Sample Midterm B_YB

October 7, 2024

[44]:
```
## B - Q1. Election Simulator (7 points)
## Don't miss () in rng. / Don't forget how to use loc and iloc in df.
```

[8]:
```
import numpy as np
import pandas as pd
from numpy.random import default_rng
rng = default_rng()

df=pd.DataFrame([[10,2,8,1.5,15],[8, 1.
↪5,6,1,10],[6,1,8,2,13],[5,1,8,1,10]],index=['Region A','Region B','Region␣
↪C','Region D'],\
columns=['Dem-Mean','Dem-Std','Rep-Mean','Rep-Std','Delegates'])

T = 10

df
```

[8]:

	Dem-Mean	Dem-Std	Rep-Mean	Rep-Std	Delegates
Region A	10	2.0	8	1.5	15
Region B	8	1.5	6	1.0	10
Region C	6	1.0	8	2.0	13
Region D	5	1.0	8	1.0	10

[10]:
```
regions = df.index

dem_delegates = []
rep_delegates = []

for i in range(T):
    dem_num = 0
    rep_num = 0
    for region in regions:
        dem_voters = rng.normal(df.loc[region, 'Dem-Mean'], df.loc[region,␣
↪'Dem-Std'])
        print(f'Dem voters for {region}: {dem_voters}')
        rep_voters = rng.normal(df.loc[region, 'Rep-Mean'], df.loc[region,␣
↪'Rep-Std'])
        print(f'Rep voters for {region}: {rep_voters}')
```

```
        if dem_voters > rep_voters:
            dem_num += df.loc[region, 'Delegates']
            print(f'Dem num for {region}: {dem_num}')
        else:
            rep_num += df.loc[region, 'Delegates']
            print(f'rep num for {region}: {rep_num}')
    dem_delegates.append(dem_num)
    rep_delegates.append(rep_num)

print(dem_delegates)
print(rep_delegates)
```

```
Dem voters for Region A: 8.395776034495041
Rep voters for Region A: 8.048926962134312
Dem num for Region A: 15
Dem voters for Region B: 6.018232980528971
Rep voters for Region B: 5.984474751517843
Dem num for Region B: 25
Dem voters for Region C: 5.3109362273025456
Rep voters for Region C: 10.108569802201465
rep num for Region C: 13
Dem voters for Region D: 5.0717733810199155
Rep voters for Region D: 8.006386412372608
rep num for Region D: 23
Dem voters for Region A: 9.149012474296446
Rep voters for Region A: 8.315471524247075
Dem num for Region A: 15
Dem voters for Region B: 8.588428283869728
Rep voters for Region B: 6.633566087636663
Dem num for Region B: 25
Dem voters for Region C: 5.487878051005218
Rep voters for Region C: 10.244833898523593
rep num for Region C: 13
Dem voters for Region D: 7.106397935731505
Rep voters for Region D: 6.210711390983574
Dem num for Region D: 35
Dem voters for Region A: 9.843488474716406
Rep voters for Region A: 8.295670908212843
Dem num for Region A: 15
Dem voters for Region B: 6.510357911780737
Rep voters for Region B: 4.249628724963383
Dem num for Region B: 25
Dem voters for Region C: 6.2002210416844505
Rep voters for Region C: 8.679938235497042
rep num for Region C: 13
Dem voters for Region D: 4.3113132682527375
Rep voters for Region D: 9.407835245136921
rep num for Region D: 23
```

```
Dem voters for Region A: 12.31814202800986
Rep voters for Region A: 6.370139660547364
Dem num for Region A: 15
Dem voters for Region B: 7.901742637816867
Rep voters for Region B: 6.65939474034853
Dem num for Region B: 25
Dem voters for Region C: 4.230129648345399
Rep voters for Region C: 3.5649485528268263
Dem num for Region C: 38
Dem voters for Region D: 5.573080543901773
Rep voters for Region D: 8.432900675485858
rep num for Region D: 10
Dem voters for Region A: 5.55362077281747
Rep voters for Region A: 5.69630513847283
rep num for Region A: 15
Dem voters for Region B: 6.046967269659128
Rep voters for Region B: 5.971808482853115
Dem num for Region B: 10
Dem voters for Region C: 6.711928241394017
Rep voters for Region C: 9.373573998227291
rep num for Region C: 28
Dem voters for Region D: 2.917246494063431
Rep voters for Region D: 8.123618479578075
rep num for Region D: 38
Dem voters for Region A: 10.81313457479722
Rep voters for Region A: 7.252885325733575
Dem num for Region A: 15
Dem voters for Region B: 7.048201078891848
Rep voters for Region B: 4.402925859681455
Dem num for Region B: 25
Dem voters for Region C: 6.628035343913049
Rep voters for Region C: 10.188024813908495
rep num for Region C: 13
Dem voters for Region D: 3.8645699648708987
Rep voters for Region D: 6.1116039637989275
rep num for Region D: 23
Dem voters for Region A: 13.642455258958512
Rep voters for Region A: 10.282607386604319
Dem num for Region A: 15
Dem voters for Region B: 10.252802445448616
Rep voters for Region B: 8.577178590341795
Dem num for Region B: 25
Dem voters for Region C: 5.74887006365471
Rep voters for Region C: 8.257674577280593
rep num for Region C: 13
Dem voters for Region D: 5.590395755357491
Rep voters for Region D: 8.023576555601368
rep num for Region D: 23
```

```
Dem voters for Region A: 8.974483107843739
Rep voters for Region A: 4.959215532607321
Dem num for Region A: 15
Dem voters for Region B: 6.183552746810929
Rep voters for Region B: 5.845375828297727
Dem num for Region B: 25
Dem voters for Region C: 7.012704169259729
Rep voters for Region C: 9.158758670596107
rep num for Region C: 13
Dem voters for Region D: 6.112234551220102
Rep voters for Region D: 8.766927935855614
rep num for Region D: 23
Dem voters for Region A: 6.673387375998719
Rep voters for Region A: 9.193597497473089
rep num for Region A: 15
Dem voters for Region B: 8.49009044735523
Rep voters for Region B: 6.298254652505221
Dem num for Region B: 10
Dem voters for Region C: 5.733563384113166
Rep voters for Region C: 9.575337647952825
rep num for Region C: 28
Dem voters for Region D: 2.8018670093922236
Rep voters for Region D: 8.367601556763478
rep num for Region D: 38
Dem voters for Region A: 10.658486963775134
Rep voters for Region A: 7.554665221333066
Dem num for Region A: 15
Dem voters for Region B: 7.770791607072517
Rep voters for Region B: 4.406312655366403
Dem num for Region B: 25
Dem voters for Region C: 6.728602119083691
Rep voters for Region C: 7.563833872873282
rep num for Region C: 13
Dem voters for Region D: 5.000377287235964
Rep voters for Region D: 5.805207892175305
rep num for Region D: 23
[25, 35, 25, 38, 10, 25, 25, 25, 10, 25]
[23, 13, 23, 10, 38, 23, 23, 23, 38, 23]
```

[12]:
```python
def election(df, T):
    regions = df.index

    dem_delegates = []
    rep_delegates = []

    for i in range(T):
        dem_num = 0
```

```
        rep_num = 0
        for region in regions:
            dem_voters = rng.normal(df.loc[region, 'Dem-Mean'], df.loc[region,␣
 ↪'Dem-Std'])
            rep_voters = rng.normal(df.loc[region, 'Rep-Mean'], df.loc[region,␣
 ↪'Rep-Std'])
            if dem_voters > rep_voters:
                dem_num += df.loc[region, 'Delegates']
            else: # dem_voters <= rep_voters
                rep_num += df.loc[region, 'Delegates']
        dem_delegates.append(dem_num)
        rep_delegates.append(rep_num)
    return dem_delegates, rep_delegates
```

[14]:
```
# Sample run 1
dem,rep=election(df,10)
print('Democrat # of Delegates:',dem)
print('Republican # of Delegates:',rep)

# Output should be:
# Democrat   # of Delegates: [25, 38, 25, 15, 25, 25, 10, 15, 25, 25]
# Republican # of Delegates: [23, 10, 23, 33, 23, 23, 38, 33, 23, 23]
```

```
Democrat # of Delegates: [25, 10, 25, 38, 25, 25, 25, 25, 35, 15]
Republican # of Delegates: [23, 38, 23, 10, 23, 23, 23, 23, 13, 33]
```

[46]:
```
## B - Q2. Waiting Time Simulator (8 points)
```

[20]:
```
arrivalsList=[5,9,14,5,3,0,9,20,30,0,0]
n = 15
k = 2

queue = 0
total_queue = 0
occupancy = 0
admitted_list = []

print('Minute\tArrivals\tExit\tAdmitted\tOccupancy\tQueue\tTotal Queue')

for i in range(len(arrivalsList)):
    # Arrivals
    arrivals = arrivalsList[i]
    # Exit
    if i >= k:
        exit = admitted_list[i-k]
        occupancy -= exit
    else:
```

```
        exit = 0
    # Admitted & Queue
    if queue + arrivals <= (n - occupancy):
        admitted = queue + arrivals
        queue = 0
    else: # queue + arrivals > (n - occupancy):
        admitted = n - occupancy
        queue = (queue + arrivals) - admitted
    admitted_list.append(admitted)
    # Occupancy
    occupancy += admitted
    # Total Queue
    total_queue += queue
    # Check the logic
    
    print(f'{i}\t{arrivals}\t\t{exit}\t{admitted}\t\t{occupancy}\t\t{queue}\t{total_queue}')

# Return the average waiting time of customers, rounded to two decimal places.
print(round((total_queue / sum(arrivalsList)),2))
```

```
Minute  Arrivals        Exit    Admitted        Occupancy       Queue   Total
Queue
0       5               0       5               5               0       0
1       9               0       9               14              0       0
2       14              5       6               15              8       8
3       5               9       9               15              4       12
4       3               6       6               15              1       13
5       0               9       1               7               0       13
6       9               6       9               10              0       13
7       20              1       6               15              14      27
8       30              9       9               15              35      62
9       0               6       6               15              29      91
10      0               9       9               15              20      111
1.17
```

[22]:
```
def waiting_time(arrivalsList, n, k):
    queue = 0
    total_queue = 0
    occupancy = 0
    admitted_list = []

    for i in range(len(arrivalsList)):
        # Arrivals
        arrivals = arrivalsList[i]
        # Exit
        if i >= k:
            exit = admitted_list[i-k]
```

```
            occupancy -= exit
        else:
            exit = 0
        # Admitted & Queue
        if queue + arrivals <= (n - occupancy):
            admitted = queue + arrivals
            queue = 0
        else: # queue + arrivals > (n - occupancy):
            admitted = n - occupancy
            queue = (queue + arrivals) - admitted
        admitted_list.append(admitted)
        # Occupancy
        occupancy += admitted
        # Total Queue
        total_queue += queue
    # Return the average waiting time of customers, rounded to two decimal␣
↪places.
    return round((total_queue / sum(arrivalsList)),2)
```

[24]:
```
# Sample run 1
arrivalsList=[5,9,14,5,3,0,9,20,30,0,0]
waiting_time(arrivalsList,15,2)

# Output should be: 1.17
```

[24]: 1.17

[26]:
```
# Sample run 2
arrivalsList=[5,9,14,5,3,0,9,20,30,0,0]
print(f'Occupancy limit = 10\tWait time = {waiting_time(arrivalsList,10,3)} min.
↪')

# Output should be: Occupancy limit = 10    Wait time = 3.23 min.
```

```
Occupancy limit = 10    Wait time = 3.23 min.
```

[28]:
```
# Sample run 3
arrivalsList=[5,9,14,5,3,0,9,20,30,0,0]
print(f'Occupancy limit = 50\tWait time = {waiting_time(arrivalsList,50,3)} min.
↪')

# Output should be: Occupancy limit = 50    Wait time = 0.09 min.
```

```
Occupancy limit = 50    Wait time = 0.09 min.
```

[5]:
```
## B - Q3. Job Decision Simulator (9 points)
## Wrong > Need to work > Still don't get this.
```

This question asks you to write Python code to simulate the responses of a given student to a series

of job offers based on the timing of the offers and the student's personal preferences. Some job offers are acceptable to her and others are unacceptable, and among the acceptable offers, she may like some better than others. Every job offer has a deadline, and if she does not accept an offer by the deadline, then the offer expires. The student is very risk averse, meaning that she never lets an acceptable job offer expire without having another offer in hand that she likes at least as much. More precisely, assume that the student always responds to job offers based on the following rules: - She turns down unacceptable offers immediately. - She holds on to the first acceptable job offer that she receives. (Here, "holding on" to an offer does not necessarily mean that she will eventually accept the offer, but it means that she does not turn it down unless she gets another offer at least as good.) - At any time, she would hold on to at most one offer, which is her favorite acceptable offer received so far. If she receives a new acceptable offer while already holding on to an offer, she would compare the two and hold on to the offer she likes better, while turning down the other. If she likes the new offer equally as much as her favorite offer received so far, then she holds on to the one with the later deadline to respond, while turning down the one with the earlier deadline. If both of these favorite offers have the same deadline to respond, then she holds on to the one received earlier, while turning down the one received later. - For the favorite offer that she is holding on to, if by the end of the last day to respond to this offer, the offer is still her favorite (i.e. she has not turned it down for something else), then she will accept this offer, at which point she is required to turn down all future offers since her acceptance of an offer cannot be reneged.

Write a function called *job_decision* **with two input arguments:** - **offers**: a list in which each item corresponds to a job offer, and the order of the items corresponds to the order by which she receives the offers. Each job offer is represented by a list of three objects, *[offer_date,job_name,days_to_respond]*. The first object, offer_date, is an integer representing the day she receives the offer. The second object, job_name, is a string which serves as an unique identifier for the job. The third object, days_to_respond, is a positive integer indicating how many days she has to respond to the offer. For simplicity, assume that every offer is received on the morning of a day, and the deadline is always in the evening. example, if offer_date=5 and days_to_respond=3, then she receives the offer on the morning of Day 5, and she must respond to the offer by the evening of Day 8 (because 5+3=8). You may assume that the offers are sorted so that those with earlier offer dates appear earlier in the list. - **utility**: a dictionary in which the key is the unique identifier of a job (i.e. the same as job_name above), and the value is how much she likes the offer, with higher values corresponding to jobs she likes better. If two jobs have the same value, then she likes the two jobs equally. All of the entries in this dictionary are jobs she find acceptable, and if a job is not in the dictionary, then it means she does not find that job acceptable.

The function should return a string denoting the unique identifier (i.e. job_name) of the offer she accepts. If none of the offers are acceptable to her, then the function should return the empty string ''. See the sample inputs and outputs below for examples.

[25]:
```
# My code

def job_decision(offers, utility):
    best_offer_so_far = ''
    best_offer_value = -1
    best_offer_deadline = float('inf')
```

```
    for offer in offers:
        offer_date, job_name, days_to_respond = offer
        if offer_date > best_offer_deadline: # Check if we've passed the␣
↪deadline for the best offer so far
            return best_offer_so_far # if the offer date is later than the␣
↪deadline, accept the existing best offer

        if job_name not in utility:
            continue # Skip the unacceptable offers

        offer_value = utility[job_name]
        deadline = offer_date + days_to_respond

        # Update the best offer if:
        if best_offer_so_far == '': # 1. Add the current offer as the best␣
↪offer, if there is no acceptable offer yet
            best_offer_so_far = job_name
            best_offer_value = offer_value
            best_offer_deadline = deadline
        else: # 2. Compare the current offer with the existing best offer:
            # whether a current offer has a higher utility value or has equal␣
↪utility but an earlier deadline
            if (offer_value > best_offer_value or (offer_value ==␣
↪best_offer_value and deadline > best_offer_deadline)):
                best_offer_so_far = job_name
                best_offer_value = offer_value
                best_offer_deadline = deadline

    return best_offer_so_far
```

[31]:
```
# Solution

def job_decision(offers,utility):
    favorite_job=''
    for offer in offers:
        #print('New offer:',offer)
        offer_date,job_name,days_to_respond=offer
        deadline=offer_date+days_to_respond
        if job_name not in utility:
            #print('\tTurn down because not acceptable')
            continue
        if favorite_job=='':
            favorite_job=job_name
            key_deadline=deadline
            #print('\tReceived an acceptable offer')
        else:
            if offer_date>key_deadline:
```

```
                #print('\tOffer is too late. Already off the job market')
                break
            if utility[job_name]>utility[favorite_job] or␣
 ↪(utility[job_name]==utility[favorite_job] and deadline>key_deadline):
                favorite_job=job_name
                key_deadline=deadline
                #print('\tUpdated to favorite job')
    return favorite_job
```

[33]:
```
# Sample input 1
offers=[[5,'Intel',3],[8,'Amazon',7],[12,'Disney',3],[15,'Google',2],[15,'Facebook',2]]
utility={'Intel':5,'Amazon':8,'Google':8,'Facebook':8}
print(f'The student chose {job_decision(offers, utility)}')

# Output should be: The student chose Google.
```

```
The student chose Google
```

[35]:
```
# Sample input 2
# She first holds on to the Intel offer, but nothing as good appears by Day 8,␣
 ↪so she accepts Intel.
offers=[[5,'Intel',3],[9,'Amazon',7],[12,'Disney',3],[15,'Google',2],[15,'Facebook',2]]
utility={'Intel':5,'Amazon':9.5,'Google':10,'Facebook':10}
print(f'The student chose {job_decision(offers,utility)}.')

# Output should be: The student chose Intel.
```

```
The student chose Intel.
```

[37]:
```
# Sample input 3
# She first holds on to Amazon, then to Google, then to Facebook (each has␣
 ↪later deadline than the previous.)
offers=[[5,'Intel',3],[8,'Amazon',7],[12,'Disney',3],[15,'Google',1],[15,'Facebook',2]]
utility2={'Amazon':8,'Google':8,'Facebook':8}
print(f'The student chose {job_decision(offers,utility2)}.')

# Output should be: The student chose Facebook.
```

```
The student chose Facebook.
```

[39]:
```
# Sample input 4
# She first holds on to Amazon, and by Day 15, there's no offer that is as good␣
 ↪with later deadline.
offers=[[5,'Intel',3],[8,'Amazon',7],[12,'Disney',3],[13,'Google',2],[16,'Facebook',2]]
utility2={'Amazon':8,'Google':8,'Facebook':8}
print(f'The student chose {job_decision(offers,utility2)}.')

# Output should be: The student chose Amazon.
```

```
The student chose Amazon.
```

[41]:
```
# Sample input 5
# None of the offers are acceptable to her
offers3=[[8,'Amazon',7],[12,'Disney',3],[13,'Google',2],[17,'Facebook',2]]
utility3={'Apple':100, 'Intel': 80}
job_decision(offers3,utility3)
```

[41]:
```
''
```