue until the set of new variables explains most of the variability (frequently 90% is considered enough)

An illustration with the Iris data set

	Comp.1	Comp.2
Sepal.Length	0.361	−0.657
Sepal.Width	−0.085	−0.730
Petal.Length	0.857	0.173
Petal.Width	0.358	0.075

$$
\begin{aligned}
Comp.1 = \; & 0.361 \times Sepal.Length \\
& - 0.085 \times Sepal.Width \\
& + 0.857 \times Petal.Length \\
& + 0.358 \times Petal.Width
\end{aligned}
$$



The example in R

```
data(iris)
pca.data <- iris[,-5] # each case is described by the first 4 variables
pca <- princomp(pca.data)
loadings(pca)
```

```
##
## Loadings:
##              Comp.1 Comp.2 Comp.3 Comp.4
## Sepal.Length  0.361  0.657  0.582  0.315
## Sepal.Width          0.730 -0.598 -0.320
## Petal.Length  0.857 -0.173        -0.480
## Petal.Width   0.358        -0.546  0.754
##
##                Comp.1 Comp.2 Comp.3 Comp.4
## SS loadings      1.00   1.00   1.00   1.00
## Proportion Var   0.25   0.25   0.25   0.25
## Cumulative Var   0.25   0.50   0.75   1.00
```


The example in R

```
pca$scores[1:5,]
```

```
##          Comp.1     Comp.2      Comp.3       Comp.4
## [1,] -2.684126  0.3193972  0.02791483  0.002262437
## [2,] -2.714142 -0.1770012  0.21046427  0.099026550
## [3,] -2.888991 -0.1449494 -0.01790026  0.019968390
## [4,] -2.745343 -0.3182990 -0.03155937 -0.075575817
## [5,] -2.728717  0.3267545 -0.09007924 -0.061258593
```

```
scs <- pca$scores[,1:2]
dadosNovos <- data.frame(pca$scores[,1:2],
                         Species=iris$Species)
head(dadosNovos,3)
```

```
##      Comp.1     Comp.2 Species
## 1 -2.684126  0.3193972  setosa
## 2 -2.714142 -0.1770012  setosa
## 3 -2.888991 -0.1449494  setosa
```

```
plot(scs,col=as.numeric(iris$Species),
     pch=as.numeric(iris$Species))
legend('topright',levels(iris$Species),
       pch=1:3,col=1:3)
```