유전자 발현량을 통한
약물 후보 제시

생물학 공용DB를 이용한 데이터 프로세싱

팀프로젝트 결과 보고서

생명과학과 오승원

의생명공학과 지유빈

Introduction

질병은 다양한 생물학적, 환경적 요인과 진행 과정이 복합적으로 작용하며 나타나는 현상이며, 이것이 축적되면 유전자 발현 조절에 교란이 동반되기도 한다. 이런 교란은 특정 유전자 발현의 증가 혹은 감소로 나타나며 이는 질병 상태의 세포에서 나타나는 특징 중 하나이다. 특정 유전자의 발현이 과하게 증가하거나 억제되면 세포의 정상적인 기능 수행이 장애를 일으키며 이것이 지속되는 경우 조직, 기관 수준의 장애로 이어질 수 있다. 결과적으로 발현의 교란을 방치할 시 단백질 합성의 변화, 대사 장애, 세포 자살, 시냅스 손실 같은 세포 수준의 결과에서 다른 질병 발발로 인한 합병증, 노화 촉진, 암과 같은 개체 수준의 문제까지 일으키게 된다.[1].

때문에 질병으로 인해 교란된 발현 패턴을 파악하고, 이를 회복하는 것이 치료의 핵심 전략이라고 볼 수 있다. 실제로 암과 당뇨병 같은 질병에서 유전자 발현 패턴을 조절하여 증상을 완화시키고 치료를 시도한 사례가 존재한다. 약물 아자시티딘(Azacitidine)은 암 세포의 암 억제 유전자의 발현을 활성화시켜서 암세포 사멸을 유도하는 식으로 작동하며[2], 바리시티닙(baricitinib)은 염증 반응에서 DNA와 결합하는 분자를 억제하여 관련 유전자의 활성을 억제하여 교란된 면역 반응을 억제하는 식으로 작용한다[3]. 이는 유전자의 발현량이 질병이 세포의 기능에 미치는 영향을 정량적으로 보여줄 뿐 아니라 발현 패턴의 조절 자체가 치료적 효과로 이어질 수 있음을 보여준다. 따라서 질병상태의 세포에서 관찰되는 교란을 정상으로 되돌려주는 약물을 탐색하는 것은 질병의 효과적인 치료법을 찾기 위한 전략이 된다.

본 조에서는 이런 아이디어를 기반으로 질병 상태에 있는 세포의 유전자 발현 데이터를 입력으로 받아 이를 정상 상태에 가깝게 되돌릴 수 있는 약물의 후보군을 찾기 위한 파이프라인을 제시하고자 한다. 이를 위해 본 조에서는 입력 받은 유전자 발현의 패턴과 약물로 인해 변화되는 유전자 발현의 방향성을 비교하여 치료 가능성을 가진 약물의 후보를 제시하는 방식을 제시한다.

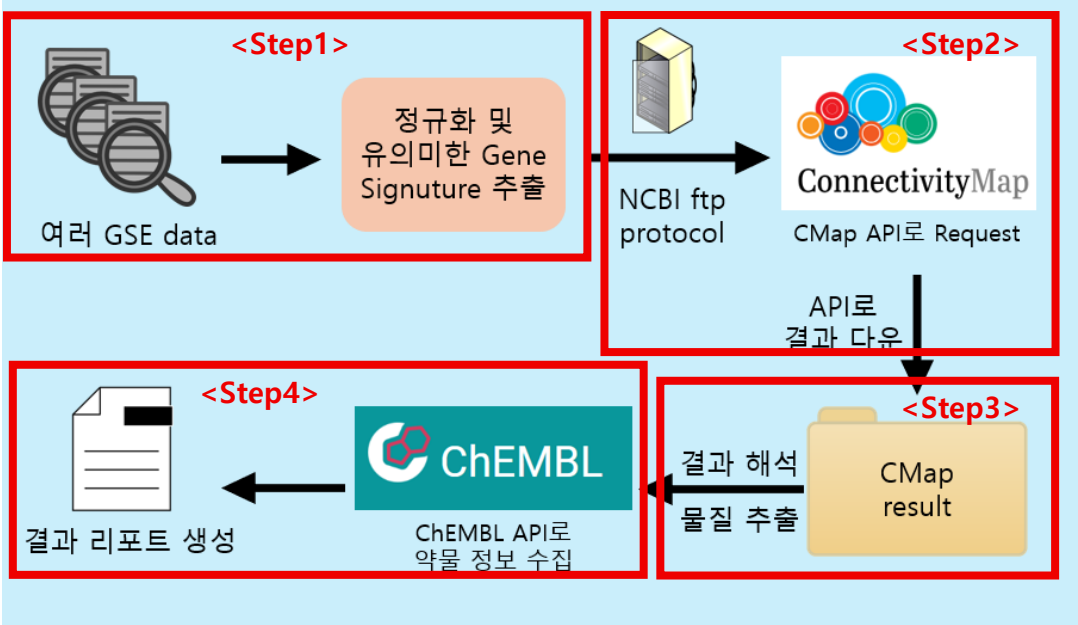


figure 1 전체 파이프라인의 모식도

figure 1은 파이프라인에 대한 모식도이다. 이후 나눠진 <Step1> ~ <Step4>의 흐름으로 설명한다.

Step1. 여러 유전자 발현데이터를 통해서 Deseq2를 이용하여 정규화를 진행하고, 유의미한 Gene 시그니처인 Up regulated gene과 Down regulated gene을 추출한다.

본 조에서 제시하는 파이프라인은 질병 상태에 있는 세포의 유전자 발현량과 정상 상태 세포의 유전자 발현량을 비교하여 질병의 유전자 발현 패턴의 변화를 파악하는 것으로 시작한다. 이를 위해 해당 분석에서는 DESeq2라는 라이브러리를 사용했다. DESeq2는 RNA-seq의 결과로 받은 count matrix를 바탕으로 조건간 유전자 발현량 비교를 지원하는 라이브러리이다. 분석할 때에 count matrix 뿐 아니라 실험의 설계 정보를 분석 모델에 포함시킬 수 있어 질병만의 요인을 다른 요인으로부터 분리하여 해석할 수 있다는 장점이 있다. 이는 질병이 만들어내는 유전자 발현량의 교란만을 추출해야 하는 본 프로젝트의 파이프라인의 요구에 부합하는 분석 패키지라고 볼 수 있다.

DESeq2를 통한 발현량 분석은 크게 3가지의 요소로 이루어져 있다. 이는 샘플별로 유전자의 발현량 데이터를 저장하고 있는 counts 데이터, 각 샘플의 상태, 유래한 실험 등과 같은, 샘플이 어떤 조건에 속하고 있는지에 대한 데이터를 담고 있는 metadata, 어떤 조건에 의한 차이를 분석할 것인지를 결정하는 design이다. 본 프로젝트의 pipeline은 질병에 영향을 받는 세포의 유전자 발현량의 변화 패턴을 DESeq2로 분석하는 것으로 시작하기 때문에 입력한 발현량 데이터 메트릭스를 counts, metadata가 요구하는 조건에 맞도록 변환시키는 것으로 pipeline이 시작되게 된다.

1	gene1	gene2	gene3
wt1	40	15	12
wt2	26	35	21
mut1	12	48	75
mut2	7	52	98

2	group	experiment
wt1	wt	exp1
wt2	wt	exp2
mut1	mut	exp1
mut2	mut	exp2

3

~experiment + group

figure 2 DESeq2에서 요구하는 필수 요소들의 간단한 예시

1. counts 데이터 프레임, 2. metadata 데이터 프레임, 3. design의 예시, design은 덧셈으로 metadata의 열이름이 연결된 형태를 하고 있으며 ~가 붙은 조건은 분석에서 그 효과를 배제하고 보겠다는 것이다.

Log2 fold change & Wald test p-value: group mut vs wt

Gene	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
ATAD3B	650.015601	-0.225744	0.295705	-0.763410	0.445219	0.999804
DDX11L17	1.434999	0.928091	1.704572	0.544472	0.586117	NaN
PRDM16	0.388774	-0.774574	3.310231	-0.233994	0.814990	NaN
PEX10	253.165980	0.038097	0.133448	0.285482	0.775275	0.999804
LINC01345	0.000000	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...
MT-TT	0.000000	NaN	NaN	NaN	NaN	NaN
MT-TP	0.000000	NaN	NaN	NaN	NaN	NaN
MAFIP	1.920109	2.479589	2.538978	0.976609	0.328763	NaN
RNA5-8SN4	27.505294	-0.273481	0.467861	-0.584534	0.558861	0.999804
RNA5-8SN5	37.230954	-0.248805	0.428071	-0.581222	0.561091	0.999804

figure 3 실제 DESeq2 결과 표

figure 3는 실제 DESeq2를 이용하여 정규화하고 나온 결과 표이다.

- baseMean: 모든 샘플에서의 평균 발현량

- log2FoldChange: mut가 wt보다 얼마나 변화하였는지 (+)면 증가, (-)면 감소

- lfcSE: log2FoldChange의 불확실성

- stat: 차이가 있다고 말할 근거의 크기(절댓값이 클수록 강함)

- pvalue: 보정 전 유의확률

- padj: 여러 유전자를 동시에 본 걸 보정한 p값(FDR)

Gene 시그니처 추출시에는 log2FoldChange와 padj를 기준으로 본다. padj값이 작을수록 유전자간의 발현량 차이가 유의미하게 크다는 뜻이고, log2FoldChange값이 음수이면 mut가 wt에 비해 적게 발현된 것이므로 Down regulated gene, log2FoldChange값이 양수이면 mut가 wt에 비해 많이 발현된 것이므로, Up regulated gene으로 분류한다.

Step2. <Step1>에서 추출된 Gene 시그니처를 CMap에 전송한다. 이후 CMap 분석이 완료되면 결과를 다운로드한다.

CMap은 유전자 발현 및 단백질체 분석을 통해 생성된 교란 데이터 세트를 분석하기 위한 클라우드 기반 소프트웨어 플랫폼이다[4]. 본 조는 CMap을 유전자 발현 분석을 위해 사용한다. CMap에는 소분자 처리, 유전자 기작 변경, 상호작용 기작 변경을 통해서 얻어낸 각 실험에 대한 Gene 시그니처를 가지고 있다. 이에, CMap에 <Step1>에서 얻은 시그니처를 API를 이용하여 쿼리로 전송하면, CMap에 있는 모든 시그니처 간의 연관성을 분석하여 -100 ~ 100사이의 connectivity score로 계산하여 결과를 보여준다. Figure 2를 보면, 왼쪽의 쿼리로 보낸 시그니처와 오른쪽에 존재하는 CMap의 모든 시그니처와 연관성을 계산한다. Method 2의 경우에는 쿼리의 시그니처와 정확히 일치하고 있는 것을 볼 수 있고, 이는 Method 2방식을 이용하면 쿼리에서의 시그니처와 동일한 양상을 보인다는 뜻이다. 이때는 Connectivity score인 TAG값이 +100으로 보일 것이다. Method 3의 경우 쿼리의 시그니처와 완전히 반전된 결과를 보이는 것을 볼 수 있다. 이는, Method 3를 이용하면 쿼리의 시그니처와 정확히 반대되는 결과를 얻을 수 있다는 것을 의미하고, 이때는 TAG값이 -100으로 나올 것이다[5]. 본 조에서는 CMap을 통해 유전자 발현 양상을 뒤집는 물질을 찾는 것이 목적이기에, 음의 값을 크게 가지는 물질들을 위주로 볼 것이다. 또한, 약물을 찾는 것이 목적이기에, Method중 BRD에 해당하는 소분자 화합물에 집중할 것이다.

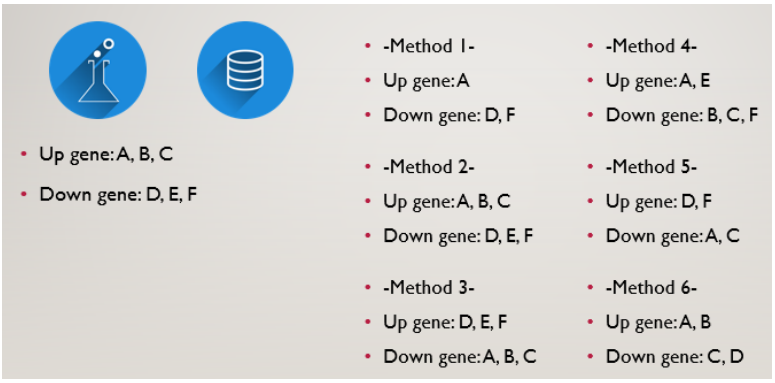


figure 4 CMap 분석 예시

Step3. 다운로드한 CMap결과를 기반으로 의미 있는 물질을 추출한다.

CMap 결과 파일은 figure 4와 같다. 해당 결과 파일은 쿼리로 전송한 시그니처와 CMap에 존재하는 시그니처 간의 분석 결과를 포함하고 있다. 이에 본 조는 cs_n1x476251.gct와 pert_id_summary.gct 파일을 집중적으로 분석한다.

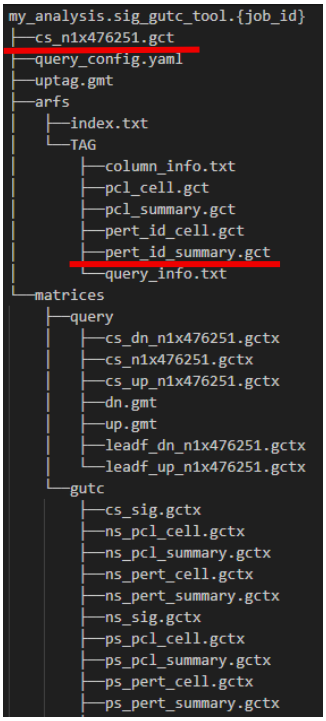


figure 5 CMap 결과 파일

1) pert_id_summary.gct: 해당 파일에는 <Step2>에서 설명한 것과 같이 CMap에서 가지고 있는 시그니처와 쿼리로 보낸 시그니처 간의 연관성이 TAG값으로 계산되어 있다. 이에 BRD를 처리하여 얻은 시그니처 중에서 TAG값을 기준으로 음의 값부터 오름차순으로 정렬하여 상위 10개 pert_id를 가져온다. figure 5를 보면 BRD 중에 TAG 값을 기준으로 오름차순 정렬하여 상위 10개의 값을 가져온 것을 볼 수 있다.

rid	cid	TAG
BRD-K92991072		-95.910004
BRD-K11757396		-95.629997
BRD-K03406345		-93.419998
BRD-K56301217		-93.290001
BRD-K86003836		-92.750000
BRD-K88761633		-91.480003
BRD-K08502430		-87.239998
BRD-K48735772		-86.790001
BRD-A81541225		-86.059998
BRD-K23984367		-85.260002

figure 6 1) 결과

2) cs_n1x476251.gct: 해당 파일에는 CMap에 존재하는 각 시그니처가 어떤 방식으로 얻었는지 메타데이터를 통해 제시되고 있다. 이에 1)에서 얻은 각 pert_id에 대응되는 pert_iname과 함께 저장한다. 이는 <Step4>에서 사용될 예정이다.

Step4. <Step3>에서 나온 물질들을 ChEMBL에 검색하여 약물관련 정보를 추출한다. 이후 취합된 정보를 기반으로 최종 보고서를 작성한다.

ChEMBL은 약물과 유사한 특성을 지닌 생체 활성 분자들을 선별하여 구축한 데이터베이스로, 신약 개발에 활용될 수 있도록 지원한다[6]. <Step3>에서 나온 물질들을 ChEMBL에 API를 이용하여 약물관련 정보를 받아온다. 받아오는 정보는 ChEMBL access ID, 공식 약물 이름, 최대 임상 단계, 치료에 사용되는 여부 등을 가져오고, 추후 유사도 검사를 위한 smiles와 같은 구조 정보를 가져온다. smiles는 화학 분자 구조를 한 줄의 문자열로 표현하는 화학 표기 언어로, 사람과 컴퓨터가 모두 이해할 수 있는 분자 구조 입력 언어이다[7]. 해당 smiles를 이용하여 ChEMBL에서 유사한 물질을 검색할 수 있다. 이후 앞선 과정과 유사하게 약물관련 정보를 가져오고, 원 물질의 smiles와 얼마나 유사한지 similarity를 추가로 가져온다. 이렇게 모인 정보들을 바탕으로 Recommendation_{job_id}.txt에 저장한다. <Step3>에서 나온 물질들은 임상정보와 상관없이 결과를 보여주고, 유사한 물질들은 최대 임상 단계가 2 이상인 경우에만 정보를 제공한다.

Method/Meterial

<Step0> 모든 워크플로우는 main.py에서 진행되었으며, 각 단계에 대한 코드들은 각각 <Step1>signiture.py, <Step2>cmap.py, <Step3>analyze.py, <Step4>recommendation.py로 나누어 모듈화하였다.

```
python main.py --same --api_key "04e156ce21731d97c6646c84a283ed10"
```

figure는 실제 terminal에서 어떻게 입력해야 실행할 수 있는지를 보여준다.

```
def main():
    parser = argparse.ArgumentParser(description="Process gene expression files.")
    # dataset_label.txt를 사용하므로 files 인자 제거
    parser.add_argument('--same', action='store_true', help="Use same protocol for all files (default: False)")
    parser.add_argument('--api_key', type=str, required=True, help="CMap API Key")
```

figure 7 main.py에서 인자 설정 코드

그림 1C에서 argparse.ArgumentParser를 이용하여 --same과 --api_key를 인자로 받을 수 있게 하였다.

--same은 여러 GSE파일을 입력으로 사용할 때, 해당 파일에 있는 실험 데이터가 모두 같은 프로토콜을 이용해 나온 결과일 경우 표기한다. 이때에는 Gene expression signiture를 추출하는 과정에서 모든 파일을 한 번에 통합하여 결과를 처리한다.

--api_key는 CMap에 사용될 API key를 인자로 입력받는다.

```
# data 폴더 생성
os.makedirs(work_path, exist_ok=True)                    (1)

# dataset_label.txt 유효성 검사
target = os.path.join(home, 'dataset_label.txt')
with open(target, 'r', encoding='utf-8') as f:
    for line in f:
        if line.startswith('#'):
            continue
        elif line.startswith('>>'):
            break
        raise ValueError("dataset_label.txt 파일에 '>>'로 시작하는 라인이 없습니다.")   (2)
```

figure 8 main.py에서 data폴더 생성 및 입력 데이터 유효성 검사

(1) 모든 중간 데이터와 결과 데이터가 포함될 ./data/라는 폴더를 생성한다.

(2) dataset_label.txt파일에 내용이 있는지 확인한다. 각 GSE파일은 실험자가 임의로 열 이름을 지정할 수 있기에, 열 이름만 가지고서 해당 실험 셋이 MUT인지, WT인지 확실하게 알 수 없다는 단점이 있었다. 이에, 실제 사용자가 dataset_label.txt에 각 파일이름과 해당 파일 안에 각 열이 MUT인지 WT인지 명시할 수 있게 하였다. (2)는 이를 입력하지 않았을 때 에러를 발생하는 역할이다.

```
# 첫 줄에는 >>각 파일 이름,
# 그 다음줄에는 각 열이 WT인지 MUT인지 적어주세요.
# ex>
# >>file1.tsv
# MUT, MUT, WT, WT
# >>file2.tsv
# MUT, WT, MUT, WT
>>test_data1.csv
MUT, WT
>>test_data2.csv
MUT, WT
>>test_data3.csv
MUT, WT
>>test_data4.csv
MUT, WT
```

figure 9 dataset_label.txt

```
# python main.py (--same) --api_key YOUR_KEY
print(f"Protocol: {'Same' if args.same else 'Different'}")

# 1) Signature 추출
up_genes, down_genes = extract_signiture(args.same, home)
print(f"Extracted Signature: {len(up_genes)} Up, {len(down_genes)} Down")

# 2) CMap 분석
result_name = cmap_analysis(args.api_key, up_genes, down_genes, work_path)

# 3) CMap 결과 처리
final_compound = get_drug_list(work_path, result_name)

# 4) 결과 추천
output_fname = result_name.replace('cmap_result', 'Recommendations').replace('.tar.gz', '.txt')
output_path = os.path.join(work_path, output_fname)
make_result(final_compound, output_path)
print(f"Analysis saved to {output_path}")
```

(1) (2) (3) (4)

figure 10 main.py에서의 흐름

(1) signiture.py (extract_signiture), Deseq2를 이용하여 유전자 발현량 데이터를 정규화하고 분석하여 Gene 시그니처(Up gene, Down gene)을 반환한다. <Step1>에서 설명한다.

(2) cmap.py (cmap_analysis), signature.py에서 얻은 Up gene과 Down gene을 이용하여 CMap에서 사용하는 Entrez ID로 변환하여 CMap API를 이용하여 쿼리를 전송한다. CMap분석이 완료되면 결과 파일을 다운로드한다. <Step2>에서 설명한다.

(3) analyze.py (get_drug_list), 앞서 다운로드 받은 결과 파일을 통해 시그니처를 반전시킬 수 있는 물질을 추출하여 반환한다. <Step3>에서 설명한다.

(4) recommendation.py (make_result), (3)에서 반환한 물질을 ChEMBL API를 이용하여 검색한다. API결과에서 ChEMBL ID와 약물 관련 정보, 추후 사용될 Smiles와 같은 구조 정보를 가져온다. 이후 Smiles를 통한 유사 물질 검색하여 약물관련 정보를 가져온다. 해당 정보들을 취합하여 Recommendation_{job_id}.txt파일에 보고서를 작성한다. <Step4>에서 설명한다.

main.py는 앞서 설명한 파이프라인을 총제적으로 하나의 파일로 다루기 위한 py파일이다.

<Step1> signiture.py에서 사용자가 만든 dataset_label.txt를 읽고 DESeq2가 요구하는 데이터프레임을 만든 후에 해당 형식으로 발현량 데이터 파일을 읽고 정규화한 후 가공하여 Gene signature인 Up regulated gene과 Down regulated gene을 추출한다. 이후 <Step2>로 제공한다.

```python
def _read_col_file(home):
    file_groups = {}
    target = os.path.join(home, 'dataset_label.txt')  ----(1)
    with open(target, 'r', encoding='utf-8') as f:
        current_key = None
        for line in f:                                  (2)
            line = line.strip()
            if not line:
                continue
            if line.startswith('#'):
                continue
            elif line.startswith('>>'):
                current_key = line.replace('>>', '').strip()
            else:
                if current_key:
                    value = line.split(',')
                    # 대소문자 통일 (mut, wt) 및 공백 제거
                    file_groups[current_key] = [val.strip().lower() for val in value]   (3)
    return file_groups
```

figure 11 signiture.py에서 _read_col_file 함수

(1) 사용자가 사전에 작성한 dataset_label.txt라는 파일을 읽는다. Figure 12에서 볼 수 있듯, 해당 파일에는 각 발현량 데이터 파일의 이름과 각 샘플(열)이 wt인지 mut인지 순서대로 적혀 있다. 이는 사용자에 따라 열 이름을 다르게 설정할 수 있기에, 사용자가 직접 명시해 줌으로써 해결하려고 하였다.

```
# 첫 줄에는 >>각 파일 이름,
# 그 다음줄에는 각 열이 WT인지 MUT인지 적어주세요.
# ex)
# >>file1.tsv
# MUT, MUT, WT, WT
# >>file2.tsv
# MUT, WT, MUT, WT
>>test_data1.csv
MUT, WT
>>test_data2.csv
MUT, WT
>>test_data3.csv
MUT, WT
>>test_data4.csv
MUT, WT
```

figure 12 실제 dataset_label.txt

(2) 빈 줄과 주석(#)을 무시하고 '>>' 표시된 부분부터 읽기 시작한다. ',' 를 기준으로 나누어 발현량 데이터 파일의 이름을 key로, 해당 파일에서의 각 열에 대한 정보를 value으로 하여 file_groups 딕셔너리를 구성한다. 구성된 딕셔너리는

{ 'test_data1.csv': ['mut', 'wt'], 'test_data2.csv': ['mut', 'wt'],
'test_data3.csv': ['mut', 'wt'], 'test_data4.csv': ['mut', 'wt'] }

와 같은 형태로 만들어진다.

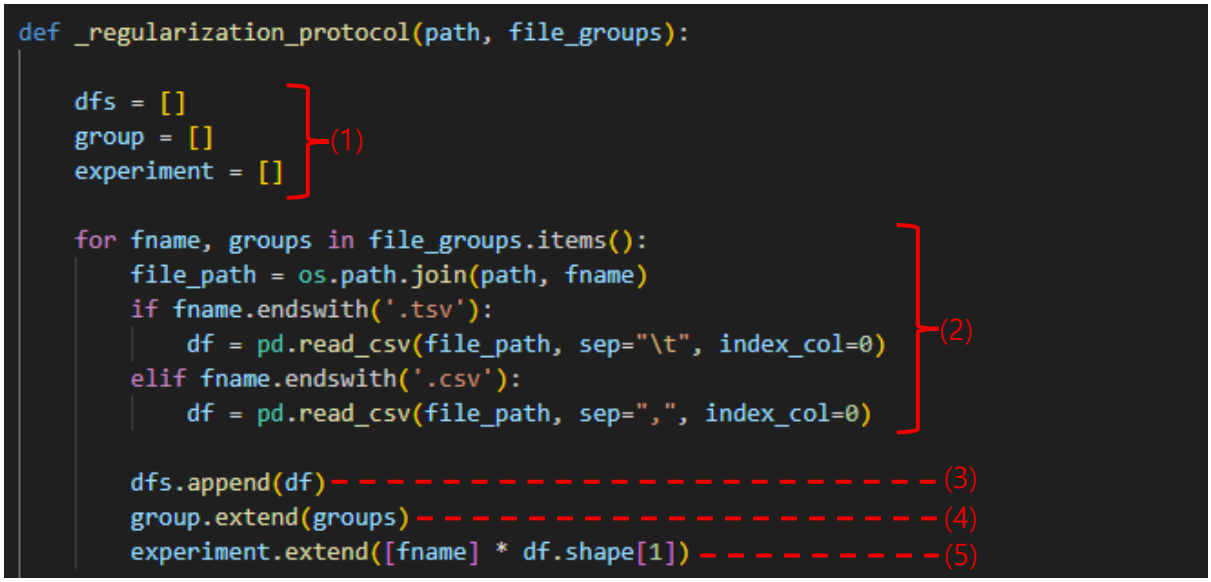


figure 13 signiture.py에서 _regularization_protocol 함수 part 1

(1) 발현량 matrix data를 저장할 리스트 dfs, 각 발현량 샘플이 wt/mut인지 순서대로 저장할 리스트 group, 각 샘플이 유래한 실험 데이터의 파일을 저장하는 리스트 experiment를 선언한다.

(2) 위의 _read_col_file 함수에서 저장한 file_groups의 key에 있는 파일에서 gene의 이름을 행 이름으로, 각 샘플을 열 이름으로 한 데이터 프레임을 생성한다.

(3) 위에서 만든 데이터프레임들을 dfs 리스트에 누적시킨다.

(4) 각 발현량 샘플의 wt/mut 정보를 group 리스트에 누적시킨다.

(5) 각 발현량 샘플이 유래한 실험 데이터 파일을 experiment 리스트에 누적시킨다.

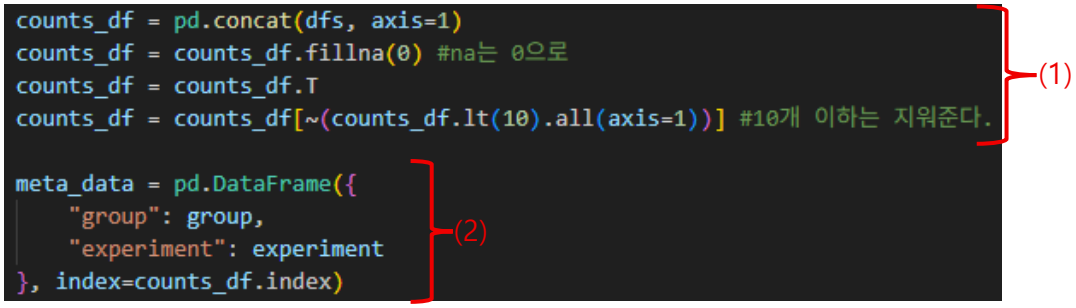
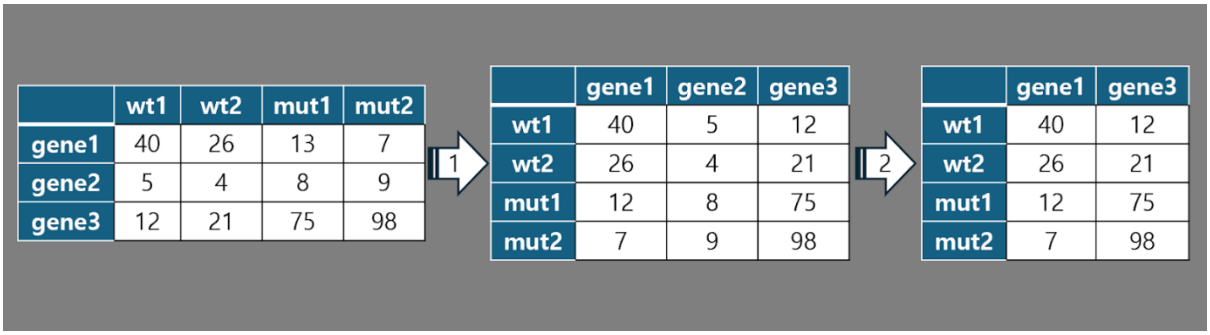


figure 14 signiture.py에서 _regularization_protocol 함수 part 2

(1) counts 데이터의 형식에 맞추기 위한 과정으로 dfs에 있던 모든 데이터 프레임을 하나의 데이터 프레임으로 만든 후 행과 열을 바꿔준다. 그 이후 모든 샘플에서 유전자의 발현량이 10보다 작은 유전자를 제거한다.(너무 적은 발현량을 가진 유전자는 실험 결과 교란의 위험이 있음, DESeq2 권장사항)

(2) metadata의 형식에 맞추기 위해 행을 각 샘플의 이름으로, 열을 위의 for 문에서 만든 리스트 2개로 구성한다.



	wt1	wt2	mut1	mut2
gene1	40	26	13	7
gene2	5	4	8	9
gene3	12	21	75	98

1

	gene1	gene2	gene3
wt1	40	5	12
wt2	26	4	21
mut1	12	8	75
mut2	7	9	98

2

	gene1	gene3
wt1	40	12
wt2	26	21
mut1	12	75
mut2	7	98

figure 15 counts_df 데이터프레임이 counts가 요구하는 형식으로 변환되는 과정 예시

1은 **counts_df = counts_df.T**, 2는 **counts_df.loc[:, ~(counts_df.lt(10).all(axis=0))]**이다. 발현량 데이터는 보통 행에 gene, 열에 샘플을 놓지만 counts 데이터프레임은 그 반대를 요구하기 때문에 필요한 과정이다.

(1) DeseqDataSet는 DESeq2의 분석의 대상이 되는 객체의 형식으로 여기에 필수적인 인자로 counts, metadata, design 이 있다. Counts와 metadata는 앞에서 만든 데이터 프레임을 넣는다. design은 분석의 설계를 표시해주는 부분으로 이 의미는 실험에 따른 차이는 무시한다는 것이며, wt/mut의 차이에 따른 효과만 보겠다는 것이다.

```
inference = DefaultInference(n_cpus=1)

dds = DeseqDataSet(
    counts=counts_df,
    metadata=meta_data,
    design="~ experiment + group",
    refit_cooks=True,
    inference=inference
)                                          (1)

dds.deseq2()  - - - - - - - - - - - - - - - (2)

ds = DeseqStats(
    dds,
    contrast=["group", "mut", "wt"],       (3)
    inference=inference
)

ds.summary()
res = ds.results_df                        (4)
return res
```

figure 16 signiture.py에서 _regularization_protocol 함수 part 3

(2) 만든 DeseqDataSet 객체를 통해 DESeq2를 실행한다.

(3) 실행한 DESeq2에 대해 통계 분석을 진행한다.

(4) 통계분석한 결과를 반환한다.

```python
def _signiture_analysis(res):
    N = 30   # CMap에 넣을 up / down 유전자 개수

    # 1) 유의한 DEG만 보고 싶으면 padj 기준 0.1 이하 필터링
    deg = res[res["padj"] < 0.1]
    print(f"유의한 DEG 수: {deg.shape[0]}")

    # 2) CMap용 Up / Down 상위 N개 선택
    up = deg[deg["log2FoldChange"] > 0].sort_values("padj", ascending=True)
    down = deg[deg["log2FoldChange"] < 0].sort_values("padj", ascending=True)

    # 원하는 갯수 미달일 경우 padj에서 상위 N개로 채우기
    if len(up) < N:
        up = res[res["log2FoldChange"] > 0].sort_values("padj", ascending=True)[:N]
    else:
        up = up[:N]  # 너무 많으면 N개로 자름

    if len(down) < N:
        down = res[res["log2FoldChange"] < 0].sort_values("padj", ascending=True)[:N]
    else:
        down = down[:N] # 너무 많으면 N개로 자름

    # pandas -> list
    up_genes = up.index.tolist()
    down_genes = down.index.tolist()

    print(f"Final Up genes: {len(up_genes)}, Down genes: {len(down_genes)}")

    return up_genes, down_genes
```

(1)

(2)

(3)

figure 17 signiture.py에서 _signiture_analysis 함수

(1) 통계 분석 결과 발현량이 유의미하게 up/down된 gene N개를 선정한다.

(2) padj 0.1기준 통계적으로 유의미한 발현량의 차이가 난다고 볼 수 있는 up/down gene이 N개가 되지 못할 경우 padj 기준을 무시하고 상위 N개를 선정한다.

(3) 데이터프레임에서 유전자 이름만 가지고 리스트를 만든다

```python
def _regularization_single_protocol(path, file_groups):
        dds = DeseqDataSet(
            counts=counts_df,
            metadata=meta_data,
            design="~ experiment + group",
            refit_cooks=True,
            inference=inference
        )

        dds.deseq2()
        ds = DeseqStats(
            dds,
            contrast=["group", "mut", "wt"],
            inference=inference
        )

        ds.summary()
        res = ds.results_df
        sig_up, sig_down = _signiture_analysis(res)
        up_gene.extend(sig_up)
        down_gene.extend(sig_down)
    return up_gene, down_gene
```

(1)

(2)

figure 18 signiture.py에서 _regularization_single_protocol 함수

(1) 만약 합치고자 하는 실험 데이터가 서로 다른 protocol을 사용한 실험인 경우, 각 데이터를 합쳐서 한번에 DESeq2로 분석할 시 통계적 오류가 날 가능성이 있음. 따라서 앞의 _regularization_ protocol과 다르게 분석 부분이 for문 안에 위치하도록 하여 각각의 파일에 대해 따로 DESeq2를 통해 분석한다

(2) 이후 통계 처리를 통해 유의미하게 up/down 된 유전자들을 up_gene, down_gene이라는 리스트에 누적한다(이때는 중복 허용).

```python
def extract_signiture(same, path):
    file_groups = _read_col_file(path)

    if same:  # 동일한 실험 프로토콜 -> 한번에 합쳐서 진행
        res = _regularization_protocol(path, file_groups)
        up_genes, down_genes = _signiture_analysis(res)
    else:     # 전혀 다른 프로토콜 -> 각각 분석 후 합침
        up_gene, down_gene = _regularization_single_protocol(path, file_groups)

        up_genes = [g for g in up_gene if up_gene.count(g) >= len(file_groups) / 2]
        down_genes = [g for g in down_gene if down_gene.count(g) >= len(file_groups) / 2]

    return up_genes, down_genes
```

(1)

(2)

figure 19 signiture.py에서 extract_signiture 함수

(1) 파일 실행시 –same으로 실행한다면 _regularization_protocol, 아니면 _regularization _single_protocol을 통해 실행된다.

(2) 이때 _regularization _single_protocol로 실행 시 누적 되어있는 유의미하게 up/down

된 유전자들에 대해 그 개수가 사용한 발현량 데이터 파일의 개수의 절반 이상인 유전자들만 up/down_genes에 남긴다

<Step2> cmap.py에서 <Step1>을 통해 얻은 Up regulated gene, Down regulated gene 리스트를 입력으로 받는다. CMap에서 사용하는 L1000_gene_info.txt.gz을 이용하여 Entrez ID로 변환하고 CMap API를 이용하여 CMap에 요청을 보낸다. 이후 CMap 분석 완료 시 결과를 다운로드한다. 해당 다운로드 파일 이름을 <Step3>으로 넘겨준다.

figure 20 cmap.py에서 _get_L1000_BING_genes 함수

(1) CMap에서 사용하는 Entrez ID를 얻기 위해 NCBI의 ftp 포로토콜을 이용하여 GSE92742_Broad_LINCS_gene_info.txt.gz 파일을 L1000_BING_genes.txt.gz으로 저장한다 [8].

(2) 앞서 (1)에서 저장한 L1000_BING_genes.txt.gz 파일을 pandas로 읽어 gene_info에 저장한다.

(3) gene_info에서 Entrez ID에 해당하는 pr_gene_id와 Gene symbol에 해당하는 pr_gene_symbol을 딕셔너리로 저장하고 다음 함수를 위해 해당 딕셔너리를 반환한다.

```python
def _find_another_symbol(gene_symbol):
    # fetch/symbol을 사용하면 해당 유전자의 모든 정보(prev, alias 포함)를 가져옵니다.
    headers = {"Accept": "application/json"}

    r = requests.get(f"https://rest.genenames.org/fetch/symbol/{gene_symbol}", headers=headers)

    alt_symbol = []
    if r.ok:
        data = r.json()
        if data['response']['numFound'] > 0:
            doc = data['response']['docs'][0]
            alt_symbol.extend(doc.get('prev_symbol', []))
            alt_symbol.extend(doc.get('alias_symbol', []))
    return alt_symbol
```

figure 21 cmap.py에서 _find_another_symbol 함수

figure 21은 실제 Entrez ID로 변환하는 과정에서 동일한 Gene이지만, 다른 이름으로 저장되어 있을 가능성이 있기에 HGNC API를 사용하여 Previous symbol과 Alias symbol을 가져온다. 해당 symbol로 다시 검색을 진행하여 유의미한 차이를 보이는 Gene이 CMap에서 사용되는 Entrez ID가 없어 활용되지 못하는 상황을 최대한 방지하고자 하였다.

```
Extracted Signature: 30 Up, 30 Down
Entrez ID 매칭 성공: Up: 15, Down: 21
UP: TMEM238 []  TMEM238 Not found
DOWN: MMP7  ['MPSL1', 'PUMP-1'] MMP7    Not found
DOWN: UCHL1 7345
DOWN: CAVIN1    ['PTRF', 'cavin-1', 'CGL4'] CAVIN1  (alt: PTRF) 284119
```

figure 22 실제 Entrez ID 매핑 결과

figure22를 보면 up gene 30개, down gene 30개로 전달을 하였지만, Entrez ID로 매핑하는 과정에서 up gene 15개, down gene 9개가 손실되었다. TMEM238의 경우 다른 symbol이 없어 아예 매핑이 되지 않았고, MMP7의 경우 다른 symbol이 있었으나 해당 symbol도 entrez_ids 딕셔너리에 없어 매핑되지 않았다. UCHL1의 경우 정상적으로 매핑되었고, CAVIN1은 처음에는 매핑에 실패하였으나, prev_symbol이 entrez_ids에 존재해 매핑에 성공한 경우이다. 이처럼 실제, up gene은 다른 symbol로도 매핑에 성공하지 못했지만, down gene의 경우 2개의 gene이 다른 symbol을 통해 매핑에 성공하였다.

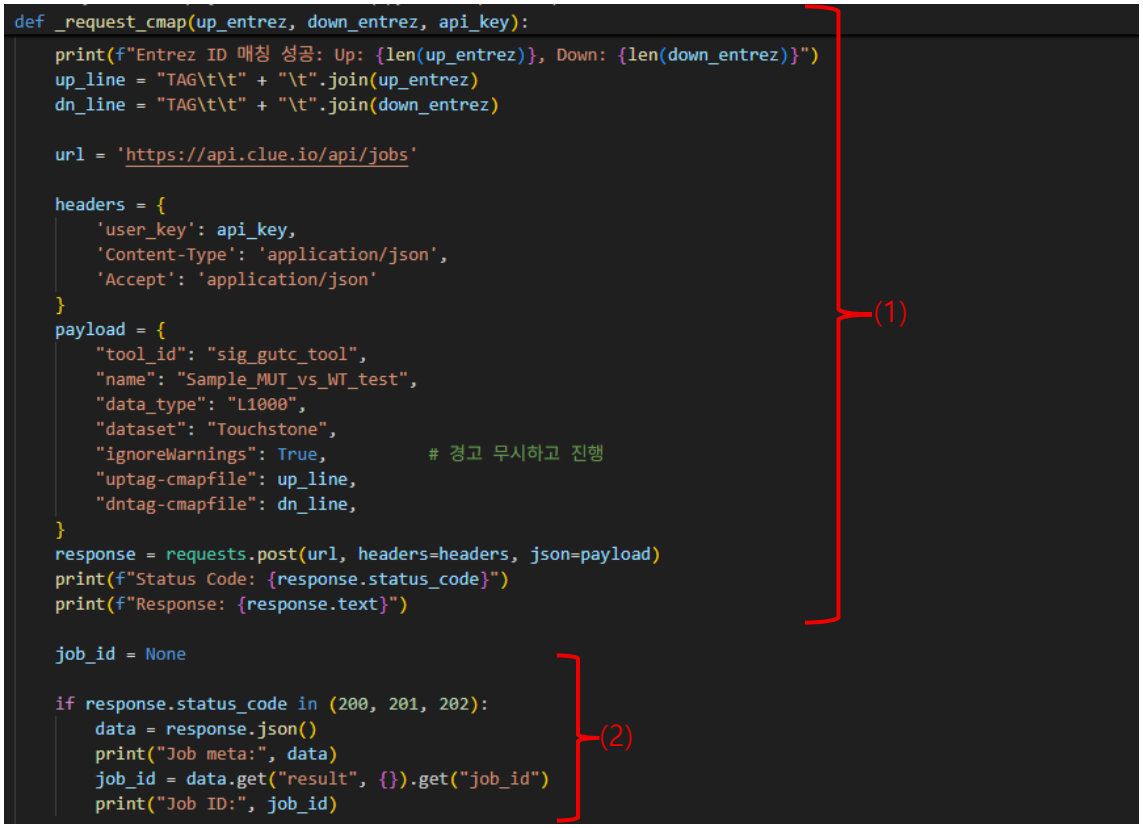


figure 23 cmap.py에서 _request_cmap 함수

(1) 앞서 Entrez ID로 매핑에 성공한 유전자를 CMap API에 맞는 형식으로 변환하여 CMap API를 이용하여 요청을 보내는 코드이다[9].

(2) (1)에서 요청을 보내고 받은 응답에서 job_id를 가져온다. Job_id는 CMap에서 얼마나 경과되었는지 알 수 있다.

```
def _status(job_id, api_key):
    status_url = f"https://api.clue.io/api/jobs/findByJobId/{job_id}"
    status_resp = requests.get(status_url, headers={"user_key": api_key})
    r = status_resp.json()
    status = r.get('status')
    print(status)
    if status == 'submitted' or status == 'pending':
        print(f"{status}... 3분 대기")
        return False
    elif status == 'completed':
        return True
```

figure 24 cmap.py에서 _status 함수

해당 URL에 job_id를 이용하여 요청을 보내면 응답에서 내가 보낸 요청이 어떤 상태에 있는지 status를 통해 알 수 있다. Pending은 분석 전에 요청이 처리되는 상태,

submitted는 요청이 처리되어 분석이 진행되는 상태이다. 분석이 완료되면 completed가 반환한다. CMap은 짧게는 15분, 길게는 1시간까지 걸리는 작업이기에 3분마다 지속적으로 작업 상태를 확인하여, 사용자가 오류가 발생한 줄 알고 작업을 중단시키는 일을 방지하기 위함이다. 작업이 완료되면 True를 반환한다.

```python
def _get_cmap_result(out_path, job_id, api_key):

    url = f"https://api.clue.io/api/jobs/findByJobId/{job_id}"
    resp = requests.get(url, headers={"user_key": api_key})
    print("Status code:", resp.status_code)
    print("Status body:", resp.text)

    standard_result = resp.json().get("download_url")
    print("raw download_url field:", standard_result)

    if standard_result:
        # 앞에 프로토콜 붙이기
        if standard_result.startswith("//"):
            download_url = "https:" + standard_result
        else:
            download_url = standard_result

        print("Download URL:", download_url)

        r = requests.get(download_url, stream=True)
        r.raise_for_status()

        out_fname = f"cmap_result_{job_id}.tar.gz"
        result_path = os.path.join(out_path, out_fname)

        with open(result_path, "wb") as f:
            for chunk in r.iter_content(chunk_size=8192):
                if chunk:
                    f.write(chunk)
        print("CMap result downloaded:", out_path)
    else:
        print("Download_url 필드가 없음")
    return out_fname
```

(1)

(2)

figure 25 cmap.py에서 _get_cmap_result 함수

(1) _status에서 True값이 반환되면 응답 결과에서 Download 링크를 추출한다.

(2) (1)에서 다운로드 링크가 존재한다면, 앞에 'https:'프로토콜을 붙여서 다운로드 링크를 완성하고 requests를 이용하여 해당 링크에서 결과 파일을 다운받는다. 결과 파일의 이름을 cmap_result_{job_id}.tar.gz으로 저장한다. 파일 이름에 job_id를 포함하여 사용자가 분석을 여러 번 진행할 때 결과 파일이 겹치지 않게 하기 위함이다.

```
raw download_url field: //s3.amazonaws.com/data.clue.io/api/this0655@dgu.ac.kr/results/Dec_15_2025/my_analysis.sig_gutc_tool.693f90769888b700137250ae/my_analysis.sig_gutc_tool.693f90769888b700137250ae.tar.gz (1)
Download URL: https://s3.amazonaws.com/data.clue.io/api/this0655@dgu.ac.kr/results/Dec_15_2025/my_analysis.sig_gutc_tool.693f90769888b700137250ae/my_analysis.sig_gutc_tool.693f90769888b700137250ae.tar.gz (2)
CMap result downloaded: C:\Users\this0\pyprogramming\biodatabase\Project_BioDB\data (3)
```

figure 26 실제 다운로드 링크와 저장된 파일 주소

실제 결과를 보면 앞선 코드에서 얻은 다운로드 경로가 (1)인 것을 볼 수 있고, https:를 붙여 완성한 다운로드 링크가 (2)이며, 다운로드 받은 파일의 경로가 (3)에 나타나 있다. 앞서 <Step0>에서 만든 ./data/폴더에 생성된 것을 볼 수 있다.

<Step3> analysis.py에서 <Step2>에서 다운받은 결과파일을 기반으로 후보 물질을 추출한다. 이후 해당 물질의 ID와 TAG score, 이름을 딕셔너리로 만들어 <Step4>에 전달한다.

```python
def get_drug_list(work_path, result_name):
    # 1. 결과 압축 해제 및 첫 디렉토리 경로 얻기
    first_path = _decompression_cmap_results(work_path, result_name)  (1)

    # 2. pert_id_summary.gct 파일 읽기
    pert_summary = _get_pert_summary(work_path, first_path)  (2)

    # 3. cs_*.gct 파일 읽기 (동적 탐색)
    csn = _get_csn(work_path, first_path)  (3)

    df1 = pert_summary['df']
    df1 = df1[df1.index.str.startswith('BRD')]
    # TAG 점수가 낮은 순서대로 상위 10개 추출
    df1 = df1.sort_values('TAG', ascending=True).head(10)
    drug_list = df1.index.to_list()  (4)

    df2 = csn['row_metadata']
    drug_dict = {}
    for brd in drug_list:
        row = df2['pert_iname'].loc[df2['pert_id'] == brd]
        tag_score = df1.loc[brd, 'TAG']
        drug_dict[brd] = {'name': row.values[0], 'tag': float(tag_score)}
    return drug_dict  (5)
```

figure 27 analysis.py에서 get_drug_list 함수

(1) <Step2>에서 얻은 결과 파일을 압축해제한다.

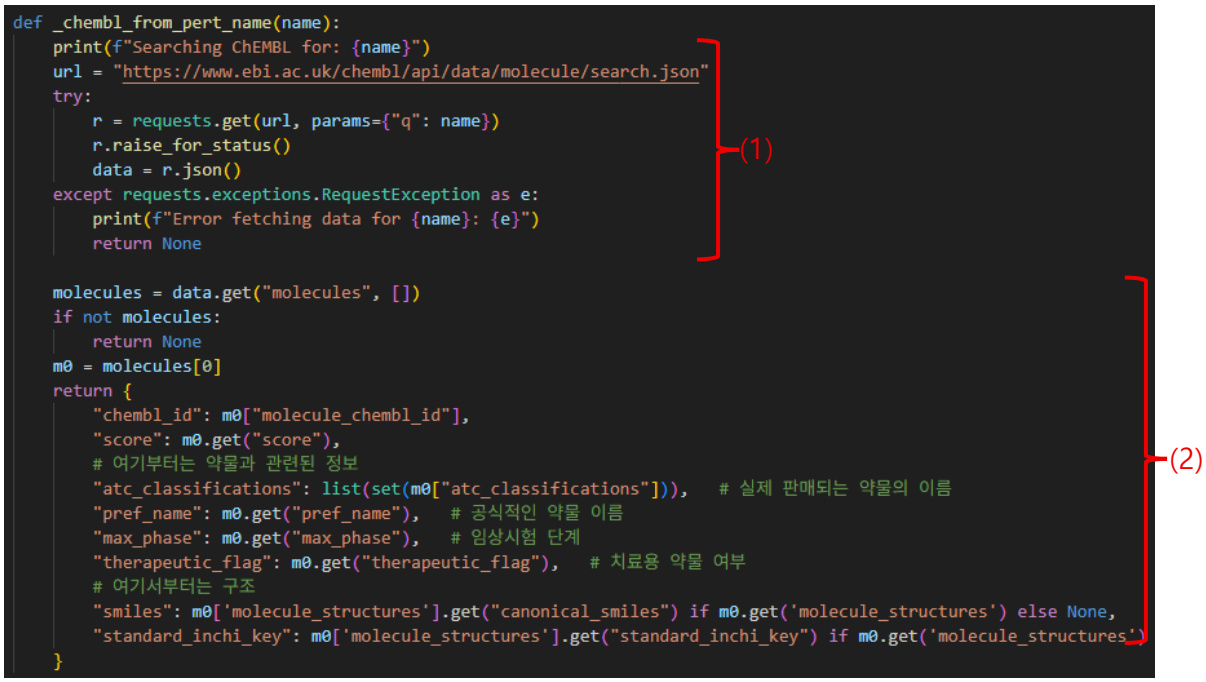
(2) (1)에서 압축 해제한 파일에서 ./arfs/TAG/pert_id_summary.gct 파일을 읽어서 pert_summary에 저장한다.

(3) (1)에서 압축 해제한 파일에서 ./cs_n1x476254.gct 파일을 읽어서 csn에 저장한다.

(4) pert_summary에서 ['df']를 df1으로 저장하고, df1에서 BRD로 시작하는 물질 중에 TAG값을 기준으로 오름차순으로 정렬한다. 이후 상위 10개 물질을 선정한다.

(5) (4)에서 얻은 물질은 pert_id로 존재한다. 이에 csn에서 각 pert_id에 해당하는 pert_iname을 추출하여, TAG값과 함께 딕셔너리로 저장하여 <Step4>로 제공한다.

<Step 4> Recommendation.py에서 <Step 3>에 얻은 물질들을 바탕으로, ChEMBL에서 약물 관련 정보를 가져오고, 해당 물질의 smiles를 이용하여 구조상 비슷한 물질들 중 임상 단계가 있는 물질을 정리하여 보고서를 생성한다.




```python
def _chembl_from_pert_name(name):
    print(f"Searching ChEMBL for: {name}")
    url = "https://www.ebi.ac.uk/chembl/api/data/molecule/search.json"
    try:
        r = requests.get(url, params={"q": name})
        r.raise_for_status()
        data = r.json()
    except requests.exceptions.RequestException as e:
        print(f"Error fetching data for {name}: {e}")
        return None

    molecules = data.get("molecules", [])
    if not molecules:
        return None
    m0 = molecules[0]
    return {
        "chembl_id": m0["molecule_chembl_id"],
        "score": m0.get("score"),
        # 여기부터는 약물과 관련된 정보
        "atc_classifications": list(set(m0["atc_classifications"])),   # 실제 판매되는 약물의 이름
        "pref_name": m0.get("pref_name"),   # 공식적인 약물 이름
        "max_phase": m0.get("max_phase"),   # 임상시험 단계
        "therapeutic_flag": m0.get("therapeutic_flag"),   # 치료용 약물 여부
        # 여기서부터는 구조
        "smiles": m0['molecule_structures'].get("canonical_smiles") if m0.get('molecule_structures') else None,
        "standard_inchi_key": m0['molecule_structures'].get("standard_inchi_key") if m0.get('molecule_structures')
    }
```

figure 28 recommendation.py에서 _chembl_from_pert_name 함수

(1) <Step3>에서 얻은 물질들에서 ChEMBL API를 이용하여 검색을 진행한다. try: ~ except:를 이용하여 검색이 되지 않는 경우에 에러를 발생시키지 않고 결과 없음으로 설정한다.

(2) (1)의 요청에서 얻은 결과 데이터에서 ①~⑧까지의 정보를 가져온다.

① chembl_id: 해당 물질에 대한 공식적인 ChEMBL ID를 가져온다.

② score: 여러 결과 중 가장 관련성이 높은 결과임을 보여주는 지표이다.

③ atc_classification: 실제 판매 중이라면 해당 약물의 이름을 가져온다.

④ pref_name: 공식적인 약물 이름을 가져온다.

⑤ max_phase: 해당 물질의 최대 임상 단계를 가져온다.

⑥ therapeutic_flag: 임상을 통과한 물질 중, 치료에 사용되는 약물인지를 가져온다.

⑦ smiles: 해당 물질의 구조인, smiles 코드를 가져온다.

⑧ standard_inchi_key: 물질마다 고유하게 가지는 구조 관련 코드를 가져온다.

```python
def _similar_from_smile(smile):
    print(f"Searching similar molecules for SMILES: {smile}")
    score = 60
    ls_num = 20

    # SMILES에 특수문자(# 등)가 포함될 수 있으므로 URL 인코딩 필요
    encoded_smile = quote(smile)
    url  = f"https://www.ebi.ac.uk/chembl/api/data/similarity/{encoded_smile}/{score}?format=json&limit={ls_num}"

    try:
        r = requests.get(url)
        r.raise_for_status()
        data = r.json()
        return [_parsing_to_json(item) for item in data['molecules']]
    except requests.exceptions.RequestException as e:
        print(f"Error fetching similar molecules for SMILES {smile}: {e}")
        return None
```

(1)

(2)

figure 29 recommendation.py에서 _similar_from_smile 함수

(1) _chembl_from_pert_name에서 얻은 smiles를 입력으로 받아 ChEMBL API를 이용하여 유사한 물질을 검색한다. Score 60점 이상인 물질만을 가져오며, ls_num = 20으로 최대 20개의 물질을 가져온다.

(2) (1)에서 나온 결과들은 최대 20개의 물질이 있으므로, 리스트 컴프리핸션을 이용하여 각 물질들에 대한 정보를 _parsing_to_json을 통해 가져온다. 가져오는 정보는 _chembl_from_pert_name과 동일하게 8가지를 가져오며, 그 중 score대신 원 물질의 smiles과 얼마나 유사한지를 나타내는 similarity를 저장한다.

(1), (2)에서 얻은 내용을 기반으로 하여, Recommendation_{job_id}.txt에 결과 파일을 작성한다.

```
<Abstract>
1. BRD-K88761633(CHEMBL4303583) [TAG: -94.51]

2. XMD-885(CHEMBL5303523) [TAG: -90.88]

3. metrizamide(CHEMBL462394) [TAG: -90.14]

4. CGP-52411(CHEMBL268868) [TAG: -89.16]

5. 17-beta-estradiol(CHEMBL1200430) [TAG: -87.41]
├─ESTRADIOL BENZOATE(2.0)
└─ESTRAMUSTINE(3.0)

6. anisomycin(CHEMBL423192) [TAG: -87.40]

7. vinorelbine(CHEMBL553025) [TAG: -85.57]
└─VINORELBINE TARTRATE(4.0)

8. TG-101348(CHEMBL4465517) [TAG: -82.73]
├─FEDRATINIB HYDROCHLORIDE(4.0)
└─FEDRATINIB(4.0)

9. cholic-acid(CHEMBL205596) [TAG: -81.54]
└─GLYCOCHOLIC ACID(3.0)

10. PAC-1(CHEMBL3707398) [TAG: -80.92]
```

```
<Detailed Information>
===후보 물질 1: BRD-K88761633===
TAG Score: -94.51
ChEMBL Accession ID: CHEMBL4303583
SMILES: Cc1ccc(Cl)cc1N(CC(=O)Nc1c(F)cccc1F)S(C)(=O)=O
Standard InChI Key: WAOBCCBUTHNTFO-UHFFFAOYSA-N
임상 시험 정보가 없습니다.

===후보 물질 5: 17-beta-estradiol===
TAG Score: -87.41
ChEMBL Accession ID: CHEMBL1200430
SMILES: CC(=O)Oc1ccc2c(c1)CC[C@@H]1[C@@H]2CC[C@]2(C)[C@@H](O)CC[C@@H]12
Standard InChI Key: FHXBMXJMKMWVRG-SLHNCBLASA-N
약물 정보
- 공식 약물 이름: ESTRADIOL ACETATE
- 최대 임상시험 단계: 4.0
- 실제 치료에 사용되고 있습니다.

    후보 물질 BRD-K66766661과 유사물질 1번
    후보 물질과의 유사도 점수: 76.8%
    ChEMBL Accession ID: CHEMBL282575
    SMILES: C[C@]12CC[C@@H]3c4ccc(OC(=O)c5ccccc5)cc4CC[C@H]3[C@@H]1CC[C@@H]2O
    Standard InChI Key: UYIFTLBWAOGQBI-BZDYCCQFSA-N
    약물 정보
    - 공식 약물 이름: ESTRADIOL BENZOATE
    - 최대 임상시험 단계: 2.0
    - 실제 치료에 사용되고 있습니다.

    후보 물질 BRD-K66766661과 유사물질 2번
    후보 물질과의 유사도 점수: 72.9%
    ChEMBL Accession ID: CHEMBL1575
    SMILES: C[C@]12CC[C@@H]3c4ccc(OC(=O)N(CCCl)CCCl)cc4CC[C@H]3[C@@H]1CC[C@@H]2O
    Standard InChI Key: FRPJXPJMRWBBIH-RBRWEJTLSA-N
    약물 정보
    - 공식 약물 이름: ESTRAMUSTINE
    - 실제 판매되는 이름: L01XX11
    - 최대 임상시험 단계: 3.0
    - 아직 임상단계로 상용화되지 않았습니다.
```

figure 30 Recommendation_{job_id}.txt

figure 30은 임의의 GSE에 대하여 진행한 실험 결과이다. <Step3>에서 나온, CMap결과는 임상여부에 상관없이 정보를 제공해준다. 후보 물질 1의 경우 TAG score는 높으나, 임상 단계 정보가 없는 것을 알 수 있다. 하지만, 후보물질 5의 경우 TAG score는 상대적으로 낮으나, 최고 임상 단계가 4이며, 실제 치료에 사용되고 있고, 그와 유사한 물질 모두 임상단계가 존재하여 표시되고 있는 것을 볼 수 있다.

Result

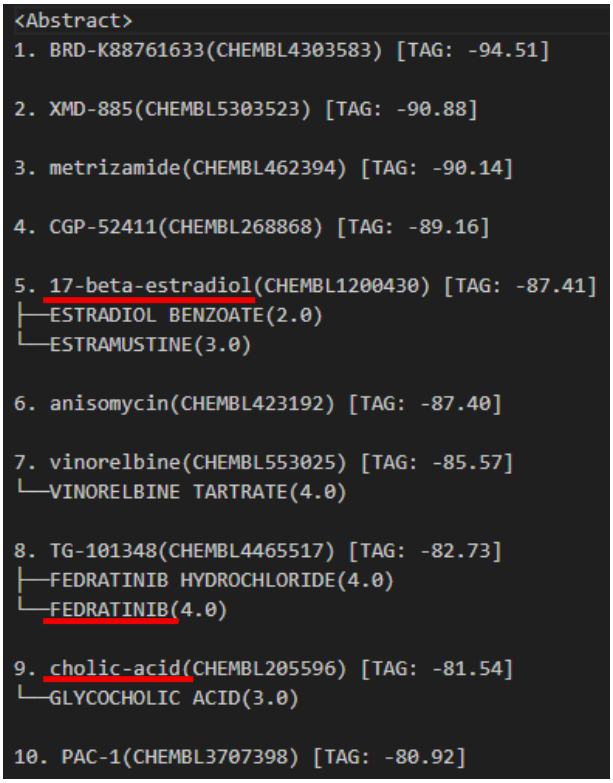
실제 파이프라인에 따라서 파일을 실행하고, 그 결과를 분석해보고자 한다.

사용한 유전자 발현량 파일은 GEO 데이터베이스에서 가져온 파킨슨병에서 LRRK2-G2019S 유전자의 돌연변이를 보유한 4명의 건강 환자와 4명의 PD(Parkinson's Disease) 환자의 iPSC 유래 microglia에서 추출한 대량 RNAseq이다. 해당 유전자 발현량 데이터를 제공한 논문은 LRRK2-G2019S 유전자에서 돌연변이가 발생하였을 때, microglia의 대사와 염증 상태가 어떻게 달라지는지를 보고자 하였다. 실제 결과는 G2019S microglia에서 정상 대비 염증성 활성(TNF-a)이 증가했고 동시에 대사가 해당과정(Glycolysis)으로 기울면서 특정 대사경로 이상이 동반되었다. 하여 논문에서는 대사축을 겨냥한 oxamic acid처리가 microglia의 대사/염증 지표를 낮춘다는 점을 제시하였다[10].

오른쪽 사진은 figure 30에서 abstract만 가져온 것이다. 해당 사진에는 CMap결과로 나온 10가지 후보 물질과 유사한 물질이 소개되어 있다.

앞서 본 조의 목적은 유전자 발현량을 기반으로 하여 치료 가능성을 가진 물질 후보들을 제시하는 것이었다. 이에 실제 PD 환자의 microglia RNAseq데이터를 기반으로 총 16개의 물질을 추천해주었고, 실제 논문에서는 oxamic acid(NH2-CO-COOH)가 가장 좋은 효과를 보인 물질이었다. 하지만 이 물질이 결과로 포함되지 않은 이유는 발현/억제시키는 유전자의 종류만 반영하고 구체적인 발현량에 대한 정보는 적용하지 않아서 나온 결과로 추측한다.

그러나 CMap의 결과로 나온 10가지의 후보 물질과 유사한 물질에 대해 조사해본 결과,



후보 5번에 해당하는 17β-Estradiol은 다른 논문에서 microglia의 염증 활성을 낮추고, 항염/회복을 올리는 조절인자로 사용될 수 있음을 보였고[11], 후보 8번의 유사한 물질인 Fedratinib은 원래는 골수증식성 종양에 사용되는 JAK2 억제제이다. JAK2를 억제하면 관련된 STAT3, 5등이 억제되면서 염증성 사이토카인이 억제될 가능성을 제시해볼 수 있다[12]. 마지막으로, Cholic acid는 해당 물질에서 유도된 INT-777이 TGR5를 활성화시켜 microglia에서 TNF-a가 억제되는 효과를 보였다[13]. 이는 문제가 되는 세포의 분자적 기전에 영향을 미치는 약물을 제시하는 것으로 볼 수 있다. 이런 점으로 보아 구체적인 발현량 데이터 없이도 특정한 돌연변이 혹은 질병 상태의 세포에 대해 교란된 유전자 발현을 복구할 가능성이 있는 물질들을 제시할 가능성이 있다고 볼 수 있다.

또한, INT-777은 아직 임상이 진행되지 않아, 결과를 제시하는 과정에서 빠진 것으로 보인다. 이를 제외하고 보면 총 16개 중에 2개가 항염과 비슷한 기능을 보이는 물질인 것으로 파악된다.

Discussion

본 pipeline에서는 프로토콜만 동일하다면 여러 실험에서 얻은 발현량 데이터를 합쳐서 DESeq2를 통해 분석 할 수 있도록 하고 있다. 그러나 이는 생물학적 의미를 반영하여 분석하기 보다는 입력한 데이터에 대해 단순히 통계적 처리를 진행하는 것에 불과하다. 때문에 서로 다른 유전자 명명법을 사용하는 경우 생물학적으로 동일한 유전자를 사용하는 것이라도 독립적인 항목으로 처리될 가능성이 존재한다.

이 문제점을 해결하는 방법으로 각 count matrix에 명시된 유전자의 이름을 gene annotation API 등을 사용하여 명명법을 통일하는 방식으로 해결할 수 있을 것이다. 하지만 발현 데이터는 포함하고 있는 유전자의 개수가 매우 많아 분석 시간이 크게 증가하게 되어 본 파이프라인에서는 포함하지 않았다.

앞서 CMap API를 이용하여 쿼리를 보내는 과정에서 Gene symbol을 반드시 Entrez ID로 변환해주어야 한다. 이 과정에서 Gene symbol이 Entrez ID로 매핑되지 않아 손실되는 일이 발생하였다. 이는 CMap의 근본적인 한계이다. CMap은 BING(Best Inferred Genes)이라는 유전자 공간 안에서 Connectivity score를 계산한다. 이때 계산에 사용되는 gene은 총 10174개이다[14]. Figure 31에서 볼 수 있듯이 실제 cmap.py에서 _get_L1000_BING_genes 함수를 통해 파일을 읽어 BING 유전자에 해당하는 열의 개수를 세어보면 총 10174개이다. 현재 GENCODE에서 제시한 protein-coding gene

figure 31 entrez_ids를 파일로 저장한 결과

수는 19433개이다[15]. 약 9000개 가량의 유전자는 계산에 활용되지 않는다. 물론 CMap도 통계적으로 중요한 유전자를 선별하였겠지만, 설령 중요하지 않은 유전자에서 유의미한 시그널이 나오게 된다면 이를 놓치게 된다는 단점이 존재한다.

또한, CMap은 유전자 발현량을 직접 비교하여 물질을 추천해주지 않는다. 만약 실험자가 특정 유전자의 silence 등의 목적을 가지고 해당 파이프라인을 돌린다면 이는 부적절할 수 있다. 이런 점을 인지한 후에, 실제 CMap에서 사용하는 유전자 발현 원 데이터셋을 이용하여 특정 물질을 처리하였을 때, 특정 유전자의 발현량이 어땠는가를 볼 수 있게 코드를 구현하였다[8].

```python
from cmapPy.pandasGEXpress.parse import parse

fname = 'level3_beta_trt_cp_n1805898x12328.gctx'

try:
    meta_col_obj = parse(target, col_meta_only=True)
    all_cids = meta_col_obj.index.to_list()
    print(f"총 컬럼 개수: {len(all_cids)}")

    search_col = "ABY001"
    target_cids = [cid for cid in all_cids if search_col in cid]
    print(f"'{search_col}'을 포함하는 컬럼 개수: {len(target_cids)}")

    target_gene_id = "672"

    if target_cids:
        gct_data = parse(
            target,
            cid=target_cids[:10],
            rid=[target_gene_id]
        )
        print("\n데이터 로딩 성공:")
    else:
        print("\n[경고] 조건에 맞는 컬럼이 없어 데이터를 로드하지 않았습니다.")
except Exception as e:
    print(f"데이터 로딩 오류: {e}")
```

rid	672
cid	
ABY001_A375_XH_X1_B15:A13	6.570550
ABY001_A375_XH_X1_B15:A14	7.920500
ABY001_A375_XH_X1_B15:A15	5.053700
ABY001_A375_XH_X1_B15:A16	6.581700
ABY001_A375_XH_X1_B15:A17	6.845201
ABY001_A375_XH_X1_B15:A18	6.461550
ABY001_A375_XH_X1_B15:A19	5.990100
ABY001_A375_XH_X1_B15:A20	5.957550
ABY001_A375_XH_X1_B15:A21	6.090800
ABY001_A375_XH_X1_B15:A22	5.995100

```
rid
672    6.346675
dtype: float32
```

figure 32 원 발현량 데이터를 읽어 특정 유전자의 발현량을 보는 코드 및 그 결과

Figure 32를 보면 ABY001이라는 물질을 처리하였을 때, Entrez ID가 672인 유전자, BRCA1의 발현량을 보기 위한 코드이다. 실제, 발현량이 약 6정도로 나오는 것을 볼 수 있다. 이처럼 실제 CMap결과에서 유전자 발현량을 기준으로 2차 필터링을 진행하자는 의견이 있었으나, 결과 물질을 어떤 유전자를 기준으로 필터링 할 것인지, 그 차이는 얼마로 기준을 세울 것인지 등의 모호함이 있어 파이프라인에서는 제외하였다. 하지만, 추후 사용자가 특정 물질을 처리하였을 때, 특정 유전자의 발현량을 알고 싶다면 그 정보를 제공하는 쪽으로 발전 가능성이 있다고 생각한다.

Reference

[1] F. Lopez, "Gene Expression: The Basics of Transcription and Translation," Journal of Biology and Today's World, vol. 12, no. 1, 2023, doi: 10.35248/2322-3308-12.1.008.

[2] European Medicines Agency, "Azacitidine Kabi," EMA (European Medicines Agency), [Online]. Available: https://www.ema.europa.eu/en/medicines/human/EPAR/azacitidine-kabi. [Accessed: Dec. 15, 2025].

[3] Liu C, Arnold R, Henriques G and Djabali K, "Inhibition of JAK-STAT Signaling with Baricitinib Reduces Inflammation and Improves Cellular Homeostasis in Progeria Cells" Cells, vol. 8, no 10 2019, doi: 10.3390/cells8101276.

[4] CLUE Team, "About CLUE," CLUE, Broad Institute Connectivity Map, [Online]. Available: https://clue.io/about. [Accessed: Dec. 17, 2025].

[5] CLUE Team, "Connectivity scores," CLUE Connectopedia, Broad Institute Connectivity Map, [Online]. Available: https://clue.io/connectopedia/connectivity_scores. [Accessed: Dec. 17, 2025].

[6] European Bioinformatics Institute, "ChEMBL," EMBL-EBI, [Online]. Available: https://www.ebi.ac.uk/chembl/. [Accessed: Dec. 17, 2025].

[7] D. Weininger, "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules," J. Chem. Inf. Comput. Sci., vol. 28, no. 1, pp. 31–36, 1988.

[8] CLUE Team, "Guide to GEO L1000 data," CLUE Connectopedia, Broad Institute Connectivity Map, [Online]. Available: https://clue.io/connectopedia/guide_to_geo_l1000_data. [Accessed: Dec. 17, 2025].

[9] CLUE Team, "Query API tutorial," CLUE Connectopedia, Broad Institute Connectivity Map, [Online]. Available: https://clue.io/connectopedia/query_api_tutorial. [Accessed: Dec. 16, 2025].

[10] H. Kurniawan *et al.*, "The Parkinson's disease-associated LRRK2-G2019S variant restricts serine metabolism, leading to microglial inflammation and dopaminergic neuron degeneration," *Journal of Neuroinflammation*, vol. 22, Art. no. 244, 2025, doi: 10.1186/s12974-025-03577-2.

[11] R. Thakkar, R. Wang, J. Wang, R. K. Vadlamudi, and D. W. Brann, "17β-Estradiol regulates microglia activation and polarization in the hippocampus following global

cerebral ischemia," Oxidative Medicine and Cellular Longevity, vol. 2018, Art. no. 4248526, 2018, doi: 10.1155/2018/4248526.

[12] A. Jarneborn, P. K. Kopparapu, and T. Jin, "The Dual-Edged Sword: Risks and Benefits of JAK Inhibitors in Infections," Pathogens, vol. 14, art. no. 324, 2025, doi: 10.3390/pathogens14040324.

[13] R. Huang et al., "TGR5 agonist INT-777 alleviates inflammatory neurodegeneration in Parkinson's disease mouse model by modulating mitochondrial dynamics in microglia," Neuroscience, vol. 490, pp. 100–119, 2022, doi: 10.1016/j.neuroscience.2022.02.028.

[14] CLUE Team, "L1000 Gene Space," CLUE Connectopedia, Broad Institute Connectivity Map, [Online]. Available: https://clue.io/connectopedia/l1000_gene_space. [Accessed: Dec. 17, 2025].

[15] GENCODE Project, "Human gene statistics," GENCODE, [Online]. Available: https://www.gencodegenes.org/human/stats.html. [Accessed: Dec. 18, 2025].