gseapy Documentation

Release 1.3.0

Zhuoqing Fang

Jun 29, 2026

TABLE OF CONTENTS

CHAPTER

ONE

GSEAPY: GENE SET ENRICHMENT ANALYSIS IN PYTHON.

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

CHAPTER

TWO

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

THREE

INSTALLATION

Install gseapy package from bioconda or pypi.

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

# or use pip
$ pip install gseapy
```

CHAPTER

FOUR

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

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

1. The `gsea` module produces **GSEA** results. The input 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 `gsva` module performs **GSVA** analysis, which described by Hänzelmann et al.

5. The `replot` module reproduces GSEA desktop version results. The only input for GSEAPY is the location to GSEA Desktop output results.

6. The `enrichr` module enables you to perform gene set enrichment analysis using `Enrichr` API. Enrichr is open source and freely available online at: http://amp.pharm.mssm.edu/Enrichr . It runs very fast and generates results in txt format.

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

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

FIVE

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
```

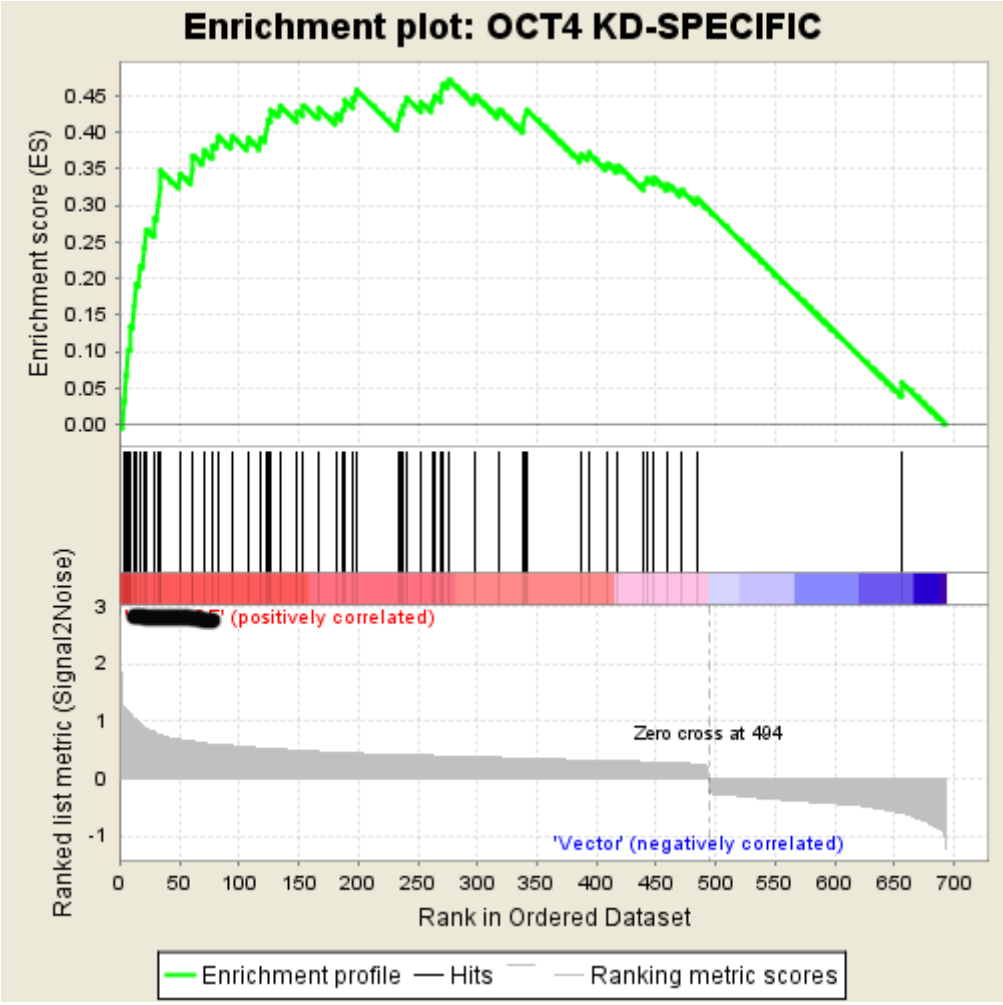
5.1.4 Installation

Install gseapy package from bioconda or pypi.

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

# or use pip
$ 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

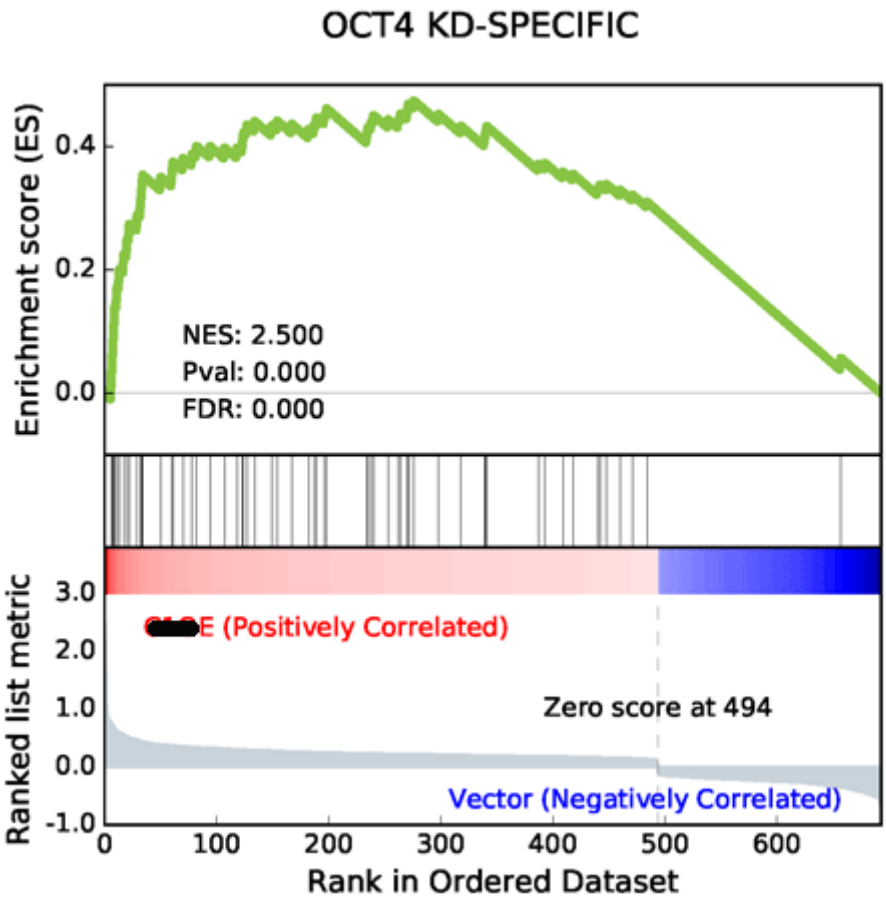


Fig. 1: Generated by GSEAPY

GSEApy figures are supported by all matplotlib figure formats.

You can modify `GSEA` plots easily in .pdf files. Please Enjoy.

5.1.7 GSEApy `enrichr` module

A graphical introduction of Enrichr

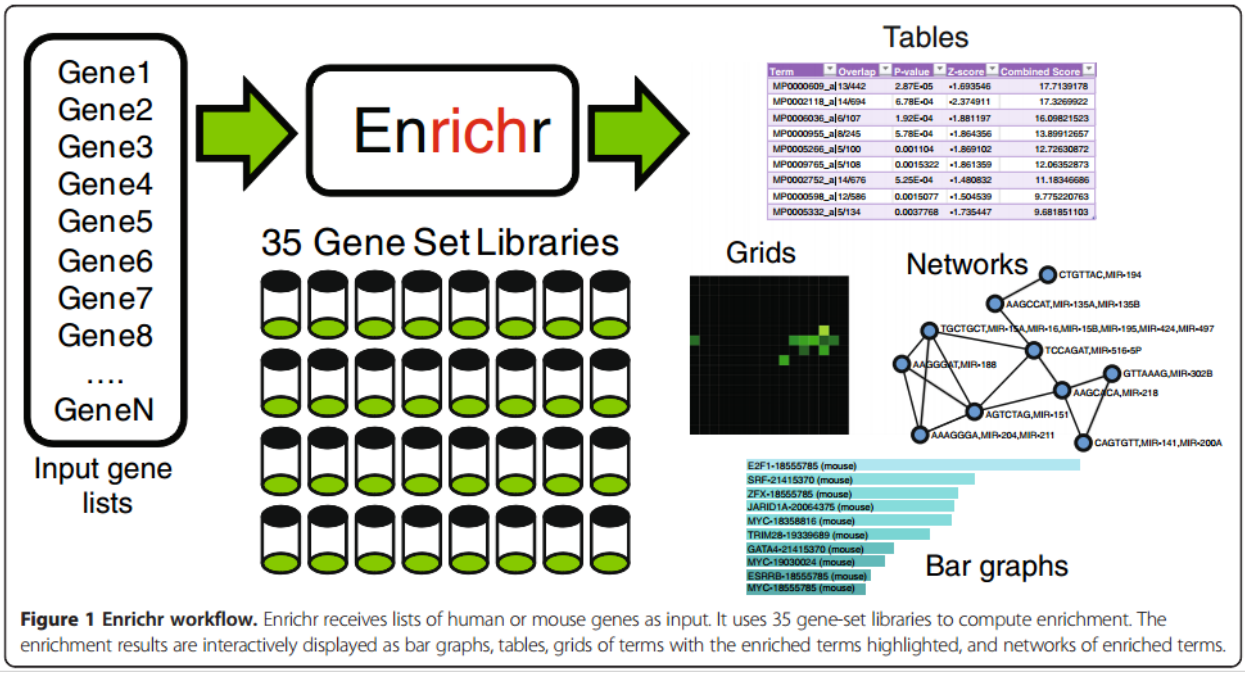


Figure 1 Enrichr workflow. Enrichr receives lists of human or mouse genes as input. It uses 35 gene-set libraries to compute enrichment. The enrichment results are interactively displayed as bar graphs, tables, grids of terms with the enriched terms highlighted, and networks of enriched terms.

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())


CTLA2B
SCARA3
LOC100044683
CMBL
CLIC6
IL13RA1
TACSTD2
DKKL1
CSF1
CITED1
SYNPO2L
```

(continues on next page)

(continued from previous page)

```
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 Tutorial

GSEApy is a Python/Rust toolkit for **Gene Set Enrichment Analysis** and related methods. This notebook walks through every public entry point with small, runnable demos using the data bundled in the repository's `tests/` folder.

5.2.1 What this notebook covers

Section	Function	What it does
1. Setup	—	imports and version check
2. Biomart	`Biomart`	convert gene IDs, map orthologs between species
3. MSigDB	`Msigdb`	download MSigDB gene-set collections (`.gmt`)
4. Enrichr libraries	`get_library_name`, `get_library`	list / fetch Enrichr gene-set libraries
5. Over-representation (online)	`enrichr`	Enrichr web service
6. Over-representation (offline)	`enrich`	local hypergeometric test
7. Preranked GSEA	`prerank`	GSEA on a single ranked list
8. Standard GSEA	`gsea`	phenotype-permutation GSEA
9. ssGSEA	`ssgsea`	single-sample GSEA
10. GSVA	`gsva`	gene set variation analysis
11. Replot	`replot`	re-plot results from the Broad GSEA Java app

Tip: run the cells in order. Sections 2–5 call online services (Biomart, MSigDB, Enrichr) and need an internet connection; everything else runs fully offline on the bundled test data.

Setup

Import GSEApy together with pandas and matplotlib. The two `%autoreload` magics are only useful if you are editing the GSEApy source while you work — they reload changed modules automatically. They are harmless otherwise.

[1]:
```
# %matplotlib inline
# %config InlineBackend.figure_format = 'retina'  # crisper figures on HiDPI/Mac screens
%load_ext autoreload
%autoreload 2

import pandas as pd
import matplotlib.pyplot as plt
import gseapy as gp
```

Check the installed GSEApy version (this tutorial targets v1.x).

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

[2]:
```
'1.3.0'
```

Biomart API

`Biomart` is a thin client over the Ensembl BioMart web service. Use it to **convert gene identifiers** (Ensembl ID gene symbol Entrez ID) and to **map orthologs between species**.

> **Note:** BioMart support is limited and the public server can be slow or occasionally unavailable. Skip this section if you do not need ID conversion.

5.2.2 Convert gene identifiers

[3]:
```
from gseapy import Biomart

bm = Biomart()
```

[4]:
```
# The commented helpers below let you discover valid marts/datasets/attributes/filters.
# Run them once to explore, then build your real query.
# marts    = bm.get_marts()                                    # available marts
# datasets = bm.get_datasets(mart='ENSEMBL_MART_ENSEMBL')     # datasets in a mart
# attrs    = bm.get_attributes(dataset='hsapiens_gene_ensembl')  # columns you can request
# filters  = bm.get_filters(dataset='hsapiens_gene_ensembl')     # columns you can filter␣
↪on

# A query needs: a dataset, the attributes (columns) to return, and optional filters.
# `filters` must be a dict mapping a filter name to the values to keep.
queries = {'ensembl_gene_id': ['ENSG00000125285', 'ENSG00000182968']}

results = bm.query(
    dataset='hsapiens_gene_ensembl',
    attributes=['ensembl_gene_id', 'external_gene_name', 'entrezgene_id', 'go_id'],
    filters=queries,
)
results.tail()
```

[4]:

	ensembl_gene_id	external_gene_name	entrezgene_id	go_id
27	ENSG00000182968	SOX1	6656	GO:0007420

(continues on next page)

(continued from previous page)

```
28  ENSG00000182968              SOX1          6656  GO:0030182
29  ENSG00000182968              SOX1          6656  GO:0043565
30  ENSG00000182968              SOX1          6656  GO:0048713
31  ENSG00000182968              SOX1          6656  GO:1904936
```

Inspect the column types of the returned table.

[5]:
```
results.dtypes
```

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

5.2.3 Map gene symbols between mouse and human

This is handy when you need to translate gene symbols across species — for example, to run a human gene-set library against mouse data. We pull the full ortholog mapping in both directions (no filter all genes), so each query returns the whole genome and may take a little while.

[6]:
```
from gseapy import Biomart

bm = Biomart()

# Mouse -> Human orthologs. Note the dataset and attribute names differ per direction.
m2h = bm.query(
    dataset='mmusculus_gene_ensembl',
    attributes=['ensembl_gene_id', 'external_gene_name',
                'hsapiens_homolog_ensembl_gene',
                'hsapiens_homolog_associated_gene_name'],
)

# Human -> Mouse orthologs.
h2m = bm.query(
    dataset='hsapiens_gene_ensembl',
    attributes=['ensembl_gene_id', 'external_gene_name',
                'mmusculus_homolog_ensembl_gene',
                'mmusculus_homolog_associated_gene_name'],
)
```

[7]:
```
# Peek at a random sample of the human -> mouse mapping.
h2m.sample(5)
```

[7]:
```
       ensembl_gene_id external_gene_name mmusculus_homolog_ensembl_gene  \
79602  ENSG00000105750              ZNF85             ENSMUSG00000048280
85423  ENSG00000137135           ARHGEF39             ENSMUSG00000051517
71287  ENSG00000137077              CCL21             ENSMUSG00000094065
20341  ENSG00000306096                NaN                            NaN
5195   ENSG00000273092            TAS2R20                            NaN

      mmusculus_homolog_associated_gene_name
79602                                 Zfp738
```

(continues on next page)

(continued from previous page)

```
85423                                  Arhgef39
71287                                    Ccl21d
20341                                       NaN
5195                                        NaN
```

5.2.4 Translate a `.gmt` gene-set file to another species

GSEApy gene symbols are case-sensitive when used offline, and most public libraries (Enrichr, MSigDB) ship **human** symbols. To run them against mouse data, convert the gene members with the ortholog map built above.

Example: convert a human KEGG library to mouse symbols.

[8]:
```
# Build a {human_symbol: mouse_symbol} lookup, dropping rows with missing values.
h2m_dict = {}
for _, row in h2m[['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 a human KEGG .gmt into a {term: [genes]} dict.
kegg = gp.read_gmt(path="tests/extdata/enrichr.KEGG_2016.gmt")
print("Human symbols:", kegg['MAPK signaling pathway Homo sapiens hsa04010'][:10])
```

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

[9]:
```
# Translate every gene in every term to its mouse ortholog (skip genes with no mapping).
kegg_mouse = {}
for term, genes in kegg.items():
    kegg_mouse[term] = [h2m_dict[g] for g in genes if g in h2m_dict]

print("Mouse symbols:", kegg_mouse['MAPK signaling pathway Homo sapiens hsa04010'][:10])
```

```
Mouse symbols: ['Egf', 'Il1r1', 'Il1r2', 'Hspa1l', 'Cacna2d2', 'Cacna2d1', 'Cacna2d4',
↪'Cacna2d3', 'Mapk8ip3', 'Mapk8ip1']
```

MSigDB API

`Msigdb` downloads gene-set collections from the MSigDB (Broad Institute). You pick a `category` (e.g. hallmark, curated, GO) and a database version (`dbver`), and get back a `{term: [genes]}` dict ready to pass to any GSEApy function.

[10]:
```
from gseapy import Msigdb

msig = Msigdb()
# Mouse hallmark gene sets (category 'mh.all') from the 2023.1.Mm release.
gmt = msig.get_gmt(category='mh.all', dbver="2023.1.Mm")
```

Two helpers let you discover valid values before downloading:

```
msig.list_dbver()                    # list available database versions
msig.list_category(dbver="2023.1.Hs")  # list categories for a given version
```

[11]:
```
# Inspect one hallmark set.
print(gmt['HALLMARK_WNT_BETA_CATENIN_SIGNALING'])
```

```
['Ctnnb1', 'Jag1', 'Myc', 'Notch1', 'Ptch1', 'Trp53', 'Axin1', 'Ncstn', 'Rbpj', 'Psen2',
↪'Wnt1', 'Axin2', 'Hey2', 'Fzd1', 'Frat1', 'Csnk1e', 'Dvl2', 'Hey1', 'Gnai1', 'Lef1',
↪'Notch4', 'Ppard', 'Adam17', 'Tcf7', 'Numb', 'Ccnd2', 'Ncor2', 'Kat2a', 'Nkd1', 'Hdac2
↪', 'Dkk1', 'Wnt5b', 'Wnt6', 'Dll1', 'Skp2', 'Hdac5', 'Fzd8', 'Dkk4', 'Cul1', 'Jag2',
↪'Hdac11', 'Maml1']
```

Enrichr gene-set libraries

Enrichr hosts hundreds of curated gene-set libraries. GSEApy can list them and download any of them as a `{term: [genes]}` dict for use with `enrichr`, `enrich`, `prerank`, `gsea`, `ssgsea`, or `gsva`.

List the available library names.

Choose the organism from: `'Human'`, `'Mouse'`, `'Yeast'`, `'Fly'`, `'Fish'`, `'Worm'`.

[12]:
```
# Default organism is Human.
names = gp.get_library_name()
names[:10]
```

[12]:
```
['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']
```

[13]:
```
# Library names differ per organism.
yeast = gp.get_library_name(organism='Yeast')
yeast[:10]
```

[13]:
```
['Cellular_Component_AutoRIF',
 'Cellular_Component_AutoRIF_Predicted_zscore',
 'GO_Biological_Process_2018',
 'GO_Biological_Process_AutoRIF',
 'GO_Biological_Process_AutoRIF_Predicted_zscore',
 'GO_Cellular_Component_2018',
 'GO_Cellular_Component_AutoRIF',
 'GO_Cellular_Component_AutoRIF_Predicted_zscore',
 'GO_Molecular_Function_2018',
 'GO_Molecular_Function_AutoRIF']
```

Download a library into a ``{term: [genes]}`` dict.

[14]:
```
# Fetch one library by name (or pass a local .gmt path to read_gmt instead).
go_mf = gp.get_library(name='GO_Molecular_Function_2018', organism='Yeast')
print(go_mf['ATP binding (GO:0005524)'])
```

```
['MLH1', 'ECM10', 'RLI1', 'SSB1', 'SSB2', 'MSH2', 'YTA12', 'CDC6', 'HMI1', 'YNL247W',
↪'MSH6', 'SSQ1', 'MCM7', 'SRS2', 'HSP104', 'SSA1', 'MCX1', 'SSC1', 'ARP2', 'ARP3', 'SSE1
```

(continues on next page)

(continued from previous page)

```
↪', 'SMC2', 'SSZ1', 'TDA10', 'ORC5', 'VPS4', 'RBK1', 'NEW1', 'SSA4', 'ORC1', 'KAR2',
↪'SSA2', 'DYN1', 'SSA3', 'PGK1', 'VPS33', 'LHS1', 'CDC123', 'PMS1']
```

Over-representation analysis (Enrichr web service)

`gp.enrichr()` sends a gene list to the Enrichr web service and returns enrichment results. The only required input is a list of gene **symbols**.

Accepted inputs

- `gene_list`: a `list`, `pandas.Series`, single-column `DataFrame`, or a `.txt` file with one gene symbol per line.
- `gene_sets`: one or more Enrichr library names (comma-separated string or list), a local `.gmt` path, or a `{term: [genes]}` dict.

```
gene_list = "./data/gene_list.txt"      # or a Python list / Series
gene_sets = "KEGG_2016"                  # one library
gene_sets = "KEGG_2016,KEGG_2013"        # several, comma-separated
gene_sets = ["KEGG_2016", "KEGG_2013"]   # several, as a list
```

Note: for the online service, gene symbols are **not** case-sensitive — they are upper-cased automatically.

[15]:
```
# Load an example gene list (one symbol per line).
gene_list = pd.read_csv("./tests/data/gene_list.txt", header=None, sep="\t")
gene_list.head()
```

[15]:
```
                0
0          IGKV4-1
1             CD55
2             IGKC
3          PPFIBP1
4            ABHD4
```

[16]:
```
# A DataFrame/Series can be flattened to a plain Python list if you prefer.
glist = gene_list.squeeze().str.strip().to_list()
print(glist[:10])
```

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

5.2.5 Enrichr without a custom background

Set `outdir=None` to keep results in memory only (nothing written to disk). The returned object stores the table on both `.res2d` (last query) and `.results` (all queries).

[17]:
```
enr = gp.enrichr(
    gene_list=gene_list,                                # or "./tests/data/gene_list.txt"
    gene_sets=['MSigDB_Hallmark_2020', 'KEGG_2021_Human'],
    organism='human',                                   # set to your organism, e.g. 'Yeast'
    outdir=None,                                         # keep results in memory only
)
```

[18]:
```
# All enrichment results across the requested libraries.
enr.results.head(5)
```

[18]:
```
              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...
```

5.2.6 Enrichr with a custom background

Provide your own background gene list (e.g. all genes expressed in your assay).

Note: when a background is supplied, the output omits the `Overlap` column.

[19]:
```
enr_bg = gp.enrichr(
    gene_list=gene_list,
    gene_sets=['MSigDB_Hallmark_2020', 'KEGG_2021_Human'],
    background="tests/data/background.txt",            # the 'organism' arg is ignored␣
↪when set
    outdir=None,
)
```

[20]:
```
enr_bg.results.head()
```

[20]:
```
              Gene_set                          Term       P-value  \
0  MSigDB_Hallmark_2020       IL-6/JAK/STAT3 Signaling  3.559435e-11
1  MSigDB_Hallmark_2020  TNF-alpha Signaling via NF-kB  3.401526e-10
2  MSigDB_Hallmark_2020                    Complement  3.813953e-10
3  MSigDB_Hallmark_2020          Inflammatory Response  3.380686e-08
4  MSigDB_Hallmark_2020                heme Metabolism  8.943634e-08

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

   Combined Score                                              Genes
0      205.300064  IL4R;TGFB1;IL1R1;IFNGR1;IL10RB;ITGB3;IFNGR2;IL...
```

(continues on next page)

(continued from previous page)

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

Over-representation analysis (offline hypergeometric test)

`gp.enrich()` runs the same over-representation test **locally** — it does **not** call the Enrichr web service. Use it for custom or private gene sets, or when you have no internet access.

Important

1. Input gene symbols are **case-sensitive** here.
2. The identifier type in your `gene_list` must match the one used in the gene sets (`.gmt`/dict).
3. `gene_sets` accepts a `.gmt` path or a `{term: [genes]}` dict:

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

[21]:
```
# Note: the offline function is `enrich` (no trailing "r"), unlike the online `enrichr`.
# gene_sets here mixes a .gmt file, a name that won't resolve ("unknown"), and a dict␣
↪(kegg).
enr2 = gp.enrich(
    gene_list="./tests/data/gene_list.txt",           # or a Python list
    gene_sets=["./tests/data/genes.gmt", "unknown", kegg],
    background=None,    # None | int | gene list | text file | a Biomart dataset name
    outdir=None,
    verbose=True,
)
```

```
2026-06-24 11:23:41,352 [INFO] User defined gene sets is given: ./tests/data/genes.gmt
2026-06-24 11:23:41,356 [INFO] Input dict object named with gs_ind_2
2026-06-24 11:23:41,735 [WARNING] Input library not found: unknown. Skip
2026-06-24 11:23:41,736 [INFO] Off-line enrichment analysis with library: genes.gmt
2026-06-24 11:23:41,738 [INFO]   Background is not set! Use all 682 genes in genes.gmt.
2026-06-24 11:23:41,743 [INFO] Off-line enrichment analysis with library: gs_ind_2
2026-06-24 11:23:41,768 [INFO]   Background is not set! Use all 7010 genes in gs_ind_2.
2026-06-24 11:23:41,791 [INFO] Done.
```

[22]:
```
enr2.results.head()
```

[22]:
```
    Gene_set        Term Overlap   P-value  Adjusted P-value  Odds Ratio  \
0  genes.gmt  BvA_UpIN_A    8/139  0.457390          0.568432    1.169168
1  genes.gmt  BvA_UpIN_B   12/130  0.026744          0.187208    2.275467
2  genes.gmt  CvA_UpIN_A     1/12  0.481190          0.568432    2.334966
3  genes.gmt  DvA_UpIN_A   16/284  0.426669          0.568432    1.134623
4  genes.gmt  DvA_UpIN_D   13/236  0.487227          0.568432    1.088533

   Combined Score                                              Genes
0        0.914547   PADI2;MAP3K5;IL1R1;MBOAT2;MSRB2;HAL;PCSK6;IQGAP2
1        8.240477  SUOX;MBNL3;LPAR1;ST3GAL6;DYSF;FAM65B;HEBP1;ARH...
2        1.708011                                             MBOAT2
```

(continues on next page)

(continued from previous page)

```
3        0.966412  VNN1;NMNAT1;PADI2;MAP3K5;ATP6V1B2;IL1R1;KIF1B;...
4        0.782682  FAM198B;MBNL3;LPAR1;ST3GAL6;DYSF;GNB4;FAM65B;T...
```

5.2.7 Choosing a background for the offline test

By default, **all genes in ``gene_sets``** are used as the background. A more accurate background is usually one of the following:

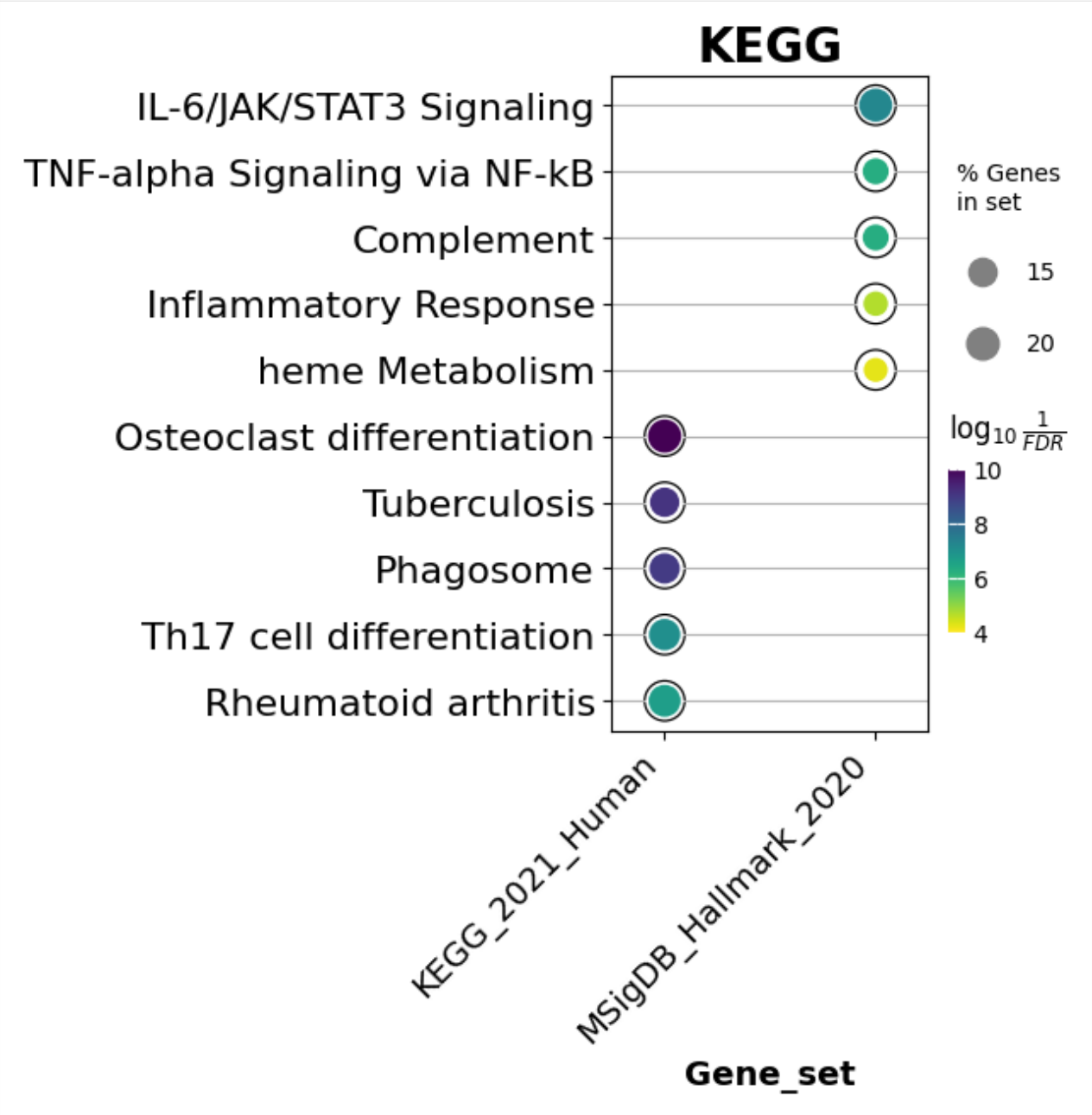
1. **(Recommended) A list of background genes**, `['gene1', 'gene2', ...]` — typically the genes detectable in your experiment (e.g. expressed genes from your RNA-seq). The identifier type must match the gene sets.
2. **A total gene count**, e.g. `20000`. Simple, but it assumes every gene set gene is present in the background; genes in the sets but missing from the background still distort the test.
3. **A BioMart dataset name**, e.g. `"hsapiens_gene_ensembl"`. GSEApy fetches all annotated genes for that dataset and uses them as the background.

5.2.8 Plotting enrichment results

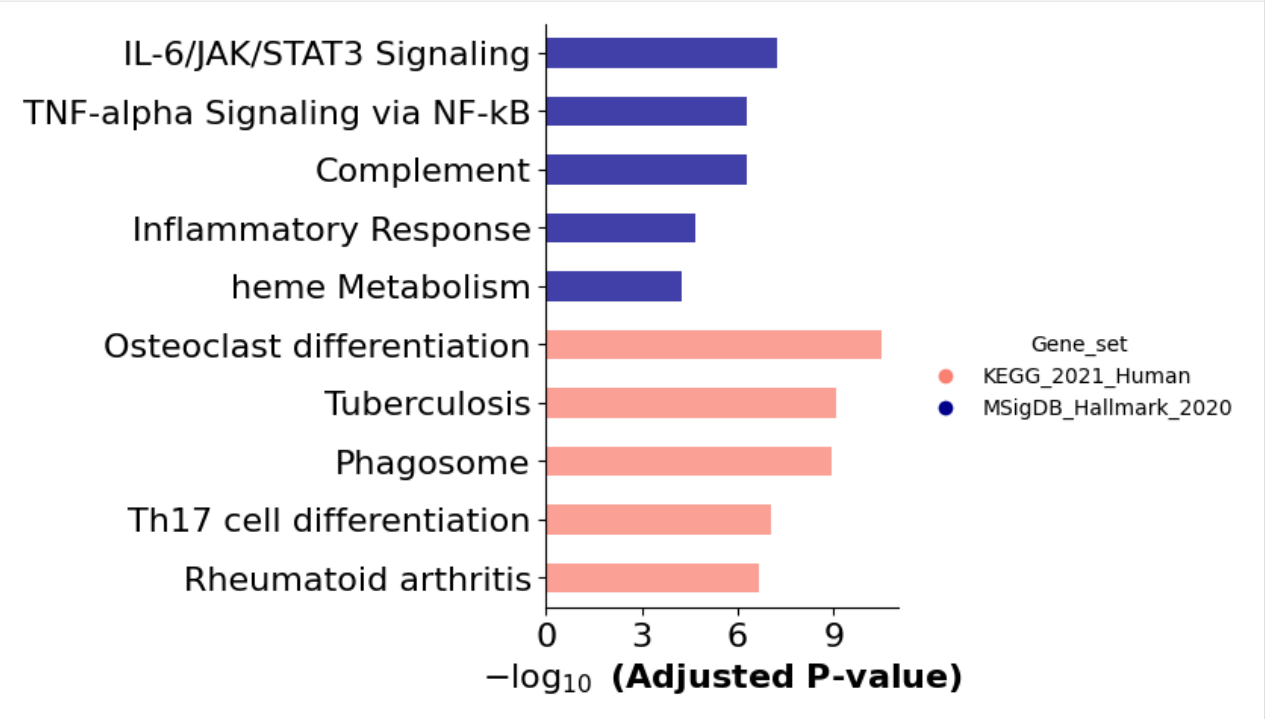
`dotplot` and `barplot` summarize an enrichment table. Below we show the top 5 terms per library, ranked by adjusted p-value.

[23]:
```
from gseapy import barplot, dotplot
```

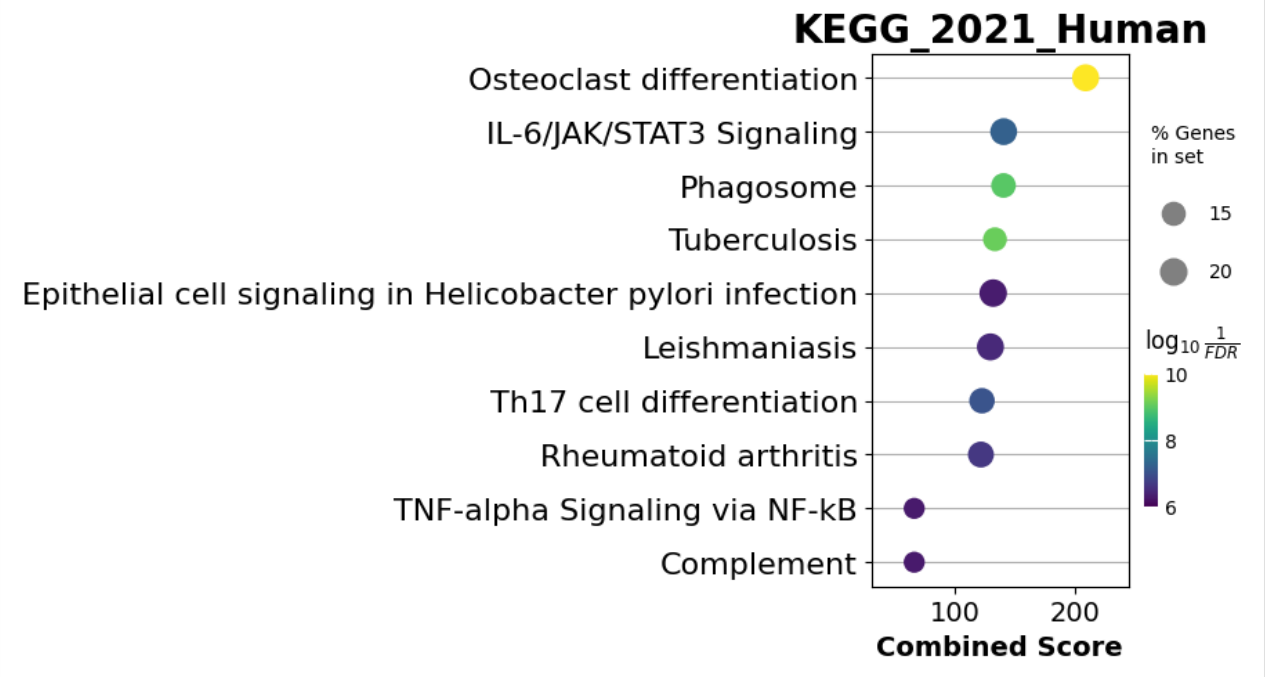
[24]:
```
# Dot plot: dot size encodes gene overlap, color encodes significance.
# Setting x='Gene_set' faces the plot by library so you can compare them side by side.
ax = dotplot(
    enr.results,
    column="Adjusted P-value",
    x='Gene_set',          # split the x-axis by library (multi-library comparison)
    size=5,
    top_term=5,
    figsize=(3, 5),
    title="KEGG",
    xticklabels_rot=45,    # rotate x tick labels for readability
    show_ring=True,        # set False to remove the outer ring around dots
    marker='o',
)
```



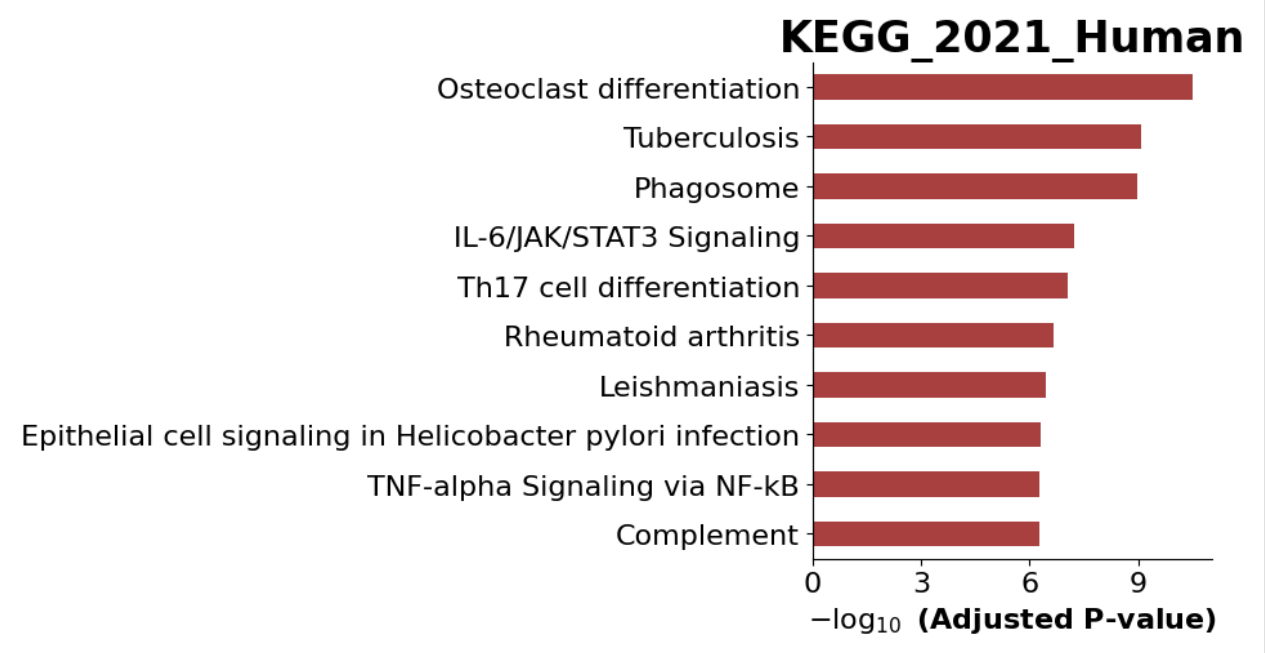
[25]:
```
# Bar plot grouped by library, with an explicit color per group.
ax = barplot(
    enr.results,
    column="Adjusted P-value",
    group='Gene_set',      # group bars by library (multi-library comparison)
    size=5,
    top_term=5,
    figsize=(3, 5),
    color={'KEGG_2021_Human': 'salmon', 'MSigDB_Hallmark_2020': 'darkblue'},
)
```



[26]:
```
# Single-library views use .res2d (the most recent query).
# Pass ofname='figure.pdf' to save the figure to disk.
ax = dotplot(enr.res2d, title='KEGG_2021_Human', cmap='viridis', size=5, figsize=(3, 5))
```



[27]:
```
ax = barplot(enr.res2d, title='KEGG_2021_Human', figsize=(4, 5), color='darkred')
```



5.2.9 Command-line equivalent

The same analysis is available from the `gseapy enrichr` CLI. The `-v` flag prints progress.

[28]:
```
# !gseapy enrichr -i ./tests/data/gene_list.txt \
#                 -g GO_Biological_Process_2017 \
#                 -v -o test/enrichr_BP
```

Preranked GSEA (`prerank`)

`gp.prerank()` runs GSEA on a **single, pre-ranked gene list** — for example, genes ranked by a differential-expression statistic.

Accepted ``rnk`` inputs

- a 2-column `DataFrame` (gene, score), or a 1-column `DataFrame` indexed by gene
- a `pandas.Series` indexed by gene
- a `.rnk` text file (no header; anything after a `#` is ignored)

Accepted ``gene_sets`` inputs — same as the other functions: Enrichr library name(s), a `.gmt` path, or a `{term: [genes]}` dict.

> **Note on symbols:** Enrichr libraries use **UPPERCASE** human symbols. Convert your gene identifiers to match (see Section 2.3 for cross-species conversion).

[29]:
```
# A .rnk file: column 0 = gene, column 1 = ranking metric.
rnk = pd.read_csv("./tests/data/temp.rnk", header=None, index_col=0, sep="\t")
rnk.head()
```

[29]:
```
                1
0
ATXN1   16.456753
UBQLN4  13.989493
```

(continues on next page)

(continued from previous page)

```
CALM1   13.745533
DLG4    12.796588
MRE11A  12.787631
```

[30]:
```
rnk.shape
```

[30]:
```
(22922, 1)
```

5.2.10 Run prerank

`permutation_num` is reduced to 1000 here to keep the demo fast. `outdir=None` keeps everything in memory. We use `method='multilevel'` (the fgsea multilevel p-value, explained just below); the default is `method='permutation'`.

[31]:
```
pre_res2 = gp.prerank(
    rnk="./tests/data/temp.rnk",   # or rnk=rnk
    gene_sets='KEGG_2016',          # Enrichr library name (downloaded automatically)
    threads=4,
    min_size=5,
    max_size=1000,
    permutation_num=1000,           # raise for publication-quality p-values
    outdir=None,
    seed=6,                         # set a seed for reproducibility
    method='multilevel',            # fgsea multilevel p-values (see below)
    verbose=True,
)
pre_res2.res2d.head(5)
```

```
2026-06-24 11:23:42,569 [WARNING] Duplicated values found in preranked stats: 4.97% of␣
↪genes
The order of those genes will be arbitrary, which may produce unexpected results.
2026-06-24 11:23:42,569 [INFO] Parsing data files for GSEA...
2026-06-24 11:23:42,583 [INFO] Enrichr library gene sets already downloaded in: /Users/
↪fangzq/.cache/gseapy, use local file
2026-06-24 11:23:42,590 [INFO] 0001 gene_sets have been filtered out when max_size=1000␣
↪and min_size=5
2026-06-24 11:23:42,591 [INFO] 0292 gene_sets used for further statistical testing...
2026-06-24 11:23:42,591 [INFO] Start to run GSEA...Might take a while...
2026-06-24 11:23:57,327 [INFO] Congratulations. GSEApy runs successfully...
```

[31]:
```
      Name                                             Term        ES  \
0  prerank            Adherens junction Homo sapiens hsa04520  0.784625
1  prerank   Estrogen signaling pathway Homo sapiens hsa04915  0.766347
2  prerank                        Glioma Homo sapiens hsa05214  0.784678
3  prerank  Thyroid hormone signaling pathway Homo sapiens...  0.757700
4  prerank       ErbB signaling pathway Homo sapiens hsa04012  0.765574

        NES     NOM p-val     FDR q-val   Tag %  Gene %  \
0  1.906344  9.365577e-17  1.012870e-15   47/74  10.37%
1  1.898225  2.680224e-19  5.217502e-18   74/99  16.57%
2  1.894231  2.712993e-15  2.263411e-14   52/65  16.29%
3  1.893138  1.934581e-22  8.069965e-21  84/118  16.29%
4  1.880505  1.112913e-17  1.477140e-16   65/87  16.29%
```

(continues on next page)

(continued from previous page)

```
                                          Lead_genes   log2err
0  CTNNB1;EGFR;RAC1;TGFBR1;SMAD4;MET;EP300;CDC42;...  1.039888
1  CALM1;PRKACA;GRB2;SP1;EGFR;KRAS;HRAS;HSP90AB1;...  1.120089
2  CALM1;GRB2;EGFR;PRKCA;KRAS;HRAS;TP53;MAPK1;PRK...  0.991232
3  CTNNB1;PRKACA;PRKCA;KRAS;NOTCH1;EP300;CREBBP;H...  1.210593
4  GRB2;EGFR;PRKCA;KRAS;HRAS;MAPK1;PRKCB;SRC;NRAS...  1.069466
```

For the comparison later in this section, also run the **default permutation** method on the same ranking.

[32]:
```
pre_res = gp.prerank(
    rnk="./tests/data/temp.rnk",
    gene_sets='KEGG_2016',
    threads=4,
    min_size=5,
    max_size=1000,
    permutation_num=1000,
    outdir=None,
    seed=6,
    # method='permutation' is the default
    verbose=True,
)
pre_res.res2d.head(5)
```

```
2026-06-24 11:23:57,360 [WARNING] Duplicated values found in preranked stats: 4.97% of␣
↪genes
The order of those genes will be arbitrary, which may produce unexpected results.
2026-06-24 11:23:57,361 [INFO] Parsing data files for GSEA...
2026-06-24 11:23:57,374 [INFO] Enrichr library gene sets already downloaded in: /Users/
↪fangzq/.cache/gseapy, use local file
2026-06-24 11:23:57,382 [INFO] 0001 gene_sets have been filtered out when max_size=1000␣
↪and min_size=5
2026-06-24 11:23:57,382 [INFO] 0292 gene_sets used for further statistical testing...
2026-06-24 11:23:57,382 [INFO] Start to run GSEA...Might take a while...
2026-06-24 11:24:02,252 [INFO] Congratulations. GSEApy runs successfully...
```

[32]:
```
      Name                                              Term        ES  \
0  prerank            Adherens junction Homo sapiens hsa04520  0.784625
1  prerank   Estrogen signaling pathway Homo sapiens hsa04915  0.766347
2  prerank                        Glioma Homo sapiens hsa05214  0.784678
3  prerank  Thyroid hormone signaling pathway Homo sapiens...  0.757700
4  prerank       Long-term potentiation Homo sapiens hsa04720  0.778249

        NES  NOM p-val  FDR q-val  FWER p-val   Tag %  Gene %  \
0  1.917092        0.0        0.0         0.0   47/74  10.37%
1  1.896928        0.0        0.0         0.0   74/99  16.57%
2  1.895572        0.0        0.0         0.0   52/65  16.29%
3  1.890973        0.0        0.0         0.0  84/118  16.29%
4  1.890416        0.0        0.0         0.0   42/66   9.01%

                                          Lead_genes
0  CTNNB1;EGFR;RAC1;TGFBR1;SMAD4;MET;EP300;CDC42;...
```

(continues on next page)

(continued from previous page)

```
1  CALM1;PRKACA;GRB2;SP1;EGFR;KRAS;HRAS;HSP90AB1;...
2  CALM1;GRB2;EGFR;PRKCA;KRAS;HRAS;TP53;MAPK1;PRK...
3  CTNNB1;PRKACA;PRKCA;KRAS;NOTCH1;EP300;CREBBP;H...
4  CALM1;PRKACA;PRKCA;KRAS;EP300;CREBBP;HRAS;PRKA...
```

5.2.11 The fgsea multilevel method

With the default gene-permutation method, the smallest reportable nominal *p*-value is bounded by `1 / permutation_num`. Passing ``**method='multilevel'**`` switches to GSEApy's faithful Rust port of the fgsea multilevel algorithm, which resolves *p*-values far below `1/nperm` (down to `eps`) via adaptive split-sampling.

Things to keep in mind:

- It runs on a **single ranked list**, like classic preranked GSEA.
- The **NES is fgsea-style**: `NES = ES / mean(|null ES| of the same sign)`, where the null comes from random gene sets. This differs from the gene-permutation NES, so NES values are **not directly comparable** between the two methods.
- An extra ``**log2err**`` column reports the estimated log2 standard error of the multilevel *p*-value.

[33]:
```
pre_mult = gp.prerank(
    rnk="./tests/data/temp.rnk",     # single ranked list
    gene_sets="KEGG_2016",
    method="multilevel",             # use fgsea multilevel p-values
    sample_size=101,                 # multilevel sampling depth (default 101)
    eps=1e-50,                       # smallest p-value resolved; 0 => machine precision
    threads=4,
    min_size=5,
    max_size=1000,
    seed=6,
    outdir=None,
)
pre_mult.res2d.head()
```

```
2026-06-24 11:24:02,297 [WARNING] Duplicated values found in preranked stats: 4.97% of␣
↪genes
The order of those genes will be arbitrary, which may produce unexpected results.
```

[33]:
```
      Name                                             Term        ES  \
0  prerank            Adherens junction Homo sapiens hsa04520  0.784625
1  prerank   Estrogen signaling pathway Homo sapiens hsa04915  0.766347
2  prerank                       Glioma Homo sapiens hsa05214  0.784678
3  prerank  Thyroid hormone signaling pathway Homo sapiens...  0.757700
4  prerank       ErbB signaling pathway Homo sapiens hsa04012  0.765574

        NES     NOM p-val     FDR q-val   Tag %  Gene %  \
0  1.906344  9.365577e-17  1.012870e-15   47/74  10.37%
1  1.898225  2.680224e-19  5.217502e-18   74/99  16.57%
2  1.894231  2.712993e-15  2.263411e-14   52/65  16.29%
3  1.893138  1.934581e-22  8.069965e-21  84/118  16.29%
4  1.880505  1.112913e-17  1.477140e-16   65/87  16.29%

                                        Lead_genes   log2err
0  CTNNB1;EGFR;RAC1;TGFBR1;SMAD4;MET;EP300;CDC42;...  1.039888
```

(continues on next page)

(continued from previous page)

```
1  CALM1;PRKACA;GRB2;SP1;EGFR;KRAS;HRAS;HSP90AB1;...  1.120089
2  CALM1;GRB2;EGFR;PRKCA;KRAS;HRAS;TP53;MAPK1;PRK...  0.991232
3  CTNNB1;PRKACA;PRKCA;KRAS;NOTCH1;EP300;CREBBP;H...  1.210593
4  GRB2;EGFR;PRKCA;KRAS;HRAS;MAPK1;PRKCB;SRC;NRAS...  1.069466
```

`pre_mult.res2d` carries the usual preranked columns plus `log2err`, and drops the `FWER p-val` column (that statistic is specific to the gene-permutation null):

Column	Meaning
`ES` / `NES`	enrichment score and fgsea-style normalized ES
`NOM p-val`	multilevel nominal *p*-value (can be far below `1/nperm`)
`FDR q-val`	Benjamini–Hochberg adjusted *p*-value
`log2err`	estimated log2 standard error of the multilevel *p*-value
`Tag %` / `Gene %`	leading-edge tag and gene fractions
`Lead_genes`	leading-edge gene members

[34]:
```
# Compare the permutation run (pre_res) against the multilevel run (pre_mult) on
# the same ranking. The NES columns are on different scales between methods, but
# the multilevel NOM p-val resolves far below 1/permutation_num (here 1e-3),
# exposing tails the permutation null cannot reach.
cols = ["Term", "NES", "NOM p-val", "FDR q-val"]
cmp = pre_res.res2d[cols].merge(
    pre_mult.res2d[cols], on="Term", suffixes=("_perm", "_multilevel")
)
cmp.sort_values("FDR q-val_multilevel").head(10)
```

[34]:
```
                                                Term  NES_perm  \
60            Pathways in cancer Homo sapiens hsa05200  1.718575
23   Epstein-Barr virus infection Homo sapiens hsa0...  1.805814
27       Proteoglycans in cancer Homo sapiens hsa05205  1.798016
28        cAMP signaling pathway Homo sapiens hsa04024  1.796236
43          Viral carcinogenesis Homo sapiens hsa05203  1.752638
49               Focal adhesion Homo sapiens hsa04510  1.734311
3    Thyroid hormone signaling pathway Homo sapiens...  1.890973
30           MicroRNAs in cancer Homo sapiens hsa05206  1.791565
116   PI3K-Akt signaling pathway Homo sapiens hsa04151  1.565976
94              HTLV-I infection Homo sapiens hsa05166  1.631478

     NOM p-val_perm  FDR q-val_perm  NES_multilevel  NOM p-val_multilevel  \
60              0.0        0.000066        1.717224          5.820484e-39
23              0.0        0.000000        1.800147          4.800341e-28
27              0.0        0.000036        1.796120          2.580557e-27
28              0.0        0.000035        1.783906          2.453085e-26
43              0.0        0.000023        1.746884          1.766868e-23
49              0.0        0.000020        1.730402          9.352231e-23
3               0.0        0.000000        1.893138          1.934581e-22
30              0.0        0.000032        1.793027          9.644428e-22
116             0.0        0.001157        1.561759          3.286497e-21
94              0.0        0.000465        1.626543          1.115876e-20

     FDR q-val_multilevel
```

(continues on next page)

(continued from previous page)

```
60          1.699581e-36
23          7.008498e-26
27          2.511742e-25
28          1.790752e-24
43          1.031851e-21
49          4.551419e-21
3           8.069965e-21
30          3.520216e-20
116         1.066286e-19
94          3.258357e-19
```

- ``**sample_size**`` (default `101`) — samples drawn per multilevel level. Larger values reduce the variance of very small *p*-values (smaller `log2err`) at the cost of runtime.
- ``**eps**`` (default `1e-50`) — lower bound on resolvable *p*-values; anything below `eps` is reported as `eps`. Set ``**eps=0**`` to push to machine precision (slowest, most accurate tails).

The multilevel method is exposed by the `gseapy prerank` CLI via `--method`, `--sample-size`, and `--eps`.

[35]:
```
# !gseapy prerank -r ./tests/data/temp.rnk -g KEGG_2016 \
#         --method multilevel --sample-size 101 --eps 1e-50 \
#         --min-size 5 --max-size 1000 --threads 4 \
#         -o prerank_multilevel_report
```

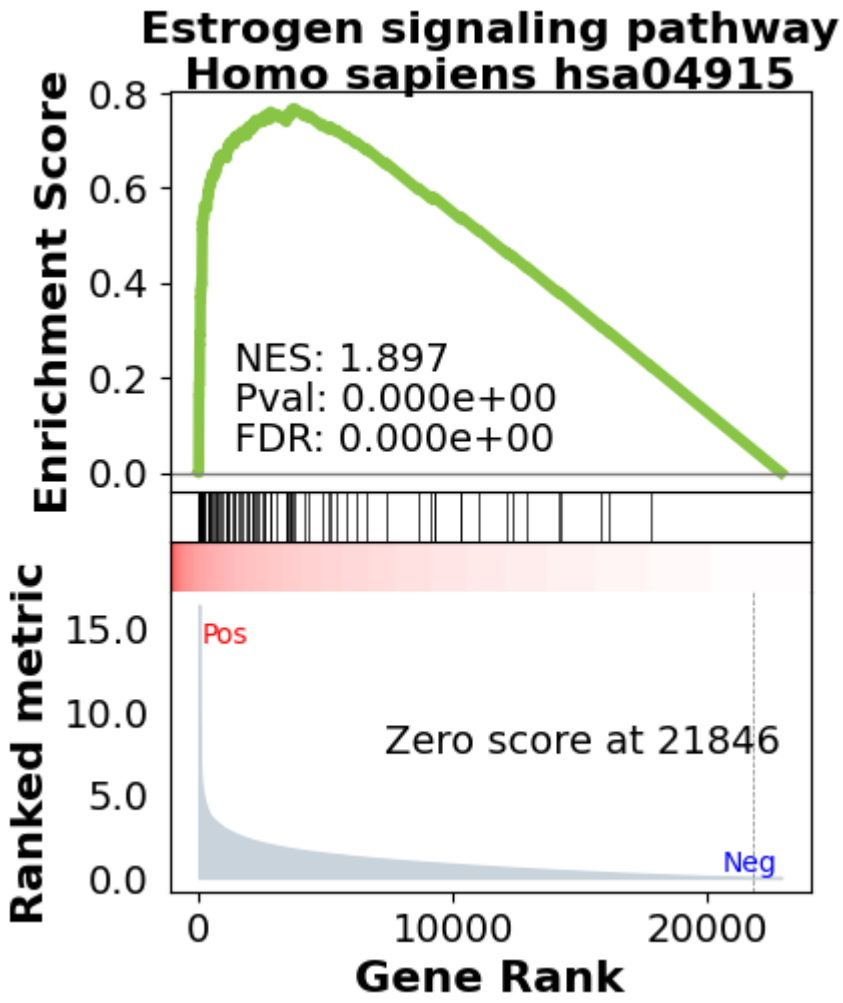
5.2.12 GSEA enrichment plots

Visualize the running enrichment score with `obj.plot()` (or the standalone `gseaplot` / `gseaplot2` helpers). Pass `ofname='figure.pdf'` to save to disk.

[36]:
```
pre_res.res2d.head(5)
```

[36]:
```
      Name                                             Term        ES  \
0  prerank            Adherens junction Homo sapiens hsa04520  0.784625
1  prerank   Estrogen signaling pathway Homo sapiens hsa04915  0.766347
2  prerank                       Glioma Homo sapiens hsa05214  0.784678
3  prerank  Thyroid hormone signaling pathway Homo sapiens...  0.757700
4  prerank       Long-term potentiation Homo sapiens hsa04720  0.778249

        NES  NOM p-val  FDR q-val  FWER p-val   Tag %  Gene %  \
0  1.917092        0.0        0.0         0.0   47/74  10.37%
1  1.896928        0.0        0.0         0.0   74/99  16.57%
2  1.895572        0.0        0.0         0.0   52/65  16.29%
3  1.890973        0.0        0.0         0.0  84/118  16.29%
4  1.890416        0.0        0.0         0.0   42/66   9.01%

                                        Lead_genes
0  CTNNB1;EGFR;RAC1;TGFBR1;SMAD4;MET;EP300;CDC42;...
1  CALM1;PRKACA;GRB2;SP1;EGFR;KRAS;HRAS;HSP90AB1;...
2  CALM1;GRB2;EGFR;PRKCA;KRAS;HRAS;TP53;MAPK1;PRK...
3  CTNNB1;PRKACA;PRKCA;KRAS;NOTCH1;EP300;CREBBP;H...
4  CALM1;PRKACA;PRKCA;KRAS;EP300;CREBBP;HRAS;PRKA...
```

[37]:
```
# Plot a single term by name. obj.plot() is the convenient wrapper (v1.0.5+).
terms = pre_res.res2d.Term
axs = pre_res.plot(terms=terms[1])

# For finer control, use the standalone helper:
# from gseapy import gseaplot
# gseaplot(rank_metric=pre_res.ranking, term=terms[0],
#          ofname='your.plot.pdf', **pre_res.results[terms[0]])
```



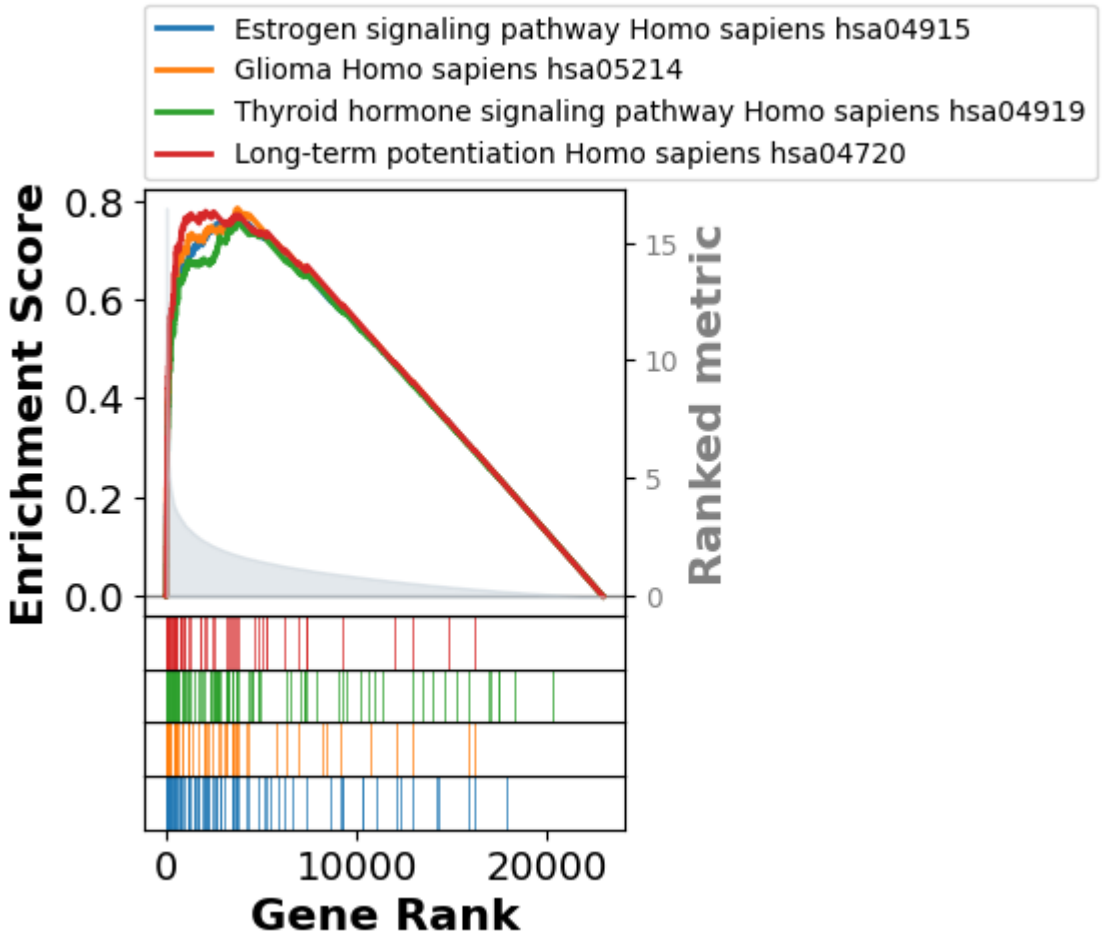
Plot several pathways together in one figure.

[38]:
```
axs = pre_res.plot(
    terms=terms[1:5],
    show_ranking=True,     # show the ranking metric on a secondary y-axis
    figsize=(3, 4),
    # legend_kws={'loc': (1.2, 0)},  # reposition the legend if needed
)

# Equivalent low-level call:
# from gseapy import gseaplot2
```

(continues on next page)

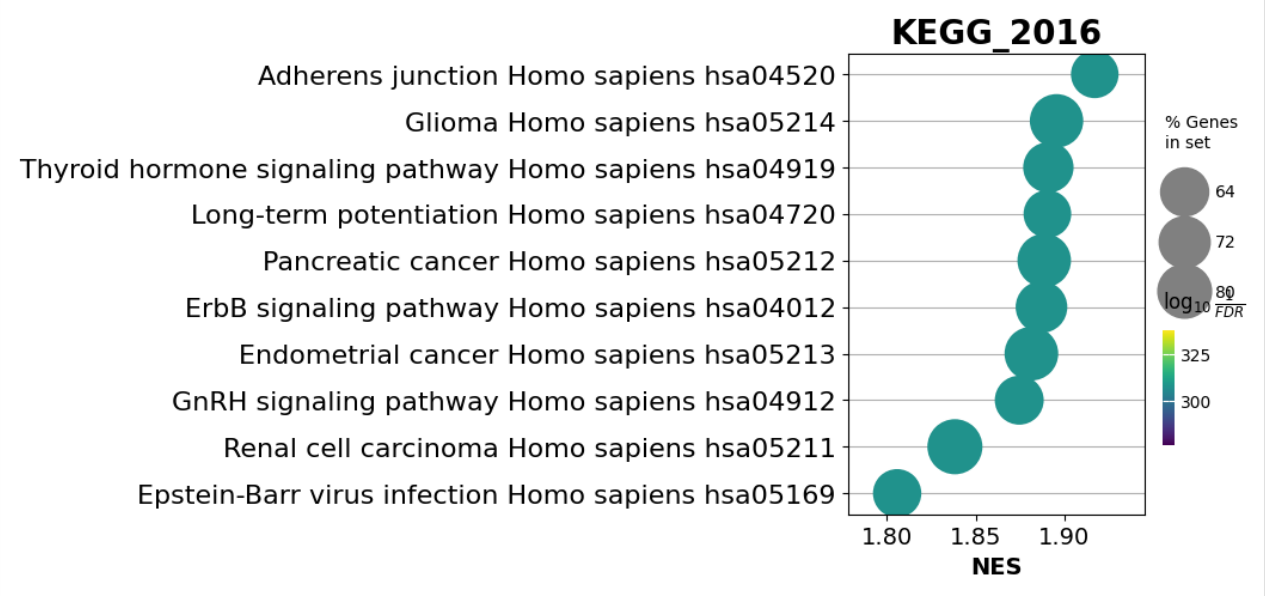
(continued from previous page)

```
# terms = pre_res.res2d.Term[1:5]
# hits  = [pre_res.results[t]['hits'] for t in terms]
# runes = [pre_res.results[t]['RES'] for t in terms]
# fig = gseaplot2(terms=terms, ress=runes, hits=hits,
#                 rank_metric=pre_res.ranking, figsize=(4, 5))
```



dotplot also works on GSEA results.

[39]:
```
from gseapy import dotplot

ax = dotplot(
    pre_res.res2d,
    column="FDR q-val",
    title='KEGG_2016',
    cmap=plt.cm.viridis,
    size=6,                # dot size scaling
    figsize=(4, 5),
    cutoff=0.25,           # only show terms with FDR q-val <= cutoff
    show_ring=False,
)
```



5.2.13 Network visualization (enrichment map)

`enrichment_map` builds a network where nodes are enriched terms and edges connect terms that share genes. It returns `nodes` and `edges` DataFrames that you can also export for Cytoscape.

[40]:
```
from gseapy import enrichment_map

# Returns two DataFrames: node table and edge table.
nodes, edges = enrichment_map(pre_res.res2d)
```

[41]:
```
import networkx as nx
```

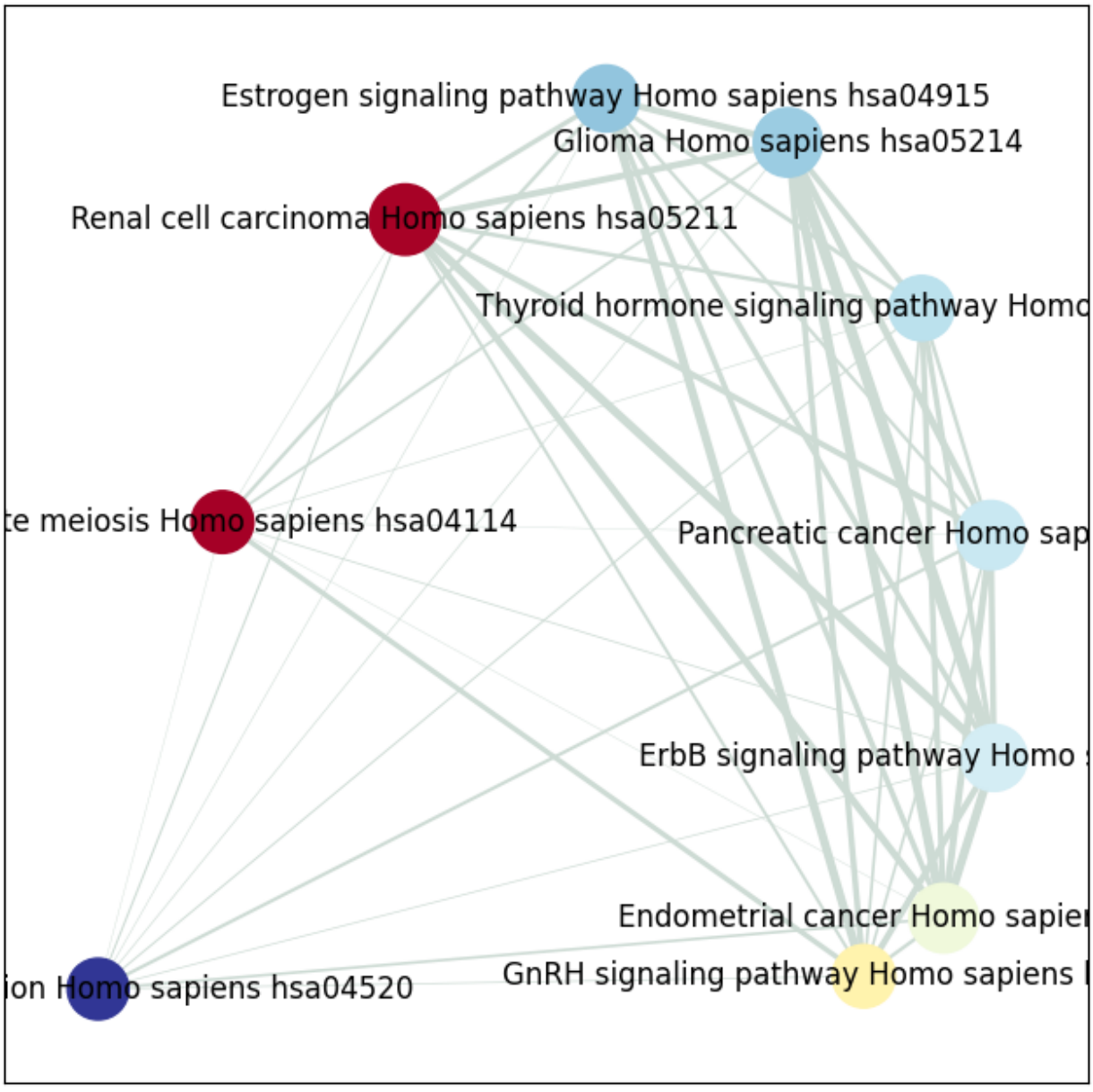
[42]:
```
# Build a graph from the edge table; carry similarity coefficients as edge attributes.
G = nx.from_pandas_edgelist(
    edges,
    source='src_idx',
    target='targ_idx',
    edge_attr=['jaccard_coef', 'overlap_coef', 'overlap_genes'],
)
```

[43]:
```
fig, ax = plt.subplots(figsize=(8, 8))

# Lay out the nodes.
pos = nx.layout.spiral_layout(G)

# Nodes: color by NES, size by the fraction of genes hit.
nx.draw_networkx_nodes(
    G, pos=pos,
    cmap=plt.cm.RdYlBu,
    node_color=list(nodes.NES),
    node_size=list(nodes.Hits_ratio * 1000),
)
```

(continues on next page)

(continued from previous page)

```
# Node labels (term names).
nx.draw_networkx_labels(G, pos=pos, labels=nodes.Term.to_dict())

# Edges: width proportional to Jaccard similarity.
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()
```



[44]:
```
# !gseapy prerank -r temp.rnk -g temp.gmt -o prerank_report_temp
```

Standard GSEA (gsea)

`gp.gsea()` runs the classic GSEA on an **expression matrix** with **phenotype labels**, computing the ranking metric and a phenotype-permutation null internally.

Accepted inputs

- `data`: a `DataFrame` (genes × samples), a `.gct` file, or a tab-delimited text file.
- `cls`: a `.cls` file, or a Python list of class labels (one per sample).
- `gene_sets`: an Enrichr library name, a `.gmt` path, or a `{term: [genes]}` dict.

  Note on symbols: as with `prerank`, match your gene identifiers to the library (Enrichr libraries use UPPERCASE human symbols).

[45]:
```
# A .cls file labels each sample with its phenotype/class.
phenoA, phenoB, class_vector = gp.parser.gsea_cls_parser("./tests/extdata/Leukemia.cls")
```

[46]:
```
# class_vector holds the per-sample class label.
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']
```

[47]:
```
# Expression matrix: genes in rows, samples in columns.
gene_exp = pd.read_csv("./tests/extdata/Leukemia_hgu95av2.trim.txt", sep="\t")
gene_exp.head()
```

[47]:

	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
4	257.0	439.0	386.0

[5 rows x 50 columns]

[48]:
```
print("Positively correlated phenotype:", phenoA)
print("Negatively correlated phenotype:", phenoB)
```

```
Positively correlated phenotype: ALL
Negatively correlated phenotype: AML
```

[49]:
```
gs_res = gp.gsea(
    data=gene_exp,                                          # or a .gct / .txt path
    gene_sets='./tests/extdata/h.all.v7.0.symbols.gmt',  # or an Enrichr library name
    cls="./tests/extdata/Leukemia.cls",                     # or cls=class_vector
    permutation_type='phenotype',  # use 'phenotype' when each group has >= 15 samples
    permutation_num=1000,          # reduced for the demo
    outdir=None,
    method='signal_to_noise',      # ranking metric
    threads=4,
    seed=7,
)
```

```
2026-06-24 11:24:17,956 [WARNING] Found duplicated gene names, values averaged by gene␣
↪names!
```

You can also use the GSEA class directly and set the phenotype labels manually via pheno_pos / pheno_neg.

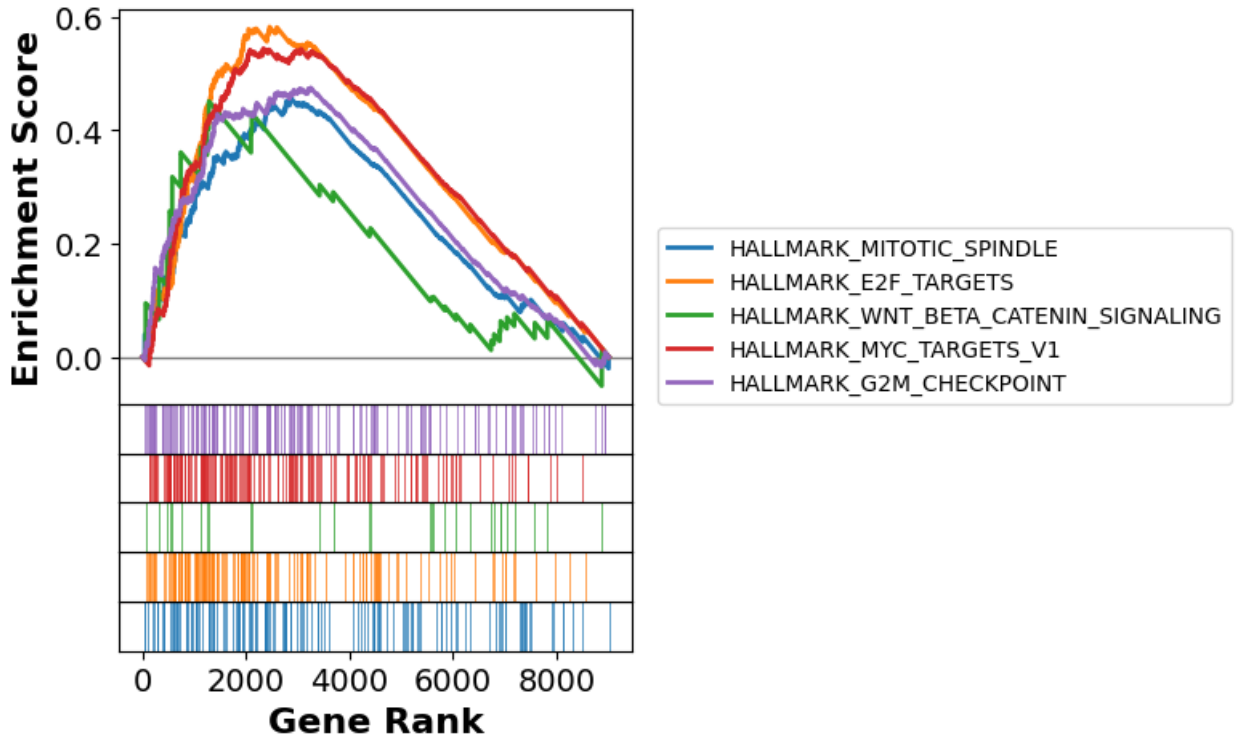
[50]:
```
from gseapy import GSEA

gs = GSEA(
    data=gene_exp,
    gene_sets='KEGG_2016',
    classes=class_vector,          # or cls=class_vector
    permutation_type='phenotype',
    permutation_num=1000,
    outdir=None,
    method='signal_to_noise',
    threads=4,
    seed=8,
)
gs.pheno_pos = "AML"               # name of the positive phenotype
gs.pheno_neg = "ALL"               # name of the negative phenotype
gs.run()
```

```
2026-06-24 11:24:19,582 [WARNING] Found duplicated gene names, values averaged by gene␣
↪names!
```

5.2.14 GSEA plots

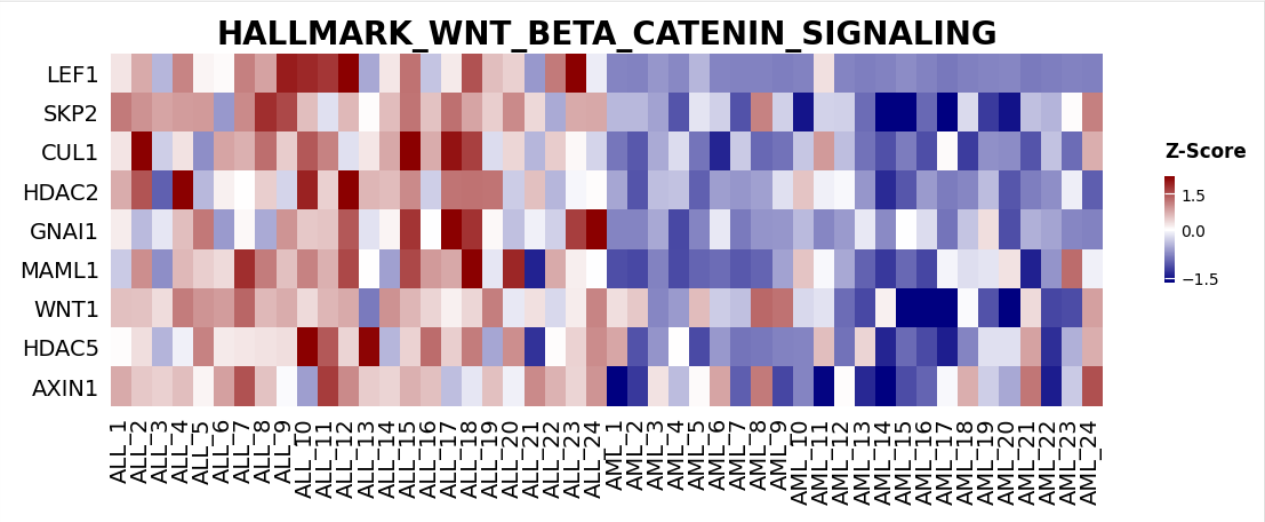
The gsea module can auto-generate per-set heatmaps in the background. To plot yourself, use the snippets below.

[51]:
```
# Running enrichment score for the top 5 terms.
terms = gs_res.res2d.Term
axs = gs_res.plot(terms[:5], show_ranking=False, legend_kws={'loc': (1.05, 0)})
```



[52]:
```
# Low-level equivalent with gseaplot2:
# from gseapy import gseaplot2
# terms = gs_res.res2d.Term[:5]
# hits  = [gs_res.results[t]['hits'] for t in terms]
# runes = [gs_res.results[t]['RES'] for t in terms]
# fig = gseaplot2(terms=terms, ress=runes, hits=hits,
#                 rank_metric=gs_res.ranking, figsize=(4, 5))
```

[53]:
```
from gseapy import heatmap

# Heatmap of the leading-edge genes for one enriched term.
i = 2
genes = gs_res.res2d.Lead_genes[i].split(";")
# z_score=0 standardizes across rows (genes). Pass ofname='figure.pdf' to save.
ax = heatmap(df=gs_res.heatmat.loc[genes], z_score=0, title=terms[i], figsize=(14, 4))
```



[54]:
```
# The underlying expression values behind that heatmap.
gs_res.heatmat.loc[genes]
```

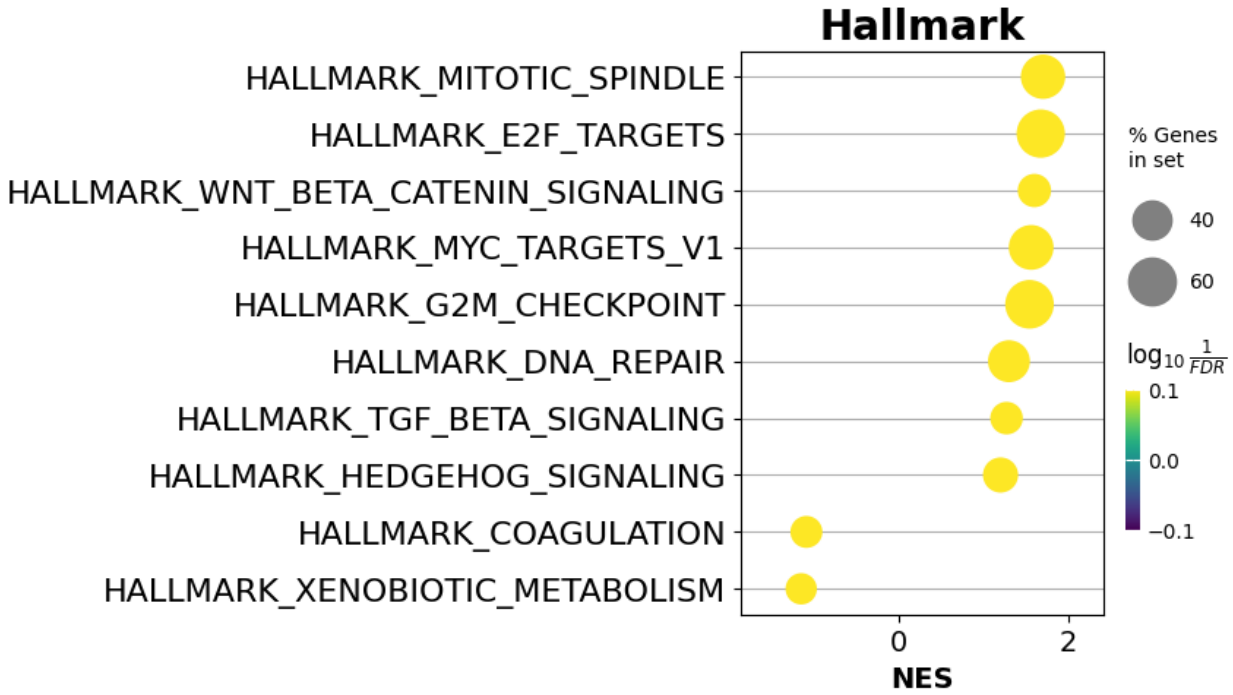
[54]:
```
          ALL_1    ALL_2   ALL_3    ALL_4   ALL_5   ALL_6    ALL_7    ALL_8  \
Gene
LEF1   8544.10  12552.0  2869.0  15265.0  7446.0  6991.0  15520.0  13114.0
SKP2     23.80    -45.0   -95.0    -71.0   -65.0  -547.0    -24.0    230.0
CUL1   1712.75   3309.0  1273.5   1726.5   947.5  2160.0   2065.0   2524.5
HDAC2  4542.90   6030.0  1195.0   9368.0  2281.0  3407.0   3175.0   3962.0
GNAI1   588.50    163.0   364.0    882.0  1317.0    17.0    518.0     89.0
MAML1   871.40   1871.0   578.0   1589.0  1448.0  1364.0   2494.0   1989.0
WNT1   -872.50   -875.0 -1012.0   -535.0  -654.0  -694.0   -421.0   -827.0
HDAC5  2137.20   2374.0  1651.0   2012.0  3132.0  2279.0   2314.0   2349.0
AXIN1  -433.50   -722.0  -808.0   -623.0 -1167.0  -326.0    448.0   -661.0

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

       AML_21  AML_22  AML_23  AML_24
Gene
LEF1   -242.0   -47.0   176.0    14.0
SKP2   -464.0  -490.0  -333.0     7.0
CUL1    646.0  1214.5   770.0  2088.5
HDAC2  1571.0  1749.0  2942.0  1174.0
GNAI1   115.0    55.0   -80.0   -94.0
MAML1    48.0   609.0  2090.0  1056.0
WNT1   -995.0 -1861.0 -1835.0  -714.0
```

(continues on next page)

(continued from previous page)

```
HDAC5  2872.0   848.0  1629.0  2763.0
AXIN1    93.0 -2951.0 -1666.0   471.0

[9 rows x 48 columns]
```

[55]:
```
from gseapy import dotplot

ax = dotplot(
    gs_res.res2d,
    column="FDR q-val",
    title='Hallmark',
    cmap=plt.cm.viridis,
    size=5,
    figsize=(4, 5),
    cutoff=1,               # show all terms (no significance filter)
)
```



5.2.15 Command-line equivalent

[56]:
```
# !gseapy gsea -d ./tests/extdata/Leukemia_hgu95av2.trim.txt \
#              -g KEGG_2016 -c ./tests/extdata/Leukemia.cls \
#              -o test/gsea_report \
#              -v --no-plot \
#              -t phenotype
```

Single-sample GSEA (ssgsea)

`gp.ssgsea()` scores **each sample independently**, producing an enrichment score per gene set per sample — useful when you do not have two phenotype groups to contrast.

Not sure whether to use `prerank` or `ssgsea`? See the FAQ.

Accepted ``data`` inputs: a `.txt/.gct` file, a `DataFrame` (genes × samples), or a `Series` indexed by gene. `gene_sets` accepts the usual name / `.gmt` / dict.

[57]:
```
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',    # 'rank' ranks genes per sample; 'custom' uses raw values
    no_plot=True,
)
```

```
2026-06-24 11:24:23,205 [WARNING] Found duplicated gene names, values averaged by gene␣
↪names!
```

[58]:
```
ss.res2d.head()
```

[58]:

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

[59]:
```
# ssGSEA also accepts a DataFrame or Series (gene symbols as the index).
ssdf = pd.read_csv("./tests/data/temp.rnk", header=None, index_col=0, sep="\t")
ssdf.head()
```

[59]:

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

[60]:
```
# Collapse the single-column DataFrame to a Series.
ssdf2 = ssdf.squeeze()
```

[61]:
```
# Series / DataFrame input is supported directly.
temp = gp.ssgsea(data=ssdf2, gene_sets="./tests/data/temp.gmt")
```

5.2.16 Access the enrichment scores (ES / NES)

Results live on `obj.res2d`. Pivot to a term × sample matrix for downstream use.

[62]:
```
ss.res2d.sort_values('Name').head()
```

[62]:

	Name	Term	ES	NES
601	ALL_1	HALLMARK_PANCREAS_BETA_CELLS	-1280.654659	-0.267153

(continues on next page)

(continued from previous page)

```
934   ALL_1                  HALLMARK_APOPTOSIS   970.818772  0.202519
1774  ALL_1          HALLMARK_HEDGEHOG_SIGNALING   431.446694  0.090003
279   ALL_1  HALLMARK_INTERFERON_ALPHA_RESPONSE  1721.458034  0.359108
1778  ALL_1       HALLMARK_BILE_ACID_METABOLISM  -429.127871 -0.089519
```

[63]:
```
# Reshape to NES values, terms in rows and samples in columns.
nes = ss.res2d.pivot(index='Term', columns='Name', values='NES')
nes.head()
```

[63]:
```
Name                              ALL_1    ALL_10    ALL_11    ALL_12  \
Term
HALLMARK_ADIPOGENESIS          0.287384  0.274548  0.290059  0.285388
HALLMARK_ALLOGRAFT_REJECTION   0.061770  0.028062  0.096589  0.080713
HALLMARK_ANDROGEN_RESPONSE     0.133453  0.113911  0.193074  0.201531
HALLMARK_ANGIOGENESIS         -0.113481 -0.182411 -0.195637 -0.094817
HALLMARK_APICAL_JUNCTION       0.051372  0.063763  0.054601  0.014385

Name                              ALL_13    ALL_14    ALL_15    ALL_16  \
Term
HALLMARK_ADIPOGENESIS          0.322757  0.305239  0.275686  0.266209
HALLMARK_ALLOGRAFT_REJECTION   0.082701  0.102735  0.125250  0.147262
HALLMARK_ANDROGEN_RESPONSE     0.151001  0.129670  0.173563  0.144836
HALLMARK_ANGIOGENESIS         -0.163717 -0.139243 -0.119084 -0.154526
HALLMARK_APICAL_JUNCTION       0.049019  0.052690  0.064787  0.052192

Name                              ALL_17    ALL_18  ...     AML_22    AML_23  \
Term                                                ...
HALLMARK_ADIPOGENESIS          0.315803  0.282617  ...   0.277755  0.261477
HALLMARK_ALLOGRAFT_REJECTION   0.124621  0.091077  ...   0.185738  0.157852
HALLMARK_ANDROGEN_RESPONSE     0.180214  0.180801  ...   0.180443  0.188891
HALLMARK_ANGIOGENESIS         -0.068290 -0.121156  ...   0.054883 -0.023782
HALLMARK_APICAL_JUNCTION       0.056070  0.064936  ...   0.109270  0.090065

Name                              AML_24     AML_3     AML_4     AML_5  \
Term
HALLMARK_ADIPOGENESIS          0.200083  0.312948  0.342963  0.253282
HALLMARK_ALLOGRAFT_REJECTION   0.055585  0.218827  0.172395  0.199077
HALLMARK_ANDROGEN_RESPONSE     0.197979  0.174892  0.142850  0.184843
HALLMARK_ANGIOGENESIS          0.119022 -0.067741  0.048430  0.012808
HALLMARK_APICAL_JUNCTION       0.155801  0.091556  0.110045  0.101659

Name                              AML_6     AML_7     AML_8     AML_9
Term
HALLMARK_ADIPOGENESIS          0.298924  0.410395  0.387433  0.343606
HALLMARK_ALLOGRAFT_REJECTION   0.158945  0.138350  0.110787  0.121643
HALLMARK_ANDROGEN_RESPONSE     0.157449  0.162843  0.180475  0.181878
HALLMARK_ANGIOGENESIS          0.032505 -0.024058 -0.039492 -0.043769
HALLMARK_APICAL_JUNCTION       0.128808  0.095511  0.080076  0.098644

[5 rows x 48 columns]
```

Warning: if you set `permutation_num > 0`, `ssgsea` behaves like `prerank` but with the ssGSEA statistic. **Do not** do this unless you know exactly why you need it.

```
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',
    permutation_num=20,   # > 0 makes it behave like prerank
    no_plot=True,
    threads=4, seed=9,
)
ss_permut.res2d.head(5)
```

5.2.17 Command-line equivalent

[64]:
```
# !gseapy ssgsea -d ./tests/extdata/Leukemia_hgu95av2.trim.txt \
#                -g ./tests/data/temp.gmt \
#                -o test/ssgsea_report \
#                -p 4 --no-plot
```

Gene Set Variation Analysis (gsva)

`gp.gsva()` is GSEApy's port of GSVA — an unsupervised method that turns a gene × sample expression matrix into a **gene-set × sample** matrix of enrichment scores, with no phenotype labels required.

[65]:
```
es = gp.gsva(
    data='./tests/extdata/Leukemia_hgu95av2.trim.txt',
    gene_sets='./tests/extdata/h.all.v7.0.symbols.gmt',
    outdir=None,
)
```

```
2026-06-24 11:24:23,575 [WARNING] Found duplicated gene names, values averaged by gene␣
↪names!
```

[66]:
```
# Reshape to a gene-set x sample matrix of GSVA enrichment scores.
es.res2d.pivot(index='Term', columns='Name', values='ES').head()
```

[66]:
```
Name                              ALL_1    ALL_10    ALL_11    ALL_12  \
Term
HALLMARK_ADIPOGENESIS         -0.213310 -0.080960  0.003289 -0.017909
HALLMARK_ALLOGRAFT_REJECTION  -0.210468 -0.373787 -0.086016 -0.169623
HALLMARK_ANDROGEN_RESPONSE    -0.136330 -0.308572  0.008126  0.048490
HALLMARK_ANGIOGENESIS          0.035895 -0.287645 -0.214951 -0.291145
HALLMARK_APICAL_JUNCTION      -0.088652 -0.128757 -0.050282 -0.248682

Name                              ALL_13    ALL_14    ALL_15    ALL_16  \
Term
HALLMARK_ADIPOGENESIS          0.207841  0.023294 -0.085392 -0.221273
HALLMARK_ALLOGRAFT_REJECTION  -0.158775 -0.016488 -0.050703  0.104430
HALLMARK_ANDROGEN_RESPONSE    -0.061181 -0.203036  0.070416 -0.125240
HALLMARK_ANGIOGENESIS         -0.311917 -0.236717 -0.345662 -0.250202
HALLMARK_APICAL_JUNCTION      -0.145164  0.001997 -0.082962 -0.091691

Name                              ALL_17    ALL_18  ...     AML_22    AML_23  \
Term                                                ...
```

(continues on next page)

(continued from previous page)

```
HALLMARK_ADIPOGENESIS          0.161470 -0.018250  ...  0.033440 -0.190436
HALLMARK_ALLOGRAFT_REJECTION -0.075816 -0.193654  ...  0.023653  0.032892
HALLMARK_ANDROGEN_RESPONSE     0.080075  0.022248  ...  0.031898  0.064394
HALLMARK_ANGIOGENESIS         -0.233296 -0.318353  ...  0.244374 -0.076852
HALLMARK_APICAL_JUNCTION      -0.168941 -0.139766  ...  0.005859 -0.067385

Name                              AML_24     AML_3     AML_4     AML_5  \
Term
HALLMARK_ADIPOGENESIS         -0.098500  0.105208  0.196799 -0.296305
HALLMARK_ALLOGRAFT_REJECTION -0.113577  0.307030  0.134581  0.188905
HALLMARK_ANDROGEN_RESPONSE     0.070232  0.199349 -0.079399 -0.016658
HALLMARK_ANGIOGENESIS         -0.010928 -0.210787  0.387912  0.269447
HALLMARK_APICAL_JUNCTION       0.062719 -0.022434  0.076593  0.138664

Name                              AML_6     AML_7     AML_8     AML_9
Term
HALLMARK_ADIPOGENESIS         -0.084042  0.450832  0.226921  0.209835
HALLMARK_ALLOGRAFT_REJECTION  0.132169  0.024078 -0.092054 -0.195987
HALLMARK_ANDROGEN_RESPONSE    -0.127327  0.018847  0.121426  0.163149
HALLMARK_ANGIOGENESIS          0.348230  0.157249  0.075479 -0.064515
HALLMARK_APICAL_JUNCTION       0.240647  0.039307  0.016764  0.057512

[5 rows x 48 columns]
```

5.2.18 Command-line equivalent

[67]:
```
# !gseapy gsva -d ./tests/data/expr.gsva.csv \
#              -g ./tests/data/geneset.gsva.gmt \
#              -o test/gsva_report
```

Replot (`replot`)

`gp.replot()` re-creates GSEA plots from the output of the **Broad Institute's Java GSEA** application. It needs the `edb/` folder produced by that tool, with this layout:

```
data/
└── edb/
    ├── C1OE.cls
    ├── gene_sets.gmt
    ├── gsea_data.gsea_data.rnk
    └── results.edb
```

[68]:
```
rep = gp.replot(indir="./tests/data", outdir="tests/replot_test")
```

5.2.19 Command-line equivalent

[69]:
```
# !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]:
```
'1.1.5'
```

5.3.1 Read Demo Data

Convert demo data from seurat to scanpy

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

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

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

[5]:

	orig.ident	nCount_RNA	nFeature_RNA	stim	seurat_annotations
AAACATACATTTCC.1	IMMUNE_CTRL	3017.0	877	CTRL	CD14 Mono
AAACATACCAGAAA.1	IMMUNE_CTRL	2481.0	713	CTRL	CD14 Mono
AAACATACCTCGCT.1	IMMUNE_CTRL	3420.0	850	CTRL	CD14 Mono
AAACATACCTGGTA.1	IMMUNE_CTRL	3156.0	1109	CTRL	pDC
AAACATACGATGAA.1	IMMUNE_CTRL	1868.0	634	CTRL	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
Mk                  STIM     121
                    CTRL     115
NK                  STIM     321
                    CTRL     298
T activated         STIM     333
                    CTRL     300
pDC                 STIM      81
                    CTRL      51
Name: count, 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")
```

(continues on next page)

(continued from previous page)

```
#     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'
```

5.3.2 GSEA

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

```
/Users/fangzq/Github/GSEApy/gseapy/gsea.py:173: UserWarning: Boolean Series key will be␣
↪reindexed to match DataFrame index.
  df = df[df_std.abs().sum(axis=1) > 0]
```

```
39.00995206832886
```

[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	

(continues on next page)

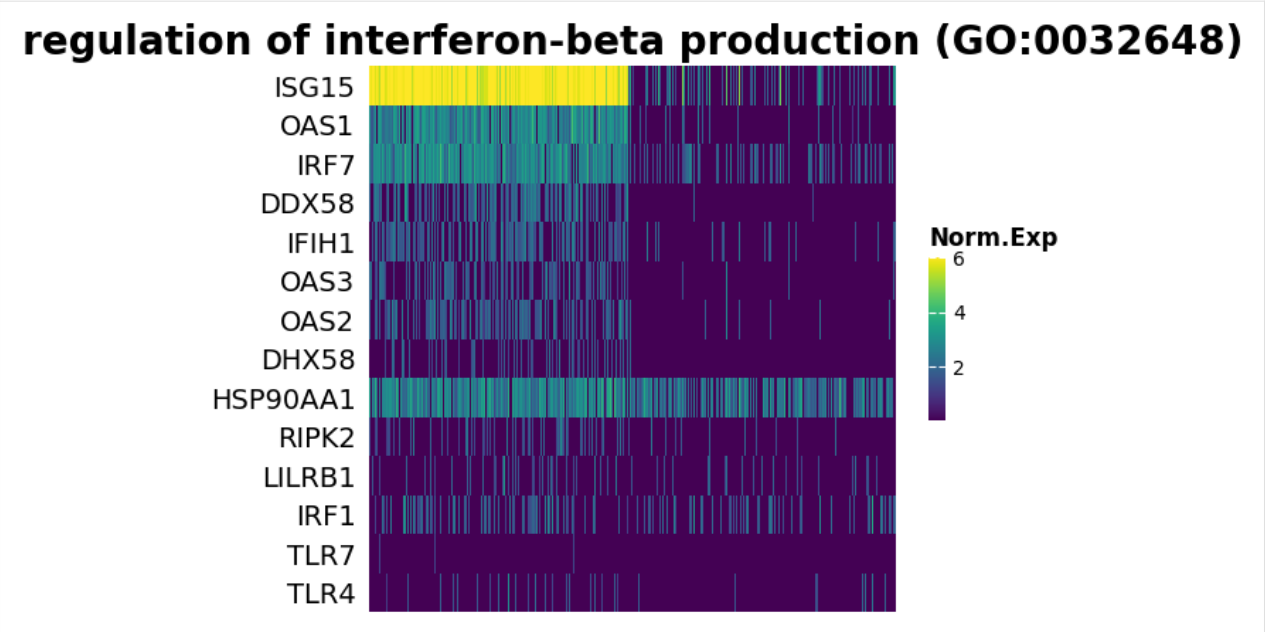
(continued from previous page)

```
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   99/490   5.14%
1   3.66143       0.0       0.0        0.0   52/188   5.33%
2  3.549856       0.0       0.0        0.0   42/140   6.07%
3  3.438759       0.0       0.0        0.0   42/108   2.85%
4   3.37735       0.0       0.0        0.0   31/120   4.49%
5  3.362051       0.0       0.0        0.0   41/100   2.85%
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...
```

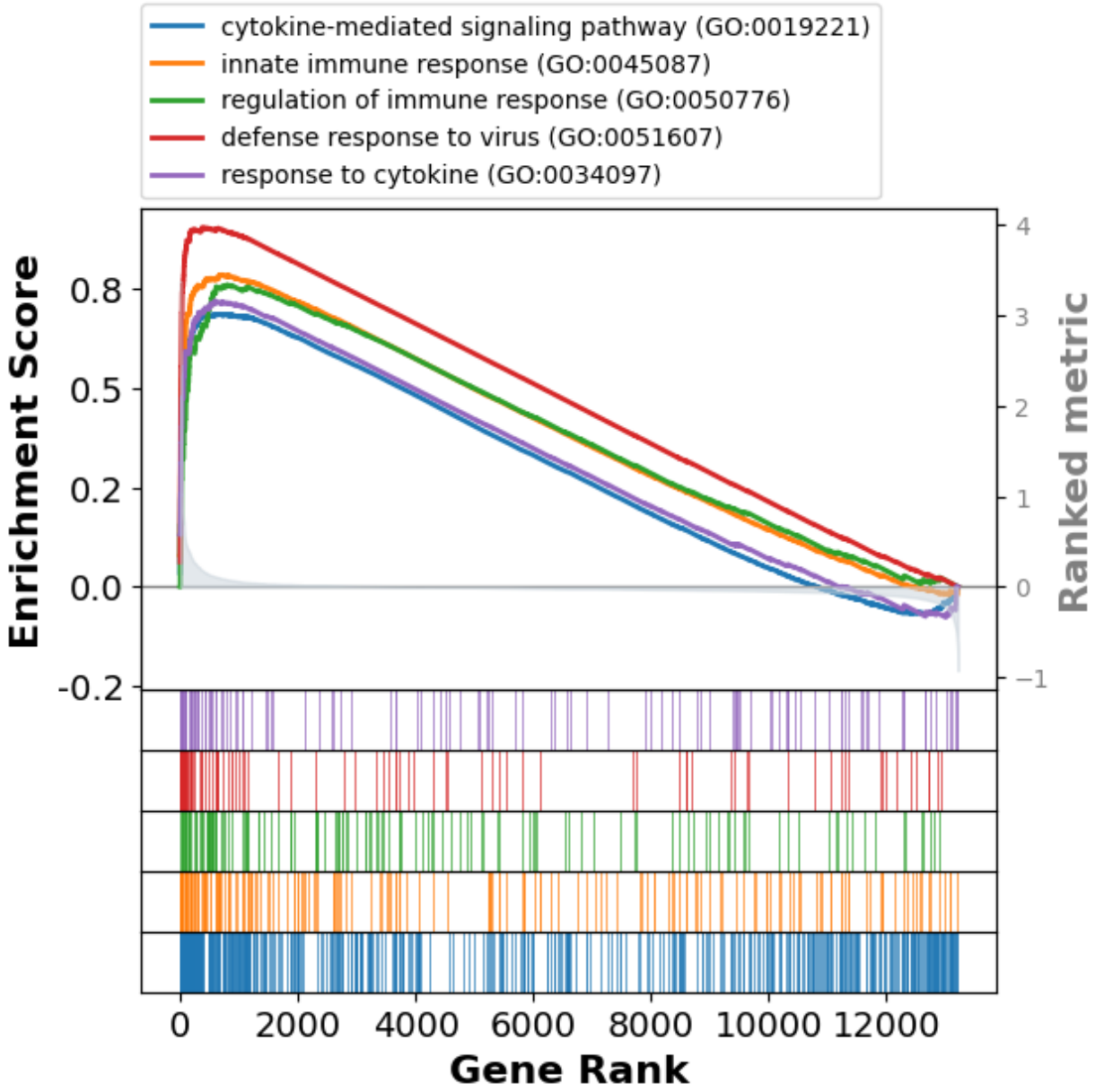
[14]:
```
res.ranking.shape # raking metric
```

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

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



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

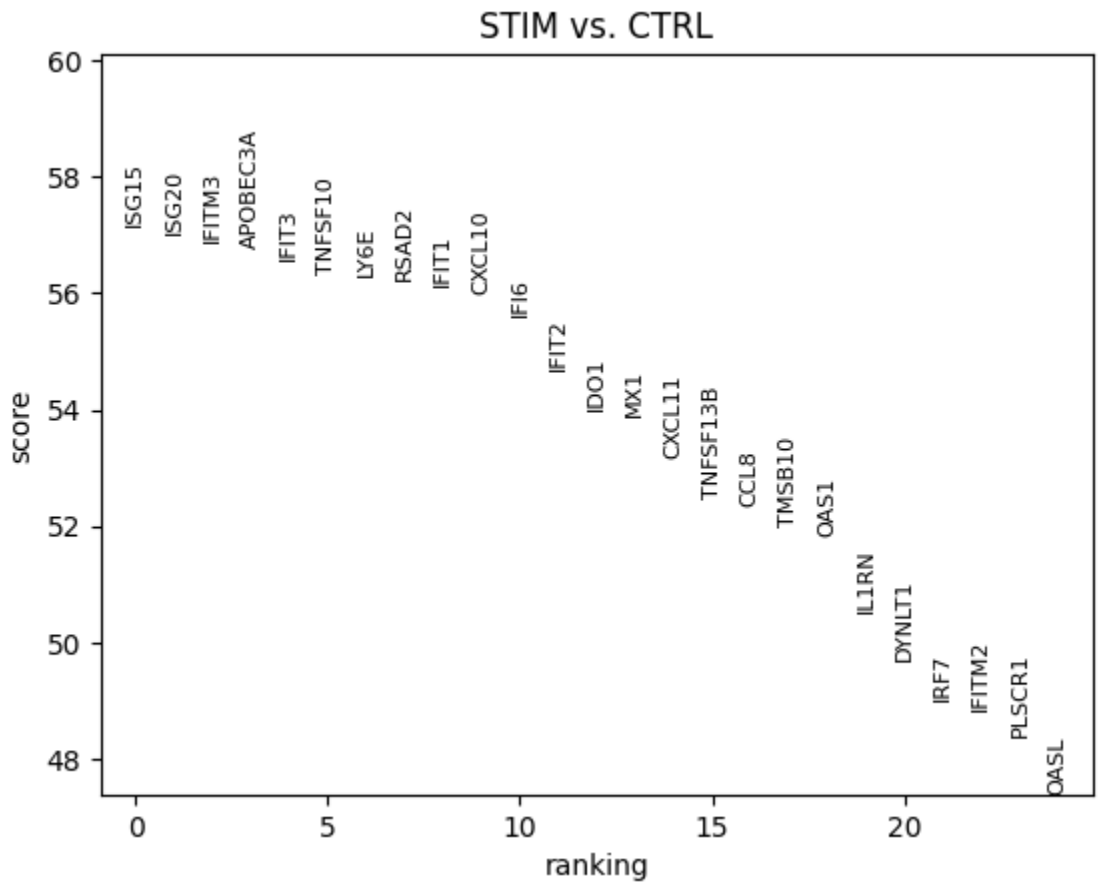


[ ]:

5.3.3 DEG Analysis

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

[18]:
```
sc.pl.rank_genes_groups(bdata, n_genes=25, sharey=False)
```



[19]:
```
# 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
↪']})
```

[20]:
```
degs.head()
```

[20]:

	STIM_names	STIM_scores	STIM_pvals	STIM_pvals_adj	STIM_logfoldchanges
0	ISG15	57.165920	0.0	0.0	8.660480
1	ISG20	57.010384	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

Prerank

[21]:
```
pre_res = gp.prerank(degs.loc[:,['STIM_names', 'STIM_logfoldchanges']], gene_sets='KEGG_
↪2016')
```

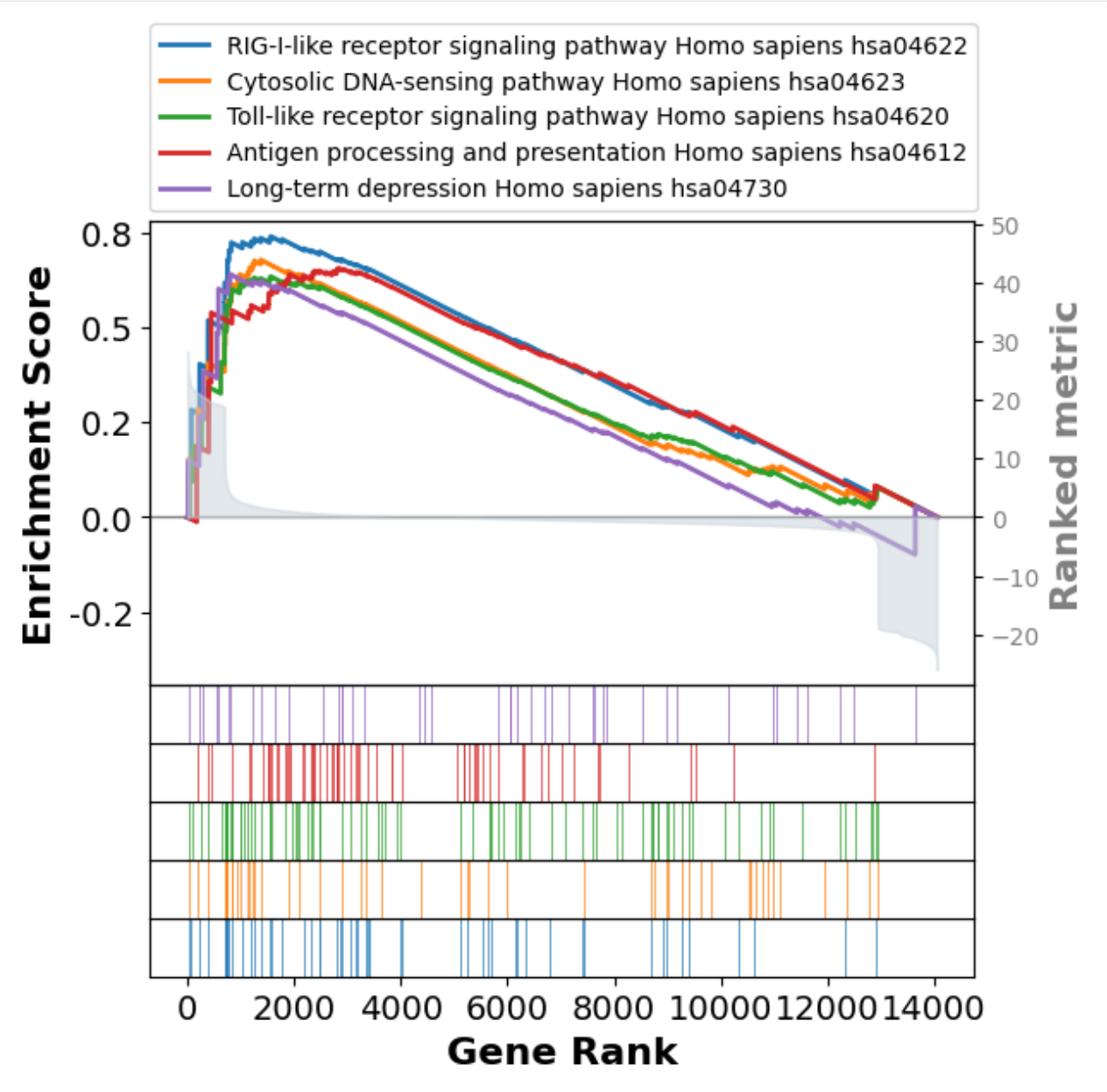
```
2025-02-04 15:49:59,570 [WARNING] Duplicated values found in preranked stats: 6.95% of␣
↪genes
The order of those genes will be arbitrary, which may produce unexpected results.
```

[22]:
```
pre_res.res2d.head(5)
```

[22]:

	Name	Term	ES	NES	NOM p-val	FDR q-val	FWER p-val	Tag %	Gene %	Lead_genes
0	prerank	RIG-I-like receptor signaling pathway Homo sap...	0.739246	1.72491	0.00625	0.099502	0.068	16/52	11.12%	IFNB1;IFNE;IFNW1;IKBKG;CXCL10;ISG15;DDX58;DHX5...
1	prerank	Cytosolic DNA-sensing pathway Homo sapiens hsa...	0.677289	1.599571	0.003226	0.377562	0.391	16/49	9.78%	IFNB1;POLR3B;IKBKG;CXCL10;ZBP1;IL6;DDX58;IRF7;...
2	prerank	Toll-like receptor signaling pathway Homo sapi...	0.633578	1.584673	0.0	0.303049	0.454	23/87	11.12%	IFNB1;TLR3;PIK3R3;IKBKG;CTSK;CXCL10;CXCL11;IL6...
3	prerank	Antigen processing and presentation Homo sapie...	0.655068	1.568956	0.003289	0.274653	0.501	31/64	20.25%	KIR2DL4;KLRC2;KIR3DL1;HSPA1A;KLRC1;TAP1;PSME2;...
4	prerank	Long-term depression Homo sapiens hsa04730	0.639757	1.470933	0.021212	0.590469	0.837	7/43	5.68%	GUCY1A3;PLCB4;PLCB2;GNA11;IGF1;GUCY1B3;CACNA1A

[23]:
```
term2 = pre_res.res2d.Term
axes = pre_res.plot(terms=term2[:5])
```



5.3.4 Over-representation analysis (Enrichr API)

[24]:
```
# 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]
```

[25]:
```
degs_up.shape
```

[25]:
```
(687, 5)
```

[26]:
```
degs_dw.shape
```

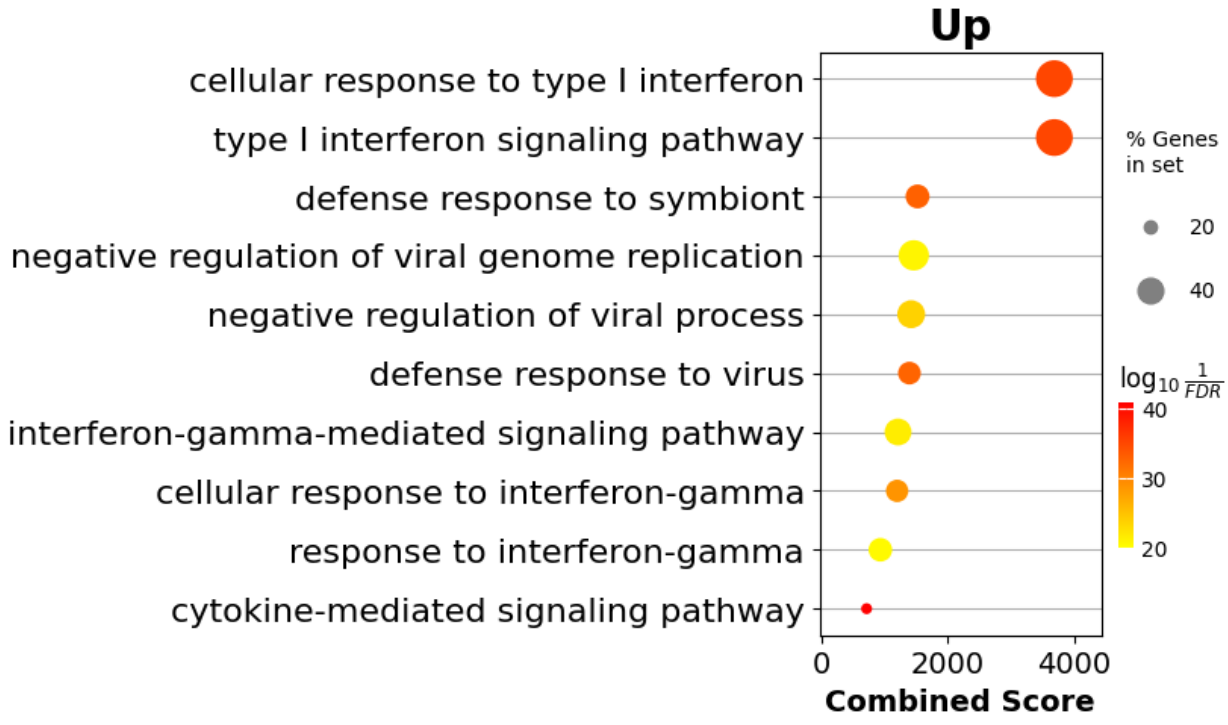
[26]:
```
(1030, 5)
```

[27]:
```
# Enricr API
enr_up = gp.enrichr(degs_up.STIM_names,
                    gene_sets='GO_Biological_Process_2021',
                    outdir=None)
```

[28]:
```
# trim (go:...)
enr_up.res2d.Term = enr_up.res2d.Term.str.split(" \(GO").str[0]
```

```
<>:2: SyntaxWarning: invalid escape sequence '\('
<>:2: SyntaxWarning: invalid escape sequence '\('
/var/folders/hp/199f0v0n5097qlbwt_5wp4dm0000gp/T/ipykernel_68875/3113486406.py:2:␣
↪SyntaxWarning: invalid escape sequence '\('
  enr_up.res2d.Term = enr_up.res2d.Term.str.split(" \(GO").str[0]
```

[29]:
```
# dotplot
gp.dotplot(enr_up.res2d, figsize=(3,5), title="Up", cmap = plt.cm.autumn_r)
plt.show()
```



[30]:
```
enr_dw = gp.enrichr(degs_dw.STIM_names,
                    gene_sets='GO_Biological_Process_2021',
                    outdir=None)
```

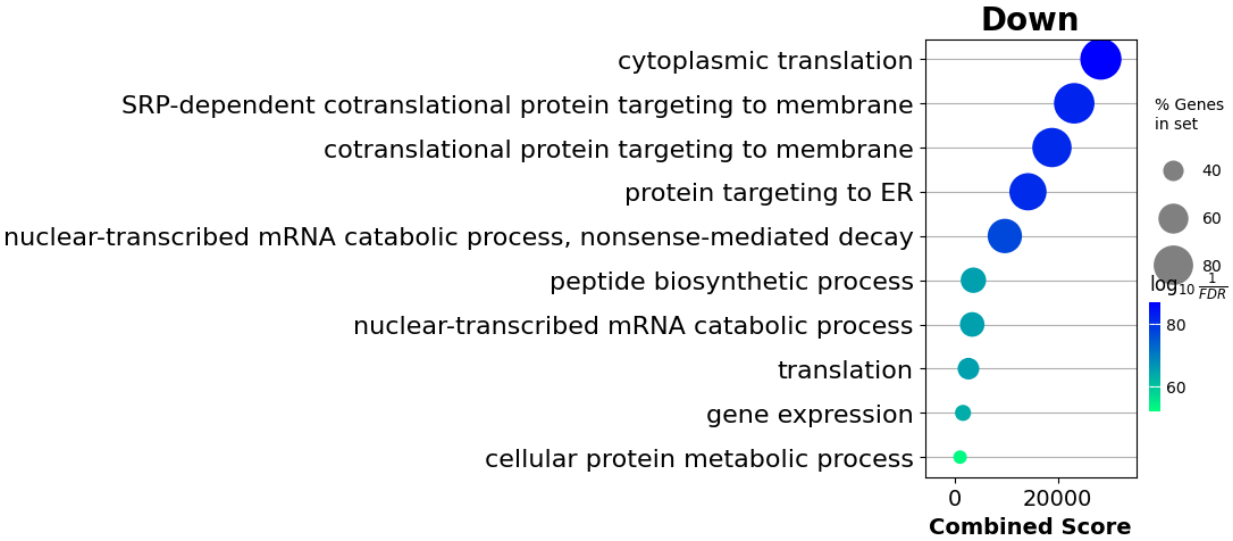
[31]:
```
enr_dw.res2d.Term = enr_dw.res2d.Term.str.split(" \(GO").str[0]
gp.dotplot(enr_dw.res2d,
           figsize=(3,5),
           title="Down",
```

(continues on next page)

(continued from previous page)

```
            cmap = plt.cm.winter_r,
            size=5)
plt.show()
```

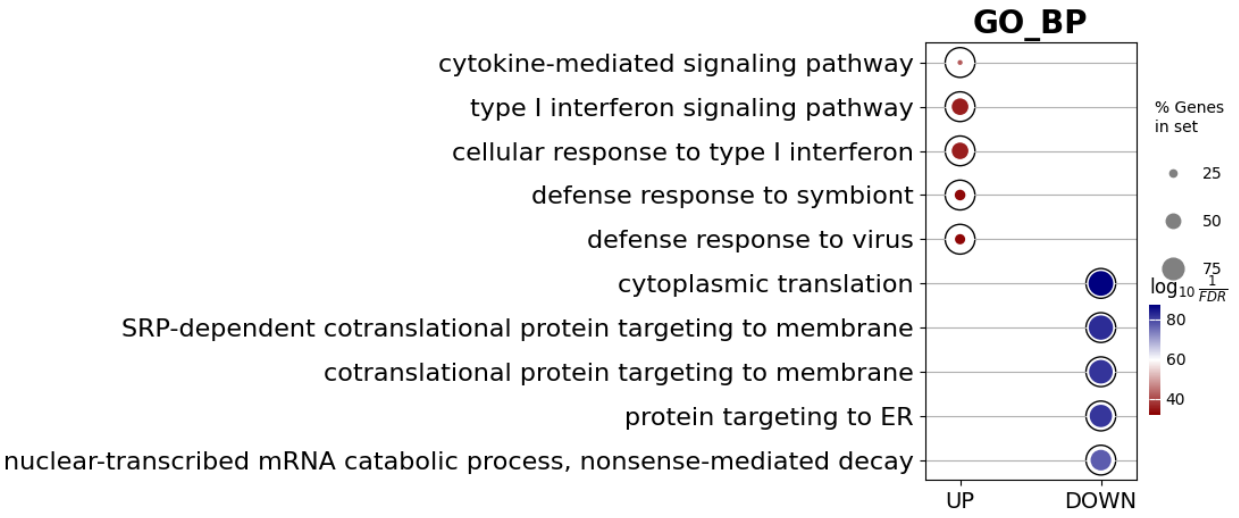
```
<>:1: SyntaxWarning: invalid escape sequence '\('
<>:1: SyntaxWarning: invalid escape sequence '\('
/var/folders/hp/199f0v0n5097qlbwt_5wp4dm0000gp/T/ipykernel_68875/1100866918.py:1:␣
↪SyntaxWarning: invalid escape sequence '\('
  enr_dw.res2d.Term = enr_dw.res2d.Term.str.split(" \(GO").str[0]
```



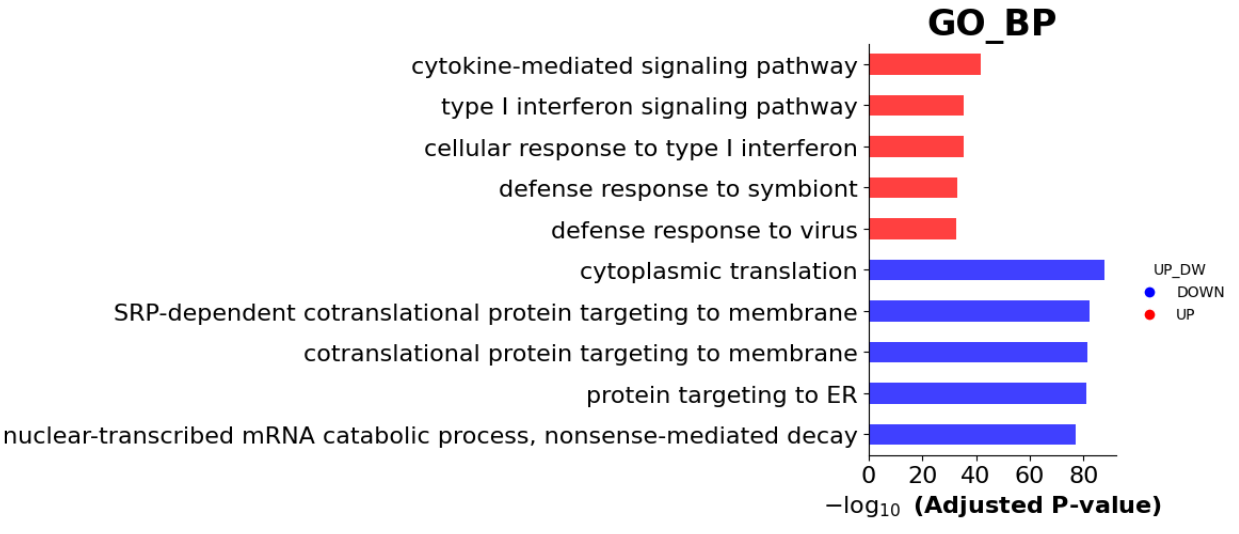
[32]:
```
# concat results
enr_up.res2d['UP_DW'] = "UP"
enr_dw.res2d['UP_DW'] = "DOWN"
enr_res = pd.concat([enr_up.res2d.head(), enr_dw.res2d.head()])
```

[33]:
```
from gseapy.scipalette import SciPalette
sci = SciPalette()
NbDr = sci.create_colormap()
# NbDr
```

[34]:
```
# 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()
```



[35]:
```
ax = gp.barplot(enr_res, figsize=(3,5),
                group ='UP_DW',
                title ="GO_BP",
                color = ['b','r'])
```



5.3.5 Network Visualization

[36]:
```
import networkx as nx
```

[37]:
```
res.res2d.head()
```

[37]:

	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

(continues on next page)

(continued from previous page)

```
  NOM p-val FDR q-val FWER p-val   Tag %  Gene %  \
0       0.0       0.0         0.0  99/490   5.14%
1       0.0       0.0         0.0  52/188   5.33%
2       0.0       0.0         0.0  42/140   6.07%
3       0.0       0.0         0.0  42/108   2.85%
4       0.0       0.0         0.0  31/120   4.49%

                                        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...
```

[38]:
```
# res.res2d.to_csv("data/test.out.txt", sep="\t", index=False)
```

[39]:
```
nodes, edges = gp.enrichment_map(res.res2d)
```

```
/Users/fangzq/Github/GSEApy/gseapy/plot.py:738: FutureWarning: Downcasting behavior in␣
↪`replace` is deprecated and will be removed in a future version. To retain the old␣
↪behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future␣
↪behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df[self.colname] = df[self.colname].replace(0, np.nan).bfill()
```

[40]:
```
nodes.head()
```

[40]:
```
          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   41/100   2.85%

                                            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
```

(continues on next page)

(continued from previous page)

```
node_idx
0           0.416149
1           0.547009
2           0.318182
3           0.494949
4           0.410000
```

[41]:
```
edges.head()
```

[41]:
```
   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)      0.145833

   overlap_coef                                      overlap_genes
0      0.203125  NCBP2,DDX39A,SRSF2,POLR2L,SNRNP40,SRSF5,POLR2G...
1      0.928571  RPS8,RPL12,RPS16,RPL23,RPL27A,MRPL51,MRPL43,RP...
2      0.051020                POLR2J,POLR2G,POLR2E,POLR2L,POLR2F
3      0.428571                     IRF1,OAS1,OAS2,IRF7,OAS3,TLR4
4      0.500000             OAS1,OAS2,IFIH1,ISG15,IRF7,OAS3,DDX58
```

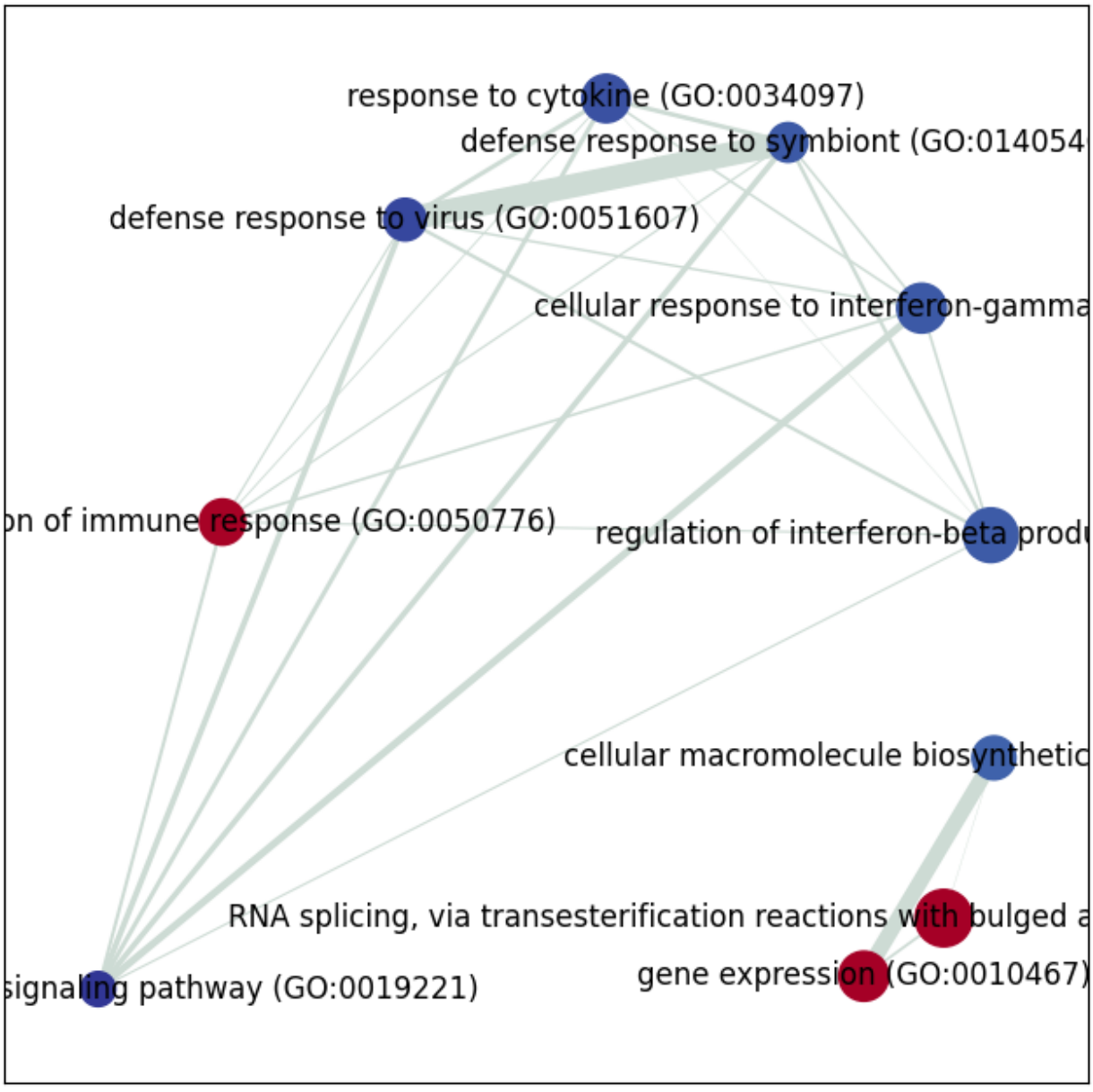
[42]:
```
# build graph
G = nx.from_pandas_edgelist(edges,
                            source='src_idx',
                            target='targ_idx',
                            edge_attr=['jaccard_coef', 'overlap_coef', 'overlap_genes'])

# Add missing node if there is any
for node in nodes.index:
    if node not in G.nodes():
        G.add_node(node)
```

[43]:
```
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))
```

(continues on next page)

(continued from previous page)

```
# draw node label
nx.draw_networkx_labels(G,
                        pos=pos,
                        labels=nodes.Term.to_dict())
# draw edge
edge_weight = nx.get_edge_attributes(G, 'jaccard_coef').values()
nx.draw_networkx_edges(G,
                       pos=pos,
                       width=list(map(lambda x: x*10, edge_weight)),
                       edge_color='#CDDBD4')
plt.show()
```



[ ]:

5.4 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

```
groups = ['C1OE', 'C1OE', 'C1OE', 'Vector', 'Vector', 'Vector']
with open('gsea/edb/C1OE.cls', "w") as cl:
   line = f"{len(groups)} 2 1\n# C10E Vector\n"
   cl.write(line)
   cl.write(" ".join(groups) + "\n")
```

Gene_sets file in gmt format.

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

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

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

An example of gmt file looks like below:

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

```
ES-SPECIFIC Arid3a_used     ACTA1   CALML4  CORO1A  DHX58   DPYS    EGR1    ESRRB   GLI2␣
↪   GPX2    HCK     INHBB
HDAC-UNIQUE     Arid3a_used 1700017B05RIK   8430427H17RIK   ABCA3   ANKRD44 ARL4A   BNC2␣
↪   CLDN3
XEN-SPECIFIC        Arid3a_used     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',
```

(continues on next page)

(continued from previous page)

```
'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',
'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',
```

(continues on next page)

(continued from previous page)

```
'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. NOTE: If multiple gene sets are provided, the FDR null distribution will be based on the combined gene sets. This may lead to slight differences in FDR values compared to running GSEA separately for each gene set. See github issue for more details: https://github.com/zqfang/GSEApy/issues/323
- **cls** – A list or a .cls file format required for GSEA.
- **organism** (*str*) – Organism for Enrichr library names (human, mouse, yeast, fly, fish, worm); does not affect custom gene sets (gmt or dict).
- **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.
- **weight** (*float*) – Refer to `algorithm.enrichment_score()`. Default:1.
- **method** – The method used to calculate a correlation or ranking. Default: 'signal_to_noise'. 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. NOTE: If multiple gene sets are provided, the FDR null distribution will be based on the combined gene sets. This may lead to slight differences in FDR values compared to running GSEA separately for each gene set. See github issue for more details: https://github.com/zqfang/GSEApy/issues/323
- **organism** (*str*) – Organism for Enrichr library names (human, mouse, yeast, fly, fish, worm); does not affect custom gene sets (gmt or dict).
- **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.
- **weight** (*str*) – Refer to `algorithm.enrichment_score()`. Default:1.
- **ascending** (*bool*) – Sorting order of rankings. Default: False for descending. If None, do not sort the ranking.
- **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.
- **method** (*str*) – P-value / significance estimation procedure. Default: 'permutation'. Choose from:

  1. 'permutation'

     Classic gene-set permutation: the null distribution of ES is built by permuting gene-set membership `permutation_num` times. NES, nominal p-value and FDR are derived from this null. Supports both a single preranked list and a multi-column ranking DataFrame.

  2. 'multilevel'

     fgsea multilevel procedure (a faithful port of the fgsea C++ core). Estimates arbitrarily small p-values via adaptive multilevel sampling instead of plain permutation, so it can resolve significance well below `1 / permutation_num`. NES is computed from fgsea's random-gene-set null (`NES = ES / mean(same-sign null ES)`), which differs by design from the classic permutation NES. Supports a single preranked list only (a multi-column DataFrame raises `NotImplementedError`).

- **sample_size** (*int*) – Only used when `method='multilevel'`. Sample size for the multilevel split step of the fgsea algorithm; larger values give more accurate (but slower) tail p-value estimates. Default: 101.
- **eps** (*float*) – Only used when `method='multilevel'`. Lower boundary for the estimated p-value; p-values smaller than `eps` are reported as `eps`. Set to 0 to estimate p-values as small as machine precision allows. Default: 1e-50.

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),
```

(continues on next page)

(continued from previous page)

```
|  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.
- **organism** (*str*) – Organism for Enrichr library names (human, mouse, yeast, fly, fish, worm); does not affect custom gene sets (gmt or dict).
- **outdir** – Results output directory. If None, nothing will write to disk.
- **sample_norm_method** (*str*) – Sample normalization method. Choose from {'rank', 'log', 'log_rank', None}. Default: rank. this argument will be used for ordering genes.

  1. 'rank': Rank your expression data, and transform by 10000*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. None or 'custom': Do nothing, and use your own rank value to calculate enrichment score.

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

Parameters

- **correl_norm_type** (*str*) – correlation normalization type. Choose from {'rank', 'symrank', 'zscore', None}. Default: rank. After ordering genes by sample_norm_method, further data transformed could be applied to get enrichment score.

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

  sample_norm_method will first transformed and rank original data. the data is named correl_vector for each sample. then correl_vector is transformed again by

  1. correl_norm_type is None or 'rank' : do nothing, genes are weighted by actual correl_vector.
  2. correl_norm_type =='symrank': symmetric ranking.
  3. correl_norm_type =='zscore': standardizes the correl_vector before using them to calculate scores.

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

Returns

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

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

`gseapy.enrichr()`

Enrichr API.

Parameters

- **gene_list** – str, list, tuple, pd.Series, pd.DataFrame. Also supports input txt file path with one gene id per row. The gene identifiers in *gene_list* should match the type used in *gene_sets*.
- **gene_sets** – str, list of Enrichr Library name(s), or custom defined gene_sets (dict, or gmt file). 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` – Organism for Enrichr library names (human, mouse, yeast, fly, fish, worm); does not affect custom gene sets (gmt or dict).

  Does not affect gmt or dict input of *gene_sets*.

- `outdir` – Output file directory

- `background` – Background gene set for statistical testing. Type: None | int | list | str.

  When is this used? - Only applies to CUSTOM gene sets (gmt file or dict) - Ignored when using Enrichr library names (e.g., 'KEGG_2016')

  Default behavior: - If None: All genes in your gene_sets will be used as background

  Recommended usage (3 options):

  Option 1: Gene list (RECOMMENDED)
  Provide your experiment-specific background genes:

  background=['gene1', 'gene2', 'gene3', ...]

  Example: All expressed genes from your RNA-seq experiment Note: Gene identifiers must match those in your gene_sets

  Option 2: Gene count (simple but less accurate)
  Specify total number of genes tested:

  background=20000 # total genes in your experiment

  Warning: Assumes all genes could be detected. May affect statistical accuracy if gene sets contain genes not in your actual background.

  Option 3: BioMart dataset (automatic download)
  Use a BioMart database name for genome-wide background:

  background='hsapiens_gene_ensembl' # human genes background='mmusculus_gene_ensembl' # mouse genes

  The program downloads all annotated genes with Entrez IDs. First download may take a few minutes; results are cached.

  Cached location: ~/.cache/gseapy/{dataset}.background.genes.txt

  Why does background matter? Background genes define the "universe" for hypergeometric testing. Using the correct background (e.g., detected genes in your experiment) improves statistical accuracy compared to using all possible genes in a genome.

- `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** – 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** – Background gene set for statistical testing. Type: None | int | list | str.

  When is this used? - Only applies to CUSTOM gene sets (gmt file or dict) - Ignored when using Enrichr library names (e.g., 'KEGG_2016')

  Default behavior: - If None: All genes in your gene_sets will be used as background

  Recommended usage (3 options):

  Option 1: Gene list (RECOMMENDED)
  Provide your experiment-specific background genes:

  background=['gene1', 'gene2', 'gene3', ...]

  Example: All expressed genes from your RNA-seq experiment Note: Gene identifiers must match those in your gene_sets

  Option 2: Gene count (simple but less accurate)
  Specify total number of genes tested:

  background=20000 # total genes in your experiment

  Warning: Assumes all genes could be detected. May affect statistical accuracy if gene sets contain genes not in your actual background.

  Option 3: BioMart dataset (automatic download)
  Use a BioMart database name for genome-wide background:

  background='hsapiens_gene_ensembl' # human genes background='mmusculus_gene_ensembl' # mouse genes

  The program downloads all annotated genes with Entrez IDs. First download may take a few minutes; results are cached.

  Cached location: ~/.cache/gseapy/{dataset}.background.genes.txt

  Why does background matter? Background genes define the "universe" for hypergeometric testing. Using the correct background (e.g., detected genes in your experiment) improves statistical accuracy compared to using all possible genes in a genome.

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

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

gseapy.**replot**()

The main function to reproduce GSEA desktop outputs.

Parameters

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

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

5.5.2 GSEA Statistics

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

GSEA main tool

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

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

Parameters

- **df** – gene_expression DataFrame.
- **method** – The method used to calculate a correlation or ranking. Default: 'log2_ratio_of_classes'. Others methods are:

1. 'signal_to_noise' (s2n) or 'abs_signal_to_noise' (abs_s2n)

You must have at least three samples for each phenotype. The more distinct the gene expression is in each phenotype, the more the gene acts as a "class marker".

2. 't_test'

   Uses the difference of means scaled by the standard deviation and number of samples. Note: You must have at least three samples for each phenotype to use this metric. The larger the t-test ratio, the more distinct the gene expression is in each phenotype and the more the gene acts as a "class marker."

3. 'ratio_of_classes' (also referred to as fold change).

   Uses the ratio of class means to calculate fold change for natural scale data.

4. 'diff_of_classes'

   Uses the difference of class means to calculate fold change for natural scale data

5. 'log2_ratio_of_classes'

   Uses the log2 ratio of class means to calculate fold change for natural scale data. This is the recommended statistic for calculating fold change for log scale data.

- **pos** (*str*) – one of labels of phenotype's names.
- **neg** (*str*) – one of labels of phenotype's names.
- **classes** (*dict*) – column id to group mapping.
- **ascending** (*bool*) – bool or list of bool. Sort ascending vs. descending.

Returns

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

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

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

Parse group (classes)

load_data() → Tuple[DataFrame, Dict]

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

run()

GSEA main procedure

to_cls(*outdir: str*)

Save group information to cls file

class gseapy.gsea.**Prerank**(*rnk: DataFrame | Series | str*, *gene_sets: List[str] | str | Dict[str, str]*, *organism: str = 'human'*, *outdir: str | None = None*, *pheno_pos='Pos'*, *pheno_neg='Neg'*, *min_size: int = 15*, *max_size: int = 500*, *permutation_num: int = 1000*, *weight: float = 1.0*, *ascending: bool | None = 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*, *method: str = 'permutation'*, *sample_size: int = 101*, *eps: float = 1e-50*)

GSEA prerank tool

load_ranking()

parse rnk input

run()

GSEA prerank workflow

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

To reproduce GSEA desktop output results.

gsea_edb_parser(*results_path*)

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

Parameters
results_path – the path of results.edb file.

Returns
a dict contains { enrichment_term: [es, nes, pval, fdr, fwer, hit_ind]}

run()

main replot function

class gseapy.base.**GMT**(*mapping: Dict[str, List[str]] | None = None*, *description: str | None = None*, *source: str | None = None*, *name: str | None = 'default'*)

A collection of gene set dictionaries with metadata.

Attributes:

_collections: Dict[str, Dict[str, Any]] - Stores gene set collections
key: collection name value: {

'genes': Dict[str, List[str]] - Gene set mappings 'description': str - Collection description 'source': str - Source of the gene sets

}

add(*mapping: Dict[str, List[str]]*, *description: str | None = None*, *source: str | None = None*, *name: str | None = 'default'*)

Add a gene set collection with metadata.

Args:
mapping: Gene set dictionary to add description: Description of the gene sets source: Source of the gene sets name: Name for this collection

filter(*min_size: int | None = None*, *max_size: int | None = None*, *gene_list: List[str] | None = None*, *collections: List[str] | None = None*) → *GMT*

Filter gene sets based on size and gene membership.

Args:
min_size: Minimum number of genes in a set max_size: Maximum number of genes in a set gene_list: Only keep genes present in this list collections: Only keep these named collections

Returns:
A new filtered GMT object

get(*name: str = 'default'*) → Dict[str, List[str]]

Get gene sets by collection name.

get_metadata(*name: str = 'default'*) → Dict[str, Any]

Get metadata for a collection.

items()

Iterate over (name, gene_sets) pairs.

classmethod read(*paths: str*, *source: str | None = None*) → *GMT*

Read GMT files into a collection.

Args:

paths: Comma-separated list of GMT file paths source: Source annotation for the files

write(*ofname: str*)

Write GMT file to disk.

class gseapy.base.**GSEAbase**(*outdir: str | None = None*, *gene_sets: List[str] | str | Dict[str, str] = 'KEGG_2016'*, *module: str = 'base'*, *threads: int = 1*, *organism: str = 'human'*, *verbose: bool = False*)

base class of GSEA.

check_uppercase(*gene_list: List[str | int]*) → bool

Check whether a list of gene names are mostly in uppercase.

Parameters

gene_list

[list, int] A list of gene names or Entrez IDs

Returns

bool

Whether the list of gene names are mostly in uppercase

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: List[str] | str | Dict[str, str]*) → Dict[str, List[str]]

load gene set dict

load_gmt_only(*gmt: List[str] | str | Dict[str, str]*) → Dict[str, List[str]]

parse gene_sets. gmt: List, Dict, Strings

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

make_unique(*rank_metric: DataFrame*, *col_idx: int*) → DataFrame

make gene id column unique by adding a digit, similar to R's make.unique

parse_gmt(*gmt: str*) → Dict[str, List[str]]

gmt parser when input is a string

plot(*terms: str | List[str]*, *colors: str | List[str] | None = None*, *legend_kws: Dict[str, Any] | None = None*, *figsize: Tuple[float, float] = (4, 5)*, *show_ranking: bool = True*, *ofname: str | None = None*)

terms: str, list. terms/pathways to show colors: str, list. list of colors for each term/pathway legend_kws: kwargs to pass to ax.legend. e.g. *loc*, *bbox_to_achor*. ofname: savefig

prepare_outdir()

create temp directory.

property results

compatible to old style

to_df(*gsea_summary: List[Dict]*, *gmt: Dict[str, List[str]]*, *rank_metric: Series | DataFrame*, *indices: List | None = None*)

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.

For 2*2 contingency table:

in query | not in query | row total

=> in gene_set | a | b | a+b => not in gene_set | c | d | c+d

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

Then, in R

x=a the number of white balls drawn without replacement
from an urn which contains both black and white balls.

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

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

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

without replacement from the total population.

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

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

oddsRatio = (1.0 * x * d) / Math.max(1.0 * b * c, 1)

where:

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

`gseapy.stats.`**`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: List[str] | str | Dict[str, str]*, *organism: str = 'human'*, *outdir: str | None = 'Enrichr'*, *background: List[str] | int | str = None*, *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*) → None

Compare the genes sent and received to get successfully recognized genes.

check_uppercase(*gene_list: List[str]*) → bool

Check whether a list of gene names are mostly in uppercase.

Parameters

gene_list
[list] A list of gene names

Returns

bool
Whether the list of gene names are mostly in uppercase

close()

Explicitly close logger handlers.

enrich_local(*gmt: Dict[str, List[str]]*) → DataFrame | None

Perform local enrichment analysis using hypergeometric test.

p-value: computed using the Fisher exact test (Hypergeometric test) z-score: Odds Ratio combined score: -log(p) * z

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

Columns: Term, Overlap, P-value, Odds Ratio, Combined Score, Adjusted_P-value, Genes

enrich_online(*genes_list: str*, *geneset_libraries: List[str]*) → DataFrame | None

Perform online enrichment analysis using Enrichr API.

filter_gmt(*gmt: Dict[str, List[str]]*, *background: Set[str]*) → Dict[str, List[str]]

Filter GMT to only include genes that exist in background.

This substantially affects the significance of the hypergeometric test.

Parameters

gmt
[dict] A dict of gene sets

background
[set] A set of custom background genes

Returns

dict
Filtered gene sets

get_background() → Set[str]

Get background genes from file or BioMart.

get_results(*gene_list: str*, *gene_set_libraries: List[str]*) → Tuple[str, DataFrame]

Get enrichment results from Enrichr API.

get_results_with_background(*gene_list: str*, *background: List[str]*, *gene_set_libraries: List[str]*) → Tuple[str, DataFrame]

Get enrichment results with custom background.

parse_background(*gmt: Dict[str, List[str]] | None = None*) → Set[str] | int

Parse and set background genes.

parse_genelists() → str

Parse gene list with single-pass processing.

parse_genesets(*gene_sets=None*) → List[Dict[str, List[str]] | str]

parse gene_sets input file type

prepare_outdir()

create temp directory.

run() → None

Run enrichr for one sample gene list against multiple libraries.

send_genes(*payload*, *url*) → Dict[str, int | str]

Send gene list to enrichr server.

set_organism() → None

Initialize EnrichrAPI base with the selected organism, setting base_url.

class gseapy.enrichr.**EnrichrAPI**(*organism: str = 'human'*)

A Python client for the modEnrichr suite and Speedrichr REST APIs.

add_background(*background_genes: List[str]*) → Dict[str, Any]

Uploads a background gene set to Speedrichr.

add_list(*genes: List[str]*, *description: str = 'Gene list'*) → Dict[str, Any]

Analyzes a gene set and returns a userListId.

add_list_speedrichr(*genes: List[str]*, *description: str = 'Gene list with background'*) → Dict[str, Any]

Uploads a gene set to Speedrichr for background analysis.

download_libraries(*libname: str*) → Dict[str, List[str]]

Download enrichr libraries

find_terms_by_gene(*gene: str*, *include_json: bool = True*, *include_setup: bool = True*) → Dict[str, Any]

Finds terms that contain a given gene across Enrichr libraries.

get_background_enrichment(*user_list_id: int*, *background_id: str*, *gene_set_library: str | List[str]*) → Dict[str, Any]

Gets enrichment results calculated against a custom background for one or more libraries.

get_enrichment(*user_list_id: int*, *gene_set_library: str | List[str]*) → Dict[str, Any]

Gets enrichment results for one or more gene set libraries.

get_libraries() → List[str]

Fetches a list of all available gene set library names for the current organism.

get_results_dataframe(*user_list_id: int*, *gene_set_library: str | List[str]*) → DataFrame

Fetches enrichment analysis results and returns a combined pandas DataFrame.

view_list(*user_list_id: int*) → Dict[str, Any]

Views an added gene set using its userListId.

exception gseapy.enrichr.**EnrichrAPIError**

Raised when Enrichr API requests fail.

exception gseapy.enrichr.**EnrichrError**

Base exception for Enrichr-related errors.

exception gseapy.enrichr.**EnrichrNetworkError**

Raised when network connectivity issues occur.

exception gseapy.enrichr.**EnrichrParseError**

Raised when parsing responses or files fails.

exception gseapy.enrichr.**EnrichrValidationError**

Raised when input validation fails.

5.5.5 BioMart API

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

query from BioMart

add_filter(*name: str*, *value: Iterable[str]*)

key: filter names value: Iterable[str]

get_attributes(*dataset: str = 'hsapiens_gene_ensembl'*)

Get available attritbutes from dataset you've selected

get_datasets(*mart: str = 'ENSEMBL_MART_ENSEMBL'*)

Get available datasets from mart you've selected

get_filters(*dataset: str = 'hsapiens_gene_ensembl'*)

Get available filters from dataset you've selected

get_marts()

Get available marts and their names.

get_xml_body()

Return only the XML body without the URL prefix.

This is suitable for POST requests where the XML is sent in the body as the 'query' form field to the biomart endpoint.

query(*dataset: str = 'hsapiens_gene_ensembl'*, *attributes: List[str] | None = None*, *filters: Dict[str, Iterable[str]] | None = None*, *filename: str | None = None*)

mapping ids using BioMart.

Parameters

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

Returns

a dataframe contains all attributes you selected.

Example:

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

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

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

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

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

Example::

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

5.5.6 Parser

gseapy.parser.**download_library**(*name: str*, *organism: str = 'human'*, *filename: str = None*) → 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' }
- **filename** (*str*) – the file name to save if not None.

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*, *save: str | None = None*, *gene_list: List[str] | None = None*) → Dict[str, List[str]]

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

Parameters

- **name** (*str*) – the gene_sets.gmt file or an enrichr library name. checkout full enrichr library name here: https://maayanlab.cloud/Enrichr/#libraries
- **organism** (*str*) – choose one from { 'Human', 'Mouse', 'Yeast', 'Fly', 'Fish', 'Worm' }. This 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.
- **save** (*str*) – the path to save the filtered gene set database.

- **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

`gseapy.parser.write_gmt(gene_sets: Dict[str, List[str]], filename: str, name: str | None = None) → None`

Write gene sets to a gmt file.

Parameters

- **gene_sets** (*dict*) – a dict object contains gene sets.
- **filename** (*str*) – the path to save the gmt file.
- **setname** (*str*) – the name of gene set database.

5.5.7 Visualization

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

`inverse(value)`

Maps the normalized value (i.e., index in the colormap) back to image data value.

Parameters

value
Normalized value.

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

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

Parameters

- **df** – GSEApy DataFrame results.
- **column** – column name in *df* to map the x-axis data. Default: Adjusted P-value
- **group** – group by the variable in *df* that will produce bars with different colors.
- **title** – figure title.
- **cutoff** – terms with *column* value < cut-off are shown. Work only for ("Adjusted P-value", "P-value", "NOM p-val", "FDR q-val")
- **top_term** – number of top enriched terms grouped by *hue* are shown.
- **ax** – Matplotlib axes. If None, create a new figure.
- **figsize** – tuple, matplotlib figsize. only used when ax is None.
- **color** – color or list or dict of matplotlib.colors. Must be reconigzed by matplotlib. if dict input, dict keys must be found in the *group*
- **ofname** – output file name. If None, don't save figure
- **wrap_width** – int, optional. Maximum characters per line for y-axis labels. Long gene set names are wrapped to fit within the figure. Default: None (no wrapping).

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

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

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

Parameters

- **df** – GSEApy DataFrame results.
- **column** – column name in *df* that map the dot colors. Default: Adjusted P-value.
- **x** – Categorical variable in *df* that map the x-axis data. Default: None.
- **y** – Categorical variable in *df* that map the y-axis data. Default: Term.
- **x_order** – bool, array-like list. Default: False. If True, peformed hierarchical_clustering on X-axis. or input a array-like list of *x* categorical levels.

- **x_order** – bool, array-like list. Default: False. If True, peformed hierarchical_clustering on Y-axis. or input a array-like list of *y* categorical levels.
- **title** – Figure title.
- **cutoff** – Terms with *column* value < cut-off are shown. Work only for ("Adjusted P-value", "P-value", "NOM p-val", "FDR q-val")
- **top_term** – Number of enriched terms to show (based on values in the *column* (colormap)).
- **size** – float, scale the dot size to get proper visualization.
- **ax** – Matplotlib axes.
- **figsize** – tuple, matplotlib figure size, only used when *ax* is None.
- **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.
- **wrap_width** – int, optional. Maximum characters per line for y-axis labels. Long gene set names are wrapped to fit within the figure. Default: None (no wrapping).

Returns

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

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

Visualize GSEApy Results. Node size corresponds to the percentage of gene overlap in a certain term of interest. Colour of the node corresponds to the significance of the enriched terms. Edge size corresponds to the number of genes that overlap between the two connected nodes. Gray edges correspond to both nodes when it is the only colour edge. When there are two different edge colours, red corresponds to 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**(*term: str*, *hits: Sequence[int]*, *nes: float*, *pval: float*, *fdr: float*, *RES: Sequence[float]*, *rank_metric: Sequence[float] | None = None*, *pheno_pos: str = ''*, *pheno_neg: str = ''*, *color: str = '#88C544'*, *figsize: Tuple[float, float] = (6, 5.5)*, *cmap: str = 'seismic'*, *ofname: str | None = None*, ***kwargs*) → List[Axes] | None

This is the main function for generating the gsea plot.

Parameters

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

return matplotlib.Figure.

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

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

return matplotlib.Figure.

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

Visualize the dataframe.

Parameters

- **df** – DataFrame from expression table.
- **z_score** – 0, 1, or None. z_score axis{0, 1}. If None, not scale.
- **title** – figure title.
- **figsize** – heatmap figsize.
- **cmap** – matplotlib colormap. e.g. "RdBu_r".
- **xticklabels** – bool, whether to show xticklabels.
- **xticklabels** – bool, whether to show xticklabels.
- **ofname** – output file name. If None, don't save figure.
- **ax** – matplotlib axes. Default: None.

Returns

ax if ofname is None.

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

ringplot is deprecated, use dotplot instead

Parameters

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

Returns

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

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

Standardize the mean and variance of the data axis Parameters.

Parameters

- **data2d** – DataFrame to normalize.
- **axis** – int, Which axis to normalize across. If 0, normalize across rows, if 1, normalize across columns. If None, don't change data

Returns

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

5.5.8 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:

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

5.6.2 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 are `gene %`, and `tag %` mean in the output?

5.6.5 Q: What `Enrichr database` are supported?

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

5.6.6 Q: Use custom defined `GMT` file input in Jupyter ?

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

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

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

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

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

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

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

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

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

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

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

5.6.10 Q: What cutoff mean in functions, like enrichr(), dotplot, barplot ?

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

5.6.11 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.

5.6.12 Q: What the difference between ssGSEA and Prerank

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

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

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

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

H

I

L

M

P

Q

R

S

T

V

W

Z