# Assignment 3

Instructions: All python libraries used are listed on the top of each Jupyter notebook file.

Answers:

**Exercise 1:**

1. The requests module is used to get PubMed IDs of 1000 Alzheimer papers in 2022 and 1000 cancer papers in 2022.

```
def post_ids(target,lenth=1000,year=2022,articleOnly=True):
    a='+article' if articleOnly else ''
    r = requests.post(
    "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/"
    f"esearch.fcgi?db=pubmed&term={target}+AND+{year}[pdat]{a}&retmode
     =xml&retmax={lenth}&retstart=0"
    )
    doc = m.parseString(r.text)
    ids = doc.getElementsByTagName("Id")
    idResult=[id.childNodes[0].wholeText for id in ids]
    return idResult
```

2. The requests module is used to pull metadata for each paper.

```
def post_infoes(pmid):
    if len(pmid)==0:
        return pd.DataFrame()
    pmidStr=','.join(pmid)
    print(len(pmid))
    r = requests.post(
    "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/"
    f"efetch.fcgi?db=pubmed&retmode=xml&id={pmidStr}"
    ,timeout=30
    )
    tree = ET.fromstring(r.text)
    res=[]

    for node in tree:
        for iter in node.iter("PMID"):
            id=ET.tostring(iter, method="text").decode()
            break
        for iter in node.iter("ArticleTitle"):
            title=ET.tostring(iter, method="text").decode()
            break
```

Each category papers' titles, abstracts and queries are stored in a .json file and uploaded in the GitHub repository: Assignment3/data/, i.e., Alzheimer.json and cancer.json. These two .json files are of the general form so they can be combined for later use.


```
"36322470": {
    "ArticleTitle": "Estimating prevalence of early Alzheimer's disease in the United States, accounting for racial and ethnic diversity.",
    "AbstractText": "Updated estimates of the US Alzheimer's disease (AD) population, including under-represented populations, are needed to i
by setting recruitment goals reflecting the diversity of the AD patient population and supporting efforts toward timely diagnosis.",
    "query": "Alzheimer"
},
"36321981": {
    "ArticleTitle": "Amyloid-Related Imaging Abnormalities: An Update.",
```


3. An overlap ID: 36321615 is found by using & operation within two ID sets.

```
1  same=set(ids_az)&set(ids_cn)
2  print('find_overlap:',same)

   find_overlap: {'36321615'}
```

4. Multiple AbstractText fields in a paper are all stored by simply concatenating with a space in between. Pros is that it saves programming time and memory space. Cons is that it will lose **the paragraph structure information** in abstracts. For example, below is an abstract with Introduction, Methods, Results and Discussion, 4 paragraphs. Our method joins the 4 paragraphs with a space in between in one list as one paragraph rather than using a dictionary, {introduction:text, methods:text, results: text, discussion: text}. List is easier for later generically processing but it loses the label information.


```
▼<Abstract>
   <AbstractText Label="INTRODUCTION" NlmCategory="BACKGROUND">Supplementation wi
   of Alzheimer's disease (AD).</AbstractText>
   <AbstractText Label="METHODS" NlmCategory="METHODS">Data from 659 participants
   investigated the association between spermidine plasma levels and markers of b
   [WMH]).</AbstractText>
   <AbstractText Label="RESULTS" NlmCategory="RESULTS">Higher spermidine levels w
   -0.13 to -0.02; q =  0.026), higher AD score (ß  =  0.118; 95% CI: 0.05 to 0.1
   Sensitivity analysis revealed no substantial changes after excluding participa
   <AbstractText Label="DISCUSSION" NlmCategory="CONCLUSIONS">Elevated spermidine
   and vascular brain pathology.</AbstractText>
   <CopyrightInformation>© 2022 The Authors. Alzheimer's & Dementia published by
 </Abstract>
```


**Exercise 2:**

1. To keep track of papers' labels, I store two category papers in two .json files and do embeddings separately. Then concatenate two embeddings arrays together. Therefore, the first 1000 rows are Alzheimer and the rest rows are cancer.
2. The papers dictionaries are loaded by pd.read_json(). They are processed as follows to generate 'data' to be the input of SPECTER embeddings. Alzheimer and cancer's 768-dim embeddings are concatenated to form a (2000,768) matrix.

```
papers=pd.read_json('./data/Alzheimer.json')
for pmid, paper in tqdm.tqdm(papers.items()):
 data = [paper["ArticleTitle"] + tokenizer.sep_token +
 paper['AbstractText']]
```

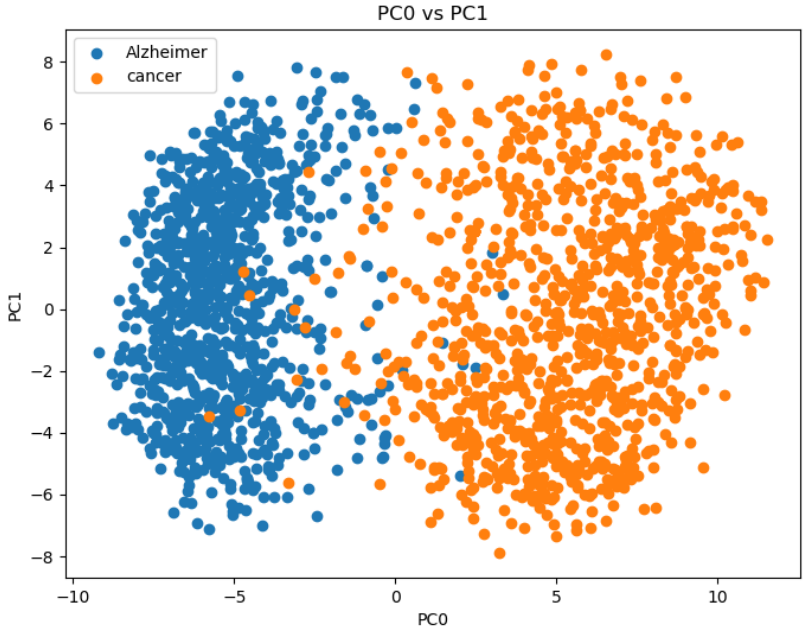
```
import numpy as np
embeddings_total=np.concatenate((embeddings,embeddings2))
print(embeddings_total.shape)
```

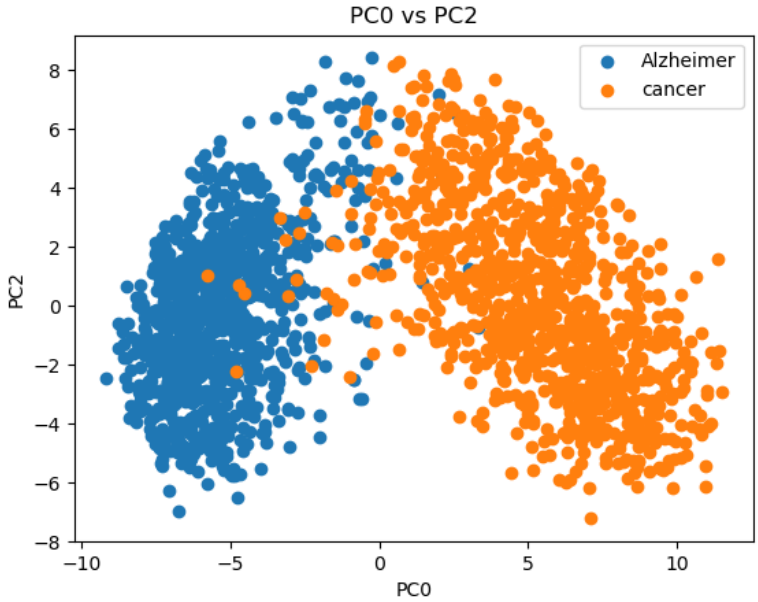
```
(2000, 768)
```

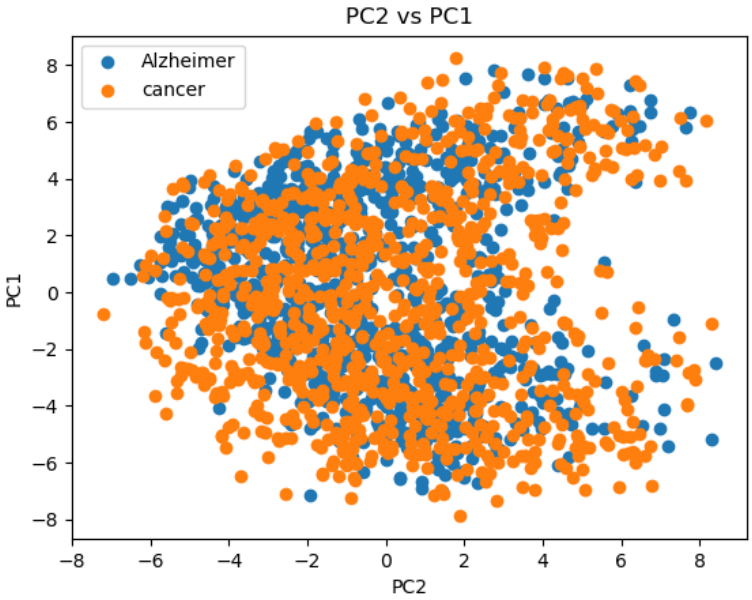
3. PCA is applied to identify the first three principal components.

```
from sklearn import decomposition
pca = decomposition.PCA(n_components=3)
embeddings_pca = pd.DataFrame(
 pca.fit_transform(embeddings_total),
 columns=['PC0', 'PC1', 'PC2']
)
query_list=['Alzheimer']*1000+['cancer']*1000
embeddings_pca["query"] = query_list
```

4. PC0 vs PC1, PC0 vs PC2, PC1 vs PC2 plots are shown below.


PC0 vs PC1
Alzheimer
cancer
PC1
PC0
8
6
4
2
0
−2
−4
−6
−8
−10
−5
0
5
10


PC0 vs PC2
Alzheimer
cancer
PC2
PC0
8
6
4
2
0
−2
−4
−6
−8
−10
−5
0
5
10




5. It is obvious that PC0-PC1 and PC0-PC2 plots' clusters are more separable than that of the PC1-PC2 plot. The reason is that PC0 is the first principle component so it contains most information about the data. PC1 and PC2 contains less information as shown below.

```
print(pca.explained_variance_ratio_)

[0.16913469 0.06433377 0.04573277]
```

With PC0 and PC1, we have sufficient information to separate Alzheimer and cancer with good accuracy. To improve the classification accuracy, we can add PC2 but it won't generate a very significant improvement in accuracy as it doesn't contain so much information as PC0 does. Also, from the PC1-PC2 plot, we can see that only using these two components can not separate the two categories as they don't contain enough information for classification.

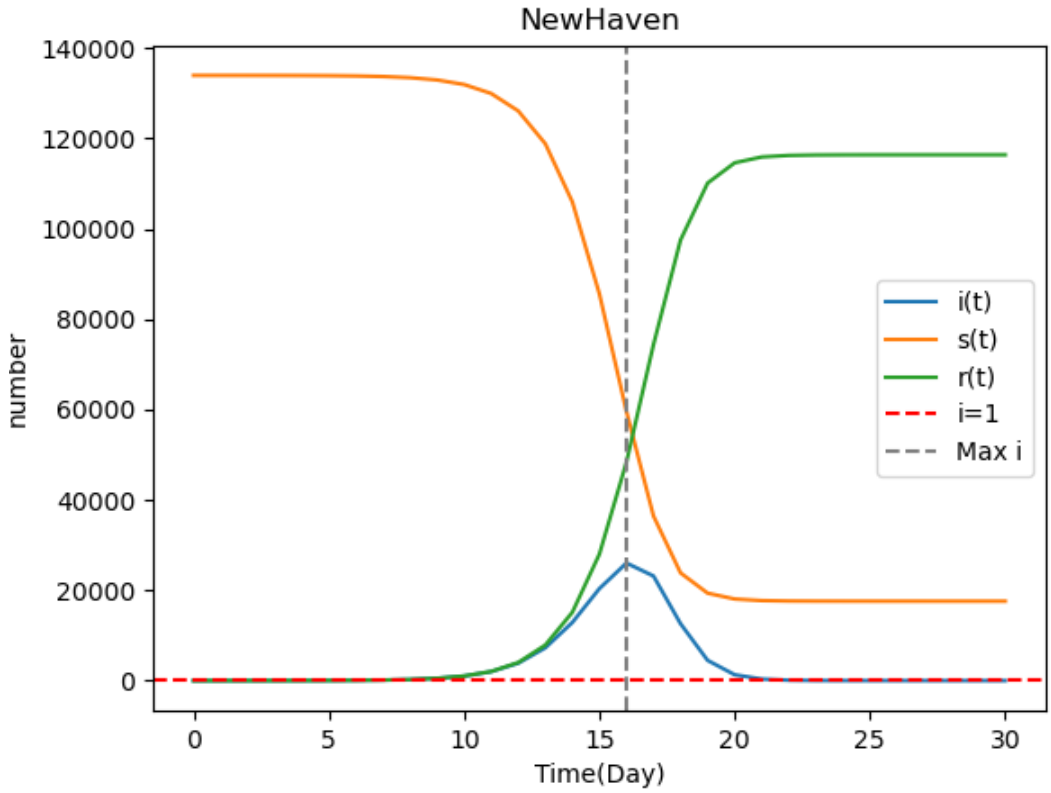
**Exercise 3:**

1. The python function Explicit_euler(s0,i0,r0,beta,gamma,Tmax) is written, following the explicit euler method: $f(t_i) = f(t_{i-1}) + f'(t_{i-1})(t_i - t_{i-1}) = f(t_{i-1}) + f'(t_{i-1})\Delta t$. Here, $\Delta t$ is set to 1 by default.

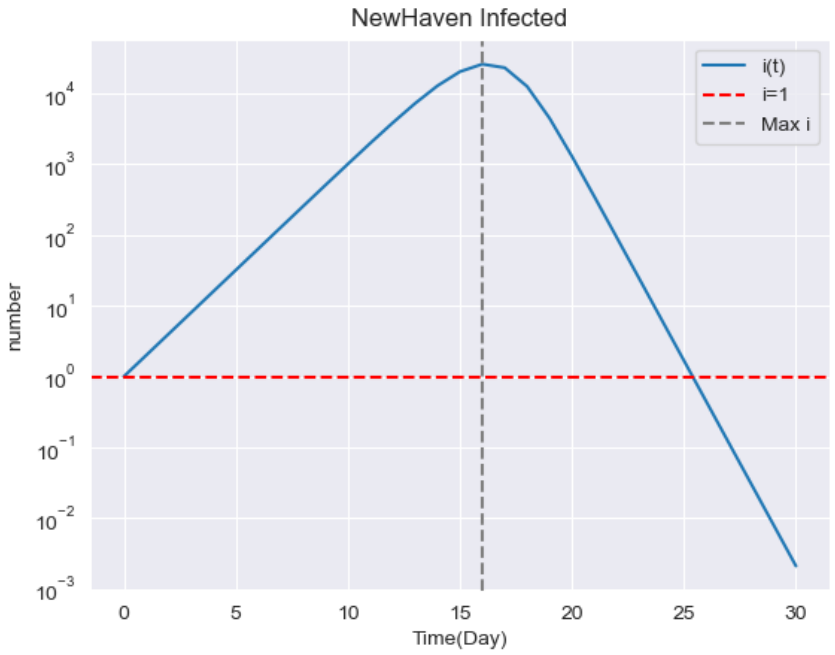
Therefore, $I_t = I_{t-1} + I'_{t-1} * \Delta t = I_{t-1} + \frac{\beta}{N} S_{t-1} I_{t-1} - \gamma I_{t-1}$. The same for S and R.

```python
def Explicit_euler(s0,i0,r0,beta,gamma,Tmax): #Tmax means the final time t, corresponding to i_t
    deltaT=1
    state=np.array([s0,i0,r0],dtype=float)
    history_data=state
    print('initial state0:',history_data)
    n=Tmax/deltaT
    N=s0+i0+r0
    for times in range(int(n)):
        s,i,r=state
        state+=deltaT*np.array([-beta*s*i/N,beta*s*i/N - gamma*i, gamma*i])
        print('state%d:'%(times+1),state)
        # print(type(history_data),type(state))
        history_data=np.vstack((history_data,state))
    return state, history_data
```

2. Below is the S(t),I(t),R(t) plot of New Haven population, starting from I(0)=1, N=134000, beta=2, gamma=1. The red dashed line means I=1.




The intersection of the red and blue lines is where the infected number drops below 1, i.e., the disease ends.

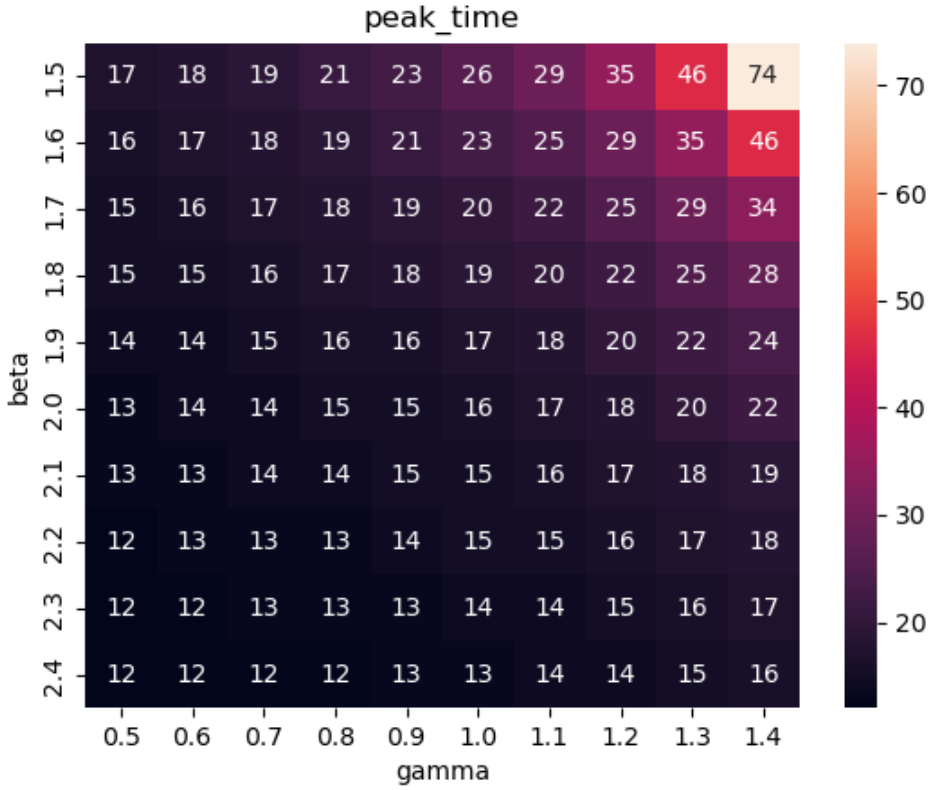



3. The peak time of infected people is Day 16 as the gray dashed line shows. The peak number of infected people is 26033.

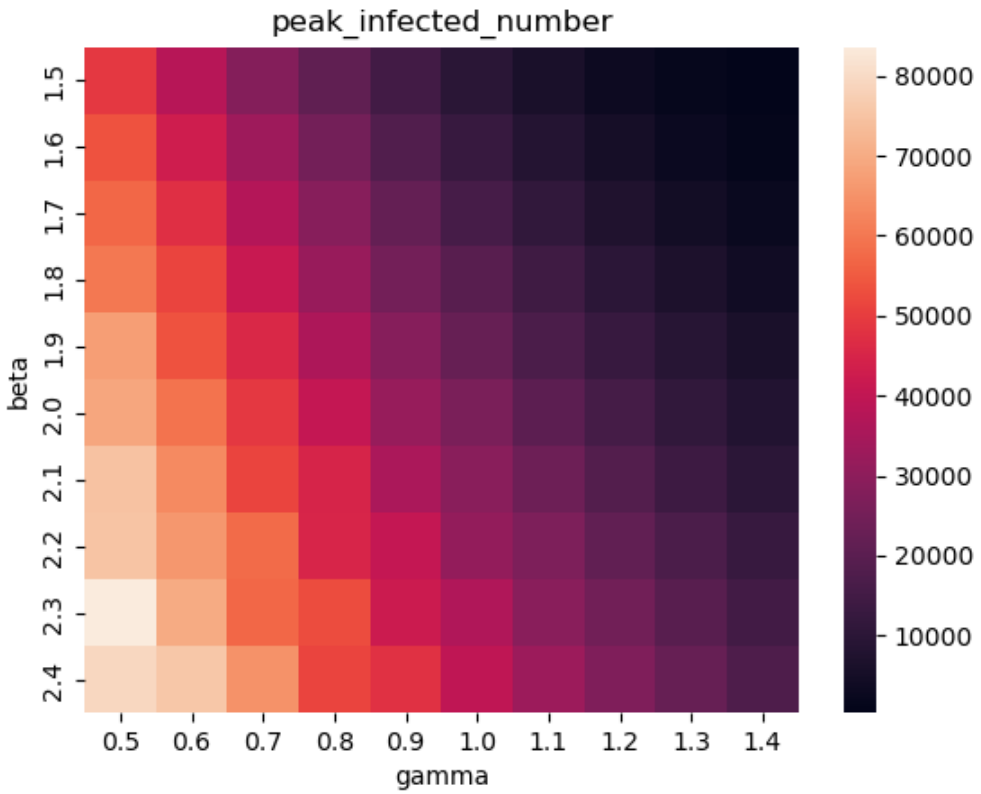
```
I=NewHavenHist[:,1]
index=np.where(I<1)
print('The disease ends at Day',index[0][0],'starts from day0')
print('Infected Peak is Day',np.argmax(I))
print('Number of infected people:',np.max(I))
# print(I)
```

```
31
The disease ends at Day 26 starts from day0
Infected Peak is Day 16
Number of infected people: 26033.391521237274
```

4. Beta varies from 1.5 to 2.4 and Gamma varies from 0.5 to 1.4 evenly. The infected_peak _time heat map is shown below. The annotated number in each grid shows what day the peak time is.




5. The infected_peak_number heat map is shown below.



**Exercise 4:**

1. Identify a data set online:
   The dataset chosen is *Heart failure clinical records Data Set* found on UCI Machine Learning Repository. The source link is https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records#.
   It contains 299 subjects' clinical data with 13 identifiable variables. The predictable target attribute is 'death event', indicating whether the patient died in the follow-up period.
2. Describe the dataset:
   There are 13 variables, all explicitly specified in numerical values.

| Variable name | Units | Data type |
|---|---|---|
| age | years old | numeric |
| anaemia | \ | Binary numeric |
| high blood pressure | \ | Binary numeric |
| creatinine phosphokinase | mcg/L | numeric |
| diabetes | \ | Binary numeric |
| ejection fraction | Percentage | numeric |
| platelets | kiloplatelets/mL | numeric |
| sex | \ | Binary numeric |
| serum creatinine | mg/dL | numeric |
| serum sodium | mEq/L | numeric |
| smoking | \ | Binary numeric |
| time | days | numeric |
| Death event | \ | Binary numeric |

There are no redundant variables but they have weak correlations between each other. The variable 'Death event' has highest correlations with other variables. (i.e. -0.268 with ejection fraction and 0.294 with serum creatinine). It is one example that can be statistically predicted from other variables.

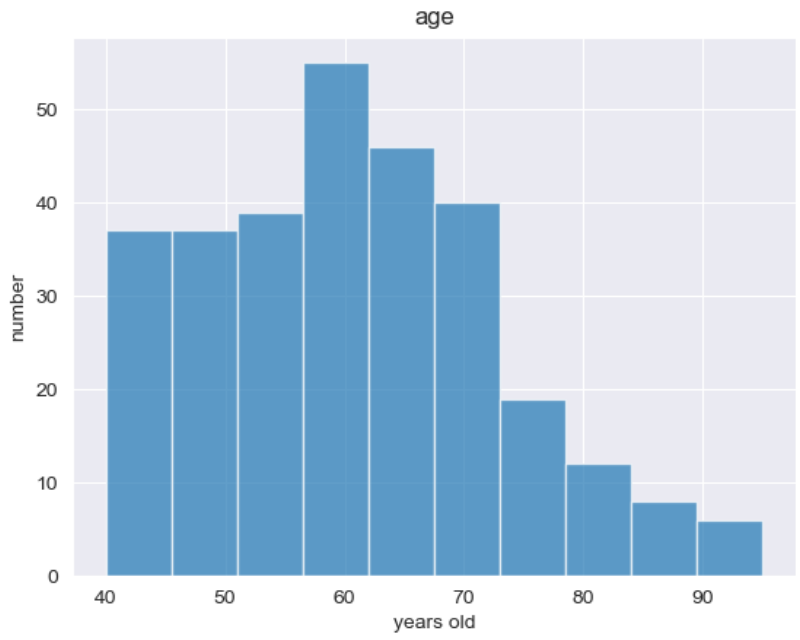
```
print(data.corr())

                              age   anaemia  creatinine_phosphokinase  \
age                      1.000000  0.088006                 -0.081584
anaemia                  0.088006  1.000000                 -0.190741
creatinine_phosphokinase -0.081584 -0.190741                  1.000000
diabetes                -0.101012 -0.012729                 -0.009639
ejection_fraction        0.060098  0.031557                 -0.044080
```

There are 299 rows. The data is not in a standard format as they don't obey the normal distribution $N(0,1)$.

| | age | anaemia | creatinine_phosphokinase | diabetes |
|---|---|---|---|---|
| count | 299.000000 | 299.000000 | 299.000000 | 299.000000 |
| mean | 60.833893 | 0.431438 | 581.839465 | 0.418060 |
| std | 11.894809 | 0.496107 | 970.287881 | 0.494067 |
| min | 40.000000 | 0.000000 | 23.000000 | 0.000000 |
| 25% | 51.000000 | 0.000000 | 116.500000 | 0.000000 |
| 50% | 60.000000 | 0.000000 | 250.000000 | 0.000000 |
| 75% | 70.000000 | 1.000000 | 582.000000 | 1.000000 |
| max | 95.000000 | 1.000000 | 7861.000000 | 1.000000 |

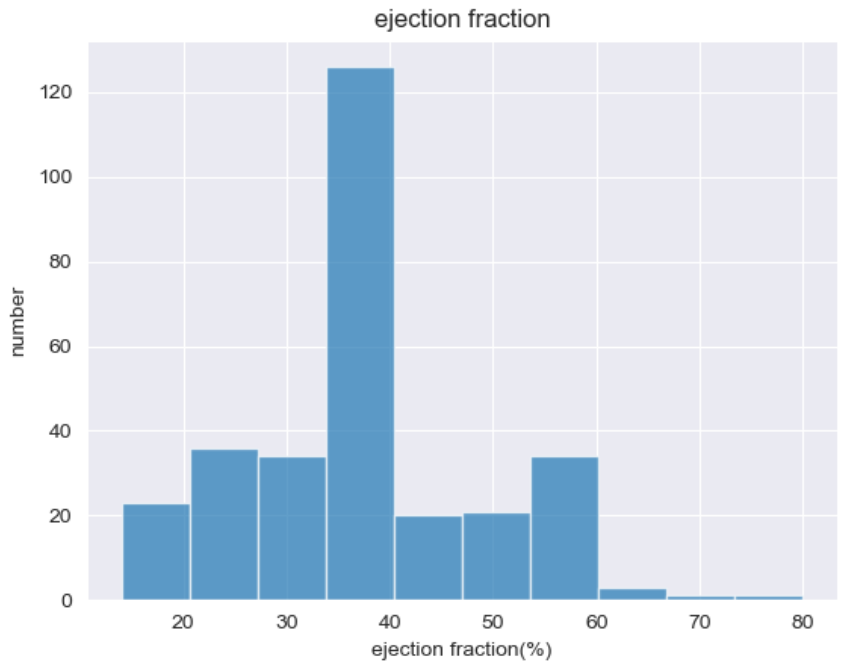
To standardize data $X$, we need to calculate its mean $\bar{X}$ and standard deviation $\sigma$. Then $X_{new} = \frac{X-\bar{X}}{\sigma} \sim N(0,1)$.

3. Terms of use:
   This dataset is an open dataset so I don't have to apply for its access.
   The data is licensed under the Creative Commons Attribution 4.0 International (CC BY 4.0: freedom to share and adapt the material) copyright in January 2020. This is the link of the license https://creativecommons.org/licenses/by/4.0/legalcode. It allows for sharing and adaptation of the data for any purpose. So I can do all kinds of analyses I need.
4. Data exploration:
   This is the 'age' variable distribution. We can see that the subjects' age ranges from 40 to 95. There is no young population (age<30) in our data.

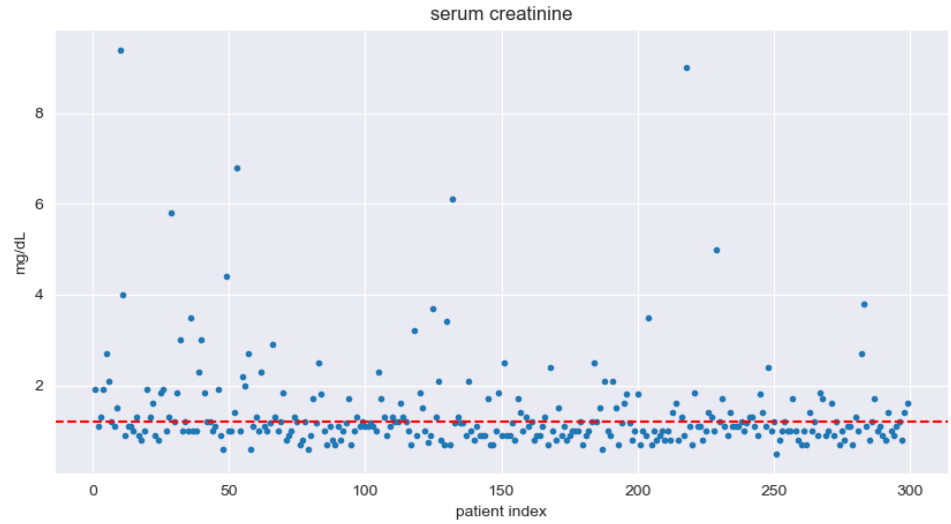



Normal value of ejection fraction is from 50% to 70%. If it is smaller than 50%, we can assume there is a probability of heart failure. Below is the ejection fraction

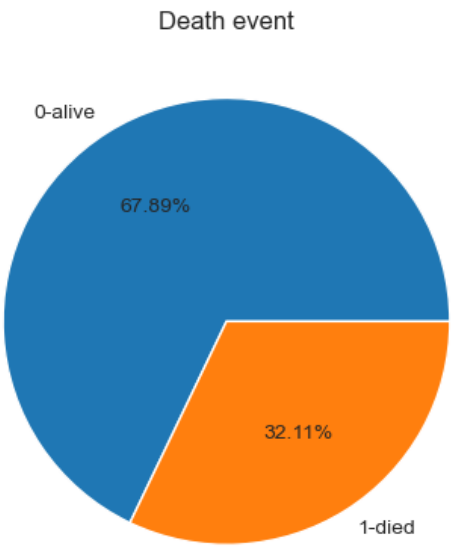
(%) plot of our population. The heart failure patients' majority distribution is below 50%, in line with our diagnostic criteria.



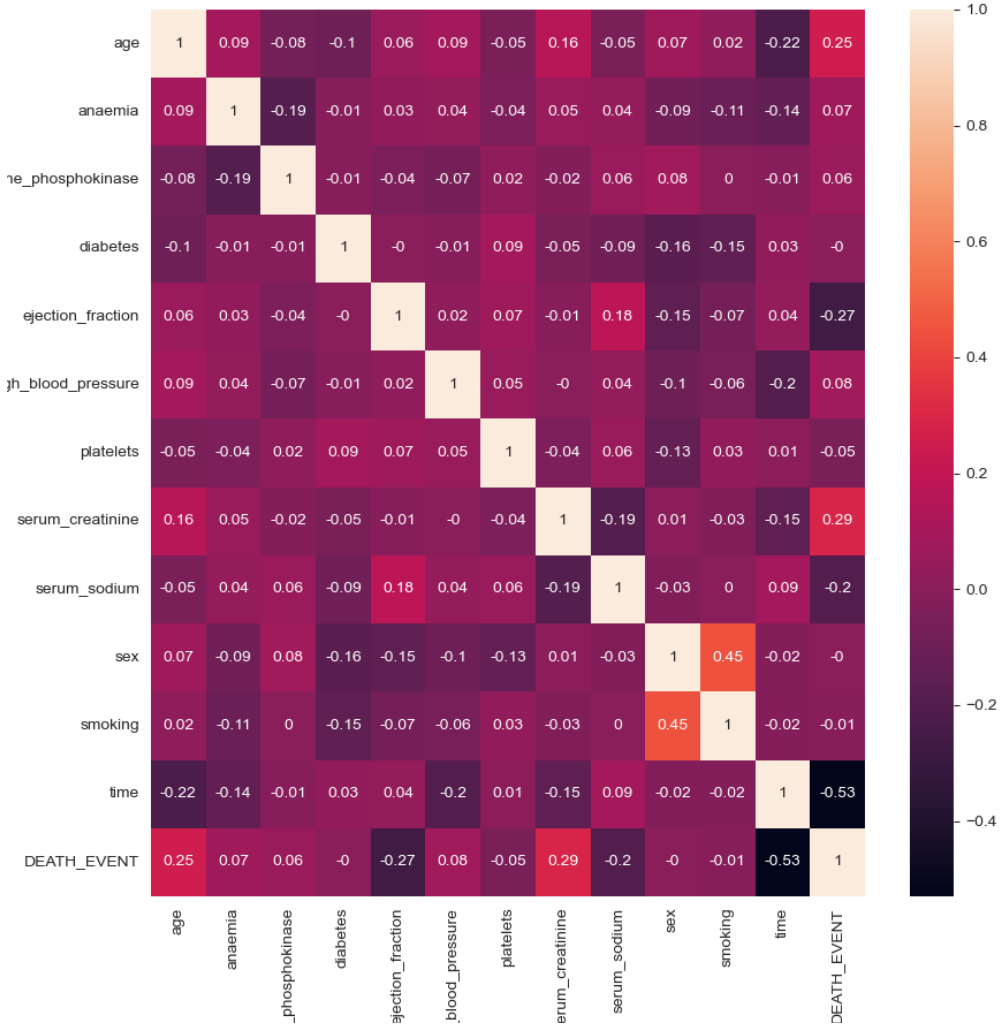

Normal value of serum creatinine is below 1.2mg/dL. If it is higher than 1.2, we can assume there is renal impairment. Below is the serum creatinine(mg/dL) scatterplot of our population. The heart failure patients' majority distribution is below 1.2. But there are still many subjects have renal impairment.




The dataset has 67.89% alive samples and 32.11% death samples, corresponding to 'death event' variable.




The correlation heatmap is shown below, indicating the weak correlation among variables. However, death event has stronger relationships with age, ejection fraction and serum creatinine. It can be predicted by these variables.




5. Data cleaning:
   There is no missing data or duplicate data.

```
print(data.isnull().any())
```

```
age                       False
anaemia                   False
creatinine_phosphokinase  False
diabetes                  False
ejection_fraction         False
high_blood_pressure       False
platelets                 False
serum_creatinine          False
serum_sodium              False
sex                       False
smoking                   False
```

```
print(data.duplicated().sum())
```

```
0
```

But there might be some outliers. Also, different variables have different units. We can standardize it to uniform scale.

To delete outliers, we first define data $x_i$ as an outlier if $|x_i - \bar{x}| > 3\sigma$, where $\bar{x}$ is the mean and $\sigma$ is the standard deviation. For the 12 variables' columns, we calculate every element's Z-score=$\frac{|x_i-\bar{x}|}{\sigma}$. If abs(Z-score)>3, it is an outlier. We save each outlier's row index and delete them. The new dataset has 280 rows, with 19 rows deleted.

```
variables=data.iloc[:,0:-1]
df_zscore = variables.copy()
cols = variables.columns
for col in cols:
    df_col = variables[col]
    z_score = (df_col - df_col.mean()) / df_col.std()
    df_zscore[col] = z_score.abs() > 3
print(df_zscore.any())
df_drop_outlier = data[df_zscore.any(axis=1) == False]
print(df_drop_outlier)
```

```
     6          0    10           1
     ..       ...   ...         ...
     293        1   270           0
     294        1   270           0
     295        0   271           0
     297        1   280           0
     298        1   285           0

     [280 rows x 13 columns]
```

We can apply $x_i = \frac{|x_i-\bar{x}|}{\sigma}$ for data standardization.

```
variables_new=df_drop_outlier.iloc[:,0:-1]
std_data=pd.DataFrame()
for col, series in variables_new.iteritems():
    std_data[col]=np.round((series-series.mean())/series.std(),3)
print(std_data.describe())
```

```
                age     anaemia  creatinine_phosphokinase    diabetes
count  280.000000  280.000000                280.000000  280.000000
mean     0.000011    0.000093                 -0.000054   -0.000257
std      1.000012    0.999933                  0.999981    1.000037
min     -1.775000   -0.877000                 -0.808000   -0.846000
25%     -0.844000   -0.877000                 -0.637000   -0.846000
50%     -0.082000   -0.877000                 -0.404500   -0.846000
75%      0.764000    1.136000                  0.233000    1.178000
max      2.880000    1.136000                  4.352000    1.178000
```

**Code for Exercise1:**

```python
import requests,time
import xml.dom.minidom as m
import xml.etree.ElementTree as ET
import pandas as pd
requests.adapters.DEFAULT_RETRIES = 5
def post_infoes(pmid):
    if len(pmid)==0:
        return pd.DataFrame()
    pmidStr=','.join(pmid)
    print(len(pmid))
    r = requests.post(
    "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/"
    f"efetch.fcgi?db=pubmed&retmode=xml&id={pmidStr}"
    ,timeout=30
    )
    tree = ET.fromstring(r.text)
    res=[]

    for node in tree:
        for iter in node.iter("PMID"):
            id=ET.tostring(iter, method="text").decode()
            break
        for iter in node.iter("ArticleTitle"):
            title=ET.tostring(iter, method="text").decode()
            break
        abstract=[]
        for iter in node.iter("AbstractText"):    #exclude copyright
            s=ET.tostring(iter, method="text").decode()
            abstract.append(s)
        abstract=' '.join(abstract)
        f=''
        if not id:
            f+=' empty id'
        if not title:
            f+=' empty title'
        if not abstract:
            f+=' empty abstract'
        if f:
            print(str(id)+f)

        res.append({'id':id,'ArticleTitle':title,'AbstractText':abstract})
    return pd.DataFrame(res).set_index('id')

def post_ids(target,lenth=1000,year=2022,articleOnly=True):
    a='+article' if articleOnly else ''
    r = requests.post(
```

```
    "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/"

f"esearch.fcgi?db=pubmed&term={target}+AND+{year}[pdat]{a}&retmode=xml&retmax={lenth}&retstart=0"
    )
    doc = m.parseString(r.text)
    ids = doc.getElementsByTagName("Id")
    idResult=[id.childNodes[0].wholeText for id in ids]
    return idResult
#%%
lenth=1000
batch=350
query='Alzheimer'
ids_az=post_ids(query,lenth,2022,True)
result=[]
for i in range(int(lenth/batch+1)):
    time.sleep(1)
    result.append(post_infoes(ids_az[i*batch:(i+1)*batch]))
result=pd.concat(result,axis=0)
print(result)
queryList=[query]*result.shape[0]
result['query']=queryList
result.to_json(f'./data/{query}.json',orient="index")
#%%
query = 'cancer'
ids_cn = post_ids(query, lenth, 2022, True)
result = []
for i in range(int(lenth / batch + 1)):
    time.sleep(1)
    result.append(post_infoes(ids_cn[i * batch:(i + 1) * batch]))
result = pd.concat(result, axis=0)
print(result)
queryList = [query] * result.shape[0]
result['query'] = queryList
result.to_json(f'./data/{query}.json', orient="index")
#%%
same=set(ids_az)&set(ids_cn)
print('find_overlap:',same)
```

**Code for Exercise2:**

```
from transformers import AutoTokenizer, AutoModel

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

```
#%%
import tqdm
import pandas as pd
# we can use a persistent dictionary (via shelve) so we can stop and restart if needed
# alternatively, do the same but with embeddings starting as an empty dictionary
embeddings = {}
papers=pd.read_json('./data/Alzheimer.json')
for pmid, paper in tqdm.tqdm(papers.items()):
 data = [paper["ArticleTitle"] + tokenizer.sep_token + paper['AbstractText']]
 inputs = tokenizer(
 data, padding=True, truncation=True, return_tensors="pt", max_length=512
 )
 result = model(**inputs)
 # take the first token in the batch as the embedding
 embeddings[pmid] = result.last_hidden_state[:, 0, :].detach().numpy()[0]
# turn our dictionary into a list
embeddings = [embeddings[pmid] for pmid in papers.keys()]
#%%
print(type(embeddings))
print(len(embeddings),len(embeddings[0]))
#%%
embeddings2 = {}
papers=pd.read_json('./data/cancer.json')
for pmid, paper in tqdm.tqdm(papers.items()):
 data = [paper["ArticleTitle"] + tokenizer.sep_token + paper['AbstractText']]
 inputs = tokenizer(
 data, padding=True, truncation=True, return_tensors="pt", max_length=512
 )
 result = model(**inputs)
 # take the first token in the batch as the embedding
 embeddings2[pmid] = result.last_hidden_state[:, 0, :].detach().numpy()[0]
# turn our dictionary into a list
embeddings2 = [embeddings2[pmid] for pmid in papers.keys()]
#%%
import numpy as np
embeddings_total=np.concatenate((embeddings,embeddings2))
print(embeddings_total.shape)
#%%
from sklearn import decomposition
pca = decomposition.PCA(n_components=3)
embeddings_pca = pd.DataFrame(
 pca.fit_transform(embeddings_total),
 columns=['PC0', 'PC1', 'PC2']
)
query_list=['Alzheimer']*1000+['cancer']*1000
embeddings_pca["query"] = query_list
```

```
#%%
import matplotlib.pyplot as plt
PC0=embeddings_pca['PC0']
PC1=embeddings_pca['PC1']
PC2=embeddings_pca['PC2']
plt.figure(figsize=(8, 6))
for c in np.unique(query_list):
    i = np.where(np.array(embeddings_pca['query'])==c)
    plt.scatter(np.array(PC0)[i], np.array(PC1)[i], label=c)
plt.xlabel('PC0')
plt.ylabel('PC1')
plt.title('PC0 vs PC1')
plt.legend()
plt.savefig('PC0-1.png')
plt.show()

for c in np.unique(query_list):
    i = np.where(np.array(embeddings_pca['query'])==c)
    plt.scatter(np.array(PC0)[i], np.array(PC2)[i], label=c)
plt.xlabel('PC0')
plt.ylabel('PC2')
plt.title('PC0 vs PC2')
plt.legend()
plt.savefig('PC0-2.png')
plt.show()

for c in np.unique(query_list):
    i = np.where(np.array(embeddings_pca['query'])==c)
    plt.scatter(np.array(PC2)[i], np.array(PC1)[i], label=c)
plt.xlabel('PC2')
plt.ylabel('PC1')
plt.title('PC2 vs PC1')
plt.legend()
plt.savefig('PC2-1.png')
plt.show()
#%%
print(pca.explained_variance_ratio_)
```

**Code for Exercise3:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#%%
def Explicit_euler(s0,i0,r0,beta,gamma,Tmax): #Tmax means the final time t,
corresponding to i_t
    deltaT=1
```

```
    state=np.array([s0,i0,r0],dtype=float)
    history_data=state.copy()
    print('initial state0:',history_data)
    n=Tmax/deltaT
    N=s0+i0+r0
    for times in range(int(n)):
        s,i,r=state
        state+=deltaT*np.array([-beta*s*i/N,beta*s*i/N - gamma*i, gamma*i])
        print('state%d:'%(times+1),state)
        # print(type(history_data),type(state))
        history_data=np.vstack((history_data,state))
    return state, history_data
#%%
Nowstate, NewHavenHist=Explicit_euler(134000-1,1,0,2,1,30)
# print(NewHavenHist)
print(len(NewHavenHist))
X=np.arange(len(NewHavenHist))
I=NewHavenHist[:,1]
index=np.where(I<1)
print('The disease ends at Day',index[0][0],'starts from day0')
print('Infected Peak is Day',np.argmax(I))
print('Number of infected people:',np.max(I))
# print(I)
#%%
plt.plot(X,I,label='i(t)')
plt.plot(X,NewHavenHist[:,0],label='s(t)')
plt.plot(X,NewHavenHist[:,2],label='r(t)')
# plt.yscale('log')
plt.xlabel('Time(Day)')
plt.ylabel('number')
plt.title('NewHaven')
plt.axhline(1,color='red',linestyle='--',label='i=1')
plt.axvline(16,color='gray',linestyle='--',label='Max i')
plt.legend()
plt.savefig('Newhaven.png')
plt.show()

plt.plot(X,I,label='i(t)')
plt.yscale('log')
plt.xlabel('Time(Day)')
plt.ylabel('number')
plt.title('NewHaven Infected')
plt.axhline(1,color='red',linestyle='--',label='i=1')
plt.axvline(16,color='gray',linestyle='--',label='Max i')
plt.legend()
```

```
plt.savefig('NewhavenInfected.png')
plt.show()

#%%
beta=np.linspace(1.5,2.5,10,endpoint=False)
gamma=np.linspace(0.5,1.5,10,endpoint=False)
gamma=np.around(gamma,3)
peak_time=np.zeros((10,10))
peak_num=np.zeros((10,10))
for i in range(len(beta)):
    for j in range(len(gamma)):
        Nowstate, NewHavenHist=Explicit_euler(134000-1,1,0,beta[i],gamma[j],100)
        I=NewHavenHist[:,1]
        index=np.where(I<1)
        # print('The disease ends at Day',index[0][0],'starts from day0')
        peak_time[i][j]=np.argmax(I)
        peak_num[i][j]=np.max(I)
        # print('Infected Peak is Day',np.argmax(I))
        # print('Number of infected people:',np.max(I))
import seaborn as sns
df_time=pd.DataFrame(peak_time,index=beta,columns=gamma)
sns.heatmap(df_time,annot=True)
#X-axis is gamma
plt.title('peak_time')
plt.xlabel('gamma')
plt.ylabel('beta')
plt.savefig('heatmap_time.png')
plt.show()
df_num=pd.DataFrame(peak_num,index=beta,columns=gamma)
sns.heatmap(df_num)
plt.title('peak_infected_number')
plt.xlabel('gamma')
plt.ylabel('beta')
plt.savefig('heatmap_num.png')
plt.show()
```

**Code for Exercise4:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#%%
data=pd.read_csv('heart_failure_clinical_records_dataset.csv')
print(data.shape)
data.describe()
#%%
```

```
print(data.corr())
plt.figure(figsize=(10,10))
sns.heatmap(np.round(data.corr(),2),annot=True)
plt.savefig('corr.png')
#%%
plt.hist(data.age,alpha=0.7)
plt.title('age')
plt.xlabel('years old')
plt.ylabel('number')
plt.savefig('age.png')
plt.show()
#%%
plt.hist(data.serum_creatinine,alpha=0.7,bins=30,range=(0,10))
plt.title('serum creatinine')
plt.xlabel('serum_creatinine(mg/dL)')
plt.ylabel('number')
plt.savefig('serum_creatinine.png')
plt.show()
#%%
plt.figure(figsize=(10,5))
plt.scatter(np.linspace(1,299,299,endpoint=True),data.serum_creatinine,marker='.')
plt.axhline(1.2,color='red',linestyle='--',label='1.2mg/dL')
plt.title('serum creatinine')
plt.xlabel('patient index')
plt.ylabel('mg/dL')
plt.savefig('serum_creatinine.png')
plt.legend()
plt.show()
#%%
plt.hist(data.ejection_fraction,alpha=0.7)
plt.title('ejection fraction')
plt.xlabel('ejection fraction(%)')
plt.ylabel('number')
plt.savefig('ejection_fraction.png')
plt.show()
#%%
ratio=data.DEATH_EVENT.value_counts()
# print(ratio[0],ratio[1])
# print(data.DEATH_EVENT.value_counts())
plt.pie(ratio/299,labels=['0-alive','1-died'], autopct='%.2f%%')
plt.title('Death event')
plt.savefig('death.png')
plt.show()
#%%
print(data.isnull().any())
#%%
```

```
print(data.duplicated().sum())
#%%
variables=data.iloc[:,0:-1]
df_zscore = variables.copy()
cols = variables.columns
for col in cols:
    df_col = variables[col]
    z_score = (df_col - df_col.mean()) / df_col.std()
    df_zscore[col] = z_score.abs() > 3
print(df_zscore.any())
df_drop_outlier = data[df_zscore.any(axis=1) == False]
print(df_drop_outlier)
#%%
variables_new=df_drop_outlier.iloc[:,0:-1]
std_data=pd.DataFrame()
for col, series in variables_new.iteritems():
    std_data[col]=np.round((series-series.mean())/series.std(),3)
print(std_data.describe())
```