Epicardial Cardiomyocyte Interactome 2020. Analysis and code.

Vincent Knight-Schrijver

23 June 2020

Introduction

Analysis rational

When epicardial cells are included with an injected graft of cardiomyocytes there is a significant increase in myocardial function. To understand more about how this epicardial augmentation occurs, it may be key to explore their secretome. This bioinformatics analysis considers that the epicardial cells secrete some form of soluble factors that are affecting the cardiomyocytes. Additionally we are keen to explore which cardiomyocyte receptors actually respond to the epicardial secreted factors. Thus we create a unidirectional interactome between both cell types.

Aims

The aim of this analysis / study is to identify which secreted epicardial factors that may be responsible for the angiogenic function of epicardial cell. To achieve this aim we will 1) generate a set of epicardial secreted factors using in-house RNA-seq data from in-house human embryonic stem cell derived epicardium (hESC-EPI), 2) generate a set of cardiomyocyte membrane-bound factors or receptors using an external RNA-seq dataset of hESC-cardiomyocytes (hESC-CM) from publicly available sources, 3) pair the epicardial secreted factors to the cardiomyoctyes membrane-bound factors from the output of 1 and 2, 4) construct a first-pass high-confidence reference interactome for hESC-epicardial to hESC-cardiomyocyte crosstalk using protein-interaction mapping databases, and 5) functionally annotate the paired interactome using gene ontology enrichment or other relevant annotation method. The end result that we aim to produce are a group of potential secreted factors from epicardial cells which may be responsible for myocardial regeneration.

Method

The approach will be to run differential expression analyses between hESC-EPI cells and their negative control, neural crest (NC) cells. We would then filter to keep genes with a putative secreted protein product, retaining only differentially upregulated genes that encode secreted proteins, the hESC-EPI secretome. Then we repeat this process for the day 30 hESC-derived cardiomyocytes (CMs) and their negative control, the day 0 H9 embryonic stem cells, creating the hESC-CM membranome. Data were taken from our in-house experiments for epicardium (1) and sourced externally for the cardiomyocytes (2).

1. GSE85331 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE85331)
2. GSE122714 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE122714)

The Analysis

Before we begin we must address the R environment. The very first thing we need to do is load the relevant functions and packages. Fortunately, I keep a home-cooked functions list in a separate directory alongside a function loader. I have not checked any dependencies upon other pre-existing packages. Some will require perhaps unusual packages. However most code is written with base R in mind. It may be best (for your own sanity) not to peruse all functions as most of them are barely annotated.

```
# Packages to load
 library(DESeq2)
 library(tximport)
 library(wordcloud)

# functions directory
  func.dir="functions"

# Function to load functions
  source(file.path(func.dir, "load_functions.r"))

# run the load functions function
  load.functions(func.dir)
```

Epicardial cell data

Raw counts for the three epicardial and three neural crest samples were obtained from the FASTQ files. Mapping and preprocessing wer originally carried out in the paper https://www.nature.com/articles/s41587-019-0197-9.pdf. For our use here, this dataset composed of raw and uncorrected reads is contained within the data matrix **bulk7_counts.csv**.

```
# Begin generally with importing the data... easy enough
  bulk7 <- read.csv("data/bulk7/bulk7_counts.csv", head=T, row.names=1, stringsAsFactors=F)

# grab only the EPI and NC columns...
  EPI.counts <- bulk7[,grep("EPI.$|NC", colnames(bulk7))]
```

Cardiomyocytes data

Raw data for hESC-CM in the form of FASTQs were accessed from the gene expression omnibus with the accession ID GSE85331. The samples that were downloaded consisted of GSM2264858, GSM2264859, GSM2264864 and GSM2264865. Initially, FASTQC was run over the samples and the paired read files were trimmed and pre-processed using FASTP. A subsequent set FASTQC reports were generated for trimmed files. Then a transcript-level estimation of the counts was created using the pseudoalignment tool Kallisto. A transcript cDNA index of the human genome (GRch38 release 98) was created from the FASTA file (acquired from ftp://ftp.ensembl.org/pub/release-98/fasta/homo_sapiens/cdna/) and built with kallisto. This is all written here within R here using the *system* function. This may take a while so do not run uneccessarily.

```
## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## FASTQC
## fastqc the files to check for quality.
system("fastqc data/raw/SRR4011*")

## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## FASTP
## Trimming before alignment
system("mkdir data/raw/trimmed")
```

```
system("fastp
    -i data/raw/SRR4011898_H9_0_1_1.fastq.gz
    -I data/raw/SRR4011898_H9_0_1_2.fastq.gz
    -o data/raw/trimmed/SRR4011898_H9_0_1_1_trimmed.fastq.gz
    -O data/raw/trimmed/SRR4011898_H9_0_1_2_trimmed.fastq.gz
    -w 8
    -h data/raw/trimmed/fastp_report_H9_0_1
  ")
system("fastp
    -i data/raw/SRR4011899_H9_0_2_1.fastq.gz
    -I data/raw/SRR4011899_H9_0_2_2.fastq.gz
    -o data/raw/trimmed/SRR4011899_H9_0_2_1_trimmed.fastq.gz
    -O data/raw/trimmed/SRR4011899_H9_0_2_2_trimmed.fastq.gz
    -w 8
    -h data/raw/trimmed/fastp_report_H9_0_2
  ")
system("fastp
    -i data/raw/SRR4011904_H9_30_1_1.fastq.gz
    -I data/raw/SRR4011904_H9_30_1_2.fastq.gz
    -o data/raw/trimmed/SRR4011904_H9_30_1_1_trimmed.fastq.gz
    -O data/raw/trimmed/SRR4011904_H9_30_1_2_trimmed.fastq.gz
    -w 8
    -h data/raw/trimmed/fastp_report_H9_30_1
  ")
system("fastp
    -i data/raw/SRR4011905_H9_30_2_1.fastq.gz
    -I data/raw/SRR4011905_H9_30_2_2.fastq.gz
    -o data/raw/trimmed/SRR4011905_H9_30_2_1_trimmed.fastq.gz
    -O data/raw/trimmed/SRR4011905_H9_30_2_2_trimmed.fastq.gz
    -w 8 -h data/raw/trimmed/fastp_report_H9_30_2
  ")

## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## FASTQC
## fastqc the trimmed files to check for further quality
system("fastqc data/raw/trimmed/SRR4011*")

## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## KALLISTO
## Place the cDNA into the data directory
system("kallisto index -i data/HS_GRCh38.cdna.all.test.idx
  ~/Documents/genomes/human_hg38/Homo_sapiens.GRCh38.cdna.all.fa.gz")

# Use this index and perform pseudoalignment of our trimmed
# sequenced fasta files...
system("kallisto quant
    -i data/HS_GRCh38.cdna.all.idx -o data/kallisto_out/d0_1/trimmed
    -b 100
    -t 12
    data/raw/trimmed/SRR4011898_H9_0_1_1_trimmed.fastq.gz
    data/raw/trimmed/SRR4011898_H9_0_1_2_trimmed.fastq.gz
  ")
system("kallisto quant
    -i data/HS_GRCh38.cdna.all.idx
    -o data/kallisto_out/d0_2/trimmed
```

```
    -b 100
    -t 12
    data/raw/trimmed/SRR4011899_H9_0_2_1_trimmed.fastq.gz
    data/raw/trimmed/SRR4011899_H9_0_2_2_trimmed.fastq.gz
  ")
system("kallisto quant
    -i data/HS_GRCh38.cdna.all.idx
    -o data/kallisto_out/d30_1/trimmed
    -b 100
    -t 12
    data/raw/trimmed/SRR4011904_H9_30_1_1_trimmed.fastq.gz
    data/raw/trimmed/SRR4011904_H9_30_1_2_trimmed.fastq.gz
  ")
system("kallisto quant
    -i data/HS_GRCh38.cdna.all.idx
    -o data/kallisto_out/d30_2/trimmed
    -b 100
    -t 12
    data/raw/trimmed/SRR4011905_H9_30_2_1_trimmed.fastq.gz
    data/raw/trimmed/SRR4011905_H9_30_2_2_trimmed.fastq.gz
  ")
```

After generating the transcript count estimates we need to import the data into R for our DESeq2 workflow. We need to first create transcript to gene map files so that our transcript IDs will be translated into gene IDs properly. The DESeq2 vignette illustrates this section nicely. To generate a map, I will parse the FASTA file that was used to make the kmer index. When we import files from kallisto we are interested in turning our estimated transcript counts into gene-level counts. That means we must collect our estimated transcript IDs into single genes. For context, one gene can have several transcripts as exons or isoforms. There are several methods that we can do this:

1. FASTA parsing
2. GTF parsing from a known genome GTF file...
3. Map transcript IDs using biomaRt - the ensembl R tool for annotations.

We opted to write a quick FASTA parser to grab the gene name information from the cDNA information file as this should be the best representation of our transcript index.

```
# We have indexed our list of transcripts using the compressed fasta.
# Let's find this and parse it.

# Our FASTA location
  FASTA.loc <- "~/Documents/genomes/human_hg38/Homo_sapiens.GRCh38.cdna.all.fa"
# Using a lazy approach to reading this FASTA and parsing by FASTA entry character: ">"
  FASTA <- read.csv(FASTA.loc, sep=">", stringsAsFactors=F, skip=5, head=F)

# calculate fields with the transcript ID
  tx.m <- nchar(FASTA[,2])
# subset by those fields with a transcript ID
  FASTA <- FASTA[which(tx.m > 0),2]

# REGEX character substitutions across the field gives us the IDs we want
  tx.id <- gsub(" cdna.*$", "", FASTA)
  gene.id <- gsub("^.*gene:| gene_bio.*$", "", FASTA)
  gene.symbol <- gsub("^.*gene_symbol:| description.*$", "", FASTA)

  tx2gene <- data.frame(tx.id, gene.id, gene.symbol, stringsAsFactors=F)
```

```
# And there we have it! our tx2gene file used for mapping the transcripts
# counted in kallisto into genes.
```

With our estimates created and our transcript to gene file available, the next step will be to import the data using the functions from the DESeq2 package. Here we import with the *tximport* function and generate a matrix called **CM.counts**.

```
# The next step is to read our kallisto estimates in using our map csv. This is
# part of the recommended workflow from DESeq2 vignette.

# kallisto output directory:
  files <- list.files("data/kallisto_out", include.dirs=F,
              recursive=T, full=T, pattern=".h5$")[
                grep("trimmed", list.files("data/kallisto_out", include.dirs=F,
                 recursive=T, full=T, pattern=".h5$"),invert=T
                )
              ]

# rename the files:
  names(files) <- gsub("(^.*/.*out/|/.*$)", "", files)
# Import using tximport (which applies a counts offset to adjust for
#  transcript estimates or something)
  txi.kallisto <- tximport(files, type = "kallisto",
                           countsFromAbundance="lengthScaledTPM",
                           txOut = F,
                           tx2gene = tx2gene[,c(1,3)]
                           )

# make a nice new dataframe for ease
  CM.counts <- txi.kallisto$counts

# For differential expression analysis, most models assume that we are dealing
# with counts data. Thus we should integerise our dataset.
  CM.counts <- round(CM.counts)
```

Technical validation

This section of the report code generates the technical validation information. We cover a few sections here.

1. Depth saturation
2. Phenotype validation
3. Replicate correlation

depth saturation

We begin with examining the depth saturation of the epicardial dataset, creating many sub-sampled counts at an increasing number using a hypergeometric distribution.

```
# Gene / read depth saturation curves / analysis

# Utilise a hypergeometric distribution to subsample our data.
  library(extraDistr)
```

```
## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
# EPI dataset
  # The matrix we want to analyse
  EPI.counts <- EPI.counts[!rownames(EPI.counts) == "null",]
  counts.data.matrix <- as.matrix(EPI.counts)
  n2sample <- round(seq.log.basic(10,max(colSums(counts.data.matrix)), 40))
  set.seed(666)

  res.epi = lapply(1:ncol(counts.data.matrix), function(n){
    z = data.frame(n2sample)
    for(i in 1:length(n2sample)){

    # the number of reads to sample
      k <- n2sample[i]

    # the number of samples to create for each sample
      m = 1

    # Generate our sub-sampled matrix
      x <- as.numeric(rmvhyper(m,
                      t(as.matrix(counts.data.matrix[,n])),
                      min(k,sum(counts.data.matrix[,n]))
                      ))

      y  = x > 0

      z[i,1:2] = c(min(k,sum(counts.data.matrix[,n])),sum(y))
    }
    z
  })


## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
# CM dataset
  # The matrix we want to analyse
  counts.data.matrix <- as.matrix(CM.counts)

  n2sample <- round(seq.log.basic(10,max(colSums(counts.data.matrix)), 40))
  set.seed(666)

  res.cm = lapply(1:ncol(counts.data.matrix), function(n){
    z = data.frame(n2sample)
    for(i in 1:length(n2sample)){

    # the number of reads to sample
      k <- n2sample[i]

    # the number of samples to create for each sample
      m = 1

    # Generate our sub-sampled matrix
```

```
      x <- as.numeric(rmvhyper(m,
                      t(as.matrix(counts.data.matrix[,n])),
                      min(k,sum(counts.data.matrix[,n]))
                      ))

      y  = x > 0

      z[i,1:2] = c(min(k,sum(counts.data.matrix[,n])),sum(y))
    }
    z
  })


## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
# here we make a combined figure for all samples
  gene.y <- sapply(1:length(res.epi), function(i) {res.epi[[i]][,2]})
  depth.x <- sapply(1:length(res.epi), function(i) {res.epi[[i]][,1]})
  x = depth.x
  y = gene.y

  plot(as.matrix(x), as.matrix(y),
    col=rep(c(Discrete[6], Discrete[6], Discrete[6],
        Discrete[9], Discrete[9], Discrete[9]
        ), each=40
      ),
    xlab="sampled aligned reads (simulated depth)",
    ylab=expression(paste("Genes counted (with ">=" 1 reads)")), cex=1,
    ylim=c(0,30000), xlim=c(0, 3.2e7), pch=5
  )

  lines(x[,1], y[,1], col=Discrete[6])
  lines(x[,2], y[,2], col=Discrete[6], lty=2)
  lines(x[,3], y[,3], col=Discrete[6], lty=3)

  lines(x[,4], y[,4], col=Discrete[9])
  lines(x[,5], y[,5], col=Discrete[9], lty=2)
  lines(x[,6], y[,6], col=Discrete[9], lty=3)
  text(tail(x[,1],1)*1.1, tail(y[,1],1)+800, c("EPI"), cex=1, col=Discrete[6])
  text(tail(x[,5],1), tail(y[,5],1)-1000, c("NC"), cex=1, col=Discrete[9])

  gene.y <- sapply(1:length(res.cm), function(i) {res.cm[[i]][,2]})
  depth.x <- sapply(1:length(res.cm), function(i) {res.cm[[i]][,1]})
  x = depth.x
  y = gene.y
  points(as.matrix(x), as.matrix(y),
    col=rep(c(Discrete[9], Discrete[9], Discrete[7], Discrete[7]),each=40),
  pch=1, cex=1)

  lines(x[,1], y[,1], col=Discrete[9])
  lines(x[,2], y[,2], col=Discrete[9], lty=2)
  lines(x[,3], y[,3], col=Discrete[7])
  lines(x[,4], y[,4], col=Discrete[7], lty=2)
  text(tail(x[,1],1), tail(y[,1],1)-1000, c("H9"), cex=1, col=Discrete[9])
```

```
  text(tail(x[,4],1), tail(y[,4],1)-1000, c("CM"), cex=1, col=Discrete[7])

  abline(v = 3e7, lty=2)
  abline(v = 2e7, lty=3)

  mtext("A", 3, adj = 0)
```

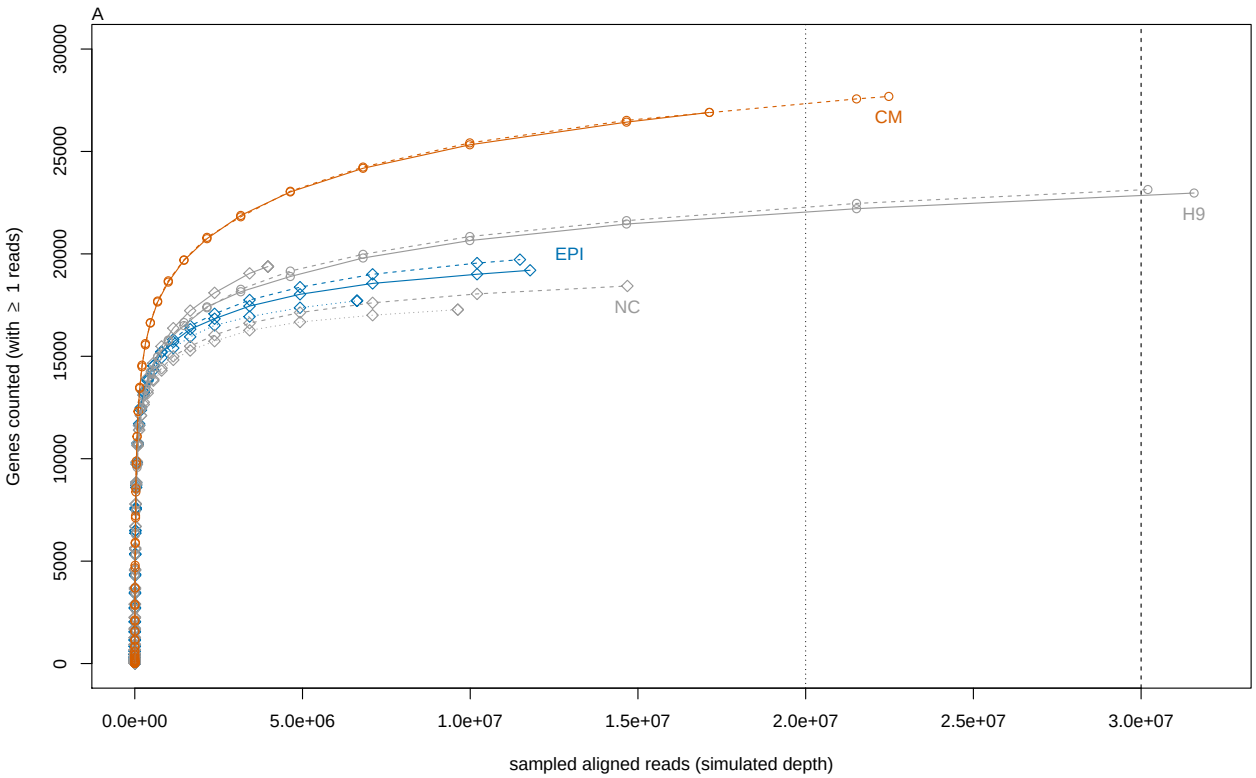

Figure 1: Expressed gene detection saturation curves.

Phenotype validation

The next step of technical validation is to ensure we are examining the correct cell types. We would not want to make an interactome of the wrong tissues after all! To do this We defined a set of putative markers, as can be readily found in scientific literature.

```
# Key markers
  CM.markers <- c(
    "MYH6",
    "MYH7",
    "MYL2",
    "MYL7"
  )

  EPI.markers <- c(
    "WT1",
    "TCF21",
    "BNC1",
    "ALDH1A2"
```

```
  )

  NC.markers <- c(
    "TFAP2A",
    "SOX9"
   )

  H9.markers <- c(
    "POU5F1",
    "SOX2"
  )
```

We are going to deal with a normalised value of expression. As such we calculate the counts per million values for the hESC-EPI dataset and convert the counts data from the output of the Kallisto pseudoalignment for the hESC-CM data into counts per million.

```
# EPI DATASET
# Create a very quick size factor calculation...
# Counts per million
  epi.sf <- colSums(EPI.counts+1)/1e6

# and normalise...
  EPI.genes <- t(t(EPI.counts+1)/epi.sf)
  eml <- length(EPI.markers)

# subset by the markers for quicker computation
  EPI.genes = EPI.genes[EPI.markers,]

# Create and format a results table for plotting
  EPI.genes.df <- data.frame(
      "log.cpm"=log(as.vector(EPI.genes)+1),
      "gene"=rep(rownames(EPI.genes), length(EPI.genes[1,])),
      "cell.type"=rep(c("EPI", "EPI", "EPI", "NC", "NC", "NC"), each = length(EPI.genes[,1])),
      "group"=paste(rep(rownames(EPI.genes), length(EPI.genes[1,])),
      rep(c("EPI", "EPI", "EPI", "NC", "NC", "NC"),
      each = length(EPI.genes[,1])),sep=".")
    )
  EPI.genes.df <- EPI.genes.df[order(EPI.genes.df$group),]

# NC gene expression
# normalise...
  NC.genes <- t(t(EPI.counts+1)/epi.sf)
  nml <- length(NC.markers)

# subset by NC markers
  NC.genes = NC.genes[NC.markers,]

# Create and format a results table for plotting
  NC.genes.df <- data.frame(
      "log.cpm"=log(as.vector(NC.genes)+1),
      "gene"=rep(rownames(NC.genes), length(NC.genes[1,])),
      "cell.type"=rep(c("EPI", "EPI", "EPI", "NC", "NC", "NC"), each = length(NC.genes[,1])),
      "group"=paste(rep(rownames(NC.genes), length(NC.genes[1,])),
        rep(c("EPI", "EPI", "EPI", "NC", "NC", "NC"), each = length(NC.genes[,1])),sep=".")
```

```
    )
  NC.genes.df <- NC.genes.df[order(NC.genes.df$group),]

# combine results tables
  EPI.genes.df <- rbind(EPI.genes.df, NC.genes.df)

## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
# CARDIO DATASET - convert from tximported counts to CPM
# Create a very quick size factor calculation...
# Counts per million + pseudocount
  cm.sf <- colSums(CM.counts+1)/1e6

# and normalise...
  CM.genes <- t(t(CM.counts+1)/cm.sf)
  cmml <- length(CM.markers)

# subset by the markers for quicker computation
  CM.genes = CM.genes[CM.markers,]

# Create and format a results table for plotting
  CM.genes.df <- data.frame(
      "log.cpm"=log(as.vector(CM.genes)+1),
      "gene"=rep(rownames(CM.genes), length(CM.genes[1,])),
      "cell.type"=rep(c("H9", "H9", "CM", "CM"), each = length(CM.genes[,1])),
      "group"=paste(rep(rownames(CM.genes), length(CM.genes[1,])),
      rep(c("H9", "H9", "CM", "CM"), each = length(CM.genes[,1])),sep=".")
    )
  CM.genes.df <- CM.genes.df[order(CM.genes.df$group),]

# NC gene expression
# normalise...
  H9.genes <- t(t(CM.counts+1)/cm.sf)
  h9ml <- length(H9.markers)

# subset by NC markers
  H9.genes = H9.genes[H9.markers,]

# Create and format a results table for plotting
  H9.genes.df <- data.frame(
      "log.cpm"=log(as.vector(H9.genes)+1),
      "gene"=rep(rownames(H9.genes), length(H9.genes[1,])),
      "cell.type"=rep(c("H9", "H9", "CM", "CM"), each = length(H9.genes[,1])),
      "group"=paste(rep(rownames(H9.genes), length(H9.genes[1,])),
        rep(c("H9", "H9", "CM", "CM"), each = length(H9.genes[,1])),sep=".")
    )
  H9.genes.df <- H9.genes.df[order(H9.genes.df$group),]

# combine results tables
  CM.genes.df <- rbind(CM.genes.df, H9.genes.df)

## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
##  BAR CHARTS
```

```
# Create our barplot dataframes...
# This is pretty arduous, but it works
# We're just formatting the data tables
  CM.cpm.df <- data.frame(row.names=unique(CM.genes.df[,2]),
      "CM.cpm" = sapply(1:(length(CM.genes.df[,1])/4), function(i){
      a <- seq(1,length(CM.genes.df[,1])-1, by=4)[i]
      b <- seq(2,length(CM.genes.df[,1]), by=4)[i]
      return(mean(CM.genes.df[a:b,1]))
    }),
      "H9.cpm" = sapply(1:(length(CM.genes.df[,1])/4), function(i){
      a <- seq(3,length(CM.genes.df[,1])-1, by=4)[i]
      b <- seq(4,length(CM.genes.df[,1]), by=4)[i]
      return(mean(CM.genes.df[a:b,1]))
    })
  )
  # And one for error bars..?
  CM.error.df <- data.frame(row.names=unique(CM.genes.df[,2]),
      "CM.cpm" = sapply(1:(length(CM.genes.df[,1])/4), function(i){
      a <- seq(1,length(CM.genes.df[,1])-1, by=4)[i]
      b <- seq(2,length(CM.genes.df[,1]), by=4)[i]
      return(sd(CM.genes.df[a:b,1]))
    }),
      "H9.cpm" = sapply(1:(length(CM.genes.df[,1])/4), function(i){
      a <- seq(3,length(CM.genes.df[,1])-1, by=4)[i]
      b <- seq(4,length(CM.genes.df[,1]), by=4)[i]
      return(sd(CM.genes.df[a:b,1]))
    })
  )

# Create our barplot dataframes...
  EPI.cpm.df <- data.frame(row.names=unique(EPI.genes.df[,2]),
      "EPI.cpm" = sapply(1:(length(EPI.genes.df[,1])/6), function(i){
      a <- seq(1,length(EPI.genes.df[,1])-1, by=6)[i]
      b <- seq(3,length(EPI.genes.df[,1]), by=6)[i]
      return(mean(EPI.genes.df[a:b,1]))
    }),
      "NC.cpm" = sapply(1:(length(EPI.genes.df[,1])/6), function(i){
      a <- seq(4,length(EPI.genes.df[,1])-1, by=6)[i]
      b <- seq(6,length(EPI.genes.df[,1]), by=6)[i]
      return(mean(EPI.genes.df[a:b,1]))
    })
  )
  # And one for error bars..?
  EPI.error.df <- data.frame(row.names=unique(EPI.genes.df[,2]),
      "EPI.cpm" = sapply(1:(length(EPI.genes.df[,1])/6), function(i){
      a <- seq(1,length(EPI.genes.df[,1])-1, by=6)[i]
      b <- seq(3,length(EPI.genes.df[,1]), by=6)[i]
      return(sd(EPI.genes.df[a:b,1]))
    }),
      "NC.cpm" = sapply(1:(length(EPI.genes.df[,1])/6), function(i){
      a <- seq(4,length(EPI.genes.df[,1])-1, by=6)[i]
      b <- seq(6,length(EPI.genes.df[,1]), by=6)[i]
      return(sd(EPI.genes.df[a:b,1]))
```

```
    })
  )

 par(mfrow=c(1,2), cex=0.7)
 # using my custom bar plot function
  bars(EPI.cpm.df, error.data = EPI.error.df,
    col=c(Discrete[6], Discrete[9]), bar.width=0.31, ylab="log CPM")
```

```
##    numeric.. numeric...1
## 1      0.825   1.9330003
## 2      1.175   0.4407521
## 3      1.825   6.1075577
## 4      2.175   2.2497316
## 5      2.825   2.6067250
## 6      3.175   0.1290380
## 7      3.825   3.8983912
## 8      4.175   0.1290380
## 9      4.825   2.7021521
## 10     5.175   5.5418227
## 11     5.825   0.2368330
## 12     6.175   5.2685095
```

```
  legend("topright", legend=c("EPI", "NC"),
    col=c(Discrete[6], Discrete[9]), pch=15, bty="n")

  bars(CM.cpm.df, error.data = CM.error.df,
    col=c(Discrete[7], Discrete[9]), bar.width=0.31, ylab="log CPM")
```

```
## Warning in arrows(mean(c(i + bp1[j], i + bp2[j])), data[i, j], mean(c(i + :
## zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(mean(c(i + bp1[j], i + bp2[j])), data[i, j], mean(c(i + :
## zero-length arrow is of indeterminate angle and so skipped
```

```
##    numeric.. numeric...1
## 1      0.825  8.61446252
## 2      1.175  0.29158935
## 3      1.825  8.29387328
## 4      2.175  0.04759196
## 5      2.825  4.93455059
## 6      3.175  0.04693648
## 7      3.825  7.30594817
## 8      4.175  0.24261002
## 9      4.825  1.92515627
## 10     5.175  6.74633108
## 11     5.825  3.98067267
## 12     6.175  6.11797995
```

```
  legend("topright", legend=c("CM", "H9"),
    col=c(Discrete[7], Discrete[9]), pch=15, bty="n")
```

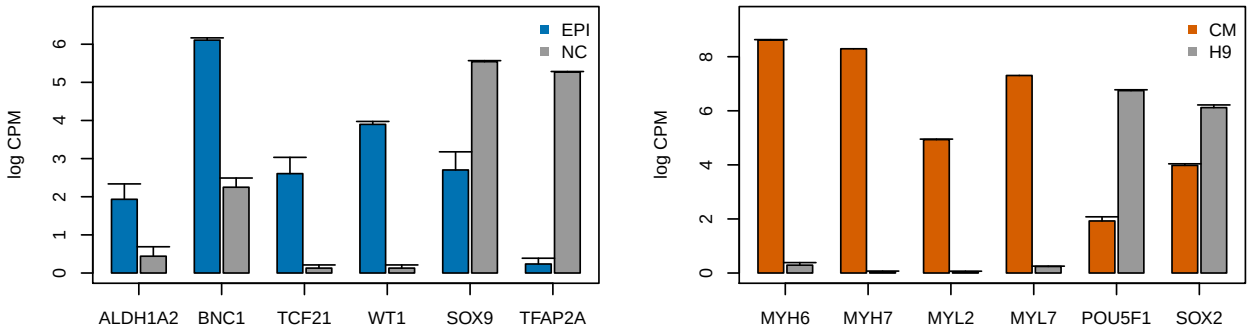

Figure 2: Gene markers for cell line validation.

We then performs some t tests to ensure that the differences between conditions are significant. We see that our cell types of interest had significantly greater expression of positive marker genes and significantly lower expression of negative marker genes.

```
# Welches two sample t tests
  CM.ttest.list <- lapply(unique(CM.genes.df[,2]), function(gene){
    tdf <- CM.genes.df[CM.genes.df[,2] == gene,]
    t.test(tdf[tdf[,3] == "CM",1], tdf[tdf[,3] == "H9",1], paired=F)
  })
  CM.ttests.df = data.frame( unique(CM.genes.df[,2]),
    Pvalue= sapply(unique(CM.genes.df[,2]), function(gene){
      tdf <- CM.genes.df[CM.genes.df[,2] == gene,]
      unlist(t.test(tdf[tdf[,3] == "CM",1], tdf[tdf[,3] == "H9",1], paired=F)[[3]])
    }
  ))
  CM.ttests.df
```

```
##   unique.CM.genes.df...2..       Pvalue
## 1                     MYH6 3.727461e-03
## 2                     MYH7 1.239737e-03
## 3                     MYL2 1.904318e-05
## 4                     MYL7 2.777603e-04
## 5                   POU5F1 1.092662e-02
## 6                     SOX2 3.975974e-03
```

```
  EPI.ttest.list <- lapply(unique(EPI.genes.df[,2]), function(gene){
    tdf <- EPI.genes.df[EPI.genes.df[,2] == gene,]
    t.test(tdf[tdf[,3] == "EPI",1], tdf[tdf[,3] == "NC",1],
    paired=F)
  })
  EPI.ttests.df = data.frame( unique(EPI.genes.df[,2]),
    Pvalue= sapply(unique(EPI.genes.df[,2]), function(gene){
      tdf <- EPI.genes.df[EPI.genes.df[,2] == gene,]
      unlist(t.test(tdf[tdf[,3] == "EPI",1], tdf[tdf[,3] == "NC",1],
      paired=F)[[3]])
    }
  ))
  EPI.ttests.df
```

```
##   unique.EPI.genes.df...2..       Pvalue
## 1                   ALDH1A2 9.230324e-03
## 2                      BNC1 7.463696e-04
```

```
## 3                    TCF21 7.896420e-03
## 4                      WT1 5.779526e-07
## 5                     SOX9 9.047456e-03
## 6                   TFAP2A 2.615421e-04
```

Replicate correlation

It is important that our within-group samples correlate relatively well together. This is essential as a lower correlation is indicative of experimental or technical errors and may affect downstream analyses. This is especially important with a low number of samples per condition as we have in this analysis. Here are some sample vs sample plots to show this correlation. The numbers are also calculated in this next section of code.

```
## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
# Correlations as per Spearmans
# To conform with guidelines for isogenic sample replicates
# Note - we write as png to reduce the file size resulting from the
# vast number of data points.

 CM.cors <- cor(CM.counts, method="spearman")
 jpeg("output/v_SciData/technical_validation/figures/CM_correlations.jpg")
 par(mar=c(5,5,5,5), omi=c(1,1,1,1), oma=c(0,0,0,0))
    plot(as.data.frame(CM.counts),main="Spearmans Rho", col=Discrete[7])
    for(i in 1:16){
    par(fig=as.numeric(subplot.coords(4,4, c(0.1, 0.9, 0.15, 0.95))[i,]))
    }
  dev.off()
```

```
## pdf
##   2
```

```
  EPI.cors <- cor(EPI.counts, method="spearman")
  jpeg("output/v_SciData/technical_validation/figures/EPI_correlations.jpg")
  par(mar=c(5,5,5,5), omi=c(1,1,1,1), oma=c(0,0,0,0))
    plot(as.data.frame(EPI.counts),main="Spearmans Rho", col=Discrete[6])
    for(i in 1:36){
    par(fig=as.numeric(subplot.coords(6,6, c(0.1, 0.9, 0.15, 0.95))[i,]))
    }
  dev.off()
```

```
## pdf
##   2
```

```
##            d0_1      d0_2     d30_1     d30_2
## d0_1  1.0000000 0.9623227 0.9062774 0.9075278
## d0_2  0.9623227 1.0000000 0.9043103 0.9074639
## d30_1 0.9062774 0.9043103 1.0000000 0.9664063
## d30_2 0.9075278 0.9074639 0.9664063 1.0000000
```

```
##           EPI1      EPI3      EPI4       NC1       NC2       NC3
## EPI1 1.0000000 0.9514115 0.9572648 0.8452472 0.9010620 0.8960638
## EPI3 0.9514115 1.0000000 0.9511272 0.8401004 0.8919098 0.8871202
## EPI4 0.9572648 0.9511272 1.0000000 0.8507881 0.9022173 0.8996547
## NC1  0.8452472 0.8401004 0.8507881 1.0000000 0.8945217 0.8993078
## NC2  0.9010620 0.8919098 0.9022173 0.8945217 1.0000000 0.9548870
## NC3  0.8960638 0.8871202 0.8996547 0.8993078 0.9548870 1.0000000
```

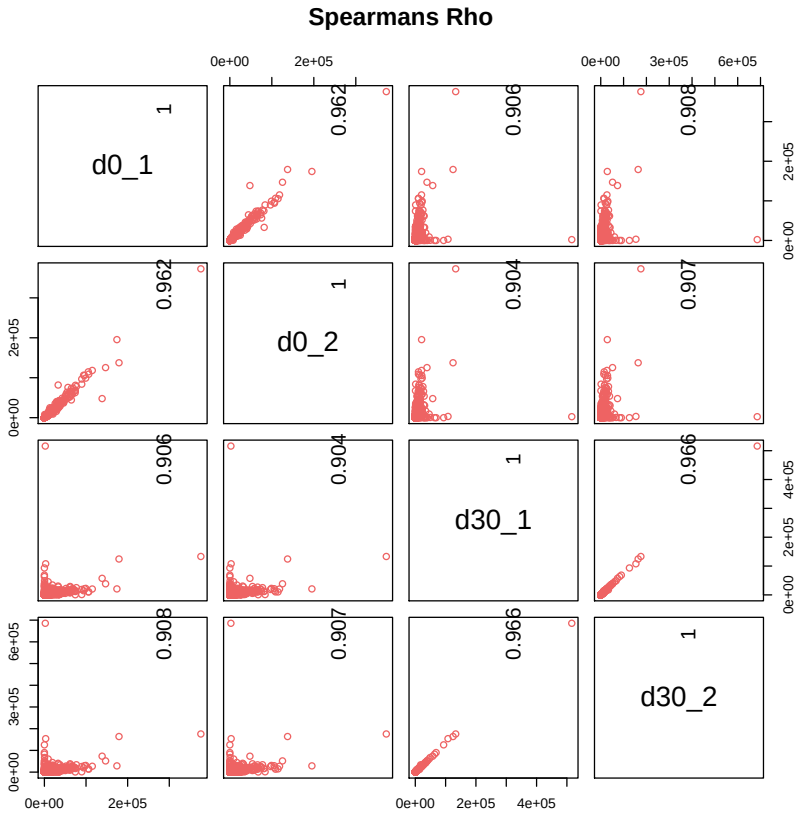

Figure 3: Replicate correlation plots for CM data

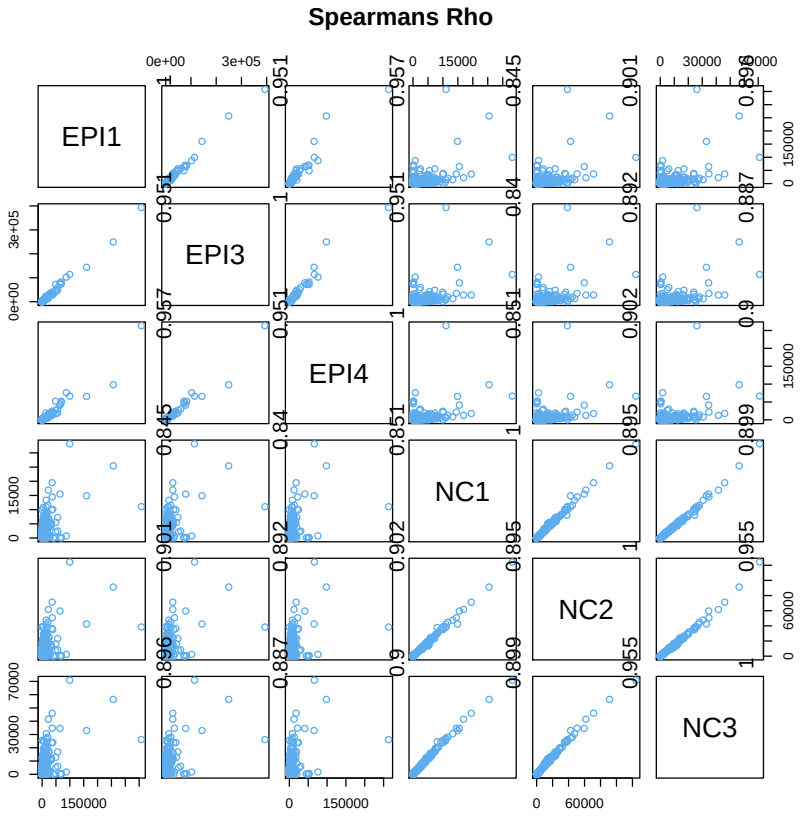

Figure 4: Replicate correlation plots for EPI data

Feature processing

Before moving forward onto differential expression analysis we must consider a very slight data cleaning. We remove all genes that are no read, doing so for each sample independently.

```
# We remove all genes with 0 reads in all samples...
# creating expressed genes vectors for both
  A.express = names(which(rowSums(EPI.counts) > 0 ))

  B.express = names(which(rowSums(CM.counts)  > 0 ))

# Here's the intersect in case we need it...
  express.intersect = A.express[A.express %in% B.express]

  A = EPI.counts[A.express,]
  B = CM.counts[B.express,]
```

This was a very simple filtering step. Because we are isolating our experiments and not combining a matrix for analysis between them, we did not really need to merge or find the intersect of gene lists. It is also important to consider retaining their separate gene dimensions because we are interested in fundamentally different genes in each cell type and they may not be expressed at all in the other cell, for example cognate receptors in list *B* from secreted factors in list *A*.

Running the differential expression analysis

Differential expression analysis is relatively simple with the toolset of the modern-day bioinformatician. Here we are also dealing with bulk RNA-seq data which is much more straightforward than its single-cell counterpart. I begin with the **EPI.counts** dataset, creating the DESeq object and performingthe analaysis. The procedure is repeated for the CM.counts data.

```
# 1. What genes are upregulated in our EPI dataset vs the NC negative control?
# We first grab the EPI dataset
  sampleTable.A <- data.frame(condition = factor(rep(c("EPI", "NC"), each = 3)))
  sampleTable.A[,1] <- relevel(sampleTable.A[,1], "NC")
  rownames(sampleTable.A) <- colnames(A)

# Construct the deseq object
    dds.A <- DESeqDataSetFromMatrix(countData = A, colData = sampleTable.A,
      design = ~condition)

  # Run DESeq2, very simple.
    dds.A <- DESeq(dds.A)

  # grab the results of course
    res.A <- results(dds.A, contrast=c("condition",  "EPI", "NC"))

  # LFC shrinkage correction
    resLFC.A <- lfcShrink(dds.A, coef="condition_EPI_vs_NC", type="apeglm")
    resLFC.A <- resLFC.A[order(resLFC.A[,5]),]

# 2. What genes are upregulated in the downloaded d30 CM data when
# compared with their d0 data? We grab the CM dataset we prepared
  sampleTable.B <- data.frame(condition = factor(rep(c("D0", "D30"), each = 2)))
    relevel(sampleTable.B[,1], "D0")
  rownames(sampleTable.B) <- colnames(B)

# Construct the deseq object
    dds.B <- DESeqDataSetFromMatrix(countData = B, colData = sampleTable.B,
      design = ~condition)

  # Run DESeq2, very simple.
    dds.B <- DESeq(dds.B)

  # grab the results of course
    res.B <- results(dds.B, contrast=c("condition",  "D30", "D0"))

  # LFC shrinkage correction
    resLFC.B <- lfcShrink(dds.B, coef="condition_D30_vs_D0", type="apeglm")
    resLFC.B <- resLFC.B[order(resLFC.B[,5]),]
```

These will be re-ordered the lists by log2 fold change to make the results a little more meaningful for now.

```
# EPI results dataset
    EPI.res <- resLFC.A
  # remove NAs for our analysis...  These are not good genes.
    EPI.res <- EPI.res[!is.na(EPI.res[,5]),]
  # filter off below 1e-2
    EPI.res.cut <- EPI.res[EPI.res[,5] < 1e-2,]
```

```
  # Order by LFC as dictated in the nature paper supplement Table 2.
    EPI.res.cut.lfc <- EPI.res.cut[order(-EPI.res.cut[,2]),]

  # CM results dataset
    CM.res <- resLFC.B
  # remove NAs for our analysis...  These are not good genes.
    CM.res <- CM.res[!is.na(CM.res[,5]),]
  # filter off below 1e-2
    CM.res.cut <- CM.res[CM.res[,5] < 1e-2,]
  # Order by LFC as dictated in the nature paper supplement Table 2.
    CM.res.cut.lfc <- CM.res.cut[order(-CM.res.cut[,2]),]
```

```
  head(EPI.res.cut.lfc[,c(2,5)],10)
```

```
## log2 fold change (MAP): condition EPI vs NC
##
## DataFrame with 10 rows and 2 columns
##         log2FoldChange                 padj
##              <numeric>            <numeric>
## HAND1  17.112348693128 5.99930991907231e-37
## LRRN4 16.7843622514661 9.81868407340099e-36
## RSPO2   16.24026548054 2.59443886148651e-33
## SST   16.2354546780935 1.35178435488434e-45
## HAND2 15.9772446261951 3.49651925798263e-32
## WNT2  15.8789767071319 7.46194091345799e-32
## GATA6 14.8505391236694 5.90810004786994e-28
## KRT19 14.8332519562987 3.74772195759734e-36
## CFI   14.6299599196924 3.60828719280638e-27
## RGCC  14.5016360921121 3.76519787044428e-26
```

```
  head(CM.res.cut.lfc[,c(2,5)],10)
```

```
## log2 fold change (MAP): condition D30 vs D0
##
## DataFrame with 10 rows and 2 columns
##            log2FoldChange                  padj
##                 <numeric>             <numeric>
## MYH7     17.1663648308412  1.16349668207104e-32
## MYOZ2    15.3682364748681  1.39008761323915e-22
## NPPA     15.1147680855878  6.98944630208494e-22
## LUM      14.5947843512588  1.63954838514091e-20
## SERPINB2 14.5561016382814  2.12359164800672e-20
## ITLN1    14.1485015677819  2.24958025254816e-19
## TRIM63   13.9919802600763  6.08768602636212e-19
## GASK1B   13.9703558114545  6.04860573843654e-19
## ANKRD1   13.9429778099603 1.04319672670078e-119
## NPPB     13.8553574609272   1.3189126007755e-18
```

Generating the secretome, membranome and interactome

Now that we have a list of differentially expressed genes, we can move onto sorting them into secreted and membrane-bound only. However, to do so we need to access a list of reputed secreted factors as well as membrane-bound factors. Both of these are accessible from the human protein atlas at **secretome** and **membranome**. The information page is here: https://www.proteinatlas.org/humanproteome/tissue/secretome#relevant_links_and_publications.

For reference, the secretome and membranome here both contain genes which can be either, for example receptors with soluble isoforms or cleaved portions. For reference, these datasets may have been updated since the analysis. The HPA data here was accessed on the 23rd of March 2020.

```
# Dataset loading Acquire the likely secretome from protein
# atlas
secretome = read.csv("https://www.proteinatlas.org/search/protein_class%3APredicted+secreted+proteins?f
    head = T, stringsAsFactors = F, sep = "\t")
secretome.genes <- secretome[, 1]
# Acquire the likely membranome from protein atlas
membranome = read.csv("https://www.proteinatlas.org/search/protein_class%3APredicted+membrane+proteins?
    head = T, stringsAsFactors = F, sep = "\t")
membranome.genes <- membranome[, 1]
```

Then we need to try identify the interactome, or set of all possible interactions between both lists. Here I access stringDB, downloading their entire database for homosapiens, including the subscores. The URL for the downloads page is here: https://string-db.org/cgi/download.pl?sessionId=cpALfCUxi1pP&species_text=Homo+sapiens. Additionally, I download the information file as we need to convert the ensemblIDs of proteins in the form of **ENSP******* into the gene names such as HGNC symbols instead. This file was edited as it was initially throwing errors in R with "In scan(file = file, what = what, sep = sep, quote = quote, dec = dec, : EOF within quoted string". This was found to be a unclosed double quote on line 12364, column 399 gene (UCHL3).

```
# We first load up the stringDB database file using the per-channel scores.
# This may be key as we can filter the list down by interactions only
# through experimental evidence.
  stringDB2 <- read.csv(stringsAsFactors=F, head=T,
                 "data/stringDB/9606.protein.links.detailed.v11.0.txt", sep=" ")

# The problem with these stringDB files is that they are written as
# ENSP* (ensembl protein IDs). Luckily, we can access the stringDB info file to
# convert to gene names and map to the rest of our data.
  stringDB.info <- read.csv(stringsAsFactors=F, head=T,
                     "data/stringDB/9606.protein.info.v11.0_edited.txt", sep="\t")

# We just need to parse and match this file with our list of StringDB proteins.
  sdb.1 <- stringDB2[,1]
  sdb.2 <- stringDB2[,2]
  sdb.map.1 <- match(sdb.1, stringDB.info[,1])
  sdb.map.2 <- match(sdb.2, stringDB.info[,1])

  stringDB2[,1] <- stringDB.info[sdb.map.1,2]
  stringDB2[,2] <- stringDB.info[sdb.map.2,2]
```

After we have created the human interactome object from stringDB we need to filter out the lower quality interactions. For example, by retaining only interactions with a score of above 700 we can drastically reduce the dimensions of our interactome. Changing this this number can be a matter of debate. Moreover, we can filter by evidence category only by filtering only over that evidence column. Additionally, I retained the scores for both the experimental only and combined columns.

```
  score.threshold <- 700
  stringDB.full <- stringDB2[,c(1,2,7,10)]
```

Next, we take the epicardial upregulated genes only, finding and retaining those which are also present in secretome. This is repeated for the cardiomyocytes and the membranome. By selectively searching for the presence of the genes in the interactome first column, we can identify the complementary protein or interactor in the second column. If we use the epicardial secreted factors that means we will find their complement. Then, we repeat for

the CMs and the membranome.

```
# Identify those interactions from the epicardial upregulated genes.
  # EPI results dataset
    EPI.res <- resLFC.A
  # remove NAs for our analysis...  These are not good genes.
    EPI.res <- EPI.res[!is.na(EPI.res[,5]),]
  # filter off below p value 1e-2
    EPI.res.cut <- EPI.res[EPI.res[,5] < 1e-2,]
  # Order by LFC as dictated in the nature paper supplement Table 2.
    EPI.res.cut.lfc <- EPI.res.cut[order(-EPI.res.cut[,2]),]

  # Find out which upregulated genes are secreted by EPI...
  # Subset by LFC (positive)
    EPI.secretome <- EPI.res.cut.lfc[EPI.res.cut.lfc[,2] > 0,]
  # filter by secreted genes only
    EPI.secretome <- rownames(EPI.secretome)[
      rownames(EPI.secretome) %in% secretome.genes]
  # Keep this Dataframe?
    secreted.genes <- EPI.secretome; length(secreted.genes)# n = 379
  # Match secreted genes with the first column of our interactions database
  # This makes the assumption that as these are secreted proteins.
    EPI.secretome <- EPI.secretome[EPI.secretome %in% stringDB.full[,1]]
  # The fraction of membrane genes that are mapped in stringDB
    length(EPI.secretome)/length(secreted.genes)# 99.5 %

  # Find all interactions in stringDB where stringDB is in the EPI.secretome
    EPI.secretome.interactome <- stringDB.full[stringDB.full[,1] %in% EPI.secretome,]
      colnames(EPI.secretome.interactome) <-c(
        "secreted", "interactor", "exp.score", "score"
        )


## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
# CM and membranome

# Identify those interactions from the cardiomyocytes upregulated genes
  # CM results dataset
    CM.res <- resLFC.B
  # remove NAs for our analysis...  These are not good genes.
    CM.res <- CM.res[!is.na(CM.res[,5]),]
  # filter off below p value 1e-2
    CM.res.cut <- CM.res[CM.res[,5] < 1e-2,]
  # Order by LFC as dictated in the nature paper supplement Table 2.
    CM.res.cut.lfc <- CM.res.cut[order(-CM.res.cut[,2]),]

  # Find out which upregulated genes are membrane-bound by CM...
  # Subset by LFC (positive)
    CM.membranome <- CM.res.cut.lfc[CM.res.cut.lfc[,2] > 0,]
  # filter by membrane genes only
    CM.membranome <- rownames(CM.membranome)[
      rownames(CM.membranome) %in% membranome.genes]
  # Keep this Dataframe?
```

```
    membrane.genes <- CM.membranome; length(membrane.genes)# n = 1417
  # match membrane genes with the first column of our interactions database
  # This makes the assumption that as these are membrane proteins.
    CM.membranome <- CM.membranome[CM.membranome %in% stringDB.full[,1]]
  # The fraction of membrane genes that are mapped in stringDB
    length(CM.membranome)/length(membrane.genes)# 95.5 %

  # Find all interactions in stringDB where stringDB is in the CM.membranome
    CM.membranome.interactome <- stringDB.full[stringDB.full[,1] %in% CM.membranome,]
      colnames(CM.membranome.interactome) <- c(
        "membrane", "interactor", "exp.score", "score"
        )
```

Finding the crosstalk interactome

We can now find the overlap between the two datasets, finding our potential crosstalk by matching the different columns together. Since **EPI[,1]** is the epicardial secretome, **EPI[,2]** are interactors with this epicardial secretome, and **CM[,1]** are the membrane-proteins of the cardiomyocytes, if we then identify where **EPI[,2]**is in **CM[,1]** we will find the CM membrane-bound proteins that interact with EPI-secreted proteins.

```
# Matching the secretome where secreted genes appear in the secretome column
# of the CM membrane interaction.
  EPI.CM.interactome <- EPI.secretome.interactome[
    which(EPI.secretome.interactome[,2] %in% CM.membranome.interactome[,1]),]
# Picking up the essential information from the dataframes created in this analysis
  EPI.CM.interactome <- EPI.CM.interactome[order(EPI.CM.interactome[,1]),]
  EPI.CM.interactome$BaseMean.EPI <- EPI.res[EPI.CM.interactome[,1],1]
  EPI.CM.interactome$L2FC.EPI <- EPI.res[EPI.CM.interactome[,1],2]
  EPI.CM.interactome$BaseMean.CM <- CM.res[EPI.CM.interactome[,2],1]
  EPI.CM.interactome$L2FC.CM <- CM.res[EPI.CM.interactome[,2],2]
  EPI.CM.interactome$L2FC.average <- (EPI.CM.interactome$L2FC.CM +
                                      EPI.CM.interactome$L2FC.EPI) / 2

# ordering the full interactome by the score
  EPI.CM.interactome <- EPI.CM.interactome[order(-EPI.CM.interactome$exp.score),]

# This list is then cut by the experimental scores
  EPI.CM.interactome.cut <- EPI.CM.interactome[EPI.CM.interactome[,3] > 700,]
  EPI.CM.interactome.cut <- EPI.CM.interactome.cut[
                              order(-EPI.CM.interactome.cut$L2FC.average),]

# and then extending the network within these interactions
  EPI.CM.interactome.ext <- EPI.CM.interactome[which(
    EPI.CM.interactome[,1] %in% as.vector(unlist(EPI.CM.interactome.cut[,1:2])) &
    EPI.CM.interactome[,2] %in% as.vector(unlist(EPI.CM.interactome.cut[,1:2]))
  ),]

# Cutting these intra-interactions with a score of over 400
  EPI.CM.interactome.ext <- EPI.CM.interactome.ext[EPI.CM.interactome.ext[,3] > 400,]
```

We further refine this list to retain only proteins that are more highly secreted in the EPI than the CM cells. The rational for this is to exclude secreted factors from CM that are autocrine in the hopes of creating a list of EPI specific genes for further exploration of the EPI effect on CM-based therapies. To do this we normalised within

samples to the counts per million and perform a basic independent samples t test. We found that secreted genes were ($p < 0.05$).

```
  sec.genes <- unique((EPI.CM.interactome.cut[,1]))

  # First we create the CPM matrices
  # CM DATASET
  # Create a very quick size factor calculation...
  # Counts per million
    cm.sf <- colSums(CM.counts+1)/1e6

  # and normalise...
    CM.cpm <- t(t(CM.counts+1)/cm.sf)

    CM.cpm.sec = CM.cpm[sec.genes,3:4]


  # EPI DATASET
  # Create a very quick size factor calculation...
  # Counts per million
    epi.sf <- colSums(EPI.counts+1)/1e6

  # and normalise...
    EPI.cpm <- t(t(EPI.counts+1)/epi.sf)

    EPI.cpm.sec = EPI.cpm[sec.genes,1:3]


# Take a log of the CPM (logCPM)
    EPI.logcpm.sec <- log(EPI.cpm.sec + 1)
    CM.logcpm.sec  <- log(CM.cpm.sec  + 1)


# CPM means dataframe
# Dataframe for results
  cpm.df <- data.frame(row.names=sec.genes, rowMeans(EPI.logcpm.sec),
                       rowMeans(CM.logcpm.sec))
    cpm.order <- order(-(cpm.df[,1]  - cpm.df[,2]))
  cpm.df <- cpm.df[cpm.order,]

# sd CPM error bars dataframe
  error.df <- data.frame(row.names=sec.genes, rowSds(EPI.logcpm.sec),
                         rowSds(CM.logcpm.sec))
    error.df <- error.df[cpm.order,]

# t tests for our datasets
# We can't really be sure of the distributions though...
# Certainly because the n = 3 and n = 2
  ttest.p <- data.frame(row.names=sec.genes[cpm.order],
              -(cpm.df[,1]  - cpm.df[,2]),
              round(sapply(1:length(sec.genes), function(i) {
              t.test(EPI.logcpm.sec[i,], CM.logcpm.sec[i,], paired=F)$p.value
              }),3)[cpm.order]
             )
```

```
# Threshold    for t test cutoff
  ttest.cut <- ttest.p[,2] < 0.05

# A list of all genes which are significantly more expressed in EPI
  epi.sec <- rownames(ttest.p)[ttest.p[,2] < 0.05 & ttest.p[,1] < 0]

# subset the interactome results table by the significantly "more EPI" genes
  EPI.CM.interactome_EPI.secreted <- EPI.CM.interactome.cut[
                                      EPI.CM.interactome.cut[,1] %in% epi.sec,
                                     ]
```

```
# dataframe of x and y coordinates (results from bars function)     #
  bar.coords <- bars(cpm.df, error.data = error.df,
    col=c(Discrete[6], Discrete[7]), bar.width=0.31,
    ylab="log CPM", xaxt="n"
  )
# legend
  legend("topright", legend=c("EPI", "CM"),
    col=c(Discrete[6], Discrete[7]), pch=15, bty="n")

# Axis labels
  axis(1, at=c(1:length(cpm.df[,1])), labels=rownames(cpm.df), las=2)

# Error points
  points(bar.coords[rep(ttest.cut, each=2),1],
    bar.coords[rep(ttest.cut, each=2),2]+0.6, pch="*")

# line to separate more or less expressedin EPI
  abline(v=which(ttest.p[,1] > 0)[1]-0.5, lty=2)
```

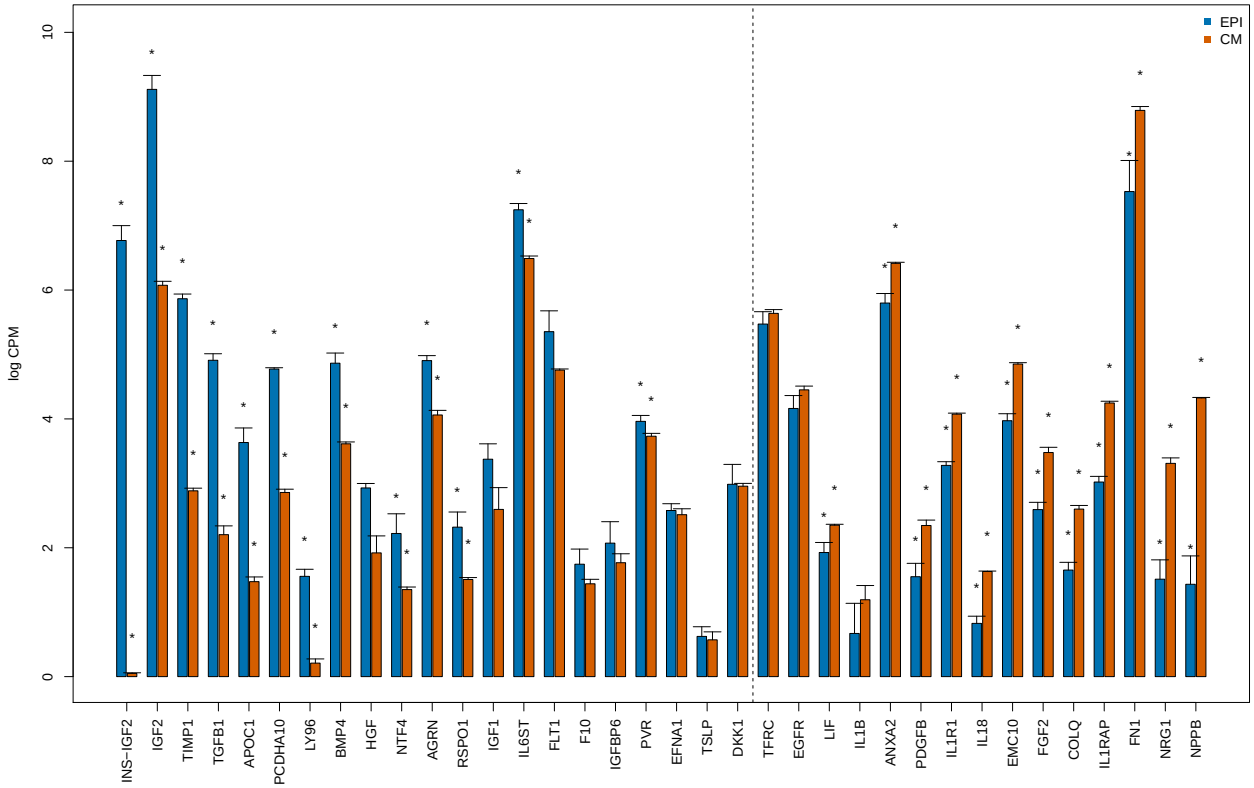

Figure 5: bar chart for the CPM of EPI and CM secreted factors. Asterisks illustrate significant genes.

Plotting

Now that we have created our interactome. We can plot some of the results in a volcano plot. Because of the nature of the interactome and connections between cells, it was only fair to draw a combined volcano plot figure.

```
  interactome.dataframe <- EPI.CM.interactome.cut
# DESEQ2 results visualisation:
# Cardio
  cmdf <- data.frame(
    "genes" = rownames(CM.res),
    "l2fc" = -CM.res[,2],
    "p.adj" = CM.res[,5],
    "BM" = CM.res[,1]
  )
  cmdf[cmdf[,3]==0,3] <- 1e-310

  CM.receptors <- cmdf[,1] %in% interactome.dataframe[,2]
#  CM.receptors <- rownames(cmdf) %in% CM.membranome
# We then generate the cm plot, make a report of it
  colour3 <- c("white", Discrete[7])
  colour4 <- c("white", Discrete[9])
  t1 <- plot.volcano(cmdf, input.format="custom", legend=F,
        colourset1=colour3, colourset2=colour4,
    fade_style=3, p.level=0.00, lines=F, new.plot=F, report=T)
  t1[,"x"] = t1[,"x"]+max(epidf[,2])+gap+abs(min(t1[,1]))
```

```
# epi
  epidf <- data.frame(
    "genes" = rownames(EPI.res),
    "l2fc" = EPI.res[,2],
    "p.adj" = EPI.res[,5],
    "BM" = EPI.res[,1]
  )
  epidf[epidf[,3]==0,3] <- 1e-310
  EPI.secreted <- epidf[,1] %in% interactome.dataframe[,1]
#  EPI.secreted <- rownames(epidf) %in% EPI.secretome
# We then generate the epi plot, make a report of it
  colour1 <- c("white", Discrete[9])
  colour2 <- c("white", Discrete[6])
  t2 <- plot.volcano(epidf, input.format="custom", legend=F,
  colourset1=colour1, colourset2=colour2,
    fade_style=3, p.level=0.00, lines=F, new.plot=F, report=T)
  t2[,"x"] = t2[,"x"]

  par(cex=2, mar=c(4,4,4,4))
  gap=10
  plot(
    c(min(epidf[,2]), max(epidf[,2])+gap+diff(range(cmdf[,2]))),
    c(0,max(-log10(c(cmdf[,3],epidf[,3])))),
    pch=NA,
    xaxt="n",
    ylab="significance (-log10 P)",
    xlab="log 2 fold change"
  )

# Association lines:
  for(i in 1:length(interactome.dataframe[,1])){
    x0 = epidf[epidf[,1] == interactome.dataframe[i,1],"l2fc"]
    y0 = -log10(epidf[epidf[,1] == interactome.dataframe[i,1],"p.adj"])
    x1 = t1[rownames(t1) == interactome.dataframe[i,2],"x"]
    y1 = t1[rownames(t1) == interactome.dataframe[i,2],"y"]

    c1 = rgb(colorRamp(colors=c("white", "black"))(  y0 /
      max(-log10(c(cmdf[,3],epidf[,3])))/255)
    c2 = rgb(colorRamp(colors=c("white", "black"))(  y1 /
      max(-log10(c(cmdf[,3],epidf[,3])))/255)

    gradient.line(x0,y0,x1,y1, length=100, col=c(c1,c2), lwd=1, lend=3)
  }


# generate points
        points(t1[,"x"], -log10(t1[,2]), col=t1$cols, pch=16)
# and overlay the interactome membrane factors
        points(t1[CM.receptors,"x"], -log10(t1[CM.receptors,2]),
          bg=t1[CM.receptors,"cols"], pch=21)
  top.genes <- match(interactome.dataframe[,2], rownames(t1[CM.receptors,]))[1:10]
  text.overlay(t1[CM.receptors,"x"][top.genes], -log10(t1[CM.receptors,2])[top.genes],
    rownames(t1)[CM.receptors][top.genes], cex = 0.5, void.x = t1[CM.receptors,"x"],
```

```
    void.y = t1[CM.receptors,"y"])
  mtext("Cardiomyocytes", 3, adj=0.7, col=colour3)
  mtext("H9", 3, adj=0.95, col=colour4)
  axis(1, at=seq(-15,15,5)+max(epidf[,2])+gap+abs(min(t1[,1])), labels=seq(-15,15,5))


# We then generate the EPI plot
        points(t2[,"x"], -log10(t2[,2]), col=t2$cols, pch=16)
# and overlay the interactome membrane factors
        points(t2[EPI.secreted,"x"], -log10(t2[EPI.secreted,2]),
          bg=t2[EPI.secreted,"cols"], pch=21)
  top.genes <- match(interactome.dataframe[,1], rownames(t2[EPI.secreted,]))[1:10]
  text.overlay(t2[EPI.secreted,"x"][top.genes],-log10(t2[EPI.secreted,2])[top.genes],
    rownames(t2)[EPI.secreted][top.genes], cex=0.5, void.x = t2[EPI.secreted,"x"],
    void.y = t2[EPI.secreted,"y"])
  mtext("Epicardial cells", 3, adj=0.3, col=colour2)
  mtext("Neural Crest", 3, adj=0.05, col=colour1)
  axis(1,at=seq(-15,15,5), labels=seq(-15,15,5))
```

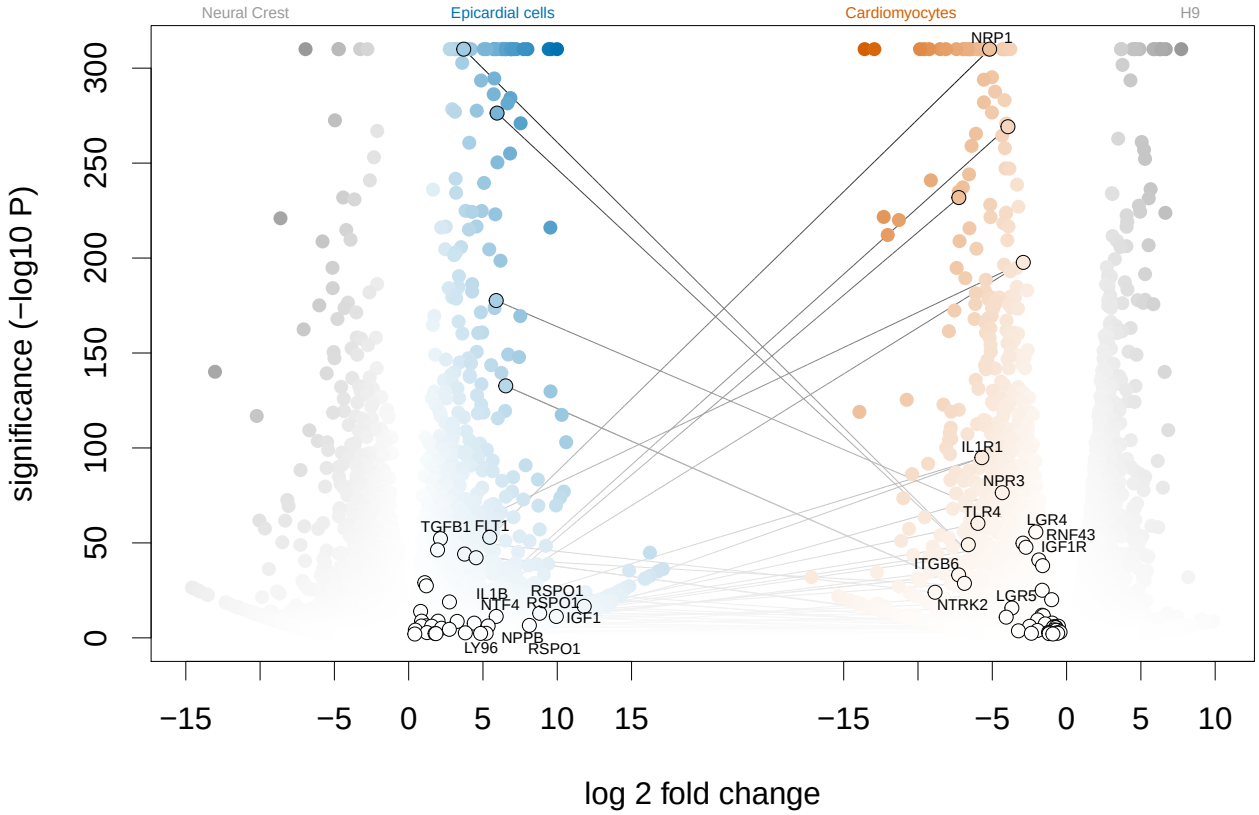

Figure 6: The top 10 differentially upregulated hits for both EPI and CM, linked by their interactome from stringDB

Finishing words

This closes the R-based analysis of the two data sets. We generated a number of interactome results files which were then passed into cytoscape. Specifically, the gene pairs, log-fold change and combined score of the interactome pairs were used as network parameters. The network was then manually layed out and embellished with the relevant diagramatic annotations.

```
sessionInfo()
```

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.4 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_GB.UTF-8        LC_COLLATE=en_GB.UTF-8
##  [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_GB.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] extraDistr_1.8.11           xml2_1.2.0
##  [3] jsonlite_1.6.1              httr_1.4.0
##  [5] wordcloud_2.6               RColorBrewer_1.1-2
##  [7] tximport_1.12.3             DESeq2_1.20.0
##  [9] SummarizedExperiment_1.10.1 DelayedArray_0.10.0
## [11] BiocParallel_1.14.2         matrixStats_0.54.0
## [13] Biobase_2.40.0              GenomicRanges_1.32.6
## [15] GenomeInfoDb_1.16.0         IRanges_2.18.0
## [17] S4Vectors_0.22.0            BiocGenerics_0.30.0
## [19] knitr_1.28                  rmarkdown_1.13
##
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-6           bit64_0.9-7            numDeriv_2016.8-1
##  [4] tools_3.6.3            backports_1.1.6        R6_2.4.1
##  [7] rpart_4.1-15           Hmisc_4.2-0            DBI_1.0.0
## [10] colorspace_1.4-1       nnet_7.3-13            apeglm_1.6.0
## [13] tidyselect_1.0.0       gridExtra_2.3          bit_1.1-14
## [16] curl_3.3               compiler_3.6.3         formatR_1.6
## [19] htmlTable_1.13.3       scales_1.1.0           checkmate_2.0.0
## [22] genefilter_1.62.0      stringr_1.4.0          digest_0.6.25
## [25] foreign_0.8-75         XVector_0.20.0         base64enc_0.1-3
## [28] jpeg_0.1-8.1           pkgconfig_2.0.3        htmltools_0.4.0
## [31] bbmle_1.0.20           htmlwidgets_1.5.1      rlang_0.4.5
## [34] rstudioapi_0.11        RSQLite_2.1.1          acepack_1.4.1
```

```
## [37] dplyr_0.8.5            RCurl_1.95-4.12        magrittr_1.5
## [40] GenomeInfoDbData_1.1.0 Formula_1.2-3          Matrix_1.2-18
## [43] Rcpp_1.0.4.6           munsell_0.5.0          Rhdf5lib_1.6.0
## [46] lifecycle_0.2.0        yaml_2.2.1             stringi_1.4.6
## [49] MASS_7.3-51.5          zlibbioc_1.26.0        plyr_1.8.6
## [52] rhdf5_2.28.0           grid_3.6.3             blob_1.1.1
## [55] crayon_1.3.4           lattice_0.20-40        splines_3.6.3
## [58] annotate_1.58.0        locfit_1.5-9.1         pillar_1.4.3
## [61] geneplotter_1.58.0     XML_3.98-1.19          glue_1.4.0
## [64] evaluate_0.14          latticeExtra_0.6-29    data.table_1.12.8
## [67] png_0.1-7              vctrs_0.2.4            gtable_0.3.0
## [70] purrr_0.3.4            assertthat_0.2.1       ggplot2_3.3.0
## [73] emdbook_1.3.11         xfun_0.13              xtable_1.8-4
## [76] coda_0.19-3            survival_2.44-1.1      tibble_3.0.1
## [79] AnnotationDbi_1.42.1   memoise_1.1.0          cluster_2.1.0
## [82] ellipsis_0.3.0
```