Intoduction to the tidyverse

UND Genomics Core Bioinformatics Workshop

2/22/2019

In this session we will be using some of the functions from the tidyverse (https://www.tidyverse.org) package to manipulate data. The tidyverse is actually a collection of packages created by Hadley Wickham, that are designed to be used together and have intuitive syntax.

We will be covering five functions from the tidyverse:
rename()
filter()
select()
mutate()

We will be using two datasets in this session: a differential expression table, and some normalized counts. Both of these datasets can be generated using the RNA-Seq data analysis pipeline we will be describing tomorrow. The two files you will need today were sent by email yesterday night. They're called:
DEtable_Day1_pm.csv
normCounts_Day1_pm.csv
setwd("/home/")

Install the packages you need for this session:

```
install.packages("tidyverse")
install.packages("plotly")
```

After the packages has beeen installed, you need to load the packages into your workspace. You only need to install packages once. The next time you need to used the tidyverse package, you only need to load the package using the *library()* function.

```
library(tidyverse)
```

```
## -- Attaching packages ----------------------------------------------------------------------------

## v ggplot2 3.1.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------------------------- t
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(plotly)
```

```
##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout
```

When the tidyverse package loads, it will list all the packages that have been loaded into your workspace. Next, import the two datasets we'll be using.

```
detable  <- read.csv("DEtable_Day1_pm.csv")
counts <- read.csv("normCounts_Day1_pm.csv")
```

Let's look at the two dataframes we'll be using for this session.

```
head(detable)
```

```
##        X    baseMean log2FoldChange     lfcSE       stat       pvalue
## 1   Xkr4   19.630476      4.8961503 1.2786657  3.8291089 0.0001286081
## 2    Rp1    3.013894      2.6860518 1.5873427  1.6921688 0.0906137886
## 3  Sox17    0.000000             NA        NA         NA           NA
## 4 Mrpl15 1621.512498     -0.1667415 0.2770196 -0.6019121 0.5472326500
## 5 Lypla1 2968.963407     -0.6996000 0.1836766 -3.8088682 0.0001396044
## 6  Tcea1 2690.908655     -0.3750856 0.2731069 -1.3734021 0.1696273625
##           padj
## 1 0.0002983894
## 2 0.1297462076
## 3           NA
## 4 0.6125575329
## 5 0.0003223510
## 6 0.2259874681
```

This detable table is the output of DESeq2, which we'll be covering tomorrow. One thing we need to fix is the name of the first column which is named "X" - not informative -let's rename it using the *rename* function. All tidyverse functions take the format:
function(data,action)

```
detable <- rename(detable,geneid = X)
head(detable)
```

```
##   geneid    baseMean log2FoldChange     lfcSE       stat       pvalue
## 1   Xkr4   19.630476      4.8961503 1.2786657  3.8291089 0.0001286081
## 2    Rp1    3.013894      2.6860518 1.5873427  1.6921688 0.0906137886
## 3  Sox17    0.000000             NA        NA         NA           NA
## 4 Mrpl15 1621.512498     -0.1667415 0.2770196 -0.6019121 0.5472326500
## 5 Lypla1 2968.963407     -0.6996000 0.1836766 -3.8088682 0.0001396044
## 6  Tcea1 2690.908655     -0.3750856 0.2731069 -1.3734021 0.1696273625
##           padj
## 1 0.0002983894
## 2 0.1297462076
## 3           NA
```

```
## 4 0.6125575329
## 5 0.0003223510
## 6 0.2259874681
```

Let's look at the counts dataframe:

```
head(counts)
```

```
##   geneid X15.007.001 X15.007.003 X15.007.004 X15.008.001  X15.008.003
## 1   Xkr4    52.28333    1.144850   61.545284    1.008126    0.9352279
## 2    Rp1     6.85683    3.434551    5.917816    1.008126    0.0000000
## 3  Sox17     0.00000    0.000000    0.000000    0.000000    0.0000000
## 4 Mrpl15  1049.09499 1741.317465 1793.098191 2010.202611 1537.5147447
## 5 Lypla1  2502.74294 2319.466919 1962.347723 3657.479977 3710.9844933
## 6  Tcea1  2719.59019 2744.206419 1561.119811 3361.091026 2766.4042669
##    X15.008.004
## 1    0.8660417
## 2    0.8660417
## 3    0.0000000
## 4 1597.8469907
## 5 3660.7583902
## 6 2993.0402168
```

In this dataframe, each column is a sample, and each row is a gene.

Filtering

The *filter()* function takes at dataframe and a filtering criteria. The filtering criteria must return a logical (TRUE/FALSE) vector. As an example, to filter for significant genes, the criteria is: an FDR (padj) less than or equal to 0.05:
padj<=0.05 .

```
sig <- filter(detable,padj <=0.05)
```

How many genes are significantly DE? If you're using Rstudio, you can just look at environment pane and see that the sig object had 10990 obs. But you can also use the function *nrow()* which returns the number of rows of dataframe

```
nrow(sig)
```

```
## [1] 10990
```

It's also possible to string commands together using the pipe "%>%" operator.

```
filter(detable,padj <=0.05) %>% nrow()
```

```
## [1] 10990
```

Now you want to know how many genes are upregulated. How would you do that? The filter command allows you to combine criteria as well. Now we're adding the criteria of the log2FoldChange being greater than 0:

log2FoldChange > 0

```
filter(detable,padj<=0.05,log2FoldChange>0) %>% nrow()
```

```
## [1] 6060
```

You try! Find how many genes are down regulated.

You can also use *grep* to filter for particular genes, but you'll need to a variation on the *grep* function: the *grepl* function which returns a logical (T/F) vector, instead of the rownumber that contain a match. Compare:

```
grep("Act",detable$geneid) %>% head()
```

```
## [1]  155  712 2084 2594 4208 4572
```

and grepl("Act",detable$geneid) %>% head()

```
grepl("Act",detable$geneid) %>% head()
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

To find the actinb genes:

```
filter(detable,grepl("Actb",geneid))
```

```
##   geneid baseMean log2FoldChange     lfcSE      stat      pvalue
## 1 Actbl2      0.0             NA        NA        NA          NA
## 2   Actb 221732.2      -1.351257 0.1771932 -7.625897 2.42344e-14
##           padj
## 1           NA
## 2 1.230651e-13
```

You can also fiter the counts dataframe to look at the counts of individual genes.

```
filter(counts,grepl("Actb",geneid))
```

```
##   geneid X15.007.001 X15.007.003 X15.007.004 X15.008.001 X15.008.003
## 1 Actbl2           0         0.0         0.0         0.0         0.0
## 2   Actb      110515    126968.5    136767.8    341112.4    302200.2
##   X15.008.004
## 1           0
## 2      312829
```

You try! Now try filtering the detable and counts dataframes for your favourite genes.

Selecting Columns

In the tidyverse, selecting columns is done with the *select()* function. Some example:

```
select(detable,geneid) %>% head()
```

```
##   geneid
## 1   Xkr4
## 2    Rp1
## 3  Sox17
## 4 Mrpl15
## 5 Lypla1
## 6  Tcea1
```

```
select(detable,geneid:log2FoldChange) %>% head() # select a range of columns
```

```
##   geneid    baseMean log2FoldChange
## 1   Xkr4   19.630476      4.8961503
## 2    Rp1    3.013894      2.6860518
## 3  Sox17    0.000000             NA
## 4 Mrpl15 1621.512498     -0.1667415
## 5 Lypla1 2968.963407     -0.6996000
## 6  Tcea1 2690.908655     -0.3750856
```

```
select(counts,-geneid) %>% head() # drop columns
```

```
##   X15.007.001 X15.007.003 X15.007.004 X15.008.001  X15.008.003
## 1    52.28333    1.144850   61.545284    1.008126    0.9352279
## 2     6.85683    3.434551    5.917816    1.008126    0.0000000
## 3     0.00000    0.000000    0.000000    0.000000    0.0000000
## 4  1049.09499 1741.317465 1793.098191 2010.202611 1537.5147447
## 5  2502.74294 2319.466919 1962.347723 3657.479977 3710.9844933
## 6  2719.59019 2744.206419 1561.119811 3361.091026 2766.4042669
##    X15.008.004
## 1    0.8660417
## 2    0.8660417
## 3    0.0000000
## 4 1597.8469907
## 5 3660.7583902
## 6 2993.0402168
```

The select function is more useful after a filtering step, For example to get a list of the upregulated gene names

```
up_reg_genes <- filter(detable,padj <=0.05,log2FoldChange>0) %>% select(geneid)
head(up_reg_genes)
```

```
##          geneid
## 1          Xkr4
## 2         Rgs20
## 3        Adhfe1
## 4 3110035E14Rik
## 5         Mybl1
## 6         Snhg6
```

Notice that when joining functions together using the "%>%" you don't have to specify that dataframe in the second function.This genelist can used to create venn diagrams, or to import pathway anaysis programs. To export this list for use in pathway analysis programs like pantherDB. you would do:

```
write.table(up_reg_genes,file = "up_regulated_genes.txt",row.names = F,col.names = F,quote = F)
```

The option *row.names* = *F* tells R to not export rownumbers.
The option *col.names* = *F* tells R to include the column name (geneid)
The option *quote* = *F* tells R to not put quotes around the gene names.

You try! Try exporting a list of down-regulated genes.

Add columns using mutate

The final function we'll be covering is the *mutate()* function, which allows your to create new columns. An example is adding a column indicated if a gene is differentially expressed based on the adjusted p-value.

```
detable <- mutate(detable,Sig = ifelse(padj<=0.05,"yes","no"))
head(detable)
```

```
##   geneid    baseMean log2FoldChange     lfcSE       stat       pvalue
## 1   Xkr4   19.630476      4.8961503 1.2786657  3.8291089 0.0001286081
## 2    Rp1    3.013894      2.6860518 1.5873427  1.6921688 0.0906137886
## 3  Sox17    0.000000             NA        NA         NA           NA
## 4 Mrpl15 1621.512498     -0.1667415 0.2770196 -0.6019121 0.5472326500
## 5 Lypla1 2968.963407     -0.6996000 0.1836766 -3.8088682 0.0001396044
## 6  Tcea1 2690.908655     -0.3750856 0.2731069 -1.3734021 0.1696273625
##           padj  Sig
## 1 0.0002983894  yes
## 2 0.1297462076   no
## 3           NA <NA>
## 4 0.6125575329   no
## 5 0.0003223510  yes
## 6 0.2259874681   no
```

I'm noticing that there are some NA in that column. You can also use mutate to replace NA's

```
detable <- mutate(detable,Sig = replace_na(Sig,"no"))
head(detable)
```

```
##   geneid    baseMean log2FoldChange     lfcSE       stat       pvalue
## 1   Xkr4   19.630476      4.8961503 1.2786657  3.8291089 0.0001286081
## 2    Rp1    3.013894      2.6860518 1.5873427  1.6921688 0.0906137886
## 3  Sox17    0.000000             NA        NA         NA           NA
## 4 Mrpl15 1621.512498     -0.1667415 0.2770196 -0.6019121 0.5472326500
## 5 Lypla1 2968.963407     -0.6996000 0.1836766 -3.8088682 0.0001396044
## 6  Tcea1 2690.908655     -0.3750856 0.2731069 -1.3734021 0.1696273625
##           padj Sig
## 1 0.0002983894 yes
## 2 0.1297462076  no
## 3           NA  no
## 4 0.6125575329  no
## 5 0.0003223510 yes
## 6 0.2259874681  no
```

You try! Try adding a column to indicate if a gene is significant or not.

Plotting with ggplot

We learned how to make basic plots this morning, in this session we'll go over some ways to make some nicer plots with the ggplot package found within the tidyverse. The function we'll be using is *ggplot()*. The basic structure of the command is:

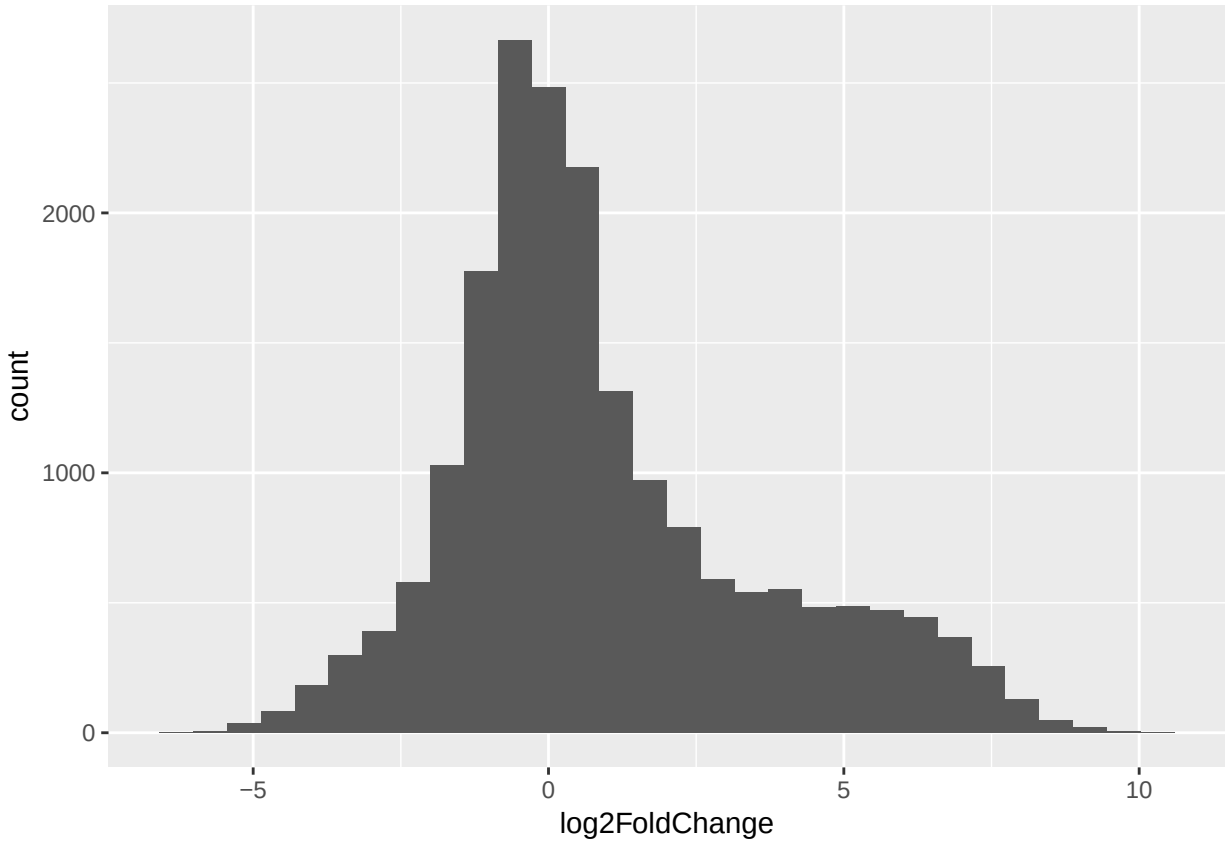
ggplot(dataset,aes(x,y)) + geom()

Geoms can be though of as ways to display your data, ex
geom_histogram()
geom_boxplot()
geom_point()
geom_line()

```
ggplot(detable, aes(log2FoldChange)) +
        geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 4879 rows containing non-finite values (stat_bin).
```

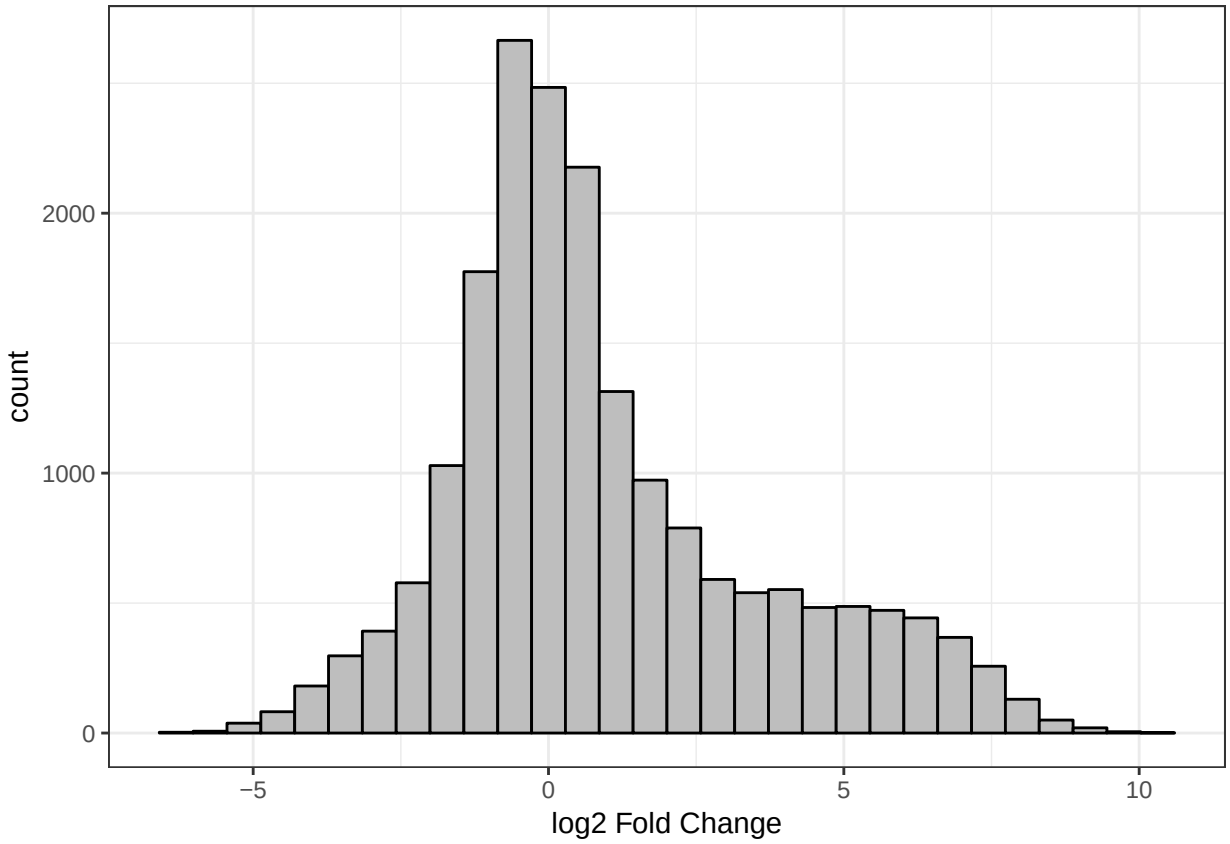


You will get some warnings about number of bins, and that rows were removed due to "non-finite values", that's okay, we'll ignore that for now. This plot is okay, but we can make it better:

```
ggplot(detable, aes(log2FoldChange)) +
         geom_histogram(color = "black",fill = "gray") +
          xlab("log2 Fold Change") +
            theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 4879 rows containing non-finite values (stat_bin).
```



Then save plot using the function *ggsave*:

```
ggsave("log2FoldChanges.png")
```

```
## Saving 6.5 x 4.5 in image
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 4879 rows containing non-finite values (stat_bin).
```

Let's move on to some more interesting plots. These next few plots will be of the normalized counts for selected genes. First we need to get our data into a format that is suitable for ggplot, including addaing some information about the samples like cell type.

```
cellType <- c(rep("Astrocytes",3),rep("Microglia",3))
sampleName <- paste(cellType,1:3,sep = "_")
rownames(counts) <- counts$geneid
counts <- counts[,-1]
t_counts <- t(counts) %>%
  as.data.frame() %>%
  mutate(cellType=cellType,sampleName = sampleName)

plot_counts <- gather(t_counts,geneName,counts,-cellType,-sampleName)
```

Look at the plot_counts dataframe:

```
head(plot_counts)
```

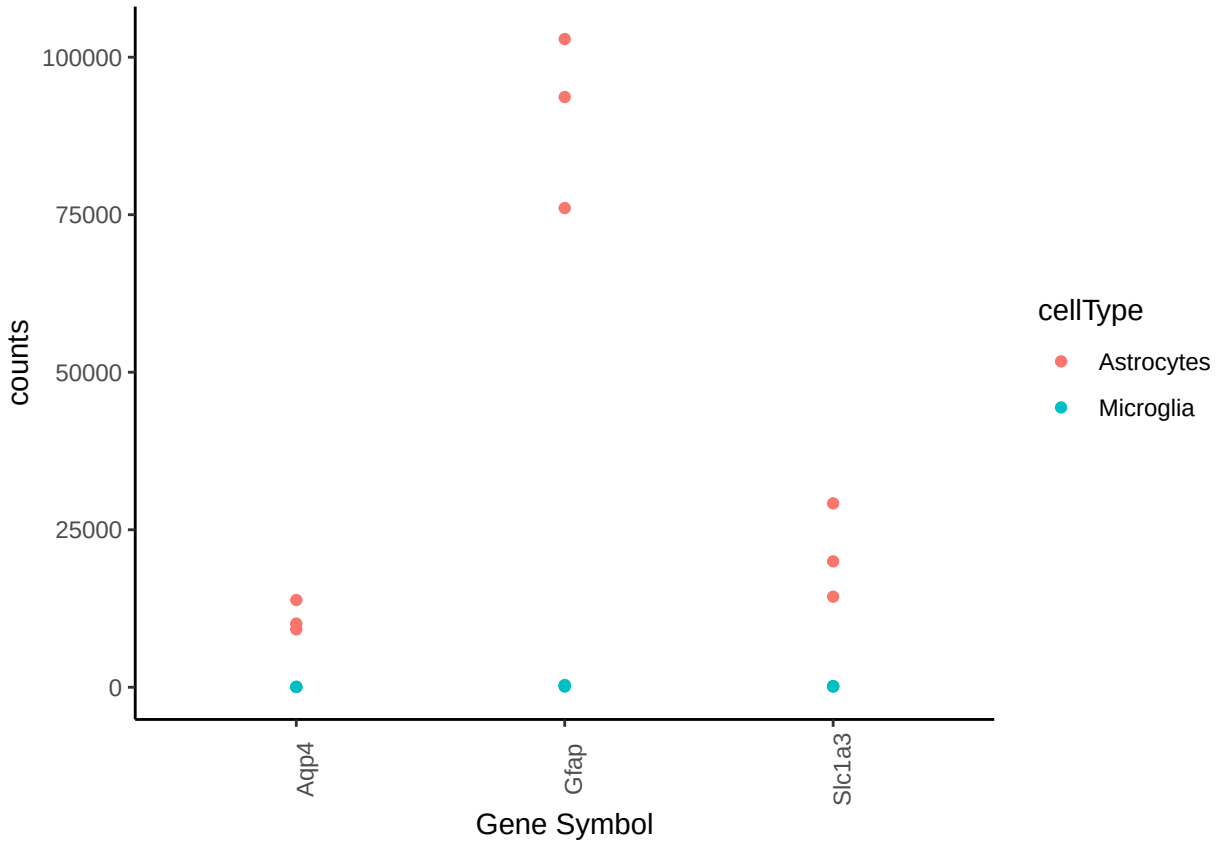
```
##     cellType   sampleName geneName     counts
## 1 Astrocytes Astrocytes_1     Xkr4 52.2833285
## 2 Astrocytes Astrocytes_2     Xkr4  1.1448504
## 3 Astrocytes Astrocytes_3     Xkr4 61.5452844
## 4  Microglia  Microglia_1     Xkr4  1.0081257
## 5  Microglia  Microglia_2     Xkr4  0.9352279
## 6  Microglia  Microglia_3     Xkr4  0.8660417
```

Use *filter()* to select some genes to plot:

```
genes_to_plot <- c("Gfap","Slc1a3","Aqp4")
gene_plot_df <- filter(plot_counts,geneName %in% genes_to_plot)
```

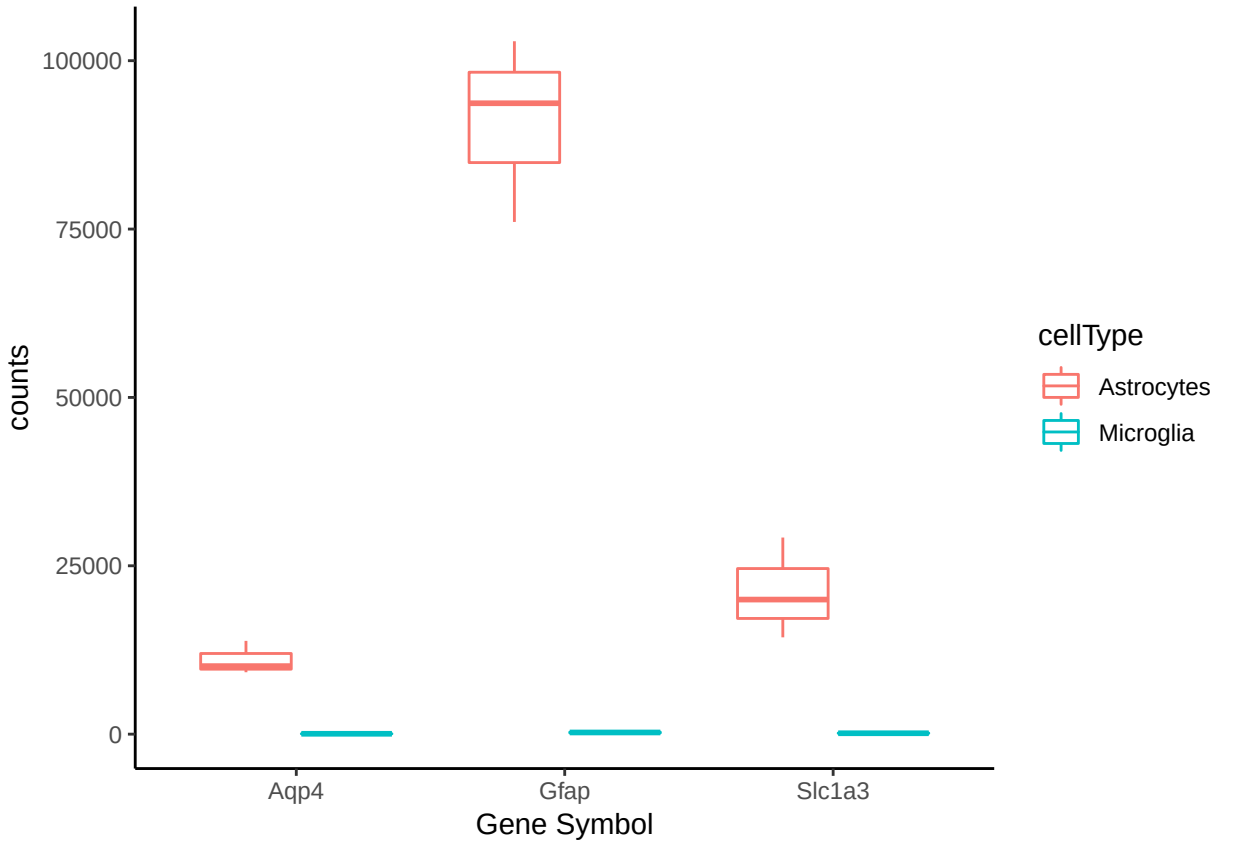
and make some plots:

```
ggplot(gene_plot_df,aes(geneName,counts)) +
  geom_point(aes(color = cellType)) +
  xlab("Gene Symbol") +
  theme_classic() +
  theme(axis.text.x = element_text(angle = 90))
```



You can change to type of using the *geom* line, ex change the above graph to a boxplot:

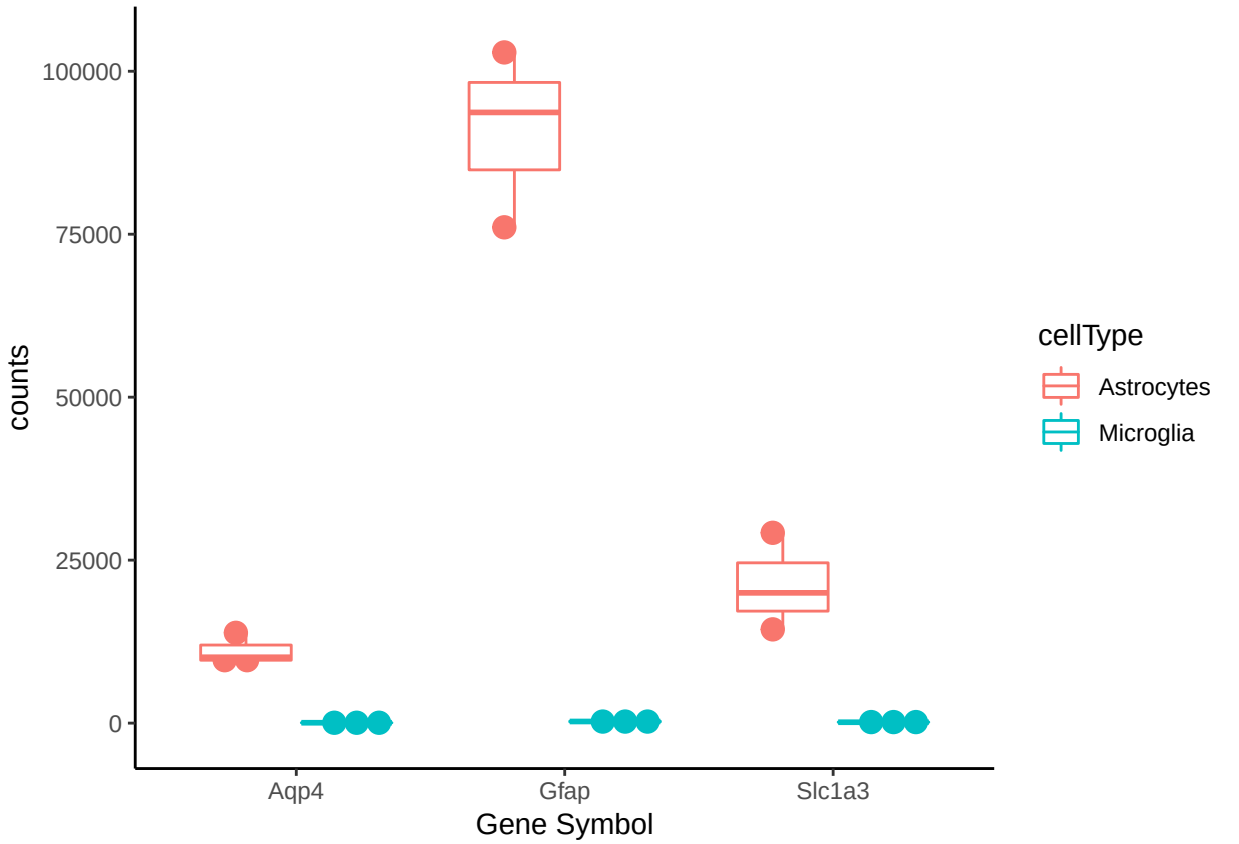
```
ggplot(gene_plot_df,aes(geneName,counts)) +
   geom_boxplot(aes(color = cellType)) +
   xlab("Gene Symbol") +
  theme_classic()
```



Or combine the points and boxplots together:

```
ggplot(gene_plot_df,aes(geneName,counts)) +
  geom_dotplot(aes(color = cellType,fill = cellType),position = position_dodge(),binaxis = "y",stackdir
   geom_boxplot(aes(color = cellType)) +
   xlab("Gene Symbol") +
   theme_classic()
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```

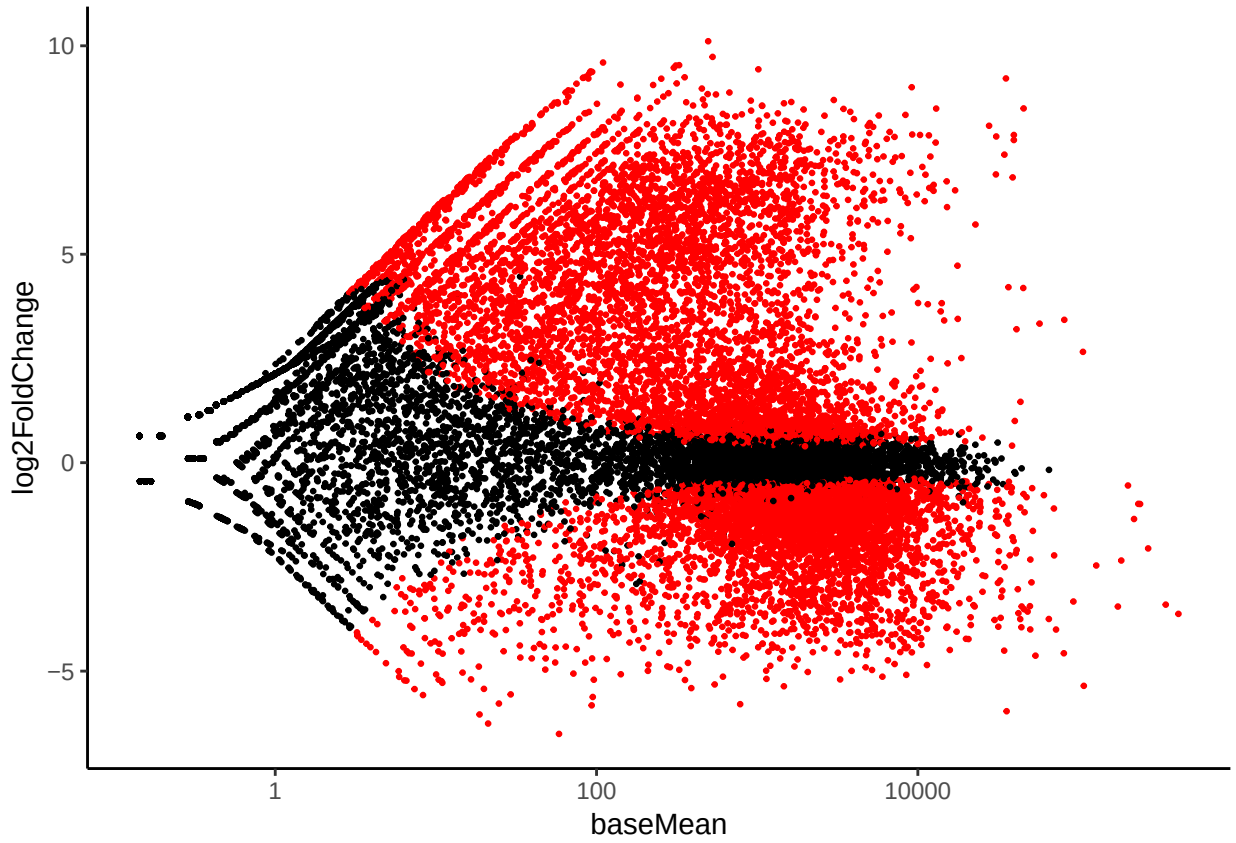


Try making some plots of your favourite genes! To do so, in the line:
genes_to_plot <- **c("Gfap","Slc1a3","Aqp4")**, replace whats' in bold, with gene names, and run the rest of the code

Making an MA plot

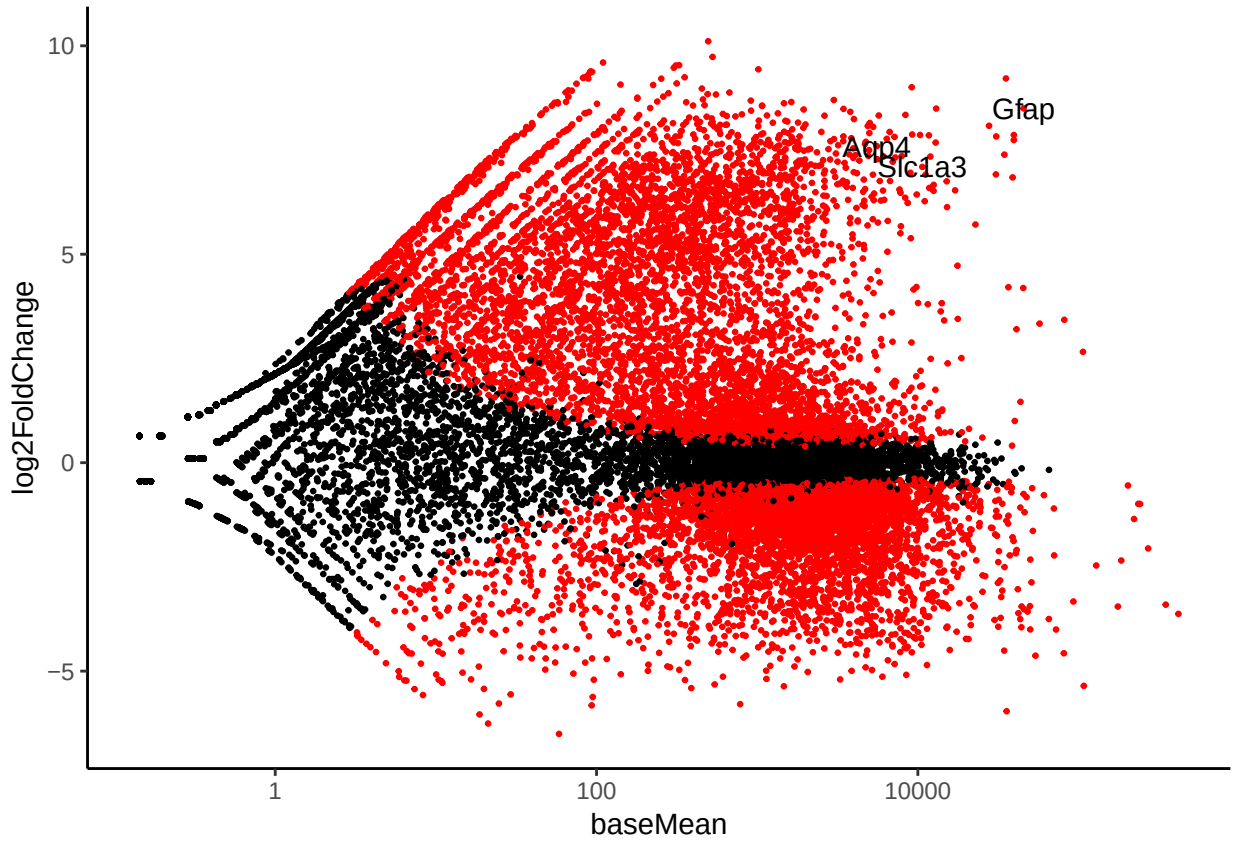
```
detable_filtered <- filter(detable,baseMean>0)

ggplot(detable_filtered,aes(baseMean,log2FoldChange)) +
  geom_point(aes(color = Sig),size =  0.5) +
  scale_x_continuous(trans= "log10") +
  scale_color_manual(breaks = c("ns","Sig","NA"), values = c("black","red","black")) +
  theme_classic()
```



You can also add gene names to your plot using geom_text.

```
ggplot(detable_filtered,aes(baseMean,log2FoldChange)) +
geom_point(aes(color = Sig),size =  0.5) +
geom_text(data = filter(detable,geneid %in% genes_to_plot), aes(x= baseMean,y=log2FoldChange,label = g
scale_x_continuous(trans= "log10") +
scale_color_manual(breaks = c("ns","Sig","NA"), values = c("black","red","black")) +
theme_classic()
```



The last feature I'll show you is how to make interactive plot using plotly:

```
 p <- ggplot(detable_filtered,aes(baseMean,log2FoldChange,text = paste0("Gene name: ",geneid,"<br> log2
  geom_point(aes(color = Sig),size =  0.5) +
  scale_x_continuous(trans= "log10") +
  scale_color_manual(breaks = c("ns","Sig","NA"), values = c("black","red","black")) +
  theme_classic()


ggplotly(p,tooltip = "text")
```