CS251 Proj2

CS251 Project 2 - Data Management

The purpose of this project was to start building a Data class with capability of reading .csv files. To start, we had to create a Data class that can read and access data and store it in two forms, Raw form and Matrix form. The we had design a read() method that takes in a file and separates the data into Raw Data form in a string format as a list of lists, with one sublist for each data point. The overall goal of this lab was the create a Data class that can read and organize data from a file and to create an analysis class that smoothly performs data analysis functions on the data.

Task 1. In this task, we are ask to add new code to our read method. The main goal of this task was to update our Data class to support the numeric form of the data and produce a numpy matrix.* *First, I added code that looped through each column of *numeric data only* and converts the string data to floated numeric form. Then I stored this information in a single numpy matrix. Then, because every type in my original data is not numeric, I had to create a second dictionary, *header2matrix*, to map the headers to their corresponding column indexes in the numeric data. Here is my code for task 1:

```
#Project2 Task1
# worked with Julia Saul, help from Bruce & CP
convert_data = []
num_col = 0
self.header2matrix = {}

for i in range(len( self.raw_data)):
    rowList=[]
    for j in range(len(self.raw_headers)):
        if self.raw_types[j] == "numeric":
            rowList.append(float(self.raw_data[i][j]))
        #second dictionary that maps headers to corresponding columns
        #advice from Bruce to reduce amount of code
        #assign this within the for loop to create new numeric indices
    convert_data.append(rowList)
num_col = 0
i = 0
for types in self.raw_types:
    if types == "numeric":
        self.header2matrix[self.raw_headers[i]] = num_col
        num_col += 1
    i +=1
#the matrix of the numeric data
self.matrix_data = np.matrix(convert_data)
```

Task 2. In this task, we were ask to create an analysis.py class. The goal of this task was to design 5 methods that computed different simple data analysis function. All of the analysis functions take a list of column header strings to specify what numeric data to analyze. We were asked to code a method to compute the data range, the mean, the standard deviation, and then two functions to normalize the data columns(one separately, one together). Here is an example of one of the functions, data_range():

```
#Help from Bruce
# help from CP majgaard
def data_range(column_headers, data):
    colNum = []
    mx = data.matrix_data

    for col in column_headers:
        colNum.append(data.header2matrix[col])

    mins =  mx.min(axis = 0)
    maxs = mx.max(axis = 0)

    data_range = []
    for col in colNum:
        pair = []
        pair.append(mins.item(col))
        pair.append(maxs.item(col))
        data_range.append(pair)
    return data_range
```

==========

Data API and Functions

Like I described in Task 1 and 2, this project involved two main classes: ***Data and Analysis:***

Data: is a class that reads a .csv file using cread in Python. First, the class will place all the raw data in string format as a list of lists. Then, another methods reads through the raw data and locates only the numerical data. The data class takes this numerical data and stores it in a numpy matrix.

Here are my Data class methods:

read: this method reads a .csv file using .cread. After reading in the .csv file, the read method separates headers, types and data as a list of strings using cread.next(). Using a for loop, the read method stores a copy of raw data as lists. Finally, using the code I wrote in task 1 of the

project, the read method creates a numpy matrix of only the numerical data.

get_raw_headers: returns a list of all headers

get_raw_types: returns a list of all data types

get_raw_num_columns: returns the number of columns in the raw data set (the length of the header list)

get_raw_num_rows: returns the number of rows in the raw data set, not including headers or types

get_raw_row(row): returns a row of data (type is list) given a row index (int).

get_raw_value(row, column): takes a row index(int) and column header(str) and returns a string type of the raw data at that location

Functions associated with matrix:

get_headers: returns a list of headers of columns with numeric data

get_row(row): *takes a row index and returns a row of numeric data

get_value(row, column): takes a row index (int) and column header (string) and returns the data in the numeric matrix

get_data(column_headers): takes a list of columns headers and return a matrix with all numeric data for all rows but just the specified column data segment.

printAll: prints all the contents of the raw_data field

testingMethods: test function for some my data class methods

testingAnalysis: test function for my analysis class methods

Analysis is the python class I created in task 2 that contains 5(+2 extensions) functions that compute different simple data analyses on the data read from the .csv file in the Data class.

All of the analysis method take a list of column header strings to specify what numeric data to analyze and a data object.

Here are my Analysis class methods:

data_range: returns a list of two element list for all columns specified. Each two-element list contains a min and a max for each specified column.

mean: returns a list of the mean values for each column specified.

stdev: returns a list of the standard deviation values for each column specified.

normalizeColumnsSeparately: computes the min and max of *each* column and normalizes data for all columns within their column. Returns a matrix with each column separately normalized so its minimum value is mapped to zero and it maximum value is mapped to 1.

normalizeColumnsSeparately: the same as normalize_columns_separately but computes the min and max of *the entire matrix* and normalizes all the data together so the matrix's minimum value is mapped to zero and the matrix's maximum value is mapped to 1.

sum:(ext1) returns a list of the sum of each column

round:(ext2) returns the same matrix with all the data rounded to a specified decimal

==========

Task 3. For this task, we were asked to find our own data-set and put it into a .csv and prove that our Data class can properly read the file. Steven Parrott and I created an Excel.csv file titled "MLBPitching.csv" with professional baseball pitching statistics. In a separate excel file, we calculated the mean, standard deviation, and data range of our data. The answers compare perfectly with running the file through my data class. Here is the evidence:

Values of Mean, StDev, and Range calculate for my Data set in Excel:

Mean		12.95	9.85	3.73	0.78	0.11	0.45
STD		4.43	2.74	0.66	0.18	0.03	0.08
Range		(8, 24)	(5, 17)	(2.4, 4.75)	(0.46, 1.01)	(0.06, 0.17)	(0.29, 0.57)

Values of Mean, StDev, and Range calculate for my Data set in my Analysis class(terminal screenshot):

```
Data range:
[[8.0, 24.0], [5.0, 17.0], [2.4, 4.75], [0.46, 1.01], [0.06, 0.17], [0.29, 0.57]]
Mean:
[12.95, 9.85, 3.7305, 0.7830000000000001, 0.10800000000000001, 0.4535]
Standard Deviation:
[4.0324342915650195, 2.739093203771527, 0.6600037878679182, 0.1757120850769954, 0.02894641146743591, 0.07700136704186356]
```

Extension 1. For this extension, I added another type of simply data analysis. I gave my data class the capability of summing/totaling each column in my matrix. Here is a snippet of my sum() function:

```
#EXTENSION 1
#sums the data in each column
def sum(column_headers, data):
    colNum = []
    mx = data.matrix_data

    for col in column_headers:
        colNum.append(data.header2matrix[col])

    sum = mx.sum(axis=0)
    sums = []
    for col in colNum:
        sums.append(sum.item(col))

    return sums
```

Extension 2. For this extension, I added another type of simply data analysis. I gave my data class the capability of rounding all the values in my matrix. Here is a snippet of my round() function:

```
#EXTENSION 2
#round to nearest
def round(column_headers, data):
    a = data.get_data(column_headers)
    round = a.round(1)
    return round
```

Finally, look how beautifully my terminal looks when I run my analysis.py file with all of the print statements testing each method:

```
tonys-new-mbp:Project2 Tony$ python2.7 analysis.py MLBPitching.csv
Here is our Data Matrix:
[[ 24.     5.     2.4    1.     0.1    0.29]
 [ 13.     7.     2.89   0.91   0.11   0.34]
 [ 16.     7.     2.94   0.92   0.07   0.4 ]
 [ 19.     8.     3.     0.97   0.07   0.37]
 [ 16.    10.     3.17   0.81   0.08   0.38]
 [  9.     9.     3.32   0.72   0.06   0.43]
 [ 11.    12.     3.38   0.78   0.11   0.42]
 [ 15.     9.     3.47   0.95   0.1    0.4 ]
 [ 14.    14.     3.47   0.95   0.08   0.42]
 [ 13.     9.     3.59   0.53   0.1    0.45]
 [  9.    10.     3.74   1.01   0.11   0.44]
 [  8.    10.     4.     0.82   0.13   0.52]
 [ 12.     7.     4.25   0.54   0.15   0.48]
 [  9.    13.     4.3    0.46   0.1    0.55]
 [  8.    12.     4.33   0.79   0.11   0.52]
 [  9.    17.     4.33   0.63   0.13   0.54]
 [ 14.    10.     4.4    0.84   0.17   0.51]
 [ 15.     9.     4.43   0.89   0.15   0.52]
 [ 11.    10.     4.45   0.57   0.13   0.52]
 [ 14.     9.     4.75   0.57   0.1    0.57]]
Data range:
[[8.0, 24.0], [5.0, 17.0], [2.4, 4.75]]
Mean:
[12.95, 9.85, 3.7305]
Standard Deviation:
[4.0324342915650195, 2.739093203771527, 0.6600037878679182]
```

```
Normalize Column Separatley:
[[ 1.         0.         0.        ]
 [ 0.3125     0.16666667 0.20851064]
 [ 0.5        0.16666667 0.22978723]
 [ 0.6875     0.25       0.25531915]
 [ 0.5        0.41666667 0.32765957]
 [ 0.0625     0.33333333 0.39148936]
 [ 0.1875     0.58333333 0.41702128]
 [ 0.4375     0.33333333 0.45531915]
 [ 0.375      0.75       0.45531915]
 [ 0.3125     0.33333333 0.50638298]
 [ 0.0625     0.41666667 0.57021277]
 [ 0.         0.41666667 0.68085106]
 [ 0.25       0.16666667 0.78723404]
 [ 0.0625     0.66666667 0.80851064]
 [ 0.         0.58333333 0.8212766 ]
 [ 0.0625     1.         0.8212766 ]
 [ 0.375      0.41666667 0.85106383]
 [ 0.4375     0.33333333 0.86382979]
 [ 0.1875     0.41666667 0.87234043]
 [ 0.375      0.33333333 1.        ]]
Normalize Column Together:
[[ 1.         0.12037037 0.        ]
 [ 0.49074074 0.21296296 0.02268519]
 [ 0.62962963 0.21296296 0.025     ]
 [ 0.76851852 0.25925926 0.02777778]
 [ 0.62962963 0.35185185 0.03564815]
 [ 0.30555556 0.30555556 0.04259259]
 [ 0.39814815 0.44444444 0.04537037]
 [ 0.58333333 0.30555556 0.04953704]
 [ 0.53703704 0.53703704 0.04953704]
 [ 0.49074074 0.30555556 0.05509259]
 [ 0.30555556 0.35185185 0.06203704]
 [ 0.25925926 0.35185185 0.07407407]
 [ 0.44444444 0.21296296 0.08564815]
 [ 0.30555556 0.49074074 0.08796296]
 [ 0.25925926 0.44444444 0.08935185]
 [ 0.30555556 0.67592593 0.08935185]
 [ 0.53703704 0.35185185 0.09259259]
 [ 0.58333333 0.30555556 0.09398148]
 [ 0.39814815 0.35185185 0.09490741]
 [ 0.53703704 0.30555556 0.1087963 ]]
```

(Check out the sum and round functions!!):

```
Sum:
[259.0, 197.0, 74.61]
Round:
[[ 24.    5.    2.4]
 [ 13.    7.    2.9]
 [ 16.    7.    2.9]
 [ 19.    8.    3. ]
 [ 16.   10.    3.2]
 [  9.    9.    3.3]
 [ 11.   12.    3.4]
 [ 15.    9.    3.5]
 [ 14.   14.    3.5]
 [ 13.    9.    3.6]
 [  9.   10.    3.7]
 [  8.   10.    4. ]
 [ 12.    7.    4.2]
 [  9.   13.    4.3]
 [  8.   12.    4.3]
 [  9.   17.    4.3]
 [ 14.   10.    4.4]
 [ 15.    9.    4.4]
 [ 11.   10.    4.4]
 [ 14.    9.    4.8]]
tonys-new-mbp:Project2 Tony$
```

What I learned. I learned how to construct a Data class that can read and organize data from a file and to create an analysis class that smoothly performs data analysis functions on the data. Through this project, I became more comfortable with numpy! Also, I beginning to get my understanding of Python back. Finally, I am understanding how to visualize the work more clearly to better understand the goal with data analysis and visualization.

Who helped me. I received a lot of help from the *WONDERFUL* Bruce Maxwell and CP Majgaard. Also, I collaborated with Julia Saul, Steven Parrott, and Brendan Doyle.