CoViD-19 Data Analysis

Dictionary data structure, TKinter data visualization, ethics and social consequences of medical data and data ethics, basic statistical analysis

Due: due date here

Part I: Prepping the data

Pull the data from the CSV file and prep it for analysis

The data provided in the file **COVID19.csv** contains data in a time-frame from January 22, 2020 to February 8, 2020 that contains confirmed cases, deaths, and recoveries of CoViD-19. Taken from the README of this dataset: "There are a total of 60,371 confirmed cases worldwide, with 59,804 in Mainland China. Sadly at least 1367 people have died from this virus."

This dataset captures the most recent data (as of this writing) from global outbreak information. As you can probably guess, the data is now outdated. The goal of this homework is not to do a real-time analysis of the current situation, but rather to use a relevant and grave situation to examine how data and numbers can be manipulated to portray different things.

For this part of the homework, you will be expected to do the following:

- Read the data in from the CSV file
- Create a nested dictionary and store the data using the following categories:
  - Day
  - Date
  - Confirmed
  - Deaths
  - Recoveries

Example:

```
dataPointID = 0
dataPoints_byDay = dict()
dataPoints_byDay[dataPointID]= { "Date": "01/22", "Confirmed": 555, "Deaths": 0,
    "Recoveries": 0}
```

Remember that to read data from a CSV, you can use code similar to this:

```
data_file = open("filename.csv",'r')
for line in data_file:
```

```
    data = line.strip().split(",")
    #Do the things with the data
data_file.close()
```

Part II: Analysis Functions

Design and implement functions to analyze the data

In this part, you will be designing, implementing, and testing algorithms to perform various statistical analyses of the data. The goal of this exercise is to get you used to manipulated data and finding specific attributes in datasets.

You won't need to be an expert in statistical analysis for this one. We'll provide you with the operations to implement and pseudo-code to provide a framework for your algorithms. However, it will ultimately be up to you to design and implement them.

Your functions will need to take 3 parameters: the first parameter will be the X-axis values in a graph; the second parameter will be the Y-axis values in a graph; the third parameter will be the constraint on the data in the form of a tuple: `(category, value)` where the latter is the maximum value that will be considered (this parameter will have a default value of `None` which signifies a general comparison of x and y parameters). Your functions will return a dictionary that will be passed into a TKinter interface in Part 3. This returned dictionary will have the following categories:

- `"X values"`
- `"Y values"`
- `"Constraint"`

Please make sure your keys match these exactly, as they will be used in the provided code. As an example:

```
def findAll(x, y, constraint=None):
    returnData = {"X values": [], "Y values": [], "Constraint": constraint}
    if constraint is None:
        for key in x.keys():
            returnData["X values"].append(key)
            returnData["Y values"].append(x[key][y])
    else:
        for key in x.keys():
            if x[key][contraint[0]] <= constraint[1]:
                returnData["X values"].append(key)
                returnData["Y values"].append(x[key][y])
    return returnData

byDay = {0: {"Date": "01/22", "Confirmed": 555, "Deaths": 0, "Recoveries": 0},
         1: {"Date": "01/23", "Confirmed": 653, "Deaths": 0, "Recoveries": 30},
         2: {"Date": "01/24", "Confirmed": 941, "Deaths": 26, "Recoveries": 36}}
data = sum(byDay, "Deaths", ("Confirmed", 2000))

#data = {"X values": [0,1], "Y values": [0,0], "Constraint": ("Confirmed", 2000)}
```

Naturally, your data will be much larger than this and might not be so clearly defined. You'll need to be careful with how you define your algorithms and how you choose to track data points.

Required Operations

- Average: This function will return the average value that appears across all data points that match the given parameters. For example:

```
average[dataID] = avg(data if data within constraints for data in x]
```

- Minimum: This function will return the minimum value that appears across all data points that match the given parameters. For example:

```
minimum[dataID] = min(data if data within constraints for data in x)
```

- Maximum: This function will return the maximum value that appears across all data points that match the given parameters. For examples:

```
maximum[dataID] = max(data if data within constraints for data in x)
```

- Mode: This function will return the most common value that appears across all data points that match teh given parameter. For example:

```
mode[dataID] = mostCommon(data if data within constraints for data in x)
```

- findAll: This function will return all values that appear across all data points that match the given parameter. For example:

```
findAll[dataID] = [data if data within constraints for data in x]
```

It is **VERY IMPORTANT** that your functions have these exact names. If they don't, the backend we're using to convert the dictionaries to graphs won't work.

Part III: TKinterface

Design and implement a UI with TKinter to display your analysis

For this final part of the homework, you will be implementing a TKinter interface that will display your graphs. In the provided files, you will see two files; one of these files contains a MatPlotLib backend that will be used to call your functions and provide the graphs to the TKinter interface; the other is the TKinter interface skeleton that you are to implement.

Make sure to follow the instructions as described in the comments of the provided code. Please make sure you meet these specifications exactly. Any inconsistencies can cause bugs in your code that can be hard to debug.

Part IV: README

Final Thoughts

Please respond to the questions in the provided README.txt file and submit it along with your code.