Read data

There are three read functions to read raw scRNA-seq count matrix, and build "GOTermActivity" object.

(1) Read from "Seurat" object

Function `Read_Seurat_scData()` can read the raw matrix from exist "Seurat" object, and store the "Seurat" object in the sub-list of "GOTermActivity" object. In the function:

```
Read_Seurat_scData <- function(Input_Seurat, Str_genes = "mt-")
```

"`Input_Seurat`" is the "Seurat" object you would like to perform further analysis;

"`Str_genes`" is to calculate the proportion of the expression of some genes in the total counts of the cell, using the set of all genes starting with string "`Str_genes`" as a set of your interested genes. The default value (default: mitochondrial genes) is `"mt-"`.

The function returns a list variable including sub-lists "`Original_List`" and "`Seurat_Object`":

"`Seurat_Object`" is the original "Seurat" object the function input;

In the sub-list "`Original_List`", there are 8 variables:

"`Size_orig`": Numbers of genes and cells in the data;

"`Data_orig`": A count matrix of the data; (row: gene, col: cell)

"`Gene_orig`": Gene list of the data; (The order is consistent with the data matrix)

"`Cell_ID_orig`": Cell list of the data; (The order is consistent with the data matrix)

"`Expression_percent_orig`": In a single cell, the percentage of all expressed genes in the gene list;

"`nCount_RNA_log2_orig`": In a single cell, the total counts of all expressed genes; (log2)

"`nFeature_RNA_orig`": In a single cell, the number of genes the cell express;

"`Percent_genes_interested_orig`": In a single cell, the count percentage of the selected gene set (default: mitochondrial genes).

Example data:

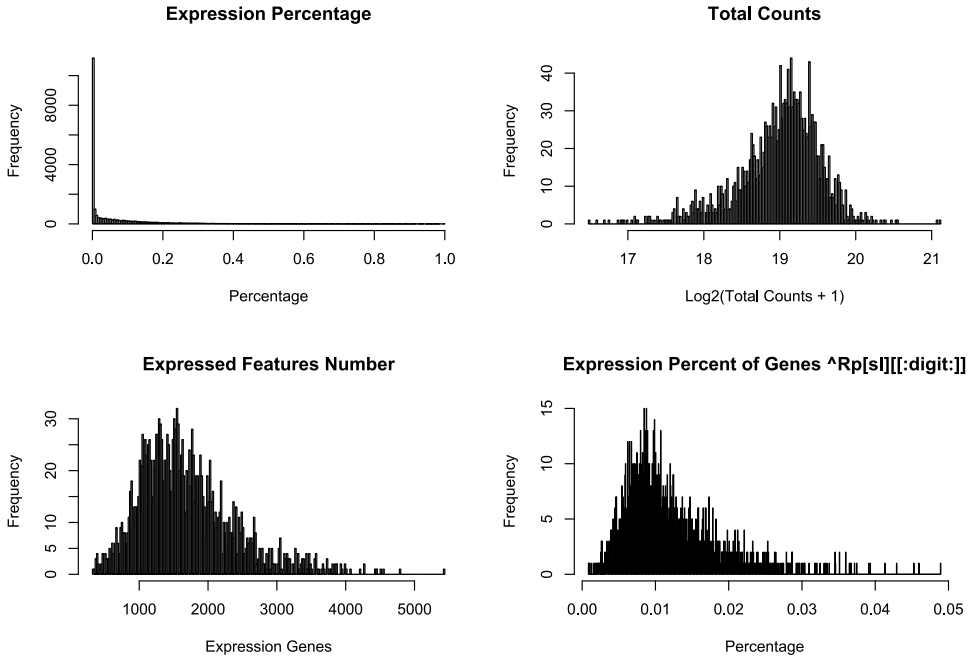
Example data are stored in the "Seurat_Example_for_GOTermActivity.RDS" file, located in the "Example" folder. This dataset consists of scRNA-seq data from the published paper:

Li, Qingyun, Zuolin Cheng, Lu Zhou, Spyros Darmanis, et al. "Developmental heterogeneity of microglia and brain myeloid cells revealed by deep single-cell RNA sequencing." Neuron 101, no. 2 (2019): 207-223.

Example code:

```
Seurat_obj <- readRDS("Seurat_Example_for_GOTermActivity.RDS")
GOTerm_analysis_1 <- Read_Seurat_scData(
  Input_Seurat = Seurat_obj,
  Str_genes = "^Rp[sl][[:digit:]]"
  )
```

The return variable is named as "`GOTerm_analysis`", and we will continue to use this name in subsequent analyses. Besides, the visualized results will also be displayed like this:



(2) Read from count matrix file

Function `Read_CountFile_scData()` can read the original scRNA-seq data count file (format like .txt or .tsv), then create and store the "Seurat" object in the sub-list of "GOTermActivity" object. In the function:

```
Read_CountFile_scData <- function(Input_str, Str_genes = "mt-", project_Name
= "GO_Activity_Analysis")
```

"`Input_str`": The input ".txt" file including its path. The file should have a matrix, including the row names and column names in the first row and column (row names are gene names, and column names are cell IDs);

"`Str_genes`": The same parameter as it in function `Read_Seurat_scData()`;(check "Read from Seurat object")

"`project_Name`": Project name for creating Seurat object.

The function returns a list variable including sub-lists "`Original_List`" and "`Seurat_Object`" (check "Read from Seurat object")

Example code:

```
Input_str <- "Countdata_Example_for_GOTermActivity.tsv"
GOTerm_analysis <- Read_CountFile_scData(
  Input_str = Input_str,
  Str_genes = "^Rp[sl][[:digit:]]"
  )

# Load metadata for the sample
Meta_data <- read.csv("Metadata_Example_for_GOTermActivity.csv", row.names =
1)
GOTerm_analysis_3$Seurat_Object <- Seurat::AddMetaData(
  object = GOTerm_analysis_1$Seurat_Object,
  metadata = Meta_data
  )
```

Check "Read from Seurat object" for the result.

(3) Read from "matrix-format" variable

Function `Read_Count_scData()` can read the original scRNA-seq data variable, then create and store the "Seurat" object in the sub-list of "GOTermActivity" object. In the function:

```
Read_Count_scData <- function(Input_data, Str_genes = "mt-", project_Name =
"GO_Activity_Analysis")
```

"`Input_data`": Input matrix including gene names and cell IDs (row names are gene names, and column names are cell IDs);

"`Str_genes`": The same parameter as it in function `Read_Seurat_scData()`; (check "Read from Seurat object")

"`project_Name`": Project name for creating Seurat object.

The function returns a list variable including sub-lists "Original_List" and "Seurat_Object" (check "Read from Seurat object")

Example code:

```
Count_matrix <- read.table("Countdata_Example_for_GOTermActivity.tsv",
                           header = TRUE, row.names = 1, check.names = FALSE)
GOTerm_analysis_3 <- Read_Count_scData(
  Input_data = Count_matrix,
  Str_genes = "^Rp[sl][[:digit:]]"
  )

# Load metadata for the sample
Meta_data <- read.csv("Metadata_Example_for_GOTermActivity.csv", row.names =
1)
GOTerm_analysis_3$Seurat_Object <- Seurat::AddMetaData(
  object = GOTerm_analysis_1$Seurat_Object,
  metadata = Meta_data
  )
```

Check "Read from Seurat object" for the result.

QC (optional & recommended)

Function `QC_scData()` can perform QC process. In the function:

```
QC_scData <- function(List, Count_threshold = 0, Cell_threshold = 0,
Interested_genes_threshold = 1)
```

"`List`": The "GoTermActivity" object that was built after run "read" function (`Read_Seurat_scData`, `Read_CountFile_scData` or `Read_Count_scData`).

"`Count_threshold`": The threshold for total counts. Keep those cells: "`Sum_Count_Log2 > a`"(for single real input a) and "`a < Sum_Count_Log2 < b`" (for pair input `c(a, b)`)

"`Cell_threshold`": The threshold for the number of genes the cell expresses, to remove some cells that have low numbers of expressed genes. Keep those cells: "`nFeature_RNA > Cell_threshold`"

"`Interested_genes_threshold`" The threshold for the percentage of the selected gene set (default: mitochondrial genes). Keep those cells: "`Percent_genes_interested < Interested_genes_threshold`"

The function returns an object including sub-list "`Seurat_Object`" & "`Original_List`" & "`AfterQC_List`".

In the sub-list "`AfterQC_List`", there are 8 variables:

“Size”: Numbers of genes and cells in the data.

“Data”: A count matrix of the data.

“Gene”: Gene list of the data. (The order is consistent with the data matrix)

“Cell_ID”: Cell list of the data. (The order is consistent with the data matrix)

“Expression_percent”: In a single cell, the percentage of all expressed genes in the gene list.

“nCount_RNA_log2”: In a single cell, the total counts of all expressed genes. (log2)

“nFeature_RNA”: In a single cell, the number of genes the cell expresses.

“Percent_genes_interested”: In a single cell, the count percentage of the selected gene set (default: mitochondrial genes).

In the sub-list “Seurat_Object”, the same QC parameters will be used to the updated “Seurat” object.

Example code:

```
GOTerm_analysis <- QC_scData(
  List = GOTerm_analysis,
  Count_threshold = c(17.5, 20.5),
  Cell_threshold = 300,
  Interested_genes_threshold = 0.05
  )
```

Integrate GO term dataset & Mapping current sample

Function `Map_GOSet()` will be used on the step. In the function:

```
Map_GOSet <- function(List)
```

“List”: The "GoTermActivity" object.

The function returns an object including sub-list "Seurat_Object" & "Original_List" & "AfterQC_List" & "GO_Dataset" & "AfterMapping_List".

In the sub-list "GO_Dataset", there are 3 variables:

“GO_Term_list”: A table that records the GO terms' ID & Description. (Built-in dataset of package)

“Gene_list”: A list that saves the whole gene list of the GO term dataset. (Built-in dataset of package)

“Map”: A binary matrix; rows mean GO terms and columns mean genes, showing the mapping relationship between GO terms and genes.

In the sub-list "AfterMapping_List", there are 6 variables:

“Size”: Numbers of genes, cells, and GO terms in intersection of the sample dataset and the GO term dataset.

“Data”: Intersection data of the sample dataset and the GO term dataset.

“Data_Bin”: Binary transformation result of Data.

“Gene”: Intersection gene list of the sample dataset and the GO term dataset.

“GO_Term_Filted”: Intersecting GO term table including ID & Description.

“Map”: Intersecting map matrix.

“Table_overview”: Overview information of GO dataset mapping process.

Example code:

```
GOTerm_analysis <- Map_GOSet(GOTerm_analysis)
```

Visible output:

Overview	Number
GO terms in GO dataset	8270
Genes in GO dataset	42476
Genes in input scRNA-seq sample	23337
Overlap genes in both	18776
Filtered GO terms (Overlap genes > 3 for each GO)	2553

Run CMC model

Null distribution of the sample data will be calculated by CMC model, using function `Run_CMC()`. In the function:

```
Run_CMC <- function(List)
```

“List”: The "GoTermActivity" object.

The function returns an object including sub-list "Seurat_Object" & "Original_List" & "AfterQC_List" & "GO_Dataset" & "AfterMapping_List" & "CMC".

In the sub-list "CMC", there are 3 variables:

"P_out": the probability of every single entry.

"Gene_factor": the "sensitivity" of gene factor.

"Cell_factor": the "sensitivity" of cell factor.

Example code:

```
GOTerm_analysis <- Run_CMC(GOTerm_analysis)
```

Visible output:

```
Run CMC Model...
Number of dimensions:2
18776 X 1784
The type of Y is: type 2
The data has no missing values
Iteration=0;  diff=890
Iteration=1;  diff=258.582
Iteration=2;  diff=13.0615
Iteration=3;  diff=0.592463
Iteration=4;  diff=0.026766
Iteration=5;  diff=0.00120915
Iteration=6;  diff=5.46207e-05
Iteration=7;  diff=2.46861e-06
Iteration=8;  diff=1.12721e-07
Iteration=9;  diff=4.50359e-09
elapsed time: 0.578936s
```

Calculate GO term activity score

Use function GO_Scores(). In the function:

```
GO_Scores <- function(List)
```

"List": The "GoTermActivity" object.

The function returns an object including sub-list "Seurat_Object" & "Original_List" & "AfterQC_List" & "GO_Dataset" & "AfterMapping_List" & "CMC" & "GO_Term_Activity_Scores".

In the sub-list "GO_Term_Activity_Scores", there are 3 variables:

"p_Value": The p-value (GO term activity score) of every cell in different GO terms.

"Z_Score": The Z-score (GO term activity score) of every cell in different GO terms.

"N_Gene": Numbers of genes considered in every GO term.

Example code:

```
GOTerm_analysis <- GO_Scores(GOTerm_analysis)
```

Feature selection

Use function `GO_Selected_Order_Statistics()` and calculate the significant level of each GO term for the sample. Sort and select the significant GO terms. These will be used in downstream analysis, like PCA. In the function:

```
GO_Selected_Order_Statistics <- function(List)
```

"`List`": The "GoTermActivity" object.

The function returns an object including sub-list "`Seurat_Object`" & "`Original_List`" & "`AfterQC_List`" & "`GO_Dataset`" & "`AfterMapping_List`" & "`CMC`" & "`GO_Term_Activity_Scores`" & "`Feature_Selection`".

In the sub-list "`Feature_Selection`", there are 4 variables:

"`Adjusted_p_Value`": Adjusted p Value of GO terms; the smaller the value, the more significant the corresponding GO term.

"`Adjusted_p_Value_sorted`": Sorted adjusted p Value of GO terms (Ascending order).

"`Index_Sort`": Index of sorted GO terms in original List.

"`GO_Term_sorted`": Sorted GO term list.

"`Table_overview`": Table for the number of significant GO terms in different thresholds.

Example code:

```
GOTerm_analysis <- GO_Selected_Order_Statistics(GOTerm_analysis)
```

Visible output:

Significance_level_threshold	Top_GO_terms_number
p-value(adjusted) < 0.01	210
p-value(adjusted) < 0.05	226
p-value(adjusted) < 0.10	228

Run PCA

Use function `Run_PCA()` on the GO term activity scores of each cell. Only the previously determined sorted significant GO terms are used (Check "Feature selection"). In the function:

```
Run_PCA <- function(List, Score_type = "p-value", Selected_GOs = -1,
                    Show_npcs = 50, Name_pca_reduction = "pca_GO_analysis")
```

“List”: The "GoTermActivity" object.

“Score_type”: "p-value" or "z-score" or "-log10p"; Default value is “p-value”;

“Selected_GOs”: The number of selected top significant GO terms (Integer > 0). Default value is -1; if (-1), select the GO terms that have: $Feature_Selection$Adjusted_p_Value_sorted < 0.01.

“Show_npcs”: The number of PCAs shown in Variance plot (Integer > 0).

“Name_pca_reduction”: Name of pca reduction for Seurat object, stored in "$Seurat_Object@reductions". ("pca_GO_analysis" by default)

The function returns an object including sub-list "Seurat_Object" & "Original_List" & "AfterQC_List" & "GO_Dataset" & "AfterMapping_List" & "CMC" & "GO_Term_Activity_Scores" & "Feature_Selection"& "PCA".

In the sub-list "PCA", there are 4 variables:

“PCA_matrix”: PCA matrix.

“Variance”: Variance of each PC.

“Cumulate_Percent”: Sum of variance percentages in top PCs.

“Score_Type”: Input parameter "Score_type".

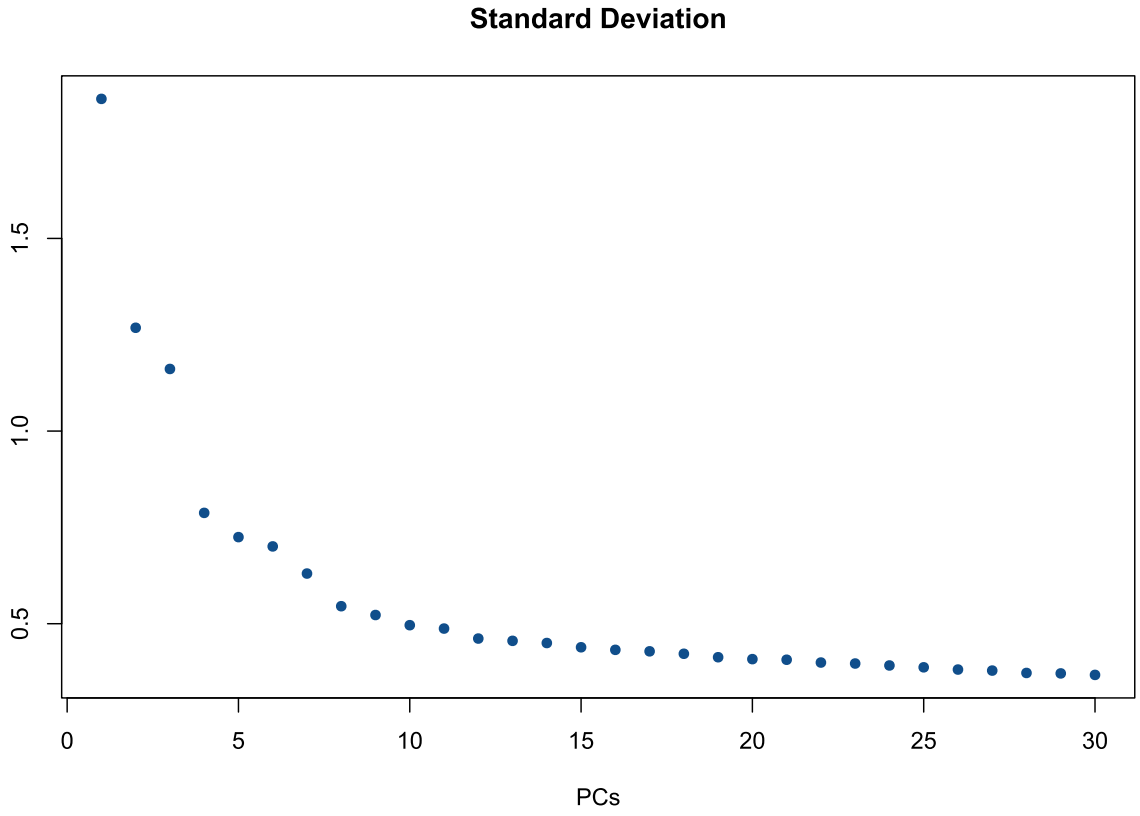
In the sub-list "Feature_Selection", "Selected_GOs" will be added in the sub-list.\cr

In the sub-list "Seurat_Object", the PCA results will be added in the “[…]@reduction” Name_pca_reduction ("pca_GO_analysis" by default)

Example code:

```
GOTerm_analysis <- Run_PCA(
  List = GOTerm_analysis,
  Score_type = "p-value",
  Selected_GOs = 228,
  Show_npcs = 30,
  Name_pca_reduction = "pca_GO_analysis"
  )
```

Visible output (the standard deviation of each PC):



Cluster the cells

Use clustering-related functions of “Seurat” package. The PCA results of Go term analysis have been saved to the “`@reduction`” of "`$Seurat_Object`", and this reduction can be used directly for clustering. (Check the usage of “Seurat” package)

Example code:

```
nPCs <- 1:15
GOTerm_analysis$Seurat_Object <- FindNeighbors(
  object = GOTerm_analysis$Seurat_Object,
  dims = nPCs,
  reduction = "pca_GO_analysis",
  graph.name = "GO_analysis"
  )
GOTerm_analysis$Seurat_Object <- FindClusters(
  object = GOTerm_analysis$Seurat_Object,
  resolution = 0.75,
  graph.name = "GO_analysis",
  cluster.name = "Clusters_GOTerm"
  )
```

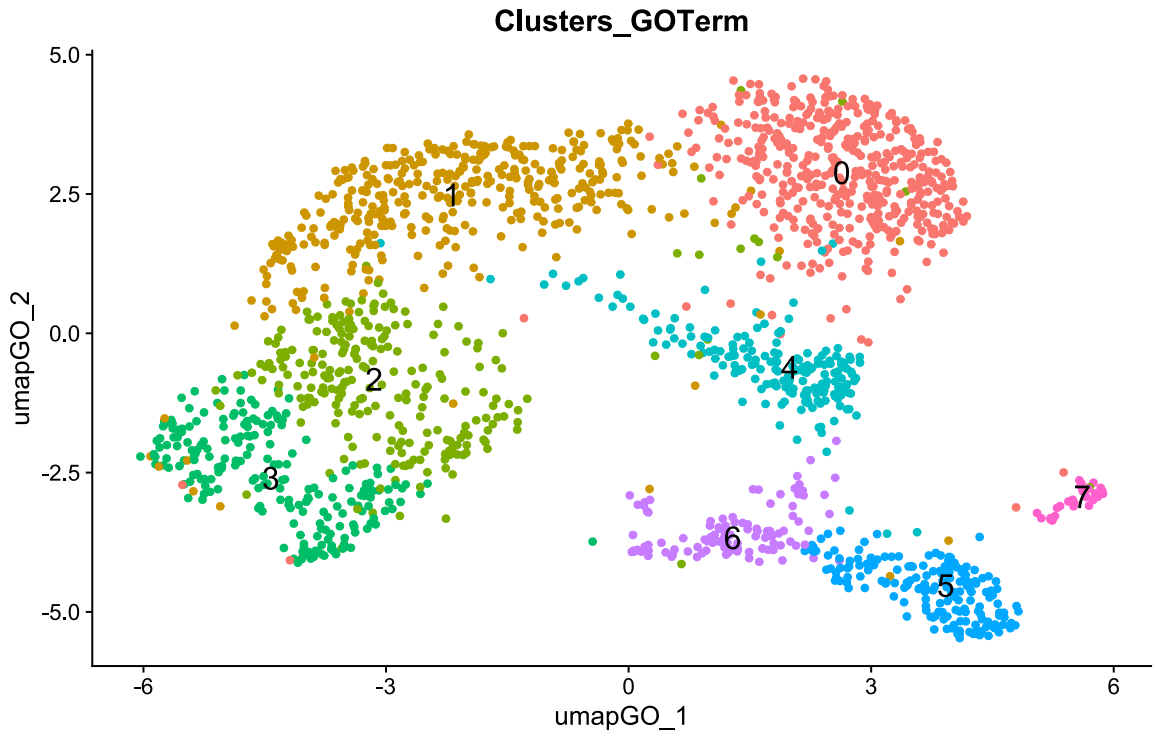
Run non-linear dimensional reduction (UMAP/tSNE)

Use clustering-related functions of “Seurat” package. The PCA results of Go term analysis have been saved to the “`@reduction`” of "`$Seurat_Object`", and this reduction can be used directly for UMAP/tSNE. (Check the usage of “Seurat” package)

Example code (UMAP):

```
nPCs <- 1:15
GOTerm_analysis$Seurat_Object <- RunUMAP(
  object = GOTerm_analysis$Seurat_Object,
  dims = nPCs,
  reduction = "pca_GO_analysis",
  reduction.name = "umap_GO"
  )
DimPlot(
  object = GOTerm_analysis$Seurat_Object,
  reduction = "umap_GO",
  pt.size = 1.5,
  group.by = "Clusters_GOTerm",
  label = TRUE,
  label.size = 6
  ) + NoLegend()
```

Visible output:

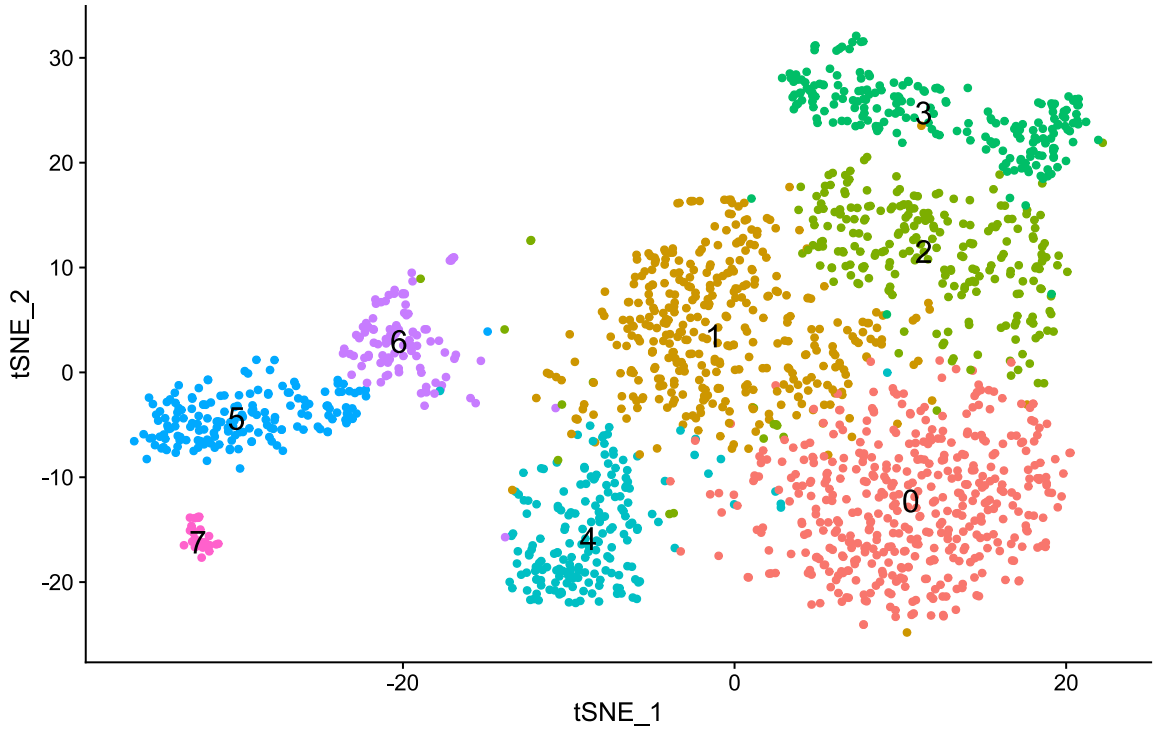


Example code (tSNE):

```
nPCs <- 1:15
```

```
GOTerm_analysis$Seurat_Object <- RunTSNE(
  object = GOTerm_analysis$Seurat_Object,
  dims = nPCs,
  reduction = "pca_GO_analysis",
  reduction.name = "tsne_GO"
  )
DimPlot(
  object = GOTerm_analysis$Seurat_Object,
  reduction = "tsne_GO",
  pt.size = 1.5,
  label = TRUE,
  label.size = 6
) + NoLegend()
```

Visible output:



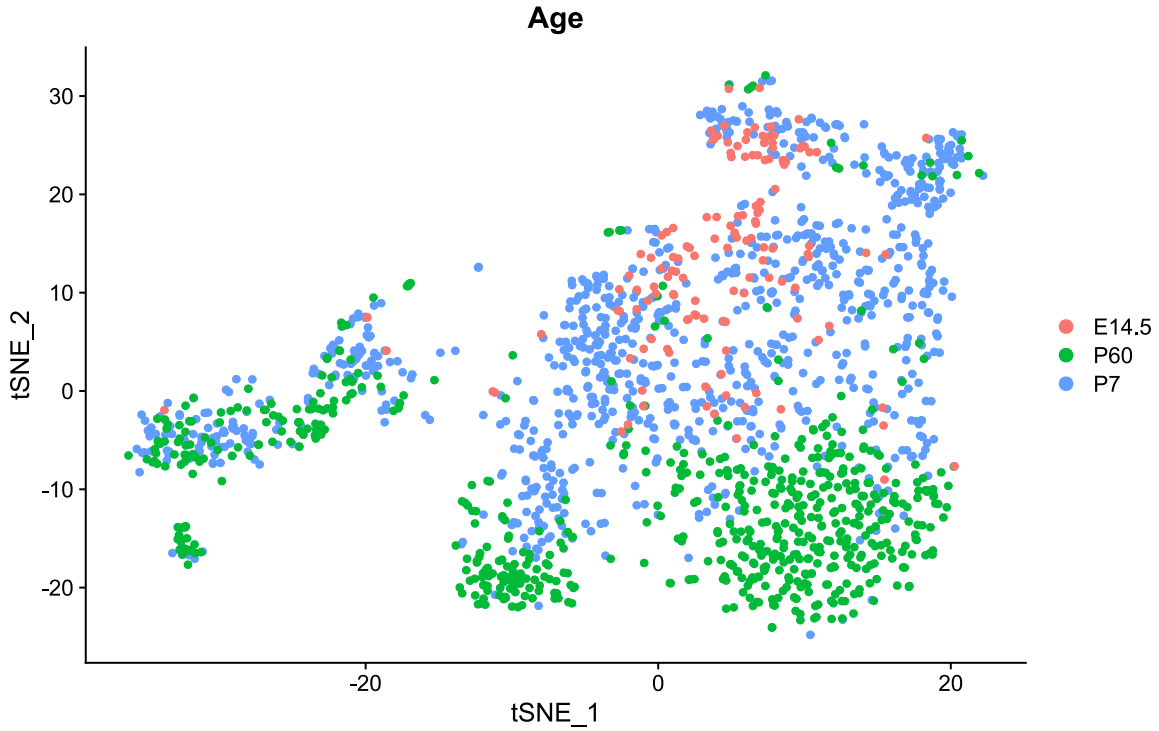
For the new UMAP/tSNE reduction, we can also use these when check other metadata of the sample, such as “Age” in metadata of “Seurat” object.

Example code (tSNE as an example):

```
DimPlot(
  object = GOTerm_analysis$Seurat_Object,
  reduction = "tsne_GO",
  pt.size = 0.8,
  group.by = "Age"
```

```
)
```

Visible output:



Extract cell name vector of target group (optional)

Use function `Cell_Names()`. In the function:

```
Cell_Names <- function(List, Ident_mark, Group_by, If_not = FALSE)
```

"`List`": The "GoTermActivity" object.

"`Score_type`": "`p-value`" or "`z-score`" or "`-log10p`"; Default value is "`p-value`";

"`Selected_GOs`": The number of selected top significant GO terms (Integer > 0). Default value is `-1`; if (`-1`), select the GO terms that have: `$Feature_Selection$Adjusted_p_Value_sorted < 0.01`.

"`Ident_mark`": A vector including factor name you are interested in metadata column `'Group_by'`.

"`Group_by`": Name of one metadata column to group (color) cells by (for example, "`seurat_clusters`") (should be factor variable with limited discrete levels).

"`If_not`": Whether to choose the complement of `'Ident_mark'` in metadata column `'Group_by'`. (for example: if `'Ident_mark' = c('1', '2')`, and `If_not = TRUE`; The output will be the names of all cells except cluster '1' and '2')

Example code (select cells in cluster "5", "6", and "7"):

```
# Cells of "5", "6", "7"
Ident_mark_1 <- c("5", "6", "7")
Cell_Group1 <- Cell_Names(
  List = GOTerm_analysis,
  Ident_mark = Ident_mark_1,
  Group_by = "Clusters_GOTerm",
  If_not = FALSE
  )
```

Example code (select cells not cluster "5", "6", or "7"):

```
# All other cells except "5", "6", "7"
Ident_mark_2 <- c("5", "6", "7")
Cell_Group2 <- Cell_Names(
  List = GOTerm_analysis,
  Ident_mark = Ident_mark_2,
  Group_by = "Clusters_GOTerm",
  If_not = TRUE
  )
```

Finding differentially activity GO terms

Use function `Differential_Activity_Detection()`. By default, it finds both positive and negative GO terms between two cell groups. In the function:

```
Differential_Activity_Detection <- function(List, Cell_group1, Cell_group2,
Min_pct_pos = 0.1, Min_avg_Diff = 0.05, Thre_p_value = 0.01, Only_pos =
FALSE)
```

"`List`": The "GoTermActivity" object.

"`Cell_group1`": Name vector of cells in group1. (Generated by `Cell_Names()` or other method by user)

"`Cell_group2`": Name vector of cells in group2. (Generated by `Cell_Names()` or other method by user)

"`Min_pct_pos`": Threshold, Lower bound limit for the percent of positive activity scores (z score) (`Group1 or Group2 > Min_pct_pos`)(The default value is 0.25).

"`Min_avg_Diff`": Threshold, Lower bound limit for the difference of average activity scores (z score) abs(Group1 - Group2)(The default value is 0.1).

"`Thre_p_value`": Threshold, Upper bound limit for significant level (`p-value`)(The default value is 0.01).

"`Only_pos`": Only return positive GO terms for group1 Cell (`FALSE` by default).

The function returns a differential activity GO term table for Cell_group1 (Positive/Negative).

Example code (use previous "`Cell_Group1`" and "`Cell_Group2`"):

```
DA_GO_567 <- Differential_Activity_Detection(
  List = GOTerm_analysis,
  Cell_group1 = Cell_Group1,
  Cell_group2 = Cell_Group2,
  Min_pct_pos = 0.25,
  Min_avg_Diff = 0.25,
  Thre_p_value = 0.01,
  Only_pos = FALSE
  )
print(DA_GO_6_7[1:10, ])
```

Visible output:

```
Top 10 differential activity GO terms:
```

	z-score	pct_1(z>0)	pct_2(z>0)	Avd_Diff_z	Type
GO:0009253	25.05416	0.4642857	0.0142276	8.120941	Positive
GO:0032827	25.05416	0.4642857	0.0142276	8.120941	Positive
GO:0034341	22.57497	0.9512987	0.2479675	4.833122	Positive
GO:0010700	-22.30000	0.0194805	0.3279133	-11.244009	Negative
GO:0006958	-21.81481	0.1266234	0.6842818	-6.958451	Negative
GO:0007195	21.20118	0.7694805	0.1781843	9.318173	Positive
GO:0018149	21.05146	0.7629870	0.1998645	9.576777	Positive
GO:0003081	21.04730	0.3506494	0.0128726	6.113718	Positive
GO:0002068	20.96980	0.6071429	0.1077236	8.715199	Positive
GO:0030168	-20.46061	0.0909091	0.6097561	-2.505244	Negative

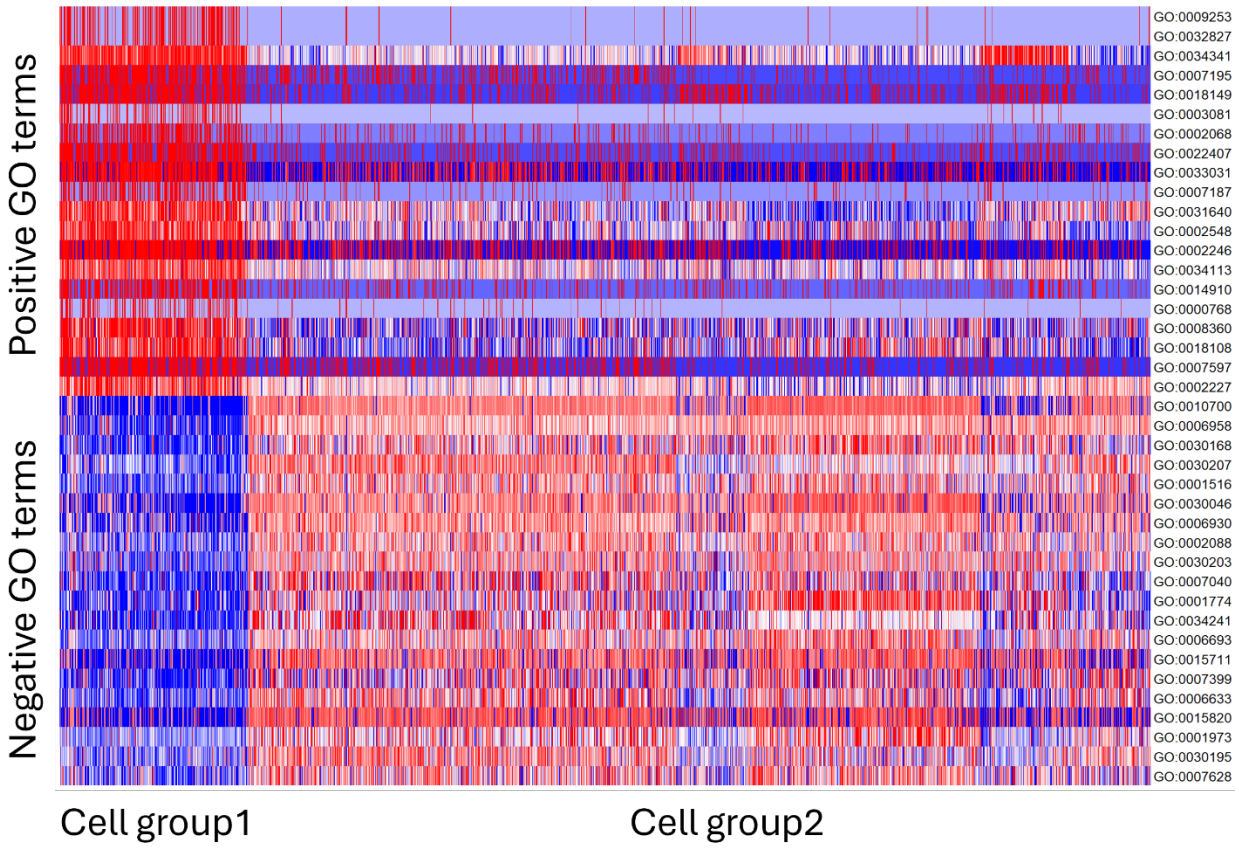
To check the validity of differential activity GO terms detection, we can draw a heatmap of activity scores (z-score) on top 20 positive and 20 negative GO terms. The example code is as follows:

Example code:

```
# Sort the cell name
Cell_sorted <-c(Cell_Group1, Cell_Group2)

# select GO terms
GO_pos <- DA_GO_567[DA_GO_567$Type == "Positive", ][1:20,]
GO_neg <- DA_GO_567[DA_GO_567$Type == "Negative", ][1:20,]
GO_term_heatmap <- c(rownames(GO_pos), rownames(GO_neg))

# extract score (z-score) matrix
Data_to_H <- GOTerm_analysis$GO_Term_Activity_Scores$Z_Score[GO_term_heatmap,
Cell_sorted]
```

```
# modify and scale the score matrix
Min_scores <- min(Data_to_H[is.finite(Data_to_H)])
Max_scores <- max(Data_to_H[is.finite(Data_to_H)])
Data_to_H[Data_to_H == -Inf] <- min(2 * Min_scores, 0)
Data_to_H[Data_to_H == Inf] <- max(2 * Max_scores, 0)
Data_to_H <- t(scale(t(Data_to_H)))
hist(Data_to_H, breaks = 100)
Data_to_H[Data_to_H < -1] <- -1
Data_to_H[Data_to_H > 1] <- 1

# draw heatmap
library(pheatmap)
col <- colorRampPalette(c("blue","white","red"))(256)
pheatmap(Data_to_H, scale = "none",
         treeheight_col=0, threeheright_row=0,
         display_numbers = F, color = col,
         show_colnames = F, cluster_rows = FALSE,
         cluster_cols = FALSE, fontsize_row = 7)
```

Visible output:



Visualization of GO term activity score

Use function `GOPlot()`. In the function:

```
GOPlot <- function(List, GO_ID, reduction_name, Score_type = "z-score",
point_size =0.8, Bar_colors ='Default', Boundary_constraint = c(0.025,0.975))
```

“List”: The "GoTermActivity" object.

“GO_ID”: GO ID (In List$AfterMapping_List$GO_Term_Filted$ID) for target GO Term (For example: GO_ID = 'GO:0001822').

“reduction_name”: Reduction name (results of UMAP or TSNE) in 'List$Seurat_Object@reductions'.

“Score_type”: "z-score" or "p-value" (show -log10p in "p-value"); Default by "z-score".

“point_size”: Adjust point size for plotting.

“Bar_colors”: The two or more than two colors to form the gradient over. Provide as string vector with the first color corresponding to low values, the last to high. Default by a ‘jet’ color vector.

“Boundary_constraint”: The percentile constraint for extreme values in visualization plot. Default by c(0.025,0.975). The first value is lower limit for data to show, and the second value is upper limit for the data to show. For example, for the default value, data that larger than upper percentile value will be modified into upper percentile value (just for plot function), and data that less than lower percentile value be modified into lower percentile value.

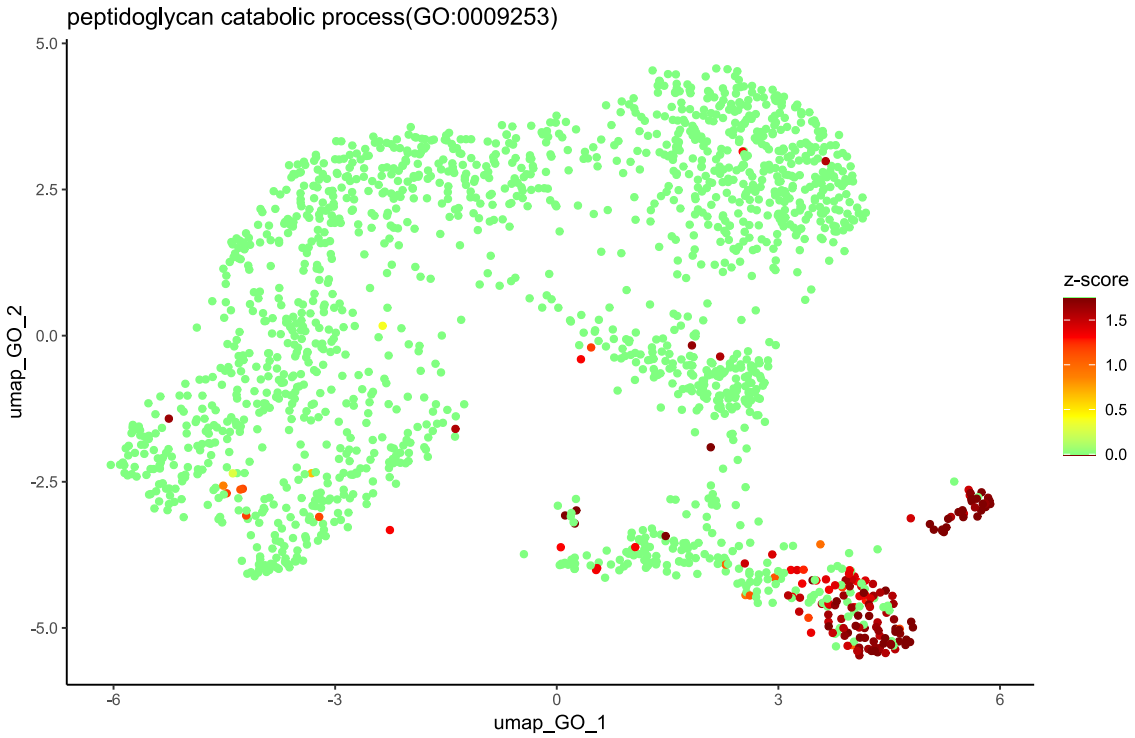
The function returns a ggplot figure.

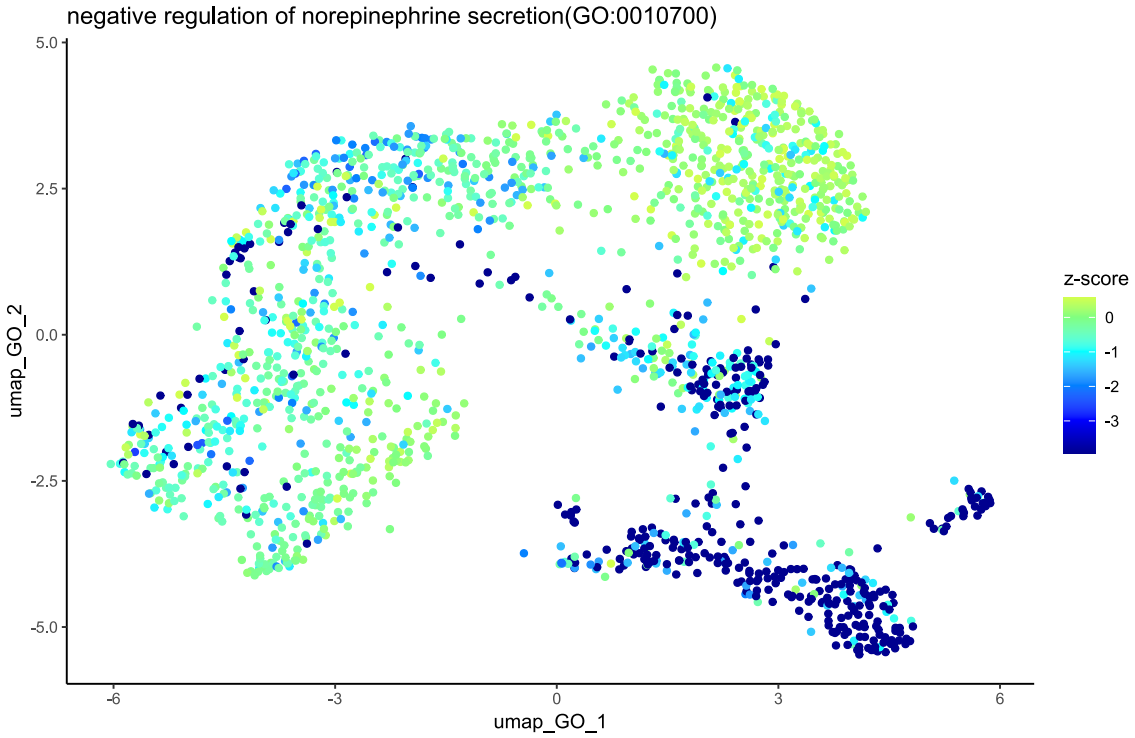
Example code:

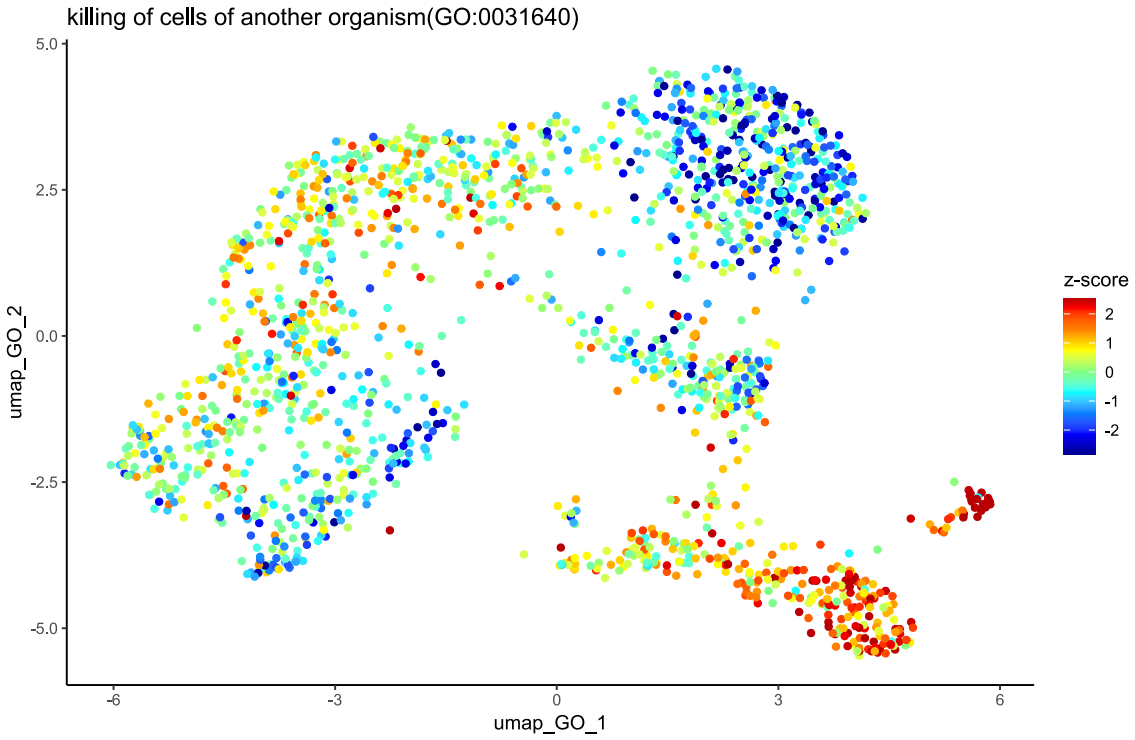
```
GO_ID <- 'GO:0009253'
GOPlot(
  List = GOTerm_analysis,
  GO_ID = GO_ID,
  reduction_name = "umap_GO",
  point_size = 1.5,
  Score_type = "z-score",
  Boundary_constraint = c(0.025,0.975)
  )

GO_ID <- 'GO:0010700'
GOPlot(
  List = GOTerm_analysis,
  GO_ID = GO_ID,
  reduction_name = "umap_GO",
  point_size = 1.5,
  Score_type = "z-score",
  Boundary_constraint = c(0.025,0.975)
)

GO_ID <- 'GO:0031640'
GOPlot(
  List = GOTerm_analysis,
  GO_ID = GO_ID,
  reduction_name = "umap_GO",
  point_size = 1.5,
```

```
  Score_type = "z-score",
  Boundary_constraint = c(0.025,0.975)
)
```

Visible output:



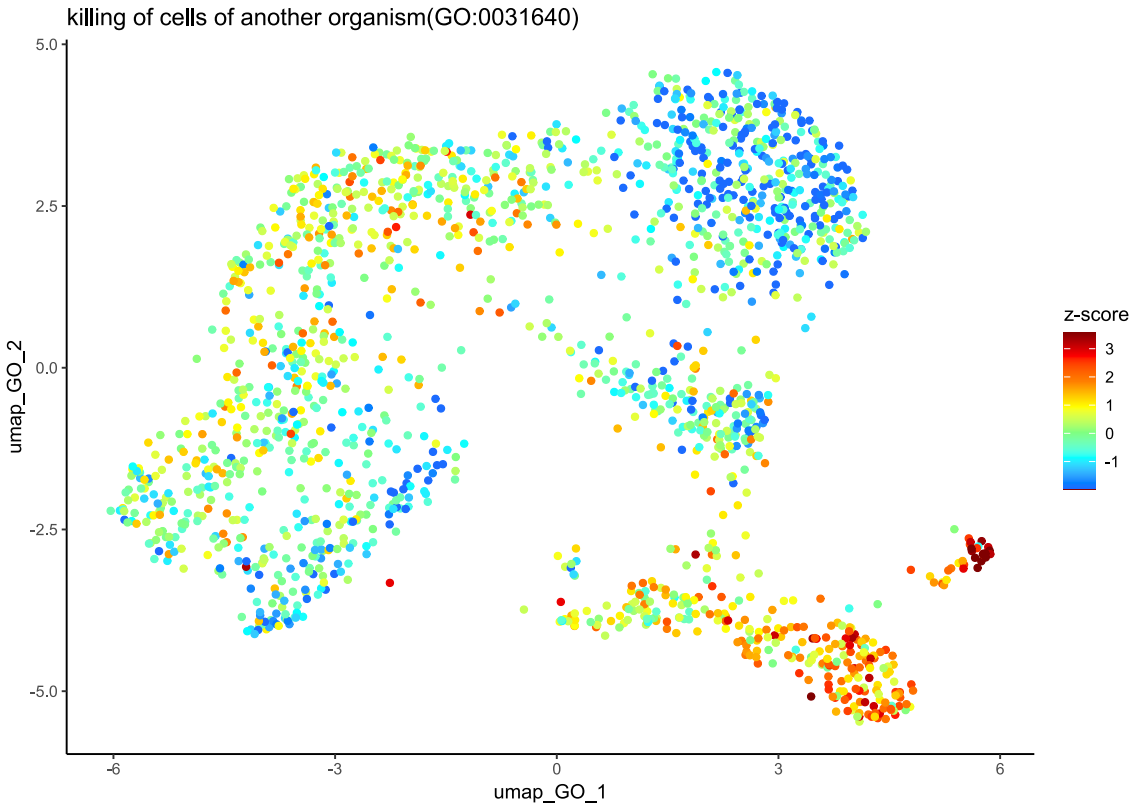




"`Boundary_constraint`" can set the lower and upper percentile limit in the plot. This is a example when change it from default values `c(0.025,0.975)` to `c(0.1,0.995)`.

Example code (GO:0031640):

```
GO_ID <- 'GO:0031640'
GOPlot(
  List = GOTerm_analysis,
  GO_ID = GO_ID,
  reduction_name = "umap_GO",
  point_size = 1.5,
  Score_type = "z-score",
  Boundary_constraint = c(0.1,0.995)
)
```



Extract subset of current “GOTermActivity” object (optional)

Use function `Subset_GoTermActivity()`. In the function:

```
Subset_GoTermActivity <- function(List, sub_Cells)
```

“`List`”: The "GoTermActivity" object.

“`sub_Cells`”: A vector of cell names to keep in subset. (Generated by `Cell_Names()` or other method by user)

The function returns the subset "GoTermActivity" object. The sub-list "`Seurat_Object`" will also be updated in the process.

Example code:

```
GOTerm_analysis_sub <- Subset_GoTermActivity(
  List = GOTerm_analysis,
  sub_Cells = Cell_Group2
  )
```