BTEP Lessons

Alexandra L Emmons Ph.D. & Joe Wu Ph.D.
BTEP/GAU/CCR/NCI/NIH - email ncibtep@mail.nih.gov
Bioinformatics Training and Education Program

Table of Contents

Welcome

Data Visualization with ggplot2

Welcome

These pages include associated documentation from 2024 BTEP training events that were not a part of larger courses.

Events

June 11, 2024: Data Visualization with ggplot2

Data Visualization with ggplot2

Learning Objectives

1. Understand the ggplot2 syntax.
2. Learn the grammar of graphics for plot construction.
3. Create simple, pretty, and effective figures.

What is R?

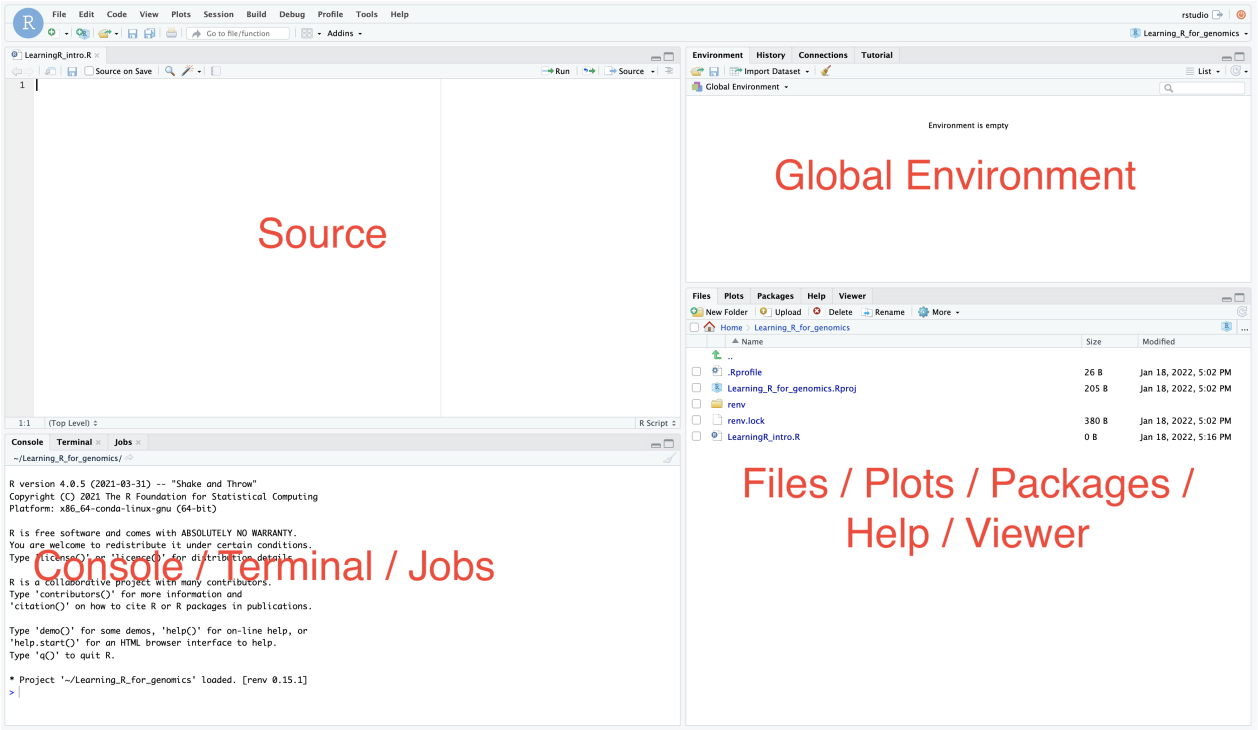
R is a computational language and environment for statistical computing and graphics.

Advantages of R programming:

- open-source
- extensible (Packages on CRAN (> 19,000 packages), Github, Bioconductor)
- Wide community
- allows reproducibility (R scripts, Rmarkdown, Quarto).
- includes fantastic options for data viz (base R, ggplot2, lattice, plotly)

RStudio

An integrated development environment (IDE) for R, and now python. RStudio includes a console, editor, and tools for plotting, history, debugging, and work space management.



What is ggplot2?

An R graphics package from the tidyverse collection, which are popular packages for data science that work really well with data organized in data frames (or tibbles *(https://tibble.tidyverse.org/)*).

Why ggplot2?

- Widespread popularity.
- Used to create informative plots quickly.
- Used to create high resolution plots.

- Used to customize many package specific plots.
- Over 100 related extensions *(https://exts.ggplot2.tidyverse.org/gallery/)*

Outside of base R plotting, one of the most popular packages used to generate graphics in R is `ggplot2`, which is associated with a family of packages collectively known as the tidyverse. `GGplot2` allows the user to create informative plots quickly by using a 'grammar of graphics' implementation, which is described as "a coherent system for describing and building graphs" R4DS *(https://r4ds.had.co.nz/data-visualisation.html#:~:text=ggplot2%20implements%20the%20grammar%20of,applying%20it%20in%20many*
We will see this in action shortly. The power of this package is that plots are built in layers and few changes to the code result in very different outcomes. This makes it easy to reuse parts of the code for very different figures.

Being a part of the tidyverse collection, `ggplot2` works best with data organized so that individual observations are in rows and variables are in columns.

Getting started with ggplot2

To begin plotting, we need to load the `ggplot2` package. R packages are loadable extensions that contain code, data, documentation, and tests in a standardized shareable format that can easily be installed by R users.

R packages must be loaded from your R library every time you open and use R. If you haven't yet installed the ggplot2 package on your local machine, you will need to do that using `install.packages("ggplot2")`.

```
#load the ggplot2 library; you could also load library(tidyverse)
library(ggplot2)
```

Getting help

The R community is extensive and getting help is now easier than ever with a simple web search. If you can't figure out how to plot something, give a quick web search a try. Great resources include internet tutorials, R bookdowns, and stackoverflow. You should also use the help features within RStudio to get help on specific functions or to find vignettes. Try entering `ggplot2` in the help search bar in the lower right panel under the `Help` tab.

Resources for Learning

1. ggplot2 cheatsheet
2. The R Graph Gallery *(https://www.r-graph-gallery.com/)*
3. The R Graphics Cookbook *(https://r-graphics.org/recipe-quick-bar)*

4. BTEP Courses *(https://bioinformatics.ccr.cancer.gov/btep/class-documents/)*

Example Data

The example data we will use for plotting are from a bulk RNA-Seq experiment described by Himes et al. (2014) *(https://pubmed.ncbi.nlm.nih.gov/24926665/)* and available in the Bioconductor package airway *(https://bioconductor.org/packages/release/data/experiment/html/airway.html)*. In this experiment, the authors were comparing transcriptomic differences in primary human ASM cell lines treated with dexamthasone, a common therapy for asthma. Each cell line included a treated and untreated negative control resulting in a total sample size of 8.

```
#data import from excel
exdata<-readxl::read_xlsx("./data/RNASeq_totalcounts_vs_totaltrans.x
                          1,.name_repair = "universal", skip=3)
exdata
```

```
# A tibble: 8 × 4
  Sample.Name Treatment     Number.of.Transcripts Total.Counts
  <chr>       <chr>                         <dbl>        <dbl>
1 GSM1275863  Dexamethasone                 10768     18783120
2 GSM1275867  Dexamethasone                 10051     15144524
3 GSM1275871  Dexamethasone                 11658     30776089
4 GSM1275875  Dexamethasone                 10900     21135511
5 GSM1275862  None                          11177     20608402
6 GSM1275866  None                          11526     25311320
7 GSM1275870  None                          11425     24411867
8 GSM1275874  None                          11000     19094104
```

These derived data include total transcript read counts summed by sample and the total number of transcripts recovered by sample that had at least 100 reads.

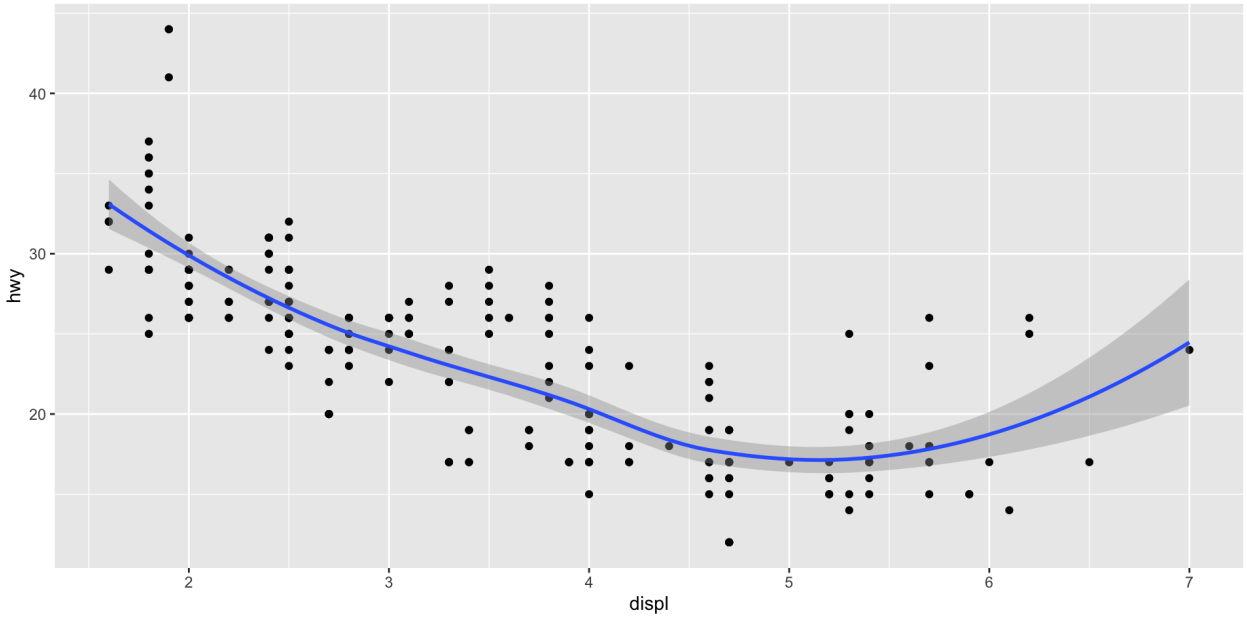
Get the data

You can grab this file here.

Practice Data

There are a number of built-in data sets available for practicing with ggplot2. Check these out here *(https://ggplot2.tidyverse.org/reference/#data)*!

For example, `mtcars` is commonly used in ggplot2 documentation:



The ggplot2 template

The basic ggplot2 template:

```
ggplot(data = DATA) +
  GEOM_FUNCTION(mapping = aes(<MAPPINGS>))
```

The only required components to begin plotting are the data we want to plot, geom function(s), and mapping aesthetics. Notice the + symbol following the `ggplot()` function. This symbol will precede each additional layer of code for the plot, and it is important that it is **placed at the end of the line**. More on geom functions and mapping aesthetics to come.

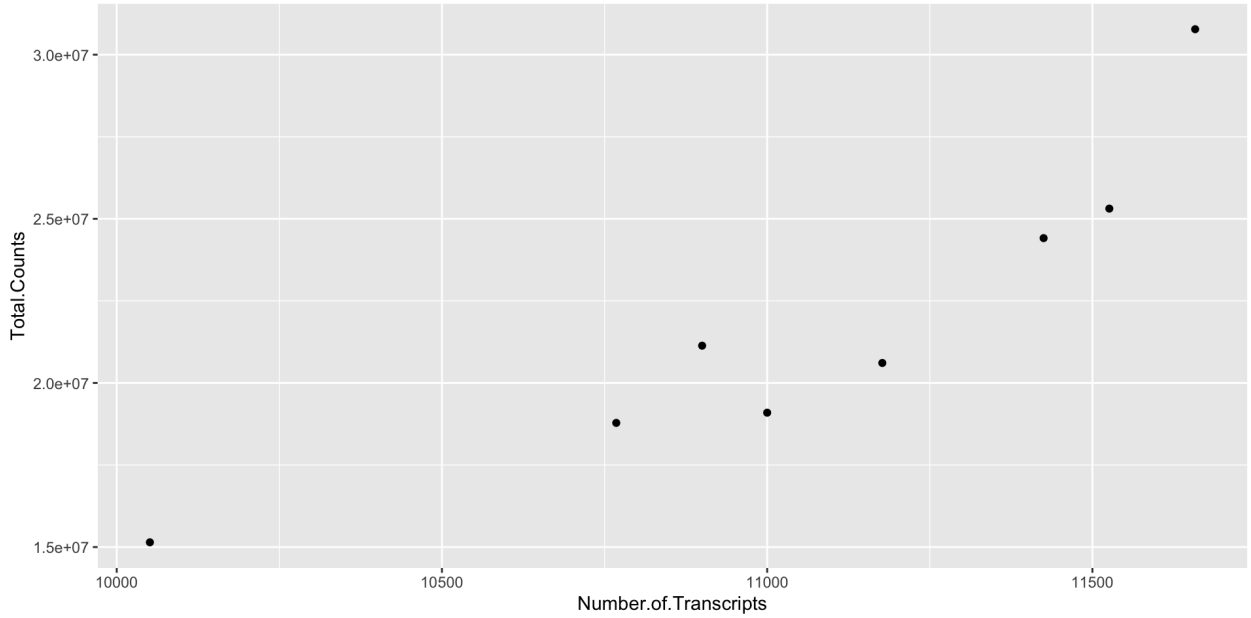
Using the template

To get familiar with the basic ggplot2 template, lets answer the following question:

What is the relationship between total transcript sums per sample and the number of recovered transcripts per sample?

We can plot using:

```
#let's plot our data
ggplot(data=exdata) +
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts))
```



How did we create this plot?

The first step in creating this plot was initializing the ggplot object using the function `ggplot()`. Remember, we can look further for help using `?ggplot()`. The function `ggplot()` takes data, mapping, and further arguments. However, none of this needs to actually be provided at the initialization phase, which creates the coordinate system from which we build our plot. But, typically, you should at least call the data at this point.

The data we called was from the data frame `exdata`, which we created above. Next, we provided a geom function (`geom_point()`), which created a scatter plot. This scatter plot required mapping information, which we provided for the x and y axes. More on this in a moment.

Geom functions

> A geom is the geometrical object that a plot uses to represent data. People often describe plots by the type of geom that the plot uses. --- R4DS *(https://r4ds.had.co.nz/data-visualisation.html#geometric-objects)*

There are multiple geom functions (>40 in ggplot2) that change the basic plot type or the plot representation.

- scatter plots (`geom_point()`),
- line plots (`geom_line()`,`geom_path()`),
- bar plots (`geom_bar()`, `geom_col()`),
- line modeled to fitted data (`geom_smooth()`),
- heat maps (`geom_tile()`),
- geographic maps (`geom_polygon()`), etc.

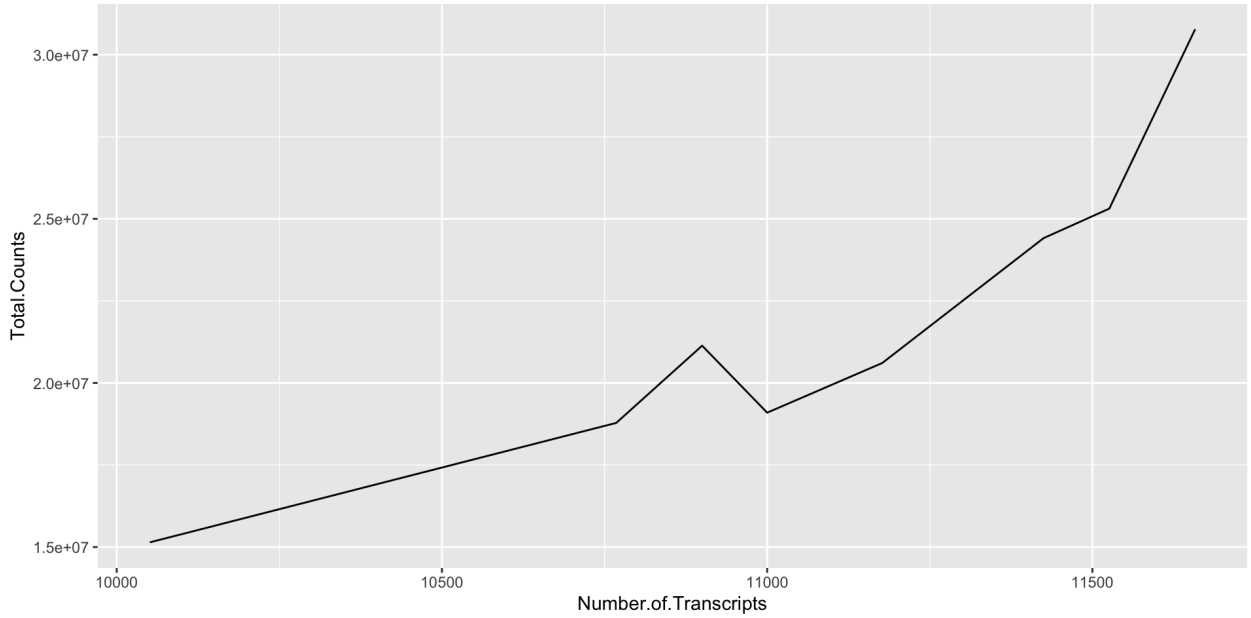
You can also see a number of options pop up when you type `geom` into the script pane of RStudio, or you can look up the `ggplot2` documentation in the help tab.

Changing the Geom function

We can see how easy it is to change the way the data is plotted. Let's plot the same data using `geom_line()`.

Creating a line plot

```
ggplot(data=exdata) +
  geom_line(aes(x=Number.of.Transcripts, y = Total.Counts))
```

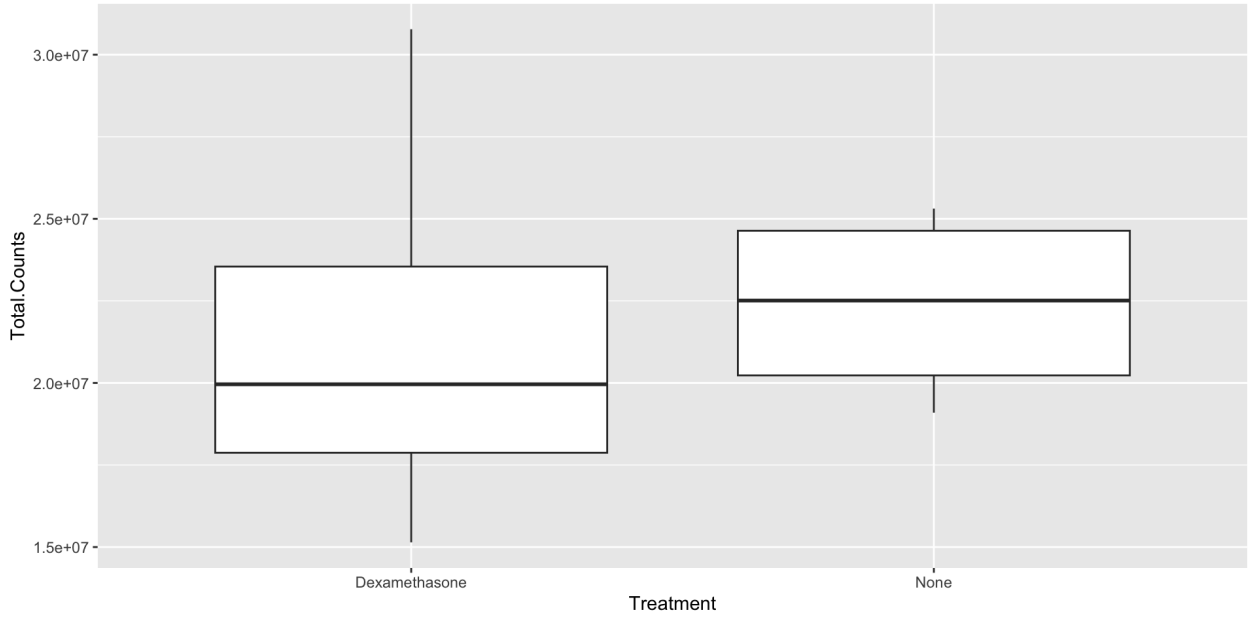


Here we can see one of the advantages of ggplot2, which is that it is easy to change the overall plot representation with small edits to the code.

Creating a boxplot

Let's plot the same data using `geom_boxplot()`.A boxplot *(https://www.data-to-viz.com/caveat/boxplot.html)* can be used to summarize the distribution of a numeric variable across groups.

```
ggplot(data=exdata) +
  geom_boxplot(aes(x=Treatment, y = Total.Counts))
```



This time we also modified the `x` argument.

Mapping and aesthetics (`aes()`)

The geom functions require a mapping argument. The mapping argument includes the `aes()` function, which "describes how variables in the data are mapped to visual properties (aesthetics) of geoms" (ggplot2 R Documentation). If not included it will be inherited from the `ggplot()` function.

> An aesthetic is a visual property of the objects in your plot.---R4DS *(https://r4ds.had.co.nz/data-visualisation.html)*

Mapping aesthetics include some of the following:

1. the x and y data arguments
2. shapes
3. color
4. fill
5. size
6. linetype
7. alpha

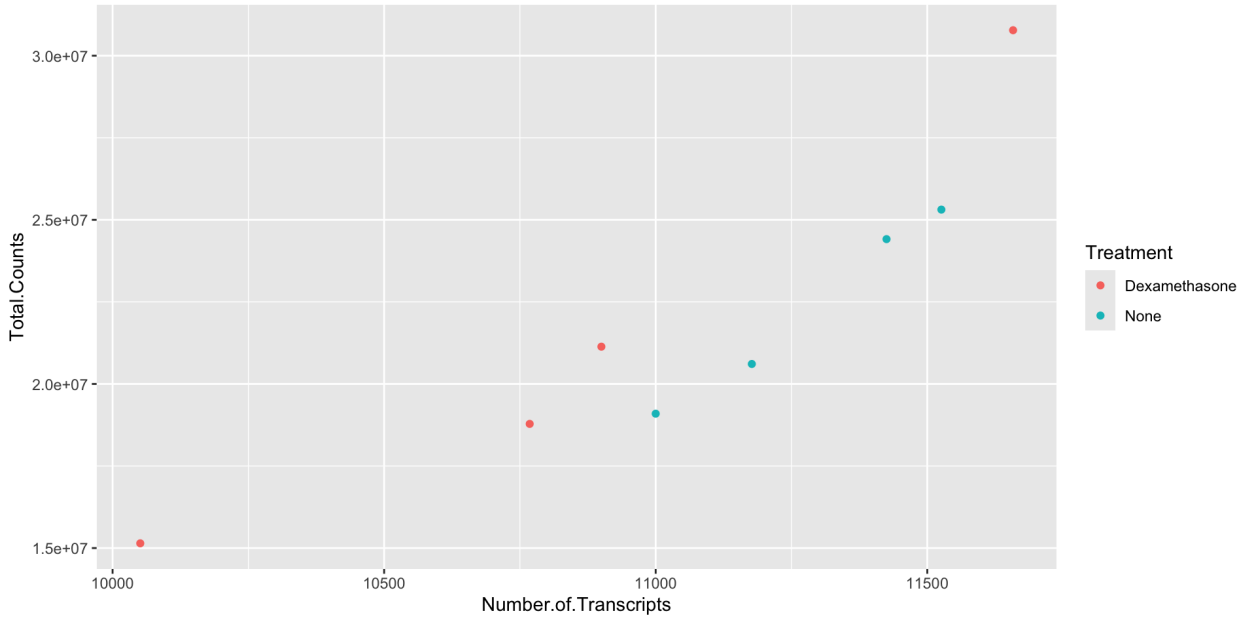
This is not an all encompassing list of mapping aesthetics.

Map a Color to a Variable

Now that we know what we mean by "aesthetics", let's map color to a variable within the data.

Is there a relationship between treatment ("dex") and the number of transcripts or total counts?

```
#adding the color argument to our mapping aesthetic
ggplot(exdata) +
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts,
                 color=Treatment))
```



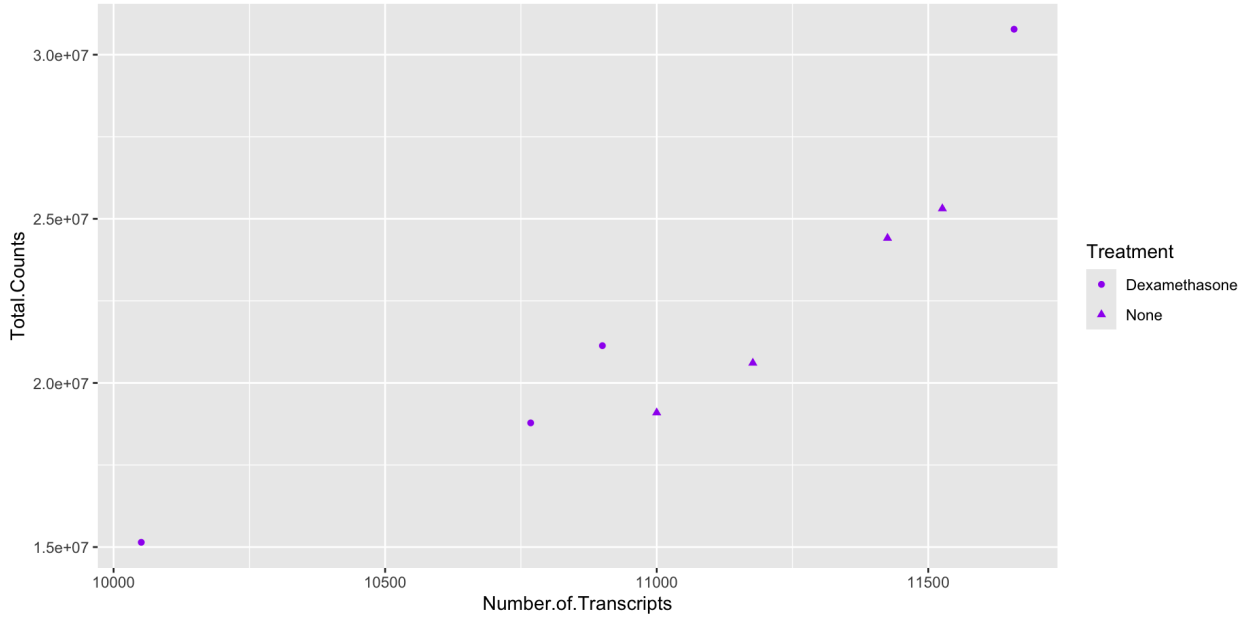
Notice how we changed the color of our points to represent the variable "Treatment". We did this by setting color equal to 'Treatment' within the `aes()` function. This mapped our aesthetic, color, to a variable we were interested in exploring.

From this, we can see that there is potentially a relationship between treatment and the number of transcripts or total counts. ASM cells treated with dexamethasone in general have lower total numbers of transcripts and lower total counts.

Changing the color of all points

Aesthetics that are not mapped to our variables are placed outside of the `aes()` function. These aesthetics are manually assigned and do not undergo the same scaling process as those within `aes()`. For example, we can color all points on the plot purple.

```
#map the shape aesthetic to the variable "dex"
#use the color purple across all points (NOT mapped to a variable)
ggplot(exdata) +
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts,
                 shape=Treatment), color="purple")
```

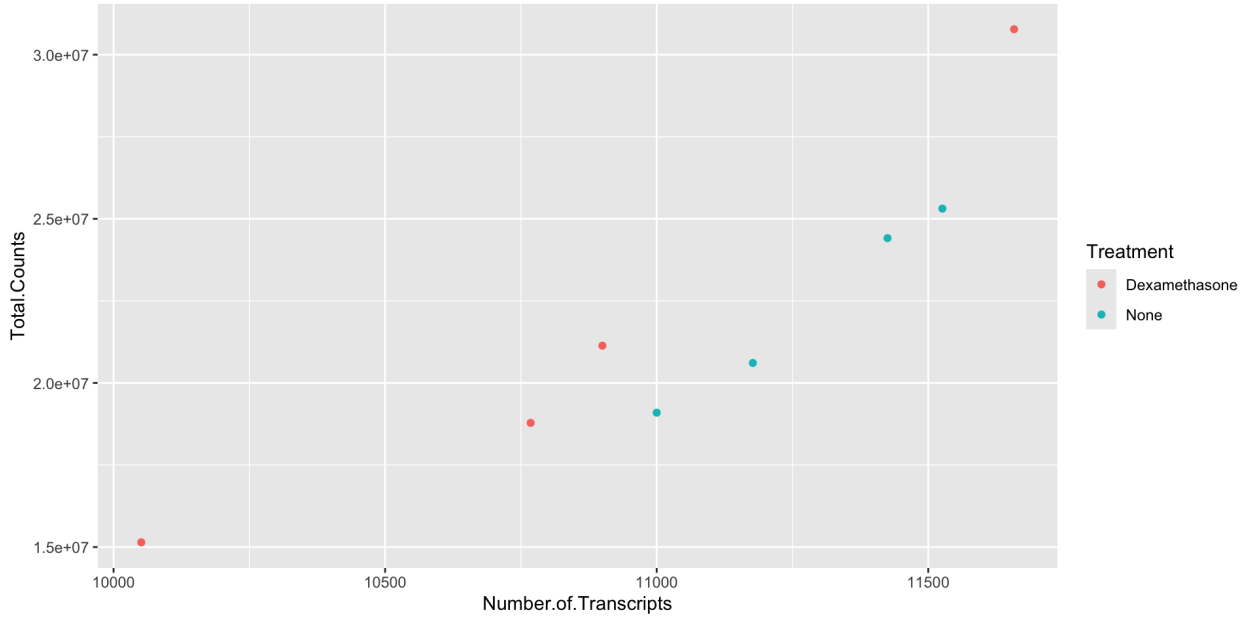


Here, we also mapped 'Treatment' to an aesthetic other than color, shape. **By default, ggplot2 will only map six shapes at a time, and if your number of categories goes beyond 6, the remaining groups will go unmapped.** This is by design because it is hard to discriminate between more than six shapes at any given moment. This is a clue from ggplot2 that you should choose a different aesthetic to map to your variable. However, if you choose to ignore this functionality, you can manually assign more than six shapes *(https://r-graphics.org/recipe-scatter-shapes)*.

We could have just as easily mapped "Treatment" to alpha, which adds a gradient to the point visibility by category, or we could map it to size. There are multiple options, so feel free to explore a little with your plots.

Defaults

There are many defaults when generating a plot with ggplot2, but almost everything you see can be customized.



Here we can see:

- Assigned colors
- A legend
- axis titles
- a plot background
- tick marks

The assignment of color, shape, or alpha to our variable occurs automatically, with a unique aesthetic level representing each category (i.e., 'Dexamethasone', 'none') within our variable. Most of what we see on this plot is autogenerated with defaults and we can change these defaults, for example, what colors are used, by adding additional layers to our code.

How can we modify colors?

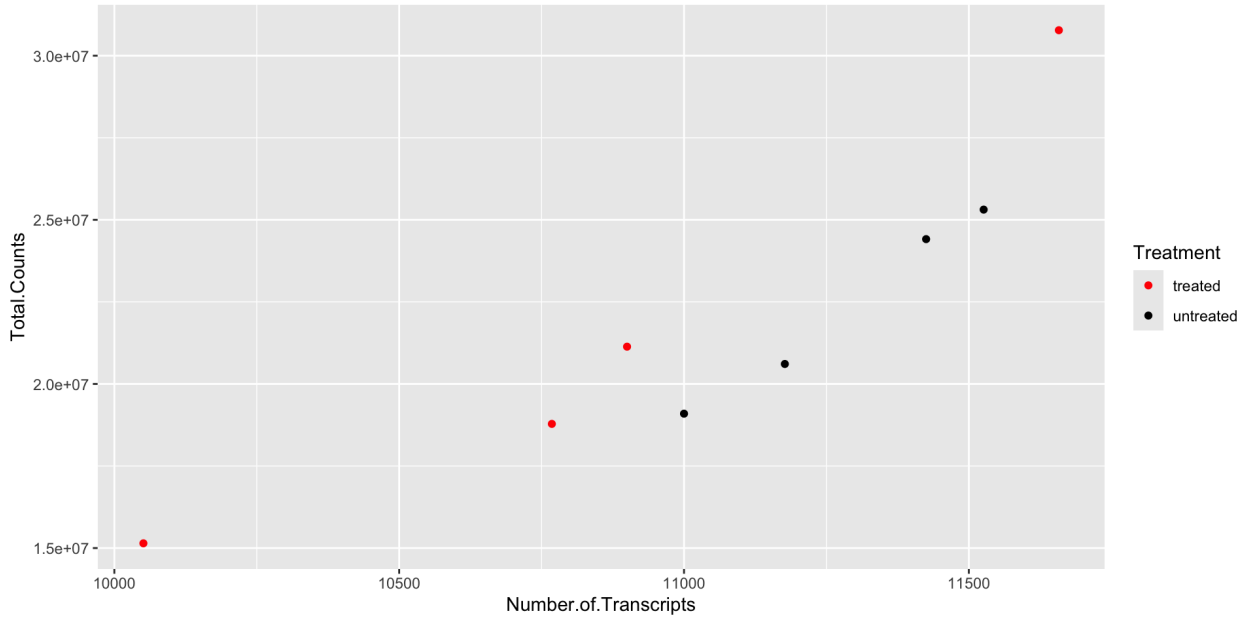
Colors are assigned to the fill and color aesthetics in `aes()`. We can change the default colors by providing an additional layer to our figure. To change the color, we use the scale_color functions:

- `scale_color_manual()`,
- `scale_color_brewer()` *(https://r-graph-gallery.com/38-rcolorbrewers-palettes.html)*,
- `scale_color_grey()`, etc.

Example:

```
scatter_plot <- ggplot(exdata) +
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts,
                 color=Treatment))
scatter_plot +
```

```
  scale_color_manual(values=c("red","black"),
                     labels=c('treated','untreated'))
```



We can also modify the behavior by adding additional arguments. Here we changed the color labels in the legend using the `labels` argument.

There are scale functions for other aesthetics (e.g., shape, alpha, line) as well.

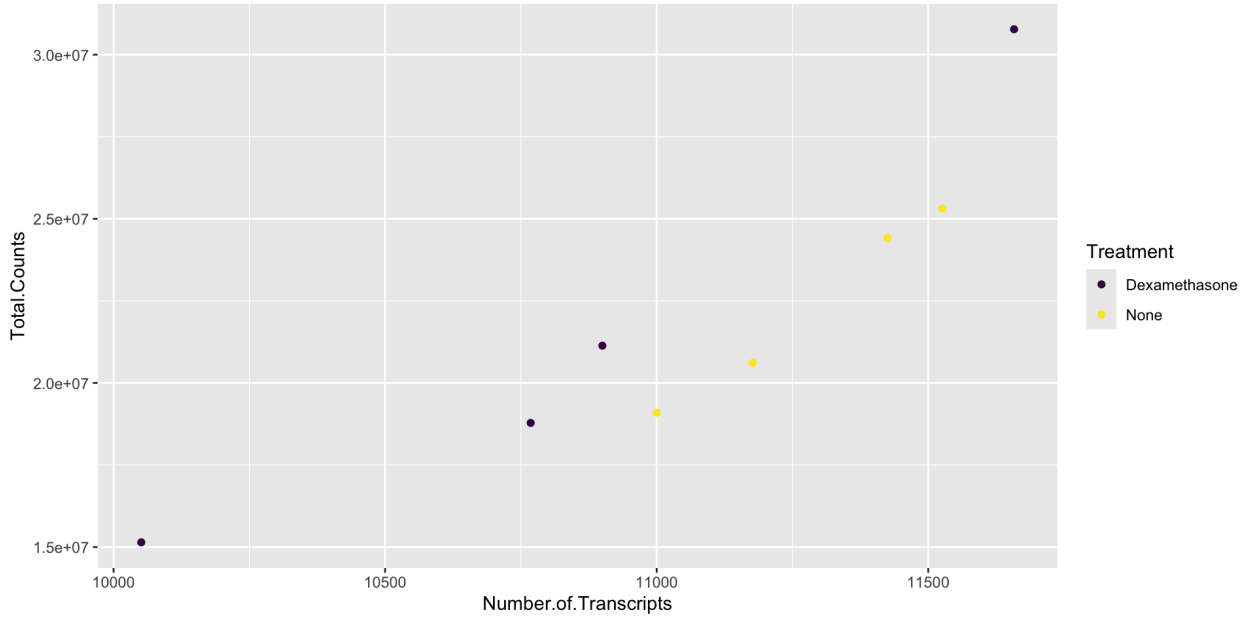
More on Colors

There are a number of ways to specify the color argument including by name, number, and hex code. Here *(https://www.r-graph-gallery.com/ggplot2-color.html)* is a great resource from the R Graph Gallery *(https://www.r-graph-gallery.com/index.html)* for assigning colors in R.

There are also a number of complementary packages in R that expand our color options.

- `viridis` *(https://cran.r-project.org/web/packages/viridis/index.html)* - provides colorblind friendly palettes.
- `randomcoloR` *(https://cran.r-project.org/web/packages/randomcoloR/index.html)* - generates large numbers of random colors.
- `Paletteer` *(https://github.com/EmilHvitfeldt/paletteer)* - contains a comprehensive set of color palettes to load the palettes from multiple packages all at once.

```
library(viridis)
ggplot(exdata) +
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts,
                 color=Treatment)) +
scale_color_viridis(discrete=TRUE, option="viridis")
```



Expanding our ggplot2 template

What do we need to make a plot:

1. the data
2. one or more geoms
3. aesthetic mappings
4. facets (i.e., subplots)
    - use `facet_grid()`, `facet_wrap()`
5. optional parameters that customize our plot (e.g., themes, axis settings, legend settings).
6. coordinate systems *(https://ggplot2.tidyverse.org/reference/#coordinate-systems)*
7. statistical transformations.

The first three line items are required, while the others are controlled by defaults, necessitating additional modification.

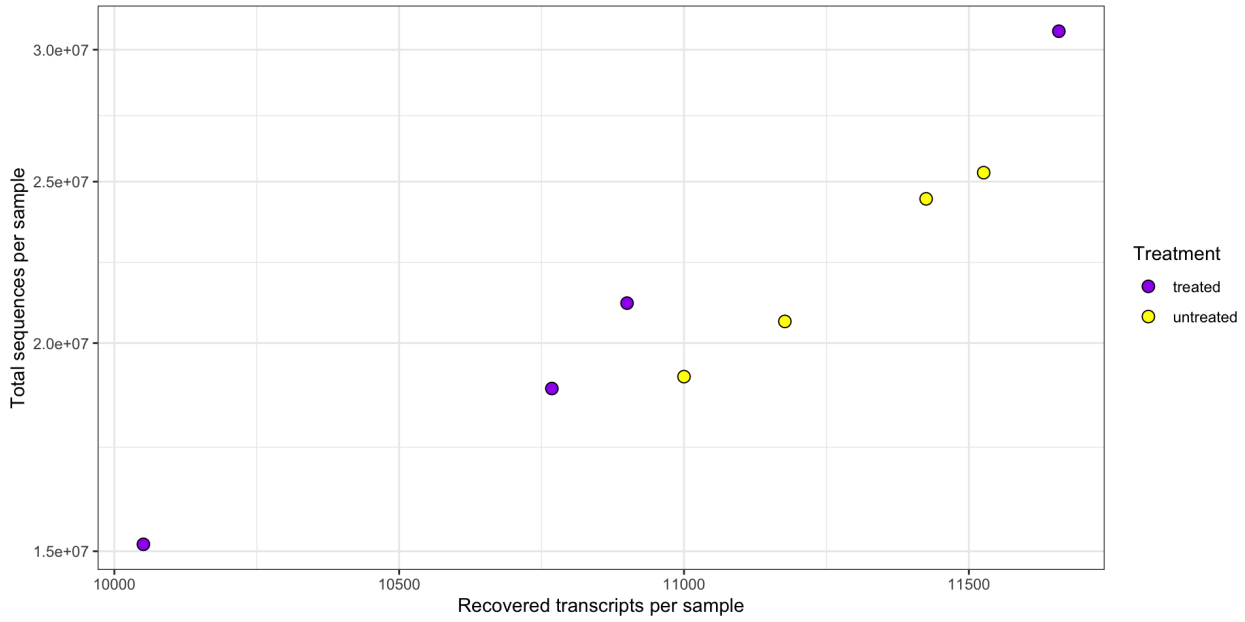
Making our plot ready for publication

How do we ultimately get our figures to a publishable state? The bread and butter of pretty plots really falls to the additional non-data layers of our ggplot2 code. These layers will include code to label the axes *(https://ggplot2.tidyverse.org/reference/labs.html)*, scale the axes *(https://ggplot2.tidyverse.org/reference/#scales)*, and customize the legends *(https://ggplot2.tidyverse.org/articles/faq-customising.html#legends)* and theme *(https://ggplot2.tidyverse.org/reference/theme.html)*.

For example,

```
ggplot(exdata) +
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts,
                 fill=Treatment),
             shape=21,size=3) +
  #can change labels of fill levels along with colors
  scale_fill_manual(values=c("purple", "yellow"),
                    labels=c('treated','untreated'))+

  labs(x="Recovered transcripts per sample",
       y="Total sequences per sample", fill="Treatment")+
  scale_y_continuous(trans="log10") + #log transform the y axis
  theme_bw() #add a complete theme black / white
```



Saving your plot

The easiest way to save our plot with ggplot2 is `ggsave()`. This function will save the last plot that you displayed by default. Look at the function parameters using `?ggsave()`.

```
ggsave("Plot1.png",width=5.5,height=3.5,units="in",dpi=300)
```

Key Points

- ggplot2 is a popular package for data visualization.
- We learned how to create a plot, change plot types, and add layers for further customization.

- The best way to learn ggplot2 is to use ggplot2.
    - Use online resources (e.g., Google) to help you build your plot.
    - Reuse your code and modify as needed.
- Check out other resources:
    - Data Visualization with R *(https://bioinformatics.ccr.cancer.gov/docs/data-visualization-with-r/index.html)*
    - Coursera lessons *(https://bioinformatics.ccr.cancer.gov/btep/self-learning/)*
- Email us at ncibtep@nih.gov
    - General bioinformatics help
    - Training requests

Related packages to check out

There are so many different extensions. Here are a few to check out:

- patchwork *(https://github.com/thomasp85/patchwork#patchwork)* - combine multiple plots into a single figure
- ggfortify *(https://github.com/sinhrks/ggfortify)* - autoplot functions for quick and easy plotting
- ggpubr *(https://rpkgs.datanovia.com/ggpubr/)* - integrate statistical results
- ggExtra *(https://github.com/daattali/ggExtra)* - add subplots along the plot margins

Other packages like EnhancedVolcano *(https://bioconductor.org/packages/release/bioc/html/EnhancedVolcano.html)* can be modified using ggplot2 layers.