Assignment 2

##Exercise 1.

1. Function `get_id()` was defined to return the PubmedId list of a specific disease. First, we used `requests.get()` to search for a term in Alzheimers+AND+2022[pdat], and set the retmax =1000. Then, using minidom, we parsed text strings from the response object and obtained the Id elements, and safe using `.getElementsByTagName()` into the list PubmedId & IdList that was created to save all of the PubmedIds. Lastly, similarly to a binary problem in Assignment 2, and finding children in Assignment 1, we used a loop to get all the data of the firstchild elements, and saved them into the IDList.

```
def get_id(disease):
    r = requests.get(f"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.f
    time.sleep(1)
    doc = m.parseString(r.text)
    PubmedId = doc.getElementsByTagName('Id')
    IdList = []
    for i in range(len(PubmedId)):
        IdList.append(PubmedId[i].firstChild.data)

    return IdList
```

Futhermore, it could be noted that there are totally 2000 papers

```
len(get_id("Alzheimers") + get_id("Cancer"))
2000
```

2. There is an overlap in two sets of papers that are identified as '36314209'. To get the answer we first created a function `overlap_papers()` that was defined to determine whether there is an overlap in the papers. The function first runs through the disease1, and disease2 that are being saved at the IDList1, and IDList2 and then remove the duplicates, which allows us to see an overlapping paper.

```
def overlap_papers(disease1,disease2):
    IdList1 = get_id(disease1)
    IdList2 = get_id(disease2)
    set1 = set(IdList1)
    set2 = set(IdList2)
    overlap = list(set1&set2)
    if len(overlap) == 0:
        print("There is no overlap in the two sets of papers")
    elif len(overlap) == 1:
        print(f"There is a overlap in the two sets of papers, the Pubmed Id is
        return overlap[0]
    else:
        print(f"There are overlaps in the two sets of papers, the Pubmed Ids a
        return overlap


overlap_papers('Alzheimers','cancer')
There is a overlap in the two sets of papers, the Pubmed Id is 36314209.
'36314209'
```

Note: by running the same fruction on Thursday 11/3 showed that there are two overlaps

```
overlap_papers('Alzheimers','cancer')
There are overlaps in the two sets of papers, the Pubmed Ids are['36321363',
['36321363', '36321615']
```

3. By defining function `find_metadata()` that finds metadata of the papers in Alzheimers and Cancer sets. We can pull the data separatly for Alzheimers and Cancer papers and save them as json files using `with open` and `json.dump`.

```
alz_data = find_metadata('Alzheimers')
cancer_data = find_metadata('cancer')

# alz_data into a JSON file paper.json.
with open('alzheimers.json','w') as f:
    json.dump(alz_data,f)

# cancer_data into a JSON file paper.json.
with open('cancer.json', 'w') as f:
    json.dump(cancer_data, f)
```

Then using a similar method we could safe all the papers together, use `.update()` to update the dictionary both_papers_data, so that data from both Alzheimer's and Cancer are saved.

```
combined_data = find_metadata('Alzheimers')
cancer_data = find_metadata('cancer')
combined_data.update(cancer_data)

with open('combined.json','w') as f:
    json.dump(combined_data,f)
```

However, it should be noted that now by looking at the size of the combined data list of papers there are only 1999, as it assumed that the paper which is an overlap '36314209' is not counted in this file (fell thorught, and saved a

separate file somewhere else)

##Exercise 2.
AutoTokenizer, AutoModel, and model and tokenizer were imported.

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

Then for each paper separtaly the data was framed and viewed using the `pd.DataFrame.from_dict` , after which it was combined into the `both_papers` and then formed and saved a JSON file

```
# read JSON file using the open function.
with open('combined.json') as f:
    both_papers_data = json.load(f)
```

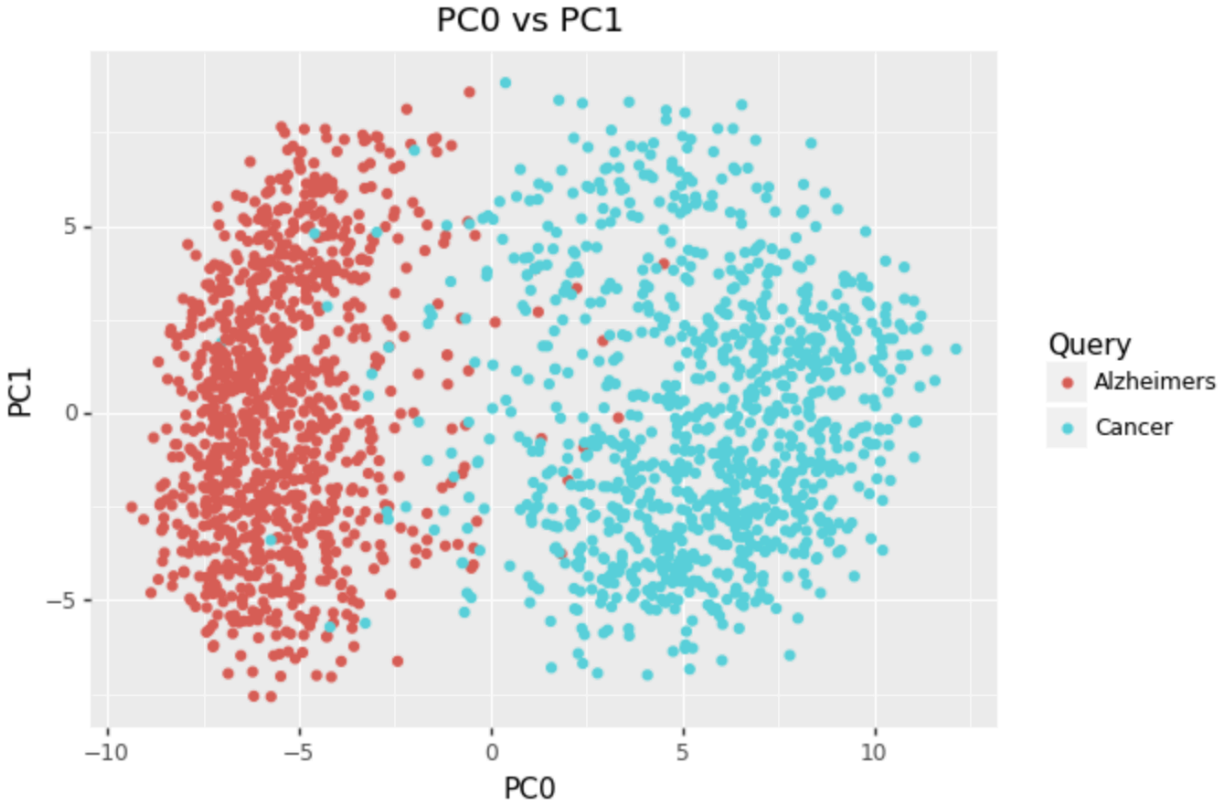
Then the pca was constructed using taken into the account of the first token in the batch as the embedding. embeddings[i] is the 768-dim vector for the ith paper (https://stackoverflow.com/questions/22662996/merge-numpy-arrays-returned-from-loop). After the application of the principal component analysis (PCA) was used to identify the first three principal components that become PC0, PC1 and PC2

	PC0	PC1	PC2	Query
0	-4.590909	5.278322	0.860856	Alzheimers
1	-5.436074	3.557438	-2.107252	Alzheimers
2	-7.084583	5.023624	-1.084028	Alzheimers
3	-5.853919	3.719535	-0.961973	Alzheimers
4	-5.235915	-0.035065	4.223931	Alzheimers
...	...	...	...	...
1994	1.158302	3.854089	4.710636	Cancer
1995	5.926261	7.595884	3.735972	Cancer
1996	3.211766	-0.818690	3.197684	Cancer
1997	2.942921	-3.873605	7.225671	Cancer
1998	3.679746	-0.931543	-1.036632	Cancer

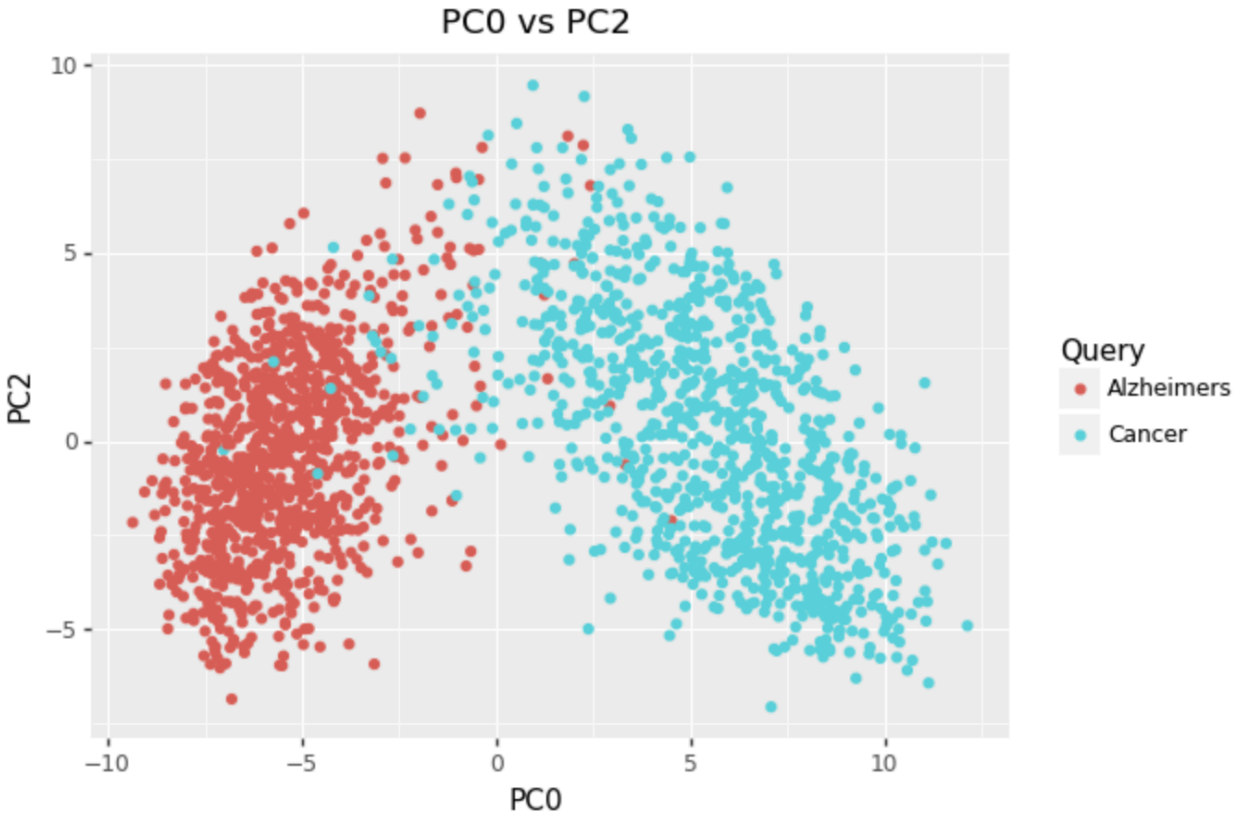
1999 rows × 4 columns

Take the first token in the batch as the embedding for each paper and pca. The first three principal components PC0, PC1, and PC2 were identified using principal component analysis.

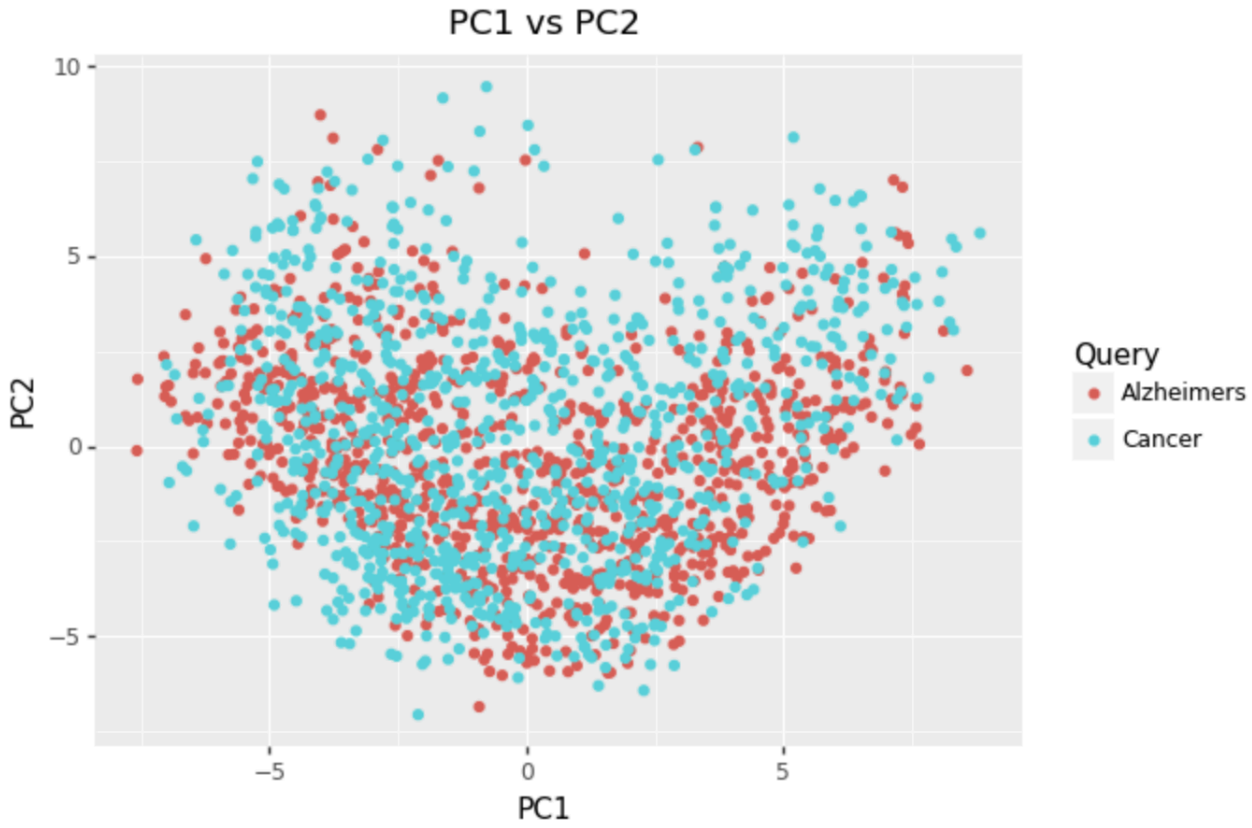
- Scatter plot for PC0 vs PC1 using PCA method Due to the definition of the PCA, we can see that the variation (separation) is obvious, which means that PCA holds the variance of data and we can get more information about it by comparing the dimensions PC0 vs PC1.



- Scatter plot for PC0 vs PC2 using PCA method Similarly to PC0 vs PC1, PC0 vs PC2 we can note a large variance of data form which we can get information



- Scatter plot for PC1 vs PC2 using PCA method However, there are many overlaps in the dimension of PC1 vs PC2, and the variance is not obvious, so we will get less information about the data from this dimension.



##Exercise 3.

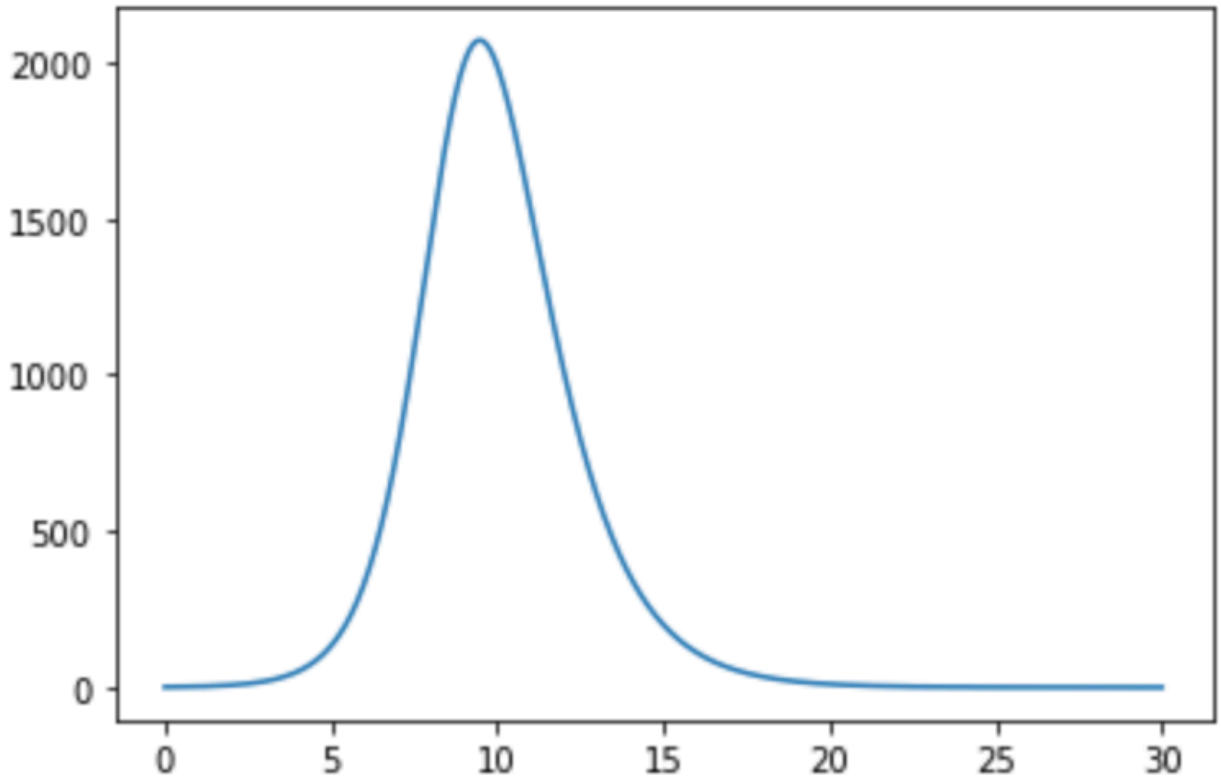
The function `plot_with_explicit_Eulers` was built using the set numbers of parameters `(s0,i0,r0,B,g, tMax)`, that then were Initialize, and arrays had been created. Furthermore, the function included calculating i over a time range and checking if the peak was reached.

Note: `peak_day=t[j−1]` could have been rounded to a neraby date/day using `round(t[j−1])` to the nearest integer/day

1. By looking at the graph we can note that Peak day:9.47999999999999 and was Peak infections: 2071 Note: since The New Haven population is approximately N = 13400 . Suppose that on day 0, there was 1 person infected with a new disease in New Haven and everyone else was susceptible (as the disease is new) - the value used was 13399 (13400-1). And in the parameter we picked 30 days (30 days = month)

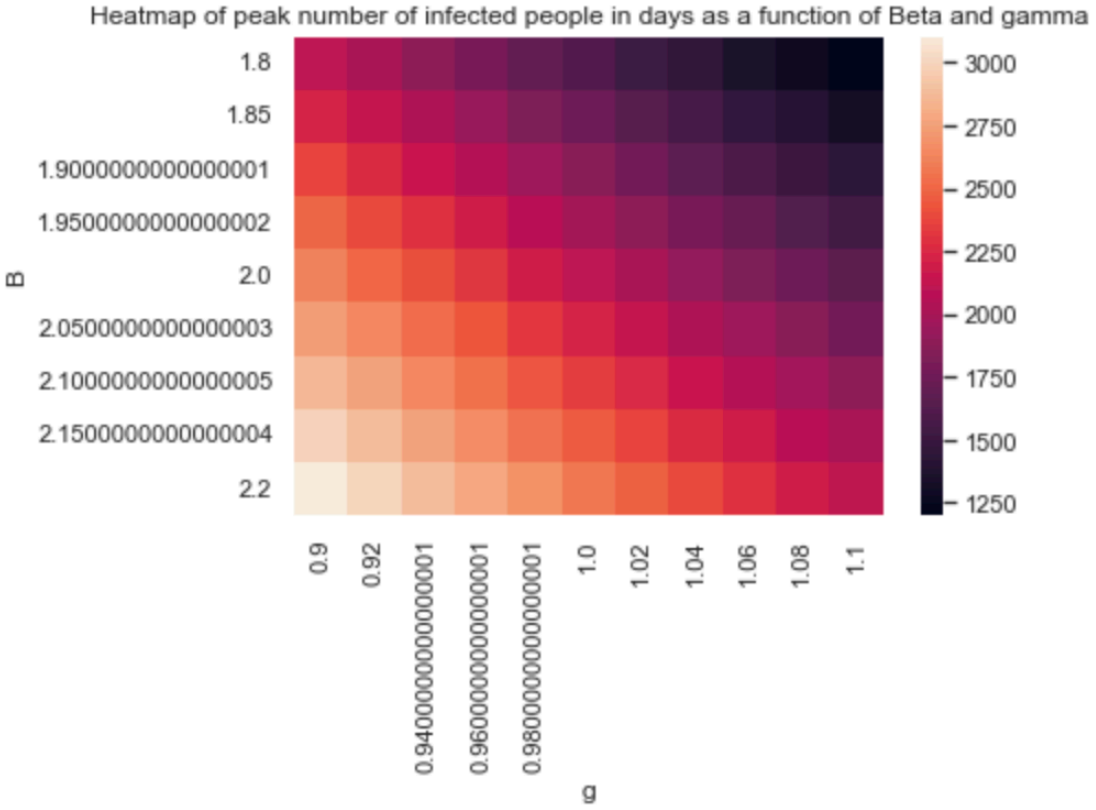
```
[peak_day,peak_infections]=plot_with_explicit_Eulers(13399,1,0,2,1,30)
print("Peak day: ", peak_day)
print("Peak infections: ", peak_infections)
```

```
Peak day:  9.47999999999999
Peak infections:  2071
```



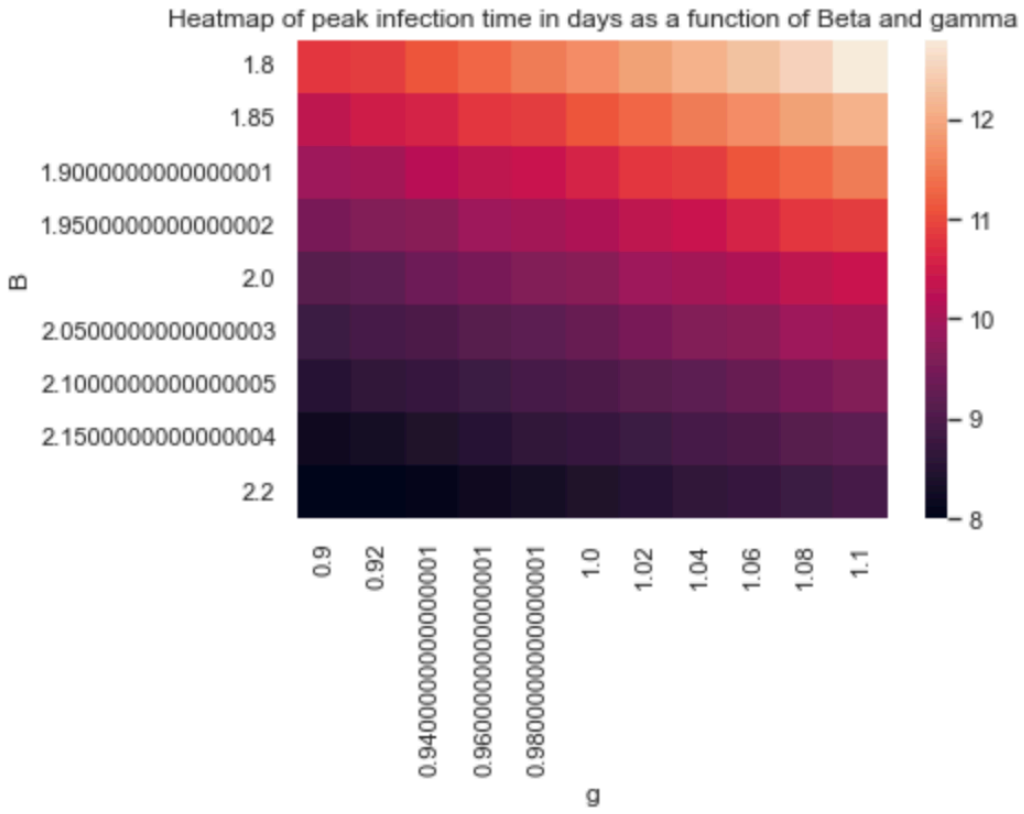
2. Heatmap of peak number of infected people in days as a function of Beta and gamma

```
Text(0.5, 1.0, 'Heatmap of peak number of infected people in days as a fu
nction of Beta and gamma')
```



3. Heatmap of peak infection time in days as a function of Beta and gamma

```
Text(0.5, 1.0, 'Heatmap of peak infection time in days as a function of B
eta and gamma')
```



##Exercise 4.

1. Identify a data set online

```
df = pd.read_csv('/Users/polina/Desktop/Life Expectancy Data.csv')
df
df.info()
```

The online data set I chose is the Life Expectancy(WHO), which consists of data from the period of 2000 to 2015 for all the countries

2. Describe the dataset

- The dataset contains 193 unique values/ countries

- The key variables are represented as numerical values, all the variables in the dataset are numerical except for Country, and Status of the countries
- There aren't almost no varaibles that could be exactlt derived from other variables.
- There are some variables that could be statistically predicted from other variables. For exmaple, statistically we could predict the adult mortality rate based on life expentency, number of infant death and the population of each country given specific year.
- There are 2939 rows and 22 columns,so there are around 64,658 data points (some valus might be missinng)
- The data is in .csv format that could be open in Python

```
# Shape of the data (How many rows and columns in the dataset)
df.shape
# Statistics about the data (mean , std, min etc.)
df.describe()
```

3. The term of use and key restrictution: The data-sets are made available to public for the purpose of health data analysis.

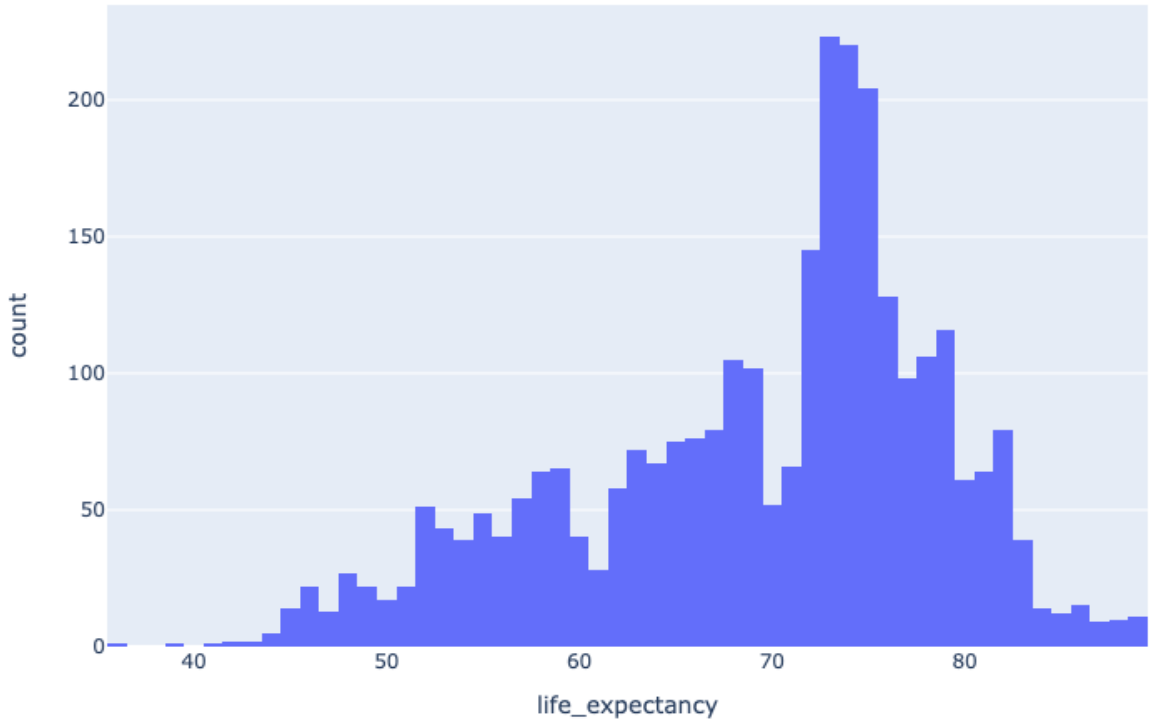
4. Data cleaning: By runnig `data.isnull().sum()` some null values, have been detected. Thus, those values were addressed below by using Python to fill in the values with the data's mean values. The results showed that the majority of the missing data were for the population, Hepatitis B, and GDP.

```
# Replacing the Null Values with mean values of the data
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan,strategy='mean',fill_value=None)
data['Life expectancy ']=imputer.fit_transform(data[['Life expectancy ']])
```

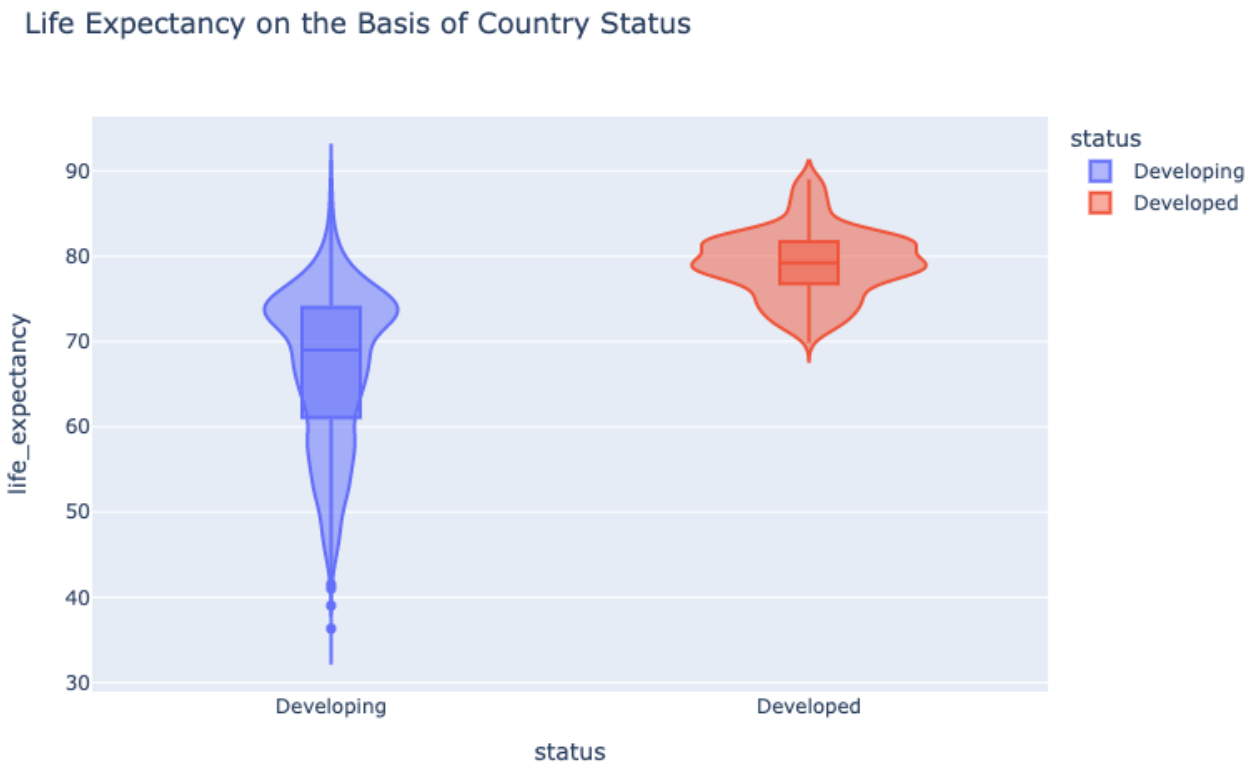
```
data['Adult Mortality']=imputer.fit_transform(data[['Adult Mortality']])
data['Alcohol']=imputer.fit_transform(data[['Alcohol']])
data['Hepatitis B']=imputer.fit_transform(data[['Hepatitis B']])
data[' BMI ']=imputer.fit_transform(data[[' BMI ']])
data['Polio']=imputer.fit_transform(data[['Polio']])
data['Total expenditure']=imputer.fit_transform(data[['Total expenditure']])
data['Diphtheria ']=imputer.fit_transform(data[['Diphtheria ']])
data['GDP']=imputer.fit_transform(data[['GDP']])
data['Population']=imputer.fit_transform(data[['Population']])
data[' thinness  1-19 years']=imputer.fit_transform(data[[' thinness  1-19 yea
data[' thinness 5-9 years']=imputer.fit_transform(data[[' thinness 5-9 years']
data['Income composition of resources']=imputer.fit_transform(data[['Income co
data['Schooling']=imputer.fit_transform(data[['Schooling']])


# Looking for null value in the data after fitting
df.isnull().sum()
```

> Data exploration on the dataset, and present a representative set of figures:

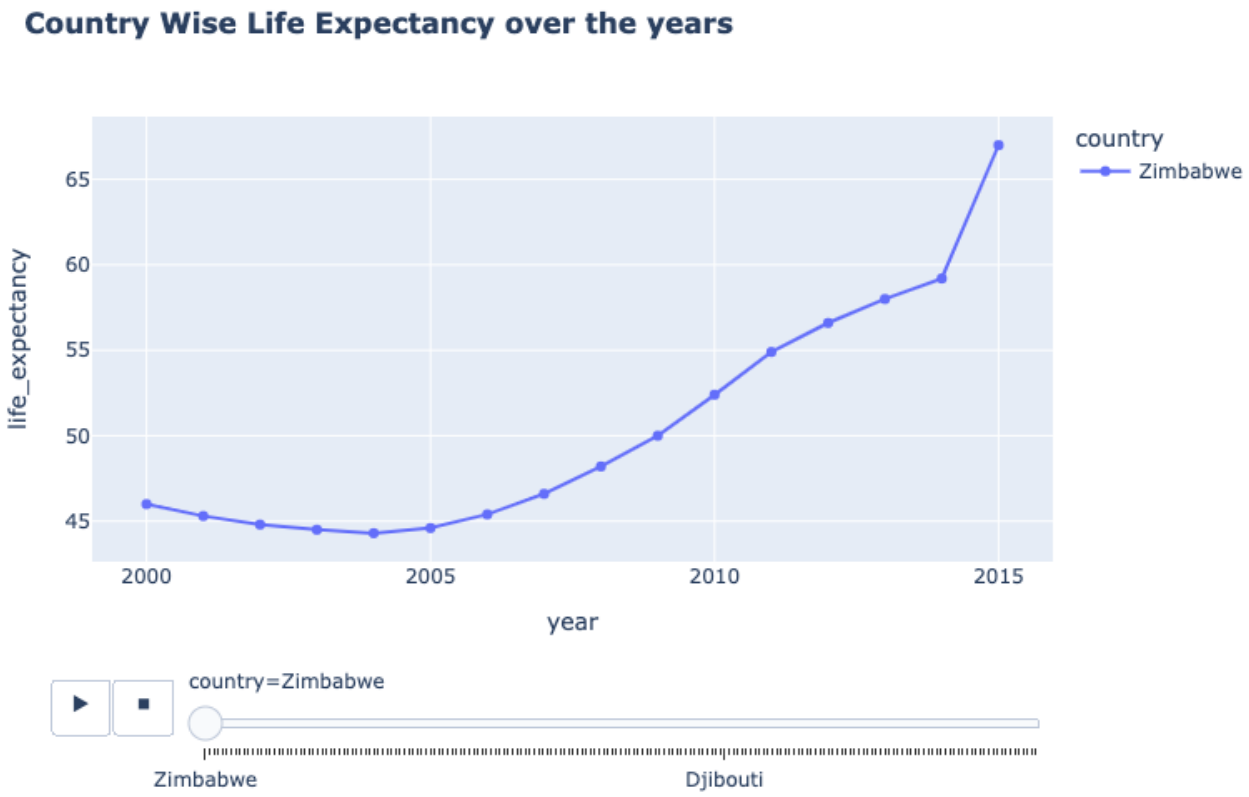
1. Distribution of Life Expectancy according to the age - can be interpreted as the life expectancy is high between the age of 70 to 75 all over the world



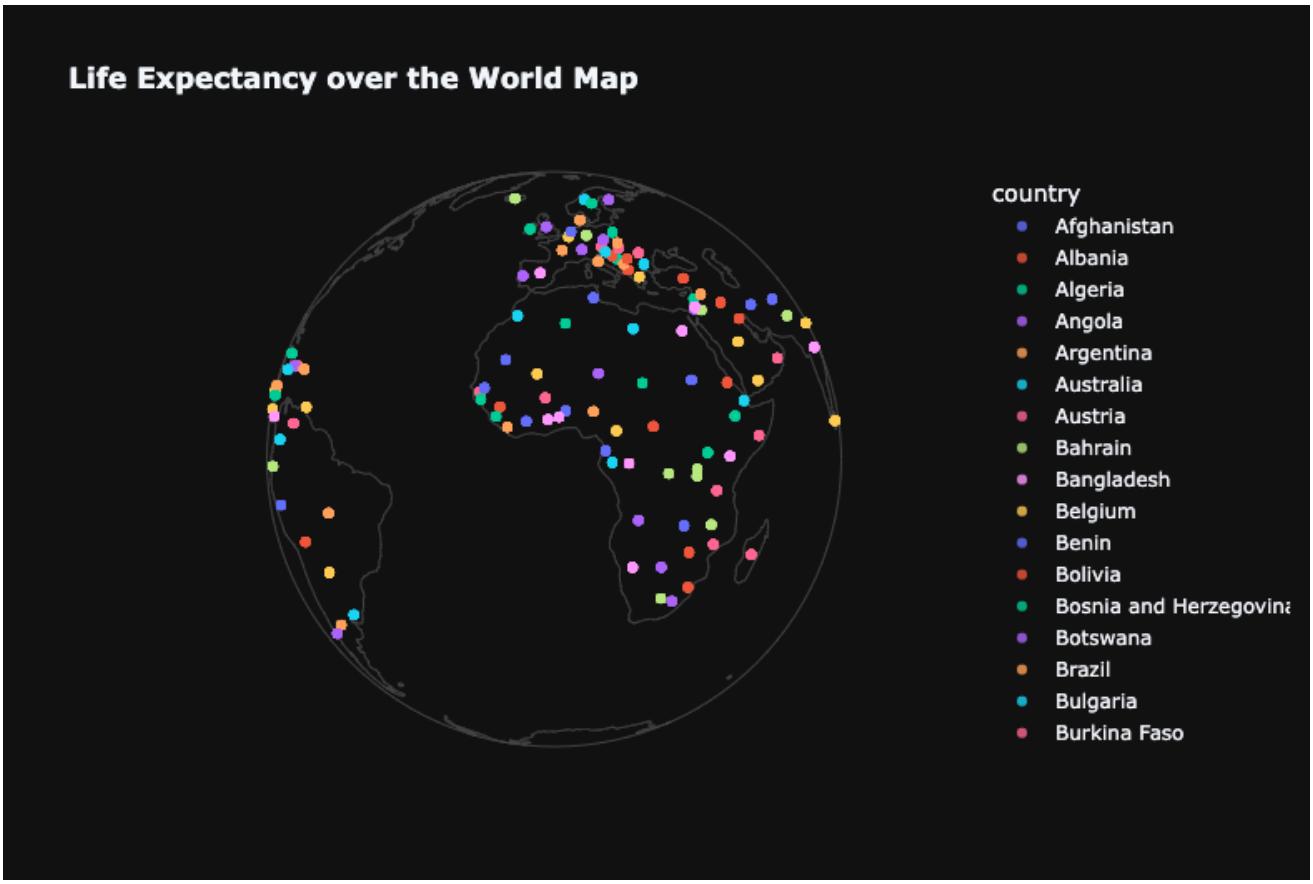
2. Comparing the life expectancy of Developing and Developed Countries - Developing countries have low life expectancy and the developed countries have high life expectancy all over the world



3. Country Wise Life Expectancy over the years



4. (for fun) Life Expectancy over the World Map



Note: there are many more types of figures that could be done for each variable of the data, however, picking the ones above allows us to focus on the main variables of the dataset. Furthermore, playing around with the map figure would give a good representation of the data.

Acknowledgements The data was collected from WHO and United Nations website with the help of Deeksha Russell and Duan Wang.

Refrences (Kumar R.(2017).Life Expectancy(WHO).from https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who))

Assignment 3

November 3, 2022

###Assignment 3

##1

[1]:
```
# import modules
import requests
import xml.dom.minidom as m
import xml.etree.ElementTree as et
import json
import time
```

[2]:
```
def get_id(disease):
    r = requests.get(f"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.
 ↪fcgi?db=pubmed&term={disease}+AND+2022[pdat]&retmax=1000&retmode=xml")
    time.sleep(1)
    doc = m.parseString(r.text)
    PubmedId = doc.getElementsByTagName('Id')
    IdList = []
    for i in range(len(PubmedId)):
        IdList.append(PubmedId[i].firstChild.data)

    return IdList
```

[3]:
```
len(get_id("Alzheimers") + get_id("Cancer"))
```

[3]:
```
2000
```

Finding an overlap between two sets of papers

[4]:
```
def overlap_papers(disease1,disease2):
    IdList1 = get_id(disease1)
    IdList2 = get_id(disease2)
    set1 = set(IdList1)
    set2 = set(IdList2)
    overlap = list(set1&set2)
    if len(overlap) == 0:
        print("There is no overlap in the two sets of papers")
    elif len(overlap) == 1:
```

```
        print(f"There is a overlap in the two sets of papers, the Pubmed Id is␣
↪{overlap[0]}")
        return overlap[0]
    else:
        print(f"There are overlaps in the two sets of papers, the Pubmed Ids␣
↪are{overlap}")
        return overlap
```

[5]:
```
overlap_papers('Alzheimers','cancer')
```

```
There are overlaps in the two sets of papers, the Pubmed Ids are['36321363',
'36321615']
```

[5]:
```
['36321363', '36321615']
```

Finding the Metadata of the papers in Alzheimers and Cancer sets

[6]:
```
def find_metadata(disease):
    PubmedIdList = get_id(disease)
    disease_dictionary = {}
    for PubmedId in PubmedIdList:
        time.sleep(1)
        r = requests.post(f"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/
↪efetch.fcgi?db=pubmed&retmode=xml&id={int(PubmedId)}")
        doc = m.parseString(r.text)

        ArticleTitle = doc.getElementsByTagName('ArticleTitle')
        Title = ""
        if len(ArticleTitle) > 0:
            for elm in ArticleTitle:
                for textmessage in elm.childNodes:
                    try:
                        Title += textmessage._get_wholeText()
                        # reference: https://docs.python.org/3/tutorial/errors.
↪html
                        Title = et.tostring(Title, method = "text").decode()

                    except AttributeError:
                        for subnode in textmessage.childNodes:
                            if subnode.nodeType == m.Node.TEXT_NODE:
                                Title += subnode.data

        AbstractText = doc.getElementsByTagName('AbstractText')
        Abstract = ""
        if len(AbstractText) > 0:
            for elm in AbstractText:
                for textmessage in elm.childNodes:
                    try:
```

```
                        Abstract += textmessage._get_wholeText()
                        Abstract = et.tostring(Abstract, method = "text").
↪decode()
                    except AttributeError:
                        for subnode in textmessage.childNodes:
                            if subnode.nodeType == m.Node.TEXT_NODE:
                                Abstract += subnode.data


        MeshHeading = doc.getElementsByTagName('MeshHeading')
        ArticleMeshTerms = []
        if len(MeshHeading) > 0:
            try:
                for i in MeshHeading:
                    ArticleMeshTerms.append(i.firstChild.childNodes[0].
↪nodeValue)
            except AttributeError: pass

        disease_dictionary[PubmedId] = {
            'ArticleTitle': Title,
            'ArticleAbstract': Abstract,
            'Query': disease,
            'Mesh': ArticleMeshTerms
        }

    return  disease_dictionary
```

Saving JSON files separatly of Alzheimers & Cancer

[ ]:
```
alz_data = find_metadata('Alzheimers')
cancer_data = find_metadata('cancer')

# alz_data into a JSON file paper.json.
with open('alzheimers.json','w') as f:
    json.dump(alz_data,f)

# cancer_data into a JSON file paper.json.
with open('cancer.json', 'w') as f:
    json.dump(cancer_data, f)
```

[ ]:
```
alz_data
```

[ ]:
```
cancer_data
```

[ ]:
```
both_papers_data = find_metadata('Alzheimers')
cancer_data = find_metadata('cancer')
both_papers_data.update(cancer_data)
```

```
with open('combined.json','w') as f:
    json.dump(both_papers_data,f)
```

[ ]:
```
len(both_papers_data)
```

##2

[ ]:
```
conda install pytorch torchvision -c pytorch
```

[ ]:
```
pip install transformers
```

[7]:
```
from transformers import AutoTokenizer, AutoModel
# load model and tokenizer
tokenizer = AutoTokenizer.from_pretrained('allenai/specter')
model = AutoModel.from_pretrained('allenai/specter')
```

[ ]:
```
import numpy as np
import json
import pandas as pd
```

[ ]:
```
with open("alzheimers.json") as f:
    alz_meta = json.load(f)

alz_data_format = pd.DataFrame.from_dict(alz_meta, orient = 'index')
alz_data_format.head()
```

[ ]:
```
with open("cancer.json") as f:
    cancer_meta = json.load(f)

cancer_data_frame = pd.DataFrame.from_dict(cancer_meta, orient = 'index')
cancer_data_frame.head()
```

[ ]:
```
both_papers = pd.concat([alz_data_format,cancer_data_frame])
```

[ ]:
```
both_papers
```

[ ]:
```
# read JSON file using the open function.
with open('combined.json') as f:
    both_papers_data = json.load(f)
```

[ ]:
```
data = [paper["ArticleTitle"] + tokenizer.sep_token + paper["ArticleAbstract"]␣
 ↪for paper in both_papers_data.values()]
inputs = tokenizer([data[0]], padding=True, truncation=True,␣
 ↪return_tensors="pt", max_length=512)
result = model(**inputs)
embed_total = result.last_hidden_state[:, 0, :].detach().numpy()
```

```
for i in range(1,len(data)):
    inputs = tokenizer([data[i]], padding=True, truncation=True,␣
 ↪return_tensors="pt", max_length=512)
    result = model(**inputs)
    embed = result.last_hidden_state[:, 0, :].detach().numpy()
    embed_total = np.concatenate((embed_total, embed),axis = 0)
```

[ ]:
```
import pandas as pd
from sklearn import decomposition
```

[ ]:
```
# first 3 principal components.
pca = decomposition.PCA(n_components=3)
embed_pca = pd.DataFrame(
    pca.fit_transform(embed_total),
    columns=['PC0', 'PC1', 'PC2']
)
embed_pca["Query"] = [paper["Query"] for paper in both_papers_data.values()]
```

[ ]:
```
embed_pca
```

[ ]:
```
import plotnine as p9
```

[ ]:
```
#PC0 vs PC1
(p9.ggplot(data = embed_pca, mapping = p9.aes(x='PC0', y='PC1'))
+ p9.geom_point(p9.aes(x = 'PC0', y = 'PC1', color = 'Query'))
+ p9.labs(title = "PC0 vs PC1"))
```

[ ]:
```
# PC0 vs PC2
(p9.ggplot(data = embed_pca, mapping = p9.aes(x='PC0', y='PC2'))
+ p9.geom_point(p9.aes(x = 'PC0', y = 'PC2', color = 'Query'))
+ p9.labs(title = "PC0 vs PC2"))
```

[ ]:
```
#PC1 vs PC2
(p9.ggplot(data = embed_pca, mapping = p9.aes(x='PC1', y='PC2'))
+ p9.geom_point(p9.aes(x = 'PC1', y = 'PC2', color = 'Query'))
+ p9.labs(title = "PC1 vs PC2"))
```

##3

[ ]:
```
import matplotlib.pyplot as plt
def plot_with_explicit_Eulers(s0,i0,r0,B,g, tMax):

    #Initialize parameters and arrays
    numSteps=1000
    tStep=tMax/numSteps
    N=s0+i0+r0
    t=[0]*(numSteps+1)
```

```
    s=[0]*(numSteps+1)
    i=[0]*(numSteps+1)
    t[0]=0
    s[0]=s0
    i[0]=i0
    peak_not_reached=False

    #Calculate i over time range
    for j in range(1,numSteps+1):
        t[j]=t[j-1]+tStep
        s[j]=s[j-1]+(tStep)*(-B/N*s[j-1]*i[j-1])
        i[j]=i[j-1]+(tStep)*(B/N*s[j-1]*i[j-1]-g*i[j-1])
        #Check if peak reached
        if (i[j]<i[j-1] and not peak_not_reached):
            peak_day=t[j-1] #round(t[j-1]) to the nearest integer/day
            peak_infections=round(i[j-1])
            peak_not_reached=True

    plt.plot(t,i,label="Infected People")
    return peak_day,peak_infections
```

[ ]:
```
[peak_day,peak_infections]=plot_with_explicit_Eulers(13399,1,0,2,1,30)
print("Peak day: ", peak_day)
print("Peak infections: ", peak_infections)
```

[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
Bs=[]
gs=[]
peak_days=[]
peak_infections=[]

for B in np.arange(1.8,2.2,0.05):
    for g in np.arange(0.9,1.1,0.02):
        [peak_day,peak_infection]=plot_with_explicit_Eulers(13399,1,0,B,g,100)
        Bs.append(B)
        gs.append(g)
        peak_days.append(peak_day)
        peak_infections.append(peak_infection)
```

[ ]:
```
dicts={"B":Bs,"g":gs,"peak_days":peak_days,"peak_infections":peak_infections}
data=pd.DataFrame(dicts)
```

[ ]:
```
#infection time in days
sns.set()
sns.heatmap(data.pivot("B","g","peak_days"))
```

```
plt.title("Heatmap of peak infection time in days as a function of Beta and␣
 ↪gamma")
```

[ ]:
```
#infected people in days
sns.set()
sns.heatmap(data.pivot("B","g","peak_infections"))
plt.title("Heatmap of peak number of infected people in days as a function of␣
 ↪Beta and gamma")
```

##4

[ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
import datetime as dt
import warnings
```

[ ]:
```
df = pd.read_csv('/Users/polina/Desktop/Life Expectancy Data.csv')
df
```

[ ]:
```
df.info()
```

[ ]:
```
df.shape
```

[ ]:
```
df.describe()
```

[ ]:
```
# Looking for null value in the data
df.isnull().sum()
```

[ ]:
```
# Replacing the Null Values with mean values of the data
from sklearn.impute import SimpleImputer
#reference: https://scikit-learn.org/stable/modules/generated/sklearn.impute.
 ↪SimpleImputer.html
imputer=SimpleImputer(missing_values=np.nan,strategy='mean',fill_value=None)
df['Life expectancy ']=imputer.fit_transform(df[['Life expectancy ']])
df['Adult Mortality']=imputer.fit_transform(df[['Adult Mortality']])
df['Alcohol']=imputer.fit_transform(df[['Alcohol']])
df['Hepatitis B']=imputer.fit_transform(df[['Hepatitis B']])
df[' BMI ']=imputer.fit_transform(df[[' BMI ']])
df['Polio']=imputer.fit_transform(df[['Polio']])
df['Total expenditure']=imputer.fit_transform(df[['Total expenditure']])
df['Diphtheria ']=imputer.fit_transform(df[['Diphtheria ']])
df['GDP']=imputer.fit_transform(df[['GDP']])
df['Population']=imputer.fit_transform(df[['Population']])
df[' thinness  1-19 years']=imputer.fit_transform(df[[' thinness  1-19 years']])
```

```
df[' thinness 5-9 years']=imputer.fit_transform(df[[' thinness 5-9 years']])
df['Income composition of resources']=imputer.fit_transform(df[['Income␣
 ↪composition of resources']])
df['Schooling']=imputer.fit_transform(df[['Schooling']])
```

[ ]:
```
# Looking for null value in the data after fitting
df.isnull().sum()
```

[ ]:
```
# Changing/Renaming the columns for easy access.
df = df.rename(columns={'Country': 'country', 'Year': 'year', 'Status':␣
 ↪'status', 'Life expectancy ': 'life_expectancy', 'Adult Mortality':␣
 ↪'adult_mortality',
       'infant deaths':'infant_death', 'Alcohol':'alcohol', 'percentage␣
 ↪expenditure': 'percentage_expenditure', 'Hepatitis B':'Hepatitis_b',
       'Measles ':'measles', ' BMI ':'bmi', 'under-five deaths ':
 ↪'under_five_deaths', 'Polio':'polio', 'Total expenditure':␣
 ↪'total_expenditure','Diphtheria ':'diphtheria', ' HIV/AIDS':'hiv_Aids',␣
 ↪'GDP':'gdp', 'Population':'population',
       ' thinness  1-19 years':'thinness_1_to_19', ' thinness 5-9 years':
 ↪'thinness_5_to_9',
       'Income composition of resources':'income_composition_of_resources',␣
 ↪'Schooling': 'schooling'})
```

[ ]:
```
# Looking for columns after rename
df.columns
```

[ ]:
```
#Distribution of Life Expectancy according to the age
fig = px.histogram(df, x = 'life_expectancy')
fig.show()
```

[ ]:
```
#Comparing the life expectancy of Developing and Developed Countries
fig = px.violin(df, x= 'status', y= 'life_expectancy',
                color = 'status',box = True,title='Life Expectancy on the Basis␣
 ↪of Country Status')
fig.show()
```

[ ]:
```
#Country Wise Life Expectancy over the years
fig = px.line((df.sort_values(by = 'year')), x = 'year', y = 'life_expectancy',
    animation_frame= 'country',animation_group='year',color='country',
    markers=True,title='<b>Country Wise Life Expectancy over the years')
fig.show()
```

[ ]:
```
country_df = px.data.gapminder()
country_df.tail()
```

[ ]:
```
#Life Expectancy over the World Map
```

```
map_fig = px.scatter_geo(country_df,locations = 'iso_alpha', projection =␣
 ↪'orthographic',
                          opacity = 0.8, color = 'country', hover_name =␣
 ↪'country',
                          hover_data = ['lifeExp', 'year'],template =␣
 ↪'plotly_dark',title = '<b>Life Expectancy over the World Map')
map_fig.show()
```

Refrences (Kumar R.(2017).Life Expectancy(WHO).from https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who))