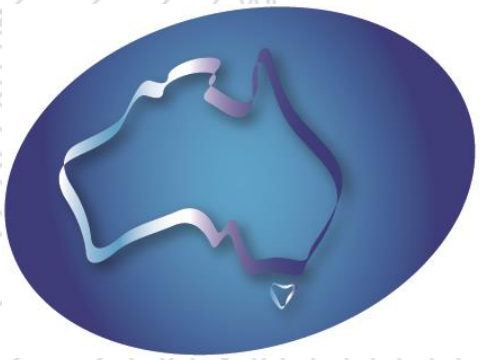


BIOPLATFORMS
AUSTRALIA

RNAseq analysis with R

NSW GOVERNMENT Health

NCRIS
National Research Infrastructure for Australia
An Australian Government Initiative

QFAB Bioinformatics

QFAB

qcif

Queensland Cyber Infrastructure Foundation Ltd, ABN 13 225 133 729, Axon building, 47 – The University of Queensland - St Lucia, Qld, 4072
(QCIF incorporates QFAB Bioinformatics). 16.01.036-20170419

Contents

Chapter 1

NGS data exploration

When working on RNA-Seq data using R the short reads are usually mapped to the reference genome using a genome mapper and the DNA sequences themselves are handled by the mapping software. However, it is useful to know that we can perform a number of analyses on the DNA sequences themselves in R. The QC analysis and evaluation of the sequence collection content can be managed as well.

There are a number of different DNA sequencing platforms in use that have their own characteristics and challenges.

1.1 Required packages for these exercises

If you would like to run these exercises again on your own machine, you will need to install the following R packages.

- `Biostrings`
- `ShortRead`
- `Rsamtools`
- `Rsubread`
- `mixOmics`
- `edgeR`
- `limma`
- `tidyr`
- `ggplot2`

1.2 Reading from a fasta file

FASTA format is a text-based format for representing nucleotide and peptide sequences using their single-letter IUPAC codes. The format also allows for sequence names and comments to precede the sequences. The format originates from the FASTA software package, but has now become a standard in the field of bioinformatics.

The simplicity of FASTA format makes it easy to manipulate and parse sequences using text-processing tools that are built into R. A number of packages make the process of loading a fasta file very much easier.

```
library(Biostrings)
library(ShortRead)

NGS_DIR <- "../data/NGS_sequences"
comt <- readFasta(file.path(NGS_DIR, "comt.fasta"))
print(comt)
```

```
## class: ShortRead
## length: 2 reads; width: 2437 28369 cycles
```

```
id(comt)
```

```
##   A BStringSet instance of length 2
##     width seq
## [1]    41 ENST00000361682 cdna:KNOWN_protein_coding
## [2]    58 22 dna:chromosome chromosome:GRCh37:22:19929130:19957498:1
```

```
sread(comt)
```

```
##   A DNAStringSet instance of length 2
##     width seq
## [1]  2437 CGGGGACACCCTGGCCACCGCCGCGCGGACA...TACCAATAGTCTTATTTTGGCTTATTTTTAA
## [2] 28369 CGGGGACACCCTGGCCACCGCCGCGCGGACA...TACCAATAGTCTTATTTTGGCTTATTTTTAA
```

```
width(comt)
```

```
## [1]  2437 28369
```

```
length(comt)
```

```
## [1] 2
```

From this code it can be seen that we have created an object of `ShortRead` type that contains a `DNAStringSet` containing two DNA sequences. The `width()` function reports the length of each DNA sequence while the `length()` function reports the number of DNA sequences in the sequence collection.

The character representation of the sequence remains accessible:

```
comtStr <- toString(sread(comt[1]))
class(comtStr)
```

```
## [1] "character"
```

```
substring(comtStr,1,50)
```

```
## [1] "CGGGGACACCCTGGCCACCGCCGCGCGGACACCCTCACGAGGACACCCCG"
```

Find all positions in the sequence with “ATG” codon:

```
gregexpr("ATG", comtStr)
```

```
## [[1]]
##  [1]  383  533  579  650  758  836  921  924  941 1000 1300 1337 1390 1441
## [15] 1705 1823 1888 1934 1944 2163 2247
## attr(,"match.length")
##  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## attr(,"useBytes")
## [1] TRUE
```

```
as.vector(gregexpr("ATG", comtStr)[[1]])
```
```
##  [1]  383  533  579  650  758  836  921  924  941 1000 1300 1337 1390 1441
## [15] 1705 1823 1888 1934 1944 2163 2247
```

There are a wide range of different functions that can be applied to the manipulation of individual DNA sequences. We could apply functions that include `strsplit`, `replaceLetterAt`, `subseq`, `maskMotif`, `reverseComplement` etc. These methods work well but for single sequences or for a small collection of sequences, for batch jobs, other software might be more suitable.

There are a number of other convenience utilities in R/Bioconductor:

```
GENETIC_CODE
```
```
## TTT TTC TTA TTG TCT TCC TCA TCG TAT TAC TAA TAG TGT TGC TGA TGG CTT CTC
## "F" "F" "L" "L" "S" "S" "S" "S" "Y" "Y" "*" "*" "C" "C" "*" "W" "L" "L"
## CTA CTG CCT CCC CCA CCG CAT CAC CAA CAG CGT CGC CGA CGG ATT ATC ATA ATG
## "L" "L" "P" "P" "P" "P" "H" "H" "Q" "Q" "R" "R" "R" "R" "I" "I" "I" "M"
## ACT ACC ACA ACG AAT AAC AAA AAG AGT AGC AGA AGG GTT GTC GTA GTG GCT GCC
## "T" "T" "T" "T" "N" "N" "K" "K" "S" "S" "R" "R" "V" "V" "V" "V" "A" "A"
## GCA GCG GAT GAC GAA GAG GGT GGC GGA GGG
## "A" "A" "D" "D" "E" "E" "G" "G" "G" "G"
```

```
IUPAC_CODE_MAP
```
```
##      A      C      G      T      M      R      W      S      Y      K
##    "A"    "C"    "G"    "T"   "AC"   "AG"   "AT"   "CG"   "CT"   "GT"
##      V      H      D      B      N
##  "ACG"  "ACT"  "AGT"  "CGT" "ACGT"
```

1.3 Parsing FastQ files

For a more comprehensive review of what can be performed using `R/Bioconductor` for short reads let us have a look at some data from the public domain. The Short Read Archive (SRA) hosted by the NBCI and provides a public repository where high dimensional data can be shared with the community. Study `ERX337002` looks at the metagenomics of food (http://www.ncbi.nlm.nih.gov/sra/ERX337002). The Ion Torrent platform has been used to sequence the microbial content of an artisanal cheese.

The raw data has been downloaded from the public domain and has been converted from the `sra` format into a `fastq` file format (using SRA Toolkit) that can be used with a number of packages. The fastq file prepared has been gzipped to save space (and increase read performance).

```
curl -O
ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-instant/reads/ByExp/sra/ERX/ERX337/ERX337002/
ERR364233.sra
fastq-dump ERR364233.sra
gzip ERR364233.fastq
```

A brief summary of the content in the file can be prepared by loading the fastq file with the `readFastq` function.

```
cheese <- readFastq(file.path(NGS_DIR, "ERR364233.subset.fastq.gz"))
print(cheese)
```
```
## class: ShortReadQ
## length: 200000 reads; width: 8..358 cycles
```

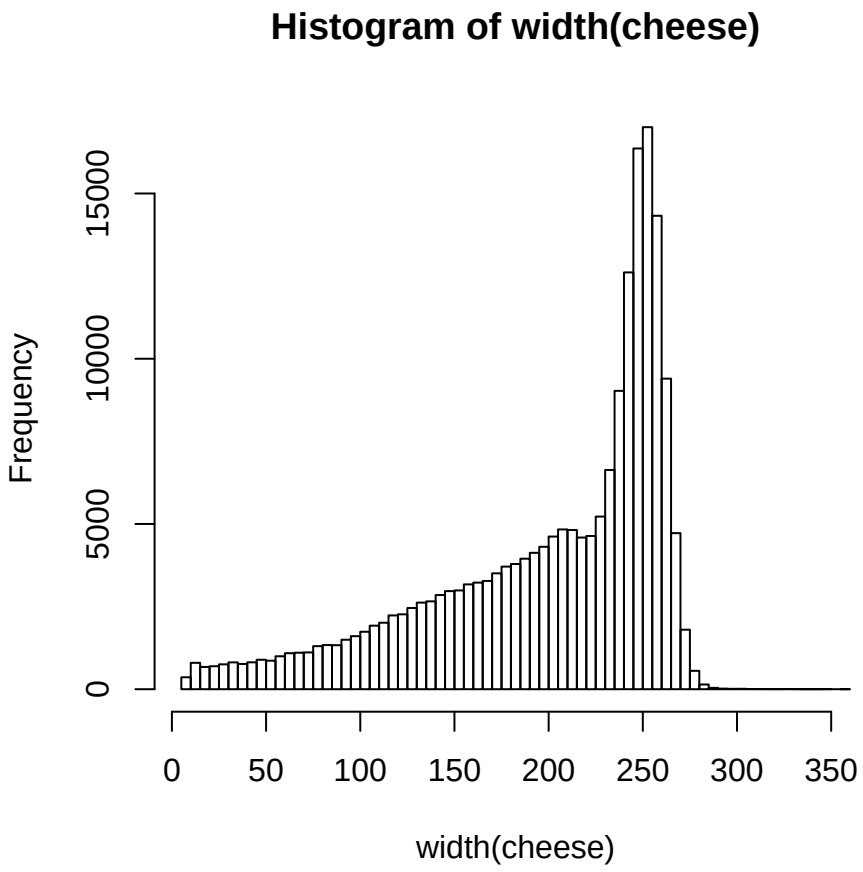
```
qaSummary <- qa(cheese, NGS_DIR)
report(qaSummary, dest="./results")
```

```
## Warning in dir.create(dest, recursive = TRUE): './results' already exists
## [1] "./results/index.html"
```

1. Enter the above commands.
2. Click on the **Files** tab in the bottom-right corner of RStudio, then click on **More** > **Go to working directory**.
3. Click on **New Folder** and create the `results` folder.
4. Run the `qaSummary()` and `report()` commands as above, if not already done.
5. Click on the **results** folder and the **index.html** file.
6. Select the **View in Web Browser** (if a popup window appears click Try again).
7. Have a look a the report from the `qaSummary()` function.

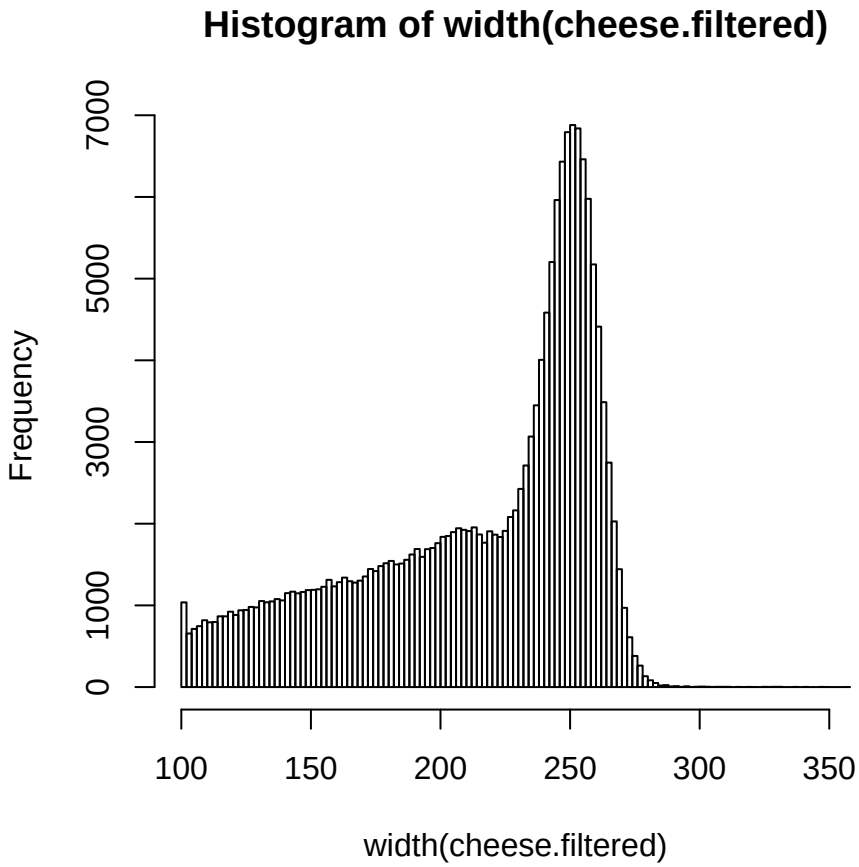
This shows that there are 200K sequence reads in the sequence collection that have a length of between 8 and 358 nucleotides. Let's have a look at the distribution of read lengths:

```
hist(width(cheese), breaks=100)
```



We recommend that we do not use such short reads in our sequence collection. Filter out sequences that are less than 100 nt in length and replot the distribution.

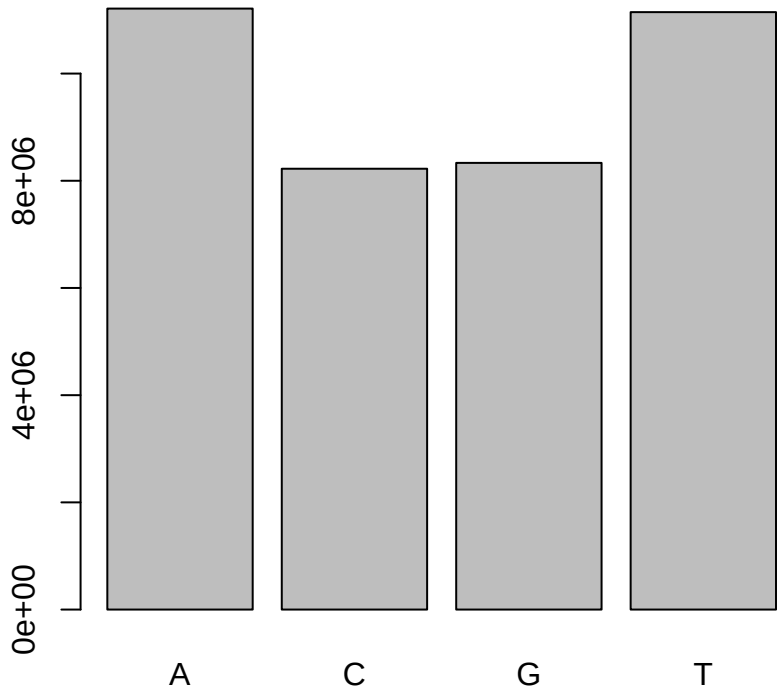
```
cheese.filtered <- cheese[-which(width(cheese) < 100)]
hist(width(cheese.filtered), breaks=100)
```



How many reads were removed from the filtering step? _____ (**hint** `length()`)

We have looked at the distribution of sequence lengths in the collection. We can also check the distribution of individual nucleotides in the collection. The `alphabetFrequency` function collects frequency counts for each of the IUPAC nucleotides in a `DNAStringSet` object. The method will generate a table that be summarised and plotted using your preferred graphics library.

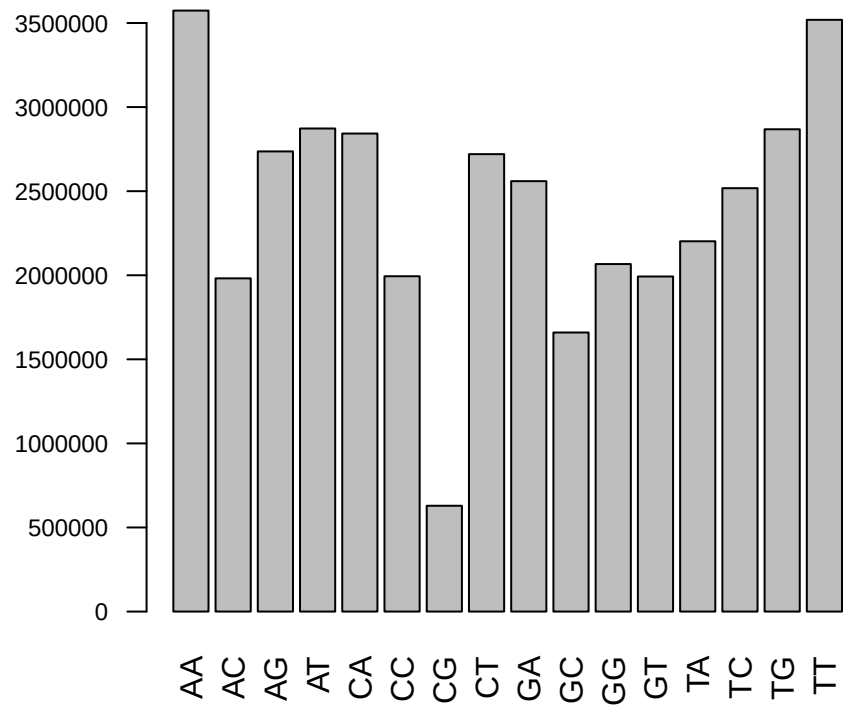
```
freq <- alphabetFrequency(sread(cheese.filtered))
totalCount <- colSums(freq)
barplot(totalCount[1:4])
```



1. Run the command above.
2. What *data type* is the variable `freq`? _____
3. What are the dimensions of `freq`? _____
4. Why did we only plot the first 4 elements of `totalCount`? _____

Similar plots can be prepared for di- and trinucleotide frequencies, this looks at all possible words. You can similarly look for hexanucleotide frequencies or any other number, check the `alphabetFrequency()` documentation. This can assists in detecting any over-represented subsequences.

```
barplot(colSums(dinucleotideFrequency(sread(cheese.filtered))),cex.axis = 0.75, las=2)
```



Depending on the sequencing strategy that you applied, you may have ligated synthetic adapter sequences, primers and other constructs within the target amplicons. There are a number of methods for stripping such synthetic sequences from your sequence collection.

```
head(sread(cheese.filtered))
```

```
##   A DNAStringSet instance of length 6
##     width seq
## [1]   173 AACATTTGTCCTAATCATTGGGGATACAGGG...TTTCTGCAAATCCTTTTTCTAAAAGATAGAG
## [2]   201 AGAAAATAAAGAGAGAAAACAAACAAACAAA...GAAAGAAACAGAGAAGAAGAAAGAAAAAAAA
## [3]   198 GGTGACCTTTTTTTTTTTTTTTTTTCAAGAA...AGACTTGGCAAACAGAAAAGGAGGGTATCAA
## [4]   123 TTTTATTTGTTTTTTGATGCTATAGTAAATG...TTTCTATATGATGAGTTTAGCATTCTGCAAT
## [5]   189 CACGACTGGATGCAGATCGCTGCAGAGACCA...CCTCAGGTGGCCTGGATCTTCGTGGCTGTGG
## [6]   246 TTATAAAATGTAACTTGTAATTATACTGTAG...AAACTCTTGCATGAGAAGAGTTTTAGCACTG
```

```
cloningPrimer <- "AACATTTGTCCTAATCATTGGGGATA"
cheese.clipped <- trimLRPatterns(Lpattern = cloningPrimer, subject=cheese.filtered,
                                 max.Lmismatch = 0.33)
head(sread(cheese.clipped))
```

```
##   A DNAStringSet instance of length 6
##     width seq
## [1]   147 CAGGGATTTGATAGATCATTTCCCTATCCTC...TTTCTGCAAATCCTTTTTCTAAAAGATAGAG
## [2]   200 GAAAATAAAGAGAGAAAACAAACAAACAAAT...GAAAGAAACAGAGAAGAAGAAAGAAAAAAAA
## [3]   198 GGTGACCTTTTTTTTTTTTTTTTTTCAAGAA...AGACTTGGCAAACAGAAAAGGAGGGTATCAA
## [4]   123 TTTTATTTGTTTTTTGATGCTATAGTAAATG...TTTCTATATGATGAGTTTAGCATTCTGCAAT
## [5]   189 CACGACTGGATGCAGATCGCTGCAGAGACCA...CCTCAGGTGGCCTGGATCTTCGTGGCTGTGG
```

```
## [6]   246 TTATAAAATGTAACTTGTAATTATACTGTAG...AAACTCTTGCATGAGAAGAGTTTTAGCACTG
```

There is of course a lot more that can be done to allow for perfect, imperfect and truncated primer sequences or adapters at either the forward or reverse ends of the sequence.

One of the reasons why the `fastq` file format has become so popular in genomics is that in addition to storing information relating to the DNA sequence, the file stores information on the DNA sequence and the quality (confidence score that the base has been measured correctly).

The quality scores in a ShortRead data collection are accessible using the `quality()` function.

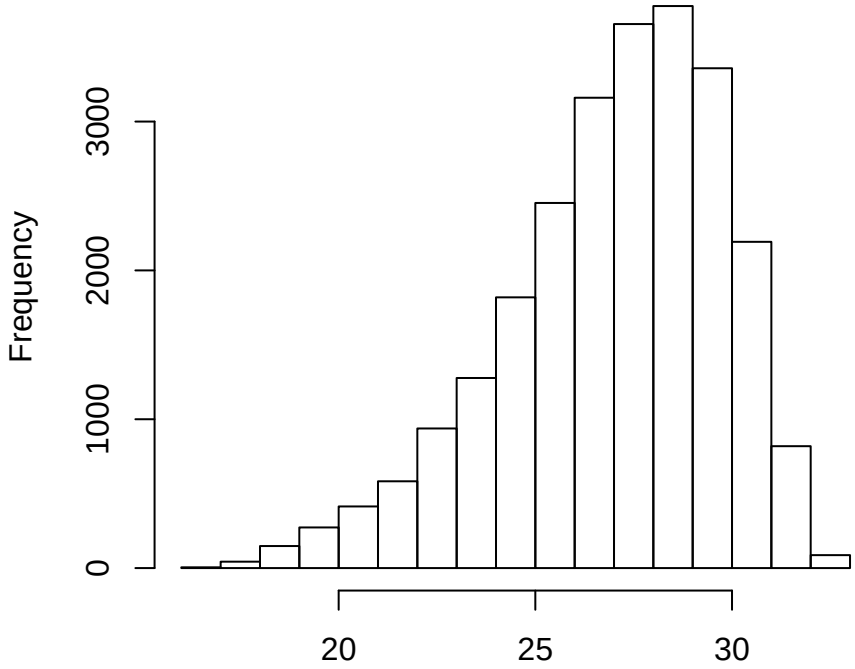
```
head(quality(cheese.clipped))
```

```
## class: FastqQuality
## quality:
##   A BStringSet instance of length 6
##     width seq
## [1]   147 99@B:999-499?9999999049599=B=>;...//*//////+4924444+44444,4444//6
## [2]   200 A>>>4999-999A9999/444-444049967...77<6;<;:;@@<>A:<=B777.777777777
## [3]   198 A@BA;;2444999944444444444"4424;@...6>@AA@@>79969;9@@@4><999-7@?888
## [4]   123 499+444-444444(4;9=@CB@==44408>...8828==ACB<;9>>99909@8-,*,,,//,4
## [5]   189 ?@>1//8<1/9BCCBDDC@@ADCCDCDDD@D...;,4444844242442//88,/0443.3488,
## [6]   246 4-44444,444989@=;;@;>311>>>CCCC...99198774773333339771111,1111116
```

```
encoding(quality(cheese.clipped))
```

```
##  !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /  0  1  2  3  4  5  6  7  8  9
##  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
##  :  ;  <  =  >  ?  @  A  B  C  D  E  F  G  H  I  J
## 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
```

```
cheese.quality <- as(quality(cheese.clipped), "matrix")
hist(rowMeans(cheese.quality[sample(nrow(cheese.quality), 25000),],na.rm=TRUE),
     main="", xlab="")
```



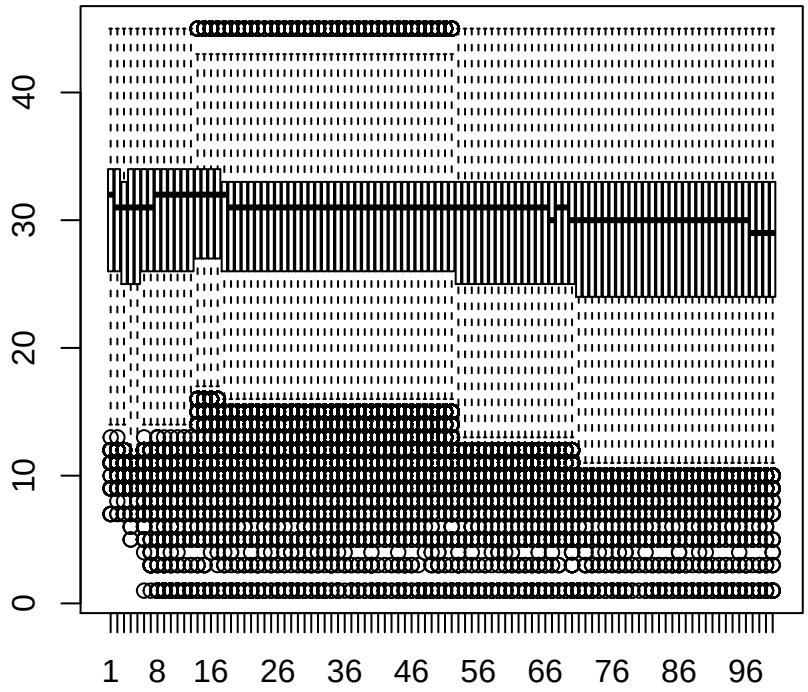
1. Type in the commands above.
2. What *data type* is the variable `cheese.quality`? _____
3. What is the dimension of `cheese.quality`? _____
4. How does the number of columns relate to the length of the reads? _____
5. Below are the broken-down parts of the `hist(...)` function that produced the plot above. The list starts with the most inner function and works it's way out. Can you describe what each inner function is doing and the expected result? The letter in the curly braces e.g. {a} means substitute the result from the part {a} into this function.

a. `nrow(cheese.quality)`: _____
b. `sample({a}, 25000)` : _____
c. `cheese.quality[{b},]` : _____
d. `rowMeans({c}, na.rm=TRUE)`: _____
e. `hist({d}, main="", xlab="")`: _____

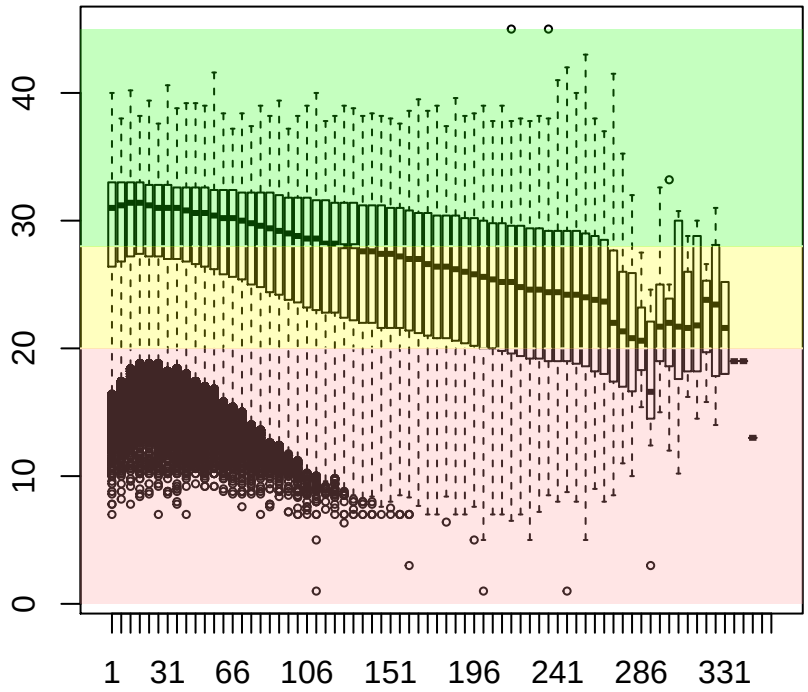
The code above presents a text format (representation of the integer based quality scores) of the quality data. This is converted into a qualityMatrix that can then be plotted to show the distribution of quality scores across the sequence collection.

Plotting the per-base quality scores is the goal of many software applications and this can be managed simply in R as well. One of the requirements to produce these plots is however, to reduce the dimensions of the data. Plotting the characteristics for every read is unnecessary since a sample should convey trends within the data.

```
boxplot(cheese.quality[sample(nrow(cheese.quality), 25000), 1:100],out.cex=0.5)
```




```
mung <- function(i) {
  lower <- i
  upper <- i + 4
  if (upper > ncol(cheese.quality)) {
    upper <- ncol(cheese.quality)
  }
  return(rowMeans(cheese.quality[,seq(lower,upper)], na.rm=TRUE))
}

sequence <- seq(1, ncol(cheese.quality), 5)
groupedQuality <- as.data.frame(sapply(sequence, mung))
colnames(groupedQuality) <- as.character(sequence)

boxplot(groupedQuality[sample(nrow(groupedQuality), 25000),], outcex=0.5)
abline(h=c(20,28),lty=2,col='white')
rect(-5,28,358,45,col=rgb(0.1,1,0,alpha=0.25),border=NA)
rect(-5,20,358,28,col=rgb(1,1,0,alpha=0.25),border=NA)
rect(-5,0,358,20,col=rgb(1,0.6,0.6,alpha=0.25),border=NA)
```

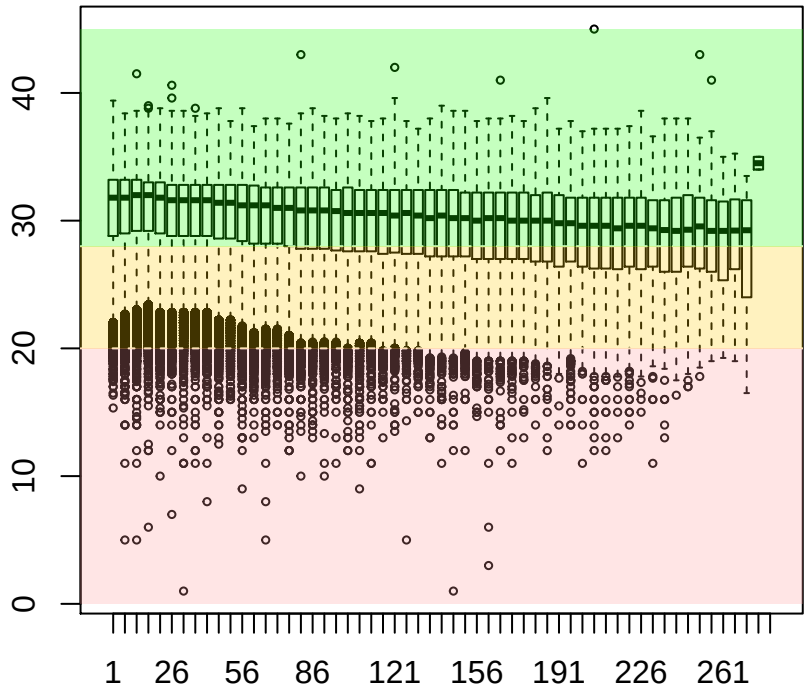


Having prepared this box-and-whisker plot we can see that the distribution of quality scores is generally OK but there are quite a large number of low quality bases described as the interquartile ranges within the whiskers. To make the data most valuable to the analysis it would be worthwhile to filter the sequences to exclude the substrings of lowest quality from the 3'-end forwards. The `ShortRead` package implements some easy to use functions such as `trimTails` and `trimTailw`.

```
cheese.clipped <- trimTailw(cheese.clipped, k=4, a="4", halfwidth=5)
cheese.quality <- as(quality(cheese.clipped), "matrix")
dim(cheese.quality)
```

```
## [1] 160256    284
```

```
sequence <- seq(1, ncol(cheese.quality), 5)
groupedQuality <- as.data.frame(sapply(sequence, mung))
colnames(groupedQuality) <- as.character(sequence)
boxplot(groupedQuality[sample(nrow(groupedQuality), 25000),], outcex=0.5)
abline(h=c(20,28),lty=2,col='white')
rect(-5,28,358,45,col=rgb(0.1,1,0,alpha=0.25),border=NA)
rect(-5,20,358,28,col=rgb(1,0.8,0,alpha=0.25),border=NA)
rect(-5,0,358,20,col=rgb(1,0.6,0.6,alpha=0.25),border=NA)
```



This shows that we have greatly improved the overall structure of the data and have removed a large number of the less-than-perfect bases. This example may have been a little more aggressive than we would really wish to apply in the laboratory setting.

The `ShortRead` package has a well implemented framework for filtering sequences and it would be simple to implement other filters as required. The filter can be illustrated by filtering out the homopolymers from the sequence collection. These provide mechanisms for restricting the sequence collection on the basis of e.g. base composition, number of N-residues and even relative abundance.

A whole lot more can be done using the `ShortRead` and `Biostrings` packages.

Prepare some QC information for an Illumina data collection

Using the RNA-Seq reads that you will perform differential expression with, prepare a brief report of the data. Using the `qaSummary` function to prepare a synopsis of the library content and investigate the per-base quality scores across the sequence collection. Would you recommend that we re-run the analysis after some data trimming?

Have a look at the over-represented k-mer words in the sequence collection and see if you can create a filter to strip-out the sequence that contain the most abundant k-mer.

Chapter 2

RNAseq data analysis

2.1 Data pre-processing

To start, create a new R project:

1. Click on **File** > **New Project**
2. Click on **New Directory** > **Empty Project**
3. Enter a name in **Directory name** (e.g. RNAseq_analysis). Click on **Create Project**. Wait for the project to be created and the page will refresh.
4. Check in the **Files** tab on the bottom-right corner. You should now see that you are in the **Home** > **RNAseq_analysis** directory. This is the current working directory and in here, there should be a new file called **RNAseq_analysis.Rproj**.

The data considered for the RNAseq part of the workshop is BioProject PRJEB5297. It is available via ENA (http://www.ebi.ac.uk/ena/data/view/PRJEB5297) or NCBI (https://www.ncbi.nlm.nih.gov/bioproject/PRJEB5297). The study corresponds to 8 RNA sequencing libraries from Human brain and liver.

> ***Allele-Specific Expression in Human Brain and Liver*** *Systematic survey of gene and isoform allele-specific expression in human brain and liver tissues, and description of optimised bioinformatic and statistical methods to accurately measure allele-specific expression.*

Our biological question for this study:

What are the list of genes differentially expressed between the *liver* and the *brain* samples?

Raw sequencing data are usually available in FASTQ format, which is a well defined text-based format for storing both biological sequences (usually nucleotide sequences) and their corresponding quality scores. The raw data from this study have been downloaded (8Gb / fastq file) into the shared directory `../data/RNAseq/raw_data`.

To see a list of files in this directory, enter the following commands:

```
RNASeq_DATA_DIR <- "../data/RNAseq/raw_data"
dir(RNASeq_DATA_DIR)
```

```
##  [1] "ERR420386_1.fastq.gz"         "ERR420386_1.subset1.fastq.gz"
##  [3] "ERR420386_1.subset2.fastq.gz" "ERR420386_1.subset3.fastq.gz"
##  [5] "ERR420386_1.subset4.fastq.gz" "ERR420386_1.subset5.fastq.gz"
```

```
##  [7] "ERR420386_1.subset6.fastq.gz" "ERR420386_1.subset7.fastq.gz"
##  [9] "ERR420386_2.fastq.gz"         "ERR420386_chr21_R1.fastq.gz"
## [11] "ERR420386_chr21_R2.fastq.gz"  "ERR420386_chr21.fastq.gz"
## [13] "ERR420387_1.fastq.gz"         "ERR420387_2.fastq.gz"
## [15] "ERR420388_1.fastq.gz"         "ERR420388_2.fastq.gz"
## [17] "ERR420389_1.fastq.gz"         "ERR420389_2.fastq.gz"
## [19] "ERR420390_1.fastq.gz"         "ERR420390_2.fastq.gz"
## [21] "ERR420391_1.fastq.gz"         "ERR420391_2.fastq.gz"
## [23] "ERR420392_1.fastq.gz"         "ERR420392_2.fastq.gz"
## [25] "ERR420393_1.fastq.gz"         "ERR420393_2.fastq.gz"
## [27] "experiment_design.txt"
```

The first step in a RNAseq analysis is to run a quick quality check on your data, this will give you an idea of the quality of your raw data in terms of number of reads per library, read length, average quality score along the reads, GC content, sequence duplication level, adaptors that might have not been removed correctly from the data etc.

The `fastQC` tool is quick and easy to run and can be downloaded from here: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/.

To ensure highest quality of the sequences for subsequent mapping and differential expression analysis steps, the reads can also be trimmed using the `Trimmomatic` tool (Lohse et al. 2012).

For the scope of this course we will focus on the R-based steps and will assume that the data are fit for purpose.

```
RNASeq_REF_DATA_DIR <- "../data/RNAseq/ref_data"

REF_GENOME <- file.path(RNASeq_REF_DATA_DIR, "hg19.fa")
RSUBREAD_INDEX_PATH <- file.path(RNASeq_REF_DATA_DIR, "RsubreadIndex")

RSUBREAD_INDEX_BASE <- "hg19"

RESULTS_DIR <- 'results/RNAseq'
MAPPING_DIR <- file.path(RESULTS_DIR,'mapping')
dir.create(MAPPING_DIR, recursive=T)
```

2.2 Mapping

Once the reads have been quality checked and trimmed, the next step is to map the reads to the reference genome (in our case the human genome "hg19"). This can be done with the Bioconductor package `Rsubread` Y et al. (2013).

Before mapping the reads to the reference genome you will need to build a Rsubread index for that genome. Below are the commands for building an index for the human reference genome using the `buildindex` command.

PLEASE DO NOT RUN the `buildindex()` code in the workshop as this can take awhile. We have already build the index for you.

```
library(Rsubread)
buildindex(basename=file.path(RSUBREAD_INDEX_BASE, RSUBREAD_INDEX_PATH),
reference=REF_GENOME)
```

Once the Rsubread index has been created you can map your reads to the genome by running the `align` command. The code below could be used to map the reads for a specific library against the "hg19" genome.

```
library(Rsubread)
sample <- "ERR420386"

inputFileFWD <- file.path(RNASeq_DATA_DIR, paste0(sample,"_chr21_R1.fastq.gz"))
inputFileRVS <- file.path(RNASeq_DATA_DIR, paste0(sample,"_chr21_R2.fastq.gz"))

output.bamFile <- file.path(MAPPING_DIR, paste0(sample,".bam"))

inputFileFWD
inputFileRVS
output.bamFile
```

```
## [1] "../data/RNAseq/raw_data/ERR420386_chr21_R1.fastq.gz"
## [1] "../data/RNAseq/raw_data/ERR420386_chr21_R2.fastq.gz"
## [1] "results/RNAseq/mapping/ERR420386.bam"
```

For the purpose of this workshop the mapping has already been done. This step can take up to a couple of hours per library.

Please only run the following command using the subset sample `*_chr21_R1.fastq.gz`, which is much smaller.

```
align(index = file.path(RSUBREAD_INDEX_PATH, RSUBREAD_INDEX_BASE),
     readfile1 = inputFileFWD,
     readfile2 = inputFileRVS,
     input_format = "gzFASTQ",
     output_file = output.bamFile,
     output_format = "BAM")
```

```
##
##          ==========     _____ _     _ ____  _____  ______          _____
##          =====         / ____| |   | |  _ \|  __ \|  ____|   /\   |  __ \
##            =====      | (___ | |   | | |_) | |__) | |__     /  \  | |  | |
##              ====      \___ \| |   | |  _ <|  _  /|  __|   / /\ \ | |  | |
##                ====    ____) | |__| | |_) | | \ \| |____ / ____ \| |__| |
##          ==========   |_____/ \____/|____/|_|  \_\______/_/    \_\_____/
##         Rsubread 1.24.2
##
## //========================== subread-align setting ==========================\\
## ||                                                                           ||
## || Function      : Read alignment (RNA-Seq)                                 ||
## || Input file 1  : ../data/RNAseq/raw_data/ERR420386_chr21_R1.fastq.gz      ||
## || Input file 2  : ../data/RNAseq/raw_data/ERR420386_chr21_R2.fastq.gz      ||
## || Output file   : results/RNAseq/mapping/ERR420386.bam (BAM)               ||
## || Index name    : ../data/RNAseq/ref_data/RsubreadIndex/hg19               ||
## ||                                                                           ||
```

```
## ||                    ------------------------------------                    ||
## ||                                                                            ||
## ||                        Threads : 1                                         ||
## ||                   Phred offset : 33                                        ||
## ||       # of extracted subreads : 10                                        ||
## ||                Min read1 vote : 3                                         ||
## ||                Min read2 vote : 1                                         ||
## ||             Max fragment size : 600                                       ||
## ||             Min fragment size : 50                                        ||
## ||    Maximum allowed mismatches : 3                                         ||
## ||   Maximum allowed indel bases : 5                                         ||
## || # of best alignments reported : 1                                         ||
## ||                Unique mapping : yes                                       ||
## ||                                                                            ||
## \\==================== http://subread.sourceforge.net/ ====================//
##
## //================= Running (12-Apr-2017 22:40:54, pid=1773) =================\\
## ||                                                                            ||
## || The input file contains base space reads.                                 ||
## || The range of Phred scores observed in the data is [2,40]                  ||
## || Load the 1-th index block...                                              ||
## ||    4% completed, 0.2 mins elapsed, rate=3.9k fragments per second         ||
## ||   10% completed, 0.3 mins elapsed, rate=4.0k fragments per second         ||
## ||   17% completed, 0.3 mins elapsed, rate=3.9k fragments per second         ||
## ||   24% completed, 0.4 mins elapsed, rate=3.9k fragments per second         ||
## ||   30% completed, 0.4 mins elapsed, rate=3.9k fragments per second         ||
## ||   37% completed, 0.4 mins elapsed, rate=3.9k fragments per second         ||
## ||   44% completed, 0.5 mins elapsed, rate=3.7k fragments per second         ||
## ||   50% completed, 0.6 mins elapsed, rate=3.7k fragments per second         ||
## ||   57% completed, 0.6 mins elapsed, rate=3.7k fragments per second         ||
## ||   64% completed, 0.7 mins elapsed, rate=3.7k fragments per second         ||
## ||   69% completed, 0.7 mins elapsed, rate=2.7k fragments per second         ||
## ||   73% completed, 0.7 mins elapsed, rate=2.8k fragments per second         ||
## ||   76% completed, 0.7 mins elapsed, rate=2.9k fragments per second         ||
## ||   80% completed, 0.7 mins elapsed, rate=2.9k fragments per second         ||
## ||   83% completed, 0.7 mins elapsed, rate=3.0k fragments per second         ||
## ||   86% completed, 0.8 mins elapsed, rate=3.1k fragments per second         ||
## ||   90% completed, 0.8 mins elapsed, rate=3.1k fragments per second         ||
## ||   93% completed, 0.8 mins elapsed, rate=3.2k fragments per second         ||
## ||   96% completed, 0.8 mins elapsed, rate=3.2k fragments per second         ||
## ||                                                                            ||
## ||                          Completed successfully.                           ||
## ||                                                                            ||
## \\============================================================================//
##
## //================================== Summary ==================================\\
## ||                                                                            ||
## ||          Processed : 161,297 fragments                                     ||
## ||             Mapped : 160,518 fragments (99.5%)                             ||
## ||   Correctly paired : 126,376 fragments                                     ||
## ||             Indels : 2,635                                                 ||
## ||                                                                            ||
## ||       Running time : 0.8 minutes                                           ||
## ||                                                                            ||
## \\==================== http://subread.sourceforge.net/ ====================//
```

The `nthreads` parameter can be used in the align command to speed up the process and run the alignment using several CPUs in parallel.

The function `propmapped` returns the proportion of mapped reads in the output SAM file: total number of input reads, number of mapped reads and proportion of mapped reads.

```
propmapped(output.bamFile)
```

```
## The input file is opened as a BAM file.
## The fragments in the input file are being counted.
## Finished. All records: 322594; all fragments: 161297; mapped fragments: 160518; the
mappability is 99.52%
##                                Samples NumTotal NumMapped PropMapped
## 1 results/RNAseq/mapping/ERR420386.bam   161297    160518    0.99517
```

You can run the `propmapped()` on multiple `bam` files to return a summary of the total number of reads per file and the number of reads that were mappable or unmappable. **However**, this can take a very long time to run for big bam files.

PLEASE DO NOT RUN

For example:

```
all.bam.files <- grep('.bam',dir('../data/RNAseq/mapping',full.names =
T),value=T)
pm <- propmapped(all.bam.files)
```

2.2.1 Examining the mapped reads

Create a `BamFile` object and load the file into memory so we can interact with it and find out some information. The `seqinfo()` function outputs the headding information, in this exercise, this is the

```
library(Rsamtools)
bf <- BamFile(output.bamFile)
seqinfo(bf)
```

```
## Seqinfo object with 25 sequences from an unspecified genome:
##   seqnames seqlengths isCircular genome
##   chrM          16571       <NA>   <NA>
##   chr1      249250621       <NA>   <NA>
##   chr2      243199373       <NA>   <NA>
##   chr3      198022430       <NA>   <NA>
##   chr4      191154276       <NA>   <NA>
##   ...             ...        ...    ...
##   chr20      63025520       <NA>   <NA>
##   chr21      48129895       <NA>   <NA>
##   chr22      51304566       <NA>   <NA>
##   chrX      155270560       <NA>   <NA>
##   chrY       59373566       <NA>   <NA>
```

We can take a closer look and find out how many of the reads map to each chromosome. To do this, we need to first sort and index the `bam` file.

```
output.sorted.bamFile <- file.path(MAPPING_DIR,paste0(sample, '.sorted'))
sortBam(output.bamFile, output.sorted.bamFile)
```
```
## [1] "results/RNAseq/mapping/ERR420386.sorted.bam"
```

```
output.sorted.bamFile <- paste0(output.sorted.bamFile, ".bam")
indexBam(output.sorted.bamFile)
```
```
##       results/RNAseq/mapping/ERR420386.sorted.bam
## "results/RNAseq/mapping/ERR420386.sorted.bam.bai"
```

```
dir(MAPPING_DIR, full.names=T)
```
```
## [1] "results/RNAseq/mapping/ERR420386.bam"
## [2] "results/RNAseq/mapping/ERR420386.bam.indel"
## [3] "results/RNAseq/mapping/ERR420386.sorted.bam"
## [4] "results/RNAseq/mapping/ERR420386.sorted.bam.bai"
```

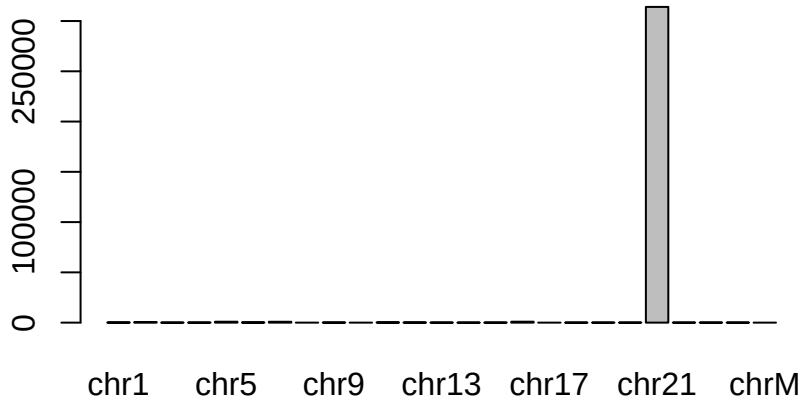
```
output.bam.index <- dir(MAPPING_DIR, full.names=T)[grep(".bai",dir(MAPPING_DIR))]
output.bam.index
```
```
## [1] "results/RNAseq/mapping/ERR420386.sorted.bam.bai"
```

Once the index bam file has been created, we can find out the number of mapped reads per chromosome:

```
chr.mapping.stats <- idxstatsBam(output.bamFile, index=output.bam.index)
chr.mapping.stats
```
```
##    seqnames seqlength mapped unmapped
## 1      chr1 249250621    150        0
## 2      chr2 243199373    559        0
## 3      chr3 198022430     20        0
## 4      chr4 191154276     79        0
## 5      chr5 180915260    892        0
## 6      chr6 171115067    200        0
## 7      chr7 159138663    850        0
## 8      chr8 146364022      7        0
## 9      chr9 141213431    101        0
## 10    chr10 135534747     10        0
## 11    chr11 135006516    212        0
## 12    chr12 133851895    155        0
## 13    chr13 115169878     42        0
## 14    chr14 107349540     19        0
## 15    chr15 102531392     56        0
## 16    chr16  90354753    924        0
## 17    chr17  81195210      6        0
## 18    chr18  78077248     27        0
## 19    chr19  59128983     24        0
## 20    chr20  63025520     20        0
## 21    chr21  48129895 314047        0
## 22    chr22  51304566     15        0
## 23     chrX 155270560     46        0
## 24     chrY  59373566     66        0
## 25     chrM     16571      0        0
```

This is easiest to view as a plot:

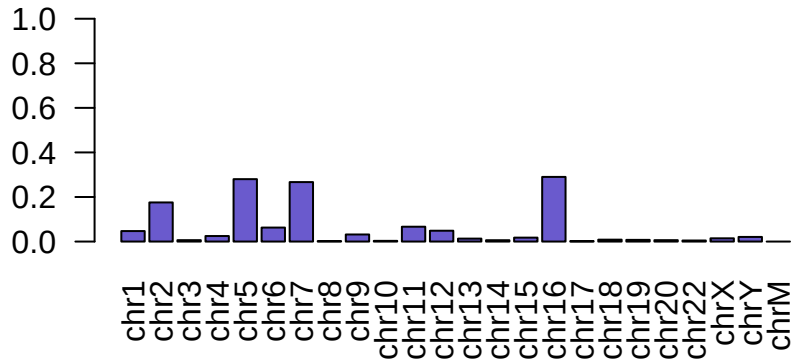
```
rownames(chr.mapping.stats) <- chr.mapping.stats$seqnames
barplot(chr.mapping.stats$mapped,
        names.arg=as.character(chr.mapping.stats$seqnames))
```



Mapping percentage

Using a barplot, can you find out which other chromsome has the highest number of mapped reads?

Hint: repeat the barplot but without the bar for `chr21`.



Solution

```
total.mapped.reads <- sum(chr.mapping.stats$mapped)
```

```
chr.mapping.stats$mapped.prop <- chr.mapping.stats$mapped/total.mapped.reads*100
barplot(chr.mapping.stats$mapped.prop[-21],
names.arg=chr.mapping.stats$seqnames[-21],
las=2, col='slateblue', ylim=c(0,1))
```

2.3 Quantification

Rsubread provides a read summarization function `featureCounts`, which takes two inputs:

1. the aligned reads (BAM or SAM) and assigns them to
2. genomic features (GTF annotation file)

This gives the number of reads mapped per feature, which can then be normalised and tested for differential expression.

Rsubread comes with in-built annotations for *mm*9, *mm*10 and *hg*19 for users' convenience, but you can also supply your own annotation file (GTF), see the tip below.

For experiments with lots of samples or on big genomes, this step can also take a while. We will only be performing the feature counts on the subset `chr21` BAM file we created previously.

```
mini.counts <- featureCounts(output.bamFile,
                        annot.inbuilt="hg19",
                        isGTFAnnotationFile=FALSE,
                        isPairedEnd=TRUE)
## NCBI RefSeq annotation for hg19 (build 37.2) is used.
##
##          ==========     _____ _    _ ____  _____  ______          _____
##          =====         / ____| |  | |  _ \|  __ \|  ____|   /\   |  __ \
##            =====      | (___ | |  | | |_) | |__) | |__     /  \  | |  | |
##              ====      \___ \| |  | |  _ <|  _  /|  __|   / /\ \ | |  | |
##                ====    ____) | |__| | |_) | | \ \| |____ / ____ \| |__| |
##          ==========   |_____/ \____/|____/|_|  \_\______/_/    \_\_____/
##         Rsubread 1.24.2
##
## //========================== featureCounts setting ===========================\\
## ||                                                                            ||
## ||             Input files : 1 BAM file                                       ||
## ||                           P results/RNAseq/mapping/ERR420386.bam           ||
## ||                                                                            ||
## ||      Dir for temp files : .                                                ||
## ||                 Threads : 1                                                ||
## ||                   Level : meta-feature level                               ||
## ||              Paired-end : yes                                              ||
## ||         Strand specific : no                                               ||
## ||      Multimapping reads : not counted                                      ||
## || Multi-overlapping reads : not counted                                      ||
## ||   Min overlapping bases : 1                                                ||
## ||                                                                            ||
## ||          Chimeric reads : counted                                          ||
## ||        Both ends mapped : not required                                     ||
## ||                                                                            ||
## \\====================== http://subread.sourceforge.net/ ======================//
```

```
##
## //================================ Running ================================\\
## ||                                                                         ||
## || Load annotation file /usr/lib64/R/library/Rsubread/annot/hg19_RefSeq_e ... ||
## ||    Features : 225074                                                    ||
## ||    Meta-features : 25702                                                ||
## ||    Chromosomes/contigs : 52                                             ||
## ||                                                                         ||
## || Process BAM file results/RNAseq/mapping/ERR420386.bam...                ||
## ||    Paired-end reads are included.                                       ||
## ||    Assign fragments (read pairs) to features...                         ||
## ||    Total fragments : 161297                                             ||
## ||    Successfully assigned fragments : 72440 (44.9%)                      ||
## ||    Running time : 0.01 minutes                                          ||
## ||                                                                         ||
## ||                          Read assignment finished.                      ||
## ||                                                                         ||
## \\================== http://subread.sourceforge.net/ ====================//
```

Examine the attributes in the returned `mini.counts` object:

```
summary(mini.counts)
```

```
##            Length Class      Mode
## counts     25702  -none-     numeric
## annotation     6  data.frame list
## targets        1  -none-     character
## stat           2  data.frame list
```

Find the dimension of the `counts` table:

```
dim(mini.counts$counts)
```

```
## [1] 25702     1
```

```
mini.counts$counts[1:6,]
```

```
##    653635 100422834    645520     79501    729737 100507658
##         0         0         0         0         0         0
```

Look at the annotations, which corresponds to the rows of the `counts` table:

```
head(mini.counts$annotation)
```

```
##      GeneID                                                          Chr
## 1    653635 chr1;chr1;chr1;chr1;chr1;chr1;chr1;chr1;chr1;chr1;chr1
## 2 100422834                                                          chr1
## 3    645520                                              chr1;chr1;chr1
## 4     79501                                                          chr1
## 5    729737                                         chr1;chr1;chr1;chr1
## 6 100507658                                              chr1;chr1;chr1
##                                                                    Start
## 1 14362;14970;15796;16607;16858;17233;17606;17915;18268;24738;29321
## 2                                                                30366
## 3                                                34611;35277;35721
## 4                                                                69091
## 5                                      136698;136953;139790;140075
## 6                                             319944;320881;321032
##                                                                      End
## 1 14829;15038;15947;16765;17055;17368;17742;18061;18366;24891;29370
## 2                                                                30503
```

```
## 3                                                    35174;35481;36081
## 4                                                                70008
## 5                                              136805;139696;139847;140566
## 6                                                    320653;320938;321056
##                   Strand Length
## 1 -;-;-;-;-;-;-;-;-;-;-   1769
## 2                     +    138
## 3                   -;-;-   1130
## 4                     +    918
## 5                 -;-;-;-   3402
## 6                   +;+;+    793
```

`featureCounts` also returns a very hand summary of the number of reads that were *assigned* or *unassiged*:

```
mini.counts$stat
```

```
##                      Status results.RNAseq.mapping.ERR420386.bam
## 1                  Assigned                                72440
## 2       Unassigned_Ambiguity                                  227
## 3    Unassigned_MultiMapping                                    0
## 4      Unassigned_NoFeatures                                87851
## 5        Unassigned_Unmapped                                  779
## 6  Unassigned_MappingQuality                                    0
## 7  Unassigned_FragmentLength                                    0
## 8        Unassigned_Chimera                                     0
## 9      Unassigned_Secondary                                     0
## 10   Unassigned_Nonjunction                                     0
## 11     Unassigned_Duplicate                                     0
```

1. Run the above command to perform counting reads for the bam file and then take a look at the summary output.
2. Lookup the user guide for Rsubread and find the defintions for the status in `mini.counts$stat` table, specifically `Unassigned_Ambiguity`, `Unassigned_NoFeatures` and `Unassigned_Unmapped`.

If you want to get read counts using another annotation.GTF file, use the `annot.ext` parameter. For example: counts <- featureCounts(output.bamFile, annot.ext="annotation.GTF", isGTFAnnotationFile=TRUE, isPairedEnd=TRUE)

For the purpose of this workshop the read summarisation step has already been performed for all libraries. You will need to load the corresponding `Rdata` file to get these read counts. You can then print out these counts in a text file for future use.

To load the Rdata object file:

```
load("../data/RNAseq/quantification/rawCounts.Rdata")
summary(counts)
```

```
##            Length Class      Mode
## counts     205616 -none-     numeric
## annotation      6 data.frame list
```

```
## targets         8 -none-     character
## stat            9 data.frame list
```

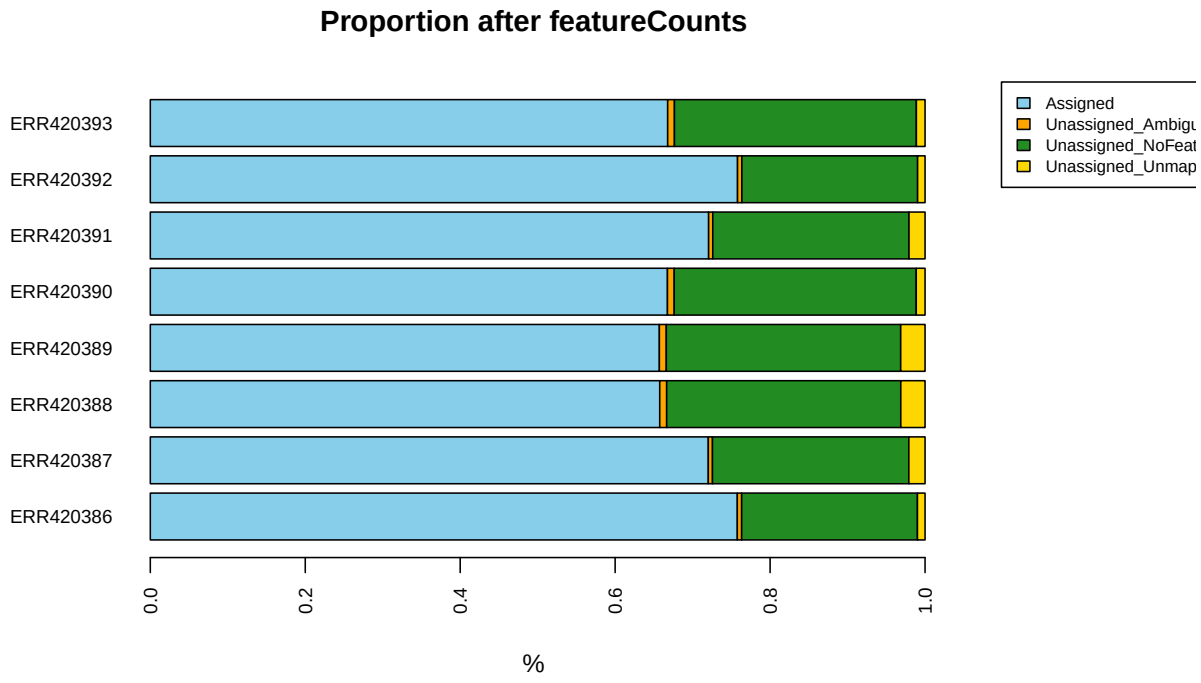
Since we will be using it later for DE analysis, let's create a `raw.counts` object to hold only the count data:

```
raw.counts <- counts$counts
dim(raw.counts)
```
```
## [1] 25702     8
```

```
raw.counts[1:6,1:4]
```
```
##           ERR420386 ERR420387 ERR420388 ERR420389
## 653635           47       151        93       112
## 100422834         1         0         0         0
## 645520            1         2         0         0
## 79501             0         0         0         0
## 729737           15        35        47        57
## 100507658        10         5        15         8
```

Challenge

1. What are the count statistics for the "Brain and Liver" study?
2. create a copy of the stats object and change the counts into proportion per sample.
3. The `stat` information from `featureCounts()` will be much easier to digest if we can plot the proportion (or number) of reads assigned and unassigned due to the different status type. Can you create the following plot?



Solution

All lines starting with ## are comments.

```
##------------------------------------------------------------------------
## Create a new matrix to just hold the statistics data and remove the first
## column, making this the rownames of the table instead

stats <- counts$stat
rownames(stats) <- stats$Status
stats <- stats[,-1]

##------------------------------------------------------------------------
## Double check the `stats` matrix. Notice how there are many rows where all
## are 0, we do not want to bother showing this in the plot, so we will just
## remove them from the matrix. Remember to double check the matrix again.

stats
rows.remove <- which(rowSums(stats) == 0)
stats <- stats[-rows.remove,]
dim(stats)
stats

##------------------------------------------------------------------------
## There are two ways to calculate the proportion, the first is using a
## user-defined function and then `apply()`, the second is using a new
## function `sweep()`.
##
## Only pick one approach.

##### Approach 1
## Given a vector, we want to divide each element by the sum of that vector.
## Create your own function to do this and use `apply()` to call this
## function, acting on each column of the matrix.

proportion <- function(x) { x/sum(x) }
stats.prop <- apply(stats, 2, proportion)

#### Approach 2
## The alternative approach using the `sweep()` function does not required
## a user-defined function:

stats.prop <- sweep(stats, 2, colSums(stats), "/")

##------------------------------------------------------------------------
## Create the barplot and rotate the plot using `horiz=T`.

par(mar=c(5.1, 5, 4.1, 9), xpd=TRUE)
barplot(as.matrix(stats.prop),las=2, cex.names=0.75,cex.axis=0.75, horiz = T,
        main="Proportion after featureCounts",
        xlab="%",
        col=c('skyblue','orange','forestgreen','gold'))
legend("topright", inset=c(-0.4,0), cex = 0.7,
       legend = rownames(stats.prop),
       fill = c('skyblue','orange','forestgreen','gold'))
```

Export out the counts table for every sample into a tab-separated file:

```
write.table(counts$counts, file=file.path(RESULTS_DIR,"raw_read_counts.txt"),
            sep="\t", quote=F, append=F)
```

2.4 Exploratory analysis

Rsubread provides the number of reads mapped to each gene which can then be used for ploting quality control figures and for differential expression analysis.

QC figures of the mapped read counts can be plotted and investigated for potential outlier libraries and to confirm grouping of samples.

Before plotting QC figures it is useful to get the experiment design. This will allow labeling of the data with the sample groups they belong to, or any other parameter of interest.

The experiment design file corresponding to this study has been downloaded from the ArrayExpress webpage and formatted as a tab separated file for this analysis purposes. You can find it in the shared directory `../data/Data_Analysis_with_R/RNAseq/raw_data`.

```
EXPMT_DESIGN_FILE <- file.path(RNASeq_DATA_DIR, 'experiment_design.txt')

expr.design <- read.table(EXPMT_DESIGN_FILE, header=T, sep='\t')
rownames(expr.design) <- expr.design$SampleID

#order the design in the same ordering as the counts object
expr.design <- expr.design[colnames(counts$counts),]

expr.design
```

```
##            SampleID    Source.Name     organism  sex age tissue
## ERR420386 ERR420386 brain_sample_1 Homo sapiens male  26  brain
## ERR420387 ERR420387 brain_sample_1 Homo sapiens male  26  brain
## ERR420388 ERR420388 liver_sample_1 Homo sapiens male  30  liver
## ERR420389 ERR420389 liver_sample_1 Homo sapiens male  30  liver
## ERR420390 ERR420390 liver_sample_1 Homo sapiens male  30  liver
## ERR420391 ERR420391 brain_sample_1 Homo sapiens male  26  brain
## ERR420392 ERR420392 brain_sample_1 Homo sapiens male  26  brain
## ERR420393 ERR420393 liver_sample_1 Homo sapiens male  30  liver
##           Extract.Name Material.Type Assay.Name technical.replicate.group
## ERR420386       GCCAAT           RNA     Assay4                   group_2
## ERR420387       ACAGTG           RNA     Assay2                   group_1
## ERR420388       GTGAAA           RNA     Assay7                   group_4
## ERR420389       GTGAAA           RNA     Assay8                   group_4
## ERR420390       CTTGTA           RNA     Assay6                   group_3
## ERR420391       ACAGTG           RNA     Assay1                   group_1
## ERR420392       GCCAAT           RNA     Assay3                   group_2
## ERR420393       CTTGTA           RNA     Assay5                   group_3
```

```
samples <- as.character(expr.design$SampleID)
group <- factor(expr.design$tissue)
group
```

```
## [1] brain brain liver liver liver brain brain liver
## Levels: brain liver
```

The samples are in random order and not sorted by the tissue type, this will make visualisation tricker in downstream analysis. We will reorder the samples by tissue type.

```
sample.order <- order(expr.design$tissue)
sample.order
```
```
## [1] 1 2 6 7 3 4 5 8
```

```
expr.design <- expr.design[sample.order,]
raw.counts <- raw.counts[,sample.order]

expr.design
```
```
##            SampleID    Source.Name     organism  sex age tissue
## ERR420386 ERR420386 brain_sample_1 Homo sapiens male  26  brain
## ERR420387 ERR420387 brain_sample_1 Homo sapiens male  26  brain
## ERR420391 ERR420391 brain_sample_1 Homo sapiens male  26  brain
## ERR420392 ERR420392 brain_sample_1 Homo sapiens male  26  brain
## ERR420388 ERR420388 liver_sample_1 Homo sapiens male  30  liver
## ERR420389 ERR420389 liver_sample_1 Homo sapiens male  30  liver
## ERR420390 ERR420390 liver_sample_1 Homo sapiens male  30  liver
## ERR420393 ERR420393 liver_sample_1 Homo sapiens male  30  liver
##           Extract.Name Material.Type Assay.Name technical.replicate.group
## ERR420386       GCCAAT           RNA     Assay4                   group_2
## ERR420387       ACAGTG           RNA     Assay2                   group_1
## ERR420391       ACAGTG           RNA     Assay1                   group_1
## ERR420392       GCCAAT           RNA     Assay3                   group_2
## ERR420388       GTGAAA           RNA     Assay7                   group_4
## ERR420389       GTGAAA           RNA     Assay8                   group_4
## ERR420390       CTTGTA           RNA     Assay6                   group_3
## ERR420393       CTTGTA           RNA     Assay5                   group_3
```

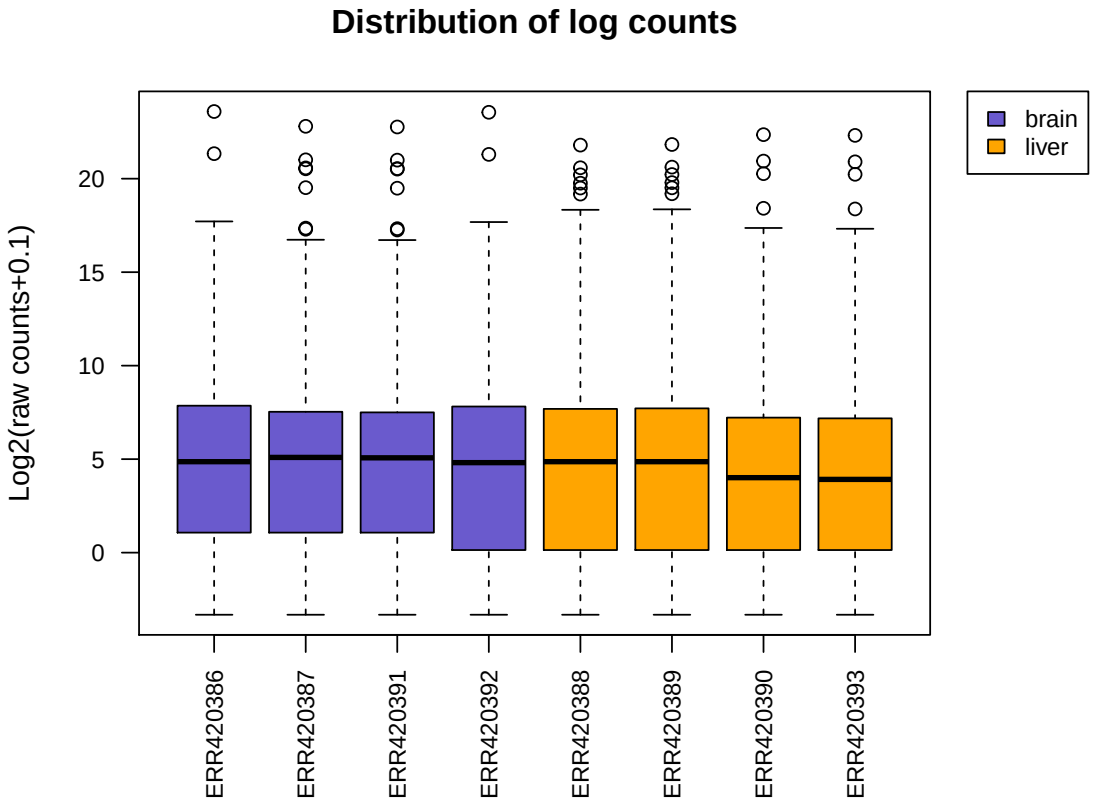
This is will be much easier when we come to visualise our data later. Remember to reassign the groups:

```
group <- expr.design$tissue
group
```
```
## [1] brain brain brain brain liver liver liver liver
## Levels: brain liver
```

2.5 Understanding the dataset

Density plots of log-intensity distribution of each library can be superposed on a single graph for a better comparison between libraries and for identification of libraries with weird distribution. On the boxplots the density distributions of raw log-intensities are not expected to be identical but still not totally different.

```
logcounts <- log2(raw.counts+0.1)

group.colours <- c('slateblue','orange')[group];
par(mar=c(5.1, 5, 4.1, 7), xpd=TRUE)
boxplot(logcounts,
        col=group.colours,
        main="Distribution of log counts",
        xlab="",
        ylab="Log2(raw counts+0.1)",
        las=2,cex.axis=0.8)
legend("topright", inset=c(-0.2,0), cex = 0.8,
       legend = levels(group),
       fill = unique(group.colours))
```



2.6 Prefiltering

Before proceeding with differential expression analysis, it is useful to filter out very lowly expressed genes. This will help increasing the statistical power of the analysis while keeping genes of interest. A common way to do this is by filtering out genes having less than 1 count-per-million reads (cpm) in half the samples.

The `edgeR` Robinson et al. (2009) library provides the `cpm` function which can be used here.

```
library(edgeR)
isexpr <- rowSums(cpm(raw.counts)> 1) >= 4
table(isexpr)
```

```
## isexpr
## FALSE  TRUE
```

```
## 10652 15050
```

```
filtered.raw.counts <- raw.counts[isexpr,]
dim(filtered.raw.counts)
## [1] 15050     8
```

That means that `nrow(raw.counts)-nrow(filtered.raw.counts)` are removed.

2.7 Normalisation

Since we want to make between sample comparisons, we need to normalize the dataset.

2.7.1 Defining the model matrix

`Limma` Ritchie et al. (2015) requires a design matrix to be created for the DE analysis. This is created using `model.matrix()` function and `formula` notation in R. It is required in all linear modeling.

```
design <- model.matrix(~0 + expr.design$tissue, data=expr.design)
colnames(design) <- levels(expr.design$tissue)
design
##           brain liver
## ERR420386     1     0
## ERR420387     1     0
## ERR420388     0     1
## ERR420389     0     1
## ERR420390     0     1
## ERR420391     1     0
## ERR420392     1     0
## ERR420393     0     1
## attr(,"assign")
## [1] 1 1
## attr(,"contrasts")
## attr(,"contrasts")$`expr.design$tissue`
## [1] "contr.treatment"
```

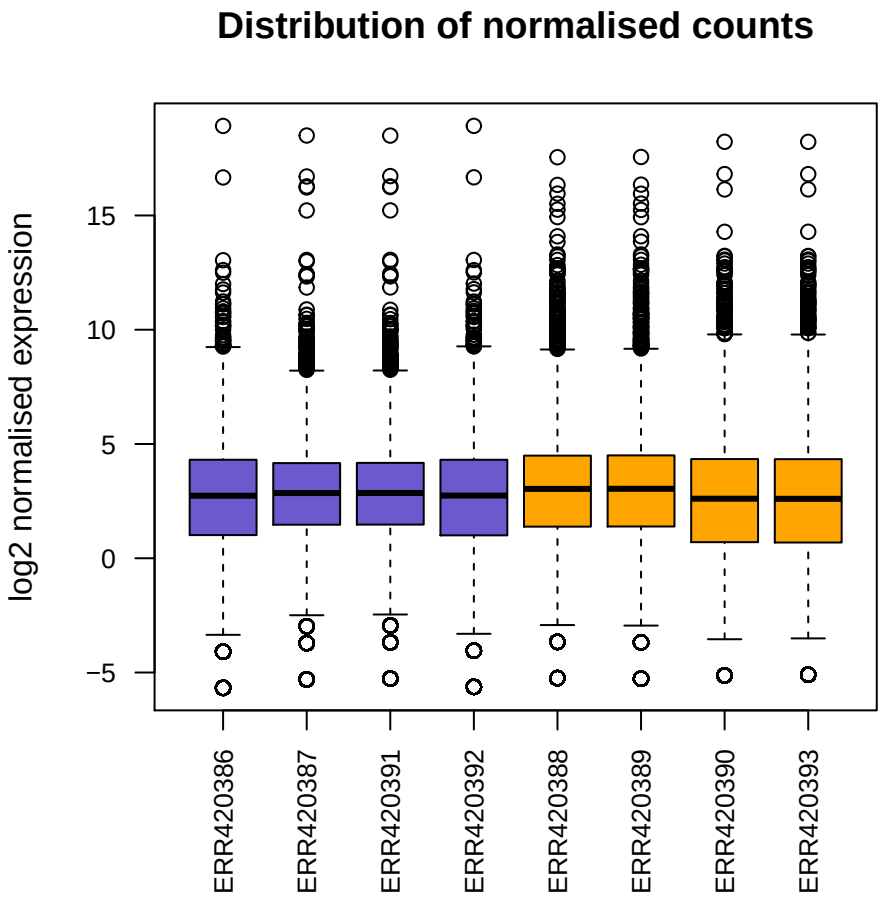
Now, we can normalise the dataset using the following commands. The `calcNormFactors()`, calculates the normalization factors to scale the library sizes.

The `limma` package (since version 3.16.0) offers the `voom` function that will normalise read counts and apply a linear model to the normalised data before computing moderated t-statistics of differential expression.

The returned data object consists of a few attributes, which you can check using `names(y)`, one of which is the *normalised expression* (`y$E`) values in log2 scale.

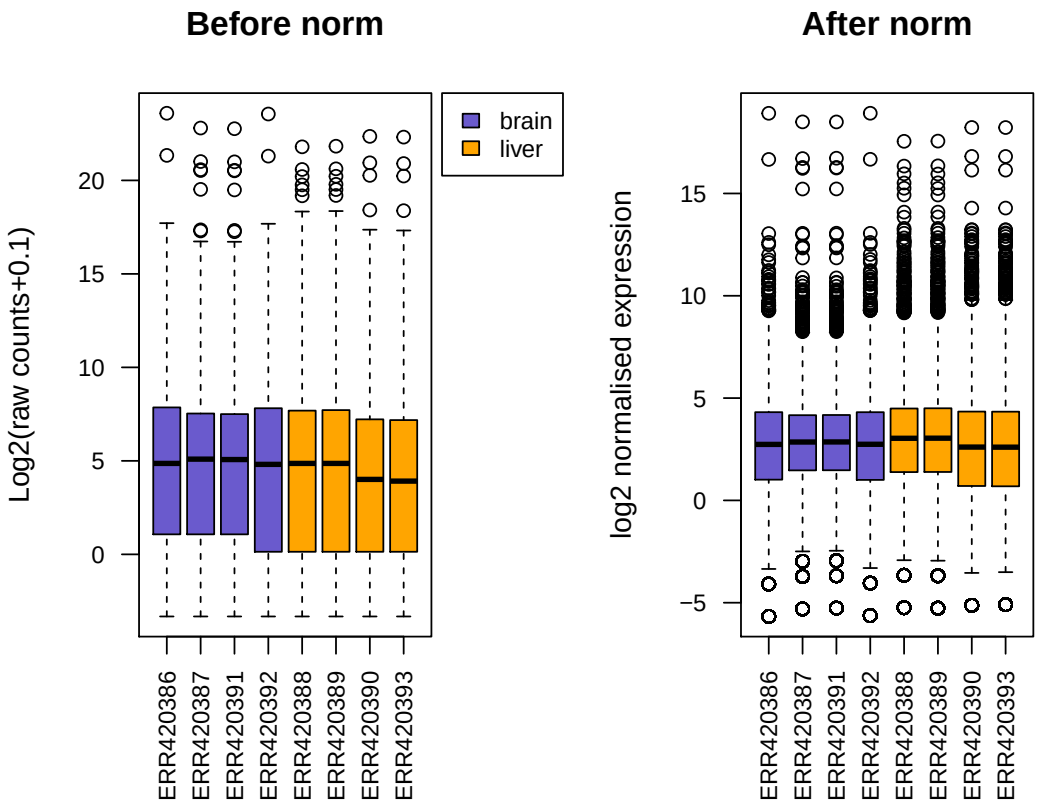
```
library(limma)

dge <- DGEList(filtered.raw.counts)
dge <- calcNormFactors(dge)
y <- voom(dge, design)
norm.expr <- y$E

write.table(norm.expr, file=file.path(RESULTS_DIR, "normalised_counts.txt"),
            row.names=T, quote=F, sep="\t")
```

```
boxplot(norm.expr,
        col=group.colours,
        main="Distribution of normalised counts",
        xlab="",
        ylab="log2 normalised expression",
        las=2,cex.axis=0.8)
```



Challenge 1. Add in the legend to the plot above (hint: see code for previous boxplot)

2. Can you put the boxplots side by side to show before and after normalisation? (hint: `mfrow=c(X,X)`)

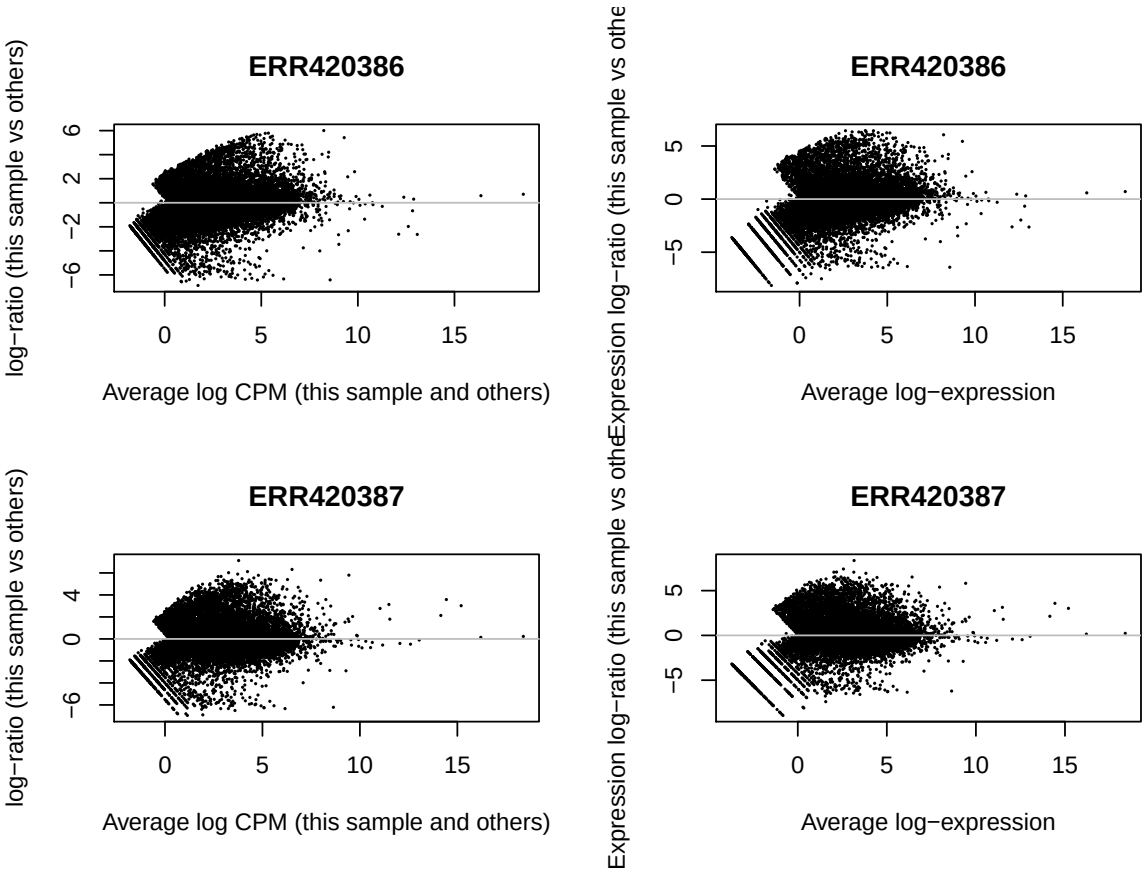


Solution

```
par(mfrow=c(1,2),mar=c(5.1, 5, 4.1, 4),xpd=TRUE)
boxplot(logcounts,
        col=group.colours,
        main="Before norm",
        xlab="",
        ylab="Log2(raw counts+0.1)",
        las=2,cex.axis=0.8)
legend("topright", inset=c(-0.45,0), cex = 0.8,
       legend = levels(group),
       fill = unique(group.colours))
boxplot(norm.expr,
        col=group.colours,
        main="After norm",
        xlab="",
        ylab="log2 normalised expression",
        las=2,cex.axis=0.8)
```

2.7.2 MA-plots

```
par(mfrow=c(2,2))
for (ix in 1:2) {
  plotMD(dge,ix)
  abline(h=0,col='grey')
  plotMD(y,ix)
  abline(h=0,col='grey')
}
```



2.8 Using visualisation to verify (sanity checks)

2.8.1 Principal Component Analysis (PCA)

A Principal Component Analysis (PCA) can also be performed with these data using the `mixOmics` package Le Cao et al. (2016). The proportion of explained variance histogram will show how much of the variability in the data is explained by each components.

Reads counts need to be transposed before being analysed with the `mixomics` functions, i.e. genes should be in columns and samples should be in rows. This is the code for transposing and checking the data before further steps:

```
library(mixOmics)

norm.expr.df <- t(norm.expr)
dim(norm.expr.df)
```

```
## [1]     8 15050
```

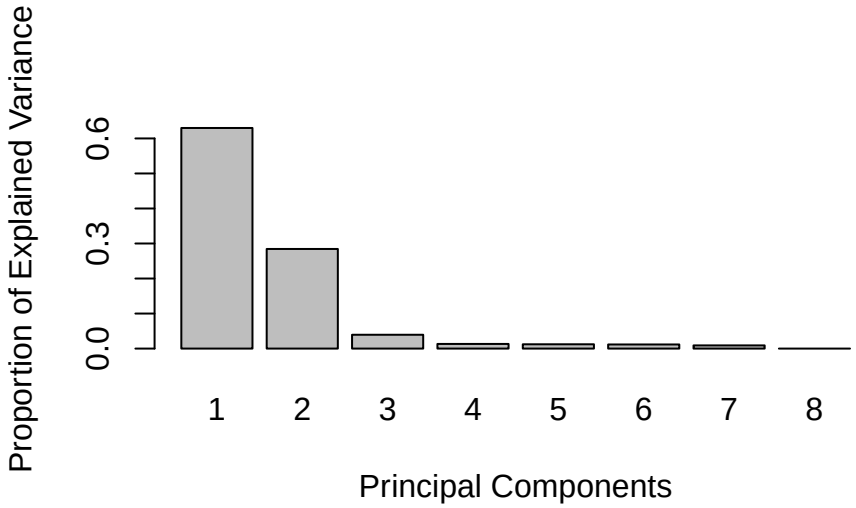
```
## check if any feature has 0 variance, if so might need to remove
colVar <- apply(norm.expr.df,2,var)
length(which(colVar==0))
```

```
## [1] 0
```

The proportion of explained variance helps you determine how many components can explain the variability in your dataset and thus how many dimensions you should be looking at.

```
tuning <- tune.pca(norm.expr.df, center=TRUE, scale=TRUE)
```

```
## Eigenvalues for the first 8 principal components, see object$sdev^2:
##          PC1          PC2          PC3          PC4          PC5
## 9.478900e+03 4.280479e+03 5.932513e+02 1.994188e+02 1.850595e+02
##          PC6          PC7          PC8
## 1.763042e+02 1.365877e+02 4.915698e-26
##
## Proportion of explained variance for the first 8 principal components, see
object$explained_variance:
##          PC1          PC2          PC3          PC4          PC5
## 6.298272e-01 2.844172e-01 3.941869e-02 1.325042e-02 1.229631e-02
##          PC6          PC7          PC8
## 1.171456e-02 9.075594e-03 3.266245e-30
##
## Cumulative proportion explained variance for the first 8 principal components, see
object$cum.var:
##       PC1       PC2       PC3       PC4       PC5       PC6       PC7
## 0.6298272 0.9142444 0.9536631 0.9669135 0.9792098 0.9909244 1.0000000
##       PC8
## 1.0000000
##
##  Other available components:
##  --------------------
##  loading vectors: see object$rotation
```



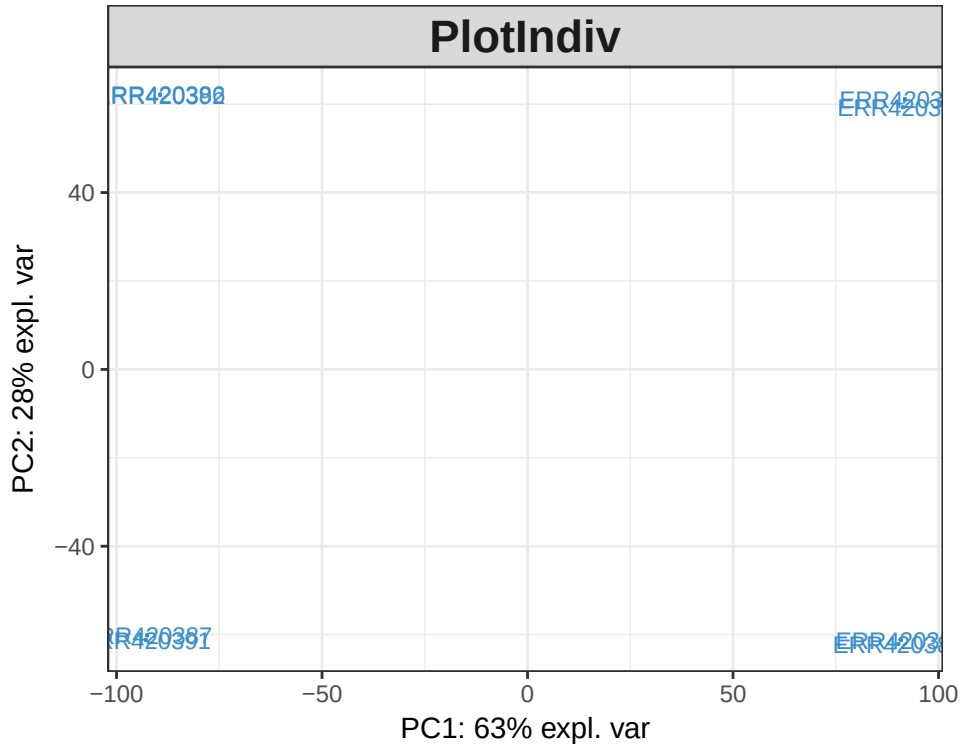
The variable `tune$prop.var` indicates the proportion of explained variance for the first 10 principal components:

Plotting this variable makes it easier to visualise and will allow future reference.

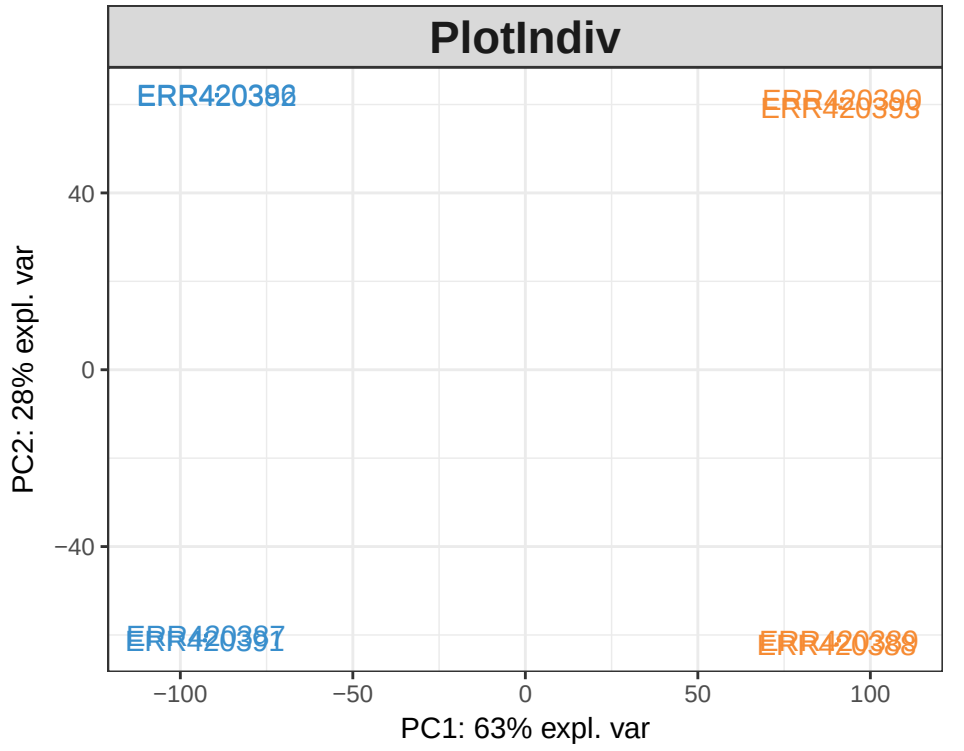
In most cases, the first 2 or 3 components explain more than half the variability in the dataset and can be used for plotting. The `pca` function will perform a principal components analysis on the given data

matrix. The `plotIndiv` function will provide scatter plots for sample representation.

```
pca.result <- pca(norm.expr.df, ncomp=3, center=T, scale=T)
plotIndiv(pca.result, comp=c(1,2))
```




```
plotIndiv(pca.result, comp=c(1,2), group=group, cex=4, xlim=c(-110,110))
```



Once we colour by the tissue types, there is obviously a difference between the two groups. This is clearly visible by the gap on the x-axis (principal component 1), which explains 63% of the variability observed in the data.

Challenge

There is also a vertical gap (principal component 2) separating the 2 samples in each tissue type. Can you explain what this separation is? (hint: look at the experiment design `expr.design`).

Solution

Colour the PCA plot by `group=expr.design$technical.replicate.group`.

Food for thought: also colour the plot using `group=expr.design$age`, how does this compare when you coloured using the `tissue` type? They look the same, so we cannot confirm if the horizontal separation is in fact due to tissue type or due to the age. This is why experimental design and replicates are extremely important when performing an experiment.

The PCA plot of the first two components show a clear separation of the Brain and Liver samples across the 1st dimension. Within each sample group we can also notice a split between the 4 samples of each group, which seem to cluster in pair. This observation can be explained by another factor of variability in the data, commonly batch effect or another biological bias such as age or sex.

For the 30 most highly expressed genes, we want to identify the reason for the split between samples from the same tissues. To do this, break the problem down:

- Get the read counts for the 30 most highly expressed genes
- Transpose this matrix of read counts
- Check the number of dimensions explaining the variability in the dataset
- Run the PCA with an appropriate number of components
- Annotate the samples with their age
  - re-run PCA
  - plot the main components
- Annotate the samples with other clinical data
  - re-run the PCA
  - plot the main components until you can separate the samples within each tissue group

2.9 Differential expression analysis

2.9.1 Fitting the model

To fit the model, use the `lmFit()` function, which takes in the normalised data object and the design matrix:

```
fit <- lmFit(y,design)
```

2.9.2 Comparisons

To have a look at the pairwise comparison(s) of interest, we may need to create a constrast matrix. This is usually always the case when there are multiple factors being compared. To do this, we use the `makeConstrasts()` function:

```
cont.matrix <- makeContrasts(liver-brain, levels=design)
cont.matrix
```

```
##        Contrasts
## Levels  liver - brain
##   brain            -1
##   liver             1
```

Refit the model using the comparisons defined:

```
fit2 <- contrasts.fit(fit, cont.matrix)
fit2 <- eBayes(fit2)
options(digits = 3)

dim(fit2)
```

```
## [1] 15050     1
```

2.9.3 Extract top DE features

The `topTable` function summarises the output from limma in a table format. Significant DE genes for a particular comparison can be identified by selecting genes with a p-value smaller than a chosen cut-off value and/or a fold change greater than a chosen value in this table. By default the table will be sorted by increasing adjusted p-value, showing the most significant DE genes at the top.

Set the threshold values:

```
THRES_PVAL <- 0.01
THRES_FC <- 3

colnames(fit$coefficients)
```

```
## [1] "brain" "liver"
```

Get the output table for the 10 most significant DE genes for this comparison: *livervsbrain*.

```
comparison='liver - brain'
topTable(fit2, coef=comparison)
```

```
##      logFC AveExpr     t  P.Value adj.P.Val    B
## 3240 10.82    7.85 100.7 5.60e-14  7.44e-10 19.2
## 5265 10.15    7.49  94.0 9.89e-14  7.44e-10 19.0
## 213  11.22   10.97  71.2 9.58e-13  2.06e-09 18.5
## 2335  6.80    8.58  67.7 1.45e-12  2.44e-09 18.5
## 338  11.09    8.52  71.8 9.00e-13  2.06e-09 18.3
## 2243 11.10    7.40  79.8 3.76e-13  1.46e-09 18.1
## 2244 11.18    7.18  75.4 6.02e-13  1.81e-09 17.7
## 229  10.58    6.61  79.5 3.88e-13  1.46e-09 17.7
## 125  10.92    7.40  67.7 1.46e-12  2.44e-09 17.6
```

```
## 716   7.47    6.77  59.4 4.24e-12  4.25e-09 17.5
```

Get the full table (n is the number of genes in the fit):

```
limma.result <- topTable(fit2, coef=comparison, n=nrow(fit2))

## Get significant DEGs only (adjusted p-value < THRES_PVAL)
limma.sigP.DEG <- topTable(fit2, coef=comparison, n=nrow(fit2), p.val=THRES_PVAL)
dim(limma.sigP.DEG)
```

```
## [1] 8190    6
```

```
head(limma.sigP.DEG,10)
```

```
##      logFC AveExpr     t  P.Value adj.P.Val    B
## 3240 10.82    7.85 100.7 5.60e-14  7.44e-10 19.2
## 5265 10.15    7.49  94.0 9.89e-14  7.44e-10 19.0
## 213  11.22   10.97  71.2 9.58e-13  2.06e-09 18.5
## 2335  6.80    8.58  67.7 1.45e-12  2.44e-09 18.5
## 338  11.09    8.52  71.8 9.00e-13  2.06e-09 18.3
## 2243 11.10    7.40  79.8 3.76e-13  1.46e-09 18.1
## 2244 11.18    7.18  75.4 6.02e-13  1.81e-09 17.7
## 229  10.58    6.61  79.5 3.88e-13  1.46e-09 17.7
## 125  10.92    7.40  67.7 1.46e-12  2.44e-09 17.6
## 716   7.47    6.77  59.4 4.24e-12  4.25e-09 17.5
```

Get significant DEG with low adjusted p-values and high fold change

```
limma.sigFC.DEG <- subset(limma.sigP.DEG, logFC > THRES_FC)
dim(limma.sigFC.DEG)
```

```
## [1] 1291    6
```

```
head(limma.sigFC.DEG,10)
```

```
##      logFC AveExpr     t  P.Value adj.P.Val    B
## 3240 10.82    7.85 100.7 5.60e-14  7.44e-10 19.2
## 5265 10.15    7.49  94.0 9.89e-14  7.44e-10 19.0
## 213  11.22   10.97  71.2 9.58e-13  2.06e-09 18.5
## 2335  6.80    8.58  67.7 1.45e-12  2.44e-09 18.5
## 338  11.09    8.52  71.8 9.00e-13  2.06e-09 18.3
## 2243 11.10    7.40  79.8 3.76e-13  1.46e-09 18.1
## 2244 11.18    7.18  75.4 6.02e-13  1.81e-09 17.7
## 229  10.58    6.61  79.5 3.88e-13  1.46e-09 17.7
## 125  10.92    7.40  67.7 1.46e-12  2.44e-09 17.6
## 716   7.47    6.77  59.4 4.24e-12  4.25e-09 17.5
```

Write the limma output table for significant genes to a tab-delimited file:

```
filename <- paste0("DEG_",comparison,
                   "_pval_",THRES_PVAL,
                   "_logFC_", THRES_FC, ".txt");
write.table(limma.sigFC.DEG, file=file.path(RESULTS_DIR,filename),
            row.names=T, quote=F, sep="\t")
filename
```

```
## [1] "DEG_liver - brain_pval_0.01_logFC_3.txt"
```

Get the number of DE genes between technical group 1 and technical group 2 (all Brain samples) with adj pvalue<0.01

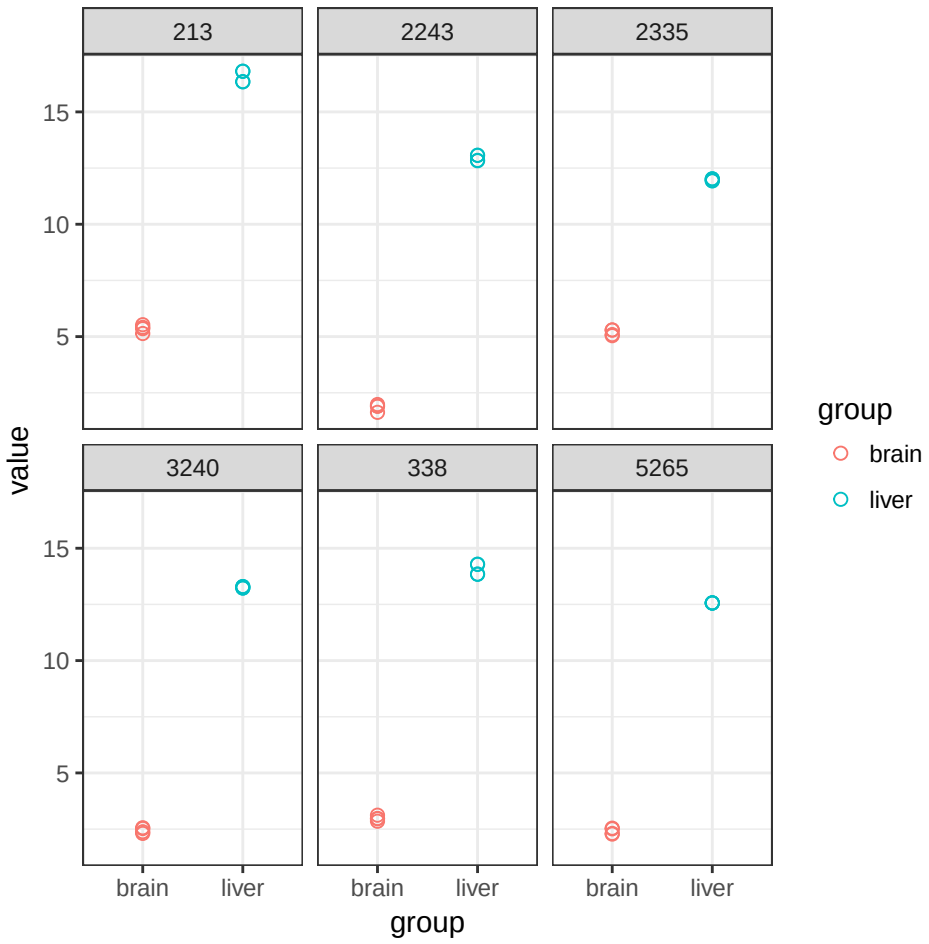
- Create a new design matrix for limma with the technical replicate groups
- Re-normalise the read counts with 'voom' function with new design matrix
- Fit a linear model on these normalised data
- Make the contrast matrix corresponding to the new set of parameters
- Fit the contrast matrix to the linear model
- Compute moderated t-statistics of differential expression
- Get the output table for the 10 most significant DE genes for this comparison
- Get the number of genes significantly DE (adjusted p-value < 0.01)

2.10 Verification using visualisation

Plot the top 6 DEGs to verify that they are indeed different between the groups *brainvsliver*.

The `tidyr` library helps us reshape the data from the wide form into a long form, which is much more flexible to work with when using `ggplot` for plotting graphs.

```
library(tidyr)
topDEG <- rownames(limma.sigFC.DEG)[1:6]
topDEG.norm <- as.data.frame(norm.expr[which(rownames(norm.expr) %in% topDEG),])
topDEG.norm$geneID <- rownames(topDEG.norm)
topDEG.norm.long <- gather(topDEG.norm, key=sample, value=value, -geneID)
topDEG.norm.long$group <- expr.design[topDEG.norm.long$sample,'tissue']

ggplot(topDEG.norm.long) + geom_point(aes(group,value,col=group),size=2,pch=1) +
  theme_bw() + facet_wrap(~geneID)
```



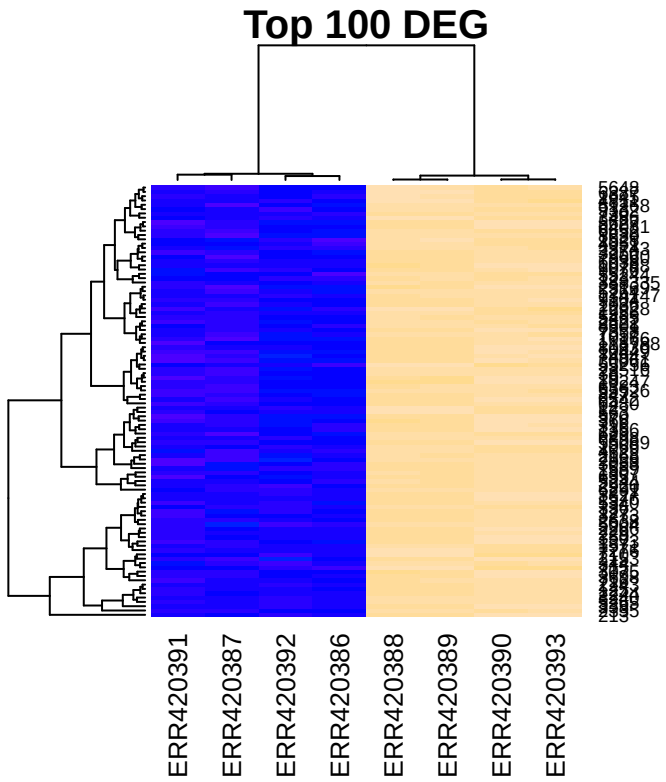
2.10.1 Hierachical clustering

In order to investigate the relationship between samples, hierarchical clustering can be performed using the `heatmap` function. In this example, `heatmap` calculates a matrix of euclidean distances from the normalised expression for the 100 most signficant DE genes.

```
topDEG <- rownames(limma.sigFC.DEG)[1:100]
highNormGenes <- norm.expr[topDEG,]
dim(highNormGenes)
```

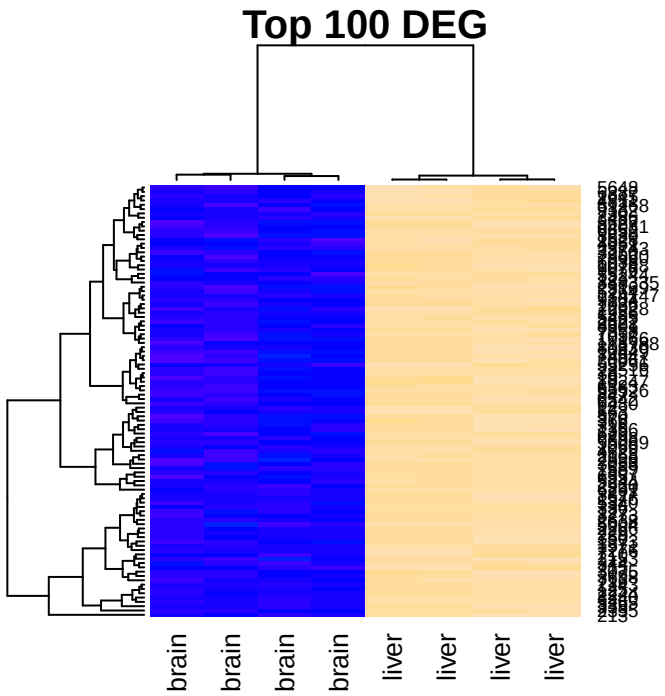
```
## [1] 100   8
```

```
par(cex.main=1)
heatmap(highNormGenes, col=topo.colors(50), cexCol=1,
        main='Top 100 DEG')
```



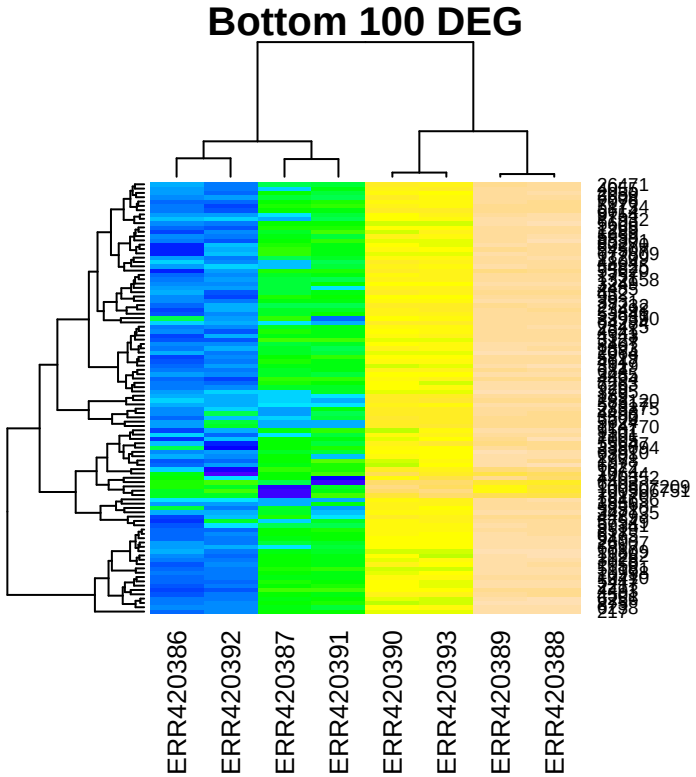
You will notice that the samples clustering does not follow the original order in the data matrix (alphabetical order "ERR420386" to "ERR420393"). They have been re-ordered according to the similarity of the 100 genes expression profiles. To understand what biological effect lies under this clustering, one can use the samples annotation for labeling (samples group, age, sex etc).

```
par(cex.main=1)
heatmap(highNormGenes, col=topo.colors(50),cexCol=1,
        main='Top 100 DEG', labCol = group)
```



Produce a heatmap for the bottom 100 significant genes.

- How many "groups" do you see?
- Can you explain them with the experimental design?



Challenge

- Just to be extra sure and for our own confidence, randomly select another 100 genes that are not in the DEG list and repeat the hierachical clustering using the `heatmap` plot.
- Out of these 100, pick 6 to plot the gene wise differences as seen previously.

Solution

```
nonDEG <- setdiff(rownames(filtered.raw.counts),rownames(limma.sigFC.DEG))
length(nonDEG)
randGene <- sample(nonDEG, 100, replace=F)
heatmap(norm.expr[randGene,],col=topo.colors(50),main="Non DEG",cexCol=1)
```

2.11 Gene Annotation

The annotation of EntrezGene IDs from RNAseq data can be done using the BioMart database which contains many species including Human, Mouse, Zebrafish, Chicken and Rat.

Get the Ensembl annotation for human genome. Since we used the build-in `hg19` annotation file that comes with Rsubread, we will specify the database version (`GRCh=37`) as a argument in the command.

```
library(biomaRt)
mart<- useEnsembl(biomart="ensembl",dataset="hsapiens_gene_ensembl",GRCh=37)
```

Get the Entrez gene IDs from the list of significant DEGs as identified using `limma-voom`:

```
DEG.entrezID <- rownames(limma.sigFC.DEG)
head(DEG.entrezID)
```

```
## [1] "3240" "5265" "213"  "2335" "338"  "2243"
```

Query the BioMart database to get the gene symbols and description for these genes:

```
DEG.annot <- getBM(filters= "entrezgene",
                   attributes= c("entrezgene","external_gene_name","description"),
                   values= DEG.entrezID,
                   mart= mart)
```

```
dim(DEG.annot)
```

```
## [1] 1262    3
```

```
head(DEG.annot)
```

```
##   entrezgene external_gene_name
## 1         10               NAT2
## 2  100128553             CTAGE4
## 3  100128553             CTAGE9
## 4  100128893          GATA6-AS1
## 5  100129046      RP5-1033H22.2
## 6  100130231          LINC00861
```

```
##
description
## 1 N-acetyltransferase 2 (arylamine N-acetyltransferase) [Source:HGNC
Symbol;Acc:7646]
## 2                               CTAGE family, member 4 [Source:HGNC
Symbol;Acc:24772]
## 3                               CTAGE family, member 9 [Source:HGNC
Symbol;Acc:37275]
## 4                 GATA6 antisense RNA 1 (head to head) [Source:HGNC
Symbol;Acc:48840]
## 5
## 6           long intergenic non-protein coding RNA 861 [Source:HGNC
Symbol;Acc:45133]
```

In many cases, several annotations are available per entrez gene ID. This results in duplicate entries in the output table from `getBM()`. The simplest way to deal with this issue is to remove duplicates, although they can also be concatenated in some ways.

Once the annotation has been obtained for all DE genes, this table can be merged with the output table from limma for a complete result and an easier interpretation.

```
DEG.annot.unique <- DEG.annot[-which(duplicated(DEG.annot$entrezgene)),]
dim(DEG.annot.unique)
```

```
## [1] 1221    3
```

```
head(DEG.annot.unique)
```

```
##   entrezgene external_gene_name
## 1         10               NAT2
## 2  100128553             CTAGE4
## 4  100128893          GATA6-AS1
## 5  100129046      RP5-1033H22.2
## 6  100130231          LINC00861
## 7  100131733          USP30-AS1
##
description
## 1 N-acetyltransferase 2 (arylamine N-acetyltransferase) [Source:HGNC
Symbol;Acc:7646]
## 2                               CTAGE family, member 4 [Source:HGNC
Symbol;Acc:24772]
## 4                 GATA6 antisense RNA 1 (head to head) [Source:HGNC
Symbol;Acc:48840]
## 5
## 6           long intergenic non-protein coding RNA 861 [Source:HGNC
Symbol;Acc:45133]
## 7                                USP30 antisense RNA 1 [Source:HGNC
Symbol;Acc:40909]
```

```
rownames(DEG.annot.unique) <- DEG.annot.unique$entrezgene
entrezGenes.annot <- DEG.annot.unique[DEG.entrezID,]
limma.sigFC.DEG <- cbind(entrezGenes.annot,limma.sigFC.DEG)
head(limma.sigFC.DEG)
```

```
##      entrezgene external_gene_name
## 3240       3240                 HP
## 5265       5265           SERPINA1
## 213         213                ALB
## 2335       2335                FN1
```

```
## 338          338               APOB
## 2243        2243                FGA
##
                       description
## 3240
haptoglobin [Source:HGNC Symbol;Acc:5141]
## 5265 serpin peptidase inhibitor, clade A (alpha-1 antiproteinase, antitrypsin),
member 1 [Source:HGNC Symbol;Acc:8941]
## 213
albumin [Source:HGNC Symbol;Acc:399]
## 2335
fibronectin 1 [Source:HGNC Symbol;Acc:3778]
## 338
apolipoprotein B [Source:HGNC Symbol;Acc:603]
## 2243                                                              fibrinogen alpha
chain [Source:HGNC Symbol;Acc:3661]
##      logFC AveExpr     t  P.Value adj.P.Val    B
## 3240  10.8    7.85 100.7 5.60e-14  7.44e-10 19.2
## 5265  10.2    7.49  94.0 9.89e-14  7.44e-10 19.0
## 213   11.2   10.97  71.2 9.58e-13  2.06e-09 18.5
## 2335   6.8    8.58  67.7 1.45e-12  2.44e-09 18.5
## 338   11.1    8.52  71.8 9.00e-13  2.06e-09 18.3
## 2243  11.1    7.40  79.8 3.76e-13  1.46e-09 18.1
```

You can find the list of attributes to extract from the `getBM()` function by using the following command: `View(listAttributes(mart))`.

2.12 Gene set enrichment

Gene Ontology (GO) enrichment is a method for investigating sets of genes using the Gene Ontology system classification, in which genes are assigned to a particular set of terms for three major domains: cellular component, biological process and molecular function.

The `GOstats` package can test for both over and under representation of GO terms using the standard hypergeometric test. The output of the analysis is typically a ranked list of GO terms, each associated with a p-value.

The hypergeometric test will require both a list of selected genes (e.g. your DE genes) and a "universe" list (e.g. all genes represented that were tested for differential expression), all represented by their "EntrezGene" ID.

Get the list of universe entrez IDs:

```
library(GOstats)
universe.entrezID <- rownames(filtered.raw.counts)
length(universe.entrezID)
```

```
## [1] 15050
```

Before running the hypergeometric test with the `hyperGTest` function, you need to define the parameters for the test (gene lists, ontology, test direction) as well as the annotation database to be used. The ontology to be tested for can be any of the three GO domains: biological process ("BP"), cellular component ("CC") or molecular function ("MF").

In the example below, we will test for over-represented Biological Processes in our list of differentially expressed genes.

```
annotationDB <- "org.Hs.eg.db"
hgCutoff <- 0.05

params <- new("GOHyperGParams",
              geneIds=DEG.entrezID,
              universeGeneIds=universe.entrezID,
              annotation=annotationDB,
              ontology="BP",
              pvalueCutoff=hgCutoff,
              testDirection="over")
# Run the test
hg <- hyperGTest(params)
hg
```

```
## [1] 2010    7
##        GOBPID   Pvalue OddsRatio ExpCount Count Size
## 1  GO:0006082 6.07e-66      4.96     73.5   236  820
## 2  GO:0043436 1.15e-58      4.77     68.3   216  762
## 3  GO:0019752 9.45e-58      4.73     67.9   214  758
## 4  GO:0044281 1.90e-54      3.34    141.8   326 1582
## 5  GO:0032787 9.53e-54      5.89     42.2   159  471
## 6  GO:0006952 2.58e-53      3.97     88.3   242  985
## 7  GO:0044710 5.22e-51      2.69    289.3   509 3229
## 8  GO:0006954 1.37e-48      5.64     40.1   148  448
## 9  GO:0006955 2.44e-43      3.48     91.0   229 1016
## 10 GO:0008202 1.76e-41      8.04     20.2    95  225
##                                    Term
## 1         organic acid metabolic process
## 2              oxoacid metabolic process
## 3      carboxylic acid metabolic process
## 4       small molecule metabolic process
## 5  monocarboxylic acid metabolic process
## 6                       defense response
## 7      single-organism metabolic process
## 8                 inflammatory response
## 9                      immune response
## 10             steroid metabolic process
```

We need to adjust for multiple testing using the `p.adjust()` function. You can specify the type of adjustment method to use, we are using `bonferroni` in this example. Assign the adjusted pvalues back to the `hg.df` data object. Reorder the columns so that the unadjusted and adjusted p-values are next to each other.

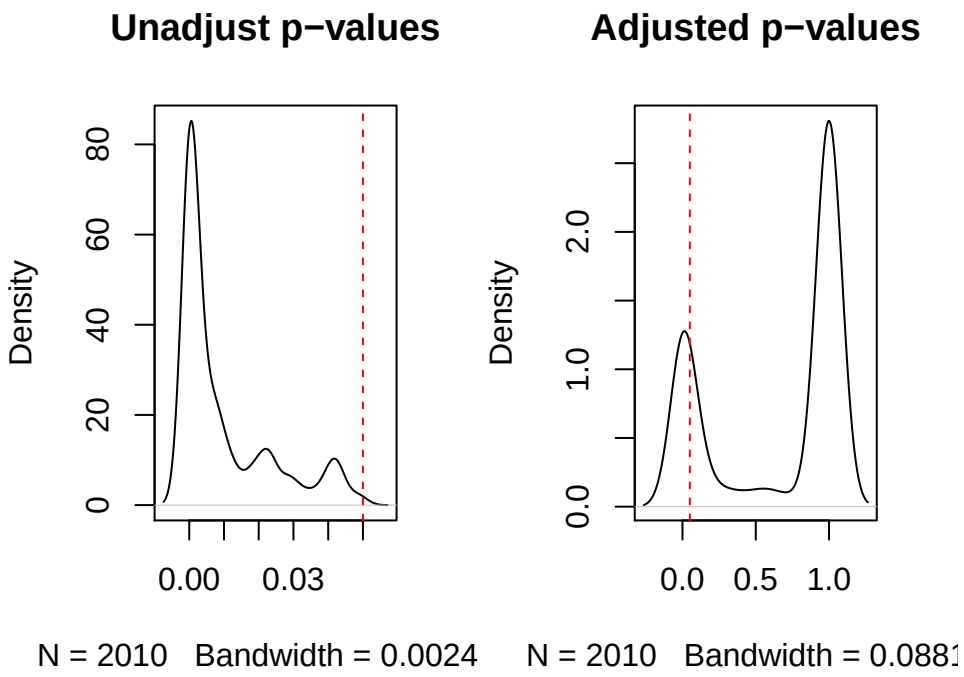
```
hg.df <- summary(hg)
hg.df$Adj.Pvalue <- p.adjust(hg.df$Pvalue, 'bonferroni')
hg.df <- hg.df[,c(1:2,8,3:7)]
head(hg.df,10)
```

```
##        GOBPID       Pvalue   Adj.Pvalue OddsRatio  ExpCount Count Size
## 1  GO:0006082 6.074927e-66 1.221060e-62  4.964910  73.47815   236  820
## 2  GO:0043436 1.146564e-58 2.304594e-55  4.766285  68.28092   216  762
## 3  GO:0019752 9.451120e-58 1.899675e-54  4.729580  67.92249   214  758
## 4  GO:0044281 1.899138e-54 3.817267e-51  3.336444 141.75907   326 1582
## 5  GO:0032787 9.530910e-54 1.915713e-50  5.891811  42.20513   159  471
## 6  GO:0006952 2.581186e-53 5.188184e-50  3.968852  88.26339   242  985
## 7  GO:0044710 5.216264e-51 1.048469e-47  2.692125 289.34263   509 3229
## 8  GO:0006954 1.371656e-48 2.757028e-45  5.643188  40.14416   148  448
## 9  GO:0006955 2.441550e-43 4.907515e-40  3.477192  91.04122   229 1016
## 10 GO:0008202 1.761494e-41 3.540603e-38  8.038462  20.16169    95  225
##                                    Term
```

```
## 1         organic acid metabolic process
## 2              oxoacid metabolic process
## 3      carboxylic acid metabolic process
## 4       small molecule metabolic process
## 5  monocarboxylic acid metabolic process
## 6                       defense response
## 7      single-organism metabolic process
## 8                  inflammatory response
## 9                        immune response
## 10             steroid metabolic process
```

Compare before and after multiple adjustment:

```
par(mfrow=c(1,2))
plot(density(hg.df$Pvalue),'Unadjust p-values')
abline(v=hgCutoff,col='red',lty=2)

plot(density(hg.df$Adj.Pvalue),'Adjusted p-values')
abline(v=hgCutoff,col='red',lty=2)
```



Keep only the significant GO terms after adjusting for multiple testing:

```
sigGO.table <- subset(hg.df, Adj.Pvalue < hgCutoff)
dim(sigGO.table)
```

```
## [1] 517   8
```

```
head(sigGO.table,10)
```

```
##        GOBPID   Pvalue Adj.Pvalue OddsRatio ExpCount Count Size
## 1  GO:0006082 6.07e-66   1.22e-62      4.96     73.5   236  820
## 2  GO:0043436 1.15e-58   2.30e-55      4.77     68.3   216  762
## 3  GO:0019752 9.45e-58   1.90e-54      4.73     67.9   214  758
```

```
## 4  GO:0044281 1.90e-54   3.82e-51      3.34    141.8   326 1582
## 5  GO:0032787 9.53e-54   1.92e-50      5.89     42.2   159  471
## 6  GO:0006952 2.58e-53   5.19e-50      3.97     88.3   242  985
## 7  GO:0044710 5.22e-51   1.05e-47      2.69    289.3   509 3229
## 8  GO:0006954 1.37e-48   2.76e-45      5.64     40.1   148  448
## 9  GO:0006955 2.44e-43   4.91e-40      3.48     91.0   229 1016
## 10 GO:0008202 1.76e-41   3.54e-38      8.04     20.2    95  225
##                                      Term
## 1         organic acid metabolic process
## 2              oxoacid metabolic process
## 3      carboxylic acid metabolic process
## 4       small molecule metabolic process
## 5  monocarboxylic acid metabolic process
## 6                      defense response
## 7      single-organism metabolic process
## 8                 inflammatory response
## 9                       immune response
## 10             steroid metabolic process
```

Other software can be used to investigate over-represented pathways, such as GeneGO https://portal.genego.com/ and Ingenuity http://www.ingenuity.com/products/ ipa. The advantage of these applications is that they maintain curated and up-to-date extensive databases. They also provide intuitive visualisation and network modelling tools.

Save an image of your RNAseq analysis.

```
save.image(file=file.path(RESULTS_DIR,"RNAseq.Rdata"))
```

SessionInfo

```
sessionInfo()
```

```
## R version 3.3.3 (2017-03-06)
## Platform: x86_64-redhat-linux-gnu (64-bit)
## Running under: CentOS Linux 7 (Core)
##
## locale:
##  [1] LC_CTYPE=en_AU.UTF-8       LC_NUMERIC=en_AU.UTF-8
##  [3] LC_TIME=en_AU.UTF-8        LC_COLLATE=en_AU.UTF-8
##  [5] LC_MONETARY=en_AU.UTF-8    LC_MESSAGES=en_AU.UTF-8
##  [7] LC_PAPER=en_AU.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_AU.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] GO.db_3.4.0                GOstats_2.40.0
##  [3] graph_1.52.0               Category_2.40.0
##  [5] Matrix_1.2-8               AnnotationDbi_1.36.2
##  [7] biomaRt_2.30.0             tidyr_0.6.1
##  [9] edgeR_3.16.5               limma_3.30.13
## [11] mixOmics_6.1.2             ggplot2_2.2.1
## [13] lattice_0.20-35            MASS_7.3-45
## [15] Rsubread_1.24.2            ShortRead_1.32.1
## [17] GenomicAlignments_1.10.1   SummarizedExperiment_1.4.0
## [19] Biobase_2.34.0             Rsamtools_1.26.2
## [21] GenomicRanges_1.26.4       GenomeInfoDb_1.10.3
## [23] BiocParallel_1.8.2         Biostrings_2.42.1
## [25] XVector_0.14.1             IRanges_2.8.2
## [27] S4Vectors_0.12.2           BiocGenerics_0.20.0
##
## loaded via a namespace (and not attached):
##  [1] splines_3.3.3          jsonlite_1.4           ellipse_0.3-8
##  [4] shiny_1.0.1            assertthat_0.2.0       latticeExtra_0.6-28
##  [7] RBGL_1.50.0            yaml_2.1.14            RSQLite_1.1-2
## [10] backports_1.0.5        digest_0.6.12          RColorBrewer_1.1-2
## [13] colorspace_1.3-2       htmltools_0.3.5        httpuv_1.3.3
## [16] plyr_1.8.4             GSEABase_1.36.0        XML_3.98-1.6
## [19] genefilter_1.56.0      bookdown_0.3           zlibbioc_1.20.0
## [22] xtable_1.8-2           corpcor_1.6.9          scales_0.4.1
## [25] tibble_1.3.0           annotate_1.52.1        lazyeval_0.2.0
## [28] survival_2.41-3        magrittr_1.5           mime_0.5
## [31] memoise_1.0.0          evaluate_0.10          hwriter_1.3.2
## [34] tools_3.3.3            stringr_1.2.0          munsell_0.4.3
## [37] locfit_1.5-9.1         grid_3.3.3             RCurl_1.95-4.8
## [40] rstudioapi_0.6         AnnotationForge_1.16.1 htmlwidgets_0.8
## [43] igraph_1.0.1           labeling_0.3           bitops_1.0-6
## [46] rmarkdown_1.4          codetools_0.2-15       gtable_0.2.0
## [49] DBI_0.6-1              reshape2_1.4.2         R6_2.2.0
## [52] knitr_1.15.1           dplyr_0.5.0            rprojroot_1.2
## [55] stringi_1.1.5          Rcpp_0.12.10
```

Bibliography

Le Cao, K.-A., Rohart, F., Gonzalez, I., with key contributors Benoit Gautier, S. D., Bartolo, F., contributions from Pierre Monget, Coquery, J., Yao, F., and Liquet., B. (2016). *mixOmics: Omics Data Integration Project*. R package version 6.1.1.

Ritchie, M. E., Phipson, B., Wu, D., Hu, Y., Law, C. W., Shi, W., and Smyth, G. K. (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research*, 43(7):e47.

Robinson, M. D., Mccarthy, D. J., and Smyth, G. K. (2009). edger: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140.

Y, L., GK, S., and W, S. (2013). The subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Research*, 41:e108.