Welcome to spSAM's documentation!

spSAM: 10X visium **sp**ot **S**plit **A**lign **M**ap

In [1]:
```
import spsam as ss
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

plt.rcParams['figure.figsize'] = (6, 6)
```

Load Data

Load anndata object intergrating 10X Visium data with scRNA-seq reference of cell types through **cell2location** package.

In [2]:
```
adata = ss.read_h5ad('demo.h5ad')
```

In [3]:
```
adata.obs.keys()
```

Out[3]:
```
Index(['in_tissue', 'array_row', 'array_col', 'sample', 'n_genes_by_counts',
       'log1p_n_genes_by_counts', 'total_counts', 'log1p_total_counts',
       'pct_counts_in_top_50_genes', 'pct_counts_in_top_100_genes',
       'pct_counts_in_top_200_genes', 'pct_counts_in_top_500_genes',
       'total_counts_mt', 'log1p_total_counts_mt', 'pct_counts_mt', '_indices',
       '_scvi_batch', '_scvi_labels', 'B cells', 'B-cell lineage',
       'Cycling cells', 'DC', 'ILC', 'Macrophages', 'Monocytes',
       'Plasma cells', 'T cells', 'pDC', 'clusters'],
      dtype='object')
```

Through `adata.obs.keys()`, we can see total 10 cell types `['B cells', 'B-cell lineage', 'Cycling cells', 'DC', 'ILC', 'Macrophages', 'Monocytes', 'Plasma cells', 'T cells', 'pDC']` were mapped to 10X Visium data.

Score Genes

This reproduces the approach in Seurat (**Satija**) and has been implemented for Scanpy by Davide Cittar.

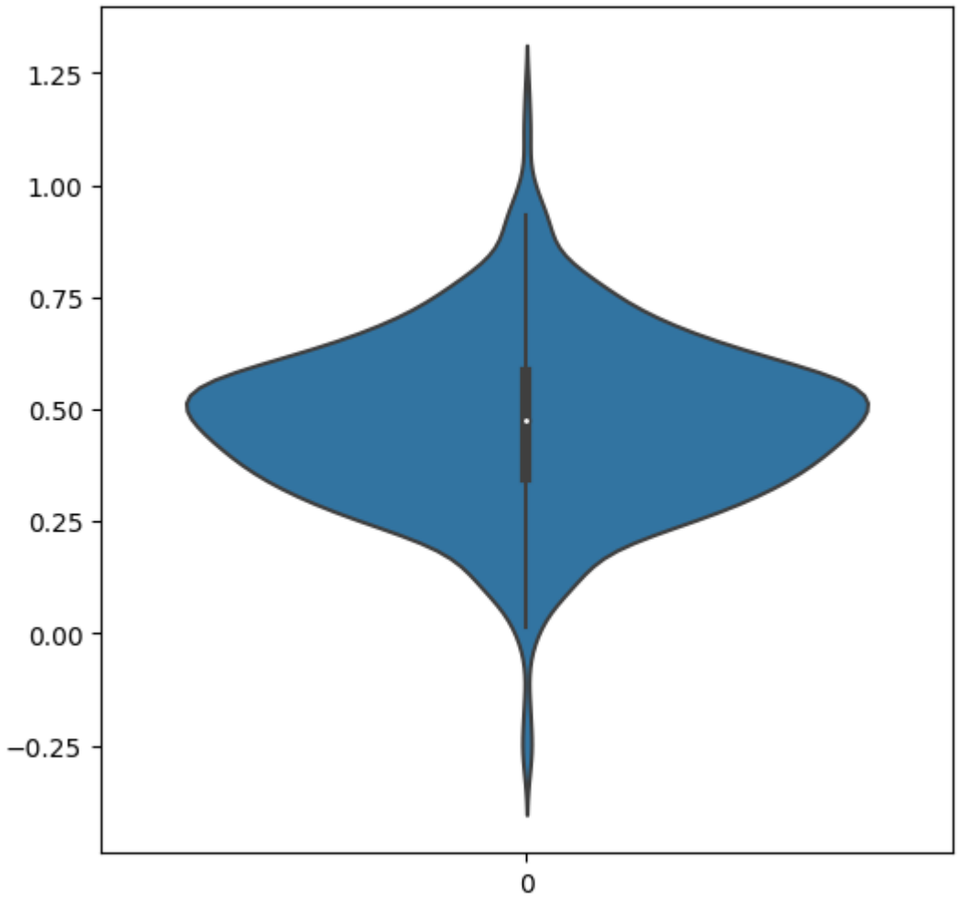
Here we use 15 top-ranked hypoxia-associated genes, `['VEGFA', 'SLC2A1', 'PGAM1', 'ENO1', 'LDHA', 'TP11', 'P4HA1', 'MRPS1', 'CDKN3', 'ADM', 'NDRG1', 'TUBB6', 'ALDOA', 'MIF', 'ACOT7']`, which are collectively considered to be hypoxia signature (**Buffa signature**) to assess hypoxia statu. Custom gene sets are also supported.

In [4]:
```
hypoxia_gene_lt = ['VEGFA', 'SLC2A1', 'PGAM1', 'ENO1', 'LDHA', 'TP11', 'P4HA1', 'MRPS1', 'CDKN3', 'ADM', 'NDRG1', 'TUBB6', 'ALDOA', 'MIF', 'ACOT7']
ss.tl.score_genes(adata, gene_list=hypoxia_gene_lt, ctrl_size=len(hypoxia_gene_lt), score_name='hypoxia_score')
```

```
WARNING:root:genes are not in var_names and ignored: ['VEGFA', 'LDHA', 'TP11', 'MRPS1', 'ALDOA']
computing score 'hypoxia_score'
finished
hypoxia_score, score of gene set (adata.obs).
104 total control genes are used.
```

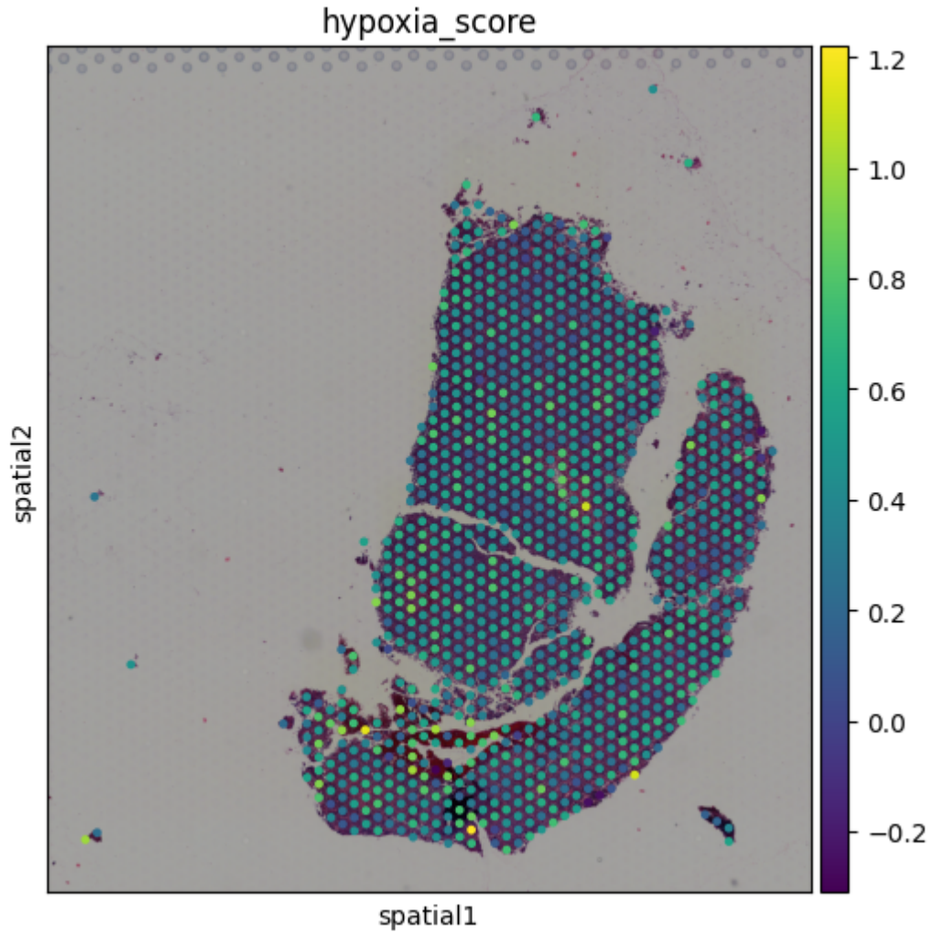
Plot hypoxia_score distribution

In [5]:
```
ss.pl.violin(adata, 'hypoxia_score')
```



Show score in visium image

In [6]:
```
ss.pl.spatial(adata, img_key='hires', color=['hypoxia_score'])
```



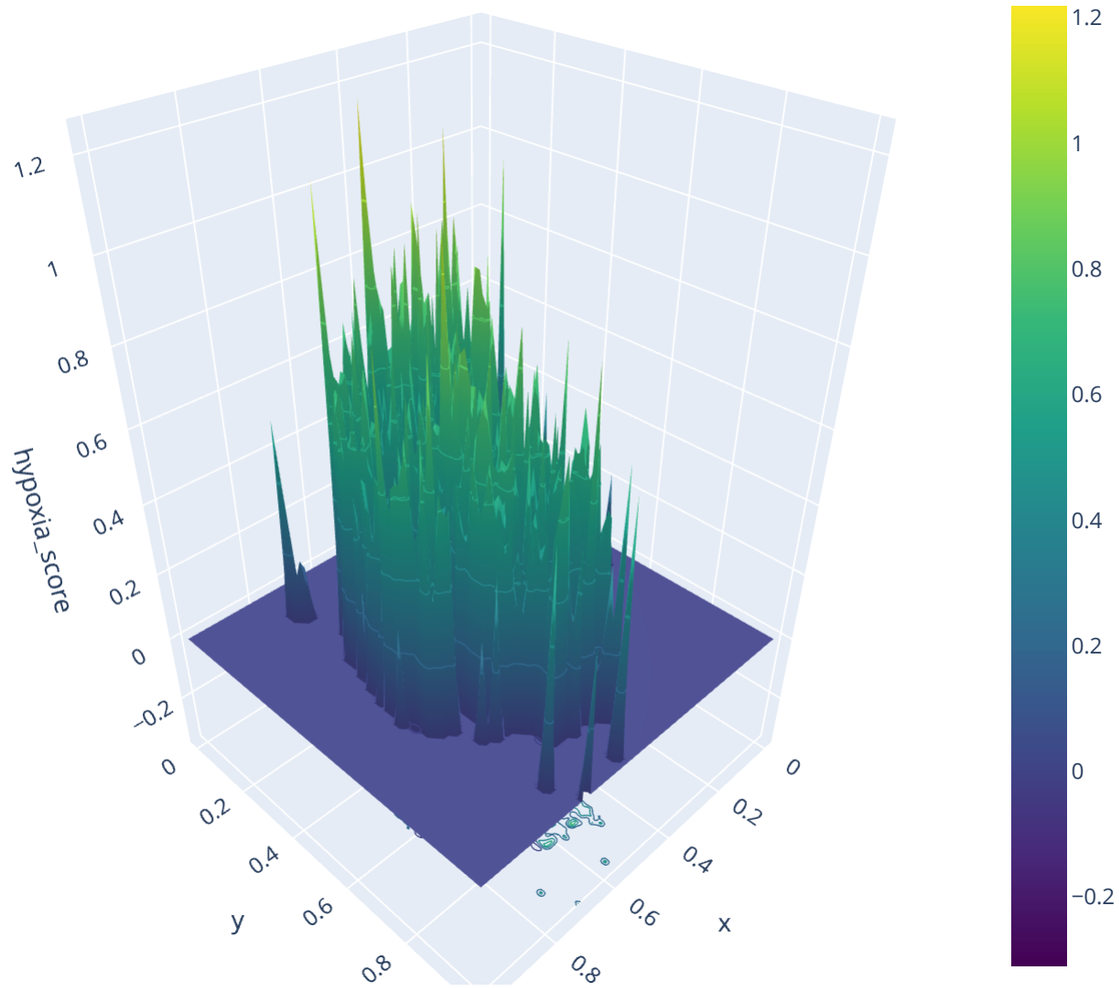
In [7]:
```
adata.obs['hypoxia_score']
```

Out[7]:
```
AAACCGGGTAGGTACC-1    0.456343
AAACCGTTCGTCCAGG-1    0.357551
AAACCTCATGAAGTTG-1    0.857315
AAACGAGACGGTTGAT-1    0.499969
AAACTGCTGGCTCCAA-1    0.822195
                        ...
TTGTGGTAGGAGGGAT-1    0.690741
TTGTGTATGCCACCAA-1    0.468725
TTGTGTTTCCCGAAAG-1    0.570668
TTGTTTCACATCCAGG-1    0.631536
TTGTTTCCATACAACT-1    0.762182
Name: hypoxia_score, Length: 1090, dtype: float64
```

Additionally, you can use `ss.pl.plot_3d_score()` function to plot hypoxia_score in 3D spatial dimension of 10x visium image

In [8]:
```
ss.pl.plot_3d_score(adata, 'hypoxia_score')
```

hypoxia_score



Score Lever

Divide spot into three parts based on the distribution of the score values.
lever1(default): hypoxia_score >= 95% distribution of the score values.
lever2(default): hypoxia_score >= 50% distribution of the score values.
background: hypoxia_score < 50% distribution of the score values.
If using other thresholds to divide spot, pass two values through (lever1, lever2) parameter, like `ss.pp.score_lever(adata, 'hypoxia_score', lever1=0.8, lever2=0.6)`, then background part will set hypoxia_score < 80% automatically.

In [9]:
```
ss.pp.score_lever(adata, 'hypoxia_score')
```

```
hypoxia_score median:0.4765155938955453; mean:0.4701082144745993; std:0.18736050562146309
hypoxia_score lever1 value:0.7705623546013465; lever2 value:0.4765155938955453
score lever definition finished, score_type key is added to adata.obs, use adata.obs_keys() to check
```

In [10]:
```
adata.obs.head()
```

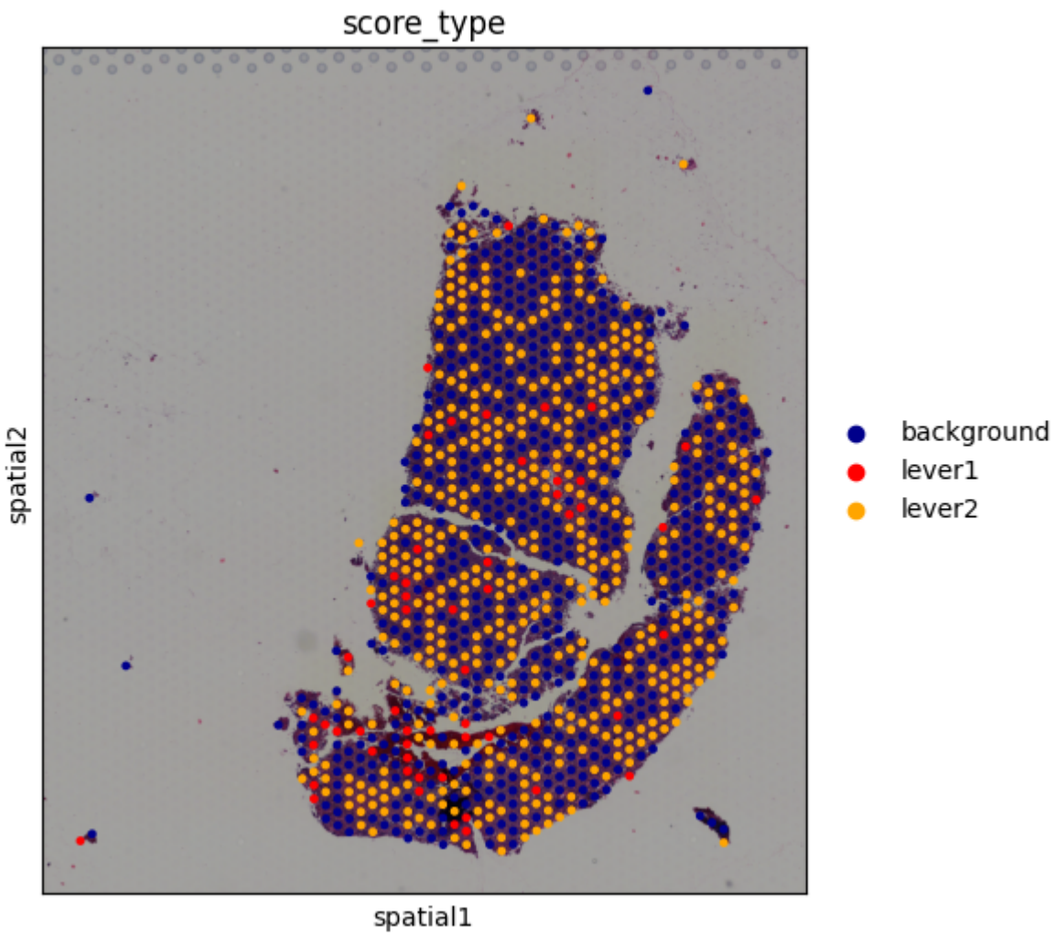
Out[10]:

	in_tissue	array_row	array_col	sample	n_genes_by_counts	log1p_n_genes_by_counts	total_counts	log1p_total_counts	pct_counts_in_top_5
AAACCGGGTAGGTACC-1	1	42	28	LD	7324	8.899048	25961.0	10.164390	1
AAACCGTTCGTCCAGG-1	1	52	42	LD	7850	8.968396	37458.0	10.531002	1
AAACCTCATGAAGTTG-1	1	37	19	LD	3961	8.284504	7936.0	8.979291	1
AAACGAGACGGTTGAT-1	1	35	79	LD	7348	8.902320	29413.0	10.289227	1
AAACTGCTGGCTCCAA-1	1	45	67	LD	7902	8.974998	35196.0	10.468717	1

5 rows × 31 columns

Use a color_map directory to plot dividing three parts into visium image.

In [11]:
```
color_map_dt = {
    'lever1': 'red',
    'lever2': 'orange',
    'background': 'darkblue',
}
ss.pl.spatial(adata, img_key='hires', color='score_type', palette=color_map_dt)
```



Split

In this step, we use the DFS algorithm starting from the red spot to expand outward, with the expansion condition being that the adjacent point is either a red spot or a yellow spot, until it cannot extend any further or reaches the edge of the image. After traversal, the red spots will be clustered into clusters. In a cluster, if there is only one cluster of red spots, it is called an 'independent' type; if there are more than two clusters of red spots in a cluster, the cluster will be further divided into smaller units based on the boundaries of each red spot cluster, known as 'adjacent' type. A new column 'class' will be created in the cluster_df.

In [12]:
```
cluster_df = ss.pp.find_lever_core(adata, 'hypoxia_score')
```

In [13]:
```
cluster_df
```

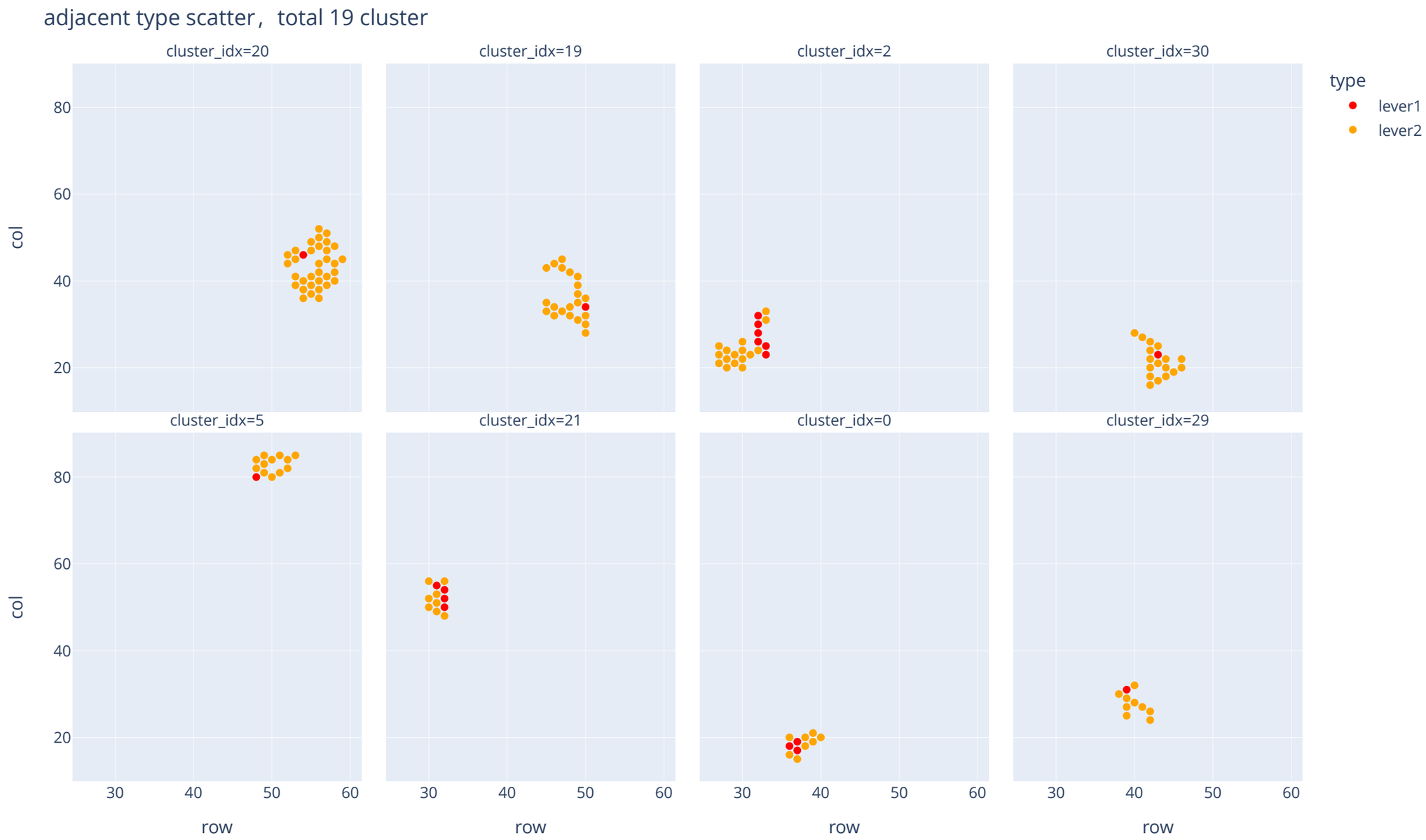
Out[13]:

	row	col	score	type	cluster_idx	class
0	37	19	0.857315	lever1	0	adjacent
1	37	17	1.219024	lever1	0	adjacent
2	37	15	0.697438	lever2	0	adjacent
3	36	16	0.661241	lever2	0	adjacent
4	36	18	0.774277	lever1	0	adjacent
...	...	...	...	...	...	...
272	59	65	0.535874	lever2	6	independent
273	61	65	0.540075	lever2	6	independent
274	60	60	0.602142	lever2	6	independent
275	61	59	0.533772	lever2	6	independent
276	29	29	0.781194	lever1	7	independent

277 rows × 6 columns

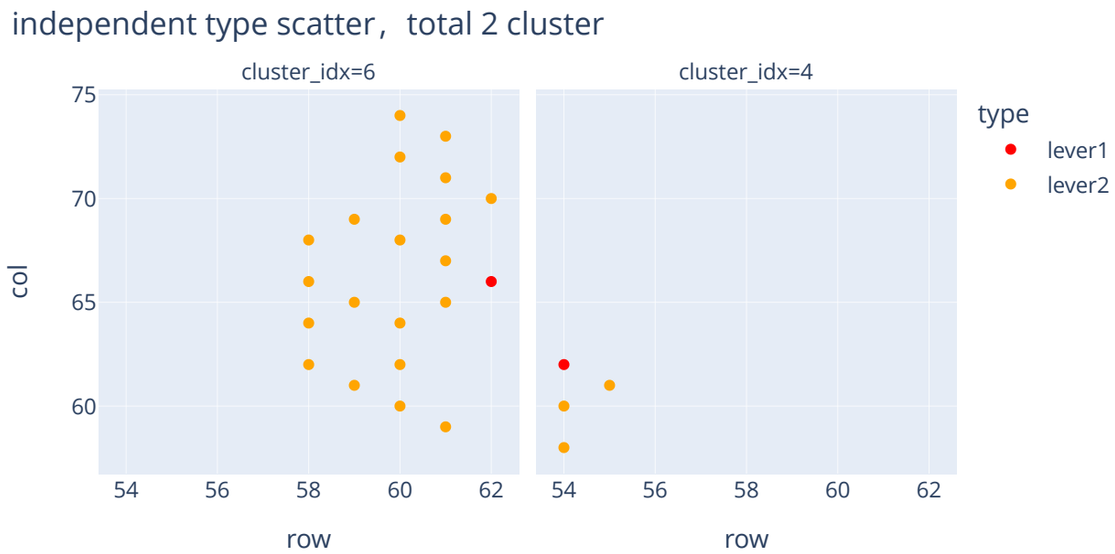
Plot 'adjacent' type spot cluster, 8 means show top8 clusters.
`ss.pl.plot_lever_core(cluster_df, 'adjacent', core_num=8, col_wrap=4, width=11.5, height=3.5)` : In order to display the points more clearly, you can use col_wrap(default: int=4) width(default: float=11.5) height(default: float=3.5) to adjust the layout.

In [14]:
```
ss.pl.plot_lever_core(cluster_df, 'adjacent', core_num=8)
```


adjacent type scatter， total 19 cluster
cluster_idx=20
cluster_idx=19
cluster_idx=2
cluster_idx=30
cluster_idx=5
cluster_idx=21
cluster_idx=0
cluster_idx=29
type
lever1
lever2
col
row
20
40
60
80
30
40
50
60

Plot 'independent' type spot cluster

In [15]:
```
ss.pl.plot_lever_core(cluster_df, 'independent', width=6.5)
```



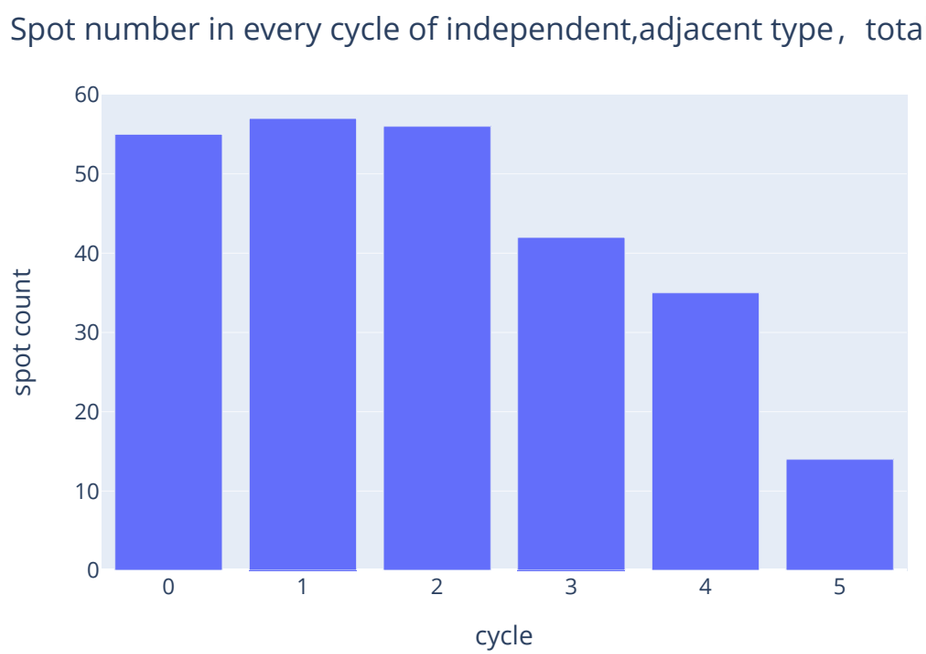
Align

Aligning with the red spots as reference points, merge all clustered units partitioned based on their distances to the red spots into one cluster.

In [16]:
```
ss.pp.cycle_spot_count(cluster_df)
```

Next, show spot number in every cycle through `ss.pl.plot_cycle_bar()`, cycles containing less than 10 spots were filtered.

In [17]:
```
ss.pl.plot_cycle_bar(cluster_df,ylabel='spot count')
```



Cycle 0 means core red cluster(lever1), it contains 55 spots.
Cycle 1 means yellow spot(lever2) that have a distance of 1 unit from cycle 0.
Cycle 2 means yellow spot(lever2) that have a distance of 2 units from cycle 0.
...
Cycle 5 means yellow spot(lever2) that have a distance of 5 units from cycle 0.

In [18]:
```
cluster_df
```

Out[18]:

	row	col	score	type	cluster_idx	class	cycle
0	37	19	0.857315	lever1	0	adjacent	0.0
1	37	17	1.219024	lever1	0	adjacent	0.0
2	37	15	0.697438	lever2	0	adjacent	1.0
3	36	16	0.661241	lever2	0	adjacent	1.0
4	36	18	0.774277	lever1	0	adjacent	0.0
...	...	...	...	...	...	...	...
272	59	65	0.535874	lever2	6	independent	3.0
273	61	65	0.540075	lever2	6	independent	1.0
274	60	60	0.602142	lever2	6	independent	3.0
275	61	59	0.533772	lever2	6	independent	4.0
276	29	29	0.781194	lever1	7	independent	0.0

277 rows × 7 columns

Map

Map the expression levels of specified cell types to the corresponding cycles.

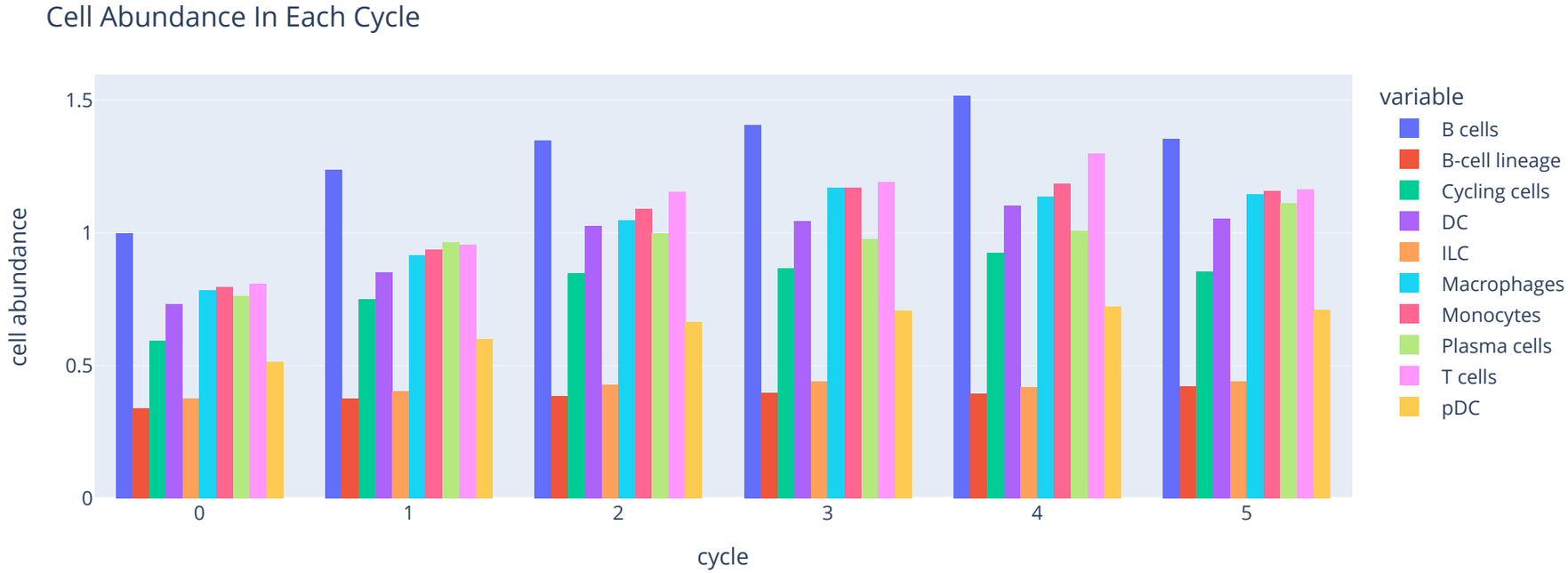
In [19]:
```
cell_type_lt = ['B cells', 'B-cell lineage', 'Cycling cells', 'DC', 'ILC', 'Macrophages', 'Monocytes', 'Plasma cells', 'T cells', 'pDC']
```

The `ss.pp.get_cell_abundance()` function requires three input parameters: adata, cluster, and cell_type_lt, where the cell_type_lt cell type list can be customized but must be included in the adata.obs object.

In [20]:
```
ss.pp.get_cell_abundance(adata, cluster_df, cell_type_lt)
```

Visualization of different cell abundance in different cycles.

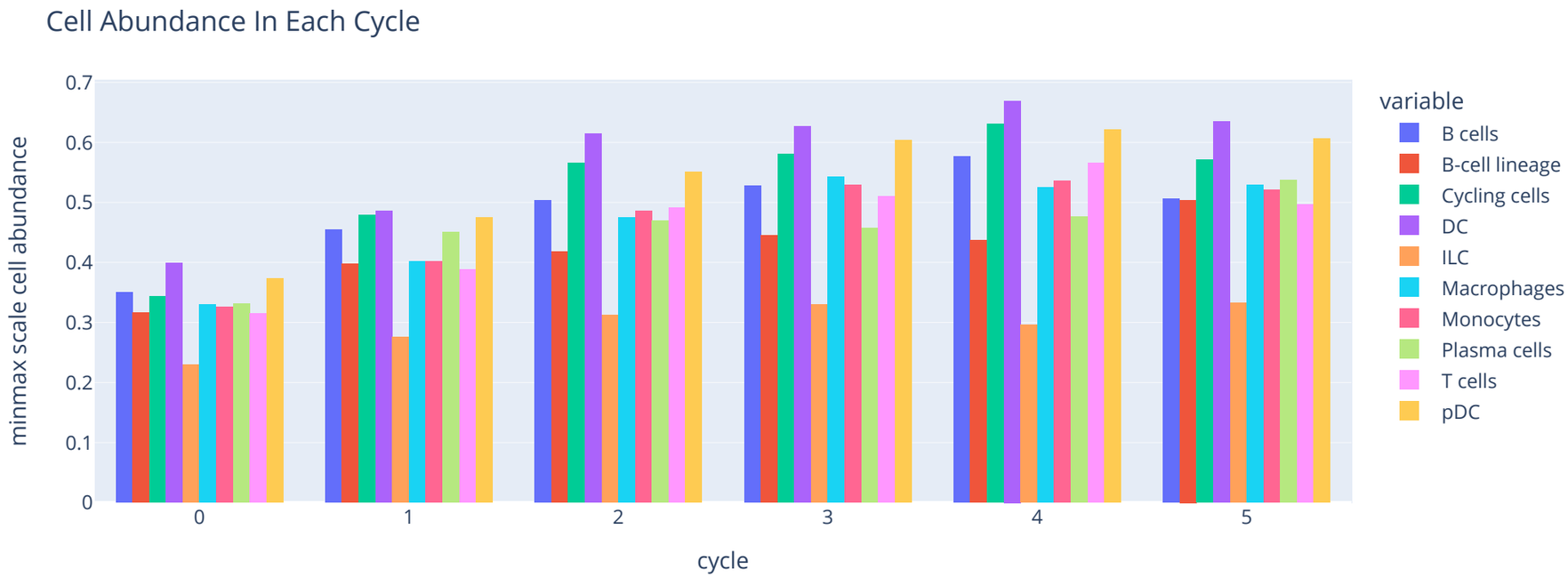
In [21]:
```
ss.pl.plot_cycle_abundance(cluster_df, cell_type_lt)
```



From the above bar graph, it can be observed that the expression levels of various immune cells increase with the number of cycles from 0 to 4, and slightly decrease at the 5th cycle. However, there are significant differences in expression levels among different cell types. For better comparison, the `ss.pp.minmax_scaler()` function can be used to map the expression levels of different cells to the range of 0-1.

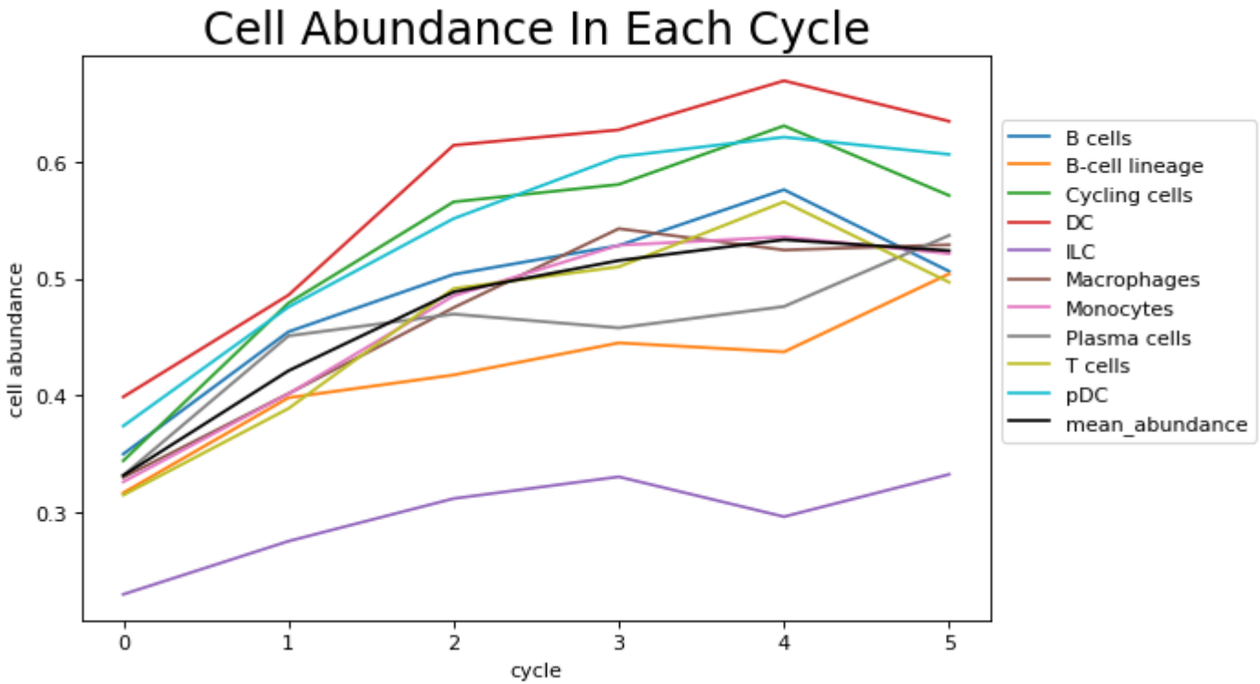
In [22]:
```
cluster_scale_df = ss.pp.minmax_scaler(cluster_df, cell_type_lt)
```

In [23]:
```
ss.pl.plot_cycle_abundance(cluster_scale_df, cell_type_lt,ylabel='minmax scale cell abundance')
```



To represent cell abundance using a line graph, the `ss.pl.plot_line_abundance()` function can be used.

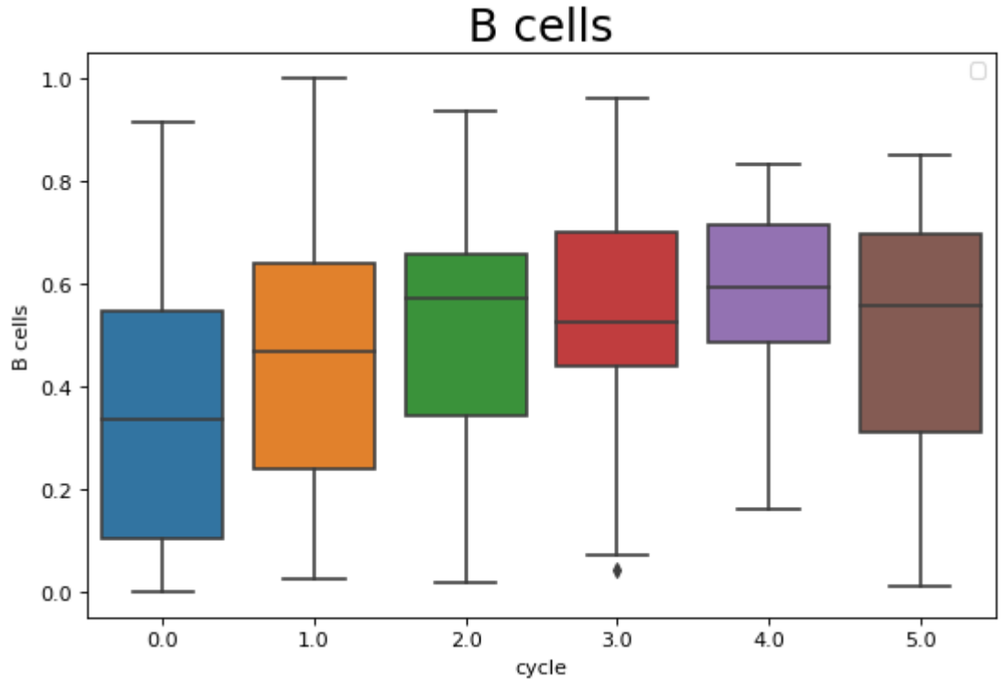
In [24]:
```
ss.pl.plot_line_abundance(cluster_scale_df, cell_type_lt)
```



To view the expression level of a specific cell type, the `ss.pl.plot_cell_abundance()` function can be used

In [25]:
```
ss.pl.plot_cell_abundance(cluster_scale_df, 'B cells')
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
```


B cells
1.0
0.8
0.6
0.4
0.2
0.0
B cells
0.0
1.0
2.0
3.0
4.0
5.0
cycle