

Tutorial on Stereo-seq Analysis

Louis Xiong

Table of contents

1	Cell Clustering Pipeline for Single-cell Analysis	1
1.1	Let's Dive In !	2
1.2	Quick Navi	2
2	Preprocessing	3
2.1	Summary table	3
2.2	Code	4
2.3	Summary: The Art of Preprocessing	27
3	Clustering – Cell Types Identification	27
3.1	Code	28
4	Subclass Clustering	36
4.1	Code	36
5	RCTD: Integration & Prediction	42
5.1	Code	43
6	Summary	50
6.1	Keep Exploring...	50
6.2	Keep Discovering...	50

1 Cell Clustering Pipeline for Single-cell Analysis

Author:

Deng Yuan – Coding

Xiong Yihan – Coding & Paper Writing

Date: 03/09/2026

Version: 1.0.0

Key words: Spatial Transcriptomics; Single-cell Analysis; Cell Annotation; Clustering; Quality Control

In this notebook, we will introduce the [Scanpy](#)-based pipeline we use to deal with **Annotated Data (AnnData)**, the Stereo-seq analysis workflow outputs, including main steps of **Pre-processing, Clustering, Cell Type Annotation** and related plottings. Running all the codes, we will obtain a h5ad file storing information of both spatial coordinates and cell type annotation, which can be plotted to construct a **Spatial Transcriptomic Atlas**. Here, we take sequencing data of the sparrow (*Passer Domesticus*, sample ID: cDNA_c_D06050A3_01) retinal tissue as an example.

Please Note: Workflows and functions presented in this notebook only provide a basic structure for cell annotation. In general, parameters described in each function (particularly the quality control and clustering steps) need fine-tuning for better results. Additionally, in annotation step, marker genes used to infer each cell type might vary depending on specific tissues we are studying.

1.1 Let's Dive In !

1.2 Quick Navi

- I. Preprocessing
- II. Clustering
- III. Subclustering
- IV. RCTD
- V. Summary

2 Preprocessing

Preprocessing(PP) is not just data cleaning; it determines the accuracy of all downstream analyses.

Why preprocessing is necessary – **Single-cell sequencing data are not perfect!**

For the following reasons:

1. Noise that can mask real biological signals
 - Low-quality Cells: ruptured membranes -> lose cytoplasmic RNA & retain high mitochondrial content (mtRNA)
 - Doublets: when two cells are captured in one droplet (e.g. droplet-based scRNA-seq)
 - Other Background Noise
2. Sequencing depth varies
 - The total number of counts (UMIs) per bin varies significantly due to technical factors (capture efficiency) or biological factors (cell size, RNA content & tissue heterogeneity).
3. Sparsity / Dropout
 - Single-cell data is highly sparse due to **dropout**, where genes are not expressed or genes that are expressed are not captured.
4. Batch effect
 - Technical artifacts between different batches can outweigh biological differences.

2.1 Summary table

Problems	Solutions: what PP can do
Noise	Quality control: Filter low-quality cells and doublets
Sequencing Depth	Normalization: Shifted Log Transformation / Pearson Residuals
Sparsity / Dropout	Feature selection (Highly Variable Genes, HVGs)
Batch Effect	Integration: e.g. Harmony

2.2 Code

i. Import Necessary Packages

```
import os
import anndata
import warnings
import numpy as np
import pandas as pd
import scanpy as sc
import seaborn as sns
import scrublet as scr
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from scipy.sparse import issparse
from scipy.stats import median_abs_deviation
```

ii. Define A Series of Functions

```
def calculate_qc_metrics(adata):
    adata.obs['n_counts'] = adata.X.sum(axis=1)
    adata.obs['n_genes'] = (adata.X > 0).sum(axis=1)
    adata.var['mt'] = adata.var_names.str.startswith('mt-')
    adata.obs['percent_mito'] = adata[:, adata.var['mt']].X.sum(axis=1) /
    ↪ adata.X.sum(axis=1) * 100
```

This `calculate_qc_metrics` function helps quickly calculate three important metrics of the h5ad data:

- **n_counts**: Total RNA counts (UMIs) in a single cell
- **n_genes**: The number of expressed genes detected in a single cell
- **percent_mito**: The ratio of UMIs from mitochondrial genes (starting with 'mt-')

```
def QC_violin(adata, stage, output_dir):
    qc_metrics = ['n_genes', 'n_counts', 'percent_mito']
    num_metrics = len(qc_metrics)
    num_cols = 3
    num_rows = (num_metrics + num_cols - 1) // num_cols
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5 * num_rows))
    axes = axes.flatten()
```

```

    for i, metric in enumerate(qc_metrics):
        sc.pl.violin(adata, keys=metric, groupby='batch', jitter=0,
↪ rotation=90, stripplot=False, palette='Set2', ax=axes[i], show=False)
        axes[i].set_title(f"{stage}: {metric}")
    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])
    plt.tight_layout()
    plt.savefig(f"{output_dir}/{stage}_violinPlots.pdf", dpi=50)
    plt.show()

```

The QC_violin function helps draw violin plots to visualize the three metrics.

```

def QC_dot(adata, stage, output_dir):
    nCount_RNA = adata.obs['n_counts']
    nFeature_RNA = adata.obs['n_genes']
    percent_mt = adata.obs['percent_mito']
    mt_genes = adata.var_names[adata.var_names.str.startswith('MT_')]
    if len(mt_genes) > 0:
        correlations = [
            (nCount_RNA, percent_mt, 'percent.mt'),
            (nCount_RNA, nFeature_RNA, 'nFeature_RNA')
        ]
    else:
        correlations = [
            (nCount_RNA, percent_mt, 'percent.mt'),
            (nCount_RNA, nFeature_RNA, 'nFeature_RNA')
        ]
    plt.figure(figsize=(10, 4))
    for i, (x, y, ylabel) in enumerate(correlations, 1):
        corr, _ = pearsonr(x, y)
        plt.subplot(1, len(correlations), i)
        sns.scatterplot(x=x, y=y)
        plt.title(f'nCount_RNA vs {ylabel}\nPearson correlation: {corr:.2f}')
        plt.xlabel('nCount_RNA')
        plt.ylabel(ylabel)
    plt.tight_layout()
    plt.savefig(f"{output_dir}/{stage}_dotPlots.pdf", dpi=50)
    plt.show()

```

The QC_dot function draws the scatter plot of `n_counts` vs `n_genes` to observe the relationship. High-quality cells should follow a clear linear trend.

```

def cell_number(input_path, output_file):
    results = []
    for file_name in sorted(os.listdir(input_path)):
        if file_name.endswith('.h5ad'):
            file_path = os.path.join(input_path, file_name)
            adata = sc.read_h5ad(file_path)
            cell_count = adata.n_obs
            total_gene = adata.n_vars
            mean_counts = round(adata.obs['n_counts'].mean(), 2)
            mean_genes = round(adata.obs['n_genes'].mean(), 2)
            results.append({'file_name': file_name, 'cell_count': cell_count,
↪
                                'total_gene': total_gene, 'mean_counts':
↪ mean_counts, 'mean_genes': mean_genes})
    df = pd.DataFrame(results)
    df = df.sort_values(by='file_name')
    df.to_csv(output_file, index=False)
    print(f"Cell counts saved to {output_file}")

```

The function `cell_number` outputs a table to store stats information of each h5ad file, including `cell_counts`, `total_gene`, `mean_counts` and `mean_genes`.

```

def doublets_filter(input_h5ad, output_h5ad, output_txt):
    adata = sc.read_h5ad(input_h5ad)
    if not issparse(adata.X):
        adata.X = scipy.sparse.csr_matrix(adata.X)
    if np.isnan(adata.X.data).any() or np.isinf(adata.X.data).any():
        print(f"Warning: adata.X contains NaN or Inf values in {input_h5ad}.
↪ Skipping this file.")
    #Doublets Prediction
    scrub = scr.Scrublet(adata.X, expected_doublet_rate=0.05)
    doublet_scores, predicted_doublets = scrub.scrub_doublets()
    #Checking Prediction Results
    if predicted_doublets is None:
        print(f"Warning: Scrublet failed to predict doublets for
↪ {input_h5ad}. Skipping this file.")
    adata.obs['doublet_scores'] = doublet_scores
    adata.obs['predicted_doublets'] = predicted_doublets
    #remove doublets prediction
    adata_filtered = adata[~predicted_doublets, :]
    #save as new data after doublets prediction
    adata_filtered.write(output_h5ad)

```

```

print(f"Processed {file_name} and saved as {output_h5ad}")

#save doublets_prediction information into ".txt" file
doublets_info = adata.obs[predicted_doublets]
doublets_info.to_csv(output_txt, sep='\t', index=True)
print(f"Doublets information saved to {output_txt}")

```

The `doublets_filter` function utilizes the tool `Scrublet` to predict potential doublets to be filtered.

iii. Conduct Quality Control (QC)

```

##### step1, get basic statistic before qc
h5ad_list = "h5ad.txt"
h5ad_outdir = "./1_BeforeQC_h5ad"
os.makedirs(h5ad_outdir, exist_ok=True)
h5ad_df = pd.read_csv(h5ad_list, sep="\t", names=["h5adpath", "sample",
↪ "species"])

adatas = {}
for h5ad, sam, spe in zip(h5ad_df.iloc[:, 0], h5ad_df.iloc[:, 1],
↪ h5ad_df.iloc[:, 2]):
    hread = sc.read_h5ad(h5ad)
    hread.obs_names_make_unique()
    hread.var_names_make_unique()
    hread.obs['batch'] = sam
    hread.obs['species'] = spe
    calculate_qc_metrics(hread)
    output_path = os.path.join(h5ad_outdir, f'{sam}.BeforeQC.h5ad')
    hread.write(output_path)
    adatas[sam] = hread

#visualization and calculate cell number before qc
os.makedirs("figures", exist_ok=True)
adata = sc.concat(adatas, join='outer', index_unique='-')
QC_violin(adata, "Retina_BeforeQC", "figures/")
QC_dot(adata, "Retina_BeforeQC", "figures/")
cell_number(input_path=h5ad_outdir,
↪ output_file='Retina_CellNumber_BeforeQC.csv')

```

/opt/conda/lib/python3.10/site-packages/scanpy/plotting/_anndata.py:916: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign

```
ax = sns.violinplot(  
/opt/conda/lib/python3.10/site-packages/scanpy/plotting/_anndata.py:916: FutureWarning:
```

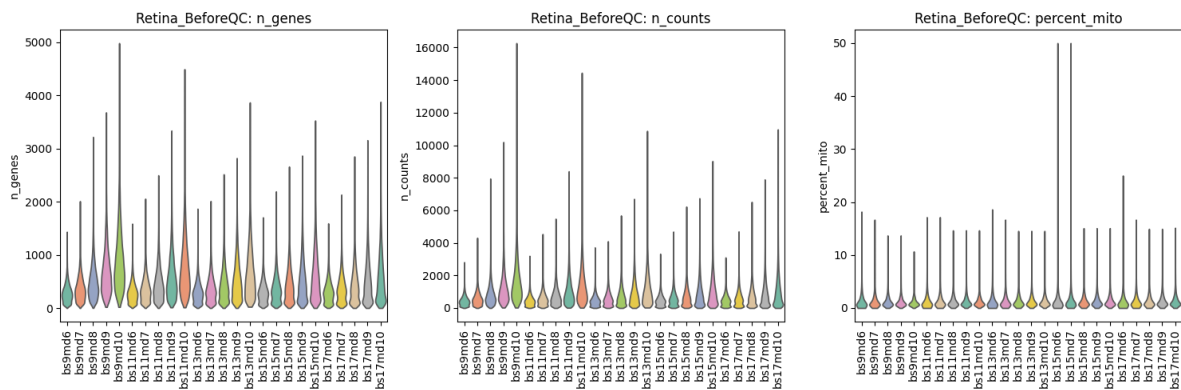
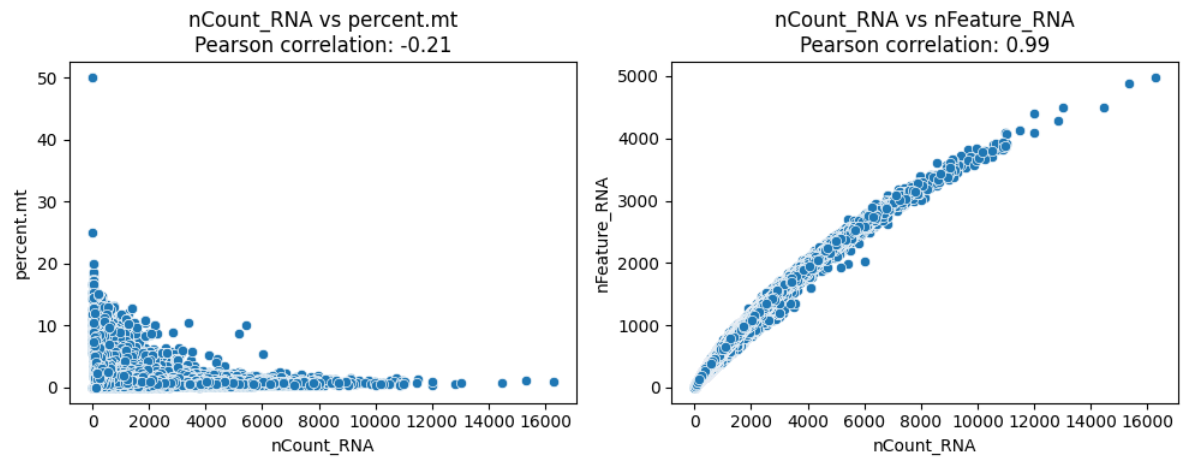
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign

```
ax = sns.violinplot(  
/opt/conda/lib/python3.10/site-packages/scanpy/plotting/_anndata.py:916: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign

```
ax = sns.violinplot(  

```



Cell counts saved to Retina_CellNumber_BeforeQC.csv

For batch processing, we create a list `h5ad_list` (three columns) to record the path to our h5ad files (adata) with different segmentation parameter values and the sample name, as well as a dictionary (`adatas = {}`) to store the h5ad files we read and operate.

`adata = sc.concat(adatas, join='outer', index_unique='-')` concatenates all Ann-data objects. `join='outer'` implies it takes the **union** of all genes (variables) in all adata objects. `index_unique='-'` appends a suffix (in this case, a hyphen -) to each CellID to avoid duplication in combined dataset. (CellID → CellID-SampleName)

As displayed above, the violin plots presents the distribution of cell numbers as measured by `n_genes`, `n_counts` and `percent_mito` in each group of parameters before quality control, allowing us to perform side-by-side comparisons.

The scatter plots demonstrate the relationship between `nCount_RNA` vs `percent.mt` / `nFeature_RNA` before quality control.

```
##### step2, quality control for each h5ad, and get basic statistic after qc
h5ad_indir = "./1_BeforeQC_h5ad/"
h5ad_outdir = "./2_AfterQC_h5ad/"
os.makedirs(h5ad_outdir, exist_ok=True)

import logging
logging.basicConfig(format='%(asctime)s %(levelname)-8s %(message)s',
    ↪ level=logging.INFO, datefmt='%Y-%m-%d %H:%M:%S')
def is_outlier(adata, metric: str, nmads: int, max_cutoff=None):
    M = adata.obs[metric]
    if max_cutoff != None:
        outlier = (M < np.median(M) - nmads * median_abs_deviation(M)) |
    ↪ (max_cutoff < M)
    else:
        outlier = (M < np.median(M) - nmads * median_abs_deviation(M)) |
    ↪ (np.median(M) + nmads * median_abs_deviation(M) < M)
    return outlier

adatas = {}
for file_name in os.listdir(h5ad_indir):
    if file_name.endswith(".h5ad"):
        sample_name = file_name.replace(".BeforeQC.h5ad", "")
        hdata = sc.read_h5ad(os.path.join(h5ad_indir, file_name))
        #QC-1: filter outliers
        sc.pp.calculate_qc_metrics(hdata, inplace=True, percent_top=[20],
    ↪ log1p=True)
        hdata.obs["outlier"] = (
```

```

is_outlier(hdata, "log1p_total_counts", 5)
    | is_outlier(hdata, "log1p_n_genes_by_counts", 5)
    | is_outlier(hdata, "pct_counts_in_top_20_genes", 5))
logging.info(hdata.obs.outlier.value_counts())
hdata.obs["mt_outlier"] = is_outlier(hdata, 'percent_mito', 3, 5)
logging.info(f"    Total number of cells: {hdata.n_obs}")
hdata = hdata[(~hdata.obs.outlier) & (~hdata.obs.mt_outlier)].copy()
#QC-2: mt (,5%), min_cells [3,). min_genes [200,), n_genes_by_counts
↪ (,5000]
sc.pp.filter_cells(hdata, min_genes=200)
sc.pp.filter_genes(hdata, min_cells=3)
hdata = hdata[hdata.obs['percent_mito'] < 5, :]
#save as new h5ad (AfterQC.h5ad)
output_file_name = f"{sample_name}.AfterQC.h5ad"
hdata.write(os.path.join(h5ad_outdir, output_file_name))
adatas[sample_name] = hdata

#visualization and calculate cell number after qc
os.makedirs("figures", exist_ok=True)
adata = sc.concat(adatas, join='outer', index_unique='-')
QC_violin(adata, "Retina_AfterQC", "figures/")
QC_dot(adata, "Retina_AfterQC", "figures/")
cell_number(input_path=h5ad_outdir,
↪ output_file='Retina_CellNumber_AfterQC.csv')

```

```

2025-10-09 16:28:06 INFO      outlier
False      19144
True       1274
Name: count, dtype: int64
2025-10-09 16:28:06 INFO      Total number of cells: 20418
2025-10-09 16:28:08 INFO      outlier
False      11374
True        899
Name: count, dtype: int64
2025-10-09 16:28:08 INFO      Total number of cells: 12273
2025-10-09 16:28:09 INFO      outlier
False      24810
True       1397
Name: count, dtype: int64
2025-10-09 16:28:09 INFO      Total number of cells: 26207
2025-10-09 16:28:12 INFO      outlier
False      28258

```

True	1315	
Name: count, dtype: int64		
2025-10-09 16:28:12	INFO	Total number of cells: 29573
2025-10-09 16:28:14	INFO	outlier
False	43922	
True	3122	
Name: count, dtype: int64		
2025-10-09 16:28:14	INFO	Total number of cells: 47044
2025-10-09 16:28:16	INFO	outlier
False	24264	
True	1312	
Name: count, dtype: int64		
2025-10-09 16:28:16	INFO	Total number of cells: 25576
2025-10-09 16:28:18	INFO	outlier
False	42569	
True	3362	
Name: count, dtype: int64		
2025-10-09 16:28:18	INFO	Total number of cells: 45931
2025-10-09 16:28:20	INFO	outlier
False	20223	
True	1254	
Name: count, dtype: int64		
2025-10-09 16:28:20	INFO	Total number of cells: 21477
2025-10-09 16:28:22	INFO	outlier
False	13950	
True	1246	
Name: count, dtype: int64		
2025-10-09 16:28:22	INFO	Total number of cells: 15196
2025-10-09 16:28:25	INFO	outlier
False	29721	
True	1410	
Name: count, dtype: int64		
2025-10-09 16:28:25	INFO	Total number of cells: 31131
2025-10-09 16:28:28	INFO	outlier
False	47521	
True	2060	
Name: count, dtype: int64		
2025-10-09 16:28:28	INFO	Total number of cells: 49581
2025-10-09 16:28:30	INFO	outlier
False	29575	
True	1668	
Name: count, dtype: int64		
2025-10-09 16:28:30	INFO	Total number of cells: 31243

```

2025-10-09 16:28:33 INFO      outlier
False      20139
True       1297
Name: count, dtype: int64
2025-10-09 16:28:33 INFO      Total number of cells: 21436
2025-10-09 16:28:36 INFO      outlier
False      35869
True       1431
Name: count, dtype: int64
2025-10-09 16:28:36 INFO      Total number of cells: 37300
2025-10-09 16:28:40 INFO      outlier
False      22918
True       1323
Name: count, dtype: int64
2025-10-09 16:28:40 INFO      Total number of cells: 24241
2025-10-09 16:28:42 INFO      outlier
False      16418
True       1232
Name: count, dtype: int64
2025-10-09 16:28:42 INFO      Total number of cells: 17650
2025-10-09 16:28:44 INFO      outlier
False      36898
True       1623
Name: count, dtype: int64
2025-10-09 16:28:44 INFO      Total number of cells: 38521
2025-10-09 16:28:47 INFO      outlier
False      36638
True       1974
Name: count, dtype: int64
2025-10-09 16:28:47 INFO      Total number of cells: 38612
2025-10-09 16:28:49 INFO      outlier
False      29291
True       1902
Name: count, dtype: int64
2025-10-09 16:28:49 INFO      Total number of cells: 31193
2025-10-09 16:28:51 INFO      outlier
False      35456
True       2656
Name: count, dtype: int64
2025-10-09 16:28:51 INFO      Total number of cells: 38112
2025-10-09 16:28:52 INFO      outlier
False      24390
True       1477

```

```

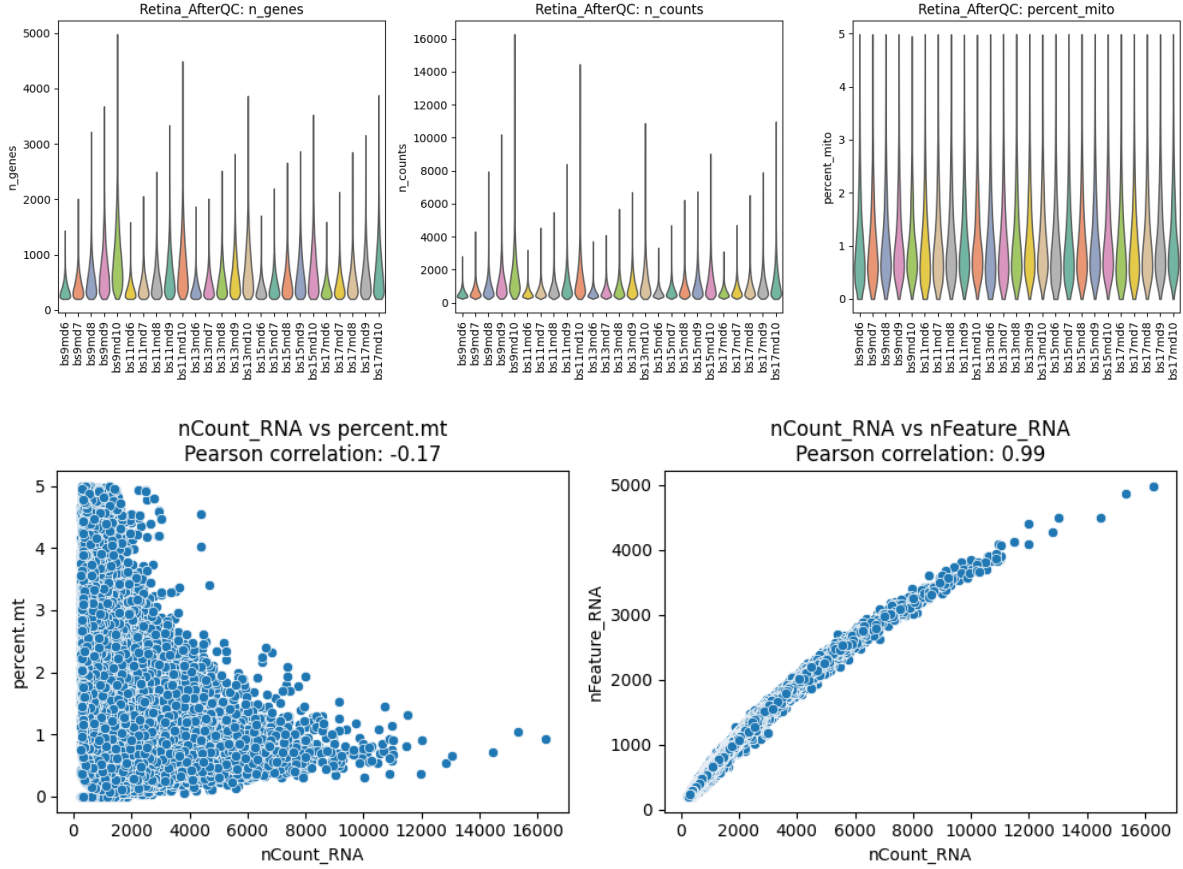
Name: count, dtype: int64
2025-10-09 16:28:52 INFO          Total number of cells: 25867
2025-10-09 16:28:53 INFO      outlier
False      16935
True       1331
Name: count, dtype: int64
2025-10-09 16:28:53 INFO          Total number of cells: 18266
2025-10-09 16:28:55 INFO      outlier
False      45417
True       2802
Name: count, dtype: int64
2025-10-09 16:28:55 INFO          Total number of cells: 48219
2025-10-09 16:28:57 INFO      outlier
False      46577
True       2306
Name: count, dtype: int64
2025-10-09 16:28:57 INFO          Total number of cells: 48883
2025-10-09 16:28:58 INFO      outlier
False      36151
True       2373
Name: count, dtype: int64
2025-10-09 16:28:58 INFO          Total number of cells: 38524
... storing 'CellID' as categorical
/opt/conda/lib/python3.10/site-packages/scanpy/plotting/_anndata.py:916: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assigning `palette` via the `hue` parameter is preferred.

    ax = sns.violinplot(
/opt/conda/lib/python3.10/site-packages/scanpy/plotting/_anndata.py:916: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assigning `palette` via the `hue` parameter is preferred.

    ax = sns.violinplot(
/opt/conda/lib/python3.10/site-packages/scanpy/plotting/_anndata.py:916: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assigning `palette` via the `hue` parameter is preferred.

    ax = sns.violinplot(

```



Cell counts saved to Retina_CellNumber_AfterQC.csv

An important step – **Quality Control**

The function `is.outlier` uses **Median Absolute Deviation (MAD)** to perform adaptive outlier detection by auto-adjusting threshold:

$$\text{Threshold} = \text{median}(M) \pm n \cdot \text{MAD}(M)$$

Where M is the metric and n is the number of deviations.

QC-1: Calculate three metrics for each batch sample and filter

- `log1p_total_counts`: Filter cells with abnormally high or low sequencing depth.
- `log1p_n_genes_by_counts`: Filter cells with too few or too many detected genes.

- `pct_counts_in_top_20_genes`: Filter cells in which a few genes account for a massive percentage of total counts (low library complexity).

Combined Filtering:

`hdata.obs["outlier"] = (...) | (...) | (...) – Remove cells that are outliers in any of these three dimensions.`

Mitochondrial Filter: `is_outlier(hdata, 'percent_mito', 3, 5)`

- Adaptive detection – 3 MADs
- Max Cutoff: any cell with > 5% mitochondrial content is filtered. (gold standard)

QC-2: Hard Thresholds – Cleanup

- `sc.pp.filter_cells(hdata, min_genes=200)`: Ensure every cell has at least 200 genes.
- `sc.pp.filter_genes(hdata, min_cells=3)`: Remove genes that express in less than 3 cells.

Filtering Steps

Read & Calculate → Adaptive Scrubbing → Mito-Control → Minimum Standard

iv. Doublets Filtering

```
##### step3, process doublets for each h5ad, and get basic statistic after
↳ doublets filtering
h5ad_indir = "./2_AfterQC_h5ad/"
h5ad_outdir = "./3_AfterDP_h5ad/"
os.makedirs(h5ad_outdir, exist_ok=True)

# Doublets filtering
for file_name in os.listdir(h5ad_indir):
    if file_name.endswith(".h5ad"):
        sample_name = file_name.replace(".AfterQC.h5ad", "")
        h5ad_infile = os.path.join(h5ad_indir, file_name)
        h5ad_outfile = os.path.join(h5ad_outdir,
↳ f"{sample_name}.AfterDP.h5ad")
        doublets_txt = os.path.join(h5ad_outdir,
↳ f"{sample_name}.Doublets.txt")
        doublets_filter(h5ad_infile, h5ad_outfile, doublets_txt)

# Visualization and calculate cell number after filtering doublets
os.makedirs("figures", exist_ok=True)
```

```

adata = {}
for file_name in os.listdir(h5ad_outdir):
    if file_name.endswith(".h5ad"):
        sample_name = file_name.replace(".AfterDP.h5ad", "")
        hdata = sc.read_h5ad(os.path.join(h5ad_outdir, file_name))
        adata[sample_name] = hdata
adata = sc.concat(adata, join='outer', index_unique='-')

QC_violin(adata, "Retina_AfterDP", "figures/")
QC_dot(adata, "Retina_AfterDP", "figures/")
cell_number(input_path=h5ad_outdir,
    ↪ output_file='Retina_CellNumber_AfterDP.csv')

```

```

Preprocessing...
Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.69
Detected doublet rate = 0.1%
Estimated detectable doublet fraction = 6.1%
Overall doublet rate:
    Expected   = 5.0%
    Estimated  = 0.9%
Elapsed time: 77.1 seconds
Processed bs15md9.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs15md9.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs15md9.Doublets.txt
Preprocessing...
Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.71
Detected doublet rate = 0.0%
Estimated detectable doublet fraction = 7.2%
Overall doublet rate:
    Expected   = 5.0%
    Estimated  = 0.4%
Elapsed time: 115.7 seconds
Processed bs9md7.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs9md7.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs9md7.Doublets.txt
Preprocessing...
Simulating doublets...
Embedding transcriptomes using PCA...

```

Calculating doublet scores...
 Automatically set threshold at doublet score = 0.62
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 12.0%
 Overall doublet rate:
 Expected = 5.0%
 Estimated = 1.0%
 Elapsed time: 50.6 seconds
 Processed bs9md10.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs9md10.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs9md10.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.72
 Detected doublet rate = 0.0%
 Estimated detectable doublet fraction = 6.9%
 Overall doublet rate:
 Expected = 5.0%
 Estimated = 0.4%
 Elapsed time: 112.6 seconds
 Processed bs13md7.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs13md7.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs13md7.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.70
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 7.3%
 Overall doublet rate:
 Expected = 5.0%
 Estimated = 0.8%
 Elapsed time: 66.0 seconds
 Processed bs11md8.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs11md8.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs11md8.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.67
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 8.9%

Overall doublet rate:
 Expected = 5.0%
 Estimated = 0.9%

Elapsed time: 109.2 seconds
 Processed bs11md6.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs11md6.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs11md6.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.72
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 6.8%

Overall doublet rate:
 Expected = 5.0%
 Estimated = 0.9%

Elapsed time: 89.5 seconds
 Processed bs11md7.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs11md7.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs11md7.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.68
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 8.1%

Overall doublet rate:
 Expected = 5.0%
 Estimated = 1.2%

Elapsed time: 63.9 seconds
 Processed bs11md9.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs11md9.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs11md9.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.65
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 7.5%

Overall doublet rate:
 Expected = 5.0%
 Estimated = 1.4%

Elapsed time: 41.7 seconds

Processed bs11md10.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs11md10.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs11md10.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.73
 Detected doublet rate = 0.0%
 Estimated detectable doublet fraction = 5.0%
 Overall doublet rate:
 Expected = 5.0%
 Estimated = 0.7%
 Elapsed time: 98.2 seconds
 Processed bs17md6.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs17md6.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs17md6.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.65
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 8.9%
 Overall doublet rate:
 Expected = 5.0%
 Estimated = 1.1%
 Elapsed time: 45.3 seconds
 Processed bs9md9.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs9md9.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs9md9.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.70
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 6.4%
 Overall doublet rate:
 Expected = 5.0%
 Estimated = 1.2%
 Elapsed time: 62.4 seconds
 Processed bs15md8.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs15md8.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs15md8.Doublets.txt
 Preprocessing...
 Simulating doublets...

```

Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.67
Detected doublet rate = 0.1%
Estimated detectable doublet fraction = 6.3%
Overall doublet rate:
    Expected    = 5.0%
    Estimated   = 1.4%
Elapsed time: 45.0 seconds
Processed bs15md10.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs15md10.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs15md10.Doublets.txt
Preprocessing...

```

```

/opt/conda/lib/python3.10/site-packages/scrublet/helper_functions.py:252: RuntimeWarning: in
CV_input = np.sqrt(b);

```

```

Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.73
Detected doublet rate = 0.0%
Estimated detectable doublet fraction = 5.7%
Overall doublet rate:
    Expected    = 5.0%
    Estimated   = 0.7%
Elapsed time: 105.6 seconds
Processed bs13md6.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs13md6.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs13md6.Doublets.txt
Preprocessing...
Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.69
Detected doublet rate = 0.1%
Estimated detectable doublet fraction = 6.2%
Overall doublet rate:
    Expected    = 5.0%
    Estimated   = 0.8%
Elapsed time: 56.1 seconds
Processed bs13md9.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs13md9.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs13md9.Doublets.txt
Preprocessing...

```

```

Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.66
Detected doublet rate = 0.1%
Estimated detectable doublet fraction = 7.4%
Overall doublet rate:
    Expected    = 5.0%
    Estimated   = 1.6%
Elapsed time: 44.2 seconds
Processed bs13md10.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs13md10.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs13md10.Doublets.txt
Preprocessing...
Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.73
Detected doublet rate = 0.0%
Estimated detectable doublet fraction = 5.5%
Overall doublet rate:
    Expected    = 5.0%
    Estimated   = 0.4%
Elapsed time: 105.7 seconds
Processed bs15md6.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs15md6.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs15md6.Doublets.txt
Preprocessing...
Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.70
Detected doublet rate = 0.1%
Estimated detectable doublet fraction = 6.5%
Overall doublet rate:
    Expected    = 5.0%
    Estimated   = 0.9%
Elapsed time: 65.4 seconds
Processed bs17md8.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs17md8.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs17md8.Doublets.txt
Preprocessing...
Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.72

```

```

Detected doublet rate = 0.0%
Estimated detectable doublet fraction = 5.8%
Overall doublet rate:
    Expected    = 5.0%
    Estimated    = 0.5%
Elapsed time: 55.9 seconds
Processed bs17md7.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs17md7.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs17md7.Doublets.txt
Preprocessing...

```

```

/opt/conda/lib/python3.10/site-packages/scrublet/helper_functions.py:252: RuntimeWarning: in
    CV_input = np.sqrt(b);

```

```

Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.73
Detected doublet rate = 0.0%
Estimated detectable doublet fraction = 5.8%
Overall doublet rate:
    Expected    = 5.0%
    Estimated    = 0.3%
Elapsed time: 80.0 seconds
Processed bs9md6.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs9md6.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs9md6.Doublets.txt
Preprocessing...
Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...
Automatically set threshold at doublet score = 0.69
Detected doublet rate = 0.1%
Estimated detectable doublet fraction = 7.4%
Overall doublet rate:
    Expected    = 5.0%
    Estimated    = 1.4%
Elapsed time: 49.3 seconds
Processed bs9md8.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs9md8.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs9md8.Doublets.txt
Preprocessing...
Simulating doublets...
Embedding transcriptomes using PCA...
Calculating doublet scores...

```

Automatically set threshold at doublet score = 0.70
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 7.1%
 Overall doublet rate:
 Expected = 5.0%
 Estimated = 1.2%
 Elapsed time: 48.6 seconds
 Processed bs13md8.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs13md8.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs13md8.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.67
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 7.3%
 Overall doublet rate:
 Expected = 5.0%
 Estimated = 1.2%
 Elapsed time: 31.9 seconds
 Processed bs17md10.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs17md10.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs17md10.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.68
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 6.7%
 Overall doublet rate:
 Expected = 5.0%
 Estimated = 1.3%
 Elapsed time: 39.6 seconds
 Processed bs17md9.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs17md9.AfterDP.h5ad
 Doublets information saved to ./3_AfterDP_h5ad/bs17md9.Doublets.txt
 Preprocessing...
 Simulating doublets...
 Embedding transcriptomes using PCA...
 Calculating doublet scores...
 Automatically set threshold at doublet score = 0.72
 Detected doublet rate = 0.1%
 Estimated detectable doublet fraction = 7.3%
 Overall doublet rate:

```

Expected    = 5.0%
Estimated   = 0.8%
Elapsed time: 63.9 seconds
Processed bs15md7.AfterQC.h5ad and saved as ./3_AfterDP_h5ad/bs15md7.AfterDP.h5ad
Doublets information saved to ./3_AfterDP_h5ad/bs15md7.Doublets.txt

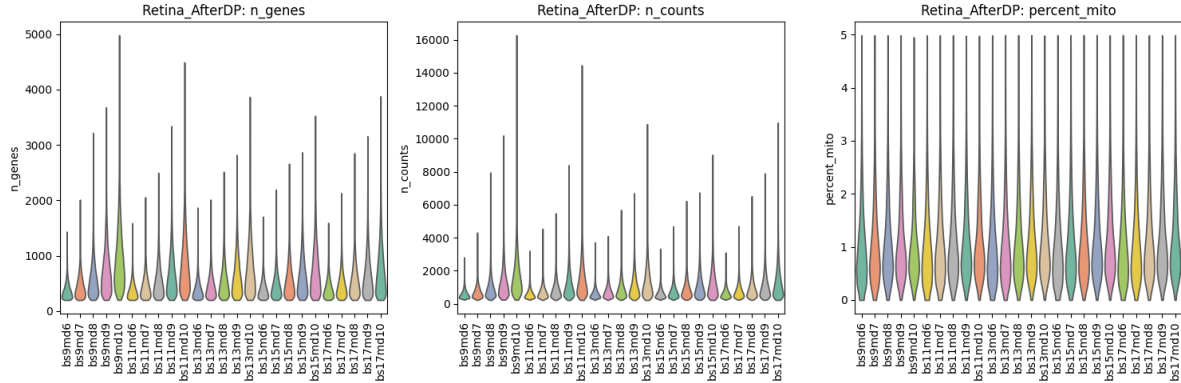
... storing 'CellID' as categorical
/opt/conda/lib/python3.10/site-packages/scanpy/plotting/_anndata.py:916: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assigning `palette` via the `hue` parameter is preferred.

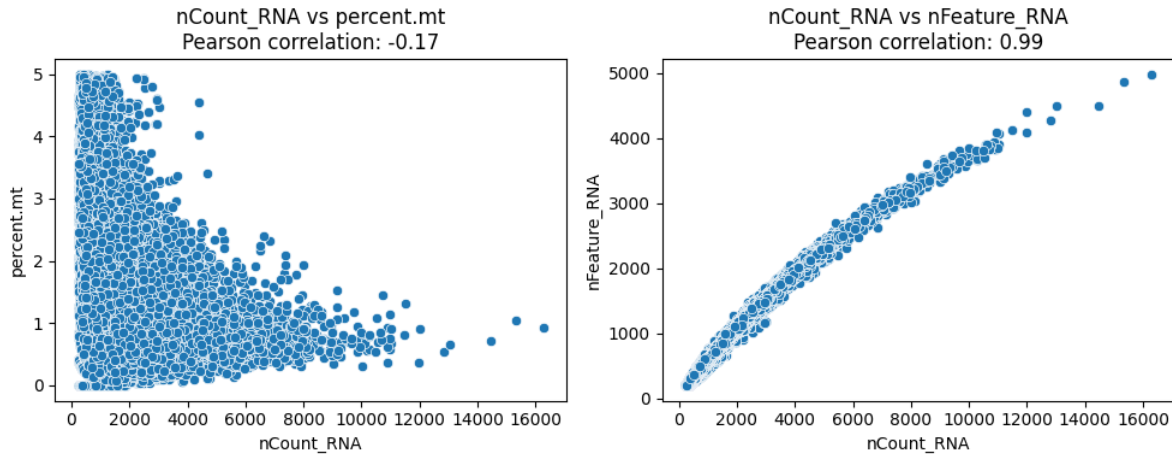
ax = sns.violinplot(
/opt/conda/lib/python3.10/site-packages/scanpy/plotting/_anndata.py:916: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assigning `palette` via the `hue` parameter is preferred.

ax = sns.violinplot(
/opt/conda/lib/python3.10/site-packages/scanpy/plotting/_anndata.py:916: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assigning `palette` via the `hue` parameter is preferred.

ax = sns.violinplot(

```





Cell counts saved to Retina_CellNumber_AfterDP.csv

The function `doublet_filter` is called for filtering doublets in each batch. In this example, the estimated doublet rate is 5%.

```
##### step4, normalization for each h5ad
h5ad_indir = "./3_AfterDP_h5ad/"
h5ad_outdir = "./4_AfterNL_h5ad/"
os.makedirs(h5ad_outdir, exist_ok=True)

# Normalization
for file_name in os.listdir(h5ad_indir):
    if file_name.endswith(".h5ad"):
        sample_name = file_name.replace(".AfterDP.h5ad", "")
        h5ad_infile = os.path.join(h5ad_indir, file_name)
        h5ad_outfile = os.path.join(h5ad_outdir,
        ↪ f"{sample_name}.AfterNL.h5ad")
        adata = sc.read_h5ad(h5ad_infile)
        adata.layers["counts"] = adata.X.copy()
        sc.pp.normalize_total(adata, target_sum=1e4)
        sc.pp.log1p(adata)
        adata.layers["log1p"] = adata.X.copy()
        adata.X = adata.layers["counts"].copy()
        adata.write_h5ad(h5ad_outfile, compression='gzip')
```

Normalization aims to make gene expression levels comparable across cells. **Shifted Log Transformation** and **Pearson Residuals** are two main tricks. Here, we choose the classic method – **Shifted Log Transformation**.

a. Scaling Step

```
sc.pp.normalize_total(adata, target_sum=1e4)
```

The formula of:

$$Y_{ig} = \ln \left(\frac{X_{ig}}{\sum_g X_{ig}} \cdot s + 1 \right)$$

Where X_{ig} is the count of gene g in cell i , and s is the scaling factor (`target_sum`, in this case, 10^4).

If `target_sum` is `None`, the median of total counts across cells will be calculated and used as the scaling factor for every cell.

b. Log-step

```
sc.pp.log1p(adata)
```

The $X = \ln(X + 1)$ transformation compresses the differences between high values, stabilizing the variance of high-expression genes. (In sequencing data, genes with higher average expression also have much higher variance.)

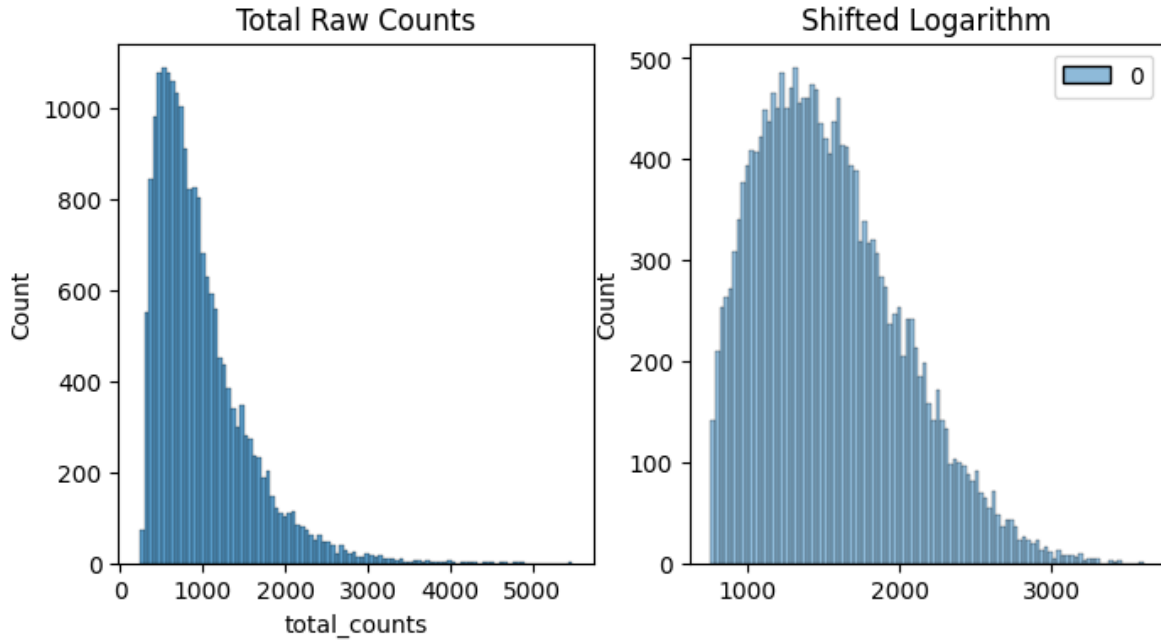
Raw count data is highly skewed (long tail to the right). After normalization, the distribution of gene expression is more symmetric and bell-shaped, approaching a **Gaussian distribution**, which satisfies the statistical assumptions of downstream tools (e.g. PCA).

We can take a look:

```
print(adata.layers)

fig, axes = plt.subplots(1, 2, figsize = (8, 4))
p1 = sns.histplot(adata.obs["total_counts"], bins=100, kde=False, ax=axes[0])
axes[0].set_title("Total Raw Counts")
p2 = sns.histplot(adata.layers["log1p"].sum(1), bins=100, kde=False,
    ↪ ax=axes[1])
axes[1].set_title("Shifted Logarithm")
plt.show()
```

Layers with keys: counts, log1p



2.3 Summary: The Art of Preprocessing

“Quality In, Quality Out”

Just like washing and cutting ingredients before cooking: if the ingredients (data) is contaminated with sand (low-quality cells or technical noise), even the most sophisticated chef (algorithm) cannot produce a reliable dish (biological conclusion).

By performing **Quality Control** and **Normalization**, we have successfully transformed a raw, noisy count matrix into a clean, comparable and statistically sound h5ad dataset, ready for downstream analyses.

3 Clustering – Cell Types Identification

This section will bring us to probably the most thrilling part in single-cell analysis – Clustering.

Clustering is an unsupervised learning process that groups cells with similar gene expression patterns together. In single-cell analysis, we do not use all genes; Instead, we project the cells into a low-dimensional space (this process is called **Dimensionality Reduction**) using

algorithms like PCA (Principal Component Analysis) and build a **Neighbor Graph** to find clusters – that are, communities of cells. These communities often correspond to specific cell types or functional states.

Clustering separates different cell groups, making it possible to perform **Cell Annotation**, the step to identify cell types for each cluster.

Now let us take a look at how clustering is realized.

3.1 Code

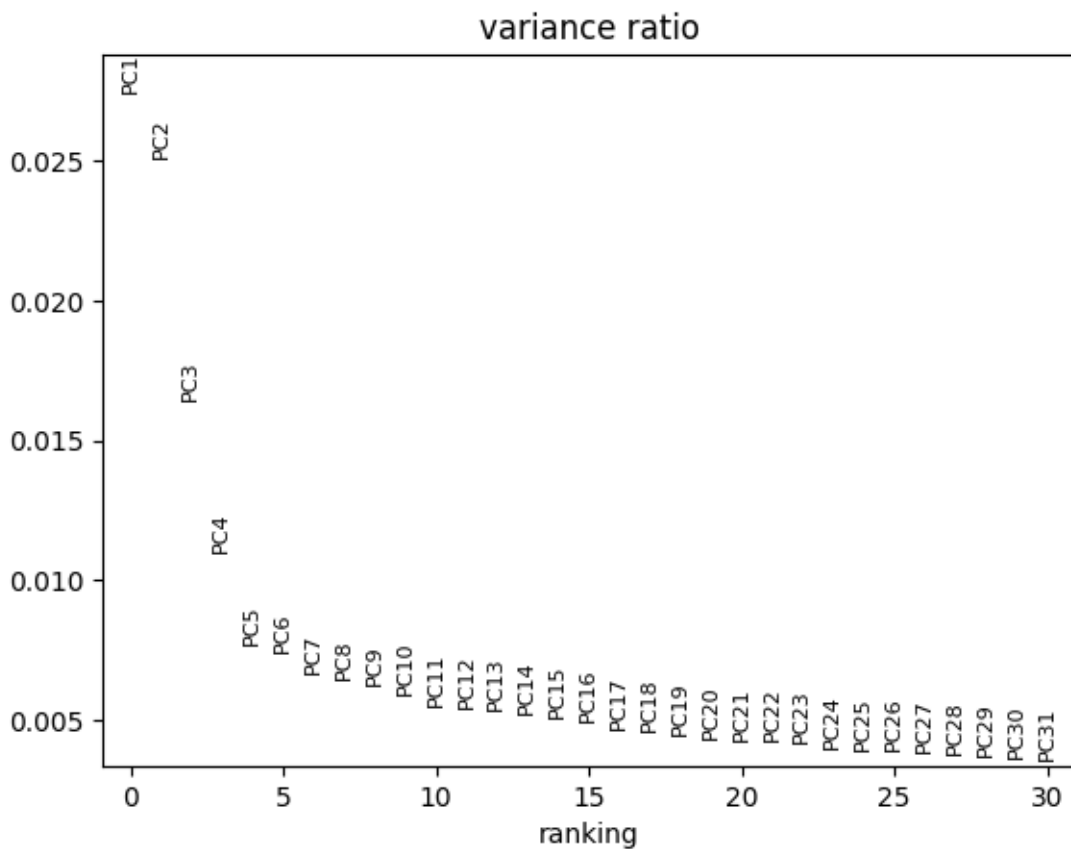
```
h5ad_in = "4_AfterNL_h5ad/bs11md8.AfterNL.h5ad"
h5ad_out = "5_Cluster_h5ad/bs11md8.Seurat-3000_npcs30.h5ad"
prefix = os.path.basename(h5ad_out).split(".h5ad")[0]

adata = sc.read_h5ad(h5ad_in)
## highly variable genes
batch_key = "batch"
adata.X = adata.layers["log1p"].copy()
sc.pp.highly_variable_genes(adata, n_top_genes=3000, batch_key= batch_key)

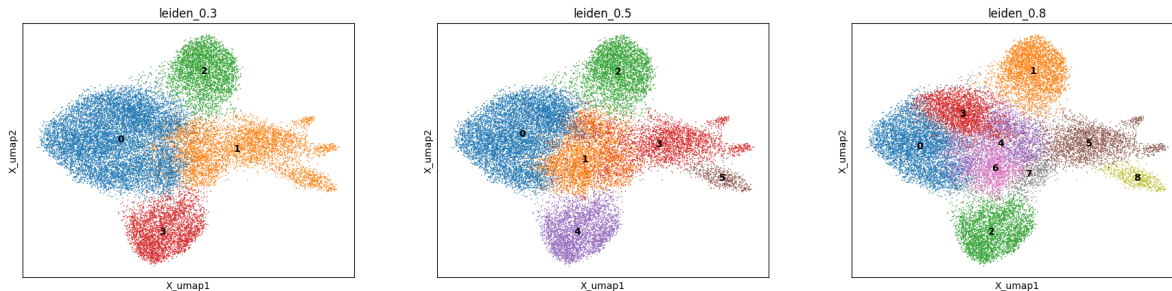
## integration graphs
adata_hvg = adata[:, adata.var.highly_variable].copy()
# dimensionality reduction
sc.tl.pca(adata_hvg, n_comps=50)
sc.pl.pca_variance_ratio(adata_hvg, n_pcs=30)
variance_ratio = adata_hvg.uns['pca']['variance_ratio']
cum_var = np.cumsum(variance_ratio)
print(cum_var)

# visualization
sc.pp.neighbors(adata_hvg, n_pcs=30)
sc.tl.umap(adata_hvg)

for resolution in [0.3,0.5,0.8]:
    key = f'leiden_{resolution}'
    sc.tl.leiden(adata_hvg, resolution=resolution, key_added=key)
cell_clusters = [f"leiden_{res}" for res in [0.3,0.5,0.8]]
sc.pl.embedding(adata_hvg, basis=f"X_umap", color=cell_clusters, ncols=3,
    ↪ legend_loc="on data")
```



```
[0.0274271  0.05252235 0.06897991 0.07999188 0.08767833 0.09507739
0.10173637 0.10823592 0.11450795 0.12045795 0.1259786  0.1314163
0.13675761 0.1419691  0.14705968 0.1520384  0.15671286 0.1613106
0.16579221 0.17013283 0.17441842 0.17866398 0.18284488 0.18687083
0.19082421 0.1947394  0.19857015 0.20235774 0.20607363 0.20972194
0.21327946 0.21682866 0.2202198  0.22354734 0.22684036 0.23008855
0.23332904 0.23655245 0.23970899 0.24280179 0.24587244 0.24886717
0.25184187 0.254757   0.25762734 0.2604785  0.26328495 0.2660734
0.2688347  0.27156976]
```



Scanpy offers a set of functions to run clustering:

1. `sc.pp.highly_variable_genes` – Feature Gene Selection

- Purpose: To separate biological signals from technical noise. We only focus on genes that vary significantly across cells.
- Function: Ranks genes by their mean-variance relationship and select the top N genes (normally 2,000 - 3,000, here we choose `n_top_genes = 3000`).
- `batch_key`: Specifying `batch_key` ensures genes are selected **within** each batch to avoid picking up batch effect as variation.

2. `sc.tl.pca`: Dimensionality Reduction (PCA)

- Purpose: To compress the thousands of selected genes into a few dozen **Principal Components (PCs)** that capture the most variance. Here, we select 50 most informative PCs `n_comps = 50`, which is usually plenty.
- Function: Using a linear transformation to project data into a lower-dimensional space while preserving as much diversity as possible.
- Note: PCA should only be run on the **Highly Variable Genes (HVGs)** identified in the previous step.
- Other dimensionality reduction algorithms: UMAP, tSNE, etc.

3. `sc.pl.pca_variance_ratio`: The Diagnostic Elbow Graph

- Purpose: To determine how many PCs are actually meaningful before noise starts to dominate.
- Function: Plots the percentage of total variance explained by each PC.
- The ‘elbow rule’: Look for the point where the curve starts to flatten (the ‘elbow’). This is where the return on adding more PCs diminishes.

4. `sc.pp.neighbors`: Building Neighbor Graph

- Purpose: To build a mathematical network where cells are connected to their most similar peers.
- Function: Calculates the k-nearest neighbors (KNN) for each cell in the low-dimensional PC space.

5. `sc.tl.umap`: Projection

- Purpose: To project the complex multidimensional graph into a 2D plot that we can easily interpret.
- Function: A non-linear algorithm trying to keep neighbors close together in 2D space.
- UMAP is only for visualization.

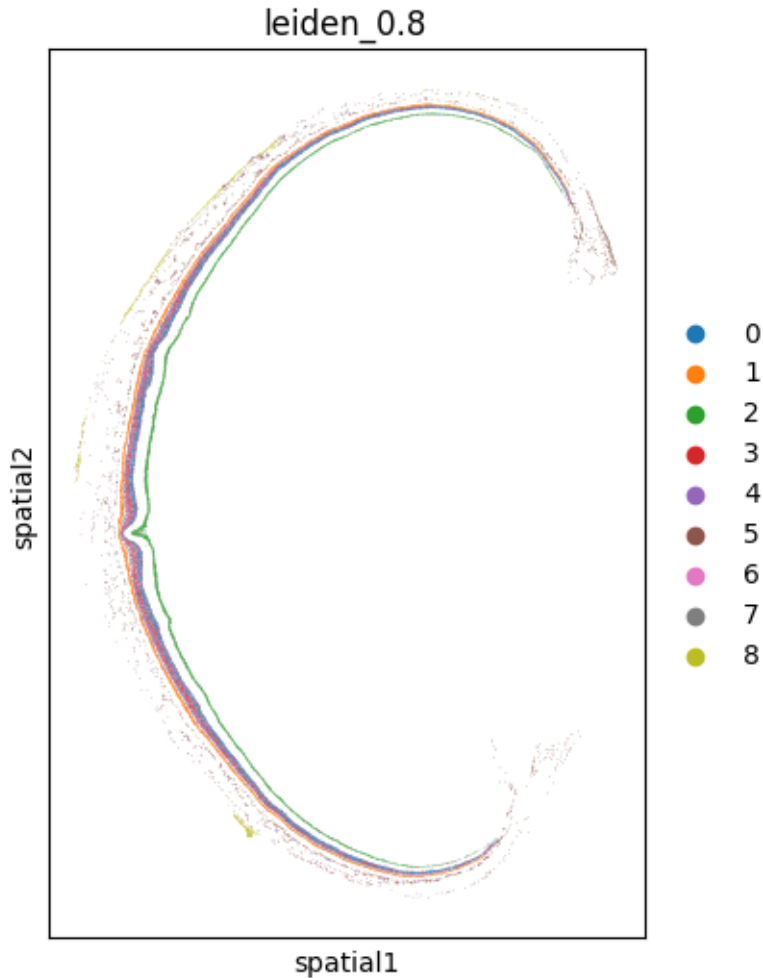
6. `sc.tl.leiden`: Clustering with Leiden

- Function: To group cells based on their transcriptional similarity (identifying cell communities that are more closely connected to each other than to the rest of the network).
- Key parameters:
 - **resolution**: Controls how coarse or fine the clusters are. Lower resolution – fewer, larger clusters; Higher resolution – more, smaller clusters.
 - **key_added**: Specifies the column name in `adata.obs` where the cluster labels will be stored. When testing multiple resolutions, give them unique names to avoid overwriting.

```
for annot in ['obs', 'uns', 'obsm', 'obsp']:
    for key in eval(f"adata_hvg.{annot}"):
        if key not in eval(f"adata.{annot}"):
            try:
                exec(f"adata.{annot}['{key}'] =
                    ↪ adata_hvg.{annot}['{key}'].copy()")
            except:
                exec(f"adata.{annot}['{key}'] = adata_hvg.{annot}['{key}']")
adata.X = adata.layers['counts'].copy()
adata.write_h5ad(h5ad_out, compression='gzip')

from matplotlib import rc_context
with rc_context({'figure.figsize': (4, 6)}):
    name = prefix.split(".")[0] + "_leiden_0.8"
    sc.pl.embedding(adata, basis="spatial", color='leiden_0.8', save=('.' +
    ↪ name + '.pdf'), size=0.6)
```

WARNING: saving figure to file figures/spatial.bs11md8_leiden_0.8.pdf



This script iterates through the main annotation slots of the `Anndata` object and dynamically copies the metadata from the HVG project to the full project. It then generates a spatial plot where each point represents a cell's physical location, colored by its Leiden cluster ID at a certain resolution (0.8 in this case).

tip: `adata.X = adata.layers['counts'].copy()` – It is the best practice to store the raw counts in `.X` when saving the `.h5ad` file so that other tools could use the raw data directly.

Clustering has identified distinct cell groups, yet their specific biological identities remain unassigned. So the next step is to perform **Cell Type Annotation**.

To annotate a cluster, one strategy is to look for marker genes through a **Dot Plot / Bubble Plot**.

A bubble plot (often called a `dotplot` in Scanpy) visualizes two critical dimensions of gene expression simultaneously across different clusters:

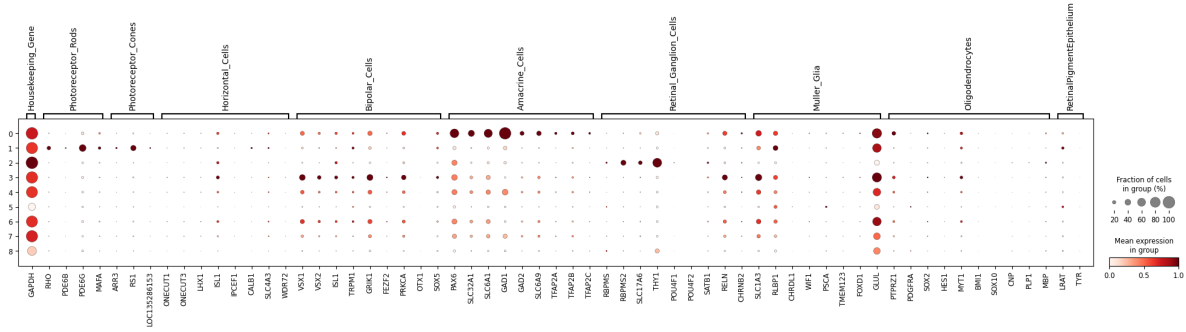
- Color – Mean Expression: Represents how strongly a gene is expressed on average in a cluster. Darker (redder) means higher expression.
- Size – Fraction of Cells: Represents what percentage of cells in a cluster actually express the gene. Larger dots mean more cells are expressing.

```
retina_marker_genes = {
    "Housekeeping_Gene": ["GAPDH"],
    "Photoreceptor_Rods": ["RHO", "PDE6B", "PDE6G", "MAFA"],
    "Photoreceptor_Cones": ["ARR3", "RS1", "OPN1SW", "OPN2SW", "OPN1LW",
        ↪ "LOC135286153"],
    "Horizontal_Cells": ["ONECUT1", "ONECUT2", "ONECUT3", "LHX1", "ISL1",
        ↪ "IPCEF1", "OXT", "CALB1", "SLC4A3", "WDR72"],
    "Bipolar_Cells": ["VSX1", "VSX2", "ISL1", "TRPM1", "GRIK1", "FEZF2",
        ↪ "PRKCA", "OTX1", "SOX5", "CABP5"],
    "Amacrine_Cells": ["PAX6", "PROX1", "SLC32A1", "SLC6A1", "GAD1", "GAD2",
        ↪ "SLC6A9", "TFAP2A", "TFAP2B", "TFAP2C"],
    "Retinal_Ganglion_Cells": ["RBPMs", "RBPMs2", "SLC17A6", "THY1",
        ↪ "POU4F1", "POU4F2", "POU4F3", "SATB1", "RELN", "CHRNA2", "SS2"],
    "Muller_Glia": ["SLC1A3", "RLBP1", "CHRD1", "WIF1", "PSCA", "TMEM123",
        ↪ "FOXD1", "GLUL"],
    "Oligodendrocytes": ["OLIG2", "PTPRZ1", "PDGFRA", "SOX2", "HES1", "MYT1",
        ↪ "BMI1", "SOX10", "CNP", "PLP1", "MBP", "BACS1"],
    "RetinalPigmentEpithelium": ["RPE65", "PMEL", "BEST1", "LRAT", "TYR"]
}

marker_in_adata = {cell_type: [gene for gene in genes if gene in
    ↪ adata.var_names]
                    for cell_type, genes in retina_marker_genes.items()}

sc.pl.dotplot(adata,
               marker_in_adata,
               groupby='leiden_0.8',
               standard_scale="var",
               swap_axes=False,
               save='markers_'+prefix+'_res0.8.png')
```

WARNING: saving figure to file figures/dotplot_markers_bs11md8.Seurat-3000_npcs30_res0.8.png



In this script, we build a dictionary to store marker genes of avian retinal cells. For example, the marker genes of **Photoreceptor Rods** include *RHO*, *PDE6B*, *PDE6G* and *MAFA*.

The function `sc.pl.dotplot` plots the dotplot. Always use `standard_scale='var'` to scale each gene from 0 to 1 across the clusters, making it easier to observe which cluster owns which genes regardless of the absolute expression levels.

In our case, cluster 0, for instance, exhibits much darker and larger dots for **Amacrine Cell** markers than for those of other cell types, indicating its identity.

Cell-type-specific markers are usually derived from previously published reports. A cell type is defined by a set of genes (signature), so it is always favored to search multiple markers confirming the same identity, and collect a wealth of knowledge on tissues, cells as well as their biological markers.

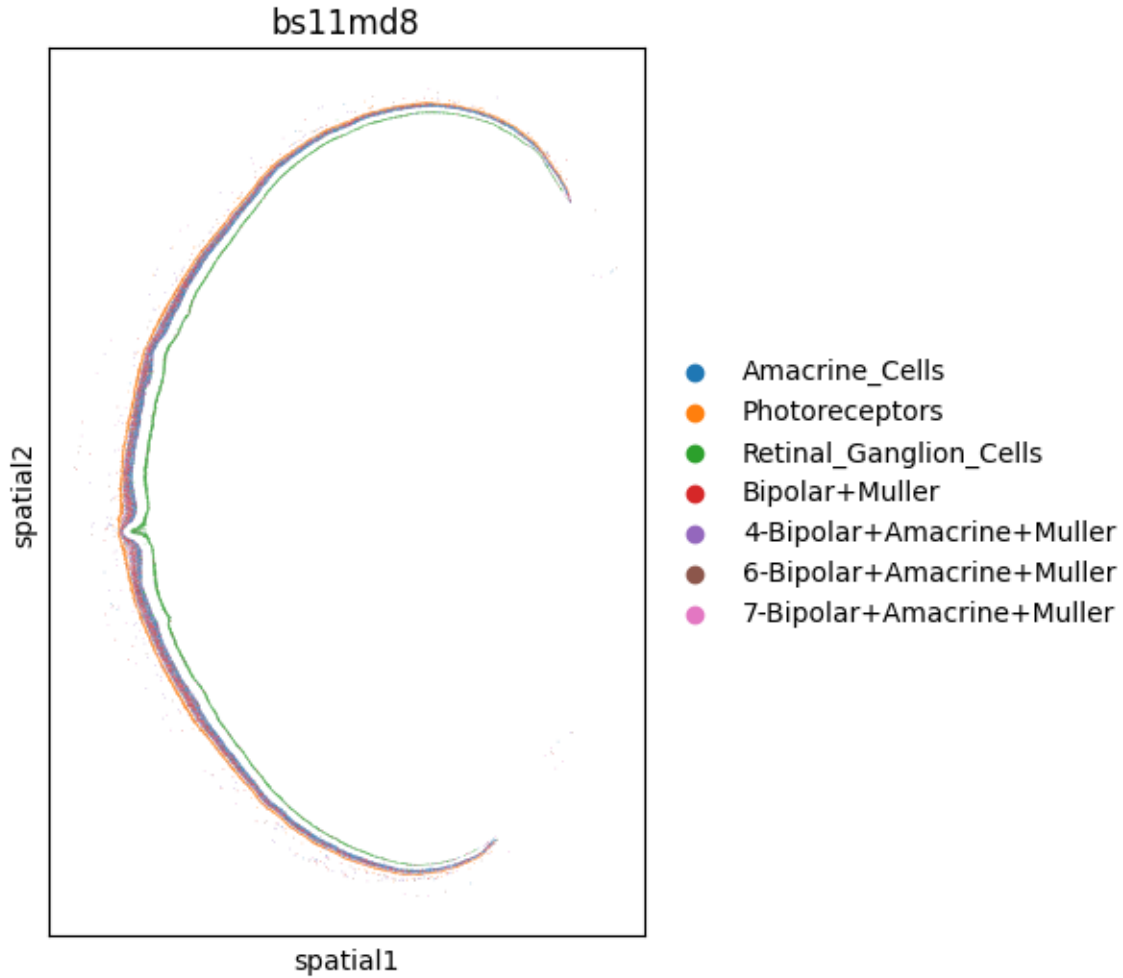
Once you observe the bubble plot patterns match the published literature – Congrats, you have successfully decoded the biological identities of your sample.

```
h5ad_out = "5_Cluster_h5ad/bs11md8.Seurat-3000_npcs30.ANN.h5ad"
prefix = os.path.basename(h5ad_out).split(".h5ad")[0].split(".")[0]

bdata = adata[~adata.obs['leiden_0.8'].isin(['5','8'])].copy()
Annotation_marker = {
    "0": "Amacrine_Cells", "1": "Photoreceptors", "2":
        ↪ "Retinal_Ganglion_Cells", "3": "Bipolar+Muller",
    "4": "4-Bipolar+Amacrine+Muller", "6": "6-Bipolar+Amacrine+Muller",
    "7": "7-Bipolar+Amacrine+Muller"}
bdata.obs["celltype"] = bdata.obs["leiden_0.8"].map(Annotation_marker)
bdata.X = bdata.layers['counts'].copy()
bdata.write_h5ad(h5ad_out, compression='gzip')
bdata.obs.to_csv(h5ad_out + '.obs.txt', sep="\t", index=True)

with rc_context({'figure.figsize': (4, 6)}):
    name = prefix.split(".")[0] + "_leiden_0.8.ANN"
    sc.pl.embedding(bdata, basis='spatial', color='celltype', save=('.' +
        ↪ name + '.pdf'), title=f"{prefix}", size=0.6)
```

WARNING: saving figure to file figures/spatial.bs11md8_leiden_0.8.ANN.pdf



Using the dotplot as a reference, we manually assign the identity of each cell cluster, then add the annotated information to the `Anndata` object, stored in a new observation `adata.obs["celltype"]`. It is a pity that in our example, cluster 5 and cluster 8 have not showed distinct dots for markers of any cell type, so their identities remain unknown. Also, cluster 4, 6 and 7 simultaneously exhibit more prominent expression of marker genes of more than one cell types, suggesting that there might be a mixture.

Therefore, further annotation is required for more accurate identification.

4 Subclass Clustering

One of the solutions to dealing with groups of mixed cells is to **Re-Cluster** them. This idea is derived from **Cell Subclass Annotation**: to extract cells in the clusters of interest, and re-run dimensionality reduction to identify the cell types of new clusters. This could remove the noise from other cell types, allowing us to focus exclusively on the differences within the specific groups.

4.1 Code

```
def umapRun(adata, batch, prefix):
    batch_key = batch
    adata_hvg = adata[:, adata.var.highly_variable].copy()

    # dimensionality reduction
    sc.tl.pca(adata_hvg, n_comps=50)
    sc.pl.pca_variance_ratio(adata_hvg, n_pcs=50)
    variance_ratio = adata_hvg.uns['pca']['variance_ratio']
    cum_var = np.cumsum(variance_ratio)
    print(cum_var)

    # integration with harmony
    # adata_hvg.obsm["Harmony"] = harmonize(adata_hvg.obsm["X_pca"],
    ↪ adata_hvg.obs, batch_key= batch_key)

    # visualization
    sc.pp.neighbors(adata_hvg, use_rep='X_pca')
    sc.tl.umap(adata_hvg)

    for resolution in [0.3,0.5,0.8,1.0]:
        key = f'leiden_subclass_{resolution}'
        sc.tl.leiden(adata_hvg, resolution=resolution, key_added=key)
        cell_clusters = [f"leiden_subclass_{res}" for res in [0.3,0.5,0.8,1.0]]
        sc.pl.embedding(adata_hvg, basis=f"X_umap", color=cell_clusters, ncols=4,
        ↪ legend_loc="on data", save="." + prefix + ".clusters.pdf")

    for annot in ['obs', 'uns', 'obsm', 'obsp']:
        for key in eval(f"adata_hvg.{annot}"):
            if key not in eval(f"adata.{annot}"):
                try:
                    exec(f"adata.{annot}['{key}'] =
                    ↪ adata_hvg.{annot}['{key}'].copy()")
```

```

        except:
            exec(f"adata.{annot}['{key}'] =
                ↪ adata_hvg.{annot}['{key}']")
adata.obsm['X_subclass_umap'] = adata_hvg.obsm['X_umap'].copy()
return adata

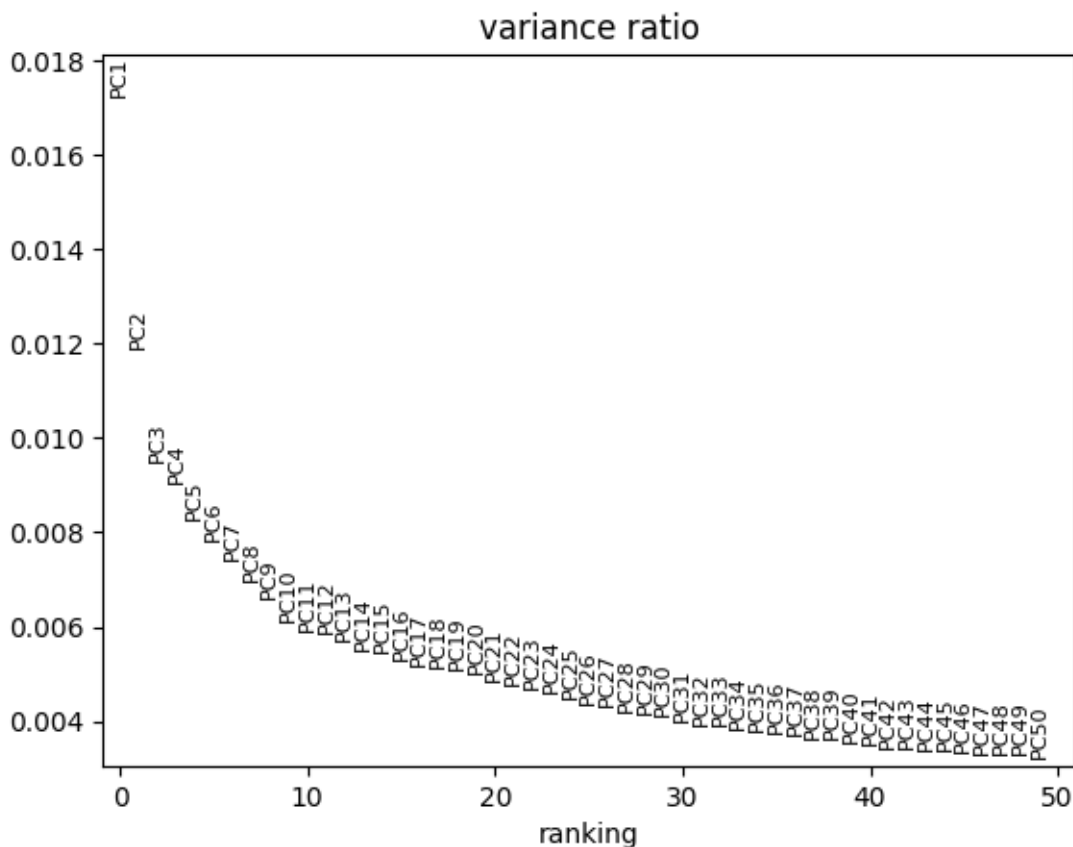
```

First, we define the function `umapRun` to conduct dimensionality reduction, clustering and visualization.

```

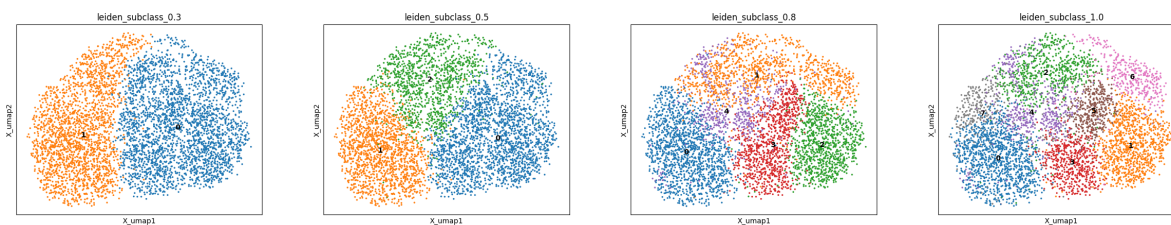
adata = sc.read_h5ad("5_Cluster_h5ad/bs11md8.Seurat-3000_npcs30.ANN.h5ad")
adata.X = adata.layers["log1p"].copy()
bdata = adata[adata.obs['celltype'].isin([
    "4-Bipolar+Amacrine+Muller",
    "6-Bipolar+Amacrine+Muller",
    "7-Bipolar+Amacrine+Muller"])]].copy()
batch_key = "batch"
prefix = "Cluster4-6-7"
umapRun(bdata, batch_key, prefix)

```



```
[0.01725123 0.0291363 0.03864829 0.04771179 0.05597928 0.06381952
0.07122134 0.07819394 0.08479687 0.09092449 0.09683844 0.1027241
0.10844418 0.1139577 0.1194131 0.1246983 0.12989478 0.13502616
0.14010623 0.14513096 0.14998223 0.15474448 0.15944609 0.16403289
0.16852918 0.17290433 0.17720948 0.18142562 0.18559726 0.18970686
0.1936907 0.19762371 0.20152447 0.20534554 0.20914435 0.21287651
0.2165687 0.2201885 0.22380257 0.22734039 0.2308414 0.234275
0.23768672 0.24106832 0.24444595 0.24777016 0.25108278 0.25438476
0.2576722 0.26089525]
```

WARNING: saving figure to file figures/X_umap.Cluster4-6-7.clusters.pdf



```

AnnData object with n_obs × n_vars = 5095 × 23605
  obs: 'CellID', 'batch', 'species', 'n_counts', 'n_genes', 'percent_mito', 'n_genes_by_counts',
  var: 'Gene', 'mt', 'n_cells_by_counts', 'mean_counts', 'log1p_mean_counts', 'pct_dropout',
  uns: 'hvg', 'leiden_0.3', 'leiden_0.3_colors', 'leiden_0.5', 'leiden_0.5_colors', 'leiden_0.5_subclass',
  obsm: 'X_pca', 'X_umap', 'spatial', 'X_subclass_umap'
  layers: 'counts', 'log1p'
  obsp: 'connectivities', 'distances'

```

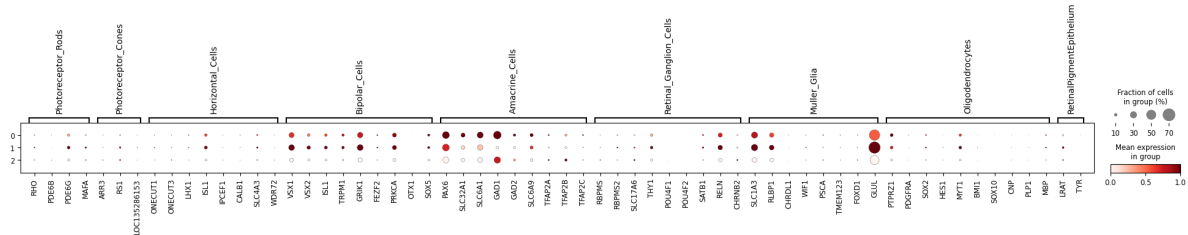
Then, we extract cluster 4, 6 and 7 (mixed with **Amacrine Cells**, **Bipolar Cells** and **Muller Glia**) into a new object `bdata`. To re-cluster them, we run `umapRun`, and we will see this subset splitting into several sub-clusters. Now we can re-assign the identity of each cluster based on the dotplot.

```

retina_marker_genes = {
    "Photoreceptor_Rods": ["RHO", "PDE6B", "PDE6G", "MAFA"],
    "Photoreceptor_Cones": ["ARR3", "RS1", "OPN1SW", "OPN2SW", "OPN1LW",
        ↪ "LOC135286153"],
    "Horizontal_Cells": ["ONECUT1", "ONECUT2", "ONECUT3", "LHX1", "ISL1",
        ↪ "IPCEF1", "OXT", "CALB1", "SLC4A3", "WDR72"],
    "Bipolar_Cells": ["VSX1", "VSX2", "ISL1", "TRPM1", "GRIK1", "FEZF2",
        ↪ "PRKCA", "OTX1", "SOX5", "CABP5"],
    "Amacrine_Cells": ["PAX6", "PROX1", "SLC32A1", "SLC6A1", "GAD1", "GAD2",
        ↪ "SLC6A9", "TFAP2A", "TFAP2B", "TFAP2C"],
    "Retinal_Ganglion_Cells": ["RBPMS", "RBPMS2", "SLC17A6", "THY1",
        ↪ "POU4F1", "POU4F2", "POU4F3", "SATB1", "RELN", "CHRNA2", "SS2"],
    "Muller_Glia": ["SLC1A3", "RLBP1", "CHRD1", "WIF1", "PSCA", "TMEM123",
        ↪ "FOXD1", "GLUL"],
    "Oligodendrocytes": ["OLIG2", "PTPRZ1", "PDGFRA", "SOX2", "HES1", "MYT1",
        ↪ "BMI1", "SOX10", "CNP", "PLP1", "MBP", "BACS1"],
    "RetinalPigmentEpithelium": ["RPE65", "PMEL", "BEST1", "LRAT", "TYR"]
}
marker_in_adata = {cell_type: [gene for gene in genes if gene in
    ↪ bdata.var_names]
                    for cell_type, genes in retina_marker_genes.items()}
sc.pl.dotplot(bdata,
              marker_in_adata,
              groupby='leiden_subclass_0.5',
              standard_scale='var',
              swap_axes=False,
              save='markers_'+prefix+'_res0.5.png')

```

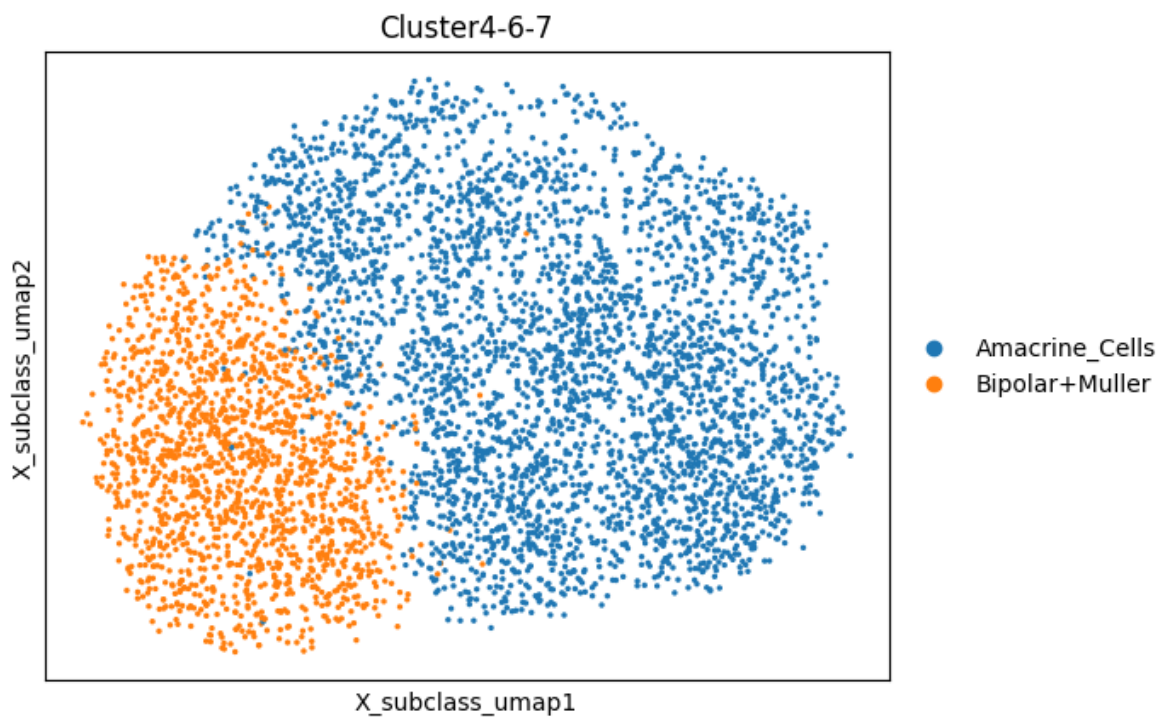
WARNING: saving figure to file figures/dotplot_markers_Cluster4-6-7_res0.5.png



```
Annotation_marker = {
    "0": "Amacrine_Cells", "1": "Bipolar+Muller", "2": "Amacrine_Cells"}
bdata.obs["subclass"] =
    ↪ bdata.obs["leiden_subclass_0.5"].map(Annotation_marker)

sc.pl.embedding(bdata, basis='X_subclass_umap', color='subclass', save=('.') +
    ↪ prefix + '.ANN.png'), title=f"{prefix}")
```

WARNING: saving figure to file figures/X_subclass_umap.Cluster4-6-7.ANN.png



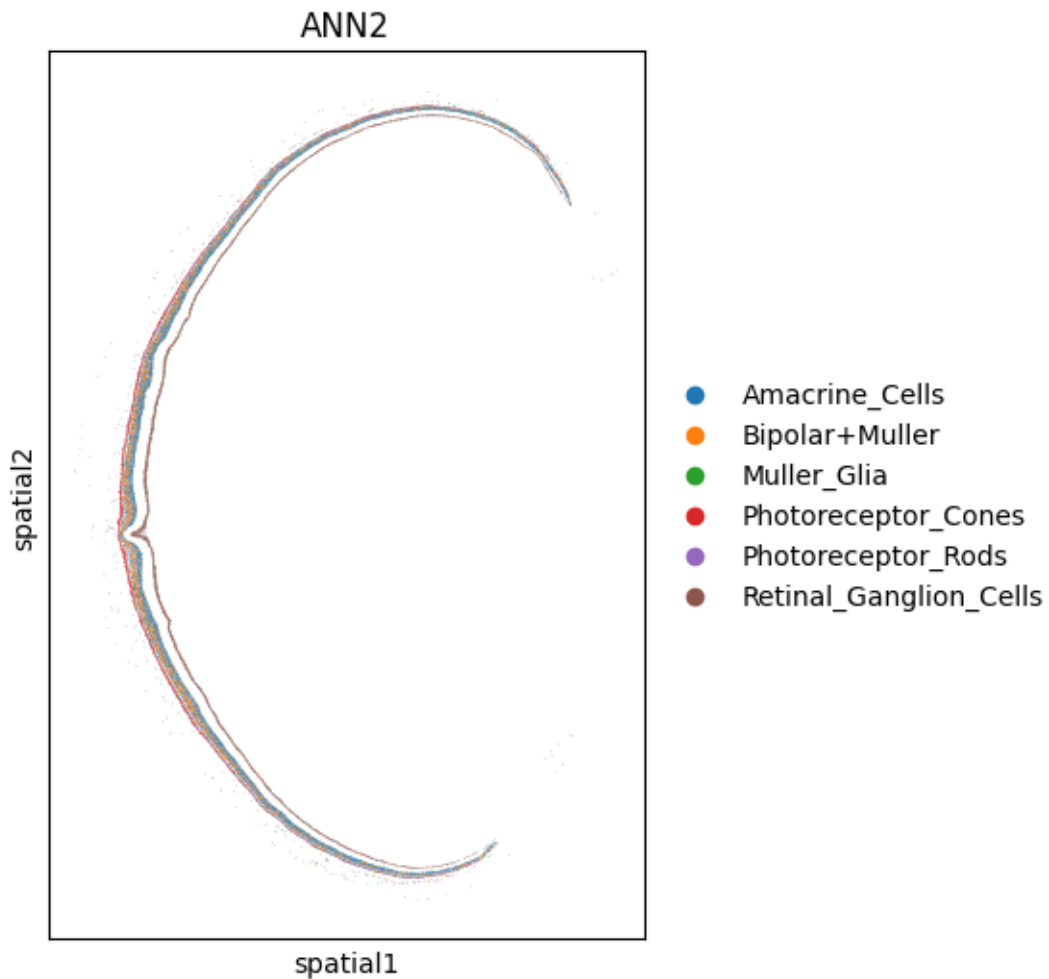
```
aobs = adata.obs
bobs = bdata.obs
```

```

mapped = pd.Series(aobs.index.map(bobs['subclass']), index=aobs.index)
aobs['ANN2'] = mapped.combine_first(aobs['ANN'])
adata.obs = aobs
from matplotlib import rc_context
with rc_context({'figure.figsize': (4, 6)}):
    sc.pl.embedding(adata, basis=f"spatial", color='ANN2',
        ↪ save=('.ANN2.pdf'), size=0.6)

```

WARNING: saving figure to file figures/spatial.ANN2.pdf



```

# Save the updated annotation results
adata.obs['celltype'] = adata.obs['ANN2']

```

```
adata.obs = adata.obs.drop(columns=['ANN', 'ANN2'])
adata.write_h5ad("5_Cluster_h5ad/bs11md8.Seurat-3000_npcs30.ANN2.h5ad")
adata.obs.to_csv("5_Cluster_h5ad/bs11md8.Seurat-3000_npcs30.ANN2.h5ad.obs.txt",
↪ sep="\t", index=True)
```

Finally, plotting the re-cluster groups along with previously annotated ones onto the 2D space, we obtain a **spatial atlas** of sparrow’s retinal tissues, with each colored dot representing a specific type of cell in a certain position.

A Hint:

As we can see, it is usually super demanding to manually annotate cell clusters using marker genes, particularly when dealing with Stereo-seq data (or other high-resolution spatial transcriptomes). There are several potential factors accounting for the sparsity of marker genes, such as low sequencing depth (capture efficiency), cell segmentation quality, species differences, or there might be novel or rare subtypes that have not been defined by any literature.

A more recommended practice is to integrate comprehensive biological insights, like re-calculating **Differentially Expressed Genes (DEGs)** and refer to related database, or leverage cell annotation tools (e.g., RCTD, Robust Cell Type Decomposition) to facilitate identification. Many studies have been focusing on developing annotation algorithms.

5 RCTD: Integration & Prediction

Stereo-seq provides high resolution, but sometimes a single cellbin might lack clear markers. **RCTD**, or **Robust Cell Type Decomposition** can help. RCTD is a supervised computational method designed to estimate cell-type proportions in spatial transcriptomics data by deconvolving the spatial signal – comparing our spatial data against a scRNA-seq reference to predict which cell types are actually present in each spot.

RCTD deals with the spatial transcriptomics data using a Poisson regression model. It is robust because it accounts for platform effects, the systematic technical differences in gene capture between single-cell sequencing (like **DNBelab C Series High-throughput Single-cell Sequencing** and **STOmics Chip scRNA-seq** in our experiment) and spatial capture technologies like Stereo-seq.

Now we can perform RCTD in R environment, using **spacexr** package.

5.1 Code

i. Loading Packages

```
library(spacexr)
library(SeuratObject)
```

Loading required package: sp

ii. Creating RCTD Object

```
ref <- readRDS("PasRetina_StereoCell_ANN.rds")
obj <- readRDS("D06050A3_bs11md8_bin20.rds")
```

To create the RCTD object, we need to provide a reference dataset **ref** and a query dataset **obj**. Here, the reference is a scRNA-seq dataset with cell type annotation (If the annotation is conducted with Python, we have to convert the **.h5ad** file into **.rds** file first). The query dataset is a “raw” seurat object transformed after cell segmentation, before any downstream analysis.

```
ref <- UpdateSeuratObject(ref)
Idents(ref) <- "celltype"
counts <- ref@assays[["RNA"]@counts
cluster <- as.factor(ref$celltype)
names(cluster) <- colnames(ref)

# Calculate nUMI & nGene in each cell
nUMI <- Matrix::colSums(ref@assays[["RNA"]@counts)
nGene <- Matrix::colSums(ref@assays[["RNA"]@counts > 0)
ref$nCount_RNA <- nUMI
ref$nFeature_RNA <- nGene
names(nUMI) <- colnames(ref)
reference <- Reference(counts, cluster, nUMI)
```

Validating object structure

Updating object slots

Ensuring keys are in the proper structure

Ensuring feature names don't have underscores or pipes

Object representation is consistent with the most current Seurat version

Warning message in `Reference(counts, cluster, nUMI)`:

"Reference: number of cells per cell type is 27836, larger than maximum allowable of 10000. I

The first step of RCTD is reference construction. Three metrics are required: `counts`, `cluster` and `nUMI`, which are all contained in the `ref` dataset. Please extract the raw `counts` only, since RCTD uses Poisson model that requires integer counts.

```
counts <- obj@assays$Spatial@counts
coords <- GetTissueCoordinates(obj)
colnames(coords) <- c("x", "y")
query <- SpatialRNA(coords, counts, colSums(counts))
```

Loading required package: Seurat

Next, we create the query object incorporating coordinates, count matrix and total UMIs of each pixel.

```
RCTD <- create.RCTD(query, reference, max_cores = 8)
```

Begin: `process_cell_type_info`

`process_cell_type_info`: number of cells in reference: 49650

`process_cell_type_info`: number of genes in reference: 27811

Amacrine_Cells	Bipolar_Cells	Horizontal_Cells
10000	10000	2953
Muller_Glia	Oligodendrocytes	Photoreceptor_Cones
6284	2080	9369
Photoreceptor_Rods	RetinalPigmentEpithelium	Retinal_Ganglion_Cells
1129	2197	5638

Warning message in `asMethod(object)`:

"sparse->dense coercion: allocating vector of size 2.1 GiB"

Warning message in `asMethod(object)`:

```
"sparse->dense coercion: allocating vector of size 2.1 GiB"
Warning message in asMethod(object):
"sparse->dense coercion: allocating vector of size 1.3 GiB"
Warning message in asMethod(object):
"sparse->dense coercion: allocating vector of size 1.9 GiB"
Warning message in asMethod(object):
"sparse->dense coercion: allocating vector of size 1.2 GiB"
End: process_cell_type_info
```

```
create.RCTD: getting regression differentially expressed genes:
```

```
get_de_genes: Amacrine_Cells found DE genes: 61
```

```
get_de_genes: Bipolar_Cells found DE genes: 71
```

```
get_de_genes: Horizontal_Cells found DE genes: 156
```

```
get_de_genes: Muller_Glia found DE genes: 183
```

```
get_de_genes: Oligodendrocytes found DE genes: 164
```

```
get_de_genes: Photoreceptor_Cones found DE genes: 159
```

```
get_de_genes: Photoreceptor_Rods found DE genes: 192
```

```
get_de_genes: RetinalPigmentEpithelium found DE genes: 115
```

```
get_de_genes: Retinal_Ganglion_Cells found DE genes: 118
```

```
get_de_genes: total DE genes: 1073
```

```
create.RCTD: getting platform effect normalization differentially expressed genes:
```

```
get_de_genes: Amacrine_Cells found DE genes: 195
```

```
get_de_genes: Bipolar_Cells found DE genes: 193
```

```
get_de_genes: Horizontal_Cells found DE genes: 311
```

```
get_de_genes: Muller_Glia found DE genes: 374
```

```
get_de_genes: Oligodendrocytes found DE genes: 320
```

```

get_de_genes: Photoreceptor_Cones found DE genes: 365
get_de_genes: Photoreceptor_Rods found DE genes: 433
get_de_genes: RetinalPigmentEpithelium found DE genes: 310
get_de_genes: Retinal_Ganglion_Cells found DE genes: 261
get_de_genes: total DE genes: 2086

```

Having constructed reference and query objects, we are going to create RCTD object. `get_de_genes` in the Cell output indicates that before deconvolution, RCTD identifies differentially expressed genes between cell types in our scRNA-seq dataset. They are used to distinguish one cell type from another.

iii. Run RCTD

```
RCTD <- run.RCTD(RCTD, doublet_mode = "doublet")
```

```

fitBulk: decomposing bulk

chooseSigma: using initial Q_mat with sigma = 1

Likelihood value: 404155.193706775

Sigma value: 1.16

Likelihood value: 401403.750435577

Sigma value: 1.32

Likelihood value: 400051.535027668

Sigma value: 1.42

Likelihood value: 399824.476889005

Sigma value: 1.42

[1] "gather_results: finished 1000"
[1] "gather_results: finished 2000"
[1] "gather_results: finished 3000"

```

```

[1] "gather_results: finished 4000"
[1] "gather_results: finished 5000"
[1] "gather_results: finished 6000"
[1] "gather_results: finished 7000"
[1] "gather_results: finished 8000"
[1] "gather_results: finished 9000"
[1] "gather_results: finished 10000"
[1] "gather_results: finished 11000"
[1] "gather_results: finished 12000"
[1] "gather_results: finished 13000"
[1] "gather_results: finished 14000"
[1] "gather_results: finished 15000"
[1] "gather_results: finished 16000"
[1] "gather_results: finished 17000"
[1] "gather_results: finished 18000"
[1] "gather_results: finished 19000"
[1] "gather_results: finished 20000"
[1] "gather_results: finished 21000"
[1] "gather_results: finished 22000"

```

`run.RCTD` is the core step and the most time-consuming step. There are two modes we can choose from – `doublet` and `full`. Doublet mode assumes each pixel contains only one or two cell types. This is ideal for high-resolution platforms like Stereo-seq, where a bin20 is close to single-cell size. Full mode allows for multiple cell types in a single pixel, so it is more suitable for lower-resolution platforms.

Now let us explain the outputs:

1. **fitBulk: decomposing bulk:** RCTD estimates the platform effect, checking which genes are systematically captured differently between scRNA-seq and Stereo-seq.
2. **chooseSigma:** This step estimates the overdispersion, which represents the noise or biological variation that is not explained by cell type proportions.
3. **Likelihood value & Sigma value:** The algorithm is iteratively searching for the best σ . The σ converges (1.42 in this example) as the likelihood value stabilizes, meaning the model fits the data better.
4. **gather_results:** After the parameters are confirmed, RCTD starts to examine each spot. The process ends with `finished 22000`, suggesting that RCTD has successfully deconvolved around 22,000 spots in our dataset.

There are four types in RCTD spot classification:

1. **Singlet:** It is confident that the gene expression in this spot is best explained by one single cell type. The likelihood of this spot being a single type is significantly higher than being a mixture. Therefore, this is the purest result.

2. **Doublet**: The spot is best explained as a mixture of two different cell types. The names of both the `first_type` and `second_type`, along with their respective weights/proportions, will be provided.
3. **Doublet_uncertain**: The spot is likely a mixture, but it cannot confidently distinguish which specific types they are, or the likelihood ratio between the best-fit doublet and other possibilities is not strong enough.
4. **reject**: The spot is discarded as it does not meet the quality threshold. This could be due to low UMI counts or the expression pattern does not match any combination of cell types in our reference.

As a result, a high proportion of **singlet** spots is a positive sign, meaning our Stereo-seq bin captures pure cell signals.

iv. Visualization

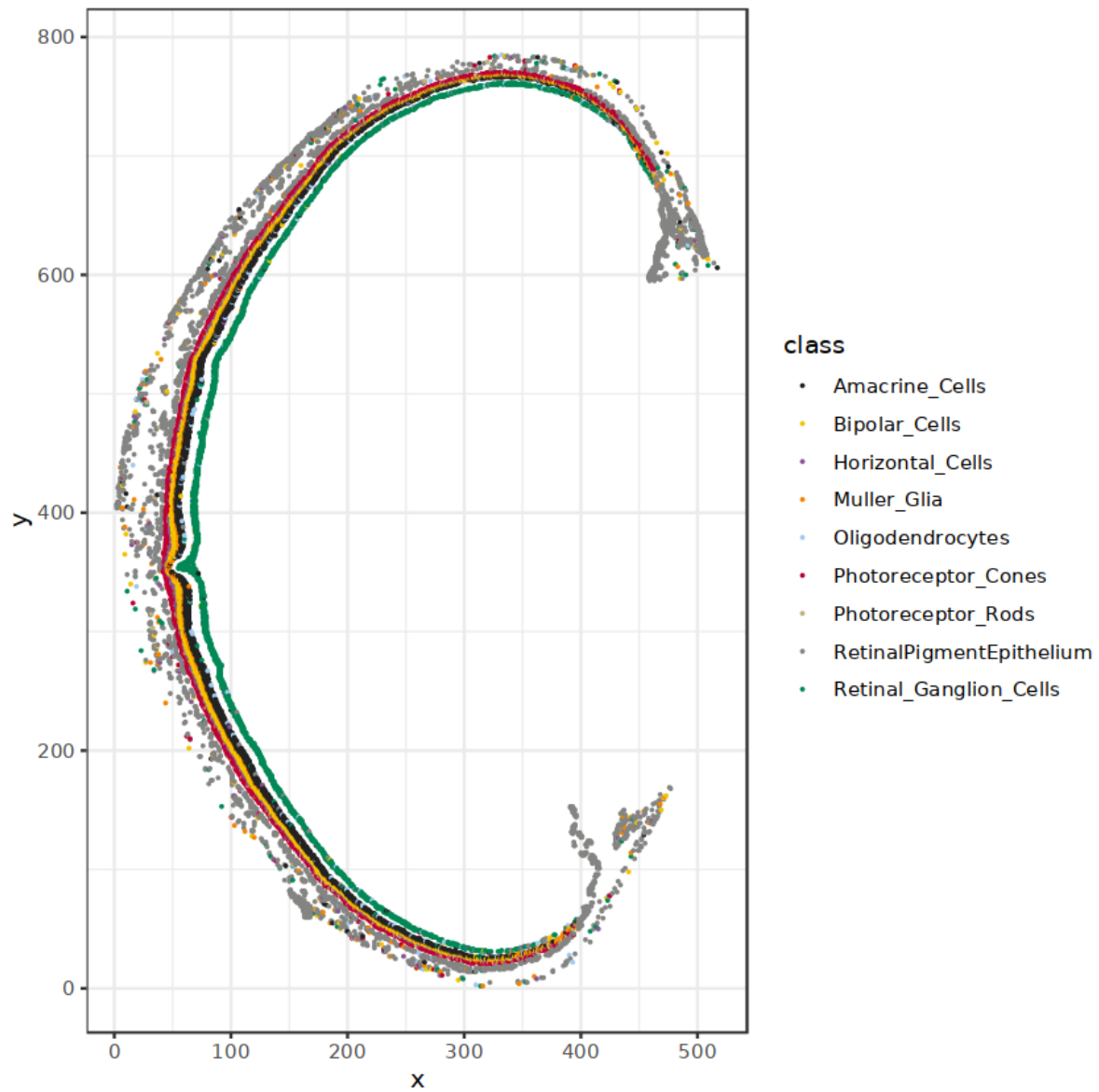
```
results <- RCTD@results
norm_weights = normalize_weights(results$weights)
cell_type_names <- RCTD@cell_type_info$info[[2]]
spatialRNA <- RCTD@spatialRNA
resultsdir <- 'RCTD_Plots'
dir.create(resultsdir)
```

```
plot_all_celltypes_size <- function(results_df, coords, cell_type_names,
  ↪ resultsdir) {
  barcodes = rownames(results_df[results_df$spot_class != "reject" &
  ↪ results_df$first_type %in% cell_type_names,])
  my_table = coords[barcodes,]
  my_table$class = results_df[barcodes,]$first_type
  n_levels = length(levels(my_table$class))
  my_pal = pals::kelly(n_levels+1)[2:(n_levels+1)]
  pres = unique(as.integer(my_table$class))
  pres = pres[order(pres)]
  if(n_levels > 21)
    my_pal = pals::polychrome(n_levels)
  if(n_levels > 36)
    stop("Plotting currently supports at most 36 cell types as colors")
  plot <- ggplot2::ggplot(my_table, ggplot2::aes(x=x, y=y)) +
  ↪ ggplot2::geom_point(ggplot2::aes(color=class),size = 0.02, shape=19) +
    ggplot2::scale_color_manual(values = my_pal[pres])+
  ↪ ggplot2::scale_shape_identity() + ggplot2::theme_bw()
  pdf(file.path(resultsdir,"all_cell_types.pdf"))
  invisible(print(plot))
  dev.off()
```

```

    return(plot)
}
plot_all_celltypes_size(results$results_df, spatialRNA@coords,
  ↪ cell_type_names, resultsdir)

```



This is the final step, spatial visualization.

`barcodes = ...` selects only reliable spots and rejects spots with low quality.

From the colorful spatial atlas, we can see clearly different cell types distributed in certain layers, compliant with authentic biological structures.

6 Summary

Congratulations!

Now we have taken high-resolution Stereo-seq spatial data, and transformed abstract raw counts into a beautiful, anatomical coordinates map of the avian retina! The next steps of data mining include: detecting fine-grained marker genes and cell subtypes, deducing cell-cell communication and so on.

6.1 Keep Exploring...

6.2 Keep Discovering...

Copyright Notice: All articles on this paper are licensed under [CC BY-NC-SA 4.0](#) unless otherwise stated.