REPORT ANALYSIS OF VAERS DATA

Name:Shusanket Basyal

TASK1 : Creating a dataset for COVID-19 Vaccination

The Vaccine Adverse Events Reporting System(VAERS) contains vital information on adverse events following vaccinations. Task1 involves creating a new dataset that contains only data related to COVID-19 vaccinations. Each row in this dataset contains all the fields from VAERSData, VAERSVAX, and VAERSSYMPTOMS. The resulting dataset has been saved as “VAERS_COVID_DataAugust2023.csv”.In this task, I created a dataset for COVID-19 vaccinations by extracting the necessary data from various CSV files containing information related to vaccinations, symptoms, and demographics. The code is structured into several functions and steps, each serving a specific purpose in the data extraction process:

1. **vaersIDHashmapCovid(filepath):** In this function, I read a specified CSV file (specified by filepath) and extracted data related to COVID-19 vaccinations. The result is a hashmap where each VAERS_ID (Vaccine Adverse Event Reporting System ID) is a key, and the corresponding value is a list of attributes specific to the COVID-19 vaccine for that VAERS_ID. The attributes are extracted from the CSV file, and I stored the header information in the headinglist variable.

2. **vaersSymptomsHashmapCovid(hasmap, filepath)**: In this function, I read a CSV file containing symptom data (specified by filepath) and extracted symptom information for records corresponding to the provided hashmap, such as vaersVaxCOVIDID2020. This function returned a hashmap where each key is a VAERS_ID, and the value is a list of symptom attributes. I applied this function to extract symptom data for each year (2020, 2021, 2022, 2023) and non-domestic sources, saving the data in variables like vaersSymptomCovid2020, vaersSymptomCovid2021, and so on.

3. **vaersDataHashmapCovid(hasmap, filepath)**: This function read a CSV file containing demographic data (specified by filepath) and extracted demographic information for records within the provided hashmap. The output was a hashmap where each key was a VAERS_ID, and the value was a list of demographic attributes. I used this function to extract demographic data for each year and non-domestic sources, saving the data in variables like vaersDataCovid2020, vaersDataCovid2021, and more.

4. **singleYearCovidData(hasmapVaersData, hasmapVaersSymptoms, hasmapVaersCovid):** This function merged the data from the three hashmaps (vaccination, symptom, and demographic data) into a single hashmap. Each key was a VAERS_ID, and the value was a list that contained all three sets of attributes. Consequently, I combined data from different years (2020 to 2023) and non-domestic sources into a single hashmap, resulting in the **allFileCovid** variable that contains all the COVID-19 vaccination data.
5. Once the data is merged into a single hashmap (**allFileCovid**), it is used to create a new CSV file called "VAERS_COVID_DataAugust2023.csv."

   TOTAL NUMBER OF UNIQUE VAERS ID FOR TASK 1 = 1589965

CODE FOR TASK 1:

```
# TASK 1
from collections import defaultdict
import csv
#
==============================================================================
=======================================================

# READING DATA FROM VAERSVAX TO EXTRACT VAERSID THAT IS ONLY RELATED
WITH COVID
def vaersIDHashmapCovid(filepath):
    # CREATING A HASMAP
    vaersHashmap = defaultdict(list)
    # ENCODING ISO-8859-1
    r = open(filepath, "r", encoding="ISO-8859-1")
    reader = csv.reader(r)
    i=0
    # TO STORE THE HEADING
    headinglist = []
    for x in reader:
        # each row will be in the form of an array
        # for the first iteration i will get the heading row. Storing heading row in headling list array
        if i==0:
            # excluding the first element
            headinglist=x[1:]
            i+=1
        singlelinearray = x
        # checking if my singlelinearray has COVID19 in its second position or not
        if singlelinearray[1]=="COVID19" or singlelinearray[1]=="COVID19-2":
            # if True i will store the entire row from [1:], i am excluding vaers id
            vaersHashmap[int(singlelinearray[0])].append(singlelinearray[1:])
    # returnning [hasmap and heading list]
    return [vaersHashmap, headinglist]

#
vaersVaxCOVIDID2020Both
=vaersIDHashmapCovid("../dataset/2020Vaers/2020VAERSVAX.csv")
vaersVaxCOVIDID2020 = vaersVaxCOVIDID2020Both[0]
vaersVaxHeading = vaersVaxCOVIDID2020Both[1]
vaersVaxCOVIDID2021 =vaersIDHashmapCovid("../dataset/2021Vaers/2021VAERSVAX.csv")[0]
vaersVaxCOVIDID2022 =vaersIDHashmapCovid("../dataset/2022Vaers/2022VAERSVAX.csv")[0]
vaersVaxCOVIDID2023 =vaersIDHashmapCovid("../dataset/2023Vaers/2023VAERSVAX.csv")[0]
vaersVaxNonDomestics
=vaersIDHashmapCovid("../dataset/NonDomestics/NONDomesticVAERSVAX.csv")[0]
print(len(vaersVaxCOVIDID2020)+len(vaersVaxCOVIDID2021)+len(vaersVaxCOVIDID2022)+len
(vaersVaxCOVIDID2023)+len(vaersVaxNonDomestics))
```

```
#
=============================================================================
=========================================================
def vaersSymptomsHashmapCovid(hasmap, filepath):
    # creating a hasmap
    vaersSymptomsHasmap = defaultdict(list)
    r = open(filepath, "r", encoding="ISO-8859-1")
    reader = csv.reader(r)
    headinglist = []
    i=0
    for x in reader:
        if i==0:
            headinglist = x[1:]
            i+=1
            continue

        singlelinearray = x

        if int(singlelinearray[0]) in hasmap:

            vaersSymptomsHasmap[int(singlelinearray[0])].append(singlelinearray[1:])
    return [vaersSymptomsHasmap, headinglist]


vaersSymptomCovid2020Both
=vaersSymptomsHashmapCovid(vaersVaxCOVIDID2020,"../dataset/2020Vaers/2020VAERSSY
MPTOMS.csv")
vaersSymptomCovid2020 = vaersSymptomCovid2020Both[0]
vaersSymptomHeading = vaersSymptomCovid2020Both[1]
vaersSymptomCovid2021
=vaersSymptomsHashmapCovid(vaersVaxCOVIDID2021,"../dataset/2021Vaers/2021VAERSSY
MPTOMS.csv")[0]
vaersSymptomCovid2022
=vaersSymptomsHashmapCovid(vaersVaxCOVIDID2022,"../dataset/2022Vaers/2022VAERSSY
MPTOMS.csv")[0]
vaersSymptomCovid2023
=vaersSymptomsHashmapCovid(vaersVaxCOVIDID2023,"../dataset/2023Vaers/2023VAERSSY
MPTOMS.csv")[0]
vaersNonDomesticsSymptoms
=vaersSymptomsHashmapCovid(vaersVaxNonDomestics,"../dataset/NonDomestics/NonDomesti
cVAERSSYMPTOMS.csv")[0]


# # Assignment1/dataset/NonDomestics/NonDomesticVAERSSYMPTOMS.csv
#
print(len(vaersSymptomCovid2020)+len(vaersSymptomCovid2021)+len(vaersSymptomCovid202
2)+len(vaersSymptomCovid2023)+len(vaersNonDomesticsSymptoms))
```

```
# # # #
==============================================================================
======================================================
def vaersDataHashmapCovid(hasmap, filepath):
    i = 0
    vaersDataHasmap = {}
    r = open(filepath, "r", encoding="ISO-8859-1")
    reader = csv.reader(r)
    headinglist = []

    for x in reader:

        if i==0:
            headinglist =x
            i+=1
            continue
        singllinearray =x
        if int(singllinearray[0]) in hasmap:
            vaersDataHasmap[int(singllinearray[0])] = singllinearray
    return [vaersDataHasmap, headinglist]


vaersDataCovid2020Both
=vaersDataHashmapCovid(vaersVaxCOVIDID2020,"../dataset/2020Vaers/2020VAERSDATA.csv
")
vaersDataCovid2020 = vaersDataCovid2020Both[0]
vaersDataHeading = vaersDataCovid2020Both[1]
vaersDataCovid2021
=vaersDataHashmapCovid(vaersVaxCOVIDID2021,"../dataset/2021Vaers/2021VAERSDATA.csv
")[0]
vaersDataCovid2022
=vaersDataHashmapCovid(vaersVaxCOVIDID2022,"../dataset/2022Vaers/2022VAERSDATA.csv
")[0]
vaersDataCovid2023
=vaersDataHashmapCovid(vaersVaxCOVIDID2023,"../dataset/2023Vaers/2023VAERSDATA.csv
")[0]
vaersNonDomesticsDataCovid
=vaersDataHashmapCovid(vaersVaxNonDomestics,"../dataset/NonDomestics/NonDomesticVAE
RSDATA.csv")[0]

# # # #
==============================================================================
==============================================================================
==========================
# # # merging 3 different hasmap into single hasmap
def singleYearCovidData(hasmapVaersData, hasmapVaersSymptoms, hasmapVaersCovid):
    vaerSingleYearcovid = {}
    for x in hasmapVaersData:
```

```
vaerSingleYearcovid[x]=[hasmapVaersData[x],hasmapVaersSymptoms[x],hasmapVaersCovid[x]]
    return vaerSingleYearcovid


covidAllthree2020 =
singleYearCovidData(vaersDataCovid2020,vaersSymptomCovid2020,vaersVaxCOVIDID2020 )
covidAllthree2021 =
singleYearCovidData(vaersDataCovid2021,vaersSymptomCovid2021,vaersVaxCOVIDID2021 )
covidAllthree2022 =
singleYearCovidData(vaersDataCovid2022,vaersSymptomCovid2022,vaersVaxCOVIDID2022 )
covidAllthree2023 =
singleYearCovidData(vaersDataCovid2023,vaersSymptomCovid2023,vaersVaxCOVIDID2023 )
covidAllthreeNonDomestics =
singleYearCovidData(vaersNonDomesticsDataCovid,vaersNonDomesticsSymptoms,vaersVaxNo
nDomestics)


# # # merging 4 hasmap into single hasmap
covidAllthree2020.update(covidAllthree2021)
covidAllthree2020.update(covidAllthree2022)
covidAllthree2020.update(covidAllthree2023)
covidAllthree2020.update(covidAllthreeNonDomestics)
allFileCovid = covidAllthree2020


# # # #
===========================================================================
=====================================================
# # # # vaersID 1106890 = does not have any symptoms

# # # # using allthreeYearsCovid to create a new csv file

outputfilename = "VAERS_COVID_DataAugust2023.csv"
r = open(outputfilename,"w")
writer = csv.writer(r)
heading = vaersDataHeading+vaersVaxHeading+vaersSymptomHeading

writer.writerow(heading)
for x in allFileCovid:

    # mergingintosingleline
    currentVaersId = x
    currentVaersData = allFileCovid[currentVaersId]
    vaersData = currentVaersData[0]
    vaersSympt = currentVaersData[1]
    vaersType = currentVaersData[2]
    newvaersSympt=[]
    if len(vaersSympt)==0:
      #  vaersID 1106890 = does not have any symptoms
       newvaersSympt=["-","-","-","-","-","-","-","-","-","-","-","-","-"]
    else:

       for y in range(len(vaersSympt[0])):
          combinetxt=""
          for x in range(len(vaersSympt)):
             combinetxt += vaersSympt[x][y]+" "
          newvaersSympt.append(combinetxt)
    vaersSympt=newvaersSympt


    newvaersType=[]

    for c in range(len(vaersType[0])):
       combinetxt=""
       for d in range(len(vaersType)):
          combinetxt+=vaersType[d][c]+" "
       newvaersType.append(combinetxt)
    vaersType = newvaersType
    row = vaersData+vaersType+vaersSympt
    writer.writerow(row)
```

Task 2: Creating and Sorting COVID-19 Vaccination Data

In Task 2, the primary objective was to create a structured dataset for COVID-19 vaccinations and evaluate the efficiency of three different sorting algorithms: Quick Sort, Insertion Sort, and Bubble Sort. The dataset is constructed by extracting data from task1 dataset into Symptoms.csv file and is then sorted based on the VAERS_ID field using the mentioned sorting algorithms. Below is a breakdown of the code's structure and the steps involved:

The SYMPTOMDATA.csv. dataset used in this task is composed of the following attributes:

VAERS_ID: A unique identifier for each VAERS report.
AGE_YRS: The age of the patient.
SEX: The gender of the patient.
VAX_NAME: The specific COVID-19 vaccine administered.
RPT_Date: The date when the report was filed.
SYMPTOM: A specific symptom reported by the patient.
DIED: Indicates whether the patient passed away.
DATEDIED: The date of death (if applicable).
SYMPTOM_TEXT: A detailed description of the symptoms experienced by the patient.

Sorting and Analysis
The process involves the following steps:

1. Reading the COVID-19 vaccination dataset from the "VAERS_COVID_DataAugust2023.csv." file created in Task 1

2. Creating a new dataset of required columns and naming the file to “SYMPTOMDATA.csv”.

3. Applying each of the three sorting algorithms to sort the dataset based on the VAERS_ID field.

4. Measuring the runtime of each sorting algorithm to evaluate its time efficiency.

5. Analyzing how the sorting algorithms perform with different dataset sizes using subsets of the data.

Empirical Analysis

Quick Sort:

The Quick Sort algorithm is implemented with the random selection of a pivot element.
The runtime of Quick Sort is measured to assess its efficiency.
The algorithm's performance is evaluated with varying subset sizes of the dataset.

Insertion Sort:

The Insertion Sort algorithm is applied to the dataset.
The execution time of Insertion Sort is recorded to assess its time efficiency.
Different subsets of the dataset are used to analyze its performance.

Bubble Sort:

The Bubble Sort algorithm is applied to the dataset.
The runtime of Bubble Sort is measured to assess its efficiency.
The algorithm's performance is analyzed with varying dataset sizes.

Key Findings:

Quick Sort, with its randomly selected pivot, demonstrated superior performance and efficiency, particularly for large datasets.

Insertion Sort is efficient for small datasets but less practical for larger datasets due to its time complexity.

Bubble Sort showed limited efficiency and is not recommended for sorting large datasets.

These insights will be valuable in selecting the most appropriate sorting algorithm based on dataset size and performance requirements in further data analysis tasks. The code for dataset creation and sorting can be found in the code section of this report.

TIME COMPLEXITY:
Quick Sort: It takes less than five minutes for Quick Sort to run on the whole data dataset.

Insertion Sort: It takes five minutes for Insertion Sort to sort 3 million data and after for full data it takes around 7+ hours to complete.

Bubble Sort: It takes 10 minutes for Bubble Sort to sort 80 thousand data and for full data it takes around 9+ hours to complete.

TOTAL NUMBER OF VAERS ID FOR TASK 2 DATASET = 7276484

CODE FOR TASK 2

```
# # # # # TASK2
# # # # # VAERS_ID, AGE_YRS, SEX, VAX_NAME, RPT_Date, SYMPTOM, DIED, DATEDIED,
headinglist2  =
["VAERS_ID","AGE_YRS","SEX","VAX_NAME","RPT_Date","SYMPTOM","DIED","DATEDIED","
SYMPTOMS TEXT"]

# # # # vaersid = [0]
# # # # ageyears= [3]
# # # # sex     = [6]
# # # # vaxname = [6]
# # # # rptdate = [7]
# # # # can have upto 5 symptoms = [0,2,4,6,8]
# # # # died    = [9]
# # # # datedied= [10]
# # # # symptomtext = [8]

outputfilename2 = "SYMPTOMDATA.csv"
r2 = open(outputfilename2,"w")
writer = csv.writer(r2)
writer.writerow(headinglist2)

def task2datasetYearly(vaersData, vaersSymptoms, vaersVax):
    for x in vaersSymptoms:
        for y in vaersSymptoms[x]:
            i=0
            for z in y:
                if i>len(y)-1:
                    break
                if len(y[i])==0:
                    break
                if len(y[i])>1:
                    vaersId = vaersData[x][0]
                    ageyrs = vaersData[x][3]
                    sex = vaersData[x][6]
                    vaxname = vaersVax[x][0][6]
                    rptdate = vaersData[x][7]
                    symptom = y[i]
                    died = vaersData[x][9]
                    datedied = vaersData[x][10]
                    symptomstext= vaersData[x][8]
                    row=[vaersId, ageyrs, sex, vaxname, rptdate, symptom, died,
datedied,symptomstext]
                    writer.writerow(row)

                i+=2


a = task2datasetYearly(vaersDataCovid2020, vaersSymptomCovid2020,
vaersVaxCOVIDID2020)
b = task2datasetYearly(vaersDataCovid2021, vaersSymptomCovid2021,
vaersVaxCOVIDID2021)
c = task2datasetYearly(vaersDataCovid2022, vaersSymptomCovid2022,
vaersVaxCOVIDID2022)
d = task2datasetYearly(vaersDataCovid2023, vaersSymptomCovid2023,
vaersVaxCOVIDID2023)
e =
task2datasetYearly(vaersNonDomesticsDataCovid,vaersNonDomesticsSymptoms,vaersVaxNon
Domestics)


import csv
import random
import time


def read_csv(filename):
    data = []
    r= open("./SYMPTOMDATA.csv", "r", encoding="ISO-8859-1")
    reader = csv.DictReader(r)

    for x in reader:
        data.append(x)
    print(len(data))
    return data
a = read_csv("./SYMPTOMDATA.csv")
#
=============================================================================
=================================================
def insertion_sort(data, field):
    for i in range(1, len(data)):
        print(i)
        pivot = int(data[i][field])
        j = i - 1
        while j >= 0 and pivot < int(data[j][field]):
            data[j + 1] = data[j]
            j -= 1
        data[j + 1][field] = pivot

#
=============================================================================
==================================================

def bubble_sort(data, field):
    n = len(data)
    for i in range(n - 1):
        swapped=False
        print(i)
        for j in range(n - i - 1):
            if int(data[j][field]) > int(data[j + 1][field]):
                data[j], data[j + 1] = data[j + 1], data[j]
                swapped=True
        if not swapped:
            print("DONE WITH BUBBLE SORT")
            return

# #
=============================================================================
============
def partition(data, field, low, high):
    if low < high:
        pivot_index = quick_sort(data, field, low, high)
        partition(data, field, low, pivot_index-1)
        partition(data, field, pivot_index + 1, high)


# # Function to partition the data for quick sort
def quick_sort(data, field, l, r):
        pivotindex = random.randint(l,r)
        data[l], data[pivotindex] = data[pivotindex], data[l]
        pivotelement = int(data[l][field])
        l=l+1
        while l<=r:

            while l<=r and int(data[l][field])<=pivotelement:
                l+=1
            while l<=r and int(data[r][field])>=pivotelement:
                r-=1
            if l<=r:
```

```
            temp = data[l]
            data[l] =data[r]
            data[r] = temp
        temp = data[r]
        data[r] = pivotelement
        data[pivotindex] = temp
        return r

#
==============================================================================
===========================================
def write_sorted_csv(data, filename):
    r= open(filename, 'w', newline='')
    fieldnames = data[0].keys()
    writer = csv.DictWriter(r, fieldnames=fieldnames)
    writer.writeheader()
    for row in data:
        writer.writerow(row)
#
==============================================================================
==================================================

a = read_csv("./SYMPTOMDATA.csv")
partition(a, "VAERS_ID", 0, len(a)-1)
insertion_sort(a, "VAERS_ID")
bubble_sort(a, "VAERS_ID")
```

Task 3 Report

Implementation Summary:

In Task 3, I was tasked with sorting the dataset I created in Task 2. The sorting criteria were to group the data by age group, then by gender, then by Vaccine Name. Additionally, I was required to compute the number of cases that resulted in death for each age group.

I organized my code into several key sections, as explained below:

1. Data Reading: I implemented a function **read_csv** that reads the dataset from a CSV file and stores it in a list.

2. Data Grouping: I defined age groups and initialized dictionaries (ageRset and ageGroups) to categorize the data based on the age of patients. Data points with missing age information were categorized as "NO AGE."

3. Sorting Functions: I implemented insertion sort functions (insertion_sort and insertion_sortWords) for potential sorting of data.

4. Death Counts: I iterated through the grouped data to count the number of cases that resulted in death for each age group. I used a unique set to ensure that each VAERSID was counted only once.

Results:

Here are the results of counting cases resulting in death for each age group:

NO AGE: 12,267 cases
<1 year: 7 cases
1-3 years: 7 cases
4-11: 46 cases
12-18: 188 cases
19-30: 480 cases
31-40: 660 cases
41-50: 1,157 cases
51-60: 2,358 cases
61-70: 4,734 cases
71-80: 6,437 cases
80: 7,739 cases

Total cases resulting in death across all age groups: 36,080.

CODE FOR TASK 3

```
import csv
def read_csv(filename):
    data = []
    r = open(filename, "r", encoding="ISO-8859-1")
    reader = csv.DictReader(r)

    for x in reader:

        data.append(x)


    return data

a = read_csv("./SYMPTOMDATA.csv")
# a = read_csv("./VAERS_COVID_DataAugust2023.csv")

ageRset = {
    "NO AGE":[-1,-1],
    "<1 year": [0, 1],
    "1-3 years": [1, 4],
    "4-11": [4, 12],
    "12-18": [12, 19],
    "19-30": [19, 31],
    "31-40": [31, 41],
    "41-50": [41, 51],
    "51-60": [51, 61],
    "61-70": [61, 71],
    "71-80": [71, 81],
    "> 80": [81, float("inf")],
}

ageGroups = {
    "NO AGE":[],
    "<1 year": [],
    "1-3 years": [],
    "4-11": [],
    "12-18": [],
    "19-30": [],
    "31-40": [],
    "41-50": [],
    "51-60": [],
    "61-70": [],
    "71-80": [],
    "> 80": []
}
blank=0

for x in a:

    if x["AGE_YRS"]=="":
        ageGroups["NO AGE"].append(x)

    else:
        age = float(x["AGE_YRS"])
        for c,d in ageRset.items():
            if d[0]<=age and d[1]>=age:
                ageGroups[c].append(x)

# ============================================================================
def insertion_sort(data, field):
    a=0
    for i in range(1, len(data)):
        pivot = float(data[i][field])
        j = i - 1

        while j >= 0 and pivot < float(data[j][field]):
            data[j + 1] = data[j]
            j -= 1
        data[j + 1][field] = pivot
        a+=1

def insertion_sortWords(data, field):
    a=0
    for i in range(1, len(data)):


        pivot = data[i][field]
        j = i - 1

        while j >= 0 and pivot < data[j][field]:
            data[j + 1] = data[j]
            j -= 1
        data[j + 1][field] = pivot
        a+=1

 for z in ageGroups:
     if z=="NO AGE":
         continue

     insertion_sort(ageGroups[z], "AGE_YRS")
for i in ageGroups:
     insertion_sortWords(ageGroups[i], "SEX")
for k in ageGroups:
    insertion_sortWords(ageGroups[k], "VAX_NAME")

# ==========================================DEATHS
COUNTS========================================================================
==============
unique = set()
a=0
b=0
count=0
res = []
for j in ageGroups:
    d = 0
    for l in ageGroups[j]:
        count+=1
        #
print(str(a)+"============================================================"+str(b))
        if l["DIED"]=="Y" and l["VAERS_ID"] not in unique:
            d+=1
            unique.add(l["VAERS_ID"])

    res.append(d)
print(res)
res1=0
for sh in res:
    res1+=sh
print(res1)

# MANY DATA WHERE THE THERE IS NO DATA IN THE AGE FIELD
```

DEATHS ACCORDING TO AGE GROUP

AGE GROUP	NUM OF DEATHS
# NO AGE GROUP	12267
# <1 year	7
# 1-3 years,	7
# 4-11,	46
# 12-18,	188
# 19-30,	480
# 31-40,	660
# 41-50,	1157
# 51-60,	2358
# 61-70,	4734
# 71-80,	6437
# > 80),	7739

# [12267, 7, 7, 46, 188, 480, 660, 1157, 2358, 4734, 6437, 7739]
# TOTAL NUMBER OF DEATHS = 36080