

Data Visualization with R



Table of Contents

Course Overview

● Course Overview	10
● Welcome to the Data Visualization with R Series	10
● A series of lessons designed to introduce learners to the R package ggplot2	10
● Course objectives	10
● What this course is not!	10
● Course Expectations	10
● Lesson 1: Introduction to plot types	10
● Lesson 2: Getting Started with ggplot2	11
● Lesson 3: Scatter plots and Non-data elements of ggplot2 customization	11
● Lesson 4: Visualizing summary statistics with ggplot2	11
● Lesson 5: Visualizing clusters with heatmaps	11
● Lesson 6: Combining multiple plots to create a figure panel	11
● Required Course Materials	11

Lesson 1: Course Introduction

● Lesson 1: Course Introduction	12
---------------------------------	----

Lesson 2: The Basics of GGplot2

● Introduction to ggplot2	13
● Objectives	13

● Going beyond Excel	13
● Why ggplot2?	16
● Getting started with ggplot2	16
● Getting help	17
● The ggplot2 template	17
● Geom functions	20
● Mapping and aesthetics (aes())	21
● R objects can also store figures	24
● Colors	24
● Facets	30
● A better example of facet_grid() using data("Titanic").	33
● Building upon our template	34
● Using multiple geoms per plot	35
● Labels, legends, scales, and themes	36
● Saving plots (ggsave())	37
● Resource list	38
● Acknowledgements	38

Lesson 3: Scatter plots and ggplot2 customization

● Scatter plots and plot customization	39
● Objectives	39
● The Data	39
● The grammar of graphics	40
● Scatter plots	41
● Basic Scatter	41
● Coordinate Systems	42

● Perform and plot PCA data using iris.	43
● What is PCA?	43
● Perform PCA	44
● Plot PCA	45
● Add custom axes labels	47
● Add a stat to our plot with <code>stat_ellipse()</code> .	48
● Plot customization	49
● Themes	49
● A helpful color trick	51
● Putting it all together	53
● Creating a publication ready volcano plot	53
● Changing axes scales	56
● Modifying legends	57
● Save to an <code>.rds</code> file	60
● References	61

Lesson 4: Stat Transformations: Bar plots, box plots, and histograms

● Stat Transformations: Bar plots, box plots, and histograms	62
● Our grammar of graphics template	62
● Our grammar of graphics template	62
● Libraries	63
● The Data	63
● Variable Types	64
● The stat argument	65
● Bar Plot	66
● <code>stat = count</code>	67
● <code>stat = identity</code>	69

● Using factors	71
● Adding error bars	73
● Histogram	75
● Box plot	78

Lesson5: Visualizing clusters with heatmap and dendrogram

● Visualizing clusters with heatmaps	81
● What is a heatmap?	81
● Heatmap can be used to visualize the following	81
● What is a dendrogram?	81
● Applications of dendrograms	81
● Methods available to produce a heatmap in R	84
● Load the libraries	85
● Import data	85
● Heatmap of the top 20 genes from differential expression analysis	87
● Distance, clustering, and scaling	88
● Distance calculation	88
● Cluster generation	88
● Scaling	88
● Working with color customization	92
● Adding treatment group information to samples (annotation_col, annotation_colors)	93
● Add the annotations column	94
● Modify the annotations colors	95
● Splitting heatmap into multiple columns and rows	96
● Add a title to heatmap	98
● Saving a non-ggplot	100

● Make this plot compatible with ggplot2 using the package ggplotify.	102
● Save as an R object	102

Lesson 6: Multi-figure panel

● Objectives	105
● Why do we need to learn to combine figures?	105
● Example journals and their figure limits:	105
● Load the libraries	106
● The Data	107
● What is cowplot?	111
● Using cowplot to arrange figures	112
● Plotting labels with cowplot	113
● Other notable features of cowplot	114
● Taking advantage of ggdraw	115
● Using cowplot we can get an image such as this:	117
● What is patchwork?	118
● Combining two plots	119
● Using plot_layout()	125
● Adding a spacer	129
● Add our cowplot image	130
● Including a title	131
● Save the plot	132
● Which package to use?	133
● Acknowledgements	133

Getting the Data

Course Data	135
-------------	-----

Practice Questions

Practice plotting using ggplot2: Lesson 2	137
---	-----

● Load the data	137
● Exercise Questions	137
● Question 1	137
● Question 2	138
● Question 3	139
● Question 4	139
● Question 5	140
● Question 6	141
● Challenge question 1	142
● Challenge question 2	143
● Want more practice?	145
● Question 1	145
● Question 2	145
● Question 3	146
● Question 4	147
● Question 5	148
● Question 6	149

Practice plotting using ggplot2: Lesson 3 151

● Question 1	151
● Question 2	152
● Question 3	153
● Question 4	155
● Question 5	156

Lesson 4: Stat Transformations: Bar plots, box plots, and histograms 157

● Activate packages	157
● Load the mtcars dataset using the code below. This is a dataset that comes with R.	157
● Question 1	157
● Question 2	158
● Question 3	159
● Question 4	160

Lesson5: Visualizing clusters with heatmap and dendrogram 162

● Load necessary packages	162
● Question 1	162
● Question 2	163
● Question 3	163
● Question 4	164
● Question 5	165

Additional Resources

For Further Reading 167

● Getting started with R	167
● General help with ggplot2	167
● Troubleshooting error messages	167

● Other Resources	167
-------------------	-----

DNAexus (Class Registrants Only)

Navigating DNAexus	169
---------------------------	------------

● Instruction for using DNAexus for the Intro to R class	169
--	-----

DNAexus Outside of Class	173
---------------------------------	------------

● Setting up the R-Studio environment outside of Class hours	173
--	-----

● Log into DNAexus	173
--------------------	-----

● To Start the Application	174
----------------------------	-----

● Open the R-Studio Web interface	176
-----------------------------------	-----

● Proceed with running content from the R Class	178
---	-----

Course Overview

Welcome to the Data Visualization with R Series

A series of lessons designed to introduce learners to the R package `ggplot2`

This course will include a series of lessons for scientists with beginner level experience in R. The primary purpose of this course is to introduce learners to data visualization in R with a primary focus on `ggplot2` and related packages.

Course objectives

1. Learn how to generate basic plot types in `ggplot2`
2. Understand how basic plot types can be customized to generate more complex plots

What this course is not!

This course will not:

1. Make anyone an R expert
2. Make anyone a `ggplot2` expert

Note

While this course may be useful to learners with intermediate R experience who would like to learn more regarding `ggplot2`, the pace of the course will be set assuming a beginner level of understanding.

Course Expectations

This course will include a series of six, 1-1.25 hour lessons over a period of three weeks. Each lesson will be followed by a 45 minute help session in which students can ask questions and / or get individual help with their data.

Lesson 1: Introduction to plot types

This lesson will answer that burning question: Why R for data visualization? In addition, we will introduce the various plot types that will be generated throughout the course and will showcase related plots that you will be able to create in the future using the foundational skills gained over the next 3 weeks. This will not be a hands-on lesson so no coding yet. The hands-on portion of

this series will start with lesson 2. In the help session afterwards we will help those having trouble with DNAnexus accounts.

Lesson 2: Getting Started with ggplot2

Lesson 2 will focus on the basics of ggplot2, including the grammar of graphics philosophy and its application. This lesson will provide a hands on introduction to the ggplot2 syntax, geom functions, mapping and aesthetics, and plot layering.

Lesson 3: Scatter plots and Non-data elements of ggplot2 customization

Lesson 3 will continue the discussion on the grammar of graphics, with a focus on ggplot2 plot customization including axes labels, coordinate systems, axes scales, and themes. This hands on lesson will showcase these features of plot building through the generation of increasingly complex scatter plots using data included with a base R installation as well as RNASeq data.

Lesson 4: Visualizing summary statistics with ggplot2

It is common to obtain summary statistics for a dataset to understand parameters like mean, standard deviation, and distribution. In this lesson, we will learn to generate plots that will help with visualization of summary statistics including bar plot with error bars, histogram, as well as the box and whiskers plot.

Lesson 5: Visualizing clusters with heatmaps

Lesson 5 will introduce the heatmap and dendrogram as tools for visualizing clusters in data. This lesson will primarily use the R package pheatmap.

Lesson 6: Combining multiple plots to create a figure panel

Scientific journals almost always have limits on the number of figures that can be included in a publication. Don't fret, in lesson 6, we will focus on generating sub plots and multi plot figure panels using ggplot2 associated packages.

Required Course Materials

To participate in this class you will need your government-issued computer and a reliable internet connection. You do not need to download or install any software to participate in the class.

Lesson 1: Course Introduction

Introduction to ggplot2

Objectives

1. Learn how to import spreadsheet data.
2. Learn the ggplot2 syntax.
3. Build a ggplot2 general template.

By the end of the course, students should be able to create simple, pretty, and effective figures.

Going beyond Excel

Excel is a great program for visualizing and manipulating small data sets. However, it isn't great for working with "big data", and resulting plots are generally not publishable. Learning R and associated plotting packages is a great way to generate publishable figures in a reproducible fashion. Using R will not only keep you from accidentally editing your data, but it will also allow you to generate scripts that can be viewed later or reused to generate the same plot using different data. This will keep you from having to rely on your memory when wondering what data was used or how a plot was generated.

Let's read in and look at some excel data.

```
#If the package readxl is not installed,  
#you will need to install the package and load the library  
#install.packages("readxl")  
#library(readxl)  
  
#data import from excel  
data<-readxl::read_xlsx(  
  "./data/RNASeq_totalcounts_vs_totaltrans.xlsx",1)  
  
#View data  
data
```

```
## # A tibble: 8 × 4  
##   `Sample Name` Treatment   `Number of Transcripts` `Total Count`  
##   <chr>          <chr>                <dbl>         <dbl>  
## 1 GSM1275863    Dexamethasone         10768         18783  
## 2 GSM1275867    Dexamethasone         10051         15144
```

##	3	GSM1275871	Dexamethasone	11658	307760
##	4	GSM1275875	Dexamethasone	10900	211351
##	5	GSM1275862	None	11177	206084
##	6	GSM1275866	None	11526	253111
##	7	GSM1275870	None	11425	244118
##	8	GSM1275874	None	11000	190941

These data include total transcript read counts summed by sample and the total number of transcripts recovered by sample that had at least 100 reads. These data derive from a bulk RNAseq experiment described by Himes et al. (2014) (<https://pubmed.ncbi.nlm.nih.gov/24926665/>). 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. Each cell line included a treated and untreated negative control resulting in a total sample size of 8.

Notice those column names.

```
#Notice those column names
colnames(data)
```

```
## [1] "Sample Name"          "Treatment"            "Number of Tra
## [4] "Total Counts"
```

Spaces cause problems for data wrangling in R, but we can change our load parameters to repair our column names.

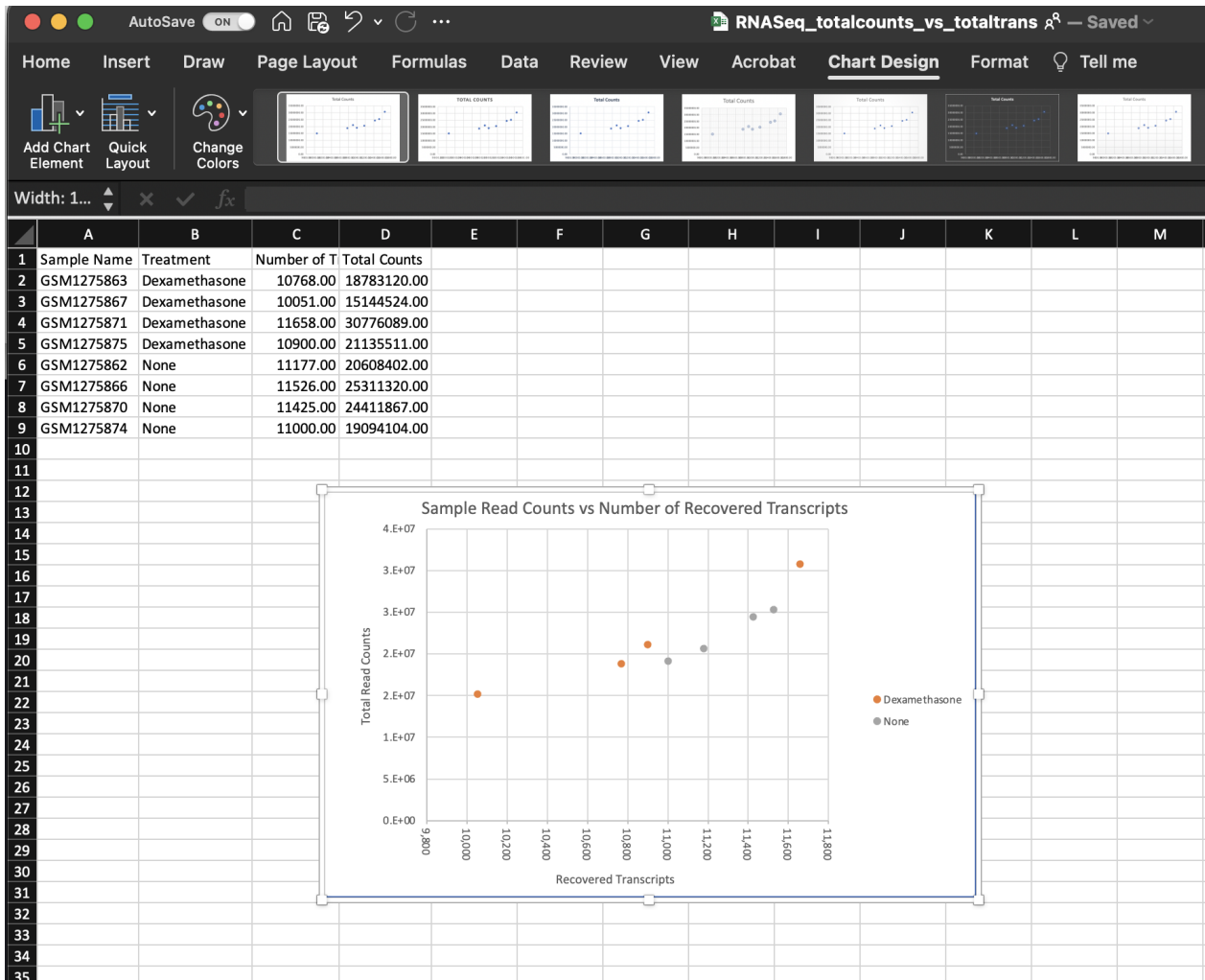
```
#data import from excel
data<-readxl::read_xlsx("./data/RNASeq_totalcounts_vs_totaltrans.xlsx",
                        1,.name_repair = "universal")
```

```
## New names:
## • `Sample Name` -> `Sample.Name`
## • `Number of Transcripts` -> `Number.of.Transcripts`
## • `Total Counts` -> `Total.Counts`
```

Readxl's default is `.name_repair = "unique"`, which ensures each column has a unique name. If that is already true of the column names, readxl won't touch them. The value `.name_repair = "universal"` goes further and makes column names syntactic, i.e. makes sure they don't contain any forbidden characters or reserved words. This makes life easier if you use packages like ggplot2 and dplyr downstream, because the column names will "just work" everywhere and won't

require protection via backtick quotes --- [readxl.tidyverse.org \(https://readxl.tidyverse.org/articles/column-names.html\)](https://readxl.tidyverse.org/articles/column-names.html).

We could plot this data in excel. If we did, we would get something like this:



This isn't too bad, but it took an unnecessary amount of time, and there weren't a lot of options for customization.

RECOMMENDATION

You should save metadata or other tabular data as either comma separated files (.csv) or tab-delimited files (.txt, .tsv). Using these file extensions will make it easier to use the data with bioinformatic programs. There are multiple functions available to read in delimited data in R. We will see a few of these over the next few weeks.

Example of .csv structure:

```

ensembl_id", "baseMean", "log2FoldChange", "lfcSE", "stat", "pvalue", "padj"
"ENSG00000000003", 708.597861536998, -0.381227063105209, 0.1007022818092, -3.78568445775159, 0.000153286081123382, 0.00128920457780635
"ENSG000000000419", 520.296296925274, 0.206840376248013, 0.112107724676557, 1.84501448802721, 0.0650354311057282, 0.194929521593848
"ENSG000000000457", 237.162103834464, 0.0379543353898249, 0.142823085132158, 0.265743702110235, 0.790436603692329, 0.909899502487221
"ENSG000000000460", 57.9323803212894, -0.0885309218784108, 0.284934403153467, -0.31070632713568, 0.756023886817396, 0.892993575626971
"ENSG000000000971", 5817.31081674539, 0.426424557319071, 0.0888056149270271, 4.801774726400669, 1.57265558569392e-06, 2.06391015438375e-05
"ENSG000000001036", 1282.10068817841, -0.241042961091579, 0.0889938344862711, -2.70853551240973, 0.00675808792114566, 0.0334601431614768
"ENSG000000001084", 609.888711231609, -0.0475897411106851, 0.166635766983845, -0.2855913947652, 0.775191095061154, 0.902579313368808
"ENSG000000001167", 369.340563608024, -0.500318601976352, 0.12056788044641, -4.14968397987832, 3.32934600825575e-05, 0.000329680215380932
"ENSG000000001460", 183.236892721017, -0.123909912875758, 0.179048826698894, -0.692045377566964, 0.488908832560624, 0.720497226931445
"ENSG000000001461", 2813.83742255416, -0.041034566859936, 0.102975768420967, 0.398487600424461, 0.690270796575656, 0.85693376962838
"ENSG000000001497", 539.821574730882, -0.00402111211466768, 0.106625405199165, -0.0377125142657762, 0.969916898154282, 0.988355229517773
"ENSG000000001561", 114.037077799749, 0.225086772189483, 0.215180413227571, 1.04603745672444, 0.295543739121845, 0.54374499580095
"ENSG000000001617", 598.664307018364, -0.250209768400065, 0.108241908382303, -2.31157942556169, 0.0208008715316545, 0.0819640770243872
"ENSG000000001626", 5.76493538185727, -1.08669601760299, 0.785342536964404, -1.38372234592489, 0.166443488535276, 0.379922103197466
"ENSG000000001629", 1779.81786317606, -0.226647248185764, 0.0809829657752523, -2.79870273971894, 0.00513083480037969, 0.0267242094513133
"ENSG000000001630", 54.727467682366, 0.0126423916643575, 0.310248740605993, 0.0407492118732336, 0.967495828750317, 0.987093175513506
"ENSG000000001631", 785.449616737739, -0.0366621302642643, 0.103319977528239, -0.354840672068905, 0.722708956583117, 0.875210739093806
"ENSG000000002016", 183.859639860491, -0.310481199896657, 0.159095625501695, -1.95153825831212, 0.0509930497165724, 0.16271574455658
"ENSG000000002079", 1.42772730704924, -1.97816123737037, 1.82334800173291, -1.08490602753304, 0.277963278103908, NA
"ENSG000000002330", 178.056083595009, 0.089027051723702, 0.154226485224129, 0.577248788327919, 0.563771403294639, 0.775299014523036
"ENSG000000002549", 1253.6302802743, 0.186692401461012, 0.10666018060959, 1.76015282626821, 0.0783818979497371, 0.22297518336777
"ENSG000000002586", 8285.39793029793, 0.08978144077015, 0.0809928752076506, 1.10851046514854, 0.267641422046851, 0.512310608632904
"ENSG000000002745", 6.89982954221215, -0.268215790154203, 0.770920108901835, -0.347916453413406, 0.727902928286622, 0.877987797979603
"ENSG000000002746", 108.44215151566, 0.10248912688139, 0.197661473824769, 0.518508361281617, 0.604103627865951, 0.803807306499323
"ENSG000000002822", 349.806186146653, -0.16017187802095, 0.132107891171917, -1.21243232785021, 0.225346932319337, 0.460041188418529
"ENSG000000002834", 7168.80172554647, 0.396667977907531, 0.0934251755172931, 4.24583604698828, 2.1779838631325e-05, 0.00022660161859204
"ENSG000000002919", 250.507676868456, 0.40442197106257, 0.139588471820168, 2.89724477809017, 0.00376455904694788, 0.0205083168382369
"ENSG000000002933", 540.277871458637, 0.361113197620368, 0.146306758604392, 2.46819218103795, 0.0135797390922249, 0.0583718367845051
"ENSG000000003056", 1065.26652768511, -0.072541165347929, 0.0923741119491342, -0.785297566788775, 0.432279122106419, 0.674820979120678
"ENSG000000003096", 377.975261226872, -0.940044816014956, 0.143511098727286, -6.55659962441673, 5.50485486004384e-11, 1.3844287703809e-09
"ENSG000000003137", 104.515194808615, -0.897047973877925, 0.291001052723961, -3.08262793375132, 0.00205181514216274, 0.0123286673253844
"ENSG000000003249", 116.349026825881, -0.280248279024878, 0.193329638527801, -1.44958776708507, 0.147173509317249, 0.350526049400164
"ENSG000000003393", 851.090875287858, 0.171929520433662, 0.0958496846240396, 1.79374111775158, 0.0728544991445922, 0.211581950165129
"ENSG000000003400", 80.1567179115993, -0.407996077954461, 0.225638600633938, -1.8081838692856, 0.070577888314414, 0.206869389155643
"ENSG000000003402", 2546.60932638954, 1.19046752343508, 0.121018689324023, 9.83705516961643, 7.79593500262112e-23, 6.509892628954291e-21
"ENSG000000003436", 4469.59766637531, -0.196579386205128, 0.1502454008959662, -1.30838863940199, 0.190741523945396, 0.414691482973641
"ENSG000000003509", 290.814036626452, -0.0510529401567553, 0.135437417484113, -0.376948564917403, 0.706211829323584, 0.865579914788123
"ENSG000000003756", 2117.4540023601, -0.188724021052601, 0.095097090045001, 1.07621677926116, 0.048118010487648, 0.15613744886641

```

Why ggplot2?

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). 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. GGplot2 is incredibly versatile and can create most types of plots, especially when you consider the [numerous packages \(https://exts.ggplot2.tidyverse.org/gallery/\)](https://exts.ggplot2.tidyverse.org/gallery/) that further extend its capabilities.

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 ("tidy data (<https://r4ds.had.co.nz/tidy-data.html>)").

Getting started with ggplot2

To begin plotting, we need to load our ggplot2 library. Package libraries must be loaded 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
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.

Note

You may also find [Microsoft's Bing GPT-4 chatbot](https://www.bing.com/new?toWww=1&redig=7AA1192E6A3E4A04938B4E1AB3C98A19) (<https://www.bing.com/new?toWww=1&redig=7AA1192E6A3E4A04938B4E1AB3C98A19>) useful for creating code examples that can be modified to fit your data. Currently, Bing is the only GPT-4 browser that is authorized for use by NCI.

Though it was created for ChatGPT, you may find [this resource from Datacamp](https://www.datacamp.com/cheat-sheet/chatgpt-cheat-sheet-data-science) (<https://www.datacamp.com/cheat-sheet/chatgpt-cheat-sheet-data-science>) useful for prompting appropriate responses.

The ggplot2 template

The following represents the basic ggplot2 template.

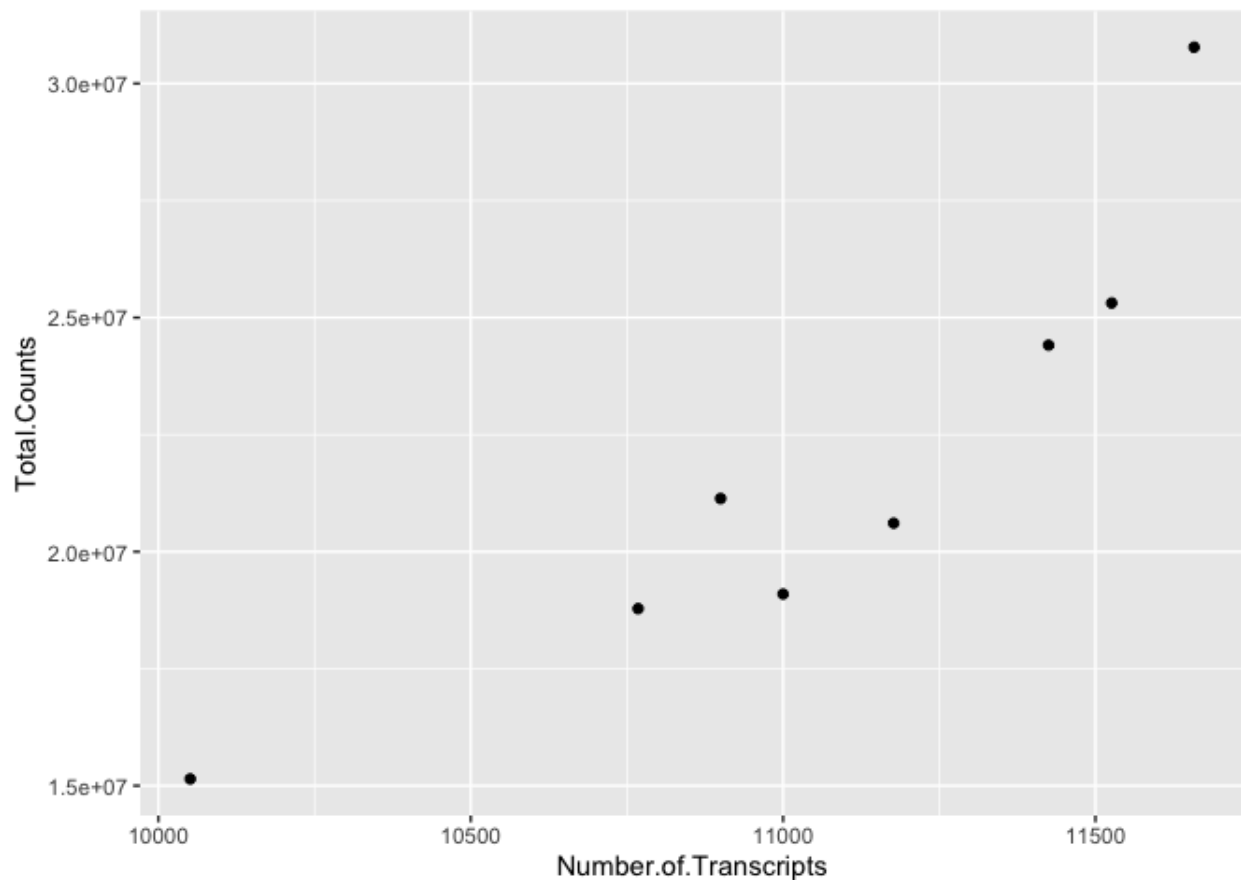
```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

The main components include 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.

Let's see this template in practice.

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

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



We can easily see that there is a relationship between the number of transcripts per sample and the total transcripts recovered per sample. `ggplot2` default parameters are great for exploratory data analysis. But, with only a few tweaks, we can make some beautiful, publishable figures.

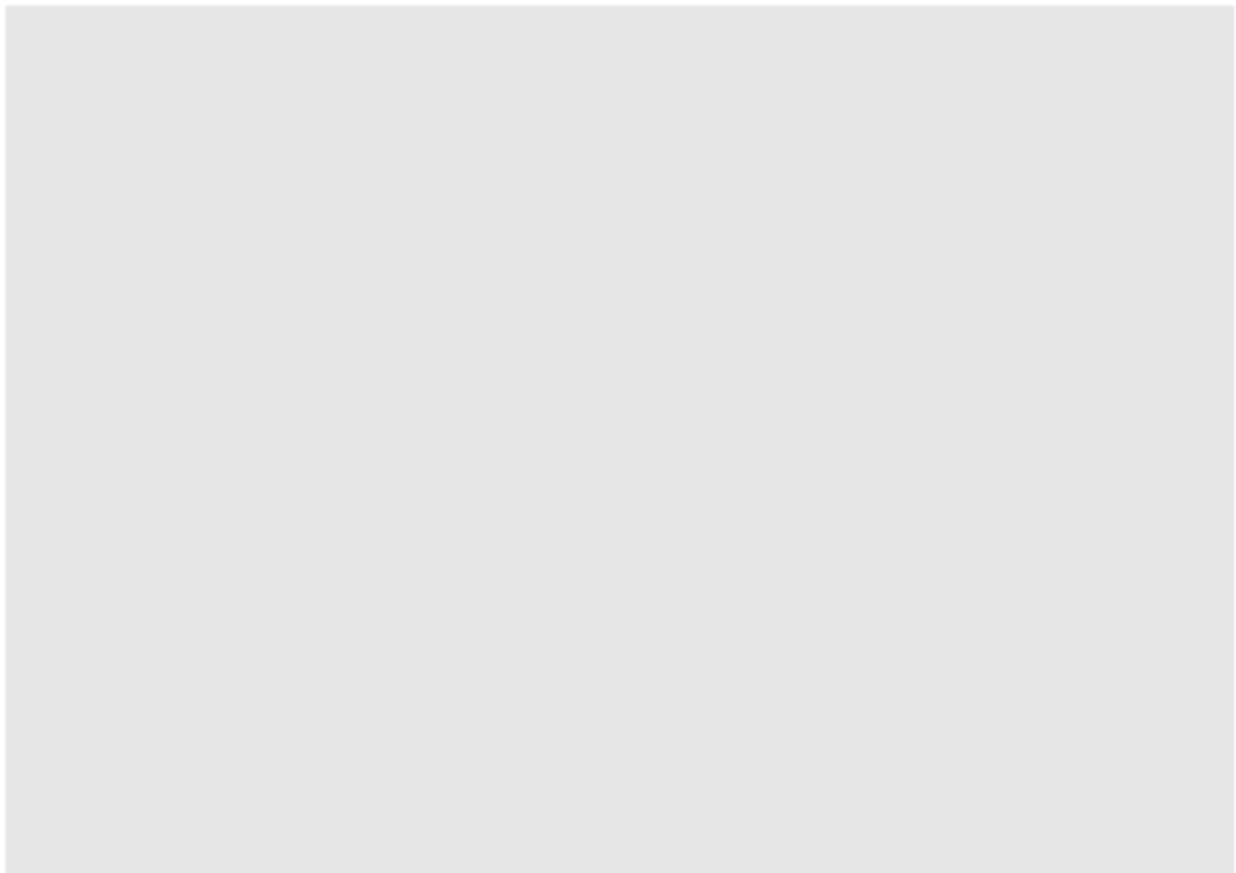
Let's take a closer look at the above code

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 `data`, 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.

Let's break down the individual components of the code.

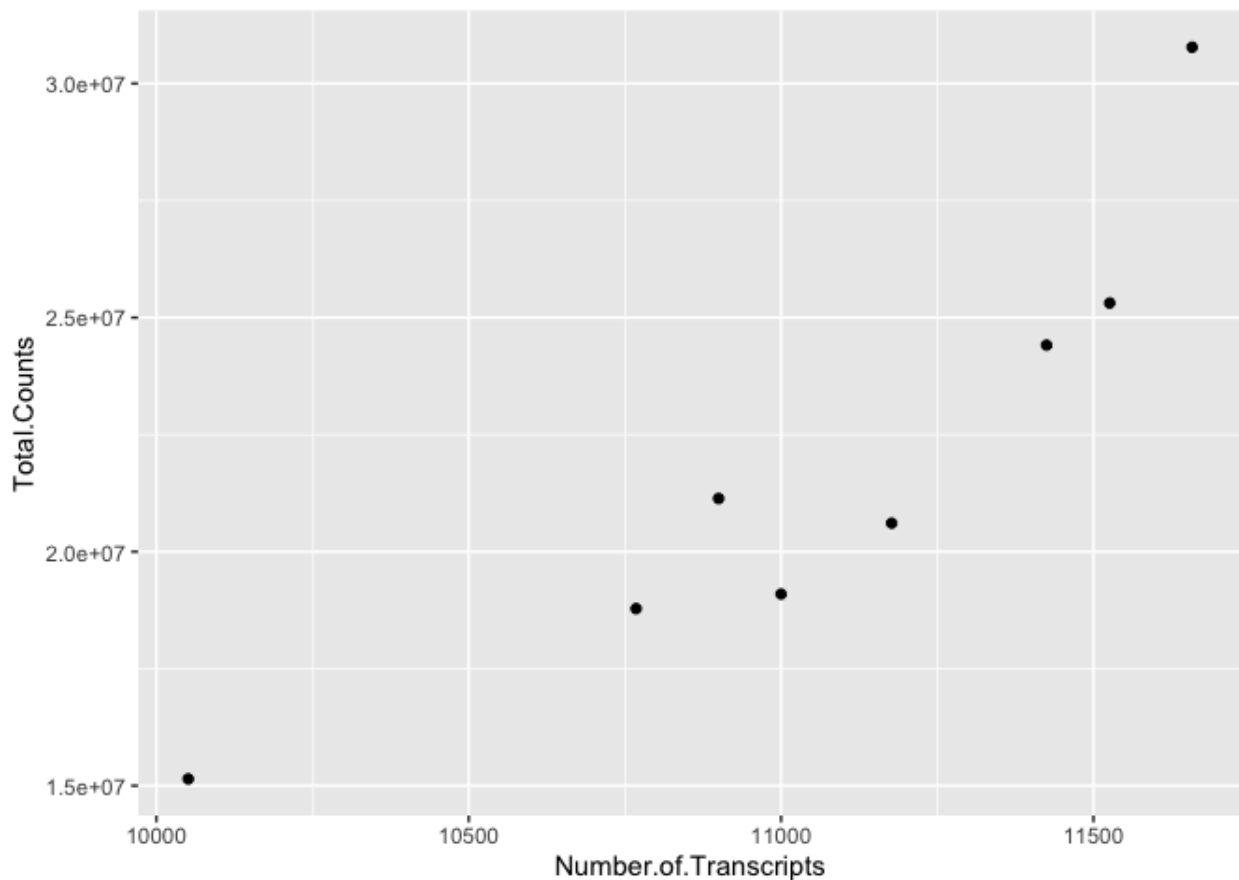
```
#What does running ggplot() do?  
ggplot(data=data)
```

```
#What about just running a geom function?  
geom_point(data=data,aes(x=Number.of.Transcripts, y = Total.Counts))
```

```
## mapping: x = ~Number.of.Transcripts, y = ~Total.Counts  
## geom_point: na.rm = FALSE  
## stat_identity: na.rm = FALSE  
## position_identity
```

```
#what about this  
ggplot() +  
geom_point(data=data,aes(x=Number.of.Transcripts, y = Total.Counts))
```



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>)

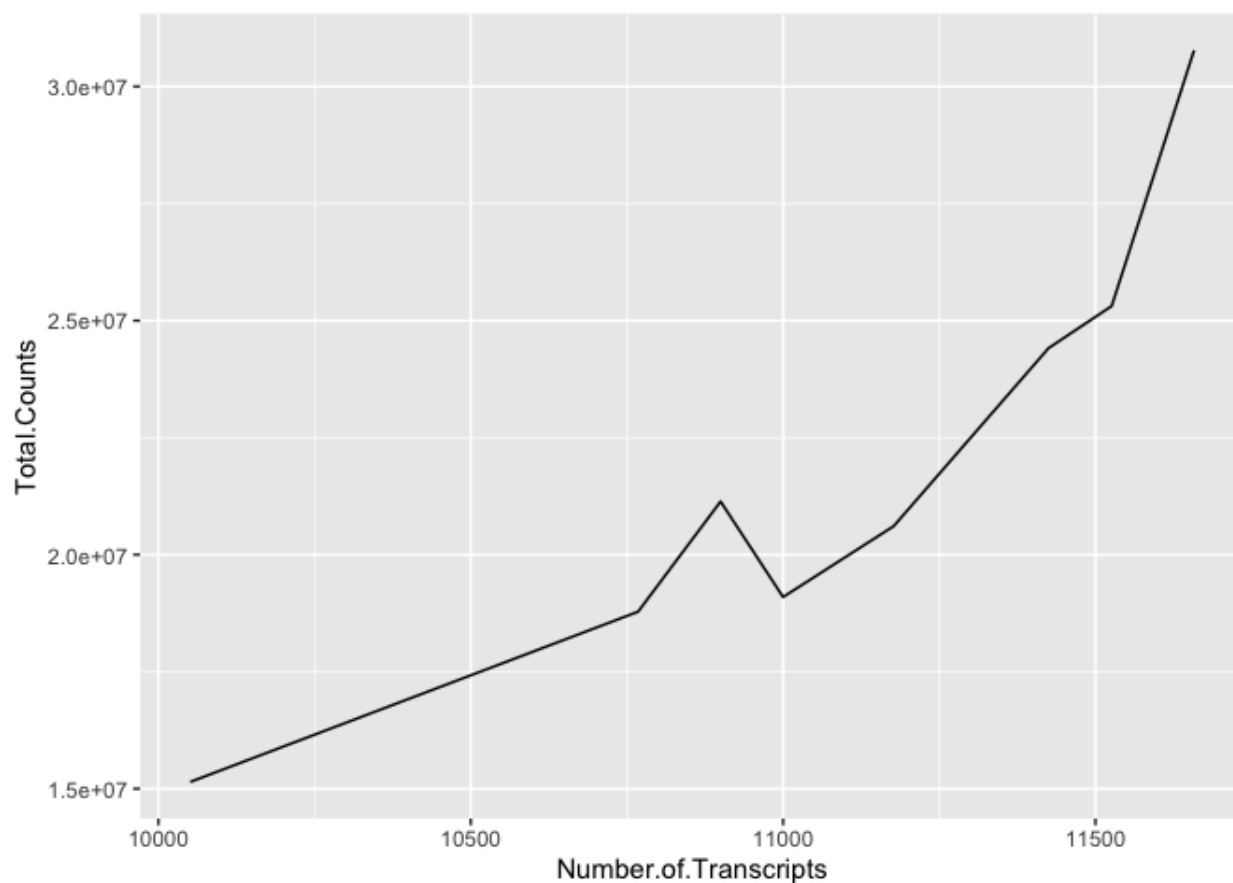
There are multiple geom functions that change the basic plot type or the plot representation. We can create scatter plots (`geom_point()`), line plots (`geom_line()`, `geom_path()`), bar plots (`geom_bar()`, `geom_col()`), line modeled to fitted data (`geom_smooth()`), heat maps (`geom_tile()`), geographic maps (`geom_polygon()`), etc.

ggplot2 provides over 40 geoms, and extension packages provide even more (see <https://exts.ggplot2.tidyverse.org/gallery/> (<https://exts.ggplot2.tidyverse.org/gallery/>) for a sampling). The best way to get a comprehensive overview is the ggplot2 cheatsheet, which you can find at <http://rstudio.com/resources/cheatsheets> (<http://rstudio.com/resources/cheatsheets>). --- R4DS (<https://r4ds.had.co.nz/data-visualisation.html>)

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

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

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



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>)

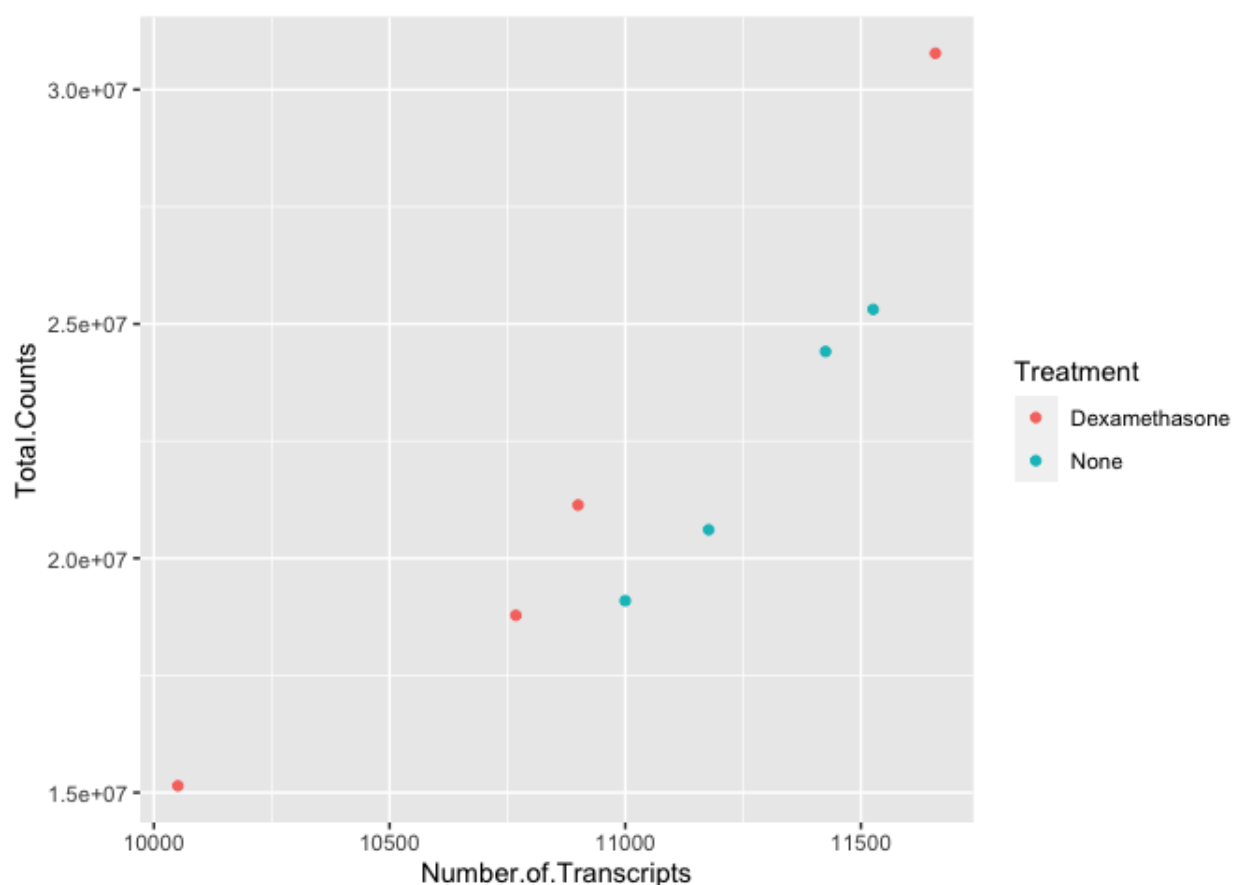
Mapping aesthetics include some of the following:

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.

Let's return to our plot above. Is there a relationship between treatment ("dex") and the number of transcripts or total counts?

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



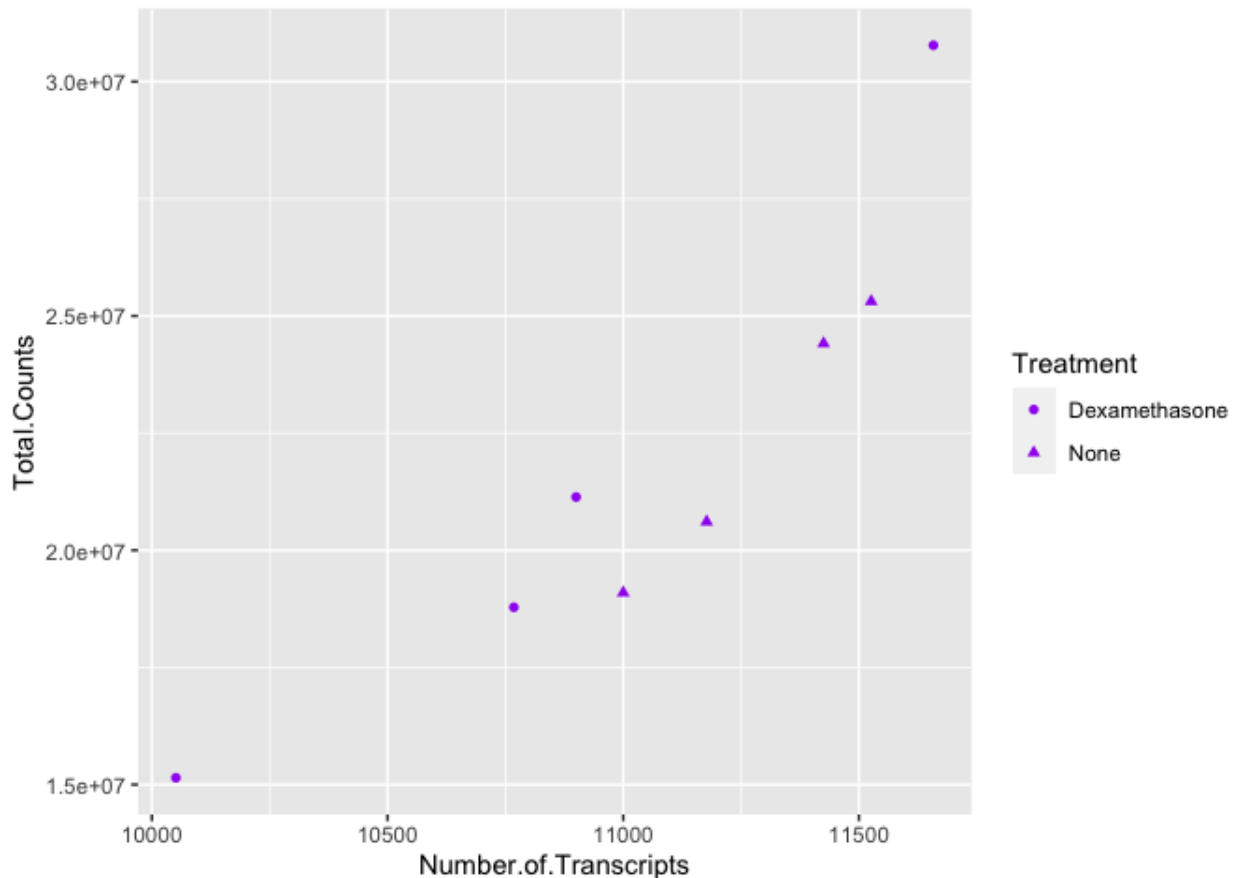
There is potentially a relationship. ASM cells treated with dexamethasone in general have lower total numbers of transcripts and lower total counts.

Notice how we changed the color of our points to represent a variable, in this case. To do this, we set color equal to 'Treatment' within the aes() function. This mapped our aesthetic, color, to a variable we were interested in exploring. 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

```
#map the shape aesthetic to the variable "dex"
```

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



We can also see from this that 'Treatment' could be mapped to other aesthetics. In the above example, we see it mapped to shape rather than color. 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) (<https://r-graphics.org/recipe-scatter-shapes>).

We could have just as easily mapped it 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.

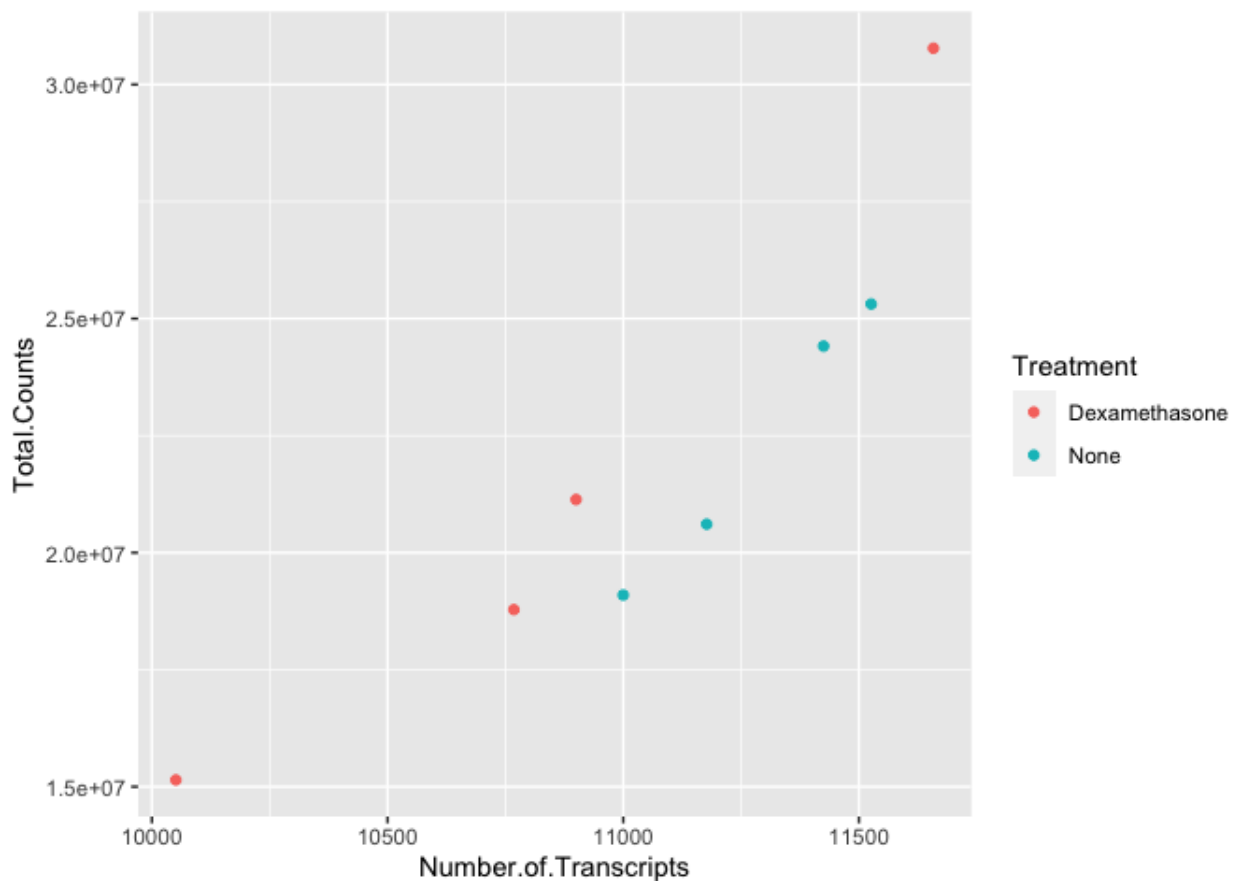
Other things to note

The assignment of color, shape, or alpha to our variable was automatic, with a unique aesthetic level representing each category (i.e., 'Dexamethasone', 'none') within our variable. You will also notice that ggplot2 automatically created a legend to explain the levels of the aesthetic mapped. We can change aesthetic parameters - what colors are used, for example - by adding additional layers to the plot.

R objects can also store figures

As we have discussed, R objects are used to store things created in R to memory. This includes plots.

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

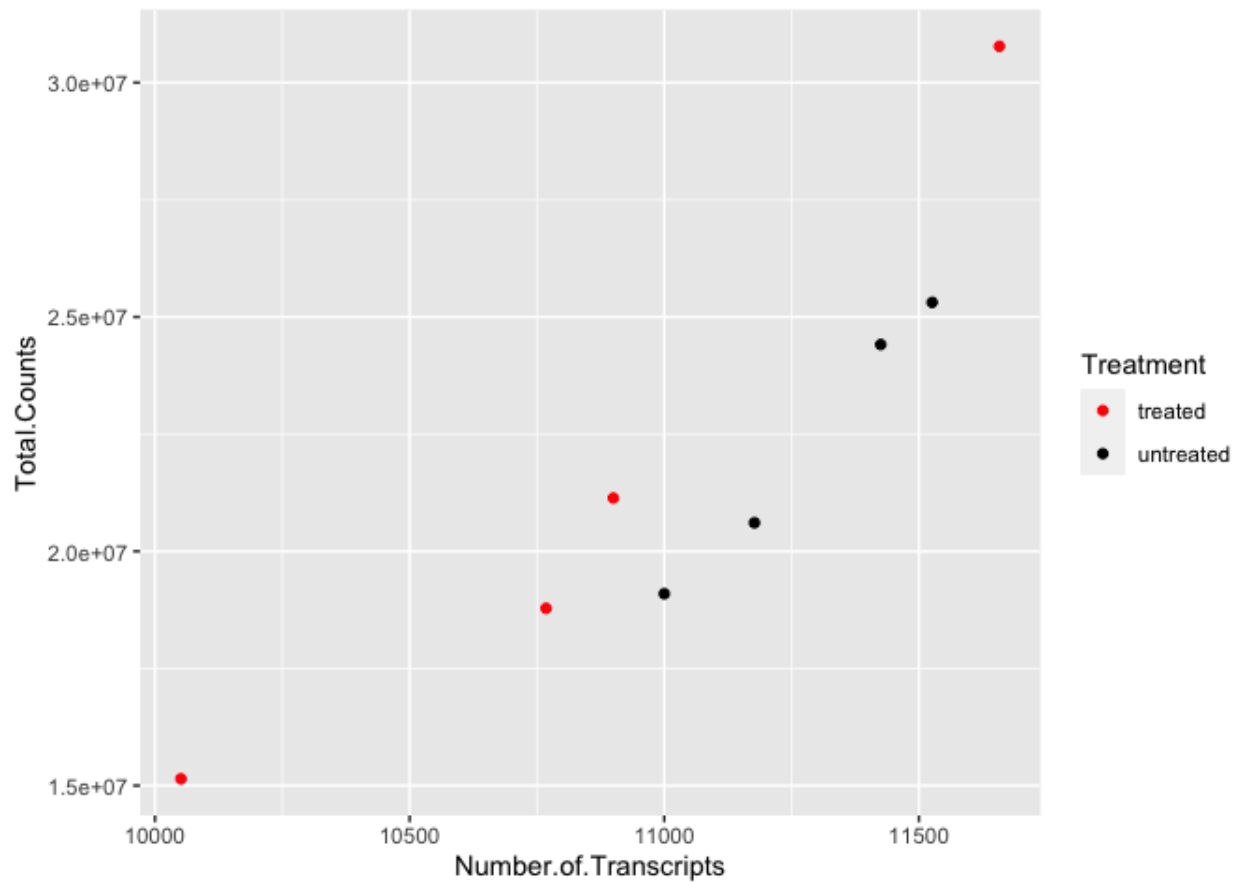


We can add additional layers directly to our object. We will see how this works by defining some colors for our 'dex' variable.

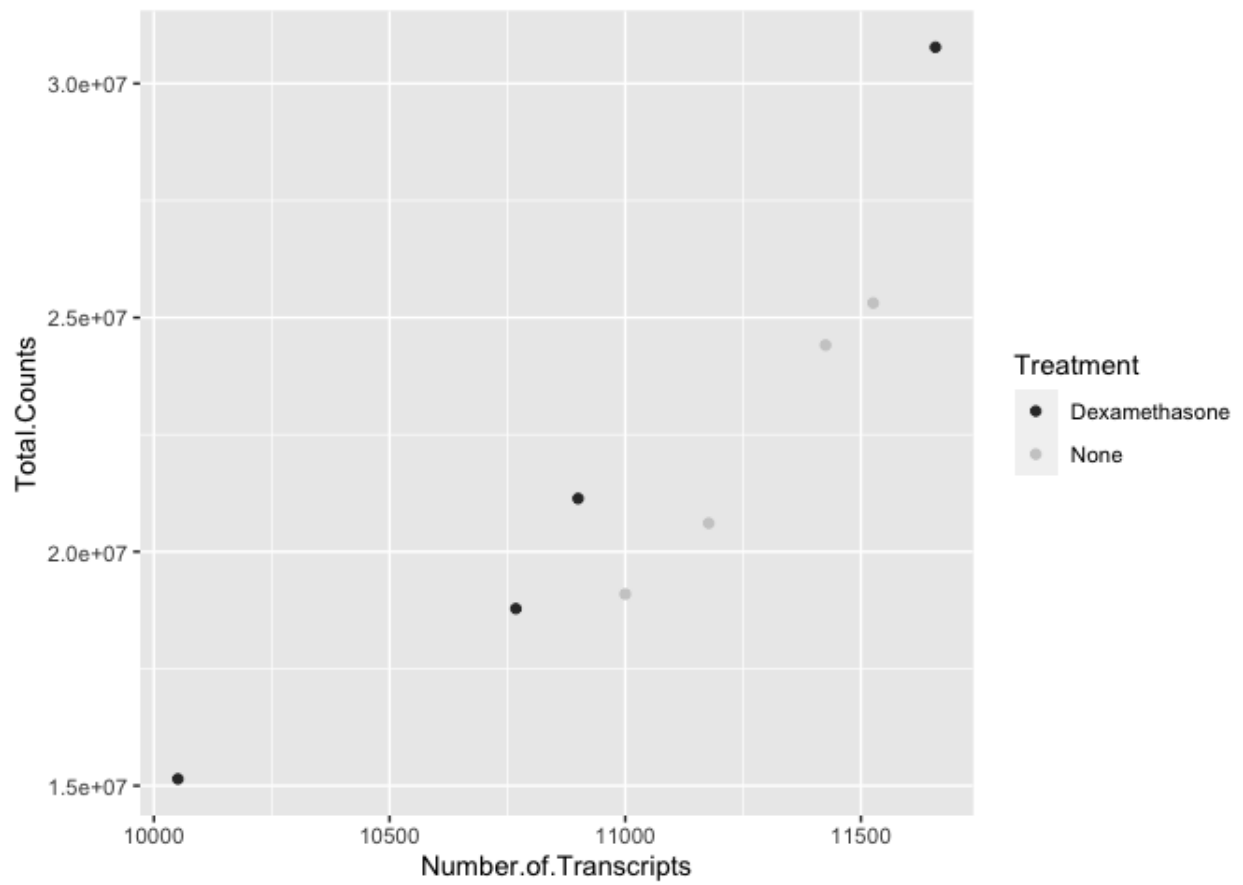
Colors

ggplot2 will automatically assign colors to the categories in our data. 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()`, `scale_color_grey()`, etc. We can also change the name of the color labels in the legend using the `labels` argument of these functions

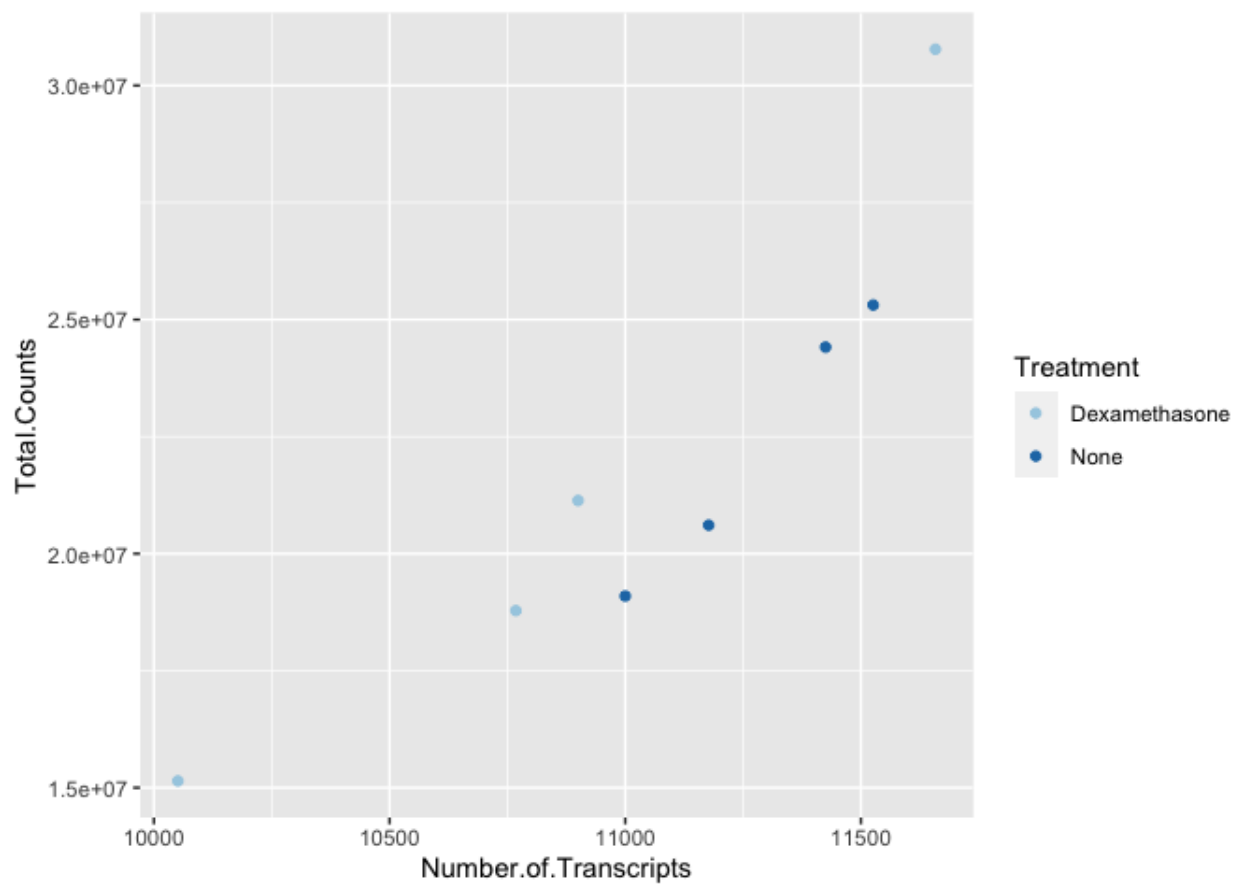
```
scatter_plot +  
  scale_color_manual(values=c("red","black"),  
                    labels=c('treated','untreated'))
```



```
scatter_plot +  
  scale_color_grey()
```




























```
scatter_plot +  
  scale_color_brewer(palette = "Paired")
```

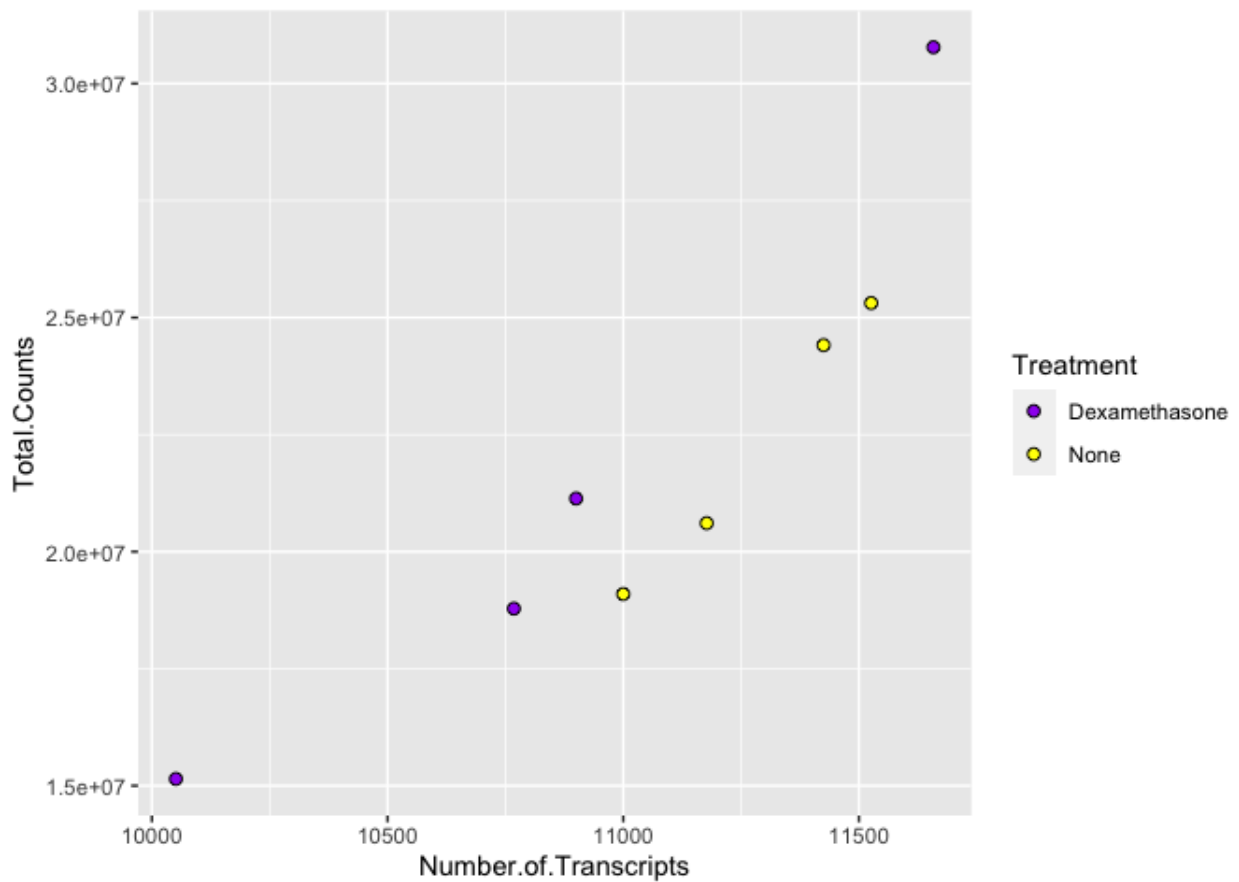



Similarly, if we want to change the fill, we would use the `scale_fill` options. To apply `scale_fill` to shape, we will have to alter the shapes, as only some shapes take a fill argument.

Image from <https://ggplot2.tidyverse.org/articles/ggplot2-specs.html> (<https://ggplot2.tidyverse.org/articles/ggplot2-specs.html>):

 0	 5	 10	 15	 22
 1	 6	 11	 16	 21
 2	 7	 12	 17	 24
 3	 8	 13	 18	 23
 4	 9	 14	 19	 20

```
ggplot(data=data) +
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts, fill=Treat
                shape=21,size=2) + #increase size and change points
  scale_fill_manual(values=c("purple", "yellow"))
```



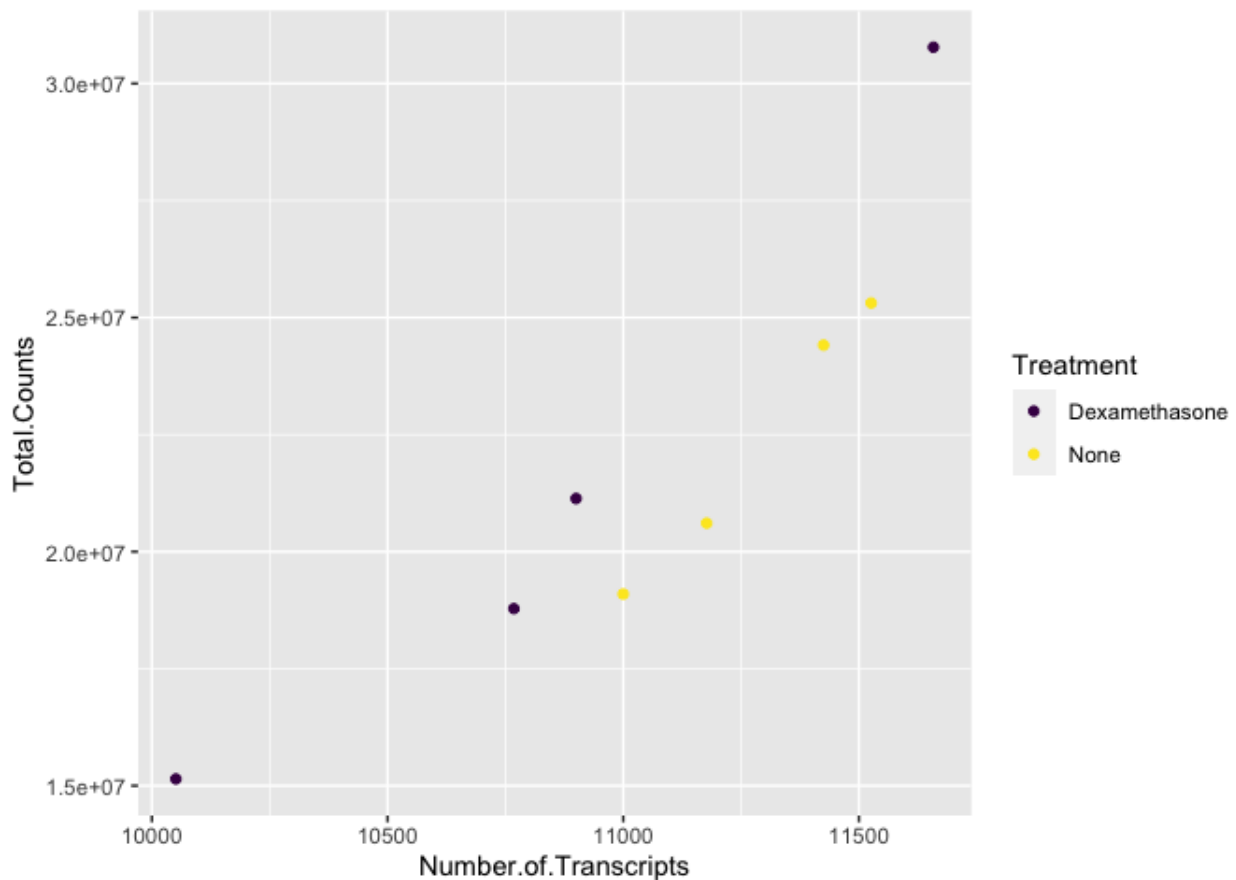
There are a number of ways to specify the color argument including by name, number, and hex code. [Here \(https://www.r-graph-gallery.com/ggplot2-color.html\)](https://www.r-graph-gallery.com/ggplot2-color.html) is a great resource from the [R Graph Gallery \(https://www.r-graph-gallery.com/index.html\)](https://www.r-graph-gallery.com/index.html)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

There are also a number of complementary packages in R that expand our color options. One of my favorites is `viridis`, which provides colorblind friendly palettes. `randomcoloR` is a great package if you need a large number of unique colors.

```
library(viridis) #Remember to load installed packages before use
```

```
## Loading required package: viridisLite
```

```
scatter_plot + scale_color_viridis(discrete=TRUE, option="viridis")
```

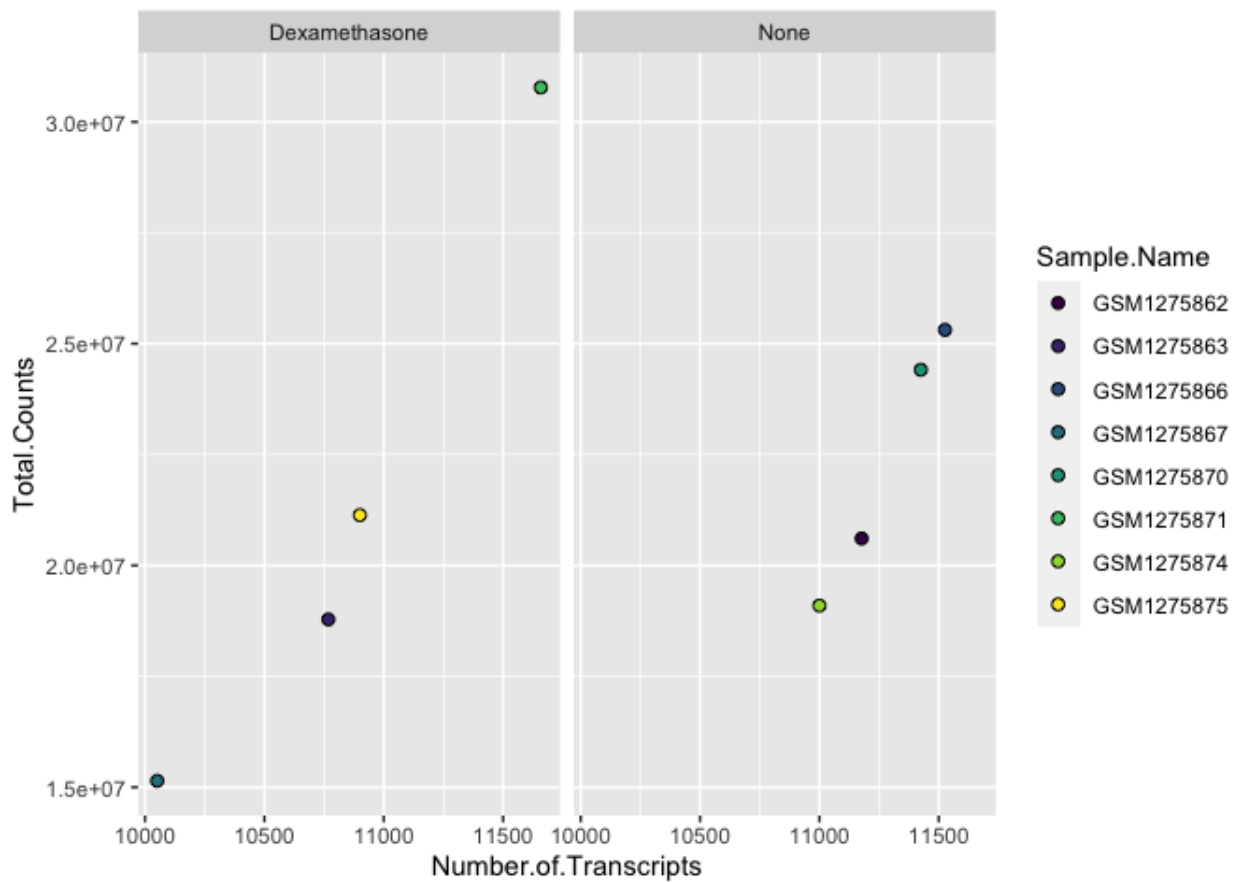


Paletteeer contains a comprehensive set of color palettes, if you want to load the palettes from multiple packages all at once. See the [Github page \(https://github.com/EmilHvitfeldt/paletteeer\)](https://github.com/EmilHvitfeldt/paletteeer) for details.

Facets

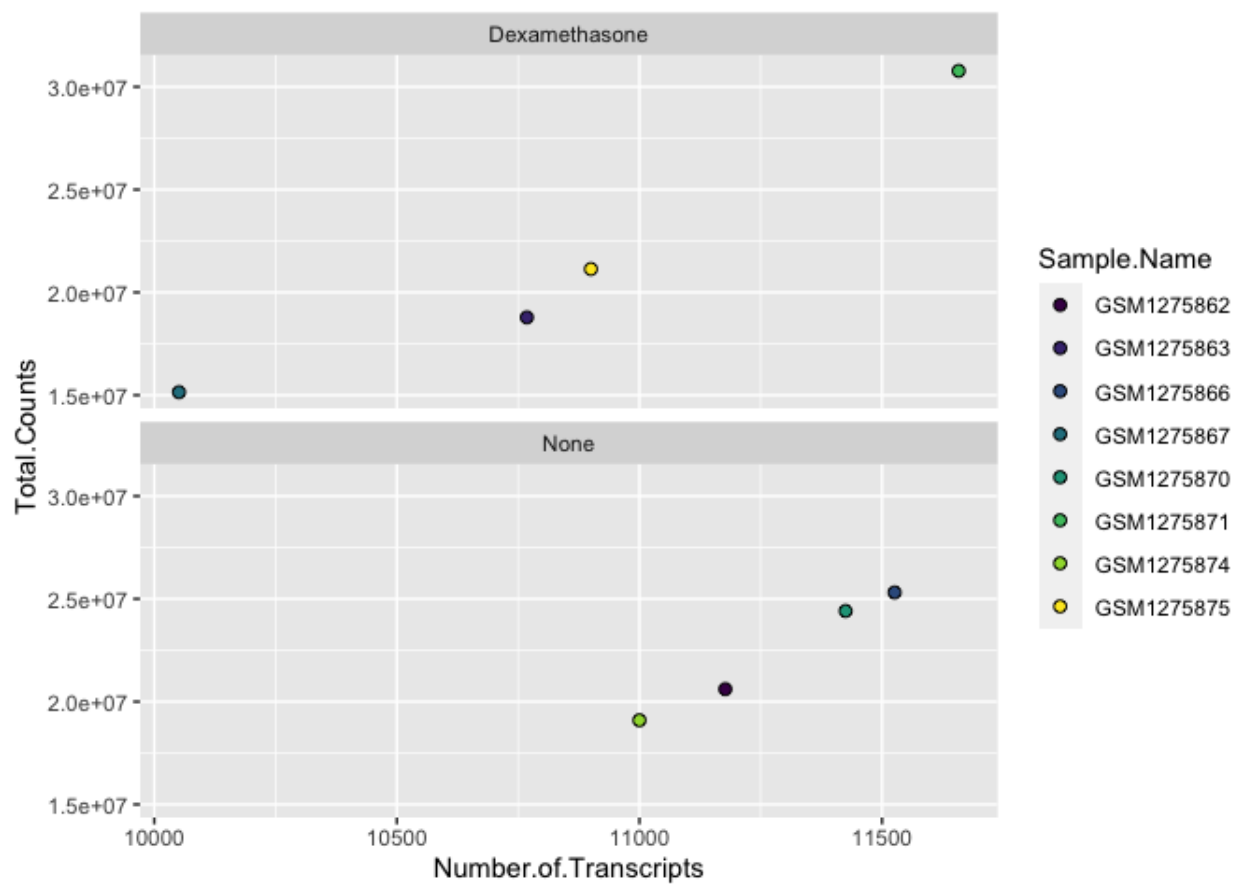
A way to add variables to a plot beyond mapping them to an aesthetic is to use facets or subplots. There are two primary functions to add facets, `facet_wrap()` and `facet_grid()`. If faceting by a single variable, use `facet_wrap()`. If multiple variables, use `facet_grid()`. The first argument of either function is a formula, with variables separated by a `~` (See below). Variables must be discrete (not continuous).

```
#plot
ggplot(data=data) +
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts,fill=SampleID,
                shape=21,size=2) + #increase size and change points
            scale_fill_viridis(discrete=TRUE, option="viridis") +
            facet_wrap(~Treatment)
```



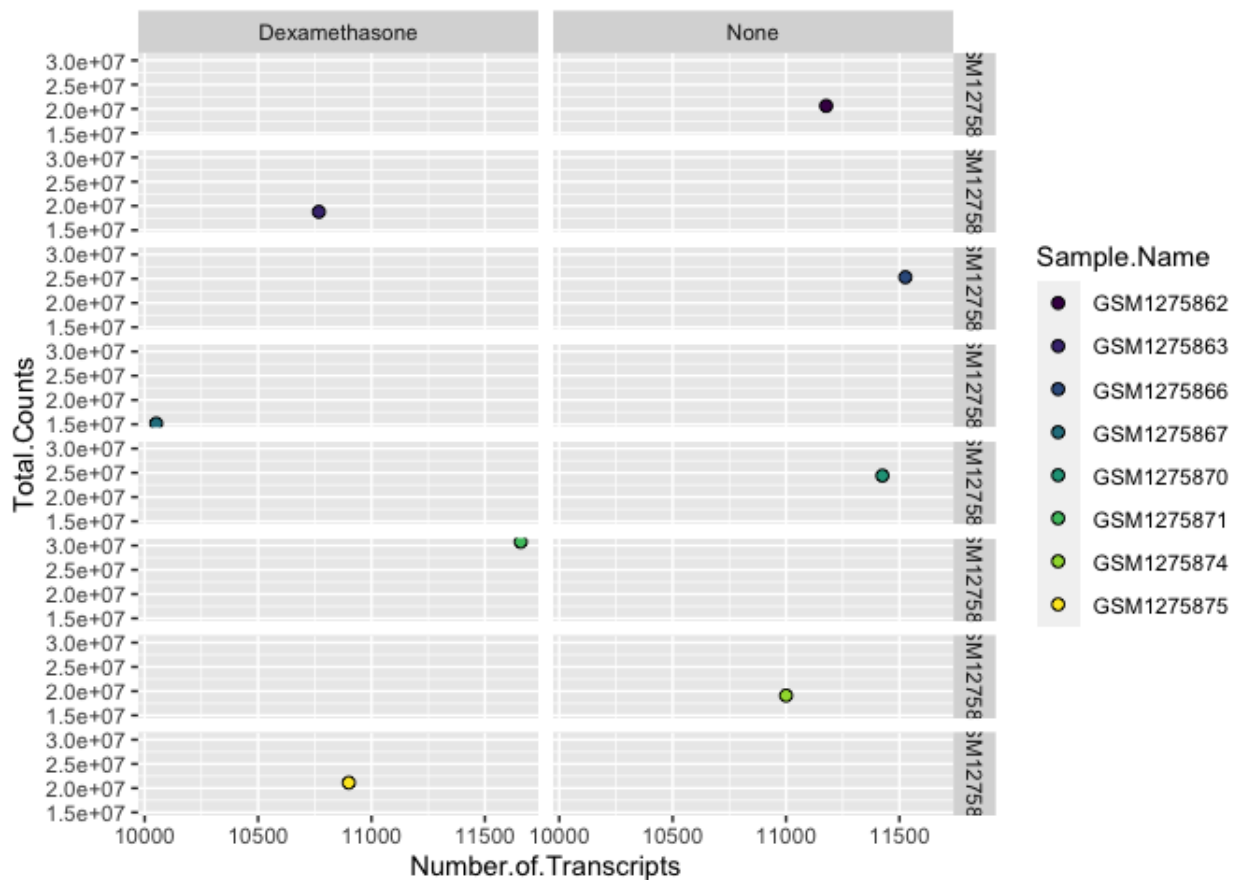
Note the help options with `?facet_wrap()`. How would we make our plot facets vertical rather than horizontal?

```
ggplot(data=data) +
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts, fill=Sample.Name,
                shape=21, size=2) + #increase size and change points
            scale_fill_viridis(discrete=TRUE, option="viridis") +
            facet_wrap(~Treatment, ncol=1)
```



Be sure to take a look at `facet_grid()`. `Facet_grid` would allow us to map even more variables in our data

```
ggplot(data=data) +
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts, fill=Sample.Name,
                shape=21, size=2)) + #increase size and change points
  scale_fill_viridis(discrete=TRUE, option="viridis") +
  facet_grid(Sample.Name~Treatment)
```



This is a silly example, but hopefully it gets across the point.

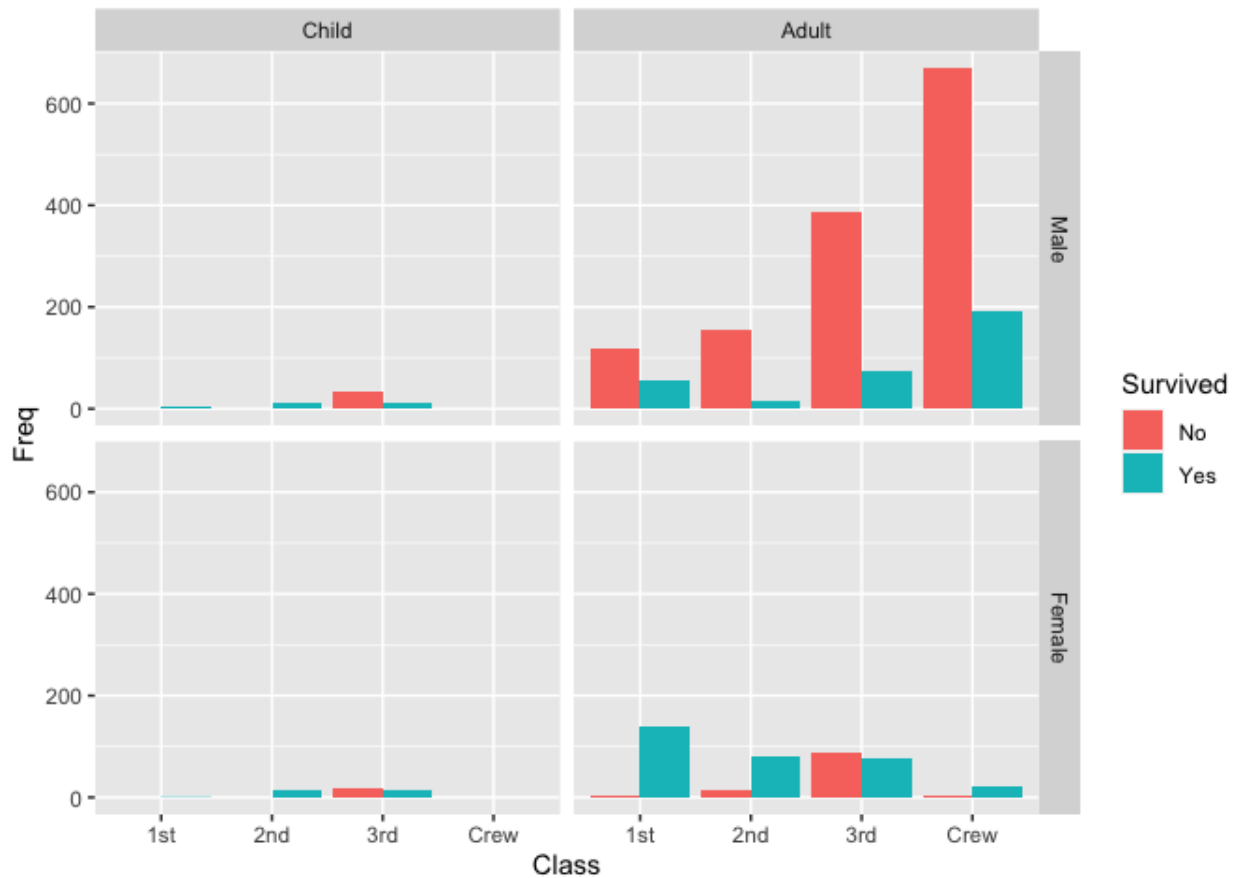
A better example of `facet_grid()` using `data("Titanic")`.

This data set provides information on the fate of passengers on the fatal maiden voyage of the ocean liner 'Titanic', summarized according to economic status (class), sex, age and survival. --- R Documentation ?Titanic

```
data("Titanic")
Titanic <- as.data.frame(Titanic)
head(Titanic)
```

```
##   Class  Sex  Age Survived Freq
## 1   1st  Male Child      No    0
## 2   2nd  Male Child      No    0
## 3   3rd  Male Child      No   35
## 4  Crew  Male Child      No    0
## 5   1st Female Child      No    0
## 6   2nd Female Child      No    0
```

```
ggplot() + geom_col(data = Titanic, aes(x = Class, y = Freq, fill = Survived)) +
  facet_grid(Sex ~ Age)
```



Building upon our template

This is the grammar of graphics. Adding layers to create unique figures.

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

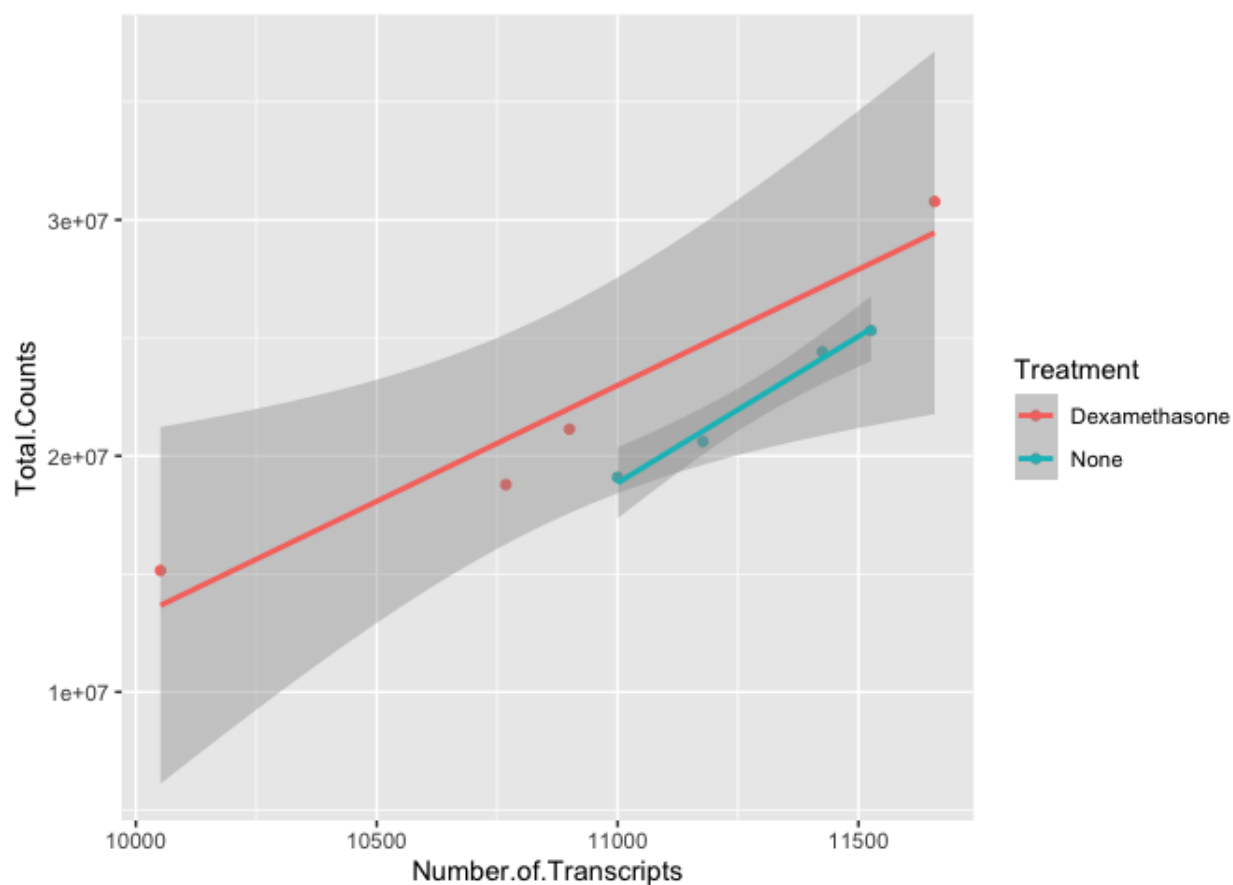
Note that there are a lot of invisible (default) layers that often go into each ggplot2, and there are ways to customize these layers. See [this chapter \(https://r4ds.had.co.nz/data-visualisation.html#the-layered-grammar-of-graphics\)](https://r4ds.had.co.nz/data-visualisation.html#the-layered-grammar-of-graphics) from R for Data Science for more information on the grammar of graphics.

Using multiple geoms per plot

Because we build plots using layers in ggplot2. We can add multiple geoms to a plot to represent the data in unique ways.

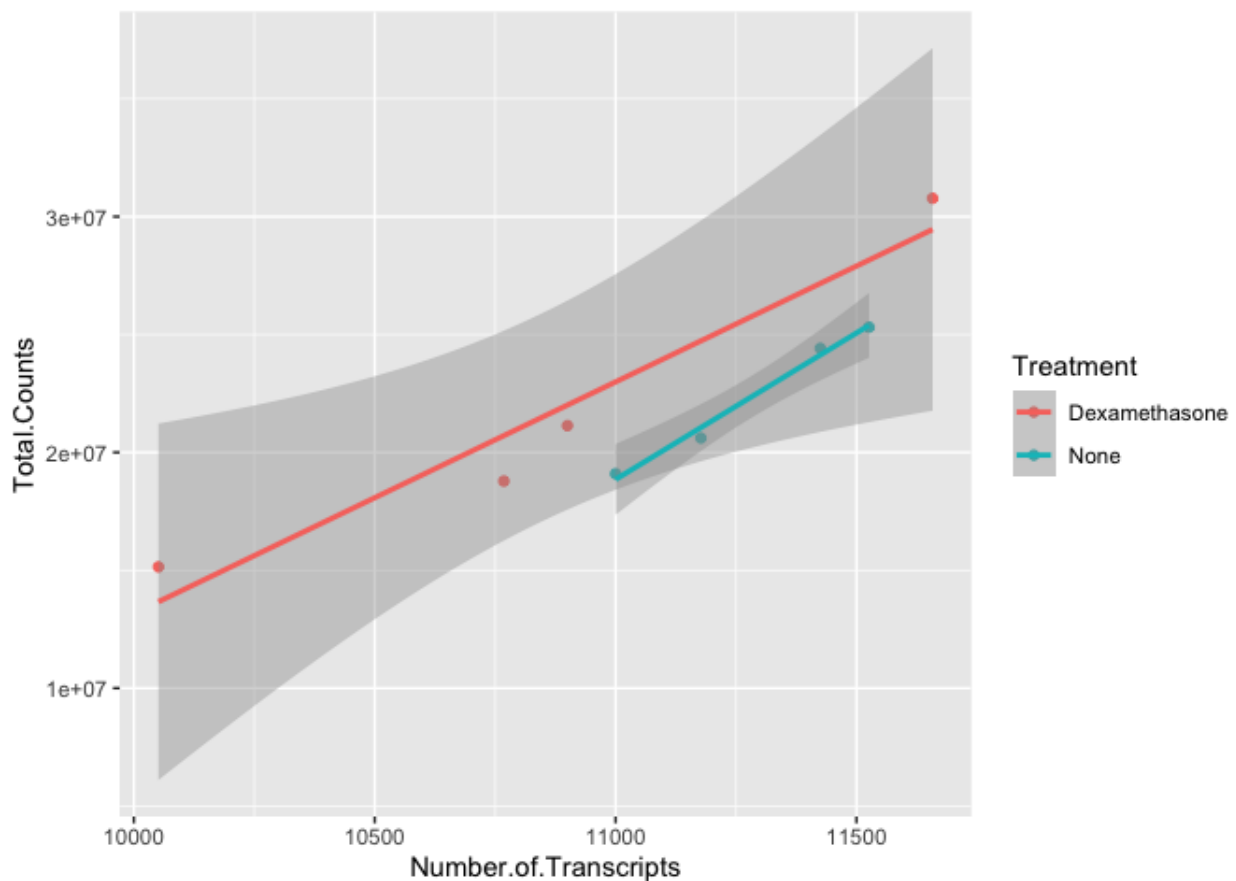
```
#We can combine geoms; here we combine a scatter plot with a  
#with a linear model regression line using geom_smooth  
ggplot(data=data) +  
  geom_point(aes(x=Number.of.Transcripts, y = Total.Counts,  
                color=Treatment)) +  
  geom_smooth(method='lm', aes(x=Number.of.Transcripts,  
                              y = Total.Counts, color= Treatment))
```

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



```
#to make our code more effective, we can put shared aesthetics in the
#ggplot function
ggplot(data=data, aes(x=Number.of.Transcripts,
                      y = Total.Counts, color= Treatment)) +
  geom_point() +
  geom_smooth(method='lm')
```

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



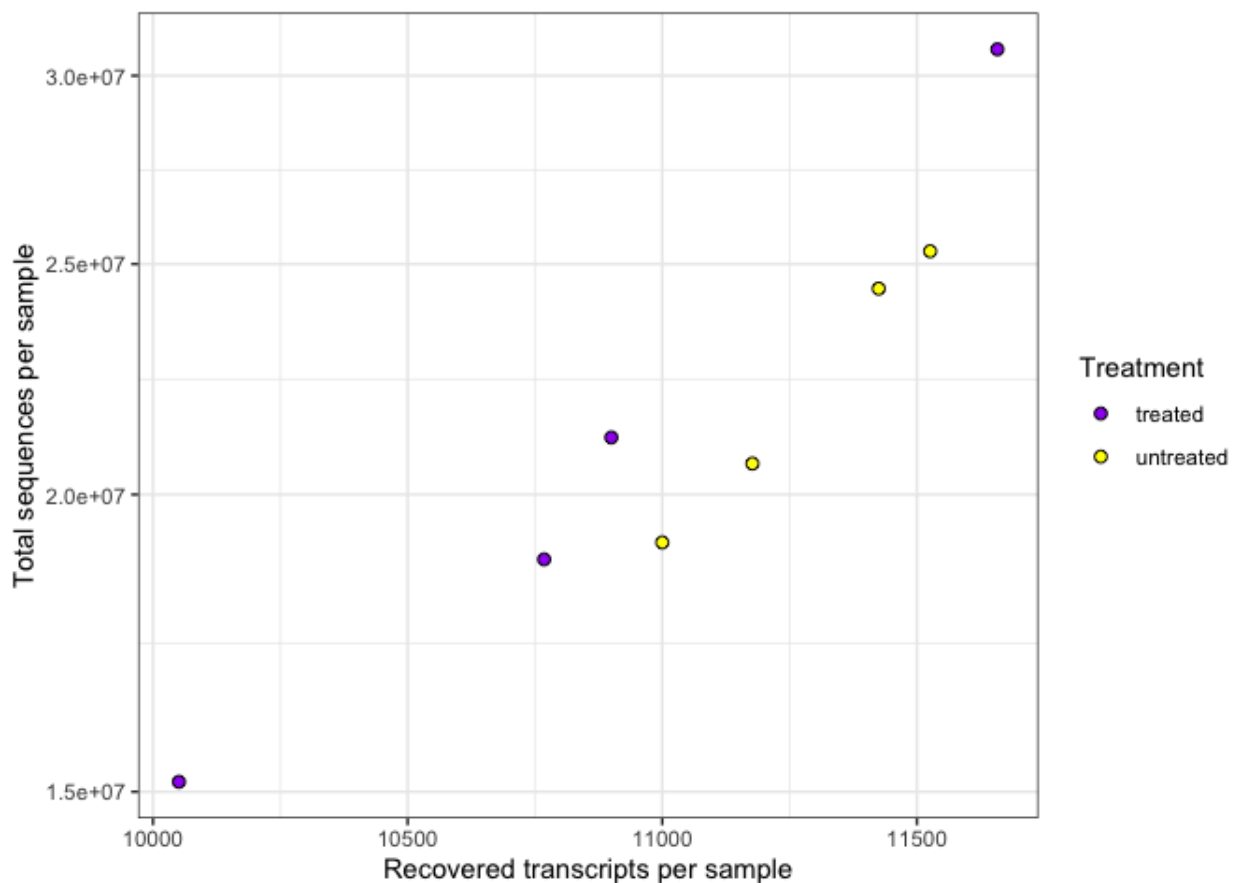
Labels, legends, scales, and themes

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, scale the axes, and customize the legends and [theme](https://ggplot2.tidyverse.org/reference/theme.html) (<https://ggplot2.tidyverse.org/reference/theme.html>). We will be working with these additional plot features in the weeks to come, so stay tuned.

Here's a teaser.

```
ggplot(data=data) +
```

```
geom_point(aes(x=Number.of.Transcripts, y = Total.Counts,
              fill=Treatment),
          shape=21,size=2) +
scale_fill_manual(values=c("purple", "yellow"),
                 labels=c('treated','untreated'))+
#can change labels of fill levels along with colors
xlab("Recovered transcripts per sample") + #add x label
ylab("Total sequences per sample") + #add y label
guides(fill = guide_legend(title="Treatment")) + #label the legend
scale_y_continuous(trans="log10") + #log transform the y axis
theme_bw()
```



Saving plots (ggsave())

Finally, we have a quality plot ready to publish. The next step is to save our plot to a file. The easiest way to do this 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)
```

Check out [this article \(https://www.tidyverse.org/blog/2020/08/taking-control-of-plot-scaling/\)](https://www.tidyverse.org/blog/2020/08/taking-control-of-plot-scaling/) for recommendations on effectively scaling plots.

Resource list

1. [ggplot2 cheatsheet](#)
2. [The R Graph Gallery \(https://www.r-graph-gallery.com/\)](https://www.r-graph-gallery.com/)
3. [The R Graphics Cookbook \(https://r-graphics.org/recipe-quick-bar\)](https://r-graphics.org/recipe-quick-bar)
4. [Ggplot2 extensions \(https://exts.ggplot2.tidyverse.org/gallery/\)](https://exts.ggplot2.tidyverse.org/gallery/)

Acknowledgements

Material from this lesson was adapted from Chapter 3 of [R for Data Science \(https://r4ds.had.co.nz/data-visualisation.html\)](https://r4ds.had.co.nz/data-visualisation.html) and from a 2021 workshop entitled [Introduction to Tidy Transcriptomics \(https://stemangiola.github.io/bioc2021_tidytranscriptomics/articles/tidytranscriptomics.html\)](https://stemangiola.github.io/bioc2021_tidytranscriptomics/articles/tidytranscriptomics.html) by Maria Doyle and Stefano Mangiola.

Scatter plots and plot customization

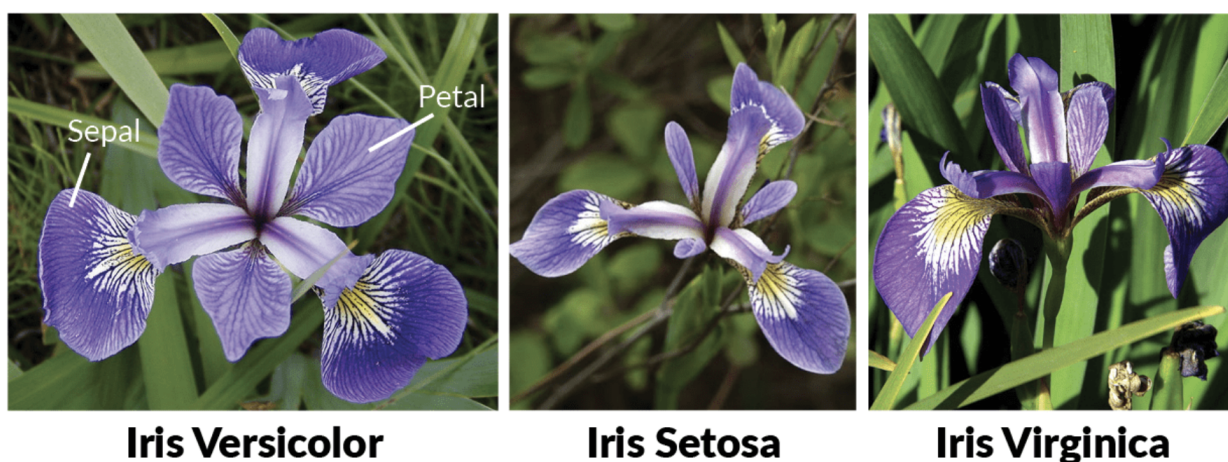
Objectives

1. Learn to customize your ggplot with labels, axes, text annotations, and themes.
2. Learn how to make and modify scatter plots to make fairly different overall plot representations.
3. Load a different data set using new load functions

The primary purpose of this lesson is to learn how to customize our ggplot2 plots. We will do this by focusing on different types of scatter plots.

The Data

In this lesson we will use two different sets of data. First, we will use data available with your base R installation, the iris data set, which is stored in object `iris`. These data include measurements from the petals and sepals of different Iris species including *Iris setosa*, *versicolor*, and *virginica*. See `?iris` for more information about these data.



The Irises: image from <https://www.datacamp.com/community/tutorials/machine-learning-in-r>

Second, we will use some more complicated bioinformatics data related to the RNAseq project introduced in Lesson 2.

First, let's load our libraries using the library function, `library()`:

```
library(ggrepel) #Needed for label repel
```

```
## Loading required package: ggplot2
```

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

```
##  
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':  
##  
##   filter, lag
```

```
## The following objects are masked from 'package:base':  
##  
##   intersect, setdiff, setequal, union
```

Now, let's load the RNA data that we will use toward the end of this lesson. We will use the function `read.delim()` to load tab delimited RNASeq data and the function `readLines()` to load the list of top genes.

```
dexp_sigtrnsc<-read.delim("../data/sig_dexp_results.txt",  
                          as.is=TRUE)  
topgenes<-readLines("../data/topgenes.txt")
```

The grammar of graphics

We are returning to the core grammar of graphics concept introduced in Lesson 2. Remember, to create a plot all you need are data, `geom_function(s)`, and mapping arguments.

Here is the basic template we left off with:

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

However, there are additional components, highlighted in bold, that can be added to our core components to enable us to generate even more diverse plot types.

Our grammar of graphics:

- one or more datasets,
- one or more geometric objects that serve as the visual representations of the data, – for instance, points, lines, rectangles, contours,
- descriptions of how the variables in the data are mapped to visual properties (aesthetics) of the geometric objects, and an associated scale (e. g., linear, logarithmic, rank),
- a facet specification, i.e. the use of multiple similar subplots to look at subsets of the same data,
- **one or more coordinate systems,**
- **optional parameters that affect the layout and rendering, such text size, font and alignment, legend positions.**
- **statistical summarization rules, [See LESSON 4]**

---(Holmes and Huber, 2021 (<https://web.stanford.edu/class/bios221/book/03-chap.html>))

We will extend our basic template throughout this lesson as we make a variety of scatter plots and in lesson 4.

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>)

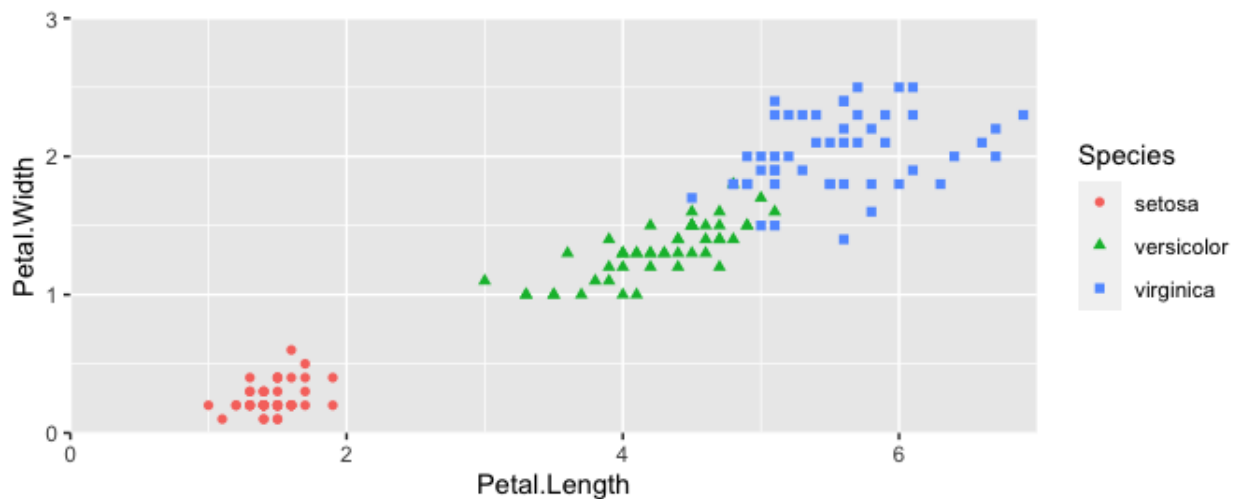
Because scatter plots involve mapping each data point, the `geom_point()` function is used. We saw a fairly basic implementation of this in Lesson 2.

Basic Scatter

Let's take another look at a simple scatter plot using the iris data. We can look at the relationship between petal length and petal width (i.e., variable association) for the various Iris species.

```
ggplot(data = iris) + #include our data
```

```
geom_point(aes(x = Petal.Length, y = Petal.Width,
               color = Species, shape = Species)) +
#geom and mappings
coord_fixed(xlim = c(0,7),
           ylim=c(0,3),expand=FALSE) # NEW CORE COMPONENT
```



This code should look fairly familiar, with the exception of a new function, `coord_fixed()`. This is a modification of the ggplot2 coordinate system.

Coordinate Systems

Coordinate systems are probably the most complicated part of ggplot2. The default coordinate system is the Cartesian coordinate system where the x and y positions act independently to determine the location of each point. --- [R4DS \(https://r4ds.had.co.nz/data-visualisation.html#coordinate-systems\)](https://r4ds.had.co.nz/data-visualisation.html#coordinate-systems)

`coord_fixed()` with the default argument `ratio=1` ensures that the units are represented equally in physical space on the plot. Because the x and y measurements were both taken in centimeters, it is good practice to make sure that the "same mapping of data space to physical space is used." --- [Holmes and Huber, 2021 \(https://web.stanford.edu/class/bios221/book/03-chap.html\)](https://web.stanford.edu/class/bios221/book/03-chap.html)

You will not need to worry about the coordinate system of your plot in most cases, but it is likely you will need to mess with the coordinate system at some point in the future. Another commonly used coordinate function is `coord_flip()`, which allows you to flip the representation of the plot, for example, by switching bars in a bar plot from vertical to horizontal. See `?coord_flip()` for more information.

Our new template:

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

Perform and plot PCA data using `iris`.

Many complex plots (e.g., PCA ordinations) are in their basic form a scatter plot. Here we are going to apply PCA to the `iris` data and generate a plot using `ggplot2`.

What is PCA?

Principal component analysis (PCA) is a linear dimension reduction method applied to highly dimensional data. 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 \(https://towardsdatascience.com/principal-component-analysis-pca-79d228eb9d24\)](https://towardsdatascience.com/principal-component-analysis-pca-79d228eb9d24) and [here \(https://www.huber.embl.de/msmb/Chap-Multivariate.html\)](https://www.huber.embl.de/msmb/Chap-Multivariate.html).

Key points:

- New variables are defined by a linear combination of original variables
- Each subsequent new variable contains less information
- Applications: dimensionality reduction, clustering, outlier identification --- [Shawhin Talebi, 2021 \(https://towardsdatascience.com/principal-component-analysis-pca-79d228eb9d24\)](https://towardsdatascience.com/principal-component-analysis-pca-79d228eb9d24)

Note

PCAs are used frequently in -omics fields. However, often than not, there will be package specific functions for PCA and plotting PCA for different -omics analyses. Because of this, we will show the main features here using a simpler data set, `iris`.

Perform PCA

We can use the function `prcomp()` to run PCA on the first four columns of the iris data. The function takes numeric data.

```
colnames(iris)[1:4]
```

```
## [1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
```

```
pca <- prcomp(iris[,1:4], scale = TRUE)  
pca
```

```
## Standard deviations (1, ..., p=4):  
## [1] 1.7083611 0.9560494 0.3830886 0.1439265  
##  
## Rotation (n x k) = (4 x 4):  
##  
##           PC1      PC2      PC3      PC4  
## Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863  
## Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096  
## Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492  
## Petal.Width  0.5648565 -0.06694199 -0.6342727  0.5235971
```

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

```
## List of 5
## $ sdev      : num [1:4] 1.708 0.956 0.383 0.144
## $ rotation: num [1:4, 1:4] 0.521 -0.269 0.58 0.565 -0.377 ...
## ..- attr(*, "dimnames")=List of 2
## .. ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
## .. ..$ : chr [1:4] "PC1" "PC2" "PC3" "PC4"
## $ center   : Named num [1:4] 5.84 3.06 3.76 1.2
## ..- attr(*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
## $ scale    : Named num [1:4] 0.828 0.436 1.765 0.762
## ..- attr(*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
## $ x        : num [1:150, 1:4] -2.26 -2.07 -2.36 -2.29 -2.38 ...
## ..- attr(*, "dimnames")=List of 2
## .. ..$ : NULL
## .. ..$ : chr [1:4] "PC1" "PC2" "PC3" "PC4"
## - attr(*, "class")= chr "prcomp"
```

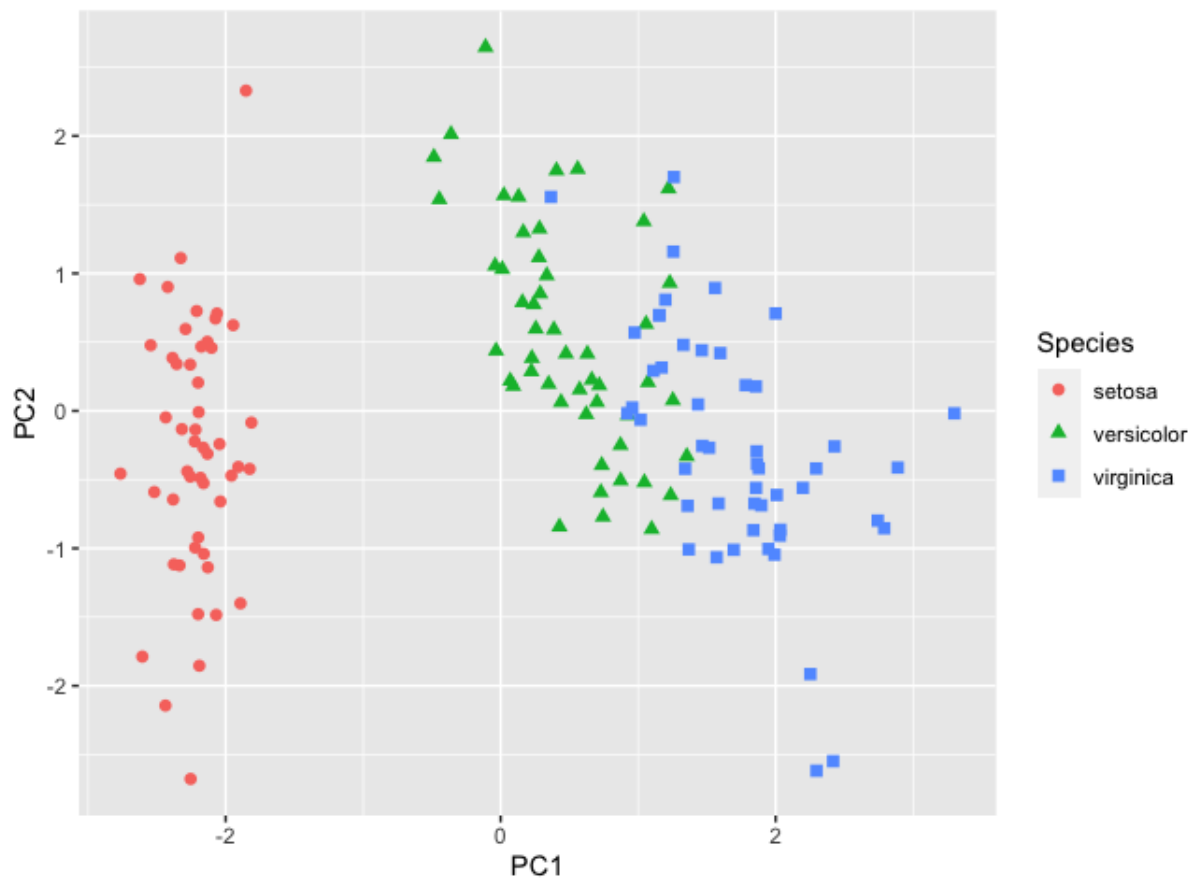
The object `pca` is a list of 5: the standard deviations of the principal components, a matrix of variable loadings, the scaling used, and the data projected on the principal components.

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$x[, 1:2]) # extract first two PCs
pcaData <- cbind(pcaData, iris$Species) # add species to df
colnames(pcaData) <- c("PC1", "PC2", "Species") # change column names

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



Info

Tip 6, [here \(https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6586259/\)](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 the species relate based on characteristics of their sepals and petals. From this plot, we see that *Iris virginica* and *Iris versicolor* are more similar than *Iris setosa*.

But, 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)
```

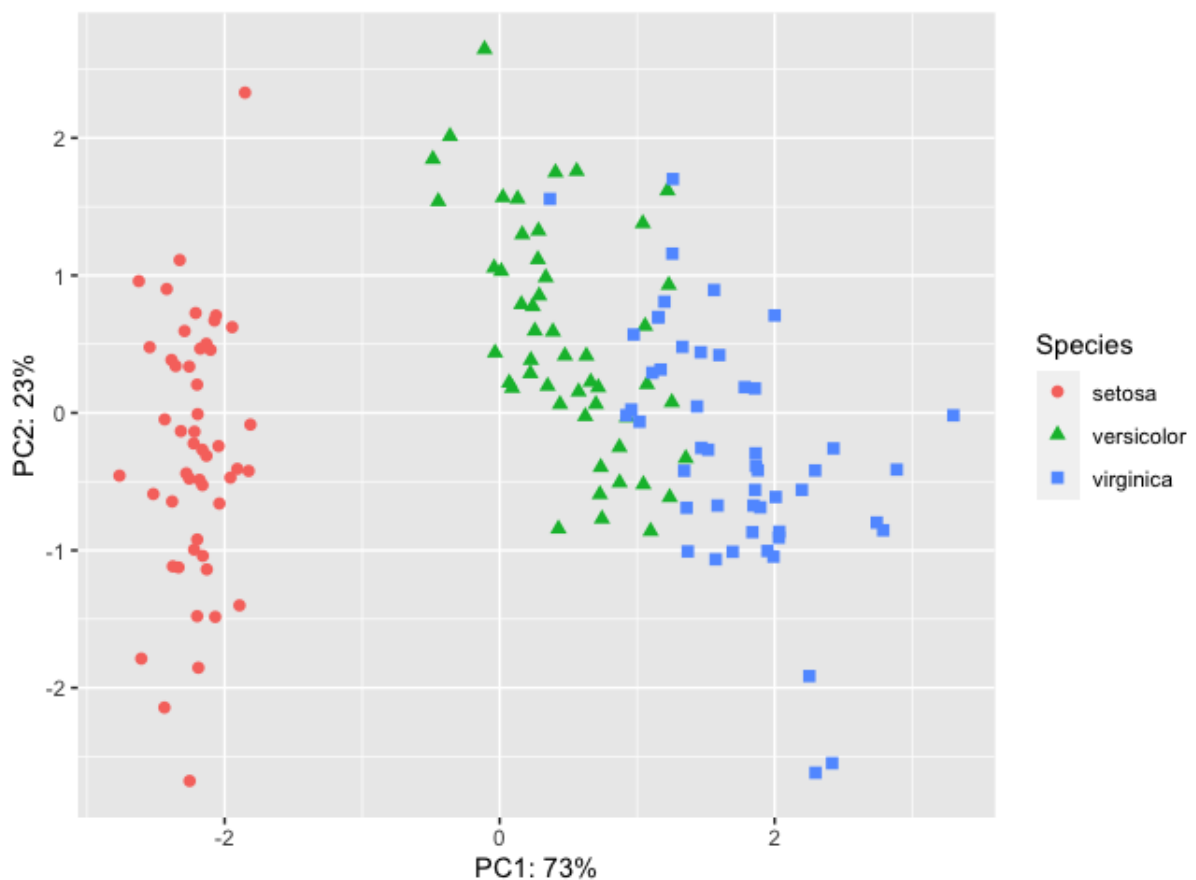
```
## Importance of components:
##              PC1      PC2      PC3      PC4
## Standard deviation  1.7084  0.9560  0.38309  0.14393
```

```
## Proportion of Variance 0.7296 0.2285 0.03669 0.00518
## Cumulative Proportion 0.7296 0.9581 0.99482 1.00000
```

PC1 and PC2 combined account for 96% 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 = Species, shape = Species) +
  geom_point(size = 2) +
  coord_fixed() +
  xlab("PC1: 73%")+ #x axis label text
  ylab("PC2: 23%") # y axis label text
```



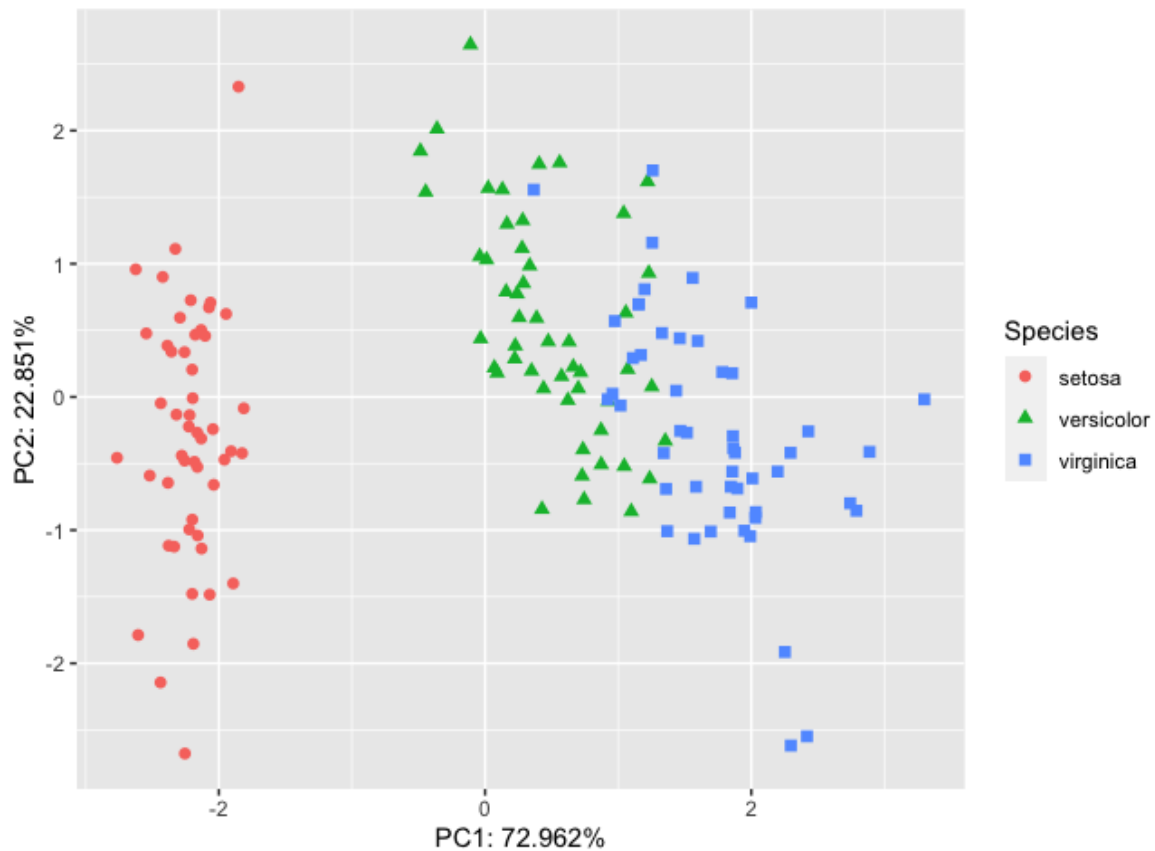
Automating % Variance in axis labels



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 = Species, shape = Species) +
```

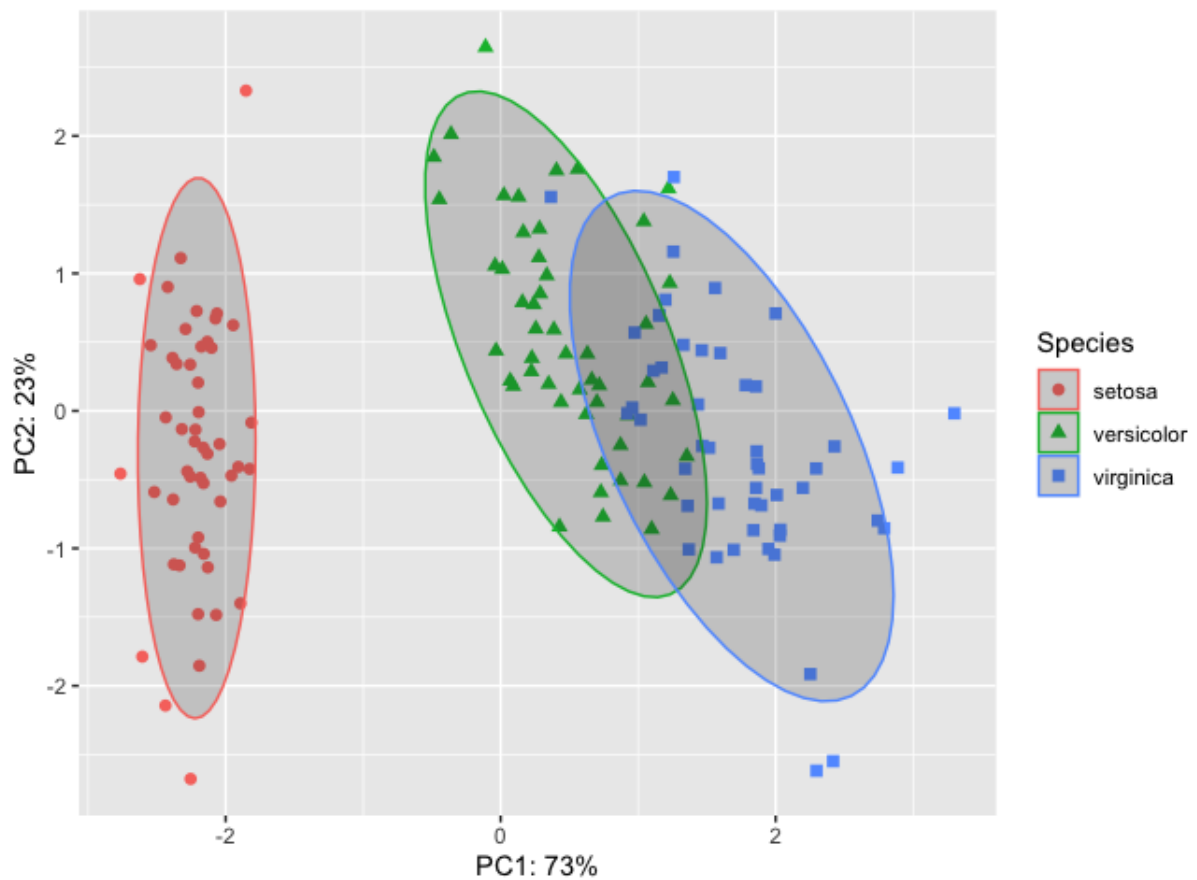
```
geom_point(size = 2) +
coord_fixed() +
labs(x=paste0("PC1: ",summary(pca)$importance[2,1]*100,"%"),
      y=paste0("PC2: ",summary(pca)$importance[2,2]*100,"%"))
```



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. We will discuss statistical transformation more in lesson 4 and how they apply. However, 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.

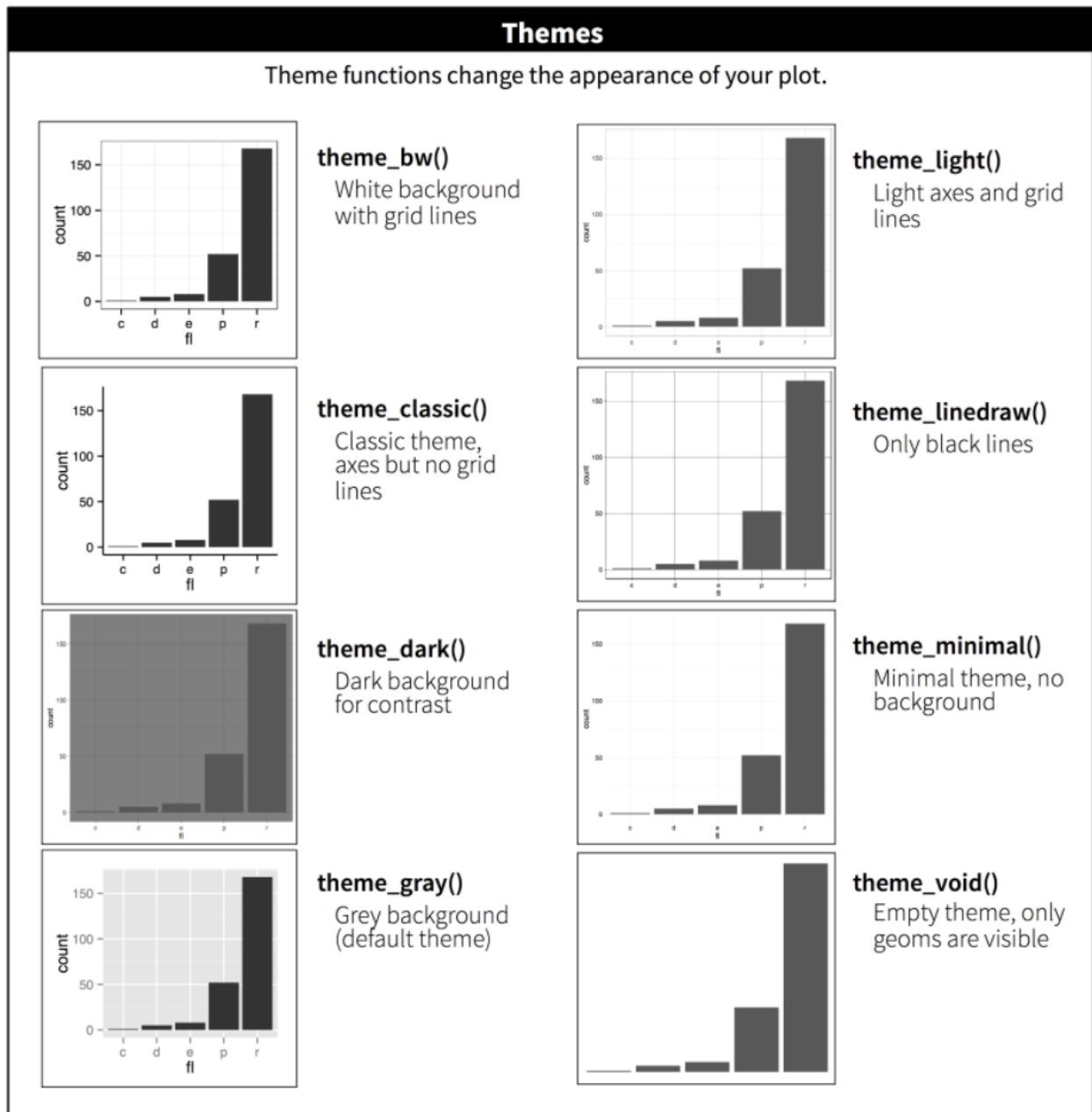
```
ggplot(pcaData) +
  aes(PC1, PC2, color = Species, shape = Species) +
  geom_point(size = 2) +
  coord_fixed() +
  xlab("PC1: 73%")+
  ylab("PC2: 23%") +
  stat_ellipse(geom="polygon", level=0.95, alpha=0.2) #adding a stat
```



Plot customization

Themes

The plot above is looking pretty good, but 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.



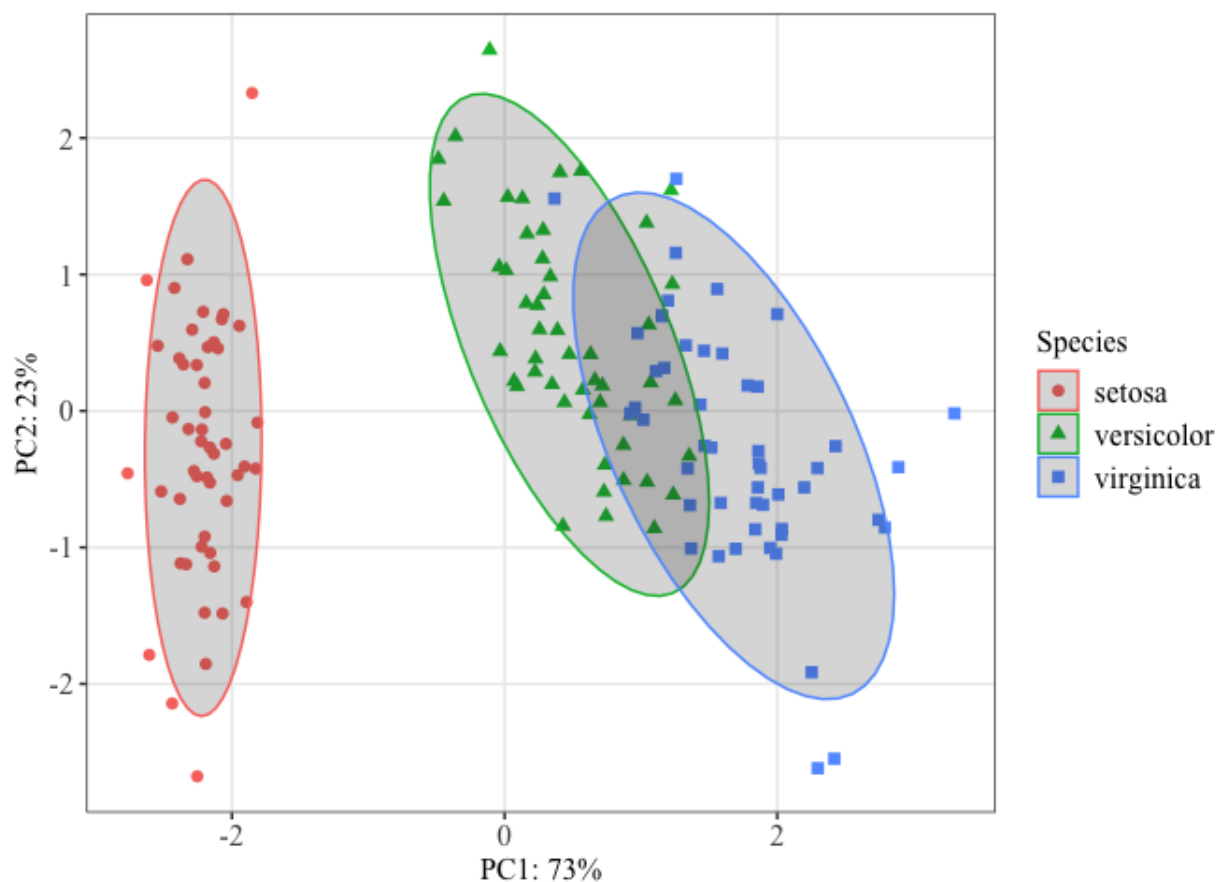
We can also specify and build a theme within our plot code or develop a custom theme to be reused across multiple plots. The theme function is the bread and butter of plot customization. Check out `?ggplot2::theme()` for a list of available parameters. There are many.

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

```
ggplot(pcaData) +
  aes(PC1, PC2, color = Species, shape = Species) + # define plot area
  geom_point(size = 2) + # adding data points
  coord_fixed() +
  xlab("PC1: 73%")+
  ylab("PC2: 23%") +
  stat_ellipse(geom="polygon", level=0.95, alpha=0.2) +
  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())
```



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

A helpful color trick

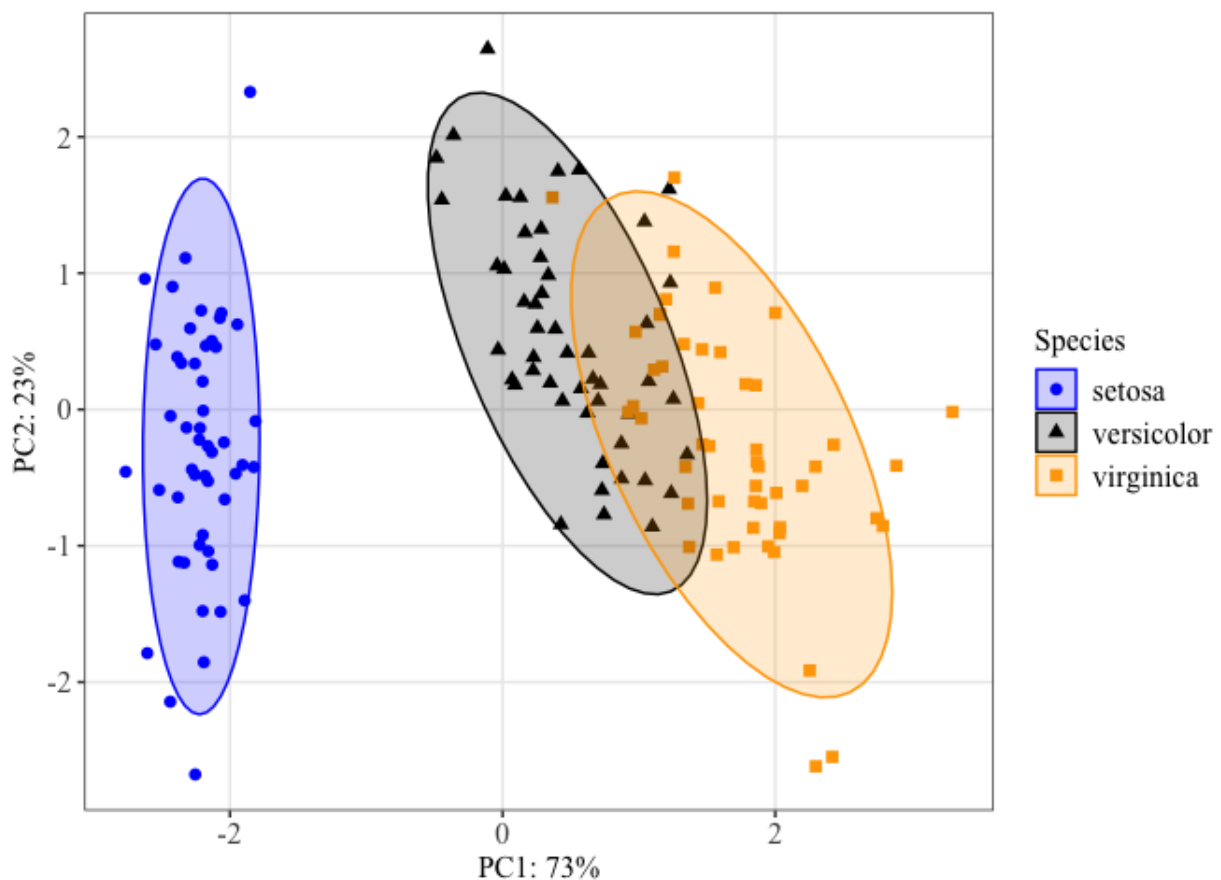
When you have a lot of colors and you want to keep these colors consistent, you can use the following convenient functions to set a name attribute for a vector of colors.

Let's do this for our iris species.

```
#defining colors
iriscolors<-setNames(c("blue","black","orange"),levels(iris$Species))

#Now plot
ggplot(pcaData) +
  aes(PC1, PC2, color = Species, shape = Species) +
```

```
geom_point(size = 2) +
scale_color_manual(values=iriscolors)+ #Adding the color argument
coord_fixed() +
xlab("PC1: 73%")+
ylab("PC2: 23%") +
stat_ellipse(geom="polygon", level=0.95, alpha=0.2,aes(fill=Species),
scale_fill_manual(values=iriscolors)+ #Fill ellipses
theme_bw() +
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())
```



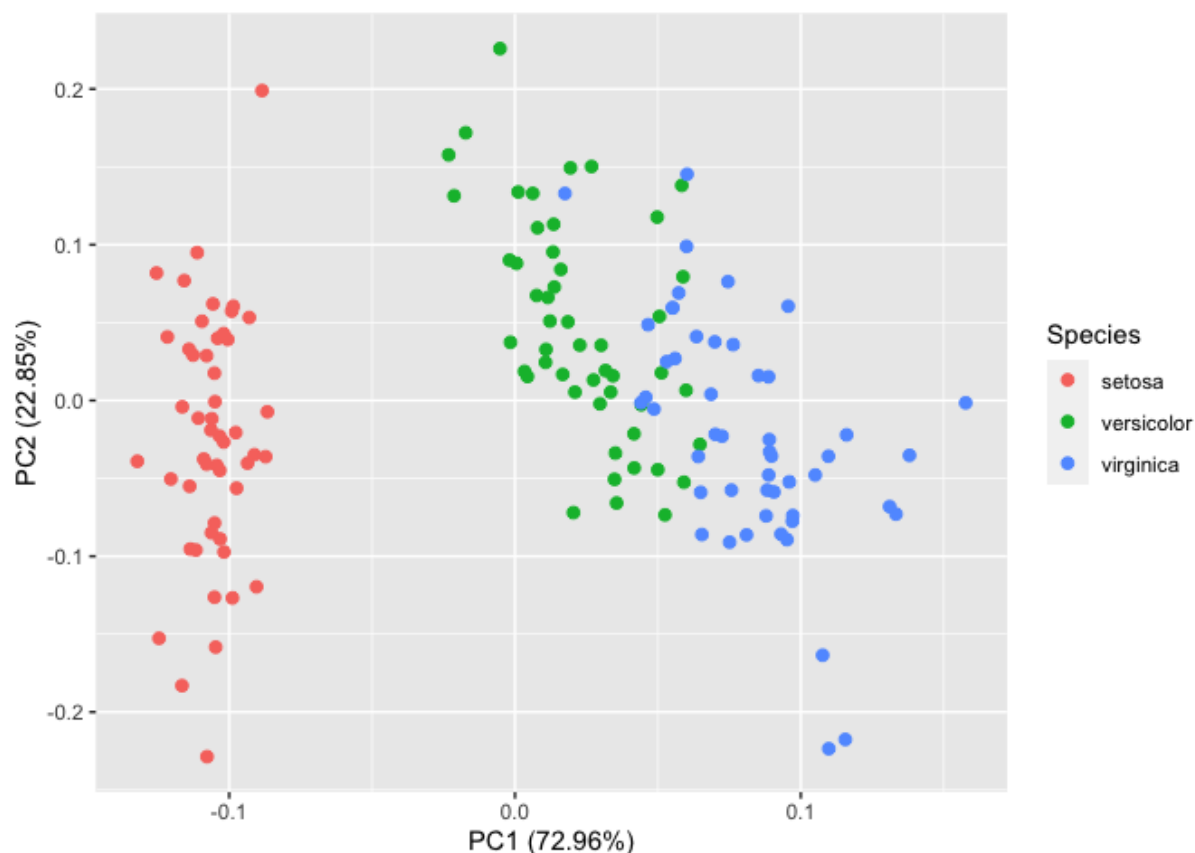
We can use this color palette for all plots of these three species to keep our figures consistent throughout a presentation or publication.

Using ggfortify to make a PCA



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 = iris, colour = 'Species', size=2)
```



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

Putting it all together

Let's update our ggplot2 grammar of graphics template:

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

Creating a publication ready volcano plot

We now know enough to put our new skills to use to make a volcano plot from RNASeq data.

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>)

Introducing EnhancedVolcano ?

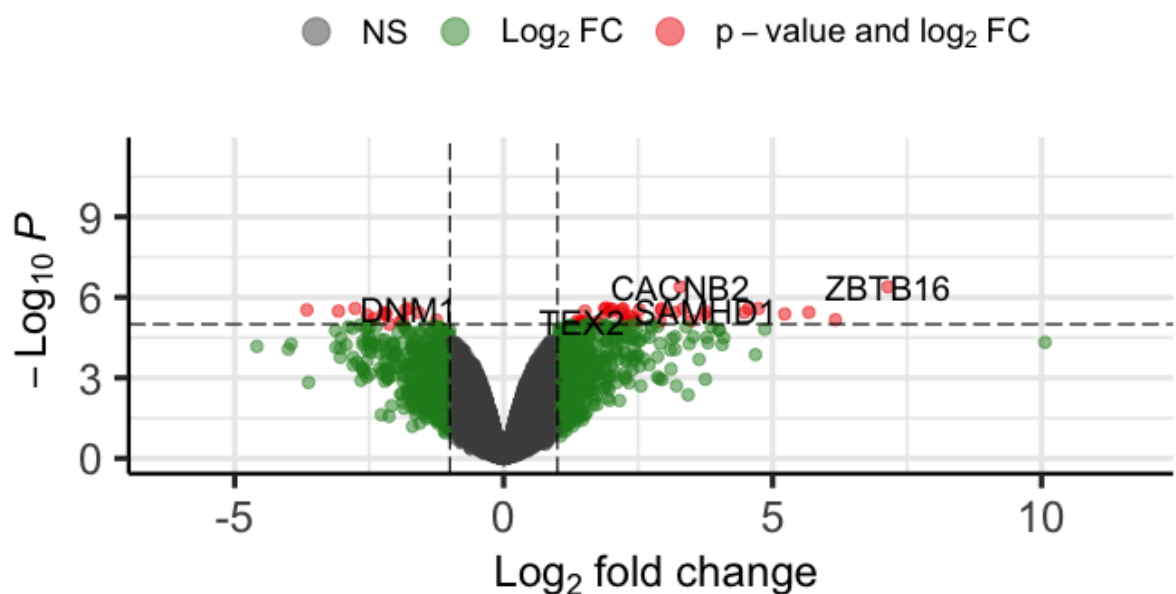
There is a dedicated package for creating volcano plots available on Bioconductor, [EnhancedVolcano](https://bioconductor.org/packages/release/bioc/html/EnhancedVolcano.html) (<https://bioconductor.org/packages/release/bioc/html/EnhancedVolcano.html>). Plots created using this package can be customized 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)
EnhancedVolcano(dexp_sigtrnsc,
                title = "Enhanced Volcano with Airways",
                lab = dexp_sigtrnsc$transcript,
                x = 'logFC',
                y = 'FDR')
```

Enhanced Volcano with Airways

EnhancedVolcano



Let's take a quick look at the data we loaded at the beginning of the lesson:

```
head(dexp_sigtrnsc)
```

```
##           feature albut transcript ref_genome .abundant  logFC
## 1 ENSG00000109906 untrt   ZBTB16    hg38      TRUE   7.146970
## 2 ENSG00000165995 untrt   CACNB2    hg38      TRUE   3.280645
## 3 ENSG00000106976 untrt     DNMT1    hg38      TRUE  -1.764478
## 4 ENSG00000120129 untrt   DUSP1    hg38      TRUE   2.938116
## 5 ENSG00000146250 untrt   PRSS35    hg38      TRUE  -2.761556
## 6 ENSG00000152583 untrt   SPARCL1   hg38      TRUE   4.555765
##           F          PValue          FDR Significant
## 1 1429.3427 5.107615e-11 4.067194e-07      TRUE
## 2 1575.2829 3.342498e-11 4.067194e-07      TRUE
## 3  645.7452 1.616082e-09 2.573772e-06     FALSE
## 4  694.4165 1.179551e-09 2.573772e-06      TRUE
## 5  806.5147 6.162483e-10 2.573772e-06      TRUE
## 6  721.3794 9.999692e-10 2.573772e-06      TRUE
```

```
topgenes
```

```
## [1] "ZBTB16" "CACNB2" "DNMT1" "DUSP1" "PRSS35" "SPARCL1"
```

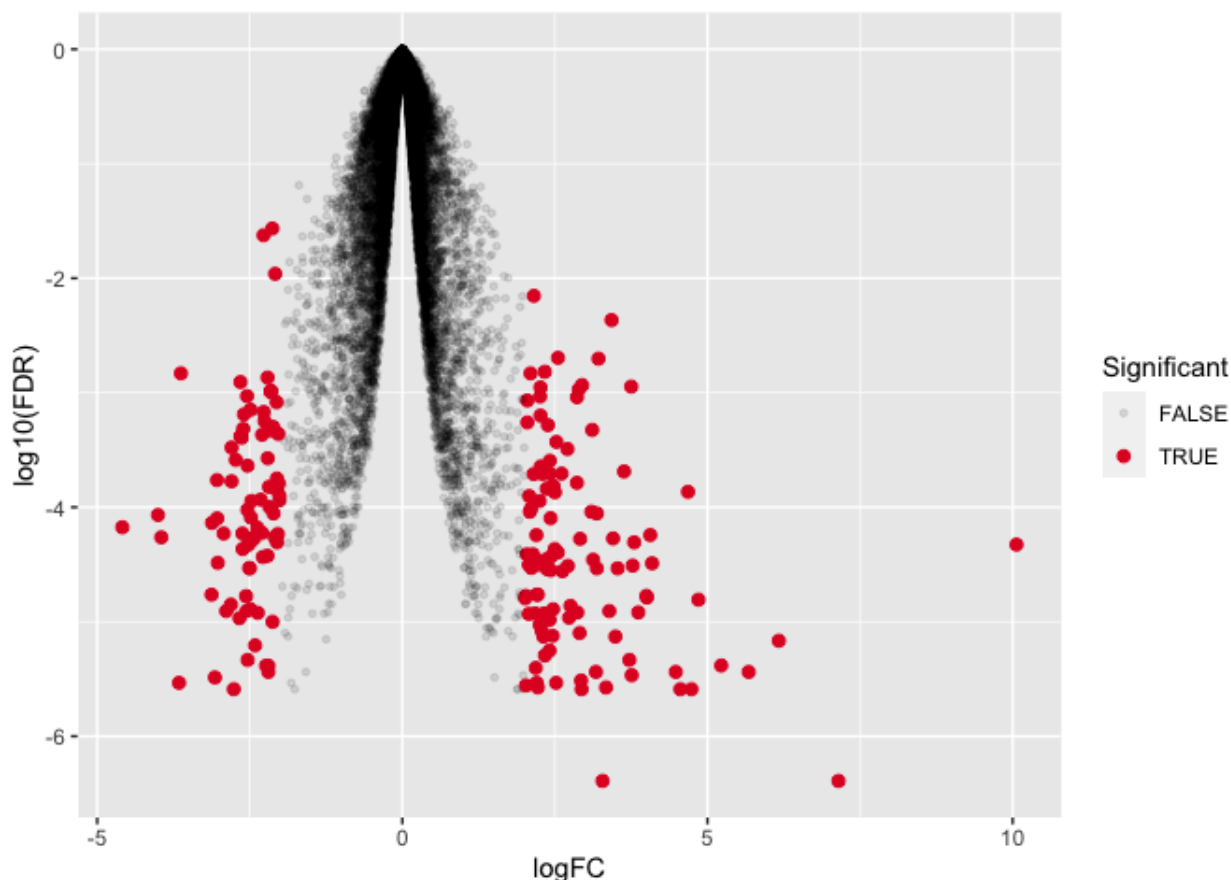
Significant differential expression was assigned based on an absolute log fold change greater than or equal to 2 and an FDR corrected p-value less than 0.05.

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 false discovery rate corrected p-values on the y-axis and set the significant p-values off from the non-significant by size and color. We can also go ahead and customize the size and color scales, since we have learned how to do that.

```
ggplot(data=dexp_sigtrnsc) +
  geom_point(aes(x = logFC, y = log10(FDR), color = Significant,
                size = Significant,
                alpha = Significant)) +
  scale_color_manual(values = c("black", "#e11f28")) +
  scale_size_discrete(range = c(1, 2))
```

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

```
## 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 FDR corrected 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

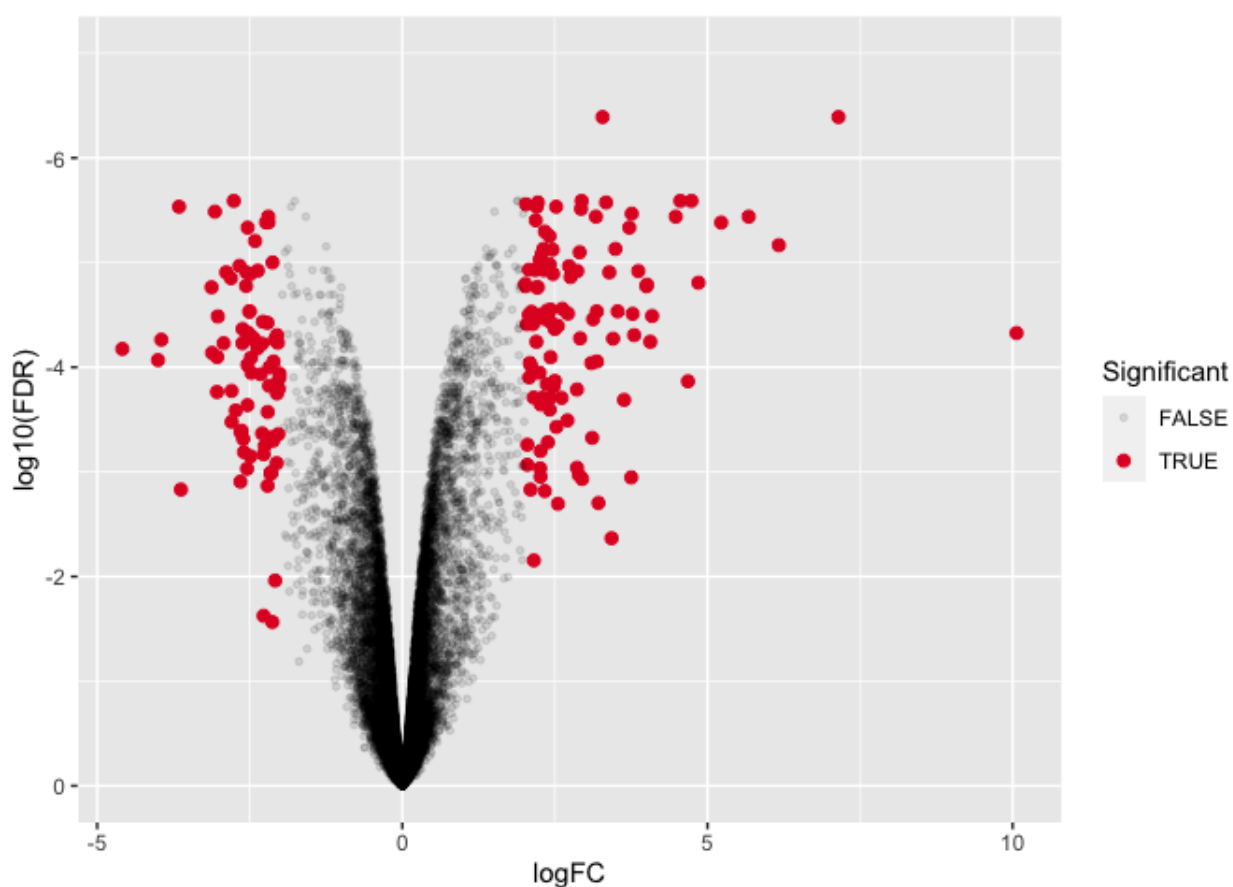
To change the y axis scale, we will need a specific function. These functions generally start with `scale_y...`. In our case we want to reverse our axis so that increasingly negative is going in the positive direction rather than the negative direction. Luckily, there is a function to reverse our axis;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

```
ggplot(data=dexp_sigtrnsc) +
  geom_point(aes(x = logFC, y = log10(FDR), color = Significant,
                 size = Significant,
                 alpha = Significant)) +
```

```
scale_color_manual(values = c("black", "#e11f28")) +
scale_size_discrete(range = c(1, 2)) +
scale_y_reverse(limits=c(0,-7)) #we can also set the limits
```

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

```
## 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()`.

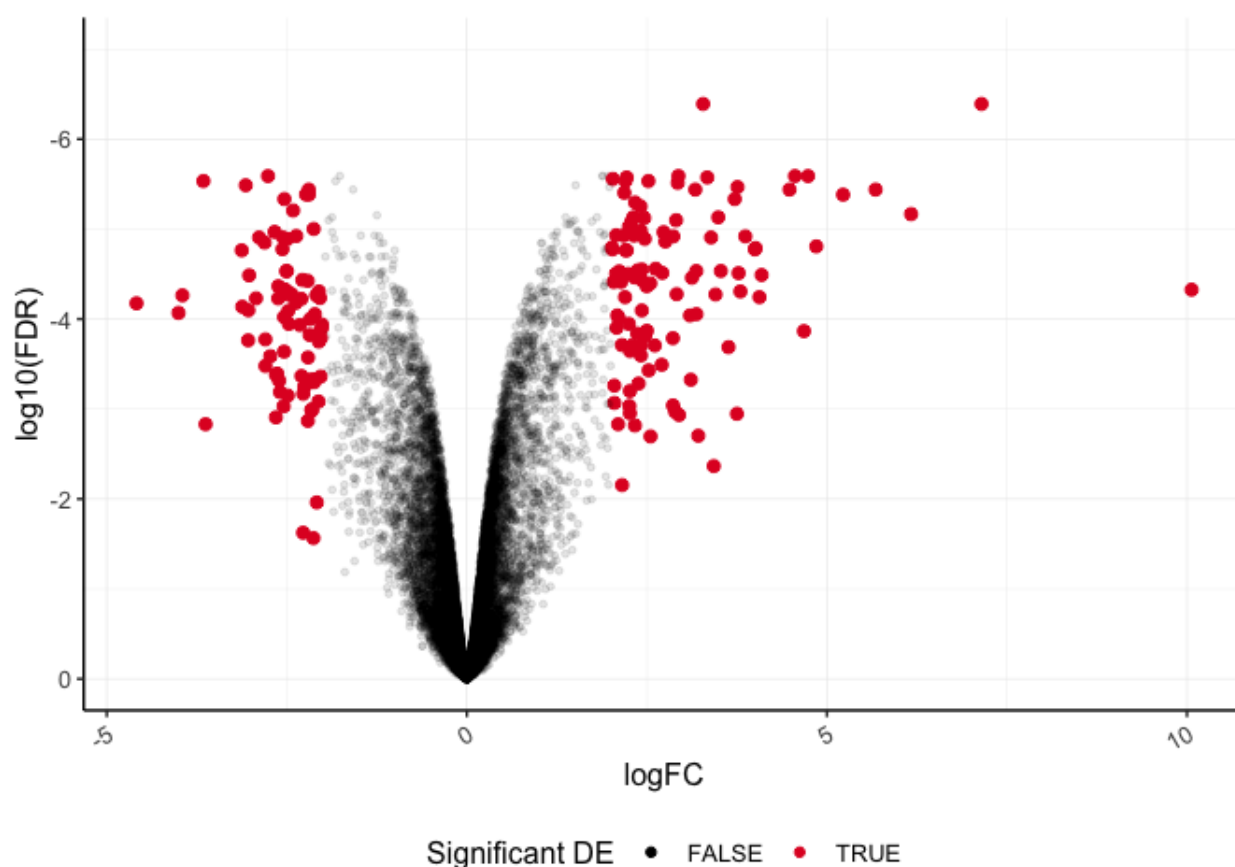
```
ggplot(data=dexp_sigtrnsc) +
  geom_point(aes(x = logFC, y = log10(FDR), color = Significant,
                size = Significant,
```

```
        alpha = Significant)) +
scale_color_manual(values = c("black", "#e11f28")) +
scale_size_discrete(range = c(1, 2)) +
scale_y_reverse(limits=c(0,-7)) + #we can also set the limits
guides(size = "none", 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 size for a discrete variable is not advised.
```

```
## Warning: The `size` argument of `element_line()` is deprecated as
## i Please use the `linewidth` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warni
## generated.
```

```
## 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()`.

```
plot_de<-ggplot(data=dexp_sigtrnsc) +
  geom_point(aes(x = logFC, y = log10(FDR), color = Significant,
                size = Significant,
                alpha = Significant)) +
  geom_text_repel(data=dexp_sigtrnsc %>%
                  filter(transcript %in% topgenes),
                  aes(x = logFC, y = log10(FDR),label=transcript),
                  nudge_y=0.5,hjust=0.5,direction="y",
                  segment.color="gray")+
  scale_color_manual(values = c("black", "#e11f28")) +
  scale_size_discrete(range = c(1, 2)) +
  scale_y_reverse(limits=c(0,-7)) + #we can also set the limits
  guides(size = "none", 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),
```

```

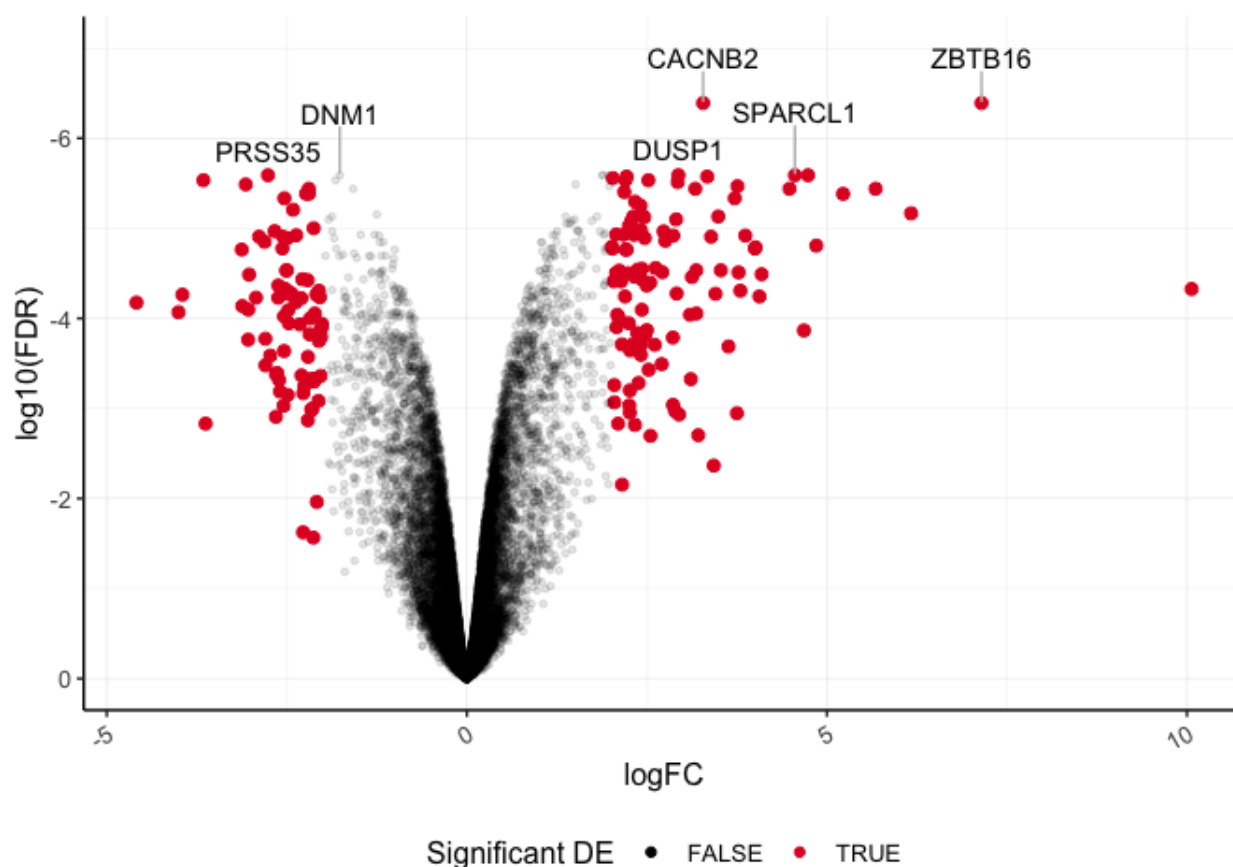
panel.grid.minor = element_line(size = 0.1),
text = element_text(size = 12),
legend.position = "bottom",
axis.text.x = element_text(angle = 30, hjust = 1, vjust = 1)
)

```

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

```
plot_de
```

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



Save to an .rds file

We want to use this in a multi-panel figure in a later lesson, so let's go ahead and save it to a file that will hold a single R object (.rds).

```
saveRDS(plot_de, "volcanoplot.rds")
```

References

Code for PCA was adapted from [Learning R through examples \(https://gexijin.github.io/learnR/index.html\)](https://gexijin.github.io/learnR/index.html) by Xijin Ge, Jianli Qi, and Rong Fan, 2021. Other sources for content included [R4DS \(https://r4ds.had.co.nz/\)](https://r4ds.had.co.nz/) and [Holmes and Huber, 2021 \(https://web.stanford.edu/class/bios221/book/\)](https://web.stanford.edu/class/bios221/book/).

Stat Transformations: Bar plots, box plots, and histograms

Objectives

1. Review the grammar of graphics template
2. Learn about the statistical transformations inherent to geoms
3. Review data types
4. Create bar plots, box & whisker plots, and histograms.

Our grammar of graphics template

This is where we left off at the end of lesson 3.

Our grammar of graphics template

- one or more datasets,
- one or more geometric objects that serve as the visual representations of the data, – for instance, points, lines, rectangles, contours,
- descriptions of how the variables in the data are mapped to visual properties (aesthetics) of the geometric objects, and an associated scale (e. g., linear, logarithmic, rank),
- a facet specification, i.e. the use of multiple similar subplots to look at subsets of the same data,
- one or more coordinate systems,
- optional parameters that affect the layout and rendering, such text size, font and alignment, legend positions.
- **statistical summarization rules**

---(Holmes and Huber, 2021 (<https://web.stanford.edu/class/bios221/book/Chap-Graphics.html>))

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

```
) +  
<FACET_FUNCTION> +  
<COORDINATE_SYSTEM> +  
<THEME>
```

While we added `stat_ellipse()` to our PCAs, scatter plots do not have a built in statistical transformation. Today, we will talk more about statistical transformations and how these impact our plot representations. For a list of available statistical transformations in `ggplot2` see <https://ggplot2-book.org/layers.html?q=stat#stat> (<https://ggplot2-book.org/layers.html?q=stat#stat>).

Libraries

In this lesson, we will continue to use the `ggplot2` package for plotting. To use `ggplot2`, we first need to load it into our R work environment using the `library` command.

```
library(ggplot2)
```

The Data

In this lesson we will use data obtained from a study that examined the effect that dietary supplements at various doses have on guinea pig tooth length. This data set is built into R, so if you want to take a look for yourself you can type `data("ToothGrowth")` either in the console or in a script. But in this exercise, we will import the data to our R work environment because it is more likely that we will import our own data for analysis.

Here, we are going to import two data sets using `read.delim`. The file `data1.txt` contains raw data from the tooth growth study. The file `data2.txt` is summary level data with mean tooth length and standard deviation pre-computed. We will assign `data1.txt` to object `a1` and `data2.txt` to object `a2`. Within `read.delim` we use `sep='\t'` to indicate that the columns in the data are tab separated. After importing, we use the `head` command to look at the first 6 rows of each data set.

```
a1 <- read.delim("./data/data1.txt", sep='\t')  
a2 <- read.delim("./data/data2.txt", sep='\t')  
head(a1)
```

```
##      len supp dose  
## 1  4.2   VC  0.5  
## 2 11.5   VC  0.5  
## 3  7.3   VC  0.5  
## 4  5.8   VC  0.5
```

```
## 5  6.4  VC  0.5
## 6 10.0  VC  0.5
```

head(a2)

```
##   supp dose treat mean_len      sd
## 1  OJ  0.5 OJ0.5   13.23 4.459709
## 2  OJ  1.0 OJ1    22.70 3.910953
## 3  OJ  2.0 OJ2    26.06 2.655058
## 4  VC  0.5 VC0.5    7.98 2.746634
## 5  VC  1.0 VC1    16.77 2.515309
## 6  VC  2.0 VC2    26.14 4.797731
```

The tooth growth data set measured tooth length for two supplement types (OJ - orange juice, VC - vitamin c) at three different doses (0.5, 1, and 2). Each supplement and dose combination has 10 measurements so we have a total of 60 measurements in this data set. In a1, we have the raw data. On the other hand, in a2, we pre-computed the mean tooth length and standard deviation for the 10 measurements taken at each supplement and dose combination.

The column headings (`colnames(a1),colnames(a2)`) in the raw data (a1) and summary level data (a2) are as follows:

- len (tooth length - in raw data)
- supp (supplement type)
- dose
- treat (treatment, which is a concatenation of supp and dose - in summary level data only)
- mean_len (mean tooth length - in summary level data only)
- sd (standard deviation - in summary level data only)

Variable Types

Before diving into the construction of bar plot, box & whisker plot, and histogram, we should do a quick review of the types of variables that we commonly work with in data analysis.

- **Categorical variables** are qualitative, for instance
 - patient gender (male or female)
 - disease status (case or control)
- **Discrete variables** are quantifiable and can take on a finite number of values, for instance
 - number of male or female patients
 - number of controls or disease cases
 - medication dose
- **Continuous variables** can take on an infinite number of values

- age
- height
- weight
- **Independent variable** is a variable whose variation does not depend on another
- **Dependent variable** is a variable whose variation depends on another
- **Factors** are independent variables in which an experimental outcome is dependent on.
Levels are variations in the factors.

The `stat` argument

Until this point we have been plotting raw data with `geom_point()`, but now we will be introducing geoms that transform and plot new values from your data.

Many graphs, like scatterplots, plot the raw values of your dataset. Other graphs, like bar charts, calculate new values to plot:

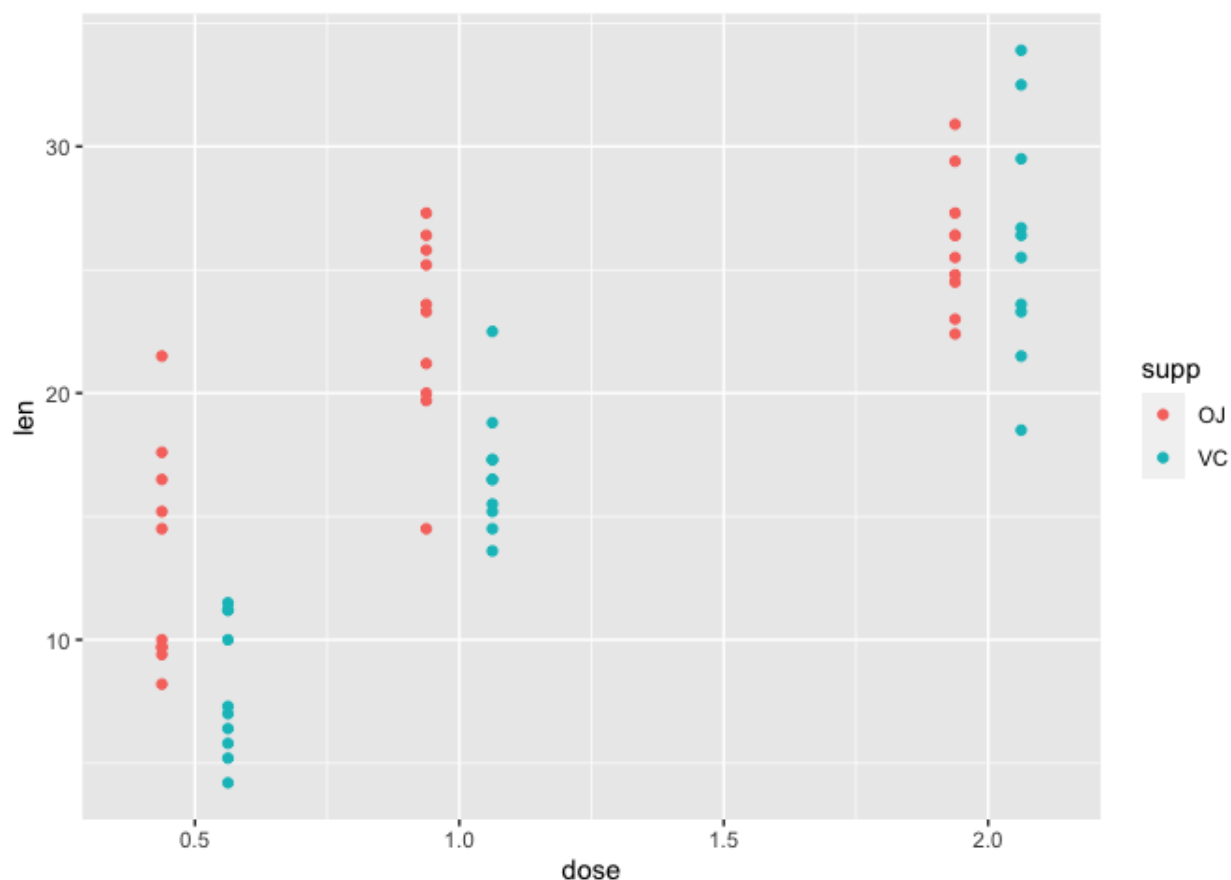
- bar charts, histograms, and frequency polygons bin your data and then plot bin counts, the number of points that fall in each bin.
- smoothers fit a model to your data and then plot predictions from the model.
- boxplots compute a robust summary of the distribution and then display a specially formatted box.

Let's explore the tooth growth data using plots. The tooth growth data has two independent variables, supplement and dose (variables `supp` and `dose`, respectively) for which the dependent variable tooth length (`len`) is measured. We would like to use the scatter plot to learn about tooth growth as a function of both dose and supplement (`supp`). Remember from Lesson 3 that a scatter plot can be generated using `geom_point`.

Within the aesthetic mapping in `geom_point`, we assign `dose` to the x axis, `len` (tooth length) to y, and the second independent variable `supp` (supplement) by assigning it to `color`. This will give us a scatter plot where `dose` is plotted along the x axis and `len` plotted along the y axis. The color code indicating which supplement (`supp`) each of the points or measurements came from is provided in the legend. In short, the color argument allows us to visualize how the dependent variable changes as a function of two independent variables.

Within `geom_point`, `position=position_dodge` is included to separate the points by supplement (`supp`), otherwise the points for the two supplements will overlap each other. The parameter `width` in `position_dodge` allows us to specify how far apart we want the points for the different supplement groups to be.

```
ggplot(data=a1)+
  geom_point(mapping=aes(x=dose,y=len,color=supp),
             position=position_dodge(width=0.25))
```



Bar Plot

A barplot is used to display the relationship between a numeric and a categorical variable. --- [R Graph Gallery \(https://r-graph-gallery.com/barplot.html\)](https://r-graph-gallery.com/barplot.html)

The tooth growth data can also be visualized via bar plot using `geom_bar`. However, if we try to plot tooth length (`len`) across each dose using the code below, we will get an error. We get this error because `geom_bar` uses `stat_count` by default where it counts the number of cases at each x position. Thus, by default, `geom_point` does not require y axis.

```
ggplot(data=a1)+
  geom_bar(mapping=aes(x=dose,y=len))
```

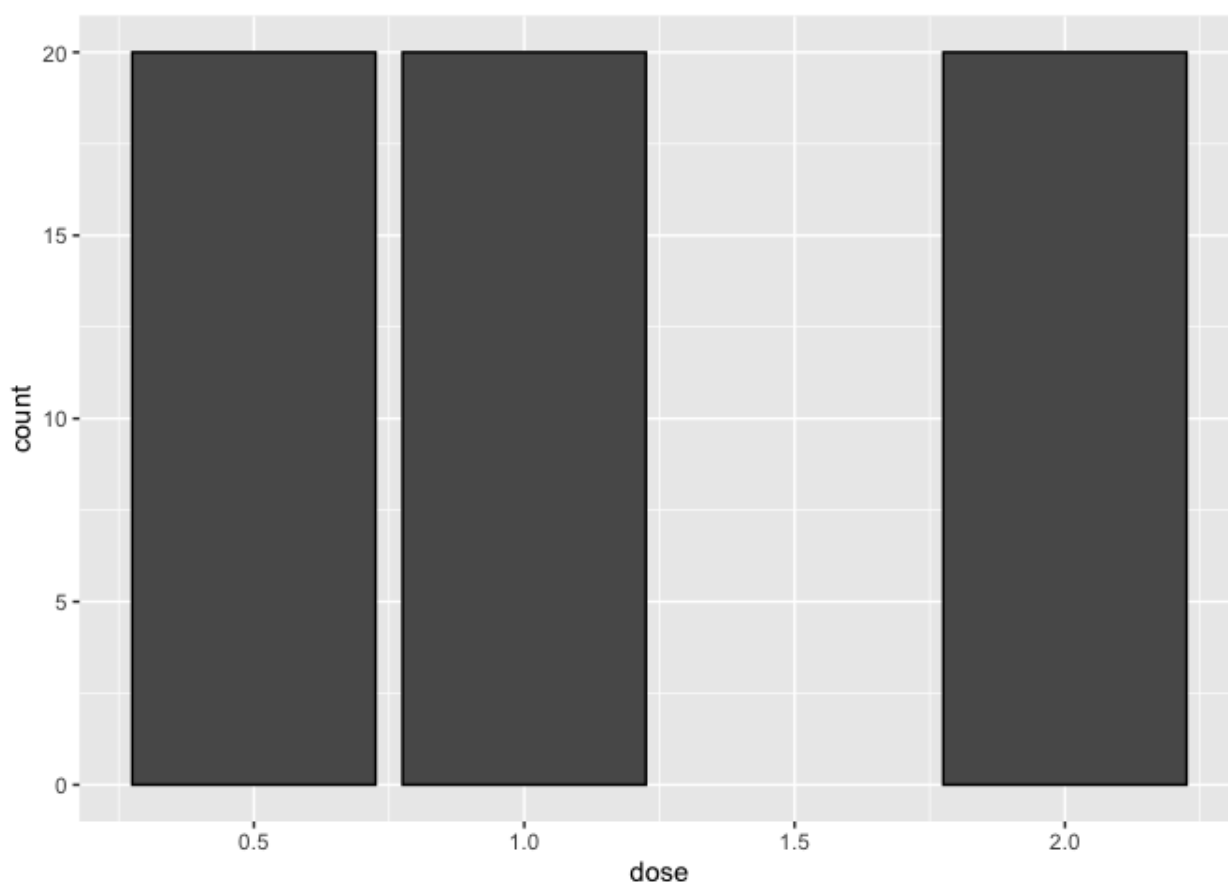
```
## Error in `f()`:
## ! stat_count() can only have an x or y aesthetic.
```


We received an error referencing `stat_count()`.

stat = count

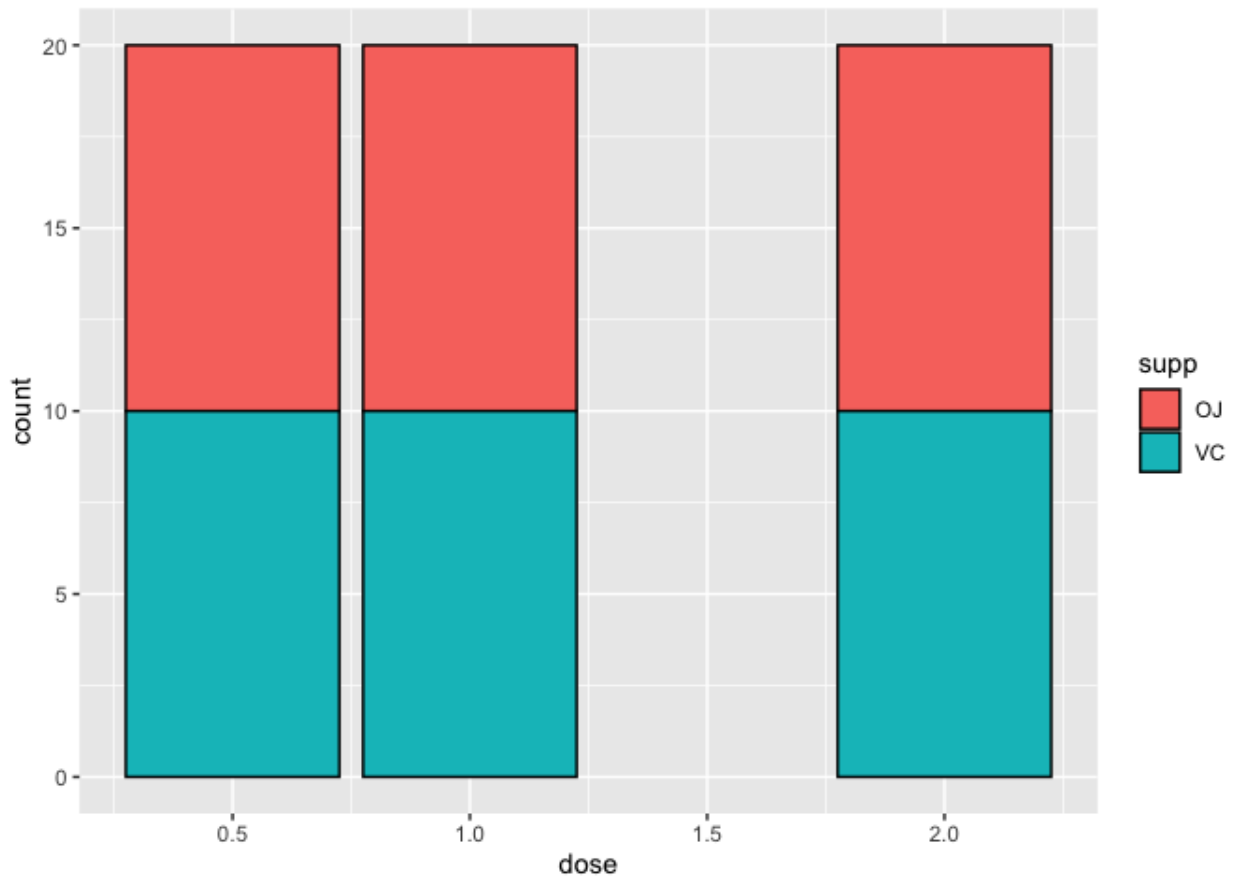
Let's take a look at a bar plot constructed using the default `stat="count"` transformation. Below, we plot the number of tooth length measurements taken at each dose. Setting `color="black"` allows us to include a black outline to the bars for better readability. In accordance with the description of this data, we see from the plot that 20 measurements were taken at each dose.

```
ggplot(data=a1)+  
  geom_bar(mapping=aes(x=dose), color="black")
```



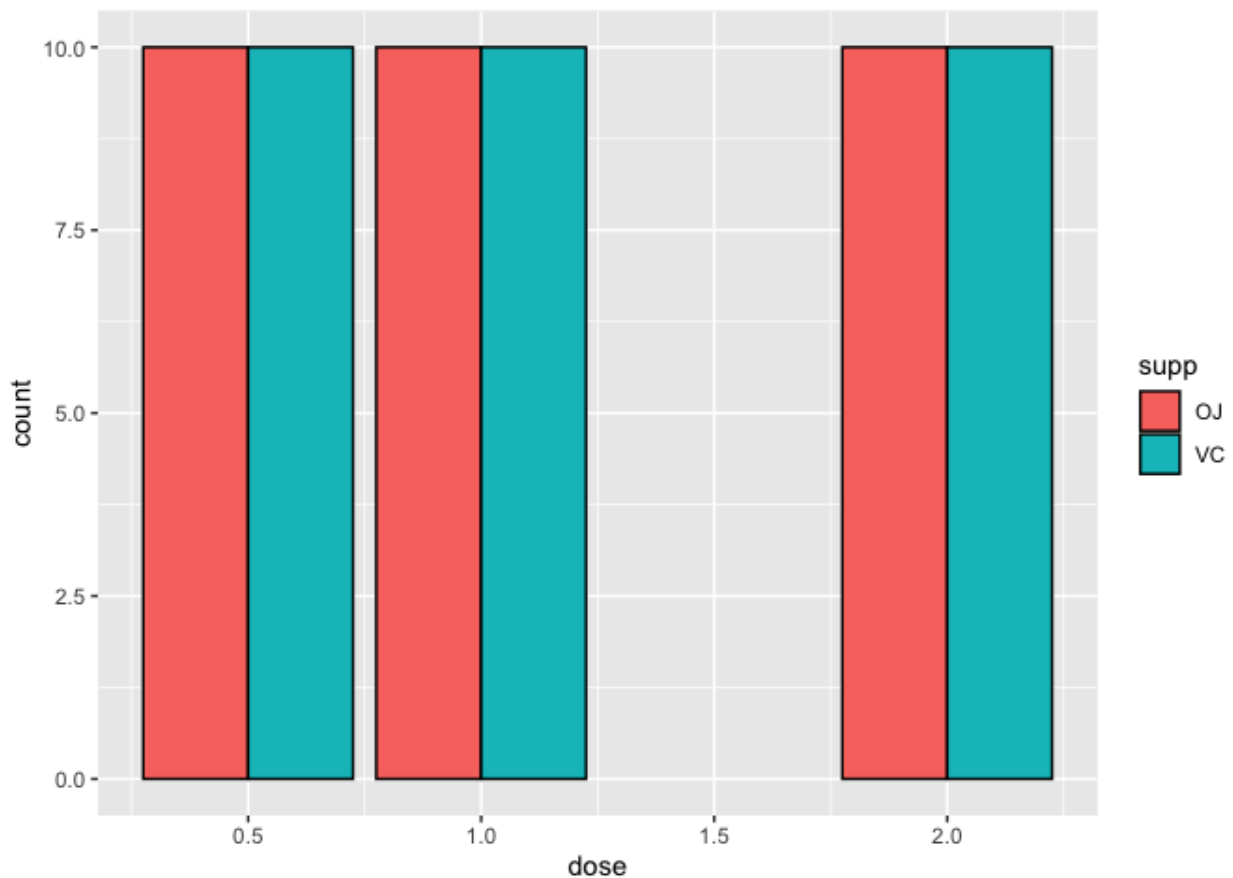
Given the above plot, how many of the 20 measurements taken at each dose came from the OJ or VC group. To find out, we can set `fill=supp`. Like `color` in the scatter plot, `fill`, allows us to include a second independent variable in our graph. The plot below tells us that 10 measurements were taken from the OJ group and 10 were taken from the VC group at each dose.

```
ggplot(data=a1)+  
  geom_bar(mapping=aes(x=dose, fill=supp), color="black")
```



By default, `geom_bar` stacks bars from different groups. If we do not like the arrangement, we can use `position_dodge` to arrange the bars from the OJ and VC groups side-by-side.

```
ggplot(data=a1)+  
  geom_bar(mapping=aes(x=dose, fill=supp), color="black",  
            position=position_dodge())
```



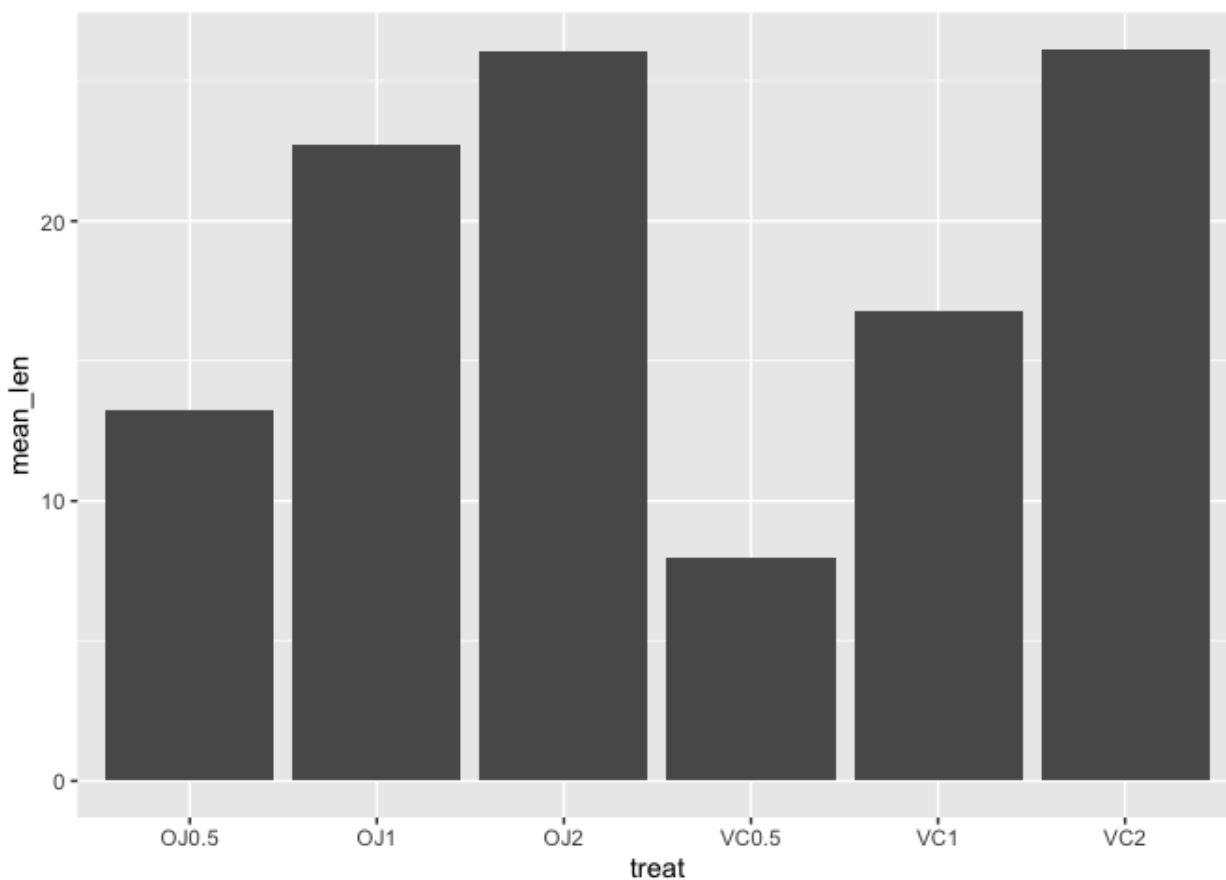
`stat_count()` is a great way to compare differences in sample sizes between treatment categories or other variables and can be a great way to explore and represent metadata associated with -omics data.

stat = identity

Above, we learned about the number of tooth length measurements taken at each dose and supplement combination using the default `stat="count"` transformation of `geom_bar`. But what if we want to specify a y axis and plot exactly the values of our dependent variable in the y axis? This can be done in `geom_bar` by setting `stat="identity"`.

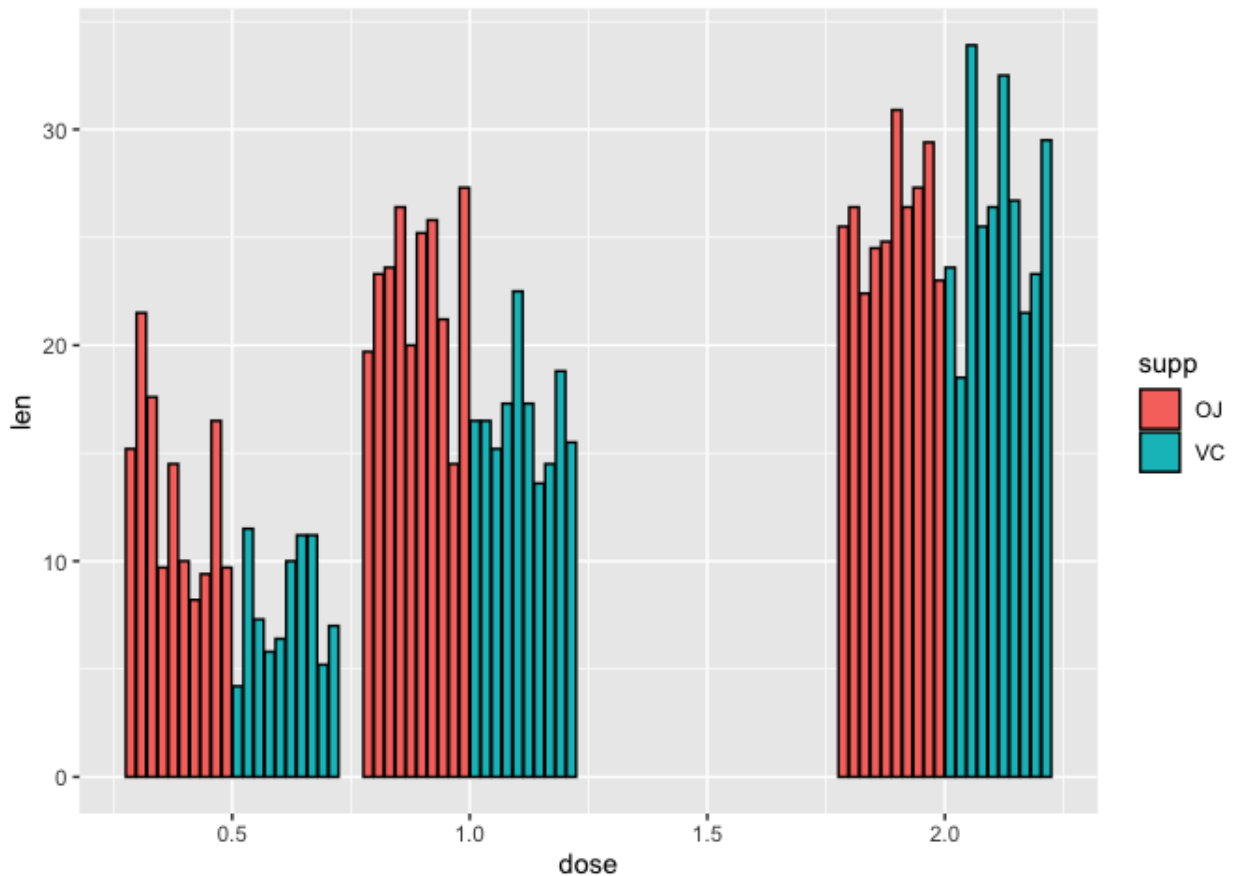
Below, we are plotting the mean tooth length (`mean_len`) across each of the treatment groups (`treat`) using the summary level data, `a2`. Using `stat="identity"`, the exact y value or `mean_len` is plotted.

```
ggplot(data=a2)+
  geom_bar(mapping=aes(x=treat,y=mean_len),stat="identity")
```



What if we wanted to look at the raw values across the treatment groups using a bar plot? We still use `geom_bar` but the aesthetic mapping will be similar to the scatter plot except we are filling the bars to provide color coding for the supplements using `fill`. Again, because we want ggplot2 to plot exact value in the y axis, we specify `stat="identity"` inside `geom_bar`. To avoid stacking the values, we can use `position_dodge2` in `geom_bar` to visualize each of the 10 measurements taken at each supplement and dose combination arranged side-by-side.

```
ggplot(data=a1)+  
  geom_bar(mapping=aes(x=dose,y=len,fill=supp),stat="identity",  
            position=position_dodge2(),color="black")
```



Note that because R interprets dose as numeric continuous variable (`class(a2$dose)`) ggplot2 gives us an extra dose of 1.5. But, the study did not measure tooth length at a dose of 1.5 for either of the supplements. Thus, we would like to remove this dose.

Using factors

```
class(a1$dose)
```

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

As it turns out, dose is really an experimental factor, so if we specify `factor(dose)` it will be interpreted as categorical or discrete. Before fixing the x axis in the above plot, let's diverge briefly and speak about factors in R.

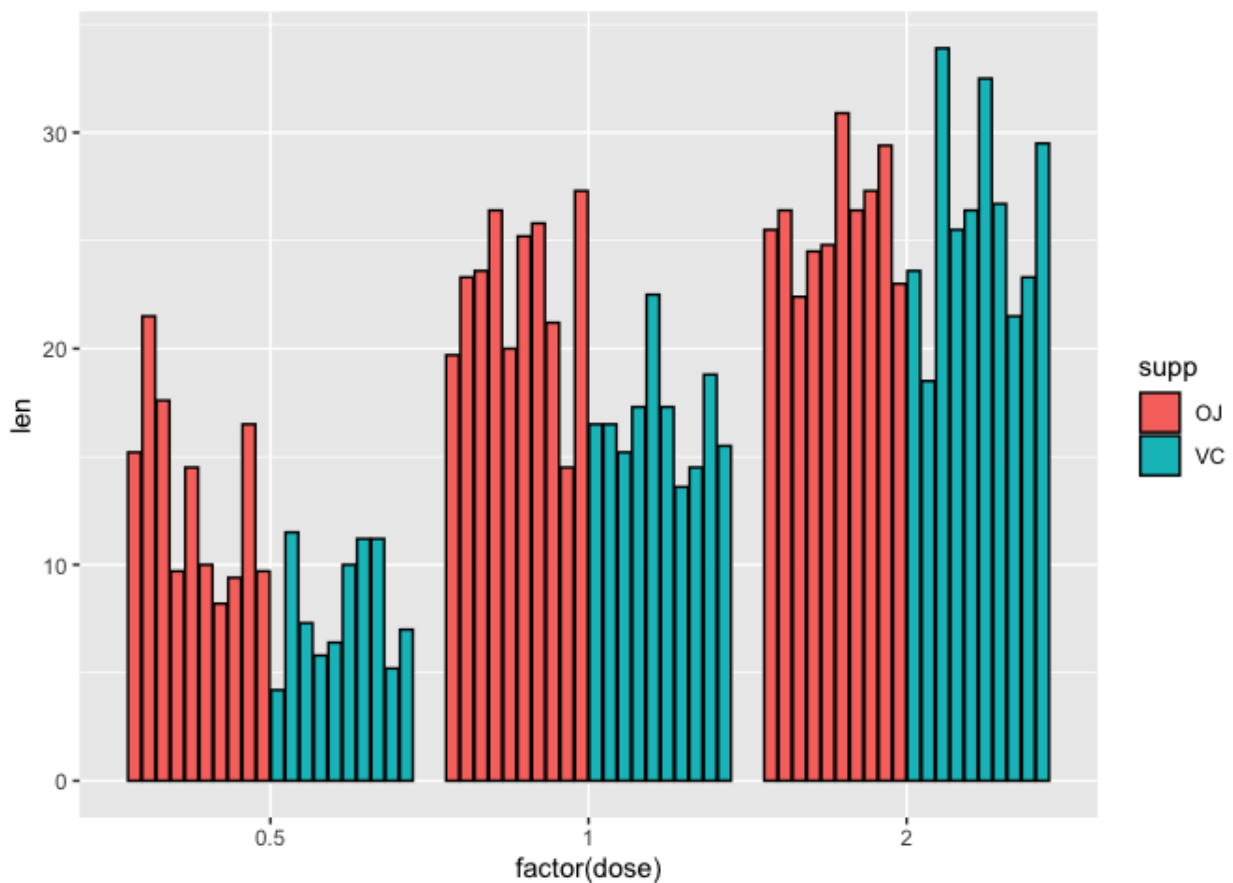
Using the `factor` function we see that there are three levels for dose (0.5, 1, and 2)

```
factor(a1$dose)
```

```
## [1] 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 1 1 1 1 1 :
## [20] 1 2 2 2 2 2 2 2 2 2 2 2 0.5 0.5 0.5 0.5 (
## [39] 0.5 0.5 1 1 1 1 1 1 1 1 1 1 1 2 2 2 ;
## [58] 2 2 2
## Levels: 0.5 1 2
```

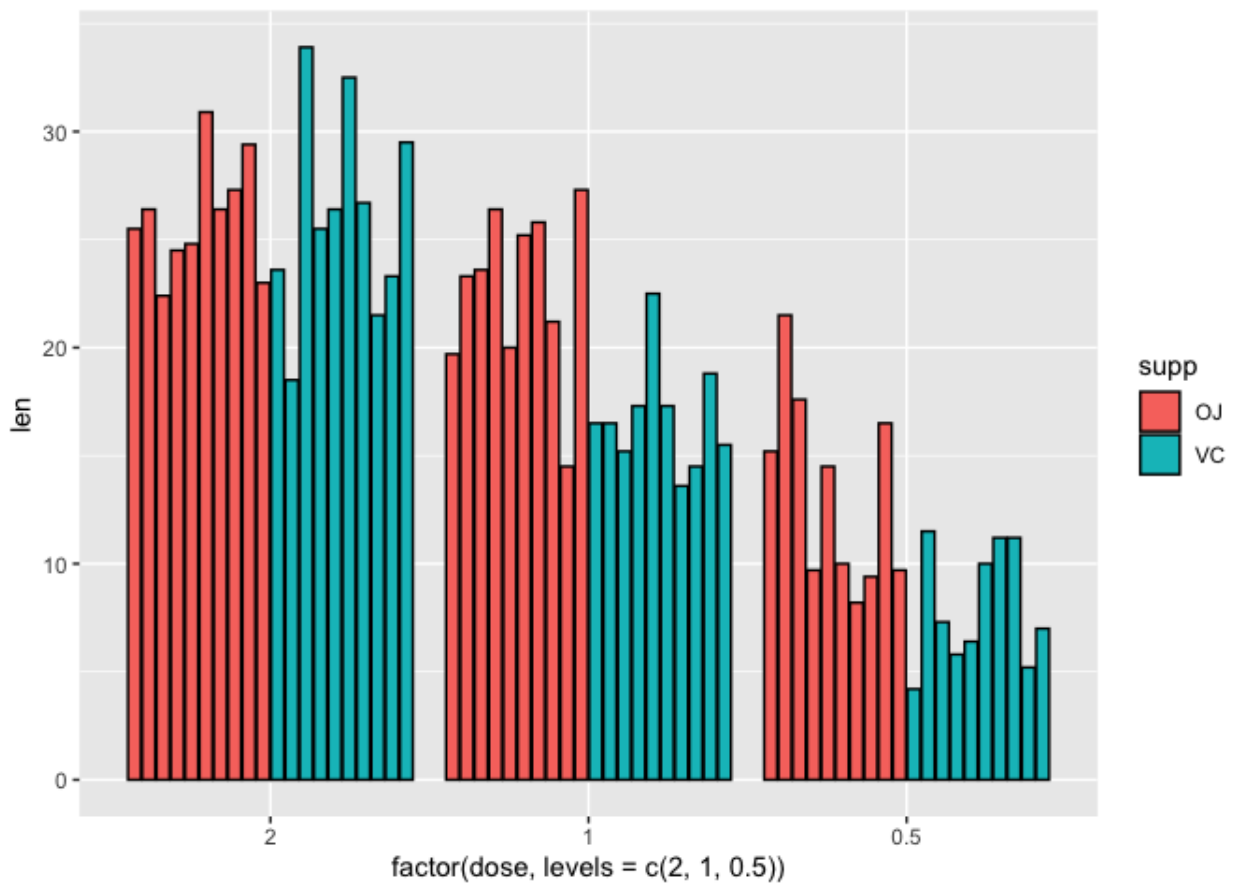
To remove the dose of 1.5, we can set the x axis to factor (dose).

```
ggplot(data=a1)+
  geom_bar(mapping=aes(x=factor(dose),y=len,fill=supp),stat="identity",
           position=position_dodge2(),color="black")
```



We can reorder factors using the function `factor`. For instance if we want to plot the doses backwards from highest to lowest on the x axis (i.e., 2, 1, 0.5) we can set the x axis in aesthetic mapping of `geom_bar` to `factor(dose,levels=c(2,1,0.5))`, where in the levels parameter, we are reassigning the order of the levels.

```
ggplot(data=a1)+
  geom_bar(mapping=aes(x=factor(dose,
    levels=c(2,1,0.5)),y=len,fill=supp),stat="identity",
    position=position_dodge2(),color="black")
```

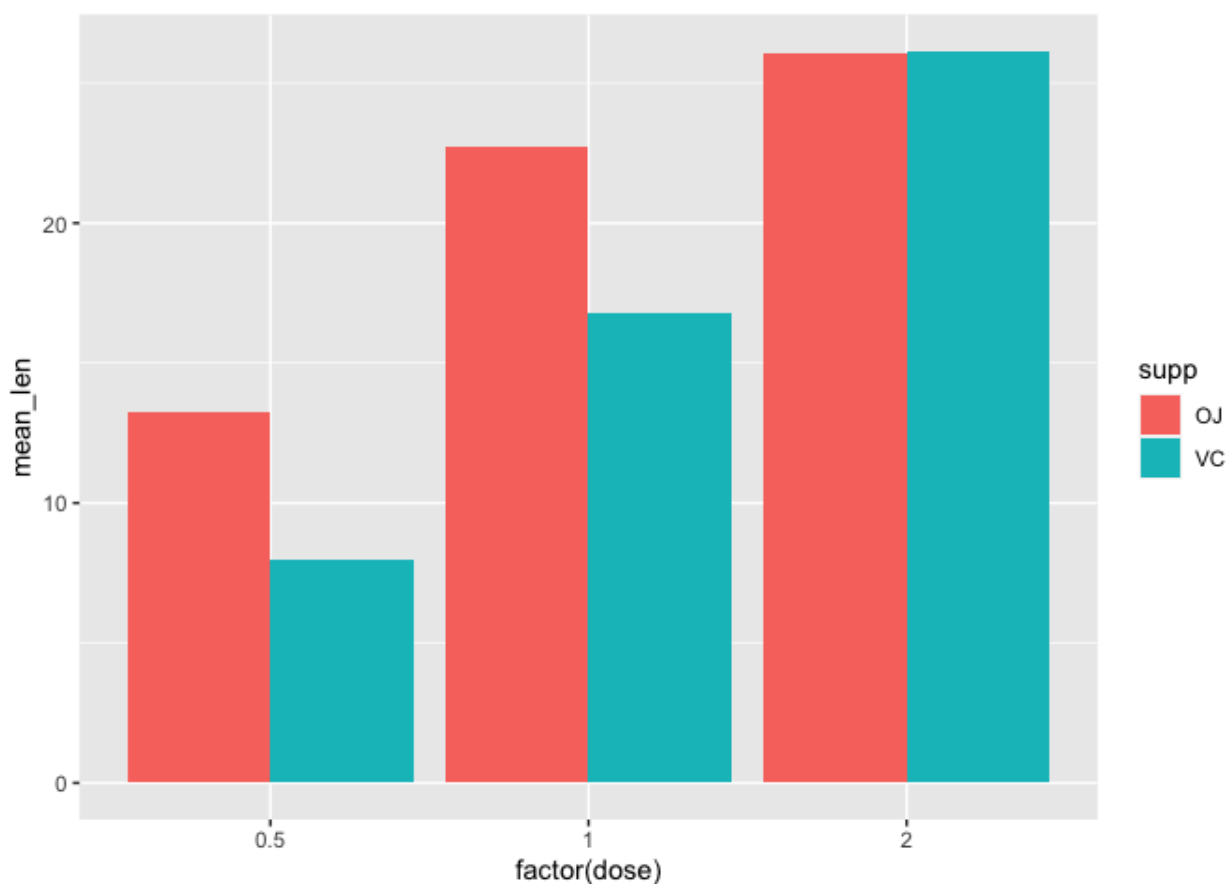


Adding error bars

Often, we will use bar plots to illustrate mean plus minus standard deviation in our data so we should learn how to incorporate error bars in our plots. We will learn to do this using the summary level data (a2) where the mean and standard deviation have been pre-computed and then with the raw data (a1) to take advantage of more statistical transformations.

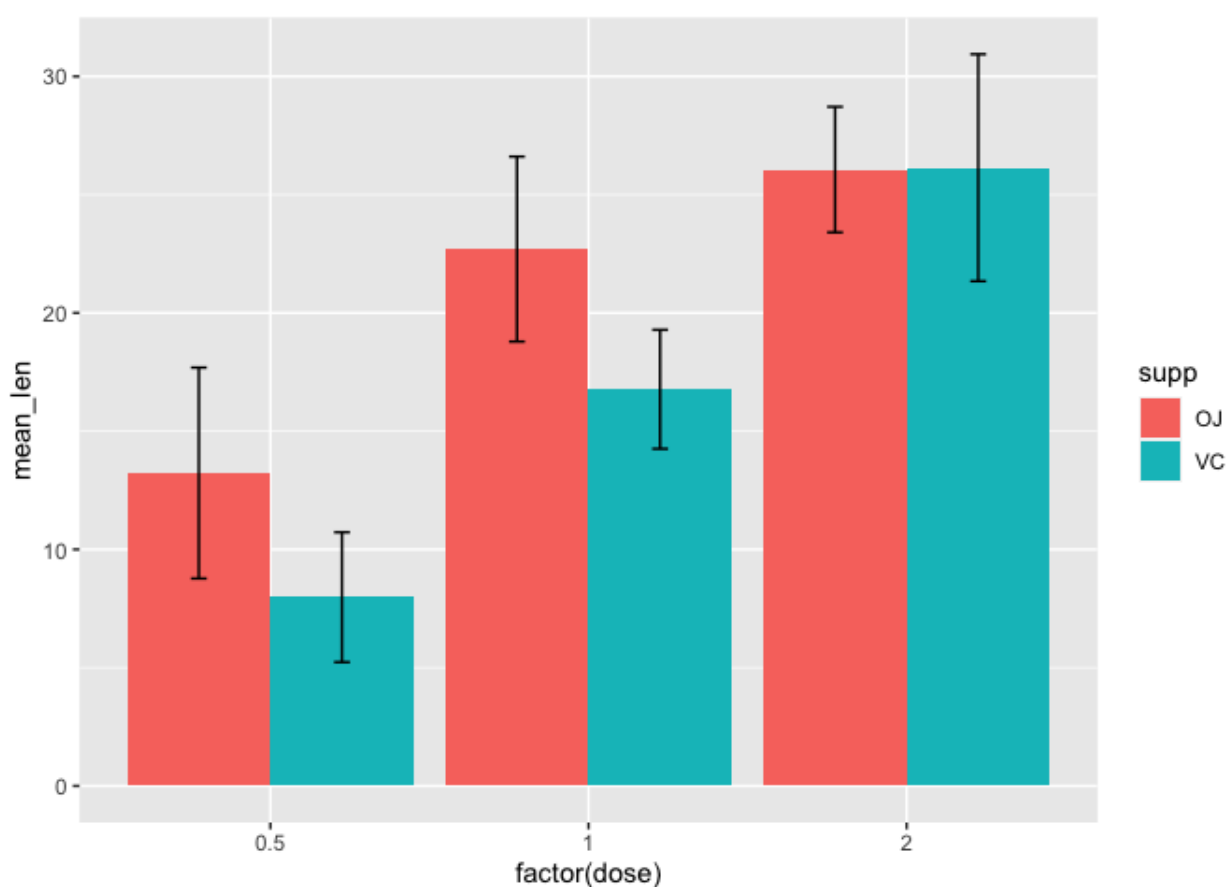
First, we use the summary level data (a2) to plot the mean tooth length (`mean_len`) across the treatment groups, remembering to set `stat="identity"` because we are providing a y axis and `position=position_dodge` to arrange the bars side-by-side.

```
ggplot(data=a2, mapping=aes(x=factor(dose),y=mean_len,fill=supp))+
  geom_bar(stat="identity", position=position_dodge())
```



Next, we add `geom_errorbar` to incorporate error bars that illustrate plus/minus 1 standard deviation from the mean. To set the upper and lower bounds of the error bar, we simply set `ymin=mean_len-sd` and `ymax=mean_len+sd` within the aesthetic mapping of `geom_errorbar`. Setting `position=position_dodge` in `geom_errorbar` again allows us to separate the error bars from each of the supplement groups. Within the `position_dodge` argument we set `width=0.9` to help center the error bars with their respective bars. The `width` parameter within `geom_errorbar` allows us to adjust the error bar width (set to 0.1 here).

```
ggplot(data=a2, mapping=aes(x=factor(dose),y=mean_len,fill=supp))+
  geom_bar(stat="identity", position=position_dodge())+
  geom_errorbar(aes(ymin=mean_len-sd,ymax=mean_len+sd),
               position=position_dodge(width=0.9), width=0.1)
```

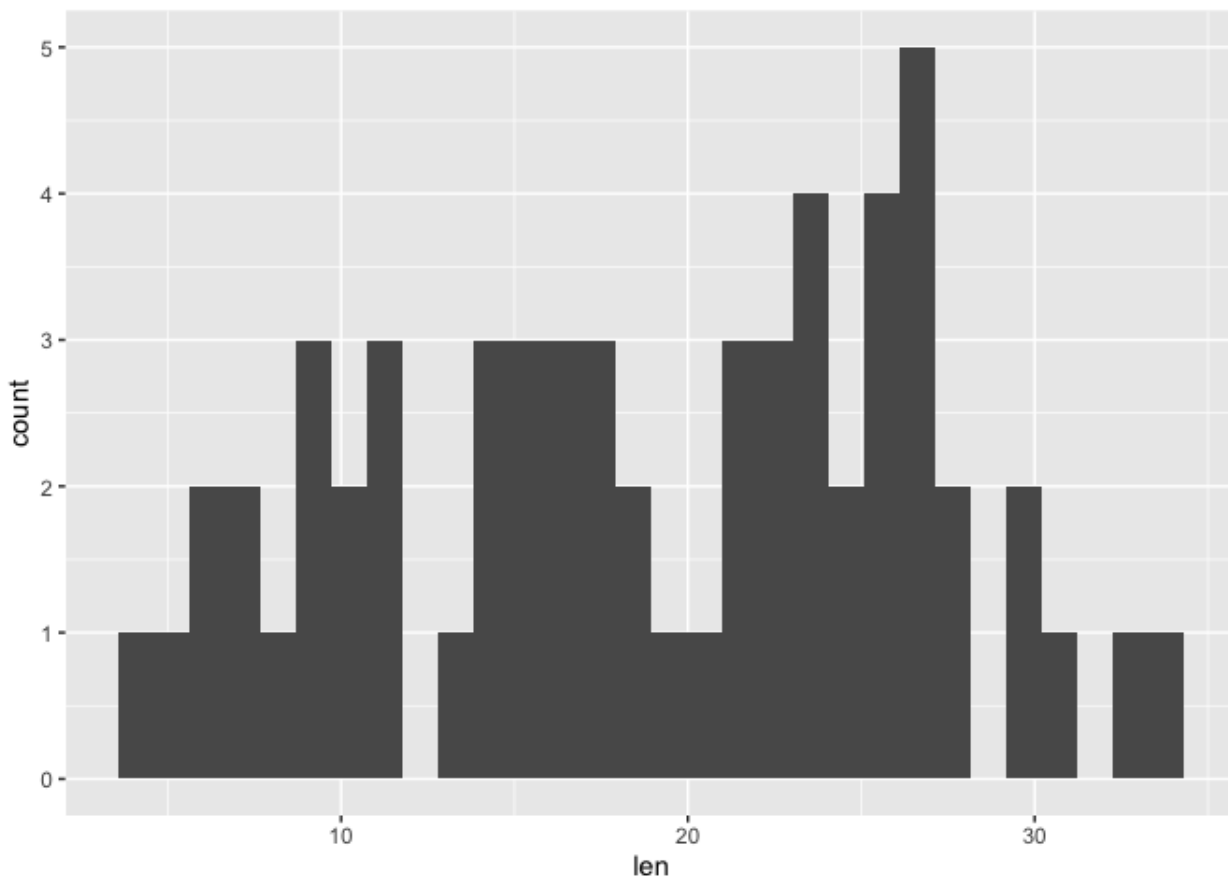
Histogram

Understanding data distribution can help us decide appropriate downstream steps in analysis such as which statistical test to use. A histogram is a good way to visualize distribution. It divides the data into bins or increments and informs the number of occurrences in each of the bins. Thus, the default statistical transformation for `geom_histogram` is `stat_bin`, which bins the data into a user specified integer (default is 30) and then counts the occurrences in each bin. In `geom_histogram` we have the ability to control both the number of bins through the `bins` argument or `binwidth` through the `binwidth` argument. Important to note is that `stat_bin` only works with continuous variables.

Below we constructed a basic histogram using the `len` column in `a1` (the raw data for the Tooth Growth study). Note that within `geom_histogram` we do not need to explicitly state `stat="bin"` because it is default. The histogram below is not very aesthetically pleasing - there are gaps and difficult to see the separation of the bins.

```
ggplot(data=a1, mapping=aes(x=len))+  
geom_histogram()
```

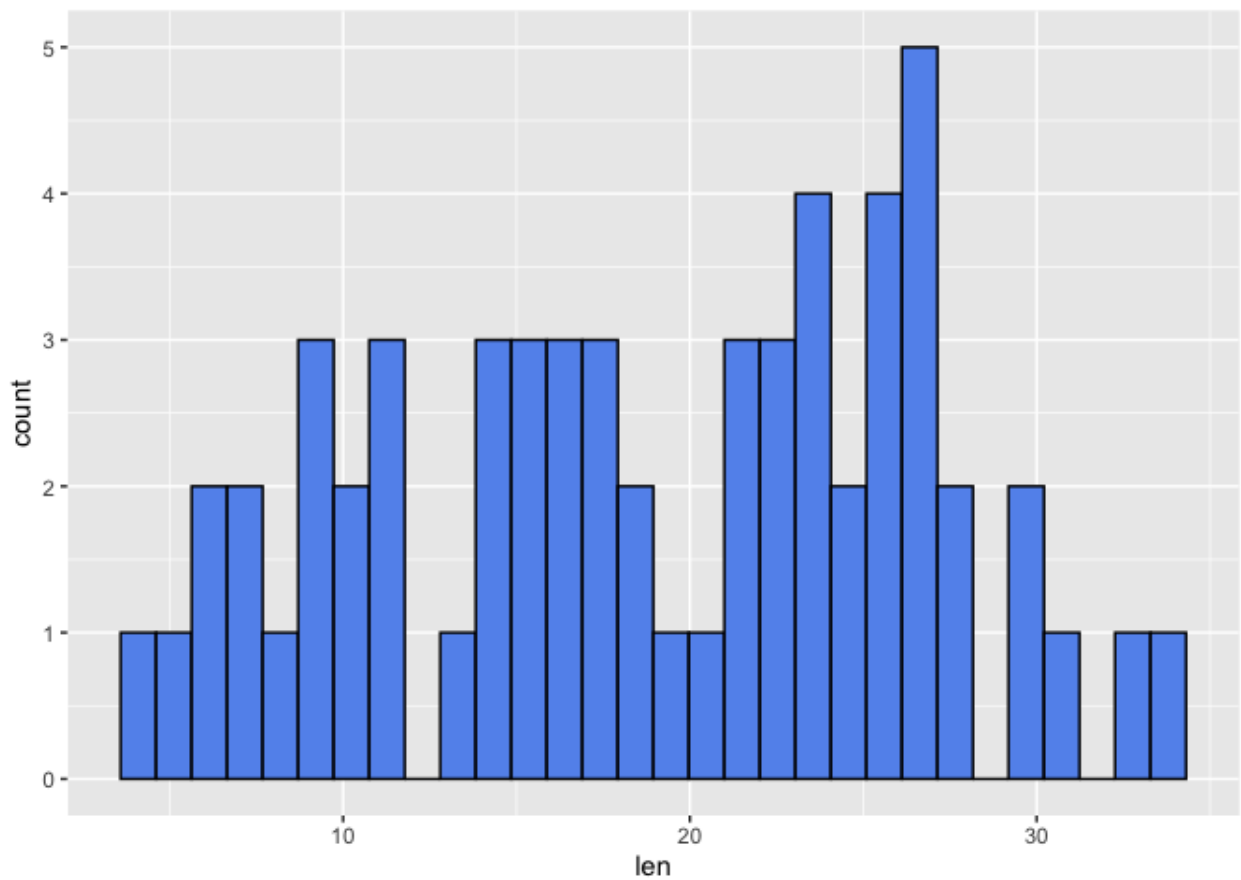
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



First, we will use the `color` argument in `geom_histogram` to assign a border color to help distinguish the bins. Then we use the `fill` argument in `geom_histogram` to change the bars associated with the bins to a color other than gray. Below we have a histogram of tooth length with a default bin of 30.

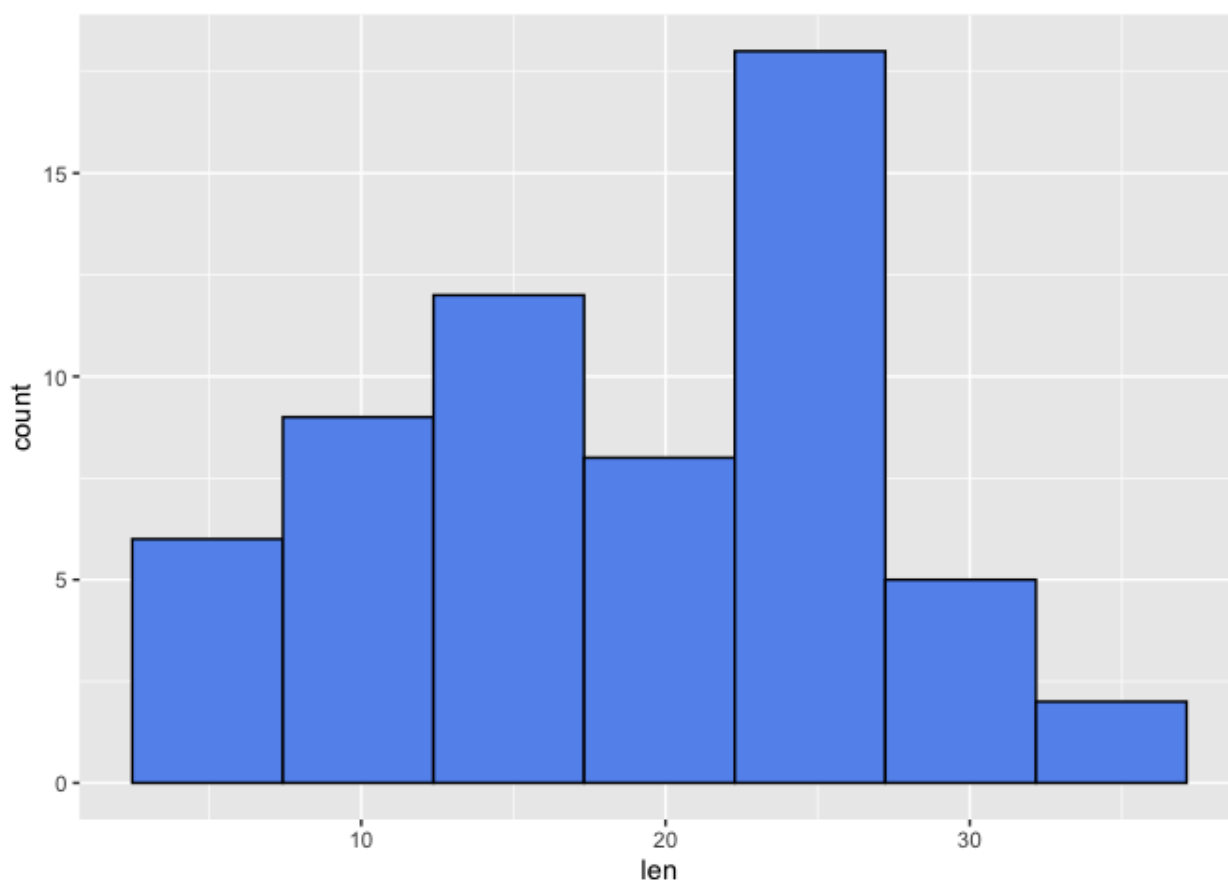
```
ggplot(data=a1, mapping=aes(x=len))+
geom_histogram(color="black", fill="cornflowerblue")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Let's alter the number of bins

```
ggplot(data=a1, mapping=aes(x=len))+  
geom_histogram(color="black", fill="cornflowerblue", bins=7)
```



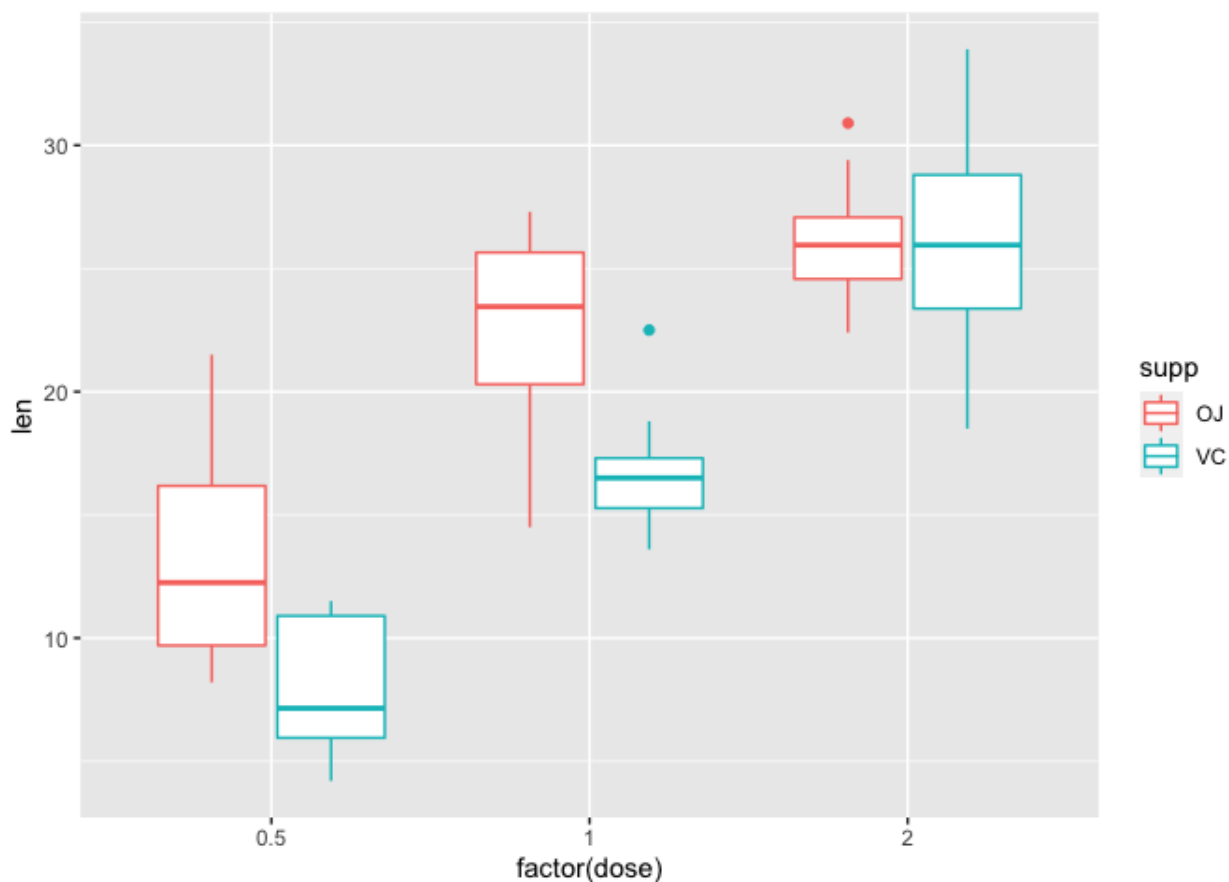
From the above, we see that altering the number of bins alters the binwidth (ie. the range in which occurrence are counted). Thus, altering the bins can influence the distribution that we see. The histograms above seem to be left skewed and a lot of the tooth length values fall between 22.5 and 27.5 when 7 bins were used.

Box plot

Box and whisker plots also show data distribution. Unlike a histogram, we can readily see summary statistics such as median, 25th and 75th percentile, and maximum and minimum.

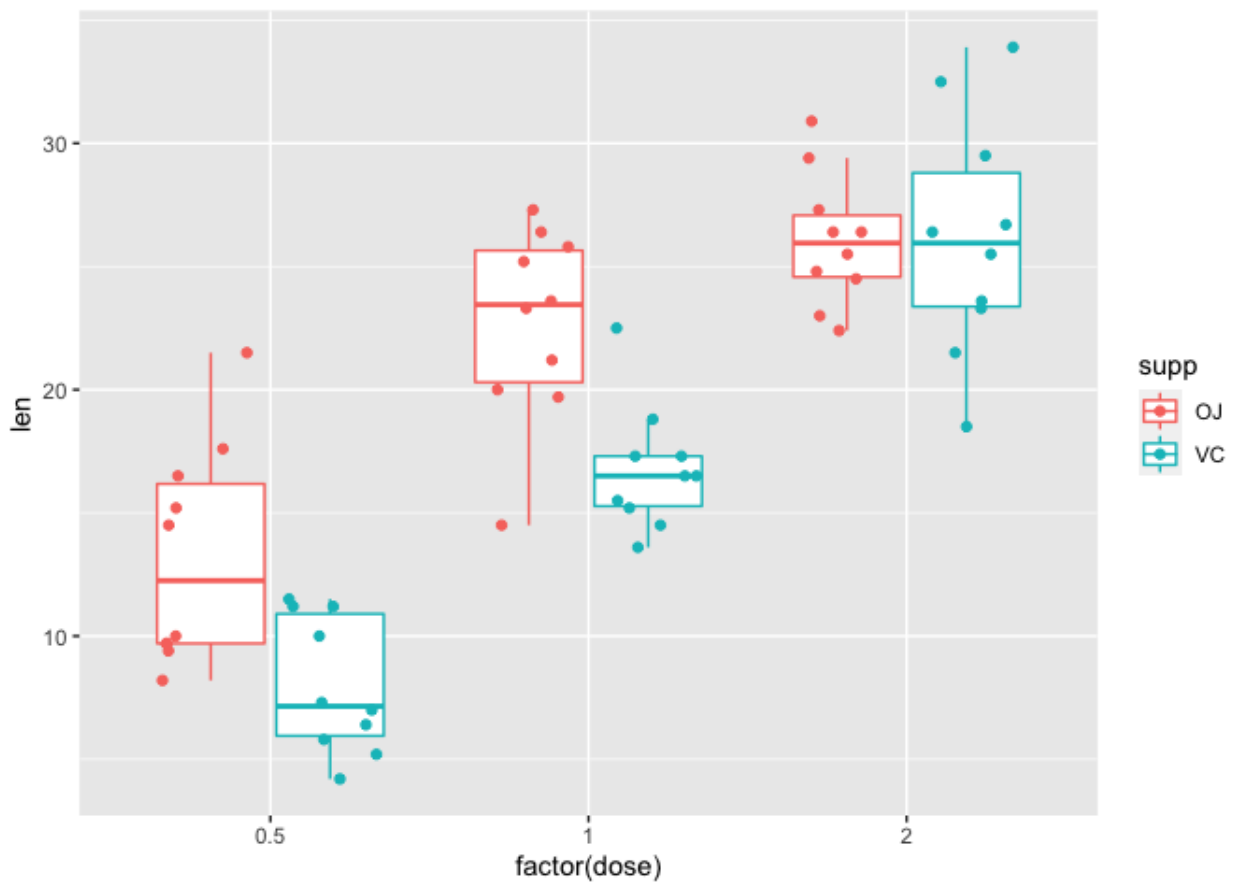
The default statistical transformation of a box plot in ggplot2 is `stat_boxplot` which calculates components of the boxplot. To construct a box plot in ggplot2, we use the `geom_boxplot` argument. Note that within `geom_boxplot` we do not need to explicitly state `stat="boxplot"` because it is default. Below, we have a default boxplot depicting tooth length across the treatment groups. Potential outliers are presented as points.

```
ggplot(data=a1, mapping=aes(x=factor(dose), y=len, color=supp))+  
geom_boxplot()
```



Rather than showing the outliers, we could instead add on `geom_point` to overlay the data points. Here, in `geom_boxplot` we set `outlier.shape=NA` to remove the outliers for the purpose of avoiding duplicating a data point. Within `geom_point` we set the position of the points to `position_jitterdodge` to avoid overlapping of points whose values are close together (set by `jitter.width`) and overlapping of points from measurements derived from different supplements (`dodge.width`).

```
ggplot(data=a1, mapping=aes(x=factor(dose), y=len, color=supp))+
  geom_boxplot(outlier.shape=NA)+
  geom_point(position=position_jitterdodge(jitter.width=0.3,
                                          dodge.width=0.8))
```



The box plot above shows several things

- Tooth length appears to be longer for the OJ treated group at doses of 0.5 and 1
- Tooth length appears to be equal for both the OJ and VC groups at a dose of 2
- At a dose of 0.5 and 1, the median (line inside the box) for the OJ group is larger than the VC group
- At a dose of 0.5 and 1, the interquartile range (IQR) which is defined by the lower (25th percentile) and upper (75th percentile) bounds of the box along the vertical axis is larger for the OJ group as compared to VC - so there is more variability in the OJ group measurements.
- At a dose of 2, the median for both the OJ and VC group are roughly equal
- At a dose of 2, the IQR for the VC group is larger than that for the OJ group

Visualizing clusters with heatmaps

Objectives

1. Introduce the heatmap and dendrogram as tools for visualizing clusters in data.
2. Learn to construct cluster heatmap using the package `pheatmap`.
3. Learn how to save a non-ggplot2 plot.
4. Introduce `ggplotify` to convert non-ggplots to ggplots.
5. Introduce `heatmaply` for constructing interactive heatmaps.

What is a heatmap?

A heatmap is a graphical representation of data where the individual values contained in a matrix are represented as colors. --- [R Graph Gallery \(https://r-graph-gallery.com/heatmap.html\)](https://r-graph-gallery.com/heatmap.html)

Heatmaps are appropriate when we have lots of data because color is easier to interpret and distinguish than raw values. --- [Dundas BI \(https://www.dundas.com/resources/blogs/best-practices/when-and-why-to-use-heat-maps\)](https://www.dundas.com/resources/blogs/best-practices/when-and-why-to-use-heat-maps)

Heatmap can be used to visualize the following

- gene expression across samples (Figure 1)
- correlation (Figure 2)
- disease cases (Figure 3)
- hot/cold zones
- topography

What is a dendrogram?

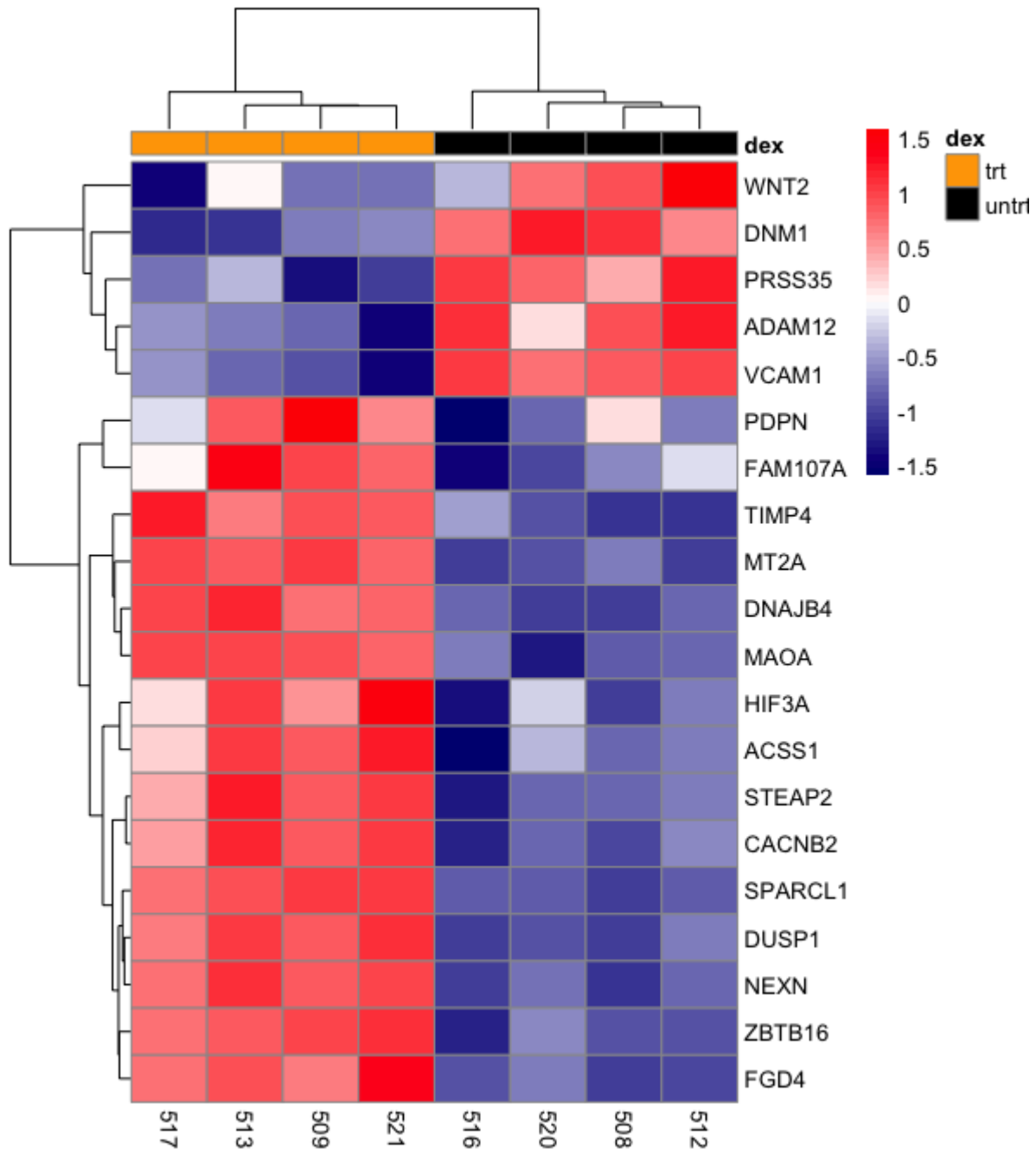
A dendrogram (or tree diagram) is a network structure and can be used to visualize hierarchy or clustering in data. --- [R Graph Gallery \(https://r-graph-gallery.com/dendrogram.html\)](https://r-graph-gallery.com/dendrogram.html)

Applications of dendrograms

Dendrograms are used in phylogenetics to help visualize relatedness of or dissimilarities between species.

In RNA sequencing, dendrogram can be combined with heatmap to show clustering of samples by gene expression or clustering of genes that are similarly expressed (Figure 1).

Figure 1: Heatmap and dendrogram showing clustering of samples with similar gene expression and clustering of genes with similar expression patterns.



Further heatmap and dendrogram can be used as a diagnostic tool in high throughput sequencing experiments. As an example, we can look at the heatmap and dendrogram in Figure 2. In Figure 2, the heatmap shows correlation of RNA sequencing samples with the idea that biological replicates should be more highly correlated compared to samples between treatment groups. The dendrogram clusters similar samples together. Figure 2 tells us that heatmaps can also be used to visualize correlation.

Figure 2: Heatmap and dendrogram showing sample correlation in an RNA sequencing experiment. This heatmap and dendrogram is generated using [DeepTools](https://deeptools.readthedocs.io/en/develop/content/tools/plotCorrelation.html) (<https://deeptools.readthedocs.io/en/develop/content/tools/plotCorrelation.html>).

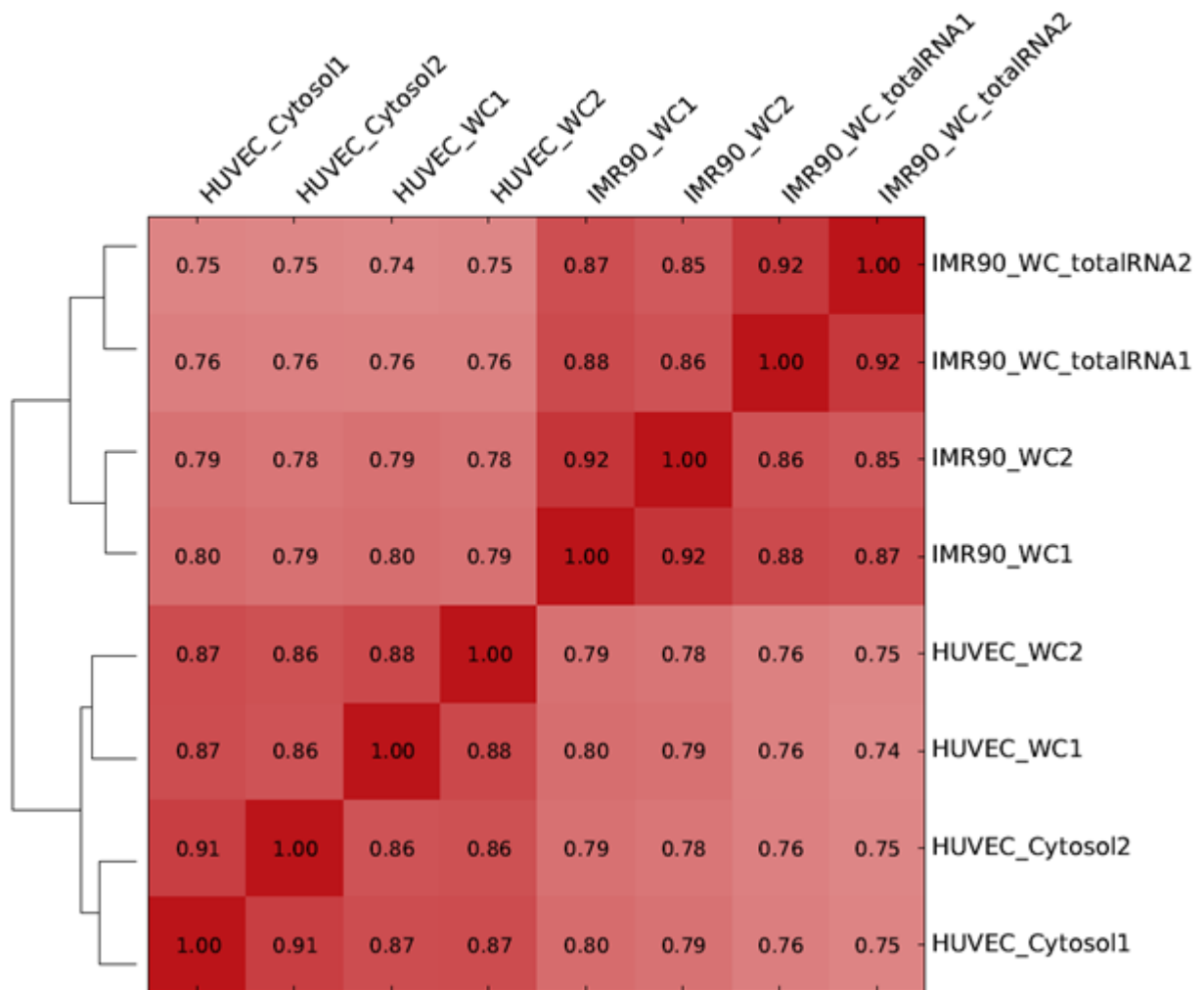
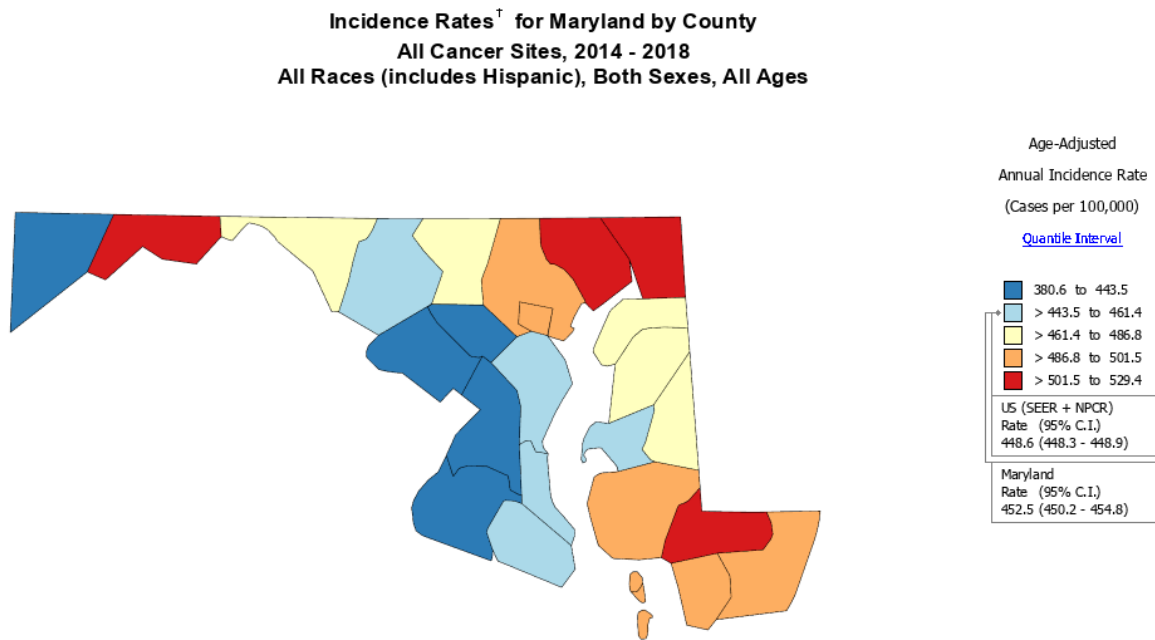


Figure 3: Maryland cancer cases - source: <https://statecancerprofiles.cancer.gov>**Notes:**

[State Cancer Registries](#) may provide more current or more local data.

Data presented on the State Cancer Profiles Web Site may differ from statistics reported by the State Cancer Registries ([for more information](#)).

[†] Incidence rates (cases per 100,000 population per year) are age-adjusted to the [2000 US standard population](#) (19 age groups: <1, 1-4, 5-9, ... , 80-84, 85+). Rates are for invasive cancer only (except for bladder which is invasive and in situ) or unless otherwise specified. Rates calculated using SEER*Stat. Population counts for denominators are based on Census populations as modified by NCI.

The [1969-2018 US Population Data](#) File is used for SEER and NPCR incidence rates.

Rates are computed using cancers classified as malignant based on ICD-O-3. For more information see [malignant.html](#)

Data for the United States does not include data from Puerto Rico

Methods available to produce a heatmap in R

ggplot2

- We would use `geom_tile` to construct the heatmap
- A disadvantage to this approach is that we have to generate the dendrogram separately, then merge and align the dendrogram with the heatmap.

heatmap built into R

- It appears that this by default does not generate a legend showing the correlation between values and color.
- It also appears that assigning distance calculation and clustering methods are not intuitive for the users.

heatmap.2 from the gplots package

- It appears assigning distance calculation and clustering methods are not intuitive for the users.
- [Click here to learn about heatmap.2 \(https://cran.r-project.org/web/packages/gplots/index.html\)](https://cran.r-project.org/web/packages/gplots/index.html)

ComplexHeatmap

- There is no scaling option so the user will have to scale the data separately using `scale`. (see <https://support.bioconductor.org/p/68340/> and <https://github.com/jokergoo/ComplexHeatmap/issues/313> for a discussion on scaling in ComplexHeatmap).
- [Click here to learn about ComplexHeatmap \(https://www.bioconductor.org/packages/release/bioc/html/ComplexHeatmap.html\)](https://www.bioconductor.org/packages/release/bioc/html/ComplexHeatmap.html).

pheatmap

- This is a versatile package that draws clustered heatmaps.
- This package has built-in scaling function, provides ways to incorporate legends, and many features that allows for customization and construction of a publication quality figure.
- [Click here to learn about pheatmap \(https://cran.r-project.org/web/packages/pheatmap/pheatmap.pdf\)](https://cran.r-project.org/web/packages/pheatmap/pheatmap.pdf)

heatmaply

- This package generates interactive heatmaps that allows the user to mouse over a tile to see information such as sample id, gene, and expression value.
- [Click here to learn about heatmaply \(https://cran.r-project.org/web/packages/heatmaply/vignettes/heatmaply.html\)](https://cran.r-project.org/web/packages/heatmaply/vignettes/heatmaply.html)

While most of the tools listed above can be used to produce publication quality heatmaps, we find that pheatmap is perhaps the most comprehensive. Therefore, in this class, we will show how to construct heatmaps using pheatmap. Because heatmaps can be filled with a lot of data, we will also demonstrate the use of heatmaply to construct interactive heatmaps that you could use to explore your data more efficiently.

Load the libraries

```
library(pheatmap) ## for heatmap generation
library(tidyverse) ## for data wrangling
library(ggplotify) ## to convert pheatmap to ggplot2
library(heatmaply) ## for constructing interactive heatmap
```

Import data

The data that we will be working with comes from the airway study that profiled the transcriptome of several airway smooth muscle cell lines under either control or dexamethasone treatment [Himes et al 2014 \(http://www.ncbi.nlm.nih.gov/pubmed/24926665\)](http://www.ncbi.nlm.nih.gov/pubmed/24926665). The dataset is available from Bioconductor (<https://bioconductor.org/packages/release/data/experiment/html/airway.html>).

Specifically, our dataset represents the normalized (log₂ counts per million or log₂ CPM) of count values from the top 20 differential expressed genes. This data is saved as the comma separated file `RNAseq_mat_top20.csv` and thus we will be using the `read.csv` command to import.

As a refresher, inside the `read.csv` command we have the following arguments

- `"/data/RNAseq_mat_top20.csv"` is the file path
- `header=TRUE` indicates that we have column headings
- we set the first column of our dataset, which contains gene names as the row names in the imported data using `row.names=1`, where 1 indicates the column number that we want to import as row names
- we set `check.names=FALSE` so that R leaves the column headings alone

The data is assigned to R object `mat`.

```
mat<-read.csv("/data/RNAseq_mat_top20.csv",header=TRUE,row.names=1,
              check.names=FALSE)
```

We will now use `head` to look at the first 6 rows of `mat`. The column headings represent sample names and the row names are the genes.

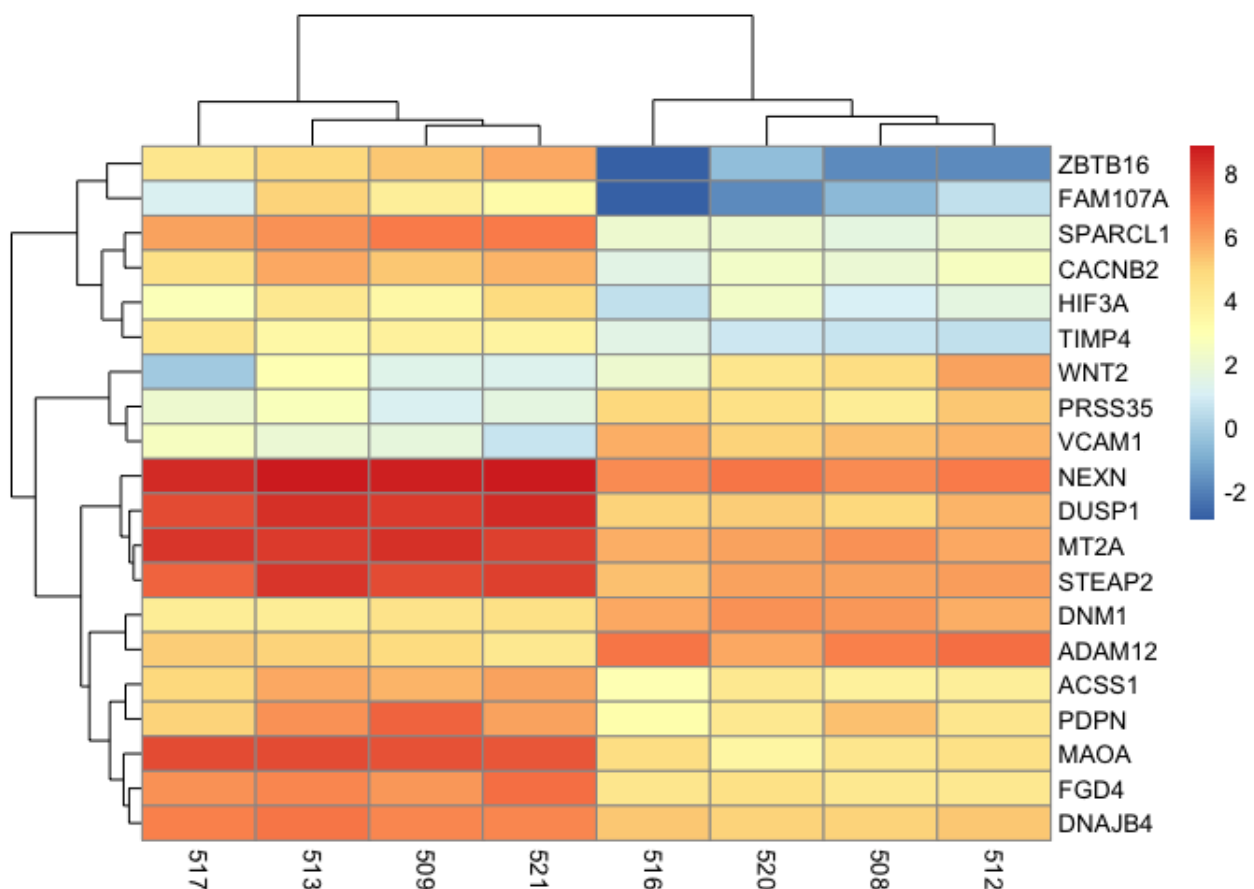
```
head(mat)
```

```
##           508      509      512      513      516      517
## WNT2      4.694554 1.332858  5.983720 2.898648  2.1105784 -0.1655252
## DNM1      6.180735 4.441965  5.660024 3.981513  5.8002923  3.9603751
## ZBTB16   -1.863523 5.257967 -1.777152 4.902223 -2.9319722  4.2830866
## DUSP1     4.936551 8.019074  5.607568 8.302196  5.0283417  7.7229898
## HIF3A     1.013991 3.374457  1.575250 4.154740  0.4928878  2.8335879
## MT2A      6.248514 8.276988  5.782855 8.070846  5.7583897  8.2015851
##
##           521
## WNT2      1.237000
## DNM1      4.488955
## ZBTB16    5.779173
## DUSP1     8.396709
## HIF3A     4.790581
## MT2A      7.933492
```

Heatmap of the top 20 genes from differential expression analysis

Below we generate a basic heatmap using the pheatmap package. We use the pheatmap command and include the data that we want to construct a heatmap of as the argument. In the heatmap below, we have the sample IDs plotted along the bottom horizontal axis, while the genes names are presented long the right vertical axis. Each tile in the heatmap corresponds to the expression of a gene for the corresponding sample and as mentioned earlier, the expression level is indicated by the color scale. Note that later on in this lesson, we can add an additional legend that color codes the treatment group that each sample belongs to. The dendrogram that spans the columns indicates how samples are clustered together, while that which spans the rows tells us how the genes are clustered together.

```
pheatmap(mat)
```



Distance, clustering, and scaling

When generating a heatmap and dendrogram to show clustering three parameters to consider are

- method for calculating distance between objects (ie. experimental samples)
- method for clustering
- scaling of data prior to inputting into heatmap generating algorithm

Distance calculation

The idea behind cluster analysis is to calculate some sort of distance between objects in order to identify the ones that are closer together. When two objects have a small distance, we can conclude they are closer and should cluster together. On the other hand, two objects that are further apart will have a larger distance. There are various approaches to calculating distance in cluster analysis so considerations should be taken for choosing the appropriate one. To learn more about distance calculation methods as well as advantages and disadvantages of each see [Shirkhorshidi et al, PLOS ONE, 2015 \(https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0144059\)](https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0144059). In heatmap, we can specify the clustering distance using either the `clustering_distance_rows` argument or `clustering_distance_cols` depending on whether we would like to cluster by row or column variables.

Cluster generation

After the distance matrix has been calculated, it is time to perform the actual clustering and again, various approaches can be used to generate clusters. The following resources are good for learning about the various hierarchical clustering methods. In heatmap, the clustering method is specified by the `clustering_method` argument.

- https://hlab.stanford.edu/brian/forming_clusters.htm (https://hlab.stanford.edu/brian/forming_clusters.htm)
- <https://dataaspirant.com/hierarchical-clustering-algorithm/> (<https://dataaspirant.com/hierarchical-clustering-algorithm/>)
- <https://www.learn datasci.com/glossary/hierarchical-clustering/> (<https://www.learn datasci.com/glossary/hierarchical-clustering/>)

Scaling

Prior to sending our data into the heatmap generating algorithm, it is a good idea to scale. There are several reasons for doing this

- Variables in the data might not have the same units, thus without scaling we will be, to borrow a phrase, comparing apples to oranges

- Scaling allows us to discern patterns in variables with low values when plotting on the color scale. Without scaling, variables with large values will drown out the signal from those with low values. We will see an example of this using the mtcars data.
- Scaling also prevents variables with large values from contributing too much weight to distance <https://medium.com/analytics-vidhya/why-is-scaling-required-in-knn-and-k-means-8129e4d88ed7> (<https://medium.com/analytics-vidhya/why-is-scaling-required-in-knn-and-k-means-8129e4d88ed7>). Without scaling, it will hard to discern whether variables with lower values contribute to separation.

A common method for scaling is to use the z score (see z score formula), which tells us how many standard deviations away from the mean is a given value in our data. This is the scaling method for pheatmap.

$$z \text{ score} = (\text{individual value} - \text{mean}) / (\text{standard deviation})$$

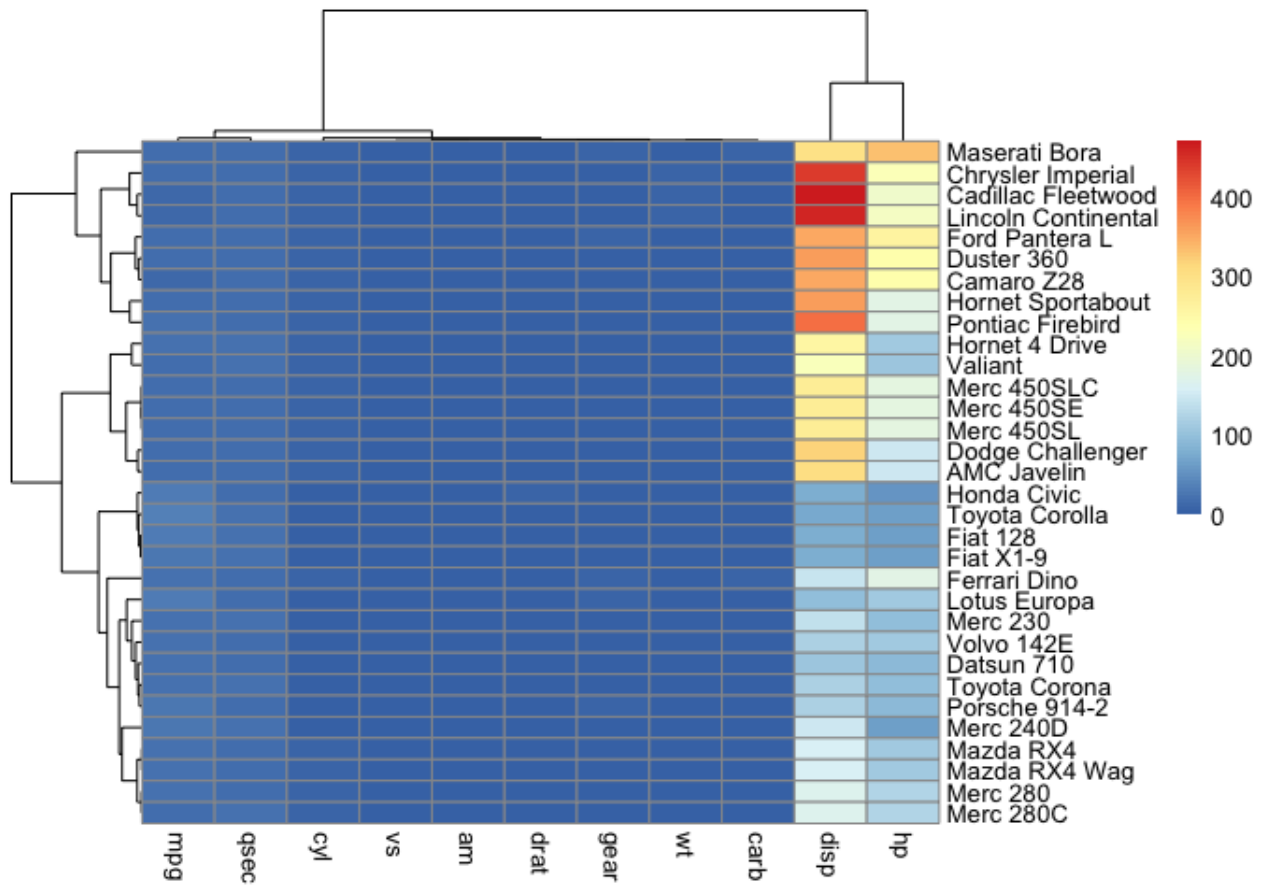
Below, we will use the mtcars data to look at how scaling influences a heatmap. Because mtcars is built into R, we can use the `data` command to load it and we will save this as an object named `cars`. Next, we will use the `head` command to view the first few rows of the cars data.

```
data(mtcars)
cars <- mtcars
head(cars)
```

```
##           mpg cyl  disp  hp  drat    wt  qsec vs  am  gear  cā
## Mazda RX4      21.0   6  160  110  3.90  2.620  16.46  0   1    4
## Mazda RX4 Wag  21.0   6  160  110  3.90  2.875  17.02  0   1    4
## Datsun 710     22.8   4  108   93  3.85  2.320  18.61  1   1    4
## Hornet 4 Drive  21.4   6  258  110  3.08  3.215  19.44  1   0    3
## Hornet Sportabout 18.7   8  360  175  3.15  3.440  17.02  0   0    3
## Valiant        18.1   6  225  105  2.76  3.460  20.22  1   0    3
```

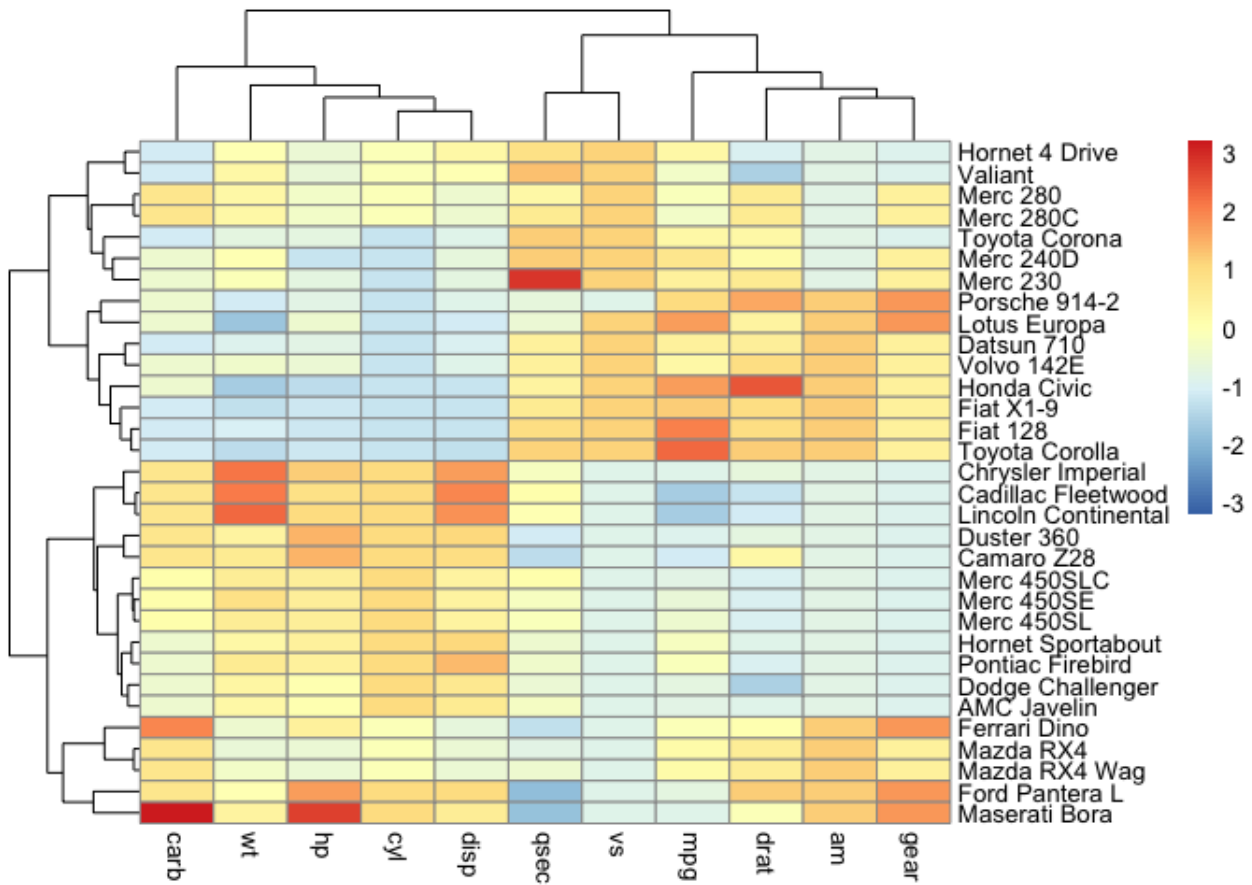
Note that variables like `disp` and `hp` has larger magnitudes as compared to one like `mpg`. Also, the variables in this data does not have the same units. If we constructed a heatmap of the mtcars data without scaling, we will not be able to discern patterns in variables like `mpg` among the samples. This is because the values for `mpg` are small in comparison to those for `disp` and `hp`, they get squeeze towards the bottom of color scale.

```
pheatmap(cars)
```



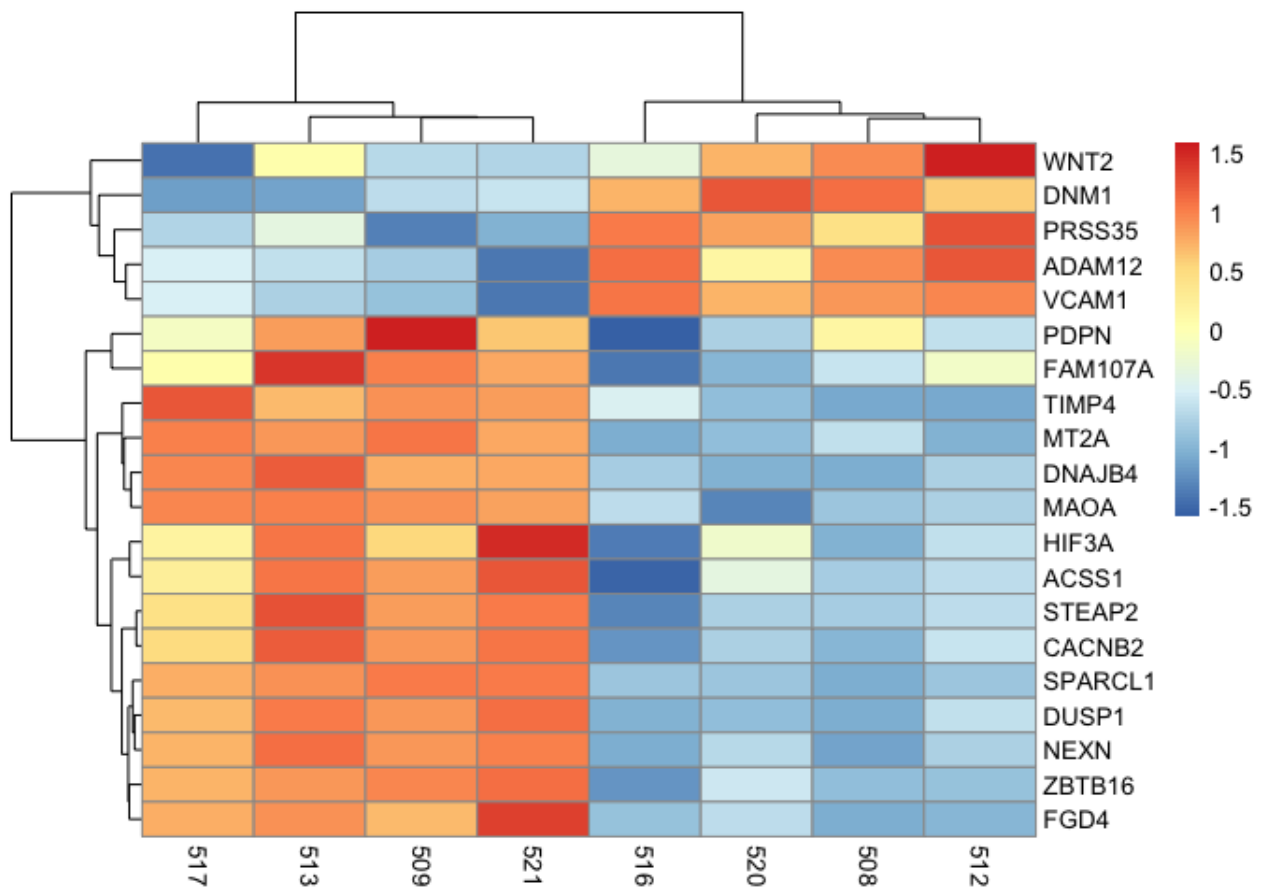
However, if we scaled then it becomes easier to observe differences in values for each of the variables. We are interested in the differences in each variable across the car types, thus, we scale by column because the sample names (ie. car brands) are listed down the rows. If the car brands were listed across columns, we would have scaled by row.

```
ph heatmap(cars, scale="column")
```

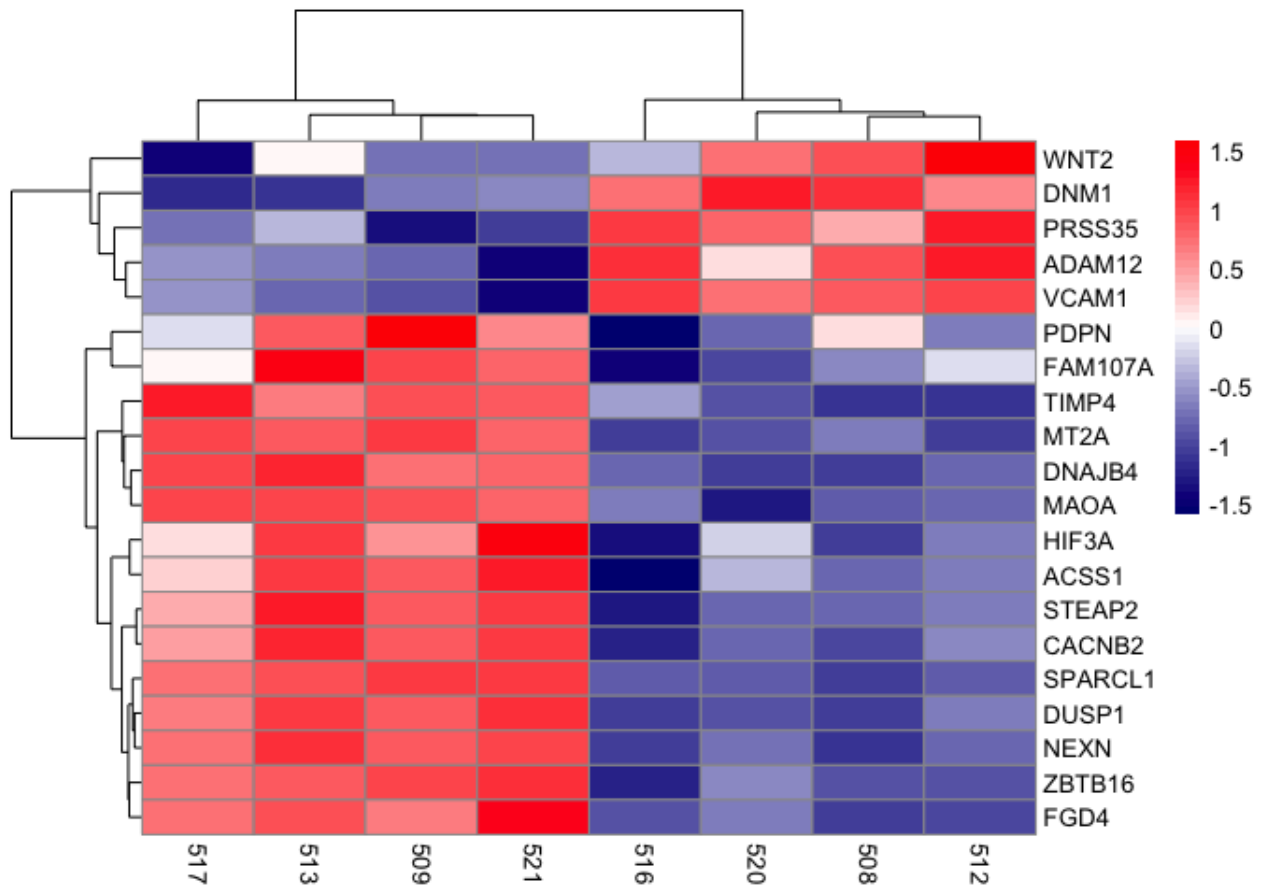
Getting back to RNA seq and something more biologically relevant, we can scale by row in mat.

```
pheatmap(mat, scale="row")
```



Working with color customization

```
pheatmap(mat, scale="row",
          color=colorRampPalette(c("navy", "white", "red"))(50))
```



Adding treatment group information to samples (`annotation_col`, `annotation_colors`)

The code below generates a data frame, `dfh`, that contains information on the treatment group in which a sample was assigned. To create a data frame in R, we use the `data.frame` command. In the `data.frame` command, we set the sample to the column names of the data mat using `colnames(mat)` to extract the mat column headings. We then want to convert the column headings in mat to character using `as.character`. Next, also in the `data.frame` command, we set the column `dex` to "Treatment".

Using `%>%`, we pass the `dfh` data frame to the `column_to_rownames` command to set the rownames of the `dfh` data frame to the sample IDs. Finally, we will change the values in the `dex` column to either `untrt` (untreated) or `trt` (treated) using `ifelse` to check if the row names of `dfh` (`rownames(dfh)`) are samples 508, 512, 516, or 520. If yes, then the value for these samples in the `dex` column becomes `untrt`. In other words, 508, 512, 516, and 520 are untreated samples. Else, we will assign the samples to the `trt` group, which indicates they are treated.

```
#create data frame for annotations
dfh<-data.frame(sample=as.character(colnames(mat)),dex="Treatment")%>%
  column_to_rownames("sample")
dfh$dex<-ifelse(rownames(dfh) %in% c("508","512","516","520"),
```

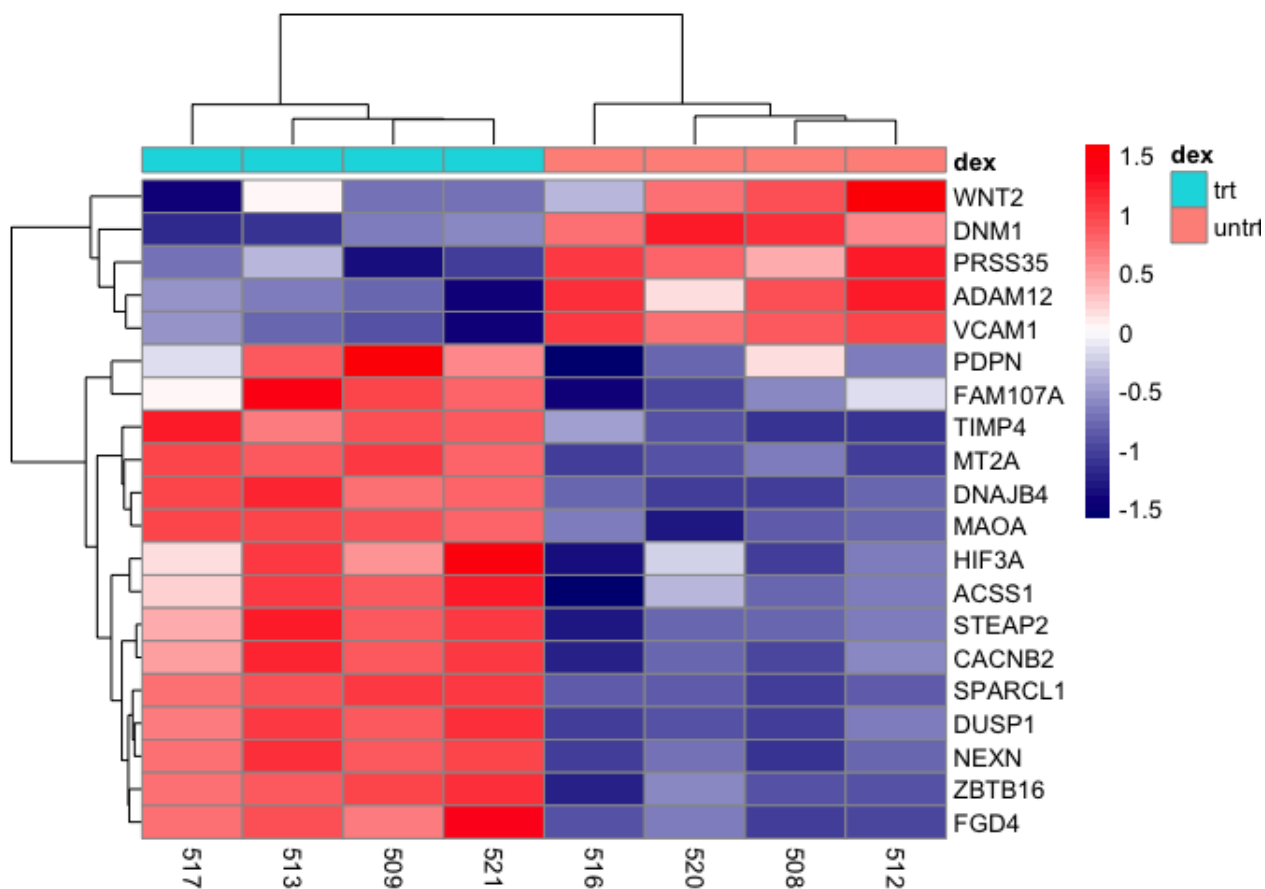
```
"untrt","trt")  
dfh
```

```
##      dex  
## 508 untrt  
## 509  trt  
## 512 untrt  
## 513  trt  
## 516 untrt  
## 517  trt  
## 520 untrt  
## 521  trt
```

Add the annotations column

We can add the sample treatment annotation by setting the `annotation_col` argument to `dfh` in `pheatmap`. We use `annotation_col` rather than `annotation_row` because the samples IDs are listed along the horizontal axis so essentially corresponding to the columns of the heatmap. The result is that the samples are now color coded by the treatment group in which they belong and this color coding is provided in the legend.

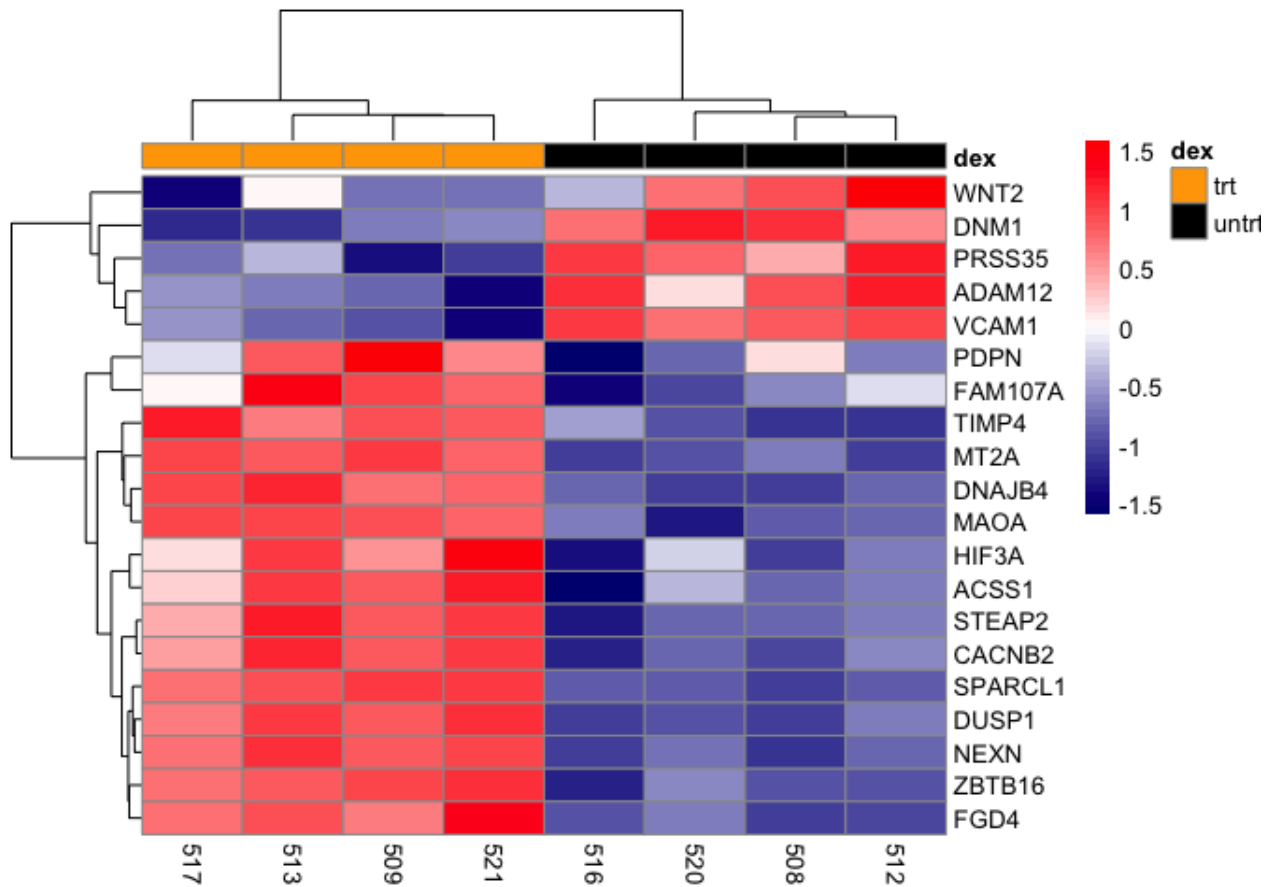
```
pheatmap(mat,scale="row", annotation_col = dfh,  
         color=colorRampPalette(c("navy", "white", "red"))(50))
```



Modify the annotations colors

Using the `annotation_colors` argument, we can reassign the colors of the sample to treatment mapping legend.

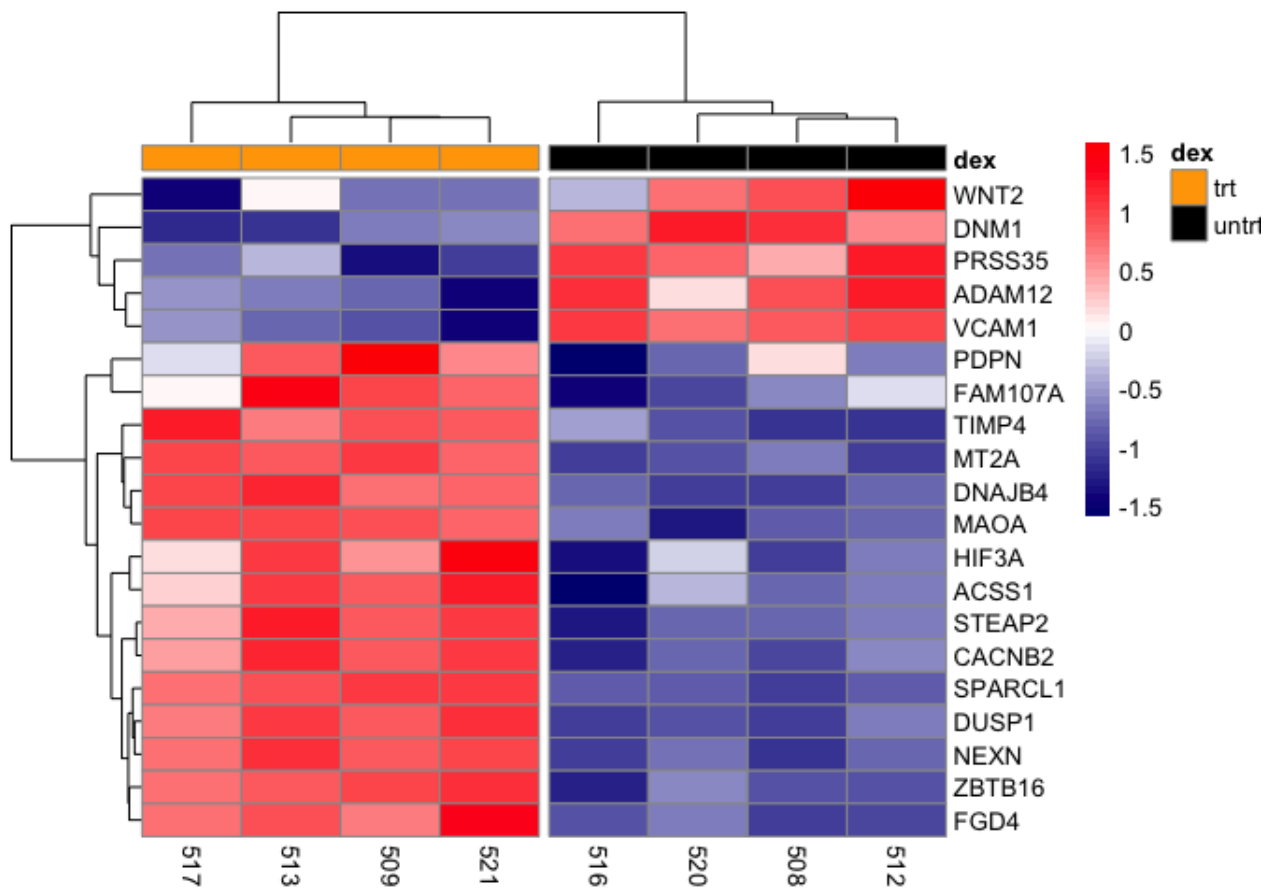
```
pheatmap(mat,scale="row", annotation_col = dfh,
          annotation_colors=list(dex=c(trt="orange",untrt="black")),
          color=colorRampPalette(c("navy", "white", "red"))(50))
```



Splitting heatmap into multiple columns and rows

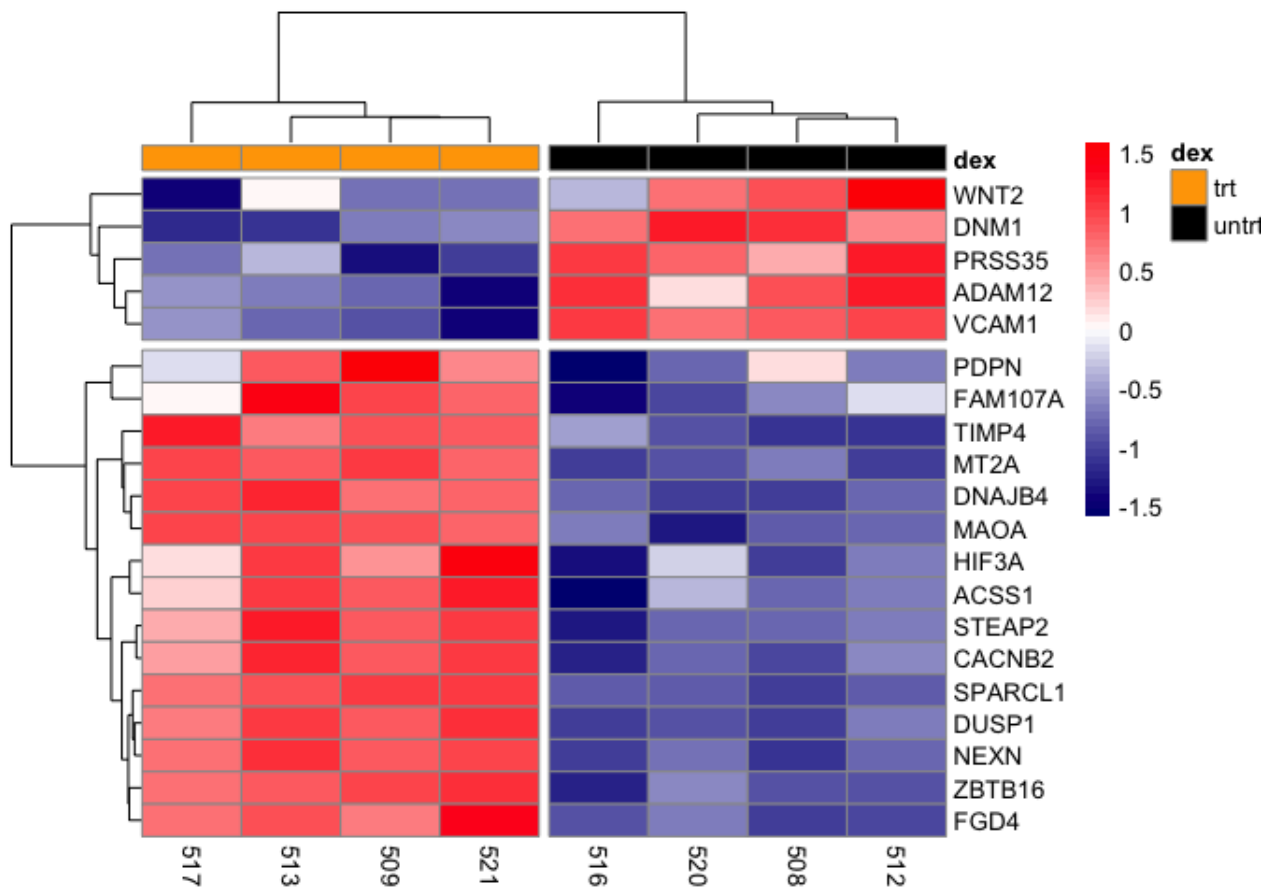
It may be helpful to split the heatmap into different portions to illustrate clusters more efficiently. Here, we can split the heatmap into two columns using the argument `cutree_col` and setting this to 2. Doing so will split the heatmap into a column containing the dexamethasone (dex) treated samples (trt) and untreated samples (untrt).

```
pheatmap(mat,scale="row", annotation_col = dfh,
          annotation_colors=list(dex=c(trt="orange",untrt="black")),
          color=colorRampPalette(c("navy", "white", "red"))(50),
          cutree_cols=2)
```



If we include `cutree_rows=2`, then the heatmap will be split into two rows. Note that it is split in a way that the top row represents genes that are down-regulated in the treated group and up-regulated in the untreated group. The bottom row represents those genes that are up-regulated by dexamethasone treatment but down-regulated when not treated.

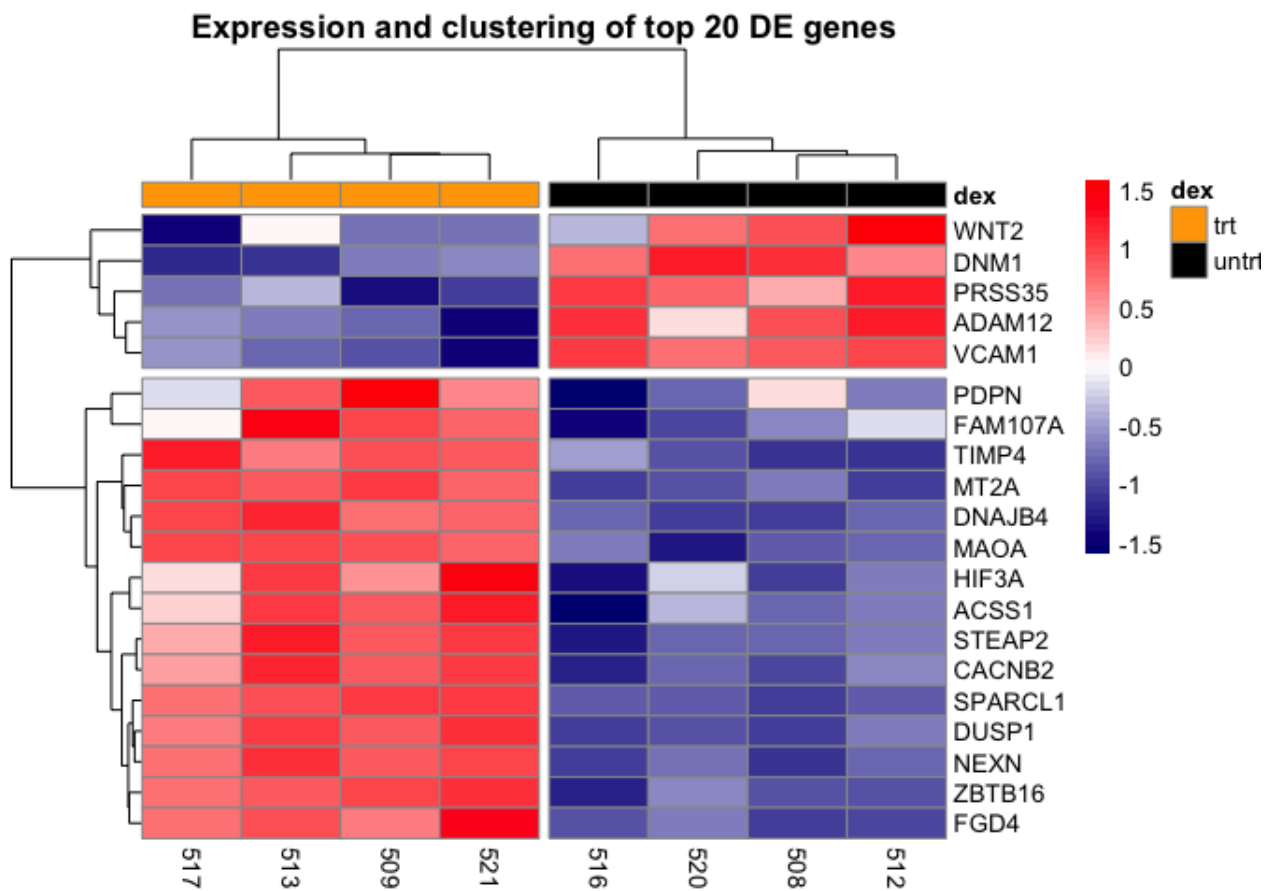
```
pheatmap(mat,scale="row", annotation_col = dfh,
          annotation_colors =list(dex=c(trt="orange",untrt="black")),
          color=colorRampPalette(c("navy", "white", "red"))(50),
          cutree_cols=2, cutree_rows=2)
```



Add a title to heatmap

A title for our heatmap can be included using the `main` argument

```
heatmap(mat,scale="row", annotation_col = dfh,
        annotation_colors =list(dex=c(trt="orange",untrl="black")),
        color=colorRampPalette(c("navy", "white", "red"))(50),
        cutree_cols=2, cutree_rows=2,
        main="Expression and clustering of top 20 DE genes")
```

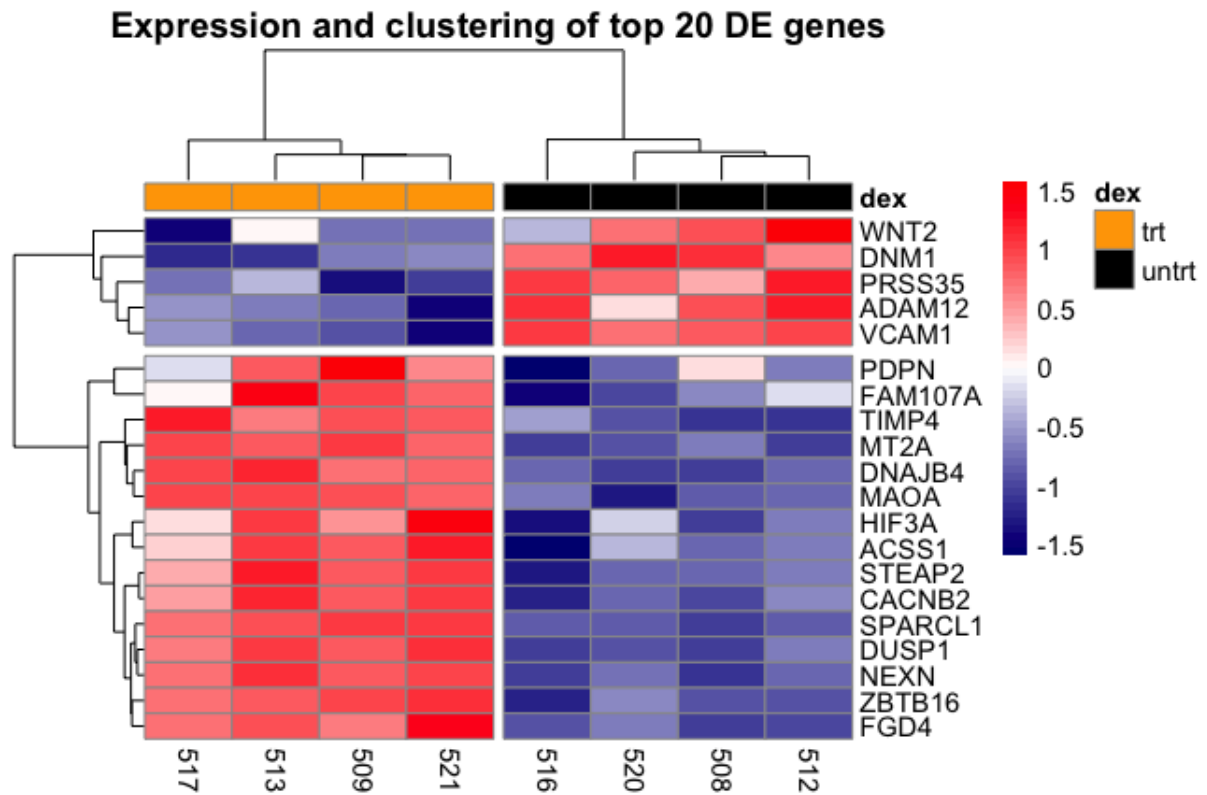



The last few customizations we will do with the heatmap is to adjust the font size using argument `fontsize`. We will adjust the `cellwidth` to move the treatment legend into the plot canvas. We also use `cellheight` to adjust the height of the heatmap to fill more of the plotting canvas.

```

heatmap(mat, scale="row", annotation_col = dfh,
        annotation_colors=list(dex=c(trt="orange", untrt="black")),
        color=colorRampPalette(c("navy", "white", "red"))(50),
        cutree_cols=2, cutree_rows=2,
        main="Expression and clustering of top 20 DE genes",
        fontsize=11, cellwidth=35, cellheight=10.25)

```



Saving a non-ggplot

Recall that we can use the `ggsave` command to save a ggplot. However, heatmaps generated using the `pheatmap` package are not ggplots, therefore we need to turn to either the image export feature (Figure 7) in R studio or use one of the several image saving commands.



The programmatic ways to save an image include the following commands

- jpeg
- bmp
- tiff
- png
- pdf

All of these take the file name in which we would like to save the image, resolution (*res*), image width (*width*), image height (*height*), and units of image dimension (*unit*) as arguments. Below, we use `png` to save our heatmap as file `pheatmap_1.png` at 300 dpi as specified in *res*. The workflow is to first create the file using one of the image save commands, then generate the plot, and set `dev.off()` to turn off the current graphical device. If we do not set `dev.off()`, subsequent plots will overwrite the file that we just saved and will not show up in the plot pane.

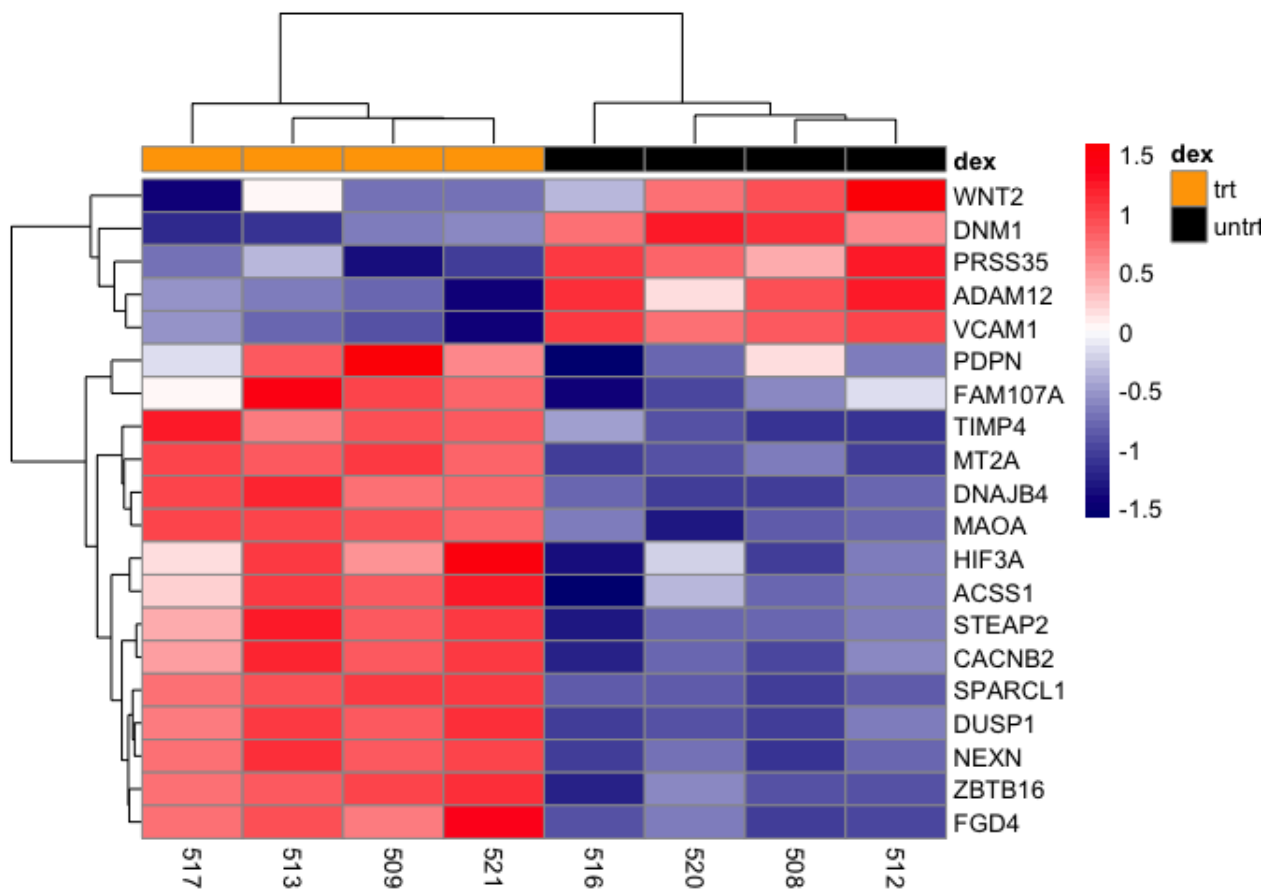
```
dev.new()
png("./data/pheatmap_1.png", res=300, width=7, height=4.5, unit="in")
pheatmap(mat,scale="row", annotation_col = dfh,
         annotation_colors=list(dex=c(trt="orange",untrt="black")),
         color=colorRampPalette(c("navy", "white", "red"))(50),
         cutree_cols=2, cutree_rows=2,
         main="Expression and clustering of top DE genes",
         fontsize=11, cellwidth=35, cellheight=10.25)
```

```
dev.off()
```

```
## quartz_off_screen
##                               3
```

Make this plot compatible with ggplot2 using the package `ggplotify`.

```
hm_gg<-as.ggplot(pheatmap(mat,scale="row", annotation_col=
  dfh,annotation_colors =list(dex=c(trt="orange",
  untrt="black")),color=colorRampPalette(c("navy",
  "white", "red"))(50)))
```



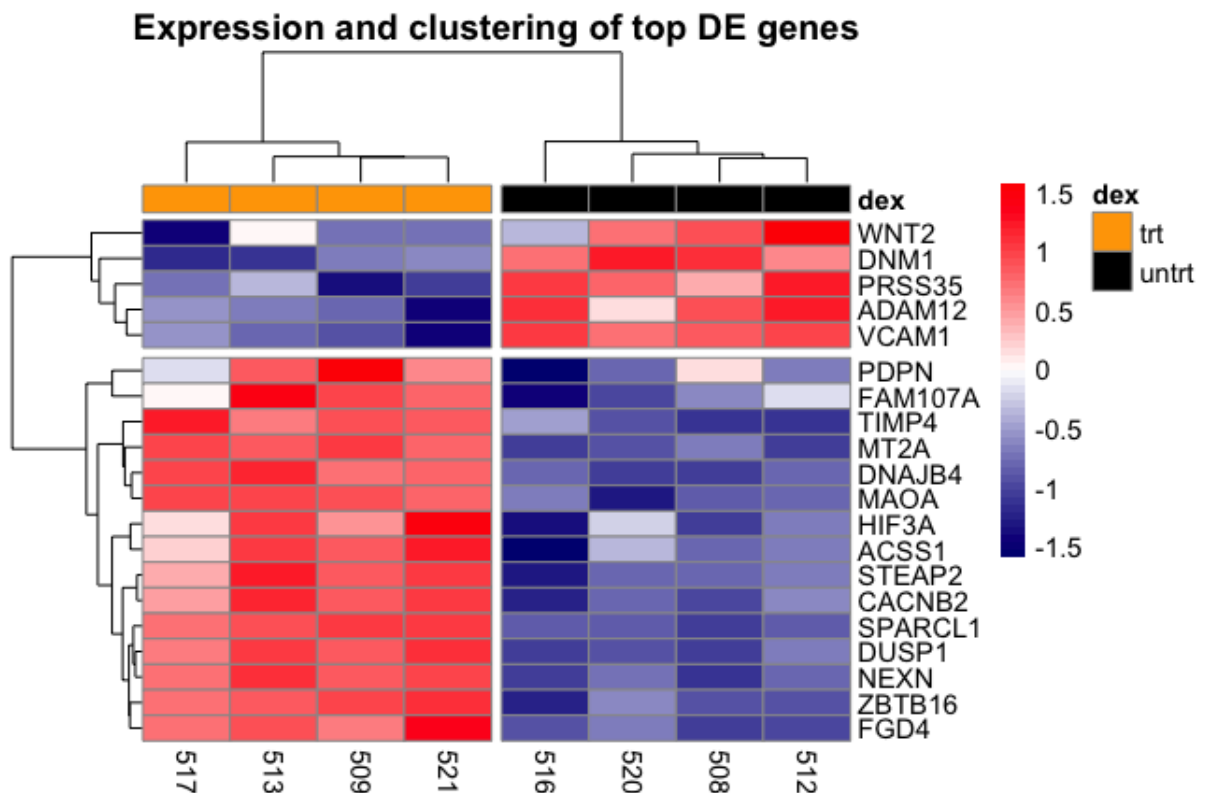
Save as an R object

Below, we assign the heatmap to the R object `hm_ph` and we can import this back to R in the future.

```

hm_ph <- pheatmap(mat,scale="row", annotation_col =
  dfh,annotation_colors =list(dex=c(trt="orange",
  untrt="black")), color=colorRampPalette(c("navy",
  "white", "red"))(50),cutree_cols=2, cutree_rows=2,
  main="Expression and clustering of top DE genes",
  fontsize=11, cellwidth=35, cellheight=10.25)

```



```

saveRDS(hm_ph, file="./data/airways_pheatmap.rds")

```

We will wrap up lesson 5 with an introduction to the package `heatmapply`, which can be used to generate interactive heatmaps. Below is a basic interactive heatmap generated using this package for the airway top 20 differentially expressed genes (`mat`).

Similar to `pheatmap`, we will start by providing `heatmapply` the data that we would like to plot. We can also scale by row like we did in `pheatmap`. Plot margins can also be set to ensure the entire plot fits on the canvas. Like in `pheatmap`, we assign the plot title using the argument `main`. Setting `col_side_colors` to the data frame `dfh`, which contains the sample to treatment mapping, creates a legend that spans the columns of the heatmap, informing us of the treatment group to which the samples belong.

```

heatmapply(mat, scale="row", margins=c(0.5,1,50,1),

```

```
main="Interactively explore airway DE genes",  
col_side_colors=dfh)
```

[heatmaply example](#)

Lesson 6: Multi-figure panel

Objectives

1. Combine multiple plots into a single figure
2. Learn how to use aspects of `cowplot` and `patchwork`

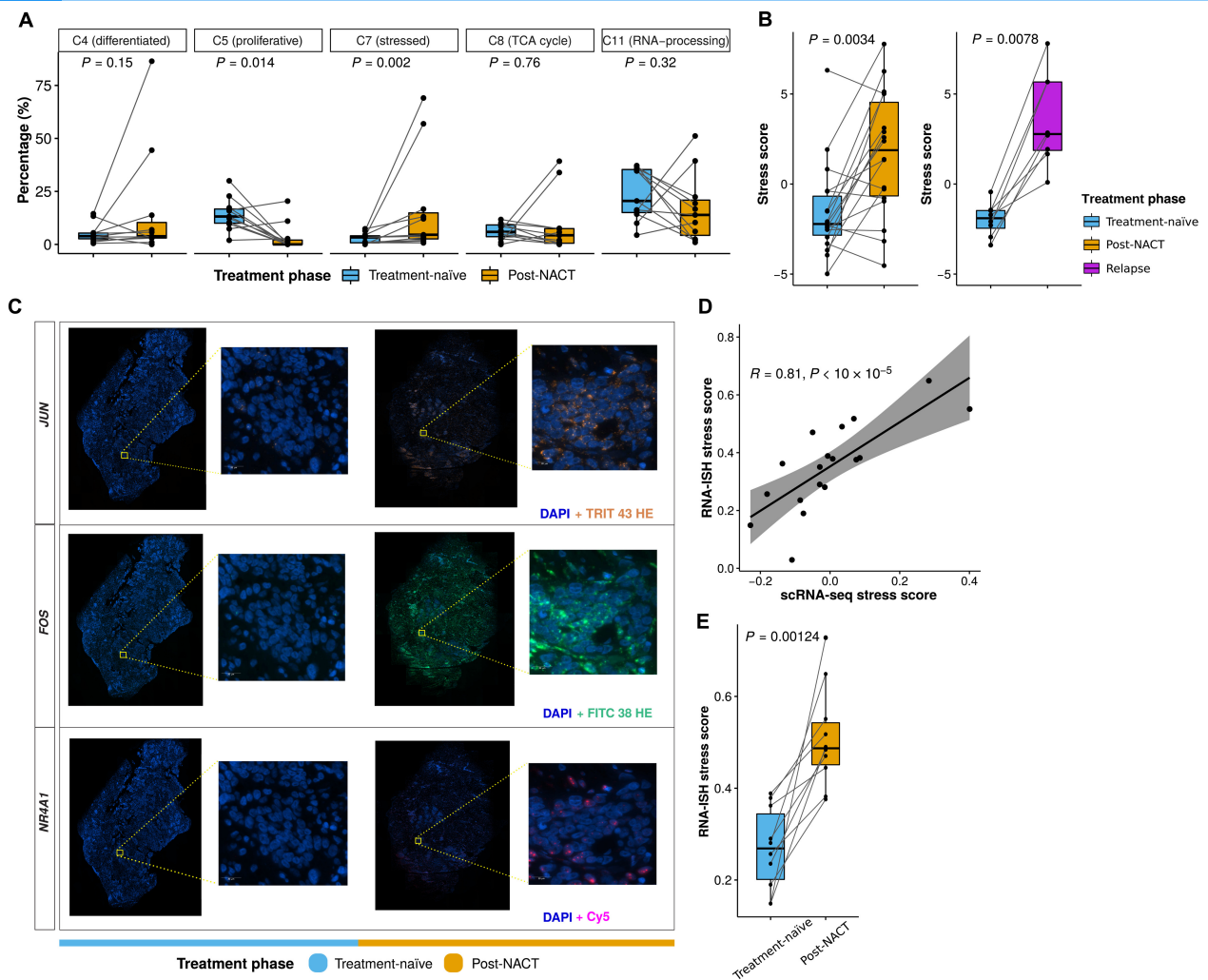
The primary purpose of this lesson is to learn how to combine multiple figures into a single multi-panel figure using `patchwork` and a few features of `cowplot`. While we will learn how to customize and arrange plots in a multi-figure panel, this is not a comprehensive lesson on all aspects of `patchwork` and `cowplot`. If you have something specific in mind for your own data, I implore you to read the documentation for these packages to understand their full potential for customization.

Why do we need to learn to combine figures?

Combining multiple figures is advantageous when preparing results for conference presentations (via poster) or publication. Most journals place limits on the number of figures permitted per publication.

Example journals and their figure limits:

Journal	Impact Factor	Number of Figures
<i>Nature Cancer</i>	60.72	5-8
<i>Science</i>	47.73	6
<i>Cancer Cell</i>	31.74	8
<i>Journal of Clinical Oncology</i>	44.54	6
<i>JAMA Oncology</i>	31.78	5
<i>Cell Host and Microbe</i>	21.02	7



Example Multi-figure panel from Zhang et al.(2022). Longitudinal single-cell RNA-seq analysis reveals stress-promoted chemoresistance in metastatic ovarian cancer. *Science advances*, 8(8), eabm1831.

Load the libraries

There are multiple ways to combine figures using R. In this lesson, we will learn how to combine figures primarily with `patchwork`. Though, we will also learn some components of `cowplot`.

To get started, load the libraries. All packages used today can be installed from CRAN using `install.packages()`.

```
#Get patchwork
#To install use install.packages('patchwork')
#load
library(patchwork)

#Get cowplot
#To install use install.packages('cowplot')
```



```
#load
library(cowplot)

#We will also use ggplot2
library(ggplot2)
```

The Data

This is the sixth lesson in our Data Visualization with R Series. At this point, we have created quite a few plots. For this lesson, we will focus on the RNA-Seq plots that we created in previous lessons. We will also include other related plots that were created using the same RNA-Seq data, but were not created throughout this course series. All plots were saved as R objects (.rds). To load the data into R, we will need to use the `readRDS()` function.

Let's load and view our plots. To view our plots, we can simply call the objects by name.

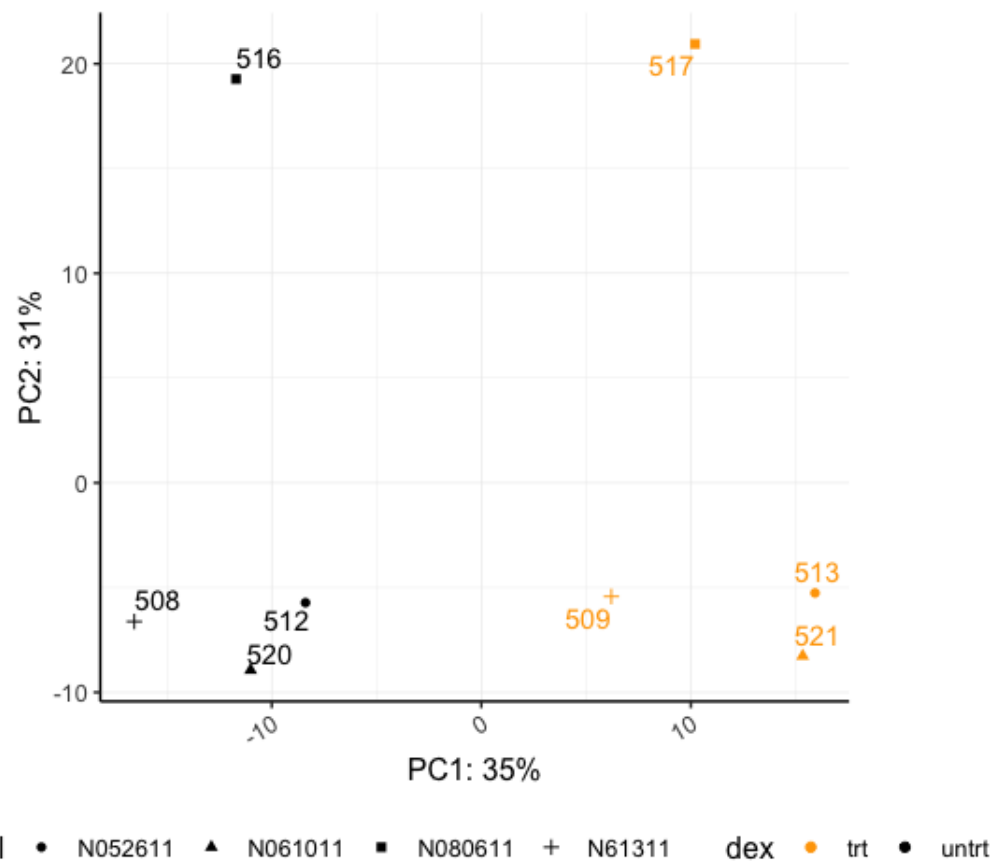
```
pca<-readRDS("./data/airwaypca.rds")

volcano<-readRDS("./data/volcanoplot.rds")

hmap<-readRDS("./data/airwayhm.rds")

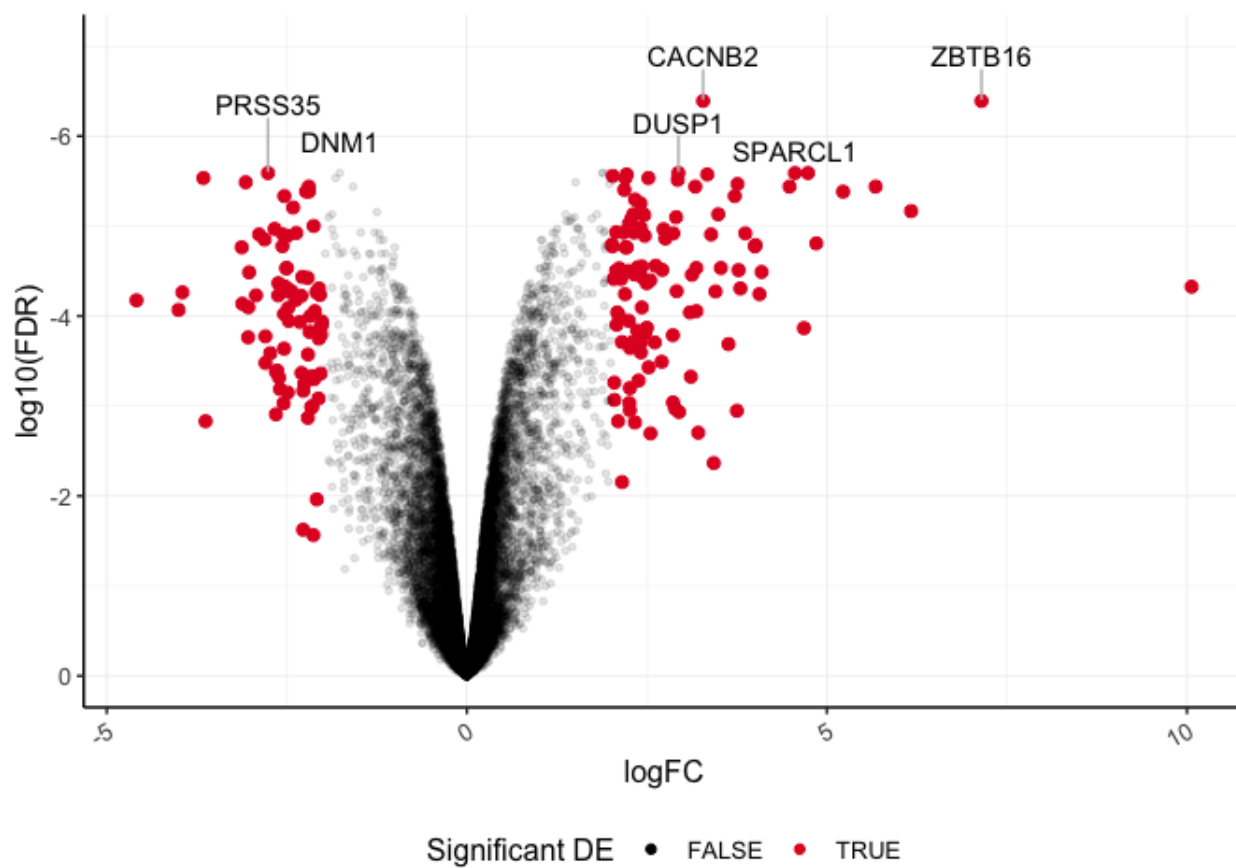
sc<-readRDS("./data/stripchart.rds")
```

```
#view objects
pca
```

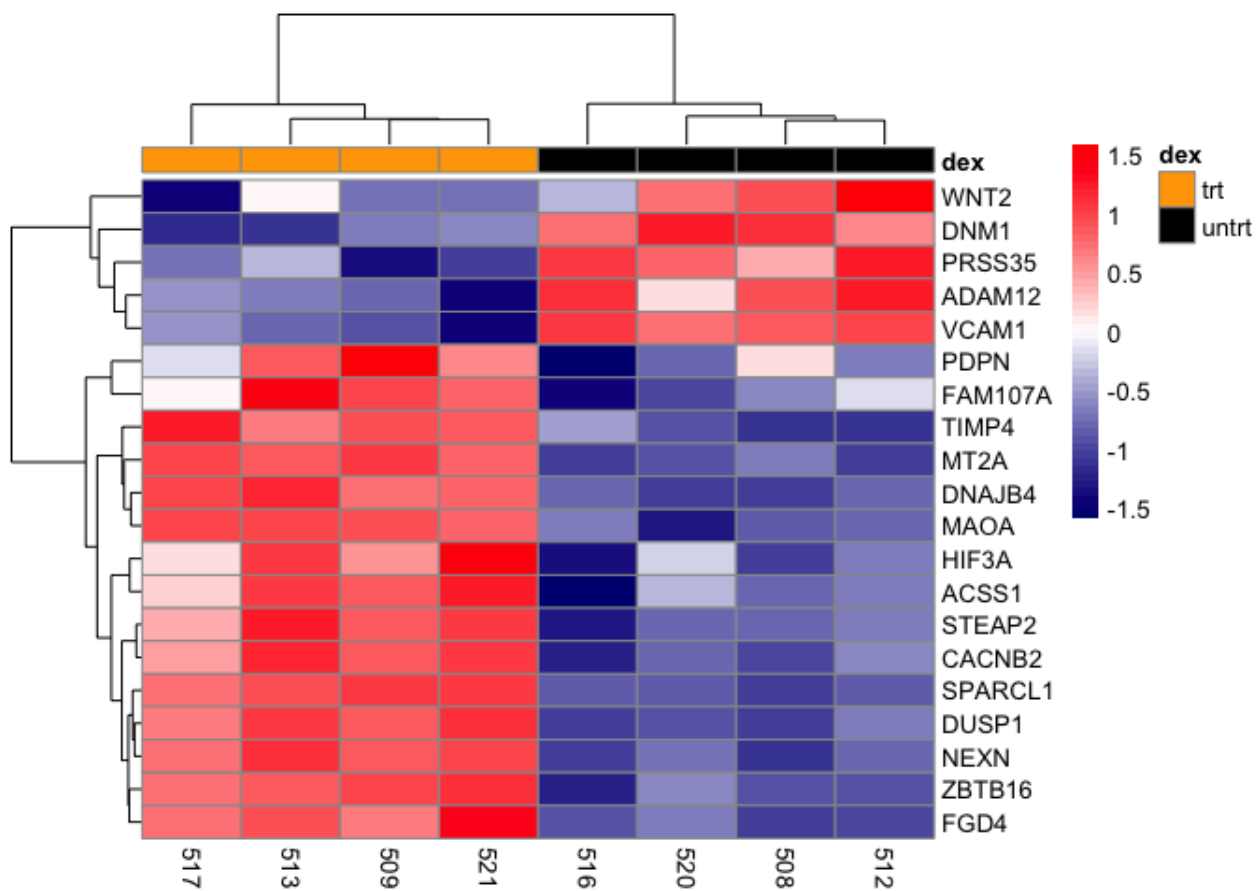


```
volcano
```

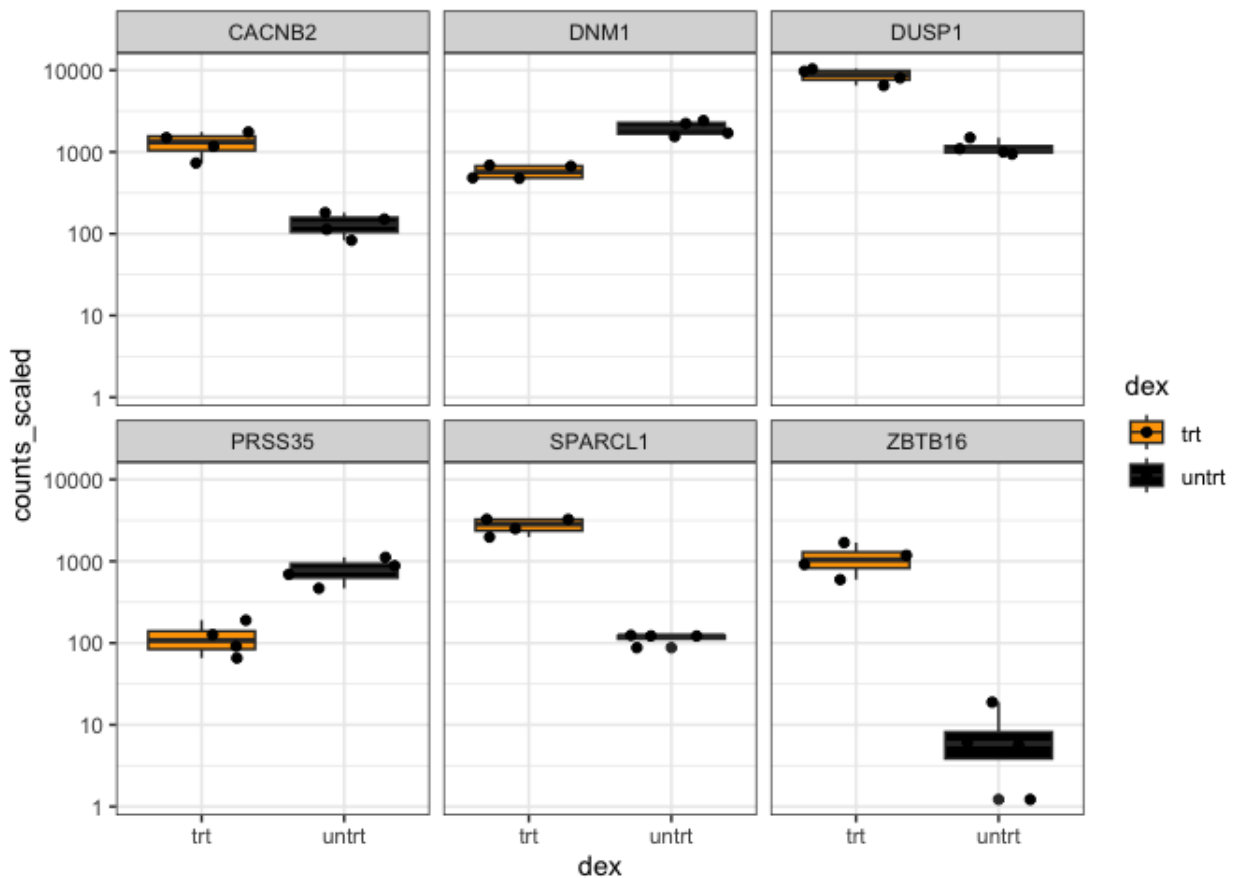
```
## Warning: Using alpha for a discrete variable is not advised.
```



hmap



SC



What is cowplot?

The `cowplot` package provides various features that help with creating publication-quality figures, such as a set of themes, functions to align plots and arrange them into complex compound figures, and functions that make it easy to annotate plots and or mix plots with images. The package was originally written for internal use in the Wilke lab, hence the name (Claus O. Wilke's plot package). --- [cowplot 1.1.1](https://wilkelab.org/cowplot/index.html) (<https://wilkelab.org/cowplot/index.html>)

The `cowplot` [documentation](https://wilkelab.org/cowplot/index.html) (<https://wilkelab.org/cowplot/index.html>) is very user friendly, so be sure to check it out. Until recently `cowplot` has been my go to for arranging and combining plots in a multi-figure plot panel. Today we will primarily focus on `patchwork`, described below, for this purpose because it is, in my opinion, far easier to use for simple figure alignment in a grid format. However, `cowplot` does have some features that are notable, especially related to label customization, unique plot arrangements, and image drawing. The [drawing functions](https://wilkelab.org/cowplot/articles/drawing_with_on_plots.html) (https://wilkelab.org/cowplot/articles/drawing_with_on_plots.html), which allow you to easily integrate outside images onto your plots, are especially useful, so take a look at the linked documentation. Some of the more difficult features of `cowplot`, such as getting a combined legend, have been simplified by wrapper functions from other packages (See `ggarrange()` from `ggpubr`).

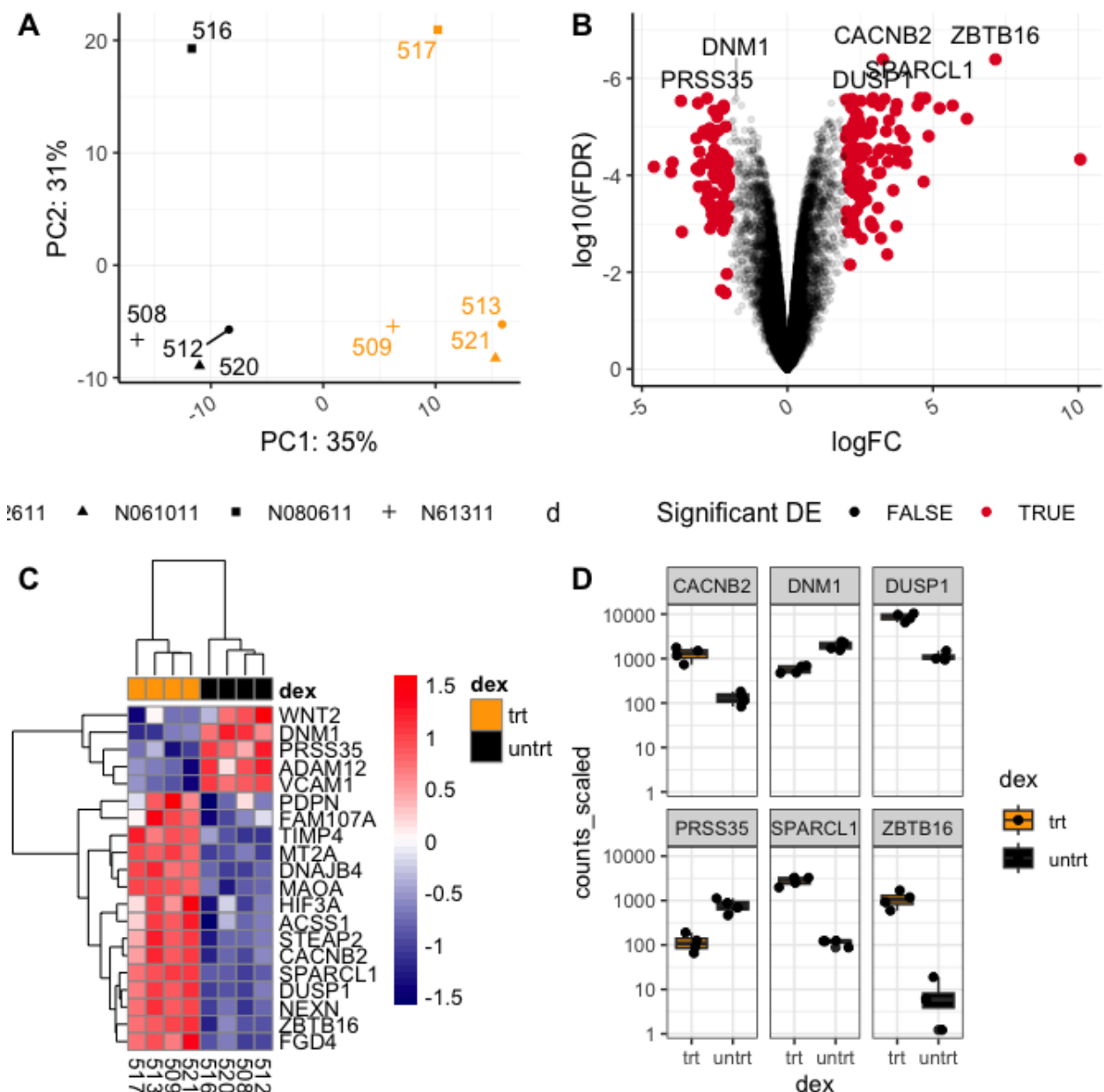
Using cowplot to arrange figures

The main function to combine figures using `cowplot` is `plot_grid()`. Let's check out the help documentation using `?plot_grid()`. The first and most important parameter is the list of plots we want to combine, `plotlist`.

Let's check out the basic use of this function by calling the plots we want to combine and by providing labels using the `labels` argument.

```
plot_grid(pca, volcano, hmap, sc, labels="AUTO")
```

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



This figure isn't bad. Though, we would likely want to change the relative sizes of the plots and work on the plot alignments. This can be done with `rel_heights`, `rel_widths`, `align`, and `axis`. We are not going to work with these today because `cowplot` does not do well with plot alignments when a given plot's aspect ratio has been fixed (e.g., `coord_fixed()`, `coord_equal()`).

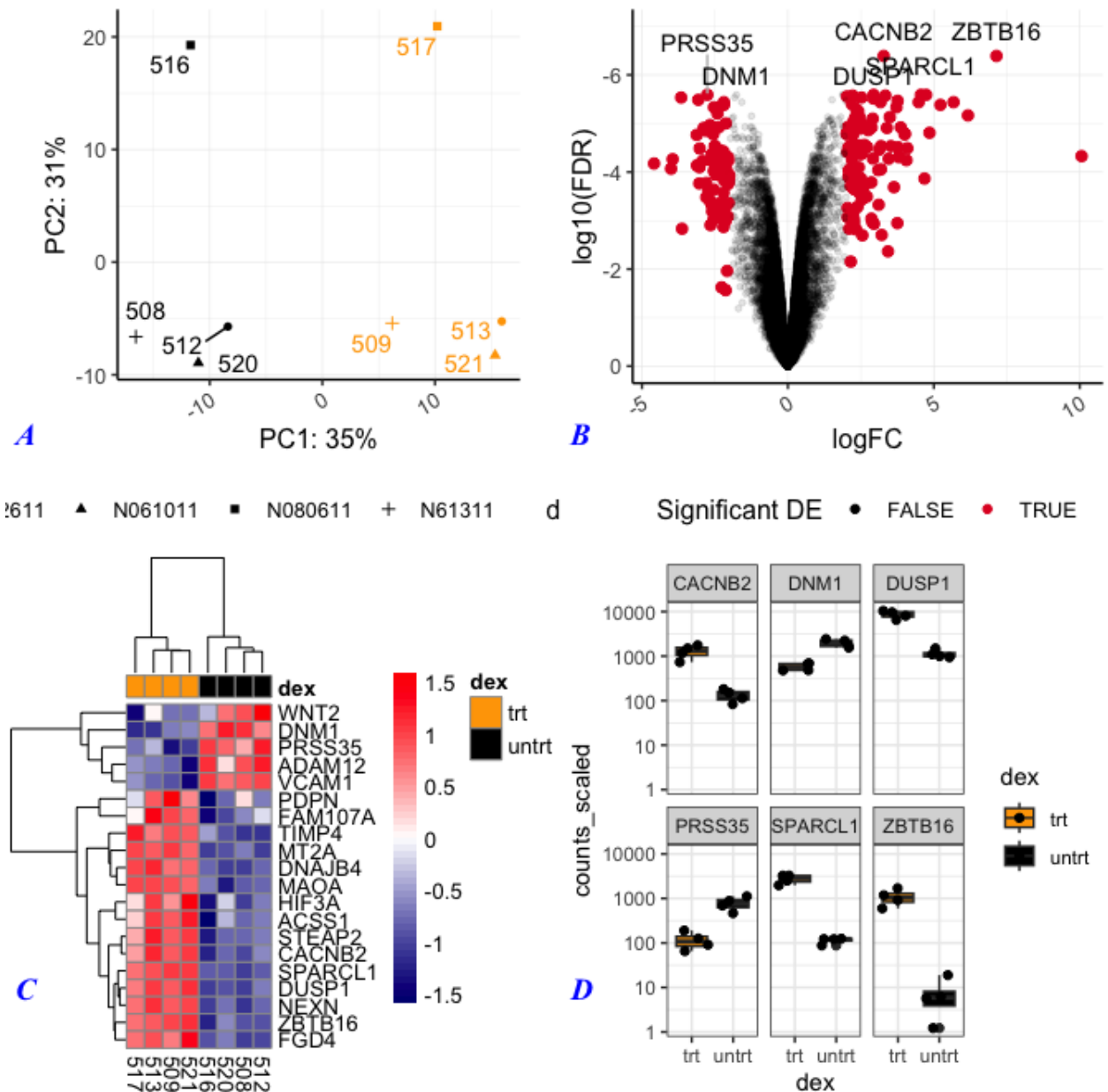
However, I do want to point out some of the options for label customization before moving on to `ggdraw()`.

Plotting labels with cowplot

There are quite a few parameters to adjust figure labels. To re-position labels, see `label_x`, `label_y`, `hjust`, and `vjust`. These each take either a single value to move all labels or a vector of values, one for each subplot. We can also change the size of the labels (`label_size`), the font (`label_fontface`), the label color (`label_colour`), and the font type (`label_fontfamily`). In general, there seem to be more options for plot customization (without adding `ggplot2` layers) with `cowplot`.

```
plot_grid(pca,volcano,hmap,sc, labels="AUTO",label_size = 14,  
          label_fontface = "bold.italic", label_colour = "blue",  
          label_fontfamily = "Times New Roman",  
          label_y=0.25)
```

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



Here we changed the size of the labels to 14 pt and the color to blue. Also, the labels are now bold and italicized, and the font is Times New Roman. We also re-positioned the labels.

Other notable features of cowplot

You can nest figures by combining figures using `plot_grid()`, saving that to an object, and then plotting those pre-combined figures with another figure using `plot_grid()` again. Shared legends can be obtained using `cowplot's get_legend()`. `cowplot` also has its own function to save plots (`save_plot()`), which is a bit more dynamic for multi-figure panels, but you may also use `ggsave()`.

Taking advantage of ggdraw

An extremely useful feature of `cowplot` is `ggdraw()` combined with `draw_plot()`, `draw_grob()`, `draw_image()` and `draw_label()`. These can be used to add or combine text, images, and plots and create more complicated plot arrangements. `ggdraw()` can be used directly with any package that creates grid grobs. It can also be used with `lattice` and base R graphics. The output of these functions can be treated like `ggplot2` objects.

Note: Grobs are graphical objects that you can make and change with grid graphics functions. The `ggplot2` package is built on top of grid graphics, so the grid graphics system “plays well” with `ggplot2` objects. --- [Pang, Kross, and Andersen, 2020 \(https://bookdown.org/rdpeng/RProgDA/the-grid-package.html\)](https://bookdown.org/rdpeng/RProgDA/the-grid-package.html)

Let's look at the basics of drawing an image.

Note: Drawing an image requires `library(magick)`.

```
plos <- "./images/himesetal2014_plos.png" #save our image file path

#Create an image to add to a multi-figure plot.
a<-ggdraw() + #create blank canvas
  draw_image(plos,scale=1) # draw the image and save to object




a
```

PLOS ONE

OPEN ACCESS PEER-REVIEWED

RESEARCH ARTICLE

RNA-Seq Transcriptome Profiling Identifies *CRISPLD2* as a Glucocorticoid Responsive Gene that Modulates Cytokine Function in Airway Smooth Muscle Cells

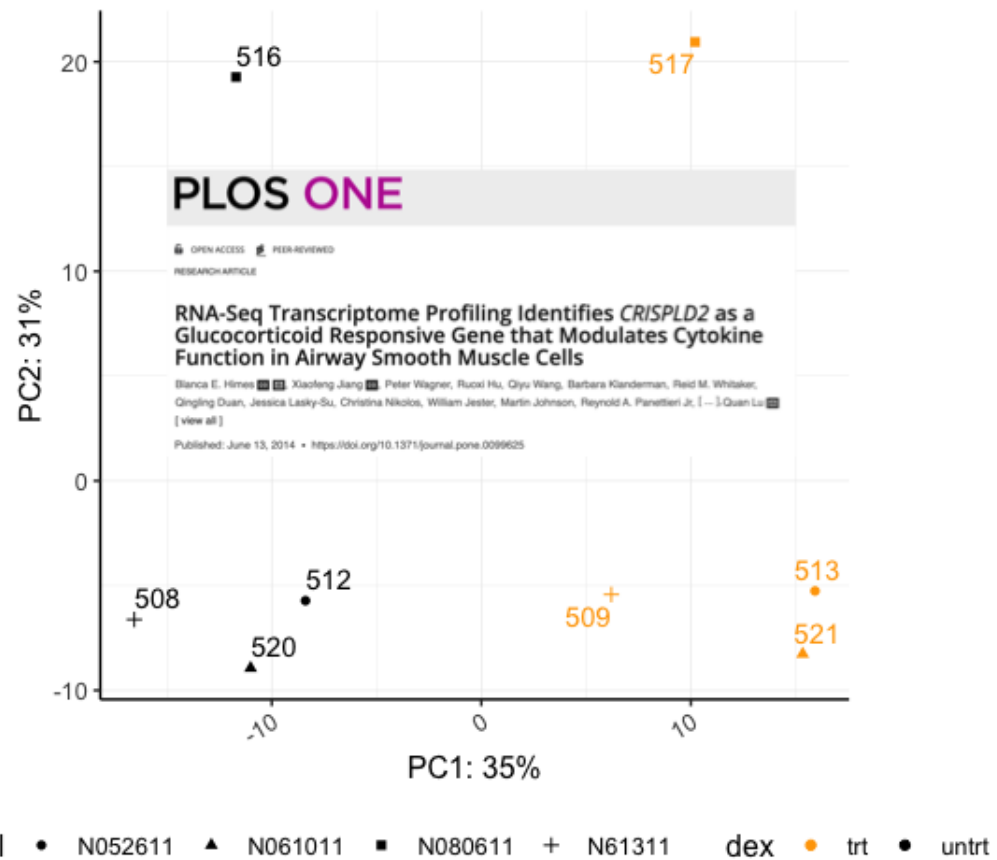
Blanca E. Himes , Xiaofeng Jiang , Peter Wagner, Ruoxi Hu, Qiyu Wang, Barbara Klanderman, Reid M. Whitaker, Qingling Duan, Jessica Lasky-Su, Christina Nikolos, William Jester, Martin Johnson, Reynold A. Panettieri Jr, [...], Quan Lu  [view all]

Published: June 13, 2014 • <https://doi.org/10.1371/journal.pone.0099625>

`ggdraw()` creates a new `ggplot2` canvas without visible axes or background grid. The `draw_*` functions are simply wrappers around regular geoms. --- [cowplot documentation \(https://wilkelab.org/cowplot/articles/drawing_with_on_plots.html\)](https://wilkelab.org/cowplot/articles/drawing_with_on_plots.html)

Let's add this to the background of our PCA.

```
pca + draw_image(plos, x=-15, y=-2, width=30, height=20)
```



Using cowplot we can get an image such as this:

```
p1<-plot_grid(NULL,a,NULL,(pca+theme(legend.position="right")),labels=
bc<-plot_grid(volcano,sc,labels = c("B","C"),align="h",axis="b")
## Warning: Using alpha for a discrete variable is not advised.
p2<-plot_grid(p1,NULL,bc,ncol=1,align="v",axis="l",rel_heights = c(1
p2b<-plot_grid(hmap,NULL,nrow=1,align="h",axis="b",rel_widths=c(0.65
p3<-plot_grid(NULL,p2,NULL,p2b,ncol=1,labels=c("", "", "D"),align="v",a
labp<-ggdraw(p3)+
  draw_label("Airway Data", color = "Black", size = 14, x=0.1,y=0.9)
labp
```

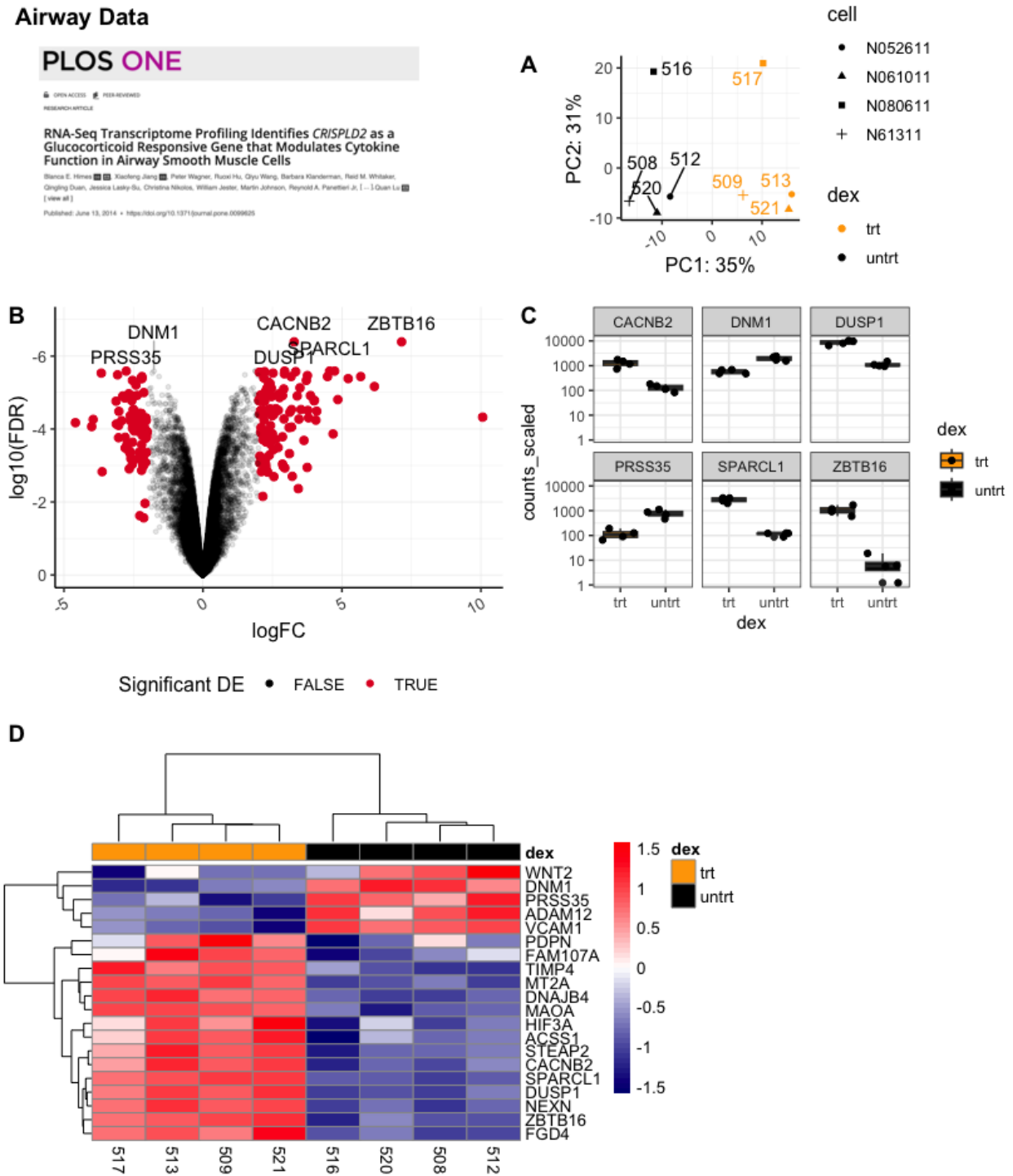
Airway Data

PLOS ONE

OPEN ACCESS RESEARCH ARTICLE

RNA-Seq Transcriptome Profiling Identifies *CRISPLD2* as a Glucocorticoid Responsive Gene that Modulates Cytokine Function in Airway Smooth Muscle Cells

Blanca E. Himm, Xiaoling Jiang, Peter Wagner, Russell H. Qiu, Wang, Barbara Klanderman, Reid M. Whittaker, Qingling Qian, Jessica Lashy-Su, Christina Nikolic, William Justice, Martin Johnson, Reynolds A. Parrott, et al. | DOI: 10.1371/journal.pone.0109925



What is patchwork?

The goal of patchwork is to make it ridiculously simple to combine separate ggplots into the same graphic. As such it tries to solve the same problem as `gridExtra::grid.arrange()` and `cowplot::plot_grid` but using an API that incites exploration and iteration, and scales to arbitrarily complex layouts. ---Thomas Lin Pederson, [Patchwork documentation \(https://patchwork.data-imaginist.com/index.html\)](https://patchwork.data-imaginist.com/index.html).

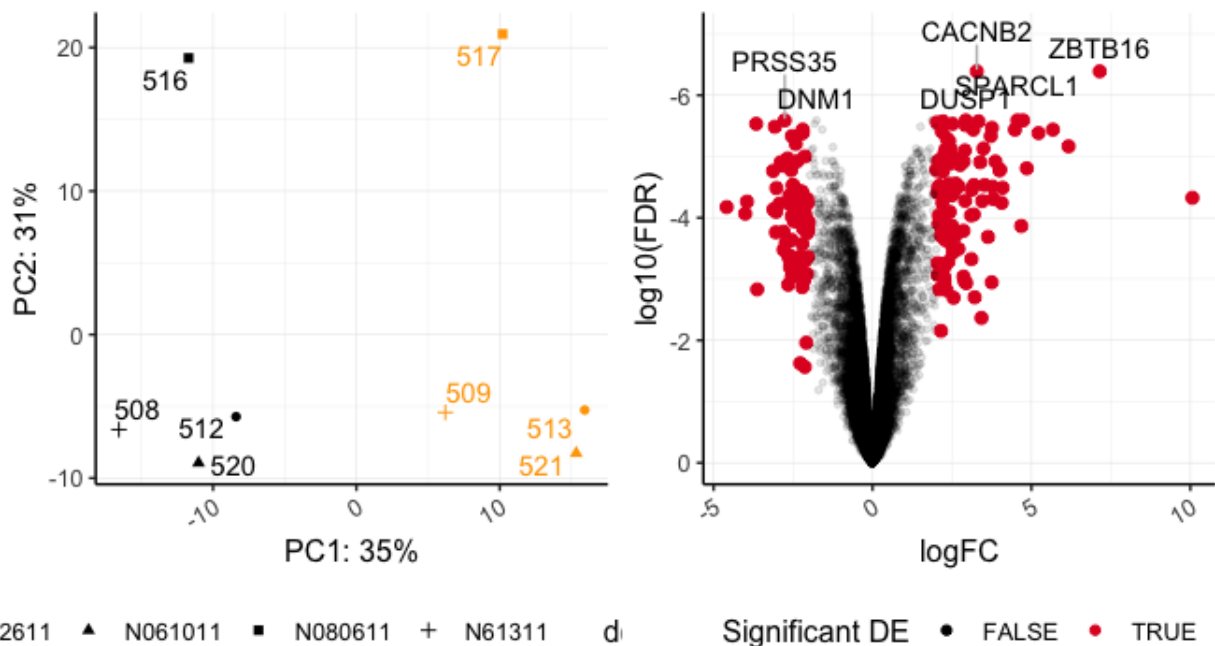
Patchwork allows users to combine plots using simple mathematic operations such as + and /.

Combining two plots

Let's combine our PCA and volcano plots.

```
pca + volcano
```

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

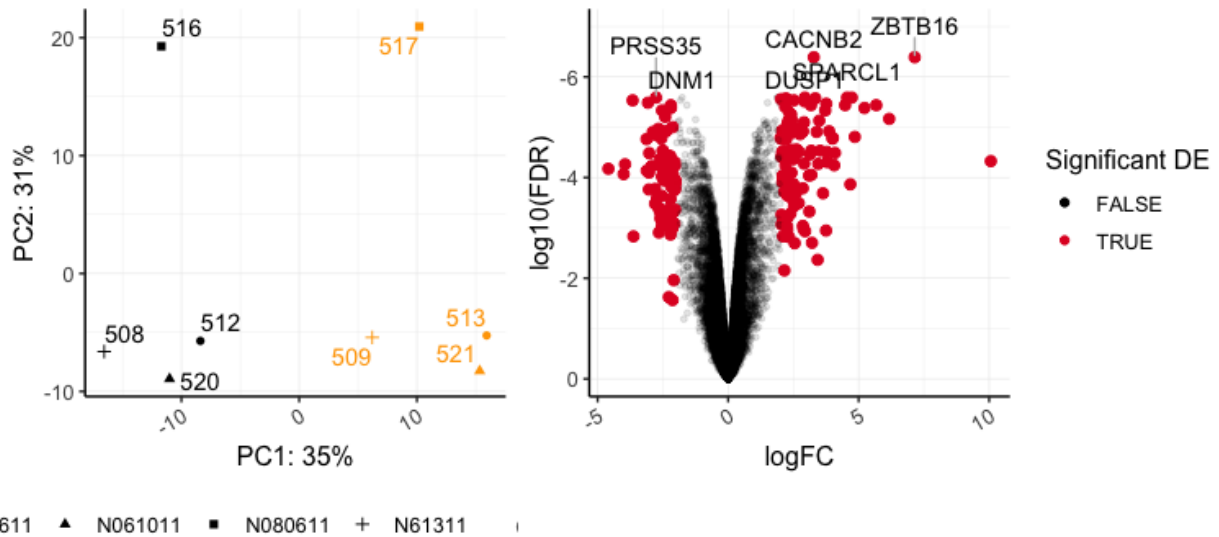


The last plot included in patchwork statements is considered the active plot, to which we can add additional ggplot2 layers. Notice the seamless alignment of these plots. Without any additional parameters, `coord_fixed()` is maintained in the `pca` plot. `patchwork` does a lot better with a fixed aspect plot.

Let's customize the legend position of the active plot.

```
pca + volcano +  
  theme(legend.position="right")
```

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

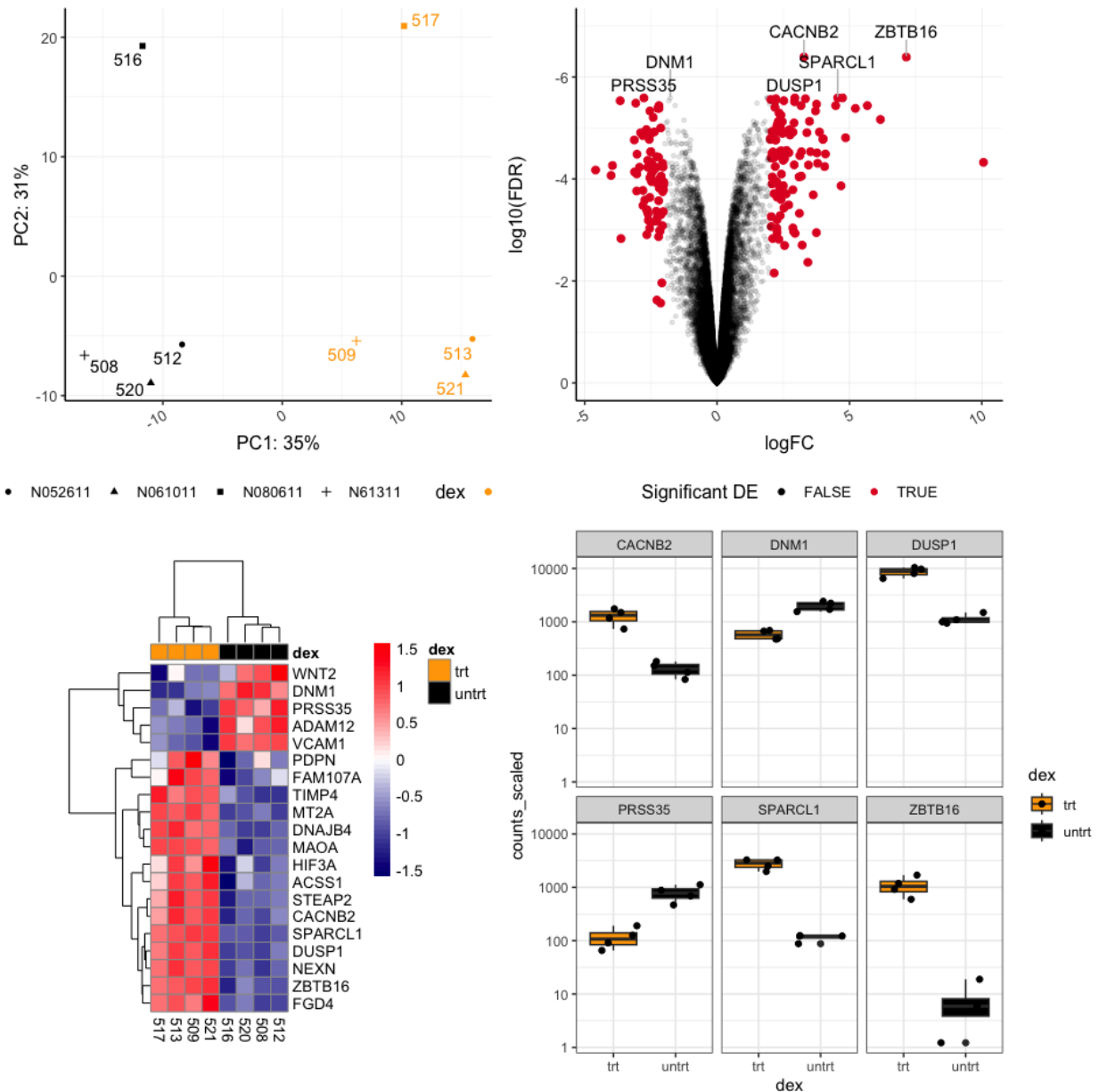


We can easily add ggplot2 layers within our patchwork code.

We can continue to add plots using the + symbol, and patchwork will try to form a grid, proceeding from left to right row-wise. Let's see this in action.

```
pca + volcano + hmap + sc
```

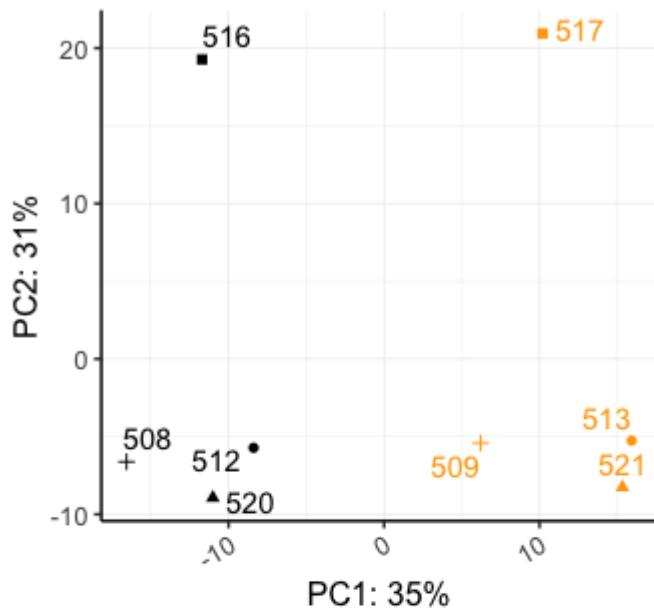
```
## Warning: Using alpha for a discrete variable is not advised.
```



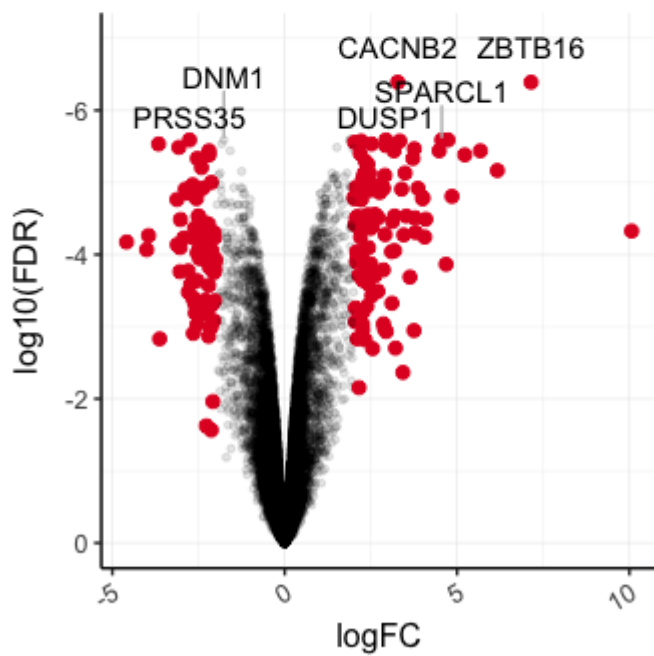
The plot layout can be controlled further with additional operators. The | symbol is used to place plots side by side, while the / symbol is used to stack plots vertically. Plotting layouts kind of follow the rules designated by the order of operations (Remember back to PEMDAS), the / occurs before | and +.

```
#vertical stacking
pca / volcano
```

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



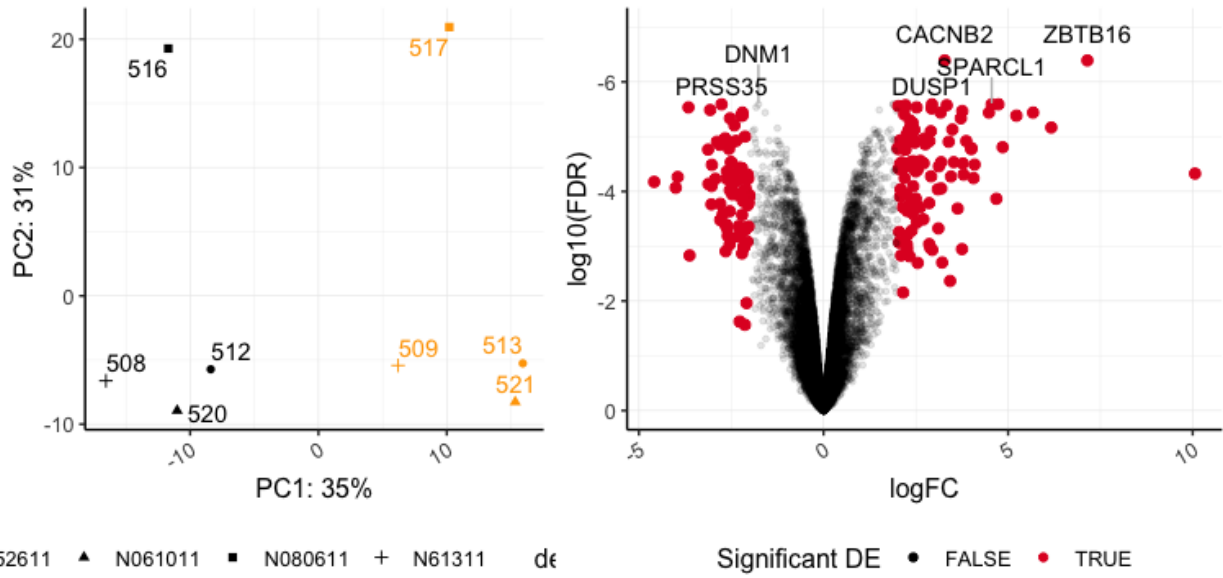
N052611 ▲ N061011 ■ N080611 + N61311 dex



Significant DE • FALSE • TRUE

```
#horizontal
pca | volcano
```

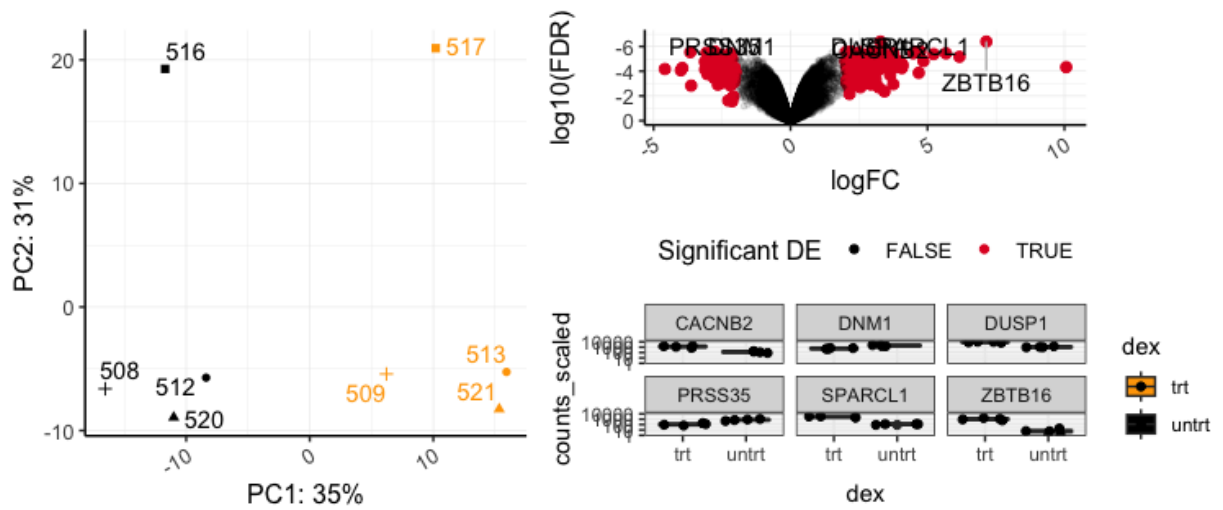
```
## Warning: Using alpha for a discrete variable is not advised.
```

52611 ▲ N061011 ■ N080611 + N61311 d

```
pca | volcano / sc
```

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



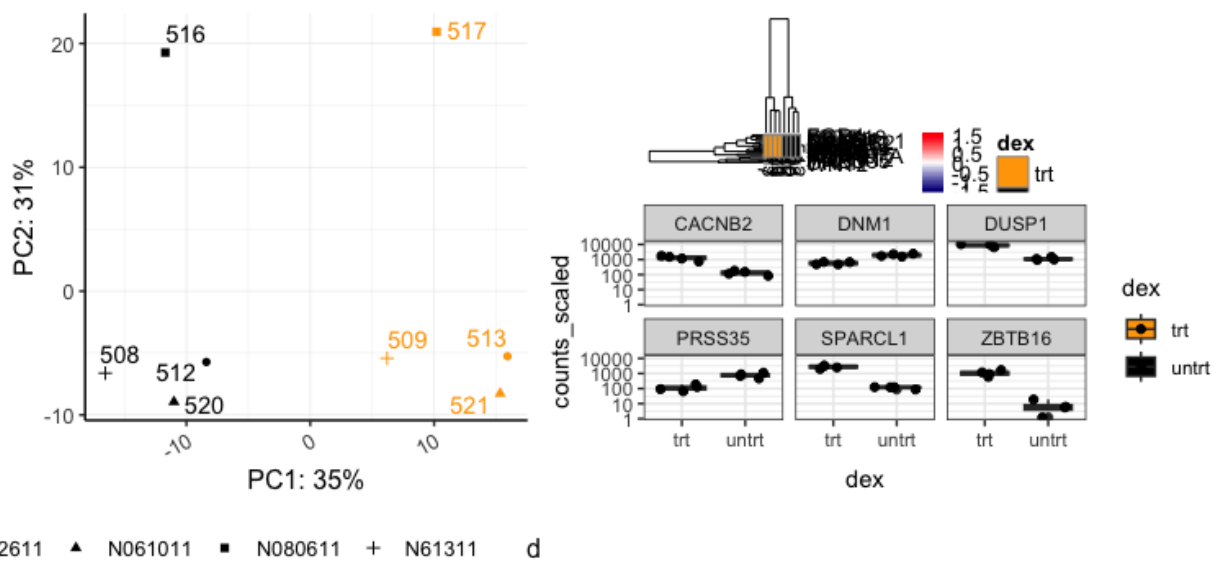
52611 ▲ N061011 ■ N080611 + N61311 d

For specific layouts, it is a good idea to use parentheses for correct evaluation.

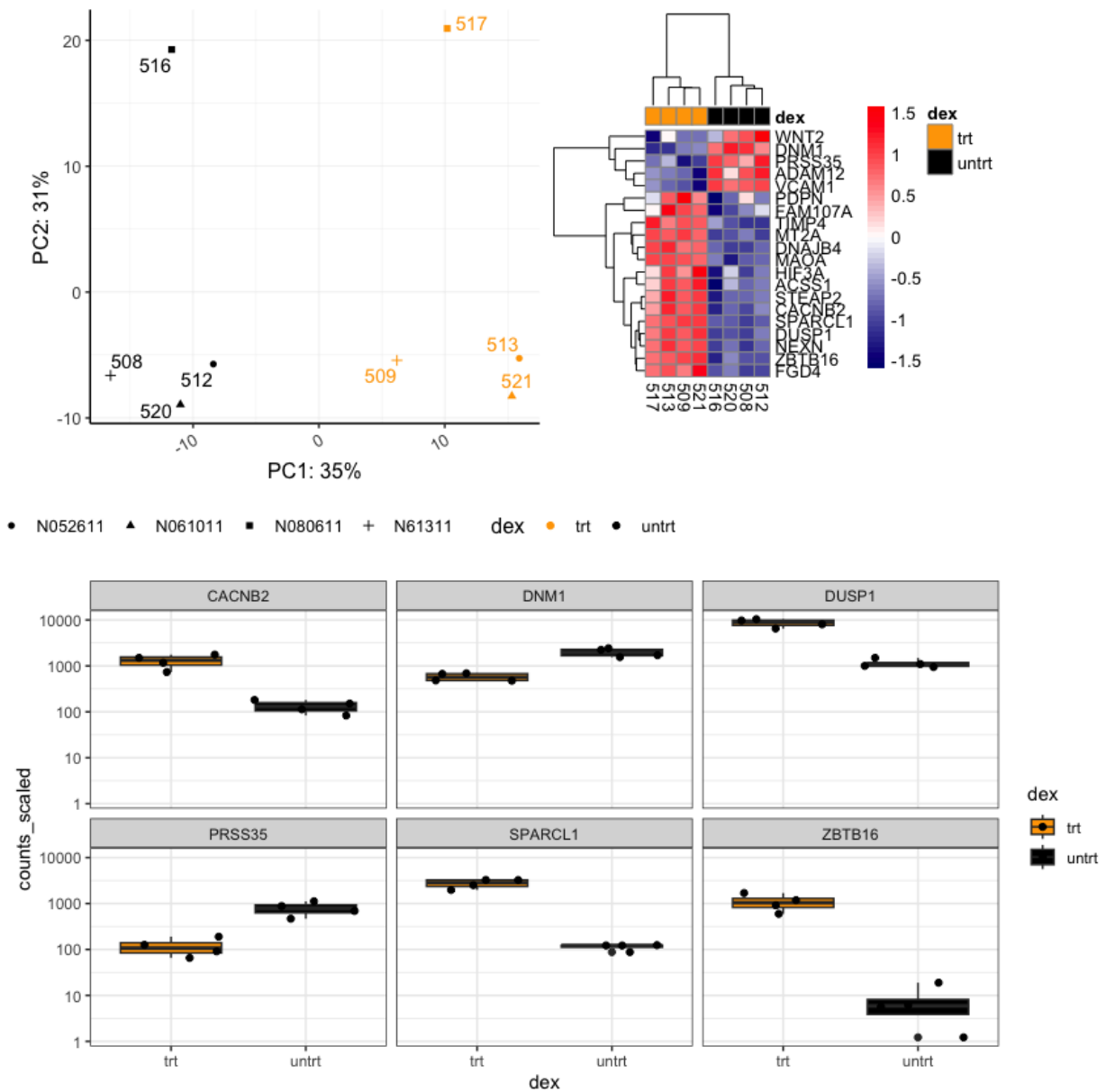
For example, what if we want our pca and heatmap side by side stacked on top of our box plot?

Compare the following:

```
pca | hmap / sc
```



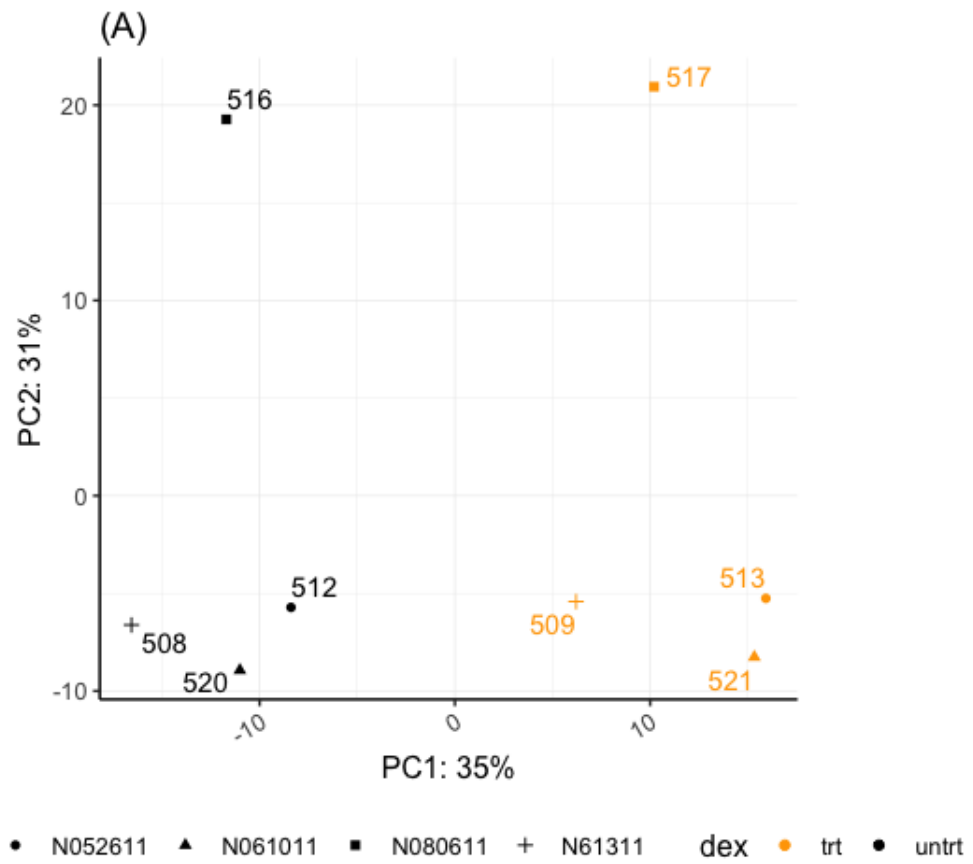
```
((pca | hmap) / sc)
```



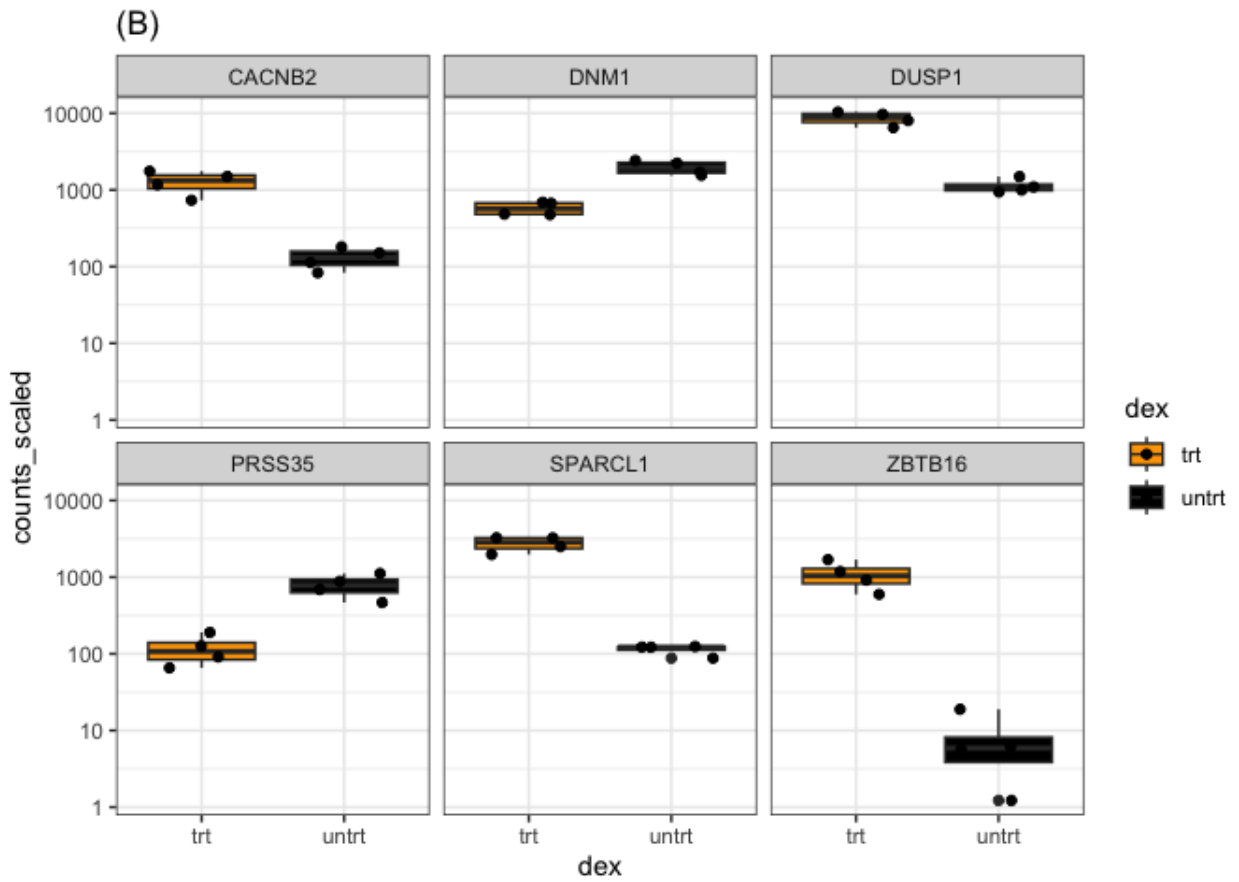
Using `plot_layout()`

The function `plot_layout()` can be used to control the layout, combine legends, and overwrite plot titles.

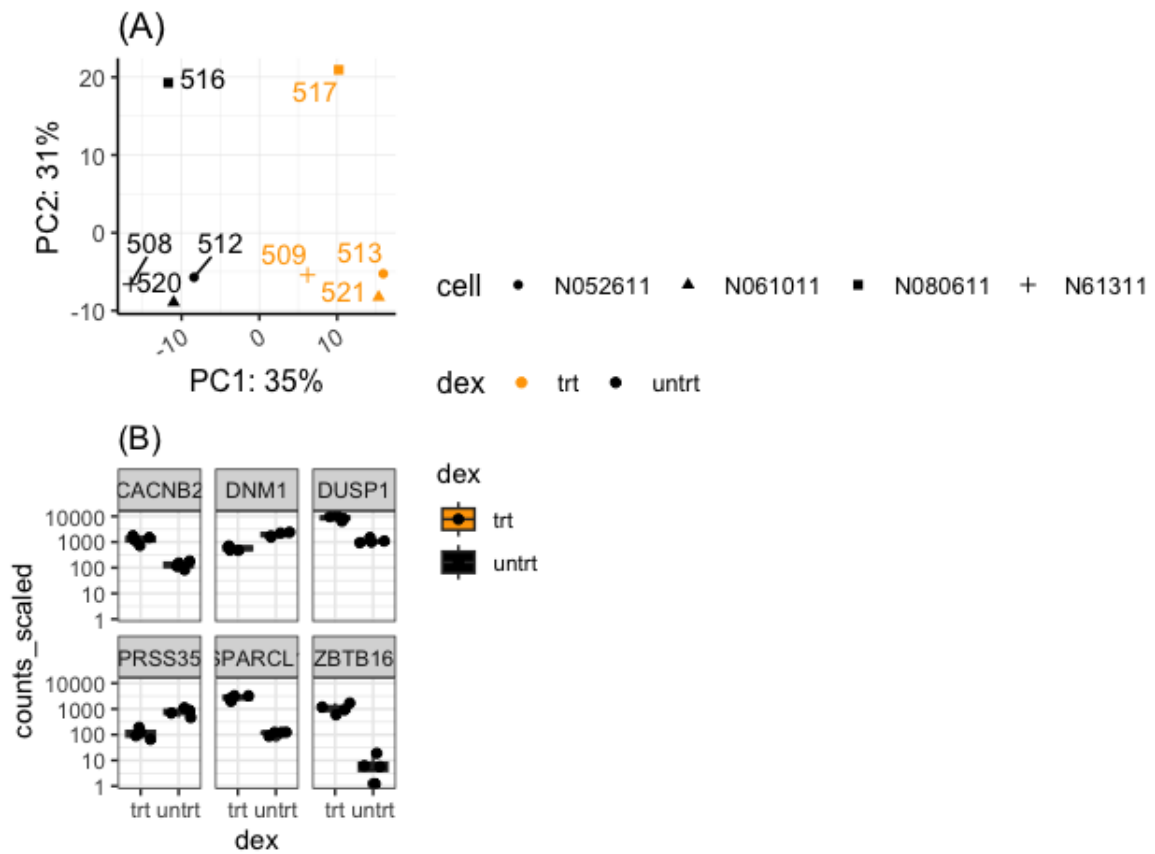
```
pca1 <- pca + ggtitle("(A)")
pca1
```



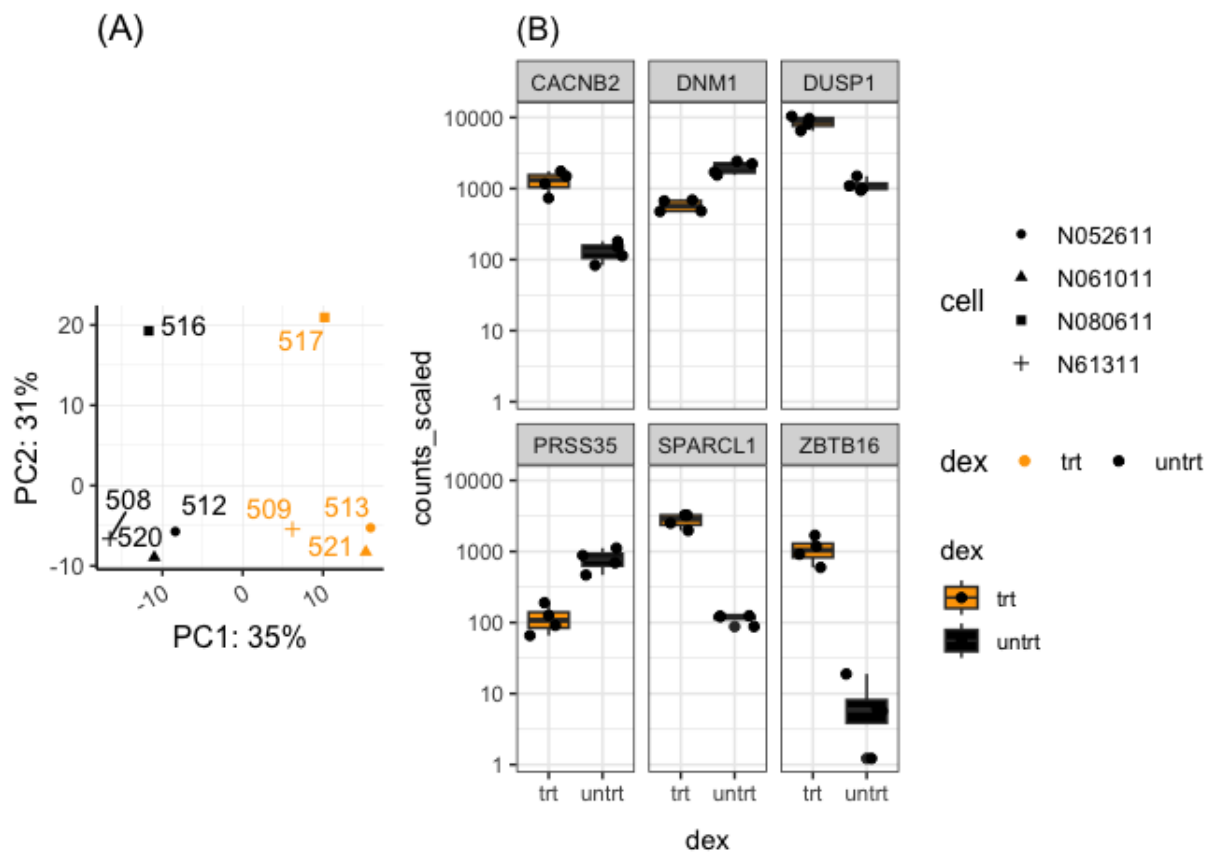
```
sc1 <- sc + ggtitle("(B)")
sc1
```



```
#combine legends  
pca1 + sc1 + plot_layout(ncol = 1, guides = 'collect')
```



```
#Can control the relative heights and widths
pca1 + guides(shape = guide_legend(nrow=4)) + sc1 +
  plot_layout(nrow=1, guides = 'collect', widths=c(1.5,2))
```



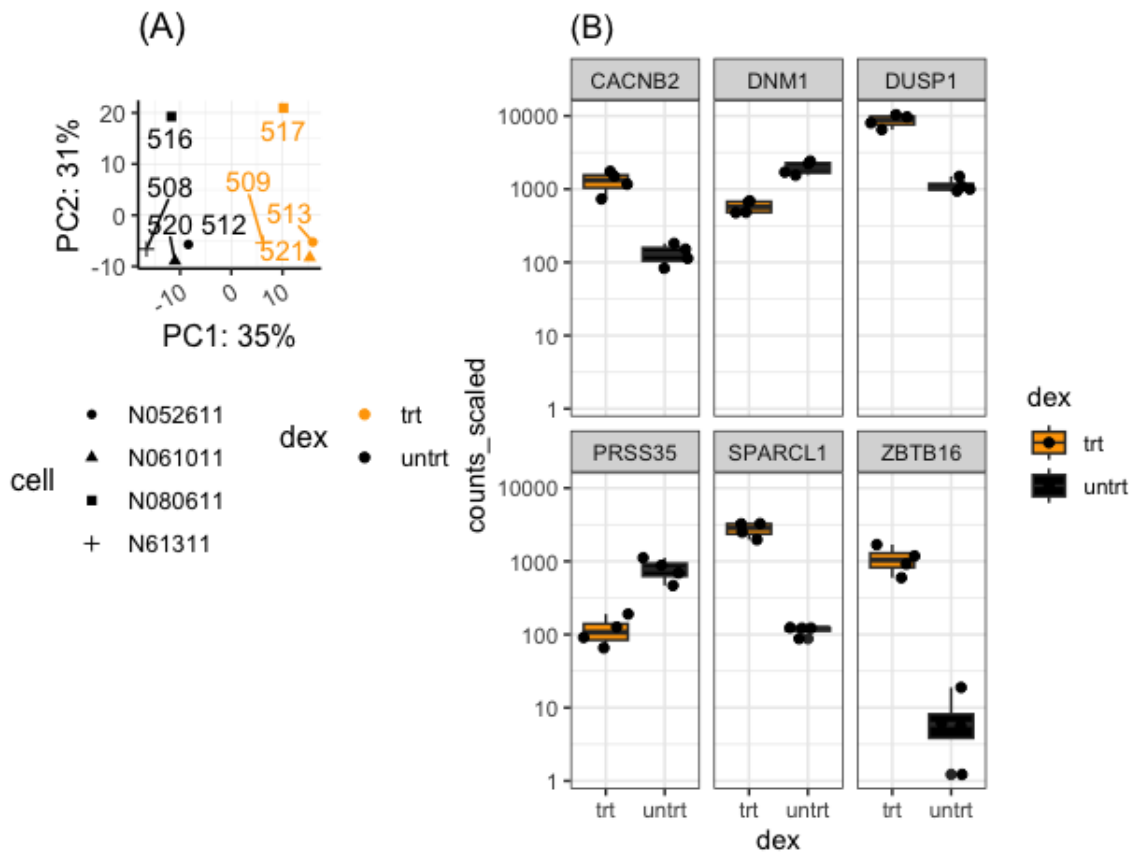
Changing the relative sizes of the figures alters the alignment. Units may also be specified to control the height or width.

It is possible to design unique (non-grid) layouts using `patchwork`, but it seems a bit more difficult than `cowplot` in that regard.

Adding a spacer

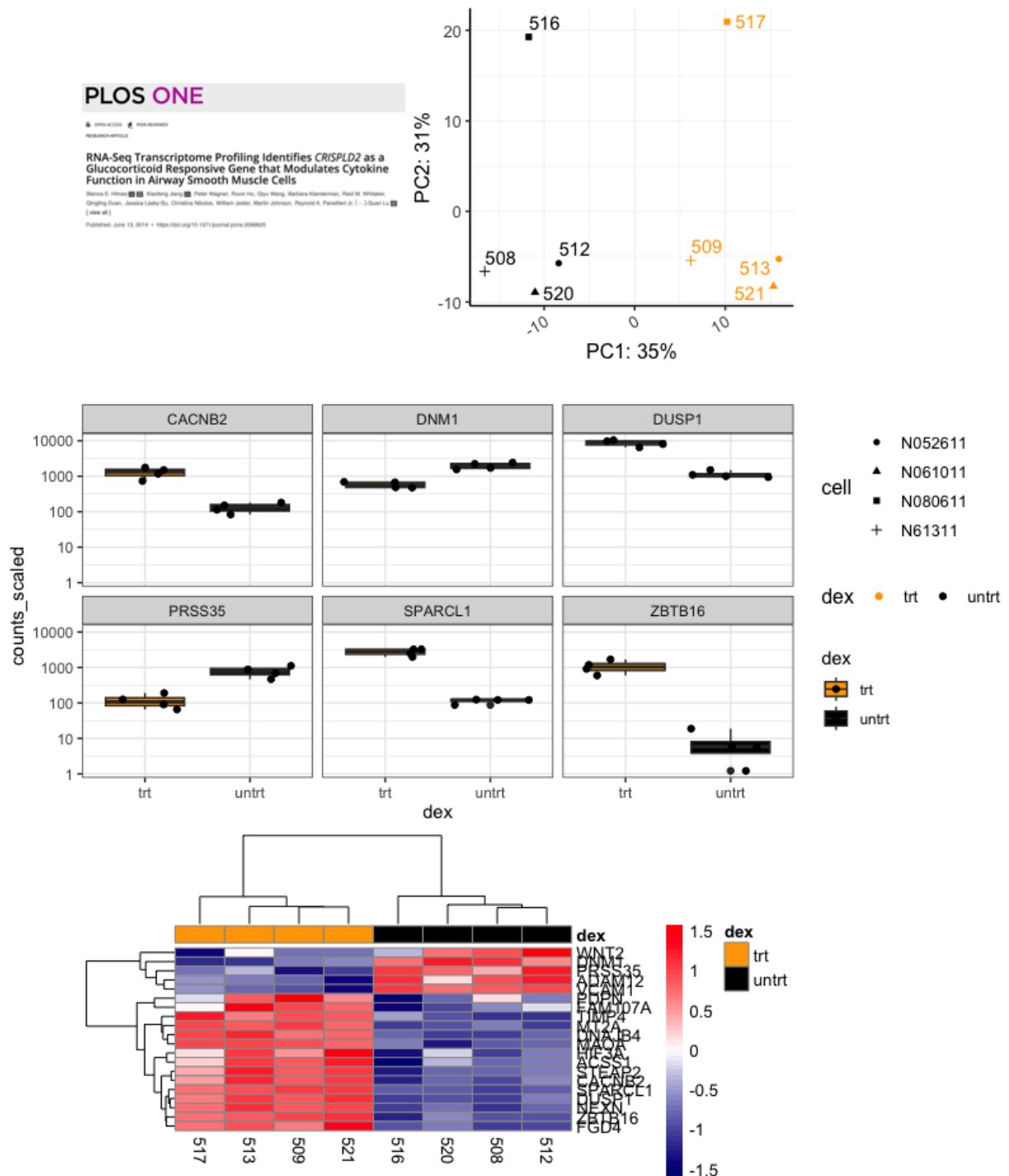
We can add a spacer using `plot_spacer()` to add blank sections to our plot. These blank sections are the size of our figure panels and may be different depending on how the plot is arranged.

```
(pca1 + guides(shape = guide_legend(nrow=4),
               color = guide_legend(nrow=2))) / plot_spacer() | sc1
```



Add our cowplot image

```
pca2<- pca + guides(shape = guide_legend(nrow=4))
((a |pca2) / sc / hmap) + plot_layout(guides='collect')
```

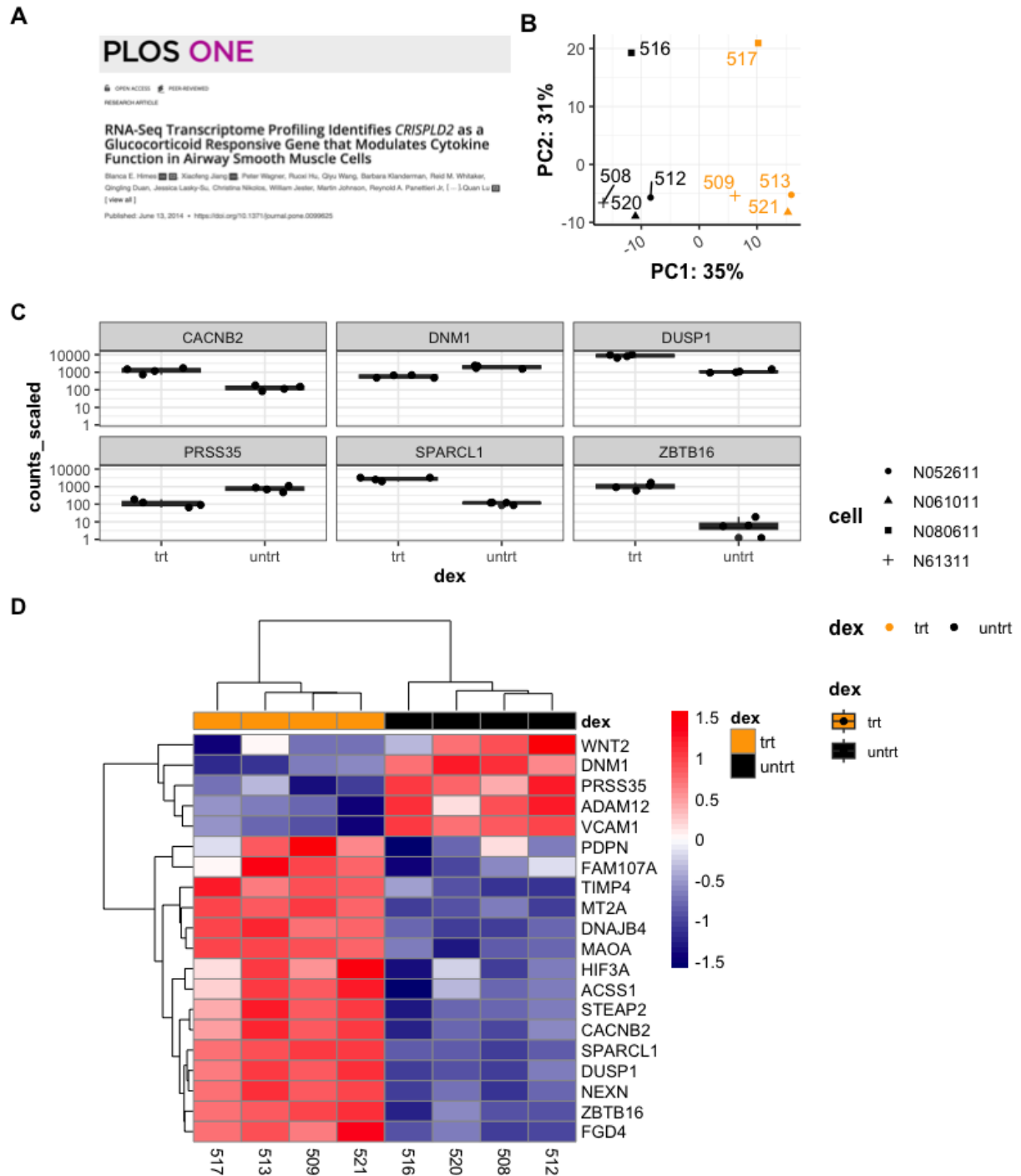
Including a title

To include a title, subtitle, or caption use the function `plot_annotation()`.

```
patchf<-((a | pca2) / sc / hmap) +
  plot_layout(guides='collect',heights=c(1,1,3)) +
  plot_annotation(title = 'Airway Data', tag_levels = "A") &
  theme(plot.tag = element_text(face = 'bold'),
        title= element_text(face = 'bold'))
```

patchf

Airway Data



Notice that the assignment of "A" and "B" were automatic with the `tag_levels` parameter. The parentheses here are required. Also, `&` was used to apply `theme()` to all subplots in the patchwork.

Save the plot

```
ggsave("patchf.png", height=5, width=4.25, dpi=300, units="in", scale=2)
```

Which package to use?

This is ultimately up to you. In general, `patchwork` is easier to combine nicely aligned grid style plots. `cowplot` seems to include greater opportunities for customization, but you could easily customize things like titles using `ggplot2` layers with either package prior to combining figures. `patchwork` also allows integration of `ggplot2` layers. Both packages also allow you to inset figures and add non-`ggplot2` figures. As you gain more experience with R, you may find yourself using both packages to achieve specific goals, or you may look to other packages. `egg` (<https://cran.r-project.org/web/packages/egg/vignettes/Overview.html>) is another popular package for combining figures. `ggarrange()` (<https://rpkgs.datanovia.com/ggpubr/reference/ggarrange.html>) from the package `ggpubr` is also popular.

Acknowledgements

Content in this tutorial was adapted from information in the `cowplot` documentation (https://wilkelab.org/cowplot/articles/plot_grid.html) and `patchwork` documentation (<https://patchwork.data-imaginist.com/>).

Getting the Data

Course Data

Course data is available in the attached zipped archive: [data.zip](#).

Practice Questions



Practice plotting using ggplot2: Lesson 2

Load the data

For these exercises, you will explore the titanic data from [kaggle.com \(https://www.kaggle.com/c/titanic/data\)](https://www.kaggle.com/c/titanic/data), which was downloaded from [here \(https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/problem12.html\)](https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/problem12.html). You will need to download the data and load into R. As this is a comma separated file, you will need to explore the `read.csv()` function.

Description of the data:

Column	Description
Survived	0 = No, 1 = Yes
Pclass	Ticket Class / Socioeconomic status (1 = 1st, 2 = 2nd, 3 = 3rd)
Name	Passenger name
Sex	Male / Female
Age	Numeric age in years
Siblings/Spouses Aboard	# of siblings / spouses aboard the Titanic
Parents/Children Aboard	# of parents / children aboard the Titanic
Fare	Passenger fare

Get the data [here](#).

Load `ggplot2`.

```
library(ggplot2)
```

Exercise Questions

Question 1

Load `titanic.csv` and save to an object named `titanic`.

{{Sdet}}

Possible Solution{{Esum}}

```
titanic <- read.csv("https://web.stanford.edu/class/archive/cs/cs109/
```

```
{{Edet}}
```

Question 2

Explore the data. What is the structure of the data? Try `str()`. What are the column names? Try `colnames()`. How can you get help if you do not know how to use these functions?

```
{{Sdet}}
```

Possible Solution{{Esum}}

```
str(titanic) # get the structure
```

```
## 'data.frame':      887 obs. of  8 variables:
##  $ Survived          : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass            : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name              : chr  "Mr. Owen Harris Braund" "Mrs. Jo
##  $ Sex               : chr  "male" "female" "female" "female"
##  $ Age               : num  22 38 26 35 35 27 54 2 27 14 ...
##  $ Siblings.Spouses.Aboard: int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parents.Children.Aboard: int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Fare              : num  7.25 71.28 7.92 53.1 8.05 ...
```

```
colnames(titanic) # get the column names.
```

```
## [1] "Survived"          "Pclass"
## [3] "Name"              "Sex"
## [5] "Age"               "Siblings.Spouses.Aboard"
## [7] "Parents.Children.Aboard" "Fare"
```

```
?str # get help
```

```
?colnames
```

```
{{Edet}}
```


Question 3

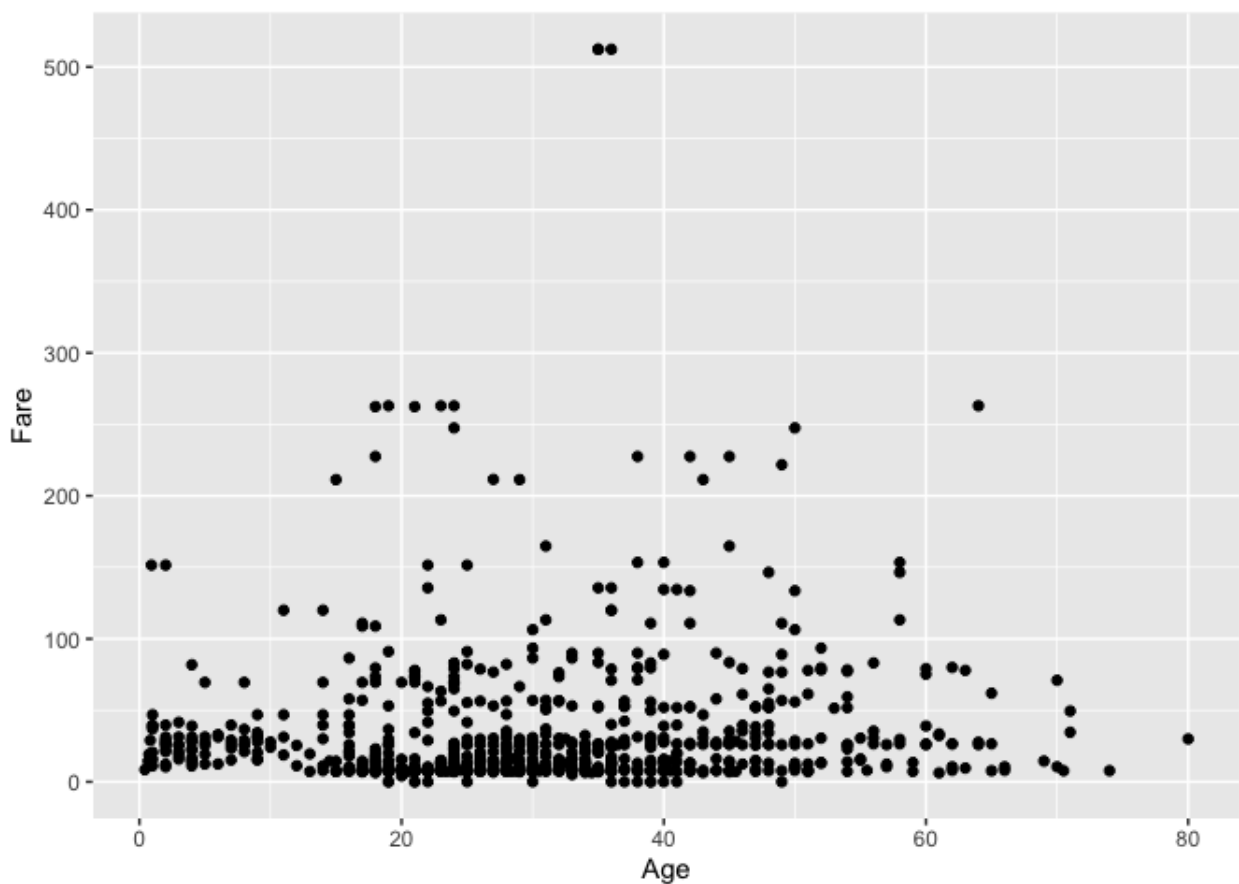


Make a simple scatter plot. Is there a relationship between the age of the passenger and the passenger fare?

{{Sdet}}

Possible Solution{{Esum}}

```
ggplot(titanic) +  
  geom_point(aes(x=Age, y=Fare))
```



{{Edet}}

Question 4

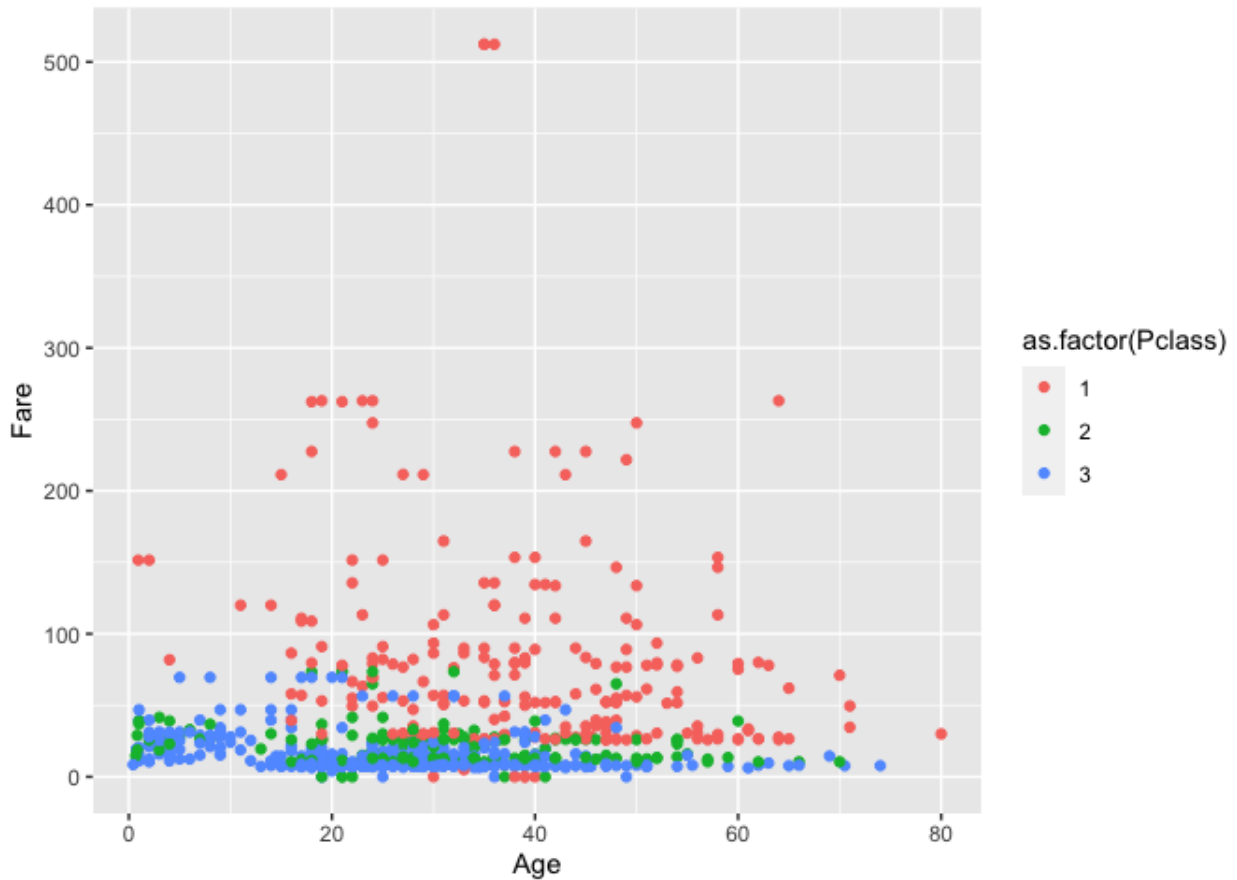
Color the points from question 3 by Pclass. Remember that Pclass is a proxy for socioeconomic status. While the values are treated as numeric upon loading, they are really categorical and should be treated as such. You will need to coerce Pclass into a categorical (factor) variable. See `factor()` and `as.factor()`.

{{Sdet}}

Possible Solution{{Esum}}



```
ggplot(titanic) +
  geom_point(aes(x=Age, y=Fare, color=as.factor(Pclass)))
```



{{Edet}}

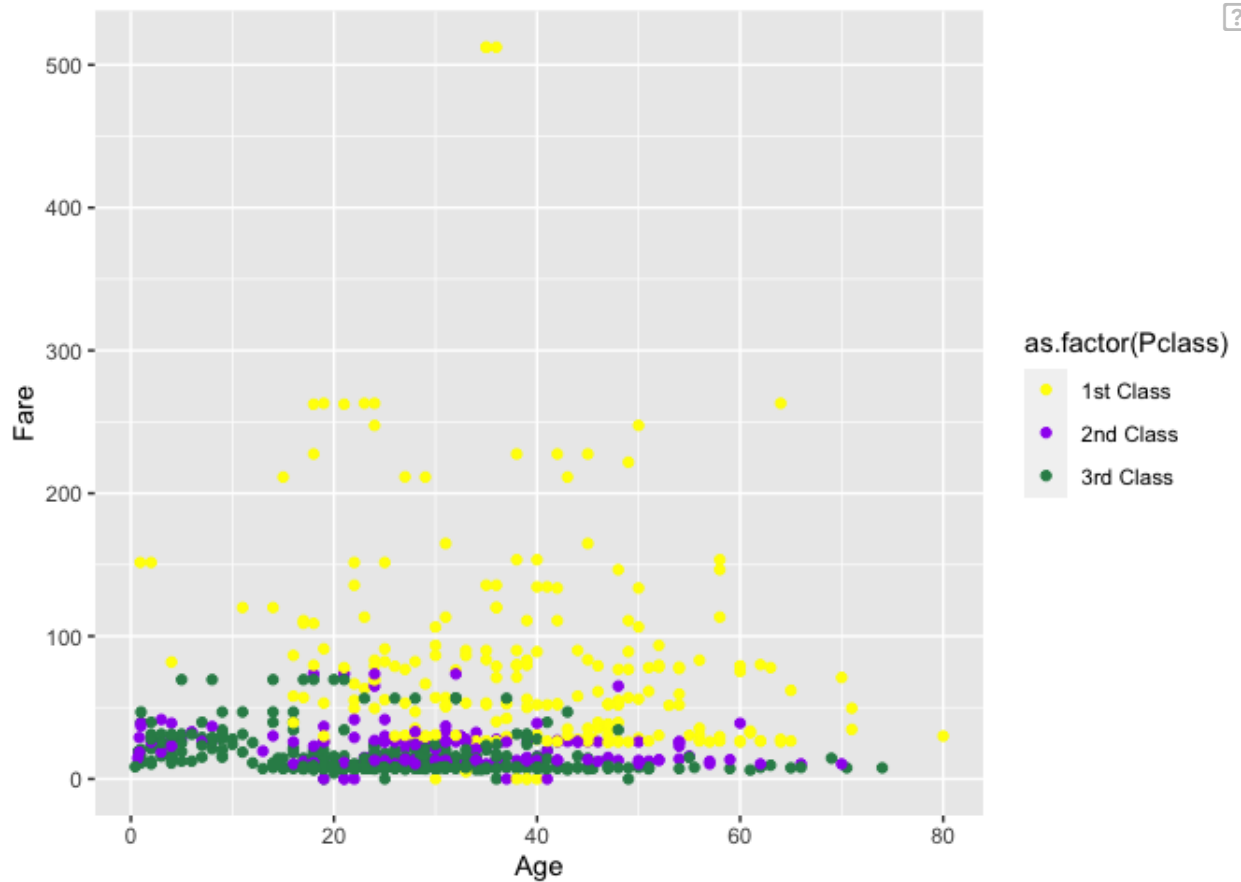
Question 5

Manually scale the colors in question 4. 1st class = yellow, 2nd class = purple, 3rd class = seagreen. Also change the legend labels (1 = 1st Class, 2 = 2nd Class, 3 = 3rd Class).

{{Sdet}}

Possible Solution{{Esum}}

```
ggplot(titanic) +
  geom_point(aes(x=Age, y=Fare, color=as.factor(Pclass)))+
  scale_color_manual(values=c("yellow", "purple", "seagreen"),
                    labels=c("1st Class", "2nd Class", "3rd Class"))
```



```
{{Edet}}
```

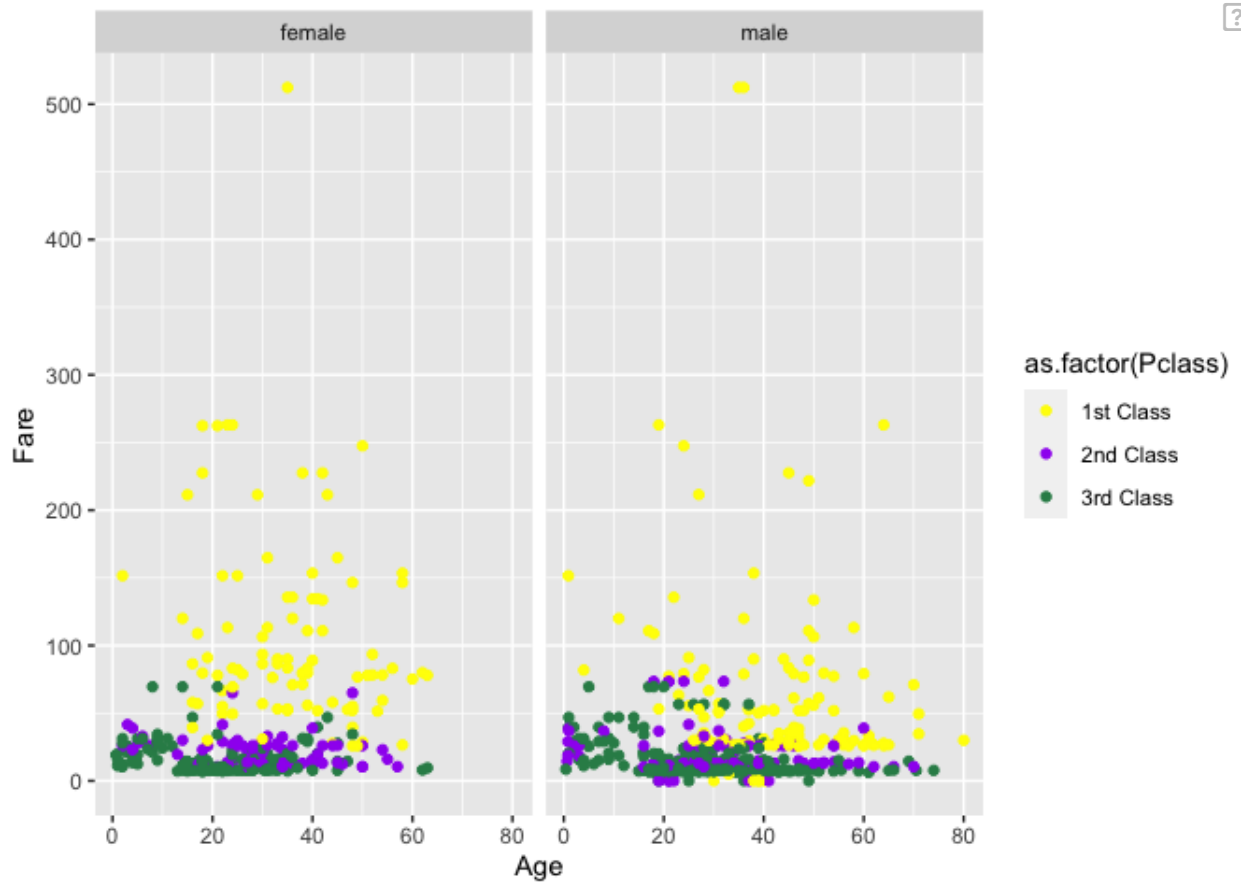
Question 6

Facet the plot made in 5 by the column 'Sex'.

```
{{Sdet}}
```

Possible Solution{{Esum}}

```
ggplot(titanic) +
  geom_point(aes(x=Age, y=Fare, color=as.factor(Pclass))) +
  scale_color_manual(values=c("yellow","purple","seagreen"),
                    labels=c("1st Class","2nd Class","3rd Class")) -
  facet_wrap(~Sex)
```



```
{{Edet}}
```

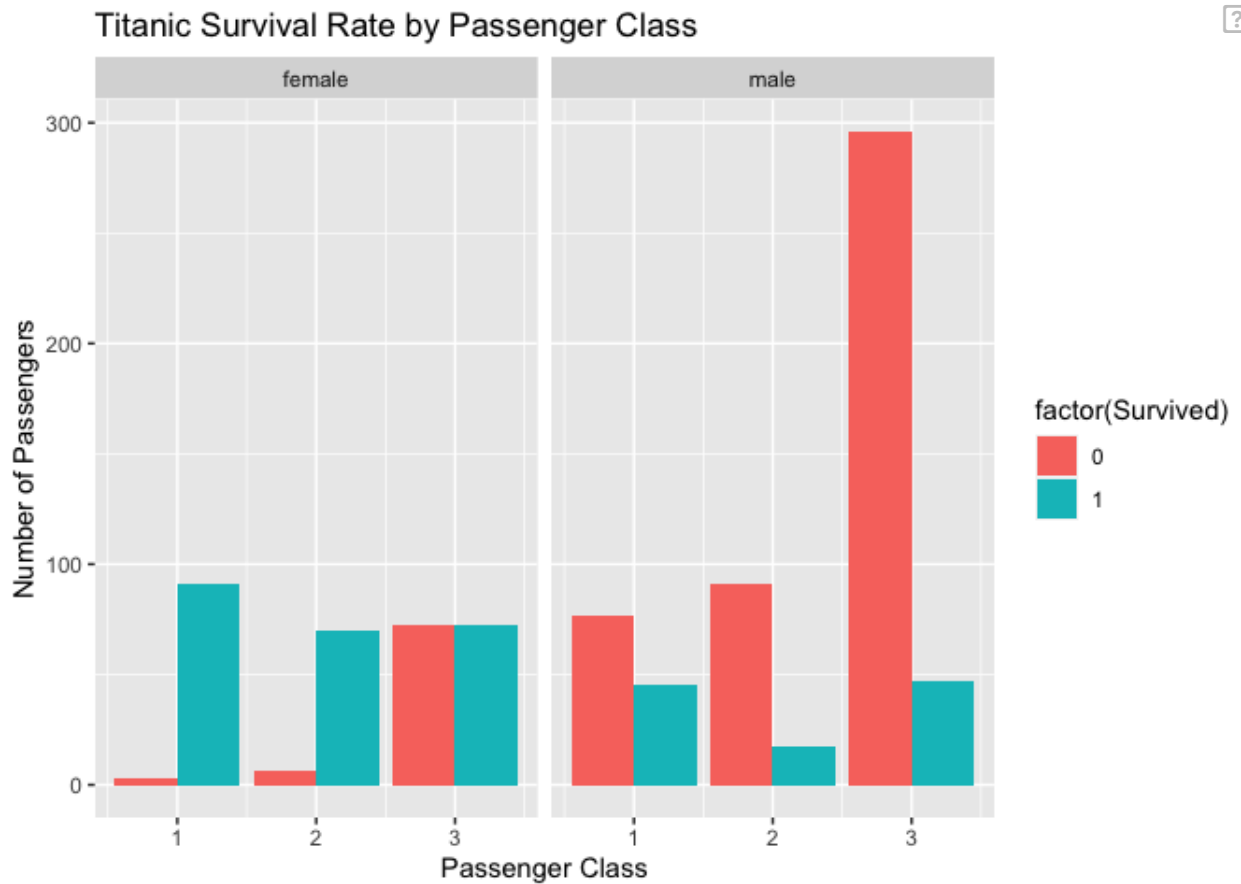
Challenge question 1

Let's use some other geoms. Plot the number of passengers (a simple count) that survived by ticket class and facet by sex.

```
{{Sdet}}
```

Possible Solution{{Esum}}

```
ggplot(titanic) +
  geom_bar(aes(x=Pclass, fill=factor(Survived)),
           position=position_dodge()) +
  facet_wrap(~Sex)+
  labs( y="Number of Passengers", x="Passenger Class",
        title="Titanic Survival Rate by Passenger Class")
```



```
{{Edet}}
```

Challenge question 2

Add a variable to the data frame called `age_cat` (`child = <12`, `adolescent = 12-17`, `adult = 18+`). Plot the number of passengers (a simple count) that survived by `age_cat`, fill by `Sex`, and facet by `class` and `survival`.

```
{{Sdet}}
```

Possible Solution{{Esum}}

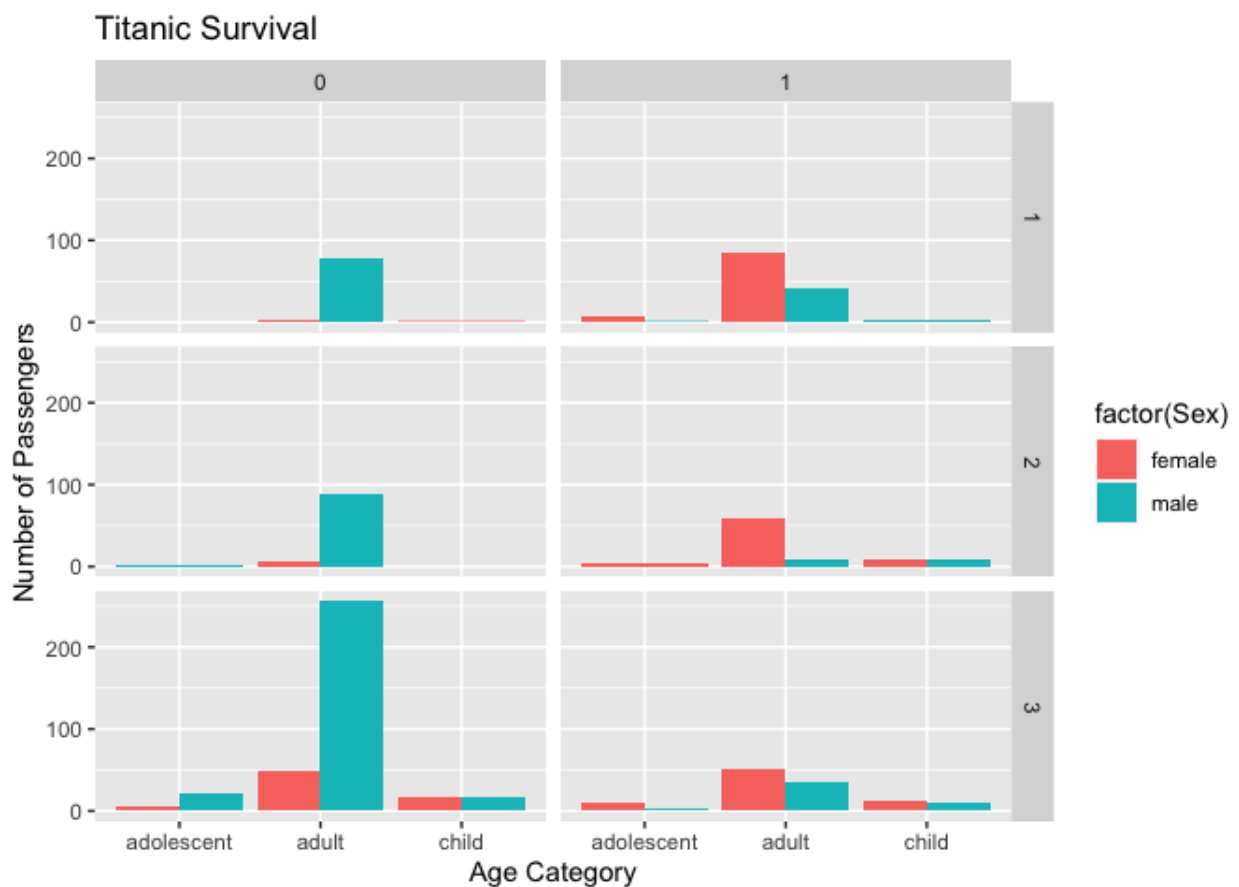
```
library(dplyr)
```

```
##  
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':  
##  
##   filter, lag
```

```
## The following objects are masked from 'package:base':
##
## intersect, setdiff, setequal, union
```

```
titanic %>%
  mutate(age_cat= case_when(Age < 12 ~ "child",
    Age >= 12 & Age < 18 ~ "adolescent",
    Age >= 18 ~ "adult"
  )) %>%
  ggplot() +
  geom_bar(aes(x=age_cat, fill=factor(Sex)),
    position=position_dodge()) +
  facet_grid(Pclass~Survived)+
  labs( y="Number of Passengers", x="Age Category",
    title="Titanic Survival")
```



```
{{Edet}}
```

Want more practice?



Let's use the dataset `mtcars`. According to the help documentation (`?mtcars`), "the data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models)." Each question below will depend on code from the previous question.

Question 1

Let's check out the structure of the data.

{{Sdet}}

Possible Solution{{Esum}}

```
str(mtcars)
```

```
## 'data.frame':   32 obs. of  11 variables:
## $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num  160 160 108 258 360 ...
## $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num  16.5 17 18.6 19.4 17 ...
## $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
## $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

{{Edet}}

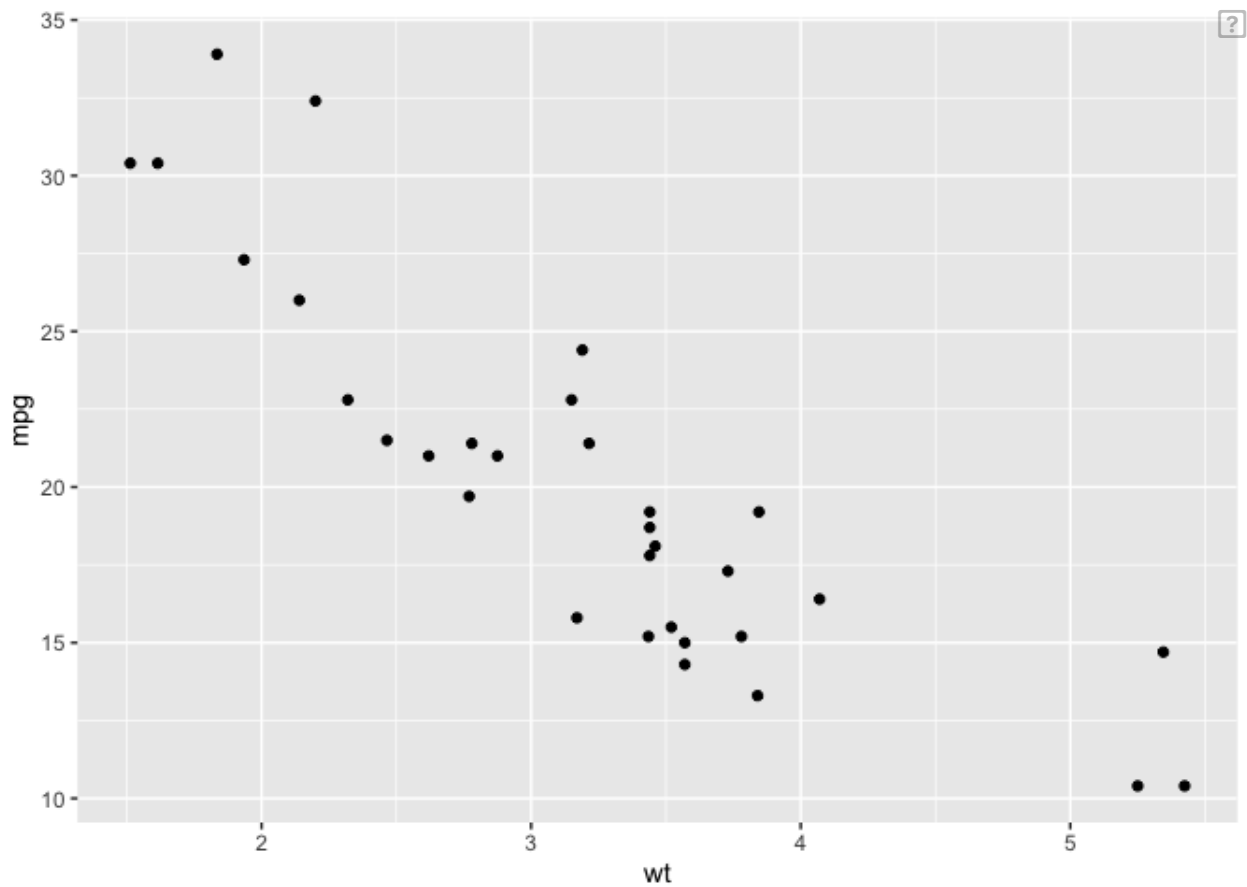
Question 2

How might we plot automobile weight (`wt`) versus miles per gallon (`mpg`).

{{Sdet}}

Possible Solution{{Esum}}

```
ggplot(mtcars, aes(wt, mpg)) +
  geom_point()
```



{{Edet}}

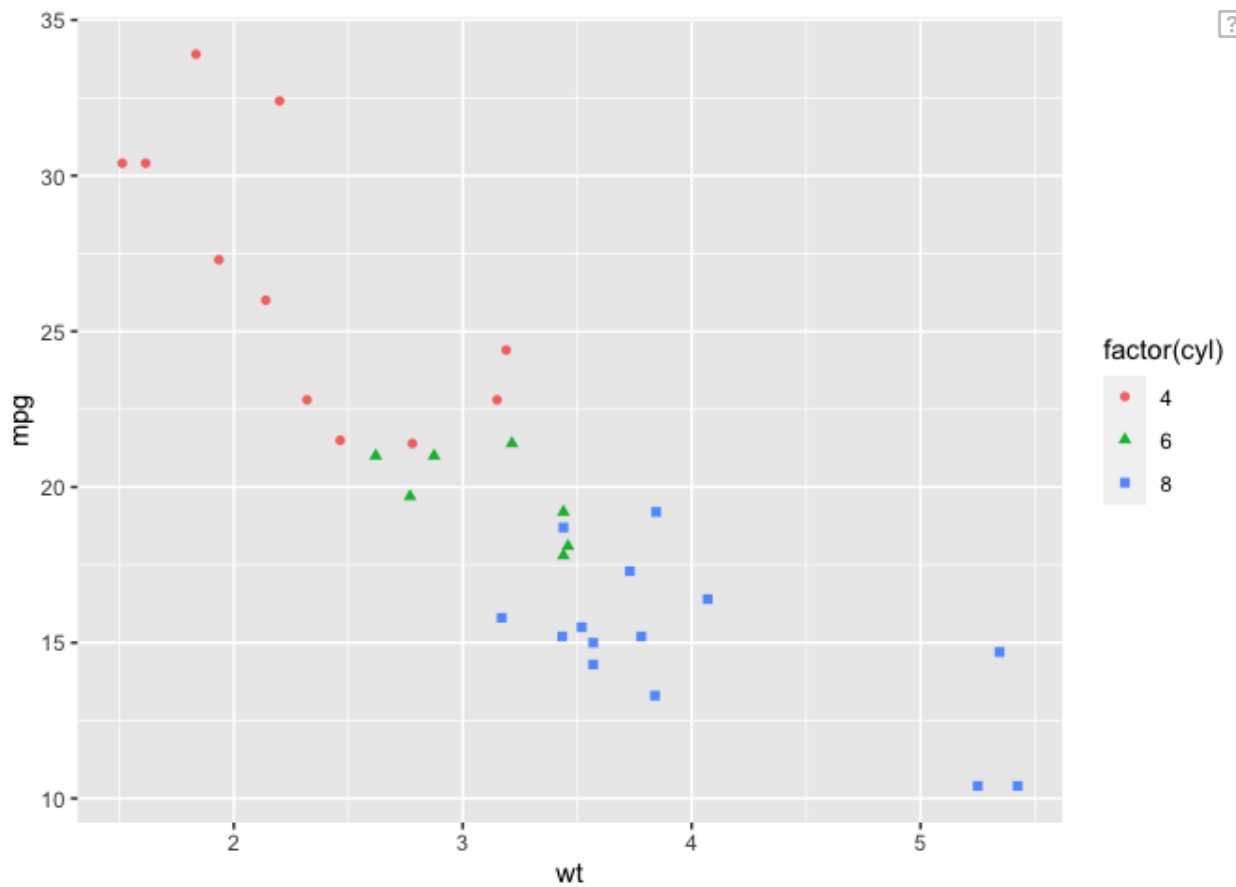
Question 3

What if we want to represent the number of cylinders (cyl) by color and shape?

{{Sdet}}

Possible Solution{{Esum}}

```
ggplot(mtcars, aes(wt, mpg)) +  
  geom_point(aes(color = factor(cyl), shape = factor(cyl)))
```

{{Edet}}

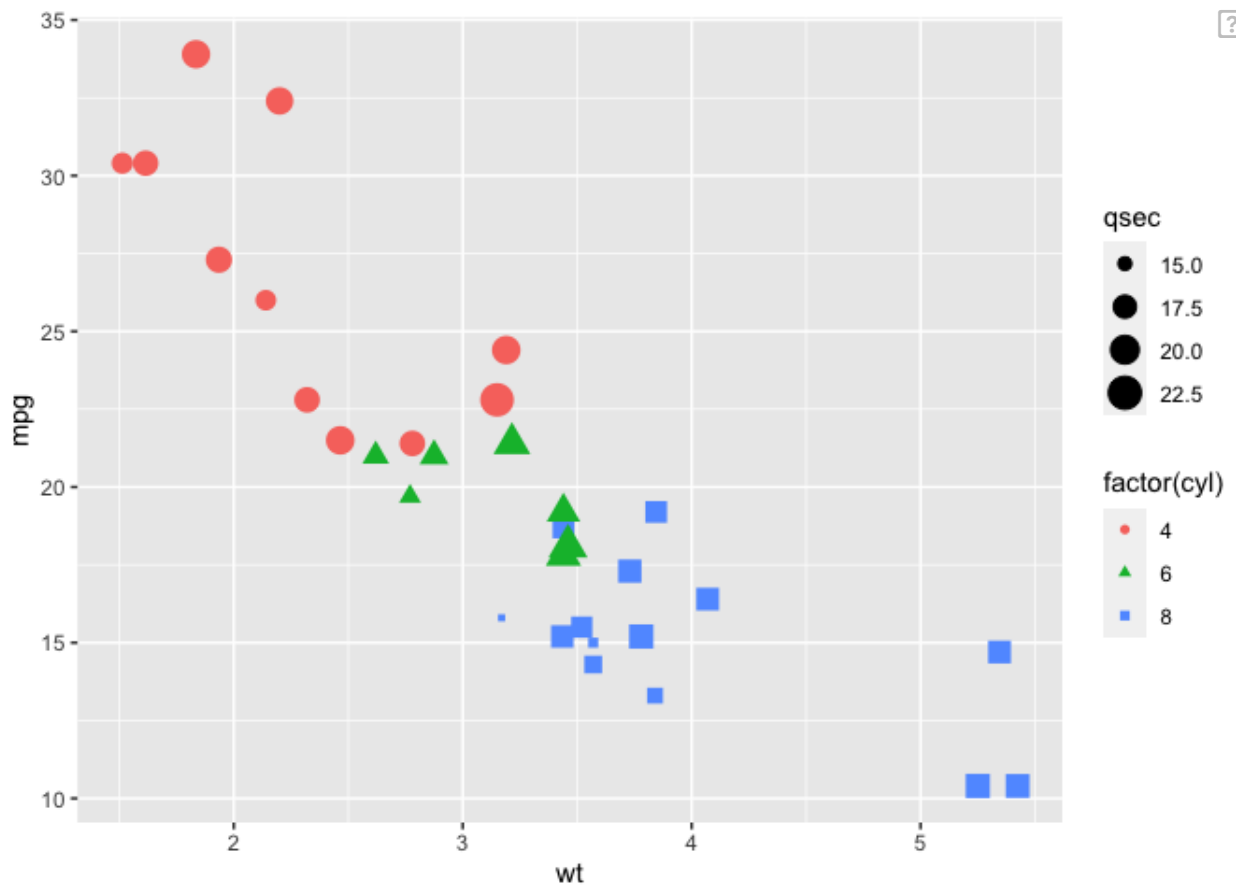
Question 4

Make the size of the points change by the quarter mile time (qsec).

{{Sdet}}

Possible Solution{{Esum}}

```
ggplot(mtcars, aes(wt, mpg)) +  
  geom_point(aes(color = factor(cyl), shape = factor(cyl), size=qsec))
```



{{Edet}}

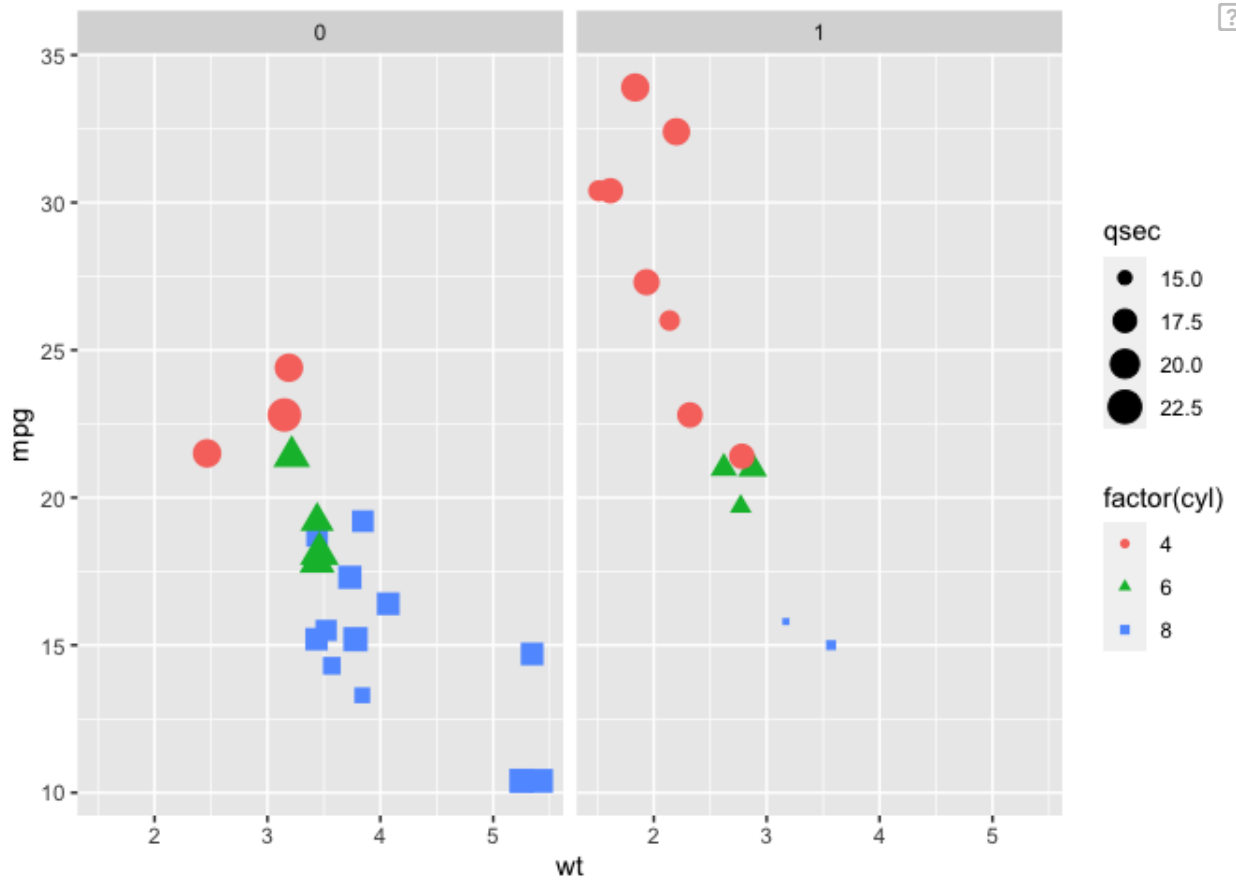
Question 5

Create subplots by transmission (am).

{{Sdet}}

Possible Solution{{Esum}}

```
ggplot(mtcars, aes(wt, mpg)) +
  geom_point(aes(color = factor(cyl), shape = factor(cyl), size=qsec)) +
  facet_wrap(~am)
```



{{Edet}}

Question 6

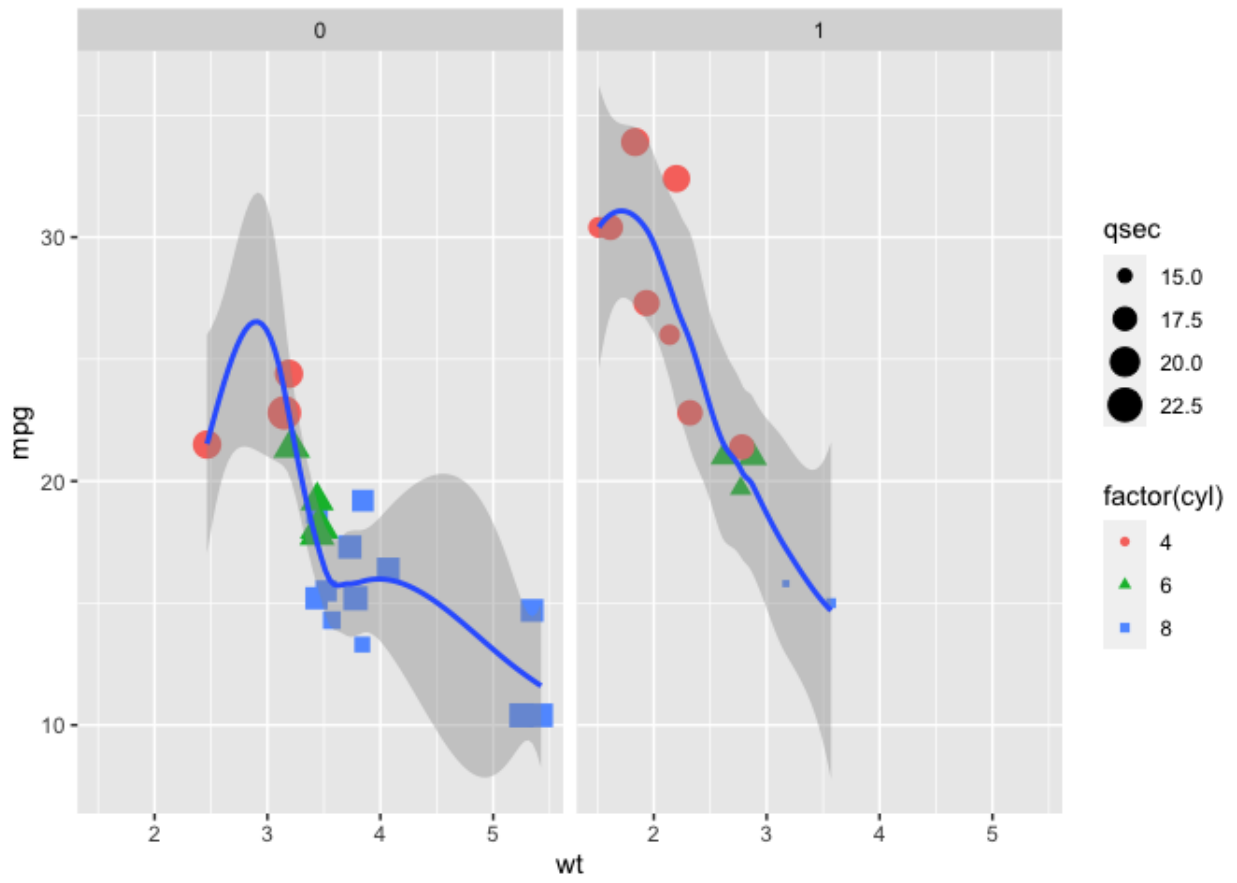
Model the trend using `geom_smooth()`. What is the default method used by `geom_smooth()`?

{{Sdet}}

Possible Solution{{Esum}}

```
ggplot(mtcars, aes(wt, mpg)) +
  geom_point(aes(color = factor(cyl), shape = factor(cyl), size=qsec)) +
  facet_wrap(~am) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



{{Edet}}



Practice plotting using ggplot2: Lesson 3

The following questions synthesize several of the skills you have learned thus far. It may not be immediately apparent how you would go about answering these questions. Remember, the R community is expansive, and there are a number of ways to get help including but not limited to google search. These questions have multiple solutions, but you should try to stick to the tools you have learned to use thus far.

Your mission is to make a publishable figure using the `iris` data set.

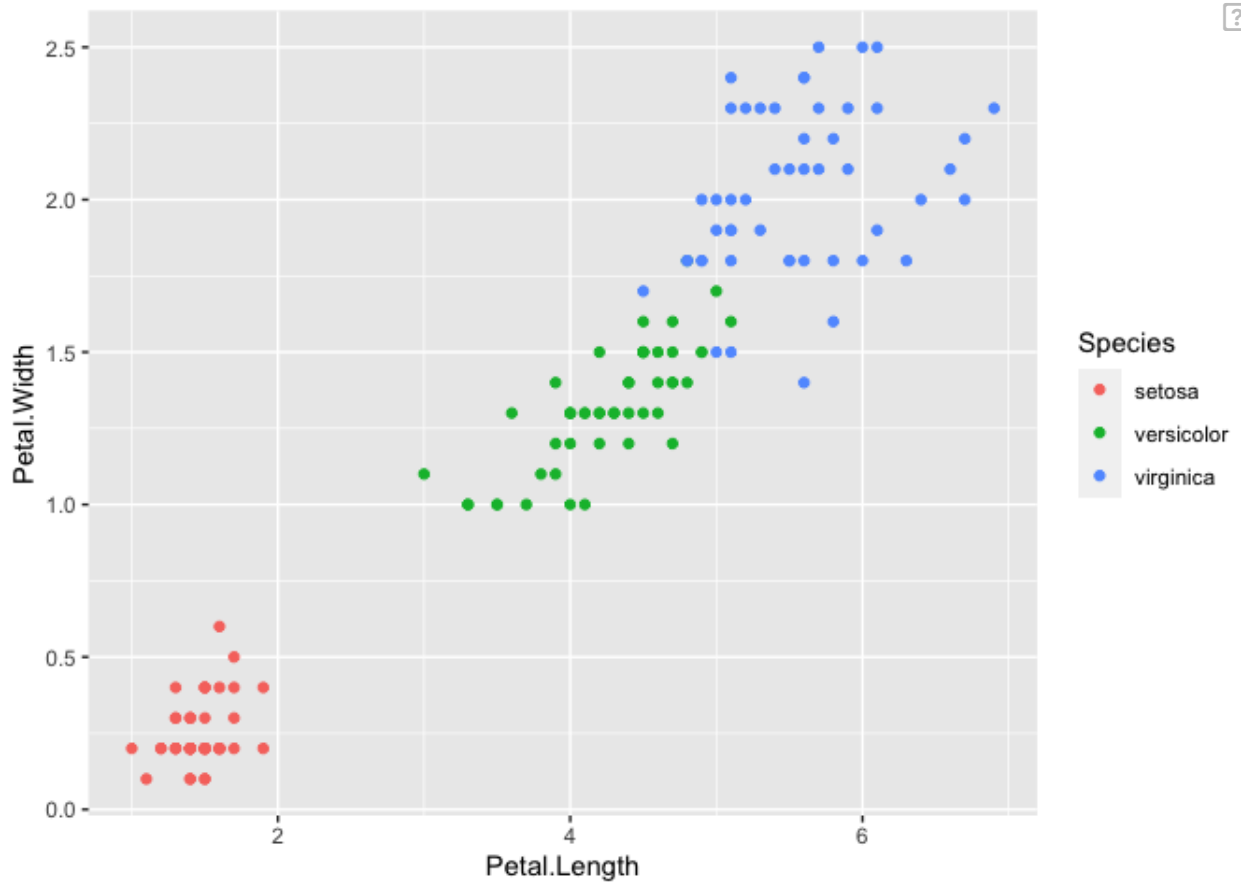
Question 1

Start by plotting `Petal.Length` on the x-axis and `Petal.Width` on the y-axis.

{{Sdet}}

Possible Solution{{Esum}}

```
library(ggplot2)
ggplot(iris)+
  geom_point(aes(Petal.Length,Petal.Width,color=Species))
```



```
{{Edet}}
```

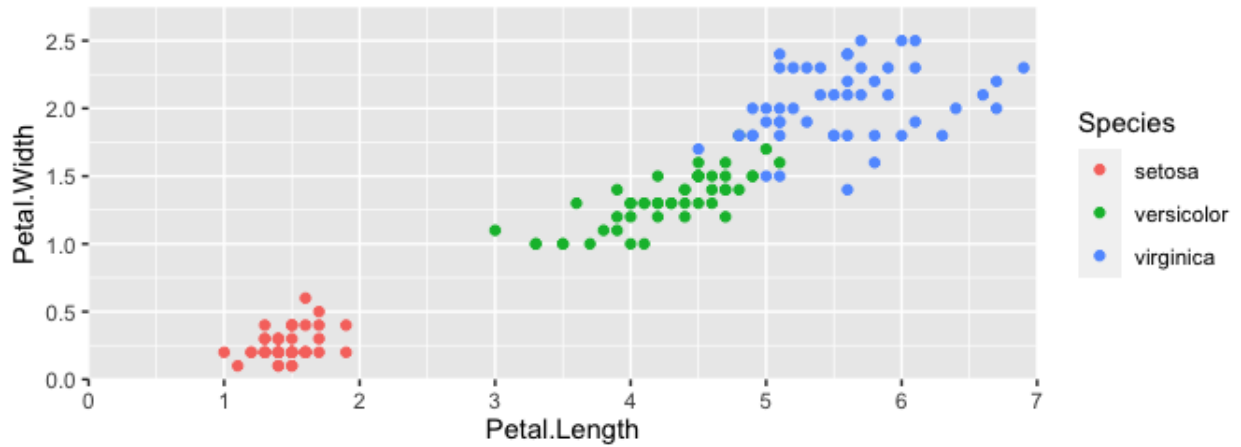
Question 2

Fix the axes so that the dimensions on the x-axis and the y-axis are equal. Both axes should start at 0. Label the axis breaks every 0.5 units on the y-axis and every 1.0 units on the x-axis.

```
{{Sdet}}
```

Possible Solution{{Esum}}

```
ggplot(iris)+
  geom_point(aes(Petal.Length,Petal.Width,color=Species))+
  coord_fixed(ratio=1,ylim=c(0,2.75),xlim=c(0,7),expand=FALSE) +
  scale_y_continuous(breaks=c(0,0.5,1,1.5,2,2.5)) +
  scale_x_continuous(breaks=c(0,1,2,3,4,5,6,7))
```



```
{{Edet}}
```

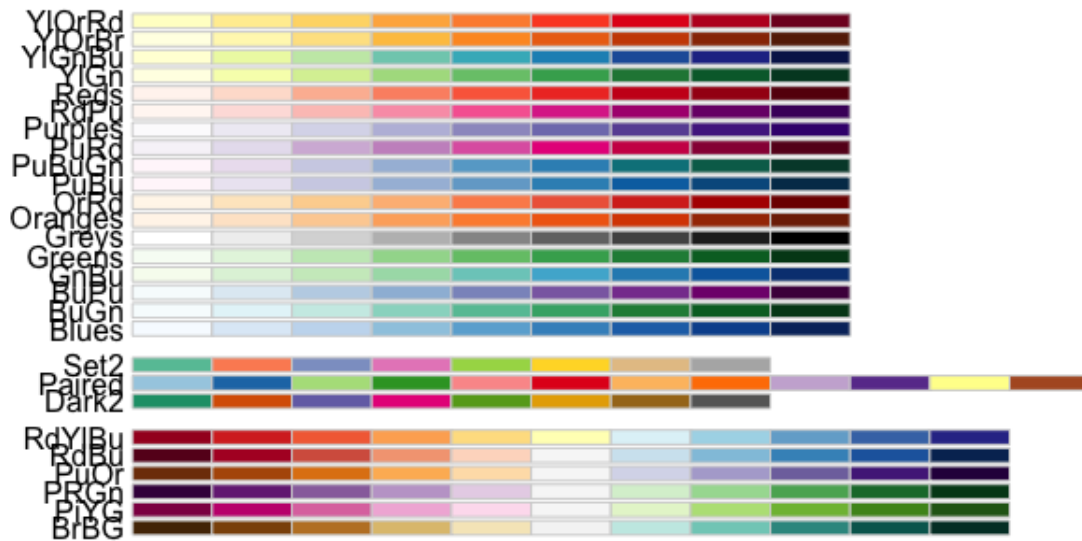
Question 3

Change to color of the points by species to be color blind friendly, and change the legend title to "Iris Species". Label the x and y axis to eliminate the variable names and add unit information.

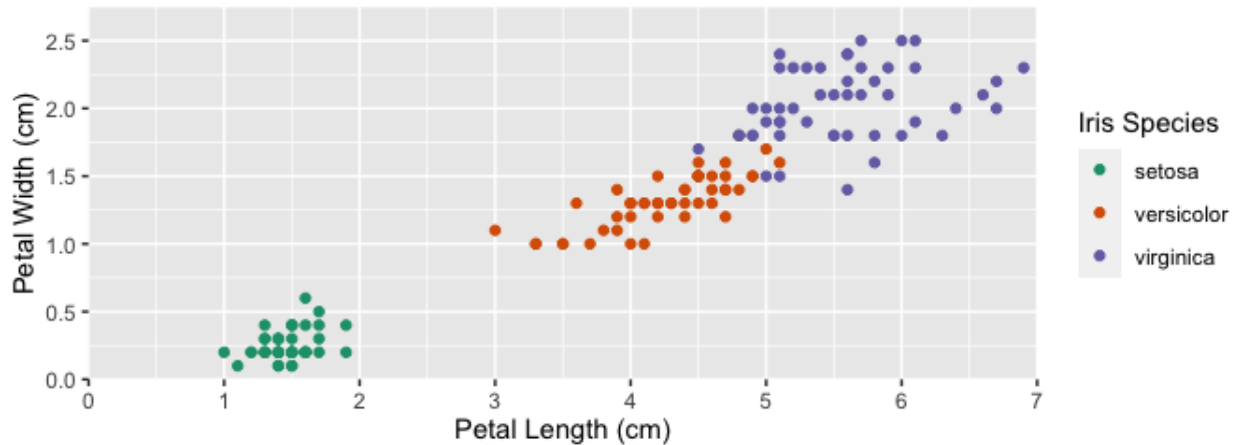
```
{{Sdet}}
```

Possible Solution{{Esum}}

```
#multiple ways to find color blind friendly palettes.
#using color brewer scales
RColorBrewer::display.brewer.all(colorblindFriendly=TRUE)
```



```
ggplot(iris)+
  geom_point(aes(Petal.Length,Petal.Width,color=Species))+
  coord_fixed(ratio=1,ylim=c(0,2.75),xlim=c(0,7),expand=FALSE) +
  scale_y_continuous(breaks=c(0,0.5,1,1.5,2,2.5)) +
  scale_x_continuous(breaks=c(0,1,2,3,4,5,6,7)) +
  scale_color_brewer(palette = "Dark2",name="Iris Species") +
  labs(x="Petal Length (cm)", y= "Petal Width (cm)")
```

{{Edet}}

Question 4

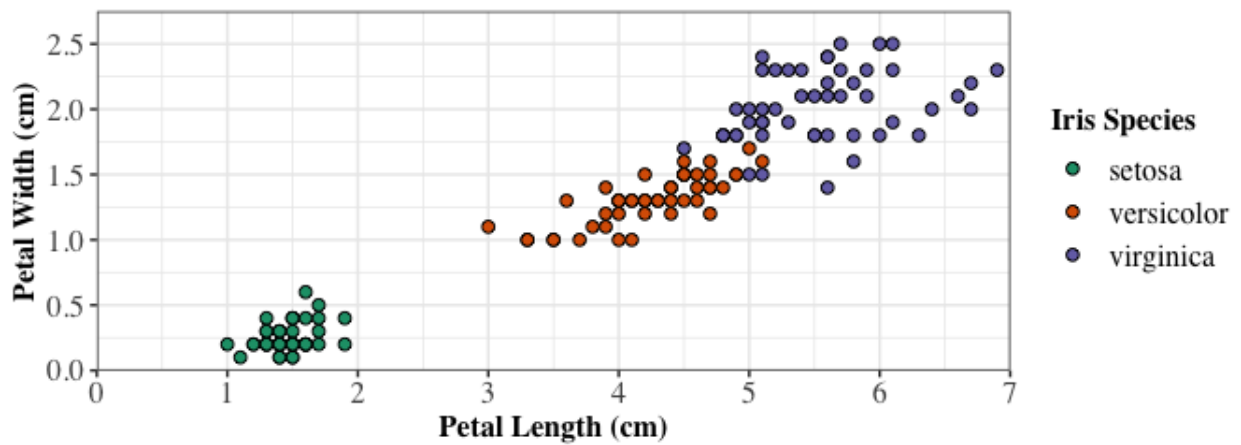
Play with the theme to make this a bit nicer. Change font style to "Times". Change all font sizes to 12 pt font. Bold the legend title and the axes titles. Increase the size of the points on the plot to 2. Bonus: fill the points with color and have a black outline around each point.

{{Sdet}}

Possible Solution{{Esum}}

```
ggplot(iris)+
  geom_point(aes(Petal.Length,Petal.Width,fill=Species),size=2,shape=
  coord_fixed(ratio=1,ylim=c(0,2.75),xlim=c(0,7),expand=FALSE) +
  scale_y_continuous(breaks=c(0,0.5,1,1.5,2,2.5)) +
  scale_x_continuous(breaks=c(0,1,2,3,4,5,6,7)) +
  scale_fill_brewer(palette = "Dark2",name="Iris Species") +
  labs(x="Petal Length (cm)", y = "Petal Width (cm)") +
  theme_bw()+
  theme(axis.text=element_text(family="Times",size=12),
        axis.title=element_text(family="Times",face="bold",size=12),
        legend.text=element_text(family="Times",size=12),
```

```
legend.title = (element_text(family="Times", face="bold", size=12))
```



{{Edet}}

Question 5

Now, save your plot using `ggsave`.

{{Sdet}}

Possible Solution{{Esum}}

```
ggsave("iris.tiff", width=5.5, height=3.5, units="in")
```

{{Edet}}



Lesson 4: Stat Transformations: Bar plots, box plots, and histograms

The following questions will have you explore the `mtcars` dataset through creating plots that were presented in Lesson 4. At the end of these exercises, you should be more comfortable creating plots that convey statistical summary information about data.

Activate packages

```
library(ggplot2)
```

Load the `mtcars` dataset using the code below. This is a dataset that comes with R.

```
data(mtcars)
```

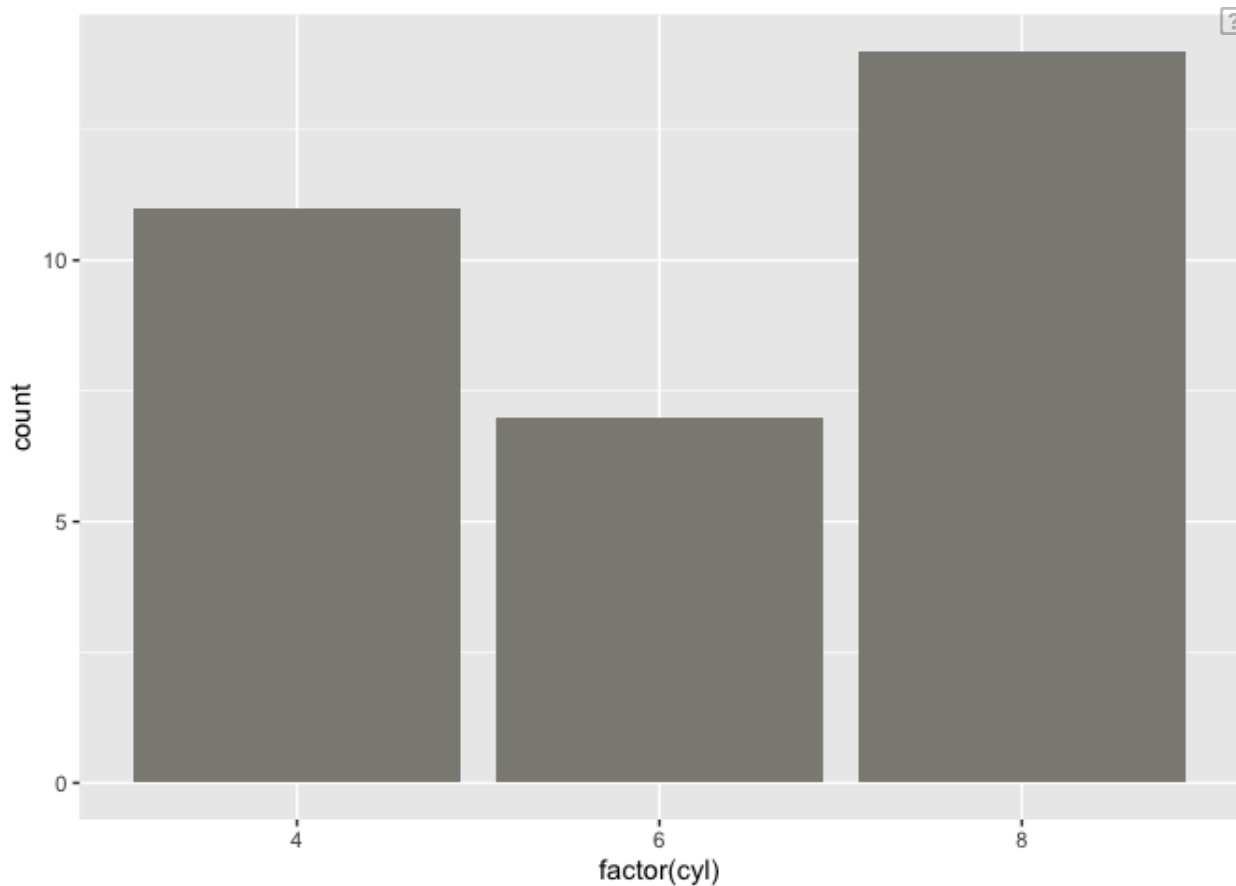
Question 1

How many cars in this dataset have 4, 6, or 8 cylinders (`cyl`)?

{{Sdet}}

Solution{{Esum}}

```
ggplot(mtcars, aes(x=factor(cyl)))+geom_bar(fill="ivory4")
```



```
{{Edet}}
```

Question 2

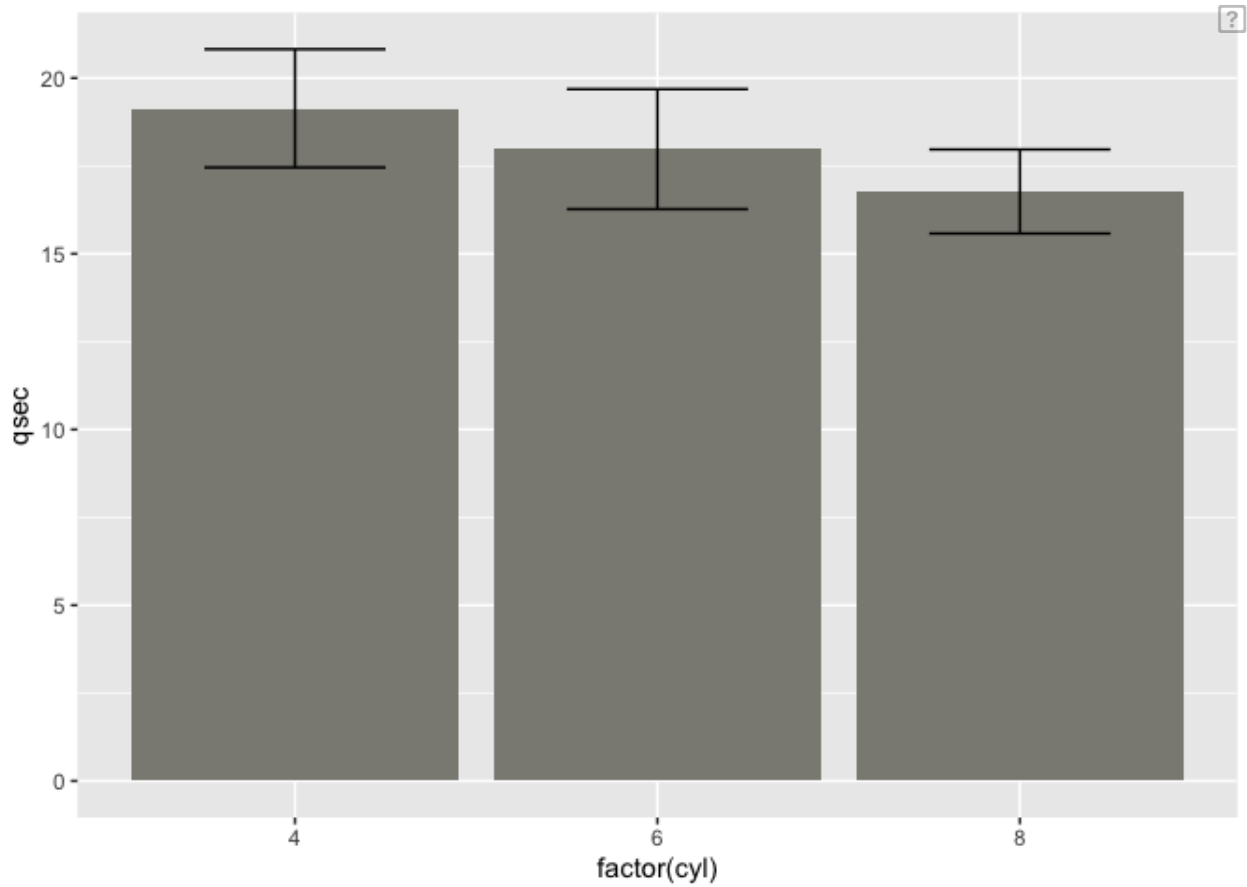
Does the number of cylinders (cyl) that a car has influence its quarter mile time (qsec)?

```
{{Sdet}}
```

```
Solution{{Esum}}
```

```
ggplot(mtcars, aes(x=factor(cyl), y=qsec)) + stat_summary(fun=mean, posit
```

```
## Warning: Ignoring unknown parameters: tion
```



{{Edet}}

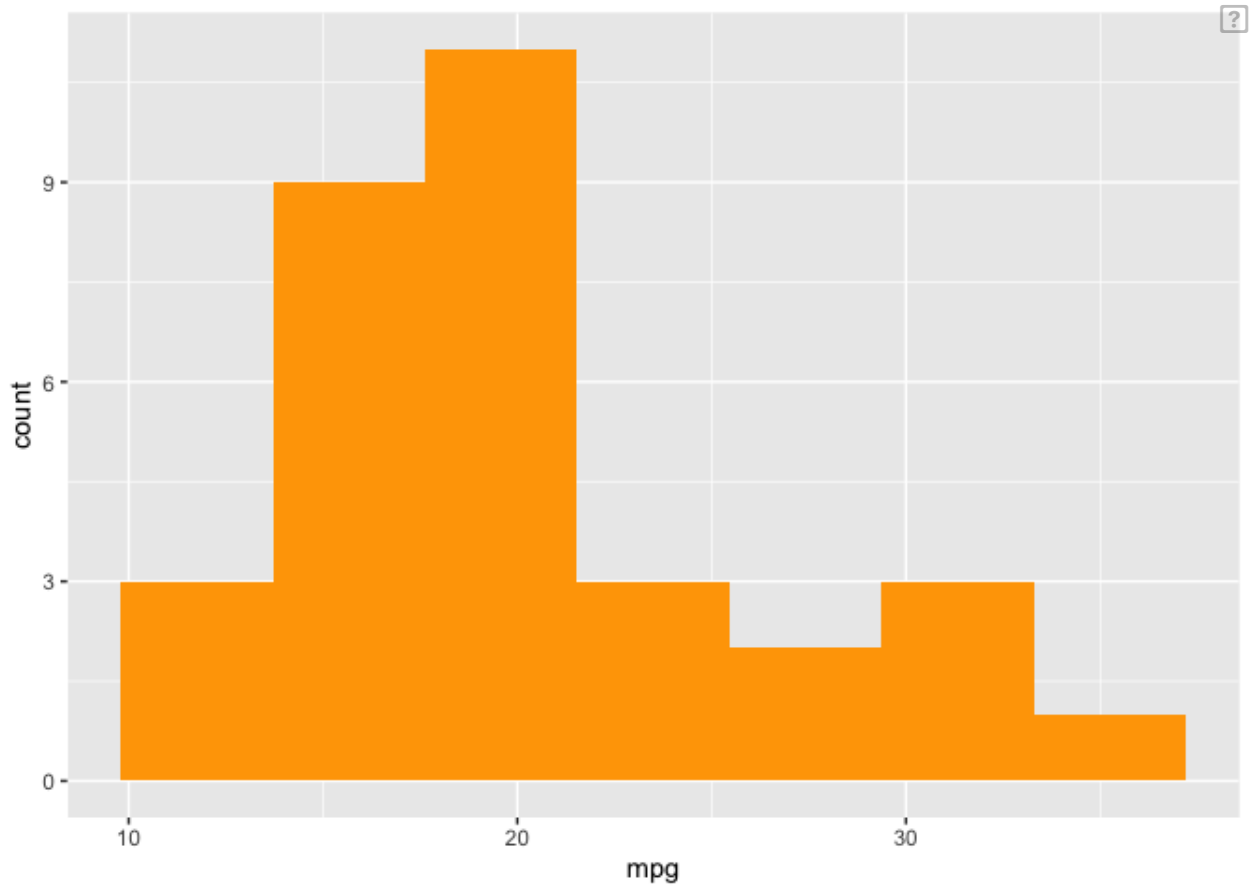
Question 3

What is the distribution of fuel efficiency (mpg)? Use 7 bins for this exercise.

{{Sdet}}

Solution{{Esum}}

```
ggplot(mtcars, aes(x=mpg))+geom_histogram(fill="orange", bins=7)
```



```
{{Edet}}
```

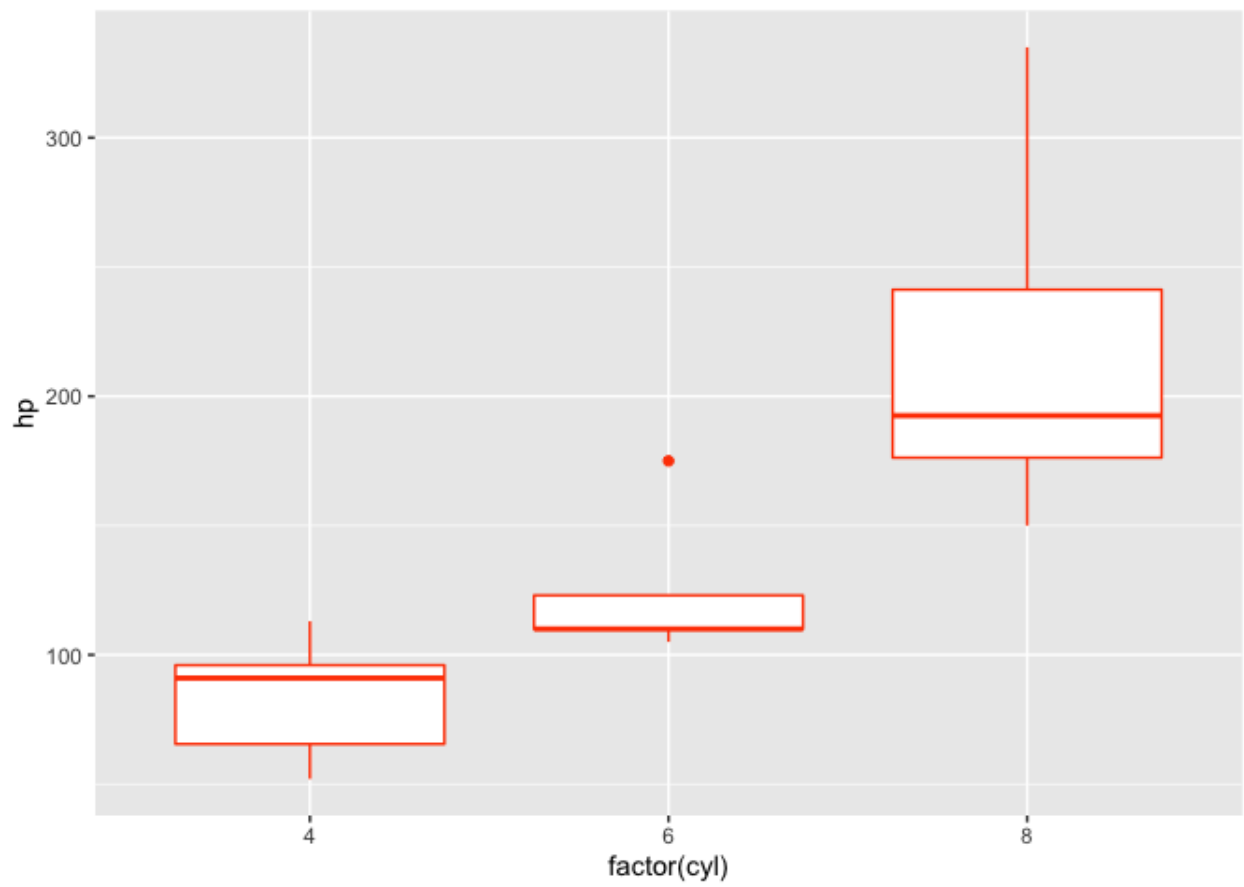
Question 4

Can you create a box plot of horsepower (hp) as a function of the number of cylinders (cyl) a car has?

```
{{Sdet}}
```

```
Solution{{Esum}}
```

```
ggplot(mtcars, aes(x=factor(cyl), y=hp))+geom_boxplot(colour="orangerec
```



{{Edet}}



Lesson5: Visualizing clusters with heatmap and dendrogram

The following questions will help you gain more confidence in exploring data through heatmap. We will work with a subset of the [Human Brain Reference \(HBR\)](#) and [Universal Human Reference \(UHR\) RNA sequencing dataset](#) (https://rnabio.org/module-01-inputs/0001/05/01/RNAseq_Data/) and use the heatmap to

- Visualize gene expression
- Determine whether subsets of genes can help us differentiate between the HBR and UHR samples

Load necessary packages

```
library(pheatmap)
library(tidyverse)
```

```
## — Attaching packages ————— tidy
## ✓ ggplot2 3.3.6      ✓ purrr   0.3.4
## ✓ tibble  3.1.8      ✓ dplyr   1.0.9
## ✓ tidyr   1.2.0      ✓ stringr 1.4.0
## ✓ readr   2.1.2      ✓ forcats 0.5.1
## — Conflicts ————— tidyverse_
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()     masks stats::lag()
```

Question 1

Could you import the `hbr_uhr_normalized_counts.csv` file into your workspace?

```
{{Sdet}}
```

```
Solution{{Esum}}
```

```
hbr_uhr_normalized_counts <- read.csv("../data/hbr_uhr_normalized_coui
```

```
{{Edet}}
```


Question 2



Explore this gene expression dataset a bit. How many samples (columns) and genes (row names) does this dataset have?

{{Sdet}}

Solution{{Esum}}

This dataset contains 6 samples (

- HBR_1.bam
- HBR_2.bam
- HBR_3.bam
- UHR_1.bam
- UHR_2.bam
- UHR_3.bam

The samples with names starting with HBR are from the Human Brain Reference (HBR) and those with names starting with UHR are from the Universal Human Reference (UHR). Remember this for a later questions.

hbr_uhr_normalized_counts

##	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam	U
## SULT4A1	375.0	343.6	339.4	3.5	6.9	
## MPPED1	157.8	158.4	162.6	0.7	3.0	
## PRAME	0.0	0.0	0.0	568.9	467.3	
## IGLC2	0.0	0.0	0.0	488.6	498.0	
## IGLC3	0.0	0.0	0.0	809.7	313.8	
## CDC45	2.6	1.0	0.0	155.0	152.5	
## CLDN5	77.6	88.5	67.2	1.4	2.0	
## PCAT14	0.0	0.0	1.2	139.8	154.4	
## RP5-1119A7.17	53.0	57.6	51.9	0.0	0.0	
## MY018B	0.0	0.0	0.0	59.5	84.2	
## RP3-323A16.1	0.0	0.0	1.2	51.9	76.2	
## CACNG2	42.7	35.0	56.6	0.0	1.0	

{{Edet}}

Question 3

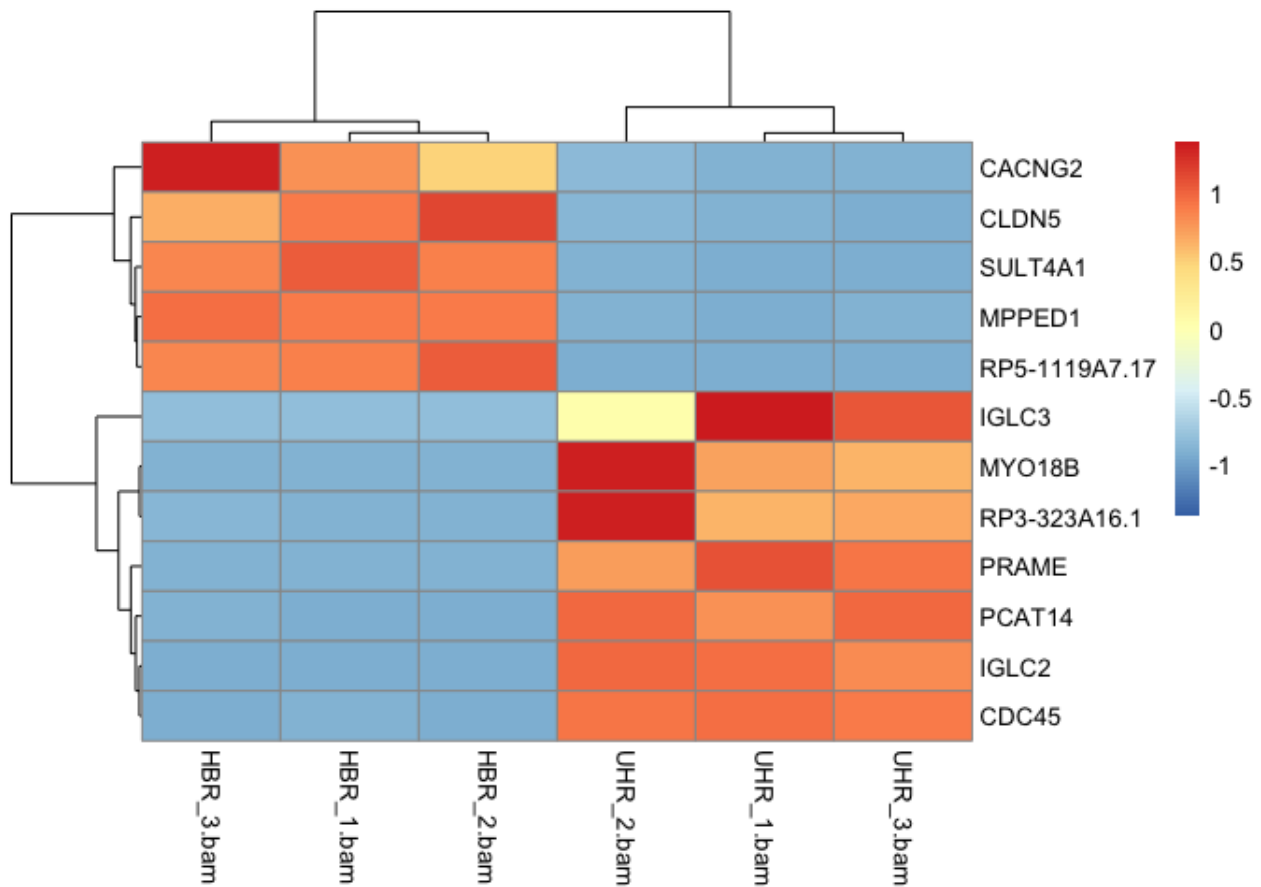
Create a heatmap for to visualize gene expression for this dataset.

{{Sdet}}

```
Solution{{Esum}}
```



```
pheatmap(hbr_uhr_normalized_counts,scale="row")
```



```
{{Edet}}
```

Question 4

Create a data frame called `annotation_df` that contains the sample and treatment group information that we will add to the legend for this heatmap.

```
{{Sdet}}
```

```
Solution{{Esum}}
```

```
annotation_df <- data.frame(sample=c("HBR_1.bam", "HBR_2.bam", "HBR_3.l",
annotation_df
```

```
##          treatment
## HBR_1.bam        HBR
## HBR_2.bam        HBR
## HBR_3.bam        HBR
```

```
## UHR_1.bam      UHR
## UHR_2.bam      UHR
## UHR_3.bam      UHR
```

```
{{Edet}}
```

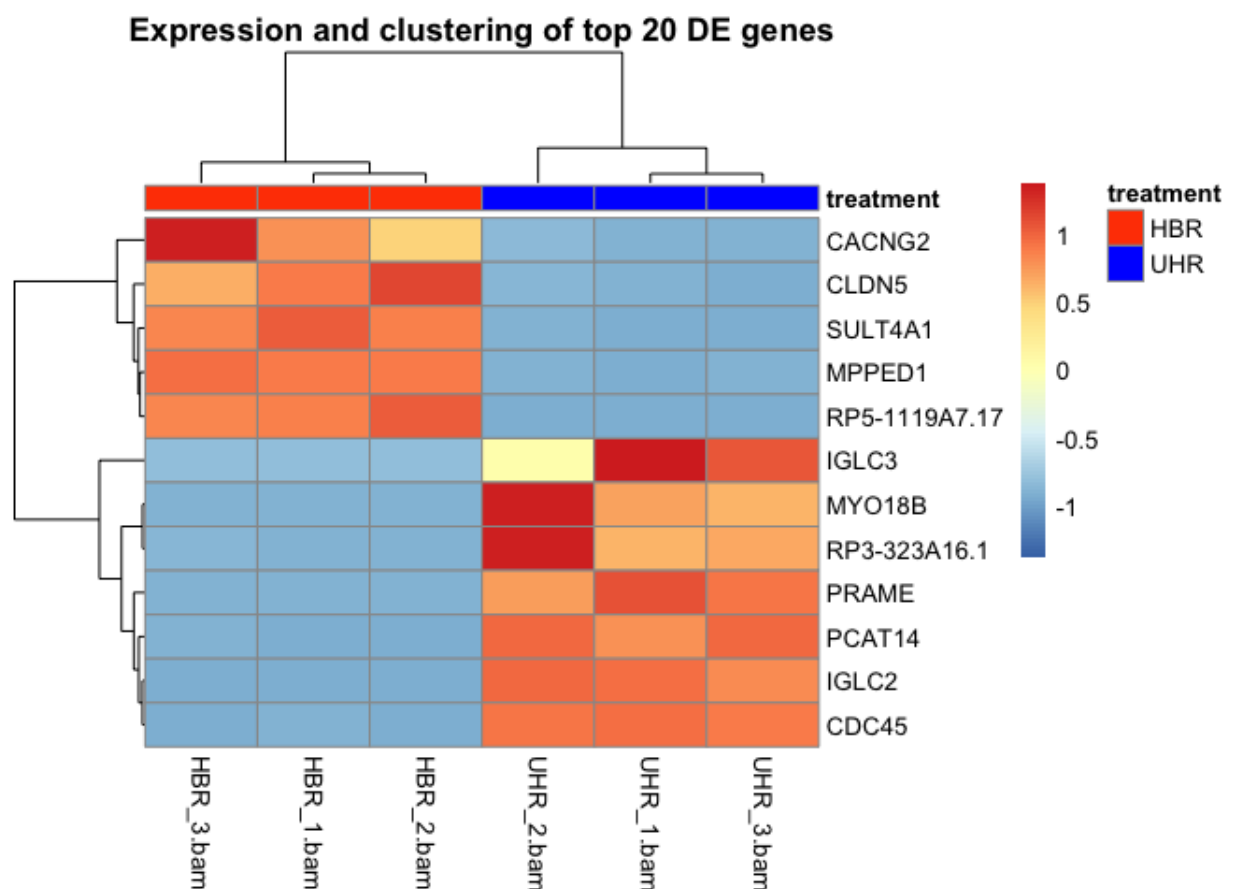
Question 5

Add the annotations for the legend and color the HBR samples orangered and the UHR samples blue. Also, add a title to the heatmap.

```
{{Sdet}}
```

```
Solution{{Esum}}
```

```
heatmap(hbr_uhr_normalized_counts,scale="row", annotation_col =annot
        annotation_colors =list(treatment=c(HBR="orangered",UHR="bli
        main="Expression and clustering of top 12 DE genes")
```



```
{{Edet}}
```

Additional Resources

For Further Reading

Getting started with R

1. Hands on Programming with R (<https://rstudio-education.github.io/hopr/index.html>)
2. R for Data Science (R4DS) (<https://r4ds.had.co.nz/index.html>)
3. BTEP R Introductory Series (<https://bioinformatics.ccr.cancer.gov/docs/rintro/index.html>)
4. Base R cheat sheet
5. RStudio cheat sheet

General help with ggplot2

1. ggplot2 cheatsheet
2. R Graph Gallery (<https://www.r-graph-gallery.com/>)
3. The R Graphics Cookbook (<https://r-graphics.org/recipe-quick-bar>)
4. R4DS (<https://r4ds.had.co.nz/data-visualisation.html>)

Troubleshooting error messages

1. Epidemiologist R Handbook (47 Common errors) (<https://epirhandbook.com/en/common-errors.html>)
2. R for Graduate Students (Troubleshooting Error Messages) (https://bookdown.org/yih_huynh/Guide-to-R-Book/trouble.html)

Other Resources

1. Helpful search engine for R: rseek (<https://rseek.org/>)

**DNAnexus (Class
Registrants Only)**

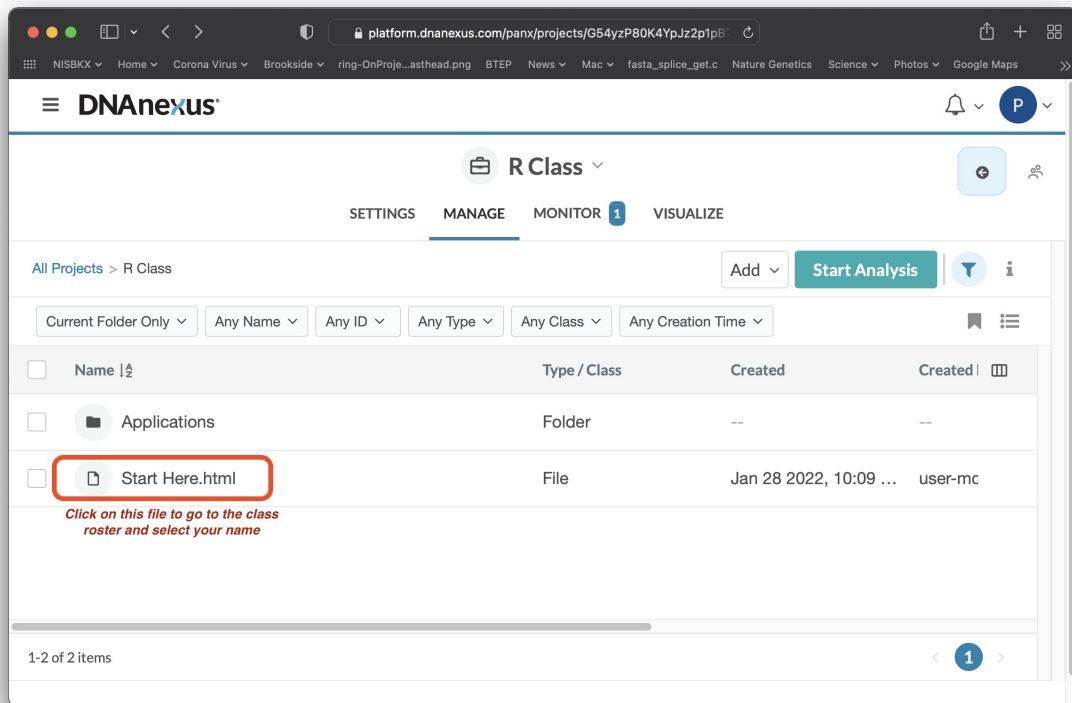
Navigating DNAnexus

DNAnexus is a Cloud-based platform for NextGen Sequence analysis for which CCR has a "site-license". For this class we are using the platform to provide a uniform, stable, preinstalled interface for R training. This interface makes use of the Web version of R-studio. In addition to the R-studio interface this process also integrates the course-notes for the class in one window.

The following instructions should be followed when using this resource during formal class time. For using this resource outside class times see the document entitled "[DNAnexus Basics](#)".

Instruction for using DNAnexus for the Intro to R class

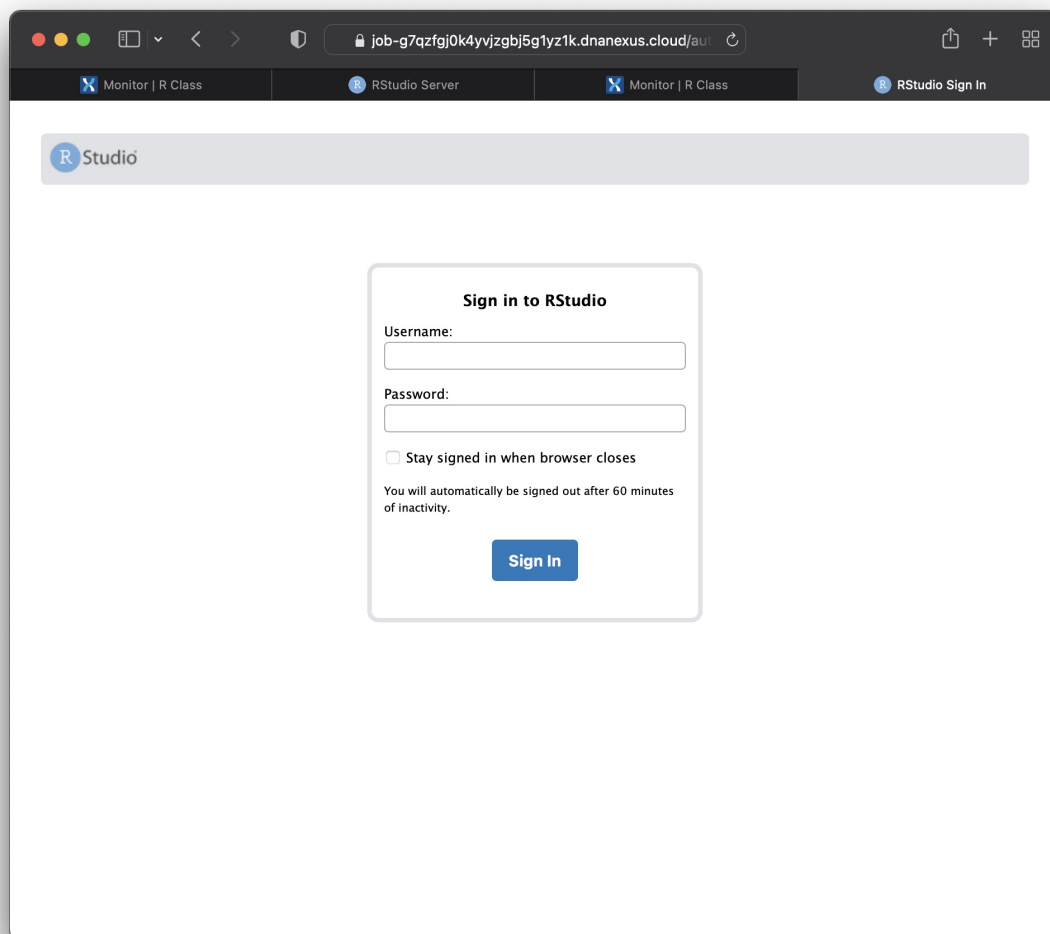
1. **Getting a DNAnexus account** - every student should go to the main DNAnexus web page (<https://dnanexus.com/>) and apply for a "free account". The BTEP staff will associate each account with the NCI/CCR paid account prior to the first class.
2. **Logging into DNAnexus account** - Prior to the class each student should log into their account, and navigate to the R Class project (`DataViz_Apr_2023`).
3. **Starting R** - Starting 30 mins before each class there will be a file labelled "**Start_Here.html**" at the top level of the project. Select this file by clicking on it, and then find your name on the list (arranged alphabetically by first name) and click on it. **Note: if there is more than one Start_Here.html file, you will need to select the file that has a range of letters in which your first name would be included.** For example, if there were two files, `Start_Here_A_J.html` and `Start_Here_K_Z.html`, and my name was "Alex", I would select `Start_Here_A_J.html`. The names of these files will vary based on the total number of students in the class.



The screenshot shows the DNAnexus web interface. At the top, the URL is `platform.dnanexus.com/panx/projects/G54yzP80K4YpJz2p1pB...`. The main header displays the DNAnexus logo and a user profile icon. Below the header, the current project is identified as 'R Class'. A navigation bar includes 'SETTINGS', 'MANAGE', 'MONITOR' (with a notification badge), and 'VISUALIZE'. The 'MANAGE' tab is active, showing a file list for 'All Projects > R Class'. The list includes a folder named 'Applications' and a file named 'Start Here.html', which is highlighted with a red box. Below the file list, a red-bordered box contains the text: 'Click on this file to go to the class roster and select your name'. The bottom of the interface shows '1-2 of 2 items' and a pagination control with the number '1'.

<input type="checkbox"/>	Name ↕	Type / Class	Created	Created
<input type="checkbox"/>	Applications	Folder	--	--
<input type="checkbox"/>	Start Here.html	File	Jan 28 2022, 10:09 ...	user-mc

Once you select your name from the correct file, a window with the RStudio login page will open.



Log in using the username "rstudio" and the password "rstudio". At this point you will be presented with the RStudio main interface (shown below).

4. **Splitting the window** - If you wish to integrate the class notes into the same window as the R-Studio interface, click on the file "**Hsplit.html**" or "**Vsplit.html**" (found in the lower right hand segment) and select the "**View in Web Browser**" option from the pop-up menu. This will add the class notes to the top portion of the browser window. There is a horizontal or vertical bar separating the class notes window from the RStudio interface, and this bar can be dragged up and down or right to left, depending on which file you selected (Hsplit.html vs Vsplit.html), to change the size of the window dedicated to each function.

The screenshot shows a web browser window displaying a course overview page for the "R Introductory Series 2022". The page has a dark header with the course title and a search bar. Below the header is a "Course Overview" section. An RStudio interface is overlaid on the bottom half of the browser window. A red double-headed arrow points to the horizontal bar separating the course overview from the RStudio interface, with the text "Drag bar up or down to resize the windows" written above it. The RStudio interface shows the console with R version information and a file browser on the right side.

Drag bar up or down to resize the windows

Course Overview

Table of contents

R Introductory Series 2022

Search

File Edit Code View Plots Session Build Debug Profile Tools Help

Environment History Connections Tutorial

Files Plots Packages Help Viewer

Name	Size	Modified
diffexp_results_edger_airways.rds	34.5 MB	Jan 28, 2022, 9:
diffexp_results_edger_airways.txt	2.1 MB	Jan 28, 2022, 9:
fitlowabund_scaledcounts_airw...	34.5 MB	Jan 28, 2022, 9:
fitlowabund_scaledcounts_airw...	23.4 MB	Jan 28, 2022, 9:
HISTORY.fitzgepe	905 B	Jan 28, 2022, 9:
Rlearning		
site		
Split_Window.html	260 B	Jan 21, 2022, 6:

DNAnexus Outside of Class

Setting up the R-Studio environment outside of Class hours

These instructions should be followed if you are setting up the R-Studio Web environment outside the normal class hours. For instructions about using the resource during class hours follow the instructions found in the document "[Navigating DNAnexus](#)".

Log into DNAnexus

Each student should log into their account, and navigate to the R Class project (DataViz_Apr_2023). For these instructions to make sense, you should use the "New Version" of DNAnexus. If you are instead using the "Classic Version", your screen will look like this:

The screenshot shows the DNAnexus interface. At the top, there is a navigation bar with 'DNAnexus' and menu items 'PROJECTS', 'TOOLS', and 'HELP'. Below this is a banner with the text 'Class Project ID: this will be class dependent.' and 'We are introducing a new look for this page! This current design will be phased out.' A red box highlights the 'Start Using New Version' button in the top right corner. Below the banner, the project name 'R_Class_Jan_2023' is displayed, along with tabs for 'SETTINGS', 'MANAGE', 'MONITOR', and 'VISUALIZE'. A red box highlights the 'Start Using New Version' button in the top right corner. Below the tabs are four green buttons: '+ Add Data', 'New Folder', 'New Workflow', and 'Start Analysis'. Below these buttons are search filters for 'SEARCH SCOPE', 'NAME', 'ID', and 'MODIFIED'. Below the filters is a table with columns for 'Name', 'Type', 'Size', and 'Created'. The table contains the following data:

Name	Type	Size	Created
Applications	Folder		
classroom_docs	Folder		
1. NIH Warning -Government Managed ...	File	10.11 KiB	Al
2.NIH Warning.html	File	9.91 KiB	Al
Practicing R - Outside of Class App	Applet	1.14 GiB	Ja

You need to select "Start Using New Version".

Note

Your class project ID WILL NOT be the same as the project ID in the picture.

Your screen should now look like the following:

DNAnexus PROJECTS TOOLS HELP

R_Class_Jan_2023 SETTINGS MANAGE MONITOR VISUALIZE

R_Class_Jan_2023

Applications
classroom_docs

All Projects > R_Class_Jan_2023

Current Folder Only Any Name Any ID Any Type Any Class

<input type="checkbox"/>	Name	Type / Class	Created	Created By
<input type="checkbox"/>	Applications	Folder	--	--
<input type="checkbox"/>	classroom_docs	Folder	--	--
<input type="checkbox"/>	1. NIH Warning -Government...	File	Aug 09 2022, 8:41 AM	user-pcf
<input type="checkbox"/>	2.NIH Warning.html	File	Aug 09 2022, 8:47 AM	user-pcf
<input type="checkbox"/>	Practicing R - Outside of Cla...	Applet	Jan 25 2023, 3:14 PM	user-mcintoshc

Back to Classic Version

Add Stop

If you are in the right version, you will see this!

Select the "Practicing R - Outside of Class" applet.

No Item
Select an item

To Start the Application

To start the application, select "Practicing R - Outside of Class App".

DNAnexus PROJECTS TOOLS HELP

R_Class_Jan_2023 SETTINGS MANAGE MONITOR VISUALIZE

R_Class_Jan_2023

Applications
classroom_docs

All Projects > R_Class_Jan_2023

Current Folder Only Any Name Any ID Any Type Any Class

<input type="checkbox"/>	Name	Type / Class	Created	Created By
<input type="checkbox"/>	Applications	Folder	--	--
<input type="checkbox"/>	classroom_docs	Folder	--	--
<input type="checkbox"/>	1. NIH Warning -Government...	File	Aug 09 2022, 8:41 AM	user-pcf
<input type="checkbox"/>	2.NIH Warning.html	File	Aug 09 2022, 8:47 AM	user-pcf
<input type="checkbox"/>	Practicing R - Outside of Cla...	Applet	Jan 25 2023, 3:14 PM	user-mcintoshc

Back to Classic Version

Add Stop

If you are in the right version, you will see this!

Select the "Practicing R - Outside of Class" applet.

No Item
Select an item

Then follow these steps:

- Step 1: Under Analysis Settings, enter your name in the Execution Name field.
- Step 2: Change Priority from "Normal" to "High".

home_rstudio_upload *.tar.gz

RStudio Session App v3.0.1

Step 1: Enter your name in "Execution Name"

Step 2: Change "Priority" from "Normal" to "High"

Run Analysis RStudio Session App v3.0.1 Start Analysis

ANALYSIS SETTINGS ANALYSIS INPUTS APPLET SETTINGS

Execution Name
Alex Emmons

Execute in Project
R_Class_Jan_2023 (project-GP4g32j0yftzV1JgG1q0YBqp0)

Execution Output Folder
/

Priority [?]
High

Spending Limit [?]
No Limit

ADVANCED

Allow SSH Access OFF
Allow SSH access to enable developer debugging features. To use these options, you need to first set up SSH access via dx toolkit. [More about SSH Access.](#)

- Step 3: Select the green button in the upper right labeled "Start Analysis".

home_rstudio_upload *.tar.gz

RStudio Session App v3.0.1

Step 3: Select "Start Analysis".

Run Analysis RStudio Session App v3.0.1 Start Analysis

ANALYSIS SETTINGS ANALYSIS INPUTS APPLET SETTINGS

Execution Name
Alex Emmons

Execute in Project
R_Class_Jan_2023 (project-GP4g32j0yftzV1JgG1q0YBqp0)

Execution Output Folder
/

Priority [?]
High

Spending Limit [?]
No Limit

ADVANCED

Allow SSH Access OFF
Allow SSH access to enable developer debugging features. To use these options, you need to first set up SSH access via dx toolkit. [More about SSH Access.](#)

- Step 4: Navigate to the "Monitor" tab to check the status of the job.

DNAnexus PROJECTS TOOLS HELP

R_Class_Jan_2023 SETTINGS MANAGE **MONITOR** VISUALIZE

Alex Emmons WAITING

Launch as new Job View Log View all Inputs/Outputs View Info Terminate

EXECUTION ID job-GP8kq7Q0yFzZ1P93XqQzVp1v PRIORITY high LAUNCHED ON 01/25/2023 4:22 pm RUNNING FOR -- EXECUTABLE RStudio Session ...

04:22:38PM	04:23:02PM	04:23:18PM	04:23:34PM
------------	------------	------------	------------

Alex Emmons

Show Inputs/Outputs

Open the R-Studio Web interface

Step 5: Wait approximately 5 minutes and then select the link under "Worker URL". The link will look similar to this: "<https://job-g7v6z280k4yp6qg4pbkz1gvb.dnanexus.cloud>").

DNAnexus PROJECTS TOOLS HELP

R_Class_Jan_2023 SETTINGS MANAGE **MONITOR** VISUALIZE

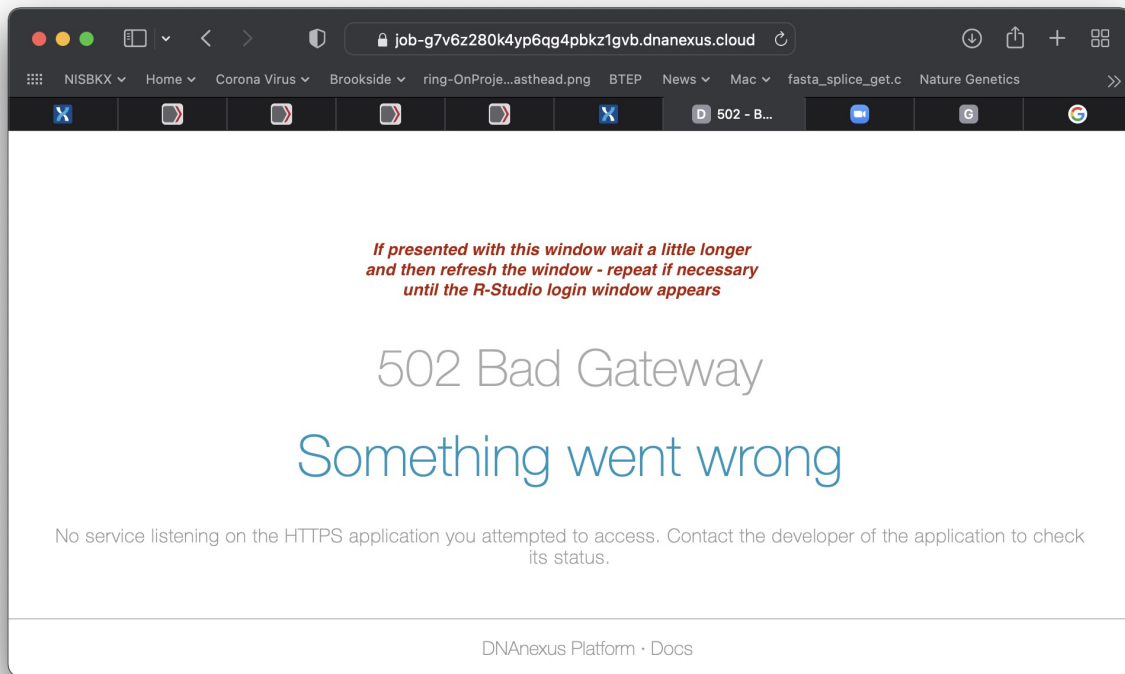
Terminate All Analyses

SEARCH SCOPE Root executions only STATE Any 307 recent jobs NAME Any ID Any CREATED Any LAUNCHED BY Any

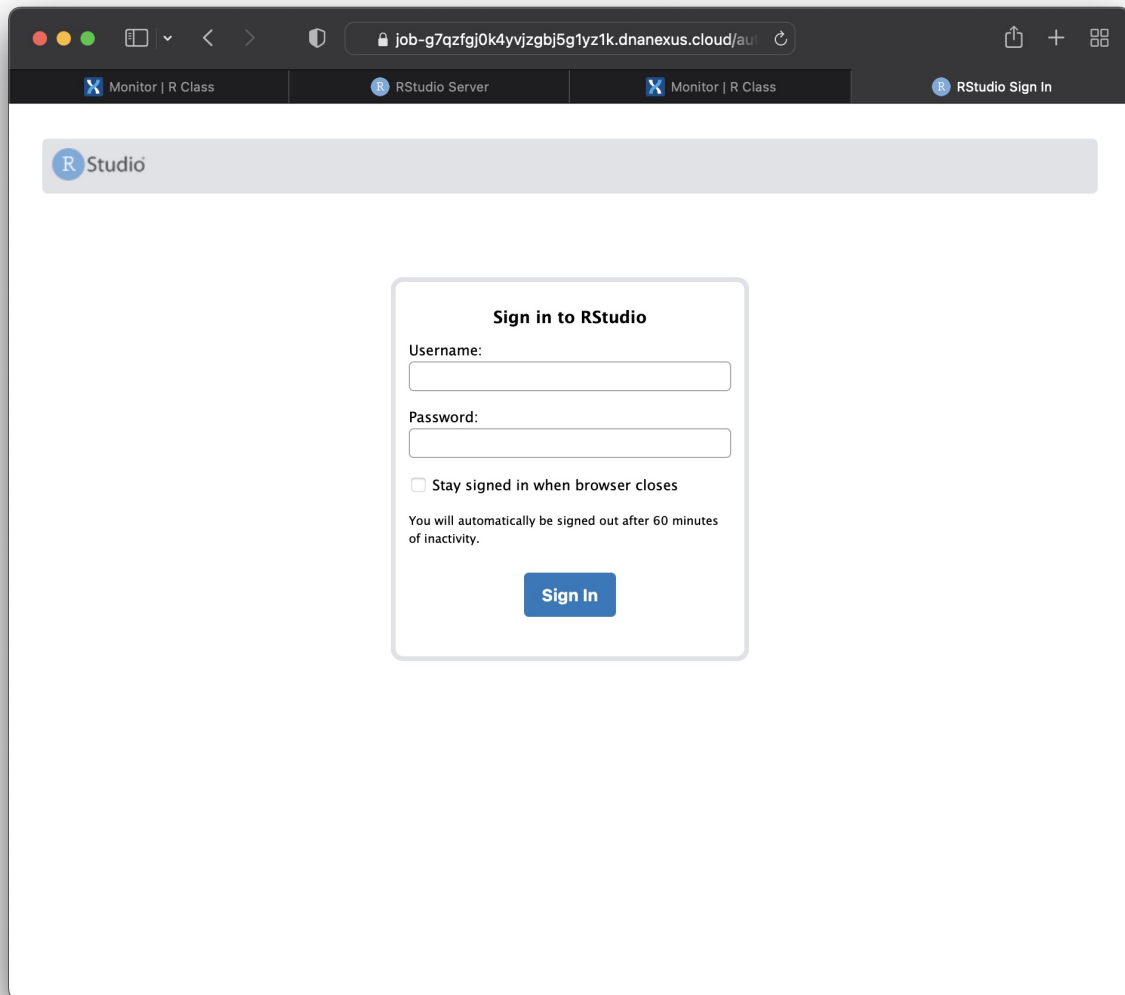
Filters Saved Filters

Status	Name	Executable	Launched by	Started running	Duration	Price	Priority	Worker URL
Running	Alex Emmons	RStudio Session App v3.0.1	Alex Emmons	01/25/2023 4:24 pm	3m	- cost	high	https://job-GP8kq7Q0yFzZ1P93XqQzVp1v.dnanexus.cloud

The reason for waiting ~5 mins is to give the system time to get everything in place. If you click too soon you will see an error message.



Don't PANIC, just wait a little longer and refresh the screen, until you are finally presented with the RStudio login screen.



Proceed with running content from the R Class

From the login screen, login with the username/passwd "rstudio/rstudio", and proceed from there.

If you encounter any problems send email to BTEP at ncibtep@nih.gov (<mailto:ncibtep@nih.gov>)