Project 3: Ovarian Cancer

## Contents

1. Overview Ovarian Cancer
2. Data Cleanup
3. Driver Mutations
4. co-existing Mutations
5. Synthetic Lethality Interactions
6. Linear Regression
7. Summary of Results
8. Conclusion

## 1. Overview Ovarian Cancer

### 1.1 General facts

Ovarian cancer is the one of the leading cancer death causes in women. One of 71 women will be diagnosed with this disease at some point during her lifetime. When treated in early stages, OC has relatively high survival rates. However, symptomes mostly start to develop when the tumour has already progressed to advanced stages, which is why the overall 5-year survival rate for OC lies under 50%.

Ovarian cancer is a very heterogenous disease and mostly represented by epithelial tumours. There are four main subtypes: serous, endometrioid, mucinous and clear cell. Furthermore, tumours can be classified dependent on their state of differentiation. Low grade (LG) Ovarian cancer is well differentiated and therefore not invasive; high grade (HG) tumours are often aggressive and frequently metastasize.

Standard therapy approaches include surgery and chemotherapy with DNA-damaging agents. Personalized immunotherapeutic approaches often target special features of cancer cell genome and proteome such as over-expressed surface receptors.

### 1.2 Common mutations in OC (literature)

Literature on Ovarian cancer states that the most common mutations are found in the DNA repair-related genes BRCA1 and BRCA2. Abnormalities in other DNA and mismatch repair genes have been found too, as well as TP53, which is also known as “guardian of the genome”. According to Testa et al., TP53 accounts for most mutations found in high grade serous Ovarian cancer. It will be our task to verify this information and to gain insight on potential driver mutations in Ovarian cancer.

## 2. Data Cleanup

At first we download the dataset and load it into R-Studio

allDepMapData <- readRDS("~/GitHub/project-01-group-03/DepMap19Q1\_allData.RDS")

We gave each matrix a new, simple name, so it is easier to work with them.

copynumber = allDepMapData[["copynumber"]]  
mutation = allDepMapData[["mutation"]]  
kd.ceres = allDepMapData[["kd.ceres"]]  
kd.prob = allDepMapData[["kd.prob"]]  
annotation = allDepMapData[["annotation"]]  
expression = allDepMapData[["expression"]]

After that we had to extract the ovarian cancer cell lines, so we only have the cell lines which are important for our project.

annotation = annotation[which(annotation$Primary.Disease == "Ovarian Cancer"), ] # The annotation matrix now only consists of Ovarian Cancer cell lines  
ID = annotation$DepMap\_ID # DepMap\_ID is the column name for the Ovarian Cancer cell lines  
expression = expression[ , which(colnames(expression) %in% ID)] # Extract the ovarian cancer cell lines from expression, copynumber, kd.ceres, kd.prob and mutation  
copynumber = copynumber [ , which(colnames(copynumber) %in% ID)]  
kd.ceres = kd.ceres [ , which(colnames(kd.ceres) %in% ID)]  
kd.prob = kd.prob [ , which(colnames(kd.prob) %in% ID)]  
mutation = mutation [ ID]

For further work with the annotation matrix it makes it a lot easier to rename the rows with the names of the cell lines. So far the rownames seem to be random numbers we can not use. In the column “DEPMap\_ID” we can find the name of the actual cell lines.

rownames(annotation) = annotation$DepMap\_ID

Further we remove columns of the annotation matrix we dont need and unnecessary objects as well.

# removing unecessary columns  
annotation = annotation[, -which(colnames(annotation) %in% c("DepMap\_ID", "Aliases", "Primary.Disease", "Gender", "Source"))]  
  
#removing the main data, because we already extracted the OC Data  
rm(allDepMapData)

Many functions don’t work with NA values. That’s why we check the matrices for NA values

#summary(copynumber) hier m?ssen wirs hinkriegen dass er den output net reinhaut  
  
  
NAV = apply(copynumber, 1, function(x) {sum(is.na(x))})   
copynumber = copynumber[-which(NAV > 0), ]   
  
'NAV = apply(expression, 1, function(x) {sum(is.na(x))})   
expression = expression[-which(NAV > 0), ]   
   
NAV = apply(kd.ceres, 1, function(x) {sum(is.na(x))})   
kd.ceres = kd.ceres[-which(NAV > 0), ]  
  
NAV = apply(kd.prob, 1, function(x) {sum(is.na(x))})   
kd.prob = kd.prob[-which(NAV > 0), ]'

## [1] "NAV = apply(expression, 1, function(x) {sum(is.na(x))}) \nexpression = expression[-which(NAV > 0), ] \n \nNAV = apply(kd.ceres, 1, function(x) {sum(is.na(x))}) \nkd.ceres = kd.ceres[-which(NAV > 0), ]\n\nNAV = apply(kd.prob, 1, function(x) {sum(is.na(x))}) \nkd.prob = kd.prob[-which(NAV > 0), ]"

### Install packages

To perform all functions we have to download a few packages.

install.packages("reshape", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/ninah/Documents/R/win-library/3.6'  
## (as 'lib' is unspecified)

## package 'reshape' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\ninah\AppData\Local\Temp\RtmpKA8fAt\downloaded\_packages

install.packages("ggplot2", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/ninah/Documents/R/win-library/3.6'  
## (as 'lib' is unspecified)

## package 'ggplot2' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\ninah\AppData\Local\Temp\RtmpKA8fAt\downloaded\_packages

install.packages("data.table", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/ninah/Documents/R/win-library/3.6'  
## (as 'lib' is unspecified)

## package 'data.table' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\ninah\AppData\Local\Temp\RtmpKA8fAt\downloaded\_packages

install.packages("gridExtra", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/ninah/Documents/R/win-library/3.6'  
## (as 'lib' is unspecified)

## package 'gridExtra' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\ninah\AppData\Local\Temp\RtmpKA8fAt\downloaded\_packages

After that the packages need to be installed

library(reshape)

## Warning: package 'reshape' was built under R version 3.6.1

library(ggplot2)  
library(data.table)

## Warning: package 'data.table' was built under R version 3.6.1

##   
## Attaching package: 'data.table'

## The following object is masked from 'package:reshape':  
##   
## melt

library(gridExtra)

## Warning: package 'gridExtra' was built under R version 3.6.1

## 3. Driver Mutations

Fuse mutation lists to one matrix

mutation.all = as.data.frame(rbindlist(mutation))

Our analyses just need the information in certain columns of our data frame, for example gene name and location (chromosome), cell line, the kind of mutation (missense, frame shift, etc.). Thus, we extract these columns and put them in the data frame “mutation.all”.

mutation.all = mutation.all[, which(colnames(mutation.all) %in% c("Hugo\_Symbol", "DepMap\_ID", "Variant\_Classification", "Variant\_annotation", "isTCGAhotspot", "Chromosome", "isDeleterious"))]

There are different types of mutations. Some of them are silent, which means that the amino acid sequence is not altered and the protein structure is not affected. Other mutations lead to an amino acid exchange, but do not provoke dramatic conformational changes in the encoded protein. Whether or not a mutation has an impact on protein function is noted in the column “isDeleterious”. We want to extract all mutations that are TRUE for isDeleterious, since these might have something to do with cancer cell development.

mutation.all = mutation.all[which(mutation.all$isDeleterious == "TRUE"), ] # only include rows (=genes) that have deleterious mutations in data frame "mutation.all"

Order matrix alphabetically

mutation.all <- mutation.all[order(mutation.all$Hugo\_Symbol),]

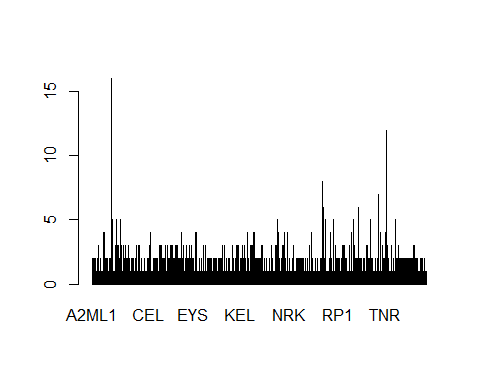
Find most frequently mutated genes by summing up all “mutation events” found in all cell lines. To do this, the names of the genes were converted to factors.

mutation.all$Hugo\_Symbol = factor(mutation.all$Hugo\_Symbol)  
  
summary(mutation.all$Hugo\_Symbol)

## ARID1A TP53 PRSS3 THBS3 PTEN SMARCA4 ARID1B ATM   
## 16 12 8 7 6 6 5 5   
## BAI1 MYH9 PTPRF REV3L SLC34A1 SYNE1 TTN ALPPL2   
## 5 5 5 5 5 5 5 4   
## CEP350 DUSP16 FAM168B FAM193A KMT2B KMT2D LMBRD1 LMTK3   
## 4 4 4 4 4 4 4 4   
## LRBA MYO5C NFASC NOX5 PKHD1L1 RBL2 RFC1 SLC22A6   
## 4 4 4 4 4 4 4 4   
## SLTM TLN2 TNRC6B TNRC6C ADAMTS20 AMER1 ATG7 ATP7B   
## 4 4 4 4 3 3 3 3   
## BCLAF1 BICC1 BRE C17orf80 CACNA1I CBX8 CENPE COL25A1   
## 3 3 3 3 3 3 3 3   
## COL6A6 CSMD2 CSMD3 CTPS2 DBN1 DDX59 DLX6 DNAH7   
## 3 3 3 3 3 3 3 3   
## ECHDC1 EIF5B EP400 ERMAP FLOT2 FOXN2 HERC2 HKDC1   
## 3 3 3 3 3 3 3 3   
## INPPL1 ITGAE JARID2 KANSL1 KIAA0947 KIAA2026 KMT2C LAMA5   
## 3 3 3 3 3 3 3 3   
## LGMN MAP3K2 MAPKBP1 MORC4 MUC16 MYCBP2 NEB NES   
## 3 3 3 3 3 3 3 3   
## NPAT OFD1 PIK3C2G PIK3R1 RETSAT RIMS2 SBNO1 SCN7A   
## 3 3 3 3 3 3 3 3   
## SCN9A SDE2 SHPRH SLC3A2 STAB1 TFPI2 TMEM131 TPTE   
## 3 3 3 3 3 3 3 3   
## TRIO TTK UBR1 (Other)   
## 3 3 3 2734

Plot all mutations to see most frequently mutated genes

plot(mutation.all$Hugo\_Symbol)



The most common mutation is found in ARID1A (16 times). Surprisingly, BRCA1 and BRCA2 are only mutated twice and once, respectively.

Problem: Many mutations occur several times in one cell line, but in different DNA loci. If we want to identify the most frequent mutations among all cell lines, these duplicates should not be included. We want to extract mutations that occur frequently in different cell lines. Therefore, we use the “duplicated” function from the dplyr package and define a new data frame called new\_uniq that only contains one mutation of a gene per cell line for simplicity (since the type of mutation is not relevant here).

duplicates <- which(duplicated(mutation.all[c('Hugo\_Symbol', 'DepMap\_ID')]), ) #find all identical combinations of mutated gene and cell line  
duplicates # show all genes that are duplicated

## [1] 103 105 109 129 176 177 178 180 182 186 188 190 195 227  
## [15] 228 236 286 358 388 415 461 642 704 877 891 895 907 908  
## [29] 967 971 972 980 981 1024 1072 1340 1346 1348 1351 1352 1370 1397  
## [43] 1404 1435 1449 1451 1513 1514 1517 1580 1589 1590 1720 1723 1778 1784  
## [57] 1832 1987 2082 2102 2117 2119 2121 2123 2137 2140 2233 2236 2237 2339  
## [71] 2424 2425 2427 2469 2476 2478 2587 2600 2605 2658 2659 2693 2759 2789  
## [85] 2790 2826 2834 2896 3003 3006 3023

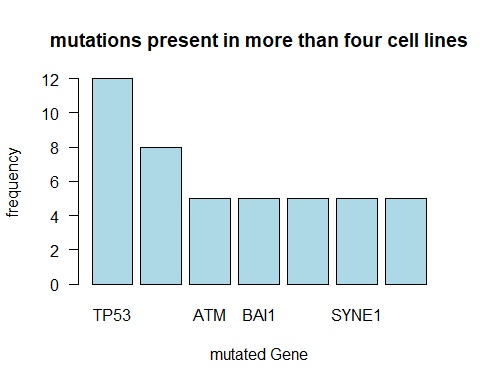
new\_uniq <- mutation.all[!duplicated(mutation.all[c('Hugo\_Symbol', 'DepMap\_ID')]),] #do not include duplicates in data frame new\_uniq  
summary(new\_uniq)

## Hugo\_Symbol Chromosome Variant\_Classification isDeleterious   
## TP53 : 12 Length:3016 Length:3016 Mode:logical   
## ARID1A : 8 Class :character Class :character TRUE:3016   
## ATM : 5 Mode :character Mode :character   
## BAI1 : 5   
## PTPRF : 5   
## SYNE1 : 5   
## (Other):2976   
## isTCGAhotspot Variant\_annotation DepMap\_ID   
## Mode :logical Length:3016 Length:3016   
## FALSE:2841 Class :character Class :character   
## TRUE :175 Mode :character Mode :character   
##   
##   
##   
##

As we can see, ARID1A is only the second most common mutation after TP53, which is mutated in 12 of 34 different cell lines. The most common mutations among our cell lines are thought to be our driver mutations. We will be conducting further analyses with these genes.

For a better overview, we solely plot genes that are mutated more than 4 times in total. Further analyses will focus on these designated driver mutations.

DriverMutation <- as.data.frame(table(new\_uniq$Hugo\_Symbol)) # Create a new data frame that contains the mutated genes (but just once per cell line, as definded in new\_uniq)  
DriverMutation = DriverMutation[which(DriverMutation$Freq > 4), ] # extract all mutations that occur more than 4 times among our cell lines in total  
DriverMutation <- DriverMutation[order(DriverMutation$Freq, decreasing = TRUE),] # order the genes in the data frame according to their frequency  
barplot(DriverMutation$Freq, names.arg = DriverMutation$Var1, xlab="mutated Gene", las = 1, horiz=FALSE, cex.axis = 1,2, col = "lightblue", main = "mutations present in more than four cell lines", ylab = "frequency") # create a barplot of the most frequent mutations, displaying their occurence



## 4. Co-existing Mutations

Here, we want to investigate whether there are certain mutations that often occur together and are therefore possibly linked to each other in their genesis. The goal will be to create a symmetrical matrix whose columns AND rows are our genes; the cells’ content will be the counts of how often two corresponding genes turn up in the same cell line together. Next, maximum values can be extracted and tested for significance.

To get a short overview about how many co existing Drivermutations are common in the data we perform a heatmap:

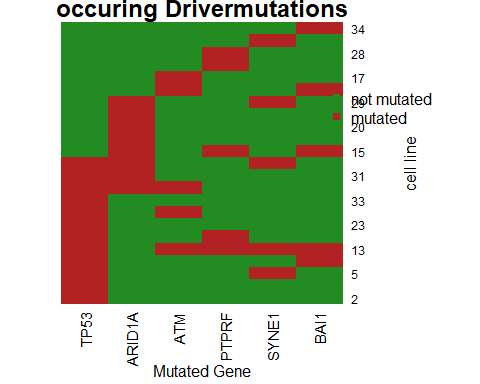
#creating a new matrix containing every cell line that has at least one of our 7 Drivermutations. We check if the name of our Drivermutation occurs in the Drivermutation matrix  
  
dm <- new\_uniq[which(new\_uniq$Hugo\_Symbol %in% DriverMutation$Var1), ]  
  
  
  
#lets check out how many Drivermutations are common in the cell lines  
table(dm$DepMap\_ID)

##   
## ACH-000116 ACH-000123 ACH-000132 ACH-000237 ACH-000278 ACH-000308   
## 1 2 1 2 3 1   
## ACH-000460 ACH-000520 ACH-000527 ACH-000584 ACH-000608 ACH-000646   
## 3 6 4 1 2 1   
## ACH-000657 ACH-000663 ACH-000696 ACH-000713 ACH-000811 ACH-000885   
## 1 1 1 2 3 2   
## ACH-000936 ACH-000947 ACH-001048 ACH-001278 ACH-001374   
## 2 3 1 1 1

#to get acces to the single cell lines we have to create single matrices for every Drivermutation which contain the cell lines with that Mutation  
  
dmTP53 = dm[which(dm$Hugo\_Symbol == "TP53"), ]  
dmARID1A = dm[which(dm$Hugo\_Symbol == "ARID1A"), ]  
dmATM = dm[which(dm$Hugo\_Symbol == "ATM"), ]  
dmBAI1 = dm[which(dm$Hugo\_Symbol == "BAI1"), ]  
dmPTPRF = dm[which(dm$Hugo\_Symbol == "PTPRF"), ]  
dmSYNE1 = dm[which(dm$Hugo\_Symbol == "SYNE1"), ]  
dmTHBS3 = dm[which(dm$Hugo\_Symbol == "THBS3"), ]  
  
  
#now we add new colums to the annotation matrix containing information about which cell line contains which Drivermutation  
annotation$TP53 <- ifelse(rownames(annotation) %in% dmTP53$DepMap\_ID, TRUE, FALSE)  
annotation$ARID1A <- ifelse(rownames(annotation) %in% dmARID1A$DepMap\_ID, TRUE, FALSE)  
annotation$ATM <- ifelse(rownames(annotation) %in% dmATM$DepMap\_ID, TRUE, FALSE)  
annotation$PTPRF <- ifelse(rownames(annotation) %in% dmPTPRF$DepMap\_ID, TRUE, FALSE)  
annotation$SYNE1 <- ifelse(rownames(annotation) %in% dmSYNE1$DepMap\_ID, TRUE, FALSE)  
annotation$BAI1 <- ifelse(rownames(annotation) %in% dmBAI1$DepMap\_ID, TRUE, FALSE)

Now we can see which cell line has which Drivermutation by just looking at the annotation matrix. For the further steps we copy the annotation marix because we will do some larger changes.

#performing a heatmap  
  
#at first we copy the annotation matrix and delete columns which are unesessary for this step  
anno <- annotation[, -which(colnames(annotation) %in% c("CCLE\_Name", "Subtype.Disease"))]  
  
#now we change the boolean variables into integer values: TRUE = 1, FALSE = 0  
anno <- apply(anno, 2, function(x) { ifelse(x == TRUE, 1, 0)})  
  
#for the further steps we have to convert the matrix back to a data frame  
anno <- as.data.frame(anno)  
  
#to get a better overview at the end we replace the cell line names into numbers  
rownames(anno) <- c(1:34)  
  
#now we sum up how many TRUE occur in each cell line  
anno$summe <- apply(anno, 1, function(x) { sum(x)})  
  
#we remove cell lines whitout any of the 7 Drivermutations  
rmv.rows = apply(anno, 1, function(x) {  
 ifelse(anno$summe == 0, 1,0) })  
  
anno = anno[-which(rmv.rows > 0), ]  
  
#the next step is done to order the cell lines after tp53 to have a better view at the end  
anno$order <- ifelse(anno$TP53 == 1 & anno$ARID1A == 0, 5, ifelse(anno$TP53 == 1 & anno$ARID1A == 1, 4, (ifelse(anno$ARID1A == 1, 3,ifelse(anno$ATM == 1, 2, ifelse(anno$PTPRF == 1, 1, 0))))))  
  
anno <- anno[order(anno$order, decreasing = TRUE),]  
  
#we remove columns which should not be plotted  
anno <- anno[, -which(colnames(anno) %in% c("order", "summe"))]  
  
#and bring the matrix in a format the function "heatmap" can plot  
anno <- data.matrix(anno)  
  
#we create a color palette containing the colors we want to use  
cols <- c("forestgreen", "firebrick" )  
mypalette <- colorRampPalette(cols)(2)  
co <- as.data.frame(cols)  
co$kat <- c("not mutated", "mutated")  
  
  
  
#plotting the heatmap  
heatmap(anno, Rowv = NA, Colv = NA, main = "occuring Drivermutations",xlab = "Mutated Gene", ylab = "cell line", col = mypalette, cexCol = 1.1)  
  
legend("topright", pch = 15, col = cols, legend = co$kat, bty = 'n', cex = 1)



The next step is only possible with less than seven Mutations. Because otherwise there would be too many combinations of Drivermutations. In the Tree-Diagramm we performed in step 4 it is visible that many Drivermutations, especially the ones that occur only five times in our cell lines, occur very randomly along the cell lines. Caused by this fact we validated our seven mutations and choosed the ones that were mentioned in an Ovarian cancer context in literature. The following are our chosen ones: TP53, ARID1A, ATM and PTPRF. We continue the PCA analysis:

#to investigate which cell lines contain which drivermutation we create a new data frame containing only the colums about our four Drivermutations  
annodm <- annotation[, -which(colnames(annotation) %in% c("CCLE\_Name", "Subtype.Disease", "BAI1", "SYNE1", "THBS3"))]  
  
  
#to seperate the cell lines in groups with similar Mutations we give every Drivermutation a specific value: If TP53 is True in the cell line it gets a "1000" , if it isnt ther will be a "0"  
annodm$TP53 <- ifelse(rownames(annotation) %in% dmTP53$DepMap\_ID, 1000, 0)  
  
  
#same here, ARID1A gets a "100" for True and a "0" for FALSE  
annodm$ARID1A <- ifelse(rownames(annotation) %in% dmARID1A$DepMap\_ID, 100, 0)  
  
#and the same for the next Mutations  
annodm$ATM <- ifelse(rownames(annotation) %in% dmATM$DepMap\_ID, 10, 0)  
annodm$PTPRF <- ifelse(rownames(annotation) %in% dmPTPRF$DepMap\_ID, 1, 0)  
  
  
#we can now sum up these values in a new column  
annodm$summe <- apply(annodm, 1, function(x) { sum(x)})

We can now use the following order to take a look at our co existing Mutations

View(annodm$summe)

We can now see that many cell lines dont even have one of our chosen mutations because their sum is zero. At many others we can see that they have the same combination of Drivermutations because they have the same value at “annodm$summe”. In the following step we gave the different combinations a specific category

#we picked the most common combinations and gave them different categories by their sums  
  
annodm$kat <- ifelse(annodm$summe == 1000, "TP53 only", ifelse(annodm$summe > 1099, "TP53 & ARID1A", ifelse(annodm$summe == 100, "ARID1A only", ifelse(annodm$summe == 110, "ARID1A & sec. Mu", ifelse(annodm$summe == 101, "ARID1A & sec. Mu", ifelse(annodm$summe < 100, "no TP53 & no ARID1A", ifelse(annodm$summe == 0, "no TP53 & no ARID1A", "TP53 & sec. Mu")))))))  
  
  
  
#we also gave them different colors by using the same function as before  
  
pcb <- ifelse(annodm$summe == 1000, "blue", ifelse(annodm$summe > 1099, "green", ifelse(annodm$summe == 100, "orange", ifelse(annodm$summe == 110, "red", ifelse(annodm$summe == 101, "red", ifelse(annodm$summe < 100, "grey",ifelse(annodm$summe == 0, "grey","darkblue")))))))  
  
  
#and we added another column to the matrix containing that colors  
  
annodm$col <- pcb

## 5. Synthetic Lethality Interactions

After identifying our most common driver mutations, we used this information to find possible synthetic lethality interaction partners First of all, we want to calculate the variance of the cell survival (kd.ceres) and estimated cell survival probability (kd.prob) when knocking out specific genes. We hereby assume that genes that display high variances when knocked out are more likely to have specific synthetic lethality interactions since genes that broadly lead to cell death in most cell lines are probably essential for survival on their own. Therefore, we will conduct the search for synthetic lethality interaction partners with genes whose variances are greater than the 75% quantile.

topVar = apply(kd.prob, 1, var) # calculate variance over all rows (genes)  
summary(topVar)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0000000 0.0002761 0.0015938 0.0080999 0.0078247 0.1797299

kd.prob.topVar = kd.prob[topVar > quantile(topVar, probs = 0.75), ] # new data frame with just the genes whose knockout leads to highly variant cell death events -> perhaps our synthetic lethality interaction partners?  
dim(kd.prob.topVar) # show me how many genes we are looking at now

## [1] 4409 34

#we do the same for the kd.ceres matrix for later analysis  
topVar = apply(kd.ceres, 1, var)  
kd.ceres.topVar = kd.ceres[topVar > quantile(topVar, probs = 0.75), ]  
  
  
mean.survival.prob <- apply(kd.prob.topVar, 1, mean) # calculate the mean cell survival probability of the remaining genes  
summary(mean.survival.prob) # show me the mean survival distribution of the genes whose knockout leads to the most variant cell survival

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.02145 0.14495 0.23453 0.32902 0.47202 0.97181

#plot(density(as.matrix(mean.survival.prob))) # plot me the distribution of cell survival probability of most variant survivals  
  
#wei? noch nicht, wie sinnvoll das alles ist :D mal sehen  
# ja lass des mal drin ich hab des f?r den n?chsten step der PCA benutzt... also die kd.prob.topVar mfG Marco

Alternative Idea: Wir nehmen unsere 7 Driver mutations (TP53, ARID1A, ATM, BAI1, PTPRF, SYNE1, THBS3) und teilen die Matrix kd.prob in zwei Submatrizen: eine, in der alle Zelllinien enthalten sind, die Driver mutations besitzen (23 von 34); und eine, wo die restlichen Zelllinien drin sind (11 von 34). Als n?chstes werden ?hnliche “Verhaltensmuster” unter den DM-Zelllinien gesucht: Erst werden alle Gene aussortiert, die eine hohe Varianz (oder hohe Differenz zw. Max und Min-Wert) haben (schlie?lich suchen wir SL partner, die m?glichst oft zum Tod f?hren), und dann jene, wo das mittlere ?berleben hoch ist => Man hat nur noch Gene, deren Knockout m?glichst oft t?dlich ist.

Jedoch muss gew?hrleistet sein, dass diese potenziellen SL partner keine allgemein essentiellen Gene sind, die auch ohne die Existenz von Driver mutations zum Zelltod f?hren. Daher behandelt man die Matrix mit den non-Driver-mutations-Zelllinien genauso, sodass man am Ende auch hier die Gene beh?lt, deren Knockout durchg?ngig t?dlich ist. Nun kann man beide Data frames vergleichen - jene Gene, die in beiden Matrizen zum Tod f?hren, k?nnen keine spezifischen SL partner f?r die Driver mutations sein. Jedoch besteht die Chance, dass der Knockout jener Gene, die nur in der DM Matrix auftauchen, auch nur dort (und wegen der DM) t?dlich wirkt. Das w?ren dann die SL partner. Genauere Analysen k?nnte man noch mit Subsetten durchf?hren, indem man aufteilt in “Zelllinien mit TP53-Mutation” (ersetze jeweils mit einer anderen DM) und “Zelllinien ohne TP53-Mutation”.

nonDM <- kd.prob[, -which(colnames(kd.prob) %in% dm$DepMap\_ID)] # prob Matrix with cell lines that do not have at least one of the 7 driver mutations  
yesDM <- kd.prob[, which(colnames(kd.prob) %in% dm$DepMap\_ID)] # prob matrix with cell lines that do have driver mutations  
  
# Steps:  
# 1. clean out high variance or those genes that have high difference between min and max survival  
  
Var.nonDM = apply(nonDM, 1, var) # calculate variance over all rows (genes)  
low.nonDM = nonDM[Var.nonDM < quantile(Var.nonDM, probs = 0.5), ] # low variance in behaviour (live or die) upon knockout  
  
Var.yesDM = apply(yesDM, 1, var) # calculate variance over all rows (genes) for the knockout matrix with dm containing cell lines  
low.yesDM = yesDM[Var.yesDM < quantile(Var.yesDM, probs = 0.5), ] # low variance in behaviour (live or die) upon knockout  
  
# 2. Clean out genes with high survival rates (we want lethal knockouts)  
  
Mean = rowMeans(low.nonDM) # calculate mean survival probability (high values: essential gene, knowckout is deadly)  
low.nonDM <- low.nonDM[which(Mean > 0.5), ] # exclude genes that lead to survival upon knockout  
  
Mean2 = rowMeans(low.yesDM) # calculate mean survival probability (high values: essential gene, knowckout is deadly)  
low.yesDM <- low.yesDM[which(Mean2 > 0.5), ] # exclude genes that lead to survival upon knockout  
  
# 3. compare both matrixes: genes that only lead to death in yesDM but not in nonDM could possibly be our synthetic lethality partners  
  
# Only genes, who are present in all dataframes, can be included for the linear regression model. There for we first established a list, which contains all three dataframes. This step allows to extract the shared genes in both dataframes in a more effectiv way.  
  
common.genes <- list(nonDM, yesDM ) # bind dataframes in one list  
names(common.genes) <- c("nonDM","yesDM" ) # rename the elements of the list  
  
# Next we will only extraxt genes, that are common in all three dataframes. Therefore all identical rownames/ genes were collated in one object/value  
  
common.names = Reduce(intersect, lapply(common.genes, row.names)) # all common rownames in the three dataframes  
  
'%ni%' <- Negate('%in%')  
not.common.genes <- lapply(common.genes, function(x) {x[row.names(x) %ni% common.names,]}) # new list reduced to only the unshared genes/ rownames  
  
dim(not.common.genes$nonDM) # find out if there are genes that don't belong to the respective other list

## [1] 0 11

rm(nonDM, yesDM, Var.nonDM, Var.yesDM, low.nonDM, low.yesDM, Mean, Mean2, common.genes, common.names, not.common.genes)

Sadly, there do not seem to be any genes among our data that lead to more cell death in cells with designated driver mutations. However, it remains to be checked whether the experimentally found synthetic lethality interaction partner from our literature for ARID1A - BRD2 - can be affirmed here. We will therefore divide all cell lines into the two groups “yesARID1A” and “nonARID1A”.

dmARID1A = dm[which(dm$Hugo\_Symbol == "ARID1A"), ]  
nonARID1A <- kd.prob[, -which(colnames(kd.prob) %in% dmARID1A$DepMap\_ID)] # prob Matrix with cell lines that do not have the ARID1A mutation  
yesARID1A <- kd.prob[, which(colnames(kd.prob) %in% dmARID1A$DepMap\_ID)] # prob matrix with cell lines that do have driver mutations  
  
# Steps:  
# 1. clean out high variance or those genes that have high difference between min and max survival  
  
Var.nonARID1A = apply(nonARID1A, 1, var) # calculate variance over all rows (genes)  
low.nonARID1A = nonARID1A[Var.nonARID1A < quantile(Var.nonARID1A, probs = 0.3), ] # low variance in behaviour (live or die) upon knockout  
  
Var.yesARID1A = apply(yesARID1A, 1, var) # calculate variance over all rows (genes) for the knockout matrix with dm containing cell lines  
low.yesARID1A = yesARID1A[Var.yesARID1A < quantile(Var.yesARID1A, probs = 0.3), ] # low variance in behaviour (live or die) upon knockout  
  
# 2. Clean out genes with high survival rates (we want lethal knockouts)  
  
Mean = rowMeans(low.nonARID1A) # calculate mean survival probability (high values: essential gene, knowckout is deadly)  
low.nonARID1A <- low.nonARID1A[which(Mean > 0.7), ] # exclude genes that lead to survival upon knockout  
  
Mean2 = rowMeans(low.yesARID1A) # calculate mean survival probability (high values: essential gene, knowckout is deadly)  
low.yesARID1A <- low.yesARID1A[which(Mean2 > 0.7), ] # exclude genes that lead to survival upon knockout  
  
# 3. compare both matrixes: genes that only lead to death in yesDM but not in nonDM could possibly be our synthetic lethality partners  
  
# Only genes, who are present in all dataframes, can be included for the linear regression model. There for we first established a list, which contains all three dataframes. This step allows to extract the shared genes in both dataframes in a more effectiv way.  
  
common.genes <- list(nonARID1A, yesARID1A ) # bind dataframes in one list  
names(common.genes) <- c("nonARID1A","yesARID1A" ) # rename the elements of the list  
  
# Next we will only extraxt genes, that are common in all three dataframes. Therefore all identical rownames/ genes were collated in one object/value  
  
common.names = Reduce(intersect, lapply(common.genes, row.names)) # all common rownames in the three dataframes  
  
'%ni%' <- Negate('%in%')  
not.common.genes <- lapply(common.genes, function(x) {x[row.names(x) %ni% common.names,]}) # new list reduced to only the unshared genes/ rownames  
  
dim(not.common.genes$nonARID1A) # find out if there are genes that don't belong to the respective other list

## [1] 0 26

rm(Var.nonARID1A, Var.yesARID1A, low.nonARID1A, low.yesARID1A, Mean, Mean2, common.genes, common.names, not.common.genes)

Since BRD2 cannot be identified as an SL partner this way, it remains to be found out which difference it makes for cancer cells to lose BRD2 expression when ARID1A is defect.

no <- nonARID1A[which(row.names(nonARID1A) == "BRD2"), ]  
apply(no, 1, mean)

## BRD2   
## 0.521879

yes <- yesARID1A[which(row.names(yesARID1A) == "BRD2"), ]  
apply(yes, 1, mean)

## BRD2   
## 0.6395188

rm(no, yes)

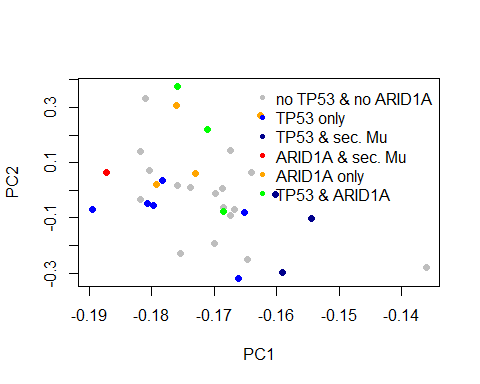
The output says that BRD2 is about 10% more essential in ARID1A mutated cells… Perhaps this is due to the fact that in the literature, BRD2 was mentioned as a SL partner explicitly in clear cell carcinomas. Our data comprise of different types of Ovarian cancer.

Other idea to find SL partners: calculate mean gene essentiality probability for both initially defined groups with/without driver mutations and then ask for significant differences (with statistical test!).

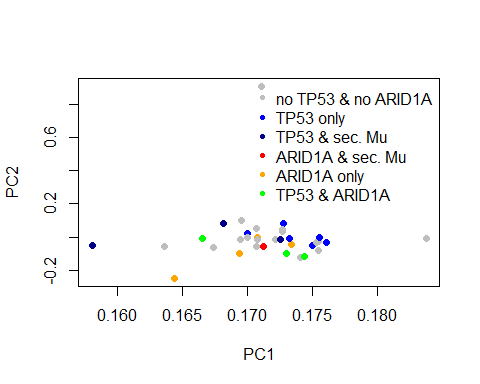
Performing Principal Component Analysis

To reduce the information of the kd.prob matrix we perform a PCA. The Analysis creates groups of cell lines with similar lethality partners. We use the variable TopVar created before.

#we use the colors defined in step 4 for our co existing mutations  
pcb <- cbind(`TP` = pcb)  
  
#keine Ahnung warum ich das machen muss ich frag david noch... aber ohne gehts net  
  
  
#this function calculates the PCA with the variance calculated before  
pca.prob = prcomp(kd.prob.topVar, center = F, scale. = F)  
  
#we plot the pca with the different colors for cell lines with similar Driver Mutations  
plot(pca.prob$rotation[, 1], pca.prob$rotation[, 2], col = pcb[, 1], pch = 19, xlab = "PC1", ylab = "PC2")  
  
#we also add a legend to the plot  
#to add the right colors to that legend we create a new dataframe containing the colors and the categories defined in step 4:  
  
colors <- unique(annodm$col)  
cl <- c(colors)  
ca <- unique(annodm$kat)  
  
legend("topright", pch = 20, col= cl, legend = ca, bty='n', cex=1)



#to have a comparisson we perform another PCA for the kd.ceres matrix  
#therefore we use the kd.ceres.topvar value generated before  
  
pca.ceres = prcomp(kd.ceres.topVar, center = F, scale. = F)  
  
plot(pca.ceres$rotation[, 1], pca.ceres$rotation[, 2], col = pcb[, 1], pch = 19, xlab = "PC1", ylab = "PC2")  
  
legend("topright", pch = 20, col= cl, legend = ca, bty='n', cex=1)



## 6. Linear Regression

In this step we wanted to investigate if it is possible to estimate the gene expression of a gene based on the copynumber. First a linear regression model should show if this issue can be applied to the totality of our data set independent of the differences between the single genes. Another regression model will focus on the relationship between the gene expression and copy number for the driver mutation individually.

### 6.1 Regression model for all genes

#### 6.1.1 Structuring the data

In the first step the data frames copynumber, knock.down and expression were ordered alphabetically, this helps to gain a better overview of the reviewed genes.

copynumber <- copynumber[order(rownames(copynumber)),]  
expression <- expression[order(rownames(expression)),]  
kd.ceres <- kd.ceres[order(rownames(kd.ceres)),]  
kd.prob <- kd.prob[order(rownames(kd.prob)),]

Only genes, who are present in all dataframes, can be included for the linear regression model. There for we first established a list, which contains the two dataframes. This step allows to extract the shared genes in both dataframes in a more effectiv way.

list\_all.genes <- list(expression,copynumber) # bind all the dataframes in one list  
names(list\_all.genes) <- c("expression","copynumber") # rename the elements of the list

Next we will only extraxt genes, that are common in both dataframes. Therefore all identical rownames/ genes were collated in one object/value

common\_names = Reduce(intersect, lapply(list\_all.genes, row.names)) # all common rownames in the three dataframes  
list\_all.genes <- lapply(list\_all.genes, function(x) {x[row.names(x) %in% common\_names,]}) # new list reduced to only the shared genes/ rownames

#### 6.1.2 Checking distribution

As a requirement for the regression model the data sets have to show a normal distribution. For an visual assessment a Q-Q-plot was used for each of the two variables. First all the expression values and copynumber vales of the different cell lines are fused to one long column. The new matrices exp1 and only contains only two columns. The first column shows the original cell line of the listed expression value in the second column.

exp1 <- melt.data.frame(list\_all.genes$expression, variable\_name = "cell line") #fuse all expression values to one long column

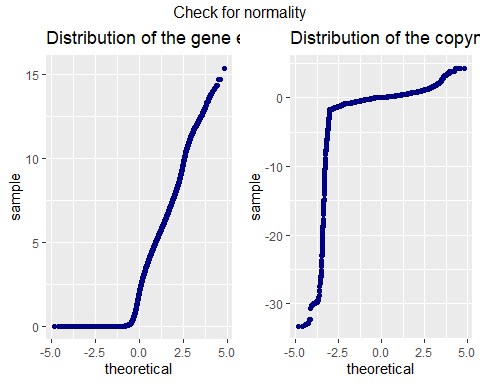
## Using as id variables

cn1 <- melt.data.frame(list\_all.genes$copynumber, variable\_name = "cellline") # fuse all cell lines

## Using as id variables

The new matices will be used to check the distribution of the gene expression and the copynumber. The q-q-plot compairs the quantils of an normal distribution with the distribution of our data set and therefor can be used to check normality of the expression data

plotex <- ggplot() + geom\_qq(aes(sample = exp1$value), color= "navy blue") + ggtitle("Distribution of the gene expression") # plot the quantils of the expression with the quantils of a gaussian distribution  
plotcn <- ggplot() + geom\_qq(aes(sample = cn1$value), color= "navy blue") + ggtitle("Distribution of the copynumber") # plot the quantils of the copynumber with the quantils of a gaussian distribution  
grid.arrange(plotex, plotcn, ncol=2, top = "Check for normality")



The left plot shows a high amount of expression values equal zero, which has a huge deviation of the ideal gaussian distribution for the expression data. In comparison to the perfect distribution the rest of the data points show a good fit for the regression model. To achieve a better fit the genes that contain expression values equal zero will be removed in the next step. The plot indicates a high deviation for vales approximately smaller than -1, to reduce the influence of this circumstance for the normality, all rows/ genes that show a value smaller than -2 or equal -2. There for we first created a vector with all genes, those contain one value smaller or equal -2

list\_all.genes$expression <- list\_all.genes$expression[apply(list\_all.genes$expression !=0, 1, all),] # removes all rows that contain a value = 0

rmv.rows = apply(list\_all.genes$copynumber, 1, function(x) {sum(x <= -2)}) # find all genes with values <= -2  
list\_all.genes$copynumber <- list\_all.genes$copynumber[-which(rmv.rows > 0), ] # remove all rows, those contain more than one value <= -2

Before checking the distribution once again the common genes of both dataframes have to be identified for a second time

common\_names = Reduce(intersect, lapply(list\_all.genes, row.names)) # identify shared rownames  
list\_all.genes <- lapply(list\_all.genes, function(x) {x[row.names(x) %in% common\_names,]}) # only keep the common rownames in the new list

With the melt function the reduced data set is fused to one column of values agian

exp1 <- melt.data.frame(list\_all.genes$expression, variable\_name = "cell line") #fuse all expression values to one long column

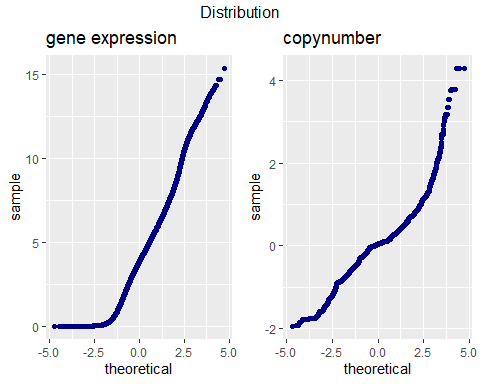
## Using as id variables

cn1 <- melt.data.frame(list\_all.genes$copynumber, variable\_name = "cellline") # fuse all cell lines

## Using as id variables

A q-q-plot is peformed to check the normality of the expression data set once again and determine if the reduction process shows a improved result

plotex <- ggplot() + geom\_qq(aes(sample = exp1$value), color= "navy blue") + ggtitle("gene expression") # plot the quantils of the expression with the quantils of a gaussian distribution  
plotcn <- ggplot() + geom\_qq(aes(sample = cn1$value), color= "navy blue") + ggtitle("copynumber") # plot the quantils of the copynumber with the quantils of a gaussian distribution  
grid.arrange(plotex, plotcn, ncol=2, top = "Distribution") #arrange the to plots next to each other

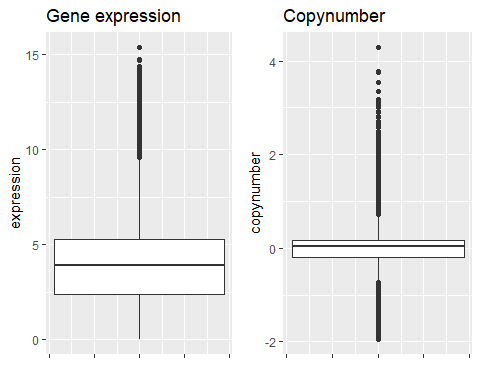


While the left plot for the gene expression indicates a higher accord with the peferct gaussian distribution, there is still a horizontal line pointing out a high amount of values near zero. The right qq-plot reveals a better fit of the copynumber data set after the reduction process, although for copynumbers higher than one a strong aberration is revealed.

#### 6.1.3 Checking for outliners

To check for outliners ab boxplot is apllied for the expression data. Because outliners can have a major impact on the validity of the regression model, both columns are checked with a boxplot.

boxplotex <- qplot(y=exp1$value, x= 1, geom = "boxplot") + ggtitle("Gene expression") + xlab("") + ylab("expression") + theme(axis.text.x=element\_blank()) # remove the scaling of the x-axis  
boxplotcn <- qplot(y=cn1$value, x= 1, geom = "boxplot")+ ggtitle("Copynumber") + xlab("") + ylab("copynumber") + theme(axis.text.x=element\_blank()) # remove the scaling of the x-axis  
grid.arrange(boxplotex, boxplotcn, ncol=2)



The graph indicates a high amount of outliners, this may lead to an inefficient model and therefore has two be considered in the testing process of the regression model. Even both dataframes do not indicate a good fit for normality and show a high amount of outliners, futher analysis will be performed with the data set.

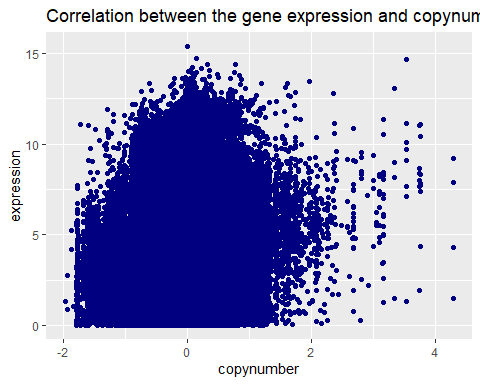
#### 6.1.4 Correlation between the two variables

In this step the correlation between the gene expression and copy number is tested, to decide wether a linear regression model can be applied for the problem. First the gene expression values and copynumber values are fused to one dataframe

regression.all.genes <- as.data.frame(cbind(exp1$value, cn1$value)) # merge the colums  
colnames(regression.all.genes) <- c("expression", "copynumber") # rename the colums

To obtain a visual overview the correlation between the two variables is illustrated in a scatter plot.

ggplot(regression.all.genes, aes(x = copynumber, y = expression)) + geom\_point(color = "navy blue")+ labs(title = "Correlation between the gene expression and copynumber") # ingtegrate title



To analyse the correlation between the gene expression and the copynumber futhermore, the person and spearman correlation coefficient is computed.

cor(regression.all.genes$expression, regression.all.genes$copynumber) # compute person correlation

## [1] 0.1396735

cor(regression.all.genes$expression, regression.all.genes$copynumber, method = "spearman") # compute spearman correlation coefficient

## [1] 0.1329078

Both correlation coefficients show a low correlation between the two varibles of approximately 0.07 to 0.08. The sligthly higher spearman correlation concludes, that the person correlation is lightly influenced by outliners. To verify the correlation a t-test for both correlation coefficient is performed. The null hypothesis indicates that there is no correlation between the copy number and the gene expression, while the alternativ hypothesis confirms a correlation between the two variables. As a common level of significance ?? = 0,05 is choosen and a p-value, which is smaller than the level of significance, will lead to a denial of the null hypothesis. First the p-value for the person correlation coefficient is verified

cor.test(regression.all.genes$expression, regression.all.genes$copynumber)

##   
## Pearson's product-moment correlation  
##   
## data: regression.all.genes$expression and regression.all.genes$copynumber  
## t = 90.024, df = 407318, p-value < 2.2e-16  
## alternative hypothesis: true correlation is not equal to 0  
## 95 percent confidence interval:  
## 0.1366611 0.1426833  
## sample estimates:  
## cor   
## 0.1396735

With a p-value < 2.2e-16 the null hypothesis can be rejected, becaue the p-value shows a way smaller value that the level of significance. Only in the case of a p-value that is higher than the level of significance the null hypothesis would be retained. The same analysis is performed for the spearman correlation coefficient

cor.test(regression.all.genes$expression, regression.all.genes$copynumber, method = "spearman")

## Warning in cor.test.default(regression.all.genes$expression,  
## regression.all.genes$copynumber, : Cannot compute exact p-value with ties

##   
## Spearman's rank correlation rho  
##   
## data: regression.all.genes$expression and regression.all.genes$copynumber  
## S = 9.7661e+15, p-value < 2.2e-16  
## alternative hypothesis: true rho is not equal to 0  
## sample estimates:  
## rho   
## 0.1329078

The test also shows a p-value < 2.2e-16, concluding the rejection of the null hypothesis. Both test lead to the conclusion, that the variables expression and copynumber have a low correlation and therefore could possibly be applied for a linear regression model.

To ensure a clean workspace the no longer needed data frames exp1 and cn1 are removed

remove(exp1, cn1, commonnames, rmv.rows) # noch die ganzen plots

## Warning in remove(exp1, cn1, commonnames, rmv.rows): Objekt 'commonnames'  
## nicht gefunden

#### 6.1.4 Univariate linear regression

Before using the regression.all.genes data for a regression model, a small amount of values have to be seperated for training purposes. 200 rows are taken randomly of the regression.all.genes matrix and will later be used as trainings data for the model.

testing.all.genes <- regression.all.genes[sample(1:nrow(regression.all.genes), 200),] # new dataframe with 200 rows of dataframe regression.all.genes

Next the 200 rows are removed from the original to exclude them from the regression model.

regression.all.genes <- regression.all.genes[ -sample(1:nrow(regression.all.genes), 200),] # select 200 rows randomly

After the reduction process the data is integrated in the regression model lrm.all.genes. The variable copynumber is used as a predictor for the variable gene expression.

lrm.all.genes <- lm(expression ~ copynumber, data = regression.all.genes) #create linear regression model  
summary(lrm.all.genes) # show regression model

##   
## Call:  
## lm(formula = expression ~ copynumber, data = regression.all.genes)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -5.7844 -1.4792 0.0177 1.3572 11.4916   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 3.874800 0.003365 1151.57 <2e-16 \*\*\*  
## copynumber 0.780659 0.008675 89.99 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 2.147 on 407118 degrees of freedom  
## Multiple R-squared: 0.0195, Adjusted R-squared: 0.0195   
## F-statistic: 8098 on 1 and 407118 DF, p-value: < 2.2e-16

With the calculated intercept of 3.874921 and a slope of 0,780919 the equation for the regression is:

gene expression = 3,874921 + 0,780919\*copynumber

With a p-value smaller that 2e-16 the t-test indicates that the slope of the regression line has a significant aberration from zero. Even with a level of significance of 0,01 (one per mill) the regression line shows a aberration, which leads to the denial of the null-hypothesis.

The standard error for the residuals indicates an average aberration of 2,147 between the predicted expression value and the expression value of the dataset. For a good fit of the model a error equal to zero is preferred.

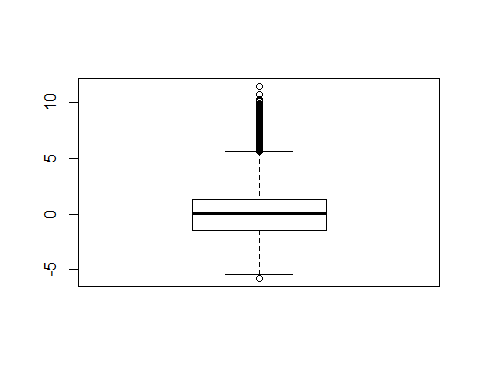
The R-squared statistic shows the proportion of variance and measures  
the linear relationship between the predictor variable (copynumber) and the response variable (expression). While a value near 1 would indicated a good fit of the model, the regression model for all genes has a value of 0.01951 –> near 0 represents a regression that does not explain the variance in the response variable well. (Adjusted R-squared: 0.01951 –> nur für multiple regression relevant)

For a good fit the residuals have to be distributed symmetric around a median of approximately 0. With a calculated value of 0.0177 the median is just above zero. Futhermore the first and third quantil indicate a similar value with reverse algebraic sign. The interquantile range is 2,8359

IQR = Q3-Q1 = 1,3571-(-1,4788) = 2,8359

Both extrema show a higher value than the IQR (min = 2IQR and max = 4IQR). This circumstances may indicate outlines wich is checkt with an boxplot

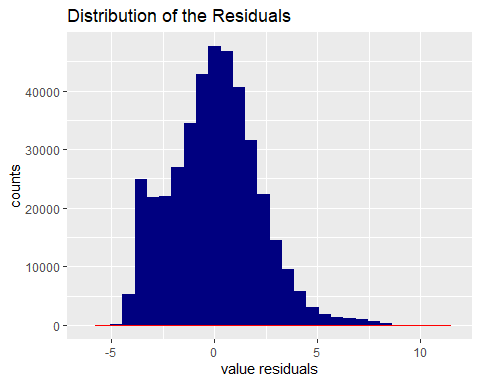
boxplot(resid(lrm.all.genes))



While only value beneath (name suchen) is shown, over the (name?) a high number of outliners is indicated. This outliners will probably lead to a higher varation between the training values and the estimated values in a later step. Another requirement for the residuals is normality, which is checked with a histogramm

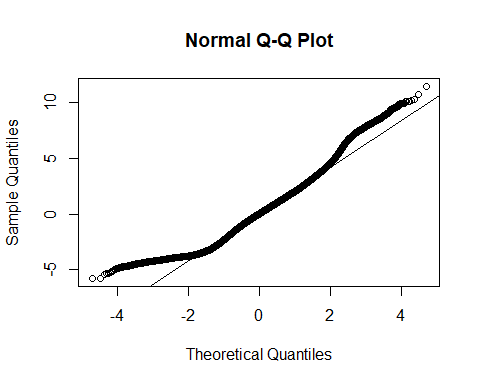
qplot(lrm.all.genes$residuals, geom = "histogram", xlab = "value residuals", ylab = "counts", main = "Distribution of the Residuals", fill = I("navy blue"), bins = 30) + geom\_histogram(aes(y = ..density..)) + stat\_function(fun = dnorm, args = list(mean = mean(lrm.all.genes$residuals), sd = sd(lrm.all.genes$residuals)), color = "red") # warum keine Kurve für Normalverteilung

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



The histogram of the residuals show a good fit for normality, futhermore the distribution is checked with a quantil-quantil-plot.

qqnorm(lrm.all.genes$residuals)  
qqline(lrm.all.genes$residuals)# noch mit ggplot machen

 Next check the correlation between the predictor varibale (copynumber) and the residuals

cor(regression.all.genes$copynumber ,lrm.all.genes$residuals)

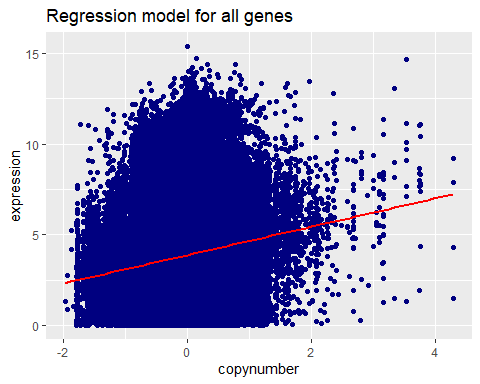
## [1] 4.674732e-16

cor.test(regression.all.genes$copynumber, lrm.all.genes$residuals)

##   
## Pearson's product-moment correlation  
##   
## data: regression.all.genes$copynumber and lrm.all.genes$residuals  
## t = 2.9828e-13, df = 407118, p-value = 1  
## alternative hypothesis: true correlation is not equal to 0  
## 95 percent confidence interval:  
## -0.003071759 0.003071759  
## sample estimates:  
## cor   
## 4.674732e-16

The correlation between the two variables is equal to zero and therefore indicate no correlation between the copynumber and the residuals. The t-Test with a p-value of 1 leads to the acceptance of the null-hypothesis. After all requierments are checked and the regression model is established a scatter plot with the regression line to gain a visual overview

ggplot(regression.all.genes, aes(x=copynumber, y=expression)) + geom\_point(color = "navy blue")+ geom\_smooth(method=lm, se=FALSE, color = "red") + ggtitle("Regression model for all genes") # title for the plot



The scatter plot shows a high aberration of the predicted values for the gene expression to the regression line. To check if the model can be applied to the issue, the testing data is used to examine the aberration between the predicted expression and the real expression of values, that were not used for the linear regression model. First the predicted values for gene expression are calculated base on the copynumber values for the testing dataset.

prediction <- predict(lrm.all.genes, newdata = testing.all.genes) # calculate the predicted expression values based on the copynumber

Next the standard error of the residuals is caculated for difference between the predicted values and the real values of the testing data set. For the calculation of the error the following formula was used

–> einfügen

In the formula n displays the number of rows in the given data frame. To calculate the residuals the difference between the real value and the predicted value of the gene expression is used.

sqrt(1/nrow(testing.all.genes) \* sum(testing.all.genes$expression-prediction^2)) # compute standard error with 200 values (rows)

## Warning in sqrt(1/nrow(testing.all.genes) \*  
## sum(testing.all.genes$expression - : NaNs wurden erzeugt

## [1] NaN

The value of 2.222435 is slightly higher than the residual standard error for the regression model, due to circumstance that the testing data did not account to the regression model.

### 6.3 Significance of the correlation coefficient

### 6.4 Creation of a linear regression model

### 6.5 Verfication of the model

## 7. Summary of Results

## 8. Conclusion