Data_Visualization_for_RNA-seq_of_Xenopus_Brain_Development

2023-05-06

1. Load packages

Install and load the necessary packages (e.g., tidyverse, DESeq2, EnhancedVolcano).

```
if (!require("pacman")) install.packages("pacman")     # Install pacman package if not already

# Use p_load() function to install packages that aren't already and load all packages
pacman::p_load(tidyverse, reader, ggrepel, DESeq2, apeglm, EnhancedVolcano, pheatmap, grid, gridExtra)
```

2. Import data sets

Load the frog brain development raw gene count and sample metadata. The data sets are provided by Dr. Rebecca L. Young.

```
# Import raw read counts
raw_counts <- read.table(file = "/stor/work/Bio321G_RY_Spring2023/Exercises/frog_brain_developmentalExp
                         row.names = 1,    # Assign 1st column as row names
                                           # Row names shouldn't contain duplicates
                         header = TRUE,    # Use 1st row of the data frame as column headers
                         sep = ",")        # Separate all data between columns by the ","


# Import gene lengths
gene_lengths <- read.table(file = "/stor/work/Bio321G_RY_Spring2023/Exercises/frog_brain_developmentalE
                         header = TRUE,    # Use 1st row of the data frame as column headers
                         sep = ",")


# Import metadata
metadata <- read.table(file = "/stor/work/Bio321G_RY_Spring2023/Exercises/frog_brain_developmentalExp/f
                       header = TRUE,
                       sep = ",")
```

3. Data Transformation

Typically, RNA sequencing studies aim to compare gene expression within or across samples. For example, studies may aim to quantify

1. The relative expression of genes within an individual

2. How gene expression differs between samples (e.g., experimental treatments, across stages of development, or populations or species).

Normalization is required to account for factors that prevent these within and across species comparisons.

The main factors often considered during normalization are:

1. *Gene length*: Accounting for gene length is necessary for **comparing expression between different genes within the same sample**. In the example below, what is the relative expression of these three genes in Sample 1? Gene A has the highest read count at 24 reads; however, it is twice as long as Gene C. Why would that matter? In principle, reads can map to any fragment of the gene, Gene A has twice as many targets for sequencing. Thus, to compare across genes, reads must be normalized by length.

The relative expression of these genes is: A = C > B

2. *Sequencing depth*: Accounting for sequencing depth is necessary for **comparison of gene expression between samples**. In the example below, let's compare expression of Gene A between our two samples. Samples 1 and 2 have 24 and 50 reads of Gene A, respectively. We might think the expression of Gene A in Sample 2 is more than double that of Sample 1. However, if we taken into account of sequencing depth, Gene A in Sample 1 has 24/48(total reads) and in Sample 2 has 50/96. Therefore, Sample 2 exhibits only slightly higher expression of Gene A.

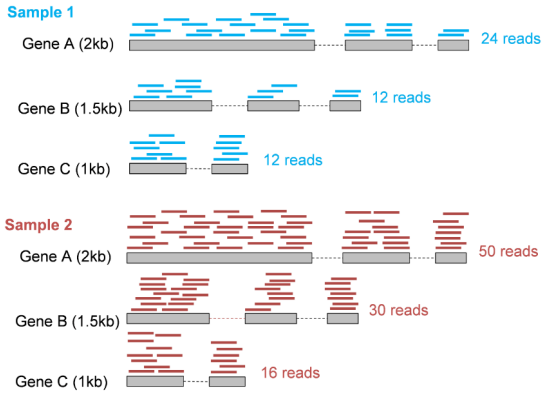


Figure 1. Genes are made up of exons (grey bars) and introns (dashed lines). Exons are coding regions that will be transcribed and translated. Introns are non-coding regions that are part of the DNA and can play a role in regulating expression, among other things but are not part of the final gene product. Sequenced fragments of cDNA (i.e., reads are indicated as lines, blue or red). Reads are aligned to genes and expression is quantified using the number of aligned reads.

A normalization technique called TPM (transcripts per kilobase million) is suitable for both (1) gene count comparisons within a sample or (2) between samples of the same sample group, but NOT for differential gene expression (DGE) analysis

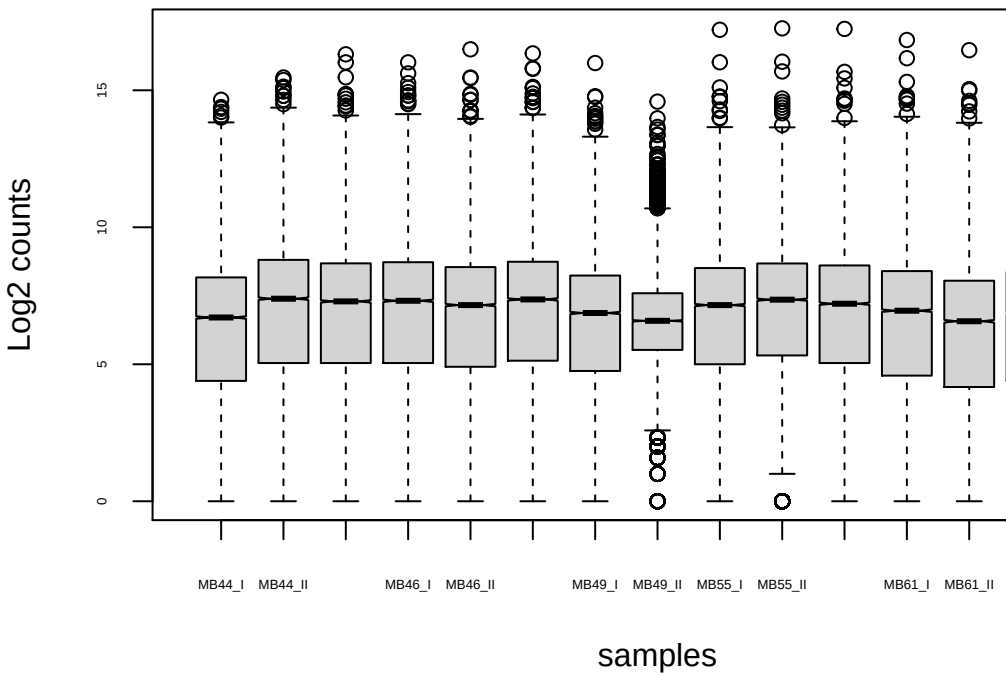
```
# TPM calculation function
TPM <- function(counts, lengths){
  reads_per_kilobase <- counts/(lengths/1000)    # Divide length by 1000 to convert to kilobase
  per_million <- sum(reads_per_kilobase, na.rm = TRUE)/1e6  # Total read in sample and divide by 10^6 (
  tpms <- reads_per_kilobase/per_million    # Total reads in a million base (that contain other bases as
                                            # Last expression, which is the tpms, is the return value
}
```

```r
# Loop the function to calculate TPMs across all samples
tpms <- raw_counts
for (i in 1:ncol(tpms)){
  tpms[,i] <- TPM(tpms[,i], gene_lengths$length)
}

# Filtering genes with zero counts
tpms<- tpms %>%
  mutate(total_count = rowSums(across(where(is.numeric)))) %>%
  filter(total_count != 0) %>%    # the != indicates that we want to keep rows that do not equal zero
  dplyr::select(-total_count)    # now that we have filtered we no longer need this column

# Keep another copy of raw_counts without genes that have 0 counts for later plotting
raw_counts <- raw_counts[rownames(raw_counts) %in% rownames(tpms), ]

# Log transform the tpms
log_tpms<- log(tpms)         # New dataframe with log transformed tpms
log_tpms[!is.finite(as.matrix(log_tpms))] <- 0     # log(0) = -inf, so replace -inf values with 0


# Group the samples into early, middle, and late groups
metadata <- metadata %>%
  mutate(grouped_stage = case_when(stage == "Stage44" ~ 'early',
                                   stage == "Stage46" ~ 'early',
                                   stage == "Stage49" ~ 'middle',
                                   stage == "Stage55" ~ 'middle',
                                   stage == "Stage61" ~ 'late',
                                   stage == "Stage66" ~ 'late'))


# Specify factors level so that plotting & legend would follow the order
# early -> middle -> late. Without this, the plots & legends will be in
# alphabetical order (i.e., early -> late -> middle)
metadata$grouped_stage <- factor(metadata$grouped_stage, levels = c("early", "middle", "late"))
```

4. Visualization of gene counts

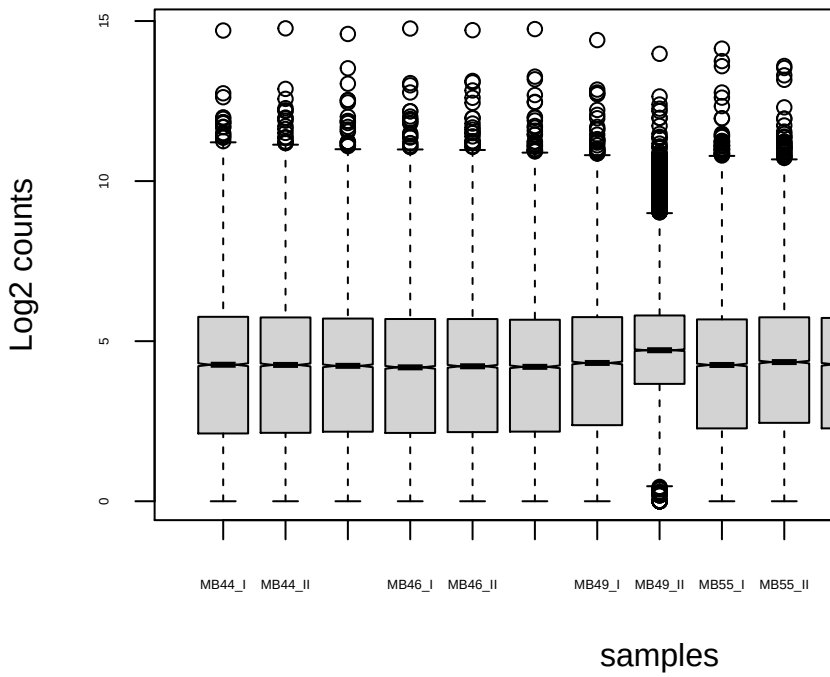
To further illustrate the need for normalization, we can plot the distribution of the counts. *Note*: for visualiztaion purposes, we are log2 transforming our data. Because log of 0 is undefined, we add 1 to all entries.

```r
par(cex.axis = 0.45)  # Reduce y-axis label size to include all sample names
boxplot(log(raw_counts + 1, 2),
        xlab = "samples",
        ylab = "Log2 counts",
        notch = TRUE)
```



4.a. Distribution of raw counts

```
par(cex.axis = 0.45)
boxplot(log(tpms + 1, 2),
        xlab = "samples",
        ylab = "Log2 counts",
        notch = TRUE)
```



4.b. Distribution of TPM normalized counts

By comparing the two boxplots, we can see that TPM calculation worked well to normalize the expression patterns of all genes across samples as the distribution of the counts are more identical, allowing us to compare the genes across samples better. One sample - `MB49_II` - looks a little different than the rest. For now we will keep this sample in our data set. However, if we find this sample has anomalous patterns in downstream analyses we can revisit this.

5. Principal Component Analysis (PCA)

```
x <- log_tpms %>%
  t()   # Transpose matrix

# Calculate the principal components of the data set
PC_x <- prcomp(x)

# Create a new data frame with the sample_id, principal components, and metadata
PCs_x <- data.frame(PC_x$x) %>%
  rownames_to_column(var = "sample_id")   # make sample IDs a column to facilitate adding other metadat

PCs_x <- left_join(PCs_x, metadata)
```

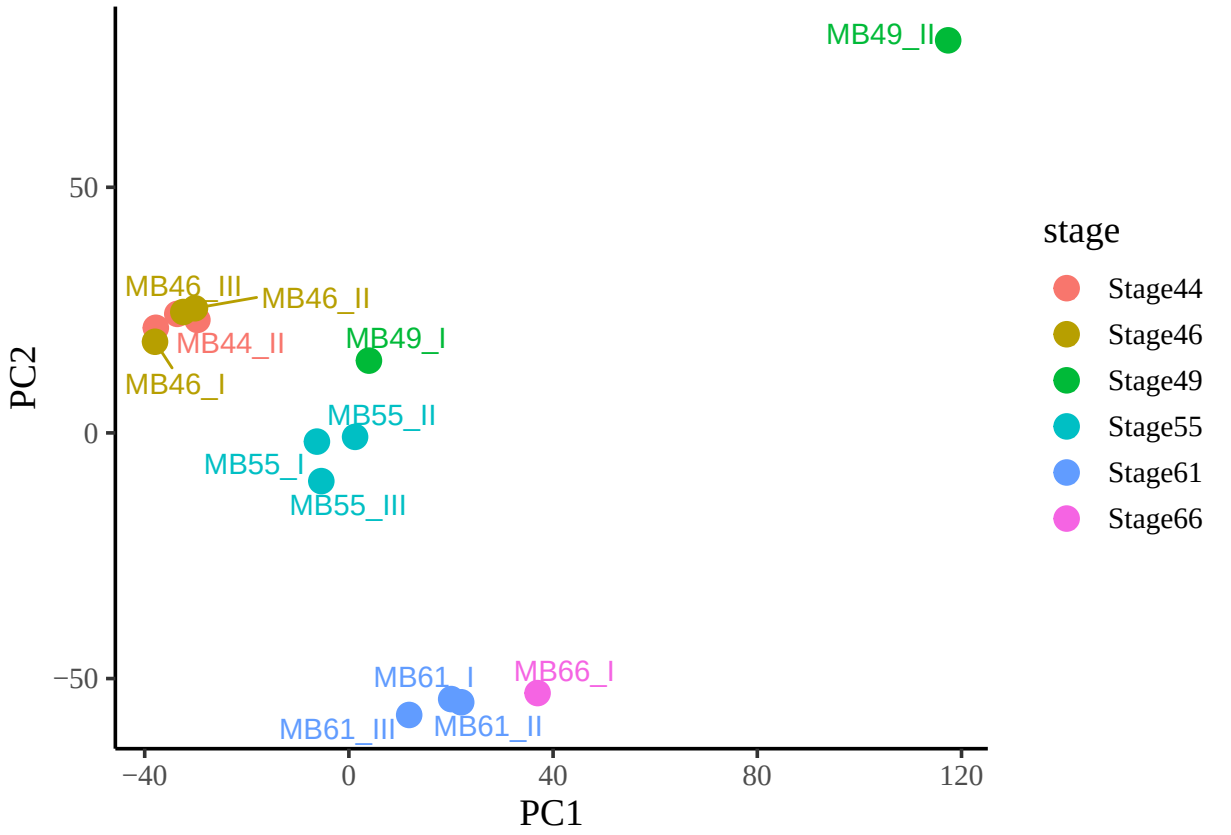
```
head(PCs_x, 10)
```

5.a. Calculate the PCs

```
##    sample_id        PC1       PC2        PC3       PC4         PC5         PC6
## 1     MB44_I -29.673210 22.998278 -17.401158  3.498801  -7.6310878   1.4814511
## 2    MB44_II -33.615948 24.225287 -17.152881  5.672460  -4.1228485   1.4878835
## 3   MB44_III -37.817073 21.394354  -7.997593  5.577685  -0.6764211  -6.4608610
## 4     MB46_I -37.976063 18.553980 -10.381690  2.542004   4.1105667  -4.2677649
## 5    MB46_II -30.155726 25.372120 -12.307397 -2.788115   7.2490186   7.2175608
## 6   MB46_III -32.438727 24.610327 -10.414909 -1.131051  10.3702785   0.1658562
## 7     MB49_I   3.928878 14.712348  21.637595 -2.821209 -29.8367165 -21.9992604
## 8    MB49_II 117.387277 79.910030  -9.905556 -5.436476   1.0581517   2.7460330
## 9     MB55_I  -6.260616 -1.772671  34.598315 -7.788663  26.3207722  28.8952724
## 10   MB55_II   1.213102 -0.808470  44.393692 10.218132  11.6363941 -21.1703902
##             PC7        PC8         PC9       PC10       PC11        PC12
## 1  -18.56553753  12.757252 -11.9210650  10.642417 -21.022481   4.4181959
## 2   -9.65588630  12.317518 -10.2238010  11.945598  -7.311397   0.8668859
## 3    0.71879802   9.087344  -0.6669072   2.126935  12.152186  -7.2092497
## 4    7.86097823 -13.177793   9.1000146 -39.559038 -20.633376   5.5270585
## 5    9.52730412  -6.840796   8.1333121   2.040742  14.394853 -36.5253175
## 6    9.29075330  -6.416654   9.0426293   7.491535  26.509403  32.4510917
## 7    7.93732776 -32.329410  -1.6497216  17.088530  -7.531112  -0.8097155
## 8   -0.02846744   4.578336  -1.3571811  -5.769996   1.635377   0.7761624
## 9    7.11678152 -11.769142 -26.0624588   4.275581  -7.195575   1.6415336
## 10  14.35367741  29.643048  14.0013210   2.370322  -6.742439  -1.0279884
##           PC13         PC14          PC15   tissue   stage grouped_stage
## 1   -9.6307366 -28.02977794  7.859899e-14 midbrain Stage44         early
## 2   -5.3885124  36.75137255  9.159021e-14 midbrain Stage44         early
## 3   41.1028462  -5.31672580  8.836679e-14 midbrain Stage44         early
## 4    0.8143724   3.03417575 -8.619949e-16 midbrain Stage46         early
## 5  -17.5038402  -4.70839672  8.338566e-14 midbrain Stage46         early
## 6  -10.2138972  -4.11826284  7.078478e-14 midbrain Stage46         early
## 7    1.3004209   0.99327767  3.400882e-14 midbrain Stage49        middle
## 8    1.9325114   0.37397006 -4.670297e-13 midbrain Stage49        middle
## 9    5.4771130  -0.09987919  5.462224e-15 midbrain Stage55        middle
## 10  -6.9628095  -0.53070502 -1.527532e-13 midbrain Stage55        middle
```

5.b. Anomaly Detection We'll quickly plot a PCA (of PC1 and PC2) to determine if there's any particular anomaly

```
ggplot(data = PCs_x, aes(x = PC1, y = PC2, color = stage, label = sample_id)) +
  geom_point(size = 4) +
  geom_text_repel() +
  theme_classic(base_family = "Times",
                base_size = 14)
```



This confirm the visualization in section 4 where the anomalous sample is `MB49_II`, indeed. Let's remove it and recalculate the PCs

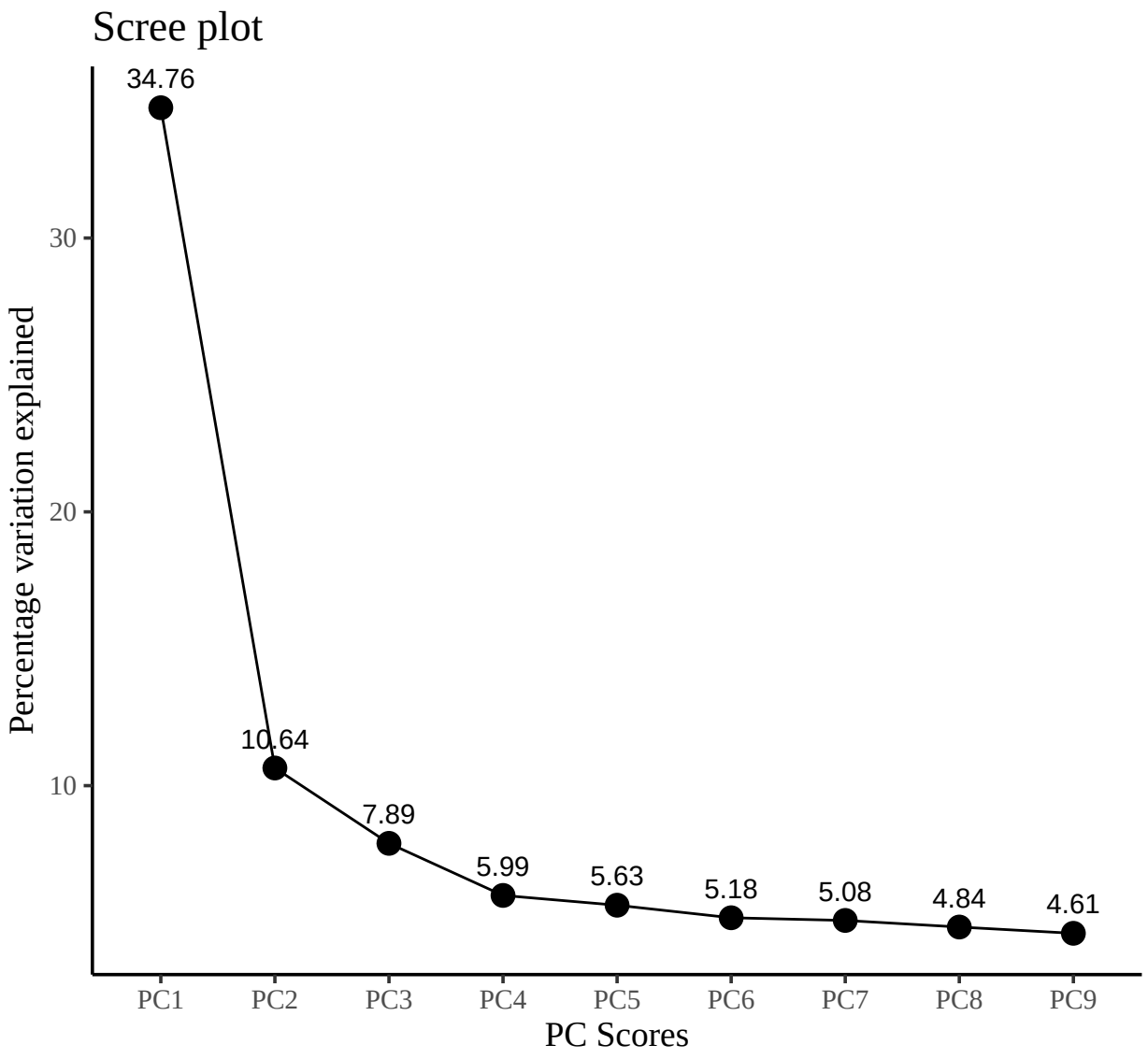
```
# Remove anomaly
log_tpms_no_outlier <- log_tpms %>%
  dplyr::select(-MB49_II)

x_no_outlier <- log_tpms_no_outlier %>%
  t()    # Transpose matrix

# Calculate the principal components of the data set
PC_x_no_outlier <- prcomp(x_no_outlier)

# Create a new data frame with the sample_id, principal components, and metadata
PCs_x_no_outlier <- data.frame(PC_x_no_outlier$x) %>%
  rownames_to_column(var = "sample_id")    # make sample IDs a column to facilitate adding other metadat

PCs_x_no_outlier <- left_join(PCs_x_no_outlier, metadata)

head(PCs_x_no_outlier, 10)
```

```
##    sample_id        PC1       PC2        PC3        PC4        PC5       PC6
## 1     MB44_I -35.767145  16.74378 -5.9716566  7.7878985 -1.8847751  4.505128
## 2    MB44_II -39.072277  17.31048 -7.2322041  3.7782435 -2.9676075  5.868687
```

```
## 3   MB44_III -39.229279  10.76389 -4.0441586  -0.5681445  -0.2862720  14.210263
## 4     MB46_I -37.030706  13.96738 -0.9462110  -5.2281534  -0.2348997   4.591367
## 5    MB46_II -37.967092  12.03939  0.5779821  -6.2963020  -0.5039350 -13.547065
## 6   MB46_III -38.684218  10.97616 -0.1019236  -9.4641238   4.1573781  -8.199647
## 7     MB49_I  -9.255835 -28.75448 -1.6588881  35.3289288  30.3300994 -22.854835
## 8     MB55_I  -1.892686 -31.98179 13.4783683 -28.6427702 -21.9613097 -17.434894
## 9    MB55_II   1.746375 -45.19732 -4.4591746 -12.3876222  11.5314349  28.568552
## 10  MB55_III   5.122039 -29.34637 13.7418957  16.9809461 -23.1889814   2.307099
##            PC7        PC8         PC9       PC10        PC11       PC12
## 1   18.4611788   7.366372 -20.9421139 -19.624809   4.4867429   9.188717
## 2    9.5374115   5.985365 -18.6549796  -5.618913   1.2528608   3.918728
## 3   -0.9120698  -1.568812  -0.8753753  12.925110  -5.2460412 -40.696507
## 4   -7.8691026  -3.874488  39.6038231 -26.532968   4.3167860   1.764795
## 5   -9.3464735  -6.177232   1.0357148  13.831232 -37.5307658  15.432420
## 6   -9.1642077  -6.998387   1.6957242  27.852337  31.6755060  12.173110
## 7   -7.5108394  10.088464   1.3204884  -3.066809   1.2113081  -5.116043
## 8   -6.9248273  27.886032  -4.8858633  -6.279176   2.2794854  -5.913050
## 9  -14.7603887 -21.479401 -12.2374353  -6.845160  -1.3217948   6.729829
## 10  34.4160109 -12.807276  14.3398827  11.252958  -0.8825911   3.444580
##           PC13          PC14   tissue   stage grouped_stage
## 1  -27.7123691 -1.718874e-13 midbrain Stage44         early
## 2   36.9435500 -1.307236e-13 midbrain Stage44         early
## 3   -6.1594812 -2.038482e-14 midbrain Stage44         early
## 4    2.6731954 -4.554354e-15 midbrain Stage46         early
## 5   -4.2124420  2.773172e-15 midbrain Stage46         early
## 6   -4.0367374  6.118641e-15 midbrain Stage46         early
## 7    1.5464708 -8.997360e-14 midbrain Stage49        middle
## 8   -0.1393748  1.115400e-13 midbrain Stage55        middle
## 9   -0.3637445  1.066574e-13 midbrain Stage55        middle
## 10   1.4591542  4.097401e-14 midbrain Stage55        middle
```

5.c. Scree Plot A scree plot shows how much variation each PC captures from the data. It can be treated as a diagnostic tool to check whether PCA works well on your data or not. Ideally, the selected PCs should be able to describe at least 80% of the variance.

```
var_explained_no_outlier <- data.frame(PC = paste0("PC", 1:ncol(PC_x_no_outlier$x)),
                                       var_explained_no_outlier = (PC_x_no_outlier$sdev)^2/sum((PC_x_no

PC1to9_Var <- var_explained_no_outlier[1:9,]

ggplot(PC1to9_Var, aes(x= PC, y = var_explained_no_outlier * 100, group = 1)) +
  geom_point(size = 4) +
  geom_line() +
  geom_text(aes(label = round(var_explained_no_outlier, 4)*100, vjust = -1)) +
  labs(title = "Scree plot", y = "Percentage variation explained", x = "PC Scores") +
  theme_classic(base_family = "Times",
                base_size = 14)
```

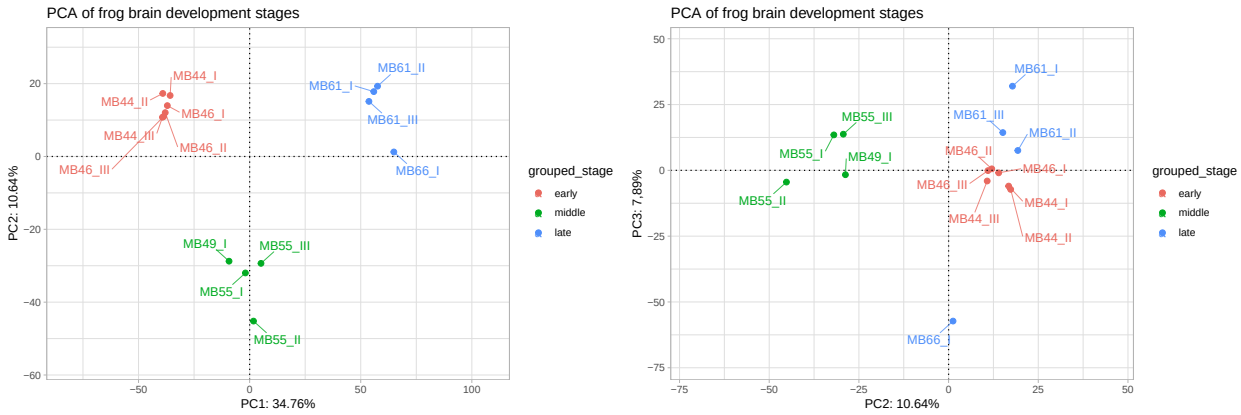


We can see that the top three PCs explain just above 50% of the variation. Although it's not the ideal the proportion of variance retained, we'll stick with it for now and see how the PCA work out.

5.d. PCA Once the outlier is removed, run a principal components analysis (PCA) on the normalized and log transformed data. Plot the PC2 on PC1 as a scatter plot.

```
# Plot the PC1 and PC2 - use early, middle, and late
pca1 <- ggplot(data = PCs_x_no_outlier,
               aes(x = PC1, y = PC2, color = grouped_stage, label = sample_id)) +
          labs(title = "PCA of frog brain development stages",
               x = "PC1: 34.76%",
               y = "PC2: 10.64%") +
          geom_point(size = 2) +
          geom_hline(yintercept = 0, linetype = "dotted") +
          geom_vline(xintercept = 0, linetype = "dotted") +
```

```
          geom_text_repel(box.padding = 0.75,      # Avoid overlapping text
                          max.overlaps = Inf,
                          segment.size = .25,
                          segment.alpha = .8,
                          force = 1) +
          scale_color_brewer(palette = "GnBu") +  # Set color palette (color is for scatter plot)
          scale_colour_hue(l = 60) +              # Darken color
          scale_x_continuous(expand = expansion(mult = 0.5)) +      # Expand x scale of the figure
          scale_y_continuous(expand = expansion(mult = 0.25)) +     # Expand y scale of the figure
          theme_light()

# Plot the PC2 and PC3 - use early, middle, and late
pca2 <- ggplot(data = PCs_x_no_outlier,
               aes(x = PC2, y = PC3, color = grouped_stage, label = sample_id)) +
          labs(title = "PCA of frog brain development stages",
               x = "PC2: 10.64%",
               y = "PC3: 7,89%") +
          geom_point(size = 2) +
          geom_hline(yintercept = 0, linetype = "dotted") +
          geom_vline(xintercept = 0, linetype = "dotted") +
          geom_text_repel(box.padding = 0.75,      # Avoid overlapping text
                          max.overlaps = Inf,
                          segment.size = .25,
                          segment.alpha = .8,
                          force = 1) +
          scale_color_brewer(palette = "GnBu") +  # Set color palette (color is for scatter plot)
          scale_colour_hue(l = 60) +              # Darken color
          scale_x_continuous(expand = expansion(mult = 0.5)) +      # Expand x scale of the figure
          scale_y_continuous(expand = expansion(mult = 0.25)) +     # Expand y scale of the figure
          theme_light()

# Set {r, fig.height = 3, fig.width = 10}
grid.arrange(pca1, pca2, ncol = 2)
```

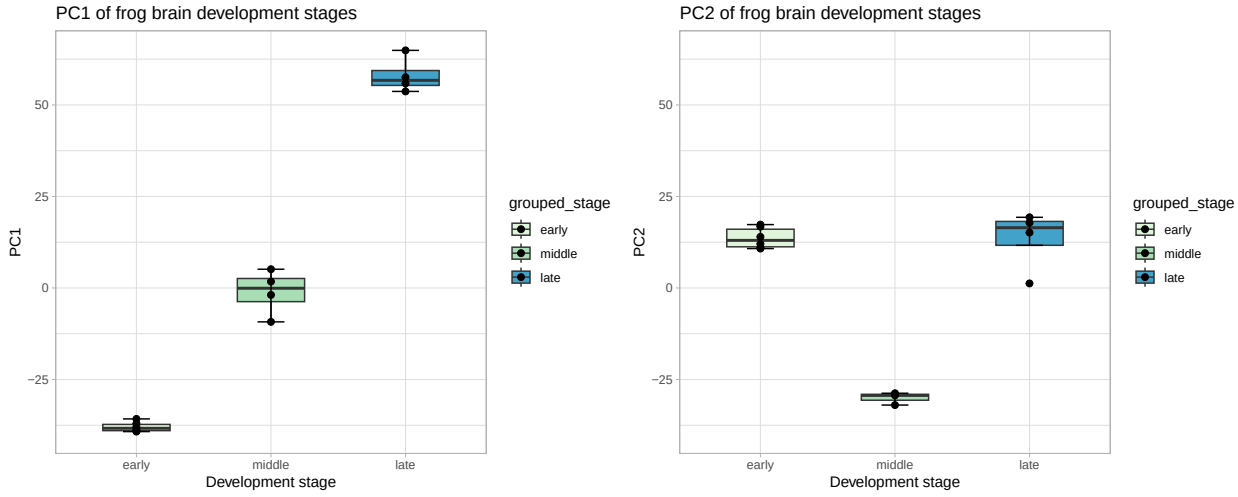


We can see that the plot with PC1 and PC2 do a much better job at forming clusters for the three brain development stages

6. Boxplot

Plot the PC1 and PC2 as a boxplot. Groups on the x-axis, PC1 and PC2 on the y-axis, and make the plot consistent with the color/theme and general aesthetics of the scatter plot for better visualization.

```
# PC1 boxplot
boxplot1 <- ggplot(data = PCs_x_no_outlier,
                   aes(x = grouped_stage, y = PC1, fill = grouped_stage)) +
  ylim(-40, 65) +
  geom_boxplot(width = 0.5) +
  geom_point(size = 2) +
  labs(title = "PC1 of frog brain development stages",
       x = "Development stage",
       y = "PC1") +
  theme_light() +
  scale_fill_brewer(palette = "GnBu") +   # Set color palette (fill is for boxplot)
  scale_colour_hue(l = 60) +              # Darken the color
  stat_boxplot(geom = "errorbar", width = 0.2)        # Add whiskers to the boxplot

# PC2 boxplot
boxplot2 <- ggplot(data = PCs_x_no_outlier,
                   aes(x = grouped_stage, y = PC2, fill = grouped_stage)) +
  ylim(-40, 65) +
  geom_boxplot(width = 0.5) +
  geom_point(size = 2) +
  labs(title = "PC2 of frog brain development stages",
       x = "Development stage",
       y = "PC2") +
  theme_light() +
  scale_fill_brewer(palette = "GnBu") +   # Set color palette (fill is for boxplot)
  scale_colour_hue(l = 60) +              # Darken the color
  stat_boxplot(geom = "errorbar", width = 0.2)        # Add whiskers to the boxplot

# Set {r, fig.height = 5, fig.width = 10}
grid.arrange(boxplot1, boxplot2, ncol = 2)
```



7. Differential gene expression (DGE) analysis

Using the raw counts data frame, perform a differential expression analysis comparing middle and late groups. Perform the analysis so that late is the numerator and middle is the denominator.

```
# Remove anomalous sample
raw_counts_noOutlier <-raw_counts %>%
  dplyr::select(-MB49_II)

# Limit the metadata table to the samples we will include in the DESeq2 analysis
# and make sure the samples are listed in the same order
samples <- data.frame(sample_id = colnames(raw_counts_noOutlier))
samples_metadata <- left_join(samples, metadata)
```

7.a. Data transformation Create a sample table with the conditions to be compared

```
sampleTable <- data.frame(time = samples_metadata$grouped_stage)
rownames(sampleTable) <- samples_metadata$sample_id


# Ensure the order of samples is the same (i.e., return TRUE)
identical(rownames(sampleTable), colnames(raw_counts_noOutlier))
```

```
## [1] TRUE
```

```
# Replace the sample_ids with grouped_stage for DESeq2 analysis
colnames(raw_counts_noOutlier) <- sampleTable$time

# DESeq2 requires counts to be a matrix not a data.frame
raw_counts_noOutlier <- as.matrix(na.omit(raw_counts_noOutlier))

head(raw_counts_noOutlier, 10)
```

```
##          early early early early early early middle middle middle middle  late
## 42sp43.L     3     0     1     1     0     0      1      2      4      3     0
## 42sp50.L     0     1     6     4     0     4      6      6     12      6    13
## ATP6       253   424   406   434   409   547    428    342    326    359   562
## ATP8        11    22    28    17    24    27     33     14     30     20    46
## COX1     13734 23094 21738 23143 19590 26856  15966  16102  19571  16291 25830
## COX2      1543  3149  3575  3174  3337  5019   3356   4101   2591   3529  6449
## COX3      2712  5602  6306  5555  5168  7277   5224   4760   3970   5186  8238
## CYTB        17    43    37    42    45    38     21     11     74     22    27
## ND1        588  1073  1025  1095   873  1242    906    858    945    839  1279
## ND2        428   747   702   738   695   823    728    593    551    629   770
##           late  late  late
## 42sp43.L     4     3     0
## 42sp50.L     1     2     4
## ATP6       383   741   373
```

```
## ATP8        27    39    16
## COX1     16094 25824 14765
## COX2      4079  5087  4289
## COX3      5170  8258  5196
## CYTB        20    26    16
## ND1        759  1170   654
## ND2        637  1071   535
```

7.b. DeSeq2 Unlike TPM normalization, DeSeq2 is a normalization technique that is suitable for gene count comparisons between samples and for DE analysis but NOT for within sample comparisons

```
dds <- DESeqDataSetFromMatrix(countData = raw_counts_noOutlier,
                              colData = sampleTable,
                              design = ~  time)

dds <- DESeq(dds)
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing
```

```
res_late_over_middle <- results(dds,
                                contrast = c("time", "late", "middle"))

summary(res_late_over_middle)
```

```
##
## out of 17361 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 2444, 14%
## LFC < 0 (down)     : 2692, 16%
## outliers [1]       : 10, 0.058%
## low counts [2]     : 1010, 5.8%
## (mean count < 1)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```
resOrdered <- res_late_over_middle[order(res_late_over_middle$padj),]  # orders the output by adjusted p
DE_late_over_middle <- as.data.frame(resOrdered)

head(DE_late_over_middle, 10)
```

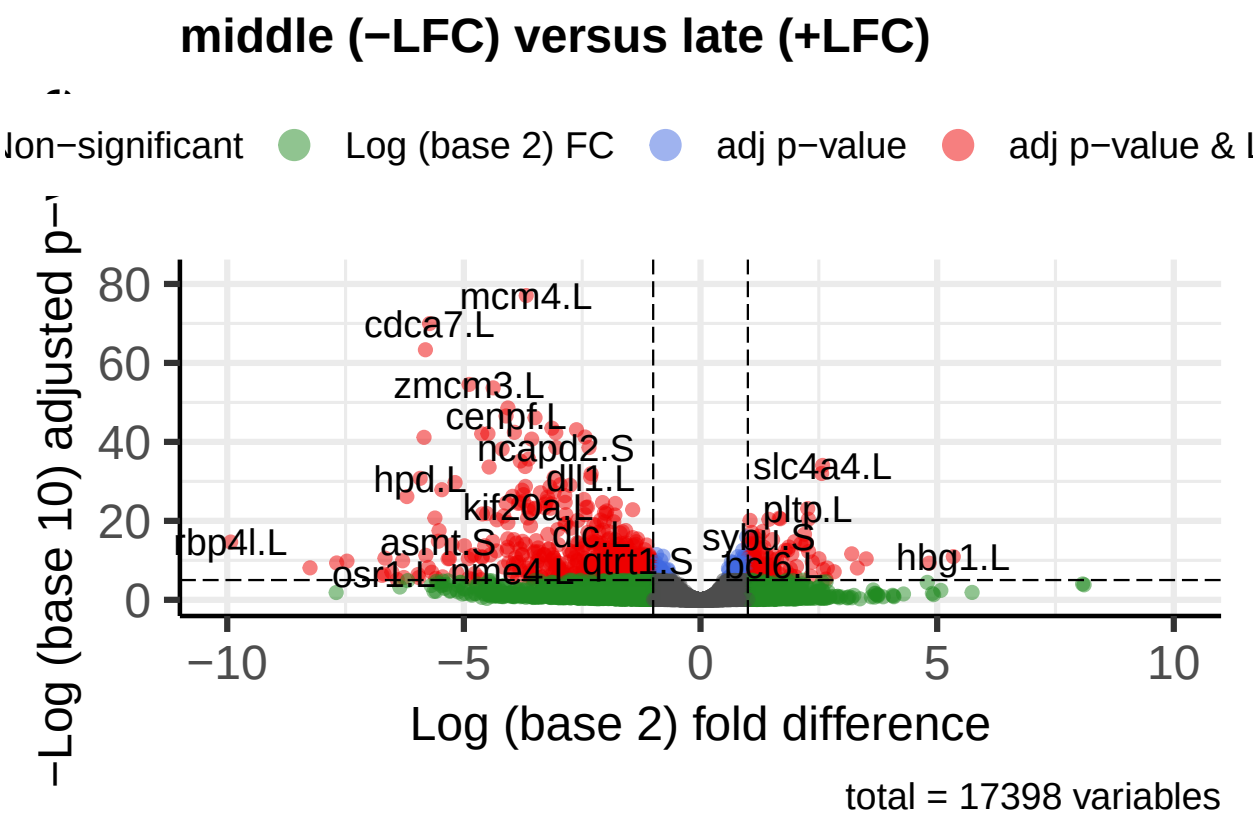
```
##          baseMean log2FoldChange     lfcSE      stat       pvalue         padj
## mcm4.L   730.5616      -3.682259 0.1919607 -19.18236 5.197038e-82 8.492479e-78
## cdca7.L  511.8440      -5.724586 0.3132717 -18.27355 1.344192e-74 1.098272e-70
## cdca7.S  305.3108      -5.812284 0.3340841 -17.39767 8.593457e-68 4.680856e-64
## zmcm3.L  377.5845      -4.889658 0.3021879 -16.18085 6.883590e-59 2.812119e-55
## kif4a.L  397.1978      -4.380624 0.2730529 -16.04313 6.385434e-58 2.086888e-54
```

```
## mcm2.L   558.0529      -4.068582 0.2661414 -15.28729 9.293572e-53 2.531104e-49
## cenpf.L  300.4581      -4.107772 0.2744272 -14.96853 1.178918e-50 2.752101e-47
## mcm6.2.S 531.9371      -3.496025 0.2347455 -14.89283 3.669060e-50 7.494513e-47
## mcm7.L   711.9449      -3.140066 0.2170098 -14.46970 1.883026e-47 3.418948e-44
## fbn3.S   349.7226      -2.620898 0.1819891 -14.40141 5.070220e-47 8.285246e-44
```

8. Volcano plot

Use EnhanceVolcano to plot adjusted p-value on Log2 Fold Difference. We'll use the default p-value cutoff 10e-6.

```
EnhancedVolcano(DE_late_over_middle,
    lab = rownames(DE_late_over_middle),
    title = 'middle (-LFC) versus late (+LFC)',
    subtitle = NULL, # no need for EnhancedVolcano subtitle
    legendLabels = c("Non-significant",
                     "Log (base 2) FC",
                     "adj p-value",
                     "adj p-value & Log (base 2) FC"), # modify labels
    x = "log2FoldChange",
    y = "padj",
    xlab = "Log (base 10) fold difference",
    ylab = "-Log (base 10) adjusted p-value",
    xlim = c(-10,10)) # balanced x-axis around zero to see larger expression changes in either directio
```



With this information, we can pinpoint, for the late stage of frog's brain development compared to the middle stage,

1. Genes that are most up-regulated: slc4a4.L, pltp.L
2. Genes that are most down-regulated: mcm4.L, cdca7.S, zmcm3.L
3. Genes that are most significantly differentially expressed: mcm4.L, cdca7.L

9. Heatmap

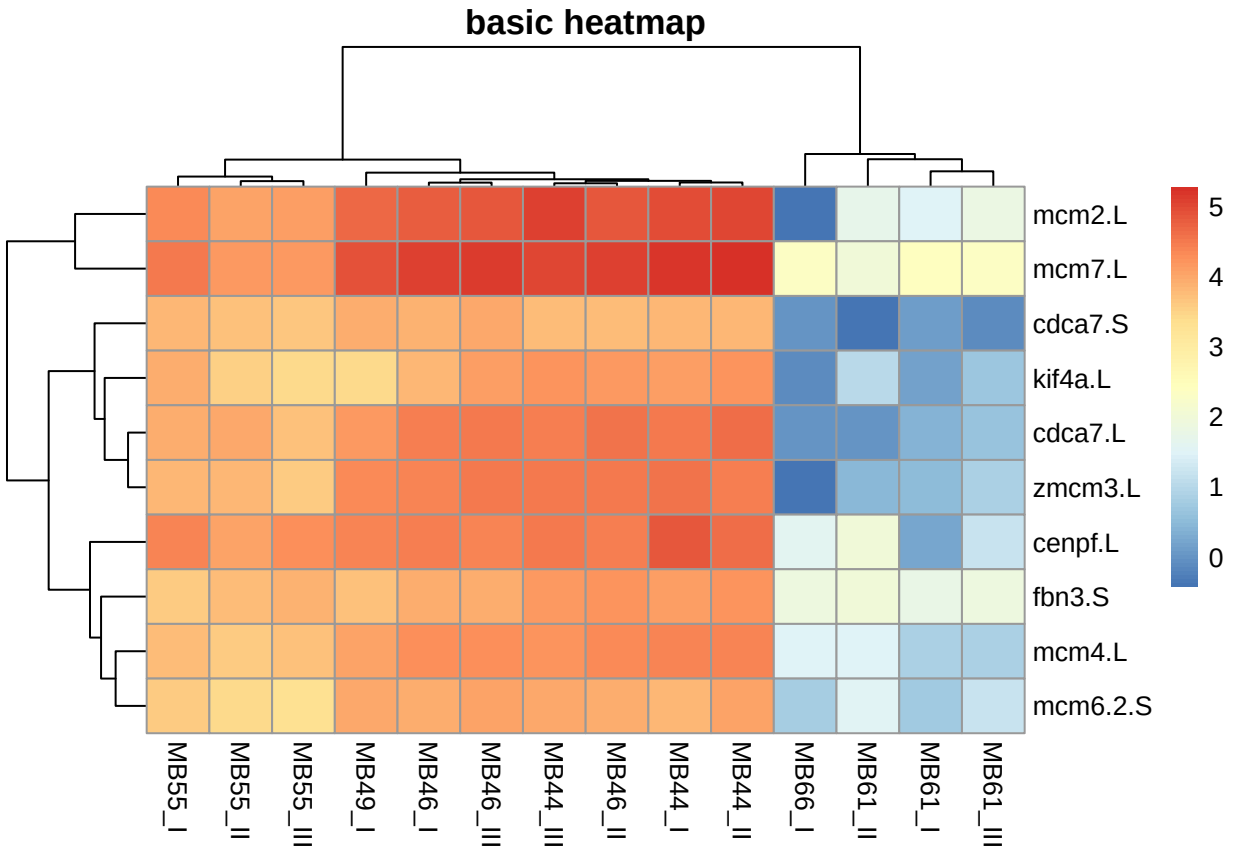
Here's let's filter the differential gene expression analysis to include the top ten most significantly differentially expressed gene and plot a heatmap for those ten genes.

```
# Order gene base on their padj
DE_late_over_middle <- arrange(DE_late_over_middle, desc("padj"))

# Filter top ten most significantly differentially expressed gene (i.e. lowest padj)
DE_late_over_middle_top_10 <- DE_late_over_middle[1:10,]

# Convert rownames to a column
DE_late_over_middle_top_10 <- rownames_to_column(DE_late_over_middle_top_10, var = "gene_id")
log_tpms_no_outlier <- rownames_to_column(log_tpms_no_outlier, var = "gene_id")   # Take the normalized

# Join the two table together
DE_late_over_middle_top_10 <- right_join(log_tpms_no_outlier, DE_late_over_middle_top_10, by = "gene_id

# Remove unnecessary columns for the heatmap
DE_late_over_middle_top_10_transformed <- dplyr::select(DE_late_over_middle_top_10, -c("baseMean", "log
DE_late_over_middle_top_10_transformed <- column_to_rownames(DE_late_over_middle_top_10_transformed, va
DE_late_over_middle_top_10_transformed
```

```
##           MB44_I  MB44_II MB44_III   MB46_I  MB46_II MB46_III   MB49_I
## cdca7.L  4.526196 4.611254 4.424630 4.437422 4.591480 4.521654 4.160052
## cdca7.S  3.838829 3.847326 3.773418 3.864534 3.761996 3.970344 3.913827
## cenpf.L  4.829487 4.648841 4.479861 4.438474 4.472156 4.418860 4.381325
## fbn3.S   4.132451 4.215495 4.154043 3.933931 4.220727 3.913094 3.735176
## kif4a.L  4.125677 4.195871 4.198184 3.821267 4.158286 4.087496 3.426848
## mcm2.L   4.961588 5.040962 5.050289 4.772618 4.870483 4.859965 4.693515
## mcm4.L   4.412902 4.387365 4.242565 4.300755 4.340124 4.268802 4.031090
## mcm6.2.S 3.809596 4.033522 4.017530 3.945674 3.958471 4.053584 3.984987
## mcm7.L   5.182966 5.276968 5.005954 5.098696 5.087112 5.140573 4.908794
## zmcm3.L  4.551867 4.436109 4.525927 4.392413 4.531960 4.497327 4.329732
##           MB55_I  MB55_II MB55_III    MB61_I    MB61_II   MB61_III     MB66_I
## cdca7.L  3.946926 3.971626 3.719960 0.3853649  0.0000000  0.6329771  0.0000000
## cdca7.S  3.815319 3.708041 3.652138 0.1442029 -0.3835248 -0.1207947  0.0000000
## cenpf.L  4.365106 4.056634 4.266909 0.2042636  2.0054383  1.1740106  1.5764731
## fbn3.S   3.614825 3.775641 3.874962 1.7651514  2.0258811  1.8927156  1.9074126
## kif4a.L  3.959255 3.561013 3.448350 0.1802287  1.0540628  0.6644678 -0.1370424
## mcm2.L   4.346057 4.075493 4.086594 1.4706036  1.7313333  1.8117419 -0.3760627
## mcm4.L   3.764344 3.619032 3.703390 0.8574349  1.4695624  0.8842228  1.5044089
## mcm6.2.S 3.568992 3.449745 3.290896 0.7349696  1.5596347  1.2237439  0.8298750
```

```
## mcm7.L   4.531380 4.193562 4.152419 2.3768200  2.0073163  2.3349660  2.3390809
## zmcm3.L  3.826458 3.822287 3.567162 0.5075785  0.4806261  0.8487168 -0.4227971
```

```
# Plot a basic heatmap
pheatmap(DE_late_over_middle_top_10_transformed, main = "basic heatmap")
```



Now, let's add more annotation to our heatmap and its axes. Furthermore, the normal convention would be plotting the samples on the horizontal axis, so we can transposed our matrix

```
# Create a data frame for heatmap's row annotation (i.e. the developmental stages)
# Firstly, set up data frame with row names as grouped stages
row_annotation <- data.frame(stage_annotation = matrix(ncol = 1,
                                                       nrow = length(colnames(DE_late_over_middle_top_1
row.names(row_annotation) <- colnames(DE_late_over_middle_top_10_transformed)    # Assign column names

# Now, insert values into the stage_annotation column
row_annotation <- mutate(row_annotation, stage_annotation = case_when(
  startsWith(row.names(row_annotation), "MB44") ~ "early",
  startsWith(row.names(row_annotation), "MB46") ~ "early",
  startsWith(row.names(row_annotation), "MB49") ~ "middle",
  startsWith(row.names(row_annotation), "MB55") ~ "middle",
  startsWith(row.names(row_annotation), "MB61") ~ "late",
  startsWith(row.names(row_annotation), "MB66") ~ "late"
))
row_annotation$stage_annotation <- factor(row_annotation$stage_annotation, levels = c("early", "middle"
```

```
row_annotation
```

```
##          stage_annotation
## MB44_I              early
## MB44_II             early
## MB44_III            early
## MB46_I              early
## MB46_II             early
## MB46_III            early
## MB49_I             middle
## MB55_I             middle
## MB55_II            middle
## MB55_III           middle
## MB61_I               late
## MB61_II              late
## MB61_III             late
## MB66_I               late
```

```
# Create a data frame for heatmap's column annotation (i.e. the genes)
col_annotation <- data.frame(expression_change = matrix(ncol = 1, nrow = 10))  # Since we're working wi
row.names(col_annotation) <- rownames(DE_late_over_middle_top_10_transformed)  # Assign the row names (
col_annotation <- cbind(col_annotation, DE_late_over_middle_top_10["log2FoldChange"])

# Now, insert values into the expression_change column
col_annotation <- mutate(col_annotation, expression_change = case_when(
  log2FoldChange > 0 ~ "up-regulated",
  log2FoldChange < 0 ~ "down-regulated"
))
col_annotation <- col_annotation["expression_change"]   # Remove the log2FoldChange column
col_annotation
```

```
##          expression_change
## cdca7.L     down-regulated
## cdca7.S     down-regulated
## cenpf.L     down-regulated
## fbn3.S      down-regulated
## kif4a.L     down-regulated
## mcm2.L      down-regulated
## mcm4.L      down-regulated
## mcm6.2.S    down-regulated
## mcm7.L      down-regulated
## zmcm3.L     down-regulated
```

Now, let's plot our annotated heatmap

```
pheatmap(t(DE_late_over_middle_top_10_transformed),
         annotation_row = row_annotation,
         annotation_col = col_annotation,
         cutree_cols = 3,
         cutree_rows = 2,
         main = "Annotated, clusterized heatmap of top ten most significantly differentially expressed
```

sterized heatmap of top ten most significantly differentially expressed gene during middle vs. late stage