

EX1

```
In [27]: print("Overlap:", set(alz_ids) & set(cancer_ids))  
Overlap: {'36321615', '36321363'}
```

A total of 2000 papers were extracted, of which two were duplicates. The speculation is that the reason is that the research areas overlap.

There are some missing texts, probably because the text is not in English

```
dictionary = {}  
for i in alz_ids:  
    title, abstract = get_info(i)  
    dictionary[i] = {'ArticleTitle': title, 'ArticleAbstract': abstract, 'query': 'Alzheimer'}  
for i in cancer_ids:  
    title, abstract = get_info(i)  
    dictionary[i] = {'ArticleTitle': title, 'ArticleAbstract': abstract, 'query': 'Cancer'}
```

I used a dictionary format, using each PMID to correspond to an article, easy to sort.

See the code for details.

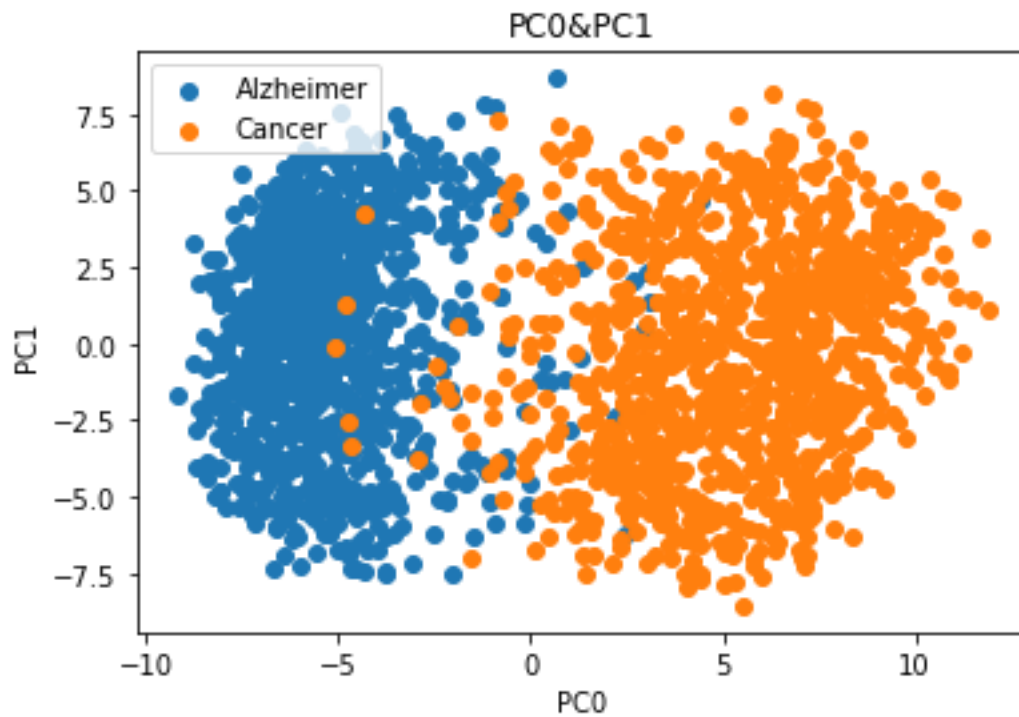
## EX2

Apply principal component analysis (PCA) to identify the first three principal components.

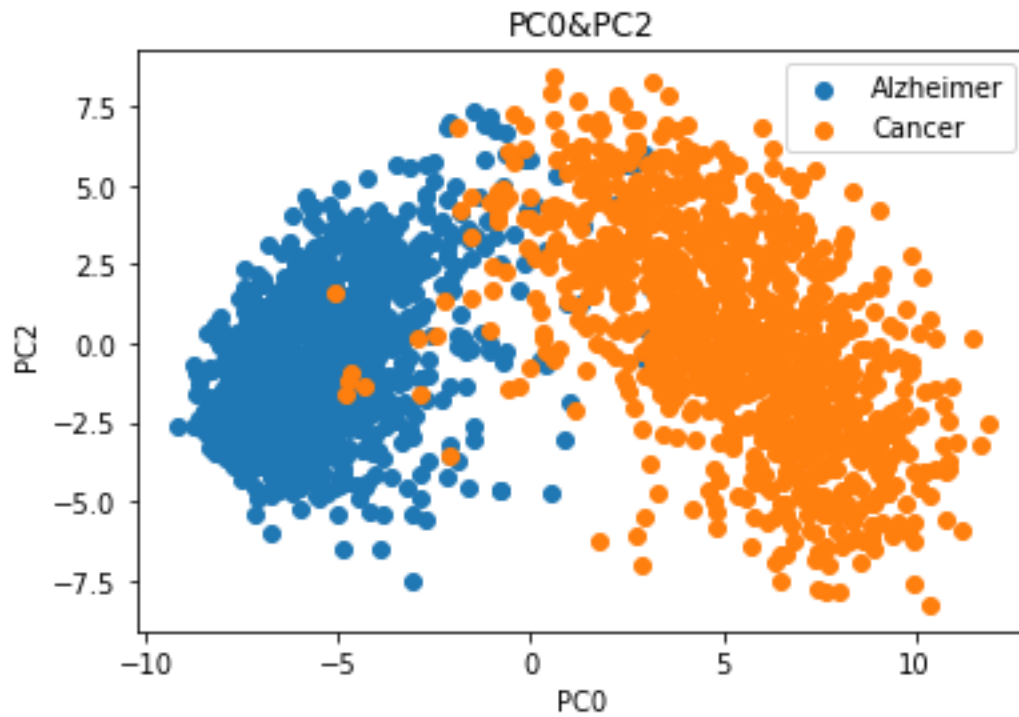
	PC0	PC1	PC2	query
0	-1.040141	-5.184434	-0.222884	Alzheimer
1	-5.402351	4.374268	0.928445	Alzheimer
2	-5.520820	4.610436	-1.998771	Alzheimer
3	-5.684639	-4.256922	-0.069922	Alzheimer
4	-5.909279	-2.143645	0.723237	Alzheimer
...	...	...	...	...
1993	1.974529	-4.315622	5.541237	Cancer
1994	5.472513	6.085035	3.709071	Cancer
1995	3.539508	-4.110456	2.859843	Cancer
1996	2.834004	-2.746227	4.597492	Cancer
1997	7.089354	-7.298180	2.489772	Cancer

1998 rows × 4 columns

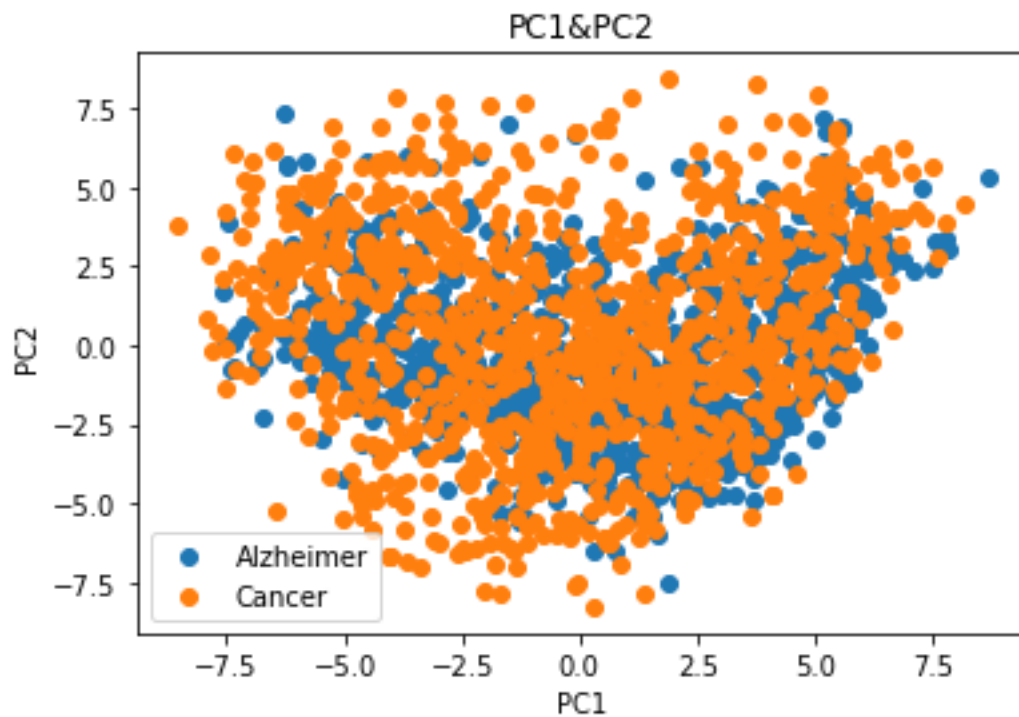
PC0 and PC1 are well separated and have close distribution directions, so there may be a co-relation



PC0 and PC2 are well separated, and the distribution direction is nearly vertical, with a possible linear relationship



PC0 and PC2 are poorly separated and have nearly identical distributions and can be considered as co-occurring variables

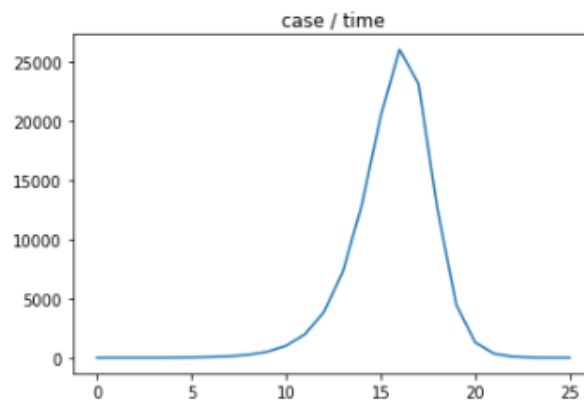


### EX3

```
In [86]: N = 134000
s, i, r, t = solve(N - 1, 1, 0, 2, 1, 30)
infected_drop = 0
while (i[infected_drop] >= 1):
    infected_drop += 1
print("The number of infected individuals drop below 1 at day:", infected_drop)
plt.plot(t[:infected_drop], i[:infected_drop])
plt.title("case / time")
```

The number of infected individuals drop below 1 at day: 26

Out[86]: Text(0.5, 1.0, 'case / time')



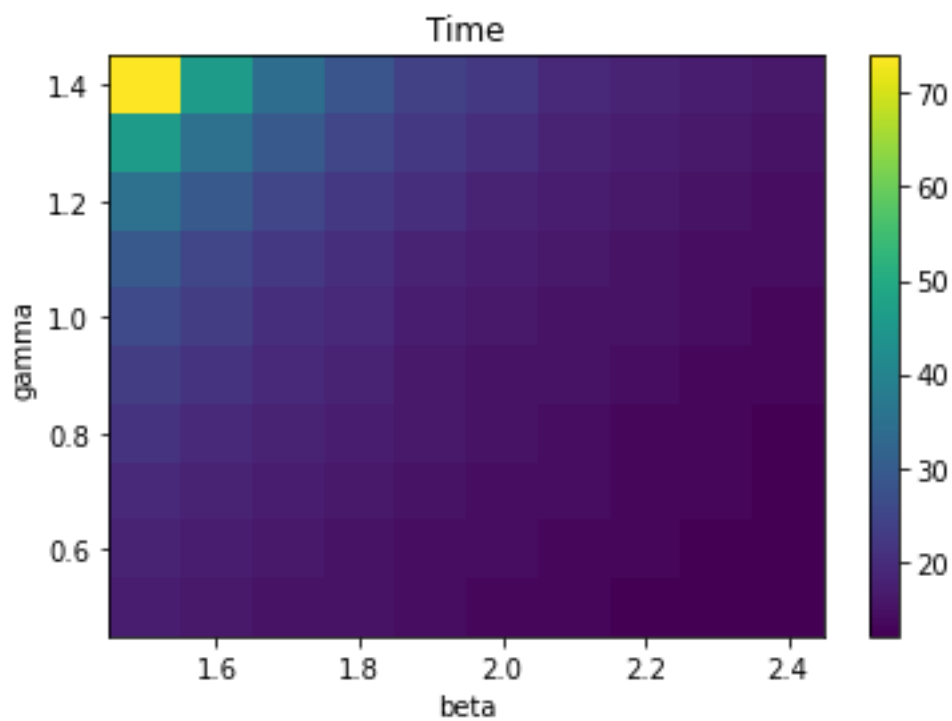
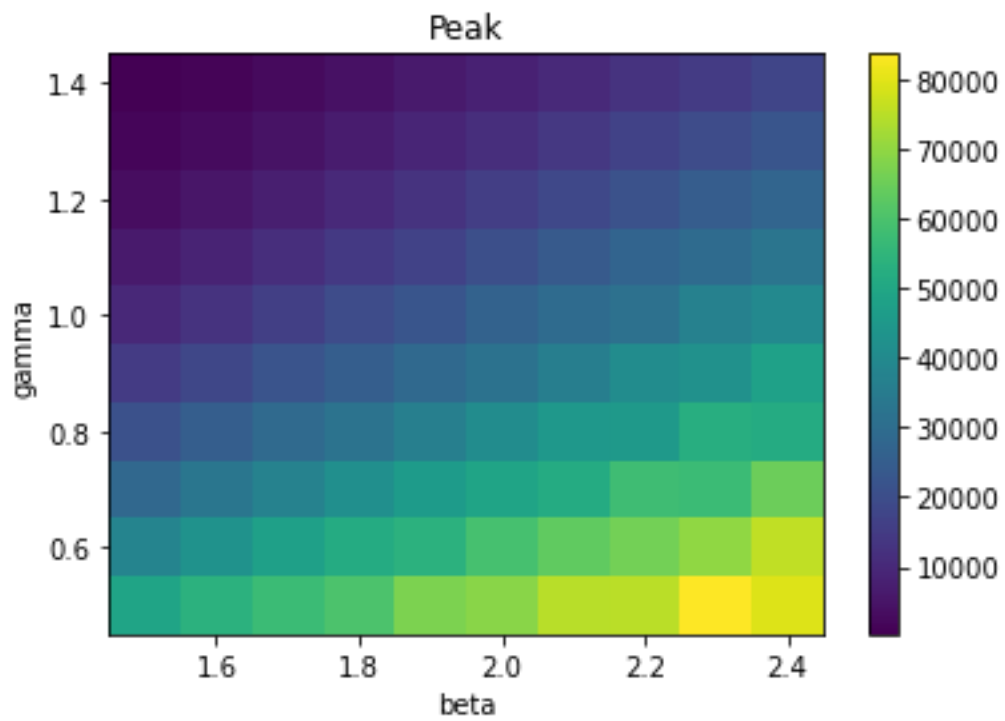
The number of infected individuals drop below 1 at day: 26

```
In [88]: N = 134000
s, i, r, t = solve(N - 1, 1, 0, 2, 1, 30)
day = 1
while (1):
    if (i[day] > i[day - 1]) and (i[day] > i[day + 1]):
        break
    day += 1
print ("Peak is day:", day, "; Infected cases is:", i[day])
```

Peak is day: 16 ; Infected cases is: 26033.391521237274

Peak is day: 16

Infected cases are: 26033



The heatmaps if beta or gamma are not confirmed

## EX4

<https://www.kaggle.com/datasets/nareshbhat/health-care-data-set-on-heart-attack-possibility>

I found a dataset on Kaggle about heart disease and causative factors. This dataset is a statistic on the likelihood of developing heart disease. There are 14 key variables, including age, sex, blood pressure, maximum heart rate, blood glucose, resting ECG results, and serum cholesterol. Key variables were not explicitly provided, but estimates could be derived from regression analysis. There were no derived variables, and each variable was indispensable. By simple observation only, there should be no variables that can be predicted by other variables.

The data format is CSV. there are 303 samples in the dataset. Including the header row there are 304 rows.

This dataset is open for learning purposes and does not require any additional permissions to gain full access.

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
10	54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
11	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
12	49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
13	64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
14	58	0	3	150	283	1	0	162	0	1.0	2	0	2	1
15	50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
16	58	0	2	120	340	0	1	172	0	0.0	2	0	2	1
17	66	0	3	150	226	0	1	114	0	2.6	0	0	2	1

A preliminary analysis of the data reveals that the 14 variables have 13 independent variables and 1 dependent variable. 13 independent variables have categorical, dummy, discrete and continuous variables. The independent variables were divided into two main categories: demographic data of the sample and medical indicators.

```
In [20]: from string import ascii_letters
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import palettable
```

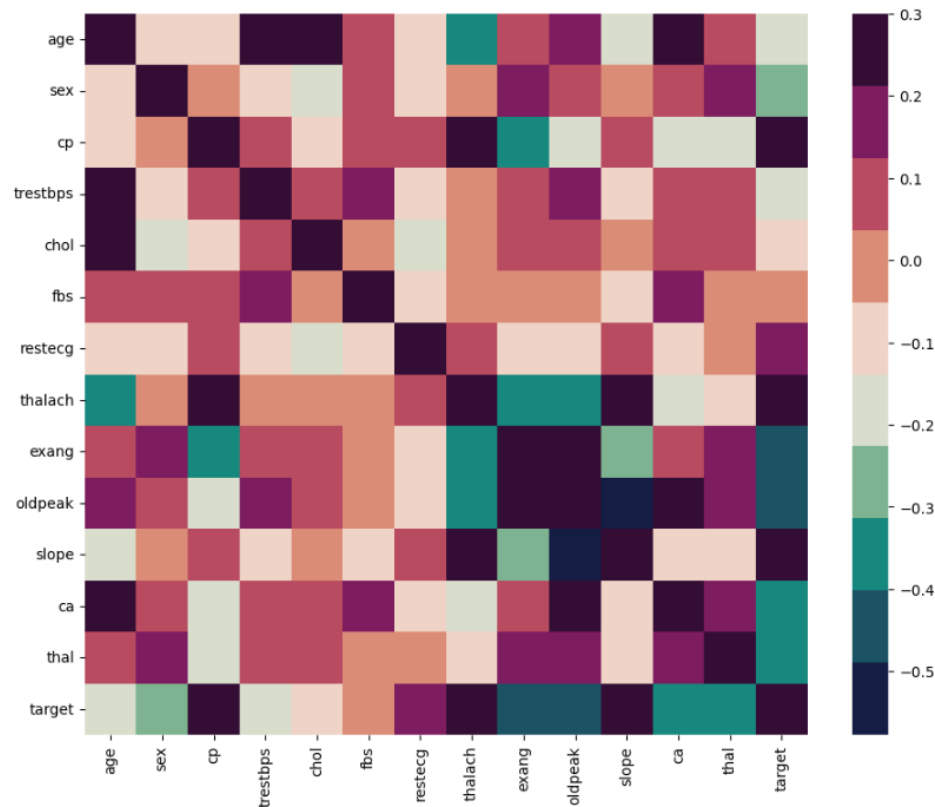
```
In [21]: dcorr = df.corr(method='pearson')
```

```
In [22]: dcorr.shape
```

```
Out[22]: (14, 14)
```

```
In [25]: plt.figure(figsize=(11, 9), dpi=100)
sns.heatmap(data=dcorr,
            vmax=0.3,
            cmap=palettable.cmocean.diverging.Curl_10.mpl_colors,
            )
```

```
Out[25]: <AxesSubplot:>
```



I performed a Pearson regression analysis on 14 variables and could find that the darker colored areas in the heat map are the more strongly correlated variables.

```
In [4]: df.isnull().sum()
```

```
Out[4]: age      0
sex        0
cp         0
trestbps   0
chol       0
fbs        0
restecg    0
thalach    0
exang      0
oldpeak    0
slope      0
ca         0
thal       0
target     0
dtype: int64
```

```
In [5]: df.dtypes
```

```
Out[5]: age      int64
sex      int64
cp       int64
trestbps int64
chol     int64
fbs      int64
restecg  int64
thalach  int64
exang    int64
oldpeak  float64
slope    int64
ca       int64
thal     int64
target   int64
dtype: object
```

```
In [7]: df.duplicated().sum()
```

```
Out[7]: 1
```

This dataset is almost clean, no null values. But one duplicated value is identified. The data types are uniform. Therefore, we can simply remove the duplicated value to make it clean.

```
In [8]: df.drop_duplicates()
```

```
Out[8]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

302 rows × 14 columns



# Code

## EX1

```
In [12]: import requests
import xml.dom.minidom as m
import time as time
import json
import xml.etree.ElementTree as ET

In [19]: def get_PubMedIds(query, retmax=100):
    search_term = '&term=' + query
    search_retmax = '&retmax=' + str(retmax)
    search_rettype = '&retmode=json'
    search_url = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed'+search_term+search_retmax+search_retty
    r = requests.get(search_url)
    return r.json()['esearchresult']['idlist']

def get_ids():
    AlzheimerTerm = 'Alzheimers+AND+2022[pdat]'
    retmax = 1000
    alz_ids = get_PubMedIds(AlzheimerTerm, retmax)

    CancerTerm = 'Cancer+AND+2022[pdat]'
    cancer_ids = get_PubMedIds(CancerTerm, retmax)
    return alz_ids, cancer_ids

alz_ids, cancer_ids = get_ids()

In [20]: print(len(alz_ids))
print(len(cancer_ids))
print(alz_ids)
print(cancer_ids)

1000
1000
['36328129', '36327964', '36327171', '36326951', '36326588', '36326095', '36325883', '36325840', '36325692', '36325483', '36
324417', '36324414', '36324408', '36324405', '36324401', '36324176', '36324157', '36324151', '36323521', '36323061', '363228
88', '36322800', '36322495', '36322470', '36321981', '36321927', '36321882', '36321654', '36321615', '36321363', '36321205',
'36321194', '36320609', '36320346', '36319674', '36319270', '36319269', '36319136', '36319095', '36319045', '36318754', '363
18594', '36318545', '36318372', '36317468', '36317413', '36316970', '36316783', '36316708', '36316501', '36316487', '3631646
1', '36316282', '36316035', '36315527', '36315115', '36314730', '36314503', '36314232', '36314212', '36314211', '36314210',
'36314209', '36314208', '36314207', '36314206', '36314205', '36314204', '36314203', '36314202', '36314201', '36314200', '363
14199', '36314055', '36313968', '36313967', '36313955', '36313229', '36312018', '36311713', '36311031', '36310167', '3630993
8', '36309725', '36309404', '36309183', '36309087', '36308033', '36307888', '36307518', '36306920', '36306735', '36306540',
'36306459', '36306458', '36305768', '36305541', '36305459', '36305148', '36305125', '36304998', '36304823', '36304723', '363
04124', '36303870', '36303331', '36303296', '36302665', '36302659', '36302488', '36302464', '36301043', '36299613', '3629960
8', '36298279', '36297317', '36297313', '36296980', '36296969', '36296692', '36296686', '36296677', '36296574', '36296397',
'36295605', '36295535', '36294010', '36293946', '36293666', '36293539', '36293528', '36293516', '36293327', '36293221', '362
93147', '36293049', '36292947', '36292945', '36292933', '36292931', '36292674', '36292623', '36292114', '36291714', '3629167
9', '36291666', '36291661', '36291618', '36291595', '36291553', '36291536', '36291224', '36291125', '36291068', '36291020',
'36291017', '36290612', '36289878', '36289859', '36289565', '36289390', '36289355', '36288997', '36288945', '36288546', '362
88285', '36287605', '36287554', '36286505', '36286438', '36286188', '36285785', '36284665', '36284403', '36284365', '3628436
In [21]: overlap=(set(cancer_ids)&set(alz_ids))
print('overlap is:', overlap)

overlap is: ['36321615', '36321363']
```

```
In [16]: def get_info(pmid, deep = 0):
time.sleep(1)
r = requests.get(
'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed&retmode=xml&id=' + pmid
)

tree = EI.fromstring(r.text)
try:
titles = tree.find('PubmedArticle/MedlineCitation/Article/ArticleTitle')
title = EI.tostring(titles, method="text").decode()
except:
if deep > 5:
print("Error: {} No title max deep reached".format(pmid))
title = ""
else:
return get_info(pmid, deep + 1)
abstracts = tree.find('PubmedArticle/MedlineCitation/Article/Abstract/AbstractText')
try:
abstract = EI.tostring(abstracts, method="text").decode()
except:
if deep > 5:
print("Error: {} No abstract max deep reached".format(pmid))
abstract = ""
else:
return get_info(pmid, deep + 1)
return title, abstract
```

```
In [17]: def json_output():
dictionary = {}
for i in alz_ids:
title, abstract = get_info(i)
dictionary[i] = {'ArticleTitle': title, 'ArticleAbstract': abstract, 'query': 'Alzheimer'}
for i in cancer_ids:
title, abstract = get_info(i)
dictionary[i] = {'ArticleTitle': title, 'ArticleAbstract': abstract, 'query': 'Cancer'}
json_obj = json.dumps(dictionary, indent=4)

print(dictionary)
with open("alz&cancer.json", "w") as outfile:
outfile.write(json_obj)
```

```
In [18]: json_output()
```

```
Error: 36319136 No abstract max deep reached
Error: 36316783 No abstract max deep reached
Error: 36314232 No abstract max deep reached
Error: 36310167 No abstract max deep reached
Error: 36306458 No abstract max deep reached
Error: 36304998 No abstract max deep reached
Error: 36299613 No abstract max deep reached
Error: 36284665 No abstract max deep reached
Error: 36284252 No abstract max deep reached
Error: 36284251 No abstract max deep reached
Error: 36281688 No abstract max deep reached
Error: 36281687 No abstract max deep reached
Error: 36281660 No abstract max deep reached
Error: 36281659 No abstract max deep reached
Error: 36281658 No abstract max deep reached
Error: 36278375 No abstract max deep reached
Error: 36278006 No abstract max deep reached
Error: 36275013 No abstract max deep reached
Error: 36271398 No abstract max deep reached
```

## EX2

```
In [1]: pip install transformers
```

```
Requirement already satisfied: transformers in d:\anaconda\lib\site-packages (4.24.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.10.0 in d:\anaconda\lib\site-packages (from transformers) (0.10.1)
Requirement already satisfied: requests in d:\anaconda\lib\site-packages (from transformers) (2.27.1)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in d:\anaconda\lib\site-packages (from transformers) (0.13.1)
Requirement already satisfied: pyyaml>=5.1 in d:\anaconda\lib\site-packages (from transformers) (6.0)
Requirement already satisfied: numpy>=1.17 in d:\anaconda\lib\site-packages (from transformers) (1.21.5)
Requirement already satisfied: tqdm>=4.27 in d:\anaconda\lib\site-packages (from transformers) (4.64.0)
Requirement already satisfied: regex!=2019.12.17 in d:\anaconda\lib\site-packages (from transformers) (2022.3.15)
Requirement already satisfied: filelock in d:\anaconda\lib\site-packages (from transformers) (3.6.0)
Requirement already satisfied: packaging>=20.0 in d:\anaconda\lib\site-packages (from transformers) (21.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in d:\anaconda\lib\site-packages (from huggingface-hub<1.0,>=0.10.0->transformers) (4.1.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in d:\anaconda\lib\site-packages (from packaging>=20.0->transformers) (3.0.4)
Requirement already satisfied: colorama in d:\anaconda\lib\site-packages (from tqdm>=4.27->transformers) (0.4.4)
Requirement already satisfied: idna<4,>=2.5 in d:\anaconda\lib\site-packages (from requests->transformers) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in d:\anaconda\lib\site-packages (from requests->transformers) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in d:\anaconda\lib\site-packages (from requests->transformers) (2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in d:\anaconda\lib\site-packages (from requests->transformers) (1.26.9)
Note: you may need to restart the kernel to use updated packages.
```

```
import torch
```

```
In [1]: import json
import tqdm
import pandas as pd
import matplotlib.pyplot as plt
```

```
In [2]: from transformers import AutoTokenizer, AutoModel

# load model and tokenizer
tokenizer = AutoTokenizer.from_pretrained('allenai/specter')
model = AutoModel.from_pretrained('allenai/specter')
```

```
Downloading: 0% | 0.00/440M [00:00<?, ?B/s]
```

```
D:\Anaconda\lib\site-packages\huggingface_hub\file_download.py:123: UserWarning: `huggingface_hub` cache-system uses symlinks by default to efficiently store duplicated files but your machine does not support them in C:\Users\weipa\.cache\huggingface\hub. Caching files will still work but in a degraded version that might require more space on your disk. This warning can be disabled by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface\_hub/how-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to see activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development
warnings.warn(message)
```

```
In [32]: alz_cancer_papers = json.load(open('alz&cancer.json'))
print(len(alz_cancer_papers))
```

```
1998
```

[illegible]

```
embeddings_pca["query"] = [paper["query"] for paper in papers.values()]
```

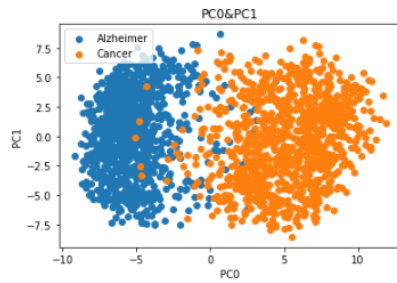
```
embeddings_pca
```

	PC0	PC1	PC2	query
0	-1.040141	-5.184434	-0.222884	Alzheimer
1	-5.402351	4.374268	0.928445	Alzheimer
2	-5.520820	4.610436	-1.998771	Alzheimer
3	-5.684639	-4.256922	-0.069922	Alzheimer
4	-5.909279	-2.143645	0.723237	Alzheimer
...	...	...	...	...
1993	1.974529	-4.315622	5.541237	Cancer
1994	5.472513	6.085035	3.709071	Cancer
1995	3.539508	-4.110456	2.859843	Cancer
1996	2.834004	-2.746227	4.597482	Cancer
1997	7.089354	-7.298180	2.489772	Cancer

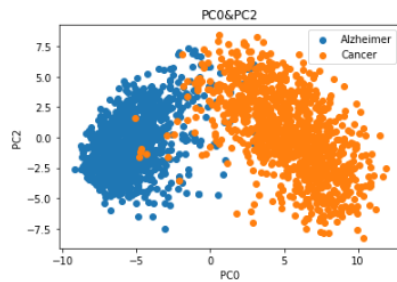
1998 rows  $\times$  4 columns

```
In [54]: alz_PC0 = embeddings_pca[embeddings_pca['query']=='Alzheimer']['PC0']
alz_PC1 = embeddings_pca[embeddings_pca['query']=='Alzheimer']['PC1']
alz_PC2 = embeddings_pca[embeddings_pca['query']=='Alzheimer']['PC2']
cancer_PC0 = embeddings_pca[embeddings_pca['query']=='Cancer']['PC0']
cancer_PC1 = embeddings_pca[embeddings_pca['query']=='Cancer']['PC1']
cancer_PC2 = embeddings_pca[embeddings_pca['query']=='Cancer']['PC2']

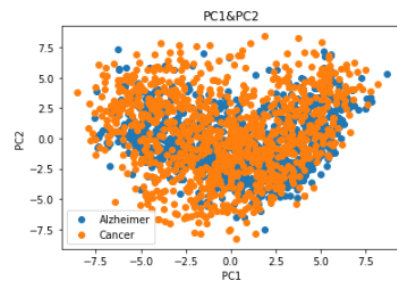
plt.scatter(alz_PC0, alz_PC1, label="Alzheimer")
plt.scatter(cancer_PC0, cancer_PC1, label="Cancer")
plt.title("PC0&PC1")
plt.xlabel("PC0")
plt.ylabel("PC1")
plt.legend()
plt.show()
```



```
In [55]: plt.scatter(alz_PC0, alz_PC2, label="Alzheimer")
plt.scatter(cancer_PC0, cancer_PC2, label="Cancer")
plt.title("PC0&PC2")
plt.xlabel("PC0")
plt.ylabel("PC2")
plt.legend()
plt.show()
```



```
In [56]: plt.scatter(alz_PC1, alz_PC2, label="Alzheimer")
plt.scatter(cancer_PC1, cancer_PC2, label="Cancer")
plt.title("PC1&PC2")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.show()
```



EX3

```
In [61]: import matplotlib.pyplot as plt
import numpy as np

def derivative(x, beta, gamma, N):
    s, i, r = x
    return np.array([-beta * s * i / N, beta * s * i / N - gamma * i, gamma * i])

# define standard explicit euler function
def explicit_euler(x0, t0, dt, T, beta, gamma, N):
    x = [x0]
    t = t0 + dt
    while t < T:
        x.append(x[-1] + dt * derivative(x[-1], beta, gamma, N))
        t += dt
    return x
```

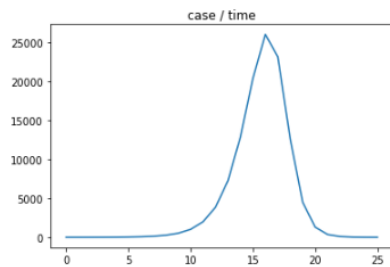
```
In [62]: # apply explicit euler to solve the SIR model
def solve(s0, i0, r0, beta, gamma, T):
    x0 = np.array([s0, i0, r0])
    t0 = 0
    N = s0 + i0 + r0
    dt = 1 # time step = 1 day
    x = explicit_euler(x0, t0, dt, T, beta, gamma, N)
    t = np.arange(0, T + dt, dt)
    s, i, r = np.array(x).T
    return s, i, r, t

def SIR_model(s0, i0, r0, beta, gamma, T):
    s, i, r, t = solve(s0, i0, r0, beta, gamma, T)
    plt.plot(t, i)
```

```
In [86]: N = 134000
s, i, r, t = solve(N - 1, 1, 0, 2, 1, 30)
infected_drop = 0
while (i[infected_drop] >= 1):
    infected_drop += 1
print("The number of infected individuals drop below 1 at day:", infected_drop)
plt.plot(t[:infected_drop], i[:infected_drop])
plt.title("case / time")
```

The number of infected individuals drop below 1 at day: 26

Out[86]: Text(0.5, 1.0, 'case / time')



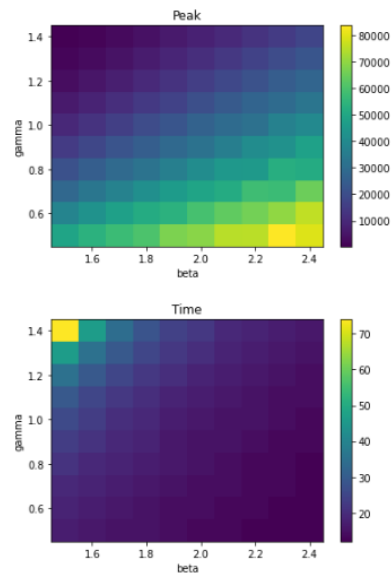
```
In [89]: N = 134000
s, i, r, t = solve(N - 1, 1, 0, 2, 1, 30)
day = 1
while (1):
    if (i[day] > i[day - 1]) and (i[day] > i[day + 1]):
        break
    day += 1
print ("Peak is day:", day, "; Infected cases are:", i[day])
```

Peak is day: 16 ; Infected cases are: 26033.391521237274

```
In [84]: def heatmap():
    beta = np.arange(1.5, 2.5, 0.1)
    gamma = np.arange(0.5, 1.5, 0.1)
    peak = np.zeros((len(gamma), len(beta)))
    time = np.zeros((len(gamma), len(beta)))
    for i in range(len(beta)):
        for j in range(len(gamma)):
            ind, inf = find_peak(beta[i], gamma[j])
            peak[j][i] = inf
            time[j][i] = ind
    return peak, time
peak, time = heatmap()
```

```
In [85]: beta = np.arange(1.5, 2.5, 0.1)
gamma = np.arange(0.5, 1.5, 0.1)
plt.pcolormesh(beta, gamma, peak)
plt.colorbar()
plt.xlabel("beta")
plt.ylabel("gamma")
plt.title("Peak")
plt.show()

plt.pcolormesh(beta, gamma, time)
plt.colorbar()
plt.xlabel("beta")
plt.ylabel("gamma")
plt.title("Time")
plt.show()
```



EX4

```
In [2]: import pandas as pd

df = pd.read_csv('heart.csv')

print(df.to_string())
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
10	54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
11	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
12	49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
13	64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
14	58	0	3	150	283	1	0	162	0	1.0	2	0	2	1
15	50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
16	58	0	2	120	340	0	1	172	0	0.0	2	0	2	1
17	66	0	3	150	226	0	1	114	0	2.6	0	0	2	1

```
In [3]: df.isnull().sum()
```

```
Out[3]: age      0
sex        0
cp         0
trestbps   0
chol       0
fbs        0
restecg    0
thalach    0
exang      0
oldpeak    0
slope      0
ca         0
thal       0
target     0
dtype: int64
```

```
In [4]: df.dtypes
```

```
Out[4]: age      int64
sex      int64
cp       int64
trestbps  int64
chol     int64
fbs      int64
restecg  int64
thalach  int64
exang    int64
oldpeak  float64
slope    int64
ca       int64
thal     int64
target   int64
dtype: object
```

```
In [7]: df.duplicated().sum()
```

```
Out[7]: 1
```

```
In [8]: df.drop_duplicates()
```

```
Out[8]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

302 rows × 14 columns



