# scpy_assess-Copy1

March 25, 2024

[33]:
```
#Importing the relevant packages
import scanpy as sc
import pandas as pd
import anndata as ad
from adpbulk import ADPBulk
import scanpy.plotting as sc_plt
import bbknn
import matplotlib.pyplot as plt
import scanpy.external as sce
import random
from pydeseq2.dds import DeseqDataSet
from pydeseq2.ds import DeseqStats
import numpy as np
from scipy.stats import ttest_ind
from statsmodels.stats.multitest import multipletests
import decoupler as dc
```

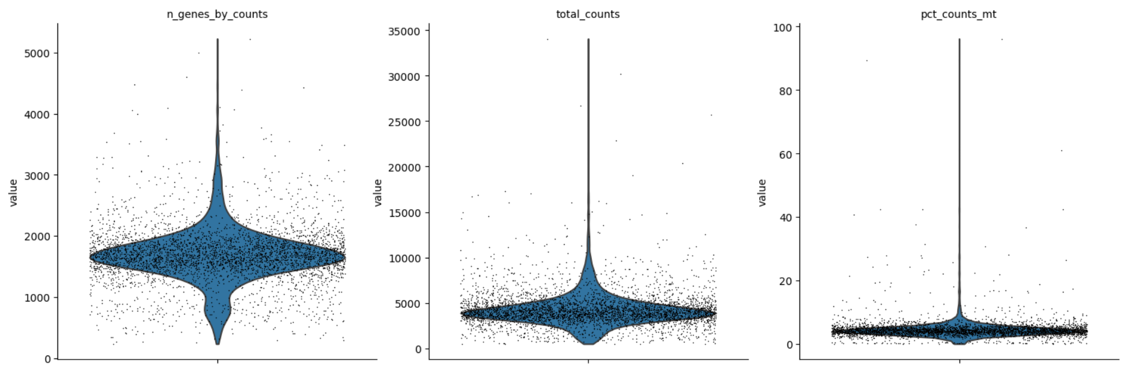
[34]:
```
#Reading the matrix data files from Knockout and wildtype samples
s1= sc.read_10x_mtx("/data/BIOL5177/Assessment/KO1/",
                    var_names= "gene_symbols",
                    cache= True)
s1.var_names_make_unique()

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()
```

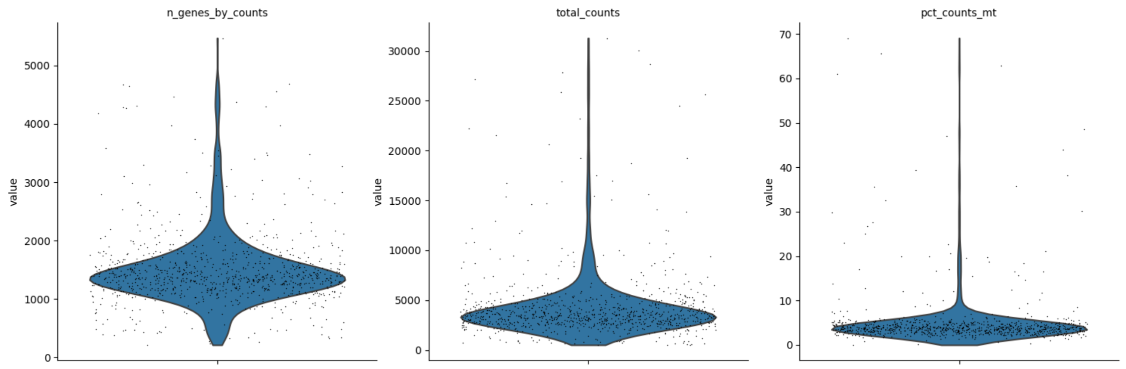
[ ]:
```
'''
# EQUIVALENT CODE IN R

# Loading required libraries

library(Seurat)

# Read 10x Genomics data for KO1

s1 <- Read10X(data.dir = "/data/BIOL5177/Assessment/KO1/")

# Create a Seurat object

s1 <- CreateSeuratObject(counts = s1, project = "KO1")
'''
```

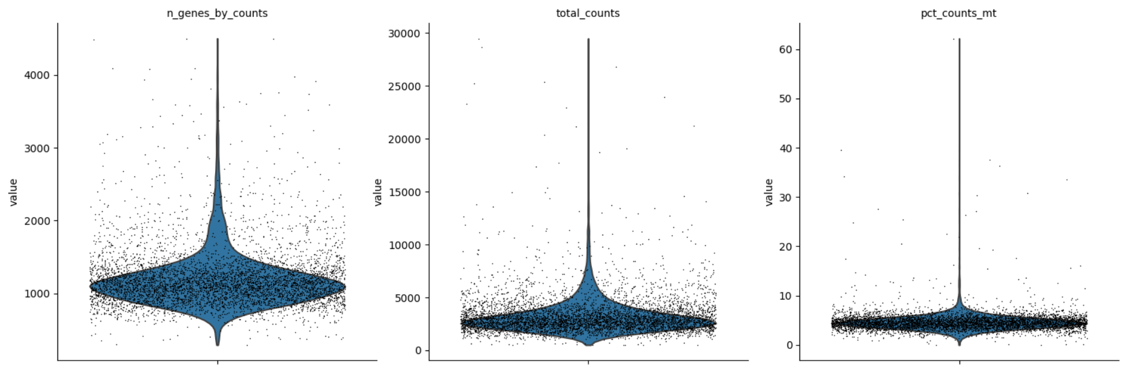
[35]:
```
#Including a column condition to assign groups to KO and wildtype samples
s1.obs["condition"]= "KO"
s2.obs["condition"]= "WT"
s3.obs["condition"]= "WT"
s4.obs["condition"]= "WT"
```

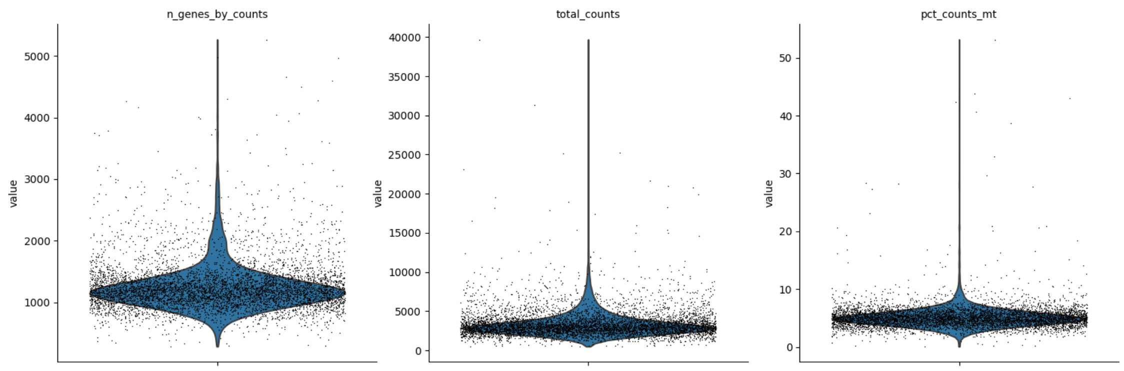
[36]:
```
#Creating a list of all the samples in AnnData format
sample_list= [s1, s2, s3, s4]
```

[37]:
```
#Filtering genes expressed in less than 200 cells
#filtering cells which express less than 3 genes
for sample in sample_list:
    sc.pp.filter_cells(sample, min_genes=200)
    sc.pp.filter_genes(sample, min_cells=3)
    sample.var["mt"] = sample.var_names.str.startswith("mt-")
    sc.pp.calculate_qc_metrics(
        sample, qc_vars=["mt"], percent_top=None, log1p=False, inplace=True
    )
    #Violin plot
    sc.pl.violin(
        sample,
        ["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
value
value

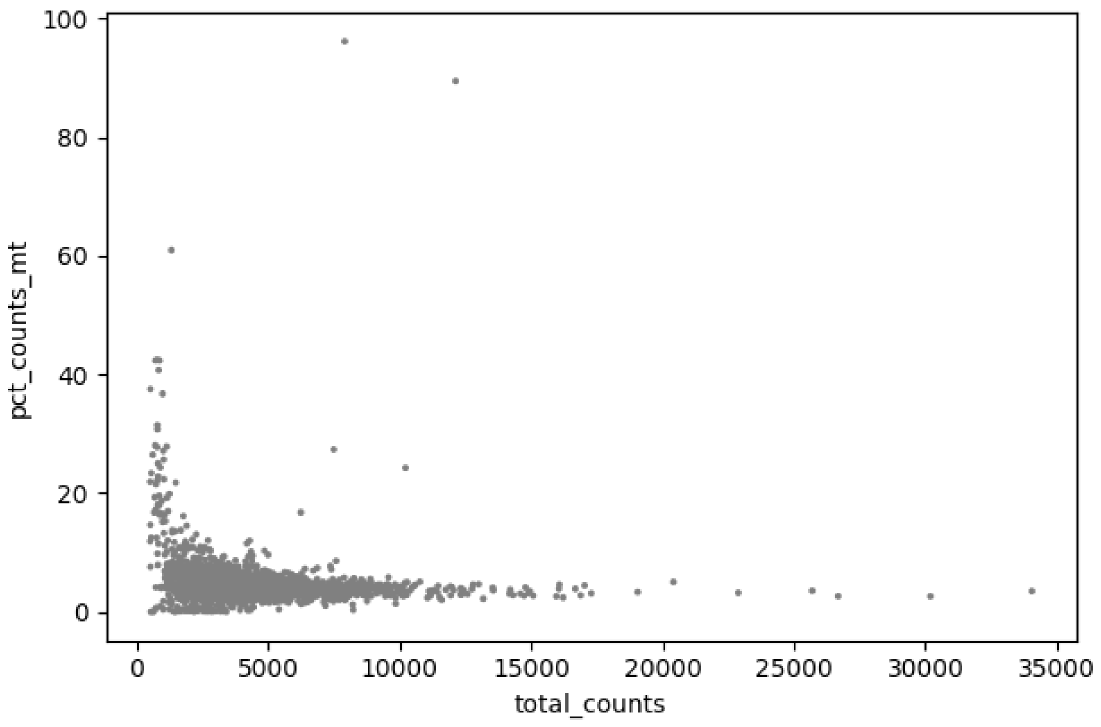

n_genes_by_counts
total_counts
pct_counts_mt
value
value
value

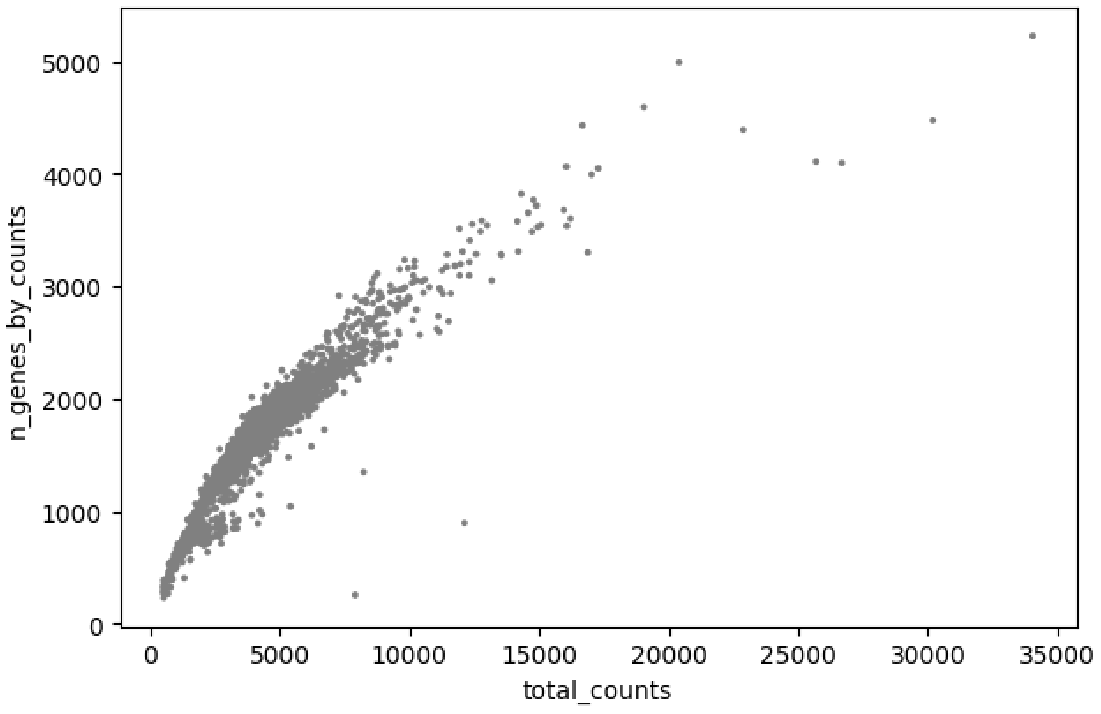

n_genes_by_counts
total_counts
pct_counts_mt
value
value
value

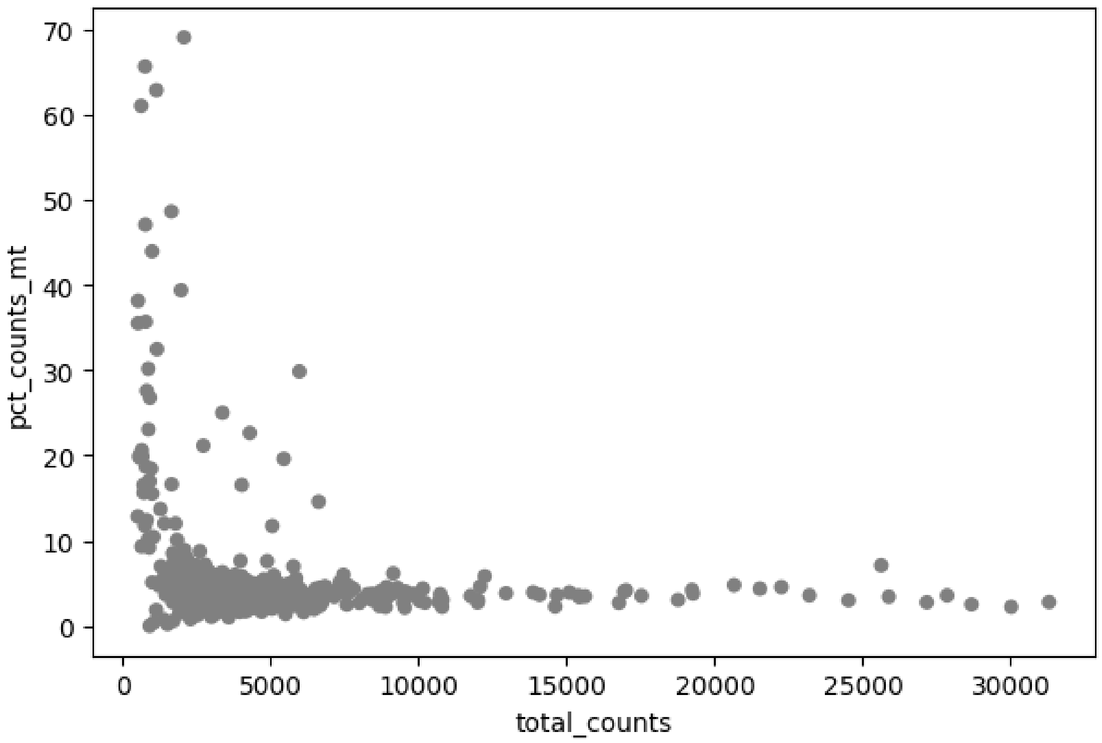


```
[ ]: '''
     # EQUIVALENT R CODE

     # Filter genes expressed in less than 200 cells and cells expressing less than␣
      ↪3 genes for each sample

     for (sample in sample_list) {
         sample <- subset(sample, subset = nFeature_RNA > 200)
         sample <- subset(sample, subset = nCount_RNA > 3)
         sample <- PercentageFeatureSet(sample, pattern = "^mt-", col.name =␣
      ↪"percent.mt")
         sample <- CalculateQCMetrics(sample, feature_controls = c("percent.mt"))
         VlnPlot(sample, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"),␣
      ↪jitter = 0.4, cols = c("skyblue", "salmon",  "orange"), combine = TRUE)
     }
     '''
```

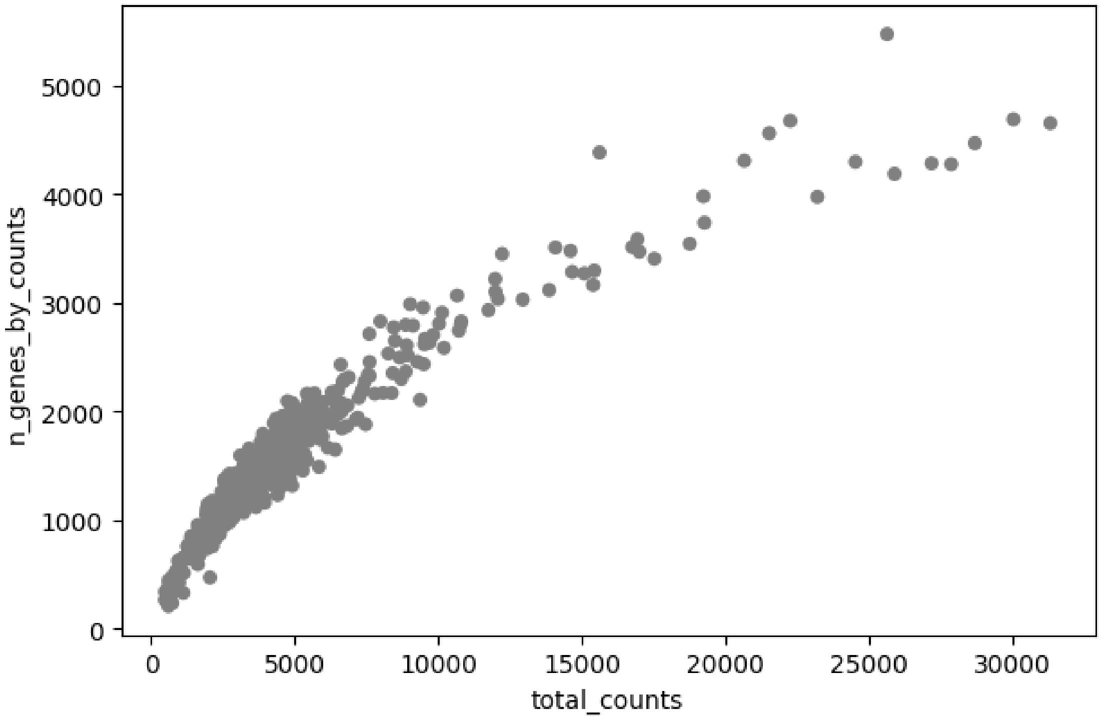
```
[38]: #Filtering genes with less than 3500 counts (doublets/undergoing apoptosis)
      #Filtering cells having proportion of mitochondrial counts more than 20 percent
      for sample in sample_list:
          sc.pl.scatter(sample, x="total_counts", y="pct_counts_mt")
          sc.pl.scatter(sample, x="total_counts", y="n_genes_by_counts")
          sample = sample[sample.obs.n_genes_by_counts < 3500, :]
          sample = sample[sample.obs.pct_counts_mt < 20, :].copy()
```


pct_counts_mt
total_counts
0
20
40
60
80
100
0
5000
10000
15000
20000
25000
30000
35000

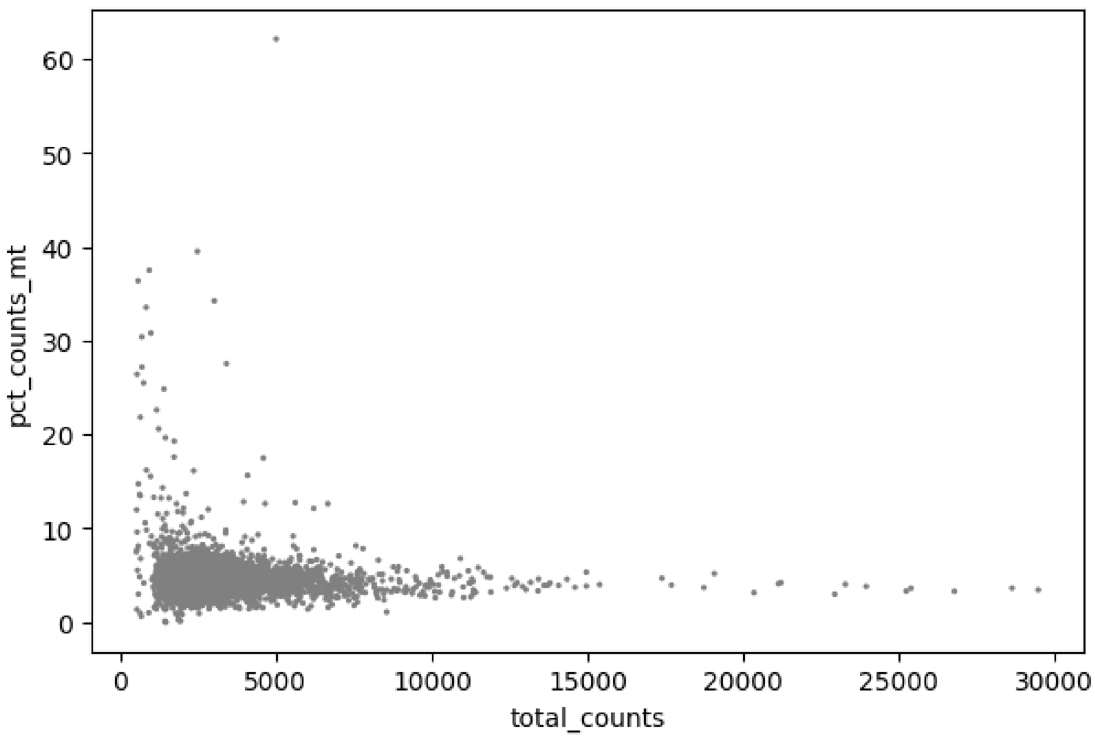

n_genes_by_counts
total_counts
0
1000
2000
3000
4000
5000
0
5000
10000
15000
20000
25000
30000
35000

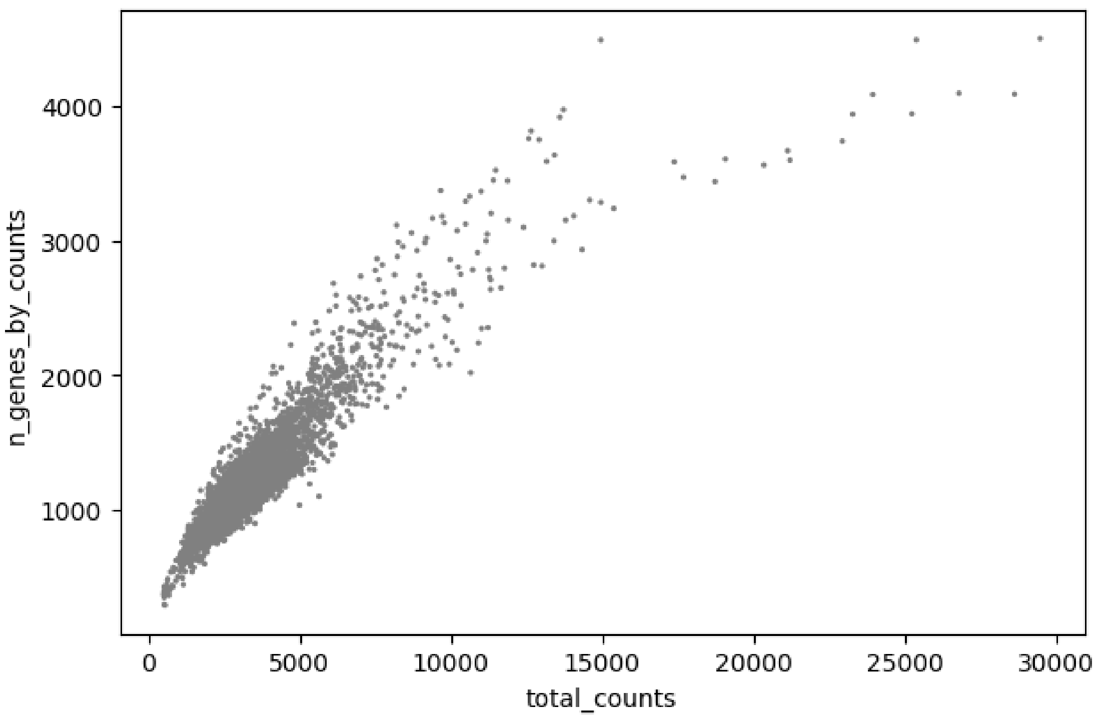


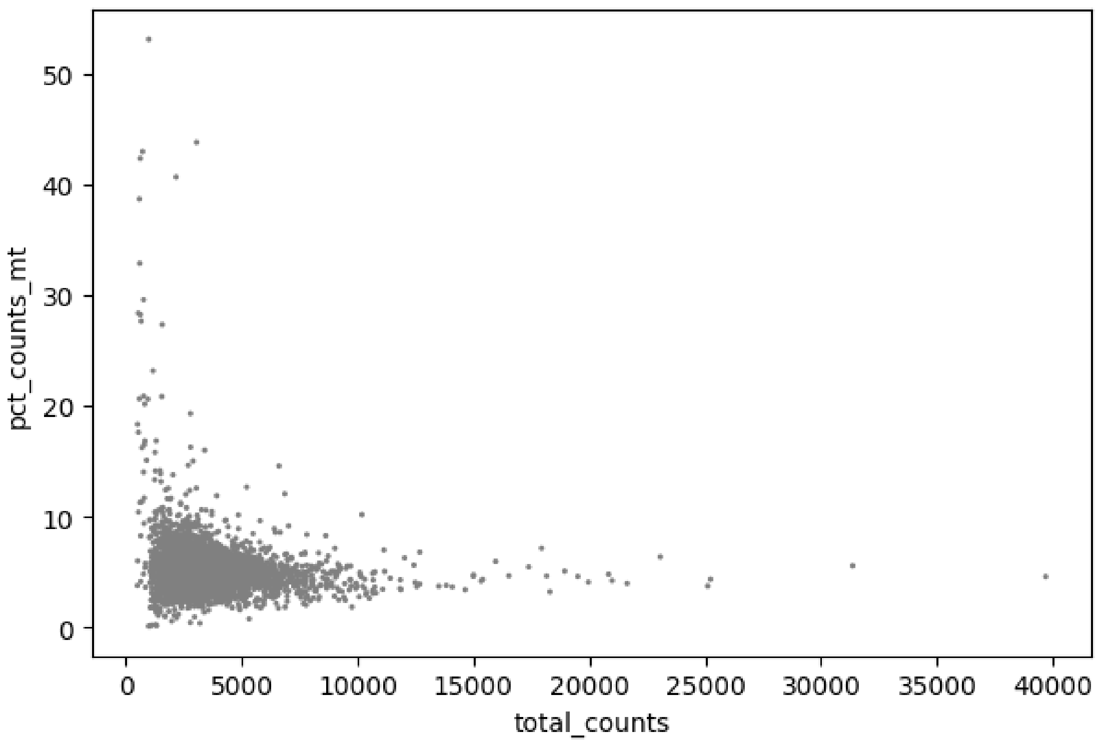
pct_counts_mt
total_counts
0
10
20
30
40
50
60
70
0
5000
10000
15000
20000
25000
30000

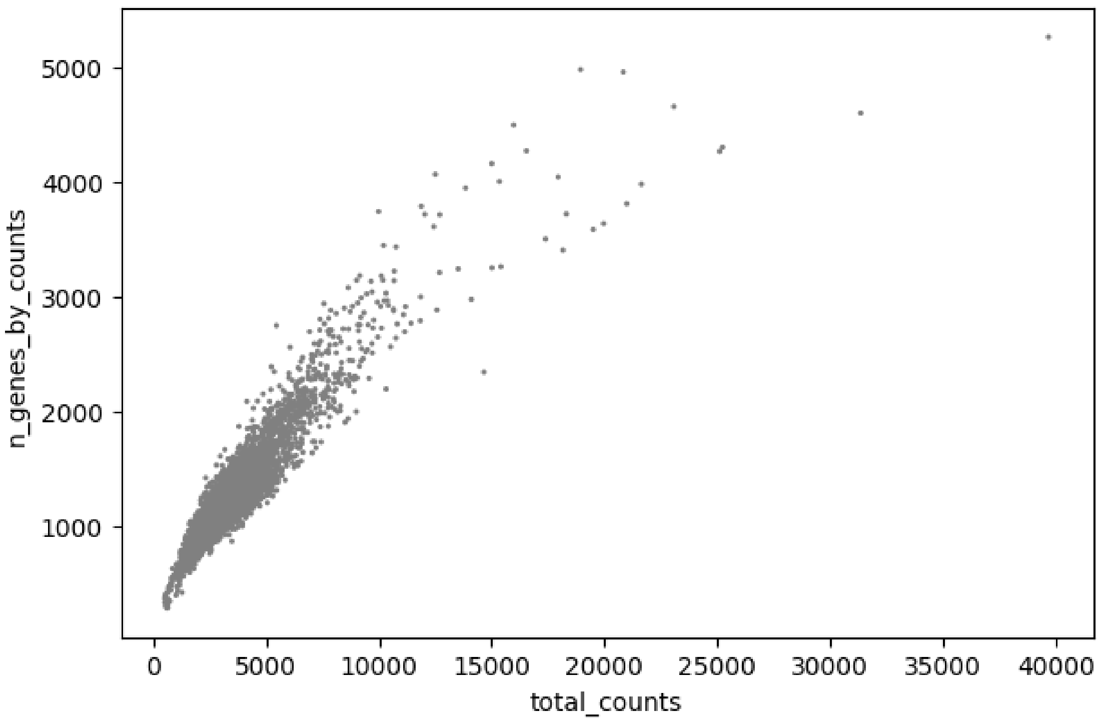


n_genes_by_counts
total_counts
0
1000
2000
3000
4000
5000
0
5000
10000
15000
20000
25000
30000


pct_counts_mt
total_counts
0
10
20
30
40
50
60
0
5000
10000
15000
20000
25000
30000


n_genes_by_counts
total_counts
1000
2000
3000
4000
0
5000
10000
15000
20000
25000
30000


pct_counts_mt
total_counts
0
10
20
30
40
50
0
5000
10000
15000
20000
25000
30000
35000
40000


n_genes_by_counts
total_counts
1000
2000
3000
4000
5000
0
5000
10000
15000
20000
25000
30000
35000
40000

```
[ ]: '''
     # EQUIVALENT R CODE
     # Scatter plot and filtering for each sample
     for (sample in sample_list) {
         # Scatter plot total counts vs. percentage of mitochondrial counts
         VlnPlot(sample, x = "nCount_RNA", y = "percent.mt")
         VlnPlot(sample, x = "nCount_RNA", y = "nFeature_RNA")
         sample <- subset(sample, subset = nFeature_RNA < 3500)
         sample <- subset(sample, subset = percent.mt < 20)
         sample <- subset(sample, select = c("nCount_RNA", "nFeature_RNA", "percent.
      ↪mt"))
     }
     '''
```

```
[39]: #Contenating the sample list to create a merged AnnData object to perform␣
       ↪integration
      all_samples= ad.concat([s1, s2, s3, s4], join= "outer", index_unique= "_")
      all_samples.obs['batch'] = ['KO1'] * s1.shape[0] + ['WT1'] * s2.shape[0] +␣
       ↪['WT2'] * s3.shape[0] + ['WT3'] * s4.shape[0]
```

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

     all_samples <- merge(x = sample_list[[1]],
                          y = sample_list[2:length(covid.list)],
                          merge.data = TRUE)

     # Assign batch information

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

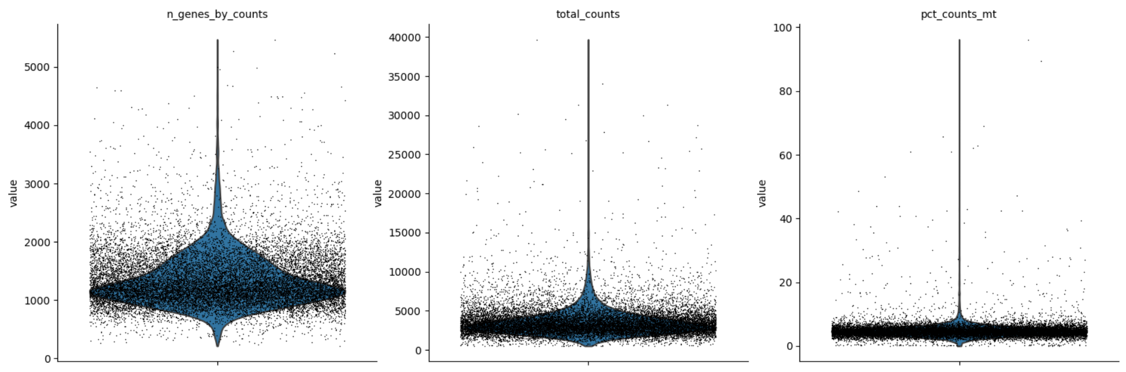
```
[40]: #Filtering genes expressed in less than 200 cells in merged samples
      #filtering cells which express less than 3 genes in merged samples
      sc.pp.filter_cells(all_samples, min_genes=200)
      sc.pp.filter_genes(all_samples, min_cells=3)
```

```
[41]: all_samples
```

```
[41]: AnnData object with n_obs × n_vars = 16841 × 15889
          obs: 'condition', 'n_genes', 'n_genes_by_counts', 'total_counts',
      'total_counts_mt', 'pct_counts_mt', 'batch'
          var: 'n_cells'
```

[42]:
```
all_samples.var["mt"] = all_samples.var_names.str.startswith("mt-")
sc.pp.calculate_qc_metrics(all_samples, qc_vars=["mt"], percent_top=None,␣
 ↪log1p=False, inplace=True)
```

[43]:
```
sc.pl.violin(
        all_samples,
        ["n_genes_by_counts", "total_counts", "pct_counts_mt"],
        jitter=0.4,
        multi_panel=True,
    )
```



[44]:
```
all_samples = all_samples[all_samples.obs.n_genes_by_counts < 3500 , :]
all_samples = all_samples[all_samples.obs.total_counts < 13000 , :]
all_samples = all_samples[all_samples.obs.pct_counts_mt < 15 , :]
```

[45]:
```
# Storing raw counts in layers
all_samples.X = np.round(all_samples.X)
all_samples.layers['counts'] = all_samples.X
```

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

# Round the counts to integers

all_samples@assays$RNA@counts <- round(all_samples@assays$RNA@counts)

# Create a new layer and assign raw counts

all_samples <- SetAssayData(object = all_samples, assay = "counts", data =␣
 ↪all_samples@assays$RNA@counts)
'''
```

```
[46]: all_samples
```

```
[46]: AnnData object with n_obs × n_vars = 16584 × 15889
          obs: 'condition', 'n_genes', 'n_genes_by_counts', 'total_counts',
      'total_counts_mt', 'pct_counts_mt', 'batch'
          var: 'n_cells', 'mt', 'n_cells_by_counts', 'mean_counts',
      'pct_dropout_by_counts', 'total_counts'
          layers: 'counts'
```

```
[47]: # Normalize total counts in Scanpy AnnData object
      sc.pp.normalize_total(all_samples, target_sum=1e4)
```

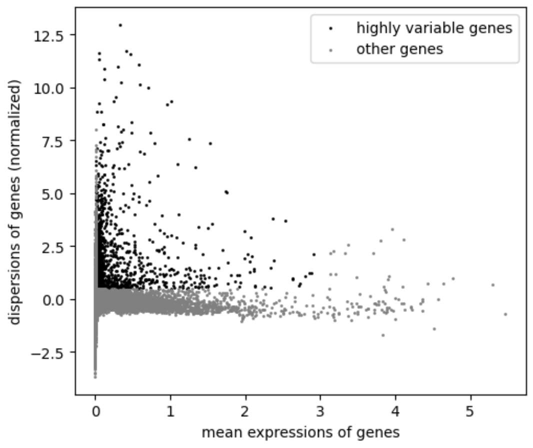
```
[ ]: '''
     # EQUIVALENT R CODE

     # Normalize total counts in Seurat object

     all_samples <- NormalizeData(all_samples, normalization.method = "total", scale.
      ↪factor = 1e4)
     '''
```

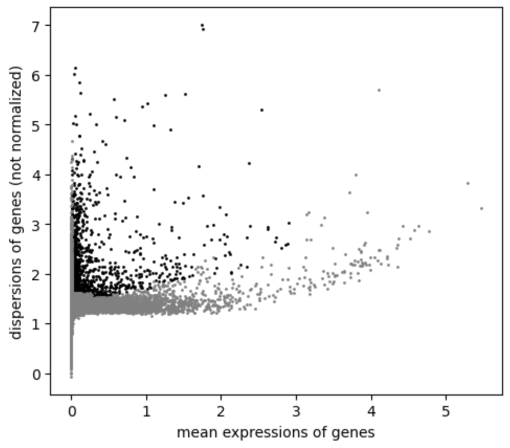
```
[48]: #Applyuing log transformation to normalised counts
      sc.pp.log1p(all_samples)
      #storing the normalised counts in normalised layer
      all_samples.layers['normalized'] = all_samples.X
```

```
[49]: #Identifying highly variable genes
      sc.pp.highly_variable_genes(all_samples, min_mean=0.0125, max_mean=3,␣
       ↪min_disp=0.5)
```

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

     all_samples <- FindVariableFeatures(all_samples, selection.method = "vst",
                                         nfeatures = 2000, min.mean = 0.0125,
                                         max.mean = 3, min.disp = 0.5)
     '''
```

```
[50]: sc.pl.highly_variable_genes(all_samples)
```





[51]:
```
all_samples.raw = all_samples
```

[52]:
```
#Subsetting the merged data with highly variable genes
all_samples=all_samples[:, all_samples.var.highly_variable]
```

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

# Subset Seurat object to retain only highly variable genes

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

[53]:
```
#Performing regression based on total counts and proportion of mitochondrial␣
 ↪features
sc.pp.regress_out(all_samples, ['total_counts', 'pct_counts_mt'])
```

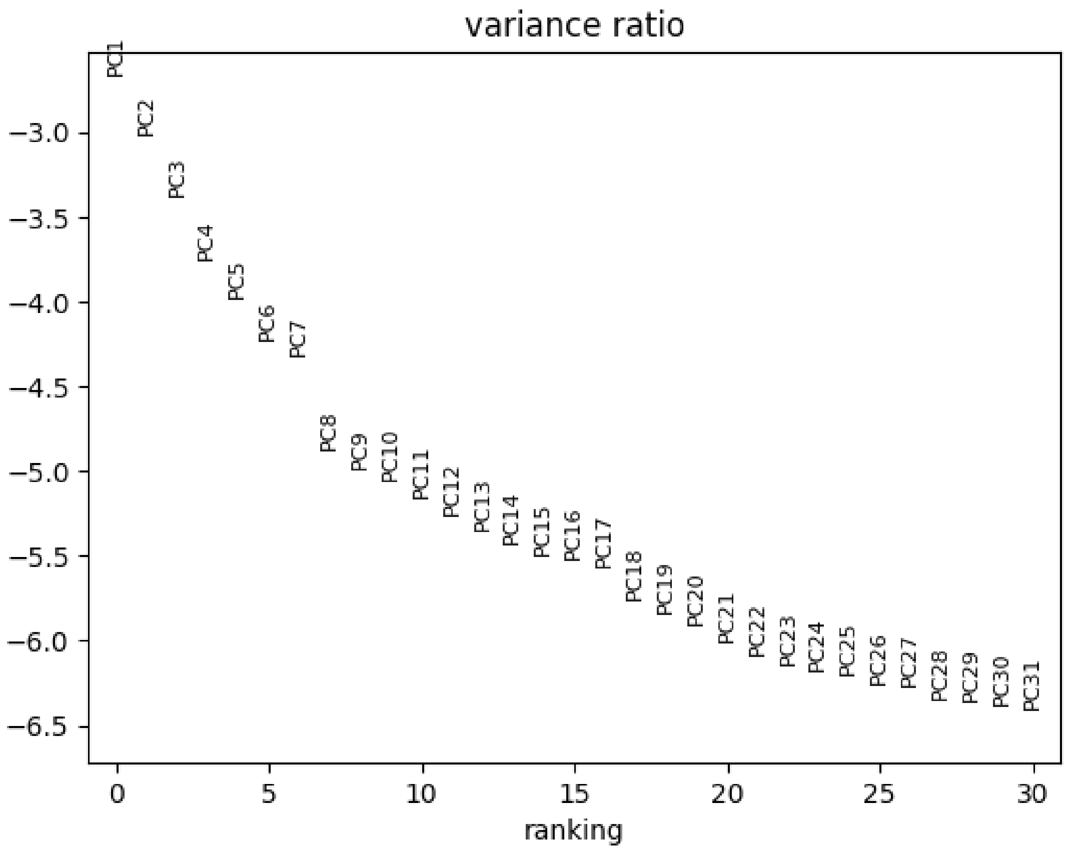
[54]:
```
# Scaling the data and set a maximum value of standard deviation 10 for each␣
 ↪feature
sc.pp.scale(all_samples, max_value=10)
```

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

all_samples <- ScaleData(all_samples, verbose = FALSE)
'''
```

[55]:
```
# Perform PCA on the data using the ARPACK solver
sc.tl.pca(all_samples, svd_solver= 'arpack')
```

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

     all_samples <- RunPCA(all_samples, npcs = 20, verbose = FALSE)
     '''
```

```
[56]: #Elbow plot
      sc.pl.pca_variance_ratio(all_samples, log=True)
```



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

     ElbowPlot(all_samples)
     '''
```

```
[57]: #Writing the unintegrating file
      all_samples.write('rawFile')
```

[58]:
```python
# ComputING neighborhood graph using 10 nearest neighbors and 20 principal␣
↪components
sc.pp.neighbors(all_samples, n_neighbors=10, n_pcs=20)
```

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

all_samples <- FindNeighbors(all_samples, reduction = "pca", dims = 1:20)
'''
```

[59]:
```python
# Computing UMAP embedding using default parameters in Scanpy
sc.tl.umap(all_samples)
```

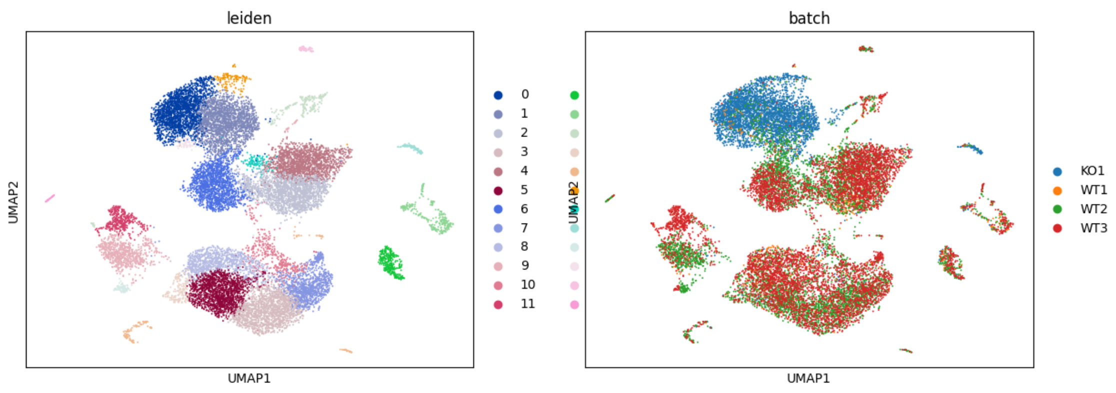
[ ]:
```python
'''
# EQUIVALENT R CODE

all_samples <- RunUMAP(all_samples, reduction = "pca", dims = 1:20)
'''
```

[60]:
```python
# Performing Leiden clustering with resolution parameter set to 1 in Scanpy
sc.tl.leiden(all_samples, resolution=1)
```

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

all_samples <- FindClusters(all_samples, resolution = 0.5)
'''
```

[61]:
```python
#UMAP plot
sc.pl.umap(all_samples, 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

     DimPlot(all_samples, group.by = "orig.ident")
     '''
```

```
[62]: # Perform batch-balanced k-nearest neighbors (bbknn) integration
      sc.external.pp.bbknn(all_samples, batch_key= 'batch')
```

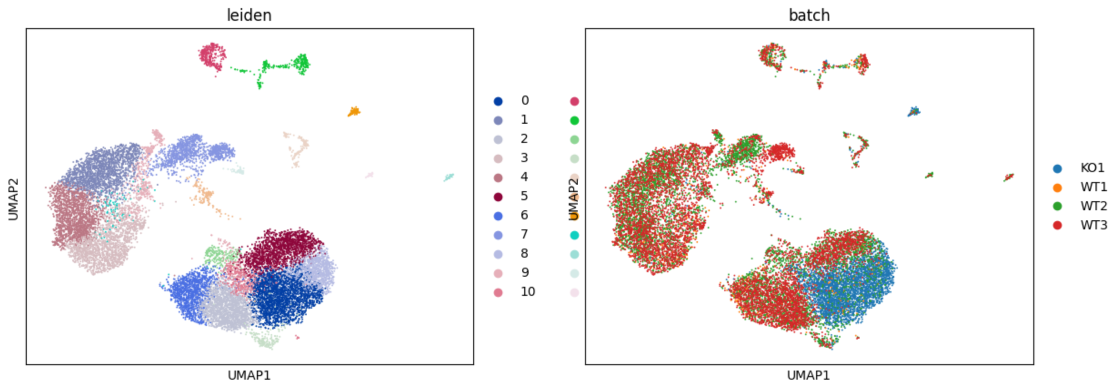
```
WARNING: consider updating your call to make use of `computation`
```

```
[63]: sc.tl.umap(all_samples)
```

```
[64]: sc.tl.leiden(all_samples, resolution=1)
```

```
[65]: sc.pl.umap(all_samples,  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

# anchors for integration

anchors <- FindIntegrationAnchors(object.list = sample_list, dims = 1:20)

# Integrating data

all_samples_integrated <- IntegrateData(anchorset = anchors)

librar(bbknnR)

all_samples= bbknnR(all_samples, batch= "orig.ident")

# Scaling integrated data

all_samples_integrated <- ScaleData(all_samples_integrated)

# Running PCA

all_samples_integrated <- RunPCA(all_samples_integrated)

# Projecting PCA to a lower-dimensional space (UMAP)

all_samples_integrated <- RunUMAP(all_samples_integrated)

# clustering

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

[66]:
```
all_samples
```

```
[66]: AnnData object with n_obs × n_vars = 16584 × 1226
          obs: 'condition', 'n_genes', 'n_genes_by_counts', 'total_counts',
      'total_counts_mt', 'pct_counts_mt', 'batch', 'leiden'
          var: 'n_cells', 'mt', 'n_cells_by_counts', 'mean_counts',
      'pct_dropout_by_counts', 'total_counts', 'highly_variable', 'means',
      'dispersions', 'dispersions_norm', 'mean', 'std'
          uns: 'log1p', 'hvg', 'pca', 'neighbors', 'umap', 'leiden', 'leiden_colors',
      'batch_colors'
          obsm: 'X_pca', 'X_umap'
          varm: 'PCs'
          layers: 'counts', 'normalized'
          obsp: 'distances', 'connectivities'
```

```
[67]: # Setting the base of the logarithm used for log1p transformation to 10
      all_samples.uns['log1p']['base']=10
```

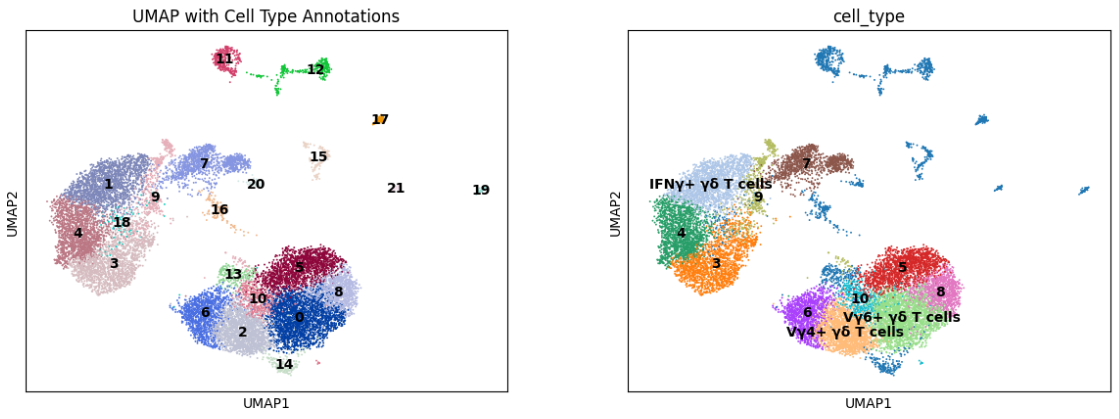
```
[68]: #Identifying the three largest clusters
      cluster_sizes = all_samples.obs['leiden'].value_counts()
      largest_clusters = cluster_sizes.nlargest(3).index.tolist()

      #Subsetting the data to include only cells from the largest clusters
      subset_data = all_samples[all_samples.obs['leiden'].isin(largest_clusters)].
       ↪copy()
```

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

     subset_data <- subset(all_samples, idents = c(0,1,2))
     '''
```

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

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

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

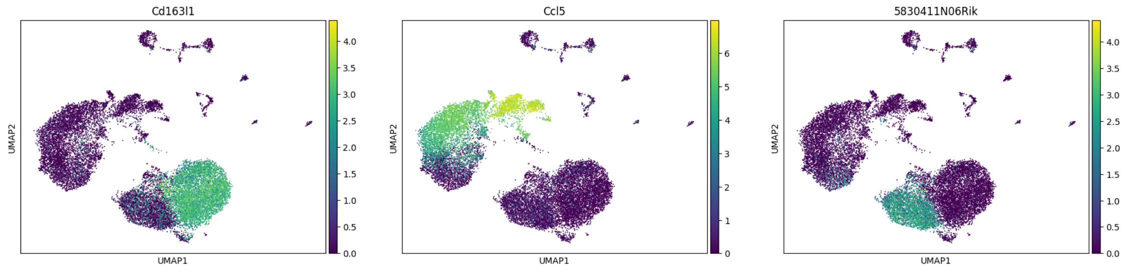
```
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
/usr/local/lib/python3.10/dist-
packages/scanpy/plotting/_tools/scatterplots.py:392: UserWarning: No data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



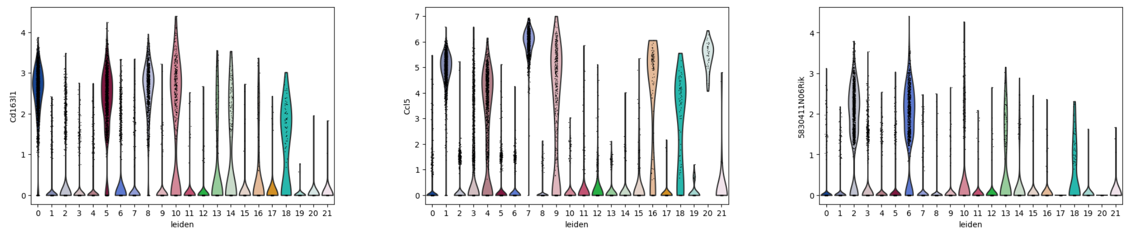
```
[70]: subset_data
```

```
[70]: AnnData object with n_obs × n_vars = 5949 × 1226
          obs: 'condition', 'n_genes', 'n_genes_by_counts', 'total_counts',
      'total_counts_mt', 'pct_counts_mt', 'batch', 'leiden'
          var: 'n_cells', 'mt', 'n_cells_by_counts', 'mean_counts',
      'pct_dropout_by_counts', 'total_counts', 'highly_variable', 'means',
      'dispersions', 'dispersions_norm', 'mean', 'std'
          uns: 'log1p', 'hvg', 'pca', 'neighbors', 'umap', 'leiden', 'batch_colors'
          obsm: 'X_pca', 'X_umap'
          varm: 'PCs'
          layers: 'counts', 'normalized'
          obsp: 'distances', 'connectivities'
```

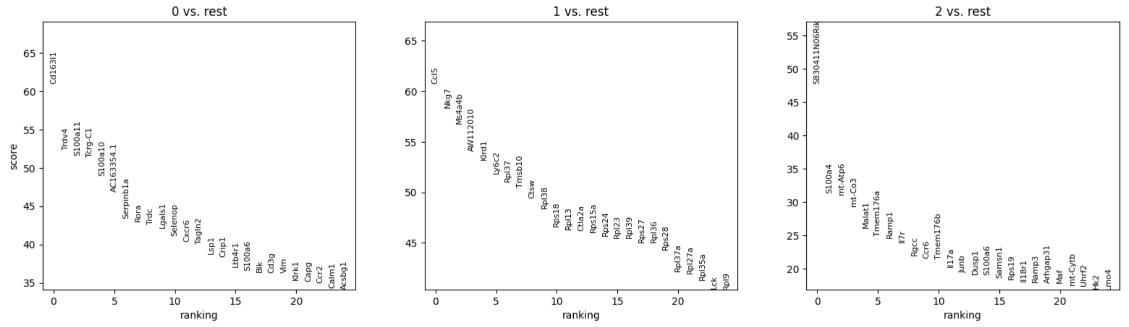
[71]:
```
sc.pl.umap(all_samples, color=["Cd163l1", "Ccl5", "5830411N06Rik"])
```



[72]:
```
sc.pl.violin(all_samples, ["Cd163l1", "Ccl5", "5830411N06Rik"],␣
↪groupby="leiden")
```



[73]:
```
sc.tl.rank_genes_groups(subset_data, 'leiden', method='wilcoxon', use_raw=True)
sc.pl.rank_genes_groups(subset_data, n_genes=25, sharey=False)
```

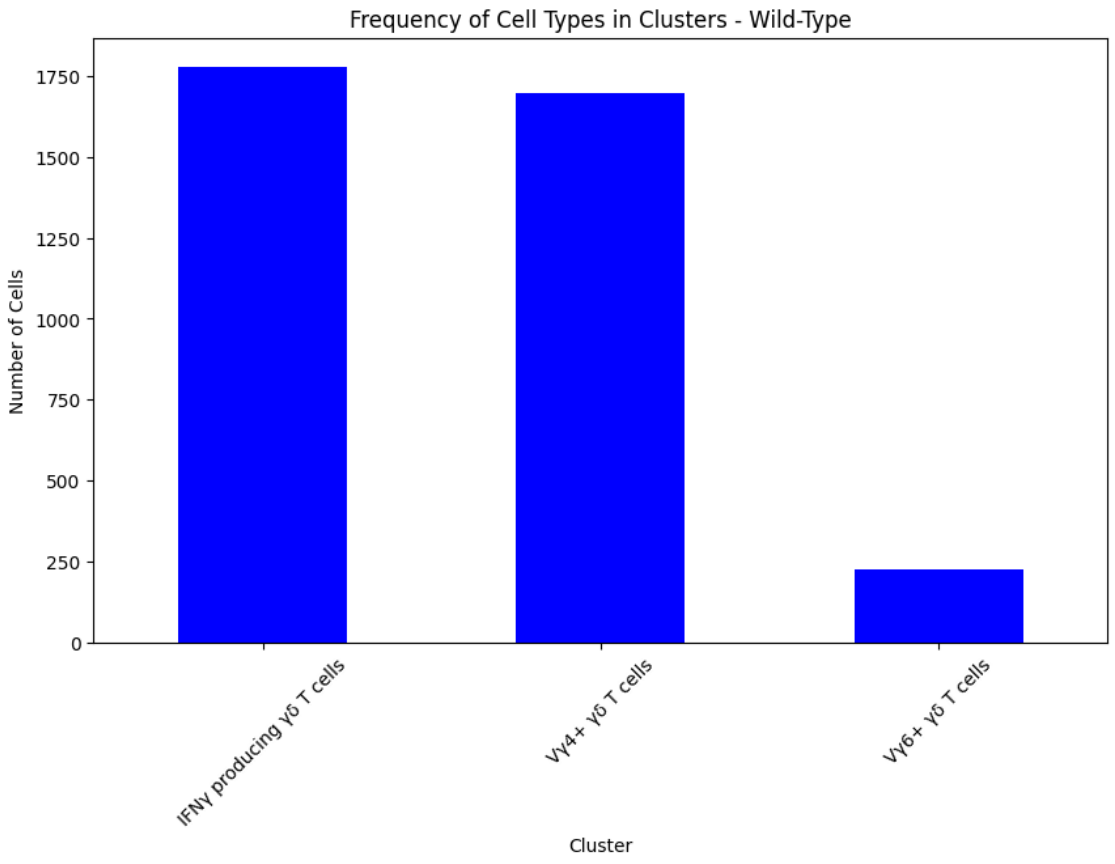


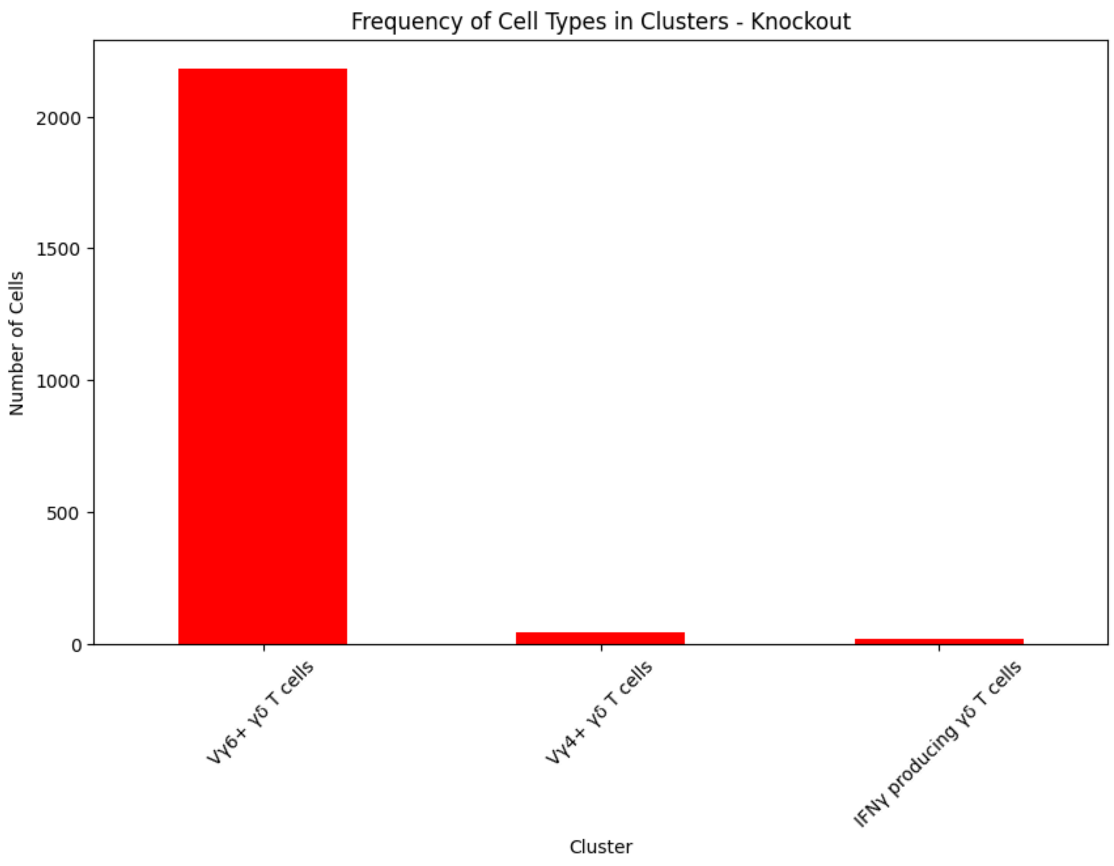
[ ]:
```
'''
# EQUIVALENT R CODE

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

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

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


# Rename clusters

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


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

[74]:
```python
# Subsetting data for WT and KO groups
wt_data = subset_data[subset_data.obs['batch'] != 'KO1'].copy()
ko_data = subset_data[subset_data.obs['batch'] == 'KO1'].copy()

# Calculate cluster frequencies for WT and KO groups
wt_freq = wt_data.obs['leiden'].value_counts()
ko_freq = ko_data.obs['leiden'].value_counts()

# Define cluster annotations
cluster_annotations = {
    '0': 'V 6+   T cells',
    '1': 'IFN  producing   T cells',
    '2': 'V 4+   T cells'
}

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

# Customize x-labels using cluster annotations
plt.xticks(range(len(wt_freq)), [cluster_annotations[str(i)] for i in wt_freq.
 ↪index])

plt.show()
```

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

# Customize x-labels using cluster annotations
plt.xticks(range(len(ko_freq)), [cluster_annotations[str(i)] for i in ko_freq.
 ↪index])

plt.show()
```





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

     # Calculating the proportion of cells in each cluster for each group

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

     # Plotting bar chart showing the proportion of cells in each cluster for each␣
      ↪group using ggplot2

     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)))
     '''
```

[75]:
```
import pandas as pd
pd.DataFrame(subset_data.uns["rank_genes_groups"]["names"]).head(20)
```

[75]:
```
             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
10     Selenop     Rps18       Tmem176b
11       Cxcr6     Rpl13          Il17a
12      Tagln2    Ctla2a           Junb
13        Lsp1    Rps15a          Dusp1
14       Crip1     Rps24         S100a6
15      Ltb4r1     Rpl23         Samsn1
16      S100a6     Rpl39          Rps19
17         Blk     Rps27         Il18r1
18        Cd3g     Rpl36          Ramp3
19         Vim     Rps28       Arhgap31
```

[76]:
```
# Identify the largest cluster
largest_cluster_d = cluster_sizes.nlargest(1).index.tolist()
# Subset the data to include only cells from the largest cluster
subset_data_d = all_samples[all_samples.obs['leiden'].isin(largest_cluster_d)].
  ↪copy()
```

[77]:
```
# Performing differential expression analysis between KO1 and combined WT␣
  ↪samples
sc.tl.rank_genes_groups(subset_data_d, groupby='batch', groups=['KO1'],␣
  ↪reference= 'rest', method='wilcoxon')
```

[78]:
```
results= subset_data_d.uns['rank_genes_groups']
```

[79]:
```
subset_data_d.uns['rank_genes_groups']['names']
```

[79]:
```
rec.array([('Klrk1',), ('Ddx5',), ('Rpl10',), …, ('Rpl37',), ('Rpl41',),
           ('Rps28',)],
          dtype=[('KO1', 'O')])
```

[80]:
```
de_genes = results['names']['KO1']
de_pvals = results['pvals_adj']['KO1']
de_logFC = results['logfoldchanges']['KO1']
```

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

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

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

```
Number of genes up-regulated in the KO with corrected p-values < 0.01 and logFC
> 0.5 (using bbkn integration): 275
```

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

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

subset_data_d <- subset(all_samples, subset = batch == "KO1")

# Running differential expression analysis

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

# Extracting 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)
'''
```

```
[120]: 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))
```

```
[121]: out.shape
```

```
[121]: (15890, 5)
```

```
[122]: markers= pd.DataFrame(out[1:], columns= ['Gene', 'scores', 'pval_adj', 'lfc',␣
        ↪'cluster'])
```

```
[123]: markers = markers[(markers['pval_adj'] < 0.01) & (markers['lfc'] > 0.5)]
```

```
[124]: markers
```

```
[124]:             Gene     scores  pval_adj       lfc cluster
       0           Klrk1  13.996684       0.0  3.077076     KO1
       1            Ddx5  12.683079       0.0  2.119973     KO1
       2           Rpl10  10.747263       0.0  1.456836     KO1
       3          mt-Nd4  10.701342       0.0  1.242422     KO1
       4          Hnrnpk   10.67269       0.0  2.588415     KO1
       ..            …          …         …        …       …
       272  2810474O19Rik   3.719263  0.008746  1.024619     KO1
       273          Slfn2   3.701287  0.009364  0.870085     KO1
       274         H2-Q4   3.700028  0.009377  1.067382     KO1
       275          Ywhae   3.699525  0.009377  0.906726     KO1
       276           Ctsd    3.68598  0.009863  0.936494     KO1

       [275 rows x 5 columns]
```

```
[85]: all_samples_h= sc.read_h5ad('rawFile')
```

```
[86]: all_samples_h.obs
```

```
[86]:                     condition  n_genes  n_genes_by_counts  total_counts  \
      AAACCCAAGAGCCTGA-1_0        KO     1548               1548        3483.0
      AAACCCAAGATGACCG-1_0        KO     2032               2032        6019.0
      AAACCCACAACTGTGT-1_0        KO     1694               1694        3892.0
      AAACCCATCGTAACCA-1_0        KO     1480               1480        3729.0
      AAACGAACATCGGAGA-1_0        KO     1804               1804        4734.0
      …                          …       …                  …            …
      TTTGTTGCATACCAGT-1_3        WT     1201               1201        3171.0
      TTTGTTGCATCAGCGC-1_3        WT     1308               1308        3362.0
      TTTGTTGGTGCTGTCG-1_3        WT      636                636        1275.0
      TTTGTTGGTGGTCTCG-1_3        WT     1468               1468        3989.0
      TTTGTTGGTTGATGTC-1_3        WT     1497               1497        4927.0

                            total_counts_mt  pct_counts_mt batch
      AAACCCAAGAGCCTGA-1_0            162.0       4.651163   KO1
      AAACCCAAGATGACCG-1_0            218.0       3.621864   KO1
      AAACCCACAACTGTGT-1_0            189.0       4.856115   KO1
```

```
AAACCCATCGTAACCA-1_0          127.0      3.405739   KO1
AAACGAACATCGGAGA-1_0          130.0      2.746092   KO1
…                             …            …    …
TTTGTTGCATACCAGT-1_3          245.0      7.726269   WT3
TTTGTTGCATCAGCGC-1_3          174.0      5.175491   WT3
TTTGTTGGTGCTGTCG-1_3          112.0      8.784313   WT3
TTTGTTGGTGGTCTCG-1_3          225.0      5.640512   WT3
TTTGTTGGTTGATGTC-1_3          297.0      6.028009   WT3

[16584 rows x 7 columns]
```

[87]:
```
sce.pp.harmony_integrate(all_samples_h, 'batch')
all_samples_h.obsm['X_pca']= all_samples_h.obsm['X_pca_harmony']
```

```
2024-03-25 03:40:20,735 - harmonypy - INFO - Computing initial centroids with
sklearn.KMeans…
2024-03-25 03:40:26,090 - harmonypy - INFO - sklearn.KMeans initialization
complete.
2024-03-25 03:40:26,129 - harmonypy - INFO - Iteration 1 of 10
2024-03-25 03:40:28,904 - harmonypy - INFO - Iteration 2 of 10
2024-03-25 03:40:31,709 - harmonypy - INFO - Iteration 3 of 10
2024-03-25 03:40:34,521 - harmonypy - INFO - Iteration 4 of 10
2024-03-25 03:40:37,634 - harmonypy - INFO - Iteration 5 of 10
2024-03-25 03:40:39,104 - harmonypy - INFO - Converged after 5 iterations
```

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

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

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

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

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

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

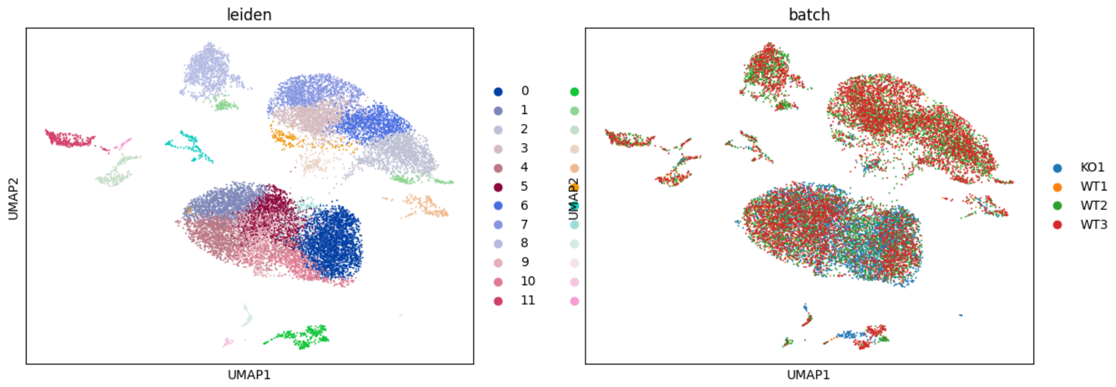
[88]:
```
sc.pp.neighbors(all_samples_h, n_neighbors=10, n_pcs=20)
```

[89]:
```
sc.tl.umap(all_samples_h)
```

[90]:
```
sc.tl.leiden(all_samples_h, resolution=1)
```

[91]:
```
sc.pl.umap(all_samples_h,  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(
```



[92]:
```
sc.pp.log1p(all_samples_h)
```

```
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)
```

[93]:
```
cluster_sizes_e = all_samples_h.obs['leiden'].value_counts()
largest_cluster_e = cluster_sizes_e.nlargest(1).index.tolist()
subset_data_e = all_samples_h[all_samples_h.obs['leiden'].
  ↪isin(largest_cluster_e)].copy()
```

[94]:
```
# Performing differential expression analysis between KO1 and combined WT␣
  ↪samples
sc.tl.rank_genes_groups(subset_data_e, groupby='batch', groups=['KO1'],␣
  ↪reference='rest', method='wilcoxon')
```

[95]:
```
results_har= subset_data_e.uns['rank_genes_groups']
```

[96]:
```
subset_data_e.uns['rank_genes_groups']['names']
```

```
[96]: rec.array([('Klrk1',), ('AC163354.1',), ('Crip1',), …, ('Rpl37a',),
                 ('Rpl38',), ('Rps28',)],
                dtype=[('KO1', 'O')])
```

```
[97]: de_genes = results_har['names']['KO1']
      de_pvals = results_har['pvals_adj']['KO1']
      de_logFC = results_har['logfoldchanges']['KO1']

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

      # Counting the number of up-regulated genes in the KO group
      num_upregulated_genes_ko_h = len(upregulated_genes_ko_har)

      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_h)
```

```
Number of genes up-regulated in the KO with corrected p-values < 0.01 and logFC
> 0.5 (using Harmony integration): 1473
```

```
[116]: out_har= np.array([[0, 0, 0, 0, 0]])
       for group in results_har['names'].dtype.names:
           out_har= np.vstack((out_har, np.vstack((results_har['names'][group],
                                                   results_har['scores'][group],
                                                   results_har['pvals_adj'][group],
                                                   results_har['logfoldchanges'][group],
                                                   np.
         ↪array([group]*len(results_har['names'][group])).astype('object'))).T))
```

```
[117]: out_har.shape
```

```
[117]: (15890, 5)
```

```
[118]: markers= pd.DataFrame(out_har[1:], columns= ['Gene', 'scores', 'pval_adj',␣
         ↪'lfc', 'cluster'])
       markers = markers[(markers['pval_adj'] < 0.01) & (markers['lfc'] > 0.5)]
```

```
[119]: markers
```

```
[119]:          Gene     scores  pval_adj       lfc  cluster
       0        Klrk1  29.995029       0.0  2.727787      KO1
       1   AC163354.1  29.754721       0.0  3.184084      KO1
       2        Crip1  29.648893       0.0  1.736999      KO1
       3        Itgb1  24.578897       0.0  3.420709      KO1
       4         Cd47  24.577991       0.0  1.767219      KO1
```

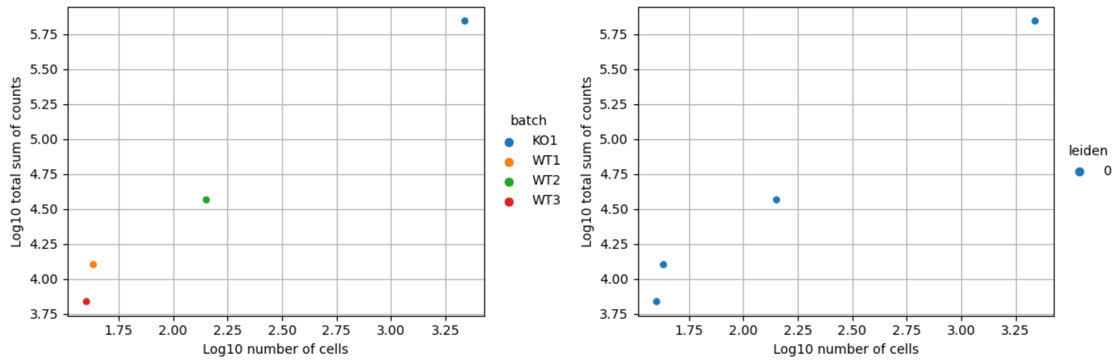
```
…           …        …        …         …        …
1569      Cdipt   3.265543  0.009575  0.618732      KO1
1570    Sec61a1   3.261837  0.009696  0.696789      KO1
1571     Gabpb1   3.258669  0.009799  0.877355      KO1
1572      Bud31   3.257254  0.009837  0.612554      KO1
1573     Rad23a   3.255302    0.0099  0.611135      KO1

[1473 rows x 5 columns]
```

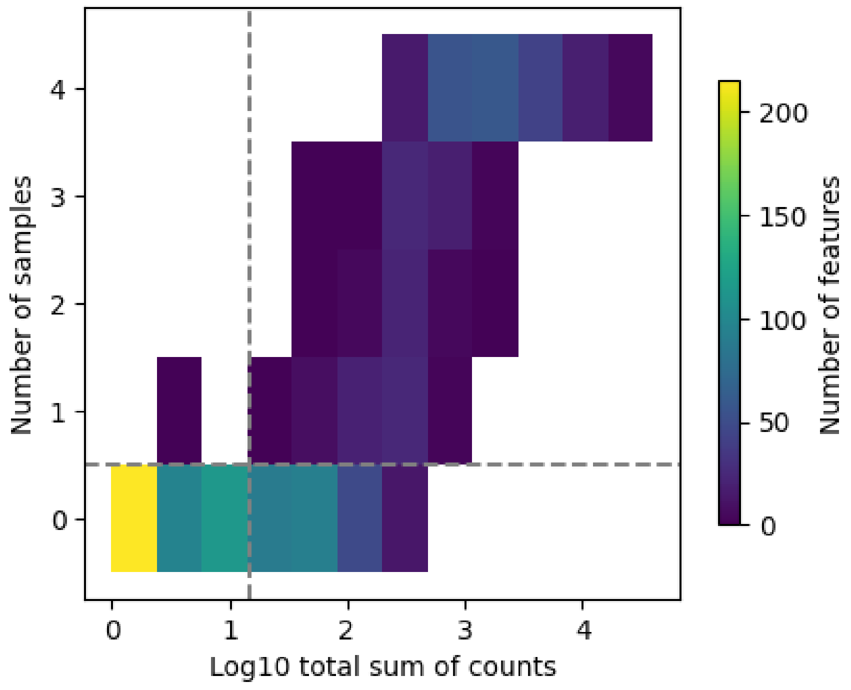
[101]:
```python
# Get pseudo-bulk profile
pdata_1 = dc.get_pseudobulk(
    subset_data_d,
    sample_col='batch',
    groups_col='leiden',
    layer='counts',
    mode='sum',
    min_cells=10,
    min_counts=1000
)
dc.plot_psbulk_samples(pdata_1, groupby=['batch', 'leiden'], figsize=(12, 4))
dc.plot_filter_by_expr(pdata_1, group='batch', min_count=10, min_total_count=15)
from pydeseq2.dds import DeseqDataSet, DefaultInference
# Build DESeq2 object
inference_1 = DefaultInference(n_cpus=8)
dds_1 = DeseqDataSet(
    adata=pdata_1,
    design_factors='condition',
    ref_level=['condition', 'WT'],
    refit_cooks=True,
    inference=inference_1,
)
# Compute LFCs
dds_1.deseq2()
# Extract contrast between COVID-19 vs normal
stat_res_1 = DeseqStats(
    dds_1,
    contrast=["condition", 'KO', 'WT'],
    inference=inference_1,
)
stat_res_1.summary()
results_df= stat_res_1.results_df
```

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

Fitting dispersions…
… done in 0.35 seconds.
```

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

/home1/bioinfo-27/.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 0.32 seconds.

Fitting LFCs…
… done in 0.31 seconds.

Replacing 0 outlier genes.

Running Wald tests…
… done in 0.31 seconds.


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]
```





```
[ ]: '''
     # Get pseudo-bulk profile using AggregateExpression with orig.ident

     pdata_1 <- Seurat::AggregateExpression(subset_data_d, group.by = c("batch",␣
      ↪"orig.ident"), assay = "RNA", FUN = sum)

     # Plot pseudo-bulk samples
```

```
Seurat::VlnPlot(pdata_1, features = rownames(pdata_1), group.by = c("batch",␣
 ↪"orig.ident"))

# Filter cells by expression

pdata_1_filtered <- Seurat::subset(x = pdata_1, cells = which(rowSums(pdata_1)␣
 ↪>= 15))

# Build DESeq2 object

dds_1 <- DESeq2::DESeqDataSetFromMatrix(countData = pdata_1_filtered, colData =␣
 ↪NULL, design = ~ batch)


dds_1 <- DESeq2::DESeq(dds_1)

# Extract contrast between KO and WT

res_1 <- DESeq2::results(dds_1, contrast = c("batch", "KO", "WT"))

results_df <- as.data.frame(res_1)
'''
```

[102]:
```python
# Filter by padj less than 0.5
fresults_df_1 = results_df[results_df['padj'] <= 0.5]

# Sort by lfc greater than 0.5
sort_results_df = fresults_df_1.sort_values(by='log2FoldChange',␣
 ↪ascending=False)
sort_results_df
```

[102]:
```
                 baseMean  log2FoldChange     lfcSE      stat        pvalue  \
Sgip1           10.598842        4.734001  1.006120  4.705206  2.536103e-06
Zeb2            16.177872        1.268040  0.355644  3.565472  3.632015e-04
Nfkbiz          29.587975        0.848357  0.282643  3.001517  2.686384e-03
Lmna           216.492020        0.730526  0.090556  8.067128  7.197104e-16
Trdv4          313.407471        0.233317  0.071892  3.245396  1.172874e-03
Ltb            739.799805       -0.178925  0.045668 -3.917948  8.930581e-05
Syne1           36.736057       -0.632492  0.195791 -3.230444  1.235981e-03
Ms4a4b         208.537048       -0.832646  0.235221 -3.539844  4.003635e-04
Ier5l            7.334450       -1.782233  0.636229 -2.801243  5.090612e-03
B930036N10Rik   25.072521       -2.023115  0.321888 -6.285158  3.275207e-10
Chil3            2.021265       -2.511917  0.780635 -3.217788  1.291833e-03
5830411N06Rik   35.163712       -4.738309  1.439340 -3.292001  9.947712e-04
Itgb2           42.082130       -6.960996  1.303531 -5.340109  9.289050e-08

                        padj
```

```
Sgip1          6.333917e-04
Zeb2           5.713758e-02
Nfkbiz         2.236415e-01
Lmna           7.189907e-13
Trdv4          1.173219e-01
Ltb            1.784330e-02
Syne1          1.173219e-01
Ms4a4b         5.713758e-02
Ier5l          3.911940e-01
B930036N10Rik  1.635966e-07
Chil3          1.173219e-01
5830411N06Rik  1.173219e-01
Itgb2          3.093254e-05
```

[103]:
```
# Filter results dataframe to find upregulated genes in KO1
upregulated_genes_ko1 = sort_results_df[(sort_results_df['log2FoldChange'] > 0)␣
 ↪& (sort_results_df['padj'] < 0.05) ]

# Get the number of upregulated genes in KO1
num_upregulated_genes_ko1 = len(upregulated_genes_ko1)

print("Number of upregulated genes in KO1:", num_upregulated_genes_ko1)
upregulated_genes_ko1
```

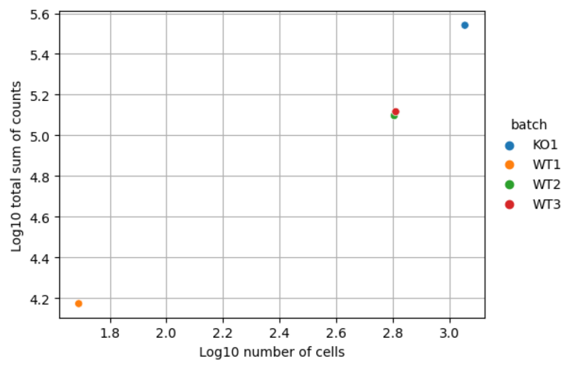
```
Number of upregulated genes in KO1: 2
```

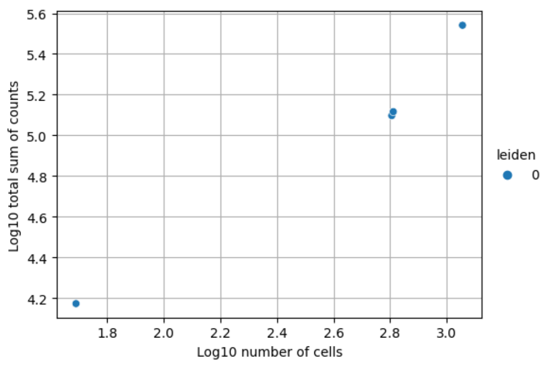
[103]:
```
          baseMean  log2FoldChange     lfcSE      stat        pvalue  \
Sgip1    10.598842        4.734001  1.006120  4.705206  2.536103e-06
Lmna    216.492020        0.730526  0.090556  8.067128  7.197104e-16

               padj
Sgip1  6.333917e-04
Lmna   7.189907e-13
```

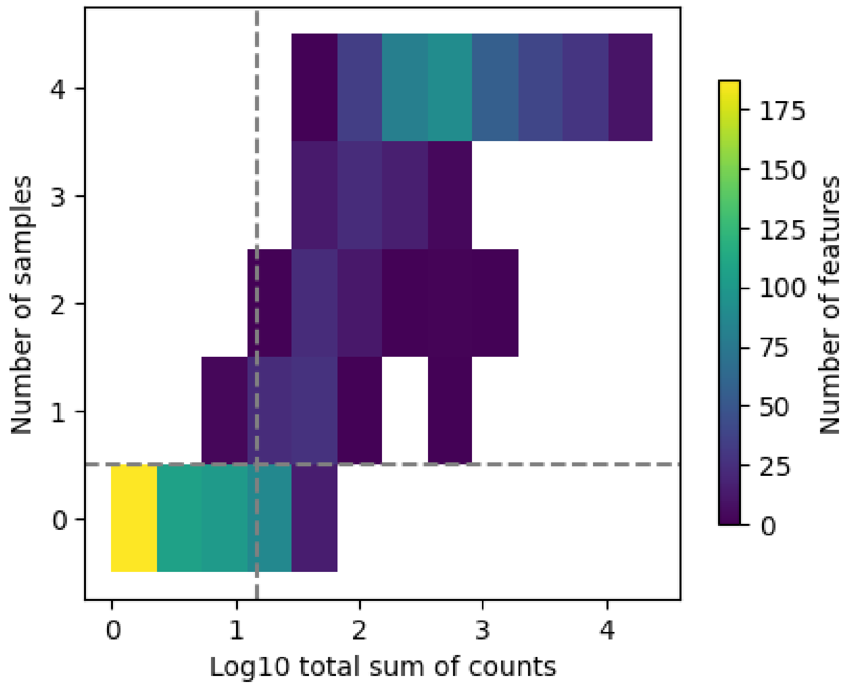
[104]:
```
# Get pseudo-bulk profile
pdata_2 = dc.get_pseudobulk(
    subset_data_e,
    sample_col='batch',
    groups_col='leiden',
    layer='counts',
    mode='sum',
    min_cells=10,
    min_counts=1000
)
dc.plot_psbulk_samples(pdata_2, groupby=['batch', 'leiden'], figsize=(12, 4))
dc.plot_filter_by_expr(pdata_2, group='batch', min_count=10, min_total_count=15)
from pydeseq2.dds import DeseqDataSet, DefaultInference
```

```python
# Build DESeq2 object
inference_2 = DefaultInference(n_cpus=8)
dds_2 = DeseqDataSet(
    adata=pdata_2,
    design_factors='condition',
    ref_level=['condition', 'WT'],
    refit_cooks=True,
    inference=inference_2,
)
# Compute LFCs
dds_2.deseq2()
# Extract contrast between COVID-19 vs normal
stat_res_2 = DeseqStats(
    dds_2,
    contrast=["condition", 'KO', 'WT'],
    inference=inference_1,
)
stat_res_2.summary()
results_df_2= stat_res_2.results_df
```

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

Fitting dispersions…
… done in 0.33 seconds.

Fitting dispersion trend curve…
… done in 0.04 seconds.

/home1/bioinfo-27/.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 0.26 seconds.

Fitting LFCs…
… done in 0.19 seconds.

Replacing 0 outlier genes.

Running Wald tests…
… done in 0.17 seconds.


Log2 fold change & Wald test p-value: condition KO vs WT
                 baseMean  log2FoldChange     lfcSE      stat     pvalue  \
1500009L16Rik  111.911522        1.090111  1.819166  0.599237  0.549015
1700055D18Rik    2.074705       -1.867374  1.857960 -1.005067  0.314865
2310001H17Rik   68.490509       -0.754630  0.632686 -1.192740  0.232971
2900026A02Rik    0.851514        3.636164  3.423473  1.062127  0.288178
5830411N06Rik   11.541553       -8.149039  3.467177 -2.350338  0.018756
…                    …              …         …         …         …
Zeb2            32.775848        2.991258  0.814759  3.671339  0.000241
Zfhx3            0.186219       -3.400754  6.434856 -0.528489  0.597160
Zfp36          385.067871       -0.899982  0.594738 -1.513242  0.130218
Zfp709           7.047067       -0.002973  1.329636 -0.002236  0.998216
Zkscan17         3.511354        0.444493  1.489878  0.298342  0.765442

                   padj
1500009L16Rik  0.811707
1700055D18Rik       NaN
2310001H17Rik  0.608313
2900026A02Rik       NaN
5830411N06Rik  0.115993
…                   …
Zeb2           0.004050
Zfhx3               NaN
Zfp36          0.419195
Zfp709         0.998216
Zkscan17            NaN

[987 rows x 6 columns]
```







```
[105]: # Filtering by padj less than 0.5
       fresults_df_2 = results_df_2[results_df_2['padj'] <= 0.5]

       # Sorting by lfc greater than 0.5
       sort_results_df_2 = fresults_df_2.sort_values(by='log2FoldChange',␣
        ↪ascending=False)
       sort_results_df_2
```

```
[105]:                   baseMean  log2FoldChange     lfcSE       stat        pvalue  \
       Klrc1            56.460835        4.865331  0.823887   5.905337  3.519256e-09
       Klrc2            31.636818        4.740199  0.422733  11.213217  3.511938e-29
       Dtx1             44.440105        3.664070  0.778380   4.707302  2.510166e-06
       Mir155hg         20.671743        3.215576  0.905547   3.550975  3.838073e-04
       Sgip1            34.558403        3.181764  1.518384   2.095493  3.612720e-02
       …                      …               …        …          …           …
       Lif              14.433813       -4.254499  1.240918  -3.428508  6.069078e-04
       Ccl4             19.950277       -5.005229  1.950499  -2.566128  1.028409e-02
       Ccr9              8.635662       -5.429845  1.968245  -2.758724  5.802759e-03
       Itgb2           135.562042       -7.758171  1.045164  -7.422922  1.145640e-13
       5830411N06Rik    11.541553       -8.149039  3.467177  -2.350338  1.875636e-02

                               padj
       Klrc1           1.837834e-07
       Klrc2           1.650611e-26
```

```
Dtx1            7.480529e-05
Mir155hg        5.819014e-03
Sgip1           1.806360e-01
…                      …
Lif             8.149905e-03
Ccl4            7.362313e-02
Ccr9            4.832462e-02
Itgb2           8.974176e-12
5830411N06Rik   1.159933e-01

[160 rows x 6 columns]
```

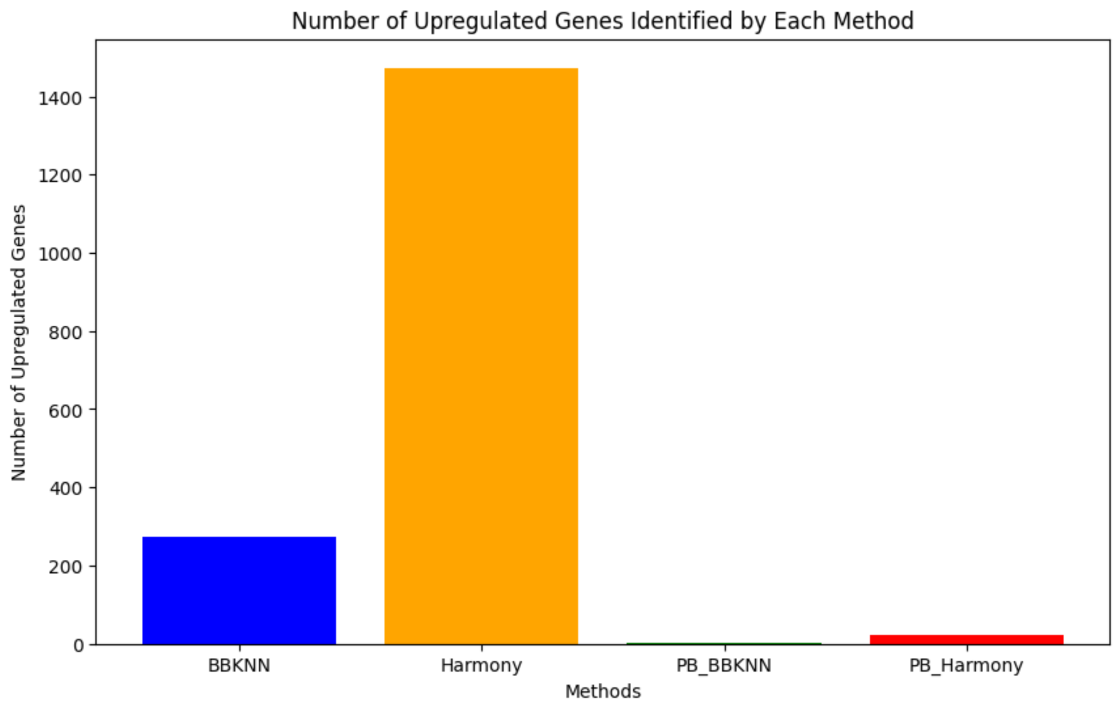
[106]:
```
# Filtering results dataframe to find upregulated genes in KO1
upregulated_genes_ko1_h =␣
  ↪sort_results_df_2[(sort_results_df_2['log2FoldChange'] > 0) &␣
  ↪(sort_results_df_2['padj'] < 0.05) ]

# Getting the number of upregulated genes in KO1
num_upregulated_genes_ko1_h = len(upregulated_genes_ko1_h)

print("Number of upregulated genes in KO1:", num_upregulated_genes_ko1_h)
upregulated_genes_ko1_h
```

```
Number of upregulated genes in KO1: 24
```

[106]:
```
              baseMean  log2FoldChange     lfcSE       stat        pvalue  \
Klrc1        56.460835        4.865331  0.823887   5.905337  3.519256e-09
Klrc2        31.636818        4.740199  0.422733  11.213217  3.511938e-29
Dtx1         44.440105        3.664070  0.778380   4.707302  2.510166e-06
Mir155hg     20.671743        3.215576  0.905547   3.550975  3.838073e-04
Zeb2         32.775848        2.991258  0.814759   3.671339  2.412828e-04
Itgb1       231.756836        2.725703  0.675243   4.036624  5.422578e-05
Cpm          69.327263        2.306817  0.665911   3.464153  5.319030e-04
AC163354.1  362.607330        2.231393  0.216003  10.330394  5.134910e-25
Cd7         144.161407        1.889377  0.421001   4.487818  7.195635e-06
Pou2f2       18.602983        1.838645  0.475219   3.869045  1.092623e-04
Serpinb6b    66.734489        1.734941  0.621500   2.791537  5.245837e-03
Dusp10       22.275589        1.687192  0.249038   6.774847  1.245380e-11
Plek         30.724781        1.577161  0.215468   7.319689  2.485467e-13
Rnf149       20.021475        1.508213  0.274549   5.493426  3.942102e-08
Itgae        18.488016        1.063956  0.259143   4.105675  4.031367e-05
Socs2       114.485695        1.029306  0.242871   4.238079  2.254407e-05
Txnip       222.072845        0.905185  0.309960   2.920325  3.496664e-03
Cks2         48.272278        0.874166  0.256597   3.406768  6.573700e-04
Icos        670.560425        0.839405  0.251508   3.337486  8.453995e-04
Rgcc        659.294556        0.739707  0.263569   2.806500  5.008291e-03
Socs1       200.263641        0.737499  0.256647   2.873591  4.058342e-03
```

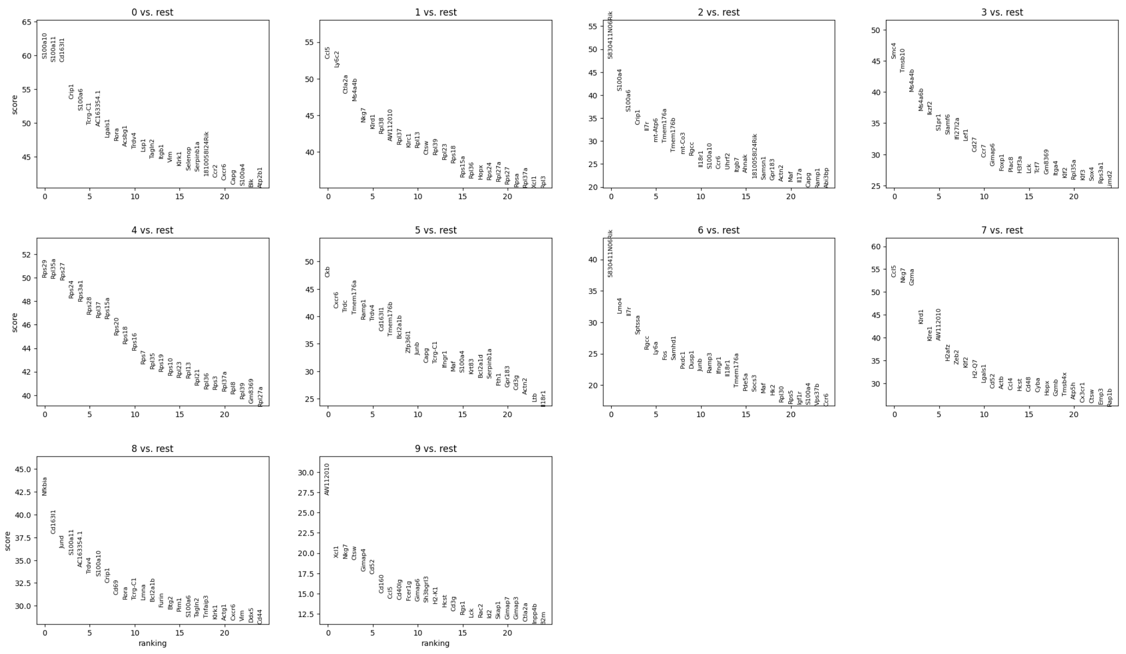
```
Gsr          71.923294        0.728258  0.227869   3.195953  1.393699e-03
Neat1       198.637115        0.659750  0.234527   2.813112  4.906460e-03
Klf4        128.551071        0.488669  0.155431   3.143969  1.666732e-03

                    padj
Klrc1       1.837834e-07
Klrc2       1.650611e-26
Dtx1        7.480529e-05
Mir155hg    5.819014e-03
Zeb2        4.050105e-03
Itgb1       1.108092e-03
Cpm         7.657297e-03
AC163354.1  1.206704e-22
Cd7         1.989381e-04
Pou2f2      2.054132e-03
Serpinb6b   4.651968e-02
Dusp10      7.316607e-10
Plek        1.668814e-11
Rnf149      1.650606e-06
Itgae       8.612467e-04
Socs2       5.297857e-04
Txnip       3.649231e-02
Cks2        8.582331e-03
Icos        9.933444e-03
Rgcc        4.526724e-02
Socs1       4.058342e-02
Gsr         1.559616e-02
Neat1       4.521639e-02
Klf4        1.821777e-02
```

[107]:
```python
import matplotlib.pyplot as plt

# Get the counts of upregulated genes for each method
counts = {
    'BBKNN': len(upregulated_genes_ko_bbkn),
    'Harmony': len(upregulated_genes_ko_har),
    'PB_BBKNN': len(upregulated_genes_ko1),
    'PB_Harmony': len(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()
```



[108]:
```
cluster_sizes_p = all_samples.obs['leiden'].value_counts()
largest_cluster_p = cluster_sizes_p.nlargest(10).index.tolist()
subset_data_p = all_samples[all_samples.obs['leiden'].isin(largest_cluster_p)].
↪copy()
```

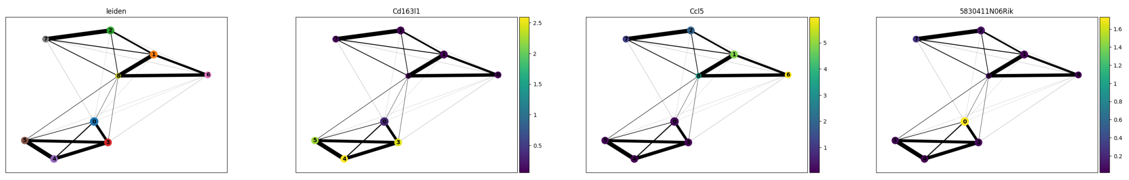
[109]:
```
sc.tl.rank_genes_groups(subset_data_p, 'leiden', method='wilcoxon',␣
↪use_raw=True)
sc.pl.rank_genes_groups(subset_data_p, n_genes=25, sharey=False)
```



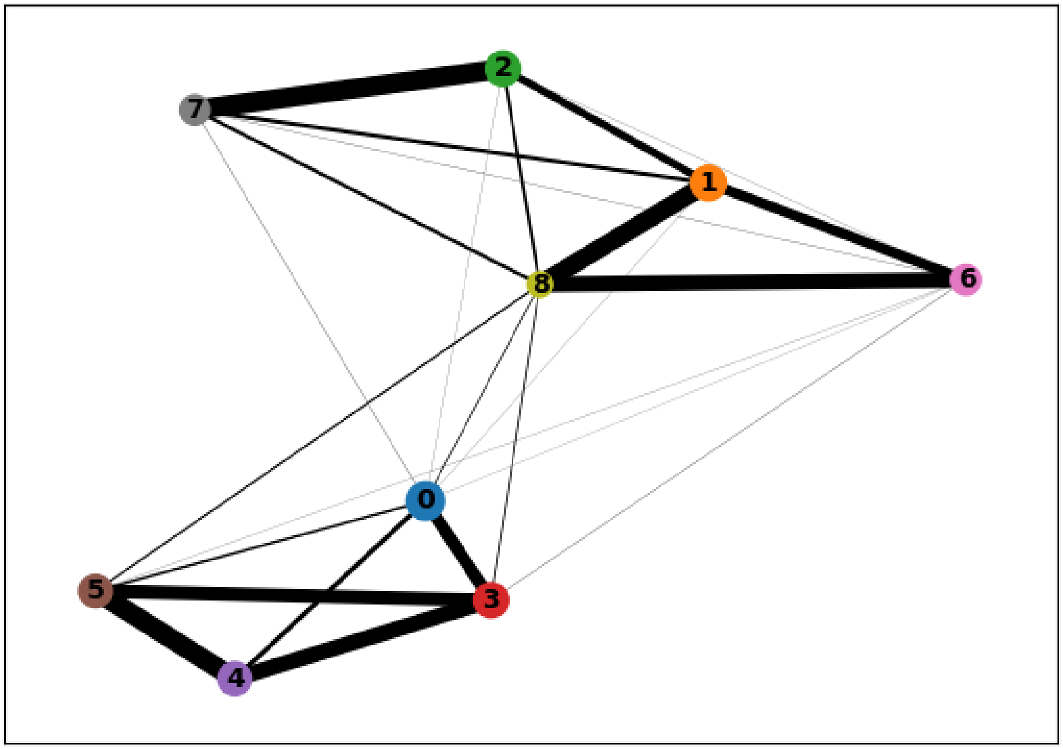
```
[110]: sc.tl.leiden(subset_data_p, resolution=1)
```

```
[111]: sc.tl.paga(subset_data_p, groups="leiden")
```

```
[112]: sc.pl.paga(subset_data_p, color=["leiden", "Cd163l1", "Ccl5", "5830411N06Rik"])
```



```
[115]: sc.pl.paga(subset_data_p, color=["leiden"])
```

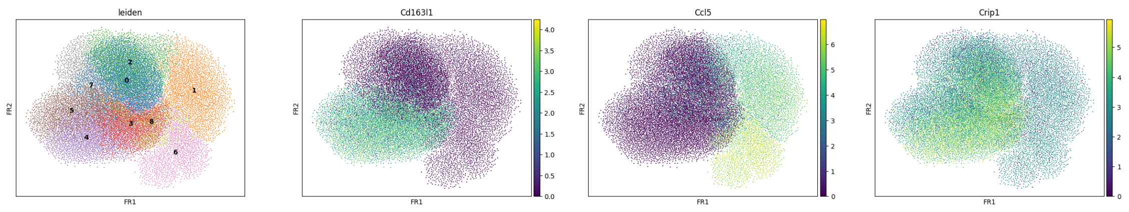


```
[113]: sc.tl.draw_graph(subset_data_p, init_pos="paga")
```

```
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`).
```

```
[114]: sc.pl.draw_graph(
           subset_data_p, color=["leiden", "Cd163l1", "Ccl5", "Crip1"], 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 object from seurat object

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