ReadMe | BIS634 HW3
Olina Zhu | qz258

Note that this readme file contains both analysis, code, and output of the assignment. The code for each exercise is attached below each question's analysis.

Question 1:

Although no written response is required for this question, I still want to mention that I received a tremendous amount of help from our TF Wenxin Xu and Professor McDougal. I was able to find the MeSH terms, but they came in very awkward formats: "mesh": "Alzheimer Diseasediagnostic imaging" (as taken straight from the Alzheimer.json file). This potentially inhibited correct analysis of MeSH terms in Q2.

Code and Output shown below:

```
1  # import libraries
   import pandas as pd
   import requests
   import xml.dom.minidom as m
   import xml.etree.ElementTree as et
   import time as time
   import json
```

```
2  # make request using get to obtain 1000 Alzheimer (al) id
   r_al_ids = requests.get("https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term=Alzheimers+AND+2019[pdat]&retmax=1000&retmode=xml")

   # parse returned response
   al_doc = m.parseString(r_al_ids.text)
   al_idlist = al_doc.getElementsByTagName("Id")

   # adding ids to id list
   al_ids = []
   i = 0
   while i < 1000:
       al_ids.append(al_idlist[i].childNodes[0].wholeText)
       i += 1
```

```
3  # get articles identified by ids through POST
   # create dictionary with parameters for POST
   al_param = {'db': 'pubmed', 'retmode': 'xml', 'id' : al_ids}
   r_al = requests.post("https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi", data = al_param)
   print(r_al.status_code)
```

```
   200
```

```
4  with open("headers", "wb") as f:
       f.write(r_al.text.encode('utf-8'))
```

ReadMe | BIS634 HW3
Olina Zhu | qz258

5
```
# function for reading xml file into dictionary
# matching each id with its title, abstract, query, and mesh
def xml_to_dict(root, dict, query):
    for paper in root:
        # getting pmid
        id = paper.find(".//PMID")
        id = et.tostring(id, method = 'text').decode()
        # print(id)

        # inner dictionary where al_dict{id} = vals
        vals = {}

        # getting ["title", "abstract", "query", "mesh"]
        title = paper.find(".//ArticleTitle")
        if title is None:
            title = paper.find(".//BookTitle")
            if title is None:
                title = "N/A"
        title = et.tostring(title, method = "text").decode()
        vals["title"] = title

        abstract = paper.find(".//Abstract")
        if abstract is None:
            vals["abstract"] = "N/A"
        else:
            abstract = et.tostring(abstract, method = 'text').decode()
            vals["abstract"] = abstract

        vals["query"] = query

        mesh = paper.find(".//MeshHeading")
        if mesh is None:
            vals["mesh"] = "N/A"
        else:
            mesh = et.tostring(mesh, method = 'text').decode()
            vals["mesh"] = mesh

        dict[id] = vals
    return dict
```

6
```
# read metadata for Alzheimer
al_root = et.fromstring(r_al.text)
al_dict = {}
al_dict = xml_to_dict(al_root, al_dict, "Alzheimer's")

# and save information from post request as json file
with open("Alzheimer.json", "w") as f:
    json.dump(al_dict, f, indent=4)
```

7
```
# make request using get to obtain 1000 cancer (cn) id
r_cn_ids = requests.get("https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term=cancer+AND+2019[pdat]&retmax=1000&retmode=xml")

# parse returned response
cn_doc = m.parseString(r_cn_ids.text)
cn_idlist = cn_doc.getElementsByTagName("Id")

# adding ids to id list
cn_ids = []
i = 0
while i < 1000:
    cn_ids.append(cn_idlist[i].childNodes[0].wholeText)
    i += 1
```

8
```
# get articles identified by ids through POST
# create dictionary with parameters for POST

cn_param = {'db': 'pubmed', 'retmode': 'xml', 'id': cn_ids}
r_cn = requests.post("https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi", data = cn_param)

print(r_cn.status_code)
```

```
200
```

9
```
# read metadata for Alzheimer
cn_root = et.fromstring(r_cn.text)
cn_dict = {}
cn_dict = xml_to_dict(cn_root, cn_dict, "Cancer")

# and save information from post request as json file
with open("Cancer.json", "w") as f:
    json.dump(cn_dict, f, indent=4)
```

ReadMe | BIS634 HW3
Olina Zhu | qz258

Question 2:

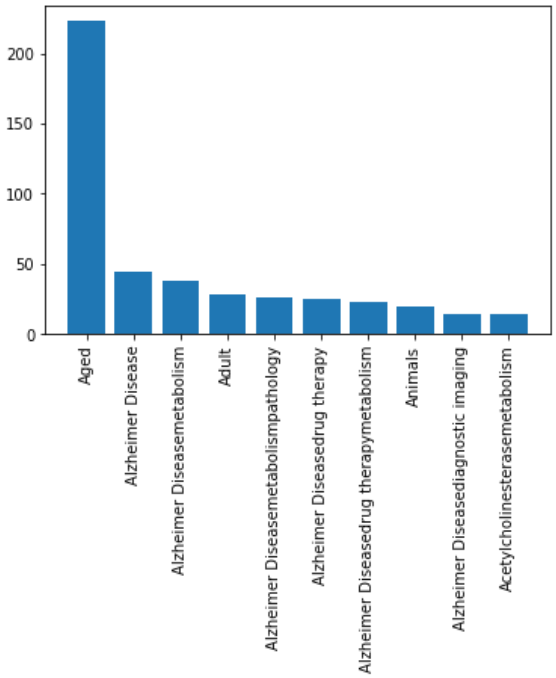
There are 164 Alzheimer papers that have no MeSH terms. That is 16.4 percent of all Alzheimer papers examined.

There are 756 cancer papers that have no MeSH terms. That is 75.6 percent of all cancer papers examined.

In comparison, there are significantly (>4.6 times) more cancer papers that are missing MeSH terms than are there Alzheimer papers in the examined subset. This could potentially be because of the significantly more intense examination and efforts needed to determine the exact state of cancer, e.g., where it originated from, where did it spread to, what cell types are affected, etc., which could potentially result in different names and MeSH terms for the illness, therefore could be too laborious to include as the spread in data would be high enough to decrease its analytical value. However, for Alzheimer's, we know that is must be neurological and cognition-related, hence is easier and potentially more meaningful to categorize.

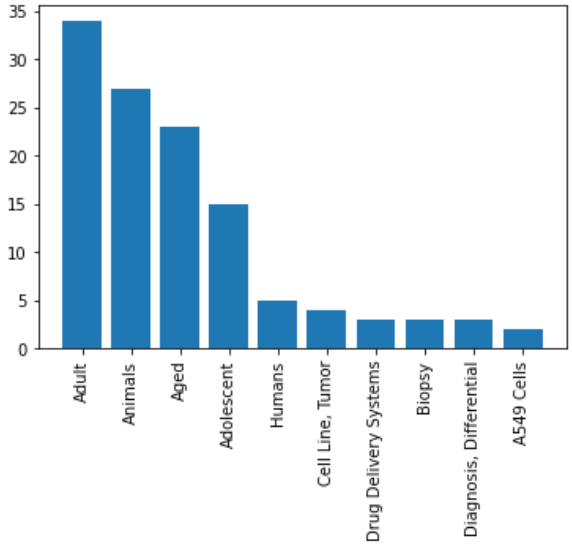
10 Most Common MeSH terms for:

Alzheimer's: 'Aged', 'Alzheimer Disease', 'Alzheimer Diseasemetabolism', 'Adult', 'Alzheimer Diseasemetabolismpathology', 'Alzheimer Diseasedrug therapy', 'Alzheimer Diseasedrug therapymetabolism', 'Animals', 'Alzheimer Diseasediagnostic imaging', 'Acetylcholinesterasemetabolism'



ReadMe | BIS634 HW3
Olina Zhu | qz258

Cancer: 'Adult', 'Animals', 'Aged', 'Adolescent', 'Humans', 'Cell Line, Tumor', 'Drug Delivery Systems', 'Biopsy', 'Diagnosis, Differential', and 'A549 Cells'



* Note that awkward merging of terms probably prohibited correct analysis of MeSH term frequency.

Table Comparing MeSH Terms

	Adult	Animals	Aged	Adolescent	Humans
Aged	257	250	246	238	228
Alzheimer Disease	79	72	68	60	50
Alzheimer Diseasemetabolism	72	65	61	53	43
Adult	62	55	51	43	33
Alzheimer Diseasemetabolismpathology	60	53	49	41	31

One apparent limitation of this table is that the MeSH terms for Alzheimer papers are somewhat merged, therefore the frequencies of correct versions of the terms are not correctly calculated and analyzed. A finding is that the MeSH term ‘Aged’ appeared very frequently in both Alzheimer papers and cancer papers, and for apparent reasons occurred together with ‘Adult’. This could potentially be because both illnesses tend to occur more frequently in older individuals. ‘Adolescent’ appeared relatively more frequently in cancer papers than in Alzheimer papers, which is somewhat true to the characteristics of the disease themselves. I am very surprised that the merged MeSH terms actually occurred more than once in the two queries, as they seemed to be unlikely errors, but perhaps this is because of a systematic error on my cleaning of data hence those with consecutive identical MeSH terms also end up in the same, incorrect state.

ReadMe | BIS634 HW3
Olina Zhu | qz258

Code and Output shown below:

22
```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import requests
import xml.dom.minidom as m
import xml.etree.ElementTree as et
import time as time
import json
from matplotlib import pyplot as plt
```

23
```
# import data for Alzheimer articles
al_dict = {}
with open("Alzheimer.json") as f:
    al_dict = json.load(f)
```

24
```
# import data for cancer articles
cn_dict = {}
with open("Cancer.json") as f:
    cn_dict = json.load(f)
```

25
```
# create dataframe from al_dict
al_df = pd.DataFrame.from_dict(al_dict, orient='index')
al_df.head()
```

25

	title	abstract	query	mesh
33939349	Electroconvulsive Therapy for the Treatment of...	Dementia refers to a state of cognitive impair...	Alzheimer's	N/A
33841007	Gintonin facilitates brain delivery of donepez...	Gintonin is a ginseng-derived exogenous G-prot...	Alzheimer's	N/A
33627920	Bayesian Scalar on Image Regression With Nonig...	Medical imaging has become an increasingly imp...	Alzheimer's	N/A
33463291	The Structural Basis of Amyloid Strains in Alz...	Amyloid fibrils represent one of the defining ...	Alzheimer's	Alzheimer Disease
33323224	Healthy ageing through internet counselling in...	Although web-based interventions have been pro...	Alzheimer's	Aged

ReadMe | BIS634 HW3
Olina Zhu | qz258

```
26 # create dataframe from cn_dict
   cn_df = pd.DataFrame.from_dict(cn_dict, orient='index')
   cn_df.head()
```

26

	title	abstract	query	mesh
34590506	Diarylpentanones from the root of Wikstroemia ...	One new diarylpentanone, 4(S)-hydroxy-1, 5-dip...	Cancer	A549 Cells
34539049	Re-evaluating standards of human subjects prot...	This study addresses ethical questions about c...	Cancer	N/A
34539046	Inference for L-estimators of location using a...	In this note we propose a new semi-parametric ...	Cancer	N/A
34493369	The ITGB6 gene: its role in experimental and c...	Integrin αvβ6 is a membrane-spanning...	Cancer	Animals
34460208	Advancing the Science of Cancer in Latinos	Cancer is the leading cause of death among Lat...	Cancer	N/A

```
27 # count number of Alzheimer papers that have no MeSH terms
   # note that the finding of mesh terms in Q1 is incorrect hence no mesh terms exist in dataframes, but th
   al_no_mesh = len(al_df[al_df['mesh'] == "N/A"])
   print("There are", al_no_mesh, "Alzheimer papers that have no MeSH terms.")
   print("That is", al_no_mesh/1000, "of all Alzheimer papers examined.")
```

```
There are 164 Alzheimer papers that have no MeSH terms.
That is 0.164 of all Alzheimer papers examined.
```

```
28 # count number of cancer papers that have no MeSH terms
   cn_no_mesh = len(cn_df[cn_df['mesh'] == "N/A"])
   print("There are", cn_no_mesh, "cancer papers that have no MeSH terms.")
   print("That is", cn_no_mesh/1000, "of all cancer papers examined.")
```

```
There are 756 cancer papers that have no MeSH terms.
That is 0.756 of all cancer papers examined.
```

ReadMe | BIS634 HW3
Olina Zhu | qz258

29
```
# function for finding 10 most frequent mesh terms
# returns a dictionary sorted by keys in descending order
def mesh_frequency(df):
    mesh_counts = {}
    meshes = df['mesh']
    for m in meshes:
        if m not in mesh_counts: mesh_counts[m] = 0
        mesh_counts[m] += 1
    mesh_counts = sorted(mesh_counts.items(), key=lambda item: item[1], reverse=True)
    return mesh_counts
```

30
```
# mesh terms in al_df ordered by frequency
al_top_10_mesh = mesh_frequency(al_df)
al_top_10_mesh = al_top_10_mesh[:11]
index_na = [i[0] for i in al_top_10_mesh].index('N/A')
na_mesh_al = al_top_10_mesh.pop(index_na)
print(al_top_10_mesh)
```

```
[('Aged', 223), ('Alzheimer Disease', 45), ('Alzheimer Diseasemetabolism', 38), ('Adult', 28), ('Alzheim
```

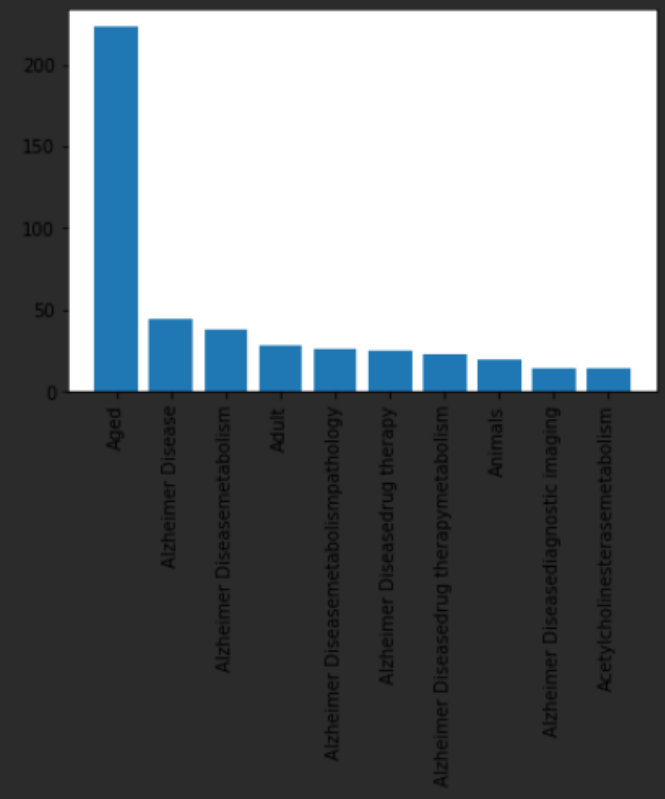
31
```
# plot frequency of 10 most common al_df mesh terms
al_x = [i[0] for i in al_top_10_mesh]
print(al_x)
al_y = [i[1] for i in al_top_10_mesh]
print(al_y)

plt.bar(range(len(al_top_10_mesh)), al_y, tick_label=al_x)
plt.xticks(rotation='vertical')
plt.show()
```

```
['Aged', 'Alzheimer Disease', 'Alzheimer Diseasemetabolism', 'Adult', 'Alzheimer Diseasemetabolismpathol
[223, 45, 38, 28, 26, 25, 23, 20, 14, 14]
```

ReadMe | BIS634 HW3
Olina Zhu | qz258

```
31  # plot frequency of 10 most common al_df mesh terms
    al_x = [i[0] for i in al_top_10_mesh]
    print(al_x)
    al_y = [i[1] for i in al_top_10_mesh]
    print(al_y)

    plt.bar(range(len(al_top_10_mesh)), al_y, tick_label=al_x)
    plt.xticks(rotation='vertical')
    plt.show()
```

```
['Aged', 'Alzheimer Disease', 'Alzheimer Diseasemetabolism', 'Adult', 'Alzheimer Diseasemetabolismpathol
[223, 45, 38, 28, 26, 25, 23, 20, 14, 14]
```



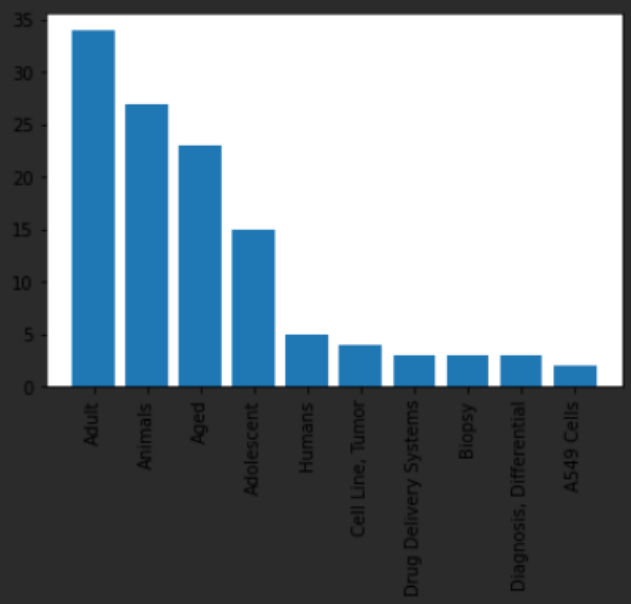
ReadMe | BIS634 HW3
Olina Zhu | qz258

```
32 # 10 most frequent mesh terms in cn_df
   cn_top_10_mesh = mesh_frequency(cn_df)
   cn_top_10_mesh = cn_top_10_mesh[:11]
   index_na = [i[0] for i in cn_top_10_mesh].index('N/A')
   na_mesh = cn_top_10_mesh.pop(index_na)
   print(cn_top_10_mesh)
```

```
[('Adult', 34), ('Animals', 27), ('Aged', 23), ('Adolescent', 15), ('Humans', 5), ('Cell Line, Tumor', 4
```

```
33 # plot frequency of 10 most common cn_df mesh terms
   cn_x = [i[0] for i in cn_top_10_mesh]
   print(cn_x)
   cn_y = [i[1] for i in cn_top_10_mesh]
   print(cn_y)

   plt.bar(range(len(cn_top_10_mesh)), cn_y, tick_label=cn_x)
   plt.xticks(rotation='vertical')
   plt.show()
```

```
['Adult', 'Animals', 'Aged', 'Adolescent', 'Humans', 'Cell Line, Tumor', 'Drug Delivery Systems', 'Biops
[34, 27, 23, 15, 5, 4, 3, 3, 3, 2]
```



34
```
# table comparing number of articles from cn_df and al_df that have both matching mesh terms as specifie
al_top_5_mesh = al_x[:5]
al_top_5_mesh_freq = al_y[:5]

cn_top_5_mesh = cn_x[:5]
cn_top_5_mesh_freq = cn_y[:5]
print(al_top_5_mesh, '\n', cn_top_5_mesh)
print(al_top_5_mesh_freq, '\n', cn_top_5_mesh_freq)

al_cn_mesh = pd.DataFrame(index=al_top_5_mesh, columns=cn_top_5_mesh)

i = 0
while i < 5:
    j = 0
    while j < 5:
        al_cn_mesh.iloc[i][j] = al_top_5_mesh_freq[i] + cn_top_5_mesh_freq[j]
        j += 1
    i += 1

al_cn_mesh.head()
```

```
['Aged', 'Alzheimer Disease', 'Alzheimer Diseasemetabolism', 'Adult', 'Alzheimer Diseasemetabolismpathol
 ['Adult', 'Animals', 'Aged', 'Adolescent', 'Humans']
[223, 45, 38, 28, 26]
 [34, 27, 23, 15, 5]
```

34

	Adult	Animals	Aged	Adolescent	Humans
Aged	257	250	246	238	228
Alzheimer Disease	79	72	68	60	50
Alzheimer Diseasemetabolism	72	65	61	53	43
Adult	62	55	51	43	33
Alzheimer Diseasemetabolismpathology	60	53	49	41	31

ReadMe | BIS634 HW3
Olina Zhu | qz258

Question 3:

I attempted this problem on google collab for an easier-to-work-with environment. I was not able to complete this question, will attempt again on my own time.

Code and Output shown below:

```
!pip install transformers -q
```

```
⊡[K     |████████████████████████████████| 3.1 MB 4.5 MB/s
⊡[K     |████████████████████████████████| 59 kB 4.4 MB/s
⊡[K     |████████████████████████████████| 596 kB 43.6 MB/s
⊡[K     |████████████████████████████████| 3.3 MB 28.7 MB/s
⊡[K     |████████████████████████████████| 895 kB 66.8 MB/s
⊡[?25h
```

```
import torch
from transformers import AutoTokenizer, AutoModel
from sklearn import decomposition
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import json as json
import pandas as pd
import seaborn as sns
from google.colab import files
```

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

```
Downloading: 0%| | 0.00/321 [00:00<?, ?B/s]
Downloading: 0%| | 0.00/612 [00:00<?, ?B/s]
Downloading: 0%| | 0.00/217k [00:00<?, ?B/s]
Downloading: 0%| | 0.00/112 [00:00<?, ?B/s]
Downloading: 0%| | 0.00/419M [00:00<?, ?B/s]
```

```
# load json file for Alzheimer's into dictionary
al_json = files.upload()
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Alzheimer.json to Alzheimer.json
```

```
# load data into dataframe
with open("Alzheimer.json") as f:
    al_dict = json.load(f)

al_df = pd.DataFrame.from_dict(al_dict, orient = 'index')
al_df.head()
```

	title	abstract	query	mesh
33939349	Electroconvulsive Therapy for the Treatment of...	Dementia refers to a state of cognitive impair...	Alzheimer's	N/A
33841007	Gintonin facilitates brain delivery of donepez...	Gintonin is a ginseng-derived exogenous G-prot...	Alzheimer's	N/A
33627920	Bayesian Scalar on Image Regression With Nonig...	Medical imaging has become an increasingly imp...	Alzheimer's	N/A
33463291	The Structural Basis of Amyloid Strains in Alz...	Amyloid fibrils represent one of the defining ...	Alzheimer's	Alzheimer Disease
33323224	Healthy ageing through internet counselling in...	Although web-based interventions have been pro...	Alzheimer's	Aged

ReadMe | BIS634 HW3
Olina Zhu | qz258

```
16 # load json file for cancer into dictionary
   al_json = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Cancer.json to Cancer.json
```

```
17 # load data into dataframe
   with open("Cancer.json") as f:
       cn_dict = json.load(f)

   cn_df = pd.DataFrame.from_dict(cn_dict, orient = 'index')
   cn_df.head()
```

17

	title	abstract	query	mesh
34590506	Diarylpentanones from the root of Wikstroemia ...	One new diarylpentanone, 4(S)-hydroxy-1, 5-dip...	Cancer	A549 Cells
34539049	Re-evaluating standards of human subjects prot...	This study addresses ethical questions about c...	Cancer	N/A
34539046	Inference for L-estimators of location using a...	In this note we propose a new semi-parametric ...	Cancer	N/A
34493369	The ITGB6 gene: its role in experimental and c...	Integrin αvβ6 is a membrane-spanning...	Cancer	Animals
34460208	Advancing the Science of Cancer in Latinos	Cancer is the leading cause of death among Lat...	Cancer	N/A

```
23 # merge dataframes
   df = al_df.merge(cn_df, how='outer')
   print("there are", len(df), "entries in df")
   df.head()
```

```
there are 2000 entries in df
```

23

	title	abstract	query	mesh
0	Electroconvulsive Therapy for the Treatment of...	Dementia refers to a state of cognitive impair...	Alzheimer's	N/A
1	Gintonin facilitates brain delivery of donepez...	Gintonin is a ginseng-derived exogenous G-prot...	Alzheimer's	N/A
2	Bayesian Scalar on Image Regression With Nonig...	Medical imaging has become an increasingly imp...	Alzheimer's	N/A
3	The Structural Basis of Amyloid Strains in Alz...	Amyloid fibrils represent one of the defining ...	Alzheimer's	Alzheimer Disease
4	Healthy ageing through internet counselling in...	Although web-based interventions have been pro...	Alzheimer's	Aged

```
24 # loop through papers
```

ReadMe | BIS634 HW3
Olina Zhu | qz258

Question 4:

Parallelization:

To parallelize the merge sort algorithm using 2 processes, I would find the tasks in merge sort that could be independently, i.e., one task does not rely on or feed into another.

The two processes could be each assigned to half (N/2) items in the N-item array at the very first recursion. Each process could then spawn threads that further break down each half of the list into halves, i.e., N/4, N/8, ..., 1.

Note that subsequent bisections of the two N/2-item arrays could occur independently of each other because no shared memory or reliance on each other's output is present.

Once a process has reached the point of producing arrays of 1 item from the initial N/2-item list, it begins to merge them back together in a sorted manner regardless of whether the other process has completed dividing its own N/2 items into single-item arrays. The first item of left array is compared with that of right array, and inserted into output array accordingly, and such comparison in order between the left and right arrays occur until all items have been added to the output array. Note that elements from the same subarray are never compared with each other, and elements in each of the two processes are not compared until the final step.

In the final step, parallelization ends and only one process is needed to merge the two sorted N/2-item arrays together.

Validation of Results:

I would validate the results by running the same array in a traditional (non-parallel) version of merge sort and a parallel version of merge sort. If the two results are identical, then we know that the parallelization produces valid results, but we do not yet know whether it produces a speed-up from this test. I would run multiple arrays of different properties (length, repeating numbers, odd/even items, etc.) to eliminate errors in edge cases.

Validation of Speed-up:

I would utilize the time package in python to track the runtime of my program. Again, I would run both a traditional (non-parallel) version and a parallel version of merge sort, but each implementation has a start_time = time.time() call in the very beginning of the code and a time.time() - start_time at the end of execution such that the duration of execution is returned.

Moreover, I would run the two versions of merge sorts repeatedly on varying sizes of arrays, and plot each version's performance on a size-of-array v.s. runtime line graph. An average could be taken for 5 runs of each array-size, such that an overview of the standard error of runtime could also be visualized along with the trend. From the graph we will be able to:

1. Determine whether there is a speed up using parallelization on merge sort;

2. Determine the trend of increase/speed-up relative to traditional version: is the increase linearly increasing, constant, or exponentially increasing, or shrinking as the size of array increases; and
3. Big O of parallelized version and compare it to that of traditional version.

ReadMe | BIS634 HW3
Olina Zhu | qz258

Question 5:

Some of the initial exploration is shown in the previous homework assignment. The code from HW2 is reproduced in Q5.jpynb with additional analysis. Note that original code for HW2 is modified such that the "Date" column is now of type "datetime". The first four figures (2 tables, 2 graphs) are not included in the below, and Figure I below refers to the 5th figure in the output, where Figure I is the first graph newly generated for HW3. All new graphs and their code are shown below, those from HW2 are in the “Code and Output” section.

```
190 # plotting the number of cases by time
    # note that distplot does not work well with datetime formats, hence a lineplot is used
    dates = data.groupby('Date')['ID'].count().reset_index()
    dates.head(10)
    date_case_plt = sns.lineplot(x='Date', y='ID', data=dates)
    date_case_plt.set_xlabel("Date (Year)")
    date_case_plt.set_ylabel("Count (Death)")
```

```
190 Text(0, 0.5, 'Count (Death)')
```

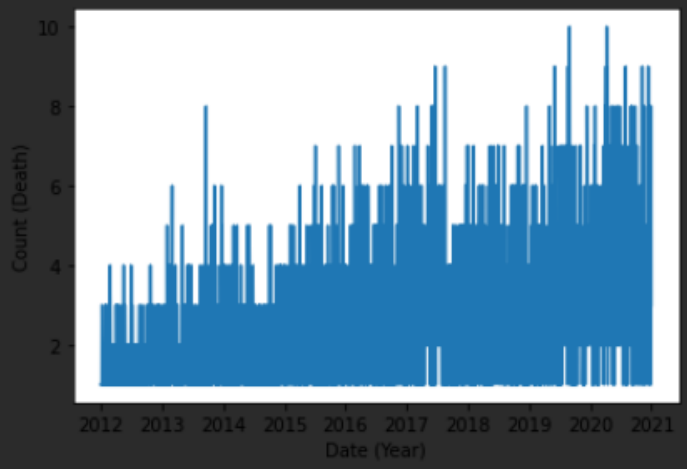


Figure I: Lineplot showing the number of deaths over time from 2012 to 2021. The units shown on the x-axis is years, but the data points themselves are specific to days, hence a very messy graph showing an overall increase throughout the years. I tried to use barplot, histplot, and displot instead to generate columns, but it seems that the datetime format does not work very well with the axes of those graph formats.

ReadMe | BIS634 HW3
Olina Zhu | qz258

```
191 # missing value analysis
    # clean dataset - NaN values in type of drug columns are to be interpreted as 'N' for No and not missing
    # only keep NaN if all entries in row for columns in type_of_drug is NaN
    type_of_drug = data[["Heroin", "Cocaine", "Fentanyl", "Fentanyl Analogue", "Oxycodone", "Oxymorphone", "
    row = 0
    col_num = len(type_of_drug.columns)

    while row < len(type_of_drug):
        num_nan = type_of_drug.iloc[row].isna().sum()
        if num_nan != col_num: type_of_drug.iloc[row].fillna("N", inplace=True)
        row += 1

    type_of_drug.head()
```

191

	Heroin	Cocaine	Fentanyl	Fentanyl Analogue	Oxycodone	Oxymorphone	Ethanol	Hydrocodon
0	N	N	N	N	N	N	N	N
1	Y	N	N	N	N	N	N	N
2	Y	N	N	N	N	N	N	N
3	Y	Y	N	N	N	N	N	N
4	Y	N	N	N	N	N	N	N

```
192 # updating data with type_of_drug
    data[["Heroin", "Cocaine", "Fentanyl", "Fentanyl Analogue", "Oxycodone", "Oxymorphone", "Ethanol", "Hydr
    data.head()
```

192

	ID	Date	Date Type	Age	Sex	Race	Description of Injury	Injury Place	Cause of Death	Other Significant Conditions	...	Benzodiazepine	Methadone	
0	12-0187	2012-07-17	DateofDeath	34.0	Female	White	Huffed Propellant	Other	1,1-Difluoroethane Toxicity	NaN	...	N	N	I
1	12-0258	2012-10-01	DateofDeath	51.0	Male	White	Injection	Residence	Heroin Toxicity	NaN	...	N	N	I
2	13-0146	2013-04-28	DateofDeath	28.0	Male	White	Substance Abuse	Other	Acute Heroin Toxicity	NaN	...	N	N	I
3	14-0150	2014-04-06	DateofDeath	46.0	Male	White	Injection	Other	Heroin and Cocaine Intoxication	NaN	...	N	N	I

Figures II, III: Figure II shows a cleaned version of the type of drug columns (18 in total), where all NaN values in these columns are to be interpreted as 'N' for No and not missing if there exists at least 1 non-NaN value in the row, hence NaN is only kept if all entries in row for type of drug columns is NaN. Figure III shows the integration of this cleaned dataset for type of drug into the whole dataset. This helps to remove confounding of missing value analysis, as prior to such cleaning the crude number and proportion of missing values in these type of drug columns are incredibly high, whereas persumeably the intention of the publishers is to use the columns as tick-charts and simply left False values as blank.

```
193 # count missing values and the proportion of each column that is missing
    nan_dict = {}
    cols = data.columns
    for col in cols:
        nan_dict[col] = [data[col].isna().sum(), round(data[col].isna().sum()/len(data[col]), 3)]
    nan_dict
```

```
193 {'ID': [0, 0.0],
     'Date': [2, 0.0],
     'Date Type': [2, 0.0],
     'Age': [3, 0.0],
     'Sex': [11, 0.001],
     'Race': [23, 0.003],
     'Description of Injury': [789, 0.103],
     'Injury Place': [79, 0.01],
     'Cause of Death': [0, 0.0],
     'Other Significant Conditions ': [7083, 0.922],
     'Heroin': [35, 0.005],
     'Cocaine': [35, 0.005],
     'Fentanyl': [35, 0.005],
     'Fentanyl Analogue': [35, 0.005],
     'Oxycodone': [35, 0.005],
     'Oxymorphone': [35, 0.005],
     'Ethanol': [35, 0.005],
     'Hydrocodone': [35, 0.005],
     'Benzodiazepine': [35, 0.005],
     'Methadone': [35, 0.005],
     'Amphet': [35, 0.005],
     'Tramad': [35, 0.005],
     'Morphine (Not Heroin)': [35, 0.005],
     'Hydromorphone': [35, 0.005],
     'Xylazine': [35, 0.005],
     'Other': [35, 0.005],
     'Opiate NOS': [35, 0.005],
     'Any Opioid': [35, 0.005]}
```

Figure IV: A dictionary showing the counts of missing values for each column, in the format: "column name": [crude number of missing values in column, proprtion of missing values in column]. Note that proportions are rounded to 3 decimal places, so 0.001 indicates a number rounded to 0.1%.

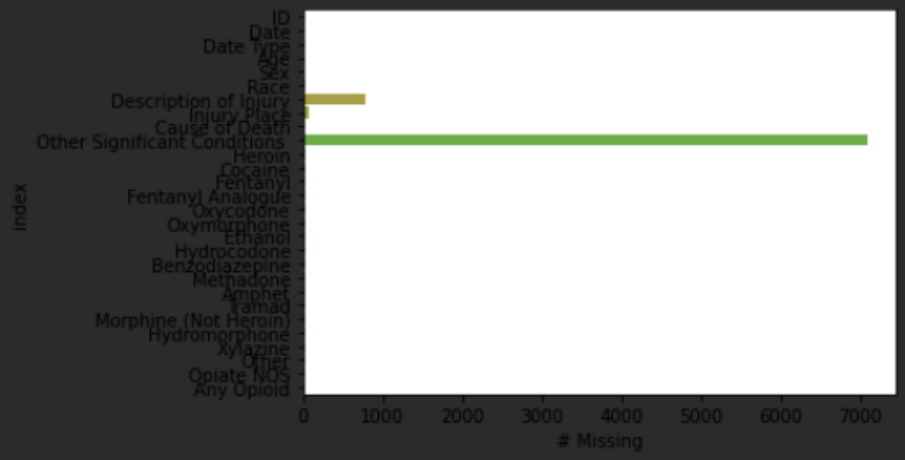
```
194 nan_df = pd.DataFrame.from_dict(nan_dict, orient='index', columns=['# Missing', 'Proportion of Column Da
    nan_df.head(10)
```

194

	index	# Missing	Proportion of Column Data Missing
0	ID	0	0.000
1	Date	2	0.000
2	Date Type	2	0.000
3	Age	3	0.000
4	Sex	11	0.001
5	Race	23	0.003
6	Description of Injury	789	0.103

Figure V: A dataframe constructed to show the information in Figure IV in a more organized, plottable way.

```
195 # plot missing values graph by crude number
    missing_crude_plt = sns.barplot(data=nan_df, x='# Missing', y='index')
```




```
196 # plot missing values graph by proportion of total column entries
    missing_prop_plt = sns.barplot(data=nan_df, x='Proportion of Column Data Missing', y='index')
```

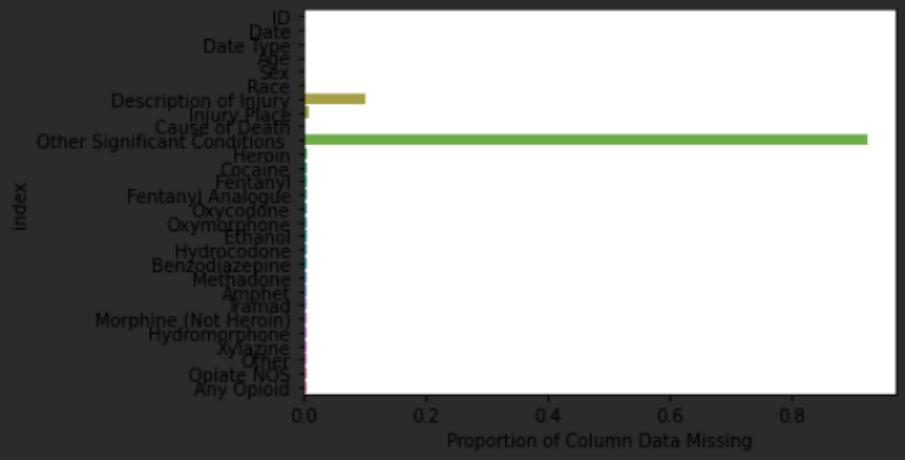
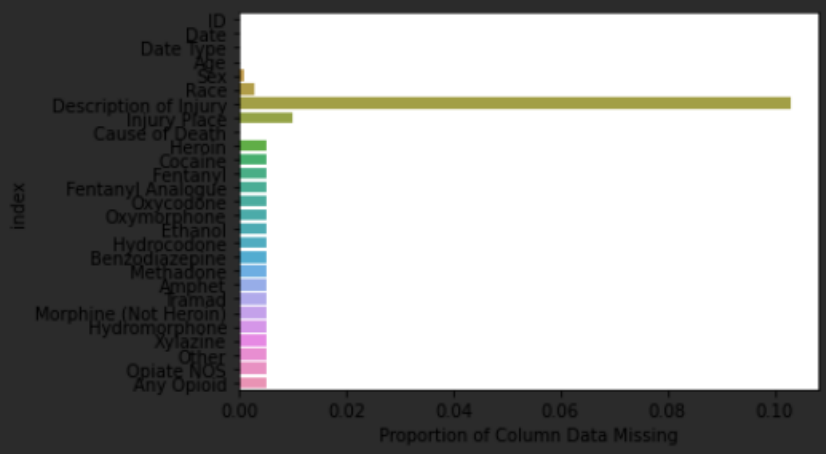


Figure VI (top): A plot of missing values by crude number.The graph was made horizontal such that the labels could show clearly. Moreover, note that the column for 'Other Significant Conditions' is extremely high, and this applies to figure VII as well, but this is very amibugous data - it is difficult to say whether missing values here indicates none observed or confirmed no other conditions. Therefore, for further analysis, this variable was removed, as it contains so many missing values its analysis is likely to lack statistical power anyways.

Figure VII (bottom): A graph similar to figure VI but showing proportion of column values missing rather than crude number.

```
197 # obtain more detailed view on previous graph by removing the other significant conditions column
    missing_prop_mag_plt = sns.barplot(data=nan_df.drop([9]), x='Proportion of Column Data Missing', y='inde
```




```
204 # examining potential associations between description of injury and other variables
    injury_assoc = data[data['Description of Injury'].isna()]
    IA_age_gb = injury_assoc.groupby('Age')['ID'].count()
    IA_age_gb.head(100)
```

```
204 Age
    17.0     2
    18.0     3
    19.0     5
    20.0     6
    21.0    13
    22.0     7
    23.0    15
    24.0    16
    25.0    18
    26.0    22
    27.0    13
    28.0    25
    29.0    20
    30.0    17
    31.0    15
    32.0    16
    33.0    22
    34.0    14
    35.0    24
    36.0    19
    37.0    16
    38.0    22
```

Figure VIII: A graph showing the information in figure VII but removing 'Other Significant Conditions' to obtain a more detailed view of the lower-valued columns. It seems that most columns have relatively small proportions of values missing except 'Description of Injury'and 'Injury Place'.

ReadMe | BIS634 HW3
Olina Zhu | qz258

```
199 IA_sex_gb = injury_assoc.groupby('Sex')['ID'].count()
    IA_sex_gb

199 Sex
    Female    214
    Male      573
    Name: ID, dtype: int64

200 IA_race_gb = injury_assoc.groupby('Race')['ID'].count()
    IA_race_gb

200 Race
    Asian, Other         2
    Black               56
    Chinese              1
    Hispanic, Black      3
    Hispanic, White     84
    Other                3
    Unknown              3
    White              635
    Name: ID, dtype: int64

201 IA_date_gb = injury_assoc.groupby('Date')['ID'].count()
    IA_date_gb = IA_date_gb.sort_values(ascending=False)
    IA_date_gb.head(50)

201 Date
    2015-11-21    7
    2015-07-05    7
    2015-10-18    6
    2015-04-04    6
    2015-12-17    6
    2015-06-28    6
    2015-10-21    6
    2015-08-12    6
    2015-06-09    5
    2015-07-26    5
    2015-09-18    5
    2015-10-23    5
    2015-02-09    5
    2015-11-01    5
    2015-06-22    5
    2015-05-14    5
    2015-07-08    5
    2015-11-04    5
    2015-02-04    5
    2015-07-12    5
    2015-07-31    5
```

```
202 IA_date_gb.head(100)

202 Date
    2015-11-21    7
    2015-07-05    7
    2015-10-18    6
    2015-04-04    6
    2015-12-17    6
                 ..
    2015-11-18    3
    2015-06-19    3
    2015-06-16    3
    2015-06-14    3
    2015-10-26    3
    Name: ID, Length: 100, dtype: int64

203 IA_date_gb.head(500)

203 Date
    2015-11-21    7
    2015-07-05    7
    2015-10-18    6
    2015-04-04    6
    2015-12-17    6
                 ..
    2015-07-01    1
    2015-07-06    1
    2015-07-10    1
    2012-10-29    1
    2020-12-20    1
    Name: ID, Length: 349, dtype: int64
```

MCAR/MAR/MNAR: Analysis was done on the relationship between the missing values in 'Description of Injury' (DOI) and 4 other variables: 'Age', 'Sex', 'Race', and 'Date'.

Age: It seems that most of the missing values for DOI is aggregated between ages 20-60, and relatively few data are missing for other age groups. This could potentially be because more cases within the age range 20-60 occured, hence a higher likelihood of having an increased crude number of missing data being associated with this age range.

Sex: Men are more than two times as likely to have missing data in terms of crude numbers, but if we look at the age_gender_plt plot (orange-blue line plot at the top), then we see that more male cases are recorded than female cases, hence a crude number comparison may not be fair in concluding that men are more likely to experience missing values in this dataset.

ReadMe | BIS634 HW3
Olina Zhu | qz258

Race: White people cases contain significantly more missing DOI data than other races, and Hispanic White people and Black people came in second and third respectively, but are both much less than the data for White people. This could be due to the same fallacy of base numbers as described above.

Date: Most of the missing data from this category came from 2015, and the density for missing data in 2015 is significantly higher than other years, as shown by the 3 calls to view the pattern of dates appearing in the dataset sorted in descending order for number of occurrences. The top 50 missing DOI date-values are in 2015, and this trend continued into the visible parts of the top 100 entries, and it is only in the top 500 entries that we begin to see appearance of other years, but with significantly lower occurrence.

Due to the above analysis, I would say that DOI data is MAR because although it could be predicted by date to some extent, we can only say that if a DOI value is missing, it is more likely to be in the year 2015, but cannot be predicted by any other value, and even this prediction is not reliable because we did not calculate the percentage of data that are missing in 2015, which could be the next step to validate an association (or lack thereof) between missing DOI and Date.

Code and Output from HW2 shown below, all those for HW3 are included above after each figure or analysis:

```
185 # import libraries
    import pandas as pd
    import numpy as np
    import seaborn as sns

    # import data
    # data URL: https://catalog.data.gov/dataset/accidental-drug-related-deaths-2012-2018
    # Public Access: This dataset is intended for public access and use
    # Publisher: data.ct.gov
    data = pd.read_csv("Accidental_Drug_Related_Deaths_2012-2020.csv")
    data.head()
```

185

	ID	Date	Date Type	Age	Sex	Race	Residence City	Residence County	Residen Sta
0	12-0187	7/17/2012	DateofDeath	34.0	Female	White	MAHOPAC	PUTNAM	NaN
1	12-0258	10/1/2012	DateofDeath	51.0	Male	White	PORTLAND	MIDDLESEX	NaN
2	13-0146	4/28/2013	DateofDeath	28.0	Male	White	NaN	NaN	NaN
3	14-0150	4/6/2014	DateofDeath	46.0	Male	White	WATERBURY	NaN	NaN
4	14-0183	4/27/2014	DateofDeath	52.0	Male	White	NEW LONDON	NaN	NaN

5 rows × 42 columns

ReadMe | BIS634 HW3
Olina Zhu | qz258

186
```
# drop redundant variables
# set format of Date column to month, day, year, source: https://www.kite.com/python/answers/how-to-chan
data = data.drop(columns=['Residence City', 'Residence County', 'Residence State', 'Death City', 'Death
data.head()
```

186

	ID	Date	Date Type	Age	Sex	Race	Description of Injury	Injury Place	Cause of Death	Other Significant Conditions	...	Benzodiazepine	Methado
0	12-0187	7/17/2012	DateofDeath	34.0	Female	White	Huffed Propellant	Other	1,1-Difluoroethane Toxicity	NaN	...	NaN	NaN
1	12-0258	10/1/2012	DateofDeath	51.0	Male	White	Injection	Residence	Heroin Toxicity	NaN	...	NaN	NaN
2	13-0146	4/28/2013	DateofDeath	28.0	Male	White	Substance Abuse	Other	Acute Heroin Toxicity	NaN	...	NaN	NaN
3	14-0150	4/6/2014	DateofDeath	46.0	Male	White	Injection	Other	Heroin and Cocaine Intoxication	NaN	...	NaN	NaN
4	14-0183	4/27/2014	DateofDeath	52.0	Male	White	Substance Abuse	Unknown	Acute Heroin Intoxication	NaN	...	NaN	NaN

5 rows × 28 columns

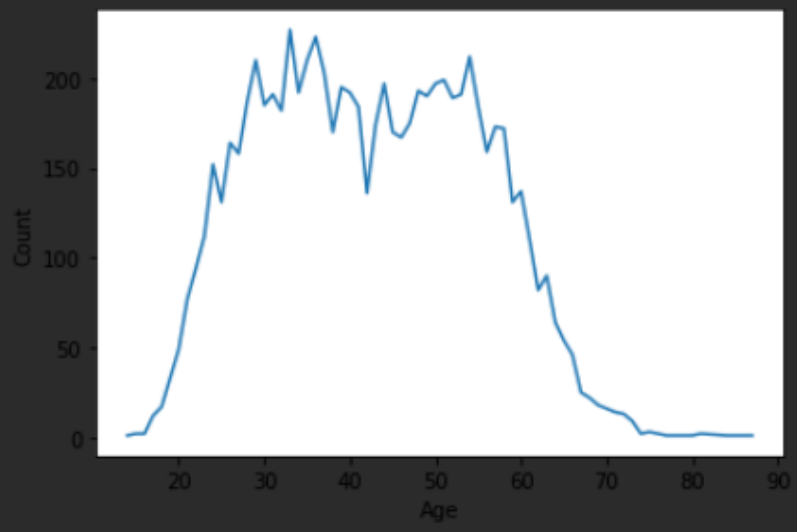
187
```
# overall dataset description
print("There are", len(data), "rows in the dataset and", len(data.columns), "variables.")

# set type of variable of Date column to Date
data['Date'] = pd.to_datetime(data['Date'])
data['Date'].dtype
```

```
There are 7679 rows in the dataset and 28 variables.
```

187
```
dtype('<M8[ns]')
```

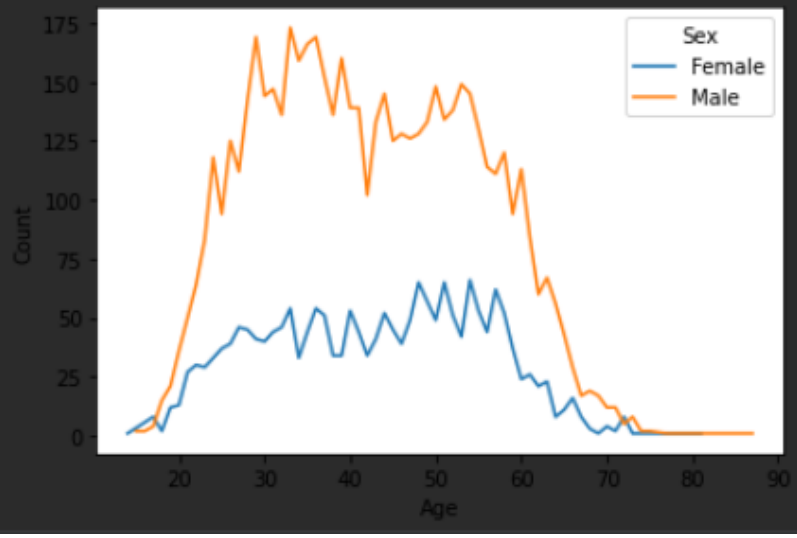
```
188 # finding relationship between age and accidental drug-caused death
    ages = data.groupby('Age')['ID'].count().reset_index()
    age_plt = sns.lineplot(data=ages, x='Age', y='ID')
    age_plt.set(ylabel='Count')
    age_plt
```

```
188 <AxesSubplot:xlabel='Age', ylabel='Count'>
```




```
189 # find relationship between age and accidental drug-caused death by gender
    age_gender = data.groupby(['Sex', 'Age'])['ID'].count().reset_index()
    age_gender = age_gender[age_gender.Sex != 'Unknown']
    age_gender_plt = sns.lineplot(data=age_gender, x='Age', y='ID', hue='Sex')
    age_gender_plt.set(ylabel='Count')
    age_gender_plt
```

```
189 <AxesSubplot:xlabel='Age', ylabel='Count'>
```



ReadMe | BIS634 HW3
Olina Zhu | qz258

Resources:

https://docs.python-requests.org/en/latest/user/quickstart/

https://www.ncbi.nlm.nih.gov/books/NBK3837/

https://www.geeksforgeeks.org/get-post-requests-using-python/

https://www.w3schools.com/python/ref_requests_post.asp