Reanalysis of Mouse Hematopoietic Cell Atlas using ScanPy

Load required modules and set global variables. Users will have to change base_dir, data_dir and results_file based on their own preferences.

In [1]:
```
import math
import numpy as np
import pandas as pd
import scanpy as sc

sc.settings.verbosity = 3
sc.logging.print_versions()

base_dir = '/Users/cordessf/UTHII'
data_dir = base_dir + '/data'
results_file = base_dir + '/output/MHCA.h5ad'
```

```
scanpy==1.4 anndata==0.6.19 numpy==1.16.3 scipy==1.2.1 pandas==0.24.
2 scikit-learn==0.20.3 statsmodels==0.9.0 python-igraph==0.7.1 louva
in==0.6.1
```

Load Datasets ¶

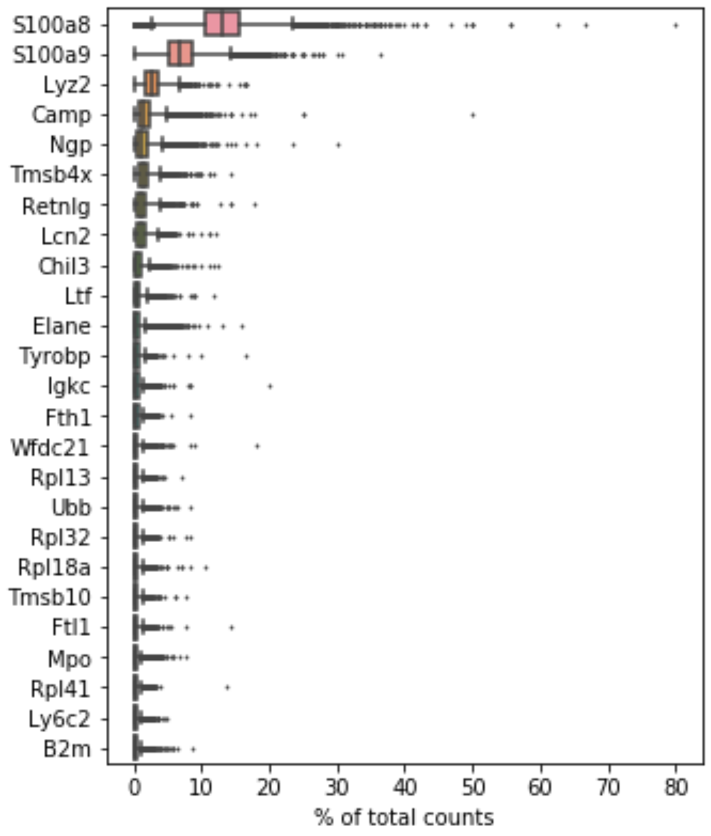
The data is originates from the paper by S. Lai et al (Cell Discovery 4(2018):34). The raw sequencing data and digital gene expression (DGE) data are accessible through the Gene Expression Omnibus (GEO), accesssion code GSE92274. In this (fairly rudimentary) implementation, the data is read from text files (compressed files won't work here). The expected format of the data is transposed from that expected by scanpy.

In [2]:
```
# Read the datasets and make the gene symbols unique
# ... First sample
BM_1_data = sc.read_text(data_dir + '/GSM2869488_Mouse_BM_1.txt')
BM_1_data = BM_1_data.transpose()
BM_1_data.var_names_make_unique()
```

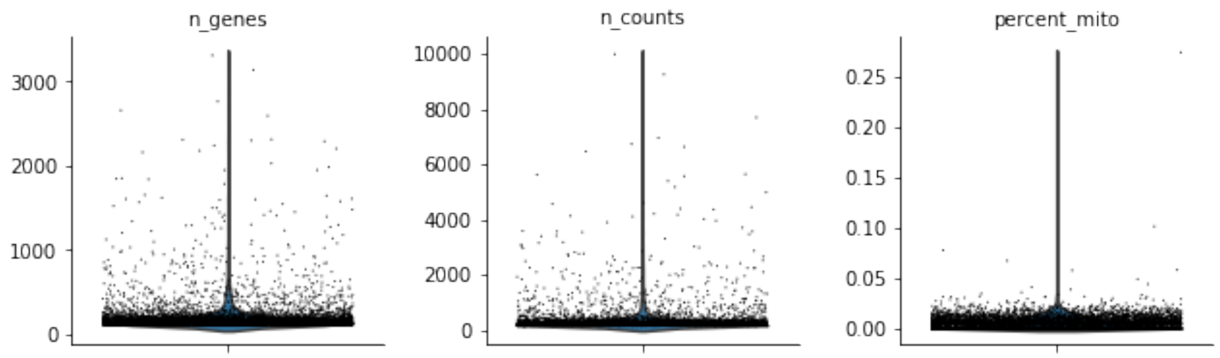
Preprocess Datasets

We will do only pretty rudimentary preprocessing here. A fuller attempt requires more time and effort than we have for this at the moment. We begin with a plot showing which genes have the highest fraction of all reads across all cells. Next we filter cells on the basis of minimum number of genes that must be detected in each cell. We then filter out genes on the basis of a the minimal number of cells in which they must be detected. Finally we compute the fraction of mitochondrial reads in each cell in preparation for filtering.

In [3]:
```
# ... Plot the genes that yields the highest fraction of counts in eac
h single
# cell, across all cells
sc.pl.highest_expr_genes(BM_1_data, n_top = 25)

# ... Basic filtering
# ... ... Filter cells on the basis of minimum numbers of genes detect
ed in each cell
sc.pp.filter_cells(BM_1_data, min_genes = 100)

# ... ... Filter genes on the basis of the minimum number of cells in
which they are detected
sc.pp.filter_genes(BM_1_data, min_cells = 5)

# ... Compute fraction of mitochondrial gene reads
mito_genes_1 = BM_1_data.var_names.str.startswith('mt-')
BM_1_data.obs['percent_mito'] = \
    np.sum(BM_1_data[:, mito_genes_1].X, axis = 1) / \
    np.sum(BM_1_data.X, axis = 1)
BM_1_data.obs['n_counts'] = BM_1_data.X.sum(axis = 1)

# Violin plots of percent mitochondial reads
sc.pl.violin(BM_1_data, ['n_genes', 'n_counts', 'percent_mito'],
             jitter = 0.4, multi_panel = True)

# Actual filtering
BM_1_data = BM_1_data[BM_1_data.obs['n_genes'] < 2500, :]
BM_1_data = BM_1_data[BM_1_data.obs['percent_mito'] < 0.05, :]

# Take regularized logarithm of expression count matrix
sc.pp.log1p(BM_1_data)
```

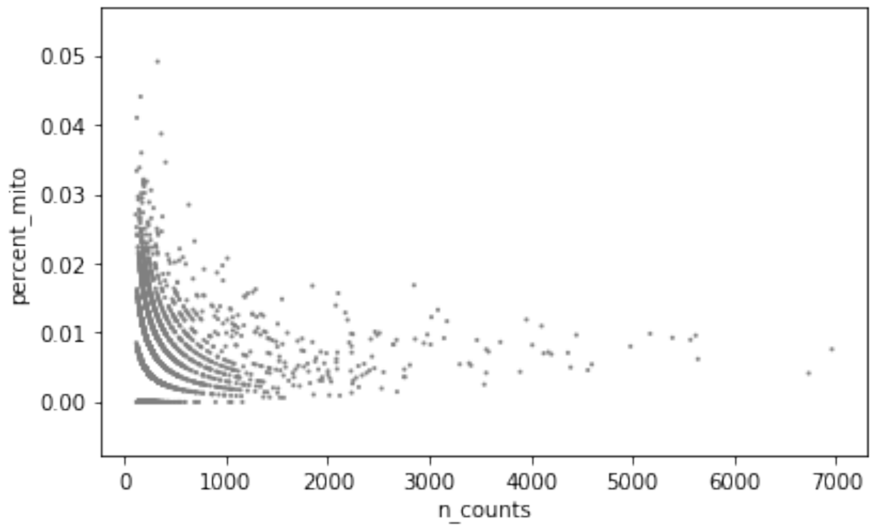


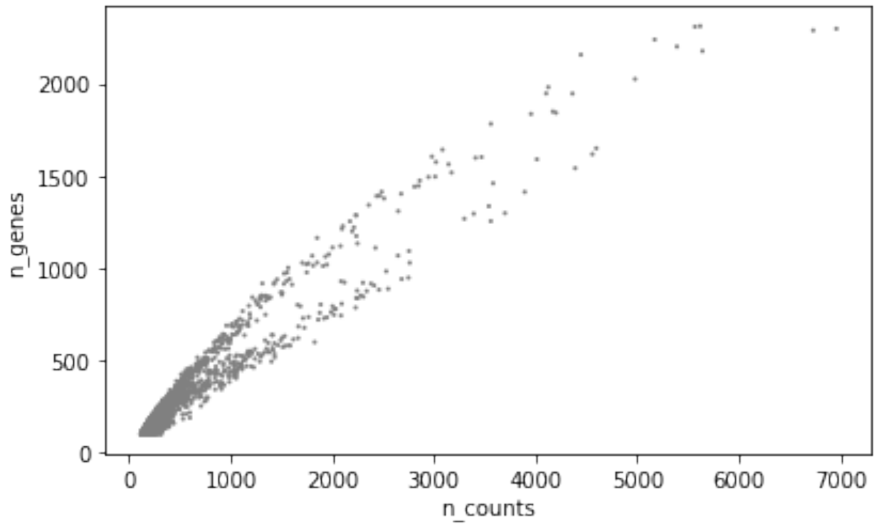
```
filtered out 2906 cells that have less than 100 genes expressed
filtered out 4269 genes that are detected in less than 5 cells
```



Plot scatter plot of number of counts versus percentage of mitochondrial genes for each cell.

In [4]:
```
sc.pl.scatter(BM_1_data, x='n_counts', y='percent_mito')
sc.pl.scatter(BM_1_data, x='n_counts', y='n_genes')
```





Total-count normalize (to correct for library size) to 10,000 reads per cell, so that counts become comparable between cells.

In [5]:
```
sc.pp.normalize_per_cell(BM_1_data, counts_per_cell_after=1e4)
```

Take the regularized logarithm of the data to improve the dynamic range.

In [6]:
```
sc.pp.log1p(BM_1_data)
```

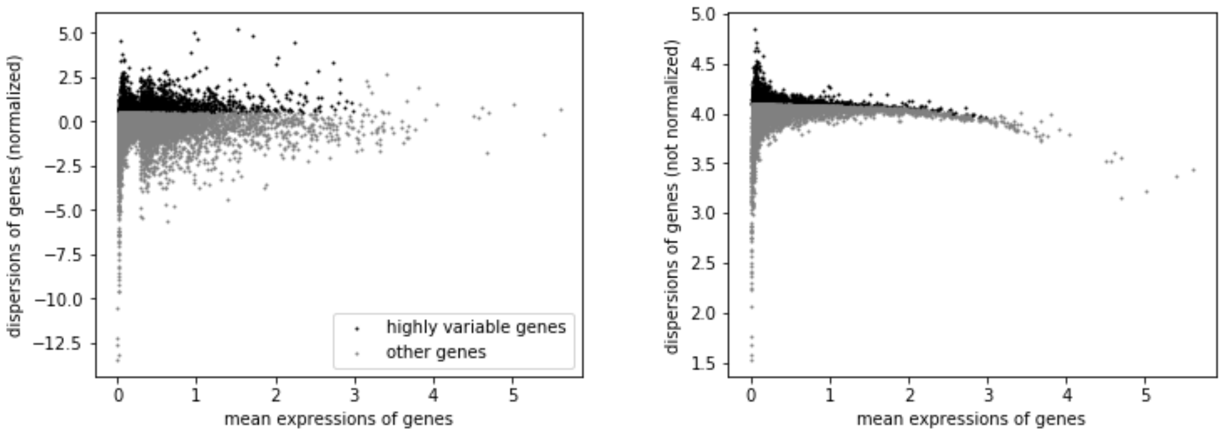
Set the .raw attribute of AnnData object to the regularized logarithm of the raw gene expression

```
In [7]: BM_1_data.raw = BM_1_data
```

Identify highly-variable genes and plot them.

```
In [8]: sc.pp.highly_variable_genes(BM_1_data, min_mean=0.0125, max_mean=3, mi
        n_disp=0.5)
        sc.pl.highly_variable_genes(BM_1_data)
```

```
--> added
    'highly_variable', boolean vector (adata.var)
    'means', float vector (adata.var)
    'dispersions', float vector (adata.var)
    'dispersions_norm', float vector (adata.var)
```



Filter on the highly variable genes.

```
In [9]: BM_1_data = BM_1_data[:, BM_1_data.var['highly_variable']]
```

Regress out total counts per cell and percentage of mitochondrial genes expressed.

```
In [10]: sc.pp.regress_out(BM_1_data, ['n_counts', 'percent_mito'])
```

```
regressing out ['n_counts', 'percent_mito']
    finished (0:00:38.13)
```

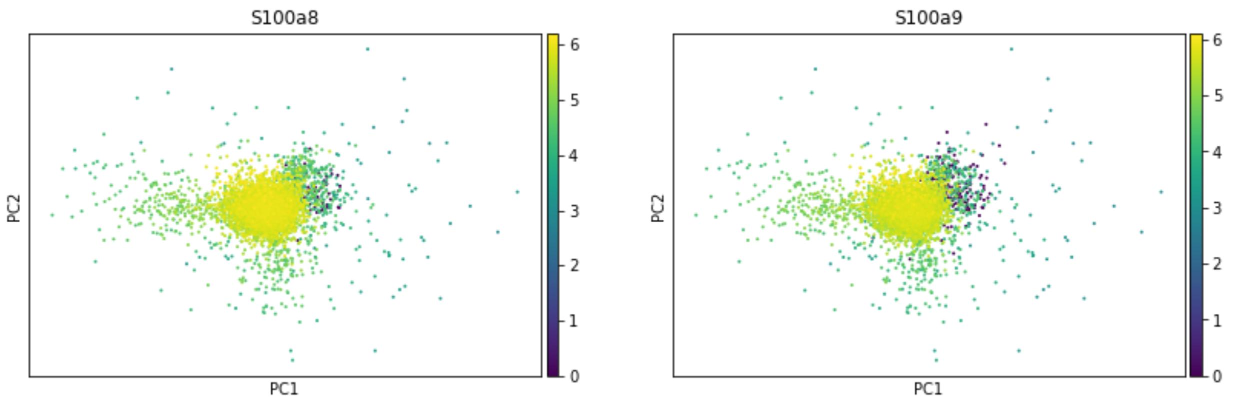
Scale the data to unit variance.

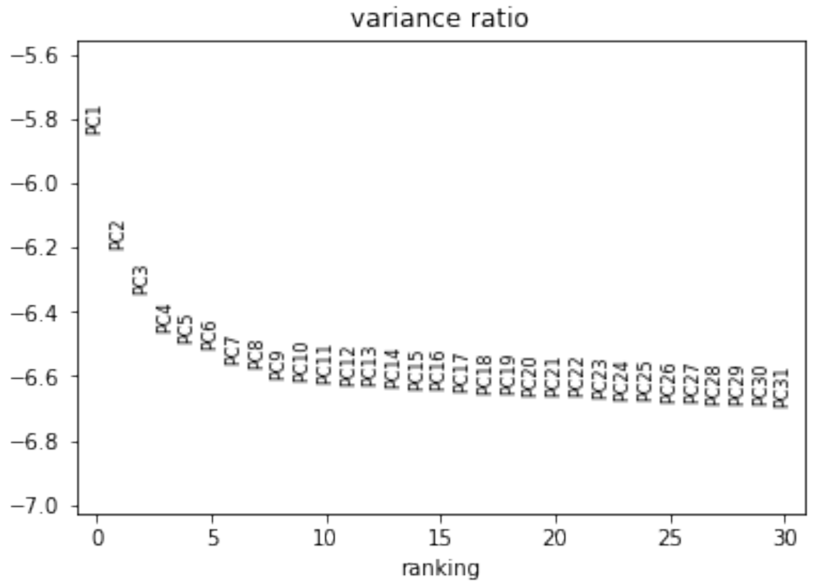
```
In [11]: sc.pp.scale(BM_1_data, max_value=10)
```

Principal Components Analysis

Denoise data and analyze principal axes of variation. Plot expression of two principal neutrophil genes onto PCA dimensional reduction. Compute the contribution of single PCs to the variance.

```
In [12]: sc.tl.pca(BM_1_data, svd_solver='arpack')
         sc.pl.pca(BM_1_data, color = ['S100a8', 'S100a9'])
         sc.pl.pca_variance_ratio(BM_1_data, log=True)
```



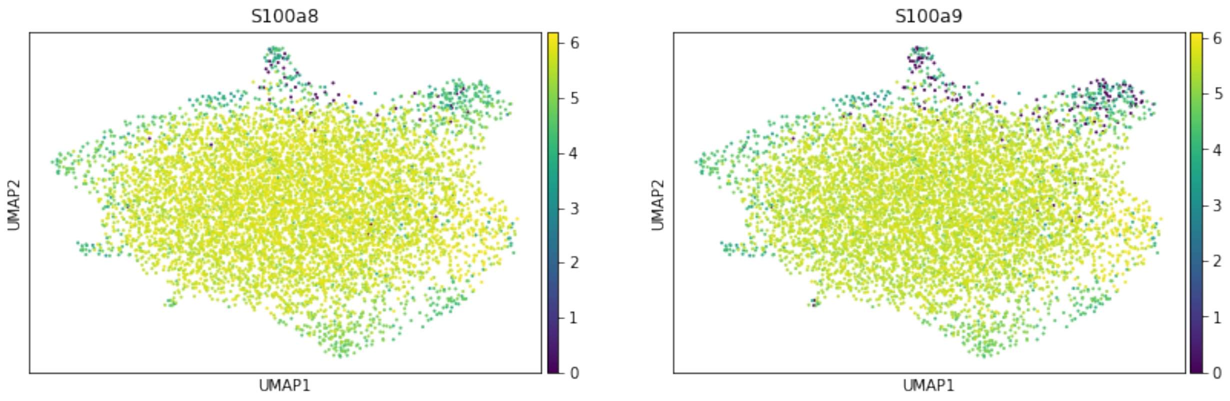


Computation of Neighborhood Graph and UMAP Dimensional Reduction

Use the PCA dimensional reduction to compute the neighborhood graph. Although overly generous, we'll use the first 30 principal components.

```
In [13]: nn_1 = math.ceil(math.sqrt(BM_1_data.shape[1]))
         sc.pp.neighbors(BM_1_data, n_neighbors = nn_1, n_pcs = 30)
         sc.tl.umap(BM_1_data)
         sc.pl.umap(BM_1_data, color=['S100a8', 'S100a9'])
```

```
computing neighbors
    using 'X_pca' with n_pcs = 30
    finished (0:00:15.17) --> added to `.uns['neighbors']`
    'distances', distances for each pair of neighbors
    'connectivities', weighted adjacency matrix
computing UMAP
    finished (0:00:25.92) --> added
    'X_umap', UMAP coordinates (adata.obsm)
```

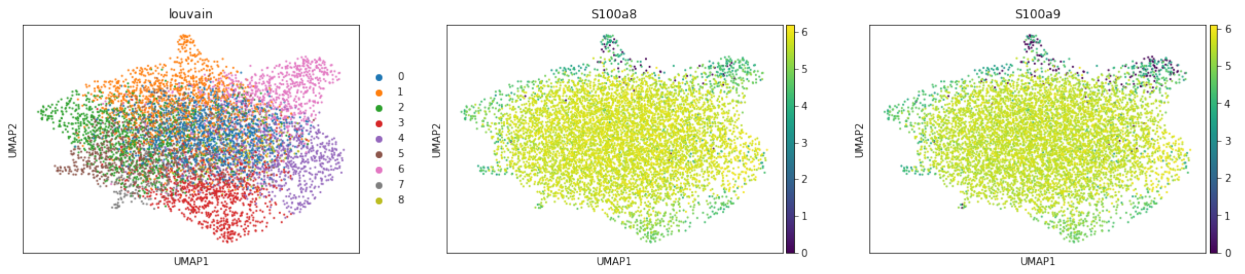


Clustering of the neighborhood graph

We cluster the dimensionally reduced data and highlight expression of some prominent neutrophil genes.

In [14]:
```python
sc.tl.louvain(BM_1_data)
sc.pl.umap(BM_1_data, color=['louvain', 'S100a8', 'S100a9'])
```

```
running Louvain clustering
    using the "louvain" package of Traag (2017)
    finished (0:00:04.69) --> found 9 clusters and added
    'louvain', the cluster labels (adata.obs, categorical)
```

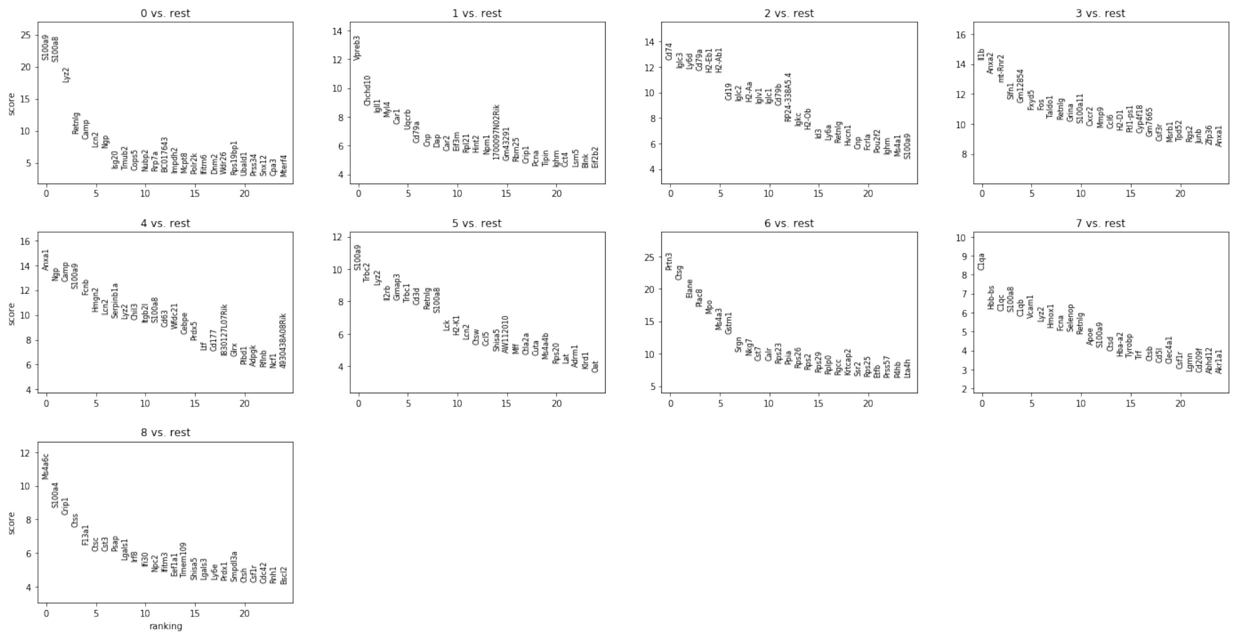


Marker Genes via the t-Test

The simplest and fastest method is the t-test. It is a parametric test that assumes that transformed expression levels are normally distributed.

In [15]:
```python
sc.tl.rank_genes_groups(BM_1_data, 'louvain', method='t-test')
sc.pl.rank_genes_groups(BM_1_data, n_genes=25, sharey=False)
```

```
ranking genes
    finished (0:00:04.82) --> added to `.uns['rank_genes_groups']`
    'names', sorted np.recarray to be indexed by group ids
    'scores', sorted np.recarray to be indexed by group ids
    'logfoldchanges', sorted np.recarray to be indexed by group ids
    'pvals', sorted np.recarray to be indexed by group ids
    'pvals_adj', sorted np.recarray to be indexed by group ids
```

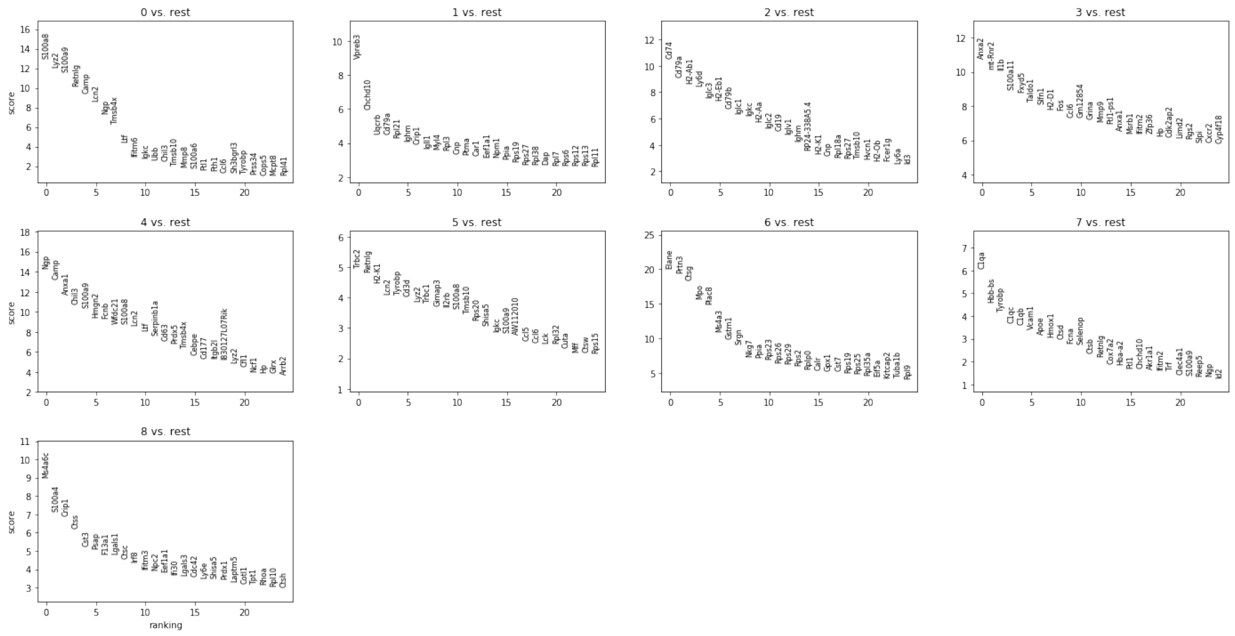


Marker Genes via the Wilcox Test

A perhaps better test, which is non-parametric is the Wilkoxon rank-sum test

In [16]:
```
sc.tl.rank_genes_groups(BM_1_data, 'louvain', method='wilcoxon')
sc.pl.rank_genes_groups(BM_1_data, n_genes=25, sharey=False)
```

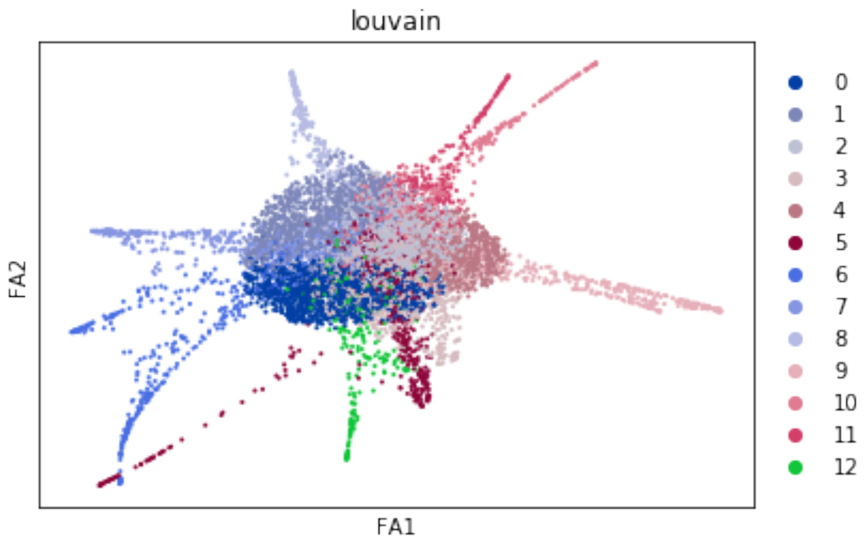
```
ranking genes
    finished (0:00:11.41) --> added to `.uns['rank_genes_groups']`
    'names', sorted np.recarray to be indexed by group ids
    'scores', sorted np.recarray to be indexed by group ids
    'logfoldchanges', sorted np.recarray to be indexed by group ids
    'pvals', sorted np.recarray to be indexed by group ids
    'pvals_adj', sorted np.recarray to be indexed by group ids
```



Trajectory Inference

In [17]:
```
sc.tl.diffmap(BM_1_data)
nn_1 = math.ceil(math.sqrt(BM_1_data.shape[1]))
sc.pp.neighbors(BM_1_data, n_neighbors = nn_1, use_rep = 'X_diffmap')
sc.tl.louvain(BM_1_data, resolution = 0.9)
sc.tl.draw_graph(BM_1_data)
sc.pl.draw_graph(BM_1_data, color = ['louvain'])
```

```
computing Diffusion Maps using n_comps=15(=n_dcs)
    eigenvalues of transition matrix
    [1.         0.86120975 0.8557562  0.8337169  0.76063544 0.740400
97
     0.71323967 0.70561975 0.6842523  0.6668384  0.6533583  0.640318
04
     0.6343892  0.6315831  0.62905353]
    finished (0:00:00.46) --> added
    'X_diffmap', diffmap coordinates (adata.obsm)
    'diffmap_evals', eigenvalues of transition matrix (adata.uns)
computing neighbors
    finished (0:00:09.61) --> added to `.uns['neighbors']`
    'distances', distances for each pair of neighbors
    'connectivities', weighted adjacency matrix
running Louvain clustering
    using the "louvain" package of Traag (2017)
    finished (0:00:03.99) --> found 13 clusters and added
    'louvain', the cluster labels (adata.obs, categorical)
drawing single-cell graph using layout "fa"
    finished (0:01:02.49) --> added
    'X_draw_graph_fa', graph_drawing coordinates (adata.obsm)
```

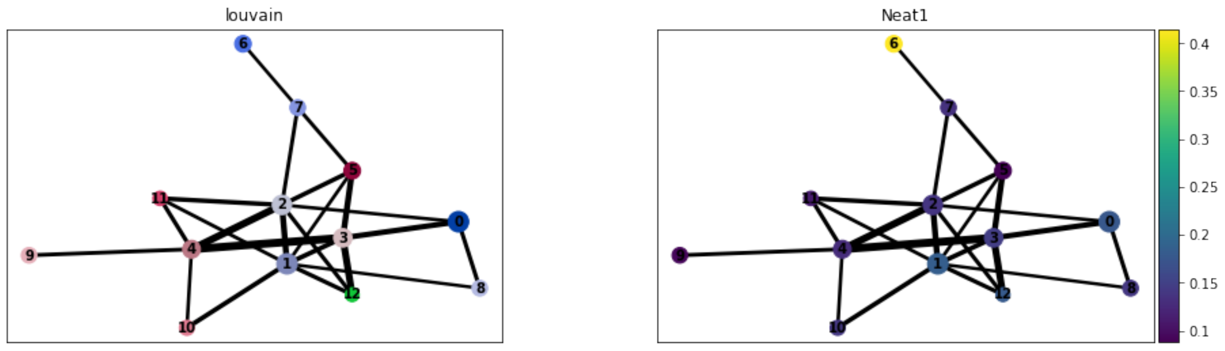


Coarse-grained Visualization

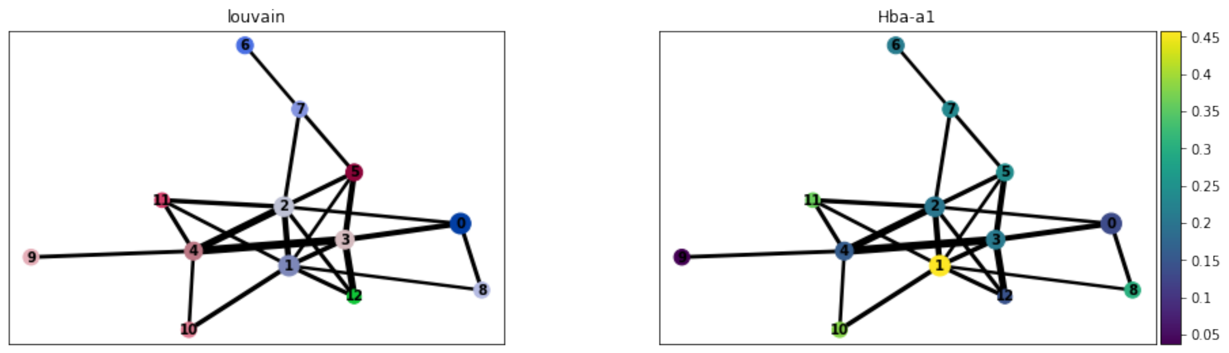
One of the nice features of scanpy/PAGA is that its clustering

In [18]:
```
# Sample 1
threshold_1 = 0.32
sc.tl.paga(BM_1_data, groups='louvain')

# ... HSC lineage
sc.pl.paga(BM_1_data, color = ['louvain', 'Neat1'], threshold = thresh
old_1)

# ... Erythoid lineage
sc.pl.paga(BM_1_data, color = ['louvain', 'Hba-a1'], threshold = thres
hold_1)

# ... Neutrophils
sc.pl.paga(BM_1_data, color = ['louvain', 'Elane', 'Mpo', 'Gfi1'], thr
eshold = threshold_1)

# ... Monocytes
sc.pl.paga(BM_1_data, color = ['louvain', 'Irf8', 'Csf1r'], threshold
= threshold_1)

# ... Megakaryocytes
sc.pl.paga(BM_1_data, color = ['louvain', 'Pbx1', 'Itga2b'], threshold
= threshold_1)

# ... B cell
sc.pl.paga(BM_1_data, color = ['louvain', 'Cd19', 'Cd79a'], threshold
= threshold_1)
```

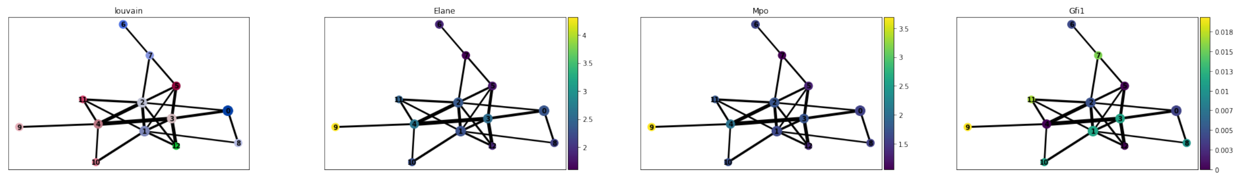
```
running PAGA
    finished (0:00:00.86) --> added
    'paga/connectivities', connectivities adjacency (adata.uns)
    'paga/connectivities_tree', connectivities subtree (adata.uns)
--> added 'pos', the PAGA positions (adata.uns['paga'])
```



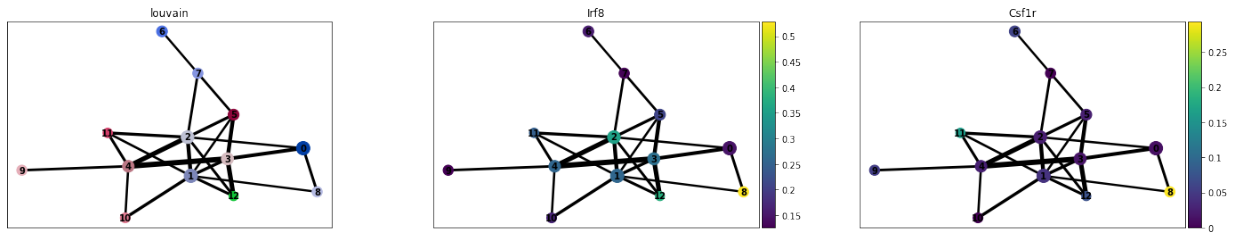
```
--> added 'pos', the PAGA positions (adata.uns['paga'])
```



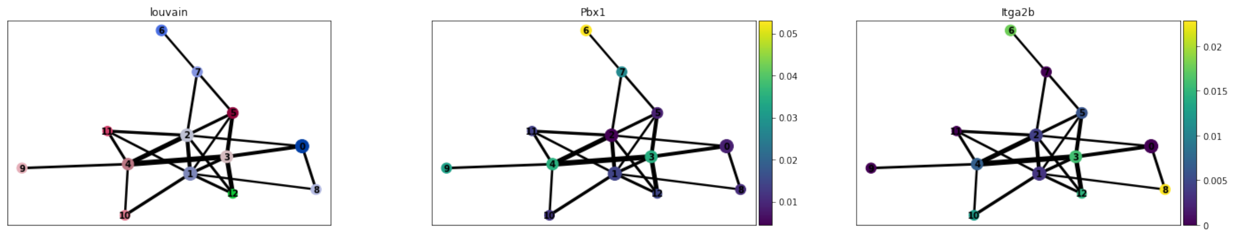
```
--> added 'pos', the PAGA positions (adata.uns['paga'])
```



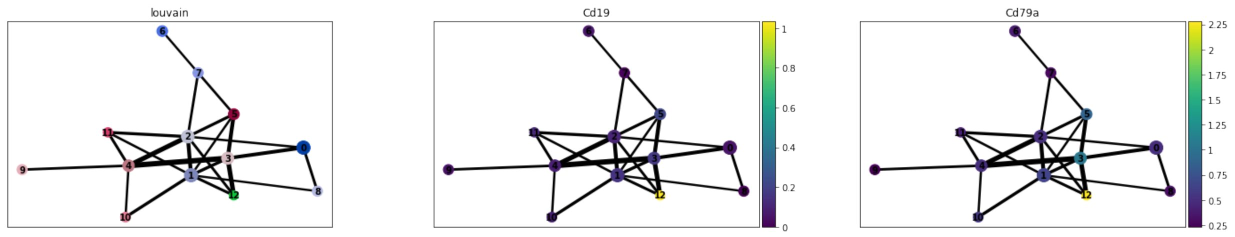
```
--> added 'pos', the PAGA positions (adata.uns['paga'])
```



```
--> added 'pos', the PAGA positions (adata.uns['paga'])
```



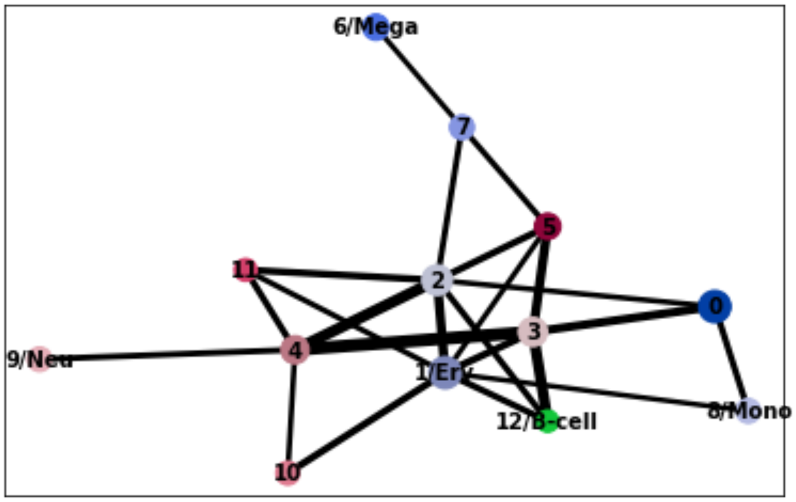
```
--> added 'pos', the PAGA positions (adata.uns['paga'])
```



Using what we have learned from the distributions of marker genes, we now add some annotation

In [19]:
```
BM_1_data.obs['louvain_anno'] = BM_1_data.obs['louvain']
BM_1_data.obs['louvain_anno'].cat.categories = ['0', '1/Ery', '2', '3'
, '4', '5', '6/Mega', '7', '8/Mono', '9/Neu', '10', '11', '12/B-cell']
sc.tl.paga(BM_1_data, groups = 'louvain_anno')
sc.pl.paga(BM_1_data, threshold = threshold_1)
```

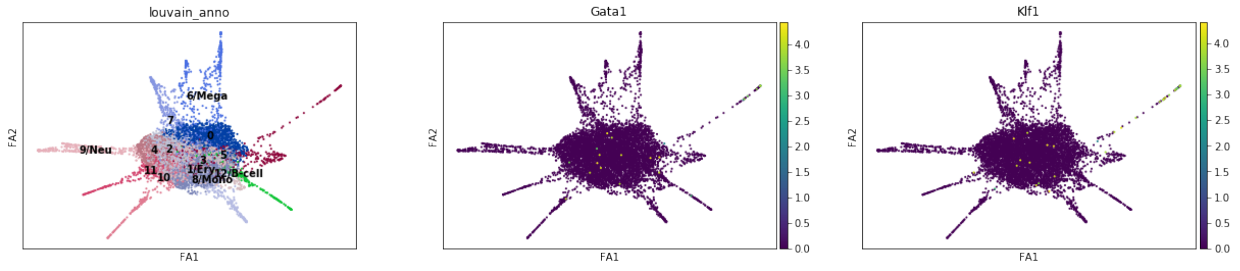
```
running PAGA
    finished (0:00:00.79) --> added
    'paga/connectivities', connectivities adjacency (adata.uns)
    'paga/connectivities_tree', connectivities subtree (adata.uns)
--> added 'pos', the PAGA positions (adata.uns['paga'])
```

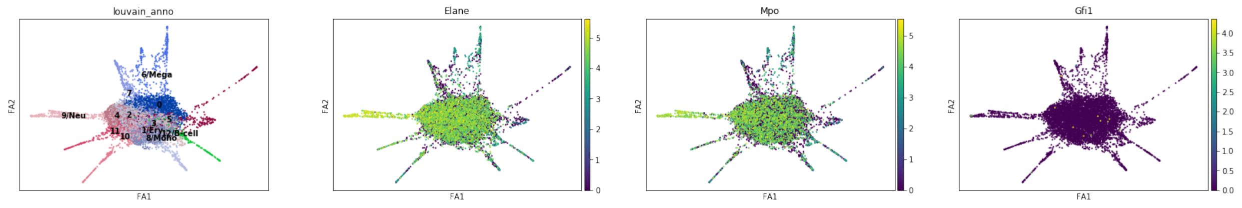


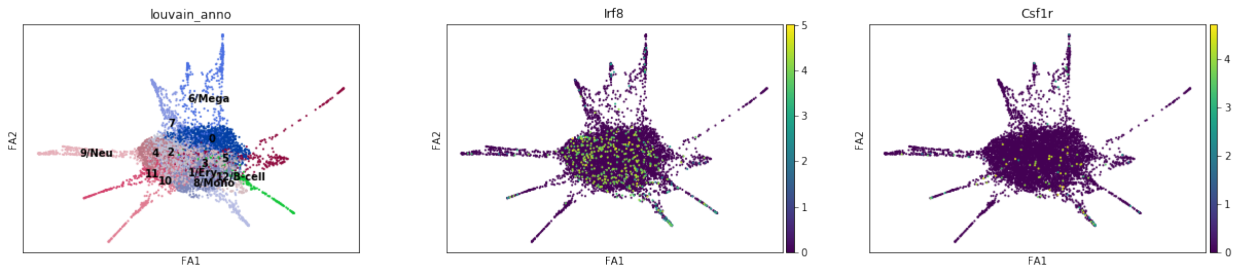
Next we recompute the embedding using PAGA for initialization. At the same time we can also plot the expression of marker genes at single cell resolution.

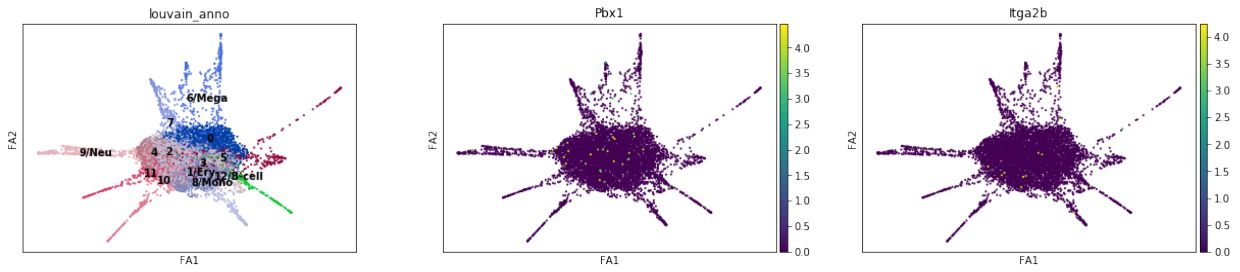
In [20]:
```
sc.tl.draw_graph(BM_1_data, init_pos='paga')
sc.pl.draw_graph(BM_1_data, color=['louvain_anno', 'Gata1', 'Klf1'], l
egend_loc = 'on data')

sc.pl.draw_graph(BM_1_data, color=['louvain_anno', 'Elane', 'Mpo', 'Gf
i1'], legend_loc = 'on data')

sc.pl.draw_graph(BM_1_data, color=['louvain_anno', 'Irf8', 'Csf1r'], l
egend_loc = 'on data')

sc.pl.draw_graph(BM_1_data, color=['louvain_anno', 'Pbx1', 'Itga2b'],
legend_loc = 'on data')

sc.pl.draw_graph(BM_1_data, color=['louvain_anno', 'Cd19', 'Cd79a'], l
egend_loc = 'on data')
```

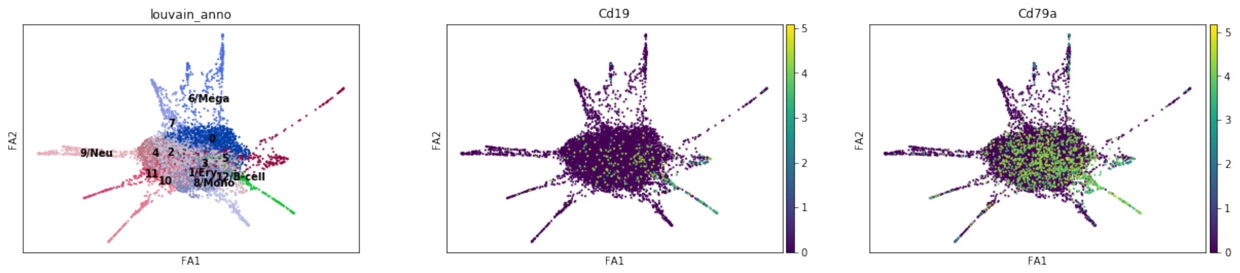
```
drawing single-cell graph using layout "fa"
    finished (0:00:58.39) --> added
    'X_draw_graph_fa', graph_drawing coordinates (adata.obsm)
```


louvain_anno
Gata1
Klf1
FA1
FA2
6/Mega
9/Neu
1/Ery
12/B-cell
8/Mono


louvain_anno
Elane
Mpo
Gfi1
FA1
FA2


louvain_anno
Irf8
Csf1r
FA1
FA2


louvain_anno
Pbx1
Itga2b
FA1
FA2


louvain_anno
Cd19
Cd79a
FA1
FA2