Data Preprocessing and Unsupervised Learning in Bioinformatics

Elena Sugis

18 July 2016

Learning Objectives

- apply data normalization
- handle missing data
- filter outliers
- apply clustering techniques
- characterise the results

Before we start

Lots of specific R packages for the analysis of the biological data can be obtained from resource called Biocinductor https://www.bioconductor.org/ Bioconductor provides tools for the analysis and comprehension of biological data. Bioconductor uses the R statistical programming language, and is open source and open development.

Get the latest version of Bioconductor by starting R and entering the commands:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

We will use biocLite() to download the specific packages from bioconductor. Additional help on installing the packages via Bioconductor you can find here http://www.bioconductor.org/install/

Getting the data

Let's first of all download our data. Go to the link and download the csv file.

```
download.file("https://s3-eu-west-1.amazonaws.com/pfigshare-u-files/5515439/RawDataPractice.csv",
              "data/RawDataPractice.csv")
```

Check in which folder you are now and set the working directory using function `getwd()`. Set your working directory.

```
# Read the data into object data_raw
setwd("MyWorkingDirectory")
```

After we have downloaded the data, let's read it into the object data_raw.

```r
# Read the data into object data_raw
data_raw <- read.csv(file = "data/RawDataPractice.csv", sep =",")
```

Meet your data

In this lesson we will use the data of gene expression measured in skin biopcy of healthy and sick people. The expression of 42 genes is measured in 24 healthy individuals and 35 patients. The biopsy samples from patients were taken from lesional and non-lesiaonal skin regions. These samples are devided into two groups PG1, PG2. Few samples were ommited from the study due to their bad quality. The final dataset is a maxtrix of size 42x88.

Functions `head()` and `str()` and `summary()` can be useful to check the content and the structure of an R object.

- Summary:
- `str()` - structure of the object and information about the class, length and content of each column
- `summary()` - summary statistics for each column

Additionally you can check the following oprions:

- Size:
  - `dim()` - returns a vector with the number of rows in the first element, and the number of columns as the second element (the ___dim___ensions of the object)
  - `nrow()` - returns the number of rows
  - `ncol()` - returns the number of columns
- Content:
  - `head()` - shows the first 6 rows
  - `tail()` - shows the last 6 rows
- Names:
  - `names()` - returns the column names (synonym of `colnames()` for `data.frame` objects)
  - `rownames()` - returns the row names

Note: most of these functions are "generic", they can be used on other types of objects.

So let's see what is inside our data.

```r
head(data_raw)
str(data_raw)
summary(data_raw)
```

You can see that the first column of the data_raw contains the names of the genes. Lets call the rownames of data_raw by the gene name present in the first column in the corresponding row.

```r
# Add row names
rownames(data_raw) <- data_raw$X
rownames(data_raw)
```

```
##  [1] "AIM2"    "CASP1"   "CCL2"    "CCL20"   "CCL27"   "CCL5"    "CTLA4"
##  [8] "CXCL1"   "CXCL10"  "CXCL2"   "DEFB1"   "EOMES"   "FOXP3"   "IFIH1"
## [15] "IFNAR1"  "INFG"    "IFNGR"   "IL10"    "IL17A"   "IL17F"   "IL1b"
## [22] "IL1F6"   "IL1RN"   "IL20RA"  "IL22"    "IL22RA1" "IL22RA2" "IL26"
## [29] "CXCL8"   "KLRK1"   "LCN2"    "MICB"    "NLRP1"   "NLRP3"   "PI3"
## [36] "PYCARD"  "REG3A"   "S100A8"  "S100A9"  "TNF"     "TRGC1"   "WIPI1"
```

```
# Remove the first column
data_raw <- data_raw[,-c(1)]
```

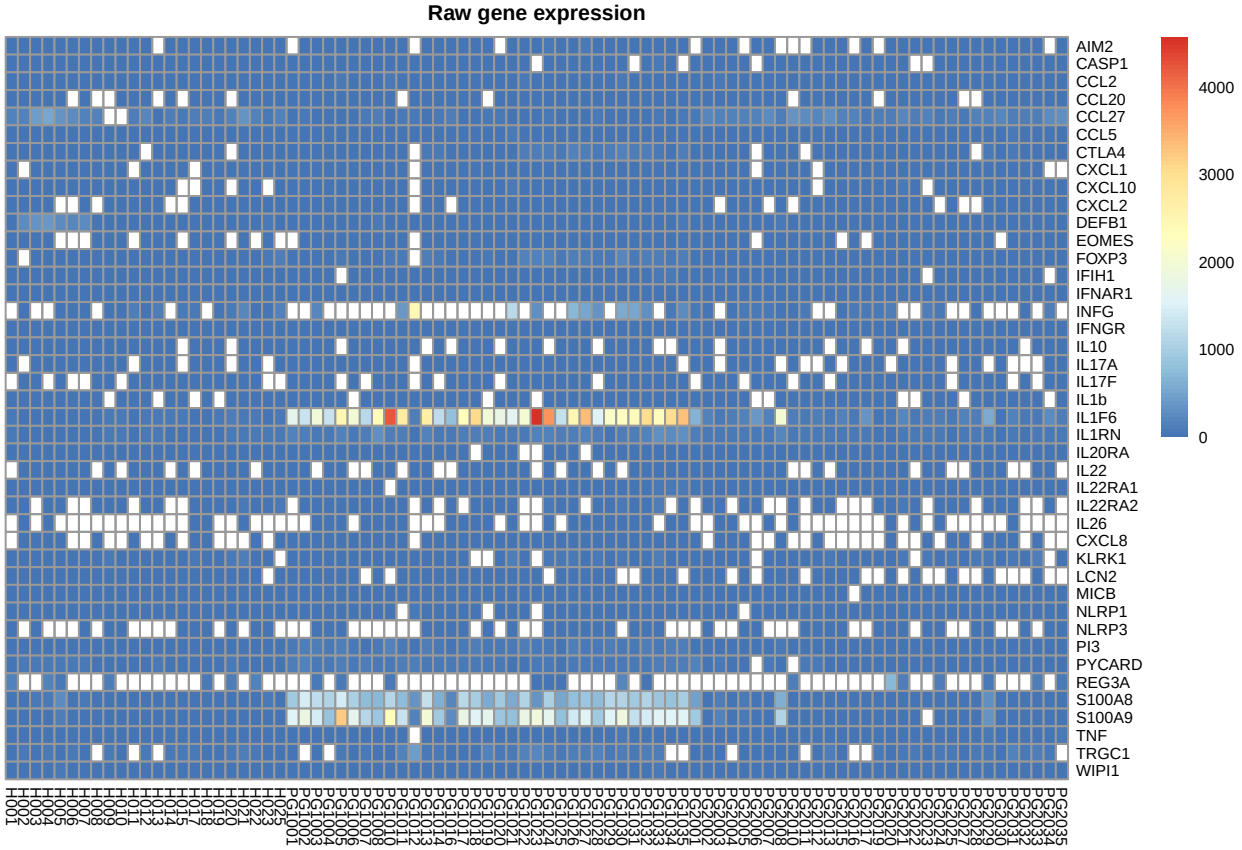
Plot raw data

One good way of getting the first impression of how your data looks like is to visualize it. Heatmaps are great to get a general idea of the dataset that you are working with. This type of visualization is well implemented in the package **pheatmap**. Install and attach this package by typing in the following commands:

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

Now you can use the function **pheatmap()**. Use **?pheatmap()** to get more information about the arguments. Inspect the argument options that this function has. Now we are ready to create our first hetmap.

```
pheatmap(data_raw, cluster_rows = F, cluster_cols = F,scale = "none", fontsize = 6,
         main = "Raw gene expression")
```



Challenge

Using the help option change the fontsize on the figure.

You can also see the distributions of the data For that we can modify our dataframe a bit using functions from the package `reshape`. Install and attach the package as following:

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

Next we will "melt" an object into a form suitable for easy plotting. Let's create a function that adds the type(group) of our samples (H, PG1, PG2) to the data and melts it.

```
meltmydata<-function(df){
  # Transpose
  dt <- t(df)

  # Convert to data.frame
  dt <- as.data.frame(dt)

  # Bind the column with sample names
  dt <- cbind(dt, sample = rownames(dt))

  # Change the type of the column
  dt$sample <- as.character(dt$sample)

  # Remove sample number
```

```
  dt$sample <- gsub("0.*", "", dt$sample)

  # Melt the data
  data_melt <- melt(dt)

  # Change the names of the columns
  colnames(data_melt) <- c("Group", "Gene", "Expression")
  return(data_melt)
}

data_raw_melt <- meltmydata(data_raw)
```

```
## Using sample as id variables
```

```
head(data_raw_melt)
```

```
##   Group Gene Expression
## 1     H AIM2  6.8092704
## 2     H AIM2  0.9582673
## 3     H AIM2  1.2405671
## 4     H AIM2  1.0702885
## 5     H AIM2  2.2485582
## 6     H AIM2  3.3349522
```

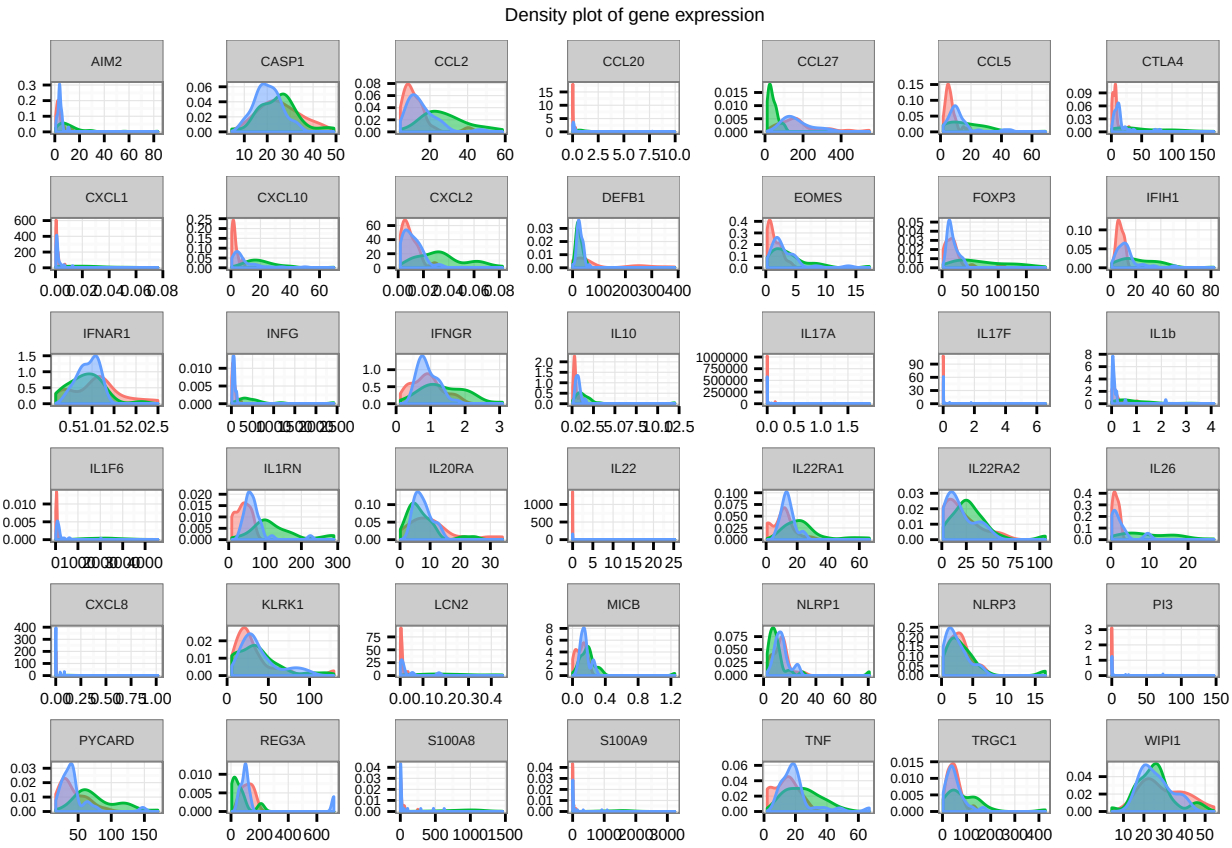
Currently we can notice that there are a lot of missing values in the data. However, since the range of the expression values varies a lot, we can't make any conclusions about the dataset just by looking at the heatmap.

To see how the expression values of each gene is distributed in each group of patients and healthy individuals, we will use R package `ggplot2` and in particular function `geom_density()`.

```
# Attach library ggplot2
library(ggplot2)

# Plot the distributions
ggplot(data_raw_melt, aes(x = Expression))+
  facet_wrap(~ Gene, scales = "free" )+
  theme_bw()+
  geom_density(aes(group = Group, colour = Group, fill = Group), alpha = 0.5)+
  ggtitle("Density plot of gene expression")+
  theme(text = element_text(size = 6),
        axis.text.x = element_text(size = 6),
        legend.text = element_text(size = 6), legend.position = "none",
        axis.title.x = element_blank(), axis.title.y = element_blank()
        )
```

```
## Warning: Removed 484 rows containing non-finite values (stat_density).
```



Impute missing values

Missing values in the datata can appear due to the different technical errors, human errors etc. There are essentially tree ways of handeling missing values. * You can remove them from the data * Substitute with 0, mean, median of the values in the raw/column. * Impute In this tutorial we will use k-nearest neighbour algorithm to impute the missing values in our data. For this purpose download and attach "impute" package from Bioconductor http://www.bioconductor.org/packages/release/bioc/html/impute.html

```
# Download "impute" package
# source("http://bioconductor.org/biocLite.R")
# biocLite("impute")

# Attach package
library("impute")
```

Function `impute.knn()` doesn't take dataframe as an argumet. Let's change it to matrix. To convert out dataframe from matrix we will use function `as.matrix()`

Impute the data with using 4 KNN and visualize the results using heatmap.

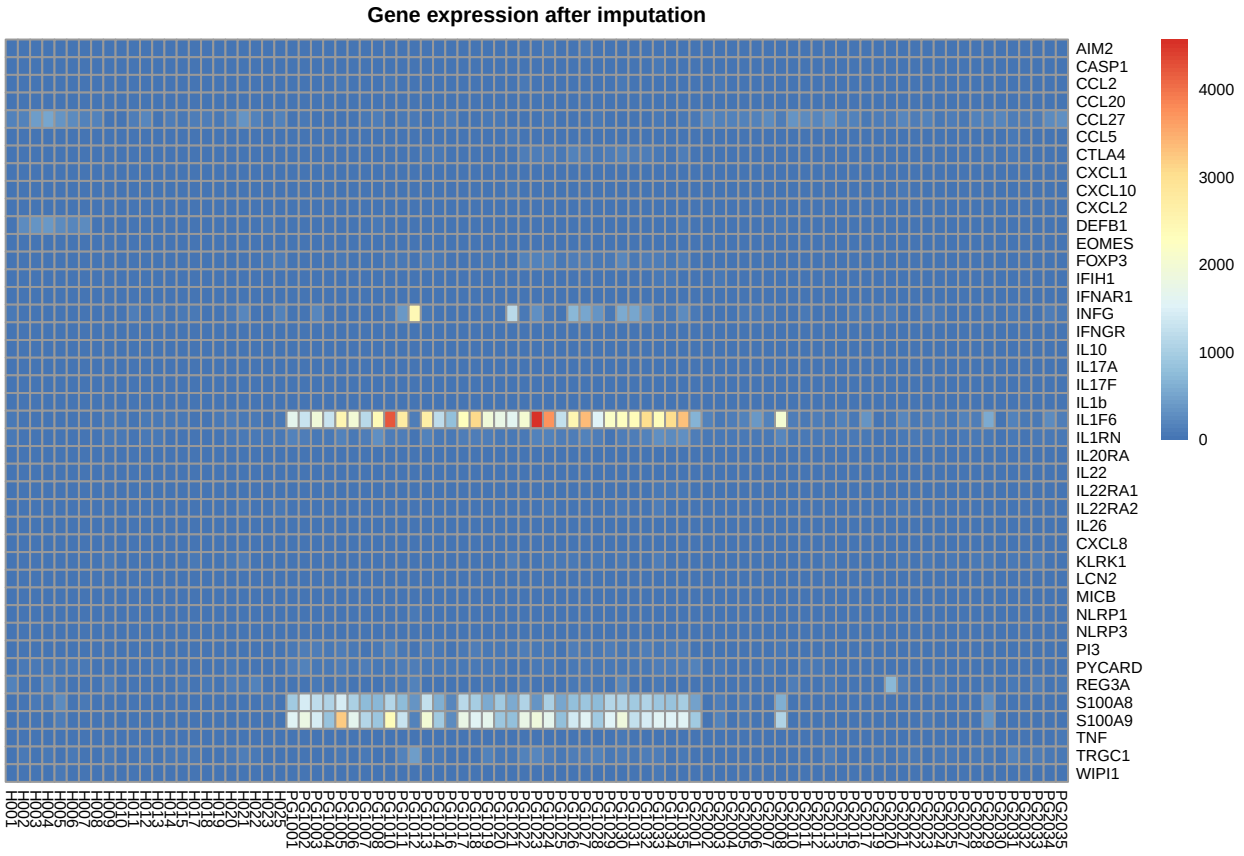
```
# Impute with 4 KNN
data_raw <- as.matrix(data_raw)
data_imp <- impute.knn(data_raw, k = 4, rowmax = 0.8, colmax = 0.8, maxp = 1500, rng.seed = 362436069)

# See the structure of data_imp
str(data_imp)
```

```
## List of 3
##  $ data     : num [1:42, 1:87] 6.8093 26.9585 16.0593 0.0413 151.4817 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:42] "AIM2" "CASP1" "CCL2" "CCL20" ...
##   .. ..$ : chr [1:87] "H001" "H002" "H003" "H004" ...
##  $ rng.seed : num 3.62e+08
##  $ rng.state: int [1:626] 403 624 -883456235 -838320942 -1535025621 -1416915088 216593585 531230398 -
```

```r
# Imputed values are located in the part called data_imp$data

# Plot the imputed data
pheatmap(data_imp$data, cluster_rows=F, cluster_cols=F,
         border_color = "grey60", fontsize = 6,
         main="Gene expression after imputation")
```



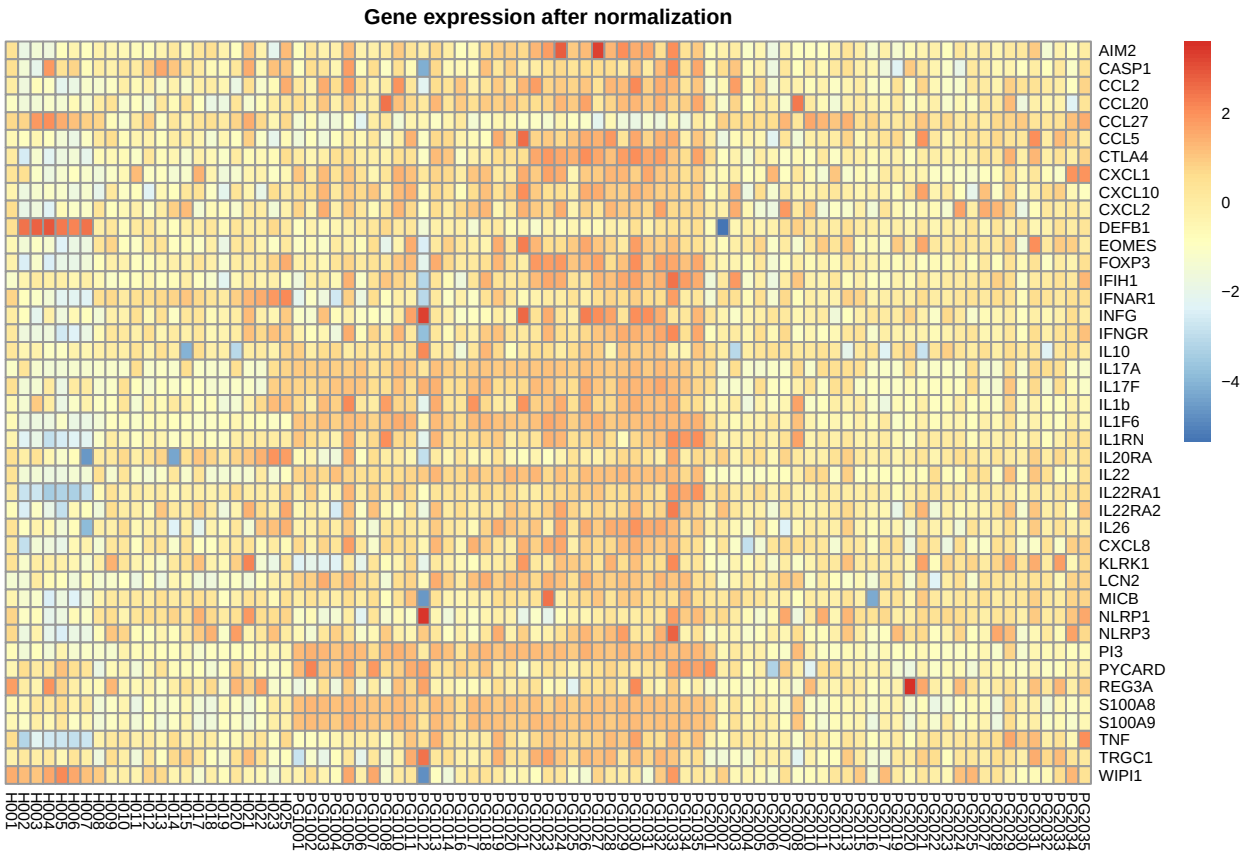
Normalize and scale your data

In order to remove technical variance in the data and make the samples more comparable we will normalize and standardise it. The methods of normalization is highly specific to the experimental origin of the data and it's properties. In this tutorial we will logarithmize, scale and center the data based on the mean and standard deviation of the expression of each individual gene(row).

```r
# Logarithmize gene expression
data_log <- log2(data_imp$data)
```

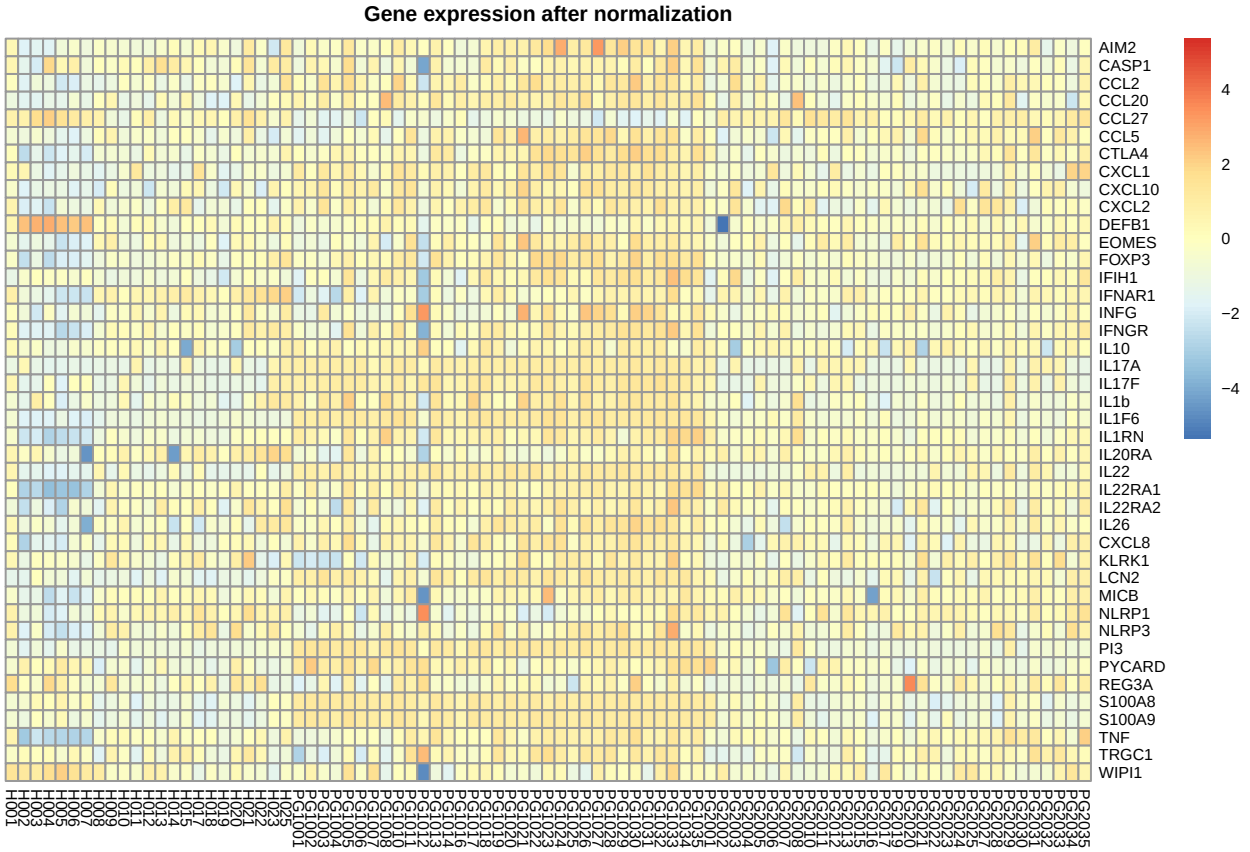
```
# Center and scale data by row
scale_rows <- function(x){
  m = apply(x, 1, mean, na.rm = T)
  s = apply(x, 1, sd, na.rm = T)
  return((x - m) / s)
}
data_norm <- scale_rows(data_log)

# Plot scaled data
pheatmap(data_norm , cluster_rows = F, cluster_cols = F,
         border_color = "grey60", fontsize = 6,
         main="Gene expression after normalization")
```



Alternatively you could just use pheatmap option center = row to get the figure of the centered data

```
# Plot scaled data
pheatmap(log2(data_imp$data) , cluster_rows=F, cluster_cols=F, scale = "row",
         border_color = "grey60", fontsize = 6,
         main="Gene expression after normalization")
```



Save preprocessed data

Save your processed data to a file using special R compressed data file format .RDS or .csv file to access it later.

```
saveRDS(data_processed,file="~/results/RawData.rds")
write.table(data_processed, file="~/results/data_processed.csv",sep=",",row.names=T, quote=F)
```

Handle outliers

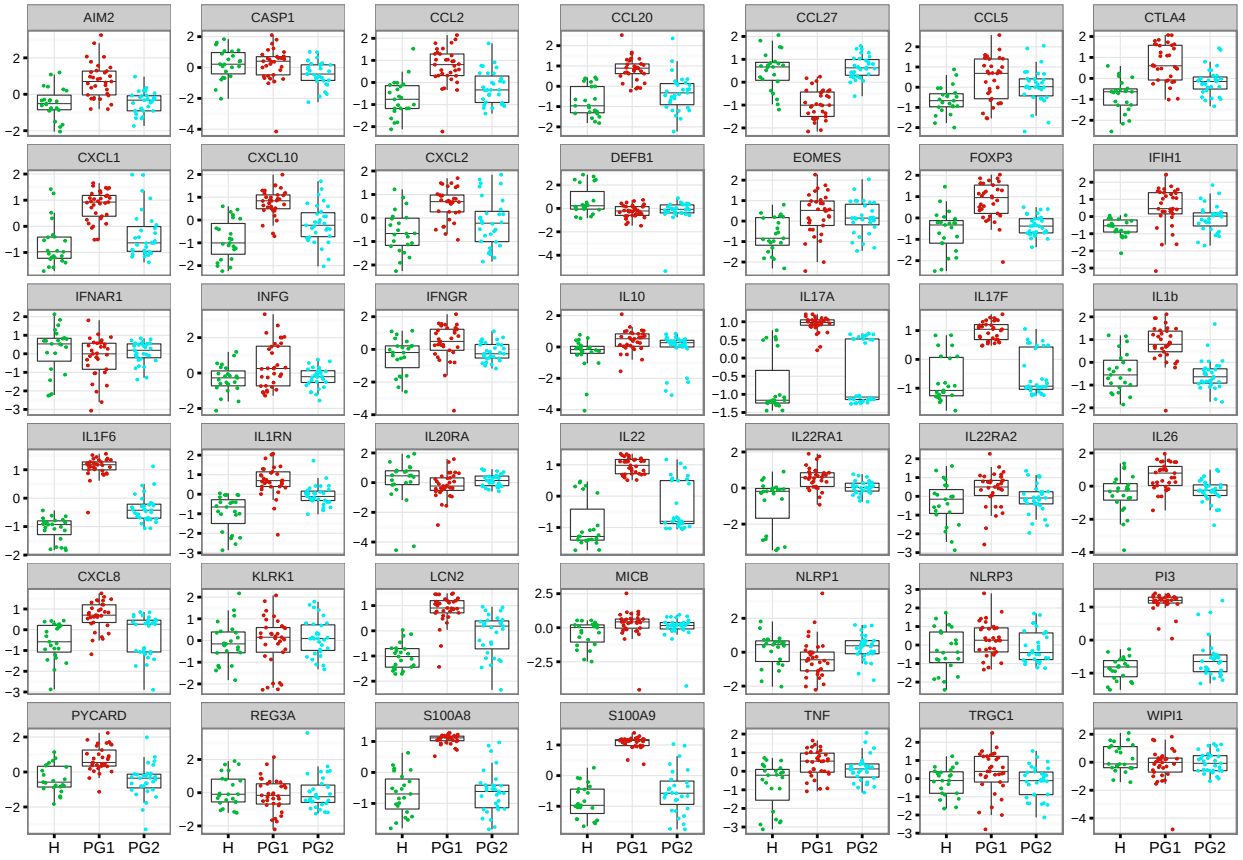
You can detect the outliers in the distribution of your data by looking at the boxplots. In case you decided to filter out outliers interquartile rate can be applied as a filtering criterion.

```
data_norm_melt <- meltmydata(data_norm)
```

```
## Using sample as id variables
```

```
ggplot(data_norm_melt, aes(x = Group, y = Expression))+
  # Plot for each gene in the separate facet
  facet_wrap(~ Gene, scales = "free_y")+
  # Add white background
  theme_bw()+
  theme_bw(base_size = 6)+
```

```
  geom_boxplot(outlier.shape=NA, fatten = 0.2, lwd=0.2)+
  geom_jitter(width = .7, size = 0.01,aes(colour = Group))+
  # Manually assign colors to the groups
  scale_color_manual(values = c("#00ba38", "#d4170a", "#00ebf2"))+
  theme(text = element_text(size = 6),
        axis.text.x = element_text(size = 6),
        legend.text = element_text(size = 6), legend.position = "none",
        axis.title.x = element_blank(), axis.title.y = element_blank()
        )
```



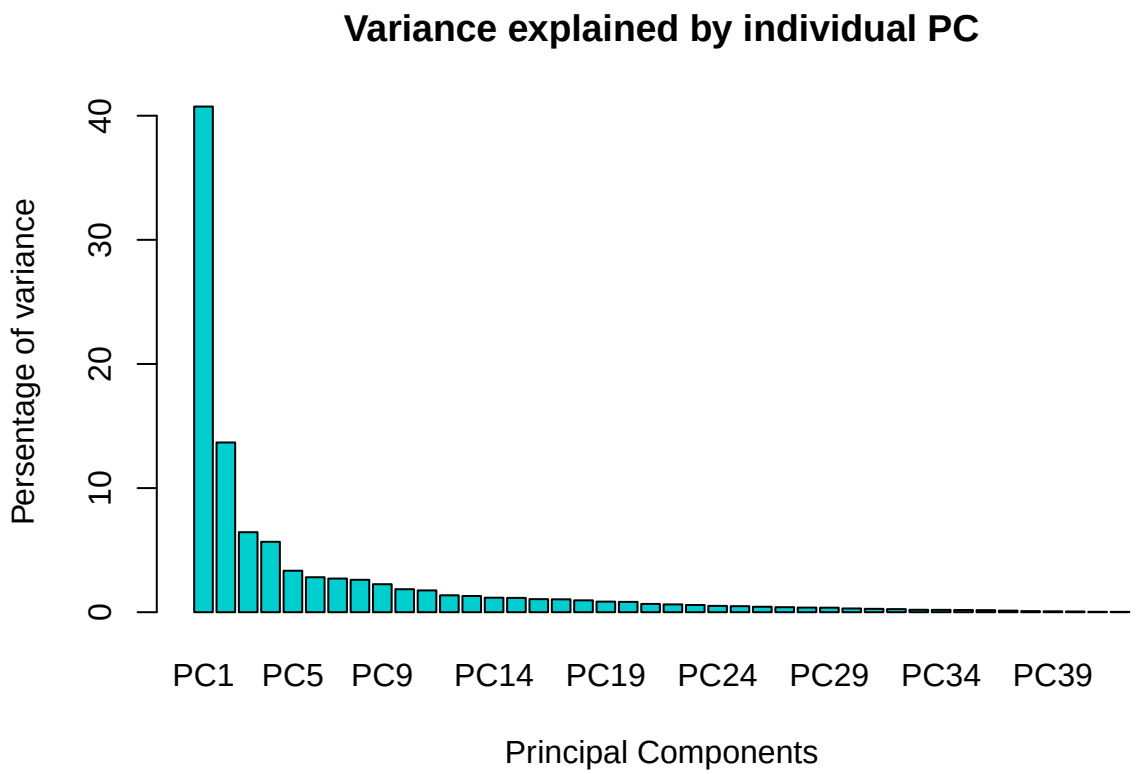
Principal Component Analysis (PCA)

PCA is a great tool for data visualization and dimentionality reduction. Let's create principal components for our data by applying function `prcomp()` and find out how much varience of the data is explained by each principle component.

```
# Transpose the data
data_pca <- t(data_norm)

# Create Principal Components by applying function prcomp()
my_pca <- prcomp(data_pca, scale=T, center=T)
imp <- summary(my_pca)$importance
```

```r
# Plot the variance covered by each PC
barplot(imp[2,]*100, ylab = "Persentage of variance", xlab = "Principal Components", main = "Variance e
```

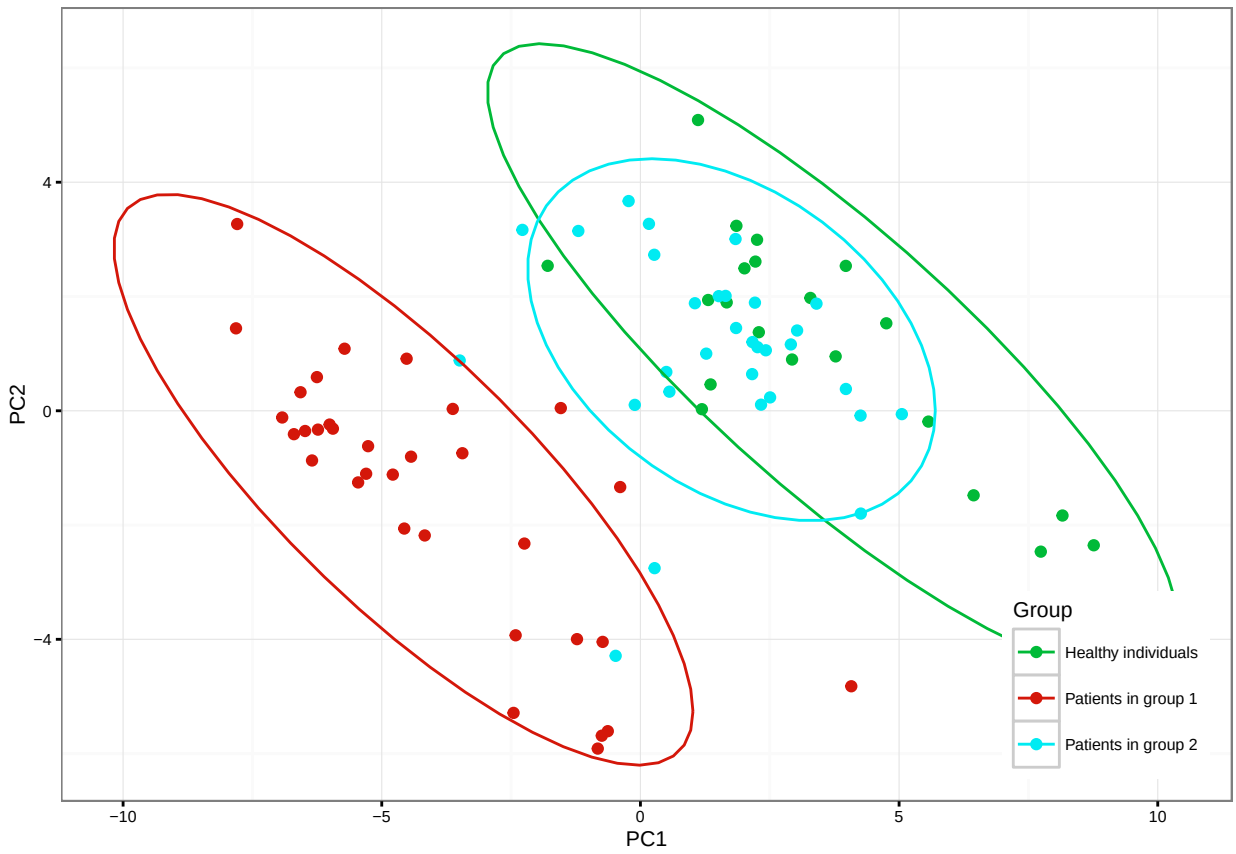


Now we can visualize the results of PCA using ggplot2.

```r
# Plot with ggplot and color samples by group
my_pca <- data.frame(my_pca$x)
# Add group annotations
my_pca <- cbind(my_pca, Group = rownames(data_pca) )
# Remove the indexes

my_pca$Group <- gsub("H.*","Healthy individuals", my_pca$Group)
my_pca$Group <- gsub("PG1.*","Patients in group 1", my_pca$Group)
my_pca$Group <- gsub("PG2.*","Patients in group 2", my_pca$Group)

# Plot PCA with ggplot2
ggplot(my_pca, aes(x = PC1, y = PC2, colour = Group)) +
  theme_bw(base_size = 8) +
  geom_point()+
  labs(x = 'PC1', y = 'PC2') +
  scale_colour_manual(values = c("#00ba38","#d4170a","#00ebf2")) +
  theme(
    legend.position = c(1, 0),
    legend.direction = "vertical",
    legend.justification = c(1, 0)
  )+
  stat_ellipse()
```



Cluster your data

K-means clustering

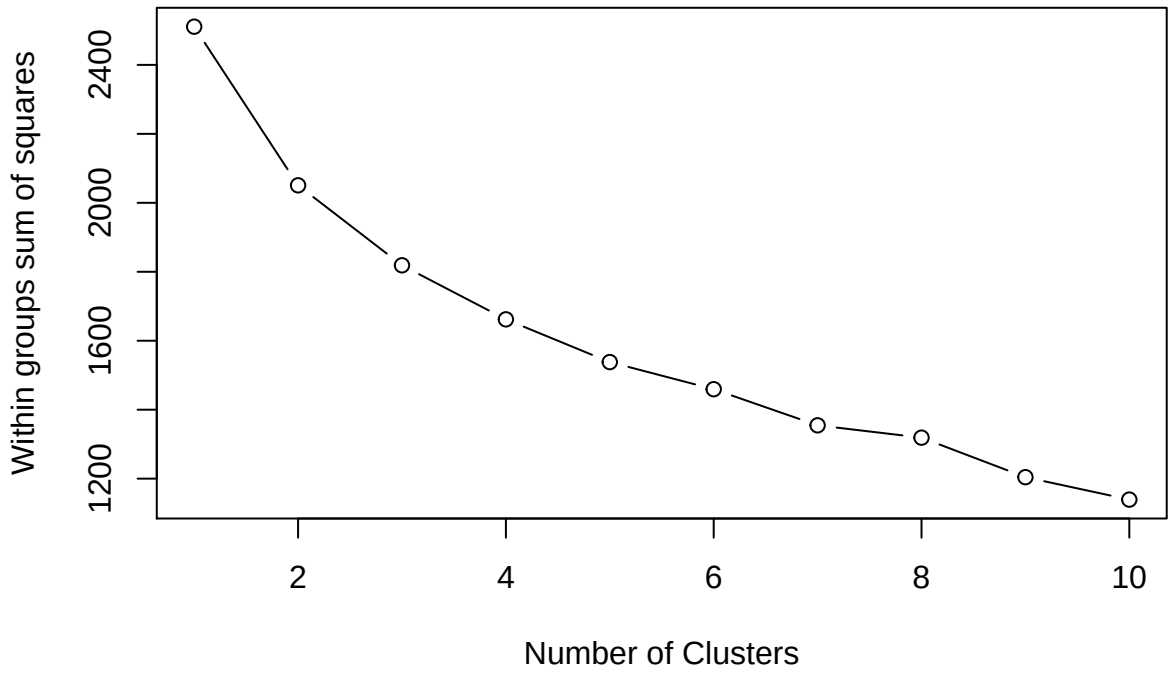
First of all you want to understand how many K(s) to select. To do that you could use so called "elbow" rule. The appropriate number of clusters can be selected based on the the sum of squared error (SSE) for a number of cluster solutions. SSE is defined as the sum of the squared distance between each member of a cluster and its cluster centroid. An appropriate cluster solution is the solution at which the reduction in SSE decreases dramatically. This produces a bending point in the plot of SSE against cluster solutions.

```
# Install package 'amap'
# install.packages("amap")

# Attach package 'amap'
library(amap)

# Calculate and plot SSE for different number of k in k-means
SSE <-  (nrow(data_norm)-1)*sum(apply(data_norm,2,var))
for (i in 2:10) SSE[i] <- sum(kmeans(data_norm,
                                     centers = i)$withinss)

# Plot the results with k from 2 to 10
plot(1:10, SSE, type = "b", xlab = "Number of Clusters",
     ylab = "Within groups sum of squares")
```



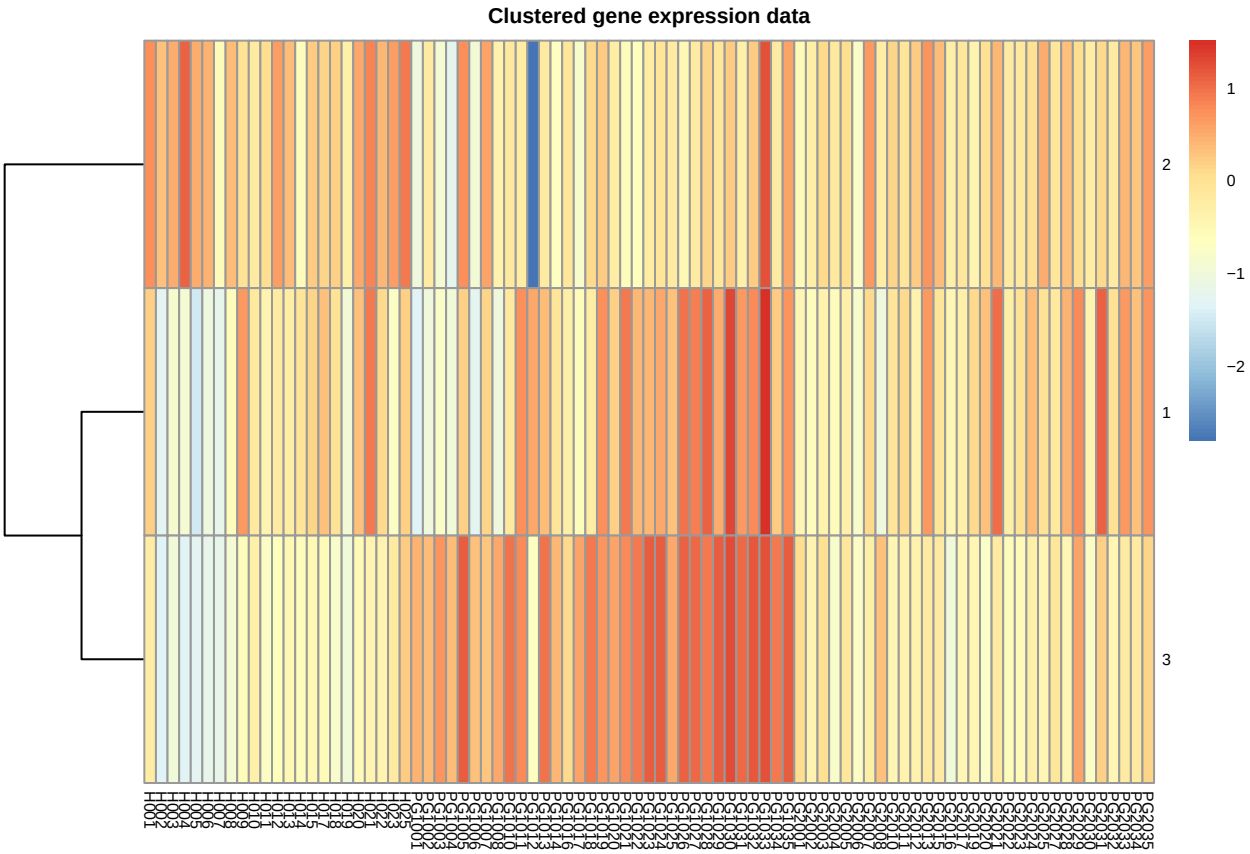
Let's select number of clusters 3 and run K-means clustering on our data.

```
km <- Kmeans(data_norm, 3, iter.max = 100500, nstart = 50,
          method = "correlation")

# Explore the structure
str(km)
```

```
## List of 4
##  $ cluster : Named int [1:42] 3 2 3 3 2 1 3 3 3 3 ...
##   ..- attr(*, "names")= chr [1:42] "AIM2" "CASP1" "CCL2" "CCL20" ...
##  $ centers : num [1:3, 1:87] 0.186 0.746 -0.245 -1.262 0.341 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "1" "2" "3"
##   .. ..$ : chr [1:87] "H001" "H002" "H003" "H004" ...
##  $ withinss: num [1:3] 0.0649 0.1184 0.0288
##  $ size    : int [1:3] 10 6 26
##  - attr(*, "class")= chr "kmeans"
```

```
pheatmap(data.frame(km$centers), cluster_rows = T, cluster_cols = F, scale = "none",
         clustering_distance_rows = "correlation",
         border_color = "grey60", fontsize = 6,
         main = "Clustered gene expression data")
```



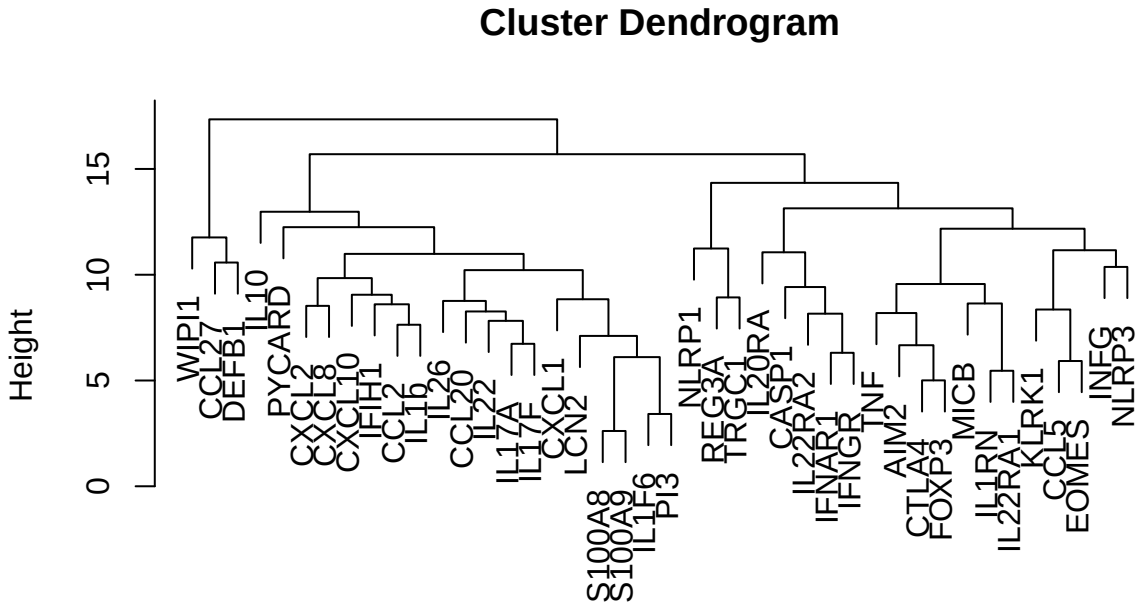
Hierarchical clustering

Cluster you data using hierarchical clustering. Apply function `hcluster()`.

```
library(amap)

hc_data <- hcluster(data_norm, method = "euclidean", diag = FALSE, upper = FALSE,
                 link = "complete", members = NULL, nbproc = 2,
                doubleprecision = TRUE)

# Plot the clustering results
plot(hc_data)
```




```
# Split into 3 clusters
ct=cutree(hc_data, k=3)
sort(ct)
```

```
##    AIM2   CASP1    CCL5   CTLA4   EOMES   FOXP3  IFNAR1    INFG   IFNGR
##       1       1       1       1       1       1       1       1       1
##   IL1RN  IL20RA IL22RA1 IL22RA2   KLRK1    MICB   NLRP1   NLRP3   REG3A
##       1       1       1       1       1       1       1       1       1
##     TNF   TRGC1    CCL2   CCL20   CXCL1  CXCL10  CXCL2   IFIH1    IL10
##       1       1       2       2       2       2       2       2       2
##   IL17A   IL17F    IL1b   IL1F6    IL22    IL26   CXCL8    LCN2     PI3
##       2       2       2       2       2       2       2       2       2
##  PYCARD  S100A8  S100A9   CCL27   DEFB1   WIPI1
##       2       2       2       3       3       3
```

```
# Save results in .RData file format
#save(ct, hc_data, file="results/hc.RData")
```

Extract knowledge

Gene Ontology cluster annotations

After you have identified the clusters, you can characterise the genes that are located in each of them. We can identify the biological processed that they are involved in using package `GOsummaries` or using a web-tool gProfiler http://biit.cs.ut.ee/gprofiler/.

```
# Get gene names in each of the cluster
g1 <- as.character(names(sort(ct[ct==1])))
g2 <- as.character(names(sort(ct[ct==2])))
g3 <- as.character(names(sort(ct[ct==3])))
```

Save the genes from one cluster to the file and execute query in gProfiler web tool. Go to http://biit.cs.ut.ee/gprofiler/ and paste the list of genes from your cluster. You will see the list of processes they are involved in and the level of significance.

```
write.table(g1, file="results/genelist.txt",sep="\t",row.names = F,col.names = F, quote = F)
```

Alternatively use R package `GOsummaries` to annotate the clusters and produce pretty pictures.

```
# Add annotations
library(GOsummaries)
```

```
## Loading required package: Rcpp
```

```
gl <- list(Cluster1 = g1, Cluster2 = g2, Cluster3 = g3)
gs <- gosummaries(gl)

#plot(gs, fontsize = 8)

# Create samples' annotation
Groups <- colnames(data_norm)
Groups <- gsub("H.*", "Healthy individuals", Groups)
Groups <- gsub("PG1.*", "Patients in group 1", Groups)
Groups <- gsub("PG2.*", "Patients in group 2", Groups)

my_annotation <- cbind(Groups, Groups)
rownames(my_annotation) <- colnames(data_norm)
my_annotation <- as.data.frame(my_annotation[,1])
colnames(my_annotation) <- "Groups"
gs_exp <- add_expression.gosummaries(gs, exp = data_norm,
                                     annotation = my_annotation)
```

```
## Using  as id variables
```

```
## Using  as id variables
## Using  as id variables
```

```
#plot(gs_exp, fontsize = 8, classes = "Groups")
```