Preprocessing I

Aggregation and transformation

Tyrone Ryba

Last modified: 09 Sep 2021

Contents

1 Aggregation/data collection

This week we will focus on methods to collect datasets from a variety of sources and formats, read those into R, and begin transforming them into structures amenable to analysis.

1.1 Read from local file

Reading from a local file is straightforward in R. By default, files will be read from (and written to) the working directory, which is get and set using getwd() and setwd() respectively.

```r
# getwd()

project.dir <- "D:/Project"
dataset.dir <- "Datasets"
outputs.dir <- "Output"

# setwd(project.dir)
```

To avoid changing the working directory when reading in files from new locations, we can read or write files from file paths constructed from the file.path() function. This also helps to translate machine-specific path formats into a more compatible standard format.

```r
file.path(project.dir, dataset.dir)
```

```
## [1] "D:/Project/Datasets"
```

```
file.path(project.dir, outputs.dir)
```

```
## [1] "D:/Project/Output"
```

For reading in large tables using base R, there are some helpful tips in read.table()'s help file. See ?read.table.

In general, whenever possible, specify the type and size of the object in advance, then fill in the object slots later. For example, when importing text, it is much faster and more memory-efficient to specify the amount and types of data to be read first by including the nrows and colClasses arguments – this allows the right amount of memory to be reserved for the object and avoids inefficient methods for growing arrays. We might encapsulate that in a function for convenience:

```
  # gds <- read.delim("GDS5093_full.txt", skip=323)
  # gds <- read.delim(file.path(project.dir, dataset.dir, "GDS5093_full.txt"), skip=323)

  # Generic function for reading in tabular text datasets
  # with column names in the first row
  readDataset <- function(filename, rowNum, ...) {
    tab5rows  <- read.delim(filename, header = TRUE, nrows = 5, ...)
    classes   <- sapply(tab5rows, class)
    dataset   <- read.delim(filename, header = TRUE , nrows=rowNum,
                            comment.char = "", colClasses=classes, ...)
  }

  # yourDataset <- readDataset("Somefile.txt", rowNum=730e3)
  # gds.test <- readDataset("GDS5093_full.txt", rowNum=55000, skip=323)
```

Note: There is a special method for reading large files in the data.table package called fread() (for fast read). Although there are some side effects (the object is stored as both a data.frame and data.table by default), this is usually more convenient for any tab- or comma-delimited file larger than 100 MB.

```
  # install.packages('data.table')
  library(data.table)
  gds.dt <- fread("GDS5093_full.txt", skip = 323)
  str(gds.dt)
```

Performance of file reading methods

Try profiling the performance of reading data:
1. Using read.table() or read.delim() with default parameters
2. Using the readDatasets() function to specify column data types and row lengths
3. Using the fread() function from the data.table package.
Use system.time() or Rprof() to measure the time elapsed, as below.

```
# See:
# ?system.time()
# ?Rprof()
# ?summaryRprof()
```

We will talk more about optimization later, but see the lineprof package and Hadley Wickham's discussion of code profiling here: https://adv-r.had.co.nz/Profiling.html#measure-perf.

https://adv-r.hadley.nz/perf-measure.html http://adv-r.had.co.nz/Profiling.html#measure-perf

In some cases, R packages maintain databases that can be queried from local files. An example of this is the refGenome package, which provides objects for storing genome assembly and annotation data:

```
#install.packages("refGenome")
library(refGenome)

# From refGenome package
Ens_file  <- system.file("extdata", "hs.ensembl.62.small.RData", package="refGenome")
Ens_gen   <- loadGenome(Ens_file)
Ens_junc  <- getSpliceTable(Ens_gen)
Ens_gene  <- getGenePositions(Ens_junc)
Ens_gen
```

1.2 Read from remote files

R can also read from remote files. The most common way to do this is to use a URL in read.table. For this, we will download a record from The Cancer Genome Atlas (TCGA), which houses a comprehensive collection of genomic datasets froma variety of cancer types. The data are stored in two tiers, with one containing open access data freely available, and another controlled access data with potentially identifiable information.

Read the patients annotation table, nationwidechildrens.org_clinical_patient_hnsc.txt, into a variable called patn.df.

```
# First, try read.delim() on the following URL:
# http://people.csail.mit.edu/yueli/tcga/clin/nationwidechildrens.org_clinical_patient_hnsc.txt

patn.df <- read.delim("https://people.csail.mit.edu/yueli/tcga/clin/nationwidechildrens.org_clinical_pa
```

Did this work? We can check the properties of newly read datasets with a variety of methods, but as before str() and View() are useful for a first look:

```
str(patn.df)
View(patn.df)
```

1.3 Read from the clipboard

For quick plots or tables, R can read from and write to the clipboard:

```
cor(mtcars)

write.table(cor(mtcars), file="clipboard", sep="\t")
write.table(cor(mtcars), file="test.txt", sep="\t")
write.table(cor(mtcars), file="clipboard", row.names=FALSE, quote=FALSE, sep="\t")

read.delim(file="clipboard", header=TRUE)
write.table(Ens_gen@ev$genes, file="clipboard", row.names=FALSE, quote=FALSE, sep="\t")
# From here, data can be pasted into a spreadsheet, text editor, etc.

# More descriptive, though works with character strings:
# ?readClipboard()
# ?writeClipboard()
```

A common issue is that the clipboard will be just short of the memory needed. Try “clipboard-128” or “clipboard-256” if so. On Linux this generally requires a workaround with xclip or a similar package as described here: https://stackoverflow.com/questions/10959521.

```
# write.table(1:10, pipe("xclip -i", "w"))
```

For working with commands for transforming data today, we will use data from a dataset on patients with chronic kidney disease, available at the UCI Machine Learning Archive: http://archive.ics.uci.edu/ml/.

The files are in .arff format (Attribute-Relation File Format), which is from a machine learning package called Weka. To read these, we can use read.arff(), which is in the foreign package, now installed by default. Unfortunately, the data structure for this file is a bit misformed and requires debugging. We can work on a comma-separated version to focus on import and reformatting for today.

(More generally, see ls("package:foreign") or autocomplete "foreign::" for some other built-in functions to read from and write to other common statistical packages, like read.spss (SPSS), read.xport (SAS), etc.)

```
# library(foreign)
# ckd.arff <- read.arff(file.path(project.dir, dataset.dir,"chronic_kidney_disease.arff"))

ckd.df <- read.delim("chronic_kidney_disease.csv")
ckd.df <- read.csv("chronic_kidney_disease.csv")  # First version
str(ckd.df)
```

```
## 'data.frame':    400 obs. of  25 variables:
##  $ age  : Factor w/ 77 levels "?","11","12",..: 38 62 54 38 42 52 60 13 43 44 ...
##  $ bp   : Factor w/ 11 levels "?","100","110",..: 10 7 10 9 10 11 9 1 2 11 ...
##  $ sg   : Factor w/ 6 levels "?","1.005","1.010",..: 5 5 3 2 3 4 3 4 4 5 ...
##  $ al   : Factor w/ 7 levels "?","0","1","2",..: 3 6 4 6 4 5 2 4 5 4 ...
##  $ su   : Factor w/ 7 levels "?","0","1","2",..: 2 2 5 2 2 2 2 6 2 2 ...
##  $ rbc  : Factor w/ 3 levels "?","abnormal",..: 1 1 3 3 3 1 1 3 3 2 ...
##  $ pc   : Factor w/ 3 levels "?","abnormal",..: 3 3 3 2 3 1 3 2 2 2 ...
##  $ pcc  : Factor w/ 3 levels "?","notpresent",..: 2 2 2 3 2 2 2 2 3 3 ...
##  $ ba   : Factor w/ 3 levels "?","notpresent",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ bgr  : Factor w/ 147 levels "?","100","101",..: 23 1 116 19 8 123 2 114 37 122 ...
##  $ bu   : Factor w/ 119 levels "?","1.5","10",..: 65 34 83 86 53 52 84 59 89 6 ...
##  $ sc   : Factor w/ 85 levels "?","0.4","0.5",..: 10 6 16 55 12 9 48 9 17 76 ...
##  $ sod  : Factor w/ 35 levels "?","104","111",..: 1 1 1 3 1 27 2 1 1 5 ...
##  $ pot  : Factor w/ 41 levels "?","2.5","2.7",..: 1 1 1 2 1 7 16 1 1 12 ...
##  $ hemo : Factor w/ 116 levels "?","10","10.1",..: 56 15 113 14 18 24 26 26 10 112 ...
##  $ pcv  : Factor w/ 43 levels "?","14","15",..: 32 26 19 20 23 27 24 32 21 17 ...
##  $ wbcc : Factor w/ 90 levels "?","10200","10300",..: 71 55 69 61 67 71 1 63 87 17 ...
##  $ rbcc : Factor w/ 46 levels "?","2.1","2.3",..: 32 1 1 19 26 24 1 30 20 17 ...
##  $ htn  : Factor w/ 3 levels "?","no","yes": 3 2 2 3 2 3 2 2 3 3 ...
##  $ dm   : Factor w/ 3 levels "?","no","yes": 3 2 3 2 2 3 2 3 3 3 ...
##  $ cad  : Factor w/ 3 levels "?","no","yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ appet: Factor w/ 3 levels "?","good","poor": 2 2 3 3 2 2 2 2 2 3 ...
##  $ pe   : Factor w/ 3 levels "?","no","yes": 2 2 2 3 2 3 2 3 2 2 ...
##  $ ane  : Factor w/ 3 levels "?","no","yes": 2 2 3 3 2 2 2 2 3 3 ...
##  $ class: Factor w/ 2 levels "ckd","notckd": 1 1 1 1 1 1 1 1 1 1 ...
```

```
?read.csv
```

```
## starting httpd help server ... done
```

```
ckd.df <- read.csv("chronic_kidney_disease.csv")
str(ckd.df)
```

```
## 'data.frame':    400 obs. of  25 variables:
##  $ age  : Factor w/ 77 levels "?","11","12",..: 38 62 54 38 42 52 60 13 43 44 ...
##  $ bp   : Factor w/ 11 levels "?","100","110",..: 10 7 10 9 10 11 9 1 2 11 ...
##  $ sg   : Factor w/ 6 levels "?","1.005","1.010",..: 5 5 3 2 3 4 3 4 4 5 ...
##  $ al   : Factor w/ 7 levels "?","0","1","2",..: 3 6 4 6 4 5 2 4 5 4 ...
##  $ su   : Factor w/ 7 levels "?","0","1","2",..: 2 2 5 2 2 2 2 6 2 2 ...
##  $ rbc  : Factor w/ 3 levels "?","abnormal",..: 1 1 3 3 3 1 1 3 3 2 ...
##  $ pc   : Factor w/ 3 levels "?","abnormal",..: 3 3 3 2 3 1 3 2 2 2 ...
##  $ pcc  : Factor w/ 3 levels "?","notpresent",..: 2 2 2 3 2 2 2 2 3 3 ...
##  $ ba   : Factor w/ 3 levels "?","notpresent",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ bgr  : Factor w/ 147 levels "?","100","101",..: 23 1 116 19 8 123 2 114 37 122 ...
##  $ bu   : Factor w/ 119 levels "?","1.5","10",..: 65 34 83 86 53 52 84 59 89 6 ...
##  $ sc   : Factor w/ 85 levels "?","0.4","0.5",..: 10 6 16 55 12 9 48 9 17 76 ...
##  $ sod  : Factor w/ 35 levels "?","104","111",..: 1 1 1 3 1 27 2 1 1 5 ...
##  $ pot  : Factor w/ 41 levels "?","2.5","2.7",..: 1 1 1 2 1 7 16 1 1 12 ...
##  $ hemo : Factor w/ 116 levels "?","10","10.1",..: 56 15 113 14 18 24 26 26 10 112 ...
##  $ pcv  : Factor w/ 43 levels "?","14","15",..: 32 26 19 20 23 27 24 32 21 17 ...
##  $ wbcc : Factor w/ 90 levels "?","10200","10300",..: 71 55 69 61 67 71 1 63 87 17 ...
##  $ rbcc : Factor w/ 46 levels "?","2.1","2.3",..: 32 1 1 19 26 24 1 30 20 17 ...
##  $ htn  : Factor w/ 3 levels "?","no","yes": 3 2 2 3 2 3 2 2 3 3 ...
##  $ dm   : Factor w/ 3 levels "?","no","yes": 3 2 3 2 2 3 2 3 3 3 ...
##  $ cad  : Factor w/ 3 levels "?","no","yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ appet: Factor w/ 3 levels "?","good","poor": 2 2 3 3 2 2 2 2 2 3 ...
##  $ pe   : Factor w/ 3 levels "?","no","yes": 2 2 2 3 2 3 2 3 2 2 ...
##  $ ane  : Factor w/ 3 levels "?","no","yes": 2 2 3 3 2 2 2 2 3 3 ...
##  $ class: Factor w/ 2 levels "ckd","notckd": 1 1 1 1 1 1 1 1 1 1 ...
```

Did this work correctly? Check str(ckd.df). It seems the flag for missing values did not properly convert. How to fix this? See ?read.csv.

2 Transformation

2.1 Aggregation by factor level

There will be many times where we need to perform a calculation on each of the unique levels of a factor, and return the result. We will spend some time with a set of packages by Hadley Wickham (tidyr, plyr, dplyr, and reshape2) designed for this, but there are also useful methods in base R. One of these is aggregate().

```r
aggregate(Petal.Width ~ Species, iris, sum)
```

```
##      Species Petal.Width
## 1     setosa        12.3
## 2 versicolor        66.3
## 3  virginica       101.3
```

```r
aggregate(Petal.Length ~ Species, iris, sum)
```

```
##      Species Petal.Length
## 1     setosa         73.1
## 2 versicolor        213.0
## 3  virginica        277.6
```

```r
aggregate(wt ~ cyl + am, mtcars, median)
```

```
##   cyl am     wt
## 1   4  0 3.1500
## 2   6  0 3.4400
## 3   8  0 3.8100
## 4   4  1 2.0375
## 5   6  1 2.7700
## 6   8  1 3.3700
```

```
# What is the sample size for each factor combination?
# For this we can use table().
table(mtcars$cyl, mtcars$am)
```

```
##
##       0  1
##   4  3  8
##   6  4  3
##   8 12  2
```

```
summary(as.factor(mtcars$cyl))
```

```
##  4  6  8
## 11  7 14
```

Here, we collected the total and petal width and length for each level of species in the iris dataset. For mtcars, we found the number of entries with each unique combination of gear and cylinder number.

The next section covers an extension of this: creating new features or factor levels based on the distribution of values in one column.

Practice using aggregate: Collect the average miles per gallon (mpg) in the mtcars dataset as a function of number of cylinders (cyl), gears (gear) and transmission type (am).

A closely related and equally useful function for tabulating the numbers of values in individual or combined categories (factor variables) is table(). The output is made clearer by assigning labels to table rows and columns.

```
# Single-factor table
table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

```
# Two-factor table; label margins for clarity
table(Cylinders=mtcars$cyl, Gears=mtcars$gear)
```

```
##          Gears
## Cylinders  3  4  5
##         4  1  8  2
##         6  2  4  1
##         8 12  0  2
```

Practice using table: How does the distribution of normal versus abnormal red blood cell counts compare between patients with or without chronic kidney disease in the ckd.df dataset read above?

2.2 Constructing new features

The features most useful for analysis will often need to be constructed from some combination of other variables. For example, there are significant relationships between heart disease and diabetes progression that may make other values in the chronic kidney disease dataset easier to predict. In this case, we would like to construct a feature that records one of four states, depending on the combined states of diabetes mellitus (dm) and coronary artery disease (cad).

Create a new feature in the chronic diabetes dataset, called class_cad_dm, that creates an overall category from the combination of cad and dm variables.

```r
# Check first rows
head(ckd.df)

# Combine features - see ?paste / paste0
ckd.df$class_cad_dm <- paste(ckd.df$cad, ckd.df$dm, sep = "_")

# How many observations are in each combined category?
summary(as.factor(ckd.df$class_cad_dm))
```

2.3 Discretization

Generally, discretization is the process of creating a discrete or categorical variable from the values of a continuous one. These might be created from intervals of equal distances or equal numbers of values along the variable's range, or by specialized methods for classification problems. Discretization is often useful or necessary for applying certain algorithms in machine learning.

2.3.1 Interval-based

The simplest way to discretize is to create a series of bins with equally spaced breakpoints chosen in advance, and collect the number of values that fall into each bin.

Create a new categorical variable, wbcc_rng, that contains equally spaced bins along the range of white blood cell count (wbcc) values.

```r
# See ?cut / ?seq
# ?cut
```

2.3.2 Equal frequency / depth

Equal frequency bins have the benefit of balancing the amount of information contained among observations in each category. These can be created similarly, here using quantile() to find breakpoints.

Create a new categorical variable, wbcc_num, that contains an equal number of observations in each of ten age categories.

```r
# Create category; split wbcc by decile age groups
ckd.df$age_cat <- cut(ckd.df$age, breaks=quantile(ckd.df$age, probs=seq(from=0, to=1, by=0.10), na.rm=T
wbcc_num <- split(ckd.df$wbcc, ckd.df$age_cat)

# Check distribution
summary(wbcc_num)

# An intuitive way to see where these fall in the distribution
# is with plot(density()) and rug() or abline(v=...):

# What edit is needed?
plot(density(ckd.df$wbcc, na.rm=TRUE))
```

2.3.3 Entropy-based

Discretization can also be done in a supervised fashion, and in such a way that the least information is lost – for instance, by using bins that minimize the loss in relationship between variables in the dataset.

These methods can be applied through the discretization package.

Using discretize, find bins that minimize entropy in a column in the TCGA patients dataset or the chronic kidney disease dataset.

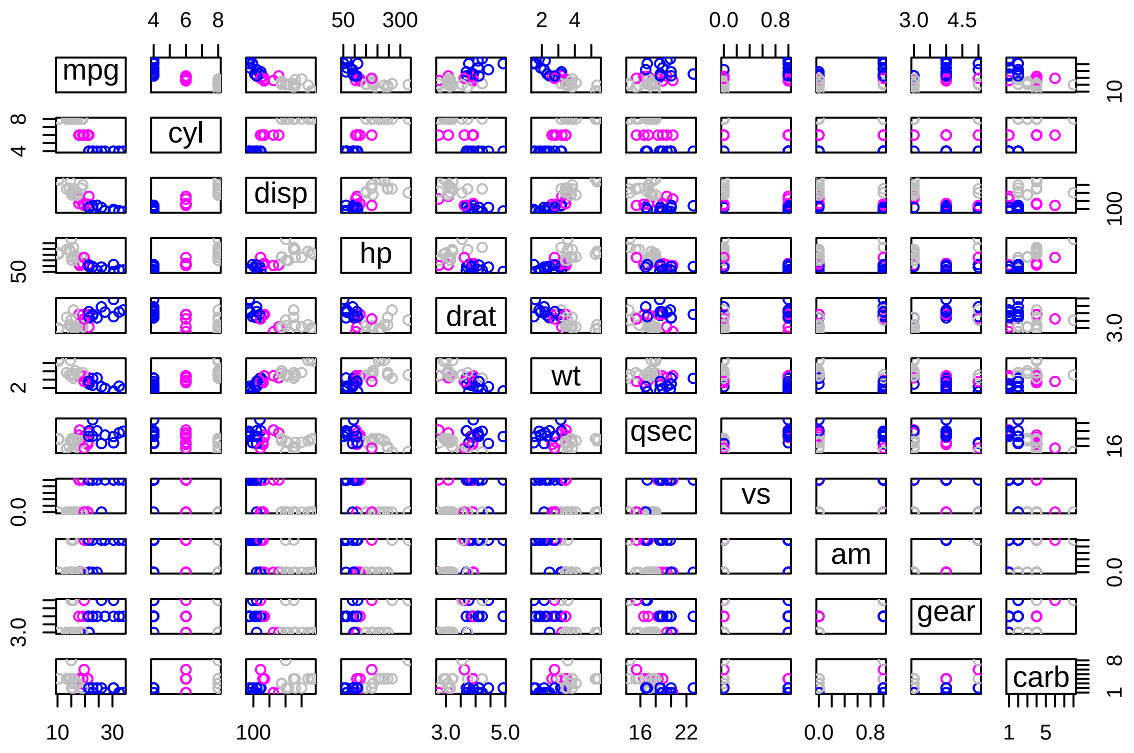
```
# install.packages("discretization")
library("discretization")

?discretization::disc.Topdown()

disc.Topdown(mtcars)
```

```
## $cutp
## $cutp[[1]]
## [1] 10.40 14.85 16.10 17.55 21.20 31.40 33.90
##
## $cutp[[2]]
## [1] 4 5 7 8
##
## $cutp[[3]]
## [1]  71.10 120.20 153.35 196.30 266.90 334.00 472.00
##
## $cutp[[4]]
## [1]  52.0  63.5 111.5 177.5 192.5 299.5 335.0
##
## $cutp[[5]]
## [1] 2.760 3.035 3.075 3.875 4.000 4.325 4.930
##
## $cutp[[6]]
## [1] 1.5130 1.7250 2.5425 2.6950 3.6500 4.6600 5.4240
##
## $cutp[[7]]
## [1] 14.500 16.580 17.350 18.410 18.605 21.560 22.900
##
## $cutp[[8]]
## [1] 0.0 0.5 1.0
##
## $cutp[[9]]
## [1] 0.0 0.5 1.0
##
## $cutp[[10]]
## [1] 3.0 3.5 4.5 5.0
##
##
## $Disc.data
##                       mpg cyl disp hp drat wt qsec vs am gear carb
## Mazda RX4               4   2    3  2    4  3    1  1  2    2    4
## Mazda RX4 Wag           4   2    3  2    4  4    2  1  2    2    4
## Datsun 710              5   1    1  2    3  2    5  2  2    2    1
## Hornet 4 Drive          5   2    4  2    3  4    5  2  1    1    1
## Hornet Sportabout       4   3    6  3    3  4    2  1  1    1    2
## Valiant              4   2    4  2    1  4    5  2  1    1    1
## Duster 360           1   3    6  5    3  4    1  1  1    1    4
## Merc 240D            5   1    2  1    3  4    5  2  1    2    2
## Merc 230             5   1    2  2    4  4    6  2  1    2    2
## Merc 280             4   2    3  3    4  4    3  2  1    2    4
## Merc 280C            4   2    3  3    4  4    5  2  1    2    4
## Merc 450SE           3   3    5  4    2  5    3  1  1    1    3
## Merc 450SL           3   3    5  4    2  5    3  1  1    1    3
## Merc 450SLC          2   3    5  4    2  5    3  1  1    1    3
## Cadillac Fleetwood   1   3    6  5    1  6    3  1  1    1    4
## Lincoln Continental  1   3    6  5    1  6    3  1  1    1    4
## Chrysler Imperial    1   3    6  5    3  6    3  1  1    1    4
## Fiat 128             6   1    1  2    5  2    5  2  2    2    1
## Honda Civic          5   1    1  1    6  1    4  2  2    2    2
## Toyota Corolla       6   1    1  2    5  2    5  2  2    2    1
## Toyota Corona        5   1    1  2    3  2    5  2  1    1    1
## Dodge Challenger     2   3    5  3    1  4    2  1  1    1    2
## AMC Javelin          2   3    5  3    3  4    2  1  1    1    2
## Camaro Z28           1   3    6  5    3  5    1  1  1    1    4
## Pontiac Firebird     4   3    6  3    3  5    2  1  1    1    2
## Fiat X1-9            5   1    1  2    5  2    5  2  2    2    1
## Porsche 914-2        5   1    2  2    6  2    2  1  2    3    2
## Lotus Europa         5   1    1  3    3  1    2  2  2    3    2
## Ford Pantera L       2   3    6  5    5  4    1  1  2    3    4
## Ferrari Dino         4   2    2  3    3  4    1  1  2    3    6
## Maserati Bora        2   3    5  6    3  4    1  1  2    3    8
## Volvo 142E           5   1    2  2    5  4    4  2  2    2    2
```

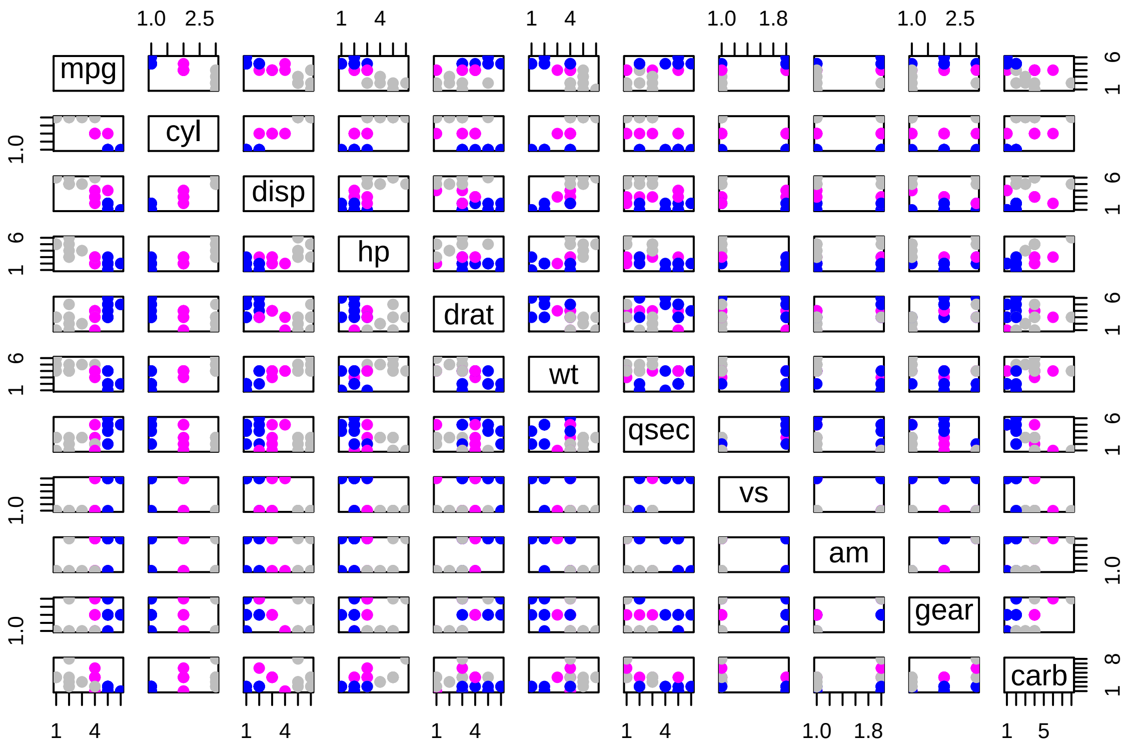
```r
str(disc.Topdown(mtcars))
```

```
## List of 2
##  $ cutp     :List of 10
##   ..$ : num [1:7] 10.4 14.8 16.1 17.6 21.2 ...
##   ..$ : num [1:4] 4 5 7 8
##   ..$ : num [1:7] 71.1 120.2 153.3 196.3 266.9 ...
##   ..$ : num [1:7] 52 63.5 111.5 177.5 192.5 ...
##   ..$ : num [1:7] 2.76 3.04 3.08 3.88 4 ...
##   ..$ : num [1:7] 1.51 1.73 2.54 2.7 3.65 ...
##   ..$ : num [1:7] 14.5 16.6 17.4 18.4 18.6 ...
##   ..$ : num [1:3] 0 0.5 1
##   ..$ : num [1:3] 0 0.5 1
##   ..$ : num [1:4] 3 3.5 4.5 5
##   ..- attr(*, "dim")= int 10
##  $ Disc.data:'data.frame':   32 obs. of  11 variables:
##   ..$ mpg : int [1:32] 4 4 5 5 4 4 1 5 5 4 ...
##   ..$ cyl : int [1:32] 2 2 1 2 3 2 3 1 1 2 ...
##   ..$ disp: int [1:32] 3 3 1 4 6 4 6 2 2 3 ...
##   ..$ hp  : int [1:32] 2 2 2 2 3 2 5 1 2 3 ...
##   ..$ drat: int [1:32] 4 4 3 3 3 1 3 3 4 4 ...
##   ..$ wt  : int [1:32] 3 4 2 4 4 4 4 4 4 4 ...
##   ..$ qsec: int [1:32] 1 2 5 5 2 5 1 5 6 3 ...
##   ..$ vs  : int [1:32] 1 1 2 2 1 2 1 2 2 2 ...
##   ..$ am  : int [1:32] 2 2 2 1 1 1 1 1 1 1 ...
##   ..$ gear: int [1:32] 2 2 2 1 1 1 1 2 2 2 ...
##   ..$ carb: num [1:32] 4 4 1 1 2 1 4 2 2 4 ...
```

```r
plot(mtcars, col=mtcars$cyl)
```



```r
plot(disc.Topdown(mtcars)$Disc.data, col=mtcars$cyl, pch=19)
```



2.3.4 Segmentation methods

For some projects, the categories we would like to use are not naturally encoded as intervals in the distribution, but can be extracted from abrupt changes in space or time. One example is the segmentation of copy number variants along the genome. Another would be segmentation of a time series of stock performance. There are now many methods to do this. For the former, packages like DNACopy, which performs the circular binary segmentation (CBS) algorithm of Olshen and Venkatraman, is a good choice. Methods can be found in the documentation in Bioconductor: http://www.bioconductor.org/packages/release/bioc/html/DNAcopy.html. For more general change point detection we will work with the changepoint package, which is simpler to apply. We will look at this next time.