Objectives
Introduction
Loading the data
Getting to know the data
Plotting with ggplot2
Our first ggplot
Bar plots
Box plots
Violin plots
Density plots
Saving plots
One last thing
Key Points

Code

Introduction to R

visualising RNA-seq data with ggplot2

Maria Doyle

26 June 2019

Acknowledgements

Material created by Peter Mac Data Science with some content remixed from CRUK Cambridge (http://bioinformatics-core-shared-training.github.io/r-intermediate/) and Data Carpentry (https://datacarpentry.org/R-ecology-lesson/04-visualization-ggplot2.html).

Objectives

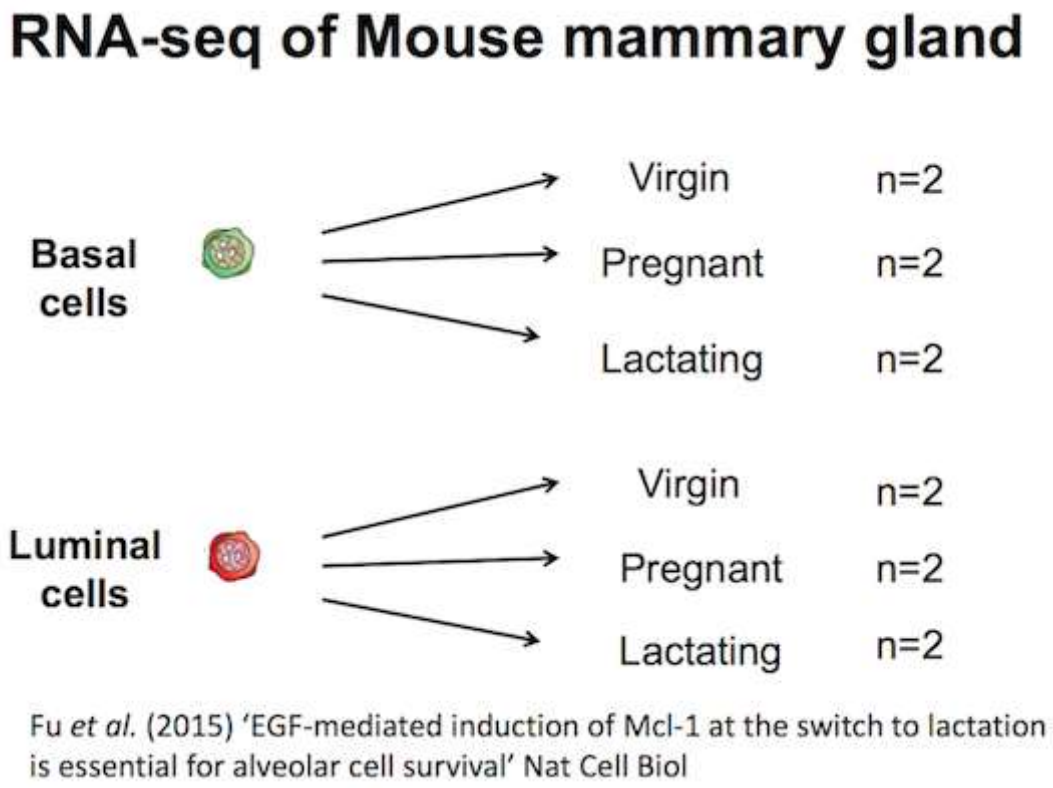
- Demonstrate how R and ggplot2 can be used to visualise data, using RNA-seq as an example
- Demonstrate basic R syntax (`<-` , `dim()` , `head()` , `View()` , `str()` , `summary()`)
- Demonstrate how to use ggplot2 to create bar plots, density plots, box plots and violin plots

Introduction

In this tutorial, we will learn some R through creating plots to visualise data from an RNA-seq experiment.

RNA-seq dataset

We will create some plots using published RNA-seq data from the Nature Cell Biology paper by Fu et al. 2015 (https://www.ncbi.nlm.nih.gov/pubmed/25730472). This study examined expression in basal and luminal cells from mice at different stages (virgin, pregnant and lactating). There are 2 samples per group and 6 groups, 12 samples in total. In this first session we will generate some plots to explore the data, to see if it looks ok.



Fu *et al.* (2015) 'EGF-mediated induction of Mcl-1 at the switch to lactation is essential for alveolar cell survival' Nat Cell Biol

We will use the raw counts here. These are the counts of reads for each gene for each sample. The higher the number of counts the more the gene is expressed. Some of the plots we will make are based on those in the COMBINE RNA-seq R workshop (http://combine-australia.github.io/RNAseq-R/06-rnaseq-day1.html), except here we will create plots using **ggplot2** from the **tidyverse**.

Tidyverse

www.tidyverse.org

The **tidyverse** is a collection of packages that includes `ggplot2` and we will introduce you to some of these packages in this course.

Tidyverse packages

The tidyverse (https://www.tidyverse.org/) makes data science faster, easier and more fun.

The tidyverse package is already installed for you on the server but if you need to install it on your own computer you can use `install.packages('tidyverse')`.

Loading the data

First let's open a new R script. From the top menu in RStudio: `File > New File > R Script`. Let's save it as `first_plots.R`.

We will begin by loading in the packages that we need.

Hide

```
library(tidyverse)
```

`library()` is the **function** in R that we use to load packages. We will see many functions in the course. Functions are "canned scripts" that automate more complicated sets of commands. Many functions are predefined, or can be made available by importing R *packages*. A function usually takes one or more inputs called *arguments*. Here tidyverse is the argument to the `library()` function. Note that functions require parentheses after the function name.

The file we will use is tab-separated, so we will use the `read_tsv()` function from the tidyverse **`readr`** package to read it in. Tsv stands for tab-separated values. There is also a `read_csv()` function for csv files.

To see what the `read_tsv()` function (or any function in R) does, type a `?` before the name and the help will appear in the Help panel on the right in RStudio. Or you can search the function name in the Help panel search box.

Hide

```
?read_tsv
```

We will use the counts file called `counts.tsv.gz` that's in a folder called `data` i.e. the path to the file should be `data/counts.tsv.gz`.

We can read the counts file into R with the command below. We'll store the contents of the counts file in an **object** called `counts`. This stores the file contents in R's memory making it easier to use.

Hide

```
counts <- read_tsv("data/counts.tsv.gz")
```

```
Parsed with column specification:
cols(
  Sample = col_character(),
  CellType = col_character(),
  Status = col_character(),
  SYMBOL = col_character(),
  Counts = col_double()
)
```

In R we use `<-` to assign values to objects. `<-` is the **assignment operator**. It assigns values on the right to objects on the left. So to create an object, we need to give it a name (e.g. `counts), followed by the assignment operator <-`, and the value we want to give it. We can read in a file from a path on our computer on on the web and use this as the value. Note that we need to put quotes ("") around file paths.

Assignment operator shortcut

In RStudio, typing `Alt + -` (push `Alt` at the same time as the `-` key) will write `<-` in a single keystroke in a PC, while typing > `Option + -` (push `Option` at the same time as the `-` key) does the same in a Mac.

The value of `counts` is the contents of the counts file. There is some information output by `read_tsv` on column specifications, this is the data type that `read_tsv` has guessed is contained in each column, we will discuss this more later.

Naming objects

Objects can be given any name, such as `counts`, `rnaseq_counts` or `x`, but some recommendations on naming objects are:

- You want your object names to be explicit and not too long.
- They cannot start with a number (`2x` is not valid, but `x2` is).
- They cannot contain spaces.
- R is case sensitive (e.g., `counts` is different from `Counts`).
- There are some names that cannot be used because they are the names of fundamental functions in R (e.g., `if`, `else`, `for`, see here (https://stat.ethz.ch/R-manual/R-devel/library/base/html/Reserved.html) for a complete list). In general, even if it's allowed, it's best to not use other function names (e.g., `c`, `T`, `mean`, `data`, `df`, `weights`). If in doubt, check the help to see if the name is already in use.
- It's also best to avoid dots (`.`) within an object name as in `my.dataset`. There are many functions in R with dots in their names for historical reasons, but because dots have a special meaning in R (for methods) and other programming languages, it's best to avoid them. Underscores (`_`) would be a better separator, as recommended in the tidyverse style guide (https://style.tidyverse.org/syntax.html#object-name).

Getting to know the data

When assigning a value to an object, R does not print the value. For example, here we don't see what's in the counts file. But there are ways we can look at the data.

We can type the name of the object and this will print the first few lines and some information, such as number of rows.

Hide

```
counts
```

Sample <chr>	CellType <chr>	Status <chr>	SYMBOL <chr>	Counts <dbl>
DG	basal	virgin	Pzp	1
DG	basal	virgin	Aanat	1
DG	basal	virgin	Aatk	746
DG	basal	virgin	Abca1	3307
DG	basal	virgin	Abca4	19
DG	basal	virgin	Abca2	1924
DG	basal	virgin	Abcb7	584
DG	basal	virgin	Abcg1	150
DG	basal	virgin	Abi1	1850
DG	basal	virgin	Abl1	4396

1-10 of 326,148 rows Previous **1** 2 3 4 5 6 … 100 Next

We can use `dim()` to see the dimensions of an object, the number of rows and columns.

Hide

```
dim(counts)
```

```
[1] 326148      5
```

This show us there are 326,148 rows and 5 columns.

In the Environment Tab in the top right panel in RStudio we can also see the number of rows and columns in the objects we have in our session.

We can also take a look the first few lines with `head()` . This shows us the first 6 lines.

Hide

```
head(counts)
```

Sample <chr>	CellType <chr>	Status <chr>	SYMBOL <chr>	Counts <dbl>
DG	basal	virgin	Pzp	1
DG	basal	virgin	Aanat	1
DG	basal	virgin	Aatk	746
DG	basal	virgin	Abca1	3307

Sample <chr>	CellType <chr>	Status <chr>	SYMBOL <chr>	Counts <dbl>
DG	basal	virgin	Abca4	19
DG	basal	virgin	Abca2	1924

6 rows

We can look at the last few lines with `tail()`. This shows us the last 6 lines. This can be useful to check the bottom of the file, that it looks ok.

Hide

```
tail(counts)
```

Sample <chr>	CellType <chr>	Status <chr>	SYMBOL <chr>	Counts <dbl>
LF	luminal	lactate	Gm37254	16
LF	luminal	lactate	*NA*	2
LF	luminal	lactate	*NA*	2
LF	luminal	lactate	*NA*	0
LF	luminal	lactate	*NA*	0
LF	luminal	lactate	*NA*	0

6 rows

Or we can see the whole file with `View()`.

Hide

```
View(counts)
```

Other useful commands for checking data are `str()` and `summary()`.

`str()` shows us the structure of our data. It shows us what columns there are, the first few entries, and what data type they are e.g. character or numbers (double or integer).

Hide

```
str(counts)
```

```
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':   326148 obs. of  5 variables:
 $ Sample  : chr  "DG" "DG" "DG" "DG" ...
 $ CellType: chr  "basal" "basal" "basal" "basal" ...
 $ Status  : chr  "virgin" "virgin" "virgin" "virgin" ...
 $ SYMBOL  : chr  "Pzp" "Aanat" "Aatk" "Abca1" ...
 $ Counts  : num  1 1 746 3307 19 ...
 - attr(*, "spec")=
  .. cols(
  ..   Sample = col_character(),
  ..   CellType = col_character(),
  ..   Status = col_character(),
  ..   SYMBOL = col_character(),
  ..   Counts = col_double()
  .. )
```

`summary()` generates summary statistics of our data. For numeric columns (columns of type double or integer) it outputs statistics such as the min, max, mean and median. For character columns it shows us the length (how many rows).

Hide

```
summary(counts)
```

```
    Sample            CellType            Status             SYMBOL              Count
s
 Length:326148      Length:326148      Length:326148      Length:326148      Min.   :
0.0
 Class :character   Class :character   Class :character   Class :character   1st Qu.:
0.0
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Median :
14.0
                                                                             Mean   :
824.9
                                                                             3rd Qu.:
402.0
                                                                             Max.   :2
887969.0
```

Try running the above commands yourself.

Plotting with `ggplot2`

`ggplot2` is a plotting package that makes it simple to create complex plots. One really great benefit of ggplot2 versus the older base R plotting is that we only need to make minimal changes if the underlying data change or if we decide to change our plot type, for example, from a bar plot to a box plot. This helps in creating publication quality plots with minimal amounts of adjustments and tweaking.

`ggplot2` likes data in the 'long' format, also called 'tidy' format, i.e., a column for every variable, and a row for every observation. Well-structured data will save you lots of time when making figures with `ggplot2` . We will discuss tidy data more later in the course.

ggplot graphics are built step by step by adding new elements. Adding layers in this fashion allows for extensive flexibility and customization of plots.

To build a ggplot, we use the following basic template that can be used for different types of plots:

Hide

```
ggplot(data=, mapping=aes()) + geom_ ()
```

```
Error in geom_() : could not find function "geom_"
```

Three things are required for a ggplot:

1. The data
2. The mapping of variables (columns) in the data to visual properties (called aesthetics in ggplot2) of objects in the plot
3. The type of plot – this is called a geom in ggplot2 terminology

There are different geoms we can use to create different types of plot e.g. bar plot versus box plot, to see some of the geoms available see the ggplot2 help or the handy ggplot2 cheatsheet (https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf).

Our first ggplot

Before we start plotting the RNA-seq data we'll demonstrate how ggplot works using one of the built-in datasets in R. You can see what built-in datasets there are by typing `data()` . We will use the dataset called `women` that contains the average heights and weights for American women.

First take a look at the dataset.

Hide

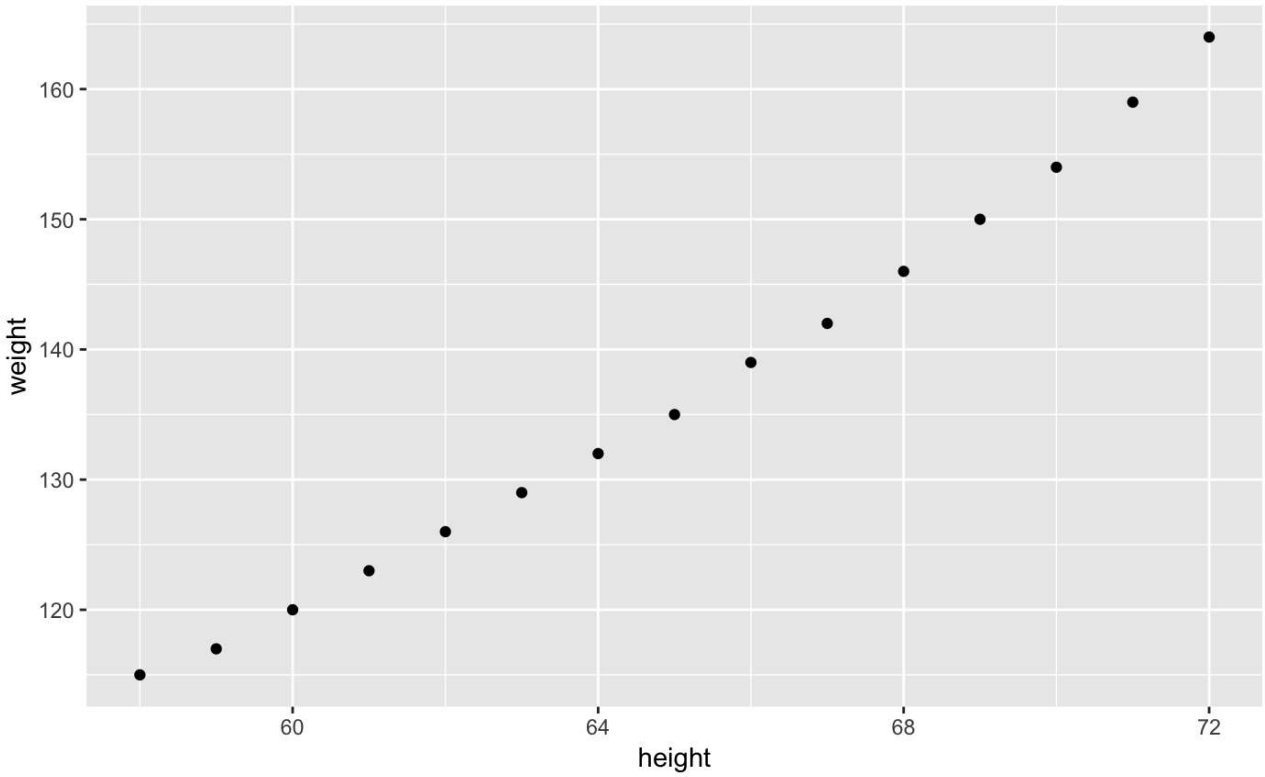
```
women
```

```
   height weight
1      58    115
2      59    117
3      60    120
4      61    123
5      62    126
6      63    129
7      64    132
8      65    135
9      66    139
10     67    142
11     68    146
12     69    150
13     70    154
14     71    159
15     72    164
```

Let's make a scatterplot. We give `ggplot()` the dataset (`women`) and in `aes()` we say which columns are the x and y axis values. To say what type of plot we want we add a geom using the `+` operator. This must go at the end of the line, and not at the beginning or ggplot2 will give an error.

Hide

```
ggplot(data=women, mapping=aes(x=height, y=weight)) +
  geom_point()
```



We have generated our first plot!

Now we will explore the RNA-Seq data using different types of plots.

Bar plots

How many counts do we have for each sample?

We can make bar plots to visualise this. To do this we will use `geom_bar()`.

You might think we specify the `x=` and `y=` values as before with `geom_point()` but look what happens.

Hide

```
ggplot(data=counts, mapping=aes(x=Sample, y=Counts)) +
  geom_bar()
```

```
Error: stat_count() must not be used with a y aesthetic.
```

We get an error `Error: stat_count() must not be used with a y aesthetic.`

We can use the bioinformatician's good friend Google to try to figure out why. Through Google we can find an answer on Stack Overflow (Stack Overflow is another good friend of bioinformaticians) https://stackoverflow.com/questions/39679057/r-ggplot2-stat-count-must-not-be-used-with-a-y-aesthetic-error-in-bar-graph/39679104 (https://stackoverflow.com/questions/39679057/r-ggplot2-stat-count-must-not-be-used-with-a-y-aesthetic-error-in-bar-graph/39679104). As pointed out in that post, we could specify `y=Counts` if we had already calculated the value we wanted to plot (the total counts for each sample). However, our input data contains the counts for *each gene* for each sample. We want to **sum** these gene counts for each sample. Let's take a look at the help for `geom_bar()` (as shown in that Stack Overflow post).

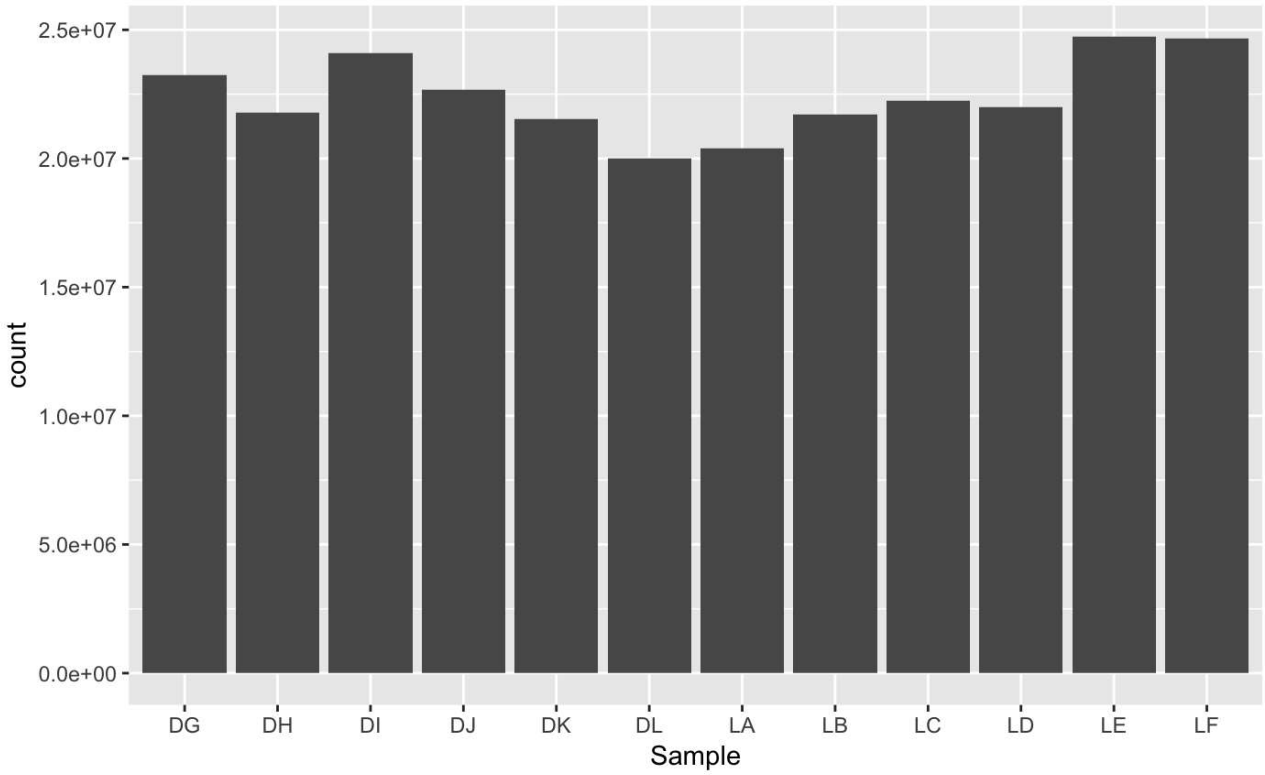
Hide

```
?geom_bar
```

It says “geom_bar() makes the height of the bar proportional to the number of cases in each group (or if the weight aesthetic is supplied, the sum of the weights).” So we can specify `x=Sample` and `weight=Counts` and ggplot will automatically sum the counts for each sample for us.

Hide

```
ggplot(data=counts, mapping=aes(x=Sample, weight=Counts)) +
  geom_bar()
```



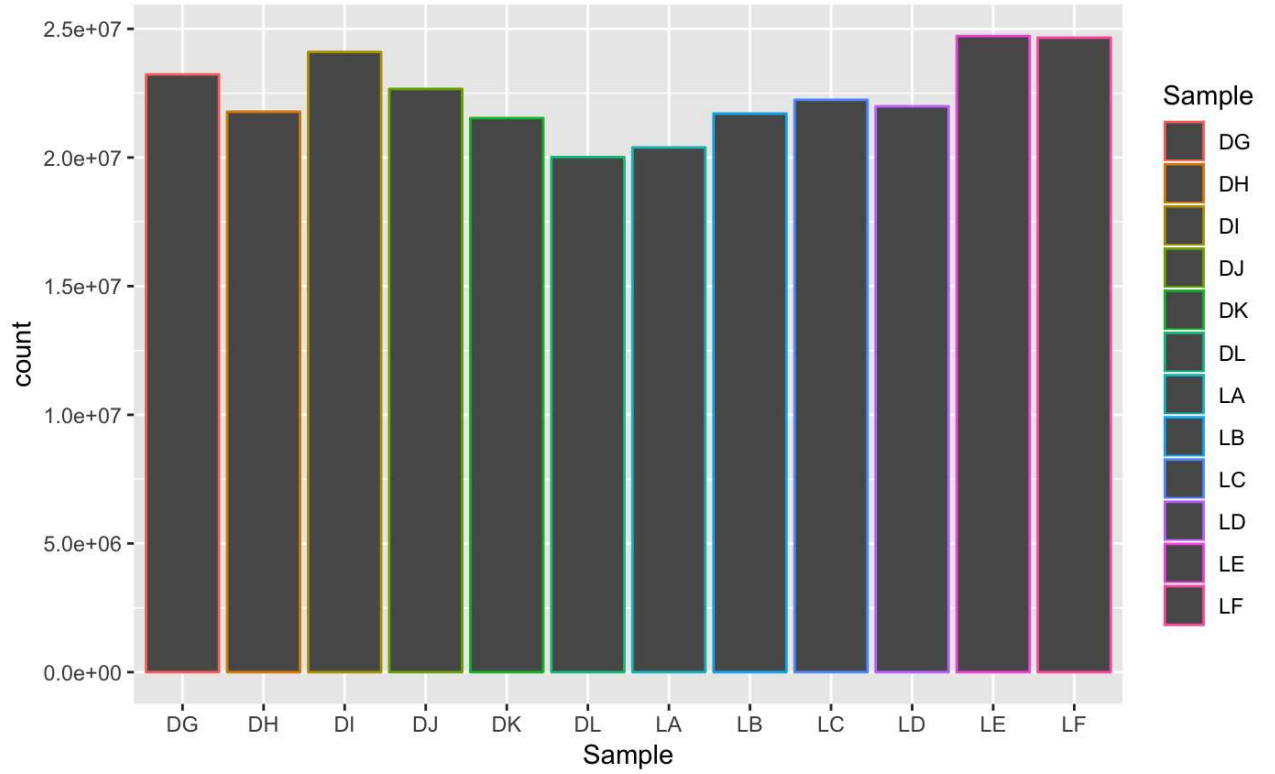
This shows us we have 20-25 million counts per sample, none are very different.

What if we would like to add some colour to the plot, for example, a different colour bar for each sample.

If we look at the `geom_bar` help again we can see under the heading called “Aesthetics” that there’s an option for colour. Let’s try adding that to our plot. We’ll specify we want to map the Sample column to `colour=`. As we are mapping colour to a column in our data we need to put this inside the `aes()`.

Hide

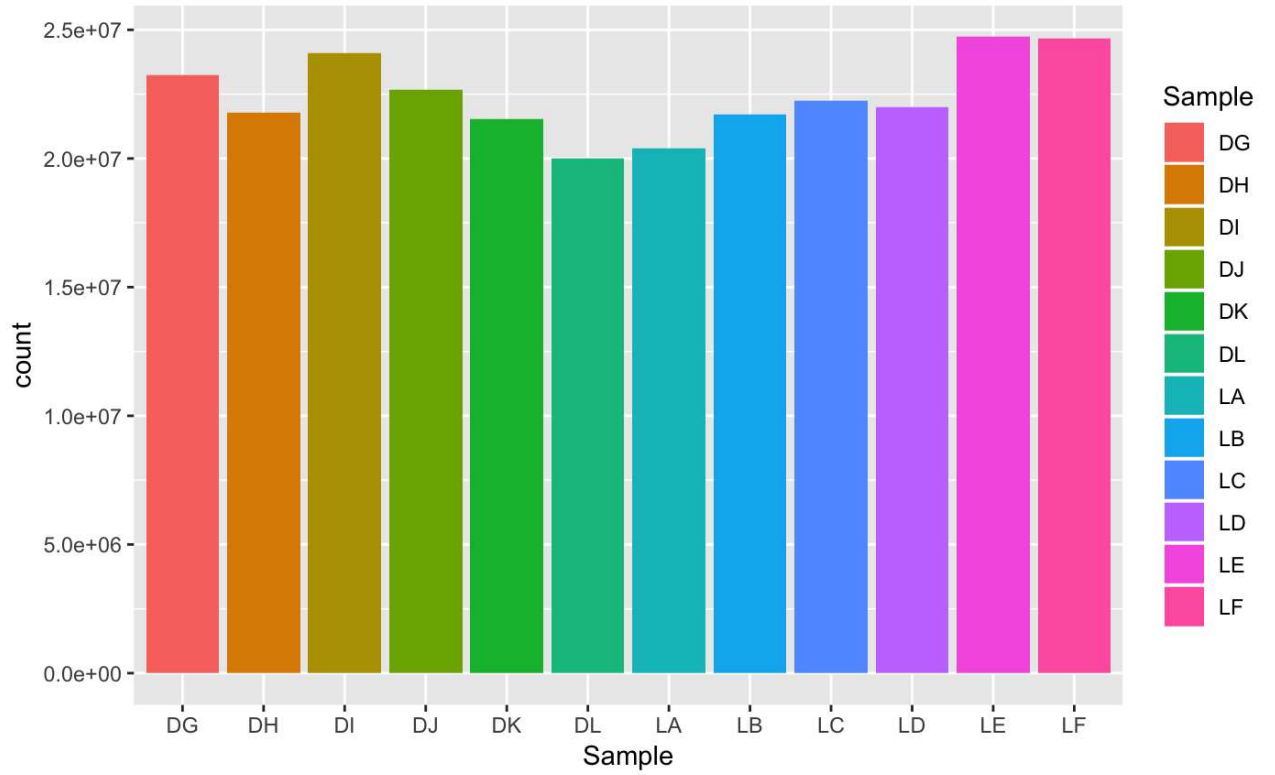
```
ggplot(data=counts, mapping=aes(x=Sample, weight=Counts, colour=Sample)) +
  geom_bar()
```



Hmm colouring the edges wasn't quite what we had in mind. Look at the help for `geom_bar` to see what other aesthetic we could use. Let's try `fill=` instead.

Hide

```
ggplot(data=counts, mapping=aes(x=Sample, weight=Counts, fill=Sample)) +
  geom_bar()
```

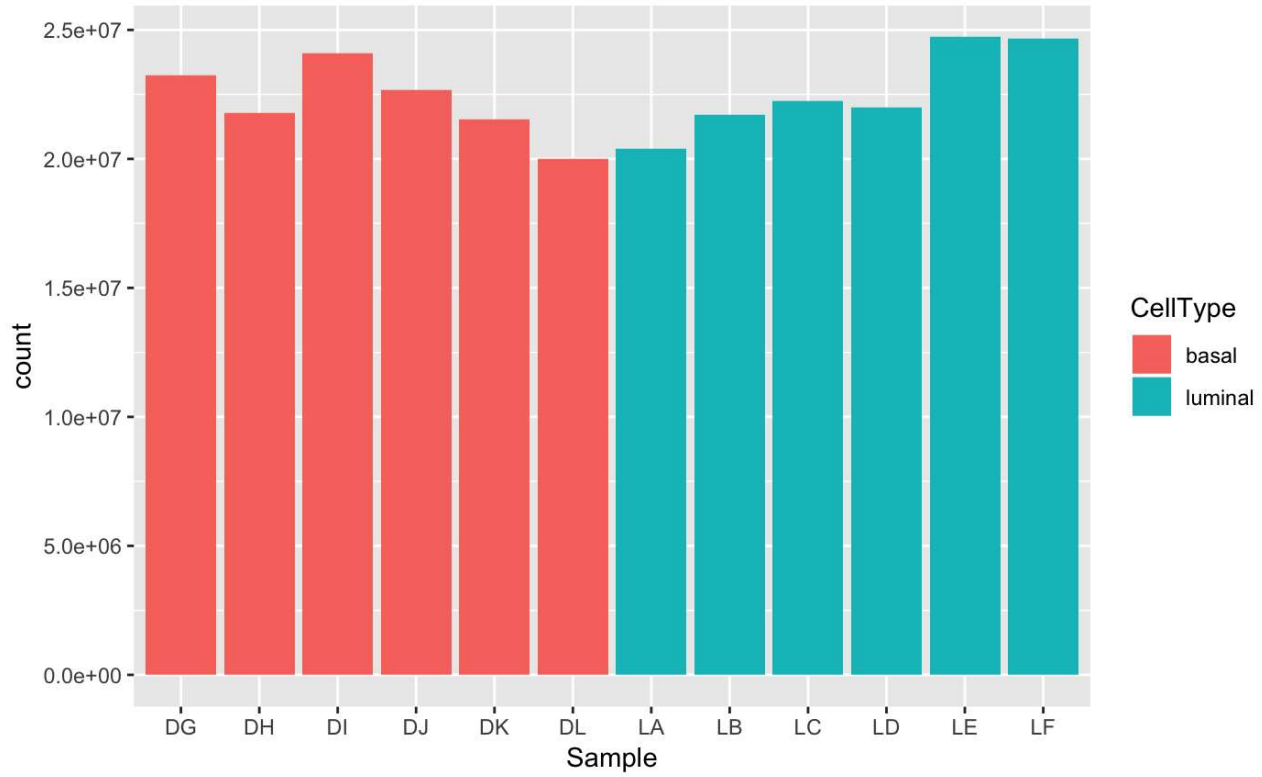


That looks better. `fill=` is used to **fill** in areas in ggplot2 plots, whereas `colour=` is used to colour lines and points.

A really nice feature about ggplot is that we can easily colour by another variable e.g. cell type (basal vs luminal) by simply changing the column we give to `fill=`.

Hide

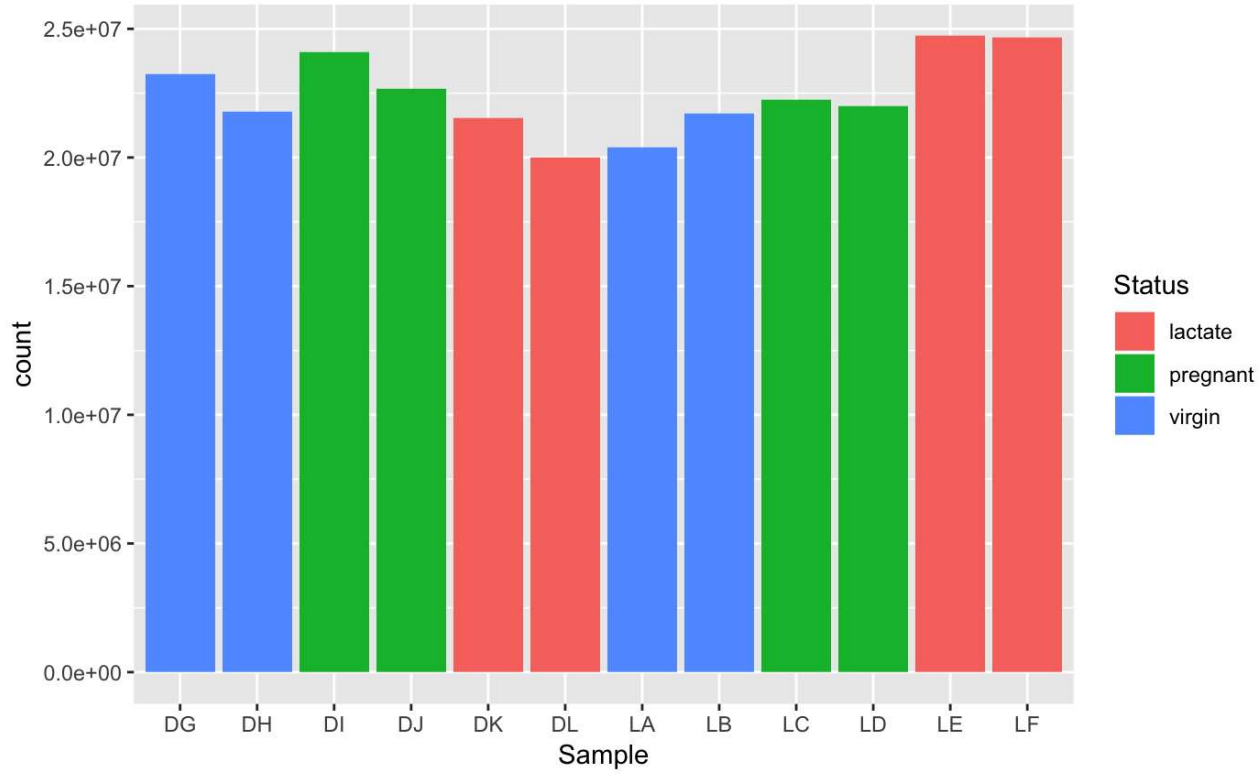
```
ggplot(data=counts, mapping=aes(x=Sample, weight=Counts, fill=CellType)) +
  geom_bar()
```



Or we can colour by status (virgin vs pregnant vs lactating).

Hide

```
ggplot(data=counts, mapping=aes(x=Sample, weight=Counts, fill=Status)) +
  geom_bar()
```



For more details on bar plots see this R bloggers post (https://www.r-bloggers.com/detailed-guide-to-the-bar-chart-in-r-with-ggplot/). R bloggers is a useful resource for R news and tutorials, you can subscribe to their mailing list for free.

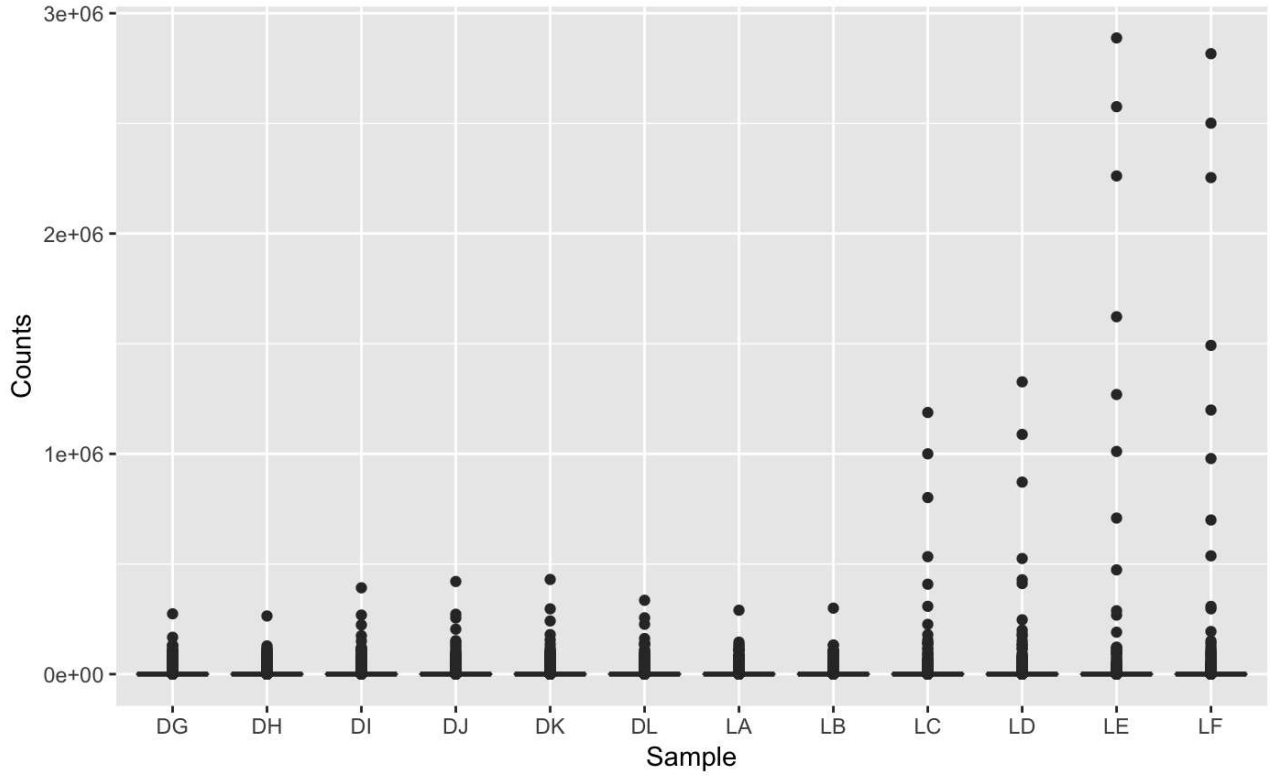
Box plots

In addition to viewing the number of counts per sample it is also useful to visualise the distribution of the counts. This helps us to compare the samples and check if any look unusual. We can make box plots to visualise the distribution of counts for each sample.

Let's try that.

Hide

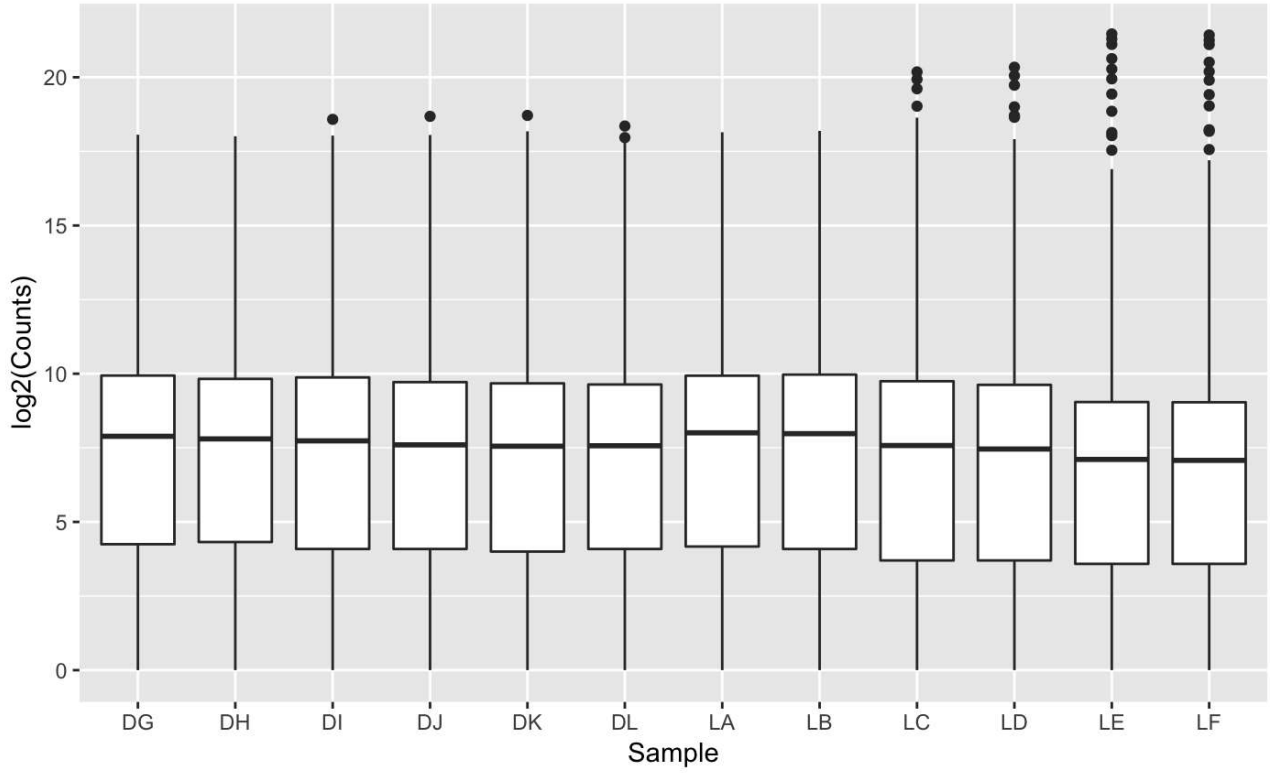
```
ggplot(data=counts, mapping=aes(x=Sample, y=Counts)) +
  geom_boxplot()
```



That looks weird. It's because we have some genes with extremely high counts. To make it easier to visualise the distributions we usually plot the logarithm of RNA-seq counts. We'll plot the Sample on the X axis and log2 Counts on the y axis. We can log the Counts within the `aes()`.

Hide

```
ggplot(data=counts, mapping=aes(x=Sample, y=log2(Counts))) +
  geom_boxplot()
```



Note that we get a warning here about rows containing non-finite values being removed. This is because some of the genes have a count of zero in the samples and a log of zero is undefined. Usually we would filter lowly expressed genes before creating the box plots so we may not always get a warning like this. But here we are just demonstrating the different types of plots that can be made.

The box plots show that the distributions of the samples are not identical but they are not very different.

Exercise

Colour the box plots by Sample, then CellType, then Status.

Violin plots

Box plots are useful summaries, but hide the *shape* of the distribution. For example, if the distribution is bimodal, we would not see it in a boxplot. An alternative to the boxplot is the violin plot, where the shape (of the density of points) is drawn. See here (https://blog.bioturing.com/2018/05/16/5-reasons-you-should-use-a-violin-graph/) for an example of how differences in distribution may be hidden in box plots but revealed with violin plots.

Exercise

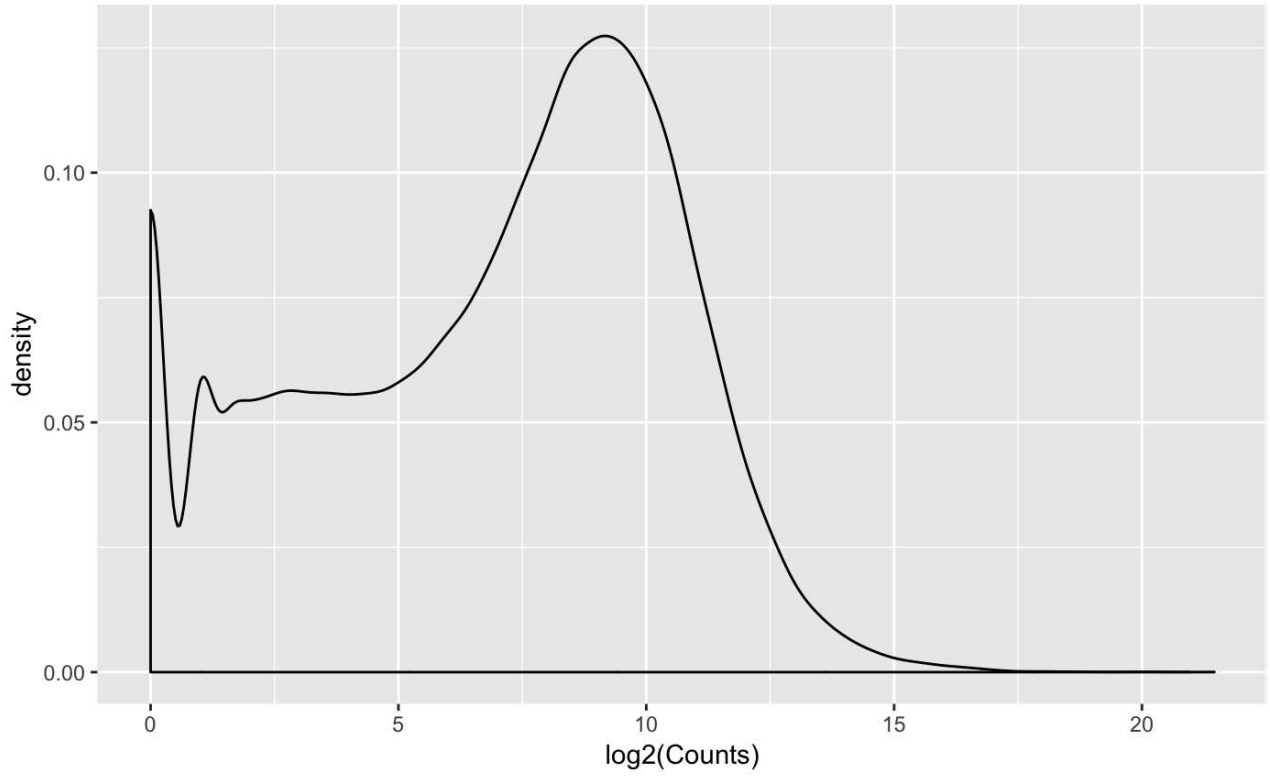
Make violin plots instead of box plots using `geom_violin()` and colour by Sample, then CellType, then Status. How do the violin plots for the samples compare?

Density plots

We can also visualise distributions with density plots. In this case we specify `x=log2(Counts)` (no need to specify `y=`) and `geom_density()` will create a density plot of the counts.

Hide

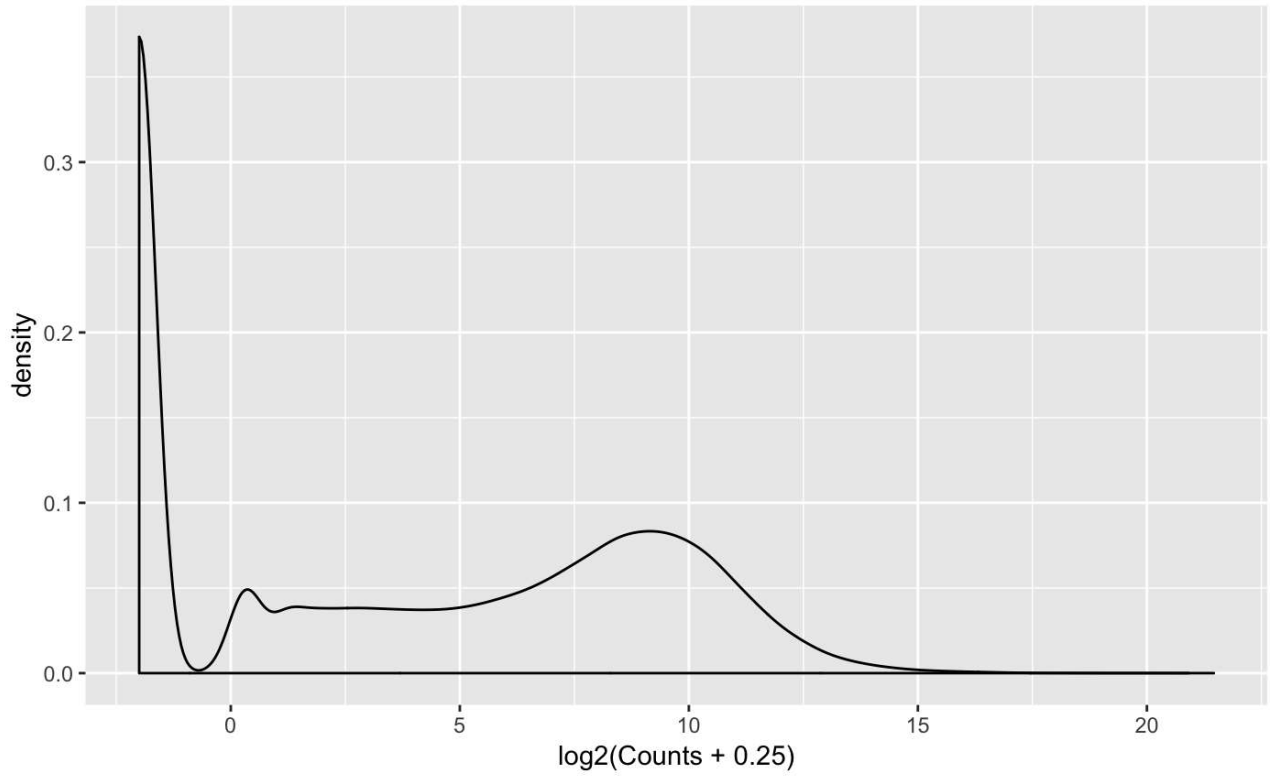
```
ggplot(data=counts, mapping=aes(x=log2(Counts))) +
  geom_density()
```



We can add a small number to every count (we'll choose 0.25) in order to visualise the amount of genes with counts of zero in this plot.

Hide

```
ggplot(data=counts, mapping=aes(x=log2(Counts+0.25))) +
  geom_density()
```

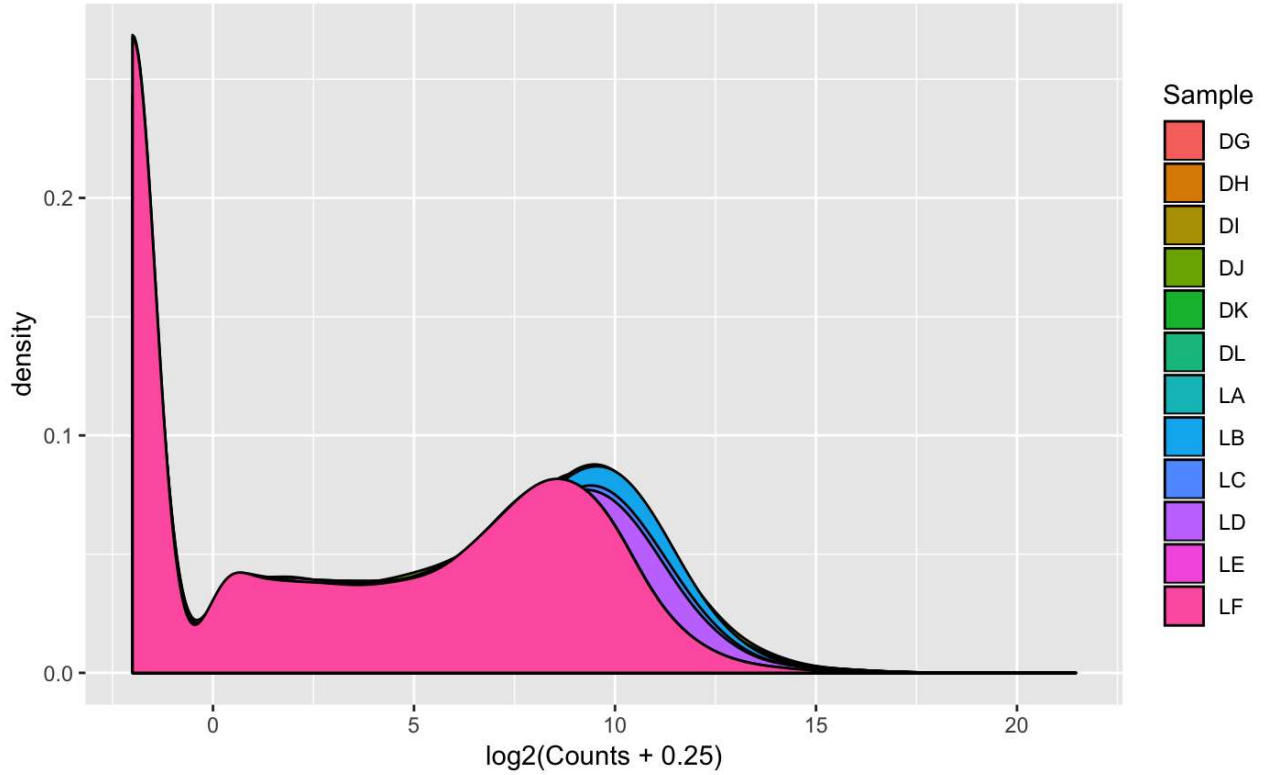


Note the large peak that appears on the left. This is the large number of genes with zero counts. In an RNA-seq analysis filtering lowly expressed genes would remove or reduce that peak.

We can colour by sample using `fill=` .

Hide

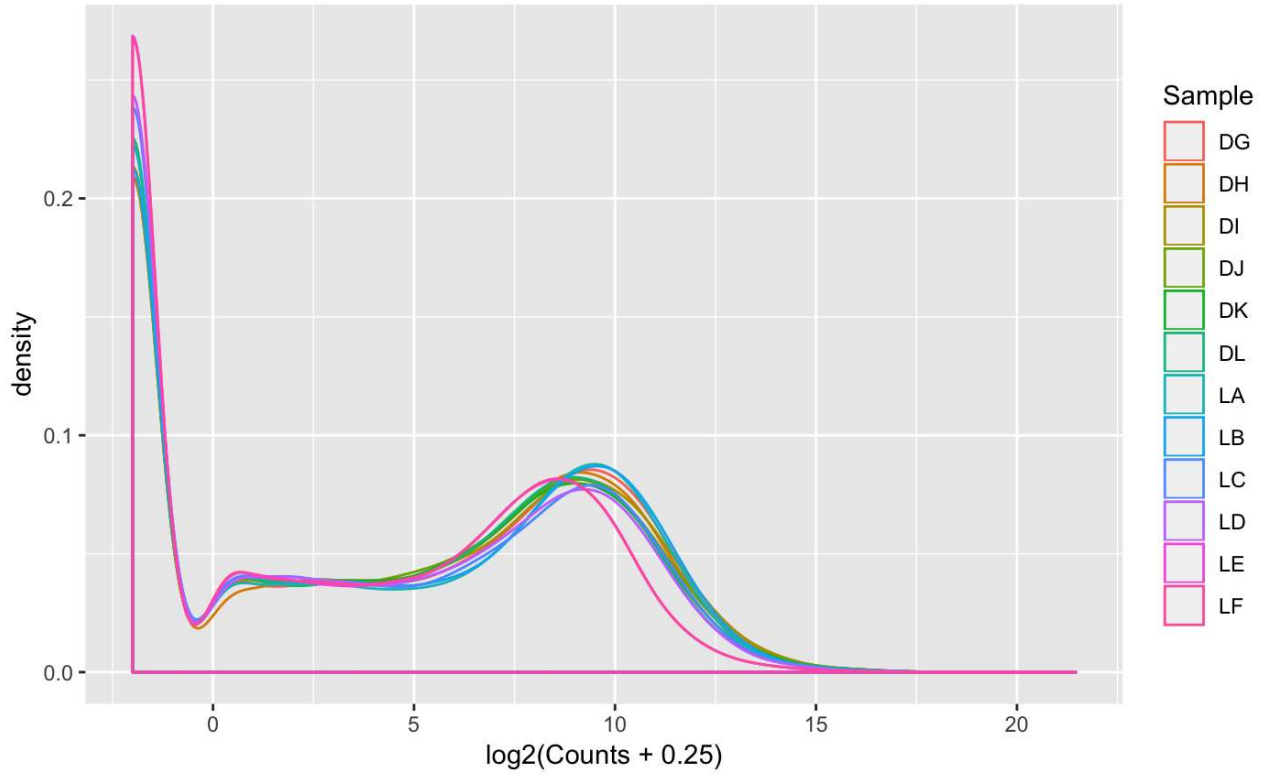
```
ggplot(data=counts, mapping=aes(x=log2(Counts+0.25), fill=Sample)) +
  geom_density()
```



If we would like to colour the lines in the plot instead of the areas we can use `colour=` instead of `fill=` .

Hide

```
ggplot(data=counts, mapping=aes(x=log2(Counts+0.25), colour=Sample)) +
  geom_density()
```

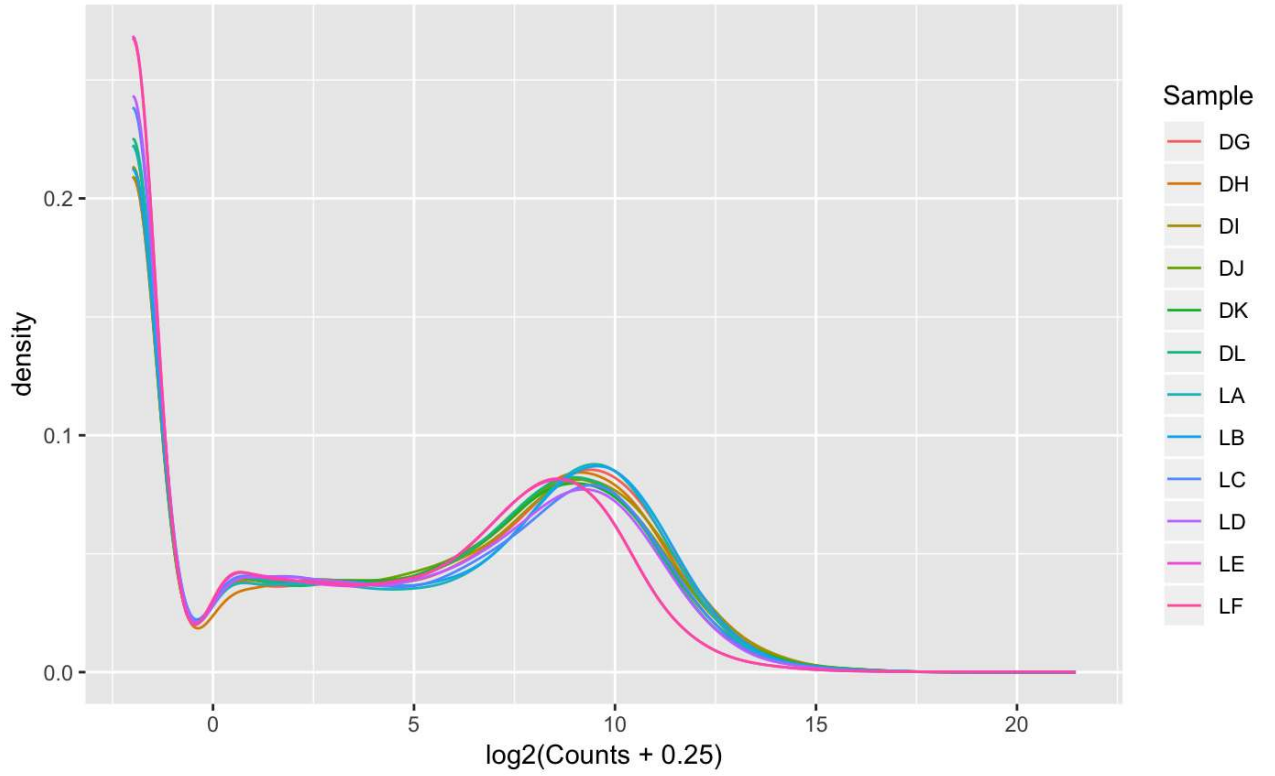


Why is there a line across the bottom of the plot?

We can use Google and Stack Overflow again to find out why: https://stackoverflow.com/questions/49593634/unintended-line-across-x-axis-of-density-plot-r (https://stackoverflow.com/questions/49593634/unintended-line-across-x-axis-of-density-plot-r). It is because `geom_density()` plots a polygon shape. If we don't want that line we can adapt the code given in that Stack Overflow post, using `geom_line(stat="density")`.

Hide

```
ggplot(data=counts, mapping=aes(x=log2(Counts+0.25), colour=Sample)) +
  geom_line(stat="density")
```



Statistical transformations

Many graphs, like scatterplots and line plots, plot the raw values from your dataset; other graphs, like density plots and bar plots, calculate new values to plot.

- bar plots and histograms (geom_bar, geom_histogram, geom_freqpoly) bin your data and then plot bin counts
- density plots (geom_density) compute and draw kernel density estimate, which is a smoothed version of the histogram
- smoothing functions (geom_smooth) fit a model to your data and then plot predictions from the model
- box plots (geom_boxplot) compute a robust summary of the distribution and display a specially formatted box

The algorithm used to calculate new values for a graph is called a stat

Saving plots

We can output plots to pdf using `pdf()` followed by `dev.off()`. We put our plot code after the call to `pdf()` and before closing the plot device with `dev.off()`.

Let's save our last plot.

Hide

```
pdf("myplot.pdf")
ggplot(data=counts, mapping=aes(x=log2(Counts+0.25), colour=Sample)) +
  geom_line(stat="density")
dev.off()
```

One last thing

It is recommended to use bar plots only for visualising *single values*, for example, the total counts per sample here. Dynamite plots, a type of bar plot commonly used by scientists and in journals, are not recommended, see this post called “Dynamite plots must die” (https://simplystatistics.org/2019/02/21/dynamite-plots-must-die/) for an explanation. Box plots, violin plots and density plots can be a better way to visualise distributions of data.

Exercise

- Read in the normalised counts (from “data/normcounts.tsv.gz”). This is the `counts` data after filtering lowly expressed genes and normalising for differences in sequencing depth and composition bias.
- How many rows and columns are in the file?
- What are the column names?
- In the first row, what is the sample id, the gene symbol and the value for the counts?
- In the last row, what is the sample id, the gene symbol and the value for the counts?
- What are the minimum, maximum, mean and median values in the Norm_counts column?
- Make as many plots as you can from bar plots, box plots, violin plots and density plots using the normalised counts (some starter code is below)
- Make separate plots coloured by Sample, CellType, Status, where appropriate.
- Do you notice any differences with the normalised counts versus the unnormalised counts that we used previously (`counts` dataset)?

Hide

```
norm_counts <- read_tsv()
```

Hide

```
ggplot(data= , mapping=aes(x= , weight= , fill= )) +
  geom_bar()
```

Hide

```
ggplot(data= , mapping=aes(x= , y= , fill= )) +
  geom_boxplot()
```

Hide

```
ggplot(data= , mapping=aes(x= , y= , fill= )) +
  geom_violin()
```

Hide

```
ggplot(data= , mapping=aes(x= , colour= )) +
  geom_density()
```

Key Points

- We use the `library()` function to load packages we want to use. Note that they need to be already installed with e.g. `install.packages()`.
- We can read in tab-separated files with `read_tsv()`.
- We can check our data with the functions `dim()`, `head()`, `tail()`, `View()`, `str()` and `summary()`.
- **`ggplot2`** is a plotting package that makes it simple to create complex plots.
- ggplot2 plots have 3 components: data (dataset), mapping (columns to plot) and geom (type of plot).
- The `+` sign is used to add geoms (and new layers as we will see later in the course). It must be placed at the end of the preceding line. If it is added at the beginning of the line, **`ggplot2`** will return an error message.
- ggplot2 plots can be easily coloured by columns in the dataset. `fill=` can be used to colour areas and `colour=` to colour lines.
- A pdf of the plot can be generated with `pdf()` and `dev.off()`