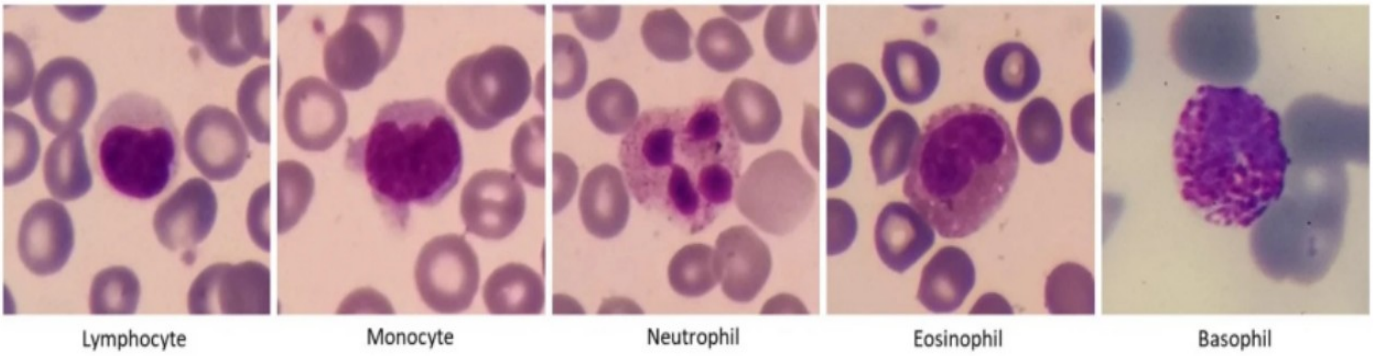
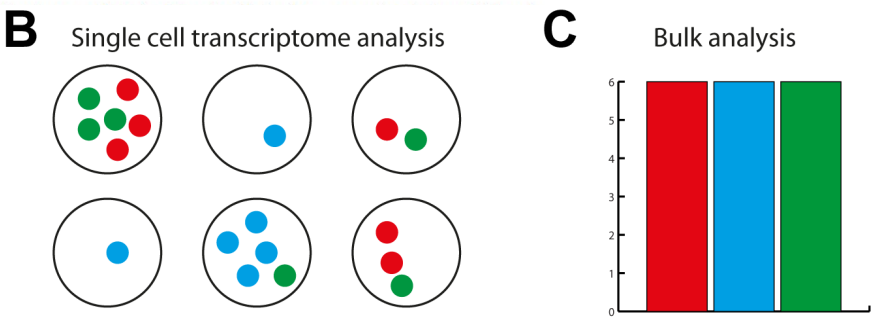
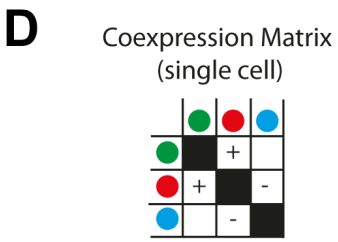
Wozu Single Cell?

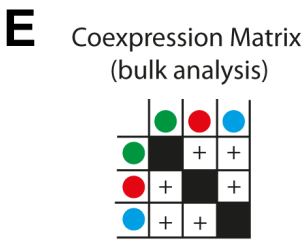
→ **Single Cell RNA-seq ermöglicht Transkriptomanalyse auf Einzellzellebene!**

- Heterogene Zellpopulationen
  - Charakterisierung
  - Identifizierung
  - Krankheiten
- Untersuchung von Co-Expression-Mustern
- Expressionsunterschiede
- Untersuchung seltener Zellpopulationen



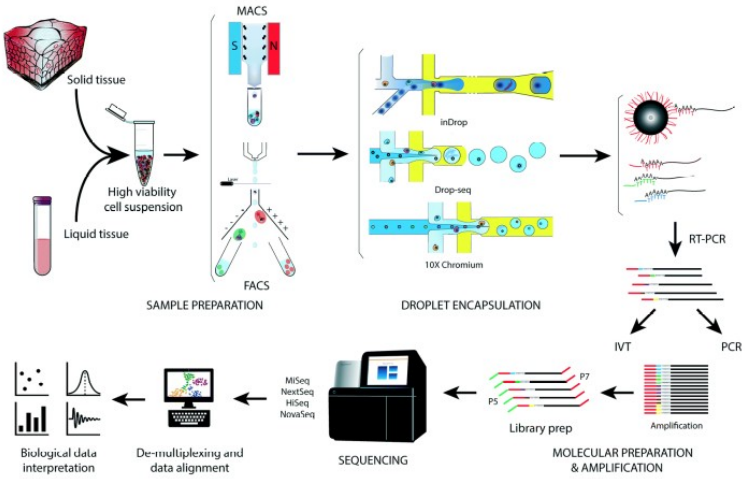






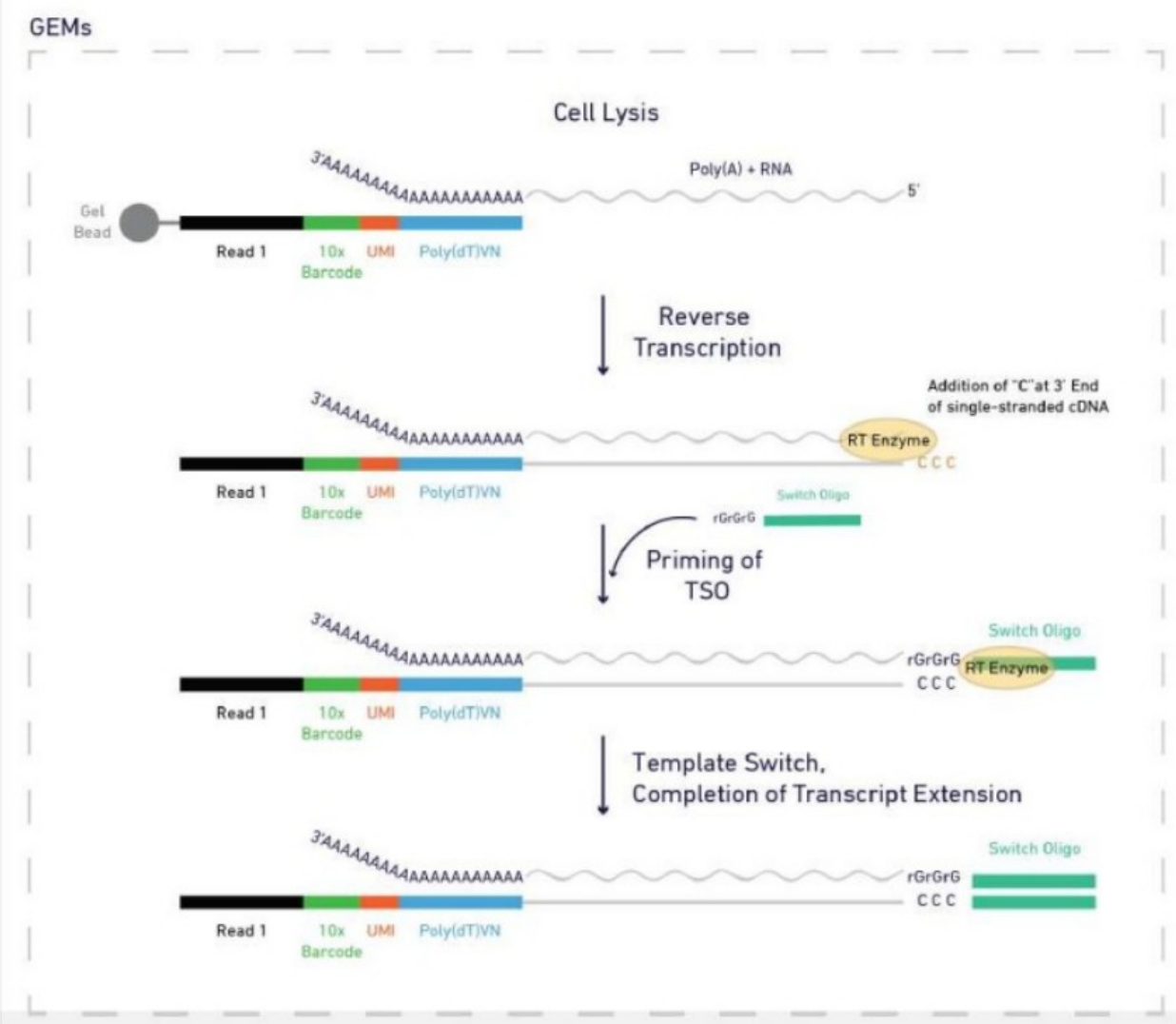
Plot zeigt 6 Zellen mit heterogener Expression.
In der Summe hat man von jedem Gen (rot,grün,blau) jeweils 6 Kugeln.
Bulk Analysen schmeißen alles zusammen und detektieren nur die Summe der Expressionen.
Single Cell detektiert es auf Zellebene.
Single Cell kann Coexpression auf Zellebene detektieren: Rot und grün werden zusammen exprimiert in Zelle 1,3,6. Bulk kann es nicht.

1. Vorbereitung der Zellsuspension
2. Zellsortierung
3. Einkapseln einzelner Zellen in Droplets
4. CDNA-Synthese und Amplifikation
5. Bibliothek
6. Sequenzierung, data alignment und Interpretation

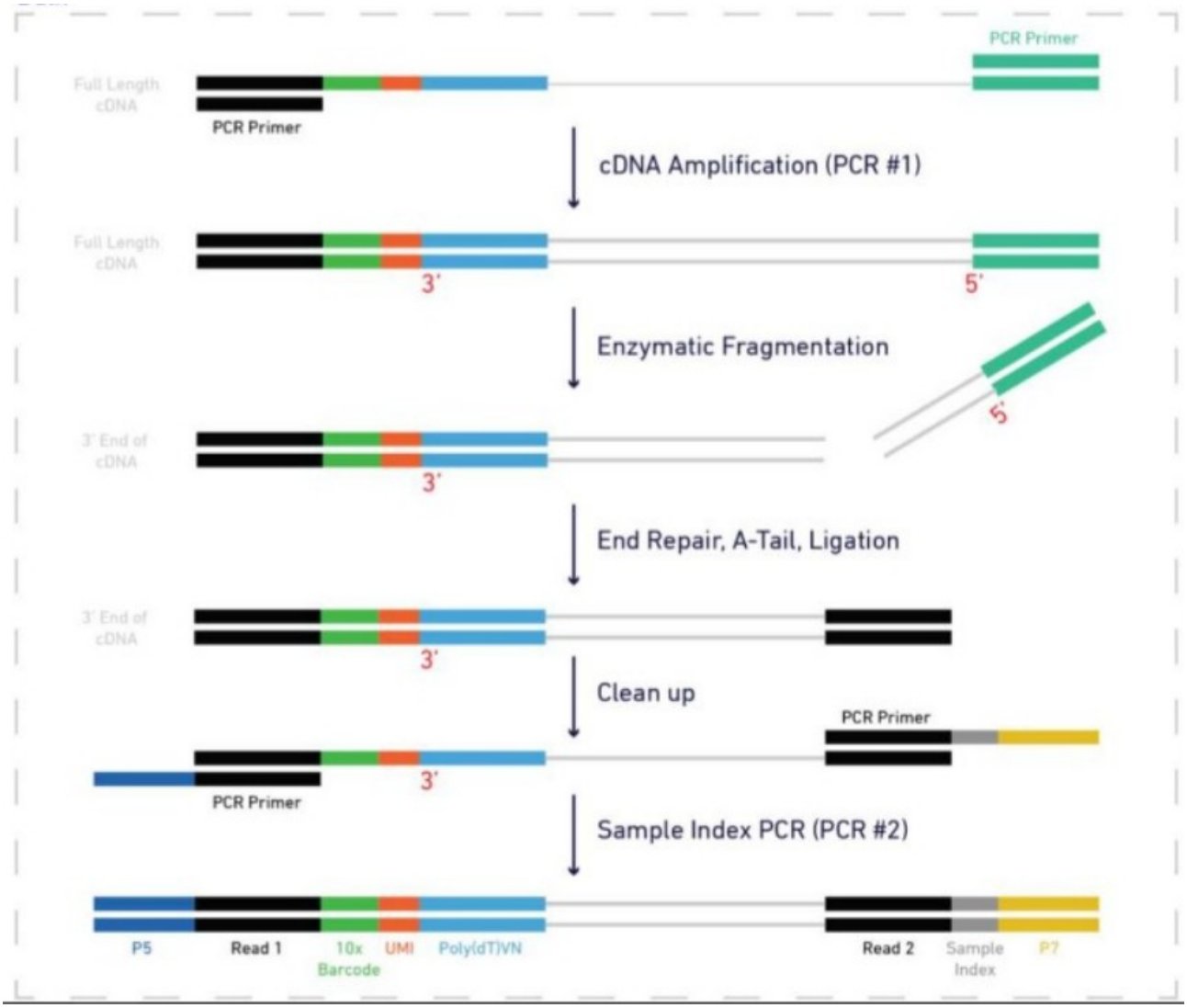


In einem Öl-Medium haben wir Tröpfchen. In diesen Tröpfchen haben wir ein Gel Bead. Wir haben NUR 1 Zelle im Tropfen. Nicht mehr oder weniger. In dem Tropfen sind auch alle Reagenzien für die Lyse der Zelle und Freisetzung der Transkripte

Bei 10 X Genomics kriegen wir innerhalb eines Tubes gewährleistet, dass alle Beads spezifisch verschiedene Barcodes aufweisen.
Diese Barcodes sind für einen Bead identisch, aber andere Beads haben andere Barcodes. → So kann man Transkripte später einer Zelle zuweisen basierend auf dem Barcode.
Man hat UMIs und Poly-dT Tails. Über die Poly-dt-Tails können NUR REIFE mRNAs über ihre Poly-A- Tails binden. Das heißt, nur Sie können genutzt werden.
Zellen werden in Tropfen lysiert und Transkripte freigesetzt. Sie binden dann an die Beads.
Die Single-Cell-Maschinen gewährleisten, dass über ein Micro-Fluid-System ein Tropfen entsteht, welchen nur ein Bead und nur 1 Zelle enthält.
Leere Tropfen und Tropfen mit nur Beads stören nicht.
→ Stören tut es, wenn man mehr als 1 Zelle hat oder fremde Transkripte (nach Zellschaden), weil dann Barcode dann auf zwei Zellen verweist. Man nennt sowas Multiplet.



1. Nur reife mRNAs werden nachgewiesen, da Sie über den Poly-A-Schwanz an Beads binden.
2. Reverse Transkriptase durchgeführt. Dabei wird der Nukleotid-Anteil, der an Beads gebunden war, losgelöst. Hier haben wir es mit einer normalen In vitro Reverse Transkriptase Reaktion zu tun.
Besonderheit: Das verwendete Enzym fügt 3 mal C an das 3-Ende.
Zusätzlich wird dann über die 3 Cs ein Switch Oligo mit Gs angebracht.
Switch Oligos sorgt dafür, dass das 3´-Ende doppelsträngigen Anteil hat. Dieser wird später für Amplifikation benötigt.
Hiernach kann der Tropfen aufgebrochen werden, da jedes Transkript jetzt einen Barcode trägt, der auf eine Zelle verweist.
Wir haben jetzt zwei definierte Enden → ein definiertes Ende am 3-Ende, weil dort Switch-Oligo liegt und der Teil am 5-Ende aus den Bead.
Beide können als Priomer für cDNA-Amplifikationen genutzt werden (PCR).
→ Die cDNAs werden dann fragmentiert.
→ Die mittlere Größe der Inserts lässt sich durch Art der Behandlung steuern.
→ Normalerweise 300 bp Länge für Insert.
→ Enden werden repariert, indem ein A-Schwanz an 3-Enden an Fragmente angebracht wird. Dadurch erhält man neue Primer für Illumina Sequenzierungen.
→ Vorher werden die Samples über PCR-Primer amplifiziert. Dadurch wird auch P5 und P7-Ende angebracht. Die P5 und P7 binden dann bei Illumin direkt an die FlowCell. Der Sample-Index erlaubt verschiedene 10 x Proben zu unterscheiden.



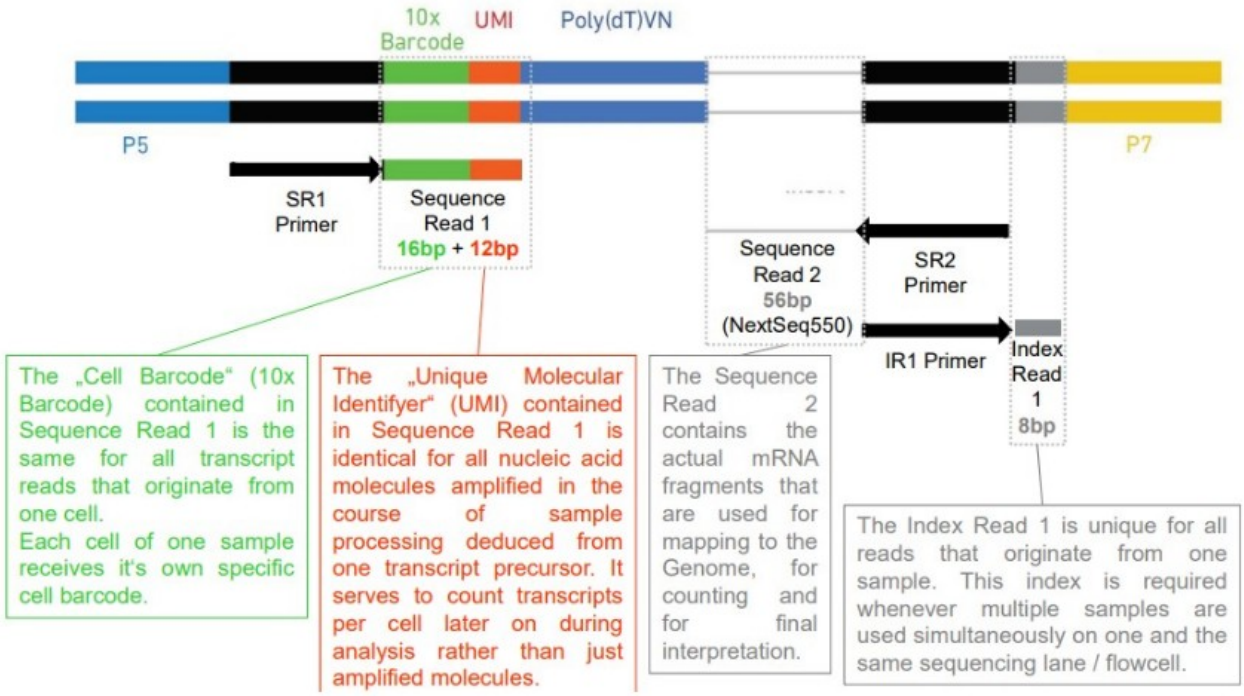
Bei der Sequenzierung mit Illumina wird die Sequenzierung in mehrere Sequenzierungsabschnitte unterteilt.

Das heißt, zuerst wird durch den Sequencing Primer 1 der Bereich mit dem 10 X Barcode und dem Umi abgelesen. Das ist der Sequence Read 1.
→ Der Sequence Read 1 wird normalerweise benutzt, um bereit den Insert abzulesen. Bei 10x Genomics qird stattdessen benutzt, um den 10x Barcode und UMI abzulesen.

10x Barcode → Enthalten in Sequence Read 1. Für alle Transkripte, die aus Was derselben Zelle kommen gleich. Dient dazu, um Transkripte später auf gemeinsame Zellen zurückzuführen. → 16 bp lang

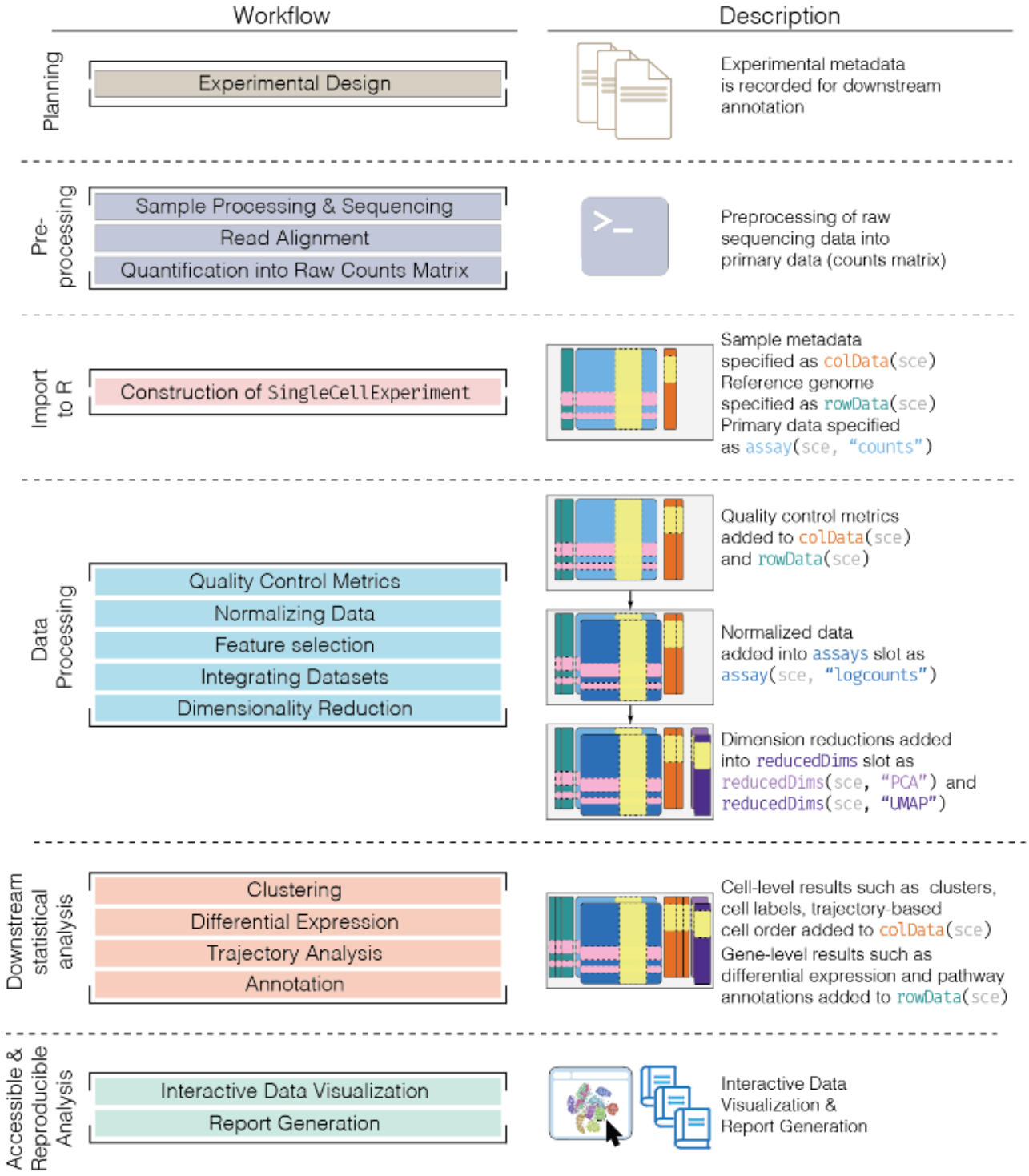
UMI → Unique Molecular Identifier. CDNA wird mit begrenzten Zyklen amplifiziert und die Library wird auch amplifiziert. Wir wollen aber wissen, wie viel Transkripte wir pro Zelle haben. Die Amplifikate interessieren uns nicht, die auf cDNA und Library-Ebene entstehen. Der UMI hilft bei der Unterscheidung, und
weil der UMI ein spezifisches 12 bp Motiv enthält. Jedes Transkript innerhalb der Blase hat sein eigenes einzigartiges UMI-Motiv. Wenn ich ein Transkript 100 mal sequenziere und damit 100 Amplifikate vorfinde, diese aber denselben UMI haben, dann kann ich die 100 Ampflifikate einem Ursprungstranskript zuweisen und zähle die 100 Amplifikate als 1 Transkript. So kann ich Transkripte und nicht Amplifikate zählen.
Nach der Sequenzierung des Barcodes und der UMI wird der zweite Sequence

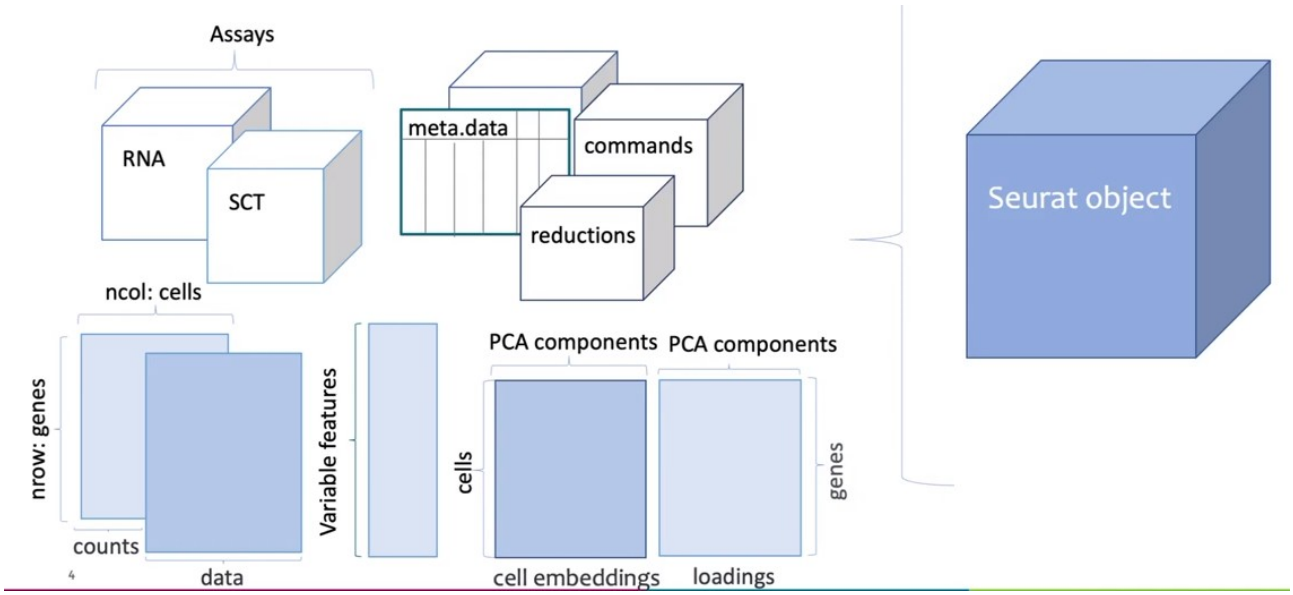
Primer benutzt.
Der liest aus der anderen Richtung den Insert.
→ Normalerweise ist der Read 1 der wichtige Read. Read 2 wird dann nur für Paired End Sequencing benutzt. Hier ist Read 2 wichtig, weil er die unbekannte Sequenz sequenziert.
→ Als drittes gibt es den Index Read Primer. Auch bei Single Cell poolt man verschiedene Proben. Damit kann man Libraries zu den verschiedenen Proben zuteilen.
→ Man teilt also zuerst Reads Proben zu (Index-Read) und dann den einzelnen Zellen (Barcode) und dann einzelnen Transkripten (UMI).



Die Sensitivität ist bei Single Cell geringer als bei Bulk Ansätzen. Man erlaubt sich auf Einzelzellebene zu analysieren. Muss dafür aber Sensitivität einspaaren.

Das hat zur Folge, dass einige RNAs nicht so einfach auf Single Cell Ebene nachzuweisen sind. Wie z.b CD4-Transkripte. Deshalb ist es gut, wenn man über Antikörper CD4-Expression auf Protein-Ebene nachweisen kann.


Workflow
Description
Planning
Experimental Design
Experimental metadata is recorded for downstream annotation
Pre-processing
Sample Processing & Sequencing
Read Alignment
Quantification into Raw Counts Matrix
Preprocessing of raw sequencing data into primary data (counts matrix)
Import to R
Construction of SingleCellExperiment
Sample metadata specified as colData(sce)
Reference genome specified as rowData(sce)
Primary data specified as assay(sce, "counts")
Data Processing
Quality Control Metrics
Normalizing Data
Feature selection
Integrating Datasets
Dimensionality Reduction
Quality control metrics added to colData(sce) and rowData(sce)
Normalized data added into assays slot as assay(sce, "logcounts")
Dimension reductions added into reducedDims slot as reducedDims(sce, "PCA") and reducedDims(sce, "UMAP")
Downstream statistical analysis
Clustering
Differential Expression
Trajectory Analysis
Annotation
Cell-level results such as clusters, cell labels, trajectory-based cell order added to colData(sce)
Gene-level results such as differential expression and pathway annotations added to rowData(sce)
Accessible & Reproducible Analysis
Interactive Data Visualization
Report Generation
Interactive Data Visualization & Report Generation



SeuratObj	S4 [27998 x 9918] (SeuratObject:	S4 object of class Seurat
assays	list [1]	List of length 1
meta.data	list [9918 x 3] (S3: data.frame)	A data.frame with 9918 rows and 3 columns
active.assay	character [1]	'RNA'
active.ident	factor	Factor with 1 level: "Seurat_test_1"
graphs	list [0]	List of length 0
neighbors	list [0]	List of length 0
reductions	list [0]	List of length 0
images	list [0]	List of length 0
project.name	character [1]	'Seurat_test_1'
misc	list [0]	List of length 0
version	list [1] (S3: package_version, num	List of length 1
commands	list [0]	List of length 0
tools	list [0]	List of length 0

assays. Alle Assays, die ich abgespeichert habe - bsp Rawdata und processed data
meta.data: Alle Metadaten zu den Zellen.
active.assay: Welches Assay wird als default benutzt, wenn nichts anderes angegeben wird. DADRAUF ACHTEN!
reductions: Welche Dimensionsreduktionen wurden abgespeichert
graphs: Welche Plots (UMAP) wurden abgespeichert.

Assays:
Spalten _> Zellen
Reihen → Gene.

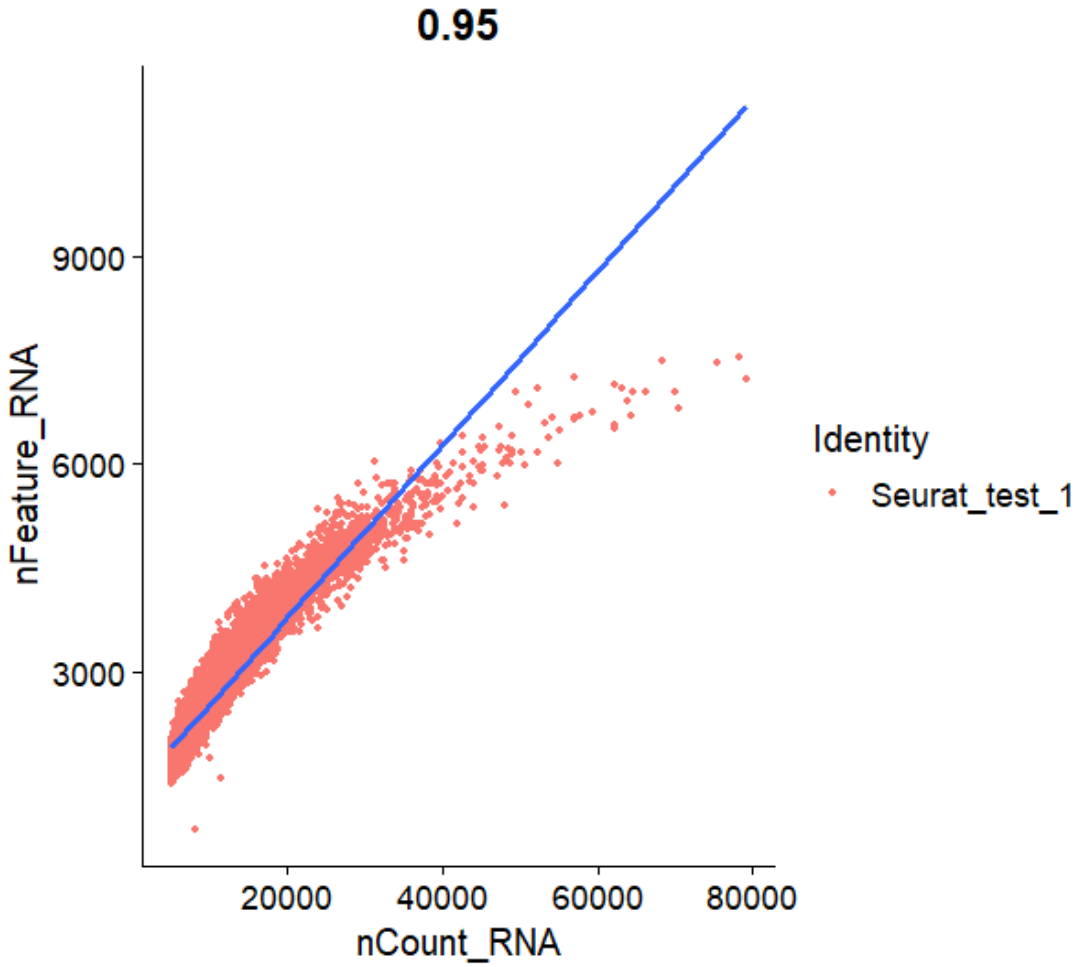
`nFeature_RNA` is **the number of genes detected in each cell**. `nCount_RNA` is **the total number of molecules detected within a cell**. Low `nFeature_RNA` for a cell indicates that it may be dead/dying or an empty droplet. High `nCount_RNA` and/or `nFeature_RNA` indicates that the "cell" may in fact be a doublet (or multiplet)

meta.data
→ Alle Metadaten pro Zelle
→ ncount : Anzahl aller Moleküle in der Zelle
-> nFeature Anzahl einzigartiger Gene pro Zelle
→ mtPercentage: Anzahl an Mitochondrien Genen pro Zelle.

Active.Assay → Aktueller Essay. Aufpassen, dass man nicht ausversehen prozessierte Daten nutzt.

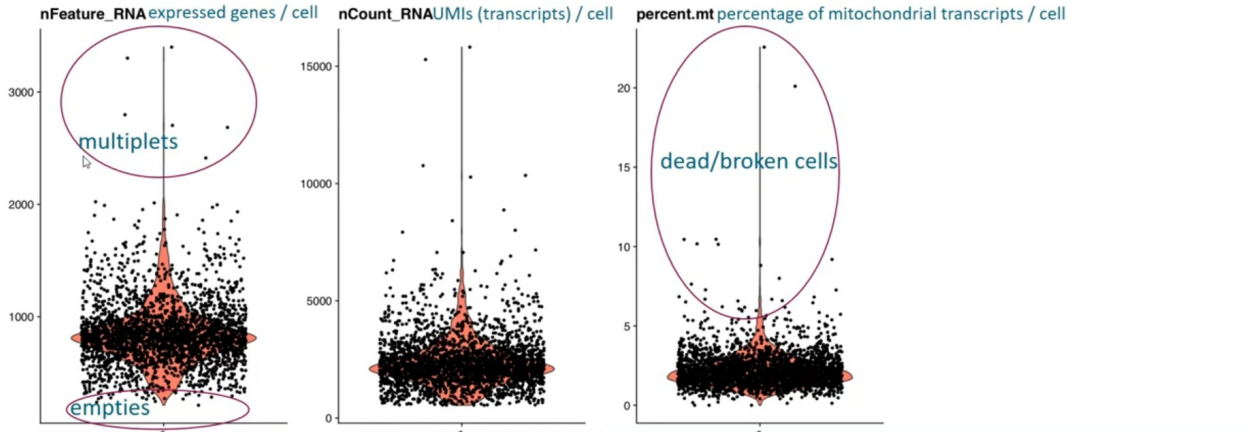
PCAs gespeichert in reductions
Columns sind die PCA Dimensionen
Reihen die Zellen.

QC

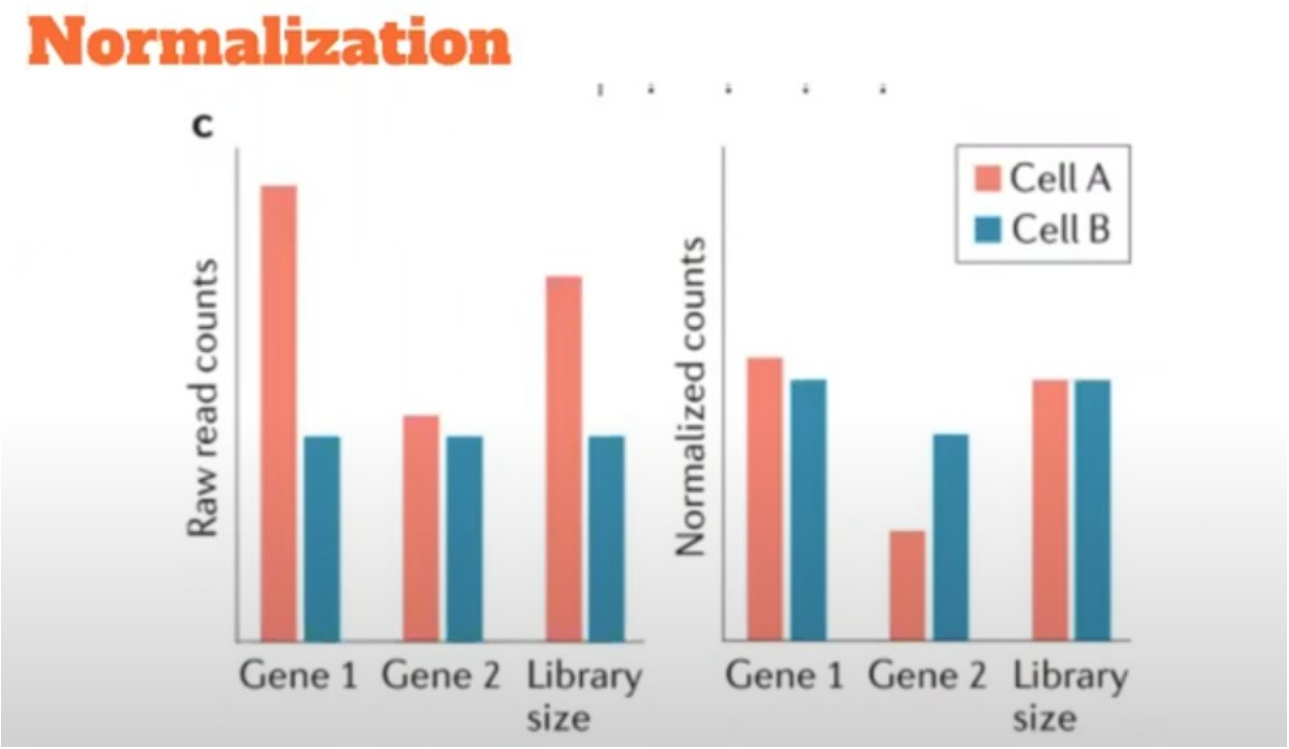


Idealerweise folgt es der Linie.
Biegt nach unten rechts ab → Hohe nCount und geringe nFeatures → Es sind wenig Features, die einfach oft sequenziert wurden.
Oben links → Hohe nFeatures, geringe nCount → Viele nFeatures, aber Sie wurden nicht gut genug sequenziert.

- Empty = no cell in droplet: low gene count (nFeature_RNA < 200)
- Doublet/multiplet = more than one cell in droplet: large gene count (nFeature_RNA > 2500)
- Broken/dead cell in droplet: lot of mitochondrial transcripts (percent.mt > 5%)



Normalisierung in Seurat



Library Size → Anzahl an Reads in der Zelle.

Hier hat Zelle 1 für Gen 1 viel mehr Reads. Es hat aber auch mehr UMIs. Viel mehr wurde sequenziert.
Wir normalisieren, um dieselbe Library Size (Anzahl an Reads) zu erlangen.
So lassen sich die Zellen vergleichen.

Log Normalize

```
data <- t(t(data) / Matrix::colSums(data)) * scale.factor
data <- log1p(data)
return(data)
```

scale.factor ist default 10000

Beispiel:

```
mat <- matrix(1:12, nrow = 3, ncol = 4)
print("Matrix:")
mat
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

gehen einfach von Ganzen Zahlen aus. Wir haben 3 gene und 4 Zellen.

```
t(t(mat) / Matrix::colSums(mat))
```

```
          [,1]      [,2]      [,3]      [,4]
[1,] 0.1666667 0.2666667 0.2916667 0.3030303
[2,] 0.3333333 0.3333333 0.3333333 0.3333333
[3,] 0.5000000 0.4000000 0.3750000 0.3636364
```

t() steht für Transpose. Zeile und Reihe werden vertauscht. ColSums summiert alle Werte innerhalb einer Spalte (z.b alle Werte in Spalte 1).Jeder Wert in der Reihe (vorher die Spalte) wird mit der Summe der jeweiligen Spalte geteilt.

```
Data ← t(t(mat) / Matrix::colSums(mat))*10000
```

```
        [,1]     [,2]     [,3]     [,4]
[1,] 1666.667 2666.667 2916.667 3030.303
[2,] 3333.333 3333.333 3333.333 3333.333
[3,] 5000.000 4000.000 3750.000 3636.364
```

Als Default wird mit dem Wert 10.000 multipliziert.

```
log1p(data)
```

```
        [,1]     [,2]     [,3]     [,4]
[1,] 7.419181 7.888959 7.978539 8.016748
[2,] 8.112028 8.112028 8.112028 8.112028
[3,] 8.517393 8.294300 8.229778 8.199014
```

log1p() berechnet log(1+x) für alle Werte x in der Tabelle. Log() ist dabei der natürliche Log. Wird genutzt, da Expressionswerte sehr klein sind anders als die Zahlen hier. Die +1 gewichtet sie.

TLDR Fassung:

Divide each cell by the total number of molecules measured in the cell

Multiply that number by a scaling factor (i.e. 10000)

Add 1, and take a natural log

RC – Relative Counts → Kein Log genutzt.

Normalize count data to relative counts per cell by dividing by the total per cell. Optionally use a scale factor, e.g. for counts per million (CPM)

```
mat <- matrix(data = rbinom(n = 25, size = 5, prob = 0.2), nrow = 5)
mat
```

```
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    0    2    0    1    2
#> [2,]    1    3    0    1    0
#> [3,]    1    1    1    3    1
#> [4,]    2    1    1    1    0
#> [5,]    0    0    2    2    0
```

mat <- mat / rep.int(Matrix::colSums(mat), diff(mat@p)) * scale.factor

scale.factor ist 1 als Default.

rep.int(Matrix::colSums(mat), diff(mat@p)) bildet die Summe pro Spalte/Zelle und wiederholt den Wert entsprechend der Nicht-0-Werte x mal.

Zum Beispiel:
Für Spalte 1 ist die Summe 4 (1+1+2) und man hat zwei Werte in der zweiten Spalte. Also kriegt man die Zahlen 4,4,4
Für Spalte 2 → 7,7,7,7
Spalte 3 → 4,4,4
Spalte 4 → 8,8,8,8,8
Spalte 5 → 3,3

Insgesamt: 4,4,4,7,7,7,7,4,4,4,8,8,8,8,3,3

Man teilt jetzt alle Nicht-0-Werte durch diese Zahlen.

Für Spalte 1 → ¼,1/4, 2/4
Spalte 2 → 2/7,3/7,1/7,1/7
Spalte 3 → ¼,1/4,2/4
Dasselbe macht man für die anderen Spalten.

Ergebnis:

```
#> 5 x 5 sparse Matrix of class "dgCMatrix"
#>
#> [1,] .    0.2857143 .    0.125 0.6666667
#> [2,] 0.25 0.4285714 .    0.125 .
#> [3,] 0.25 0.1428571 0.25 0.375 0.3333333
#> [4,] 0.50 0.1428571 0.25 0.125 .
#> [5,] .    .         0.50 0.250 .
```

CLR → centered log ratio transformation

log1p(x = x / (exp(x = sum(log1p(x = x[x > 0]), na.rm = TRUE) / length(x = x))))

1. sum(log1p(x = x[x > 0]), na.rm = TRUE) → Bilde den logp1 für alle Gene, die einen expressionswert haben. Also einen Wert ungleich 0 haben. Dann summiert man die Werte.

Bsp.

```
> # Beispielvektor x

> x <- c(1, 2, 3, 4, NA, 5,0)
>
> # Berechnung der Summe der logarithmierten positiven Elemente
> result <- sum(log1p(x[x > 0]), na.rm = TRUE)
>
> # Ausgabe des Ergebnisses
> print(result)
[1] 6.579251
```

2. sum(log1p(x = x[x > 0]), na.rm = TRUE) / length(x = x) → Das berechnet den Mittelwert. Soweit ist Schritt 1.+2 nur Mittelwert Berechnung.

Bsp.

```
> # Beispielvektor x

> x <- c(1, 2, 3, 4, NA, 5,0)
>
> # Berechnung des Durchschnitts der logarithmierten positiven Elemente
> result <- sum(log1p(x[x > 0]), na.rm = TRUE) / length(x)
>
> # Ausgabe des Ergebnisses
> print(result)
[1] 0.939893
```

6.579251/7 = 0.939893

3.exp(x = sum(log1p(x = x[x > 0]), na.rm = TRUE) / length(x = x)) → Wir berechnen aber nicht den Mean, sondern den geometrischen Mittelwert.

Exp(0.939893) = 2.559708

4.x / (exp(x = sum(log1p(x = x[x > 0]), na.rm = TRUE) / length(x = x))) -> Wir teilen alle Werte durch den geometrischen Mittelwert

x/2.559708 → 0.3906696 0.7813391 1.1720087 1.5626782 NA 1.9533478 0.0000000

5. log1p(x = x / (exp(x = sum(log1p(x = x[x > 0]), na.rm = TRUE) / length(x = x)))) → Nutzen logp1()

log1p(x/2.559708) → 0.3297853 0.5773654 0.7756524 0.9410529 NA 1.0829394 0.0000000

Wir zentrieren hier also um den log1p des geometrischen Mittelwertes für alle Expressionswerte.

Hi,
Usually we recommend using log-normalization because it has the advantages as followed:

1.Variance stabilization: scRNA-seq data is inherently noisy, and the counts can have high variability due to technical factors and biological heterogeneity. Log-normalization helps stabilize the variance across different expression levels. It reduces the influence of highly expressed genes and amplifies the signal from lowly expressed genes, allowing for better identification of differentially expressed genes and improved downstream analysis.

2.Interpretability: Log-normalization makes the data more interpretable by transforming the count data into a log scale, which is more in line with how researchers usually think about fold-changes in gene expression. This is especially useful when comparing gene expression across different cell types or conditions, as fold-changes are more meaningful than absolute differences in counts.

3.Handling zero counts: A significant proportion of scRNA-seq data may contain zero counts for some genes in certain cells due to dropout events (i.e., failure to detect lowly expressed genes). Log-normalization typically adds a small pseudocount to the raw counts before taking the log transformation, which helps handle zero counts and reduces the impact of dropout events on downstream analysis.

Using "RC"only divide the count with a cell's total count, but without the log1p step as in log-normalization. It is definitely okay to use RC, but you should do it based on the fact that it is appropriate for your data. You should Not do it because it gives you more DEGs.

Wann CLR nutzen

Contrary to the negative binomial distribution of UMI counts, ADT data is less sparse with a negative peak for non-specific antibody binding and a positive peak resembling enrichment of specific cell surface proteins[Zheng *et al.*, 2022]. The capture efficiency varies from cell to cell due to difference in biophysical properties. Since CITE-seq experiments enrich for a priori selected features, compositional biases are more severe. Analogously to scRNA-seq data, many approaches to normalization exist. We cover the two most widely used ideas methods that require different input data and starting points.
ADT data can be normalized using Centered Log-Ratio (CLR) transformation [Stoeckius *et al.*, 2017].

“**For CITE-seq data, we do not recommend typicalLogNormalization. Instead, we use a centered log-ratio (CLR) normalization**” https://satijalab.org/seurat/archive/v3.0/multimodal_vignette.html

CITE-Seq (Cellular Indexing of Transcriptomes and Epitopes by Sequencing) is a method for performing RNA sequencing along with gaining quantitative and qualitative information on surface proteins with available antibodies on a single cell level.
Antibody-Derived Tags (ADTs).

PCA

Braucht Skalierte Daten + Variable Gene müssen angegeben werden.
Berechnet PC Scores für Gene.

→ Fluch der Dimensionalität: Distanz und Ähnlichkeit geht verloren.
→ Ineffizient mit allen Genen zu arbeiten.
→ Kann nicht in 20000 Dimensionen plotten.
→ Entfernt technische Noise. Wahre Expression hat eine Correlation Structure, Technische nicht.
→ Die ersten paar PC Dim enthalten biologische Unterschiede. Technische haben höhere Dimensionen.