gseapy Documentation

Release 1.0.0

Zhuoqing Fang

Dec 20, 2022

Table of Contents

CHAPTER 1

GSEAPY: Gene Set Enrichment Analysis in Python.

Release notes : https://github.com/zqfang/GSEApy/releases

CHAPTER 2

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
```

CHAPTER 3

Installation

Install gseapy package from bioconda or pypi.

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

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

CHAPTER 4

GSEApy is a Python/Rust implementation of **GSEA** and wrapper for **Enrichr**.

GSEApy has six subcommands: `gsea`, `prerank`, `ssgsea`, `replot` `enrichr`, `biomart`.

1. The `gsea` module produces **GSEA** results. The input requries 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 `replot` module reproduces GSEA desktop version results. The only input for GSEAPY is the location to GSEA Desktop output results.

5. 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.

6. The `biomart` module helps you convert gene ids using BioMart API.

GSEApy could be used for **RNA-seq, ChIP-seq, Microarry** 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.

CHAPTER 5

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, Microarry** 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
$ conda install -c conda-forge -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.

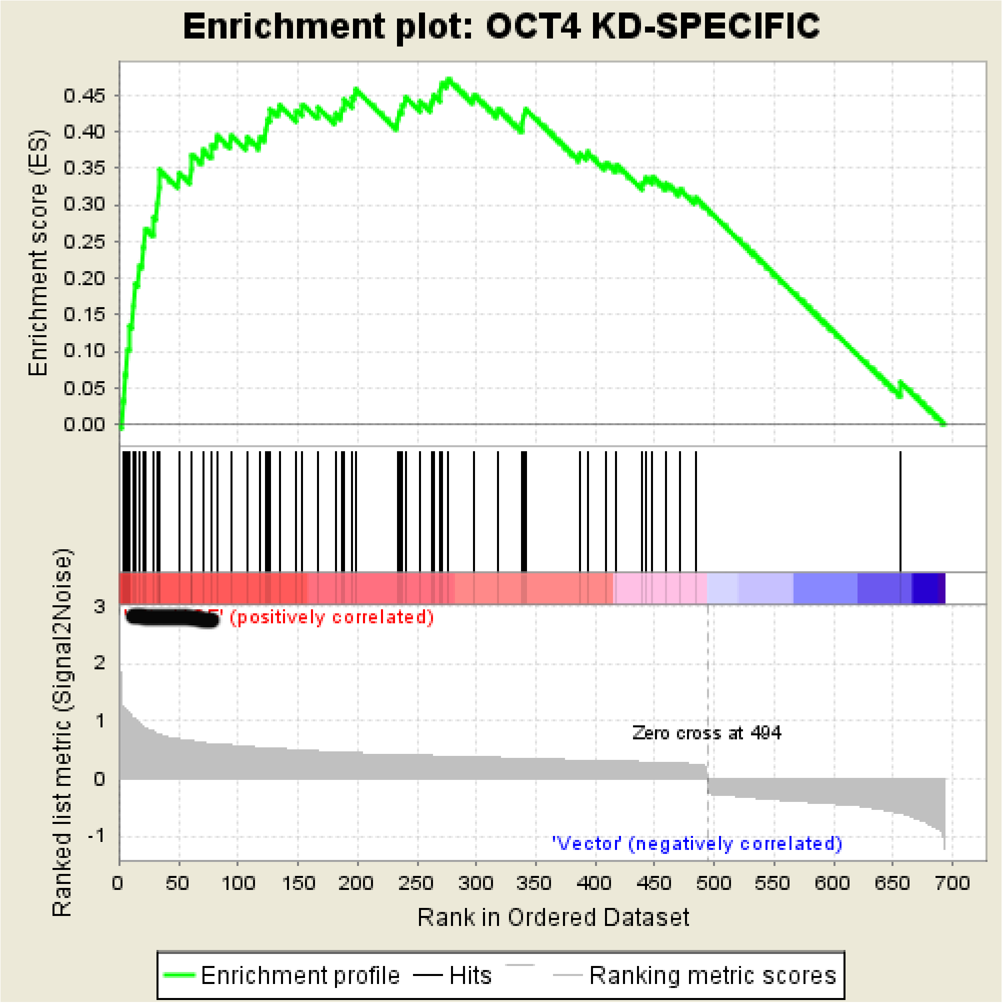
Note: Enrichr uses a list of Entrez gene symbols as input. You should convert all gene names to uppercase.

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

```
# if you prefer to run gseapy.enrchr() inside python console, you could assign a list␣
↪object to
# gseapy like this.
gene_list = ['SCARA3', 'LOC100044683', 'CMBL', 'CLIC6', 'IL13RA1', 'TACSTD2', 'DKKL1',
                'CSF1', 'CITED1', 'SYNPO2L']
```

```
# 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())
```

(continues on next page)


Enrichment plot: OCT4 KD-SPECIFIC
Enrichment score (ES)
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Ranked list metric (Signal2Noise)
3
2
1
0
-1
[redacted]' (positively correlated)
Zero cross at 494
'Vector' (negatively correlated)
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
Rank in Ordered Dataset
Enrichment profile
Hits
Ranking metric scores

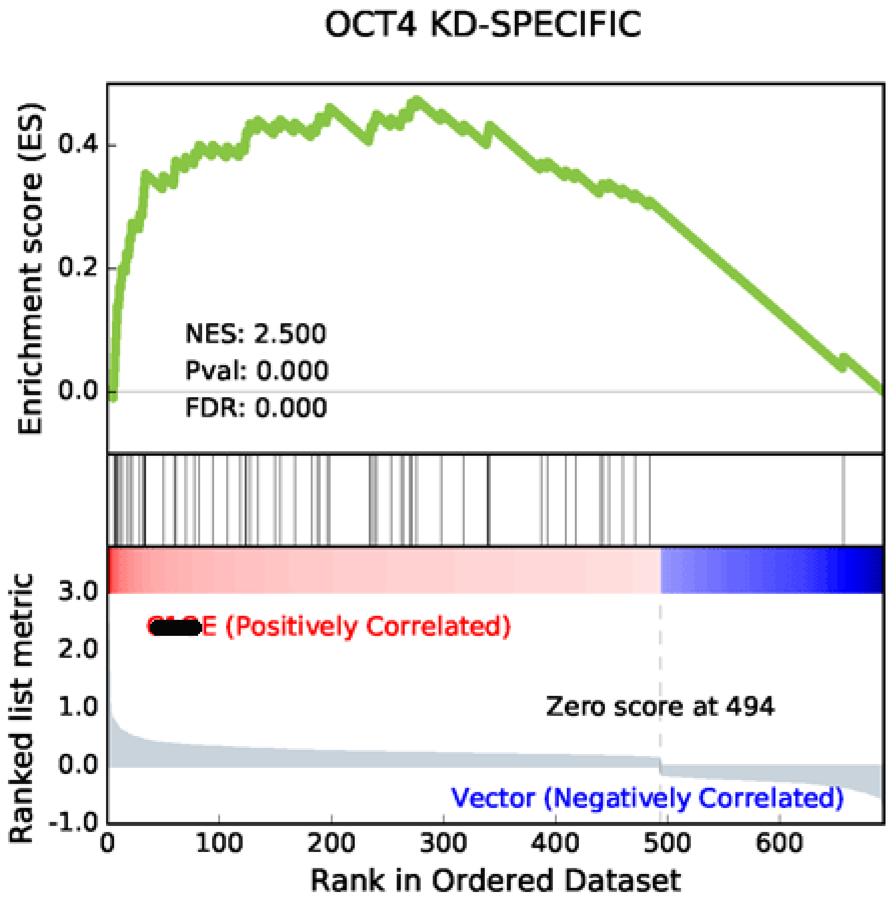
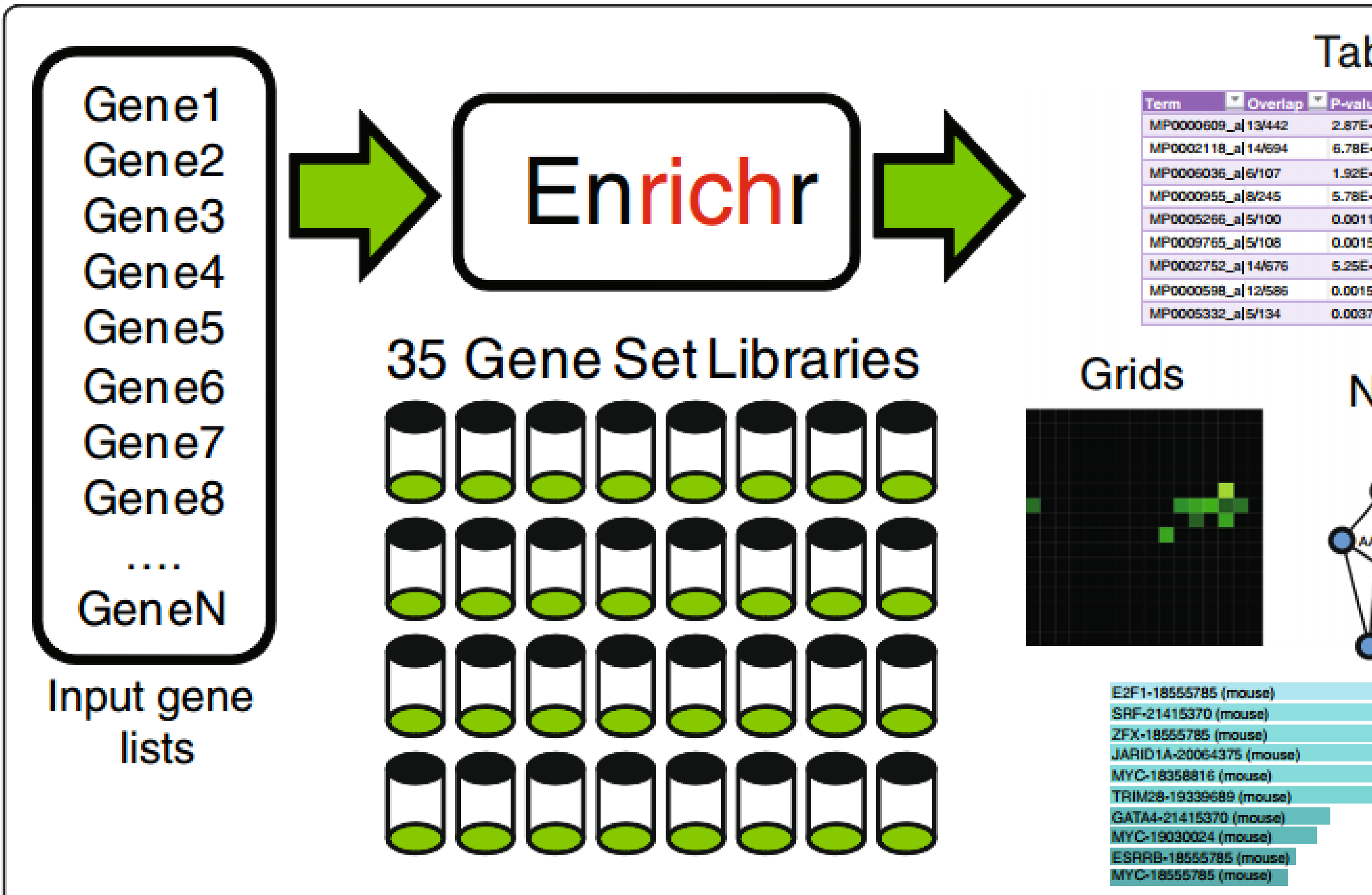


Fig. 1: Generated by GSEAPY
GSEApy figures are supported by all matplotlib figure formats.
You can modify `GSEA` plots easily in .pdf files. Please Enjoy.



Term	Overlap	P-valu
MP0000609_a	13/442	2.87E
MP0002118_a	14/694	6.78E
MP0006036_a	6/107	1.92E
MP0000955_a	8/245	5.78E
MP0005266_a	5/100	0.0011
MP0009765_a	5/108	0.0015
MP0002752_a	14/676	5.25E
MP0000598_a	12/586	0.0015
MP0005332_a	5/134	0.0037

Figure 1 Enrichr workflow. Enrichr receives lists of human or mouse genes as input. It uses 35 gene-set librarie
enrichment results are interactively displayed as bar graphs, tables, grids of terms with the enriched terms highlig

(continued from previous page)

```
CTLA2B
SCARA3
LOC100044683
CMBL
CLIC6
IL13RA1
TACSTD2
DKKL1
CSF1
CITED1
SYNPO2L
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]:
```
'1.0.3'
```

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.head()
```

[4]:
```
   ensembl_gene_id external_gene_name  entrezgene_id       go_id
0  ENSG00000125285              SOX21          11166  GO:0006355
1  ENSG00000125285              SOX21          11166  GO:0005634
2  ENSG00000125285              SOX21          11166  GO:0003677
3  ENSG00000125285              SOX21          11166  GO:0003700
4  ENSG00000125285              SOX21          11166  GO:0000981
```

Mouse gene symbols maps to Human, or Vice Versa

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

[5]:
```
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'])
```

[6]:
```
# 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.

[7]:
```
# 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']
```

[8]:
```
kegg_mouse = {}
for term, genes in kegg.items():
    new_genes = []
    for gene in genes:
        if gene in h2m_dict:
            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 Enrichr API

See all supported enrichr library names

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

[9]:
```
# default: Human
names = gp.get_library_name()
names[:10]
```

[9]:
```
['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']
```

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

[10]:
```
['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',
```

(continues on next page)

(continued from previous page)

```
 'GO_Cellular_Component_AutoRIF',
 'GO_Cellular_Component_AutoRIF_Predicted_zscore',
 'GO_Molecular_Function_2018',
 'GO_Molecular_Function_AutoRIF']
```

Parse Enrichr library into dict

[11]:

```
## 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']
```

[12]:

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

[12]:

```
           0
0    IGKV4-1
1       CD55
2       IGKC
3    PPFIBP1
4      ABHD4
```

[13]:

```
# 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 Sevices need `gene symbols` as input 2. `Gene symbols` will convert to upcases automatically.

[14]:

```
# run enrichr
# 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
                )
```

[15]:

```
# obj.results stores all results
enr.results.head(5)
```

[15]:

```
              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

   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...
```

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 identifers** 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',...],
           ...}
```

[16]:
```
# 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, #"hsapiens_gene_ensembl",
                 outdir=None,
                 verbose=True)
```

```
2022-12-18 15:22:47,130 [INFO] User defined gene sets is given: ./tests/data/genes.gmt
2022-12-18 15:22:47,132 [INFO] Input dict object named with gs_ind_2
2022-12-18 15:22:47,670 [WARNING] Input library not found: unknown. Skip
2022-12-18 15:22:47,672 [INFO] Run: genes.gmt
2022-12-18 15:22:47,673 [INFO] Background is not set! Use all 682 genes in genes.gmt.
2022-12-18 15:22:47,680 [INFO] Run: gs_ind_2
2022-12-18 15:22:47,724 [INFO] Background is not set! Use all 7010 genes in gs_ind_2.
2022-12-18 15:22:47,962 [INFO] Done.
```

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 experment. e.g. the expressed genes in your RNA-seq.
   - The gene identifer in gmt/dict should be the same type to the backgound 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 choosen.
   - The program will try to retrieve the background information automatically.

Plotting

Show top 5 terms of each gene_set ranked by “Adjusted P-value”

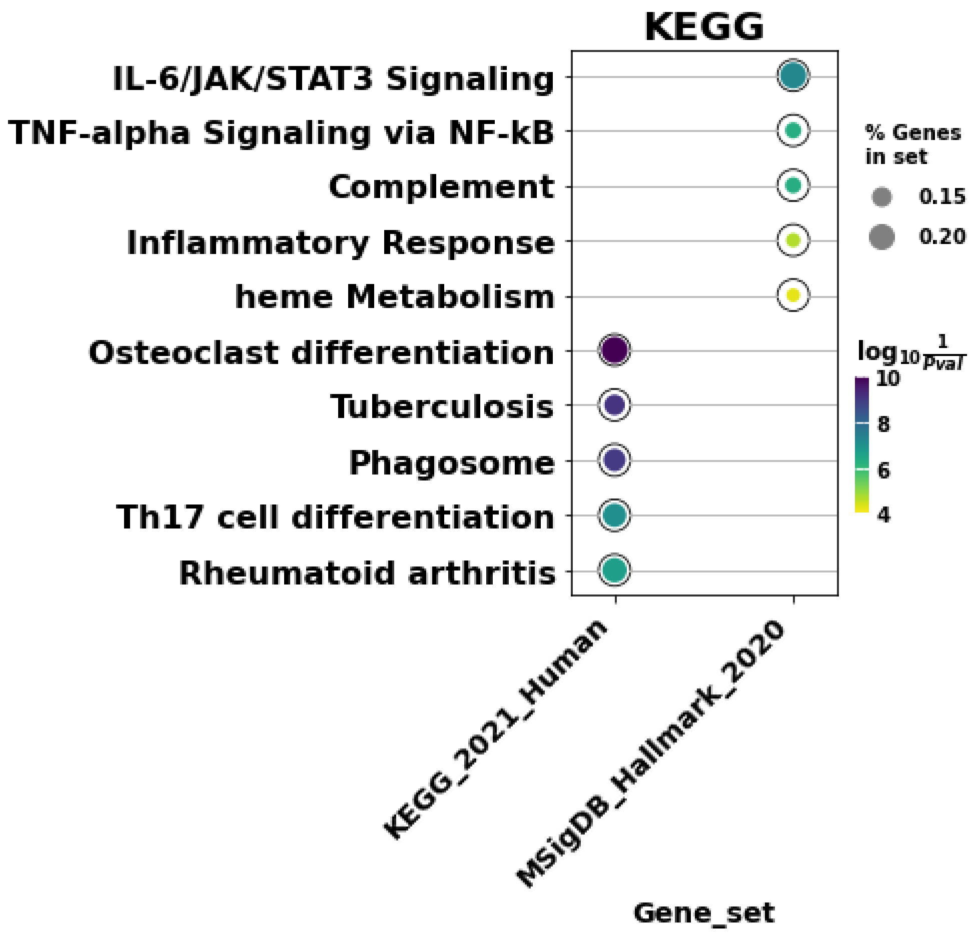
[17]:
```
# simple plotting function
from gseapy import barplot, dotplot
```

[18]:
```
# categorical scatterplot
ax = dotplot(enr.results,
              column="Adjusted P-value",
```

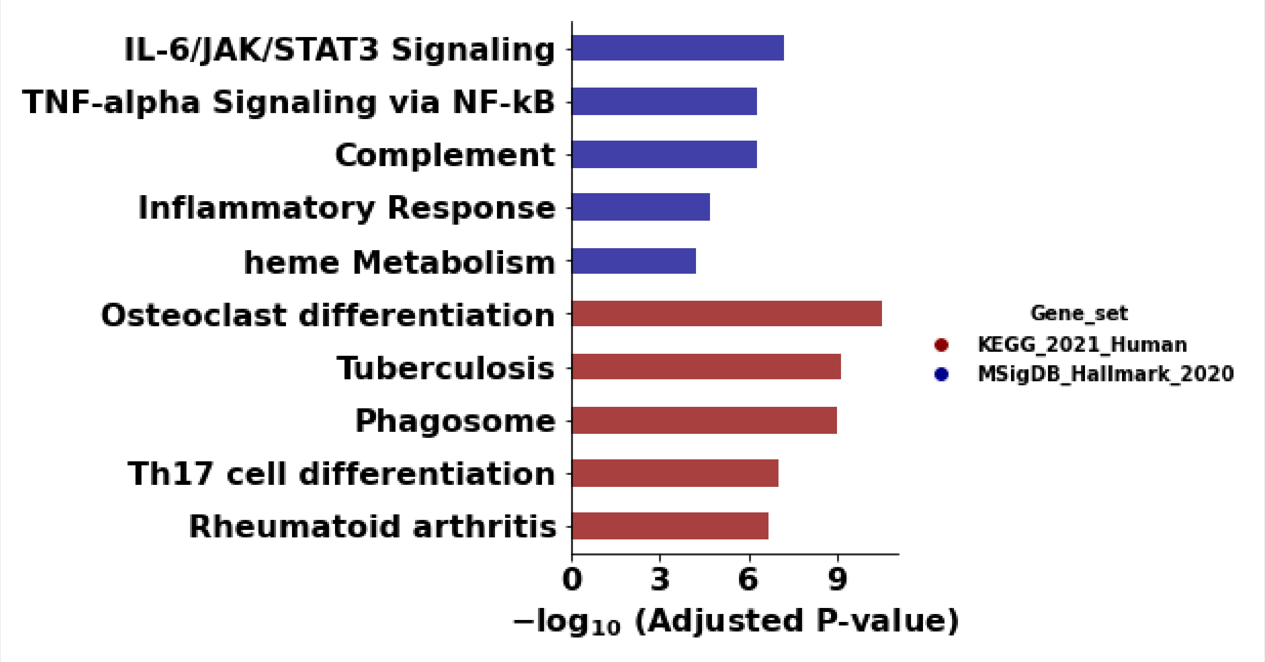
(continues on next page)

(continued from previous page)

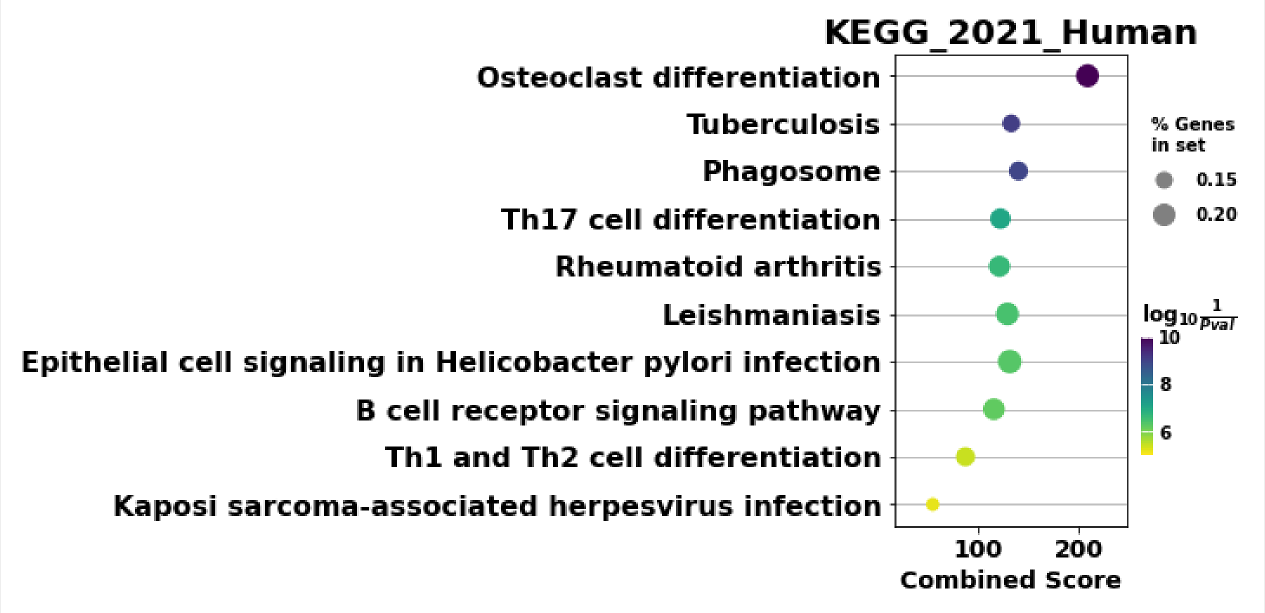
```
              x='Gene_set', # set x axis, so you could do a multi-sample/library comparsion
              size=10,
              top_term=5,
              figsize=(3,5),
              title = "KEGG",
              xticklabels_rot=45, # rotate xtick labels
              show_ring=True, # set to False to revmove outer ring
              marker='o',
             )
```



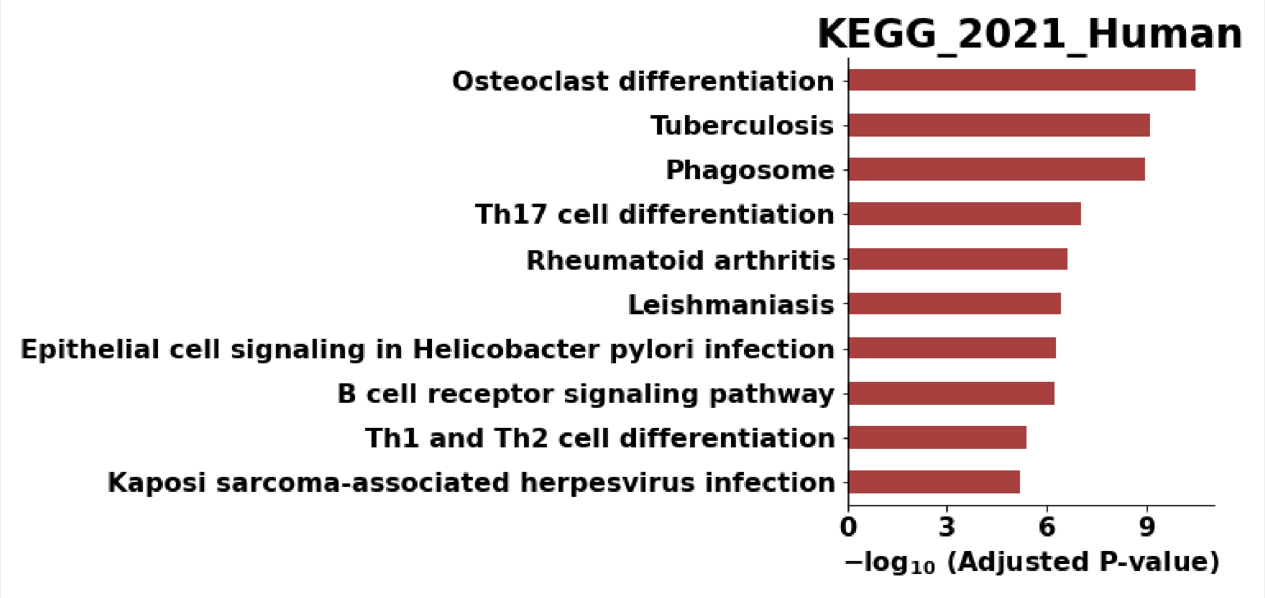
[19]:
```
# categorical scatterplot
ax = barplot(enr.results,
              column="Adjusted P-value",
              group='Gene_set', # set group, so you could do a multi-sample/library comparsion
              size=10,
              top_term=5,
              figsize=(3,5),
              color=['darkred', 'darkblue'] # set colors for group
             )
```



```
[20]: # 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))
```



```
[21]: # 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

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

5.2.3 Prerank example

Assign prerank() with

- pd.DataFrame: Only contains two columns, or one cloumn 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 Libaries. 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',...],
           ...}
```

[23]:
```
rnk = pd.read_csv("./tests/data/temp.rnk", header=None, index_col=0, sep="\t")
rnk.head()
```

[23]:
```
                1
0
ATXN1   16.456753
UBQLN4  13.989493
CALM1   13.745533
DLG4    12.796588
MRE11A  12.787631
```

[24]:
```
rnk.shape
```

[24]:
```
(22922, 1)
```

[25]:
```
# # 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
                    )
```

```
2022-12-18 15:22:49,042 [WARNING] Duplicated values found in preranked stats: 4.97%␣
↪of genes
The order of those genes will be arbitrary, which may produce unexpected results.
2022-12-18 15:22:49,042 [INFO] Parsing data files for GSEA...
2022-12-18 15:22:49,044 [INFO] Enrichr library gene sets already downloaded in: /home/
↪fangzq/.cache/gseapy, use local file
2022-12-18 15:22:49,060 [INFO] 0001 gene_sets have been filtered out when max_
↪size=1000 and min_size=5
2022-12-18 15:22:49,060 [INFO] 0292 gene_sets used for further statistical testing...
2022-12-18 15:22:49,061 [INFO] Start to run GSEA...Might take a while...
2022-12-18 15:23:02,690 [INFO] Congratulations. GSEApy runs successfully...
```

How to generate your GSEA plot inside python console

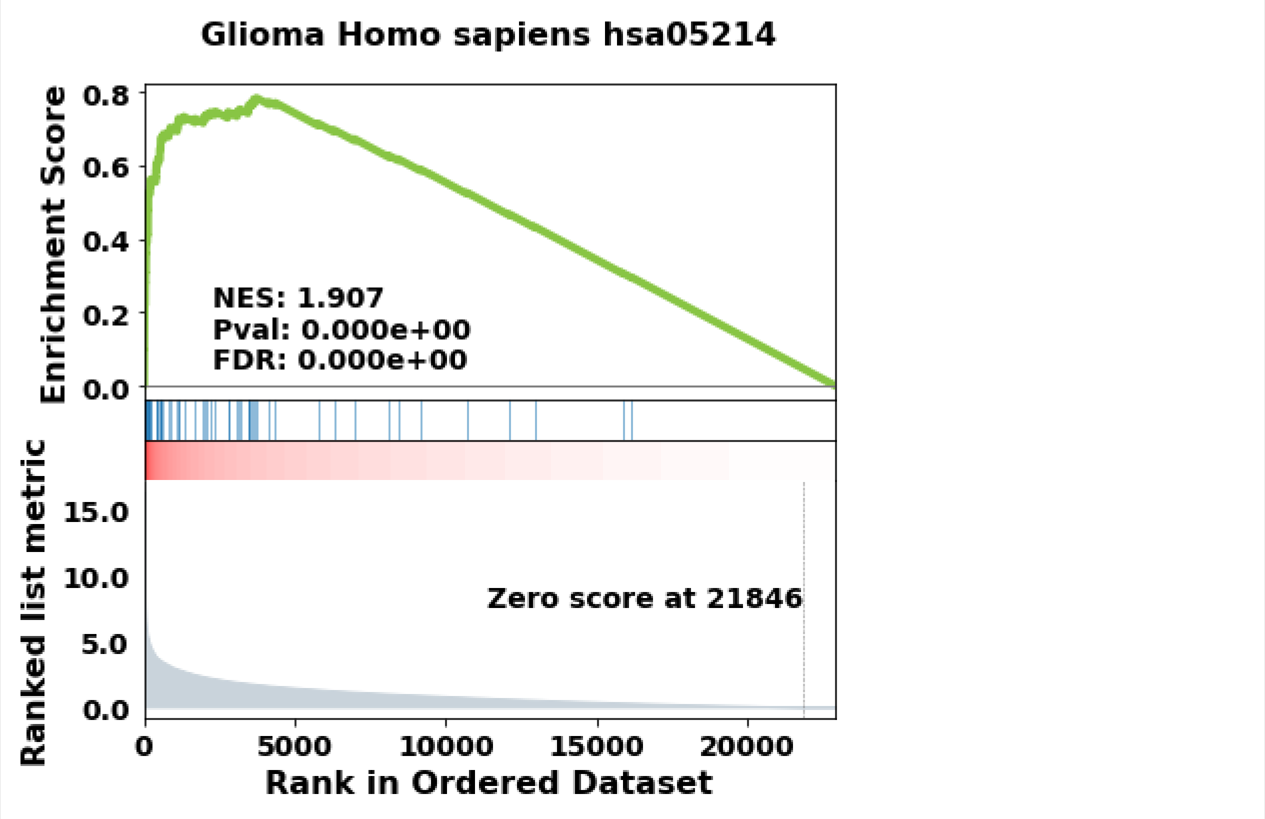
Visualize it using `gseaplot`

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

[26]:
```
pre_res.res2d.head(5)
```

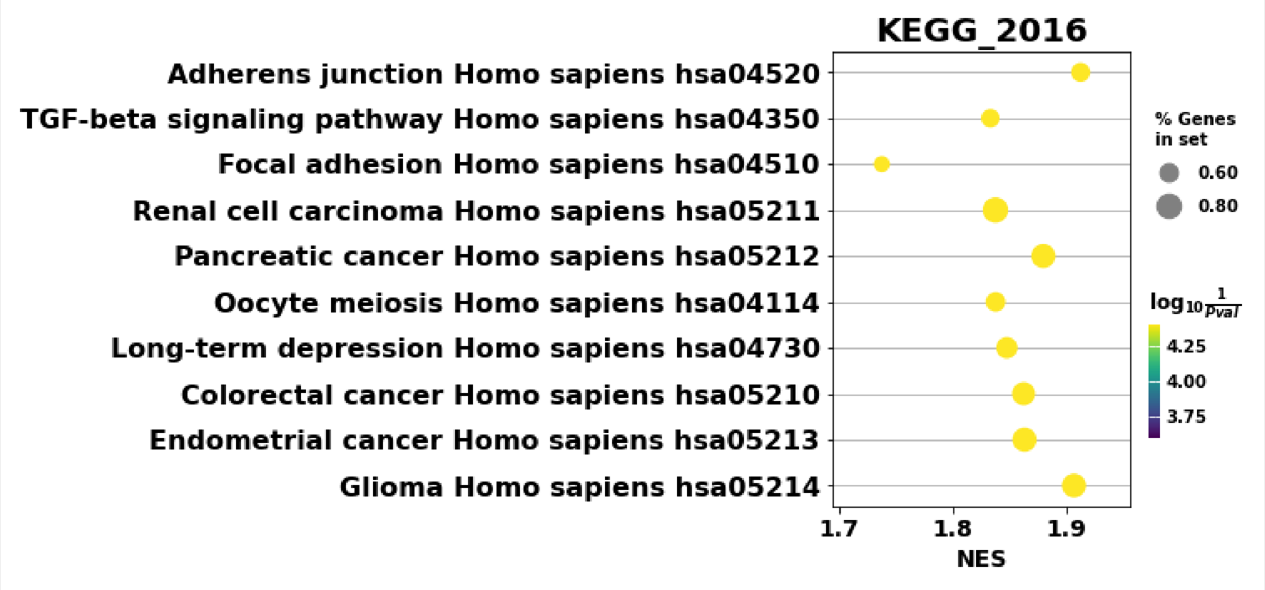
[26]:
```
      Name                                          Term        ES  \
0  prerank            Adherens junction Homo sapiens hsa04520  0.784625
1  prerank                       Glioma Homo sapiens hsa05214  0.784678
2  prerank   Estrogen signaling pathway Homo sapiens hsa04915  0.766347
3  prerank  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;TGFBR1;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...
```

[27]:
```
## easy way
from gseapy import gseaplot
terms = pre_res.res2d.Term
i = 1
# to save your figure, make sure that ofname is not None
gseaplot(rank_metric=pre_res.ranking,
         term=terms[i],
         **pre_res.results[terms[i]])

# save figure
# gseaplot(rank_metric=pre_res.ranking, term=terms[0], ofname='your.plot.pdf', **pre_
↪res.results[terms[0]])
```



dotplot for GSEA resutls

```
[28]: 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=3, # 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.

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

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

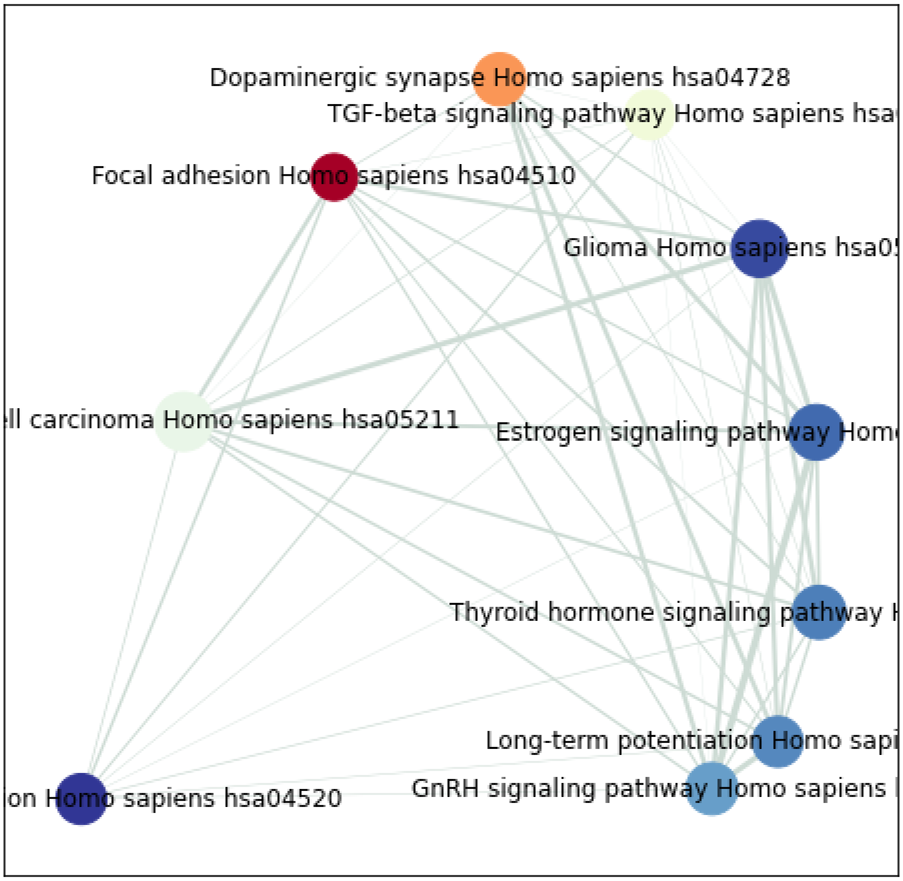
```
[31]: # build graph
      G = nx.from_pandas_edgelist(edges,
                                  source='src_idx',
                                  target='targ_idx',
                                  edge_attr=['jaccard_coef', 'overlap_coef', 'overlap_genes
      ↪'])
```

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

      # init node cooridnates
      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()
```

(continues on next page)

(continued from previous page)

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


Dopaminergic synapse Homo sapiens hsa04728
TGF-beta signaling pathway Homo sapiens hsa
Focal adhesion Homo sapiens hsa04510
Glioma Homo sapiens hsa05
ll carcinoma Homo sapiens hsa05211
Estrogen signaling pathway Homo
Thyroid hormone signaling pathway H
Long-term potentiation Homo sapi
GnRH signaling pathway Homo sapiens
on Homo sapiens hsa04520

Command line usage

You may also want to use prerank in command line

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

5.2.4 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 Libaries. 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

[34]:
```
phenoA, phenoB, class_vector =  gp.parser.gsea_cls_parser("./tests/extdata/Leukemia.
↪cls")
```

[35]:
```
#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',
↪'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML',
↪'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML', 'AML']
```

[36]:
```
gene_exp = pd.read_csv("./tests/extdata/Leukemia_hgu95av2.trim.txt", sep="\t")
gene_exp.head()
```

[36]:
```
      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

    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
```

(continues on next page)

(continued from previous page)

```
4   257.0   439.0    386.0

[5 rows x 50 columns]
```

[37]:

```
print("positively correlated: ", phenoA)
```

```
positively correlated:  ALL
```

[38]:

```
print("negtively correlated: ", phenoB)
```

```
negtively correlated:  AML
```

[39]:

```
# 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)
```

```
2022-12-18 15:23:03,861 [WARNING] Dropping duplicated gene names, only keep the first␣
↪values
```

[40]:

```
#access the dataframe results throught res2d attribute
gs_res.res2d.head()
```

[40]:

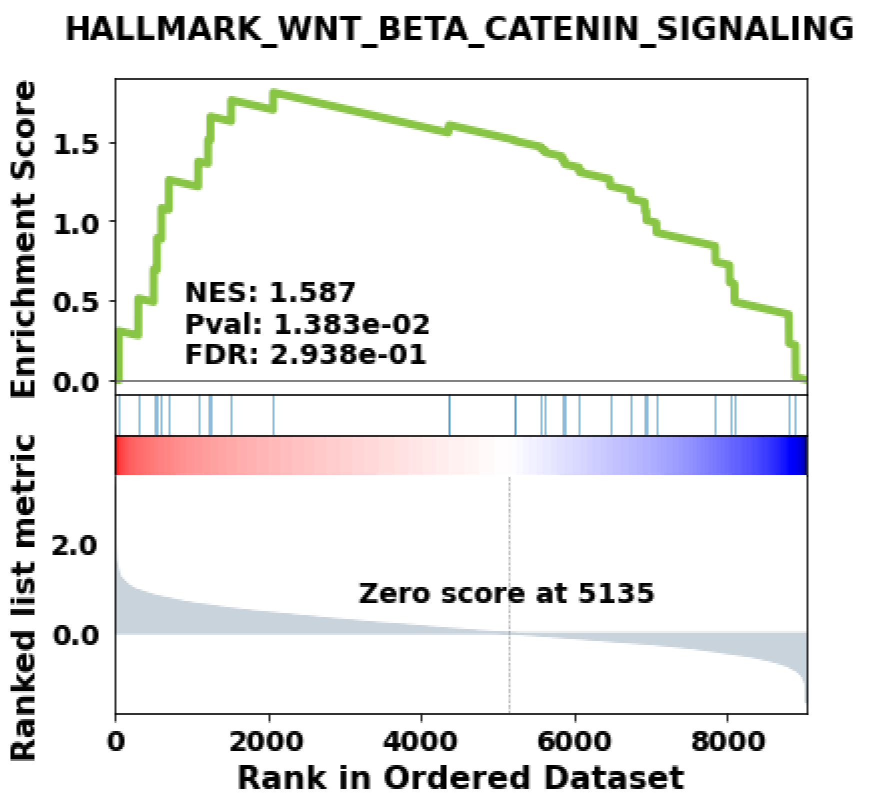
```
  Name                                Term        ES       NES NOM p-val  \
0 gsea                  HALLMARK_E2F_TARGETS  0.574187  1.661335  0.052521
1 gsea                HALLMARK_MITOTIC_SPINDLE  0.430183  1.646924  0.026804
2 gsea  HALLMARK_WNT_BETA_CATENIN_SIGNALING  0.438876  1.586567  0.013834
3 gsea     HALLMARK_TNFA_SIGNALING_VIA_NFKB  -0.49294 -1.521229  0.111562
4 gsea                HALLMARK_MYC_TARGETS_V1  0.535105  1.519305  0.156448

  FDR q-val FWER p-val    Tag %   Gene %  \
0  0.577605      0.259   87/151   23.65%
1   0.31929      0.279   84/147   37.31%
2  0.293792      0.353    11/30   22.99%
3       1.0   0.466934  104/177   28.92%
4  0.341741      0.481  115/174   33.61%

                                        Lead_genes
0  DCK;BARD1;NASP;SRSF2;STMN1;SRSF1;TRA2B;EZH2;SM...
1  SPTAN1;SEPT9;ATG4B;SMC1A;MYH10;BIN1;CYTH2;TUBG...
2  LEF1;SKP2;HDAC2;GNAI1;CUL1;MAML1;WNT1;HDAC5;AX...
3  MCL1;CEBPB;PLAU;IL18;PLEK;BCL3;CEBPD;PLAUR;JUN...
4  HNRNPA3;HDDC2;RFC4;SRSF2;SRSF1;TRA2B;RRM1;HNRN...
```

Show the gsea plots

The **gsea** module will generate heatmap for genes in each gene sets in the backgroud.
But if you need to do it yourself, use the code below

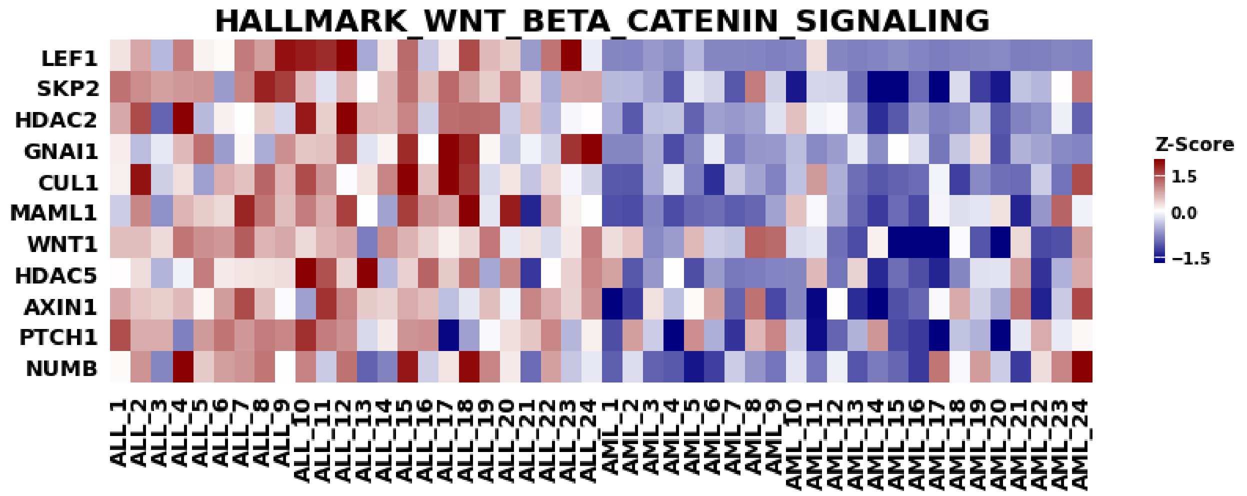
[41]:

```
from gseapy import gseaplot, heatmap
terms = gs_res.res2d.Term
i = 2
# Make sure that ``ofname`` is not None, if you want to save your figure to disk
gseaplot(gs_res.ranking, term=terms[i], **gs_res.results[terms[i]])
```

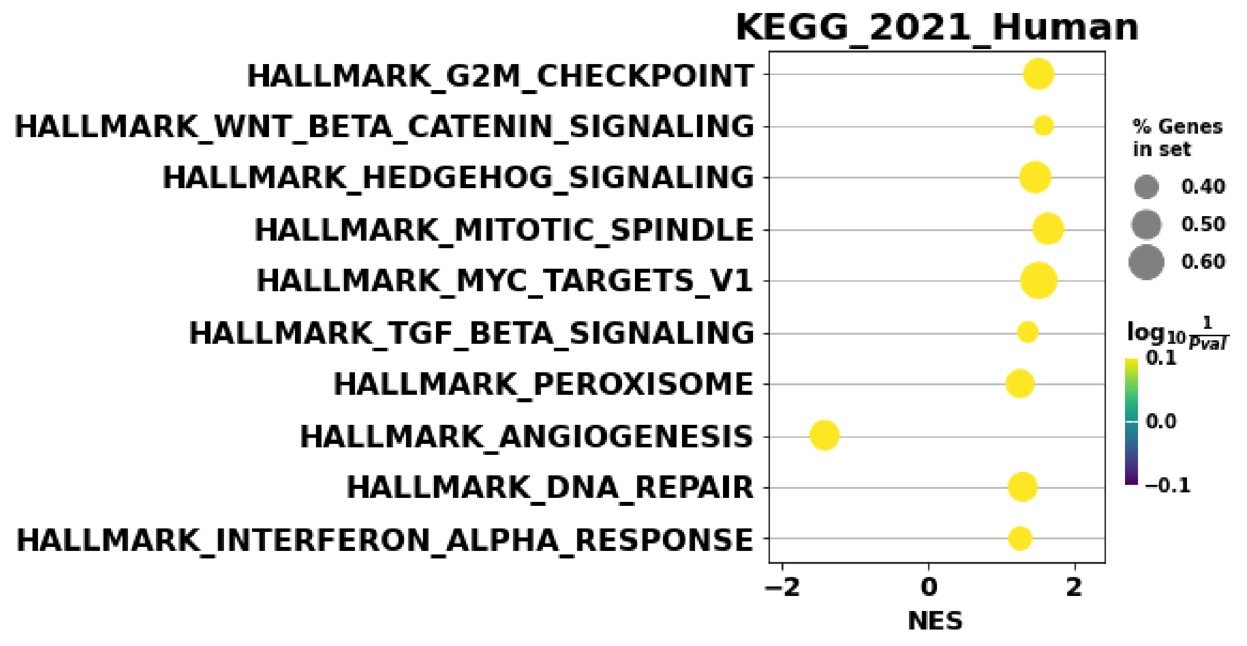


[42]:

```
# plotting heatmap
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.heatmat.loc[genes], z_score=0, title=terms[i], figsize=(14,
↪4))
```



[43]:
```
from gseapy import dotplot, ringplot
# 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

[44]:
```
# !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
```

5.2.5 Single Sample GSEA example

Note: When you run ssGSEA, all genes names in your gene_sets file should be found in your expression table

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",
```

(continues on next page)

(continued from previous page)

```
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 Libaries. 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

[45]:

```
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)
```

```
2022-12-18 15:23:08,013 [WARNING] Dropping duplicated gene names, values averaged by␣
↪gene names!
```

[46]:

```
ss.res2d.head()
```

[46]:

```
     Name                    Term           ES       NES
0   ALL_2  HALLMARK_MYC_TARGETS_V1  3483.418994  0.628864
1  ALL_12  HALLMARK_MYC_TARGETS_V1  3479.271006  0.628115
2  ALL_14  HALLMARK_MYC_TARGETS_V1  3444.345325   0.62181
3  AML_11  HALLMARK_MYC_TARGETS_V1  3428.399787  0.618931
4  ALL_17  HALLMARK_MYC_TARGETS_V1  3390.155261  0.612027
```

[47]:

```
# 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()
```

[47]:

```
                1
0
ATXN1   16.456753
UBQLN4  13.989493
CALM1   13.745533
DLG4    12.796588
MRE11A  12.787631
```

[48]:

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

[49]:

```
# 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

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

```
[50]:        Name                                 Term           ES       NES
      769    ALL_1        HALLMARK_BILE_ACID_METABOLISM -1076.226223 -0.194292
      1826   ALL_1           HALLMARK_ANDROGEN_RESPONSE   366.565793  0.066176
      499    ALL_1   HALLMARK_INTERFERON_ALPHA_RESPONSE   1374.47258  0.248134
      1260   ALL_1       HALLMARK_INFLAMMATORY_RESPONSE  -686.472877 -0.123929
      520    ALL_1             HALLMARK_SPERMATOGENESIS -1350.143332 -0.243742
```

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

```
[51]: Name                            ALL_1    ALL_10    ALL_11    ALL_12  \
      Term
      HALLMARK_ADIPOGENESIS        0.196548   0.17984  0.206196  0.196183
      HALLMARK_ALLOGRAFT_REJECTION -0.035737 -0.079017  0.001587 -0.021818
      HALLMARK_ANDROGEN_RESPONSE    0.066176  0.026662  0.097368  0.111017
      HALLMARK_ANGIOGENESIS        -0.212581 -0.277432 -0.259029 -0.185131
      HALLMARK_APICAL_JUNCTION     -0.069044 -0.057257 -0.073727 -0.090612

      Name                           ALL_13    ALL_14    ALL_15    ALL_16  \
      Term
      HALLMARK_ADIPOGENESIS        0.226729   0.22492  0.192256  0.166602
      HALLMARK_ALLOGRAFT_REJECTION -0.024654 -0.003921  0.017489  0.037328
      HALLMARK_ANDROGEN_RESPONSE    0.076096  0.046818  0.085992   0.06848
      HALLMARK_ANGIOGENESIS         -0.24831 -0.207648 -0.194121 -0.245831
      HALLMARK_APICAL_JUNCTION     -0.068634 -0.057682 -0.046341 -0.073614

      Name                           ALL_17    ALL_18  ...    AML_22    AML_23  \
      Term                                             ...
      HALLMARK_ADIPOGENESIS        0.223924  0.198694  ...  0.188801  0.175198
      HALLMARK_ALLOGRAFT_REJECTION 0.019548 -0.013216  ...  0.065169  0.046979
      HALLMARK_ANDROGEN_RESPONSE   0.100885  0.091473  ...   0.11494  0.122153
      HALLMARK_ANGIOGENESIS       -0.159116  -0.21193  ... -0.029889 -0.126933
      HALLMARK_APICAL_JUNCTION    -0.078191  -0.05357  ... -0.017849 -0.036942

      Name                           AML_24     AML_3     AML_4     AML_5  \
      Term
      HALLMARK_ADIPOGENESIS          0.1005  0.217502  0.251881  0.153244
      HALLMARK_ALLOGRAFT_REJECTION -0.027275  0.113908  0.065444  0.087849
      HALLMARK_ANDROGEN_RESPONSE    0.100573  0.116135  0.063621   0.10465
      HALLMARK_ANGIOGENESIS        -0.020275 -0.161988 -0.054431 -0.093919
      HALLMARK_APICAL_JUNCTION      0.048176 -0.041205 -0.023632 -0.018138

      Name                            AML_6     AML_7     AML_8     AML_9
      Term
      HALLMARK_ADIPOGENESIS        0.197175  0.327672  0.315413   0.25545
      HALLMARK_ALLOGRAFT_REJECTION 0.049636  0.033845 -0.007233  0.013453
      HALLMARK_ANDROGEN_RESPONSE   0.058016  0.086909  0.109792   0.12101
      HALLMARK_ANGIOGENESIS       -0.057732 -0.119079 -0.121271 -0.156819
      HALLMARK_APICAL_JUNCTION     0.011795 -0.040168 -0.044827 -0.044005

      [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

[52]:

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

5.2.6 Replot Example

Locate your directory

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

```
data
 |--- edb
 |     |--- C1OE.cls
 |     |--- gene_sets.gmt
 |     |--- gsea_data.gsea_data.rnk
 |     |--- results.edb
```

[53]:

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

Command line usage of replot

[54]:

```
# !gseapy replot -i data -o test/replot_test
```

[ ]:

```

```

5.3 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]:
```
'1.0.2'
```

5.3.1 Read Demo Data

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

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

[5]:
```
                   orig.ident  nCount_RNA  nFeature_RNA  stim  \
AAACATACATTTCC.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
AAACATACATTTCC.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
DC                  CTRL     258
                    STIM     214
Eryth               STIM      32
                    CTRL      23
```

(continues on next page)

(continued from previous page)

```
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_annotations', 'stim']).index
adata = adata[indices,:]
```

[10]:

```
# # # subset and write GCT and CLS file
# outdir = "ifnb/"
# for cell in adata.obs.seurat_annotations.unique():
#     bdata = adata[adata.obs.seurat_annotations == 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_annotations == "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'
```

5.3.2 GSEA

[12]:
```
import time
t1 = time.time()
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)
```

```
64.49783539772034
```

[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  140/490   9.03%
1   3.66143       0.0       0.0        0.0   56/188   6.30%
2  3.549856       0.0       0.0        0.0   49/140   8.77%
3  3.438759       0.0       0.0        0.0   42/108   2.85%
4   3.37735       0.0       0.0        0.0   37/120   7.26%
5  3.362051       0.0       0.0        0.0   49/100   4.90%
6  3.327923       0.0       0.0        0.0    49/99   7.18%
7  3.259412       0.0       0.0        0.0    14/44   4.94%
8 -3.225436       0.0       0.0        0.0  128/234  19.45%
9 -3.219153       0.0       0.0        0.0  134/322  10.13%

                                          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...
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
```

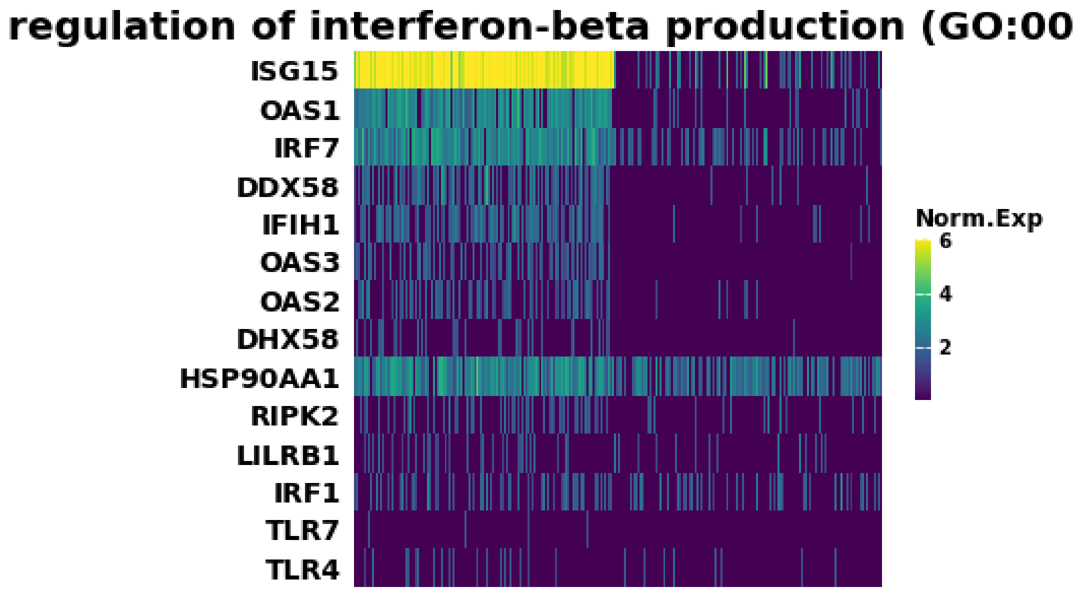
[14]:
```
(13216,)
```

[15]:

```
## Heatmap of gene expression
i = 7
genes = res.res2d.Lead_genes.iloc[i].split(";")
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)
```

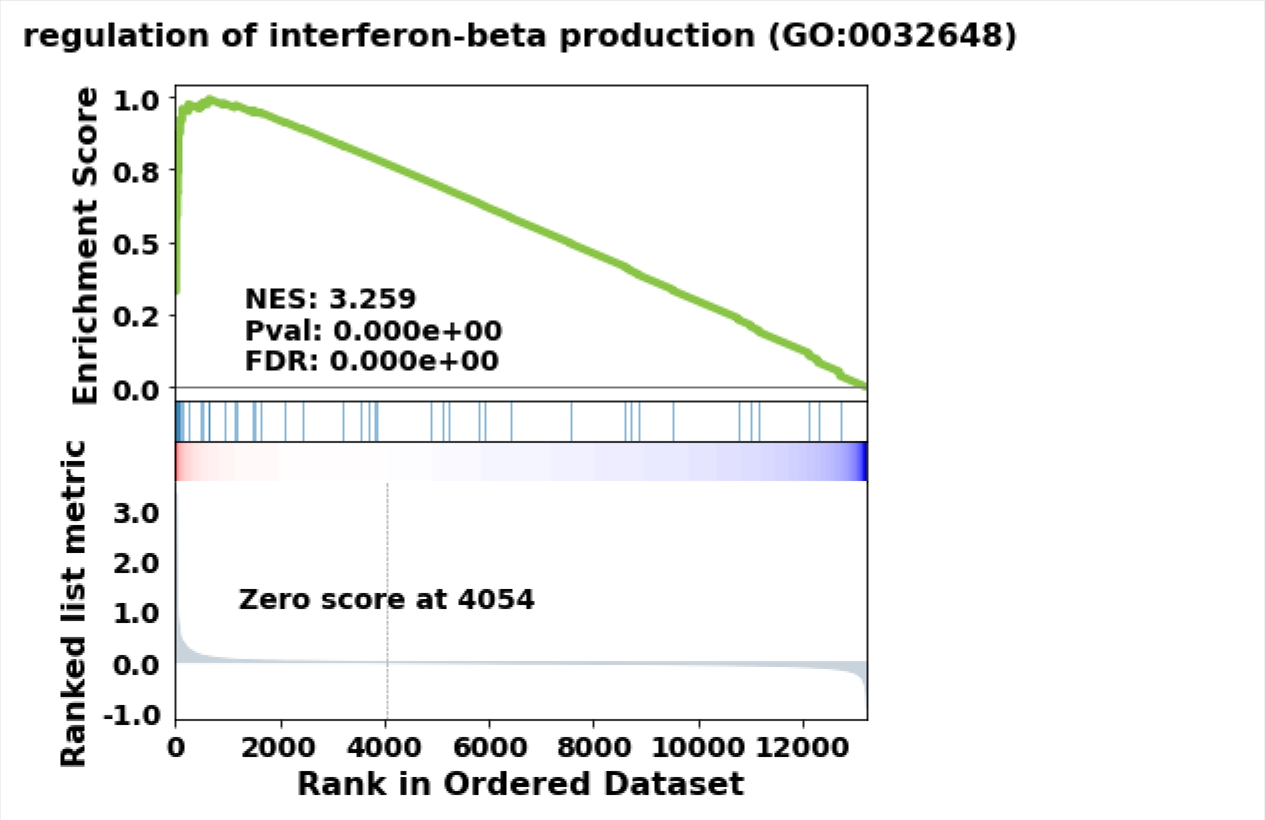
[15]:

```
<AxesSubplot:title={'center':'regulation of interferon-beta production (GO:0032648)'}>
```



[16]:

```
term = res.res2d.Term.iloc[i]
gp.gseaplot(res.ranking, term=term, **res.results[term])
```



5.3.3 DEG Analysis

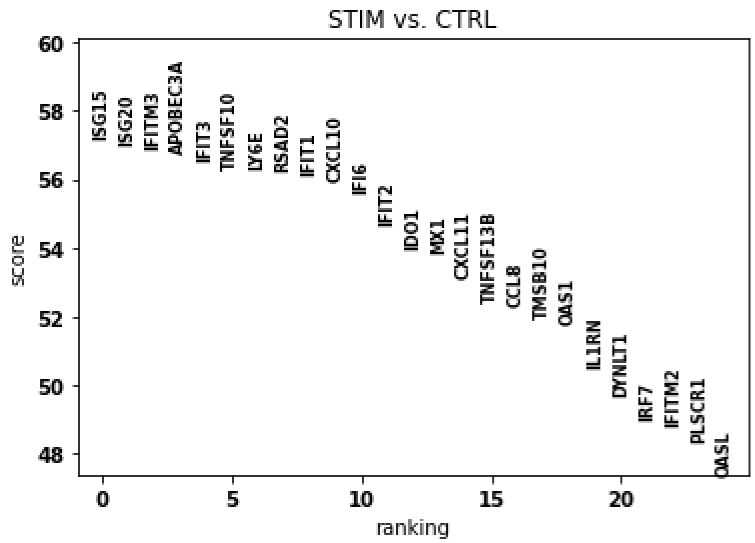
```
[17]: # find degs
      sc.tl.rank_genes_groups(bdata,
                              groupby='stim',
                              use_raw=False,
                              layer='lognorm',
                              method='wilcoxon',
                              groups=["STIM"],
       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

```
... storing 'orig.ident' as categorical
... storing 'seurat_annotations' as categorical
```

```
[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()
```

```
[21]:   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)
```

5.3.4 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)
```

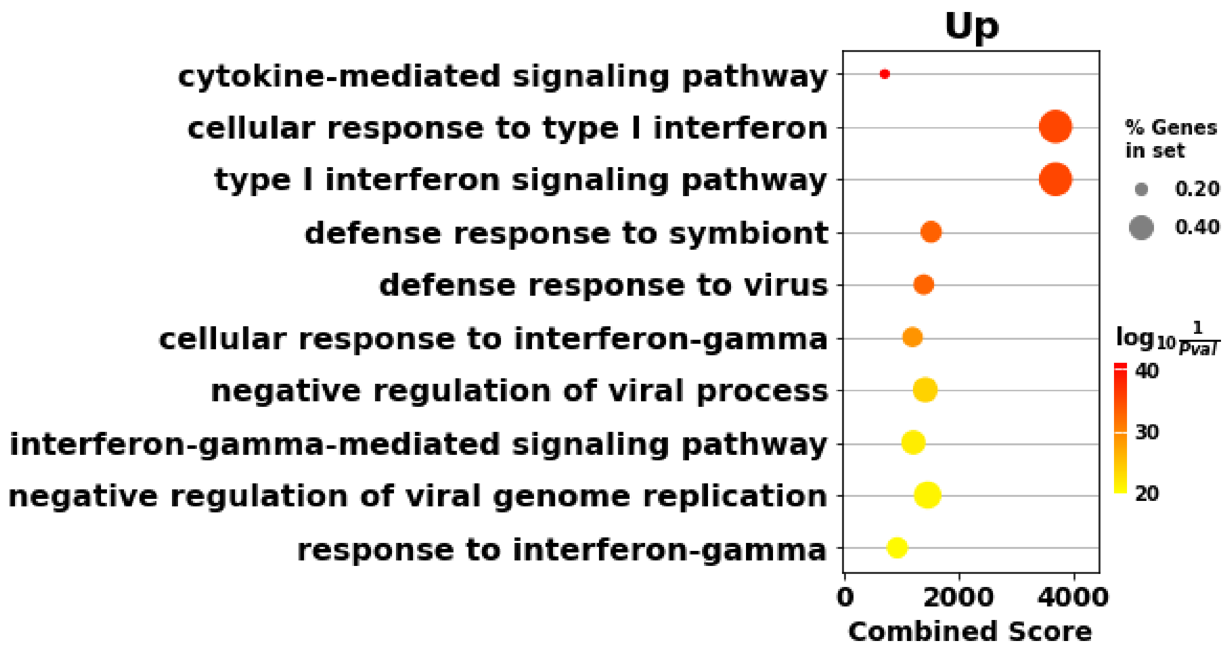
```
[25]: degs_dw.shape
```

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

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

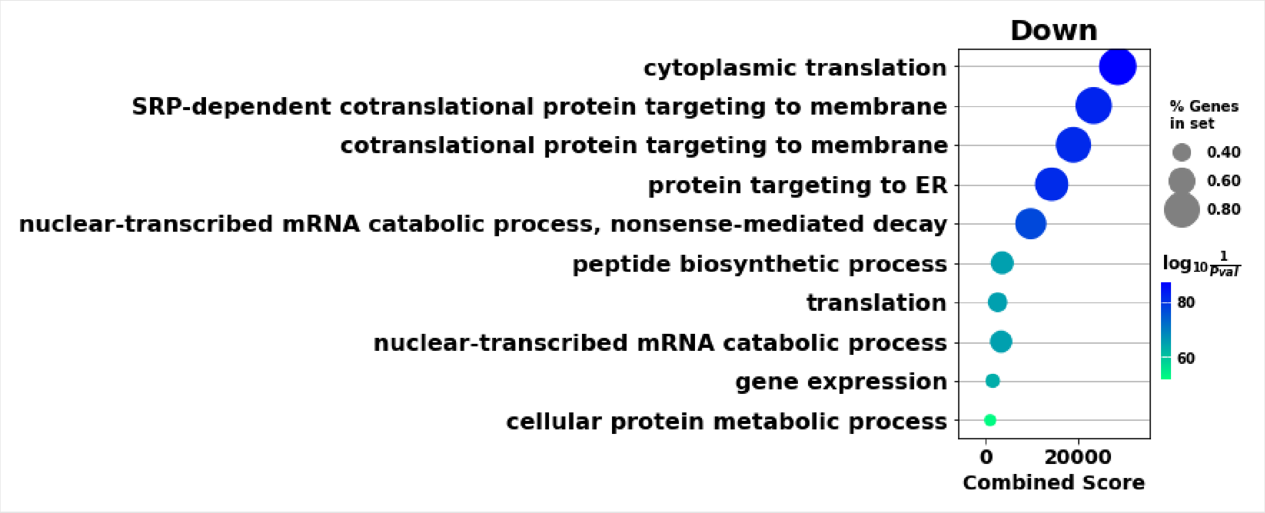
```
[27]: # trim (go:...)
enr_up.res2d.Term = enr_up.res2d.Term.str.split(" \(GO").str[0]
```

```
[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)
```

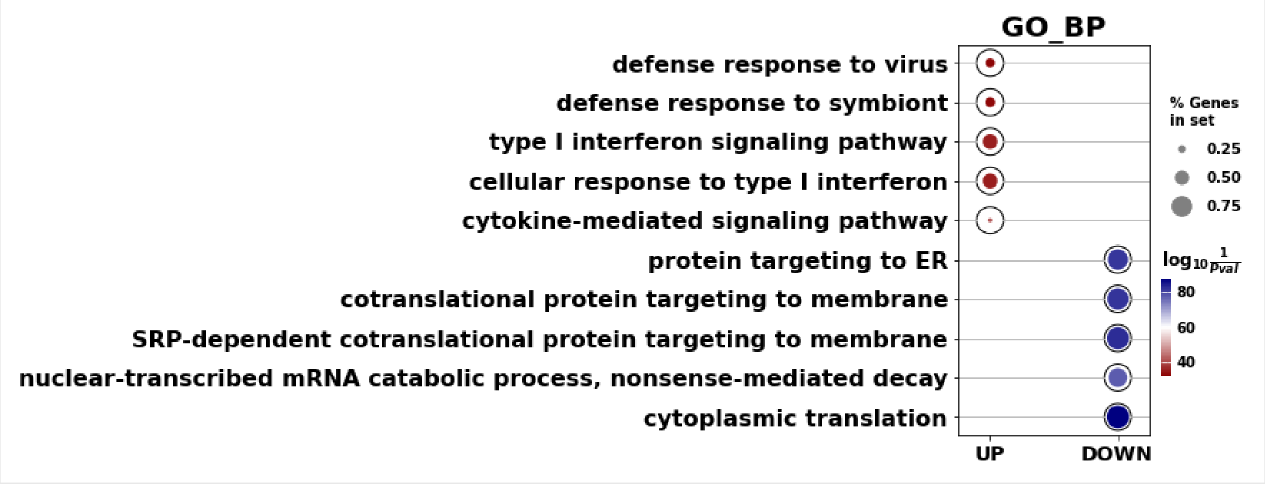
```
[30]: enr_dw.res2d.Term = enr_dw.res2d.Term.str.split(" \(GO").str[0]
gp.dotplot(enr_dw.res2d,
           figsize=(3,5),
           title="Down",
           cmap = plt.cm.winter_r,
           size=5)
plt.show()
```



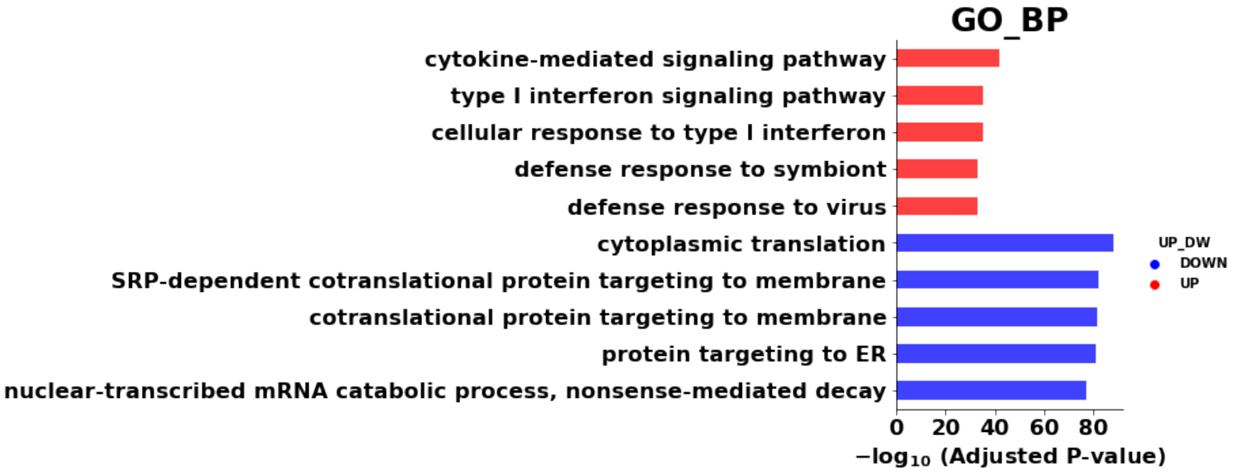
```
[31]: # concat results
      enr_up.res2d['UP_DW'] = "UP"
      enr_dw.res2d['UP_DW'] = "DOWN"
      enr_res = pd.concat([enr_up.res2d.head(), enr_dw.res2d.head()])
```

```
[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'])
```



5.3.5 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  EIF4A3,POLR2B,U2AF1,HNRNPU,CDC40,POLR2L,SRRM1,...
1      0.928571  PABPC4,RPL15,RPL24,RPS20,POLR2F,RPS27,MRPS12,R...
2      0.051020                 POLR2E,POLR2J,POLR2G,POLR2L,POLR2F
3      0.428571                      OAS2,OAS1,TLR4,OAS3,IRF1,IRF7
4      0.714286  OAS2,IFIH1,OAS1,ISG15,OAS3,LILRB1,TLR7,DDX58,I...
```

[41]:

```
# build graph
G = nx.from_pandas_edgelist(edges,
```

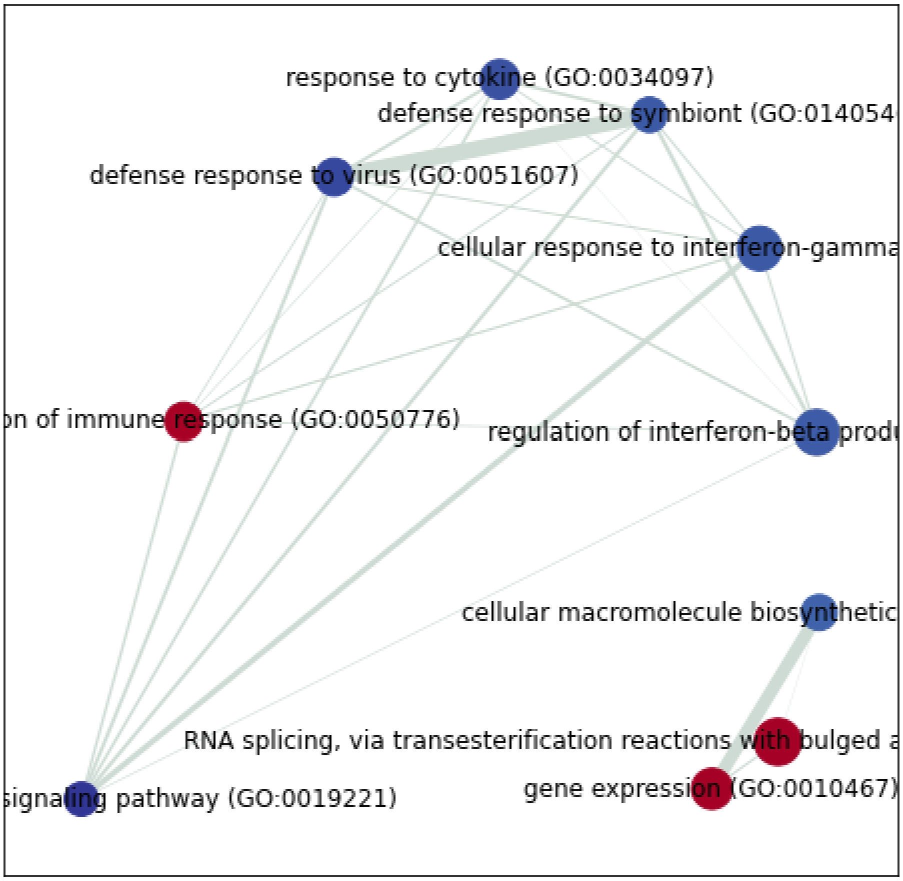
(continues on next page)

(continued from previous page)

```
                            source='src_idx',
                            target='targ_idx',
                            edge_attr=['jaccard_coef', 'overlap_coef', 'overlap_genes
↪'])
```

[42]:

```
fig, ax = plt.subplots(figsize=(8, 8))

# init node cooridnates
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()
```


response to cytokine (GO:0034097)
defense response to symbiont (GO:014054
defense response to virus (GO:0051607)
cellular response to interferon-gamma
on of immune response (GO:0050776)
regulation of interferon-beta produ
cellular macromolecule biosynthetic
RNA splicing, via transesterification reactions with bulged a
signaling pathway (GO:0019221)
gene expression (GO:0010467)

```
[ ]:
```

5.4 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
```

5.4.1 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/C1OE.cls') as cls:
    print(cls.read())

# or assign a list object to parameter 'cls' like this
# cls=['C1OE', 'C1OE', 'C1OE', 'Vector', 'Vector', 'Vector']
```

```
6 2 1
# C1OE Vector
C1OE C1OE C1OE 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 .. code:: python

> groups = [‘C1OE’, ‘C1OE’, ‘C1OE’, ‘Vector’, ‘Vector’, ‘Vector’] with open(‘gsea/edb/C1OE.cls’, “w”) as cl:
>
>> line = f”{len(groups)} 2 1n# C10E Vectorn” 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     1110036O03RIK   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
```

5.4.2 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
DKKL1
CSF1
CITED1
SYNPO2L
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',
'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',
```

(continues on next page)

(continued from previous page)

```
'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',
'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/

5.4.3 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*.

5.5 Developmental Guide

5.5.1 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. Minimial 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.
- **weighted_score_type** (*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. Minimial 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.
- **weighted_score_type** (*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'}. Default: rank.

1. 'rank': Rank your expression data, and transform by 10000*rank_dat/gene_numbers
2. 'log' : Do not rank, but transform data by log(data + exp(1)), while data = data[data<1] =1.
3. 'log_rank': Rank your expression data, and transform by log(10000*rank_dat/gene_numbers+ exp(1))
4. '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

- **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.
- **weighted_score_type** (*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),
```

(continues on next page)

(continued from previous page)

```
|  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

- **`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 identifer in gmt/dict should be the same type to the backgound 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 experment. e.g. the expressed genes in your RNA-seq. The gene identifer in gmt/dict should be the same type to the backgound 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.**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.
- **weighted_score_type** (*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'.

5.5.2 GSEA Statistics

class gseapy.gsea.**GSEA**(*data: Union[pandas.core.frame.DataFrame, str], gene_sets: Union[List[str], str, Dict[str, str]], classes: Union[List[str], str, Dict[str, str]], outdir: Optional[str] = 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

calculate_metric(*df: pandas.core.frame.DataFrame, method: str, pos: str, neg: str, classes: Dict[str, List[str]], ascending: bool*) → pandas.core.series.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 a pd.Series of correlation to class of each variable. Gene_name is index, and value is rankings.

visit here for more docs: http://software.broadinstitute.org/gsea/doc/GSEAUserGuideFrame.html

load_data(*cls_vec: List[str]*) → Tuple[pandas.core.frame.DataFrame, Dict]

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

run()
GSEA main procedure

class gseapy.gsea.**Prerank**(*rnk: Union[pandas.core.frame.DataFrame, pandas.core.series.Series, str], gene_sets: Union[List[str], str, Dict[str, str]], outdir: Optional[str] = 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.gsea.**SingleSampleGSEA**(*data: Union[pandas.core.frame.DataFrame, pandas.core.series.Series, str], gene_sets: Union[List[str], str, Dict[str, str]], outdir: Optional[str] = None, sample_norm_method: str = 'rank', min_size: int = 15, max_size: int = 500, permutation_num: Optional[int] = None, weight: float = 0.25, ascending: bool = False, threads: int = 1, figsize: Tuple[float, float] = (6.5, 6), format: str = 'pdf', graph_num: int = 20, no_plot: bool = True, seed: int = 123, verbose: bool = False*)
GSEA extension: single sample GSEA

corplot()
NES Correlation plot TODO

norm_samples(*dat: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame
normalization samples see here: http://rowley.mit.edu/caw_web/ssGSEAProjection/ssGSEAProjection.Library.R

run()
run entry

runSamplesPermu(*df: pandas.core.frame.DataFrame, gmt: Optional[Dict[str, List[str]]] = None*)
Single Sample GSEA workflow with permutation procedure

setplot()
ranked genes' location plot TODO

class gseapy.base.**GSEAbase**(*outdir: Optional[str] = None, gene_sets: Union[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.Offical API

load_gmt(*gene_list: Iterable[str], gmt: Union[List[str], str, Dict[str, str]]*) → Dict[str, List[str]]

load gene set dict

load_gmt_only(*gmt: Union[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

prepare_outdir()

create temp directory.

to_df(*gsea_summary: List[Dict], gmt: Dict[str, List[str]], rank_metric: Union[pandas.core.series.Series, pandas.core.frame.DataFrame], indices: Optional[List] = None*)

Convernt 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

5.5.3 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.

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

background genes number = a + b + c + d.

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)

`gseapy.stats.fdrcorrection`(*pvals*, *alpha=0.05*)

benjamini hocheberg 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

5.5.4 Enrichr API

class gseapy.enrichr.**Enrichr**(*gene_list: Iterable[str], gene_sets: Union[List[str], str, Dict[str, str]], organism: str = 'human', outdir: Optional[str] = 'Enrichr', background: Union[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)

Not implemented here:

combine score = log(p)·z

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

columns contain:

Term Overlap P-value 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 significance of the test, the hypergeometric distribution.

Parameters

- **gmt** – a dict of gene sets.
- **background** – list, set, or tuple. A list of custom backgound 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, pandas.core.frame.DataFrame]

Enrichr API

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(*gene_list, url*) → AnyStr

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

5.5.5 BioMart API

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

query from BioMart

add_filter(*name*, *value*)

key: filter names value: Iterable[str]

get_attributes(*dataset='hsapiens_gene_ensembl'*)

Get available attritbutes from dataset you've selected

get_datasets(*mart='ENSEMBL_MART_ENSEMBL'*)

Get available datasets from mart you've selected

get_filters(*dataset='hsapiens_gene_ensembl'*)

Get available filters from dataset you've selected

get_marts()

Get available marts and their names.

query(*dataset='hsapiens_gene_ensembl'*, *attributes=[]*, *filters={}*, *filename=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',
↪'entrezgene_id', 'go_id'],
                       filters=queries)
```

query_simple(*dataset='hsapiens_gene_ensembl'*, *attributes=[]*, *filters={}*, *filename=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',
```

(continues on next page)

(continued from previous page)

```
                                                        'hsapiens_homolog_associated_
↪gene_name',
                                                        'hsapiens_homolog_ensembl_gene
↪'])
```

5.5.6 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: Optional[List[str]] = 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 arugment 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

5.5.7 Visualization

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

gseapy.plot.**barplot**(*df: pandas.core.frame.DataFrame*, *column: str = 'Adjusted P-value'*, *group: Optional[str] = None*, *title: str = ''*, *cutoff: float = 0.05*, *top_term: int = 10*, *figsize: Tuple[float, float] = (4, 6)*, *color: Union[str, List[str]] = 'salmon'*, *ofname: Optional[str] = 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 of matplotlib.colors. Must be reconigzed by matplotlib.
- **ofname** – output file name. If None, don't save figure

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

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

Visualize GSEApy Results with categorical scatterplot When multiple datasets exist in the input dataframe, the *group* 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** – X-axis order to plot the *x* categorical levels. Default: None.

- **y_order** – Y-axis order to plot the *y* categorical levels. Default: None.
- **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
- **bool** (*show_ring*) – Whether to draw outer ring.

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

gseapy.plot.**enrichment_map**(*df: pandas.core.frame.DataFrame*, *column: str = 'Adjusted P-value'*, *cutoff: float = 0.05*, *top_term: int = 10*, ***kwargs*) → Tuple[pandas.core.frame.DataFrame, pandas.core.frame.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 positve 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**(*rank_metric: Iterable, term: str, hits: List[int], nes: float, pval: float, fdr: float, RES: float, pheno_pos: str = '', pheno_neg: str = '', figsize: Tuple[float] = (6, 5.5), cmap: str = 'seismic', ofname: Optional[str] = None, **kwargs*)

This is the main function for reproducing the gsea plot.

Parameters

- **rank_metric** – pd.Series for rankings, rank_metric.values.
- **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.
- **pheno_pos** – phenotype label, positive correlated.
- **pheno_neg** – phenotype label, negative correlated.
- **figsize** – matplotlib figsize.
- **ofname** – output file name. If None, don't save figure

gseapy.plot.**heatmap**(*df: pandas.core.frame.DataFrame*, *z_score: Optional[int] = None*, *title: str = ''*, *figsize: Tuple[float] = (5, 5)*, *cmap: Optional[str] = None*, *xticklabels: bool = True*, *yticklabels: bool = True*, *ofname: Optional[str] = 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.
- **xticklabels** – bool, whether to show xticklabels.
- **ofname** – output file name. If None, don't save figure

gseapy.plot.**ringplot**(*df: pandas.core.frame.DataFrame*, *column: str = 'Adjusted P-value'*, *x: Optional[str] = None*, *title: str = ''*, *cutoff: float = 0.05*, *top_term: int = 10*, *size: float = 5*, *figsize: Tuple[float] = (4, 6)*, *cmap: str = 'viridis_r'*, *ofname: Optional[str] = None*, *xticklabels_rot: Optional[float] = None*, *yticklabels_rot: Optional[float] = 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

- **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.`**`traceplot`**(*obj, terms: Union[str, List[str], None] = None, pheno_pos: str = ", pheno_neg: str = ", figsize: Tuple[float] = (6, 4), cmap: str = 'seismic', ofname: Optional[str] = None, **kwargs*)

Trace plot for terms

Parameters

- **obj** – GSEA or Prerank Object.
- **terms** – terms to show in trace plot

`gseapy.plot.`**`zscore`**(*data2d: pandas.core.frame.DataFrame, axis: Optional[int] = 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.

5.5.8 Scientific Journal and Sci- themed Color Palettes

5.5.9 Utils

5.6 Frequently Asked Questions

5.6.1 Q: What kind of gene identifiers are supported in GSEApy?

A:

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

5.6.2 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.

5.6.3 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 / (number of permutations) #. increase the permutation number (but more running time needed)

5.6.4 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.

5.6.5 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`

5.6.6 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"
                     )
```

5.6.7 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, ...)
```

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

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

5.6.9 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.

5.6.10 Q: ssGSEA missing p value and FDR?

A: The original ssGSEA alogrithm will not give you pval or FDR, so, please ignore the gseaplot generated by `ssgsea`. It's useless and misleading, therefore, fdr, and pval are not shown on the plot. If you'er 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.

5.6.11 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)*

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index