Logbook
Multi-Omics Project

Date: 30-04-2025

Written by: Jarno Jacob Duiker
Student Bio-informatics 2nd year, Hanze Honours, Faculty of Life sciences, Hanze UAS.

Contents

1 Introduction

This is a logbook for the Omics liver transplant project. The goal of this project is to perform a PCA on both the proteomics data and RNA transcriptomics data, and find a link.

This logbook shows the process of exploring data and processing this, applying different statistical principles like PLS-DA and PCA and some biological background on the results from these statistical principles. First, the paper that the project will be based on has been read and needs to be understood completely before a research question can be made. The paper that was chosen was the Thorne et al. (2023) paper, and this paper talked about biliary regeneration during normothermic preservation of donor livers. This paper looked into whether ECD livers could be recovered to a useable liver for transplantation by putting them through an NMP machine that would supposedly kickstart the regeneration of the liver cells.

1.1 Load libraries

```
library(tidyverse) # For data manipulation and visualization
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library(ggplot2) # For data visualization
library(ggfortify) # For PCA visualization
library(factoextra) # For PCA visualization
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(Rsamtools) # For Reading Bam files
```

```
## Loading required package: GenomeInfoDb
## Loading required package: BiocGenerics
##
## Attaching package: 'BiocGenerics'
##
## The following objects are masked from 'package:lubridate':
##
##     intersect, setdiff, union
##
## The following objects are masked from 'package:dplyr':
##
##     combine, intersect, setdiff, union
##
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
##
## The following objects are masked from 'package:base':
##
```

```
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, table,
##     tapply, union, unique, unsplit, which.max, which.min
##
## Loading required package: S4Vectors
## Loading required package: stats4
##
## Attaching package: 'S4Vectors'
##
## The following objects are masked from 'package:lubridate':
##
##     second, second<-
##
## The following objects are masked from 'package:dplyr':
##
##     first, rename
##
## The following object is masked from 'package:tidyr':
##
##     expand
##
## The following object is masked from 'package:utils':
##
##     findMatches
##
## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname
##
## Loading required package: IRanges
##
## Attaching package: 'IRanges'
##
## The following object is masked from 'package:lubridate':
##
##     %within%
##
## The following objects are masked from 'package:dplyr':
##
##     collapse, desc, slice
##
## The following object is masked from 'package:purrr':
##
##     reduce
##
## Loading required package: GenomicRanges
## Loading required package: Biostrings
## Loading required package: XVector
##
## Attaching package: 'XVector'
##
```

```
## The following object is masked from 'package:purrr':
##
##     compact
##
##
## Attaching package: 'Biostrings'
##
## The following object is masked from 'package:base':
##
##     strsplit
```

```
library(Rsubread) # Feature count
library(mdatools)
```

1.2 Reading the Paper

The paper selected for this multi-omics project is the *Bile proteome reveals biliary regeneration during normothermic preservation of human donor livers* Thorne et al. (2023)

The paper looks at the effects of NMP (Normothermic machine perfusion) on livers that are to be transplanted that may not be in the optimal shape, meaning they have high biliary injuries (Traumatic damage to the bile ducts) or lower biliary viability (a lower amount of living cells). The research was conducted on 55 livers that were deemed suitable for transplantation analysis. The livers came from ECD (extended criteria donors) and DCD (donation after circulatory death) donors.

To analyse the livers, a multitude of samples were taken. Before the NMP 46 bile duct biopsies were taken, 50 liver tissue biopsies were taken (these biopsies were taken before the livers underwent Hypothermic oxygenated machine perfusion - DHOPE) after undergoing DHOPE and COR (controlled oxygenated rewarming). The livers were put in the NMP, where metabolic processes, blood flow and simulation of other physiological conditions were started. 30 minutes after these livers were in the NMP, a Bile sample was taken, which was taken from all 55 livers. At 150 minutes, another bile sample was taken of 54 (not 55 because one sample did not contain any proteins) livers. At this point in the NMP process, all livers were analysed for a multitude of things to determine organ viability. The two distinct areas that were looked at were the hepatocellular viability, which focused on the metabolic functionality of the liver parenchyme (functional tissue of the liver, which makes up 80% of the liver volume as hepatocytes), this is assessed by criteria such as lactate clearance and bile production. The other area was biliary viability, which focuses on the functional capacity of cholangiocytes (epithelial cells of the bile duct). This was assessed by the specific biochemical composition of the bile Thorne et al. (2023) . After the viability assessment was concluded, 35 livers were viable for further NMP treatment pre-transplantation, 20 livers were deemed non-viable and therefore had their NMP stopped and were then discarded. For the 35 livers that continued, another sample of bile was taken at the END of the NMP treatment. However, 33 samples were taken because of complications with 2 livers at transplantation.

The Bile, a very important part of this paper. But what is this bile? According to Boyer (2013), bile consists of a number of things. ~95% of bile is water, Electrolytes, Organic Anions like Bile salts (Steroid/primary acids like cholic acid, chemodeoxycholic acid or secondary acids like deoxycholic acid and lithocholic acid), Lipids like cholesterol, Steroid hormones and estrogen. Proteins, peptides. Heavy metals and others are also found in bile. For a more in-depth look into what bile contains, I would suggest reading table 1 in the Boyer (2013) paper, which gives the known composition of bile in most species.

The BDI, meaning Bile Duct Injury, is a metric that shows how much damage the liver has based on 4 criteria. These include the presence of vascular lesions (abnormal growth or malformations in the blood vessel), stromal necrosis (Organ tissue/cell death), Injury to the periluminal peribiliary gland and injury to the deep peribiliary glands. Using these 4 criteria, a score was made called "a total histological bile duct injury score" This score ranged from 2 being the lowest to 14 being the highest, with the median being 7.

Using the previously mentioned median, the livers were divided into 2 groups. High BDI and Low BDI using this group system, several analyses were made, like seeing if BDI had the influence of biliary viability and other analyses.

PCA, Principal component analysis, was used in this paper and will be used in our research as well. For this reason, I made an explanation on how PCA works that is understandable to me, and I will share it here. PCA is an excellent tool to process datasets with a large number of factors. This is because when comparing factors, you can only plot so many; otherwise, the dimensions would be too large and too difficult to interpret. Using PCA, u can turn, for example, 200 factors into 5 principal components (Principal components are ranked from most important to least important, meaning PRC1 will be more important than PCR3). To see how much info of the data set is explained within each principal component, a SCREE plot could be used. This plot gives percentages on how much variance/info each PC contain

Here is an example of a scatter plot

```
# take only the collums with ints
df <- iris[, -5]

# Perform PCA on the iris dataset
pca <- prcomp(df, scale = TRUE)

# Extract the eigenvalues from the PCA object to plot importance per component
eigenvalues <- pca$sdev^2

summary(pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.7084 0.9560 0.38309 0.14393
## Proportion of Variance 0.7296 0.2285 0.03669 0.00518
## Cumulative Proportion  0.7296 0.9581 0.99482 1.00000
```

```
# Percentage of variance explained
plot(eigenvalues/sum(eigenvalues), type = "b",
     xlab = "Principal Component",
     ylab = "Percentage of Variance Explained")
```

In Fig. 1, A scree plot was made of the PCA of the iris dataset

The scree plot shows that the first two principal components explain the most variance in the data. The third and fourth principal components explain much less variance.

Based on the scree plot, we can conclude that the first two principal components are sufficient for capturing the most important information in the data.

What is an eigenvalue? It is a measure of the strength or importance of its corresponding Principal Component. A larger eigenvalue means the Principal Component captures more of the total variance in the data. The sum of all eigenvalues is equal to the total variance in the original dataset. The code eigenvalues <- psa@sdev^2 calculates the squared standard deviations, calculating these variances along the principal component directions, which are the eigenvalues of the data's covariance/correlation matrix. These eigenvalues are needed for understanding how much information each PC retains, for this in turn helps decide how many PC will be necessary to use.

If we use PC1 and PC2 from the PCA of the iris dataset, we will have about 90% of the info of the original dataset. This will be more than enough. After deciding how many PC's need to be used, it is important to plot them using a scatterplot, for example.

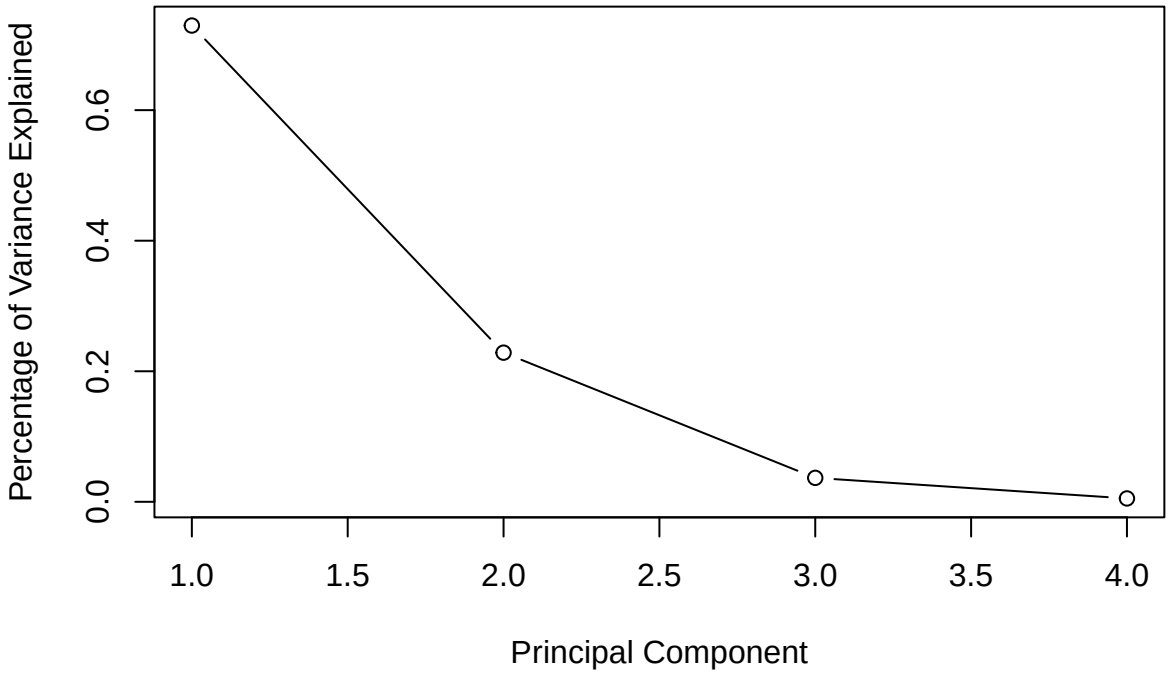
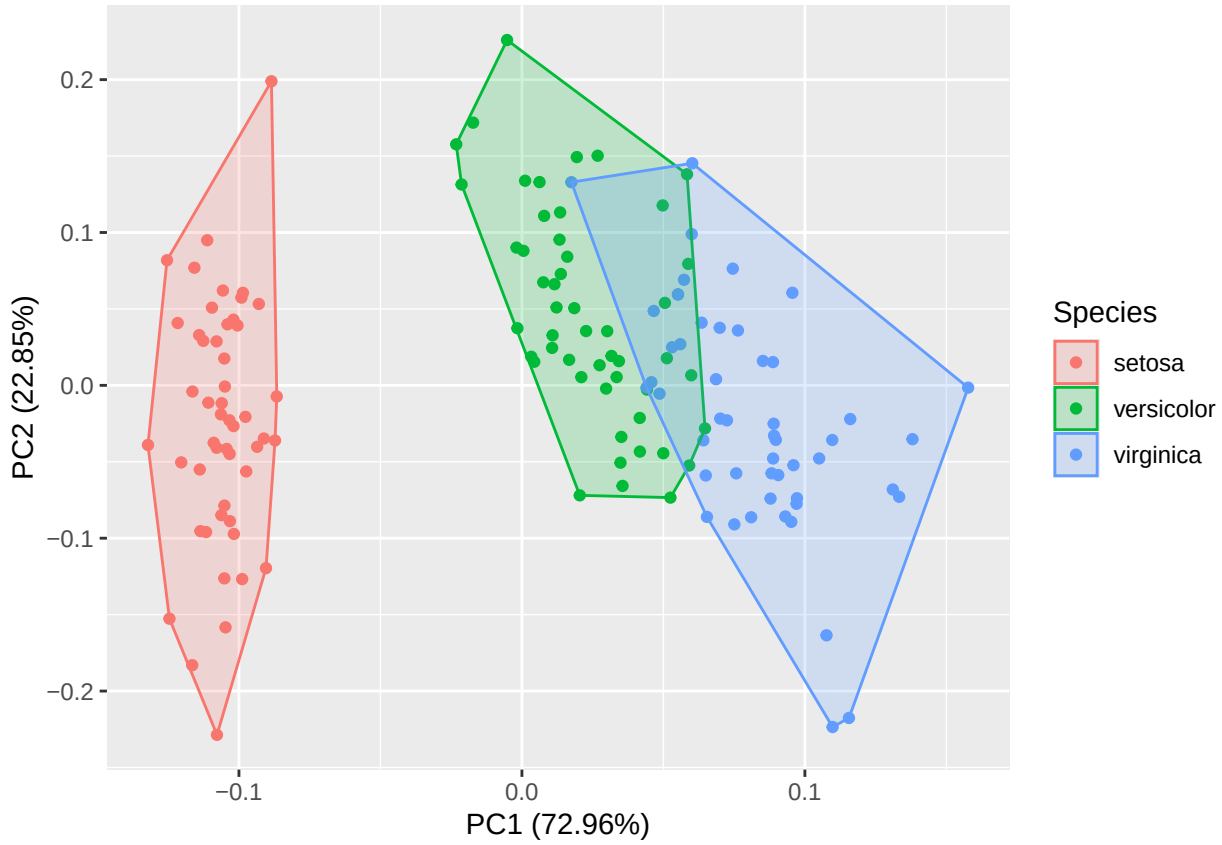


Figure 1: Fig.1 - SCREE plot

```
# Make a scatter plot that shows the different groups
iris.pca.plot <- autoplot(pca,
                          data = iris,
                          frame = TRUE,
                          colour = 'Species')

iris.pca.plot
```



Here, a scatterplot was made of the PCA of the iris dataset. The plot shows the first two principal components (PC1 and PC2) on the x and y axes, respectively. Each point represents an observation (in this case, a flower), and the colour indicates the species of the flower. The groups are shown by taking all the most distant points and drawing a line around the rest of the dots.

```
# Graph of the variables
fviz_pca_var(pca, col.var = "black")
```

This plot shows the variables and their correlation, also known as loadings. The arrows represent the original variables, and their direction and length indicate how they contribute to the principal components. A longer arrow means a stronger contribution. The plot also shows the correlation between the variables. For example, Petal Length and Petal Width are highly correlated, as indicated by their proximity and similar direction. Sepal Length is positively correlated with Petal Length and Width, but not as strongly. Sepal Width is negatively correlated with both Petal Length and Width.

Applying PCA to the data. After reading the PCA that was done in the paper and exploring and explanation of different sides of a good PCA, I wrote down a small list of suggestions that we should keep in mind when we are doing our PCA. This is to first, do a proper analysis of the data, which means looking

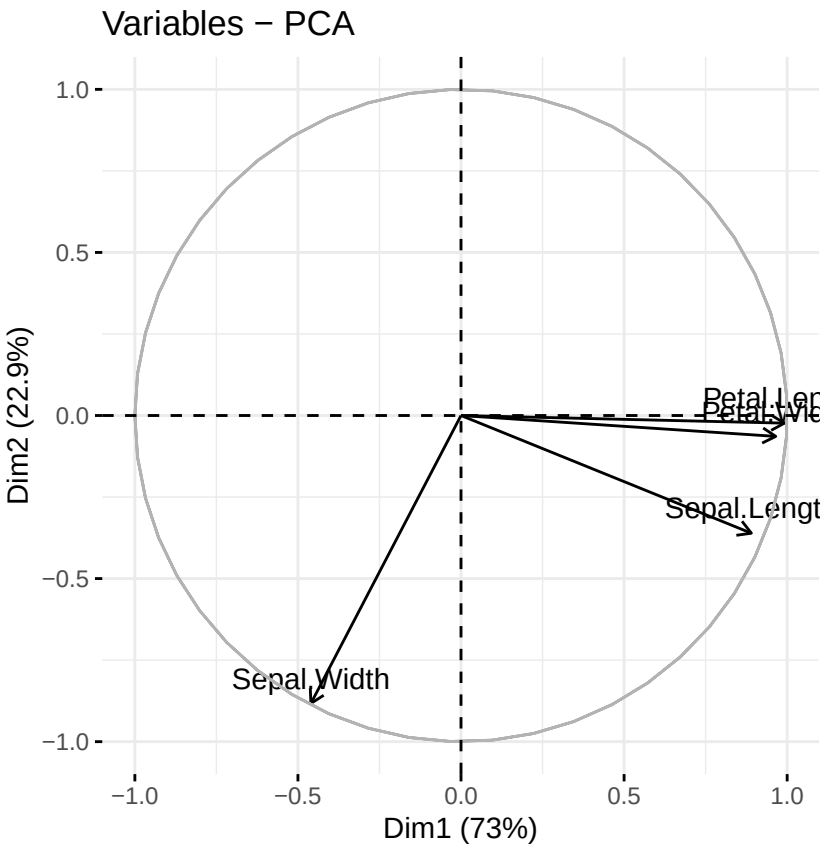


Figure 2: Fig.2 - Loading plot

at the data and seeing if there are any outliers or missing values. If there are outliers or missing values, we should remove them from the dataset. After this, we should scale the data, this is important because PCA is sensitive to the scale of the data. If the data is not scaled, the PCA will be biased towards the variables with larger scales. After scaling, we can do a PCA on the data and plot it using a scatterplot. This will give us a good overview of what the data looks like and if there are any patterns in it. After this, we can do a loading plot to see how much each variable contributes to each principal component. This will help us understand which variables are important for the PCA and which ones are not.

After this PCA, we should analyse the cluster groups by subsetting the data and looking at the different groups. This will help us understand how the different groups are related to each other and if there are any patterns in the data. After this, we can do a correlation analysis to see how the different variables are related to each other. Also, a heatmap can be made to see the correlation. E

ANOVA?

KEGG,

Diablo

$$\max_{\alpha_h^{(1)},\ldots,\alpha_h^{(Q)}} \sum_{\substack{i,j=1 \\ i \neq j}}^{Q} c_{i,j} \cdot \operatorname{cov}\left(X_h^{(i)}\alpha_h^{(i)}, X_h^{(j)}\alpha_h^{(j)}\right) \tag{1a}$$

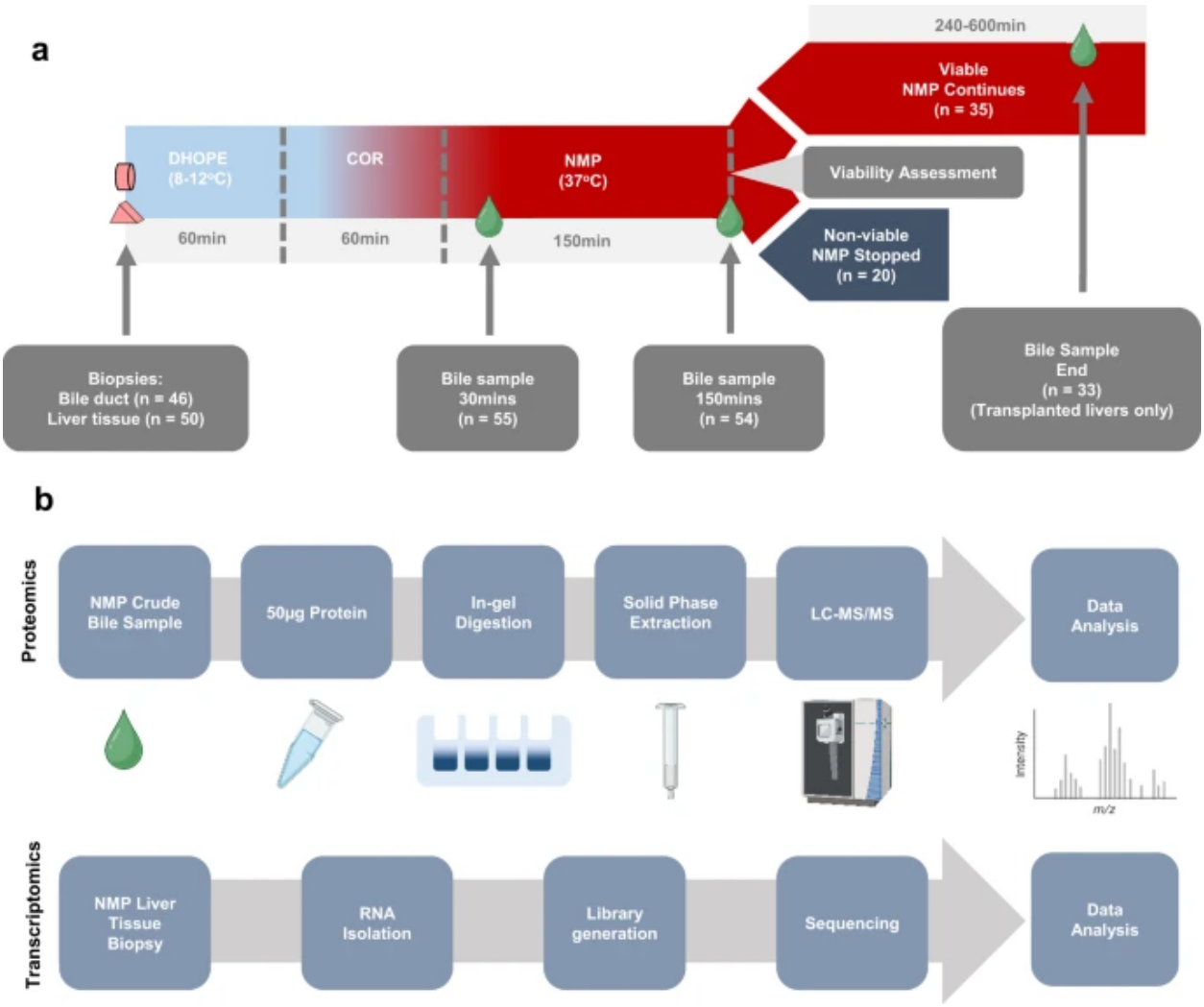
MOFA+

Transcriptomics

1.2.1 Proposal for research question

Question: Does longer usage of NMP lead to benefits in preservation and functional assessment of Sub-optimal ECD livers?

Why? The reason for this research question is because the paper itself also refers to a study that is going towards this and also because we have quite a lot of data that has been taken over multiple points in the NMP cycle, so that we can use the change between these points for the estimates of the next points. Finding certain biomarkers that could support this hypothesis should be reasonably doable, and the paper also mentions interesting genes/proteins that could support this hypothesis.

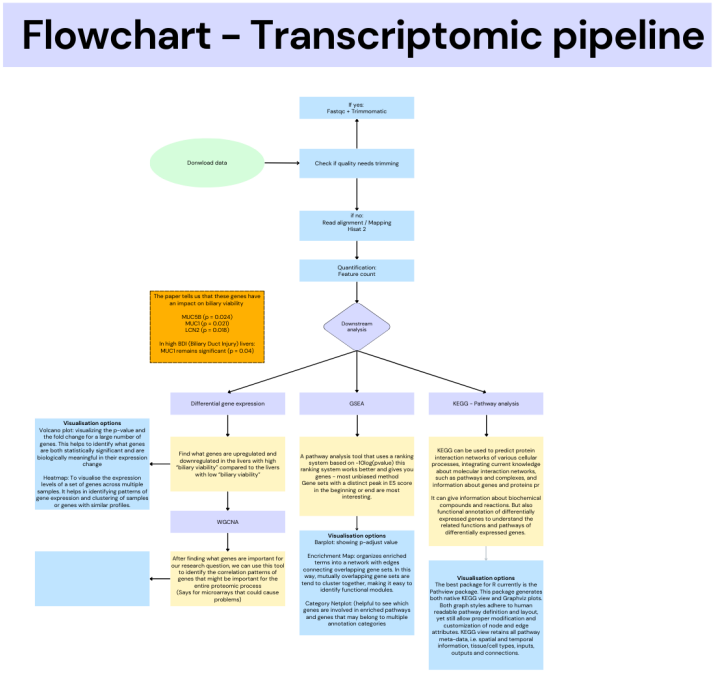


1.2.2 Chosen research question

Using multi-omics to identify bile- and liver-biomarkers that correlate with biliary viability of ECD livers

Look at this by comparing the samples from the 2 'biliary viability' groups (high vs low) and do this for the bile proteomics data and the liver transcriptomics data, and then see which proteins are more/less abundant or which metabolic pathways in the liver are more/less expressed per group. The biliary viability is determined as follows: "The addition of biliary viability criteria, such as bile pH, glucose reabsorption and bicarbonate secretion, for selection of viable liver grafts has been explored by us and others", but it is not discussed whether there is a certain proteomics/transcriptomics profile associated with this 'biliary viability'. The biliary viability score describes the state of 'cholangiocytes', which are specialised epithelial cells lining the bile ducts. Determining the biliary viability in advance (before transplantation) reduces the risk of cholangiopathies (cholangiocyte disorders) after transplantation. The article also states that the data is suitable for further refining the 'viability assessment criteria' ("Our comprehensive bile proteomics and liver transcriptomics data sets provide the potential to further evaluate molecular mechanisms during NMP and refine viability assessment criteria viability assessment criteria"). A reason to look for biomarkers, instead of using the normal tests, could be, for example, that values at the proteomics/transcriptomics level provide a better picture.

1.2.3 Flowchart for a potential pipeline of transcriptomics data



Note - *After giving the mid-term presentation, it has been decided that besides the DGE, also a PCA and PLS-DA will be performed on the data.*

2 Transcriptomics experimentation

First, it is important to check whether our data is usable before starting the pipeline. The paper where we are using the data from referenced a document that contained the links for each transcriptomics file used. There were two files with links to the transcriptomics data, one sent me to a website that had BAM files available. Bam, meaning Binary alignment map, this means the file should then be ready for feature count.

Let's look at the BAM header to see if the reads are mapped.

```
scanBamHeader(files = "data/ERR12161053.bam")
```

The BAM header is quite empty, which gives me some concern, seeing how this isn't a very common thing, but still running them through featurecount will give a clearer00 overview.

To use feature count, the following is done. A reference genome is chosen in this case, the hg38 version, because this is also the genome used in the paper Thorne et al. (2023)

Then we want to specify that the feature count only should look for nucleotides that are based on an exon, so they will for sure get expressed and that the data is paired end.

```
featureCounts("data/ERR12161053.bam", annot.inbuilt = "hg38", GTF.featureType = "exon",isPairedEnd = TRU
```

Description: df [14 × 2]

Status <chr>	ERR12161053.... <int>
Assigned	0
Unassigned_Unmapped	10711879
Unassigned_Read_Type	0
Unassigned_Singleton	0
Unassigned_MappingQuality	0
Unassigned_Chimera	0
Unassigned_FragmentLength	0
Unassigned_Duplicate	0
Unassigned_MultiMapping	0
Unassigned_Secondary	0

1–10 of 14 rows Previous 1 2 Next

There seems to be a big problem with the BAM files, it is not at all aligned. Therefore, we have to download the FASTA files and start from the beginning.

First, for the indexing, I will need to index the reference genome.

Important note - The commands will be commented out due to these being run on the assemblix server (our study's private server network that has bioinformatic tools) and can not be run on my laptop.

Using hisat2, I indexed the reference genome. Due to the article not saying what ref genome they used, we will be using the most recent GRC38 human genome. The following bash code was used to index the genome.

```
#hisat2-build GCF_000001405.40_GRCh38.p14_genomic.fna h38_index
```

This command is used to build an index for the reference genome, which is quite important pre-read alignment due to it being requered by hisat2 and also making the whole process faster.

After the index with reference genome 38 (the most recent human reference genome released which is used widely), hisat gave me about 20% mapping result. This resulted in me also indexing and mapping with an old reference genome, the 37 human reference genome. This gave me a 38% mapping rate with an unpaired fastq sample. This is ofcourse incredibely low and shows there were problems when the sample was read to a fastq.

```
#!/bin/bash
#
#SBATCH --job-name=hg19index
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=4G

hisat2-build hg19.fa.gz h19_index
```

(This is using Slurm) Slurm uses multiple nodes (computes) in combination to perform a task; in this case, the task was to index a reference genome. i Used slurm to make an indexed h19 genome this is because sometimes researches use old genomes instead of the newest one because it was more widely used so better to compare with when looking at old studies.

Here is the file that gave me the highest mapping, this being ~38%

```
#hisat2 -x h19_index -U R10_1.fastq -S R10_h19_unpaired.sam --phred33
```

Now I will try using feature count again,

```
# The same as before only look for exon and because im using a non inbuild ref genome i have to say its
counts_results <- featureCounts("data/R10.sam", annot.ext = "data/Homo_sapiens.GRCh37.75.gtf", GTF.featu
```

We got a mapping percentage around ~22% this is incredibly low agian because this is ~22% of the earlier 38% resulting in a feature count of 8.36% of the orignal data. Simply said that is way too low and we cannot use that.

```
# write the results from feature counts to a file
write.table(
  counts_results$counts,
  file = "data/R10_counts_matrix.txt", # output file name
  sep = "\t", # look for tabs
  quote = FALSE,
  row.names = TRUE # Gen IDs as rownames
)
```

To look at the count results more clearly i wrote them to a file called counts_results. This way i could load it and plot it. But in the end the feature count was also very low and didn't show anything significant really. Because this was a test on one file after this find i communicated it back to my teammates to see what the propper action was now.

```
counts_results
```

The feature count results show that the number of reads mapped to each gene is very low, with most genes having 0 counts. This indicates that the data is not usable for further analysis.

3 MDA TOOLS - practice

To practice with MDA tools and our data not yet being usable, I decided to use the iris data and perform PLS-DA on it.

PLS-DA wants to look at groups and see what separates them based on variance importance. To do this, we give

```
# calibration set from iris data (3 classes)

# x has the numerical data from the iris dataset
x.cal = iris[seq(1, nrow(iris), 2), 1:4]

# c has all the classes for the iris data set
c.cal = iris[seq(1, nrow(iris), 2), 5]

#Here the plsda is called using the variables from before and asking the model to generate 3 components
model = plsda(x.cal, c.cal, ncomp = 3, cv = 1, info = 'IRIS data example')
model = selectCompNum(model, 1)
```

```
summary(model)
plot(model)
```

The model generates an output that shows the 3 classes. For each class, it shows the true positive, false positive, true negative and false negatives. This can then give us a calculation on how accurate the model is and how good it will perform on the new data.

```
# a specific look at the 2nd component
summary(model, nc = 2)
plot(model, nc = 2)
```

Here the component 2 is shown, and it shows low accuracy for the versicolor class, which is expected because the versicolor class is the most difficult to separate from the other two classes.

```
# a specific look at the 3rd component
summary(model, nc = 3, ncomp = 3)
plot(model, nc = 3, ncomp = 3)
```

This shows the 3rd component, which is not very useful because it does not separate the classes well. The versicolor class is still the most difficult to separate from the other two classes.

```
par(mfrow = c(2, 2))
plotSpecificity(model)
plotSensitivity(model)
plotMisclassified(model)
par(mfrow = c(1, 1))
```

This code plots the specificity, sensitivity, and misclassified points for the model. The specificity and sensitivity plots show how well the model performs for each class, while the misclassified plot shows which points were misclassified by the model. It seems like all components have low missclassification rate, which is expected because the iris dataset is a well-known dataset with clear separations between the classes. the specificty and sensitivity seem the same no weird output.

```
par(mfrow = c(2, 2))
plotPredictions(model)
plotPredictions(model, res = "cal", ncomp = 2, nc = 2)
plotPredictions(structure(model, class = "regmodel"))
plotPredictions(structure(model, class = "regmodel"), ncomp = 2, ny = 2)
par(mfrow = c(1, 1))
```

This code plots the predictions of the model for each class. The first plot shows the predictions for all components, while the second plot shows the predictions for the calibration set with 2 components. The third and fourth plots show the predictions for the regression model with 2 components and 2 classes. The predictions seem to be quite accurate, with most points being correctly classified.

```
par(mfrow = c(2, 2))
plotXYScores(model)
plotYVariance(model)
plotXResiduals(model)
plotRegcoeffs(model, ny = 2)
par(mfrow = c(1, 1))
```

This code plots the X and Y scores, the Y variance, the X residuals, and the regression coefficients for the model. The X scores plot shows how well the model separates the classes based on the first two components. The Y variance plot shows how much variance is explained by each component. The X residuals plot shows how well the model fits the data, and the regression coefficients plot shows how much each variable contributes to the model.

3.0.1 X-distances (Score vs. Orthogonal distance)

The X-distance shows score distance (h/h0) this shows how far a point lies from the centre of the model and in the direction of the most important componten(s). The orthogonal distance (q/q0) shows how far a point lies from the modelface (so the deviation from the model). The grey lines indicate boundaries for outliers and influential points. Points close to (0,0) are well explained by the model, while points outside the boundary lines (especially to the right or high) may be outliers. In my plots, most points fall within the boundaries → no major outliers.

3.0.2 Regression Coefficients

The weights (coefficients) of each **predictor** (the 4 flower characteristics: sepal length, sepal width, petal length, petal width). This tells the importance of each variable for the distinction of the respective class (*setosa*, *versicolor* or *virginica*). Positive/negative values indicate whether a variable contributes positively or negatively to the likelihood of a point belonging to that class. For example, in *setosa* (first figure), "Predictor 3" (likely **petal length**) has a strong negative influence.

3.0.3 Misclassified

This shows the misclassifications of the model, which is important to understand how well the model performs. The plot shows it for the training and for the cross-validation. The x-axis shows the number of used components. The lower the number of misclassifications, the better the model performs. In my plots, *setosa* (first figure) has no misclassifications, while *versicolor* and *virginica* have some. The third figure shows that using 3 components improves the classification of *virginica*.

3.0.4 Predictions (cv)

The predictions show the true classes in colours versus the expected classes per object. Every point is an object from the dataset. The y-axis shows the predicted class (*setosa*, *versicolor*, *virginica*), and the x-axis shows the observation numbers. Correct predictions are points on their correct row, while misclassifications are points on the wrong row (e.g., a red point on the "versicolor" row means a *virginica* was classified as *versicolor*). In the first figure, almost all predictions are correct, especially for *setosa*.

Figure 1 (Setosa): perfect classifcation, pred 3 plays a crucial negative role. no misclasifcations and score/distance ok meaning a trustable model.

Figuur 2 (Versicolor): Some more misclassifactions, positive impact of predictor 3 and some errors visible in the "Predictions" plot. The model is not as good as the setosa model, but still decent.

Figuur 3 (Virginica, ncomp = 3): Better classification because of more components, positive impact of predictors 2 through 4 and a few fewer mistakes in the versicolor figure.

4 PLS-DA proteomics data

4.1 PLS-DA our data

PLS Discriminant Analysis (PLS-DA) is a discrimination method based on PLS regression. The used R library that we will use is the mdatools library, which is a library that is used for multivariate data analysis. PLS-DA is a supervised method, meaning that it uses the classes of the data to train the model. This means that we need to have a dataset with classes to use PLS-DA. The PLS-DA model will then try to find the best linear combination of the variables that separates the classes. The two classes we have are High and Low viability.

```
load("/students/2024-2025/Thema08/liver-transplant/proteomics/normalized_imputed_count_data_150min.Rdata

first_data <- assay_norm # Assay norm Rdata object
as.data.frame(first_data) # Transform the data to a data frame
```

The first step is to load the data and transform it into a data frame. The data is loaded from an Rdata file, which contains the normalized and imputed count data for the proteomics data at 150 minutes. The reason the 150 min data is chosen because this only showed to be the significant one out of previous trials and other teammate data.

```
data_for_plsda <- t(first_data) #transpose the dataframe so the model runs quicker
data_for_plsda <- as.data.frame(data_for_plsda)
```

The data is then transposed, which means that the rows and columns are switched. This is done because the PLS-DA model expects the samples to be in the rows and the variables in the columns. After transposing, the data is converted back to a data frame.

```
# collect all the sample names in a seperate list
sample_names <- rownames(data_for_plsda)
classes <-factor(c(sample_names))
```

The sample names are collected in a separate list, and the classes are created as a factor. The classes are the sample names, which will be used to train the PLS-DA model. The classes are important because they tell the model which samples belong to which class.

```
print(dim(data_for_plsda)) # printing the dimensions of data for plsda
print(length(classes)) # printing length of the classes to check if they are the same as the dimensions
ncomp_to_try <- min(nrow(data_for_plsda) - 1, ncol(data_for_plsda), 10) #The ncomp settings for the mod
```

The dimensions of the data for PLS-DA are printed to check if they are correct. The length of the classes is also printed to check if they are the same as the dimensions. The number of components to try is set to the minimum of the number of rows minus 1, the number of columns, and 10. This is done because the number of components should not be larger than the number of samples minus 1, and it should not be larger than the number of variables.

```
# training the model using the prepped data, ncomp and classes
model_plsda <- plsda(data_for_plsda, classes, ncomp = ncomp_to_try, cv = 1) # cv is cross validation
```

The PLS-DA model is trained using the data for PLS-DA, the classes, the number of components to try, and cross-validation. The cross-validation is set to 1, which means that leave-one-out cross-validation is used. This means that for each sample, the model is trained on all other samples and tested on the left-out sample.

```r
summary(model_plsda)
```

This shows the results of the models for each class

1.

4.2 PLS-DA model (class plsda) summary

Info: Number of selected components: 1 Cross-validation: full (leave one out)

Class #1 (High_NMP_Bile_Proteomics_09) X cumexpvar Y cumexpvar TP FP TN FN Spec. Sens. Accuracy Cal 26.65 2.38 0 0 42 1 1 0 0.977 Cv NA NA 0 0 42 1 1 0 0.977

Class #2 (High_NMP_Bile_Proteomics_102) X cumexpvar Y cumexpvar TP FP TN FN Spec. Sens. Accuracy Cal 26.65 2.38 0 0 42 1 1 0 0.977 Cv NA NA 0 0 42 1 1 0 0.977

Class #3 (High_NMP_Bile_Proteomics_109) X cumexpvar Y cumexpvar TP FP TN FN Spec. Sens. Accuracy Cal 26.65 2.38 0 0 42 1 1 0 0.977 Cv NA NA 0 0 42 1 1 0 0.977

Class #4 (High_NMP_Bile_Proteomics_14) X cumexpvar Y cumexpvar TP FP TN FN Spec. Sens. Accuracy Cal 26.65 2.38 0 0 42 1 1 0 0.977 Cv NA NA 0 0 42 1 1 0 0.977

Here is a small snippet of the first results.

The summary shows the number of selected components, the cross-validation method used, and the results for each class. The results include the true positives (TP), false positives (FP), true negatives (TN), false negatives (FN), specificity (Spec.), sensitivity (Sens.), accuracy, and calibration (Cal). The results show that the model has a high accuracy and specificity for all classes, which is a good sign.

Now it is possible to tweak the model settings. Right now all the different samples are used, but what if we divide them between HIGH_NMP and LOW_NMP subsets?

To do this, we need to create a new factor that contains the classes for each sample. The classes will be "High_NMP" and "Low_NMP", based on the sample names. This is done by looking for the sample names that contain "High_NMP_Bile_Proteomics" or "Low_NMP_Bile_Proteomics" and assigning them to the respective class.

```r
sample_identifiers <- rownames(data_for_plsda)

actual_classes <- character(length(sample_identifiers))

# for each item in sample_identifiers look if it matches High_nmp_bile then put it in the class High_nm
for (i in 1:length(sample_identifiers)) {
  if (grepl("High_NMP_Bile_Proteomics", sample_identifiers[i])) {
    actual_classes[i] <- "High_NMP"
  } else if (grepl("Low_NMP_Bile_Proteomics", sample_identifiers[i])) {
    actual_classes[i] <- "Low_NMP"
  } else {
    actual_classes[i] <- "Unknown"
    warning(paste("Sample", sample_identifiers[i], "couldn't be put in a class"))
  }
}

classes_corrected <- factor(actual_classes)

print(length(classes_corrected)) # Check the length to see if its correct
print(summary(classes_corrected)) # check how the classses look
```

This code creates a new factor called `classes_corrected` that contains the classes for each sample. The length of the classes is printed to check if it is correct, and the summary of the classes is printed to see how the classes look. The warning message will be shown if there are any samples that could not be put in a class.

```
# now re run the model after tweaking the options
model_plsda_corrected <- plsda(data_for_plsda, classes_corrected, ncomp = 10, cv = 1)

summary(model_plsda_corrected)
```

4.3 PLS-DA model (corrected plsda) summary

Info: Number of selected components: 3 Cross-validation: full (leave one out)

Class #1 (High_NMP) X cumexpvar Y cumexpvar TP FP TN FN Spec. Sens. Accuracy Cal 36.75 89 28 0 15 0 1.0 1.000 1.000 Cv NA NA 25 6 9 3 0.6 0.893 0.791

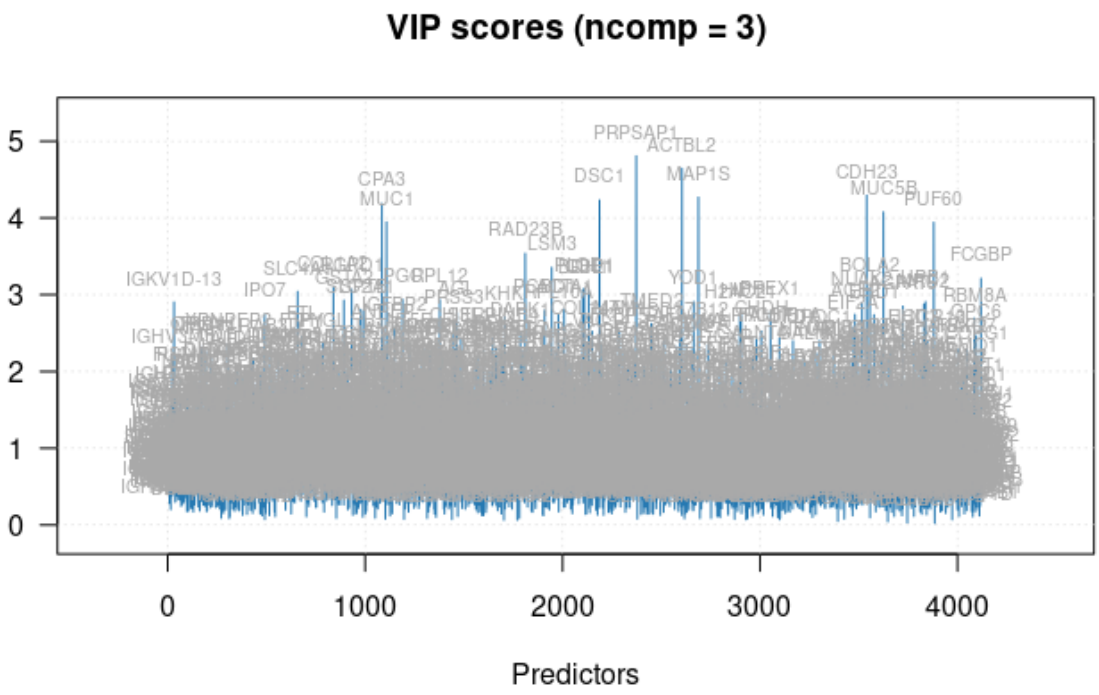
Class #2 (Low_NMP) X cumexpvar Y cumexpvar TP FP TN FN Spec. Sens. Accuracy Cal 36.75 89 15 0 28 0 1.000 1.0 1.000 Cv NA NA 9 3 25 6 0.893 0.6 0.791

The summary shows the results of the model after correcting the classes. The number of selected components is 3, and the cross-validation method used is still leave-one-out. The results for each class show that the model has a high accuracy and specificity for both classes, which is a good sign. The true positives, false positives, true negatives, and false negatives are also shown for each class. However the model seems to have a bit of overfitting, but let's look at the results anyway.

5 Results

5.1 Results - plots

```
plotVIPScores(model_plsda_corrected, show.labels = TRUE, show.legend = FALSE, ncomp = 1)
```



A VIP plot, also known as a variable importance plot, tells us the importance of each variable. In this model, there are a few really important variables seen by the height of their bar. The most important proteins in this model are PRPSAP1, ACTBL2, CDH23, MUC5B, MUC1, and CPA3; these proteins should be looked at in the main dataset. Some of these proteins were also called important in our paper.

```
plotPredictions(model_plsda_corrected, show.legend = FALSE, ncomp = 3)
```

There is

```
plotXLoadings(model_plsda_corrected, show.labels = T)
```

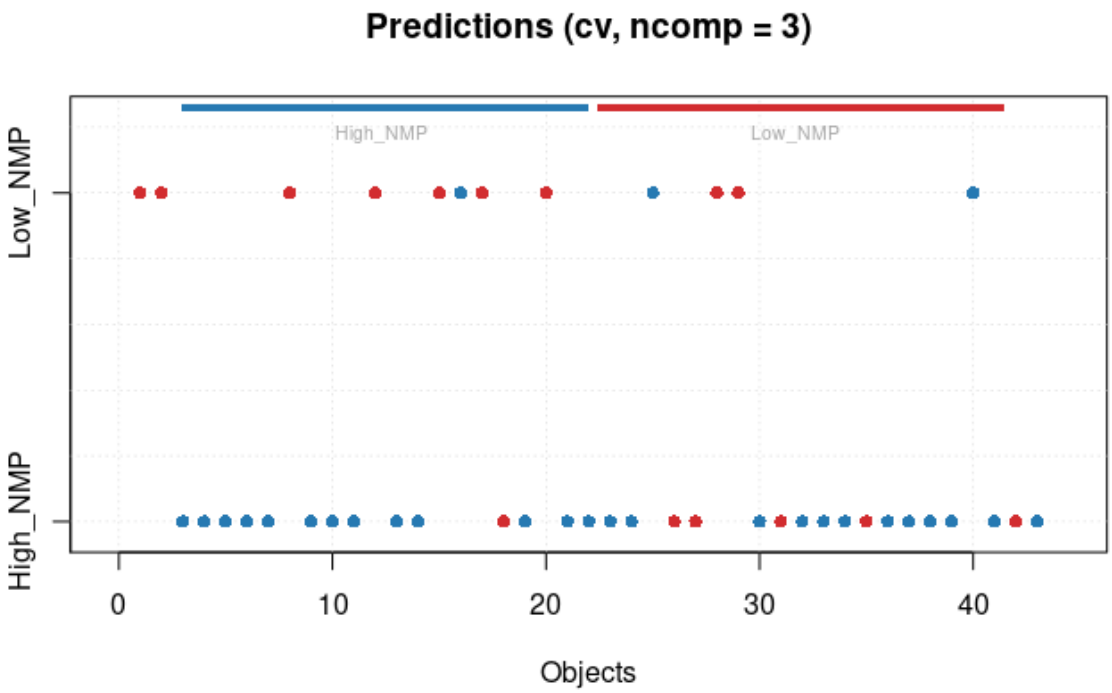
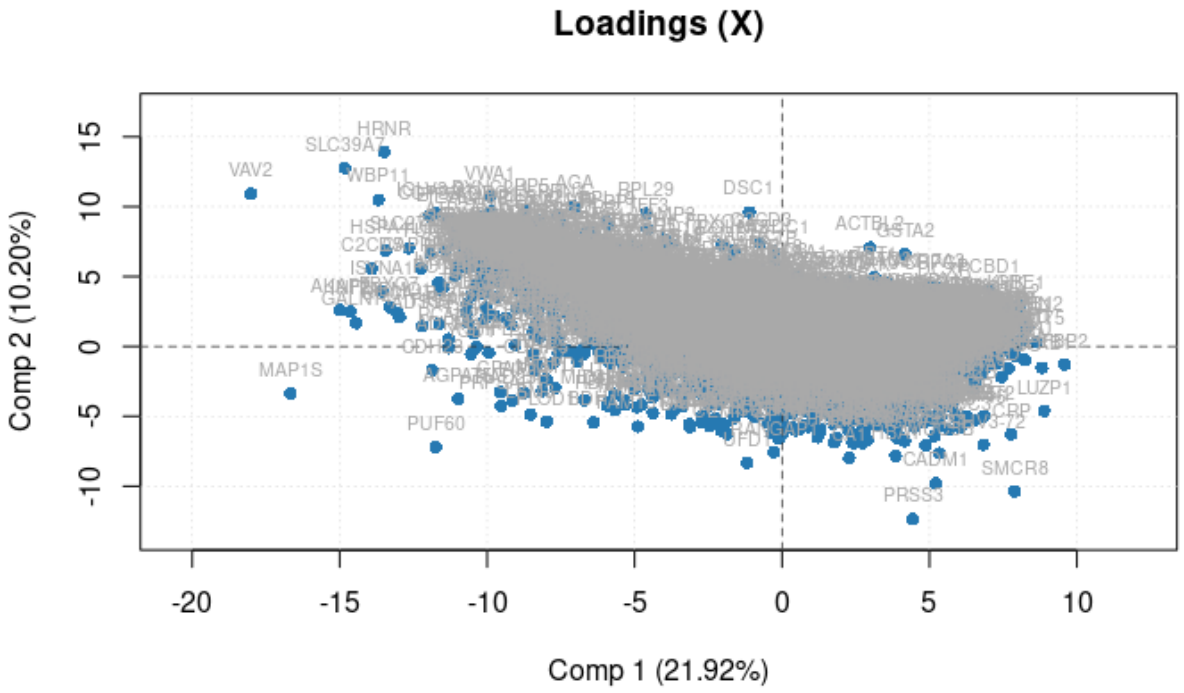


Figure 3: The model seems to be good at differentiating between the two classes, there are some misqualifications seen by the red dots in the high NMP field.



Loading plot identifies the original variables that are most impactful for the 2 main components, it helps understand what variables work together and if there are any patterns. Vav2, slc39A7 and HRNR have a high loading for component 2, and SMCR8, PRSS3, and LUZP1 have a high loading for the component 1.

```
plotXVariance(model_plsda_corrected, show.labels = T)
```

```
plotVIPScores(model_plsda_corrected, show.labels = TRUE, show.legend = FALSE, ncomp = 1)
```

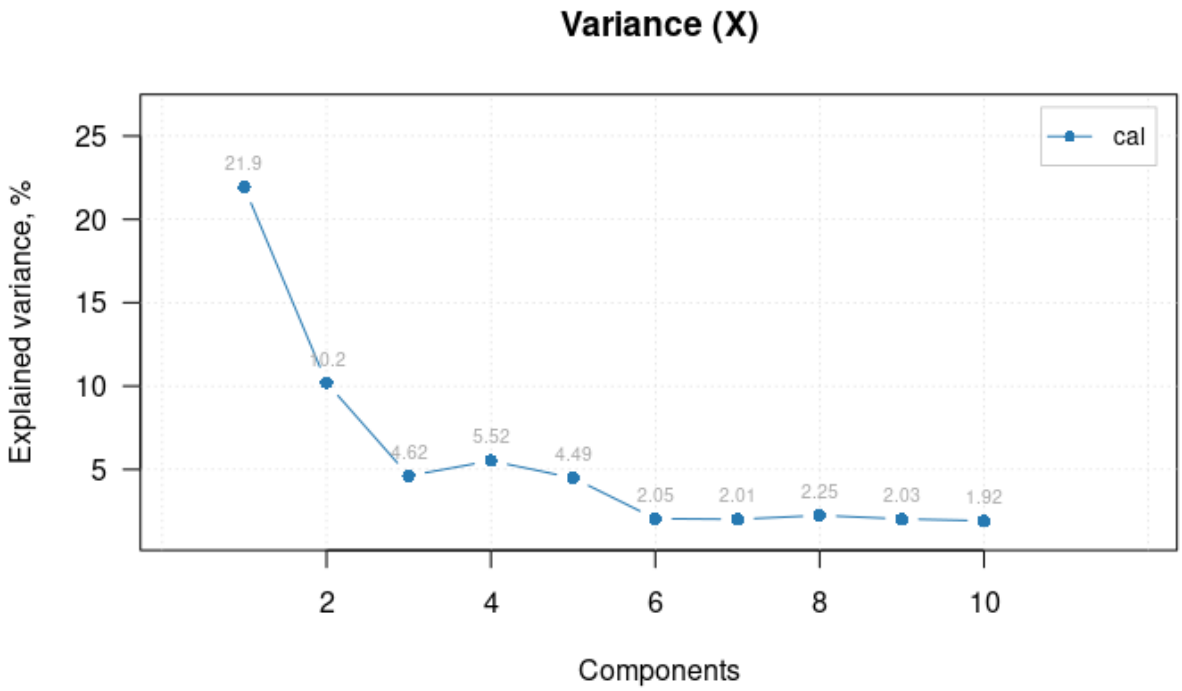
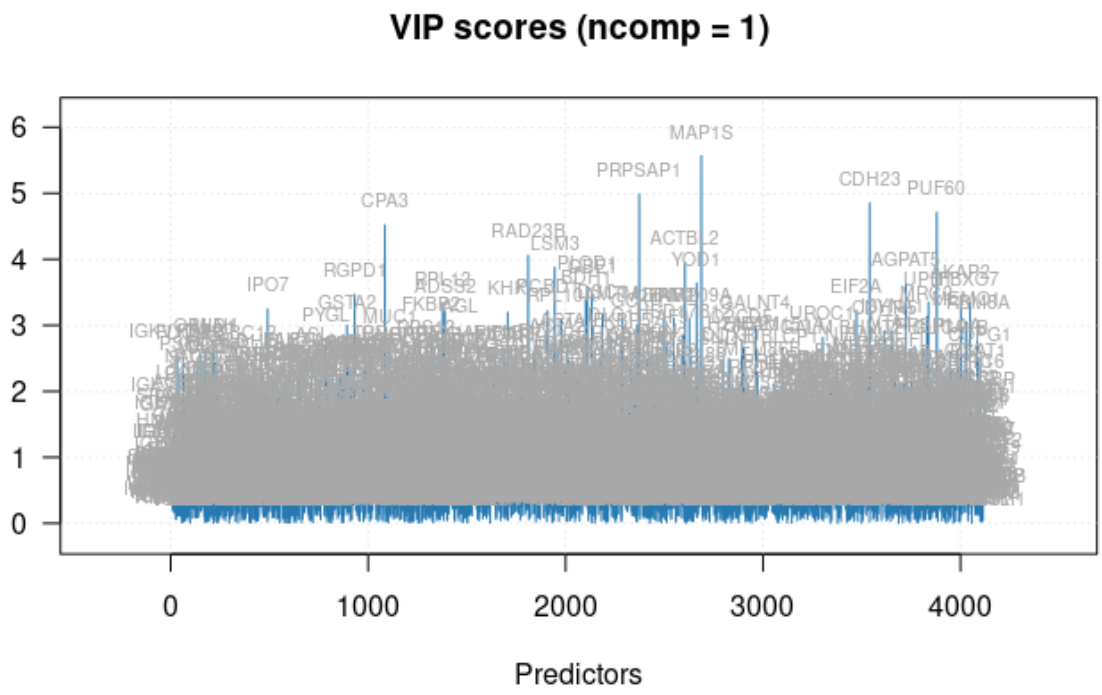


Figure 4: It seems that component 1 is the most important, so we should analyse this for the most significant proteins. After component 1 we can look at what other components give us.



It seems that our ncomp 1, which was shown in the previous Variance graph, was shown to contain the most information and has different important variables than ncomp 3. The most important proteins in this component are the MAP1S, PRPSAP1, CDH23, PUF60 and CPA3. These are proteins to look at in a subset of the total data.

```
# Scatter plot of Component 1 vs. Component 3
plot(model_plsda_corrected, ncomp = 1)
```

Regression coefficients - predictions with positive coëfficiënts say it belongs to the High_nmp class, and the ones with a negative prediction value say they don't belong to this class. The line is the point 0, so this is used as the reference point.

```
# Take the xloadings from the model data
xloadings <- model_plsda_corrected$xloadings

# put the data that was used for the model in a new variable
X <- data_for_plsda

# scale and center the data according to the model
X_scaled <- scale(X, center = model_plsda_corrected$center,
                     scale  = model_plsda_corrected$scale)

# Use matrix multiplication to calculate the xscores
xscores <- X_scaled %*% xloadings

# put these xscores in a new dataframe
as.data.frame(xscores)
```

The xscores are calculated by multiplying the scaled data with the xloadings from the model. This gives us

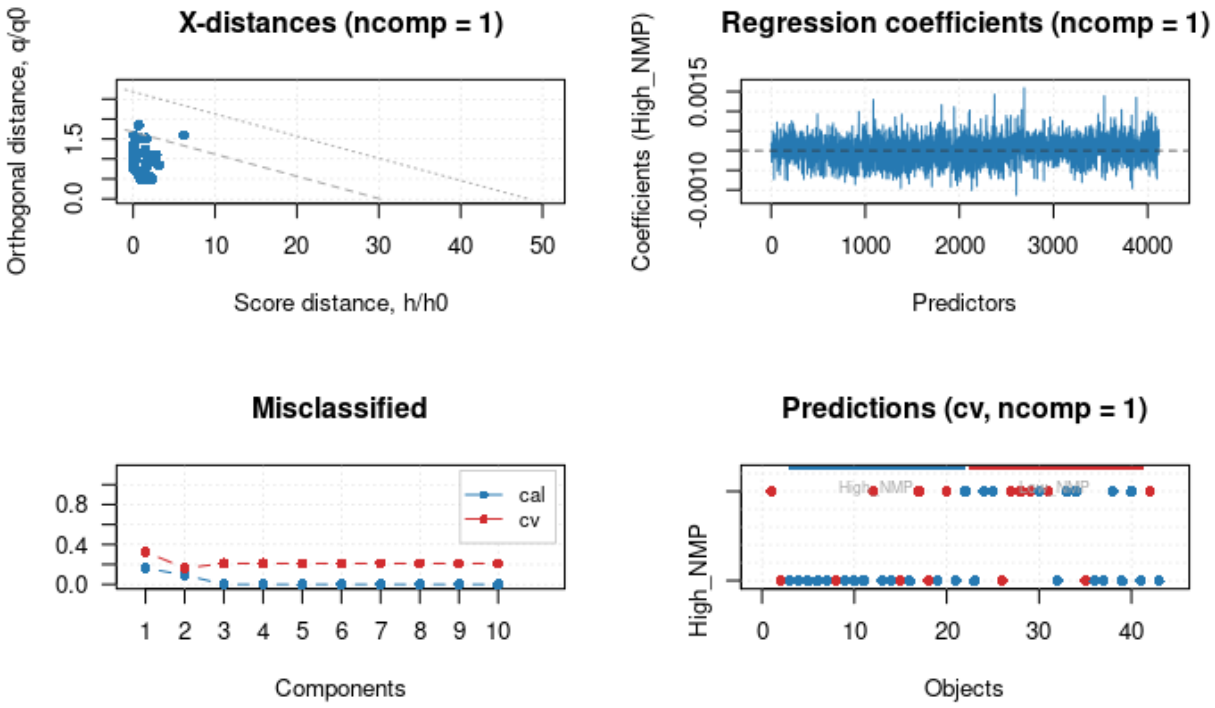


Figure 5: **X-distances** - This plot shows us how good the object can be moddeled in the X space. Our model shows that most of the dots are under the line, meaning there are no big outliers in the X space, and they are described well.

the scores for each sample in the model space. The xscores are then put in a new data frame for further analysis.

```
# devide the NMP's to high and low groups
NMP <- ifelse(grepl("High_NMP_Bile_Proteomics", rownames(xscores)), "High",
              ifelse(grepl("Low_NMP_Bile_Proteomics", rownames(xscores)), "Low", "Unknown"))

# Give each score the correct nmp group (high or low)
xscores <- cbind(xscores, NMP)

# take the xscores from the components that have the most variance
important_xscores <- subset(xscores, select=c("Comp 1", "Comp 2", "Comp 3", "NMP"))
```

The xscores are then divided into two groups, High and Low NMP, based on the sample names. The NMP variable is created to indicate which group each sample belongs to. The xscores are then combined with the NMP variable to create a new data frame called `important_xscores`, which contains only the components that have the most variance.

```
# plot xscores using ggplot, first plot comp 1 and comp 2
ggplot(important_xscores, aes(x = `Comp 1`, y = `Comp 2`, color = NMP )) +
  geom_point(size = 3) +
  labs(title = "PLS-DA Scores Plot", x = "Component 1", y = "Component 2") +
  theme_minimal() +
  theme(legend.title = element_blank()) +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank()) +
  theme(axis.text.y=element_blank(),
        axis.ticks.y=element_blank())
```

This code plots the xscores using ggplot, showing the first component on the x-axis and the second component on the y-axis. The points are colored based on the NMP group (High or Low). The plot is titled "PLS-DA Scores Plot", and the axes are labeled accordingly. The theme is set to minimal, and the legend title is removed for clarity. There doesn;tt seem to be a clear separation between the two groups.

```
# plot xscores using ggplot, plot comp 2 and comp 3
ggplot(important_xscores, aes(x = `Comp 2`, y = `Comp 3`, color = NMP )) +
  geom_point(size = 3) +
  labs(title = "PLS-DA Scores Plot", x = "Component 2", y = "Component 3") +
  theme_minimal() +
  theme(legend.title = element_blank()) +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank()) +
  theme(axis.text.y=element_blank(),
        axis.ticks.y=element_blank())
```

This code plots the xscores using ggplot, showing the second component on the x-axis and the third component on the y-axis. The points are colored based on the NMP group (High or Low). The plot is titled "PLS-DA Scores Plot", and the axes are labelled accordingly. The theme is set to minimal, and the legend title is removed for clarity. There doesn't seem to be a clear separation between the two groups, but it is better than the previous plot.

```r
# plot xscores using ggplot, plot comp 3 and comp 1
ggplot(important_xscores, aes(x = `Comp 1`, y = `Comp 3`, color = NMP )) +
  geom_point(size = 3) +
  labs(title = "PLS-DA Scores Plot", x = "Component 1" + variance_pc1, y = "Component 3" + variance_pc3
  theme_minimal() +
  theme(legend.title = element_blank()) +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank()) +
  theme(axis.text.y=element_blank(),
        axis.ticks.y=element_blank())
```

If we look at this Component score plot, we are comparing component 3 with component 1 and are looking for groups that are formed. It seems that there is a better separation between the two groups, but it is still not perfect. This is because there are some Low points in the high group. The points are colored based on the NMP group (High or Low), and the plot is titled "PLS-DA Scores Plot". The axes are labeled with the component names, and the theme is set to minimal. The legend title is removed for clarity.

5.1.1 Variance calculation

Concept: Take the eigenvalue from the model. Calculate per column the variance for the normal data and add this all up. Divide the eigenvalue from each component by the original data variance and multiply by 100%

5.1.2 Variance first try

```r
# Transpose the data_for_plsda dataframe.
t_data_for_plsda <- t(data_for_plsda)
# Convert the transposed matrix back into a dataframe for easier manipulation.
t_data_for_plsda <- as.data.frame(t_data_for_plsda)
```

Transpose the columns so they are samples and the rows are the variables. This is done to make it easier to calculate the variance for each sample.

```r
# Calculate the variance for each column (which are now samples) in the transposed dataframe.

test_variance <- apply(t_data_for_plsda, 2, var)
# Extract the eigenvalue for the first component from the PLS-DA model's Y-eigenvalues.
pc_1_eigenvalue <- model_plsda_corrected$yeigenvals[1]
pc_3_eigenvalue <- model_plsda_corrected$yeigenvals[3]
```

Take the eigenvalue for the first and third component from the PLS-DA model's Y-eigenvalues. This is done to calculate the percentage of variance explained by these components relative to the total variance.

```r
# Calculate the percentage of variance explained by PC3 relative to the sum of variances
variance_pc3 <- pc_3_eigenvalue / sum(test_variance) * 100
variance_pc1 <- pc_1_eigenvalue / sum(test_variance) * 100
```

The variance for each component is calculated by dividing the eigenvalue of the component by the total variance of the data. This gives the percentage of variance explained by each component. but it was so low i thought i made a mistake so i took another approach

5.1.3 Variance, second try.

```
# Calculate the total sum of all Y-eigenvalues from the corrected PLS-DA model.
total_variance <- sum(model_plsda_corrected$yeigenvals)

# Extract the eigenvalue for the first component from the PLS-DA model's Y-eigenvalues.

pc_1_eigenvalue_test <- model_plsda_corrected$yeigenvals[1]

# Extract the eigenvalue for the third component from the PLS-DA model's Y-eigenvalues.

pc_3_eigenvalue_test <- model_plsda_corrected$yeigenvals[3]
```

Calculate the total variance from the Y-eigenvalues of the PLS-DA model. This is done to get the total variance explained by all components in the model. The eigenvalues for the first and third components are extracted for further calculations.

```
# Calculate the percentage of Y-variance explained by PC1, rounded to 2 decimal places.
variance_pc1_test <- round(pc_1_eigenvalue / total_variance * 100, 2)
# Calculate the percentage of Y-variance explained by PC3, rounded to 2 decimal places.

variance_pc3_test <- round(pc_3_eigenvalue / total_variance * 100, 2)
```

Calculate the percentage of variance explained by the first and third components relative to the total variance. This gives a clearer understanding of how much each component contributes to the overall model.

** Plot with variance added **

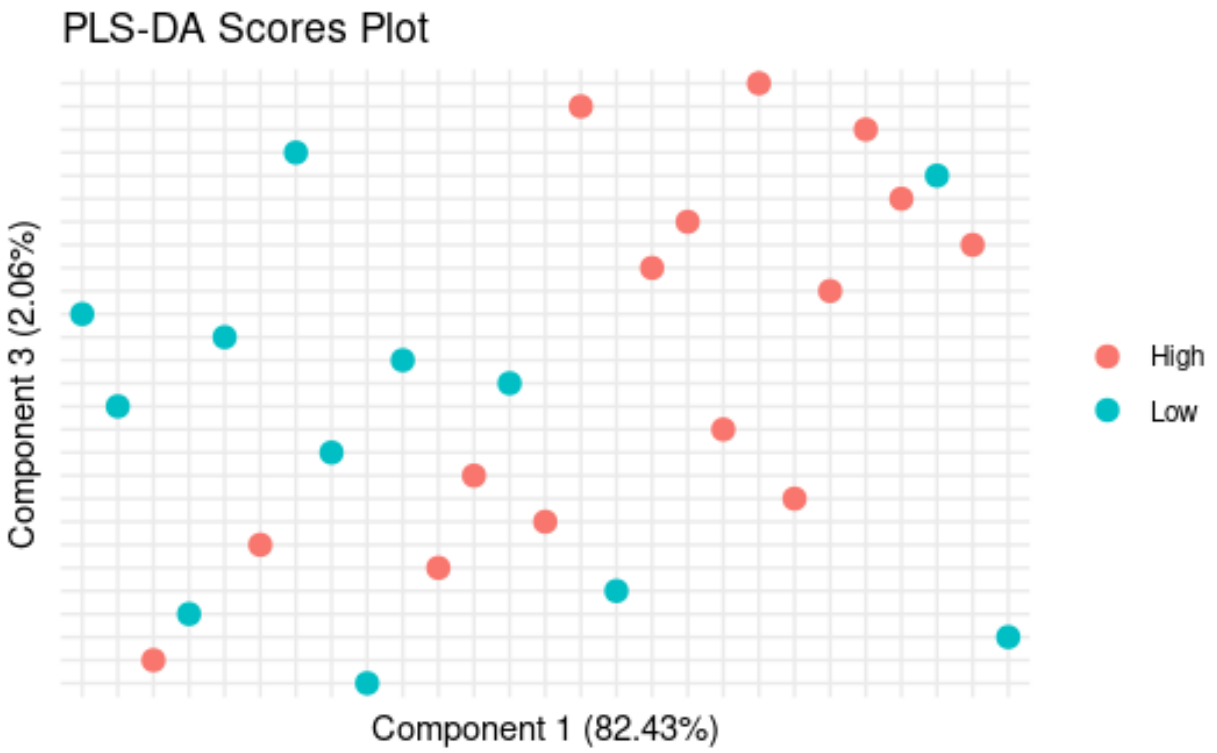
```
# Convert the calculated variance percentages to character strings. This is necessary for concatenating


as.character(variance_pc3_test)
as.character(variance_pc1_test)

# Create a scatter plot of PLS-DA scores using ggplot2.
ggplot(important_xscores, aes(x = `Comp 1`, y = `Comp 3`, color = NMP )) +
  geom_point(size = 3) +
  labs(
    title = "PLS-DA Scores Plot",
    x = paste0("Component 1 (", variance_pc1_test, "%)"),
    y = paste0("Component 3 (", variance_pc3_test, "%)")
  ) +
    # Apply a minimal theme for a clean look.
  theme_minimal() +
  theme(legend.title = element_blank()) +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank()) +
  theme(axis.text.y=element_blank(),
        axis.ticks.y=element_blank())
```

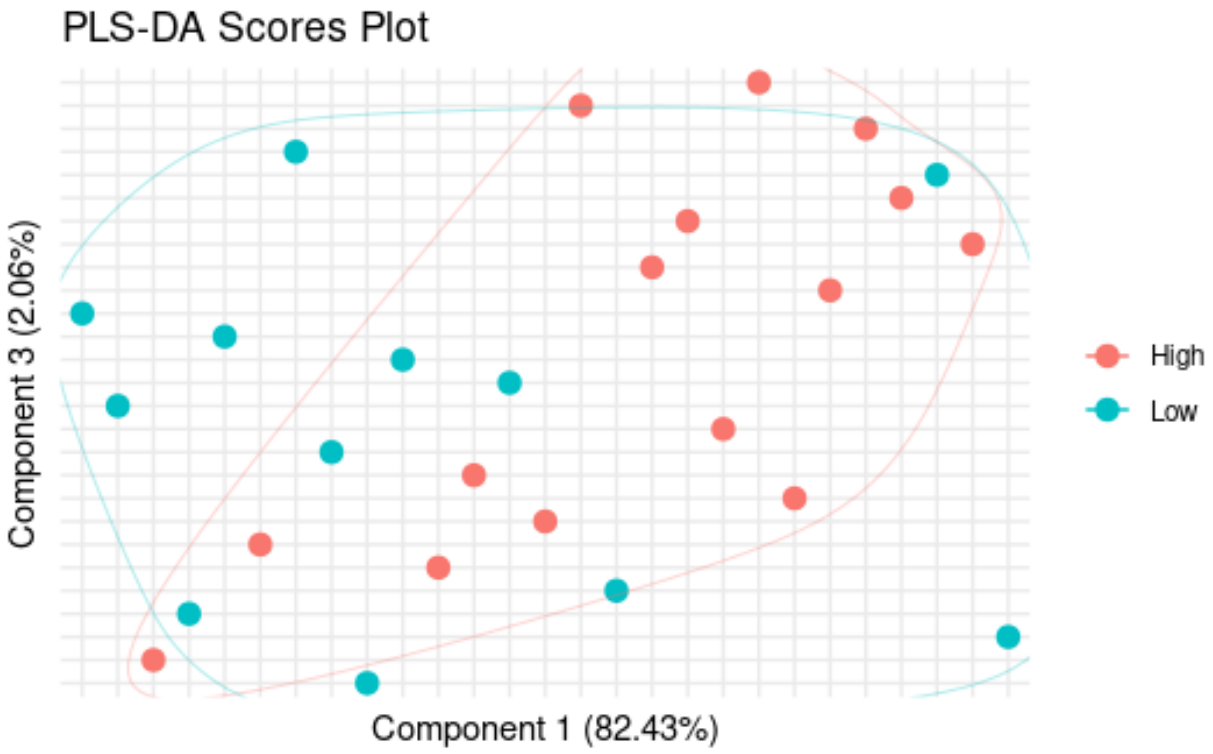
The code above creates a scatter plot of the PLS-DA scores using ggplot2. The x-axis represents Component 1, and the y-axis represents Component 3. The points are colored based on the NMP group (High or Low).

The plot is titled “PLS-DA Scores Plot”, and the axes are labeled with the percentage of variance explained by each component. The theme is set to minimal for a clean look, and the legend title is removed for clarity.




```r
# Same plot as before
ggplot(important_xscores, aes(x = `Comp 1`, y = `Comp 3`, color = NMP )) +
  geom_point(size = 3) +
  labs(
    title = "PLS-DA Scores Plot",
    x = paste0("Component 1 (", variance_pc1_test, "%)"),
    y = paste0("Component 3 (", variance_pc3_test, "%)")
  ) +
  # This is supossed to add a line arround each group to show seperation
  geom_encircle(aes(group = NMP), expand = 0.05, alpha = 0.3) +
  theme_minimal() +
  theme(legend.title = element_blank()) +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank()) +
  theme(axis.text.y=element_blank(),
        axis.ticks.y=element_blank())
```

The code above creates a scatter plot of the PLS-DA scores using ggplot2, similar to the previous plot. However, it adds a line around each group (High and Low NMP) to show separation between the groups. The `geom_encircle` function is used to create the encircling lines, with an expansion factor of 0.05 and an alpha value of 0.3 for transparency. The axes are labeled with the percentage of variance explained by each component, and the theme is set to minimal for a clean look. however this did not show the correct split of groups at all/



5.2 Biological background for the PLS-DA

COPZ1

Our PLS-DA analysis identified COPZ1 as an important variable differentiating between the biliary viability groups. COPZ1 was also found to be downregulated in high biliary viability livers. COPZ1 encodes for a subunit that makes up the cytoplasmic coatomer protein complex. The coatomer protein complex is involved in autophagy and intracellular protein trafficking (NCBI, n.d.). The primary transport it facilitates is from the Golgi complex to the endoplasmic reticulum, also known as retrograde transport. This is important to maintain homeostasis by removing ageing proteins and organelles for degradation and recycling of components. (Kucheryavskiy, 2020) This protein is a prognostic marker in liver hepatocellular carcinoma. Other info about this protein is the essential role it plays in maintaining the survival of some types of tumours (Hong et al., 2023).

MMP7

The matrix metalloproteinase MMP7 was highlighted in our PLS-DA. MMP7 is part of a family of enzymes responsible for the breakdown of the extracellular matrix. The blood flow through the liver could be messed up, and the blood vessels could be worn down, causing a lower biliary viability (Lambert et al., 2005).

SLC6A19

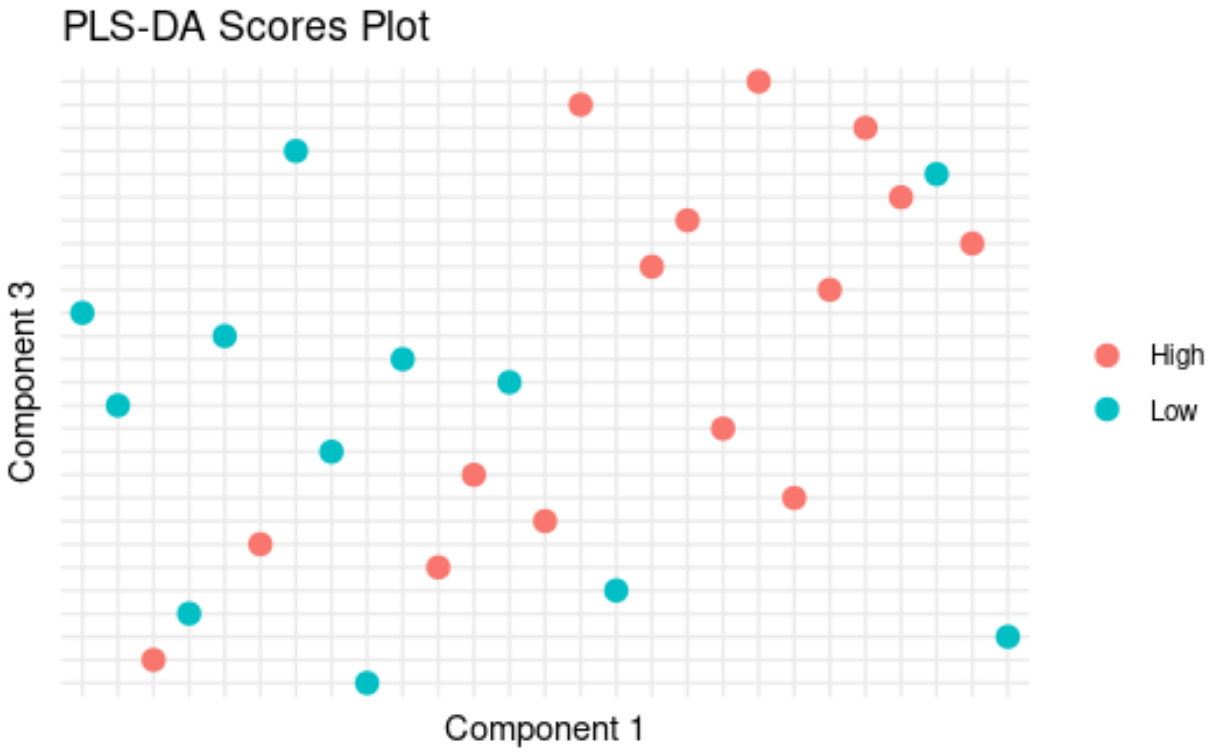
SLC6A19, encoding for the system B transmembrane protein, was also identified in our analysis. This protein actively transports neutral amino acids across the apical membrane of epithelial cells (SLC6A19 Solute Carrier Family 6 Member 19 [Homo Sapiens (Human)] - Gene - NCBI, z.d.). This gene and protein don't seem to have any relation to biliary viability and mainly focus on transporting amino acids in organs such as the kidneys and intestine (Bröer, 2008).

MUC1

MUC1 was also found in our PLS-DA analysis, where it was found to be a significant variable that allows for differentiating between biliary viability groups. MUC1, the transmembrane glycoprotein mucin 1, is a mucin

family member that has different functions in normal and cancer cells. This is caused by its structure and biochemical properties. Mucin1 can act as a lubricant, moisturiser, and physical barrier in normal cells. In cancer cells, mucin 1 gets overexpressed and often faces aberrant glycosylation (Chen et al., 2021). MUC1 is also expressed in the liver and is key for maintaining cellular functions, specifically for the epithelial surfaces (Kasprzak & Adamek, 2019). This gene could help in the regeneration of liver cells and thereby increase liver viability.

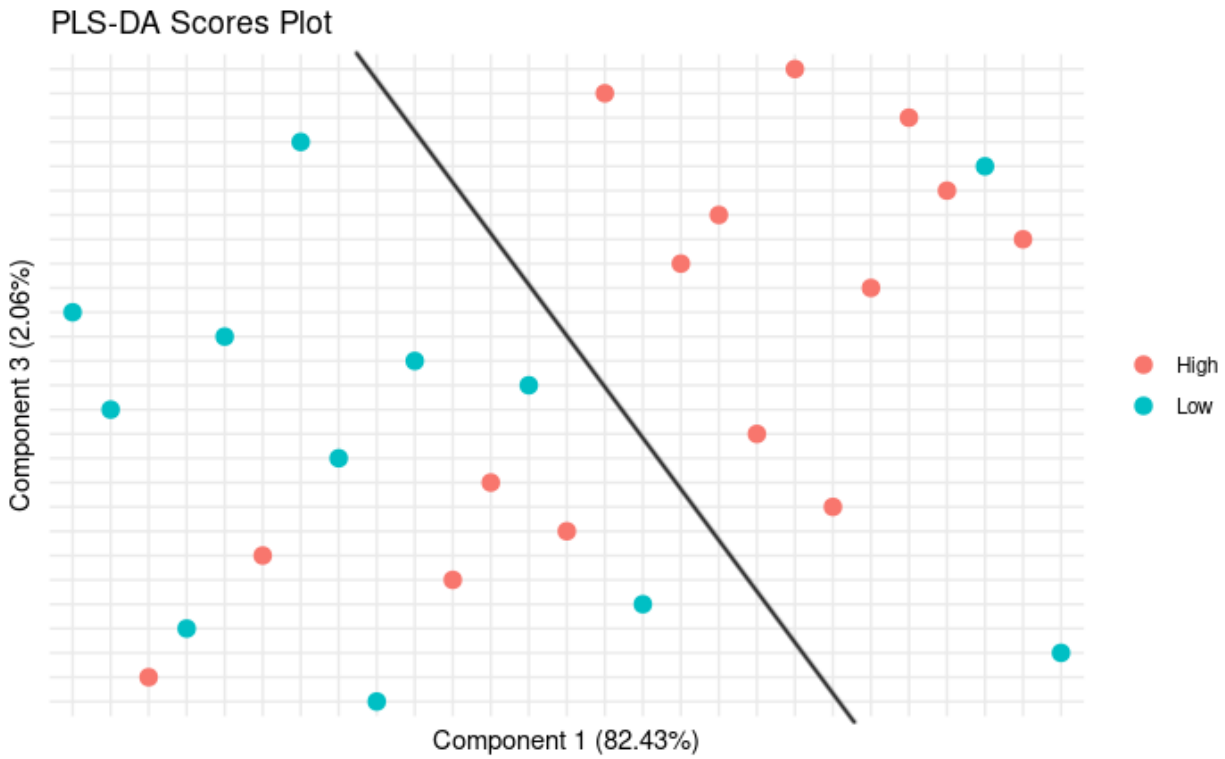
These were my original findings from what i could see from the weights plot this has later been updated when i made a more clear loadings plot



PLS-DA Score plot component 1 – component 3
PLS-DA score plot performed on proteomic samples from 150 min after starting NMP.
Showing how the samples are laid out over components 1 & 3. Viability is shown with colour. There is a clear divide visible in the middle.

PLS-DA was performed on the normalised, imputed proteomics data. This data was then split into two groups: high biliary viability and low biliary viability. Figure 8 shows a comparison between the 3rd component, which contains 4.62% of the info, and Component 1, which contains 21.9% of the info. The mentioned info is the amount of each component that relates to the original data file. The reason for this comparison is that component 1, component 2 & component 2 component 3 showed nothing significant and no groups; therefore, there were no clear results from these comparisons.

5.2.1 Final PLS-DA score plot



Now with a clear line we can see how the groups are separated. The high biliary viability group is on the left side of the plot, and the low biliary viability group is on the right side of the plot. The line is drawn to show the separation between the two groups. The points are colored based on the NMP group (High or Low), and the axes are labeled with the percentage of variance explained by each component. The two outliers of the low group could be interesting to research in another project. Without making it too complicated, I downloaded the plot and I drew the line manually to what my interpretation was. Without changing any of the datapoints.

5.3 Loadings barplot component 1

```
# take the loadings from the model
weight_data <- model_plsda_corrected$xloadings

# transform from a matrix to a workable datagrame
weight_data <- as.data.frame(weight_data)
```

The loadings from the model are taken and transformed from a matrix to a data frame. This is done to make it easier to work with the data and to plot the loadings.

```
#select all weights from component 1
important_weights <- weight_data %>%
  select(`Comp 1`)
```

This code selects all the weights from component 1 and puts them in a new data frame called `important_weights`. This is done to focus on the loadings that are most important for component 1, which is the component that contains the most variance in the data.

```r
# filter for the loadings that have atleast and importance above 8 because this shows only the most imp
filtered_weights <- important_weights %>%
  filter(important_weights$`Comp 1` > 8)
```

I have decided that any loadings above 8 are important enough to be included in the plot. This is done to focus on the most significant proteins that contribute to component 1. Other wise the plot would again be unreadable like the other weight plot was.

```r
# filter for the loadings that work against the component
negative_weights <- important_weights %>%
  filter(important_weights$`Comp 1` < -8)
```

This code filters for the loadings that work against component 1, meaning they have a negative loading value. These proteins are also important to include in the plot, as they contribute to the overall understanding of the component's significance.

```r
# combine both groups to plot
filtered_weights <- rbind(filtered_weights, negative_weights)
```

This code combines the filtered weights for both positive and negative loadings into a single data frame called `filtered_weights`. This is done to create a comprehensive plot that shows both the proteins that contribute positively and negatively to component 1.

```r
# using ggplot to make a barplot that shows the loadings
ggplot(filtered_weights, aes(x=`Comp 1`, y=rownames(filtered_weights))) +
  labs(
    title = "PLS-DA loadings Plot - Component 1",
    x = "Contribution to Component 1",
    y = "Protein Name"
  ) +
  geom_bar(stat = "identity")
```

This code uses ggplot to create a bar plot that shows the loadings for component 1. The x-axis represents the contribution to component 1, and the y-axis represents the protein names. The `geom_bar` function is used to create the bars in the plot. This plot is very primentive and with some styling it could better the readability for the reader.

```r
# using some styling to make the ggplot more clear
ggplot(filtered_weights, aes(x = `Comp 1`, y = reorder(rownames(filtered_weights), `Comp 1`))) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(
    x = "Contribution to Component 1",
    y = "Protein Name"
  ) +
  theme_minimal() +
  theme(
    axis.text.y = element_text(size = 7),    # Smaller y axis text to fit more names
  ) +
  geom_text(aes(label = round(`Comp 1`, 2)), # Add contribution labels on bars
            hjust = -0.1, size = 2)
```

This code uses ggplot to create a bar plot that shows the loadings for component 1 with some styling. The x-axis represents the contribution to component 1, and the y-axis represents the protein names. The bars

are filled with a steel blue color, and the theme is set to minimal for a clean look. The y-axis text size is reduced to fit more names, and contribution labels are added on the bars for clarity.

5.3.1 Loadings for component 3

```
# Now subsetting component 3 to make the loadings plot
important_weights_comp_3 <- weight_data %>%
  select(`Comp 3`)
```

This code selects all the weights from component 3 and puts them in a new data frame called `important_weights_comp_3`. This is done to focus on the loadings that are most important for component 3, which is the component that contains some variance in the data.

```
# filter for the loadings that have atleast and importance above 8 because this shows only the most imp
filtered_weights_comp_3 <- important_weights_comp_3 %>%
  filter(important_weights_comp_3$`Comp 3` > 8)
```

This code filters for the loadings that have an importance above 8, meaning they contribute significantly to component 3. This is done to focus on the most significant proteins that contribute to component 3.

```
# filter for the loadings that work against the component using also the 8 as minimum
negative_weights_comp_3 <- important_weights_comp_3 %>%
  filter(important_weights_comp_3$`Comp 3` < -8)
```

This code filters for the loadings that work against component 3, meaning they have a negative loading value and an importance below -8. These proteins are also important to include in the plot, as they contribute to the overall understanding of the component's significance.

```
filtered_weights_comp_3 <- rbind(filtered_weights_comp_3, negative_weights_comp_3)
```

This code combines the filtered weights for both positive and negative loadings into a single data frame called `filtered_weights_comp_3`. This is done to create a comprehensive plot that shows both the proteins that contribute positively and negatively to component 3.

```
# using ggplot to make a barplot that shows the loadings for comp 3

ggplot(filtered_weights_comp_3, aes(x=`Comp 3`, y=rownames(filtered_weights_comp_3))) +
  labs(
    x = "Contribution to Component 1",
    y = "Protein Name"
  ) +
  geom_bar(stat = "identity")
```

This code uses ggplot to create a bar plot that shows the loadings for component 3. The x-axis represents the contribution to component 3, and the y-axis represents the protein names. The `geom_bar` function is used to create the bars in the plot. This plot is very primitive and with some styling it could better the readability for the reader. as with the comp 1 plot.

```r
# using some styling to make the ggplot more clear and easier to read.
ggplot(filtered_weights_comp_3, aes(x = `Comp 3`, y = reorder(rownames(filtered_weights_comp_3), `Comp 3
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(
    x = "Contribution to Component 3",
    y = "Protein Name"
  ) +
  theme_minimal() +
  theme(
    axis.text.y = element_text(size = 7)) +          # Smaller y axis text to fit more names
  geom_text(aes(label = round(`Comp 3`, 2)),       # Add contribution labels on bars
            hjust = -0.1, size = 2)
```

This code uses ggplot to create a bar plot that shows the loadings for component 3 with some styling. The x-axis represents the contribution to component 3, and the y-axis represents the protein names. The bars are filled with a steel blue color, and the theme is set to minimal for a clean look. The y-axis text size is reduced to fit more names, and contribution labels are added on the bars for clarity.

5.4 Biological background PLS-DA loadings

PLS-DA loadings revealed two important liver proteins. The ABCC2, also known as the Multidrug Resistance Protein 2, is involved in the transport of substances out of cells. It plays a crucial role in drug disposition by mediating ATP-dependent efflux of a wide range of compounds, including drug metabolites, from cells into bile or urine. (MedlinePlus Genetics (2018)). ABCC2's transport of bilirubin out of the liver cells and into the bile is particularly relevant. Bilirubin is a substance that is produced during the breakdown of old red blood cells and is frequently used to check the health of a patient's liver; therefore, the identification of ABCC2 as an important potential biomarker for biliary viability. (MedlinePlus (2025)). Another crucial protein identified was ABCB11, also known as the Bile Salt Export Pump (BSEP). BSEP is responsible for the ATP-dependent transport of primary hydrophobic bile salts, such as taurine and glycine-conjugated cholic acid, from hepatocytes into the bile (UniProt Consortium (2025)). This process is fundamental for hepatic bile acid homeostasis and, consequently, for lipid homeostasis through the regulation of biliary lipid secretion. Dysregulation or dysfunction of BSEP can lead to severe cholestatic liver injury (Byrne et al. (2002)), as observed in conditions like Progressive Familial Intrahepatic Cholestasis Type 2 (PFIC2), where biallelic mutations in ABCB11 cause an accumulation of hepatic bile acids and subsequent hepatotoxicity, ultimately leading to liver failure (Gruget, Reddy, and Moore (2025)). Given BSEP's critical role in bile acid transport, it's an important protein for evaluating biliary viability and, therefore, a significant biomarker.

Used refrences

Boyer, James L. 2013. "Bile Formation and Secretion." *Comprehensive Physiology* 3 (3): 1035–78. https://doi.org/10.1002/cphy.c120027.

Byrne, Jane A., Sandra S. Strautnieks, Giorgina Mieli-Vergani, Christopher F. Higgins, Kenneth J. Linton, and Richard J. Thompson. 2002. "The Human Bile Salt Export Pump: Characterization of Substrate Specificity and Identification of Inhibitors." *Gastroenterology* 123 (5): 1649–58. https://doi.org/10.1053/gast.2002.36591.

Gruget, Camille, Bhargav G. Reddy, and James M. Moore. 2025. "A Structural and Mechanistic Model for BSEP Dysfunction in PFIC2 Cholestatic Disease." *Communications Biology* 8: 531. https://doi.org/10.1038/s42003-025-07908-0.

MedlinePlus. 2025. "Bilirubin Blood Test." U.S. National Library of Medicine; https://medlineplus.gov/lab-tests/bilirubin-blood-test/.

MedlinePlus Genetics. 2018. "ABCC2 Gene — ATP-Binding Cassette Subfamily c Member 2." U.S. National Library of Medicine; https://medlineplus.gov/genetics/gene/abcc2/#conditions.

Thorne, Adam M., Justina C. Wolters, Bianca Lascaris, Silke B. Bodewes, Veerle A. Lantinga, Otto B. van Leeuwen, Iris E. M. de Jong, et al. 2023. "Bile Proteome Reveals Biliary Regeneration During Normothermic Preservation of Human Donor Livers." *Nature Communications* 14 (1): 7880. https://doi.org/10.1038/s41467-023-43368-y.

UniProt Consortium. 2025. "UniProtKB Entry for O95342 (ABCB11, Bile Salt Export Pump)." UniProt; https://www.uniprot.org/uniprotkb/O95342/entry.