SC-ASSESSMENT

March 25, 2024

[1]:
```
#IMPORTING THE REQUIRED PACKAGES

import scanpy as sc
import pandas as pd
import numpy as np
import random
import bbknn
from adpbulk import ADPBulk
import anndata as ad
import matplotlib.pyplot as plt
import scanpy.plotting as sc_plt
import scanpy.external as sce
from statsmodels.stats.multitest import multipletests
from scipy.stats import ttest_ind
from pydeseq2.dds import DeseqDataSet, DefaultInference
from pydeseq2.ds import DeseqStats
import decoupler as dc
```

[2]:
```
#READING IN THE MATRIX FILES

s1 = sc.read_10x_mtx("/data/BIOL5177/Assessment/KO1/",
                     var_names = "gene_symbols",
                     cache = True)
s1.var_names_make_unique()

s2 = sc.read_10x_mtx("/data/BIOL5177/Assessment/WT1/",
                     var_names = "gene_symbols",
                     cache = True)
s2.var_names_make_unique()

s3 = sc.read_10x_mtx("/data/BIOL5177/Assessment/WT2/",
                     var_names = "gene_symbols",
                     cache = True)
s3.var_names_make_unique()

s4 = sc.read_10x_mtx("/data/BIOL5177/Assessment/WT3/",
                     var_names = "gene_symbols",
```

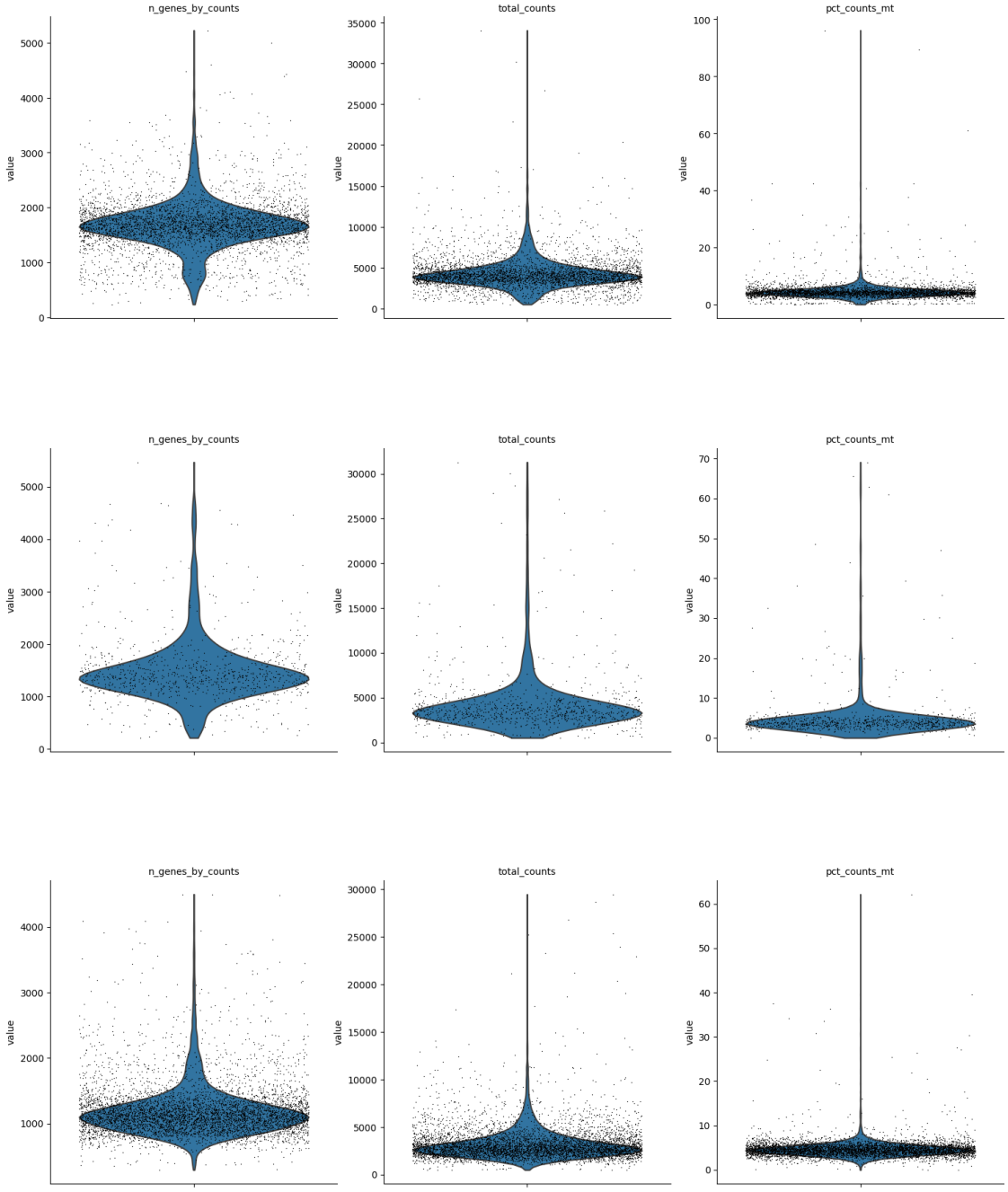
```
                        cache = True)
s4.var_names_make_unique()
```

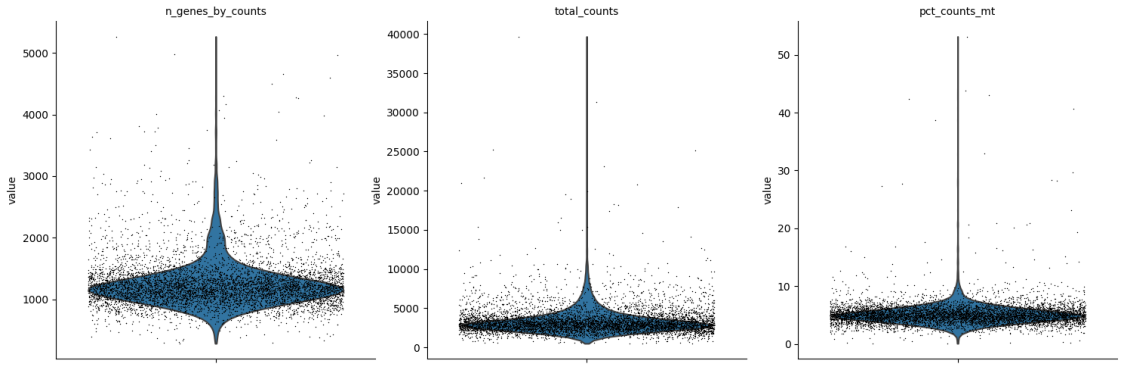
[ ]:
```
'''
# EQUIVALENT R CODE

library(Seurat)

# Read in data for samples

s <- Read10X(data.dir = "/data/BIOL5177/Assessment/")

s <- CreateSeuratObject(counts = s, project = " ")
'''
```

[3]:
```
# ADDING IN A COLUMN OF CONDITIONS SPECIFYING KO AND WT SAMPLES
s1.obs['condition'] = "KO"
s2.obs['condition'] = "WT"
s3.obs['condition'] = "WT"
s4.obs['condition'] = "WT"
```

[4]:
```
# CREATING A LIST OF ALL 4 SAMPLES FOR FURTHER ITERATIONS
sample_list = (s1,s2,s3,s4)
```

[5]:
```
# ITERATING OVER THE LIST
for adata in sample_list:

    # filter genes
    sc.pp.filter_cells(adata, min_genes=200)
    sc.pp.filter_genes(adata, min_cells=3)

    # annotate the group of mitochondrial genes as "mt"
    adata.var["mt"] = adata.var_names.str.startswith("mt-")
    sc.pp.calculate_qc_metrics(
        adata, qc_vars=["mt"], percent_top=None, log1p=False, inplace=True
    )

    # violin plot
    sc.pl.violin(
        adata,
        ["n_genes_by_counts", "total_counts", "pct_counts_mt"],
        jitter=0.4,
        multi_panel=True,
    )
```


n_genes_by_counts
total_counts
pct_counts_mt
value
0
1000
2000
3000
4000
5000
0
5000
10000
15000
20000
25000
30000
35000
0
20
40
60
80
100
n_genes_by_counts
total_counts
pct_counts_mt
value
0
1000
2000
3000
4000
5000
0
5000
10000
15000
20000
25000
30000
0
10
20
30
40
50
60
70
n_genes_by_counts
total_counts
pct_counts_mt
value
1000
2000
3000
4000
0
5000
10000
15000
20000
25000
30000
0
10
20
30
40
50
60



[ ]:
```
'''
# EQUIVALENT R CODE

library(Seurat)

for (adata in sample_list) {
  adata <- subset(adata, subset = nFeature_RNA > 200)
  adata <- subset(adata, subset = nCount_RNA >= 3)
  adata[["mt"]] <- grepl("^mt-", rownames(adata))
  adata <- CalculateQCMetrics(adata,
                              vars.to.regress = "mt",
                              percent.top = NULL,
                              log1p = FALSE)


  VlnPlot(adata, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"),

          pt.size = 0.4, cols = c("lightblue", "lightgreen", "lightcoral"))
}
'''
```

[6]:
```
# REMOVING POTENTIAL DOUBLETS/NOISE/CELLS UNDERGOING APOPTOSIS
for adata in sample_list:
    adata = adata[adata.obs.n_genes_by_counts < 3500, :]
    adata = adata[adata.obs.pct_counts_mt < 20, :].copy()
```

[ ]:
```
'''
# EQUIVALENT R CODE

for (adata in sample_list) {

  # Filter cells based on number of genes detected

  adata <- subset(adata, subset = nFeature_RNA < 3500)
```

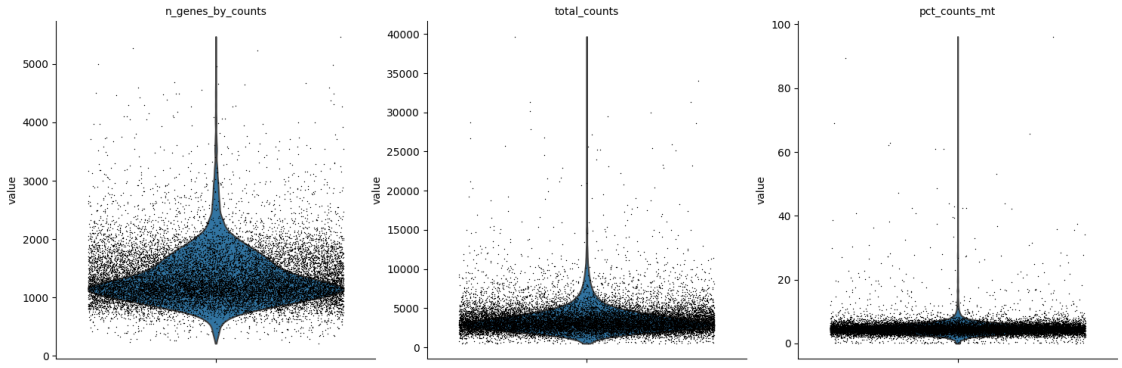
```
  # Filter cells based on percentage of counts from mitochondrial genes

  adata <- subset(adata, subset = percent.mt < 20)
}
'''
```

[7]:
```
# CONCATENATING THE LIST AND CREATING A MERGED OBJECT
merged = ad.concat([s1,s2,s3,s4], join="outer", index_unique="_")
# Assign batch categories
merged.obs['batch'] = ['KO1'] * s1.shape[0] + ['WT1'] * s2.shape[0] + ['WT2'] *␣
 ↪s3.shape[0] + ['WT3'] * s4.shape[0]
```

[ ]:
```
'''
# EQUIVALENT R CODE

merged <- cbind(s1, s2, s3, s4)

merged$batch <- c(rep("KO1", nrow(s1)), rep("WT1", nrow(s2)), rep("WT2",␣
 ↪nrow(s3)), rep("WT3", nrow(s4)))
'''
```

[8]:
```
sc.pp.filter_cells(merged, min_genes=200)
sc.pp.filter_genes(merged, min_cells=3)
```

[9]:
```
# calculating QC metrics related to mitochondrial genes
merged.var['mt'] = merged.var_names.str.startswith('mt-')
sc.pp.calculate_qc_metrics(
        merged, qc_vars=["mt"], percent_top=None, log1p=False, inplace=True
    )

sc.pl.violin(
    merged,
    ["n_genes_by_counts", "total_counts", "pct_counts_mt"],
    jitter=0.4,
    multi_panel=True,
)
```

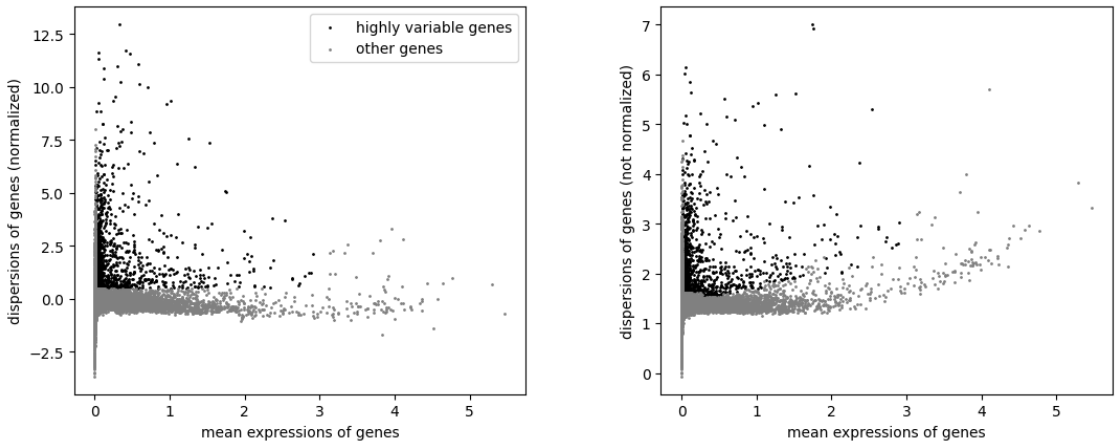


[10]:
```
merged = merged[merged.obs.n_genes_by_counts < 3500, :]
merged = merged[merged.obs.total_counts < 13000, :]
merged = merged[merged.obs.pct_counts_mt < 15, :]
```

[11]:
```
# STORING RAW COUNTS IN LAYERS
merged.X = np.round(merged.X)
merged.layers['counts'] = merged.X
```

[ ]:
```
'''
# EQUIVALENT R CODE

# Round the values in merged data matrix

merged$X <- round(merged$X)

# Assign rounded values to a layer named 'counts'

merged$layers$counts <- merged$X
'''
```

[12]:
```
# Normalize the total counts of each cell to a target sum
sc.pp.normalize_total(merged, target_sum=1e4)

# Log-transform the normalized data
sc.pp.log1p(merged)

# Identify highly variable genes based on their mean expression and dispersion
sc.pp.highly_variable_genes(merged, min_mean=0.0125, max_mean=3, min_disp=0.5)

# Visualize highly variable genes
sc.pl.highly_variable_genes(merged)

# Store the normalized data in a new layer named 'normalized'
merged.layers['normalized'] = merged.X
```



[ ]:
```
'''
# EQUIVALENT R CODE

# Normalize the total counts of each cell to a target sum

merged <- NormalizeData(merged, normalization.method = "total", scale.factor =␣
 ↪1e4)

# Log-transform the normalized data

merged <- ScaleData(merged, do.scale = TRUE, do.center = FALSE, scale.max = 10)

# Identify highly variable genes based on their mean expression and dispersion

merged <- FindVariableFeatures(merged, selection.method = "vst",
                               nfeatures = 2000, verbose = FALSE)

# Visualize highly variable genes

VlnPlot(merged, features = VariableFeatures(object = merged))

# Store the normalized data in a new layer named 'normalized'

merged <- SetAssayData(merged, assays = "normalized", value = as.
 ↪matrix(assay(merged)))
'''
```

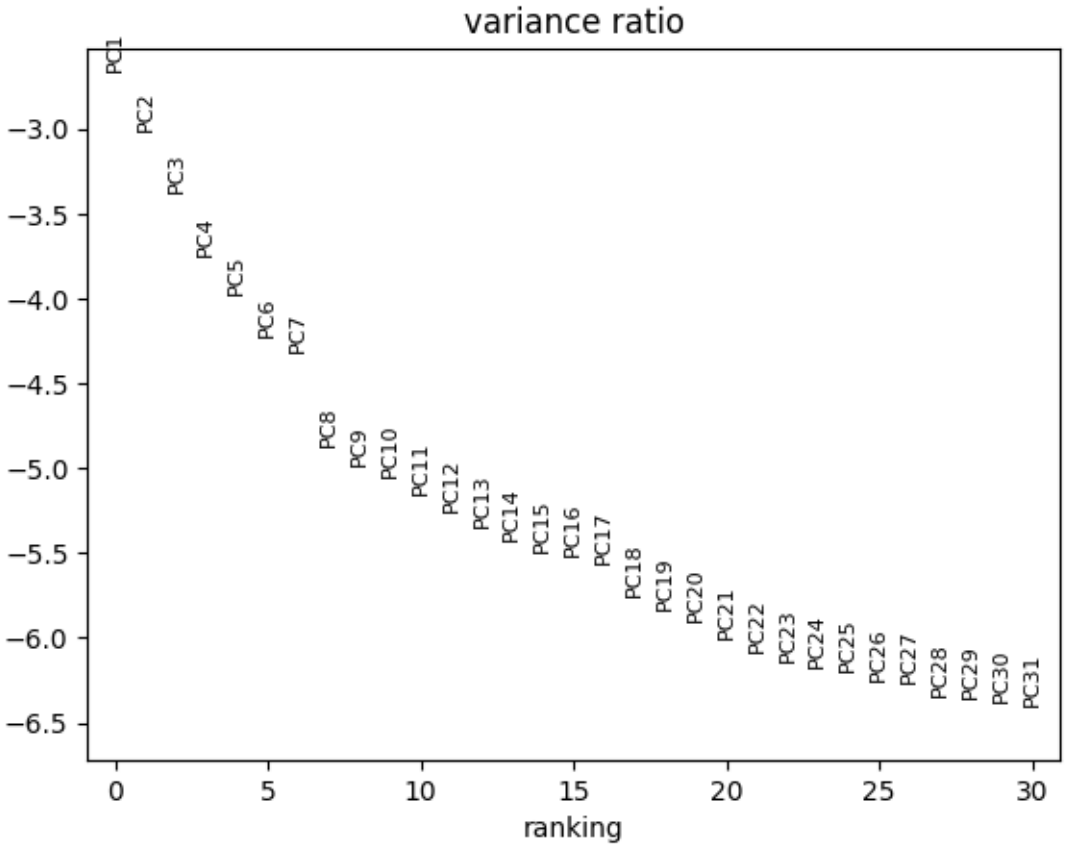
[13]:
```
# FREEZING THE STATE OF ANNDATA OBJECT
merged.raw = merged
```

[14]:
```
# FILTERING
merged = merged[:, merged.var.highly_variable]
```

[ ]:
```
'''
# EQUIVALENT R CODE

# Filter merged dataset to include only highly variable genes

merged <- merged[, VariableFeatures(object = merged)]
'''
```

[15]:
```
# Regressing out effects of total counts per cell and the percentage of␣
 ↪mitochondrial genes expressed
sc.pp.regress_out(merged, ["total_counts", "pct_counts_mt"])
```

[16]:
```
# Scaling and Clipping values exceeding standard deviation 10
sc.pp.scale(merged, max_value=10)

# Performing pca and visualising
sc.tl.pca(merged,svd_solver="arpack")
sc.pl.pca_variance_ratio(merged, log=True)
```



[ ]:
```
'''
# EQUIVALENT R CODE

# Perform Principal Component Analysis (PCA) on the merged dataset

merged <- RunPCA(merged, verbose = FALSE)

# Visualize the variance ratio explained by each principal component on a␣
 ↪logarithmic scale

VizDimLoadings(merged, dims = 1:30, reduction = "pca", pc.use = 1:30, text.var␣
 ↪= "PC", show.var = 0.5, log.scale = TRUE)
'''
```

[17]:
```
# WRITING THE MERGED FILE TO BE USED FOR HARMONY
merged.write('mfile')
```

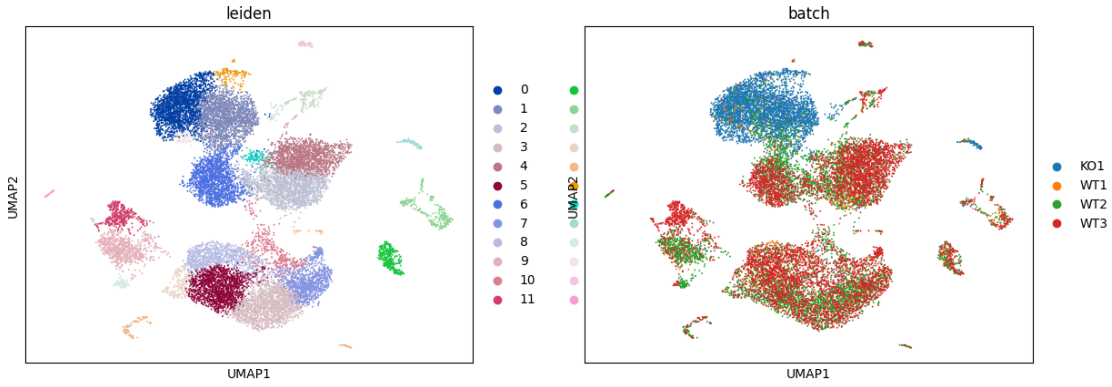
[18]:
```
# Compute a neighborhood graph for the cells based on their gene expression␣
 ↪profiles
# using a shared nearest neighbor (SNN) approach with 10 neighbors and␣
 ↪considering the top 20 principal components
sc.pp.neighbors(merged, n_neighbors=10, n_pcs=20)
```

[19]:
```
# Perform UMAP dimensionality reduction on the merged dataset
sc.tl.umap(merged)
```

[20]:
```
sc.tl.leiden(merged,resolution = 1)
```

[21]:
```
# Visualize the UMAP embedding of the merged dataset
sc.pl.umap(merged,color=['leiden','batch'])
```

```
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
```



[ ]:
```
'''
# EQUIVALENT R CODE

# Compute a neighborhood graph

merged <- FindNeighbors(merged, dims = 1:20, k.param = 10)

# Perform UMAP dimensionality reduction

merged <- RunUMAP(merged, dims = 1:20)

# Perform Leiden clustering

merged <- FindClusters(merged, resolution = 1)

# Visualize UMAP embedding with Leiden clusters and batch information

DimPlot(merged, reduction = "umap", group.by = "orig.ident")
'''
```

[22]:
```
# BBKNN INTEGRATION
sc.external.pp.bbknn(merged, batch_key='batch')
```

```
WARNING: consider updating your call to make use of `computation`
```

[ ]:
```
'''
# EQUIVALENT R CODE

# identify variable features for each dataset independently

features <- SelectIntegrationFeatures(object.list = merged)

# Find commonalities between the data
```

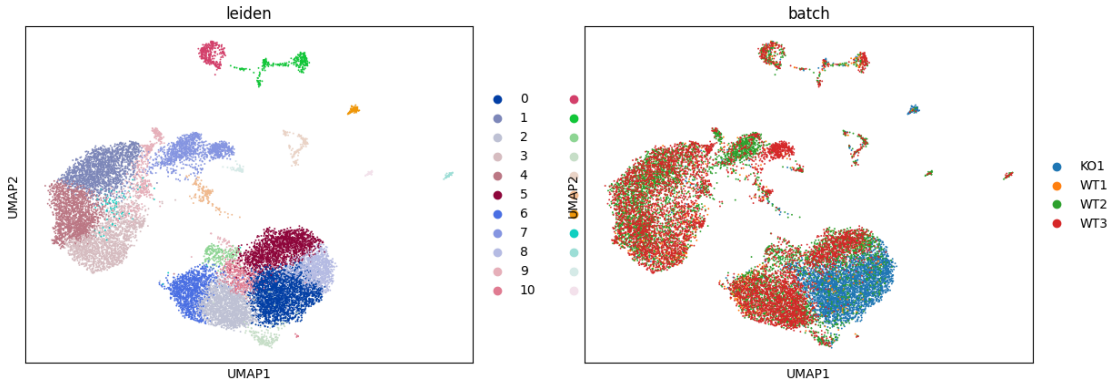
```
merged.anchors <- FindIntegrationAnchors(object.list = merged, anchor.features␣
↪= features)

# Perform integration

merged <- IntegrateData(anchorset = merged.anchors)

library(bbknnR)

# Perform bbknn integration

merged <- bbknnR(merged, batch = "orig.ident")
'''
```

[23]:
```
sc.tl.umap(merged)
```

[24]:
```
sc.tl.leiden(merged, resolution = 1)
```

[25]:
```
sc.pl.umap(merged, color=['leiden','batch'])
```

```
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
```



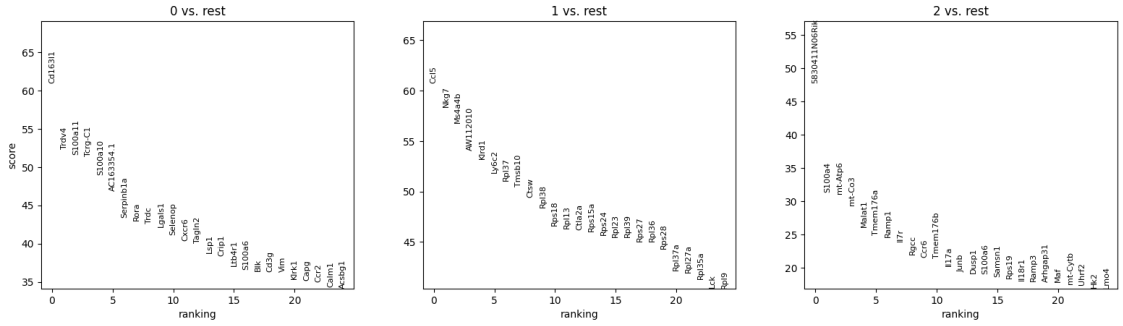
[ ]:
```
'''
# EQUIVALENT R CODE
```

```
DefaultAssay(merged) <- "integrated"

# Run the standard workflow for visualization and clustering

merged <- ScaleData(merged, verbose = FALSE)

merged <- RunPCA(merged, npcs = 20, verbose = FALSE)

# Look at dimensions

ElbowPlot(merged)

# UMAP and clustering

merged <- RunUMAP(merged, reduction = "pca", dims = 1:20)

merged <- FindNeighbors(merged, reduction = "pca", dims = 1:20)

merged <- FindClusters(merged, resolution = 1)
'''
```

[26]:
```
# SETTING BASE PARAMETERS
merged.uns['log1p']['base']=10
```

[27]:
```
# Compute the size of each cluster based on the Leiden clustering results
cluster_size = merged.obs['leiden'].value_counts()

# Get the indices of the top 3 largest clusters
largest_cluster = cluster_size.nlargest(3).index.tolist()

# Subset the data to include cells belonging to the top 3 largest clusters
subset_data = merged[merged.obs['leiden'].isin(largest_cluster)].copy()
```

[28]:
```
# Identify marker genes
sc.tl.rank_genes_groups(subset_data, groupby='leiden', method='wilcoxon')

# Visualize marker genes
sc.pl.rank_genes_groups(subset_data, n_genes=25, sharey=False)  # Adjust␣
 ↪n_genes as needed
```



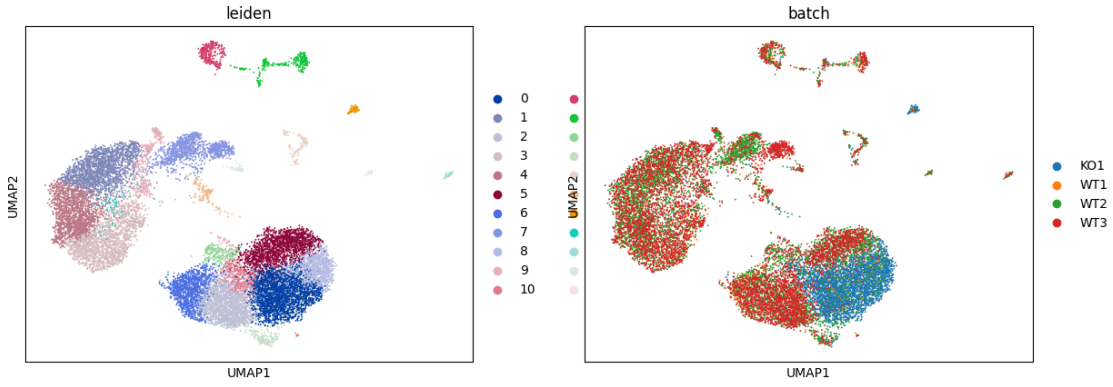
[ ]:
```
'''
# EQUIVAENT R CODE

# Subset covid

subset_data <- subset(merged, idents = c(0,1,2))

markers <- FindAllMarkers(subset_data, only.pos = TRUE, min.pct = 0.25, logfc.
 ↪threshold = 0.25, test.use = "wilcox")

library(dplyr)

top.markers <- markers %>% group_by(cluster) %>% slice_max(n = 10, order_by =␣
 ↪avg_log2FC)
'''
```

[29]:
```
# Annotating clusters
merged.obs['cell_type'] = ''  # Create a new column for cell type annotations
merged.obs.loc[merged.obs['leiden'] == '0', 'cell_type'] = 'V 6+   T cells'  #␣
 ↪Annotate cluster 0
merged.obs.loc[merged.obs['leiden'] == '1', 'cell_type'] = 'IFN- +   T cells' ␣
 ↪# Annotate cluster 1
merged.obs.loc[merged.obs['leiden'] == '2', 'cell_type'] = 'V 4+   T cells'  #␣
 ↪Annotate cluster 2
merged.obs.loc[merged.obs['leiden'] == '3', 'cell_type'] = '3'
merged.obs.loc[merged.obs['leiden'] == '4', 'cell_type'] = '4'
merged.obs.loc[merged.obs['leiden'] == '5', 'cell_type'] = '5'
merged.obs.loc[merged.obs['leiden'] == '6', 'cell_type'] = '6'
merged.obs.loc[merged.obs['leiden'] == '7', 'cell_type'] = '7'
merged.obs.loc[merged.obs['leiden'] == '8', 'cell_type'] = '8'
merged.obs.loc[merged.obs['leiden'] == '9', 'cell_type'] = '9'
merged.obs.loc[merged.obs['leiden'] == '10', 'cell_type'] = '10'
```

[ ]:
```
'''
# EQUIVALENT R CODE

# Annotating clusters

subset_data <- RenameIdents(merged,
                            `0` = "V 6+   T cells",
                            `1` = "IFN- +   T cells",
                            `2` = "V 4+   T cells")

DimPlot(subset_data, label = TRUE)

subset_data$cell_types <- Idents(subset_data)
'''
```

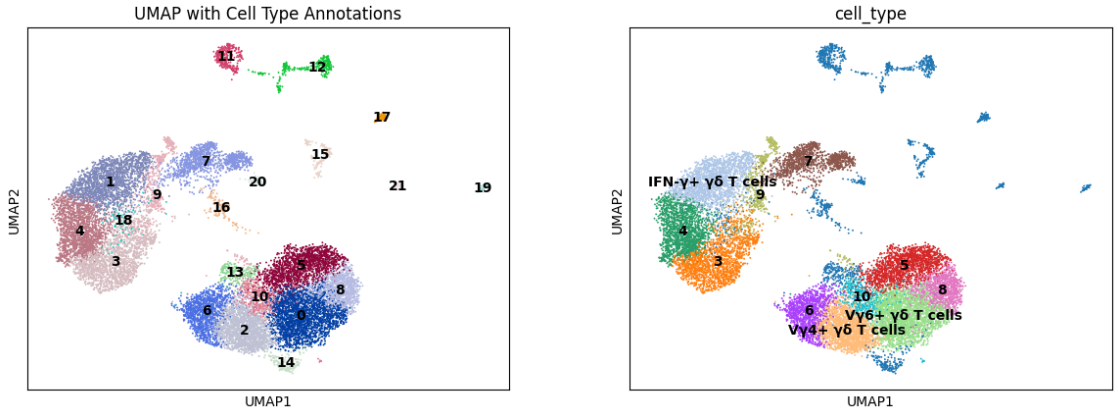
[30]:
```
sc.pl.umap(merged, color=['leiden','batch'])

# Plot UMAP with cell type annotations inside the plot
sc.pl.umap(merged, color=['leiden', 'cell_type'], legend_loc='on data',␣
 ↪title='UMAP with Cell Type Annotations')
```

```
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
```




```
WARNING: The title list is shorter than the number of panels. Using 'color'
value instead for some plots.

/usr/local/lib/python3.10/dist-
```

```
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
```



[31]:
```
pd.DataFrame(subset_data.uns["rank_genes_groups"]["names"]).head(10)
```

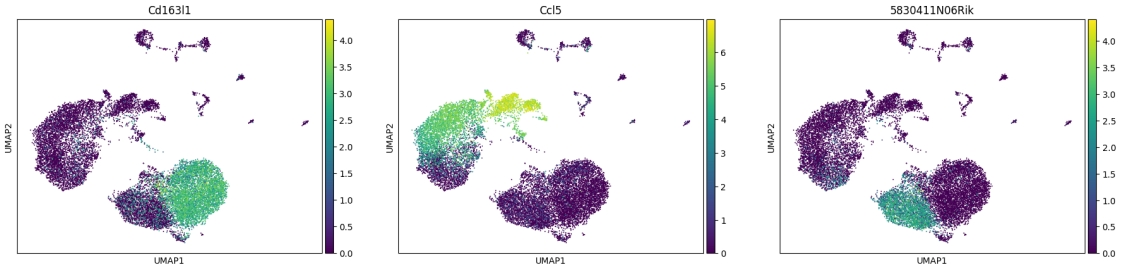
[31]:

	0	1	2
0	Cd163l1	Ccl5	5830411N06Rik
1	Trdv4	Nkg7	S100a4
2	S100a11	Ms4a4b	mt-Atp6
3	Tcrg-C1	AW112010	mt-Co3
4	S100a10	Klrd1	Malat1
5	AC163354.1	Ly6c2	Tmem176a
6	Serpinb1a	Rpl37	Ramp1
7	Rora	Tmsb10	Il7r
8	Trdc	Ctsw	Rgcc
9	Lgals1	Rpl38	Ccr6

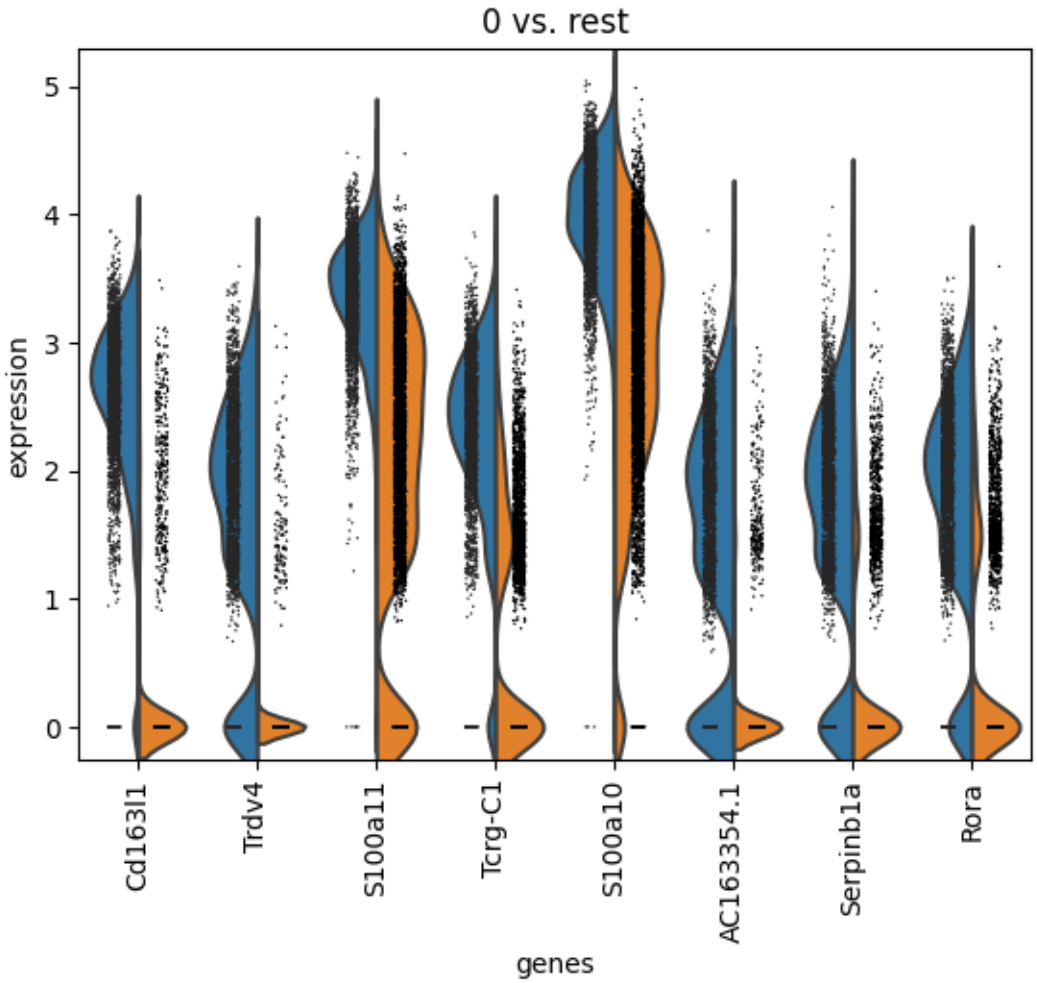
[32]:
```
# Get the ranking of genes for the specified group
ranked_genes_df = sc.get.rank_genes_groups_df(subset_data, group="0").head(10)

# Sort the DataFrame by log-fold change in decreasing order
ranked_genes_df_sorted = ranked_genes_df.sort_values(by='logfoldchanges',␣
 ↪ascending=False)

# Print the sorted DataFrame
print(ranked_genes_df_sorted)
#sc.get.rank_genes_groups_df(subset_data, group="0")
```

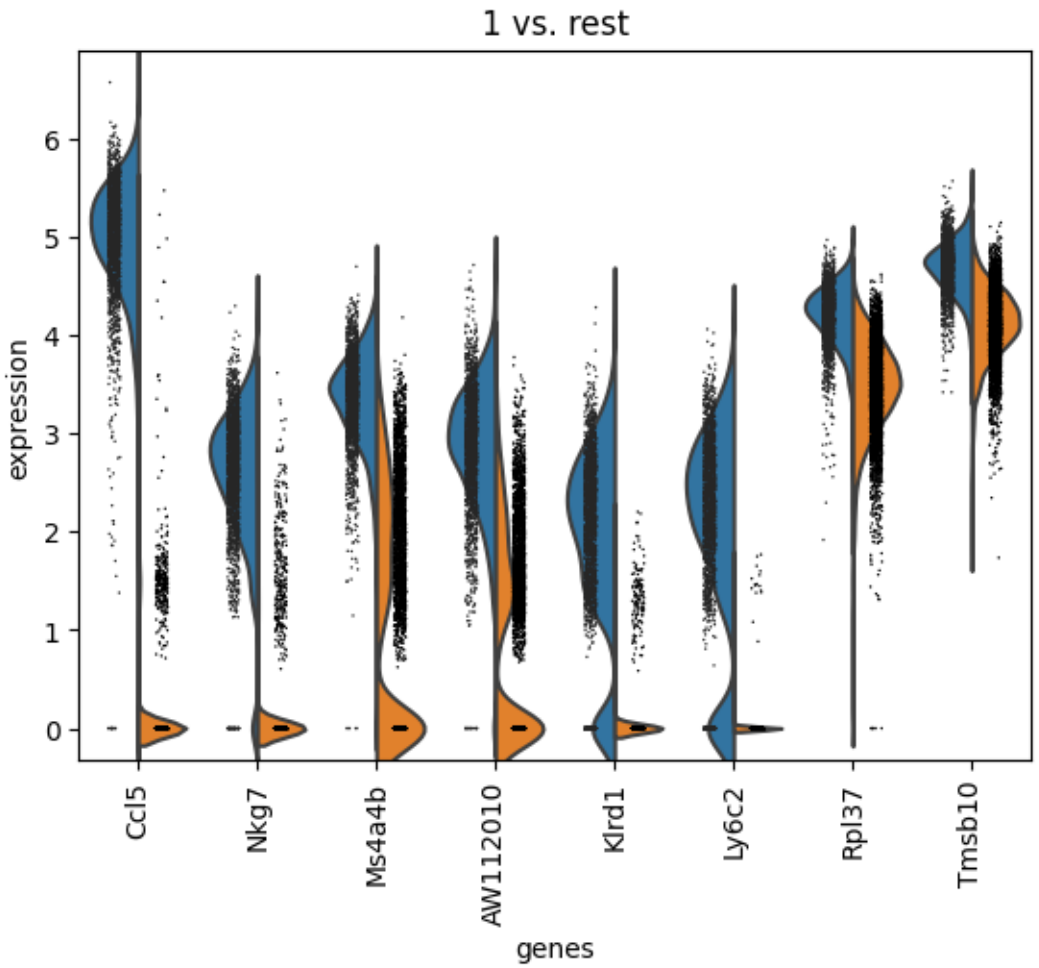
	names	scores	logfoldchanges	pvals	pvals_adj
0	Cd163l1	60.993080	9.266567	0.0	0.0
1	Trdv4	52.400326	8.147546	0.0	0.0
5	AC163354.1	46.864265	6.510146	0.0	0.0
3	Tcrg-C1	51.462463	5.157525	0.0	0.0
6	Serpinb1a	43.393314	4.934445	0.0	0.0
7	Rora	43.041595	4.598277	0.0	0.0
2	S100a11	51.589321	4.595429	0.0	0.0
9	Lgals1	42.117382	3.985237	0.0	0.0
4	S100a10	49.006325	3.825691	0.0	0.0
8	Trdc	42.624172	3.316069	0.0	0.0

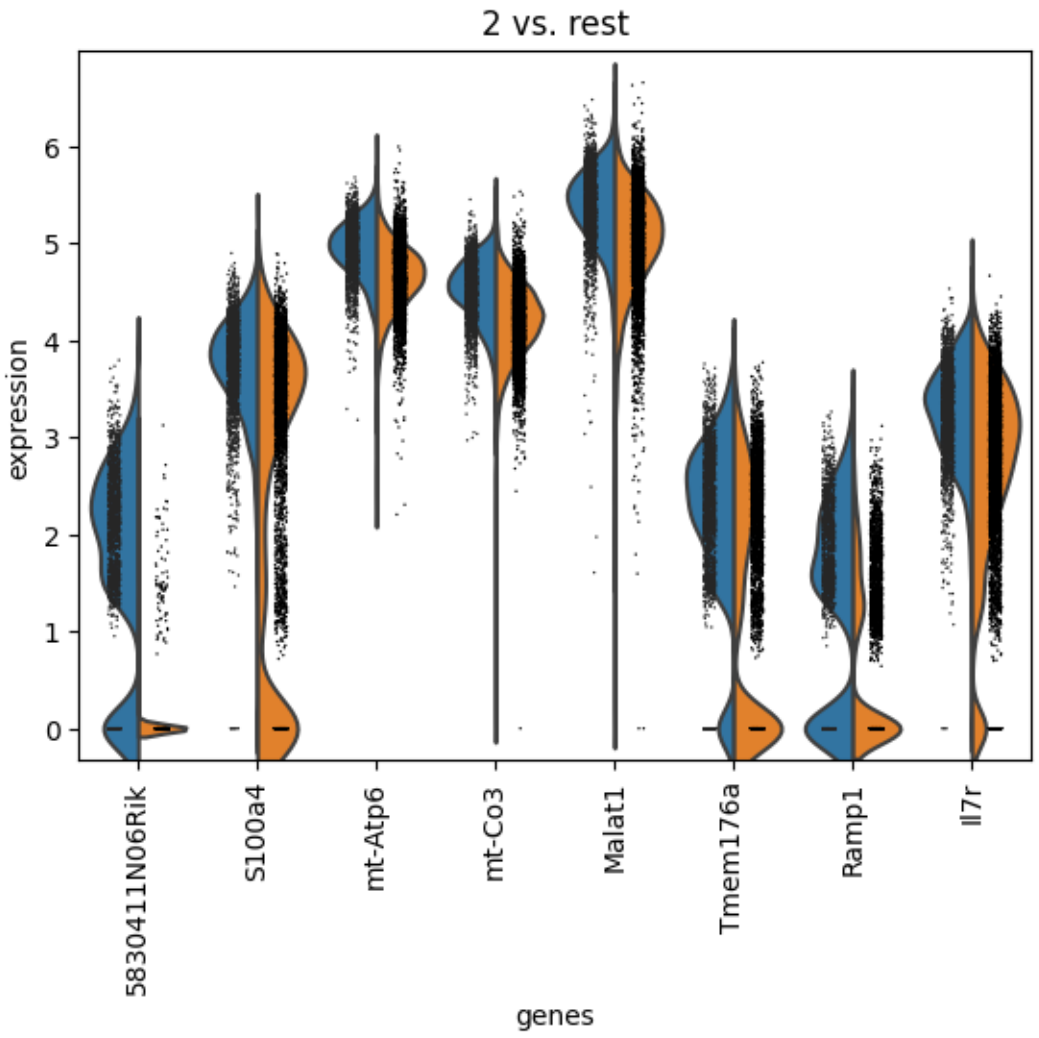
[33]:
```
sc.pl.umap(merged, color=["Cd163l1", "Ccl5", "5830411N06Rik"])
```



[34]:
```
sc.pl.rank_genes_groups_violin(subset_data, groups="0", n_genes=8)
sc.pl.rank_genes_groups_violin(subset_data, groups="1", n_genes=8)
sc.pl.rank_genes_groups_violin(subset_data, groups="2", n_genes=8)
```

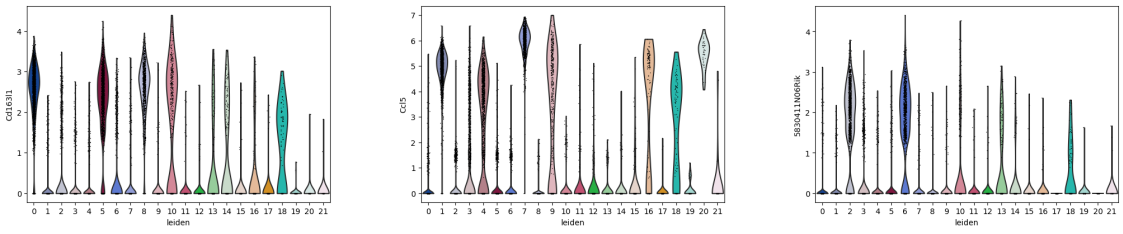

0 vs. rest
expression
0
1
2
3
4
5
Cd163l1
Trdv4
S100a11
Tcrg-C1
S100a10
AC163354.1
Serpinb1a
Rora
genes


1 vs. rest
expression
0
1
2
3
4
5
6
Ccl5
Nkg7
Ms4a4b
AW112010
Klrd1
Ly6c2
Rpl37
Tmsb10
genes



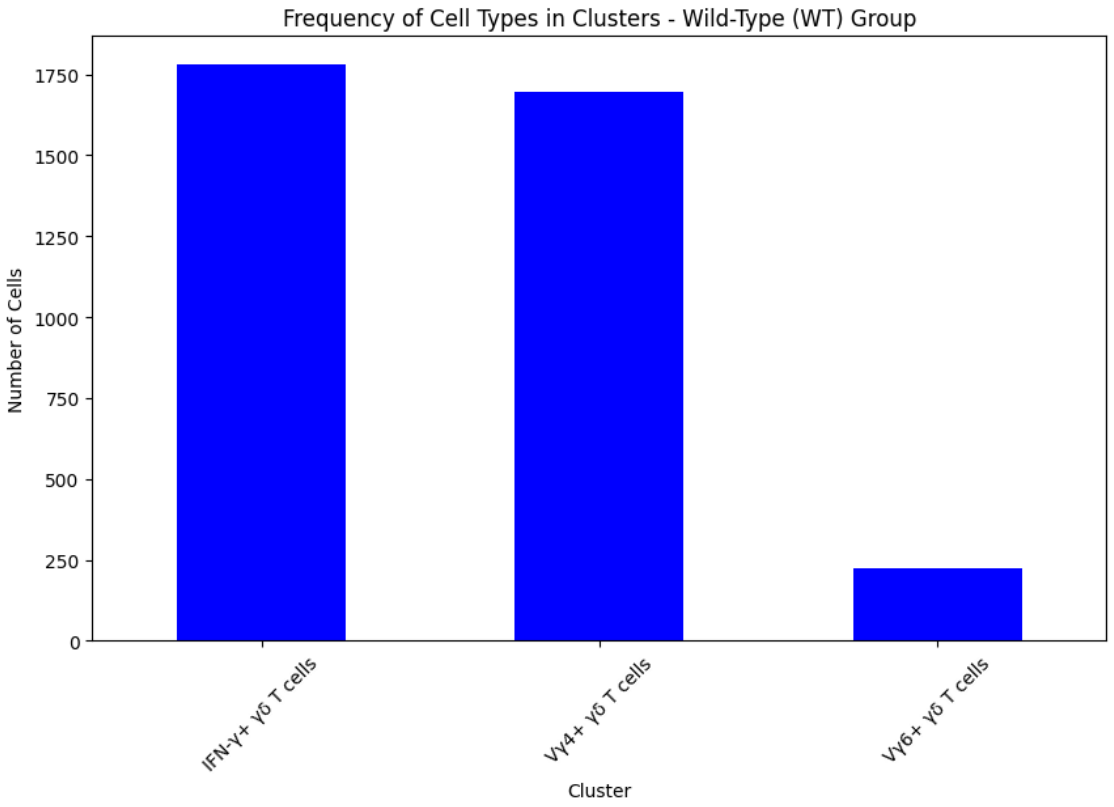
[ ]:
```
'''
# EQUIVALENT R CODE

# Plot UMAP with specific gene expression coloring

DimPlot(merged, reduction = "umap", cols = c("Cd163l1", "Ccl5",␣
 ↪"5830411N06Rik"))

# Plot violin plots for the top differentially expressed genes in cluster 0

VlnPlot(subset_data, features = c("Cd163l1", "Ccl5", "5830411N06Rik"), group.by␣
 ↪= "leiden", group.colors = brewer.pal(8, "Set1"))
'''
```

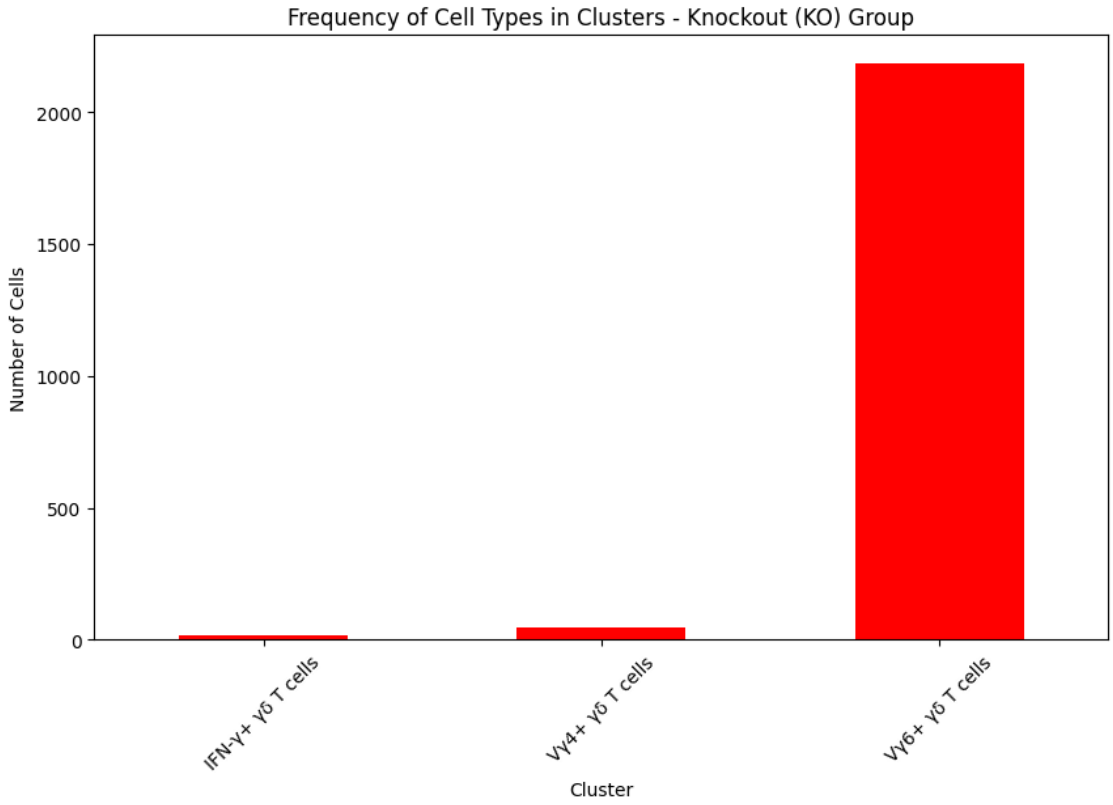
[35]:
```
sc.pl.violin(merged, ["Cd163l1", "Ccl5", "5830411N06Rik"], groupby="leiden")
```



[36]:
```
import matplotlib.pyplot as plt

# Subsetting data for WT and KO groups
wt_samples = subset_data[subset_data.obs['batch'] != 'KO1'].copy()
ko_samples = subset_data[subset_data.obs['batch'] == 'KO1'].copy()

# Calculate cluster frequencies for WT and KO groups
wt_cluster_freq = wt_samples.obs['leiden'].value_counts()
ko_cluster_freq = ko_samples.obs['leiden'].value_counts()

# Define cluster names based on your annotation
cluster_names = {
    '0': 'V 6+   T cells',
    '1': 'IFN- +   T cells',
    '2': 'V 4+   T cells'
}

# Plot frequency of cell types in clusters for WT group
plt.figure(figsize=(10, 6))
wt_cluster_freq.rename(index=cluster_names).sort_index().plot(kind='bar',␣
 ↪color='blue')
plt.xlabel('Cluster')
plt.ylabel('Number of Cells')
plt.title('Frequency of Cell Types in Clusters - Wild-Type (WT) Group')
plt.xticks(rotation=45)
plt.show()

# Plot frequency of cell types in clusters for KO group
plt.figure(figsize=(10, 6))
ko_cluster_freq.rename(index=cluster_names).sort_index().plot(kind='bar',␣
 ↪color='red')
plt.xlabel('Cluster')
plt.ylabel('Number of Cells')
plt.title('Frequency of Cell Types in Clusters - Knockout (KO) Group')
plt.xticks(rotation=45)
```

```
plt.show()
```


Frequency of Cell Types in Clusters - Wild-Type (WT) Group
Number of Cells
0
250
500
750
1000
1250
1500
1750
IFN-γ+ γδ T cells
Vγ4+ γδ T cells
Vγ6+ γδ T cells
Cluster



[ ]:
```
'''
# EQUIVALENT R CODE

freq.table.group <- as.data.frame(prop.table(x = table(Idents(subset_data),␣
 ↪subset_data$Status), margin = 2))


ggplot(data=freq.table.group, aes(x=freq.table.group$Var2, y = freq.table.
 ↪group$Freq, fill=freq.table.group$Var1)) +

geom_bar(stat="identity",color="black") +

labs(x="Group", y="Proportion of cells", fill="Cell Type") +␣
 ↪scale_x_discrete(limits = rev(levels(freq.table.group$Var2)))

'''
```

[37]:
```
# Identify the largest cluster containing cells from both KO and WT groups
largest_cluster = merged.obs['leiden'].value_counts().idxmax()

# Subset the data to include only cells from the largest cluster
```

```
cluster_subset = merged[merged.obs['leiden'] == largest_cluster].copy()
```

[38]:
```
# Perform differential expression analysis comparing gene expression between␣
 ↪KO1 and WT cells
sc.tl.rank_genes_groups(cluster_subset, 'batch', groups=['KO1'],␣
 ↪reference='rest', method='wilcoxon')

# Access the differential expression results
de_results = cluster_subset.uns['rank_genes_groups']
```

[39]:
```
de_genes = de_results['names']['KO1']
de_pvals = de_results['pvals_adj']['KO1']
de_logFC = de_results['logfoldchanges']['KO1']

# Identify up-regulated genes in the KO1 group
upregulated_genes_ko1 = [gene for gene, pval, logFC in zip(de_genes, de_pvals,␣
 ↪de_logFC)
                         if pval < 0.01 and logFC > 0.5]

# Count the number of up-regulated genes in the KO1 group
num_upregulated_genes_ko1 = len(upregulated_genes_ko1)

print("Number of genes up-regulated in KO1 with corrected p-values <0.01 and␣
 ↪logFC >0.5:", num_upregulated_genes_ko1)
```

```
Number of genes up-regulated in KO1 with corrected p-values <0.01 and logFC
>0.5: 275
```

[ ]:
```
'''
# EQUIVALENT R CODE

# Performing differential expression analysis between KO1 and combined WT␣
 ↪samples

cluster_subset <- subset(merged, subset = batch == "KO1")

# Run differential expression analysis

de_results <- FindMarkers(cluster_subset, ident.1 = "KO1", ident.2 = NULL, test.
 ↪use = "wilcox", logfc.threshold = 0.5, min.pct = 0.1)

# Extract up-regulated genes in the KO group

upregulated_genes_ko <- de_results$gene[de_results$p_val_adj < 0.01 &␣
 ↪de_results$log2fc > 0.5]

# Count the number of up-regulated genes in the KO group
```

```
num_upregulated_genes_ko <- length(upregulated_genes_ko)
'''
```

[40]:
```
pd.DataFrame(de_results['names'])
```

[40]:
```
          KO1
0       Klrk1
1        Ddx5
2       Rpl10
3      mt-Nd4
4      Hnrnpk
…       …
15884  Rpl37a
15885   Rpl38
15886   Rpl37
15887   Rpl41
15888   Rps28

[15889 rows x 1 columns]
```

[41]:
```
# HARMONY INTEGRATION
h_merged = sc.read_h5ad('mfile')
```

[42]:
```
# Step 1: Perform integration using Harmony
# ,basis='X_pca'
sce.pp.harmony_integrate(h_merged,'batch')
```

```
2024-03-25 03:58:29,781 - harmonypy - INFO - Computing initial centroids with
sklearn.KMeans…
2024-03-25 03:58:40,301 - harmonypy - INFO - sklearn.KMeans initialization
complete.
2024-03-25 03:58:40,348 - harmonypy - INFO - Iteration 1 of 10
2024-03-25 03:58:43,680 - harmonypy - INFO - Iteration 2 of 10
2024-03-25 03:58:47,090 - harmonypy - INFO - Iteration 3 of 10
2024-03-25 03:58:50,528 - harmonypy - INFO - Iteration 4 of 10
2024-03-25 03:58:53,545 - harmonypy - INFO - Iteration 5 of 10
2024-03-25 03:58:55,779 - harmonypy - INFO - Converged after 5 iterations
```

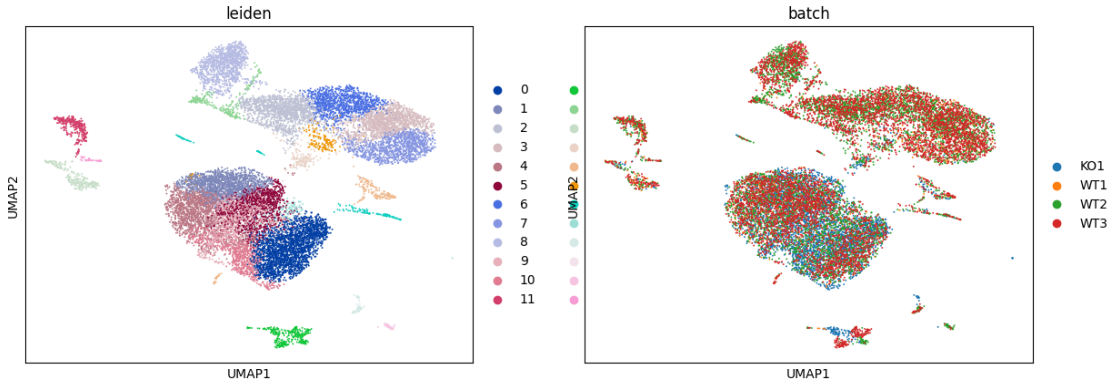
[43]:
```
h_merged.obsm['X_pca'] = h_merged.obsm['X_pca_harmony']
```

[44]:
```
sc.pp.neighbors(h_merged, n_neighbors=10, n_pcs=20)
sc.tl.umap(h_merged)
```

[45]:
```
sc.tl.leiden(h_merged,resolution = 1)
```

[46]:
```
sc.pl.umap(h_merged, color=['leiden','batch'])
```

```
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
```



[ ]:
```
'''
# EQUIVALENT R CODE

library(harmony)

h_merged <- NormalizeData(h_merged) %>% FindVariableFeatures() %>% ScaleData()␣
 ↪%>% RunPCA(verbose = FALSE)

h_merged <- RunHarmony(h_merged, group.by.vars = "orig.ident")

h_merged <- RunUMAP(h_merged, reduction = "harmony", dims = 1:20)

h_merged <- FindNeighbors(h_merged, reduction = "harmony", dims = 1:20) %>%␣
 ↪FindClusters()

DimPlot(h_merged, group.by = "orig.ident",  ncol = 3)
'''
```

[47]:
```
sc.pp.log1p(h_merged)
cluster_size_h = h_merged.obs['leiden'].value_counts()
largest_cluster_h = cluster_size_h.nlargest(1).index.tolist()
cluster_subset_h = h_merged[h_merged.obs['leiden'].isin(largest_cluster_h)].
 ↪copy()
```

```
WARNING: adata.X seems to be already log-transformed.
```

```
/usr/local/lib/python3.10/dist-packages/scanpy/preprocessing/_simple.py:352:
RuntimeWarning: invalid value encountered in log1p
  np.log1p(X, out=X)
```

[48]:
```
# Step 2: Perform differential expression analysis comparing gene expression␣
 ↪between KO and WT cells
sc.tl.rank_genes_groups(cluster_subset_h, 'batch', groups=['KO1'],␣
 ↪method='wilcoxon')

# Step 3: Access the differential expression results
de_results = cluster_subset_h.uns['rank_genes_groups']

# Filter genes based on corrected p-values and log fold change for KO1 group

de_genes = de_results['names']['KO1']
de_pvals = de_results['pvals_adj']['KO1']
de_logFC = de_results['logfoldchanges']['KO1']

# Identify up-regulated genes in the KO group
upregulated_genes_ko = [gene for gene, pval, logFC in zip(de_genes, de_pvals,␣
 ↪de_logFC)
                        if pval < 0.01 and logFC > 0.5]

# Count the number of up-regulated genes in the KO group
num_upregulated_genes_ko = len(upregulated_genes_ko)

print("Number of genes up-regulated in the KO with corrected p-values < 0.01␣
 ↪and logFC > 0.5 (using Harmony integration):", num_upregulated_genes_ko)
```

```
Number of genes up-regulated in the KO with corrected p-values < 0.01 and logFC
> 0.5 (using Harmony integration): 1470
```

[49]:
```
pd.DataFrame(de_results['names'])
```

[49]:

	KO1
0	Klrk1
1	AC163354.1
2	Crip1
3	Itgb1
4	Cd47
…	…
15884	Rpl37
15885	Rps24
15886	Rpl37a
15887	Rpl38
15888	Rps28

```
[15889 rows x 1 columns]
```

[50]:
```
results= cluster_subset_h.uns['rank_genes_groups']
out= np.array([[0, 0, 0, 0, 0]])
for group in results['names'].dtype.names:
    out= np.vstack((out, np.vstack((results['names'][group],
                                    results['scores'][group],
                                    results['pvals_adj'][group],
                                    results['logfoldchanges'][group],
                                    np.
 ↪array([group]*len(results['names'][group])).astype('object'))).T))
markers= pd.DataFrame(out[1:], columns= ['Gene', 'scores', 'pval_adj', 'lfc',␣
 ↪'cluster'])
markers = markers[(markers['pval_adj'] < 0.01) & (markers['lfc'] > 0.5)]
```

[51]:
```
markers
```

[51]:
```
            Gene     scores  pval_adj       lfc cluster
0          Klrk1  29.967012       0.0  2.728586     KO1
1      AC163354.1  29.726397       0.0  3.182883     KO1
2          Crip1  29.622301       0.0   1.73646     KO1
3          Itgb1  24.571079       0.0  3.420422     KO1
4           Cd47  24.542936       0.0   1.76682     KO1
…           …         …       …        …      …
1569       Copz1   3.25732  0.009851  0.629711     KO1
1570        Miip  3.255506  0.009909  0.965825     KO1
1571      Ube2j2  3.255081  0.009918  0.543831     KO1
1572     Slc35b1  3.254712  0.009926  0.858133     KO1
1573       Itpk1  3.254542  0.009926  1.266832     KO1

[1470 rows x 5 columns]
```

[52]:
```
#PSEUDOBULK

pdata = dc.get_pseudobulk(
    cluster_subset,
    sample_col='batch',
    groups_col='leiden',
    layer='counts',
    mode='sum',
    min_cells=10,
    min_counts=1000
)

dc.plot_psbulk_samples(pdata, groupby=['batch', 'leiden'], figsize=(12, 4))
dc.plot_filter_by_expr(pdata, group='batch', min_count=10, min_total_count=15)
```

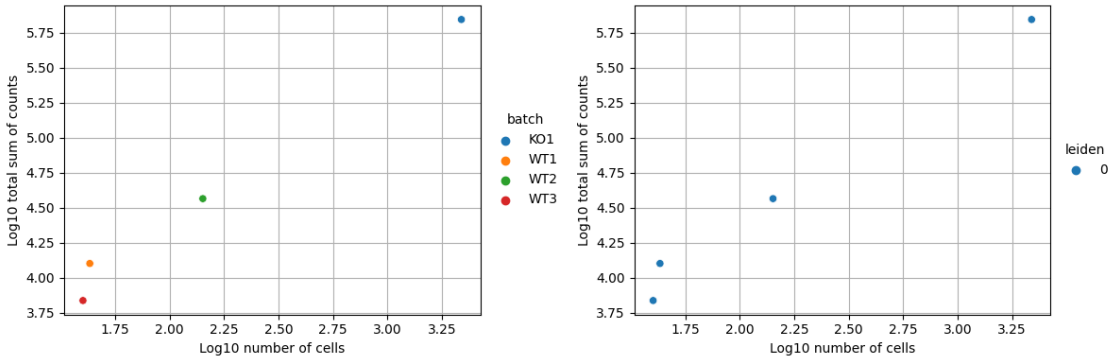
```python
# Build DESeq2 object
inference = DefaultInference(n_cpus=8)
dds = DeseqDataSet(
    adata=pdata,
    design_factors='condition',
    ref_level=['condition', 'WT'],
    refit_cooks=True,
    inference=inference,
)

# Compute LFCs
dds.deseq2()

# Extract contrast between COVID-19 vs normal
stat_res = DeseqStats(
    dds,
    contrast=["condition", 'KO', 'WT'],
    inference=inference,
)

stat_res.summary()

results_df = stat_res.results_df
results_df
```

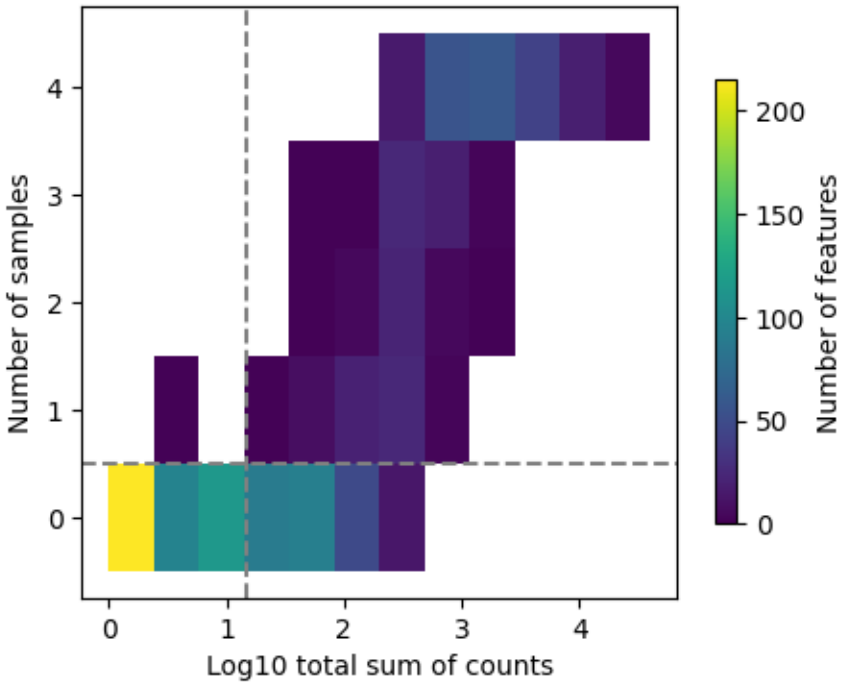
```
Fitting size factors…
… done in 0.00 seconds.

Fitting dispersions…
… done in 0.41 seconds.

Fitting dispersion trend curve…
… done in 0.05 seconds.

/home1/bioinfo-26/.local/lib/python3.10/site-packages/pydeseq2/dds.py:442:
UserWarning: As the residual degrees of freedom is less than 3, the distribution
of log dispersions is especially asymmetric and likely to be poorly estimated by
the MAD.
  self.fit_dispersion_prior()
Fitting MAP dispersions…
… done in 0.28 seconds.

Fitting LFCs…
… done in 0.25 seconds.

Replacing 0 outlier genes.
```

```
Running Wald tests…

Log2 fold change & Wald test p-value: condition KO vs WT
                 baseMean  log2FoldChange     lfcSE      stat    pvalue  \
1500009L16Rik  136.713806       -0.197608  1.321901 -0.149488  0.881169
1700055D18Rik    0.138730        0.240733  4.341970  0.055443  0.955785
2310001H17Rik   25.468281       -0.159767  1.315200 -0.121477  0.903313
2900026A02Rik    0.124857        0.088731  4.483132  0.019792  0.984209
5830411N06Rik   35.163712       -4.738309  1.439340 -3.292001  0.000995
…                     …              …         …         …         …
Zbtb7a          22.448473        0.064577  1.307784  0.049379  0.960617
Zeb2            16.177872        1.268040  0.355644  3.565472  0.000363
Zfp36           63.967201       -0.009519  1.351177 -0.007045  0.994379
Zfp709           2.513373        1.022839  2.268324  0.450923  0.652045
Zkscan17         2.704908       -0.212667  1.762723 -0.120647  0.903971

                   padj
1500009L16Rik  0.997137
1700055D18Rik  0.997137
2310001H17Rik  0.997137
2900026A02Rik  0.997137
5830411N06Rik  0.117322
…                   …
Zbtb7a         0.997137
Zeb2           0.057138
Zfp36          0.998376
Zfp709         0.997137
Zkscan17       0.997137

[999 rows x 6 columns]

… done in 0.19 seconds.
```

[52]:
```
                 baseMean  log2FoldChange     lfcSE      stat    pvalue  \
1500009L16Rik  136.713806       -0.197608  1.321901 -0.149488  0.881169
1700055D18Rik    0.138730        0.240733  4.341970  0.055443  0.955785
2310001H17Rik   25.468281       -0.159767  1.315200 -0.121477  0.903313
2900026A02Rik    0.124857        0.088731  4.483132  0.019792  0.984209
5830411N06Rik   35.163712       -4.738309  1.439340 -3.292001  0.000995
…                     …              …         …         …         …
Zbtb7a          22.448473        0.064577  1.307784  0.049379  0.960617
Zeb2            16.177872        1.268040  0.355644  3.565472  0.000363
Zfp36           63.967201       -0.009519  1.351177 -0.007045  0.994379
Zfp709           2.513373        1.022839  2.268324  0.450923  0.652045
Zkscan17         2.704908       -0.212667  1.762723 -0.120647  0.903971

                   padj
```

```
1500009L16Rik  0.997137
1700055D18Rik  0.997137
2310001H17Rik  0.997137
2900026A02Rik  0.997137
5830411N06Rik  0.117322
…                 …
Zbtb7a         0.997137
Zeb2           0.057138
Zfp36          0.998376
Zfp709         0.997137
Zkscan17       0.997137

[999 rows x 6 columns]
```





[53]:
```
#PSEUDOBULK

pdata_h = dc.get_pseudobulk(
    cluster_subset_h,
    sample_col='batch',
    groups_col='leiden',
    layer='counts',
    mode='sum',
    min_cells=10,
    min_counts=1000
)

dc.plot_psbulk_samples(pdata_h, groupby=['batch', 'leiden'], figsize=(12, 4))
dc.plot_filter_by_expr(pdata_h, group='batch', min_count=10, min_total_count=15)

# Build DESeq2 object
inference = DefaultInference(n_cpus=8)
dds_h = DeseqDataSet(
    adata=pdata_h,
    design_factors='condition',
    ref_level=['condition', 'WT'],
    refit_cooks=True,
    inference=inference,
)

# Compute LFCs
dds_h.deseq2()

# Extract contrast between COVID-19 vs normal
stat_res_h = DeseqStats(
    dds_h,
    contrast=["condition", 'KO', 'WT'],
    inference=inference,
)

stat_res_h.summary()

results_df_h = stat_res_h.results_df
results_df_h
```

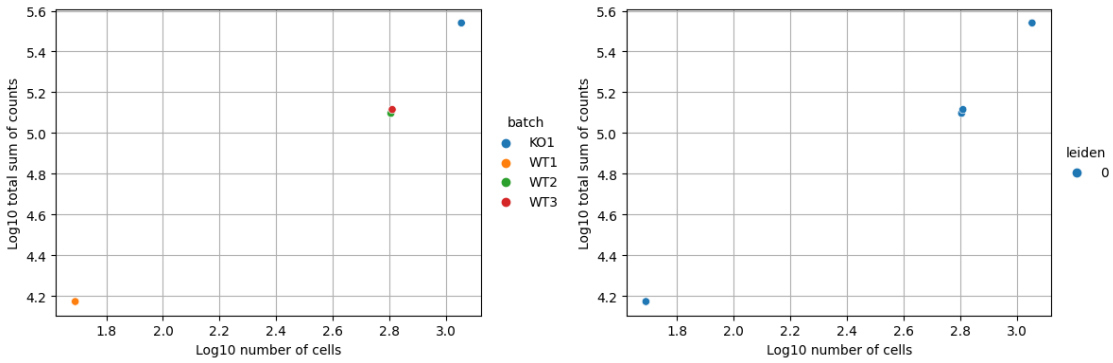
```
Fitting size factors…
… done in 0.00 seconds.

Fitting dispersions…
… done in 0.48 seconds.
```

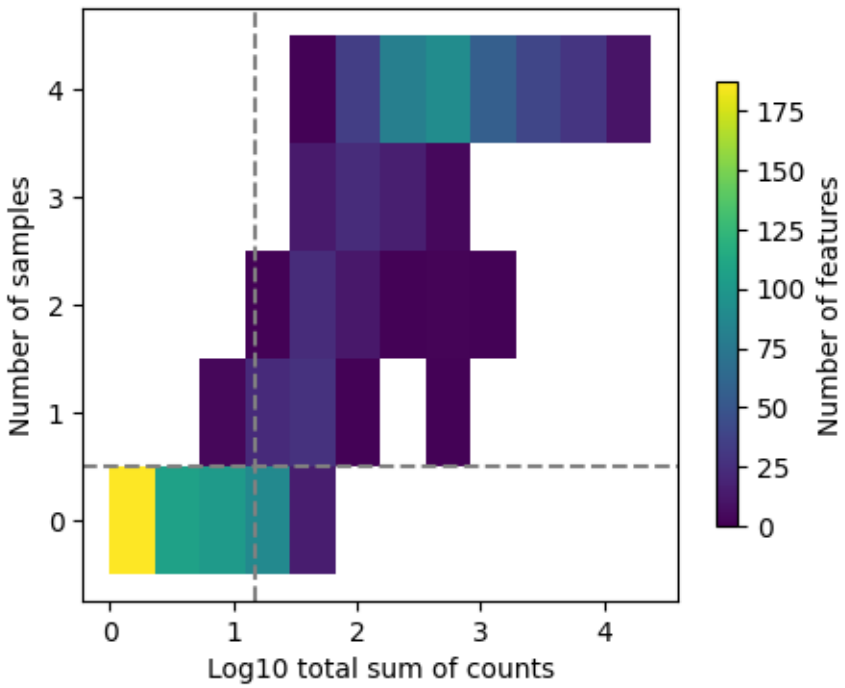
```
Fitting dispersion trend curve…
… done in 0.05 seconds.

/home1/bioinfo-26/.local/lib/python3.10/site-packages/pydeseq2/dds.py:442:
UserWarning: As the residual degrees of freedom is less than 3, the distribution
of log dispersions is especially asymmetric and likely to be poorly estimated by
the MAD.
  self.fit_dispersion_prior()
Fitting MAP dispersions…
… done in 0.27 seconds.

Fitting LFCs…
… done in 0.23 seconds.

Replacing 0 outlier genes.

Running Wald tests…

Log2 fold change & Wald test p-value: condition KO vs WT
                 baseMean  log2FoldChange     lfcSE      stat    pvalue  \
1500009L16Rik  111.914749        1.136082  1.172412  0.969013  0.332539
1700055D18Rik    2.075713       -1.869665  1.847549 -1.011970  0.311552
2310001H17Rik   68.542564       -0.757807  0.618945 -1.224352  0.220819
2900026A02Rik    0.851600        3.635296  3.568999  1.018576  0.308404
5830411N06Rik   11.548050       -8.105703  3.371351 -2.404289  0.016204
…                     …              …         …         …         …
Zeb2            32.779896        2.963882  0.890258  3.329239  0.000871
Zfhx3            0.186906       -3.403427  6.434639 -0.528923  0.596859
Zfp36          384.375458       -0.912059  0.541023 -1.685805  0.091833
Zfp709           7.049066       -0.165971  1.627468 -0.101981  0.918772
Zkscan17         3.510900        0.446437  1.501158  0.297395  0.766165

                   padj
1500009L16Rik  0.660830
1700055D18Rik       NaN
2310001H17Rik  0.551720
2900026A02Rik       NaN
5830411N06Rik  0.105776
…                   …
Zeb2           0.011554
Zfhx3               NaN
Zfp36          0.342553
Zfp709         0.980679
Zkscan17            NaN

[987 rows x 6 columns]

… done in 0.23 seconds.
```

[53]:

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
1500009L16Rik	111.914749	1.136082	1.172412	0.969013	0.332539	0.660830
1700055D18Rik	2.075713	-1.869665	1.847549	-1.011970	0.311552	NaN
2310001H17Rik	68.542564	-0.757807	0.618945	-1.224352	0.220819	0.551720
2900026A02Rik	0.851600	3.635296	3.568999	1.018576	0.308404	NaN
5830411N06Rik	11.548050	-8.105703	3.371351	-2.404289	0.016204	0.105776
…	…	…	…	…	…	…
Zeb2	32.779896	2.963882	0.890258	3.329239	0.000871	0.011554
Zfhx3	0.186906	-3.403427	6.434639	-0.528923	0.596859	NaN
Zfp36	384.375458	-0.912059	0.541023	-1.685805	0.091833	0.342553
Zfp709	7.049066	-0.165971	1.627468	-0.101981	0.918772	0.980679
Zkscan17	3.510900	0.446437	1.501158	0.297395	0.766165	NaN

[987 rows x 6 columns]





[ ]:
```
'''
# EQUIVALENT R CODE
# Get pseudo-bulk profile

pdata <- Seurat::AggregateExpression(

  object = cluster_subset,

  features = rownames(cluster_subset),

  group.by = c("batch", "orig.ident"),

  assay = "RNA",

  agg.func = "sum"
)

# Filter cells by expression

pdata_filtered <- Seurat::subset(

  x = pdata,

  cells = which(rowSums(pdata) >= 15)
```

```
)

# Build DESeq2 object

dds <- DESeq2::DESeqDataSetFromMatrix(

  countData = assay(pdata_filtered, "RNA"),

  colData = NULL,

  design = ~ batch
)

dds <- DESeq2::DESeq(dds)

# Extract contrast between KO and WT

res <- DESeq2::results(dds, contrast = c("batch", "KO", "WT"))

results_df <- as.data.frame(res)
'''
```

[54]:
```
# Filter results dataframe to find upregulated genes in KO1
pb_upregulated_genes_ko1_h = results_df_h[(results_df_h['log2FoldChange'] > 0.
  ↪5) & (results_df_h['padj'] < 0.05) ]

# Get the number of upregulated genes in KO1
num_upregulated_genes_ko1_h = len(pb_upregulated_genes_ko1_h)

print("Number of upregulated genes in KO1:", num_upregulated_genes_ko1_h)
```

```
Number of upregulated genes in KO1: 18
```

[55]:
```
# Filter results dataframe to find upregulated genes in KO1
pb_upregulated_genes_ko1 = results_df[(results_df['log2FoldChange'] > 0) &␣
  ↪(results_df['padj'] < 0.05) ]

# Get the number of upregulated genes in KO1
num_upregulated_genes_ko1 = len(pb_upregulated_genes_ko1)

print("Number of upregulated genes in KO1:", num_upregulated_genes_ko1)
```

```
Number of upregulated genes in KO1: 2
```

[56]:
```
# Filter results_df_h to remove rows with padj > 0.5
results_df_h_filtered = results_df_h[results_df_h['padj'] <= 0.5]
```

```
# Display the filtered DataFrame
results_df_h_filtered
```

[56]:
```
                  baseMean  log2FoldChange     lfcSE       stat        pvalue  \
5830411N06Rik    11.548050       -8.105703  3.371351  -2.404289  1.620395e-02
AC160336.1      766.690063       -1.222201  0.477954  -2.557150  1.055337e-02
AC163354.1      362.467712        2.227595  0.220082  10.121663  4.428258e-24
AW112010        370.289368       -0.892577  0.590471  -1.511635  1.306268e-01
Abi3bp          257.248383        1.729048  0.920925   1.877512  6.044794e-02
…                      …              …         …         …           …
Txnip           222.034454        0.906060  0.310878   2.914523  3.562329e-03
Vim            1148.858887        0.814147  0.543686   1.497458  1.342741e-01
Zbtb20           92.205925       -1.304023  0.719450  -1.812529  6.990454e-02
Zeb2             32.779896        2.963882  0.890258   3.329239  8.708376e-04
Zfp36           384.375458       -0.912059  0.541023  -1.685805  9.183336e-02

                       padj
5830411N06Rik  1.057758e-01
AC160336.1     7.294239e-02
AC163354.1     1.040641e-21
AW112010       4.148285e-01
Abi3bp         2.688688e-01
…                       …
Txnip          3.348589e-02
Vim            4.161511e-01
Zbtb20         2.906811e-01
Zeb2           1.155399e-02
Zfp36          3.425530e-01

[175 rows x 6 columns]
```

[57]:
```
# Sort results_df_h by 'padj' column in ascending order
results_df_h_sorted = results_df_h_filtered.sort_values(by='log2FoldChange',␣
 ↪ascending=False)

# Display the sorted DataFrame
results_df_h_sorted.head(70)
```

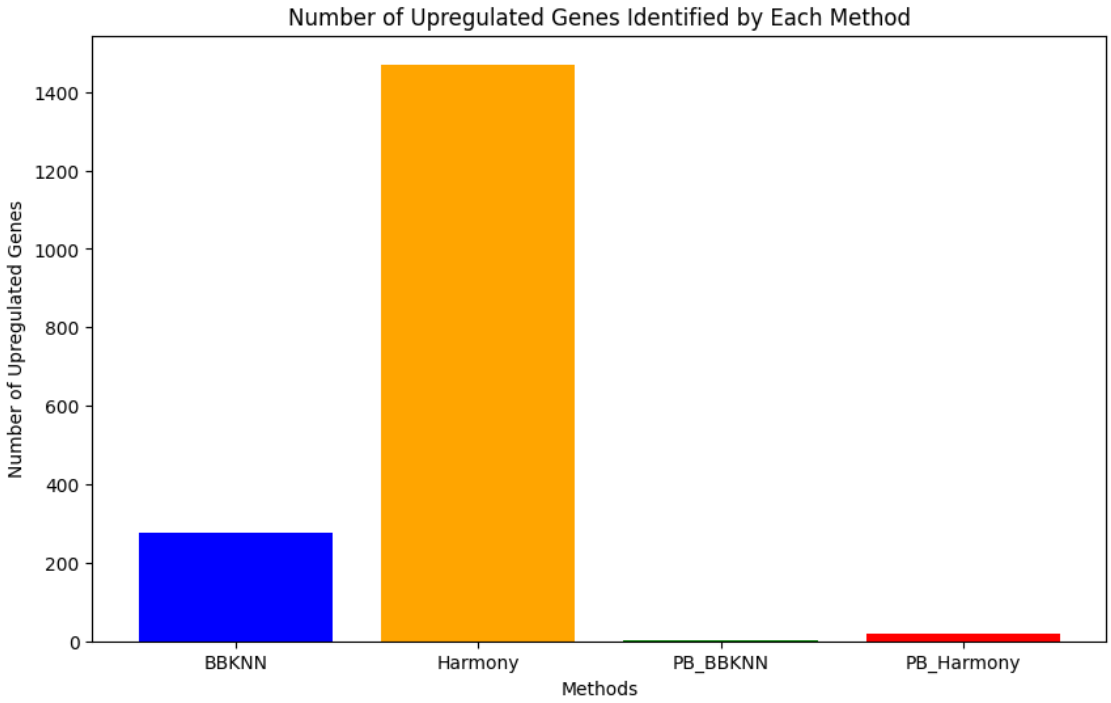
[57]:
```
              baseMean  log2FoldChange     lfcSE      stat        pvalue  \
Klrc1        56.467621        4.860837  0.832172  5.841145  5.184322e-09
Klrc2        31.641348        4.714959  0.806865  5.843550  5.109991e-09
Dtx1         44.448593        3.641707  0.832760  4.373058  1.225184e-05
Sgip1        34.562977        3.415121  1.042868  3.274740  1.057592e-03
Mir155hg     20.601807        3.192055  0.983927  3.244199  1.177813e-03
…                  …              …         …         …           …
Csrnp1      102.802795        0.538222  0.303621  1.772675  7.628253e-02
Lgals1     2816.129883        0.520345  0.351092  1.482074  1.383206e-01
```

```
Furin      329.313538        0.475303  0.290440  1.636492  1.017366e-01
Prkcd       25.646336        0.435794  0.222820  1.955807  5.048790e-02
Cd44       421.324371        0.412259  0.082481  4.998238  5.785669e-07

                 padj
Klrc1    3.045789e-07
Klrc2    3.045789e-07
Dtx1     3.199091e-04
Sgip1    1.242671e-02
Mir155hg 1.287376e-02
…                …
Csrnp1   3.011142e-01
Lgals1   4.249065e-01
Furin    3.636569e-01
Prkcd    2.421359e-01
Cd44     2.266054e-05

[70 rows x 6 columns]
```

[58]:
```
# Filter results_df_h to remove rows with padj > 0.5
results_df_filtered = results_df[results_df['padj'] <= 0.05]

# Display the filtered DataFrame
results_df_filtered

# Sort results_df_h by 'padj' column in ascending order
results_df_sorted = results_df_filtered.sort_values(by='log2FoldChange',␣
 ↪ascending=False)

# Display the sorted DataFrame
results_df_sorted.head(70)
```

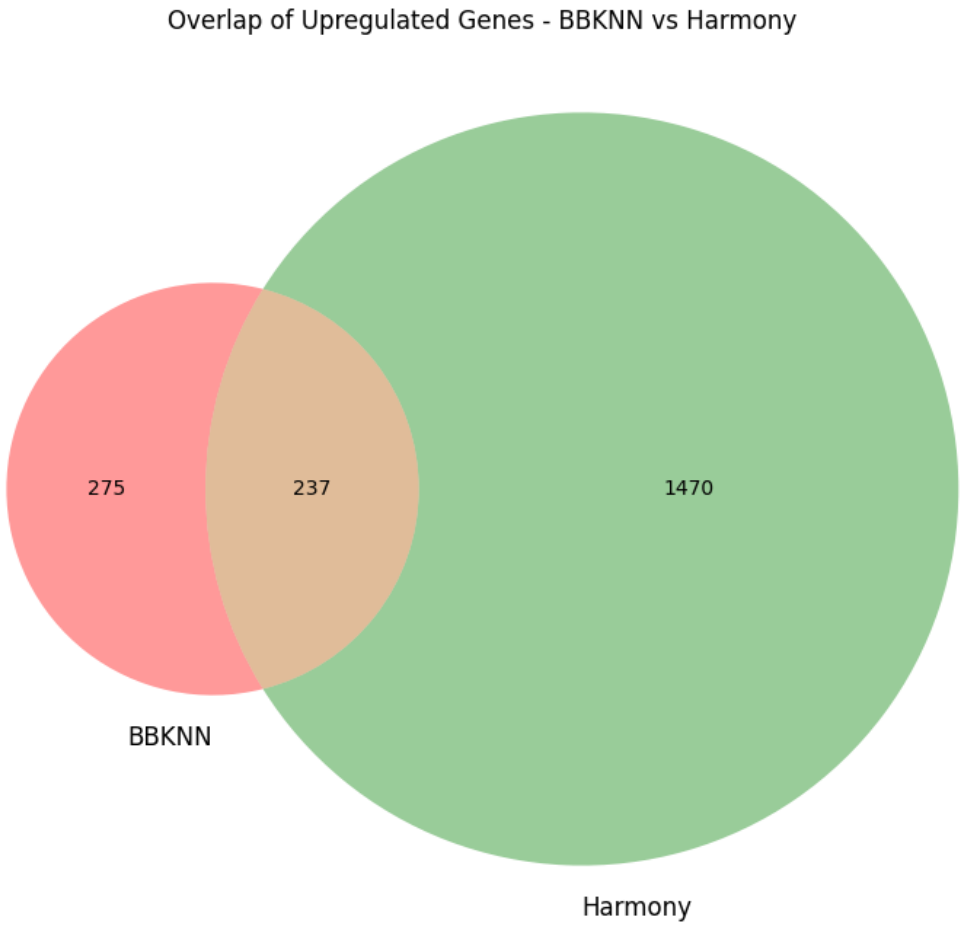
[58]:
```
                 baseMean  log2FoldChange     lfcSE      stat        pvalue  \
Sgip1           10.598842        4.734001  1.006120  4.705206  2.536103e-06
Lmna           216.492020        0.730526  0.090556  8.067128  7.197104e-16
Ltb            739.799805       -0.178925  0.045668 -3.917948  8.930581e-05
B930036N10Rik   25.072521       -2.023115  0.321888 -6.285158  3.275207e-10
Itgb2           42.082130       -6.960996  1.303531 -5.340109  9.289050e-08

                      padj
Sgip1          6.333917e-04
Lmna           7.189907e-13
Ltb            1.784330e-02
B930036N10Rik  1.635966e-07
Itgb2          3.093254e-05
```

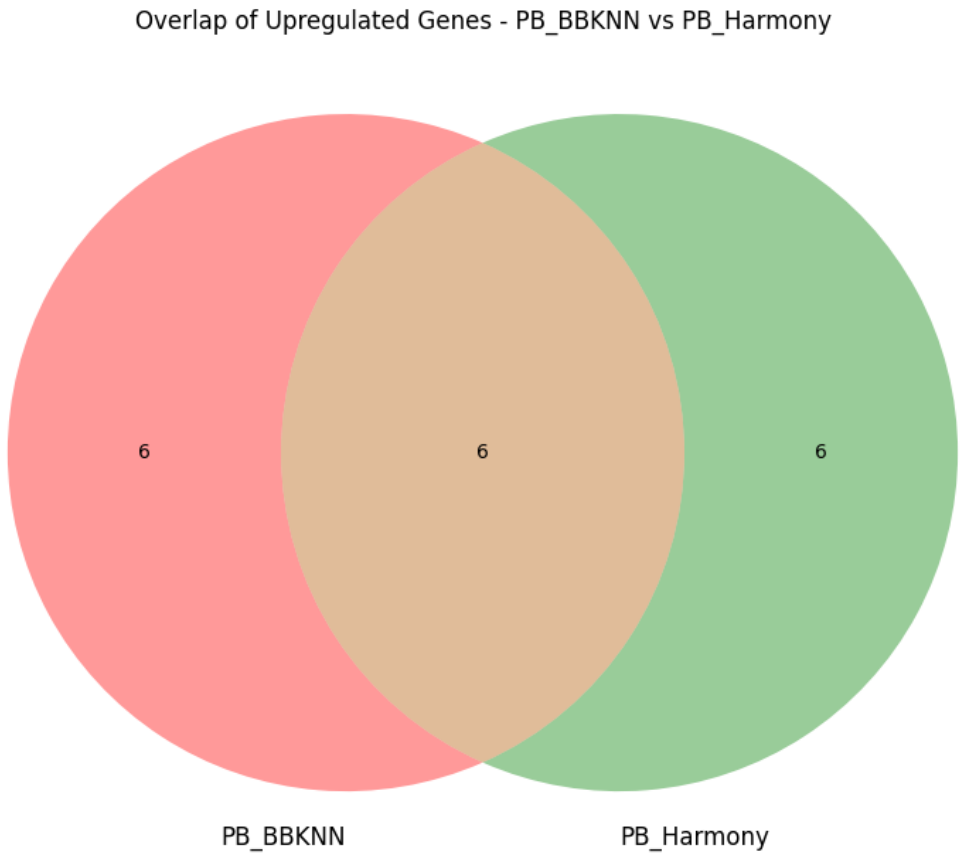
[59]:
```
import matplotlib.pyplot as plt

# Get the counts of upregulated genes for each method
counts = {
    'BBKNN': len(upregulated_genes_ko1),
    'Harmony': len(upregulated_genes_ko),
    'PB_BBKNN': len(pb_upregulated_genes_ko1),
    'PB_Harmony': len(pb_upregulated_genes_ko1_h)
}

# Create a bar plot
plt.figure(figsize=(10, 6))
plt.bar(counts.keys(), counts.values(), color=['blue', 'orange', 'green',␣
 ↪'red'])
plt.xlabel('Methods')
plt.ylabel('Number of Upregulated Genes')
plt.title('Number of Upregulated Genes Identified by Each Method')
plt.show()
```



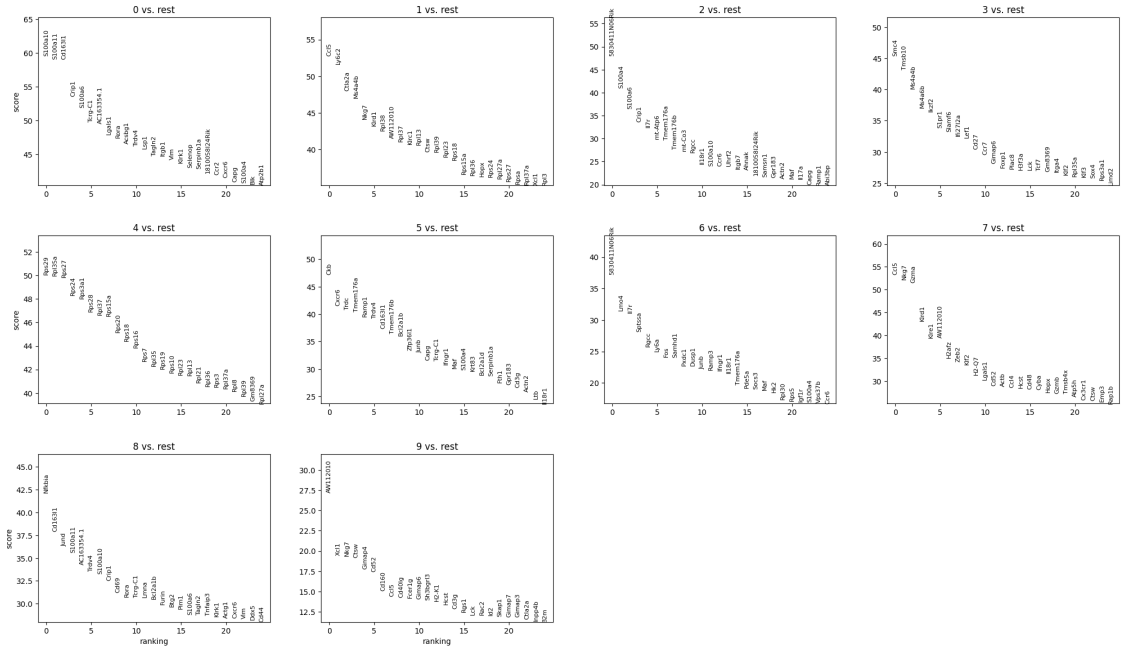
[60]:
```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2

# Convert lists to sets for Venn diagram
set_d = set(upregulated_genes_ko1)
```

```
set_e = set(upregulated_genes_ko)
set_f = set(pb_upregulated_genes_ko1)
set_g = set(pb_upregulated_genes_ko1_h)

# Calculate the overlaps between sets
overlap_de = set_d.intersection(set_e)
overlap_df = set_d.intersection(set_f)
overlap_dg = set_d.intersection(set_g)
overlap_ef = set_e.intersection(set_f)
overlap_eg = set_e.intersection(set_g)
overlap_fg = set_f.intersection(set_g)
overlap_def = set_d.intersection(set_e, set_f)
overlap_deg = set_d.intersection(set_e, set_g)
overlap_dfg = set_d.intersection(set_f, set_g)
overlap_efg = set_e.intersection(set_f, set_g)

# Create Venn diagram for BBKNN and Harmony
plt.figure(figsize=(10, 8))
venn2(subsets=(len(set_d), len(set_e), len(overlap_de)),
      set_labels=('BBKNN', 'Harmony'))
plt.title('Overlap of Upregulated Genes - BBKNN vs Harmony')
plt.show()

# Create Venn diagram for PB_BBKNN and PB_HARMONY
plt.figure(figsize=(10, 8))
venn2(subsets=(len(set_f), len(set_g), len(overlap_fg)),
      set_labels=('PB_BBKNN', 'PB_Harmony'))
plt.title('Overlap of Upregulated Genes - PB_BBKNN vs PB_Harmony')
plt.show()
```

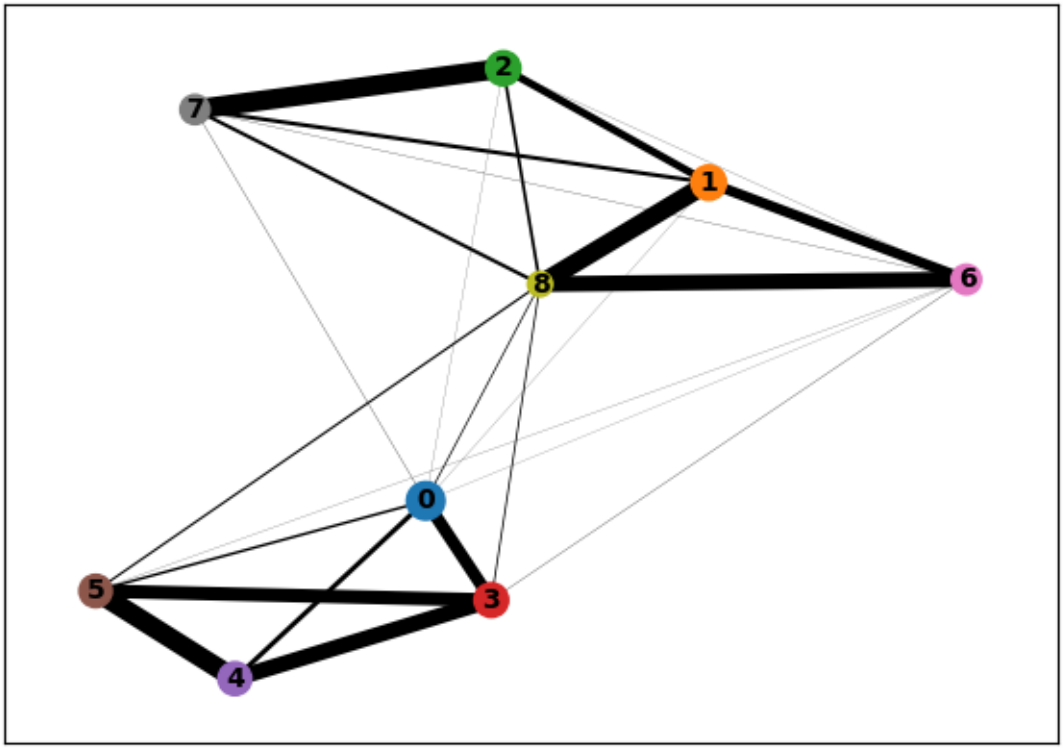

Overlap of Upregulated Genes - BBKNN vs Harmony
275
237
1470
BBKNN
Harmony



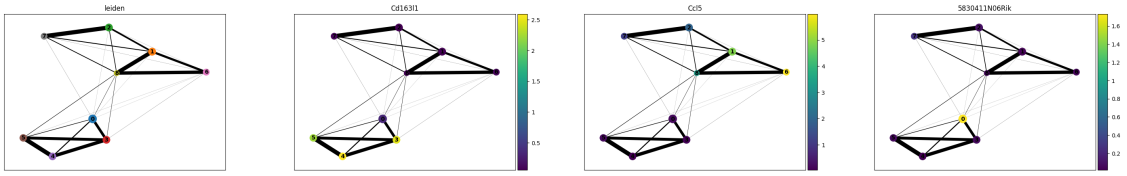
[61]:
```
# PSEUDOTIME ANALYSIS
cluster_size_p = merged.obs['leiden'].value_counts()
largest_cluster_p = cluster_size_p.nlargest(10).index.tolist()
cluster_subset_p = merged[merged.obs['leiden'].isin(largest_cluster_p)].copy()
# Identify marker genes
cluster_subset_p.uns['log1p']['base'] = 10
sc.tl.rank_genes_groups(cluster_subset_p, groupby='leiden', method='wilcoxon',␣
 ↪use_raw=True)

# Visualize marker genes
sc.pl.rank_genes_groups(cluster_subset_p, n_genes=25, sharey=False)  # Adjust␣
 ↪n_genes as needed
```



[63]:
```
sc.tl.leiden(cluster_subset_p, resolution=1)
sc.tl.paga(cluster_subset_p, groups="leiden")
sc.pl.paga(cluster_subset_p, color=["leiden"])
```



[76]:
```
# TRAJECTORY ANALYSIS
sc.pl.paga(cluster_subset_p, color=["leiden", "Cd163l1", "Ccl5",␣
↪"5830411N06Rik"])
```

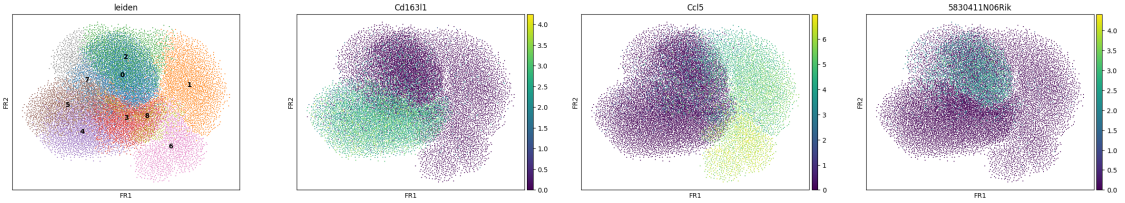


[74]:
```
sc.tl.draw_graph(cluster_subset_p, init_pos="paga")
```

```
WARNING: Package 'fa2' is not installed, falling back to layout 'fr'.To use the
faster and better ForceAtlas2 layout, install package 'fa2' (`pip install fa2`).
```

[77]:
```
sc.pl.draw_graph(
    cluster_subset_p, color=["leiden", "Cd163l1", "Ccl5", "5830411N06Rik"],␣
↪legend_loc="on data"
)
```

```
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
```



[ ]:
```
'''
# EQUIVALENT R CODE

cds <- new_cell_data_set(as.matrix(merged@assays$RNA@counts),
                                    cell_metadata = merged@meta.data,
                                    gene_metadata = data.frame("gene_short_name"␣
↪= row.names(merged),
                                                                row.names = row.
↪names(merged)))
```

```
cds <- preprocess_cds(cds, num_dim = 50)

plot_pc_variance_explained(cds)

cds <- preprocess_cds(cds, num_dim = 5)

cds <- reduce_dimension(cds)

cds <- cluster_cells(cds, cluster_method = "louvain", k = 40)

plot_cells(cds)

plot_cells(cds, color_cells_by = "sim_time")

cds <- learn_graph(cds)

cds <- order_cells(cds, root_cells = colnames(cds)[which(cds$sim_time ==␣
 ↪min(cds$sim_time))])

plot_cells(cds, color_cells_by = "pseudotime")

monocle_pseudo <- pseudotime(cds)

actual_pseudo <- cds$sim_time

names(actual_pseudo) <- colnames(cds)

monocle_pseudo <- subset(monocle_pseudo, monocle_pseudo != Inf)

actual_pseudo <- actual_pseudo[names(monocle_pseudo)]

cor(actual_pseudo, monocle_pseudo)
'''
```