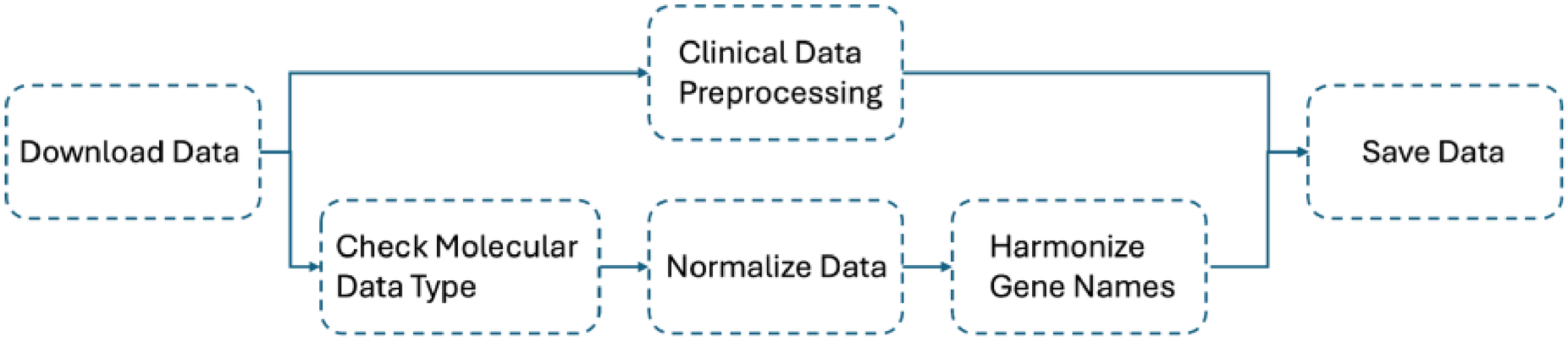
EPIGENE LABS

COMPUTATIONAL BIOLOGY INTERNSHIP-CHALLENGE

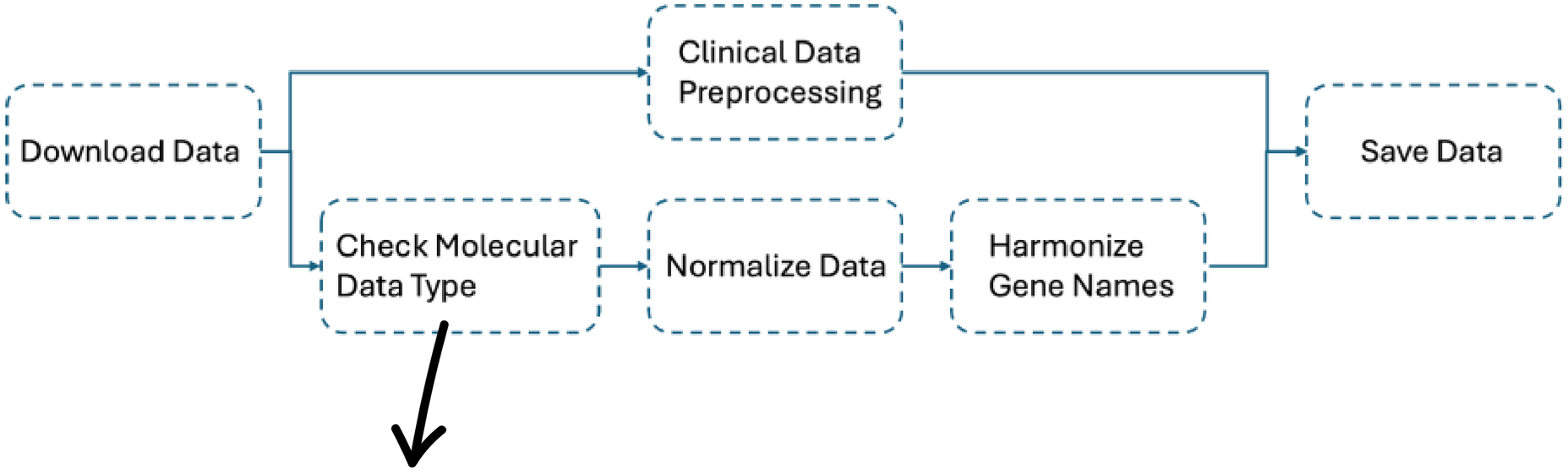
JHONATAN FELIX

Task 2 : Improve the RNAseq data integration pipeline



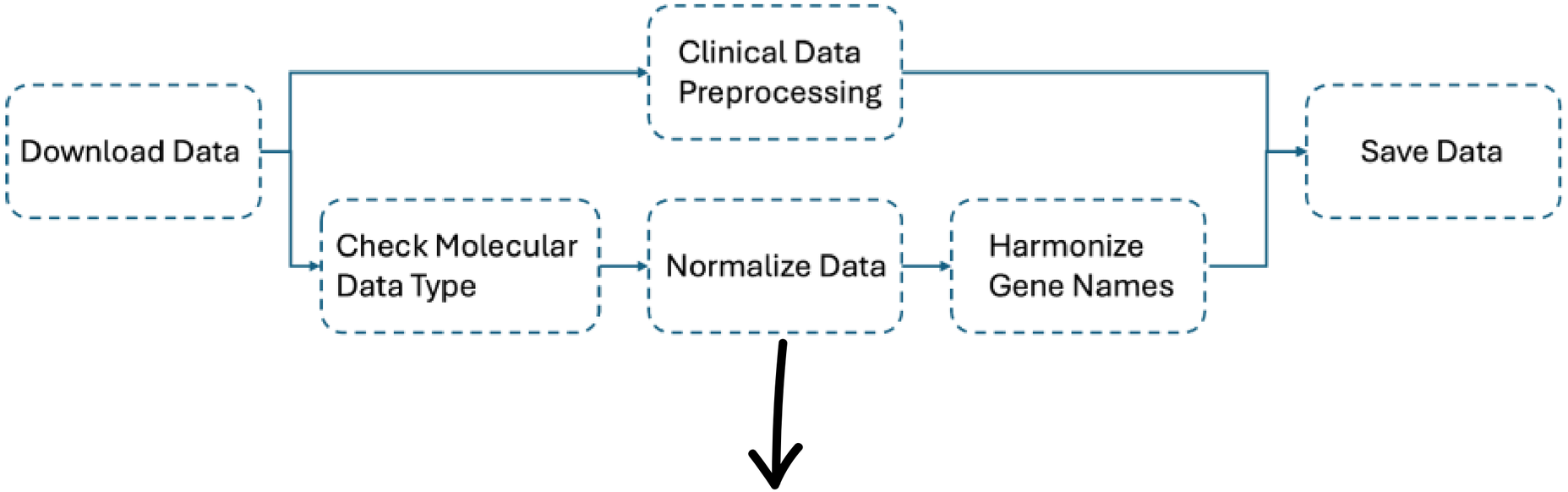
FIRST PART:
COMMENT ON THE PIPELINE

UPDATING THE WORKFLOW



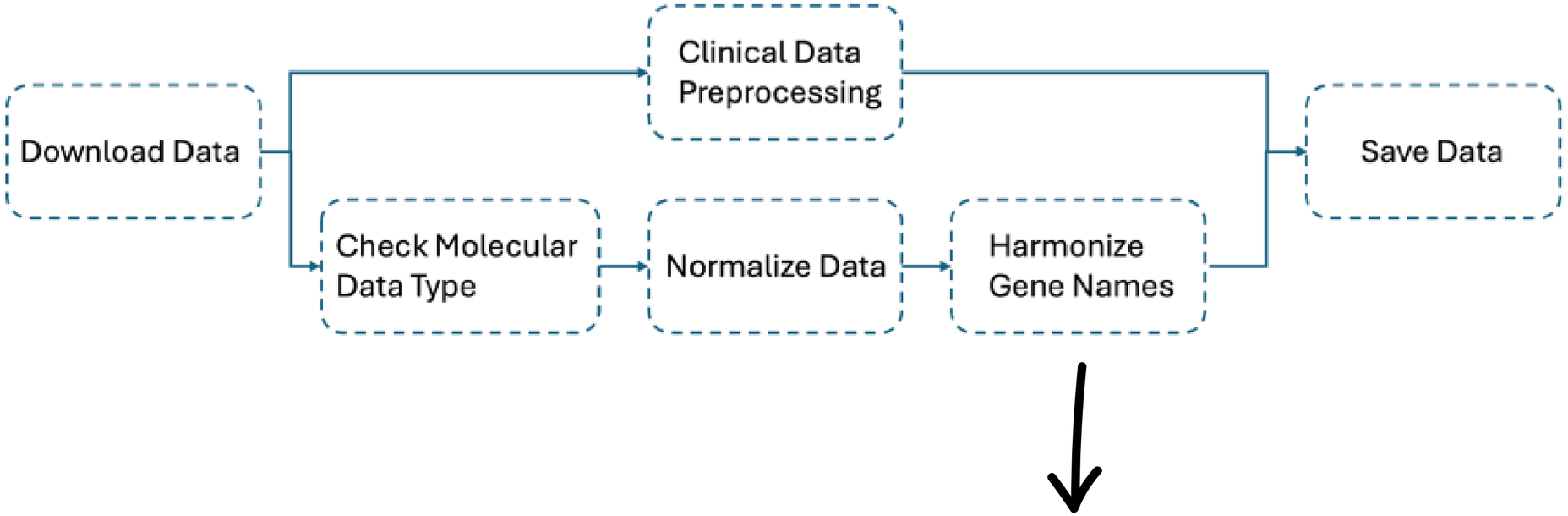
- Verify the datatype such as raw counts, TPM, TPKM
- Identify and flag missing or inconsistent data
- Validate the structure of the dataset , avoid duplicates

UPDATING THE WORKFLOW



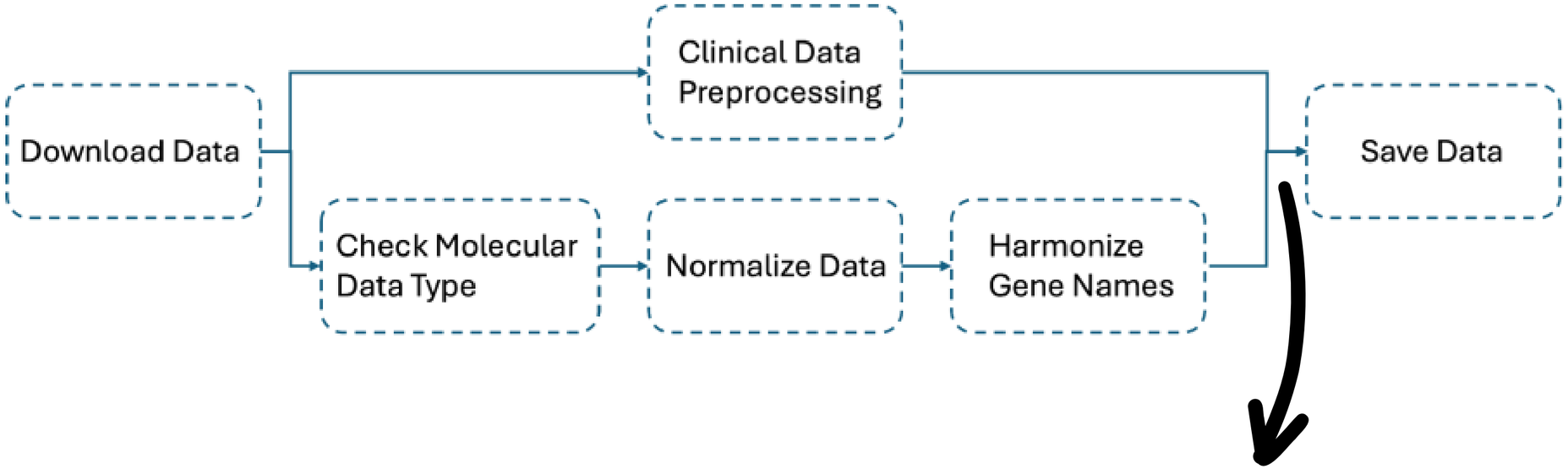
- Normalize the data either to TPM or FPKM
- Using appropriate scaling techniques to ensure values are comparable accross datasets
- Handle batch effects or technical biases

UPDATING THE WORKFLOW



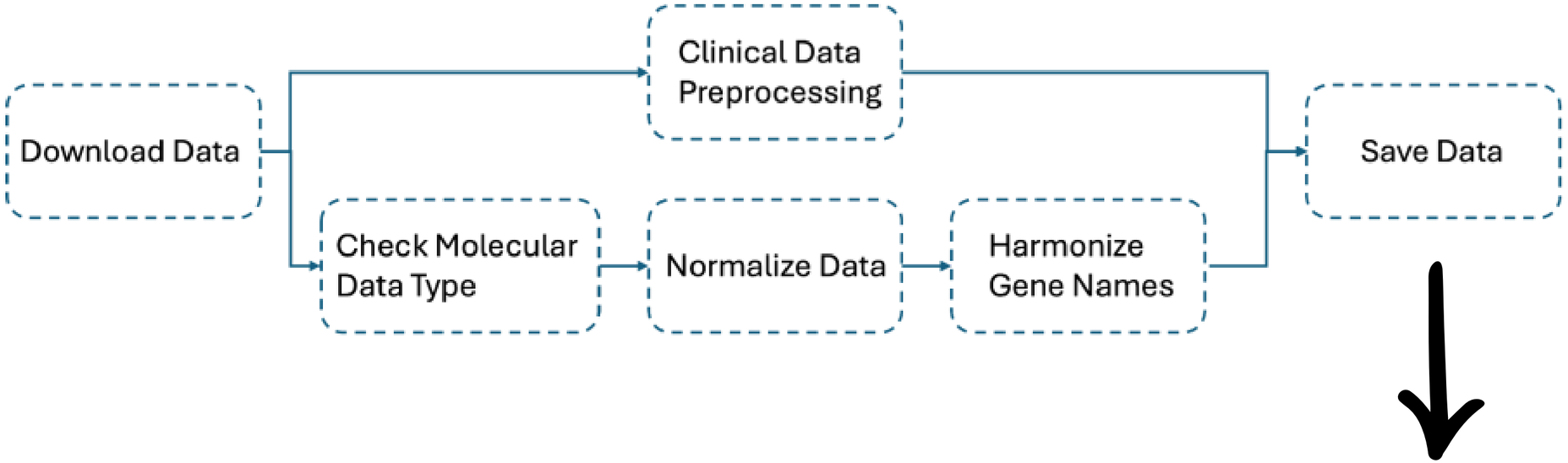
- Standardize gene identifiers across datasets (Ensembl)
- Remove or flag genes with mismatched or missing identifiers
- Map aliases or outdated gene names to the latest standards using reference database (optional)

UPDATING THE WORKFLOW



- Map molecular data to clinical features with patient metadata for instance
- Ensure a one-to-one relationship between molecular samples and clinical entries
- Perform QC to verify consistent sample annotations

UPDATING THE WORKFLOW

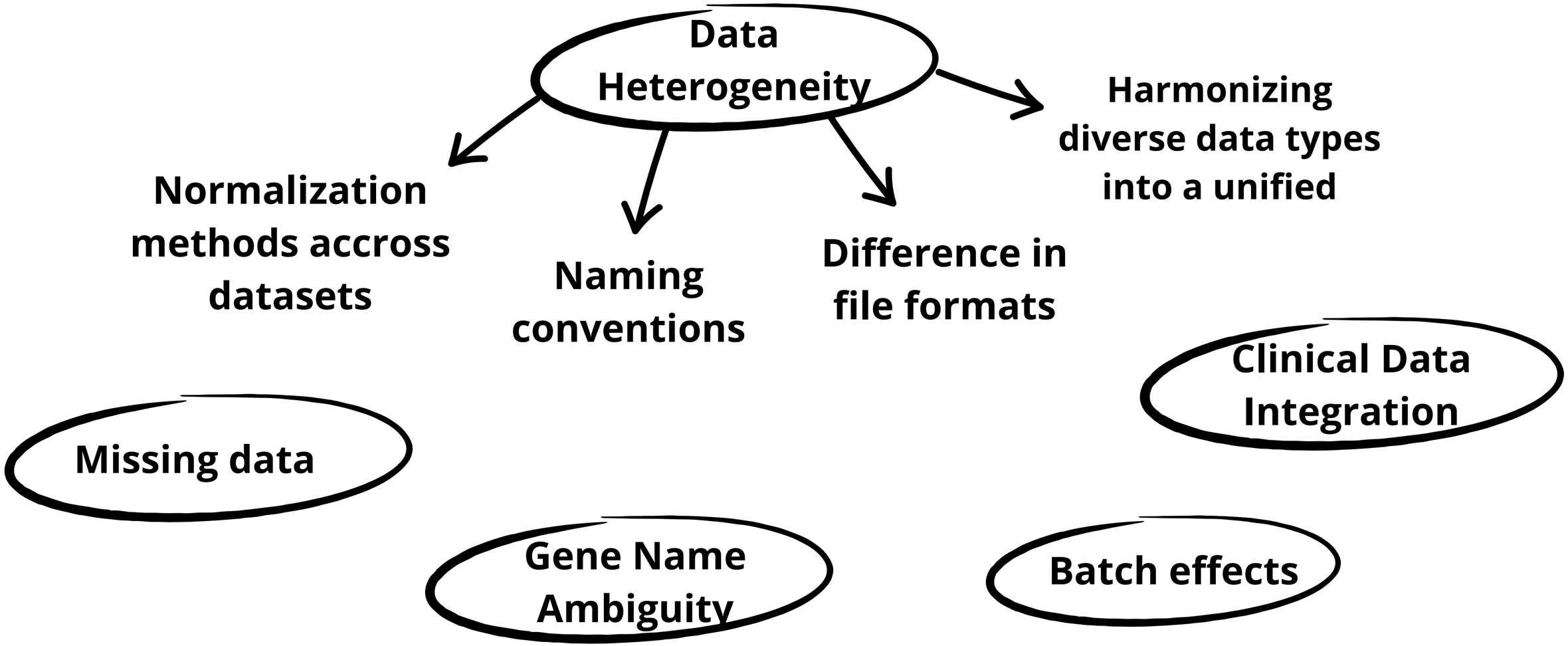


- Ensure formats are consistent for downstream analysis

ENHANCING THE PIPELINE TO ENSURE CONSISTENCY OF DATA

Automated QC	→	**Verify data completennes, formating and scaling consistency**
Use External Gene Databases	→	**Integrating tools like Ensembl API or UniProt to ensure a standard format**
Missing data Handling	→	**Specify a uniform strategy for missing values (imputation, exclusion, flagging)**
Batch Effect correction	→	**Using tools to correct batch effects when integrating datasets from different experiments**
Document each steps	→	**Generate logs for each step**

POSSIBLE CHALLENGES



SECOND PART:
COMMENT ON THE DIFFERENT OUTPUT FILES

WHY DO WE HAVE DIFFERENT FILES FOR ONE DATASET?

Because each output serves a specific analytical purpose

Raw Counts → **The unprocessed data from sequencing typically used for DEA**

Transcripts per million (TPM → **Used for cross-sample comparisons since it accounts for both sequencing depth and gene length**

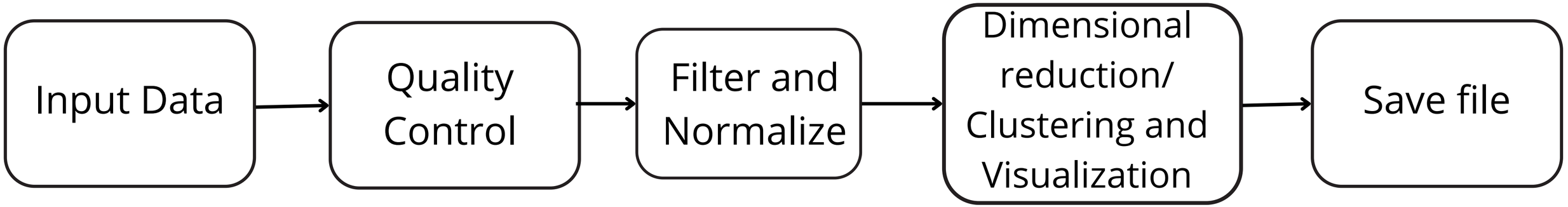
Fragments Per Kilobase Per Million (FPKM) → **Used for within-dataset comparisons or visualization**

And also we need flexibility for Downstream Analysis and to preserve biological information

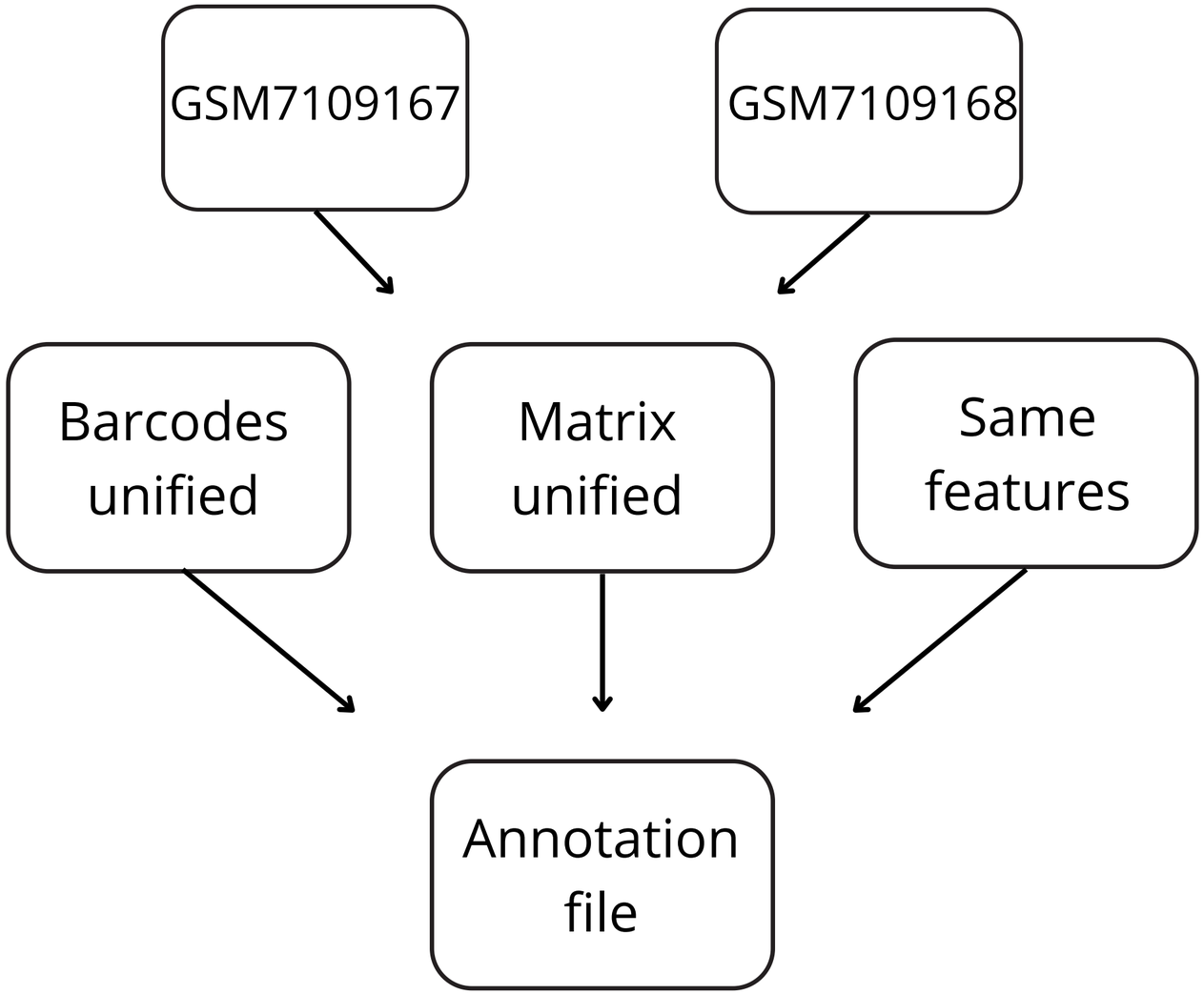
DIFFERENT FILES FOR DOWNSTREAM ANALYSIS

Raw Counts	→	**Differential expression analysis**
Transcripts per million (TPM)	→	**Cross-sample comparison, clustering and visualization (e.g., heatmaps and PCA)**
Fragments Per Kilobase Per Million (FPKM)	→	**Within-dataset comparison, exploratory analysis and visualizations**
Preprocessed Data	→	**Machine learning, pathway analysis, and integration with clinical data**

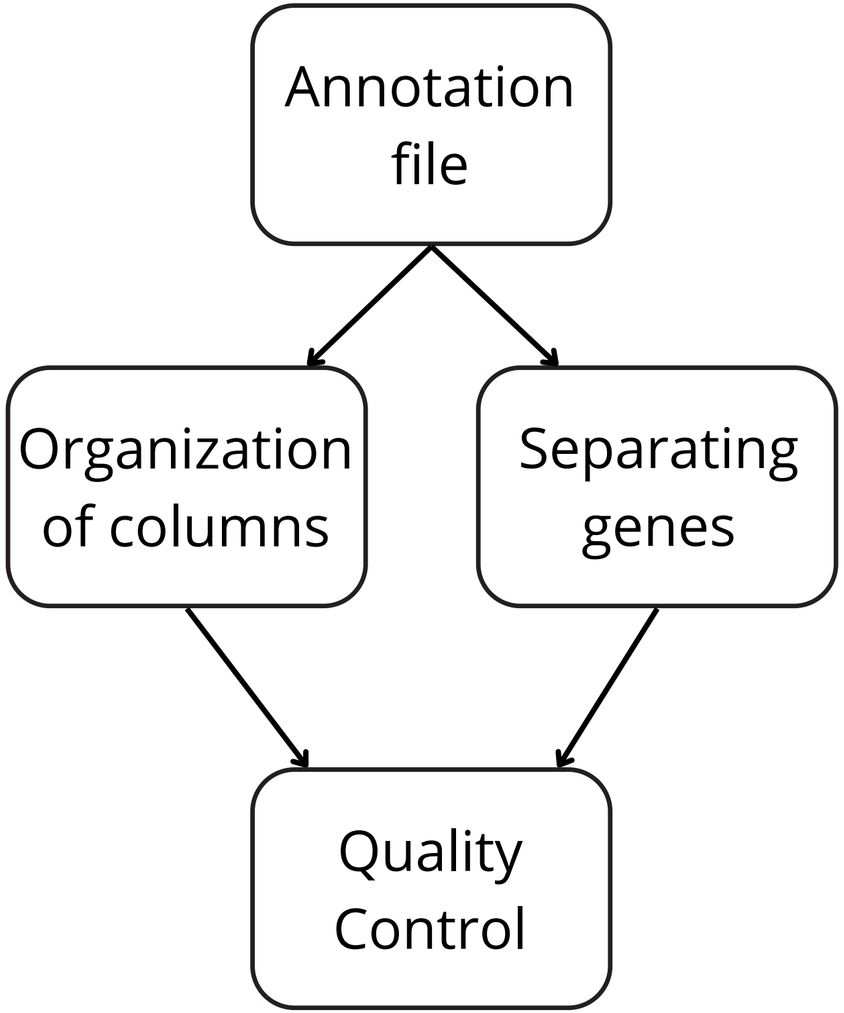
Task 1 : Single-cell RNAseq data integration



INPUT DATA AND UNIFYING FILES

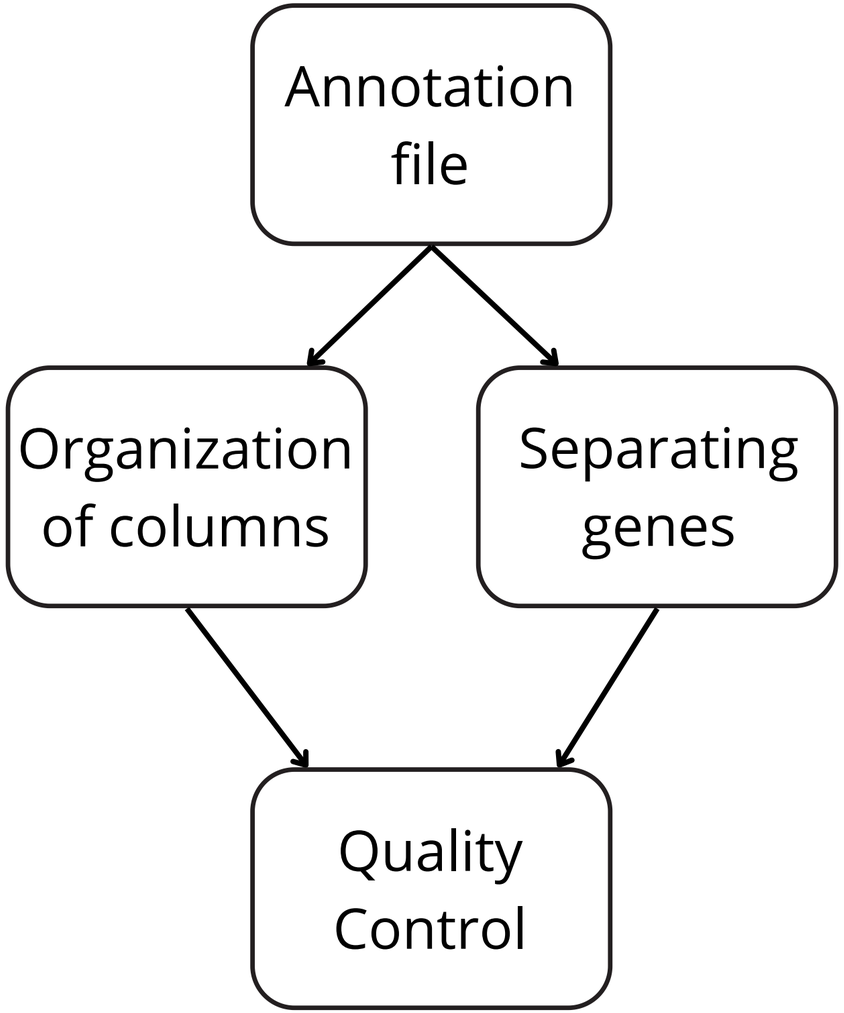


INPUT DATA AND UNIFYING FILES



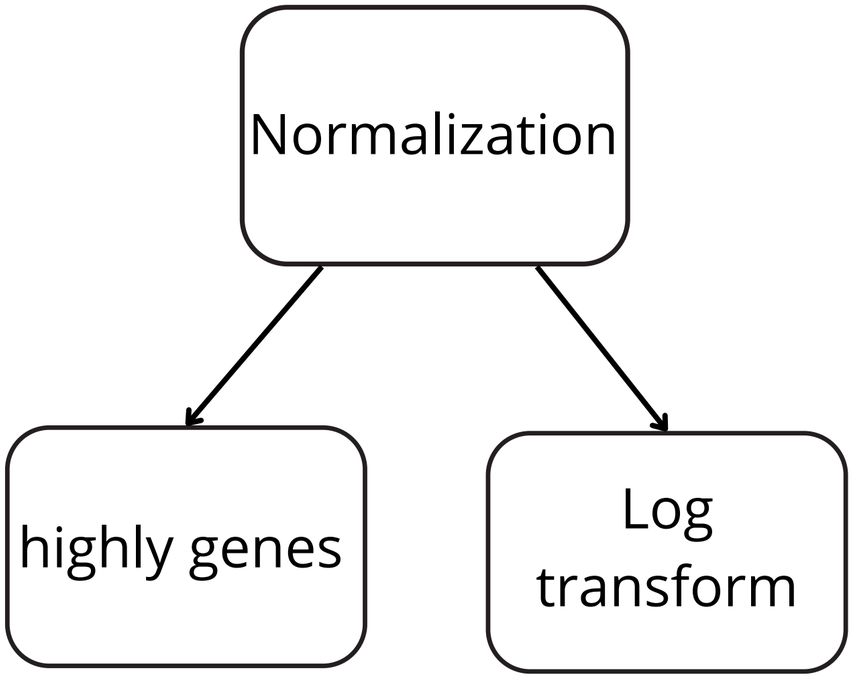

```python
def organizing_and_QC(matrix_all, barcodes_all, features):
  adata = ad.AnnData(X= matrix_all, var=features ,obs=barcodes_all.iloc[:,[0]])

  adata.obs = adata.obs.rename(columns = {0: 'barcodes'})
  adata.obs['sample_id'] = barcodes_all['sample_id'].astype(str).tolist()
  adata.obs['cell_id']   = adata.obs.sample_id + '_'+ adata.obs.barcodes
  adata.obs = adata.obs.set_index('cell_id')
  adata.obs['dataset'] = barcodes_all['dataset'].astype(str).tolist()
  adata.obs = adata.obs.drop(columns=  'barcodes')

  #. Organizing the var dataset also
  #adata.var = adata.var.rename(columns = {1: '',2: 'feature_types'})
  adata.var = adata.var.set_index('code')

  # Separating in different genes:
  # mitochondrial genes
  adata.var["mt"] = adata.var_names.str.startswith("MT-")
  # ribosomal genes
  adata.var["ribo"] = adata.var_names.str.startswith(("RPS", "RPL"))
  # hemoglobin genes.
  adata.var["hb"] = adata.var_names.str.contains(("^HB[^(P)]"))

  #### Calculating quality control metrics
  sc.pp.calculate_qc_metrics(
      adata, qc_vars=["mt", "ribo", "hb"], inplace=True, percent_top=[20,30,50],
      log1p=True
  )
  return adata
```

INPUT DATA AND UNIFYING FILES



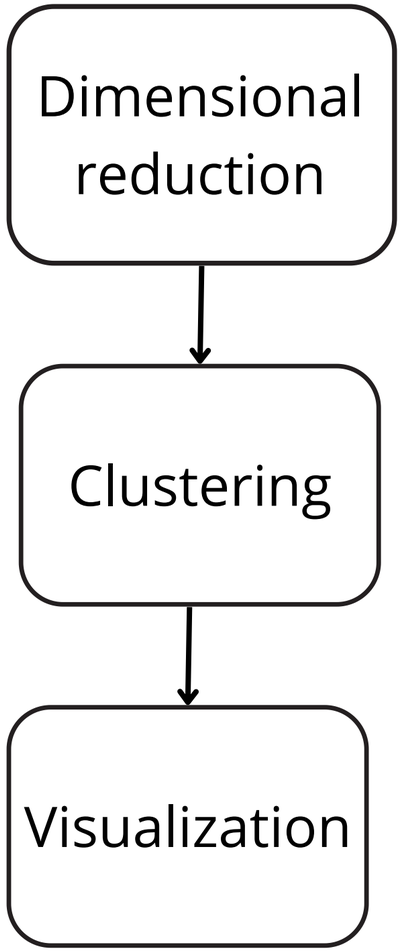

```python
def organizing_and_QC(matrix_all, barcodes_all, features):
  adata = ad.AnnData(X= matrix_all, var=features ,obs=barcodes_all.iloc[:,[0]])

  adata.obs = adata.obs.rename(columns = {0: 'barcodes'})
  adata.obs['sample_id'] = barcodes_all['sample_id'].astype(str).tolist()
  adata.obs['cell_id']   = adata.obs.sample_id + '_'+ adata.obs.barcodes
  adata.obs = adata.obs.set_index('cell_id')
  adata.obs['dataset'] = barcodes_all['dataset'].astype(str).tolist()
  adata.obs = adata.obs.drop(columns=  'barcodes')

  #. Organizing the var dataset also
  #adata.var = adata.var.rename(columns = {1: '',2: 'feature_types'})
  adata.var = adata.var.set_index('code')

  # Separating in different genes:
  # mitochondrial genes
  adata.var["mt"] = adata.var_names.str.startswith("MT-")
  # ribosomal genes
  adata.var["ribo"] = adata.var_names.str.startswith(("RPS", "RPL"))
  # hemoglobin genes.
  adata.var["hb"] = adata.var_names.str.contains(("^HB[^(P)]"))

  #### Calculating quality control metrics
  sc.pp.calculate_qc_metrics(
      adata, qc_vars=["mt", "ribo", "hb"], inplace=True, percent_top=[20,30,50],
      log1p=True
  )
  return adata
```

INPUT DATA AND UNIFYING FILES



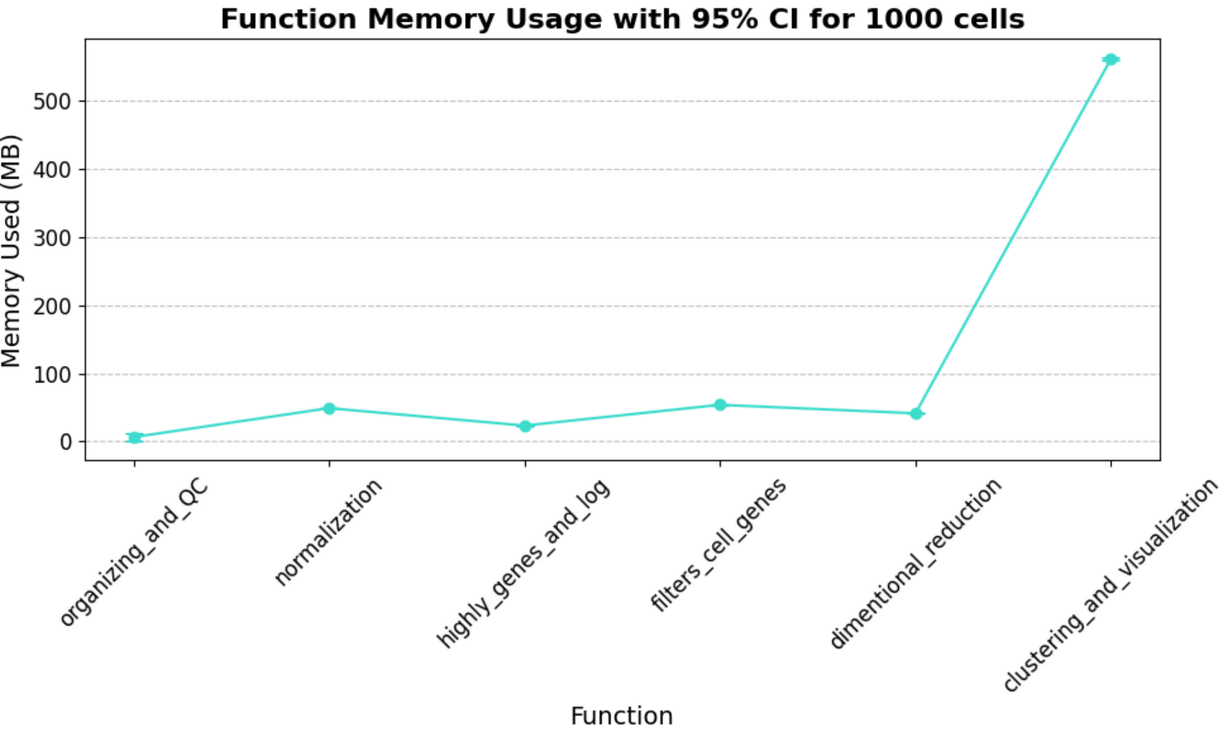

```python
# Part 2 ##############################################################################
# Initial normalization
def normalization(adata, target_sum = 1e4):
  adata.layers["raw_counts"] = adata.X.copy()
  sc.pp.normalize_total(adata, target_sum= target_sum)
  adata.layers['norm_counts'] = adata.X.copy()

  return adata

# Selection of highly variable genes
def highly_genes_and_log(adata, n_top_genes = 2000, subset = False):
  sc.pp.highly_variable_genes(
      adata, flavor="cell_ranger", n_top_genes= n_top_genes, subset= subset
  )  ### It could be setted subset= True to really filter this dataset

  # Logaritmization
  sc.pp.log1p(adata)
  adata.layers["log1p_norm"] = adata.X.copy()

  return adata
```

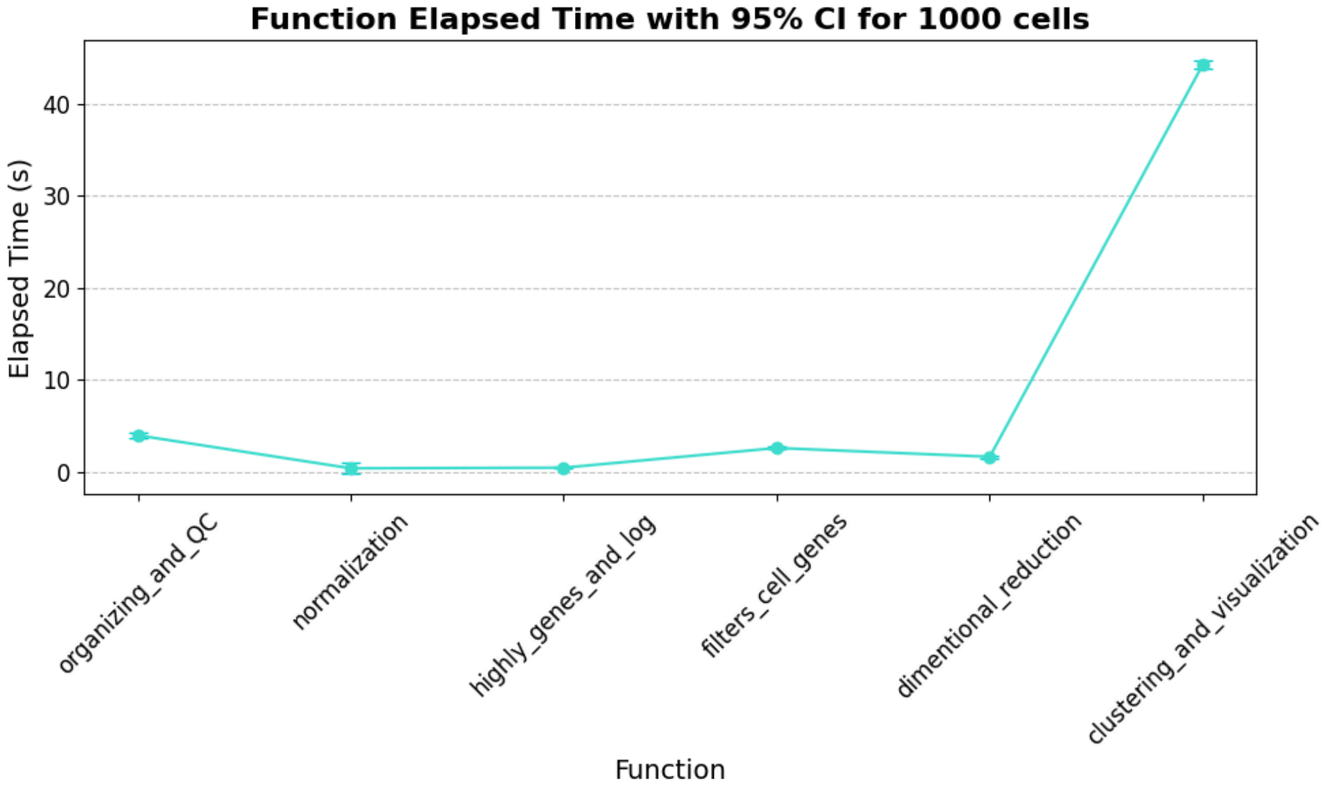
INPUT DATA AND UNIFYING FILES



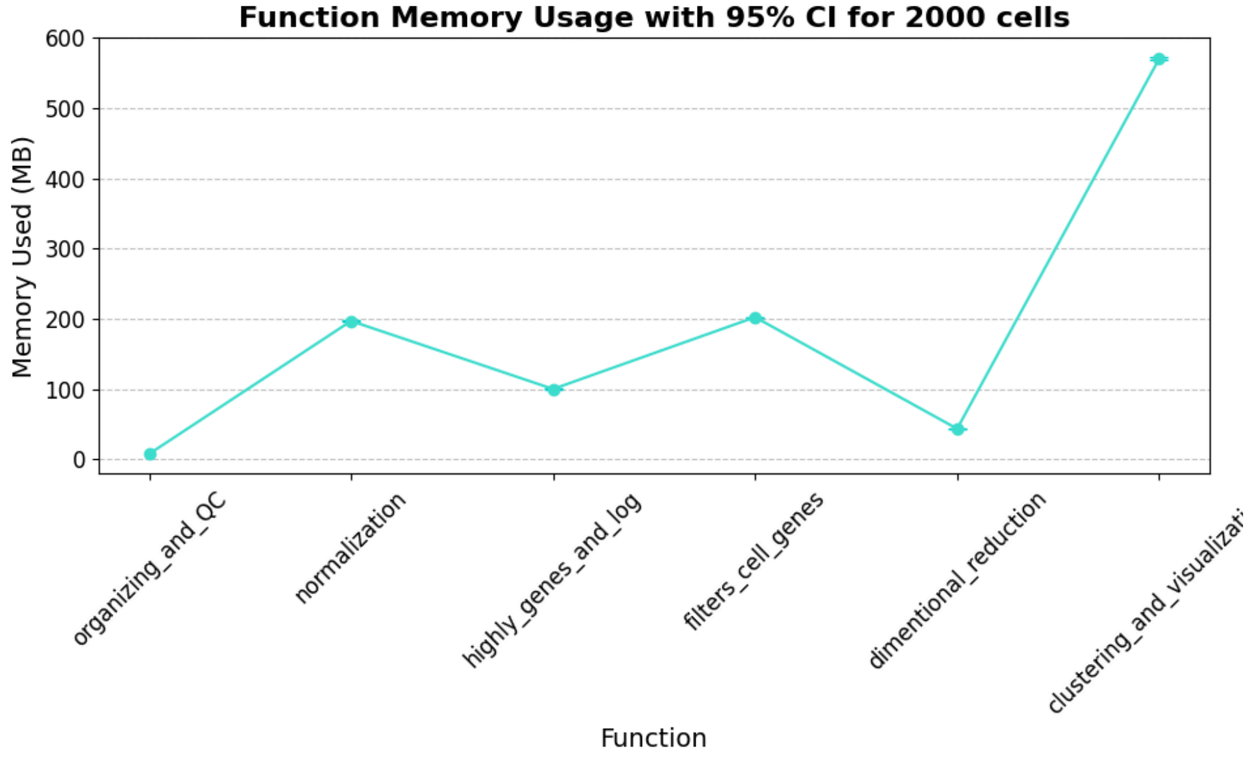

```python
# Part 3 ######################################################################
# dimentional reduction
def dimentional_reduction(adata, n_comps = 50):
  sc.tl.pca(adata, n_comps= n_comps)

  return adata

# Part 4 ######################################################################
# Clustering and Visualization

def clustering_and_visualization(adata, n_neighbors = 15, n_pcs = 30,
                                 early_exaggeration = 12, learning_rate = 1000,
                                 metric = 'euclidean', n_jobs = 1, perplexity = 30,
                                 use_rep = 'X_pca', resolution = 1.0,
                                 key_added = 'leiden_res_1'):

  sc.pp.neighbors(adata, n_neighbors=15, n_pcs=30)
  sc.tl.tsne(adata, n_pcs=50, early_exaggeration = 12, learning_rate = 1000,
             metric = 'euclidean', n_jobs = 1, perplexity = 30, use_rep = 'X_pca')
  sc.tl.umap(adata)
  sc.tl.leiden(adata, resolution=1.0, key_added = 'leiden_res_1')
  return adata
```

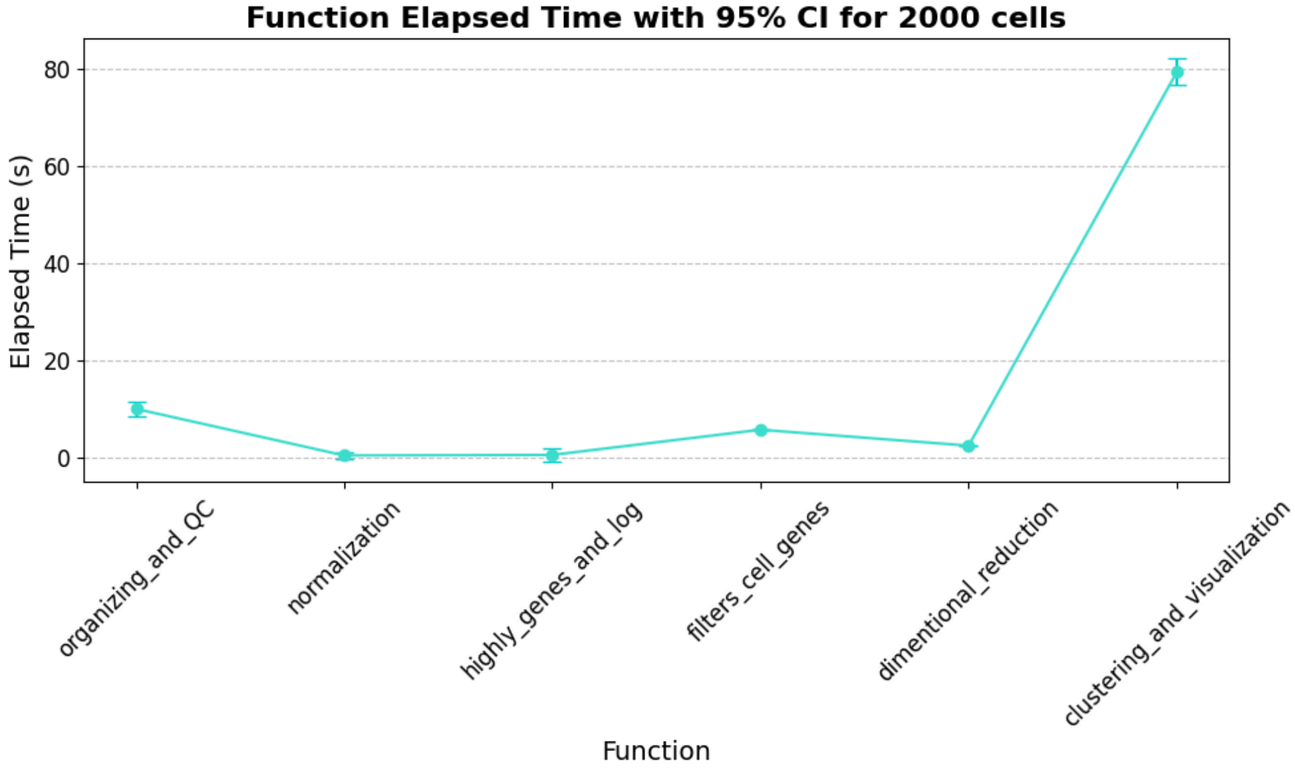
BENCHMARKS AND EFFICIENCY



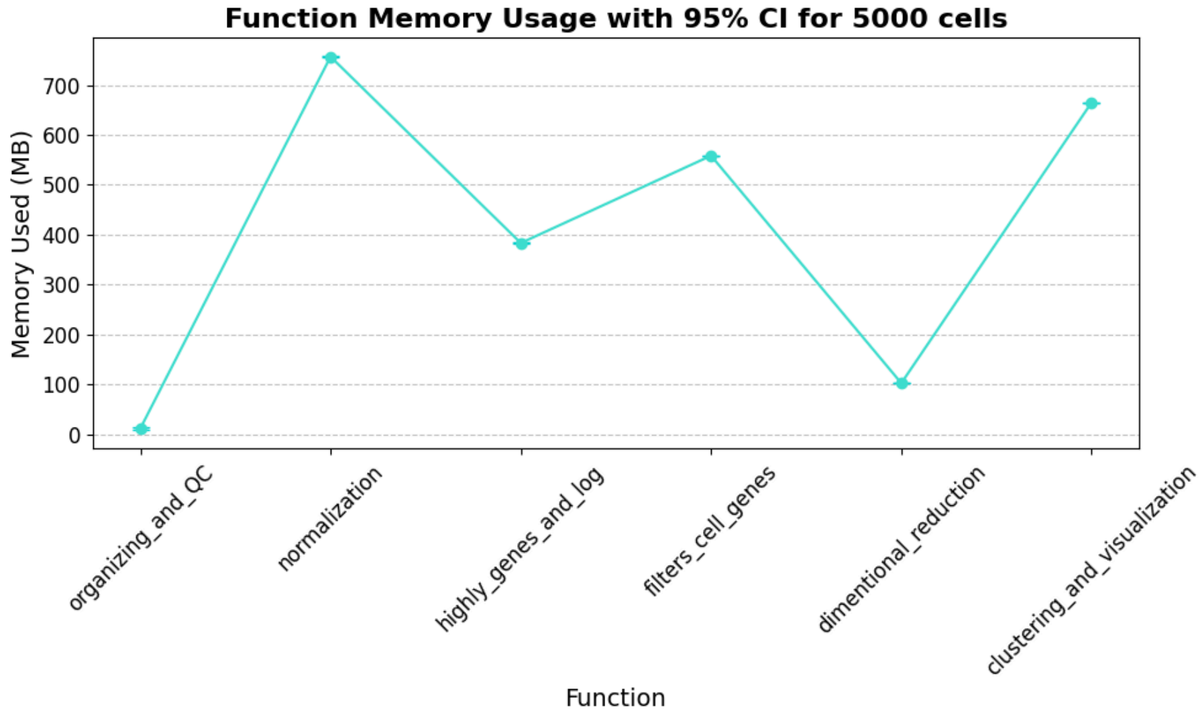


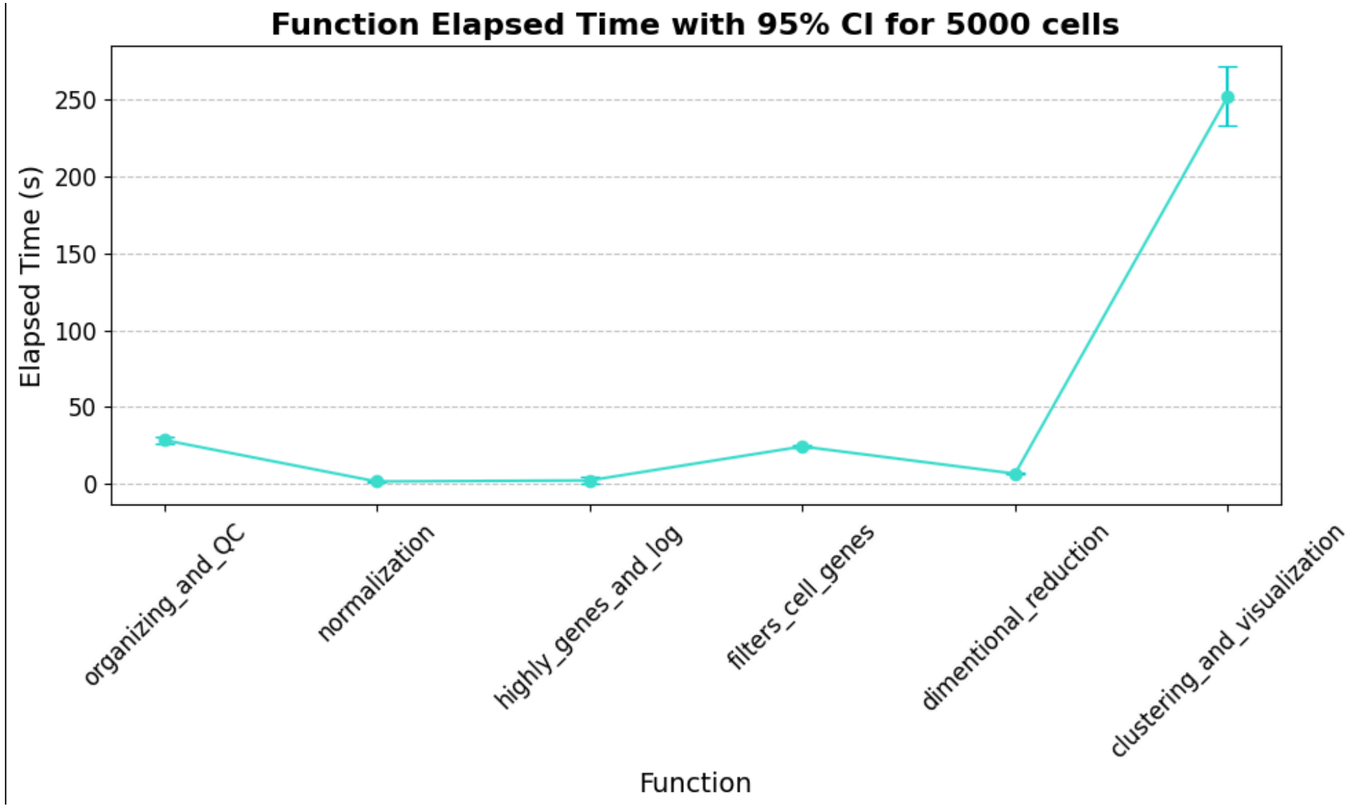
BENCHMARKS AND EFFICIENCY



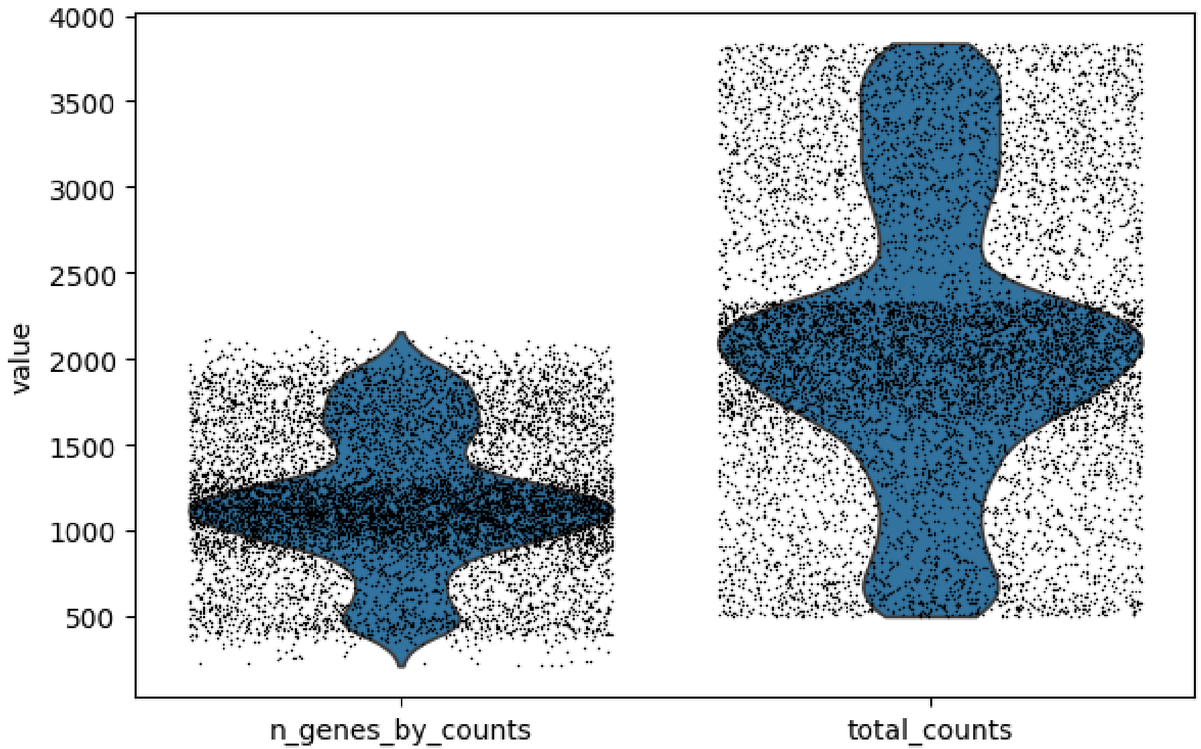


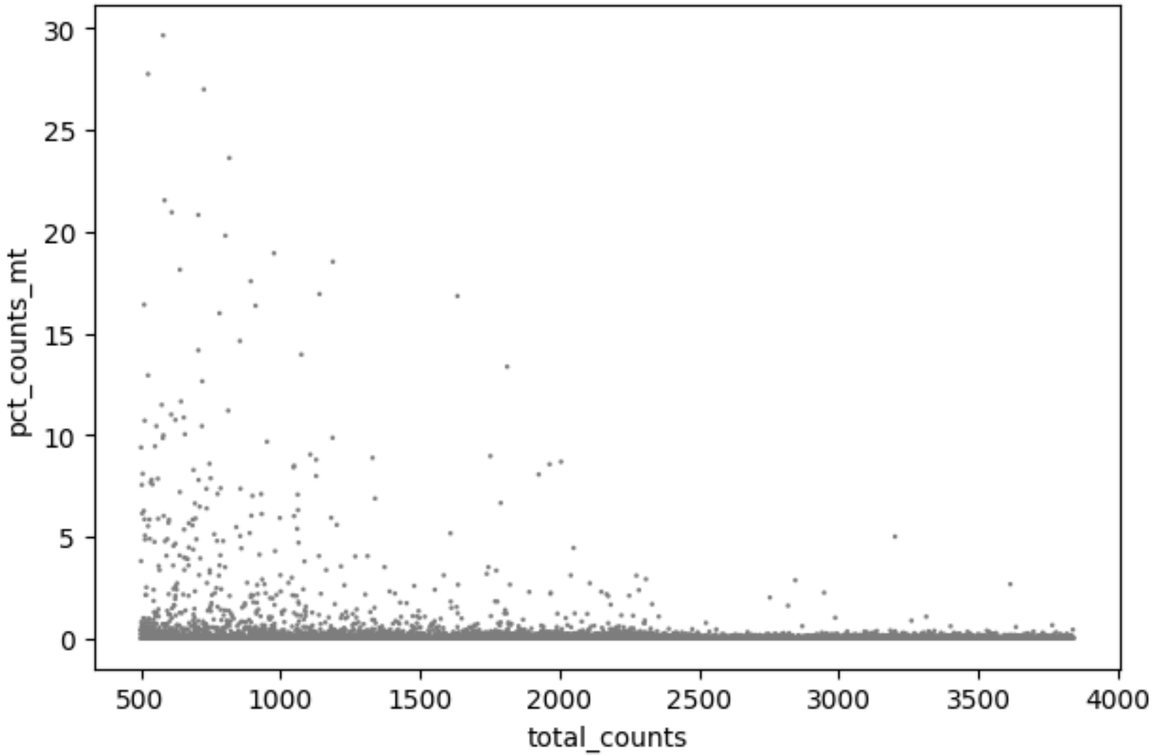
BENCHMARKS AND EFFICIENCY



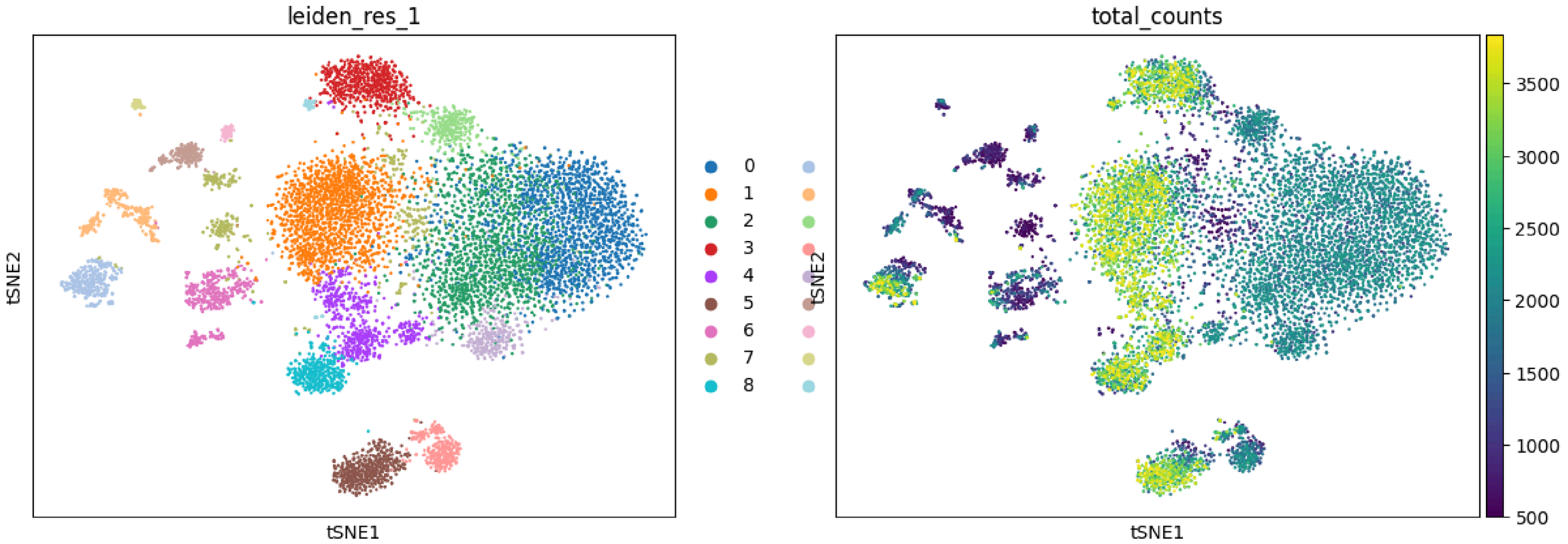


RESULTS OF THE PIPELINE

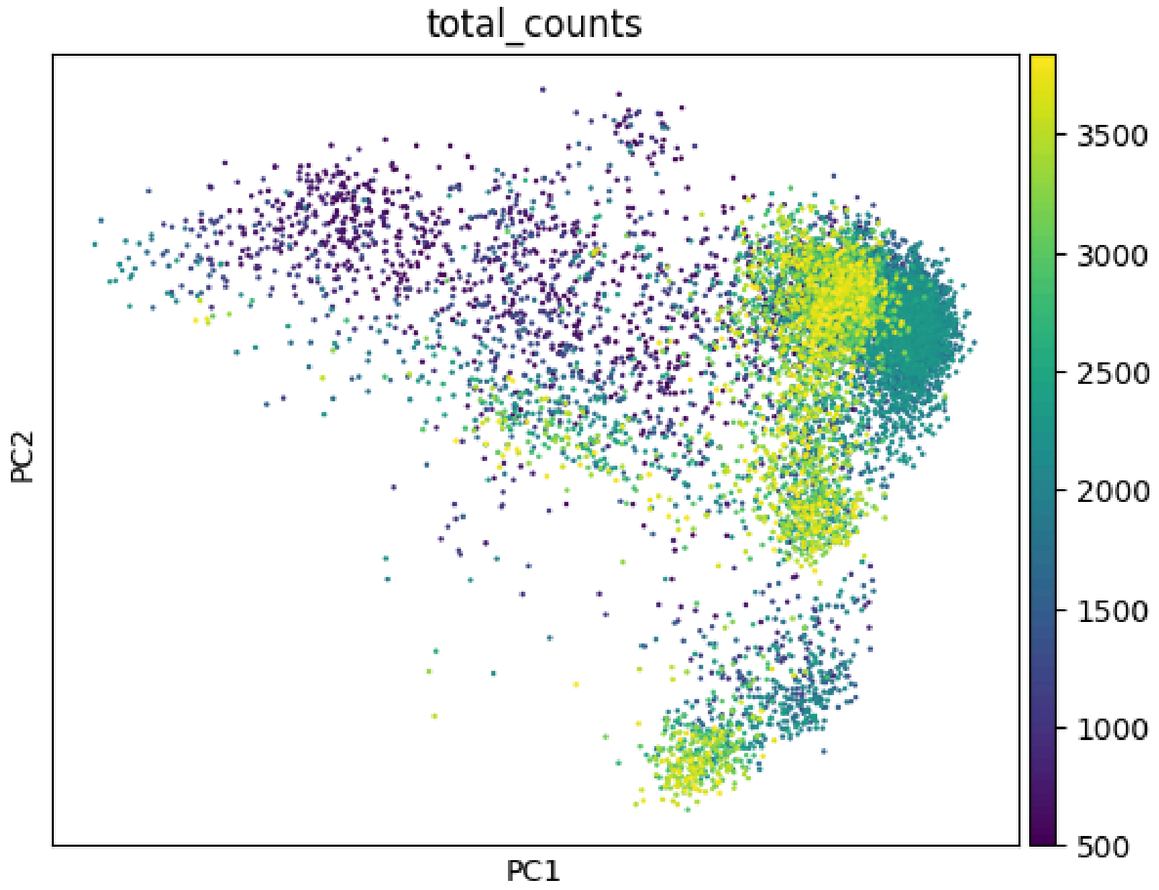


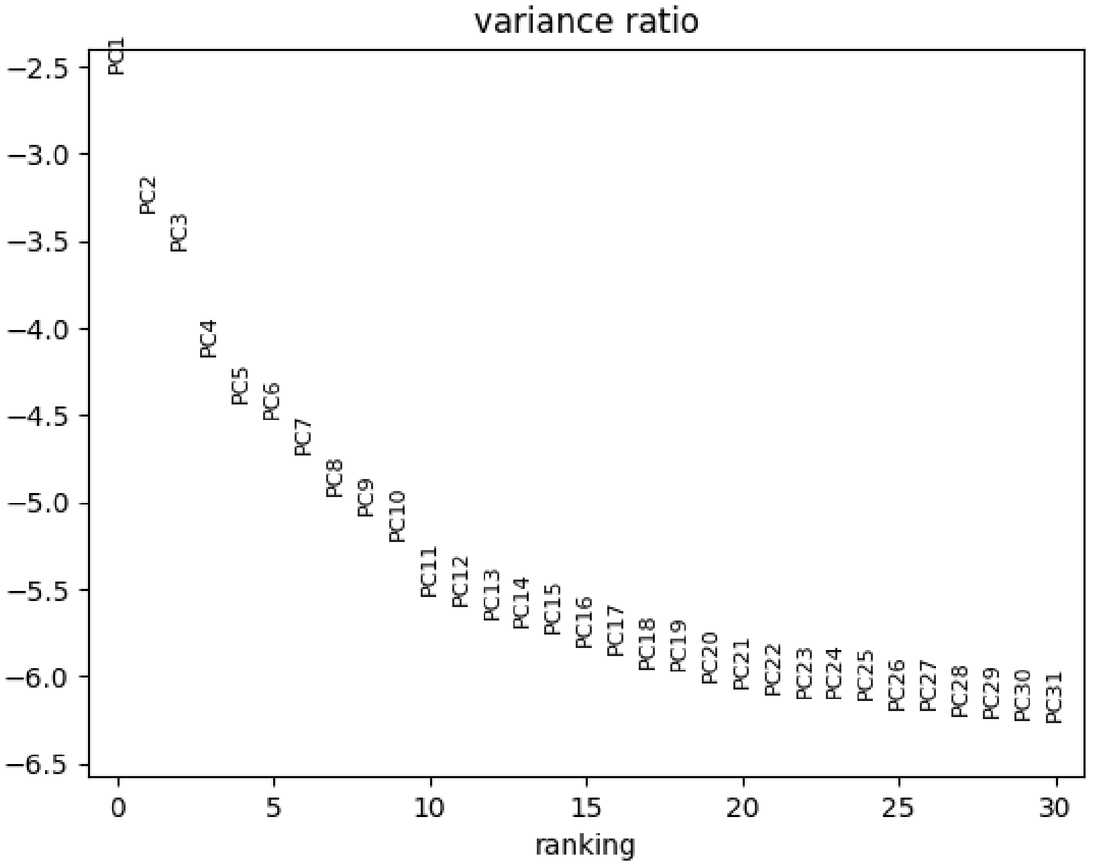


RESULTS OF THE PIPELINE

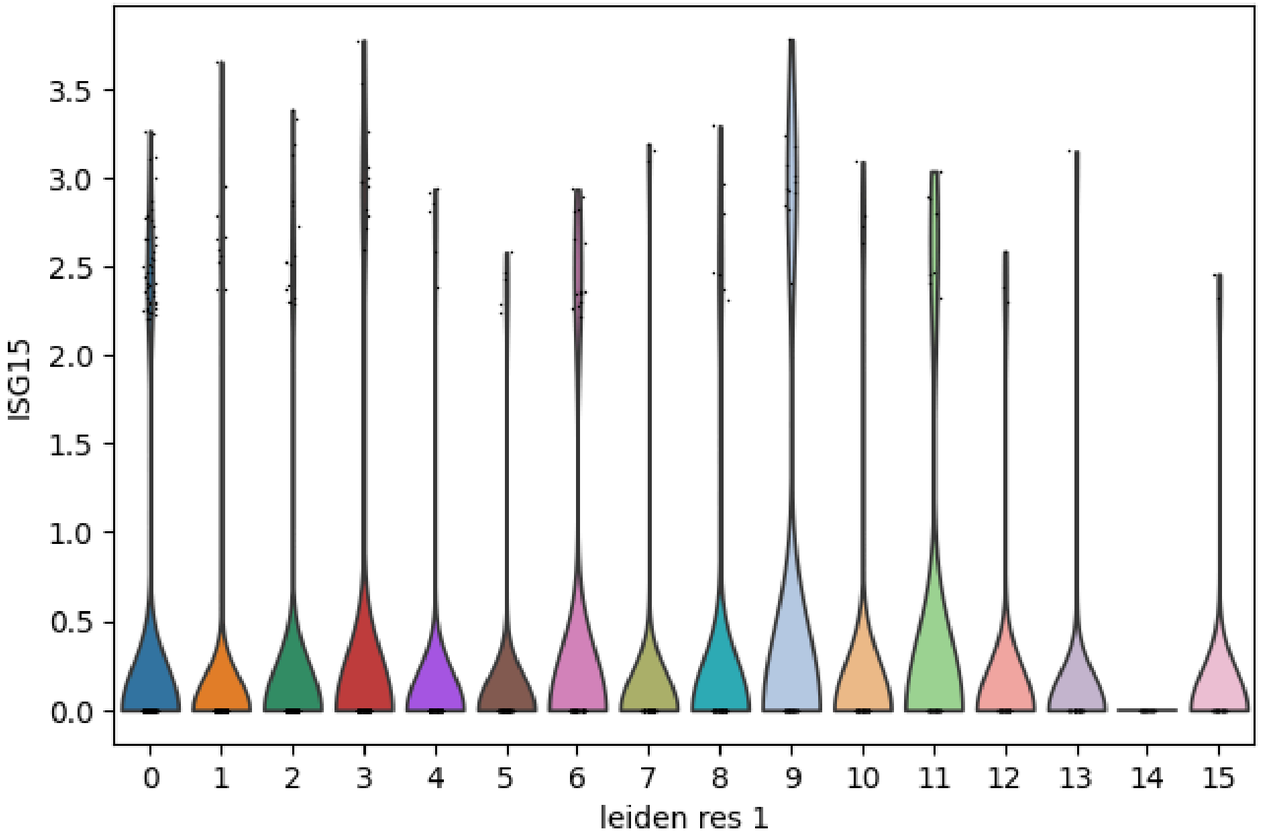


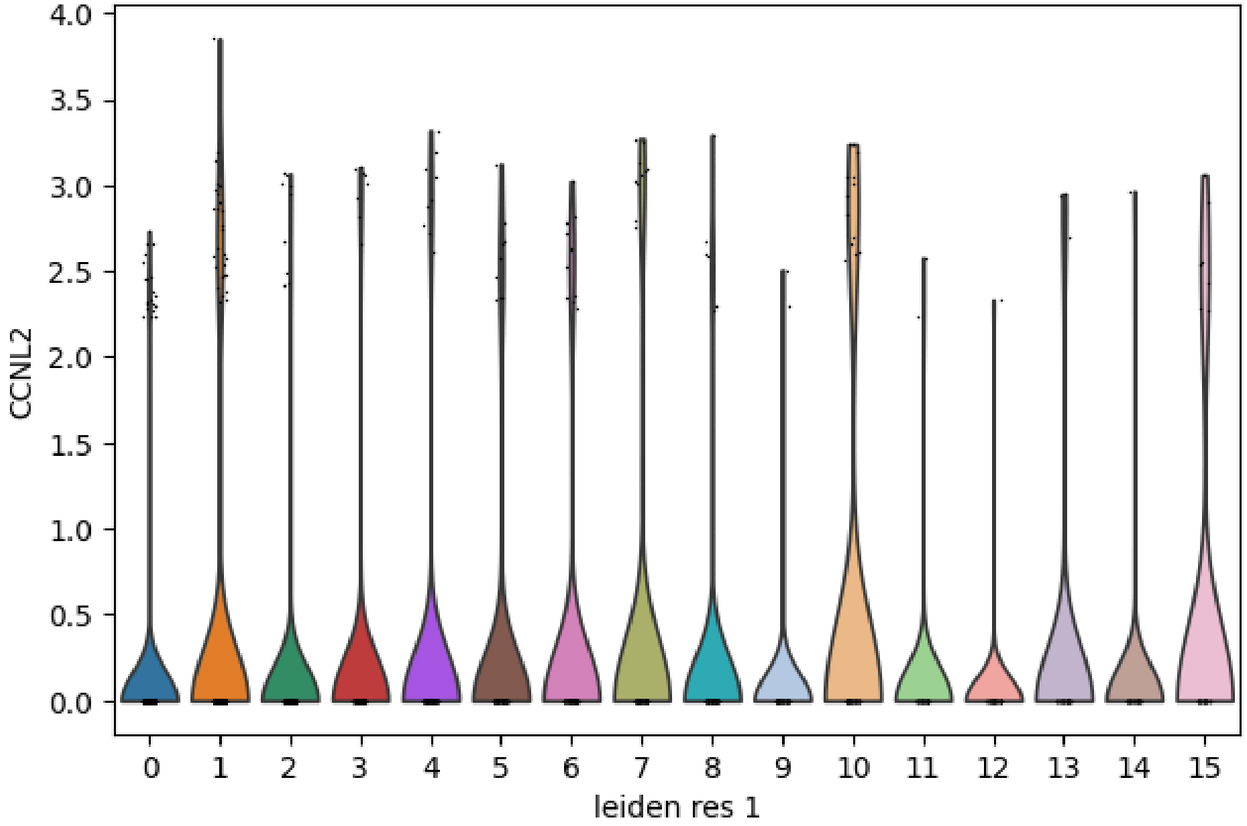
RESULTS OF THE PIPELINE



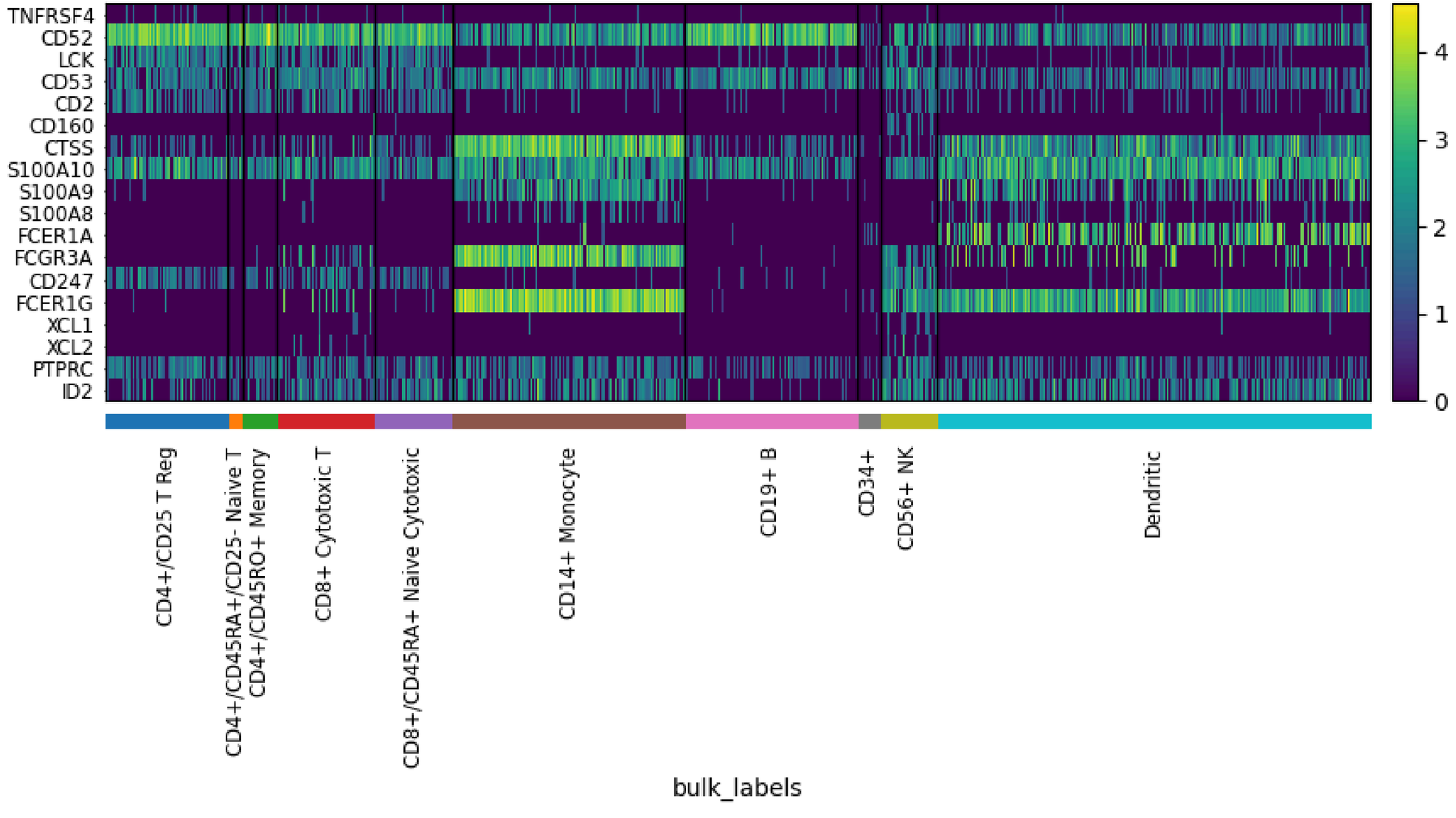


RESULTS OF THE PIPELINE





RESULTS OF THE PIPELINE



ANY QUESTIONS?

THANK YOU!