

The Basics

The R CLI

The command line interface (CLI) is the basis of most functionality in R. This can be used to run scripts, or simple commands. For example, one can type

```
> 4+5
```

which will print on the screen

```
[1] 9
```

You can also type in

```
> if (length(grep("^X[\\d]", colnames(pca_data), perl=TRUE)) != 0)
{gsub("^X([\\d].*)", "\\1", colnames(pca_data), perl=TRUE)} else {colnames(pca_data)}
```

which will have some effect in context, however looks somewhat disturbing at first glance, but this will be covered later.

Using scripts

There are two main methods of running scripts in R. The first is to open the file containing the script in a text editor, and copy and paste the lines into the CLI, either line-by-line or as a block. The second is to use the command 'source' (provided the script is present in the working directory – see “Setting the working directory” below)

```
> source("name_of_script.r")
```

which will run the script. The script may contain a function, which is a block of code that has a particular purpose, which is then executed by using the command that is specified by the function (see below).

Functions

These are a blocks of code that execute many lines of code at once. It is possible to have a script that defines a number of functions that are then executed within the script. This can prevent re-typing the whole code block by using a short command instead. For example, if you needed to make a number of plots of the same type, you might have a function that generated the plot which takes an argument (that which is in the brackets) of which data matrix to use, e.g.

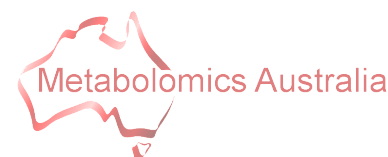
```
> plot_image(density)
> plot_image(mass)
```

rather than

```
> plot(density, main="Density vs Size", ylab="Size (km)", xlab="Density (kg/m³)", cex=3)
> plot(mass, main="Mass vs Gravity", ylab="Gravity (g)", xlab="Mass (kg)", cex=3)
```

etc.

R Tutorial



***R* basics**

Variable assignment

Assigning a result to an object is one of the basic steps in using R. To do this, you indicate that you wish to put a result into a variable by using the syntax `variable <- <R command>`, for example

```
data <- read.csv("input.csv", header=TRUE, rownames=1)
```

which reads the file "input.csv" and stores it as the variable `data`.

Setting the working directory

To see where you are, you can type

```
getwd()
```

which will show you the path to the directory that you are working in (where you will be able to access files). To change the directory, you can use the command (for a GNU/Linux machine)

```
setwd("/home/user/folder/with/data/or/scripts/")
```

or for a Windows machine:

```
setwd("C:\\Documents and Settings\\user\\My Documents\\folder\\with\\data\\or\\scripts")
```

(e.g. copy from Windows Explorer and double the backslashes), or

```
setwd("C:/Documents and Settings/user/My Documents/folder/with/data/or/scripts")
```

(e.g. replace the backslashes with forward slashes), or select **File > Change dir...** from the menu. All of these variants (on a Windows machine) perform the same function, so choose whichever method works best for you. After using **Change dir...** from the file menu, it is a good practice to use `getwd()` to verify your working directory is what you expect it to be.

Getting help

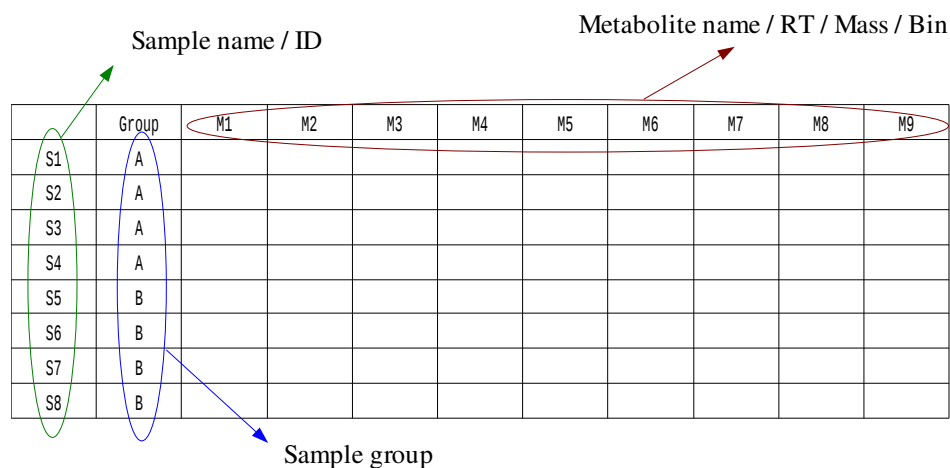
For all in-built functions in R, you can use `help("command")` or `?command` to show a help file. Under GNU/Linux, this usually comes up in the current terminal window, while in Windows it appears in a browser window.

MA scripts

The bioinformatics group at Metabolomics Australia have created a number of scripts for use with R. These provide basic statistical functionality in a manner that allows a standard data input format to be transformed to a standard output format, allowing much more simple data analysis methods.

Input data format

If there is a consistent input format, scripts are much simpler to execute and troubleshoot if they do not work. In brief, the matrix should have the first column containing sample names, the second the groups and the remaining columns the variables to be processed by the script (Figure 1). These can be metabolites, retention times, masses, bins, or anything else that has been measured for the samples.



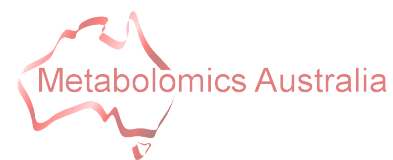
	Group	M1	M2	M3	M4	M5	M6	M7	M8	M9
S1	A									
S2	A									
S3	A									
S4	A									
S5	B									
S6	B									
S7	B									
S8	B									

Figure 1: The standard format of the data matrix for statistical analyses

Currently available scripts

All of the scripts that are in the ma-scripts repository (<http://code.google.com/p/ma-bioinformatics/>) use the input data format specified above, and will process that as necessary. These are still under development, but they will provide the necessary output as they currently stand. These scripts have comments in them to let you know what each step achieves.

R Tutorial



Examples provided below are created using the file `input.csv`.

boxplot.r

This script will produce a boxplot for both samples (Figure 2) and variables (Figure 3).

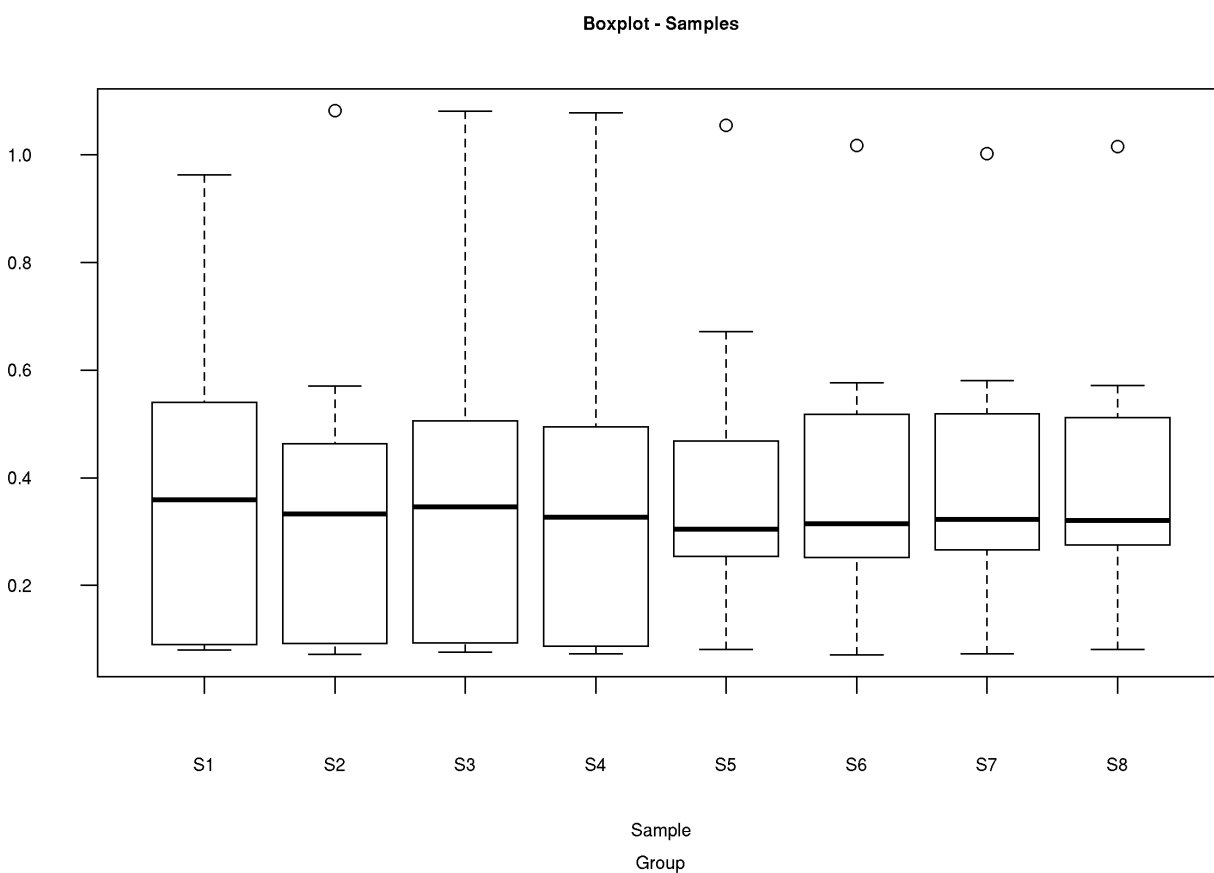


Figure 2: Boxplot for samples using data from `input.csv`

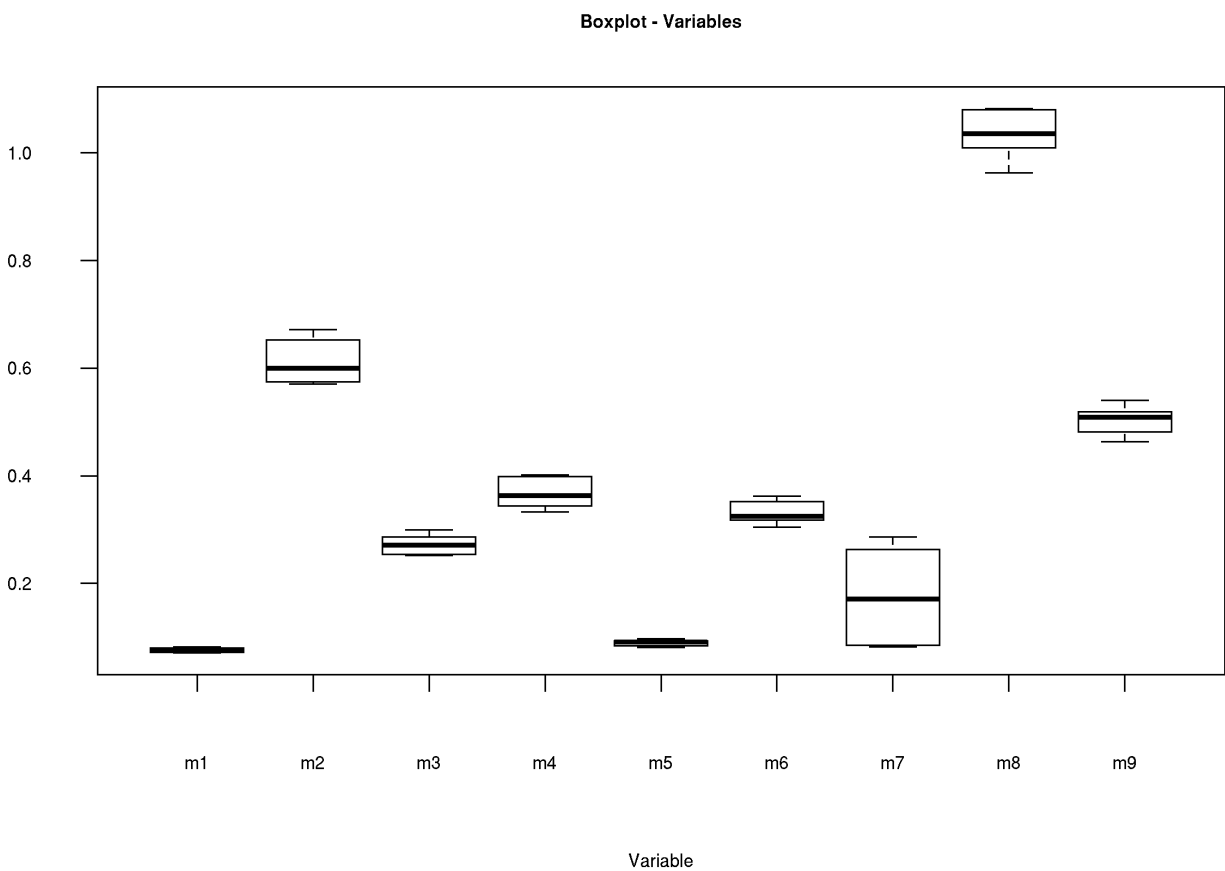


Figure 3: Boxplot of variables from input.csv

mms_plot.r

Produces a plot of the mean, median and standard deviation across all samples (Figure 4). Enables a graphical overview of similarity of samples, and allows for rapid identification of outliers.

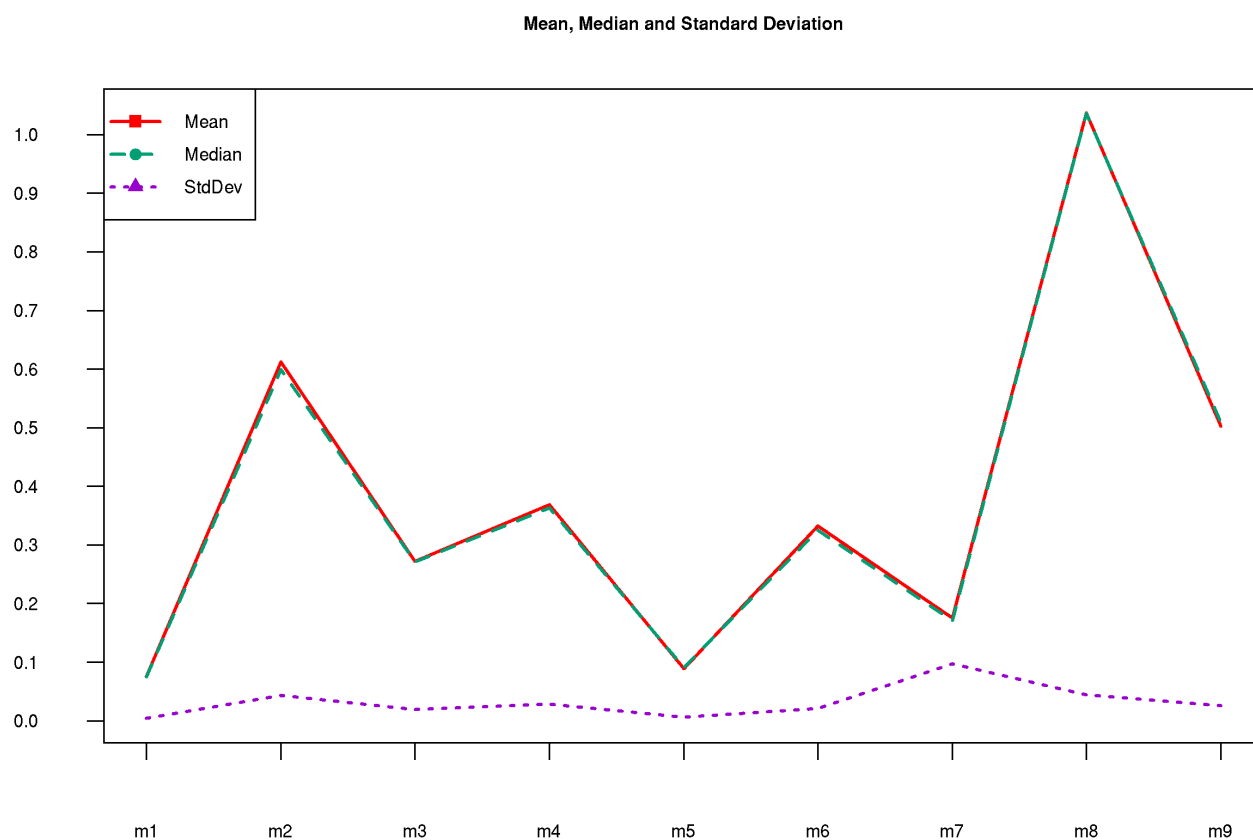


Figure 4: Line plot showing the mean, median and standard deviation .

mean_sd.r

It is important to explore a data set prior to choosing any normalisation or transformation method. Heteroscedastic noise (where the standard deviation of each metabolite in replicate samples changes with the mean of the metabolite) may change the results from the normalised data profoundly.

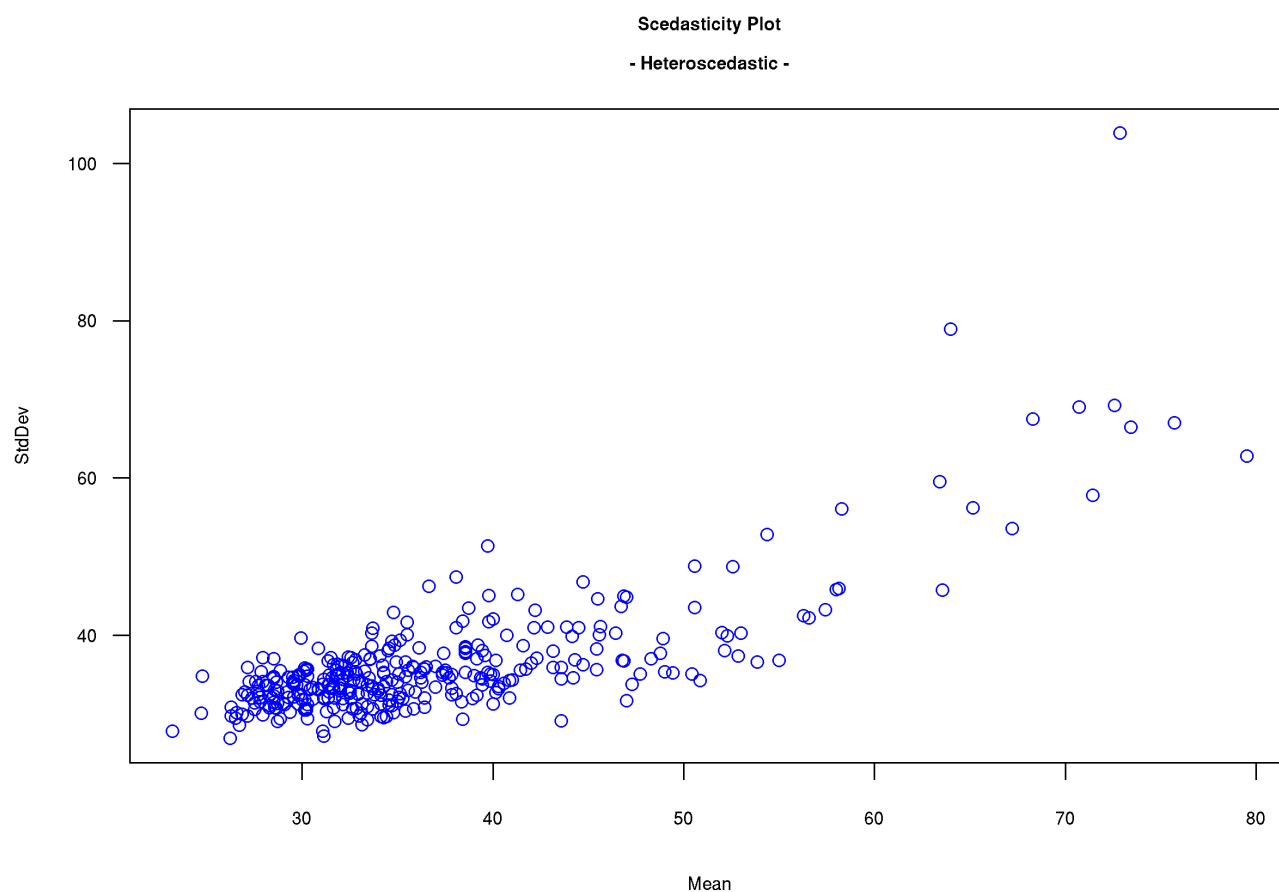


Figure 5: Heteroscedastic data. The overall shape of the data is a fanned line, as the standard deviation increases with the size of the mean.

The optimal transformation to change heteroscedastic noise into homoscedastic noise depends on the structure of the heteroscedasticity in the signals. If the standard deviation is proportional to the mean of the signal (as in Figure 5), then a log transform is optimal to treat the data. If the standard deviation is

proportional to the root of the mean then square root transformation provides a homoscedastic noise pattern (as in Figure 6).

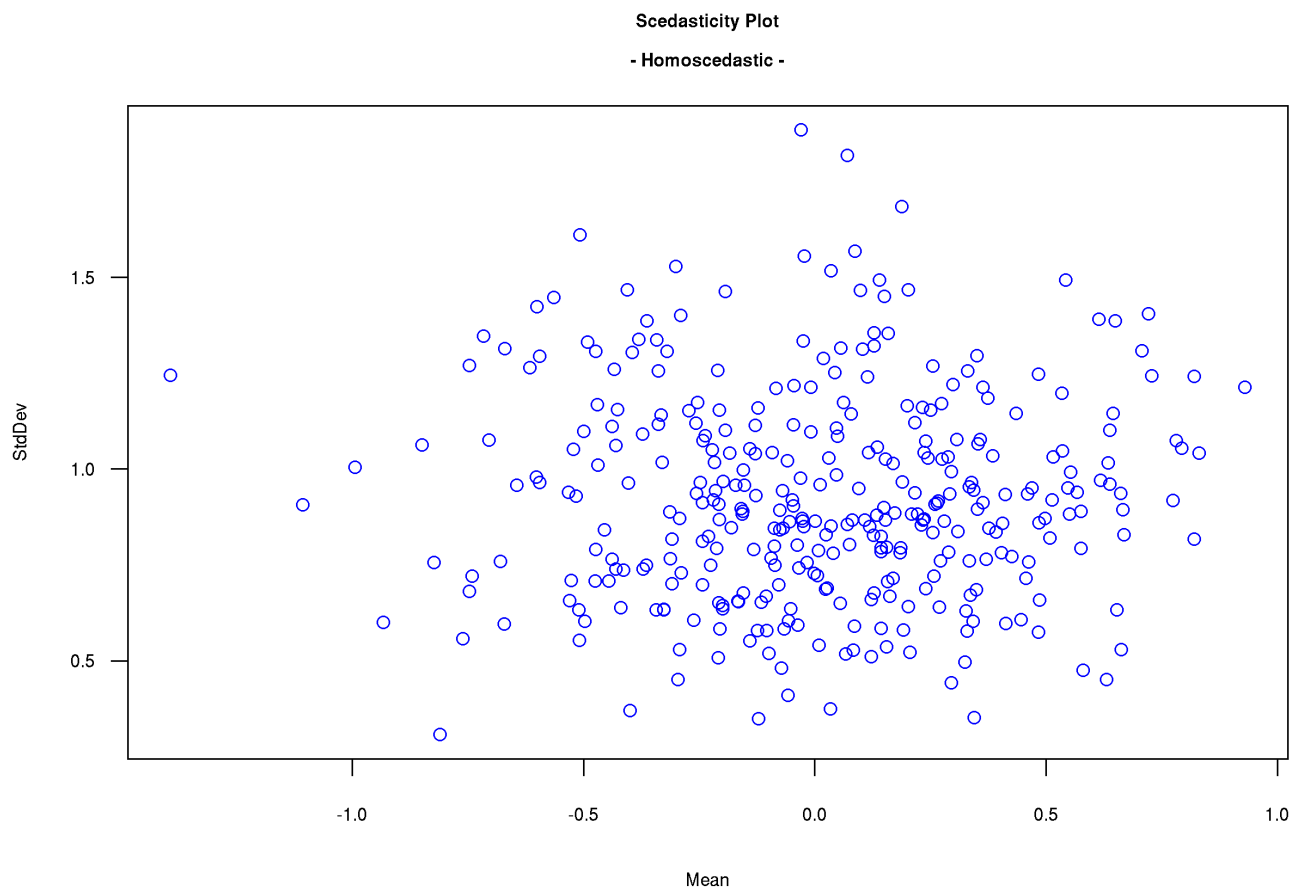


Figure 6: Homoscedastic data. Points are distributed randomly over the plot, and standard deviation does not vary with the size of the mean.

Transformation

This is an operation that is applied to each variable, and the type of transformation applied depends on the statistical test that is necessary to answer a biological question. For example, a t -test assumes that the data is normally distributed, hence it is important to transform a right- or left-skewed data set to a normal distribution.

log.r

Performs a \log_{10} transform of the input data, and generates a new data matrix [log_transform.csv].

Normalisation

This process is applied to the entire data set, and enables samples to be directly compared to one another by reducing the influence that is non-biological in origin. As an example, running samples on different days can cause samples to *appear* different when in actual fact they are the same. Normalisation attempt to removes this influence so that samples from different groups can directly be compared.

The output of these scripts are all .csv files.

norm_IS.r

Uses an internal standard (in the column immediately following the “Group” column) to normalise the data set [norm_data_IS.csv].

norm_median.r

Takes the median value for each sample (i.e. row) and uses this value to normalise the data set [norm_data_median.csv].

norm_Zscore.r

Z score is a measure of how many standard deviations a sample is from the mean. This script calculates the Z score for each sample and uses that value to normalise the data set [norm_data_zscore.csv].

norm_sum.r

This script normalises the values on a per-sample basis by taking the sum of all values for each sample and using that value as the normalisation factor [norm_data_sum.csv].

Cluster analysis

hca.r

Hierarchical cluster analysis (HCA) is a technique that identifies groups of samples that show similar characteristics, and then quantifies the structural characteristics of the sample (or variable) data by constructing a hierarchy in a tree-like structure. There are two kinds of procedures that are commonly used to construct the tree, namely agglomerative and divisive. In an agglomerative procedure, each sample (or variable) starts in a cluster of its own, and then continually joins clusters until there is only one cluster containing all samples. The divisive method operates in the exact reverse.

This script uses complete linkage method but other different linkage measures can be used in HCA, including average, single and Ward's, which may result in different clusters. The main objective of HCA is to classify the data into groups by structuring it, which helps to identify relationships among observations (samples/variables).

This script uses Manhattan distance to calculate the similarity, but other methods can be used by changing "manhattan" to one of "euclidean", "maximum", "canberra", "binary" or "minkowski".

It will generate a plot of both the sample distances (see Figure 7) as well as the distance for the variables (Figure 8).

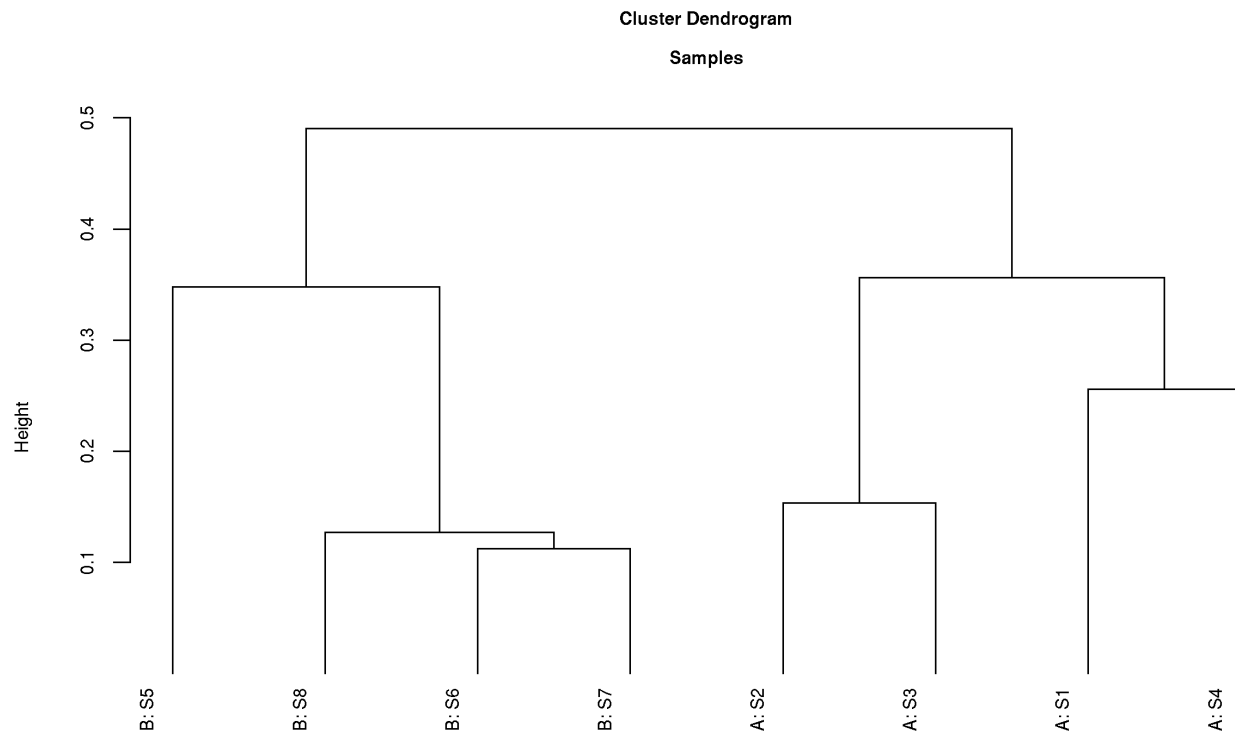


Figure 7: Hierarchical Cluster Analysis (HCA) for samples. Group labels are incorporated in the names on the leaves of the dendrogram.

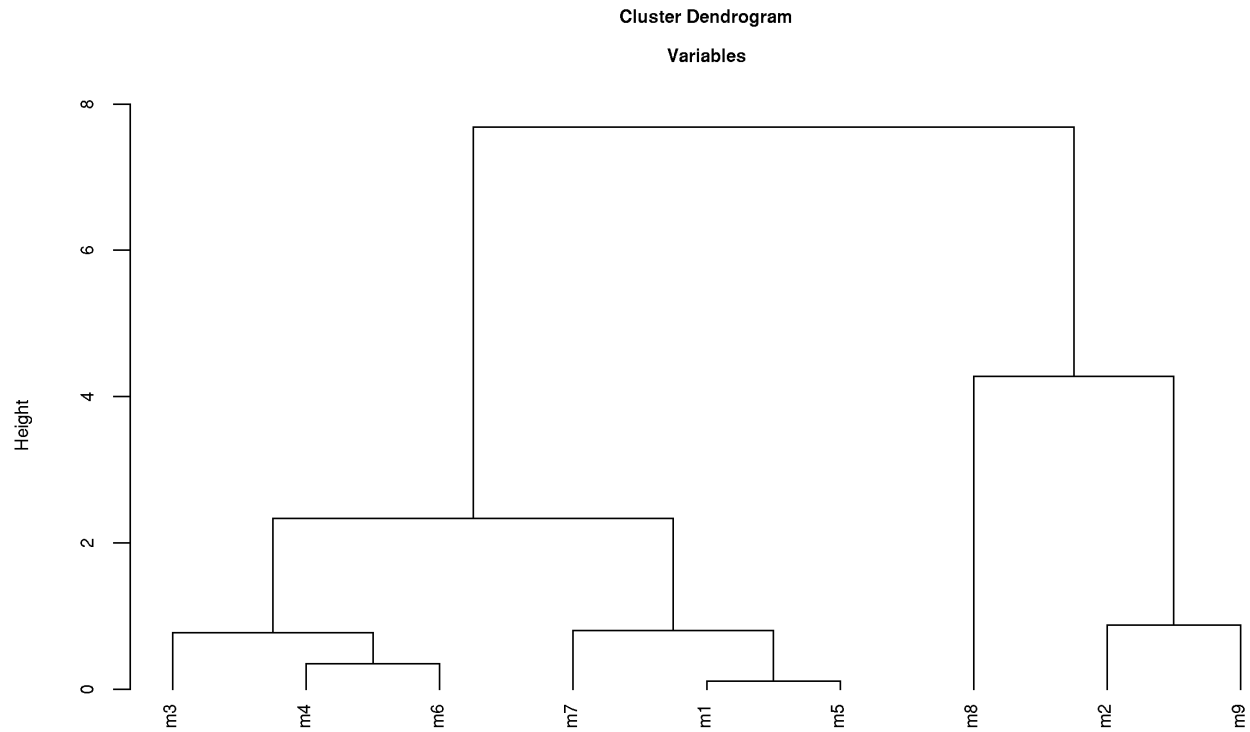


Figure 8: HCA for variables.

R Tutorial



pca.r

Principal Component Analysis (PCA) is a data transformation technique which is used to reduce a multidimensional data set to a lower number of dimensions for further analysis. In PCA, a data set of interrelated variables is transformed to a new set of variables called principal components (PCs) in such a way that they are uncorrelated, and the first few of these PCs retain most of the variation present in the entire data set. The first PC is a linear combination of all the actual variables in such a way that it has the greatest amount of variation, and the second PC is a combination of the variables that have the next greatest variation in the remaining PCs.

This script produces multiple plots – residual variance (Figure 9), scores plot labelled with sample names (Figure 10), scores plot labelled with group names (Figure 11) and a loading plot (Figure 12).

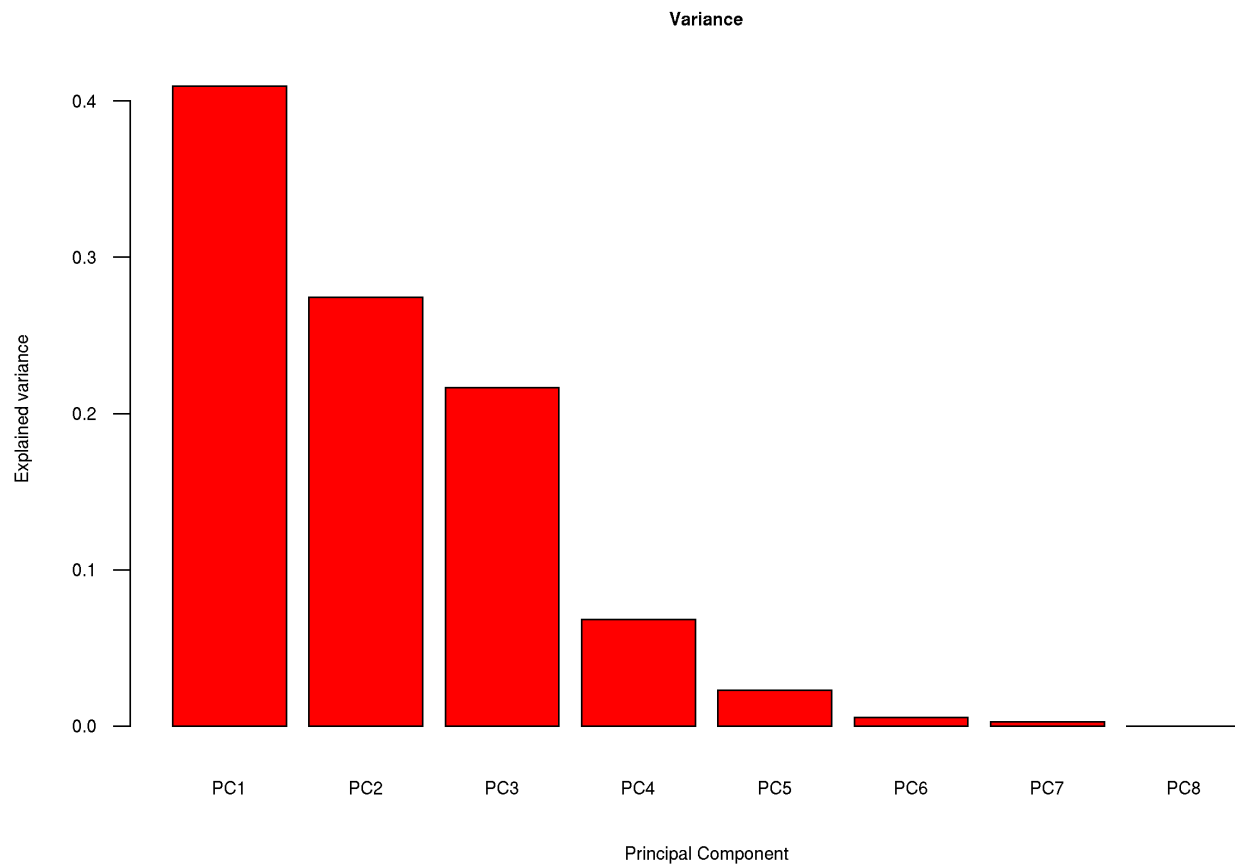


Figure 9: Residual variance plot.

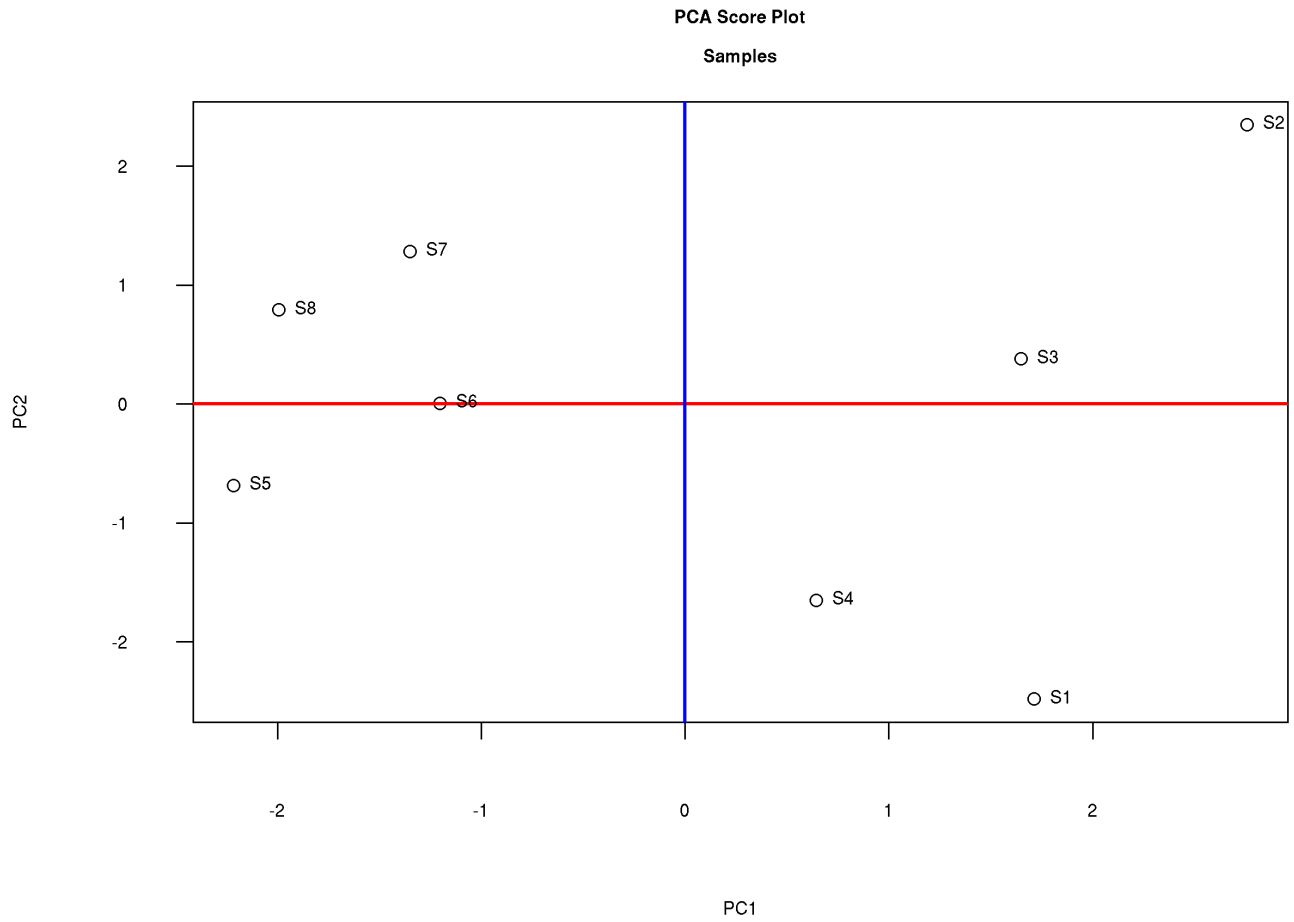


Figure 10: Scores plot with sample names as labels

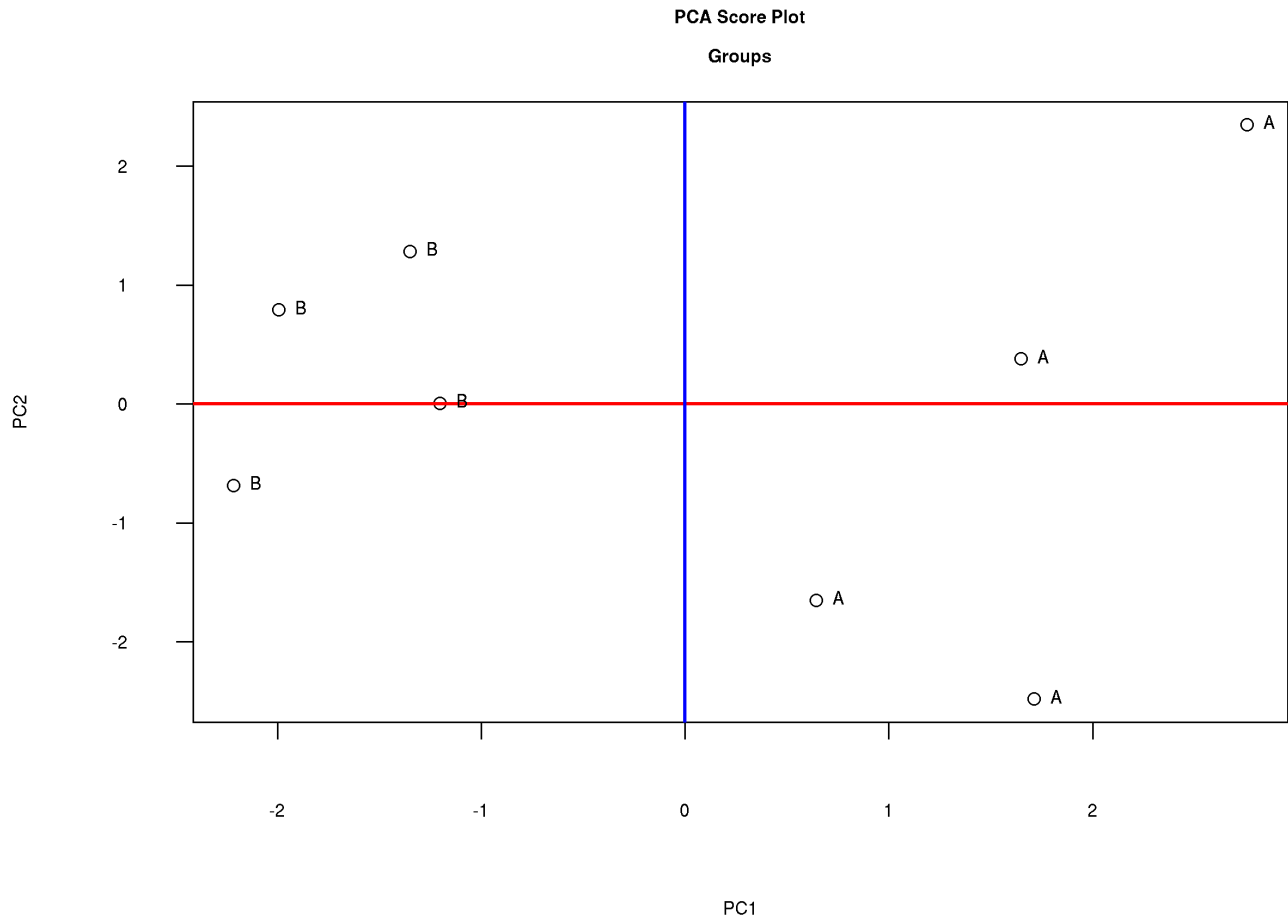


Figure 11: Scores plot with group names as labels

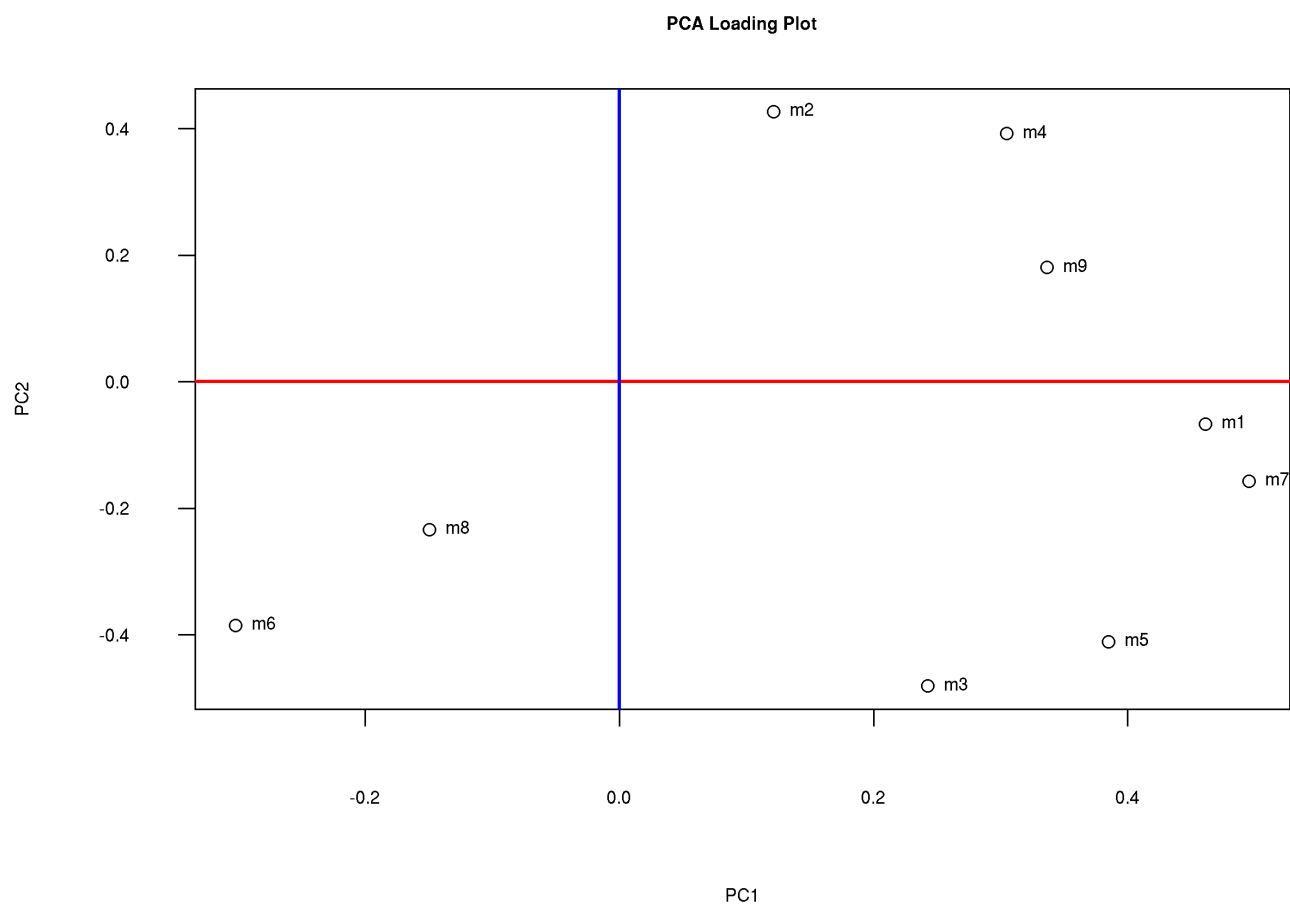


Figure 12: Loadings plot. Indicates which variables contributed to the separation seen in the scores plot.

lda.r

Linear Discriminant Analysis (LDA) is a classical technique to predict groups of samples. This is a supervised method that requires prior knowledge of the groups. LDA is therefore well suited for non-targeted metabolic profiling data which is almost always “grouped”. LDA is very similar to PCA, except that the technique maximises the ratio of between-class variance to the within-class variance in a set of data, and thus gives maximal separation between the classes, as shown in Figure 13.

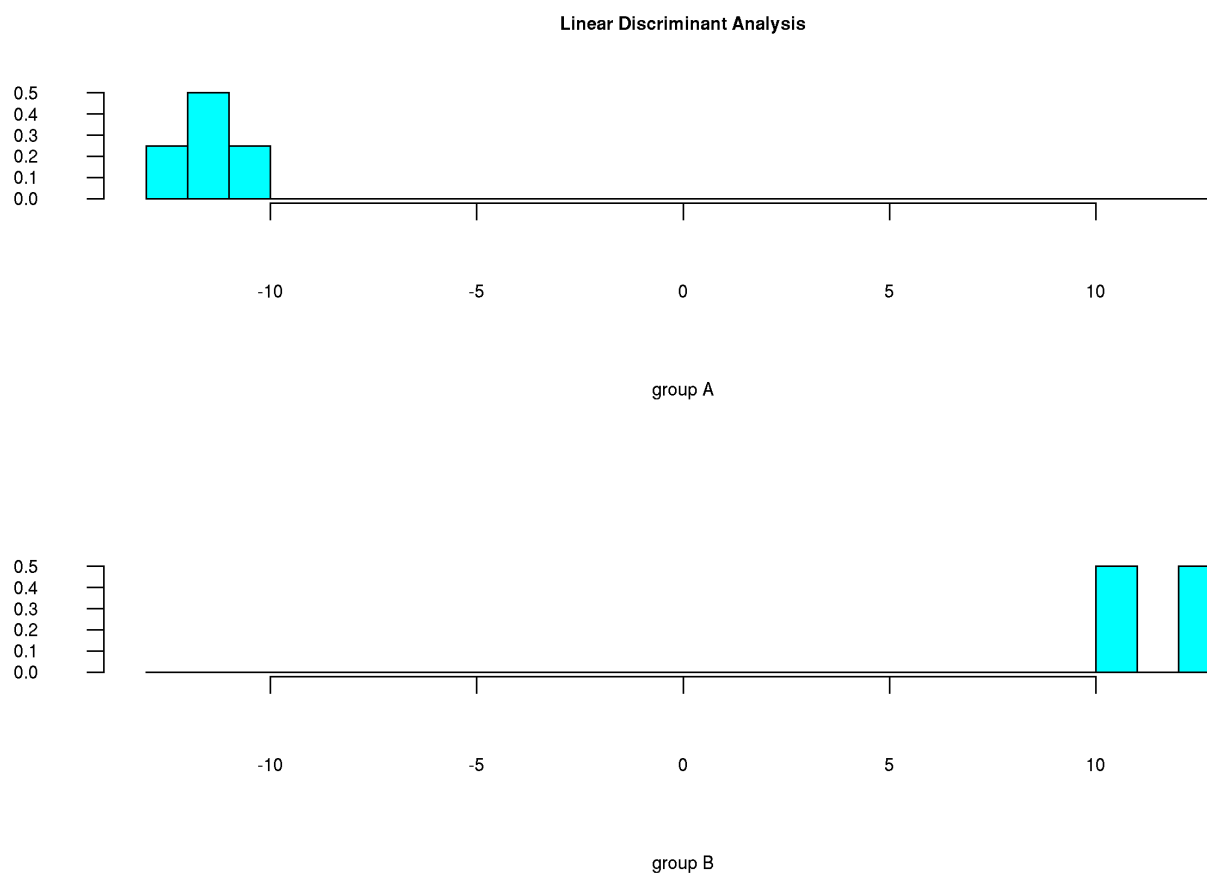


Figure 13: Linear Discriminant Analysis for two groups. This example shows a very clear separation between the groups.

t_test.r

Performs a Student's t -test, and outputs a .csv file with the P -values for each variable. This test assumes a normal distribution [p_t_test.csv].

wilcoxn.r

Performs Wilcoxon Rank Sum Test, a test that does not assume a normal distribution. Output is a .csv file containing the P -values for each variable [p_wilcoxon_test.csv].