



 table.png	K	2 days ago
 top10_A.png	K	2 days ago
 top10_C.png	K	2 days ago
 year_b.png	k	2 days ago
 year_l.png	k	2 days ago

BIS634 Yuechen Liu HW3

Exercise 1

Use the requests module (or urllib) to use the Entrez API (see slides8) to identify the PubMed IDs for 1000 Alzheimers papers from 2019 and for 1000 cancer papers from 2019.

Answer: Citation: <https://stackoverflow.com/questions/317413/get-element-value-with-minidom-with-python> I create a function called `get_id(disease)`; when input the disease name, say, Cancer, the function will grab the id of Cancer paper, and add it to a list by for loop. For test whether the function works, I get the length of the two lists, and the answer is 2000.

There are of course many more papers of each category, but is there any overlap in the two sets of papers that you

⋮ Readme.md



... I create a function called `overlap(disease1, disease2)`, when input the two disease names, we can check if the two disease papers have overlap by checking if they have duplicate ids. I use 'set' to check it, and get the result is '32501203'.

Use the Entrez API via requests/urllib to pull the metadata for each such paper found above (both cancer and Alzheimers)

(and save a JSON file storing each paper's title, abstract, MeSH terms (DescriptorName inside of MeshHeading), and the query that found it that is of the general form

Answer: Citation: <https://www.geeksforgeeks.org/python-ways-to-create-a-dictionary-of-lists/> <https://docs.python.org/3/tutorial/errors.html>

I create a function called `get_info(disease1,disease2)` in order to combine two dictionaries into one for better use in the future. I met problems when adding paper information into the json file, such as there is missing value in title, so I use `if & else` to catch the issue to make sure that the json file can be finished. Finally, I save the json file called `pb2000.json`. This function works because I check the json file with `'find'` to locate some specific content, such as the first, last, and middle papers by finding their IDs, and all papers show in a right format.

Some papers like 32008517 (Links to an external site.) have multiple AbstractText fields (e.g. when the abstract is structured). Be sure to store all parts. You could do this in many ways, from using a dictionary or a list or simply concatenating with a space in between. Discuss any pros or cons of your choice in your readme

Answer: I use for loop to go through all child nodes, and if the child is `None`, the function will print "someone is missing abstract"; if it is not null, the function will add the information to the "abstract". I think the advantage may be in this way, the function can avoid possible attribute errors, so the function can process smoothly regardless the format of the abstract; the potential disadvantage may be it will cost more time to check if there is missing abstract.

Exercise 2

What fraction of the Alzheimer's papers have no MeSH terms?

Answer: I create a function called `mesh_fraction(disease)` to calculate the fraction of the disease's papers having no MeSH terms. The function firstly counts the number of empty MeSH papers, and divided it by the total number of papers. When the disease is ALZHEIMERS, the result is 0.165, which means that there are 16.5% of Alzheimer's papers without MeSH terms. To be honest, I don't know how to prove my function works, so I checked the answer with classmates, and we got similar answers.

What fraction of the cancer papers have no MeSH terms?

Answer: The answer is 0.759, which means that there are 75.9% of Cancer's papers without MeSH terms.

Comment on how the fractions compare. (1 point; i.e. if they're essentially the same, do you think that's a coincidence? If they're different, do you have any theories why?)

Answer: The two results are totally different. I think the one potential reason is that because cancer is a really broad term, which it has many subterms such as breast cancer, stomach cancer, etc.. Compared with cancer, Alzheimer is a more narrow term, which maybe the reason of the missing MeSH terms percentage of Alzheimer lower than Cancer.

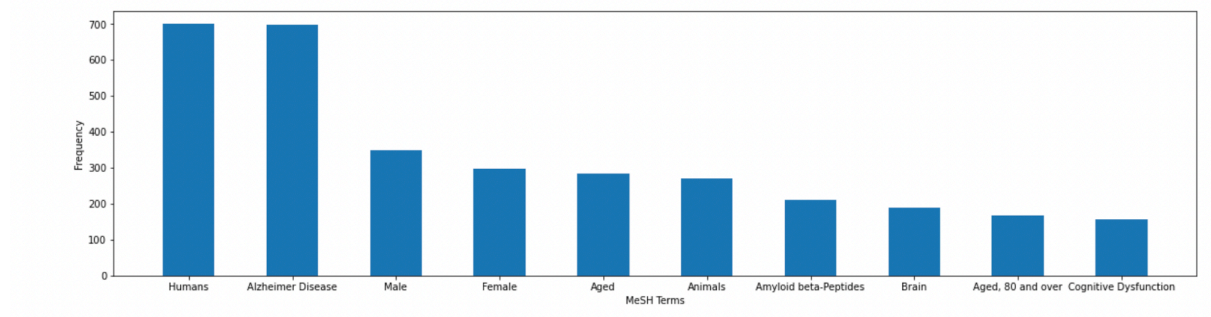
<https://www.definitivehc.com/resources/glossary/medical-subject-headings-mesh>

What are the 10 most common MeSH terms for the Alzheimer's papers whose metadata you found in Exercise 1?

Answer: Citation: [#https://stackoverflow.com/questions/613183/how-do-i-sort-a-dictionary-by-value](https://stackoverflow.com/questions/613183/how-do-i-sort-a-dictionary-by-value) I create function called `mesh_count(disease)` to count how many MeSH headings(terms) in each disease, and for Alzheimer, the function returns a list of MeSH terms with the count of each term. Then I create a function called `first_ten_items(disease)` to get the first ten common MeSH terms for Alzheimer papers. The top ten are: Humans, Alzheimer Disease, Male, Female, Aged, Animals, Amyloid beta-Peptides, Brain, Aged, 80 and over and Cognitive Dysfunction. I used the first function to test my second functions; I manually find the top terms in total MeSH terms based on my function `mesh_count`, and the result is correct.

Provide a graphic illustrating their relative frequency.

Answer:

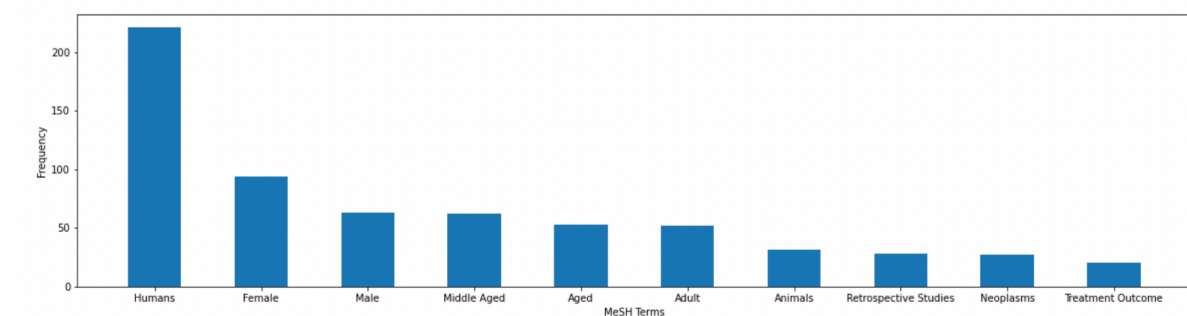


What are the 10 most common MeSH terms for the cancer papers whose metadata you found in Exercise 1?

Answer: I use `mesh_count(disease)` and `first_ten_items(disease)` to get the first ten common MeSH terms for Cancer papers. The top ten are Humans, Female, Male, Middle Aged, Aged, Adult, Animals, Retrospective Studies, Neoplasms, Treatment Outcome.

Provide a graphic illustrating their relative frequency.

Answer:



Make a labeled table with rows for each of the top 5 MeSH terms from the Alzheimer's query and columns for each of the top 5 MeSH terms from the cancer query. For the values in the table, provide the count of papers (combined, from both sets) having both the matching MeSH terms.

Answer: Citation: #<https://towardsdatascience.com/how-to-easily-create-tables-in-python-2eaea447d8fd> In order to get top 5 MeSH terms, I change the above function a little bit to create a new one called first_5(disease). Then, for the table, I first create a matrix, and for loop the count of top five Mesh Terms paper in to the matrix. Finally, I create a table based on the matrix.

Alzheimer Disease/Cancer	Humans	Female	Male	Middle Aged	Aged
Humans	922	381	208	369	338
Alzheimer Disease	584	236	109	281	227
Male	369	327	180	413	288
Female	381	394	192	327	295
Aged	338	295	175	288	338

To test the funcation works, I compared the first five with first ten(from last question), and the result is the same.

Comment on any findings or limitations from the table and any ways you see to get a better understanding of how the various MeSH terms relate to each other.

Answer: I think the table has limitations to see the relationship among MeSH terms. The table only shows the numbers, which is not straightforward to actually see the relation; while other ways, such as graphs, may be much more easier for researchers to understand how the various MeSH terms relate to each other.

Exercise 3

In particular, for each paper identified from exercise 1, compute the SPECTER embedding (a 768-dimensional vector). Keep track of which papers came from searching for Alzheimers, which came from searching for cancer.

Answer: Firstly, I find the index of overlap paper, which is 885; then, I set the query of the overlap paper as 'BOTH'. I utilize SPECTER model to process my dictionary of papers, and as we can see from the below graph:

```
embeddings_pca.loc[885, "Query"]
```

```
' BOTH '
```

```
embeddings_pca.loc[999, "Query"]
```

```
' CANCER '
```

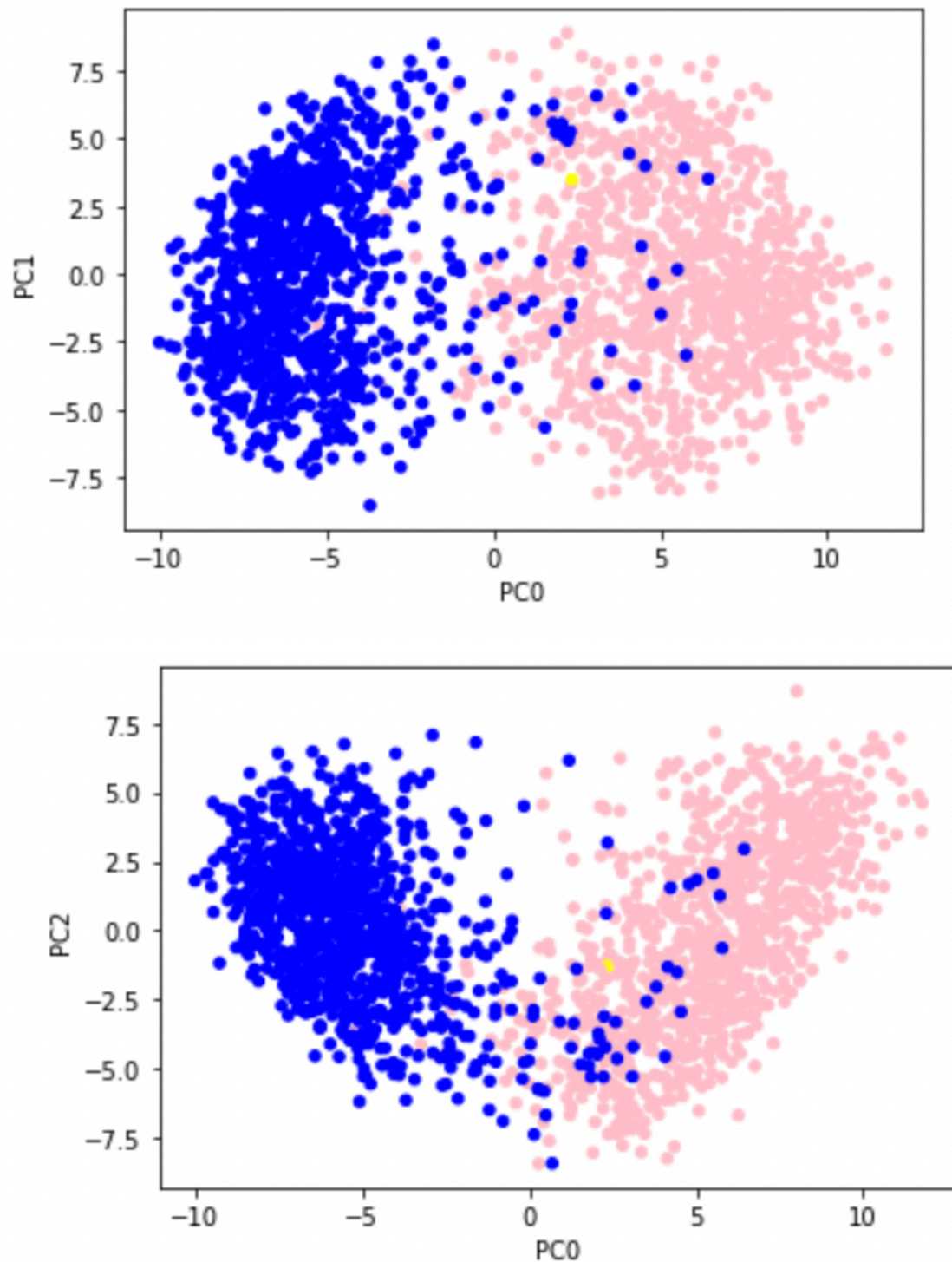
```
embeddings_pca.loc[1000, "Query"]
```

```
' ALZHEIMERS '
```

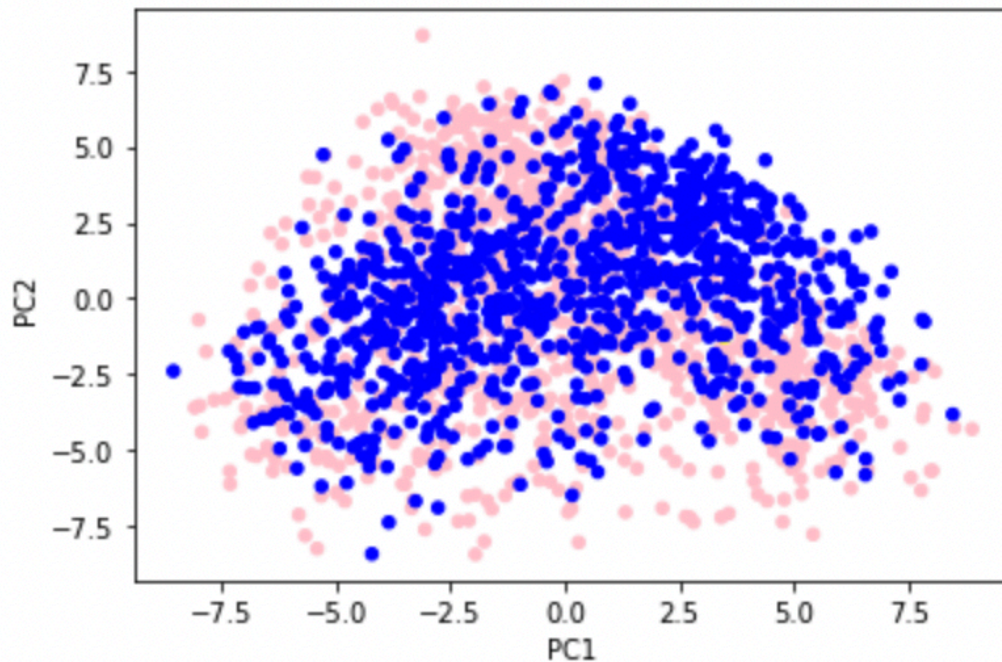
The overlap paper have index of 885; from index 0-884 and 886-999 are Cancer, and 1000-1998 are Alzheimers, based on the fact that there are total 2000 papers, 1000 for Cancer, 1000 for Alzheimers.

Plot 2D scatter plots for PC0 vs PC1, PC0 vs PC2, and PC1 vs PC2; color code these by the search query used (Alzheimers vs cancer). (3 points) Comment on the separation or lack thereof, and any take-aways from that. (2 points)

PC0 vs PC1



According to the definition of PCA, we can see that the variation (separation) is obviously, which means that it PCA holds the variance of data and we can access more information of the by looking at the dimension PC0 vs PC1. Same as to the dimension of PC0 vs PC2, we can see the great variance of data, which we can get information from it.

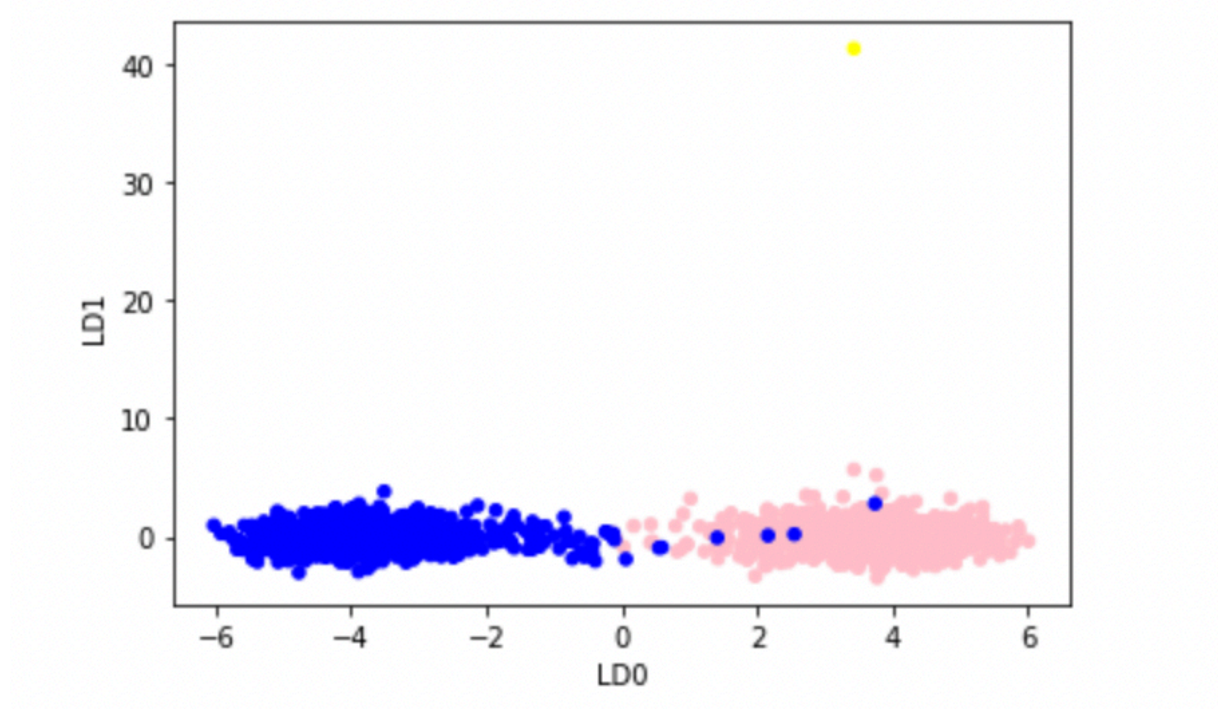


But for dimension of PC1 vs PC2, there are many overlaps, and the variance is not obvious, thus we will get less information about the data from this dimension. According to

<https://www.datacamp.com/community/tutorials/principal-component-analysis-in-python>

Now look at the distribution with the LDA projection. Note that if you have n categories (presumably 2 or maybe 3 for you), LDA will give at most $n-1$ reduced dimensions... so graphically show the LDA projection results in the way that you feel best captures the distribution. Comment on your choice, things you didn't chose and why, and any other differences about what you saw with PCA vs LDA.

Answer:



From the graph we can see that LDA is to gather the subgroup and maximizes the separability between different groups. As a result, we can see that the data of Cancer is more concentrated together; the data of Alzheimer is more concentrated together; but two clusters are separate, and there is a data point, 'BOTH', which is even more far away. However, PCA is finding the maximal variance of data, so in the graphs of PCA, we can see many overlaps. I will choose different analysis ways based on what I need for my research.

Exercise 4

Describe in words how you would parallelize this algorithm to work with two processes.

Answer: Original data: 14, 1, 18, 2, 16, 4, 7, 21 STEP1: Assign to two processors P0 and P1: P0: 14, 1, 18, 2 P1: 16, 4, 7, 21 Step2: P0 and P1 quickly sort the two sets of data: P0: 1, 2, 14, 18 P1: 4,7,16,21 Step3: Compare the smallest number in P0 with the smallest number in P1, 1 in P0 is smaller than 4 in P1, and then proceed to the next step, continue to compare the second number,2, in P0 with 4 in P1, and set the smallest number in P1 using insertion sort in P0, and P1 is inserted between 2 and 14 of P0. At this time, there are: P0: 1,2,4,14,18 P1: 7,16,21 Step4: Insert the 7 of P1 from the 4 of P0 to the back to sort, recursively, and finally we have: P0: 1, 2, 4, 7, 14, 16, 18, 21 After sorting, P1 is empty.

Original link: https://blog.csdn.net/xudong_98/article/details/51622219

How you would validate the results and the speedup?

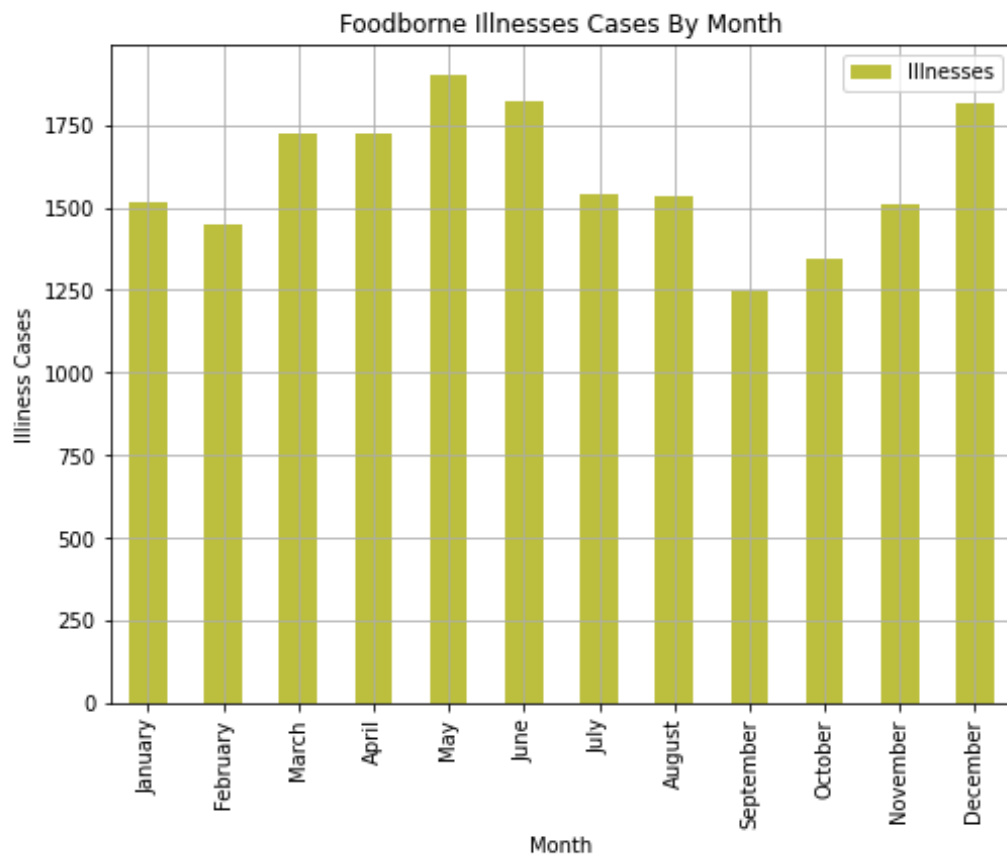
Answer: For validate the result, we can use both merge sort and parallel merging sort to process one group of data, to test if they can have the same result. For the speedup, we can import time module in python, to get the working time of both algorithms. Normally, parallel merging sort will be faster. Parallel merging algorithm is very effective in processing large amounts of data, but the performance of traditional merging algorithm is very poor, because each time the merging process, the processor will be reduced by half, aggravating the data length processed by each processor. When merging to the end, only one processor processes all the data, which is very slow. As a result, the merge sort algorithm must be parallelize to achieve the maximum advantages.

Exercise 5

Do data exploration on the dataset you identified in exercise 4 in the last homework, and present a representative set of figures that gives insight into the data. Comment on the insights gained.

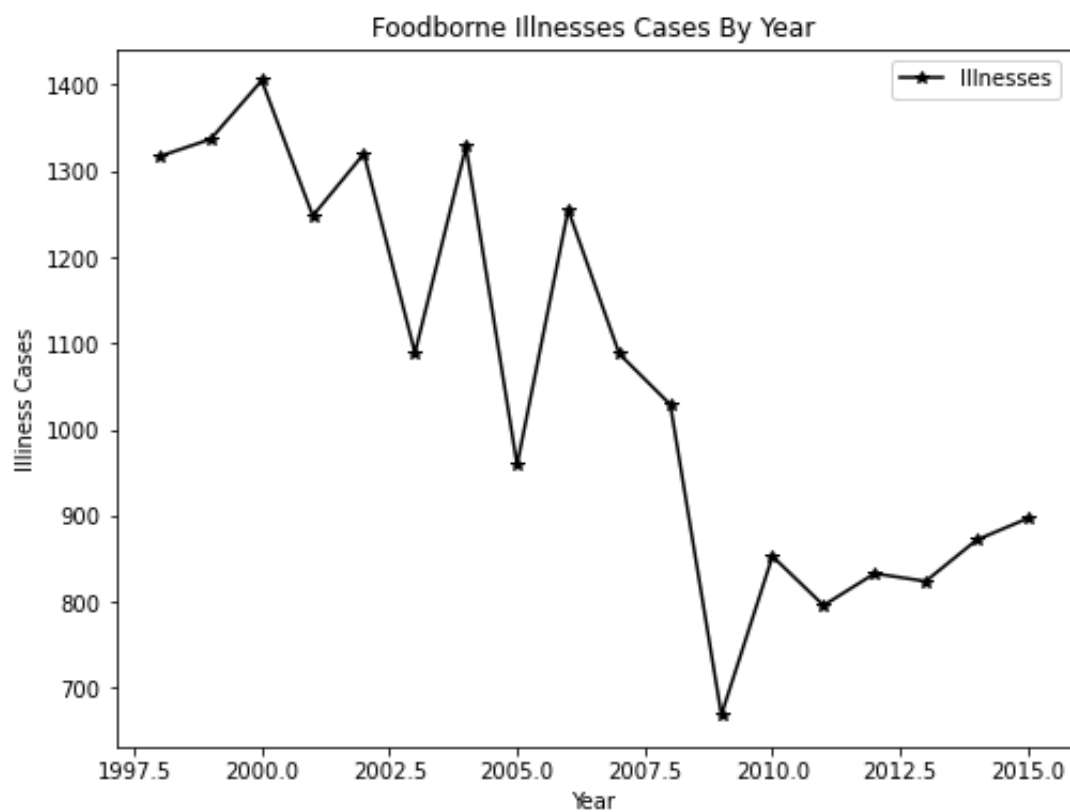
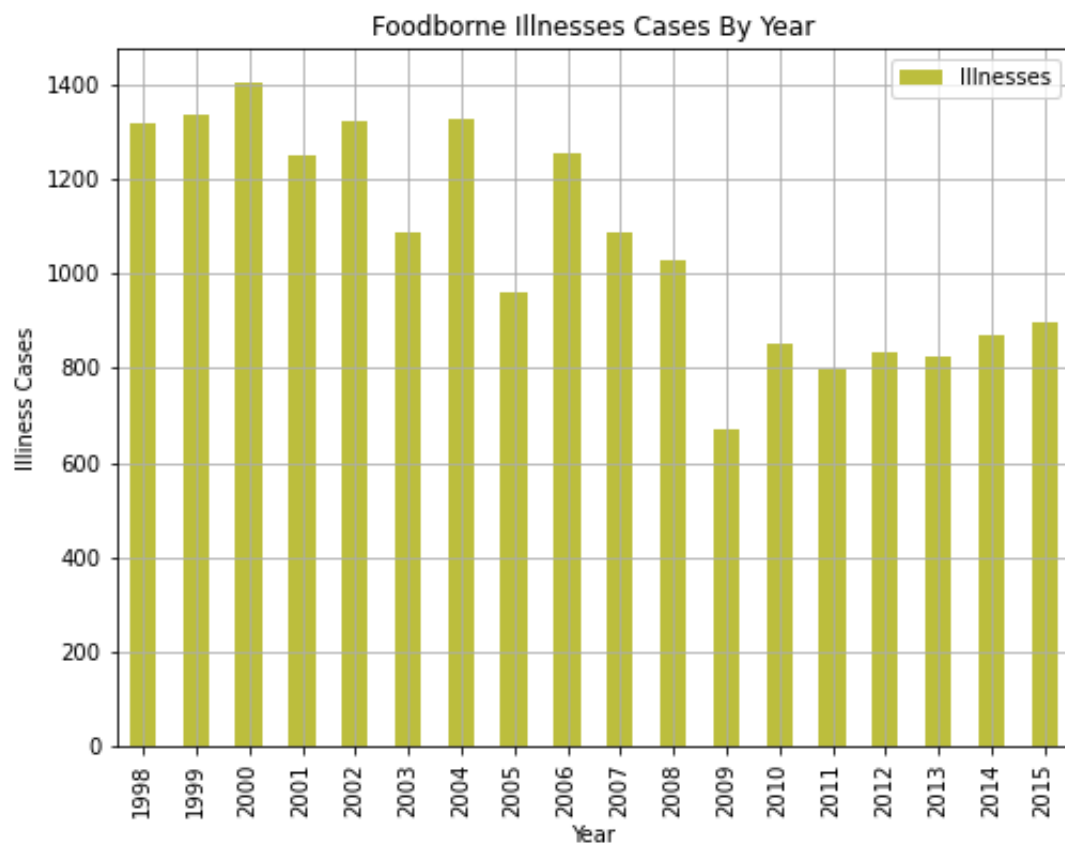
Answer:

1. Which month does foodborne tend to occur the most?



As the graph showing, foodborne tends to occur most in May, June, and December.

2. Has foodborne disease outbreaks gotten worse over the years?

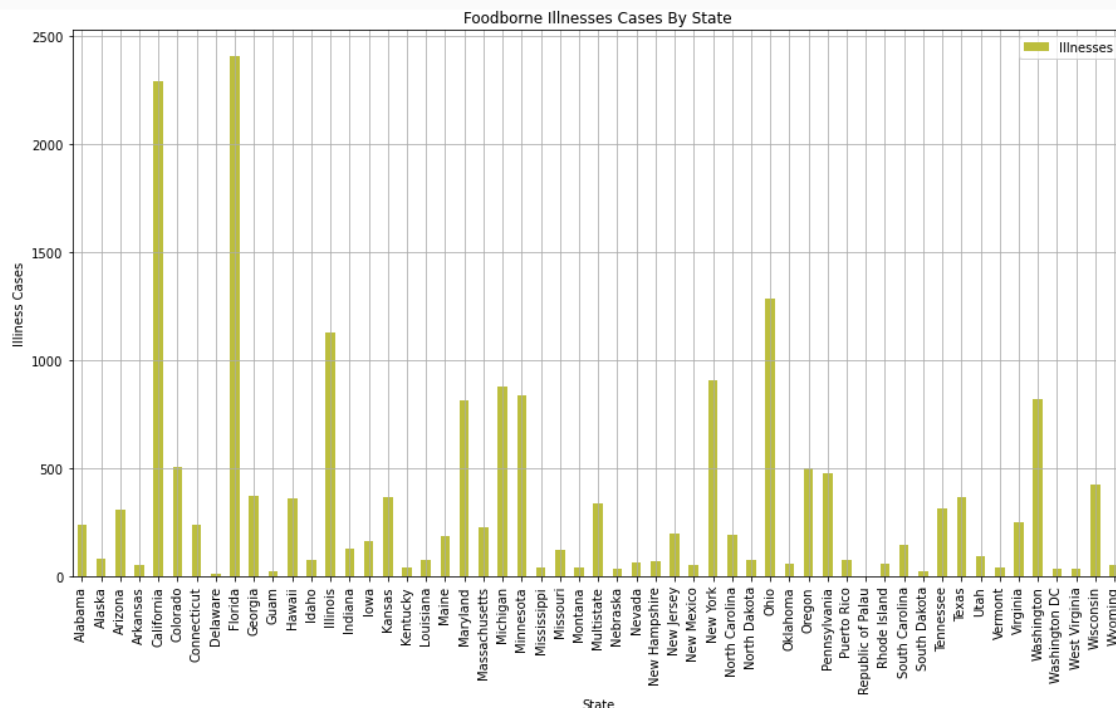


As

the graph showing, the trend of foodborne illnesses has decreased overall since 1998. There are some significant drops, one is from 2004 - 2005, the

other is from 2008-2009, but now seems to be steady around 800 - 900 cases per year. The data can be concluded that foodborne diseases are occurring less frequently by years.

3. Which state of the US do they appear the most frequently?



Clearly, California and Florida have extremely high rate of foodborne illnesses. The possible reasons could be related to their population rate during the years of 1998-2015. An exceptional circumstance would be the outbreak of Salmonella Infections linked to Peanut Butter: at least 36 people across 17 states have been affected by a salmonella outbreak between May 9 and July 27, 2008 - 2009. Majority of peanut butter was sent to California and Ohio, according to <https://www.healthline.com/health-news/salmonella-outbreak-in-17-states-linked-to-italian-meats-what-to-know#:~:text=At%20least%2036%20people%20across,abdominal%20pain%2C%20and%20usually%20diarrhea.>

Are there any missing values in your data? (Whether the answer is yes or no, include in your answer the code you used to come to this conclusion.) If so, quantify. Do they appear to be MAR/MCAR/MNAR? Explain.


```
outbreak_df.isnull().sum()
```

```
Year          0
Month         0
State         0
Location     2166
Food         8963
Ingredient   17243
Species      6619
Serotype/Genotype 15212
Status       6619
Illnesses    0
Hospitalizations 3625
Fatalities   3601
dtype: int64
```

Answer:

```
missing_value = outbreak_df.isnull().sum()*100/len(outbreak_df)
missing_value
```

```
Year          0.000000
Month         0.000000
State         0.000000
Location     11.329044
Food         46.880067
Ingredient   90.187771
Species      34.620012
Serotype/Genotype 79.564831
Status       34.620012
Illnesses    0.000000
Hospitalizations 18.960197
Fatalities   18.834667
dtype: float64
```

Clearly, the dataset has missing values. The data missing can be MNAR, for example, the restaurant is unwilling to report the location, food, ingredient and species; or the customers are not able to recall what food or ingredient caused the outbreaks. Besides, most foodborne infections go undiagnosed and unreported, because those people don't see the doctors, or the doctor doesn't make a specific diagnosis according to the CDC. The missing data also can be MAR, for example, the missing data of ingredient and species can be related to missing data of food; the missing data of fatalities can be related to missing data of hospitalizations.

Identify any data cleaning needs and write code to perform them. If the data does not need to be cleaned, explain how you reached this conclusion.

Answer: Filter the dataset, drop the columns that are not related to the topic. Since the ratio of null fields in the column 'Serotype/Genotype' and 'Ingredient' to the total number of rows exceeds 60%, which are proved not useful for the data analysis. Therefore I choose to drop those two columns from my table. Then, drop all null values.

```
df2.isnull().sum()
```

Year	0
Month	0
State	0
Location	0
Food	0
Species	0
Status	0
Illnesses	0
Hospitalizations	0
Fatalities	0

dtype: int64

The data cleansing is finish, and the data does need to be cleaned, because the null values and excessive null columns are dropped.

```
In [1]: import requests
import xml.dom.minidom as m
```

```
In [253]: #https://stackoverflow.com/questions/317413/get-element-value-with-minidom-with-python
```

```
In [4]: #Use the requests module (or urllib) to use the Entrez API (see slides8) to identify the PubMed IDs
#for 1000 Alzheimers papers from 2019 and for 1000 cancer papers from 2019. (9 points)
```

```
In [2]: def get_id(disease):
    r = requests.get(f"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term={disease}")
    doc = m.parseString(r.text)
    get_id = doc.getElementsByTagName("Id")
    #print(get_id)
    my_n_node = []
    for i in range(len(get_id)):
        my_n_node.append(get_id[i])
        i+=1
    my_child = []
    for j in range(len(my_n_node)):
        my_child.append(my_n_node[j].firstChild)
        j+=1
    my_text = []
    for k in range (len(my_child)):
        my_text.append(my_child[k].data)
        k+=1
    return my_text
```

```
In [3]: get_id("ALZHEIMERS")
```

```
Out[3]: ['34692095',  
        '34692000',  
        '33939349',  
        '33841007',  
        '33627920',  
        '33463291',  
        '33323224',  
        '33243028',  
        '33097841',  
        '32954270',  
        '32821704',  
        '32802097',  
        '32641955',  
        '32598641',  
        '32501203',  
        '32489952',  
        '32477473',  
        '32477472',  
        '32399472',  
        '32375155']
```

```
In [4]: get_id ("CANCER")
```

```
Out[4]: ['34702262',
        '34692085',
        '34691993',
        '34691992',
        '34691990',
        '34691989',
        '34691986',
        '34691933',
        '34691895',
        '34691351',
        '34590506',
        '34539049',
        '34539046',
        '34493369',
        '34460208',
        '34460207',
        '34460206',
        '34460205',
        '34460204',
        '34460203']
```

```
In [28]: len(get_id("ALZHEIMERS")+get_id("CANCER"))
```

```
Out[28]: 2000
```

```
In [84]: #There are of course many more papers of each category, but is there any overlap in
         #the two sets of papers that you identified?
```

```
In [29]: def overlap(disease1,disease2):
         return set(get_id(disease1))&set(get_id (disease2))
```



```
In [30]: overlap("ALZHEIMERS", "CANCER")
```

```
Out[30]: {'32501203'}
```

```
In [5]: import json
import time
```

```
In [10]: def get_info(disease1,disease2):
    pmid_list=get_id(disease1)+get_id(disease2)
    info_dic = {}
    #print(pmid_list)
    for id in pmid_list:
        time.sleep(1)
        r = requests.post(
            f"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed&retmode=xml&id={int(id)}")
        doc = m.parseString(r.text)

    #https://www.geeksforgeeks.org/python-ways-to-create-a-dictionary-of-lists/
    #https://docs.python.org/3/tutorial/errors.html
    article_titles = doc.getElementsByTagName("ArticleTitle")
    title = ""
    if len(article_titles)!=0:
        child = article_titles[0].firstChild
        if child is None:
            print("someone is missing a title")
        else:
            if child.nodeType == m.Node.TEXT_NODE:
                title+=child.data

    article_abstract = doc.getElementsByTagName("AbstractText")
    abstract = ""
    if len(article_abstract)!=0:
        child = article_abstract[0].firstChild
        if child is None:
            print("someone is missing abstract")
```

```

        print("someone is missing abstract")
    else:
        if child.nodeType == m.Node.TEXT_NODE:
            abstract+=child.data

    article_mesh = doc.getElementsByTagName("MeshHeading")
    mesh = []
    if len(article_mesh)>0:
        for i in article_mesh:
            if i.firstChild.childNodes[0].nodeValue is None:
                print("someone is missing mesh")
            else:
                mesh.append(i.firstChild.childNodes[0].nodeValue)

    query=[]
    if id in get_id(disease1):
        query.append(disease1)
    if id in get_id(disease2):
        query.append(disease2)

    info_dic[id]={"ArticleTitle":title,
                 "AbstractText":abstract,
                 "Query":query,
                 "MeshHeading":mesh,
                 }
    with open('pb2000.json', 'w') as outfile:
        json.dump(info_dic, outfile)

```

In [11]: `get_info("CANCER","ALZHEIMERS")`

someone is missing a title

In []: *#Q2.1What fraction of the Alzheimer's papers have no MeSH terms?*

```
In [12]: with open("pb2000.json", "r") as read_it:
         all_data = json.load(read_it)
```

```
In [13]: #total=len(all_data)
         def mesh_fraction(disease):
             count_paper=0
             count_mesh_empty = 0
             disease_list = get_id(disease)
             for i in disease_list:
                 if all_data[i]["MeshHeading"]==[]:
                     count_mesh_empty+=1
                 count_paper+=1
             mesh_fraction=count_mesh_empty/count_paper
             return mesh_fraction
```

```
In [14]: mesh_fraction("ALZHEIMERS")
```

```
Out[14]: 0.165
```

```
In [15]: #What fraction of the cancer papers have no MeSH terms?
```

```
In [16]: mesh_fraction("CANCER")
```

```
Out[16]: 0.759
```

```
In [17]: #What are the 10 most common MeSH terms for the Alzheimer's papers whose metadata you found in Exercise
```

```
In [27]: from collections import Counter
```

```
In [28]: def mesh_count(disease):  
    mesh = []  
    for i in all_data:  
        if all_data[i]["Query"]==[disease] or all_data[i]["Query"]==["ALZHEIMERS","CANCER"]:  
            mesh+=all_data[i]["MeshHeading"]  
    d = {}  
    for i in set(mesh):  
        count = 0  
        for j in mesh:  
            if i == j:  
                count+=1  
        d[i]=count  
    return d
```

```
In [29]: mesh_count("ALZHEIMERS")
```

```
Out[29]: {'Dystonia': 1,
          'Angelica': 1,
          'Cell Survival': 14,
          'RNA Interference': 1,
          'Macrophages': 1,
          'Continental Population Groups': 1,
          'Calcium Channels': 1,
          'Anxiety': 8,
          'Depressive Disorder': 1,
          'Drosophila melanogaster': 3,
          'Cerebral Cortex': 21,
          'Receptors, Nicotinic': 2,
          'Fluorescence Recovery After Photobleaching': 1,
          'Zebrafish': 1,
          'Alpinia': 1,
          'Ophthalmic Solutions': 1,
          'Dementia': 68,
          'Pueraria': 1,
          'Fruit': 3,
          'Infection Control': 1}
```

```
In [30]: #https://stackoverflow.com/questions/613183/how-do-i-sort-a-dictionary-by-value
def first_ten_items(disease):
    disease_mesh = mesh_count(disease)
    c=Counter(disease_mesh).most_common()
    #print(c)
    first_ten = c[:10]
    return(first_ten)
```



```
In [31]: first_ten_items("ALZHEIMERS")
```

```
Out[31]: [('Humans', 701),
          ('Alzheimer Disease', 699),
          ('Male', 349),
          ('Female', 296),
          ('Aged', 285),
          ('Animals', 270),
          ('Amyloid beta-Peptides', 210),
          ('Brain', 190),
          ('Aged, 80 and over', 168),
          ('Cognitive Dysfunction', 157)]
```

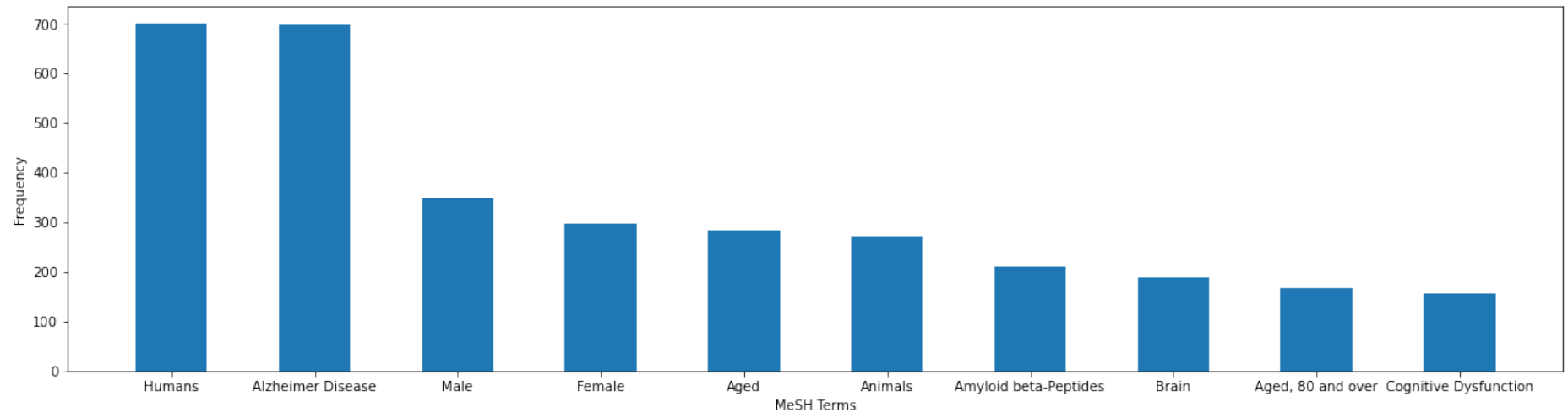
```
In [23]: #Provide a graphic illustrating their relative frequency. (3 points)
```

```
In [32]: import matplotlib.pyplot as plt
import numpy as np
```

```
In [52]: #https://stackoverflow.com/questions/34029865/how-to-plot-bar-chart-for-a-list-in-python
labels, ys = zip(*first_ten_items("ALZHEIMERS"))
xs = np.arange(len(labels))
width = 0.5
plt.figure(figsize=(20,5))
plt.bar(xs, ys,width,align='center')
plt.ylabel('Frequency')
plt.xlabel('MeSH Terms')
plt.xticks(xs,labels)
plt.yticks()
```

```
Out[52]: (array([ 0., 100., 200., 300., 400., 500., 600., 700., 800.]),
          [Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ')],
```

```
Text(0, 0, ''),  
Text(0, 0, ''),  
Text(0, 0, ''),  
Text(0, 0, ''),  
Text(0, 0, '')[0]]
```



In []: *#What are the 10 most common MeSH terms for the cancer papers whose metadata you found in Exercise 1?*

In [35]: `mesh_count("CANCER")`

```
Out[35]: {'Bayes Theorem': 1,
          'Nausea': 1,
          'Polymerase Chain Reaction': 5,
          'Wound Healing': 2,
          'Poland': 4,
          'Hydrogels': 1,
          'Free Radical Scavengers': 1,
          'Cell Survival': 1,
          'Neutrophils': 1,
          'Vaccination': 1,
          'Debridement': 1,
          'Heart Diseases': 1,
          'Spermatogonia': 1,
          'Video-Assisted Surgery': 1,
          'Silicon Dioxide': 1,
          'Red-Cell Aplasia, Pure': 1,
          'Microfluidic Analytical Techniques': 2,
          'Biosensing Techniques': 1,
          'Lung': 4,
          'Global Health': 1}
```

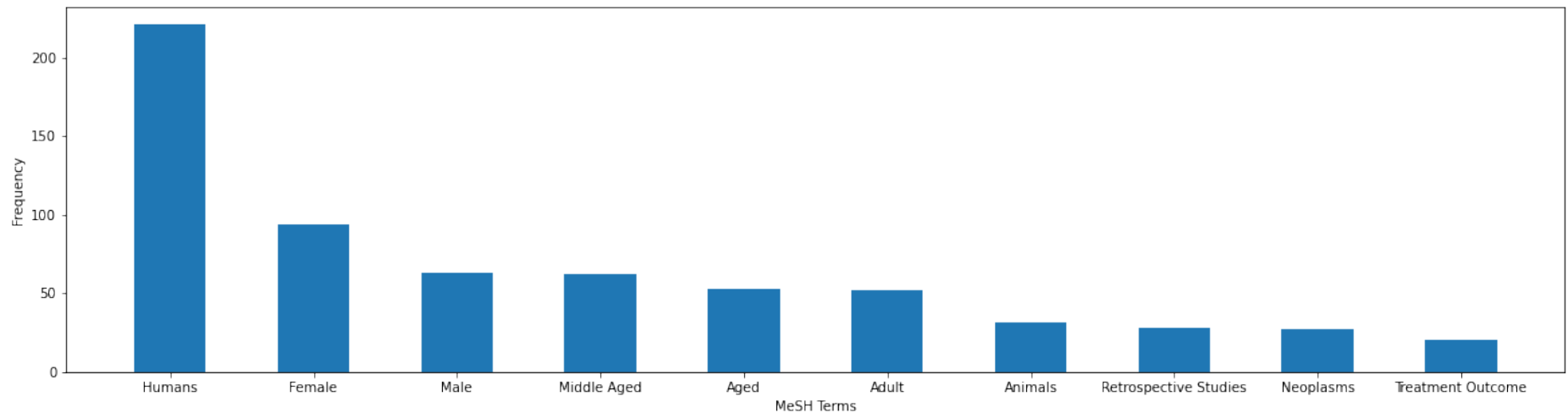
In [36]: `first_ten_items("CANCER")`

```
Out[36]: [('Humans', 221),
          ('Female', 94),
          ('Male', 63),
          ('Middle Aged', 62),
          ('Aged', 53),
          ('Adult', 52),
          ('Animals', 31),
          ('Retrospective Studies', 28),
          ('Neoplasms', 27),
          ('Treatment Outcome', 20)]
```

In []: *#Provide a graphic illustrating their relative frequency.*

```
In [53]: labels, ys = zip(*first_ten_items("CANCER"))
xs = np.arange(len(labels))
width = 0.5
plt.figure(figsize=(20,5))
plt.bar(xs, ys,width,align='center')
plt.ylabel('Frequency')
plt.xlabel('MeSH Terms')
plt.xticks(xs, labels)
plt.yticks()
```

```
Out[53]: (array([ 0.,  50., 100., 150., 200., 250.]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



```
In [ ]: #Make a labeled table with rows for each of the top 5 MeSH terms from the Alzheimer's query and columns  
#each of the top 5 MeSH terms from the cancer query. For the values in the table, provide the count of p  
#(combined, from both sets) having both the matching MeSH terms.
```

```
In [39]: pip install tabulate
```

```
Collecting tabulate  
  Downloading tabulate-0.8.9-py3-none-any.whl (25 kB)  
Installing collected packages: tabulate  
Successfully installed tabulate-0.8.9  
Note: you may need to restart the kernel to use updated packages.
```

```
In [40]: from tabulate import tabulate
```

```
In [48]: def first_5(disease):  
         disease_mesh = mesh_count(disease)  
         c=Counter(disease_mesh).most_common()  
         first_5 = c[:5]  
         return(first_5)
```



```

In [49]: matrix = [
    [0,0,0,0,0],
    [0,0,0,0,0],
    [0,0,0,0,0],
    [0,0,0,0,0],
    [0,0,0,0,0]
]

alz_5 = first_5("ALZHEIMERS")
can_5 = first_5("CANCER")

for i in range(len(alz_5)):
    for j in range(len(can_5)):
        count = 0
        for k in all_data:
            if (alz_5[i][0] in all_data[k]["MeshHeading"]) and (can_5[j][0] in all_data[k]["MeshHeading"]):
                count+=1
            matrix[i][j]= count
print(matrix)

[[923, 379, 369, 207, 339], [585, 234, 280, 108, 228], [369, 326, 413, 179, 288], [379, 391, 326, 191,
295], [339, 295, 288, 174, 339]]

```

```

In [50]: #https://towardsdatascience.com/how-to-easily-create-tables-in-python-2eaea447d8fd
table = [['Alzheimer Disease/Cancer', 'Humans', 'Female', 'Male', 'Middle Aged', 'Aged'],
        ['Humans', 922, 381, 208, 369, 338], ['Alzheimer Disease', 584, 236, 109, 281, 227],
        ['Male', 369, 327, 180, 413, 288], ['Female', 381, 394, 192, 327, 295], ['Aged', 338, 295, 175, 288

```

```
In [51]: print(tabulate(table, headers='firstrow', tablefmt='grid'))
```

Alzheimer Disease/Cancer	Humans	Female	Male	Middle Aged	Aged
Humans	922	381	208	369	338
Alzheimer Disease	584	236	109	281	227
Male	369	327	180	413	288
Female	381	394	192	327	295
Aged	338	295	175	288	338

```
In [2]: conda install pytorch torchvision -c pytorch
```

```
Collecting package metadata (current_repodata.json): done  
Solving environment: done
```

```
# All requested packages already installed.
```

Note: you may need to restart the kernel to use updated packages.

```
In [4]: pip install transformers
```

```
Collecting transformers
```

```
  Downloading transformers-4.11.3-py3-none-any.whl (2.9 MB)
```

```
    |████████████████████| 2.9 MB 2.1 MB/s eta 0:00:01
```

```
Requirement already satisfied: pyyaml>=5.1 in /opt/anaconda3/lib/python3.8/site-packages (from transformers) (5.4.1)
```

```
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/lib/python3.8/site-packages (from transformers) (20.9)
```

```
Requirement already satisfied: numpy>=1.17 in /opt/anaconda3/lib/python3.8/site-packages (from transformers) (1.20.1)
```

```
Requirement already satisfied: requests in /opt/anaconda3/lib/python3.8/site-packages (from transformers) (2.25.1)
```

```
Collecting sacremoses
```

```
  Downloading sacremoses-0.0.46-py3-none-any.whl (895 kB)
```

```
    |████████████████████| 895 kB 28.9 MB/s eta 0:00:01
```

```
Requirement already satisfied: tqdm>=4.27 in /opt/anaconda3/lib/python3.8/site-packages (from transformers) (4.59.0)
```

```
Requirement already satisfied: regex!=2019.12.17 in /opt/anaconda3/lib/python3.8/site-packages (from transformers) (2021.4.4)
```

```
Requirement already satisfied: filelock in /opt/anaconda3/lib/python3.8/site-packages (from transformers) (3.0.12)
```

```
Collecting huggingface-hub>=0.0.17
```

```

Downloading huggingface_hub-0.0.19-py3-none-any.whl (56 kB)
|████████████████████████████████████████| 56 kB 10.3 MB/s eta 0:00:01
Collecting tokenizers<0.11,>=0.10.1
  Downloading tokenizers-0.10.3-cp38-cp38-macosx_10_11_x86_64.whl (2.2 MB)
  |████████████████████████████████████████| 2.2 MB 18.2 MB/s eta 0:00:01
Requirement already satisfied: typing-extensions in /opt/anaconda3/lib/python3.8/site-packages (from huggingface_hub>=0.0.17->transformers) (3.7.4.3)
Requirement already satisfied: pyparsing>=2.0.2 in /opt/anaconda3/lib/python3.8/site-packages (from packaging>=20.0->transformers) (2.4.7)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.8/site-packages (from requests->transformers) (2020.12.5)
Requirement already satisfied: chardet<5,>=3.0.2 in /opt/anaconda3/lib/python3.8/site-packages (from requests->transformers) (4.0.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/anaconda3/lib/python3.8/site-packages (from requests->transformers) (1.26.4)
Requirement already satisfied: idna<3,>=2.5 in /opt/anaconda3/lib/python3.8/site-packages (from requests->transformers) (2.10)
Requirement already satisfied: six in /opt/anaconda3/lib/python3.8/site-packages (from sacremoses->transformers) (1.15.0)
Requirement already satisfied: click in /opt/anaconda3/lib/python3.8/site-packages (from sacremoses->transformers) (7.1.2)
Requirement already satisfied: joblib in /opt/anaconda3/lib/python3.8/site-packages (from sacremoses->transformers) (1.0.1)
Installing collected packages: tokenizers, sacremoses, huggingface-hub, transformers
Successfully installed huggingface-hub-0.0.19 sacremoses-0.0.46 tokenizers-0.10.3 transformers-4.11.3
Note: you may need to restart the kernel to use updated packages.

```

```

In [ ]: #In particular, for each paper identified from exercise 1, compute the SPECTER embedding (a 768-dimension vector) for each paper.
#Keep track of which papers came from searching for Alzheimers, which came from searching for cancer.

```

```
In [40]: from transformers import AutoTokenizer, AutoModel
```

```
# load model and tokenizer
```

```
tokenizer = AutoTokenizer.from_pretrained('allenai/specter')
```

```
model = AutoModel.from_pretrained('allenai/specter')
```

```
In [5]: import json
import numpy as np
import pandas as pd
```

```
In [41]: f = open('pb2000.json')
papers = json.load(f)
```

```
In [42]: data = []
for paper in papers.values():
    data += [paper["ArticleTitle"] + tokenizer.sep_token + paper["AbstractText"]]

inputs = tokenizer(data[0], padding=True, truncation=True, return_tensors="pt", max_length=512)
result = model(**inputs)
# take the first token in the batch as the embedding
embeddings_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)
    embeddings = result.last_hidden_state[:, 0, :].detach().numpy()
    embeddings_total = np.concatenate((embeddings_total, embeddings), axis=0)
```

```
In [43]: #Find the index of the overlap paper.
paper_list = list(papers)
for i in range(len(paper_list)):
    if paper_list[i] == "32501203":
        print(i)
```

885

```
In [44]: from sklearn import decomposition

pca = decomposition.PCA(n_components=3)
embeddings_pca = pd.DataFrame(
    pca.fit_transform(embeddings_total),
    columns=['PC0', 'PC1', 'PC2']
)
embeddings_pca["Query"] = [paper["Query"][0] for paper in papers.values()]
embeddings_pca.loc[885,"Query"] = 'BOTH' #Name the overlap paper as "Both" #https://www.geeksforgeeks.org
```

In [45]: embeddings_pca

Out [45]:

	PC0	PC1	PC2	Query
0	-4.686102	-4.797193	-1.501220	CANCER
1	4.000983	-2.101142	-3.992321	CANCER
2	1.513947	-3.910135	-4.383604	CANCER
3	3.747759	-0.245759	-4.093632	CANCER
4	2.980736	-3.004155	-5.297383	CANCER
...
1994	-4.649996	-0.426457	-0.557408	ALZHEIMERS
1995	-4.186847	-0.315715	1.904060	ALZHEIMERS
1996	-8.014501	0.748415	3.851795	ALZHEIMERS
1997	-7.775526	-2.819993	-0.221032	ALZHEIMERS
1998	-6.987017	-3.781791	-1.301347	ALZHEIMERS

1999 rows × 4 columns

In [46]: embeddings_pca.loc[885, "Query"]

Out [46]: 'BOTH'

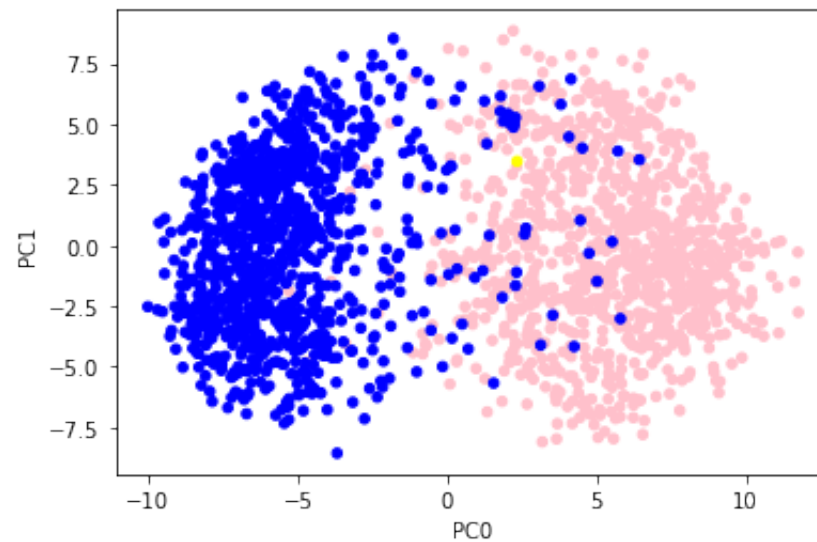
In [50]: embeddings_pca.loc[999, "Query"]

Out [50]: 'CANCER'

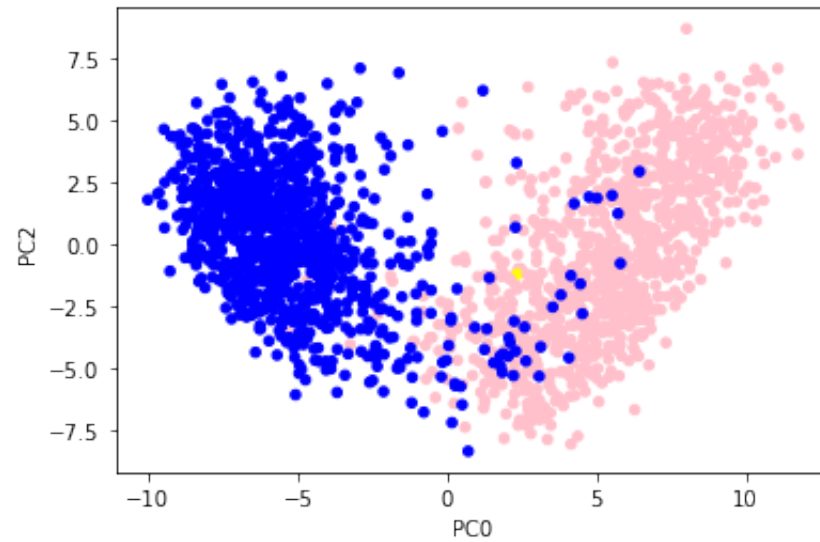
```
In [63]: embeddings_pca.loc[1000,"Query"]
```

```
Out[63]: 'ALZHEIMERS'
```

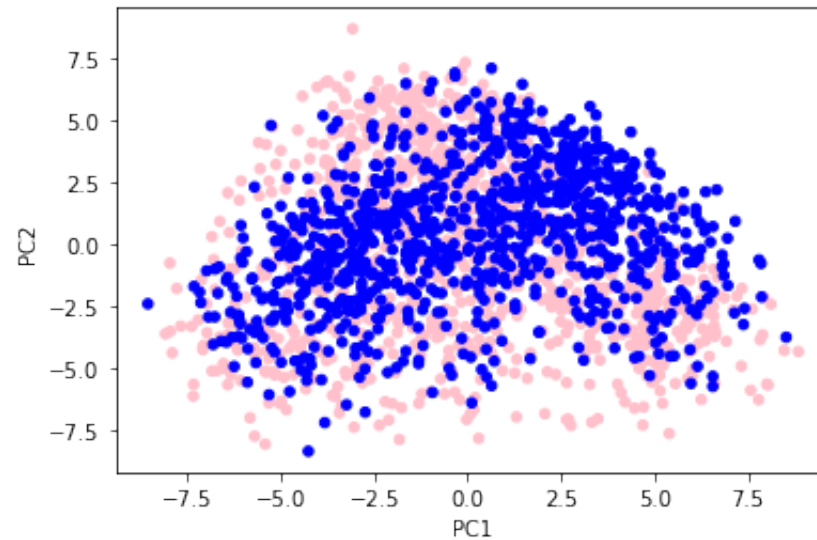
```
In [52]: embeddings_pca.plot.scatter(x = 'PC0', y = 'PC1', c=embeddings_pca.Query.map(dict(ALZHEIMERS='blue', CAN
```




```
In [53]: embeddings_pca.plot.scatter(x = 'PC0', y = 'PC2', c=embeddings_pca.Query.map(dict(ALZHEIMERS='blue', CAN
```



```
In [54]: embeddings_pca.plot.scatter(x = 'PC1', y = 'PC2', c=embeddings_pca.Query.map(dict(ALZHEIMERS='blue', CAN
```



```
In [16]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

```
In [71]: l = []
for i in range(885):# From index 0-884 the paper is Cancer, tag as 1
    l.append(1)
l.append(2) #The overlap, tag as 2
for i in range(114):#From 886-999 the paper is Cancer, still tag as 1
    l.append(1)
for i in range(999):#From index 1000-1998 are Alzheimers, tag as 0
    l.append(0)
total = np.array(l)

lda = LDA(n_components=2)
embeddings_lda = pd.DataFrame(
    lda.fit_transform(embeddings_total,total),
    columns=['LD0','LD1']
)

embeddings_lda["Query"] = [paper["Query"][0] for paper in papers.values()]
embeddings_lda.loc[885,"Query"] = 'BOTH'
```

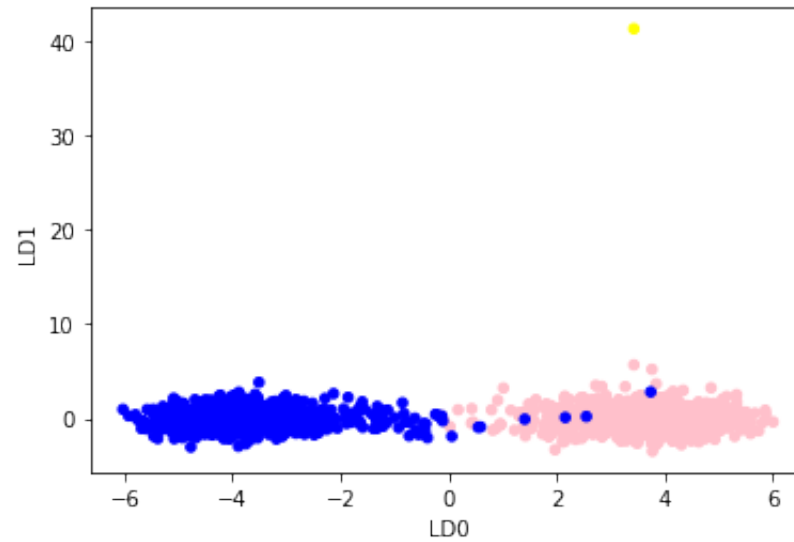
In [72]: embeddings_lda

Out[72]:

	LD0	LD1	Query
0	2.075702	1.164759	CANCER
1	3.819197	-0.315971	CANCER
2	3.798645	0.861194	CANCER
3	2.895172	-0.432372	CANCER
4	4.449149	-0.713219	CANCER
...
1994	-2.163132	-0.797396	ALZHEIMERS
1995	-3.535121	-2.088272	ALZHEIMERS
1996	-4.141159	-1.050867	ALZHEIMERS
1997	-4.080945	0.309695	ALZHEIMERS
1998	-4.240591	-1.729434	ALZHEIMERS

1999 rows × 3 columns

```
In [74]: embeddings_lda.plot.scatter(x = 'LD0', y = 'LD1', c=embeddings_lda.Query.map(dict(ALZHEIMERS='blue', CAN
```



```
In [31]: import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np
```

```
In [ ]: #Foodborne Disease Outbreaks, 1998-2015  
#https://www.kaggle.com/cdc/foodborne-diseases  
#Using the dataset from January 1998 to December 2015
```

```
In [33]: outbreak_df = pd.read_csv("outbreaks.csv")
```

In [34]: outbreak_df

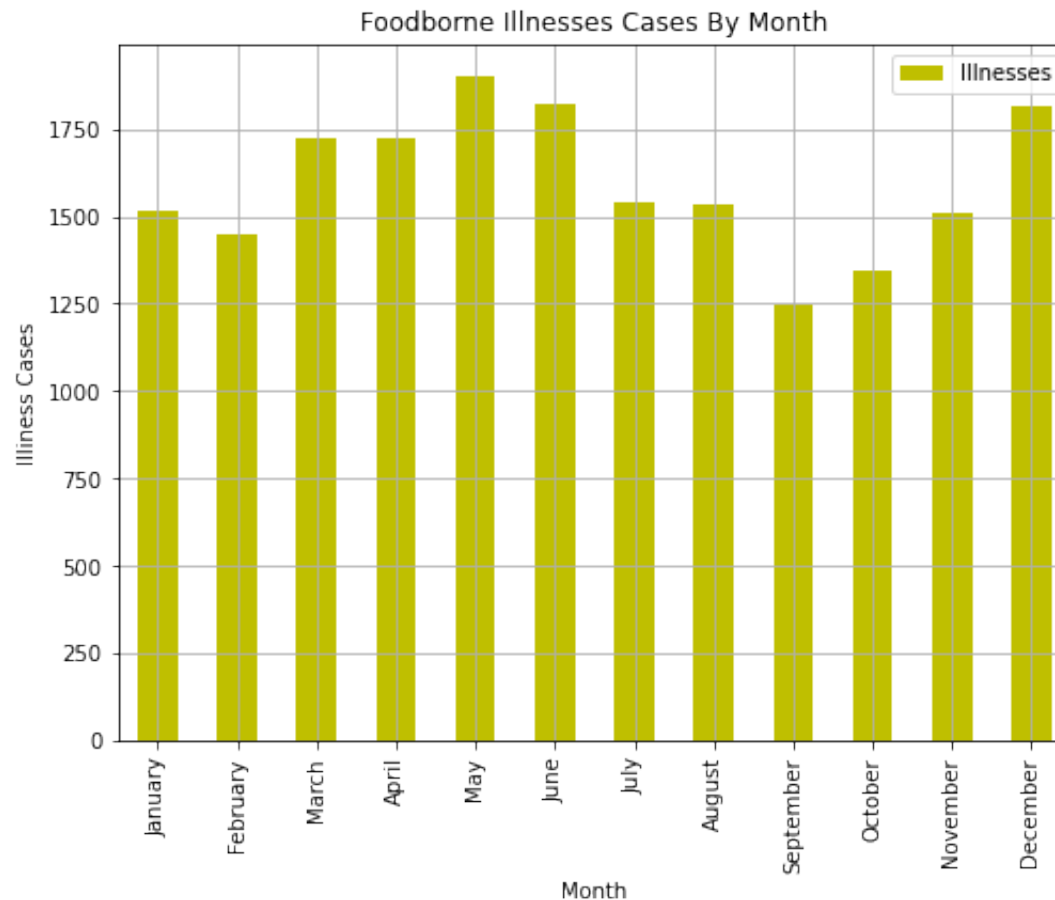
Out[34]:

	Year	Month	State	Location	Food	Ingredient	Species	Serotype/Genotype	Status	Illnesses	Hospitalizati
0	1998	January	California	Restaurant	NaN	NaN	NaN	NaN	NaN	20	
1	1998	January	California	NaN	Custard	NaN	NaN	NaN	NaN	112	
2	1998	January	California	Restaurant	NaN	NaN	NaN	NaN	NaN	35	
3	1998	January	California	Restaurant	Fish, Ahi	NaN	Scombroid toxin	NaN	Confirmed	4	
4	1998	January	California	Private Home/Residence	Lasagna, Unspecified; Eggs, Other	NaN	Salmonella enterica	Enteritidis	Confirmed	26	
...	
19114	2015	December	Wisconsin	Restaurant	NaN	NaN	Norovirus genogroup II	GII_14	Confirmed	4	
19115	2015	December	Wisconsin	Private Home/Residence	Salsa	NaN	Norovirus genogroup II	GII_2	Confirmed	16	
19116	2015	December	Wisconsin	Nursing Home/Assisted Living Facility	NaN	NaN	Norovirus genogroup II	GII_17 Kawasaki (2014)	Confirmed	43	
19117	2015	December	Wisconsin	Fast Food Restaurant	NaN	NaN	Norovirus genogroup II	GII_2	Confirmed	5	
19118	2015	December	Wyoming	NaN	NaN	NaN	NaN	NaN	NaN	3	

19119 rows × 12 columns

In []: #1. Which month does foodborne tend to occur the most?

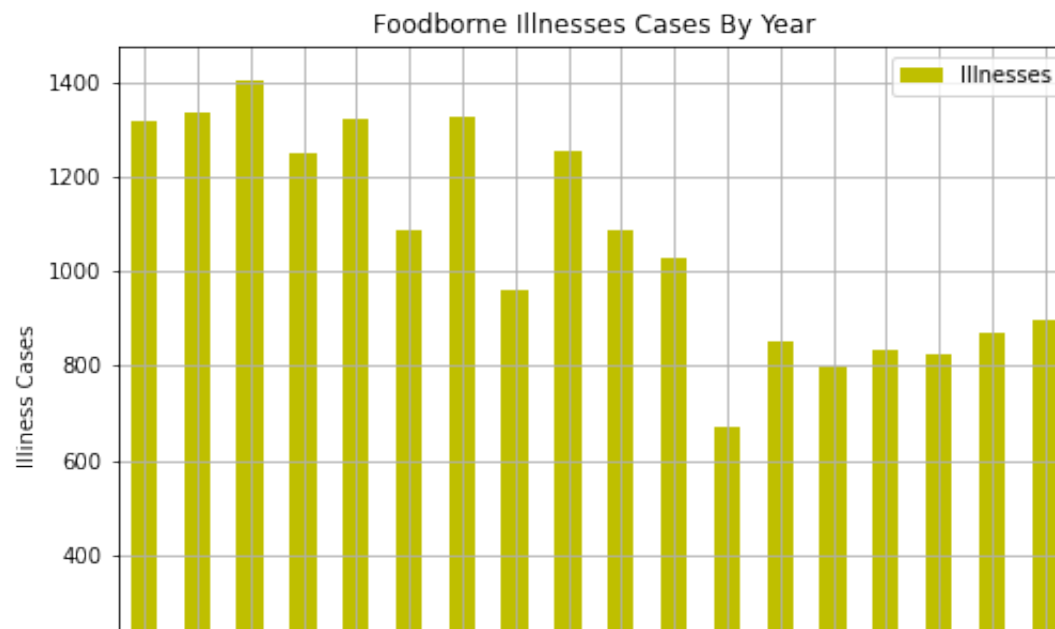
```
In [145]: df1 = pd.pivot_table(outbreak_df, index='Month', values='Illnesses', aggfunc='count')#https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of-figures-drawn-with-matplotlib
df1 = df1.reindex(['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December'])
ax = df1.plot(kind='bar', color='y', grid=True)
plt.title('Foodborne Illnesses Cases By Month')
plt.ylabel('Illness Cases')
fig = plt.gcf()#https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of-figures-drawn-with-matplotlib
fig.set_size_inches(8, 6)
```

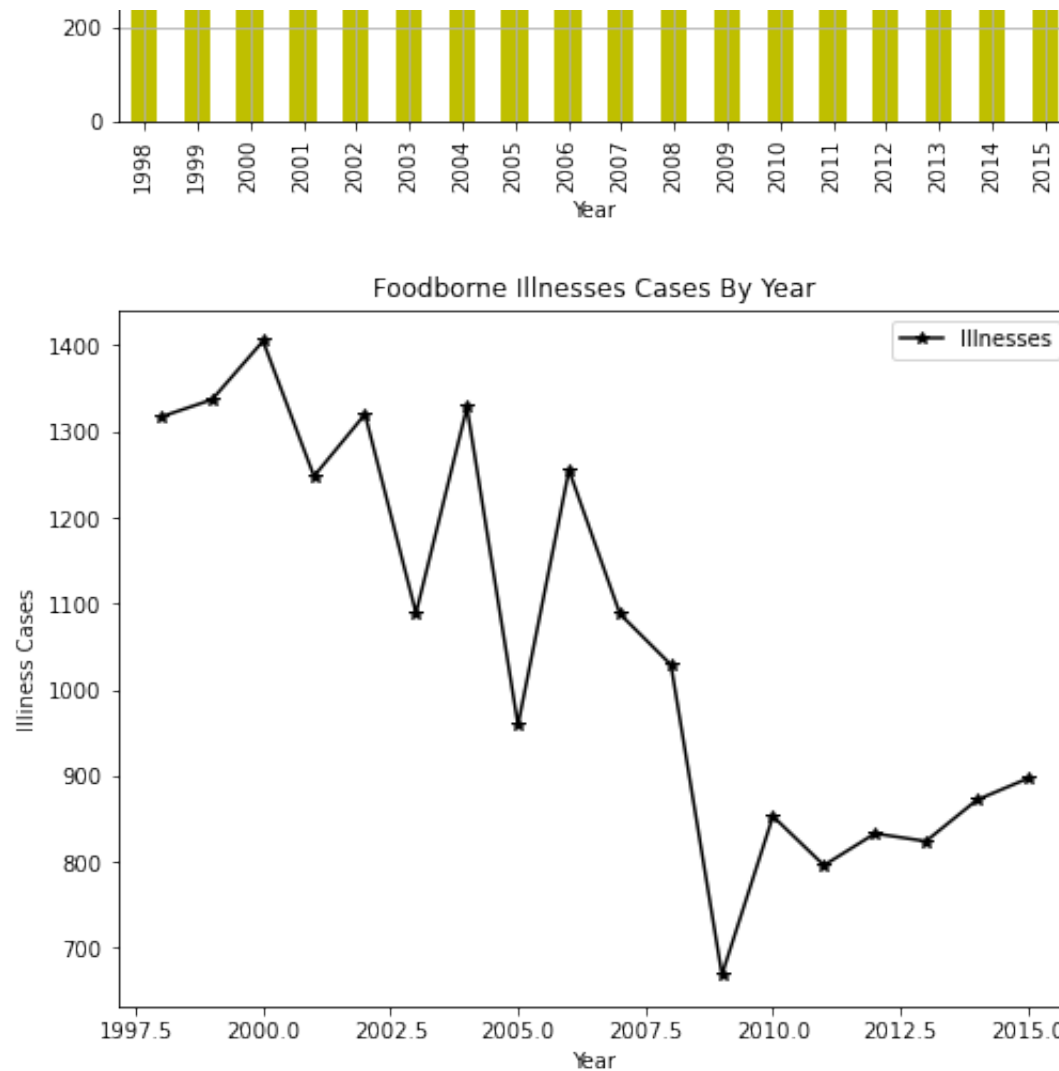



```
In [ ]: #Foodborne tends to occur most in May,June, and December.
```

```
In [ ]: #2. Has foodborne disease outbreaks gotten worse over the years?
```

```
In [153]: df1 = pd.pivot_table(outbreak_df, index='Year', values='Illnesses', aggfunc='count')
ax1 = df1.plot(kind='bar', color='y',grid=True)
fig = plt.gcf()
fig.set_size_inches(8, 6)
plt.title('Foodborne Illnesses Cases By Year')
plt.ylabel('Illness Cases')
ax2 = df1.plot(kind='line',marker='*',color='black')
plt.title('Foodborne Illnesses Cases By Year')
plt.ylabel('Illness Cases')
fig = plt.gcf()
fig.set_size_inches(8, 6)
```

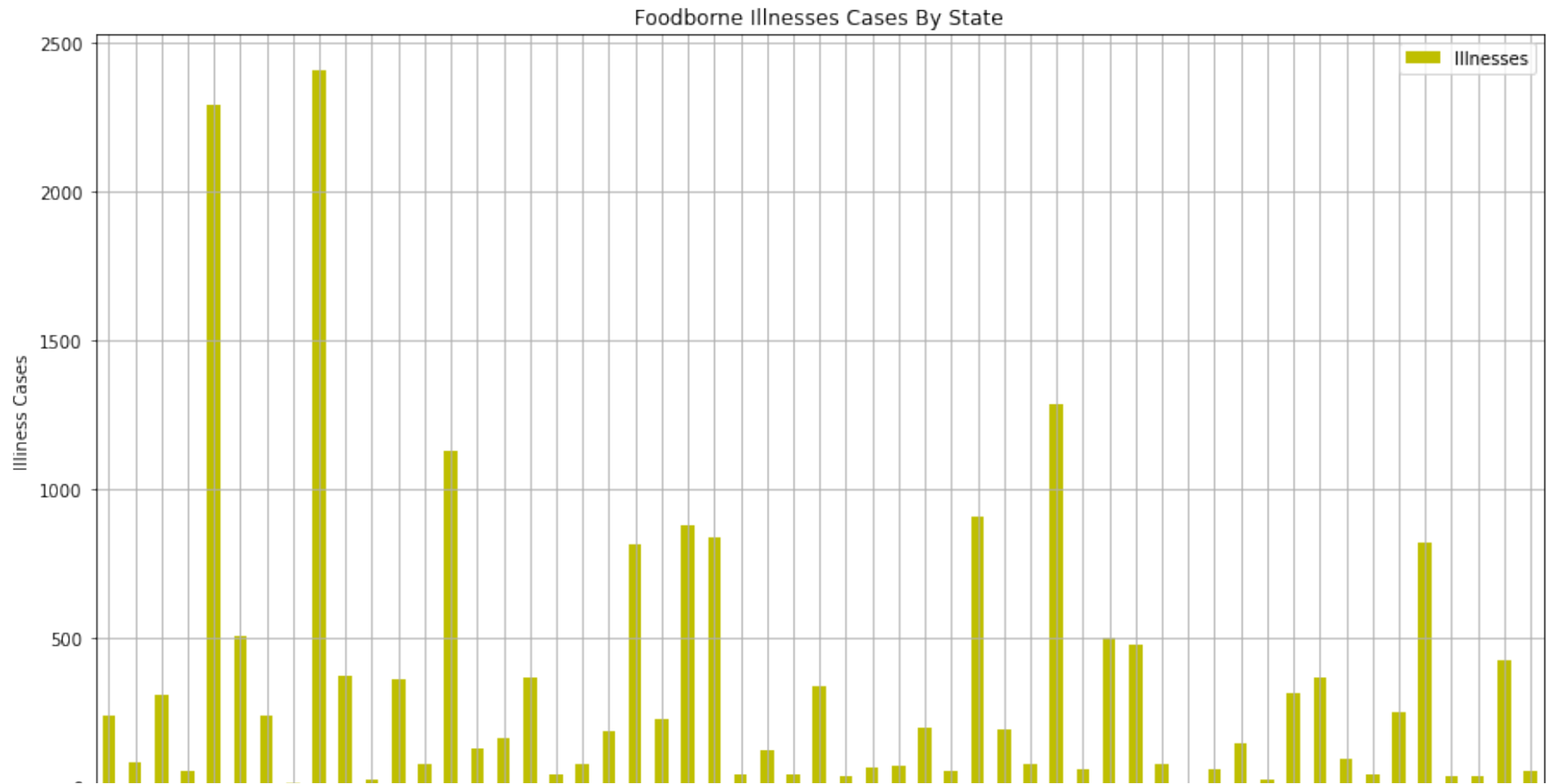


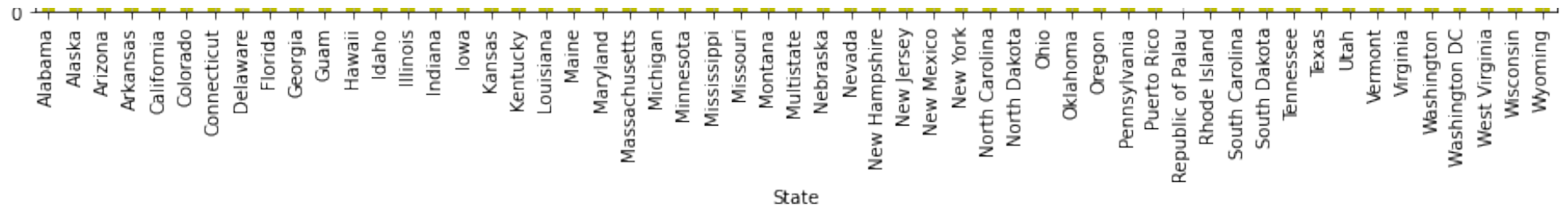


```
In [ ]: # As the graph showing, the trend of foodborne illnesses has decreased overall since 1998.  
#There are some significant drops, one is from 2004 - 2005,  
#the other is from 2008-2009, but now seems to be steady around 800 - 900 cases per year.  
#The data can be concluded that foodborne diseases are occurring less frequently by years.
```

```
In [ ]: # 3. Which state of the US do they appear the most frequently?
```

```
In [95]: df1 = pd.pivot_table(outbreak_df, index='State', values='Illnesses', aggfunc='count')
ax = df1.plot(kind='bar', color='y', grid=True)
plt.title('Foodborne Illnesses Cases By State')
plt.ylabel('Illness Cases')
fig = plt.gcf()
fig.set_size_inches(15, 8)
```





```
In [ ]: # Clearly, California and Florida have extremely high rate of foodborne illnesses. The possible reasons
# could be related to their population rate during the years of 1998–2015. An exceptional circumstance was
# the outbreak of Salmonella Infections linked to Peanut Butter: at least 36 people across 17 states have
# been affected, 2008 – 2009. Majority of peanut butter was sent to California and Ohio, according to
# https://www.healthline.com/health-news/salmonella-outbreak-in-17-states-linked-to-italian-meats-what-to
```

```
In [ ]: ##Data cleaning& organizing
```

```
In [ ]: #1) Any missing value?
```

```
In [12]: outbreak_df.isnull().sum()
```

```
Out[12]: Year          0
Month          0
State          0
Location      2166
Food          8963
Ingredient    17243
Species       6619
Serotype/Genotype 15212
Status        6619
Illnesses      0
Hospitalizations 3625
Fatalities     3601
dtype: int64
```

```
In [15]: missing_value = outbreak_df.isnull().sum()*100/len(outbreak_df)
missing_value
```

```
Out[15]: Year                0.000000
Month                0.000000
State                0.000000
Location            11.329044
Food                46.880067
Ingredient           90.187771
Species             34.620012
Serotype/Genotype    79.564831
Status              34.620012
Illnesses            0.000000
Hospitalizations     18.960197
Fatalities           18.834667
dtype: float64
```

```
In [ ]: #Clearly, the dataset has missing values. The data missing can be MNAR, for example, the restaruant is u
#to report the location, food, ingredient and species;
#or the customers are not able to recall what food or ingredient caused the outbreaks.
#Besides, most foodborne infections go undiagnosed and unreported, because the those people
#don't see the doctors, or the doctor doesn't made a specific diagnoses according to the CDC.
#The missing data also can be MAR, for example, the missing data of ingredient and species can be relate
#missing data of food; the missing data of fatalities can be related to missing data of hospitalizations
```

```
In [19]: #2) Filter the dataset, drop the columns that are not related to the topic
##Since the ratio of null fields in the column'Serotype/Genotype' and 'Ingredient' to the total number o
#exceeds 60%,which are proved not useful for the data anlysis.
#Therefore I choose to drop those two columns from my table.
#Then, drop all null values.
```

```
In [27]: df2 = outbreak_df.copy()
cols = [5,7]
df2.drop(df2.columns[cols], axis=1, inplace=True)
df2.dropna(inplace=True)
df2.head()
```

```
Out [27]:
```

	Year	Month	State	Location	Food	Species	Status	Illnesses	Hospitalizations	Fatalities
3	1998	January	California	Restaurant	Fish, Ahi	Scombroid toxin	Confirmed	4	0.0	0.0
4	1998	January	California	Private Home/Residence	Lasagna, Unspecified; Eggs, Other	Salmonella enterica	Confirmed	26	3.0	0.0
7	1998	January	California	Restaurant	Stuffing, Unspecified; Sandwich, Turkey	Salmonella enterica	Confirmed	4	3.0	0.0
15	1998	January	Florida	Restaurant	Ethnic Style, Unspecified	Clostridium perfringens	Suspected	3	0.0	0.0
17	1998	January	Florida	Restaurant	Ground Beef, Cheeseburger	Staphylococcus aureus	Suspected	2	0.0	0.0

```
In [28]: df2.isnull().sum()
```

```
Out [28]: Year      0
Month      0
State      0
Location   0
Food       0
Species    0
Status     0
Illnesses  0
Hospitalizations  0
Fatalities  0
dtype: int64
```

```
In [ ]: #The data cleansing is finish, and the data does need to be cleaned,  
#because the null values and excessive null columns are dropped.
```