test_2_notebook

July 14, 2021

1 Y-Data Test 2

1.1 D2: Implement

You are given a chess board with n rows and n columns. There are m chess rooks located on the board. Determine a number of cells that are free* and not under attack by any rook. (See illustration below – places threatened by the rook are marked with 'x') * free = not occupied by a rook

input:
N - size of the chess board
rooks - 2D array of integers, rooks[i][0] is a row and rooks[i][1] is a column

1. board - empty [1..N]x[1..N] array of integers, initialized with 0
2.
3. for id from 0 to length(rooks) - 1:
4. ```
   for row from 1 to N:
   ```
5. ```
       for column from 1 to N:
   ```
6. ```
           if row == rooks[id][0] and column == rooks[id][1]:
   ```
7. ```
               board[row][column] = 1 // the rook
   ```
8. ```
           if row == rooks[id][0] and column != rooks[id][1]:
   ```
9. ```
               board[row][column] = 2 // attacked cell
   ```
10. ```
          if row != rooks[id][0] and column == rooks[id][1]:
   ```
11. ```
              board[row][column] = 2 // attacked cell
   ```
12.
13.
14. unattacked_cells = N * N // number of unattacked cells
15. for row from 1 to N:

16. for column from 1 to N:
17. if board[row][column] == 2:
18. unattacked_cells -= 1
19.
20. return unattacked_cells

In this question you are required to write code in the window below or submit a source file implementing a solution to the problem presented in the previous question (D1). The first input line contains two integers n and m (1 n 1 0 0 , 1 m 1 0 0). Each of the m following lines contains two integers r i and c i (1 r i n , 1 c i n). Print one integer, the number of cells that are free* and not under attack of any rook.

[9]:
```
import numpy as np
import os
os.chdir(r'C:\Users\sfrie\Ydata\ydata_test_2\test2_attempt2')

input = open('d_input.txt').read().strip().split('\n')
input = [x.split(' ') for x in input]
rooks = [list(map(int, x)) for x in input]
print('Input: ', rooks)
n= rooks[0][0]
rooks.pop(0)
print('Cleaned input: ', rooks)
```

```
Input:  [[5, 4], [2, 2], [1, 1], [4, 2], [1, 3]]
Cleaned input:  [[2, 2], [1, 1], [4, 2], [1, 3]]
```

[15]:
```
N = n*n
board = np.reshape(list(np.zeros(N)), [5,5])

for id in range(0, len(rooks)):
    for row in range(1, n+1):
        for column in range(1, n+1): ##When just leaving it as range(1, n),␣
↪this gives a range of 1,2,3. However, we also need 4, in order to cover the␣
↪last rows and columns.
            if row == rooks[id][0] and column == rooks[id][1]:  ## Additonally,␣
↪each conditional here needs -1 offsets in the change, to convert from the␣
↪input (1, for example) to an index (0)
                board[row-1][column-1] = 1 ## the rook
            if row == rooks[id][0] and column != rooks[id][1]:
                board[row-1][column-1] = 2 ## attacked cell
            if row != rooks[id][0] and column == rooks[id][1]:
                board[row-1][column-1] = 2 ## attacked cell
    print('round', id, ': \n', board)
```

```
round 0 :
 [[0. 2. 0. 0. 0.]
```

```
 [2. 1. 2. 2. 2.]
 [0. 2. 0. 0. 0.]
 [0. 2. 0. 0. 0.]
 [0. 2. 0. 0. 0.]]
round 1 :
 [[1. 2. 2. 2. 2.]
 [2. 1. 2. 2. 2.]
 [2. 2. 0. 0. 0.]
 [2. 2. 0. 0. 0.]
 [2. 2. 0. 0. 0.]]
round 2 :
 [[1. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2.]
 [2. 2. 0. 0. 0.]
 [2. 1. 2. 2. 2.]
 [2. 2. 0. 0. 0.]]
round 3 :
 [[2. 2. 1. 2. 2.]
 [2. 2. 2. 2. 2.]
 [2. 2. 2. 0. 0.]
 [2. 1. 2. 2. 2.]
 [2. 2. 2. 0. 0.]]
```

[16]:
```
unattacked_cells = N ## number of unattacked cells
for row in range(0, n):
    for column in range(0, n):
        if board[row][column] == 2 or board[row][column] ==1 :
            unattacked_cells -= 1

print(unattacked_cells)
```

```
4
```

1.2 D3: Optimize

The pseudocode given in D1 has sub-optimal complexity. Optimize your solution for the previous problem (D2) if necessary, allowing it to run efficiently. Your solution will run on tests with larger constraints:

[17]:
```
input = open('d_input.txt').read().strip().split('\n') ##Important to add in .
 ↪strip(), in case of any leading or trailing space
input = [x.split(' ') for x in input]
print(input)
rooks = [list(map(int, x)) for x in input]
print(rooks)

n= rooks[0][0]
rooks.pop(0)
```

```
rows = set()
cols = set()
for x in rooks:
    cols.add(x[0])
    rows.add(x[1])
missing_rows = (n- len(rows))
missing_cols = (n- len(cols))
sum = missing_rows * missing_cols
print(sum)
```

```
[['5', '4'], ['2', '2'], ['1', '1'], ['4', '2'], ['1', '3']]
[[5, 4], [2, 2], [1, 1], [4, 2], [1, 3]]
4
```

1.3 E1: Shopping, Find Common

In this question (E1-E3) you are asked to assist a store in building a series of tools to understand its customers' behavior and improve future sales. You are provided with a file, purchase_data.csv, containing a dataset of past purchases by customers in the store's loyalty club. Each row in the dataset represents a single past purchase by a club member of a specific product. In this question you are asked to write a code implementing a function find_common, which receives as input the IDs of two products and returns the list of all members who bought both products.

Input format Two integers - product_1, product_2, corresponding to product IDs in the datafile (product_1 product_2).

Output format Print a sequence of integers corresponding to memberIDs of members who bought both products, each number in a single line. Return 0 if no members bought both products.

[19]:
```
import pandas as pd
df = pd.read_csv('purchase_data (1).csv')
df.head(5)
```

[19]:

	timestamp	memberID	productID	productScore
0	827500641	215	10	2
1	827500641	215	11	4
2	827500641	215	16	3
3	827500641	215	21	4
4	827500641	215	40	3

[21]:
```
file = open('e_input.txt').read().split(' ')
print(file)
prod1, prod2 = [int(x) for x in file]
```

```
['59', '72']
```

[35]:
```
# def find_common(prod1, prod2):
    # prod1_frame = df[df.productID == prod1]['memberID']
```

```
    # prod2_frame = df[df.productID == prod2]['memberID']
    # result = set(prod1_frame) & set(prod2_frame)
    # if len(result) == 0:
    #     return [0]
    # return result

#memberID is int64 type. We need to convert our str file into int.
def find_common(prod1, prod2):
    checked_customers = set()
    customers = []
    for id in df['memberID'].unique():
        if id not in customers:
            check_frame = df[df['memberID'] == id]

            if prod1 in check_frame['productID'].to_list() and prod2 in␣
 ↪check_frame['productID'].to_list():
                customers.append(id)
            checked_customers.add(id)
    if len(customers) == 0:
        return [0]
    else:
        return sorted(customers)

print(find_common(prod1, prod2))
# for x in find_common(prod1, prod2):
#     print(x)
```

```
[1, 36, 50, 73, 78, 80, 91, 96, 104, 119, 126, 131, 132, 156, 174, 196, 201,
205, 208, 247]
```

1.4 E2: Shopping, Similarity

In this question, you are asked to implement a measurement of similarity of purchasing history between two products. To do this, you must find the common customers who bought both products using the function you wrote in the previous question, and count how many times each of them bought each of the two products (count[memberID,productID]).

The formula for the measurement is as follows:

similarity (product_1, product_2):
  common = find_common(product_1, product_2)
  for member in common:
    sum_mult += (count[member,product_1] - productScore[product_1]) *
          (count[member,product_2] - productScore[product_2])
    sum_sq_1 += (count[member,product_1] - productScore[product_1])^2
    sum_sq_2 += (count[member,product_2] - productScore[product_2])^2
  similarity = sum_mult / (sqrt(sum_sq_1) * sqrt(sum_sq_2))

You are requested to write code implementing the measurement described above, so when given a

pair of products (product_1, product_2), it will return the score resulting from the formula.

Note: The data is the same as in question E1. In addition to columns used in E1 (memberID, productID), the formula makes use of productScore: a score given by the store to a specific product based on past sales performance. productScore is constant for all rows with the same productID.

Input format Two integers - product_1, product_2, corresponding to product IDs in the datafile (product_1 product_2).

Output format Print one float value, the score resulting from the running the comparison formula on the two products. Return NA if there are <2 common members or the similarity cannot be computed. Round your output to two decimal digits.

[30]:
```
import pandas as pd
import math

def similarity(prod1, prod2):
    common = find_common(prod1, prod2)
    if len(common) >=2:
        sum_mult =0
        sum_sq_1 = 0
        sum_sq_2 = 0
        for member in common:
            prod1_count = len(df[(df['memberID'] == member) & (df['productID']␣
↪== prod1)])
            prod2_count = len(df[(df['memberID'] == member) & (df['productID']␣
↪== prod2)])
            prod1_score = int(df[df['productID'] == prod1]['productScore'].
↪unique())
            prod2_score = int(df[df['productID']== prod2]['productScore'].
↪unique())

            sum_mult += (prod1_count - prod1_score ) * (prod2_count -␣
↪prod2_score)
            sum_sq_1 += (prod1_count - prod1_score)**2
            sum_sq_2 += (prod2_count - prod2_score)**2
        similarity = round( sum_mult / (math.sqrt(sum_sq_1) * math.
↪sqrt(sum_sq_2)), 2)
        if similarity != 0:
            return similarity
        else:
            return 'NA'
    else:
        return 'NA'
print(similarity(prod1, prod2))
#Return NA if there are <2 common members or the similarity cannot be computed.
#Round your output to two decimal digits
```

```
0.61
```

1.4.1 E3:

In this question, you are asked to use your work in the two previous questions to offer relevant products to a new member who has just made their first purchase at the store. Given a list of 3 product IDs and their counts representing the client's current history, you should find and return the ID of the product with the highest predicted purchase count. To do this, you should use the similarity function (from E2) to find the most similar products to the ones in the new client's purchase among all other products, predict how much of each product is the new customer likely to buy in the future using the formula below, and return the one among them with the highest predicted count.

predicted_count (new_member, candidate_product):
  for bought_product:
    sum_mult += similarity(candidate_product, bought_product) * count(new_member, bought_product)
    sum_similarities += abs(similarity(candidate_product, bought_product))
  predicted_count = (sum_mult / sum_similarities)

Calculate only for products whose similarity with target is not NA If predicted_count cannot be calculated, treat it as 0 If several products tie for highest predicted count, return the one with lowest ID.

Input format Three lines, each containing two integers separated with space - productID and count, corresponding to Product IDs in the datafile, representing products purchased by new member and the count for each product.

Output format Print one integer, corresponding to the product_ID of the product with the highest predicted count.

[39]:
```
import pandas as pd
import math
df = pd.read_csv('purchase_data (1).csv')

#In order to solve this question, we need to optimize our code by making our␣
 ↪dataframe into a less time-consuming dictionaries and lists.
csv = open("purchase_data (1).csv").read().strip().split('\n')
members = []
products = []
scores = []
productCount = {}
productScore = {}
for line in csv[1:]:
    line = line.split(',')
    memid = int(line[1])
    prodid = int(line[2])
    productScore[prodid] = int(line[3])
```

```
    count = productCount.get((memid, prodid), 0)
    productCount[(memid, prodid)] = count+1

    members.append(memid)
    products.append(prodid)
    scores.append(scores)

#Running this new 'Find_common' saves the program 60 seconds on a small input,␣
 ↪let alone the time it saves on the rest of the code.
def find_common(prod1, prod2):
    customers = []
    for id in set(members):
        if (id, prod1) in productCount.keys() and (id, prod2) in productCount.
 ↪keys():
            customers.append(id)
    if len(customers) == 0:
        return [0]
    else:
        return sorted(customers)
def similarity(prod1, prod2):
    common = find_common(prod1, prod2)
    if len(common) >=2:
        sum_mult =0
        sum_sq_1 = 0
        sum_sq_2 = 0
        for member in sorted(common):
            sum_mult += (productCount[(member, prod1)] - productScore[prod1]) *␣
 ↪(productCount[(member, prod2)] - productScore[prod2])
            sum_sq_1 += (productCount[(member, prod1)] - productScore[prod1])**2
            sum_sq_2 += (productCount[(member, prod2)] - productScore[prod2])**2
        similarity = sum_mult / (math.sqrt(sum_sq_1) * math.sqrt(sum_sq_2))
        if similarity != 0:
            return similarity
        else:
            return 'NA'
    else:
        return 'NA'


def input(new_member):
    bought_products = []
    for line in new_member:
        line = line.split(' ')
        bought_product, prod_count = [int(x) for x in line]
        bought_products.append(bought_product)
    return bought_products
```

```
def predict_count(bought_products):
    for candidate_product in products:
        sum_mult = 0
        sum_similarities = 0
        for line in new_member:
            line = line.split(' ')
            bought_product, prod_count = [int(x) for x in line]

            similar_result = similarity(candidate_product, bought_product)
            if similar_result == 'NA':
                break
            if bought_product != candidate_product:
                sum_mult += similar_result * prod_count
                sum_similarities += abs(similar_result)
        if sum_similarities == 0:
            predicted_count == 0
        else:
            predicted_count = (sum_mult / sum_similarities)
        if predicted_count >0 and candidate_product not in bought_products:
            all_counts[candidate_product] = predicted_count

    return max(all_counts, key = all_counts.get)


new_member = open('e_input.txt').read().strip().split('\n')
products = set(list(df['productID']))
all_counts = {}
print(predict_count(input(new_member)))
```

```
35
```

1.5 F1: Pre-Process

In this question you are provided with a dataset of school statistics and performance on standardised national tests. You are required to perform a series of manipulations and calculations on the dataset. Clean the dataset and correct errors in the following manner (in the listed order):

Remove entries (rows) with any of the following data entry errors:

Student Attendance Rate 0

Rigorous Instruction 0

Students Chronically Absent 1 0 0 %

Remove entries (rows) with N/A in “Student Achievement Rating” OR “Average Proficiency Score”

Convert “Student Achievement Rating” column from text into numeric values as follows:

Not Meeting Target = 0.0 , Approaching Target = 0.5 , Meeting Target = 0.8 , Exceeding Target = 1.0

Input format schools.csv (link to download it is located in 'Notes' section below) #### Output format Submit the resulting csv file: schools.csv. Do not reorder columns or rows in the file.

[52]:
```
import pandas as pd
import numpy as np

df = pd.read_csv('schools (1).csv')
df= df[df['Student Attendance Rate'].str.rstrip('%').astype('int')/100 > 0]
df = df[df['Rigorous Instruction'].str.rstrip('%').astype('int')/100 > 0]
df = df[df['Students Chronically Absent'].str.rstrip('%').astype('int')/100 < 1]

df = df.dropna(subset = ['Student Achievement Rating' , 'Average Proficiency␣
↪Score'])

def change(n):
    if n == 'Exceeding Target': return 1.0
    if n == 'Meeting Target': return 0.8
    if n == 'Approaching Target': return 0.5
    if n == 'Not Meeting Target': return 0.0
df['Student Achievement Rating'] = df['Student Achievement Rating'].
↪apply(change)

# df.loc[df['Student Achievement Rating'] == 'Exceeding Target', ['Student␣
↪Achievement Rating']] = 1
# df.loc[df['Student Achievement Rating'] == 'Meeting Target', ['Student␣
↪Achievement Rating']] = 0.8
# df.loc[df['Student Achievement Rating'] == 'Approaching Target', ['Student␣
↪Achievement Rating']] = 0.5
# df.loc[df['Student Achievement Rating'] == 'Not Meeting Target', ['Student␣
↪Achievement Rating']] = 0.0

result = df.to_csv('result.csv', index = False)
df.head(4)
```

[52]:

	School Index	District	Lowest Year	Highest Year	School Income Estimate	\
0	310001	5	6	8	NaN	
1	310002	2	PK	5	$26,114.78	
2	310003	2	PK	5	$103,399.19	
3	310004	6	PK	5	$34,684.21	

	Average Proficiency Score	Student Attendance Rate	\
0	65%	94%	
1	69%	98%	
2	77%	95%	
3	56%	92%	

	Students Chronically Absent	Rigorous Instruction	\

```
0                          19%                  97%
1                           6%                  93%
2                          13%                  97%
3                          33%                  91%

   Year 3 - All Students Tested  Year 4 - All Students Tested  \
0                             0                             0
1                           125                           106
2                           105                           142
3                           103                           100

   Year 5 - All Students Tested  Year 6 - All Students Tested  \
0                             0                           104
1                           104                             0
2                            61                             0
3                            81                             0

   Year 3 - Top Scoring  Year 4 - Top Scoring  Year 5 - Top Scoring  \
0                     0                     0                     0
1                     8                    13                    11
2                     9                    44                    18
3                     3                     7                     1

   Year 6 - Top Scoring  Student Achievement Rating
0                     8                        1.0
1                     0                        0.8
2                     0                        0.8
3                     0                        0.5
```

1.6 F2: Best Year

Write code solving the following problem: given a school index, find and return the grade year with the highest ratio of students who achieved the top score out of the total tested. Exclude grade years, for which performance data is not available (i.e. years where Year X All Students Tested = 0).

Return the numeric value corresponding to the grade Year. In case of a tie, return the lowest grade Year applicable. If no applicable year groups exists, return 0.

Input format One integer corresponding to School Index. #### Output format Print one integer corresponding to the grade year with maximum calculated ratio.

[53]:
```
ratio3 = df['Year 3 - All Students Tested']/ df['Year 3 - Top Scoring']
ratio4 = df['Year 4 - All Students Tested']/ df['Year 4 - Top Scoring']
ratio5 = df['Year 5 - All Students Tested']/ df['Year 5 - Top Scoring']
ratio6 = df['Year 6 - All Students Tested']/ df['Year 6 - Top Scoring']
```

```
df2 = pd.concat([ratio3, ratio4, ratio5, ratio6], axis =1).set_index(df['School␣
↪Index'])
df2.columns = ['3', '4', '5', '6']

n = int(open('f2_input.txt').read())
print(n)
highest = df2.loc[n].max()

for x in df2.loc[n]:
    if x == highest:
        print(df2.loc[n].idxmax(axis=1))
        break
else:
    print(0)
```

```
310014
5
```

1.7 F3: District Score

Write code which calculates performance by district based on the criteria provided below. Given a district code and Average Proficiency Score range (min and max), find all the schools corresponding to the criteria, and return the average Student Achievement Rating for the schools selected. For Average Proficiency Score you are given a minimum and a maximum value. Find all schools with min Average Proficiency Score (school) max. Return the average value for Student Achievement Rating.

Round your answer to two decimal digits

If no relevant schools are available, return 0.

Input format A list of 3 numbers separated by spaces: district, minimum threshold, maximum threshold. #### Output format Print out a single float value, rounded to two decimal digits corresponding to the average Student Achievement Rating among schools fulfilling the criteria described above.

[54]:
```
import pandas as pd
import numpy as np

df= pd.read_csv('result.csv')

df['Average Proficiency Score'] = df['Average Proficiency Score'].str.
↪replace('%', '').apply(pd.to_numeric)
df['District'].dtype
input = open('f3_input.txt').read().split()
input = list(map(int, input))

df = df[(df['District'] == input[0]) & (df['Average Proficiency Score'] >=␣
↪input[1]) & (df['Average Proficiency Score'] <= input[2])]
```

```
df.head()

print(round(df['Student Achievement Rating'].mean(), 2))

# def find_best(i):
#     bests = 0
#     result = 0
#     valid = []
#     six =  df.loc[i]['Year 6 - Top Scoring']/ df.loc[i]['Year 6 - All␣
 ↪Students Tested']
#     if df.loc[i]['Year 6 - All Students Tested'] >= 0:
#         if six >= bests:
#             bests = six
#             result = 6
#             valid.append(6)
#     five = df.loc[i]['Year 5 - Top Scoring']/  df.loc[i]['Year 5 - All␣
 ↪Students Tested']
#     if df.loc[i]['Year 5 - All Students Tested'] >= 0:
#         if five >= bests:
#             bests = five
#             result = 5
#             valid.append(5)
#     four =  df.loc[i]['Year 4 - Top Scoring']/ df.loc[i]['Year 4 - All␣
 ↪Students Tested']
#     if df.loc[i]['Year 4 - All Students Tested'] >= 0:
#         if four >= bests:
#             bests = four
#             result = 4
#             valid.append(4)
#     three = df.loc[i]['Year 3 - Top Scoring']/ df.loc[i, 'Year 3 - All␣
 ↪Students Tested']
#     if df.loc[i]['Year 3 - All Students Tested'] != 0:
#         if three >= bests:
#             bests = three
#             result = 3
#             valid.append(3)
#     if len(valid) == 0:
#         return 0
#     return result

# file = open('f2_input.txt').read().strip()
# i = int(file)

# df = df.set_index('School Index')
# print(find_best(i))
```

```
0.77
```