# Intro to Data Visualization and Statistics in R

## Session #4

Genome Institute of Singapore

17th October 2019

# RECAP SESSION 3 & ASSIGNMENT

# What we learnt last week

1. Binomial distribution
2. Useful functions: paste, unique, transpose, identical
3. TCGA breast cancer: clinical data cleanup
4. TCGA breast cancer: expression data cleanup
5. Combining multiple datasets: bind_rows, *_join, join_all

Capstone project

# Learning objectives for Session 4

**Objectives:**

1. Review of assignment
2. Principal component analysis
3. (Aside) Volcano plots
4. Dendrograms

# Exercise 1: Code cleanup

```
fn <- "data_tcga/brca_tcga_clinical_data.tsv"

pheno <- read_delim(fn, delim="\t") %>%

  clean_names() %>%

  rename(DFS=disease_free_status,
         DFS_months=disease_free_months) %>%

  filter(cancer_type=="Breast Cancer",
         sex=="Female",
         DFS_months > 0) %>%

  mutate(event = ifelse(DFS=="DiseaseFree", 0, 1)) %>%

  select(-patient_id, -DFS, -starts_with("overall"))
```

# Exercise 2: Illumina Quality Score

The probabilities follow a Binomial distribution.

Prob. of $\geq 3$ incorrect bases is 1 - prob. of $< 3$ incorrect bases.

```
# Read length = 150bp. All base have Q30 score.
1 - sum(dbinom(0:2, 150, 0.001))
## [1] 0.0004939301

# Read length = 75bp. All base have Q30 score.
1 - sum(dbinom(0:2,  75, 0.001))
## [1] 6.398022e-05

# Read length = 75bp. All base have Q40 score.
1 - sum(dbinom(0:2,  75, 0.0001))
## [1] 6.71614e-08
```

# DIMENSION REDUCTION TECHNIQUES

# Intuition | 3-dimension to 2-dimension

**Figure 1**

# Intuition | 3-dimension to 2-dimension

Figure 1

Figure 2

# Intuition | 3-dimension to 2-dimension

Figure 1

Figure 2

Figure 3

# Intuition | 3-dimension to 2-dimension

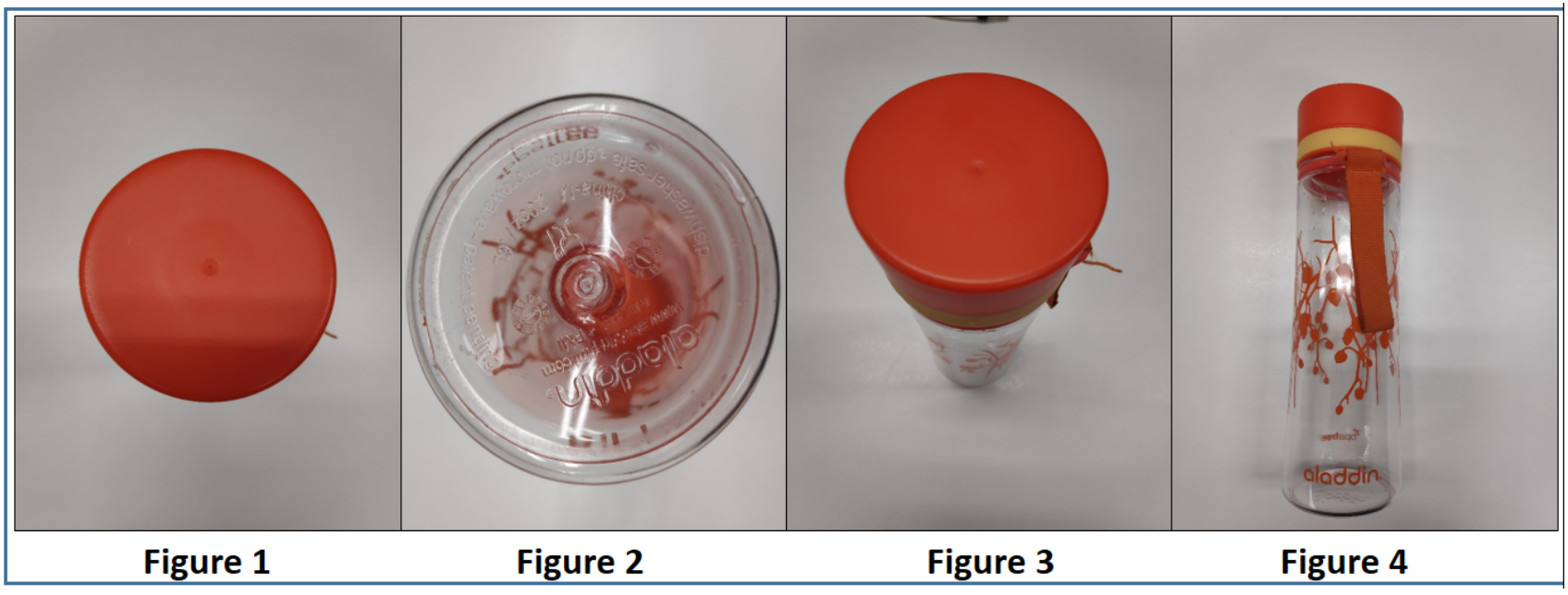
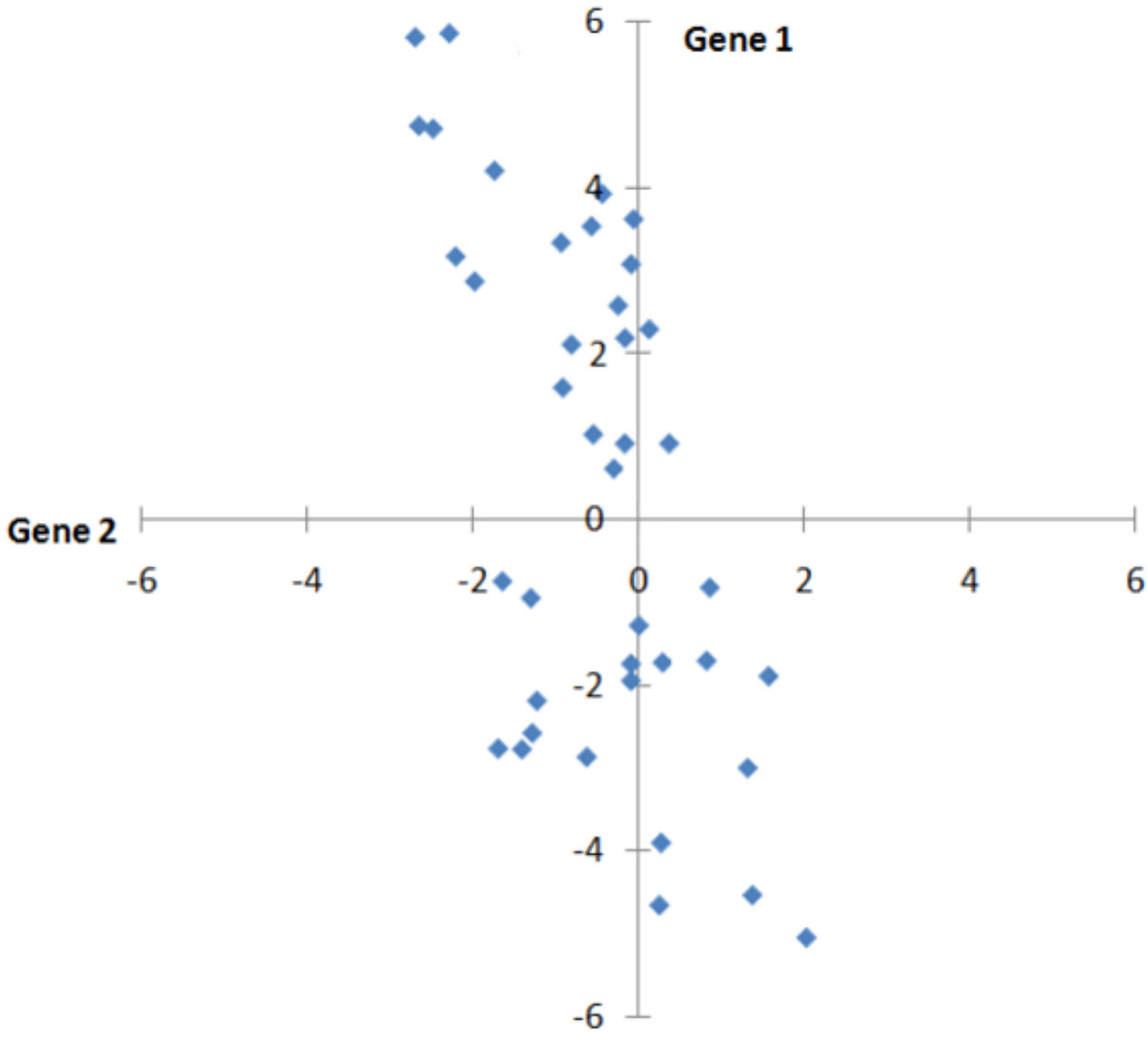


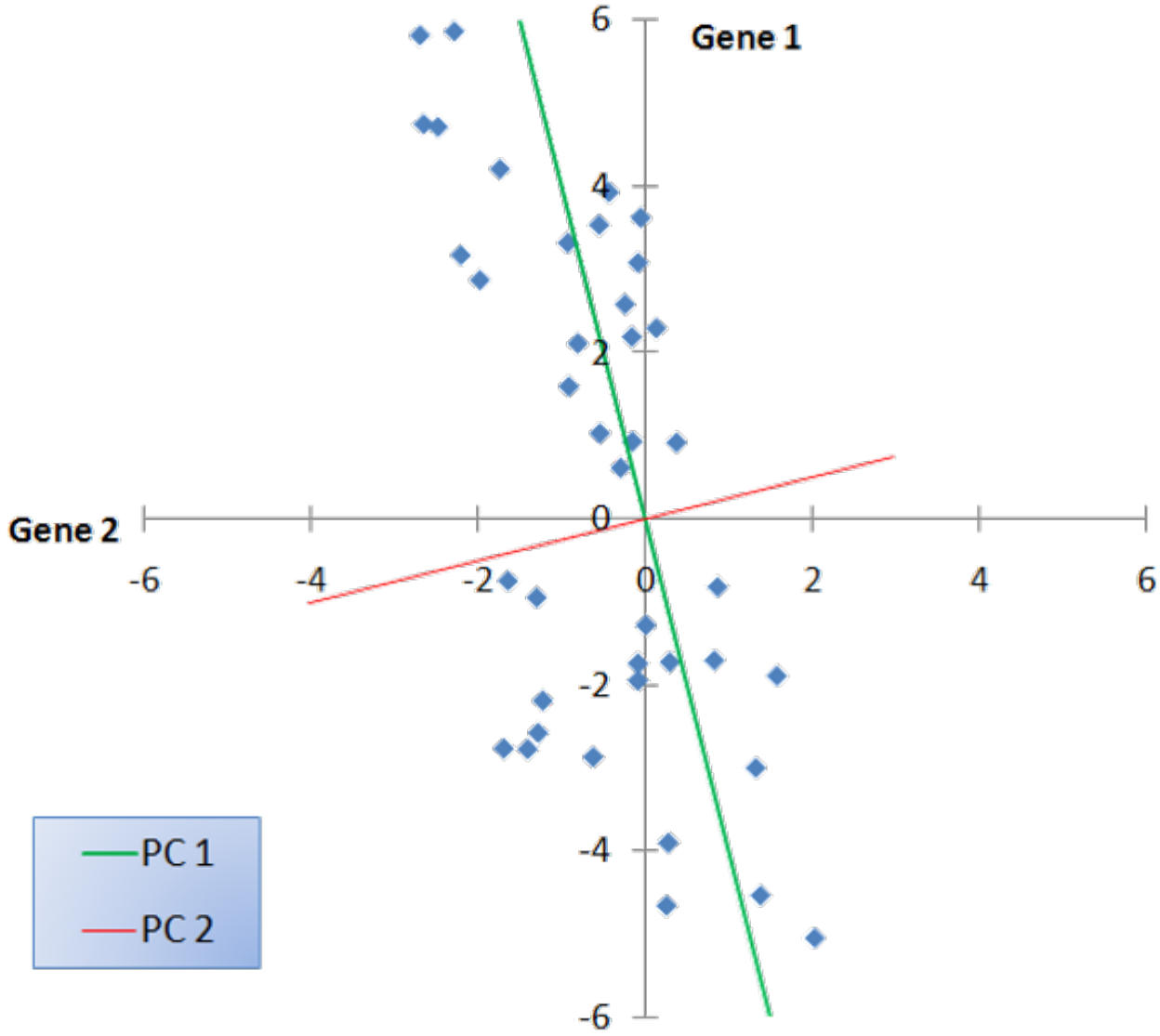

Figure 1 Figure 2 Figure 3 Figure 4

It's my water bottle! Which two pictures should I send to my insurance company if I lost it?

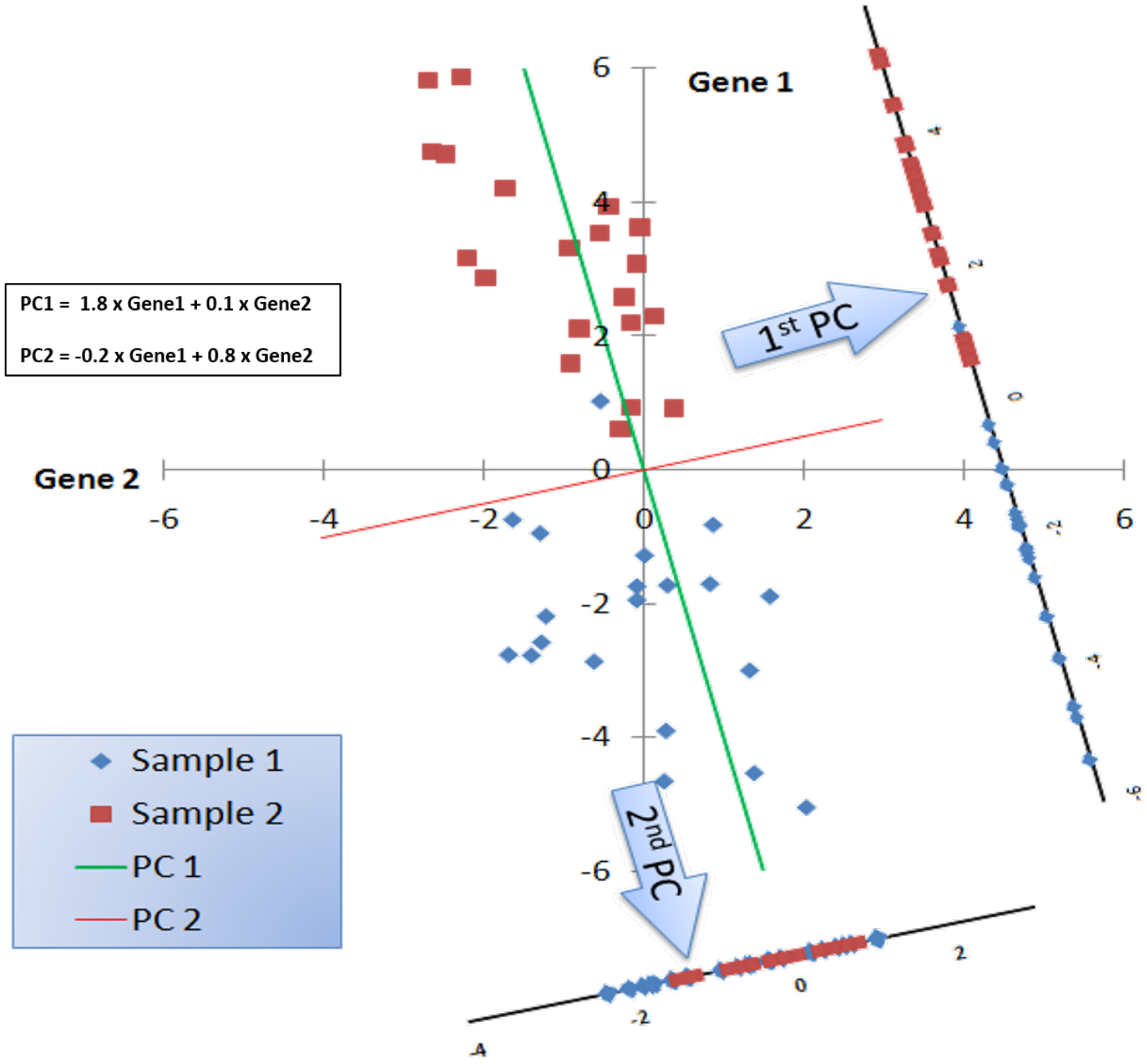
# Intuition | 2-dimension to 1-dimension



# Intuition | 2-dimension to 1-dimension



# Intuition | 2-dimension to 1-dimension



# Dimension reduction tools

**Aim:** To reduce large set of variables to a small subset that still retains most of the information.

Many different techniques:

- **Principal Component Analysis (PCA)**
- Multidimensional scaling (MDS)
- Factor analysis (FA)
- Multiple Correpondence Analysis (MCA)
- Principal Coordinates Analysis (PCoA)

# Principal component analysis (PCA) | Pseudo-algorithm

1. Find a linear combination of the original variables that can explain most of the variation in the data. Denote this as PC1.

# Principal component analysis (PCA) | Pseudo-algorithm

1. Find a linear combination of the original variables that can explain most of the variation in the data. Denote this as PC1.

2. Remove this variation. Find the next linear combination that can explain the remaining variation. Denote this as PC2.

# Principal component analysis (PCA) | Pseudo-algorithm

1. Find a linear combination of the original variables that can explain most of the variation in the data. Denote this as PC1.

2. Remove this variation. Find the next linear combination that can explain the remaining variation. Denote this as PC2.

3. Repeat Step 2 until you have as many PCs as original number of samples.

# Principal component analysis (PCA) | Pseudo-algorithm

1. Find a linear combination of the original variables that can explain most of the variation in the data. Denote this as PC1.

2. Remove this variation. Find the next linear combination that can explain the remaining variation. Denote this as PC2.

3. Repeat Step 2 until you have as many PCs as original number of samples.

4. Choose the optimal number of PCs to use using the elbow method in the scree plot.

# Principal component analysis (PCA) | Pseudo-algorithm

1. Find a linear combination of the original variables that can explain most of the variation in the data. Denote this as PC1.

2. Remove this variation. Find the next linear combination that can explain the remaining variation. Denote this as PC2.

3. Repeat Step 2 until you have as many PCs as original number of samples.

4. Choose the optimal number of PCs to use using the elbow method in the scree plot.

# Principal component analysis (PCA) | Pseudo-algorithm

1. Find a linear combination of the original variables that can explain most of the variation in the data. Denote this as PC1.

2. Remove this variation. Find the next linear combination that can explain the remaining variation. Denote this as PC2.

3. Repeat Step 2 until you have as many PCs as original number of samples.

4. Choose the optimal number of PCs to use using the elbow method in the scree plot.

We will use `prcomp()` function for PCA.

The R package FactoMineR (http://factominer.free.fr/) is a more comprehensive alternative.

# Principal component analysis (PCA) | Notes

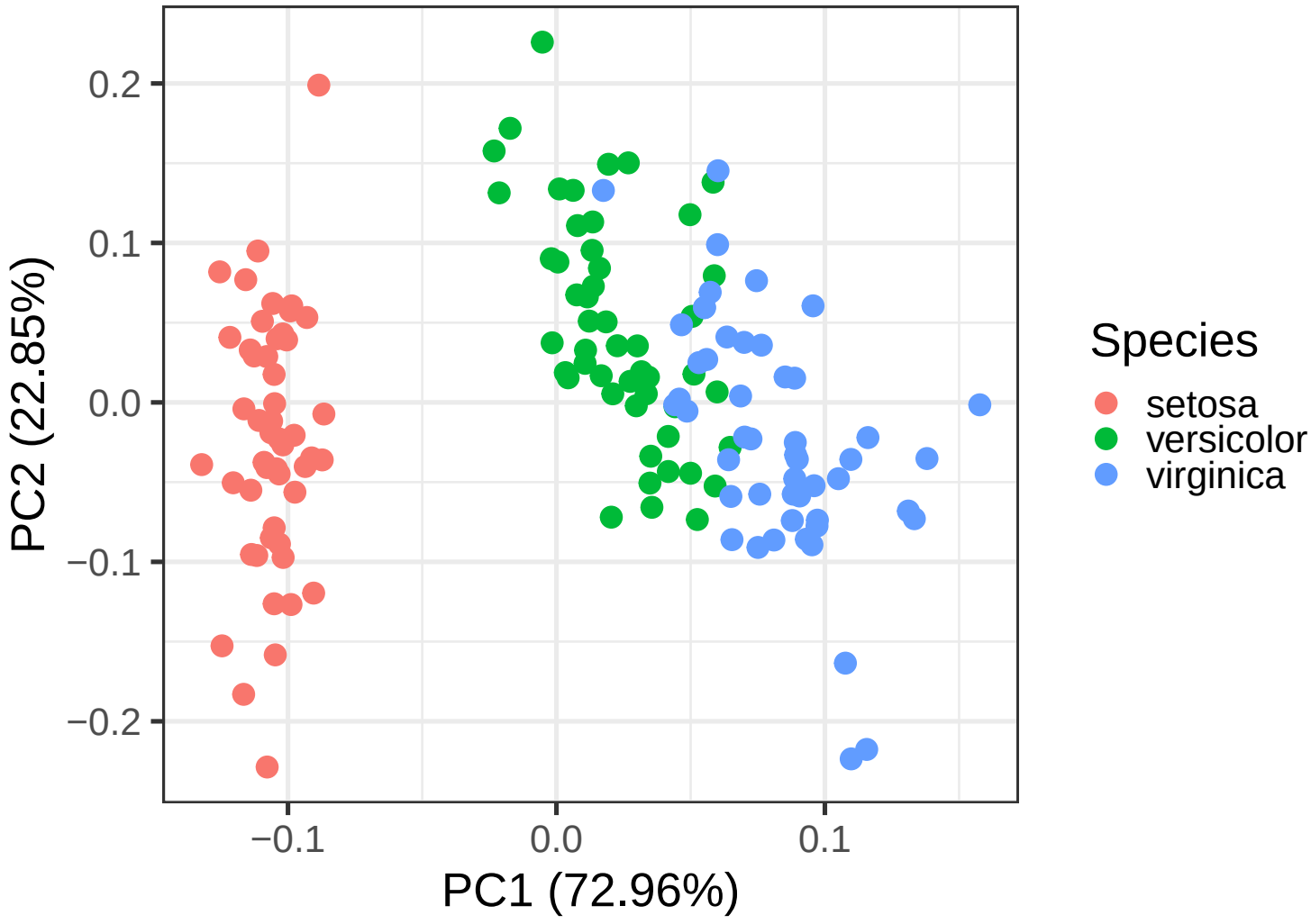
- Can only done on the numerical data.
- `prcomp()` needs sample in **rows**. Transpose data if needed.
- The PCs are uncorrelated with each other (orthogonal).
- Proportion of variance decreases by PC number.

# PCA | Codes for iris

```
pacman::p_load(ggrepel, ggfortify)
data(iris)

iris_num <- iris %>% select(-Species)
pc <- prcomp(iris_num, scale.=TRUE)

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

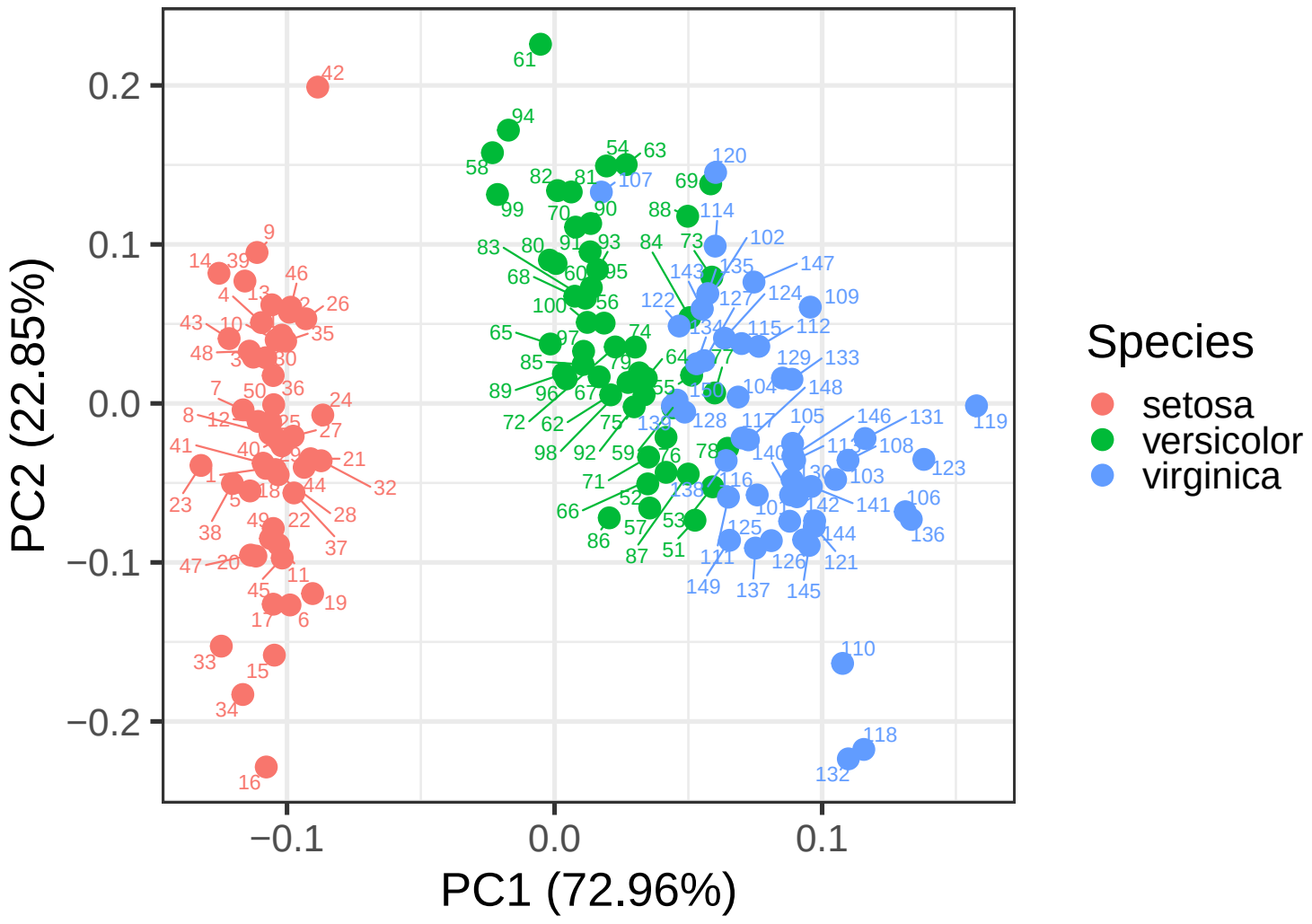
# PCA | Plot

```
autoplot(pc, data=iris, colour="Species") + theme_bw()
```



# PCA | Plot and label with rownames

```
autoplot(pc, data=iris, colour="Species",
         label=T, label.repel=T) + theme_bw()
```



# PCA | Loadings

```
pc$rotation %>% round(2)
##                PC1   PC2   PC3   PC4
## Sepal.Length  0.52 -0.38  0.72  0.26
## Sepal.Width  -0.27 -0.92 -0.24 -0.12
## Petal.Length  0.58 -0.02 -0.14 -0.80
## Petal.Width   0.56 -0.07 -0.63  0.52
```

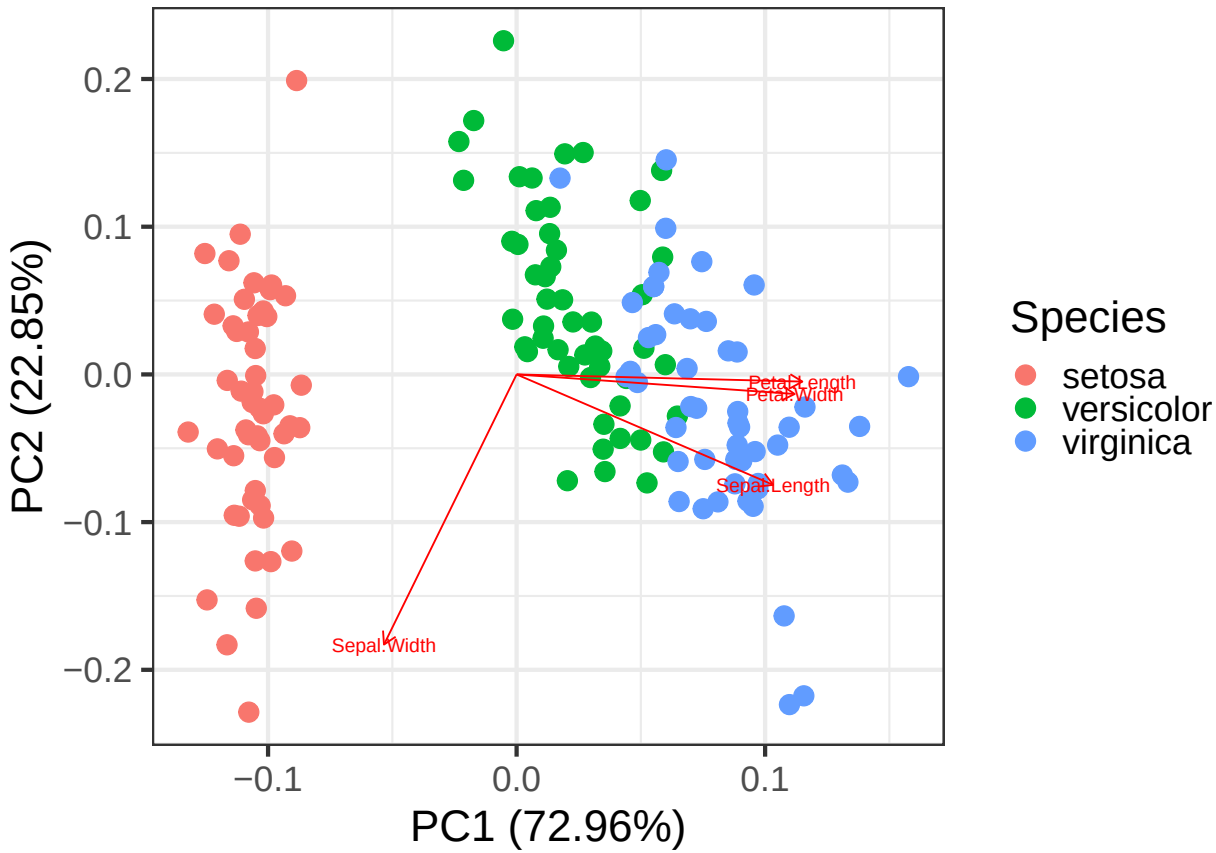
Therefore you can estimate PC1 as follows:

$$PC_1 = 0.58 \times PL + 0.56 \times PW + 0.52 \times SL - 0.27 \times SW$$

However, this is generally not useful for transcriptomics data as each PC would be a combination of all genes studied (usually $>$ 1,000) which makes intepretation impossible.

# PCA | Biplot

```
autoplot(pc, data=iris, colour="Species",
         loadings=T, loadings.label=T) + theme_bw()
```




Again, not useful for transcriptomics data.

# HANDS-ON EXERCISE WITH RNA-SEQ COUNT DATA (GSE50760)

# Setup and read in clinical data

1. Save `GSE50760_data.xlsx` into the subfolder `data`.
2. Create a new script. Set working director and load packages.
3. Read in the clinical data as `pheno`.
4. How many subjects are there and what is the tissue type distribution among them?

# Setup and read in clinical data

1. Save `GSE50760_data.xlsx` into the subfolder `data`.
2. Create a new script. Set working director and load packages.
3. Read in the clinical data as `pheno`.
4. How many subjects are there and what is the tissue type distribution among them?

```
setwd("C:/Users/aramasamy/Desktop/R_workshop")
pacman::p_load(tidyverse, readxl, ggfortify, janitor,
               ggrepel, DESeq2, pheatmap, genefilter)
rm(list=ls())

fn <- "data/GSE50760_data.xlsx"

## Read in pheno
levs  <- c("normal", "primary", "metastasized")
pheno <- read_excel(fn, sheet="pheno") %>%
  mutate(type=factor(type, levels=levs)) %>%
  column_to_rownames("run_id")
```

# If DESeq2 etc. is NOT found

`pacman::p_load()` works well for CRAN packages but not yet for Bioconductor packages. Googling the names of the package points you, for example: https://bioconductor.org/packages/release/bioc/html/DESeq2.html

which then asks you to do the following:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("DESeq2")
```

PS: Easier to search and then copy and paste the commands.

# Setup and read in clinical data

The data comes from Kim et al (PMID: 25049118). "18 patients with matched primary CRC, synchronous liver metastases and their normal colonic epithelium (> 5cm from tumour border)."

```
## Number of samples
pheno %>% nrow()
## [1] 54

## Number of subjects
pheno$subject %>% unique() %>% length
## [1] 18

## Number of samples per subject
pheno %>% tabyl(subject, type)
##  subject normal primary metastasized
##   AMC_10      1       1           1
##   AMC_12      1       1           1
##   AMC_13      1       1           1
##   AMC_17      1       1           1
```

# Read in expression data

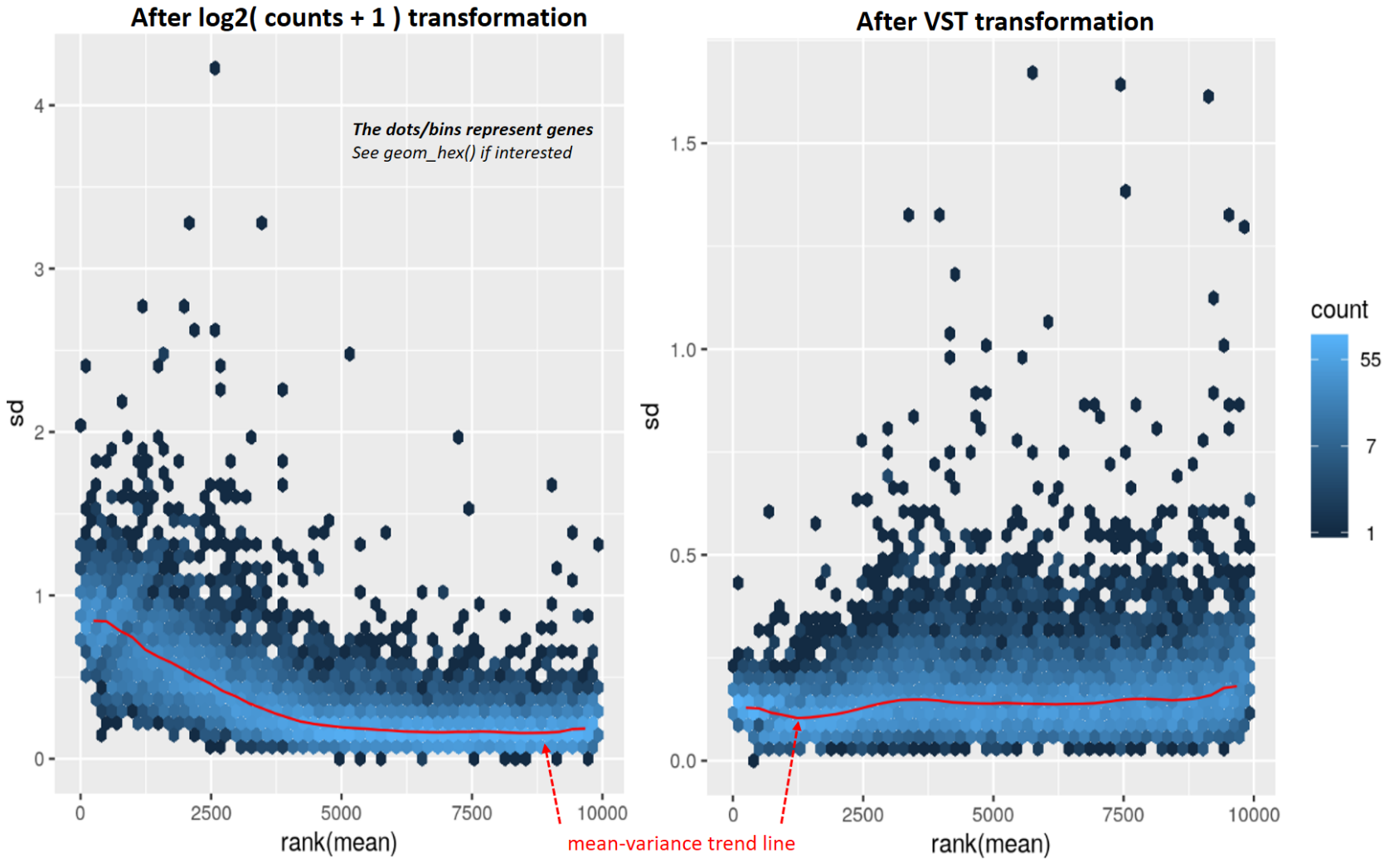
```
## Read in expression data
raw_count <- read_excel(fn, sheet="counts") %>%
  column_to_rownames("gene_id")
raw_count[1:3, 1:3]
##                     SRR975551 SRR975552 SRR975554
## ENSG00000000003.14     16360      3335     11645
## ENSG00000000005.5        175         2        90
## ENSG00000000419.12      7847      2025      4542

## Variance Stabilizing Transformation
expr <- raw_count %>%
  data.matrix() %>% vst()
expr[1:3, 1:3]
##                     SRR975551 SRR975552 SRR975554
## ENSG00000000003.14 13.338408 11.348319 13.172831
## ENSG00000000005.5   7.179136  4.620028  6.710229
## ENSG00000000419.12 12.283994 10.641570 11.823467
```

# Variance stabilizing transformation (VST)

Last week, we used `log2( counts + 1 )`. However, VST or rlog transformations are better when working with full RNA-seq data:

- removes the dependence of variance on the mean, particularly the high variance of log count data for low abundance genes.
- returns data on the log2 scale after normalizing for library size
- suitable for suitable for clustering, plotting individual genes etc.
- **not suitable** for differential expression (prefer raw counts)

# Variance stabilizing transformation (VST)

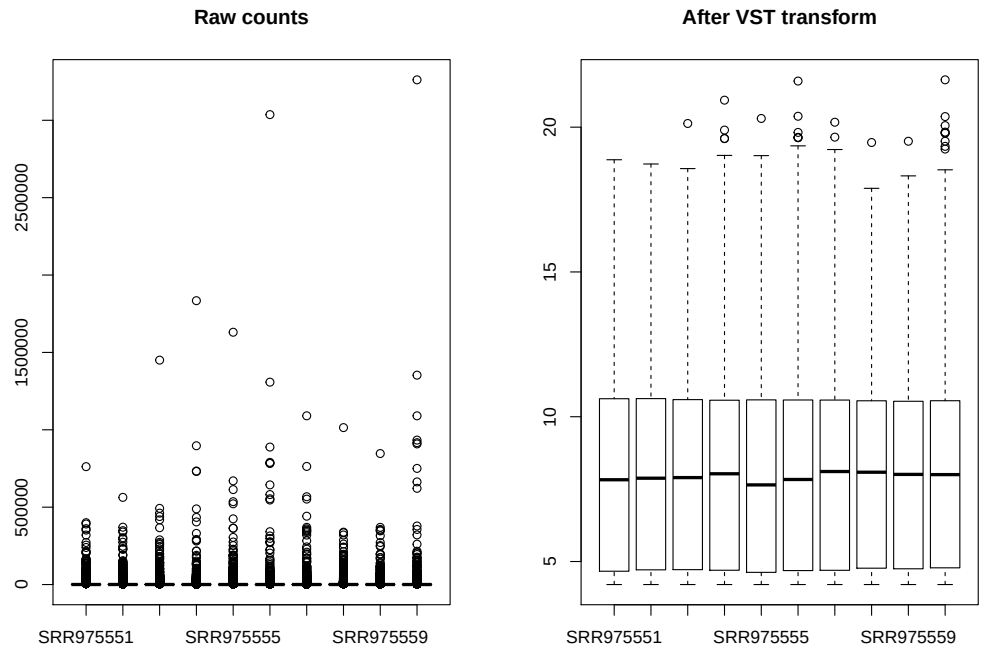



Experiment wide mean-variance trend has flattened

# Variance stabilizing transformation (VST)

Here is another view of how the data looks like for first five samples with and without the VST transformation (note: these are using base R plotting functions).

```
par(mfrow=c(1,2))
boxplot( raw_count[ , 1:10], main="Raw counts")
boxplot( expr[ , 1:10],      main="After VST transform")
```



# PCA to find outliers | Part 1

We typically use only the most variable genes for PCA of RNA-seq data. We can use `varFilter()` from the genefilter package.

```
identical( rownames(pheno), colnames(expr) )
## [1] TRUE

## Filter to top 10% most variable gene and transpose
input <- expr %>%
  varFilter(var.cutoff=0.9) %>%
  t()

dim(input)
## [1]   54 2553
```

For PCA, the rows represent samples so we need to transpose it.

# PCA to find outliers | Part 1

We typically use only the most variable genes for PCA of RNA-seq data. We can use `varFilter()` from the genefilter package.

```
identical( rownames(pheno), colnames(expr) )
## [1] TRUE

## Filter to top 10% most variable gene and transpose
input <- expr %>%
  varFilter(var.cutoff=0.9) %>%
  t()

dim(input)
## [1]   54 2553
```

For PCA, the rows represent samples so we need to transpose it.

# PCA to find outliers | Part 2

Run the PCA.

- What are the contributions of each PC?
- How many PC do you think is sufficient for checking?
- How does the PCA plot look like?

# PCA to find outliers | Part 2

Run the PCA.

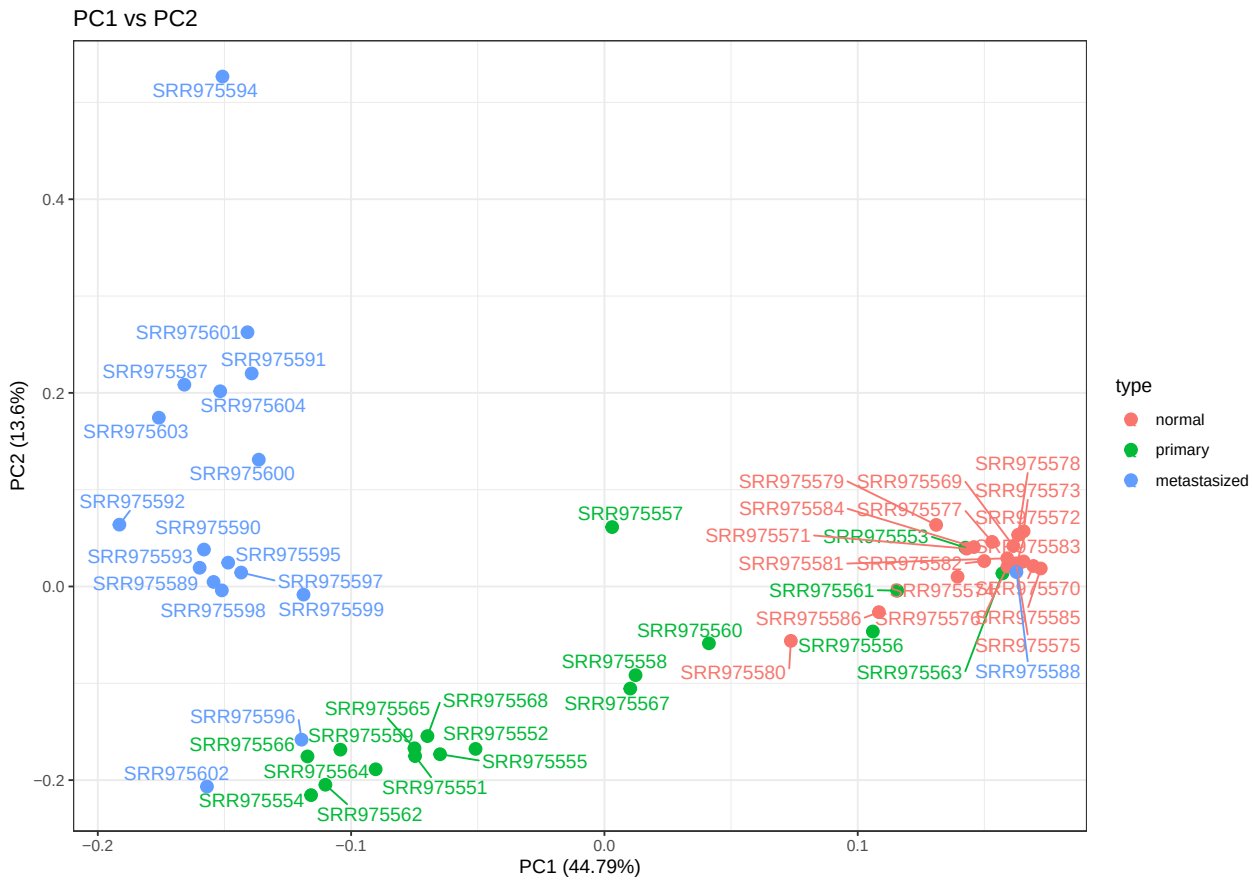
- What are the contributions of each PC?
- How many PC do you think is sufficient for checking?
- How does the PCA plot look like?

```
pc <- prcomp(input, scale.=TRUE)

summary(pc)
# Importance of components:
#                             PC1     PC2      PC3     PC4
# Standard deviation     33.8153 18.6338 14.76354 9.32578
# Proportion of Variance  0.4479  0.1360  0.08537 0.03407
# Cumulative Proportion   0.4479  0.5839  0.66927 0.70334
# ... continues up to PC54 ...
```

You can also run `screeplot(pc)`

# PCA to find outliers | Part 3

```
autoplot(pc, x=1, y=2, data=pheno, col="type", size=3,
         label=T, label.repel=T, main="PC1 vs PC2") +
  theme_bw()
```




SRR975588 is a gross outlier. Possibly a few more.

# Remove the outlier SRR975588

```
identical( rownames(pheno), colnames(raw_count) )
## [1] TRUE

## remove from expression data
toRemove <- c("SRR975588")
expr <- raw_count %>%
  select(-toRemove) %>%
  data.matrix() %>%
  vst()

## remove from clinical data using position matching
toRemove <- which(pheno$sample_id %in% toRemove)
pheno <- pheno[ -toRemove, ]

## Check and cleanup
identical(rownames(pheno), colnames(expr))
## [1] TRUE
rm(toRemove, raw_count, pc, input, levs)
```

# Differential gene expression list

Your friendly bioinformatician has analyzed the 53 samples for you. The genes significant for tumorigenesis (primary vs. normal) and metastasis (metastasized vs primary) are listed in the "dge" sheet.

| Prefix | Explanation |
|---|---|
| baseMean | The average expression of gene (results filtered for $> 1$) |
| LFC | Log2 fold change |
| P | P-value (smaller is better) |
| FDR | False Discovery Rate (adjusts p-value for multiple testing) |

See GSE50760_runDGE.R (to be emailed later) for how she derived this list for you. Alternatively, see http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html

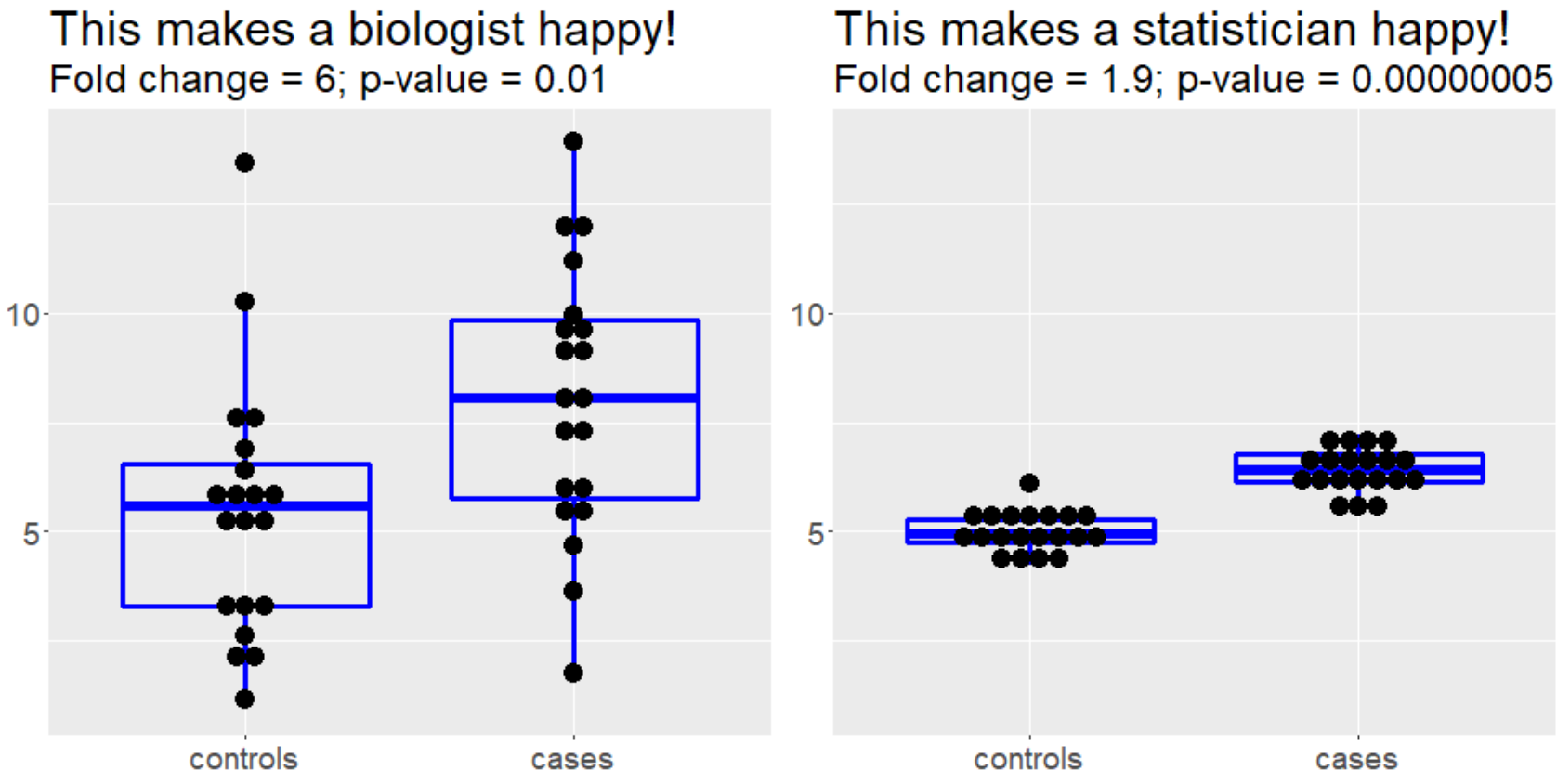
# Sanity check

1. Read in the DGE
2. How many rows are the in the DGE?
3. What are the 20 highest expression genes? Do they make sense? (You can also do this per sample/type if you wish).

```
dge <- read_excel(fn, sheet="dge") %>% data.frame()
rownames(dge) <- dge$gene_id          # *IMPORTANT*
nrow(dge)
## [1] 20682
```

```
dge %>%
  arrange(desc(baseMean)) %>%
  pull(symbol) %>%  head(20)
## "COX1"    "ND4"   "RNR2" "COX3" "ATP6" "COX2" "CYTB"
## "ND5"     "ND1"   "ALB"  "ND2"  "ATP8" "PIGR" "ACTB"
## "EEF1A1" "ND4L" "TPT1" "B2M"  "EEF2" "KRT8"
```

# Volcano plots | Rationale



# Volcano plots

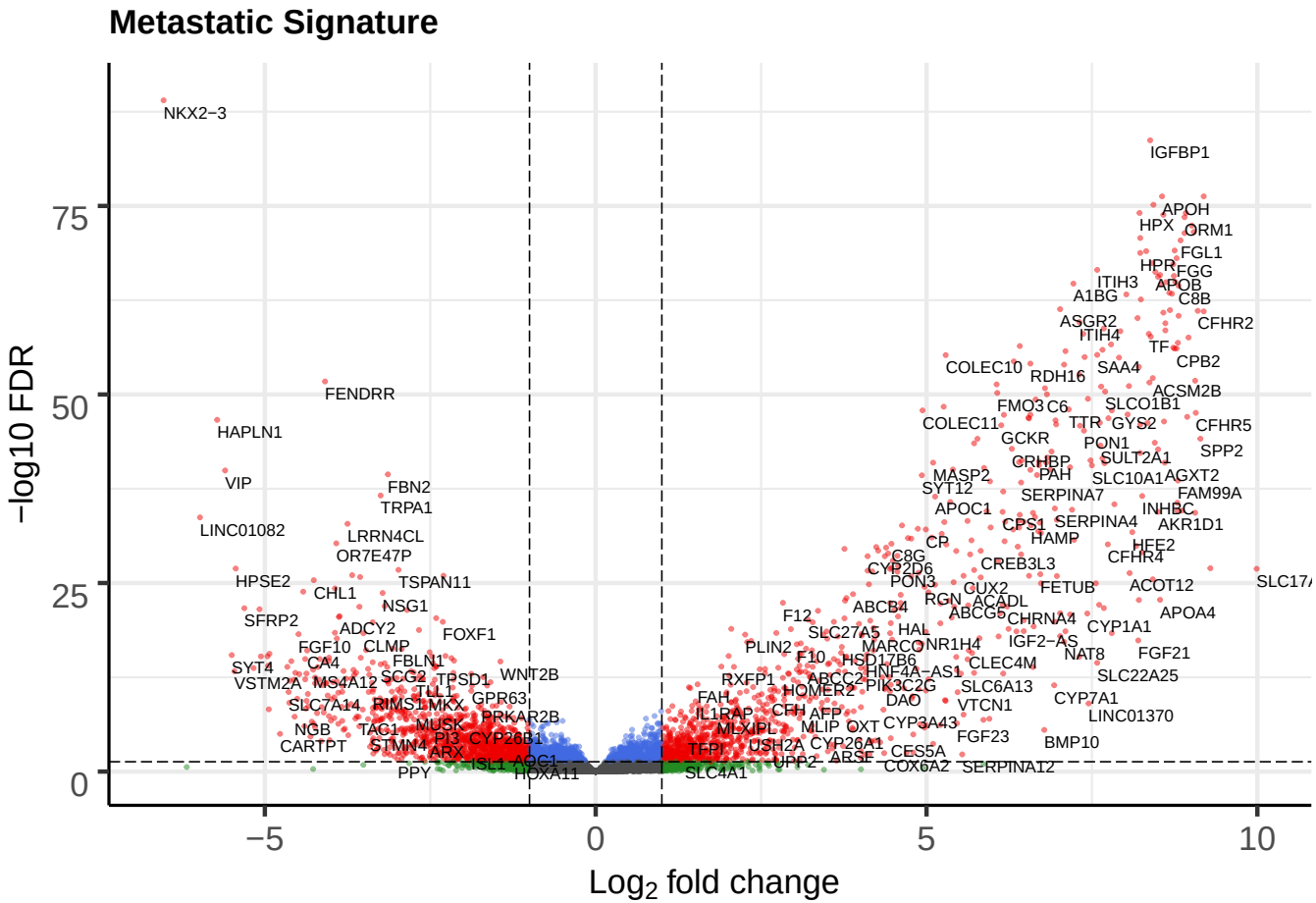
- Volcano plot is a good compromise
  - X-axis represents log2 fold change (biological significance)
  - Y-axis represents -log10 of p-value or FDR (statistical significance)

Genes in top right quadrant or top left quadrant are usually the most interesting to follow-up.

# Volcano plot

```
pacman::p_load(EnhancedVolcano)

ymax <- -log10(min(dge$FDR_metvspri)) + 0.5

EnhancedVolcano(dge,
                lab      = dge$symbol,
                x        = "LFC_metvspri",
                y        = "FDR_metvspri",
                FCcutoff = log2(2),
                pCutoff  = 0.05,
                title    = "Metastatic Signature",
                subtitle = NULL,
                caption  = NULL,
                ylab     = "-log10 FDR",
                ylim     = c(0, ymax),
                legendPosition = "none")
```

# Volcano plot




Warning: Not all significant genes are plotted unless you set `DrawConnectors=TRUE` (which can get messy for large number of significant hits). Alternatively, you can use the `selectLab` argument to highlight selected genes.

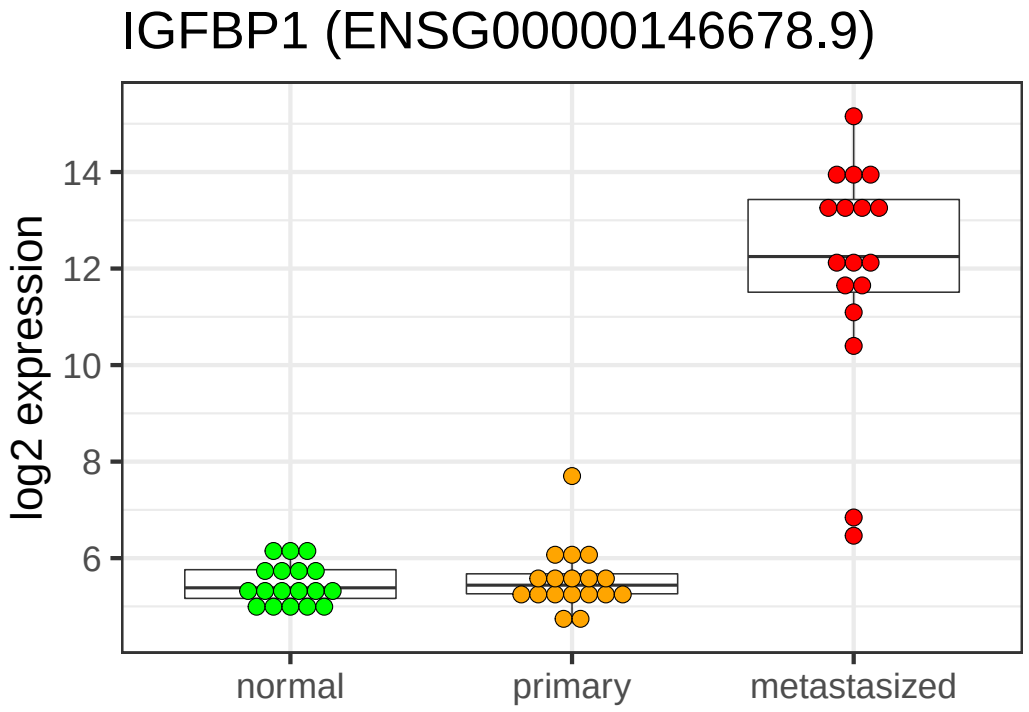
# Plotting a single gene | Part 1

To visualize the expression of top hit, we need to extract the gene and combine it with clinical data.

```
gene <- "IGFBP1"
ensg <- dge %>% filter(symbol==gene) %>% pull(gene_id)
ensg
## [1] "ENSG00000146678.9"

identical( rownames(pheno), colnames(expr) )
## [1] TRUE
tmp <- data.frame( pheno, gene=expr[ensg, ] ) # Extract
head(tmp) %>% select(-geo_accession)
##             sample_id    type subject     gene
## SRR975551 SRR975551 primary   AMC_2 5.098317
## SRR975552 SRR975552 primary   AMC_3 5.339850
## SRR975554 SRR975554 primary   AMC_6 5.481692
## SRR975553 SRR975553 primary   AMC_5 5.290842
## SRR975555 SRR975555 primary   AMC_7 5.285094
## SRR975556 SRR975556 primary   AMC_8 5.596106
```

# Plotting a single gene | Part 2

```
ggplot(tmp, aes(x=type, y=gene, fill=type)) +
  geom_boxplot(outlier.shape=NA, fill="white") +
  geom_dotplot(binaxis="y", stackdir="center") +
  labs(x=NULL, y="log2 expression",
       title=paste0(gene, " (", ensg, ")")) +
  theme_bw(base_size=30) + theme(legend.position="none") +
  scale_fill_manual(values=c("green", "orange", "red"))
```



# Plotting a single gene | Generalizing to a function

If we want to plot another gene:

- we will have to repeat all the codes in the previous two slides.
- if we want make any changes to the template (e.g. point colors), we will have to find and replace all the changes.

A better way is to generalize the codes into a **function**. Look up the `plotGenes()` function in the finalized version of `GSE50760_CRC_classroom.R`.

# Number of significant genes

How many genes are (save output as separate objects)

1. significantly upregulated for metastasis
2. significantly downregulated for metastasis
3. significantly upregulated for tumorgenesis
4. significantly downregulated for tumorgenesis

# Number of significant genes

How many genes are (save output as separate objects)

1. significantly upregulated for metastasis
2. significantly downregulated for metastasis
3. significantly upregulated for tumorgenesis
4. significantly downregulated for tumorgenesis

```
met_up   <- dge %>% filter(LFC_metvspri > 1,
                           FDR_metvspri < 0.05)   # 1062

met_down <- dge %>% filter(LFC_metvspri < -1,
                           FDR_metvspri < 0.05)   # 1195

tum_up   <- dge %>% filter(LFC_privsnorm > 1,
                           FDR_privsnorm < 0.05) # 1181

tum_down <- dge %>% filter(LFC_privsnorm < -1,
                           FDR_privsnorm < 0.05) # 1844
# then use nrow(). E.g. nrow(met_up)
```

# Number of significant genes | Overlap

How many genes are significantly up-regulated for both the tumorigenesis and metastasis signatures?

# Number of significant genes | Overlap

How many genes are significantly up-regulated for both the tumorigenesis and metastasis signatures?

Simplest is to use `intersect()`

```
intersect( met_up$gene_id, tum_up$gene_id ) %>% length
## [1] 144
intersect( met_down$gene_id, tum_down$gene_id ) %>% length
## [1] 631
```

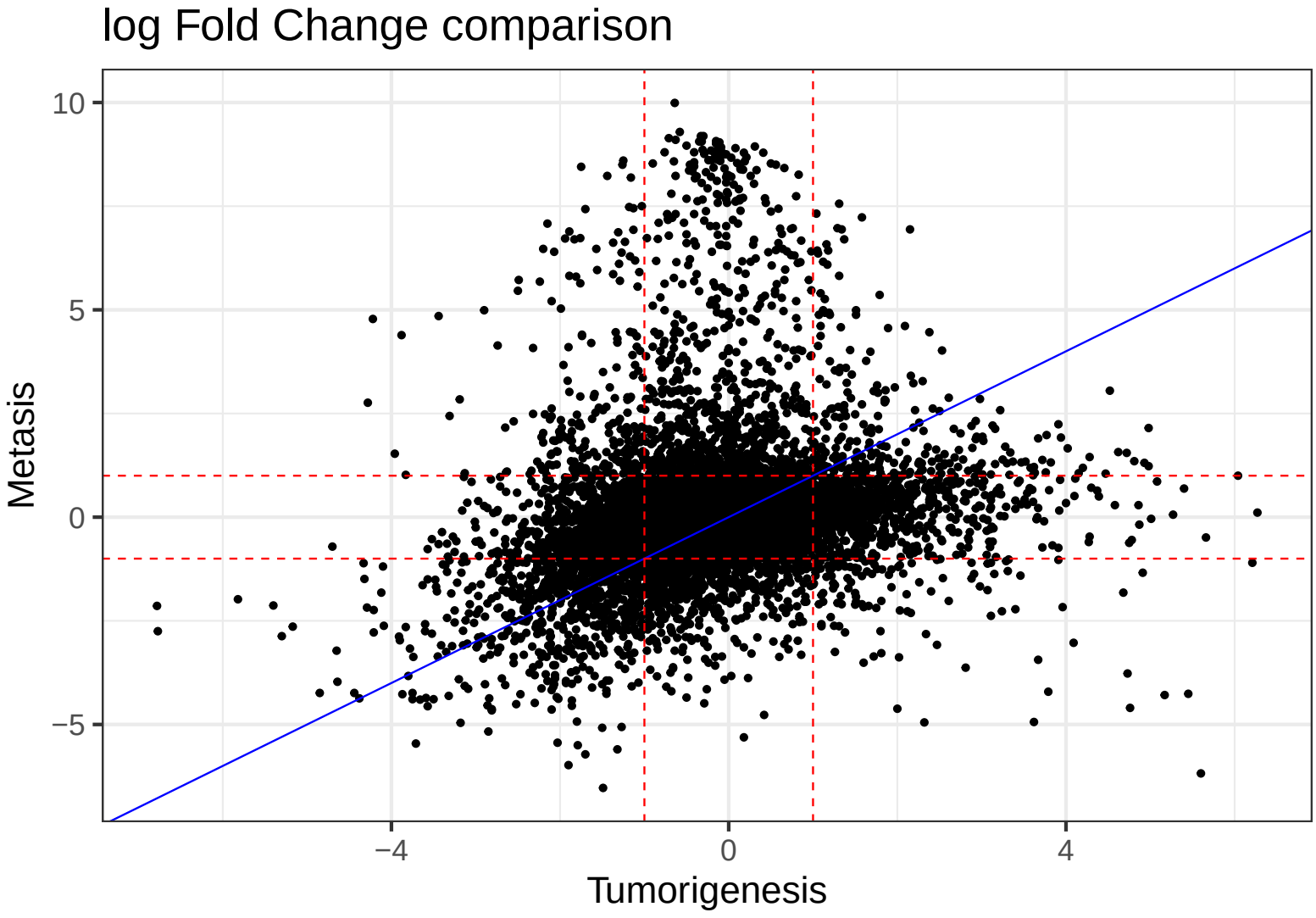
*or* you can draw Venn diagrams (see Session 3 notes).

However, this does not allow us to easily:

- see genes that have opposite effects
- quantify the magnitude of the biological differences
- have a general idea of the similarity between the two processes

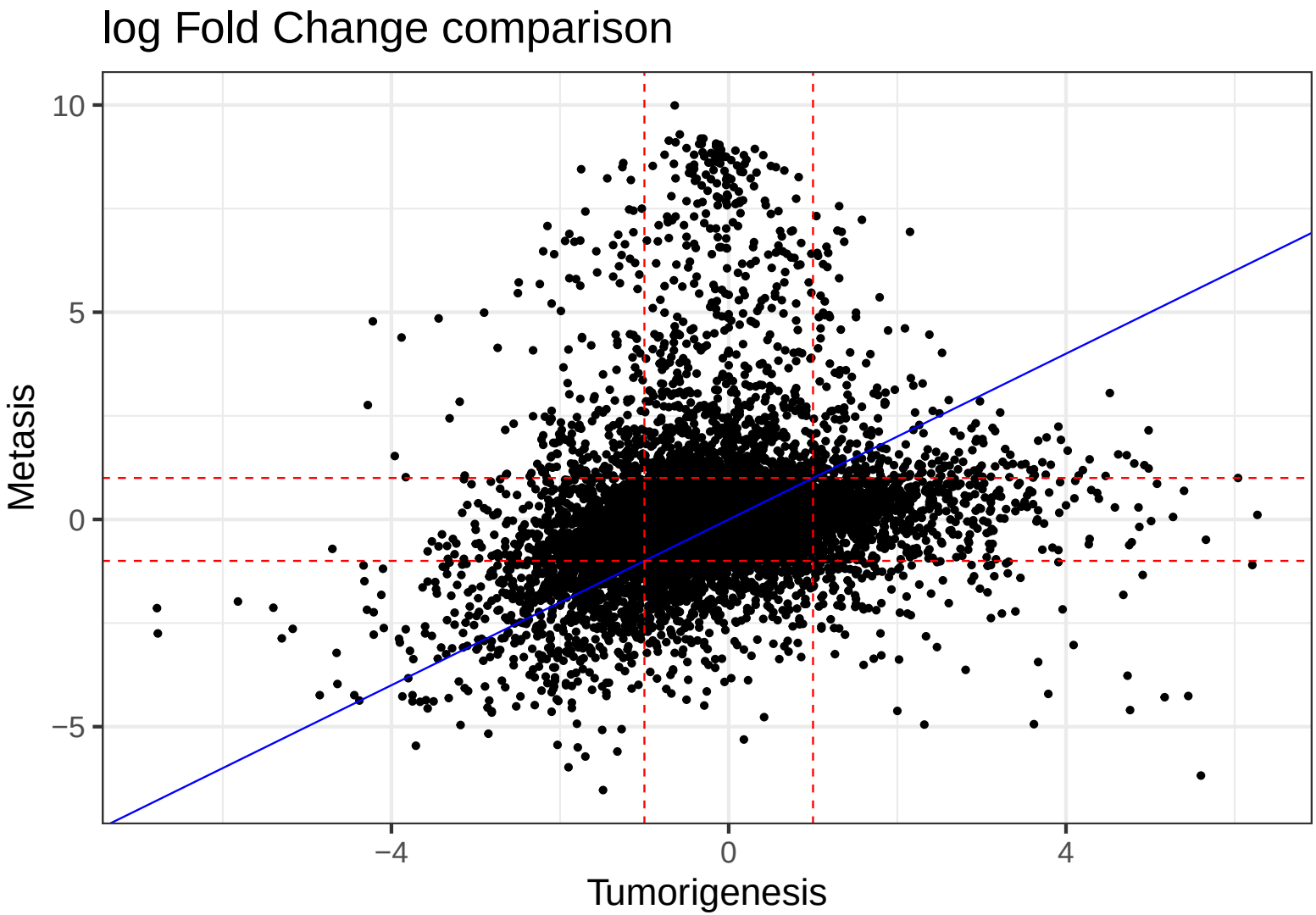
# Number of significant genes | Beta-Beta plot

What does the following graph show?



# Number of significant genes | Beta-Beta plot

What does the following graph show?




Reproduce it. `geom_abline()`, `geom_hline()`, `geom_vline()`

# Number of significant genes | Beta-beta plot solution

```
ggplot(dge, aes(x=LFC_privsnorm, y=LFC_metvspri)) +

  geom_point() +

  geom_abline(intercept=0, slope=1, col="blue") +

  geom_vline(xintercept=c(-1, 1), lty=2, col="red") +

  geom_hline(yintercept=c(-1, 1), lty=2, col="red") +

  theme_bw() +

  labs(x="Tumorigesis", y="Metagenesis",
       title="log Fold Change comparison")
```

# Prepare for heatmap | Subset results

1. Sort each list by the p-value
2. Select the top 10 genes from each list (for illustration).
3. Row bind them.

# Prepare for heatmap | Subset results

1. Sort each list by the p-value
2. Select the top 10 genes from each list (for illustration).
3. Row bind them.

```
d1 <- met_up   %>% arrange(P_metvspri) %>% head(10)
d2 <- met_down %>% arrange(P_metvspri) %>% head(10)
d3 <- tum_up   %>% arrange(P_privsnorm) %>% head(10)
d4 <- tum_down %>% arrange(P_privsnorm) %>% head(10)

goi <- bind_rows(d1, d2, d3, d4)
rm(d1, d2, d3, d4, met_up, met_down, tum_up, tum_down)

## Optional - if you want to annotate rows later
tmp <- c("met_up", "met_down", "tum_up", "tum_down")
goi$dge_list <- rep(tmp, each=10)
rm(tmp)
```

# Prepare for heatmap | Subset expression

Next, we need to extract the VST expression for these 40 genes.

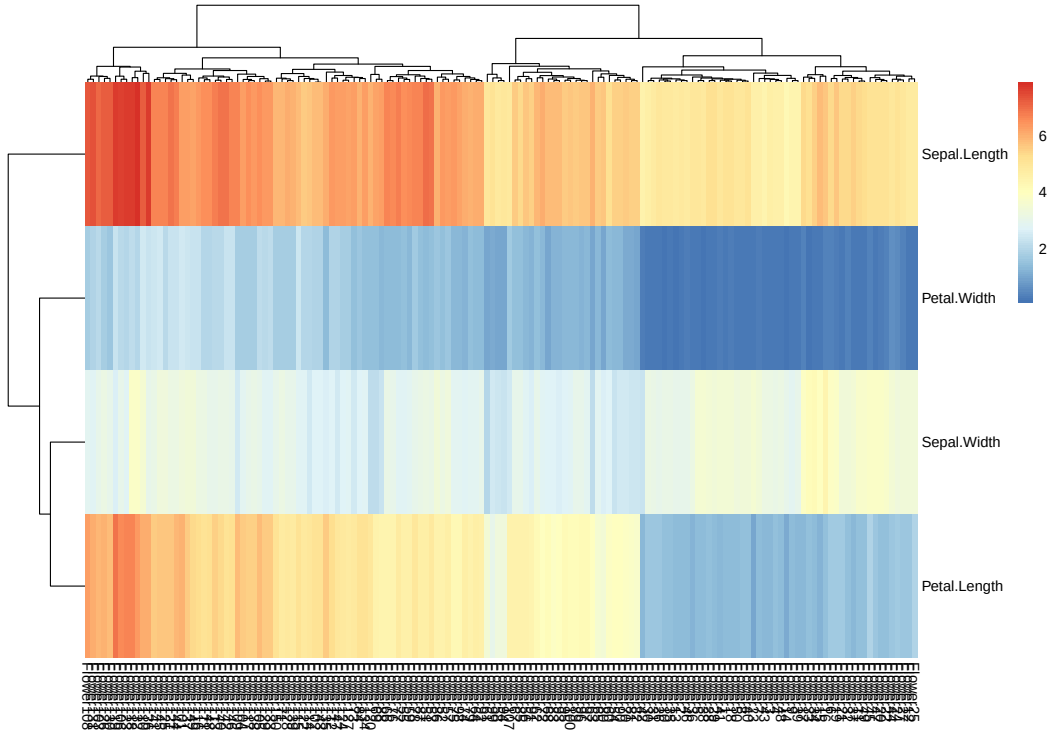
The unique identifier is the gene_id column (OK, strictly speaking the ENSGID). Since the rownames of VST expression data are also based on gene_id, we can subset the matrix as follows:

```
hm <- expr[ goi$gene_id, ]  ## Extract

identical( rownames(hm), goi$gene_id )
## [1] TRUE

identical( colnames(hm), pheno$sample_id )
## [1] TRUE
```

It is not necessary, but prudent to check the `hm` matrix is aligned to the `goi` and the `pheno` dataframes, as we will be combining information from these dataframes next.

# Heatmap | Iris - the ugly version

```
rownames(iris) <- paste0("Flower", 1:150) # Patch
iris_num <- iris %>% select(-Species) %>% t()
dim(iris_num)
## [1]   4 150
pheatmap(iris_num)    # from pheatmap package
```



# Heatmap | Iris - the better version

Objectives:

1. Add information about the Species in the columns.
2. Scale the rows (mean zero and unit variance). After scaling, the row (e.g. gene) will be comparable to each other.
3. Use correlation metric as distance measure (default is Eucledian) which is prefered measure[1] in transcriptomics as we are interested in the similarity of the shapes.
4. Remove column labels as they are not meaningful here.

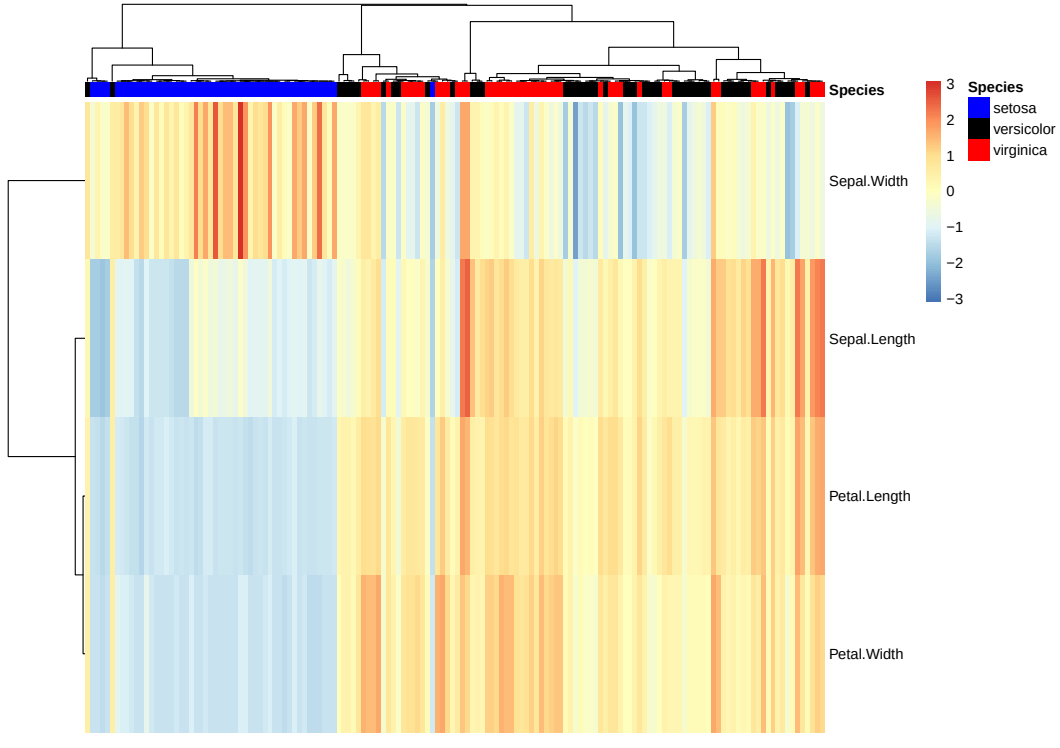
[1] See http://www.opiniomics.org/you-probably-dont-understand-heatmaps/

# Heatmap | Iris - the better version

```
ann_col <- iris %>% select(Species)
head(ann_col)
##         Species
## Flower1  setosa
## Flower2  setosa
## Flower3  setosa
## Flower4  setosa
## Flower5  setosa
## Flower6  setosa

ann_colors = list( Species = c(setosa="blue",
                               versicolor="black",
                               virginica="red") )
ann_colors
## $Species
##     setosa versicolor  virginica
##     "blue"    "black"      "red"
```

# Heatmap | Iris - the better version

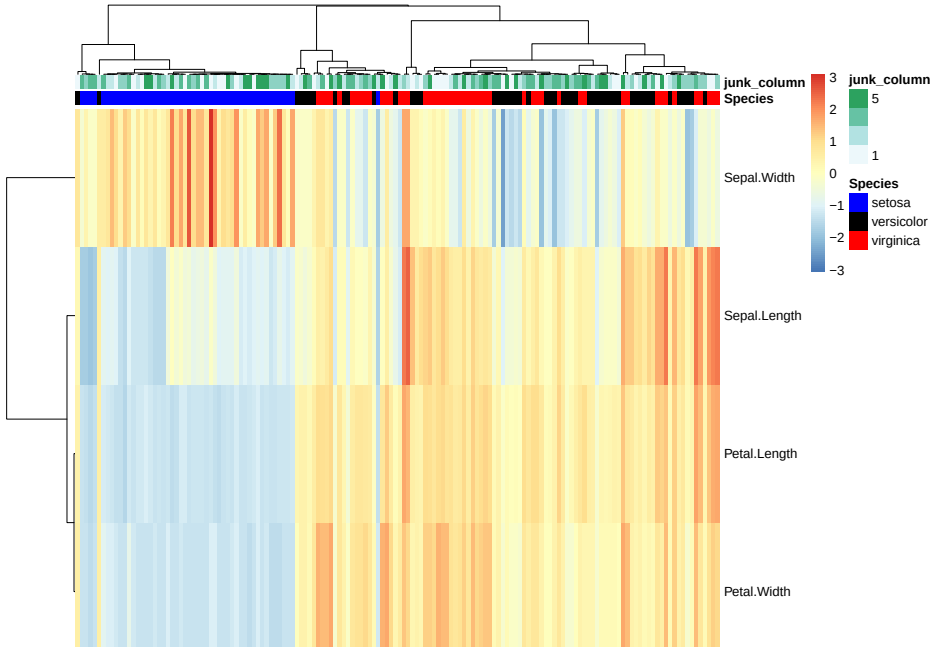
```
pheatmap(iris_num,
         scale                = "row",
         annotation_col       = ann_col,
         annotation_colors    = ann_colors,
         show_colnames        = FALSE,
         clustering_distance_rows = "correlation",
         clustering_distance_cols = "correlation")
```



# Heatmap | Iris - the better version

You can do multiple column/rows annotation. E.g.:

```
ann_col$junk_column <- rep(1:5, 30)

pheatmap(iris_num, scale = "row", show_colnames = F,
         annotation_col    = ann_col,
         annotation_colors = ann_colors,
         clustering_distance_rows = "correlation",
         clustering_distance_cols = "correlation")
```

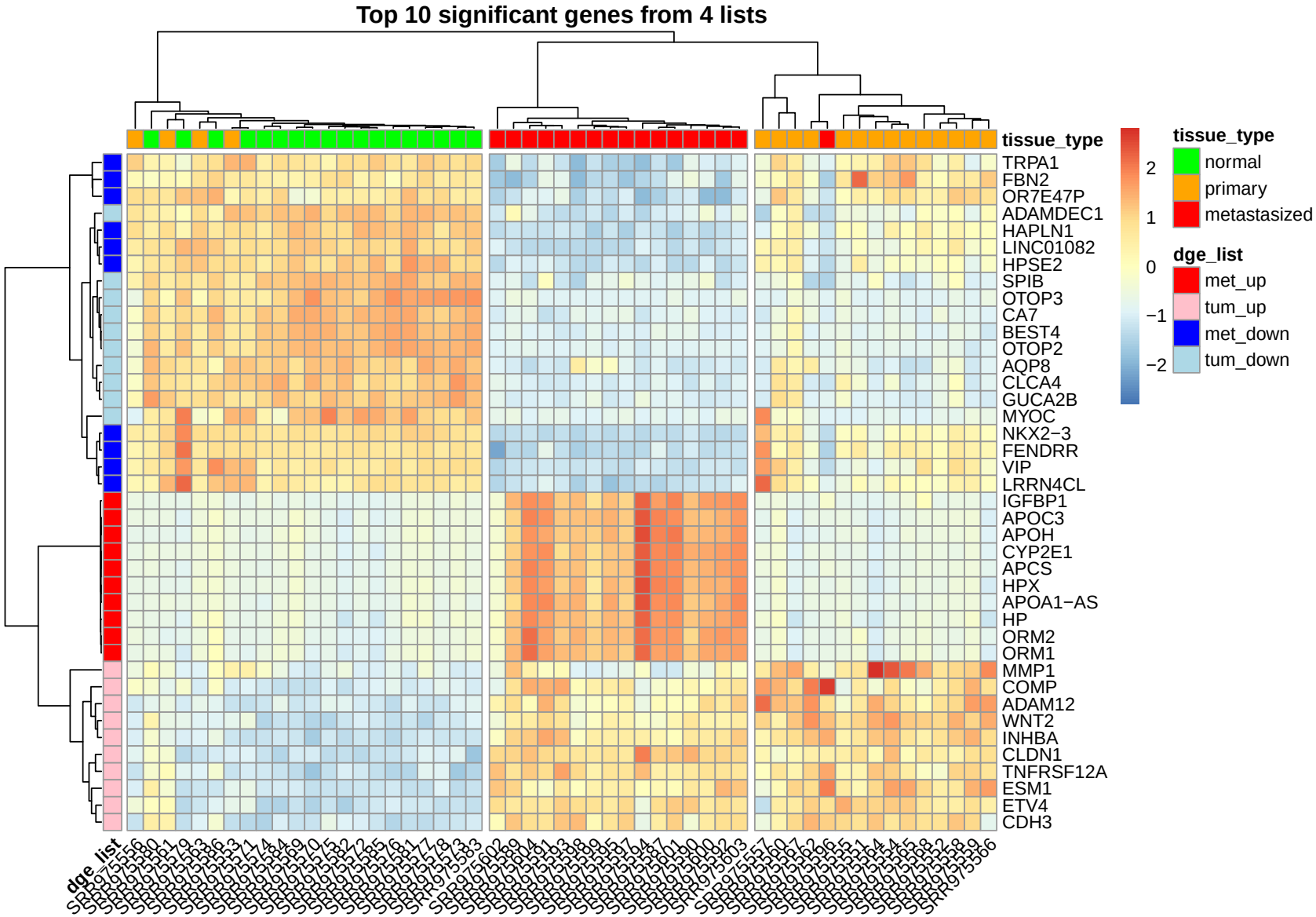


# Heatmap | Classroom exercise on 40 genes

Draw a heatmap for the 40 genes we selected earlier:

1. Annotate the samples/columns with the tissue type (normal, primary, metastasis).
2. Ensure row and column names are informative. Otherwise change or remove it.
3. Scale the genes.
4. Use correlation metric as distance metric.

# Heatmap | Classroom exercise on 40 genes (output)



# Heatmap | Classroom exercise on 40 genes (minimal)

```
ann_col <- pheno %>% select(tissue_type=type)

pheatmap(hm,
         scale                = "row",
         annotation_col       = ann_col,
         labels_row           = goi$symbol, ## Replace ENSGID
         clustering_distance_rows = "correlation",
         clustering_distance_cols = "correlation",
         main = "Top 10 significant genes from 4 lists")
```

# Heatmap | Classroom exercise on 40 genes (enhanced pt 1)

```
ann_col <- pheno %>% select(tissue_type=type)

ann_row <- goi %>%
  column_to_rownames("gene_id") %>%
  select(dge_list)

ann_colors = list(
  tissue_type = c(normal="green",
                  primary="orange",
                  metastasized="red"),

  dge_list = c(met_up="red", tum_up="pink",
               met_down="blue", tum_down="lightblue")
)
```

# Heatmap | Classroom exercise on 40 genes (enhanced pt 2)

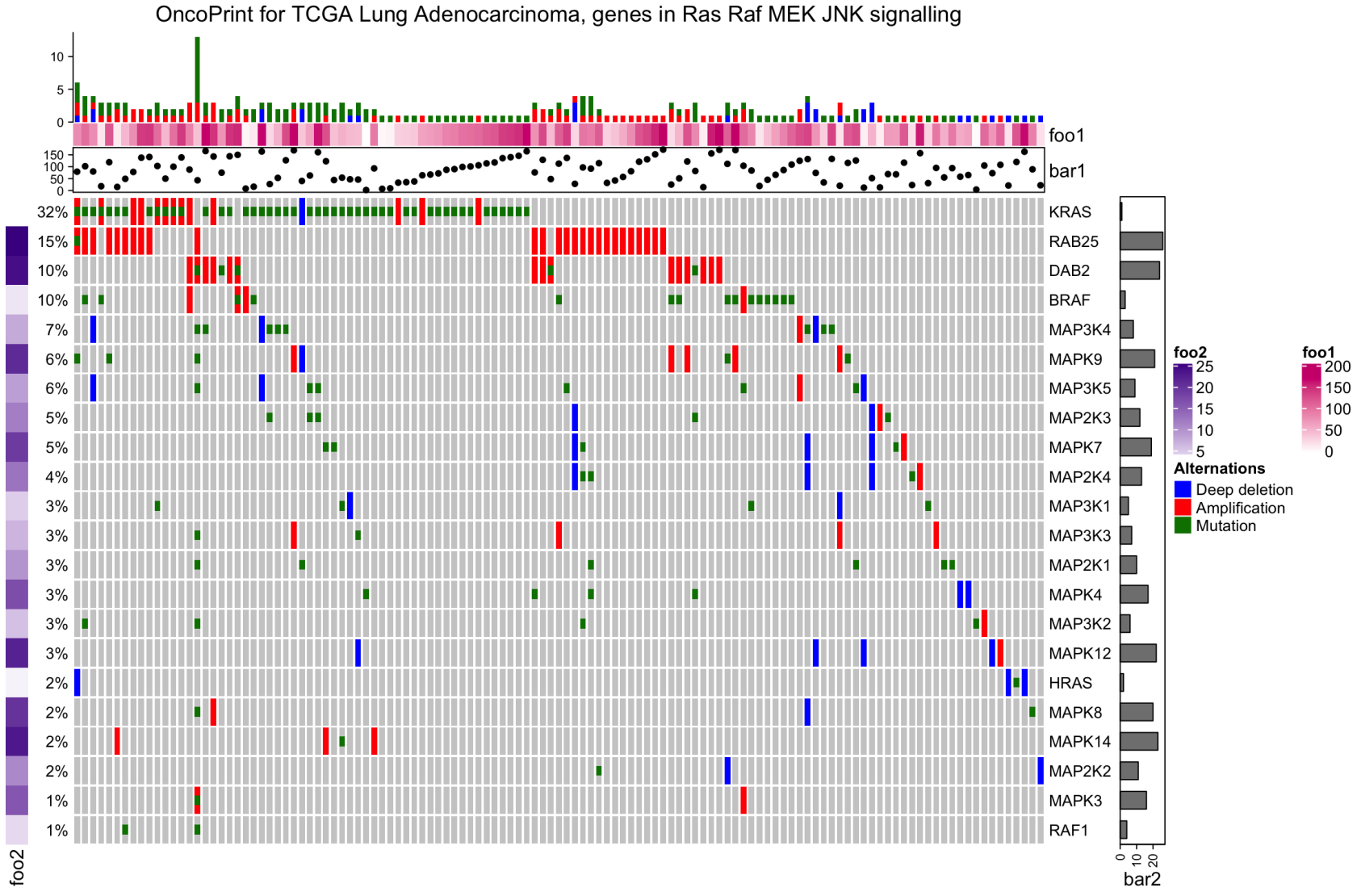
```
pheatmap(hm,
         scale                = "row",
         annotation_col       = ann_col,
         annotation_row       = ann_row,
         labels_row           = goi$symbol,
         annotation_colors    = ann_colors,
         cutree_cols          = 3,
         angle_col            = 45,
         clustering_distance_rows = "correlation",
         clustering_distance_cols = "correlation",
         main = "Top 10 significant genes from 4 lists")
```

# ComplexHeatmap | Alternative to pheatmap

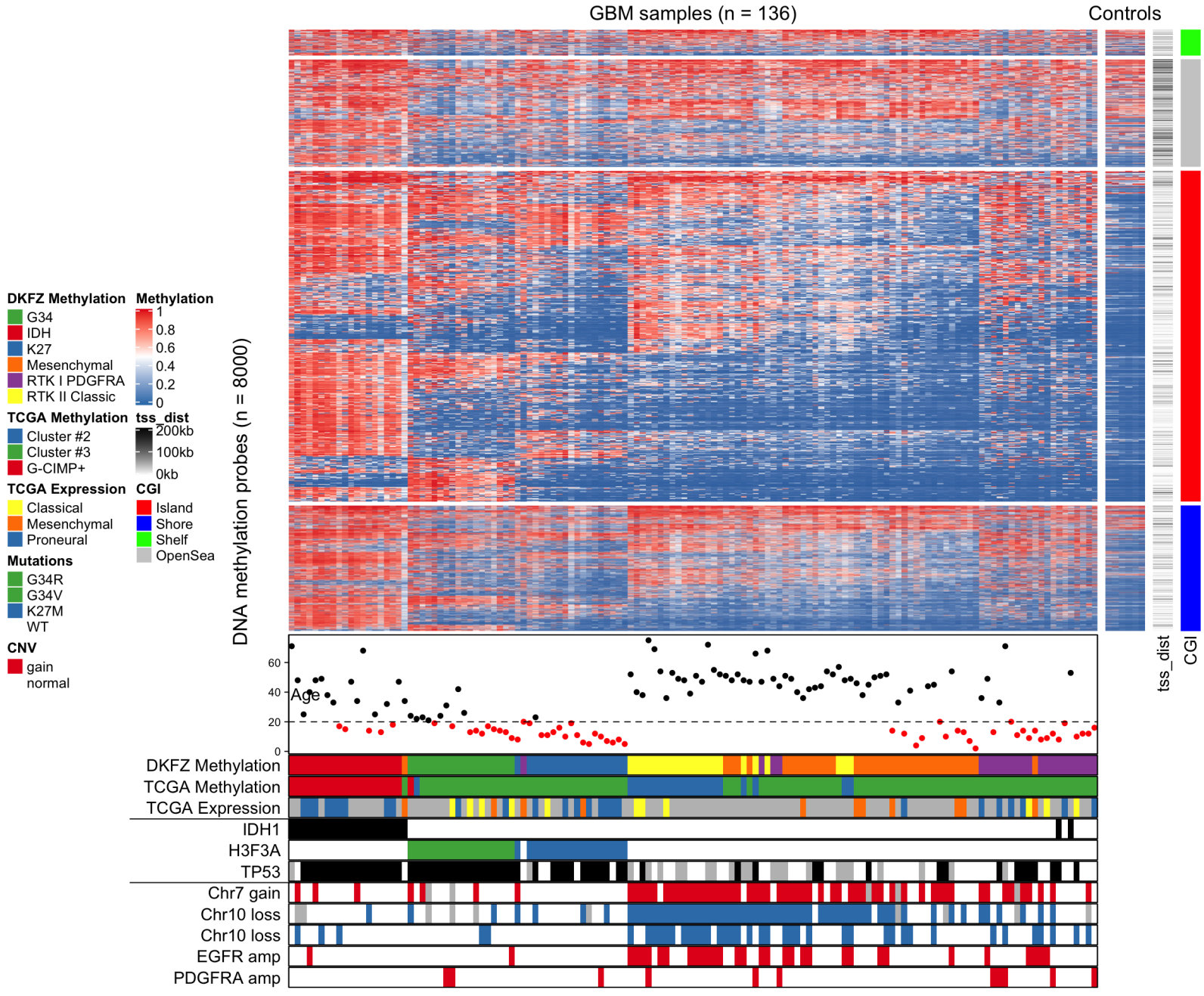
The Bioconductor package `ComplexHeatmap` allows you to make even more sophisticated heatmaps. For example, you can use this package to display multiple heatmaps (e.g. from different lists) in one display.

Please see the following sites for example:
https://jokergoo.github.io/ComplexHeatmap-reference/book/

# ComplexHeatmap | Oncoprint



# ComplexHeatmap | Methylation with multiple annotation



# ComplexHeatmap | Measles vaccine effect