R Workshop #7: PCA

Nicola RomanÃ²

Learning objectives

After completing this workshop you should be able to

- Visualize high-dimension data
- Perform unsupervised clustering on a dataset
- Perform and interpret PCA on a large dataset

Introduction

In this workshop we will have some hands-on practice with some of the topics that you have learned in Semester 2.

During this workshop we will use a fairly large dataset containing gene expression data from several cell lines, which are part of the NCI-60 panel. This is a panel of 60 cancer cell lines developed in the 1990s by the National Cancer Institute[1].

The original dataset can be retrieved from NCBI[2]; today we will use a smaller dataset, containing expression data for only some of the cell lines.

[1] See `https://en.wikipedia.org/wiki/NCI-60` and `https://dtp.cancer.gov/discovery_development/nci-60/cell_list.htm` for more info

[2] `https://www.ncbi.nlm.nih.gov/sites/GDSbrowser?acc=GDS4296`

Preparing the data

There are two files called `NCI60.csv.gz`[3] and `info.csv`. The first contains gene expression data for >50000 transcript (from a microarray experiment[4]), while the latter contains information about the samples.

The first step is to prepare your data; to make your life easier for later you should generate:

- one variable with the gene names
- one matrix with the expression levels (use `as.matrix` to convert the data frame to a matrix)
- one variable with the sample information

Normally we arrange our data to have observations on the rows and variables on the columns. However, our expression matrix has cell lines as columns and genes as rows. We can swap rows and columns in our matrix using the `t` function[5]. For example, if your expression matrix were called `expr` you would transpose it using:

```
expr <- t(expr)
```

[3] Note: .gz files are compressed files. R is able to read these files directly, using `read.csv`.

[4] Note that the human genome has only ~20000 genes; in a microarray experiment, multiple probes are used to measure transcript for the same genes, that is why the number of probes here is so high.

[5] The mathematical term for this is to *transpose the matrix*

Visualising the data

You may now start by visualising/exploring the data.

Try answering the following questions:

- How many type of cancers are represented in the dataset?
- How many cell lines from each type?
- Are the levels of the Ras gene (HRAS) changed between different groups?

Now, think of this dataset in relation to the ones you have worked with in the previous workshops. Is it easy to explore and plot the data or do you find any challenging aspects in this dataset?

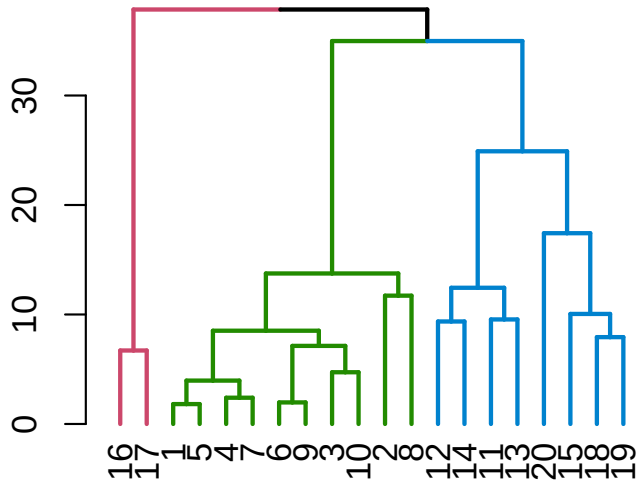
What if I wanted to know what genes are different between leukemia and melanoma? Clearly, trying one gene at a time is not the answer; furthermore the difference may lie in the specific pattern of expression of multiple genes...

How do we go about solving this problem? First of all we may want to try some way of visualising all of our data together.

Clustering our data

The first thing we can see is whether there is some structure to the data. We can use a *dendrogram*. This is a way of displaying how distant data points are in our multi-dimensional parameter space.

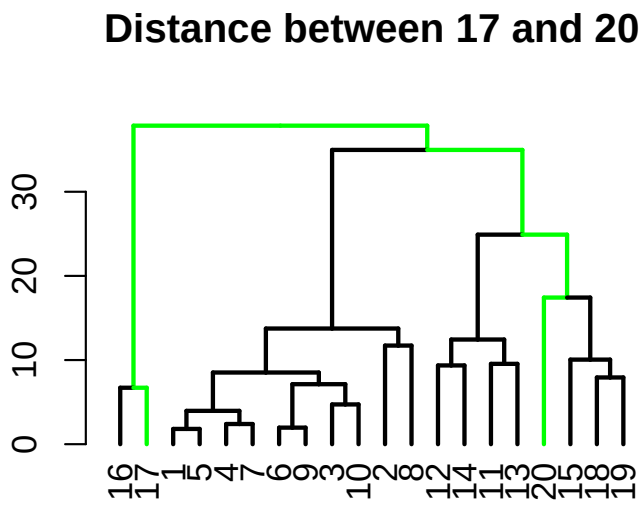
For example, consider this plot:



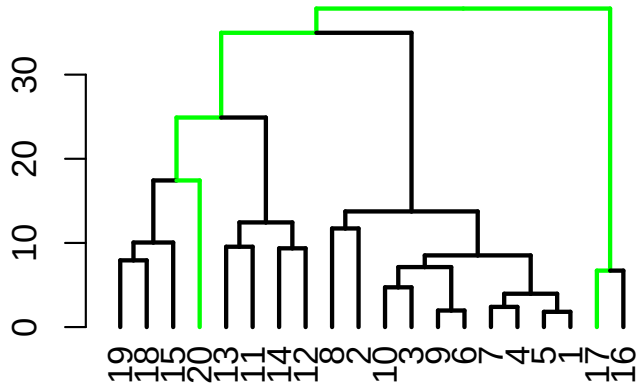
This plot shows 20 samples, that cluster in 3 main branches[6]. Samples in the same branch are more similar to themselves then to samples in other branches. To see how distant two samples are we can start from the *leaf node* corresponding to one sample and "walk" our way to the other sample.

For example, the distance between 17 and 20 is marked by the green line in the following plot.

[6] Note: this is a type of *unsupervised clustering*. To generate this plot, I have not told R that there were 3 groups!



Note that the order of the branches is arbitrary. The following dendrogram is equivalent to the one above.



Let's try to create a dendrogram of our data! We first need to calculate a distance between each sample. This can be done using the

`dist` function.

There are many ways of calculating distances, but here we use *Euclidean* distance[7], which is what you do, for example, when you apply Pythagora's theorem. In n-dimensions, the Euclidean distance between two points $P(P_1, P_2, ..., P_n)$ and $Q(Q_1, Q_2, ..., Q_n)$ is

$$d(P, Q) = \sqrt{(P_1 - Q_1)^2 + (P_2 - Q_2)^2 + ... + (P_n - Q_n)^2}.$$

[7] There are many other distance methods, you can read the help of the `dist` function for more information.

We are going to calculate the distance of the first 100 points only[8].

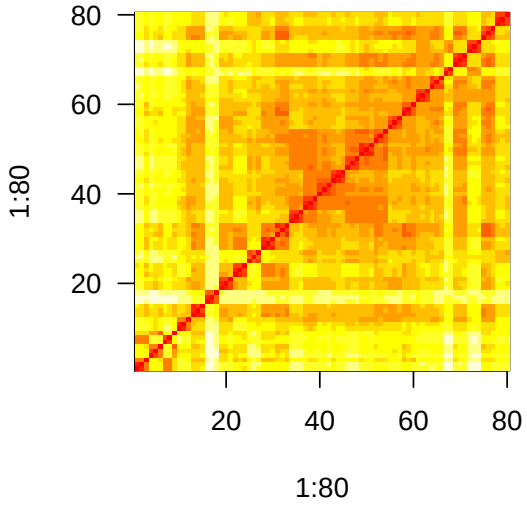
[8] **ATTENTION**: do not attempt to use `dist` on the whole dataset, it is likely to crash your computer.

```
# Calcuate the distance in a 100-dimensional
# space!  This assumes you have transposed the
# matrix as explained above
dst <- dist(expr[, 1:100], method = "euclidean")
```

We can draw the distance matrix using the `image` function. This shows a *heatmap*, a common way to represent matrix data. A heatmap consists of many little squares, one for each observation. In this case, our distance matrix will be 80 x 80, each corresponding to the distance between two cell lines. For example, the square at (10;20) is the distance between cell line 10 and 20[9]. `image` will show low values in red, mid values in yellow, and high values in white.

[9] The matrix is symmetrical with respect to the diagonal. Why?

```
image(x = 1:80, y = 1:80, z = as.matrix(dst),
    las = 1)
```



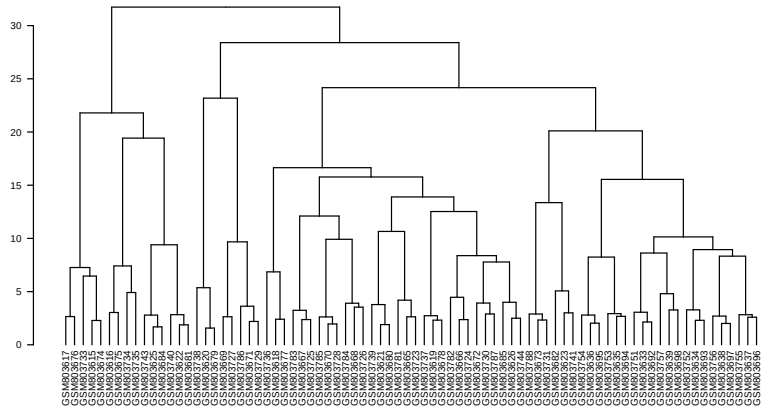
Clearly, from the image above, we can see that there are various

levels of similarity between our cell lines. Do you see anything interesting in this plot?

Let's now create the dendrogram. The `hclust` function, performs unsupervised hierarchical clustering of data, and can output a dendrogram. There are many methods to cluster the data, today we are going to use the Ward method, but there are other options[10].

[10] see `?hclust` for more details. Feel free to experiment with different clustering methods

```
hc <- hclust(dst, method = "ward.D2")
d <- as.dendrogram(hc)
# I am changing the cex plot parameter so that
# labels are visible This may not be necessary
# for you
par(cex = 0.5)
plot(d, las = 1)
```




```
par(cex = 1)  # Back to original cex
```

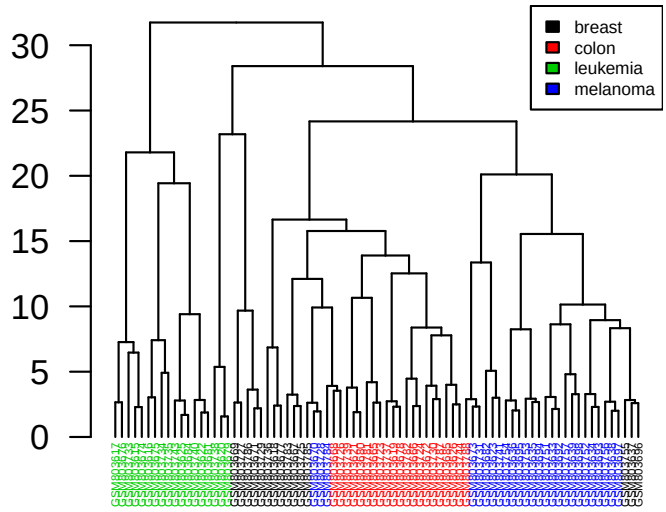
Customising the way dendrograms are plotted with the base R functions is quite complex. The `dendextend` package[11] by Tal Galili makes life much much easier. `dendextend` syntax may be a bit unusual for you, but you can find a full explanation at this address: `https://cran.r-project.org/web/packages/dendextend/vignettes/introduction.html`. For today we are not going to use this anymore, so you may want to just copy and paste the following code, and come back to it later if interested.

[11] Install it as usual `install.packages("dendextend")`

The following colors the labels according to the type of tissue.

```
d %>% set("labels_col", as.integer(info$tissue)) %>%
    set("labels_cex", 0.3) %>% plot(las = 1)

legend("topright", fill = 1:4, legend = levels(info$tissue),
    cex = 0.5)
```



With the exception of some melanoma and breast cancer lines, we can see that even using only 100 of the >50000 transcripts already shows the difference between cell lines! This tells us that:

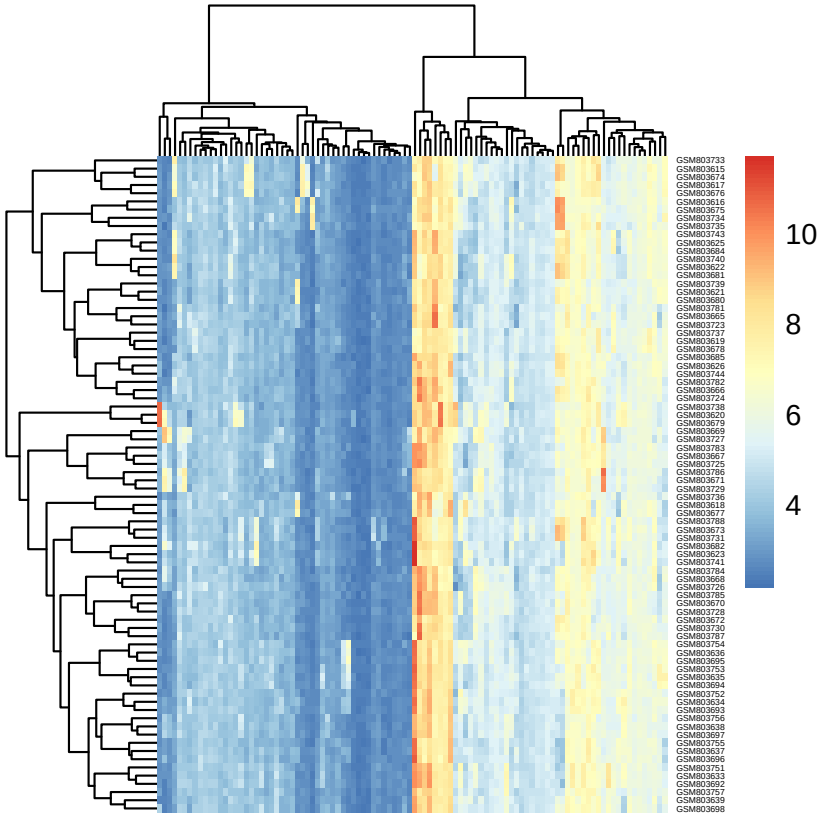
- Different cell lines have a specific gene signature
- The differences in gene signature are more dependent on the tissue of origin, rather than on the specific line
- There are some cell lines which do not "fit the pattern", and it may be worth (biologically) investigating why

Heatmaps can also be used to plot the actual data (rather than the distance), and we can combine them with dendrograms usingthe `pheatmap` function in the `pheatmap` package.

We will plot a heatmap of the first 100 genes in the dataset[12]. `pheatmap` automatically clusters and draws dendrograms for both the cell lines and the genes.

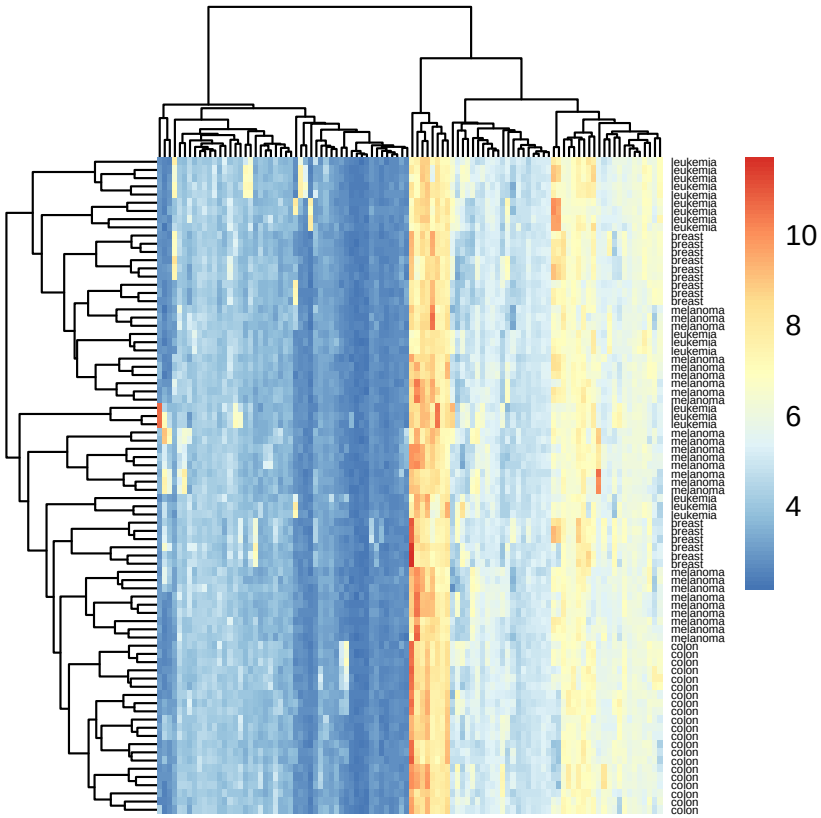
```
library(pheatmap)
# fontsize_row: the font size for the row
# (cell lines) labels border_color: the color
# of the borders, we set it to NA for no
# border for visual clarity
pheatmap(expr[, 1:100], fontsize_row = 3, border_color = NA)
```

[12] **ATTENTION**: as before, do not attempt to create a heatmap of the whole dataset, it is likely to crash your computer. You should be able to safely plot 1000-2000 genes though! Feel free to experiment with this, but save your work beforehand!



We can print the tissue of origin instead of the cell line name to make the plot more useful.

```r
rownames(expr) <- info$tissue
pheatmap(expr[, 1:100], fontsize_row = 4, border_color = NA)
```




```r
# Revert to sample names
rownames(expr) <- info$sample
```

- What can you conclude from this plot?
- Try and use a different set of genes (pherhaps a random set!), does that change the output?
- Try and calculate the variance in the expression of each of the genes (you can use the `var` or the `sd` function to do so), then select the 100 genes with the highest variance and re-plot the heatmap.
- Now do the same with the 100 genes with the lowest variance. Is there any difference?
- Do you think looking at variance is a good strategy? Why?

PCA

So far, we have only used some of the genes from our dataset which, in this particular case happened to give quite satisfactory results; however, what if some interesting difference is present in the other >50000 columns that we did not consider?

You have learned about dimension reduction techniques, specifically about PCA. Let's try it on our data!

Running PCA in R is extremely simple, using the `prcomp` function. Remember that our data needs to be centered around 0; we could do this manually, or ask `prcomp` to do it. We can also ask it to scale the data, so that each variable (gene) has a variance of 1. This is generally helpful to avoid variables with sustantially bigger variance to have more weight in the PCA output.

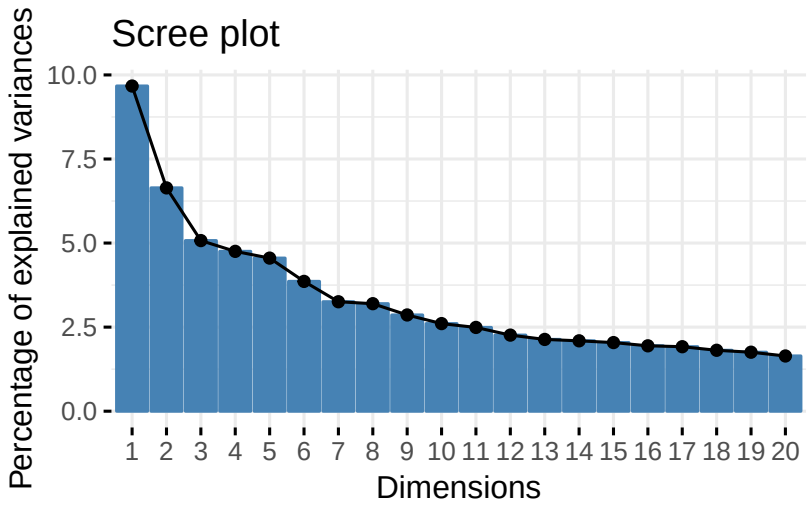
```
# Note that we are using the whole dataset
# this time!
pca <- prcomp(expr, center = TRUE, scale = TRUE)
```

To plot the result of our PCA we will use the `factoextra` package. This provides a series of visualization functions (`fviz_...`) to create nice PCA plots.

We will start by looking at the scree plot[13] of the first 20 principal components.

[13] Look back at the lecture if you do not remember what that is!

```
library(factoextra)
# Scree plot
fviz_eig(pca, ncp = 20)
```

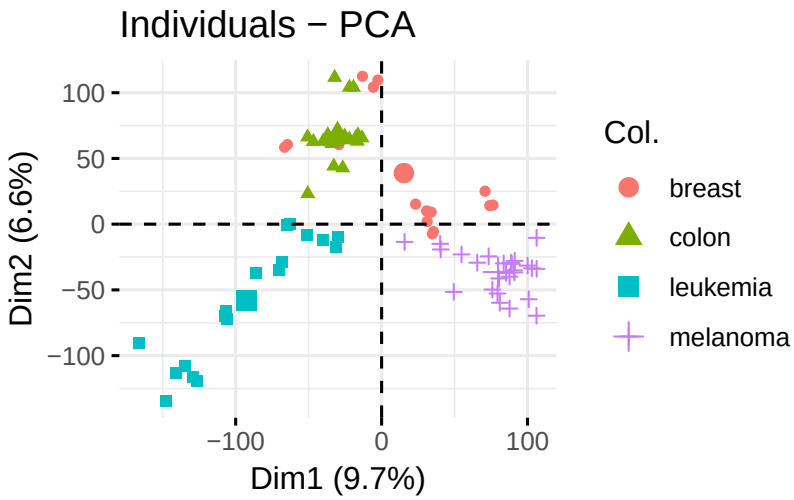


- How do you interpret this plot?
- How much variance is explained by the first 10 PCs[14]?.

[14] You can find the variance associated to each PC by squaring `pca$sdev`

Now let's plot the first two components.

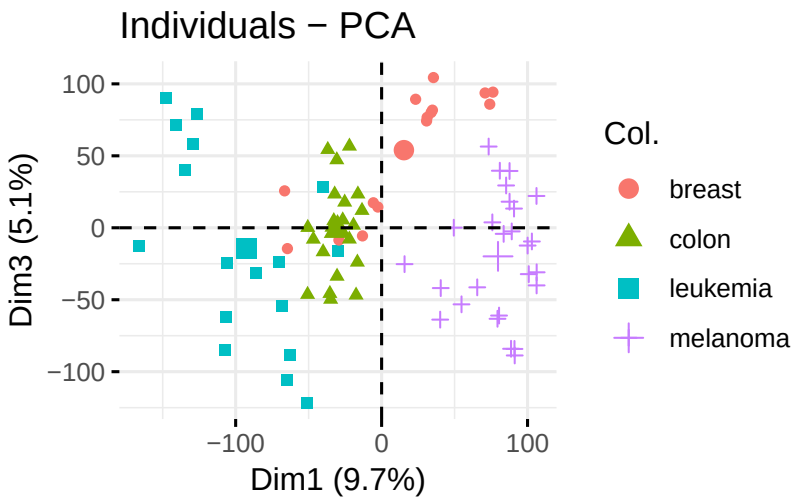
```
fviz_pca_ind(pca, col.ind = info$tissue, geom.ind = "point",
    axes = c(1, 2))
```



You can see that cells mostly cluster in separate groups, with the exception of some breast cancer lines. The function also plots the center of each group as a data point with bigger size; this is helpful if you want to look for general changes in your data.

We can visualise other components as well

```
fviz_pca_ind(pca, col.ind = info$tissue, geom.ind = "point",
    axes = c(1, 3))
```



- How does this plot compare to the previous one?
- Try to plot other PCs. What do you observe when plotting PCs with higher number? Why?

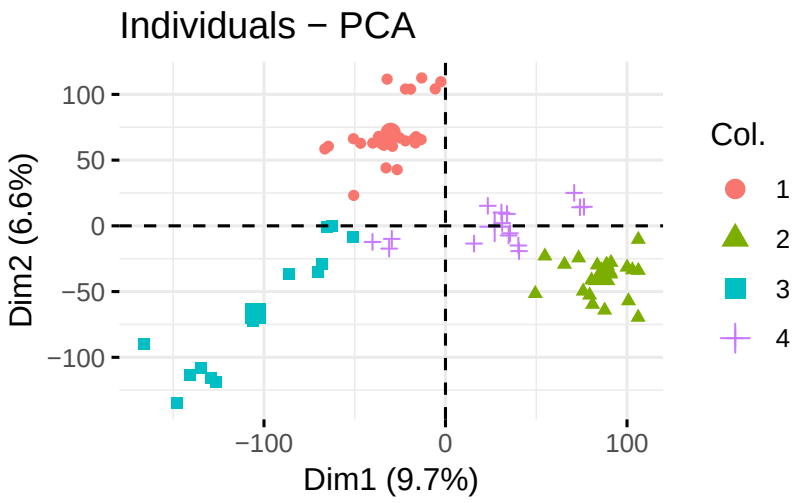
Clustering the PCA results

In this case we already know what tissue these cell lines come from, but that may not always be the case. In those situations we may want to try and cluster the observations in the PC space ourselves.

For example, we could use the k-means algorithm you have been told about in the lectures. I will use k-means on the first 20 PCs (accounting for ~66% of the dataset variance), asking for 4 clusters. We will then color the points according to the clusters assigned by the k-means algorithm[15].

[15] We are "cheating" a little bit here, since we know that, in reality, there are 4 types of tissue... in other situations choosing the number of clusters may not be as easy!

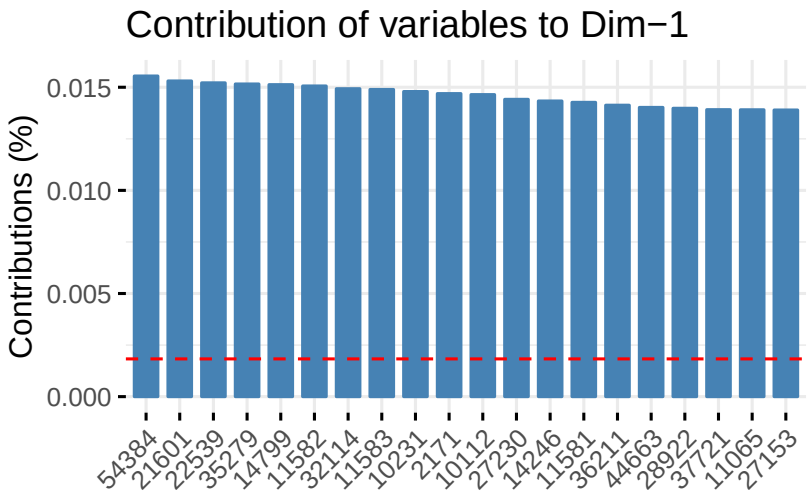
```
# We try 100 different initial centroid
# positions by setting nstart = 100
km <- kmeans(pca$x[, 1:20], 4, nstart = 100)
fviz_pca_ind(pca, col.ind = as.factor(km$cluster),
    geom.ind = "point")
```



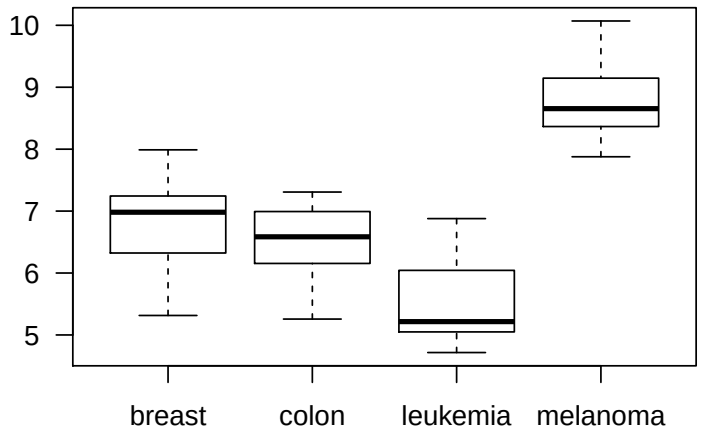
- Does k-means clustering correctly divide the samples?
- Try different values of nstart. Does this make a difference?
- Does k-means correctly identify the clusters in the data? Where does it fail?

Finally, we can also ask R to tell us which variables are the ones that most contribute to a certain PC.

```
fviz_contrib(pca, choice = "var", top = 20, axes = 1)
```




```
boxplot(expr[, 54384] ~ info$tissue, las = 1)
```



In this example, variable 54384 (corresponding to gene SASH1) is the one that most discriminates over PC1, albeit accounting for just a very small percentage of the variance. By plotting it we see that it is indeed quite different between the different groups.

Further work

Further analysis of genomic data (microarray, RNA-seq, CHIP-seq, etc) can be quite complex and definitely out of the scope of this workshop. Further analysis may involve differential gene expression

analysis, to identify genes that are over- or underexpressed in specific clusters, as well as pathway analysis to see if component of specific molecular pathways are expressed in certain clusters or conditions. This can invove using gene ontology (GO) descriptors[16].

Examples of how to perform these can be found here[17]:

- `http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html`
- `https://bioconductor.org/packages/release/bioc/vignettes/topGO/inst/doc/topGO.pdf`

For

[16] See `https://en.wikipedia.org/wiki/Gene_ontology` and `http://geneontology.org/`

[17] Note that these are just some of the possible analysis that can be performed, there is a lot more!