

---

# **gseapy Documentation**

***Release 1.1.1***

**Zhuoqing Fang**

**Dec 21, 2023**



## TABLE OF CONTENTS

<b>1</b>	<b>GSEAPY: Gene Set Enrichment Analysis in Python.</b>	<b>1</b>
<b>2</b>	<b>Citation</b>	<b>3</b>
<b>3</b>	<b>Installation</b>	<b>5</b>
<b>4</b>	<b>GSEAPy is a Python/Rust implementation of GSEA and wrapper for Enrichr.</b>	<b>7</b>
<b>5</b>	<b>Why GSEAPY</b>	<b>9</b>
<b>6</b>	<b>Indices and tables</b>	<b>79</b>
	<b>Python Module Index</b>	<b>81</b>
	<b>Index</b>	<b>83</b>



## GSEAPY: GENE SET ENRICHMENT ANALYSIS IN PYTHON.

**Release notes :** <https://github.com/zqfang/GSEAPy/releases>



## CITATION

Zhuoqing Fang, Xinyuan Liu, Gary Peltz, GSEAPy: a comprehensive package **for** performing **gene set** enrichment analysis **in** Python, Bioinformatics, 2022;, btac757, <https://doi.org/10.1093/bioinformatics/btac757>





## INSTALLATION

Install gseapy package from bioconda or pypi.

```
# if you have conda (MacOS_x86-64 and Linux only)  
$ conda install -c bioconda gseapy  
  
# or use pip to install the latest release  
$ pip install gseapy
```



## GSEAPY IS A PYTHON/RUST IMPLEMENTATION OF GSEA AND WRAPPER FOR ENRICHR.

GSEAPy has multiple subcommands: `gsea`, `prerank`, `ssgsea`, `gsva`, `replot` `enrichr`, `biomart`.

1. The `gsea` module produces **GSEA** results. The input requires a txt file(FPKM, Expected Counts, TPM, et.al), a `cls` file, and `gene_sets` file in gmt format.
2. The `prerank` module produces **Prerank tool** results. The input expects a pre-ranked gene list dataset with correlation values, which in `.rnk` format, and `gene_sets` file in gmt format. `prerank` module is an API to *GSEA* pre-rank tools.
3. The `ssgsea` module performs **single sample GSEA(ssGSEA)** analysis. The input expects a gene list with expression values(same with `.rnk` file, and `gene_sets` file in gmt format. ssGSEA enrichment score for the gene set as described by [D. Barbie et al 2009](#).
4. The `gsva` module performs **GSVA** analysis, which described by [Hänzelmann et al](#).
5. The `replot` module reproduces GSEA desktop version results. The only input for GSEAPY is the location to GSEA Desktop output results.
6. The `enrichr` module enables you to perform gene set enrichment analysis using **Enrichr** API. Enrichr is open source and freely available online at: <http://amp.pharm.mssm.edu/Enrichr> . It runs very fast and generates results in txt format.
7. The `biomart` module helps you convert gene ids using BioMart API.

GSEAPy could be used for **RNA-seq, ChIP-seq, Microarray** data. It's used for convenient GO enrichments and produce **publishable quality figures** in python.

The full *GSEA* is far too extensive to describe here; see [GSEA](#) documentation for more information. All files' formats for GSEAPy are identical to GSEA desktop version.



## WHY GSEAPY

I would like to use Pandas to explore my data, but I did not find a convenient tool to do gene set enrichment analysis in python. So, here are my reasons:

- **Ability to run inside python interactive console without having to switch to R!!!**
- User friendly for both wet and dry lab users.
- Produce or reproduce publishable figures.
- Perform batch jobs easy.
- Easy to use in bash shell or your data analysis workflow, e.g. snakemake.

### 5.1 Welcome to GSEAPY's documentation!

#### 5.1.1 GSEAPY: Gene Set Enrichment Analysis in Python.

#### 5.1.2 GSEAPy is a Python/Rust implementation of GSEA and wrapper for Enrichr.

It's used for convenient GO enrichments and produce **publication-quality figures** from python.

GSEAPy could be used for **RNA-seq, ChIP-seq, Microarray** data.

**Gene Set Enrichment Analysis** (GSEA) is a computational method that determines whether an a priori defined set of genes shows statistically significant, concordant differences between two biological states (e.g. phenotypes).

The full GSEA is far too extensive to describe here; see [GSEA](#) documentation for more information.

Enrichr is open source and freely available online at: <http://amp.pharm.mssm.edu/Enrichr> .

### 5.1.3 Citation

Zhuoqing Fang, Xinyuan Liu, Gary Peltz, GSEAPy: a comprehensive package **for** performing **↪**gene **set** enrichment analysis **in** Python, Bioinformatics, 2022;, btac757, <https://doi.org/10.1093/bioinformatics/btac757>

### 5.1.4 Installation

Install gseapy package from bioconda or pypi.

```
# if you have conda (MacOS_x86-64 and Linux only)
$ conda install -c bioconda gseapy

# or use pip to install the latest release
$ pip install gseapy
```

### 5.1.5 GSEA Java version output:

This is an example of GSEA desktop application output

### 5.1.6 GSEAPy Prerank module output

Using the same data from GSEA, GSEAPy reproduces the example above.

Using Prerank or replot module will reproduce the same figure for GSEA Java desktop outputs

### 5.1.7 GSEAPy enrichr module

A graphical introduction of Enrichr

**The only thing you need to prepare is a gene list file in txt format(one gene id per row), or a python list object.**

**Note:** Enrichr uses a list of Entrez gene symbols as input. You should convert all gene names to uppercase.

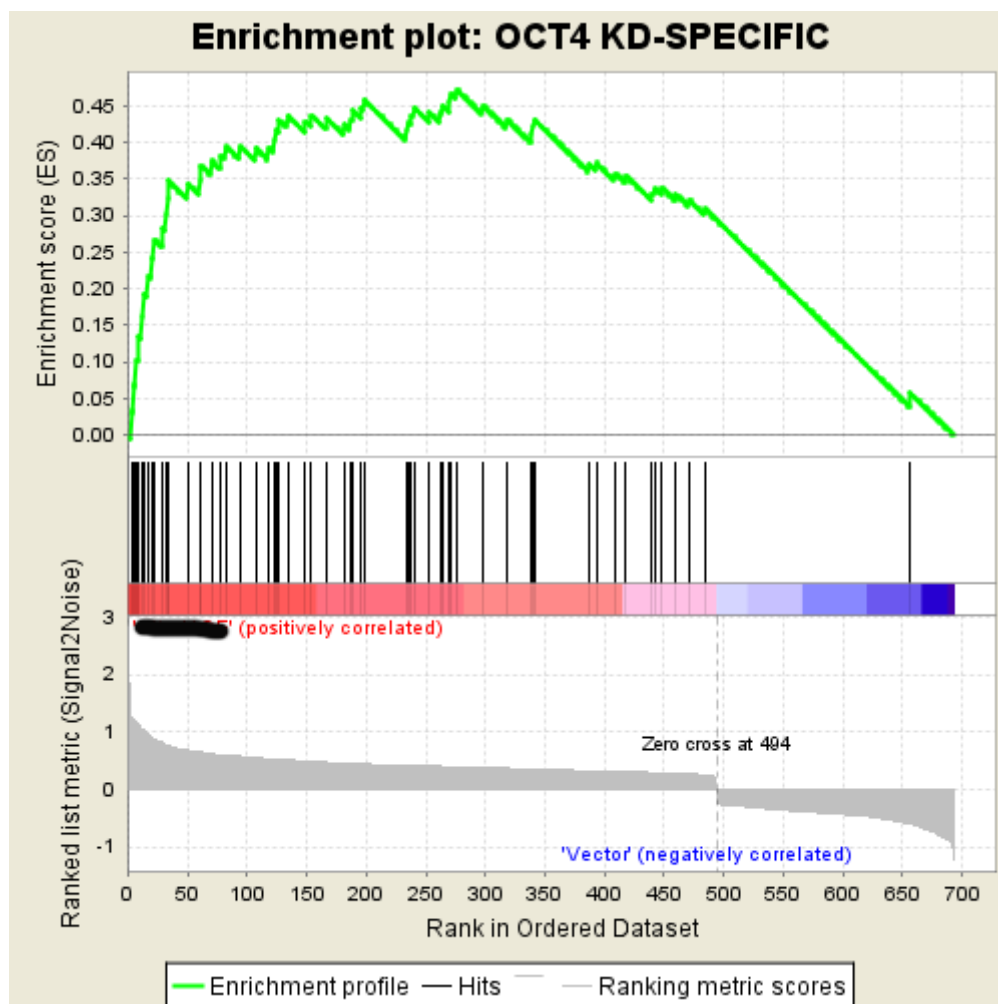
For example, both a list object and txt file are supported for enrichr API

```
# if you prefer to run gseapy.enrichr() inside python console, you could assign a list_
↪object to
# gseapy like this.
gene_list = ['SCARA3', 'LOC100044683', 'CMBL', 'CLIC6', 'IL13RA1', 'TACSTD2', 'DKKL1',
             'CSF1', 'CITED1', 'SYNP02L']
```

```
# an alternative way is that you could provide a gene list txt file which looks like_
↪this:
with open('data/gene_list.txt') as genes:
    print(genes.read())
```

```
CTLA2B
SCARA3
```

(continues on next page)



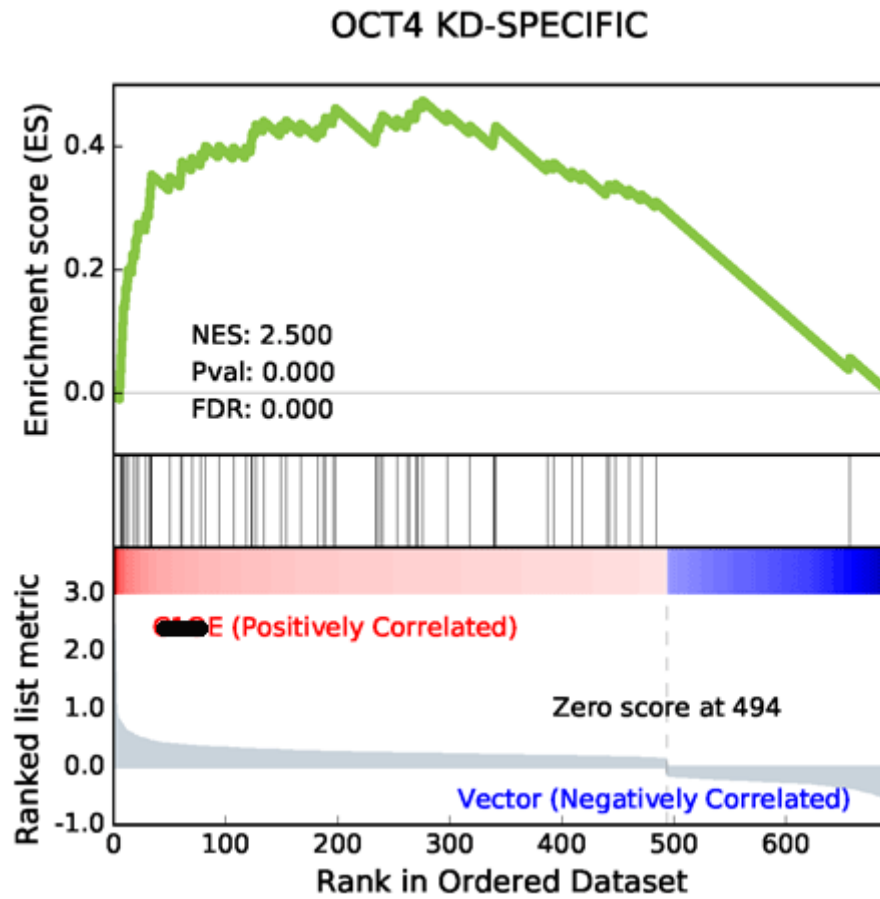
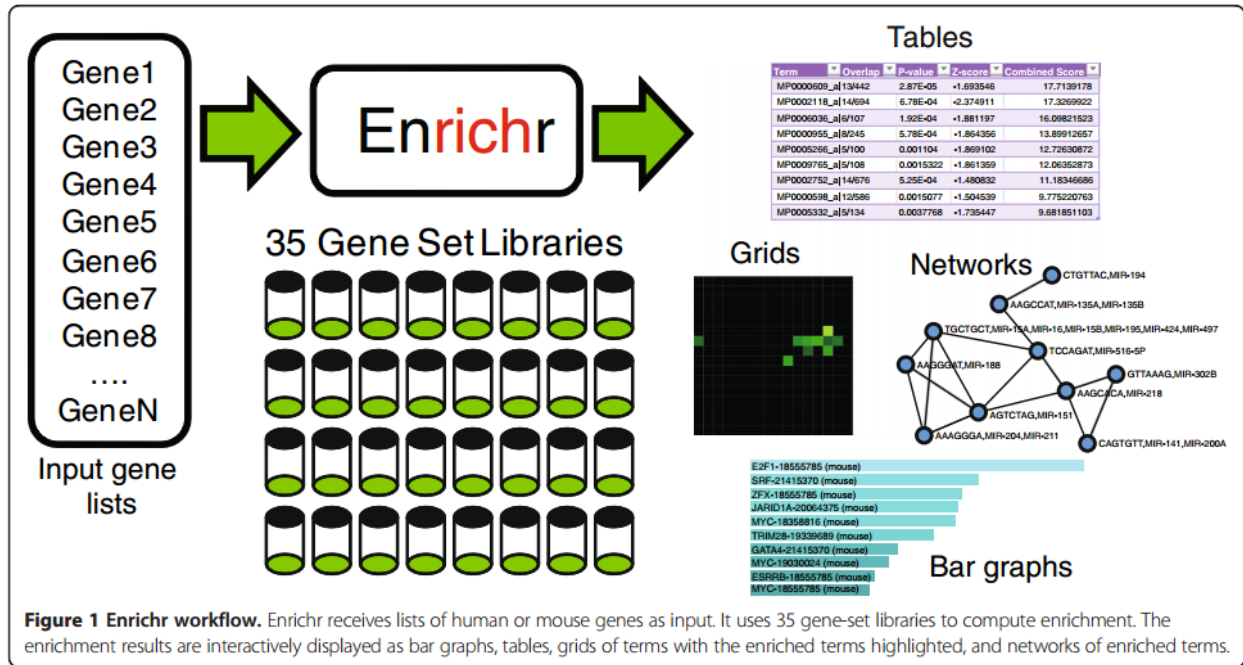


Fig. 1: Generated by GSEAPY  
GSEAPy figures are supported by all matplotlib figure formats.  
You can modify GSEA plots easily in .pdf files. Please Enjoy.





(continued from previous page)

```

LOC100044683
CMBL
CLIC6
IL13RA1
TACSTD2
DKKL1
CSF1
CITED1
SYNP02L
TINAGL1
PTX3

```

## 5.1.8 Installation

Install gseapy package from bioconda or pypi.

```

# if you have conda
$ conda install -c conda-forge -c bioconda gseapy

# or use pip to install the latest release
$ pip install gseapy

```

For API information to use this library, see the [Developmental Guide](#).

## 5.2 GSEAPY Example

Examples to use GSEAPY inside python console

```
[1]: # %matplotlib inline
# %config InlineBackend.figure_format='retina' # mac
%load_ext autoreload
%autoreload 2
import pandas as pd
import gseapy as gp
import matplotlib.pyplot as plt
```

Check gseapy version

```
[2]: gp.__version__
[2]: '1.1.0'
```

### 5.2.1 Biomart API

Don't use this if you don't know Biomart

Warning: This API has limited support now

Convert gene identifiers

```
[3]: from gseapy import Biomart
bm = Biomart()
```

```
[4]: ## view validated marts
# marts = bm.get_marts()
## view validated dataset
# datasets = bm.get_datasets(mart='ENSEMBL_MART_ENSEMBL')
## view validated attributes
# attrs = bm.get_attributes(dataset='hsapiens_gene_ensembl')
## view validated filters
# filters = bm.get_filters(dataset='hsapiens_gene_ensembl')
## query results
queries = {'ensembl_gene_id': ['ENSG00000125285', 'ENSG00000182968']} # need to be a dict_
↪ object
results = bm.query(dataset='hsapiens_gene_ensembl',
                  attributes=['ensembl_gene_id', 'external_gene_name', 'entrezgene_id',
↪ 'go_id'],
                  filters=queries)
results.tail()
```

```
[4]:   ensembl_gene_id  external_gene_name  entrezgene_id   go_id
36  ENSG00000182968                SOX1             6656  GO:0021884
37  ENSG00000182968                SOX1             6656  GO:0030900
38  ENSG00000182968                SOX1             6656  GO:0048713
39  ENSG00000182968                SOX1             6656  GO:1904936
40  ENSG00000182968                SOX1             6656  GO:1990830
```

```
[5]: results.dtypes
[5]: ensembl_gene_id      object
     external_gene_name  object
     entrezgene_id       Int32
     go_id                object
     dtype: object
```

## Mouse gene symbols maps to Human, or Vice Versa

This is useful when you have troubles to convert gene symbols between human and mouse

```
[6]: from gseapy import Biomart
     bm = Biomart()
     # note the dataset and attribute names are different
     m2h = bm.query(dataset='mmusculus_gene_ensembl',
                    attributes=['ensembl_gene_id', 'external_gene_name',
                               'hsapiens_homolog_ensembl_gene',
                               'hsapiens_homolog_associated_gene_name'])

     h2m = bm.query(dataset='hsapiens_gene_ensembl',
                    attributes=['ensembl_gene_id', 'external_gene_name',
                               'mmusculus_homolog_ensembl_gene',
                               'mmusculus_homolog_associated_gene_name'])
```

```
[7]: # h2m.sample(10)
```

## Gene Symbols Conversion for the GMT file

This is useful when runing GSEA for non-human species

**e.g. Convert Human gene symbols to Mouse.**

```
[8]: # get a dict symbol mappings
     h2m_dict = {}
     for i, row in h2m.loc[:, ["external_gene_name", "mmusculus_homolog_associated_gene_name"]].iterrows():
         if row.isna().any(): continue
         h2m_dict[row['external_gene_name']] = row["mmusculus_homolog_associated_gene_name"]
     # read gmt file into dict
     kegg = gp.read_gmt(path="tests/extdata/enrichr.KEGG_2016.gmt")
     print(kegg['MAPK signaling pathway Homo sapiens hsa04010'][:10])

     ['EGF', 'IL1R1', 'IL1R2', 'HSPA1L', 'CACNA2D2', 'CACNA2D1', 'CACNA2D4', 'CACNA2D3',
      ↪ 'MAPK8IP3', 'MAPK8IP1']
```

```
[9]: kegg_mouse = {}
     for term, genes in kegg.items():
         new_genes = []
         for gene in genes:
             if gene in h2m_dict:
```

(continues on next page)

(continued from previous page)

```

        new_genes.append(h2m_dict[gene])
    kegg_mouse[term] = new_genes
print(kegg_mouse['MAPK signaling pathway Homo sapiens hsa04010'][:10])

['Egf', 'Il1r1', 'Il1r2', 'Hspa1l', 'Cacna2d2', 'Cacna2d1', 'Cacna2d4', 'Cacna2d3',
↪ 'Mapk8ip3', 'Mapk8ip1']

```

## 5.2.2 Msigdb API

Down load gmt file from: <https://data.broadinstitute.org/gsea-msigdb/msigdb/release/>

```
[10]: from gseapy import Msigdb
```

```
[11]: msig = Msigdb()
      # mouse hallmark gene sets
      gmt = msig.get_gmt(category='mh.all', dbver="2023.1.Mm")
```

two helper method

```

# list msigdb version you wanna query
msig.list_dbver()
# list categories given dbver.
msig.list_category(dbver="2023.1.Hs") # mouse

```

```
[12]: print(gmt['HALLMARK_WNT_BETA_CATENIN_SIGNALING'])

['Ctnnb1', 'Jag1', 'Myc', 'Notch1', 'Ptch1', 'Trp53', 'Axin1', 'Ncstn', 'Rbpj', 'Psen2',
↪ 'Wnt1', 'Axin2', 'Hey2', 'Fzd1', 'Frat1', 'Csnk1e', 'Dvl2', 'Hey1', 'Gnai1', 'Lef1',
↪ 'Notch4', 'Ppard', 'Adam17', 'Tcf7', 'Numb', 'Ccnd2', 'Ncor2', 'Kat2a', 'Nkd1', 'Hdac2',
↪ 'Dkk1', 'Wnt5b', 'Wnt6', 'Dl11', 'Skp2', 'Hdac5', 'Fzd8', 'Dkk4', 'Cul1', 'Jag2',
↪ 'Hdac11', 'Mam11']

```

## 5.2.3 Enrichr API

See all supported enrichr library names

Select database from { 'Human', 'Mouse', 'Yeast', 'Fly', 'Fish', 'Worm' }

```
[13]: # default: Human
      names = gp.get_library_name()
      names[:10]

[13]: ['ARCHS4_Cell-lines',
      'ARCHS4_IDG_Coexp',
      'ARCHS4_Kinases_Coexp',
      'ARCHS4_TFs_Coexp',
      'ARCHS4_Tissues',
      'Achilles_fitness_decrease',
      'Achilles_fitness_increase',
      'Aging_Perturbations_from_GEO_down',
      'Aging_Perturbations_from_GEO_up',
      'Allen_Brain_Atlas_10x_scRNA_2021']

```

```
[14]: # yeast
yeast = gp.get_library_name(organism='Yeast')
yeast[:10]

[14]: ['Cellular_Component_AutoRIF',
'Cellular_Component_AutoRIF_Predicted_zscore',
'GO_Biological_Process_2018',
'GO_Biological_Process_AutoRIF',
'GO_Biological_Process_AutoRIF_Predicted_zscore',
'GO_Cellular_Component_2018',
'GO_Cellular_Component_AutoRIF',
'GO_Cellular_Component_AutoRIF_Predicted_zscore',
'GO_Molecular_Function_2018',
'GO_Molecular_Function_AutoRIF']
```

### Parse Enrichr library into dict

```
[15]: ## download library or read a .gmt file
go_mf = gp.get_library(name='GO_Molecular_Function_2018', organism='Yeast')
print(go_mf['ATP binding (GO:0005524)'])

['MLH1', 'ECM10', 'RLI1', 'SSB1', 'SSB2', 'YTA12', 'MSH2', 'CDC6', 'HMI1', 'YNL247W',
↪ 'MSH6', 'SSQ1', 'MCM7', 'SRS2', 'HSP104', 'SSA1', 'MCX1', 'SSC1', 'ARP2', 'ARP3', 'SSE1
↪ ', 'SMC2', 'SSZ1', 'TDA10', 'ORC5', 'VPS4', 'RBK1', 'SSA4', 'NEW1', 'ORC1', 'SSA2',
↪ 'KAR2', 'SSA3', 'DYN1', 'PGK1', 'VPS33', 'LHS1', 'CDC123', 'PMS1']
```

## Over-representation analysis by Enrichr web services

The only requirement of input is a list of gene symbols.

For online web services, gene symbols are not case sensitive.

- `gene_list` accepts
  - `pd.Series`
  - `pd.DataFrame`
  - list object
  - txt file (one gene symbol per row)
- `gene_sets` accepts:
  - Multi-libraries names supported, separate each name by comma or input a list.

For example:

```
# gene_list
gene_list="./data/gene_list.txt",
gene_list=glist
# gene_sets
gene_sets='KEGG_2016'
gene_sets='KEGG_2016,KEGG_2013'
gene_sets=['KEGG_2016', 'KEGG_2013']
```

```
[16]: # read in an example gene list
gene_list = pd.read_csv("./tests/data/gene_list.txt", header=None, sep="\t")
gene_list.head()

[16]:
      0
0      IGKV4-1
1      CD55
2      IGKC
3      PPFIBP1
4      ABHD4

[17]: # convert dataframe or series to list
glist = gene_list.squeeze().str.strip().to_list()
print(glist[:10])

['IGKV4-1', 'CD55', 'IGKC', 'PPFIBP1', 'ABHD4', 'PCSK6', 'PGD', 'ARHGDIB', 'ITGB2',
→ 'CARD6']
```

## Over-representation analysis via Enrichr web services

This is an Example of the Enrichr analysis

**NOTE:** 1. Enrichr Web Services need gene symbols as input 2. Gene symbols will convert to upcases automatically.  
3. (Optional) Input an user defined background gene list

### Enrichr Web Services (without a background input)

```
[18]: # if you are only intrested in dataframe that enrichr returned, please set outdir=None
enr = gp.enrichr(gene_list=gene_list, # or "./tests/data/gene_list.txt",
                 gene_sets=['MSigDB_Hallmark_2020', 'KEGG_2021_Human'],
                 organism='human', # don't forget to set organism to the one you desired!_
→ e.g. Yeast
                 outdir=None, # don't write to disk
                 )
```

```
[19]: # obj.results stores all results
enr.results.head(5)
```

```
[19]:
      Gene_set      Term Overlap      P-value \
0  MSigDB_Hallmark_2020  IL-6/JAK/STAT3 Signaling  19/87  1.197225e-09
1  MSigDB_Hallmark_2020  TNF-alpha Signaling via NF-kB  27/200  3.220898e-08
2  MSigDB_Hallmark_2020      Complement  27/200  3.220898e-08
3  MSigDB_Hallmark_2020  Inflammatory Response  24/200  1.635890e-06
4  MSigDB_Hallmark_2020      heme Metabolism  23/200  5.533816e-06

      Adjusted P-value  Old P-value  Old Adjusted P-value  Odds Ratio \
0      5.986123e-08      0      0      6.844694
1      5.368163e-07      0      0      3.841568
2      5.368163e-07      0      0      3.841568
3      2.044862e-05      0      0      3.343018
4      5.533816e-05      0      0      3.181358
```

(continues on next page)

(continued from previous page)

	Combined Score	Genes
0	140.612324	IL4R;TGFB1;IL1R1;IFNGR1;IL10RB;ITGB3;IFNGR2;IL...
1	66.270963	BTG2;BCL2A1;PLEK;IRS2;LITAF;IFIH1;PANX1;DRAM1;...
2	66.270963	FCN1;LRP1;PLEK;LIPA;CA2;CASP3;LAMP2;S100A12;FY...
3	44.540108	LYN;IFITM1;BTG2;IL4R;CD82;IL1R1;IFNGR2;ITGB3;F...
4	38.509172	SLC22A4;MPP1;BNIP3L;BTG2;ARHGEF12;NEK7;GDE1;FO...

## Enrichr Web Service (with background input)

NOTE: Missing Overlap column in final output

```
[20]: # background only reconigized a gene list input.
enr_bg = gp.enrichr(gene_list=gene_list,
                    gene_sets=['MSigDB_Hallmark_2020','KEGG_2021_Human'],
                    # organism='human', # organism argment is ignored because user input a
↳background
                    background="tests/data/background.txt",
                    outdir=None, # don't write to disk
                    )
```

```
[21]: enr_bg.results.head() #
```

```
[21]:
```

	Gene_set	Term	P-value	\
0	MSigDB_Hallmark_2020	IL-6/JAK/STAT3 Signaling	3.559435e-11	
1	MSigDB_Hallmark_2020	TNF-alpha Signaling via NF-kB	3.401526e-10	
2	MSigDB_Hallmark_2020	Complement	3.813953e-10	
3	MSigDB_Hallmark_2020	Inflammatory Response	3.380686e-08	
4	MSigDB_Hallmark_2020	heme Metabolism	8.943634e-08	

	Adjusted P-value	Old P-value	Old adjusted P-value	Odds Ratio	\
0	1.779718e-09	0	0	8.533251	
1	6.356588e-09	0	0	4.824842	
2	6.356588e-09	0	0	4.796735	
3	4.225857e-07	0	0	4.197067	
4	8.943634e-07	0	0	4.111306	

	Combined Score	Genes
0	205.300064	IL4R;TGFB1;IL1R1;IFNGR1;IL10RB;ITGB3;IFNGR2;IL...
1	105.189414	BTG2;BCL2A1;PLEK;IRS2;LITAF;IFIH1;PANX1;DRAM1;...
2	104.027683	FCN1;LRP1;PLEK;LIPA;CA2;CASP3;LAMP2;S100A12;FY...
3	72.200480	LYN;IFITM1;BTG2;IL4R;CD82;IL1R1;IFNGR2;ITGB3;F...
4	66.725423	SLC22A4;MPP1;BNIP3L;BTG2;ARHGEF12;NEK7;GDE1;FO...

## Over-representation analysis (hypergeometric test) by offline

This API **DO NOT** use Enrichr web services.

**NOTE:** 1. The input gene symbols are **case sensitive**. 2. You need to **match the type of the gene identifiers** which used in your gene\_list input and GMT file. 3. Input a .gmt file or gene\_set dict object for the argument gene\_sets

For example:

```
gene_sets="./data/genes.gmt",
gene_sets={'A':['gene1', 'gene2',...],
          'B':['gene2', 'gene4',...],
          ...}
```

```
[22]: # NOTE: `enrich` instead of `enrichr`
enr2 = gp.enrich(gene_list="./tests/data/gene_list.txt", # or gene_list=glist
                gene_sets=["./tests/data/genes.gmt", "unknown", kegg ], # kegg is a
↳dict object
                background=None, # or "hsapiens_gene_ensembl", or int, or text file, or
↳a list of genes
                outdir=None,
                verbose=True)

2023-10-25 10:46:28,796 [INFO] User defined gene sets is given: ./tests/data/genes.gmt
2023-10-25 10:46:28,813 [INFO] Input dict object named with gs_ind_2
2023-10-25 10:46:29,289 [WARNING] Input library not found: unknown. Skip
2023-10-25 10:46:29,291 [INFO] Run: genes.gmt
2023-10-25 10:46:29,293 [INFO] Background is not set! Use all 682 genes in genes.gmt.
2023-10-25 10:46:29,302 [INFO] Run: gs_ind_2
2023-10-25 10:46:29,327 [INFO] Done.
```

```
[23]: enr2.results.head()
```

```
[23]:
```

	Gene_set	Term	Overlap	P-value	Adjusted P-value	Odds Ratio \
0	genes.gmt	BvA_UpIN_A	8/139	0.457390	0.568432	1.161982
1	genes.gmt	BvA_UpIN_B	12/130	0.026744	0.187208	2.160059
2	genes.gmt	CvA_UpIN_A	1/12	0.481190	0.568432	2.266479
3	genes.gmt	DvA_UpIN_A	16/284	0.426669	0.568432	1.127395
4	genes.gmt	DvA_UpIN_D	13/236	0.487227	0.568432	1.084567

	Combined Score	Genes
0	0.908925	PCSK6;MAP3K5;MBOAT2;MSRB2;IQGAP2;HAL;PADI2;IL1R1
1	7.822534	FAM65B;MBNL3;GPX8;DYSF;KCTD12;HEBP1;SUOX;ARHGD...
2	1.657913	MBOAT2
3	0.960255	PCSK6;FXYP6;IFNGR2;MAP3K5;MBOAT2;VNN1;IQGAP2;H...
4	0.779830	GNB4;FAM198B;FAM65B;TXNDC5;GLIPR2;MBNL3;GPX8;D...



## About Background genes

By default, all genes in the `gene_sets` input will be used as background.

However, a better background genes would be the following:

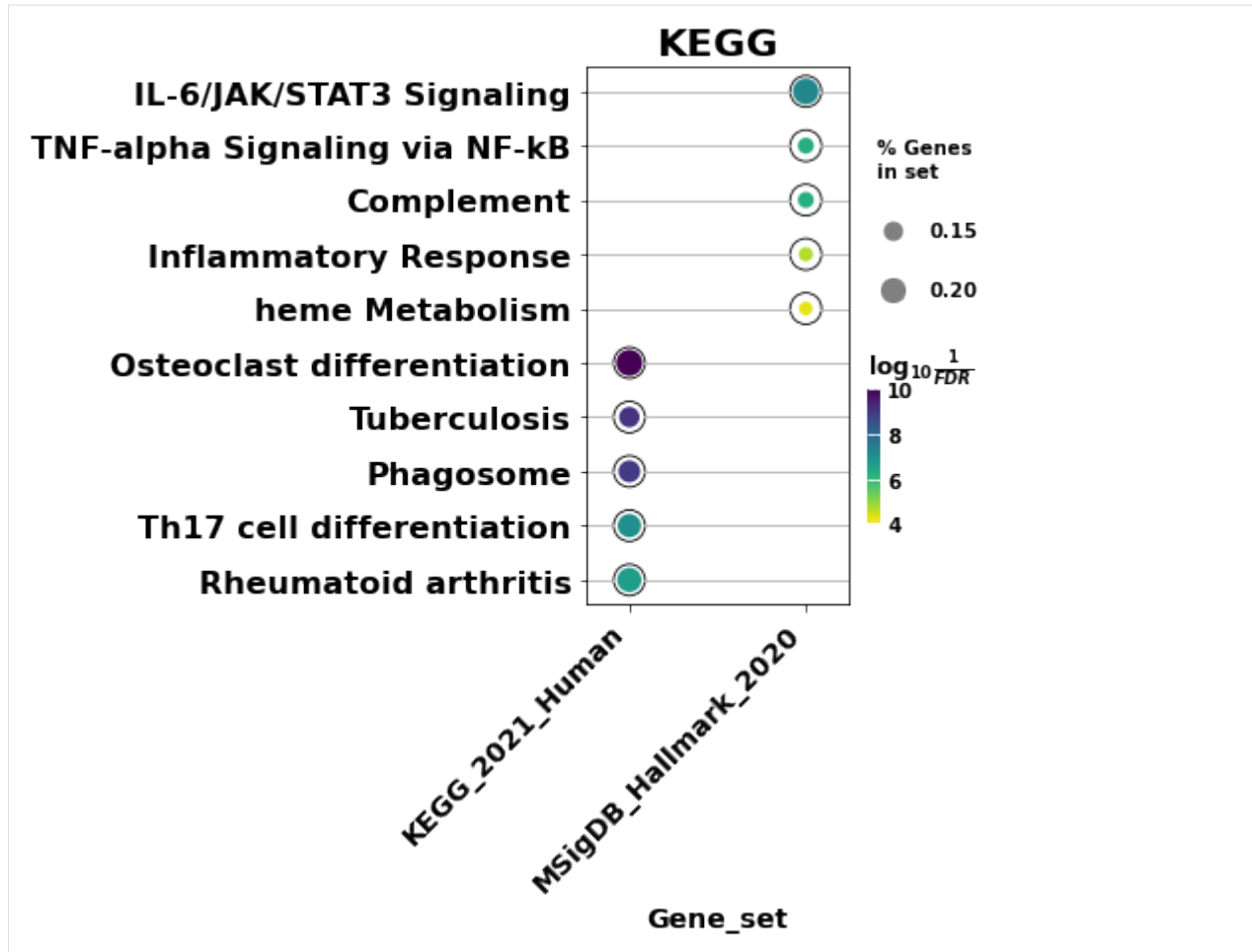
1. (Recommended) Input a list of background genes: `['gene1', 'gene2', ...]`
  - The background gene list is defined by your experiment. e.g. the expressed genes in your RNA-seq.
  - The gene identifier in `gmt/dict` should be the same type to the background genes.
2. Specify a number: e.g. 20000. (the number of total expressed genes).
  - This works, but not recommend. It assumes that all your genes could be found in background.
  - If genes exist in `gmt` but not included in background provided, they will affect the significance of the statistical test.
3. Set a Biomart dataset name: e.g. `"hsapiens_gene_ensembl"`
  - The background will use all annotated genes from the BioMart datasets you've chosen.
  - The program will try to retrieve the background information automatically.

## Plotting

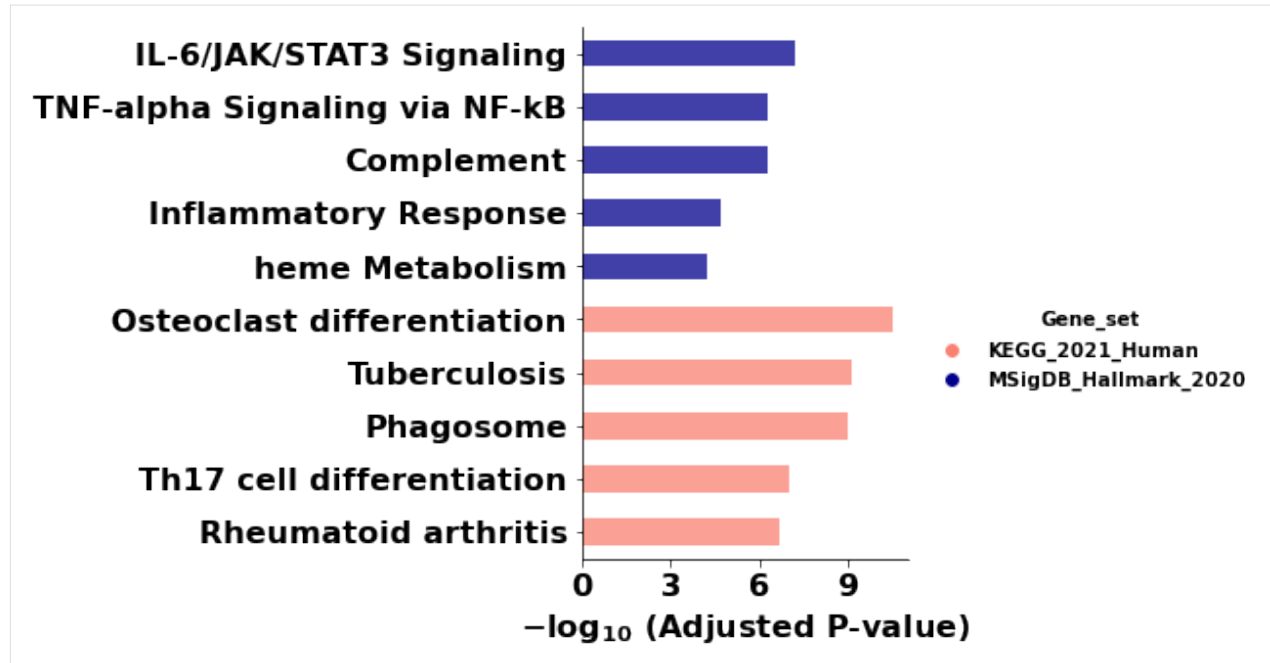
Show top 5 terms of each `gene_set` ranked by "Adjusted P-value"

```
[24]: # simple plotting function
from gseapy import barplot, dotplot
```

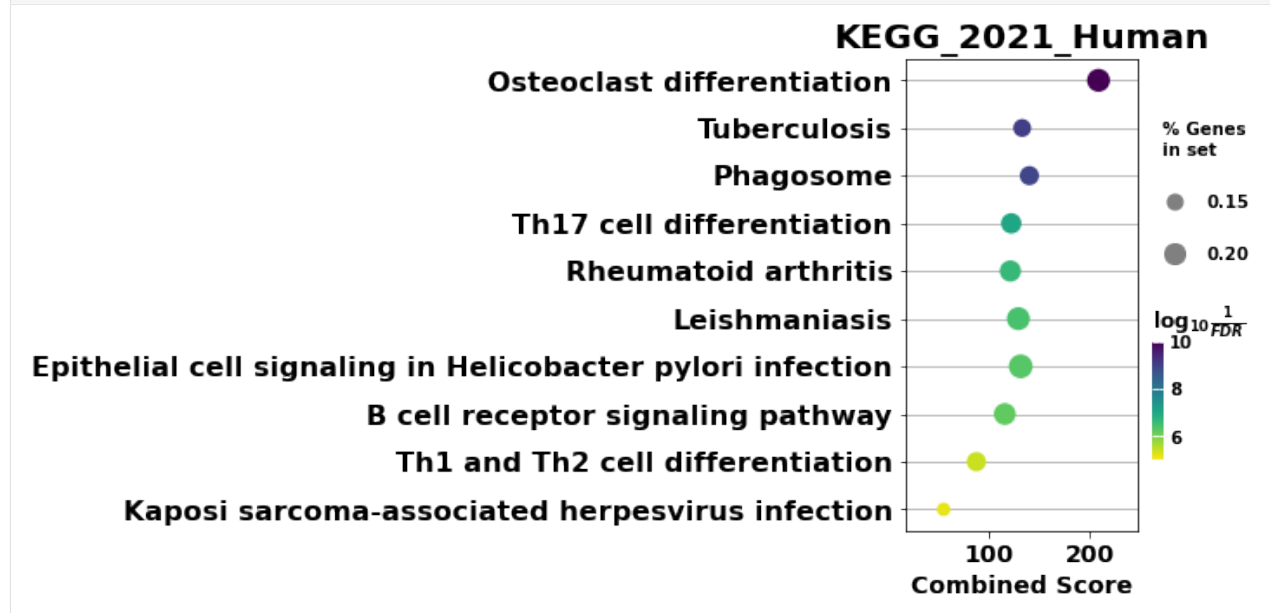
```
[25]: # categorical scatterplot
ax = dotplot(enr.results,
             column="Adjusted P-value",
             x='Gene_set', # set x axis, so you could do a multi-sample/library_
↪comparsion
             size=10,
             top_term=5,
             figsize=(3,5),
             title = "KEGG",
             xticklabels_rot=45, # rotate xtick labels
             show_ring=True, # set to False to remove outer ring
             marker='o',
             )
```



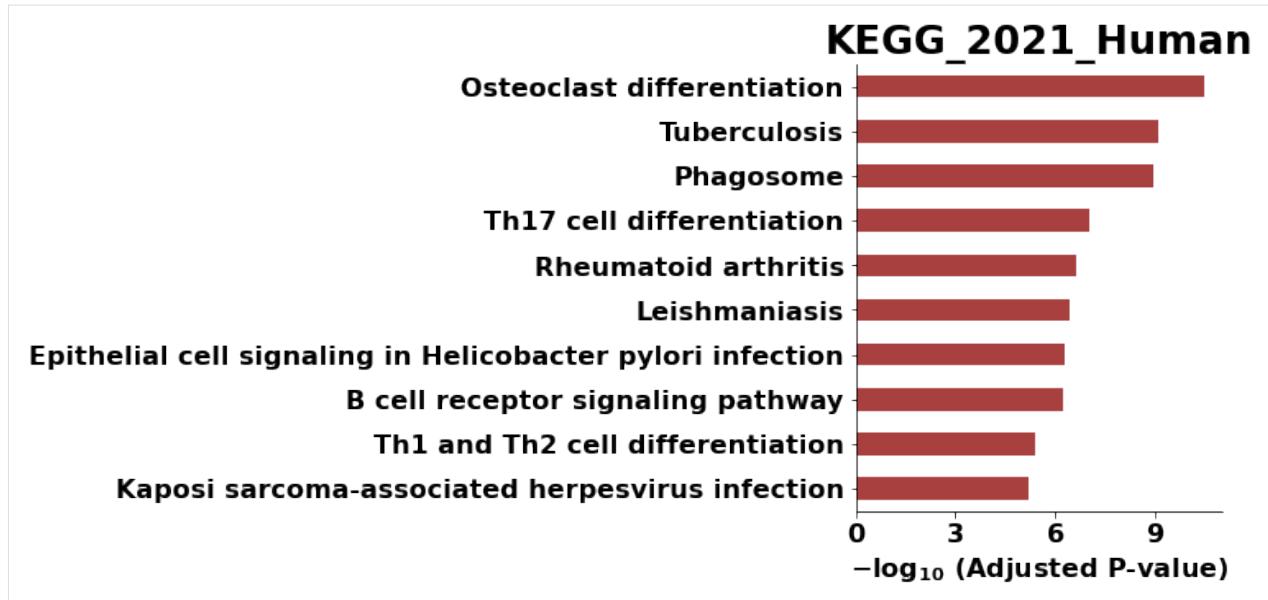
```
[26]: # categorical scatterplot
ax = barplot(enr.results,
             column="Adjusted P-value",
             group='Gene_set', # set group, so you could do a multi-sample/library_
             ↪comparsion
             size=10,
             top_term=5,
             figsize=(3,5),
             #color=['darkred', 'darkblue'] # set colors for group
             color = {'KEGG_2021_Human': 'salmon', 'MSigDB_Hallmark_2020': 'darkblue'})
```



```
[27]: # to save your figure, make sure that ``ofname`` is not None
ax = dotplot(enr.res2d, title='KEGG_2021_Human', cmap='viridis_r', size=10, figsize=(3,5))
```



```
[28]: # to save your figure, make sure that ``ofname`` is not None
ax = barplot(enr.res2d, title='KEGG_2021_Human', figsize=(4, 5), color='darkred')
```



### Command line usage

the option `-v` will print out the progress of your job

```
[29]: # !gseapy enrichr -i ./data/gene_list.txt \
#           -g GO_Biological_Process_2017 \
#           -v -o test/enrichr_BP
```

## 5.2.4 Prerank example

### Assign prerank() with

- `pd.DataFrame`: Only contains two columns, or one column with `gene_name` indexed
- `pd.Series`
- a txt file:
  - GSEAPy will skip any data after “#”.
  - Do not include header in your gene list !

### NOTE: UPCASES for gene symbols by Default

1. Gene symbols are all “UPCASES” in the Enrichr Libraries. You should convert your input gene identifier to “UPCASES” first.
2. If input `gmt`, `dict` object, please refer to 1.2 Mouse gene symbols maps to Human, or Vice Versa (in this page) to convert gene identifier

## Supported gene\_sets input

For example:

```
gene_sets="KEGG_2016",
gene_sets="KEGG_2016,KEGG2013",
gene_sets="./data/genes.gmt",
gene_sets=["KEGG_2016","./data/genes.gmt"],
gene_sets={'A':['gene1', 'gene2',...],
           'B':['gene2', 'gene4',...],
           ...}
```

```
[30]: rnk = pd.read_csv("./tests/data/temp.rnk", header=None, index_col=0, sep="\t")
      rnk.head()
```

```
[30]:      1
      0
ATXN1   16.456753
UBQLN4   13.989493
CALM1    13.745533
DLG4     12.796588
MRE11A   12.787631
```

```
[31]: rnk.shape
```

```
[31]: (22922, 1)
```

```
[32]: # # run prerank
      # # enrichr libraries are supported by prerank module. Just provide the name
      # # use 4 process to acceralate the permutation speed
pre_res = gp.prerank(rnk="./tests/data/temp.rnk", # or rnk = rnk,
                    gene_sets='KEGG_2016',
                    threads=4,
                    min_size=5,
                    max_size=1000,
                    permutation_num=1000, # reduce number to speed up testing
                    outdir=None, # don't write to disk
                    seed=6,
                    verbose=True, # see what's going on behind the scenes
                    )
```

```
2023-10-25 10:46:30,863 [WARNING] Duplicated values found in preranked stats: 4.97% of
↳ genes
The order of those genes will be arbitrary, which may produce unexpected results.
2023-10-25 10:46:30,864 [INFO] Parsing data files for GSEA...
2023-10-25 10:46:30,865 [INFO] Enrichr library gene sets already downloaded in: /home/
↳ fangzq/.cache/gseapy, use local file
2023-10-25 10:46:30,902 [INFO] 0001 gene_sets have been filtered out when max_size=1000
↳ and min_size=5
2023-10-25 10:46:30,903 [INFO] 0292 gene_sets used for further statistical testing...
2023-10-25 10:46:30,903 [INFO] Start to run GSEA...Might take a while...
2023-10-25 10:46:43,563 [INFO] Congratulations. GSEapy runs successfully...
```

[ ]:

## How to generate your GSEA plot inside python console

Visualize it using gseaplot

Make sure that ofname is not None, if you want to save your figure to the disk

```
[33]: pre_res.res2d.head(5)
```

```
[33]:
```

	Name	Term	ES	\
0	prerank	Adherens junction Homo sapiens hsa04520	0.784625	
1	prerank	Glioma Homo sapiens hsa05214	0.784678	
2	prerank	Estrogen signaling pathway Homo sapiens hsa04915	0.766347	
3	prerank	Thyroid hormone signaling pathway Homo sapiens...	0.7577	
4	prerank	Long-term potentiation Homo sapiens hsa04720	0.778249	

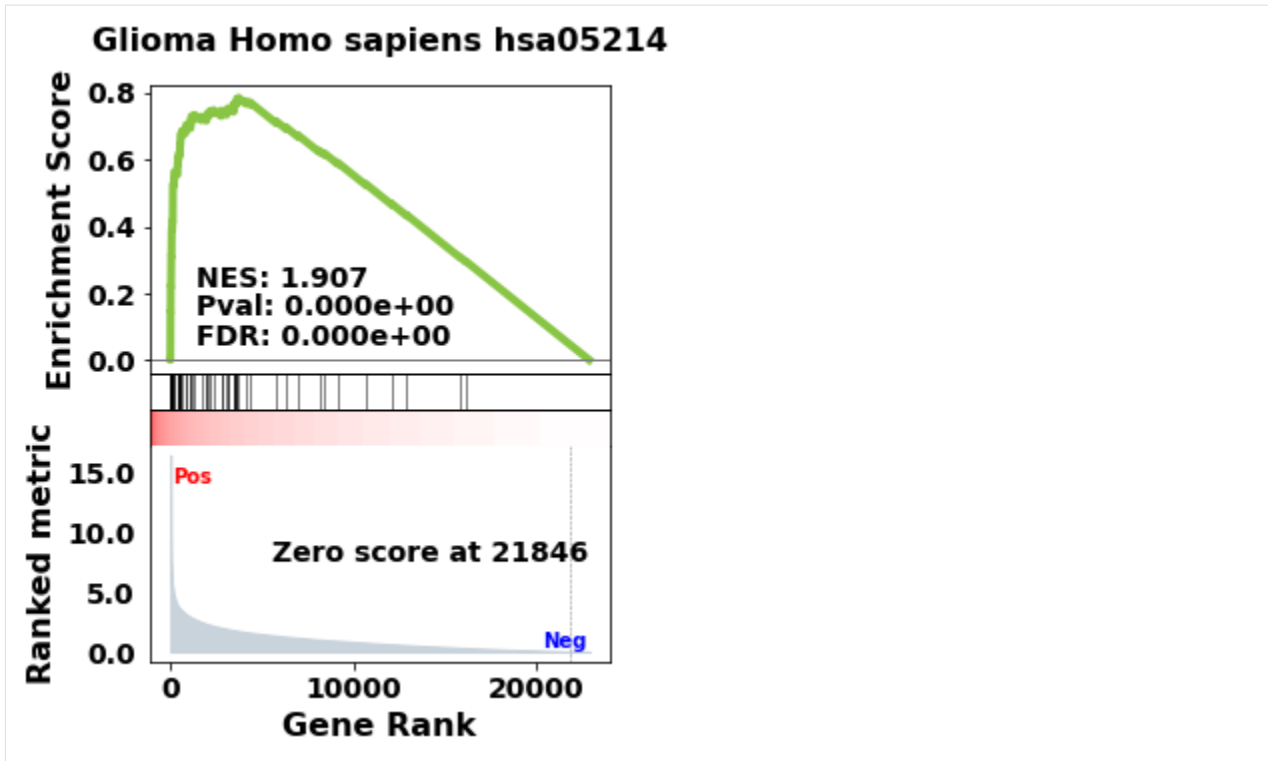
  

	NES	NOM	p-val	FDR	q-val	FWER	p-val	Tag %	Gene %	\
0	1.912548		0.0		0.0		0.0	47/74	10.37%	
1	1.906706		0.0		0.0		0.0	52/65	16.29%	
2	1.897957		0.0		0.0		0.0	74/99	16.57%	
3	1.891815		0.0		0.0		0.0	84/118	16.29%	
4	1.888739		0.0		0.0		0.0	42/66	9.01%	

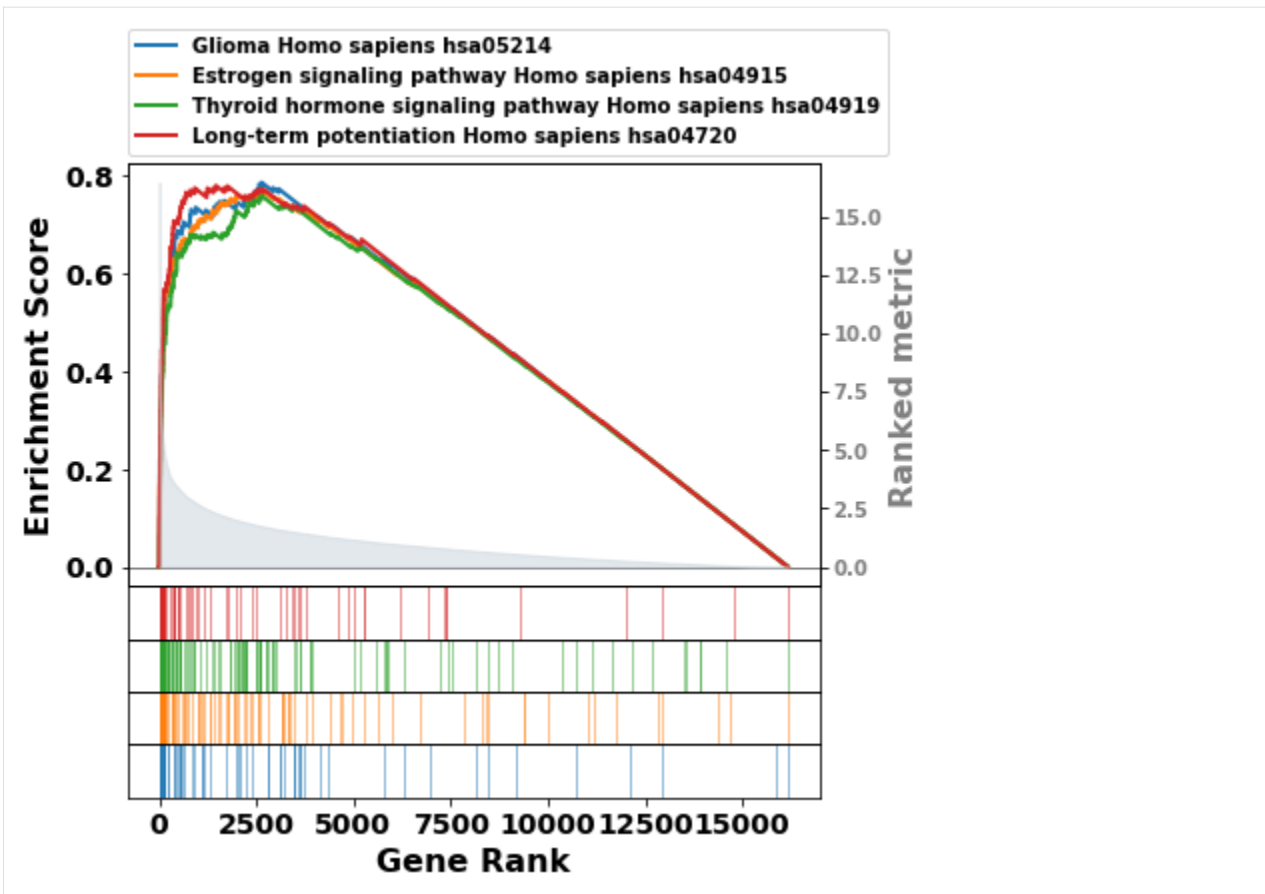
	Lead_genes
0	CTNNB1;EGFR;RAC1;TGFB1;SMAD4;MET;EP300;CDC42;...
1	CALM1;GRB2;EGFR;PRKCA;KRAS;HRAS;TP53;MAPK1;PRK...
2	CALM1;PRKACA;GRB2;SP1;EGFR;KRAS;HRAS;HSP90AB1;...
3	CTNNB1;PRKACA;PRKCA;KRAS;NOTCH1;EP300;CREBBP;H...
4	CALM1;PRKACA;PRKCA;KRAS;EP300;CREBBP;HRAS;PRKA...

```
[34]: ## easy way
terms = pre_res.res2d.Term
axs = pre_res.plot(terms=terms[1]) # v1.0.5
# to make more control on the plot, use
# from gseapy import gseaplot
# gseaplot(rank_metric=pre_res.ranking, term=terms[0], ofname='your.plot.pdf', **pre_res.
↪ results[terms[0]])
```



or multi pathway in one

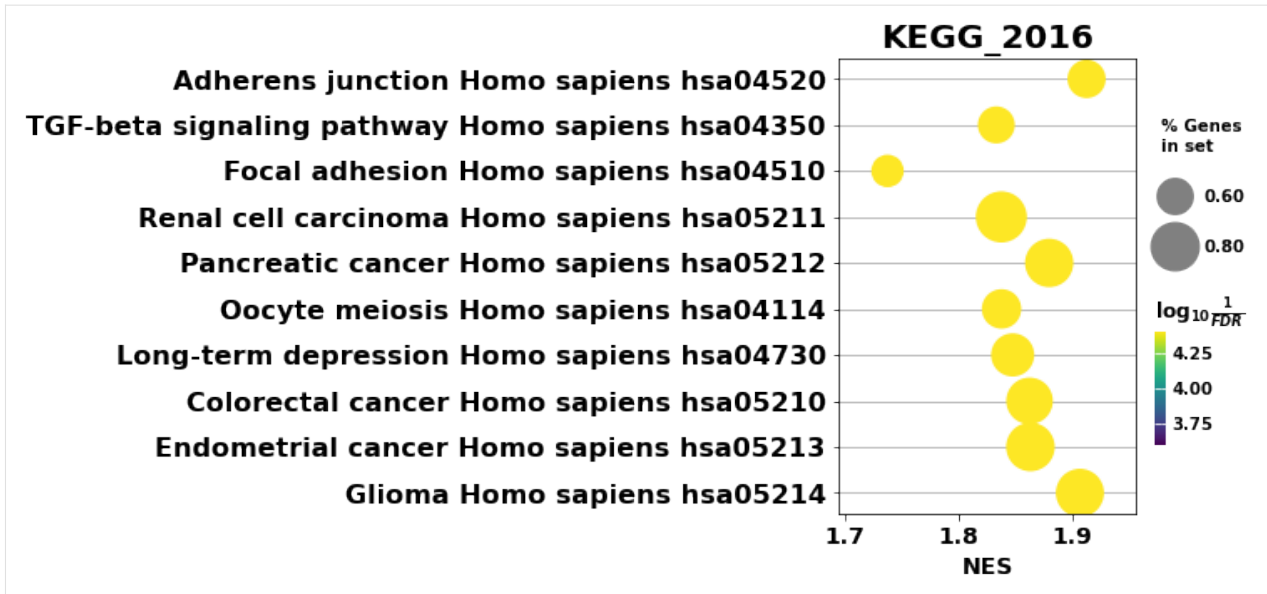
```
[35]: ax = pre_res.plot(terms=terms[1:5],
                        #legend_kws={'loc': (1.2, 0)}, # set the legend loc
                        show_ranking=True, # whether to show the second yaxis
                        figsize=(3,4)
                        )
# or use this to have more control on the plot
# from gseapy import gseaplot2
# terms = pre_res.res2d.Term[1:5]
# hits = [pre_res.results[t]['hits'] for t in terms]
# runes = [pre_res.results[t]['RES'] for t in terms]
# fig = gseaplot2(terms=terms, res=runes, hits=hits,
#                 rank_metric=gs_res.ranking,
#                 legend_kws={'loc': (1.2, 0)}, # set the legend loc
#                 figsize=(4,5)) # rank_metric=pre_res.ranking
```



dotplot for GSEA results

```
[36]: from gseapy import dotplot
# to save your figure, make sure that ``ofname`` is not None
ax = dotplot(pre_res.res2d,
             column="FDR q-val",
             title='KEGG_2016',
             cmap=plt.cm.viridis,
             size=6, # adjust dot size
             figsize=(4,5), cutoff=0.25, show_ring=False)
```





#### Network Visualization

- use `enrichment_map` to build network
- save the nodes and edges. They could be used for cytoscape visualization.

```
[37]: from gseapy import enrichment_map
# return two dataframe
nodes, edges = enrichment_map(pre_res.res2d)
```

```
[38]: import networkx as nx
```

```
[39]: # build graph
G = nx.from_pandas_edgelist(edges,
                           source='src_idx',
                           target='targ_idx',
                           edge_attr=['jaccard_coef', 'overlap_coef', 'overlap_genes'])
```

```
[40]: fig, ax = plt.subplots(figsize=(8, 8))

# init node coordinates
pos=nx.layout.spiral_layout(G)
#node_size = nx.get_node_attributes()
# draw node
nx.draw_networkx_nodes(G,
                       pos=pos,
                       cmap=plt.cm.RdYlBu,
                       node_color=list(nodes.NES),
                       node_size=list(nodes.Hits_ratio * 1000))

# draw node label
nx.draw_networkx_labels(G,
                       pos=pos,
                       labels=nodes.Term.to_dict())

# draw edge
```

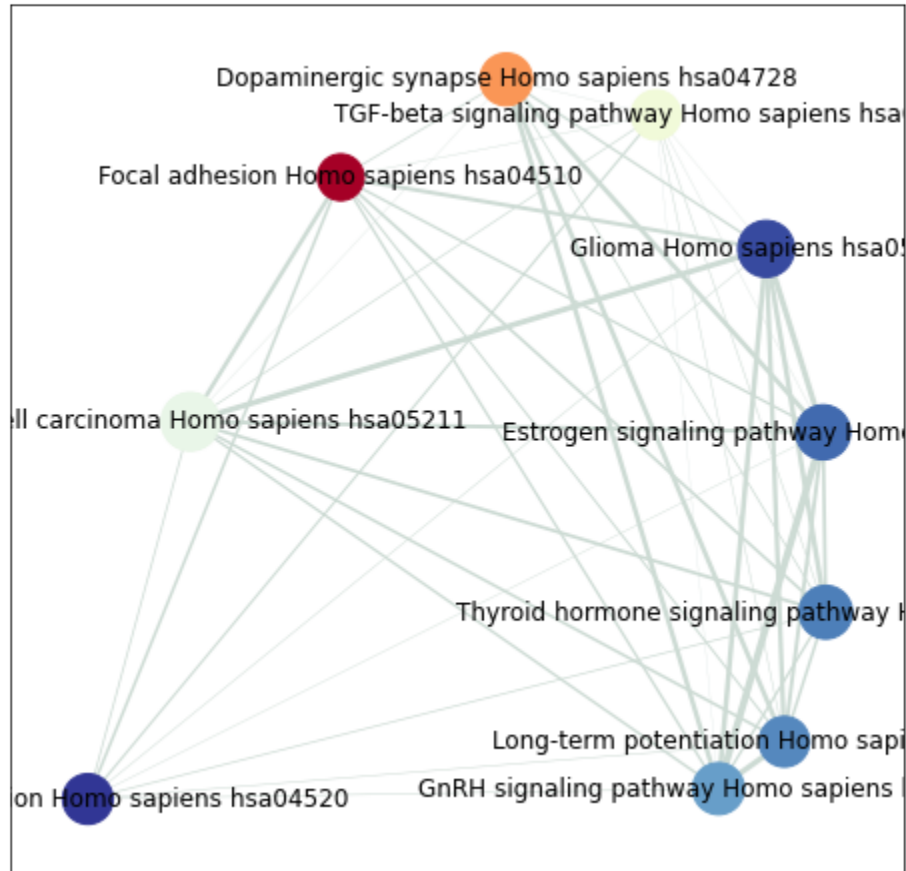
(continues on next page)

(continued from previous page)

```

edge_weight = nx.get_edge_attributes(G, 'jaccard_coef').values()
nx.draw_networkx_edges(G,
                        pos=pos,
                        width=list(map(lambda x: x*10, edge_weight)),
                        edge_color='#CDDBD4')
plt.show()

```



## Command line usage

You may also want to use prerank in command line

```
[41]: # !gseapy prerank -r temp.rnk -g temp.gmt -o prerank_report_temp
```

## 5.2.5 GSEA Example

### Inputs

Assign gsea()

- data with:
  - pandas DataFrame
  - .gct format file, or a text file
- cls with:
  - a list
  - a .cls format file
- gene\_sets with:

```
gene_sets="KEGG_2016",
gene_sets="KEGG_2016,KEGG2013",
gene_sets="./data/genes.gmt",
gene_sets=["KEGG_2016","./data/genes.gmt"],
gene_sets={'A':['gene1', 'gene2',...],
           'B':['gene2', 'gene4',...],
           ...}
```

### NOTE: UPCASES for gene symbols by Default

1. Gene symbols are all “UPCASES” in the Enrichr Libraries. You should convert your input gene identifier to “UPCASES” first.
2. If input gmt, dict object, please refer to 1.2 Mouse gene symbols maps to Human, or Vice Versa (in this page) to convert gene identifier

```
[42]: import gseapy as gp
      phenoA, phenoB, class_vector = gp.parser.gsea_cls_parser("./tests/extdata/Leukemia.cls")
```

```
[43]: #class_vector used to indicate group attributes for each sample
      print(class_vector)

['ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL',
→ 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL', 'ALL',
→ 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML',
→ 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML']
```

```
[44]: gene_exp = pd.read_csv("./tests/extdata/Leukemia_hgu95av2.trim.txt", sep="\t")
      gene_exp.head()
```

```
[44]:
```

	Gene	NAME	ALL_1	ALL_2	ALL_3	ALL_4	ALL_5	ALL_6	ALL_7	\
0	MAPK3	1000_at	1633.6	2455.0	866.0	1000.0	3159.0	1998.0	1580.0	
1	TIE1	1001_at	284.4	159.0	173.0	216.0	1187.0	647.0	352.0	
2	CYP2C19	1002_f_at	285.8	114.0	429.0	-43.0	18.0	366.0	119.0	
3	CXCR5	1003_s_at	-126.6	-388.0	143.0	-915.0	-439.0	-371.0	-448.0	
4	CXCR5	1004_at	-83.3	33.0	195.0	85.0	54.0	-6.0	55.0	

(continues on next page)

(continued from previous page)

```

    ALL_8  ...  AML_15  AML_16  AML_17  AML_18  AML_19  AML_20  AML_21  \
0  1955.0  ...  1826.0  2849.0  2980.0  1442.0  3672.0   294.0  2188.0
1  1224.0  ...  1556.0   893.0  1278.0   301.0   797.0   248.0   167.0
2   -88.0  ...  -177.0    64.0  -359.0    68.0     2.0  -464.0  -127.0
3  -862.0  ...   237.0  -834.0 -1940.0  -684.0 -1236.0 -1561.0  -895.0
4   101.0  ...    86.0   -5.0   487.0   102.0   33.0  -153.0   -50.0

    AML_22  AML_23  AML_24
0  1245.0  1934.0  13154.0
1   941.0  1398.0   -502.0
2  -279.0   301.0   509.0
3 -1016.0 -2238.0  -1362.0
4   257.0   439.0   386.0

[5 rows x 50 columns]
```

```
[45]: print("positively correlated: ", phenoA)
```

```
positively correlated:  ALL
```

```
[46]: print("negtively correlated: ", phenoB)
```

```
negtively correlated:  AML
```

```
[47]: # run gsea
# enrichr libraries are supported by gsea module. Just provide the name
gs_res = gp.gsea(data=gene_exp, # or data='./P53_resampling_data.txt'
                gene_sets='./tests/extdata/h.all.v7.0.symbols.gmt', # or enrichr_
                ↪library names
                cls= "./tests/extdata/Leukemia.cls", # cls=class_vector
                # set permutation_type to phenotype if samples >=15
                permutation_type='phenotype',
                permutation_num=1000, # reduce number to speed up test
                outdir=None, # do not write output to disk
                method='signal_to_noise',
                threads=4, seed= 7)
```

```
2023-10-25 10:46:47,125 [WARNING] Found duplicated gene names, values averaged by gene_
↪names!
```

You can set pheno\_pos, and pheno\_neg manually

```
[48]: # example
from gseapy import GSEA
gs = GSEA(data=gene_exp,
          gene_sets='KEGG_2016',
          classes = class_vector, # cls=class_vector
          # set permutation_type to phenotype if samples >=15
          permutation_type='phenotype',
          permutation_num=1000, # reduce number to speed up test
          outdir=None,
          method='signal_to_noise',
```

(continues on next page)

(continued from previous page)

```

        threads=4, seed= 8)
gs.pheno_pos = "AML"
gs.pheno_neg = "ALL"
gs.run()

```

```

2023-10-25 10:46:50,381 [WARNING] Found duplicated gene names, values averaged by gene_
names!

```

## Show the gsea plots

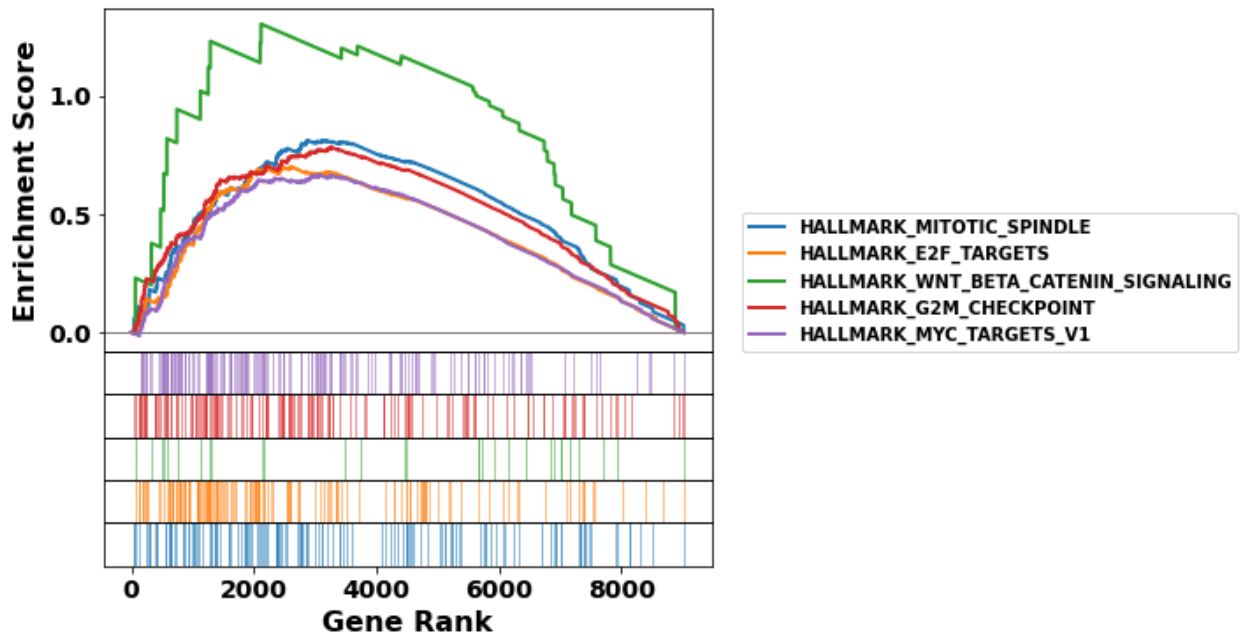
The **gsea** module will generate heatmap for genes in each gene sets in the background.

But if you need to do it yourself, use the code below

```

[49]: terms = gs_res.res2d.Term
      axs = gs_res.plot(terms[:5], show_ranking=False, legend_kws={'loc': (1.05, 0)}, )

```



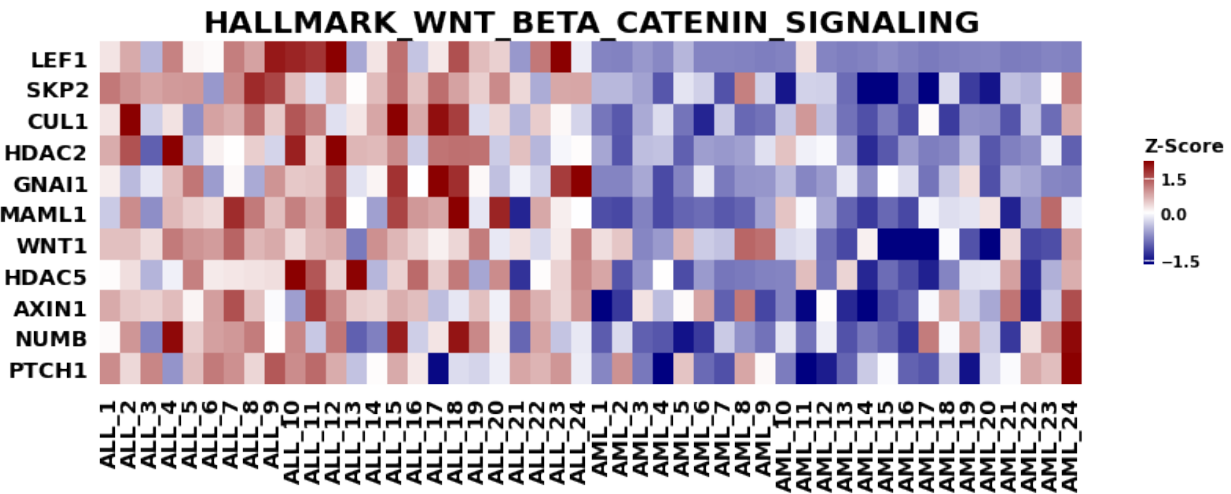
```

[50]: # or use
      # from gseapy import gseaplot2

      # # multi in one
      # terms = gs_res.res2d.Term[:5]
      # hits = [gs_res.results[t]['hits'] for t in terms]
      # runes = [gs_res.results[t]['RES'] for t in terms]
      # fig = gseaplot2(terms=terms, res=runes, hits=hits,
      #                  rank_metric=gs_res.ranking,
      #                  legend_kws={'loc': (1.2, 0)}, # set the legend loc
      #                  figsize=(4,5)) # rank_metric=pre_res.ranking

```

```
[51]: from gseapy import heatmap
# plotting heatmap
i = 2
genes = gs_res.res2d.Lead_genes[i].split(";")
# Make sure that ``ofname`` is not None, if you want to save your figure to disk
ax = heatmap(df = gs_res.heatmap.loc[genes], z_score=0, title=terms[i], figsize=(14,4))
```



```
[52]: gs_res.heatmap.loc[genes]
```

```
[52]:
```

	ALL_1	ALL_2	ALL_3	ALL_4	ALL_5	ALL_6	ALL_7	ALL_8	\
Gene									
LEF1	8544.10	12552.0	2869.0	15265.0	7446.0	6991.0	15520.0	13114.0	
SKP2	23.80	-45.0	-95.0	-71.0	-65.0	-547.0	-24.0	230.0	
CUL1	1712.75	3309.0	1273.5	1726.5	947.5	2160.0	2065.0	2524.5	
HDAC2	4542.90	6030.0	1195.0	9368.0	2281.0	3407.0	3175.0	3962.0	
GNAI1	588.50	163.0	364.0	882.0	1317.0	17.0	518.0	89.0	
MAML1	871.40	1871.0	578.0	1589.0	1448.0	1364.0	2494.0	1989.0	
WNT1	-872.50	-875.0	-1012.0	-535.0	-654.0	-694.0	-421.0	-827.0	
HDAC5	2137.20	2374.0	1651.0	2012.0	3132.0	2279.0	2314.0	2349.0	
AXIN1	-433.50	-722.0	-808.0	-623.0	-1167.0	-326.0	448.0	-661.0	
NUMB	1033.60	1474.0	600.0	2106.0	1239.0	1430.0	1468.0	1594.0	
PTCH1	352.60	86.0	372.5	-326.5	181.5	413.5	337.0	94.5	

	ALL_9	ALL_10	...	AML_15	AML_16	AML_17	AML_18	AML_19	AML_20	\
Gene			...							
LEF1	22604.0	21795.0	...	682.0	152.0	-348.0	30.0	210.0	350.0	
SKP2	159.0	-162.0	...	-865.0	-642.0	-1005.0	-413.0	-733.0	-812.0	
CUL1	1882.5	2684.5	...	851.5	614.5	1560.0	523.0	952.0	935.0	
HDAC2	2616.0	6848.0	...	1072.0	1918.0	1545.0	1653.0	2328.0	1061.0	
GNAI1	1136.0	816.0	...	470.0	313.0	-163.0	210.0	684.0	-331.0	
MAML1	1538.0	1946.0	...	390.0	233.0	1075.0	962.0	997.0	1316.0	
WNT1	-770.0	-1001.0	...	-2506.0	-2791.0	-2249.0	-1201.0	-1819.0	-2599.0	
HDAC5	2376.0	5455.0	...	1215.0	1024.0	760.0	1368.0	1923.0	1927.0	
AXIN1	-1315.0	-1991.0	...	-2590.0	-2417.0	-1321.0	-466.0	-1628.0	-1910.0	
NUMB	1014.0	1549.0	...	491.0	342.0	1594.0	990.0	1436.0	841.0	
PTCH1	534.0	354.5	...	-62.5	-454.0	-487.0	-185.0	-663.5	-150.0	

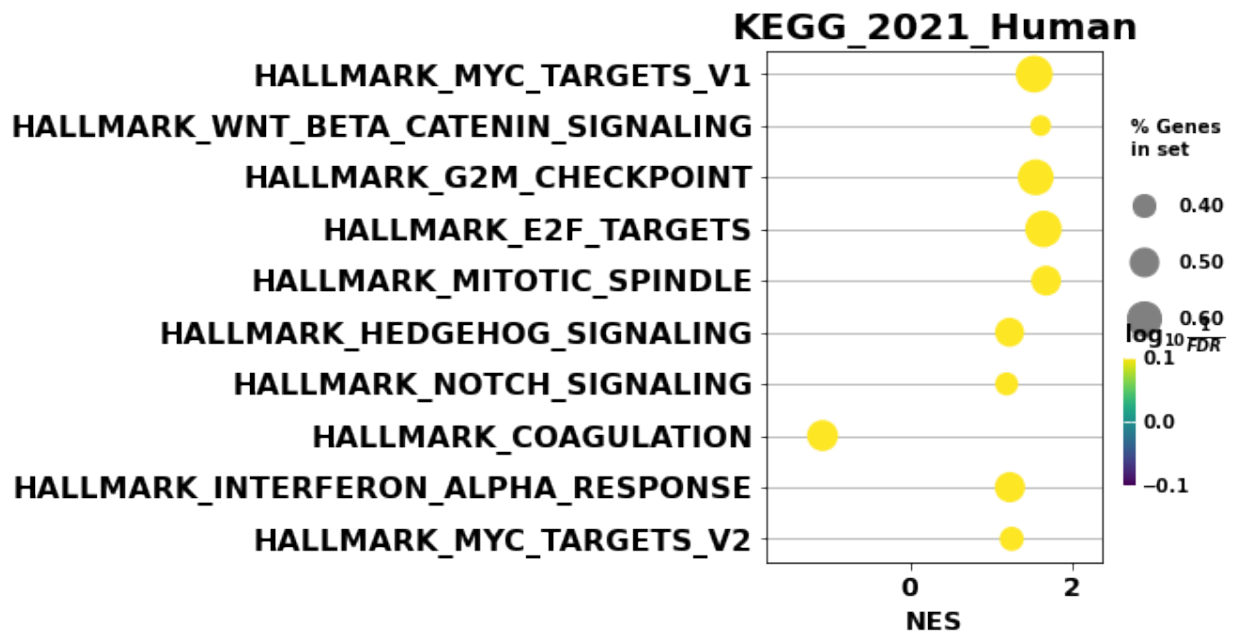
(continues on next page)

(continued from previous page)

Gene	AML_21	AML_22	AML_23	AML_24
LEF1	-242.0	-47.0	176.0	14.0
SKP2	-464.0	-490.0	-333.0	7.0
CUL1	646.0	1214.5	770.0	2088.5
HDAC2	1571.0	1749.0	2942.0	1174.0
GNAI1	115.0	55.0	-80.0	-94.0
MAML1	48.0	609.0	2090.0	1056.0
WNT1	-995.0	-1861.0	-1835.0	-714.0
HDAC5	2872.0	848.0	1629.0	2763.0
AXIN1	93.0	-2951.0	-1666.0	471.0
NUMB	352.0	1158.0	1541.0	2109.0
PTCH1	-58.0	252.0	178.5	1061.0

[11 rows x 48 columns]

```
[53]: from gseapy import dotplot
# to save your figure, make sure that ``ofname`` is not None
ax = dotplot(gs_res.res2d,
             column="FDR q-val",
             title='KEGG_2021_Human',
             cmap=plt.cm.viridis,
             size=5,
             figsize=(4,5), cutoff=1)
```



## Command line usage

You may also want to use gsea in command line

```
[54]: # !gseapy gsea -d ./data/P53_resampling_data.txt \
#         -g KEGG_2016 -c ./data/P53.cls \
#         -o test/gsea_reprot_2 \
#         -v --no-plot \
#         -t phenotype
```

## 5.2.6 Single Sample GSEA example

What's ssGSEA? Which one should I use? Prerank or ssGSEA

see FAQ [here](#)

Assign - data with - a txt file, gct file, - pd.DataFrame - pd.Seires(gene name as index)

- gene\_sets with:

```
gene_sets="KEGG_2016",
gene_sets="KEGG_2016,KEGG2013",
gene_sets="./data/genes.gmt",
gene_sets=["KEGG_2016", "./data/genes.gmt"],
gene_sets={'A': ['gene1', 'gene2', ...],
           'B': ['gene2', 'gene4', ...],
           ...}
```

1. Gene symbols are all “UPCASES” in the Enrichr Libraries. You should convert your input gene identifier to “UPCASES” first.
2. If input gmt, dict object, please refer to 1.2 Mouse gene symbols maps to Human, or Vice Versa (in this page) to convert gene identifier

```
[55]: import gseapy as gp
# txt, gct file input
ss = gp.ssgsea(data='./tests/extdata/Leukemia_hgu95av2.trim.txt',
               gene_sets='./tests/extdata/h.all.v7.0.symbols.gmt',
               outdir=None,
               sample_norm_method='rank', # choose 'custom' will only use the raw value_
               ↪of `data`
               no_plot=True)
```

```
2023-10-25 10:46:59,844 [WARNING] Found duplicated gene names, values averaged by gene_
↪names!
```

```
[56]: ss.res2d.head()
```

```
[56]:
```

	Name	Term	ES	NES
0	ALL_2	HALLMARK_MYC_TARGETS_V1	3393.823575	0.707975
1	ALL_12	HALLMARK_MYC_TARGETS_V1	3385.626111	0.706265
2	AML_11	HALLMARK_MYC_TARGETS_V1	3359.186716	0.700749
3	ALL_14	HALLMARK_MYC_TARGETS_V1	3348.938881	0.698611
4	ALL_17	HALLMARK_MYC_TARGETS_V1	3335.065348	0.695717



```
[57]: # or assign a dataframe, or Series to ssgsea()
ssdf = pd.read_csv("./tests/data/temp.rnk", header=None, index_col=0, sep="\t")
ssdf.head()
```

```
[57]:      1
0
ATXN1  16.456753
UBQLN4 13.989493
CALM1   13.745533
DLG4    12.796588
MRE11A 12.787631
```

```
[58]: # dataframe with one column is also supported by ssGSEA or Prerank
# But you have to set gene_names as index
ssdf2 = ssdf.squeeze()
```

```
[59]: # Series, DataFrame Example
# supports dataframe and series
temp = gp.ssgsea(data=ssdf2, gene_sets="./tests/data/temp.gmt")
```

## Access Enrichment Score (ES) and NES

Results are saved to obj.res2d

```
[60]: # NES and ES
ss.res2d.sort_values('Name').head()
```

```
[60]:      Name      Term      ES      NES
601  ALL_1  HALLMARK_PANCREAS_BETA_CELLS -1280.654659 -0.267153
934  ALL_1      HALLMARK_APOPTOSIS    970.818772  0.202519
1774 ALL_1  HALLMARK_HEDGEHOG_SIGNALING  431.446694  0.090003
279  ALL_1  HALLMARK_INTERFERON_ALPHA_RESPONSE 1721.458034  0.359108
1778 ALL_1  HALLMARK_BILE_ACID_METABOLISM -429.127871 -0.089519
```

```
[61]: nes = ss.res2d.pivot(index='Term', columns='Name', values='NES')
nes.head()
```

```
[61]: Name      ALL_1      ALL_10      ALL_11      ALL_12  \
Term
HALLMARK_ADIPOGENESIS      0.287384  0.274548  0.290059  0.285388
HALLMARK_ALLOGRAFT_REJECTION  0.06177  0.028062  0.096589  0.080713
HALLMARK_ANDROGEN_RESPONSE  0.133453  0.113911  0.193074  0.201531
HALLMARK_ANGIOGENESIS      -0.113481 -0.182411 -0.195637 -0.094817
HALLMARK_APICAL_JUNCTION  0.051372  0.063763  0.054601  0.014385

Name      ALL_13      ALL_14      ALL_15      ALL_16  \
Term
HALLMARK_ADIPOGENESIS      0.322757  0.305239  0.275686  0.266209
HALLMARK_ALLOGRAFT_REJECTION  0.082701  0.102735  0.12525  0.147262
HALLMARK_ANDROGEN_RESPONSE  0.151001  0.12967  0.173563  0.144836
HALLMARK_ANGIOGENESIS      -0.163717 -0.139243 -0.119084 -0.154526
HALLMARK_APICAL_JUNCTION  0.049019  0.05269  0.064787  0.052192
```

(continues on next page)

(continued from previous page)

Name	ALL_17	ALL_18	...	AML_22	AML_23	\
Term			...			
HALLMARK_ADIPOGENESIS	0.315803	0.282617	...	0.277755	0.261477	
HALLMARK_ALLOGRAFT_REJECTION	0.124621	0.091077	...	0.185738	0.157852	
HALLMARK_ANDROGEN_RESPONSE	0.180214	0.180801	...	0.180443	0.188891	
HALLMARK_ANGIOGENESIS	-0.06829	-0.121156	...	0.054883	-0.023782	
HALLMARK_APICAL_JUNCTION	0.05607	0.064936	...	0.10927	0.090065	
Name	AML_24	AML_3	AML_4	AML_5	\	
Term						
HALLMARK_ADIPOGENESIS	0.200083	0.312948	0.342963	0.253282		
HALLMARK_ALLOGRAFT_REJECTION	0.055585	0.218827	0.172395	0.199077		
HALLMARK_ANDROGEN_RESPONSE	0.197979	0.174892	0.14285	0.184843		
HALLMARK_ANGIOGENESIS	0.119022	-0.067741	0.04843	0.012808		
HALLMARK_APICAL_JUNCTION	0.155801	0.091556	0.110045	0.101659		
Name	AML_6	AML_7	AML_8	AML_9		
Term						
HALLMARK_ADIPOGENESIS	0.298924	0.410395	0.387433	0.343606		
HALLMARK_ALLOGRAFT_REJECTION	0.158945	0.13835	0.110787	0.121643		
HALLMARK_ANDROGEN_RESPONSE	0.157449	0.162843	0.180475	0.181878		
HALLMARK_ANGIOGENESIS	0.032505	-0.024058	-0.039492	-0.043769		
HALLMARK_APICAL_JUNCTION	0.128808	0.095511	0.080076	0.098644		

[5 rows x 48 columns]

**Warning !!!**

if you set `permutation_num > 0`, `ssgsea` will become `prerank` with `ssGSEA` statistics. **DO NOT** use this, unless you known what you are doing !

```
ss_permut = gp.ssgsea(data="./tests/extdata/Leukemia_hgu95av2.trim.txt",
                      gene_sets="./tests/extdata/h.all.v7.0.symbols.gmt",
                      outdir=None,
                      sample_norm_method='rank', # choose 'custom' for your custom metric
                      permutation_num=20, # set permutation_num > 0, it will act like prerank.
                      ↪tool
                      no_plot=True, # skip plotting, because you don't need these figures
                      processes=4, seed=9)
ss_permut.res2d.head(5)
```

**Command line usage of ssGSEA**

```
[62]: # !gseapy ssgsea -d ./data/testSet_rand1200.gct \
#           -g data/temp.gmt \
#           -o test/ssgsea_report2 \
#           -p 4 --no-plot
```

## 5.3 GSVA example

```
[63]: import gseapy as gp
# txt, gct file input
es = gp.gsva(data='./tests/extdata/Leukemia_hgu95av2.trim.txt',
            gene_sets='./tests/extdata/h.all.v7.0.symbols.gmt',
            outdir=None)
```

2023-10-25 10:47:01,160 [WARNING] Found duplicated gene names, values averaged by gene\_↵names!

```
[64]: es.res2d.pivot(index='Term', columns='Name', values='ES').head()
```

```
[64]: Name          ALL_1    ALL_10    ALL_11    ALL_12  \
Term
HALLMARK_ADIPOGENESIS      -0.21331  -0.08096  0.003289 -0.017909
HALLMARK_ALLOGRAFT_REJECTION -0.210468 -0.373787 -0.086016 -0.169623
HALLMARK_ANDROGEN_RESPONSE   -0.13633 -0.308572  0.008126  0.04849
HALLMARK_ANGIOGENESIS        0.035895 -0.287645 -0.214951 -0.291145
HALLMARK_APICAL_JUNCTION    -0.088652 -0.128757 -0.050282 -0.248682

Name          ALL_13    ALL_14    ALL_15    ALL_16  \
Term
HALLMARK_ADIPOGENESIS       0.207841  0.023294 -0.085392 -0.221273
HALLMARK_ALLOGRAFT_REJECTION -0.158775 -0.016488 -0.050703  0.10443
HALLMARK_ANDROGEN_RESPONSE   -0.061181 -0.203036  0.070416 -0.12524
HALLMARK_ANGIOGENESIS        -0.311917 -0.236717 -0.345662 -0.250202
HALLMARK_APICAL_JUNCTION     -0.145164  0.001997 -0.082962 -0.091691

Name          ALL_17    ALL_18    ...    AML_22    AML_23  \
Term
HALLMARK_ADIPOGENESIS       0.16147  -0.01825  ...    0.03344 -0.190436
HALLMARK_ALLOGRAFT_REJECTION -0.075816 -0.193654  ...    0.023653  0.032892
HALLMARK_ANDROGEN_RESPONSE    0.080075  0.022248  ...    0.031898  0.064394
HALLMARK_ANGIOGENESIS        -0.233296 -0.318353  ...    0.244374 -0.076852
HALLMARK_APICAL_JUNCTION     -0.168941 -0.139766  ...    0.005859 -0.067385

Name          AML_24    AML_3    AML_4    AML_5  \
Term
HALLMARK_ADIPOGENESIS      -0.0985  0.105208  0.196799 -0.296305
HALLMARK_ALLOGRAFT_REJECTION -0.113577  0.30703  0.134581  0.188905
HALLMARK_ANDROGEN_RESPONSE    0.070232  0.199349 -0.079399 -0.016658
HALLMARK_ANGIOGENESIS        -0.010928 -0.210787  0.387912  0.269447
HALLMARK_APICAL_JUNCTION     0.062719 -0.022434  0.076593  0.138664

Name          AML_6    AML_7    AML_8    AML_9
Term
HALLMARK_ADIPOGENESIS      -0.084042  0.450832  0.226921  0.209835
HALLMARK_ALLOGRAFT_REJECTION 0.132169  0.024078 -0.092054 -0.195987
HALLMARK_ANDROGEN_RESPONSE   -0.127327  0.018847  0.121426  0.163149
HALLMARK_ANGIOGENESIS        0.34823  0.157249  0.075479 -0.064515
HALLMARK_APICAL_JUNCTION     0.240647  0.039307  0.016764  0.057512
```

(continues on next page)

(continued from previous page)

```
[5 rows x 48 columns]
```

```
[65]: # !gseapy ssgsea -d ./tests/data/expr.gsva.csv \  
#           -g ./tests/data/geneset.gsva.gmt \  
#           -o test/gsva_report
```

### 5.3.1 Replot Example

#### Locate your directory

Notes: replot module need to find edb folder to work properly. keep the file tree like this:

```
data  
|--- edb  
|   |--- C10E.cls  
|   |--- gene_sets.gmt  
|   |--- gsea_data.gsea_data.rnk  
|   |--- results.edb
```

```
[66]: # run command inside python console  
rep = gp.replot(indir="./tests/data", outdir="test/replot_test")
```

#### Command line usage of replot

```
[67]: # !gseapy replot -i data -o test/replot_test
```

## 5.4 scRNA-seq Example

Examples to use GSEAPy for scRNA-seq data

```
[1]: %load_ext autoreload  
%autoreload 2  
import os  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt
```

```
[2]: import gseapy as gp  
import scanpy as sc
```

```
[3]: gp.__version__
```

```
[3]: '1.0.5'
```

### 5.4.1 Read Demo Data

Convert demo data from seurat to scanpy

```
## R code
library(Seurat)
library(SeuratDisk)
ifnb = SeuratData::LoadData("ifnb")
SaveH5Seurat(ifnb, "ifnb.h5seurat", overwrite = T)
Convert("ifnb.h5seurat", "ifnb.h5ad", overwrite = T)
```

```
[4]: adata = sc.read_h5ad("data/ifnb.h5ad") # data from SeuratData::ifnb
```

```
[5]: adata.obs.head()
```

```
[5]:
```

	orig.ident	nCount_RNA	nFeature_RNA	stim	\
AAACATACATTTC.1	IMMUNE_CTRL	3017.0	877	CTRL	
AAACATACCAGAAA.1	IMMUNE_CTRL	2481.0	713	CTRL	
AAACATACCTCGCT.1	IMMUNE_CTRL	3420.0	850	CTRL	
AAACATACCTGGTA.1	IMMUNE_CTRL	3156.0	1109	CTRL	
AAACATACGATGAA.1	IMMUNE_CTRL	1868.0	634	CTRL	

```
seurat_annotations
```

AAACATACATTTC.1	CD14 Mono
AAACATACCAGAAA.1	CD14 Mono
AAACATACCTCGCT.1	CD14 Mono
AAACATACCTGGTA.1	pDC
AAACATACGATGAA.1	CD4 Memory T

```
[6]: adata.layers['counts'] = adata.X # Save raw counts
```

```
[7]: # preprocessing
sc.pp.normalize_total(adata, target_sum=1e4)
sc.pp.log1p(adata)
adata.layers['lognorm'] = adata.X
```

```
[8]: adata.obs.groupby('seurat_annotations')['stim'].value_counts()
```

```
[8]: seurat_annotations stim
```

B	STIM	571
	CTRL	407
B Activated	STIM	203
	CTRL	185
CD14 Mono	CTRL	2215
	STIM	2147
CD16 Mono	STIM	537
	CTRL	507
CD4 Memory T	STIM	903
	CTRL	859
CD4 Naive T	STIM	1526
	CTRL	978
CD8 T	STIM	462
	CTRL	352

(continues on next page)

(continued from previous page)

DC	CTRL	258
	STIM	214
Eryth	STIM	32
	CTRL	23
Mk	STIM	121
	CTRL	115
NK	STIM	321
	CTRL	298
T activated	STIM	333
	CTRL	300
pDC	STIM	81
	CTRL	51

Name: stim, dtype: int64

```
[9]: # set STIM as class 0, CTRL as class 1, to make categorical
adata.obs['stim'] = pd.Categorical(adata.obs['stim'], categories=["STIM", "CTRL"],
    ordered=True)
indices = adata.obs.sort_values(['seurat_annotatons', 'stim']).index
adata = adata[indices,:]
```

```
[10]: # # # subset and write GCT and CLS file
# outdir = "ifnb/"
# for cell in adata.obs.seurat_annotatons.unique():
#     bdata = adata[adata.obs.seurat_annotatons == cell ]
#     groups = bdata.obs['stim'].to_list()
#     cls_dict = bdata.obs['stim'].to_dict()
#     gs = bdata.to_df().T
#     gs.index.name = "NAME"

#     gs_std = gs.groupby(by=cls_dict, axis=1).std()
#     gs = gs[gs_std.sum(axis=1) > 0]
#     gs = gs + 1e-08 # we don't like zeros!!!

#     gs.insert(0, column="Description", value=cell,)
#     outname = os.path.join( outdir, cell + ".gct")
#     outcls = os.path.join(outdir, cell + ".cls")
#     s_len = gs.shape[1] - 1
#     with open(outname,"w") as correct:
#         line1="#1.2\n"+f"{gs.shape[0]}\t{s_len}\n"
#         correct.write(line1)
#         gs.to_csv(correct, sep="\t")

#     with open(outcls, "w") as cl:
#         line = f"{len(groups)} 2 1\n# STIM CTRL\n"
#         cl.write(line)
#         cl.write(" ".join(groups) + "\n")
#     print(outname)
```

```
[11]: # subset data
bdata = adata[adata.obs.seurat_annotatons == "CD14 Mono"].copy()
bdata
```

```
[11]: AnnData object with n_obs × n_vars = 4362 × 14053
      obs: 'orig.ident', 'nCount_RNA', 'nFeature_RNA', 'stim', 'seurat_annotations'
      var: 'features'
      uns: 'log1p'
      layers: 'counts', 'lognorm'
```

## 5.4.2 GSEA

```
[12]: import time
      t1 = time.time()
      # NOTE: To speed up, use gp.prerank instead with your own ranked list.
      res = gp.gsea(data=bdata.to_df().T, # row -> genes, column-> samples
                    gene_sets="GO_Biological_Process_2021",
                    cls=bdata.obs.stim,
                    permutation_num=1000,
                    permutation_type='phenotype',
                    outdir=None,
                    method='s2n', # signal_to_noise
                    threads= 16)
      t2=time.time()
      print(t2-t1)

      60.238986015319824
```

```
[13]: res.res2d.head(10)
```

```
[13]:
```

	Name	Term	ES	\
0	gsea	cytokine-mediated signaling pathway (GO:0019221)	0.685491	
1	gsea	innate immune response (GO:0045087)	0.784391	
2	gsea	regulation of immune response (GO:0050776)	0.759354	
3	gsea	defense response to virus (GO:0051607)	0.903464	
4	gsea	response to cytokine (GO:0034097)	0.718931	
5	gsea	defense response to symbiont (GO:0140546)	0.904717	
6	gsea	cellular response to interferon-gamma (GO:0071...	0.792726	
7	gsea	regulation of interferon-beta production (GO:0...	0.856704	
8	gsea	RNA splicing, via transesterification reaction...	-0.626583	
9	gsea	gene expression (GO:0010467)	-0.70455	

	NES	NOM	p-val	FDR	q-val	FWER	p-val	Tag %	Gene %	\
0	3.759972		0.0		0.0	0.0	0.0	140/490	9.03%	
1	3.66143		0.0		0.0	0.0	0.0	56/188	6.30%	
2	3.549856		0.0		0.0	0.0	0.0	49/140	8.77%	
3	3.438759		0.0		0.0	0.0	0.0	42/108	2.85%	
4	3.37735		0.0		0.0	0.0	0.0	37/120	7.26%	
5	3.362051		0.0		0.0	0.0	0.0	49/100	4.90%	
6	3.327923		0.0		0.0	0.0	0.0	49/99	7.18%	
7	3.259412		0.0		0.0	0.0	0.0	14/44	4.94%	
8	-3.225436		0.0		0.0	0.0	0.0	128/234	19.45%	
9	-3.219153		0.0		0.0	0.0	0.0	134/322	10.13%	

```

                                Lead_genes
0  ISG15; IFIT3; IFIT1; RSAD2; ISG20; CXCL10; IFITM3; CX...
```

(continues on next page)

(continued from previous page)

```

1 ISG15;IFIT1;CXCL10;IFITM3;APOBEC3A;MX1;IFI6;OA...
2 RSAD2;IRF7;PLSCR1;HERC5;IL4I1;SLAMF7;IFITM1;HL...
3 ISG15;IFIT3;IFIT1;RSAD2;ISG20;CXCL10;IFITM3;AP...
4 ISG15;IFITM3;MX1;IFITM2;PLSCR1;MX2;BST2;EIF2AK...
5 ISG15;IFIT3;IFIT1;RSAD2;ISG20;IFITM3;APOBEC3A;...
6 CCL8;OAS1;MT2A;OASL;IRF7;GBP1;GBP4;CCL2;OAS3;O...
7 ISG15;OAS1;IRF7;DDX58;IFIH1;OAS3;OAS2;DHX58;HS...
8 YBX1;PABPC1;HNRNPA1;DDX5;SRSF9;HNRNPM;RBMX;SF3...
9 RPL6;RPL7;RPL15;RPL10;RPS3A;RPS6;RPL8;RPL21;RP...

```

```
[14]: res.ranking.shape # raking metric
```

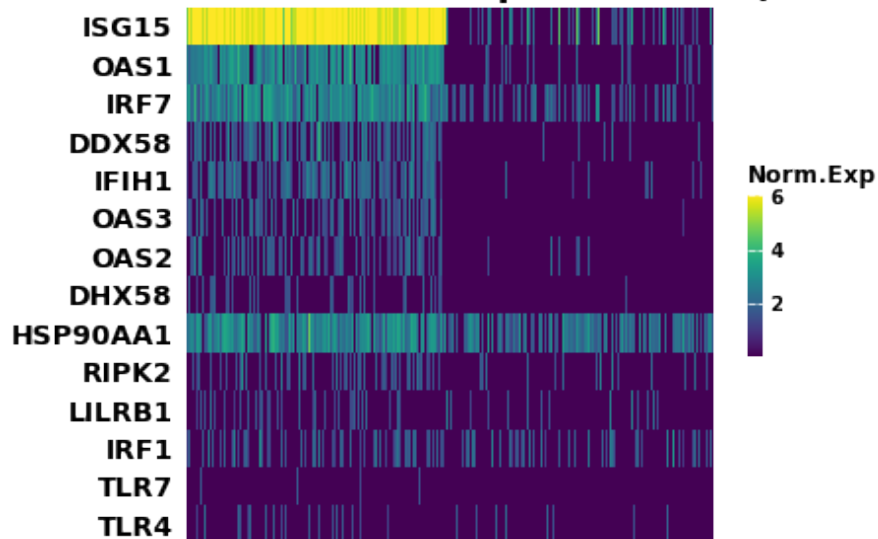
```
[14]: (13216,)
```

```

[15]: ## Heatmap of gene expression
i = 7
genes = res.res2d.Lead_genes.iloc[i].split(";")
ax = gp.heatmap(df = res.heatmat.loc[genes],
                z_score=None,
                title=res.res2d.Term.iloc[i],
                figsize=(6,5),
                cmap=plt.cm.viridis,
                xticklabels=False)

```

### regulation of interferon-beta production (GO:0032648)

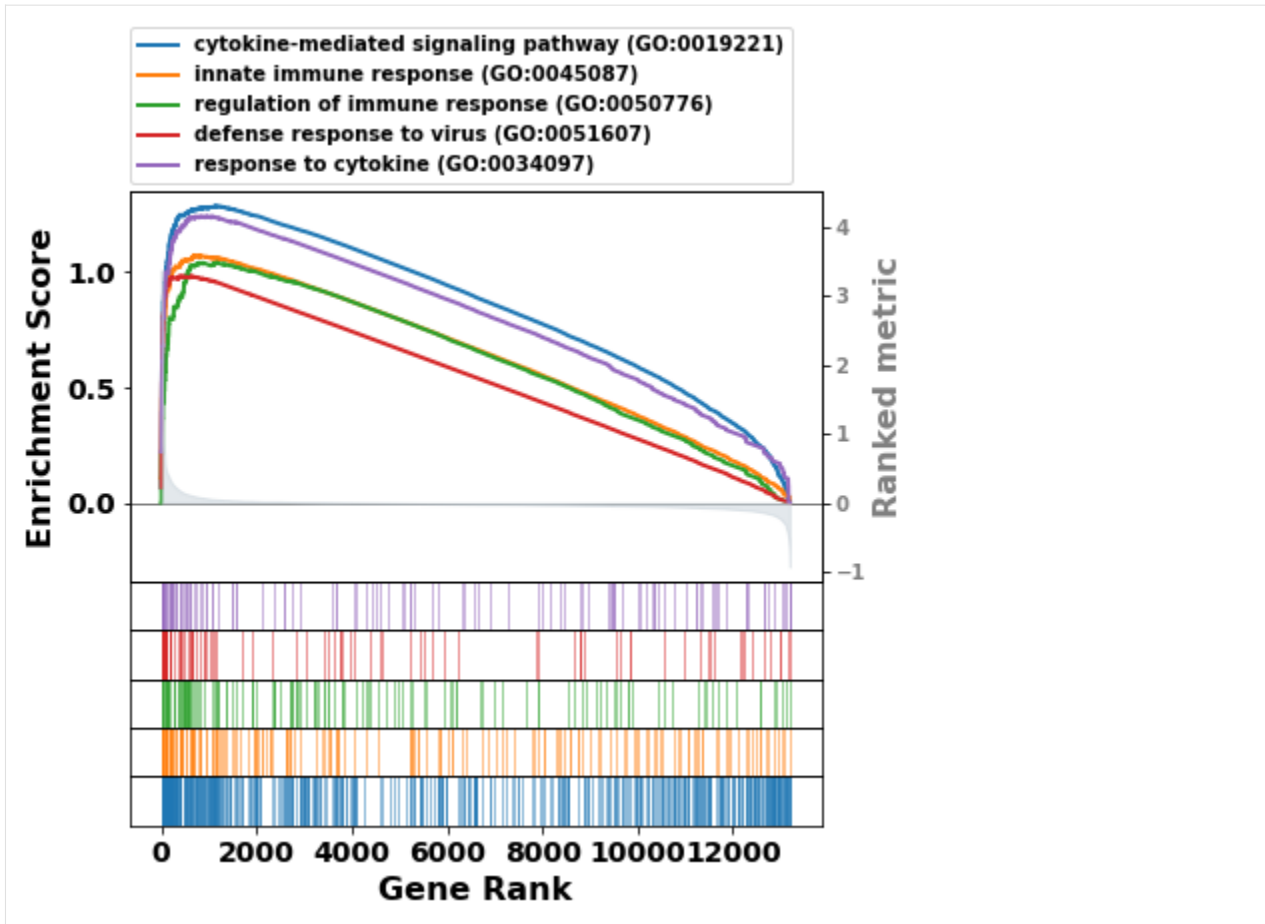


```

[16]: term = res.res2d.Term
# gp.gseaplot(res.ranking, term=term[i], **res.results[term[i]])
axs = res.plot(terms=term[:5])

```





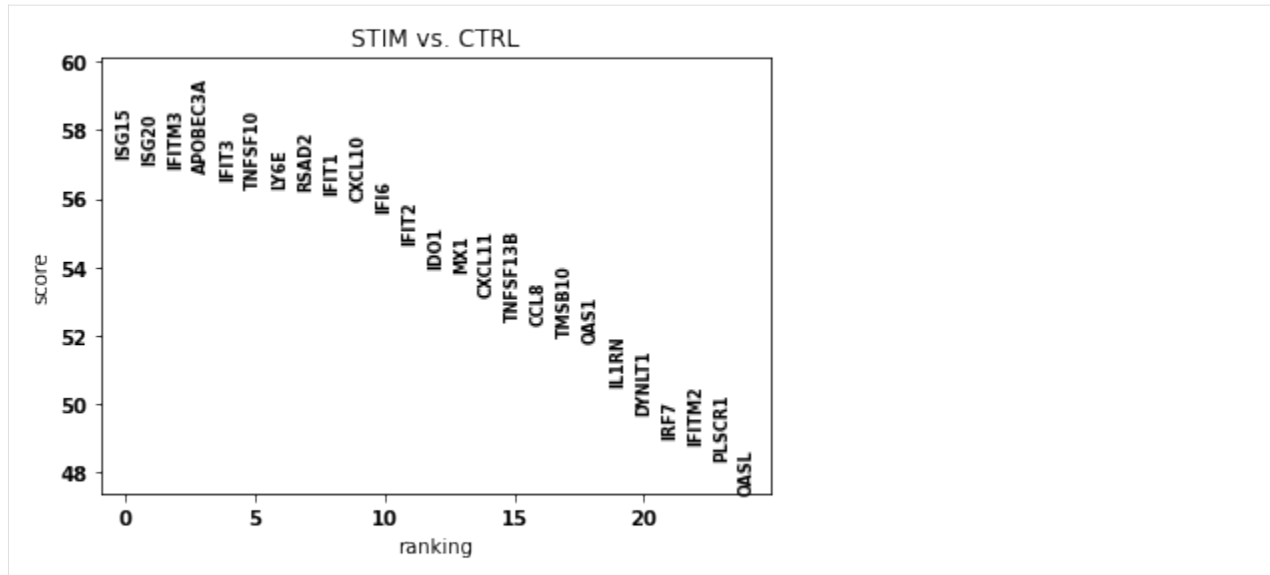
### 5.4.3 DEG Analysis

```
[17]: # find degs
sc.tl.rank_genes_groups(bdata,
                        groupby='stim',
                        use_raw=False,
                        layer='lognorm',
                        method='wilcoxon',
                        groups=["STIM"],
                        reference='CTRL')
```

```
[18]: bdata.X.max() # already log1p
```

```
[18]: 8.065909516515664
```

```
[19]: sc.pl.rank_genes_groups(bdata, n_genes=25, sharey=False)
```



```
[20]: # get deg result
result = bdata.uns['rank_genes_groups']
groups = result['names'].dtype.names
degs = pd.DataFrame(
    {group + '_' + key: result[key][group]
     for group in groups for key in ['names', 'scores', 'pvals', 'pvals_adj', 'logfoldchanges',
     ↪ ]})
```

```
[21]: degs.head()
```

	STIM_names	STIM_scores	STIM_pvals	STIM_pvals_adj	STIM_logfoldchanges
0	ISG15	57.165920	0.0	0.0	8.660480
1	ISG20	57.010372	0.0	0.0	6.850681
2	IFITM3	56.890392	0.0	0.0	6.320490
3	APOBEC3A	56.770397	0.0	0.0	6.616682
4	IFIT3	56.569122	0.0	0.0	8.313443

```
[22]: degs.shape
```

```
[22]: (14053, 5)
```

## 5.4.4 Over-representation analysis (Enrichr API)

```
[23]: # subset up or down regulated genes
degs_sig = degs[degs.STIM_pvals_adj < 0.05]
degs_up = degs_sig[degs_sig.STIM_logfoldchanges > 0]
degs_dw = degs_sig[degs_sig.STIM_logfoldchanges < 0]
```

```
[24]: degs_up.shape
```

```
[24]: (687, 5)
```

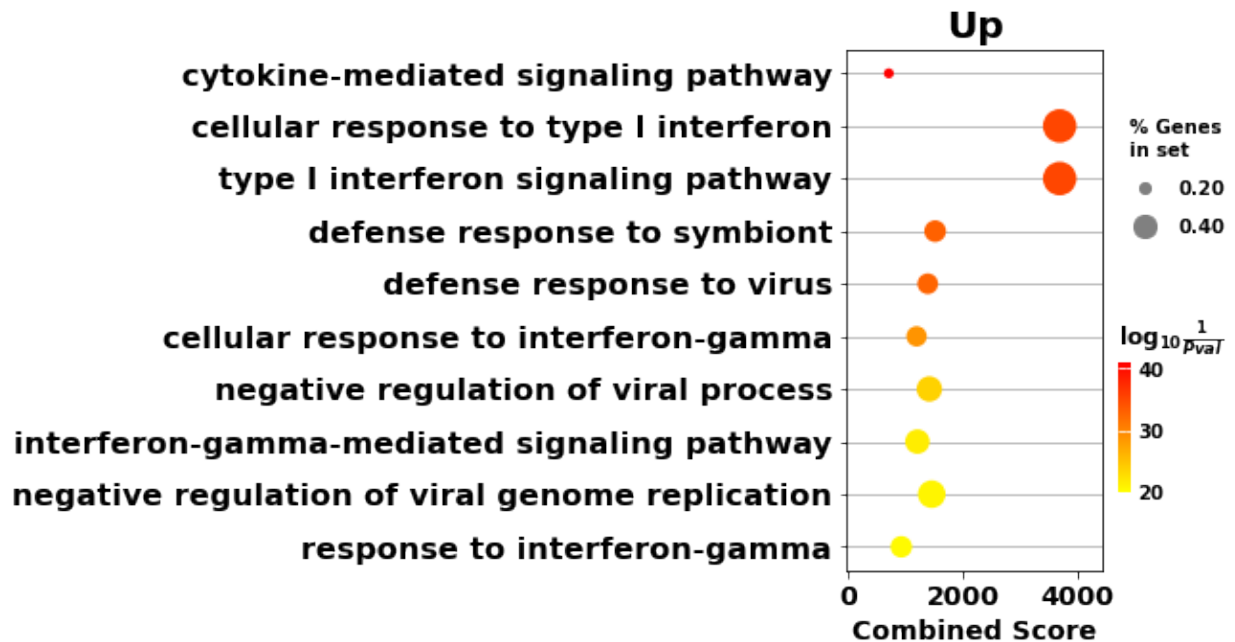
```
[25]: degs_dw.shape
```

```
[25]: (1030, 5)
```

```
[26]: # Enrichr API
enr_up = gp.enrichr(degs_up.STIM_names,
                    gene_sets='GO_Biological_Process_2021',
                    outdir=None)
```

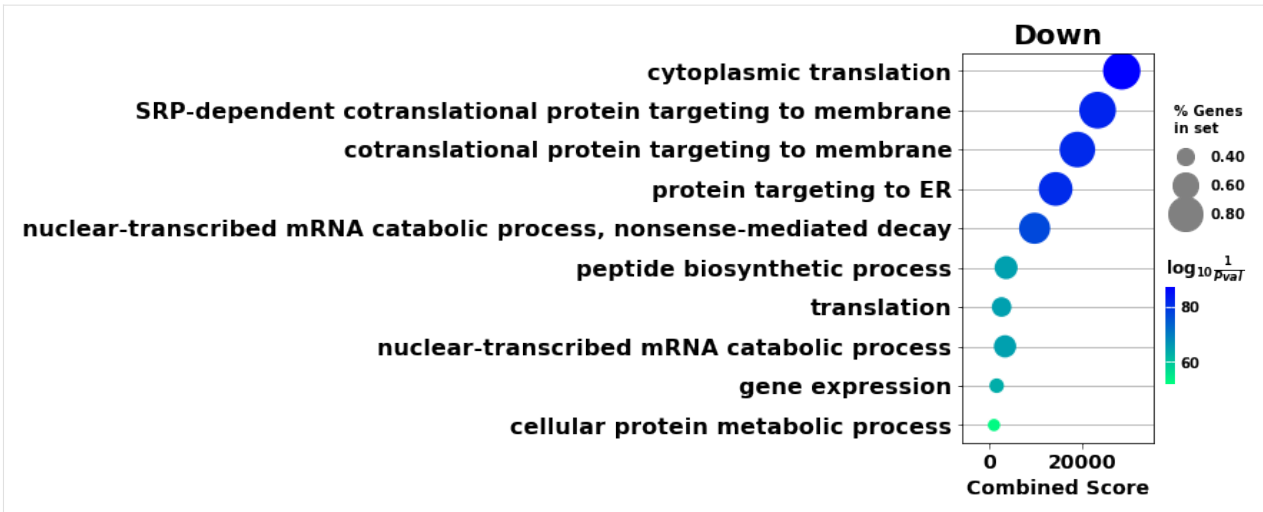
```
[27]: # trim (go:...)
enr_up.res2d.Term = enr_up.res2d.Term.str.split(" \((GO)").str[0]
```

```
[28]: # dotplot
gp.dotplot(enr_up.res2d, figsize=(3,5), title="Up", cmap = plt.cm.autumn_r)
plt.show()
```



```
[29]: enr_dw = gp.enrichr(degs_dw.STIM_names,
                        gene_sets='GO_Biological_Process_2021',
                        outdir=None)
```

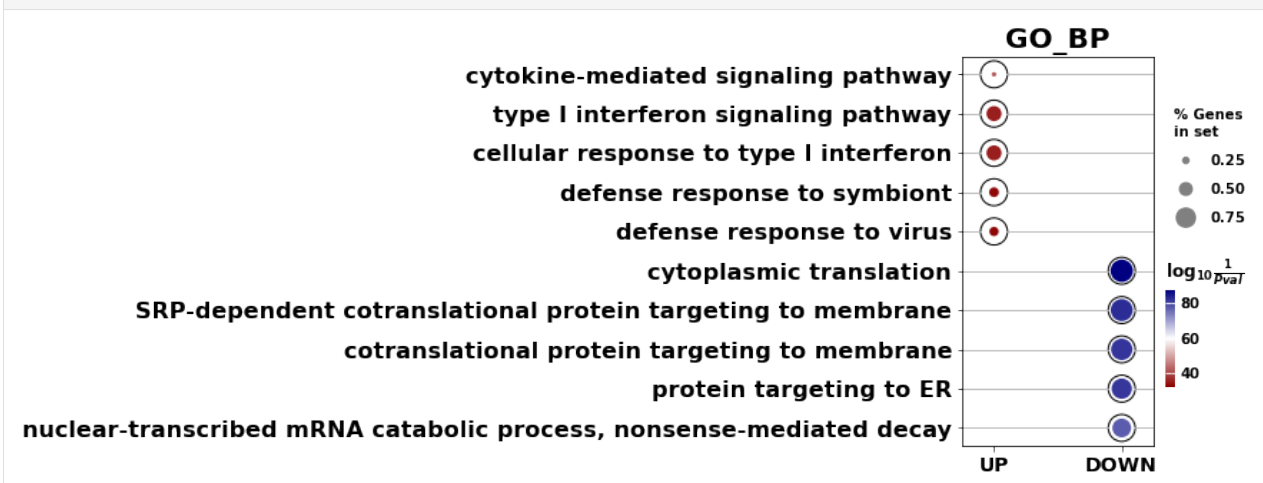
```
[30]: enr_dw.res2d.Term = enr_dw.res2d.Term.str.split(" \((GO)").str[0]
gp.dotplot(enr_dw.res2d,
            figsize=(3,5),
            title="Down",
            cmap = plt.cm.winter_r,
            size=5)
plt.show()
```



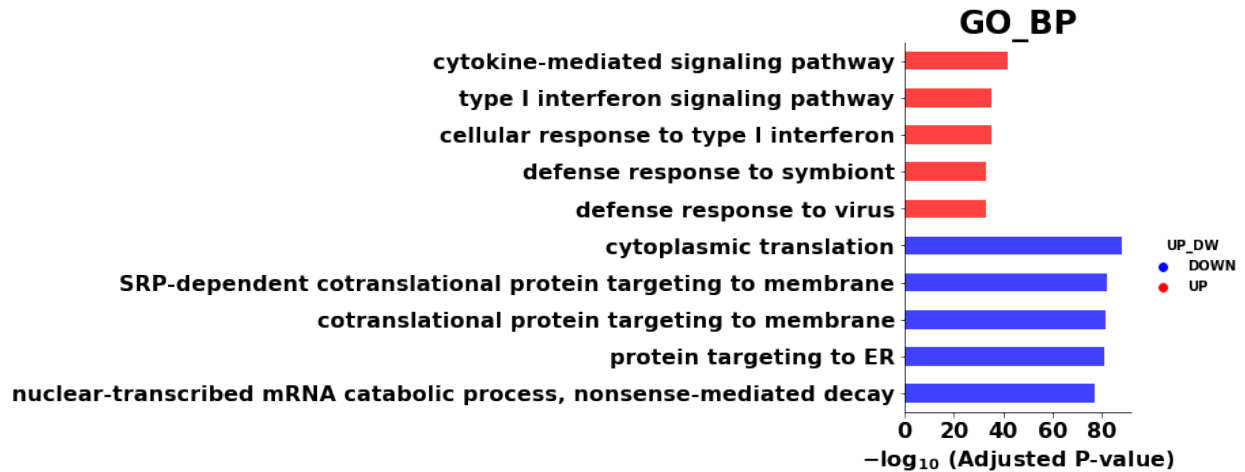
```
[31]: # concat results
enr_up.res2d['UP_DW'] = "UP"
enr_dw.res2d['UP_DW'] = "DOWN"
enr_res = pd.concat([enr_up.res2d.head(), enr_dw.res2d.head()])
```

```
[32]: from gseapy.scipalette import SciPalette
sci = SciPalette()
NbDr = sci.create_colormap()
# NbDr
```

```
[33]: # display multi-datasets
ax = gp.dotplot(enr_res, figsize=(3,5),
               x='UP_DW',
               x_order = ["UP", "DOWN"],
               title="GO_BP",
               cmap = NbDr.reversed(),
               size=3,
               show_ring=True)
ax.set_xlabel("")
plt.show()
```



```
[34]: ax = gp.barplot(enr_res, figsize=(3,5),
                    group = 'UP_DW',
                    title = "GO_BP",
                    color = ['b', 'r'])
```



## 5.4.5 Network Visualization

```
[35]: import networkx as nx
```

```
[36]: res.res2d.head()
```

```
[36]:
```

	Name	Term	ES	NES	\
0	gsea	cytokine-mediated signaling pathway (GO:0019221)	0.685491	3.759972	
1	gsea	innate immune response (GO:0045087)	0.784391	3.66143	
2	gsea	regulation of immune response (GO:0050776)	0.759354	3.549856	
3	gsea	defense response to virus (GO:0051607)	0.903464	3.438759	
4	gsea	response to cytokine (GO:0034097)	0.718931	3.37735	

	NOM p-val	FDR q-val	FWER p-val	Tag %	Gene %	\
0	0.0	0.0	0.0	140/490	9.03%	
1	0.0	0.0	0.0	56/188	6.30%	
2	0.0	0.0	0.0	49/140	8.77%	
3	0.0	0.0	0.0	42/108	2.85%	
4	0.0	0.0	0.0	37/120	7.26%	

	Lead_genes
0	ISG15; IFIT3; IFIT1; RSAD2; ISG20; CXCL10; IFITM3; CX...
1	ISG15; IFIT1; CXCL10; IFITM3; APOBEC3A; MX1; IFI6; OA...
2	RSAD2; IRF7; PLSCR1; HERC5; IL4I1; SLAMF7; IFITM1; HL...
3	ISG15; IFIT3; IFIT1; RSAD2; ISG20; CXCL10; IFITM3; AP...
4	ISG15; IFITM3; MX1; IFITM2; PLSCR1; MX2; BST2; EIF2AK...

```
[37]: # res.res2d.to_csv("data/test.out.txt", sep="\t", index=False)
```

```
[38]: nodes, edges = gp.enrichment_map(res.res2d)
```

[39]: nodes.head()

```

[39]:
      Name                                     Term      ES  \
node_idx
0      gsea                                gene expression (GO:0010467) -0.70455
1      gsea  RNA splicing, via transesterification reaction... -0.626583
2      gsea  regulation of interferon-beta production (GO:0...  0.856704
3      gsea  cellular response to interferon-gamma (GO:0071...  0.792726
4      gsea                defense response to symbiont (GO:0140546)  0.904717

      NES NOM p-val FDR q-val FWER p-val   Tag %  Gene %  \
node_idx
0      -3.219153      0.0  0.000009      0.0  134/322  10.13%
1      -3.225436      0.0  0.000009      0.0  128/234  19.45%
2       3.259412      0.0  0.000009      0.0   14/44   4.94%
3       3.327923      0.0  0.000009      0.0   49/99   7.18%
4       3.362051      0.0  0.000009      0.0   49/100   4.90%

      Lead_genes      p_inv  \
node_idx
0  RPL6;RPL7;RPL15;RPL10;RPS3A;RPS6;RPL8;RPL21;RP...  5.061359
1  YBX1;PABPC1;HNRNPA1;DDX5;SRSF9;HNRNPM;RBMX;SF3...  5.061359
2  ISG15;OAS1;IRF7;DDX58;IFIH1;OAS3;OAS2;DHX58;HS...  5.061359
3  CCL8;OAS1;MT2A;OASL;IRF7;GBP1;GBP4;CCL2;OAS3;O...  5.061359
4  ISG15;IFIT3;IFIT1;RSAD2;ISG20;IFITM3;APOBEC3A;...  5.061359

      Hits_ratio
node_idx
0      0.416149
1      0.547009
2      0.318182
3      0.494949
4      0.490000

```

[40]: edges.head()

```

[40]:
      src_idx  targ_idx      src_name  \
0          0          1      gene expression (GO:0010467)
1          0          8      gene expression (GO:0010467)
2          1          8  RNA splicing, via transesterification reaction...
3          2          3  regulation of interferon-beta production (GO:0...
4          2          4  regulation of interferon-beta production (GO:0...

      targ_name  jaccard_coef  \
0  RNA splicing, via transesterification reaction...  0.110169
1  cellular macromolecule biosynthetic process (G...  0.645390
2  cellular macromolecule biosynthetic process (G...  0.022624
3  cellular response to interferon-gamma (GO:0071...  0.105263
4      defense response to symbiont (GO:0140546)  0.188679

      overlap_coef      overlap_genes
0      0.203125  POLR2F,RBM8A,HNRNPK,POLR2E,SYNCRIP,SRSF6,SRRM1...
1      0.928571  RPL26,RPL36,RPS2,RPL41,POLR2J,RPL24,RPL29,RPS2...

```

(continues on next page)

(continued from previous page)

```

2      0.051020      POLR2J,POLR2G,POLR2F,POLR2E,POLR2L
3      0.428571      IRF1,OAS2,OAS1,TLR4,OAS3,IRF7
4      0.714286      IRF1,TLR7,DDX58,OAS2,ISG15,LILRB1,OAS1,IFIH1,0...
```

```

[41]: # build graph
G = nx.from_pandas_edgelist(edges,
                           source='src_idx',
                           target='targ_idx',
                           edge_attr=['jaccard_coef', 'overlap_coef', 'overlap_genes'])
```

```

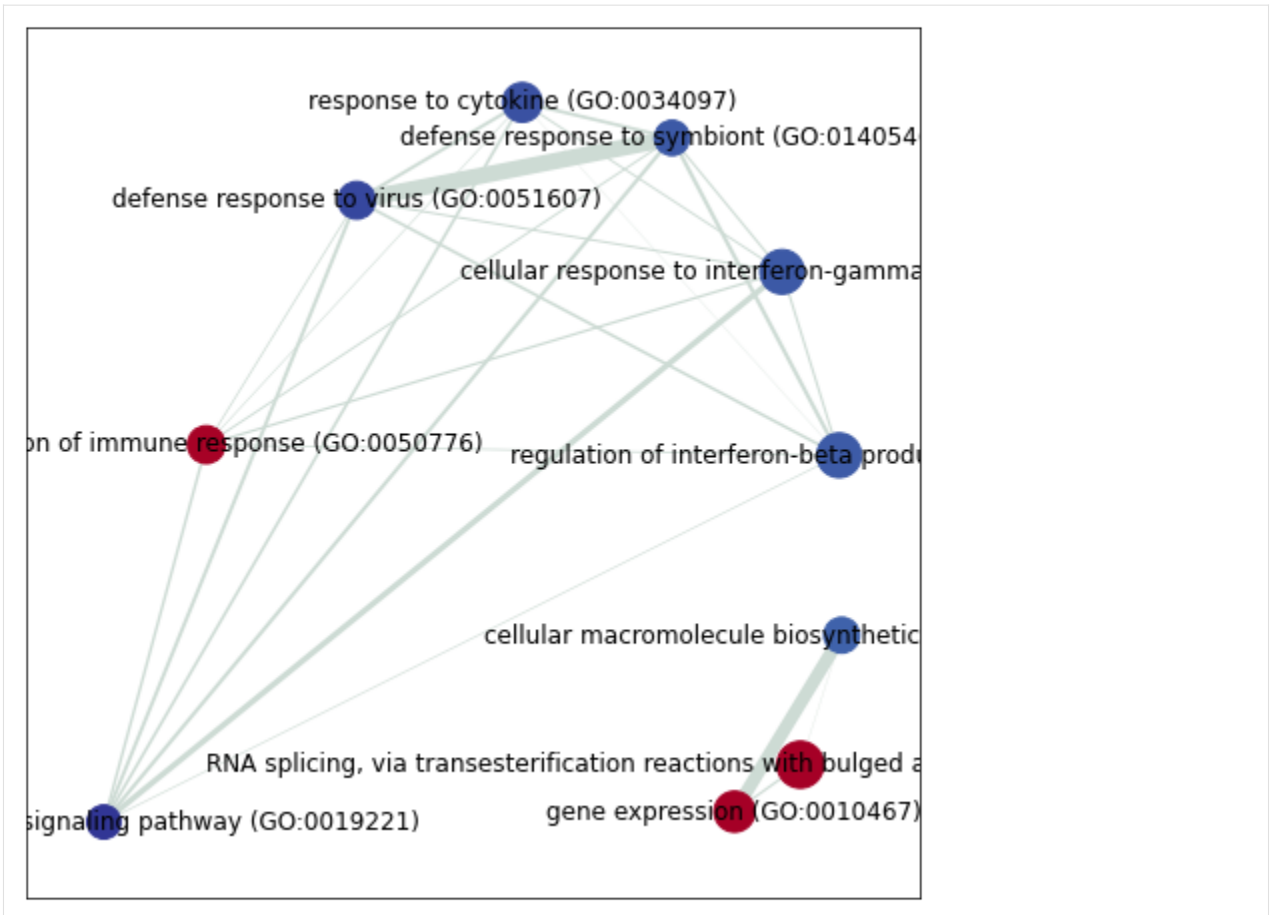
[42]: fig, ax = plt.subplots(figsize=(8, 8))

# init node coordinates
pos=nx.layout.spiral_layout(G)
#node_size = nx.get_node_attributes()
# draw node
nx.draw_networkx_nodes(G,
                       pos=pos,
                       cmap=plt.cm.RdYlBu,
                       node_color=list(nodes.NES),
                       node_size=list(nodes.Hits_ratio *1000))

# draw node label
nx.draw_networkx_labels(G,
                       pos=pos,
                       labels=nodes.Term.to_dict())

# draw edge
edge_weight = nx.get_edge_attributes(G, 'jaccard_coef').values()
nx.draw_networkx_edges(G,
                       pos=pos,
                       width=list(map(lambda x: x*10, edge_weight)),
                       edge_color='#CDDBD4')

plt.show()
```



[ ]:

## 5.5 A Protocol to Prepare files for GSEAPy

As a biological researcher, I like protocols.

Here is a short tutorial for you to walk you through gseapy.

For file format explanation, please see [here](#)

In order to run gseapy successfully, install gseapy use pip.

```
pip install gseapy
```

```
# if you have conda
```

```
conda install -c bioconda gseapy
```



### 5.5.1 Use gsea command, or gsea()

Follow the steps blow.

One thing you should know is that the gseapy input files are the same as GSEA desktop required. You can use these files below to run GSEA desktop, too.

#### Prepare an tabular text file of gene expression like this:

RNA-seq,ChIP-seq, Microarry data are all supported.

Here is to see what the structure of expression table looks like

```
import pandas as pd
df = pd.read_table('./test/gsea_data.txt')
df.head()

#or assign dataframe to the parameter 'data'
```

#### An cls file is also expected.

This file is used to specify column attributes in step 1, just like GSEA asked.

An example of cls file looks like below.

```
with open('gsea/edb/C10E.cls') as cls:
    print(cls.read())

# or assign a list object to parameter 'cls' like this
# cls=['C10E', 'C10E', 'C10E', 'Vector', 'Vector', 'Vector']
```

```
6 2 1
# C10E Vector
C10E C10E C10E Vector Vector Vector
```

The first line specify the total samples and phenotype numbers. Leave number 1 always be 1.

The second line specify the phenotype class(name).

The third line specify column attributes in step 1.

So you could prepare the cls file in python like this

```
groups = ['C10E', 'C10E', 'C10E', 'Vector', 'Vector', 'Vector']
with open('gsea/edb/C10E.cls', "w") as cl:
    line = f"{len(groups)} 2 1\n# C10E Vector\n"
    cl.write(line)
    cl.write(" ".join(groups) + "\n")
```

### Gene\_sets file in gmt format.

All you need to do is to download gene set database file from GSEA or Enrichr website.

Or you could use enrichr library. In this case, just provide library name to parameter 'gene\_sets'

If you would like to use you own gene\_sets.gmts files, build such a file use excel:

An example of gmt file looks like below:

```
with open('gsea/edb/gene_sets.gmt') as gmt:
    print(gmt.read())
```

ES-SPECIFIC	Arid3a_used	ACTA1	CALML4	CORO1A	DHX58	DPYS	EGR1	ESRRB	GLI2
→ GPX2	HCK	INHBB							
HDAC-UNIQUE	Arid3a_used	1700017B05RIK	8430427H17RIK	ABCA3	ANKRD44	ARL4A	BNC2		
→ CLDN3									
XEN-SPECIFIC	Arid3a_used	1110036003RIK	A130022J15RIK	B2M	B3GALNT1				
→ CBX4	CITED1	CLU	CTSH	CYP26A1					
GATA-SPECIFIC	Arid3a_used	1200009I06RIK	5430407P10RIK	BAIAP2L1					
→ BMP8B	CITED1	CLDN3	COBLL1	CORO1A	CRYAB	CTDSPL	DKKL1		
TS-SPECIFIC	Arid3a_used	5430407P10RIK	AFAP1L1	AHNAK	ANXA2	ANXA3	ANXA5	B2M	
→ BIK	BMP8B	CAMK1D	CBX4	CLDN3	CSRP1	DKKL1	DSC2		

### 5.5.2 Use enrichr command, or enrichr()

The only thing you need to prepare is a gene list file.

**Note:** Enrichr uses a list of Entrez gene symbols as input.

For enrichr , you could assign a list object

```
# assign a list object to enrichr
l = ['SCARA3', 'LOC100044683', 'CMBL', 'CLIC6', 'IL13RA1', 'TACSTD2', 'DKKL1', 'CSF1',
     'SYNPO2L', 'TINAGL1', 'PTX3', 'BGN', 'HERC1', 'EFNA1', 'CIB2', 'PMP22', 'TMEM173']

gseapy.enrichr(gene_list=l, gene_sets='KEGG_2016', outfile='test')
```

or a gene list file in txt format(one gene id per row)

```
gseapy.enrichr(gene_list='gene_list.txt', gene_sets='KEGG_2016', outfile='test')
```

Let's see what the txt file looks like.

```
with open('data/gene_list.txt') as genes:
    print(genes.read())
```

```
CTLA2B
SCARA3
LOC100044683
CMBL
CLIC6
IL13RA1
TACSTD2
```

(continues on next page)

(continued from previous page)

```
DKKL1
CSF1
CITED1
SYNP02L
TINAGL1
PTX3
```

Select the library you want to do enrichment analysis. To get a list of all available libraries, run

```
#s get_library_name(), it will print out all library names.
import gseapy
names = gseapy.get_library_name()
print(names)
```

```
['Genome_Browser_PWMs',
'TRANSFAC_and_JASPAR_PWMs',
'ChEA_2013',
'Drug_Perturbations_from_GEO_2014',
'ENCODE_TF_ChIP-seq_2014',
'BioCarta_2013',
'Reactome_2013',
'WikiPathways_2013',
'Disease_Signatures_from_GEO_up_2014',
'KEGG_2013',
'TF-LOF_Expression_from_GEO',
'TargetScan_microRNA',
'PPI_Hub_Proteins',
'GO_Molecular_Function_2015',
'GeneSigDB',
'Chromosome_Location',
'Human_Gene_Atlas',
'Mouse_Gene_Atlas',
'GO_Cellular_Component_2015',
'GO_Biological_Process_2015',
'Human_Phenotype_Ontology',
'Epigenomics_Roadmap_HM_ChIP-seq',
'KEA_2013',
'NURSA_Human_Endogenous_Complexome',
'CORUM',
'SILAC_Phosphoproteomics',
'MGI_Mammalian_Phenotype_Level_3',
'MGI_Mammalian_Phenotype_Level_4',
'Old_CMAP_up',
'Old_CMAP_down',
'OMIM_Disease',
'OMIM_Expanded',
'VirusMINT',
'MSigDB_Computational',
'MSigDB_Oncogenic_Signatures',
'Disease_Signatures_from_GEO_down_2014',
'Virus_Perturbations_from_GEO_up',
'Virus_Perturbations_from_GEO_down',
```

(continues on next page)

(continued from previous page)

```

'Cancer_Cell_Line_Encyclopedia',
'NCI-60_Cancer_Cell_Lines',
'Tissue_Protein_Expression_from_ProteomicsDB',
'Tissue_Protein_Expression_from_Human_Proteome_Map',
'HMDB_Metabolites',
'Pfam_InterPro_Domains',
'GO_Biological_Process_2013',
'GO_Cellular_Component_2013',
'GO_Molecular_Function_2013',
'Allen_Brain_Atlas_up',
'ENCODE_TF_ChIP-seq_2015',
'ENCODE_Histone_Modifications_2015',
'Phosphatase_Substrates_from_DEPOD',
'Allen_Brain_Atlas_down',
'ENCODE_Histone_Modifications_2013',
'Achilles_fitness_increase',
'Achilles_fitness_decrease',
'MGI_Mammalian_Phenotype_2013',
'BioCarta_2015',
'HumanCyc_2015',
'KEGG_2015',
'NCI-Nature_2015',
'Panther_2015',
'WikiPathways_2015',
'Reactome_2015',
'ESCAPE',
'HomoloGene',
'Disease_Perturbations_from_GEO_down',
'Disease_Perturbations_from_GEO_up',
'Drug_Perturbations_from_GEO_down',
'Genes_Associated_with_NIH_Grants',
'Drug_Perturbations_from_GEO_up',
'KEA_2015',
'Single_Gene_Perturbations_from_GEO_up',
'Single_Gene_Perturbations_from_GEO_down',
'ChEA_2015',
'dbGaP',
'LINCS_L1000_Chem_Pert_up',
'LINCS_L1000_Chem_Pert_down',
'GTEx_Tissue_Sample_Gene_Expression_Profiles_down',
'GTEx_Tissue_Sample_Gene_Expression_Profiles_up',
'Ligand_Perturbations_from_GEO_down',
'Aging_Perturbations_from_GEO_down',
'Aging_Perturbations_from_GEO_up',
'Ligand_Perturbations_from_GEO_up',
'MCF7_Perturbations_from_GEO_down',
'MCF7_Perturbations_from_GEO_up',
'Microbe_Perturbations_from_GEO_down',
'Microbe_Perturbations_from_GEO_up',
'LINCS_L1000_Ligand_Perturbations_down',
'LINCS_L1000_Ligand_Perturbations_up',
'LINCS_L1000_Kinase_Perturbations_down',

```

(continues on next page)

(continued from previous page)

```
'LINCS_L1000_Kinase_Perturbations_up',
'Reactome_2016',
'KEGG_2016',
'WikiPathways_2016',
'ENCODE_and_ChEA_Consensus_TFs_from_ChIP-X',
'Kinase_Perturbations_from_GEO_down',
'Kinase_Perturbations_from_GEO_up',
'BioCarta_2016',
'Humancyc_2016',
'NCI-Nature_2016',
'Panther_2016']
```

For more details, please track the official links: <http://amp.pharm.mssm.edu/Enrichr/>

### 5.5.3 Use replot Command, or replot()

You may also want to use `replot()` to reproduce GSEA desktop plots.

The only input of `replot()` is the directory of GSEA desktop output.

The input directory(e.g. `gsea`), must contained **edb** folder, gseapy need 4 data files inside edb folder. The `gsea` document tree looks like this:

```
gsea
├── edb
│   ├── test.cls
│   ├── gene_sets.gmt
│   ├── gsea_data.rnk
│   └── results.edb
```

After this, you can start to run `gseapy`.

```
import gseapy
gseapy.replot(indir='gsea', outdir='gseapy_out')
```

If you prefer to run in command line, it's more simple.

```
gseapy replot -i gsea -o gseapy_out
```

For advanced usage of library, see the *Developmental Guide*.

## 5.6 Developmental Guide

### 5.6.1 Module APIs

`gseapy.gsea()`

Run Gene Set Enrichment Analysis.

**Parameters**

- **data** – Gene expression data table, Pandas DataFrame, gct file.
- **gene\_sets** – Enrichr Library name or .gmt gene sets file or dict of gene sets. Same input with GSEA.
- **cls** – A list or a .cls file format required for GSEA.
- **outdir** (*str*) – Results output directory. If None, nothing will write to disk.
- **permutation\_num** (*int*) – Number of permutations. Default: 1000. Minimal possible nominal p-value is about 1/nperm.
- **permutation\_type** (*str*) – Type of permutation reshuffling, choose from {"phenotype": 'sample.labels', "gene\_set": gene.labels}.
- **min\_size** (*int*) – Minimum allowed number of genes from gene set also the data set. Default: 15.
- **max\_size** (*int*) – Maximum allowed number of genes from gene set also the data set. Default: 500.
- **weight** (*float*) – Refer to `algorithm.enrichment_score()`. Default: 1.
- **method** – The method used to calculate a correlation or ranking. Default: 'log2\_ratio\_of\_classes'. Others methods are:
  1. 'signal\_to\_noise'

You must have at least three samples for each phenotype to use this metric. The larger the signal-to-noise ratio, the larger the differences of the means (scaled by the standard deviations); that is, the more distinct the gene expression is in each phenotype and the more the gene acts as a "class marker."
  2. 't\_test'

Uses the difference of means scaled by the standard deviation and number of samples. Note: You must have at least three samples for each phenotype to use this metric. The larger the tTest ratio, the more distinct the gene expression is in each phenotype and the more the gene acts as a "class marker."
  3. 'ratio\_of\_classes' (also referred to as fold change).

Uses the ratio of class means to calculate fold change for natural scale data.
  4. 'diff\_of\_classes'

Uses the difference of class means to calculate fold change for nature scale data
  5. 'log2\_ratio\_of\_classes'

Uses the log2 ratio of class means to calculate fold change for natural scale data. This is the recommended statistic for calculating fold change for log scale data.
- **ascending** (*bool*) – Sorting order of rankings. Default: False.
- **threads** (*int*) – Number of threads you are going to use. Default: 4.
- **figsize** (*list*) – Matplotlib figsize, accept a tuple or list, e.g. [width,height]. Default: [6.5,6].
- **format** (*str*) – Matplotlib figure format. Default: 'pdf'.
- **graph\_num** (*int*) – Plot graphs for top sets of each phenotype.
- **no\_plot** (*bool*) – If equals to True, no figure will be drawn. Default: False.
- **seed** – Random seed. expect an integer. Default: None.

- **verbose** (*bool*) – Bool, increase output verbosity, print out progress of your job, Default: False.

### Returns

Return a GSEA obj. All results store to a dictionary, obj.results, where contains:

```
| {
|   term: gene set name,
|   es: enrichment score,
|   nes: normalized enrichment score,
|   pval: Nominal p-value (from the null distribution of the gene set,
|   fdr: FDR qvalue (adjusted False Discory Rate),
|   fwerp: Family wise error rate p-values,
|   tag %: Percent of gene set before running enrichment peak (ES),
|   gene %: Percent of gene list before running enrichment peak (ES),
|   lead_genes: leading edge genes (gene hits before running enrichment.
|   ↪peak),
|   matched genes: genes matched to the data,
| }
```

### gseapy.prerank()

Run Gene Set Enrichment Analysis with pre-ranked correlation defined by user.

#### Parameters

- **rnk** – pre-ranked correlation table or pandas DataFrame. Same input with GSEA .rnk file.
- **gene\_sets** – Enrichr Library name or .gmt gene sets file or dict of gene sets. Same input with GSEA.
- **outdir** – results output directory. If None, nothing will write to disk.
- **permutation\_num** (*int*) – Number of permutations. Default: 1000. Minimal possible nominal p-value is about 1/nperm.
- **min\_size** (*int*) – Minimum allowed number of genes from gene set also the data set. Default: 15.
- **max\_size** (*int*) – Maximum allowed number of genes from gene set also the data set. Defaults: 500.
- **weight** (*str*) – Refer to `algorithm.enrichment_score()`. Default: 1.
- **ascending** (*bool*) – Sorting order of rankings. Default: False.
- **threads** (*int*) – Number of threads you are going to use. Default: 4.
- **figsize** (*list*) – Matplotlib figsize, accept a tuple or list, e.g. [width,height]. Default: [6.5,6].
- **format** (*str*) – Matplotlib figure format. Default: 'pdf'.
- **graph\_num** (*int*) – Plot graphs for top sets of each phenotype.
- **no\_plot** (*bool*) – If equals to True, no figure will be drawn. Default: False.
- **seed** – Random seed. expect an integer. Default: None.
- **verbose** (*bool*) – Bool, increase output verbosity, print out progress of your job, Default: False.

## Returns

Return a Prerank obj. All results store to a dictionary, obj.results, where contains:

```
| {  
|   term: gene set name,  
|   es: enrichment score,  
|   nes: normalized enrichment score,  
|   pval: Nominal p-value (from the null distribution of the gene set,  
|   fdr: FDR qvalue (adjusted False Discory Rate),  
|   fwerp: Family wise error rate p-values,  
|   tag %: Percent of gene set before running enrichment peak (ES),  
|   gene %: Percent of gene list before running enrichment peak (ES),  
|   lead_genes: leading edge genes (gene hits before running enrichment,  
|   ↪ peak),  
|   matched genes: genes matched to the data,  
| }
```

## `gseapy.ssgsea()`

Run Gene Set Enrichment Analysis with single sample GSEA tool

### Parameters

- **data** – Expression table, pd.Series, pd.DataFrame, GCT file, or .rnk file format.
- **gene\_sets** – Enrichr Library name or .gmt gene sets file or dict of gene sets. Same input with GSEA.
- **outdir** – Results output directory. If None, nothing will write to disk.
- **sample\_norm\_method** (*str*) – Sample normalization method. Choose from {'rank', 'log', 'log\_rank', None}. Default: rank. this argument will be used for ordering genes.
  1. 'rank': Rank your expression data, and transform by  $10000 * \text{rank\_dat} / \text{gene\_numbers}$
  2. 'log' : Do not rank, but transform data by  $\log(\text{data} + \exp(1))$ , while  $\text{data} = \text{data}[\text{data} < 1]$  =1.
  3. 'log\_rank': Rank your expression data, and transform by  $\log(10000 * \text{rank\_dat} / \text{gene\_numbers} + \exp(1))$
  4. None or 'custom': Do nothing, and use your own rank value to calculate enrichment score.

see here: <https://github.com/GSEA-MSigDB/ssGSEAProjection-gpmodule/blob/master/src/ssGSEAProjection.Library.R>, line 86

### Parameters

- **correl\_norm\_type** (*str*) – correlation normalization type. Choose from {'rank', 'sym-rank', 'zscore', None}. Default: rank. After ordering genes by sample\_norm\_method, further data transformed could be applied to get enrichment score.

when weight == 0, sample\_norm\_method and correl\_norm\_type do not matter; when weight > 0, the combination of sample\_norm\_method and correl\_norm\_type dictate how the gene expression values in input data are transformed to obtain the score – use this setting with care (the transformations can skew scores towards +ve or -ve values)

sample\_norm\_method will first transformed and rank original data. the data is named correl\_vector for each sample. then correl\_vector is transformed again by

1. correl\_norm\_type is None or 'rank' : do nothing, genes are weighted by actual correl\_vector.



2. `correl_norm_type == 'symrank'`: symmetric ranking.
  3. `correl_norm_type == 'zscore'`: standardizes the `correl_vector` before using them to calculate scores.
- **`min_size (int)`** – Minimum allowed number of genes from gene set also the data set. Default: 15.
  - **`max_size (int)`** – Maximum allowed number of genes from gene set also the data set. Default: 2000.
  - **`permutation_num (int)`** – For ssGSEA, default is 0. However, if you try to use `ssgsea` method to get `pval` and `fdr`, set to an interger.
  - **`weight (str)`** – Refer to `algorithm.enrichment_score()`. Default:0.25.
  - **`ascending (bool)`** – Sorting order of rankings. Default: False.
  - **`threads (int)`** – Number of threads you are going to use. Default: 4.
  - **`figsize (list)`** – Matplotlib figsize, accept a tuple or list, e.g. [width,height]. Default: [7,6].
  - **`format (str)`** – Matplotlib figure format. Default: 'pdf'.
  - **`graph_num (int)`** – Plot graphs for top sets of each phenotype.
  - **`no_plot (bool)`** – If equals to True, no figure will be drawn. Default: False.
  - **`seed`** – Random seed. expect an integer. Default:None.
  - **`verbose (bool)`** – Bool, increase output verbosity, print out progress of your job, Default: False.

### Returns

Return a ssGSEA obj. All results store to a dictionary, access enrichment score by `obj.resultsOnSamples`, and normalized enrichment score by `obj.res2d`. if `permutation_num > 0`, additional results contain:

```
| {
|   term: gene set name,
|   es: enrichment score,
|   nes: normalized enrichment score,
|   pval: Nominal p-value (from the null distribution of the gene set,
|   ↪(if permutation_num > 0),
|   fdr: FDR qvalue (adjusted FDR) (if permutation_num > 0),
|   fwerp: Family wise error rate p-values (if permutation_num > 0),
|   tag %: Percent of gene set before running enrichment peak (ES),
|   gene %: Percent of gene list before running enrichment peak (ES),
|   lead_genes: leading edge genes (gene hits before running enrichment,
|   ↪peak),
|   matched genes: genes matched to the data,
| }
```

`gseapy.enrichr()`

Enrichr API.

### Parameters

- **`gene_list`** – str, list, tuple, series, dataframe. Also support input txt file with one gene id per row. The input *identifier* should be the same type to *gene\_sets*.

- **gene\_sets** – str, list, tuple of Enrichr Library name(s). or custom defined gene\_sets (dict, or gmt file).

Examples:

**Input Enrichr Libraries (<https://maayanlab.cloud/Enrichr/#stats>):**

str: 'KEGG\_2016' list: ['KEGG\_2016','KEGG\_2013'] Use comma to separate each other, e.g. "KEGG\_2016,huMAP,GO\_Biological\_Process\_2018"

**Input custom files:**

```
dict: gene_sets={'A':['gene1', 'gene2',...],  
               'B':['gene2', 'gene4',...], ... }
```

```
gmt: "genes.gmt"
```

see also the online docs: [https://gseapy.readthedocs.io/en/latest/gseapy\\_example.html#2.-Enrichr-Example](https://gseapy.readthedocs.io/en/latest/gseapy_example.html#2.-Enrichr-Example)

- **organism** – Enrichr supported organism. Select from (human, mouse, yeast, fly, fish, worm). This argument only affects the Enrichr library names you've chosen. No any affects to gmt or dict input of *gene\_sets*.

see here for more details: <https://maayanlab.cloud/modEnrichr/>.

- **outdir** – Output file directory
- **background** – int, list, str. Background genes. This argument works only if *gene\_sets* has a type Dict or gmt file. If your input are just Enrichr library names, this argument will be ignored.

However, this argument is not straightforward when *gene\_sets* is given a custom input (a gmt file or dict).

By default, all genes listed in the *gene\_sets* input will be used as background.

There are 3 ways to tune this argument:

- (1) (Recommended) Input a list of background genes: ['gene1', 'gene2',...] The background gene list is defined by your experment. e.g. the expressed genes in your RNA-seq. The gene identifier in gmt/dict should be the same type to the background genes.
- (2) Specify a number: e.g. 20000. (the number of total expressed genes). This works, but not recommend. It assumes that all your genes could be found in background. If genes exist in gmt but not included in background provided, they will affect the significance of the statistical test.
- (3) Set a Biomart dataset name: e.g. "hsapiens\_gene\_ensembl" The background will be all annotated genes from the *BioMart datasets* you've choosen. The program will try to retrieve the background information automatically.

**Enrichr module use the code below to get the background genes:**

```
>>> from gseapy.parser import Biomart  
>>> bm = Biomart()  
>>> df = bm.query(dataset=background, # e.g. 'hsapiens_gene_ensembl'  
                  attributes=['ensembl_gene_id', 'external_gene_name',  
                  'entrezgene_id'],  
                  filename=f'~/ .cache/gseapy/{background}.background.  
                  genes.txt')  
>>> df.dropna(subset=["entrezgene_id"], inplace=True)
```

So only genes with entrezid above will be the background genes if not input specify by user.

- **cutoff** – Show enriched terms which Adjusted P-value < cutoff. Only affects the output figure, not the final output file. Default: 0.05
- **format** – Output figure format supported by matplotlib, ('pdf', 'png', 'eps'...). Default: 'pdf'.
- **figsize** – Matplotlib figsize, accept a tuple or list, e.g. (width,height). Default: (6.5,6).
- **no\_plot** (*bool*) – If equals to True, no figure will be drawn. Default: False.
- **verbose** (*bool*) – Increase output verbosity, print out progress of your job, Default: False.

### Returns

An Enrichr object, which `obj.res2d` stores your last query, `obj.results` stores your all queries.

### `gseapy.enrich()`

Perform over-representation analysis (hypergeometric test).

### Parameters

- **gene\_list** – str, list, tuple, series, dataframe. Also support input txt file with one gene id per row. The input *identifier* should be the same type to *gene\_sets*.
- **gene\_sets** – str, list, tuple of Enrichr Library name(s). or custom defined *gene\_sets* (dict, or gmt file).

Examples:

```
dict: gene_sets={'A':['gene1', 'gene2',...],
                'B':['gene2', 'gene4',...], ... }
```

```
gmt: "genes.gmt"
```

- **outdir** – Output file directory
- **background** – None | int | list | str. Background genes. This argument works only if *gene\_sets* has a type Dict or gmt file.

However, this argument is not straightforward when *gene\_sets* is given a custom input (a gmt file or dict).

By default, all genes listed in the *gene\_sets* input will be used as background.

There are 3 ways to tune this argument:

- (1) (Recommended) Input a list of background genes: ['gene1', 'gene2',...] The background gene list is defined by your experiment. e.g. the expressed genes in your RNA-seq. The gene identifier in gmt/dict should be the same type to the background genes.
- (2) Specify a number: e.g. 20000. (the number of total expressed genes). This works, but not recommend. It assumes that all your genes could be found in background. If genes exist in gmt but not included in background provided, they will affect the significance of the statistical test.
- (3) Set a Biomart dataset name: e.g. "hsapiens\_gene\_ensembl" The background will be all annotated genes from the *BioMart datasets* you've chosen. The program will try to retrieve the background information automatically.

**Enrichr module use the code below to get the background genes:**

```
>>> from gseapy.parser import Biomart
>>> bm = Biomart()
>>> df = bm.query(dataset=background, # e.g. 'hsapiens_gene_
↳ ensembl'
               attributes=['ensembl_gene_id', 'external_gene_name',
↳ 'entrezgene_id'],
               filename=f'~/ .cache/gseapy/{background}.background.
↳ genes.txt')
>>> df.dropna(subset=["entrezgene_id"], inplace=True)
```

So only genes with entrezid above will be the background genes if not input specify by user.

- **cutoff** – Show enriched terms which Adjusted P-value < cutoff. Only affects the output figure, not the final output file. Default: 0.05
- **format** – Output figure format supported by matplotlib, ('pdf', 'png', 'eps'...). Default: 'pdf'.
- **figsize** – Matplotlib figsize, accept a tuple or list, e.g. (width,height). Default: (6.5,6).
- **no\_plot** (*bool*) – If equals to True, no figure will be drawn. Default: False.
- **verbose** (*bool*) – Increase output verbosity, print out progress of your job, Default: False.

#### Returns

An Enrichr object, which `obj.res2d` stores your last query, `obj.results` stores your all queries.

#### `gseapy.replot()`

The main function to reproduce GSEA desktop outputs.

#### Parameters

- **indir** – GSEA desktop results directory. In the sub folder, you must contain edb file folder.
- **outdir** – Output directory.
- **weight** (*float*) – weighted score type. choose from {0,1,1.5,2}. Default: 1.
- **figsize** (*list*) – Matplotlib output figure figsize. Default: [6.5,6].
- **format** (*str*) – Matplotlib output figure format. Default: 'pdf'.
- **min\_size** (*int*) – Min size of input genes presented in Gene Sets. Default: 3.
- **max\_size** (*int*) – Max size of input genes presented in Gene Sets. Default: 5000. You are not encouraged to use `min_size`, or `max_size` argument in `replot()` function. Because gmt file has already been filtered.
- **verbose** – Bool, increase output verbosity, print out progress of your job, Default: False.

#### Returns

Generate new figures with selected figure format. Default: 'pdf'.

## 5.6.2 GSEA Statistics

```
class gseapy.gsea.GSEA(data: DataFrame | str, gene_sets: List[str] | str | Dict[str, str], classes: List[str] | str | Dict[str, str], outdir: str | None = None, min_size: int = 15, max_size: int = 500, permutation_num: int = 1000, weight: float = 1.0, permutation_type: str = 'phenotype', method: str = 'signal_to_noise', ascending: bool = False, threads: int = 1, figsize: Tuple[float, float] = (6.5, 6), format: str = 'pdf', graph_num: int = 20, no_plot: bool = False, seed: int = 123, verbose: bool = False)
```

GSEA main tool

```
calc_metric(df: DataFrame, method: str, pos: str, neg: str, classes: Dict[str, str], ascending: bool) → Tuple[List[int], Series]
```

The main function to rank an expression table. works for 2d array.

### Parameters

- **df** – gene\_expression DataFrame.
- **method** – The method used to calculate a correlation or ranking. Default: 'log2\_ratio\_of\_classes'. Others methods are:
  1. 'signal\_to\_noise' (s2n) or 'abs\_signal\_to\_noise' (abs\_s2n)
 

You must have at least three samples for each phenotype. The more distinct the gene expression is in each phenotype, the more the gene acts as a “class marker”.
  2. 't\_test'
 

Uses the difference of means scaled by the standard deviation and number of samples. Note: You must have at least three samples for each phenotype to use this metric. The larger the t-test ratio, the more distinct the gene expression is in each phenotype and the more the gene acts as a “class marker.”
  3. 'ratio\_of\_classes' (also referred to as fold change).
 

Uses the ratio of class means to calculate fold change for natural scale data.
  4. 'diff\_of\_classes'
 

Uses the difference of class means to calculate fold change for natural scale data
  5. 'log2\_ratio\_of\_classes'
 

Uses the log2 ratio of class means to calculate fold change for natural scale data. This is the recommended statistic for calculating fold change for log scale data.
- **pos** (*str*) – one of labels of phenotype's names.
- **neg** (*str*) – one of labels of phenotype's names.
- **classes** (*dict*) – column id to group mapping.
- **ascending** (*bool*) – bool or list of bool. Sort ascending vs. descending.

### Returns

returns argsort values of a tuple where 0: argsort positions (indices) 1: pd.Series of correlation value. Gene\_name is index, and value is rankings.

visit here for more docs: <http://software.broadinstitute.org/gsea/doc/GSEAUUserGuideFrame.html>

```
load_classes(classes: str | List[str] | Dict[str, Any])
```

Parse group (classes)

**load\_data**(groups: List[str] | Dict[str, Any]) → Tuple[DataFrame, Dict]

pre-processed the data frame.new filtering methods will be implement here.

**run()**

GSEA main procedure

```
class gseapy.gsea.Prerank(rnk: DataFrame | Series | str, gene_sets: List[str] | str | Dict[str, str], outdir: str | None = None, pheno_pos='Pos', pheno_neg='Neg', min_size: int = 15, max_size: int = 500, permutation_num: int = 1000, weight: float = 1.0, ascending: bool = False, threads: int = 1, figsize: Tuple[float, float] = (6.5, 6), format: str = 'pdf', graph_num: int = 20, no_plot: bool = False, seed: int = 123, verbose: bool = False)
```

GSEA prerank tool

**run()**

GSEA prerank workflow

```
class gseapy.gsea.Replot(indir: str, outdir: str = 'GSEApY_Replot', weight: float = 1.0, min_size: int = 3, max_size: int = 1000, figsize: Tuple[float, float] = (6.5, 6), format: str = 'pdf', verbose: bool = False)
```

To reproduce GSEA desktop output results.

**gsea\_edb\_parser**(results\_path)

Parse results.edb file stored under **edb** file folder.

**Parameters**

**results\_path** – the path of results.edb file.

**Returns**

a dict contains { enrichment\_term: [es, nes, pval, fdr, fwer, hit\_ind]}

**run()**

main replot function

```
class gseapy.base.GMT(mapping: Dict[str, str] | None = None, description: str | None = None)
```

**apply**(func)

apply function in place

**write**(ofname: str)

write gmt file to disk

```
class gseapy.base.GSEAbase(outdir: str | None = None, gene_sets: List[str] | str | Dict[str, str] = 'KEGG_2016', module: str = 'base', threads: int = 1, enrichr_url: str = 'http://maayanlab.cloud', verbose: bool = False)
```

base class of GSEA.

```
enrichment_score(gene_list: Iterable[str], correl_vector: Iterable[float], gene_set: Dict[str, List[str]], weight: float = 1.0, nperm: int = 1000, seed: int = 123, single: bool = False, scale: bool = False)
```

This is the most important function of GSEApY. It has the same algorithm with GSEA and ssGSEA.

**Parameters**

- **gene\_list** – The ordered gene list gene\_name\_list, rank\_metric.index.values
- **gene\_set** – gene\_sets in gmt file, please use gmt\_parser to get gene\_set.

- **weight** – It's the same with gsea's `weighted_score` method. Weighting by the correlation is a very reasonable choice that allows significant gene sets with less than perfect coherence. options: 0(classic),1,1.5,2. default:1. if one is interested in penalizing sets for lack of coherence or to discover sets with any type of nonrandom distribution of tags, a value  $p < 1$  might be appropriate. On the other hand, if one uses sets with large number of genes and only a small subset of those is expected to be coherent, then one could consider using  $p > 1$ . Our recommendation is to use  $p = 1$  and use other settings only if you are very experienced with the method and its behavior.
- **correl\_vector** – A vector with the correlations (e.g. signal to noise scores) corresponding to the genes in the gene list. Or rankings, `rank_metric.values`
- **nperm** – Only use this parameter when computing esnull for statistical testing. Set the esnull value equal to the permutation number.
- **seed** – Random state for initializing gene list shuffling. Default: `seed=None`

### Returns

ES: Enrichment score (real number between -1 and +1)

ESNULL: Enrichment score calculated from random permutations.

Hits\_Indices: Index of a gene in `gene_list`, if gene is included in `gene_set`.

RES: Numerical vector containing the running enrichment score for all locations in the gene list .

**get\_libraries()** → List[str]

return active enrichr library name. Official API

**load\_gmt**(*gene\_list: Iterable[str], gmt: List[str] | str | Dict[str, str]*) → Dict[str, List[str]]

load gene set dict

**load\_gmt\_only**(*gmt: List[str] | str | Dict[str, str]*) → Dict[str, List[str]]

parse gene\_sets. gmt: List, Dict, Strings

However, this function will merge different gene sets into one big dict to save computation time for later.

**parse\_gmt**(*gmt: str*) → Dict[str, List[str]]

gmt parser when input is a string

**plot**(*terms: str | List[str], colors: str | List[str] | None = None, legend\_kws: Dict[str, Any] | None = None, figsize: Tuple[float, float] = (4, 5), show\_ranking: bool = True, ofname: str | None = None*)

terms: str, list. terms/pathways to show colors: str, list. list of colors for each term/pathway legend\_kws: kwargs to pass to ax.legend. e.g. *loc, bbox\_to\_anchor*. ofname: savefig

**prepare\_outdir()**

create temp directory.

**property results**

compatible to old style

**to\_df**(*gsea\_summary: List[Dict], gmt: Dict[str, List[str]], rank\_metric: Series | DataFrame, indices: List | None = None*)

Convert GSEASummary to DataFrame

**rank\_metric: if a Series, then it must be sorted in descending order already**  
if a DataFrame, indices must not None.

indices: Only works for DataFrame input. Stores the indices of sorted array

### 5.6.3 Over-representation Statistics

`gseapy.stats.calc_pvalues(query, gene_sets, background=20000, **kwargs)`

calculate pvalues for all categories in the graph

#### Parameters

- **query** (*set*) – set of identifiers for which the p value is calculated
- **gene\_sets** (*dict*) – gmt file dict after background was set
- **background** (*set*) – total number of genes in your annotated database.

#### Returns

pvalues x: overlapped gene number n: length of gene\_set which belongs to each terms hits:  
overlapped gene names.

#### For 2\*2 contingency table:

in query | not in query | row total

=> in gene\_set | a | b | a+b => not in gene\_set | c | d | c+d

column total | a+b+c+d = anno database

#### Then, in R

**x=a the number of white balls drawn without replacement**

from an urn which contains both black and white balls.

m=a+b the number of white balls in the urn n=c+d the number of black balls in the urn k=a+c the number of balls drawn from the urn

In Scipy: for args in `scipy.hypergeom.sf(k, M, n, N, loc=0)`:

M: the total number of objects, n: the total number of Type I objects. k: the random variate represents the number of Type I objects in N drawn

without replacement from the total population.

Therefore, these two functions are the same when using parameters from 2\*2 table: R: `> phyper(x-1, m, n, k, lower.tail=FALSE)` Scipy: `>>> hypergeom.sf(x-1, m+n, m, k)`

For Odds ratio in Enrichr (see <https://maayanlab.cloud/Enrichr/help#background&q=4>)

$$\text{oddsRatio} = (1.0 * x * d) / \text{Math.max}(1.0 * b * c, 1)$$

where:

x are the overlapping genes, b (m-x) are the genes in the annotated set - overlapping genes, c (k-x) are the genes in the input set - overlapping genes, d (bg-m-k+x) are the 20,000 genes (or total genes in the background) - genes in the annotated set - genes in the input set + overlapping genes

`gseapy.stats.fdr correction(pvals, alpha=0.05)`

benjamini hochberg fdr correction. inspired by statsmodels

`gseapy.stats.multiple_testing_correction(ps, alpha=0.05, method='benjamini-hochberg', **kwargs)`

correct pvalues for multiple testing and add corrected q value

#### Parameters

- **ps** – list of pvalues



- **alpha** – significance level default : 0.05
- **method** – multiple testing correction method [bonferroni|benjamini-hochberg]

**Returns (q, rej)**

two lists of q-values and rejected nodes

## 5.6.4 Enrichr API

```
class gseapy.enrichr.Enrichr(gene_list: Iterable[str], gene_sets: List[str] | str | Dict[str, str], organism: str
                             = 'human', outdir: str | None = 'Enrichr', background: List[str] | int | str =
                             'hsapiens_gene_ensembl', cutoff: float = 0.05, format: str = 'pdf', figsize:
                             Tuple[float, float] = (6.5, 6), top_term: int = 10, no_plot: bool = False,
                             verbose: bool = False)
```

Enrichr API

**check\_genes**(gene\_list: List[str], usr\_list\_id: str)

Compare the genes sent and received to get successfully recognized genes

**enrich**(gmt: Dict[str, List[str]])

use local mode

p = p-value computed using the Fisher exact test (Hypergeometric test) z = z-score (Odds Ratio) combine  
score = - log(p)-z

see here: <http://amp.pharm.mssm.edu/Enrichr/help#background&q=4>

columns contain:

Term Overlap P-value Odds Ratio Combindede Score Adjusted\_P-value Genes

**filter\_gmt**(gmt, background)

the gmt values should be filtered only for genes that exist in background this substantially affect the sig-  
nificance of the test, the hypergeometric distribution.

**Parameters**

- **gmt** – a dict of gene sets.
- **background** – list, set, or tuple. A list of custom background genes.

**get\_background**() → Set[str]

get background gene

**get\_libraries**() → List[str]

return active enrichr library name. Official API

**get\_results**(gene\_list: List[str]) → Tuple[AnyStr, DataFrame]

Enrichr API

**parse\_background**(gmt: Dict[str, List[str]] | None = None)

set background genes

**parse\_genelists**() → str

parse gene list

**parse\_genesets**(gene\_sets=None)

parse gene\_sets input file type

**prepare\_outdir()**

create temp directory.

**run()**

run enrichr for one sample gene list but multi-libraries

**send\_genes(payload, url)** → Dict

send gene list to enrichr server

**set\_organism()**

Select Enrichr organism from below:

Human & Mouse, H. sapiens & M. musculus Fly, D. melanogaster Yeast, S. cerevisiae Worm, C. elegans  
Fish, D. rerio

## 5.6.5 BioMart API

**class gseapy.biomart.Biomart**(host: str = 'www.ensembl.org', verbose: bool = False)

query from BioMart

**add\_filter**(name: str, value: Iterable[str])

key: filter names value: Iterable[str]

**get\_attributes**(dataset: str = 'hsapiens\_gene\_ensembl')

Get available attributes from dataset you've selected

**get\_datasets**(mart: str = 'ENSEMBL\_MART\_ENSEMBL')

Get available datasets from mart you've selected

**get\_filters**(dataset: str = 'hsapiens\_gene\_ensembl')

Get available filters from dataset you've selected

**get\_marts()**

Get available marts and their names.

**query**(dataset: str = 'hsapiens\_gene\_ensembl', attributes: List[str] | None = [], filters: Dict[str, Iterable[str]] | None = {}, filename: str | None = None)

mapping ids using BioMart.

### Parameters

- **dataset** – str, default: 'hsapiens\_gene\_ensembl'
- **attributes** – str, list, tuple
- **filters** – dict, {'filter name': list(filter value)}
- **host** – www.ensembl.org, asia.ensembl.org, useast.ensembl.org

### Returns

a dataframe contains all attributes you selected.

Example:

```
>>> queries = {'ensembl_gene_id': ['ENSG00000125285', 'ENSG00000182968']} #_
↪ need to be a python dict
>>> results = bm.query(dataset='hsapiens_gene_ensembl',
                        attributes=['ensembl_gene_id', 'external_gene_name',
```

(continues on next page)

(continued from previous page)

```
→ 'entrezgene_id', 'go_id'],
    filters=queries)
```

```
query_simple(dataset: str = 'hsapiens_gene_ensembl', attributes: List[str] = [], filters: Dict[str,
    Iterable[str]] = {}, filename: str | None = None)
```

This function is a simple version of BioMart REST API. same parameter to query().

However, you could get cross page of mapping. such as Mouse 2 human gene names

**Note:** it will take a couple of minutes to get the results. A xml template for querying biomart. (see <https://gist.github.com/keithshep/7776579>)

**Example::**

```
>>> from gseapy import Biomart
>>> bm = Biomart()
>>> results = bm.query_simple(dataset='mmusculus_gene_ensembl',
    attributes=['ensembl_gene_id',
    'external_gene_name',
    'hsapiens_homolog_associated_
→ gene_name',
    'hsapiens_homolog_ensembl_gene'])
```

### 5.6.6 Parser

`gseapy.parser.download_library(name: str, organism: str = 'human') → Dict[str, List[str]]`

download enrichr libraries.

#### Parameters

- **name** (str) – the enrichr library name. see `gseapy.get_library_name()`.
- **organism** (str) – Select one from { 'Human', 'Mouse', 'Yeast', 'Fly', 'Fish', 'Worm' }

#### Return dict

gene\_sets of the enrichr library from selected organism

`gseapy.parser.get_library(name: str, organism: str = 'Human', min_size: int = 0, max_size: int = 2000, gene_list: List[str] | None = None) → Dict[str, List[str]]`

Parse gene\_sets.gmt(gene set database) file or download from enrichr server.

#### Parameters

- **name** (str) – the gene\_sets.gmt file or an enrichr library name. checkout full enrichr library name here: <https://maayanlab.cloud/Enrichr/#libraries>
- **organism** (str) – choose one from { 'Human', 'Mouse', 'Yeast', 'Fly', 'Fish', 'Worm' }. This argument has not effect if input is a .gmt file.
- **min\_size** – Minimum allowed number of genes for each gene set. Default: 0.
- **max\_size** – Maximum allowed number of genes for each gene set. Default: 2000.
- **gene\_list** – if input a gene list, min and max overlapped genes between gene set and gene\_list are kept.

#### Return dict

Return a filtered gene set database dictionary.

Note: **DO NOT** filter gene sets, when use `replot()`. Because GSEA Desktop have already done this for you.

`gseapy.parser.get_library_name(organism: str = 'Human') → List[str]`

return enrichr active enrichr library name. see also: <https://maayanlab.cloud/modEnrichr/>

**Parameters**

**organism** (*str*) – Select one from { ‘Human’, ‘Mouse’, ‘Yeast’, ‘Fly’, ‘Fish’, ‘Worm’ }

**Returns**

a list of enrichr libraries from selected database

`gseapy.parser.gsea_cls_parser(cls: str) → Tuple[str]`

Extract class(phenotype) name from .cls file.

**Parameters**

**cls** – the a class list instance or .cls file which is identical to GSEA input .

**Returns**

phenotype name and a list of class vector.

`gseapy.parser.gsea_edb_parser(results_path: str) → Dict[str, List[str]]`

Parse results.edb file stored under **edb** file folder.

**Parameters**

**results\_path** – the path of results.edb file.

**Returns**

a dict contains { enrichment\_term: [es, nes, pval, fdr, fwer, hit\_ind]}

`gseapy.parser.read_gmt(path: str) → Dict[str, List[str]]`

Read GMT file

**Parameters**

**path** (*str*) – the path to a gmt file.

**Returns**

a dict object

## 5.6.7 Visualization

`class gseapy.plot.MidpointNormalize(vmin=None, vmax=None, vcenter=None, clip=False)`

`gseapy.plot.barplot(df: DataFrame, column: str = 'Adjusted P-value', group: str | None = None, title: str = "", cutoff: float = 0.05, top_term: int = 10, figsize: Tuple[float, float] = (4, 6), color: str | List[str] | Dict[str, str] = 'salmon', ofname: str | None = None, **kwargs)`

Visualize GSEApY Results. When multiple datasets exist in the input dataframe, the *group* argument is your friend.

**Parameters**

- **df** – GSEApY DataFrame results.
- **column** – column name in *df* to map the x-axis data. Default: Adjusted P-value
- **group** – group by the variable in *df* that will produce bars with different colors.
- **title** – figure title.
- **cutoff** – terms with *column* value < cut-off are shown. Work only for (“Adjusted P-value”, “P-value”, “NOM p-val”, “FDR q-val”)

- **top\_term** – number of top enriched terms grouped by *hue* are shown.
- **figsize** – tuple, matplotlib figsize.
- **color** – color or list or dict of matplotlib.colors. Must be reconigzed by matplotlib. if dict input, dict keys must be found in the *group*
- **ofname** – output file name. If None, don't save figure

#### Returns

matplotlib.Axes. return None if given ofname. Only terms with *column*  $\leq$  *cut-off* are plotted.

```
gseapy.plot.dotplot(df: DataFrame, column: str = 'Adjusted P-value', x: str | None = None, y: str = 'Term',
                    x_order: List[str] | bool = False, y_order: List[str] | bool = False, title: str = "", cutoff:
                    float = 0.05, top_term: int = 10, size: float = 5, figsize: Tuple[float, float] = (4, 6), cmap:
                    str = 'viridis_r', ofname: str | None = None, xticklabels_rot: float | None = None,
                    yticklabels_rot: float | None = None, marker: str = 'o', show_ring: bool = False,
                    **kwargs)
```

Visualize GSEapy Results with categorical scatterplot When multiple datasets exist in the input dataframe, the *x* argument is your friend.

#### Parameters

- **df** – GSEapy DataFrame results.
- **column** – column name in *df* that map the dot colors. Default: Adjusted P-value.
- **x** – Categorical variable in *df* that map the x-axis data. Default: None.
- **y** – Categorical variable in *df* that map the y-axis data. Default: Term.
- **x\_order** – bool, array-like list. Default: False. If True, peformed hierarchical\_clustering on X-axis. or input a array-like list of *x* categorical levels.
- **y\_order** – bool, array-like list. Default: False. If True, peformed hierarchical\_clustering on Y-axis. or input a array-like list of *y* categorical levels.
- **title** – Figure title.
- **cutoff** – Terms with *column* value  $<$  cut-off are shown. Work only for (“Adjusted P-value”, “P-value”, “NOM p-val”, “FDR q-val”)
- **top\_term** – Number of enriched terms to show.
- **size** – float, scale the dot size to get proper visualization.
- **figsize** – tuple, matplotlib figure size.
- **cmap** – Matplotlib colormap for mapping the *column* semantic.
- **ofname** – Output file name. If None, don't save figure
- **marker** – The matplotlib.markers. See [https://matplotlib.org/stable/api/markers\\_api.html](https://matplotlib.org/stable/api/markers_api.html)
- **bool (show\_ring)** – Whether to draw outer ring.

#### Returns

matplotlib.Axes. return None if given ofname. Only terms with *column*  $\leq$  *cut-off* are plotted.

```
gseapy.plot.enrichment_map(df: DataFrame, column: str = 'Adjusted P-value', cutoff: float = 0.05, top_term:
                           int = 10, **kwargs) → Tuple[DataFrame, DataFrame]
```

Visualize GSEapy Results. Node size corresponds to the percentage of gene overlap in a certain term of interest. Colour of the node corresponds to the significance of the enriched terms. Edge size corresponds to the number of genes that overlap between the two connected nodes. Gray edges correspond to both nodes when it is the only

colour edge. When there are two different edge colours, red corresponds to positive nodes and blue corresponds to negative nodes.

#### Parameters

- **df** – GSEapy DataFrame results.
- **column** – column name in *df* to map the node colors. Default: Adjusted P-value or FDR q-val. choose from (“Adjusted P-value”, “P-value”, “FDR q-val”, “NOM p-val”).
- **group** – group by the variable in *df* that will produce bars with different colors.
- **title** – figure title.
- **cutoff** – nodes with *column* value < cut-off are shown. Work only for (“Adjusted P-value”, “P-value”, “NOM p-val”, “FDR q-val”)
- **top\_term** – number of top enriched terms are selected as nodes.

#### Returns

tuple of dataframe (nodes, edges)

```
gseapy.plot.gseaplot(term: str, hits: Sequence[int], nes: float, pval: float, fdr: float, RES: Sequence[float],
rank_metric: Sequence[float] | None = None, pheno_pos: str = "", pheno_neg: str = "",
color: str = '#88C544', figsize: Tuple[float, float] = (6, 5.5), cmap: str = 'seismic',
ofname: str | None = None, **kwargs) → List[Axes] | None
```

This is the main function for generating the gsea plot.

#### Parameters

- **term** – gene\_set name
- **hits** – hits indices of rank\_metric.index presented in gene set S.
- **nes** – Normalized enrichment scores.
- **pval** – nominal p-value.
- **fdr** – false discovery rate.
- **RES** – running enrichment scores.
- **rank\_metric** – pd.Series for rankings, rank\_metric.values.
- **pheno\_pos** – phenotype label, positive correlated.
- **pheno\_neg** – phenotype label, negative correlated.
- **color** – color for RES and hits.
- **figsize** – matplotlib figsize.
- **ofname** – output file name. If None, don't save figure

return matplotlib.Figure.

```
gseapy.plot.gseaplot2(terms: List[str], hits: List[Sequence[int]], RESs: List[Sequence[float]], rank_metric:
Sequence[float] | None = None, colors: str | List[str] | None = None, figsize:
Tuple[float, float] = (6, 4), legend_kws: Dict[str, Any] | None = None, ofname: str |
None = None, **kwargs) → List[Axes] | None
```

Trace plot for combining multiple terms/pathways into one plot :param terms: list of terms to show in trace plot :param hits: list of hits indices correspond to each term. :param RESs: list of running enrichment scores correspond to each term. :param rank\_metric: Optional, rankings. :param figsize: matplotlib figsize. :legend\_kws: Optional, control the location of legends :param ofname: output file name. If None, don't save figure

return matplotlib.Figure.

```
gseapy.plot.heatmap(df: DataFrame, z_score: int | None = None, title: str = "", figsize: Tuple[float, float] = (5, 5), cmap: str | None = None, xticklabels: bool = True, yticklabels: bool = True, ofname: str | None = None, **kwargs)
```

Visualize the dataframe.

#### Parameters

- **df** – DataFrame from expression table.
- **z\_score** – 0, 1, or None. z\_score axis{0, 1}. If None, not scale.
- **title** – figure title.
- **figsize** – heatmap figsize.
- **cmap** – matplotlib colormap. e.g. “RdBu\_r”.
- **xticklabels** – bool, whether to show xticklabels.
- **yticklabels** – bool, whether to show yticklabels.
- **ofname** – output file name. If None, don’t save figure

```
gseapy.plot.ringplot(df: DataFrame, column: str = 'Adjusted P-value', x: str | None = None, title: str = "", cutoff: float = 0.05, top_term: int = 10, size: float = 5, figsize: Tuple[float, float] = (4, 6), cmap: str = 'viridis_r', ofname: str | None = None, xticklabels_rot: float | None = None, yticklabels_rot: float | None = None, marker='o', show_ring: bool = True, **kwargs)
```

ringplot is deprecated, use dotplot instead

#### Parameters

- **df** – GSEApY DataFrame results.
- **x** – Group by the variable in *df* that will produce categorical scatterplot.
- **column** – column name in *df* to map the dot colors. Default: Adjusted P-value
- **title** – figure title
- **cutoff** – terms with *column* value < cut-off are shown. Work only for (“Adjusted P-value”, “P-value”, “NOM p-val”, “FDR q-val”)
- **top\_term** – number of enriched terms to show.
- **size** – float, scale the dot size to get proper visualization.
- **figsize** – tuple, matplotlib figure size.
- **cmap** – matplotlib colormap for mapping the *column* semantic.
- **ofname** – output file name. If None, don’t save figure
- **marker** – the matplotlib.markers. See [https://matplotlib.org/stable/api/markers\\_api.html](https://matplotlib.org/stable/api/markers_api.html)
- **bool (show\_ring)** – whether to show outer ring.

#### Returns

matplotlib.Axes. return None if given ofname. Only terms with *column* <= *cut-off* are plotted.

```
gseapy.plot.zscore(data2d: DataFrame, axis: int | None = 0)
```

Standardize the mean and variance of the data axis Parameters.

#### Parameters

- **data2d** – DataFrame to normalize.

- **axis** – int, Which axis to normalize across. If 0, normalize across rows, if 1, normalize across columns. If None, don't change data

**Returns**

Normalized DataFrame. Normalized data with a mean of 0 and variance of 1 across the specified axis.

## **5.6.8 Scientific Journal and Sci- themed Color Palettes**

## **5.6.9 Utils**

# **5.7 Frequently Asked Questions**

## **5.7.1 Q: What kind of gene identifiers are supported in GSEAPy?**

**A:**

- If you select `Enrichr` library as your input `gene_sets` (gmt format), then gene symbols in upper cases are needed.
- If you use your own GMT file, you need to use the same type of your gene identifiers in GMT and input gene list.

## **5.7.2 Q: Why gene symbols in Enrichr library are all UPPER cases for mouse, fly, fish, worm ?**

**A:** GSEAPy can't change the Enrichr databases. So convert your gene symbols into UPPER cases first, then run the analysis you want.

## **5.7.3 Q: Why P-value or FDR is 0, not a very small number?**

**A:** GSEA methodology use random permutation procedure (e.g. 1000 permutation) to obtain a null distribution. Then, an observed ES is compared to the 1000 shuffled ES to calculate a P-value. When observed ES is not within the null ESs, you'll get 0s. if you don't want 0, you could

- set the smallest pvalue to  $1 / (\text{number of permutations})$
- increase the permutation number (but more running time needed)

## **5.7.4 Q: What are gene %, and tag % mean in the output?**



### 5.7.5 Q: What Enrichr database are supported?

**A:** Support modEnrich (<https://amp.pharm.mssm.edu/modEnrichr/>) . Now, Human, Mouse, Fly, Yeast, Worm, Fish are all supported.

### 5.7.6 Q: Use custom defined GMT file input in Jupyter ?

**A:** argument `gene_sets` accept dict input. This is useful when define your own `gene_sets`. An example dict looks like this:

```
gene_sets = {
    "term_1": ["gene_A", "gene_B", ...],
    "term_2": ["gene_B", "gene_C", ...],
    ...
    "term_100": ["gene_A", "gene_T", ...]
}
```

APIs support dict object input: `gsea`, `prerank`, `ssgsea`, `enrichr`

### 5.7.7 Q: How to use Yeast database in `gseapy.enrichr()`?

Because some library names are the same in different Enrichr database, you have to set an additional augment organism when no use **Human**

```
gss = gseapy.get_library_name(organism='Yeast')
enr = gseapy.enrichr(gene_list=...,
                    gene_sets=gss,
                    organism='Yeast', # don't forget to set organism="Yeast"
                    )
```

### 5.7.8 Q: How to use Yeast database in `gseapy.prerank()`?

There is no augment organism in `prerank`, `gsea`, `ssgea`, but you could input these Enrichr libraries as follow:

```
# get libraries you'd like to use
gss = gseapy.get_library_name(organism='Yeast')
# get a custom gmt_dict
gmt_dict = gseapy.parser.gsea_gmt_parser('GO_Biological_Process_2018', organism='Yeast')
# run
prn_res = gseapy.prerank( ..., gene_sets=gmt_dict, ...)
```

### 5.7.9 Q: How to save plots using gseaplot, barplot, dotplot, ``heatmap`` in Jupyter ?

A: e.g. `gseaplot(..., ofname='your.plot.pdf')`. That's it

### 5.7.10 Q: What cutoff mean in functions, like `enrichr()`, `dotplot`, `barplot` ?

A: This argument control the terms (e.g  $FDR < 0.05$ ) that will be shown on figures, not the result table output.

### 5.7.11 Q: ssGSEA missing p value and FDR?

A: The original ssGSEA algorithm will not give you pval or FDR, so, please ignore the gseaplot generated by `ssgsea`. It's useless and misleading, therefore, `fd`r, and `pval` are not shown on the plot. If you're seeking for ssGSEA with p-value output, please see here: <https://github.com/broadinstitute/ssGSEA2.0> Actually, ssGSEA2.0 use the same method with GSEAPy to calculate P-value, but FDR is not.

### 5.7.12 Q: What the difference between ssGSEA and Prerank

A: In short, - prerank is used for comparing **two group of samples** (e.g. control and treatment), where the gene ranking are defined by your custom rank method (like t-statistic, signal-to-noise, et.al). - ssGSEA is used for comparing individual samples to the rest of all, trying to find the gene signatures which samples shared the same (use ssGSEA when you have a lot of samples).

The statistic between prerank (GSEA) and ssGSEA are different. Assume that we have calculated each *running enrichment score* of your ranked input genes, then

- es for GSEA: *max(running enrichment scores)* or *min(running enrichment scores)*
- es for ssGSEA: *sum(running enrichment scores)*

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### g

- `gseapy`, [57](#)
- `gseapy.base`, [66](#)
- `gseapy.biomart`, [70](#)
- `gseapy.enrichr`, [69](#)
- `gseapy.gsea`, [65](#)
- `gseapy.parser`, [71](#)
- `gseapy.plot`, [72](#)
- `gseapy.scipalette`, [76](#)
- `gseapy.stats`, [68](#)



## A

`add_filter()` (*gseapy.biomart.Biomart method*), 70  
`apply()` (*gseapy.base.GMT method*), 66

## B

`barplot()` (*in module gseapy.plot*), 72  
*Biomart* (class in *gseapy.biomart*), 70

## C

`calc_metric()` (*gseapy.gsea.GSEA method*), 65  
`calc_pvalues()` (*in module gseapy.stats*), 68  
`check_genes()` (*gseapy.enrichr.Enrichr method*), 69

## D

`dotplot()` (*in module gseapy.plot*), 73  
`download_library()` (*in module gseapy.parser*), 71

## E

`enrich()` (*gseapy.enrichr.Enrichr method*), 69  
`enrich()` (*in module gseapy*), 63  
`enrichment_map()` (*in module gseapy.plot*), 73  
`enrichment_score()` (*gseapy.base.GSEABase method*), 66  
*Enrichr* (class in *gseapy.enrichr*), 69  
`enrichr()` (*in module gseapy*), 61

## F

`fdr correction()` (*in module gseapy.stats*), 68  
`filter_gmt()` (*gseapy.enrichr.Enrichr method*), 69

## G

`get_attributes()` (*gseapy.biomart.Biomart method*), 70  
`get_background()` (*gseapy.enrichr.Enrichr method*), 69  
`get_datasets()` (*gseapy.biomart.Biomart method*), 70  
`get_filters()` (*gseapy.biomart.Biomart method*), 70  
`get_libraries()` (*gseapy.base.GSEABase method*), 67  
`get_libraries()` (*gseapy.enrichr.Enrichr method*), 69  
`get_library()` (*in module gseapy.parser*), 71  
`get_library_name()` (*in module gseapy.parser*), 72  
`get_marts()` (*gseapy.biomart.Biomart method*), 70

`get_results()` (*gseapy.enrichr.Enrichr method*), 69  
*GMT* (class in *gseapy.base*), 66  
*GSEA* (class in *gseapy.gsea*), 65  
`gsea()` (*in module gseapy*), 57  
`gsea_cls_parser()` (*in module gseapy.parser*), 72  
`gsea_edb_parser()` (*gseapy.gsea.Replot method*), 66  
`gsea_edb_parser()` (*in module gseapy.parser*), 72  
*GSEABase* (class in *gseapy.base*), 66  
`gseaplot()` (*in module gseapy.plot*), 74  
`gseaplot2()` (*in module gseapy.plot*), 74  
*gseapy*  
    module, 57  
*gseapy.base*  
    module, 66  
*gseapy.biomart*  
    module, 70  
*gseapy.enrichr*  
    module, 69  
*gseapy.gsea*  
    module, 65  
*gseapy.parser*  
    module, 71  
*gseapy.plot*  
    module, 72  
*gseapy.scipalette*  
    module, 76  
*gseapy.stats*  
    module, 68

## H

`heatmap()` (*in module gseapy.plot*), 74

## L

`load_classes()` (*gseapy.gsea.GSEA method*), 65  
`load_data()` (*gseapy.gsea.GSEA method*), 65  
`load_gmt()` (*gseapy.base.GSEABase method*), 67  
`load_gmt_only()` (*gseapy.base.GSEABase method*), 67

## M

`MidpointNormalize` (class in *gseapy.plot*), 72  
*module*  
    *gseapy*, 57

- gseapy.base, 66
- gseapy.biomart, 70
- gseapy.enrichr, 69
- gseapy.gsea, 65
- gseapy.parser, 71
- gseapy.plot, 72
- gseapy.scipalette, 76
- gseapy.stats, 68
- multiple\_testing\_correction() (in module gseapy.stats), 68

## P

- parse\_background() (gseapy.enrichr.Enrichr method), 69
- parse\_genelists() (gseapy.enrichr.Enrichr method), 69
- parse\_genesets() (gseapy.enrichr.Enrichr method), 69
- parse\_gmt() (gseapy.base.GSEABase method), 67
- plot() (gseapy.base.GSEABase method), 67
- prepare\_outdir() (gseapy.base.GSEABase method), 67
- prepare\_outdir() (gseapy.enrichr.Enrichr method), 69
- Prerank (class in gseapy.gsea), 66
- prerank() (in module gseapy), 59

## Q

- query() (gseapy.biomart.Biomart method), 70
- query\_simple() (gseapy.biomart.Biomart method), 71

## R

- read\_gmt() (in module gseapy.parser), 72
- Replot (class in gseapy.gsea), 66
- replot() (in module gseapy), 64
- results (gseapy.base.GSEABase property), 67
- ringplot() (in module gseapy.plot), 75
- run() (gseapy.enrichr.Enrichr method), 70
- run() (gseapy.gsea.GSEA method), 66
- run() (gseapy.gsea.Prerank method), 66
- run() (gseapy.gsea.Replot method), 66

## S

- send\_genes() (gseapy.enrichr.Enrichr method), 70
- set\_organism() (gseapy.enrichr.Enrichr method), 70
- ssgsea() (in module gseapy), 60

## T

- to\_df() (gseapy.base.GSEABase method), 67

## W

- write() (gseapy.base.GMT method), 66

## Z

- zscore() (in module gseapy.plot), 75