Data processing and quality control

What we produced:
- FASTQ files
- FASTQC reports
- SAM and BAM files

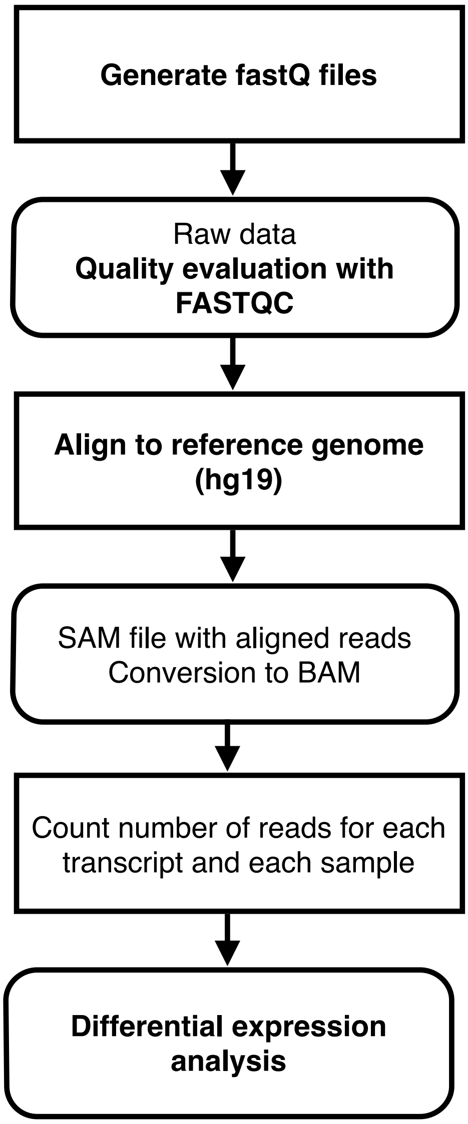


Figure: Schematic overview of the pipeline for RNA-seq data analysis

Differential expression analysis

What we produced: - R script: DE_analysis.R - Table with read counts (tab separated format, 7 columns, ENSG ids)

RNA-seq data can be difficult to interpret (especially in terms of differential expression quantification). Thus, we decided to adopt a simple method for the analysis, based on counting, for each gene and for each sample, the number of available reads and then testing for significant differences between two experimental conditions or groups.

We wrote an R script that automatically creates a PDF file (in the current directory) with all the figures necessary for visual inspection and result interpretation. The input is a tab separated file with reads counts.

```
ensembl_id      melanocyte_1    melanocyte_2    melanome_1      melanome_2
ENSG00000000003 1964    2409    2328    2451
ENSG00000000005 0       2       10      12
ENSG00000000419 15122   19592   38225   36654
ENSG00000000457 12129   14893   7483    7812
ENSG00000000460 21930   25575   13123   13840
ENSG00000000938 48      58      26      42
ENSG00000000971 125     229     124     236
ENSG00000001036 11611   14125   14067   13518
ENSG00000001084 11429   13795   3549    3279
```

Figure: Example input format for DE analysis

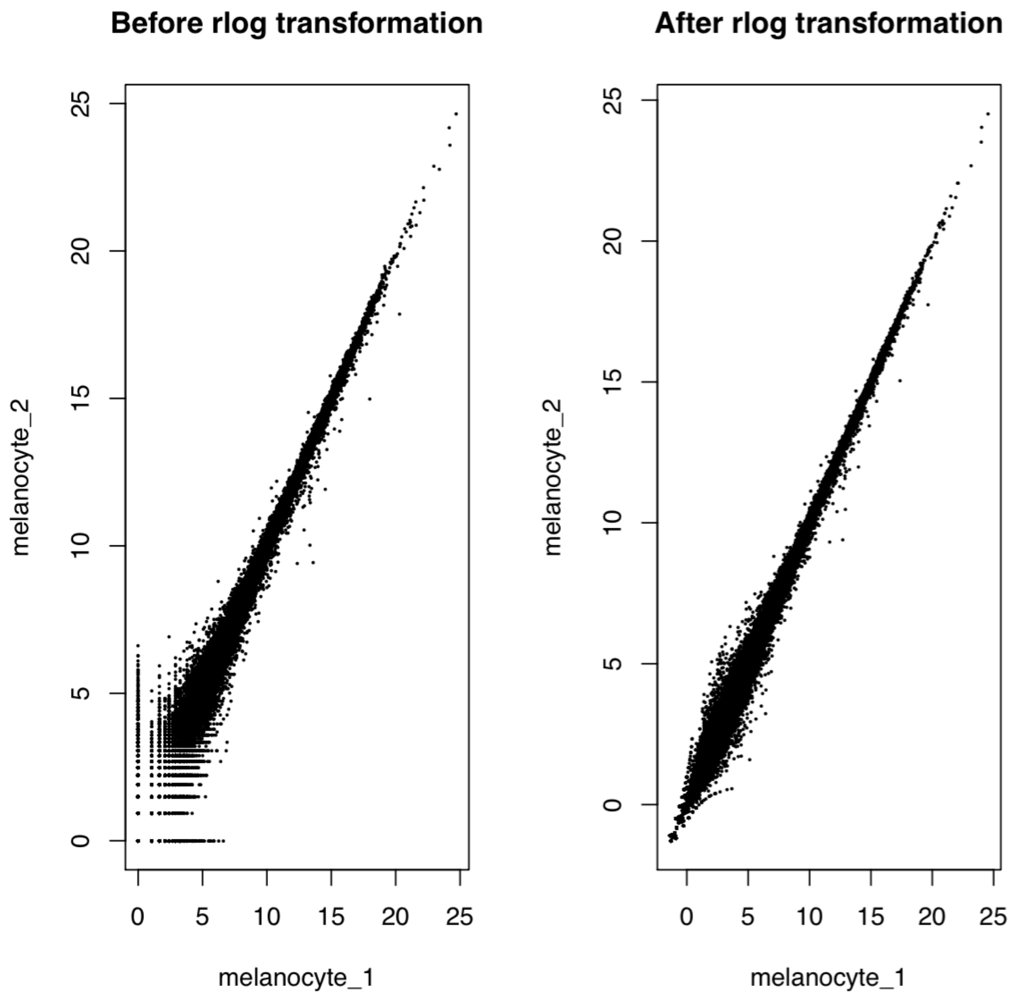
We tested two designs, as illustrated in the tables below:
- normal cells vs cancerous cells (4 samples)
- cancerous cells vs cancerous drug treated (4 samples)

Sample name	Condition
melanocyte_1	M
melanocyte_2	M
melanome_1	C
melanome_2	C

Sample name	Condition
melanoma_1	C
melanoma_2	C
melanome_drug_1	D
melanome_drug_2	D

A) Visual exploration of the samples

Prior to checking distances between our samples, we applied a regularized-logarithm transformation (rlog) to stabilise the variance across the mean. The effects of the transformation are shown in the figure below.



We noticed that this step was particularly important for genes with low read counts.

We then checked the distances between our samples by performing Principal Components Analysis of the count data.

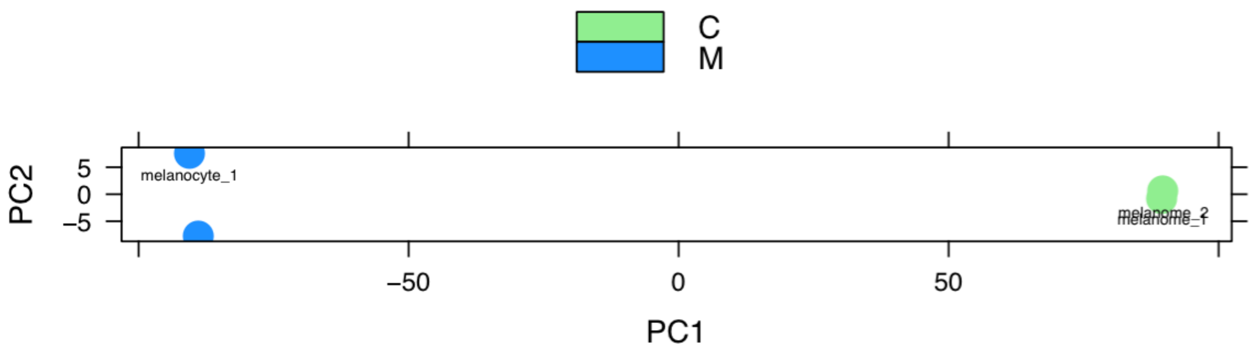


Figure: Principal Components Analysis (PCA) plot, normal vs cancerous cells

We observed that differences between groups (normal vs cancerous cells represented in the PCA plot above) were greater than intra-groups differences, which is expected in this kind of design. However, as the inter-group differences were so pronounced, we figured that a great amount of genes would appear as differentially expressed: this is why we decided to apply really stringent thresholds for the detection:

- log2 fold change (logFC) > 5 for upregulated genes or log2 fold change (logFC) < -5 for downregulated genes
- AND adjusted-p-value < 0.01

B) Differential expression analysis

Firstly, we took a look at the raw data (prior to any kind of normalization). We calculated mean counts for each gene and by condition and then the log2 fold change.

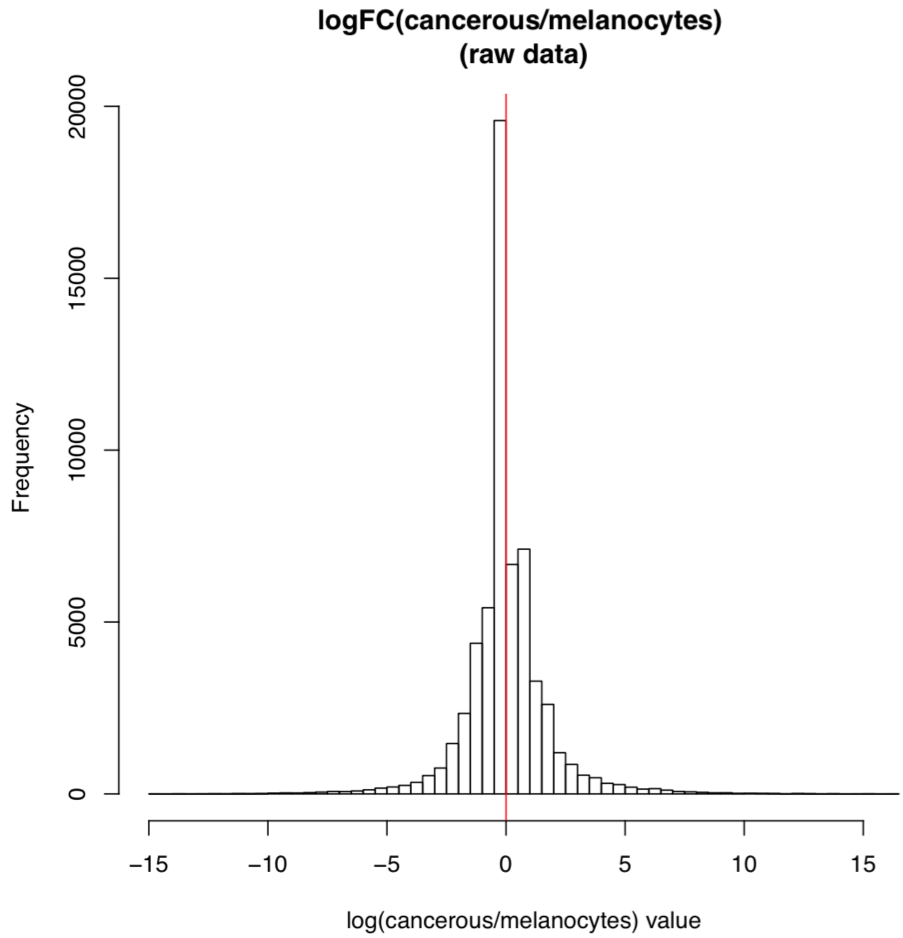


Figure: Distribution of logFC(cancerous/normal) values - raw data

Prior to normalization, we filtered the data set to remove rows with very little or no information (remove genes with no counts or with just a single count). This allows to eliminate 17 386 transcripts already.

Using the DESeq R package (Bioconductor, https://bioconductor.org/packages/release/bioc/html/DESeq.html), we were able to perform normalization of our data after calculation of size factors and we then were able to calculate mean counts for each gene and by condition and finally the logFC.

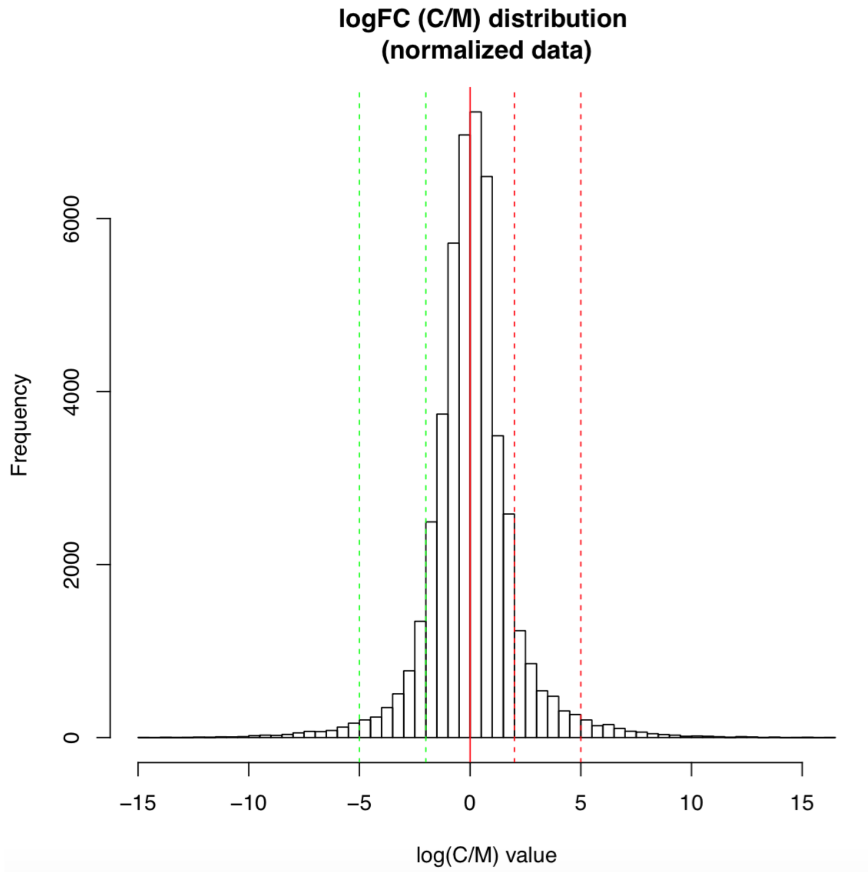


Figure: Distribution of logFC(cancerous/normal) values - normalized data

Finally, we applied the nbinomWaldTest() function from the DESeq package to test for significance of coefficients in a negative binomial GLM, the model we used to assess differences in expression.

As previously stated, selection of significantly up- or downregulated genes was based on the establishment of two selection thresholds: logFC and adjusted p-value (Wald test M vs C).

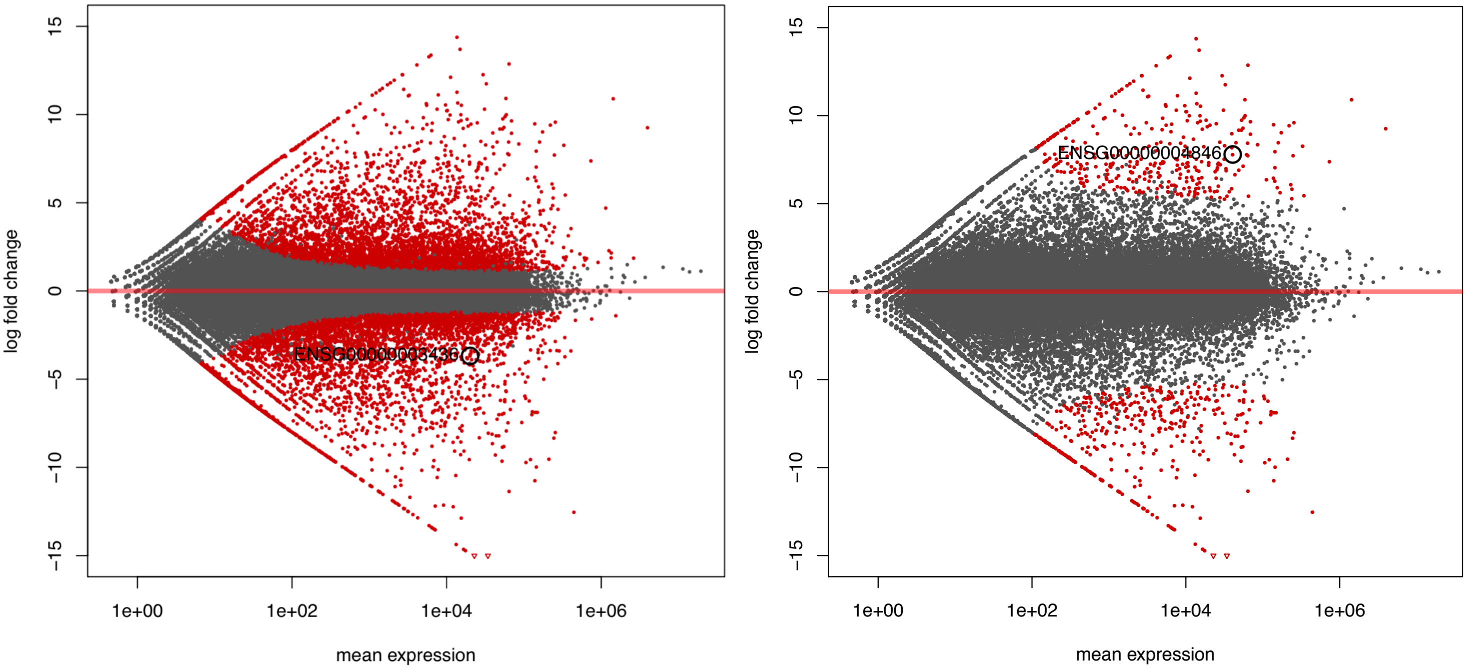


Figure: Differential expression as a function of mean expression. Left panel: threshold set at logFC > 2 or < -2. Right panel: threshold set at logFC > 5 or < -5.
The red dots indicate genes for which the logFC was significantly higher than 5 or lower than -5. The circled point indicates the gene with the lowest adj-p-value.

We obtained a list of 1 649 differentially expressed genes: 931 upregulated genes and 718 downregulated genes.

C) Enrichment analysis

We retrieved the list of the 931 unregulated genes and the list of the 718 downregulated genes and looked for significantly enriched GO (Genome Ontology) terms in these lists (independently).

The results are summarized in the figures below:

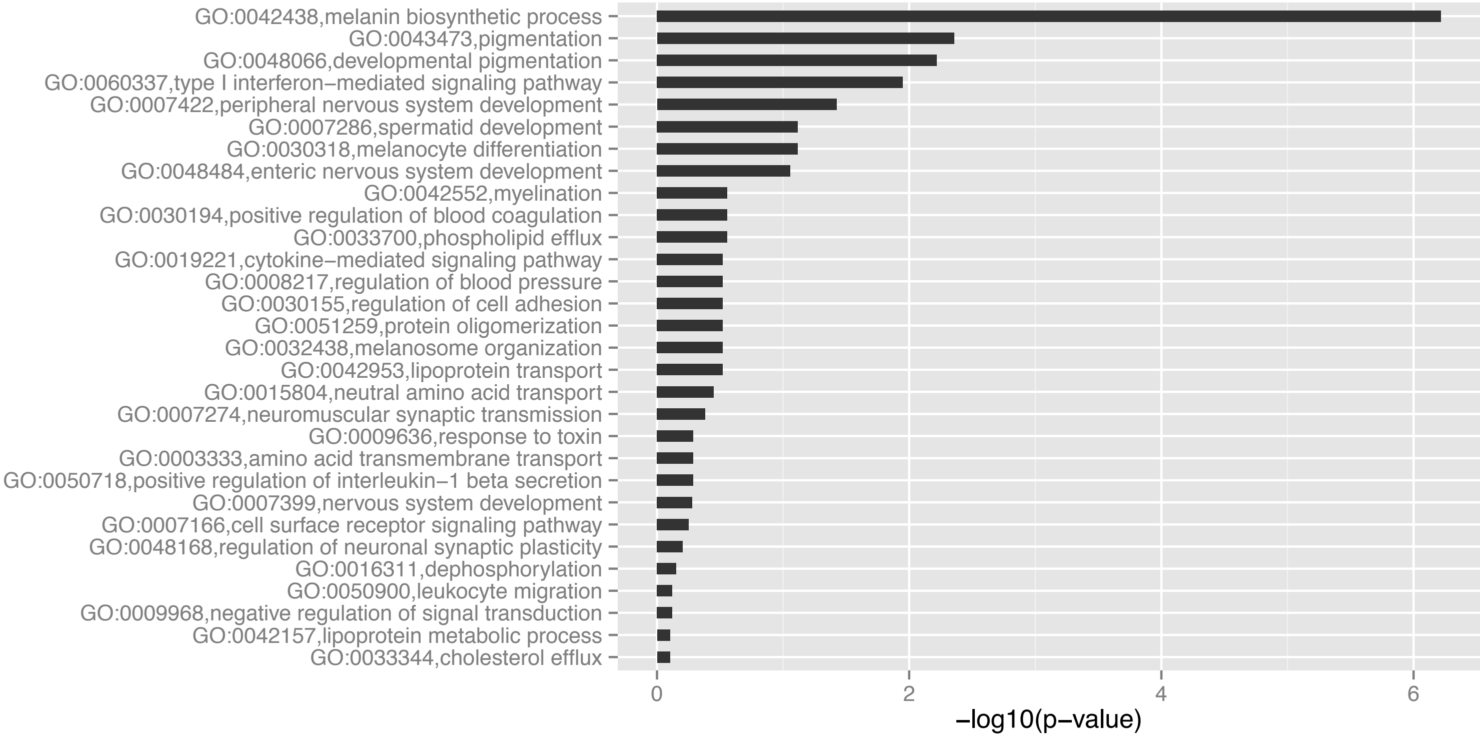


Figure: Enrichment in GO terms, downregulated genes

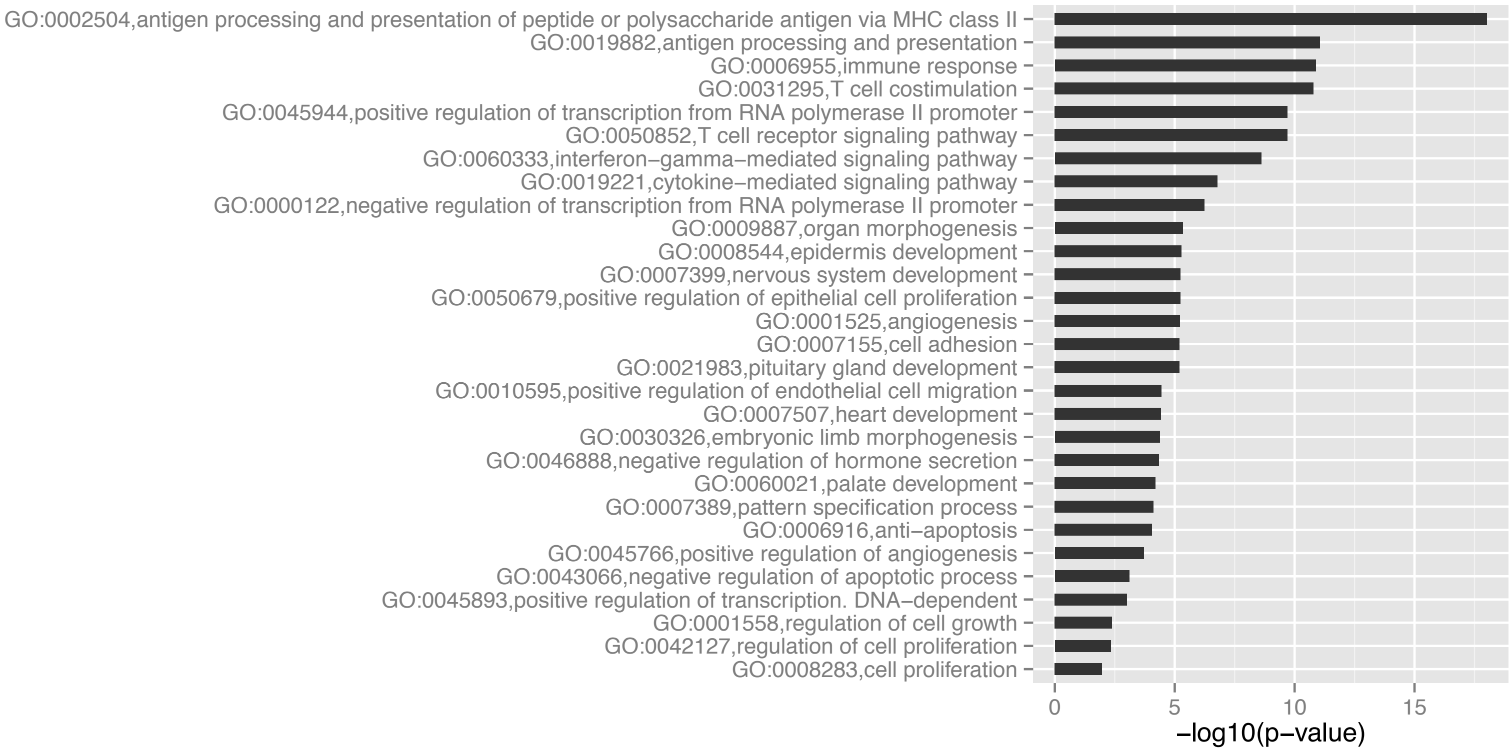


Figure: Enrichment in GO terms, unregulated genes

D) DE script

```
pdf("DE_analysis_graphics.pdf")

# Read data file
dataRNAseq = read.table("../TrimmedData/merged_counts_ENSG_identifiers.tsv",
header = TRUE, row.names = 1)

# Calculate logFC values using read counts

# mean values for melanocytes and cancerous cells
meanMcounts = apply(dataRNAseq[,1:2],1,mean)
meanCcounts = apply(dataRNAseq[,3:4],1,mean)

# logFC on raw data
logFC = log2((meanCcounts + 1)/(meanMcounts + 1))

# distribution of logFC on raw data
hist(logFC, nclass = 100, main = "logFC(cancerous/melanocytes) \n(raw data)",
xlab = "log(cancerous/melanocytes) value")
abline(v = 0, col = "red")


# DESeq package

library(DESeq2)

# Loading data for the experiment
# M = "normal" melanocyte
# C = cancerous cell
# design.txt = text file with 2 columns, first experiment and second condition
(M/C)

colData = read.table("design.txt", row.names = 1, header = TRUE)

# DESeqDataSet object creation
dds = DESeqDataSetFromMatrix(countData = dataRNAseq[,1:4], colData = colData,
design = ~condition)
#nrow(dds)
#60234

# Pre-filtering the data set (removing rows with no counts or a single count)
dds = dds[rowSums(counts(dds))>1,]
#nrow(dds)
#47451

# calculation of sizeFactors
dds = estimateSizeFactors(dds)
sizeFactors(dds)

# Visual exploring of the data

# rlog transformation (regularized log transforlation, stabilize variance across
the mean)
# for fully unsupervised transformation, set blind=TRUE
rld = rlog(dds,blind=TRUE)

# Effect of the rlog transformation, first two samples
par(mfrow=c(1,2))
dds=estimateSizeFactors(dds)
plot(log2(counts(dds,normalized=TRUE)[,1:2]+1),pch=16,cex=0.3,
main="Before rlog transformation")
plot(assay(rld)[,1:2],pch=16,cex=0.3,
main="After rlog transformation")
```

```
# PCA plot
par(mfrow=c(1,1))
p_rld = plotPCA(rld,intgroup=c("condition"))
p_rld = update(p_rld, panel = function(x, y, ...) {lattice::panel.xyplot(x,
y, ...);
lattice::ltext(x=x, y=y, labels=rownames(colData(rld)), pos=1, offset=1,
cex=0.5)})
print(p_rld)

# Sample distances
sampleDists = dist(t(assay(rld)))

# Heatmaps distances
library("RColorBrewer")
library("pheatmap")

sampleDistMatrix = as.matrix(sampleDists)
colnames(sampleDistMatrix) = NULL
colors = colorRampPalette(rev(brewer.pal(9,"Blues")))(255)
pheatmap(sampleDistMatrix, clustering_distance_rows = sampleDists,
clustering_distance_cols = sampleDists, col = colors,
main="Heatmat of sample distances")


# Normalization of the data

# get normalized count values
cdsNorm = counts(dds, normalized = TRUE)

# mean values
meanMcountsNorm = apply(cdsNorm[,1:2], 1, mean)
meanCcountsNorm = apply(cdsNorm[,3:4], 1, mean)
# sd values for log(H/N) replicates
sdMcountsNorm   = apply(cdsNorm[,1:2], 1, sd)
sdCcountsNorm   = apply(cdsNorm[,3:4], 1, sd)

# logFC (after normalization)
logFCNorm = log2((meanCcountsNorm + 1)/(meanMcountsNorm + 1))

hist(logFCNorm, nclass = 100, main = "logFC (C/M) distribution \n(normalized
data)",
xlab = "log(C/M) value")
abline(v = 0, col = "red")

# thresold can be chosen (here the values are 2 and 5) to select up and down
regulated genes
abline(v = 2, col = "red", lty = "dashed")
abline(v = -2, col = "green", lty = "dashed")
abline(v = 5, col = "red", lty = "dashed")
abline(v = -5, col = "green", lty = "dashed")

upGenes2 = names(logFCNorm[logFCNorm > 2])
downGenes2 = names(logFCNorm[logFCNorm < -2])
upGenes5 = names(logFCNorm[logFCNorm > 5])
downGenes5 = names(logFCNorm[logFCNorm < -5])

# evaluate expression level of genes
exprLevel = apply(cdsNorm, 1, mean)

# logFC versus the level of gene expression
plot(log(exprLevel), logFCNorm, pch = 20,
      xlab = "Gene expression level (log scale)", ylab = "logFC",
      main = "RNAseq data")
abline(h = 2, col = "green", lty = "dashed")
abline(h = -2, col = "red", lty = "dashed")
points(log(exprLevel[upGenes2]), logFCNorm[upGenes2], pch = 20,
```

```
      col = "green")
points(log(exprLevel[downGenes2]), logFCNorm[downGenes2], pch = 20,
      col = "red")

plot(log(exprLevel), logFCNorm, pch = 20,
      xlab = "Gene expression level (log scale)", ylab = "logFC",
      main = "RNAseq data")
abline(h = 5, col = "green", lty = "dashed")
abline(h = -5, col = "red", lty = "dashed")
points(log(exprLevel[upGenes5]), logFCNorm[upGenes5], pch = 20,
      col = "green")
points(log(exprLevel[downGenes5]), logFCNorm[downGenes5], pch = 20,
      col = "red")


######
# Perform the DE analysis with DESeq
######

## Differential analysis
dds = estimateDispersions(dds)
dds = nbinomWaldTest(dds)
res = results(dds)
mcols(res,use.names=TRUE)

# compare logFC values obtained with DESeq
plot(res[, "log2FoldChange"], logFCNorm, pch = 20,
      xlab = "logFC calculated with DESeq", ylab = "LogFC (after
normalization)")

hist(res$padj, breaks = 20, col = "black", border="white",
      xlab = "pvalues calculated with DESeq",
      main = "Distribution of adjusted pvalues (DESeq)")

hist(-log(res$padj), breaks = 20, col = "black", border="white",
      xlab = "-log(p-value)",
      main = "Distribution of -log(adjusted pvalues)")

# writing of the results
write.table(res, "DESeq2_statistics.txt", row.name=T, quote=F, sep='\t')
write.table(upGenes2, "up_genes_2.txt", row.name=F, col.name=F, quote=F)
write.table(downGenes2, "down_genes_2.txt", row.name=F, col.name=F, quote=F)
write.table(upGenes5, "up_genes_5.txt", row.name=F, col.name=F, quote=F)
write.table(downGenes5, "down_genes_5.txt", row.name=F, col.name=F, quote=F)

dev.off()

topGenes = head(order(res$padj),100)
write.table(res[topGenes,],"results_DESeq_100topGenes.txt",sep="\t",quote=F,row.
name=T)


# raise logFC threshold
res.FC2 = results(dds,lfcThreshold=2)

res.FC5 = results(dds,lfcThreshold=5)


# plotMA topGene in graphics
pdf("plotMA_resFC2_topGene.pdf")
plotMA(res.FC2,ylim=c(-15,15))
topGene = rownames(res.FC2)[which.min(res.FC2$padj)]
with(res[topGene,], {
      points(baseMean,log2FoldChange,col="black",cex=2,lwd=2)
      text(baseMean,log2FoldChange,topGene,pos=2,col="black")
      })
```

```
dev.off()

pdf("plotMA_resFC5_topGene.pdf")
plotMA(res.FC5,ylim=c(-15,15))
topGene_LC1 = rownames(res.FC5)[which.min(res.FC5$padj)]
with(res[topGene,], {
      points(baseMean,log2FoldChange,col="black",cex=2,lwd=2)
      text(baseMean,log2FoldChange,topGene,pos=2,col="black")
      })
dev.off()
```

Variant discovery

What we produced:
- Bash script: quality control (filtering steps)
- Bash script: variant association analysis
- VCF files (before and after QC)
- Table: identified variants (exonic, non-synonymous)

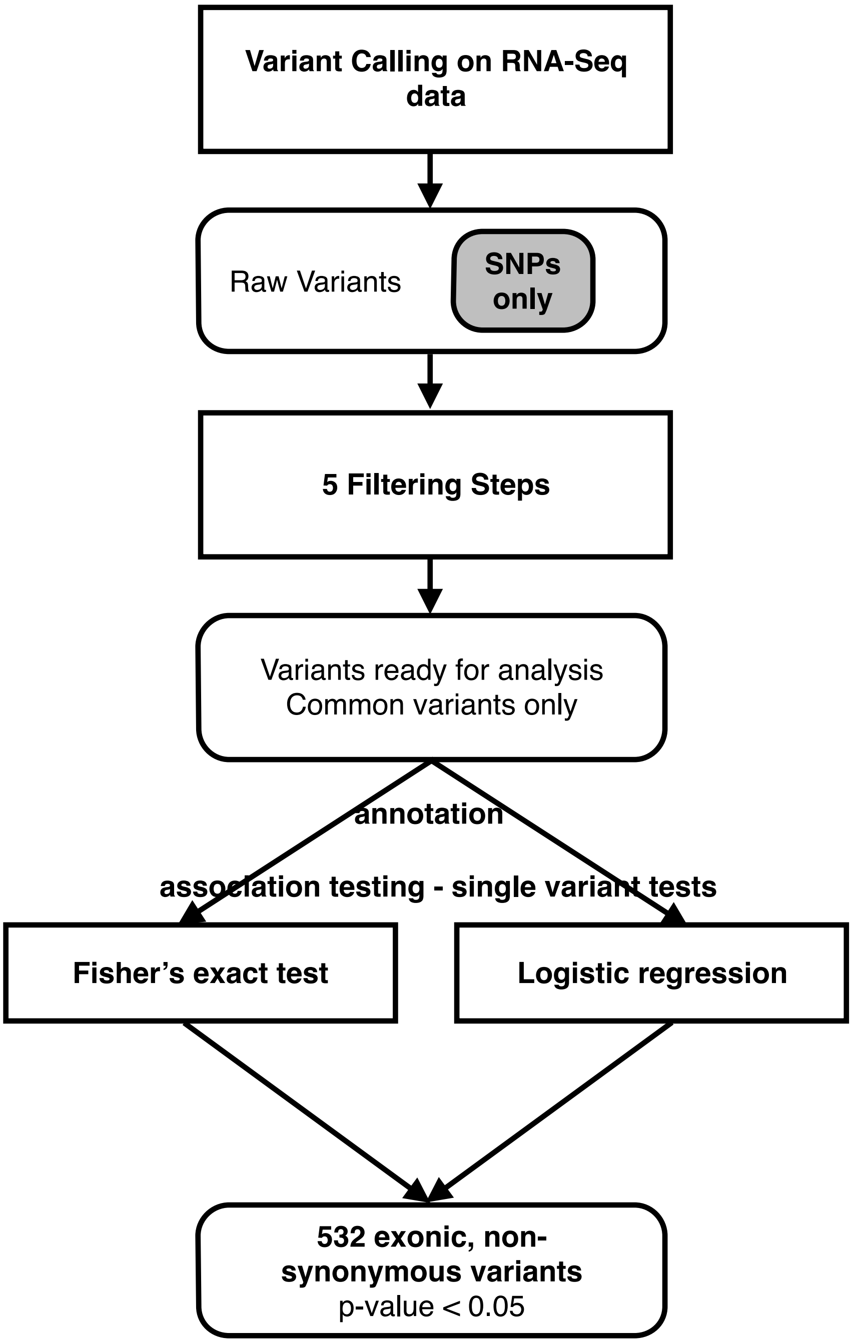


Figure: Schematic overview of the pipeline for variant discovery and evaluation

A) Variant calling

[François]

B) Quality control (filtering steps)

As recommended in the GATK Best Practices Guideline for variant discovery using RNA-Seq data, we applied hard filters to the raw variants obtained after variant calling, in an attempt to optimise both high sensitivity and specificity.

Furthermore, as we only have 4 samples, we decided to use quite stringent parameters / thresholds to filter the data, hoping to retain "true" and of as high a quality as possible variants. *Filtering was performed using scripts from GATK and VCFtools.*

Filters:

(1) **Diallelic** variants only.

(2) **Hardy-Weinberg equilibrium (HWE) deviation test**. It is a common practice to remove sites that deviate from HWE because the deviation can be caused by genotyping errors. Normally, for case-control data, only controls should be tested for deviation from HWE (because for cases, sites associated with disease status can deviate from HWE). In our case, as all tests were performed in a bidirectional manner, deviation from HWE was tested in all the samples and we excluded sites with a HWE p-value < 1.10^{-7}.

(3) **Call rate (percentage of samples with a non-missing genotype, CR)**. The proportion of missing genotypes is an useful indicator of poor genotype quality. We decided to keep variants with a CR > 98%, which allows to keep good quality variants only. As mean CR in raw data was of about 64%, we discarded over 60% of variants using this filter.

(4) Filtering based on **Fisher Strand values** (FS > 30.0) and **Quality by Depth** (QD < 2.0), as well as filtering out clusters of at least 3 SNPs in a window of 35 bases between them.

In order to assess the quality gain at each QC step, we estimated the ratio of transitions (Ti, purine to purine or pyrimidine to pyrimidine mutation) to transversions (Tv, purine to pyrimidine or vice versa) in the identified single nucleotide variants (SNVs). Particularly in coding regions, a higher number of transitions is expected, as transversions are more likely to change the underlying amino acid and lead to a deleterious mutation. Ti/Tv ratios are an approximate measure of quality: higher Ti/Tv ratios are associated with lower false positives.

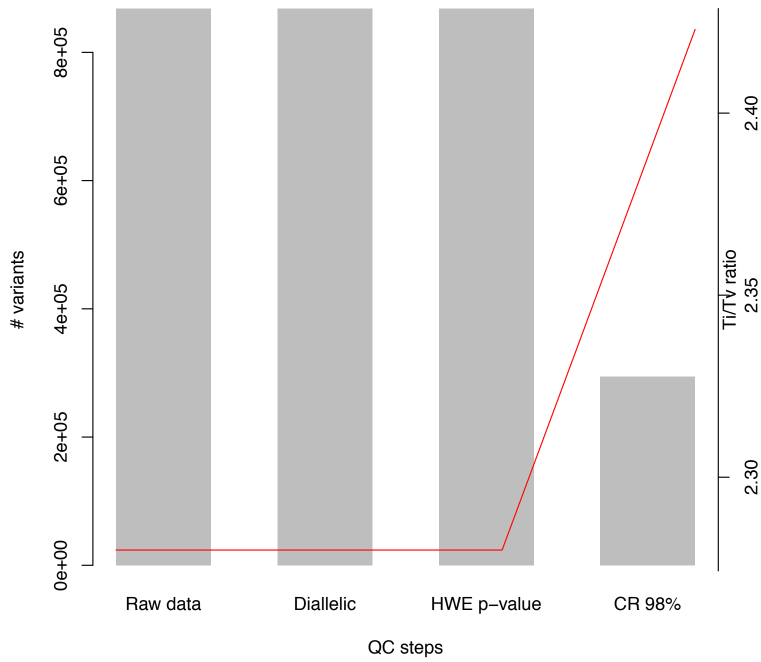


Figure: Number of variants retained and Ti/Tv ratio for every QC step

QC_stage	NVAR	Call Rate	TiTv	meanQUAL
Raw_data	868330	0.68	2280	98
Diallelic_only	868037	0.69	2280	98
HWE_pvalue	868037	0.69	2280	98
CR_98	294034	1	2423	223

C) <u>Annotation</u>

Annotation attributes such as genomic region, gene name, variant type and consequence are attached to the variants list according to the reference hg19 using ANNOVAR (AnnotateVariation perl script). The primary genomic effects that are annotated include splice sites, nonsense, non-synonymous and synonymous variants.

D) <u>Association testing between individual variants and phenotypic traits</u> (i.e control / cancerous cells)

Here, common variants were defined as being those that are present in more than one sample. Of the 294 034 variants retained after quality control, 233 294 were identified as common. We identified 24 347 exonic variants only, over 19 000 of these were common. Thus, we decided to work only on common variants.
We performed standard single variant test to assess association: logistic regression and fisher's exact test.

We found 531 exonic non-synonymous variants having a Fisher's p-value < 0.05 (p = 0.02, being the lowest value we could get with 4 samples). 315 of these variants were only present in the melanoma cell lines (all were homozygous variants).

E) <u>Script</u>

a. <u>Filtering steps</u>

```
#######################################################
### Variant Filtering                               ###
### Hard filters -> optimize both high sensitivity ###
### and specificity together ;                      ###
### !!! some real sites will get filtered out !!!  ###
###                                                 ###
### v1.0 15/09/2015                                 ###
#######################################################


#1) Keep diallelic variants only

/path/to/bin/vcftools --vcf melanocytes_melanomes_var.vcf --min-alleles 2 --max-alleles 2 --recode --out N_C_diallelic


#2) Annotation, beforeQC

java -Xmx32g -jar /path/to/snpEff.jar -v GRCh37.75 N_C_diallelic.recode.vcf > N_C_diallelic_annot.vcf

# annotate unknown variants only (unknown as not reported in dbSNP)

java -jar /path/to/SnpSift.jar annotate -dbsnp N_C_diallelic.recode.vcf >
```

```
bQC_dbsnp.vcf
java -Xmx4g -jar /path/to/snpEff.jar eff -v GRCh37.75 bQC_dbsnp.vcf >
bQC_eff.vcf
java -jar /path/to/SnpSift.jar filter -f bQC_eff.vcf "! exists ID" >
bQC_eff_not_in_dbSnp.vcf
java -Xmx32g -jar /path/to/snpEff.jar eff -v GRCh37.75 bQC_eff_not_in_dbSnp.vcf
> bQC_not_in_db_annot.vcf


#3) High Quality variants (CR>98% and HWE p > 10-7)

./vcftools_0.1.13/bin/vcftools --vcf N_C_diallelic.recode.vcf --hwe 0.0000001 --
recode --out N_C_HF_hwe
./vcftools_0.1.13/bin/vcftools --vcf N_C_HF_hwe.recode.vcf --max-missing 0.98 --
recode --out N_C_CR98


#4) GATK filters (as in BEST PRACTICES for RNAseq data and variant calling)

# Filtering based on Fisher Strand values and Qual by Depth
# Filter out clusters of at least 3 SNPs in a window of 35 bases between them

java -jar GenomeAnalysisTK.jar -T VariantFiltration -R hg_19.fasta -V
N_C_CR98.recode.vcf -window 35 -cluster 3 -filterName FS -filter "FS > 30.0" -
filterName QD -filter "QD < 2.0" -o afterQC_variants.vcf


#5) Annotation, afterQC

java -Xmx32g -jar /path/to/snpEff.jar -v GRCh37.75 afterQC_variants.vcf >
afterQC_variants_annot.vcf
```

b. <u>Variant evaluation</u>

```
##### PIPELINE VARIANT ANALYSIS #####
##### python - variant tools    #####

## vtools project set-up ##

# initialize project and import vcf file with variant calls

vtools init proj
vtools import N_C_CR98.recode.vcf --build hg19 --var_info AA AC AN DP --
geno_info DP_geno

# import phenotypes
# phone.txt is a tab separated file: column 1 = sample_name ; column_2 =
#phenotype N (controls) or C (cancerous cells)

vtools phenotype --from_file pheno.txt

# ANNOVAR annotations
# if necessary, download database
#/path/to/annovar/annotate_variation.pl --downdb refGene /path/to/annovar/
humandb/ -build hg19

vtools export variant --output ANNOVAR.input --format ANNOVAR
perl /path/to/annovar/annotate_variation.pl -geneanno ANNOVAR.input -buildver
hg19 /path/to/annovar/humandb/
vtools update variant --format ANNOVAR_exonic_variant_function --from_file
ANNOVAR.input.exonic_variant_function --var_info mut_type function genename
vtools update variant --format ANNOVAR_variant_function --from_file
ANNOVAR.input.variant_function --var_info region_type region_name
```

```
# annotation: refGene, dbSNP

vtools use refGene
vtools use dbSNP

# alternative allele frequency calculations
vtools update variant --from_stat 'total_ie=#(GT)' 'num_ie=#(alt)'
'het_ie=#(het)' 'hom_ie=#(hom)' 'other_ie=#(other)' 'num_var=#(mutGT)'
vtools update variant --set 'af_ie=num_ie/(total_ie * 2.0)'

### creating variant subsets

vtools select variant "af_ie > 0.005" -t variants "variant table (MAF>0.5%)"

# usually, RV defined as having MAF ≤ 5%
# here, working with 4 samples, RV defined as having MAF ≤ 25%

#vtools select variants "af_ie<=0.05 AND af_ie > 0.005" -t rare_var "rare
variants defined as having a MAF≤5%"
#vtools select variants "af_ie > 0.05" -t com_var "common variants defined as
having a MAF>5%"

vtools select variants "af_ie<=0.25 AND af_ie > 0.005" -t rare_var "rare
variants defined as having a MAF≤25%"
vtools select variants "af_ie > 0.25" -t com_var "common variants defined as
having a MAF>25%"

# non-synonymous variants only

vtools select variants "mut_type like 'nonsynonymous%' OR mut_type like
'stoploss%' OR mut_type like 'stopgain%' OR mut_type like 'splicing%' OR
mut_type like 'frameshift%' OR mut_type like 'nonframeshift%'" -t fvar

vtools select rare_var "mut_type like 'nonsynonymous%' OR mut_type like
'stoploss%' OR mut_type like 'stopgain%' OR mut_type like 'splicing%' OR
mut_type like 'frameshift%' OR mut_type like 'nonframeshift%'" -t rare_fvar
"nonsynonymous, stoploss, stopgain, splicing and indel variants selected from
table rare_var"

vtools select com_var "mut_type like 'nonsynonymous%' OR mut_type like 'stoploss
%' OR mut_type like 'stopgain%' OR mut_type like 'splicing%' OR mut_type like
'frameshift%' OR mut_type like 'nonframeshift%'" -t com_fvar "nonsynonymous,
stoploss, stopgain, splicing and indel variants selected from table com_var"

# exonic variants only

vtools select variants "region_type = 'exonic' OR region_type =
'exonic;splicing' OR region_type = 'ncRNA_exonic'" -t exo_var "exonic variants
from table variant"

vtools select rare_var "region_type = 'exonic' OR region_type =
'exonic;splicing' OR region_type = 'ncRNA_exonic'" -t exo_RV "exonic variants
from table rare_var"

vtools select com_var "region_type = 'exonic' OR region_type = 'exonic;splicing'
OR region_type = 'ncRNA_exonic'" -t exo_CV "exonic variants from table comm_var"

vtools select fvar "region_type = 'exonic' OR region_type = 'exonic;splicing' OR
region_type = 'ncRNA_exonic'" -t exo_fvar "exonic variants from table fvar"

vtools select rare_fvar "region_type = 'exonic' OR region_type =
'exonic;splicing' OR region_type = 'ncRNA_exonic'" -t exo_fRV "exonic variants
from table rare_fvar"

vtools select com_fvar "region_type = 'exonic' OR region_type =
```

```
'exonic;splicing' OR region_type = 'ncRNA_exonic'" -t exo_fCV "exonic variants
from table com_fvar"


########## Association testing ##########

## COMMON variants

# Fisher's exact test

vtools update variant --from_stat 'num_gt_case=#(GT)'
'num_var_alleles_case=#(alt)' --samples "phenotype = 2 "
vtools update variant --from_stat 'num_gt_ctrl=#(GT)'
'num_var_alleles_ctrl=#(alt)' --samples "phenotype = 1 "
vtools update variant --set "prop_pval=Fisher_exact(num_var_alleles_case,
num_var_alleles_ctrl, 2*num_gt_case, 2*num_gt_ctrl)"

vtools output com_var \

chr pos ref alt refGene.name2 refGene.cdsStart refGene.cdsEnd refGene.strand \

mut_type region_type num_var_alleles_case num_var_alleles_ctrl het_ie hom_ie
prop_pval \

--header CHR POS REF ALT GENE CDS_START CDS_END STRAND \

MUT_TYPE REGION NUM_VAR_ALLELES_C NUM_VAR_ALLELES_N NUM_HTZ NUM_HMZ PVAL_FISHER
> pval_CV_fisher.txt


# Logistic regression

vtools associate com_var phenotype \

             --discard_variants "%(NA)>0.1" \

             --method "LogitRegBurden --name logReg --alternative 2" \

             --group_by refGene.name2 \

             --to_db logReg_CV \

             -j8 > logReg_CV.txt
```