assistant:

Welcome to BioMANIA! How can I help?

user:

The data consist of 3k PBMCs from a Healthy Donor and are freely available from 10x Genomics. Read in the count matrix into an AnnData object, which holds many slots for annotations and different representations of the data.

assistant:

Can you confirm which of the following 3 candidates:
Candidate [1]: scanpy.read_10x_h5
Read 10x-Genomics-formatted hdf5 file.
Candidate [2]: scanpy.read_visium
Read 10x-Genomics-formatted visum dataset.
Candidate [3]: scanpy.read_10x_mtx
Read 10x-Genomics-formatted mtx directory.

user:

3

assistant:

Predicted API: scanpy.read_10x_mtx:
The task is to read a directory containing files in a specific format used by 10x Genomics. We solved it by using the `scanpy.read_10x_mtx` function, which reads the `.mtx` and `.tsv` files in the directory and returns an `AnnData` object. This object contains the data from the files and can be used for further analysis.
Executed results [Success]:
We obtain a new AnnData object with n_obs × n_vars = 2700 × 32738
var: 'gene_ids'
Executed results [Success]:
We visualize the first 5 rows of the table data

Table:

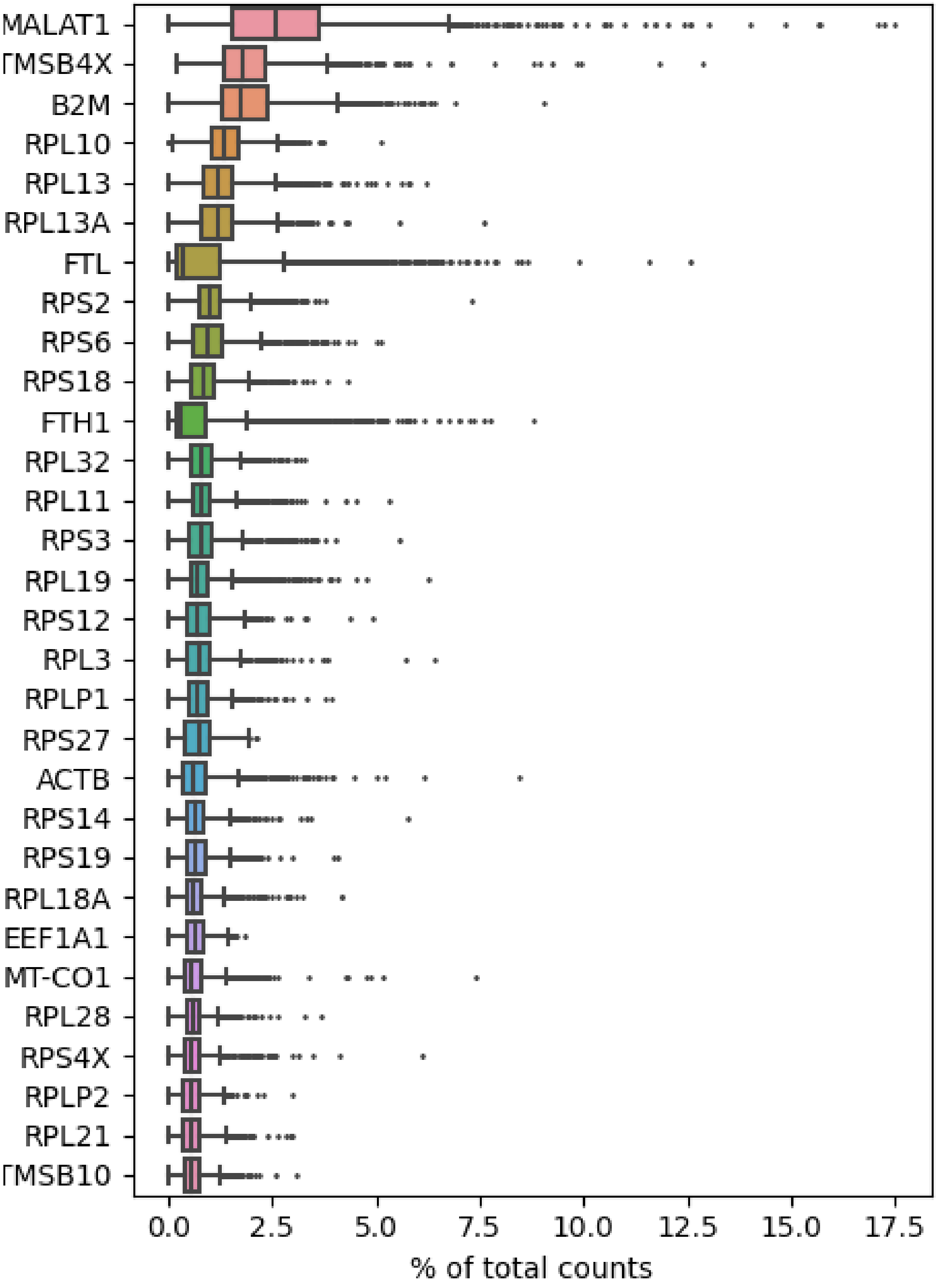
""
AAACATACAACCAC-1
AAACATTGAGCTAC-1
AAACATTGATCAGC-1
AAACCGTGCTTCCG-1
AAACCGTGTATGCG-1

user:

Show those genes that yield the highest fraction of counts in each single cell, across all cells.

assistant:

Predicted API: scanpy.pl.highest_expr_genes:
The task is to identify the genes that have the highest fraction of counts in each single cell, across all cells. We solved this by using the `scanpy.pl.highest_expr_genes` API function, which calculates the fraction of counts assigned to each gene within a cell and plots the top genes as boxplots. This helps in identifying important genes and potential issues with the data.
Executed results [Success]:
We visualize the obtained figure



user:

filtered out genes that are detected in less than 3 cells

assistant:

Non API chitchat:

To filter out genes that are detected in less than 3 cells, you can use the following steps:
1. Calculate the number of cells in which each gene is detected. This can be done by summing the expression values of each gene across all cells.
2. Create a filter based on the number of cells. In this case, you want to keep genes that are detected in at least 3 cells. So, any gene with a count less than 3 should be filtered out.
3. Apply the filter to your gene expression data. This can be done by removing the rows (genes) that do not meet the filter criteria.

Here is an example code snippet in Python using pandas library to perform the filtering:
`` `python ``
import pandas as pd

Assuming you have a dataframe called 'gene_expression_data' with genes as rows and ce

columns

Step 1: Calculate the number of cells in which each gene is detected

gene_counts = gene_expression_data.sum(axis=1)

Step 2: Create a filter based on the number of cells

filter_criteria = gene_counts >= 3

Step 3: Apply the filter to your gene expression data

filtered_gene_expression_data = gene_expression_data[filter_criteria]
`` ` ``

After executing this code, the `filtered_gene_expression_data` dataframe will contain only the genes that are detected in at least 3 cells.

user:

Can you compute quality control metrics for the data?

assistant:

Predicted API: scanpy.pp.calculate_qc_metrics:
The task is to compute quality control metrics for a given dataset. We solved it by using the `scanpy.pp.calculate_qc_metrics` function, which calculates various metrics such as the number of genes with positive counts in a cell, the total number of counts for a cell, and the proportion of counts for specific variables. The function returns these metrics as a DataFrame or updates the dataset with the calculated metrics.
Executed results [Success]:

user:

A violin plot of some of the computed quality measures:
1. the number of genes expressed in the count matrix
2. the total counts per cell
3. the percentage of counts in mitochondrial genes

assistant:

Predicted API: scanpy.pl.violin:
The task is to create a violin plot showing the number of genes expressed, total counts per cell, and the percentage of counts in mitochondrial genes. We solved it by using the `scanpy.pl.violin` function, which wraps the `seaborn.violinplot` function and allows us to plot these measures using the provided parameters. The function returns a matplotlib axes object that can be used to further customize or save the plot.

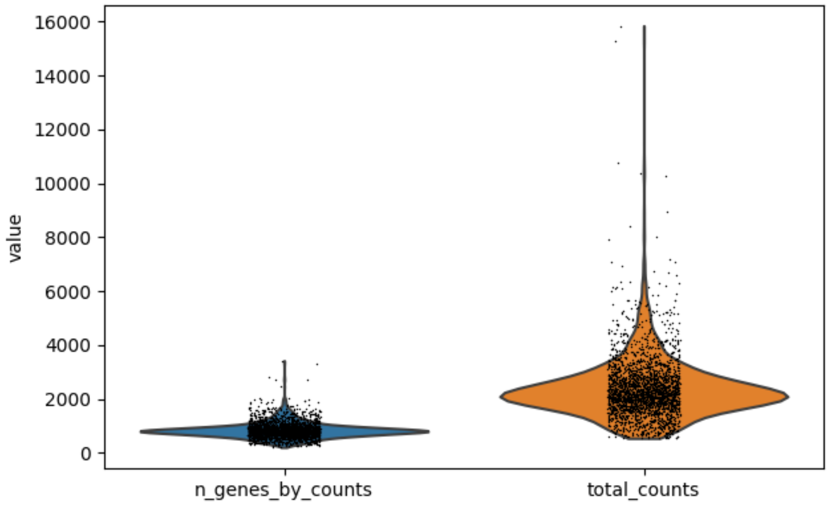
Enter Parameters: basic type:
The predicted API takes Keys for accessing variables of `.var_names` or fields of `.obs`.('keys': Union[str, Sequence[str]]), as input. However, there are still some parameters undefined in the query.
Enter Parameters: basic type:
Which value do you think is appropriate for the parameters 'keys'?

user:

['n_genes_by_counts', 'total_counts']

assistant:

Executed results [Success]:
We visualize the obtained figure



user:

Remove cells that have too many mitochondrial genes expressed or too many total counts:

assistant:

Non API chitchat:
To remove cells that have too many mitochondrial genes expressed or too many total counts, you can follow these steps:
1. Calculate the total counts for each cell in your dataset. This can be done by summing up the expression values for all genes in each cell.
2. Calculate the number of mitochondrial genes expressed in each cell. You can do this by counting the number of genes in your dataset that are annotated as mitochondrial genes and have non-zero expression values in each cell.
3. Define a threshold for the maximum number of mitochondrial genes expressed or the maximum total counts allowed. This threshold can be determined based on the characteristics of your dataset or by considering the distribution of these values.
4. Create a filter to identify cells that exceed the defined threshold for either the number of

mitochondrial genes expressed or the total counts. For example, you can create a logical filter that returns "True" for cells that meet the criteria and "False" for cells that do not.
5. Apply the filter to your dataset to remove the cells that exceed the threshold. This can be done by subsetting your dataset based on the logical filter.
Here is an example code snippet in Python using the scanpy library to remove cells with too many mitochondrial genes expressed or too many total counts:

```python
import scanpy as sc
# Assuming your dataset is stored in an AnnData object called 'adata'
# Calculate total counts for each cell
adata.obs['total_counts'] = adata.X.sum(axis=1)
# Calculate number of mitochondrial genes expressed in each cell
mito_genes = adata.var_names.str.startswith('MT-')
adata.obs['n_mito_genes'] = adata[:, mito_genes].X.sum(axis=1)
# Define thresholds for maximum mitochondrial genes expressed and total counts
max_mito_genes = 10
max_total_counts = 5000
# Create filter for cells that exceed the thresholds
filter_cells = ((adata.obs['n_mito_genes'] <= max_mito_genes) &
(adata.obs['total_counts'] <= max_total_counts))
# Apply the filter to remove cells that exceed the thresholds
adata_filtered = adata[filter_cells, :]
# Remove the temporary columns added for calculations
adata_filtered.obs.drop(['total_counts', 'n_mito_genes'], axis=1, inplace=True)
```

In this example, cells with more than 10 mitochondrial genes expressed or more than 5000 total counts will be removed from the dataset. Adjust the threshold values according to your specific requirements.

user:

Can you create a scatter plot for the data?

assistant:

Predicted API: scanpy.pl.scatter:
The task is to create a scatter plot for the given data. We solved it by using the `scanpy.pl.scatter` function, which allows us to plot the data points and color them based on annotations. The function provides various parameters to customize the plot, such as choosing the x and y coordinates, selecting the color scheme, and specifying the point size.
Executed results [Fail]:
Execution failed! Either provide a `basis` or `x` and `y`.

user:

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

assistant:

Predicted API: scanpy.pp.normalize_total:

The task is to normalize the data to have a consistent count of 10,000 reads per cell, making the counts comparable among cells. We solved this by using the `scanpy.pp.normalize_total` function, which normalizes each cell by the total counts over all genes. This ensures that every cell has the same total count after normalization.
Executed results [Success]:

user:

Logarithmize the data

assistant:

Predicted API: scanpy.pp.log1p:
The task is to logarithmize the data matrix, which means taking the natural logarithm of each element in the matrix. We solved it by using the `scanpy.pp.log1p` function, which computes the logarithm of the data matrix by adding 1 to each element and then taking the natural logarithm. The function also allows for specifying a different base for the logarithm if needed.
Executed results [Success]:

user:

Identify highly-variable genes.

assistant:

Can you confirm which of the following 3 candidates:
Candidate [1]: scanpy.pp.highly_variable_genes
Annotate highly variable genes [Satija15]_ [Zheng17]_ [Stuart19]_.
Candidate [2]: scanpy.pl.highly_variable_genes
Plot dispersions or normalized variance versus means for genes.
Candidate [3]: scanpy.experimental.pp.highly_variable_genes
Select highly variable genes using analytic Pearson residuals [Lause21]_.

user:

1

assistant:

Predicted API: scanpy.pp.highly_variable_genes:
The task is to identify highly variable genes in a dataset. We solved it by using the `scanpy.pp.highly_variable_genes` function, which annotates the highly variable genes based on different methods such as Seurat and Cell Ranger. The function returns a DataFrame with calculated metrics or updates the dataset with fields indicating highly variable genes.
Executed results [Success]:

user:

Identify highly-variable genes.

assistant:

Can you confirm which of the following 3 candidates:

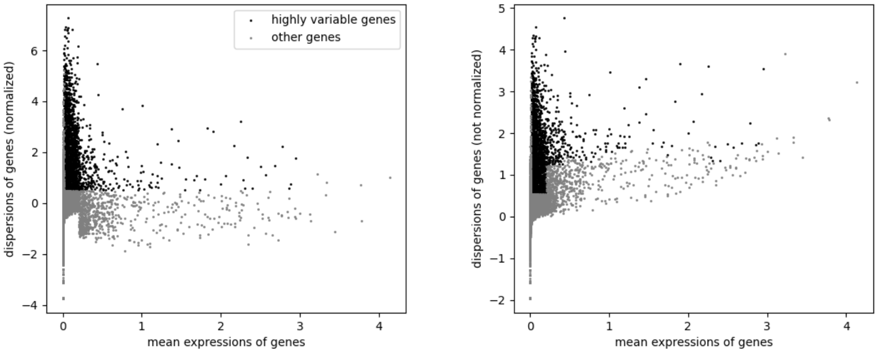
Candidate [1]: scanpy.pp.highly_variable_genes
Annotate highly variable genes [Satija15]_ [Zheng17]_ [Stuart19]_.
Candidate [2]: scanpy.pl.highly_variable_genes
Plot dispersions or normalized variance versus means for genes.
Candidate [3]: scanpy.experimental.pp.highly_variable_genes
Select highly variable genes using analytic Pearson residuals [Lause21]_.

user:

2

assistant:

Predicted API: scanpy.pl.highly_variable_genes:
The task is to plot dispersions or normalized variance versus means for genes. We solved it by using the `scanpy.pl.highly_variable_genes` function, which produces the desired plot. The function takes parameters such as whether to plot on logarithmic axes and whether to show or save the plot. The function returns a variable with no specific type or description.
Executed results [Success]:
We visualize the obtained figure



user:

Regress out effects of total counts per cell and the percentage of mitochondrial genes expressed. Scale the data to unit variance.

assistant:

Predicted API: scanpy.pp.regress_out:
The task is to remove unwanted sources of variation from the data, such as total counts per cell and the percentage of mitochondrial genes expressed, and then scale the data to have the same variance. We solved this by using the `scanpy.pp.regress_out` function, which performs linear regression to remove these unwanted effects. The function takes the data matrix, the keys for observation annotation to regress on, and the number of parallel jobs as parameters, and returns the corrected data matrix.
Enter Parameters: basic type:
The predicted API takes Keys for observation annotation on which to regress on.('keys': Union[str, Sequence[str]]), as input. However, there are still some parameters undefined in the query.
Enter Parameters: basic type:
Which value do you think is appropriate for the parameters 'keys'?

user:

['total_counts']

assistant:

Executed results [Success]:

user:

Can you standardize the data to have zero mean and unit variance?

assistant:

Predicted API: scanpy.pp.scale:
The task is to standardize the data by making its mean zero and variance one. We solved it by using the `scanpy.pp.scale` function, which scales the data to have unit variance and zero mean. The function also handles variables that do not display any variation by setting them to zero.
Executed results [Success]:

user:

Reduce the dimensionality of the data by running principal component analysis (PCA), which reveals the main axes of variation and denoises the data.

assistant:

Can you confirm which of the following 3 candidates:
Candidate [1]: scanpy.tl.pca
Principal component analysis [Pedregosa11]_.
Candidate [2]: scanpy.pl.pca
Scatter plot in PCA coordinates.
Candidate [3]: scanpy.pp.pca
Principal component analysis [Pedregosa11]_.

user:

1

assistant:

Predicted API: scanpy.tl.pca:
The task is to perform Principal Component Analysis (PCA) on a dataset, which helps to reduce the dimensionality of the data and identify patterns. We solved this task by using the `scanpy.tl.pca` function from the Scanpy library, which implements PCA using the scikit-learn library. The function takes the data matrix as input and returns the PCA coordinates, loadings, and variance decomposition.
Executed results [Success]:
Execute success. We didn't obtain new variable

user:

We can make a scatter plot in the PCA coordinates

assistant:

Can you confirm which of the following 3 candidates:

Candidate [1]: scanpy.tl.pca

Principal component analysis [Pedregosa11]_.

Candidate [2]: scanpy.pl.pca

Scatter plot in PCA coordinates.

Candidate [3]: scanpy.pp.pca

Principal component analysis [Pedregosa11]_.

user:

2

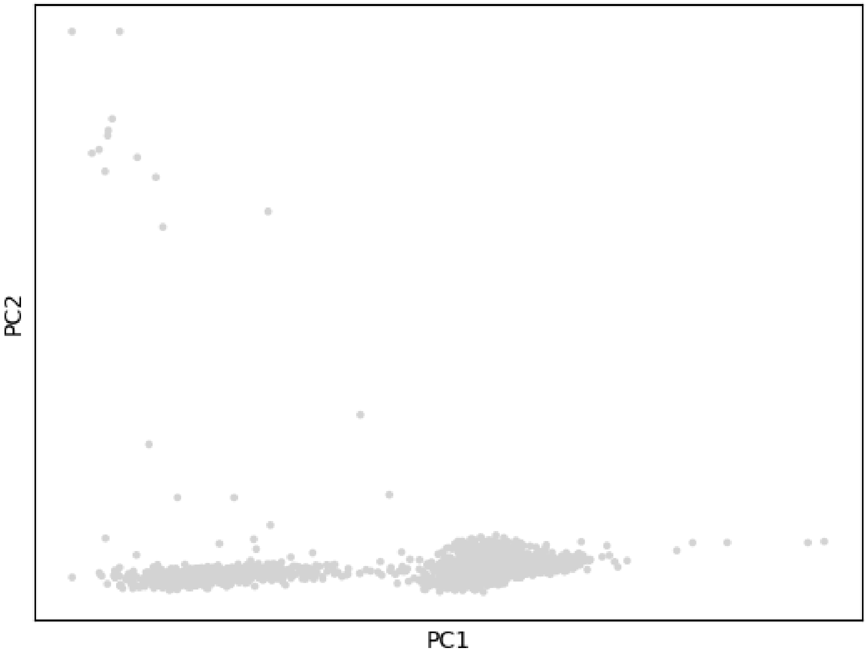
assistant:

Predicted API: scanpy.pl.pca:

The task is to create a scatter plot in PCA coordinates and annotate the explained variance. We solved it by using the `scanpy.pl.pca` function with the `annotate_var_explained` parameter set to `True`. This function returns a matplotlib axes object that can be used to show or save the plot.

Executed results [Success]:

We visualize the obtained figure



user:

Can you plot the variance ratio of the principal components for this data with 10 principal components?

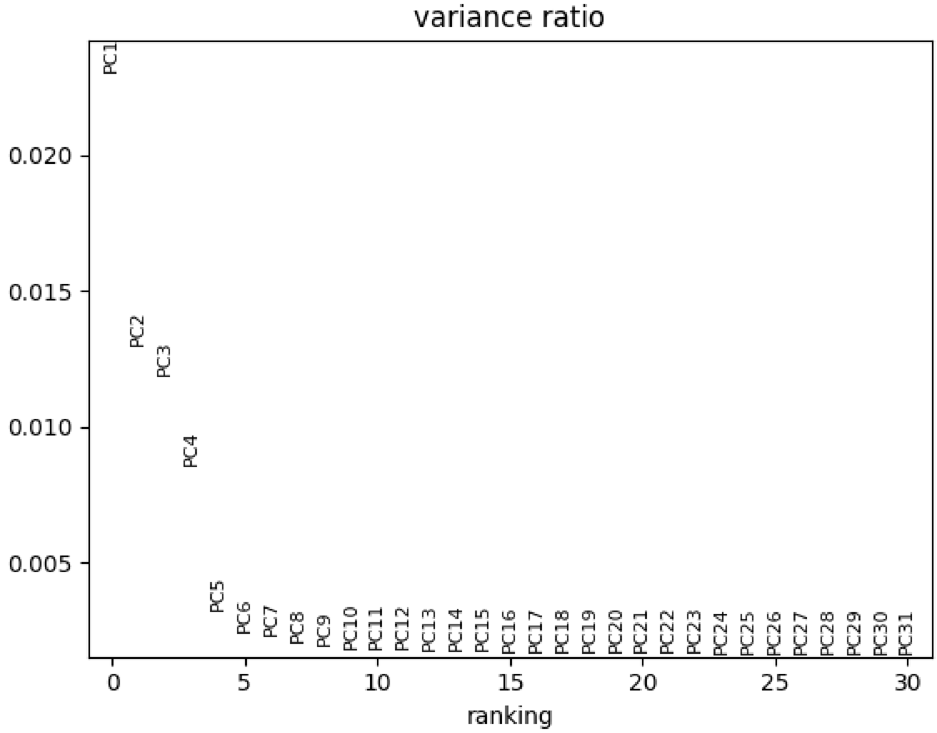
assistant:

Predicted API: scanpy.pl.pca_variance_ratio:

The task is to plot the variance ratio of the principal components for a given dataset with 10 principal components. We solved it by using the `scanpy.pl.pca_variance_ratio` API function, which plots the variance ratio. The function takes the dataset, the number of principal components to show, and other optional parameters. The function does not return any variable.

Executed results [Success]:

We visualize the obtained figure



user:

Let us compute the neighborhood graph of cells using the PCA representation of the data matrix.

assistant:

Predicted API: scanpy.pp.neighbors:

The task is to compute the neighborhood graph of cells using the PCA representation of the data matrix. We solved it by using the `scanpy.pp.neighbors` function, which calculates the neighborhood graph of observations based on the provided data. The function uses either UMAP or Gaussian kernel to estimate the connectivities of data points, and returns the weighted adjacency matrix and distances for each pair of neighbors.

Executed results [Success]:

```
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/umap/distances.py:1063:
NumbaDeprecationWarning: ■[1mThe 'nopython' keyword argument was not supplied to the
'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-
for details.■[0m
@numba.jit()
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/umap/distances.py:1071:
```

```
NumbaDeprecationWarning: ■[1mThe 'nopython' keyword argument was not supplied to the
'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-
for details.■[0m
@numba.jit()
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/umap/distances.py:1086:
NumbaDeprecationWarning: ■[1mThe 'nopython' keyword argument was not supplied to the
'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-
for details.■[0m
@numba.jit()
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/umap/umap_.py:660:
NumbaDeprecationWarning: ■[1mThe 'nopython' keyword argument was not supplied to the
'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-
for details.■[0m
@numba.jit()
```

user:

Can you project the neighborhood graph of the annotated data matrix into 2 dimensions using UMAP?

assistant:

Predicted API: embed_umap_paga_graph:
The task is to project the neighborhood graph of a data matrix into two dimensions using UMAP, which helps to visualize the data in a simplified way. We solved this by using the `embed_umap_paga_graph` API function, which takes an annotated data matrix as input and applies UMAP to embed the graph into two dimensions. The function returns an annotated data matrix with the embedded neighborhood graph in two dimensions, making it easier to analyze and understand the data.
Executed results [Success]:

user:

We suggest embedding the graph in two dimensions using UMAP (McInnes et al., 2018)

assistant:

Can you confirm which of the following 2 candidates:
Candidate [1]: scanpy.pl.umap
Scatter plot in UMAP basis.
Candidate [2]: scanpy.tl.umap
Embed the neighborhood graph using UMAP [McInnes18]_.

user:

1

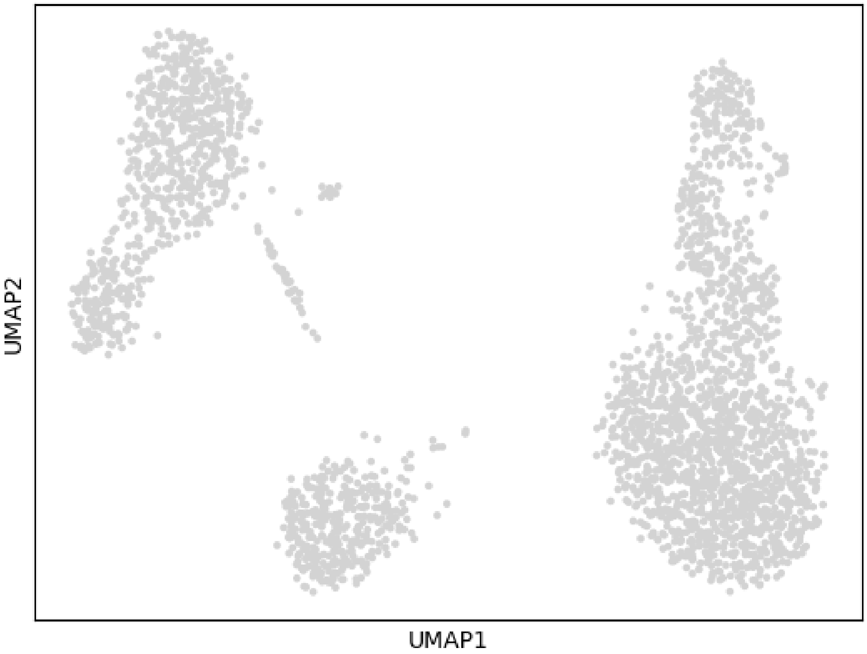
assistant:

Predicted API: scanpy.pl.umap:

The task is to create a scatter plot using the UMAP basis. We solved it by using the `scanpy.pl.umap` function from the Scanpy library, which takes in the annotated data matrix and the color annotations as parameters. The function returns a matplotlib axes object or a list of axes if multiple components are plotted.

Executed results [Success]:

We visualize the obtained figure



user:

As with Seurat and many other frameworks, we recommend the Leiden graph-clustering method (community detection based on optimizing modularity) by Traag et al. (2018)

assistant:

Can you confirm which of the following 2 candidates:

Candidate [1]: scanpy.tl.leiden

Cluster cells into subgroups [Traag18]_.

Candidate [2]: scanpy.tl.louvain

Cluster cells into subgroups [Blondel08]_ [Levine15]_ [Traag17]_.

user:

1

assistant:

Predicted API: scanpy.tl.leiden:
The task is to cluster cells into subgroups using the Leiden algorithm, which is an improved version of the Louvain algorithm. This requires running the `scanpy.tl.leiden` function after performing neighbor calculations on the data. The function returns the subgroup IDs for each cell and a dictionary with the parameter values used for clustering.
Executed results [Success]:

user:

Visualize a scatter plot in UMAP basis with arrows and groups using the provided annotated data matrix.

assistant:

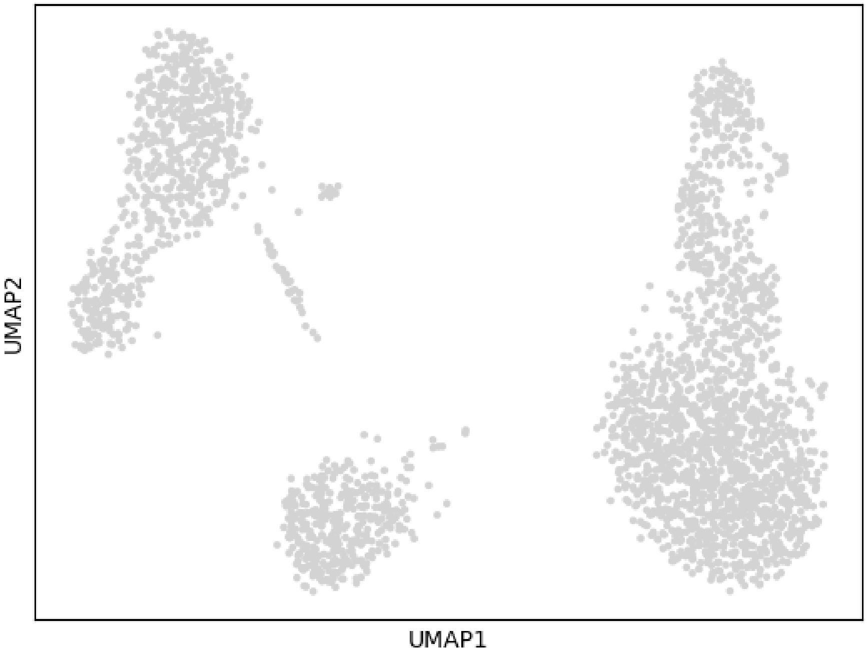
Can you confirm which of the following 2 candidates:
Candidate [1]: scanpy.pl.umap
Scatter plot in UMAP basis.
Candidate [2]: scanpy.tl.umap
Embed the neighborhood graph using UMAP [McInnes18]_.

user:

1

assistant:

Predicted API: scanpy.pl.umap:
The task is to create a scatter plot using the UMAP basis. We solved it by using the `scanpy.pl.umap` function from the Scanpy library, which takes in the annotated data matrix and the color annotations as parameters. The function returns a matplotlib axes object or a list of axes if the plot is not shown.
Executed results [Success]:
We visualize the obtained figure



user:

Let us compute a ranking for the highly differential genes in each cluster.

assistant:

Can you confirm which of the following 2 candidates:
Candidate [1]: scanpy.pl.rank_genes_groups
Plot ranking of genes.
Candidate [2]: scanpy.tl.rank_genes_groups
Rank genes for characterizing groups.

user:

2

assistant:

Predicted API: scanpy.tl.rank_genes_groups:
The task is to rank genes for characterizing groups using logarithmized data. We solved it by using the `scanpy.tl.rank_genes_groups` function from the Scanpy library. This function takes in annotated data matrix, groupby key, and other optional parameters, and returns structured arrays and dataframes containing gene names, scores, logfoldchanges, p-values, and other relevant information.
Enter Parameters: basic type:
The predicted API takes The key of the observations grouping to consider.('groupby': str), as input. However, there are still some parameters undefined in the query.
Enter Parameters: basic type:
Which value do you think is appropriate for the parameters 'groupby'?

user:

leiden

assistant:

Executed results [Success]:

```
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/scanpy/tools/_rank_genes_groups.py:422:
RuntimeWarning: invalid value encountered in log2
self.stats[group_name, 'logfoldchanges'] = np.log2(
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/scanpy/tools/_rank_genes_groups.py:422:
RuntimeWarning: invalid value encountered in log2
self.stats[group_name, 'logfoldchanges'] = np.log2(
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/scanpy/tools/_rank_genes_groups.py:422:
RuntimeWarning: invalid value encountered in log2
self.stats[group_name, 'logfoldchanges'] = np.log2(
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/scanpy/tools/_rank_genes_groups.py:422:
RuntimeWarning: invalid value encountered in log2
self.stats[group_name, 'logfoldchanges'] = np.log2(
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/scanpy/tools/_rank_genes_groups.py:422:
RuntimeWarning: invalid value encountered in log2
self.stats[group_name, 'logfoldchanges'] = np.log2(
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/scanpy/tools/_rank_genes_groups.py:422:
RuntimeWarning: invalid value encountered in log2
self.stats[group_name, 'logfoldchanges'] = np.log2(
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/scanpy/tools/_rank_genes_groups.py:422:
RuntimeWarning: invalid value encountered in log2
self.stats[group_name, 'logfoldchanges'] = np.log2(
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/scanpy/tools/_rank_genes_groups.py:422:
RuntimeWarning: invalid value encountered in log2
self.stats[group_name, 'logfoldchanges'] = np.log2(
/home/z6dong/anaconda3/envs/py310/lib/python3.10/site-packages/scanpy/tools/_rank_genes_groups.py:422:
RuntimeWarning: invalid value encountered in log2
self.stats[group_name, 'logfoldchanges'] = np.log2(
```

user:

Let us compute a ranking for the highly differential genes in each cluster.

assistant:

Can you confirm which of the following 2 candidates:
Candidate [1]: scanpy.pl.rank_genes_groups
Plot ranking of genes.
Candidate [2]: scanpy.tl.rank_genes_groups
Rank genes for characterizing groups.

user:

1

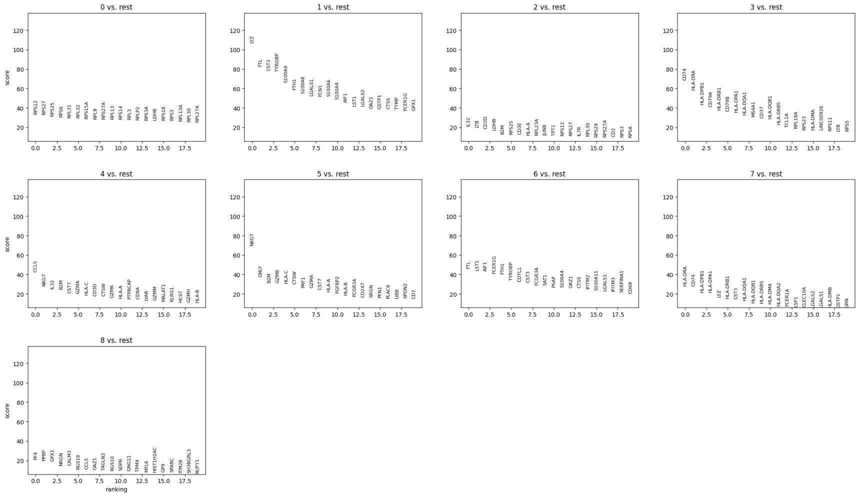
assistant:

Predicted API: scanpy.pl.rank_genes_groups:

The task is to plot the ranking of genes based on certain groups in a dataset. We solved it by using the `scanpy.pl.rank_genes_groups` function, which takes in the annotated data matrix, the groups to show the gene ranking for, and other optional parameters such as the number of genes to show and the fontsize for gene names. The function returns a variable with no specific type or description.

Executed results [Success]:

We visualize the obtained figure



user:

Let us also define a list of marker genes for later reference.

marker_genes = ['IL7R', 'CD79A', 'MS4A1', 'CD8A', 'CD8B', 'LYZ', 'CD14',
'LGALS3', 'S100A8', 'GNLY', 'NKG7', 'KLRB1',
'FCGR3A', 'MS4A7', 'FCER1A', 'CST3', 'PPBP']

Plot the ranking of genes for all tested comparisons using the provided genes from the data.

assistant:

Predicted API: scanpy.tl.marker_gene_overlap:

The task is to plot the ranking of genes for all tested comparisons using a list of marker genes. We solved it by using the `scanpy.tl.marker_gene_overlap` function, which calculates the overlap score between data-derived marker genes and provided markers. The function returns a pandas dataframe that can be used to annotate clusters based on marker gene overlaps.

Enter Parameters: basic type:

The predicted API takes A marker gene dictionary object. Keys should be strings with the
cell identity name and values are sets or lists of strings which match
format of `adata.var_name`.('reference_markers': Union[Dict[str, set], Dict[str, list]]), as input.
However, there are still some parameters undefined in the query.

Enter Parameters: basic type:

Which value do you think is appropriate for the parameters 'reference_markers'?

user:

['IL7R', 'CD79A', 'MS4A1', 'CD8A', 'CD8B', 'LYZ', 'CD14','LGALS3', 'S100A8', 'GNLY', 'NKG7', 'KLRB1', 'FCGR3A', 'MS4A7', 'FCER1A', 'CST3', 'PPBP']

assistant:

Executed results [Fail]:
Execution failed! Writing Pandas dataframes to h5ad is currently under development.
Please use `inplace=False`.

user:

Please visualize using compact violin plot

assistant:

Predicted API: scanpy.pl.stacked_violin:
The task is to create a compact violin plot to visualize gene expression per cluster. We solved it by using the `scanpy.pl.stacked_violin` function, which stacks individual violin plots on top of each other. This function provides a convenient way to create the plot and offers flexibility in customizing the visual parameters.
Enter Parameters: basic type:
The predicted API takes `var_names` should be a valid subset of `adata.var_names`.
If `var_names` is a mapping, then the key is used as label
to group the values (see `var_group_labels`). The mapping values
should be sequences of valid `adata.var_names`. In this
case either coloring or 'brackets' are used for the grouping
of var names depending on the plot. When `var_names` is a mapping,
then the `var_group_labels` and `var_group_positions` are set.('var_names': Union[str, Sequence[str], Mapping[str, Union[str, Sequence[str]]]]), and The key of the observation grouping to consider.('groupby': Union[str, Sequence[str]]), as input. However, there are still some parameters undefined in the query.
Enter Parameters: basic type:
Which value do you think is appropriate for the parameters 'var_names'?

user:

['IL7R', 'CD79A', 'MS4A1', 'CD8A', 'CD8B', 'LYZ', 'CD14','LGALS3', 'S100A8', 'GNLY', 'NKG7', 'KLRB1', 'FCGR3A', 'MS4A7', 'FCER1A', 'CST3', 'PPBP']

assistant:

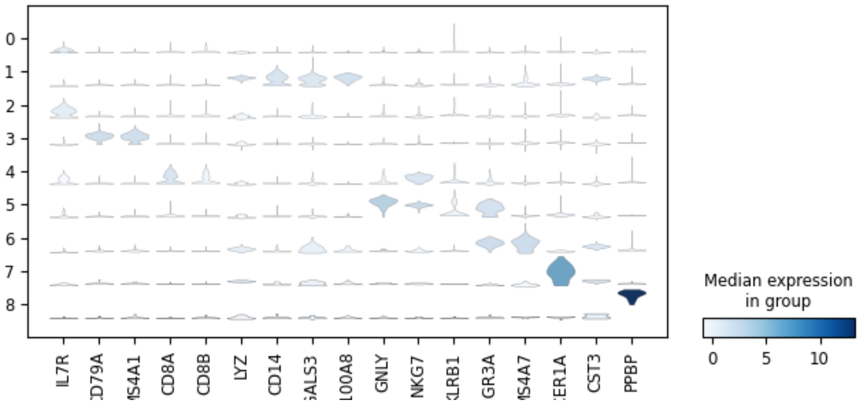
Enter Parameters: basic type:
Which value do you think is appropriate for the parameters 'groupby'?

user:

leiden

assistant:

Executed results [Success]:
We visualize the obtained figure



user:

Can you create a dot plot of gene expression values for two genes, `gene1` and `gene2`?

assistant:

Predicted API: scanpy.pl.dotplot:

The task is to create a dot plot of gene expression values for two genes, `gene1` and `gene2`. We solved it by using the `scanpy.pl.dotplot` function, which plots dots representing the mean expression and fraction of cells expressing the genes for different categories. This allows us to visualize the gene expression across multiple clusters or groups.

Enter Parameters: basic type:

The predicted API takes `var_names` should be a valid subset of `adata.var_names`.
If `var_names` is a mapping, then the key is used as label
to group the values (see `var_group_labels`). The mapping values
should be sequences of valid `adata.var_names`. In this
case either coloring or 'brackets' are used for the grouping
of var names depending on the plot. When `var_names` is a mapping,
then the `var_group_labels` and `var_group_positions` are set.('var_names': Union[str, Sequence[str], Mapping[str, Union[str, Sequence[str]]]]), and The key of the observation grouping to consider.('groupby': Union[str, Sequence[str]]), as input. However, there are still some parameters undefined in the query.

Enter Parameters: basic type:

Which value do you think is appropriate for the parameters 'var_names'?

user:

['IL7R', 'CD79A', 'MS4A1', 'CD8A', 'CD8B', 'LYZ', 'CD14','LGALS3', 'S100A8', 'GNLY', 'NKG7', 'KLRB1', 'FCGR3A', 'MS4A7', 'FCER1A', 'CST3', 'PPBP']

assistant:

Enter Parameters: basic type:

Which value do you think is appropriate for the parameters 'groupby'?

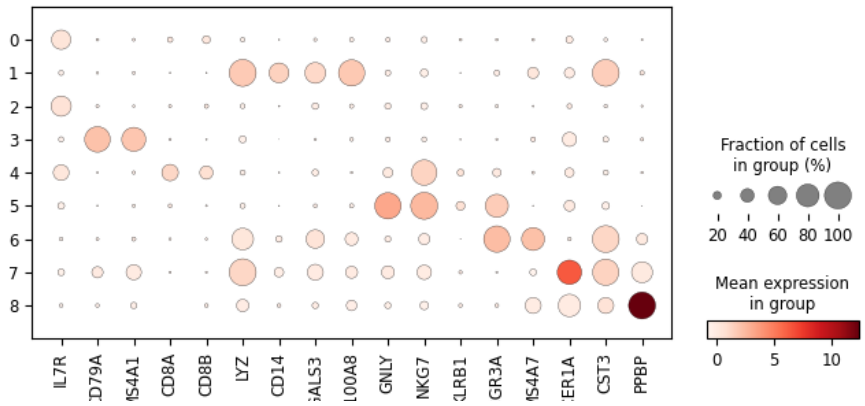
user:

leiden

assistant:

Executed results [Success]:

We visualize the obtained figure


IL7R
D79A
S4A1
CD8A
CD8B
LYZ
CD14
ALS3
100A8
GNLY
NKG7
LRB1
GR3A
S4A7
ER1A
CST3
PPBP
Fraction of cells in group (%)
Mean expression in group