https://doi.org/10.1038/s41596-023-00925-5

Protocol

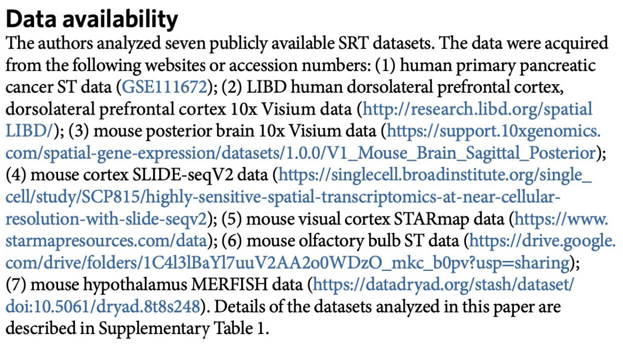
Check for updates

Streamlining spatial omics data analysis with Pysodb

In the format provided by the authors and unedited

Supplementary Figure 1

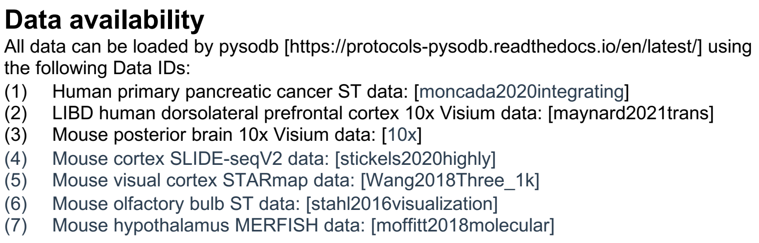
A



Data availability

The authors analyzed seven publicly available SRT datasets. The data were acquired from the following websites or accession numbers: (1) human primary pancreatic cancer ST data (GSE111672); (2) LIBD human dorsolateral prefrontal cortex, dorsolateral prefrontal cortex 10x Visium data (http://research.libd.org/spatialLIBD/); (3) mouse posterior brain 10x Visium data (https://support.10xgenomics.com/spatial-gene-expression/datasets/1.0.0/V1_Mouse_Brain_Sagittal_Posterior); (4) mouse cortex SLIDE-seqV2 data (https://singlecell.broadinstitute.org/single_cell/study/SCP815/highly-sensitive-spatial-transcriptomics-at-near-cellular-resolution-with-slide-seqv2); (5) mouse visual cortex STARmap data (https://www.starmapresources.com/data); (6) mouse olfactory bulb ST data (https://drive.google.com/drive/folders/1C4l3lBaYl7uuV2AA2o0WDzO_mkc_b0pv?usp=sharing); (7) mouse hypothalamus MERFISH data (https://datadryad.org/stash/dataset/doi:10.5061/dryad.8t8s248). Details of the datasets analyzed in this paper are described in Supplementary Table 1.

B



Data availability

All data can be loaded by pysodb [https://protocols-pysodb.readthedocs.io/en/latest/] using the following Data IDs:

(1) Human primary pancreatic cancer ST data: [moncada2020integrating]
(2) LIBD human dorsolateral prefrontal cortex 10x Visium data: [maynard2021trans]
(3) Mouse posterior brain 10x Visium data: [10x]
(4) Mouse cortex SLIDE-seqV2 data: [stickels2020highly]
(5) Mouse visual cortex STARmap data: [Wang2018Three_1k]
(6) Mouse olfactory bulb ST data: [stahl2016visualization]
(7) Mouse hypothalamus MERFISH data: [moffitt2018molecular]

Supplementary Figure 1. "Data availability" section of SpaGCN paper.
This figure uses SpaGCN paper as an example to show the advantage of pysodb in simplifying the data availability.
(A): In its original format, the data is linked to a variety of data storage platforms, requiring readers to navigate each platform for download. Additionally, because these data sets originate from different technologies, substantial processing time is required prior to applying the method itself.
(B): In contrast, with the application of pysodb, the authors only need to supply the dataset identifiers in SODB, enabling readers to retrieve the processed data in a more time-efficient manner with pysodb.

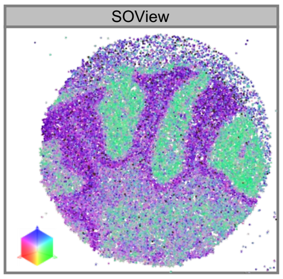
Supplementary Figure 2

A

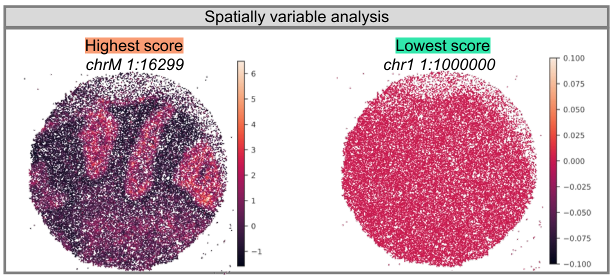
Spatial genomics data

Dataset Name: zhao2022spatial
Reference: Zhao et al. Nature
Technology: Slide-DNA-Seq
Species: Mouse
Tissue: Cerebellum

B



C

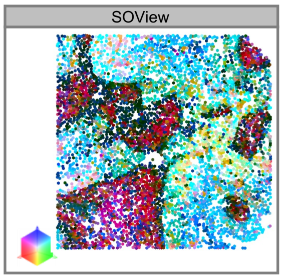


D

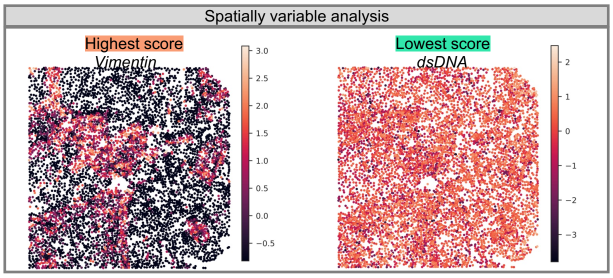
Spatial proteomics data

Dataset Name: keren2018a
Reference: Keren et al. Cell
Technology: MIBI-TOF
Species: Human
Tissue: Triple Negative Breast Cancer

E



F

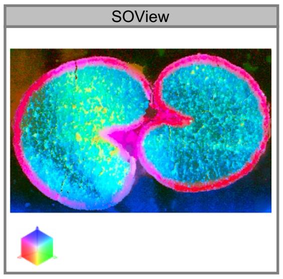


G

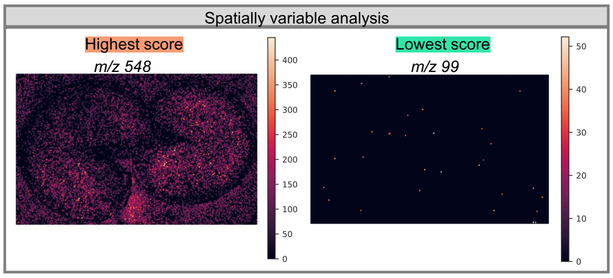
Spatial metabolomics data

Dataset Name: MALDI_seed
Reference: METASPACE
Technology: MALDI
Species: Wheat
Tissue: Seed

H



I

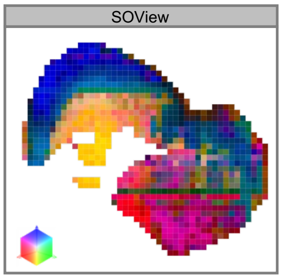


J

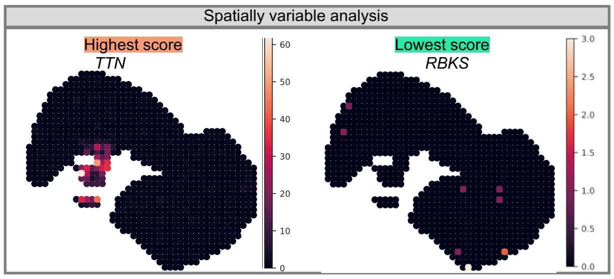
Spatial multi-omics data

Dataset Name: liu2020high
Reference: Liu et al. Cell
Technology: DBiT-Seq
Species: Mouse
Tissue: Embroy

K



L



Supplementary Figure 2. Applications of pysodb and spatially variable analysis on other spatial omics.
(A-C): Application on a spatial genomics data generated by Slide-DNA-Seq.
(D-F): Application on a spatial proteomics data generated by MIBI-TOF.
(G-I): Application on a spatial metabolomics data generated by MALDI.
(J-L): Application on a spatial multi-omics data generated by DBiT-Seq.
Taking (A-C) as an example, the data information is shown in (A). SOView visualization is shown in (B). Spatially variable analysis results (using Sepal) are shown in (C), left: top spatial variation, right: bottom spatial variation. Note that different spatial omics generated different features, e.g., loci for spatial genomics, protein for spatial proteomics, mass-to-charge ratio (m/z) for spatial metabolomics, and gene/protein for spatial multi-omics.

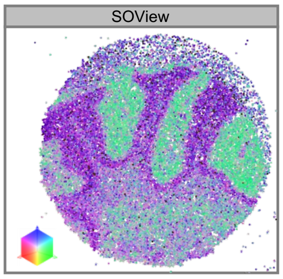
Supplementary Figure 3

A

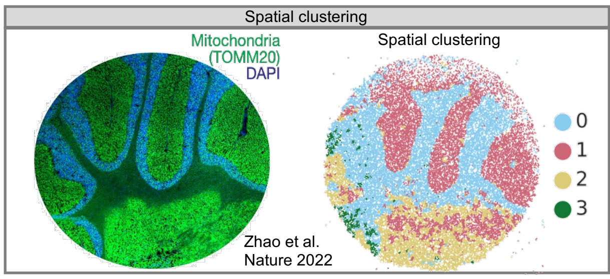
Spatial genomics data

Dataset Name: zhao2022spatial
Reference: Zhao et al. Nature
Technology: Slide-DNA-Seq
Species: Mouse
Tissue: Cerebellum

B



C

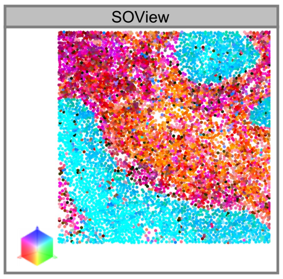


D

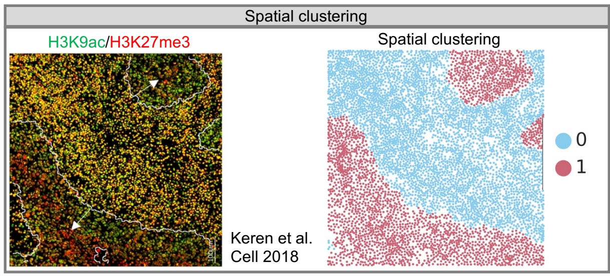
Spatial proteomics data

Dataset Name: keren2018a
Reference: Keren et al. Cell
Technology: MIBI-TOF
Species: Human
Tissue: Triple Negative Breast Cancer

E



F

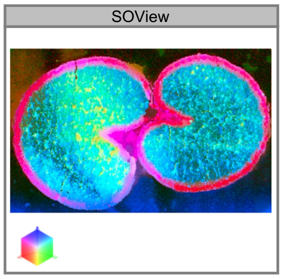


G

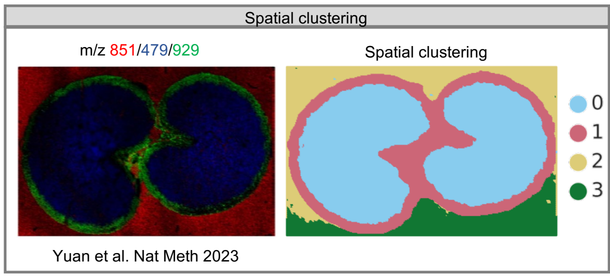
Spatial metabolomics data

Dataset Name: MALDI_seed
Reference: METASPACE
Technology: MALDI
Species: Wheat
Tissue: Seed

H



I



Supplementary Figure 3. Applications of pysodb and spatially clustering on other spatial omics.
(A-C): Application on a spatial genomics data generated by Slide-DNA-Seq.
(D-F): Application on a spatial proteomics data generated by MIBI-TOF.
(G-I): Application on a spatial metabolomics data generated by MALDI.
Taking (A-C) as an example, the data information is shown in (A). SOView visualization is shown in (B). Spatial clustering results (using SpaceFlow) and reference are shown in (C), left: specific markers (edited from Zhao et al. Nature 2022) are used for tissue structure reference, right: spatial clustering result.

Supplementary Figure 4

A

Spatial proteomics data

Dataset Name:
Keren2018a

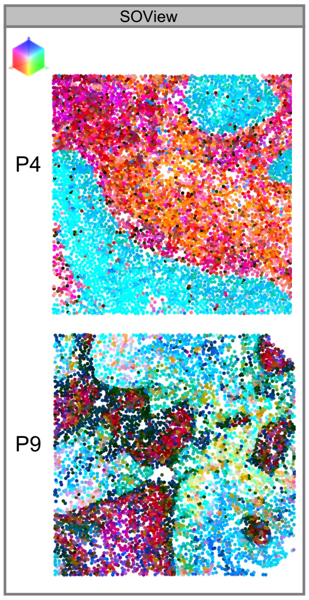
Reference:
Keren et al. Cell

Technology:
MIBI-TOF

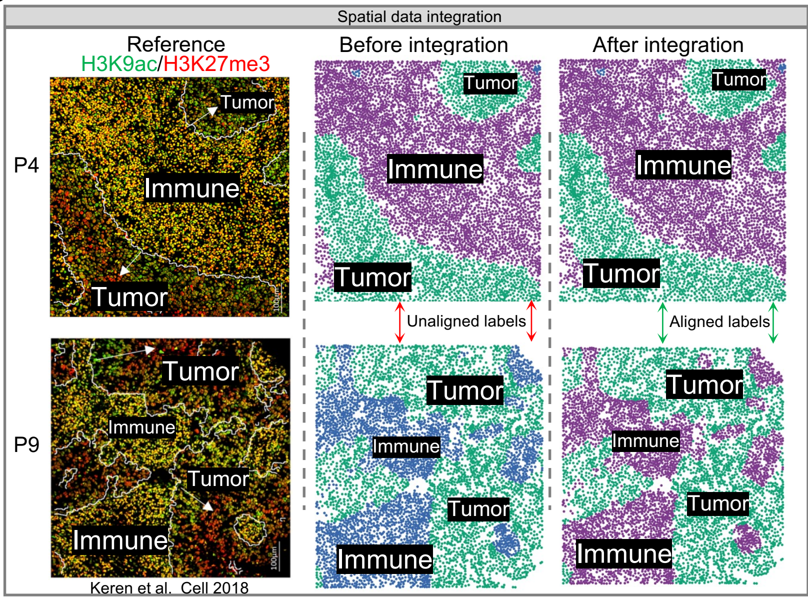
Species:
Human

Tissue:
Triple Negative Breast Cancer

B



C



D

Spatial proteomics data

Dataset Name:
Gut2018multiplexed

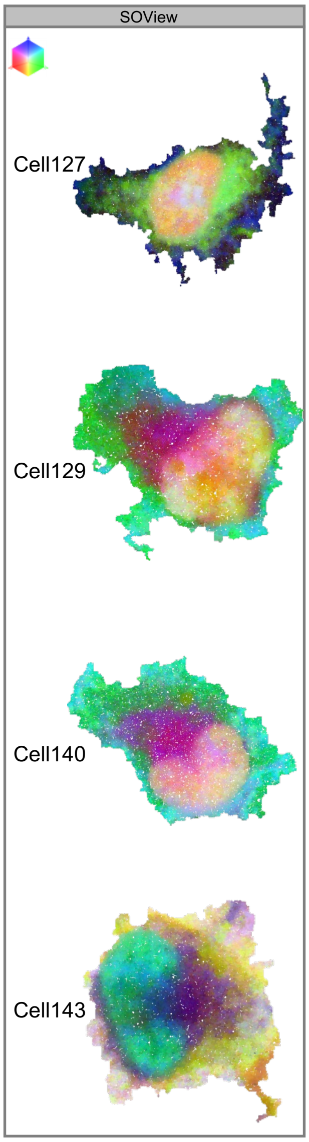
Reference:
Gut et al. Science

Technology:
Iterative Indirect Immunofluorescence Imaging

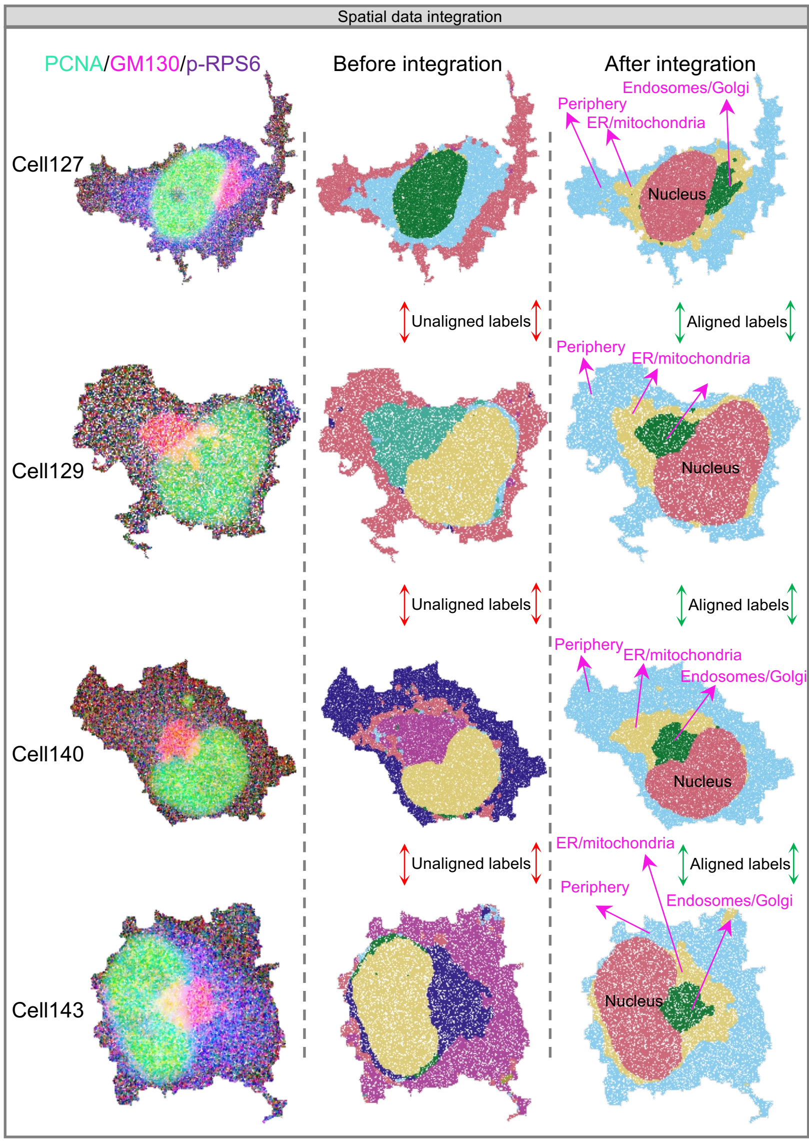
Species:
Cell line

Tissue:
N/A

E



F

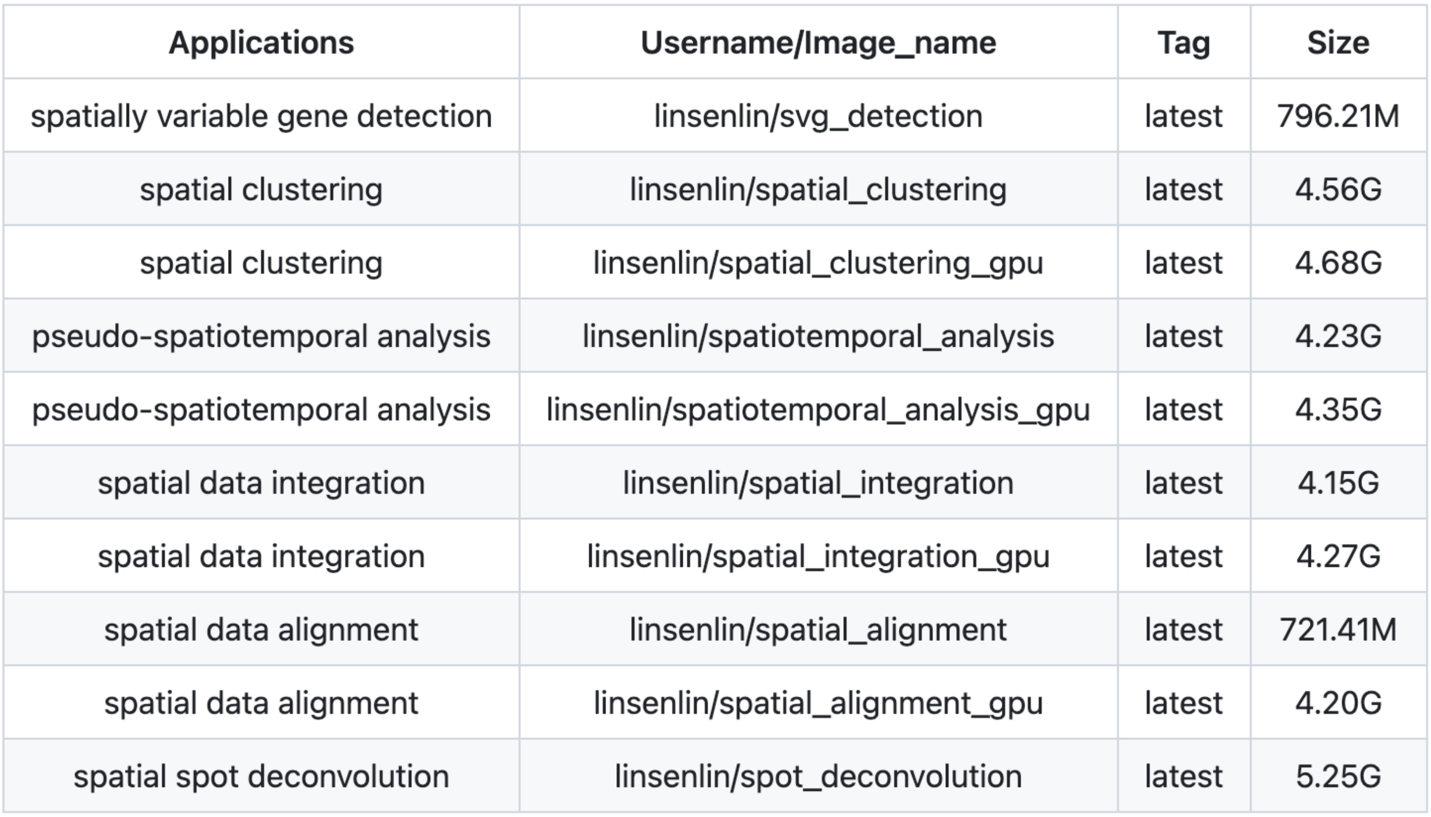


Supplementary Figure 4. Applications of pysodb and spatial data integration on other spatial omics.
(A-C): Application on a spatial proteomics data generated by MIBI-TOF.
(D-F): Application on a spatial proteomics data generated by Iterative Indirect Immunofluorescence Imaging (4i).
Taking (A-C) as an example, the data information is shown in (A). SOView visualization for data slices to be integrated are shown in (B), top: patient 4, bottom: patient 9. The results before and after integration, and the tissue structure references are shown in (C), left: specific markers (edited from Keren et al. Cell 2018) are used for tissue structure reference and white arrows indicate tumor region, middle: before integration, right: after integration.

Supplementary Table 1. Summary of parameters.

Please find the excel file in Supplementary Information.

Supplementary Table 2. Summary of Docker images.



Applications	Username/Image_name	Tag	Size
spatially variable gene detection	linsenlin/svg_detection	latest	796.21M
spatial clustering	linsenlin/spatial_clustering	latest	4.56G
spatial clustering	linsenlin/spatial_clustering_gpu	latest	4.68G
pseudo-spatiotemporal analysis	linsenlin/spatiotemporal_analysis	latest	4.23G
pseudo-spatiotemporal analysis	linsenlin/spatiotemporal_analysis_gpu	latest	4.35G
spatial data integration	linsenlin/spatial_integration	latest	4.15G
spatial data integration	linsenlin/spatial_integration_gpu	latest	4.27G
spatial data alignment	linsenlin/spatial_alignment	latest	721.41M
spatial data alignment	linsenlin/spatial_alignment_gpu	latest	4.20G
spatial spot deconvolution	linsenlin/spot_deconvolution	latest	5.25G

Supplementary Protocols

Case study 7: Generalizability to more spatial omics data

CRITICAL: Six case studies are previously supplied on using pysodb, including spatially variable gene detection, spatial clustering, pseudo-spatiotemporal analysis, spatial data integration, spatial data alignment, and spatial spot deconvolution. For this particular case study, we aim to provide more analysis to explore other spatial omics data using pysodb, such as spatial proteomics, metabolomics, genomics, and multi-omics.

Spatially variable analysis for Spatial genomics data (Timing ~ 8~9 h)

This section demonstrates the detection of spatially variable features using Pysodb and Sepal, applied to spatial genomics data obtained from a mouse cerebellum via Slide-DNA-Seq.

1. Import packages and set configurations, enter the following commands:

```
import numpy as np
# Import sepal package and its modules
import sepal.datasets as d
import sepal.models as m
import sepal.utils as ut
```

2. Import, initialize, and load data using Pysodb:

```
# Import pysodb package
import pysodb
# Initialization
sodb = pysodb.SODB()

# Define names of the dataset_name and experiment_name
dataset_name = 'zhao2022spatial'
experiment_name = 'mouse_cerebellum_1_dna_200114_14'
# Load a specific experiment
# It takes two arguments: the name of the dataset and the name of the experiment to load.
adata = sodb.load_experiment(dataset_name,experiment_name)
```

CRITICAL STEP: A *'load_experiment'* function is called with two *'dataset_name'* and *'experiment_name'* variables to load a specific experiment. Users can download data according to their needs by changing different *'dataset_name'* and *'experiment_name*.'

Then, an AnnData object can be saved to an H5AD file format with the filename 'mouse_cerebellum_1_dna_200114_14.h5ad' by entering the following command:

```
adata.write_h5ad('mouse_cerebellum_1_dna_200114_14.h5ad')
```

3. Perform Sepal to spatially variable gene detection for spatial genomics data

First, create an instance of the 'RawData' class with the input file 'mouse_cerebellum_1_dna_200114_14.h5ad,' and enter the following command:

```
raw_data = d.RawData('mouse_cerebellum_1_dna_200114_14.h5ad')
```

Next, create an instance of the *'UnstructuredData'* class, which is a subclass of the *'CountData'* class in Sepal, specifically designed to handle non-Visium or non-ST data, enter the following command:

```
data = m.UnstructuredData(raw_data, eps = 0.1)
```

CRITICAL STEP: A new 'UnstructuredData' object named 'data' using the following input arguments:(1) raw_data: A 'RawData' object containing the count matrix and other related data for the non-Visium or non-ST dataset; (2) eps: A float value representing the allowed difference in distance from the specified radius for finding approximate neighbors in the dataset. The default value is 0.1.

The *'propagate'* function simulates diffusion on spatial genomics data, applies normalization and scaling, and saves the scaled diffusion times for each profile in a data frame named *'times.'* These diffusion times represent the rate at which gene expression levels disperse spatially during the diffusion process:

```
times = m.propagate(data, normalize = True, scale =True)
```

Next, select the top 10 and bottom 10 profiles based on their diffusion times, and enter the following commands:

```
# Selects the top 10 and bottom 10 profiles based on their diffusion times
# Set the number of top and bottom profiles to be selected as 10
n_top = 10
# Computes the indices that would sort the times DataFrame in ascending order
sorted_indices = np.argsort(times.values.flatten())
# Reverses the order of the sorted indices to obtain a descending order
sorted_indices = sorted_indices[::-1]
# Retrieves the profile names corresponding to the sorted indices
sorted_profiles = times.index.values[sorted_indices]
# Select the top 10 profile names with the highest diffusion times
top_profiles = sorted_profiles[0:n_top]
# Selects the bottom 10 profile names with the lowest diffusion times
tail_profiles = sorted_profiles[-n_top:]
# Retrieves the top 10 profiles from the times DataFrame
times.loc[top_profiles,:]
```

Define a *'pltargs'* dictionary and then invoke the *'plot_profiles'* function twice by changing *'tail_profiles'* and *'top_profiles'* parameters to create separate visualizations of the top and low spatial genomics profiles. Execute the following commands in turn:

```
# Inspect detecition visually by using the "plot_profiles function for first 10 SVG
# Define a custom pltargs dictionary with plot style options
pltargs = dict(s = 5, cmap = "magma", edgecolor = 'none', marker = 'H',)
# plot the profiles
fig,ax = ut.plot_profiles(cnt = data.cnt.loc[:,top_profiles], crd = data.real_crd, rank_values = times.loc[top_profiles,:].values.flatten(), pltargs = pltargs,)
# See Supplementary Fig. 2c (left)

# Inspect detecition visually by using the "plot_profiles function for last 10 SVG
# Define a custom pltargs dictionary with plot style options
pltargs = dict(s = 5, cmap = "magma", edgecolor = 'none', marker = 'H', )
# plot the profiles
fig,ax = ut.plot_profiles(cnt = data.cnt.loc[:,tail_profiles], crd = data.real_crd, rank_values = times.loc[tail_profiles,:].values.flatten(), pltargs = pltargs,)
# See Supplementary Fig. 2c (right)
```

Spatially variable analysis for Spatial proteomics data (Timing ~ 1 min)

This section illustrates the application of Pysodb and Sepal in detecting spatially variable features, utilizing unprecedented spatial proteomics data derived from human triple-negative breast cancer through MIBI-TOF.

4. Import packages and set configurations, enter the following commands:

```
import numpy as np
# Import sepal package and its modules
import sepal.datasets as d
import sepal.models as m
import sepal.utils as ut
```

5. Import, initialize, and load data using Pysodb:

```
# Import pysodb package
import pysodb
# Initialization
sodb = pysodb.SODB()

# Define names of the dataset_name and experiment_name
```

```
dataset_name = 'keren2018a'
experiment_name = 'p9'
# Load a specific experiment
# It takes two arguments: the name of the dataset and the name of the experiment to load.
adata = sodb.load_experiment(dataset_name,experiment_name)
```

Then, save an H5AD file format with the filename *'*keren2018a_p9.h5ad*'* by entering the following command:

```
adata.write_h5ad('keren2018a_p9.h5ad')
```

6. Perform Sepal to spatially variable gene detection for spatial proteomics data

First, create an instance of the *'RawData'* class with the input file *'*keren2018a_p9.h5ad*'*, and enter the following command:

```
raw_data = d.RawData('keren2018a_p9.h5ad')
```

Next, create an instance of the 'UnstructuredData' class. This class is a specialized subclass of the 'CountData' class in Sepal, designed specifically for processing data that is neither Visium nor ST type, enter the following command:

```
data = m.UnstructuredData(raw_data, eps = 0.1)
```

The 'propagate' function models diffusion on spatial proteomics data normalizes and scales it, and stores the scaled diffusion times in a data frame called 'times':

```
times = m.propagate(data, normalize = True, scale =True)
```

Afterward, choose the profiles with the highest and lowest diffusion times, consisting of the top 10 and bottom 10 profiles, respectively, and execute the following commands:

```
# Selects the top 10 and bottom 10 profiles based on their diffusion times
# Set the number of top and bottom profiles to be selected as 10
n_top = 10
# Computes the indices that would sort the times DataFrame in ascending order
sorted_indices = np.argsort(times.values.flatten())
# Reverses the order of the sorted indices to obtain a descending order
sorted_indices = sorted_indices[::-1]
# Retrieves the profile names corresponding to the sorted indices
sorted_profiles = times.index.values[sorted_indices]
# Select the top 10 profile names with the highest diffusion times
top_profiles = sorted_profiles[0:n_top]
# Selects the bottom 10 profile names with the lowest diffusion times
tail_profiles = sorted_profiles[-n_top:]
```

```
# Retrieves the top 10 profiles from the times DataFrame
times.loc[top_profiles,:]
```

Similarly, create a dictionary called 'pltargs' and use it to call the 'plot_profiles' function twice, with different parameters for 'tail_profiles' and 'top_profiles,' in order to generate two separate visualizations of the top and bottom spatial proteomics profiles. Execute the following commands sequentially:

```
# Inspect detecition visually by using the "plot_profiles function for first 10 SVG
# Define a custom pltargs dictionary with plot style options
pltargs = dict(s = 15, cmap = "magma", edgecolor = 'none', marker = 'H', )
# plot the profiles
fig,ax = ut.plot_profiles(cnt = data.cnt.loc[:,top_profiles], crd = data.real_crd, rank_values = times.loc[top_profiles,:].values.flatten(), pltargs = pltargs, )
# See Supplementary Fig. 2f (left)

# Inspect detecition visually by using the "plot_profiles function for last 10 SVG
# Define a custom pltargs dictionary with plot style options
pltargs = dict(s = 15, cmap = "magma", edgecolor = 'none', marker = 'H', )
# plot the profiles
fig,ax = ut.plot_profiles(cnt = data.cnt.loc[:,tail_profiles], crd = data.real_crd, rank_values = times.loc[tail_profiles,:].values.flatten(), pltargs = pltargs, )
# See Supplementary Fig. 2f (right)
```

Spatially variable analysis for Spatial metabolomics data (Timing ~ 20 min)

This section exemplifies the use of Pysodb and Sepal for identifying spatially variable features, leveraging new spatial metabolomics data sourced from wheat seed via MALDI.

7. Import packages and set configurations, enter the following commands:

```
import numpy as np
# Import sepal package and its modules
import sepal.datasets as d
import sepal.models as m
import sepal.utils as ut
```

8. Import, initialize, and load data using Pysodb:

```
# Import pysodb package
import pysodb
```

```
# Initialization
sodb = pysodb.SODB()

# Define names of the dataset_name and experiment_name
dataset_name = 'MALDI_seed'
experiment_name = 'S655_WS22_320x200_15um_E110'
# Load a specific experiment
# It takes two arguments: the name of the dataset and the name of the experiment to load.
adata = sodb.load_experiment(dataset_name,experiment_name)
```

Subsequently, store an AnnData object in the H5AD file format under the filename 'S655_WS22_320x200_15um_E110.h5ad' by executing the following command:

```
adata.write_h5ad('S655_WS22_320x200_15um_E110.h5ad')
```

9. Perform Sepal to spatially variable gene detection for spatial metabolomics data

First, create an instance of the 'RawData' class with the input file 'S655_WS22_320x200_15um_E110.h5ad,' enter the following command:

```
raw_data = d.RawData('S655_WS22_320x200_15um_E110.h5ad')
```

Next, create an instance of the 'UnstructuredData' class, which is a subclass of the 'CountData' class in Sepal, specifically designed to handle non-Visium or non-ST data by entering the following command:

```
data = m.UnstructuredData(raw_data, eps = 0.1)
```

The 'propagate' function simulates diffusion on spatial metabolomics data, conducts normalization and scaling procedures, and subsequently records the scaled diffusion times for each profile into a data frame called 'times.' These diffusion times symbolize the rate at which gene expression levels spread spatially throughout the diffusion process.

```
times = m.propagate(data, normalize = True, scale =True)
```

Next, select the top 10 and bottom 10 profiles based on their diffusion times, enter the following commands:

```
# Selects the top 10 and bottom 10 profiles based on their diffusion times
# Set the number of top and bottom profiles to be selected as 10
n_top = 10
# Computes the indices that would sort the times DataFrame in ascending order
sorted_indices = np.argsort(times.values.flatten())
# Reverses the order of the sorted indices to obtain a descending order
sorted_indices = sorted_indices[::-1]
```

```
# Retrieves the profile names corresponding to the sorted indices
sorted_profiles = times.index.values[sorted_indices]
# Select the top 10 profile names with the highest diffusion times
top_profiles = sorted_profiles[0:n_top]
# Selects the bottom 10 profile names with the lowest diffusion times
tail_profiles = sorted_profiles[-n_top:]
# Retrieves the top 10 profiles from the times DataFrame
times.loc[top_profiles,:]
```

Generate two separate visualizations of the top and bottom spatial metabolomics profiles. Execute the following commands sequentially:

```
# Inspect detecition visually by using the "plot_profiles function for first 10 SVG
# Define a custom pltargs dictionary with plot style options
pltargs = dict(s = 3, cmap = "magma", edgecolor = 'none', marker = 'H', )
# plot the profiles
fig,ax = ut.plot_profiles(cnt = data.cnt.loc[:,top_profiles], crd = data.real_crd, rank_values = times.loc[top_profiles,:].values.flatten(), pltargs = pltargs, )
# See Supplementary Fig. 2i (left)

# Inspect detecition visually by using the "plot_profiles function for last 10 SVG
# Define a custom pltargs dictionary with plot style options
pltargs = dict(s = 3, cmap = "magma", edgecolor = 'none', marker = 'H', )
# plot the profiles
fig,ax = ut.plot_profiles(cnt = data.cnt.loc[:,tail_profiles], crd = data.real_crd, rank_values = times.loc[tail_profiles,:].values.flatten(), pltargs = pltargs, )
# See Supplementary Fig. 2i (right)
```

Spatially variable analysis for Spatial multiomics Data (Timing ~ 1 min)

This section showcases the utilization of Pysodb and Sepal to discern spatially variable features, drawing upon spatial multi-omics data derived from a mouse embryo through DBiT-Seq.

10. Import packages and set configurations, enter the following commands:

```
import numpy as np
# Import sepal package and its modules
import sepal.datasets as d
import sepal.models as m
import sepal.utils as ut
```

11. Import, initialize, and load data using Pysodb:

```
# Import pysodb package
import pysodb
# Initialization
sodb = pysodb.SODB()

# Define names of the dataset_name and experiment_name
dataset_name = 'liu2020high'
experiment_name = 'E10_whole_gene_best'
# Load a specific experiment
# It takes two arguments: the name of the dataset and the name of the experiment to load.
adata = sodb.load_experiment(dataset_name,experiment_name)
```

Next, save an H5AD file format with the filename 'E10_whole_gene_best.h5ad' by entering the following command:

```
adata.write_h5ad('E10_whole_gene_best.h5ad')
```

12. Perform Sepal to spatially variable gene detection for spatial multiomics data
Firstly, create an instance of the *'RawData'* class with the input file 'E10_whole_gene_best.h5ad', and enter the following command:

```
raw_data = d.RawData('E10_whole_gene_best.h5ad')
```

To enhance the quality of the count matrix in the *'raw_data'* object, utilize the *'filter_genes'* function to eliminate genes that are irrelevant or of low quality selectively, enter the following command:

```
raw_data.cnt = ut.filter_genes(raw_data.cnt, min_expr=10, min_occur=5)
```

Next, initiate an instance of the 'UnstructuredData' class. This class, a subclass of the 'CountData' class in Sepal, is specifically tailored to process data that does not conform to Visium or ST type. Proceed by entering the following command:

```
data = m.UnstructuredData(raw_data, eps = 0.1)
```

The 'propagate' function simulates the process of diffusion on spatial multiomics data, executes normalization and scaling, and subsequently saves the scaled diffusion times in a data frame referred to as 'times':

```
times = m.propagate(data, normalize = True, scale =True)
```

Afterward, choose the profiles with the highest and lowest diffusion times, consisting of the top 10 and bottom 10 profiles, respectively, and run the following commands:

```
# Selects the top 10 and bottom 20 profiles based on their diffusion times
# Set the number of top and bottom profiles to be selected as 10
n_top = 10
# Computes the indices that would sort the times DataFrame in ascending order
sorted_indices = np.argsort(times.values.flatten())
# Reverses the order of the sorted indices to obtain a descending order
sorted_indices = sorted_indices[::-1]
# Retrieves the profile names corresponding to the sorted indices
sorted_profiles = times.index.values[sorted_indices]
# Select the top 10 profile names with the highest diffusion times
top_profiles = sorted_profiles[0:n_top]
# Selects the bottom 10 profile names with the lowest diffusion times
tail_profiles = sorted_profiles[-n_top:]
# Retrieves the top 10 profiles from the times DataFrame
times.loc[top_profiles,:]
```

Create separate visualizations of the top and low spatial multiomics profiles. Execute the following commands in turn:

```
# Inspect detecition visually by using the "plot_profiles function for first 10 SVG
# Define a custom pltargs dictionary with plot style options
pltargs = dict(s = 25, cmap = "magma", edgecolor = 'none', marker = 'H', )
# plot the profiles
fig,ax = ut.plot_profiles(cnt = data.cnt.loc[:,top_profiles], crd = data.real_crd, rank_values = times.loc[top_profiles,:].values.flatten(), pltargs = pltargs, )
# See Supplementary Fig. 2l (left)
```

```
# Inspect detecition visually by using the "plot_profiles function for last 10 SVG
# Define a custom pltargs dictionary with plot style options
pltargs = dict(s = 25, cmap = "magma", edgecolor = 'none', marker = 'H', )
# plot the profiles
fig,ax = ut.plot_profiles(cnt = data.cnt.loc[:,tail_profiles], crd = data.real_crd, rank_values = times.loc[tail_profiles,:].values.flatten(), pltargs = pltargs, )
# See Supplementary Fig. 2l (right)
```

Spatial clustering for Spatial genomics data (Timing ~ 1~2 min)

This section highlights the application of Pysodb and SpaceFlow in identifying spatial clustering, employing novel spatial genomics data procured from a mouse cerebellum through Slide-DNA-Seq.

13. Import packages and set configurations, enter the following commands:

```
import scanpy as sc
# from SpaceFlow package import SpaceFlow module
from SpaceFlow import SpaceFlow

import palettable
cmp_pspace = palettable.cartocolors.diverging.TealRose_7.mpl_colormap
cmp_domain = palettable.cartocolors.qualitative.Pastel_10.mpl_colors
cmp_ct = palettable.cartocolors.qualitative.Safe_10.mpl_colors
```

CRITICAL: The Palettable package can be imported to access a wide range of color palettes and colormap generators suitable for data visualization.

14. Import, initialize, and load data using Pysodb:

```
# Import pysodb package
import pysodb
# Initialize the sodb object
sodb = pysodb.SODB()

# Define names of the dataset_name and experiment_name
dataset_name = 'zhao2022spatial'
experiment_name = 'mouse_cerebellum_1_dna_200114_14'
# Load a specific experiment
# It takes two arguments: the name of the dataset and the name of the experiment to load.
adata = sodb.load_experiment(dataset_name,experiment_name)
```

15. Perform SpaceFlow to spatial clustering for spatial genomics data
To analyze spatial genomics data, a *'SpaceFlow'* object must first be created. This object takes in expression count data, spatial location coordinates, sample names, and feature names from an AnnData object called *'adata'* as input. To create the *'SpaceFlow'* object, enter the following commands:

```
sf = SpaceFlow.SpaceFlow(
    count_matrix=adata.X,
    spatial_locs=adata.obsm['spatial'],
    sample_names=adata.obs_names,
    gene_names=adata.var_names
)
```

After creating the 'SpaceFlow' object, the next step involves preprocessing the spatial genomics data, and enter the following command:

```
sf.preprocessing_data()
```

CRITICAL: When dealing with anndata (adata) where the count or expression matrix is extremely sparse, or where there are a very limited number of features, it may be preferable to forego data preprocessing. This is because over-processing in these instances could lead to errors or diminished performance in downstream tasks. To skip preprocessing, user will need to make modifications to the preprocessing_data function within the "SpaceFlow.py" file of the SpaceFlow package. Specifically, user should comment out the sc.pp.normalize_total(), sc.pp.log1p(), and sc.pp.highly_variable_genes() functions.

Following the above preprocessing step, train a SpaceFlow model and return the embedding by entering the following command:

```
embedding = sf.train(
    spatial_regularization_strength=0.1,
    z_dim=50,
    lr=1e-3,
    epochs=1000,
    max_patience=50,
    min_stop=100,
    random_seed=42,
    gpu=0,
    regularization_acceleration=True,
    edge_subset_sz=1000000
)
```

Then, save the embeddings of the trained SpaceFlow model to adata.obsm['SpaceFlow'], enter the following command:

```
adata.obsm['SpaceFlow'] = embedding
```

Subsequently, compute neighborhood graph utilizing the 'SpaceFlow' representation, conduct dimensionality reduction using the UMAP algorithm, and apply clustering to the representation employing the Leiden algorithm, enter the following commands:

```
sc.pp.neighbors(adata, use_rep= 'SpaceFlow')
sc.tl.umap(adata)
sc.tl.leiden(adata, resolution= 0.3)
```

Next, generate a plot of the UMAP embedding colored by *'leiden,'* enter the following command:

```
sc.pl.umap(adata, color= 'leiden', color_map= cmp_pspace)
# See Supplementary Fig. 3c (left)
```

Last, display a spatial embedding plot with clustering information*,* enter the following commands:

```
ax = sc.pl.embedding(adata, basis= 'spatial', color= 'leiden', show=False, color_map=cmp_pspace)
ax.axis('equal')
# See Supplementary Fig. 3c (right)
```

Spatial clustering for Spatial proteomics data (Timing ~ 0.5 min)

This section presents the application of Pysodb and SpaceFlow in identifying spatial clustering, employing novel spatial proteomics data procured from a human triple negative breast cancer through MIBI-TOF.

16. Import packages and set configurations, enter the following commands:

```
import scanpy as sc
# from SpaceFlow package import SpaceFlow module
from SpaceFlow import SpaceFlow

import palettable
cmp_pspace = palettable.cartocolors.diverging.TealRose_7.mpl_colormap
cmp_domain = palettable.cartocolors.qualitative.Pastel_10.mpl_colors
cmp_ct = palettable.cartocolors.qualitative.Safe_10.mpl_colors
```

17. Import, initialize, and load data using Pysodb:

```
# Import pysodb package
import pysodb
# Initialize the sodb object
sodb = pysodb.SODB()

# Define names of the dataset_name and experiment_name
dataset_name = 'keren2018a'
experiment_name = 'p4'
# Load a specific experiment
# It takes two arguments: the name of the dataset and the name of the experiment to load.
adata = sodb.load_experiment(dataset_name,experiment_name)
```

18. Perform SpaceFlow to spatial clustering for spatial proteomics data
To generate a *'SpaceFlow'* object for spatial clustering, input expression count data, spatial location coordinates, sample names, and feature names from an AnnData object called *'adata'* by executing the subsequent command:

```
sf = SpaceFlow.SpaceFlow(
    count_matrix=adata.X,
    spatial_locs=adata.obsm['spatial'],
    sample_names=adata.obs_names,
    gene_names=adata.var_names
)
```

Upon creating the '*SpaceFlow*' object, the next step entails preprocessing the spatial proteomics data. The following command should be entered:

```
sf.preprocessing_data()
```

CRITICAL: In situations where anndata (adata) is characterized by an extremely sparse count or expression matrix, or when dealing with limited-feature data such as spatial proteomics, it might be more beneficial to omit data preprocessing. Over-processing in these cases could potentially result in errors or impede performance in subsequent tasks. In order to bypass preprocessing, modifications must be made to the preprocessing_data function located in the "SpaceFlow.py" file, which is a part of the SpaceFlow package. Specifically, users should deactivate the sc.pp.normalize_total(), sc.pp.log1p(), and sc.pp.highly_variable_genes() functions by commenting them out.
After the preprocessing step has been completed, the user can proceed to train a SpaceFlow model by executing the following command:

```
embedding = sf.train(
    spatial_regularization_strength=0.1,
    z_dim=50,
    lr=1e-3,
    epochs=1000,
    max_patience=50,
    min_stop=100,
    random_seed=42,
    gpu=0,
    regularization_acceleration=True,
    edge_subset_sz=1000000
)
```

Next, store the embeddings of the trained SpaceFlow model in the adata.obsm['SpaceFlow']:

```
adata.obsm['SpaceFlow'] = embedding
```

Next, calculate the neighborhood graph of cells using 'SpaceFlow' representation, perform UMAP dimensionality reduction, and cluster the representation using the leiden algorithm, enter the following commands:

```
sc.pp.neighbors(adata, use_rep= 'SpaceFlow')
sc.tl.umap(adata)
sc.tl.leiden(adata, resolution=0.05)
```

Next, visualize a UMAP embedding, enter the following commands:

```
sc.pl.umap(adata, color= 'leiden', color_map= cmp_pspace)
# See Supplementary Fig. 3f (left)
```

Last, visualize a spatial embedding with clustering information, enter the following commands:

```
ax = sc.pl.embedding(adata, basis= 'spatial', color='leiden', show= False, color_map=cmp_pspace)
ax.axis('equal')
# See Supplementary Fig. 3f (right)
```

Spatial clustering for Spatial metabolomics data (Timing ~ 2 min)

This section demonstrates the use of Pysodb and SpaceFlow to detect spatial clustering, using innovative spatial metabolomics data derived from a wheat seed via MALDI.

19. Import packages and set configurations, enter the following commands:

```
import scanpy as sc
# from SpaceFlow package import SpaceFlow module
from SpaceFlow import SpaceFlow

import palettable
cmp_pspace = palettable.cartocolors.diverging.TealRose_7.mpl_colormap
cmp_domain = palettable.cartocolors.qualitative.Pastel_10.mpl_colors
cmp_ct = palettable.cartocolors.qualitative.Safe_10.mpl_colors
```

20. Import, initialize, and load data using Pysodb:

```
# Import pysodb package
import pysodb
# Initialize the sodb object
sodb = pysodb.SODB()

# Define names of the dataset_name and experiment_name
dataset_name = 'MALDI_seed'
experiment_name = 'S655_WS22_320x200_15um_E110'
# Load a specific experiment
# It takes two arguments: the name of the dataset and the name of the experiment to load.
adata = sodb.load_experiment(dataset_name,experiment_name)
```

21. Perform SpaceFlow to spatial clustering for spatial metabolomics data

In order to construct a 'SpaceFlow' object for the analysis of spatial metabolomics data, users can supply expression count data, spatial location coordinates, sample names, and feature names from an AnnData object, execute the following command:

```
sf = SpaceFlow.SpaceFlow(
        count_matrix=adata.X,
        spatial_locs=adata.obsm['spatial'],
        sample_names=adata.obs_names,
        gene_names=adata.var_names
)
```

After the SpaceFlow object is successfully created, the next step involves preprocessing the spatial metabolomics data, execute the following command:

```
sf.preprocessing_data()
```

Upon the completion of the preprocessing stage, move forward to train the SpaceFlow model and return the embedding by executing the following command:

```
embedding = sf.train(
        spatial_regularization_strength=0.1,
        z_dim=50,
        lr=1e-3,
        epochs=1000,
        max_patience=50,
        min_stop=100,
        random_seed=42,
        gpu=0,
        regularization_acceleration=True,
        edge_subset_sz=1000000
)
```

Next, store the embeddings generated by the trained SpaceFlow model into 'adata.obsm['SpaceFlow']':

```
adata.obsm['SpaceFlow'] = embedding
```

The following steps include computing the neighborhood graph of cells employing the 'SpaceFlow' representation, performing UMAP dimensionality reduction, and clustering the representation with the 'leiden' algorithm, run the following commands:

```
sc.pp.neighbors(adata, use_rep= 'SpaceFlow')
sc.tl.umap(adata)
sc.tl.leiden(adata, resolution=0.04)
```

Next, generate a plot of the UMAP embedding colored by 'leiden', execute the following command:

```
sc.pl.umap(adata, color= 'leiden', color_map= cmp_pspace)
# See Supplementary Fig. 3i (left)
```

Finally, a spatial embedding can be visualized with 'leiden' color-coding and equal axis scaling, enter the following commands:

```
ax = sc.pl.embedding(adata, basis= 'spatial', color='leiden', show= False, color_map=cmp_pspace)
ax.axis('equal')
# See Supplementary Fig. 3i (right)
```

Spatial data integration for Spatial proteomics data (Timing ~ 1~2 min)

This section illustrates the application of Pysodb, STAGATE, and Harmony for the integration of spatial proteomics data obtained from a human triple-negative breast cancer sample using MIBI-TOF.

22. Import packages and set configurations, enter the following commands:

```
import pandas as pd
import scanpy as sc
import matplotlib.pyplot as plt
import STAGATE_pyG as STAGATE
import harmonypy as hm

import palettable
cmp_old = palettable.cartocolors.qualitative.Bold_10.mpl_colors
cmp_old_biotech = palettable.cartocolors.qualitative.Safe_4.mpl_colors
```

23. Import, initialize, and load data using Pysodb:

Firstly, a Pysodb package is imported and initialized by creating a 'SODB' object and enter the following commands:

```
import pysodb
sodb = pysodb.SODB()
```

Proceed to load the first spatial proteomics dataset using Pysodb:

```
dataset_name = 'keren2018a'
```

```
experiment_name = 'p4'
adata = sodb.load_experiment(dataset_name,experiment_name)
```

Create a dictionary named 'adata_list,' modify the names in the 'adata' object by appending '_p4', and save a copy of the modified 'adata' object in the dictionary with the key 'p4'. Execute the following commands:

```
adata_list = {}
adata.obs_names = [x+'_p4' for x in adata.obs_names]
adata_list['p4'] = adata.copy()
```

Continuing further, load the second spatial proteomics dataset with Pysodb:

```
dataset_name = 'keren2018a'
experiment_name = 'p9'
adata = sodb.load_experiment(dataset_name,experiment_name)
```

Similarly, update names in another 'adata' object by adding '_p9,' and save a copy of the modified object in the 'adata_list' dictionary under the key 'p9,' execute the following commands:

```
adata.obs_names = [x+'_p9' for x in adata.obs_names]
adata_list['p9'] = adata.copy()
```

24. Running STAGATE for training

Firstly, construct spatial neighbor networks and calculate statistics for 'adata_list['p4']' and 'adata_list['p9']' by executing the following commands:

```
STAGATE.Cal_Spatial_Net(adata_list['p4'], rad_cutoff=50)
STAGATE.Stats_Spatial_Net(adata_list['p4'])

STAGATE.Cal_Spatial_Net(adata_list['p9'], rad_cutoff=50)
STAGATE.Stats_Spatial_Net(adata_list['p9'])
```

Next, train the STAGATE model on each individual sample in the adata_list, execute the following command:

```
for section_id in ['p4', 'p9']:
    adata_list[section_id] = STAGATE.train_STAGATE(adata_list[section_id],n_epochs=500)
```

Then, concatenate an *'adata'* object stored in the *'adata_list'* dictionary with the keys *'*p4*'* and *'*p9,*'* enter the following command:

```
adata = sc.concat([adata_list['p4'], adata_list['p9']], keys=None)
```

Calculates neighbors in the 'STAGATE' representation, applies UMAP, and performs leiden clustering，enter the following commands:

```
sc.pp.neighbors(adata, use_rep='STAGATE')
sc.tl.umap(adata)
sc.tl.leiden(adata,resolution=0.08)
```

Save UMAP and Leiden clustering results before integration and delete the STAGATE embedding from each individual sample，enter the following commands:

```
adata.obsm['UMAP_before'] = adata.obsm['X_umap']
adata.obs['leiden_before'] = adata.obs['leiden']
del adata.obsm['STAGATE']
```

Combine their 'Spatial_Net,' and summarize cells and edges information for the whole adata, enter the following commands:

```
adata.uns['Spatial_Net'] = pd.concat([adata_list['p4'].uns['Spatial_Net'], adata_list['p9'].uns['Spatial_Net']])
STAGATE.Stats_Spatial_Net(adata)
```

In the subsequent step, train a STAGATE model on the whole samples by executing the following command:

```
adata = STAGATE.train_STAGATE(adata, n_epochs=500)
```

Next, create a new column 'Sample' by splitting each name and selecting the last element:

```
adata.obs['Sample'] = [x.split('_')[-1] for x in adata.obs_names]
```

Immediately following, visualize the UMAP projection (across different samples) , UMAP embedding (for spatial clustering), and spatial distribution (for spatial clustering) before integration, execute the following commands:

```
# Plot a UMAP projection across different samples before integration
plt.rcParams["figure.figsize"] = (3, 3)
sc.pl.embedding(adata, basis= 'UMAP_before', color='Sample', title='Unintegrated',show=False,palette=cmp_old_biotech)

# Generate a plot of the UMAP embedding colored by leiden before integration
plt.rcParams["figure.figsize"] = (3, 3)
sc.pl.embedding(adata, basis= 'UMAP_before',
color='leiden_before',show=False,palette=cmp_old)

# Display spatial distribution of cells colored by leiden clustering for two samples ('p4' and 'p9')
fig, axs = plt.subplots(1, 2, figsize=(6, 3))
it=0
for temp_tech in ['p4', 'p9']:
    temp_adata = adata[adata.obs['Sample']==temp_tech, ]
    if it == 1:
        ax = sc.pl.embedding(temp_adata, basis="spatial", color="leiden_before",s=6, ax=axs[it],
                             show=False, title=temp_tech)
        ax.axis('equal')
    else:
        ax = sc.pl.embedding(temp_adata, basis="spatial", color="leiden_before",s=6, ax=axs[it], legend_loc=None,
                             show=False, title=temp_tech)
        ax.axis('equal')
    it+=1
# See Supplementary Fig. 4c (middle)
```

25. Perform Harmony for spatial data intergration

After obtaining the *'STAGATE'* representation in the aforementioned steps, a *'run_harmony'* function is utilized to integrate spatial data. The Harmony algorithm identifies shared structures across different batches and aligns them in a common space. Enter the following commands:

```
data_mat = adata.obsm['STAGATE'].copy()
meta_data = adata.obs.copy()
ho = hm.run_harmony(data_mat, meta_data, ['Sample'])
```

Post-process the Harmony output to prepare the data for downstream analysis. It involves creating a Pandas data frame with the correlation matrix of the Harmony-corrected data, assigning sample names to the data frame, mapping spatial coordinates to the Harmony-corrected data, and adding a 'Sample' column to the metadata by mapping the 'Sample' information from the original dataset. Execute the following commands:

```
res = pd.DataFrame(ho.Z_corr)
res.columns = adata.obs_names
adata_Harmony = sc.AnnData(res.T)
adata_Harmony.obsm['spatial'] = pd.DataFrame(adata.obsm['spatial'], index=adata.obs_names).loc[adata_Harmony.obs_names,].values
adata_Harmony.obs['Sample'] = adata.obs.loc[adata_Harmony.obs_names, 'Sample']
```

Create UMAP and spatial embedding plots with 'leiden' clustering after integration by executing the following commands:

```
sc.pp.neighbors(adata_Harmony)
sc.tl.umap(adata_Harmony)
sc.tl.leiden(adata_Harmony, resolution=0.08)
```

Next, save UMAP and Leiden clustering results after integration, enter the following commands:

```
adata.obsm['UMAP_after'] = adata_Harmony.obsm['X_umap']
adata.obs['leiden_after'] = adata_Harmony.obs['leiden']
```

Lastly, visualize the UMAP projection (across different samples), UMAP embedding (for spatial clustering), and spatial distribution (for spatial clustering) after the integration process, execute the following commands:

```
# Plot a UMAP projection different samples after integration
plt.rcParams["figure.figsize"] = (3, 3)
sc.pl.embedding(adata, basis= 'UMAP_after', color='Sample', title='STAGATE + Harmony',show=False, palette=cmp_old_biotech)

# Generate a plot of the UMAP embedding colored by leiden after integration
plt.rcParams["figure.figsize"] = (3, 3)
sc.pl.embedding(adata, basis= 'UMAP_after', color='leiden_after', show=False, palette=cmp_old)

# Display spatial distribution of cells colored by leiden clustering for two samples ('p4' and 'p9') after integration
fig, axs = plt.subplots(1, 2, figsize=(6, 3))
it=0
for temp_tech in ['p4', 'p9']:
        temp_adata = adata[adata.obs['Sample']==temp_tech, ]
        if it == 1:
                ax = sc.pl.embedding(temp_adata, basis="spatial", color="leiden_after",s=6, ax=axs[it],
                                        show=False, title=temp_tech)
                ax.axis('equal')
        else:
                ax = sc.pl.embedding(temp_adata, basis="spatial", color="leiden_after",s=6, ax=axs[it],
legend_loc=None,
                                        show=False, title=temp_tech)
                ax.axis('equal')
        it+=1
# See Supplementary Fig. 4c (right)
```

Spatial data integration for Spatial proteomics data2 (Timing ~ 11~12 min)

This section showcases the use of Pysodb, STAGATE, and Harmony in the integration of additional spatial proteomics data, specifically data gathered from a cell line using Iterative Indirect Immunofluorescence Imaging.

26. Import packages and set configurations, enter the following commands:

```
import pandas as pd
import scanpy as sc
import matplotlib.pyplot as plt
import STAGATE_pyG as STAGATE
import harmonypy as hm

import palettable
cmp_old = palettable.cartocolors.qualitative.Bold_10.mpl_colors
cmp_old_biotech = palettable.cartocolors.qualitative.Safe_4.mpl_colors
```

27. Import, initialize, and load data using Pysodb:

```
import pysodb
sodb = pysodb.SODB()
```

Then define a section_list with samples from different experiments and load the experiments, enter the following commands:

```
section_list = ['cell_129', 'cell_143', 'cell_140', 'cell_127']

dataset_name = 'gut2018multiplexed'
adata_list = {}
for section_id in section_list:
    temp_adata = sodb.load_experiment(dataset_name,section_id)
    temp_adata.var_names_make_unique()
    temp_adata.obs_names = [x+'_'+section_id for x in temp_adata.obs_names]
    adata_list[section_id] = temp_adata.copy()
```

Then, visualize different experiments color by 'cluster', enter the following commands:

```
fig, axs = plt.subplots(1, 4, figsize=(12, 3))
it=0
for section_id in section_list:
    if it == 3:
        ax = sc.pl.embedding(adata_list[section_id], basis= 'spatial', ax=axs[it],
                             color=['cluster'], title=section_id, show=False)
        ax.axis('equal')
    else:
        ax = sc.pl.embedding(adata_list[section_id], basis= 'spatial', ax=axs[it],
                             color=['cluster'], title=section_id, show=False)
        ax.axis('equal')
    it+=1
```

28. Running STAGATE for training
To begin, spatial neighbor networks should be constructed and statistics calculated separately for different samples in the 'adata_list,' execute the following command:

```
for section_id in section_list:
    STAGATE.Cal_Spatial_Net(adata_list[section_id], rad_cutoff=3)
    STAGATE.Stats_Spatial_Net(adata_list[section_id])
```

Next, proceed to train the STAGATE model on each individual sample present in the 'adata_list', run the following command:

```
for section_id in section_list:
    adata_list[section_id] = STAGATE.train_STAGATE(adata_list[section_id],n_epochs= 1500)
```

Next, Concatenate each individual sample in the adata_list into a AnnData object named 'adata_before', execute the following command:

```
adata_before = sc.concat([adata_list[x] for x in section_list], keys=None)
```

Calculate the nearest neighbors in the 'STAGATE' representation and computes the UMAP embedding, and use Mclust_R to cluster cells in the 'STAGATE' representation into 10 clusters, execute the following commands:

```
sc.pp.neighbors(adata_before, use_rep='STAGATE')
sc.tl.umap(adata_before)
adata_before = STAGATE.mclust_R(adata_before, used_obsm='STAGATE', num_cluster=10)
adata_before.obs['mclust10_before'] = adata_before.obs['mclust']
```

Next, concatenate each individual sample in the adata_list into another new AnnData object named 'adata' by executing the following command:

```
adata = sc.concat([adata_list[x] for x in section_list], keys=None)
```

Save UMAP and mclust clustering results before integration and delete the STAGATE embedding from each individual sample ,enter the following commands:

```
adata.obsm['UMAP_before'] = adata_before.obsm['X_umap']
adata.obs['mclust10_before'] = adata_before.obs['mclust10_before']
del adata.obsm['STAGATE']
```

To amalgamate their 'Spatial_Net' and summarize cells and edges information for the entire 'adata', execute the following commands:

```
adata.uns['Spatial_Net'] = pd.concat([adata_list[x].uns['Spatial_Net'] for x in section_list])
STAGATE.Stats_Spatial_Net(adata)
```

In the next step, train a STAGATE model on all samples collectively by running the following command:

```
adata = STAGATE.train_STAGATE(adata, n_epochs= 1500)
```

Construct a new 'Sample' column by dividing each name and selecting the last two elements, execute the following command:

```
adata.obs['Sample'] = [x.split('_')[-2] + '_' + x.split('_')[-1] for x in adata.obs_names]
```

Next, visualize the UMAP projection (across different samples), UMAP embedding (for spatial clustering), and spatial distribution (for spatial clustering) before the integration, run the following commands:

```
# Plot a UMAP projection before integration
plt.rcParams["figure.figsize"] = (3, 3)
sc.pl.embedding(adata, basis= 'UMAP_before', color='Sample', title='Unintegrated',show=False,
palette=cmp_old_biotech)

# Generate a plot of the UMAP embedding colored by mclust before integration
plt.rcParams["figure.figsize"] = (3, 3)
sc.pl.embedding(adata, basis= 'UMAP_before', color='mclust10_before', show=False,
palette=cmp_old)

# Display spatial distribution of cells colored by mclust clustering for four samples
fig, axs = plt.subplots(1, 4, figsize=(12, 3))
it=0
for section_id in section_list:
    ax = sc.pl.embedding(adata[adata.obs['Sample']==section_id], basis= 'spatial', ax=axs[it],
                          color=['mclust10_before'], title=section_id, show=False)
    ax.axis('equal')
    it+=1
```

```
# See Supplementary Fig. 4f (middle)
```

29. Perform Harmony for spatial data intergration

Upon obtaining the 'STAGATE' representation from the previous steps, the 'run_harmony' function is employed to integrate the spatial data. The Harmony algorithm detects shared structures across different batches and aligns them within a unified space, execute the following commands:

```
data_mat = adata.obsm['STAGATE'].copy()
meta_data = adata.obs.copy()
ho = hm.run_harmony(data_mat, meta_data, ['Sample'])
```

Post-process the Harmony output to ready the data for subsequent analysis. This involves constructing a Pandas data frame from the correlation matrix of the Harmony-corrected data, assigning sample names to the data frame, mapping spatial coordinates to the Harmony-corrected data, and appending a 'Sample' column to the metadata by mapping the 'Sample' information from the original dataset, run the following commands:

```
res = pd.DataFrame(ho.Z_corr)
res.columns = adata.obs_names
adata_Harmony = sc.AnnData(res.T)
adata_Harmony.obsm['spatial'] = pd.DataFrame(adata.obsm['spatial'], index=adata.obs_names).loc[adata_Harmony.obs_names,].values
adata_Harmony.obs['Sample'] = adata.obs.loc[adata_Harmony.obs_names, 'Sample']
```

Calculate the nearest neighbors and generate the UMAP embedding for the integrated data. Subsequently, apply Mclust_R to cluster the 'Harmony' representation into four clusters, input the following commands:

```
sc.pp.neighbors(adata_Harmony)
sc.tl.umap(adata_Harmony)
adata_Harmony.obsm['Harmony'] = adata_Harmony.X
adata_Harmony = STAGATE.mclust_R(adata_Harmony, used_obsm='Harmony', num_cluster=4)
adata_Harmony.obs['mclust4_after'] = adata_Harmony.obs['mclust']
```

Next, save UMAP and mclust clustering results after integration, enter the following commands:

```
adata.obsm['UMAP_after'] = adata_Harmony.obsm['X_umap']
adata.obs['mclust4_after'] = adata_Harmony.obs['mclust4_after']
```

Lastly, visualize the UMAP projection (across different samples), UMAP embedding (for spatial clustering), and spatial distribution (for spatial clustering) after the integration, execute the following commands:

```
# Plot a UMAP projection different samples after integration
plt.rcParams["figure.figsize"] = (3, 3)
sc.pl.embedding(adata, basis= 'UMAP_after', color='Sample', title='STAGATE + Harmony',show=False, palette=cmp_old_biotech)

# Generate a plot of the UMAP embedding colored by mclust after integration
plt.rcParams["figure.figsize"] = (3, 3)
sc.pl.embedding(adata, basis= 'UMAP_after', color='mclust4_after', show=False, palette=cmp_old)

# Display spatial distribution of cells colored by mclust clustering for four samples after integration
fig, axs = plt.subplots(1, 4, figsize=(12, 3))
it=0
for section_id in section_list:
    ax = sc.pl.embedding(adata[adata.obs['Sample']==section_id], basis= 'spatial', ax=axs[it],
                         color=['mclust4_after'], title=section_id, show=False)
    ax.axis('equal')
    it+=1
# See Supplementary Fig. 4f (right)
```