Big data visualization

Kim Dill-McFarland

2024-09-09

Table of contents

Introduction

In this workshop, we present several methods for improving data visualizations of large data sets. The example data come from human RNA sequencing, but the methods presented are data agnostic and applicable in many fields! If you want to learn more about RNA sequencing data analysis or see some other fun workshops, checkout Kim's work at https://bigslu.github.io/workshops/

Packages

Load packages for this workshop.

```
#Data manipulation and plotting
library(tidyverse)

#RNA-seq data format
library(limma)
#Automatically position non-overlapping text labels with ggplot2
library(ggrepel)
#Plot bee-swarm with ggplot2
library(ggbeeswarm)
#Plot pairwise comparisons with ggplot2
library(GGally)
#Combine ggplot2 objects
library(patchwork)
```

Data

We will use RNA sequencing data as an example of "big data". These data were pre-cleaned and normalized so we can get right to plotting!

To download the data locally or on the R server, use the following code.

```
#Create data directory
dir.create("data", showWarnings = FALSE)
#Down data
download.file(
  url = "https://github.com/BIGslu/2022_ASM_Microbe_RNAseq/raw/main/0_data/dat_voom.RData",
  destfile = "data/rna_data.RData")
```

Then load the data into your R session and create a combined data frame for plotting.

```
#Load data
attach("data/rna_data.RData")

dat <- as.data.frame(dat.abund.norm.voom$E) %>%
  #Pivot expression data to long format
  rownames_to_column("gene") %>%
  pivot_longer(-gene, names_to = "libID",
               values_to = "log2Expression") %>%
  #Combine in data metadata
  left_join(dat.abund.norm.voom$targets, by ="libID") %>%
  #Keep columns necessary for this workshop
  select(libID, ptID, condition, gene, log2Expression) %>%
  #Return to wide format
  pivot_wider(names_from = "gene", values_from = "log2Expression")
```

Data overview

```
dat[1:4,1:5]
```

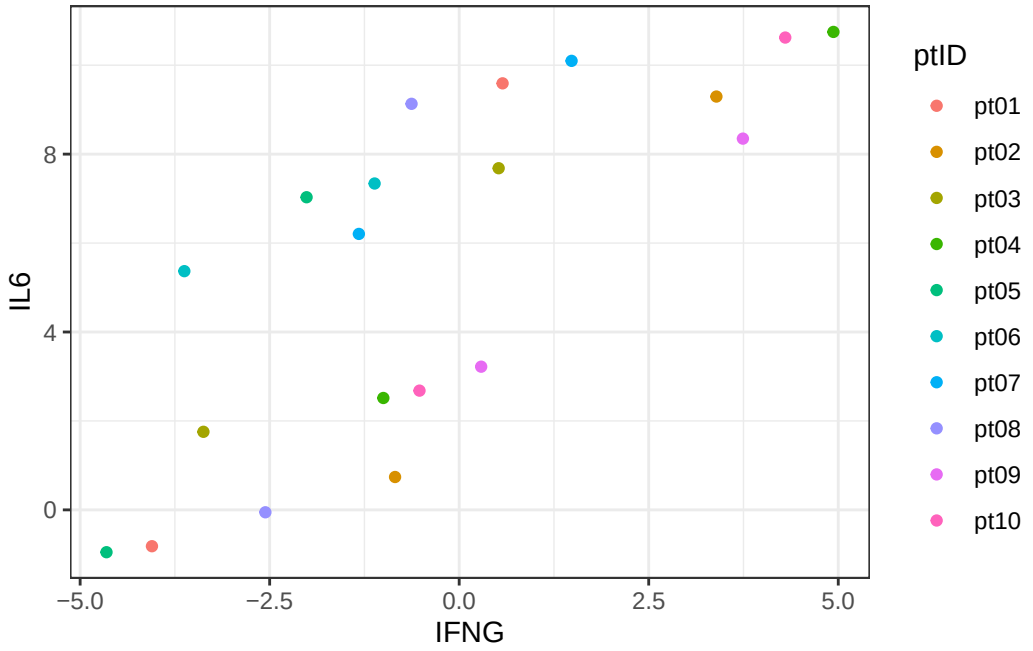
```
# A tibble: 4 x 5
  libID      ptID  condition  A1BG   A2M
  <chr>      <chr> <chr>     <dbl> <dbl>
1 pt01_Media pt01  Media      1.62  7.26
2 pt01_Mtb   pt01  Mtb        1.72  6.68
3 pt02_Media pt02  Media      1.09  5.94
4 pt02_Mtb   pt02  Mtb        1.10  5.32
```

- Bulk RNA sequencing to measure gene expression
- 10 human patients (`ptID`)
- Immune cells in media or infected with *Mycobacterium tuberculosis* (`condition`)
  - = 20 unique samples (`libID`)
- Each gene is a column with 13419 genes
  - Gene expression is measured in log2 counts per million

Challenge 1: Too many colors!

Here, we compare the expression of two genes within patients. Even with only 10 unique patients, the colors are difficult to read and definitely not colorblind accessible. We could try to find a better palette, but there are also better ways to display these data!

```
ggplot(data = dat) +
  aes(x = IFNG, y = IL6, color = ptID) +
  geom_point() +
  theme_bw()
```

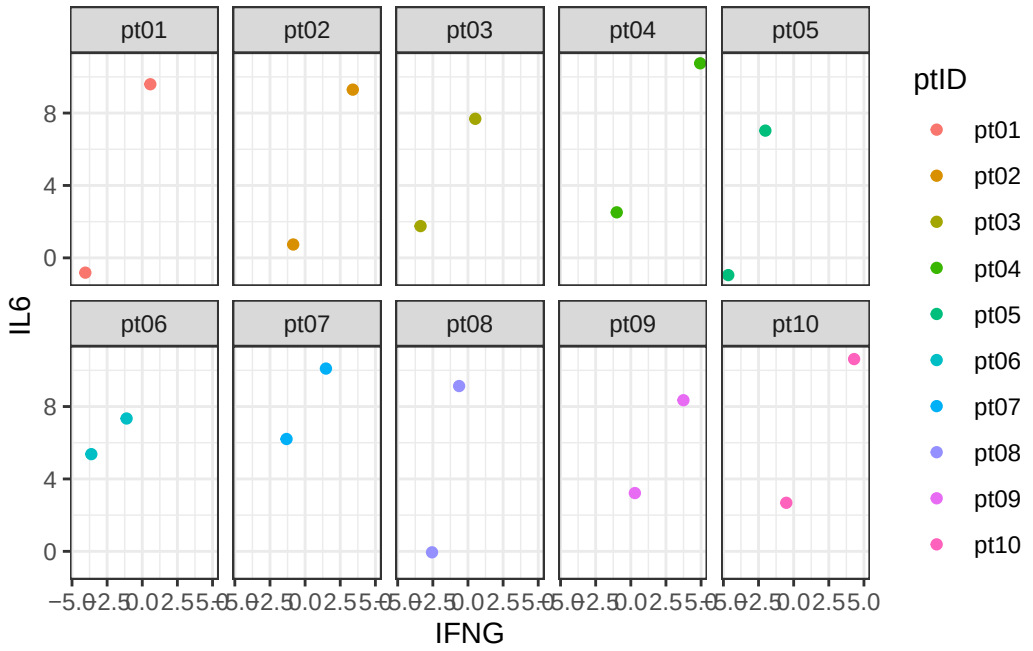


Solution 1A: Facets

One option when faced with too many colors is to split the plot into multiple subplots (*i.e.* facets). In our example here, patients are now clearly labeled but it is difficult make comparisons between patients (which was the original intent of the plot). So, this is not the best solution for this plot but can be for others.

```
ggplot(data = dat) +
  aes(x = IFNG, y = IL6, color = ptID) +
  geom_point() +
  theme_bw() +
```

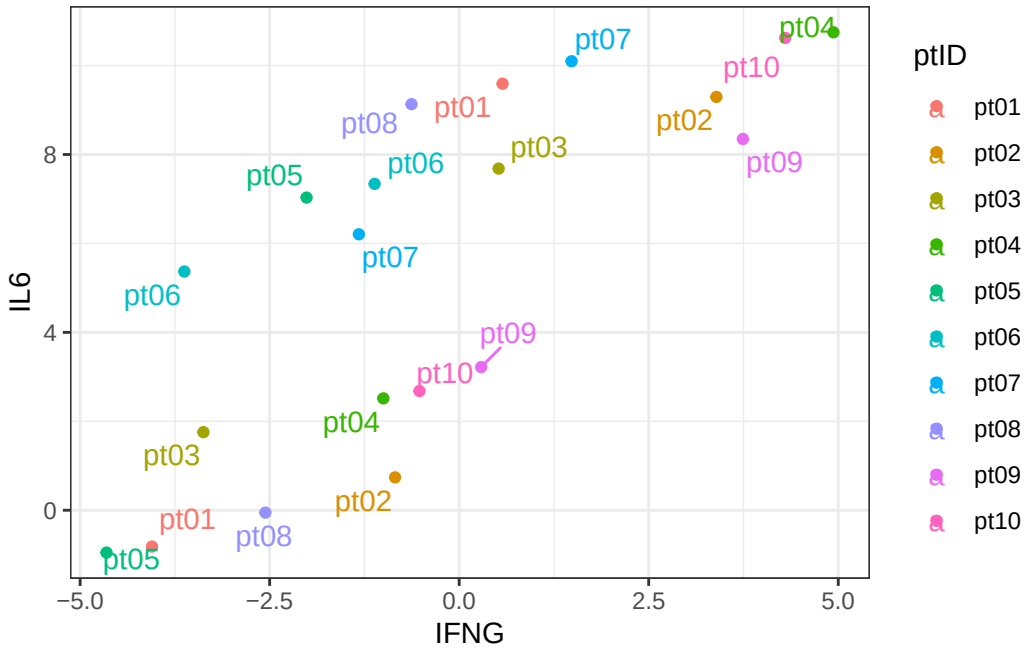
```
  #Add facets
  facet_wrap(~ptID, nrow = 2)
```



Solution 1B: Label points

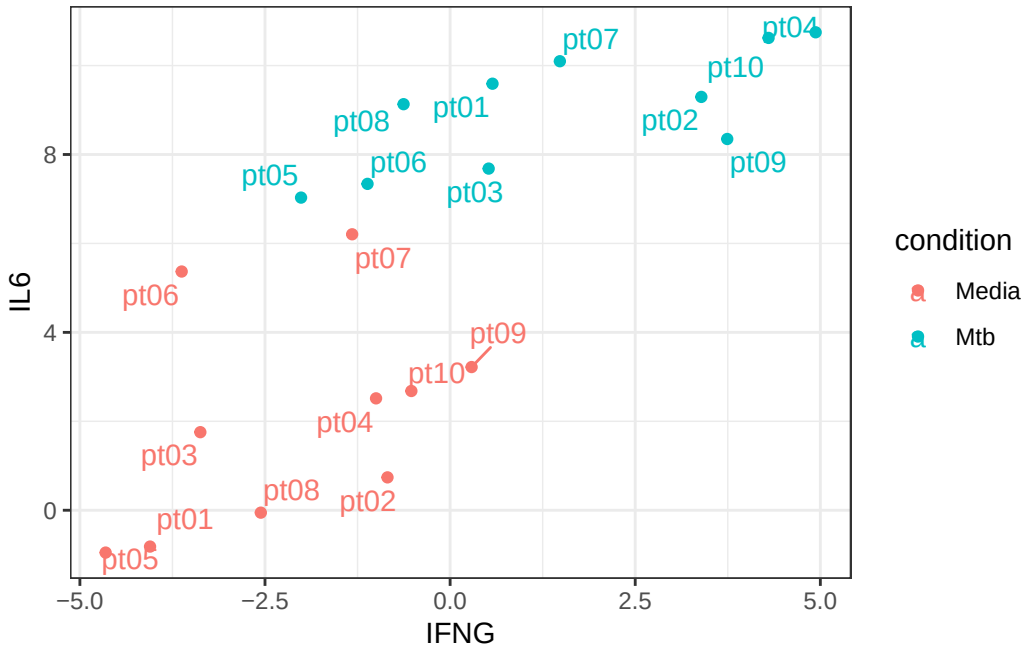
Another option is to label the points instead of coloring them. The package `ggrepel` is a great extension to `ggplot2` to achieve this goal. Now, we clearly see the patient labels and if any were overlapping, `ggrepel` nicely moves them for readability.

```
ggplot(data = dat) +
  aes(x = IFNG, y = IL6, color = ptID) +
  geom_point() +
  theme_bw() +
  #Add labels
  geom_text_repel(aes(label = ptID))
```



This also opens up the color aesthetic for a more informative plot that shows the sample condition.

```
ggplot(data = dat) +
  aes(x = IFNG, y = IL6, color = condition) + #Change color aesthetic
  geom_point() +
  theme_bw() +
  geom_text_repel(aes(label = ptID))
```



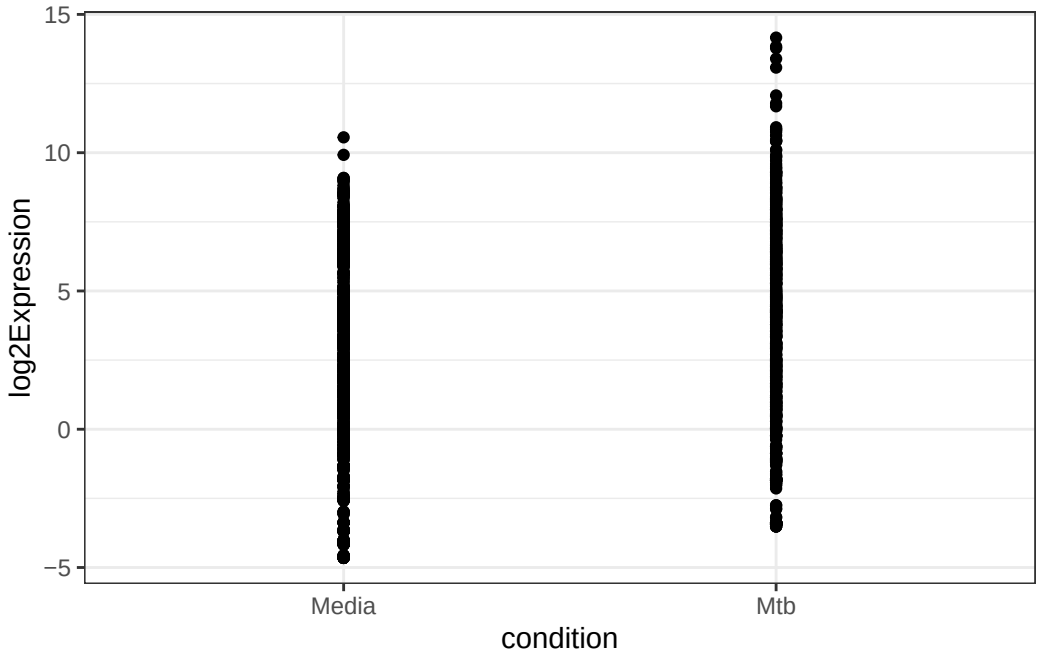
Challenge 1 summary

Thus, when faced with too many colors, consider how you can move the color variable to another layer like facets, labels, shape, size, etc. This will make that variable more interpretable as well as may allow you to use color for another informative variable with fewer levels.

Challenge 2: Too many dots!

Now let's compare the expression of interleukin (IL) genes in media and infected samples. There are 64 IL genes and 10 samples per condition. Thus, we have 640 data points per condition. When we plot this, it's basically just a smear of points. Not very helpful right?

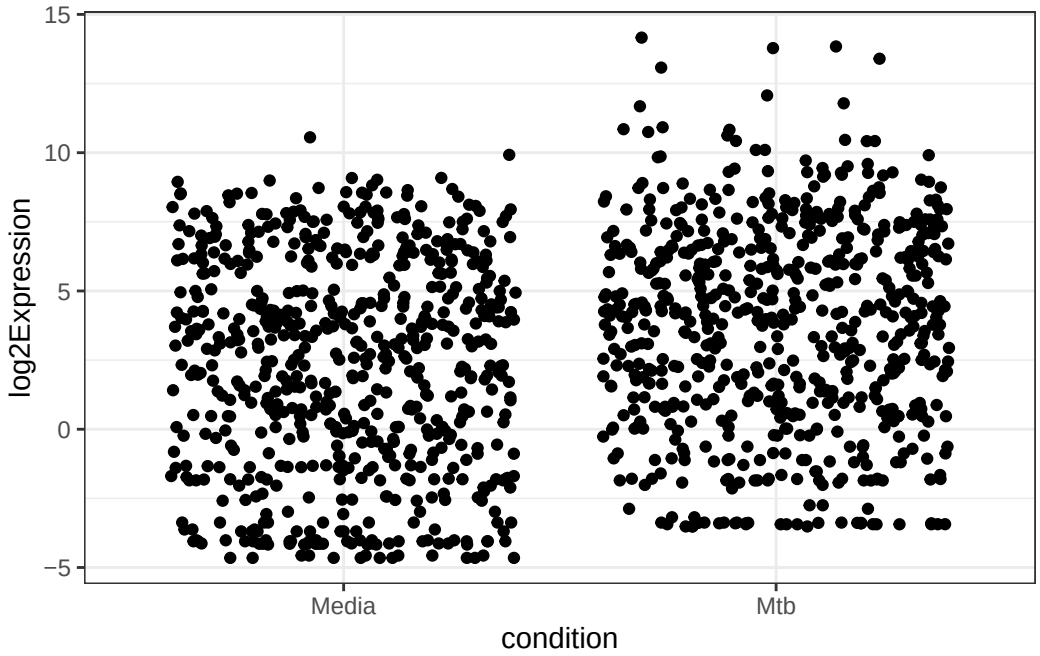
```
dat_IL <- dat %>%
  select(libID, ptID, condition, starts_with("IL")) %>%
  pivot_longer(-c(libID, ptID, condition),
               names_to = "gene", values_to = "log2Expression")

ggplot(data = dat_IL) +
  aes(x = condition, y = log2Expression) +
  geom_point() +
  theme_bw()
```



Solution 1A: Jitter

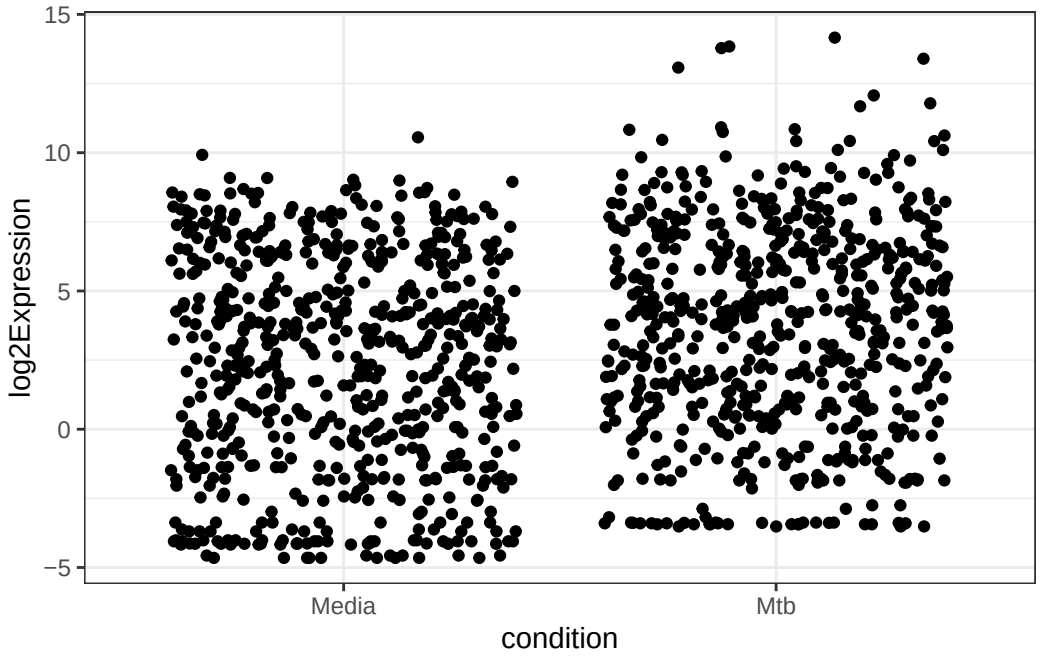
We can separate the points by randomly moving them a little bit away from each other, which is called a jitter. One way to do this is with `geom_jitter()`. Note that we specifically force the jitter to be 0 for the y-axis (height) as this is our numeric outcome and a jitter would technically alter the data we're trying to analyze.

```
ggplot(data = dat_IL) +
  aes(x = condition, y = log2Expression) +
  geom_jitter(height = 0) + #Add jitter
  theme_bw()
```



You can also achieve a jitter within `geom_point()`.

```
ggplot(data = dat_IL) +
  aes(x = condition, y = log2Expression) +
  geom_point(position = position_jitter(height = 0)) + #Add jitter
  theme_bw()
```



Help! Why doesn't my plot look exactly like yours?

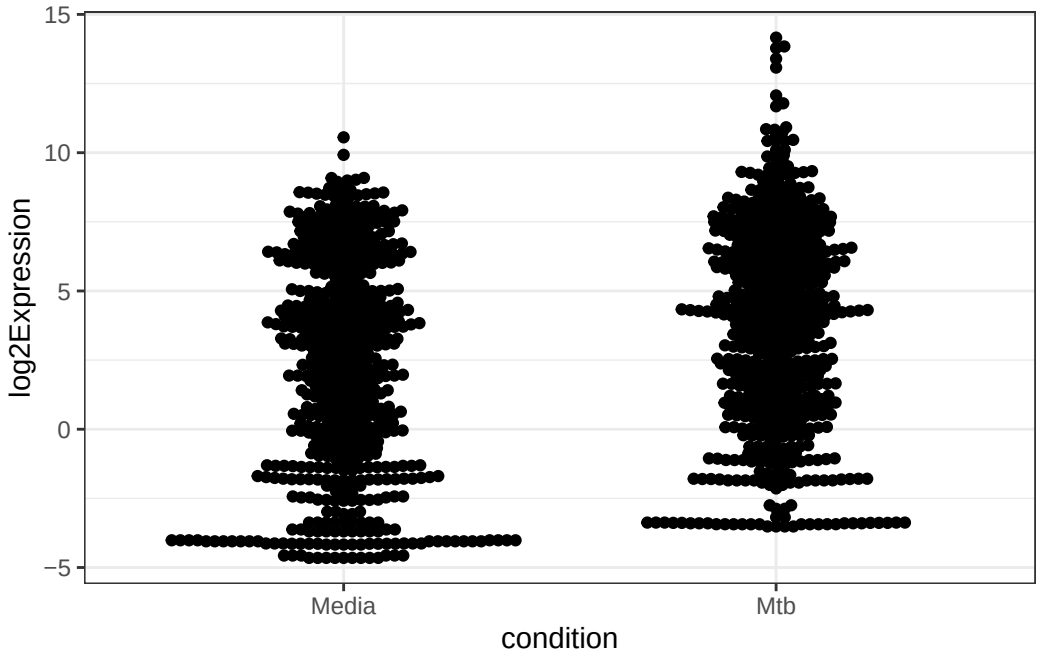
Jittering uses a random seed to determine where points fall. Every time you run a jitter, the plot will look slightly different. To prevent this, you can set the seed prior to plotting.

```
set.seed(42)
```

Solution 1B: Bee-swarm

You may have noticed that some points still overlap in the jittered plot. If you really want to see all points, you can use a bee-swarm plot instead. This comes from the `ggplot2` extension package `ggbeeswarm`.

```
ggplot(data = dat_IL) +
  aes(x = condition, y = log2Expression) +
  geom_beeswarm() + #Use bee-swarm
  theme_bw()
```

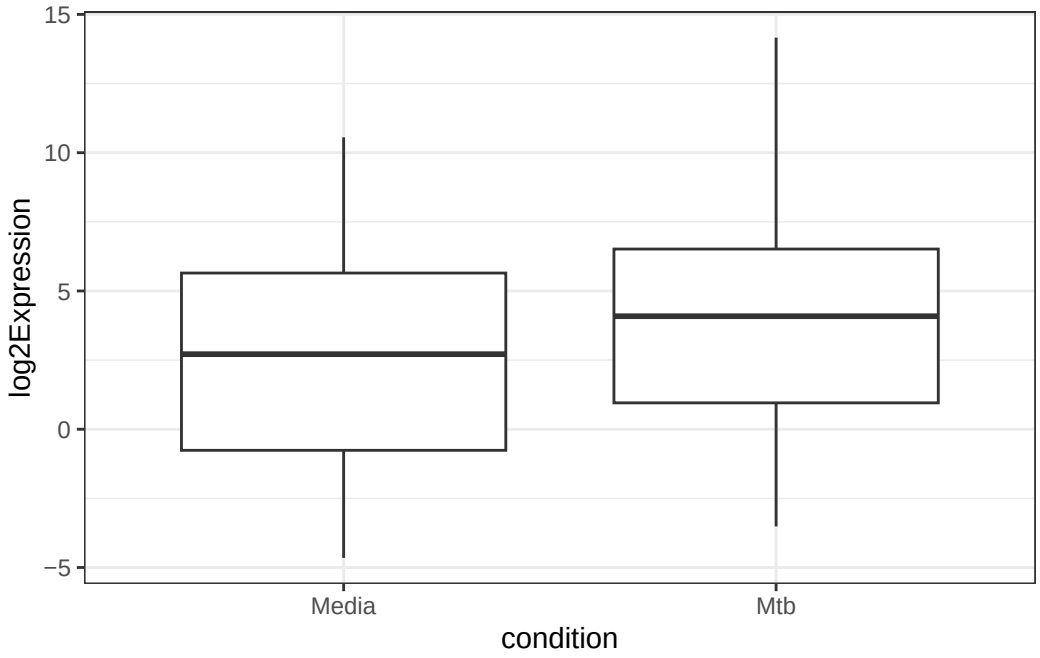


Solution 1C: No points

When you have a lot of data points, do you really need to see every point? Or is the overall trend more important? When looking at trends between groups, you have a lot of options in `ggplot2` like

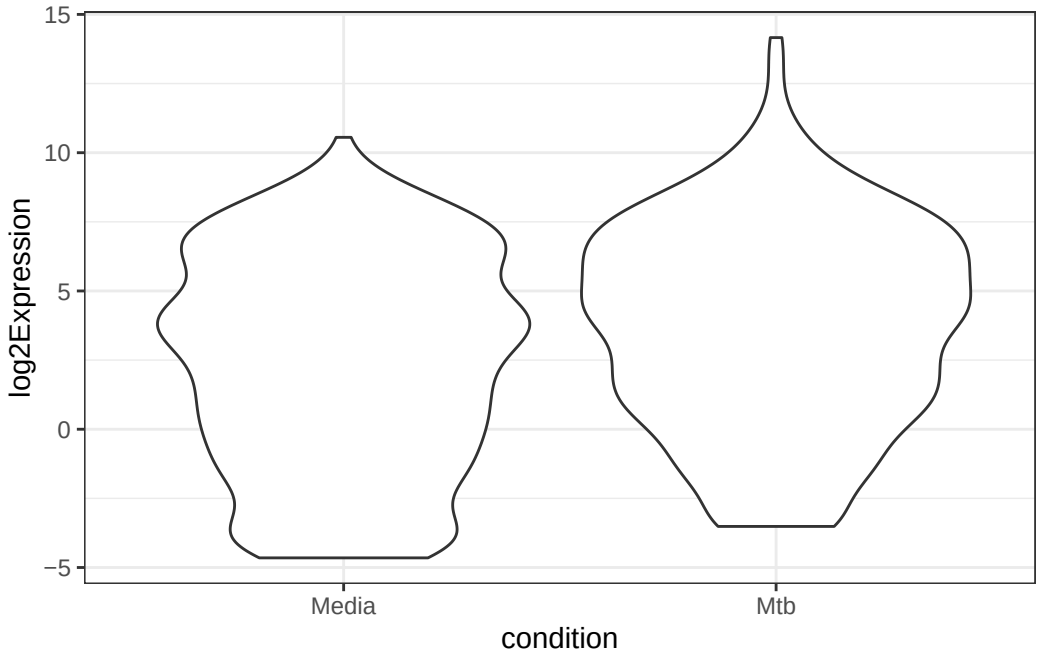
Boxplots

```
ggplot(data = dat_IL) +
  aes(x = condition, y = log2Expression) +
  geom_boxplot() + #Add boxplot
  theme_bw()
```



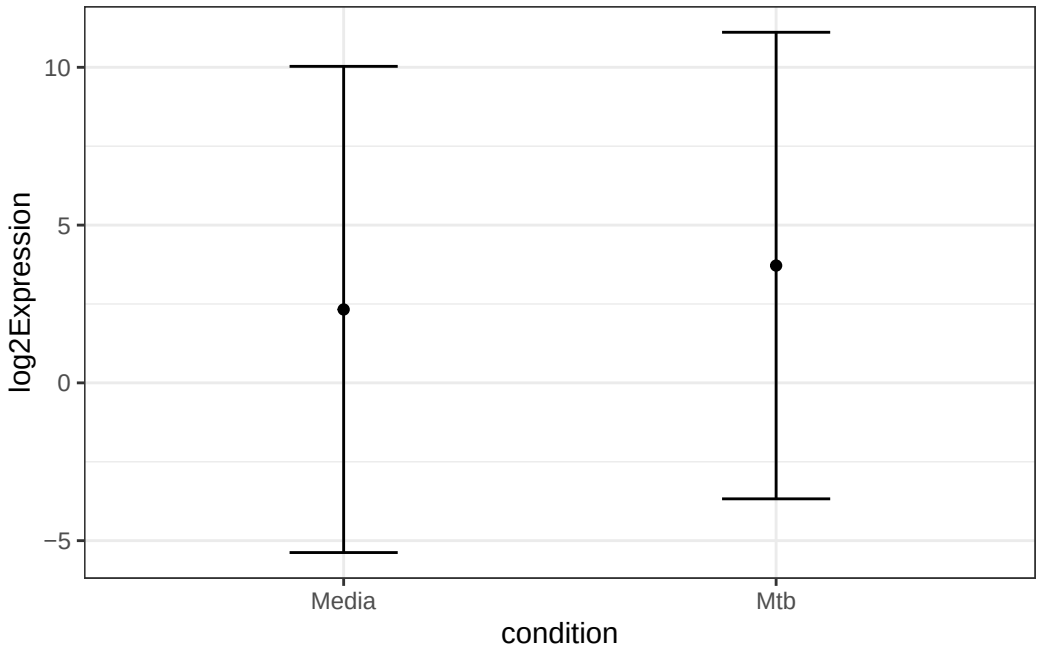
Violin plots

```
ggplot(data = dat_IL) +
  aes(x = condition, y = log2Expression) +
  geom_violin() + #Add violin
  theme_bw()
```



Means with error bars

```
ggplot(data = dat_IL) +
  aes(x = condition, y = log2Expression) +
  #Add mean
  stat_summary(fun=mean, geom="point") +
  #Add error bars
  stat_summary(fun.data=mean_sdl, geom="errorbar", width=0.25) +
  theme_bw()
```



And more!

Challenge 2 summary

When you have a lot of individual data points, you can minimize overlaps with jitter or bee-swarm methods. Also, consider if you actually need to see every data point and if not, use a summary plot type instead!

Challenge 3: Too many variables!

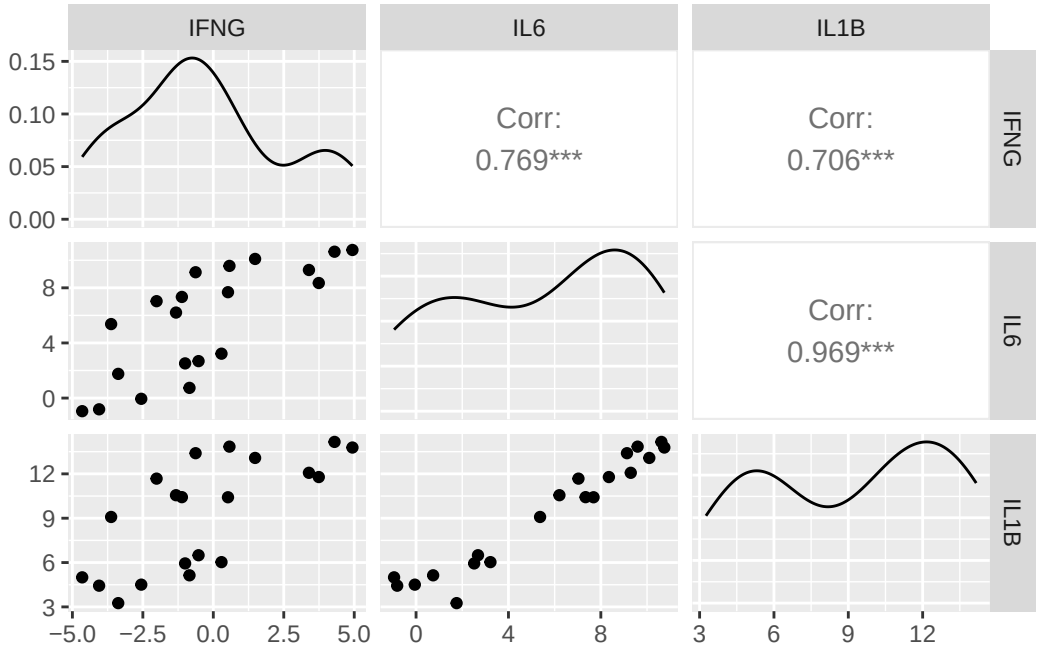
Let's say you want to compare gene expression between more than two genes but still see the individual level data. In our previous code for Challenge 1, the x and y variables are hard coded so how do you change them without having to copy-paste this code for every plot?

```
ggplot(data = dat) +
  #How do you automatically change x and y?
  aes(x = IFNG, y = IL6, color = condition) +
  geom_point() +
  theme_bw() +
  geom_text_repel(aes(label = ptID))
```

Solution 3A: Facets again!

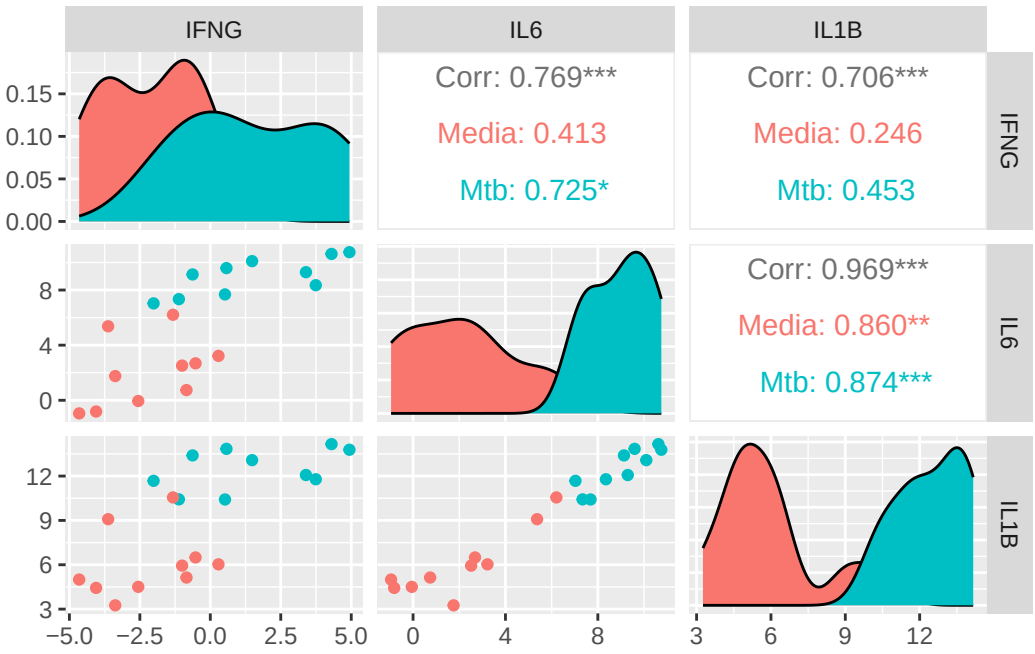
One option is to turn to facets again. There is a way to modify the current data to give us all pairwise comparisons between genes, but it's a bit of a nightmare to format. Instead, `GGally` (another `ggplot2` extension package) make this easy. You simply give it a vector of the variables you want to compare, and it plots the comparisons while also including correlation metrics.

```
ggpairs(data = dat,
        columns = c("IFNG", "IL6", "IL1B"))
```



Moreover, we can add condition as a color and `GGally` gives us correlations within each group. Sadly, `ggrepel` does not play nicely with `GGally` so we cannot easily add labels that way.

```
ggpairs(data = dat,
        columns = c("IFNG", "IL6", "IL1B"),
        ggplot2::aes(color = condition))
```



Solution 3B: Loops

You may have heard "for loop" used like a four letter word. But I'm here to tell you that loops are okay! In particular, they are a great way to make many plots that differ only slightly from each other.

Here, we use `combn()` to get our pairwise comparisons and then loop through this result to make our plots. We're also using `patchwork` to combine the plots all together.

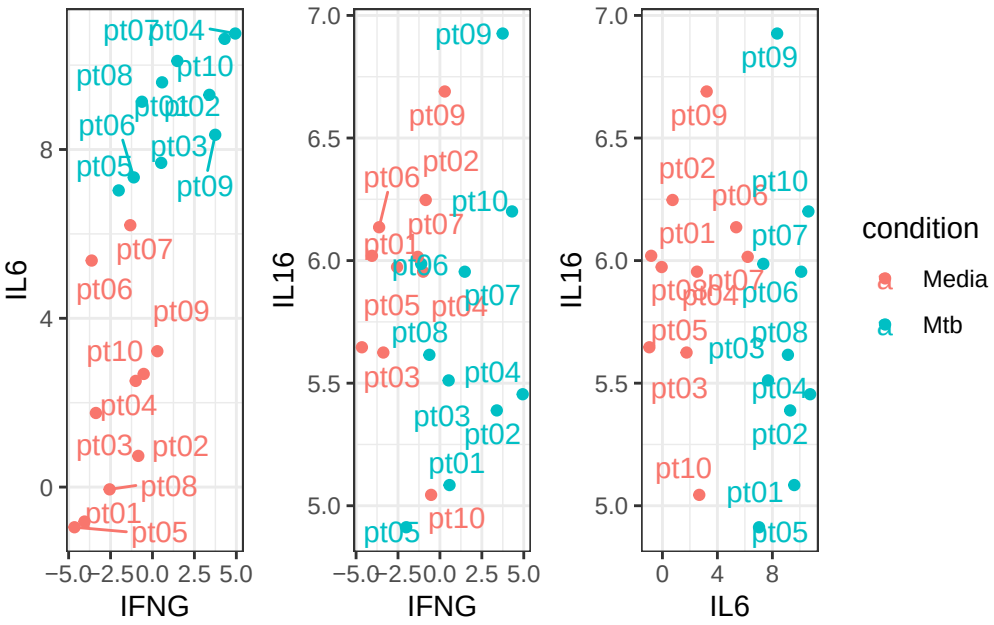
```
#Genes to plot
genes <- c("IFNG", "IL6", "IL16")
gene_pairs <- combn(genes, m = 2)
gene_pairs
```

```
     [,1]   [,2]   [,3]
[1,] "IFNG" "IFNG" "IL6"
[2,] "IL6"  "IL16" "IL16"
```

```
#List to hold plots
plot_ls <- list()

#Loop through genes for x
for(i in 1:ncol(gene_pairs)){
```

```
  #Get genes to plot
  g <- gene_pairs[1,i]
  g2 <- gene_pairs[2,i]

  #Plot
  plot_ls[[paste(g, g2, sep="_")]] <-
    ggplot(data = dat) +
    #Use .data[[]] to use character strings as variables
    aes(x = .data[[g]], y = .data[[g2]], color = condition) +
    geom_point() +
    theme_bw() +
    geom_text_repel(aes(label = ptID))
}

wrap_plots(plot_ls) + plot_layout(guides = "collect")
```

```
Warning: ggrepel: 1 unlabeled data points (too many overlaps). Consider
increasing max.overlaps
```



Help! I got a warning about unlabeled points

```
Warning message:
```

```
ggrepel: 1 unlabeled data points (too many overlaps). Consider
increasing max.overlaps
```

You might get a warning that 1 or more data points are unlabeled. This is because `ggrepel` removes some overlapping labels to prevent overcrowding your plot. You can force it to label all points no matter what by altering the `max.overlaps` parameter.

```
geom_text_repel(aes(label = ptID, max.overlaps = Inf))
```

Challenge 3 summary

When you want to make a lot of comparisons, first consider if you're interested in all pairs or just a subset. When interested in all pairs, `GGally` can quickly make your plot. When interested in a select set (or `GGally` isn't customizable enough), consider using a loop to plot only what you need. A general rule when working with plot loops is to make a single plot first to ensure it looks how you want. Then modify that code to work in a loop.

R session

```
sessionInfo()
```

```
R version 4.4.0 (2024-04-24)
Platform: aarch64-apple-darwin20
Running under: macOS Sonoma 14.5

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Los_Angeles
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base
```

```
other attached packages:
 [1] patchwork_1.2.0  GGally_2.2.1     ggbeeswarm_0.7.2 ggrepel_0.9.6
 [5] limma_3.60.3     lubridate_1.9.3  forcats_1.0.0    stringr_1.5.1
 [9] dplyr_1.1.4      purrr_1.0.2      readr_2.1.5      tidyr_1.3.1
[13] tibble_3.2.1     ggplot2_3.5.1    tidyverse_2.0.0

loaded via a namespace (and not attached):
 [1] gtable_0.3.5       beeswarm_0.4.0     xfun_0.45          htmlwidgets_1.6.4
 [5] tzdb_0.4.0         vctrs_0.6.5        tools_4.4.0        generics_0.1.3
 [9] fansi_1.0.6        cluster_2.1.6      pkgconfig_2.0.3    data.table_1.15.4
[13] checkmate_2.3.1    RColorBrewer_1.1-3 lifecycle_1.0.4    compiler_4.4.0
[17] farver_2.1.2       statmod_1.5.0      munsell_0.5.1      tinytex_0.51
[21] vipor_0.4.7        htmltools_0.5.8.1  yaml_2.3.8         htmlTable_2.4.2
[25] Formula_1.2-5      pillar_1.9.0       Hmisc_5.1-3        rpart_4.1.23
[29] ggstats_0.6.0      tidyselect_1.2.1   digest_0.6.36      stringi_1.8.4
[33] labeling_0.4.3     fastmap_1.2.0      grid_4.4.0         colorspace_2.1-1
[37] cli_3.6.3          magrittr_2.0.3     base64enc_0.1-3    utf8_1.2.4
[41] foreign_0.8-87     withr_3.0.1        backports_1.5.0    scales_1.3.0
[45] timechange_0.3.0   rmarkdown_2.27     nnet_7.3-19        gridExtra_2.3
[49] hms_1.1.3          evaluate_0.24.0    knitr_1.47         rlang_1.1.4
[53] Rcpp_1.0.13        glue_1.7.0         rstudioapi_0.16.0  jsonlite_1.8.8
[57] R6_2.5.1           plyr_1.8.9
```