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

Creating and modifying scatter plots: PCA and Volcano

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

December 19, 2024: Creating and modifying scatter plots: PCA and Volcano

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?

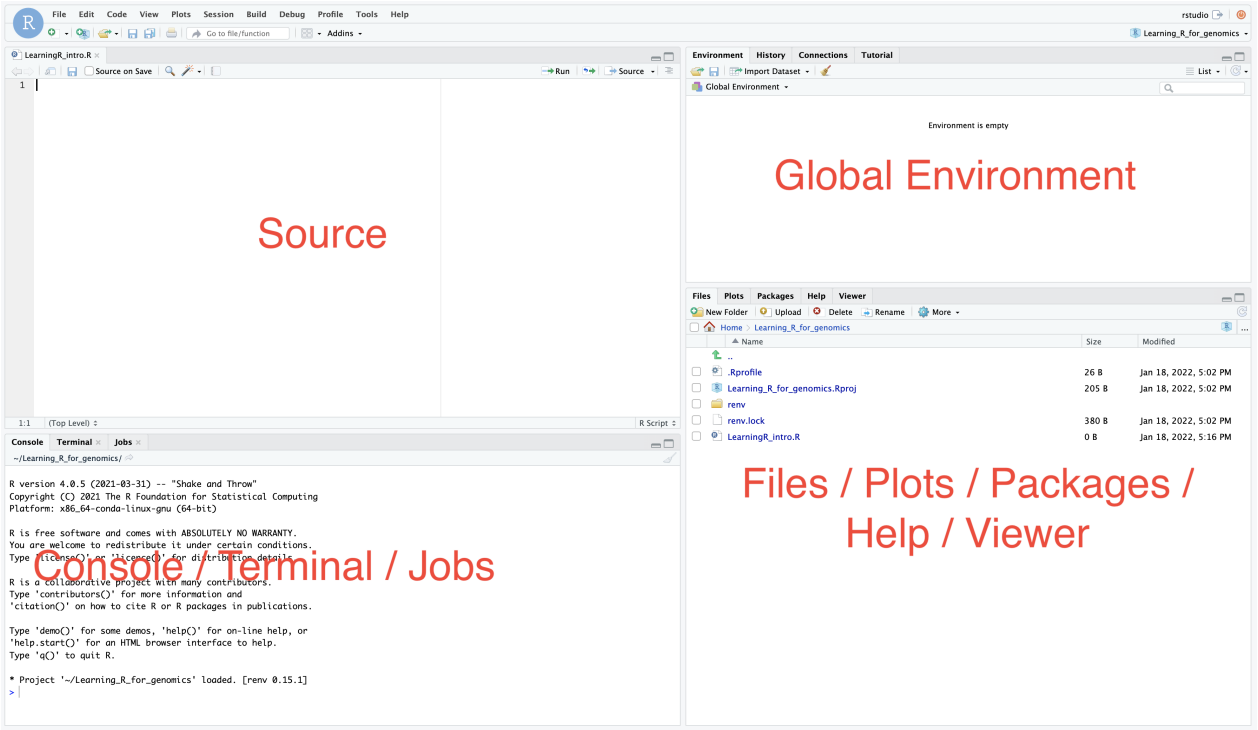
R is a computational language and environment for statistical computing and graphics.

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.

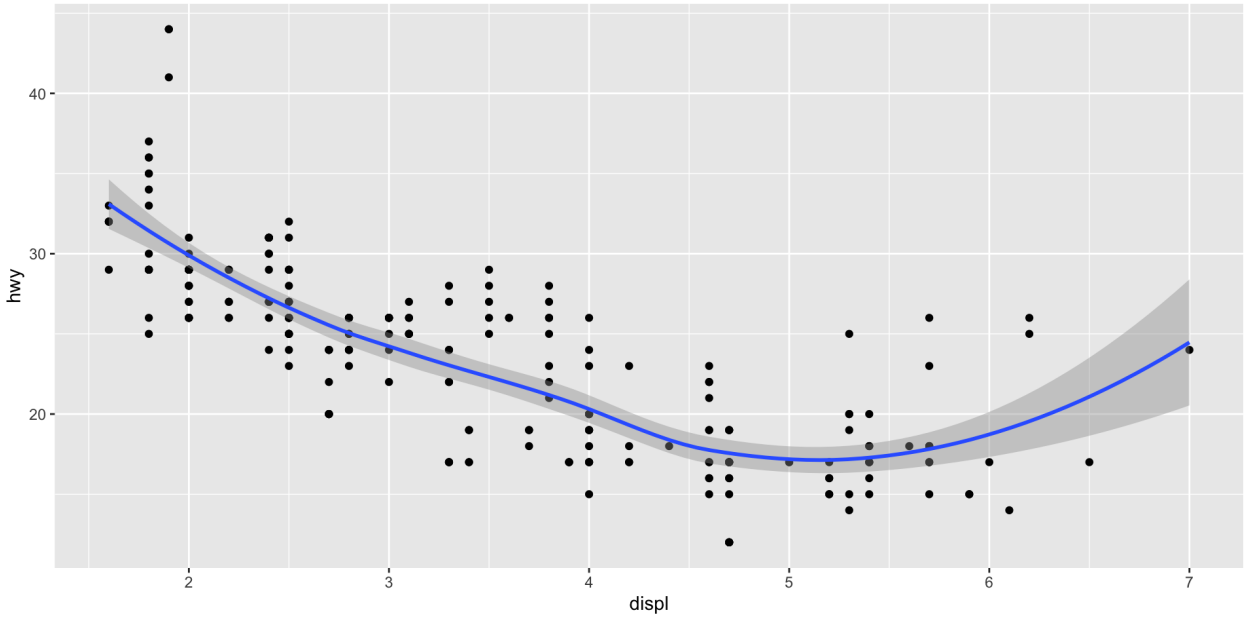
Get the data

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.

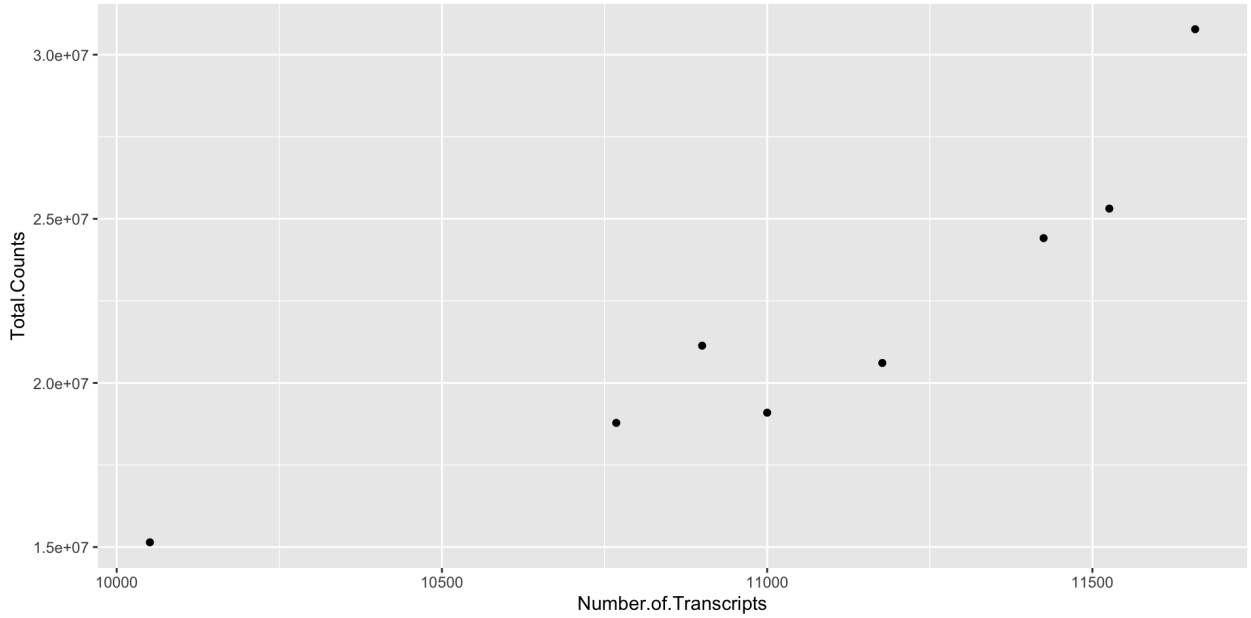
Using the template

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.

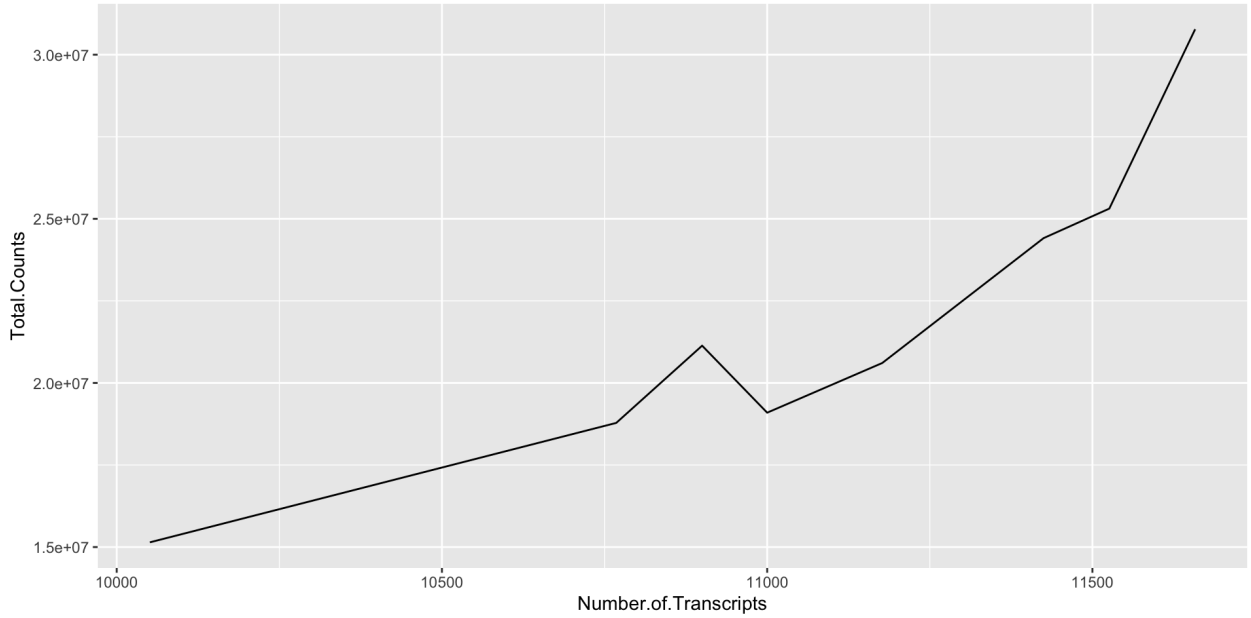
You can also see a number of options pop up when you type `geom` into the script pane of RStudio, or you can look up the `ggplot2` documentation in the help tab.

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))
```

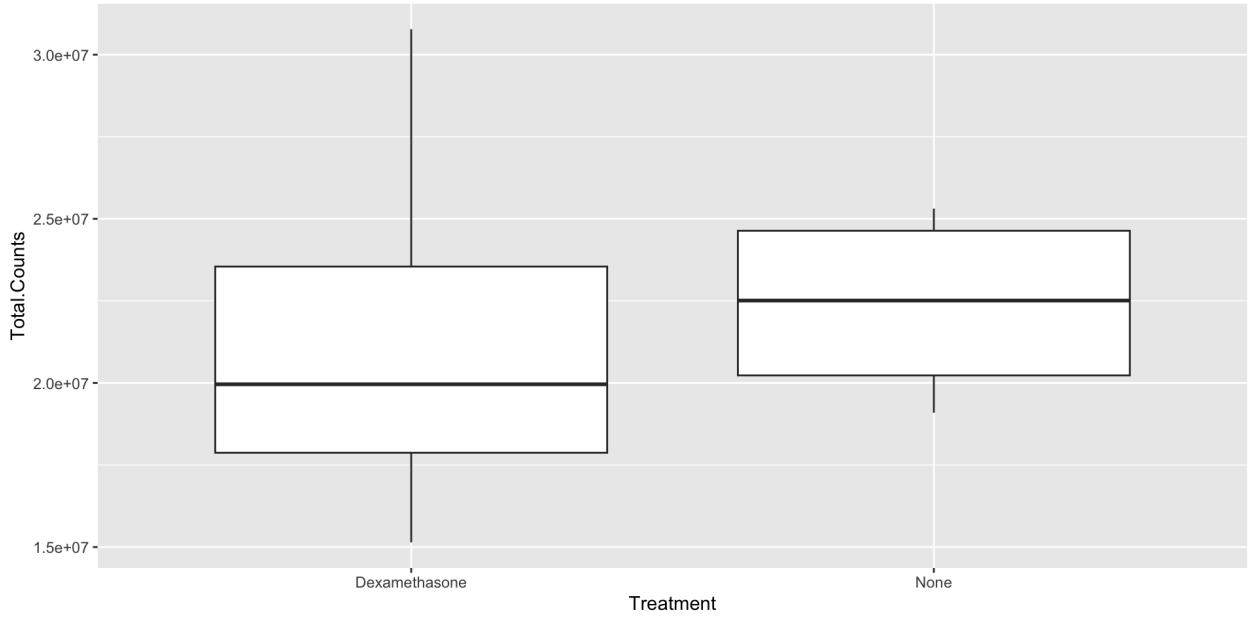


Here we can see one of the advantages of ggplot2, which is that it is easy to change the overall plot representation with small edits to the code.

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

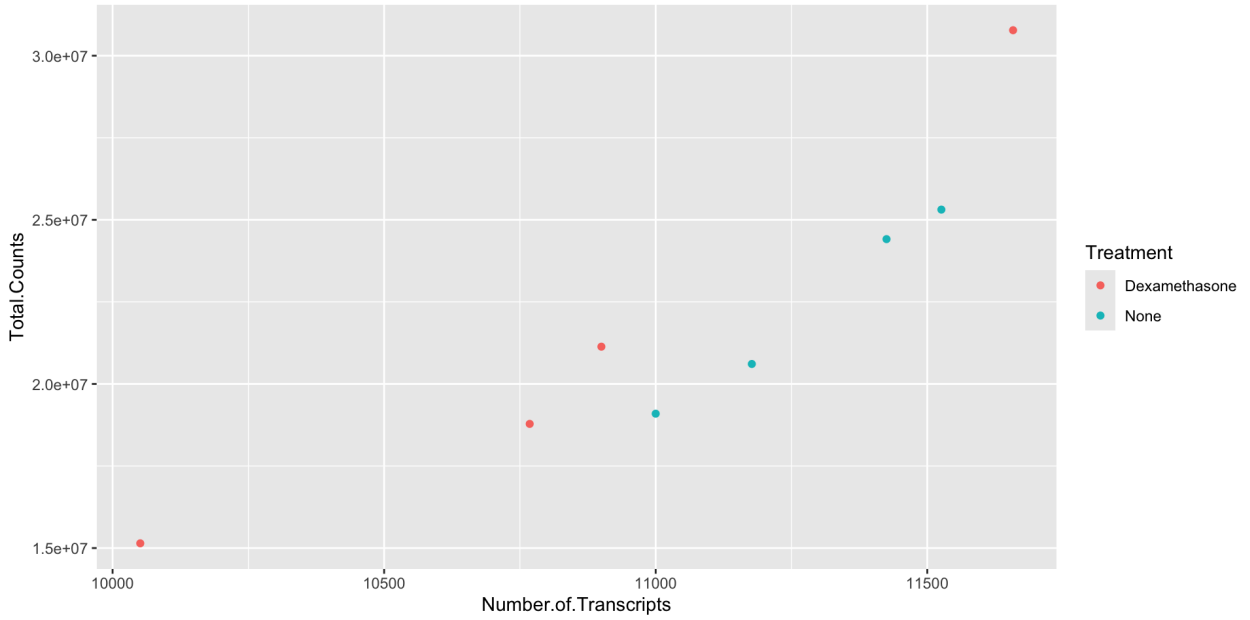
This is not an all encompassing list of mapping aesthetics.

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))
```



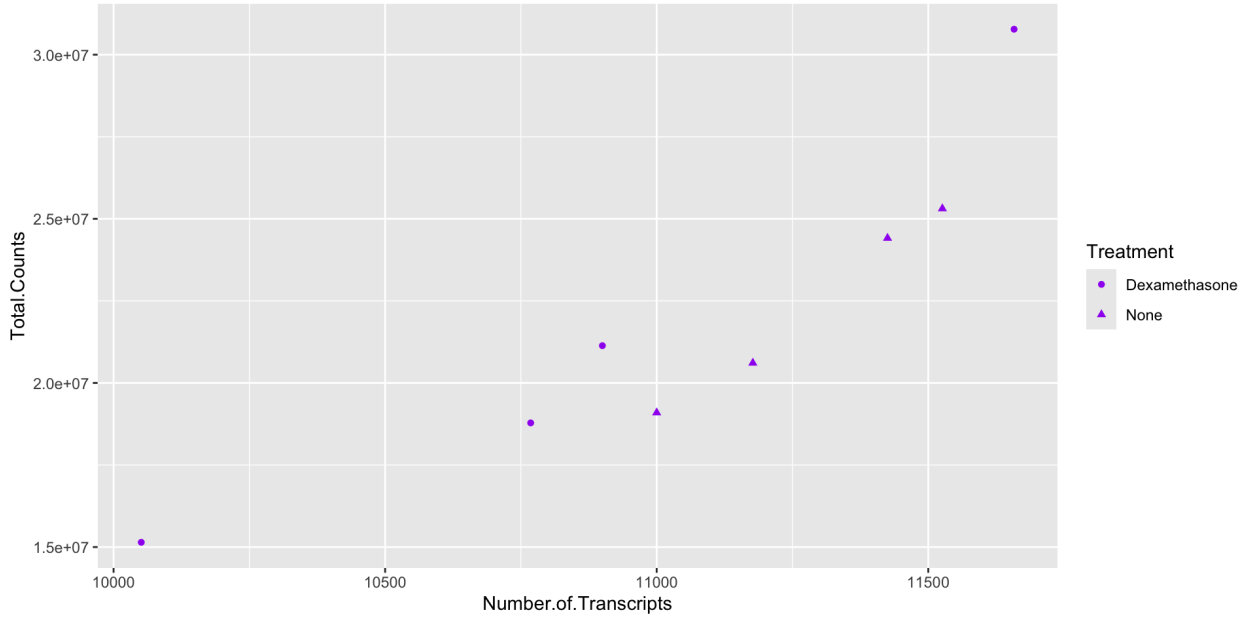
Notice how we changed the color of our points to represent the variable "Treatment". We did this by setting color equal to 'Treatment' within the `aes()` function. This mapped our aesthetic, color, to a variable we were interested in exploring.

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")
```

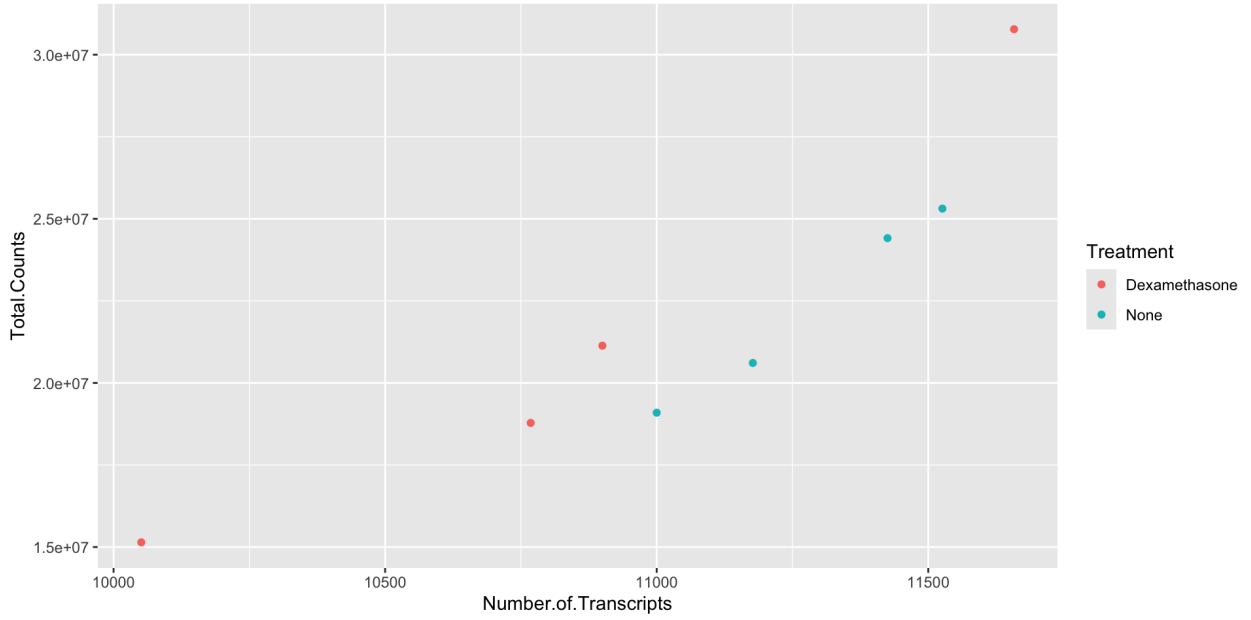


Here, we also mapped 'Treatment' to an aesthetic other than color, shape. **By default, ggplot2 will only map six shapes at a time, and if your number of categories goes beyond 6, the remaining groups will go unmapped.** This is by design because it is hard to discriminate between more than six shapes 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?

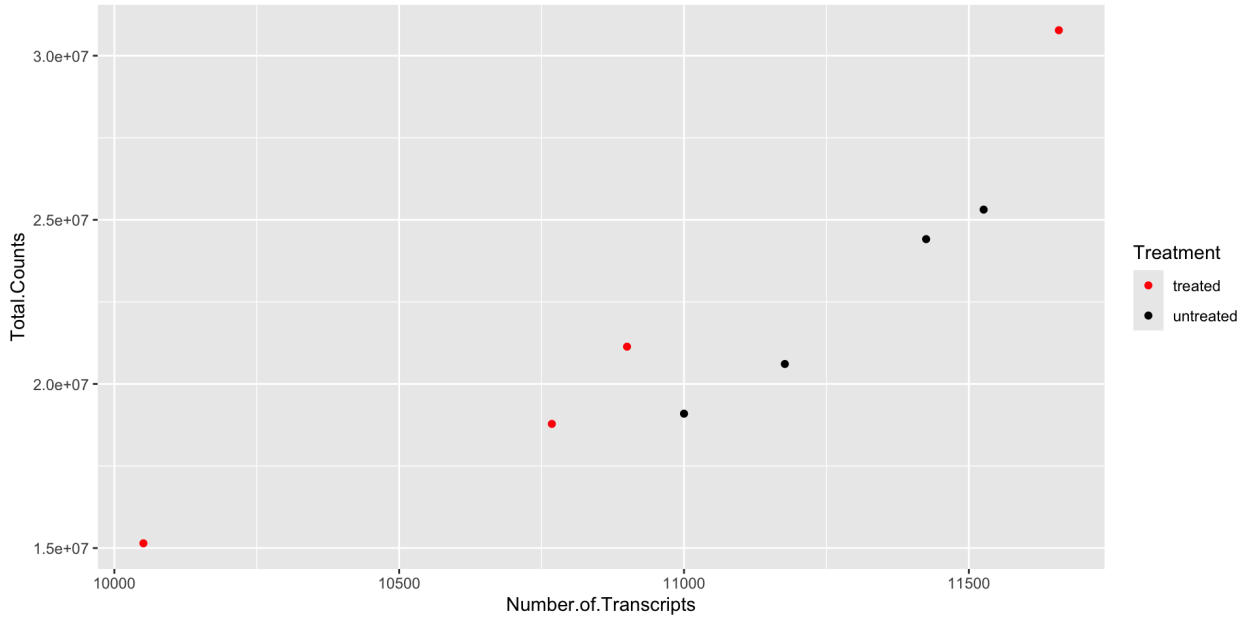
Colors are assigned to the fill and color aesthetics in `aes()`. We can change the default colors by providing an additional layer to our figure. To change the color, we use the scale_color functions:

- `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.

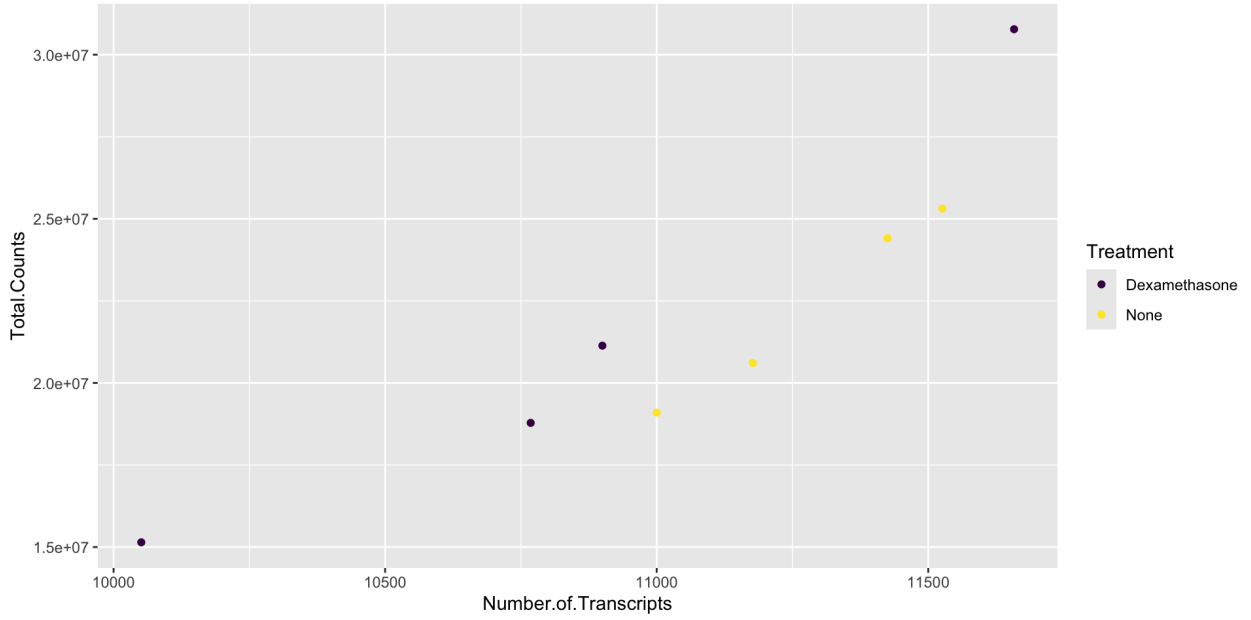
More on Colors

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.

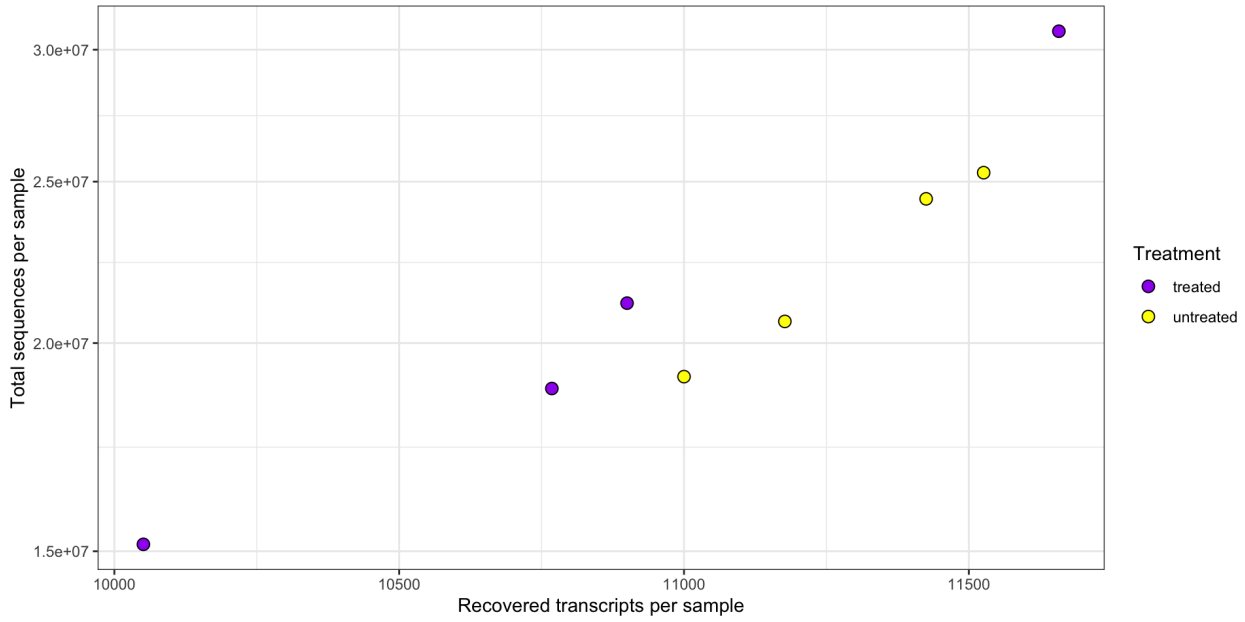
The first three line items are required, while the others are controlled by defaults, necessitating additional modification.

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.

Creating and modifying scatter plots: PCA and Volcano

Objectives

1. Learn how to make and modify scatter plots using ggplot2.
2. Learn how to visualize PCA results.
3. Learn how to create a Volcano plot.

Load the libraries

```
library(ggplot2)
library(dplyr)
library(DESeq2)
```

Other R packages used in this tutorial include `patchwork`, `ggfortify`, `EnhancedVolcano`, and `ggrepel`.

What is ggplot2?

`ggplot2` is a popular R graphics package associated with a family of packages known as the tidyverse. Tidyverse packages work effectively on data stored in data frames (or tibbles), which store variables in columns and observations in rows. In

ggplot2, plots are built in layers, allowing one to incorporate multiple data sets and advanced features, thus allowing the generation of fairly complex plots.

To build a plot, we need:

- data to plot
- one or more geoms - the visual representation of the plot.
- mapping aesthetics - describe how the variables are mapped to the geoms

We can also include additional parameters:

- facets - subplots
- coordinate systems (default is Cartesian)
- plot layout and rendering options (customize legends, lines, text, and other features).
- statistical transformations / summarizations

The basic ggplot2 template:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>
  ) +
  <FACET_FUNCTION> +
  <COORDINATE SYSTEM> +
  <THEME>
```

Example data

Here we will use bulk RNA-Seq data available in the R package `airway`, which is from an experiment published by Himes et al. (2014). These data, which are available in R as a `RangedSummarizedExperiment` object, are from a bulk RNAseq experiment. In the experiment, the authors "characterized transcriptomic changes in four primary human ASM cell lines that were treated with dexamethasone," a common therapy for asthma. The airway package includes RNAseq count data from 8 airway smooth muscle cell samples. Each cell line includes a treated and untreated negative control.

We will use some normalized count data (rlog) and differential expression results processed according to the Bioconductor workflow, `rnaseqGene` *(https://bioconductor.org/packages/release/workflows/vignettes/rnaseqGene/inst/doc/rnaseqGene.html)*. The normalized data is stored within a `DESeqTransform` object, and differential expression results are provided in a comma separated file. The data used in this tutorial is available for download here. See `sessionInfo()` at the end of this tutorial for information related to package versions.

```
#load R object
air<- readRDS("./Data/normalized_air.rds")

#read differential expression results
dexp<- read.csv("./Data/deseq2_DEGs.csv",row.names=1)
dexp<- dexp %>% filter(!is.na(padj))
```

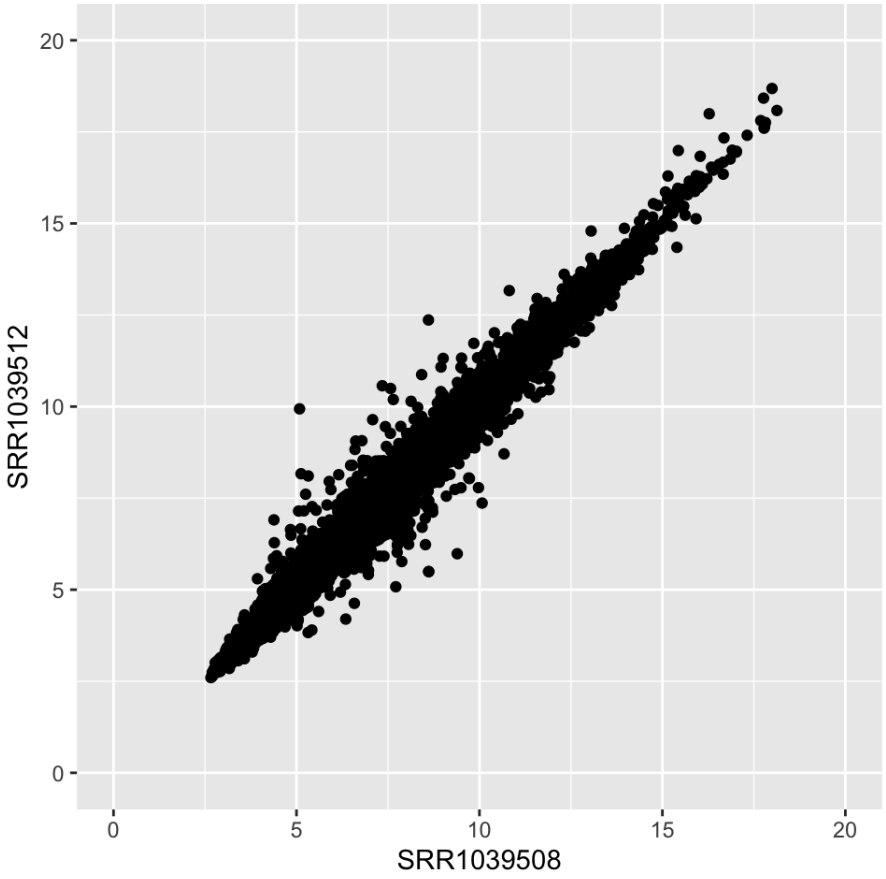
Scatter plots

> Scatterplots are useful for visualizing treatment–response comparisons, associations between variables, or paired data (e.g., a disease biomarker in several patients before and after treatment). - Holmes and Huber, 2021 *(https://web.stanford.edu/class/bios221/book/03-chap.html)*

Because scatter plots involve mapping each data point, the geom function used is `geom_point()`.

Let's check out the basic scatter. We can look at the relationship of our normalized count data from all genes between samples.

```
ggplot(data=assay(air))+ #DATA
  geom_point(aes(x=SRR1039508,y=SRR1039512)) + #what's on x and y ax
  coord_fixed(xlim=c(0,20),ylim=c(0,20))
```



Here, we include the data we want to plot (`assay(air)`), a geom function to represent the plot (`geom_point` for a scatter plot), and mapping aesthetics assigning the x-axis to the genes from one sample (`SRR1039508`) and the y-axis to the genes from another sample (`SRR1039512`).

Note

The loaded gene counts include normalized data via a regularized-logarithm transformation (rlog). Read more on this here *(https://genomebiology.biomedcentral.com/articles/10.1186/s13059-014-0550-8)*. This transformation effectively stabilizes the variance across the mean and is recommended for smaller data sets.

Common scatter plots used in genomics (PCA and Volcano)

1. PCA is mostly used in genomics for dimensionality reduction and uncovering patterns in highly dimensional data, including identifying technical and biological variation.

Examples of applications in genomics:

- RNA-Seq
- Batch correction.
- Identifying Co-expressed Genes.
- scRNA-Seq

- Dimensionality reduction.
- Evolutionary Genomics
- Examining population structure.
- Examining relationships between species.

Visualizing PCA also includes other types of plots

What we think of as a PCA plot, is usually a scatter plot of PC1 vs PC2 (sometimes called a biplot). However, PCA is multidimensional, and we may want to visualize other aspects (e.g., other PCs, degree of variance by PC (Scree plots), scores vs loadings (biplot)). For this, check out factoextra *(https://rpkgs.datanovia.com/factoextra/index.html)* or `PCAtools` *(https://www.bioconductor.org/packages/devel/bioc/vignettes/PCAtools/inst/doc/PCAtools.html)*.

1. Volcano plots are used to visualize statistical significance versus magnitude of change (fold change) between treatments or conditions in large genomic data sets (e.g., RNA-Seq, ChIP-Seq). Volcano plots are great for identifying genes or other features for further exploration.

Note

There are many other scatter plots used in genomics. We are simply focusing on PCA and volcano in this tutorial.

What is PCA?

Principal component analysis (PCA) is an exploratory linear dimension reduction method applied to highly dimensional (multivariate) data. It is an usupervised learning technique that treats all variables equally. The goal of PCA is to reduce the dimensionality of the data by transforming the data in a way that maximizes the variance explained. Read more here *(https://towardsdatascience.com/principal-component-analysis-pca-79d228eb9d24)* and here *(https://www.huber.embl.de/msmb/Chap-Multivariate.html)*.

There is an assumption that variables are highly correlated.

How does it work?

1. Data standardization - variables should be on the same scale (e.g., z-score transformation)
2. Calculate covariance- create a covariance matrix to understand the relationship between variables
3. Calculate eigenvectors (direction of variation) and eigenvalues (amount of variance) from the covariance matrix
4. Select the top k principal components
5. Transform the original data - obtain a representation of the data in the newly transformed space

Key points:

- The PCs represent a linear combination of the original variables
- PCs are uncorrelated
- The number of PCs is equivalent to the number of original variables in your data set.
- PC1 accounts for the most variance, and each subsequent PC will explain less and less variance.

For a more detailed explanation of PCA, see the following:

- https://benthamopen.com/contents/pdf/TOBIOIJ/TOBIOIJ-7-19.pdf *(https://benthamopen.com/contents/pdf/TOBIOIJ/TOBIOIJ-7-19.pdf)*
- https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues *(https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues)*
- https://www.billconnelly.net/?p=697 *(https://www.billconnelly.net/?p=697)*
- https://builtin.com/data-science/step-step-explanation-principal-component-analysis *(https://builtin.com/data-science/step-step-explanation-principal-component-analysis)*

Perform **PCA**

There are many packages and functions available in R programming for performing PCA. Some of the most popular functions are `stats::prcomp()`, `stats::princomp()`, `FactoMineR::PCA()`, and `ade4::dudi.pca()`. These functions largely differ in the method used for PCA and the default arguments; see here *(https://aedin.github.io/PCAworkshop/articles/b_PCA.html#pca-in-r-1)*. We will focus on `stats::prcomp()` for this tutorial. If you are using an alternative function, make sure you are aware of the function arguments (and defaults).

PCA is used frequently in -omics fields. Often than not, there will be package specific functions for PCA and plotting PCA for different -omics analyses.

> **Note**
>
> `stats::princomp()`, which uses eigenvalue decomposition of the covariance matrix, is faster to run than `stats::prcomp()`, which uses singular value decomposition (SVD) on the original data matrix. *(https://stackoverflow.com/questions/14249156/principal-component-analysis-pca-in-r-which-function-to-use)*

We can use the function `prcomp()` to run PCA on our rlog transformed count data. This function requires numeric data. Notice by default, the function does not scale the data. If you have not transformed your data and you are working with variables of different units, consider scaling the data.

> **Why is scaling important?**
>
> Scaling and centering typically occur prior to PCA. Scaling removes biases from variables with high variances. Centering shifts the data to the origin by subtracting the mean of each feature. This is important for finding linear

subspaces and removing the affect of the mean. *(https://stats.stackexchange.com/questions/385775/normalizing-vs-scaling-before-pca)*

> Large variance in observed variable will contribute most to the overall variance of computed principal component. If the observed values of variables have very different ranges, then the data needs to be normalized/scaled. - Aniket Patil *(https://medium.com/analytics-vidhya/principal-component-analysis-pca-8a0fcba2e30c)*

Setting a scaling parameter to TRUE will standardize the data (i.e., perform a z-score transformation) so that the data has a mean of 0 and standard deviation of 1.

Now we run `prcomp()`:

```
#run PCA
pca<-prcomp(t(assay(air)))

#get structure of df
str(pca)
```

```
List of 5
 $ sdev    : num [1:8] 23.28 17.78 14.84 12.08 7.12 ...
 $ rotation: num [1:16139, 1:8] -0.00677 0.00359 0.00056 -0.00108 0.(
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:16139] "ENSG00000000003" "ENSG00000000419" "ENSG00(
  .. ..$ : chr [1:8] "PC1" "PC2" "PC3" "PC4" ...
 $ center  : Named num [1:16139] 9.45 9.02 7.89 5.83 12.45 ...
  ..- attr(*, "names")= chr [1:16139] "ENSG00000000003" "ENSG0000000(
 $ scale   : logi FALSE
 $ x       : num [1:8, 1:8] -25.8 14.1 -17.5 26.8 -22.4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:8] "SRR1039508" "SRR1039509" "SRR1039512" "SRR1039!
  .. ..$ : chr [1:8] "PC1" "PC2" "PC3" "PC4" ...
 - attr(*, "class")= chr "prcomp"
```

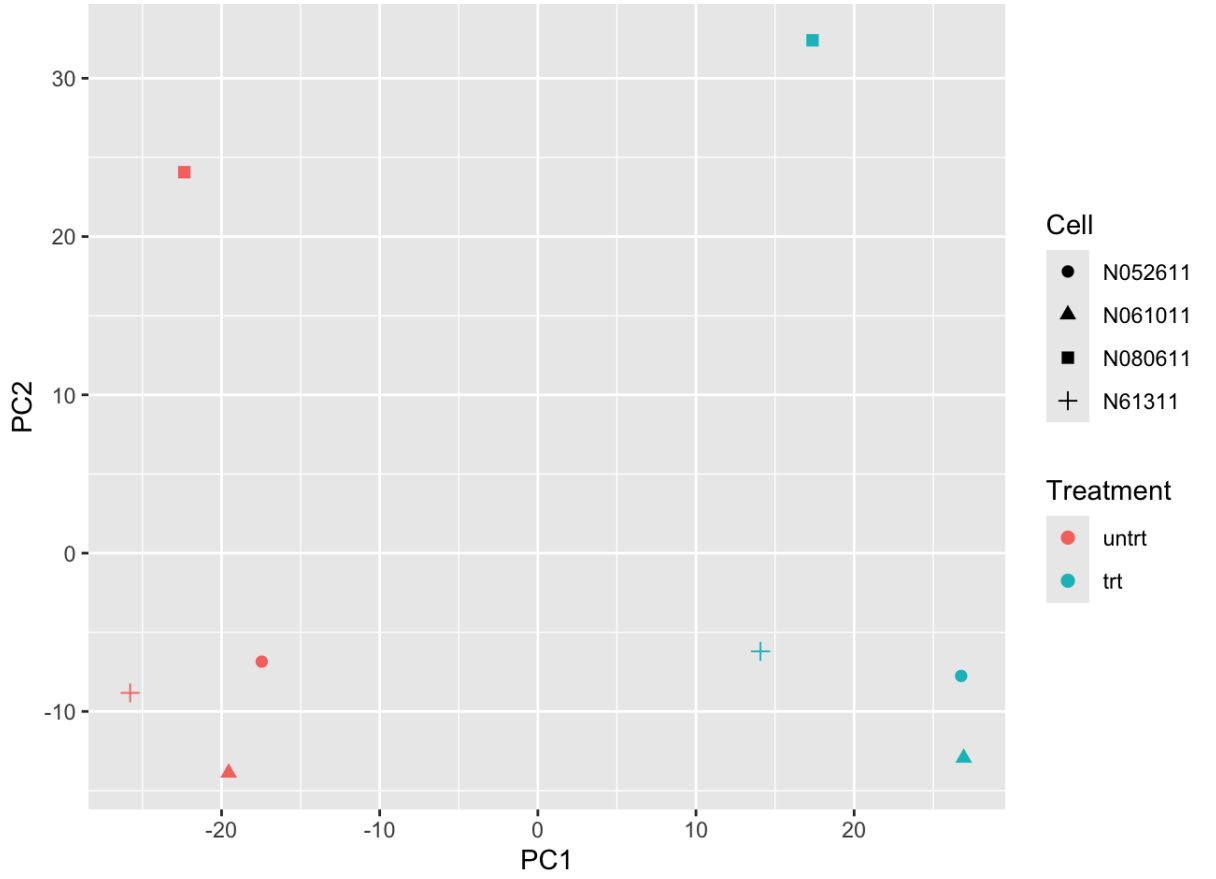
The object `pca` is a list of 5: the standard deviations of the principal components, a matrix of variable loadings, the centering and scaling parameters used, and the data projected on the principal components.

Plot PCA

To plot the first two axes of variation along with species information, we will need to make a data frame with this information. The axes are in `pca$x`.

```
#Build a data frame
pcaData <- as.data.frame(pca$x[, 1:3]) # extract first three PCs
```

```
# add the sample information
pcaData<- data.frame(colData(air)) |> select(cell, dex, Run) |> cbinc

# modify column names
colnames(pcaData) <- c("Cell","Treatment","Sample","PC1","PC2","PC3")

#Plot
ggplot(pcaData, aes(PC1, PC2, color = Treatment, shape = Cell)) +
  geom_point(size = 2) + # adding data points
  coord_fixed() # fixing coordinates
```



coord_fixed()

Tip 6, here *(https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6586259/)*, discusses the importance of aspect ratio and using `coord_fixed()` more in detail.

This is a decent plot showing us how our RNA-Seq samples are related based on gene expression. PC1 always explains the most variation followed by each successive PC. From this plot, we see that our samples cluster by treatment along PC1, while PC2 is capturing variation in the cell lines.

In the plot, the axes are missing the % explained variance. Let's add custom axes. We can do this with the `xlab()` and `ylab()` functions or the functions `labs()`. But first we need to grab some information from our PCA analysis. Let's use `summary(pca)`. This function provides a summary of results for a variety of model fitting functions and methods.

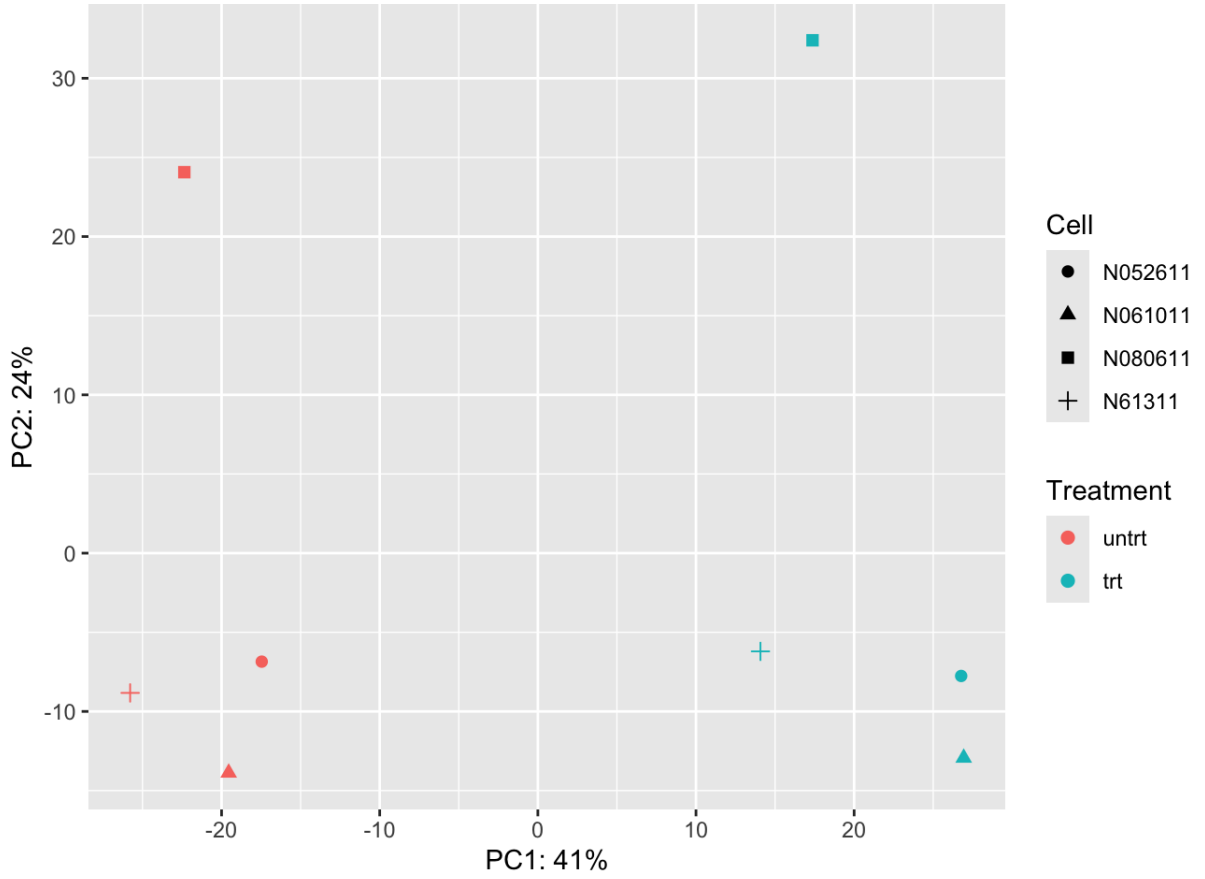
```
#Extract % Variance Explained
summary(pca)
```

```
Importance of components:
                          PC1     PC2     PC3     PC4     PC5     PC
Standard deviation     23.2778 17.7783 14.8386 12.0827 7.11804 5.662(
Proportion of Variance  0.4071  0.2375  0.1654  0.1097 0.03807 0.024(
Cumulative Proportion   0.4071  0.6446  0.8101  0.9197 0.95781 0.981(
                           PC8
Standard deviation     1.09e-13
Proportion of Variance 0.00e+00
Cumulative Proportion  1.00e+00
```

PC1 and PC2 combined account for 65% of variance in the data. We can add this information directly to our plot using custom axes labels.

Add custom axes labels

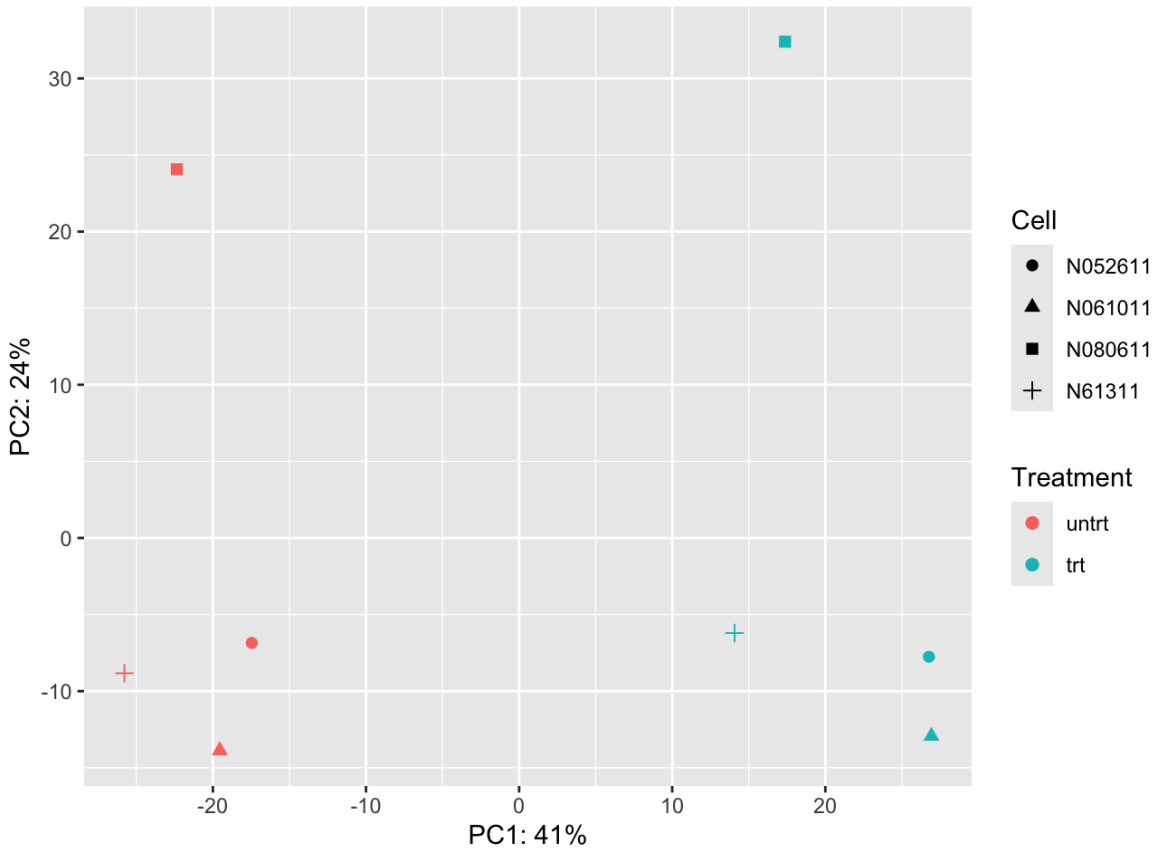
```
#Plot
ggplot(pcaData, aes(PC1, PC2, color = Treatment, shape = Cell)) +
  geom_point(size = 2) +
  coord_fixed() +
  xlab("PC1: 41%")+ #x axis label text
  ylab("PC2: 24%") # y axis label text
```



Automating the "Proportion of Variance"

If you want to automate the "Proportion of Variance", you should call it directly in the code. For example,

```
ggplot(pcaData, aes(PC1, PC2, color = Treatment, shape = Cell)) +
  geom_point(size = 2) +
  coord_fixed() +
  labs(x=paste0("PC1: ",round(summary(pca)$importance[2,1]*100),"%"),
       y=paste0("PC2: ",round(summary(pca)$importance[2,2]*100),"%"))
```



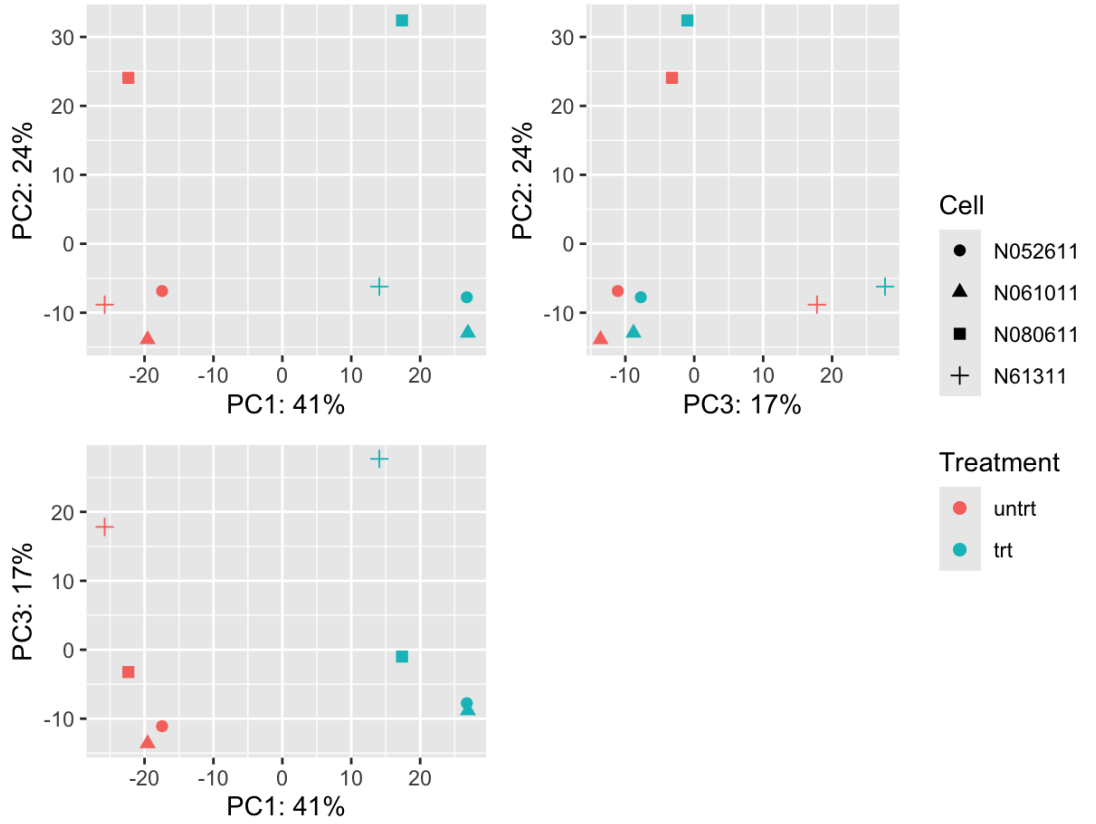
Add PC3

To add multiple principal components, you can build multiple plots

```
a<-ggplot(pcaData, aes(PC1, PC2, color = Treatment, shape = Cell)) +
  geom_point(size = 2) +
  coord_fixed() +
  labs(x=paste0("PC1: ",round(summary(pca)$importance[2,1]*100),"%"),
       y=paste0("PC2: ",round(summary(pca)$importance[2,2]*100),"%"))

b<-ggplot(pcaData, aes(PC3, PC2, color = Treatment, shape = Cell)) +
  geom_point(size = 2) +
  coord_fixed() +
  labs(x=paste0("PC3: ",round(summary(pca)$importance[2,3]*100),"%"),
       y=paste0("PC2: ",round(summary(pca)$importance[2,2]*100),"%"))

c<-ggplot(pcaData, aes(PC1, PC3, color = Treatment, shape = Cell)) +
  geom_point(size = 2) +
  coord_fixed() +
  labs(x=paste0("PC1: ",round(summary(pca)$importance[2,1]*100),"%"),
       y=paste0("PC3: ",round(summary(pca)$importance[2,3]*100),"%"))+
  theme(legend.position="none")


library(patchwork)
```

```
Warning: package 'patchwork' was built under R version 4.4.1
```

```
a + b + c + plot_layout(nrow=2,guides = 'collect')
```



A great package for simplifying this is `PCAtools` *(https://www.bioconductor.org/packages/devel/bioc/vignettes/PCAtools/inst/doc/PCAtools.html)*, using the function `pairsplot()`. This package has other functionality that also may be of interest, and it plays nicely with `ggplot2` functions for customization.

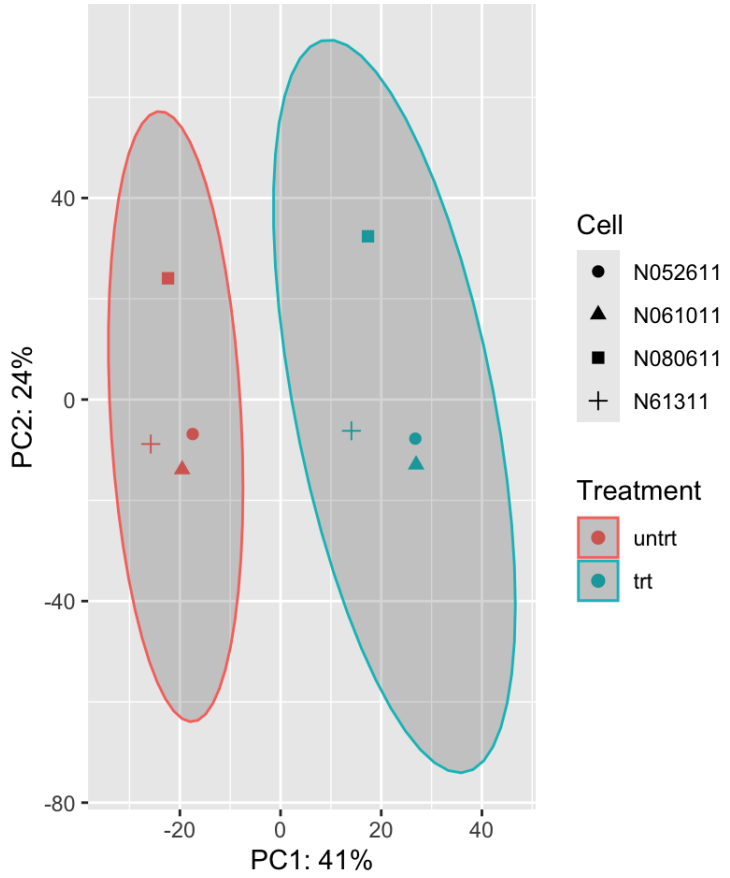
Add a stat to our plot with `stat_ellipse()`.

In scatter plots, the raw data is the focus of the plot, but for many other plots, this is not the case. You may wish to overlay a stat on your PCA. For example, ellipses are often added to PCA ordinations to emphasize group clustering with confidence intervals. By default, `stat_ellipse()` uses the bivariate t distribution, but this can be modified. Let's add ellipses with 95% confidence intervals to our plot.

```
ggplot(pcaData, aes(PC1, PC2, color = Treatment, shape = Cell)) +
  geom_point(size = 2) +
  coord_fixed() +
  labs(x=paste0("PC1: ",round(summary(pca)$importance[2,1]*100),"%")
       y=paste0("PC2: ",round(summary(pca)$importance[2,2]*100),"%")
  stat_ellipse(geom="polygon",aes(group=Treatment),
```

```
                level=0.95, alpha=0.2) #adding a stat
```

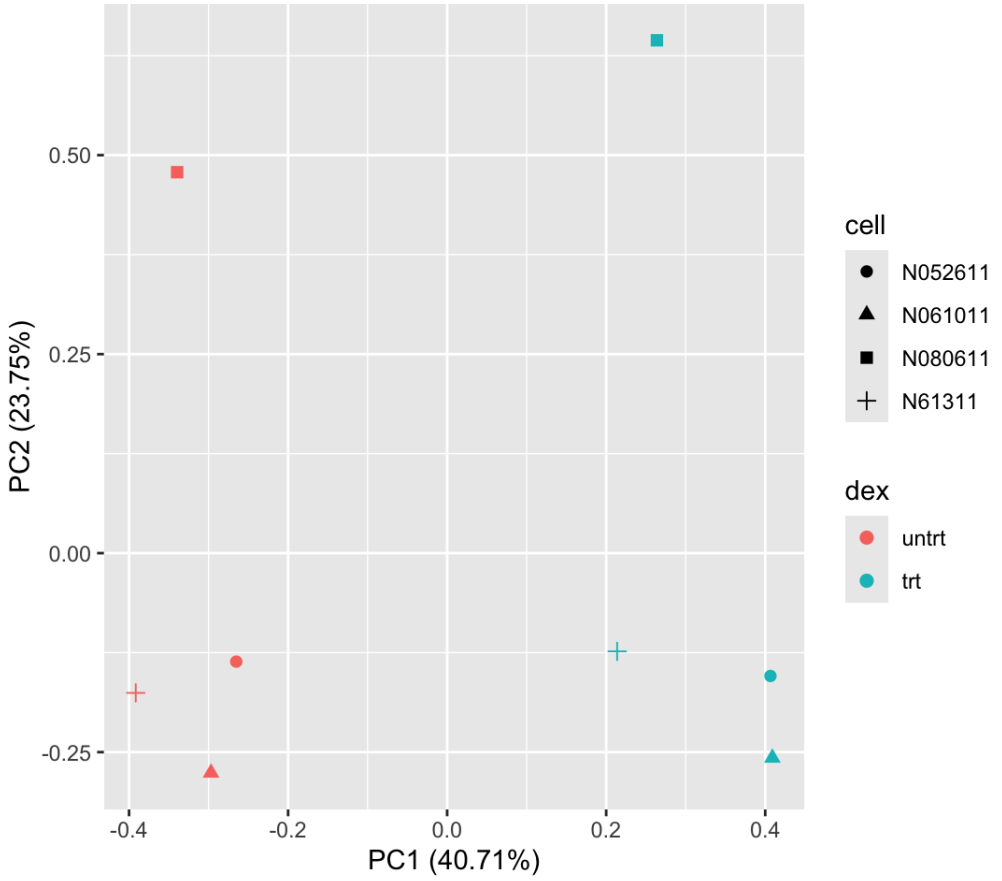
```
Warning: The following aesthetics were dropped during statistical tra
ℹ This can happen when ggplot fails to infer the correct grouping st
  the data.
ℹ Did you forget to specify a `group` aesthetic or to convert a nume
  variable into a factor?
```



Using ggfortify

`ggfortify` is an excellent package to consider for easily generating PCA plots. `ggfortify` provides "unified plotting tools for statistics commonly used, such as GLM, time series, PCA families, clustering and survival analysis. The package offers a single plotting interface for these analysis results and plots in a unified style using 'ggplot2'." *(https://cran.r-project.org/web/packages/ggfortify/)*

```
library(ggfortify)
autoplot(pca, data = colData(air), colour = 'dex',shape='cell',size=
  coord_fixed()
```



Since this is a ggplot2 object, this can easily be customized by adding ggplot2 customization layers.

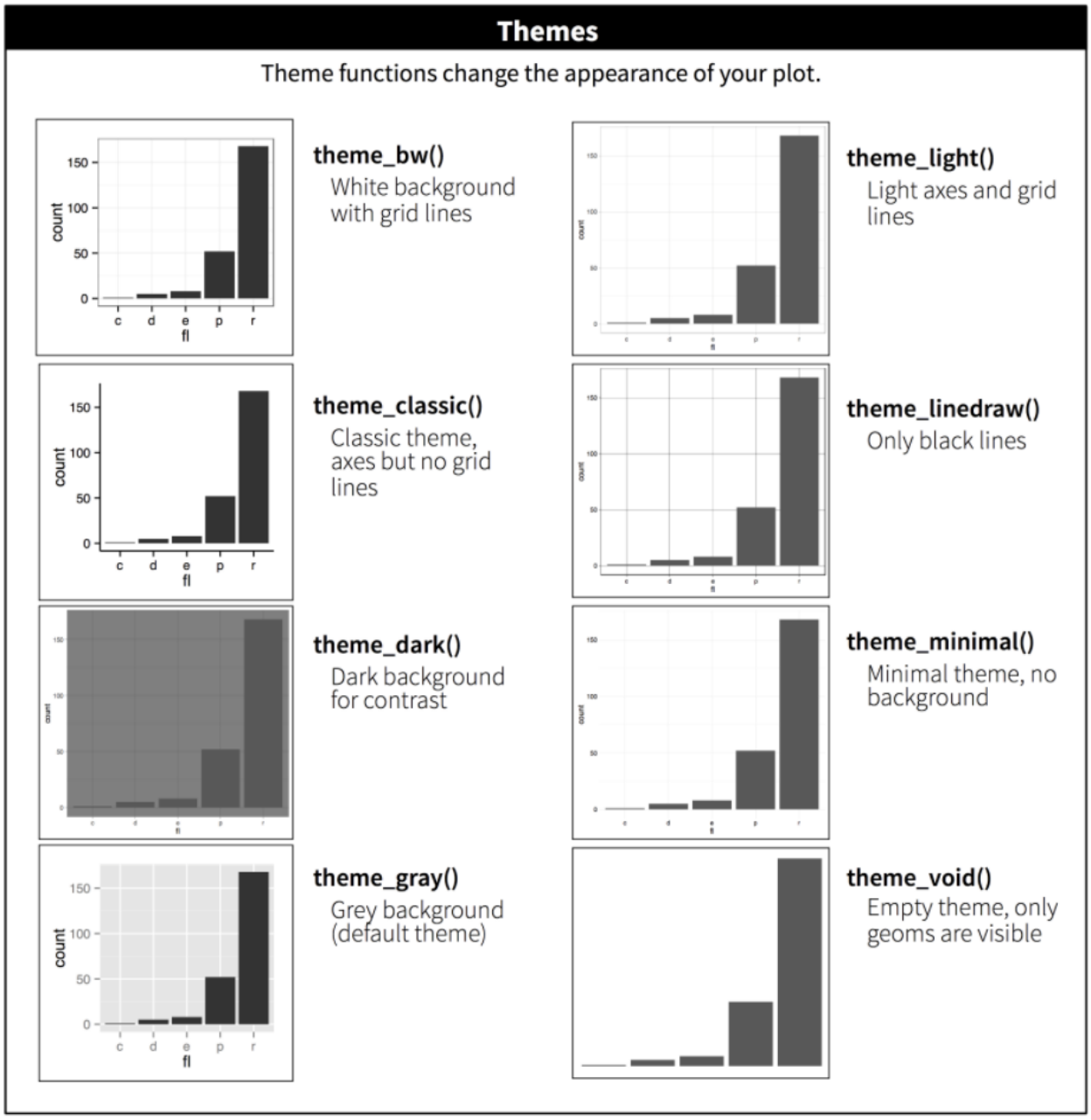
Using DESeq2 function

DESeq2 has its own function for PCA analysis, `plotPCA()`. If you check out the source code for this function (`getMethod("plotPCA","DESeqTransform")`), you will see that the data is filtered prior to `prcomp()` to only include the 500 top most variable genes by default.

If you are interesting in examining the PC loadings, these are difficult to plot using `ggplot2` alone. I recommend `PCAtools` or `factoextra` *(https://rpkgs.datanovia.com/factoextra/)*.

Plot Customization: Using themes

To create a publication quality plot, you will need to make several modifications to your basic PCA biplot code. We have already seen how to modify the default coordinate system, how to add additional statistics (e.g., ellipses with confidence intervals), and how to modify the axes labels. There are many more features that can be customized to make this publishable or fit a desired style.Changing non-data elements (related to axes, titles subtitles, gridlines, legends, etc.) of our plot can be done with `theme()`. GGplot2 has a definitive default style that falls under one of their precooked themes, `theme_gray()`. `theme_gray()` is one of eight complete themes provided by ggplot2.



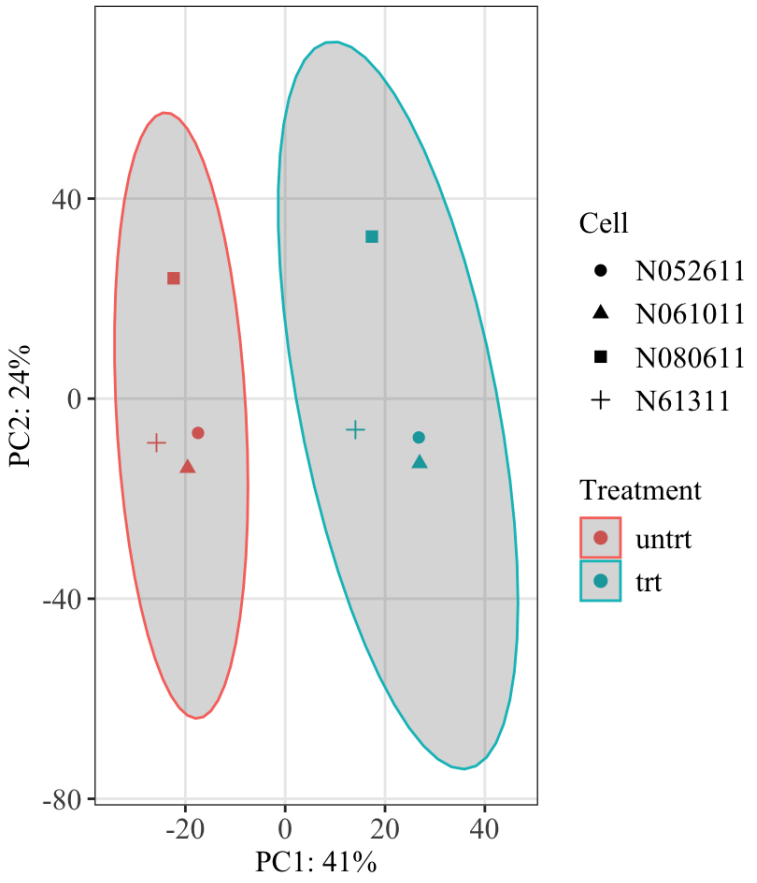
We can also specify and build a theme within our plot code or develop a custom theme to be reused across multiple plots. The theme function is the bread and butter of plot customization. Check out `?ggplot2::theme()` for a list of available parameters. There are many.

Let's see how this works by changing the fonts and text sizes and dropping minor grid lines:

```
ggplot(pcaData, aes(PC1, PC2, color = Treatment, shape = Cell)) +
  geom_point(size = 2) +
  coord_fixed() +
  labs(x=paste0("PC1: ",round(summary(pca)$importance[2,1]*100),"%")
       y=paste0("PC2: ",round(summary(pca)$importance[2,2]*100),"%")
  stat_ellipse(geom="polygon",aes(group=Treatment), level=0.95, alpha
theme_bw() + #start with a custom theme
  theme(axis.text=element_text(size=12,family="Times New Roman"),
```

```
        axis.title = element_text(size=12,family="Times New Roman"),
        legend.text = element_text(size=12,family="Times New Roman")
        legend.title = element_text(size=12,family="Times New Roman")
        panel.grid.minor = element_blank())
```

```
Warning: The following aesthetics were dropped during statistical tra
ℹ This can happen when ggplot fails to infer the correct grouping st
  the data.
ℹ Did you forget to specify a `group` aesthetic or to convert a nume
  variable into a factor?
```



You may want to establish a custom theme for reuse with a number of plots. See this great tutorial *(https://rpubs.com/mclaire19/ggplot2-custom-themes)* by Madeline Pickens for steps on how to do that.

A helpful color trick

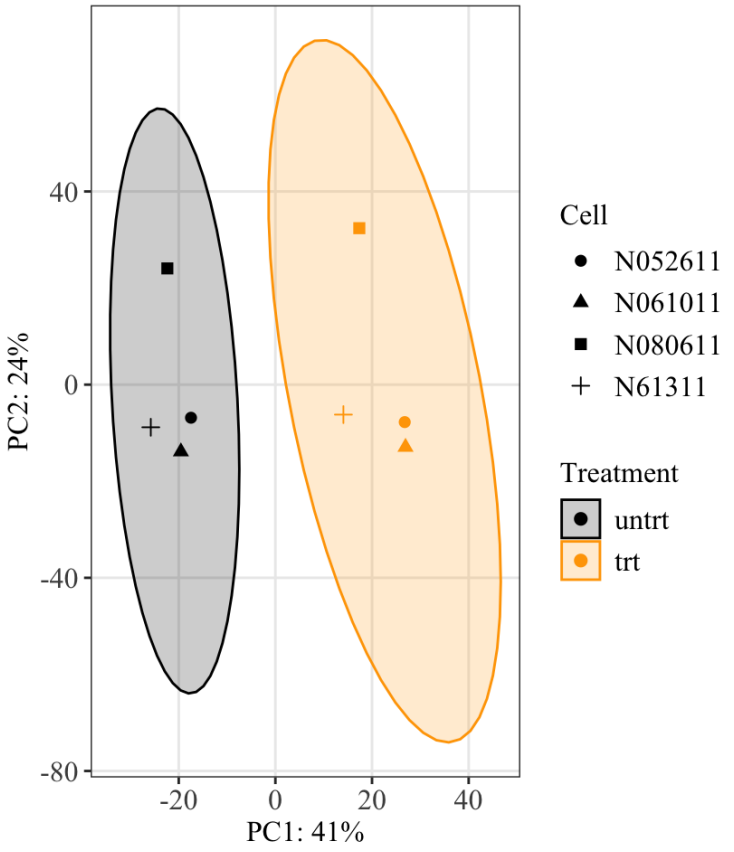
When you have a lot of colors and you want to keep these colors consistent, you can use the following convenient functions to set a name attribute for a vector of colors.

Let's do this for our asthma treatments.

```
#defining colors
```

```
gcolors<-setNames(c("black","orange"),levels(pcaData$Treatment))

#Now plot
ggplot(pcaData, aes(PC1, PC2, color = Treatment, shape = Cell)) +
  geom_point(size = 2) +
scale_color_manual(values=gcolors)+ #Adding the color argument
  coord_fixed() +
  labs(x=paste0("PC1: ",round(summary(pca)$importance[2,1]*100),"%"),
      y=paste0("PC2: ",round(summary(pca)$importance[2,2]*100),"%"))+
  stat_ellipse(geom="polygon",aes(group=Treatment,fill=Treatment),
               level=0.95, alpha=0.2)+
  scale_fill_manual(values=gcolors)+
theme_bw() + #start with a custom theme
  theme(axis.text=element_text(size=12,family="Times New Roman"),
        axis.title = element_text(size=12,family="Times New Roman"),
        legend.text = element_text(size=12,family="Times New Roman"),
        legend.title = element_text(size=12,family="Times New Roman"),
        panel.grid.minor = element_blank())
```

```
Warning: The following aesthetics were dropped during statistical transformation: s
ℹ This can happen when ggplot fails to infer the correct grouping structure in
  the data.
ℹ Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
```



We can use this color palette for all plots including these two treatments to keep our figures consistent throughout a presentation or publication.

Creating a publication ready volcano plot

Now that we know how to create a PCA biplot, let's use what we have learned to also make a volcano plot.

> A volcano plot is a type of scatterplot that shows statistical significance (P value) versus magnitude of change (fold change). It enables quick visual identification of genes with large fold changes that are also statistically significant. These may be the most biologically significant genes. --- Maria Doyle, 2021 *(https://training.galaxyproject.org/training-material/topics/transcriptomics/tutorials/rna-seq-viz-with-volcanoplot/tutorial.html)*

First, we need to do a bit of data wrangling. The `tidyverse` packages work great for this task.

```
#Wrangle the data

#add gene names to differential expression results
gene_names<- data.frame(rowData(air)) |> select(gene_name) |> tibble

dexp<- dexp |> tibble::rownames_to_column("ID") |>
      left_join(gene_names, by="ID")

# Call genes significantly differentially expressed if they have a
#p-value less than 0.05 and a logFC greater than or equal to 2.
dexp_sigtrnsc <- dexp |>
  mutate(Significant = padj < 0.05 & abs(log2FoldChange) >= 2) |>
  arrange(padj)

#extract top 6 genes to use for the labels
topgenes <- dexp_sigtrnsc$gene_name[c(1:6)]
```

Let's take a quick look at the wrangled data and our top genes for later labeling:

```
head(dexp_sigtrnsc)
```

```
                ID   baseMean log2FoldChange     lfcSE        pvalue
1 ENSG00000152583   997.9612       4.555676 0.1858858 7.129668e-136
2 ENSG00000165995   495.5675       3.276456 0.1326038 5.859407e-136
3 ENSG00000120129  3411.9661       2.935137 0.1227877 2.414560e-128
4 ENSG00000101347 12712.9456       3.754493 0.1579042 7.501584e-128
5 ENSG00000189221  2343.5733       3.337353 0.1432593 1.140346e-122
6 ENSG00000211445 12298.1966       3.711558 0.1693331 5.201708e-110
           padj gene_name Significant
```

```
1 5.530127e-132   SPARCL1        TRUE
2 5.530127e-132    CACNB2        TRUE
3 1.248569e-124     DUSP1        TRUE
4 2.909302e-124    SAMHD1        TRUE
5 3.538038e-119      MAOA        TRUE
6 1.344901e-106      GPX3        TRUE
```

```
topgenes
```

```
[1] "SPARCL1" "CACNB2"  "DUSP1"   "SAMHD1"  "MAOA"    "GPX3"
```

Significant differential expression was assigned based on an absolute log fold change greater than or equal to 2 and an FDR corrected p-value less than 0.05.

Enhanced Volcano

There is a dedicated package for creating volcano plots available on Bioconductor, `EnhancedVolcano` *(https://bioconductor.org/packages/release/bioc/html/EnhancedVolcano.html)*. Plots created using this package can be customized using ggplot2 functions and syntax.

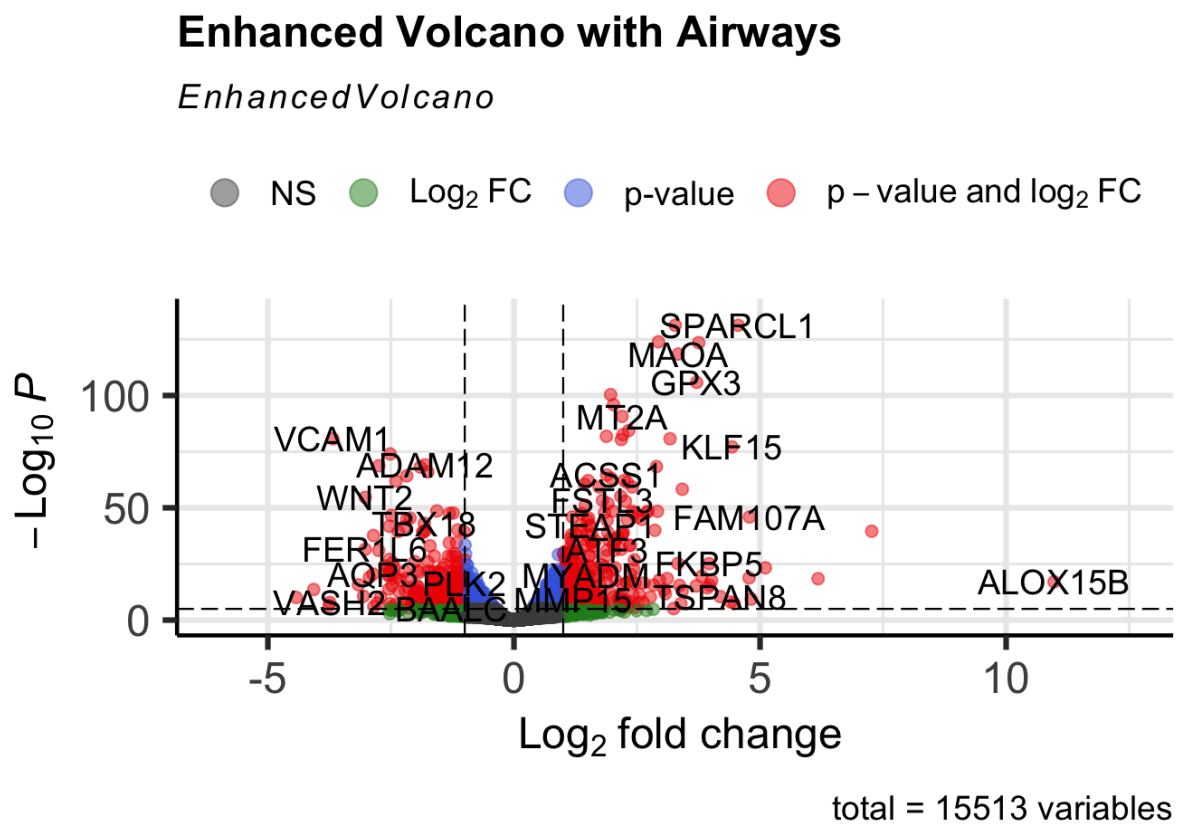
Using EnhancedVolcano:

```
#The default cut-off for log2FC is >|2|
#the default cut-off for P value is 10e-6
library(EnhancedVolcano)
```

```
Loading required package: ggrepel
```

```
Warning: package 'ggrepel' was built under R version 4.4.1
```

```
EnhancedVolcano(dexp,
                title = "Enhanced Volcano with Airways",
                lab = dexp_sigtrnsc$gene_name,
                x = 'log2FoldChange',
                y = 'padj')
```

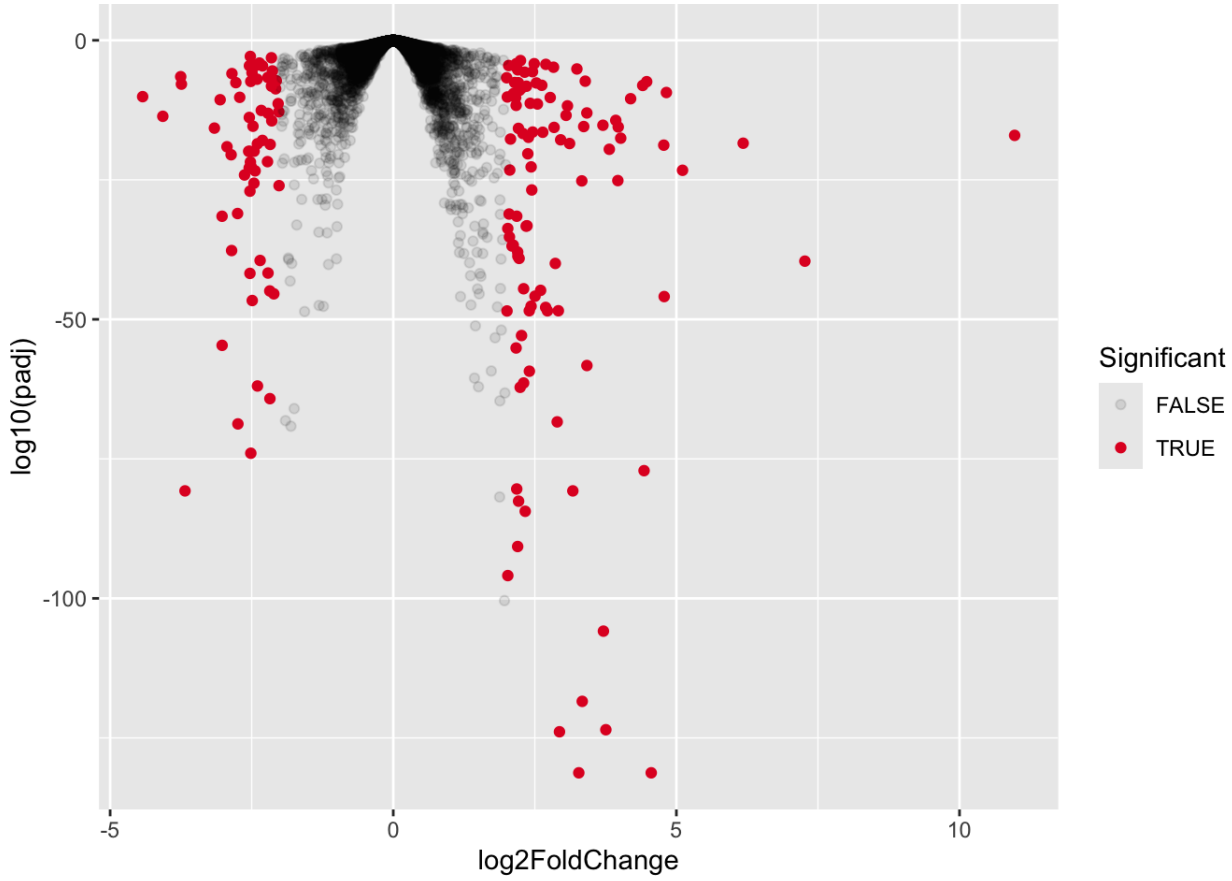


If you are interested in using this package, check out this tutorial *(https://bioinformatics.ccr.cancer.gov/docs/btep-coding-club/CC2023/complex_heatmap_enhanced_volcano/#background-on-volcano-plot)*.

Let's start our plot with the `<DATA>`, `<GEOM_FUNCTION>`, and `<MAPPING>`. We do not need to fix the coordinate system because we are working with two different values on the x and y and we don't need any special coordinate system modifications. Let's plot logFC on the x axis and the mutated column with our adjusted p-values on the y-axis and set the significant p-values off from the non-significant by color. We can also go ahead and customize the color scale, since we have learned how to do that.

```
ggplot(data=dexp_sigtrnsc) +
  geom_point(aes(x = log2FoldChange, y = log10(padj), color = Signif
                 alpha = Significant)) +
  scale_color_manual(values = c("black", "#e11f28"))
```

```
Warning: Using alpha for a discrete variable is not advised.
```



This is not exactly what we want so let's keep working.

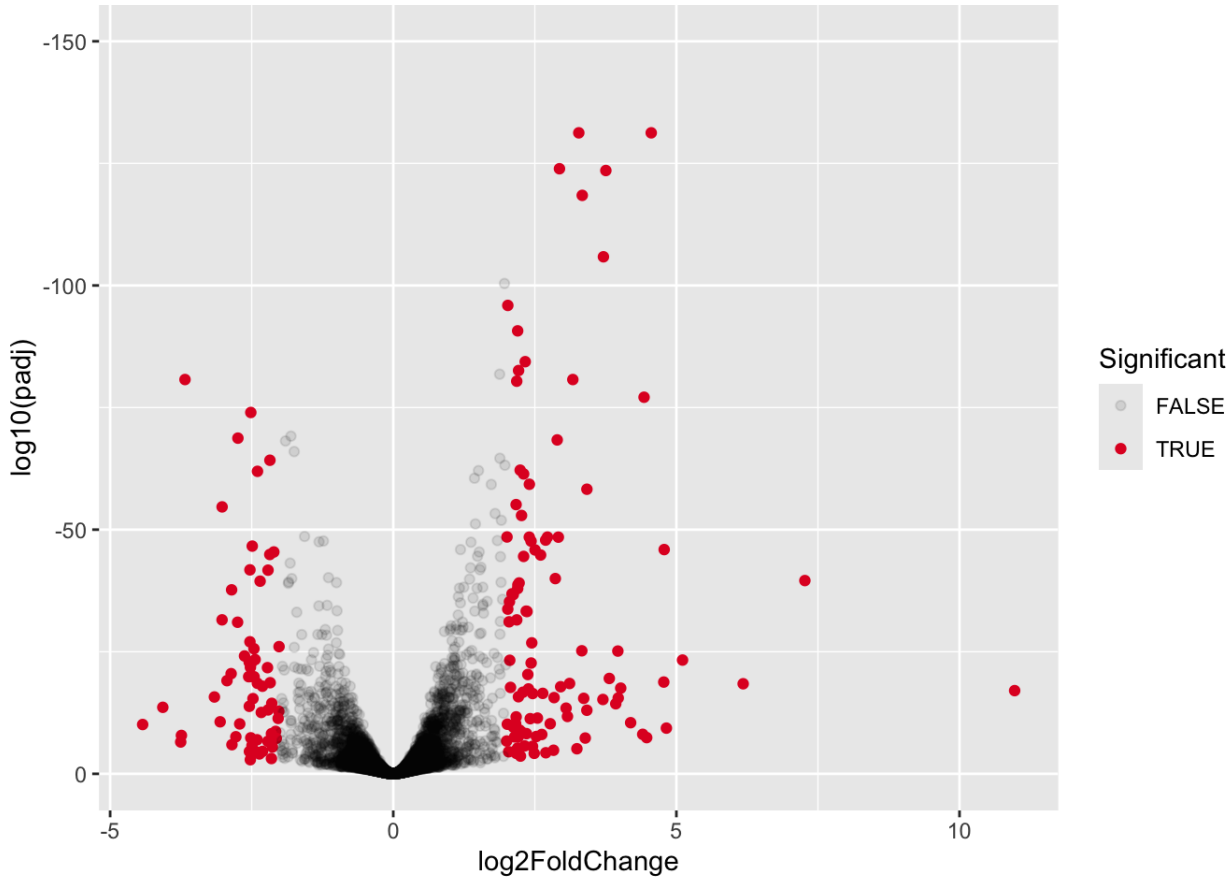
Immediately, you should notice that the figure is upside down compared to what we would expect from a volcano plot. There are two possible ways to fix this. We could transform the adjusted p-values by multiplying by -1 OR we could work with our axes scales. Aside from text modifications, we haven't yet changed the scaling of the axes. Let's see how we can modify the scale of the y-axis.

Changing axes scales

To change the y axis scale, we will need a specific function. These functions generally start with `scale_y...`. In our case we want to reverse our axis so that increasingly negative is going in the positive direction rather than the negative direction. Luckily, there is a function to reverse our axis; see `?scale_y_reverse()`.

```
ggplot(data=dexp_sigtrnsc) +
  geom_point(aes(x = log2FoldChange, y = log10(padj), color = Signif
                 alpha = Significant)) +
  scale_color_manual(values = c("black", "#e11f28")) +
  scale_y_reverse(limits=c(0,-150))
```

```
Warning: Using alpha for a discrete variable is not advised.
```



This looks pretty good, but we can tidy it up more by working with our legend guides and our theme.

Modifying legends

We can modify many aspects of the figure legend using the function `guide()`. Let's see how that works and go ahead and customize some theme arguments. Notice that the legend position is specified in `theme()`.

```
ggplot(data=dexp_sigtrnsc) +
  geom_point(aes(x = log2FoldChange, y = log10(padj), color = Signif
                 alpha = Significant)) +
  scale_color_manual(values = c("black", "#e11f28")) +
  scale_y_reverse(limits=c(0,-150))+
  guides(alpha= "none",
         color= guide_legend(title="Significant DE")) +
       theme_bw() +
      theme(
        panel.border = element_blank(),
        axis.line = element_line(),
```

```
        panel.grid.major = element_line(size = 0.2),
        panel.grid.minor = element_line(size = 0.1),
        text = element_text(size = 12),
        legend.position = "bottom",
        axis.text.x = element_text(angle = 30, hjust = 1, vjust = 1)
      )
```

```
Warning: The `size` argument of `element_line()` is deprecated as of
ℹ Please use the `linewidth` argument instead.
```

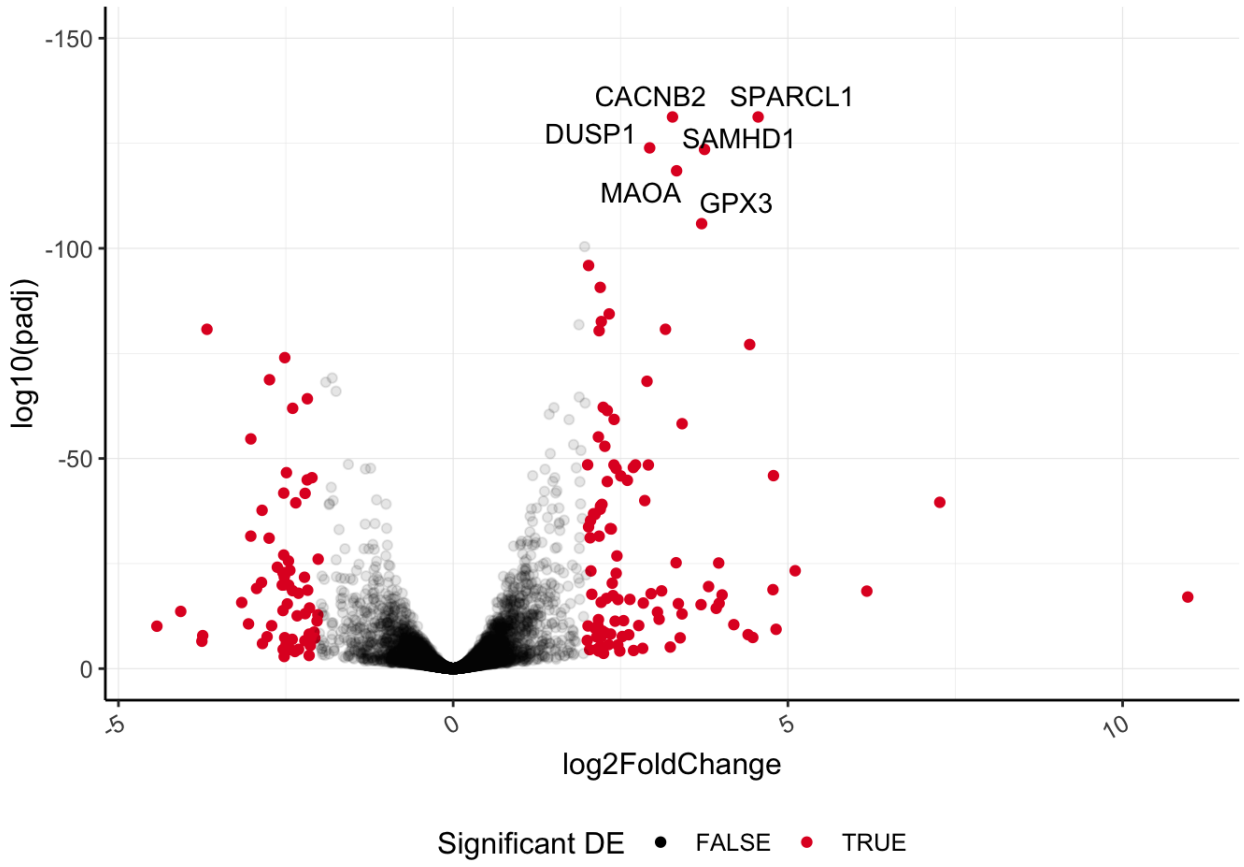
```
Warning: Using alpha for a discrete variable is not advised.
```

Lastly, let's layer another geom function to label our top six differentially abundant genes based on significance. We can use `geom_text_repel()` from `library(ggrepel)`, which is a variation on `geom_text()`.

```
ggplot(data=dexp_sigtrnsc) +
  geom_point(aes(x = log2FoldChange, y = log10(padj), color = Signif
                 alpha = Significant)) +
  geom_text_repel(data=dexp_sigtrnsc %>%
```

```
                    filter(gene_name %in% topgenes),
                  aes(x = log2FoldChange, y = log10(padj),label=gene_
                  nudge_y=0.5,hjust=0.5,direction="both",
                  segment.color="gray")+
  scale_color_manual(values = c("black", "#e11f28")) +
  scale_y_reverse(limits=c(0,-150))+
  guides(alpha= "none",
         color= guide_legend(title="Significant DE")) +
       theme_bw() +
      theme(
        panel.border = element_blank(),
        axis.line = element_line(),
        panel.grid.major = element_line(size = 0.2),
        panel.grid.minor = element_line(size = 0.1),
        text = element_text(size = 12),
        legend.position = "bottom",
        axis.text.x = element_text(angle = 30, hjust = 1, vjust = 1)
      )
```

```
Warning: Using alpha for a discrete variable is not advised.
```



To save the file, use `ggsave()`.

Session Info

```
sessionInfo()
```

```
R version 4.4.0 (2024-04-24)
Platform: aarch64-apple-darwin20
Running under: macOS Sonoma 14.7.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/

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: Europe/Berlin
tzcode source: internal

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  meth
[8] base

other attached packages:
 [1] EnhancedVolcano_1.22.0      ggrepel_0.9.6
 [3] ggfortify_0.4.17            patchwork_1.3.0
 [5] DESeq2_1.44.0               SummarizedExperiment_1.34.0
 [7] Biobase_2.64.0              MatrixGenerics_1.16.0
 [9] matrixStats_1.4.1           GenomicRanges_1.56.2
[11] GenomeInfoDb_1.40.1         IRanges_2.38.1
[13] S4Vectors_0.42.1            BiocGenerics_0.50.0
[15] dplyr_1.1.4                 ggplot2_3.5.1

loaded via a namespace (and not attached):
 [1] gtable_0.3.6            xfun_0.49               htmlwidgets_1.6
 [4] lattice_0.22-6          vctrs_0.6.5             tools_4.4.0
 [7] generics_0.1.3          parallel_4.4.0          tibble_3.2.1
[10] pkgconfig_2.0.3         Matrix_1.7-1            lifecycle_1.0.4
[13] GenomeInfoDbData_1.2.12 compiler_4.4.0          farver_2.1.2
[16] stringr_1.5.1           munsell_0.5.1           codetools_0.2-2
[19] htmltools_0.5.8.1       yaml_2.3.10             pillar_1.10.0
[22] crayon_1.5.3            tidyr_1.3.1             MASS_7.3-61
[25] BiocParallel_1.38.0     DelayedArray_0.30.1     abind_1.4-8
[28] tidyselect_1.2.1        locfit_1.5-9.10         digest_0.6.37
[31] stringi_1.8.4           purrr_1.0.2             labeling_0.4.3
```

```
[34] fastmap_1.2.0          grid_4.4.0             colorspace_2.1-1
[37] cli_3.6.3              SparseArray_1.4.8      magrittr_2.0.3
[40] S4Arrays_1.4.1         withr_3.0.2            scales_1.3.0
[43] UCSC.utils_1.0.0       rmarkdown_2.29         XVector_0.44.0
[46] httr_1.4.7             gridExtra_2.3          evaluate_1.0.1
[49] knitr_1.49             rlang_1.1.4            Rcpp_1.0.13-1
[52] glue_1.8.0             rstudioapi_0.17.1      jsonlite_1.8.9
[55] R6_2.5.1               zlibbioc_1.50.0
```

References

Sources for content included R4DS *(https://r4ds.had.co.nz/)* and Holmes and Huber, 2021 *(https://web.stanford.edu/class/bios221/book/)*.