Introduction to R and the tidyverse

Practice exercises

Kim Dill-McFarland, kadm@uw.edu

version June 21, 2021

Contents

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)?
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

Data types

1. What is the difference between a `character` and `facter`?
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.
   - Challenge: Checkout the package `lubridate` for functions to effectively work with dates in R.
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?

Subsetting and filtering

Using `dat`:

1. What is the mean library size `lib.size`?
2. Try running `summary(dat$targets)`. What kinds of data does it provide? Why are the results different for different variables?
3. How many libraries have a library size `lib.size` greater than 5 million and a normalization factor `norm.factors` less than 1?
4. Challenge: Using the function `grepl`, how many libraries are from a donor with an `RSID` that starts with "RS1025"?

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?
2. How many sequences (`lib.size`) does 89448-1-04 have in their TB sample?
3. What are the maximum and minimum normalization factors (`norm.factors`) for MEDIA samples?
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.

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)
```

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

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!
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")

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

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

Plotting

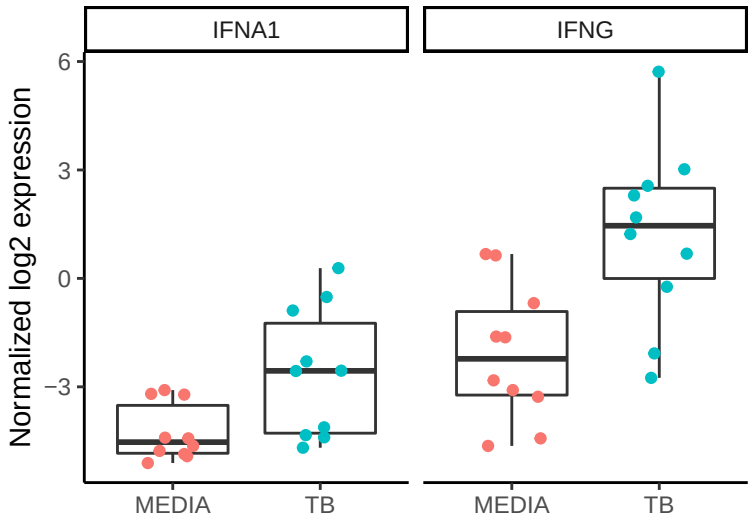
Using `dat$targets`,

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

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.

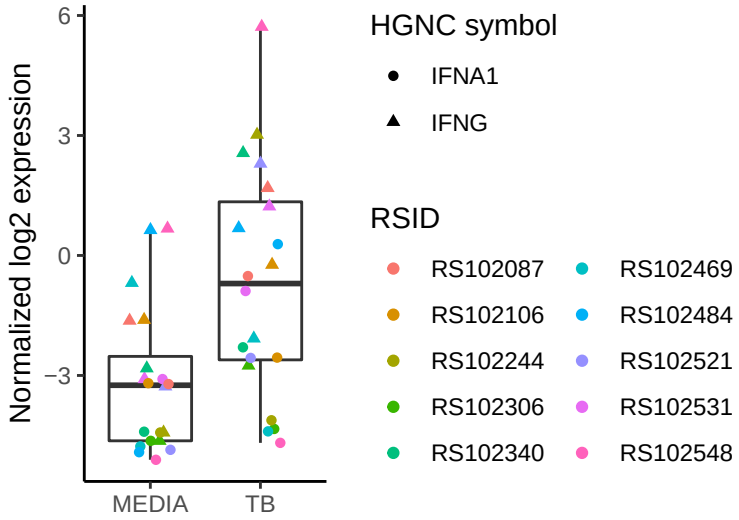


To make an alternate plot with shapes for gene (`hgnc_symbol`) and colors for donor (`RSID`), modify the following code by filling in places marked with `[ SOMETHING ]`. For simplicity, you can use the following code from the workshop to make `dat.format` to use in plotting. The goal plot is also shown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

```
dat.format %>%
  ggplot(aes(x=[ SOMETHING ], 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=[ SOMETHING ], color=[ SOMETHING ]), 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? What happens if you switch the shape and color variables? 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] broom_0.7.6     limma_3.44.3    forcats_0.5.1   stringr_1.4.0
##  [5] dplyr_1.0.6     purrr_0.3.4     readr_1.4.0     tidyr_1.1.3
##  [9] 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         haven_2.4.1       colorspace_2.0-1
##  [5] vctrs_0.3.8       generics_0.1.0    htmltools_0.5.1.1 yaml_2.2.1
##  [9] utf8_1.2.1        rlang_0.4.11      pillar_1.6.1      glue_1.4.2
## [13] withr_2.4.2       DBI_1.1.1         dbplyr_2.1.1      modelr_0.1.8
## [17] readxl_1.3.1      lifecycle_1.0.0   munsell_0.5.0     gtable_0.3.0
## [21] cellranger_1.1.0  rvest_1.0.0       evaluate_0.14     labeling_0.4.2
## [25] knitr_1.33        fansi_0.4.2       highr_0.9         Rcpp_1.0.6
## [29] scales_1.1.1      backports_1.2.1   jsonlite_1.7.2    farver_2.1.0
## [33] fs_1.5.0          hms_1.1.0         digest_0.6.27     stringi_1.6.2
## [37] grid_4.0.2        cli_2.5.0         tools_4.0.2       magrittr_2.0.1
## [41] crayon_1.4.1      pkgconfig_2.0.3   ellipsis_0.3.2    xml2_1.3.2
## [45] reprex_2.0.0      lubridate_1.7.10  assertthat_0.2.1  rmarkdown_2.8
## [49] httr_1.4.2        rstudioapi_0.13   R6_2.5.0          compiler_4.0.2
```