Group 02 - Skin Cancer

Leonie Thomas, Isabel Potthof, Elif Tosun and Marlene Khin

24.07.2019

# Preparations

## 1.Load following packages:

library(ggplot2)  
library(relaimpo)  
library(factoextra)  
library(gridExtra)  
library(reshape2)  
library(data.table)  
library(cluster)  
library(rstudioapi)  
library(pheatmap)  
library(caret)  
library(tidyverse)  
library(dendextend)  
library(factoextra)  
library(devtools)  
library(ggfortify)  
library(rstudioapi)   
library(data.table)   
library(ggplot2)   
library(scales)   
library(stats)  
library(caTools)

## (2. Setting the sys-path and loading the data)

We used the sys-path to work in R, but markdown could not knit it so we loadet the dataset for the markdown like explained in point 3.

root.dir = dirname(rstudioapi::getSourceEditorContext()$path)  
data = readRDS(paste0(root.dir, "/DepMap19Q1\_allData.RDS"))

## 3. Loading the dataset:

data = readRDS("C:/Users/LeoTh/Documents/GitHub/project-01-group-02/DepMap19Q1\_allData.RDS")

# Part 1: Data Cleanup

## 1.1 Extracting and splitting our data

The mutation data is different from the other matrices, so we define a new matrix only containing the mutation dat.

mut <- data$mutation

Additionally to the mutation matrix we want another matrix only containing all data exept the mutation data.

'%!in%' <- function(x,y)!('%in%'(x,y)) #We define an operator that will only pick the data that is NOT defined in the list; so the data that needs to be excluded  
dt\_new <- lapply(which(names(data) %!in% "mutation"), function(a) data[[a]]) #extracting the non-mutation data   
names(dt\_new) <- names(data)[which(names(data) %!in% "mutation")] #rename the data with the original names

Defining which samples we will take out of the original dataset

sample\_case = c("Skin Cancer")

We look at the annotation matrix and search only for the Primary diseases which match the previous defined sample\_case. We are looking at the column which tells us to which cancer type this cell line belongs and only taking the cell lines for which the sample case ist true. We are getting a vector of all the cell line names we want to look at

samples = data$annotation$DepMap\_ID[which(data$annotation$Primary.Disease == sample\_case)]

We extract all cell lines we defined in the previous step out of our data (except the mutation matrix)

processed\_data <- lapply(1:length(dt\_new), function(a) { #picking the data for our sample   
 dat\_picker <- dt\_new[[a]] #picking one file at each iteration   
 if(names(dt\_new[a])== "annotation"){ # treating the annotations differnetly because the cell line names are in a colum and are not the columnames like in the other matrices  
 output <- dat\_picker[which(dat\_picker[,1] %in% samples),]  
 } else {  
 output <- dat\_picker[,which(colnames(dat\_picker) %in% samples)]# only taking the skin cancer cell lines   
 output <- output[complete.cases(output),] # only taking rows without NAs   
 output <- output[order(rownames(output)),] # reordering the Genes acording to their name  
 }  
 return(output)  
})  
names(processed\_data) <- names(dt\_new) # rename the objects according to the original data  
rm(dt\_new,sample\_case) # remove objects we don`t need anymore

Now we extract our cellines from the mutation data.

ids = which(names(mut) %in% samples)   
allDepMap\_mutation\_SkinCancer = lapply(ids, function(a) {  
 mut[[a]]})  
rm(mut, ids, data) #tidying

Losing the mutations which are not deleterious meaning not interesting to us.

allDepMap\_mutation\_SkinCancer = lapply(1:34, function(a) {  
 allDepMap\_mutation\_SkinCancer[[a]][which(allDepMap\_mutation\_SkinCancer[[a]][,"isDeleterious"]== TRUE), ]  
 })  
names(allDepMap\_mutation\_SkinCancer) <- samples

Losing all the genes which are not in every data frame. First we need to pick all Gene names we have out of our data.

Genenames <- unique(c(rownames(processed\_data[[1]]),rownames(processed\_data[[2]]),rownames(processed\_data[[3]]),rownames(processed\_data[[4]])))

Then we pick these genes which are in all 4 dataframes we need for further analysis.

i <- 1  
out <- vector("character", length(seq\_along(1:16970)))  
for (x in seq\_along(Genenames)) {  
 if(Genenames[x] %in% rownames(processed\_data$expression) & Genenames[x] %in% rownames(processed\_data$copynumber) & Genenames[x] %in% rownames(processed\_data$kd.ceres) & Genenames[x] %in% rownames(processed\_data$kd.prob))  
 {out[i] <- Genenames[x]  
 i <- i+1  
 }   
}  
  
allDepMap\_annotation\_SkinCancer <- processed\_data$annotation # saving the annotation object in a seperate dataframe  
# because it doesnt contain any information about the genes   
  
processed\_data <- lapply(processed\_data[1:4], function(a) {  
 a <- a[which(rownames(a) %in% out),]  
 return(a)  
})  
  
processed\_data$mutation <- allDepMap\_mutation\_SkinCancer  
processed\_data$annotation <- allDepMap\_annotation\_SkinCancer  
rm(i,out, Genenames,x, allDepMap\_annotation\_SkinCancer, samples, allDepMap\_mutation\_SkinCancer)

# Part 2: Data vizualisation

## 2.1 Preparing our data for plotting

### 2.1.1 Extracting our data for plotting

We will not need all our data for plotting so we have to prepare our data for the following plots.

generalPlottingData <- lapply(1:(length(processed\_data)-2), function(a) { # we will not need annotation  
 dtPicker <- processed\_data[[a]]  
 out <- melt(dtPicker) #bind the data togehter that we have samples and values as columns  
 out$Gene <- rep(rownames(dtPicker), ncol(dtPicker)) #add the genes; probably this might be useful in a later stage  
 out$Case <- names(processed\_data)[1:(length(processed\_data)-1)][a] #add a labelling column  
 colnames(out) <- c("Sample", "Value", "Gene", "Case") #rename the columns  
 return(out)  
})

## No id variables; using all as measure variables  
## No id variables; using all as measure variables  
## No id variables; using all as measure variables  
## No id variables; using all as measure variables

names(generalPlottingData) <- names(processed\_data)[1:(length(processed\_data)-2)] #rename the data

### 2.1.2 Plotting Data - Driver Mutations

Producing a vector which encompases every gene which at least mutated once.

singleGenes <- as.vector(unique(as.data.frame(rbindlist(lapply(seq\_along(processed\_data$mutation), function(a) {  
 out <- as.data.frame(as.vector(unique(processed\_data$mutation[[a]]$Hugo\_Symbol)))}))))[,1])

Creating a dataframe which contains how often every gene is mutated.

geneCounts <- sapply(seq\_along(singleGenes), function(a) {  
 genePicker <- singleGenes[a] #pick one gene  
 sumGene <- lapply(seq\_along(processed\_data$mutation), function(b) {  
 mutPicker <- processed\_data$mutation[[b]] #pick one of the 34 mutation lists  
 out <- as.data.frame(length(which(mutPicker$Hugo\_Symbol == genePicker))) #look how often an entry is in the mutation list  
 return(out)  
 })  
 geneCount <- colSums(as.data.frame(rbindlist(sumGene))) #sum it up to get the total count for each gene  
 return(geneCount)  
})  
names(geneCounts) <- singleGenes #rename   
geneCounts <- as.data.frame(geneCounts) #make a nice dataframe  
colnames(geneCounts) <- c("Value")  
geneCounts <- geneCounts[order(-geneCounts$Value), , drop = FALSE] #sort the data frame  
head(geneCounts)

## Value  
## TTN 13  
## TP53 9  
## HMCN1 8  
## TMTC2 7  
## RYR2 7  
## CACNA1I 7

Extacting the data for the top 10 which will be our driver mutations in our further investigation.

dataTopDriverGenes <- lapply(1:(length(processed\_data)-2), function(a) { #picking the data for our sample   
 dat\_picker <- processed\_data[[a]] #pick one file at each iteration   
 output <- dat\_picker[which(rownames(dat\_picker) %in% rownames(geneCounts)[1:10]),] # compare the rownames of the picked data with the names of the 10 most mutated genes  
 return(output)  
})  
names(dataTopDriverGenes) <- names(processed\_data)[1:4]  
  
rm(singleGenes)

### 2.1.3 Extracting the drivermutations for every Celline

Putting all mutation data in one Matrix

OneMatrix <- data.frame()  
for (i in c(1:34)) {  
 OneMatrix <- rbind(OneMatrix,processed\_data$mutation[[i]][,Hugo\_Symbol:DepMap\_ID])  
}

Extracting just the column of the Gene name and the cell line

CelllinesMutations <- OneMatrix[which(OneMatrix$Hugo\_Symbol %in% rownames(geneCounts)[1:10] ),]  
CelllinesMutations <- cbind(CelllinesMutations$Hugo\_Symbol, CelllinesMutations$DepMap\_ID)

Extracting the drivermuations for every cell line out of the dataframe and putting it into another dataframe so it can be used for the plotting

Genes <- c("COL11A1,TMTC2,TTN", " HMCN1", "COL11A1,HMCN1,SLC510", "HMCN1,TMTC2", "COL11A1,TP53,TTN","none","ZNF292","RYR2","HMCN" ,"none2","none3", "TP53, TTN","HMCN1", "TTN,ZNF292","TMTC2,TP53,NEB","TP53", "TMTC2,NEB","none4","TMTC2,TTN,ZNF292", "none5","CACNA1I","HMCN1,TP53,ZNF292","none6","none7","HMCN1,TMTC2,ZNF292","RYR2,TMTC2,NEB","RYR2,NEB,TTN,CACNA1I","HMCM1,TP53","TTN","COL11A1,SLC5A10","COL11A1,CACNA1I","TTN,CACNA1I","RYR2,CACNA1I,ZNF292","TP53,TTN,CACNA1I" )  
Zelllines <- c(colnames(processed\_data$expression))  
zellinesMutations <- as.data.frame(cbind(Zelllines, Genes))  
  
rm(OneMatrix, Genes, ZelllinesMutations, Zelllines,i)

## Warning in rm(OneMatrix, Genes, ZelllinesMutations, Zelllines, i): Objekt  
## 'ZelllinesMutations' nicht gefunden

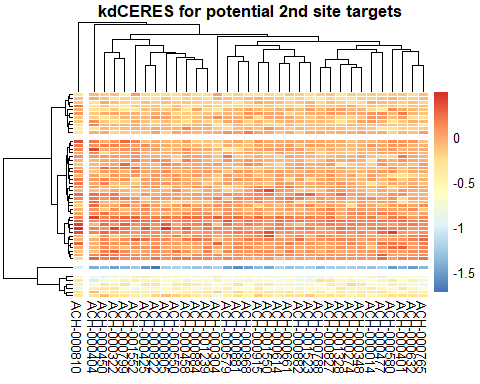
The explanation for the previous extraction will be outlined in the following visualization part

## 2.2 Visualizing our data

### 2.2.1 Heatmap with the knock down data

We start with a heatmap of our knock down data (the kd.ceres matrix). This matrix consist of gene knockdown scores. The impact of the knocked out gene on the cell survival is reflected by that score. The impact can be a reduction or an increase in proliferation. It could also mean that there is no change in cell proliferation at all. Smaller values refer to higher importance.

pheatmap(as.matrix(processed\_data$kd.ceres[1:50,]), clustering\_method = "ward.D2",border\_color = "white", fontsize = 10,   
 main = paste0("kdCERES for potential 2nd site targets"),  
 show\_rownames = F, show\_colnames = T,  
 cutree\_rows = 4,  
 cutree\_cols = 2,   
 fontsize\_row=10)

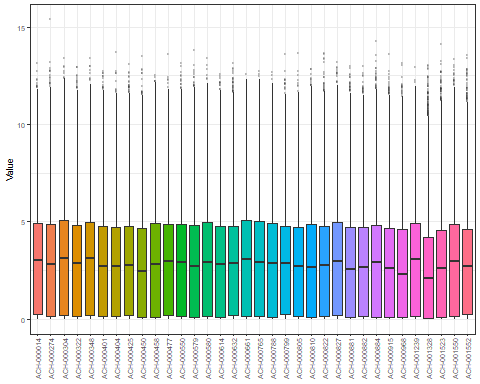


* *We can see that there are clear differences between the knockdown data depending on the knocked out gene in a specific cell.*
* *The cell lines behave differently when the same gene is knocked out.*

### 2.2.2 Distribution of the expression values between the different cell lines

We create a boxplot with the expression matrix to see how the expression of the genes is distributed over the different cell lines.

ggplot(data = generalPlottingData$expression, aes(x=Sample, y=Value)) +  
 geom\_boxplot(aes(fill = Sample), outlier.size = 0.1, outlier.alpha = 0.2) + #reconstruct the outliers a bit (so reduce them in size; because we are interested in the boxplots and not the outliers)  
 theme\_bw(base\_size = 7) + #format the size of the theme nicely  
 theme(legend.position= "none", #define the legend position (here no leghend will be needed)  
 legend.direction="horizontal", #define the legend direction if one is there  
 plot.title = element\_text(hjust = 0.5), #make the title of the plot into the middle  
 axis.text.x = element\_text(angle = 90, vjust = 0.5, hjust=1), #define the orientation of the text on the x-axis  
 legend.title= element\_blank(), #no title of the legend should be plotted  
 axis.title.x = element\_blank(), #no title of the x-axis is relevant; because that would be samples and that is cleare due to the naming  
 strip.text.y = element\_text(angle = 90)) #define the orientation of the text of the y-axis

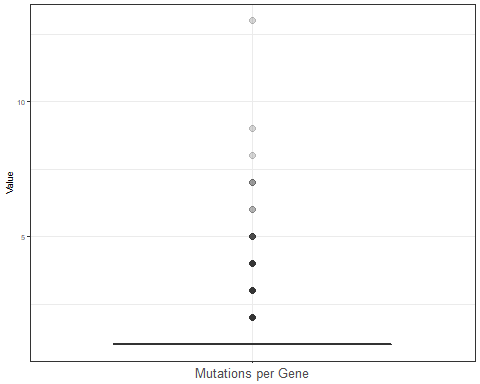


* *Many genes are distributed between the 25 and 75 quantile. But there are also some outliers which are of special interest for us in the following data analysis.*
* *For now we can say that the data is differnetly distributed between the celllines based on different mutations in the different cell lines.*

### 2.2.3 Plotting how often a gene is mutated over all cell line

This plot should show us the mutation rate of a gene.

geneCounts <- cbind(geneCounts, "Mutations per Gene")  
  
ggplot(data = geneCounts, aes(x="Mutations per Gene", y=Value)) +  
 geom\_boxplot(aes(fill = "Mutations per Gene"), outlier.size = 2, outlier.alpha = 0.2) + #reconstruct the outliers a bit (so reduce them in size; because we are interested in the boxplots and not the outliers)  
 theme\_bw(base\_size = 7) + #format the size of the theme nicely  
 theme(legend.position= "none", #define the legend position (here no leghend will be needed)  
 legend.direction="horizontal", #define the legend direction if one is there  
 plot.title = element\_text(hjust = 0.5), #make the title of the plot into the middle  
 axis.text.x = element\_text(angle = 0, vjust = 0.5, hjust= 0.5, size = 10), #define the orientation of the text on the x-axis  
 legend.title= element\_blank(), #no title of the legend should be plotted  
 axis.title.x = element\_blank(), #no title of the x-axis is relevant; because that would be samples and that is cleare due to the naming  
 strip.text.y = element\_text(angle = 90)) #define the orientation of the text of the y-axis

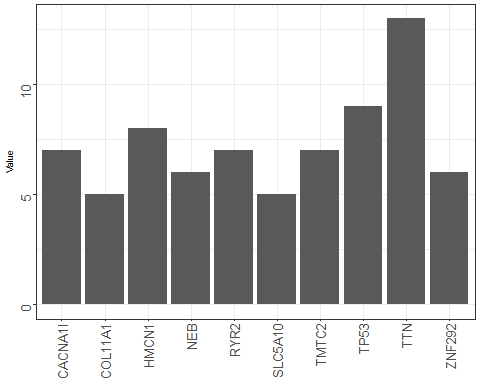


* *So we have seen that the expression values show a different distribution for the different cell lines. Different expression values can arise from gene mutaitons of specific genes*
* *So the question is if there are mutations occuring more often than others. We suspect that these may be one of the reasons for the differing expression values.*
  + So as we can see: yes there are Mutations which occure significantly more often.\*

### 2.2.4 Top 10 mutated genes

Now we want to see which Mutations are the top 10 mutated Genes.

plotData <- geneCounts[1:10, ,drop = FALSE]  
  
plotData$Gene <- rownames(plotData)  
  
ggplot(data = plotData) +  
 (geom\_bar(mapping = aes(x = Gene, y = Value), stat = "identity")) +  
 theme\_bw(base\_size = 7) + #format the size of the theme nicely  
 theme(legend.position= "none", #define the legend position (here no legend will be needed)  
 legend.direction="horizontal", #define the legend direction if one is there  
 plot.title = element\_text(hjust = 0.5), #make the title of the plot into the middle  
 axis.text.x = element\_text(angle = 90, vjust = 0.5, hjust=1, size = 10), #define the orientation of the text on the x-axis  
 axis.text.y = element\_text(angle = 90, vjust = 0.5, hjust=1, size = 10), #define the orientation of the text on the x-axis  
 legend.title= element\_blank(), #no title of the legend should be plotted  
 axis.title.x = element\_blank(), #no title of the x-axis is relevant; because that would be samples and that is cleare due to the naming  
 strip.text.y = element\_text(angle = 90)) #define the orientation of the text of the y-axis



rm(plotData)

* *These 10 Genes are our driver genes of which we want to identify interactions with other genes.*

# 3. Dimensionality reduction

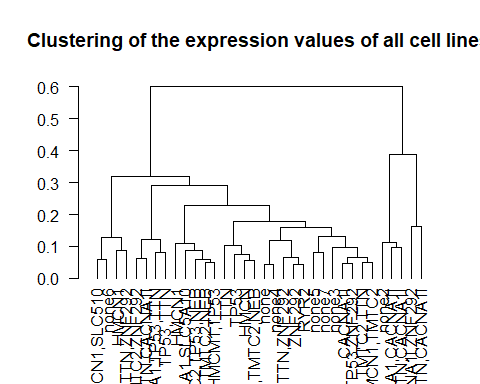
General questions:

* can we group the different Driver mutations together so that we can see in which other genes the Cell lines with a specific driver mutation differentiate?
* With Dimnsionality reduction we could gain insight which other genes are our secound targets.

## 3.1 Hierachical clustering

Creating a hierachical cluster with our driver mutations.

drivergene <- 3   
# determines which of the driver mutations will be seen in the cluster at the x axis  
dataset <- processed\_data$expression # determines which dataset we use  
  
colnames(dataset)[which(colnames(dataset) %in% unique(zellinesMutations[which(zellinesMutations[,1] == rownames(geneCounts)[drivergene]),2]))] <- rownames(geneCounts)[drivergene]  
colnames(dataset) <- zellinesMutations$Genes  
  
cor.mat = cor(dataset[1:50,], method = "spearman")  
cor.dist = as.dist(1 - cor.mat)  
cor.hc = hclust(cor.dist, method = "ward.D2")  
cor.hc = as.dendrogram(cor.hc)  
plot(cor.hc, las = 2, cex.lab = 2, main = "Clustering of the expression values of all cell lines")



rm(drivergene, realcelllinenames, dataset, cor.hc, cor.mat, cor.dist)

## Warning in rm(drivergene, realcelllinenames, dataset, cor.hc, cor.mat,  
## cor.dist): Objekt 'realcelllinenames' nicht gefunden

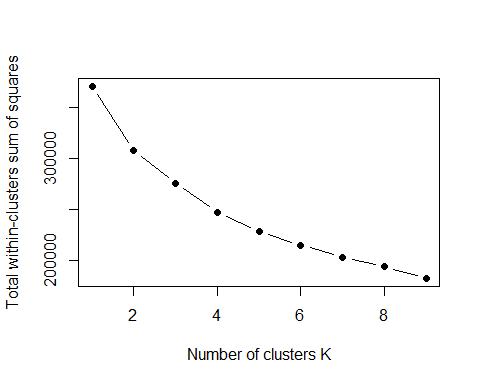
## 3.2 K-means

Performing a k-means.

dataset <- t(processed\_data$expression[-which(rownames(processed\_data$expression) %in% rownames(geneCounts)[1:10]),])   
# determines which dataset we use  
# we are trying to cluster the cell lines with the same drivermutations in the same cluster according to the   
# expression data without the expression of the Drivermutaitons  
# because we want to see what is driving the differences betweent the cell lines except for the Drivermutation expression values  
  
rownames(dataset) <- zellinesMutations$Genes  
  
dataset <- dataset[,-which(apply(dataset, 2, function(x) {  
 var(x)  
}) == 0)]

For choosing the best number centers for the clusters we try the kink method.

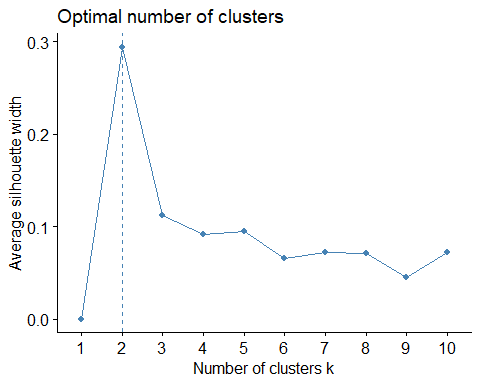
wss = sapply(1:9, function(k) {  
 kmeans(x = dataset, centers = k)$tot.withinss  
})  
plot(1:9, wss, type = "b", pch = 19, xlab = "Number of clusters K", ylab = "Total within-clusters sum of squares")



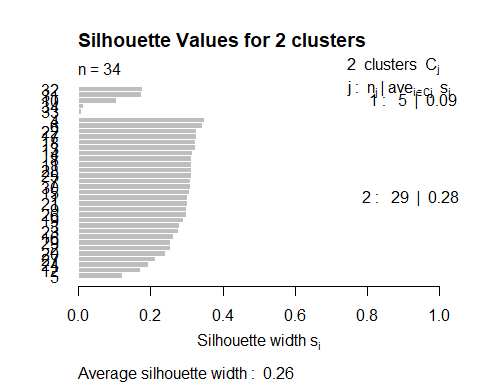
* *But theres no kink in this curve so we need to use other methods to tell us how much centers to choose.*

Now we try the silouette method.

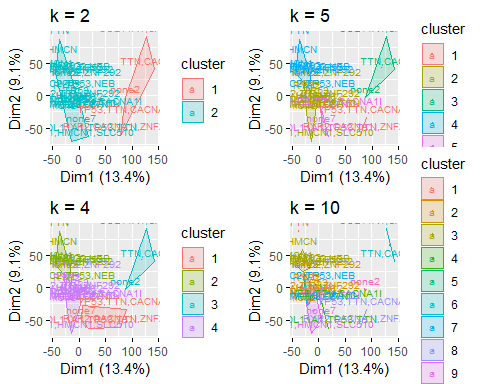
fviz\_nbclust(dataset, kmeans, method = "silhouette")



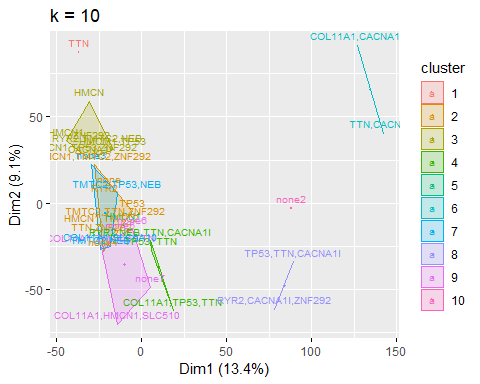
# the clustering with two centers seems to be the best by far according to the   
  
# were taking a look at the silhouette values for the clustering with two centers and   
km = kmeans(x =dataset, centers = 2, nstart = 100)  
plot(silhouette(km$cluster,dist(dataset)), main = "Silhouette Values for 2 clusters")



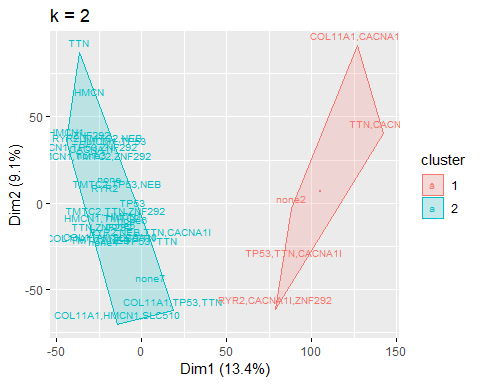
km2 <- kmeans(dataset, centers = 2, nstart = 100)  
km3 <- kmeans(dataset, centers = 5, nstart = 100)  
km4 <- kmeans(dataset, centers = 4, nstart = 100)  
km5 <- kmeans(dataset, centers = 10, nstart = 100)  
  
p1 <- fviz\_cluster(km2,geom = "text", labelsize = 8, data = dataset) + ggtitle("k = 2")  
p2 <- fviz\_cluster(km3, geom = "text", labelsize = 8, data = dataset) + ggtitle("k = 5")  
p3 <- fviz\_cluster(km4, geom = "text", labelsize = 8, data = dataset) + ggtitle("k = 4")  
p4 <- fviz\_cluster(km5, geom = "text", labelsize = 8, data = dataset) + ggtitle("k = 10")  
  
grid.arrange(p1, p2, p3, p4, nrow = 2)



plot(p4) # clearly the clustering with 10 centers does not conclude in clusters with the same Drivermutations



# the reason for that may be that most of our cell lines have more than one Driver mutation  
  
plot(p1)



rm(km,km2,km3,km4,km5,p1,p2,p3,p4, dataset,wss)

* *The clustering with two centers seems to be the best one.*
* *Our next step in the pca will be to see which of the genes drive the differentation of the celllines in this plot because they will be the most variable and thus most interesting ones.*

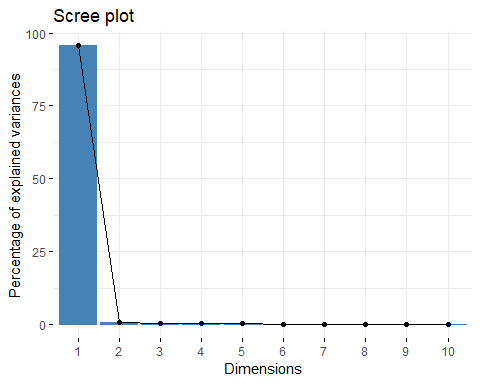
# 3.3 PCA

Now after we saw how the data clustered together we want to see what is driving the differences. We are looking at the first two Principal Components

#drivergene <- 4 # determines which of the Drivermutations will be seen in the cluster at the x axis  
dataset <- processed\_data$expression # determines which dataset we use  
  
  
#colnames(dataset)[which(colnames(dataset) %in% unique(ZelllinesMutations[which(ZelllinesMutations[,1] == topDriverGenes[drivergene]),2]))] <- topDriverGenes[drivergene]  
colnames(dataset)<- zellinesMutations$Genes  
  
pca = prcomp(t(dataset), center = F, scale. = F)  
summary(pca)

## Importance of components:  
## PC1 PC2 PC3 PC4 PC5  
## Standard deviation 495.3869 46.27533 35.11722 26.07913 24.64502  
## Proportion of Variance 0.9573 0.00835 0.00481 0.00265 0.00237  
## Cumulative Proportion 0.9573 0.96567 0.97048 0.97313 0.97550  
## PC6 PC7 PC8 PC9 PC10  
## Standard deviation 21.01126 20.73196 19.77350 19.31929 17.76823  
## Proportion of Variance 0.00172 0.00168 0.00153 0.00146 0.00123  
## Cumulative Proportion 0.97723 0.97890 0.98043 0.98188 0.98312  
## PC11 PC12 PC13 PC14 PC15  
## Standard deviation 17.71421 16.8245 16.18356 15.71160 15.43367  
## Proportion of Variance 0.00122 0.0011 0.00102 0.00096 0.00093  
## Cumulative Proportion 0.98434 0.9854 0.98647 0.98743 0.98836  
## PC16 PC17 PC18 PC19 PC20  
## Standard deviation 15.26658 14.96113 14.46341 13.90066 13.62804  
## Proportion of Variance 0.00091 0.00087 0.00082 0.00075 0.00072  
## Cumulative Proportion 0.98927 0.99014 0.99096 0.99171 0.99243  
## PC21 PC22 PC23 PC24 PC25  
## Standard deviation 13.48726 13.20117 13.08648 12.66417 12.34321  
## Proportion of Variance 0.00071 0.00068 0.00067 0.00063 0.00059  
## Cumulative Proportion 0.99314 0.99382 0.99449 0.99512 0.99571  
## PC26 PC27 PC28 PC29 PC30  
## Standard deviation 11.96792 11.74491 11.66818 11.40272 11.21416  
## Proportion of Variance 0.00056 0.00054 0.00053 0.00051 0.00049  
## Cumulative Proportion 0.99627 0.99681 0.99734 0.99785 0.99834  
## PC31 PC32 PC33 PC34  
## Standard deviation 10.83176 10.65546 10.26134 9.49512  
## Proportion of Variance 0.00046 0.00044 0.00041 0.00035  
## Cumulative Proportion 0.99879 0.99924 0.99965 1.00000

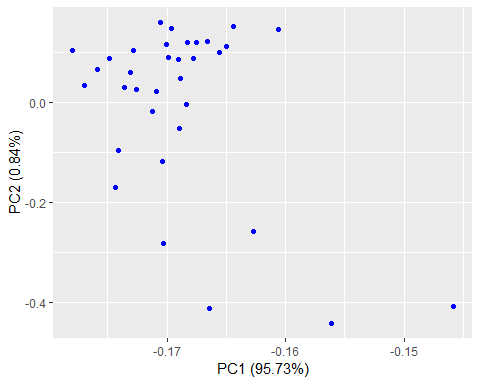
#zum Anzeigen von labels (Zelllinien)  
  
  
fviz\_eig(pca)



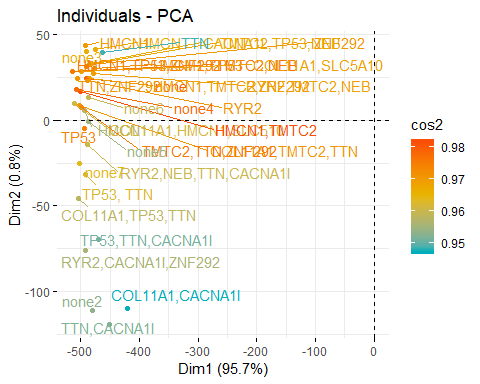
str(pca)

## List of 5  
## $ sdev : num [1:34] 495.4 46.3 35.1 26.1 24.6 ...  
## $ rotation: num [1:16970, 1:34] -9.20e-03 -3.19e-05 -7.79e-03 -1.45e-04 -2.30e-03 ...  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : chr [1:16970] "A1BG" "A1CF" "A2M" "A2ML1" ...  
## .. ..$ : chr [1:34] "PC1" "PC2" "PC3" "PC4" ...  
## $ center : logi FALSE  
## $ scale : logi FALSE  
## $ x : num [1:34, 1:34] -502 -494 -511 -500 -504 ...  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : chr [1:34] "COL11A1,TMTC2,TTN" " HMCN1" "COL11A1,HMCN1,SLC510" "HMCN1,TMTC2" ...  
## .. ..$ : chr [1:34] "PC1" "PC2" "PC3" "PC4" ...  
## - attr(\*, "class")= chr "prcomp"

autoplot(pca, colour = 'blue')



fviz\_pca\_ind(pca,  
 col.ind = "cos2", # Color by the quality of representation  
 gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),  
 repel = TRUE # Avoid text overlapping  
)

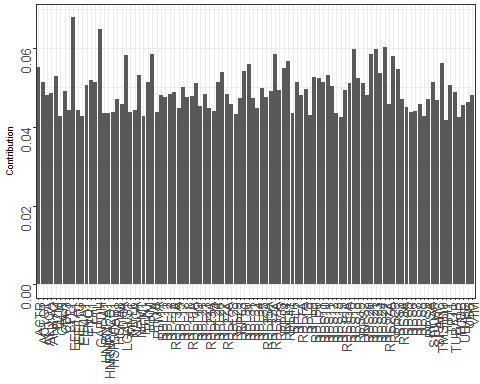


* *Again we see two clusters.*
* *The first principal component contains the most information about the data.*

var\_coord\_func <- function(loadings, comp.sdev){  
 loadings\*comp.sdev  
}  
  
loadings <- pca$rotation  
sdev <- pca$sdev  
var.coord <- t(apply(loadings, 1, var\_coord\_func, sdev))   
  
var.cos2 <- var.coord^2  
comp.cos2 <- apply(var.cos2, 2, sum)  
contrib <- function(var.cos2, comp.cos2){var.cos2\*100/comp.cos2}  
  
var.contrib <- t(apply(var.cos2,1, contrib, comp.cos2))  
head(var.contrib[, 1:4])

## PC1 PC2 PC3 PC4  
## A1BG 8.458140e-03 4.042140e-02 2.820464e-03 1.626858e-03  
## A1CF 1.017925e-07 9.118405e-08 6.046034e-06 1.241073e-05  
## A2M 6.063349e-03 4.405875e-02 8.758703e-02 1.918378e-02  
## A2ML1 2.113426e-06 8.293155e-04 5.108695e-04 6.804066e-04  
## A4GALT 5.277506e-04 5.244157e-02 2.770271e-03 6.278979e-03  
## A4GNT 3.632784e-06 1.020123e-08 5.599425e-05 1.554282e-05

top100var.contrib <- var.contrib[,1]  
top100var.contrib <- as.data.frame(top100var.contrib[order(-top100var.contrib)])  
top100var.contrib$Genes <- rownames(top100var.contrib)  
top100var.contrib <- top100var.contrib[1:100,]  
colnames(top100var.contrib)[1] <- "Contribution"  
  
  
ggplot(data = top100var.contrib) +  
 (geom\_bar(mapping = aes(x = Genes, y = Contribution), stat = "identity")) +  
 theme\_bw(base\_size = 7) + #format the size of the theme nicely  
 theme(legend.position= "none", #define the legend position (here no leghend will be needed)  
 legend.direction="horizontal", #define the legend direction if one is there  
 plot.title = element\_text(hjust = 0.5), #make the title of the plot into the middle  
 axis.text.x = element\_text(angle = 90, vjust = 0.5, hjust=1, size = 10), #define the orientation of the text on the x-axis  
 axis.text.y = element\_text(angle = 90, vjust = 0.5, hjust=1, size = 10), #define the orientation of the text on the x-axis  
 legend.title= element\_blank(), #no title of the legend should be plotted  
 axis.title.x = element\_blank(), #no title of the x-axis is relevant; because that would be samples and that is cleare due to the naming  
 strip.text.y = element\_text(angle = 90)) #define the orientation of the text of the y-axis



* *These are the Components which are contributing the most to our variation in the data. Maybe we will find some of these in our result of the p-test.*

rm(drivergene, realcelllinenames, dataset, loadings, pca, realcelllinenames, var.contrib, var.coord, var.cos2, comp.cos2, sdev)

## Warning in rm(drivergene, realcelllinenames, dataset, loadings, pca,  
## realcelllinenames, : Objekt 'drivergene' nicht gefunden

## Warning in rm(drivergene, realcelllinenames, dataset, loadings, pca,  
## realcelllinenames, : Objekt 'realcelllinenames' nicht gefunden  
  
## Warning in rm(drivergene, realcelllinenames, dataset, loadings, pca,  
## realcelllinenames, : Objekt 'realcelllinenames' nicht gefunden

# 4. Statistical test

We want to perform a p-test and compare the p-values.

driverGenes <- rownames(geneCounts)[1:10] #only use the TOP 10 driver genes  
ttestgenes <- rownames(processed\_data$kd.ceres)  
  
potSecondSites <- lapply(seq\_along(driverGenes), function(a) {  
 genePicker <- driverGenes[a] #pick one driver gene  
 print(paste0("I am doing driver mut: ", a))  
 output <- sapply(seq\_along(rownames(processed\_data$kd.ceres)), function(b) { #the kdCERES matrix is of interest take its' rownames as refrence  
 secondSitePicker <- rownames(processed\_data$kd.ceres)[b] #pick a potetnial 2nd site target  
 if (secondSitePicker != genePicker) {  
 drMUT <- processed\_data$kd.ceres[which(rownames(processed\_data$kd.ceres) == genePicker),] #pick the driver mut data  
 sndMUT <- as.vector(processed\_data$kd.ceres[which(rownames(processed\_data$kd.ceres) == secondSitePicker),]) #pick the 2nd site data  
 cor.val <- cor.test(unlist(drMUT, use.names=FALSE) , unlist(sndMUT, use.names=FALSE), method = "spearman") #make a spearman correlation  
 return(cor.val$p.value) #return the p-values  
 } else {  
 return(1)  
 }  
 })  
 names(output) <- rownames(processed\_data$kd.ceres) #rename all  
 output <- as.data.frame(output) #get a nice dataframe  
 return(output)  
})

## [1] "I am doing driver mut: 1"  
## [1] "I am doing driver mut: 2"  
## [1] "I am doing driver mut: 3"  
## [1] "I am doing driver mut: 4"  
## [1] "I am doing driver mut: 5"  
## [1] "I am doing driver mut: 6"  
## [1] "I am doing driver mut: 7"  
## [1] "I am doing driver mut: 8"  
## [1] "I am doing driver mut: 9"  
## [1] "I am doing driver mut: 10"

names(potSecondSites) <- driverGenes #rename the list of lists  
lapply(potSecondSites, head) #look at the nice data

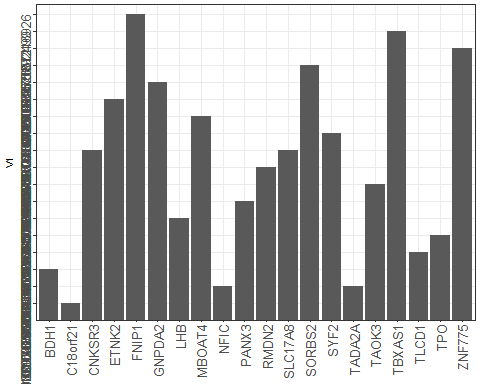
## $TTN  
## output  
## A1BG 0.70023480  
## A1CF 0.39115670  
## A2M 0.34286907  
## A2ML1 0.11397865  
## A4GALT 0.19132453  
## A4GNT 0.01504808  
##   
## $TP53  
## output  
## A1BG 0.28160340  
## A1CF 0.70023480  
## A2M 0.39697321  
## A2ML1 0.64097590  
## A4GALT 0.09015868  
## A4GNT 0.60183071  
##   
## $HMCN1  
## output  
## A1BG 0.4227534  
## A1CF 0.8657359  
## A2M 0.6534159  
## A2ML1 0.7917565  
## A4GALT 0.8725280  
## A4GNT 0.3437615  
##   
## $TMTC2  
## output  
## A1BG 0.45154526  
## A1CF 0.43701759  
## A2M 0.95872743  
## A2ML1 0.75867863  
## A4GALT 0.01716129  
## A4GNT 0.62127398  
##   
## $RYR2  
## output  
## A1BG 0.879329218  
## A1CF 0.669727445  
## A2M 0.002884766  
## A2ML1 0.213302043  
## A4GALT 0.088108676  
## A4GNT 0.304196025  
##   
## $CACNA1I  
## output  
## A1BG 0.93400823  
## A1CF 0.09259686  
## A2M 0.14278128  
## A2ML1 0.61030460  
## A4GALT 0.10401228  
## A4GNT 0.53711418  
##   
## $ZNF292  
## output  
## A1BG 0.27458108  
## A1CF 0.75736391  
## A2M 0.57435603  
## A2ML1 0.07565286  
## A4GALT 0.38922902  
## A4GNT 0.08058396  
##   
## $NEB  
## output  
## A1BG 0.42275339  
## A1CF 0.07595393  
## A2M 0.36190890  
## A2ML1 0.33314716  
## A4GALT 0.04869324  
## A4GNT 0.05084604  
##   
## $COL11A1  
## output  
## A1BG 0.07535272  
## A1CF 0.49777121  
## A2M 0.80910520  
## A2ML1 0.26539892  
## A4GALT 0.18653536  
## A4GNT 0.68237842  
##   
## $SLC5A10  
## output  
## A1BG 0.991742072  
## A1CF 0.467398656  
## A2M 0.834622511  
## A2ML1 0.453641927  
## A4GALT 0.916192275  
## A4GNT 0.007613915

Now that we got all those p-values we want to order the data according to their p-values. So we can see the smallest ones which are the most important ones.

potSecondSites <- lapply(potSecondSites, function(a){  
 a <- as.data.frame(cbind(a$output, rownames(a)))  
 a <- a[order(a[1]), ]  
})

Selecting the 20 Genes out of every DriverGene List with the lowest p score

potSecondSitestop20 <- lapply(seq\_along(potSecondSites), function (a){  
 output <- potSecondSites[[a]][1:20,]  
 return(output)  
})  
names(potSecondSitestop20) <- driverGenes  
  
ggplot(data = potSecondSitestop20$TTN) +  
 (geom\_bar(mapping = aes(x = V2, y = V1), stat = "identity")) +  
 theme\_bw(base\_size = 7) + #format the size of the theme nicely  
 theme(legend.position= "none", #define the legend position (here no leghend will be needed)  
 legend.direction="horizontal", #define the legend direction if one is there  
 plot.title = element\_text(hjust = 0.5), #make the title of the plot into the middle  
 axis.text.x = element\_text(angle = 90, vjust = 0.5, hjust=1, size = 10), #define the orientation of the text on the x-axis  
 axis.text.y = element\_text(angle = 90, vjust = 0.5, hjust=1, size = 10), #define the orientation of the text on the x-axis  
 legend.title= element\_blank(), #no title of the legend should be plotted  
 axis.title.x = element\_blank(), #no title of the x-axis is relevant; because that would be samples and that is cleare due to the naming  
 strip.text.y = element\_text(angle = 90)) #define the orientation of the text of the y-axis



rm(potSecondSites, ttestgenes)

# 5. Multiple linear regression analysis

## 5.1 Predicting the expression of our driver genes with all the data

First creating the dataframe for the multiple linear regression with all the dataframes as columns, the rows are every gene in every cell line

Doing the multiple linear regression. Then comparing the predicted values of our model with the real values of the test\_data by spearman correlaiton We perform this with every driver gene.

a <- generalPlottingData$expression[,1:3]  
a <-a[,c(1,3,2)]  
copynumber <- generalPlottingData$copynumber[,2]  
kd.ceres <- generalPlottingData$kd.ceres[,2]  
kd.prob <- generalPlottingData$kd.prob[,2]  
  
RegData <- cbind(a,copynumber,kd.ceres,kd.prob)  
  
# doing the multiple linear Regression   
# then comparint the predicted values of our model with the real values   
# of the test\_data by spearman correlaiton  
# doing this for every Driver Gene  
  
Regressionanalysis <-lapply(1:10, function(x){  
 RegData <- cbind(a,copynumber,kd.ceres,kd.prob)  
 Driverexpression <- c()  
 for (i in 1:34) {  
 a <- 16970\*i  
 c <- (16970\* (i-1))+1  
 b <- colnames(processed\_data$expression)[i]  
 Driverexpression[c:a] <- processed\_data$expression[rownames(geneCounts)[x],b]  
 }  
 print(paste0("I am doing driver mut: ", rownames(geneCounts)[x]))  
 RegData <- cbind(RegData,Driverexpression)  
 RegData <-as.data.frame(RegData)  
 colnames(RegData) <- as.vector(colnames(RegData))  
 set.seed(123) #initialize the random numbers  
 split = sample.split(RegData, SplitRatio = 0.8) #split the dataset into 4/5 Training and 1/5 Testing dataset  
 training\_set = subset(RegData, split == TRUE) #use the labels to get the training data  
 test\_set = subset(RegData, split == FALSE)   
 rm(RegData)  
 # Fitting Multiple Linear Regression to the Training set  
 regressor = lm(Driverexpression ~ Value + copynumber + kd.ceres + kd.prob , data = training\_set) #predict profit based on ALL (=.) the input variables for one company   
 # Predicting the Test set results  
 y\_pred = predict(regressor, newdata = test\_set, se.fit = TRUE) #predict the expression based on your testing data   
 test\_set$Prediction = y\_pred$fit #add your predictions to the dataset  
 #Now compare the Predictions (last column) with the real values of the startups (2nd last column)  
 Results <- cor.test(test\_set$Driverexpression, test\_set$Prediction, method = "spearman", exact=FALSE)  
 return(Results)  
})

## [1] "I am doing driver mut: TTN"  
## [1] "I am doing driver mut: TP53"  
## [1] "I am doing driver mut: HMCN1"  
## [1] "I am doing driver mut: TMTC2"  
## [1] "I am doing driver mut: RYR2"  
## [1] "I am doing driver mut: CACNA1I"  
## [1] "I am doing driver mut: ZNF292"  
## [1] "I am doing driver mut: NEB"  
## [1] "I am doing driver mut: COL11A1"  
## [1] "I am doing driver mut: SLC5A10"

names(Regressionanalysis) <- rownames(geneCounts)[1:10]  
Regressionanalysis <- as.vector(Regressionanalysis)  
rm(RegData,kd.ceres,kd.prob,copynumber,a)  
  
  
ResultsRegression <- melt(lapply(1:length(Regressionanalysis), function(x){  
 return(Regressionanalysis[[x]][3])  
}))  
ResultsRegression <- cbind(ResultsRegression,melt(lapply(1:length(Regressionanalysis), function(x){  
 return(Regressionanalysis[[x]][1])  
})))  
  
ResultsRegression$L2 <- rownames(geneCounts)[1:10]  
ResultsRegression <- ResultsRegression [,c(2,1,4)]  
colnames(ResultsRegression) <- c("DriverGene", "pvalue", "Svalue" )  
  
print(ResultsRegression)

## DriverGene pvalue Svalue  
## 1 TTN 5.509211e-16 7.317806e+14  
## 2 TP53 4.997652e-09 7.359220e+14  
## 3 HMCN1 5.739113e-37 7.233205e+14  
## 4 TMTC2 1.486016e-36 7.234577e+14  
## 5 RYR2 1.671884e-30 7.255677e+14  
## 6 CACNA1I 7.949257e-20 7.299161e+14  
## 7 ZNF292 9.481412e-12 7.341441e+14  
## 8 NEB 5.286592e-14 7.328378e+14  
## 9 COL11A1 1.548789e-77 7.124150e+14  
## 10 SLC5A10 4.727922e-25 7.276653e+14

# with these low p-values we can say with confidence that our Model is able to reproduce and predict   
# the Expressionvalues of our drivergenes   
  
# using just our top 20 out of the statistical testing we hoped to see that the p values would not increase that much   
# this would verify our these that these genes are the essential components which drive the differnet expression of the Driver Gene   
# as you can see below this ist not the case and the p values are very much increased  
  
Regressionanalysistop20 <-lapply(1:10, function(x){  
 a <- generalPlottingData$expression[which(generalPlottingData$expression[,3] %in% as.character(potSecondSitestop20[[x]][,2])),1:3]  
 a <-a[,c(1,3,2)]  
 copynumber <- generalPlottingData$copynumber[which(generalPlottingData$copynumber[,3] %in% as.character(potSecondSitestop20[[x]][,2])),2]  
 kd.ceres <- generalPlottingData$kd.ceres[which(generalPlottingData$kd.ceres[,3] %in% as.character(potSecondSitestop20[[x]][,2])),2]  
 kd.prob <- generalPlottingData$kd.prob[which(generalPlottingData$kd.prob[,3] %in% as.character(potSecondSitestop20[[x]][,2])),2]  
 RegData <- cbind(a,copynumber,kd.ceres,kd.prob)  
 h <- length(generalPlottingData$expression[which(generalPlottingData$copynumber[,3] %in% as.character(potSecondSitestop20[[x]][,2])),2])  
 Driverexpression <- c()  
 for (i in 1:34) {  
 a <- h\*i  
 c <- (h\* (i-1))+1  
 b <- colnames(processed\_data$expression)[i]  
 Driverexpression[c:a] <- processed\_data$expression[rownames(geneCounts)[x],b]  
 }  
 print(paste0("I am doing driver mut: ", rownames(geneCounts)[x]))  
 RegData <- cbind(RegData,Driverexpression)  
 RegData <-as.data.frame(RegData)  
 colnames(RegData) <- as.vector(colnames(RegData))  
 set.seed(123) #initialize the random numbers  
 split = sample.split(RegData, SplitRatio = 0.8) #split the dataset into 4/5 Training and 1/5 Testing dataset  
 training\_set = subset(RegData, split == TRUE) #use the labels to get the training data  
 test\_set = subset(RegData, split == FALSE)   
 rm(RegData)  
 # Fitting Multiple Linear Regression to the Training set  
 regressor = lm(Driverexpression ~ Value + copynumber + kd.ceres + kd.prob , data = training\_set) #predict profit based on ALL (=.) the input variables for one company   
 # Predicting the Test set results  
 y\_pred = predict(regressor, newdata = test\_set, se.fit = TRUE) #predict the expression based on your testing data   
 test\_set$Prediction = y\_pred$fit #add your predictions to the dataset  
 #Now compare the Predictions (last column) with the real values of the startups (2nd last column)  
 Results <- cor.test(test\_set$Driverexpression, test\_set$Prediction, method = "spearman", exact=FALSE)  
 return(Results)  
})

## [1] "I am doing driver mut: TTN"

## Warning in data.frame(..., check.names = FALSE): row names were found from  
## a short variable and have been discarded

## [1] "I am doing driver mut: TP53"

## Warning in data.frame(..., check.names = FALSE): row names were found from  
## a short variable and have been discarded

## [1] "I am doing driver mut: HMCN1"

## Warning in data.frame(..., check.names = FALSE): row names were found from  
## a short variable and have been discarded

## [1] "I am doing driver mut: TMTC2"

## Warning in data.frame(..., check.names = FALSE): row names were found from  
## a short variable and have been discarded

## [1] "I am doing driver mut: RYR2"

## Warning in data.frame(..., check.names = FALSE): row names were found from  
## a short variable and have been discarded

## [1] "I am doing driver mut: CACNA1I"

## Warning in data.frame(..., check.names = FALSE): row names were found from  
## a short variable and have been discarded

## [1] "I am doing driver mut: ZNF292"

## Warning in data.frame(..., check.names = FALSE): row names were found from  
## a short variable and have been discarded

## [1] "I am doing driver mut: NEB"

## Warning in data.frame(..., check.names = FALSE): row names were found from  
## a short variable and have been discarded

## [1] "I am doing driver mut: COL11A1"

## Warning in data.frame(..., check.names = FALSE): row names were found from  
## a short variable and have been discarded

## [1] "I am doing driver mut: SLC5A10"

## Warning in data.frame(..., check.names = FALSE): row names were found from  
## a short variable and have been discarded

names(Regressionanalysistop20) <- rownames(geneCounts)[1:10]  
Regressionanalysistop20 <- as.vector(Regressionanalysistop20)  
  
ResultsRegressiontop20 <- melt(lapply(1:length(Regressionanalysistop20), function(x){  
 return(Regressionanalysistop20[[x]][3])  
}))  
ResultsRegressiontop20 <- cbind(ResultsRegressiontop20,melt(lapply(1:length(Regressionanalysistop20), function(x){  
 return(Regressionanalysistop20[[x]][1])  
})))  
  
ResultsRegressiontop20$L2 <- rownames(geneCounts)[1:10]  
ResultsRegressiontop20 <- ResultsRegressiontop20 [,c(2,1,4)]  
colnames(ResultsRegressiontop20) <- c("DriverGene", "pvalue", "Svalue" )  
  
print(ResultsRegressiontop20)

## DriverGene pvalue Svalue  
## 1 TTN 0.09746992 49026556057  
## 2 TP53 0.04382470 49238451647  
## 3 HMCN1 0.88944564 48128985972  
## 4 TMTC2 0.12381370 48956640123  
## 5 RYR2 0.10252586 49012065541  
## 6 CACNA1I 0.42022190 48523361801  
## 7 ZNF292 0.07410218 49102647128  
## 8 NEB 0.79788512 48198209845  
## 9 COL11A1 0.80113984 48195718948  
## 10 SLC5A10 0.20397571 48797830898

* + so with this result we can not define confidently the secound targets.\*

# RESULTS

Resultspresentation <- lapply(1:length(potSecondSitestop20), function(x){  
 return(potSecondSitestop20[[x]][2])  
})  
  
names(Resultspresentation) <- rownames(geneCounts)[1:10]  
print(Resultspresentation)

## $TTN  
## V2  
## 1619 C18orf21  
## 9311 NFIC  
## 14283 TADA2A  
## 1308 BDH1  
## 14708 TLCD1  
## 15183 TPO  
## 7750 LHB  
## 10332 PANX3  
## 14321 TAOK3  
## 12208 RMDN2  
## 2967 CNKSR3  
## 13171 SLC17A8  
## 14213 SYF2  
## 8296 MBOAT4  
## 4571 ETNK2  
## 5693 GNPDA2  
## 13698 SORBS2  
## 16853 ZNF775  
## 14435 TBXAS1  
## 5161 FNIP1  
##   
## $TP53  
## V2  
## 2499 CDKN1A  
## 11816 RAD50  
## 14298 TAF4  
## 14764 TMCC1  
## 4371 ELP5  
## 2900 CLNS1A  
## 3317 CSNK1E  
## 15153 TP53BP1  
## 1038 ATE1  
## 5012 FEM1B  
## 10303 PAGR1  
## 12378 RPL23  
## 16904 ZNF862  
## 11548 PSME3  
## 11231 PPP1R42  
## 4149 DYNLT1  
## 11199 PPP1R12A  
## 9363 NIPBL  
## 16084 WDR83  
## 14889 TMEM207  
##   
## $HMCN1  
## V2  
## 3855 DLEC1  
## 7868 LPA  
## 14241 SYT1  
## 2696 CHI3L2  
## 1086 ATP13A4  
## 15195 TPRX1  
## 3970 DNM1  
## 997 ASGR1  
## 5454 GCKR  
## 321 ADO  
## 2893 CLMN  
## 564 AMTN  
## 12134 RHBDD2  
## 14043 STBD1  
## 5219 FPGT  
## 8695 MRGPRX1  
## 289 ADCY2  
## 11622 PTPN13  
## 16185 XPO4  
## 6637 IFNW1  
##   
## $TMTC2  
## V2  
## 6823 INPP1  
## 6415 HPCAL4  
## 6838 INSL4  
## 10807 PIM1  
## 1630 C19orf44  
## 13439 SLC6A5  
## 12357 RPF2  
## 13018 SHBG  
## 14524 TENM4  
## 9418 NME1  
## 12112 RGS13  
## 11239 PPP2R2A  
## 13478 SLCO2A1  
## 12557 RWDD3  
## 4841 FAM78A  
## 15160 TP53RK  
## 7610 LAMC1  
## 14552 TEX33  
## 5893 GRIK4  
## 16316 ZC3H7A  
##   
## $RYR2  
## V2  
## 5170 FOS  
## 10974 PLSCR1  
## 4126 DUSP7  
## 13270 SLC26A8  
## 4693 FAM118B  
## 13662 SNX21  
## 12493 RRP8  
## 6148 HERC4  
## 2600 CEP57L1  
## 15432 TSPO  
## 13161 SLC16A7  
## 6609 IFITM2  
## 10984 PLXNA2  
## 4233 EFCAB14  
## 8288 MBIP  
## 16664 ZNF501  
## 12256 RNF14  
## 13123 SLC10A1  
## 4750 FAM177A1  
## 173 ACSF3  
##   
## $CACNA1I  
## V2  
## 13544 SMDT1  
## 8832 MSX2  
## 7726 LGALS12  
## 9657 NUCB1  
## 7536 KRTAP21-3  
## 4033 DPRX  
## 3564 DAOA  
## 10200 OSBPL3  
## 10618 PENK  
## 4608 EXOC3L4  
## 11025 PNPLA5  
## 12802 SEH1L  
## 760 APOBEC3F  
## 9013 MYO1H  
## 15604 UBA52  
## 12857 SERHL2  
## 1598 C16orf97  
## 3660 DDR1  
## 39 ABCA7  
## 15108 TNRC6B  
##   
## $ZNF292  
## V2  
## 7630 LARP4B  
## 10984 PLXNA2  
## 13213 SLC24A1  
## 13373 SLC39A9  
## 10203 OSBPL7  
## 15231 TRAPPC10  
## 12266 RNF152  
## 1824 C6orf99  
## 5586 GLB1  
## 8021 LSM5  
## 10214 OSMR  
## 14875 TMEM190  
## 15768 UNC93A  
## 10717 PHF5A  
## 3094 COPA  
## 11354 PRKCSH  
## 16138 WNT5A  
## 8275 MAX  
## 2703 CHL1  
## 4649 F2RL2  
##   
## $NEB  
## V2  
## 3318 CSNK1G1  
## 3931 DNAJC10  
## 7082 KCNAB3  
## 12359 RPGRIP1L  
## 2675 CHCHD5  
## 15311 TRIM61  
## 9390 NKX6-1  
## 11828 RAD9B  
## 16674 ZNF514  
## 3429 CWC25  
## 2332 CD22  
## 13376 SLC40A1  
## 8895 MTRNR2L2  
## 4583 EVA1A  
## 5809 GPR183  
## 14900 TMEM219  
## 1213 BAALC  
## 11643 PTPRH  
## 15120 TOM1L2  
## 8621 MOCOS  
##   
## $COL11A1  
## V2  
## 3713 DEFB108B  
## 5233 FRK  
## 10256 OXT  
## 16705 ZNF560  
## 2882 CLIC6  
## 9029 MYOM1  
## 14943 TMEM37  
## 930 ARMS2  
## 11982 RBP1  
## 1922 CACNG7  
## 7372 KLHL36  
## 4932 FBXO11  
## 15437 TSPYL6  
## 1078 ATP10D  
## 8008 LRTOMT  
## 14339 TAS2R13  
## 1898 CACHD1  
## 5332 GABBR2  
## 15616 UBASH3B  
## 11202 PPP1R13B  
##   
## $SLC5A10  
## V2  
## 8177 MAP2K4  
## 13850 SPNS2  
## 16525 ZNF256  
## 6648 IFT52  
## 13815 SPEM1  
## 7249 KIAA1324L  
## 4226 EEF2K  
## 8987 MYL4  
## 10888 PLAC8L1  
## 14539 TEX10  
## 16738 ZNF593  
## 15072 TNFSF12  
## 13018 SHBG  
## 3795 DHRS11  
## 11358 PRKD3  
## 11415 PRPSAP2  
## 10410 PCBP4  
## 13153 SLC16A11  
## 14303 TAF6L  
## 4107 DUSP10

print(top100var.contrib[1:20,2])

## [1] "EEF1A1" "GAPDH" "RPS27" "RPS23" "RPS18" "RPL37A" "PPIA"   
## [8] "RPS21" "LGALS1" "RPS29" "RPL41" "TMSB10" "RPL31" "ACTB"   
## [15] "RPL4" "RPS3" "RPL30" "RPL27" "RPS24" "RPS11"

which(top100var.contrib[1:20,2] %in% as.character(melt(Resultspresentation)[,1]))

## Using V2 as id variables  
## Using V2 as id variables  
## Using V2 as id variables  
## Using V2 as id variables  
## Using V2 as id variables  
## Using V2 as id variables  
## Using V2 as id variables  
## Using V2 as id variables  
## Using V2 as id variables  
## Using V2 as id variables

## integer(0)

* *The 2nd targets from the pca and the regression are not the same like we have wished (due to the data or mistakes in the skript) because the kmeans and the pca doesnt reproduce the same 2nd side targets.*
* *However we present our results: the 20top 2nd site genes from our p-test*
* *The regression model shows a really low p-value at which grounds we conclude that following gens should be taken in account as targets for drug development in skin cancer.*

Predicting the expression of the driver mutations with the expression of the other genes

# we also wanted to do another Regression model on just the expression data with the so that the coeffizients of the   
# analysis would tell us how improtant each gene is for the model, but this regression does not work and we doesnt know why   
# maybe you can take a look and help us   
#dataset <- t(processed\_data$expression)  
  
#dataset <- as.data.frame(dataset)  
  
  
  
#set.seed(123) #initialize the random numbers  
#split = sample.split(dataset, SplitRatio = 0.5) #split the dataset into 4/5 Training and 1/5 Testing dataset  
#training\_set = subset(dataset, split == TRUE) #use the labels to get the training data  
#test\_set = subset(dataset, split == FALSE)   
#rm(dataset)  
  
  
# Fitting Multiple Linear Regression to the Training set  
# regressor = lm(TP53 ~ ., data = training\_set) #predict profit based on ALL (=.) the input variables for one company   
  
# Predicting the Test set results  
# y\_pred = predict(regressor, newdata = test\_set, se.fit = TRUE) #predict the expression based on your testing data   
# test\_set$Prediction = y\_pred$fit #add your predictions to the dataset  
#Now compare the Predictions (last column) with the real values of the startups (2nd last column)  
# Results <- cor.test(test\_set$Driverexpression, test\_set$Prediction, method = "spearman", exact=FALSE)  
  
  
  
  
  
  
  
sessionInfo()

## R version 3.5.1 (2018-07-02)  
## Platform: i386-w64-mingw32/i386 (32-bit)  
## Running under: Windows 10 x64 (build 17763)  
##   
## Matrix products: default  
##   
## locale:  
## [1] LC\_COLLATE=German\_Germany.1252 LC\_CTYPE=German\_Germany.1252   
## [3] LC\_MONETARY=German\_Germany.1252 LC\_NUMERIC=C   
## [5] LC\_TIME=German\_Germany.1252   
##   
## attached base packages:  
## [1] grid stats graphics grDevices utils datasets methods   
## [8] base   
##   
## other attached packages:  
## [1] caTools\_1.17.1.2 scales\_1.0.0 ggfortify\_0.4.7   
## [4] usethis\_1.5.0 devtools\_2.0.2 dendextend\_1.12.0  
## [7] forcats\_0.3.0 stringr\_1.3.1 dplyr\_0.7.7   
## [10] purrr\_0.2.5 readr\_1.1.1 tidyr\_0.8.2   
## [13] tibble\_1.4.2 tidyverse\_1.2.1 caret\_6.0-84   
## [16] lattice\_0.20-35 pheatmap\_1.0.12 rstudioapi\_0.8   
## [19] cluster\_2.0.7-1 data.table\_1.12.2 reshape2\_1.4.3   
## [22] gridExtra\_2.3 factoextra\_1.0.5 relaimpo\_2.2-3   
## [25] mitools\_2.4 survey\_3.36 survival\_2.42-3   
## [28] Matrix\_1.2-14 boot\_1.3-20 MASS\_7.3-50   
## [31] ggplot2\_3.1.0   
##   
## loaded via a namespace (and not attached):  
## [1] stringi\_1.2.4 evaluate\_0.13 memoise\_1.1.0   
## [4] processx\_3.3.1 haven\_1.1.2 callr\_3.2.0   
## [7] bitops\_1.0-6 ps\_1.2.0 cli\_1.1.0   
## [10] prodlim\_2018.04.18 DBI\_1.0.0 desc\_1.2.0   
## [13] bindr\_0.1.1 nlme\_3.1-137 ggrepel\_0.8.1   
## [16] rprojroot\_1.3-2 tools\_3.5.1 magrittr\_1.5   
## [19] Rcpp\_0.12.19 xml2\_1.2.0 pkgload\_1.0.2   
## [22] readxl\_1.1.0 httr\_1.3.1 rmarkdown\_1.12   
## [25] assertthat\_0.2.0 sessioninfo\_1.1.1 R6\_2.3.0   
## [28] fs\_1.2.6 nnet\_7.3-12 timeDate\_3043.102   
## [31] munsell\_0.5.0 cellranger\_1.1.0 digest\_0.6.18   
## [34] codetools\_0.2-15 splines\_3.5.1 generics\_0.0.2   
## [37] colorspace\_1.3-2 stats4\_3.5.1 pkgconfig\_2.0.2   
## [40] pillar\_1.3.0 gower\_0.2.1 bindrcpp\_0.2.2   
## [43] iterators\_1.0.10 plyr\_1.8.4 gtable\_0.2.0   
## [46] xfun\_0.6 tidyselect\_0.2.5 rvest\_0.3.2   
## [49] knitr\_1.22 viridisLite\_0.3.0 pkgbuild\_1.0.3   
## [52] rlang\_0.3.0.1 broom\_0.5.0 glue\_1.3.0   
## [55] backports\_1.1.2 prettyunits\_1.0.2 RColorBrewer\_1.1-2  
## [58] ipred\_0.9-9 lubridate\_1.7.4 modelr\_0.1.2   
## [61] lava\_1.6.5 hms\_0.4.2 recipes\_0.1.5   
## [64] remotes\_2.1.0 labeling\_0.3 class\_7.3-14   
## [67] htmltools\_0.3.6 yaml\_2.2.0 lazyeval\_0.2.1   
## [70] ggpubr\_0.2 ModelMetrics\_1.2.2 crayon\_1.3.4   
## [73] withr\_2.1.2 corpcor\_1.6.9 viridis\_0.5.1   
## [76] jsonlite\_1.5 rpart\_4.1-13 foreach\_1.4.4   
## [79] compiler\_3.5.1

# finally done at least for now :)