Introduction to R and the tidyverse

Practice exercises - Answer key

Kim Dill-McFarland, kadm@uw.edu

version June 30, 2021

Contents

Overview **1**

Setup **1**

Exercises **2**
Day 1: Base R 2
Day 2: R tidyverse 5

R session **13**

Overview

These exercises will help you to practice tidyverse and other functions covered in the Intro R workshop including:

- Subsetting
- Filtering
- Pivoting
- Joining
- Plotting

Setup

Open the Intro R Rproject and start a new working script. Install any new packages, load packages, set a seed, and load data as we did in the workshop.

```
install.packages("broom")
```

```
library(tidyverse)
library(limma)
library(broom)
#Set seed
set.seed(4389)

#RNAseq expression and metadata
load("data/RSTR_data_clean_subset.RData")
```

Exercises

Day 1: Base R

Project setup

1. What are the benefits of storing data in `RData` versus tables (`csv`, `tsv`, etc)?

- Several tables can be stored in a single object and can be loaded together in R with a single `load()` command
- Data are automatically compressed so they take up less hard-drive space
- Data formats are preserved in R such as factors, numbers you want to treat as characters, etc

2. Imagine a hypothetical project with the following data and results. How would you choose to setup your Rproject directory and sub-directories? This is something that may evolve over time, but it is helpful to start with a defined structure to make it easier for you and others to find things.
    - `.RData` file containing all cleaned data for the project
    - 2 `.csv` of raw RNAseq counts and sample metadata (what was cleaned to make the `.RData`)
    - 4 `.csv` with linear model results
    - 25 `.png` plots of gene expression, individual genes
    - 1 `.png` plot of gene expression, faceted with many genes
    - 2 `.R` scripts, 1 for linear modeling and 1 for making plots
    - 1 `.Rmd` report summarizing and interpreting the results

There are many options for this! I would do the following. Note that I personally like to use a lot of sub-directories.

```
project_name/
    data_clean/
        .RData
    data_raw/
        2 data .csv
    figs/
        genes/
            25 individual gene plot .png
        1 facetd gene plot .png
    results/
        4 linear model .csv
    scripts/
        2 .R scripts
    .Rmd
```

Another option.

```
project_name/
    data/
        .RData
        2 data .csv
    results/
        models/
            4 linear model .csv
        figs/
            25 individual gene plot .png
            1 facetd gene plot .png
    2 .R scripts
    .Rmd
```

Data types

1. What is the difference between a `character` and `factor`?

- A character is any combination of alphanumeric (A-Z, 0-9) and other symbols (_ . - etc) that R treats like a single word. These are analogous to categorical variables in statistics. So for example, we have a variable with data on "MEDIA" vs "TB" in the workshop data set
- A factor is a character variable with some additional formatting. Factors have defined levels in a defined order. If you try to add data not of one of the defined levels, it is seen as an `NA` or missing.

For example, we could format our MEDIA/TB variable to a factor

```
factor(dat$targets$condition)
```

```
##  [1] MEDIA TB    MEDIA TB    MEDIA TB    MEDIA TB    TB    MEDIA MEDIA TB
## [13] MEDIA TB    MEDIA TB    MEDIA TB    MEDIA TB
## Levels: MEDIA TB
```

and force TB to be the first level even though it is the second alphabetically

```
factor(dat$targets$condition, levels=c("TB","MEDIA"))
```

```
##  [1] MEDIA TB    MEDIA TB    MEDIA TB    MEDIA TB    TB    MEDIA MEDIA TB
## [13] MEDIA TB    MEDIA TB    MEDIA TB    MEDIA TB
## Levels: TB MEDIA
```

And if we only allow TB as a level, we can see the how R replaces everything else with `NA`

```
factor(dat$targets$condition, levels=c("TB"))
```

```
##  [1] <NA> TB   <NA> TB   <NA> TB   <NA> TB   TB   <NA> <NA> TB   <NA> TB   <NA>
## [16] TB   <NA> TB   <NA> TB
## Levels: TB
```

2. What data type does R classify the date `2021.06`? What about `2021/06`? If it is not classified as a "date", how could this impact downstream analyses? Try to predict the outcomes before checking in R.

- Both are treated as numeric. The first as a number with 2 decimal digits and the second as the result of 2021 divided by 6

```
2021.06
```

```
## [1] 2021.06
```

```
class(2021.06)
```

```
## [1] "numeric"
```

```
2021/06
```

```
## [1] 336.8333
```

```
class(2021/06)
```

```
## [1] "numeric"
```

- This could dramatically impact results if these data were actually dates because 1) they are not being treated the same even though they are the same date, 2) scales will be wrong (as in 2021.06 is one month apart from 2021.07 but will be treated at 0.01 apart), 3) some functions that require a date won't run on a numeric, and other issues

2. Challenge: Checkout the package `lubridate` for functions to effectively work with dates in R.

- You can force date formatting like so and it is a lot more intuitive that base R's `as.Date()`. Here, you simply list which date components you have for year (y), month (m), and day (d) in the order they are in. That function does the rest!

```
library(lubridate)

ym("2021.06")
```

```
## [1] "2021-06-01"
```

```
ym("2021/06")
```

```
## [1] "2021-06-01"
```

```
#And if we had a day
ymd("2021.06.2")
```

```
## [1] "2021-06-02"
```

```
#A different order
mdy("06.2.2021")
```

```
## [1] "2021-06-02"
```

- https://lubridate.tidyverse.org/ has more on the `lubridate` package

3. You have an `S3` list object named `myData`and it contains 2 data frames named `A` and `B`. Within `B` there is a column named `variable1`. How do you access this variable?

```
myData$B$variable1
```

Subsetting and filtering

Using `dat`:

1. What is the mean library size `lib.size`?

```
mean(dat$targets$lib.size)
```

```
## [1] 9287558
```

2. Try running `summary(dat$targets)`. What kinds of data does it provide? Why are the results different for different variables?

```
summary(dat$targets)
```

```
##     libID              lib.size         norm.factors      FULLIDNO
##  Length:20          Min.   : 2776897   Min.   :0.7735   Length:20
##  Class :character   1st Qu.: 4625309   1st Qu.:0.8871   Class :character
##  Mode  :character   Median :10186231   Median :1.0224   Mode  :character
##                     Mean   : 9287558   Mean   :1.0082
##                     3rd Qu.:12526319   3rd Qu.:1.0903
##                     Max.   :17242699   Max.   :1.3324
##      RSID            condition
##  Length:20          Length:20
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

- You will get min, max, mean, and quartiles for numeric data

- You get the class and length of character vectors
- R automatically detects the data type and provides as much info as it can. Since character variables are simply words, this class has the least summary info.

3. How many libraries have a library size `lib.size` greater than 5 million and a normalization factor `norm.factors` less than 1?

```
size.logical <- dat$targets$lib.size > 5E6
norm.logical <- dat$targets$norm.factors < 1

dat$targets[size.logical & norm.logical, ]
```

```
##          libID lib.size norm.factors   FULLIDNO     RSID condition
## 4  RS102306_TB 10101705    0.9096415 84437-1-02 RS102306        TB
## 8  RS102244_TB  8695434    0.8544133 84457-1-02 RS102244        TB
## 9  RS102521_TB  8859096    0.7735422 91360-1-04 RS102521        TB
## 12 RS102340_TB  7368365    0.8892822 84317-1-03 RS102340        TB
## 18 RS102469_TB 10536450    0.9812365 89448-1-04 RS102469        TB
## 20 RS102484_TB 10270757    0.8390154 84427-1-02 RS102484        TB
```

```
#Or combine it all together
dat$targets[dat$targets$lib.size > 5E6 & dat$targets$norm.factors < 1, ]
```

```
##          libID lib.size norm.factors   FULLIDNO     RSID condition
## 4  RS102306_TB 10101705    0.9096415 84437-1-02 RS102306        TB
## 8  RS102244_TB  8695434    0.8544133 84457-1-02 RS102244        TB
## 9  RS102521_TB  8859096    0.7735422 91360-1-04 RS102521        TB
## 12 RS102340_TB  7368365    0.8892822 84317-1-03 RS102340        TB
## 18 RS102469_TB 10536450    0.9812365 89448-1-04 RS102469        TB
## 20 RS102484_TB 10270757    0.8390154 84427-1-02 RS102484        TB
```

```
#And bonus, you can make R count the rows for you
nrow(dat$targets[size.logical & norm.logical, ])
```

```
## [1] 6
```

4. Challenge: Using the function `grepl`, how many libraries are from a donor with an `RSID` that starts with "RS1025"?

```
dat$targets[grepl("^RS1025", dat$targets$RSID), ]
```

```
##             libID lib.size norm.factors   FULLIDNO     RSID condition
## 2     RS102531_TB  2776897    0.9161955 92527-1-08 RS102531        TB
## 3  RS102531_MEDIA  4258343    1.0732788 92527-1-08 RS102531     MEDIA
## 9     RS102521_TB  8859096    0.7735422 91360-1-04 RS102521        TB
## 11 RS102521_MEDIA 14509963    1.0349902 91360-1-04 RS102521     MEDIA
## 14    RS102548_TB 12858053    1.2734417 89902-1-07 RS102548        TB
## 15 RS102548_MEDIA 17242699    1.3323830 89902-1-07 RS102548     MEDIA
```

- Note that `^` means the start of and `$` means the end of in a regular expression (regex) used in `grepl`

Day 2: R tidyverse

Subsetting and filtering

Subset the `dat$targets` data frame to the rows and/or columns you need to answer the following questions. For several questions, you can just filter rows and look for the answer. However, many of our real data sets are too wide (too many variables) to do this, so please practice also selecting the column with the solution.

1. How many MEDIA samples are there?

```
dat$targets %>%
  filter(condition == "MEDIA") %>%
  nrow()
```

```
## [1] 10
```

2. How many sequences (`lib.size`) does 89448-1-04 have in their TB sample?

```
dat$targets %>%
  filter(FULLIDNO =="89448-1-04" & condition == "TB") %>%
  select(lib.size)
```

```
##   lib.size
## 1 10536450
```

3. What are the maximum and minimum normalization factors (`norm.factors`) for MEDIA samples?

```
dat$targets %>%
  filter(condition == "MEDIA") %>%
  select(norm.factors) %>%
  max()
```

```
## [1] 1.332383
```

```
dat$targets %>%
  filter(condition == "MEDIA") %>%
  select(norm.factors) %>%
  min()
```

```
## [1] 1.021158
```

4. Challenge: Checkout `group_by` with `summarise` to see how you can get summary statistics for multiple groups at once. For example, #1 and 3 can be calculated for MEDIA and TB simultaneously.

```
dat$targets %>%
  group_by(condition) %>%
  summarise(min.norm = min(norm.factors),
            max.norm = max(norm.factors))
```

```
## # A tibble: 2 x 3
##   condition min.norm max.norm
##   <chr>        <dbl>    <dbl>
## 1 MEDIA        1.02      1.33
## 2 TB           0.774     1.27
```

Pivoting

1. Without running the following code, sketch the resulting data frame structures. Once you've done this, check the outputs in R.

```
dat$targets %>%
  select(RSID, condition, lib.size) %>%
  pivot_wider(names_from=RSID, values_from=lib.size)
```

```
## # A tibble: 2 x 11
##   condition RS102106 RS102531  RS102306 RS102087  RS102244  RS102521  RS102340
##   <chr>        <dbl>    <dbl>     <dbl>    <dbl>     <dbl>     <dbl>     <dbl>
## 1 MEDIA     4575023. 4258343. 12415741. 4642072. 10744984. 14509963. 10601433.
## 2 TB        2935653. 2776897. 10101705. 3571641.  8695434.  8859096.  7368365.
## # ... with 3 more variables: RS102548 <dbl>, RS102469 <dbl>, RS102484 <dbl>
```

```
dat$targets %>%
  pivot_longer(c(lib.size, norm.factors))
```

```
## # A tibble: 40 x 6
##    libID          FULLIDNO   RSID     condition name                value
##    <chr>          <chr>      <chr>    <chr>     <chr>               <dbl>
##  1 RS102106_MEDIA 92527-1-02 RS102106 MEDIA     lib.size      4575023.
##  2 RS102106_MEDIA 92527-1-02 RS102106 MEDIA     norm.factors        1.15
##  3 RS102531_TB    92527-1-08 RS102531 TB        lib.size      2776897.
##  4 RS102531_TB    92527-1-08 RS102531 TB        norm.factors        0.916
##  5 RS102531_MEDIA 92527-1-08 RS102531 MEDIA     lib.size      4258343.
##  6 RS102531_MEDIA 92527-1-08 RS102531 MEDIA     norm.factors        1.07
##  7 RS102306_TB    84437-1-02 RS102306 TB        lib.size     10101705.
##  8 RS102306_TB    84437-1-02 RS102306 TB        norm.factors        0.910
##  9 RS102306_MEDIA 84437-1-02 RS102306 MEDIA     lib.size     12415741.
## 10 RS102306_MEDIA 84437-1-02 RS102306 MEDIA     norm.factors        1.02
## # ... with 30 more rows
```

Joining

1. Using the sample information in `dat$targets` and a join function, relabel the samples in the expression data `dat$E` with `FULLIDNO` and `condition`. Note: You'll also need to employ a pivot!

```
as.data.frame(dat$E) %>%
  #Move rownames into a data column named geneName
  rownames_to_column("geneName") %>%
  #move expression data into 1 column for libID and 1 for all the values
  pivot_longer(-geneName, names_to = "libID") %>%
  #Join with metadata
  inner_join(dat$targets)  %>%
  #Make a new variable to rename the columns with.
  ##By calling it "name", our later pivot with automatically recognize it
  mutate(name = paste(FULLIDNO, condition, sep="_")) %>%
  #Keep only the data we want
  select(geneName, name, value) %>%
  #Return to wider format
  pivot_wider()
```

```
## Joining, by = "libID"
```

```
## # A tibble: 14,576 x 21
##    geneName   `92527-1-02_MEDI~ `92527-1-08_TB` `92527-1-08_MED~ `84437-1-02_TB`
##    <chr>                  <dbl>           <dbl>            <dbl>           <dbl>
##  1 ENSG00000~             4.90             4.91            4.75            5.15
##  2 ENSG00000~             4.01             4.24            4.28            4.05
##  3 ENSG00000~             3.07             2.81            2.64            2.96
##  4 ENSG00000~             6.52             8.54            8.23            8.80
##  5 ENSG00000~            -3.19             1.43           -1.51            1.59
##  6 ENSG00000~             6.20             4.49            5.50            5.49
##  7 ENSG00000~             6.69             3.87            5.02            7.04
##  8 ENSG00000~             5.70             6.80            5.94            6.65
##  9 ENSG00000~             0.266            1.92            0.369           0.618
## 10 ENSG00000~             3.38             3.36            2.84            3.36
## # ... with 14,566 more rows, and 16 more variables: 84437-1-02_MEDIA <dbl>,
## #   91587-1-04_TB <dbl>, 91587-1-04_MEDIA <dbl>, 84457-1-02_TB <dbl>,
```

```
## #   91360-1-04_TB <dbl>, 84457-1-02_MEDIA <dbl>, 91360-1-04_MEDIA <dbl>,
## #   84317-1-03_TB <dbl>, 84317-1-03_MEDIA <dbl>, 89902-1-07_TB <dbl>,
## #   89902-1-07_MEDIA <dbl>, 92527-1-02_TB <dbl>, 89448-1-04_MEDIA <dbl>,
## #   89448-1-04_TB <dbl>, 84427-1-02_MEDIA <dbl>, 84427-1-02_TB <dbl>
```

2. Use the following code to create two new data frames.

```
df1 <- data.frame(donor = c("A","B","C"),
                  age = c(10,14,4))

df2 <- data.frame(donor = c("A","C","D"),
                  sex = c("F","M","F"))
```

Then sketch what the resulting data frames would be from the following join functions before running them in R.

```
left_join(df1, df2, by = "donor")
```

```
##   donor age  sex
## 1     A  10    F
## 2     B  14 <NA>
## 3     C   4    M
```

```
right_join(df1, df2, by = "donor")
```

```
##   donor age sex
## 1     A  10   F
## 2     C   4   M
## 3     D  NA   F
```

```
full_join(df1, df2, by = "donor")
```

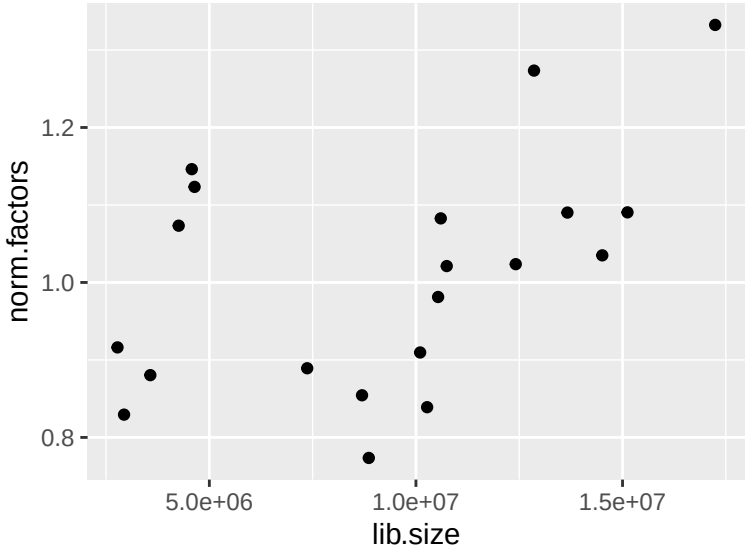
```
##   donor age  sex
## 1     A  10    F
## 2     B  14 <NA>
## 3     C   4    M
## 4     D  NA    F
```

Plotting

Using `dat$targets`,

1. Create a dot plot of library size (`lib.size`) by normalization factor (`norm.factors`)

```
dat$targets %>%
  ggplot(aes(x=lib.size, y=norm.factors)) +
  geom_point()
```

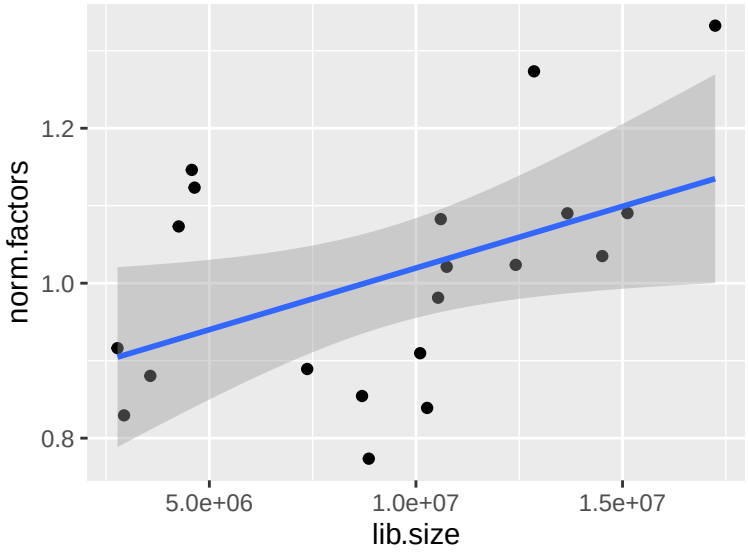


2. Does there appear to be a linear relationship between library size and normalization?

a. Add a linear fit to your plot with the new function `geom_smooth`. Remember to use help `?` when learning new functions.

```
dat$targets %>%
  ggplot(aes(x=lib.size, y=norm.factors)) +
  geom_point() +
  geom_smooth(method="lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



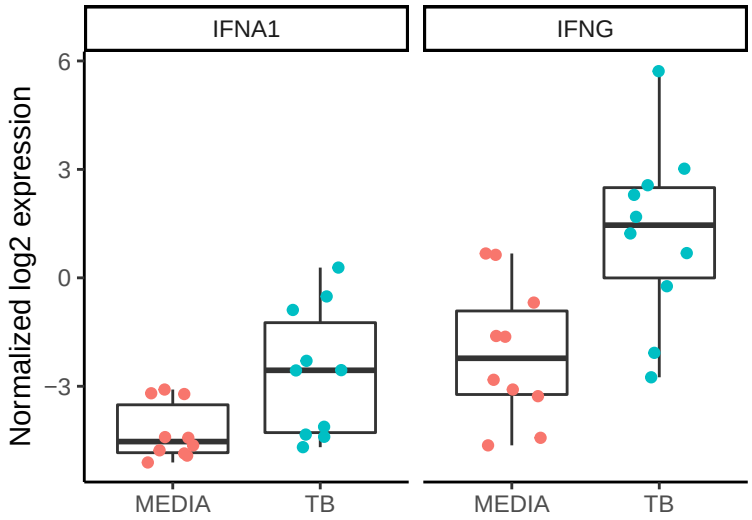
b. Run the following code to assess if the linear fit is significant. This is an example of how the tidyverse can be piped directly into statistical models in base R. The `.` in our linear model `lm` is used as a place holder for the data frame piped in.

```
dat$targets %>%
  lm(norm.factors ~ lib.size , data=.) %>%
  tidy()
```

```
## # A tibble: 2 x 5
```

```
##   term            estimate     std.error statistic  p.value
##   <chr>              <dbl>         <dbl>     <dbl>    <dbl>
## 1 (Intercept) 0.860        0.0726            11.9  6.17e-10
## 2 lib.size    0.0000000159 0.00000000711      2.24 3.80e- 2
```

3. There are many way to represent data in plots, especially when you have more than a single x-y comparison. We saw one such way with facets, where we created multiple x-y comparisons in multiple plots. Another way to do this is to add additional layers to a single plot, like color or shape. Here, we see the faceted plot from the workshop.



To make an alternate plot with shapes for gene (`hgnc_symbol`) and colors for donor (`RSID`), modify the following code by filling in places marked with `[ SOMETHING ]`. For simplicity, you can use the following code from the workshop to make `dat.format` to use in plotting. The goal plot is also shown below.

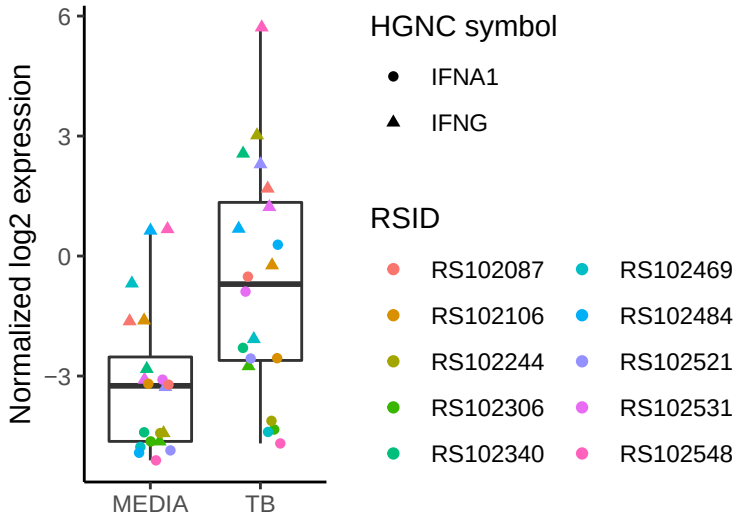
```
dat.format <- as.data.frame(dat$E) %>%
  #Move rownames to a column. Unlike before, let's name it
  #to match what we'll be joining with next
  rownames_to_column("geneName") %>%
  #Join with gene key to get HGNC symbol
  inner_join(dat$genes) %>%
  #filter IFNG and TNF expression
  filter(hgnc_symbol %in% c("IFNG", "IFNA1")) %>%
  #Pivot to long format so can combine with metadata
  pivot_longer(-c(geneName, hgnc_symbol:locus_group),
               names_to="libID", values_to="expression") %>%
  #join with library metadata
  #Notice that you can nest function so that you never need to save the subset metadata
  inner_join(select(.data = dat$targets, libID, FULLIDNO, RSID, condition)) %>%
  #select variables of interest. Note we need to keep hgnc_symbol to tell
  #data from the two genes apart
  select(libID, hgnc_symbol, expression, FULLIDNO, RSID, condition)
```

```
## Joining, by = "geneName"
```

```
## Joining, by = "libID"
```

```
dat.format %>%
  ggplot(aes(x=condition, y=expression)) +
  #Create boxplots
  ## Set the outlier shape to NULL so that you don't get duplicate
```

```
  ## dots in the next layer.
  geom_boxplot(outlier.shape=NA) +
  #Add points for each sample.
  ## Color by donor, shape by gene
  ## "jitter" left and right (width) to avoid overlap
  geom_jitter(aes(shape=hgnc_symbol, color=RSID), width = 0.2, height=0) +
  #Format axis labels
  labs(y="Normalized log2 expression", x="",
       shape="HGNC symbol") +
  #Change theme to classic to remove grey background and other
  #default aspects of ggplot
  theme_classic() +
  #Make legend 2 columns
  guides(color = guide_legend(ncol = 2))
```



4. How interpretable is the alternate plot? Do you prefer the alternate or the original faceted plot?

- I find the alternate plot difficult to interpret because some of the colors are very close and the trends between circles vs triangles are overlapping.

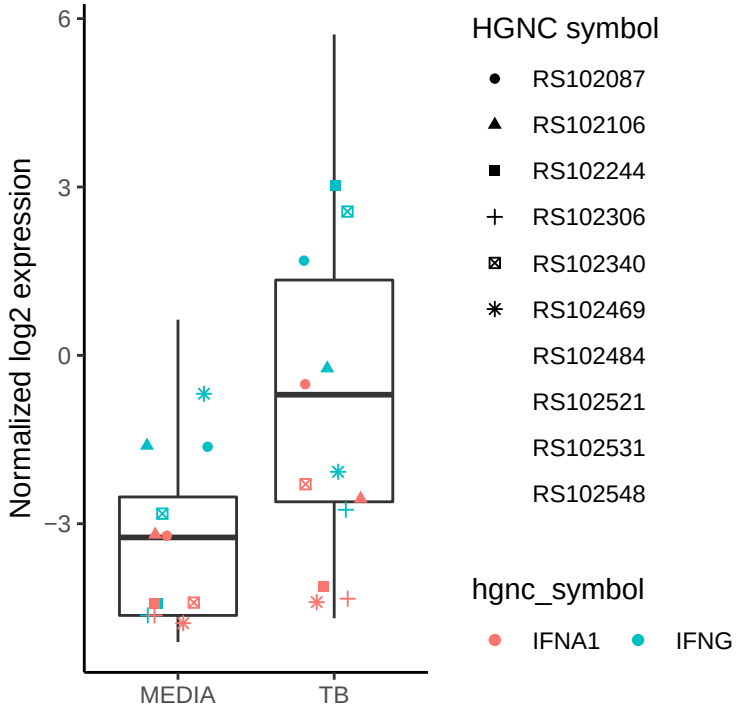
What happens if you switch the shape and color variables?

```
dat.format %>%
  ggplot(aes(x=condition, y=expression)) +
  #Create boxplots
  ## Set the outlier shape to NULL so that you don't get duplicate
  ## dots in the next layer.
  geom_boxplot(outlier.shape=NA) +
  #Add points for each sample.
  ## Color by donor, shape by gene
  ## "jitter" left and right (width) to avoid overlap
  geom_jitter(aes(color=hgnc_symbol, shape=RSID), width = 0.2, height=0) +
  #Format axis labels
  labs(y="Normalized log2 expression", x="",
       shape="HGNC symbol") +
  #Change theme to classic to remove grey background and other
  #default aspects of ggplot
  theme_classic() +
  #Make legend 2 columns
```

```
  guides(color = guide_legend(ncol = 2))
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 10. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 16 rows containing missing values (geom_point).
```



Note that R has a limit on the number of shapes it will allow. If you go beyond this, it will remove data without an assigned shape. You could force more by defining them in `scale_shape_manual` but this is not recommended. The 6 default shapes are a good limit on how many can be differentiated by eye. And even that is pushing it a little.

However, you may notice that the use of color for the two genes makes it much easier to see trends like IFNG is more highly expressed than IFNA1.

These questions bring up some important considerations in figure generation, particularly for complex data. Making a good figure is not too difficult but making an excellent, compelling figure is extremely challenging. So, here are some guidelines to help.

Within an image, humans generally perceive aspects in a defined and consistent order: outline, color, shape, size. Thus, you should consider the importance of each data feature and align that with what aspect of the plot you use.

- outline: In ggplot, this is best accomplished with facets though you can consider white space is general as well. When the brain sees a box or white space around a subsection of a plot, it automatically assumes everything in the box is related and processes it accordingly. This is the first perceived aspect, because in the real world, it differentiates one object from another.
- color: You may have noticed in the above plot that the shapes get lost behind all the different colors. This is because we tend to see color first. A lot of this has to due with our emotional and cultural connections to different colors. So if you have a single plot and want to highlight 1 variable, use color. Also consider your color choices including
  - Are they color-blind safe?

  - Can you tell all discrete colors apart?
  - Do you need to color everything differently or use a single highlight color?
  - Do they match cultural assumptions like high values being "hot" and low being "cool"?
  - Are they appealing? Do you have an emotional response to them?

- shape: Shape perception actually has a lot more to due with outline, color, and size than the shape itself. For example, consider an open circle, closed circle, and closed triangle. You can more easily tell the open and closed circles apart than the closed circle and triangle. This is because open vs closed is actually a difference in color! Similarly, a closed circle versus a plus sign are easy to differentiate because their outlines are so different. So, in choosing shapes, consider how different they are and try to limit the number used.
- size: Large differences in size are easy to discern but subtle differences are difficult. Arguably if there are large differences, color or shape should be used with large, medium, and small groups. Small differences are likely better as a color gradient. If you must use size, uniform shapes are more easily compared (circle, square, diamond) than irregular (triangle, plus).

R session

```
sessionInfo()
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS  10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/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
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] lubridate_1.7.10 broom_0.7.6      limma_3.44.3     forcats_0.5.1
##  [5] stringr_1.4.0    dplyr_1.0.6      purrr_0.3.4      readr_1.4.0
##  [9] tidyr_1.1.3      tibble_3.1.2     ggplot2_3.3.3    tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.1.1  xfun_0.23         lattice_0.20-44   splines_4.0.2
##  [5] haven_2.4.1       colorspace_2.0-1  vctrs_0.3.8       generics_0.1.0
##  [9] htmltools_0.5.1.1 mgcv_1.8-35       yaml_2.2.1        utf8_1.2.1
## [13] rlang_0.4.11      pillar_1.6.1      glue_1.4.2        withr_2.4.2
## [17] DBI_1.1.1         dbplyr_2.1.1      modelr_0.1.8      readxl_1.3.1
## [21] lifecycle_1.0.0   munsell_0.5.0     gtable_0.3.0      cellranger_1.1.0
## [25] rvest_1.0.0       evaluate_0.14     labeling_0.4.2    knitr_1.33
## [29] fansi_0.4.2       highr_0.9         Rcpp_1.0.6        scales_1.1.1
## [33] backports_1.2.1   jsonlite_1.7.2    farver_2.1.0      fs_1.5.0
## [37] hms_1.1.0         digest_0.6.27     stringi_1.6.2     grid_4.0.2
## [41] cli_2.5.0         tools_4.0.2       magrittr_2.0.1    crayon_1.4.1
## [45] pkgconfig_2.0.3   Matrix_1.3-3      ellipsis_0.3.2    xml2_1.3.2
## [49] reprex_2.0.0      assertthat_0.2.1  rmarkdown_2.8     httr_1.4.2
```

```
## [53] rstudioapi_0.13   R6_2.5.0          nlme_3.1-152      compiler_4.0.2
```