QingyangYu0529 / BIS-634-QingyangYu Private

Code Issues Pull requests Actions Projects Security Insights Settings

main

BIS-634-QingyangYu / Homework3 / README.md

QingyangYu0529 Merge branch 'main' of https://github.com/QingyangYu0529/BIS-6...

1 contributor

Readme for HW #3

Instruction

This readme file contains:

1)Instructions that are necessary for running scripts under this folder.

2)Answers to each question asked in the exercises 1-5.

3)How I proved my code was correct using different tests.

4)Any other relevant information.

Support

If you have any question, please feel free to contact me.

Any comments or insights would be greatly appreciated.

Exercise 1

Question

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) Note: To search for a disease and a publication year, structure the term like: Alzheimers+AND+2019[pdat] (Here [pdat] indicates that this is a publication year, and the AND (has to be all caps) means both conditions should apply.) I'm asking for papers from 2019 instead of 2021 because they're still reasonably recent but the new ones may not have been fully curated and may still be missing MeSH terms... for the papers from 2019, if they're going to get MeSH terms, they were processed long ago. There are of course many more papers of each category, but is there any overlap in the two sets of papers that you identified? (3 points) Use the Entrez API via requests/urllib to pull the metadata for each such paper 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: (12 points)

```
{
    "32008517": {
        "ArticleTitle": "Deep Learning for Alzheimer's Disease
Classification using Texture Features",
        "AbstractText": "We propose a classification method for
```

935 lines (689 sloc) | 54.6 KB

```
        "mesh": ["Alzheimer Disease", "Brain", "Case-Control St
udies", ...]
    }, ...
}
```

Here 32008517 is the PubMed ID of one of the 2000 papers, specifically one that came from searching for Alzheimer's papers. You should include the full AbstractText and list of MeSH terms; I'm abridging here for clarity. Hint: To do this, you'll probably want to look at one of the XML responses with a text editor so that you understand how it is structured. Hint: 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 (1 point). Caution: the PubMed API allows a rate of at most one query at a time and no more than 3 per second unless you have an API key. To be safe, use

```
time.sleep(1)
```

after each query to the PubMed API. Note: This doesn't require 2002 separate queries. You can get the metadata for many articles at a time by using a comma separated list of ids. While GET queries have a total line length limit, you could use a POST query instead and get the information for all the papers in one pass. (We can use POST instead of GET here in part because this is not a RESTful API.) Note: BioPython provides functions for accessing the PubMed API. Do not use them; use the requests module to do an HTTP or HTTPS request directly on a URL that you specify with the parameters that you specify. Why? Because this approach is general and will work in many contexts whereas BioPython only works for PubMed and only from Python.

Solution

>> Code explanation

```
# import module minidom and libraries requests, time, json.
import requests
import xml.dom.minidom as m
import time
import json
```

Imported module minidom and libraries requests, time, json.

```
# function get_id_of_disease() was defined to return the PubmedId list of a speci
def get_id_of_disease(disease):
    # send a GET request to the specified url, and return a response object.
    r = requests.get(f"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi
    time.sleep(1)
    # use minidom to parse strings of text from response object.
    doc = m.parseString(r.text)
    # get the Id elements by .getElementByTagName().
    PubmedId = doc.getElementsByTagName('Id')
    # create a list IdList, to save all the PubmedId.
    IdList = []
    # use for loop to get values of element, save into the list IdList.
    # reference: https://stackoverflow.com/questions/317413/get-element-value-wit
    for i in range(len(PubmedId)):
        IdList.append(PubmedId[i].firstChild.data)

    return IdList
```

1. Function get_id_of_disease() was defined to return the PubmedId list of a specific disease. First I used requests.get() to send a GET request to the specified url and returned a response object: search term is Alzheimers/cancer+AND+2019[pdat], retmode is xml, retmax = 1000. Then used minidom to parse strings of text from the response object, and got the Id elements by .getElementsByTagName(), saved

into the list PubmedId. List IdList was created to save all the PubmedId. Finally I used for loop to get all the data of the elements' firstchild, and saved into the list IdList.

```
# function overlap_in_two_papers() was defined, to determine whether there is a c
def overlap_in_two_papers(disease1,disease2):
    # get the PubmedId list from disease1 and disease2, save into lists IdList1,
    IdList1 = get_id_of_disease(disease1)
    IdList2 = get_id_of_disease(disease2)
    # turn list into set to remove the duplicates.
    set1 = set(IdList1)
    set2 = set(IdList2)
    overlap = list(set1&set2)
    # If there is no elements in the list overlap, there is no overlap.
    if len(overlap) == 0:
        print('There is no overlap in the two sets of papers that I identified.')
    # If there is one element in the list overlap, there is a overlap.
    elif len(overlap) == 1:
        print(f"There is a overlap in the two sets of papers that I identified, t
        return overlap[0]
    # If there are more than one elements in the list overlap, there are multiple
    else:
        print(f"There are overlaps in the two sets of papers that I identified, t
        return overlap



overlap_in_two_papers('Alzheimers','cancer')
```

2. Function overlap_in_two_papers() was defined to determine whether there is a overlap in the two sets of papers. First I ran above function to get the PubmedId list from disease1 and disease2, and save into lists IdList1, IdList2, separately. Then turned list into set to remove the duplicates, found the overlap between two sets and returned the overlap.

```
# function pull_metadata() was defined, to pull the metadata for each paper of a
def pull_metadata(disease):
    global paper
    # run function get_id_of_disease(), to get the PubmedId list of a specific di
    IdList = get_id_of_disease(disease)
    # create dictionary paper, to save the metadata for each paper of this diseas
    paper = {}
    for PubmedId in IdList:
        # suspends execution for each second.
        time.sleep(1)
        # send a GET request to the url of each PubmedId and returned a response
        r = requests.post(f"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.
        doc = m.parseString(r.text)

        # get the ArticleTitle elements by .getElementByTagName().
```

```
            ArticleTitle = doc.getElementsByTagName('ArticleTitle')
            Title = ""
            if len(ArticleTitle) > 0:
                # if ArticleTitle is not empty, loop through childnodes of all the el
                for elm in ArticleTitle:
                    for textmessage in elm.childNodes:
                        try:
                            Title += textmessage._get_wholeText()
                        # reference: https://docs.python.org/3/tutorial/errors.html
                        # if AttributeError is reported, check if the next childnode
                        except AttributeError:
                            for subnode in textmessage.childNodes:
                                if subnode.nodeType == m.Node.TEXT_NODE:
                                    Title += subnode.data
            # same as AbstractText.
            AbstractText = doc.getElementsByTagName('AbstractText')
            Abstract = ""
            if len(AbstractText) > 0:
                for elm in AbstractText:
                    for textmessage in elm.childNodes:
                        try:
                            Abstract += textmessage._get_wholeText()
                        # reference: https://docs.python.org/3/tutorial/errors.html
                        except AttributeError:
                            for subnode in textmessage.childNodes:
                                if subnode.nodeType == m.Node.TEXT_NODE:
                                    Abstract += subnode.data

            # reference: https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db
            MeshHeading = doc.getElementsByTagName('MeshHeading')
            Mesh = []
            # reference: https://stackoverflow.com/questions/6520192/how-to-get-the-t
            if len(MeshHeading) > 0:
                try:
                    for mesh in MeshHeading:
                        Mesh.append(mesh.firstChild.childNodes[0].nodeValue)
                except AttributeError: pass

            # set the dictionary key as PubmedId, dictionary values include each pape
            paper[PubmedId] = {
                'ArticleTitle': Title,
                'AbstractText': Abstract,
                'Query': disease,
                'Mesh': Mesh
            }
        # return the dictionary paper.
        return paper
```

3. Function pull_metadata() was defined, to pull the metadata for each paper of a specific disease. First I ran the function get_id_of_disease(), to get the PubmedId list of a specific disease. Then I created dictionary paper, to save the metadata for each paper of this disease. For each PubmedId, I sent a GET request to the url of each

PubmedId and returned a response object, and parsed strings of text from the response object.

For ArticleTitle, first I got the ArticleTitle elements by .getElementByTagName(). If ArticleTitle is not empty, loop through childnodes of all the elements, and save the text message into Title. If AttributeError was reported, checked if the next childnode is a text node, and also saved the data of next childnode into Title.

AbstractText and Mesh are same as above.

Set the dictionary key as PubmedId, dictionary values include each paper's title, abstract, MeSH terms and query.

```
# save metadata of Alzheimer's into dictionary all_data, save metadata of cancers
all_data = pull_metadata('Alzheimers')
cancer_data = pull_metadata('cancer')
# use .update() to update the dictionary all_data, so that data from both Alzheim
all_data.update(cancer_data)
# run function overlap_in_two_papers(), to find the overlap in the two sets of pa
overlap_PubmedId = overlap_in_two_papers('Alzheimers','cancer')
# change the query of this overlap into 'Alzheimer's', 'cancer'.
all_data[overlap_PubmedId]['Query'] = "Alzheimers or cancer"

# save the dictionary all_data into a JSON file paper.json.
with open('paper.json','a') as f:
    json.dump(all_data,f)
```

4. Ran the function pull_metadata(), and saved metadata of Alzheimer's into dictionary all_data, saved metadata of cancers into dictionary cancer_data. Then I used .update() to update the dictionary all_data, so that data from both Alzheimer's and cancer are saved into all_data. Ran the function overlap_in_two_papers(), to find the overlap in the two sets of papers. Changed the query of this overlap into 'Alzheimer's', 'cancer'. Finally the dictionary all_data was saved into a JSON file paper.json(in the 'Files' folder).

>> **Question answer**

1. There is a overlap in the two sets of papers that I identified, the pubmed id is 32501203.

2. In order to save all the AbstractText fields of some papers, when looping through childnodes of all the elements, if there is a AttributeError(which suggest there is a childnode of the childnode), I would use try/except AttributeError and save the data of the childnode of the childnode into Abstract, concatenate with other data with a space in between.

Advantages: In this way, I could save all the data when a paper has multiple AbstractText fields.

Disadvantages: However, the running time and memory cost inevitably increase, since for cases that AttributeError happens, I have to save the data of a childnode of the childnode. Also, I did not save the information of AbstractText Label(e.g., BACKGROUND, METHODS, RESULTS, CONCLUSION), this might cause information loss.

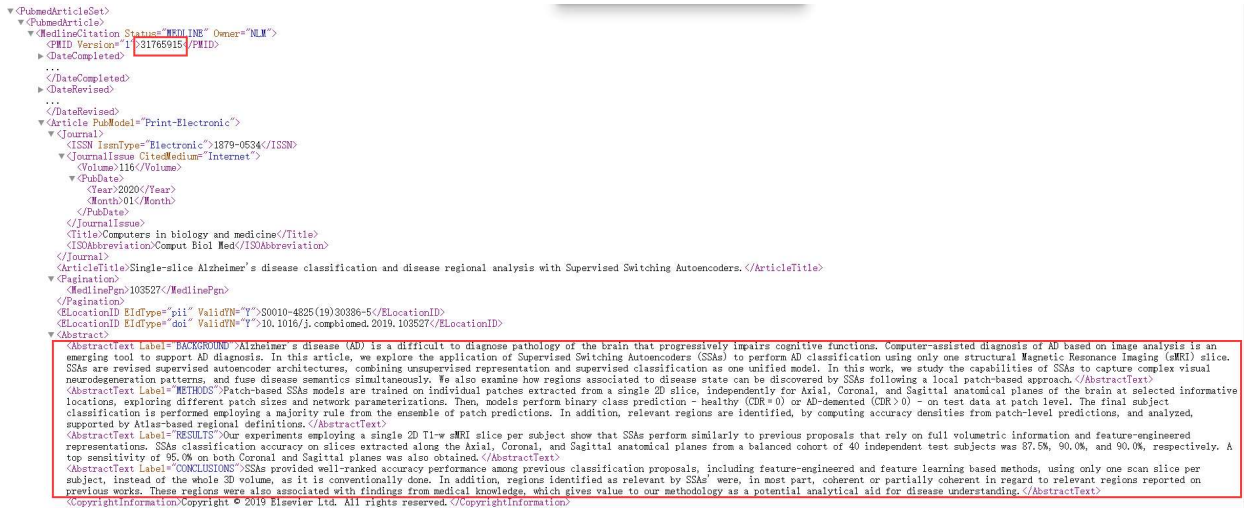
```
AbstractText = doc.getElementsByTagName('AbstractText')
Abstract = ""
if len(AbstractText) > 0:
    for elm in AbstractText:
        for textmessage in elm.childNodes:
            try:
                Abstract += textmessage._get_wholeText()
            # reference: https://docs.python.org/3/tutorial/errors.html
            except AttributeError:
                for subnode in textmessage.childNodes:
                    if subnode.nodeType == m.Node.TEXT_NODE:
                        Abstract += subnode.data
```

>> Testing

```
# import library random.
import random as rd
# read JSON file using the open function.
with open('paper.json') as f:
    all_data = json.load(f)

# separate data into alzheimer's data(alz_data) and cancer's data(cancer_data) ba
alz_data = {PubmedId: data for PubmedId, data in all_data.items()
            if data["Query"] == "Alzheimers" or data["Query"] == ['Alzheimers','c
cancer_data = {PubmedId: data for PubmedId, data in all_data.items()
            if data["Query"] == "cancer" or data["Query"] == ['Alzheimers','cance

# function random_sample_for_testing() was defined to randomly select n elements
def random_sample_for_testing(dictionary, n):
    random_sample_dic = {}
    for key in rd.sample(list(dictionary.keys()), n):
        random_sample_dic[key] = dictionary[key]
    return random_sample_dic

# merge the two dictionaries.
random_total_sample = random_sample_for_testing(alz_data,5)
random_cancer_sample = random_sample_for_testing(cancer_data,5)
random_total_sample.update(random_cancer_sample)

# save into JSON file paper_random_sample.json.
with open('paper_random_sample.json','a') as f:
    json.dump(random_total_sample,f)
```

To test the results in exercise 1(to prove the validity of pull_metadata()), I randomly select n elements from the dictionaries alz_data and cancer_data, and return the metadata of these elements. Then merged the two dictionaries and saved into JSON file paper_random_sample.json(in the 'Files' folder). Then I manually checked the results by one by one comparing contents in the JSON file and contents in the XML file.


```
"Query": "Alzheimers", "Mesh": ["Alzheimer Disease", "Disease Progression", "Humans", "Therapeutics"]}, "31765915": {"ArticleTitle":
"Single-slice Alzheimer's disease classification and disease regional analysis with Supervised Switching Autoencoders.", "AbstractText":
"Alzheimer's disease (AD) is a difficult to diagnose pathology of the brain that progressively impairs cognitive functions.
Computer-assisted diagnosis of AD based on image analysis is an emerging tool to support AD diagnosis. In this article, we explore the
application of Supervised Switching Autoencoders (SSAs) to perform AD classification using only one structural Magnetic Resonance Imaging
(sMRI) slice. SSAs are revised supervised autoencoder architectures, combining unsupervised representation and supervised classification
as one unified model. In this work, we study the capabilities of SSAs to capture complex visual neurodegeneration patterns, and fuse
disease semantics simultaneously. We also examine how regions associated to disease state can be discovered by SSAs following a local
patch-based approach.Patch-based SSAs models are trained on individual patches extracted from a single 2D slice, independently for Axial,
Coronal, and Sagittal anatomical planes of the brain at selected informative locations, exploring different patch sizes and network
parameterizations. Then, models perform binary class prediction - healthy (CDR\u202f=\u202f0) or AD-demented (CDR\u202f>\u202f0) - on
test data at patch level. The final subject classification is performed employing a majority rule from the ensemble of patch predictions.
In addition, relevant regions are identified, by computing accuracy densities from patch-level predictions, and analyzed, supported by
Atlas-based regional definitions.Our experiments employing a single 2D T1-w sMRI slice per subject show that SSAs perform similarly to
previous proposals that rely on full volumetric information and feature-engineered representations. SSAs classification accuracy on
slices extracted along the Axial, Coronal, and Sagittal anatomical planes from a balanced cohort of 40 independent test subjects was 87.
5%, 90.0%, and 90.0%, respectively. A top sensitivity of 95.0% on both Coronal and Sagittal planes was also obtained.SSAs provided
well-ranked accuracy performance among previous classification proposals, including feature-engineered and feature learning based
methods, using only one scan slice per subject, instead of the whole 3D volume, as it is conventionally done. In addition, regions
identified as relevant by SSAs' were, in most part, coherent or partially coherent in regard to relevant regions reported on previous
works. These regions were also associated with findings from medical knowledge, which gives value to our methodology as a potential
analytical aid for disease understanding.", "Query": "Alzheimers", "Mesh": ["Adolescent", "Adult", "Aged", "Aged, 80 and over",
```

For papers that have multiple AbstractText parts(e.g., PubmedId = 31765915, 32490210), by using pull_metadata() function, all parts were successfully stored in the JSON file.

Exercise 2

Question

This question refers to the Alzheimer's and cancer metadata stored in Exercise 1. It is not intended to involve any new queries to PubMed nor to make statements about PubMed beyond that of the papers whose data was obtained. What fraction of the Alzheimer's papers have no MeSH terms? (2 points) What fraction of the cancer papers have no MeSH terms? (2 points) 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?) What are the 10 most common MeSH terms for the Alzheimer's papers whose metadata you found in Exercise 1? (2 points) Provide a graphic illustrating their relative frequency. (3 points) What are the 10 most common MeSH terms for the cancer papers whose metadata you found in Exercise 1? (2 points) Provide a graphic illustrating their relative frequency. (3 points) 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. (5 points) Ideally, you can have the computer generate the table directly, but if not you could use nested for loops, label your output, and manually assemble a table in your readme document. 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. (5 points)

Solution

>> Code explanation

```
# import json
import json
# reference: python read JSON file: https://www.programiz.com/python-programming/
# read JSON file using the open function.
with open('paper.json') as f:
    all_data = json.load(f)
```

Imported libarary json. Read JSON file using the open function. The file was parsed using json.load() method which gave us a dictionary named all_data.

```
# suppose the data is in all_data. separate data into alzheimer's data(alz_data)
alz_data = {PubmedId: data for PubmedId, data in all_data.items()
            if data["Query"] == "Alzheimers" or data["Query"] == ['Alzheimers','c
cancer_data = {PubmedId: data for PubmedId, data in all_data.items()
            if data["Query"] == "cancer" or data["Query"] == ['Alzheimers','cance

# use variable alz_count_noMesh to save the number of Alzheimer's papers that hav
alz_count_noMesh = 0
# reference: https://www.delftstack.com/howto/python/number-of-keys-in-dictionary
# use for loop to count number of keys that have no MeSH terms in the dictionary
for key in alz_data.keys():
```

```
    if len(alz_data[key]["Mesh"]) == 0:
        alz_count_noMesh = alz_count_noMesh + 1
# calculate the fraction.
alz_total_count = len(alz_data.keys())
alz_fraction = alz_count_noMesh/alz_total_count
# reference: https://www.kite.com/python/answers/how-to-format-a-number-as-a-perc
alz_fraction_percentage = "{:.0%}".format(alz_fraction)
print (f"The number of Alzheimer's papers that have no MeSH terms is {alz_count_r

# use variable cancer_count_noMesh to save the number of cancer papers that have
cancer_count_noMesh = 0
# use for loop to count number of keys that have no MeSH terms in the dictionary
for key in cancer_data.keys():
    if len(cancer_data[key]["Mesh"]) == 0:
        cancer_count_noMesh = cancer_count_noMesh + 1
# calculate the fraction.
cancer_total_count = len(cancer_data.keys())
cancer_fraction = cancer_count_noMesh/cancer_total_count
# reference: https://www.kite.com/python/answers/how-to-format-a-number-as-a-perc
cancer_fraction_percentage = "{:.0%}".format(cancer_fraction)
print (f"The number of cancer papers that have no MeSH terms is {cancer_count_noM
```

1. Then separated data into alzheimer's data(alz_data) and cancer's data(cancer_data) based on content in the Query. Used variable alz_count_noMesh, cancer_count_noMesh to save the number of Alzheimer's/cancer papers that have no MeSH terms. Used for loop to count number of keys that have no MeSH terms in the dictionary alz_data/cancer_data. Then the fraction was calculated.

```
# reference: https://www.delftstack.com/howto/python/python-counter-most-common/
# import module Counter.
from collections import Counter
# function most_common_Mesh() was defined to find the 10 most common MeSH terms c
def most_common_Mesh(List,disease,n):
    # a Counter is a collection where elements are stored as dictionary keys, anc
    Mesh_count = Counter(List)
    # use the most_common() of Counter to find the most common elements of a list
    top_n_Mesh = Mesh_count.most_common(n)
    # reference: https://stackoverflow.com/questions/7558908/unpacking-a-list-tup
    # separate tuple top_n_Mesh into two lists top_n_Mesh_term, top_n_Mesh_freque
    top_n_Mesh_term, top_n_Mesh_frequency = list(zip(*top_n_Mesh))
    print(f"The {n} most common MeSH terms for the {disease} papers are: ")
    print(*top_n_Mesh_term, sep = '; ')
    return top_n_Mesh
```

2. Imported module Counter. A Counter is a collection where elements are stored as dictionary keys, and the key's counts are stored as dictionary values. Function most_common_Mesh() was defined to find the n most common MeSH terms of a specific disease when a list is entered. I used the most_common() of Counter to

find the most common elements of a list in python, and saved into the tuple top_n_Mesh. Then separated tuple top_n_Mesh into two lists top_n_Mesh_term, top_n_Mesh_frequency.

```
# generate list alz_Mesh, to save all the MeSH terms of Alzheimer's.
alz_Mesh = []
for key in alz_data.keys():
    # select only the MeSH terms that are not empty.
    if len(alz_data[key]["Mesh"]) != 0:
        # merge two lists.
        # reference: https://www.w3schools.com/python/gloss_python_join_lists.asp
        alz_Mesh.extend(alz_data[key]["Mesh"])
# run function most_common_Mesh() on list alz_Mesh.
alz_top_10_Mesh = most_common_Mesh(alz_Mesh, 'Alzheimers')

# generate list cancer_Mesh, same as above.
cancer_Mesh = []
for key in cancer_data.keys():
    if len(cancer_data[key]["Mesh"]) != 0:
        # merge two lists.
        # reference: https://www.w3schools.com/python/gloss_python_join_lists.asp
        cancer_Mesh.extend(cancer_data[key]["Mesh"])
cancer_top_10_Mesh = most_common_Mesh(cancer_Mesh, 'cancer')
```

Lists alz_Mesh, cancer_Mesh are generated to save all the MeSH terms of Alzheimer's/cancer. Run function most_common_Mesh() on these two lists.

```
# import libraries matplotlib and pandas.
import matplotlib.pyplot as plt
import pandas as pd

# create pandas dataframe to save the results of top 10 MeSH terms of Alzheimer's
# Then draw the bar chart.
df = pd.DataFrame(alz_top_10_Mesh, columns = ['MeSH','Frequency'])
plt.bar(df.MeSH, height = df.Frequency, edgecolor = "black")
plt.xlabel("MeSH terms",fontsize = 12)
plt.ylabel("Frequency",fontsize = 12)
plt.xticks(rotation = 90, fontsize=10)
plt.yticks(fontsize = 10)
plt.title("10 most common MeSH terms for Alzheimer's papers", fontsize = 12)
for x,y in enumerate(df.Frequency):
    plt.text(x, y+1, '%.0f'%y, ha = 'center')
plt.show()

# same as above.
df = pd.DataFrame(cancer_top_10_Mesh, columns = ['MeSH','Frequency'])
plt.bar(df.MeSH, height = df.Frequency, edgecolor = "black")
plt.xlabel("MeSH terms",fontsize = 12)
plt.ylabel("Frequency",fontsize = 12)
plt.xticks(rotation = 90, fontsize=10)
```

```
plt.yticks(fontsize = 10)
plt.title("10 most common MeSH terms for cancer papers", fontsize = 12)
for x,y in enumerate(df.Frequency):
    plt.text(x, y+1, '%.0f'%y, ha = 'center')
plt.show()
```

Imported libraries matplotlib and pandas to draw the bar charts.

```
# find the top 5 MeSH terms from Alzheimer's query, save into the list alz_top_5_
# same as cancer, save into the list cancer_top_5_Mesh.
alz_top_5_Mesh = Counter(alz_Mesh).most_common(5)
alz_top_5_Mesh = list(zip(*alz_top_5_Mesh))
alz_top_5_Mesh = list(alz_top_5_Mesh[0])
cancer_top_5_Mesh = Counter(cancer_Mesh).most_common(5)
cancer_top_5_Mesh = list(zip(*cancer_top_5_Mesh))
cancer_top_5_Mesh = list(cancer_top_5_Mesh[0])
# find the MeSH terms that both present in top 5 MeSH terms of Alzheimer's or car
same_Mesh = set(alz_top_5_Mesh)&set(cancer_top_5_Mesh)

# function count_paper_top_5_Mesh() was defined, to return the counts of papers t
def count_paper_top_5_Mesh():
    count_result = []
    # use nested for loop to traverse all the top 5 MeSH terms of Alzheimer's anc
    for elm1 in alz_top_5_Mesh:
        for elm2 in cancer_top_5_Mesh:
            # use variable count to record the counting numbers.
            count = 0
            for key in all_data.keys():
                # if the MeSH terms of one paper have two of top 5 MeSH terms, cc
                if elm1 in all_data[key]["Mesh"] and elm2 in all_data[key]["Mesh"
                    count = count + 1
            count_result_elm = (elm1,elm2,count)
            # result save into tuple count_result.
            count_result.append(count_result_elm)
    return count_result

count_result = count_paper_top_5_Mesh()
```

3. Found the top 5 MeSH terms from Alzheimer's query, saved into the list alz_top_5_Mesh. Same as cancer, saved into the list cancer_top_5_Mesh. Then found the MeSH terms that both present in top 5 MeSH terms of Alzheimer's or cancers, save into the set same_Mesh. Function count_paper_top_5_Mesh() was defined, to return the counts of papers that have two of top 5 MeSH terms. First used nested for loop to traverse all the top 5 MeSH terms of Alzheimer's and cancer. If the MeSH terms of one paper have two of top 5 MeSH terms, variable count increases by one. Result was saved into tuple count_result.

```
table_content = []
# extract only the counts, save into the list table_content.
for i in range(len(count_result)):
    table_content.append(count_result[i][2])
# reference: https://stackoverflow.com/questions/9671224/split-a-python-list-intc
# split list table_content into sublists in group of five.
table_content = [table_content[data:data + 5] for data in range(0, len(table_cont
table_content
```

Extracted only the counts, saved into the list table_content. Then split table_content into sublists in group of five. Dataframe df_table was created, in which the column name is top 5 MeSH terms of Alzheimer's. Data inside are the counts of papers having both the matching MeSH terms.

```
# reference: https://stackoverflow.com/questions/15514005/how-to-change-the-table
# reference: https://www.delftstack.com/howto/matplotlib/plot-table-using-matplot
# plot table using matplotlib.
fig, sub_axes = plt.subplots(1,1)
# create dataframe, in which the column name is top 5 MeSH terms of Alzheimer's.
df_table = pd.DataFrame(table_content, columns = alz_top_5_Mesh)
sub_axes.axis('off')
sub_axes.set_title("The top 5 MeSH terms in Alzheimer's and cancer papers", fonts
table = sub_axes.table(cellText = df_table.values, colLabels = df_table.columns,
table.set_fontsize(30)
table.scale(5,5)
```

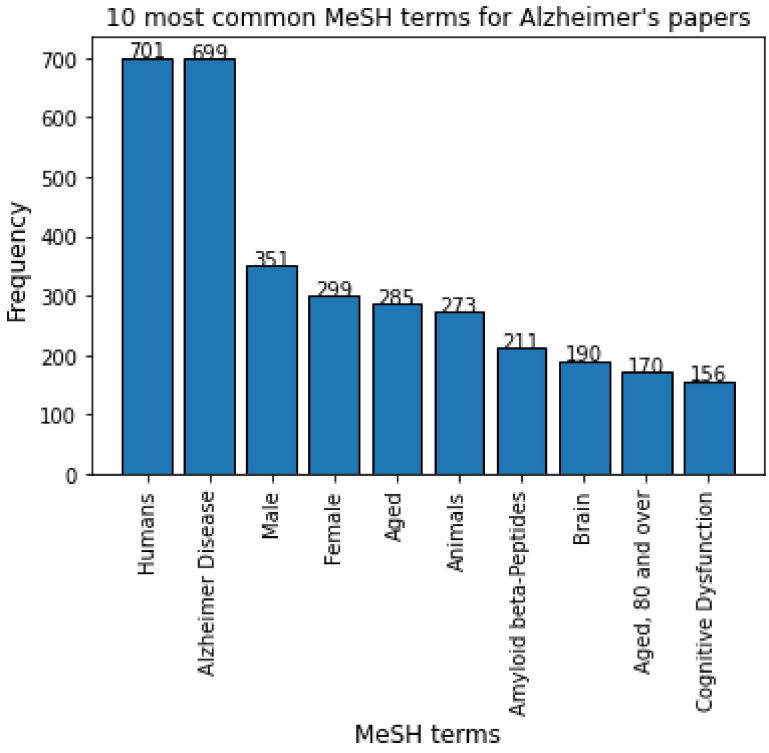
Table was plotted using matplotlib.

>> Question answer

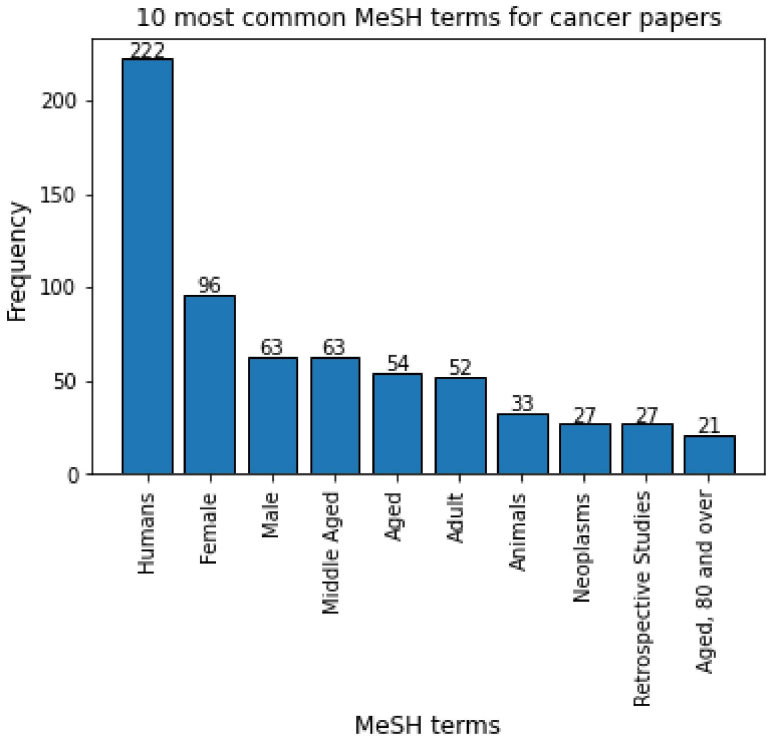
1. The number of Alzheimer's papers that have no MeSH terms is 164, which is 16% of total Alzheimer's papers. The number of cancer papers that have no MeSH terms is 758, which is 76% of total cancer papers.

The fraction of cancer papers have no MeSH terms is much more than that of Alzheimer's papers. Compared with the term "Alzheimer's", "cancer" is a much broader term, which is associated with multiple non-cancer diseases, affecting people no matter what their nationality, genders, ages are, thus it is difficult to summarize the MeSH terms that related to "cancer"(it is not useful to list all the terms).

2. The 10 most common MeSH terms for the Alzheimer's papers are: Humans; Alzheimer Disease; Male; Female; Aged; Animals; Amyloid beta-Peptides; Brain; Aged, 80 and over; Cognitive Dysfunction.



The 10 most common MeSH terms for the cancer papers are: Humans; Female; Male; Middle Aged; Aged; Adult; Animals; Neoplasms; Retrospective Studies; Aged, 80 and over.



3. The table with rows(top 5 MeSH terms from Alzheimer's query) and columns(top 5 MeSH terms from cancer query) is as below, values in the table are the counts of papers having both the matching MeSH terms.

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

Findings from the table:

(1) The most popular MeSH term is "Humans": As one of the top 5 MeSH terms from both Alzheimer's and cancer papers, 922 of the papaers has this MeSH term.

(2) While on the contrary, among all the top 5 MeSH terms from both Alzheimer's and cancer papers, "Female" remain to be the least popular: 109 of the papers has this MeSH term.

(3) According to top 5 MeSH terms in Alzheimer's or cancer papers: Aged people are more likely to have Alzheimer's, while both middle aged and aged people are more likely to have cancers. Compared with female(appears in 109 of papers), male are more likely to have Alzheimer's and cancers(appears in 413 of papers).

Limitations from the table: For all the top 5 Mesh terms in Alzheimer's or cancer papers, they are too general to describe a specific disease: 4 out of 5 Mesh terms are same regardless of disease.

From my perspective, although table could show the exact count of papers having both the matching MeSH terms, it is not a good choice for visualization. If using the network/sankey diagrams, I could get a better understanding of how various MeSH terms relate to each other.

>> Testing

To test the results in exercise 2(to prove the validity of function most_common_Mesh()), I used the JSON file paper_random_sample.json(in the 'Files' folder) generated in the exercise 1, which contains metadata of randomly selected elements from the dictionaries alz_data and cancer_data.

```
# testing code.
with open('paper_random_sample.json') as f:
    random_sample_data = json.load(f)

# separate data into sample_alz_data and sample_cancer_data.
sample_alz_data = {PubmedId: data for PubmedId, data in random_sample_data.items(
            if data["Query"] == "Alzheimers" or data["Query"] == ['Alzheimers','c
sample_cancer_data = {PubmedId: data for PubmedId, data in random_sample_data.ite
            if data["Query"] == "cancer" or data["Query"] == ['Alzheimers','cance
```

First, separated random_sample_data into sample_alz_data and sample_cancer_data.

```
# use for loop to find all the Mesh terms in the dictionaries sample_alz_data anc
mesh_alz = []
mesh_cancer = []
for key in sample_alz_data.keys():
    mesh_alz.extend(sample_alz_data[key]["Mesh"])
for key in sample_cancer_data.keys():
    mesh_cancer.extend(sample_cancer_data[key]["Mesh"])
print ("The MeSH terms of randomly selected Alzheimers' elements are")
print (*mesh_alz, sep = '; ')
print ("The MeSH terms of randomly selected cancer elements are")
print (*mesh_cancer, sep = '; ')
```

I used for loop to find all the Mesh terms in the dictionaries sample_alz_data and sample_cancer_data, results were saved into the lists mesh_alz and mesh_cancer.

```
# run most_common_Mesh() function to find the 10 most common MeSH terms in sample
top_5_Mesh_sampled_alz = most_common_Mesh(mesh_alz, 'Alzheimers',10)
top_5_Mesh_sampled_alz

# same for sample_cancer_data.
top_5_Mesh_sampled_cancer = most_common_Mesh(mesh_cancer, 'cancer',10)
top_5_Mesh_sampled_cancer
```

```
The MeSH terms of randomly selected Alzheimers' elements are
Alzheimer Disease; Disease Progression; Humans; Therapeutics; Adolescent; Adult; Aged; Aged, 80 and over; Alzheimer Disease;
Brain; Female; Humans; Image Interpretation, Computer-Assisted; Magnetic Resonance Imaging; Male; Middle Aged; Neural
Networks, Computer; Supervised Machine Learning; Young Adult; Alzheimer Disease; Amyloid; Amyloid beta-Peptides; Amyloid
beta-Protein Precursor; Animals; Brain; Connectome; Cortical Excitability; Disease Models, Animal; Hippocampus; Mice; Mice,
Transgenic; Parietal Lobe; Presenilin-2; Protein Aggregation, Pathological
The MeSH terms of randomly selected cancer elements are
Amyloid; Bence Jones Protein; Biomarkers; Biopsy; Diagnosis, Differential; Giant Cell Arteritis; Humans; Immunoglobulin
Light-chain Amyloidosis; Temporal Arteries; Drug Delivery Systems; Humans; Hyperthermia, Induced; Metal-Organic Frameworks;
Phototherapy; Precision Medicine
```

```
The 10 most common MeSH terms for the Alzheimers papers are:
Alzheimer Disease; Humans; Brain; Disease Progression; Therapeutics; Adolescent; Adult; Aged; Aged, 80 and over; Female
[('Alzheimer Disease', 3),
 ('Humans', 2),
 ('Brain', 2),
 ('Disease Progression', 1),
 ('Therapeutics', 1),
 ('Adolescent', 1),
 ('Adult', 1),
 ('Aged', 1),
 ('Aged, 80 and over', 1),
 ('Female', 1)]
```

```
The 10 most common MeSH terms for the cancer papers are:
Humans; Amyloid; Bence Jones Protein; Biomarkers; Biopsy; Diagnosis, Differential; Giant Cell Arteritis; Immunoglobulin
Light-chain Amyloidosis; Temporal Arteries; Drug Delivery Systems
[('Humans', 2),
 ('Amyloid', 1),
 ('Bence Jones Protein', 1),
 ('Biomarkers', 1),
 ('Biopsy', 1),
 ('Diagnosis, Differential', 1),
 ('Giant Cell Arteritis', 1),
 ('Immunoglobulin Light-chain Amyloidosis', 1),
 ('Temporal Arteries', 1),
 ('Drug Delivery Systems', 1)]
```

Then I run most_common_Mesh() function to find the 10 most common MeSH terms in sample_alz_data/sample_cancer_data. I manually checked the lists mesh_alz and mesh_cancer, it turns out that the results from top_5_Mesh_sampled_alz, top_5_Mesh_sampled_cancer are correct.

Exercise 3

Question

Machine learning and data visualization strategies generally work best on data that is numeric, but exercise 1 gave us text data, and indeed text is common. Fortunately, modern NLP algorithms powered by machine learning trained on massive datasets exist that can take words (e.g. word2vec) or titles and abstracts (e.g. SPECTER) and return a vector of numbers in a way that similar items are given similar vectors. Since we have titles and abstracts, let's use SPECTER. 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. (5 points) If you are familiar with SPECTER and wish to do it another way, that's great, if not here's one approach based on Links to an external site: Install pytorch (a deep learning library) by following the instructions here Install the huggingface transformers module: pip install transformers provides access to a number of pre-trained NLP language models.) Have your code load the SPECTER model (the first time you do this, it will take a bit to download the model; it will be stored locally for fast reuse later):

```
from transformers import AutoTokenizer, AutoModel
```

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

Process your dictionary of papers:

```
data = [paper["ArticleTitle"] + tokenizer.sep_token + paper["Ab
stractText"] for paper in papers.values()]
inputs = tokenizer(data, padding=True, truncation=True, return
_tensors="pt", max_length=512)result = model(**inputs)# take t
he first token in the batch as the embeddingembeddings = result.
last_hidden_state[:, 0, :].detach().numpy()
```

At this point, embeddings[i] is the 768-dim vector for the ith paper. Apply principal component analysis (PCA) to identify the first three principal components. (5 points) I suggest using the sklearn module, e.g.

```
from sklearn import decomposition
pca = decomposition.PCA(n_components=3)
embeddings_pca = pd.DataFrame(
    pca.fit_transform(embeddings),
    columns=['PC0', 'PC1', 'PC2']
)
embeddings_pca["query"] = [paper["query"] for paper in papers.v
alues()]
```

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) Repeat the above using LDA instead of PCA. In your commentary, be sure to compare PCA vs LDA. (10 points)

```
TODO: example
```

Solution

>> Code explanation

```
# import AutoTokenizer, AutoModel.
from transformers import AutoTokenizer, AutoModel

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

# import json, and load the file paper.json into all_data.
import json
with open('paper.json') as f:
    all_data = json.load(f)
```

Imported AutoTokenizer, AutoModel, and loaded model and tokenizer. Imported json, and load the file paper.json into dictionary all_data. Imported pandas and sklearn.

```
data = [paper["ArticleTitle"] + tokenizer.sep_token + paper["AbstractText"] for p
inputs = tokenizer([data[0]], padding=True, truncation=True, return_tensors="pt",
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="
    result = model(**inputs)
    # take the first token in the batch as the embedding. embeddings[i] is the 76
    embeddings = result.last_hidden_state[:, 0, :].detach().numpy()
    # reference: https://stackoverflow.com/questions/22662996/merge-numpy-arrays-
    embeddings_total = np.concatenate((embeddings_total, embeddings),axis = 0)

# import pandas and sklearn.
import pandas as pd
from sklearn import decomposition

# apply principal component analysis (PCA) to identify the first three principal
pca = decomposition.PCA(n_components=3)
embeddings_pca = pd.DataFrame(pca.fit_transform(embeddings_total),columns=['PC0',
embeddings_pca["Query"] = [paper["Query"] for paper in all_data.values()]
```

1. I used the code provided in the exercise 3 as references. For each paper, take the first token in the batch as the embedding. Then principal component analysis was applied to identify the first three principal components PC0, PC1, PC2.

```
# import library plotnine.
import plotnine as p9

# reference: https://datacarpentry.org/python-ecology-lesson/07-visualization-ggp
# use plotnine to draw the scatterplot between PC0 and PC1. Legends are three que
(p9.ggplot(data = embeddings_pca, mapping = p9.aes(x='PC0', y='PC1'))
+ p9.geom_point(p9.aes(x = 'PC0', y = 'PC1', color = 'Query'))
+ p9.labs(title = "Scatter plot for PC0 vs PC1 using PCA method"))

# use plotnine to draw the scatterplot between PC0 and PC2. Legends are three que
(p9.ggplot(data = embeddings_pca, mapping = p9.aes(x='PC0', y='PC2'))
+ p9.geom_point(p9.aes(x = 'PC0', y = 'PC2', color = 'Query'))
+ p9.labs(title = "Scatter plot for PC0 vs PC2 using PCA method"))

# use plotnine to draw the scatterplot between PC1 and PC2. Legends are three que
(p9.ggplot(data = embeddings_pca, mapping = p9.aes(x='PC1', y='PC2'))
+ p9.geom_point(p9.aes(x = 'PC1', y = 'PC2', color = 'Query'))
+ p9.labs(title = "Scatter plot for PC1 vs PC2 using PCA method"))
```

Imported library plotnine to draw the scatterplot between PC0 and PC1,PC0 and PC2, PC1 and PC2. Legends are three queries: Alzheimers, Alzheimers or cancer(outlier), cancer.

```
# import classifier LinearDiscriminantAnalysis and library numpy.
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import numpy as np

# create a one-dimensional vector y as target values.
y = []
for key in all_data.keys():
    # If query was Alzheimer's, y = 0; if query was Alzheimer's and cancer, y = 1
    if all_data[key]['Query'] == 'Alzheimers':
        y.append(0)
    elif all_data[key]['Query'] == 'Alzheimers or cancer':
        y.append(1)
    elif all_data[key]['Query'] == 'cancer':
        y.append(2)
y = np.array(y)

# linear discriminant analysis was applied to identify the first two principal cc
lda = LDA(n_components=2)
embeddings_lda = pd.DataFrame(lda.fit_transform(embeddings_total,y),columns=['PC0
embeddings_lda["Query"] = [paper["Query"] for paper in all_data.values()]
```

2. Imported classifier LinearDiscriminantAnalysis and library numpy. First I created a one-dimensional vector y as target values: If query was Alzheimer's, y = 0; if query was Alzheimer's and cancer, y = 1; if query was cancer, y = 2. Then linear discriminant analysis was applied to identify the first two principal components PC0 and PC1.

```
# use plotnine to draw the scatterplot between PC0 and PC1. Legends are three que
(p9.ggplot(data = embeddings_lda, mapping = p9.aes(x='PC0', y='PC1'))
+ p9.geom_point(p9.aes(x = 'PC0', y = 'PC1', color = 'Query'))
+ p9.labs(title = "Scatter plot for PC0 vs PC1 using LDA method"))
```

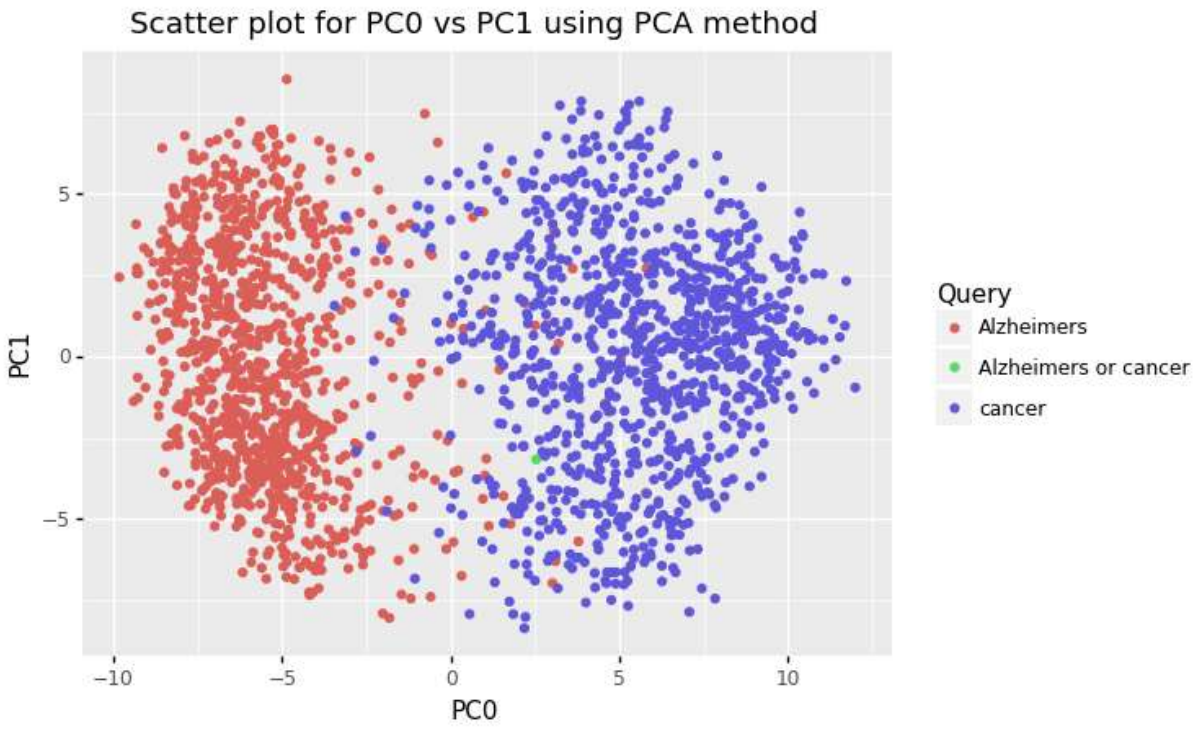
Used library plotnine to draw the scatterplot between PC0 and PC1.

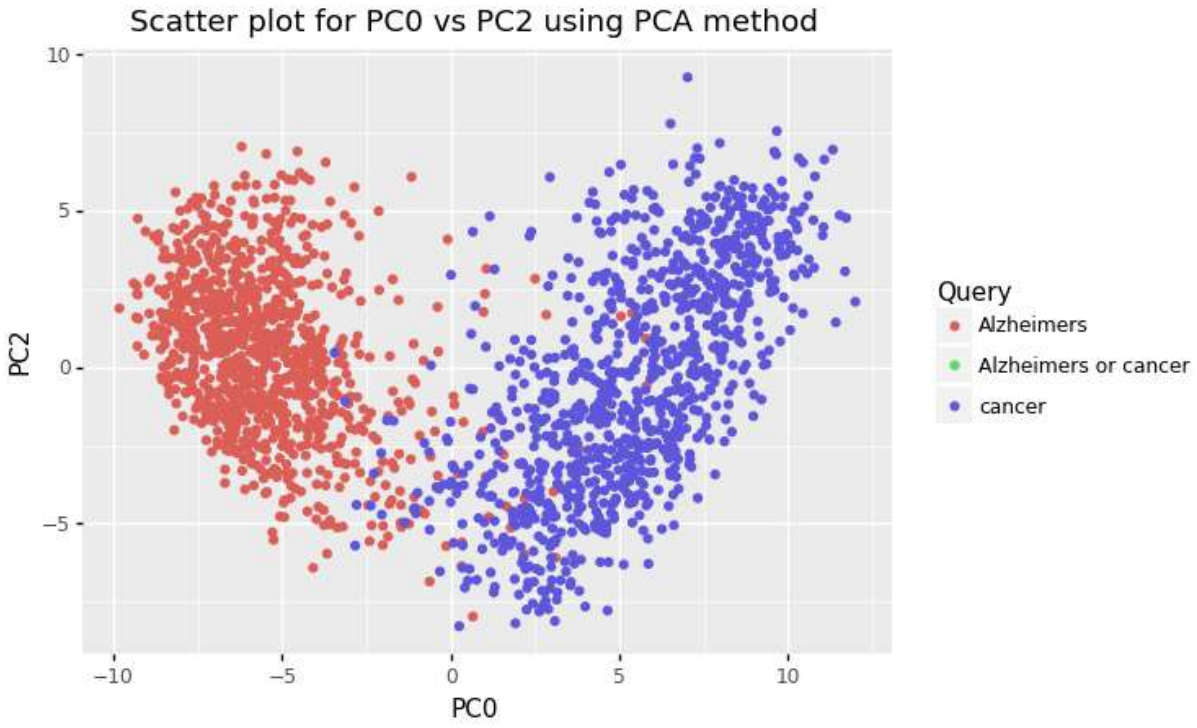
>> Question answer

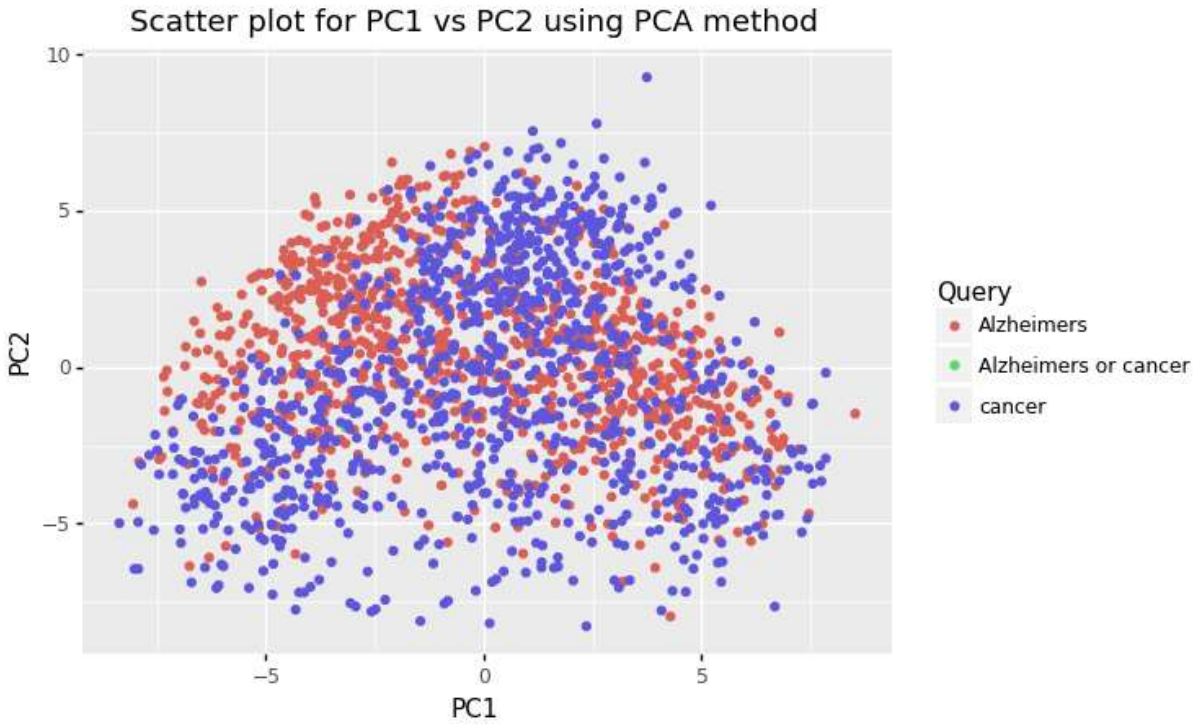
1. By labeling the PCA/LDA result with queries from the JSON file generated before, I could keep track of sources of papers, i.e, which paper came from searching for Alzheimers, which came from searching for cancer.

```
embeddings_pca["Query"] = [paper["Query"] for paper in all_data.values()]
```

2. 2D scatter plots for PC0 vs PC1, PC0 vs PC2, and PC1 vs PC2 are as follows, colored by the search query: 'Alzheimers', 'cancer', 'Alzheimers and cancer'.

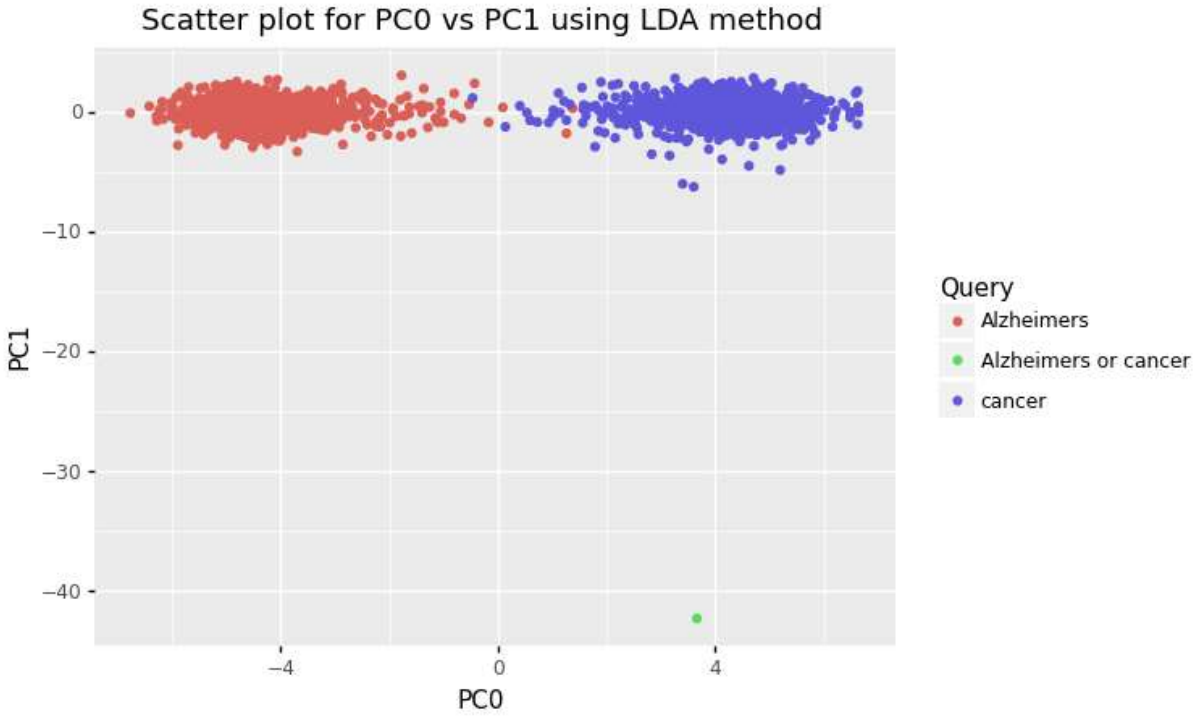






Among these three figures, PC0 vs PC1, PC0 vs PC2 show more obvious separation between different queries('Alzheimers', 'cancer', 'Alzheimers and cancer'). While PC1 vs PC2 cannot separate different queries. Note that the overlap(showed in green) could be seen from PC0 vs PC1.

3. For sklearn of LDA, the number of components n_components is the minimum of [number of classes - 1 and number of features]. Since I want the n_component to be at least 2, n_classes should be at least 3. That is why I chose to separate my data into three classes: For Alzheimer's, target value y = 0, for Alzheimer's and cancer, target value y = 1, for cancer, target value y = 2.



Differences between LDA and PCA:

The main goal of LDA is classification. It is aimed to maximize the separation of known categories(maximize variance between different projected classes). LDA needs labels since it wants to evaluate the accuracy of classification.

For PCA, it is aimed to maximize variance within each projected class.

Exercise 4

Question

In slides3, we introduced the merge sort, a recursive sorting algorithm that works by recursively splitting the data in two until there is at most one data point in a group to be sorted. (Groups of size 0 or 1 cannot be out of order.) Sorted sub-lists are then merged. See the implementation in the slides for details or feel free to search online for more information. Describe in words how you would parallelize this algorithm to work with two processes (5 points) and how you would validate the results and the speedup (5 points). Extra credit: Use the multiprocessing module to implement a 2 process parallel version of the merge sort from slides3. Demonstrate that your solution can achieve at least a speedup of 1.5x. (5 points)

>> Question answer

How to parallelize the merge sort algorithm:

Merge sort uses the idea of recursion. After the left and right part of a unsorted list become sorted sublists, the two sublists are merged into a sorted list.

Since the processing of the left and right part does not use the same critical resource, I could parallelize the processing of left and right parts. After parallelization, use join() method before merging to ensure the left and right sublists are in order.

How to validate the result and speedup:

I would compare the sorted result with normal merge sort, to see whether the parallelized one could get a right answer.

To compare the speedup, I would randomly generate n=10, 100, 1000, 10000, 100000, 100000, 200000, 300000, 400000, 500000 numbers. Run normal merge sort and parallelized merge sort separately, use time.time() to record the running time. Finally I would draw line graph(x-axis: number of unsorted numbers, y-axis: corresponding running time) and calculate speedup ratio to compare the speedup result.

Exercise 5

Question

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. (5 points) 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. (5 points) <-- This will be discussed in class on October 12. 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. (5 points)

>> Code explanation

```
# import libraries pandas and matplotlib.
import pandas as pd
import matplotlib.pyplot as plt

# load dataset and other corresponding files.
data_2020 = pd.read_csv("BRFSS_2020_data.csv")
state_FIPS = pd.read_csv("state_name_FIPS.csv")
income_value = pd.read_csv("income_value.csv")
education_value = pd.read_csv("education_level_value.csv")
```

Imported libraries pandas and matplotlib. The BRFSS dataset and other corresponding datasets(state_FIPS, income_value, education_value, which are used for data processing) were loaded.

```
# replace the _STATE column based on STATE_NAME, FIPS CODE from another dataframe
# reference: https://stackoverflow.com/questions/36413993/replace-column-values-i
data_2020['_STATE'] = data_2020['_STATE'].map(state_FIPS.set_index('_STATE')['STA

# count the frequencies of each state, save into the dataframe df_state_frequency
# reference: https://www.marsja.se/pandas-count-occurrences-in-column-unique-valu
# reference: https://stackoverflow.com/questions/47136436/python-pandas-convert-v
df_state_frequency = data_2020['_STATE'].value_counts().rename_axis('State').rese
df_state_frequency
```

1. In order to explore the state data('_STATE'), first I replaced the _STATE column based on STATE_NAME, FIPS CODE from another dataframe state_FIPS. Then I counted the frequencies of each state using value_counts(), and saved into the dataframe df_state_frequency.

```
# use matplotlib to draw the bar charts of participants' states.
# set figure size, x/y label names, x/y sticks, figure title.
plt.figure(figsize=(15,15))
```

```
plt.bar(df_state_frequency.State, height = df_state_frequency.Frequency, edgecolc
plt.xlabel("State",fontsize = 18)
plt.ylabel("Frequency",fontsize = 18)
plt.xticks(rotation = 90, fontsize= 13)
plt.yticks(fontsize = 10)
plt.title("Participants' states", fontsize = 20)
for x,y in enumerate(df_state_frequency.Frequency):
   plt.text(x, y+1, '%.0f'%y, ha = 'center', fontsize = 7)
plt.show()
```

I used matplotlib to draw the bar charts of participants' states. The figure size, x/y label names, x/y sticks, figure title were set.

```
# same for variable sex(_SEX).
# Replace the _SEX column: if value = 1, sex is male. if value = 2, sex is female
# reference: https://www.kite.com/python/answers/how-to-replace-column-values-in-
data_2020['_SEX'].replace({1:'male',2:'female'}, inplace = True)
df_sex_frequency = data_2020['_SEX'].value_counts().rename_axis('Sex').reset_inde
df_sex_frequency
# use matplotlib to draw the bar charts of participants' sexes.
plt.figure(figsize=(5,5))
plt.bar(df_sex_frequency.Sex, height = df_sex_frequency.Frequency, width = 0.3, e
plt.xlabel("Sex",fontsize = 18)
plt.ylabel("Frequency",fontsize = 18)
plt.xticks(fontsize= 15)
plt.yticks(fontsize = 15)
plt.title("Participants' sexes", fontsize = 20)
for x,y in enumerate(df_sex_frequency.Frequency):
   plt.text(x, y+1, '%.0f'%y, ha = 'center', fontsize = 13)
plt.show()

# same for variable income(_INCOMG).
data_2020['_INCOMG'] = data_2020['_INCOMG'].map(income_value.set_index('_INCOMG')
df_income_frequency = data_2020['_INCOMG'].value_counts().rename_axis('Income').r
df_income_frequency
# use matplotlib to draw the bar charts of participants' income.
plt.figure(figsize=(8,8))
plt.bar(df_income_frequency.Income,height = df_income_frequency.Frequency, edgecc
plt.xlabel("Income",fontsize = 18)
plt.ylabel("Frequency",fontsize = 18)
plt.xticks(rotation = 330, fontsize= 13)
plt.yticks(fontsize = 13)
plt.title("Participants' income", fontsize = 20)
for x,y in enumerate(df_income_frequency.Frequency):
   plt.text(x, y+1, '%.0f'%y, ha = 'center', fontsize = 12)
plt.show()

# same for variable education level(_EDUCAG').
data_2020['_EDUCAG'] = data_2020['_EDUCAG'].map(education_value.set_index('_EDUCA
df_education_frequency = data_2020['_EDUCAG'].value_counts().rename_axis('Educati
df_education_frequency
```

```
# use matplotlib to draw the bar charts of participants' education levels.
plt.figure(figsize=(8,8))
plt.bar(df_education_frequency.Education_level,height = df_education_frequency.Fr
plt.xlabel("Education level",fontsize = 18)
plt.ylabel("Frequency",fontsize = 18)
plt.xticks(rotation = 330, fontsize= 13)
plt.yticks(fontsize = 13)
plt.title("Participants' education level", fontsize = 20)
for x,y in enumerate(df_education_frequency.Frequency):
   plt.text(x, y+1, '%.0f'%y, ha = 'center', fontsize = 12)
plt.show()
```

Same for variable sex(_SEX), income(_INCOMG) and education level(_EDUCAG').

```
# to see whether there is duplicate data in the dataset.
data_2020_duplicate = data_2020[data_2020.duplicated()]
data_2020_duplicate
```

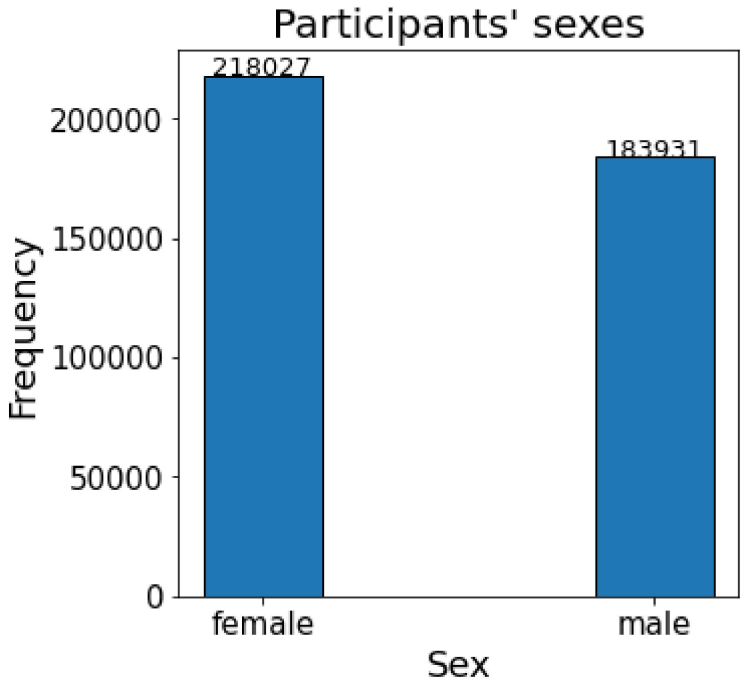
2. To do the data cleaning, first I checked whether there are duplicates in the dataset data_2020.

```
# count the missing values of each column in the dataset, save into the dataframe
data_2020_missing = data_2020.isnull().sum()
data_2020_missing = data_2020_missing.rename_axis('Variable').reset_index(name =
data_2020_missing

# in the data_2020_missing, only keep the variables that count_missing_values > 1
data_2020_missing = data_2020_missing[data_2020_missing.Count_missing_values > 10

# clean the dataset data_2020, so that it only contains variables that have < 100
for column in data_2020.columns:
    if column in list(data_2020_missing['Variable']):
        data_2020 = data_2020.drop(column, 1)
```

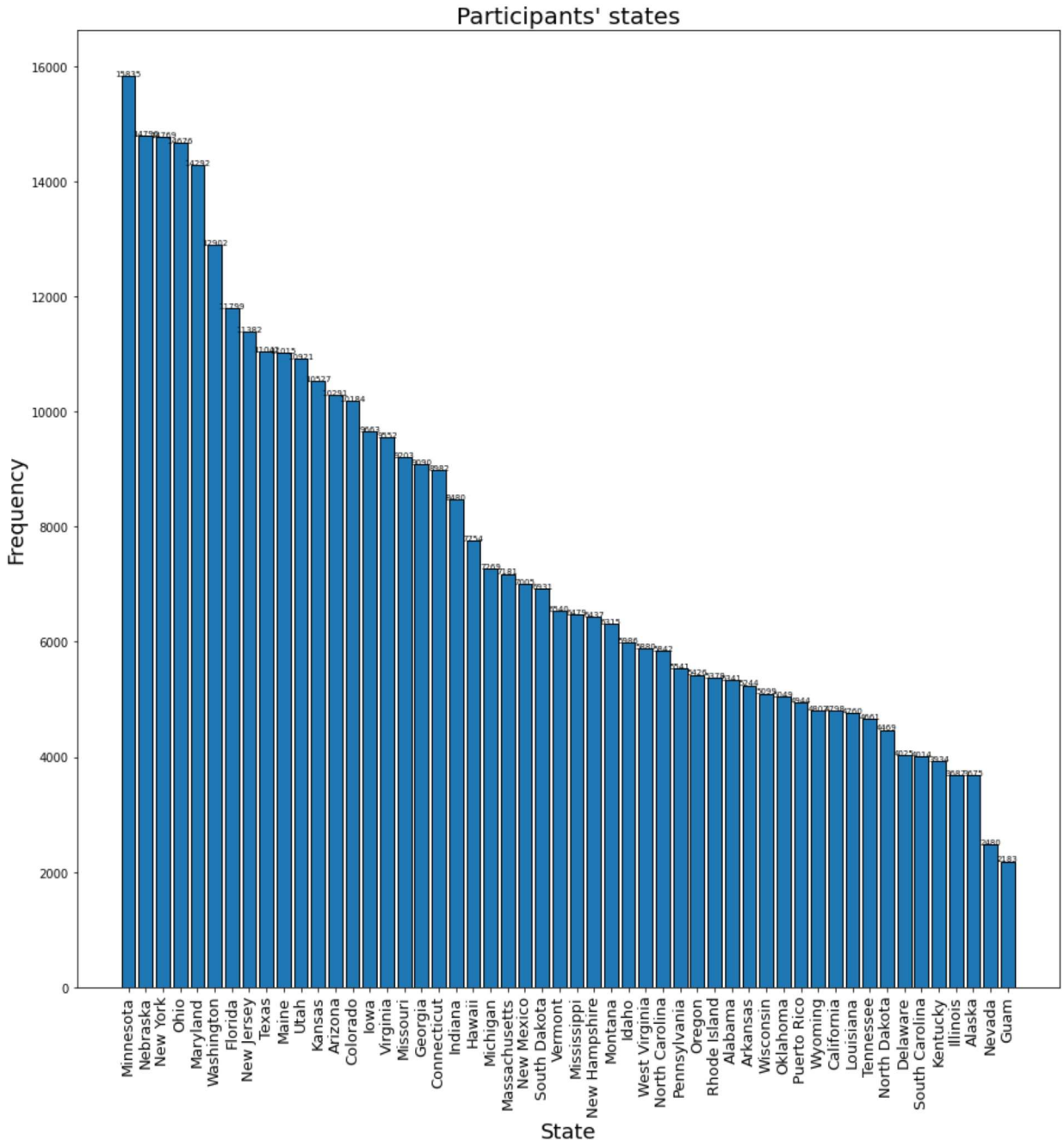
Then I counted the missing values of each column in the dataset, and saved into the dataframe data_2020_missing. In data_2020_missing, only kept the variables that count_missing_values > 1000. Then I cleaned the dataset data_2020, so that it only contains variables that have < 1000 missing values (i.e., drop the entire field if there are > 1000 missing values in this field).

>> Question answer

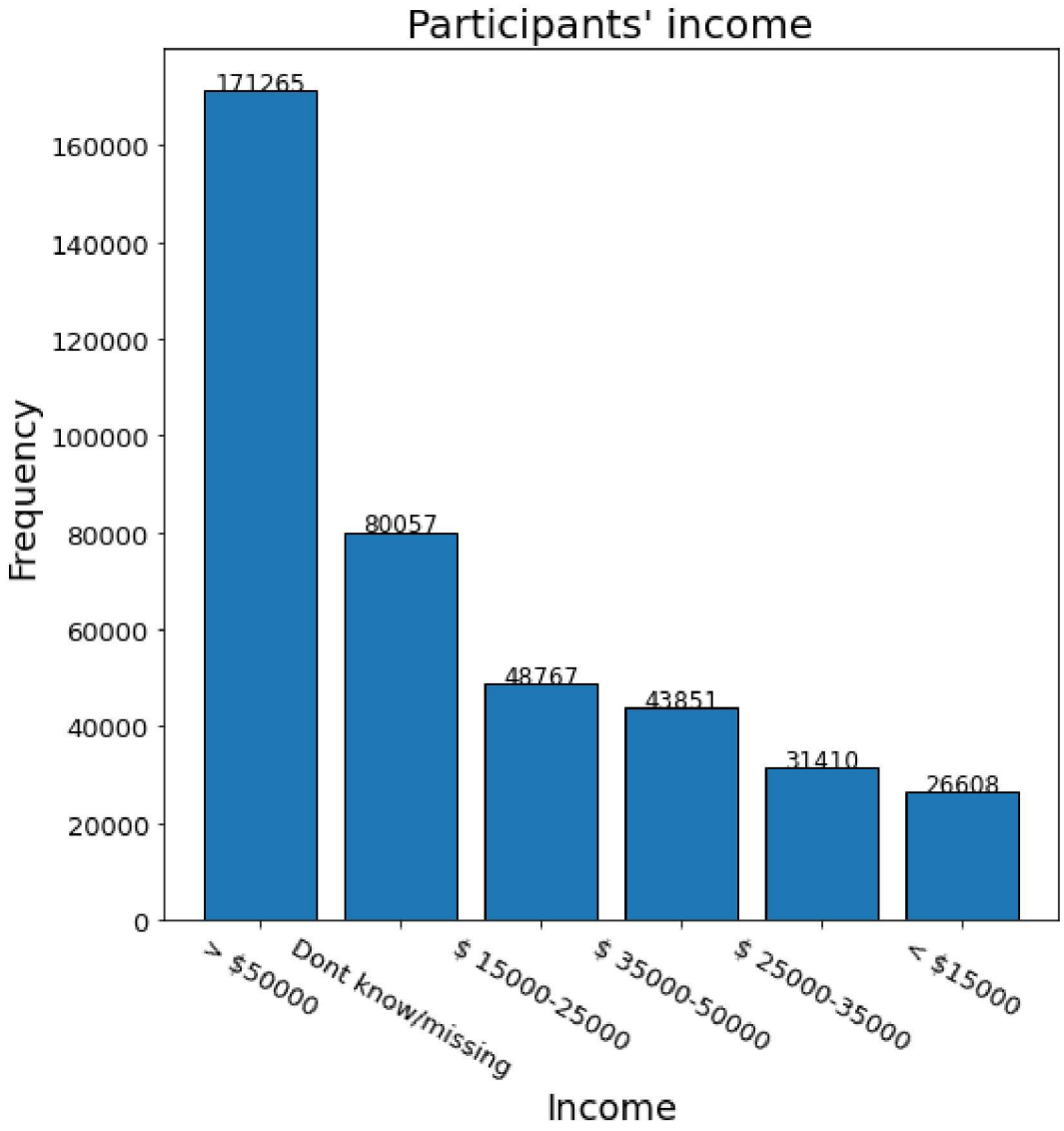
1.



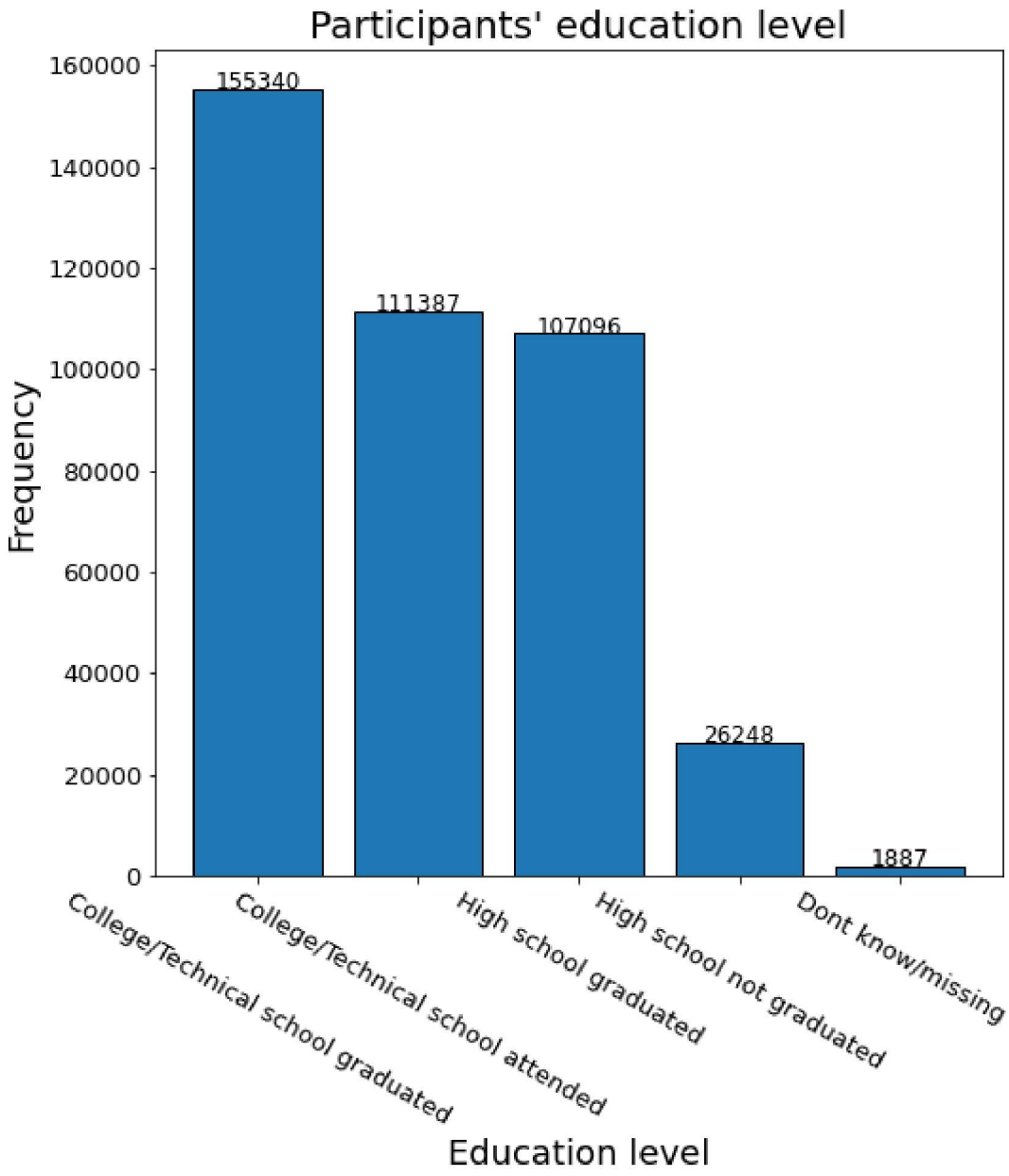
> From the barplot of participants' sex, we could see that among participants in the Behavioral Risk Factor Surveillance System (BRFSS), there are more females than males(1.185:1). In addition, BRFSS has collected sex data of all participants since there is no missing values.



> From the barplot of participants' state, we could see that among participants in the BRFSS, most people are from states such as Minnesota, Nebraska, New York, Ohio, and Maryland. This might cause selection bias since some most populated states(California, Texas, Florida, Pennsylvania) have fewer participants. This study was not conducted by using proportional sampling of state population.



> From the barplot of participants' income, we could see that among participants in the BRFSS, most people's annual household income are over $50,000. However, there are a large proportion of participants(n = 80057) who did not report their annual income, which could also cause bias to the study results.



> From the barplot of participants' education level, we could see that among participants in the BRFSS, most people are college/technical school graduated(n = 155340), college/technical school attended(n = 111387) and high school graduated(n = 107096). Since there are fewer missing values(n = 1887), the data seems to be reliable and could be used for further analysis.

2. Yes, there are missing values in my data. I used isnull().sum() to count the number of(or quantify) missing values in each column.

From my perspective, these missing values are MAR, since there is a correlation between the ages of population and missing values: Most of the missing values come from columns _CLNSCPY, _SGMSCPY, _SGMS10Y, _RFBLDS4, _STOLDNA, _VIRCOLN, _SBONTIM, _CRCREC1. According to the codebook, these variables are data target at respondents aged 50-75, e.g., _STOLDNA refers to respondents aged 50-75 who have had a stool DNA test within the past three years. For people whose age are not in the range of 50 to 75, these columns are filled with missing values.

3. I checked whether there are duplicates in the dataset(the answer is no). The strategy I used for missing values was to drop the entire field(column) if there are over 1000 missing values in this field.

Data source

Data of exercise 1-3 comes from Entrez programming utilities.

In exercise 3, I also used SPECTER, pytorch, transformers and sklearn.

Data of exercise 5 comes from 2020 BRFSS Data (SAS Transport Format).

Unfortunately, the dataset is too large to be uploaded(307Mb).

Maintainer

@QingyangYu0529

Name	Email	Organization
Qingyang Yu	qingyang.yu@yale.edu	Graduate student, Yale School of Public Health, Yale University