REPRODUCIBILITY

L1

Repeatability → ability of obtaining the same result with the same operator & reagents

≠

Reproducibility → " " " " " in ≠ places with ≠ reagents

Biology has a certain amount of intrinsic stochasticity that sometimes renders it impossible to obtain the exact same result.

High throughput analysis, even when starting from the same raw data, are usually not 100% reproducible.
Also high-throughput technology evolves → the statistics behind them is ≠ → final results are ≠.
⇒ **important to standardize both instruments & data analysis protocols.**
Important to include all relevant info to reproduce your results when publishing something

Ten Simple Rules for Reproducible Computational Research

From Sandve GK, Nekrutenko A, Taylor J, Hovig E (2013) - Ten Simple Rules for Reproducible Computational Research.

PLoS Comput Biol 9(10): e1003285. https://doi.org/10.1371/journal.pcbi.1003285

1. For Every Result, Keep Track of How It Was Produced
2. Avoid Manual Data Manipulation Steps
3. Archive the Exact Versions of All External Programs Used
4. Version Control All Custom Scripts
5. Record All Intermediate Results, When Possible in Standardized Formats
6. For Analyses That Include Randomness, Note Underlying Random Seeds
7. Always Store Raw Data behind Plots
8. Generate Hierarchical Analysis Output, Allowing Layers of Increasing Detail to Be Inspected
9. Connect Textual Statements to Underlying Results
10. Provide Public Access to Scripts, Runs, and Results

DOCKER THEORY

Docker engines/containers

Another problem that is generated from the problem of reproducibility is that if a certain number of software and libraries are installed on the PC, the user cannot be always sure that, by installing one libraries, the others will be updated. This is a critical issue.

Since few years a specific infrastructure, called **Docker**, has been developed to help with the organisation of the data. The **docker containers** represent exactly the same approach that is used to distribute goods using tucks, ships or airplanes with the physical metal boxes.

The physical containers have been built with a common structure that allows them to be piled up and also to contain any kind of product; the advantage of this system is that the same container can transport different things.

The same approach has been applied to the use of software; today, especially for RNA-seq and sc-RNAseq analysis, the problem is the software are quite sophisticated to install them locally and they work on Linux.

✅ Docker helps to store a Linux software within a box that will work on a Window/IOS computer and are constant through time.

The emulator of the Linux interface (WSL for Windows) will read what is present on these containers and execute. Thus, you don't need to install on the PC everything necessary to do an analysis, you will put everything inside a docker and utile is to run par of the analysis itself.

The docker box remains the same through time, so even if the infrastructure (hardware) is changed, the data obtained will be the same. For example, things stored in the docker now, will remain the same in three years and can be reused even if the PC is different.

Moreover, if you change your pc you can download the same docker and you have your software back and running even on different environments.

Dockers are one of the way that enable us to guarantee a certain level of reproducibility.

Normally, with a virtual machine, some part of the RAM and disk will be dedicated to it, e.g. if you install a Linux virtual machine on Windows.
When the virtual machine is started, those amounts of RAM and disk will be used by it even if the machine isn't actively working.

On the other hand, the Docker software normally runs in background and can execute its image as a normal window software, thus using RAM, disk...only when it's computing.
So, dockers are similar to a virtual machines, with the difference that they do not require a fixed RAM space on the computer.

Then, when the computation is finished, all the RAM and the space is free again. Thus, the space is used only if the container is running.

The advantage is that you don't limit the space on your PC and hardware when you do not use the specific object (the docker in this case).

The dockers will occupy some space on the computer, but this is ROM and is relatively small compared to the requirements of virtual machines, which are very large.

❗ To run docker on Windows you need a very simple virtual machine that changes its dimensions depending on the amount of RAM requested by the docker to be used.

Normally, the docker engine is designed for Linux, thus if installed on a Linux or on a Mac (which is very Linux oriented), the helping virtual machine is not needed.

✅ Dockers are the way to guarantee reproducibility of the analysis

Example of docker application:

The data of one RNAseq experiment, stored as FastQ files on the GEO database, are analysed with Star2.5, RSem3.1 and DeSeq2.5.
Instead of installing on the PC all these software that will require much space, a docker container is built will all of these inside and put aside.
The docker can be also stored in public repositories.

The docker stay constant in time, thus it will give the same results if the input data is the same.

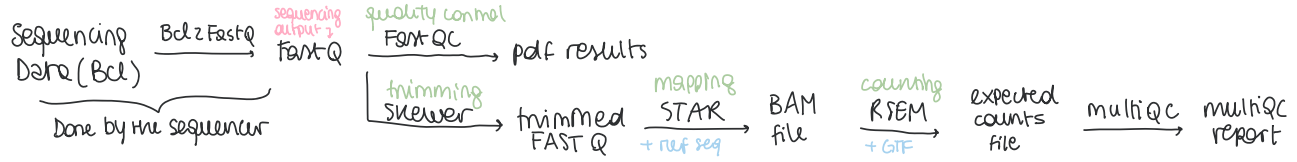
This is why dockers are very useful and guarantee reproducibility. They cannot be built at the beginning of the experiments, they are made after the pipeline is defined.

After building the docker, the same analysis is repeated to check if the results obtained are the same; then at that point, you can store the workload.

Dockers are also useful with reviewers that want to check out the software used in the analysis.

At the end, dockers of another published work can be used to analyse your own data. The container becomes a piece of software that can be used to run part of the analysis described on GitHub.

BULK RNA SEQ



Aim: understanding if there are any changes in gene expression after some perturbations (ex: treatment)
⇒ we are comparing ≠ conditions

Once you have your counts

- normalization (TPM, RPKM, FPKM ...)
- dimensionality reduction (PCA)
- DE analysis → DE genes can be used to understand biological events from a transcriptional POV.

SINGLE CELL RNA SEQ

1- **Sequencing output** : BCL file

↓ MKFASTQ

2- **3 FastQ filer** → R1 → 10X barcode (cell identifier) + UMI (transcript id)
→ R2 → contains the actual sequence
\+ → I1 → contains the sample index sequence (6 nts) : allows to separate the different samples that are run together

Index.html → quality control info

3- **Trimming of R1 and R2** → poly A is used to build the 1st cDNA filament, while TSO = template switch oligo is used for the II strand.
It pairs to untemplate c in the 1st filament ⇒ must be removed prior to the analysis together with the poly A at the 3'.

5' TSO + cDNA + poly A 3'

trimming via tagmentation
(yeast transposase TNAS)

Trimmed R1 + Trimmed R2 + GTF + ref

↓ UMI quality control

↓ STAR Alignment

Mapped reads

UMI counting

Counts → single cell data are 0 inflated : conversion of a sparse matrix into a dense matrix using **cellranger**

Once you have your counts
- normalization (TPM, FPKM, RPKM)
- dimensionality reduction (UMAP, t-SNE)
- clustering → identification of cells with similar transcriptional profile = transcriptional characterization of cell subsets / identification of novel subsets charact. by specific markers.
Markers can be used for isolation of these subpop. with FACS for ex.

DIMENSIONAL REDUCTION

- **PCA with SVD (single value decomposition)** → linear, for bulk

1. Plot the data in a high dimensional space & calculate the mean on each dimension.
2. Change the reference axis: the intersection of the means becomes the center
3. Find the line passing through the origin that best fits the data:
   - project the points onto the line
   - calculate the sum of the squared distances betw. each point & the origin
   - find the max value possible → this is the 1st PC

   It will be a linear combination of the variables and will have a certain slope.
4. Knowing the slope, you can find the linear combination of the line: if the slope is 0.25, a unit increase in the PC1 is composed of 4 steps along the first variable and one along the second.
   You can solve with the pithagorean theorem and find the length of the PC1.
5. This value is then set to 1 and everything is scaled accordingly:
   - the unit vector of PC1 = **eigenvector for PC1**
   - the values of the linear combination composing it = **loading scores**
   - the sum of squares distances = **eigenvalues**
   - √sum of squares = **singular value**
6. Draw PC2 ⊥ PC1
7. Repeat the same procedure to calculate eigenvector & loading scores for PC2
8. Draw the plot: the intersection of the points on the 2 dimension is the position of that point in the PCA plot.
9. Repeat 6-7-8 for every other dimension

Scree plot → graphical repr. of % variance along each PC
↳ obtained by converting sum of squared dist for each PC into variation around the origin:

$$\frac{SSD_{PCx}}{n-1}$$ where n = sample size

- **tSNE** → t-distributed stochastic neighbor embedding, for scRNA-seq

① Measure the distances (various ways to do that → ✓ flexibility) between each couple of points in the high dimensional space

② Plot this distance on a normal curve centered on the point of interest and with a σ inversely prop to expected point density (perplexity parameter)

③ Draw a line from the other point to the curve. This is the **unscaled similarity**.

④ Repeat 2-3 for every couple of points.

⑤ Calculate **scaled similarities**: each unscaled similarity is divided by the sum of all the unscaled similarities

Sum of scaled sim = 1 ⟹ we can compare distances obtained with curves with different σ

We obtain a **scaled similarity matrix**: each point in a row is a similarity score of that point and another point in the dataset.
On the diagonal we have similarities with points & their own → max value but set to 0

⑥ Place the points randomly in the dimensionally reduced space

⑦ Calculate again similarity scores in the dim. reduced space & scale them. This time use a **t-distribution** → makes it easier to keep more similar data together in the d.r. space.

Obtain another similarity matrix which is much different from the other.

⑧ Move one point at the time & repeat ⑥-⑧ until the 2 matrices are ideally equal.

! **dimensional reduction ≠ clustering**

- **UMAP** = uniform Manifold Approx. & Projection → non linear, ~ t-SNE but focuses more on the big picture and its faster ✓

① Calculate distances between each pair of points in the high dimensional space.

② Put the points on a 2D graph with the point of interest on the 0 and all the other at the correspective distances

③ Draw a curve to calculate **similarity scores**. Its shape depends on the **n° of nearest neighbors parameter**, usually 15: the sum of the y-coord. of the points must be equal to $\log_2(n° \text{ of } nn)$
⇒ similar scaling to t-SNE, but ≠ t-SNE distances here are made symmetrical

④ Repeat 2-3 for all the other points

⑤ Clusters of points are created in the high dim. space & then plotted in the dim. red. graph

⑥ Then this aggregation is adjusted:
- pick 2 random points that should be moved together. The pr of being selected increases for points that are near in the high dim space (i.e. high similarity score) (A & B)
- it selects randomly (50-50) one of the 2 points and moves it closer to the other (A → B)
- it selects another point out of the high dimensional cluster (similarity scores do not influence pr of choice) which will be moved further from the previously moved one (E)
- It then calculates how much must these 2 points be moved:
  To do so **low dim. similarity scores** are calculated using a t-distribution centered on the point to move (~ t-SNE)

  the aim is to maximise similarity scores betw A-B & minimize B-E
- points are moved → little movements only ✓ good with big data

⑦ Repeat ⑥ again & again until you have a dim. red. graph similar to the high dim one.

Differences umap & t-SNE

ⓐ Speed umap >> tSNE

ⓑ t-SNE results are always different ≠ UMAP → random initialization of low d. graph

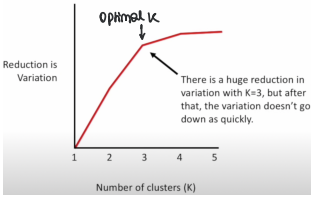
ⓒ t-SNE moves every point each iteration, UMAP only few

ⓓ t-SNE more focused on details, UMAP on the big picture
↳ the higher the nnn parameter, the more focused on the big picture.

CLUSTERING

- **K-MEANS**

optimal K chosen with the elbow plot



✓ Easiest and ↓ hardware demanding

① Random selection of **K** datapoints to be set as **centroids**

② Measure of all the distances datapoints - centroids

③ Assign each data point to the closest centroid

④ Calculate the mean variance of each cluster

⑤ Calculate all the distances datapoints - means & reassign to the closest cluster

⑥ Repeat ⑤ until each point does not move anymore after mean calculation.

⑦ Calculate the variance of each cluster : a good cluster has small variance (i.e. points are very similar).

Everything is repeated ~ 10K times for biological datasets and the best result is the one where clusters have the lowest total variance.

DESEQ2

- **library normalization** for:

a - library size (n° of reads obtained → problem of sequencing depth usually)

b - library composition → usually a problem when Δ tissues are compared:

Problem #1: adjusting for differences in library sizes

Gene	Sample #1 635 reads	Sample #2 1,270 reads
A1BG	30	60
A1BG-AS1	24	48
A1CF	0	0
A2M	563	2126
A2M-AS1	5	10
A2ML1	13	26

The read counts for each gene in Sample #2 are twice the read counts in Sample #1.

This difference is not due to biology, but to sequencing depth.

However, there is another problem...

RPKM, FPKM, TPM and CPM all deal with this

Problem #2: Adjusting for differences in library composition

Gene	Sample #1 635 reads	Sample #2 635 reads
A1BG	30	235
A1BG-AS1	24	188
A1CF	0	0
A2M	563	0
A2M-AS1	5	39
A2ML1	13	102

The read counts for everything but *A2M* are crazy high in Sample #2

However, the only differentially expressed gene is *A2M*

Assume that only Sample #1 transcribes *A2M*

This means that the 563 reads used up by *A2M* in Sample #1 will be distributed to other genes in Sample #2

① Deseq takes the log of all values

	Sample #1	Sample #2	Sample #3
Gene1	0	10	4
Gene2	2	6	12
Gene3	33	55	200

$\log(0) = -\inf$ in R

$e^{2.3} = 10$

Step 1: Take the log of all the values

	log(Sample #1)	log(Sample #2)	log(Sample #3)
Gene1	-Inf	2.3	1.4
Gene2	0.7	1.8	2.5
Gene3	3.5	4.0	5.3

DESeq2 uses $\log_e$ ("log base *e*"), so these numbers are what we would need to raise *e* to in order to get the original value.

② It makes an average of each row: average log are not easily moved by outliers.

	Sample #1	Sample #2	Sample #3
Gene1	0	10	4
Gene2	2	6	12
Gene3	33	55	200

Avg(Gene3) = 96

Remember that logs are exponents, and in this case they are exponents of *e*, so we have to raise *e* by 4.3 to get a "normal number".

Now convert the average log value for Gene3 into a normal number.

Step 2: Average Each Row → geometric average

	Average of log values
Gene1	-Inf
Gene2	1.7
Gene3	4.3

$e^{4.3} = 73.7$ → this average obtained with log values is smaller because it's less distorted by outliers

③ Filter out genes with too low expression → mean = $-\inf$ ⇒ in this case gene 1.

! This step filters out all genes that have 0 counts in one or more samples ⇒ we are removing those genes transcribed only in some samples (e.g. tissue specific genes).
⇒ **normalization on library composition**

④ Subtract the average log to log(counts): log(reads gene n) - average(log values for gene n) =

$$= \log\left(\frac{\text{reads gene n}}{\text{average for gene n}}\right)$$

⇒ we are dividing the ratio of the reads in each sample to the average across samples. This allows to identify genes that are either higher or lower expressed vs the average

⑤ Calculate the median of the ratios of each sample → the median value is used to avoid extreme genes from swaying the value too much in one direction: genes with huge differences have no more influence on the median than genes with little differences

Actually there are not many genes with huge differences ⇒ we are giving just more influence to moderate differences in gene expression.

⑥ Convert the median to normal values (without log ⇒ e^{median}) we have obtained the **scaling factor for each sample**

⑦ Divide the original read counts by the scaling factor.

• Dispersion estimation

① Calculate dispersion estimates for each gene → BCV^2 = biological coeff of variation

② Plot dispersion over mean normalized counts and fit a curve to it

③ Adjust the dispersion of each gene bringing them towards the value of the curve:

- Genes with higher dispersion vs the model will have their dispersion shrinked
- ↘ lower ↗ enlarged

So dispersion is adjusted borrowing info from genes with a similar expression: the important assumption is that genes with a similar expression are assumed to have similar dispersion too.

There will be some genes for which deseq2 cannot be moved towards the curve: their dispersion is so high that deseq2 assumes that they are not following the model (i.e. the fitted curve).
There outliers are not shrunken ⇒ they are the DE genes: there are other variables accounting for their dispersion which are not comprised in technical or biological variability.

Second important assumption: the majority of the genes is not DE → otherwise we would shrink everything and lose some information.

• Statistical testing

Empirical Bayesian methods are used **to shrink log FC estimates towards zero**. Since this shrinkage is higher when counts are low / dispersion is high, this step is used to remove all those genes whose log FC is not reliable

For statistical testing, it uses a **Wald test for significance**: very similar to a t-test but data is converted into z-score = distance data - expected mean.

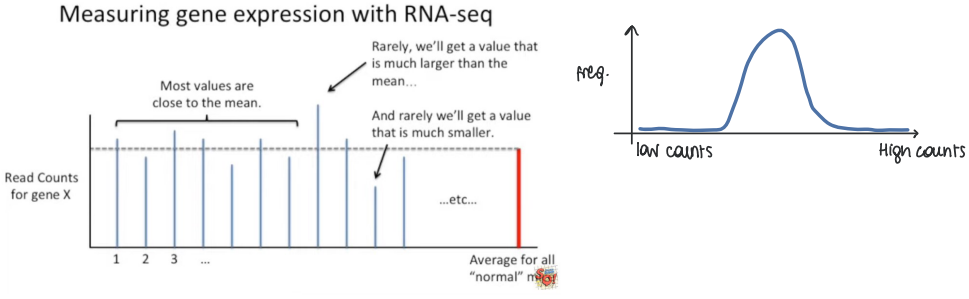
DE genes will have their z-score dropping on the tails of the normal distribution of z-scores.
⇒ we obtain **raw p-values** which are indicative of DE genes.

These **p-values are corrected using Cook's distance**, a measure of how much a single sample is influencing the fitted coeff for a gene.
Genes with a high Cook's distance value are eliminated → due to artifact.

Another **correction** is done **for the multiple testing problem** → **Benjamini-Hochberg correction**

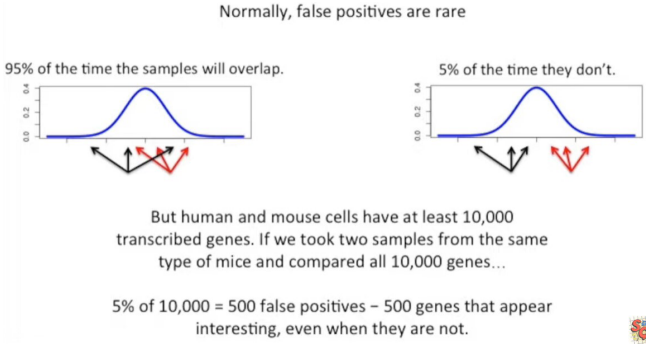
↓

RNA seq is not perfect: we will have read counts for each gene normally distributed

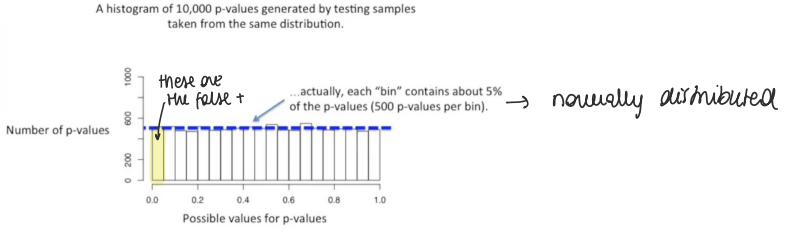


Those rare values are assigned a low p-value, meaning that they seem to come from another distribution, although they are not ⇒ false positives

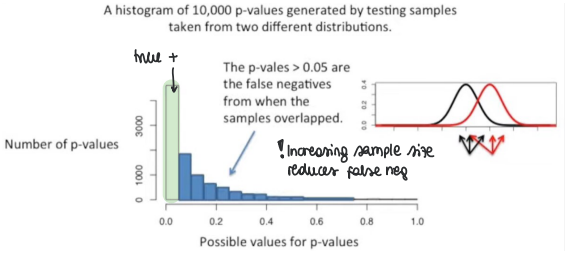
Normally they are rare: for p-value 0,05 → 5% but 5% of 10K genes is still a lot (500 genes that are false +) ⇒ correction

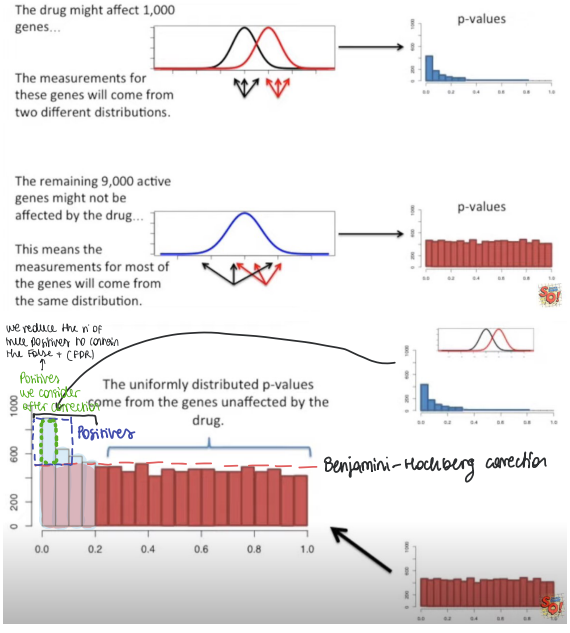


Considering 1 gene measured 10K times, their p-values will be normally distributed:



Instead when we compare ≠ conditions, p-values are distributed differently, with most of them being below 0,05, and some false negatives (>0,05).





Intuitively, p-values for non DE genes will follow the normal distribution, while DE don't

We sum these istograms to obtain the full picture:

By eye we can see that there is a part of the plot which is uniformly distributed.

Benjamini-Hochberg correction "draws" (= sets a cutoff) a line dividing the positives into 2 parts. Only the ones above the line which is a continuous of the normal distribution will be considered

The BH correction method:

1. Order p-values from smallest to longest and rank them
2. The longer p-value will be = to the adj. p-value
3. The others are chosen between → the previous adj. p-value
   → their $\text{p-value} \cdot \frac{\text{tot n° of p-values}}{\text{p-value rank}}$
   ] depending on which one is smaller

1. Order to p-values from smallest to largest.
2. Rank the p-values
3. The largest FDR adjusted p-value... and the largest p-value are the same
4. The next largest adjusted p-value...

a: The previous adjusted p-value = 0.90

...is the smaller of these two options:

b: $0.71 \cdot \left(\frac{10}{8}\right) = 0.89$

p-values:	0.01	0.11	0.21	0.31	0.41	0.51	0.61	0.71	0.81	0.91
rank:	1	2	3	4	5	6	7	8	9	10
adj p-values:									0.90	0.91