```
# Single-Cell RNA-seq Analysis Report

**Dataset:** AdamsonWeissman2016_GSM2406677_10X005 (`/Users/azratuncay/Downloads/A
**Date of Analysis:** May 8, 2025 (Placeholder)
**Analysis Tool:** Scanpy (via Python script `data_report.py`)

## 1. Introduction

This report details the analysis of a single-cell RNA sequencing (scRNA-seq) datas

## 2. Setup and Data Loading

### 2.1. Library Import and Settings

The analysis began by importing necessary Python libraries, including `scanpy`, `p

Terminal Output:
```

Libraries imported and Scanpy settings configured.
scanpy==... anndata==... umap==... numpy==... scipy==... pandas==... scikit-learn==... statsmodels==... python-igraph==... leidenalg==... (Versions will be listed by print_header())

```
### 2.2. Data Loading

The dataset was loaded from the H5AD file: `/Users/azratuncay/Downloads/AdamsonWei

Terminal Output:
```

Successfully loaded: /Users/azratuncay/Downloads/AdamsonWeissman2016_GSM2406677_10X005.h5ad

AnnData object summary:
AnnData object with n_obs × n_vars = 15006 × 32738
obs: 'perturbation', 'read count', 'UMI count', 'tissue_type', 'cell_line', 'cancer', 'disease', 'perturbation_type', 'celltype', 'organism', 'ncounts', 'ngenes', 'percent_mito', 'percent_ribo', 'nperts'
var: 'ensembl_id', 'ncounts', 'ncells'

```
The dataset initially contained 15,006 cells and 32,738 genes/features. Cell metad

### 2.3. Initial Data Matrix Inspection

The data matrix `adata.X` had a shape of (15006, 32738) with a `float32` data type

Terminal Output (snippet):
```

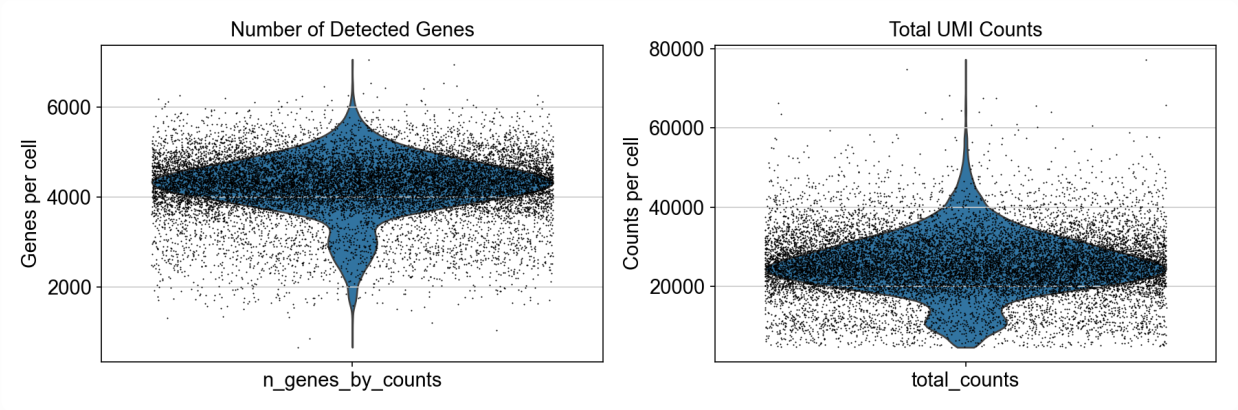
Data matrix values appear to be non-negative integers (likely raw counts).

```
## 3. Quality Control (QC)

A copy of the original data was stored in `adata.raw`. Standard QC metrics (`n_ger

Terminal Output:
```

Stored a copy of the original AnnData object in adata.raw.
Calculated QC metrics. Added to adata.obs:
... (head of adata.obs showing new columns)

```
### 3.1. Visualization of QC Metrics

The distributions of `n_genes_by_counts` (number of detected genes per cell) and `
```

Figure_1.jpg (Violin Plots): These plots show the distribution density of detected genes per cell and total UMI counts per cell. They are used to identify the main population of cells and potential outliers (e.g., cells with very few genes or counts, which might be empty droplets or damaged cells, or cells with extremely high counts, which might be doublets).

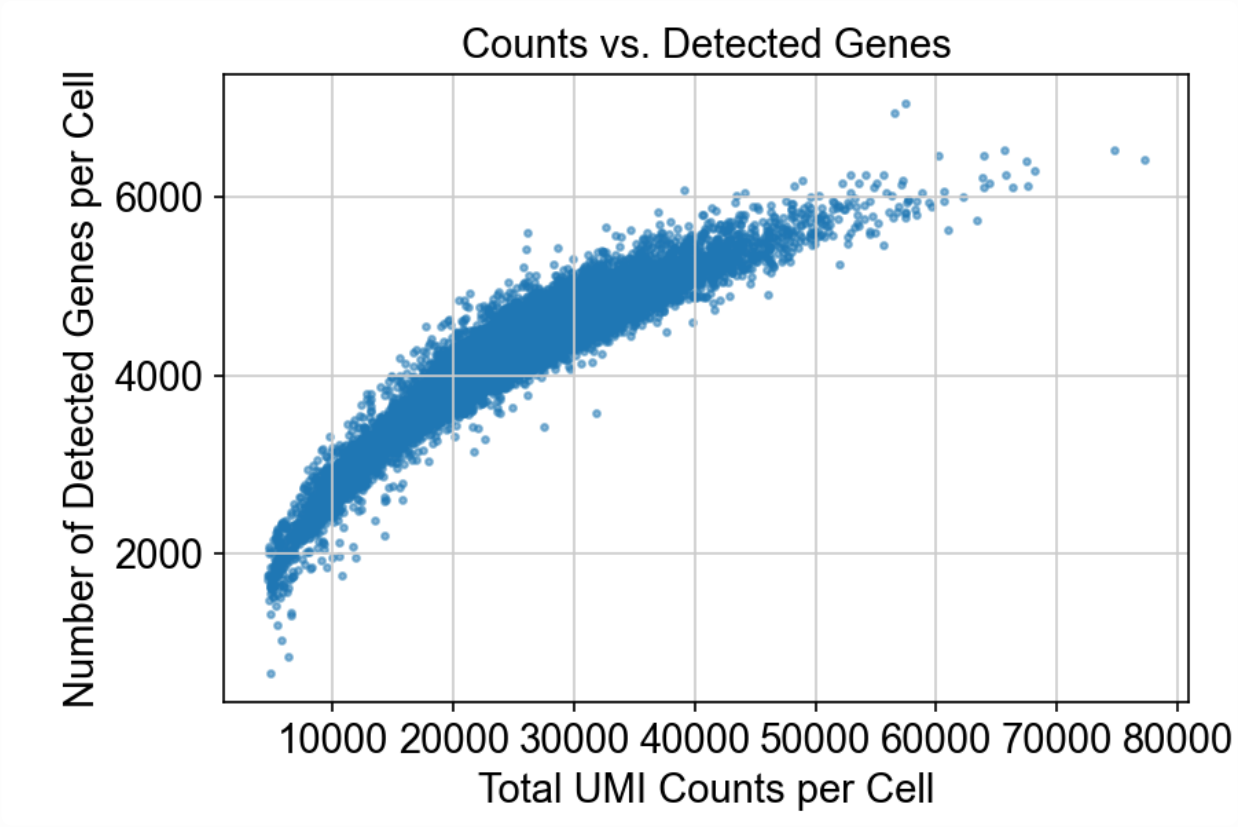


These violin plots show the distribution of two key quality control metrics across all cells: the number of detected genes per cell and the total UMI counts per cell.

Number of Detected Genes: The left plot shows that the majority of cells have between approximately 2000 and 6000 genes detected, with the peak of the distribution around 4000-5000 genes. The violin shape indicates a dense cluster of cells within this range. There are tails extending to both lower and higher numbers of detected genes, suggesting a small population of cells with fewer than 2000 genes and some with more than 6000. The population with fewer detected genes could be candidates for filtering, as they may represent low-quality cells or debris.
Total UMI Counts: The right plot shows that most cells have total UMI counts falling

between approximately 20,000 and 60,000, with the highest density between 30,000 and 50,000 counts. Similar to the gene detection plot, there are tails indicating cells with lower and higher total counts. Cells with very low total counts (e.g., below 20,000) might also be indicative of low-quality captures or cellular debris and could be considered for filtering.

Figure_2.png (Scatter Plot): This plot shows the relationship between the total UMI counts and the number of detected genes for each cell. A positive correlation is expected, where cells with more UMIs generally have more genes detected. Outliers deviating from this trend can be identified.



This scatter plot shows the relationship between the total UMI counts per cell and the number of detected genes per cell. There is a strong positive correlation between these two metrics, which is expected; cells with more total transcripts (higher UMI counts) generally have a greater variety of genes detected. The relationship appears somewhat non-linear, potentially plateauing at higher counts/genes, suggesting that sequencing depth reaches a saturation point for gene detection. There are no obvious separate populations of cells with unusual ratios of counts to genes, which would indicate potential quality issues.

3.2. Filtering

Based on example thresholds (which should ideally be adjusted based on the QC plots):

- Minimum genes per cell: 200
- Minimum UMI counts per cell: 500
- Minimum cells per gene: 3

Terminal Output:

```
Number of cells before filtering: 15006
Number of genes/features before filtering: 32738
Number of cells after filtering by min_genes_per_cell (200): 15006
Number of cells after filtering by min_counts_per_cell (500): 15006
filtered out 15131 genes that are detected in less than 3 cells
Number of genes/features after filtering by min_cells_per_gene (3): 17607
Final cells: 15006, Final genes/features: 17607
```

The cell filtering steps with the chosen example thresholds (200 genes/cell, 500 counts/cell) did not remove any cells, indicating that all cells passed these initial generous cutoffs. Gene filtering removed 15,131 genes detected in fewer than 3 cells. After filtering, the dataset contained 15,006 cells and 17,607 genes/features.

(Note for your report: Justify why these specific example thresholds were used or, if you adjusted them based on your plots, explain your rationale.)

4. Preprocessing

4.1. Normalization and Log-Transformation

The filtered data was normalized by scaling counts per cell to a target sum of 10,000 (library size normalization) using `scanpy.pp.normalize_total()`. Subsequently, a log1p transformation ($\log(X+1)$) was applied using `scanpy.pp.log1p()` to stabilize variance and make the data distributions more symmetric.

Terminal Output:

```
Applied library size normalization (target_sum=1e4) and log1p transformation.
adata.X sample after normalization & log-transformation:
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

The sample of `adata.X` shows zeros, which is common for sparse scRNA-seq data even after transformation, as many gene-cell combinations will still have zero counts.

4.2. Highly Variable Gene (HVG) Selection

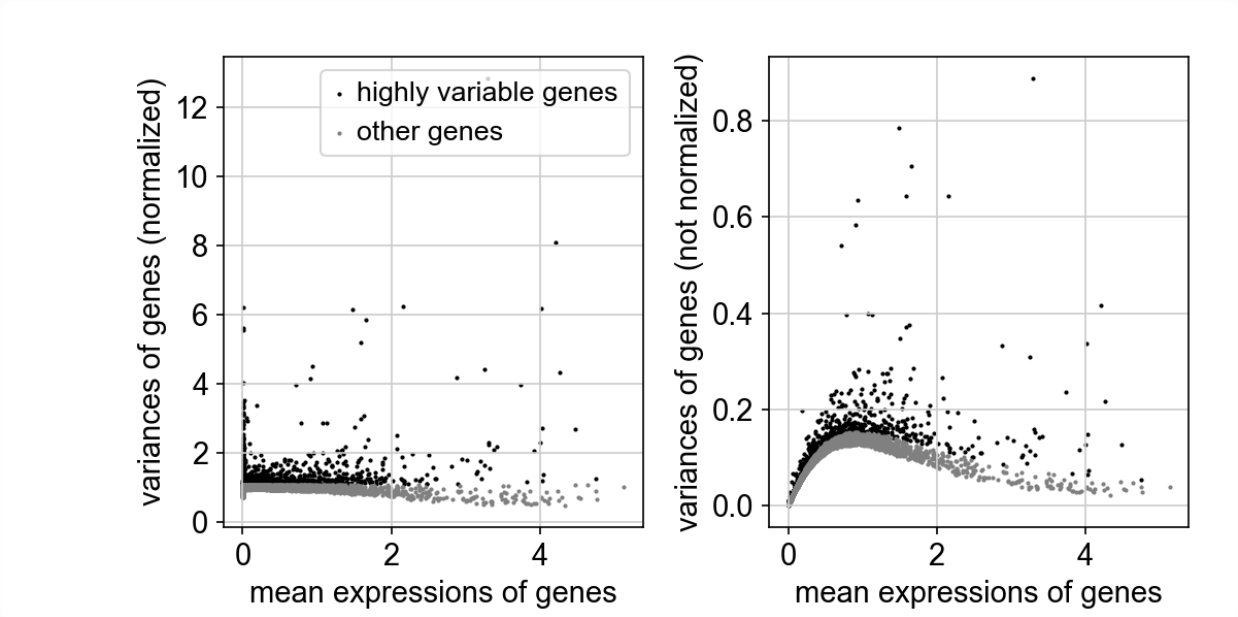
Highly variable genes were identified using the `seurat_v3` method, selecting the top 2000 HVGs.

Terminal Output:

```
/Users/azratuncay/Single_cell_h5ad_Data_analysis_HW/.venv/lib/python3.13/site-pack
warnings.warn(
--> added
    'highly_variable', boolean vector (adata.var)
    'highly_variable_rank', float vector (adata.var)
    'means', float vector (adata.var)
    'variances', float vector (adata.var)
    'variances_norm', float vector (adata.var)

Identified highly variable genes. `adata.var['highly_variable']` shows which are s
Number of highly variable genes: 2000
```

A `UserWarning` was issued because the `seurat_v3` flavor expects raw count data, but the input `adata.X` had already been normalized and log-transformed. Despite this, the function proceeded and identified 2000 HVGs. This is a common practice in Scanpy workflows, though ideally, for `seurat_v3`, raw counts would be used for variance modeling. The selection added several columns to `adata.var`, including `highly_variable` (a boolean mask) and `means` / `variances` (calculated on the log-normalized data).

Figure_3.png (Highly Variable Genes Plot): This plot typically shows gene variances (often normalized) against mean expressions, highlighting the selected HVGs.



(These plots visualize the relationship between the mean expression of genes and their variance. The left plot shows the variance of genes after normalization, while the right plot shows the variance before normalization. Highly variable genes (marked in black) are those that exhibit particularly high variance relative to their mean expression across the cell population, distinguishing them from other genes (marked in grey).
Normalized Variance: In the left plot, highly variable genes are primarily located above the bulk of the other genes, particularly at lower and moderate mean expression levels, indicating that their high variability is not simply a function of higher expression.
Unnormalized Variance: The right plot shows a strong dependence of variance on the mean

expression, where genes with higher mean expression tend to have higher variance. The highly variable genes are the outliers that show exceptionally high variance for their given mean expression level.
The plot effectively highlights genes that contribute most to the biological variability in the dataset, and these genes are often prioritized for downstream analysis like PCA.

4.3. Scaling

The data (all 17,607 genes, as HVG subsetting was not performed on `adata.X`) was scaled to unit variance and zero mean, with values clipped at a maximum of 10 using `scanpy.pp.scale()`.

Terminal Output:

```
... as `zero_center=True`, sparse input is densified and may lead to large memory

Scaled the data to unit variance and zero mean (adata.X). Clipped at max_value=10.
adata.X sample after scaling:
[[-0.07117818 -0.02898227 -0.01411406 -0.012744 -0.04029365]
 [-0.07117818 -0.02898227 -0.01411406 -0.012744 -0.04029365]
 [-0.07117818 -0.02898227 -0.01411406 -0.012744 -0.04029365]
 [-0.07117818 -0.02898227 -0.01411406 -0.012744 -0.04029365]
 [-0.07117818 -0.02898227 -0.01411406 -0.012744 -0.04029365]]
```

A warning regarding potential memory consumption due to densification of sparse input during scaling was noted. The sample output shows scaled, centered (negative and positive float) values.

5. Dimensionality Reduction

5.1. Principal Component Analysis (PCA)

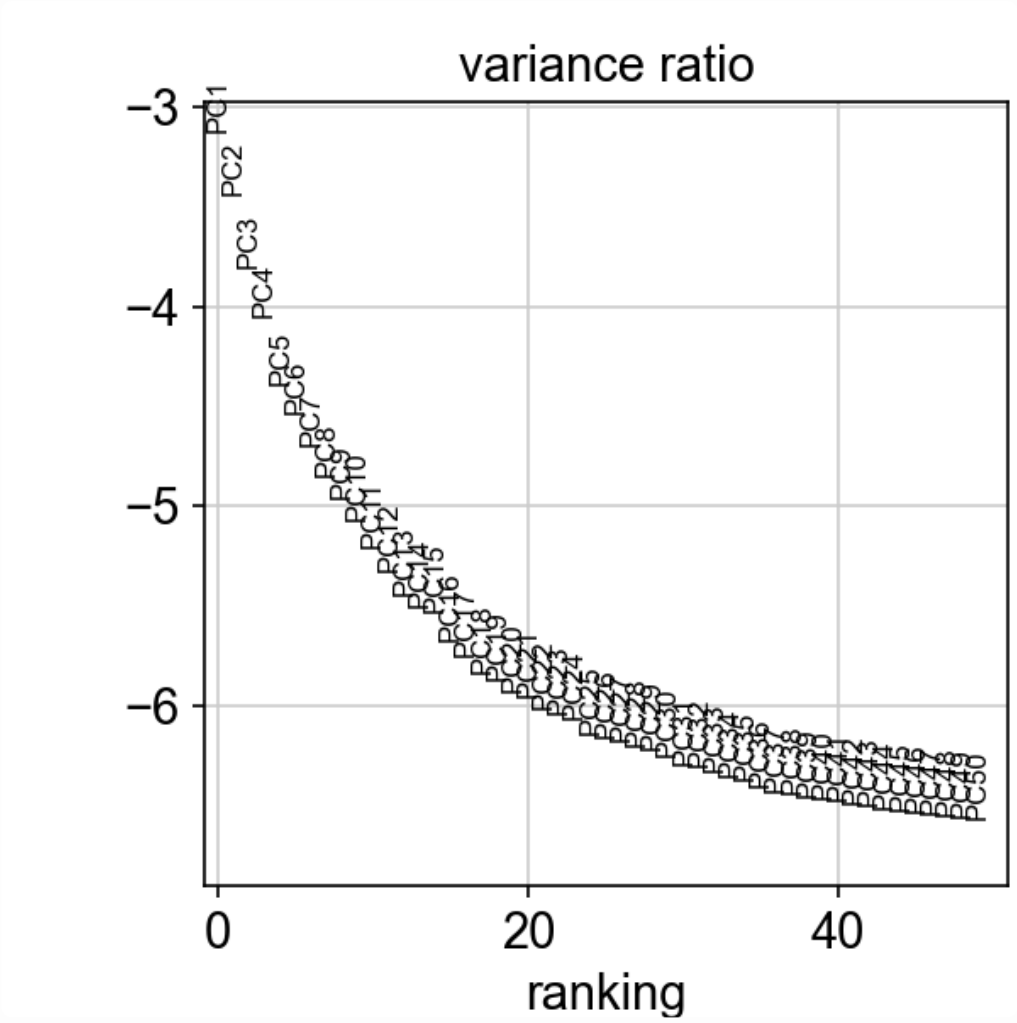
PCA was performed on the scaled data to reduce its dimensionality. The number of principal components (`n_comps_pca`) was calculated as 50. (min_dim = min(15006 cells, 17607 genes) = 15006. n_comps_pca = min(15006 - 2, 50) = 50).

Terminal Output:

```
Current adata dimensions for PCA: n_obs=15006, n_vars=17607
Smallest dimension (min(n_obs, n_vars)): 15006
Calculated n_comps_pca: 50
Attempting PCA with n_comps=50, svd_solver='arpack'.
computing PCA
    with n_comps=50
finished (0:00:01)

Performed PCA.
```

PCA completed successfully using 50 components with the 'arpack' solver.

Figure_4.png (PCA Variance Ratio Plot): This "elbow plot" shows the variance explained by each principal component. It is used to determine how many PCs capture most of the biological signal and should be used for downstream steps like UMAP and clustering.



This plot displays the amount of variance explained by each principal component (PC) in a dimensionality reduction analysis (PCA). The PCs are ranked on the x-axis, and the logarithm of the variance ratio (presumably the ratio of variance explained by each PC to the total variance) is on the y-axis. To determine the number of PCs to retain for further analysis, an "elbow point" is typically sought where the rate of decrease in explained variance slows down considerably.
In this plot, a clear elbow appears to be present around PC 15. Before this point, there is a relatively steep drop in the variance explained by each successive PC. After PC 15, the curve flattens out, indicating that subsequent PCs explain a much smaller proportion of the remaining variance. Therefore, selecting n_pcs_param = 15 for the next step is justified as it captures most of the significant biological variation in the data while reducing dimensionality and noise.

5.2. Neighborhood Graph Construction

A neighborhood graph was computed using `scanpy.pp.neighbors()`. This step utilized the top 15 principal components (a common choice, but should be informed by the PCA elbow plot) and considered 15 neighbors for each cell.

Terminal Output:

```
Using n_neighbors=15 and n_pcs=15 for neighborhood graph.
computing neighbors
    using 'X_pca' with n_pcs = 15
finished: added to `.uns['neighbors']`
    `.obsp['distances']`, distances for each pair of neighbors
    `.obsp['connectivities']`, weighted adjacency matrix (0:00:12)
Computed neighborhood graph.
```

5.3. UMAP Embedding

Uniform Manifold Approximation and Projection (UMAP) was computed for 2D visualization of the cell data, based on the neighborhood graph.

Terminal Output:

```
computing UMAP
finished: added
    'X_umap', UMAP coordinates (adata.obsm)
    'umap', UMAP parameters (adata.uns) (0:00:06)
Computed UMAP embedding.
```

The UMAP coordinates were added to `adata.obsm['X_umap']`.

6. Clustering

6.1. Leiden Clustering

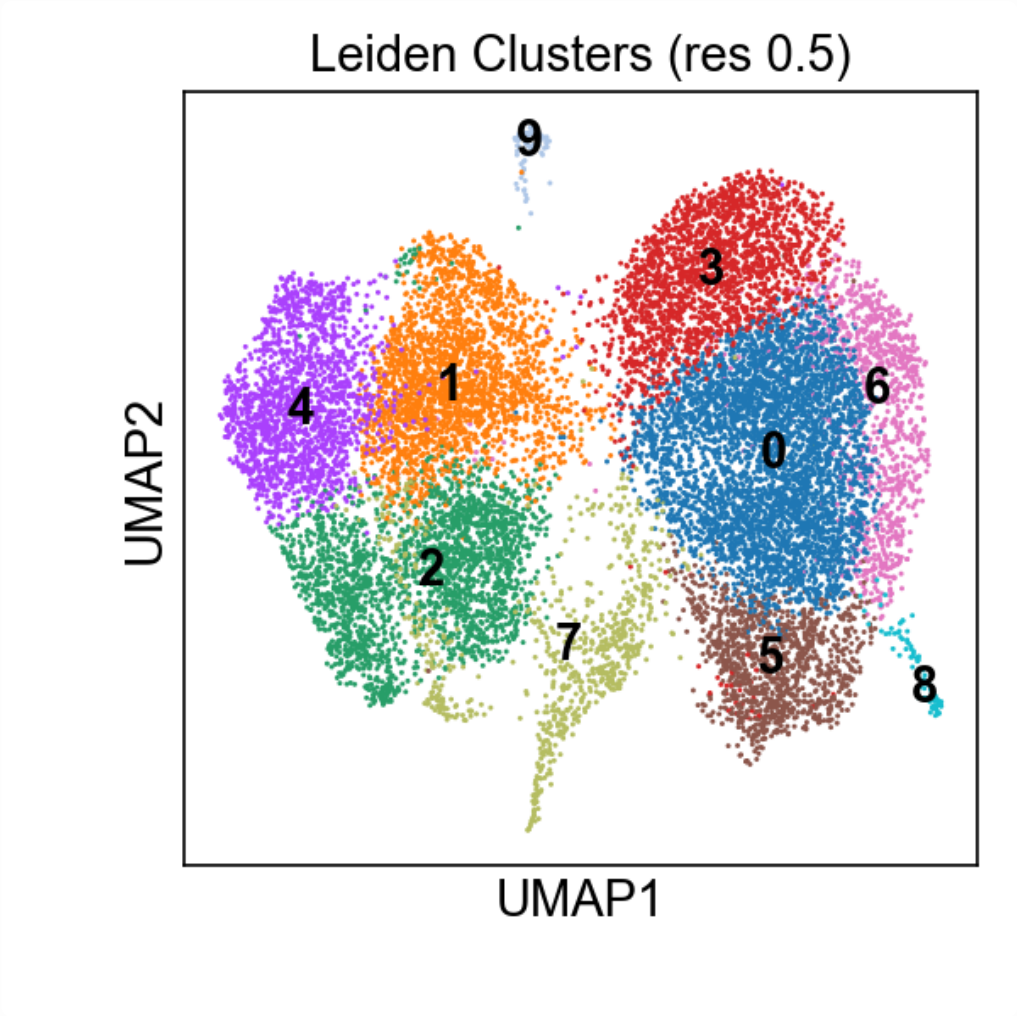
Cells were clustered using the Leiden algorithm (`scanpy.tl.leiden()`) on the computed neighborhood graph. A resolution parameter of 0.5 was used.

Terminal Output:

```
/Users/azratuncay/Single_cell_h5ad_Data_analysis_HW/data_report.py:434: FutureWarn

To achieve the future defaults please pass: flavor="igraph" and n_iterations=2. di
scanpy.tl.leiden(adata, resolution=resolution_param, key_added=f'leiden_res{resolu
running Leiden clustering
finished: found 10 clusters and added
    'leiden_res0.5', the cluster labels (adata.obs, categorical) (0:00:07)
Performed Leiden clustering with resolution 0.5. Clusters stored in 'adata.obs["le
```

A `FutureWarning` indicated a potential change in the default backend for Leiden in future Scanpy versions. The clustering algorithm identified 10 distinct clusters, and the results were stored in `adata.obs['leiden_res0.5']`.

UMAP.png (UMAP by Leiden Clusters): This plot visualizes the 10 identified cell clusters on the UMAP embedding.



(This UMAP plot visualizes the cellular heterogeneity in the dataset by embedding cells in a two-dimensional space based on their principal components, with cells colored and labeled according to their assigned Leiden clusters at a resolution of 0.5.
Cluster Separation: Overall, the clusters appear reasonably well-separated in the UMAP space. Distinct groupings of cells corresponding to different cluster labels (0 through 9) are discernible. Some clusters, like 0, 3, and 4, form large, relatively cohesive masses. Other clusters, such as 1 and 2, are also quite distinct. Clusters 5, 6, 7, 8, and 9 are smaller and appear somewhat more dispersed or connected to larger clusters, though still largely form their own groups. Cluster 9 is particularly small and somewhat isolated.
Spatial Arrangements: There are interesting spatial arrangements that suggest relationships between clusters. For example, clusters 0, 3, and 6 are located in close proximity to each other, hinting at potential relatedness or a continuum between these cell populations. Clusters 1, 2, and 4 are also positioned near each other, suggesting they might represent

related cell types or states. Cluster 7 is located between the larger groupings of clusters 0, 3, 6 and 1, 2, 4, potentially representing an intermediate or transitional population. Cluster 5 is somewhat separate but relatively close to clusters 0 and 7. Cluster 8 is a small, distinct cluster located somewhat apart from the main cellular masses. These spatial relationships in the UMAP can often reflect biological similarities or developmental trajectories between cell populations.

7. Visualization of Gene Expression on UMAP (Attempted)

The script attempted to visualize the expression of example genes on the UMAP. However, this step encountered an issue:

Terminal Output:

```
Error: 'adata' not found, or UMAP/clustering not performed. Skipping gene expressi
UMAP embedding available: False
Cluster key 'leiden_res0.5' available: True
```

The script printed an error message indicating that it was skipping the gene expression visualization. The debug output stated `UMAP embedding available: False`, which means the condition `'umap' in adata.obsm` (as printed by the script's debug line) was false. However, the main conditional for plotting was `if (adata is not None and 'X_umap' in adata.obsm and f'leiden_res{resolution_param}' in adata.obs):`. The UMAP computation log clearly stated that `'X_umap'` was added to `adata.obsm`. The fact that the `else` branch of this main `if` was executed implies that the condition `'X_umap' in adata.obsm` evaluated to `False` when this section of the script was reached. This prevented the plotting of example genes. The cluster key `'leiden_res0.5'` was correctly found.
This discrepancy (UMAP reported as computed and added to `adata.obsm['X_umap']`, but then not found by the conditional for plotting) requires further investigation. It could be a subtle issue in the script logic checking for `'X_umap'` at that specific point, or an unexpected modification to the `adata` object.

Despite this, the script indicates the next steps for systematic marker gene identification:

Terminal Output:

```
To find marker genes systematically, you would typically run:
sc.tl.rank_genes_groups(adata, groupby='leiden_res0.5', method='wilcoxon')
sc.pl.rank_genes_groups(adata, n_genes=10, sharey=False, show=True)
```

8. Conclusion and Next Steps

This analysis successfully processed the AdamsonWeissman2016_GSM2406677_10X005.h5ad single-cell RNA-seq dataset through a standard Scanpy workflow. The data was loaded, quality controlled (resulting in 15,006 cells and 17,607 genes), normalized, and key highly variable genes were identified. Dimensionality reduction via PCA (50 components) and UMAP was performed, followed by Leiden clustering which identified 10 distinct cell clusters (at resolution 0.5).

An issue was encountered in the final step of plotting example gene expressions on the UMAP, where the UMAP coordinates (`adata.obsm['X_umap']`) were unexpectedly not detected by the plotting conditional, despite earlier logs indicating their computation.

Key Findings:

- The dataset comprises 15,006 cells and 17,607 genes after initial QC.
- PCA captured significant variance within 50 components.
- Leiden clustering at resolution 0.5 resolved 10 distinct cell clusters, visualized on a UMAP plot.

Next Steps:

1. Investigate and resolve the issue preventing visualization of gene expression on the UMAP plot.
2. Perform systematic marker gene identification for the 10 Leiden clusters using `sc.tl.rank_genes_groups()` to characterize the biological identity of each cluster.
3. Visualize top marker genes for each cluster.
4. Further biological interpretation of the identified clusters based on marker genes and available cell metadata (e.g., `perturbation`, `celltype`).
5. It could be considered to explore different Leiden resolution parameters to assess the granularity of clustering.
6. Evaluate the impact of the `seurat_v3` HVG warning and potentially rerun HVG selection on raw counts if necessary.