Einzelzell-RNA-Sequenzierungsanalyse mittels Scanpy – Ein Vergleich von Snakemake Workflows

Florian Mai

Eine Abschlussarbeit präsentiert zur Erlangung des
Bachelor of Science

Algorithmische Bioinformatik
Heinrich Heine Universität Düsseldorf
Deutschland

6. September 2021

Danksagung

Ich bedanke mich bei Prof. Dr. Gunnar Klau, für das Ermöglichen dieser Arbeit sowie bei Prof. Dr. Tobias Lautwein für das Übernehmen der Rolle des Zweitkorrektors. Weiterhin bedanke ich mich bei My Ky Huynh für das Entwickeln und Bereitstellen des Seuratworkflows sowie dem Beantworten meiner Fragen.

Abstrakt

Einzelzell-RNA-Sequenzierung erlaubt uns einen Einblick in die Abläufe von individuellen Zellen durch das Erstellen von Genexpressionsprofilen. Für diese Art von Daten müssen neue Werkzeuge und Verarbeitungsmethoden eingesetzt werden, welche sich von älteren RNA-Sequenzierungsverfahren unterscheiden. In dieser Arbeit werden wir zwei in Snakemake implementierte Einzelzell-RNA-Sequenzierungsanalyse-Workflows vergleichen und ihre Ergebnisse gegenüberstellen. Die jeweiligen Hauptkomponenten dieser Workflows sind Scanpy und Seurat, entwickelt in Python und R.

Wir kommen zu dem Ergebnis, dass der neu vorgestellte Workflow eine schnelle und gut skalierbare Alternative mit vergleichbaren analytischen Ergebnissen zum bestehenden Seuratworklow darstellt. Durch den Fokus der eingesetzten Software auf Skalierbarkeit und die Nutzung strukturbedingter Parallelisierung ermöglicht er uns die Analyse großer Datenmengen.

Inhaltsverzeichnis

Liste der Abkürzungen

Ez-RNA-Seq	Einzelzell-RNA-Sequenzierung
mRNA	Messenger ribonucleic acid
DNA	Deoxyribonucleic acid
PCR	Polymerase chain reaction
Bp	Basenpaar
UMI	Unique molecular identifier
DAG	Directed acyclic graph
UMAP	Uniform manifold approximation and projection
PCA	Principal component analysis
PC	Principal component
HVG	Highly variable gene
kNN	k-nearest neighbors
SNN	Shared nearest neighbors
NNS	Nearest neighbor search
PBMC	Peripheral blood mononuclear cell
RAM	Random-access memory
IO	Input / Output
RI	Rand index
ARI	Adjusted Rand index
AMI	Adjusted mutual information
tsv	Tab-separated values

1 Einführung

Die Einzelzell-RNA-Sequenzierungsanalyse (Ez-RNA-Seq.) erlaubt einen Einblick in die Transkriptome individueller Zellen und ermöglicht damit eine bessere Analyse von Zellstadien und Phänotypen. Die erzeugten Daten haben im Vergleich zu Massensequenzierungsverfahren eine höhere Auflösung und können daher unter anderem zum Entdecken neuer Genexpressionsmuster oder Zellsubpopulationen verwendet werden. Jedoch sind die generierten Daten nicht fehlerfrei. Durch technische Fehler bei der Sequenzierung oder durch biologische Fehlsignale der sequenzierten Zellen enthalten sie Störsignale, die vor der eigentlichen Datenanalyse beseitigt werden müssen. Hinzukommt, dass durch die erhöhte Auflösung der Sequenzierung die zu verarbeitende Datenmenge ebenfalls stark ansteigt. Große Datensätze können bis zu mehrere hunderttausend Zellen umfassen, jeweils mit mehreren zehntausend Genen (Angerer et al. 2017).

Gleichzeitig erfreut sich Ez-RNA-Seq. einer stetig wachsenden Beliebtheit und Nutzergemeinschaft. Allein für die Datenanalyse existieren eine Menge von Softwaretools und Paketen[1]. Obwohl einige dieser Softwaretools das gleiche Ziel verfolgen, unterscheidet sich ihre Performanz und ihre Methodik. Dies macht den Vergleich verschiedener Software lohnenswert und stellt gleichzeitig das Motiv dieser Bachelorarbeit dar: Der Vergleich zweier Ez-RNA-Seq.-Analyseworkflows, welche mit dem Workflow-Management-System Snakemake (Mölder et al. 2021) realisiert wurden.

In dieser Arbeit stellen wir einen neuen Ez-RNA-Seq.-Analyseworkflow vor, welcher auf dem Softwareframework Scanpy (Wolf, Angerer und Theis 2018) basiert und in der Programmiersprache Python geschrieben ist. Diesen vergleichen wir mit einem weiteren Snakemake Workflow, welcher auf dem R Paket Seurat V3 (Stuart et al. 2019) aufbaut. Wir werden die strukturellen Unterschiede der Workflows analysieren, sowie die Laufzeiten und analytischen Ergebnisse vergleichen.

[1] *1027 verschiedene Softwaretools, mit Stand vom 6. September 2021, (Zappia, Phipson und Oshlack 2018)*

2 Hintergrundinformationen

Im folgenden Abschnitt werden wir die für das Verständnis dieser Arbeit benötigten Hintergrundinformationen darstellen.

2.1 Einzelzell-RNA-Sequenzierung

Einzelzell-RNA-Sequenzierung wird dazu verwendet, um das Transkriptom einzelner Zellen zu betrachten und ermöglicht damit das Erstellen von Genexpressionsmustern individueller Zellen. Methoden, die in diese Kategorie fallen, müssen jedoch zwei Schwierigkeiten überwinden, welche bei herkömmlichen Sequenzierungsverfahren nicht auftreten: (1) Das Isolieren und Sequenzieren einzelner Zellen einer Zellpopulation und (2) das Amplifizieren von geringen Mengen von mRNA.

Die meisten Ez-RNA-Seq.-Verfahren folgen derselben grundlegenden Idee: Einzelne Zellen werden isoliert und gekapselt, danach wird die Lyse herbeigeführt und mittels reverser Transkription aus der messenger-Ribonukleinsäure (englisch: messenger ribonucleic acid, mRNA) im Lysat komplementäre DNS (englisch: complementary DNA, cDNA) hergestellt. Die cDNA wiederum wird angereichert, zum Beispiel durch eine Polymerase-Kettenreaktion (englisch: polymerase chain reaction, PCR) oder mittels In-vitro-Transkription. Letztlich wird die cDNA sequenziert (Kolodziejczyk et al. 2015).

Der in dieser Arbeit verwendete Datensatz wurde mittels 10X Genomics Chromium Controller vorbereitet und anschließend mit einem NextSeq 550 sequenziert. Wir werden deshalb die Funktionsweise des für uns relevanten Chromium Controller exemplarisch erläutern und darstellen.

Dieser baut auf einer Microdroplet basierten Herangehensweise auf und besitzt damit einen hohen Durchsatz von Zellen. Zellen werden mittels Emulsion zusammen mit Gelperlen in einzelne Öl-Tröpfchen (englisch: droplets) isoliert (Abbildung 1 A). Die Gelperlen sind mit Oligonukleotiden versehen, welche aus mehreren Teilen bestehen, die verschiedene Funktionen übernehmen: Sie dienen (1) als PCR-Primer und Sequenzierungsadapter, (2) als 14 Basenpaar (Bp) langer Barcode zum Identifizieren des Droplets und damit der Zelle, (3) als eine 10 Bp lange, einzigartige Kennung (englisch: unique molecular identifier, UMI) zum Markieren der unterschiedlichen cDNA und (4) mittels 30 Bp langer Oligo-dT Sequenz als Primer für die reverse Transkription der mRNA (Zheng et al. 2017).

Um nach der PCR zurückzuverfolgen, welche mRNA von welcher cDNA exprimiert wurde, wird die cDNA durch UMIs markiert. Nach der PCR wird davon ausgegangen, dass alle mRNA-Moleküle mit der gleichen UMI von derselben cDNA abstammen. UMIs ermöglichen damit die Unterscheidung zwischen amplifizierten Kopien desselben mRNA-Moleküls und Transkripten desselben Gens. (Luecken und Theis 2019; Sena et al. 2018)

Zellen werden während des Isolationsprozesses nur verdünnt geladen, um dem Effekt von Multiplets entgegenzuwirken. Multiplets sind Droplets, welche mehr als eine Zelle enthalten und somit alle nachfolgenden Schritte, sowohl in der Sequenzierung als auch in der Analyse,

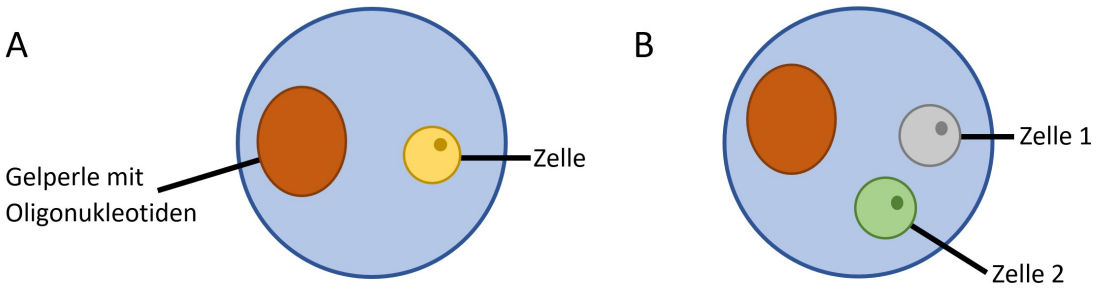


Abbildung 1: Visuelle Darstellung von Droplets. A: Ein korrekt erzeugtes Droplet mit einer Zelle. B: Ein Droplet mit zwei Zellen, ein Doublet

stark beeinträchtigen können (Abbildung 1 (B)). Ein Multiplet wird am Ende der Sequenzierung in den Daten als ein einzelner Barcode auftreten, dessen Expressionsmuster jedoch ein hybrides Transkriptom darstellt (Sun et al. 2021).

Nach der Zellisolation wird direkt die Lyse eingeleitet. Die Gelperlen setzen ihre Oligonukleotide frei und die reverse Transkription beginnt. Jedes daraus entstehende cDNA Molekül ist mit einer UMI sowie einem gemeinsamen Barcode pro Droplet markiert. Nun wird das Droplet aufgelöst und die markierte cDNA für die PCR gebündelt. Letztlich wird die amplifizierte cDNA für ein geeignetes Short-Read-Sequenzierungsverfahren vorbereitet und anschließend sequenziert (Zheng et al. 2017).

Die Sequenzierungsdaten werden von der Analysepipeline CellRanger verarbeitet. CellRanger führt ein Sequenzalignment durch und verknüpft die UMIs zu ihrem jeweiligen Barcode und Gen. Die UMIs werden anschließend gezählt und es entsteht eine Gen-Barcode Matrix, in der ein Eintrag darstellt, wie oft ein Gen von welchem Barcode exprimiert wurde. (Zheng et al. 2017, Figure 1f). Aufgrund der geringen Erfassungseffizienz ist ein Großteil der Matrix jedoch leer bzw. 0. Nach AlJanahi, Danielsen und Dunbar 2018, Tabelle 1 hat der Chromium Controller eine Erfassungsrate von ca. 14% des Transkriptoms einer Zelle. Die von CellRanger generierte dünnbesetzte Matrix dient uns als Input für unsere Analysepipelines (Tabelle 1).

	Barcode 1	Barcode 2	...	Barcode N
Gen 1	2	0	...	3
Gen 2	0	0	...	7
⋮	⋮	⋮	⋱	⋮
Gen M	5	0	...	0

Tabelle 1: Beispielhafte Ausgabematrix der CellRanger Pipeline

2.2 Snakemake

Snakemake (Mölder et al. 2021) ist ein dateibasiertes Workflow-Management-System und ermöglicht uns, Datenanalysen in einem reproduzierbaren, transparenten und adaptierbaren

```
rule read_raw_data:
    input:
        expand('resources/{{projectName}}/')
    output:
        expand('results/{{projectName}}/anndata_files/whole_basic_filtered.h5ad')
    params:
        min_genes = config['min_genes_per_cell'],
        min_cells = config['min_cells_per_gene'],
    script:
        'scripts/ReadRawData.py'
```

Abbildung 2: Beispiel einer Snakemakeregel, Auszug aus der Snakefile des Scanpyworkflows. Neben input, output, params und script wird mit *projectName* auch ein Platzhalter verwendet.

Umfeld durchzuführen. Snakemake definiert einen Workflow durch eine Abfolge von Regeln, die in einer Snakefile angegeben werden. Jede Regel beschreibt, wie eine Reihe von Ausgabedateien aus einer Menge von Eingabedateien generiert werden kann (Abbildung 2). Die Transformation von Input nach Output kann dabei zum Beispiel durch externe Skripte oder Shell-Kommandos erfolgen. Regeln werden verknüpft, indem für jede Eingabedatei eine Regel gesucht wird, die diese als Ausgabe erzeugt. Dies wird rekursiv durchgeführt, bis entweder alle Eingabedateien von anderen Regeln erzeugt werden oder die benötigte Datei bereits auf der Festplatte vorhanden ist. Dabei erstellt Snakemake einen gerichteten azyklischen Graphen (englisch: directed acylic graph, DAG). Der DAG bestimmt die Reihenfolge von Regeln und ob diese parallel ausgeführt werden können (Abbildung 3 und 8 in Abschnitt 3.1 bzw. 3.2).

Beide in dieser Arbeit vorgestellten Workflows arbeiten ausschließlich mit externen Skripten. Um in diesen auf die in der Snakefile definierten Parameter wie Ein-/Ausgabedatei zugreifen zu können, reicht Snakemake ein Objekt in das Skript herein, von dem aus auf die Werte zugegriffen werden kann. Snakemake erlaubt auch zusätzliche Parameter zu übergeben sowie eine Konfigurationsdatei zu verwenden, mit der projekt- bzw. jobabhängige Variablen definiert werden können. Wir benutzen dies, um probenspezifische Werte zu übergeben, die von Experiment zu Experiment unterschiedlich sein können.

Snakemake erlaubt das Verwenden von Platzhaltern, um Dateien und Parameter dynamisch zu bestimmen. So können Regeln generisch gestaltet werden und bieten damit, zum Beispiel, eine hervorragende Möglichkeit projektabhängige Ordnerstrukturen zu realisieren.

Jede Regel in Snakmake kann mittels Conda-Integration (Abschnitt 2.3) ihre eigene vordefinierte Umgebung verwenden. Dies erlaubt zum Beispiel das Verwenden verschiedener Python- oder R-Versionen zwischen Regeln oder eine Laufzeitoptimierung durch spezifische Paketversionen.

2.3 Conda

Conda (Distribution 2016) ist ein Paketmanager und Umgebungsverwaltungssystem. Als solches erlaubt es uns mehrere Umgebungen zu definieren und verschiedene Versionen derselben Software zu installieren, sowie zwischen Umgebungen zu wechseln. In einer Umgebung kön-

nen einfach neue Pakete installiert oder aktualisiert werden, wobei Abhängigkeiten automatisch aufgelöst und mit installiert werden. Umgebungen können über .yaml Dateien definiert und direkt aus diesen erzeugt werden.

Der Scanpyworkflow arbeitet in einer einzigen Umgebung, die vor dem Starten von Snakemake erstellt und aktiviert werden muss. Dafür liegt dem Workflow eine .yaml Datei bei. Der Seuratworkflow hingegen spezifiziert mittels Conda-Integration in Snakemake an jeder Regel, welche Umgebung benutzt werden soll. Die Umgebungen werden automatisch aus .yaml Dateien erzeugt und Snakemake wechselt selbständig je nach Regel zwischen diesen hin und her.

In Kombination mit Snakemake sorgt das Festlegen spezifischer Versionen der verwendeten Software in Conda somit für die Reproduzierbarkeit der Ergebnisse dieser Arbeit.

2.4 Standardverfahren

Die meisten Ez-RNA-Seq.-Datensätze durchlaufen während der Analyse dieselben Schritte, welche man grob in zwei Bereiche einteilen kann: Die Vorverarbeitung zum Filtern von Störsignalen und Vorbereiten der Daten sowie die eigentliche Analyse. Aufgrund der vielen Softwaretools und der stetigen Forschung im Bereich Ez-RNA-Seq. ist es jedoch schwierig, ein standardisiertes Verfahren zu definieren, da sich die Methodik bzw. die eingesetzten Verfahren oft ändern (Luecken und Theis 2019). Luecken und Theis stellen daher in ihrer Arbeit "Current best practices in singlecell RNAseq analysis: a tutorial" eine Reihe von Erfolgsmethoden vor, welche sich bei der Analyse von Ez-RNA-Seq.-Daten bewährt haben.

In der Qualitätskontrolle werden normalerweise drei Metriken verwendet, um sicherzustellen, dass jeder Barcode mit einer vollständigen funktionsfähigen Zelle übereinstimmt. Diese Metriken sind (1) die Gesamtsumme aller UMIs pro Barcode (count depth), (2) die Anzahl der verschiedenen Gene pro Barcode und (3) der prozentuale Anteil der mitochondrialen RNA pro Barcode. Diese drei Metriken können, in Kombination mit geeigneten Grenzwerten, einen Großteil ungültiger Barcodes entfernen. Ein hoher Anteil mitochondrialer RNA weist zum Beispiel auf eine Zelle hin, welche sich zum Zeitpunkt der Isolation / Sequenzierung bereits in der Lyse befand oder deren Zellmembran beschädigt ist. Eine hohe count depth und viele verschiedene Gene weisen aufgrund des hybriden Transkriptoms auf ein Doublet hin. Diese können mit einer einfachen Schranke für die count depth gefiltert werden. Unsere Workflows setzten jedoch auf komplexere Methoden der Doubletklassifizierung.

Um die Expression zwischen Barcodes vergleichbar zu machen, werden die Daten nach der Filterung für gewöhnlich normalisiert. Ein oft dafür verwendetes Verfahren ist das 'count depth scaling'. Dabei werden die Zellen der Gen-Barcodes-Matrix erst durch die count depth des jeweiligen Barcodes geteilt und anschließend mit einer Potenz von 10 multipliziert. Oft folgt auf die Normalisierung noch die Logarithmierung mit $\log(x + 1)$. Die Logarithmierung hat unter anderem den Vorteil, dass auf der Distanz zwischen log-transformierten Expressionswerten die Änderung der Expressionsrate gemessen werden kann.

Anschließend werden die Daten mit verschiedenen Methoden in ihrer Dimension reduziert. Zum einen über das Selektieren von hoch variablen Genen (HVG). Dies sind Gene, welche zwischen Barcodes stark differenziell exprimiert werden. Zusätzlich werden die Daten durch Algorithmen wie PCA (principal component analysis) oder UMAP (uniform manifold approximation and projection) weiter reduziert. Dies hat den Grund, dass sich die unterliegenden Expressionsmuster bzw. Strukturen der Daten sich durch wesentlich weniger Datenpunkte beschreiben lassen als der ursprüngliche Datensatz besitzt. Die Reduktion wird sowohl für die Visualisierung in 2- bzw. 3D Grafiken durchgeführt (UMAP) als auch für das Zusammenfassen der Daten für die folgende Analyse (PCA).

Sollte der Datensatz mehrere Proben umfassen, welche zum selben Experiment gehören, dann müssen diese vor der Analyse noch integriert werden. Die Datenintegration wird meist zusammen bzw. in Kombination mit der Dimensionsreduktion durchgeführt. Die unterschiedlichen Umgebungen der verschiedenen Proben können sich auf die Messung des Transkriptoms der Barcodes auswirken, es entsteht der sogenannte Batcheffekt. Die Datenintegration versucht diesem entgegenzuwirken und somit gleiche Barcodes von verschiedenen Proben zusammenzufügen.

Die nun gefilterten und reduzierten Daten können für die eigentliche Analyse verwendet werden. Hier gibt es eine Auswahl von verschiedenen Analysen die durchgeführt werden können, zum Beispiel Clusteranalyse, Clusterannotation, Kompositionsanalyse oder auch differenzielle Genexpressionsanalyse. Wir beschreiben kurz die Methoden, welche beide Workflows implementieren: Clusteranalyse und differenzielle Genexpressionsanalyse.

Bei der Clusteranalyse werden Barcodes aufgrund ihres Expressionsmusters gruppiert und dadurch gleiche Zelltypen zusammengeführt. Beide Workflows dieser Arbeit verwenden für das Clustering eine graphbasierte Herangehensweise und führen auf kNN Netzwerken eine sogenannte Community Detektion durch. Es handelt sich damit nicht um das klassische Clustern, sondern um eine Partitionierung eines Graphen.

Die differenzielle Genexpressionsanalyse folgt auf das Clustering. Ihr Ziel ist es, die Frage zu beantworten, ob sich Barcodes vom gleichen Typ (und damit innerhalb eines Clusters) aber in unterschiedlichen Probenstadien voneinander unterschieden. Dies wird erreicht, indem ein Cluster in die Barcodes der unterschiedlichen Stadien unterteilt wird und diese miteinander verglichen werden.

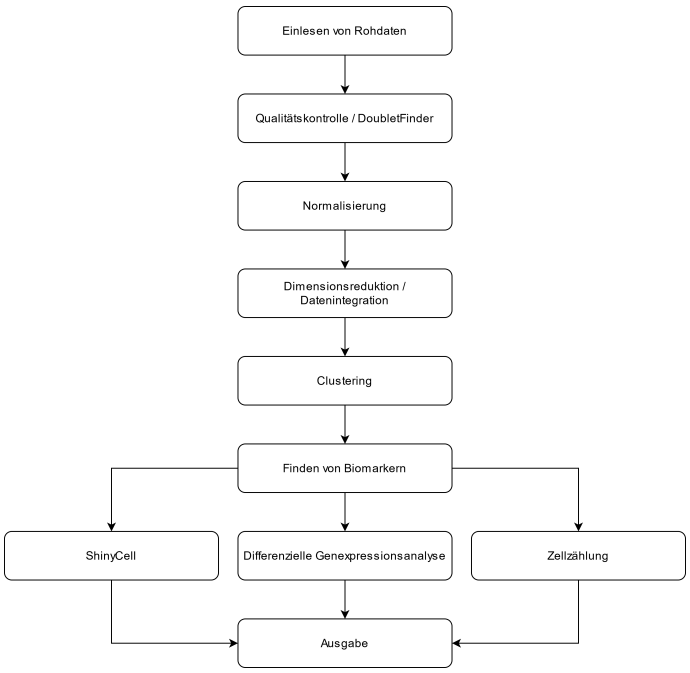


Abbildung 3: Visualisierung des Seuratworkflows. Der original von Snakemake generierte DAG sowie eine Auflistung aller Regeln des Workflows kann im Anhang aus Abbildung 16 entnommen werden.

3 Einzelzell-RNA-Sequenzierungsanalyse Workflows

In diesem Abschnitt werden wir beide Analyseworkflows vorstellen, bevor wir sie in Abschnitt 4 vergleichen. Alle nachfolgenden Abbildungen basieren auf dem Datensatz Maus67k, beschrieben in Abschnitt 4.1.

Der Quellcode der beiden Workflows kann unter

https://git.hhu.de/myhuy100/wggc-single-cell/-/tree/scanpyBSc/ für den Seuratworkflow und

https://git.hhu.de/flmai102/bachelor-florian-mai für den Scanpyworkflow gefunden werden.

3.1 Seurat Workflow

Der in diesem Abschnitt erläuterte Snakemakeworkflow basiert auf dem Softwarepaket Seurat V3 (Stuart et al. 2019), geschrieben in R. Seurat ist ein umfassendes Analyseframework für Ez-RNA-Seq.-Daten und bietet daher eine weitreichende Auswahl von Funktionen zum Filtern, Transformieren, Analysieren und Visualisieren der Daten an. Der hier vorgestellte Workflow baut auf der bereits in Abschnitt 2.4 erläuterten Standardprozedur auf und umfasst die Schritte Qualitätskontrolle, Normalisierung, Datenintegration, Dimensionsreduktion, Clusteranalyse und differenzielle Genexpressionsanalyse. Ebenso kann der Workflow mehrere Proben eines Experimentes verarbeiten, diese zusammenfügen sowie verschiedene Zustände zwischen den Proben vergleichen. Der Workflow kann sowohl auf einem einzelnen Rechner als auch auf einem Clustersystem ausgeführt werden.

Abbildung 3 zeigt, dass die Struktur des Workflows stark linear ist. Unabhängig von der Anzahl der Proben werden fast alle Regeln hintereinander ausgeführt. Erst nach dem Auffinden der Biomarker können die letzten Schritte parallelisiert werden. Die Linearität entsteht, weil im ersten Schritt eine Liste von Objekten erstellt wird, über die in den nachfolgenden Regeln seriell iteriert wird.

3.1.1 Vorverarbeitung

Qualitätskontrolle Der Workflow beginnt mit dem Einlesen der von CellRanger erstellten Gen-Barcode-Matrix durch die in Seurat inkludierte Lesefunktion. Diese vollzieht auch die erste Qualitätskontrolle, indem Barcodes, welche weniger als 200 Gene exprimieren, nicht übernommen werden. Auch werden Gene, die in weniger als drei Barcodes exprimiert werden, exkludiert. Die Lesefunktion erstellt ein Seuratobjekt, die primäre Datenstruktur welche Seurat während der Verarbeitung der Daten verwendet. Anschließend wird, falls nötig, das Seuratobjekt in einzelne Proben unterteilt und jeder Barcode mit dem jeweiligen Probennamen indiziert. Wir erhalten eine Liste von Seuratobjekten; diese wird ab jetzt fortlaufend bis zur Integration im Workflow verwendet und zwischen den Regeln als eine einzige .rds Datei gespeichert. Der nächste Schritt der Qualitätskontrolle umfasst das Berechnen der Standardmetriken: Anzahl der verschiedenen Gene pro Barcode, Summe der exprimierten Moleküle pro Barcode und prozentualer Anteil von mitochondrialer RNA pro Barcode. Der Workflow filtert nur basierend auf dem prozentualen Anteil der mitochondrialen RNA, die Obergrenze wird aus der Snakemake Konfigurationsdatei ausgelesen. Als Veranschaulichung erstellt der Workflow sechs Diagramme für die Qualitätsmetriken, jeweils drei vor und drei nach der Filterung (Abbildung 4).

Anschließend werden mit DoubletFinder (Murrow und Gartner 2019) Doublets detektiert und entfernt. DoubletFinder benötigt a priori Wissen in Form einer Schätzung des erwarteten Anteils von Doublets an der Probe. Dieser Wert wird ebenfalls aus der Snakemake Konfigurationsdatei eingelesen und kann so von uns spezifiziert werden. Die Anzahl von Doublets in einer Probe hängt stark von deren Größe sowie von dem Gerät bzw. dem Verfahren ab, welches zur Isolation der Barcodes verwendet wurde. Bevor DoubletFinder angewendet wird, muss das Seuratobjekt noch mit *SCTransfrom* (Hafemeister und Satija 2019) vorbereitet werden. SCTransform führt eine Reihe von Schritten durch: Es normalisiert die Matrix, selektiert hoch variable Gene und skaliert die Matrix, anschließend werden die neuen Ergebnisse neben den bereits vorhandenen Daten im Seuratobjekt gespeichert. Gleichzeitig kann SCTranform auch vorher berechnete Eigenschaften, wie zum Beispiel die Standardmetriken, mittels linearer Regression filtern. Der Workflow benutzt dieses Feature, um nochmals Barcodes mit einem zu hohen Anteil mitochondrialer RNA auszusortieren. Nach dem Detektieren, aber vor der eigentlichen Filterung, wird in einer UMAP-Grafik dargestellt, welche Barcodes als Doublet identifiziert wurden (Abbildung 5).

Wir gehen kurz auf die Funktionsweise von DoubletFinder ein. DoubletFinder arbeitet mit

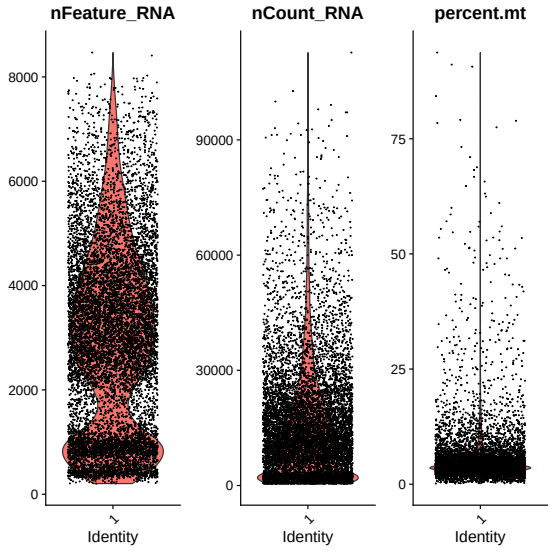


Abbildung 4: Violinendiagramm für die drei Standardmetriken.
nFeature_RNA zeigt die Anzahl der verschiedenen Gene pro Barcode, nCount_RNA ist die Gesamtexpression pro Barcode und percent.mt stellt den prozentualen Anteil von mitochondrialer RNA pro Barcode dar.
Probe 1 des Datensatzes, vor der Filterung.

künstlichen, aus den echten Daten generierten Doublets. Diese werden erzeugt, indem die Expressionsmuster zweier zufällig gewählter Barcodes gemittelt werden. Als Nächstes werden die künstlichen Doublets mit den echten Daten gemischt und erneut vorverarbeitet. Nun wird für jeden Barcode ein Doublet-Score berechnet. Der Score eines Barcodes basiert auf dem Verhältnis von künstlichen Doubltes zu seinen k-nächsten-Nachbarn im PCA-Raum. Ein großer Anteil von künstlichen Doublets bedeutet, dass ein Barcode ihnen ähnelt, vermutlich, da er selber ein Doublet repräsentiert. k wird von DoubletFinder selbst automatisch bestimmt, der Benutzer kann jedoch die Anzahl der Dimensionen des PCA-Raumes bestimmen. Letztlich werden Barcodes nach ihrem Doublet-Score sortiert und die ersten x Barcodes entfernt. x ist dabei der erwartete prozentuale Anteil von Doublets an der Probe (Murrow und Gartner 2019; Xi und Li 2021)

In diesem Workflow wird vor dem Ausführen von DoubletFinder für jede Probe ein Ellenbogendiagramm erstellt. Mit diesem können wir die Anzahl relevanter PCs ermitteln und für die nachfolgende Regel mit DoubletFinder in unsere Konfigurationsdatei eintragen (Abbildung 6).

Normalisierung Als nächstes werden die gefilterten Seuratobjekte mit Metadaten wie z.B. den verschiedenen Probenzuständen annotiert und anschließend auf die ursprünglichen Expressionsdaten zurückgesetzt. Die Objekte enthalten nun wieder eine Gen-Barcode-Matrix, die jedoch mehrfach gefiltert wurde.

Für die bevorstehende Dimensionsreduktion und, falls nötig, Datenintegration müssen die Daten erneut normalisiert werden. Es kommt wieder SCTransform zum Einsatz. Im Gegensatz zu Scanpys einfacher *regress_out* Funktion benutzt SCTransfrom regulierte negative binomi-

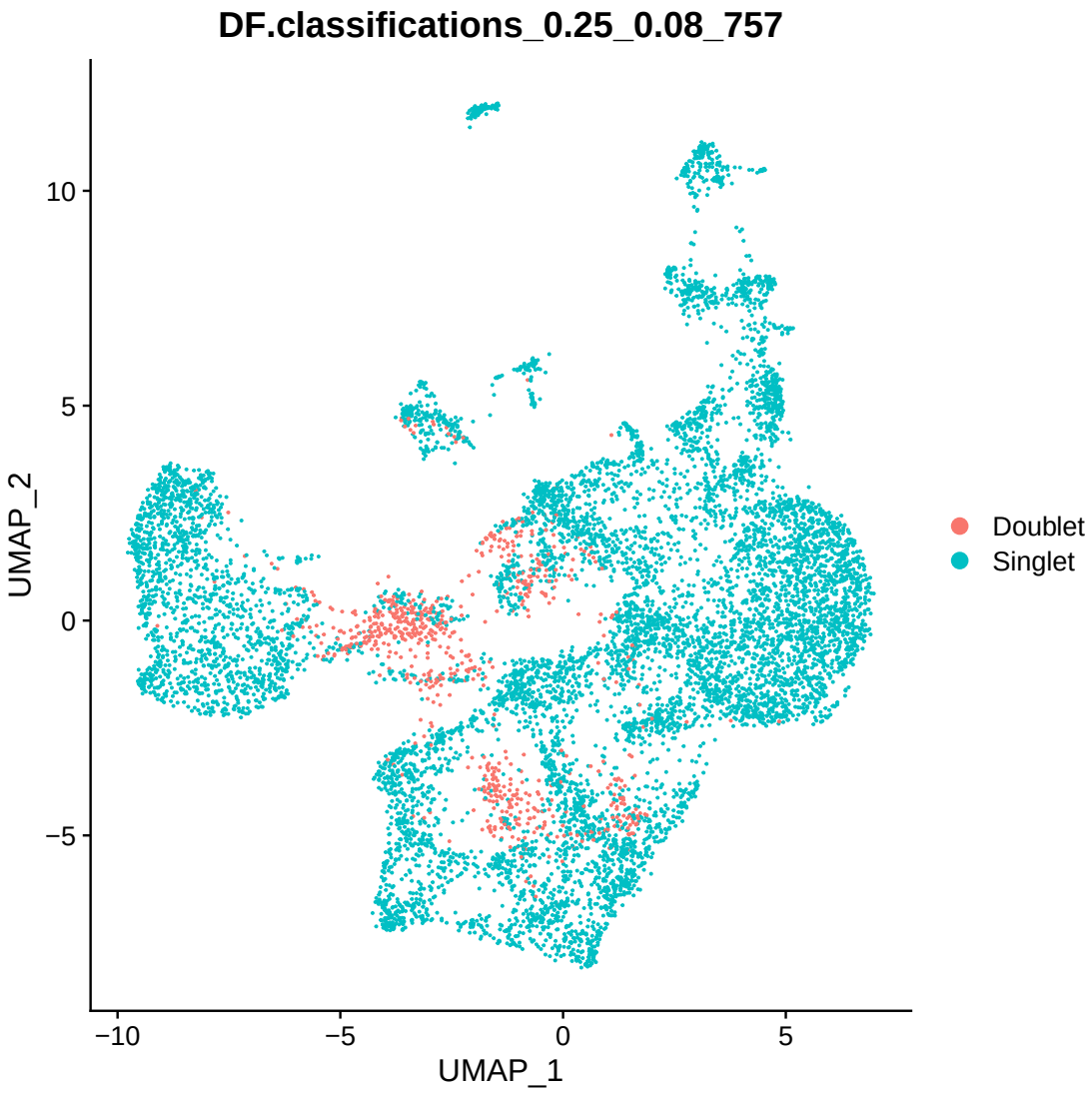


Abbildung 5: Darstellung erkannter Doublets in einem UMAP-dimensionsreduzierten Streudiagramm.

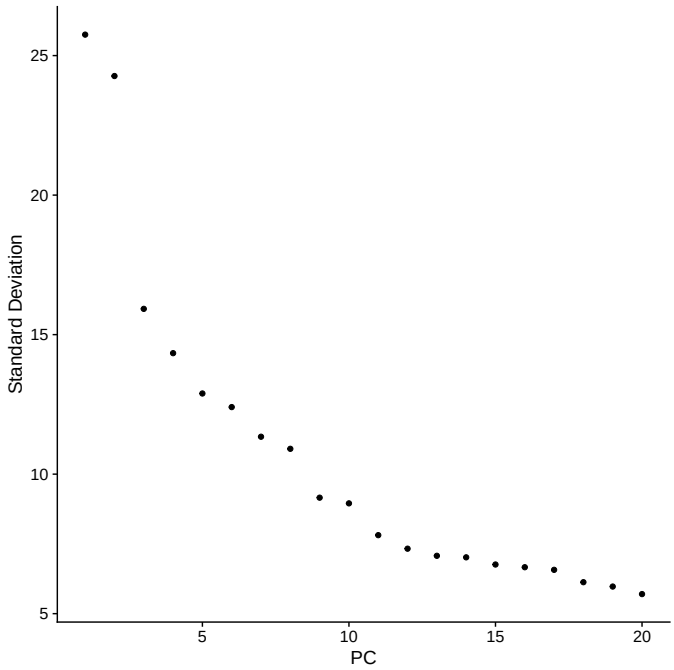


Abbildung 6: Standardabweichung der Top 20 PCs von Probe 1.

nale Regression. Diese eliminiert die Notwendigkeit der log-Transformation und weicht damit vom Standardworkflow ab (Hafemeister und Satija 2019).

Integration und Dimensionsreduktion Nach der Normalisierung werden die verschiedenen Proben zusammengefügt und integriert. Dies geschieht, indem der Workflow Gene identifiziert, welche in mehreren Datensätzen HVGs darstellen. Dann werden die 3000 höchst gestuften Gene selektiert und basierend auf ihnen im gesamten Datensatz Barcodepaare gesucht, welche sich stark ähneln. Diese Paare dienen in der folgenden Integration als sogenannte Anker, da sie die zelluläre Beziehung zweier Proben darstellen. Die Integration korrigiert den Batcheffekt und speichert ihr Ergebnis ebenfalls neben den bereits vorhandenen Rohdaten und den Ergebnissen von SCT. Anschließend wird auf den neuen integrierten Daten eine PCA durchgeführt. Sollte der Workflow mit nur einer Probe arbeiten, wird lediglich die PCA durchgeführt, da die Integrationsschritte nicht benötigt werden. Als letzter Schritt der Vorverarbeitung wird mittels UMAP die Dimension der integrierten Daten nochmals reduziert und als Graph dargestellt. Wieder können wir über die Konfigurationsdatei festlegen, wie viele der zuvor berechneten PCs in der UMAP-Reduzierung benutzt werden sollen.

3.1.2 Analyse und Ausgabe

Clustering Der Seuratworkflow beginnt die Analyse mit dem Clustern der dimensionreduzierten Daten. Dazu wird zuerst auf den reduzierten Daten ein kNN-Graph konstruiert. Wieder können wir über die Konifgurationsdatei festlegen, wie viele PCs verwendet werden. Als Nächstes wird aus dem kNN-Graph ein gemeinsame nächste Nachbarn Graph (englisch: shared

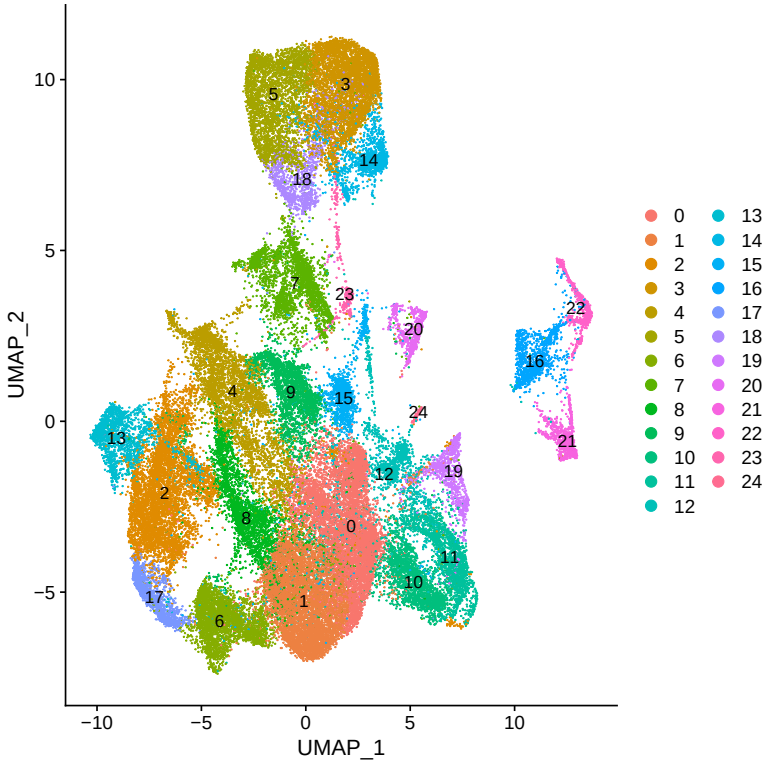


Abbildung 7: Das von Seurat berechnete Clustering der integrierten Daten im UMAP-Format.

nearest neighbor graph, SNN-Graph) konstruiert, indem eine Überlappung zwischen einem Barcode und seiner 20 nächsten Nachbarn berechnet wird. Anschließend werden aus dem SNN-Graph mit dem Louvain Algorithmus (Blondel et al. 2008) Cluster extrahiert (Abbildung 7). Über die Konfigurationsdatei können wir den *resolution* Parameter festlegen. Dieser bestimmt, wie groß bzw. klein die Cluster sein dürfen. Werte über 1.0 erzeugen größere und damit weniger Cluster, Werte unter 1.0 haben das Gegenteil zur Folge. Letztlich erzeugt der Workflow eine Grafik in UMAP-Format, in der die verschiedenen gefundenen Cluster dargestellt werden. Sollten mehrere Zustände von Proben vorhanden sein, wird eine zusätzliche UMAP-Grafik erstellt, die die Barcodes in ihre jeweiligen Zustände unterteilt.

Wir können in der Konfigurationsdatei die Resolution weglassen und stattdessen eine Menge von verschiedenen Werten angeben. Der Workflow erzeugt dann Grafiken für jeden unserer angegebenen Werte. Daraufhin können wir uns für den Wert entscheiden, der eine adäquate Menge von Clustern generiert.

Auffinden von Biomarkern Der nächste Schritt in der Analyse ist das Auffinden von Biomarkern bzw. Markergenen. Markergene sind Gene, die ein anderes Expressionsmuster als die anderen Gene aufweisen. Vor dem Auffinden müssen die Daten jedoch erneut normalisiert werden. Hier wird jedoch nicht die SCTransform Funktion verwendet, sondern die Standardherangehensweise von Normalisierung, Logarithmieren und Skalierung umgesetzt. Dies wird über die Seuratfunktionen *NormalizeData* und *ScaleData* realisiert. NormalizeData teilt die

Werte der Gen-Barcode-Matrix durch die Summe der Werte pro Barcode und multipliziert diese dann mit einem Skalierungsfaktor; in unserem Workflow beträgt dieser 10.000. Anschließend werden die Daten log-transformiert. ScaleData zentriert die Daten erst, indem für jedes Gen die durchschnittliche Expression dieses Gens abgezogen wird. Anschließend wird durch das Dividieren der zentrierten Werte durch deren Standardabweichung skaliert.

Um letztlich die Biomarker zu finden, wird jeder Cluster gegen den Rest der Barcodes mittels Wilcoxon-Rangsummentest verglichen. Das Ergebnis ist eine Matrix mit allen relevanten Werten für die Analyse pro Gen: (1) p-Werte, (2) der Anteil von Barcodes, welche dieses Gen im Cluster exprimieren, (3) der Anteil von Barcodes, welche dieses Gen exprimieren und außerhalb des Clusters liegen, sowie (4) die logFoldChange Werte (Tabelle 2). Diese Matrix wird als .csv Datei abgespeichert. Zusätzlich generiert der Workflow eine .csv Datei, welche die durchschnittliche Expression pro Gen über alle Barcodes hinweg enthält.

	p_val 1	avg_log2FC	pct.1	pct.2	cluster
Gen 1	0	1.32109E+14	0.726	0.257	0
Gen 2	0	1.10134E+13	0.998	0.956	0
⋮	⋮	⋮	⋮	⋮	⋮
Gen M	0.227	3.12459E+14	0.337	0.259	2

Tabelle 2: Beispielhafter Auszug der .csv Datei. pct.1 und pct.2 stehen jeweils für die anteilige Expression innerhalb/außerhalb des Clusters.

Differenzielle Genexpressionsanalyse Falls die Daten mehrere Probenzustände beinhalten, werden in der nachfolgenden Regel ihre Unterschiede in der Genexpression festgestellt. Für jeden Cluster werden die Barcodes innerhalb des Clusters in ihre Zustände unterteilt und diese mittels Wilcoxontest verglichen. Es entsteht eine .csv Datei pro Cluster sowie eine .csv Datei pro Zustand, jeweils wieder mit den bereits erwähnten Werten.

Zellzählung Zusätzlich erstellt der Workflow .csvs mit der Anzahl der Barcodes pro Cluster. Einmal pro Probe und, falls möglich, einmal pro Probenzustand.

ShinyCell ShinyCell (Ouyang et al. 2021) ist ein R Softwarepaket zum Erstellen von interaktiven Webanwendungen zur Darstellung von Ez-RNA-Seq.-Analysedaten. ShinyCell kann Genexpressionsdaten in reduzierten Dimensionen (zum Beispiel UMAP) darstellen oder Zellinformationen visualisieren. Der Workflow erstellt automatisch eine ShinyApp, basierend auf dem Seuratobjekt, welches nach der Biomarkererkennung generiert wird.

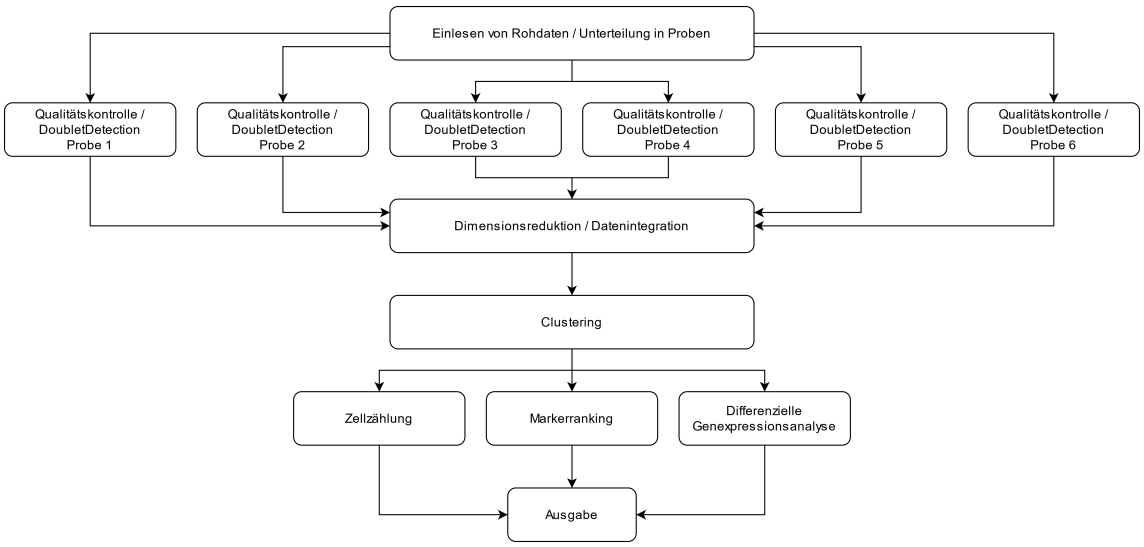


Abbildung 8: Visuelle Darstellung der verschiedenen Schritte des Scanpyworkflows. Der originale Snakemake DAG sowie eine Auflistung aller Regeln kann in Abbildung 17 im Anhang gefunden werden.

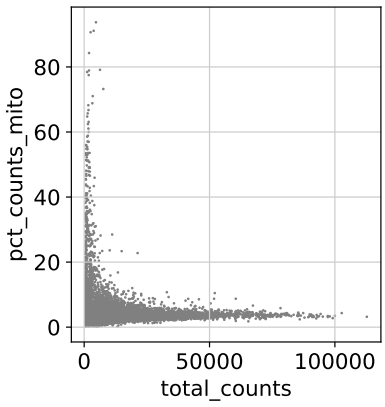
3.2 Scanpy Workflow

Der hier vorgestellte Workflow baut auf dem Analysesoftwarepaket Scanpy (Wolf, Angerer und Theis 2018) auf und ist damit in der Programmiersprache Python angesiedelt. Scanpy ist wie Seurat ein voll umfassendes Analyseframework für Ez-RNA-Seq.-Daten und stellt damit eine vergleichbar weitreichende Funktionalität zum Bearbeiten dieser zur Verfügung. Der Workflow basiert ebenfalls auf der Standardprozedur aus Abschnitt 2.4 und umfasst damit die gleichen generellen Schritte: Qualitätskontrolle, Datenintegration, Dimensionsreduktion, Clusteranalyse und differenzielle Genexpressionsanalyse. Der Workflow kann ebenfalls mehrere Proben verarbeiten und für die Analyse zusammenfügen sowie verschiedene Probenstadien untereinander vergleichen.

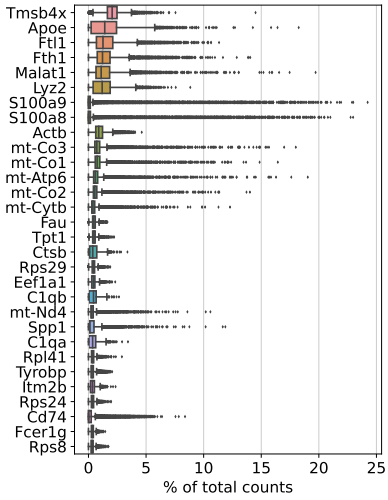
Abbildung 8 stellt den Ausführungsgraphen des Workflows dar. Dieser ist im Bereich der Qualitätskontrolle über die verschiedenen Proben sowie am Ende bei der Auswertung nach der Clusteranalyse parallelisiert. Dazu werden die Rohdaten in der ersten Regel in die unterschiedlichen Proben unterteilt und für jede eine eigene Datei angelegt, welche nachfolgend durch Snakemake parallel bearbeitet werden.

Im Gegensatz zum Seuratworkflow werden probenspezifische Parameter in einer tabellarischen .tsv Datei von uns festgelegt und dann von Snakemake an das jeweilige Skript weitergegeben. Alle workflowübergreifende Parameter sind weiterhin in einer Konfigurationsdatei angesiedelt.

Als Zwischenergebnis zwischen den einzelnen Snakemakeregeln benutzt der Workflow das .h5ad Dateiformat, welches eine Abwandlung des HDF5 (Koziol, Robinson und Science 2018) Formats ist. HDF5 und damit auch .h5ad wurden für schnelle IO-Operationen und das Organisieren von großen Datenmengen konzipiert.



(a) Streudiagramm, welches den prozentualen Anteil von mitochondrialer RNA einer Zelle gegenüber ihres Transkriptoms abbildet. Prozentualer Anteil der RNA auf der y-Achse, totale Expression auf der x-Achse.



(b) Box-Plot für die Top 30 Gene einer Probe. Die Grafik zeigt den Median, das obere und untere Quartil sowie Ausreißer pro Gen an.

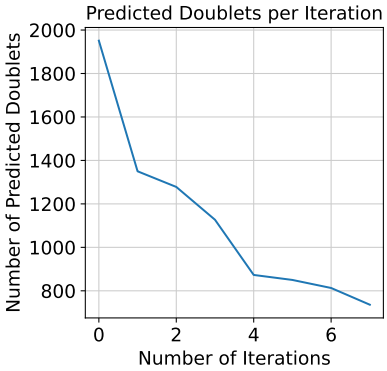
Abbildung 9: Diagramme, die von dem Workflow bei der Filterung von Genen erzeugt werden. Streudiagramm für die Anzahl verschiedener Gene analog zu (a)

3.2.1 Vorverarbeitung

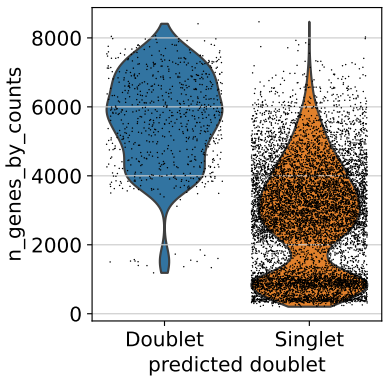
Qualitätskontrolle Der Workflow beginnt ebenfalls mit dem Einlesen von Rohdaten. Dafür wird die vollständige Gen-Barcode-Matrix aufgenommen und in ein Anndataobjekt konvertiert. Erst anschließend wird das Objekt grundlegend nach Zellen und Genen, die nicht den Anforderungen entsprechen, gefiltert. Im nächsten Schritt wird das Anndataobjekt in die einzelnen Proben zerlegt und mit dem jeweiligen Probennamen und Probenzustand annotiert. Die zugehörigen Werte werden aus der .tsv Datei ausgelesen. Als Ergebnis entsteht eine .h5ad Datei pro Probe, welche dann anschließend von Snakemake bis zur Datenintegration parallel verarbeitet werden.

Als nächster Schritt werden die Standardmetriken berechnet und in verschiedenen Diagrammen dargestellt. Dabei werden über die zum Seuratworkflow analogen Violinendiagramme (Abbildung 4) hinaus noch Streudiagramme für den prozentualen Anteil mitochondrialer RNA sowie die Anzahl verschiedener Gene pro Barcode erstellt (Abbildung 9 (a)). Dazu kommt ein Box-Plot für die Top 30 Gene, sortiert nach dem prozentualen Anteil ihrer Expression am gesamten Transkriptom (Abbildung 9 (b)). Dabei werden die insgesamt sechs Diagramme jeweils vor und nach dem Filtern erstellt, um deren Auswirkung visuell darzustellen.

Darauf folgt die Filterung von Doublets mit dem Softwarepaket DoubletDetection (Gayoso und Shor 2020). DoubletDetection arbeitet auf den Rohexpressionsdaten ohne vorherige Normalisierung oder Logarithmierung. DoubletDetection erzeugt wie DoubletFinder auch zunächst künstliche Doublets durch das Addieren zweier zufälliger Barcodes, jedoch ohne diese zu mitteln. Die künstlichen Doublets werden darauf mit den echten Barcodes zusammengefügt und der Datensatz mittels Louvain Algorithmus gruppiert. Das Clustering wird dabei auf einer durch PCA dimensionsreduzierten Darstellung der Daten berechnet. Die Anzahl der verwende-



(a) Anzahl der von DoubletDetection erfassten Doublets pro Iteration.



(b) Durch die Natur der Enstehung von Doublets haben diese durch ihr hybrides Transkriptom eine signifikant höhere Anzahl an verschiedenen Genen.

Abbildung 10: Diagramme, die vom Workflow bei der Filterung von Doublets erzeugt werden.

ten PCs wird von uns über die .tsv Tabelle pro Probe festgelegt. Dazu wird vor der Filterung mit DoubletDetection ein Ellenbogendiagramm mit der Varianz der ersten 30 PCs erstellt (analog zu Abbildung 6).

Anschließend wird auf jedem der Cluster ein hypergeometrischer Test durchgeführt und der clusterweite p-Wert berechnet.

$$p = 1 - \text{ hypergeom.cdf}(N, K, n, k)$$

Hierbei steht N für die Anzahl an Barcodes in der Probe, K für die Anzahl aller künstlichen Doublets, n ist die Anzahl von Barcodes im Cluster und k repräsentiert die Anzahl der künstlichen Doublets im Cluster. hypergeom.cdf ist die hypergeometrische kumulative Verteilungsfunktion. Dieser Prozess wird, inklusive der Erzeugung künstlicher Doublets, von DoubletDetection mehrfach wiederholt, die Anzahl der Iterationen kann vom Anwender bestimmt werden. Der Doubletscore eines Barcodes ergibt sich aus dem Durchschnitt der p-Werte aller Durchläufe (Xi und Li 2021). Xi und Li vermerken, dass sich mit mehr Iterationen zwar die Laufzeit signifikant verlängert, jedoch ohne wirkliche Verbesserung der Performanz. Der Workflow durchläuft daher acht Wiederholungen. Nach der Detektion von Doublets mit DoubletDetection werden wie bei dem Seuratworkflow ebenfalls die erkannten Doublets in einer UMAP-Grafik dargestellt (analog Abbildung 7). Dazu kommt ein Diagramm, welches die Anzahl der erkannten Doublets pro Iteration von DoubletDetection darstellt (Abbildung 10 (a)), sowie ein Violinendiagramm, welches den Unterschied zwischen Doublets und Singlets in ihrer Anzahl der verschiedenen Gene verdeutlicht (Abbildung 10 (b)).

Datenintegration und Dimensionsreduktion Nach dem Filtern von Doublets werden die probenspezifischen Anndataobjekte wieder zusammengefügt und der daraus resultierende Gesamtdatensatz normalisiert und logarithmiert. Anschließend werden HVGs selektiert, zusätzlich wird eine Version der Gen-Barcode-Matrix mit allen Genen gespeichert. Scanpy erlaubt

bei der Selektion das Unterteilen in die verschiedenen Proben. Dadurch erhalten wir zusätzliche Informationen, zum Beispiel, in wie vielen verschiedenen Proben dasselbe Gen variabel ist. Der Workflow selektiert daraufhin alle Gene, die in mindestens drei verschiedenen Proben HVGs darstellen; das sind in unserem Datensatz 2637 Gene.

Für die anstehende Integration der Proben bietet der Workflow zwei Optionen: Scanorama (Hie, Bryson und Berger 2019) und Harmony (Korsunsky et al. 2019). Welche der beiden Methoden verwendet wird, legen wir über die Konfiguration fest. Im Folgenden werden wir beide Verfahren kurz erläutern.

Scanorama erlaubt die Integration von Proben, ohne dass diese gemeinsame Zelltypen oder ein strukturell ähnliches Genexpressionsmuster besitzen müssen. Scanorama erkennt Barcodes, die ein ähnliches Expressionsmuster zwischen Proben besitzen und kann diese Ähnlichkeiten ausnutzen, um Datensätze zu integrieren. Dabei werden Proben, welche keine Überlappungen enthalten, exkludiert. Scanorama verwendet nearst neighbor search (NNS), um geteilte Zelltypen zwischen allen Proben zu finden. Die Reihenfolge, in der die Proben integriert werden, basiert auf dem Prozentsatz der übereinstimmenden Barcodes zwischen den Proben. Verknüpfte Barcodes bilden Gemeinsamkeiten, die zur Korrektur des Batcheffektes und Integration der Proben verwendet werden. Um die Suche mit NNS zu optimieren, wird zuerst mittels Singulärwertzerlegung (englisch: singular value decomposition, SVD) der hochdimensionale Genraum komprimiert.

Scanorama ermöglicht neben der Integration von Proben die Korrektur des Batcheffekts auf Gen-Barcode-Matrix Ebene. Dadurch steigt zwar die benötigte Rechenzeit enorm, erlaubt aber das Durchführen von differenzieller Genexpressionsanalyse auf korrigierten Daten, da diese nicht auf niedrigdimensionalen Darstellungen durchgeführt werden kann (Hie, Bryson und Berger 2019; Tran et al. 2020).

Harmony verwendet iteratives Clustering, um Zellen aus verschiedenen Proben zu integrieren. Zuerst werden die Proben kombiniert und mit PCA eine Dimensionsreduktion durchgeführt. Daraufhin folgt der iterative Prozess, welcher aus vier Schritten besteht. Zunächst werden die Zellen mittels Soft-K-Means-Clustering gruppiert, wobei Cluster Barcodes aus mehreren Proben enthalten können. Als nächstes berechnet Harmony Zentroide für die Cluster als auch für jede Probe. Im dritten Schritt werden aus den Schwerpunkten Korrigierungsfaktoren für jede Probe berechnet. Im letzten Schritt wird jede Zelle mit ihrem zellspezifischen Faktor korrigiert. Der Vorgang wird bis zur Konvergenz oder einer maximalen Anzahl von Iterationen wiederholt (Korsunsky et al. 2019; Tran et al. 2020). Der Workflow verwendet die Python Portierung Harmonypy (Slowikowski 2020) des ursprünglich in R geschriebenen Verfahrens. Im Gegensatz zu Scanorama erzeugt Harmony nur eine niedrigdimensionale Repräsentation der integrierten Daten und kann nicht die originale Gen-Barcode-Matrix korrigieren. Ebenfalls wird diese Repräsentation nur auf der Basis der zuvor selektierten HVGs berechnet.

Tran et al. zeigen, dass unterschiedliche Integrationsmethoden auf verschiedenste Daten-

sätze Vor- und Nachteile haben. Der Workflow bietet daher zwei verschiedene Integrationsmethoden an, falls das Resultat der einen nicht zufriedenstellend ist. Weiterhin wird Harmony als eine hervorragende Methode zum initialen Analysieren großer Datensätze vorgeschlagen, dank der extrem guten Skalierbarkeit und Laufzeit (Abschnitt 4.2). In dieser Arbeit verwenden wir für die Analyse des Datensatzes ebenfalls Harmony.

Sollte der Datensatz aus nur einer Probe bestehen, so wird keine Integration vorgenommen, sondern lediglich eine Dimensionsreduktion mittels PCA durchgeführt.

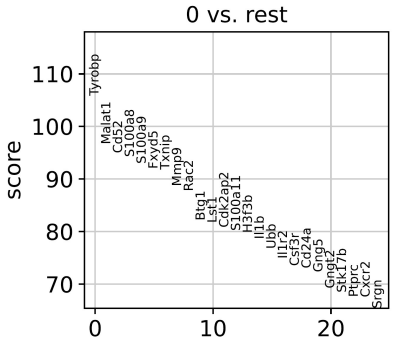
Unabhängig von der Integrations- bzw. Reduktionsmethode wird anschließend auf den dimensionsreduzierten Daten ein Nachbarschaftsgraph erstellt. Dieser dient als Basis für die nachfolgenden Schritte der Analyse. Die Effizienz der Nachbarschaftssuche hängt dabei stark von UMAP ab, da UMAP auch eine Methode zur Schätzung der Konnektivität von Datenpunkten bietet, die hier verwendet wird.

3.2.2 Analyse und Ausgabe

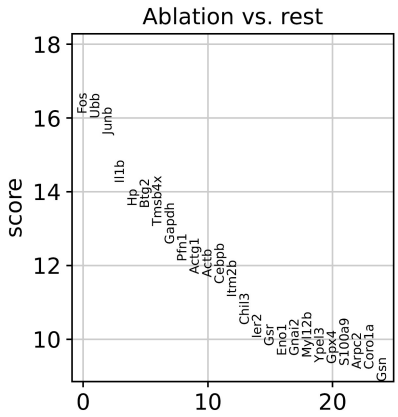
Clustering Für das Clustering verwendet der Workflow den Leiden Algorithmus (Traag, Waltman und Eck 2019). Der Leiden Algorithmus ist eine laufzeitoptimierte Version des Louvain Algorithmus mit gleichzeitiger Verbesserung der Gruppierungen. Die Verbesserung der Gruppierungen ist notwendig, da es vorkommen kann, dass der Louvain Algorithmus intern getrennte Grupierungen erzeugt, was die Ergebnisse verfälscht. Eine intern getrennte Gruppe ist eine Gruppe, welche mindestens einen Knotenpunkt (Barcode) besitzt, welcher nicht direkt mit dem Rest seiner Gruppe verbunden ist. (Traag, Waltman und Eck 2019, Abbildung 2). Der Leiden Algorithmus verbessert den Louvain Algorithmus indem er vollständig verbundenen Gruppen garantiert.

Als Grundlage für das Clustern dient der vorher berechnete Nachbarschaftsgraph. Anschließend werden die Cluster in UMAP-Grafik dargestellt (analog Abbildung 7).

Biomarker Ranking Nach dem Clustering werden wieder biologische Marker gesucht, die jeden Cluster definieren. Dafür wird die während der Datenintegration zusätzlich gespeicherte vollständige Gen-Barcode-Matrix verwendet. Diese wird vor dem Ranking noch normalisiert und logarithmiert. Anschließend wird das Expressionsmuster jedes Clusters mit dem Zusammenschluss aller anderen Barcodes mittels Wilcoxontest verglichen. Es entsteht ein Ellenbogendiagramm für jeden Cluster, der seine obersten 25 Gene darstellt (Abbildung 11 (a)), sowie ein Dotplot, welches die Top 5 Gene jedes Clusters im Vergleich zu allen anderen Clustern anzeigt (Abbildung 29, Anhang). Die Ergebnisse werden pro Cluster in eine .csv Datei abgespeichert, die alle Gene des gesamten Datensatzes umfasst. Zusätzlich wird eine .csv Datei für die durchschnittliche Expression jedes Gens pro Cluster erstellt.



(a) Top 25 Gene von Cluster 0 im Vergleich mit allen anderen Barcodes.



(b) Top 25 Gene des Zustandes Ablation[1] im Vergleich mit allen anderen Barcodes innerhalb von Cluster 0.

Abbildung 11: Ellenbogendiagramme zum Vergleich der Top-Gene.

Differenzielle Genexpressionsanalyse Die differenzielle Genepressionsanalyse folgt ebenfalls auf das Clustering, wird jedoch nur durchgeführt, falls der Datensatz mehrere Probenzustände umfasst. Es wird für jeden Cluster ein Wilcoxontest durchgeführt, welcher die Barcodes der verschiedenen Probenzustände vergleicht. Die Ergebnisse werden ebenfalls in einem Ellenbogendiagramm dargestellt (Abbildung 11 (b)) und anschließend eine .csv Datei pro Probenzustand mit den Testergebnissen erstellt.

Zellzählung Die Zellzählung erfolgt nach dem Clustering. Es wird für jede Probe eine .csv Datei angelegt, welche angibt, wie viele Zellen dieser Probe in welchem Cluster sind. Zusätzlich wird ebenfalls eine .csv Datei pro Probenzustand erstellt mit der Verteilung auf die Cluster.

Ausgabe Insgesamt erstellt der Workflow für unseren Datensatz 35 .h5ad und 78 .csv Dateien. Dazu kommen 99 verschiedene Diagramme und Statistiken. Der Workflow produziert keine interaktive Web-Applikation.

[1] *Ablation bezeichnet in diesem Kontext das Entfernen von biologischem Gewebe, zum Beispiel durch einen chirurgischen Eingriff.*

4 Vergleich

Im folgenden Abschnitt werden wir die beiden Workflows miteinander vergleichen. Wir gehen dabei auf ihre Laufzeit sowie ihre analytischen Ergebnisse ein und heben die Hauptunterschiede hervor. Dafür wurden beide Workflows auf denselben Datensatz angewendet. Dabei wurden, falls möglich, die gleichen Parameter verwendet; zum Beispiel wurden bei beiden Workflows Barcodes, die weniger als 200 Gene exprimieren, sowie Gene die in weniger als 3 Barcodes vorkommen, herausgefiltert.

4.1 Datensatz

Der in dieser Arbeit hauptsächlich verwendete Datensatz umfasst ca. 67000 Barcodes mit 32000 Genen, aufgeteilt auf sechs Proben in zwei verschiedenen Zuständen. Die Proben entstammen dem braunen Fettgewebe von Mäusen und wird in der Arbeit mit Maus67k referenziert. Nach der grundlegenden Filterung umfasst der Datensatz noch 66000 Barcodes mit 21000 Genen. Eine Aufteilung der Barcodes in Proben und Zustände kann aus Tabelle 3 entnommen werden. Alle in dieser Arbeit verwendeten Grafiken basieren auf diesem Datensatz.

Um die Skalierbarkeit der beiden Workflows messen zu können, wurden für die folgende Laufzeitanalyse zwei weitere Datensätze hinzugezogen. Einer der beiden entstammt ebenfalls dem braunen Fettgewebe von Mäusen und umfasst 13000 Barcodes mit 32000 Genen, nach der ersten Filterung sind es noch 13000 Barcodes und 18000 Gene (Tabelle 4). Er wird mit Maus13k betitelt.

Der dritte Datensatz ist öffentlich zugänglich und kann über die 10x Genomics Webseite bezogen werden (Genomics 2020). Er umfasst ca. 11000 Barcodes mit 37000 Genen vor, und 11000 Barcodes sowie 23000 Gene nach der Filterung. Es handelt sich um einen Einzelprobendatensatz bestehend aus mononukleären Zellen des peripheren Blutes (englisch: peripheral blood mononuclear cell, PBMC) vom Menschen und wird im nachfolgenden Teil als Pbmc_10k bezeichnet.

	Barcodes	Zustand
Probe 1	10266	Ablation
Probe 2	14804	Sham
Probe 3	13818	Ablation
Probe 4	10190	Ablation
Probe 5	6589	Sham
Probe 6	10373	Sham

Tabelle 3: Barcodeanzahl des Maus67k Datensatzes, aufgeteilt nach Proben.

	Barcodes	Zustand
Probe 1	2862	Tag 3
Probe 2	1236	Tag 3
Probe 3	2096	Tag 3
Probe 4	2340	Tag 7
Probe 5	2432	Tag 7
Probe 6	2262	Tag 7

Tabelle 4: Barcodeanzahl des Maus13k Datensatzes, aufgeteilt nach Proben.

4.2 Laufzeitanalyse

Die folgenden Laufzeiten in Tabelle 5 wurden mit der eingebauten Benchmarkfunktion von Snakemake ermittelt. Snakemake wiederholt daraufhin jede Regel dreimal und liefert uns detaillierte Informationen über jede Ausführung, unter anderem die Dauer der Ausführung. Die so festgestellten Zeiten wurden gemittelt. Für den Scanpyworkflow wurden Regeln, die über Proben parallelisiert sind, erst innerhalb der Probe über die Anzahl der Durchläufe des Benchmarks gemittelt und anschließend erneut durch die Probenanzahl geteilt. Jede Snakemake Ausführung wurde mit 10 Cores[2] durchgeführt und auf demselben Computer vollzogen. Der verwendete Computer besitzt einen AMD EPYC 7742 64-Kerne Prozessor mit 128 Threads, sowie 1 TiB DDR4 RAM. Der Scanpyworkflow verwendet zum Integrieren der Proben Harmony. Die Laufzeiten kategorisieren nicht die Analysesoftware an sich, sondern den übergreifenden Workflow, in dem sie benutzt werden.

Datensatz	Proben	Barcodes	Gene	Größe in MB	Laufzeit (HH:MM:SS)		Faktor
					Scanpy	Seurat	
Maus13k	6	13000	18000	93	00:02:51	00:58:50	20,6
Pbmc_10k	1	11000	23000	104	00:03:53	00:38:58	10,0
Maus67k	6	66000	21000	691	00:14:02	05:32:31	23,7

Tabelle 5: Laufzeiten der beiden Workflows zu jedem der drei Datensätze sowie den Faktor, um den der Scanpyworkflow schneller ist als der Seuratworkflow.

Laufzeit Wir sehen, dass der Scanpyworkflow in allen drei Datensätzen wesentlich schneller terminiert. Dies hat mehrere Gründe. Zum einen erlaubt das Separieren der verschiedenen Proben im Scanpyworkflows die Parallelisierung der meisten Vorverarbeitungsschritte durch Snakemake. Wie wir an Tabelle 6 und 7, (sowie Abbildung 24 und 22, Anhang) in denen die Laufzeiten zum Maus67k Datensatz aufgeschlüsselt werden, erkennen können, verbringen beide Workflows einen erheblichen Anteil der Zeit mit dem Filtern von Doublets. Da dieser Schritt ebenfalls zur Vorverarbeitung gehört, kann der Scanpyworkflow dabei signifikant Zeit einsparen.

Nach der Doubletfilterung stellt die Datenintegration einen weiteren rechenintensiven Schritt dar. Die Regeln sind prozentual zwar vergleichbar mit 12 bzw. 13 Prozent der Gesamtlaufzeit des jeweiligen Workflows, die Realwerte bilden jedoch mit 00:01:38 bzw. 00:41:57 einen starken Kontrast. Wie bereits erwähnt, wird der hier verwendete Algorithmus Harmony in "A benchmark of batch-effect correction methods for single-cell RNA sequencing data" von Tran et al. aufgrund seiner geringen Laufzeit für das initiale Analysieren von großen Datensätzen

[2] *Dies limitiert nur die gleichzeitige Ausführung von Jobs durch Snakemake, aber nicht die Möglichkeit des Multiprocessings der unterliegenden Skripte.*

empfohlen. Dabei verbraucht Harmony ebenfalls wesentlich weniger RAM als Seurats Integrationsmethode.

Darüber hinaus sind einige Funktionen in Scanpy inhärent multithreaded, Teile des Scanpyworkflows können alle 128 Threads des Computers benutzen. Der Seuratworkflow verwendet dagegen durchgehend nur einen Kern und braucht dementsprechend länger.

Regel	Laufzeit in (HH:MM:SS)	%-Anteil an Gesamtlaufzeit
read_raw_data	00:05:47	41%
split_into_samples	00:00:04	0%
generate_qc_metrices	00:00:07	1%
mito_filter	00:00:06	1%
plot_pca_variance	00:00:04	0%
doublet_detection	00:01:26	10%
data_integration	00:01:38	12%
clustering	00:01:43	12%
marker_ranking	00:01:55	14%
marker_ranking_condition	00:01:07	8%
cell_counting	00:00:04	1%

Tabelle 6: Aufteilung der Laufzeit des Scanpyworkflows nach Regeln.

Regel	Laufzeit in (HH:MM:SS)	%-Anteil an Gesamtlaufzeit
metaData	00:05:32	2%
mt_p1	00:04:08	1%
mt_p2	00:04:11	1%
doubletRemovalElbowPlot	00:17:15	5%
doubletRemoval	01:25:37	26%
addTPsMerge	00:08:30	3%
SCTranformNormalization	00:16:05	5%
IntegrationDimReduction	00:41:57	13%
RunUMAP	00:10:43	3%
useChosenClusterResolution	00:10:18	3%
markerDiscovery	01:44:17	31%
cellCounting	00:17:08	5%
DGE	00:04:12	1%
createShinyApp	00:02:39	1%

Tabelle 7: Aufteilung der Laufzeit des Seuratworkflows nach Regeln.

Input / Output Darüber hinaus ist für uns der durch die Benutzung von Snakemake erzeugte Overhead von Interesse. Die Zwischenergebnisse müssen zuerst auf die Festplatte geschrieben und anschließend wieder eingelesen werden. An Tabelle 8 (sowie Abbildung 23 und 25, Anhang) können wir den durch IO-Operationen verursachten Overhead bei der Verarbeitung des Maus67k Datensatzes bestimmen. Seurat liest die Rohdaten zwar schneller ein als Scanpy, kann die Daten jedoch zwischen den Regeln nicht effizient im .rds Format abspeichern oder einlesen. Scanpy hingegen verbringt einen signifikanten Anteil der Zeit mit dem Einlesen von Rohdaten, kann durch die Verwendung des .h5ad Formats die Daten jedoch später effizient wiederverwenden. Insgesamt verbringt der Scanpyworkflow ca. 56% der Zeit mit dem Einlesen von Daten, der Großteil davon Rohdaten. Seurat hingegen hat einen IO-Overhead von ca. 10%.

	Scanpy	Seurat
Rohdaten Einlesen und Abspeichern	00:05:44	00:03:17
Zwischenergebnis Einlesen und Abspeichern	00:00:02	00:01:56

Tabelle 8: Gegenüberstellung der Dauer der unterschiedlichen IO-Operationen beider Workflows.

Der Anteil von IO-Operationen wurde mit zwei Skripten in Snakemake bestimmt. Das erste verwendet das jeweilige Analysetool, um die Rohdaten einzulesen und anschließend in das dazugehörige Dateiformat abzuspeichern. Die zweite Regel liest daraufhin die neue Datei ein und speichert sie umgehend wieder ab. Die Laufzeit der ersten Regel wurde für das Einlesen von Rohdaten verwendet, das Ergebnis der zweiten für alle anderen Regeln. Da Dateigrößen zwischen den Regeln schwanken können, ist die Messung, insbesonders für Regeln nach dem Einlesen von Rohdaten, nicht perfekt, gibt uns aber eine gute Annäherung an den durch Lese-/Schreiboperationen verursachten Overhead.

Skalierbarkeit Xi und Li haben gezeigt, dass Seurats DoubletFinder eine bessere Skalierbarkeit auf große Datensätze als DoubletDetection besitzt (Xi und Li 2021, Figure 4 (e), Figure 5), der Scanpyworkflow kann dies aber durch das parallele Ausführen bei mehreren Proben ausgleichen. Der Effekt wird durch den Vergleich der Laufzeiten der Datensätze Maus13k und Pbmc_10k deutlich. Obwohl der Pbmc_10k Datensatz sich lediglich ca. 12% in der Dateigröße und ca. 8% in der Anzahl der Zellen in der Gen-Barcode-Matrix nach oben absetzt, braucht der Scanpyworklfow ungefähr 36 Prozent länger zur Verarbeitung (Tabelle 9). Der Seuratworkflow ist bei der Verarbeitung des Pbmc_10k Datensatzes, da nur eine Probe, sogar schneller und braucht ca. 33% weniger Zeit. Dabei ist die Veränderung des Anteils der Doubletfilterung an der Laufzeit der beiden Workflows invers, im Multiproben Datensatz Maus13k liegt er für Scanpy bei ca. 12% und für Seurat bei 44%, im Einzelprobendatensatz Pbmc_10K steigt bzw. sinkt der Anteil auf 35% für Scanpy und 39% für Seurat (Abbildung 18, 19, 20 und 21, Anhang).

	Dateigröße in MB	Zellenanzahl der Gen-Barcode-Matrix	Laufzeit in (HH:MM:SS)	
			Scanpy	Seurat
Maus13k	93	234.000.000	00:02:51	00:58:50
Pbmc_10k	104	253.000.000	00:03:53	00:38:58
Wachstum in Prozent	11,8%	8,1%	36%	-33%

Tabelle 9: Auswirkung von Einzelprobendatensätze auf die Laufzeit der Workflows.

Wenn wir den Maus13k Datensatz als Grundgröße ansehen, können wir die Skalierung der beiden Workflows auf andere Multiprobendatensätze einschätzen. Der Maus67k Datensatz ist 6,4 mal so groß in seiner Dateigröße und 4,9 mal so groß in seiner Zellenanzahl der Gen-Barcode-Matrix. Der Scanpyworkflow braucht für den Maus67k Datensatz ca. 3,9 mal so viel Zeit, der Seuratworkflow liegt bei einer Vervielfältigung von ca. 4,6 (Tabelle 10). Die Skalierbarkeit der beiden Workflows liegt also im selben Größenbereich, der Unterschied in der Baseline ist jedoch signifikant.

	Dateigröße in MB	Zellenanzahl der Gen-Barcode-Matrix	Laufzeit in (HH:MM:SS)	
			Scanpy	Seurat
Maus13k	93	234.000.000	00:02:51	00:58:50
Maus67k	691	1.386.000.000	00:14:02	05:32:31
Wachstum in Prozent	643%	492%	392%	465%

Tabelle 10: Skalierung beider Workflows auf Multiprobendatensätze.

4.3 Vergleich der analytischen Ergebnisse

In diesem Abschnitt werden wir auf die analytischen Ergebnisse, im Spezifischen die Unterschiede in Filterung, HVG Selektion sowie Clustering eingehen und diese hervorheben. Wir verwenden dafür den Maus67k Datensatz, beide Workflows wurden unabhängig voneinander auf den selben Rohdaten ausgeführt.

Vorverarbeitung / Filterung Aus Tabelle 11 können wir entnehmen, wie viele Barcodes der jeweilige Workflow vor und nach einer Filterung noch beinhaltet. Da wir beide Workflows mit denselben Parametern ausgeführt haben, sind die Werte, wie erwartet, während der grundlegenden und der mitochondrialen Filterung identisch. Bei der Doubletfilterung hingegen filtert Scanpy 5225 und Seurat 5681 Barcodes heraus. Eine Visualisierung der Doublets befindet sich in Abbildung 12.

Wir erkennen, dass am oberen Ende der Verteilung die beiden Methoden zum Filtern von Doublets überwiegend übereinstimmen oder die Barcodes nur von DoubletDetection im Scanpyworkflow klassifiziert werden. DoubletDetection klassifiziert auch im Bereich um die 2000

	Ungefiltert	Basis Filterung	Mitochondriale Filterung	Doublet Filterung
Scanpy	67.338	66.040	64.711	59.486
Seurat	67.338	66.040	64.711	59.030

Tabelle 11: Aufschlüsselung der verbliebenen Barcodes beider Workflows.

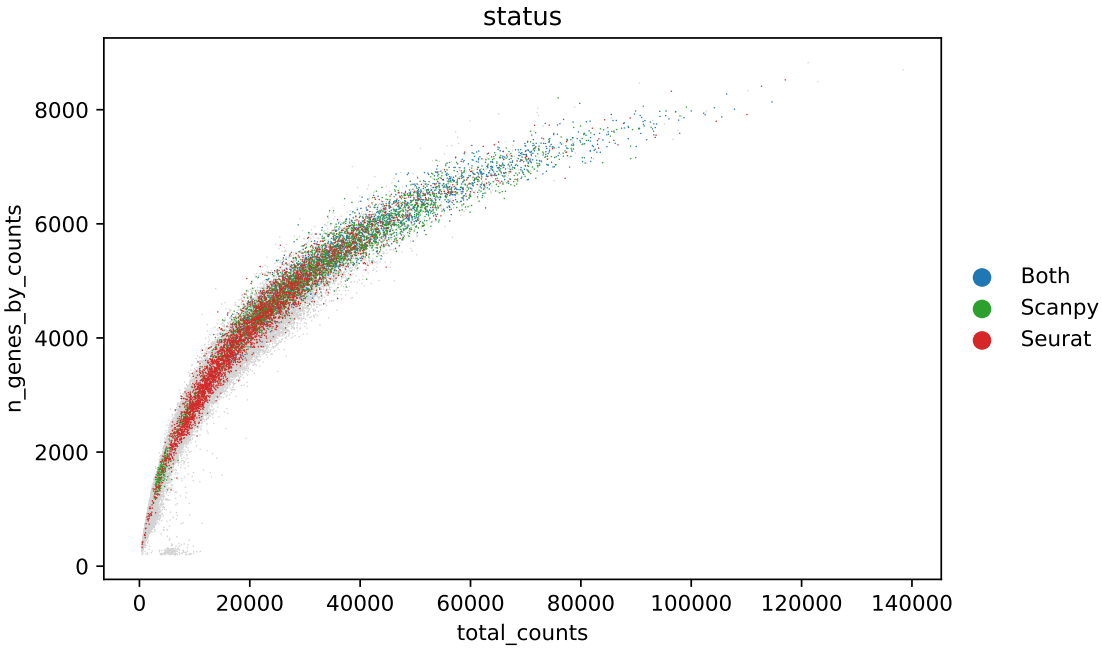


Abbildung 12: Streudiagramm der Barcodes, klassifiziert nach Doubletstatus. Blau wurde von beiden Workflows als Doublet erkannt, Rot nur von Seurat und Grün nur von Scanpy. Anzahl der verschiedenen Gene auf der y-Achse, totale Exprimierung auf der x-Achse.

verschiedenen Gene einige Barcodes als Doublets, bleibt dann in seiner Klassifizierungsrate linear bis ca. 4000 verschiedenen Genen. Im Bereich von 4000-6000 Genen liegen die meisten der Barcodes, die DoubletDetection als Doublets identifiziert. DoubletDetection klassifiziert auch viel bis in die obere Spitze hinein, startet im unteren Bereich jedoch erst ab ca. 1000 Genen (Abbildung 26, Anhang).

Seurats DoubletFinder hingegen fängt bereits ganz unten in der Verteilung an, klassifiziert hier jedoch wenige Barcodes als Doublets. Um ca. 2000 Genen steigt die Erkennungsrate von DoubletFinder stark an, bis sie bei ca. 5000 wieder abfällt (Abbildung 27, Anhang).

Allgemein kann man erkennen, dass sich die Klassifizierungsprofile ähneln, jedoch leicht verschoben sind. Dies liegt an der Methode zum Erstellen von künstlichen Doublets der jeweiligen Software. Daraus resultiert, dass die meisten Doublets, die von beiden Workflows erkannt werden, im Bereich von ca. 5000 bis 6000 Genen liegen (Abbildung 28, Anhang).

Letztendlich lässt sich mit beiden Methoden keine absolut präzise Klassifizierung durchführen. Beide Methoden versuchen den Effekt, den Doublets auf die downstream Analyse haben, zu minimieren. Wir erwarten deshalb einen Unterschied in den selektierten HVGs sowie im Clustering und damit auch in der differenziellen Expressionsanalyse. Insgesamt werden 1833 Barcodes von beiden, 3392 Barcodes nur von Scanpy und 3848 Barcodes nur von Seurat als Doublets klassifiziert.

HVG Selektierung Der Seuratworkflow detektiert die HVGs mit der SCTransform Funktion. Standardgemäß werden dabei die obersten 3000 Gene genommen, geordnet nach ihrem Score über alle Proben hinweg. Der Scanpyworkflow hingegen selektiert alle Gene, die in mindestens drei verschiedenen Proben gleichzeitig als HVG eingestuft werden. Dies sind auf unserem Datensatz 2637 Gene.

Der Scanpyworkflow selektiert 927 Gene, welche von Seurat nicht als HVG angesehen werden, umgekehrt beträgt die Differenz 1290 Gene. Wenn wir im Scanpyworkflow ebenfalls die Top 3000 Gene selektieren, vermindert sich der Unterschied auf 790 in beide Richtungen. Die Änderung der Selektion von HVGs im Scanpyworkflow wirkt sich jedoch nur marginal auf das folgende Clustering aus.

Die Methodik der Selektion kann jedoch eine große Auswirkung mit sich bringen, welche von Datensatz zu Datensatz schwankt. Der Seuratworkflow selektiert immer die Top 3000 Gene. Die Anzahl der HVGs, die der Scanpyworkflow selektiert, ist jedoch dynamisch. Da der Scanpyworkflow potenziell beliebig viele Proben verarbeiten kann, ist ein vorgeschriebenes Limit kontraproduktiv. Durch das Selektieren von zu vielen HVGs werden die anschließenden Berechnungschritte aufwendiger und die Laufzeit steigt, ohne dass sich eine merkliche Verbesserung im Clustering zeigt.

Integration und Clustering Das Clustering folgt auf die Integration der Daten. Wir erkennen an Abbildung 13 und 14, dass sich nach der Integration in beiden Workflows die gleiche globale Struktur bildet, lediglich um 90 Grad gedreht. Jedoch ist auch klar zu erkennen, dass

der Seuratworkflow die lokalen Strukturen wesentlich detailreicher darstellen kann. Dichte Barcodepopulationen werden im Scanpyworkflow nicht gut genug aufgelöst, um wirkliche Strukturen erkennen zu können.

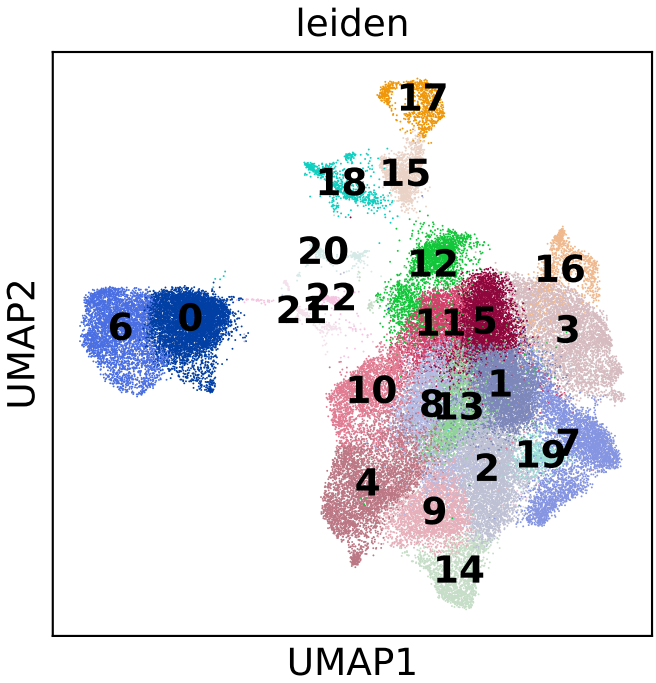


Abbildung 13: Das von Scanpy mit dem Leiden Algorithmus berechnete Clustering.

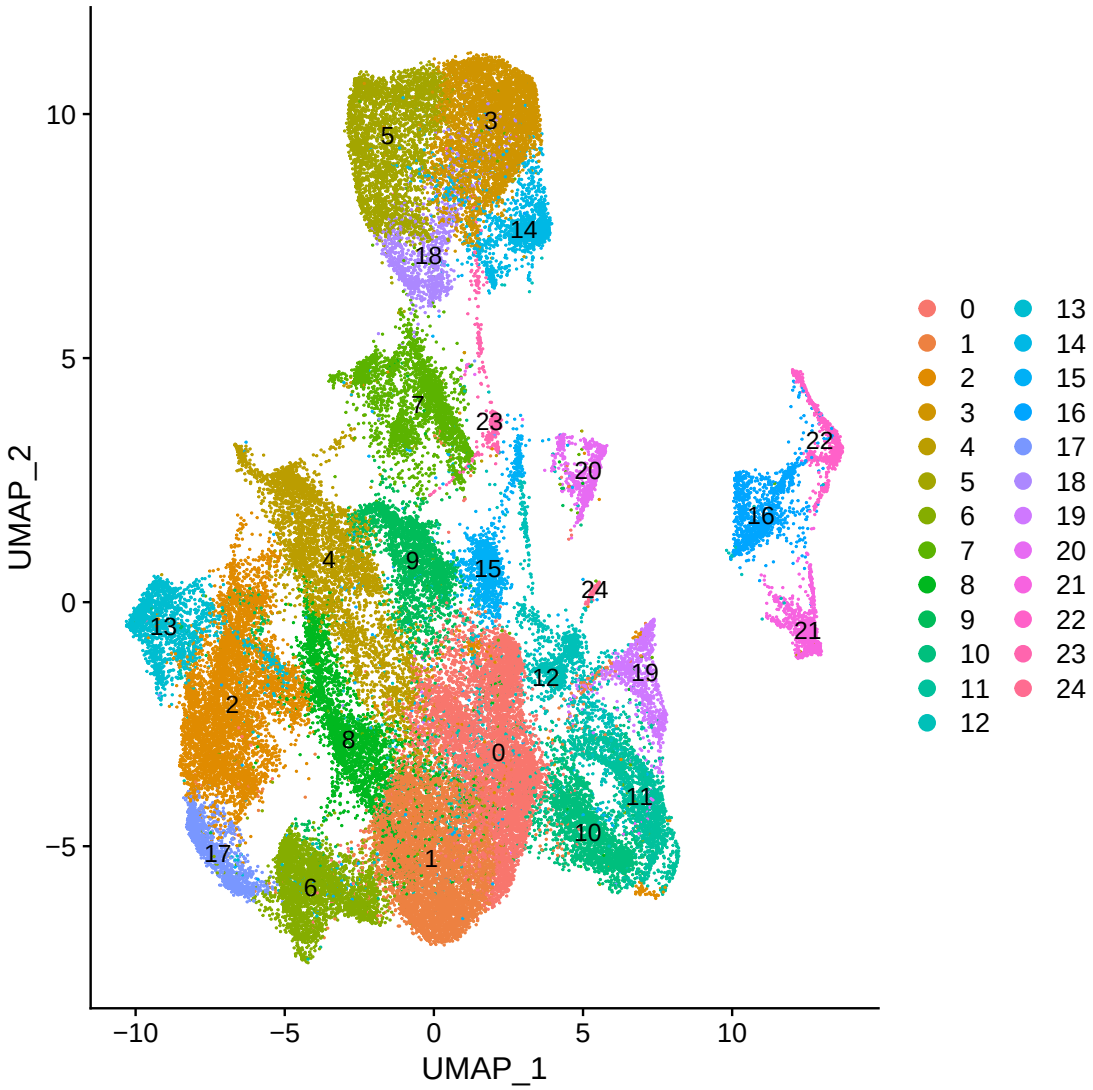
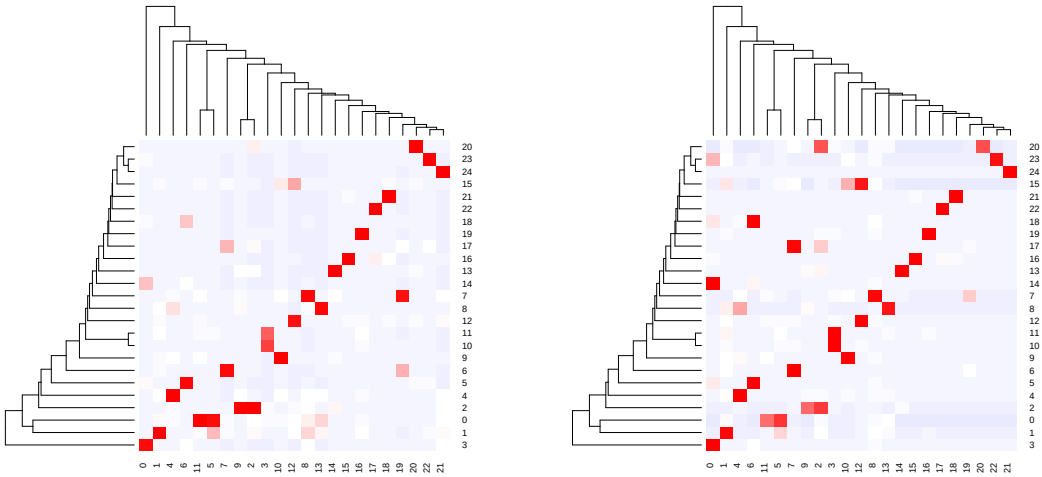


Abbildung 14: Das von Seurat mit dem Louvain Algorithmus berechnete Clustering.

Das Hauptziel des Clusterings ist es, Barcodes mit ähnlicher Expression in die gleiche Gruppe zu unterteilen. Daher liegt unser Hauptinteresse des Vergleiches darin, festzustellen, ob beide Workflows dieselben Barcodes miteinander gruppiert. Wie den Grafiken zu entnehmen ist, ordnet der Scanpyworkflow die Barcodes in 23 Cluster, mit Bezeichnung 0-22, der Seuratworkflow gruppiert in 25 Cluster mit Bezeichnung 0-24. Der Unterschied der Clusteranzahl kann an einer Reihe von Faktoren liegen. Seurat hat mehr Barcodes in der Doubletfilterung entfernt, es wurde eine signifikante Anzahl unterschiedlicher HVGs selektiert, die Integrationsmethoden unterscheiden sich ebenfalls nicht nur gering und letztlich wurde auch mit unterschiedlichen Clusteralgorithmen gruppiert. Weiterhin möchten wir hier betonen, dass, während die meisten Parameter beider Workflows gleich gehalten wurden, der Resolution Parameter sich unterscheidet. Der Seuratworkflow gruppiert mit einer Resolution von 0.5, während der Scanpyworkflow eine Resolution von 1.0 verwendet.

Um die Cluster der beiden Workflows gegenseitig zuweisen zu können, wurden die Bar-



(a) Heatmap skaliert nach Spalten um zu verdeutlichen, welche Barcodes der Scanpycluster in welche Seuratcluster eingeteilt werden.

(b) Heatmap skaliert nach Reihen um zu verdeutlichen, welche Barcodes der Seuratcluster in welche Scanpycluster eingeteilt werden.

Abbildung 15: Heatmaps basierend auf der Matchingmatrix. Scanpycluster auf der x-Achse, Seuratcluster auf der y-Achse. Je mehr Überschneidung der Barcodes zwischen zwei Clustern desto rötlicher die korrespondierende Zelle in der Heatmap

codes jedes Clusters ausgelesen und mit allen Clustern des anderen Workflows verglichen. Die Anzahl an Übereinstimmungen ergibt eine Matchingmatrix, auf deren Grundlage wir die Heatmaps in Abbildung 15 (a) und (b) erstellen können. Die Heatmaps erlauben uns eine definitive Zuweisung der Cluster zwischen den beiden Workflows. Wir erkennen, dass Seurat die Scanpycluster 6 und 0 noch weiter in Cluster 5,3,18 und 14 unterteilt. Ebenfalls unterteilt Seurat den Scanpycluster 3 nochmals in Cluster 10 und 11 und Scanpycluster 12 in 12 und 15. Dafür wird der Seuratcluster 2 von Scanpy in 2 und 9 geteilt und der Cluster 0 in 5 und 11.

Für die letztendliche Bewertung der Ähnlichkeit der beiden Clusterings verwenden wir eine Reihe von Metriken.

Randindex Der Randindex (Rand 1971) RI ist ein Wert im Intervall von $[0,1]$ berechnet über die Formel

$$RI = \frac{a+b}{\binom{n}{2}},$$

wobei a die Anzahl der Paare ist, die im gleichen Cluster über zwei verschiedenen Gruppierungen liegt, b ist die Anzahl der Paare, die sich in unterschiedlichen Clustern befinden. $\binom{n}{2}$ ist die Anzahl der ungeordneten Paare einer Menge mit n Elementen. Damit beschreibt der Randindex die Wahrscheinlichkeit, dass zwei Gruppierungen bei einem zufällig gewählten Paar übereinstimmen.

Unsere Clusterings erhalten einen RI von 0,94.

Adjusted Rand Index Der Randindex berücksichtigt jedoch nicht die Möglichkeit der zufälligen Verteilung. Indem die erwartete Ähnlichkeit aller paarweisen Vergleiche zwischen den durch ein Zufallsmodell geschaffenen Gruppierungen in die Berechnung mit eingenommen wird, erhalten wir eine Korrektur für die Zufälligkeit des Clusterings:

$$ARI = \frac{RI - Expected\_RI}{max(RI) - Expected\_RI}$$

Damit liegt der ARI im Interval von $[-1, 1]$, wobei Werte unter 0 schlechter sind als ein zufälliges Clustering.

Wir erhalten einen ARI von 0,51.

Adjusted Mutual Information Nach Romano et al. ist der ARI besonders für Clusterings geeignet, welche gleich verteilt sind. Da dies bei uns nicht der Fall ist, wird von Romano et al. die Adjusted Mutual Information (Vinh, Epps und Bailey 2009) empfohlen, welche für ungleich verteilte Clusterings bessere Ergebnisse liefert. Der AMI ist die für Zufälligkeit korrigierte Version der Mutual Information (MI):

$$AMI(U,V) = \frac{MI(U,V) - E\{MI(U,V)\}}{max\{H(U), H(V)\} - E\{MI(U,V)\}}$$

U, V sind Partitionen/Gruppierungen, $H(U)$ ist die zu der Partition U zugehörige Entropie. Der AMI liegt im Intervall $[0, 1]$, wobei 0 ein komplett zufälliges Gruppieren kennzeichnet.

Unsere Clusterings haben einen AMI von 0,67.

Insgesamt können wir feststellen, das die Clusterings zwar nicht vollständig übereinstimmen, aber trotzdem eine hohe Ähnlichkeit besitzen.

Biomarker Die in den beiden Workflows gefundenen Biomarker werden aufgrund von Unterschieden in der Filterung sowie den verschiedenen Clusterings voneinander abweichen.

Wenn wir die Biomarker zweier Cluster vergleichen, die gegenseitig eindeutig zuweisbar sind, wie zum Beispiel Seuratcluster 24 mit seinem Scanpy Gegenstück 21, dann liegt der Unterschied der Top 50 Gene beider Markerlisten bei nur 4 Genen. Diese 4 Gene sind wiederum im unmittelbaren Bereich außerhalb der Top 50 der jeweiligen Markerliste.

Um Cluster zu vergleichen, welche nicht übereinstimmen, kombinieren wir zwei Cluster und entnehmen jeweils die Top 25. Der Seuratcluster 20 wird von Scanpy relativ gleich auf Cluster 20 und 2 verteilt. Wir nehmen die Top 25 Gene aus Cluster 2 und 20 von Scanpy und vergleichen diese mit der Top 50 des Seuratclusters. Hier unterschieden sich die Listen fast vollständig mit 47 unterschiedlichen Genen. Obwohl der Seuratcluster gleich verteilt wird, so besteht der Scanpycluster 2 überwiegend nicht aus Barcodes dieses Clusters. Die Barcodes aus Seuratcluster 20 werden also in einer geringeren Anzahl vermischt und das Expressionsmuster wird verzerrt. Die Barcodes in Scanpycluster 20 werden zwar nicht mit einem anderen

Cluster vermischt, besitzen aber anscheinend ein völlig anderes Expressionsmuster und damit Markerlisten.

Dies ist jedoch nicht immer der Fall. Scanpycluster 3 wird von Seurat in Cluster 10 und 11 unterteilt, die Markerlisten unterscheiden sich jedoch lediglich in 9 Genen. Diese 9 Gene sind wieder im direkten Bereich unterhalb der Top 25 von Seuratcluster 10 bzw. 11.

Abschließend halten wir fest, dass die Markerlisten bei übereinstimmenden Clustern nahezu identisch sind. Bei Clustern, die eindeutig von einen Workflow zerlegt werden, sind auch deren Listen identisch, wenn man sie kombiniert. Nur wenn die Clusterings nicht übereinstimmen und ein Workflow Teile des Clusters mit anderen Clustern vermischt, unterscheiden sich die gefundenen Biomarker wesentlich.

5 Fazit

Wenn wir die Struktur und die Ausführungsgraphen beider Workflows betrachten, erkennen wir, dass sich die beiden Workflows in ihrer Methodik stark ähneln bzw. nahezu identisch sind. Alle vollzogenen Schritte sind analog und unterscheiden sich lediglich in ihrer angewendeten Software, gegeben durch die unterschiedlichen zugrundeliegenden Programmiersprachen. Die beiden Workflows können dieselben Funktionen erfüllen und unterscheiden sich darüber hinaus nur in ihrer Laufzeit. Diese ist auch der größte Unterschied zwischen den beiden. Der Scanpyworkflow macht effiziente Benutzung von Multithreading und wurde mit dem Kriterium Skalierbarkeit konzipiert. Ein wichtiger Bestandteil dafür ist die Parallelisierung der Vorverarbeitung über mehrere Proben. Der Seuratworkflow kann dafür die Strukturen in UMAP-Grafiken besser auflösen und bietet uns die Möglichkeit einer interaktiven Web-App zur besseren Darstellung und Präsentation der Ergebnisse.

Im Abschnitt 4.3 haben wir festgestellt, dass sich die Unterschiede der Vorverarbeitung nur geringfügig auf das Clustering und damit auf die gesamte Analyse auswirken. Daher empfehlen wir den Scanpyworkflow für initiale Untersuchungen großer Datensätze mit mehreren Proben aufgrund seiner geringen Laufzeit. Sollte eine akkuratere Repräsentation in Grafiken oder als interaktive Web-App, ein großer Einzelprobendatensatz untersucht werden oder eine Berechnung auf einem Clustersystem vonnöten sein, so kann auf den Seuratworkflow zurückgegriffen werden.

5.1 Aussicht

Das Ökosystem von Scanpy umfasst einige Softwaretools, die wir in dieser Arbeit nicht verwendet haben. Zum Beispiel bietet Scanpy einen Wrapper um die Doublet Filterungssoftware Scrublet (Wolock, Lopez und Klein 2019), welche sehr effizient und schnell arbeitet. Darüber hinaus interagiert Scanpy gut mit einigen interaktiven Web-Apps wie z.B. dem USCS Cell Browser (Speir et al. 2021). Der Scanpyworkflow kann mit diesen erweitert werden und bietet damit mehr Optionen für den Endbenutzer.

Darüber hinaus können die hier angewendeten Prinzipien in der Parallelisierung der Vorverarbeitung auf den Seuratworkflow übertragen werden.

Letztlich kann durch die Benutzung von Conda und Snakemake die Barriere zwischen Programmiersprachen gebrochen werden, solange die verwendeten Datenformate konvertierbar sind. Dies würde uns erlauben, für jeden Schritt des Workflows das optimale Tool auszuwählen, unabhängig von dessen Implementierung.

Literatur

[1] Philipp Angerer et al. “Single cells make big data: New challenges and opportunities in transcriptomics”. In: *Current Opinion in Systems Biology* 4 (2017). Big data acquisition and analysis Pharmacology and drug discovery, S. 85–91. ISSN: 2452-3100. DOI: https://doi.org/10.1016/j.coisb.2017.07.004. URL: https://www.sciencedirect.com/science/article/pii/S245231001730077X.

[2] Luke Zappia, Belinda Phipson und Alicia Oshlack. “Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database”. In: *PLOS Computational Biology* 14.6 (Juni 2018), S. 1–14. DOI: https://doi.org/10.1371/journal.pcbi.1006245.

[3] Felix Mölder et al. “Sustainable data analysis with Snakemake [version 2; peer review: 2 approved]”. In: *F1000Research* 10.33 (2021). DOI: https://doi.org/10.12688/f1000research.29032.2.

[4] F. Alexander Wolf, Philipp Angerer und Fabian J. Theis. “SCANPY: large-scale single-cell gene expression data analysis”. In: *Genome Biology* 19.15 (Feb. 2018). DOI: https://doi.org/10.1186/s13059-017-1382-0.

[5] Tim Stuart et al. “Comprehensive Integration of Single-Cell Data”. In: *Cell* 177.7 (2019), 1888–1902.e21. DOI: https://doi.org/10.1016/j.cell.2019.05.031.

[6] Aleksandra A. Kolodziejczyk et al. “The Technology and Biology of Single-Cell RNA Sequencing”. In: *Molecular Cell* 58.4 (2015), S. 610–620. ISSN: 1097-2765. DOI: https://doi.org/10.1016/j.molcel.2015.04.005. URL: https://www.sciencedirect.com/science/article/pii/S1097276515002610.

[7] Grace X. Y. Zheng et al. “Massively parallel digital transcriptional profiling of single cells”. In: *Nature Communications* 8.1 (2017). DOI: https://doi.org/10.1038/ncomms14049.

[8] Malte Luecken und Fabian J. Theis. “Current best practices in singlecell RNAseq analysis: a tutorial”. In: *Molecular Systems Biology* 15.6 (2019). DOI: https://doi.org/10.15252/msb.20188746.

[9] Johnny A. Sena et al. “Unique Molecular Identifiers reveal a novel sequencing artefact with implications for RNA-Seq based gene expression analysis”. In: *Scientific Reports* 8.1 (Sep. 2018), S. 13121. ISSN: 2045-2322. DOI: https://doi.org/10.1038/s41598-018-31064-7.

[10] Bo Sun et al. “Double-jeopardy: scRNA-seq doublet/multiplet detection using multiomic profiling”. In: *Cell Reports Methods* 1.1 (2021), S. 100008. ISSN: 2667-2375. DOI: https://doi.org/10.1016/j.crmeth.2021.100008. URL: https://www.sciencedirect.com/science/article/pii/S2667237521000084.

[11] Aisha A. AlJanahi, Mark Danielsen und Cynthia E. Dunbar. “An Introduction to the Analysis of Single-Cell RNA-Sequencing Data”. In: *Molecular Therapy - Methods & Clinical Development* 10 (Sep. 2018), S. 189–196. ISSN: 2329-0501. DOI: https://doi.org/10.1016/j.omtm.2018.07.003. Elsevier.

[12] Anaconda Software Distribution. *Conda*. Version Version 2-2.4.0. 2016. URL: https://www.anaconda.com.

[13] Christopher S. McGinnisand Lyndsay M. Murrow und Zev J. Gartner. “DoubletFinder: Doublet Detection in Single-Cell RNA Sequencing Data Using Artificial Nearest Neighbors”. In: *Cell Systems* 8.4 (2019), 329–337.e4. ISSN: 2405-4712. DOI: https://doi.org/doi:10.1016/j.cels.2019.03.003.

[14] Christoph Hafemeister und Rahul Satija. “Normalization and variance stabilization of single-cell RNA-seq data using regularized negative binomial regression”. In: *Genome Biology* 20.1 (2019), S. 296. DOI: https://doi.org/10.1186/s13059-019-1874-1.

[15] Nan Miles Xi und Jingyi Jessica Li. “Benchmarking Computational Doublet-Detection Methods for Single-Cell RNA Sequencing Data”. In: *Cell systems* 12.2 (Feb. 2021), 176–194.e6. DOI: https://doi.org/10.1016/j.cels.2020.11.008.

[16] Vincent D Blondel et al. “Fast unfolding of communities in large networks”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (Okt. 2008). DOI: https://doi.org/10.1088/1742-5468/2008/10/p10008.

[17] John F. Ouyang et al. “ShinyCell: simple and sharable visualization of single-cell gene expression data”. In: *Bioinformatics* (März 2021). ISSN: 1367-4803. DOI: https://doi.org/10.1093/bioinformatics/btab209.

[18] Quincey Koziol, Dana Robinson und USDOE Office of Science. *HDF5*. März 2018. DOI: 10.11578/dc.20180330.1. URL: https://www.osti.gov/biblio/1631295.

[19] Adam Gayoso und Jonathan Shor. *JonathanShor/DoubletDetection: doubletdetection v3.0*. Version v3.0. Dez. 2020. DOI: 10.5281/zenodo.4359992. URL: https://doi.org/10.5281/zenodo.4359992.

[20] Brain Hie, Bryan Bryson und Bonnie Berger. “Efficient integration of heterogeneous single-cell transcriptomes using Scanorama”. In: *Nature Biotechnology* 37.6 (Juni 2019), S. 685–691. ISSN: 1546-1696. DOI: https://doi.org/10.1038/s41587-019-0113-3.

[21] Ilya Korsunsky et al. “Fast, sensitive and accurate integration of single-cell data with Harmony”. In: *Nature Methods* 16.12 (Dez. 2019), S. 1289–1296. ISSN: 1548-7105. DOI: https://doi.org/10.1038/s41592-019-0619-0.

[22] Hoa Thi Nhu Tran et al. “A benchmark of batch-effect correction methods for single-cell RNA sequencing data”. In: *Genome Biology* 21.1 (Jan. 2020). ISSN: 1474-760X. DOI: https://doi.org/10.1186/s13059-019-1850-9.

[23] Kamil Slowikowski. *slowkow/harmonypy: harmonypy version 0.0.5*. Version v0.0.5. Aug. 2020. DOI: https://doi.org/10.5281/zenodo.4531401.

[24] V. A. Traag, L. Waltman und N. J. van Eck. "From Louvain to Leiden: guaranteeing well-connected communities". In: *Scientific Reports* 9.1 (März 2019). ISSN: 2045-2322. DOI: https://doi.org/10.1038/s41598-019-41695-z.

[25] 10x Genomics. *10k Peripheral blood mononuclear cells (PBMCs) from a healthy donor, Single Indexed*. Version Single Cell Gene Expression Dataset by Cell Ranger 4.0.0. Juli 2020. URL: https://support.10xgenomics.com/single-cell-gene-expression/datasets/4.0.0/SC3_v3_NextGem_SI_PBMC_10K.

[26] William M. Rand. "Objective Criteria for the Evaluation of Clustering Methods". In: *Journal of the American Statistical Association* 66.336 (1971), S. 846–850. DOI: https://doi.org/10.1080/01621459.1971.10482356. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356.

[27] Simone Romano et al. "Adjusting for Chance Clustering Comparison Measures". In: *Journal of Machine Learning Research* 17.134 (2016), S. 1–32. URL: http://jmlr.org/papers/v17/15-627.html.

[28] Nguyen Xuan Vinh, Julien Epps und James Bailey. "Information Theoretic Measures for Clusterings Comparison: Is a Correction for Chance Necessary?" In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, S. 1073–1080. ISBN: 9781605585161. DOI: https://doi.org/10.1145/1553374.1553511.

[29] Samuel L. Wolock, Romain Lopez und Allon M. Klein. "Scrublet: Computational Identification of Cell Doublets in Single-Cell Transcriptomic Data". In: *Cell Systems* 8.4 (2019), 281–291.e9. ISSN: 2405-4712. DOI: https://doi.org/10.1016/j.cels.2018.11.005. URL: https://www.sciencedirect.com/science/article/pii/S2405471218304745.

[30] Matthew L. Speir et al. "UCSC Cell Browser: visualize your single-cell data". In: *Bioinformatics* (Juli 2021). ISSN: 1367-4803. DOI: https://doi.org/10.1093/bioinformatics/btab503.

A Snakemake DAGs

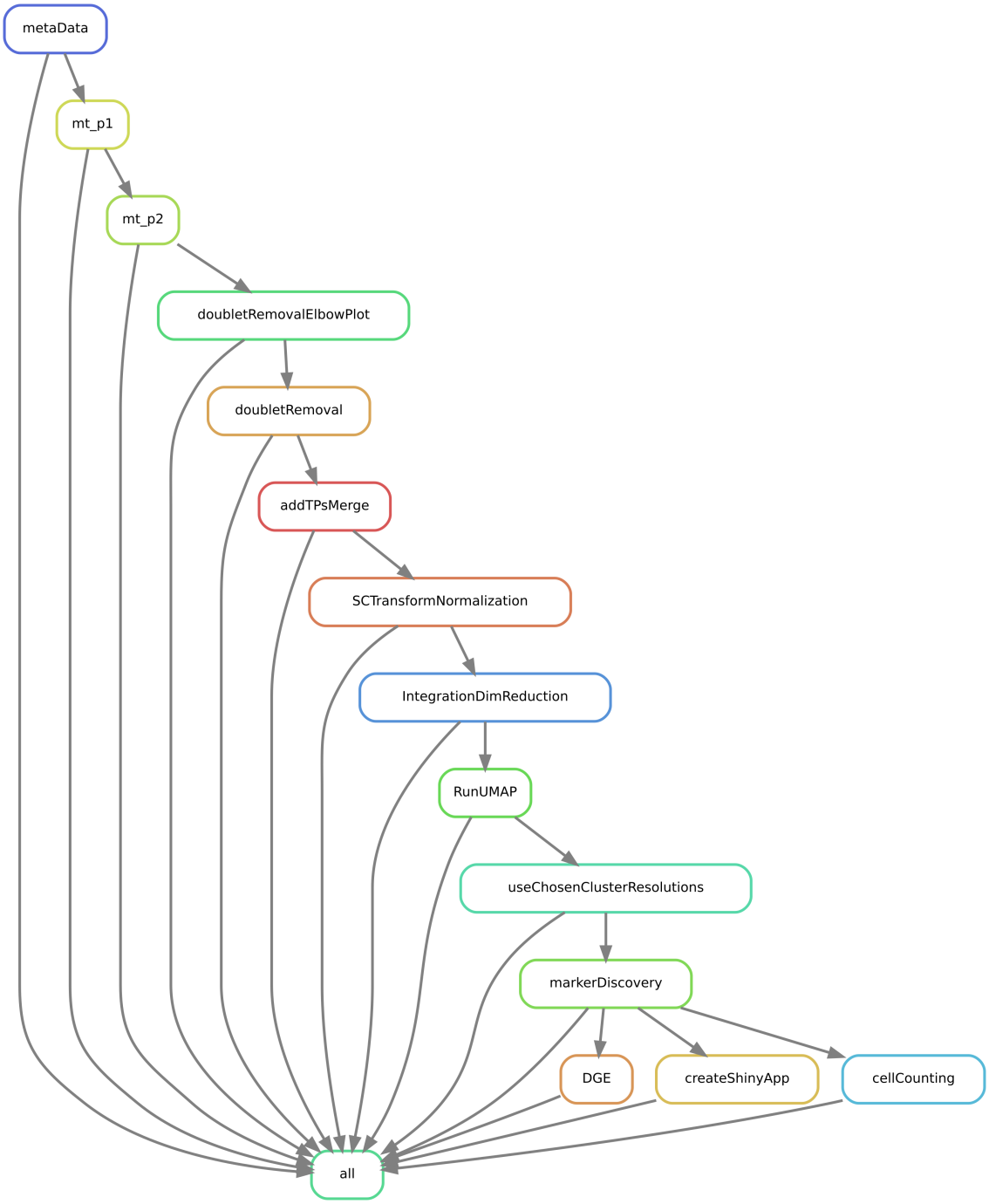


Abbildung 16: Originaler, von Snakemake erstellter DAG des Seuratworkflows.

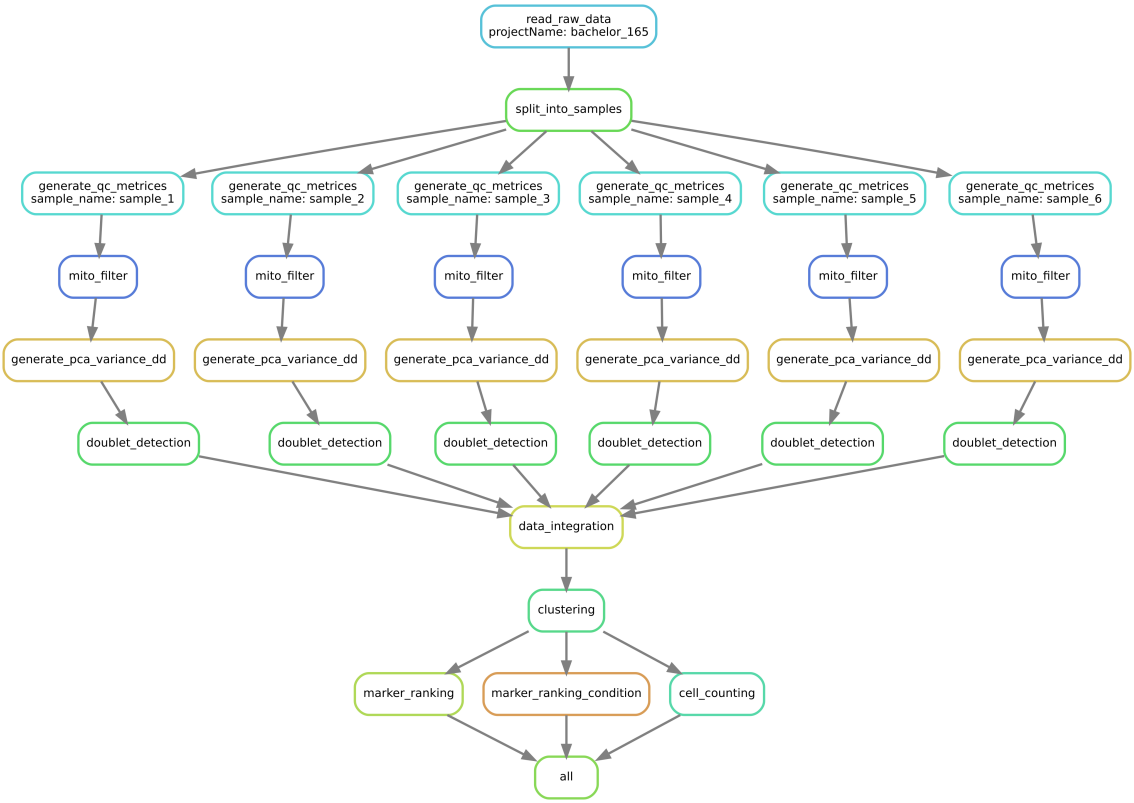


Abbildung 17: Originaler, von Snakemake erstellter DAG des Scanpyworkflows.

B Benchmark Grafiken

B.1 Maus13k

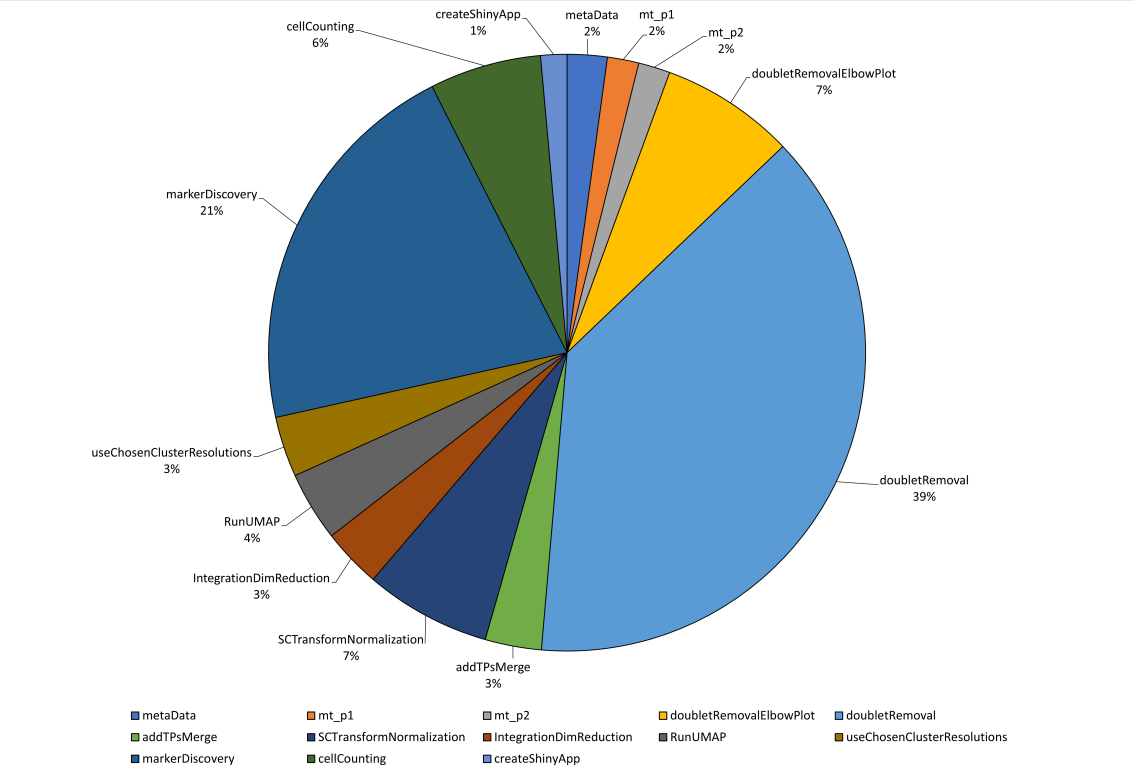


Abbildung 18: Visualisierung der Laufzeit des Seuratworkflow während der Bearbeitung des Maus13k Datensatzes, aufgeteilt nach Regeln.

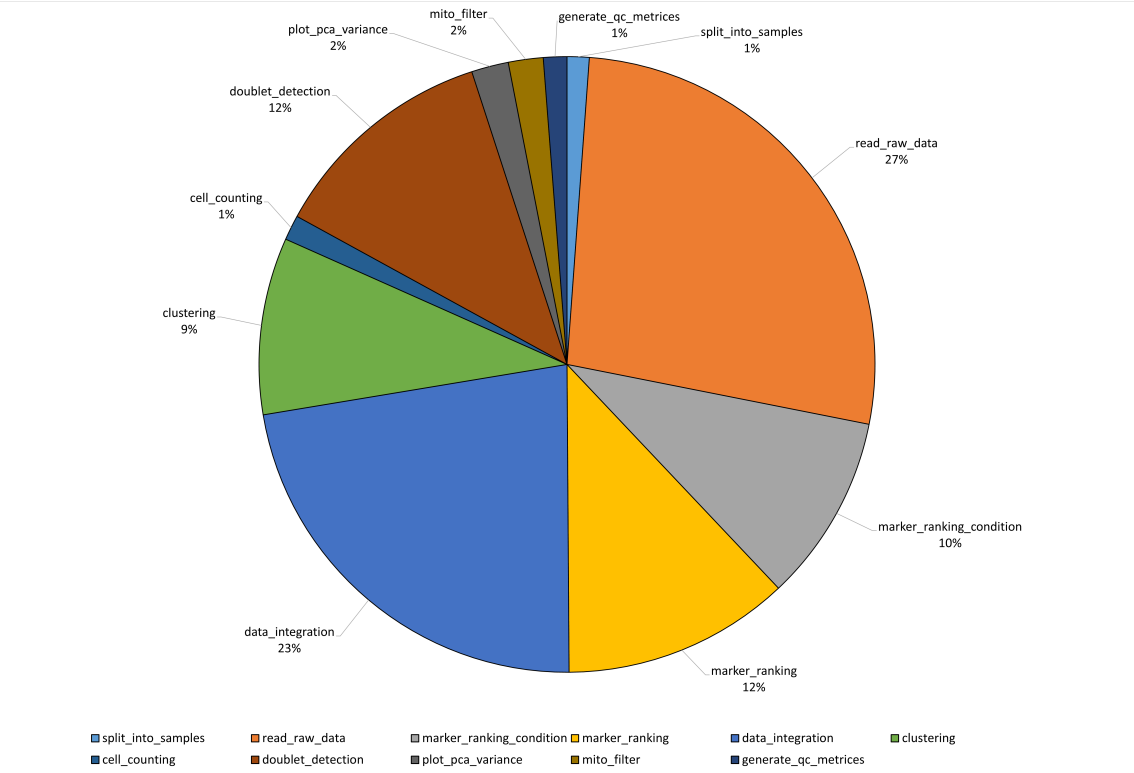


Abbildung 19: Visualisierung der Laufzeit des Scanpyworkflows während der Bearbeitung des Maus13k Datensatzes, aufgeteilt nach Regeln.

B.2 Pbmc_10k

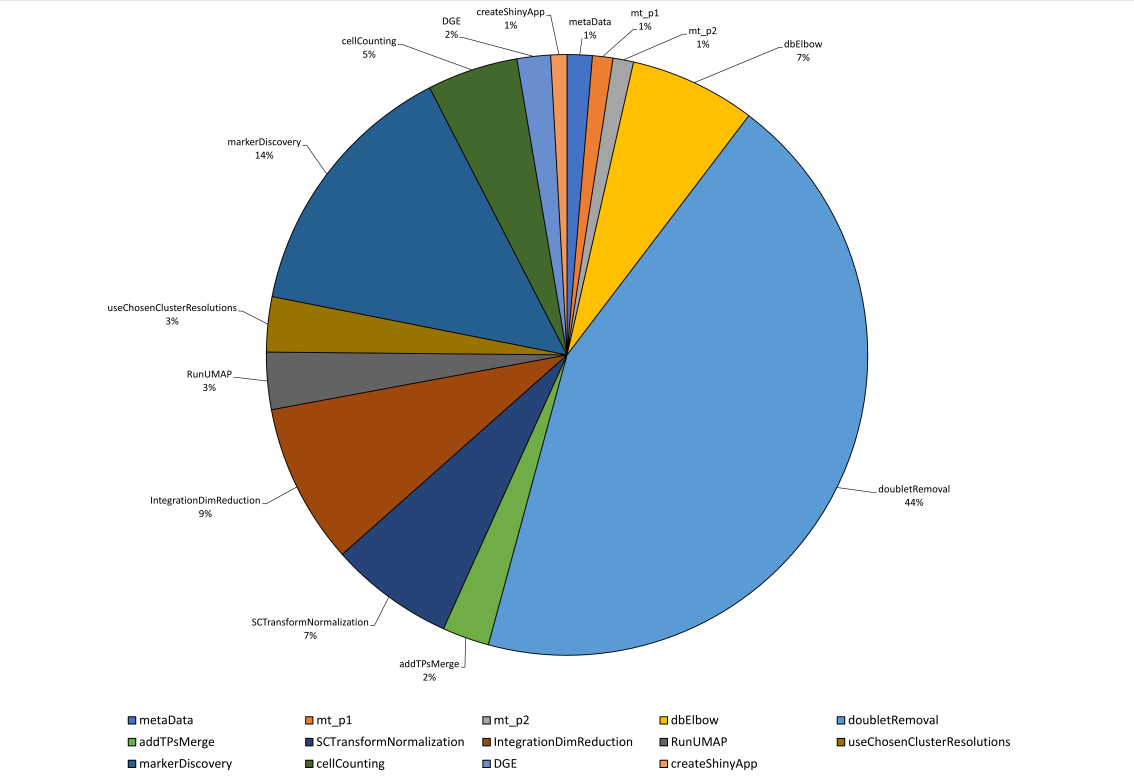


Abbildung 20: Visualisierung der Laufzeit des Seuratworkflow während der Bearbeitung des Pbmc10k Datensatzes, aufgeteilt nach Regeln.

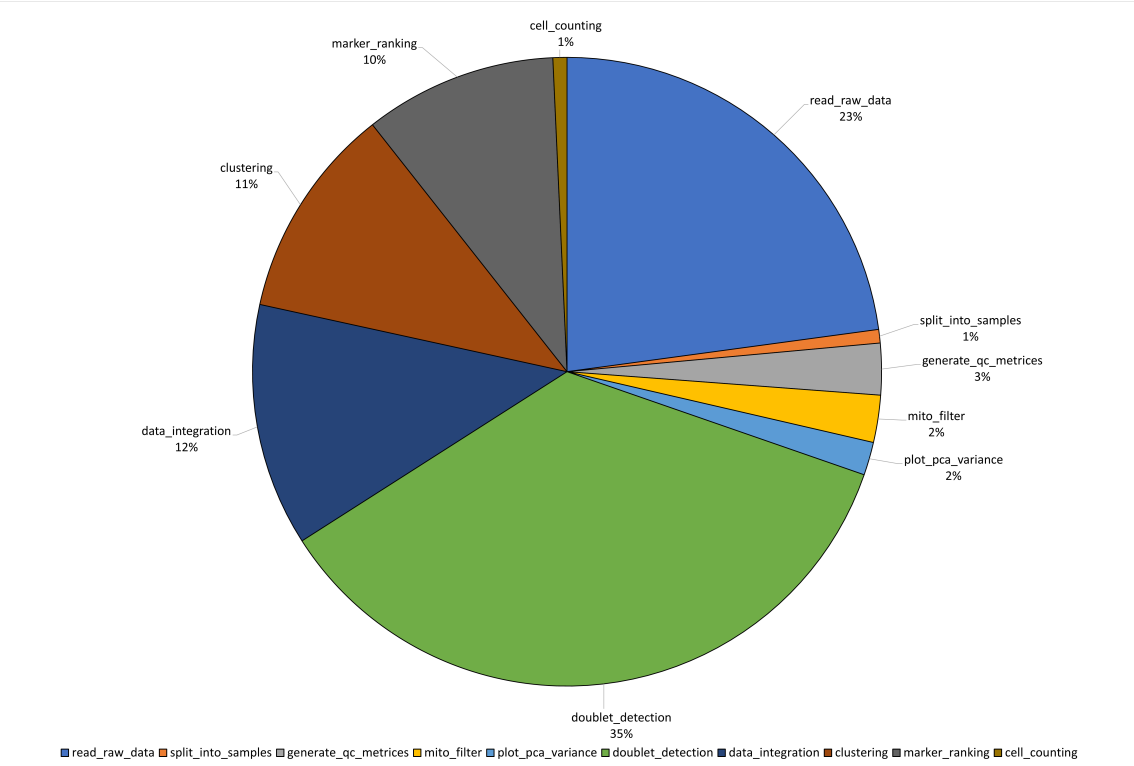


Abbildung 21: Visualisierung der Laufzeit des Scanpyworkflows während der Bearbeitung des Pbmc10k Datensatzes, aufgeteilt nach Regeln.

B.3 Maus67k

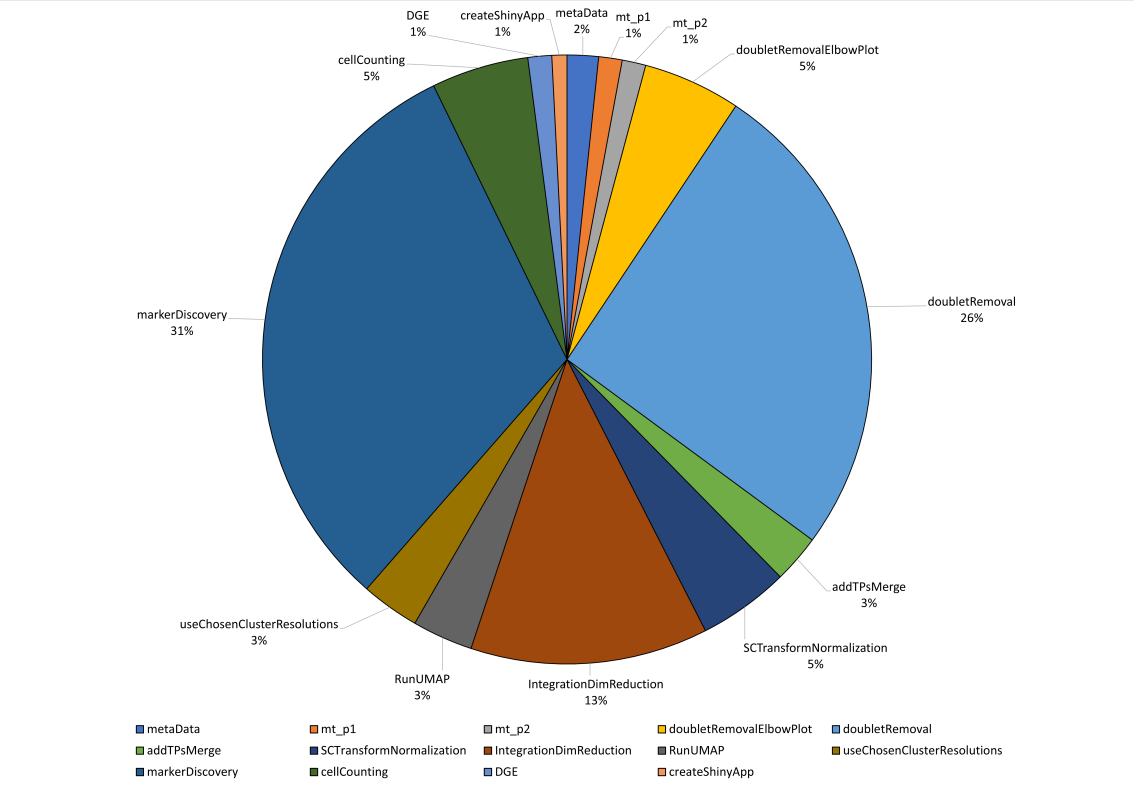


Abbildung 22: Visualisierung der Laufzeit des Seuratworkflow während der Bearbeitung des Maus67k Datensatzes, aufgeteilt nach Regeln.

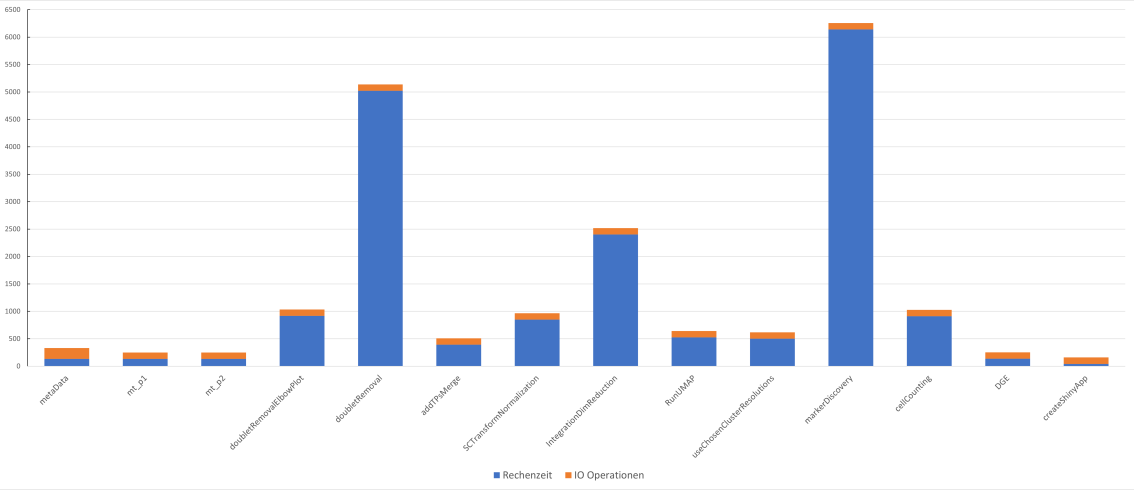


Abbildung 23: Aufteilung jeder Regel des Seuratworkflows in IO-Operationen und Rechenzeit in Sekunden.

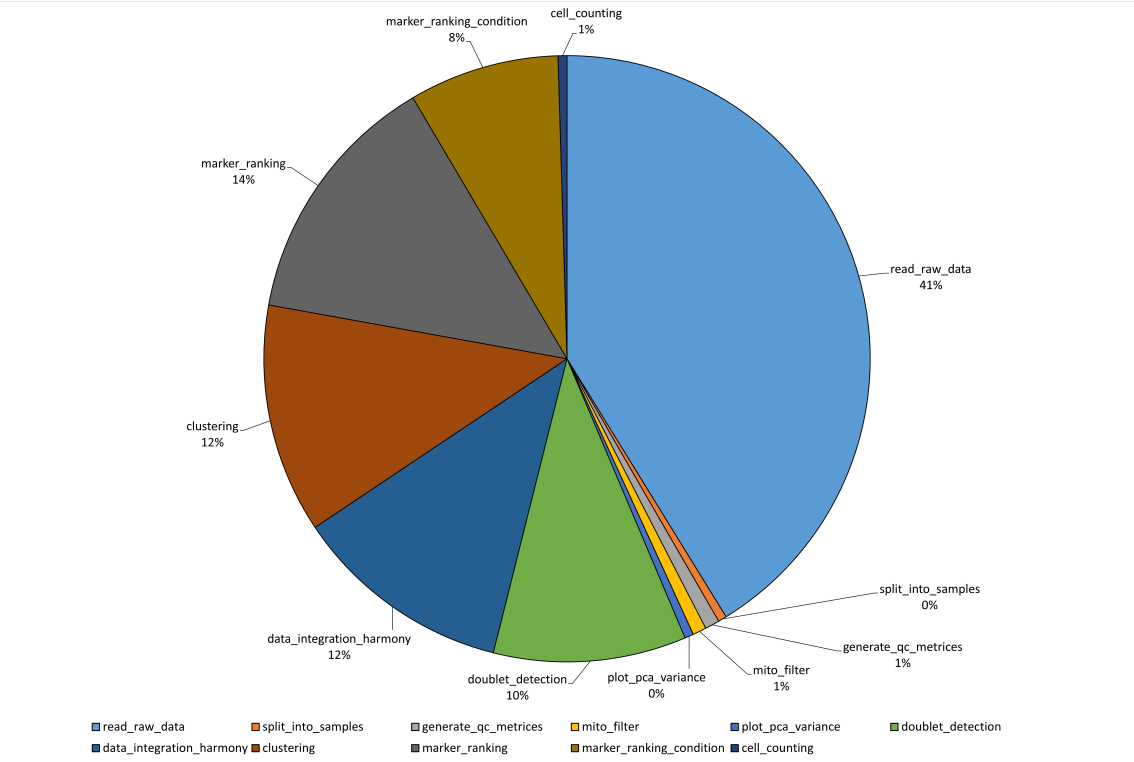


Abbildung 24: Visualisierung der Laufzeit des Scanpyworkflows während der Bearbeitung des Maus67k Datensatzes, aufgeteilt nach Regeln.

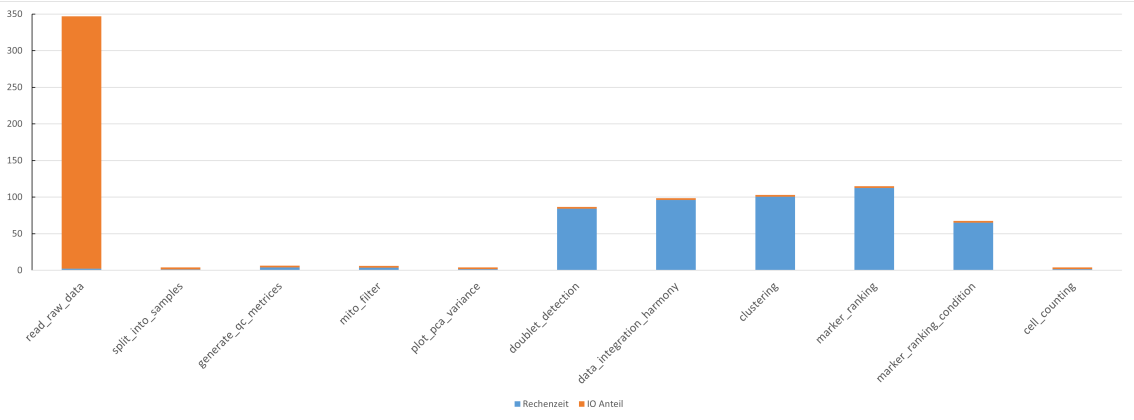


Abbildung 25: Aufteilung jeder Regel des Scanpyworkflows in IO-Operationen und Rechenzeit in Sekunden.

C Zusätzliche Grafiken

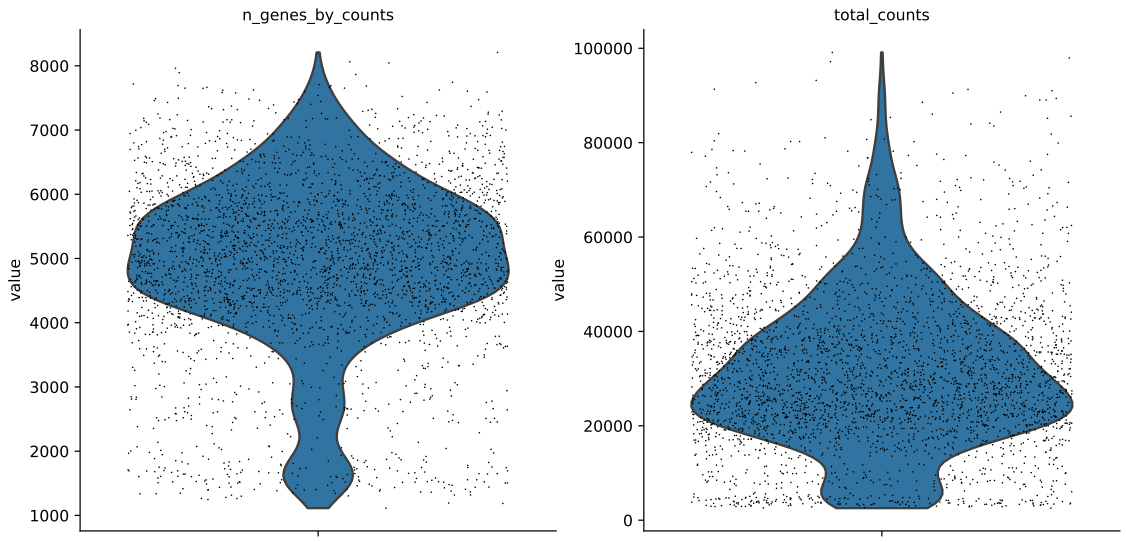


Abbildung 26: Anzahl der verschiedenen Gene und die Gesamtexpression der Barcodes, welche im Scanpyworkflow als Doublets klassifiziert wurden.

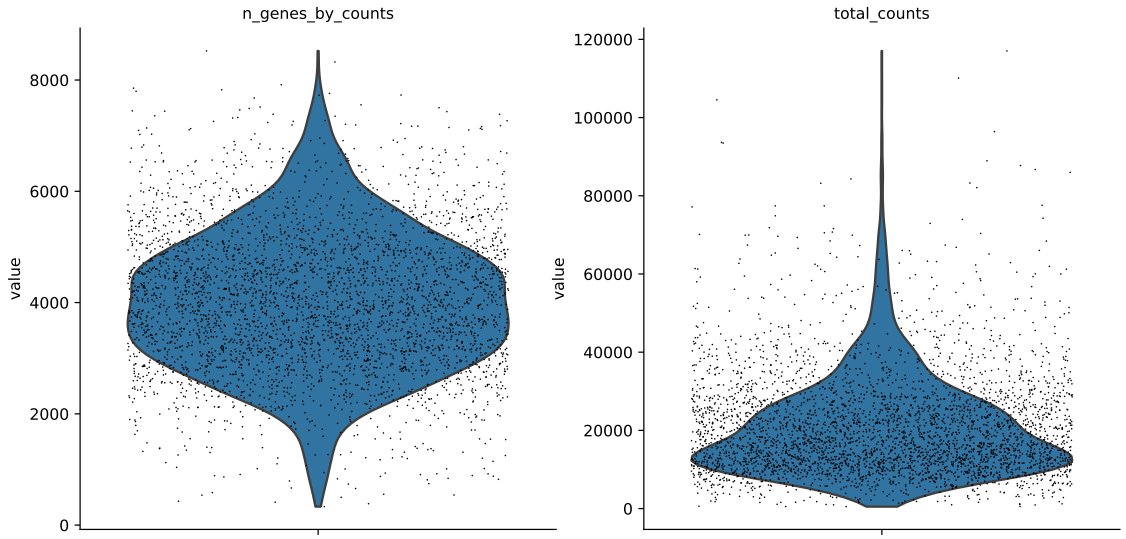


Abbildung 27: Anzahl der verschiedenen Gene und die Gesamexpression der Barcodes, welche im Seuratworkflow als Doublets klassifiziert wurden.

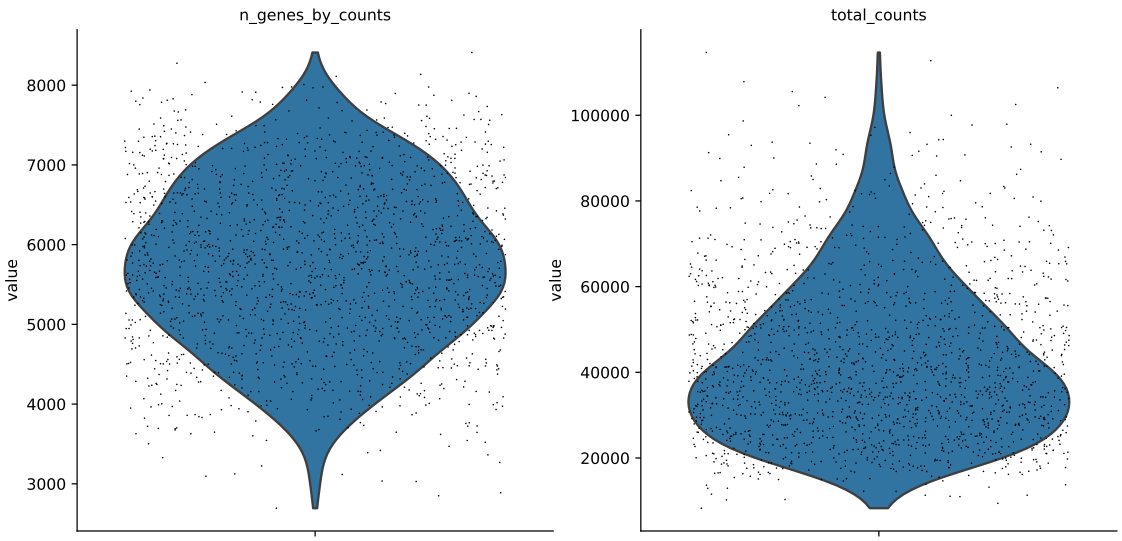


Abbildung 28: Anzahl der verschiedenen Gene und die Gesamtexpression der Barcodes, welche von beiden Workflows als Doublets klassifiziert wurden.

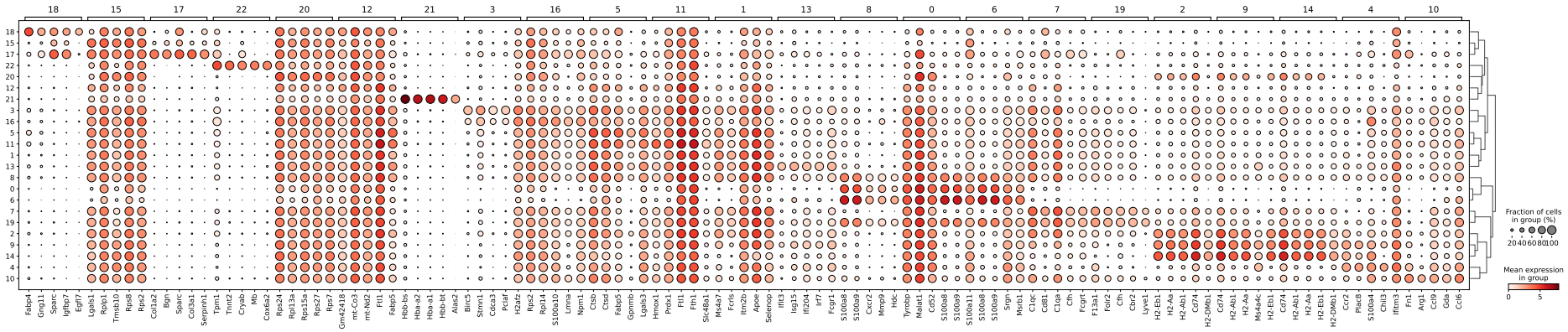


Abbildung 29: Dotplot der Expressionsrate der Top 5 Gene pro Cluster im Vergleich mit allen anderen Clustern.