MAT5983 Data Analytics: Homework 1

Carl Jones

9 July 2023

Contents

1 Assignment Introduction

A relatively small data repository was provided of approximately 380MB, containing some portion of the 1969 National Vital Statistics data of the United States of America, was provided along with the data dictionary for the file. This data was stored in a flat-file format. The data was to be searched for specific values, using a programming language of the students' choice. Afterward, questions regarding the selected data were answered.

2 Assignment Procedure and Methods

2.1 Procedure

2.1.1 Hardware and Software Used

: Hardware was a personal desktop computer, with the following components:

- Processor: AMD Ryzen 7 3700X 8-core Processor 3.60 GHz
- Installed RAM: 16.0 GB Dual-channel DDR4 Memory 3600MHz
- GPU: Nvidia GTX 1080 Ti 11GB VRAM (not used for this procedure)
- System type: 64-bit, x64-based processor

The following software were used:

- Operating System: Windows 10 Pro, version 21H2

- Programming language and version: Python 3.9
- IDE and version: Visual Studio Code, version 1.80.1
- Code Libaries: The *matplotlib* and *numpy* libraries were used. NOTE: Although *numpy* was not allowed for the purpose of data collection from the file, it was used in a single line of code for displaying the data. This was done as even *pyplot* official documentation used *numpy* for this purpose.
- MiKTeX version 23.1 and TeXworks version 0.6.7 were used to prepare the report document.

2.2 Methods

To answer the questions, scripts were written as functions called from a driver *main()* function. The flat-filed is loaded into memory using the *open()* command, and then scanned line-by-line. The data dictionary was used to identify the sought data entries, which were tabulated per question basis. A description of the functions written follow.

Counting the live-births to Texas mothers in Texas

The function *isTexasMother()* was used to answer the first programming question, which asked for the total number of births occuring in Texas to Texas mothers be found. The function takes two inputs: (1) *file*, a file-type data object and (2) *posVal*, the integer used to access the position of a record in the flat-file. Using the data dictionary, the function checks if the related positions in a record contain the values associated with a Texas birth and a Texas-residing mother. The function returns a single Boolean TRUE value if a record matching the requirements is found at the record located in the posVal position of the file, and nothing otherwise.

The function *bornAlive()* was used to determine if the birth that occurred was a live birth. It takes two inputs, the same as *isTexasMother()* above. The function scans the flat-file line-by-line and checks the values in related positions against the values from the data dictionary to determine if a birth was live or not. This was done by checking the record positions associated with “Live Birth Order”, of which there were several recodes. The function returns a single Boolean TRUE value if a record matching the requirements is found at the record located in the posVal position of the file, and nothing otherwise.

Graphical display of mother’s level of education vs birth order of child

The function *educationVsBirthOrder()* was used to tabulate total values of birth order versus mother’s education level. The function take as input a file-type data object. The function assigns an emptry Python list data type first, and then scans through the file line-by-line (record-by-record, in this case). The list is a single-dimension with multi-dimension values stored in each index - it has 9 total indices, each of which contains a triplet of data. Since the data dictionary has contradicting values for mother’s education and birth order, the data is simplified in the following way:

- Mother’s education is simplified to 3 areas - “At Most K-8th”, “At Most 9th-12th”, and “At least Some College”. These sets represent that a mother has received some number of years between 0 and 8, 9 through 12, and 13 or more of education, respectively.
- After several checks were performed, the author noticed that only the data in record positions 60 and 63 changed the total values. By printing several samplings of record statements and reading the positions associated with live birth order (positions 60 through 67), author noticed that positions 60 through 67 always have numerical values, making positions 64 through 67 superfluous. This was double-checked in code by including different combinations and logical conditions on positions 60 and 63 through 67. Thus, only record positions 60 and 63 were checked in the logic of the function.

The totals for each birth order by mother’s education were totaled and stored the list. Indices of the list correspond to the birth order, with the triplets corresponding to total births for mother’s of a certain education level. The function returns the list object.

The *main()* function loads the file using the standard Python *open()* function, calls the other defined functions, and then plots the data in a grouped bar chart.

Please see the Appendix for all code used to complete this assignment.

3 Questions and Answer

3.1 Problem #1: Why is date and time of birth no longer recorded?

1. ***Initial Answer***

There are several reasons you might wish to not record date and time of birth instead of simply week of birth. For instance, from the security perspective, data and time of birth might be too identifiable of information. There could also be concerns about the intended use of the data. That level of granularity may not be valuable for the intended purpose, or may make data overly difficult to process and analyze. It may also have been determined that date and time added nothing of value more than simply week of birth did. It may also have been determined that this data made records more difficult to originate and maintain than the value-added from their recording.

2. ***Researched Answer***

I couldn't find anything online about the change in 1988, but I did find an explanation about other changes made to the 2020 dataset here (https://www.cdc.gov/nchs/nvss/vsrr/natality-technical-notes.htm, retrieved 9 July 2023), which changed the way that record weights were defined and used. Previously, the record weights were used to generate provisional estimates, but from 2020 on weights were not generated because of completeness in data collection. Perhaps it was a similar reason to drop the date and time of births beginning in 1988 - that is, because completeness of records made the extra granularity in the data superfluous.

3.2 Problem #2: Why was a flat-file format used?

1. ***Initial Answer***

A flat file might make it easy for a less-complex computer to read in the data without the use of more complex data structures, saving memory and improving computational performance (string operations are pretty efficient, in general). It also ensures that the structure generalizes well across environments, since strings are ubiquitous data types and encodings are generally widely shared or easily translated.

2. ***Researched Answer***

Along with what I've already mentioned, there's also stability and security provided by making the file read-only.

3.3 Problem # 3: What problem is encountered when loading the data into Microsoft Excel?

The problem that occurs when loading the file into Microsoft Excel (Excel) using the *Text Import Wizard* is that Excel only displays 1,048,576 lines from the file out of the total 1,800,103 lines in the file. This is a limitation of Excel itself. sub

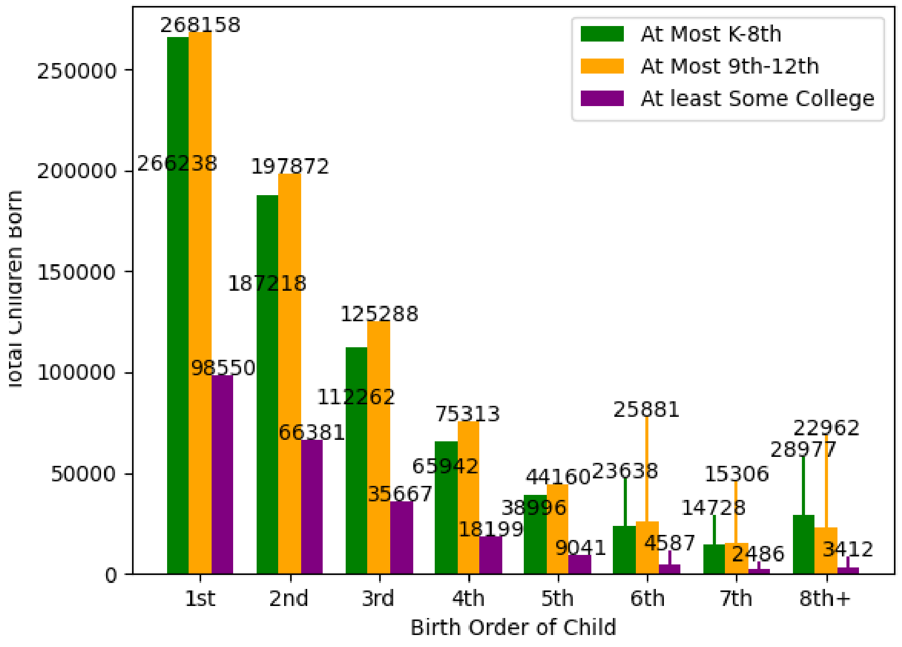
4 Programming Problems

1: How many live births occurred in Texas in 1969 from mtohers residing in Texas?

Answer: 37,697

Bonus: Since the data is highly multivariate, a visualization method such as circle segment pixel plot would probably be necessary for the quantative values. For the qualatative data, a mosaic plot might be the best method of displaying the data.

2: Show graphically how the level of education of the mother is related to the birth order (1st born, second child, third, etc.)



Bonus: I think the best way to visualize this data would be with a scatter plot matrix or circle segment plot.

5 Appendix

Python Code

Note: some lines in the following code have been formated to fit the .pdf margins.

```
import matplotlib.pyplot as plt
import numpy as np

#the following block of code in the driver function, main()
def main():

    data1969 = open("REDACTED\\US1969.dat", "r", encoding = "cp1252")
    data1969.seek(0)

    #counts total number of TX-mothers giving birth in TX
    count = 0
    data1969.seek(0)
    for x in data1969:
        if isTxMother(data1969, x) and bornAlive(data1969, x):
            count = count + 1
    print("The total number of live-births in Texas to mothers residing in Texas is: " +
str(count) + ".")

    #counts the total number of births by order vs mother’s education level
    data1969.seek(0)
    educVsNumChildren = educationVsBirthOrder(data1969)
    #breaks the data apart into easily-plottable formats
    #this hack-job is the best I could do sans libraries
    educVsNumChildren.pop(0)
    elemSchoolTotals = []
    highSchoolTotals = []
    collegeTotals = []
    for i in range(8):
        elemSchoolTotals.append(educVsNumChildren[i][0])
    for i in range(8):
        highSchoolTotals.append(educVsNumChildren[i][1])
    for i in range(8):
        collegeTotals.append(educVsNumChildren[i][2])
    birthOrder = ['1st', '2nd', '3rd', '4th', '5th', '6th', '7th', '8th+']
    educationLevel = ["At Most K-8th", "At Most 9th-12th", "At least Some College"]

    #graphs the data
    width = 0.25
    xAxis = np.arange(8) #please forgive this single use of numpy, I could not figure out
    #how to get it to work without it, even pyplots official documentation examples uses
    #numpy for this
    plt.figure()
    plt.bar(xAxis - width, elemSchoolTotals, width, color="green")
    plt.bar(xAxis, highSchoolTotals, width, color="orange")
    plt.bar(xAxis + width, collegeTotals, width, color="purple")
    plt.xticks(xAxis, birthOrder)
    plt.xlabel("Birth Order of Child")
    plt.ylabel("Total Children Born")
    plt.legend(educationLevel)
    #plots the values over the bars for elementary-educated mothers
    for i in range(len(educVsNumChildren)-3):
        plt.text(i-width, elemSchoolTotals[i]-(elemSchoolTotals[i]*.25),
elemSchoolTotals[i], ha = 'center')
    for i in range(5,8):
        plt.text(i-width, elemSchoolTotals[i]+2*(elemSchoolTotals[i]*.5),
elemSchoolTotals[i], ha = 'center')
    for i in range(5,8):
        plt.vlines(i-width, elemSchoolTotals[i], elemSchoolTotals[i]+
2*(elemSchoolTotals[i]*.5), color='green')
    #plots the values over the bars for high school-educated mothers
    for i in range(len(educVsNumChildren)-3):
        plt.text(i, highSchoolTotals[i], highSchoolTotals[i], ha = 'center')
    for i in range(5,8):
        plt.text(i, highSchoolTotals[i]+2*highSchoolTotals[i],
highSchoolTotals[i], ha = 'center')
    for i in range(5,8):
        plt.vlines(i, highSchoolTotals[i], highSchoolTotals[i]+
2*highSchoolTotals[i], color='orange')
    #plots the values over the bars for college-educated mothers
    for i in range(len(educVsNumChildren)-3):
        plt.text(i+width, collegeTotals[i], collegeTotals[i], ha = 'center')
    for i in range(5,8):
        plt.text(i+width, collegeTotals[i]+2*(collegeTotals[i]*.75), collegeTotals[i],
ha = 'center')
    for i in range(5,8):
        plt.vlines(i+width, collegeTotals[i], collegeTotals[i]+2*(collegeTotals[i]*.75),
color='purple')
    plt.show()

#the following blocks of code are the various functions
def isTxMother(file, posVal):
    for posVal in file:
        if posVal[25] == "7" and posVal[26] == "4" and
(posVal[27] == "4" and posVal[27] == "4"):
            return True

def bornAlive(file, posVal):
    for posVal in file:
        if (int(posVal[59]) or int(posVal[62]) or int(posVal[63])
or int(posVal[64]) or int(posVal[65]) or int(posVal[66])) in range(1, 10):
            return True

def educationVsBirthOrder(file):
    arr = [[0 for x in range(3)] for y in range(9)]
    for x in file:
        if int(x[59]) in range(1,9):
            #elementary school educated birth totals
            if int(x[97] + x[98]) in range(9) or int(x[99] +
 x[100]) in range(4) or int(x[101]) == 1:
                arr[int(x[59])][0] += 1
            #high school educated birth totals
            if int(x[97] + x[98]) in range(9,13) or int(x[99] +
x[100]) in range(5,9) or int(x[101]) in range(2,4):
                arr[int(x[59])][1] += 1
            #college educated birth totals
            if int(x[97] + x[98]) in range(13,18) or int(x[99] +
x[100]) in range(9,14) or int(x[101]) in range(4,6):
                arr[int(x[59])][2] += 1
        elif int(x[62]) in range(1,9):
            #elementary school educated birth totals
            if int(x[97] + x[98]) in range(9) or int(x[99] +
x[100]) in range(4) or int(x[101]) == 1:
                arr[int(x[62])][0] += 1
            #high school educated birth totals
            if int(x[97] + x[98]) in range(9,13) or int(x[99] +
x[100]) in range(5,9) or int(x[101]) in range(2,4):
                arr[int(x[62])][1] += 1
            #college educated birth totals
            if int(x[97] + x[98]) in range(13,18) or int(x[99] +
x[100]) in range(9,14) or int(x[101]) in range(4,6):
                arr[int(x[62])][2] += 1
    return arr

if __name__ == "__main__":
    main()
```